

# Rockstar Developer

အနီမောင်

Fairway Web

10x Programmer ခေါ်ဆယ်ဆိုတော်သူတစ်ဦး

ဖြစ်လာအောင် အကူအညီပေးမယ့်စာအုပ်

[ [rsdbook.com](http://rsdbook.com) ]

© Copyright 2016, **Ei Maung**, Fairway Technology.



### License - CC-BY-NC-SA

This document, “Rockstar Developer by Ei Maung ” is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License.

This document is free to share, copy, distribute and transmit. And, also allow to remix or adapt to this document. But, you must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work). And, you may not use this work for commercial purposes. If you alter, transform, or build upon this work, you may distribute the resulting work only under the same or similar license to this one.

For any reuse or distribution, you must make clear to others the license terms of this work.

THE WORK IS PROTECTED BY COPYRIGHT AND/OR OTHER APPLICABLE LAW.  
ANY USE OF THE WORK OTHER THAN AS AUTHORIZED UNDER THIS LICENSE  
OR COPYRIGHT LAW IS PROHIBITED -

<http://creativecommons.org/licenses/by-nc-sa/3.0/legalcode>

## မာတိကာ

အမှာစာများ	၃
မိတ်ဆက်	၆
<b>အပိုင်း (၁) – Key Concepts</b>	
အခန်း (၁) – Development Processes Management	၁၄
အခန်း (၂) – Test Driven Development	၂၇
အခန်း (၃) – SOLID Design Principles	၄၆
အခန်း (၄) – Service Oriented Architecture	၅၄
<b>အပိုင်း (၂) – Project Management</b>	
အခန်း (၅) – Version Control System	၈၉
အခန်း (၆) – Issue Tracking System	၁၂၄
အခန်း (၇) – Team Collaboration	၁၄၀
အခန်း (၈) – Development Workflow Automation	၁၆၀
<b>အပိုင်း (၃) – Modern Development Stack</b>	
အခန်း (၉) – BackboneJS (Client-side MVC)	၁၉၃
အခန်း (၁၀) – NodeJS (Server-side JavaScript)	၂၂၄
အခန်း (၁၁) – MongoDB (NoSQL Database)	၂၅၈
အခန်း (၁၂) – ExpressJS (REST Framework)	၂၆၄
<b>အပိုင်း (၄) – Building an Issue Tracking System</b>	
အခန်း (၁၃) – Building Back-end Service	၃၁၉
အခန်း (၁၄) – Building Client (HTML Templates)	၃၅၀
အခန်း (၁၅) – Building Client (Web App)	၃၇၁
အခန်း (၁၆) – Building Client (Hybrid Mobile App)	၄၁၀
<b>အပိုင်း (၅) – Deployment and Scaling</b>	
အခန်း (၁၇) – Deployment with Docker	၄၁၈
အခန်း (၁၈) – Load Balancing with Nginx	၄၃၃
အခန်း (၁၉) – MongoDB Replication and Sharded Cluster	၄၄၈
အခန်း (၂၀) – Common Server Architectures	၄၆၅
<b>နိဂုံးချုပ်</b>	၄၇၉
<b>ကျေးဇူးတင်း</b>	၄၈၀

## ဆရာတိုးထွန်းခိုင်၏အမှာစာ

နည်းပညာပြောင်းလဲတိုးတက်မှုတွေက အရှိန်အဟုန် မြင့်လွန်းလှသည်။ ကျွန်ုတ်တို့ ကွန်ပျူတာနှင့် စတင်ထိတွေ့ရပြီး Programming ဆိုတာဘာလဲဆိုတာကို မြည်းစမ်းခဲ့ရတဲ့ (၁၉၈၉) လောက်ကနေ ယနေ့အချိန်အထိမှာ အများကြီး အများကြီးကို ပြောင်းလဲကုန်ပြီ။

ကိုယ်တွေခေတ်က အင်တာနှင်းဆိုတာလည်းမရှိသေး။ စာအုပ်စာတမ်းဆိုတာကလည်း အလွန်ကိုရှား။ Programming ကို စလေ့လာတော့မယ်ဆိုတော့ MCC မှာ သင်တန်းမတက်ခင်၊ ပြတိသူသံရုံးက စာကြည့်တိုက်ကိုသွားပြီး Programming နဲ့ ဆိုင်မယ့် စာအုပ်သွားရှာတော့ Low Level Programming ဆိုတဲ့စာအုပ်လေးတွေရေား။ အင်း၊ ငါလည်းအခုမှုလေ့လာမှာ ဆိုတော့ Low Level ပဲ စဖတ်ကြည့်တာပေါ့ဆိုပြီး ဤားလာခဲ့တယ်။ ညီးစိုင်းကနေ စဖတ်တာ၂ နာရီ ၃ နာရီလောက် ကြိုးစားဖတ်ပါသော်လည်း တစ်ရုံးတစ်ပါဒုမှ နားမလည်း။ စိတ်အားငယ်ရပြီ။ Low Level တောင် ဒီလောက်ခက်နေရင် High Level ဆိုဘယ်နှယ်လုပ်ကြမှတ်။ အခုပြန်ပြောပြတော့တာ ပြီးရရှုပ်ရပေမဲ့ အဲဒီတုန်းကတော့မပြီးနိုင်ခဲ့။ ကူညီ သင်ပြပေးမဲ့သူလည်းနည်း၊ စာအုပ်စာတမ်းကရှား၊ ကွန်ပျူတာစာအုပ်ဝယ်လိုဂုဏ်တုလိုလို မဂိုလမ်းနဲ့ ကုန်သည်လမ်းထောင့်က ရွှေဖြိုင်ဆိုတဲ့မိဇ္ဈားကူးဆိုင်လေးပဲရှိတယ်။

(၁၉၈၉) လောက်ကနေ အခုချိန်အထိကို Software Development နဲ့ ပတ်သက်လို့ Language မျိုးစုံ BASIC, Pascal , COBOL, C, C+, VB, Standalone, Client-Server, Three-Tier, Web စသည်ဖြင့် ပုံစံမျိုးစုံ ကြော်လည်ခဲ့ရပြီ။ Sun Microsystem က John Gage (၁၉၈၄) မှာ ပြောခဲ့တဲ့ "Network is the computer" ဆိုတဲ့ စကားလေးက အခုချိန် မှ တကယ်အကောင်အထည်ပေါ်လာခဲ့တယ်။ Cloud Computing လို့ နည်းပညာတွေ၊ Mobile နည်းပညာတွေ၊ Internet of Things တို့လို့ နည်းပညာတွေနဲ့ နေရာတကာ ကွန်ပျူတာဖြစ်ကုန်ကြပြီ။ ကွန်ပျူတာဆိုတာ ကျွန်ုတ်တို့ခေတ်က လို့ Monitor နဲ့ Keyboard, Mouse တပ်ထားထဲ CPU ပုံးပါတဲ့ အရာတစ်မျိုးတည်း မဟုတ်တော့။ လက်ကိုင်ဖုန်း၊ နာရီကအစ ကိုယ်မောင်းတဲ့ ကားထဲမှာလည်း ကွန်ပျူတာပါလာပြီ။ ဒီလိုအခြေနေမှာ Software Development ရဲ့ အခန်းကဏ္ဍကလည်း အရမ်းကို စိတ်ဝင်စားဖွယ်ဖြစ်လာပြီပေါ့။

ဒီလိုအချိန်မှာ ကိုအိမောင်ရဲ့ အခုစာအုပ်လေးက မြန်မာပြည်က Software Development ကို စိတ်ပါဝင်စားတဲ့ လူငယ်များအတွက် တစ်ဖက်တစ်လမ်းက အထောက်ကူဖြစ်မယ်ဆိုတာ အသေအချာပါပဲ။ အရင်ထွက်ထားတဲ့ စာအုပ်နှစ်အုပ်မှာကတည်းက စာရေးသူရဲ့စေတနာကို မြင်ရခံစားခဲ့ရတယ်။ အခုစာအုပ်လေးမှာလည်း Background Theory များနဲ့အတူ၊ ပြောင်းလဲလာတဲ့ယနေ့ခေတ်မှာ လက်တွေ့အသုံးဝင်မယ့် နည်းပညာများအကြောင်းကို သိင်းစိုင်းပြီး တင်ပြထားတာ စိတ်ဝင်စားစရာပါ။

ကိုအိမောင်အနေနဲ့လည်း လူငယ်တွေအတွက်လိုအပ်နေတဲ့ကွက်လပ်လေးတွေကို ဆက်ပြီးဖြည့်သွားနိုင်ပါစေလိုဆန္ဒပြရင်း ...

### ထွန်းခိုင်

Managing Director (Enterprise Solutions)

Vice President 2 (MCPA)

## ကိုသာထက်၏အမှာစာ

ကိုအိမောင်ကအမှာစာရေးပေးပါဟုစာမျက်မှုများပေးဖတ်သဖြင့် Rockstar Developer စာမျက်အသေအချာဖတ်ပြီး၊ ရှုထောင့် စုအောင် စဉ်းစားကြည့်မိသည်။

ဂိုဏ်လောကတွင် Rockstar ဟူသည် ထူးခြားထင်ရှား ကျော်ကြားကာ၊ လူအများ အားကျနှစ်ချိုက်သူကို ဆိုလိုသကဲ့သို့ Software Developer လောကတွင်လည်း ထိန်ည်းတူပင် ဖြစ်သည်။ ဤစာအပ်၏ရည်ရွယ်ရင်းမှာ စိတ်ပါဝင်စားသူတိုင်း Rockstar ဖြစ်နိုင်စေရန် တတ်သိသင့်သည်များကို ပြုစတင်ပြထားသည်ဟု ရှုမြင်မိသည်။

ဤစာအပ်တွင် သို့ပြု၏ (Theory) နှင့် လက်တွေ့ကိရိယာ (Tools) များကို မျှတွေ့စားသင့်ပြထားသည်။

အခြေခံ အတွေးပုံစံသို့ပြု၏ရှိများဖြစ်သည့် TDD, SOLID, Database Scaling နှင့် အသုံးများသည် Server Architecture များကို တင်ပြထားသလို ခေတ်သစ် Development အဆင့်များဖြစ်သည့် MEAN Stack နှင့် နောက်ဆုံးခေတ်မီ Docker ကဲ့သို့ Container အခြေပြု ဖြန့်ကျက်နည်းကိုလည်း ရှင်းပြထားသည်။ ထိုပြင် စနစ်ကျသော Developer များ၏ အလေ့အကျင့်ကောင်းများကိုလည်း Project Management ကဏ္ဍတွင် လက်တွေ့အသုံးချုနိုင်ရန် ရေးသားထားသည်။

အသစ်တက်သော လေ့လာသူအများအတွက် ဖတ်ရှုရန် အနည်းငယ်ခက်ခဲနိုင်သော်လည်း၊ အတွေ့အကြီး အတန် အသင့်ရှုနှင့်ပြီးသော Developer များအဖို့ သိကောင်းစရာများစွာ ပါဝင်လေသည်။

အပိုင်း (၅) ပိုင်းနှင့် တင်ပြထားသည်ကိုဖတ်ရသည်မှာ မဝမလင် ဖြစ်သည်။ ကဏ္ဍတစ်ခုချင်းစီတွင် အဆင့်မြင့် ခေါင်းစဉ်နှင့် ခက်ခဲသောအကြောင်းရာများကို နားလည်လွယ်ကူစေရန် ကျစ်ကျစ်လစ်လစ် ရေးသားထားရာ၊ ယခုထက် ပိုမိုကျယ်ပြန်သော စာအုပ်တစ်အုပ် ထပ်မံရေးသားပါမှ ပြည့်စုံခြင်းသို့ ရောက်ပေါ်သည်။

မည်သို့ပင်ဆိုစေ လေ့လာသူများအနေဖြင့် အခန်းတစ်ခုခြင်းစီမှပေးသော အခြေခံကောင်းဖြင့် မိမိသာသာ ကျယ် ကျယ် ပြန်ပြန်လေ့လာရန် လုံလောက်သောပညာကို ယခုစာအုပ်မှ ပေးစွမ်းနိုင်မည်မှာ အသေအချာပင်ဖြစ်လေသည်။

### သာထက်

CEO (Zwenexsys International Limited)

## စာရေးသူ၏အမှာ

မဂ်လာပါပင်များ။ Software Development နဲ့ စွမ်းပျိုးတာနည်းပညာကို လေလာနေသူများ အထောက်အကူဖြစ်စေဖို့ ဖို့တဲ့ ရည်ရွယ်ချက်နဲ့၊ ကျွန်ုတ်လေလာသိရှိထားတဲ့ နည်းပညာတွေကို ကဏ္ဍအလိုက်ရေးသားပြုစုလာခဲ့တာ အခုခို ရှင် တတိယမြောက်စာအပ်ရှိ ရောက်ရှိလာခဲ့ဖြေဖြစ်ပါတယ်။

ဒီစာအပ်မှုတော့ **ပြုပြင်ထိမ်းသိမ်းရလွယ်ကူတဲ့** Code တွေရေးသားရာမှာ အထောက်အကူဖြစ်စေနိုင်တဲ့ နည်းစနစ် တွေ၊ အဖွဲ့လိုက်ဆောင်ရွက်ရာမှာ အထောက်အကူဖြစ်စေနိုင်တဲ့ Project Management ပိုင်းဆိုင်ရာ သိမှတ်ဖွေယူရာ တွေ၊ NodeJS, MongoDB စတဲ့ Software Development နည်းပညာတွေနဲ့ Server Architecture ပိုင်းဆိုင်ရာ သိမှတ်ဖွေယူရာတွေကို စုစုည်းဖော်ပြပေးသွားမှာ ဖြစ်ပါတယ်။

ဒီစာအုပ်မှာဖော်ပြုမယ့် နည်းပညာများက Web Development ပိုင်းကို အသားပေးတဲ့ နည်းပညာများ ဖြစ်ပါတယ်။ Business Solution, Web App, Mobile App, Game စသာဖြင့် Software အမျိုးအစားအမျိုးမျိုးရှုပေမယ့် ကနောက်တွေမှာ ဘယ်လို Software အမျိုးအစားမဆို Web Technology များနဲ့ ကင်းကွာလို့ မရတော့ပါဘူး။ မည်သည့် Software မဆို Web Technology ကို Back-end အဖြစ်သော်လည်ကောင်း၊ ကြေားခံဆက်သွယ်ရေး နည်းပညာအနေနဲ့သော် လည်ကောင်း အသုံးပြုကြရပါတယ်။ ဒါကြောင့် ဒီစာအုပ်မှာ ဖော်ပြုမယ့် နည်းပညာများဟာ Web Developer များအတွက် သာမက၊ Mobile App Developer များ၊ Business Solution Developer များနဲ့ Software Engineer များ အားလုံး အတွက် အသုံးဝင်ပါလိမ့်မယ်။

ဒီစာအပ်ကိုဖတ်ရှုသူဟာ အခြေခံ Software Development နဲ့ Web Development ဆိုင်ရာနည်းပညာများကို သိရှိထားပြီးဖြစ်ဖို့လိုအပ်ပါတယ်။ အချိုအခြေခံတွေကို အသေးစိတ် ပြန်ရှင်းတော့မှာမဟုတ်ပါဘူး။ ဒါအပြင် မြန်မာနိုင်ငံ၏ ပို့ဆောင်ရေးနှင့် ပို့ဆောင်ရေး အကြောင်းအမြဲးကို ထည့်သွင်းဖော်ပြုမှုဖြစ်ပြီး၊ Command Line Tool အမြောက်အမြှားကို ထည့်သွင်းဖော်ပြုမှုဖြစ်ပြီး၊ Command လက်တွေ့နှမူနာတွေမှာ Ubuntu Linux ကို နဲ့မူနာထားပြီး ဖော်ပြသားမှပါ။ ဒါကြောင့် ကျွန်ုတ်တော်၏ ပထမစာအပ်နှင့်အပ်ဖြစ်တဲ့ **Professional Web Developer** နဲ့ **Ubuntu – သင့်အတွက် Linux** တို့ကို ကြိုတင်ဖတ်ရှုထားမယ်ဆိုရင် စာဖတ်သူအတွက် ပိုမိုတိရောက်အကျိုးရှိစေမှာဖြစ်ပါတယ်။ **Professional Web Developer** နဲ့ **Ubuntu – သင့်အတွက် Linux** တို့ကို အောက်ပါလိပ်စာများမှာ အခမဲ့ Download ရယူနိုင်ပါတယ်။

- Professional Web Developer – [pwdbook.com](http://pwdbook.com)
  - Ubuntu – ഉട്ടായത്ത് Linux – [u4ubook.com](http://u4ubook.com)

ချစ်နှိမ်းနှင်းဝေလွင်ရဲ့ပိုးမှုတွေနဲ့  
ချစ်သမီးလေးစုရတနာမောင်ပေးတဲ့အားအင်တွေကြောင့်  
ဒီစာအပ်ဖြစ်ပေါ်လာခဲ့တာပါ

မိတ်ဆက်

10x Programmer

Programmer ကောင်းတစ်ယောက်ဟာ

သာမာန် Programmer တစ်ယောက်ထက်

ဆယ်ဆာအလုပ်ပိုတွင်ပါတယ်။

## ମିତ୍ରଙ୍କ

“ Any fool can write code that a computer can understand. Good programmers write code that humans can understand. ” – *Martin Fowler*

ပရိဂ်ရမ်ဆိတ်တာကတော့ ဘယ် Programmer မဆိုရေးနိုင်ပါတယ်။ Programmer ကောင်း တစ်ယောက်မှသာ ရှင်းလင်း စနစ်ကျပြီး နားလည်လွယ်တဲ့ Code ကို ရေးနိုင်တာပါ။ နားလည်လွယ်တဲ့ Code ဖြစ်မှသာ အမှား နည်းမှာ ဖြစ်ပါတယ်။ ရေရှည်ပြုပြင်ထိမ်းသိမ်းရာမှာ အချိန်ကုန် သက်သာမှာဖြစ်ပါတယ်။ လိုအပ်တဲ့အဆင့်မြှင့် တင်မှုတွေကို အလွယ်တစ်ကူ ဆောင်ရွက်နိုင်စေမှာဖြစ်ပါတယ်။

အမှားတစ်စက်မှ မရှိတဲ့ Bug Free Program ဆိုတာ လက်တွေမှာ မရှိပါဘူး။ Bug ဆိုတာမျိုးက ရှတ်တရက် သတိမပြုမိပဲ အကြောင်းတိုက်ဆိုင်တော့မှ ထပြသုနာရှာတက်တဲ့ Error မျိုးပါ။ နားလည်လွယ်အောင် ရေးသား ထားတဲ့ Code မှသာ Bug တွေ ရှိလာတဲ့အခါ အလွယ်တစ်ကူ တွေ့နှုပ်ဖြင့်ဆင်နိုင်မှာဖြစ်ပါတယ်။ ပြီးတော့ ပရီ ဂရမ်တစ်ခုဆိုတာ တစ်ယောက်တည်းရေးလို့ မရပါဘူး။ ရတယ်ပဲထား၊ အဲဒီ Programmer မရှိတော့တဲ့အခါ (ဥပမာ - မနက်ဖြန် ကားတိုက်ခံရလိုသေသွားတဲ့အခါ) ပရီဂရမ်ကို လွှဲနှုပ်ရမယ်ဆိုရင်မဟုတ်သေးပါဘူး။ ပေါက်ကရပြောတာ မဟုတ်ပါဘူး၊ ဒီအခြေအနေအတွက် Bus Factor ဆိုတဲ့ အသုံးအနှံးတစ်ခု အမှန်တစ်ကယ် ရှိပါတယ်။ Software Project တစ်ခုရဲ့ အောင်မြင်မှုအလားအလာကို Bus Factor နဲ့လည်း တိုင်းတာတက်ကြပါ တယ်။ Software Project တစ်ခုအောင်မြင်ဖို့ အများနဲ့ ပူးပေါင်းဆောင်ရွက်ရတက် ပါတယ်။ မဟုတ်ရင်လည်း အခြား Programmer က အလွယ်တစ်ကူ လက်ဆင့်ကမ်း ဆောင်ရွက်နိုင်ဖို့ လိုပါတယ်။ နားလည်လွယ်အောင် ရေးထားတဲ့ Code မှသာ အများနဲ့ ပူးပေါင်း ဆောင်ရွက်နိုင်မှာဖြစ်သလို၊ လိုအပ်ရင်လည်း နောက် Programmer တစ်ယောက်က လက်ဆင့်ကမ်း ဆောင်ရွက်သွားနိုင်မှာဖြစ်ပါတယ်။

10x Programmer ဆိုတဲ့ အသုံးအနှစ်တစ်ခုလည်းရှိပါတယ်။ Programmer ကောင်းတစ်ယောက်ဟာ သာမဏ္ဍ Programmer ထက် ဆယ်ဆအလုပ်ပိုတွင်တယ်ဆိုတဲ့ အဓိပ္ပာယ်ဖြစ်ပါတယ်။ 10x Programmer တစ်ယောက်ရဲ့ အရည်အချင်းတွေ ထဲမှာ နည်းပညာကျမ်းကျင့်မှုအပြင် အဖွဲ့လိုက်ပူးပေါင်းဆောင်ရွက်နိုင်မှာ စူးစမ်းလိုစိတ်ပြင်းပြီး၊ အသေးစိတ်ကရ ပြုတက်မှု၊ သင်လွယ်တက်လွယ်မှု စသဖိုင့် အများကြီးရှိပါတယ်။ အဲဒါထဲကမှ စနစ်ကျပြီးနားလည်လွယ်တဲ့ Code တွေ ရေးသားနိုင်ခြင်းကလည်း အဓိကကျတဲ့ အရည်အချင်းတစ်ရပ်ပဲ ဖြစ်ပါတယ်။

၁၁။ မြုပ်ထွေးပြီးနားလည်ရခက်တဲ့ Code တွေဟာ အမှားများပါတယ်။ ပြင်ဆင်ရခက်ပါတယ်။ Maintain လုပ်ရခက်ပါတယ်။ အခြေခံမကောင်းခဲ့တဲ့အတွက်၊ နောက်ပိုင်းမှာ လုပ်ဆောင်ချက်သစ်တွေနဲ့ အဆင့်မြှုင့်ဖို့ လိုအပ်လာတဲ့ အခါ အဆင်မပြေတော့ပဲ ပရိုဂရမ်ကို အစအဆုံးပြန်ရေးရတော့မလို ဖြစ်လာတက်ပါတယ်။ ဒါကြောင့် Programmer ကောင်းတစ်ယောက်ဟာ သာမာန် Programmer ထက် ဆယ်ဆုံး အလုပ်တွင်တယ်ပေါ်တာပါ။

ရေးလိုက်တဲ့ Code ရဲ့ အရည်အသွေးဟာ ပရိုဂရမ်ရဲ့ အနာဂတ်ကိုပါ သက်ရောက်စေမှာ မို့ ဖြစ်ပါတယ်။

တစ်ချို့ Programmer တွေ ရှိပါတယ်။ Code တစ်ချို့ ရေးကြည့်လိုက်၊ စမ်းကြည့်လို အလုပ်လုပ်တယ်ဆိုရင် နောက် Code တစ်ချို့ ထပ်ရေးကြည့်လိုက်၊ လိုအပ်တဲ့ အခါ အင်တာနက်မှာရှာပြီး တွေ့တဲ့ Code တစ်ချို့ ထည့်သုံး လိုက်နဲ့ ပရိုဂရမ်ကတော့ အလုပ်လုပ်သွားပါတယ်။ အသေအချာ သုံးသပ်ပြင်ဆင်ထားတဲ့ အစီအစဉ်ဆိုတာ မရှိ ပါဘူး။ အနာဂတ် အလားအလာဆိုတာလည်း ထည့်စဉ်းစားမထားပါဘူး။ နောက်ဆုံးမှာ ပရိုဂရမ်ကတော့ အလုပ်လုပ်နေပါတယ်။ ဒါပေမယ့်၊ ရေးသားထားတဲ့ Code တွေရဲ့ အမိပိုပါယိုကို Programmer ကိုယ်တိုင် သေချာ နားလည်ဘူးဆိုတဲ့ အခြေအနေမျိုးတွေ ဖြစ်နေတက်ပါတယ်။ ဒါအခြေအနေကို Code by coincidence လိုခေါ်ပါတယ်။

တစ်ခါတစ်ရုံမှာ Programmer ကောင်း တွေတောင်မှ "နောက်ပြဿနာ နောက်မှုရှင်းမယ်" လိုတွေးပြီး ဒီလိုရေးကြတာတွေ ရှိပါတယ်။ Project Deadline သိပ်ကပ်နေလိုလည်း ဖြစ်နိုင်ပါတယ်။ တစ်ခြားအခြေအနေတစ်ခုခု ကြောင့်လည်း ဖြစ်နိုင်ပါတယ်။ ဒီလို "နောက်ပြဿနာ နောက်မှုရှင်းမယ်" ဆိုတဲ့ အပြုအမှုမျိုးကို Technical Debt လိုခေါ်ပါတယ်။ နည်းပညာကြွေးတွေပါ။ အကြွေးတွေမှား လာတဲ့ အခါ ဆပ်လို့မနိုင်ဖြစ်လာတက်ပါတယ်။ ရှေ့ကအကြွေးရှင်းရှင်းနဲ့ အကြွေးသစ်တွေ တင်လာတက်ပါတယ်။ ဒီလိုနဲ့ ရှုပ်ထွေးပြီးနားလည်ရခက်တဲ့ Code တွေနဲ့ ပြည့်နှက်နေတဲ့ ပရိုဂရမ်တွေ ဖြစ်ပေါ်လာကြပါတယ်။ Bug တစ်ခုရှုလုပ်လာတဲ့ အခါ ဘယ်နားမှာ ဘယ်လို သွားရှင်းရမလဲဆိုတဲ့ ကိစ္စမျိုးတွေအတွက် အချိန်တွေ ကုန်ရပါတယ်။ တန်ဖိုးလေးတစ်ခုလောက် ပြင် လိုက်ယုံနဲ့ ပြီးသွားရမယ့် ပြဿနာကို ဘယ်နားသွားရှာရမှန်း မသိတာနဲ့ နာရီပေါင်းများစွာ၊ ရက်ပေါင်းများစွာ အချိန်တွေ ကုန်သွားတာမျိုး ဖြစ်တက်ပါတယ်။

ဒီပရိုဂရမ်ကို လိုအပ်ချက်အရ အဆင့်မြှင့်ဖို့အတွက် ဆက်လက်ဆောင်ရွက်ရသွားမှာ လည်း အခက်တွေ့ရမှာဖြစ်ပါတယ်။ လက်ရှိရေးထားတဲ့ Code က ရှုပ်ထွေးလုန်းလို သူလည်းနားမလည်ပါဘူး။ ပိုကောင်း အောင် ပြင်လိုက်မှ လုံးဝအလုပ်မလုပ်တော့တာမျိုး ဖြစ်သွားတက်ပါတယ်။ ဘာကြောင့် ဒီလိုဖြစ်မှန်းလည်း မသိပါဘူး။ ဒါနဲ့ပဲ နောက်ထပ်ရှုပ်ထွေးတဲ့ Code တွေ ထပ်ထည့်ပြီး အလုပ်မလုပ်လုပ်အောင် ဇွတ်လုပ်ရပါတော့တယ်။ ဒီအခြေအနေ ကိုတော့ Accidents of implementation လိုခေါ်ပါတယ်။

ကျွန်ုတ်တို့တွေ ဝါသနာပါလွန်းလို လုပ်ခဲ့တဲ့ Programmer အလုပ်ဟာလည်း ဒီလိုအခြေအနေမှာ စိတ်ပျက်စရာတွေ ပြည့်နှက်နေတဲ့ အလုပ် တစ်ခုဖြစ်လာပါတယ်။ တစ်ချို့နှင့် ဒေါင်းမော့ရှင်ကော့ပြီး တက်ကြောင်းလန်းခဲ့တဲ့ Programmer တွေလည်း အလုပ်သွားရမှာ ကိုတောင် ကြောက်ရွှေ့လာပါတယ်။ Code ဖိုင်ကိုဖွံ့ဖြိုးကြည့်ရမှာကို စိုးရွှေ့လာပါတယ်။ တစ်ယောက် ယောက်က ပရိုဂရမ်မှာရှိနေတဲ့ Bug တစ်ခုအကြောင်း ပြောလာမှာကို စိုးထိတ်နေရပါတယ်။ ကိုယ့်ကိုယ့်ကို ယုံကြည့်မှုတွေ ပျောက်ဆုံးလာပါတယ်။ "မန်နေဂျာစိတ်ပြောင်းပြီး ဒီ Project ကို ရပ်လိုက်ပါစေ" လို့ အကြိမ်ကြိမ် ဆူတောင်းမိလာ ပါတယ်။ ပရိုဂရမ်ကဘယ်လို့မှ အလုပ်မဖြစ်လို့ ရပ်လိုက်ရတဲ့ အခါ လူမြှင်ကောင်းအောင် စိတ်မကောင်းချင်ဟန် ဆောင်ထားပေမယ့် တိတ်တိတ်လေး ဝစ်းသာနေ့မိတ်က်ပါတယ်။

စနစ်တစ်ကျ သပ်ရပ်အောင်ရေးထားတဲ့ Code တွေမှာတော့ ဒီ Code ကို ထိုင်ကြည့်နေရတာကိုက အနေပညာ မြောက် ပန်းချိုကားတစ်ချို့ကို တစ်စိမ့်စိမ့်ထိုင်ကြည့်နေရသလို အရသာရှိပါတယ်။ စိတ်ပါတ်တွေ တက်ကြပါ

တယ်။ အရာရာကလည်း ရိုးရှင်းလွယ်ကူနေတက်ပါတယ်။ Bug တွေရှိလာရင် စိတ်မည့်ပါဘူး၊ ဝမ်းတောင်သာ ပါသေးတယ်။ မိမိကိုယ်ကိုနဲ့ Project အပေါ် ယုံကြည်မှုအပြည့်အဝရှိတဲ့အတွက် Bug Fix လုပ်တယ်ဆိုတဲ့အလုပ် ကိုက စိတ်ဝင်စားစရာအလုပ်တစ်ခု ဖြစ်လာပါတယ်။ ဒါ Code ကို ဆက်လက်တာဝန်ယူရမယ့်သူရဲ့ အသိအမှတ် ပြုမှုကိုခံရပါတယ်။ အခြေခံကောင်းတဲ့ Code ကို ဆက်ခံရတယ်ဆိုပြီး အကြိမ်ကြိမ် ချီးမွမ်းမှုကို ခံရပါလိမ့်မယ်။

စနစ်ကျသပ်ရပ်တဲ့ Code တွေရေးနိုင်ဖို့ Language ကျမ်းကျင်မှုသက်သက်နဲ့ မလုံလောက်ပါဘူး။ Process Management, Source Code Management, Design Principles နဲ့ Design Architecture တို့လို Software Engineering ဆိုင်ရာ ဆက်စပ်နည်းပညာဗဟိုသုတေသန ရှိထားဖို့လိုအပ်ပါတယ်။ ဒီပုံမှန် ပြည့်စုံတဲ့ 10x Programmer တွေမှသာ စနစ်ကျသပ်ရပ်တဲ့ Code တွေရေးနိုင်မှာဖြစ်ပါတယ်။ အဖွဲ့လိုက် ယူးပေါင်းဆောင်ရွက် တက်မှာ ဖြစ်ပါတယ်။ အရည်အသွေးကောင်းပြီး Scalable ဖြစ်တဲ့ပရိုဂရမ်တွေရေးနိုင်မှာဖြစ်ပါတယ်။ 10x Programmer ကို Rockstar Developer လိုလည်းခေါ်ပါတယ်။ ဒီစာအုပ်က စာဖတ်သူကို 10x Programmer သို့မဟုတ် Rockstar Developer တစ်ယောက်ဖြစ်လာအောင် အထောက်အကြပ်ဖြစ်မယ်လို့ မျှော်လင့်ပါတယ်။

## အပိုင်း(၁)

Key Concepts

လက်တွေအသုံးပြုမယ့်သူက သူလိုချင်တာ ဒီလိုမဟုတ်သေးဘူး  
ဆိုတာကို အတော်လေးခရီးရောက်ပြီးမှ သိရှိသတိပြုမိလာခြင်းဟာ  
Software Project တွေ မအောင်မြင်ရခြင်းရဲ့ အဓိက  
အကြောင်းရင်းတစ်ခုဖြစ်ပါတယ်။

## အခန်း(၁) – Development Processes Management

"ဗျာ... နောက်ခံကို အပြာရောင်ပြောင်းချင်တယ်ဟုတ်လား။ အခုမှပြောရသလားဗျာ။ ကျွန်တော်က အနီရောင်နဲ့ လုပ်ထားတာ၊ ပြီးတောင်ပြီးတွေ့မယ်။"

"ဟုတ်ပါတယ်။ ကျွန်တော်တို့ကလည်း၊ အနီရောင်ဆိုရင် ကောင်းမယ်ထင်လို့ ရွှေးခဲ့တာပါ။ ဒါပေမယ့် ခင်ဗျားအခုပြုတဲ့ နောက်ဆုံးရလဒ်ကို ကြည့်လိုက်မှ၊ ကျွန်တော်တို့လို့ချင်တဲ့ ပုံစံ မဟုတ်မှန်းကိုသိရလိုပါ။"

"အခုက အားလုံးပြီးသလောက်ဖြစ်နေပြီ။ ပြောင်းဖို့ အဆင်မပြနိုင်တော့ဘူး။"

"စိတ်တွေ့မရှိပါနဲ့။ နားမလည်လို့ မေးပါရစေ။ နောက်ခံအရောင်လေးပဲ ပြောင်းချင်တာ၊ ဘာလို့ အဆင်မပြောတာလဲ။ ကျွန်တာတွေကို ပြင်ဖို့မလိုဘူးလေ။"

"ကျွန်တော်က စာလုံးအရောင်၊ အနေအထားနဲ့ နေရာအထားအသိ အားလုံးကို နောက်ခံအနီဆိုတဲ့ အချက်ပေါ်မှာအခြေခံပြီး လိုက်ဖက်အောင် အမျိုးမျိုးစမ်းသပ်၊ သေးချာစီစစ်ပြီးမှ ရွှေးခဲ့ရတာပါ။ အခု နောက်ခံအရောင်ပြောင်းချင်တယ်ဆိုလို့ အရောင်လေးတင် ကွက်ပြောင်းလို့မရပါဘူး။ စာလုံးအရောင်၊ အနေအထားနဲ့ နေရာအထားအသိ အားလုံးကို ပြန်လုပ်ရတော့မှာပါ။ ပြောင်းချင်တာက နောက်ခံဆိုပေမယ့် အစအဆုံးပြန်လုပ်ရတဲ့သဘောမြို့လိုပါ။"

"ခင်ဗျားလုပ်ငန်းအသေးစိတ်တွေ့ ကျွန်တော်တို့လည်း နားမလည်ပါဘူး။ ဒါပေမယ့် ကျွန်တော်တို့ အနေနဲ့ကတော့ နောက်ခံအပြာမှုမရရင် အဆင်မပြောပါဘူး။"

"အားနာရာနဲ့ပဲ ပြောပါရစေ။ ကျွန်တော်တို့ စသေဘာတူခဲ့တုန်းက အနီနဲ့သဘောတူခဲ့တာ။ ဒါကြောင့် အနီနဲ့ပဲ လုပ်ပေးနိုင်ပါမယ်။ အပြာနဲ့ထပ်လုပ်ပေးစေချင်ရင်တော့ ဝန်ဆောင်ခ ကျွေသင့်ငွေ ထပ်ပေးရပါ ယ်။"

"ဘာဗျား။ ကျွေသင့်ငွေထပ်ပေးရမယ် ဟုတ်လား။ ဒါတွေ့ မဟုတ်သေးဘူးထင်တယ်။ ခင်ဗျားကို ကျွန်တော်တို့က တန်ရာတန်ကြေးနဲ့သဘောတူထားပြီးသားပဲ။ အဆင်ပြအောင်လုပ်ပေးရမှာပေါ့။"

"ဖြစ်နိုင်တဲ့ကိုစွဲဆိုရင် လုပ်ပေးချင်ပါတယ်။ အခုဟာက အစအဆုံးနီးပါ ပြန်လုပ်ရမှာဆိုတော့ ဘယ်လိုမှ မဖြစ်နိုင်လိုပါ။ တန်ရာတန်ကြေးနဲ့ သဘောတူထားပေမယ့် နောက်ခံအနီနဲ့ လုပ်မယ်ဆိုတာကို ခင်ဗျား တို့ ဘက်ကလည်း သဘောတူခဲ့တဲ့ပဲမဟုတ်လား။"

"အစကသဘောတူခဲ့တာမှန်ပါတယ်။ ကျွန်တော်တိုကလည်းရလဒ်ကို မမှန်းဆတက်တော့ ကောင်းမယ် ထင်လို့ သဘောတူခဲ့တာပါ။ အခုလက်တွေမြင်ရတော့မှ အဆင်မပြောမှန်းကို သိရတာလေ့ဗျာ။"

ဒါဟာ Software Project အများစုံမှ ကြိတွေရတက်တဲ့ အခြေအနေ တစ်ရပ်ပဲဖြစ်ပါတယ်။ Software ကို အမှန်တစ်ကယ် အသုံးပြုကြမယ့် User သို့မဟုတ် Customer များဟာ နည်းပညာနားမလည်းကြပါဘူး။ သူတို့လုပ်ငန်းရဲ့ သဘောသဘာဝနဲ့ ရှိနေတဲ့ပြဿနာကိုနားလည်းကြပေမယ့်၊ အဲဒီပြဿနာကို နည်းပညာနဲ့ဖြေရှင်းရမယ့် Software Solution ဟာ ဘယ်လိုပုံစံဖြစ်သူ့ပြီး ဘယ်လိုအလုပ်လုပ်သင့်တယ် ဆိုတာကိုတော့ မမှန်းဆတက်ကြပါဘူး။ နောက်ဆုံး Software ကို မြင်ရတော့မှသာ ဒါဟုတ်တယ်၊ မဟုတ်ဘူး အမှန်တစ်ကယ်သိရှိကြရတာပါ။

Software Developer အနေနဲ့ ကလည်း မျက်မြင်တွေနေရတဲ့ ပြဿနာကိုသာသိပြီး၊ နောက်ကွယ်က လုပ်ငန်းသဘော သဘာဝ အသေးစိတ်နဲ့ အခြားဆက်စပ် အချက်အလက်တွေကို အတွင်းကျကျ မသိရှိတက်ကြပါဘူး။ ဘယ်လောက်ပဲ အချက်အလက်တွေ ရနိုင်သမျှရအောင်စုစည်းပြီး လေ့လာပေမယ့် မှန်းဆလိုမရတဲ့အခြေအနေ တွေက အမြှုနေတက်ပါတယ်။ ရထားတဲ့အချက် အလက်ကို အခြေခံပြီး Software ကို တည်ဆောက်ရတဲ့အခါ အတော်လေးခရီးရောက်တော့မှ ဒါမဟုတ်သေးဘူး လွှဲနေပြီလို့ ဆိုလာတဲ့အခါ ပြဿနာတက်တော့တာပါပဲ။



ပုံ(၁.၁) – Common Project Management Problem

Source – [projectcartoon.com](http://projectcartoon.com)

အတွေ့အကြွှု တွေများလာတာနဲ့အမျှ ဘယ်လိုလိုအပ်ချက်မျိုးဆိုရင် ဘယ်လို Solution မျိုးဖြစ်သင့်တယ်ဆိုတဲ့ Sense နဲ့ အတွေ့တွေ ဗဟိုသုတေသန ကြယ်ဝလာတဲ့အခါ၊ ပိုမိုမှန်းကန်တဲ့ ဆုံးဖြတ်ချက်တွေ ချိန်လာမှာဖြစ်တဲ့ အတွက် ဒီပြဿနာမျိုးတွေ လျော့ကျလာမှာ ဖြစ်ပါတယ်။ ဒါပေမယ့် လုံးဝမရှိတော့အောင် ရှင်းပြစ်ဖို့တော့ မလွယ်ပါဘူး။

လုပ်လိုရနိုင်တာကတော့၊ အမှန်တစ်ကယ်လိုအပ်ချက်နဲ့ လွှာနေတာရှိခဲ့ရင် စောစောသိရအောင် လုပ်ခြင်းပြုစိပါတယ်။ စောစောသိပြီး စောစောပြင်ခွင့်ရတဲ့အခါ အချိန်လွန်မှသိရလို ပြဿနာဖြစ်ရပါတယ်ဆိတာမျိုးကို ရောင်လွှာနိုင်မှာဖြစ်ပါတယ်။ ဒီလိုသိရဖို့အတွက် Software ကို အမှန်တစ်ကယ်အသုံးပြုမယ့် Customer သို့မဟုတ် User များထံကနေ Feedback တွေကို စောစောလက်ခံရယူခြင်းပဲ ဖြစ်ပါတယ်။

လုပ်ငန်းတွေအတွက် ရည်ရွယ်ဖန်တီးတဲ့ Business Solution Software တွေမှာတင် ဒီပြဿနာက ရှိတာမဟုတ်ပါဘူး။ Web App, Mobile App တို့လို အများသုံး၊ တစ်ကိုယ်ရည်သုံး Software တွေမှာလည်း ဒီပြဿနာရှိတာပါပဲ။ ကိုယ့်မှာ စိတ်ကူးအိုင်ဒီယာရှိပေါ်မယ့် အဲဒီစိတ်ကူးအိုင်ဒီယာဟာ အများရဲလိုအပ်ချက်နဲ့ အမှန်တစ်ကယ်ကိုက်မကိုက်၊ လက်တွေ့အလုပ် ဖြစ်မဖြစ်ဆိတဲ့အခြေအနေ Feedback ကို စောစောသိရလေ ကောင်းလေပဲဖြစ်ပါတယ်။

Rockstar Developer တစ်ဦးဟာ အခြေခံကောင်းပြီး ပြင်လွယ်ပြောင်းလွယ်တဲ့ Software တွေကို ဖန်တီးတက်ယုံမက User Feedback ကို စောစောရယူနိုင်တဲ့ Development Process တွေကိုလည်း သိရှိအသုံးပြုနိုင်ဖို့လိုအပ်ပါတယ်။

## 1.1 – Stages of Software Development

Software Product တစ်ခုဖြစ်မောက်လာဖို့ Code တွေချည်းထိုင်ရေးနေယံနဲ့ မရပါဘူး။ Solution Idea နဲ့ Design Concept တွေ စဉ်းစားဖန်တီးရပါတယ်။ လိုအပ်တဲ့အချက်အလက်တွေ စုစည်းရပါတယ်။ အသုံးပြုမယ့် နည်းပညာတွေ ရွေးချယ်ရပါတယ်။ Unit Testing, Integration Testing စသဖိုင်း အဆင့်ဆင့် စမ်းသပ်စိစစ်မှု တွေ ပြုလုပ်ရပါတယ်။ အသုံးပြုသူတွေ လိုအပ်တဲ့ Support ပေးဖို့ အစီအစဉ်တွေ သတ်မှတ်ရပါတယ်။

Software Product တစ်ခု တည်ဆောက်ရာမှာ ပါဝင်လေ့ရှိတဲ့အဆင့်တွေကို ဖော်ပြပေးလိုက်ပါတယ်။

1. Business Model Creation
2. Requirements Analysis
3. Data Collection
4. System Design
5. UI Prototype
6. Implementation / Coding
7. Testing
8. Continuous Integration Plan
9. Deployment / Integration
10. Technical Support

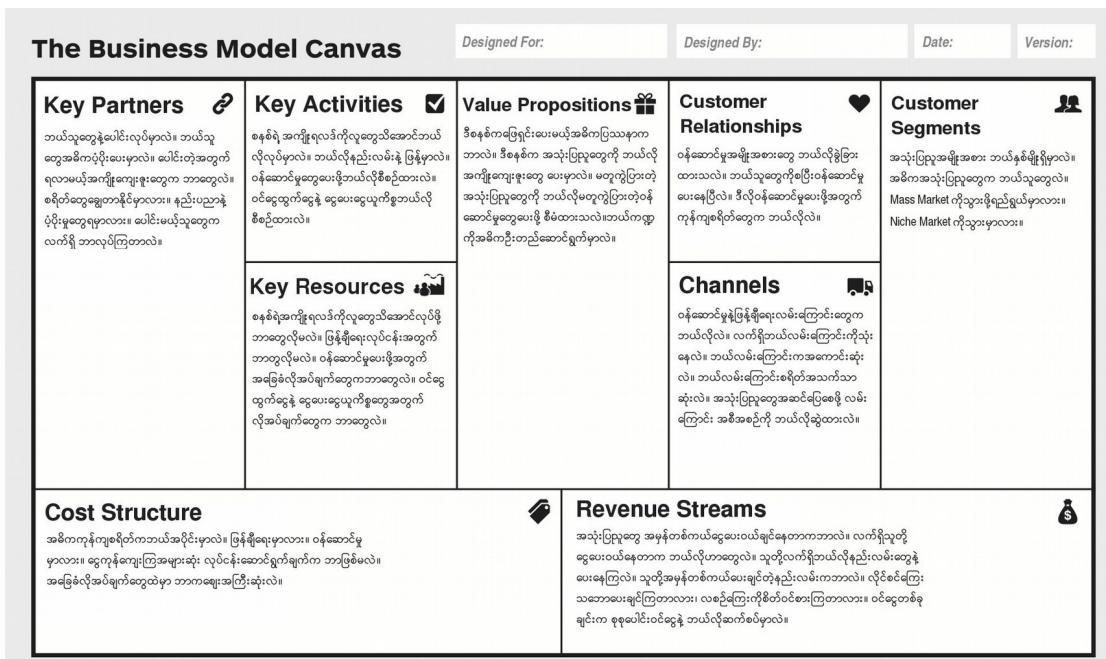
အရင်ကတော့ Software Development Process ကို Plan, Design, Code, Test ဆိုပြီး (င) ဆင့်လောက်ပဲ ခွဲကြလေ့ရှိပါတယ်။ အခုနောက်ပိုင်းမှာတော့ Development Practice တွေ အသစ်အသစ်တိုးတက်ပါပေါက်လာတာနဲ့အတူ ပါဝင်တဲ့အဆင့်တွေလည်း ဖော်ပြခဲ့သလို ပြောင်းလဲလာခဲ့ပြီပဲဖြစ်ပါတယ်။ ဥပမာ - Test လုပ်တယ်ဆိတဲ့

ကိစ္စကို အရင်က Software တည်ဆောက်ခြင်းလုပ်ငန်း ပြီးမှလုပ်ကလေ့ရှိပေမယ့် အခုနောက်ပိုင်းမှာတော့ Test Driven Development (TDD) လို နည်းစနစ်မျိုးတွေနဲ့ Coding လုပ်ငန်းဆောင်ရွက်နေစဉ်မှာပဲ တစ်ခါတည်း Unit Level, Function Level မှာ Test လုပ်နိုင်လာဖြီ ဖြစ်ပါတယ်။

Software Product တစ်ခုတည်ဆောက်ရာမှာ **Business Model** ကိုလည်း ထည့်သွင်းစဉ်းစားရလေ့ရှိပါတယ်။ Business Model ဆိုတဲ့အထဲမှာ Cost Structure နဲ့ Revenue Stream လို့ ငွေကြေးဆိုင်ရာကိစ္စတွေနဲ့ Distribution Channel တို့လို့ Software Development နဲ့ တိုက်ရှိက်မသက်ဆိုင်တဲ့အပိုင်းတွေ ပါဝင်ပါတယ်။ ဒါ ပေမယ့် ဒါ Software ကို ဘယ်နေရာမှာ အသုံးချမှာလဲ။ ဒါ Software ကပေးတဲ့အခိုက် Value ကဘာလဲ။ ဒါ Software က ဖြေရှင်းပေးမယ့် ပြဿနာကဘာလဲ စတဲ့အချက်အလက်တွေဟာ Business Model မှာ ပါဝင်ရလေ့ရှိပါတယ်။

ဗဟိုသတေသနနှင့် Business Model တစ်ခုရေးဆွဲရာမှာ အခြေခံအေးဖြင့် ပါဝင်ရမယ့်အချက်တွက် စုစုည်းဖော်ပြထားတဲ့ Business Model Canvas ကို ပုံ (၁.၂) မှာ ထည့်သွင်းဖော်ပြပေးထားပါတယ်။

Software Developer တစ်ယောက်အနေဖြင့် Business Model တစ်ခု လုံးကို ရေးဆွဲပေးဖို့ လိုချင်မှုလိုမှာပါ။ ဒါပေမယ့် တည်ဆောက်မယ့် Software ကိုတော့ Business Model နဲ့ ချိန်ညွှန် ထားဖို့လိုအပ်မှာဖြစ်ပါတယ်။ ဥပမာ - Business Model က လစဉ်ကြေး (Monthly Subscription Fee) ကို အခြေခံတဲ့ Payment Model ကို သုံးထားတယ်ဆိုရင် Software မှာလဲ အဲဒီကဏ္ဍအတွက် ထည့်သွင်းရေးသားပေးရတော့မှာ ဖြစ်ပါတယ်။ ဒါကြောင့် Business Model ဟာလည်း Software Development Process ရဲ့ အစိတ်အပိုင်းတစ်ရပ်ပဲ ဖြစ်ပါတယ်။ Business Model မှာ အပြောင်း အလဲရှိလာခဲ့ရင် Software ကိုလည်း သက်ရောက်မှာပဲဖြစ်ပါတယ်။



Business Model တစ်ခုရှိလာတဲ့အခါ ပိုပြီးတိကျတဲ့ **Requirements Specification** တစ်ခု ဆက်လက် တည်ဆောက်ဖို့ လိုအပ်ပါတယ်။ Software မှာ Feature အနေနဲ့ ပါဝင်ရမယ့် လုပ်ဆောင်ချက်တွေကာဘာတွေ လဲ။ Performance, Security စတဲ့ Non-Functional ပိုင်းအနေနဲ့ပါဝင်ရမယ့် လုပ်ဆောင်ချက်တွေကာဘာတွေ လည်း။ စသဖြင့် သတ်မှတ်ရပါတယ်။ Specification တစ်ခုမှာ ပါဝင်သင့်တဲ့အချက်တွေအကြောင်းကို အခန်း(၇) – **Team Collaboration** မှာ ဆက်လက်ဖော်ပြပါမယ်။

ဆက်လက်ပြီး လိုအပ်တဲ့အချက်အလက်တွေကို ကဏ္ဍအလိုက်ရယူစွာစည်းရတဲ့ **Data Collection** လုပ်ငန်းကို လည်း ဆောင်ရွက်ရပါတယ်။ ရရှိလာတဲ့ အချက်အလက်များကိုအခြေခံပြီး၊ Data တွေရဲ့ ဖွဲ့စည်းပုံ၊ အပြန်အလှန် ဆက်စပ်ပုံ တို့ကို ခွဲခြမ်းစိတ်ဖြာပြီး သင့်တော်တဲ့ Data Structure နဲ့ Model တွေကို တည်ဆောက်ရပါတယ်။

ရရှိထားတဲ့ Specification နဲ့ Data Model တို့ကို အခြေခံပြီး အသေးစိတ် **System Design** ကိုလည်း ရေးဆွဲထားဖို့ လိုပါတယ်။ ဘယ်လို Code Structure မျိုးကိုသုံးမှာလဲ MVC လား၊ Multi-Tire Design လား။ Architecture အနေနဲ့ရော့ Service-Oriented Architecture သုံးမှာလား၊ သုံးမယ်ဆိုရင် Communication Protocol အနေနဲ့ ဘာကိုသုံးမှာလဲ။ REST သုံးမှာလား၊ တစ်ခြား Protocol တစ်ခုကိုသုံးမှာလား၊ Authentication နဲ့ Authorization တို့အတွက် ဘယ်လို Access Control Strategy ကို သုံးမှာလဲ။ Performance အတွက် Memcached လုံနည်းပညာမျိုး၊ သုံးမှာလား၊ Scalability အတွက် Vertical Scaling ကိုရွေးမလား၊ Horizontal Scaling ကိုရွေးမလား၊ စသဖြင့် ဆုံးဖြတ်ပြင် ဆင်ထားရပါတယ်။

ပြီးတဲ့အခါ **User Interface Prototype** တွေလည်းတည်ဆောက်ရပါတယ်။ Prototype ကို Design Tool သက်သက် သုံးပြီးတည်ဆောက်တာမျိုးဖြစ်နိုင်သလို၊ Wire-framing Tool တွေကိုသုံးပြီး တည်ဆောက်တာ လည်း ဖြစ်နိုင်ပါတယ်။ ဒါမှမဟုတ် လက်တွေ့ Coding လုပ်ပြီး နမူနာရေးသားတည်ဆောက်လိုက် တာမျိုး လည်း ဖြစ်နိုင်ပါတယ်။ တစ်ခါတစ်ရုံမှာ Prototype ကို ကြည့်ပြီး၊ ပိုကောင်းတဲ့အိုင်ဒီယာရလာလို့ ဒါမှမဟုတ်၊ လက်တွေ့မဖြစ်နိုင်မှန်းသိရလို့ System Design ကို ပြောင်းလိုက်ရတာမျိုးလည်း ရှိတက်ပါတယ်။

အခြေခံလိုအပ်ချက်တွေ ပြည့်စုံလို့ လက်တွေ့ **Coding** အဆင့်ကိုသွားတဲ့အခါမှာတော့ Coding Standards, Coding Convention, Coding Design Principle, Best Practices စတဲ့ Code Quality နဲ့ပုံပုံပြုသက်တဲ့အပိုင်းတွေ ကို အလေးထားရပါမယ်။ အမိကကျတဲ့ Coding Structure Pattern တစ်ခုဖြစ်တဲ့ MVC အကြောင်းကို **Professional Web Developer** ရဲ့ အခန်း(၁၃) မှာ ဖော်ပြုခြားပြီးဖြစ်ပါတယ်။ နောက်ထပ်အရေးပါတဲ့ Practice တစ်ခုဖြစ်တဲ့ SOLID Principle အကြောင်းကိုတော့ အခန်း(၃) – **SOLID Design Principles** မှာ ဆက်လက် ဖော်ပြပါတယ်။

Software ဟာ မျှော်မှန်းထားတဲ့အနေအထားအတိုင်း မှန်မှန်ကန်ကန် အလုပ်လုပ်ကြောင်း သေချာစေဖို့ အဆင့် လိုက် **Test Method** အမျိုးမျိုးနဲ့ စမ်းသပ်စိစစ်ဆိုလိုပါတယ်။ Unit Testing လိုပေါ်တဲ့ Unit တစ်ခုချင်း၊ Function တစ်ခုချင်း စီကို စမ်းသပ်တဲ့နည်းစနစ်ရှိပါတယ်။ Integration Testing လိုပေါ်တဲ့ Unit တွေ ပေါင်းစပ်တဲ့ဖော်ထားတဲ့ ရလဒ်ကို စစ်ဆေးရတဲ့အဆင့်ရှိပါတယ်။ Specification နဲ့ ကိုက်ညီမှုရှိမရှိ စီစစ်ရတဲ့ System Testing နဲ့ Acceptance Testing လုပ်ငန်းတွေလည်းဟာ Software Development Process ရဲ့ အစိတ်အပိုင်းတစ်ရပ်ပဲဖြစ်

ပါတယ်။ Unit Testing အကြောင်းကို အခန်း (၂) – **Test Driven Development** မှာ ဆက်လက်ဖော်ပြပါမယ်။

Software Product တစ်ခုဆိုတာ ရေးပြီး ပြီးသွားပြီး ပြန်လှည့်ကြည့်စရာ မလိုတော့ဘူးဆိုတာမျိုး မရှိပါဘူး။ နည်းပညာ အဆင့်မြှင့်လာတာနဲ့အမျှ Software ကိုလည်း အဆင့်မြှင့်တင်မှုတွေ လိုက်လံဆောင်ရွက်ပေးဖို့ လိုအပ်တက်ပါတယ်။ Business Model နဲ့ Requirements ပြောင်းလဲသွားတဲ့အခါမှာလည်း လိုအပ်သလိုပြင်ဆင်မှုတွေကို ဆောင်ရွက်ပေးရမှာ ဖြစ်ပါတယ်။ ဒါကြောင့် စဉ်ဆက်မပြတ်အဆင့်မြှင့်တင်သွားနိုင်ဖို့အတွက် အစီအစဉ်တွေ ချမှတ်ထားရမှာဖြစ်ပါတယ်။ ဒီနေရာမှာ Source Code Management System တွေ၊ Build Automation နည်းစနစ်တွေနဲ့ Virtualization နည်းပညာတွေရဲ့ အကူအညီကိုယူရမှာဖြစ်ပါတယ်။ Source Code Management System အကြောင်းကို အခန်း (၅)၊ Build Automation အကြောင်းကို အခန်း (၇) နဲ့ Virtualization နည်းပညာတစ်ခုဖြစ်တဲ့ Docker အကြောင်းကို အခန်း (၁၇) တို့မှာ အသီးသီး ဆက်လက်ဖော်ပြပေးပါမယ်။

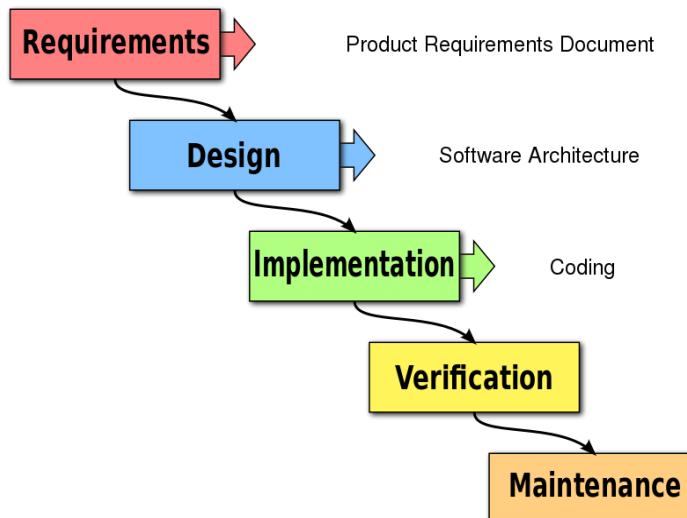
Software Solution ဟာ အသုံးပြုလို့ရတဲ့အဆင့်ကိုရောက်ပြီးဆိုရင်တော့ စတင်အသုံးပြုနိုင်အောင် တပ်ဆင်ခြင်း၊ ဖြန့်ဝေခြင်း လုပ်ငန်းတွေ ဆောင်ရွက်ရမှာပါ။ Server အတွက် လိုအပ်တဲ့ **Production Environment Setup** ပြုလုပ်ခြင်းလုပ်ငန်းတွေ ဆောင်ရွက်ရပါတယ်။ Load Balancer တွေ လိုအပ်ရင် Setup လုပ်ရပါမယ်။ Database Replication တွေ လိုအပ်ရင် စီစဉ်ရပါတယ်။ အသုံးများတဲ့ Server Architecture တွေအကြောင်းကို အခန်း (၂၀) – **Common Server Architectures** မှာ ဆက်လက်ဖော်ပြပါမယ်။

**Technical Support** ဟာလည်း အရေးပါတဲ့အဆင့်ဖြစ်ပါတယ်။ Solution ကို အသုံးပြုမယ့်သူတွေအတွက် လိုအပ်ရင် Training တွေစီမံပေးရပါမယ်။ သို့မဟုတ်ရင်လည်း Manual တွေ Documentation တွေ စီစဉ်ပေးထားရပါတယ်။ ပြဿနာ တစ်ခုတစ်ရာကြော်လာတဲ့အခါ အသုံးပြုသူက Report လုပ်လို့ ရနိုင်မယ့် နည်းစနစ်တွေ၊ လမ်းညွှန်တွေသတ်မှတ် ပေးရပါမယ်။ တစ်ချို့ မေးခွန်းတွေကို အရေးကြီးမှုအခြေအနေ အဆင့်အလိုက် ဖြေရှင်းပေးမယ့် Policy တွေ သတ်မှတ်ရပါမယ်။ ဒီနေရာမှာ တစ်ဘက်တစ်လမ်းက အထောက်အကူပေးနိုင်တဲ့ Issue Tracking System အကြောင်းကိုတော့ အခန်း (၆) မှာ ဆက်လက်ဖော်ပြပေးမှာ ဖြစ်ပါတယ်။

Software Team တွေမှာ System Analysis, Software Engineer, Programmer, Designer, Front-end Developer, Back-end Developer, System Administrator, Tester စသဖြင့် Role အမျိုးမျိုးနဲ့ တာဝန်ကိုယ်စီပေးထားကြသူတွေရှိနိုင်ပါတယ်။ ဖော်ပြခဲ့တဲ့ လုပ်ငန်းအဆင့်တွေထဲက ဘယ်အဆင့်ကို ဘယ်သူတာဝန်ယူမလဲဆိုတာကတော့ Team တစ်ခုနဲ့တစ်ခု လက်တွေမှာ တူမှာမဟုတ်ပါဘူး။ ဒီအဆင့်တွေကို ဘယ်လိုအစီအစဉ်၊ ဘယ်လို Process Management Model တွေနဲ့ ဆောင်ရွက်မလဲဆိုတဲ့ကိစ္စမှာတော့ အများစုံထားပြီးသုံးကြတဲ့ နည်းစနစ်တစ်ချို့ ရှိပါတယ်။ အဲဒီထဲက အမိကအကျဆုံးလိုဆိုရမယ့် နည်းစနစ်နှစ်ခုအကြောင်းကို ဖော်ပြပေးလိုက်ပါတယ်။

## 1.2 – Waterfall Model

Waterfall Model လိုခေါ်တဲ့ Development Process Management စနစ်တစ်မျိုး ရှိပါတယ်။ Software Development လုပ်ငန်းစဉ်မှာ ပါဝင်တဲ့ လုပ်ငန်းတွေကို အဆင့်လိုက် တစ်ခုပြီးတစ်ခု လုပ်သွားတဲ့ နည်းစနစ်ဖြစ်ပါတယ်။ ဥပမာ - ပထမဆုံးအနေနဲ့ လိုအပ်တဲ့ Requirements Specification တစ်ခုရရှိအောင်ကို အသေးစိတ် ရေးဆွဲရပါတယ်။ လိုချင်တဲ့ Requirements Specification အတိအကျရပြုဆိုတော့မှ System Design ကို စတင် ဆောင်ရွက်ရပါတယ်။ အသေးစိတ် Design ရပြုဆိုတော့မှ Coding လုပ်တဲ့ အဆင့်ကို သွားပါတယ်။ Coding လုပ်ပြီးသွားလို့ Product ထွက် လာတဲ့အချိန်မှ စမ်းသပ်တဲ့ Testing လုပ်ငန်းကို ဆက်လက်ဆောင်ရွက်သွားတဲ့ နည်းစနစ်ဖြစ်ပါတယ်။



ပုံ (၁.၃) - Waterfall Model

Source – Wikipedia

ဒီလိုအဆင့်လိုက် ဆောင်ရွက်သွားတဲ့ အတွက် ရလာတဲ့အားသာချက်ကတော့၊ အဆင့်တိုင်းမှာ ရှင်းလင်းတိကျေတဲ့ ဦးတည်ချက်ကို ရရှိမှုဖြစ်ပါတယ်။ Junior Developer တွေပါဝင်မှုမှားတဲ့ Team တွေနဲ့ ကိုက်ညီနိုင်ဖြစ်ပါတယ်။ ဘာကြောင့်လည်းဆိုတော့ ဦးဆောင်သူ Lead Developer နဲ့ Project Manager တွေက လိုအပ်တဲ့ Requirements Specification နဲ့ System Design ကြိုတင်ဆွဲထားပေးမှာမှာ Junior Developer တွေက သိပ်ပြီး ခေါင်းစားနေစရာမလိုပဲ ပေးထားတဲ့ Spec နဲ့ Design ကိုကြည့်ပြီး Coding လုပ်သွားယုံပဲမှု အခက်အခဲမရှိလှပဲ ဆောင်ရွက်နိုင်မှုဖြစ်ပါတယ်။

ဒါပေမယ့်လက်တွေမှာ တိကျေတဲ့ Specification ဆိုတာ ရရှိဖို့ခက်ပါတယ်။ ရှုပိုင်းမှာလည်း ပြောခဲ့ပြီးဖြစ်ပါတယ်။ လက်တွေအသုံးပြုမယ့် User က Specification ကိုကြည့်ပြီး သူလိုချင်တဲ့ ရလဒ်ဟုတ် ဆုံးဖြတ်နိုင်မှာ မဟုတ်ပါဘူး။ Software Product ကို မြင်ရတဲ့အချိန်ရောက်မှ တစ်ကယ်သူလိုချင်တာ မဟုတ်တဲ့အခါ အဆင့်လိုက် ဆောင်ရွက်ခြင်းဟာ ပြသနာအကြီးအကျယ်တက်ရတော့တာပါ။ Requirements Specification မှာက လည်းက လွှဲတဲ့အခါ System Design လည်းလိုက်ပြီး လွှဲတော့တာပါပဲ။ ဒါကိုကြည့်ပြီး Coding လုပ်လိုက်လို့

Product တွက်လာတော့မှ လွှဲနေပြီဆိုတော့ Specification နဲ့ System Design အတွက် ရင်းနှီးခဲ့ရတဲ့အချိန်တွေ အလကားဖြစ်ကုန်တော့မှာ ဖြစ်ပါတယ်။

ဒီလိုအားနည်းချက်ရှိနေပေမယ့်လည်း Waterfall Model ဟာ သူအနေရာနဲ့သူ အသုံးဝင်လဲဖြစ်ပါတယ်။ အထူးသဖြင့် Requirement Specification ကို အတိအကျသိနိုင်တဲ့အခြေအနေတွေမှာဆိုရင် အသုံးဝင်မှာဖြစ်ပါတယ်။ တိကျတဲ့ Specification ဆိုတာရဖို့ခေါက်တယ်လို့ ပြောခဲ့ပါတယ်။ ဒါက Product အသစ်တွေ၊ Innovative ဖြစ်တဲ့ ထူးခြားတဲ့ Product မှာပြောတာပါ။ ဥပမာ - Product တစ်ခုကို တစ်ကြိမ်ဆောင်ရွက်ခဲ့ဘူးတယ် ဆိုပါစို့။ လိုအပ်လို့ အဲဒီ Product မျိုး နောက်တစ်ကြိမ် ထပ်လုပ်ရတယ်ဆိုရင် တိကျတဲ့ Specification ရရှိဖို့ အလားအလာ အများကြီးကောင်းသွားပါတယ်။

အမှန်တစ်ကယ် အသုံးပြုမယ့် User က Product ကိုဖြင့်ရမှ သူလိုချင်တာ ဟုတ်မဟုတ် သိနိုင်တယ်လို့ ပြောခဲ့ပါတယ်။ ဒါလည်း ခြင်းချက်ရှုပါတယ်။ တစ်ခါတစ်ရုံ၊ တစ်ချိန်လုပ်ငန်းတွေမှာ လုပ်ငန်းရဲ့ Domain Knowledge အပြည့်အစုံရှုပုံမက နည်းပညာကိုလည်း ကျွမ်းကျင်နားလည်တဲ့သူတွေ ရှိနေတက်ပါတယ်။ သူတို့မှာ သူတို့ကိုယ်ပိုင် Product Vision တိတိကျကျရှုနေဖြီး ဖြစ်တက်ပါတယ်။ အဲဒါလိုအခြေအနေမျိုးအတွက် Software ရေးသားပေးရမှာဆိုရင်လည်း အတော်လေးတိကျတဲ့ Specification ကို အသင့်ရရှိမှာဖြစ်လို့ Waterfall Model က အသုံးဝင်လာမှာပဲ ဖြစ်ပါတယ်။

Waterfall Model ရဲ့ နောက်အားသာချက်တစ်ခုကို အထက်မှာလည်း ပြောခဲ့ပြီးဖြစ်ပါတယ်။ Software တစ်ခုကို အဖွဲ့ လိုက်ပူး ပေါင်းရေးသားကြတဲ့ အခါမှာ၊ အတွေ့အကြုံနည်းသေးတဲ့ Junior Developer တွေ ပါဝင်မှာများနေ တယ်ဆိုရင် Waterfall Model နဲ့ ပဲအလုပ်လုပ်ကြရမှာဖြစ်ပါတယ်။ Junior တွေမှာ ပြင်လွယ်ပြောင်းလွယ်ပြီး Flexible ဖြစ်တဲ့ Code တွေ ရေးသားနိုင်မှု အရည်အသွေး အားနည်းတက်ပါတယ်။ Waterfall Model နဲ့ မှာသာ ကြိုတင်ဖန်တီးပေးထားတဲ့ Specification နဲ့ Design ကို အားပြုပြီး Coding ပိုင်းကို Junior Developer တွေက Focus လုပ်နိုင်မှာပဲဖြစ်ပါတယ်။

## 1.3 – Iterative Development

Waterfall Model မှာရှိနေတဲ့ အားနည်းချက်တွေကို လေ့လာသုံးသပ်ပြီး၊ ထပ်မံတိထွင်လာကြတဲ့ Processes Management Method အမျိုးမျိုးရှိပါတယ်။ Iterative Development, Incremental Development, Extreme Programming, Rapid Application Development, Feature Driven Development စသဖြင့် အမျိုးမျိုးရှိပါတယ်။ ဒီနည်းစနစ် တွေကိုစုစုပေါင်းမြို့ အားလုံးတွေကို လိုလည်း ခေါ်ပါတယ်။ နည်းစနစ်တစ်ခုချင်းစီမှာ ပါဝင်တဲ့ အသေးစိတ်လုပ်ငန်းစဉ်တွေ ကဲ့ပြားသွားကြပေမယ့် ရည်ရွယ်ချက်ကတော့ တူကြပါတယ်။ Agile Development ရဲ့ ရည်ရွယ်ချက်တွေကတော့ -

1. **Adaptive Planning** – ပုံသေသတ်မှတ်ထားခြင်းမဟုတ်ပဲ လိုအပ်သလို ပြင်လွယ်ကြောင်းလွယ်တဲ့ Plan ပေါ်မှာ အခြေခံနိုင်ဖို့
  2. **Evolutionary Development** – Product တစ်ခုလုံးကို တစ်ခါတည်း တည်ဆောက်မယ့်အစား အဆင့် လိုက် တည်ဆောက်သွားနိုင်ဖို့

3. **Early Delivery** – အသုံးပြုသူက Product တစ်ခုလုံးပြီးစီးအချေသတ်အောင် စောင့်ဖို့မလိုပဲ စောနိုင်သမျှ အစောဆုံး Product ကို စတင်အသုံးပြနိုင်ဖို့
4. **Continuous Improvement** – Software ရဲ့ လုပ်ဆောင်ချက်နဲ့ အရည်အသွေး အဆင့်မြှင့်တင်မှုတွေ ကို အဆက်မပြတ် ဆောင်ရွက်သွားနိုင်ဖို့
5. **Rapid and Flexible Response to Change** – Requirement နဲ့ နည်းပညာ အပြောင်းအလဲရှိလာတိုင်း လိုအပ်သလို လက်ခံပြောင်းလဲနိုင်တဲ့ Product တစ်ခုဖြစ်အောင် စီမံရေးသားဖို့

– တိုပဲဖြစ်ပါတယ်။

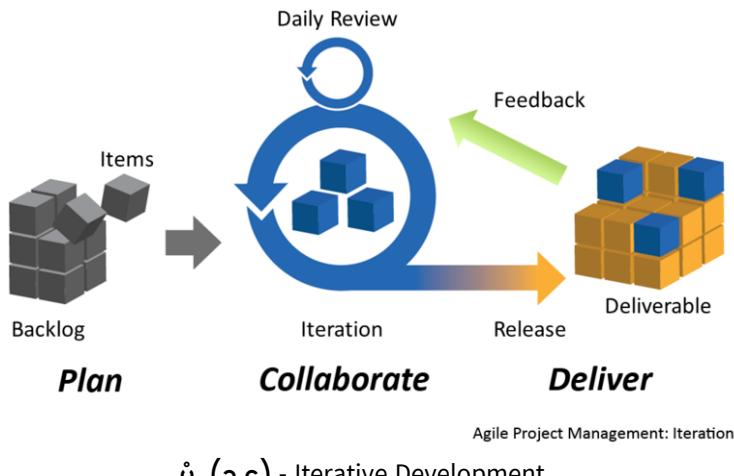
Agile Development ဆိုတာ နည်းစနစ်တစ်မျိုးကို ခေါ်တာမျိုးမဟုတ်ပါဘူး။ ရည်ရွယ်ချက် (Principle) ကို ခေါ်တာပါ။ အထက်မှာဖော်ပြခဲ့တဲ့ ရည်ရွယ်ချက်တွေနဲ့ ဖန်တီးထားတဲ့ မည်သည့် Processes Management နည်းစနစ်ကိုမဆို Agile Development (သို့မဟုတ်) Agile Method (သို့မဟုတ်) Agile Process Management လို ခေါ်ဆို နိုင်ခြင်းပဲ ဖြစ်ပါတယ်။

Iterative Development ဆိုတာ Agile နည်းစနစ်များစွာထဲမှာ အဓိကကျွတဲ့ နည်းစနစ်တစ်ခုပဲဖြစ်ပါတယ်။ ဒီ နည်းစနစ်မှာလည်း ပထမဦးဆုံးအနေနဲ့ Requirement Specification ကို စတင်တည်ဆောက်ရတာပါပဲ။ ဒါပေမယ့် အားလုံးကို အသေးစိတ်မသတ်မှတ်ပဲ၊ အခြေခံလိုအပ်ချက်တွေပါဝင်တဲ့ Project Scope ရရှိပြီး လိုအပ်နိုင်တဲ့အချိန်၊ လူအင်အားစတဲ့ Resource တွေကို Estimate လုပ်နိုင်စေဖို့ လိုအပ်သလောက်ကိုသာ စတင်ရေးဆွဲရပါတယ်။ ဒီအဆင့်ကို Iterative Development မှာ Inception လိုခေါ်ပါတယ်။

နောက်တစ်ဆင့်အနေနဲ့ Non-Functional Requirement ကို ဦးဆုံးဖြေရှင်းပါတယ်။ Software မှာ မျက်မြင်ပါဝင်မယ့် လုပ်ဆောင်ချက်တွေမဟုတ်ပဲ၊ နောက်ကွယ်ကအခြေခံတွေဖြစ်တဲ့ Maintainability, Security, Performance, Scalability စတဲ့လုပ်ငန်း တွေအတွက် အခြေခံ Architecture Foundation ကို တည်ဆောက်ခြင်းဖြစ်ပါတယ်။ Specification ဟာ အခြေအနေပေါ်မှတည်ပြီး ပြောင်းလဲနိုင်တယ်ဆုံးပေမယ့်၊ Function Requirement ပိုင်းမှာ အပြောင်းအလဲက ပိုများမှာ ဖြစ်ပါတယ်။ Architecture Requirement မှာ အပြောင်းအလဲ ရှိတယ်ဆုံးရင်တောင် Functional ပိုင်းအပြောင်းအလဲနဲ့ ယဉ်ရင် မပြောပလောက်ပါဘူး။ ဒီအဆင့်ကို Elaboration လိုခေါ်ပါတယ်။

လိုအပ်တဲ့အခြေခံ Architecture ရပြီဆိုတော့မှ Product မှာပါဝင်တဲ့ လုပ်ဆောင်ချက်များထဲက တစ်ခုကို ဦးဆုံးတည်ဆောက်ပါတယ်။ အဲဒီလုပ်ဆောင်ချက်ကို လက်တွေ့အသုံးပြုလိုရတဲ့အဆင့်ထိအပြီးဆောင်ရွက်ရပါတယ်။ Prototype တွေ လုပ်ရပါတယ်။ Coding တွေလုပ်ရပါတယ်။ ရရှိလာတဲ့ရလဒ်ကိုမှ လေ့လာသုံးသပ်မှုတွေ၊ User Feedback တွေရယူပြီး Specification ကိုရော၊ Code ကိုပါ လိုအပ်သလိုပြင်ဆင်ရပါတယ်။ Software Development Process မှာ ပါဝင်တဲ့ လုပ်ငန်းအားလုံးကို Software Product တစ်ခုလုံးအတွက် မဟုတ်ပဲ Feature တစ်ခုအတွက် အစအသုံးအကုန် ဆောင်ရွက်သွားတဲ့ သဘောဖြစ်ပါတယ်။ ဒီအဆင့်ကိုတော့ Construction လိုခေါ်ပါတယ်။ ဒီလုပ်ဆောင်ချက်တစ်ခုက လိုချင်တဲ့အနေအထားနဲ့ အမှန်တစ်ကယ်ကိုက်ညီပြီဆိုတော့မှ နောက်လုပ်ဆောင်ချက်တစ်ခုကို ဆက်လက်ဆောင်ရွက်ခြင်း ဖြစ်ပါတယ်။

ဒီနည်းနဲ့ Feature တစ်ခုကို လက်စသတ်နိုင်ပြီဆိုရင် အရင်ရှိထားပြီး Feature တွေနဲ့ ပေါင်းစပ်ရပါတယ်။ ပေါင်းစပ်ပြီး ရရှိလာတဲ့ ရလဒ်အတွက်လည်း User Feedback တွေ ထပ်မံရယူပါတယ်။ ဒီအဆင့်ကို **Transition** လို့ ခေါ်ပါတယ်။ ဒီနည်းအတိုင်း လုပ်ဆောင်ချက်တစ်ခုပြီးတစ်ခုကို အလုပ်ကျဆောင်ရွက်သွားတဲ့ နည်းစနစ်ဖြစ်လို့ Iterative Development လို့ ခေါ်ခြင်းဖြစ်ပါတယ်။ တစ်ဆင့်ပြီးတစ်ဆင့် Product မှာပါဝင်တည်း Feature တွေလည်းတိုးလာလို့ Incremental Development လိုလည်း ခေါ်ပါတယ်။



### ပုံ (၁.၄) - Iterative Development

Source – Wikipedia

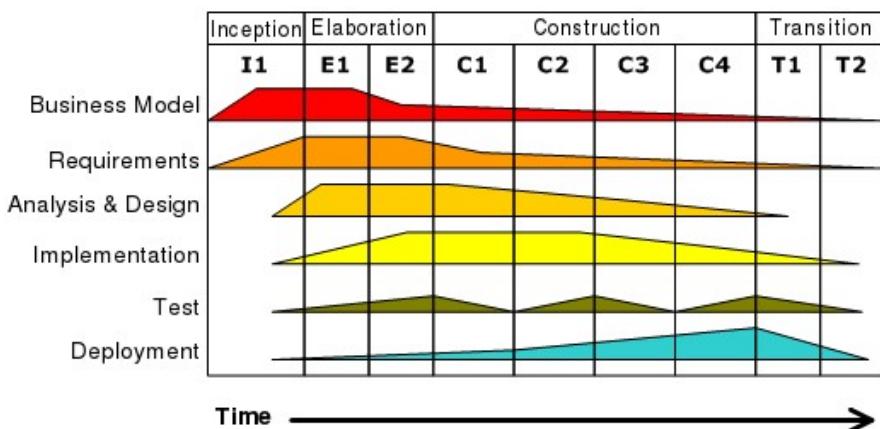
Iterative Development နည်းနဲ့ ဒီလိုတည်ဆောက်သွားတဲ့အတွက် လိုအပ်ချက်နဲ့ အမှန်တစ်ကယ်ကိုက်ညီခြင်း ရှိမရှိကို Product တစ်ခုလုံးပြီးအောင် စောင့်စရာမလိုပဲ သိရှိနိုင်လို့ Waterfall Model မှာရှိခဲ့တဲ့ Risk တွေလည်း နည်းသွားမှာ ဖြစ်ပါတယ်။ Iterative Development ကို လက်တွေ့အကောင်အထည်ဖော်ရာမှာ အသုံးပြုရတဲ့ SCRUM လိုခေါ်တဲ့ Project Management နည်းစနစ်တစ်ခုကို အခန်း (၇) – Team Collaboration မှာ ထည့်သွင်းဖော်ပြပေးပါ၏ီးမယ်။

Iterative Development မှာရှိနိုင်တဲ့ အမိကအားနည်းချက်ကတော့ Team မှာ ပါဝင်တဲ့ Developer တိုင်းဟာ အတွေ့အကြောင်းကျမ်းကျင် Developer တွေဖြစ်ဖို့ လိုအပ်နိုင်ခြင်းဖြစ်ပါတယ်။ စနစ်က Flexible ဖြစ်တဲ့ Plan, Requirement နဲ့ Design ကို အခြေခံထားတာမို့ Developer တွေကလည်း Flexible ဖြစ်ဖို့လိုပါတယ်။ အပိုင်းလိုက်ခွဲခြား တည်ဆောက်ထားပေမယ့် လိုအပ်သလို ပေါင်းစပ်နိုင်တဲ့ Modular စနစ်မျိုးကို ရေးသားနိုင်သူတွေဖြစ်ရပါတယ်။ လိုအပ်သလို ပြုပြင်ပြောင်းလဲရမှာမို့ Maintainable ဖြစ်တဲ့ Code တွေ၊ Extensible ဖြစ်တဲ့ Code တွေရေးနိုင်ဖို့ဟာ မဖြစ်မနေလိုအပ်ချက် ဖြစ်လာပါတယ်။ အဆင့်လိုက် ခွဲခြားစမ်းသပ်ရမှာဖြစ်လို့ Unit Testing နဲ့ Continues Integration နည်းစနစ်တွေကို နားလည်အသုံးချိန်ဖို့ လိုအပ်ပါတယ်။ Developer တွေမှာ ဒီလိုအရည်အရည်းတွေမရှိရင် Iterative Development နည်းစနစ်ကို အမှန်တစ်ကယ် လက်တွေ့ အသုံးချဖို့မှာ အခက်အခဲတွေနဲ့ ကြော်တွေ့ရမှာပဲဖြစ်ပါတယ်။

## Conclusion

Development Process Management တွေ အမျိုးမျိုးရှိကြပေမယ့်၊ လိုဂ်င်းကို ခြေကြည့်လိုက်ရင် အမိကက္ခာခြားချက် တစ်ခုပဲရှိတာကို တွေ့ရမှာဖြစ်ပါတယ်။ Waterfall Model လို နည်းစနစ်မျိုးမှာ Planning ပိုင်းဖြစ်တဲ့ Requirement Analysis နဲ့ System Design ရေးဆွဲခြင်းကဲသို့ လုပ်ငန်းများကို အချိန်ပေးပြီး ဦးစွာဆောင်ရွက်ရပါတယ်။ Iterative Development လို နည်းစနစ်မျိုးမှာတော့ Planning ပိုင်းကို အလွန်အမင်းအချိန်မပေးပဲ Development ပိုင်းကို အတက်နိုင်ဆုံး အရင်သွားပြီးမှ လိုအပ်သလို Plan ပိုင်းကို တွဲဖက်ဆောင်ရွက်သွားတဲ့ နည်းစနစ်ဖြစ်ပါတယ်။

Waterfall Model က Development Process မှာ ပါဝင်တဲ့ အဆင့်တွေကို အစီအစဉ်အတိုင်း တစ်ဆင့်ပြီးမှ တစ်ဆင့် ဆောင်ရွက်ရပေမယ့် Iterative Development ကတော့ ပါဝင်တဲ့ အဆင့်တွေကို အခုလိုပုံသဏ္ဌာန်မျိုး တဲ့ တွဲဖက်ဆောင် ရွက်သွားတဲ့ သဘော ဖြစ်ပါတယ်။



ပုံ (၁.၅) - Timeline of Iterative Development

Source: Wikipedia

ပထမဆုံး Inception အဆင့်မှာ Business Model နဲ့ Requirement Specification လိုက်စွဲမျိုးကို အမိက အချိန်ပေးပေမယ့် Coding နဲ့ Testing လုပ်ငန်းတွေလည်း လုပ်လို့ရသလောက် စလုပ်နေပါပြီ။ အလားတူပဲ Elaboration အဆင့်မှာ လည်း Architecture Design လုပ်ငန်းက အမိကဆိုပေမယ့် Coding နဲ့ Testing တွေကိုပါဆောင်ရွက်နေပါပြီ။ Construction အဆင့်မှာတော့ Coding က အမိကဆိုပေမယ့် Business Model, Requirement နဲ့ Design တိုကိုလည်း ရရှိလာတဲ့ Feedback များပေါ်အခြေခံပြီး လိုအပ်သလိုဆက်လက် ဆောင်ရွက်ရခြင်းဖြစ်ပါတယ်။ မသေခြားတဲ့ အတွက် Risk မြင့်တဲ့ Planning ကို အချိန်အရမ်းပေးမယ့်အစား၊ Coding ကို စောနိုင်သမျှစောအောင် Product ကို လက်တွေ့အသုံးပြု မယ့်သူလက်ထဲ စောနိုင်သမျှစောရောက်အောင် ဆောင်ရွက်ခြင်းဖြစ်ပါတယ်။ ဒါကို တစ်ချို့ကလည်း အမှတ်မှားပြီး Agile Method တွေမှာ Plan မလိုဘူးလို့ ဆိုတက်ကြပါတယ်။ အမှန်တော့ Plan မရှိတာမဟုတ်ပါဘူး။ Plan ကို Flexible ဖြစ်အောင် စီမံထားခြင်းသာဖြစ်ပါတယ်။

လိုရင်းကိုပြန်ကောက်ရရင်တော့၊ မည်သည့် Software Development Project မဆို Plan, Design, Implementation, Test စသဖြင့် အဆင့်ဆင့်ပါဝင်ကြတာချည်းပဲ ဖြစ်ပါတယ်။ အဲဒီအဆင့်တွေကို စီမံပုံစံမျဉ်း အမျိုးက ရွှေးထုတ်ဖော်ပြခဲ့တဲ့ Waterfall Model နဲ့ Iterative Development တို့မှာ အားသာချက်အားနည်းချက် ကိုယ်စီ ရှိကြပါတယ်။ Waterfall Model ဟာ သဘောသဘာဝအရာစနစ်ကျပေမယ့်၊ ပုံသေသတ်မှတ်ထားတဲ့ Specification နဲ့ Design ပေါ်မှာ အခြေခံရတဲ့အတွက် အားနည်းချက်ရှိနေပါတယ်။ ဒီအားနည်းချက်ကို ဖယ်ထုတ်နိုင်ရင်တော့ Waterfall Model ကို အသုံးပြုသင့်ပါတယ်။ တစ်နည်းအားဖြင့် တိကျသေချာတယ်လို့ အာမခံနိုင်လောက်တဲ့ Requirement ရှိထားရင် Waterfall Model ကို အသုံးပြုသင့်ပါတယ်။

ကြိုတင်မှန်းဆရာတ်ပြီး အပြောင်းအလဲများနှင့်တဲ့ အခြေအနေတွေမှာ Iterative Development ကို အသုံးပြုသင့်ပါတယ်။ လက်တွေမှာ Iterative Development နဲ့ အခြား Agile Method တွေဟာ Waterfall ရဲ့ အားနည်းချက်တွေကို လေ့လာဖန်တီးထားကြတာဖို့ ပိုကောင်းပါတယ်။ အပြောင်းအလဲဒါက်ကို ပိုခံနိုင်ပါတယ်။

နောက်အခန်းတွေမှာ Software Project နဲ့ Development Process များကို စီမံရာမှာ အသုံးဝင်မယ့် နည်းစနစ် တွေနဲ့ Tool တွေကို ဆက်လက်ဖော်ပြပေးသွားမှာဖြစ်ပါတယ်။

ပြင်တာကတစ်နေရာ၊ မဆီမဆိုင်လို့ထင်ရတဲ့ တစ်ခြားနေရာမှာ  
သွားမှားနေတယ်ဆိုတာ ဖြစ်တက်တဲ့ပြဿနာပါ။ ဒါကြောင့်  
ပြင်ဆင်မှုတစ်ခုပြုလုပ်လိုက်တိုင်း အခြားလုပ်ဆောင်ချက်တွေကိုပါ  
သေချာအောင် ပြန်စမ်းဖို့လိုနိုင်ပါတယ်။ ဒီလုပ်ငန်းကိုအမြဲ  
Manual လုပ်နေရမယ်ဆိုရင် အဆင်ပြုမှာမဟုတ်ပါဘူး။

### **Professional Web Developer Course**

HTML5, PHP/MySQL, jQuery/Ajax, Mobile Web စသည်

Professional Web Developer တစ်ဦး သိရှိထားသင့်သည်

နည်းပညာများကို စုစုပေါင်းသင်ကြားခြင်းဖြစ်သည်။

ဆက်သွယ်ရန် - (၀၉) ၇၃၁ ၆၅၄ ၆၂

## အခန်း(၂) - Test Driven Development

မနက်ဖြန် Project Presentation လုပ်ရတော့မယ်။ (၆) လလောက် အချိန်ပေးပြီး ကြိုးစားထား သမျှအားလုံးဟာ ဒီနွှာအတွက်ပဲ။ အောင်မြင်တဲ့လုပ်ငန်းရှင် Customer တွေကို ဆွဲဆောင်ရတာ လွယ်တဲ့အလုပ်မဟုတ်ဘူး။ အတော်လက်ပေါက်ကပ်တဲ့သူတွေ။ ပေါက်ပေါက် ရှာရှာလည်းမေးတက်တယ်။ ဒါကြောင့်လည်း အောင်မြင်ကြတာဖြစ်မှာပေါ့။ ပြဿနာတော့မရှိပါဘူး။ ကိုယ့်ဘက်ကလည်း အားလုံးထောင့်စီအောင် လုပ်ထားပြီးသား။

အခြေခံလုပ်ဆောင်ချက်တွေကို အရင်ရှင်းပြရမယ်။ ပြီးမှ တစ်ကယ့် Key Point ကို ထုတ်ပြရမယ်။ ခလုပ်တစ်ချက်နှင့်ယုံနှင့် အထိရောက်ဆုံး Delivery Route ကို အလိုအလျောက် ဆွဲပေးနိုင်မှာလေ။ ဒီလောက်ကောင်းတဲ့စိတ်ကူး၊ မကြိုက်စရာ ဘာအကြောင်းမှမရှိဘူး။ ဒါတင်ဘယ်ကမလဲ။ Mobile App ကနေလည်း Delivery Route ကို Maps နဲ့ ပြပေးနိုင်သေးတယ်။

အင်း... အခုမှုကြည့်စီတယ်။ Map ပေါ်မှာပြတဲ့လိပ်စာမှာ စာလုံးနည်းနည်းသေးနေတယ်။ သေချာ ကြည့်ရင်တော့ မြင်ရပါတယ်။ သိပ်ပြဿနာမရှိလှပပါဘူး။ ဒါပေမယ့် ဒီအပိုင်းက အမိန့်တွေ့ ဖြစ်နိုင်ရင် Perfect ဖြစ်ချင်တယ်။ စာလုံး Size ပြင်တာ ဘာမှခက်တဲ့ကိစ္စလည်းမဟုတ်ဘူး။ Font-Size: 9pt ဆိုတဲ့ အပိုင်းကို Font-Size: 11pt လို့ ပြင်လိုက်ယုံပဲ။ ဒါလေးတစ်ချက်တော့ ပြင်လိုက်းမယ်။ ဘာမဟုတ်တာလေးနဲ့ Impression Out နေပါ့။

အိုကေ... ပြင်ပြီးပြီး စမ်းကြည့်တာလည်း အဆင်ပြေတယ်။

- နောက်တစ်နေ့။ အခြေခံလုပ်ဆောင်ချက်များ ရှင်းလင်းတင်ပြပြီးနောက် -

"အခုတွေ့ရမှာကတော့ ဒီစနစ်ရဲ့ အမိကအကျဆုံးလုပ်ဆောင်ချက်ဖြစ်ပါတယ်။ ခလုပ်တစ်ချက်နှင့်ယုံနှင့် လက်ခံရရှိထားတဲ့ Order တွေရဲ့ လိပ်စာကိုအခြေခံပြီး အထိရောက်ဆုံး Delivery Rout ကို အလိုအလျောက် ဆွဲပေးမှာဖြစ်ပါတယ်"

"တယ်ဟုတ်ပါလား။ အဲဒီ Route ကို ပစ္စည်းပို့သမားကို ဘယ်လိုပေးမလဲ"

"အဲဒီအတွက် Mobile App တစ်ခုစီစဉ်ထားပါတယ်။ App ကိုဖွင့်ပြီး Sync လို့ ပြောလိုက်တာနဲ့ Delivery Route ကို Mobile Device ပေါ်မှာ Map နဲ့ ပြပေးမှာပါ"

"ဟုတ်လား။ စိတ်ကူးကောင်းသားပဲ။ ပြကြည့်ပါ့။"

"ဟုတ်ကဲ... အခုလို Sync Button ကို နှိပ်လိုက်တာနဲ့ ... ... ... "

- မရတော့ပါလား။ မနောက်ပဲစမ်းထားသေးတယ်။ ဘာဖြစ်တာပါလိမ့်။ ပြဿနာပဲ -

"ခက္ကလေးပါခင်ဗျာ။ Internet Connection ကြောင့်လားမသိဘူး။ Data Sync မဖြစ်ဘူးဖြစ်နေ တယ်"

- အင်တာနက်က ရနေရဲ့သားနဲ့ အလုပ်မလုပ်ဘူး။ ပြဿနာပဲ -

" ... ... ... "

"မင်းစနစ်ကဆိုးတော့မဆိုးပါဘူး။ ဒါပေမယ့် ပစ္စည်းပို့သမားကိုမပေးနိုင်ရင် ဆဲထားတဲ့ လမ်းကြောင်းက အသုံးမဝင်ဘူးလေ"

"ဟုတ်ပါတယ်ခင်ဗျာ။ ဒါကို ထည့်စဉ်းစားပြီးသာပါ။ မနောကတောင် အလုပ်လုပ်ပါသေးတယ်"

"အေး... အလုပ်လုပ်တယ်ဆိုရင် ပြလေ"

" ... ... ... "

"တစ်ကယ်ပါဗျာ... ကျိမ်ဆိုကျိမ်ပြပါမယ်။ မနောကတောင် စမ်းထားပါသေးတယ်။ အကောင်းကြီးပါ။ ခုမှ ဘာဖြစ်မှန်းမသိဘူး။ အလုပ်မလုပ်တော့ဘူး "

ဒီလိုအဖြစ်အပျက်မျိုးဟာ Software Developer တွေ Project ပြခါနီးတိုင်း ဒူးတုန်ရခြင်းရဲ့ အကြောင်းရင်းဖြစ်ပါတယ်။ ဘာမှ မှားစရာမရှိဘူးလို့ ယူဆထားတဲ့ စနစ်တစ်ခုက အရေးအကြောင်းမှ အလုပ်မလုပ်တော့တာမျိုး ဖြစ်တက်ပါတယ်။ မဖြစ်စလောက် လို့ယူဆရတဲ့ ပြင်ဆင်မှုလေးတစ်ခုကြောင့် စနစ်ကအလုပ်မလုပ်တော့မျိုး လည်း ဖြစ်နိုင်ပါတယ်။ မဆီမဆိုင်လို့ ယူဆရတဲ့ နေရာမှာ ပြဿနာသွားတက်နေတက်ပါတယ်။

ဘယ်လောက်ပဲ စနစ်ကျအောင် ရေးသားထားတဲ့ Code ဖြစ်ပါစေ၊ Code Base ကြီးလာတဲ့အခါမှာ ဒီလိုပြဿနာ တွေဟာ အနည်းနဲ့အများ ရှိလာတက်ကြတာပါပဲ။ ဒါကြောင့်ပဲ အလုပ်လုပ်နေတဲ့ Code ကို ထိရမှာ၊ ပြင်ရမှာကို Developer တွေ စိုးစွဲတက်ကြပါတယ်။ လုပ်ဆောင်ချက် (၁၀၀) လောက်ပါတဲ့ စနစ်ကြီးတစ်ခုမှာ ပြင်ဆင်မှု လေးတစ်ခု လုပ်လိုက်တိုင်း သေချာအောင်ဆိုပြီး လုပ်ဆောင်ချက် (၁၀၀) လုံးကို ပြန်လိုက်စမ်းရမလို့ဖြစ်နေပါတယ်။

Software တိုင်းမှာ ရှုတ်တရက်သတိမမှုမိန့်င်တဲ့ အမှားလေးတွေ ရှိတက်ပါတယ်။ Technical Debt တွေရဲ့ အကျိုးဆက်ပါ။ နောက်မှပြင်မယ်ဆိုပြီး အလွယ်ရေးလိုက်မိတဲ့ Code တစ်ခုက မထင်မှတ်တဲ့အချိန်မှာ ထပြီး ပြဿနာရှာခြင်းပဲဖြစ် ပါတယ်။ Technical Debt တွေကို အပြတ်ရှင်းဖို့ဆိုရင် Code ကို Refactor လုပ်ရပါတယ်။

Refactor ဆိတာ အလုပ်လုပ်နေဖြီးသား Code ကို (အလုပ်လုပ်ပုံမပြောင်းလဲစေပဲ) Code အရည်အသွေးတိုးတက်လာအောင် ပြင်ဆင်ခြင်း ဖြစ်ပါတယ်။ အရည်အသွေးကောင်းတဲ့ Quality Code ဆိတာ ရေးသားပုံရှင်းလင်းပြီး၊ နားလည်ရလွယ်တဲ့ Code ပြင်ဆင်ရလွယ်တဲ့ Code ကိုဆိုလိုတာပါ။

ဒီနေရာမှာ အရေးပါလာတဲ့ Principle နှစ်ခုကတော့ D.R.Y နဲ့ Decoupling တို့ပဲဖြစ်ပါတယ်။ ဒီနှစ်ခုအကြောင်းကို Professional Web Developer ရဲ့ အခန်း (၁၃) – Model-View-Controller မှာလည်း ပြောခဲ့ဖူးပါတယ်။

D.R.Y (Don't Repeat Yourself) ဆိုတာ၊ အရည်အသွေးကောင်းတဲ့ Code ဖြစ်ဖို့ Code Duplication ကို လျှော့ချရ မယ်ဆိုတဲ့ Principle ဖြစ်ပါတယ်။ Code Duplication လျှော့ချခြင်းဆိုတာ ထပ်ခါထပ်ခါလုပ်ဖို့လိုအပ်တဲ့ တူညီတဲ့လုပ်ဆောင်ချက်တွေကို အကြိမ်ကြိမ်ပြန်လည်ရေးသားမနေပဲ၊ တစ်ကြိမ်သာ သီးခြားခဲ့ပြီးရေးသားထားခြင်း ဖြစ်ပါတယ်။ အလားတူ လုပ်ဆောင်ချက် ထပ်မံဆောင်ချက်ဖို့လိုလာရင် အဲဒီခြားရေးသားတဲ့ Function (သို့မဟုတ်) Component ကို ခေါ်ယူ အသုံးပြုရမှာပါ။ ဒီတော့ အခြေအနေအရ ပြင်ဆင်ပြောင်း လဲဖို့လိုလာရင် ချွဲ့ခြားရေးသားထားတဲ့ တစ်နေရာမှာပြင်လိုက်ယုံပါပဲ။ အကယ်၍များ လုပ်ဆောင်ပုံတူညီတဲ့ကိစ္စတွေကို ဟိုနား ဒီနား ထပ်ခါထပ်ပါရောသားထားမိရင် နောင်လိုအပ်လို့ အဲဒီလုပ်ဆောင်ချက်ကို ပြင်ဆင်ပြောင်းလဲရတဲ့အခါ ရေးသားထားမိသမျှနေရာတိုင်းမှာ လိုက်ပြင်ရတော့မှာပါ။

Decoupling ဆိုတာကတော့ Function တစ်ခု (သို့မဟုတ်) Component တစ်ခုဟာ အလုပ်တစ်ခုကိုပဲ လုပ်သင့်တယ် ဆိုတဲ့ အမိပို့ယ်ဖြစ်ပါတယ်။ ပြီးတော့ တစ်ခြား Function တစ်ခုပေါ်မြို့ခို့နေခြင်း မရှိလောက်းလေဆိုတဲ့ အမိပို့ယ်လည်း ဖြစ်ပါတယ်။ အကယ်၍များ Function တစ်ခုရဲ့လုပ်ဆောင်ချက်က အခြား Function ကိုမြို့ခို့နေရင်၊ မြို့ခို့နေရတဲ့ Function ကို ပြင်ဆင်လိုက်တဲ့အခါ သူမှာလာပြီး သက်ရောက်နိုင်ပါတယ်။ ဒီတော့ ပြင်ချင်လိုတဲ့ Function ကတစ်ခု၊ ဒါပေမယ့် အဲဒီ Function ကိုမြို့ခို့နေတဲ့ Function ကိုပါ လိုက်ပြင်ရတော့မယ့်သဘော ဖြစ်နေတက်ပါတယ်။

Refactor လုပ်တယ်ဆိုရင် Code ထဲမှာရှိနေတဲ့ Duplicate တွေကို ရှာဖွေပြင်ဆင်ခြင်းနဲ့ Tightly Couple ဖြစ်နေတဲ့ Function တွေကို ပြင်ဆင်ရေးသားခြင်းအားဖြင့် Code Structure ကို ပြုပြင်ခြင်းဖြစ်ပါတယ်။ ဒီနေရာမှာ ဘတ်လမ်းစ တော့တာပါပဲ။ Refactor လုပ်မှ Technical Debt တွေနည်းမှာပါ။ ဒါပေမယ့် Refactor လုပ်လိုက်မှ မူလအလုပ်လုပ် နေတာတွေ၊ အလုပ်မလုပ်တော့တာမျိုးဖြစ်သွားတက်လို့ အရမ်းအရမ်းလဲ Refactor မလုပ်ရဲ ပြန်ပါဘူး။

Project ကိုစတင်စဉ်က ဘယ်လောက်ပဲသတိတွေထားပြီး Code အရည်အသွေးကို ထိမ်းသိမ်းလာခဲ့ပေမယ့် အချိန်ကြာလာတာနဲ့အမျှ Technical Debt တွေကတော့ ရှိလာတက်တာပါပဲ။ ကျွန်ုတော်တို့လိုအပ်နေတာက ရေးထားတဲ့ Code ကို ပြင်ဆင်မှုတစ်ခုလုပ်လိုက်တိုင်း အလိုအလျောက် စမ်းသပ်ပေးနိုင်မယ့် Test Method တွေ ဖြစ်ပါတယ်။ တစ်နေရာမှာပြင်လိုက်တဲ့အတွက် အခြားနေရာမှာသွားပြီး ပြသနာဖြစ်မနေကြောင်းကို အာမခံပေးနိုင်မယ့် စနစ်တွေ လိုအပ်နေတာပါ။

## 2.1 – Software Testing

Software တစ်ခုရဲ့အရည်အသေးကို ထိမ်းသိမ်းစီစစ်နိုင်ဖို့အတွက် Test လုပ်ပုံလုပ်နည်း အမျိုးမျိုးရှုပါတယ်။ သတ်မှတ်ထားတဲ့ Specification နဲ့ ကိုက်ညီခြင်းရှိမရှိ စမ်းသပ်တဲ့ System Test နဲ့ Acceptance Test တို့လို Test Method တွေရှိသလို၊ ရေးထားတဲ့ Code တွေ မှန်ကန်စွာအလုပ်လုပ်ကို Test လုပ်တဲ့ Unit Test, Integration Test, Regression Test စတဲ့ Method တွေလည်းရှုပါတယ်။

**Unit Test** – Function တစ်ခု (သို့မဟုတ်) Component တစ်ခုကို Test လုပ်ခြင်းဖြစ်ပါတယ်။ ဥပမာ - စက်ဘီး တစ်စီး တည်ဆောက်မယ်ဆိုရင်၊ တာယာကောင်းရဲ့လား၊ ကျွတ်လုံးရဲ့လား၊ ရွေ့ဖြောင့်ရဲ့လား စသဖို့ အစိတ်အပိုင်း တစ်ခုခြင်းကို စမ်းသပ်ခြင်းဖြစ်ပါတယ်။ အသေးဆုံးနဲ့ အခြေခံအကျဆုံး Unit တွေကို စမ်းသပ်ခြင်းဖြစ်ပါတယ်။

**Integration Test** – Function တစ်ခုနဲ့တစ်ခု ပေါင်းစပ်ပြီးတဲ့နောက် အားလုံးပုံမှန်အလုပ်လုပ်ရဲ့လား စမ်းသပ်တဲ့အဆင့် ဖြစ်ပါတယ်။ ဥပမာ - ရွေ့မှာ၊ ကျွတ်နဲ့တာယာတပ်ဆင်လိုက်တဲ့အခါ မျှော်မှန်းထားသလိုအလုပ်လုပ်ရဲ့လား၊ ကျွတ်သေးနေသလား၊ တာယာအံမကျဖြစ်နေသလား စပ်သပ်တဲ့အဆင့်ဖြစ်ပါတယ်။

**Regression Test** – အမှားတစ်ခု ပြင်ဆင်လိုက်တဲ့အခါ အဲဒီအမှားတစ်ကယ်ပြောလည်သွားရဲ့လား၊ ပြင်လိုက်တဲ့ အတွက် အခြားသက်ရောက်မှုတွေ ရှိနေသလား စမ်းသပ်ခြင်းဖြစ်ပါတယ်။ ဥပမာ - ရွေ့လိမ်နေလို့ ဖြောင့်လိုက်တဲ့အခါ ရွေ့ဖြောင့် သွားယုံသာမက တပ်ဆင်ထားတဲ့ ကျွတ်နဲ့တာယာပါ မူလအတိုင်း အဆင်ပြေပြေ ဆက်လက်အလုပ်လုပ်ရဲ့လား စစ်ဆေးခြင်းဖြစ်ပါတယ်။

**System Test** - Software မှာ Runtime Error တွေမရှိတာသေချာအောင် စမ်းသပ်ခြင်းဖြစ်ပါတယ်။ Specification က သတ်မှတ်ထားတဲ့ သတ်မှတ်ချက်တွေအတိုင်း မှန်မမှန်ကိုလည်း ဒီအဆင့်မှာစစ်ပါတယ်။

**Acceptance Test** - လိုအပ်တဲ့ လုပ်ဆောင်ချက်တွက် အမှန်တစ်ကယ် ပါဝင်ပြည့်စုံခြင်း ရှိမရှိ၊ လက်တွေ အသုံးဝင်ခြင်း ရှိမရှိ စစ်ဆေးခြင်းဖြစ်ပါတယ်။ အသုံးပြုသူကိုယ်တိုင် စမ်းတဲ့အဆင့်ပါ။

ဒီထဲကမှ Code Quality အတွက်အခြေခံအကျဆုံးနဲ့ ထိရောက်မှုအရှိဆုံးကတော့ Unit Test ပဲဖြစ်ပါတယ်။ Unit Test က နောက်ဆုံးရရှိလာတဲ့ Software Product ဟာ Specification နဲ့ကိုက်ညီမှုရှိမရှိ၊ လက်တွေအသုံးဝင်မ ဝင်လို ကိစ္စမျိုးကိုတော့ အာမခံမပေးပါဘူး။ ဒါပေမယ့် ရေးထားတဲ့ Code အစိတ်အပိုင်းတိုင်းဟာ မျှော်မှန်းထားတဲ့အတိုင်း မှန်ကန်စွာ အလုပ်လုပ်နေကြောင်းကိုတော့ အာမခံပေးနိုင်မှာဖြစ်ပါတယ်။ Unit Test က Code Quality ကို အာမခံပေးပါတယ်။

## 2.2 – Test Codes

Unit Test အကြောင်းမလေ့လာခင်၊ ပိုမြီးအခြေခံကျတဲ့ Test Code တွေအကြောင်းကို အရင်ဖော်ပြလိပါတယ်။ Test Code ဆိုတာ Software ရဲ့ လုပ်ဆောင်ချက်နဲ့ မသက်ဆိုင်ပဲ စမ်းသပ်စိစစ်မှုလုပ်ဖို့ ရည်ရွယ်ချက်နဲ့ ထည့်သွင်းရေးသား ထားတဲ့ Code တွေကိုဆိုလိုတာပါ။ Test Code တွေထဲမှာ အခြေခံ အကျဆုံးကတော့ Debug Log ဖြစ်ပါတယ်။

Code နဲ့နာတွေဖော်ပြတဲ့အခါ။ အချို့ Code တွေဟာ လက်တွေ့ကူးယူစစ်းသပ်လို့ရတဲ့ Code တွေ ဖြစ်ပြီး၊ အချို့ Code တွေကတော့ ဆိုလိုရင်း သဘောသဘာဝကို ပေါ်လှင်အောင် နဲ့နာ အနေနဲ့ ရေးပြတဲ့ Pseudo-Code တွေဖြစ်ပါတယ်။ လက်တွေ့စစ်းသပ်တဲ့အခါ။ ပေးထားတဲ့ အတိုင်း တိုက်ရှိက် ကူးယူမယ့်အစား နဲ့နာ Code ကို နားလည်အောင်ဖတ်ပြီး၊ ဆိုလိုရင်းကို နားလည်ပြီဆိုမှ ကိုယ်တိုင် ရေးသား စမ်းသပ်သင့်ပါတယ်။

ဥပမာ - လက်ရှိအပူချိန်ကို ရယူပြီး သတ်မှတ်အပူချိန်ကို ကျော်လာတဲ့အခါ Warning Message ပေးတဲ့ လုပ်ဆောင်ချက် တစ်ခုဆိုပါမြို့။ ဒီလုပ်ဆောင်ချက်ရှိရှိအတွက် Function နှစ်ခုလုပ်နိုင်ပါတယ်။ လက်ရှိအပူချိန် ရယူပေးတဲ့ Function နဲ့ ရရှိလာတဲ့အပူချိန်ကို စီစစ်ပေးတဲ့ Function တို့ ဖြစ်ပါတယ်။ JavaScript ကိုအသုံးပြုပြီး Code နဲ့နာဖော်ပြရရင် အခုလိုဖြစ်မှာပါ။

### Pseudo-code

```

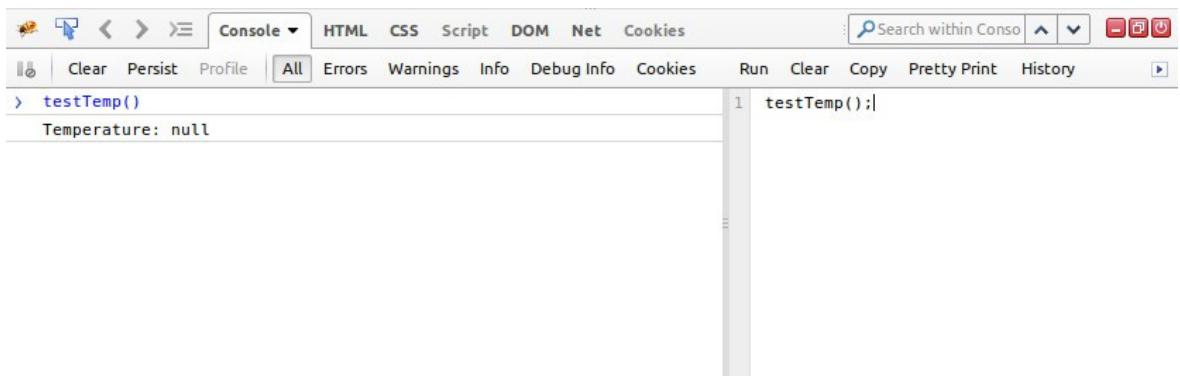
1. function getTemp() {
2.   return getTempFromServer("/path/to/api");
3. }
4.
5. function testTemp() {
6.   var temp = getTemp();
7.   if(temp.degC > 32) print("Warning: High Temperature");
8. }
```

နဲ့နာအရ getTemp() Function က လက်ရှိအပူချိန်ရယူပေးတဲ့ လုပ်ဆောင်ချက်ကိုလုပ်ပြီး၊ testTemp() Function ကတော့ ရလာတဲ့ Temperature ကိုစီစစ်ပေးပါတယ်။ အပူချိန်က 32 Degree ကို ကျော်လာရင် Warning Message ဖော်ပြပေးမှာဖြစ်ပါတယ်။ testTemp() Function ကို Evaluate လုပ်လိုက်တဲ့အခါ မျှော်မှန်းထားသလို အလုပ်မလုပ်ရင် ဖြစ်နိုင်ခြေ အမှားနှစ်ခုရှိပါတယ်။ getTemp() Function က လက်ရှိ အပူချိန်ကို ယူမပေးနိုင်တာဖြစ် နိုင်သလို၊ testTemp() Function ကိုယ်တိုင်မှာ အမှားရှိနေတာလည်း ဖြစ်နိုင်ပါတယ်။ ဒါကိုသေချာစေဖို့အတွက် အခုလို Debug Log တစ်ခု ထည့်လိုက်နိုင်ပါတယ်။

## Pseudo-code

```
5. function testTemp() {  
6.   var temp = getTemp();  
7.   console.log("Temprature: " + temp.degC);  
8.   if(temp.degC > 32) print("Warning: High Temperature");  
9. }
```

လိုင်းနံပါတ် (၇) မှာ console.log ကိုသုံးပြီး temp ခဲ့လက်ရှိတန်ဖိုးကို Console မှာ ဖော်ပြခေါ်ခြင်း ဖြစ်ပါတယ်။



ü(J.º) - Console Log in Firebug Console

testTemp () Function ကို Evaluate လုပ်ကြည့်လိုက်လို ပုံ (ပ.၁) မှာတွေ့ရတဲ့ ရလဒ်မျိုး ရရှိတယ်ဆိုရင် temp Variable ရဲ့ တန်းဖိုးဟာ null ဖြစ်နေကြောင်း သိရနိုင်ပါတယ်။ ဒါကြောင့် ပြဿနာက getTemp () Function က ပြန်ပေးတဲ့ တန်းဖိုး မမှန်လို ဖြစ်ကြောင်း သိရှိရမှာပဲ ဖြစ်ပါတယ်။ ဒါလို ကြောင်း သားတဲ့ Debug Log တွေ ကို အခြေခံအကျဆုံး Test Code တွေလို မှတ်ယူနိုင်ပါတယ်။

နောက်ထပ်အရေးပါတဲ့ Test Code တွေကတော့ **Stub** နဲ့ **Mock** တို့ပဲဖြစ်ပါတယ်။ အမှန်တစ်ကယ် အလုပ်မလုပ်ပဲ၊ မူရင်း Function ကို တုပထားတဲ့ Fake Function တွေဖြစ်ပါတယ်။ Stub နဲ့ Mock က အနည်းငယ်ကဲာပါတယ်။ Stub Function တစ်ခုဟာ အခြေခံအားဖြင့် ဘာအလုပ်မှမလုပ်ပါဘူး။ မူရင်း Function က ပေးနိုင်တဲ့ Return Value ကိုပဲ တိုက်ရှိကဲ ပြန်ပေးတဲ့ Function အတွက်တစ်မျိုးဖြစ်ပါတယ်။ Mock ကတော့ Stub လို့ တန်ဖိုးတစ်ခု ပြန်ပေးယုံမဟုတ်ပဲ၊ မူရင်း Function ရဲ့ အလုပ်လုပ်ပုံကိုပါ တုပထားတက်ပါသေးတယ်။ Code နှမှနာတစ်ချို့ ဖော်ပြုပါမယ်။

## Pseudo-code

```

1. function getTempStub() {
2.   console.log("Working with stub: getTemp()");
3.   return {
4.     degc: 24,
5.     degf: 45
6.   }
7. }
8.
9. function getTempMock(degc) {
10.  console.log("Working with mock: getTemp()");
11.  return {
12.    degc: degc,
13.    degf: Math.round((degc * (5/9) + 32))
14.  }
15. }
16.
17. function getTemp() {
18.  return getTempFromServer("/path/to/api");
19. }

```

ပေးထားတဲ့ နူးနာမှာ getTemp() Function က မူရင်း Function ဖြစ်ပါတယ်။ မူရင်း Function က Server ကို Request ပြုလုပ်ပြီး လက်ရှိအပူချိန်ကို ရယူတဲ့ အလုပ်ကို ဆောင်ရွက်ပါတယ်။ ပြဿနာကာ ဒီ Function ကို စမ်းချင်တဲ့ အခါ တိုင်းမှာ အင်တာနက် အဆက်အသွယ် လိုနိုင်ပါတယ်။ ဒီတော့ getTemp() Function ကို စမ်းချင်တဲ့ အခါ တိုင်းမှာ အင်တာနက် အဆက်အသွယ်ရှိပဲ စမ်းလို့ ရတော့မလို ဖြစ်နေပါတယ်။ ဒါကြောင့် getTempStub() ဆိုတဲ့ Stub Function တစ်ခုကို သီးခြားသတ်မှတ်ထားခြင်းဖြစ်ပါတယ်။ getTempStub() Function က getTemp() ကို တုပထားခြင်း ဖြစ်ပြီး တစ်ကယ်အလုပ်မလုပ်ပါဘူး။ getTemp() Function က ပြန်ပေးမယ့် တန်ဖိုးနဲ့ ပုံစံတူတန်ဖိုးတစ်ခုကို ပြန်ပေးတဲ့ အလုပ်ကို လုပ်ပေးခြင်း ဖြစ်ပါတယ်။ ဒါကြောင့် getTemp() Function ကို ခေါ်ယူအသုံးပြုဖို့လိုတဲ့ နေရာ တိုင်းမှာ getTempStub() Function ကို အစားထိုးအသုံးပြုခြင်း အားဖြင့် ကျွန်ုပ်ဆောင်ချက်တွေကို အင်တာနက် အဆက်အသွယ် မလိုတော့စမ်းနှင့် သွားမှာ ဖြစ်ပါတယ်။

getTempMock() ဆိုတဲ့ အမည်နဲ့ လည်း Function တစ်ခုသတ်မှတ်ထားပါသေးတယ်။ သူလည်းပဲ စမ်းသပ်ဖို့ ရည်ရွယ်ချက်နဲ့ getTemp() ကို တုပထားတဲ့ Function တစ်ခုပဲ ဖြစ်ပါတယ်။ ဒါပေမယ့် သူကတော့ getTempStub() လို တန်ဖိုးကို ဒီအတိုင်း ပြန်ပေးတာမဟုတ်တော့ပါဘူး။ သူကို ခေါ်ယူစဉ်မှာ Degree Celsius တန်ဖိုးကို လက်ခံပြီး၊ getTemp() Function က ပြန်ပေးမယ့် Object နဲ့ အလားတူ Object တစ်ခု ရအောင် အဲဒီ Degree Celsius ကို အသုံးပြုတည်ဆောက်ယူပါတယ်။ Return Value တစ်ခုပြန်ရယုံနဲ့ လုံလောက်တဲ့ အခြေအနေတွေမှာ getTempStub() ကို သုံးနိုင်ပြီး Return Value အမျိုးမျိုးနဲ့ စမ်းသပ်ဖို့လိုတဲ့ အခါ အသုံးပြုနိုင်မှာ ပဲ ဖြစ်ပါတယ်။

အမှန်တော့ Stub တွေ Mock တွေဟာ မူရင်း Function ကို တုပယ့်သက်သက်နဲ့ စမ်းသက်ယုံသက်သက်အတွက် တင် အသုံးဝင်တာမဟုတ်ပါဘူး။ သူတို့ကို Function Model အနေနဲ့ လည်း ယူဆနိုင်ပါတယ်။ မူရင်း Function ဖြစ်တဲ့ getTemp() မှာ လိုအပ်တဲ့ လုပ်ဆောင်ချက်တွေကို ရေးသားထားပေမယ့် နောက်ဆုံးရမယ့် Return Value ဟာ ဘယ်လိုဘယ်ပဲ ဖွဲ့စည်းပြီး ဘယ်လိုတန်ဖိုး ဖြစ်မယ်ဆိုတဲ့ သက်မှတ်ချက် မပါဝင်ပါဘူး။ getTempStub() Function က getTemp() ကို တုပထားယုံသက်သက်မဟုတ်ပါဘူး။ Return Value ရဲ့

ဖွဲ့စည်းပုံကိုပါ သတ်မှတ်ထားတဲ့သဘောဖြစ် ပါတယ်။ `getTempStub()` ကို ကြည့်ပြီး `getTemp()` Function က ပြန်ပေးမယ့်တန်ဖိုးဟာ ဘယ်လိုပုံစံ ဖြစ်မယ်ဆိုတာကို သိရှိနိုင်မှာဖြစ်ပါတယ်။ အလားတူပဲ `getTempMock()` Function ကို ကြည့်ပြီး `getTemp()` Function ရဲ့ တစ်ကယ့် အလုပ်လုပ်ပုံကို သိရှိနိုင်ပါ တယ်။ ဒါကြောင့် Stub တွေ Mock တွေကို စမ်းသပ်ယုံ၊ တုပယုံ သက်သက် သဘောမထားပဲ Function Model လိုလည်း သဘောထားနိုင်ပါတယ်။

Stub တွေ Mock တွေကို နေရာတိုင်းမှုသုံးဖို့တော့မလိုပါဘူး။ သုံးလေ့လည်းမရှိကြပါဘူး။ အထူးသဖြင့် Remote Data Source ကို ချိတ်ဆက်အလုပ်လုပ်ရမယ့် နေရာတွေမှာသာ အဲဒါ Remote Data Source ကို အမှန် တစ်ကယ်ချိတ်ဆက် နေစရာမလိုပဲ စမ်းသပ်နိုင်အောင် အသုံးပြုလေ့ရှိကြပါတယ်။ Server Request လုပ်တဲ့ Function တွေ၊ Database နဲ့ ချိတ်ဆက်အလုပ်လုပ်တဲ့ Function တွေနဲ့ GPS တို့ Fingerprint တို့လို External Input Device တွေနဲ့ ချိတ်ဆက် အလုပ်လုပ်ရတဲ့ နေရာတွေမှာသုံးကြလေ့ရှိပါတယ်။

## 2.3 Unit Testing

Unit Testing ဆိုတာကတော့ Function တစ်ခု (Unit တစ်ခု) ဟာ မျှော်မှန်းထားသလို အလုပ်လုပ်ရဲ့လားလို့ စမ်းသပ်တဲ့ နည်းစနစ်တစ်ခုဖြစ်ပါတယ်။ တစ်နည်းအားဖြင့် Function တစ်ခုကို စမ်းသပ်ပေးတဲ့ Test Function တစ်ချိုးဖြစ်ပါတယ်။ ဥပမာ - ပေးလိုက်တဲ့ရက်စွဲနှစ်ခုကို နှိုင်းယူဉ်ပြီး၊ "just now", "3 hours ago", "1 week ago" စသဖြင့် ကြည့်ကောင်း အောင် Format လုပ်ပြီး ပြန်ပေးတဲ့ Function တစ်ခုရှိတယ်ဆိုပါစိုး။

### JavaScript

```

1. function prettyDate(now, time) {
2.   var date = new Date(time || ""),
3.     diff = (((new Date(now)).getTime() - date.getTime()) / 1000),
4.     day_diff = Math.floor(diff / 86400);
5.
6.   return day_diff == 0 && (
7.     diff < 60 && "just now" ||
8.     diff < 120 && "1 minute ago" ||
9.     diff < 3600 && Math.floor(diff / 60) + " minutes ago" ||
10.    diff < 7200 && "1 hour ago" ||
11.    diff < 86400 && Math.floor(diff / 3600) + " hours ago"
12.  ) ||
13.  day_diff == 1 && "Yesterday" ||
14.  day_diff < 7 && day_diff + " days ago" ||
15.  day_diff < 31 && Math.ceil(day_diff / 7) + " weeks ago";
16. }

```

ဒါ Function က ရက်စွဲနှစ်ခုလက်ခံပါတယ်။ လက်ရှိအချိန် now နဲ့ နှိုင်းယူဉ်လိုတဲ့အချိန် time တို့ပဲဖြစ်ပါတယ်။ ရက်စွဲ နှစ်ခုကို ပေးလိုက်ချိန်မှာ လိုချင်တဲ့ရလဒ်ကို မှန်မှန်ကန်ကန် ပြန်ပေးနိုင်ရဲ့လား စမ်းသပ်ပေးတဲ့ Test Function တစ်ခုကို အချင်းရေးသားနိုင်ပါတယ်။

## JavaScript

```

1. function prettyDateTest(then, expected) {
2.     var result = prettyDate("2014/09/10 22:25:00", then);
3.     if (result !== expected) {
4.         console.log("Fail: Expected " + expected + ", but was " + result);
5.     } else {
6.         console.log("Pass: Expected " + expected + ", and was " + result);
7.     }
8. }

```

prettyDate() Function ကပြန်ပေးတဲ့တန်ဖိုးနဲ့ ကိုက်ညီရမယ်လို သတ်မှတ်ထားတဲ့ တန်ဖိုးကို တိုက်ဆိုင်စစ်ဆေးပေးခြင်းဖြစ်ပါတယ်။ ဒါကြောင့် အောက်ပါအတိုင်း ရေးသားစမ်းသပ်နိုင်ပြီဖြစ်ပါတယ်။

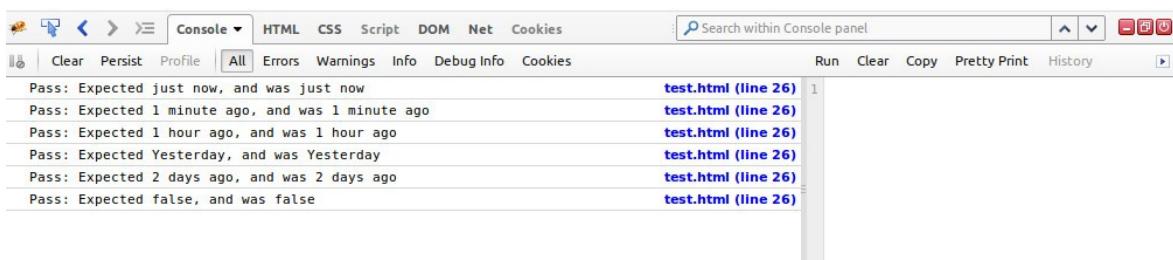
## JavaScript

```

1. prettyDateTest("2014/09/10 22:24:30", "just now");
2. prettyDateTest("2014/09/10 22:23:30", "1 minute ago");
3. prettyDateTest("2014/09/10 21:23:30", "1 hour ago");
4. prettyDateTest("2014/09/09 22:23:30", "Yesterday");
5. prettyDateTest("2014/09/08 22:23:30", "2 days ago");
6. prettyDateTest("2013/09/08 22:23:30", false);

```

prettyDateTest() Function ရဲ့ ပထမ Parameter အနေနဲ့ လက်ရှိအချိန်ကိုပေးပြီး၊ ရလဒ်အနေနဲ့ "just now" ရရှိရမယ်လို စစ်ထားပါတယ်။ လွန်ခဲ့တဲ့ တစ်မိန့်ကအချိန်ကိုပေးလိုက်ရင်တော့ "1 minute ago" ကို ရရှိရမယ်လို စစ်ထားပါတယ်။ ဒီနည်းနဲ့ ထောင့်စိအောင်၊ မဖြစ်နိုင်တဲ့ ရက်စွဲတွေအပါအဝင် ရက်စွဲအမျိုးမျိုးနဲ့ အချိန်အမျိုးမျိုးကို ထည့်သွင်းစမ်းသပ်နိုင်ပါတယ်။ ရရှိလာမယ့်ရလဒ်နမူနာကို ပုံ (J.J) မှာ ဖော်ပြထားပါတယ်။



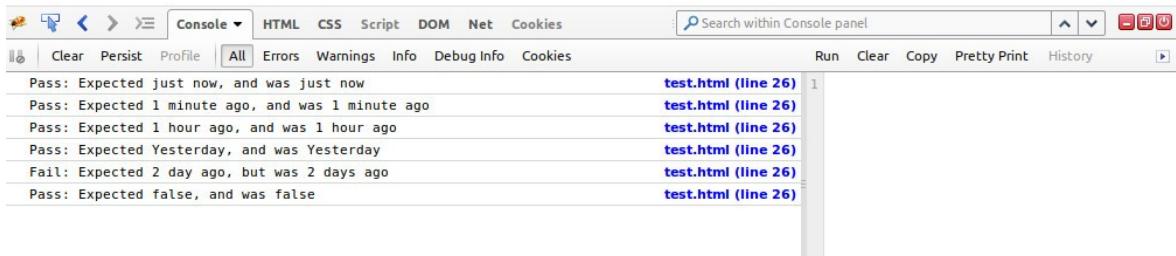
ပုံ (J.J) - Unit Test Function: All Pass

ပေးလိုက်တဲ့ ရက်စွဲအားလုံးအတွက် မျှော်မှန်းထားတဲ့ရလဒ်အတိုင်း အားလုံးမှန်မှန်ကန်ကန် ရရှိနေကြောင်းကို ဒီနည်းနဲ့ သိနိုင်ပါတယ်။ အမှားရှိတဲ့အခါ ဘယ်လိုဖော်ပြမလဲ သိလိုရင်အခုလို မျှော်မှန်းရလဒ်အမှားတစ်ခုနဲ့ စမ်းသပ်ကြည့်နိုင်ပါတယ်။

## JavaScript

```
5. prettyDateTest("2014/09/08 22:23:30", "2 day ago");
```

မျှော်မှန်းရလဒ်က **2 day ago** ဖြစ်ပေမယ့် prettyDate() Function ကပြန်ပေးတဲ့ တစ်ကယ့်ရလဒ်က **2 days ago** ဖြစ်နေတဲ့အတွက် ဒီတစ်ချက်မှာ မှားနေကြောင်းကို အခုလိုတွေ့ရမှာဖြစ်ပါတယ်။



## Ø (J.2) - Unit Test Function: One Fail

## 2.4 – Test Frameworks

Test Framework တွေက Test Function တွေ တစ်ခုချင်း ကိုယ်တိုင်လိုက်ရေးနေစရာမလိုပဲ Unit Test လုပ်နိုင် စေမယ့် လုပ်ဆောင်ချက်တွေ ပါဝင်လေ့ရှိပါတယ်။ အခြေအနေအမျိုးမျိုး၊ Condition အမျိုးမျိုးအတွက် Test လုပ်လိုရစေမယ့် လုပ်ဆောင်ချက်တွေ ပါဝင်လေ့ရှိပါတယ်။ Test Result ကို စနစ်တစ်ကျ စုစုပေါင်းဖော်ပြပေးတဲ့ Result Formatter တွေ ပါဝင်လေ့ရှိကြပါတယ်။

QUnit လိုခေါ်တဲ့ JavaScript Unit Testing Framework တစ်ခုကို နမူနာအနေဖော်ဖြေပေးပါမယ်။ Framework သုံးနည်းကို ဖော်ပြလိုတာမဟုတ်ပါဘူး။ Test Framework တွေရဲ့သဘာဝကို သိရှိသွားစေဖိုက အခိုက ရည်ရွယ်ချက် ဖြစ်တဲ့ အတွက် သဘောသဘာဝပိုင်းကို ဦးစားပေးလေ့လာစေလိုပါတယ်။ လက်တွေ့မှာ အသုံးပြုတဲ့ Language နဲ့ Project အမျိုးအစား ပေါ်မှတည်ပြီး အသုံးပြုရမယ့် Test Framework လည်းကွဲပြားသွားမှာပဲဖြစ်ပါတယ်။

QUnit ကို jQuery ဖန်တီးရှင်ဖြစ်သူ John Resig ကဖန်တီးထားခြင်းဖြစ်ပါတယ်။ jQuery, jQueryUI နဲ့ အခြား JavaScript Project အများအပြားအတွက် Test Framework အနေဖြင့်အသုံးပြု ထားပါတယ်။ QUnit ကို jQuery CDN ကနေ Download ရယူနိုင်ပါတယ်။

လက်ရှိခိုစာရေးသားနေစဉ်မှာ နောက်ဆုံးထွက်ရှိထားတဲ့ Version ကတော့ QUnit 1.15.0 ဖြစ်ပါတယ်။ စတင် အသုံးပြန်စိုင် ဖို့ အတွက် JavaScript Source Code သာမက Test Result ကို Format လုပ် ဖော်ပြရာမှာအသုံးပြ တဲ့ CSS Style ကိုပါ ရယူဖို့လိုအပ်ပါတယ်။ တိုက်ရှိက်ရယူနိုင်မယ့် Link တွေကို ဖော်ပြပေးလိုက်ပါတယ်

- <http://code.jquery.com/qunit/qunit-1.15.0.js>
- <http://code.jquery.com/qunit/qunit-1.15.0.css>

လိုအပ်တဲ့ဖိုင်တွေရယူပြီးရင် စတင်စမ်းသပ်နိုင်ဖို့အတွက် HTML Document တစ်ခုတည်ဆောက်ပြီး QUnit JavaScript File နဲ့ CSS Style File တို့ကို အခုလိုချိတ်ဆက်ပေးရပါတယ်။

## HTML

```

1. <!DOCTYPE html>
2. <html>
3. <head>
4.   <meta charset="utf-8">
5.   <title>QUnit Test</title>
6.   <link rel="stylesheet" href="css/qunit.css">
7. </head>
8. <body>
9.   <div id="qunit"></div>
10.
11.  <script src="js/qunit.js"></script>
12.  <script src="js/app.js"></script>
13.  <script src="js/tests.js"></script>
14. </body>
15. </html>

```

နမူနာရဲ့ လိုင်းနံပါတ် (၉) မှာ <div> Element တစ်ခုတည့်သွင်းထားပြီး id ကို qunit လိုပေးထားပါတယ်။ QUnit က Test Result တွေကို အဲဒီ <div> ထဲမှာ ဖော်ပြပေးမှုဖြစ်ပါတယ်။ လိုင်းနံပါတ် (၁၆) မှာ CSS Style ဖိုင်ကို ချိတ်ဆက်ထားပြီး လိုင်းနံပါတ် (၁၁) မှာ QUnit JavaScript ဖိုင်ကိုချိတ်ဆက်ထားပါတယ်။ QUnit ရဲ့ အကူအညီနဲ့ Test Code တွေကို မူရင်း Code နဲ့ ရောထားစရာမလိုပဲ သို့ခြားခဲ့ပြီး ရေးသားနိုင်မှုဖြစ်ပါတယ်။ ဒါကြောင့် လိုင်းနံပါတ် (၁၂) မှာ ချိတ်ဆက်ထားတဲ့ app.js ထဲမှာ လက်တွေ့အလုပ်လုပ်မယ့် Code တွေကို ရေးသားမှုဖြစ်ပြီး၊ လိုင်းနံပါတ် (၁၃) မှာ ချိတ်ဆက်ထားတဲ့ tests.js မှာတော့ Test Code တွေကို ခွဲခြားရေးသားမှုဖြစ်တယ်လို့ မှတ်ယူပေးပါ။

စောစောကန်မူနာအဖြစ် ဖော်ပြခဲ့တဲ့ prettyDate() Function ကိုပဲ ဆက်လက်စမ်းသပ်ကြည့်ပါမယ်။ Unit Test လုပ်ဖို့အတွက် prettyDateTest() Function မလိုတော့ပါဘူး။ QUnit အကူအညီနဲ့ Test Code တွေ ကို ရေးသွားတော့မှာပါ။

QUnit ရဲ့ test() Function ကိုသုံးပြီး Test Code တွေကိုစတင်ရေးသားနိုင်ပါတယ်။ test() Function အတွက် Parameter နှစ်ခုပေးရပါတယ်။ ပထမတစ်ခုက Test Name ဖြစ်ပြီး ဒုတိယတစ်ခုကတော့ Test Case တွေပါဝင်တဲ့ Function ဖြစ်ပါတယ်။ QUnit မှာ ok(), equal(), deepEqual(), notEqual(),

notDeepEqual() စသဖြင့် မျှော်မှန်းရလဒ်နဲ့ လက်တွေ့ရလဒ်ကို တိုက်စစ်လို့ရတဲ့ Function အမျိုးမျိုးရှိပါတယ်။ ok() ကတော့ ရလာတဲ့ရလဒ်က true ဟုတ်မဟုတ် စစ်ပေးပါတယ်။ true ဆိုရင် Test Pass ဖြစ်ပြီး ကျွန်ုရလဒ် တွေဆိုရင်တော့ Test Fail ဖြစ်မှာပဲဖြစ်ပါတယ်။ equal() နဲ့ notEqual() Function တွေ ကတော့ ရလာတဲ့ တန်ဖိုးဟာ မျှော်မှန်းထားတဲ့တန်ဖိုးနဲ့ ညီမညီကိုစစ်ပေးပါတယ်။ deepEqual() နဲ့ notDeepEqual() တို့ကတော့ အခြေခံအားဖြင့် equal(), notEqual() တို့နဲ့အတူတူပါပဲ။ ဒါပေမယ့် နှင့်ယူဉ်လိုတဲ့ရလဒ်က Object တစ်ခု ဖြစ်မယ်ဆိုရင်တော့ deepEqual() သို့မဟုတ် notDeepEqual() ကို အသုံးပြုသင့်ပါတယ်။ အခြား Function တွေရှိသေးပေမယ့် အခြေခံအားဖြင့် ဒီလောက်ဆိုရင် Test Code တွေကို QUnit အကူအညီနဲ့ စတင်ရေးသားနိုင်ပြီဖြစ်ပါ တယ်။ tests.js ထဲမှာ အခုလိုရေးသားစမ်းသပ် ကြည့်နိုင်ပါတယ်။

### JavaScript

```
1. QUnit.test("Prettydate Basics", function( assert ) {
2.   var now = "2014/09/10 22:25:00";
3.   assert.equal(prettyDate(now, "2014/09/10 22:24:30"), "just now");
4.   assert.equal(prettyDate(now, "2014/09/10 22:23:30"), "1 minute ago");
5.   assert.equal(prettyDate(now, "2014/09/10 21:23:30"), "1 hour ago");
6.   assert.equal(prettyDate(now, "2014/09/09 22:23:30"), "Yesterday");
7.   assert.equal(prettyDate(now, "2014/09/08 22:23:30"), "2 day ago");
8.   assert.equal(prettyDate(now, "2013/09/08 22:23:30"), false);
9. });


```

နဲ့မှန်သူ equal() ကိုသုံးပြီး prettyDate() Function ကို ခေါ်စဉ်မှာ ရရှိလာမယ့် လက်တွေ့ရလဒ်ဟာ မျှော်မှန်းရလဒ်နဲ့ တူညီခြင်းရှိမရှိစိစစ်ထားပါတယ်။ လက်တွေ့ရလဒ်က မျှော်မှန်းထားတာနဲ့မကိုက်ညီပဲ အမှား ရှိနေခဲ့ရင် QUnit က အခုလိုဖော်ပြပေးမှာဖြစ်ပါတယ်။

QUnit Example

Hide passed tests  Check for Globals  No try-catch

Mozilla/5.0 (X11; Ubuntu; Linux x86\_64; rv:32.0) Gecko/20100101 Firefox/32.0

Tests completed in 21 milliseconds.  
5 assertions of 6 passed, 1 failed.

1. Prettydate Basics (1, 5, 6) Rerun 3 ms

1. okay
2. okay
3. okay
4. okay
5. failed

**Expected:** "2 day ago"  
**Result:** "2 days ago"  
**Diff:** "2 day days ago"  
**Source:** @file:///home/eimg/Desktop/qunit/js/tests.js:7:2

6. okay

ဗု (၂.၄) - QUnit - Test Result - One Failed

ခြောက်ကြိမ်စမ်းသပ်မှုမှာ တစ်ကြိမ် Fail ဖြစ်နေကြောင်းဖော်ပြတဲ့အပြင် Fail ဖြစ်ပုံဖြစ်နည်းအသေးစိတ်ကိုပါတွဲဖက်ဖော်ပြထားပါတယ်။ **2 day ago** ဆိတဲ့ရလဒ်ကို မျှော်လင့်ထားပေမယ့် တစ်ကယ့်ရလဒ်က **2 days ago** ဖြစ်နေကြောင်း ကို တွဲဖက်ဖော်ပြနေခြင်းဖြစ်ပါတယ်။

အကယ်၍ Test တွေအားလုံး Pass ဖြစ်တယ်ဆိုရင်တော့ ပုံ (၂.၅) မှာဖော်ပြထားသလိုရလဒ်ကို ရရှိမှုဖြစ်ပါတယ်။ ခြောက်ကြိမ်စမ်းရာမှာ ခြောက်ကြိမ်လုံး Pass ဖြစ်ပြီး Fail တစ်ကြိမ်မှမဖြစ်တဲ့အတွက် စမ်းသပ်မှုအောင်မြင်ကြောင်း ကိုဖော်ပြခြင်းဖြစ်ပါတယ်။

ပုံ (၂.၅) - QUnit - Test Result - All Pass

တစ်ခါတစ်ရုံမှာ မမျှော်မှန်းနိုင်တဲ့အမှားတစ်ချို့ကြောင့် Test (၆) ကြိမ်လုပ်ခိုင်းထားပေမယ့် (၆) ကြိမ်လုံးမစမ်းဖြစ်ပဲ ကျွန်ုဘားတာမျိုးလည်း ဖြစ်တက်ပါတယ်။ အဲဒီလိုကျွန်ုခဲ့ရင်လည်း Test Fail ဖြစ်ကြောင်းဖော်ပြဖို့အတွက် expect () Function ကို သုံးနိုင်ပါတယ်။

### JavaScript

```
1. QUnit.test("Prettydate Basics", function( assert ) {
2.   expect(6);
3.   var now = "2014/09/10 22:25:00";
4.   assert.equal(prettyDate(now, "2014/09/10 22:24:30"), "just now");
5.   assert.equal(prettyDate(now, "2014/09/10 22:23:30"), "1 minute ago");
6.   assert.equal(prettyDate(now, "2014/09/10 21:23:30"), "1 hour ago");
7.   assert.equal(prettyDate(now, "2014/09/09 22:23:30"), "Yesterday");
8.   assert.equal(prettyDate(now, "2014/09/08 22:23:30"), "2 days ago");
9. });
```

The screenshot shows a QUnit test result for a module named 'Prettydate Basics'. The test has 6 assertions, 5 of which passed and 1 failed. The passed assertions are listed as 'okay'. The failed assertion is 'Expected 6 assertions, but 5 were run'. The test was completed in 20 milliseconds on a Mozilla/5.0 (X11; Ubuntu; Linux x86\_64; rv:32.0) Gecko/20100101 Firefox/32.0 environment.

### ပုံ (၂.၆) - QUnit - Test Result - Expect Failed

(၆) ကြိမ်စမ်းဖို့မျှော်မှန်းထားပေမယ့် (၅) ကြိမ်ပဲစမ်းဖြစ်တဲ့အတွက် Test Fail ဖြစ်ကြောင်းဖော်ပြလာခြင်းဖြစ်ပါတယ်။

Test လုပ်ဖို့အတွက် Function တစ်ခုကို QUnit ကနေခေါ်လိုက်တဲ့အခါ အဲဒီ Function က တန်ဖိုးတစ်ခု ပြန်ပေးယုံ မဟုတ်ပဲ တစ်ခြားလုပ်ဆောင်ချက်တစ်ခုကို ဆောင်ရွက်သွားတဲ့အခါ Side-Effect တွေရှိတက်ပါတယ်။ ဥပမာ - Database ထဲကို Record တစ်ခုထည့်သွင်းပေးတဲ့ Function အလုပ်လုပ်မလုပ် စမ်းဖို့အတွက် အဲဒီ Function ကို ခေါ် လိုက်တဲ့ အခါ Record ကိုလည်း Database ထဲမှာ ထည့်သွင်းသွားမှာမို့ Side-Effect ရှိနေပါတယ်။ ကျွန်ုတ်တို့က Record ထည့်ချင်တာမဟုတ်ပါဘူး။ Record ထည့်တဲ့အလုပ် တစ်ကယ်လုပ်မလုပ်ကိုပဲ Test လုပ်ချင်တာပါ။

Side-Effect တွေကို စီမံဖို့အတွက် QUnit မှာ Module, Setup နဲ့ Teardown ဆိုတဲ့လုပ်ဆောင်ချက်တွေ ပါဝင်ပါတယ်။

QUnit မှာအမျိုးတူရာ Test Code တွေကို စုစည်းဖို့အတွက် Module တွေသတ်မှတ်လိုပါတယ်။ QUnit.module() များကိုတော်လှုပါတယ်။ အောက်မှာဆက်လက်ရေးသားထားတဲ့ Test Code တွေကို သက်ဆိုင်ရာ Module Name အလိုက် စုစည်း အလုပ်လုပ်ပေးသွားမှာဖြစ်ပါတယ်။ ဥပမာ -

## JavaScript

```

1. QUnit.module( "Module A" );
2. QUnit.test("Test One", function( assert ) {
3.   //
4. });
5. QUnit.test("Test Two", function( assert ) {
6.   //
6. });
7.
8. QUnit.module( "Module B" );
9. QUnit.test("Test Three", function( assert ) {
10.  //
11. });
12. QUnit.test("Test Four", function( assert ) {
13.  //
14. });

```

ဒီနည်းနဲ့ Test One နဲ့ Test Two တို့ကို Module A အတွင်းမှာစုစည်းပေးသွားမှာဖြစ်ပြီး Test Three နဲ့ Test Four တို့ကိုတော့ Module B အတွင်းမှာ စုစည်းပေးသွားမှာဖြစ်ပါတယ်။ Module ကြော်လည်းတစ်ခါတည်းသတ်မှတ်နိုင်ပါတယ်။ setup မှာ သတ်မှတ်ထားတဲ့လုပ်ဆောင်ချက် တွေကို Test မစတင်မဲ့ QUnit က ဆောင်ရွက်ပေးသွားမှာဖြစ်ပါတယ်။ teardown မှာ သတ်မှတ်ထားတဲ့လုပ်ဆောင်ချက် တွေကို တော့ Test တွေအားလုံးပြီး သွားချိန်မှာ ဆောင်ရွက်ပေးသွားမှာဖြစ်ပါတယ်။ ဥပမာ - Database နဲ့ချိတ်ဆက်ပြီး Record တွေပယ်ဖျက်ပေးတဲ့ Function ကို စမ်းသပ်လိုတယ်ဆိုပါစို့။

## Pseudo-code

```

1. QUnit.module( "Module A", {
2.   setup: function() {
3.     records.add(98, dummy);
4.     records.add(99, dummy);
5.   },
6.   teardown: function() {
7.     records.reset();
8.   }
9. });
10.
11. QUnit.test("Database Test", function( assert ) {
12.   var total = records.count();
13.   assert.equal(record.remove([98, 99]).count(), total - 2);
14. });

```

ပေးထားတဲ့နူးမှာဖြစ်ပါတယ်။ setup မှာ Database ထဲကို Dummy Data တွေနဲ့ Record နှစ်ခုထည့်သွင်းလိုက်ပါတယ်။ ဒါကြောင့် Test လုပ်ချိန်မှာ တစ်ကယ့် Record ကိုဖျက်ပြီး စမ်းပို့မလိုတော့ပဲ အဲဒါ Dummy Record တွေကို Remove လုပ်ပြီး စမ်းကြည့်နိုင်သွားပါတယ်။ Database ထဲက အခြား Record တွေကို ဒါ Test ကြောင့် သက်ရောက်မှု မရှိစေ တော့ပါဘူး။ teardown မှာတော့ Auto-ID တွေကို Populate ပြန်လုပ်စေပါတယ်။ ဒါကြောင့် Test လုပ်လိုက်လို့ Auto-ID တွေ လွှဲသွားခဲ့ရင်လည်း ပြန်မှန်သွားစေမှာဖြစ်ပါတယ်။ ဒီနည်းနဲ့ Test Code ကြောင့် ဖြစ်ပေါ်လာနိုင်တဲ့ မလို လားအပ်တဲ့ Side-Effect တွေကို ရှောင်ရှားနိုင်မှာ ဖြစ်ပါတယ်။

QUnit ရဲ့ ကျွန်ုတ် လုပ်ဆောင်ချက်တွေကို [qunitjs.com](https://qunitjs.com) မှာ ဆက်လက်လေ့လာနိုင်ပါတယ်။

## 2.5 – Test Driven Development

TDD လိုခေါ်တဲ့ Test Driven Development ကတော့ Code ရေးသားနည် Concept တစ်ခုဖြစ်ပါတယ်။ လက်တွေ အလုပ်လုပ်မယ့် Code မရေးခင် Test Code ကို ဦးဆုံးရေးရတဲ့ Practice တစ်ခုဗျားပါ။ တစ်နည်းအားဖြင့် Test Code ကို Design အဖြစ်သော်လည်ကောင်း၊ Function Model အဖြစ်သော်လည်ကောင်း အသုံးချခြင်းလည်းဖြစ်ပါတယ်။ ဥပမာ - တည်နေရာ စာရင်းကို ပေးလိုက်ရင်၊ အကွာအဝေးအလိုက်စီပြီး ပြန် ပေးတဲ့ ရလဒ်ကို လိုချင်တယ်ဆိုပါစို့။ အဲဒါ ရလဒ် ရဖို့အတွက် Code ကို အရင်မရေးပါဘူး။ Test Code ကိုအရင် အခုလိုရေးရမှာဖြစ်ပါတယ်။

### JavaScript

```
1. QUnit.test("Sort by distance test", function( assert ) {
2.   var list = {"A": 12, "B": 32, "C": 8};
3.   assert.deepEqual(sortByDistance(list), [
4.     ["C", 8], ["A", 12], ["B", 32]
5.   ]);
6. });
```

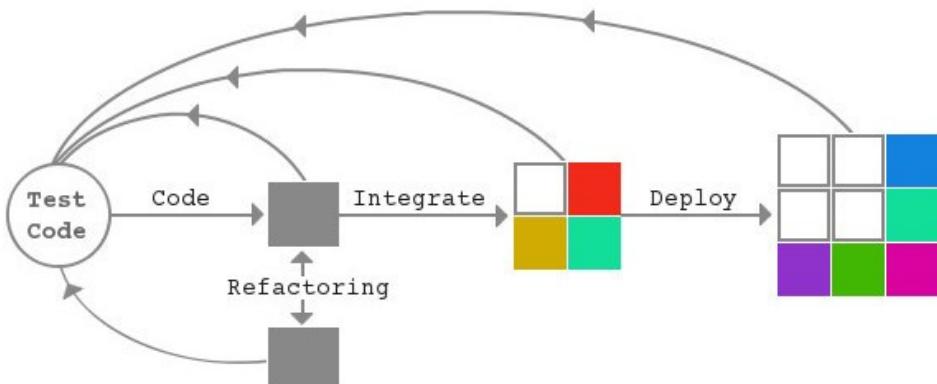
ဒါ Test ကို စမ်းကြည့်လိုက်ရင် Fail ဖြစ်မှာဖြစ်ပါတယ်။ sortByDistance() Function က ရေးတောင် မရေးရသေးတဲ့အတွက် ဖြစ်ပါတယ်။ တစ်ကယ့်လုပ်ဆောင်ချက် Code မရေးရသေးခင်ကတည်းက လိုချင်တဲ့ ရလဒ်ကို အရင်သတ် မှတ်လိုက်ခြင်းဖြစ်ပါတယ်။ Test Code ရပြီဆိုတော့မှ တစ်ကယ့်လုပ်ဆောင်ချက်အတွက် Code ကို ဆက်လက်ရေးသားရမှာ ဖြစ်ပါတယ်။

### JavaScript

```
1. function sortByDistance(list) {
2.   var result = [];
3.   for(x in list){
4.     result.push([x, list[x]])
5.   }
6.   result.sort(function(a,b){ return a[1] - b[1] });
7.   return result.reverse();
8. }
```

Function ရေးပြီးတဲ့အခါ Test Code ကို ပြန်စမ်းကြည့်ရမှာဖြစ်ပါတယ်။ Fail ဖြစ်နေသေးရင် Function ကို လိုအပ် သလိုပြင်းပြီး Test Code ကိုနောက်တစ်ကြိမ် ထပ်စမ်းရပါမယ်။ ဒီနည်းနဲ့ Test Pass မဖြစ်မချင်း Function ကို ပြုပြင်သွားရမှာဖြစ်ပါတယ်။ Test Pass ဖြစ်သွားပေမယ့် ပြုပြင်မှုတစ်ခုကြောင့် နောင်မှာ ဒီ Function က မျော်မှန်းသလို အလုပ်မလုပ်တော့တာမျိုး ဖြစ်တက်ပါတယ်။ ဥပမာ - တည်နေရာမှာ sort() Function ကိုပြင်လိုက်တဲ့အတွက် sortByDistance() ရဲ့ အလုပ်လုပ်ပုံ ပြောင်းသွားနိုင်ပါတယ်။ ဒါပေမယ့် ကြိုကာင်ရေးသားထားတဲ့ Test Code ရှိတဲ့ အတွက်ကြောင့် ဒီအမှားမျိုးရှိလာတဲ့အခါ ချက်ခြင်းသိရမှာဖြစ်ပါတယ်။ Function အားလုံးအတွက်သာ TDD Approach အတိုင်း Test Code တွေ ဦးစွာရေးသားထားခဲ့ရင် တစ်

နေရာမှာပြင်လိုက်လို့ နောက်တစ်နေရာမှာ သွားမှားနေတာကို မသိလိုက်ပဲကျန်သွားတယ်ဆိတာမျိုး မဖြစ်နိုင်တော့ပါဘူး။ မျှော်မှန်းရလဒ်လွှဲသွားတာနဲ့ Test Fail ဖြစ်တော့မှာပါ။ ဒီနည်းနဲ့ Code Quality လည်း တိုးတက်လာမှာပဲဖြစ်ပါတယ်။



ပုံ (၂.၃) - TDD Workflow

ပုံ (၂.၇) ကိုလေ့လာကြည့်ပါ။ အရင်ဆုံး မျှော်မှန်းရလဒ်ပါဝင်တဲ့ Test Code ကိုရေးရပါတယ်။ Function ရေးပြီးတဲ့ အခါ Test Code ကို Run ပြီး မျှော်မှန်းရလဒ်နဲ့ လက်တွေ့ရလဒ် ကိုက်ညီမှုရှိမရှိ စစ်ရပါတယ်။ Refactoring လုပ်ခဲ့ရင်လည်း Test Code ကို ပြန် Run ခြင်းအားဖြင့် မူလမျှော်မှန်းရလဒ်အတိုင်း ဆက်လက်ရရှိနေကြောင်း သေချာအောင်စစ် ရပါတယ်။ အဲဒီ Function ကို တစ်ခြား Function တွေနဲ့ ပေါင်းစပ်ပြီးတဲ့ အခါ Test Code ကို ထပ် Run ရပါတယ်။ အားလုံးအဆင်ပြေလို့ ဒီ Function Unit ပါဝင်တဲ့ Module ကို Deploy လုပ်လိုက်တဲ့ အခါမှာလည်း Test Code ကို ထပ် Run ပေးရပါတယ်။ ဒီနည်းနဲ့ ပြင်ဆင်မှုတွေ ဘယ်လိုပဲလုပ်လုပ် Function ရဲ့ အလုပ်လုပ်ပုံဟာ မူလမျှော်မှန်းထားတဲ့ အတိုင်း အလုပ်လုပ်နေတယ်ဆိတာ သေချာစေမှာဖြစ်ပါတယ်။ ဒီလို Test Code ကို အခြေခံရေးသားရတဲ့ နည်းစနစ် ဖြစ်တဲ့ အတွက် Test Driven Development လို့ ခေါ်ခြင်းဖြစ်ပါတယ်။ Test First Approach လို့လည်း ခေါ်ပါတယ်။

ဒီလို Test Code ကိုအခြေခံရေးသားခြင်းအားဖြင့် ရရှိလာတဲ့ အားသာချက်က အမှားနည်းယုံတင်မကပါဘူး။ Code ရဲ့ ဖွဲ့စည်းပုံနဲ့ Maintainability ပါ တိုးတက်လာပါတယ်။ ဘာဖြစ်လိုလဲဆိုတော့ Function တစ်ခုရေးတော့ မယ်ဆိုရင် အလုပ် လုပ်ပြီးရော ရေးလို့မရတော့ပါဘူး။ Test Code Run လို့ရအောင် သေချာစဉ်းစားပြီးမှ ရေးရပါတော့မယ်။ ပြီးတော့ အခြား Function တွေကို အလွန်းအမင်းမှုခိုလိုလည်း မရတော့ပါဘူး။ လုပ်စရာရှိတဲ့ အလုပ်ကို တစ်ခုတည်း သီးခြားအလုပ် လုပ်နိုင်မှာသာ Test လုပ်ရတာ ထိရောက်မှာဖြစ်ပါတယ်။ Function တစ်ခုက လုပ်ဆောင်ချက်တွေ အများအပြားကို လုပ်နေရင်လည်း အဆင်မပြေတော့ပါဘူး။ Function တစ်ခု လုပ်ဆောင်ချက်တစ်ခု ဖြစ်နေမှ Test လုပ်ရတာထိရောက်မှာပါ။ ဒါအပြင် Refactoring လုပ်ဖို့လည်း ကြောက်စရာမလိုတော့ပါဘူး။ တစ်ခုပြင်လိုက်လို့ သက်ရောက်မှုတွေ အခြားနေရာ တွေမှာ ရှုတယ်ဆိုရင်လည်း၊ ချက်ခြင်းသိရမှာမို့ ဒီအကိုး ဆက်အမှားတွေ Code ထဲမှာ ကျန်နေတာမျိုး မဖြစ်တော့ပါဘူး။ များများ Refactoring လုပ်နိုင်လေ့ Code တွေကလည်း ပိုပြီးရင်းလင်း နားလည်ရလွယ်ကူလေဖြစ်မှာပါ။

ပြောရမယ်ဆိုရင် TDD Approach ဟာ ကျွန်တော်တို့ Developer တွေ၊ ကိုယ်ရေးထားတဲ့ Code အပေါ် ယုံကြည် မှု ပိုမို တိုးတက်လာအောင် အကူအညီပေးမှာဖြစ်ပါတယ်။ တစ်နည်းအားဖြင့် Project ပြခါနီးတိုင်း ဒူးတုန်စရာ မလိုတော့အောင် အကူအညီပေးမှာပဲဖြစ်ပါတယ်။

## Conclusion

Ruby on Rails လို့ခေါ်တဲ့ အလွန်ထင်ရှားတဲ့ Web Application Framework ကို တိတွင်ခဲ့သူ David Heinemeier Hansson က (၂၀၁၄) ခုနှစ် ခြောက်တဲ့မှာ "TDD is dead. Long live testing." ဆိုတဲ့ ခေါင်းစဉ်နဲ့ ဆောင်းပါးတစ်ပုဒ် ရေးခဲ့တဲ့အတွက် Developer တွေအကြား အတော်လေးလှုပ်ခတ်သွားခဲ့ပါသေးတယ်။ အမှန်တော့ Ruby on Rails ဟာ TDD ကို အားပေးတဲ့ Framework တစ်ခုဖြစ်ပါတယ်။ ဒါပေမယ့် တစ်ချို့လူတွေက TDD ကို ကိုးကွယ်ရာတစ်ခုလောက် နှီးနှီး သဘောထားပြီး ဒီနည်းမှုမသုံးရင် Developer ကောင်းမဟုတ်တော့သလို ပြောကြတာ ကို အတော်မြင်ပြင်းကတ်နေပုံ ရပါတယ်။ DHH တို့လို အလွန်တော်တဲ့ Programmer တွေကတော့ ပြဿနာမရှိပါဘူး။ Unit Test တွေ မပါပဲနဲ့လည်း အများနည်းပြီး Maintainable ဖြစ်တဲ့ Quality Code တွေကို ရေးနိုင်ကြပါတယ်။ လူတိုင်းတော့ ဒီလိုရေးနိုင်မှာမဟုတ် ပါဘူး။ ဒါကြောင့် ဒီလိုဝေဖန်ထောက်ပြ ချက်တစ်ချို့ရှိနေပေမယ့် Test Driven Approach ဟာ Quality Code တွေ ရေးဖို့ များစွာအထောက်အကူပြုတယ်ဆိုတဲ့ အခြေခံအချက်ကို အသိအမှတ်ပြုရမှာပဲဖြစ်ပါတယ်။

တစ်ဘက်ကနားလည်ရမှာကတော့၊ TDD ကြောင့် Code အရည်အသွေးဟာ နှုန်းခြင်း အလိုအလျောက် တိုးတက် သွားမှာ မဟုတ်ပါဘူး။ အသုံးချုပ် Developer က ဘယ်လောက်ထိ ထိရောက်အောင်အသုံးချိန်းသလဲ ဆိုတဲ့ပေါ်မှာ အများကြီးမှတည်ပါသေးတယ်။ TDD စာအုပ်တွေ၊ Online Article တွေမှာ TDD Best Practice ဆိုပြီး သတိပြုလိုက်နာ သင့်တဲ့ အချက်တွေကိုဖော်ပြထားတာတွေ အများကြီးရှိပါတယ်။ ဆက်လက်လေ့လာပြီး သင့်တော်သလိုအသုံးချုပ်သင့် ပါတယ်။

Unit Test အပြင် အခြား Test Method တွေကိုလည်း TDD ပုံစံ ရေးသားမှုမှာ ထည့်သွင်းအသုံးပြုနိုင်ပါတယ်။ ဒါပေမယ့် ဒီ Test Method တွေကို မသုံးနိုင်ရင်တောင်မှ Unit Test သက်သက်နဲ့တင် အတော်ထိရောက်နိုင်ပါတယ်။ Unit Test Code တွေ ရေးရတာ လက်တွေမှာ မခက်ခဲပါဘူး။ ဒီလိုလွယ်ကူတဲ့ နည်းစနစ်တစ်ခုကြောင့် Code Quality ကို များစွာတိုးတက်စေမှာဖြစ်လို့ TDD ဟာ Rockstar Developer တွေ မဖြစ်မနေသိရှိအသုံးချေ မယ့် နည်းစနစ်တစ်ခုပဲ ဖြစ်ပါတယ်။

ပြုပြင်ထိမ်းသိမ်းရလွယ်တဲ့ Code တွေ ရေးသားနိုင်ဖို့အတွက်  
အများစံထားလိုက်နာလေ့ရှိတဲ့ Code Design Principle တွေကို  
သိရှိလိုက်နာဖို့ လိုအပ်ပါတယ်။

### **Rockstar Developer Course**

Project Management, Web Service, Server Architecture  
NodeJS အစရိတ်သည် ဤစာအုပ်ပါ အကြောင်းအရာများကို  
စာရေးသူကိုယ်တိုင် သင့်ကြားပေးခြင်းဖြစ်သည်။  
ဆက်သွယ်ရန် - (၀၉) ၇၃၁ ၆၅၄ ၆၂

## အခန်း(၃) – SOLID Design Principles

ပြီးခဲ့တဲ့ အခန်းမှာ Test Driven Development ရဲ့ အကူအညီနဲ့ အများနည်းတဲ့ Quality Code တွေရေးသားနိုင်ပုံ ကို လေ့လာခဲ့ပါတယ်။ ဒီအခန်းမှာတော့ ပြုပြင်ထိစ်းသိမ်းရလွယ်တဲ့ Maintainable Code တွေရေးသားရာမှာ အထောက်အကူ ဖြစ်စေမယ့် နည်းစနစ်များဖြစ်တဲ့ SOLID Design Principle အကြောင်းကို ဖော်ပြသွားမှာပဲဖြစ်ပါတယ်။ SOLID ဆိုတာ ဟာ –

1. Single Responsibility Principle
2. Open/Closed Principle
3. Liskov Substitution Principle
4. Interface Segregation Principle
5. Dependency Inversion Principle

– တို့ရဲ့ အတိုကောက်ဖြစ်ပါတယ်။

Object-Oriented Programming နဲ့ အတူ တွဲဖက်အသုံးပြုလေ့ရှိတဲ့ Object-Oriented Design Principle တွေဖြစ်ပါတယ်။

### 3.1 – Object-Oriented Programming

Object-Oriented Programming ဟာ Syntax အရေခက်ခဲလှတဲ့ နည်းစနစ်မဟုတ်ပေမယ့် Concept အရ နက်နတဲ့ ဘာသာရပ်ဖြစ်ပါတယ်။ ပါဝင်တဲ့ လုပ်ဆောင်ချက်တွေ များပြားပြီး ရှုပ်ထွေးကျယ်ပြန့်တဲ့ Software Product တွေတည် ဆောက်ရာမှာ Extensible ဖြစ်အောင်နဲ့ ပိုပြီးတော့ Maintainable ဖြစ်အောင် ဖွဲ့စည်းရေးသားနိုင်ဖို့ဆိုတဲ့ ရည်ရွယ်ချက်နဲ့ ပေါ်ပေါက်လာခြင်း ဖြစ်ပါတယ်။ ပိုပြီးအခြေခံကျတဲ့ Structured Programming (သို့မဟုတ်) Imperative Programming တို့မှာ ပရီဂရမ်တစ်ခုဆိုတာဟာ ညွှန်ကြားချက်တွေနဲ့ အတူ တွဲဖက်အသုံးပြုရမယ့် အချက်အလက် Data တွေကို စစည်း ရေးသားထားတဲ့ Instruction Set ကြီးတစ်ခုသာဖြစ်ပါတယ်။ Object-oriented Programming မှာတော့ ညွှန်ကြားချက် နဲ့ Data တွေကို ပြန်လည်အသုံးပြုနိုင်တဲ့ Instruction Set ငယ်များ (သို့မဟုတ်) Object များအဖြစ် ခွဲခြားစစည်းပြီး၊ အဲဒီ Object များ အပြန်အလှန် ဆက်သွယ်စေခြင်း တွဲဖက်လုပ်ငန်းဆောင်ရွက်စေခြင်းအားဖြင့် ပရီဂရမ်ကို တည်ဆောက် ရပါတယ်။

Object-oriented Programming ရဲ့ အခြေခံသဘောတရား (၅) ခု ရှိပါတယ်။ အဲဒါတွေကတော့ -

1. Object
2. Information Hiding
3. Inheritance
4. Interface
5. Polymorphism – တိုးပြန်ပါတယ်။

## Object

Object ဆိတာဟာ အချက်အလက် Data တွေနဲ့ လုပ်ဆောင်ချက် Function တွေပေါင်းစပ်ပါဝင်တဲ့ အစိတ် အပိုင်းတစ်ခု ဖြစ်ပါတယ်။ ဒါ Object တွေကို လက်တွေ့ကဗ္ဗာတဲ့ သက်ရှိသက်မဲ့ အရာဝါယူတွေနဲ့ ခိုင်းနှင့် ဖော်ပြလေ့ ရှိကြပါတယ်။ ဥပမာ - စက်ရိုင်းတစ်ခုဟာ Object တစ်ခုဖြစ်ပါတယ်။ သူမှာ အချင်း၊ အချင်းဝက်ဆိုတဲ့ အချက်အလက်တွေ ပါဝင်ပါတယ်။ နှုက်တစ်ကောင်ဟာလည်း Object တစ်ခုဖြစ်ပါတယ်။ သူမှာလဲ တောင်ပဲနှစ်ခု၊ ခြေထောက်နှစ်ချောင်း စတဲ့ အချက် အလက်တွေ ပါဝင်ပါတယ်။ ဒါအပြင် ပုံသန်းနှင့်ခြင်း၊ အသံပြုနိုင်ခြင်းတို့လို Function တွေလည်း ပါဝင်နိုင်ပါတယ်။ စက်ရိုင်းမှာ ရှိုးရှိုးစက်ရိုင်းနဲ့ ဘဲ့ခု့ပုံစက်ရိုင်းဆိုပြီး အခြေခံတည်ပေါ်ပေါ် အသေးစိတ်ကဲ့ပြားတဲ့ မျိုးကဲ့တွေ ရှိနိုင်ပါတယ်။ နှုက်မှာလည်း ပုံနှင့်တဲ့ မျို့နှင့်တဲ့ နှုက်ဆိုပြီး အခြေခံ Data နဲ့ Function တူပေါ်ပေါ် အသေးစိတ်ကဲ့ပြားတဲ့ မျိုးကဲ့တွေ ရှိနိုင်ပါတယ်။ ဒါကို Object Type နဲ့ Subtype လိုခေါ်ပြီး Object Type နဲ့ Subtype တွေ ဆက်စပ်ပဲကိုတော့ Object Hierarchy လို ခေါပါတယ်။

Object နဲ့ Object Hierarchy တွေ တည်ဆောက် ရရှိနိုင်ဖို့ အတွက် Programming Language အတော်များက Class တွေကို အသုံးပြုကြပါတယ်။ Class ဆိတာဟာ Object တစ်ခုတည်ဆောက်လိုက်တဲ့ အခါ အဲဒီ Object မှာ ပါဝင်ရမယ့် Data နဲ့ Function တို့ကို သတ်မှတ်ထားတဲ့ Specification တစ်ခုပဲဖြစ်ပါတယ်။

ဒီအခန်းမှာဖော်ပြတဲ့ Code နဲ့မှနာအများစုံဟာ ဆိုလိုရင်းသဘောသဘာဝကို ပေါ်လွှင်အောင် ဥပမာပေး တဲ့ Pseudo-Code တွေဖြစ်ပါတယ်။ လက်တွေ့ကူးယူစမ်းသပ်ဖို့မဟုတ်ပဲ ဆိုလိုရင်းသဘော သဘာဝကို နားလည်အောင် လေ့လာဖို့ဖြစ်ပါတယ်။

ဥပမာ - Class တစ်ခုက အခါလို သတ်မှတ်ထားတယ် ဆိုကြပါစ္ -

#### Pseudo-code

```

1. class Square
2. {
3.     var width = 5;
4.     var height = 5;
5.
6.     function setSize(size) {
7.         width = size;
8.         height = size;
9.     }
10.
11.    function getArea() {
12.        return width * height;
13.    }
14. }
```

ဒါ Class ကိုအသုံးပြုပြီး Object တစ်ခုတည်ဆောက်လိုက်တိုင်း၊ အဲဒီ Object မှာ width နဲ့ height ဆိုတဲ့ Data တန်ဖိုးနှစ်ခု ပါဝင်သွားမှာဖြစ်ပြီး `setSize()` နဲ့ `getArea()` ဆိုတဲ့ Function နှစ်ခုလည်း ပါဝင်သွားမှာဖြစ်ပါ တယ်။ Object တစ်ခုမှာ ပါဝင်တဲ့ Data ကို Object Property လို ခေါ်ပြီး Function ကိုတော့ Object Method လိုလည်းခေါ်ကြပါတယ်။

Object တွေ တည်ဆောက်ဖို့အတွက် Language အားလုံးနီးပါးက `new` Keyword ကို အသုံးပြုလေ့ ရှိကြပါ တယ်။ ပြီးခဲ့တဲ့ နမူနာမှာပေးထားတဲ့ Class ကိုအသုံးပြုပြီး Object တွေကို အခါလိုတည်ဆောက်နိုင်မှာဖြစ်ပါ တယ်။

#### Pseudo-code

```

1. var block = new Square();
2. var box = new Square();
```

ဒီတော့ `block` နဲ့ `box` ဆိုတဲ့ Square Object နှစ်ခုရရှိသွားပါတယ်။ Object နှစ်ခုလုံးမှာ ကိုယ်ပိုင် `width`, `height` တန်ဖိုးတွေ ရှိကြပါတယ်။ `block` Object ရဲ့ `width`, `height` တန်ဖိုးကို ပြောင်းလိုက်ခြင်းဟာ `box` Object ရဲ့ `width`, `height` တန်ဖိုးတွေကို သက်ရောက်စေမှာမဟုတ်ပါဘူး။ Class တစ်ခုတည်းကနေ ဖြစ်တည် လာတဲ့ Object နှစ်ခုဆိုပေမယ့် သီးခြားစီ အလုပ်လုပ်တဲ့သော ဖြစ်ပါတယ်။

Class တွေ Object တွေ အကြောင်းပြောပြီဆိုရင် အရေးပါလာတာကတော့ Constructor ပဲဖြစ်ပါတယ်။ Constructor က အခြေခံအားဖြင့် Object တည်ဆောက်စဉ်မှာ အလုပ်လုပ်ပေးတဲ့ Function တစ်မျိုးဖြစ်ပါ တယ်။ ပြီးခဲ့တဲ့ နမူနာ Class ကို အခါလို ပြင် ဆင်နိုင်ပါတယ်။

**Pseudo-code**

```

1. class Square
2. {
3.     var width;
4.     var height;
5.
6.     function Square(size) {
7.         width = size;
8.         height = size;
9.     }
10.
11.    function setSize(size) {
12.        width = size;
13.        height = size;
14.    }
15.
16.    function getArea() {
17.        return width * height;
18.    }
19. }

```

Language အများစုံမှာ Class Name နဲ့ အမည်တူတဲ့ Function တွေကို Constructor အဖြစ် အသုံးပြုလေ့ရှိကြပါတယ်။ နူးနာလိုင်းနံပါတ် (၆) မှာ ရေးသားထားတဲ့ `Square()` Function ဟာ ဒါ Class ကနေ Object တည်ဆောက်တိုင်းမှာ အလိုအလျောက် အလုပ်လုပ်သွားမယ့် Constructor Function ဖြစ်ပါတယ်။

**Pseudo-code**

```

1. var block = new Square(5);
2. var box = new Square(10);

```

နူးနာအရ `block` Object ရဲ့ `width`, `height` တန်ဖိုးဟာ 5 ဖြစ်မှာဖြစ်ပြီး `box` Object ရဲ့ `width`, `height` တန်ဖိုးကတော့ 10 ဖြစ်မှာပဲဖြစ်ပါတယ်။ Class တစ်ခုတည်းကနေ ဖန်တီးယူခြင်းဖြစ်ပေမယ့် Constructor အတွက် ပေးလိုက်တဲ့ Parameter တန်ဖိုးပေါ်မှုတည်ပြီး Object Property တွေရဲ့ Initial Value က လည်း ကျွဲ့ပြားသွားမှာပဲ ဖြစ်ပါတယ်။ ဒီနည်းနဲ့ ကိုယ်ပိုင် Data (Property) နဲ့ ကိုယ်ပိုင် Function (Method) တွေ ကိုယ်စိုင်တဲ့ Object တွေ တည်ဆောက်ပြီး အဲဒီ Object တွေ ဆက်သွယ်ပူးပေါင်းခြင်းအားဖြင့် ပရိုဂရမ်ကို အလုပ်လုပ်စေခြင်းဟာ Object-oriented Programming ရဲ့ အခြေခံအကျခုံး သဘောသဘာဝပဲ ဖြစ်ပါတယ်။

**Information Hiding**

Object တစ်ခုမှာ ကိုယ်ပိုင် Data နဲ့ Function တွေရှိတယ်ဆိုပေမယ့် အဲဒီ Data တွေ Function တွေအားလုံးကို အခြား Object တွေက သိရှိဖို့ ရယူအသုံးပြုဖို့ မလိုအပ်ပါဘူး။ ဥပမာ - ကားဆိုတဲ့ ပရိုဂရမ်တစ်ခု ဖြစ်ပေါ်လာဖို့ အင်ဂျင်၊ ဂိုယာ၊ ဘရိတ်၊ စတီယာရင် စသဖြင့် သူနေရာနဲ့သူ အရေးပါတဲ့ Object တွေ ပေါင်းစပ်ခြင်းအားဖြင့် တည်ဆောက်ထားတယ် ဆိုပါမြို့။ အင်ဂျင်၊ ဂိုယာ စတဲ့ Object တွေဟာ အပြန်အလုန်ဆက်သွယ်ပြီး ပူးပေါင်း ဆောင်ရွက်ဖို့လိုပေမယ့်၊ အင်ဂျင်ရဲ့ အတွင်းပိုင်းအလုပ်လုပ်ပုံကို ဂိုယာကသိစရာမလိုပါဘူး။ အင်ဂျင်လည်ပါတ်နဲ့ အနေးအမြန်ကိုဂိုယာက သိဖို့လိုနိုင်ပေမယ့်၊ အဲဒီ အင်ဂျင်လည်ဖို့အတွက် လောင်စာ `Inlet Valve` နဲ့ `Outlet`

Valve ဖွင့်/ပိတ် ပုံတိုင်မင်တွေ ဘယ်လိုအလုပ်လုပ်သလဲ၊ Piston တွေက လောင်စာပေါက်ကဲအားကို ဘယ်လိုတုံ့ပြန်ကသလဲစတဲ့ ရှုပ်ထွေးတဲ့ အသေးစိတ်အချက်အလက်တွေကို တော့ သိဖို့မလိုပါဘူး။ ဒီသဘောအတိုင်းပါပဲ၊ Object တွေမှာလည်း အခြား Object တွေကနေ ရယူအသုံးပြုခွင့် မပေးသင့်တဲ့ Data နဲ့ Function တွေ ရှိတက်ပါတယ်။ အဲဒီလို Data နဲ့ Function တွေကို အခြား Object တွေက ရယူခြင်း၊ ပြင်ဆင်ခြင်း မပြုနိုင်အောင် ဆောင်ရွက်ခြင်းကို Information Hiding လို့ ခေါ်ခြင်းပဲ ဖြစ်ပါတယ်။ Information hiding ကို Encapsulation လိုလည်း ခေါ်ကြပါတယ်။

Information Hiding ကို စနစ်တစ်ကျ အသုံးချမယ်ဆိုရင် လွှပ်လပ်ပြီး သီးခြားစီအလုပ်လုပ်တဲ့ Object တွေ ရရှိနိုင်မှာ ဖြစ်လို ပိုပြီးပြုပြင်ထိမ်းသိမ်းရလွယ်ကူတဲ့ ပရိုဂရမ်တွေကို ရရှိမှာပဲဖြစ်ပါတယ်။ ဥပမာ - အင်ဂျင်ပျက်ရင် အင်ဂျင်ကိုပဲ ပြင်စရာလိုမှာ ဖြစ်ပြီး ဂိုလာကို ထိစရာမလိုတော့ပါဘူး။ ဒါအပြင် ပိုပြီး Flexible ဖြစ်တဲ့ ပရိုဂရမ်တွေကိုလည်း တည်ဆောက်နိုင်မှာဖြစ်ပါတယ်။ စောစောကွေပမာဏကို ပြန်ကြည့်မယ်ဆိုရင် ကားထုတ်လုပ်သူတွေက လိုအပ်လာရင် အင်ဂျင်အမျိုးအစားတစ်မျိုးတည်းမှာပဲ ကားတစ်စီးအတွက် Auto ဂိုလာနဲ့တပ်ဆင်ပြီး နောက်ကားတစ်စီးအတွက် Manual ဂိုလာနဲ့ တပ်ဆင်နိုင်မှာဖြစ်ပါတယ်။ ဂိုလာက ဂိုလာအလုပ်ပဲလုပ်ပြီး၊ အင်ဂျင်က အင်ဂျင်အလုပ်ကို သီးခြားလုပ်တဲ့အတွက် အင်ဂျင် တစ်မျိုးတည်းမှာ ဂိုလာစနစ်အမျိုးမျိုးနဲ့ လိုအပ်ရင် တွေဖက်တပ်ဆင်နိုင်ခြင်းပဲ ဖြစ်ပါတယ်။

Information Hiding လုပ်ဆောင်ချက် ရရှိဖို့အတွက် Language အများစုံက Public, Private, Protected, Static စတဲ့ Access Control Modifier တွေကို အသုံးပြုကြလေ့ရှိပါတယ်။ Language ပေါ်မှုတည်ပြီး ဒီ Access Control တွေရဲ့ အသေးစိတ် အလုပ်လုပ်ပုံကဲ့ပြားနိုင်ပါတယ်။

### Pseudo-code

```

1. class Engine
2. {
3.     public var kilo;
4.     private var pistons = 4;
5.     private var valves = 12;
6.
7.     public function dirve() {
8.         kilo++;
9.     }
10.
11.    public function check() {
12.        if( dirtyOil() )
13.            print "Please change engine oil!";
14.    }
15.
16.    private function dirtyOil() {
17.        if(kilo > 3000) return true;
18.        return false;
19.    }
20. }
```

နမူနာ Engine Class မှာ kilo, pistons နဲ့ valves ဆိုပြီး Data Property သုံးခုပါဝင်ပါတယ်။ အဲဒီထဲက pistons နဲ့ valves တိုကို private လို့ သတ်မှတ်ထားတဲ့အတွက်၊ ဒီတန်ဖိုးနှစ်ခုဟာ ပြင်ပကနေ ရယူ

အသုံးပြု လိုမရတဲ့ တန်ဖိုးတွေဖြစ်သွားမှာပါ။ အလားတူပဲ Function သုံးခုပါဝင်ရာမှာ dirtyOil() Function ကို private လို့ ကြော်လုပ်စေမည့်အတွက် ဒီ Function ကိုလည်း ပြင်ပကနေ ရယူအသုံးပြနိုင်မှာ မဟုတ်ပါဘူး။ private Data နဲ့ Function တွေကို Class အတွင်းမှာတော့ရယူအသုံးပြနိုင်ပါတယ်။ နမူနာမှာ ဆိုရင် check() Function ကို ခေါ်ယူအသုံးပြထားပါတယ်။

Engine Class ကနေ Object တစ်ခုကို အခုလို တည်ဆောက်ရယူထားတယ် ဆိုပါစို့ –

#### Pseudo-code

```
1. var v6 = new Engine();
2. v6.drive();
3. v6.check();
```

တည်ဆောက်ရရှိထားတဲ့ v6 Object ကနေ public Function များဖြစ်ကြတဲ့ drive() သို့မဟုတ် check() Function တိုကို ခေါ်ယူအလုပ်လုပ်စေနိုင်ပေမယ့် dirtyOil() Function ကိုတော့ private ဖြစ်နေတဲ့ အတွက် ခေါ်ယူအလုပ် လုပ်စေနိုင်မှာ မဟုတ်ပါဘူး။ ဒီနည်းနဲ့ Data တွေ Function တွေမှာ Access Control Modifier တွေ ကြော်လုပ်စေမရှိတဲ့ အားဖြင့် Information Hiding လုပ်ဆောင်ချက် ကိုရရှိနိုင်ပါတယ်။ Access Control သတ်မှတ် ထားခြင်းမရှိတဲ့ Data နဲ့ Function တွေအတွက် Default Modifier ကို အသုံးပြုကြလေ့ရှိပါတယ်။ ဥပမာ - PHP Programming Language မှာဆိုရင် Default Modifier ကို public ဖြစ်တဲ့ အတွက် Access Control သတ်မှတ် ထားခြင်းမရှိတဲ့ Data နဲ့ Function အားလုံးဟာ public ဖြစ်နေမှာပဲဖြစ်ပါတယ်။

အခြား protected, package public စသဖြင့် အသုံးပြနိုင်တဲ့ Access Control Modifier တွေ Language အလိုက် ရှိတက်ကြပါသေးတယ်။ ဒီထဲက ထူးခြားတာကတော့ static Access Control Modifier ပဲဖြစ်ပါတယ်။ Function သို့မဟုတ် Data တန်ဖိုးတစ်ခုခုကို static လို့ ကြော်လုပ်စေမှတ်ထားမယ်ဆိုရင် အဲ ဒီ Function သို့မဟုတ် Data ကို Object တည်ဆောက်စရာမလိုပဲ တိုက်ရှိတော့ရယူအသုံးပြနိုင်ပါတယ်။ အထူးသဖြင့် Object တစ်ခုချင်းစီရဲ့ သီးခြားကိုယ်ပိုင် တန်ဖိုးဖြစ်ဖို့မလိုတဲ့ ဘုံတန်ဖိုးတွေကို static အဖြစ် ကြော်လုပ်စေမှတ်ကြလေ့ရှိပါတယ်။ ဥပမာ -

#### Pseudo-code

```
1. class Math
2. {
3.     static var PI = 3.1416;
4.     static function round(value) {
5.         return parseInt(value);
6.     }
7. }
```

ဒီ Math Class မှာပါဝင်တဲ့ PI နဲ့ round() တို့ကို static လို့ ကြော်လုပ်စေမှတ်ထားတဲ့အတွက် Object တည်ဆောက်စရာမလိုပဲ အခုလို တိုက်ရှိတော့ရယူအသုံးပြနိုင်မှာဖြစ်ပါတယ်။

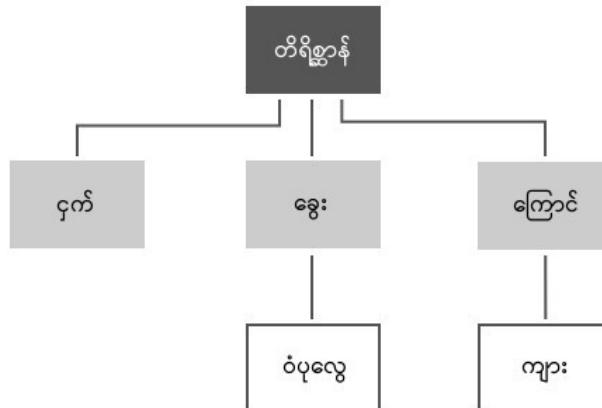
### Pseudo-code

```
1. print Math.PI;
2. print Math.round(4.2);
```

PI, round() စတဲ့ Data နဲ့ Function တွေဟာ Object ပြောင်းတိုင်း လိုက်ပြောင်းစရာမလိုတဲ့ ပုံသေတန်ဖိုးနဲ့ လုပ်ဆောင်ချက်တွေ ဖြစ်တဲ့အတွက် ဒီလိုအခြေအနေမှာ static Access Control Modifier ကို အသုံးပြုကြ လောက့်ခြင်း ဖြစ်ပါတယ်။

### Inheritance

Inheritance ဆိတ်တာကတော့ Object သို့မဟုတ် Class တစ်ခုက အခြား Object သို့မဟုတ် Class တစ်ခုရဲ့ Data နဲ့ Function တွေကို ဆက်ခံရယူနိုင်စေတဲ့ လုပ်ဆောင်ချက်ပဲ ဖြစ်ပါတယ်။ Inheritance ရဲ့ အကူအညီနဲ့ အချို့လုပ်ဆောင်ချက်တွေကို ထပ်ခါထပ်ခါ ပြန်လုပ်စရာမလိုပဲ ပြန်လည်အသုံးချိနိုင်စေသလို၊ သဘောသဘာဝ ဆင်တူနဲ့ Object တွေကို အပြန်အလှန်ဆက်စပ် စေခြင်းအားဖြင့် စနစ်ကျပြီး ပြင်ဆင်ထိမ်းသိမ်းရ လွယ်ကူတဲ့ Object Hierarchy ကို တည်ဆောက်ထားနိုင်မှာပဲဖြစ်ပါတယ်။



ပုံ (၃.၁) - Object Hierarchy

နဲ့မူနာပုံမှာလေ့လာကြည့်မယ်ဆိုရင် ငုံ၊ ခွေး၊ ကြောင်း စတဲ့ Object များက တိရိစ္ဆာန် Object ကနေဆက်ခံပြီး Inherit လုပ်ထားတဲ့အတွက် ငုံ၊ ခွေး၊ ကြောင်း အားလုံးတို့မှာ တိရိစ္ဆာန် Object မှာရှိတဲ့ Data နဲ့ Function တွေ အားလုံး ရှိ ပါဝင်သွားမှာပဲ ဖြစ်ပါတယ်။ အလားတူပဲ၊ ဝံပါဇွဲ Object ဟာ ခွေး Object ကို ဆက်လက်ဆက်ခံထားတဲ့အတွက် တိရိစ္ဆာန် Object နဲ့ ခွေး Object တို့မှာ ပါဝင်တဲ့ Data နဲ့ Function တွေအားလုံးကို ရရှိပါဝင်သွားမှာဖြစ်ပါတယ်။

ဆက်ခံတဲ့ Child Object တွေမှာ မူလ Parent Object မှာ မပါဝင်တဲ့ Data နဲ့ Function တွေ ဖြည့်စွက်ပါဝင်နိုင်သလို မူလ Parent Object ရဲ့ Data နဲ့ Function တွေကို လိုအပ်သလို ပြောင်းလဲသတ်မှတ်ပြီးတော့လည်း ထားနိုင်ပါတယ်။

## Pseudo-code

```

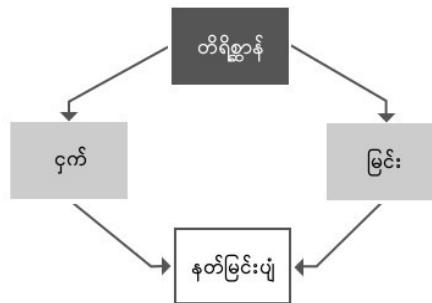
1. class Animal
2. {
3.     var legs = 4;
4.
5.     function eat() {
6.         ...
7.     }
8. }
9.
10. class Bird extends Animal
11. {
12.     var legs = 2;
13.     var wings = true;
14. }
15.
16. class Dog extends Animal
17. {
18.     function bark() {
19.         ...
20.     }
21. }

```

နမူနာမှာလေးလာကြည့်ရင် Animal Class မှာ legs ဆိုတဲ့ Data နဲ့ eat() ဆိုတဲ့ Function တို့ပါဝင်ပါတယ်။ Animal Class ကို Bird Class နဲ့ Dog Class တို့က extends Keyword ကိုသုံးပြီး ဆက်ခံ Inherit လုပ်ထားကြပါတယ်။ ဒါကြောင့် Bird နဲ့ Dog Class တို့ဟာ Animal Class ရဲ့ legs နဲ့ eat() ကို အလိုအလျောက် ဆက်ခံရရှိသွားပါတယ်။ ဒါပေမယ့် Bird Class မှာ legs ရဲ့ တန်ဖိုးကို ပြောင်းထားပြီး wings ဆိုတဲ့ တန်ဖိုး အသစ် တစ်ခုလည်း ဖြည့်စွက် ပါဝင်လာပါတယ်။ ပြီးတော့ Dog Class မှာဆိုရင်လည်း bark() ဆိုတဲ့ Function တစ်ခု ဖြည့်စွက် ပါဝင်လာပါတယ်။

ဒါကြောင့် အခုနေ Dog Object တစ်ခု တည်ဆောက်မယ်ဆိုရင် legs ဆိုတဲ့ Data နဲ့ eat() , bark() ဆိုတဲ့ Function တို့ ပါဝင်သွားမှာပဲ ဖြစ်ပါတယ်။

Inheritance မှာ Single Inheritance နဲ့ Multiple Inheritance ဆိုပြီးနှစ်မျိုးရှိပါတယ်။ Single Inheritance ဆို တာကတော့ Object သို့မဟုတ် Class တစ်ခုဟာ အခြား Object သို့မဟုတ် Class တစ်ခုကိုသာ ဆက်ခံရတဲ့ နည်းစနစ်ဖြစ်ပြီး Multiple Inheritance ဆိုတာကတော့ Class သို့မဟုတ် Object တစ်ခုဟာ အခြား တစ်ခုထက် ပိုသော Class များ၊ Object များကို ဆက်ခံရယူနိုင်တဲ့ နည်းစနစ်ဖြစ်ပါတယ်။ Java, Ruby စတဲ့ Language တွေ မှာ Multiple Inheritance ကို ခွင့်မပြုကြပဲ Single Inheritance ကိုသာ ခွင့်ပြုကြပါတယ်။ C++ နဲ့ Python တို့လို Language တွေကတော့ Multiple Inheritance ကို ခွင့်ပြုထားပါတယ်။ Language က ခွင့်ပြုသည်ဖြစ်စေ၊ ခွင့်မပြုသည်ဖြစ်စေ လက်တွေမှာ Multiple Inheritance ကို ရှောင်ကြလေ့ရှိပါတယ်။ "Deadly diamond of death" လိုခေါ်တဲ့ ပြဿနာ ကြောင့်ပဲဖြစ်ပါတယ်။



ပုံ (၃.J) - Deadly Diamond of Death

ပုံ (၃.J) မှာလေ့လာကြည့်ရင် ငုတ်နဲ့ မြင်းတို့ဟာ တိရိစ္ဆာန်ကို ဆက်ခံထားကြပါတယ်။ နတ်မြင်းပျုဆိုတဲ့ Object တစ်ခု ရရှိလိုတဲ့အတွက် ငုတ်နဲ့ မြင်း နှစ်ခုတို့ထံကနေ Multiple Inheritance အနေနဲ့ ဆက်ခံလိုက်တဲ့ အခါ အခုလို ပြဿနာ ပေါ်ပေါက်လာတက်ပါတယ်။

#### Pseudo-code

```

1. class Animal
2. {
3.     function giveBirth() {
4.     ...
5.   }
6. }
7.
8. class Bird extends Animal
9. {
10.    var legs = 2;
11.    var wings = true;
12.    function giveBirth() {
13.      layEggs();
14.    }
15. }
16.
17. class Horse extends Animal
18. {
19.    var legs = 4;
20.    function giveBirth() {
21.      birthLive();
22.    }
23. }
24.
25. class FlyingHorse extends Bird, Horse
26. {
27.   ...
28. }
29.
30. var pegasus = new FlyingHorse();
31. pegasus.giveBirth();
  
```

နမူနာမှာလေ့လာကြည့်ရင် Animal Class မှာ giveBirth() Function ပါဝင်ပြီး၊ Bird နဲ့ Horse Class

နှစ်ခု လုံးမှာလည်း giveBirth() Function ကို ပြပြင်ထားပါတယ်။ ငုက်တွေက ဗြိခိုင်းအားဖြင့်သားများ ပြီး၊ မြင်းတွေ ကတော့ အကောင်လိုက် မွေးကြလေ့ရှိတဲ့ အတွက် သဘောတရားမတူလို့ ပြပြင်ထားရခြင်းဖြစ်ပါတယ်။ FlyingHorse Class က Bird နဲ့ Horse Class နှစ်ခုလုံးတဲ့ ကနေ Inherit လုပ်ထားပါတယ်။ လိုင်းနဲ့ပါတ် (၃၁) မှာ လေ့လာကြည့်ပါ။ FlyingHorse Class ရဲ့ Object ဖြစ်တဲ့ pegasus ကနေ giveBirth() ကို ခေါ်ယူ အလုပ် လုပ်စေတဲ့ အခါ ပြသေနာကြိုရမှာဖြစ်ပါတယ်။ Bird နဲ့ Horse Class နှစ်ခုလုံးမှာ giveBirth() Function ရှိနေတဲ့ အတွက် Bird ရဲ့ giveBirth() ကို ဆက်ခံ အလုပ်လုပ်ပေးရမှာလား၊ Horse ရဲ့ giveBirth() ကို ဆက်ခံအလုပ်လုပ် ပေးရမှာလားဆိုတဲ့ ပြသေနာကို ကြော်လော်ခြင်းပဲဖြစ်ပါတယ်။ ဒီပြသေနာကို Deadly diamond of death လို့ ခေါ်တာပါ။

Multiple Inheritance ကို လက်ခံတဲ့ Language တွေမှာတော့ ဒီပြသေနာကို နည်းလမ်းအမျိုးမျိုးနဲ့ ဖြေရှင်းထားကြ ပါတယ်။ Perl, Python စတဲ့ Language တွေက extends လုပ်စဉ်မှာ ပေးခဲ့တဲ့ အစီအစဉ်ပေါ်မှတည်ပြီး "ဦးစားပေး" Class စာရင်းထား ရှိခြင်းအားဖြင့်၊ Deadly diamond ပြသေနာကြိုလော်တဲ့ အခါ စာရင်းထဲမှာ Priority ပိုမြင့်တဲ့ Class ရဲ့ Function ကို ဦးစားပေးအဖြစ် သတ်မှတ်အသုံးပြု ပေးသွားမှာပဲဖြစ်ပါတယ်။

တစ်ချို့ Language တွေမှာဆိုရင် Inherit လုပ်ခွင့်မပြုတဲ့ Syntax ရေးထုံးတွေရှိတက်ပါသေးတယ်။ ဥပမာ - Java မှာ Class တစ်ခုကို final လို့ ကြော်သတ်မှတ်ထားရင် အခြား Class များက Inherit လုပ်ယူခြင်းကို ခွင့်မပြုတဲ့ Class ဖြစ်သွားမှာပဲ ဖြစ်ပါတယ်။ ဒါအပြင် Inherit လုပ်ခွင့်ပြုပေမယ့် မူလ Parent Class/Object ရဲ့ Data နဲ့ Function ကို Child Class တွေထဲမှာ ပြပြင်ခြင်းကို ကန်သတ်ထားနိုင်ပါသေးတယ်။

## Interface

ပရိုဂရမ်တစ်ခုမှာ Text Box, Password Box နဲ့ Email Box ဆိုပြီး Input Box အမျိုးမျိုးပါဝင်တယ်ဆိုကြပါစိုး။ ဒီ Input Box တွေ တစ်ခုနဲ့တစ်ခု ဖော်ပြပုံတူကြသလို့ ပါဝင်တဲ့ လုပ်ဆောင်ချက်တွေ တူကြပါတယ်။ Size Property ကို ပြောင်းခြင်းအားဖြင့် Input Box ရဲ့ အရွယ်အစားကို ပြင်နိုင်ပါတယ်။ တန်ဖိုးတစ်ခုထည့်သွင်းလို ရင် Input Box ကို Click လုပ်ပြီး Keyboard ကနေ ထည့်သွင်းနိုင်ပါတယ်။ Input Box မှာ ထည့်သွင်းထားတဲ့ တန်ဖိုးကို လက်ခံရယူပြီး လိုအပ်တဲ့ လုပ်ငန်းတွေ ဆက်လက်ဆောင် ရွက်နိုင်ပါတယ်။ ဒါပေမယ့် အသေးစိတ် ကွာခြားချက်တွေတော့ ရှိပါတယ်။ Text Box ဆိုရင် ထည့်သွင်းလိုက်တဲ့ စာတွေကို ဖော်ပြပြီး၊ Password Box ဆိုရင်တော့ စာတွေအစား Star လေးတွေနဲ့ အစား ထိုးဖော်ပြနိုင်ပါတယ်။ ဒါပေမယ့် Text Box သုံးရမယ့် နေရာမှာ Password Box ကို အစားထိုး အသုံးပြနိုင်ပါတယ်။ Password Box သုံးရမယ့် နေရာမှာလည်း Text Box ကို အစားထိုး အသုံးပြနိုင်ပါတယ်။ ဘာဖြစ်လို့လဲဆိုတော့ ကြိုတင် သတ်မှတ်ထားတဲ့ Interface တစ်ခုတည်းကနေ ရရှိလာတဲ့ Object တွေဖို့ပဲဖြစ်ပါတယ်။ Text Box က ပြန်ပေးတဲ့ တန်ဖိုးနဲ့ Password Box က ပြန်ပေးတဲ့ တန်ဖိုး၊ သဘောသဘာဝ တူညီမှာဖြစ်တဲ့ အတွက် အခုလုံအသုံးပြု နိုင်ခြင်းပဲဖြစ် ပါတယ်။ ဒီတော့ Text Box နဲ့ တွဲဖက်အလုပ်လုပ်ရမယ့် Object ကတစ်မျိုး၊ Password Box နဲ့ တွဲဖက်အလုပ် လုပ်ရမယ့် Object က တစ်မျိုးဆိုပြီး ခွဲခြားရေးသားနေစရာ မလိုတော့ပဲ Object တစ်မျိုးတည်းနဲ့ပဲ Text Box နဲ့ရော့ Password Box နဲ့ပါ တွဲဖက်အလုပ်လုပ်စေနိုင်မှာဖြစ်လို့ Code Design လည်း အများကြီးတိုးတက်ကောင်းမွန်သွားမှာပဲ ဖြစ်ပါတယ်။

Interface ဆိုတာဟာ အဲဒီလိုမျိုး အသေးစိတ် လုပ်ဆောင်ချက်ကွာခြားပေမယ့် အပြန်အလှန် ဖလှယ်အသုံးပြု နိုင်တဲ့ Object တွေ တည်ဆောက်နိုင်ဖို့အတွက် ဘုံးစိုးတိုးတက်ပေးတဲ့ နည်းစနစ်တစ်မျိုးပဲ ဖြစ်ပါတယ်။ ဒါ

ကြောင့် Interface ကို Protocol (သိမဟုတ်) Contract လိုလည်းခေါ်ကြပါသေးတယ်။ အောက်ပါ Code နမူနာ ကိုလေ့လာကြည့်ပါ။

### Pseudo-code

```

1. interface InputBox
2. {
3.     function setValue(val);
4.     function showValue();
5.     function resize(width, height);
6. }
7.
8. class TextBox implement InputBox
9. {
10.    var value;
11.    function setValue(val) {
12.        value = val;
13.    }
14.    function showValue() {
15.        print value;
16.    }
17.    function resize(width, height) {
18.        ...
19.    }
20. }
21.
22. class PasswordBox implement InputBox
23. {
24.    var value;
25.    function setValue(val) {
26.        value = val;
27.    }
28.    function showValue() {
29.        for(var i=0; i < value.length; i++) {
30.            print "*";
31.        }
32.    }
33.    function resize(width, height) {
34.        ...
35.    }
36. }
37.
38.
39.
40.
41.
42.
43.
44.
45.
46. }
```

ထိုင်းနဲ့ပါတ် (၁) မှာ InputBox Interface ကို ကြော်လှုပ်သော်မှတ်ထားပါတယ်။ **setValue()**, **showValue()** နဲ့ **resize()** ဆိုတဲ့ Function သုံးခုကို ကြော်လှုပ်သော်မှတ်ထားပေမယ့် အဲဒီ Function တစ်ခု ချင်းရဲ့ လုပ်ဆောင်ချက် အသေးစိတ်ကိုတော့ သတ်မှတ်မထားပါဘူး။ Interface တစ်ခုရဲ့တဲ့ စံအတိုင်း ပါဝင်ရ မယ့် လုပ်ဆောင်ချက် Function စာရင်းနဲ့ Function Signature လိုပေါ်တဲ့ အဲဒီ Function တွေကိုတည်ဆောက် တဲ့အခါ ပါဝင်သင့်တဲ့ Parameter စာရင်းတို့ကိုသာ ကြော်လှုပ်သော်မှတ်ထားခြင်း ဖြစ်ပါတယ်။

ဒါ Interface ကို ဆက်ခံတည်ဆောက်ထားတဲ့ Object တိုင်းမှာ Interface က သတ်မှတ်ထားတဲ့ Function အားလုံး မဖြစ်မနေပါဝင်ရမှာဖြစ်ပြီး Function Signature ကလည်း Interface မှာ သတ်မှတ်ထားတဲ့အတိုင်း အတိအကျဖြစ်ရ မှာဖြစ်ပါတယ်။ ဒါကြောင့်လည်း Interface တစ်ခုတည်းက ဆင်းသက်လာတဲ့ Object များဟာ

တစ်ခုနဲ့တစ်ခု အပြန် အလုန် ဖလှယ်အသုံးပြနိုင်ခြင်းဖြစ်ပါတယ်။ နမူနာမှာ TextBox နဲ့ PasswordBox ဆိုတဲ့ Class နှစ်ခုက Input Box Interface ကို ဆက်ခံတည်ဆောက်ထားကြပါတယ်။ ဒါကြောင့် Class နှစ်ခုလုံးမှာ အသေးစိတ် အလုပ်လုပ်ပုံမတူ ပေမယ့် `setValue()`, `showValue()` နဲ့ `resize()` ဆိုတဲ့ ပါဝင်ရမယ့် Function အားလုံး ပါဝင်ပါတယ်။

ပြီးခဲ့တဲ့အခန်းမှာ Test Driven Development အကြောင်းဖော်ပြစ်ပေါ်က၊ Stub တွေကဲ့ အခန်းကလွှာကို ဖော်ပြခဲ့ပါတယ်။ Interface ရဲ့ အကူအညီနဲ့ တစ်ကယ့် Object အစား Mock Object တွေနဲ့ လိုအပ်သလို အစားထိုး စမ်းသပ်နိုင်တဲ့ အားသာချက်ကိုလည်း ရနိုင်ပါသေးတယ်။ တစ်ကယ့် Object ရော Mock Object ကပါ Interface တစ်ခုတည်းက ဆင်းသက်မယ်ဆိုရင် အပြန်အလုန် ဖလှယ်အသုံးပြနိုင်တဲ့ Object တွေဖြစ်မှာ မို့ တစ်ကယ့် Object အစား Mock Object နဲ့ အလွယ်တစ်ကူ အစားထိုး စမ်းသပ်နိုင်မှာပဲ ဖြစ်ပါတယ်။

## Polymorphism

Polymorphism ဆိုတာကတော့ Object တစ်ခုရဲ့ Function တွေကို တစ်မျိုးတည်းပုံသေအလုပ်လုပ်တာမျိုး မဟုတ်ပဲ ခေါ်ယူအသုံးပြုတဲ့အခြေအနေပေါ် မူတည်ပြီး ပြောင်းလဲအလုပ်လုပ်နိုင်အောင် ဖန်တီးတဲ့ နည်းစနစ် တစ်ခုဖြစ်ပါတယ်။ Static Polymorphism နဲ့ Dynamic Polymorphism ဆိုပြီး နှစ်မျိုးရှိပါတယ်။ Static Polymorphism လုပ်ဆောင် ချက်ရရှိဖို့အတွက် Function Overload နည်းစနစ် ကိုအသုံးပြုပြီး Dynamic Polymorphism လုပ်ဆောင်ချက် ရရှိဖို့ အတွက်တော့ ပြောခဲ့ပြီးဖြစ်တဲ့ Inheritance နဲ့ Interface တို့ကို သုံးနိုင်ပါတယ်။

### Pseudo-code

```

1. class Shape
2. {
3.     function area(r) {
4.         return 3.14 * r * r;
5.     }
6.
7.     function area(x, y) {
8.         return x * y;
9.     }
10.
11.    function area(x, y, z) {
12.        return x * y * z;
13.    }
14. }
```

နမူနာမှာလေ့လာကြည့်ရင် Shape Class မှာ Parameter စာရင်းမတူတဲ့ `area()` Function သုံးခပါဝင်တာကို တွေ့ရ မှာဖြစ်ပါတယ်။ ဒါကို Function Overload လိုခေါ်ပြီး၊ ခေါ်ယူတဲ့အခါ ပေးတဲ့ Parameter စာရင်း (Function Signature) ပေါ်မူတည်ပြီး အလုပ်လုပ်တဲ့ Function လည်းပြောင်းသွားမှာပဲ ဖြစ်ပါတယ်။ ဥပမာ -

## Pseudo-code

```

1. var item = new Shape();
2. item.area(2);           // => 12.56
3. item.area(2, 2);       // => 4
4. item.area(2, 2, 2);    // => 8

```

ဒီနည်းနဲ့ ခေါ်ယူတဲ့ Function Signature ပေါ်မှတည်ပြီး လုပ်ဆောင်ပုံပြောင်းလဲနိုင်ပေါ်ခြင်းကို Static Polymorphism လို ခေါ်ခြင်းဖြစ်ပါတယ်။ Dynamic Polymorphism ကိုတော့ Interface အကြောင်းဖော်ပြစဉ်က တွေ့မြင်ခဲ့ကြရပြီးဖြစ် ပါတယ်။

## Pseudo-code

```

1. interface Animal
2. {
3.     function move();
4. }
5.
6. class Dog implements Animal
7. {
8.     function move() {
9.         print "Dog walk and run";
10.    }
11. }
12.
13. class Bird implements Animal
14. {
15.     function move() {
16.         print "Bird fly";
17.    }
18. }

```

နဲ့မှန်ဘာ Animal Interface ကနေ Dog နဲ့ Bird ဆိုတဲ့ Class နှစ်ခု ဆက်ခံတည်ဆောက်ထားပါတယ်။ ပြီးတော့ Interface က သတ်မှတ်ထားတဲ့အတိုင်း move() Function ကိုလည်း Implement လုပ်ထားပါသေးတယ်။ Dog ရဲ့ move() Function နဲ့ Bird ရဲ့ move() Function တို့ အသေးစိတ် အလုပ်လုပ်ပုံ မတူကြပါဘူး။ ဒါကြောင့် Animal Interface က လာတဲ့ Object ချင်း တူပေါ်မယ့် move() Function ကို ခေါ်ယူတဲ့အခါမှာ Object ပေါ်မှတည်ပြီး move() Function ရဲ့အလုပ်လုပ်ပုံလည်း အခုလို ပြောင်းလဲသွားမှာပဲဖြစ်ပါတယ်။

## Pseudo-code

```

1. var milo = new Dog();
2. var jay = new Bird();
3.
4. milo.move();           // => Dog walk and run
5. jay.move();            // => Bird fly

```

ဒီလို Interface ကနေ Implement လုပ်ယူခြင်းအားဖြင့် သို့မဟုတ်၊ အခြား Class တစ်ခုကနေ Inheritance လုပ်ယူခြင်းအားဖြင့် အမျိုးအစားတူပေါ်မယ့် လုပ်ဆောင်ပုံမတူတဲ့ Function တွေ ရရှိနိုင်ခြင်းကို Dynamic

Polymorphism လိုခေါ်ပါ တယ်။

ဒါကြောင့် Object-oriented Programming ဟာ ကျယ်ပြန်ရှုပ်ထွေးတဲ့ ပရိုဂရမ်တွေကို Extensible ဖြစ်အောင် Maintainable ဖြစ်အောင် ကူညီပေးတယ်လိုအပိုပေမယ့် Class တွေ Object ကို အခြေခံလိုက်ယုံသက်သက်နဲ့ တော့ ပရိုဂရမ်က အလိုအလျောက် စနစ်ကျကောင်းမွန်သွားမှာ မဟုတ်ပါဘူး။ ဖော်ပြခဲ့တဲ့ Information Hiding, Inheritance, Interface နဲ့ Polymorphism တို့ ကို အသေအချာနားလည်ပြီး၊ သူနေရာနဲ့သူ ထိရောက်အောင် အသုံးချိန်မှသာ Extensible ဖြစ်ပြီး Maintainable ဖြစ်တဲ့ Object-oriented ပရိုဂရမ်တွေကို ရရှိမှာပဲဖြစ်ပါတယ်။

### 3.2 – OOP in JavaScript

JavaScript Programming Language ဟာ Object-oriented နည်းစနစ်များကို အခြေခံထားတဲ့ OOP Language တစ်ခုဖြစ်ပါတယ်။ ဒါပေမယ့် အခြား OOP Language တွေနဲ့မတူပါဘူး။ အမိကအကျခုံးထူးခြားချက်ကတော့ အခြား OOP Language တွေမှာ Object တွေ ရရှိဖို့အတွက် Class တွေကို အသုံးပြုကြပြီး၊ JavaScript မှာတော့ Class တွေ ကိုအသုံးမပြုပဲ အခြား Object တစ်ခုကို မူဖားယူခြင်း၊ ဖြည့်စွက်ပြင်ဆင်ခြင်းအားဖြင့် နောက် Object တွေကို တည် ဆောက်ရပါတယ်။ Class တွေကို အခြေခံတဲ့ OOP Language တွေကို Classical OOP လို ခေါ်ပြီး JavaScript ကို တော့ Prototype-based OOP လို ခေါ်ပါတယ်။

ဖြည့်စွက်ချက်။။ အနာဂတ်မှာ JavaScript ဖြစ်လာမယ့် ECMAScript 6 (ES6) မှာတော့ Class ရေးထုံး စတင်ပါဝင်လာပြီ ဖြစ်ပါတယ်။ NodeJS လို JavaScript Run-time မှာ စတင်အသုံးပြုလို ရနေပါပြီ။ မကြာခင်မှာ Web Browser တွေထဲမှာ လည်း Class ရေးထုံးကို အသုံးပြုထားတဲ့ JavaScript Code တွေကို ရေးသား အသုံးပြုနိုင်တော့မှာပါ။

Classical OOP မှာ Class တွေကို ဦးခုံးတည်ဆောက်ပြီးမှ အဲဒီ Class ကနေတစ်ဆင့် Object တွေ တည် ဆောက်ရပါတယ်။ Object Design ကို အရင်သတ်မှတ်ပြီးမှ တစ်ကယ့် Object ကို တည်ဆောက်တဲ့ သဘောပါ။ Prototype-based OOP မှာတော့ Object အလွတ်တစ်ခုကို အရင်တည်ဆောက်လိုက်ပြီး နောက်မှုလိုအပ်တဲ့ Data နဲ့ Function တွေကို လိုအပ်သလို တွဲဖက်သတ်မှတ်ရခြင်း ဖြစ်ပါတယ်။ Object Design တစ်ခုပုံသေမထား ပဲ လိုအပ်သလို ပြပိုင်ဖြည့် စွက်သွားတဲ့သဘော ဖြစ်ပါတယ်။ ဒါဟာ Classical OOP နဲ့ JavaScript ရဲ့ Prototype-based OOP ရဲ့ အမိကအကျတဲ့ ကွာခြားချက်ဖြစ်ပါတယ်။

### Creating Objects

JavaScript မှာ Object တွေရရှိဖို့အတွက် JSON လိုခေါ်တဲ့ နည်းစနစ်ကို အသုံးပြုနိုင်ပါတယ်။ JSON အကြောင်း ကို တော့ နောက်တစ်ခန်းမှ ဖော်ပြပါမယ်။ ဒီနေရာမှာတော့ Classical OOP ရဲ့ Class နဲ့ ပိုပြီးနီးစပ်တဲ့ Object Constructor အသုံးပြုခြင်းအားဖြင့် Object တွေ တည်ဆောက်နိုင်ပဲကို ဖော်ပြပေးပါမယ်။

JavaScript မှာ Function တွေကို Object Constructor အနေနဲ့လည်း အသုံးပြု နိုင်ပါတယ်။ ဥပမာ -

## JavaScript

```
1. function Person() {}
```

အခြေအားဖြင့် Person မှုည်နဲ့ Function အလွတ်တစ်ခုကို တည်ဆောက်လိုက်ခြင်းဖြစ်ပါတယ်။ ထူးစွားချက်အနေ နဲ့ ဒီ Function ကနေတစ်ဆင့် Object များကို အခုလို တည်ဆောက်ယူနိုင်မှာပဲဖြစ်ပါတယ်။

## JavaScript

```
1. var alice = new Person();
2. var bob = new Person();
```

ဒီနည်းနဲ့ alice နဲ့ bob လိုပေါ်တဲ့ ကိုယ်ပိုင် Data နဲ့ Function တွေ မရှိသေးတဲ့ Object အလွတ်နှစ်ခု ရရှိသွားမှာပဲ ဖြစ်ပါတယ်။ Person Object Constructor အတွက် Data နဲ့ Function တွေကို အခုလို သတ်မှတ်ပေးနိုင်ပါတယ်။

## JavaScript

```
1. Person.prototype.fullName = "Jame Doe";
2. Person.prototype.sayName = function() {
3.     console.log("Hello, I am " + this.fullName);
4. }
```

prototype Attribute ကနေတစ်ဆင့် Object Constructor အတွက် Data နဲ့ Function တွေကို တွဲဖက်ပေးရခြင်းပဲဖြစ်ပါတယ်။ ဒီလိုနောက်မှတွဲဖက်ခြင်းမဟုတ်ပဲ Object Constructor ကြော်ကတည်းက တစ်ခါတည်းသတ်မှတ် လိုရင်လည်း အခုလိုသတ်မှတ်နိုင်ပါတယ်။

## JavaScript

```
1. function Person(name) {
2.     this.fullName = name;
3.     this.sayName = function() {
4.         console.log("Hello, I am " + this.fullName);
5.     }
6. }
```

Function အတွင်းထဲမှာ this Keyword ကို သုံးပြီးသတ်မှတ်လိုတဲ့ Data နဲ့ Function တွေကို ကြိုတင်သတ်မှတ် နိုင်ခြင်းဖြစ်ပါတယ်။ ခုနေ Person Object တစ်ခုတည်ဆောက်ပြီး sayName() Function ကို အလုပ်လုပ်စေရင် အခုလိုရလဒ်ကို ရရှိမှာဖြစ်ပါတယ်။

## JavaScript

```
1. var jame = new Person('Jame Doe');
2. jame.sayName(); // => Hello, I am Jame Doe
```

## Information Hiding

JavaScript မှာ Public, Private စတဲ့ Access Control Modifier တွေ မရှိပါဘူး။ ဒါကြောင့် Information Hiding လုပ်ဆောင်ချက် ရရှိလိုရင် အခြားနည်းလမ်းတစ်ခုကို အသုံးပြုရပါတယ်။ Object တည်ဆောက်စဉ်က Data နဲ့ Function တွေကို Constructor အတွင်းမှာ this နဲ့ မတဲ့ဖောက်သတ်မှတ်ခဲ့တာကို သတိပြုမိမှပါ။ Data နဲ့ Function တွေကို this နဲ့ မတဲ့ဖောက်ပဲ Constructor အတွင်းမှာ ဒီအတိုင်းရေးသားခြင်းအားဖြင့် ပြင်ပကနေ ရယူအသုံးမပြုနိုင်တဲ့ Private Data နဲ့ Function များကို ရရှိနိုင်မှာဖြစ်ပါတယ်။

## JavaScript

```
1. function Person(name) {
2.     this.fullName = name;
3.     this.sayName = function() {
4.         sayHello(this.fullName);
5.     }
6.
7.     function sayHello(name) {
8.         console.log("Hello, I am " + name);
9.     }
10. }
11.
12. var foo = new Person("James");
13. foo.sayName(); // => Hello, I am James
14. foo.sayHello(); // => TypeError: foo.sayHello is not a function
```

Person Constructor အတွင်းမှာ ရေးသားထားတဲ့ sayHello() Function ဟာ Constructor အတွင်းမှာသာ ရယူအသုံးပြုနိုင်တဲ့ Private Function တစ်ခုဖြစ်ပါတယ်။ ပြင်ပကနေရယူဖို့ကြိုးစားတဲ့အခါ လိုင်းနံပါတ် (၁၄) မှာပြထား သလို Error ဖြစ်သွားမှာပါ။ ဒီနည်းနဲ့ Information Hiding လုပ်ဆောင်ချက်ကို JavaScript မှာ ရရှိနိုင်ပါတယ်။

## Inheritance

JavaScript မှာ Object Constructor ရဲ Prototype ကို မူဖားယူခြင်းအားဖြင့် Inheritance လုပ်ဆောင်ချက် ရရှိ နိုင် ပါတယ်။

### JavaScript

```
1. function Person() {};
2. Person.prototype.fullName = "Jame Doe";
3. function Student() {};
4. Student.prototype = Object.create(Person.prototype);
```

လိုင်းနံပါတ် (၄) မှာ Person ရဲ Prototype ကို Build-in Method တစ်ခုဖြစ်တဲ့ Object.create() ရဲ အကူအညီနဲ့ Student ရဲ Prototype အဖြစ် ကူးယူသတ်မှတ်လိုက်ခြင်းဖြစ်ပါတယ်။ ဒီနည်းနဲ့ Person မှာ ရှိနေတဲ့ Data နဲ့ Function တို့ကို Student က ဆက်ခံရရှိသွားစေပါတယ်။ Prototype အနေဖြင့် ဖော်ပြုသွားတဲ့ Data နဲ့ Function တို့ကိုတော့ အခုလို အောင်ယူနိုင် ပါတယ်။

### JavaScript

```
1. function Person(name) {
2.     this.fullName = name;
3.     this.sayName = function() {
4.         console.log("Hello, I am " + this.fullName);
5.     };
6. }
7.
8. function Student(name, major) {
9.     Person.call(this, name);
10.
11.    this.major = major;
12.    this.sayMajor = function() {
13.        console.log("I learn " + this.major);
14.    }
15. }
16.
17. foo = new Student("Jame", "Computer Sciences");
18. foo.sayName();           // => Hello, I am Jame
19. foo.sayMajor();         // => I learn Computer Sciences
```

နမူနာမှာ အဓိကကျတာကတော့ လိုင်းနံပါတ် (၉) က Person.call() ဖြစ်ပါတယ်။ JavaScript မှာ Function တွေ ကိုယ်တိုင်က Object တွေဖြစ်ကြပါတယ်။ ဒါကြောင့် Person ဟာ Object Constructor တစ်ခု ဖြစ်သလို Object တစ်ခုလည်း ဖြစ်နေပါတယ်။ call() Function ဟာ Person ဆိုတဲ့ Function Object ရဲ Build-in Method တစ်ခုပါ။

Student Constructor တဲ့မှာ Person.call() ဆိုပြီး ရေးသားထားတဲ့ အတွက် Student Constructor ကို အသုံးပြုပြီး Object တည်ဆောက်တိုင်းမှာ Person Constructor ကိုလည်း ခေါ်ယူအလုပ်လုပ်ပေးသွားမှာဖြစ်

ပါတယ်။ `call()` Function ထဲမှာ `this` ကို လက်ဆင့်ကမ်းပေးထားတာကို သတိပြုပါ။ ဒီလိုလက်ဆင့်ကမ်းပေးထားတဲ့ အတွက် Person Constructor အလုပ်လုပ်တဲ့အခါ အသုံးပြုတဲ့ `this` က Person ရဲ့ `this` မဟုတ်ဘေးပဲ အတွက်အသုံးပြု အလုပ်လုပ်သွားစေမှာဖြစ်ပါတယ်။ ဒါကြောင့် Person Constructor ထဲမှာ `this` အတွက်သတ်မှတ်ထားတဲ့ သတ်မှတ်ချက်တိုင်းဟာ Student ရဲ့ `this` ကို သက်ရောက်သွားပြီး၊ Person ရဲ့လုပ်ဆောင်ချက်အားလုံးကို Student က ဆက်ခံရရှိ သွားမှာပဲဖြစ်ပါတယ်။

Inheritance ရရှိသွားတဲ့အတွက် Polymorphism လုပ်ဆောင်ချက်လည်း တွဲဖက်ရရှိသွားနိုင်မှာပဲဖြစ်ပါတယ်။

## Interface

JavaScript မှာ Interface လုပ်ဆောင်ချက် မပါဝင်ပါဘူး။ Interface ဆိုတာဟာ အသေးစိတ် လုပ်ဆောင်ချက် ကွာခြားပေမယ့် အပြန်အလှန် ဖလှယ်အသုံးပြုနိုင်တဲ့ Object တွေ တည်ဆောက်နိုင်ဖို့အတွက် ဘုစ္စကိုသတ်မှတ်ပေးတဲ့ နည်းစနစ်တစ်မျိုးဖြစ်ပါတယ်လို့ ပြောခဲ့ပြီးဖြစ်ပါတယ်။ Static Type Language တွေမှာ Interface ရှိမှုသာ အပြန်အလှန် ဖလှယ်အသုံးပြုနိုင်တဲ့ Object တွေကို ရရှိနိုင်မှာ ဖြစ်ပေမယ့် JavaScript ကတော့ Dynamic Type Language ဖြစ်တဲ့အတွက် ဘုစ္စတစ်ခု သီးခြားမလိုအပ်ပဲ၊ Object တွေကို အလိုအလျောက် အပြန်အလှန် ဖလှယ်အသုံးပြုနိုင်တဲ့အတွက်ပဲ ဖြစ်ပါတယ်။

### JavaScript

```

1. function Student() {
2.     this.sayHello = function() {
3.         console.log("Hello, I'm a student");
4.     }
5. }
6.
7. function Worker() {
8.     this.sayHello = function() {
9.         console.log("Hello, I'm a worker");
10.    }
11. }
12.
13. function greet(person) {
14.     person.sayHello();
15. }
16.
17. var alice = new Student();
18. var bob = new Worker();
19.
20. greet(alice);      // => Hello, I'm a student
21. greet(bob);       // => Hello, I'm a worker

```

နမူနာမှာ `greet()` Function ကို ခေါ်ယူစဉ်ပေးတဲ့ Object က Student Object လည်းဖြစ်နိုင်သလို Worker Object လည်း ဖြစ်လို့ရပါတယ်။ ဒီ Object နှစ်ခုဟာ Interface တစ်ခုတည်းကနေ ဆက်ခံထားခြင်း မဟုတ်ပေမယ့် အလို အလျောက် အပြန်အလှန်ဖလှယ်အသုံးပြုလိုရနေတဲ့ သဘောပဲဖြစ်ပါတယ်။ ပေးလိုက်တဲ့ Object မှာ `sayHello()` Function ရှိမှုသာ လိုပါတယ်။ ဒီလိုလိုအပ်တဲ့ Data နဲ့ Function တွေ ရှိနေကြောင်း သေချာဖို့ကိုတော့ ရေးသားသူ ကိုယ်တိုင်က ဂရစိုက်ရေးသားပေးရမှာပဲဖြစ်ပါတယ်။

### 3.3 SOLID

SOLID Principles ရဲ့အမို့ပါယ်အကျဉ်းကို ဒီအခန်းအစမှာ ဖော်ပြခဲ့ပြီးဖြစ်ပါတယ်။ ဒီ Principles တွေဟာ ပြုပြင် ထိမ်းသိမ်းရလွယ်တဲ့ ပရိုဂရမ်တွေ ရေးသားဖန်တီးရာမှာ အမိကကျတဲ့ သဘောတရားတွေဖြစ်ပါတယ်။ OOP ကို Code ရေးသားမှုလမ်းညွှန် လိုဆိုနိုင်ပြီး SOLID ကိုတော့ Code Design လမ်းညွှန်လို ဆိုနိုင်ပါတယ်။ SOLID ကို Principle of OOD (Object Oriented Design) လိုလည်း ခေါ်ပါတယ်။ Uncle Bob လိုလူသိများတဲ့ Robert C. Martin ဆိုသူ Software ပညာရှင်တစ်ဦးက အစပြုဖော်ထုတ်ခဲ့တဲ့ သဘောတရားများဖြစ်ပါတယ်။

#### Single Responsibility Principle

SOLID ရဲ့ S ကို ကိုယ်စားပြုတဲ့ Single Responsibility Principle က အခုလိုဆိုပါတယ်။

" Object တစ်ခုဟာ လုပ်ငန်းတာဝန် တစ်ခုကိုသာ ဆောင်ရွက်သင့်တယ် "

Object တစ်ခုမှာ Data နဲ့ Function ပေါင်းများစွာပါဝင်နိုင်ပါတယ်။ ဒါပေမယ့် ရည်ရွယ်ချက်ကတော့ တစ်ခု တည်းပဲ ဖြစ်သင့်ပါတယ်။ Data နဲ့ Function တွေဟာ တူညီတဲ့ Object ရဲ့ အမိက ဗိုးတည်ချက်ရရှိရေးအတွက် ပုံပိုးပေးရန် အတွက်သာ ဖြစ်သင့်ပါတယ်။ Object ရဲ့ အမိကဗိုးတည်ချက်နဲ့ မသက်ဆိုင်တဲ့ လုပ်ဆောင်ချက်ကို ဆောင်ရွက်ပေးတဲ့ Data တွေ Function တွေ မပါဝင်သင့်ပါဘူး။

#### Pseudo-code

```

1. function Student(name, major) {
2.     this.name = name;
3.     this.major = major;
4.     this.signRollCall = function() { this.rollCall++ } ;
5.
6.     this.displayInfo = function() {
7.         return "<b>" + this.name + "</b>" +
8.                 ": <i>" + this.major + "</i>";
9.     }
10. }
```

နဲ့မှုနာ Student Constructor ရဲ့တာဝန်ကတော့ ကျောင်းသားတစ်ယောက်နဲ့ ပက်သက်တဲ့ အချက်အလက်များ ကို စုစုပေါင်းထားဖို့ဖြစ်ပါတယ်။ Object တည်ဆောက်စဉ်မှာ name နဲ့ major ကို Assign လုပ်ပြီး signRollCall() ကို ခေါ်ယူတိုင်းမှာ rollCall တန်ဖိုးကို တစ်တိုးပေးသွားမှာဖြစ်ပါတယ်။ ဒီလုပ်ငန်းတွေက ကျောင်းသားအချက် အလက်များစုစုပေါင်းထားဖို့ဆိုတဲ့ ဗိုးတည်ချက်ကိုထောက်ပံ့ပေးတဲ့ လုပ်ဆောင်ချက် များဖြစ်ပါတယ်။ displayInfo() Function ကတော့ ကျောင်းသားရဲ့ name နဲ့ major ကို HTML Format လုပ်ပြီးဖော်ပြစေဖို့ဖြစ်ပါတယ်။ ဒီလုပ်ဆောင် ချက်ကတော့ မူလရည်ရွယ်ချက်ကနေ သွေဖီသွားတဲ့အတွက် Single Responsibility ကို ချိုးဖောက်မိသွားပါပြီ။ Student Object ရဲ့ တာဝန်က ကျောင်းသားနဲ့သက်ဆိုင်တဲ့ အချက်အလက်များ စုစုပေါင်းထားဖို့အပြင် ကျောင်းသားရဲ့ name နဲ့ major ကို Format လုပ်ပြီး ဖော်ပြဖို့ဆိုတဲ့ နောက်တာဝန်တစ်ခုကိုပါ ယူထားရသလို ဖြစ်နေပါတယ်။ ကြည့်လိုက်ရင် ကျောင်းသားနဲ့ပက်သက်တဲ့ အလုပ်ပဲ လို့ အလွယ်ယူဆနိုင်ရာရှိပေမယ့်၊ လက်တွေ့မှာ ဒီတာဝန်ကိုနောက် Constructor တစ်ခုနဲ့ ခဲ့ထုတ်လိုက်တာက ပိုကောင်းပါတယ်။

### Pseudo-code

```

1. function Student(name, major) {
2.     this.name = name;
3.     this.major = major;
4.     this.signRollCall = function() { this.rollCall++ }
5. }
6.
7. function ViewFormat() {
8.     this.displayStudentInfo = function(student) {
9.         return "<b>" + student.name + "</b>" +
10.            ": <i>" + student.major + "</i>";
11.    }
12. }

```

နှမှနာမှာ ကျောင်သားအချက်အလက်တွေကို HTML Format လုပ်ပြီး ဖော်ပြနိုင်တဲ့ displayStudentInfo() ကို ViewFormat လိုပေါ်တဲ့ နောက် Constructor နဲ့ခြားထားလိုက်ပါတယ်။ ဒါကြောင့် Student က ကျောင်းသား အချက်အလက်ဆိုင်ရာလုပ်ငန်းတစ်ခုကိုသာ တာဝန်ယူပြီး လိုအပ်သလို Format လုပ်ဖော်ပြခြင်း ကိုတော့ ViewFormat က တာဝန်ယူသွားမှာပဲဖြစ်ပါတယ်။

ဒီလိုခွဲ့ခြားထားလိုက်တဲ့အတွက် Format အပြောင်းအလဲရှိလာတဲ့အခါ သက်ဆိုင်ရာ ViewFormat ကိုသာ ပြင်ဆင်ဖို့ လိုပြီး Student ကို ထိစရာမလိုတော့တဲ့ အားသာချက်ကိုရရှိပါတယ်။ ဒုဥကြောင်း ဒောင်လိုအပ်လို XMLFormat လိုပေါ်တဲ့ Format လိုပေါ်တဲ့ လုပ်ဆောင်ချက်လိုအပ်လာရင်လည်း Student ကို သွားပြင်စရာမလိုပဲ၊ သီးခြား Implement လုပ်နိုင် တဲ့အားသာချက်ကိုရရှိမှာပဲဖြစ်ပါတယ်။ Single Responsibility Principle ဆုံးတာ ဒီနည်းနဲ့ ပြုပြင်ထိမ်းသိမ်း ရဂွယ်ကူတဲ့ Code Design ရရှိစေဖို့ လမ်းညွှန်ထားတဲ့ သဘောသဘာဝတစ်ခုပဲ ဖြစ်ပါတယ်။

### Open/Closed Principle

SOLID ၄ ကို ကိုယ်စားပြုတဲ့ Open/Closed Principle ရဲ့အမိပိုယ်က ဒီလိုပါ။

” Object တစ်ခုအား ဖြည့်စွက်ခြင်းကို ကြိုဆိုသင့်ပေမယ့် ပြင်ဆင်ခြင်းကို လက်မခံသင့်ပါ ”

မူရင်းအခိုပါယ်က Open to extension, closed to modification ဖြစ်ပါတယ်။ လက်ရှိအလုပ်လုပ်နေတဲ့ အသေအချာ စမ်းသပ်ထားပြီးသား Object တစ်ခုကို Bug Fix လုပ်တာကလွှဲရင်၊ Requirement ပြောင်းလဲသွားတဲ့ အတွက်ကြောင့် သော်လည်ကောင်း၊ အခြားအကြောင်းတစ်ခုကြောင့်သော်လည်ကောင်း နောင်မှာပြင်ဆင်ခြင်း မပြုလုပ်သင့်ဘူးဆိုတဲ့ အမိပိုယ်ပါ။ ပြင်ဆင်စရာအကြောင်းရှိလာတဲ့အခါ အလုပ်လုပ်နေတဲ့ Data နဲ့ Function တွေကို ပြင်မယ့်အစား Data အသစ် Function အသစ်တွေဖြည့်စွက်ခြင်းနည်းလမ်းဖြင့်သာ လိုချင်တဲ့ လုပ်ဆောင်ချက်ရအောင် ဆောင်ရွက်သင့် တယ်လိုဆိုတဲ့သော့ ဖြစ်ပါတယ်။

## Pseudo-code

```

1. function Order(items) {
2.     this.items = items;
3.     this.amount = function() {
4.         var total = 0;
5.         for(item in this.items) {
6.             total += item.value;
7.         }
8.         return total;
9.     }
10. }

```

နှမှနာကိုလေ့လာကြည့်ပါ။ ပေးလာတဲ့ Item စာရင်းကနေ ကျသင့်ငွေတွက်ပေးတဲ့ amount() Function ပါဝင်ပါတယ်။ နောင်မှာ လိုအပ်ချက်အရ တန်ဖိုးတွက်တဲ့အခါ မြန်မာကျပ်နဲ့ပါ တွက်ချက်ပေးစေလိုတယ်ဆိုရင် အခုလိုပြင်ဆင်လိုက်နိုင် ပါတယ်။

## Pseudo-code

```

1. function Order(items) {
2.     this.items = items;
3.     this.amount = function(currency) {
4.         var total = 0;
5.         for(item in this.items) {
6.             total += item.value;
7.         }
8.
9.         if(currency == "MMK")
10.             return total * getExchangeRate();
11.         else
12.             return total;
13.     }
14. }

```

amount() Function မှာ currency Parameter ထည့်ပြီး currency က MMK ဆိုရင် တစ်မျိုးတွက်ပြီး၊ မပါရင် တစ်မျိုးတွက်အောင် ပြင်ဆင်လိုက်ခြင်း ဖြစ်ပါတယ်။ လိုအပ်တဲ့လုပ်ဆောင်ချက် ရသွားမှာဖြစ်ပေမယ့် မူလ အလုပ်လုပ် နေပြီးသား Function ကို ပြင်ဆင်လိုက်ခြင်းအားဖြင့် Open/Closed Principle ကို ချိုးဖောက်လိုက်ခြင်းပဲဖြစ်ပါတယ်။ ပိုမိုမှန်ကန်တဲ့ နည်းလမ်းကတော့ အခုလိုဖြစ်မှာပါ။

## Pseudo-code

```

1. function Order(items) {
2.     this.items = items;
3.     this.amount = function() {
4.         var total = 0;
5.         for(item in this.items) {
6.             total += item.value;
7.         }
8.         return total;
9.     }
10.
11.    this.kyatAmount = function() {
12.        var total = this.amount();
13.        return total * getExchangeRate();
14.    }
15. }

```

လက်ရှိအလုပ်လုပ်နေတဲ့ amount() Function ကို မပြင်ပဲ kyatAmount() Function ဖြည့်စွက်လိုက်ခြင်း ဖြစ်ပါတယ်။ ဒီနည်းနဲ့ Kyat Amount လိုတဲ့နေရာမှာ amount() Function အစား kyatAmount() Function ကို အစားထိုးအသုံးပြုနိုင်မှာပဲ ဖြစ်ပါတယ်။

Open/Closed Principle ဟာ SOLID မှာ လိုက်နာဖို့အခက်ဆုံး တစ်ခုဖြစ်ပါတယ်။ လက်တွေ့မှာ ရှိနေတဲ့ Code ကို လုံးဝမပြင်ပဲ ပြောင်းလဲသွားတဲ့ Requirement ရဲလိုအပ်ချက်ကို ပြည့်စုံစေဖို့ မလွယ်ပါဘူး။ အနည်းနဲ့အများ တော့ ပြင်ဆင်ဖို့ လိုအပ်တက်ပါတယ်။ တက်နိုင်သလောက သတိပြုလိုက်နာရမှာပဲ ဖြစ်ပါတယ်။ ပြီးခဲ့တဲ့ အခန်းမှာ လုပ်ဆောင်ချက်တစ်ခုကို ပြင်လိုက်လို့ အခြားမသက်ဆိုင်ဘူးလို့ ထင်ရတဲ့ လုပ်ဆောင်ချက်မှာ ပြသေနာ သွားတက်နေတက်ကြောင်း ဖော်ပြခဲ့ပြီး ဖြစ်ပါတယ်။ Open/Closed Principle က ဒီပြသေနာမျိုးကို လျှော့ချေခဲ့ချေမှု ဖို့ လုပ်ဆောင်ချက်ကို မပြင်ဖို့ တိုက်တွန်းခြင်း ဖြစ်ပါတယ်။

## Liskov Substitution Principle

SOLID ရဲ့ L ကို ကိုယ်စားပြုတဲ့ Liskov Substitution Principle ရဲ့ အဓိပ္ပာယ်က ဒီလိုပါ။

" Inherit လုပ်ယူထားတဲ့ Sub-Class ကို မူလ Class အစား သုံးလို့ ရနိုင်ရမယ် "

ဒီ Principle ကို စတင်ဖော်ထဲတဲ့ ချွေးချွေးဘာသိပုံးပညာရှင်ကို အစွဲပြုပြီး Liskov Substitution Principle လို့ခေါ်ခြင်း ဖြစ်ပါတယ်။ Class တစ်ခုကို Inherit လုပ်တယ်ဆိုတာ အခြေခံအားဖြင့် မူလ Class ရဲ့ Data နဲ့ Function တွေကို ဆက်ခံရရှိလိုတဲ့ အတွက် ဖြစ်ပါတယ်။ ဆက်ခံရရှိထားတဲ့ မူလ Class ရဲ့ Data နဲ့ Function တွေကို လိုအပ်သလို ပြင်ဆင်ရနိုင်ပါတယ်။ အဲဒီလိုပြင်ဆင်တဲ့ အခါ မူလ Class မှာ သတ်မှတ်ခဲ့တဲ့ သဘောသဘာဝကနေ လုံးဝပြောင်းလဲသွားအောင် ပြင်ဆင်ခြင်းမပြဖို့ Liskov Substitution Principle က တားမြစ်ထားပါတယ်။ ဒါအပြင် Sub-Class အတွင်းမှာ မူလ Class မှာ မပါတဲ့ Data နဲ့ Function တွေကို ဖြည့်စွက်ဖို့လည်း လိုအပ်နိုင်ပါတယ်။ အဲဒီလိုဖြည့်စွက်ရာမှာ လည်း ဖြည့်စွက်လိုက်တဲ့ လုပ်ဆောင်ချက်တွေက မူလ Class ရဲ့ သဘောသဘာဝကနေ လုံးဝ ကွပ်းသွားစေတဲ့ ဖြည့်စွက်မှုမျိုး မလုပ်ဖို့ကို လည်း တာမြစ်ထားပါတယ်။

ဒီတော့မှာ Class က ရရှိလာတဲ့ Object ကို အသုံးပြုရမယ့်နေရာမှာ၊ လိုအပ်ချက်အရ Sub-Class ကနေရရှိလာတဲ့ Object နဲ့ အစားထိုးအသုံးပြုမယ်ဆိုရင် ပြဿနာတစ်ခုတစ်ရာမရှိပဲ အသုံးပြုလိုရစေမှာဖြစ်ပါတယ်။ Sub-Class ဟာ မူလ Class ရဲ့ လုပ်ဆောင်ချက်တွေကို ဆက်ခံထားပြီး၊ ပြင်ဆင်ဖြည့်စွက်မှုတွေကြောင့်လည်း အလုပ်လုပ်ပုံ သဘာဝ ကွဲပြား သွားခြင်းမရှိတဲ့ အတွက် ဖြစ်ပါတယ်။

Liskov Substitution Principle အကြောင်းပြောကြရာမှာ နမူနာပေးလေ့ရှိတာကတော့ Square Class ဖြစ်ပါတယ်။

### Pseudo-code

```

1. function Rectangle () {
2.     this.setWidth = function(width) {
3.         this.width = width;
4.     }
5.     this.setHeight = function(height) {
6.         this.height = height;
7.     }
8.     this.area = function() {
9.         return this.width * this.height;
10.    }
11. }
12.
13. function Square() {
14.     Rectangle.call(this);
15.     this.setWidth = function(width) {
16.         this.width = width;
17.         this.height = width;
18.     }
19.     this.setHeight = this.setWidth;
20. }

```

နမူနာမှာ Square Constructor ဟာ Rectangle ကို ဆက်ခံထားပါတယ်။ ဒါပေမယ့် Square ရဲ့ ထုံးစံအတိုင်း width နဲ့ height ဟာ တူညီရမှာဖြစ်လို့ setWidth() Function မှာ width နဲ့ height ကို တူညီအောင်သတ်မှတ် ထားသလို့ setHeight() Function ကလည်း အတူတူပဲအလုပ်လုပ်ရမှာဖြစ်လို့ setWidth() ကိုပဲ ပြန်လည်ရယူ အသုံးပြုထားပါတယ်။ ဒါဟာ မူလလုပ်ဆောင်ချက်ကနေ သွေ့စီသွားတဲ့ အတွက် Liskov Substitution Principle ကို ချိုး ဖောက်လိုက်ခြင်းပဲဖြစ်ပါတယ်။ Rectangle Object အသုံးပြုရမယ့်နေရာမှာ Square Object ကို အစားထိုးအသုံးပြုခဲ့မယ့် ဆိုရင်လည်း ပြဿနာရှိနိုင်ပါတယ်။ Rectangle လို့သောထားပြီး Width နဲ့ Height ကို သီးခြားစီ သတ်မှတ်ဖို့ ကြိုးစားပေမယ့် လက်တွေမှာ သတ်မှတ်နိုင်မှာ မဟုတ်လို့ မလိုလားအပ်တဲ့ Bug တွေ ထွက်ပေါ်လာမှာပဲဖြစ်ပါတယ်။

## Interface Segregation Principle

SOLID ရဲ့ ၁ ကို ကိုယ်စားပြုတဲ့ Interface Segregation Principle ရဲ့ အဓိပ္ပာယ်က ဒီလိပါ။

" လုပ်ဆောင်ချက်ပေါင်းများစွာပါဝင်တဲ့ Interface ကြီးတစ်ခုအစား၊ ကဏ္ဍအလိုက် ခွဲခြားထားတဲ့ သီးသန့် Interface ငယ်များကိုသာ အသုံးပြုသင့်တယ် "

3 Pin Head, 2 Pin Head, USB 2.0 Port, USB 3.0 Port စသဖြင့် အပေါက်များစွာပါဝင်တဲ့ Power Outlet Interface ကြီး တစ်ခုကို မျက်စိတဲ့ ခြင်းကြည့်လိုရပါတယ်။ Single Responsibility Principle အခြင်အရကြည့်မယ်ဆိုရင် Power Outlet ဆိုတဲ့ လုပ်ဆောင်ချက် တစ်ခုတည်းကို ဆောင်ရွက်နေခြင်းဖြစ်ပေမယ့်။ Port အမျိုးအစားပေါင်းများစွာကို Support လုပ်ပေးတဲ့ Interface ကြီးတစ်ခုဖြစ်နေပါတယ်။ USB Power Outlet လုပ်ဆောင်ချက်ရဖို့အတွက် အဲဒီ Interface ကြီးကိုသုံးမယ်ဆိုရင် ကျော် Outlet တွေက အသုံးမလိုပဲနဲ့ ပါဝင်နေတဲ့သော့ ဖြစ်နေပါတယ်။ တစ်ခြား Outlet တစ်ခုမှာ Short Circuit ဖြစ်နေလို့ ကျော် Outlet တွေပါ ဘုံးမရတော့ဘူး ဆိုတာမျိုး ဖြစ်နိုင်ပါတယ်။ Interface Segregation Principle က Client (Object ကို ရယူအသုံးပြုယူ) ဟာ လိုချင်တဲ့ လုပ်ဆောင်ချက်ရဖို့အတွက် မလိုတဲ့ လုပ်ဆောင်ချက်တွေကို မိုးခိုးနေရာတာမျိုး မဖြစ်သင့်ဘူးလို့ ဆိုထားပါတယ်။ ဒါကြောင့် Interface ကြီးတစ်ခုအစား ကဏ္ဍအလိုက် ခွဲခြားထားတဲ့ သီးသန့် Interface များကိုသာ အသုံးပြုသင့်တယ်ဆိုတဲ့ အဓိပ္ပာယ်လည်း ဖြစ်ပါတယ်။

Interface ရေးထုံးပါဝင်တဲ့ OOP Language တွေမှာ Interface တွေ Design လုပ်ကတည်းက ဒီအချက်ကိုထည့်သွင်း စဉ်းစားသင့်ပြီး JavaScript လို့ Interface ရေးထုံး မပါဝင်တဲ့ Language တွေမှာတော့ Object တစ်ခုချင်းစီမှာပါဝင်တဲ့ လုပ်ဆောင်ချက်တွေကို သတိပြု ကန်သတ်ခြင်းအားဖြင့် Interface Segregation လုပ်ဆောင်ချက်ကို ရရှိနိုင်ပါတယ်။

### Pseudo-code

```

1. function Car() {
2.   this.engine = function() { ... }
3.   this.break = function() { ... }
4.   this.playRadio = function() { ... }
5.   this.ejectCD = function() { ... }
6. }
```

နှမူနာ Car Constructor မှာ engine(), break() လုပ်ဆောင်ချက်များအပြင် Audio ပိုင်းဆိုင်ရာလုပ်ဆောင် ချက်များကိုပါ ပေါင်းစပ်ထည့်သွင်းထားပါတယ်။ Interface Segregation Principle အရ အခုံလိုပြုခြင်းလိုက်နိုင်ပါတယ်။

**Pseudo-code**

```

1. function SpeedControl() {
2.   this.engine = function() { ... }
3.   this.break = function() { ... }
4. }
5. function AudioControl() {
6.   this.playRadio = function() { ... }
7.   this.ejectCD = function() { ... }
8. }
9.
10. function Car() {
11.   this.speed = new SpeedControl();
12.   this.audio = new AudioControl();
13. }

```

SpeedControl နဲ့ AudioControl ဆိုတဲ့ Interface နှစ်ခု ခွဲထုတ်လိုက်ပြီးမှ အဲဒီ Interface နှစ်ခုကို Car Constructor မှာ ရယူအသုံးပြထားခြင်းဖြစ်ပါတယ်။ ဒီနည်းနဲ့ SpeedControl နဲ့ AudioControl တိုကို သီးခြား စီ Maintain လုပ်နိုင်မှာ ဖြစ်သလို Audio လုပ်ဆောင်ချက်မလိုတဲ့ အခါမှာလဲ ချုန်လုပ်ထားနိုင်မှာ ဖြစ်ပါတယ်။ ဒီနည်းနဲ့ Client Interface တွေ ခွဲထုတ်၊ အမှန်တစ်ကယ်လိုအပ်တဲ့ Interface များကိုသာ ချိတ်ဆက် အလုပ် လုပ်စေခြင်းအားဖြင့် မလိုအပ်တဲ့ လုပ်ဆောင်ချက်တွေ ပါဝင်နေခြင်းနဲ့မလိုအပ်တဲ့ လုပ်ဆောင်ချက်တွေ ကို မိုးခိုးနေရခြင်း မရှိတော့ပဲ Interface Segregation Principle လုပ်ဆောင်ချက်ကို ရရှိသွားမှာ ဖြစ်ပါတယ်။

**Dependency Inversion Principle**

SOLID ရဲ့ D ကို ကိုယ်စားပြုတဲ့ Dependency Inversion Principle ရဲ့အဓိပ္ပာယ်က ဒီလိုပါ။

" အဆင့်မြင့် Object ဟာ အဆင့်နိမ့် Object ရဲ့ လုပ်ဆောင်ချက်ကို မမိုးခိုးသင့်ဘူး "

အဆင့်မြင့် (Higher Level) နဲ့ အဆင့်နိမ့် (Lower Level) ဆိုတဲ့ ဆိုလိုရင်းကိုမြင်သာစေဖို့အတွက် Code နမူနာကို ကြည့်ပါ။

**Pseudo-code**

```

1. function PDF() {
2.   this.read = function() {
3.     // parse and read PDF file
4.   }
5. }
6.
7. function Reader() {
8.   this.book = new PDF();
9.   this.read = this.book.read();
10. }

```

နမူနာမှာ PDF ဟာပိုပြီးအခြေခံကျတဲ့ အဆင့်နိမ့် Object ဖြစ်ပါတယ်။ Reader ကတော့ PDF ကို အသုံးချထားတဲ့ အဆင့်မြင့် Object ဖြစ်ပါတယ်။ အဆင့်မြင့် Object ဖြစ်တဲ့ Reader ရဲ့ read() Function ဟာ လက်တွေ

မှာ အဆင့်နိမ့် Object ဖြစ်တဲ့ PDF ရဲ့ `read()` ကို ပြန်လည်အသုံးပြုထားခြင်းဖြစ်ပါတယ်။ ဒါကြောင့် အဆင့်မြင့် Object ဖြစ်တဲ့ Reader ဟာ အဆင့်နိမ့် Object ဖြစ်တဲ့ PDF ရဲ့ အသေးစိတ်လုပ်ဆောင်ချက်ကို မိမိခိုနေတဲ့ သဘော ဖြစ်လာပါတယ်။ ဒီလိုပိုမိုနေတဲ့ အတွက် ဖြစ်ပေါ်လာတဲ့ အားနည်းချက်ကတော့ ပရိုဂရမ်ကို ပြင်ဆင်ရ ခက်ခဲသွားခြင်းပဲ ဖြစ်ပါတယ်။ Reader ဟာ PDF က လွှဲရင် တစ်ခြား File အမျိုးအစားနဲ့ အလုပ်လုပ်နိုင်မှာ မဟုတ်တော့ပါဘူး။ PDF ကို လုံးလုံးမိမိထားတဲ့ အတွက် ကြောင့် ဖြစ်ပါတယ်။ ဒါကြောင့် Dependency Inversion Principle က သတ်မှတ်ထားတဲ့ အတိုင်း အခုလို ပြင်ဆင်လိုက်နိုင်ပါတယ်။

#### Pseudo-code

```

1. function PDF() {
2.     this.read = function() {
3.         // parse and read PDF file
4.     }
5. }
6.
7. function Epub() {
8.     this.read = function() {
9.         // parse and read epub file
10.    }
11. }
12.
13. function Reader(book) {
14.     this.book = book;
15.     this.read = this.book.read();
16. }

```

နဲ့မှုနာမှာ PDF နဲ့ Epub ဆိုတဲ့ Low Level Object နှစ်ခုပါဝင်ပြီး Reader ဆိုတဲ့ Hight Level Object တစ်ခု ပါဝင် ပါတယ်။ Reader ဟာ လက်တွေမှာ PDF နဲ့ Epub တို့ရဲ့ `read()` Function ကို အခြေခံပြီး အလုပ်လုပ်ခြင်း ဖြစ်ပေမယ့် PDF သို့မဟုတ် Epub တစ်ခုခုကို အသေးစိတ်မိမိခြင်းမရှုပါဘူး။ အဲဒီလိုပိုမိုမယ့်အစား Reader Object တည် ဆောက်စဉ်မှာ PDF သို့မဟုတ် Epub အသုံးပြုစေလိုတဲ့ Object ကိုပေးဖို့ သတ်မှတ်ထားပြီး ပေးလာတဲ့ Object ရဲ့ `read()` Function ကို အသုံးပြုလိုက်ခြင်းအားဖြင့် Reader ဟာ PDF သို့မဟုတ် Epub သို့မဟုတ် အခြား File Format တွေကိုဖတ်ရှုပေးနိုင် Object တစ်ခုဖြစ်သွားပါတယ်။ Dependency Inversion အကူအညီနဲ့ ပရိုဂရမ်ရဲ့အလုပ် လုပ်ပဲ အများကြီးပိုပြီး Flexible ဖြစ်သွားတဲ့ သဘောပဲ ဖြစ်ပါတယ်။

#### Pseudo-code

```

1. var pdf = new PDF();
2. var PDFReader = new Reader(pdf);
3.
4. var epub = new Epub();
5. var EpubReader = new Reader(epub);

```

Interface ရေးထုံးပါဝင်တဲ့ Language တွေမှာတော့ Dependency Inversion လုပ်ဆောင်ချက် ရရှိဖို့ Interface တွေကို အားကိုးကြလေ့ရှုပါတယ်။ အဆင့်မြင့် Object က အဆင့်နိမ့် Object ကို မမှုခိုပဲ ကြားခံ Abstraction ဖြစ်တဲ့ Interface ကိုမြို့ခိုလိုက်ခြင်းအားဖြင့် Dependency Inversion လုပ်ဆောင်ချက်ကို ရရှိနိုင်ခြင်းပဲဖြစ်ပါ

တယ်။ လက်တွေမှာ ဘယ်လို နည်းစနစ်ကိုပဲသုံးသုံး၊ အဆင့်မြင့် Object က အဆင့်နိမ့် Object ရဲ့ Detail အသေးစိတ်လုပ်ဆောင်ချက်ကို မဖို့ခို့ဘူး ဆိုရင် Dependency Inversion Principle ကို လိုက်နာနေဖြင့်ပါ တယ်။

## Conclusion

ဒီအခန်းမှာဖော်ပြခဲ့တဲ့ Code နမူနာတွေဟာ လက်တွေအသုံးချုန်နာတွေမဟုတ်ပဲ၊ OOP, SOLID စတဲ့ သဘော သဘာဝတွေ ပေါ်လွင်အောင် အသားပေးဖော်ပြထားတဲ့ ရိုးစင်းသောနမူနာ Code များသာဖြစ်ပါတယ်။ လက်တွေမှာ Code တွေ က ဒီလောက် ရိုးစင်းမှာမဟုတ်တဲ့အတွက် ဘယ်လောက်ထိ လိုက်နာအသုံးချိန်သလဲ ဆိုတာ ရေးသားသူရဲ့ အတွေ့အကြုံ ပေါ်မူတည်မှာပဲ ဖြစ်ပါတယ်။ ပြောခဲ့တဲ့ နည်းစနစ်နဲ့ သဘောသဘာဝ အားလုံးကို တစ်သဝေမတိမ်းလိုက်နာပြီး အကုန်လုံး အတိအကျ လိုက်လုပ်နေဖို့ မဟုတ်ပါဘူး။ သူ့နေရာနဲ့သူ သင့်တော်သလို အသုံးချွေားနိုင်မယ်ဆိုရင် စနစ်ကျပြီး ပြုပြင်ထိမ်း သိမ်းရလွယ်ကူတဲ့ ပရိုဂရမ်တွေ ရေးသား ဖန်တီးရာမှာ အထောက်အကူဖြစ်စေမှာပဲဖြစ်ပါတယ်။

ပြန်လည်အသုံးပြုလိုရတဲ့ Reusable Code ဟာ  
အသုံးဝင်ပါတယ်။ ပြန်လည်အသုံးပြုလိုရတဲ့  
Reusable Service ကတော့ ပိုအသုံးဝင်ပါတယ်။

### **Professional Web Developer (စာအုပ်)**

Web Standard, jQuery, PHP, MySQL, Ajax, CMS, MVC,  
HTML5, Mobile Web, Web Application Security စသည့်  
အကြောင်းအရာများကို ရေးသားဖော်ပြထားသည့်စာအုပ်  
အောက်ပါလိပ်စာတွင် အခဲ့ Download ရယူနိုင်သည်။

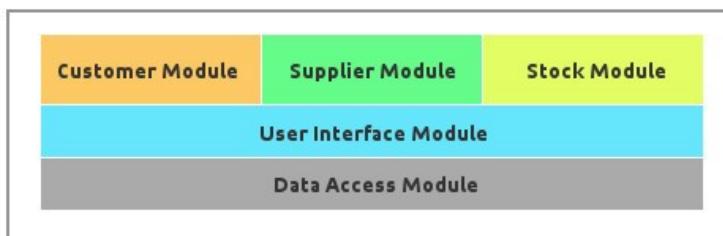
<http://pwdbook.com>

## အခန်း(၄) - Service Oriented Architecture

SOA လို့ အတိုကောက်ခေါ်တဲ့ Service Oriented Architecture ဆိုတာဟာ Software Solution တွေ တည်ဆောက်ရာမှာ တစ်ပေါင်းတစ်စည်းတည်း စုစုပေါင်း အလုပ်လုပ်တဲ့ Software Package အဖြစ် တည်ဆောက် မယ့်အစား သီးခြား အလုပ်လုပ်နိုင်တဲ့ Software ထိများ အဖြစ် ခွဲခြားတည်ဆောက်တဲ့ နည်းစနစ်ဖြစ်ပါတယ်။

Software Solution တစ်ခုကို ကဏ္ဍအလိုက် ခွဲခြားတည်ဆောက်တယ်ဆိုတာ အမှန်တော့ ထူးဆန်းတဲ့ ကိစ္စတစ် ခုမဟုတ်ပါဘူး။ ဥပမာ - စီးပွားရေးလုပ်ငန်းသုံး Business Solution တစ်ခုတည်ဆောက်တယ်ဆိုပါစို့။ Customer စာရင်းစီမံတဲ့ အစိတ်အပိုင်းကသပ်သပ်၊ Supplier စာရင်းစီမံတဲ့ အစိတ်အပိုင်းကသပ်သပ်၊ Order စာရင်းစီမံတဲ့ အစိတ်အပိုင်းက သပ်သပ်၊ Payment စိုင်းစီမံတဲ့ အစိတ်အပိုင်းကသပ်သပ်၊ Stock စိုင်းစီမံတဲ့ အစိတ်အပိုင်းကသပ်သပ် စသဖြင့် အပိုင်းလိုက်၊ အပိုင်းလိုက် ခွဲခြားရေးသားပြီး အဲဒီအစိတ်အပိုင်းတွေကို ပေါင်းစပ်ခြင်းအားဖြင့် လိုချင်တဲ့ Software Solution ရအောင် ဖန်တီးယူနိုင်ပါတယ်။

ဒီနည်းစနစ်ကို Modular Design လို့ခေါ်ပါတယ်။ Modular Design ကိုအသုံးပြုခြင်းအားဖြင့် ခွဲခြားထားတဲ့ အပိုင်းလိုက် သီးခြားစီ စမ်းသပ်နိုင်ခြင်း၊ သီးခြားစီ ပြင်ဆင်ထိမ်းသီမ်းနိုင်ခြင်းစတဲ့ အားသာချက်တွေကိုရရှိနိုင်ပါတယ်။ ဒါအပြင် Module တွေကို အမျိုးမျိုးတဲ့ စပ်ခြင်းအားဖြင့် မတူကွဲပြားတဲ့ Solution တွေကိုလည်း ရရှိနိုင်ပါသေးတယ်။ ဥပမာ - တစ်ချို့ Solution တွေ မှာ Customer, Supplier နဲ့ Stock တို့သာ ပါဝင်ပြီး တစ်ချို့ Solution တွေမှာ Stock, Order, Payment တို့ပဲ ပါဝင်တဲ့ သဘောမျိုး လိုအပ်ရင်ရရှိနိုင်ပါတယ်။

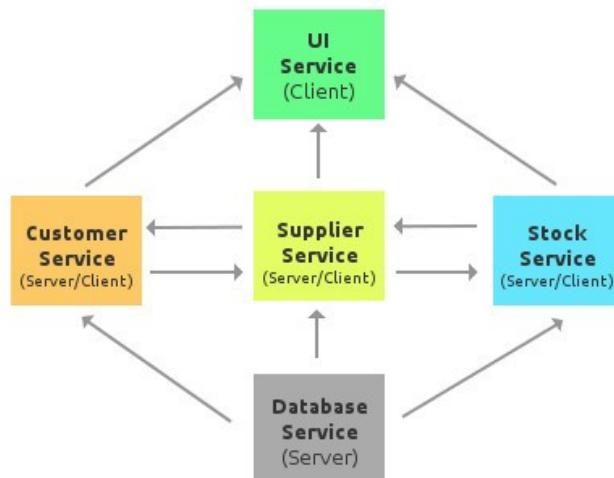


ပုံ (၄.၁) - Modular Design

Modular Design နဲ့ SOA မတူပါဘူး။ Modular Design မှာပါဝင်တဲ့ အစိတ်အပိုင်းတွေကို Module လို့ခေါ်ပြီး SOA မှာ ပါဝင်တဲ့ အစိတ်အပိုင်းတွေကို Service လို့ခေါ်ပါတယ်။ အခေါ်အဝေါ်အားဖြင့် Software ဆိုတာဟာ အသုံးပြုသူ User သုံးဖို့ ရည်ရွယ်ဖန်တီးထားတဲ့ ကွန်ပျူတာပရှိရမ်တစ်ခုဖြစ်ပါတယ်။ Service ဆိုတာကတော့ အသုံးပြုသူ User တင်မကပဲ အခြားကွန်ပျူတာပရှိရမ်တွေကပါ ဆက်သွယ်အသုံးပြုနိုင်တဲ့ Software တစ်မျိုးပဲ ဖြစ်ပါတယ်။ လူမလိုပဲ Machine to Machine ဆက်သွယ်အလုပ်လုပ်နိုင်တဲ့ Software တစ်မျိုးလိုလည်း ဆိုနိုင်ပါတယ်။

Modular Design နဲ့တည်ဆောက်ထားတဲ့ Solution တစ်ခု အလုပ်လုပ်ဖို့အတွက် သက်ဆိုင်ရာ Module တွေကို တွဲဆက် ပေးရပါတယ်။ SOA မှာပါဝင်တဲ့ Service တွေကတော့ တွဲဆက်ဖို့မလိုပဲ တစ်ခုချင်း သီးခြားအလုပ်လုပ် ကြပါတယ်။ Service တစ်ခုနဲ့တစ်ခု အပြန်အလှန် ဆက်သွယ်ခြင်းအားဖြင့် လိုအပ်တဲ့ Solution ကိုရရှိတဲ့ သဘောဖြစ်ပါတယ်။ SOA မှာ အစိတ်အပိုင်း (၃)ရပ်၊ ပါဝင်လေ့ရှိပါတယ်။ Service Provider (Server), Service Consumer (Client) နဲ့ Interface (Protocol) တို့ပဲဖြစ်ပါတယ်။

Service Provide သို့မဟုတ် Server ဆိုတာဟာ သူရဲ့လုပ်ဆောင်ချက်တွေကို အခြားပရိုကရပ်တစ်ခုကနေ ရယူ အသုံးပြု ခွင့်ပေးထားတဲ့ ပရိုကရမဲ့တစ်မျိုးဖြစ်ပါတယ်။ Service Consumer သို့မဟုတ် Client ဆိုတာကတော့ Server ပရိုကရမဲ့က ပေးထားတဲ့ လုပ်ဆောင်ချက်တွေကို ရယူအသုံးပြုထားတဲ့ ပရိုကရမဲ့တစ်မျိုးဖြစ်ပါတယ်။ SOA နည်းစနစ်မှာပါဝင်တဲ့ Service တွေမှာ တစ်ချို့ Service တွေက Server ဖြစ်ပြီး တစ်ချို့ Service တွေ ကတော့ Client တွေဖြစ်ကြပါတယ်။ Service တစ်ခုဟာ Server အနေနဲ့ ရော Client အနေနဲ့ပါ အလုပ်လုပ်မယ် ဆိုရင်လည်း လုပ်နိုင်ပါတယ်။ Interface သို့မဟုတ် Protocol ဆိုတာကတော့ Service တွေ အချင်းချင်း အပြန်အလှန် ဆက်သွယ်ရမှာ စံထားပြီးလိုက်ရမယ့် Communication Standard တစ်မျိုးပဲဖြစ်ပါတယ်။ Server တွေ Client တွေ ဟာ သတ်မှတ်ထားတဲ့ Interface ကို အသုံးပြုခြင်းအားဖြင့် အပြန်အလှန် ဆက်သွယ်နိုင်က ခြင်းပဲ ဖြစ်ပါတယ်။



ပုံ (၄.J) - Service Oriented Architecture

#### 4.1 – Service Layers

Software Solution ကို SOA နည်းစနစ်အတိုင်း Service များအဖြစ် ခွဲပြီးတည်ဆောက်တော့မယ်ဆိုရင် ဘယ်လို အပိုင်း၊ ဘယ်လို လုပ်ဆောင်ချက်တွေကို သီးခြား Service အဖြစ် တည်ဆောက်သင့်သလဲဆိုတဲ့အချက်က စဉ်းစားစရာဖြစ်လာ ပါတယ်။ အခြေခံအားဖြင့် Service Layer သုံးဆင့်ရှိတယ်လို့ ဆိုနိုင်ပါတယ်။ ပုံ (၄.J) မှာ လည်း လေ့လာကြည့်နိုင် ပါတယ်။

## Utility Services

အခြေခံအကျခုံး Service Layer ကတော့ **Utility Service** များဖြစ်ပါတယ်။ ဒီအဆင့်မှာ Database နဲ့ ဆက်သွယ် အလုပ်လုပ်ပေးနိုင်တဲ့ Service တွေ၊ File Read/Write အပါအဝင် File System ပိုင်းဆိုင်ရာအလုပ်တွေ လုပ်ပေးနိုင်တဲ့ Service တွေ၊ User Action တွေ Event တွေကို Log လုပ်ပေးနိုင်တဲ့ Logging Service တွေ၊ ကွန်ပျူးတာ Hardware နဲ့ Resource တွေကို စီမံပေးနိုင်တဲ့ Service တွေ၊ Data Format တွေ အပြန်အလုန် ပြောင်းပေးနိုင်တဲ့ Encoding Service တွေနဲ့ အခြားအခြေခံလုပ်ဆောင်ချက်ပိုင်းဆိုင်ရာ Service တွေ ပါဝင်လေ့ရှုပါတယ်။

Utility Service တွေဟာ အများအားဖြင့် Server အနေနဲ့အလုပ်လုပ်ကြဖြီး အခြေခံအားဖြင့် Application Agnostic ဖြစ်ကြပါတယ်။ Application Agnostic ဆိုတာကတော့ Project တစ်ခုအတွက်တင်မကပဲ အခြား Project များ၊ အခြား Application များမှာပါ ပြန်လည်အသုံးပြုနိုင်တဲ့ အခြေခံ Foundation Service များပဲဖြစ်ကြပါတယ်။

## Core Services

တည်ဆောက်လိုတဲ့ Software Solution ရဲ့ Common Logic သို့မဟုတ် Key Logic အဖြစ်အလုပ်လုပ်မယ့် Service တွေကတော့ Core Service များပဲဖြစ်ပါတယ်။ ဥပမာ – Customer Management Service, Supplier Management Service, Stock Management Service, Order Management Service, Payment Service စသေဖြင့် Solution လိုအပ်ချက်အလိုက် တည်ဆောက်ထားရတဲ့ Service များပဲဖြစ်ပါတယ်။

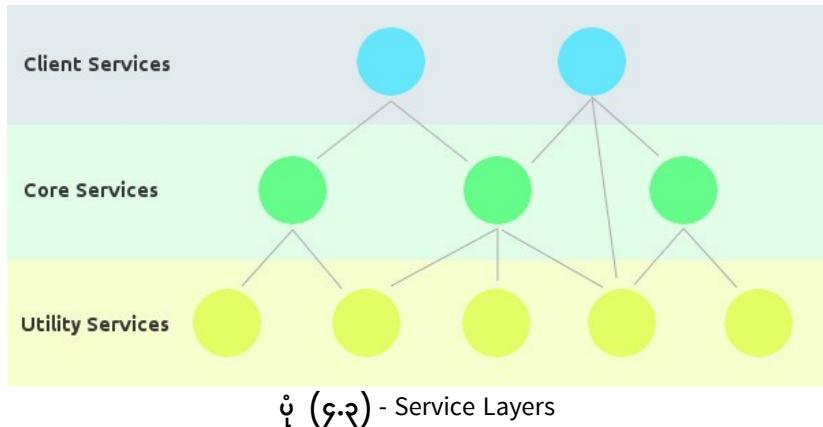
Core Service တွေဟာ Utility Service တွေလိုတော့ Application Agnostic ဖြစ်မှာမဟုတ်တော့ပါဘူး။ သက်ဆိုင်ရာ Solution အတွက်ရည်ရွယ်တည်ဆောက်ထားတဲ့ Service များဖြစ်တဲ့အတွက် သက်ဆိုင်ရာ Solution အတွက်သာ အသုံးဝင်တဲ့ Service များပဲဖြစ်ပါတယ်။ Core Service တွေဟာ Client အနေနဲ့ရော၊ Server အနေနဲ့ပါ အလုပ်လုပ်လေ့ရှုကြပါတယ်။

ဥပမာ – Core Service တစ်ခုဖြစ်တဲ့ Customer Management Service ဟာ Customer တွေရဲ့ အချက်အလက် တွေကို သိမ်းဆည်းခြင်းလုပ်ငန်းအတွက် Utility Service ဖြစ်တဲ့ Database Service ရဲ့လုပ်ဆောင်ချက်တွေကို ရယူ အသုံးပြုဖို့လိုနိုင်ပါတယ်။ တစ်ချိန်တည်းမှာပဲ၊ အခြား Service တွေဖြစ်ကြတဲ့ Report Service, Account Service စတဲ့ Client Service တွေအတွက် လိုအပ်တဲ့လုပ်ဆောင်ချက်တွေကိုလည်း ပုံပိုးပေးနိုင်ပါတယ်။ ဒီတော့ Core Service တွေ ဟာ Utility Service များနဲ့ Client Service များရဲ့ ကြားခံနေရာကနေ အလုပ်လုပ်ပေးတဲ့ Service များလိုလည်း ဆိုနိုင် ပါတယ်။

## Client Services

Services တွေထဲမှာ အသုံးပြုသူ User နဲ့ တိုက်ရှိက်ထိတွေဖို့လိုတဲ့ Service တွေကတော့ Client Service များပဲ ဖြစ်ပါတယ်။ လုပ်ဆောင်ချက်အားဖြင့် လိုအပ်သလို ပြန်လည်ရယူအသုံးပြုနိုင်ဖို့ထက် လုပ်ငန်းတစ်ခုတည်းကို တိတကျကျ ဆောင်ရွက်နိုင်ဖို့ ရည်ရွယ်ဖန်တီးထားတဲ့ Service များဖြစ်ကြပါတယ်။ ဥပမာ – Invoice Service တစ်ခုဟာ Invoice Document တွေ တည်ဆောက်ပေးတဲ့ လုပ်ငန်းတစ်ခုတည်းကိုသာလုပ်ပေးတဲ့ Client Service ဖြစ်နိုင်ပါတယ်။ အဲဒီ Invoice Service အလုပ်လုပ်နိုင်ဖို့အတွက် Core Service တစ်ခုဖြစ်တဲ့ Payment

Service ကပေးထားတဲ့ Tax Calculation, Currency Conversion စတဲ့ လုပ်ဆောင်ချက်တွေကို ရယူအသုံးပြုဖို့ လိုနိုင်ပါတယ်။ ဒုအပြင် Invoice File တွေကို သိမ်းဆည်းဖို့အတွက် Utility Service တစ်ခုဖြစ်တဲ့ File System Service ရဲ့ လုပ်ဆောင်ချက်တွေကိုလည်း ရယူ အသုံးပြုဖို့လိုနိုင်ပါသေးတယ်။



SOA Solution တွေမှာ ရှိတက်လေ့ရှိတဲ့ Service Layers တွေကို ဥပမာ အနေနဲ့ဖော်ပြခြင်းသာဖြစ်ပါတယ်။ SOA ဟာ Architecture Design Concept တစ်ခုသာဖြစ်လို့ လက်တွေ့အသုံးချရမှာတော့ မိမိလိုအပ်ချက်နဲ့သင့်တော် သလို အသုံးချနိုင်ပါတယ်။ တည်ဆောက်လိုတဲ့ Solution အမျိုးအစားနဲ့ Requirement Specification တွေ ကွဲပြားသွားတာနဲ့အမျှ SOA Solution တစ်ခုမှာ ပါဝင်ဖွံ့စည်းတက်တဲ့ Service Layer တွေ Service အမျိုးအစား တွေ၊ ကွဲပြားသွားမှာဖြစ် ပါတယ်။

## 4.2 – Service Types

SOA Solution တစ်ခုကို Design လုပ်ရာမှာ အထောက်အကူဖြစ်ဖော်နဲ့ အိုင်ဒီယာရဖော်ဖို့အတွက် ရှိတက်တဲ့ Service အမျိုးအစားများတွေကိုလည်း ဖော်ပြပေးလိုက်ပါတယ်။

**User Interface Services** – User Interface Service တွေဟာအသုံးပြုသူ User နဲ့ တိုက်ရှိက်ထိတွေရတဲ့ Service အမျိုးအစားများဖြစ်ပြီး အသုံးပြုသူပေးတဲ့ Input Data တွေကို လက်ခံရယူခြင်း၊ အသုံးပြုသူရဲ့ Request သို့မဟုတ် Event တွေကို တုန်းပြန်ခြင်းစတဲ့ လုပ်ငန်းများကို ဆောင်ရွက်ပေးတဲ့ Service များပဲ ဖြစ်ပါတယ်။ Service Layer အရ ဆိုရင် Client Layer မှာပါဝင်တဲ့ Service များဖြစ်ကြပါတယ်။ အများအားဖြင့် အခြား Service များကပေးတဲ့ လုပ်ဆောင်ချက်များကိုရယူပြီး၊ အသုံးပြုသူ User နားလည်ထိတွေနိုင်တဲ့ အသွင်အပြင် ဖြစ်အောင် User Interface တစ်ခုနဲ့ဖော်ပြပေးတဲ့ Service များဖြစ်ကြပါတယ်။

**Policy Services** – User Authentication, Authorization နဲ့ Role Management တို့လို လုပ်ဆောင်ချက်တွေ ဟာ Policy Service တွေဖြစ်ပါတယ်။ Access Right ကန့်သတ်တားတဲ့ Service တွေကို ဆက်သွယ်ဖို့အတွက် User ID, Password စတဲ့ အချက်အလက်တွေကို ပေးပို့ရမယ့်နည်းလမ်းတွေ၊ User Role အလိုက် ခွင့်ပြထားတဲ့ လုပ်ဆောင်ချက် တွေကိုလည်း Policy Service တွေမှာ ထည့်သွုံးသတ်မှတ်ထားနိုင်ပါတယ်။

**Process Services** – Process Service တွေကိုတော့ Controller Service လိုလည်း ခေါ်နိုင်ပါတယ်။ ဥပမာ – UI Service တစ်ခုက Data Service တစ်ခုထံက အချက်အလက်ကိုတောင်းခံတဲ့အခါ Process Service ကနေ တစ်ဆင့် တောင်းခံဖို့ လိုနိုင်ပါတယ်။ ဒါတော့မှ Process Service က Policy Service မှာ သတ်မှတ်ထားတဲ့ သတ်မှတ်ချက်များနဲ့ တိုက်ခိုင်စစ်ဆေးပြီး Policy နဲ့ကိုက်ညီရင် လက်ဆင့်ကမ်းဆက်သွယ်ပေးပြီး၊ Policy နဲ့မ ကိုက်ညီရင်တော့ လက်ဆင့် ကမ်းဆက်သွယ်ပေးဖို့ ငြင်းပယ်သွားနိုင်များဖြစ်ပါတယ်။ ဒါလို Process Service ကြားခံမထားပဲ UI Service က Policy ကို တိုက်ရိုက်ဆက်သွယ်ပြီး ဆုံးဖြတ်လိုလည်း ရနိုင်ပါတယ်။ ဒါပေမယ့် နောင်မှာ Policy ပြောင်းသွားတဲ့အခါ UI Service ကိုလည်း လိုက်ပြင်ရတက်လို့ ပြုပြင်ထိမ်းသိမ်းရ ပိုမိုလွယ်ကူအောင် ကြားခံ Process Service များ အသုံးပြုကြခြင်းဖြစ်ပါတယ်။

**Connector Services** – SOA Solution တစ်ခုမှာပါဝင်တဲ့ Service တွေက ကွန်ပူးတာတစ်လုံးမှာ အမြှိုနေမှာ မဟုတ်ပါဘူး။ မတူကဲပြားတဲ့ Location နဲ့ Network တွေမှာ တည်ရှိနေနိုင်တဲ့ Service တွေဖြစ်တဲ့ အတွက် Connector Service တွေက အသုံးပြုလိုတဲ့ Service တည်ရှိရာ Location ကို Resolve လုပ်ပေးခြင်း၊ ချိတ်ဆက်ဆက်သွယ်ပေးခြင်း လုပ်ငန်းများကို ဆောင်ရွက်ပေးပါတယ်။ Connector Service များရဲ့အကူအညီနဲ့ Service Location ကို လွပ်လပ်စွာ ပြောင်းနိုင်တဲ့အတွက် ပိုပြီး Flexible ဖြစ်တဲ့ Service Infrastructure ကို ရရှိနိုင်မှာပဲဖြစ်ပါတယ်။

**Infrastructure Service** – Infrastructure Service တွေကတော့ Application Specific မဟုတ်ပဲ ပြန်လည် အသုံး ပြုနိုင်တဲ့ Utility Service အမျိုးအစားများဖြစ်ပါတယ်။ ဘယ် Service တွေကို CPU Process ဘယ်လောက်ထိ ခွင့်ပြုမလဲ၊ RAM ပမာဏ ဘယ်လောက်ခွင့်ပြုမလဲ စတဲ့ Service တွေရဲ့ Resource အသုံးပြုမှု ကို စီမံပေးတဲ့ Service တွေ၊ Performance ပိုကောင်းလာဖော်အတွက် Cache လုပ်ဆောင်ချက်ကို ဆောင်ရွက်ပေးနိုင်တဲ့ Service တွေနဲ့ များပြား လာတဲ့အချက်အလက်တွေကို လိုအပ်သလို Archive လုပ်ပေးနိုင်တဲ့ Service မျိုးတွေဟာလည်း Infrastructure Service များထဲမှာပါဝင်ပါတယ်။

**Data Access Services** – File System I/O Service များနဲ့ Database Service များကို Data Access Service လို ခေါ်နိုင်ပါတယ်။ File Read/Write လုပ်ငန်းများ၊ File Format ဆိုင်ရာလုပ်ငန်းများ၊ Data Encode/Decode လုပ်ငန်းများ၊ Data Model တည်ဆောက်ခြင်းလုပ်ငန်းများ၊ Data Relationship နဲ့ Index များစီမံခြင်းလုပ်ငန်းများ စတဲ့ File နဲ့ Data တွေကို စီမံပေးနိုင်တဲ့ Utility Service များဟာ Data Access Service အမျိုးအစားထဲမှာ ပါဝင်ပါတယ်။

**Information Services** – Information Service တွေကတော့ သတင်းအချက်အလက်များကို ရယူပေးခြင်း၊ ဖြန့်ဝေပေးခြင်းလုပ်ငန်းများကို ဆောင်ရွက်ပေးတဲ့ Service များဖြစ်ပါတယ်။ အဲဒီလိုရယူဖြန့်ဝေရာမှာ သက်ဆိုင်ရာ ကဏ္ဍအလိုက် ပါဝင်သင့်တဲ့ သတင်းအချက်အလက်၊ ဖြစ်သင့်တဲ့ သတင်းအချက်အလက်ဖွဲ့စည်းပုံစံတဲ့ Logic တွေကို ဆုံးဖြတ်ပေးတဲ့ Service ပဲဖြစ်ပါတယ်။ Reporting Service တွေဟာလည်း Information Service ထဲမှာ ပါဝင်ပါတယ်။ တနည်း အားဖြင့် အချက်အလက်ဖွဲ့စည်းပုံ Information Structure ကို သတ်မှတ်ပေးထားတဲ့ Service အမျိုးအစားဖြစ်ပါတယ်။

**Monitor Services** – Monitor Service တွကတော့ အခြား Service များ တစ်ခုနဲ့တစ်ခု ဆက်သွယ်အလုပ်လုပ်မှု မှတ်တမ်းကို စီစစ်ပေးတဲ့ Service များဖြစ်ပါတယ်။ Monitor Service များရဲ့ အကူအညီနဲ့ ဘယ်လို Service တွက အမိကအလုပ်လုပ်ရသလဲဆိုတဲ့ အချက်အလက်တွေ၊ ဘယ်လို Service တွက လုပ်ငန်းဆောင်ရွက်ရမှာ Bottleneck ဖြစ်နေစေသလဲဆိုတဲ့ အချက်အလက်တွေနဲ့ ဘယ်လိုအချိန်တွေမှာ ဆက်သွယ်ဆောင်ရွက်ရမှု များတက်သလဲဆိုတဲ့ အချက်အလက်တွေကို လေ့လာသိရှိရနိုင်စေမှာဖြစ်ပါတယ်။ ဒီနည်းနဲ့စနစ်လုပ်ဆောင်ချက် ပိုကောင်းလာဖို့ ဘယ်လို Service တွေ ဖြည့်စွက်သင့်သလဲ၊ ဘယ် Service တွေကို Improve လုပ်သင့်သလဲဆိုတဲ့ အချက်အလက်တွေကို ရရှိနိုင်မှာဖြစ်ပါတယ်။ တစ်နည်းအားဖြင့် Monitor Service ဆိုတာ Solution ကလိုအပ်တဲ့ လုပ်ဆောင်ချက်ကို ဆောင်ရွက်ပေးခြင်းမဟုတ်ပဲ Service တွေကို ပြင်ဆင်ထိမ်းသိမ်းရမှာ အထောက်အကူဖြစ်စေဖို့ ရည်ရွယ်ထားတဲ့ Service များပဲဖြစ်ပါတယ်။ လုပ်ဆောင်ချက် အဆင့်တိုင်းကို မှတ်တမ်းတင်ပေးတဲ့ Logging Service တွေဟာလည်း Monitor Service ထဲမှာ ပါဝင်ပါတယ်။

**Development Services** – Service တွေ တစ်ခုနဲ့တစ်ခု ဆက်သွယ်ဆောင်ရွက်ပုံကို Simulate လုပ်ပေးနိုင်တဲ့ Service တွေ၊ Service တစ်ခုရဲ့ လုပ်ဆောင်ချက်ရလဒ်နဲ့ အခြား အထွေထွေအချက်အလက် အသေးစိတ်များကို ဖော်ပြပေးနိုင် တဲ့ Debug Service တွေ၊ Service တွေ မျှော်မှန်းထားတဲ့ အတိုင်း အလုပ်လုပ်မလုပ် စမ်းသပ်ပေးနိုင်တဲ့ Test Service တွေ၊ Service စာရင်းနဲ့ အသုံးပြုပဲ Implementation Detail ကို ဖော်ပြပေးနိုင်တဲ့ Directory Service တွေဟာ Development Service တွေပဲဖြစ်ပါတယ်။ Service အသစ်တစ်ခုဖန်တီးပြီးတဲ့ အခါမှာ အလိုအလျောက် Directory ထဲမှာ ထည့်သွင်းပေးခြင်း၊ စမ်းသပ် ပေးသွားခြင်းနဲ့ Policy Service ထဲမှာ မှတ်တမ်းတင်ပေးခြင်းတို့လို လုပ်ငန်းတွေကို အလိုအလျောက် ဆောင်ရွက်ပေးနိုင်တဲ့ Publishing Service တွေဟာလည်း Development Service တွေပဲဖြစ်ပါတယ်။ တနည်းအားဖြင့် Development Service တွေဟာ Solution ကလိုအပ်တဲ့ လုပ်ဆောင်ချက်ကို ဆောက်ရွက်ပေးခြင်း မဟုတ်ပဲ၊ အခြား Service များ တည်ဆောက်ရမှာ အထောက်အကူဖြစ်စေဖို့ ရည်ရွယ်ထားတဲ့ Development Tool များပဲဖြစ်ပါတယ်။

ပြောခဲ့တဲ့ Service အမျိုးအစားတွေထဲမှာ မပါဝင်ပဲ တိကျတဲ့ လုပ်ဆောင်ချက်တစ်ခု ရရှိဖို့အတွက် ရည်ရွယ်ဖန်တီးထားတဲ့ Service တွေကိုတော့ **Application Service** လို ခေါ်နိုင်ပါတယ်။ Application Service တွေရဲ့ သဘောသဘာဝနဲ့ အလုပ် လုပ်ပုံကတော့ တည်ဆောက်လိုတဲ့ Solution ပေါ်မှုတည်ပြီး ကွဲပြားသွားမှာပဲဖြစ်ပါတယ်။ Business Solution တစ်ခုမှာဆိုရင် Customer, Supplier, Client, Payment, Order, Stock တို့လို Service တွေ ပါဝင်နိုင်ပြီး Social Collaboration Solution တစ်ခုမှာဆိုရင်တော့ Real-time Messaging, Group Messaging, Contact List စတဲ့ Service တွေပါဝင်မှာဖြစ်ပါတယ်။ ပုံသေပြောလိုမရတော့ပဲ Requirement ပေါ်မှတည် တည်ဆောက်ရတဲ့ Service များပဲ ဖြစ်ပါတယ်။

### 4.3 – Why SOA

Software Solution တစ်ခုကို SOA နည်းစနစ်နဲ့ တည်ဆောက်လိုက်ခြင်းအားဖြင့် ရရှိလာမယ့်အကျိုးကျေးဇူးတွေ အများကြီးရှိပါတယ်။ အချို့ Service တွေဟာ လက်ရှိ Project အတွင်းမှာ Reusable ဖြစ်ယုံသာမက၊ အခြား Project တွေမှာလည်း ပြန်လည်အသုံးပြုနိုင်မှာပါ။ အထူးသဖြင့် Data Access Service တွေ၊ Infrastructure Service တွေနဲ့ Development Service တွေအပါအဝင် Utility Layer မှာရှိတဲ့ Service အများစုံ မတူဘွဲ့ပြားတဲ့ Software Project တွေမှာ ပြန်လည်အသုံးပြုနိုင်မှာပဲဖြစ်ပါတယ်။

## One Cores, Multiple Clients

တစ်ချိန်က Windows, Linux, Mac အစရှိတဲ့ Desktop Platform အမျိုးမျိုးမှာ အလုပ်လုပ်နိုင်တဲ့ Cross-platform App တွေ တည်ဆောက်ရခြင်းဟာ စိန်ခေါ်မှုတစ်ရပ် ဖြစ်ခဲ့ပါတယ်။ ကနောက်မှာ Software ဆိတာကို Desktop ကွန်ပျူတာနဲ့သာမက၊ Tablet, Smart Phone အစရှိတဲ့ Mobile Device တွေကနောက်လည်း တွင်ကျယ်စွာ အသုံးပြုလာ ကြပြီဖြစ်ပါတယ်။ တစ်ချိန်မှာ Smart Watch, Smart Glass စတဲ့ Wearable Device တွေနဲ့ အခြား အခြားသော Device တွေလည်း အသုံးပြုလာကြနိုင်ပါတယ်။ ဒီလို Device တွေ အမျိုးမျိုးဖြစ်လာတဲ့အတွက် တစ်ချိန်က Cross-platform ဆိုတဲ့ ပြဿနာအတွက်သာ ခေါင်းစားခဲ့ကြရာက၊ ယနေ့အချိန်မှာတော့ Cross-device ဆိုတဲ့ ပြဿနာက လည်း ခေါင်းစားစရာဖြစ်လာပါတယ်။

ဒီပြဿနာကိုဖြေရှင်းပေးနိုင်တဲ့ နည်းပညာတွေနဲ့ နည်းစနစ်တွေ အမျိုးမျိုးရှိရာမှာ SOA နည်းစနစ်ဟာ ထိရောက်မှုအရှိ ဆုံးနည်း စနစ်တစ်ခုပဲဖြစ်ပါတယ်။ Software ၏ အစိတ်အပိုင်းတွေကို သီးခြားစီအလုပ်လုပ်နိုင်တဲ့ ပရိုဂရမ်းယ်များအဖြစ် တည်ဆောက်ထားခြင်းအားဖြင့် Utility Layer နဲ့ Core Layer မှာရှိနေတဲ့ အခြေခံလုပ်ဆောင်ချက်နဲ့ Key Service တွေကို တစ်ကြိမ်သာ ဖန်တီးဖို့လိုပြီး၊ UI Service တွေကိုသာ Device အမျိုးမျိုးနဲ့ Platform အမျိုးမျိုးအတွက် ခွဲခြား တည်ဆောက်ပေးရတော့မှာဖြစ်ပါတယ်။ ဒီနည်းနဲ့ မတူကွဲပြားတဲ့ Device တိုင်းအတွက် App တစ်ခုစီ အစအဆုံးပြန်လည် ဖန်တီးနေစရာ မလိုတော့တဲ့ အားသာချက်ကို ရရှိမှာပဲဖြစ်ပါတယ်။

Device အမျိုးမျိုးအတွက် UI Service တွေနဲ့ Core Service တွေ၊ အပြန်အလုန်ဆက်သွယ်ဖို့အတွက် HTTP နဲ့ SOAP တို့လို Communication Protocol တွေကို Interface အဖြစ်အသုံးပြုကြပါတယ်။ ဒီအကြောင်းကို မကြေခင် ဆက်လက်ဖော်ပြပါမယ်။

## Development Productivity

SOA နည်းစနစ်ကပေးတဲ့ နောက်ထပ်အားသာချက်တော့ Development Productivity ပဲဖြစ်ပါတယ်။ Software Solution ကို သီးခြား ပရိုဂရမ်းယ်များအဖြစ် ခွဲခြားတည်ဆောက်ခြင်းဖြစ်တဲ့အတွက် သီးခြားစီခွဲခြားစမ်းသပ်နိုင်မှာ ဖြစ်ပါတယ်။ ဒါအပြင် သီးခြားစီ ပြုပြင်ထိမ်းသိမ်းနိုင်တဲ့အတွက် Solution တစ်ခုလုံးကို ထိမ်းသိမ်းရခြင်းထက် ပိုမိုရှိရှင်း ထိရောက်နိုင်မှာပဲဖြစ်ပါတယ်။

အဖွဲ့လိုက် ဆောင်ရွက်ရာမှာလည်း Developer တွေကို တာဝန်ခွဲအပ်ရတာ ပိုမိုထိရောက်နိုင်ပါသေးတယ်။ သီးခြား Service တွေတည်ဆောက်ဖို့ သီးခြားစီ တာဝန်ပေးထားတဲ့ Developer တွေဟာ မိမိတို့တာဝန်ကျရာ Service ကို နှစ်သက်ရာ နည်းပညာ၊ ကျမ်းကျင်ရာနည်းစနစ်တွေ အသုံးပြုဖန်တီးနိုင်မှာဖြစ်လို့ ရေးသားဖန်တီးရာမှာ ပိုမိုထိရောက် လျှင်မြန်စေမှာ ဖြစ်ပါတယ်။ Service အချင်းချင်း ဆက်သွယ်ဖို့ သတ်မှတ်ထားတဲ့ Interface စံ ညီနေသူး အခြား Service မှာ တာဝန်ကျသူး Developer တွေ အသုံးပြုထားတဲ့ နည်းပညာတွေ၊ နည်းစနစ်တွေနဲ့ လိုက်ညိုနေဖို့ မလိုတော့ပါဘူး။

မိမိတို့ နှစ်သက်ကျမ်းကျင်ရာနည်းပညာ ဆိုတဲ့နေရာမှာ လုံးဝမတူကွဲပြားတဲ့ Language တွေ Platform တွေပါဖြစ်နိုင်ပါသေးတယ်။ ဥပမာ – Infrastructure Service တွေကို အကောင်းဆုံး Performance ရရှိစေဖို့အတွက် C/C++, Go တို့လို လျှင်မြန်တဲ့ Language တွေကို အသုံးပြုနိုင်ပြီး၊ Monitor Service တွေကို Real-time

Concurrency လုပ်ဆောင်ချက် ပိုမိုကောင်းမွန်တဲ့ NodeJS လို နည်းပညာနဲ့ တည်ဆောက်ထားနိုင်ပါတယ်။ Application Service တွေကိုတော့ အသင့်သုံး Business Component တွေ စုံလင်တဲ့ Java, .Net တို့လို နည်းပညာမျိုးတဲ့ တည်ဆောက်နိုင်ပါတယ်။ ဒီနည်းနဲ့ Developer တော့ဟာ မိမိတို့ အပိုင်နိုင်ဆုံးနည်းပညာကို ခွဲခြား အသုံးပြန်လို Development Productivity ပိုကောင်းလာတဲ့အပြင် သက်ဆိုင်ရာကဏ္ဍအတွက် အထိရောက်ဆုံးနည်းပညာကို ရွေးချယ်အသုံးပြန်လို ပိုမိုကောင်းမွန်တဲ့ စွမ်းဆောင်ရည် Efficiency ကိုလည်း ရရှိနိုင်မှာပဲဖြစ်ပါတယ်။

## Scalability

SOA နည်းစနစ်နဲ့ ခွဲ့ခြားတည်ဆောက်ထားတဲ့ Service တွေကို တစ်လုံးထက်ပို့တဲ့ ကွန်ပျုံတာတွေမှာ ခွဲ့ခြားပြီး Deploy လုပ်ထားနိုင်ယုံသာမက၊ လိုအပ်ရင် Network အမျိုးမျိုးနဲ့ တည်နေရာအမျိုးမျိုးမှာလည်း ခွဲ့ခြားထားနိုင်ပါသေးတယ်။ ဥပမာ - Data Access Service တွေက တစ်နေရာ၊ Application Service တွေက တစ်နေရာ၊ Monitor Service တွေက တစ်နေရာ စသဖြင့် ခွဲ့ခြားထားနိုင်ပါတယ်။ အလုပ်များများလုပ်ရမယ့် Monitor Service အတွက် Processing Power ပိုကောင်းတဲ့ ကွန်ပျုံတာတွေသုံးပြီး Data Store အခိုကဖြစ်တဲ့ Data Access Service တွေအတွက် Storage Capacity ပိုများတဲ့ ကွန်ပျုံတာတွေကို အသုံးပြုနိုင်ပါတယ်။ ဒီနည်းနဲ့ Service တစ်ခုခြင်းအတွက် အထိရောက်ဆုံးဖြစ် အောင် သီးသန် Setting လုပ်ထားတဲ့ Hardware Infrastructure ကို ရရှိနိုင်မှာဖြစ်ပြီး၊ လိုအပ်လို နောင်တစ်ခါန်မှာ အဆင့်မြှင့်တင်ရမယ်ဆိုရင်လည်း လိုအပ်တဲ့ Service တစ်ခု ခြင်းစီအတွက် ရွေးချယ်တိုးဖြင့်သွားနိုင်မှာဖြစ်တဲ့ အတွက် ပိုမို ထိရောက်စေမှာဖြစ်ပါတယ်။

## Service Oriented Business Model

Software Product တစ်ခုပေါ်မှာ အခြေခံထားတဲ့ Business Model အမျိုးမျိုး ရှိနိုင်ပါတယ်။ SOA နည်းစနစ်နဲ့ တည် ဆောက်ထားတဲ့ Solution တစ်ခုမှာတော့ Business Model ကို နောက်ဆုံးရလဒ်ဖြစ်တဲ့ Product ပေါ်မှာ တင်မကပဲ Service တွေ ပေါ်မှာအခြေခြားတဲ့ Business Model တွေလည်း တည်ဆောက်ရရှိနိုင်ပါတယ်။ Service တွေကို Interface ကနေတစ်ဆင့် ဆက်သွယ်နိုင်အောင် တည်ဆောက်ရတဲ့ အတွက် အဲဒီ Service တွေ ကို ကိုယ့်ရဲ့ Solution မှာ ပါဝင်တဲ့ အခြား Service တွေကသာမက၊ ပြင်ပကနေလည်း ဆက်သွယ်အသုံးပြုခွင့် ပေးနိုင်ပါတယ်။ ဒီနည်းနဲ့ အလားတူ လုပ်ဆောင်ချက် ရရှိလိုဘူးတွေအနေနဲ့ ကိုယ်တိုင်ဖန်တီးနေဖို့မလိုတော့ပဲ ကုန်တော်တို့တည်ဆောက်ထားတဲ့ Service ကနေတစ်ဆင့် အသင့်အသုံးပြုနိုင်မှာဖြစ်ပါတယ်။

ကနောက်ချိန်မှာ Service ကို အခြေခံထားတဲ့ Business Model တွေကို အတော်လေး တွင်တွင်ကျယ်ကျယ် အသုံးပြုနေ ကြပါပြီ။ တစ်ချို့လည်း မိမိတို့ရဲ့ Service တွေကို အခမဲ့ ဆက်သွယ်အသုံးပြုခွင့် ပေးကြပြီး၊ တစ်ချို့ လည်း အသုံးပြုတဲ့ အကြမ်အရေအတွက်အလိုက် အခကြေးငွေကောက်ခံကြပါတယ်။ တစ်ချို့လည်းအသုံးပြုခွင့် ကို လစဉ်ကြေး၊ နှစ်စဉ်ကြေး အနေနဲ့ ကောက်ခံခြင်းအားဖြင့် သူတို့ရဲ့ Business Model ကို Service များပေါ်မှာ တည်ဆောက်ထားတက်ကြပါတယ်။

ဒီလို Service Oriented Business Model တွေပေါ်ပေါက်လာဖို့အတွက် အမိကကျတဲ့အခန်းကဏ္ဍကနေ ပါဝင်လာတာ ကတေသာ Web Technology ပဲဖြစ်ပါတယ်။

## 4.4 – Web Services

Service တွေဟာ တစ်ခုနဲ့တစ်ခု ဆက်သွယ်ဖို့အတွက် ကြားခံဆက်သွယ်ရေး Interface လိုပါတယ်။ SOAP, XML-RPC, REST စသဖြင့် အသုံးများတဲ့ Interface အမိုးမျိုးရှိပါတယ်။ ဒါ Interface တွေမှာ တူညီတဲ့အချက်တစ်ချက် ရှိပါတယ်။ Interface အားလုံးက HTTP ကို အခြေခံဆက်သွယ်ရေး Protocol အဖြစ် အသုံးပြုကြခြင်းပဲဖြစ်ပါတယ်။ HTTP ဟာ Web Technology တစ်ခုဖြစ်တဲ့အတွက် HTTP ကို အခြေခံ ဆက်သွယ်ရေး Interface အဖြစ် အသုံးပြုထား တဲ့ Service တွေကို Web Service လို့ ခေါ်ခြင်းပဲဖြစ်ပါတယ်။ တစ်နည်းအားဖြင့် ကနောက်မှာ Service အများစုံဟာ HTTP ကို ဆက်သွယ်ရေး Interface အဖြစ် အသုံးပြုထားတဲ့အတွက် "Service အများစုံ ဟာ Web Service များ ဖြစ်တယ်" လိုလည်း ဆိုနိုင်ပါတယ်။

Web Service တွေ တည်ဆောက်ရာမှာ အမိကသတိပြုရမယ့် နည်းပညာနှစ်ရပ်ရှိပါတယ်။ အဲဒါတွေကတော့ REST နဲ့ JSON တို့ပဲဖြစ်ပါတယ်။

### Representational State Transfer (REST)

REST လို့ အတိကောက်ခေါ်တဲ့ Representational State Transfer ဆိုတာဟာ ဆက်သွယ်ရေး Interface တွေ ဖန်တီးရာမှာ လိုက်နာဖို့ သတ်မှတ်ထားတဲ့ နည်းပညာသတ်မှတ်ချက်များ ဖြစ်ပါတယ်။ REST နည်းစနစ်ကို World Wide Web Consortium က HTTP ရဲ့ အစိတ်အပိုင်းတစ်ရပ်အဖြစ် တွဲဖက်တိတွင်ထားခြင်းဖြစ်ပါတယ်။ တစ်နည်းအားဖြင့် HTTP ဟာ REST နည်းစနစ်နဲ့အညီ ဖန်တီးထားတဲ့ နည်းပညာဖြစ်တဲ့အတွက် HTTP ကို ဆက်သွယ်ရေး Interface အဖြစ် အသုံးပြုထားတယ်ဆိုရင် REST နည်းစနစ်ကို အသုံးပြုနေခြင်းဖြစ်တယ်လို့ လည်း ဆိုနိုင်ပါတယ်။ HTTP အကြောင်းကိုတော့ ဒီနေရာမှာ အသေးစိတ်မပြောတော့ပါဘူး။ Professional Web Developer မှာ ဖော်ပြုခြုံပြီးဖြစ်လို့ လေ့လာလို့ တယ်ဆိုရင် Professional Web Developer ရဲ့ အခန်း(၁) နဲ့ အခန်း(၁၇) တို့မှာ လေ့လာနိုင်ပါတယ်။

REST နည်းစနစ်က သတ်မှတ်ထားတဲ့ သတ်မှတ်ချက်များထဲမှ ထူးခြားချက်တစ်ချို့ကို ရွေးထုတ်ဖော်ပြလိုက်ပါတယ်။

**Stateless** – Service တစ်ခုနဲ့တစ်ခု ဆက်သွယ်ရာမှာ ပါဝင်ရမယ့် အချက်အလက်များကို သိမ်းဆည်းမထားပဲ ဆက်သွယ်မှုလုပ်တိုင်းမှာ ပါဝင်ရမယ့်အချက်အလက်အားလုံးကို ထည့်သွင်းပေးပို့ခြင်းအားဖြင့် ဆက်သွယ်ရမယ်လို့ဆိုပါတယ်။ ဒီသဘောသဘာဝကို Stateless လို့ခေါ်ပြီး HTTP ဟာ Stateless Protocol တစ်ခု ဖြစ်ပါတယ်။ ဒါ Stateless ဆိုတဲ့ အချက်ကို ထည့်သွင်းစဉ်းစားခြင်းအားဖြင့် Service တွေတည်ဆောက်ရာမှာ များစွာ ရှိုးရှင်းလွယ်ကူသွားစေနိုင်ပါ တယ်။ ဥပမာ – ဆက်သွယ်မှုအကြိမ်တိုင်းမှာ ပါဝင်ရမယ့် အချက်အလက်တွေကို ပေးပို့မှုဖြစ်လို့ ဆက်သွယ်မှုမပြုးခင် အဆက်အသွယ် ပြတ်တောက်သွားခဲ့ရင် ဘယ်လိုလုပ်မလဲဆိုတဲ့ကိစ္စမျိုး အတွက် ခေါင်းစားနေဖို့ မလိုတော့ပါဘူး။ ပြတ်တောက်သွားခဲ့ရင် နောက်တစ်ကြိမ် ထပ်မံဆက်သွယ်လိုက်ယုံးသာရှိလို့ Service Design လည်း များစွာရှိုးရှင်းသွားမှာပဲ ဖြစ်ပါတယ်။

**Cachable** – ဆက်သွယ်ရယူသူ Service ဟာ ရရှိထားတဲ့ရလဒ်ကို Cache အဖြစ် ယာယီသိမ်းဆည်းထားပြီး လိုအပ်သလို ပြန်လည်အသုံးပြုနိုင်ရမယ်လိုလည်း ဆိုပါတယ်။ ဒီနည်းနဲ့ တစ်ကြိမ်ရယူထားဖူးတဲ့ရလဒ်ကို နောင်လိုအပ်တဲ့အခါ အမြတ်များ ထံကနေ ပြန်လည်တောင်းခံနေစရာမလိုပဲ ယာယီသိမ်းဆည်း

ထားတဲ့ ရလဒ်ကို ပြန်လည် အသုံးပြနိုင်မှာဖြစ်လို့ စနစ်ရဲ့စွမ်းဆောင်ရည်နဲ့ အမြန်နှင့် တိုးတက်သွားစေမှာဖြစ်ပါတယ်။ အချက်အလက်ပေးပို့သူ Server Service အနေနဲ့လည်း ဘယ်လိုရလဒ်တွေကို Cache အဖြစ်ယာယိသိမ်းဆည်းသင့်တယ်၊ ဘယ်လိုအချက်အလက် တွေကို မသိမ်းဆည်းသင့်ဘူး စသဖြင့် Client Service ကို အသိပေးနိုင်ရမှာ ဖြစ်ပါတယ်။ ဒါတော့မှ နောက်ဆုံးရလဒ်ကို ရယူရမယ့်အစား မှားယွင်းပြီး Cache ရလဒ်ကို ဖော်ပြခိုက်ဆိုတဲ့ အမှားမျိုးကို ရှောင်ရှားနိုင်မှာဖြစ်ပါတယ်။

**Uniform Interface** – Service တစ်ခုကပေးနိုင်တဲ့ လုပ်ဆောင်ချက်တွေကို ရယူအသုံးပြနိုင်ဖို့အတွက် လုပ်ဆောင်ချက် တိုင်းအတွက် သီးခြား Identification (ID) ရှိရမယ်လို့ဆိုပါတယ်။ အဲဒါ ID ဟာ လုပ်ဆောင်ချက် ရလဒ်ရဲ့အစိတ်အပိုင်း တစ်ခု မဖြစ်သင့်ပဲသီးခြားတည်ရှိနိုင်ရမယ်လို့လည်း ဆိုပါတယ်။ Web Technology မှာ တော့ ဒီလုပ်ဆောင်ချက် ရရှိဖို့ URL တွေကို အသုံးပြုပါတယ်။ Web Resource တိုင်းအတွက် ညွှန်းဆိုထားတဲ့ ကိုယ်ပိုင် URL ကိုယ်စိရှိတဲ့အတွက် သက်ဆိုင်ရာ Resource ကို ရရှိလိုသူ Client Service က အဲဒါ URL တွေကို အသုံးပြုခြင်းအားဖြင့် ဆက်သွယ်ရယူနိုင်ခြင်း ဖြစ်ပါတယ်။ ပြီးတော့ အဲဒါ ID တွေဟာ အမိပိုယ်ရှိရမယ်လို့ လည်း သတ်မှတ်ထားပါသေးတယ်။ ဆိုလိုတာက ID ကို ကြည့်လိုက်ယုံနဲ့ ဘယ်လိုလုပ်ဆောင်ချက်ကို ဆောင်ရွက်သွားစေမှာလဲဆိုတာကို သိရှိနိုင်ရမယ်ဆိုတဲ့သောပါ။ Web Technology မှာ အသုံးပြုတဲ့ URL တွေ ဟာ အခြေခံအားဖြင့် ဖတ်ရှုနားလည်နိုင်လောက်တဲ့ အရာတွေဖြစ်ပေမယ့်၊ တစ်ခါတစ်ရုံမှာလည်း ရုပ်ထွေးပြီး နားလည်ရက်တဲ့ URL တွေ ဖြစ်သွားတက်လို့၊ Service ဖန်တီးသုက ကြည့်လိုက်ယုံနဲ့ နားလည်ရလွယ်တဲ့ URL တွေဖြစ်နေအောင် သတိပြုဖန်တီးပေးဖို့ လိုက်ပါတယ်။ ဒီအကြောင်းကို **Professional Web Developer** ရဲ့ အခန်း (၁၇) မှာ ဖော်ပြထားပါတယ်။ လိုဂုဏ်းကတော့ REST နည်းစနစ်နဲ့အညီ Service တွေ တည်ဆောက်တဲ့ အခါး လုပ်ဆောင်ချက်တိုင်းအတွက် ကိုယ်ပိုင် ID ရှိအောင် Design လုပ် ရမှာဖြစ်ပြီး အဲဒါ ID တွေကို ကြည့်လိုက်ယုံနဲ့ ဆောင်ရွက်သွားမယ့်လုပ်ဆောင်ချက်ကို သိရှိနိုင်စေတဲ့ Uniform Interface ရှိစေရမယ်ဆိုတဲ့ သော့ပဲ ဖြစ်ပါတယ်။

**Code on Demand** – Service တစ်ခုနဲ့တစ်ခု ဆက်သွယ်ဆောင်ရွက်ရာမှာ Data အချက်အလက်မှားကို အပြန်အလှန် ဖလှယ်ခြင်းအားဖြင့်သာ ဆက်သွယ်ဆောင်ရွက်ထို့ REST က သတ်မှတ်ထားပါတယ်။ တစ်နည်းအားဖြင့် Service တစ်ခုရဲ့ Function ကို ရယူပြီး၊ အဲဒါ Function ကို နောက် Service တစ်ခုက ချိတ်ဆက်အလုပ်လုပ်ခြင်းမျိုးကို မလုပ်ရပါဘူး။ ဒါပေမယ့် Service တစ်ခုက ရလဒ်အနေနဲ့ ပြန်ပေးတဲ့အခါး Data ကိုသာမက ပဲ ပရိုဂရမ Code ကို ပြန်ပေးနိုင်ပါတယ်။ ဒီနည်းနဲ့ Service တစ်ခုကပေးတဲ့ Code ကို နောက် Service က လို သလို မွန်းမံဖြည့်စွာခြင်းအားဖြင့် အသုံးပြနိုင်အောင် Design လုပ်ထားနိုင်ပါတယ်။ တစ်နည်းအားဖြင့် Service တစ်ခုနဲ့တစ်ခု တွဲချိတ်ထားတဲ့သောမျိုးနဲ့ Service တစ်ခုက နောက် Service တစ်ခုရဲ့ Function ကို ခေါ်မယ့် သင့်ပေမယ့်၊ လိုအပ်ရင် Code ကိုရလဒ်အနေနဲ့ ပေးနိုင်တဲ့ဆိုတဲ့ သော့ပဲဖြစ်ပါတယ်။ သတိပြုရမှာက၊ Server Service ပေးလိုက်တဲ့ Code ဟာ Client Service ထဲ၊ Code အနေနဲ့ရောက်သွားပြီး Client Service ပေါ်မှာသာ အလုပ်လုပ်တယ်ဆိုတဲ့ အချက်ပဲဖြစ်ပါတယ်။ Server Service က Code ကို ပေးယုံသာပေးပြီး၊ အဲဒါ Code ကို အလုပ်လုပ် စေတဲ့ကိုစွာကိုတော့ သူကိုယ်တိုင်မလုပ်တဲ့ သောပါ။ Web Technology မှာ ဒီနည်းစနစ်ကို Client-side Scripting လိုခေါ်ပြီး JavaScript ကို Code Exchange လုပ်ဖို့အတွက် သုံးပါတယ်။

**Data Format** – Server Service တစ်ခုဟာ Data အချက်အလက်မှားကို ရလဒ်အနေနဲ့ ပြန်လည်ပေးပို့တဲ့အခါး အဲဒါ Data ရဲ့ဖွေစည်းပုံ Format ကို တစ်ပါတည်း တွဲဖက်ထည့်သွင်းပေးရပါတယ်။ ဒီနည်းနဲ့ Data Format တစ်

မျိုးတည်းကို မြို့ခို့ဖို့မလိုတော့ပဲ ဘယ်လို Data Format နဲ့ ရလဒ်ပြန်ပေးသလဲဆိုတာကို Server Service က Client Service ကို အသိပေးခြင်းအားဖြင့် Data Format အမျိုးမျိုးကို လိုအပ်သလို အသုံးပြုသွားနိုင်မှာပဲဖြစ်ပါတယ်။ အသုံးများတဲ့ Data Format တွေကတော့ JSON, XML နဲ့ YAML တို့ပဲဖြစ်ကြပါတယ်။ ဒါ Data Format တွေကို Programming Language အားလုံးနီးပါးက နားလည်အလုပ်လုပ်ပေးနိုင်ကြပါတယ်။ တစ်နည်းအားဖြင့် Client Service အားလုံးက ဘယ်လို Data Format နဲ့ပဲ ရလဒ်ကို ရရှိသည်ဖြစ်စေ လက်ခံအလုပ်လုပ်ပေးနိုင်တယ်ဆိုတဲ့ သဘောလည်း ဖြစ်ပါတယ်။ ဒါပေမယ့် လက်တွေမှာ စနစ် Consistency အတွက် Format တစ်ခုကို သာ ရွေးချယ်အသုံးပြုကြလေ့ရှိပါတယ်။ XML-RPC, SOAP စတဲ့ နည်းပညာတွေကို ဆက်သွယ်ရေး Interface အဖြစ်အသုံးများခဲ့တဲ့ အချိန်တွေတုန်းက XML ကို Data Format အနေနဲ့ အသုံးများခဲ့ကြပါတယ်။ ကနောက်အချိန်မှာတော့ Code on Demand လုပ်ဆောင်ချက်ကို အသုံးချ ထားတဲ့ Web Service တွေ ပိုမိုတွင်ကျယ်လာတာနဲ့ အမျှ၊ Code Exchange အတွက် Standard အဖြစ်အသုံးပြုတဲ့ JavaScript နဲ့ တွဲဖက်အလုပ်လုပ်ရာမှာ ပိုမြို့ ထိရောက်တဲ့ JSON ကို အသုံးများလာကြပါတယ်။

## JavaScript Object Notation (JSON)

JSON (ဂျေဆုံး) လိုအတိကောက်ခေါ်တဲ့ JavaScript Object Notation ဆိတာဟာ JavaScript Object တွေ တည်ဆောက်ဖို့အတွက် အသုံးပြုရတဲ့ နည်းစနစ်တစ်မျိုးဖြစ်ပါတယ်။ ပြီးခဲ့တဲ့အခန်းမှာ JavaScript Object တွေကို Object Constructor ကနေ တစ်ဆင့် တည်ဆောက်နိုင်ပုံကို ဖော်ပြခဲ့ရီးဖြစ်ပါတယ်။

## JavaScript

```
1. function Person() {  
2.     this.fullName = "James Doe";  
3.     this.age = 32;  
4.     this.walk = function() {  
5.         console.log(this.fullName + " is walking.");  
6.     }  
7. }  
8.  
9. var james = new Person();  
10. james.walk(); // => James Doe is walking.
```

နှမူနာမှာ Person() Constructor ကို သုံးပြီး james ဆိုတဲ့ Object တစ်ခုကို တည်ဆောက်ထားပါတယ်။ အလားတဲ့ Object မျိုး ရရှိဖို့အတွက် JSON ကို အသုံးပြုပြီး အခုံဖို့တည်ဆောက်နိုင်ပါတယ်။

## JavaScript

```
1. var james = {  
2.     fullName: "James Doe",  
3.     age: 32,  
4.     walk: function() {  
5.         console.log(this.fullName + " is walking.");  
6.     }  
7. }  
8.  
9. james.walk(); // =>James Doe is walking
```

Constructor ကနေတစ်ဆင့် Object ကို တည်ဆောက်ခြင်းမဟုတ်ပဲ JSON ရေးထုံးနဲ့ Object တစ်ခုကို တိုက်ရှိက် တည်ဆောက်ယူခြင်းဖြစ်ပါတယ်။ JSON ဟာ အမှန်တော့ Hash Array တစ်ခုသာဖြစ်ပါတယ်။ Array တစ်ခုဖြစ်တဲ့ အတွက် Index တစ်ခုနဲ့တစ်ခုကြေားမှာ Comma နဲ့ ပိုင်းခြားပေးရပါတယ်။ Index တစ်ခုဟာ Property တစ်ခုဖြစ်ပါတယ်။ Index ရဲ့ Key နဲ့ Value ကို တော့ Colon နဲ့ ပိုင်းခြားပေးရပါတယ်။ Key အတွက် နှစ်သက်ရာအမည်ကို ပေးနိုင်ပါတယ်။ Value အတွက် တန်ဖိုးသတ်မှတ်ရာမှာတော့ JavaScript Data Type များဖြစ်တဲ့ Number, String, Boolean တို့ သာမက Array, Function, Object နဲ့ null တို့ကိုလည်း အသုံးပြုနိုင်ပါတယ်။

### JavaScript

```

1. var james = {
2.     fullName: "James Doe",
3.     age: 32,
4.     walk: function() {
5.         console.log(this.fullName + " is walking.");
6.     },
7.     address: {
8.         street: "52, Main Road",
9.         city: "Yangon",
10.    },
11.    phones: [
12.        "123 456 789",
13.        "987 654 321"
14.    ],
15.    children: null
16. }
17.
18. james.address.city;      // => Yangon
19. james.phones[0];         // => 987 654 321

```

JSON ဟာ မူလက JavaScript Programming Language နဲ့ တွဲဖက်အသုံးပြုဖို့ ရည်ရွယ်ဖန်တီးခဲ့ပေမယ့် အခါ အခါ မှာတော့ မည်သည့် Programming Language နဲ့မဆို တွဲဖက်အသုံးပြုနိုင်တဲ့ Language Independent Data Format တစ်ခုအဖြစ် အသုံးတွင်ကျယ်လာခဲ့ပါတယ်။ အထက်မှာပြောခဲ့သလို Web Service တွေအတွက် အဓိက Client-side Language ဖြစ်တဲ့ JavaScript နဲ့ တွဲဖက်အလုပ်လုပ်ရာမှာ ပိုမိုအဆင်ပြေခြင်းနဲ့ သူ့အရင်က အသုံးတွင်ကျယ်ခဲ့တဲ့ XML ထက် ရေးထုံးပိုင်း ပိုမိုရိုးရှင်းခြင်းတို့ကြောင့် အခုနောက်ပိုင်းမှာ Service တွေ တစ်ခု နဲ့တစ်ခုကြေား ဆက်သွယ်ရာမှာ အဓိက Data Exchange Format အဖြစ် အသုံးပြုလာခဲ့ကြပါတယ်။

လက်တွေမှာ Data Exchange အတွက်သုံးတဲ့ Data JSON နဲ့ JavaScript Object JSON တို့ အနည်းငယ် ကွာပါတယ်။ အသိသာဆုံး ကွာခြားချက်ကတော့ Data JSON မှာ Function တွေ Regular Expression တွေပါဝင်လို့မရ ခြင်း ပဲဖြစ်ပါတယ်။ JavaScript Object JSON မှာတော့ Property တစ်ခုအတွက်တန်ဖိုးကို Function သို့မဟုတ် Regular Expression အဖြစ် သတ်မှတ်ထားနိုင်ပါတယ်။

ကနေအချိန်မှာ REST Web Service လို့ပြောလိုက်ရင် JSON ကို Data Exchange Format အဖြစ်သုံးပြီး HTTP ကို ဆက်သွယ်ရေး Interface အဖြစ်အသုံးပြုထားတဲ့ Service များကို ဆိုလိုခြင်းပဲဖြစ်ပါတယ်။ REST နည်းစနစ် က သတ်မှတ်ထားတဲ့ Stateless, Cachable, Uniform Interface စတဲ့ လုပ်ဆောင်ချက်အများစုံကို HTTP က ပေး

ထားပြီး ဖြစ်ပါတယ်။ ကျွန်တော်တို့ကသာ ကျွန်တော်တို့၏ Service တွေကို Design လုပ်တဲ့နေရာမှာ ဒီ သတ်မှတ်ချက်တွေ ကနေ သွေဖီသွားခြင်းမရှိအောင် သတိပြု၍ Design လုပ်ရမှာပဲဖြစ်ပါတယ်။ Language အနေ နဲ့လည်း Client Service တွေ အတွက် JavaScript ကို အမိကအဖြစ် အသုံးပြုတက်ကြပေမယ့် Core Service တွေနဲ့ Utility Service တွေကို တော့ နှစ်သက်ရာ နည်းပညာနဲ့ အသုံးပြုတည်ဆောက် နိုင်ပါတယ်။ ကြားခံဆက် သွယ်ရေး Interface အဖြစ်သာ HTTP ကို အသုံးပြုဖို့ လိုပါတယ်။

## Conclusion

Software Development မှာ 3-Tires Architecture, Modular Design, Layered System, Model-View-Controller စသဖြင့် Architecture ပိုင်းဆိုင်ရာ နည်းစနစ်အမျိုးမျိုးရှိကြပါတယ်။ တစ်ခုနဲ့တစ်ခု အသေးစိတ်တွေ မတူကြပေမယ့် လိုရင်းကတော့ Software Solution တစ်ခုကို ကဏ္ဍအလိုက် ခွဲခြားပြီးတည်ဆောက်ခြင်းပဲဖြစ် ပါတယ်။ ဒီလိုခွဲခြား တည်ဆောက်တော့မှ ရေရှည်ပြုပြင်ထိမ်းသိမ်းရေးမှာ ထိရောက်မှာဖြစ်တဲ့အတွက် အခုလို နည်းစနစ်အမျိုးမျိုးကို တိတွင် လာခဲ့ကြခြင်းပဲဖြစ်ပါတယ်။

Service Oriented Architecture ဆိုတာဟာ၊ အခုမှ ပေါ်လာတဲ့ နည်းစနစ်သစ်တစ်ခုမဟုတ်ပေမယ့် Internet နဲ့ Web Technology တို့၏ ဖွံ့ဖြိုးတိုးတက်လာမှုနဲ့အတူ ကနေအချိန်မှာ ပိုမိုအသုံးဝင်ပြီး တွင်ကျယ်လာတဲ့ Software Design Architecture တစ်ခု ဖြစ်လာခဲ့ပြီ ဖြစ်ပါတယ်။

အဂိုင်း (J)

Project Management

အမှားတစ်ခုပြုလုပ်မိလို့၊ မူလအခြေအနေပြန်ရအောင် Code ကို  
Manual ပြန်ပြင်ရတယ်ဆိုတာ မဖြစ်သင့်ပါဘူး။ အဖွဲ့လိုက်ပူးပေါင်း  
ဆောင်ရွက်ရာမှာ၊ တစ်ဦးရေးသားထားတဲ့ Code နဲ့ နောက်တစ်ဦး  
ရေးသားထားတဲ့ Code ပေါင်းစပ်တဲ့လုပ်ငန်းကို Manual လုပ်ရတယ်  
ဆိုတာလည်း မဖြစ်သင့်ပါဘူး။

## **Ubuntu – သင့်အတွက် Linux (စာအုပ်)**

Ubuntu Linux အသုံးပြန်လိုးအား Installation မှ စတင်၍ System Configuration နှင့် Development Environment တည်ဆောက်  
ခြင်းအထိ အဆင့်လိုက်ရေးသား ဖော်ပြထားသည့်စာအုပ်  
အောက်ပါလိပ်စာတွင် အခမဲ့ ရယူနိုင်သည်။

## အခန်း(၅) – Version Control System

"ကဲ... Demo လုပ်ဖို့ အားလုံးအသင့်ပဲလားလော်။"

"ဟုတ်ကဲ့.. နေ့လည်လောက်ဆိုရင်တော့ အသင့်ဖြစ်ပါပြီ။"

"ဘယ်လိုဖြစ်တာလဲ။ မနေ့ကတည်းက အားလုံးပြီးနေပြီးသား မဟုတ်လား။"

"ဟုတ်ပါတယ်။ Code တစ်ချို့ပြင်လိုက်မိတာ အလုပ်မလုပ်တော့လို့ ပြန်စစ်နေတာခင်ဗျာ။"

"ဟာ... ပြဿနာပဲကွာ။ ဘာလို့ချိန်မှပြင်တာလဲ။ ဒီနေ့ Demo ပြမယ်လို့ကြော်ပြီး နေပြီ။ နိုင် အလုပ်လုပ်နေတဲ့အတိုင်း ပြန်ထားလိုက်လော်။"

"ဟုတ်ကဲ့... နိုင် အတိုင်းပြန်ထားပြီးပါပြီ။ ဒါလည်း အလုပ်မလုပ်သေးဘူး။ ဘာဖြစ်သွားလဲ မသိဘူး။"

"ခက်တာပဲကွာ။ တော်သေးတာပေါ့။ အရင် Version တွေ ငါသိမ်းထားမိလို့။ Share Folder ထဲကိုဝင်ပြီး အရင် Version ကို ကူးယူလိုက်။"

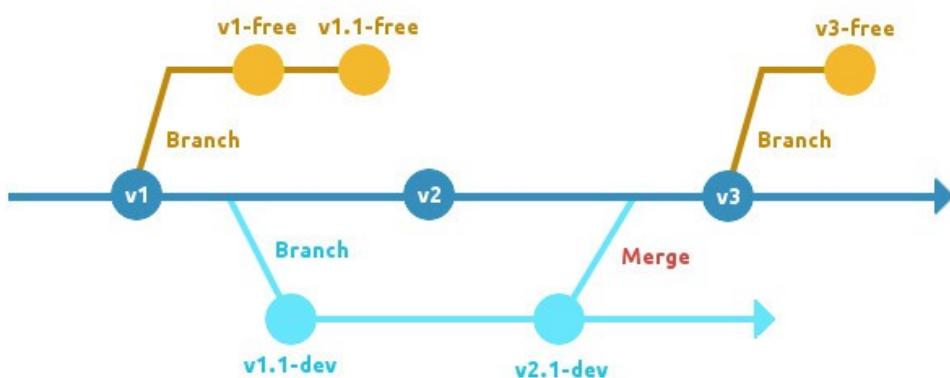
"Share Folder ထဲမှာတွေ့ပါတယ်။ ဒါပေမယ့် project-latest, project-update, project-final, project-stable, project-update-final, project-real-latest အများကြီးဖြစ်နေတယ်။ အဲဒါထဲ က ဘယ်ဟာကို ကူးယူရမလဲ"

"project-latest ကိုယူလေကွာ။ အဲ... နေ့ဦး။ project-real-latest ဖြစ်မယ်ထင်တယ်။ ရှုပ်တယ်ကွာ ဘယ်ဟာနောက်ဆုံး Version လဲသေချာအောင်၊ တစ်ခုချင်းသာ လိုက်စမ်းကြည့်လိုက်တော့။"

ဒါဟာ Version Control System အသုံးမပြုတဲ့ Software Team တွေမှာ ကြိုတွေ့ရလေ့ရှိတဲ့ ပြဿနာတစ်ရပ်ပဲ ဖြစ်ပါတယ်။ Software Project တစ်ခုခုတိတာဟာ အချိန်နဲ့အမျှ ပြောင်းလဲနေတက်တဲ့သဘာဝ ရှိပါတယ်။ ရှေ့ ပိုင်းမှာလည်း၊ တိကျတဲ့ Requirement ရရှိဖို့ ခက်ခဲတက်ကြောင်းနဲ့ အလုပ်လုပ်နေတဲ့ Code ကို တစ်နေရာမှာ ပြပြင်ခြင်းအားဖြင့် အခြား တစ်နေရာမှာ မမော်လင့်ပဲ သက်ရောက်သွားပြီး Bug တွေ ဖြစ်ပေါ်တက်ကြောင်းတို့ ကို ဆွေးနွေးခဲ့ပြီးဖြစ်ပါတယ်။ ဒီလို့ အခက်အခဲတွေ ရှိတက်လို့ဆိုပြီး ပြပြင်မှုတွေကို မလုပ်ချင်လို့ မရပါဘူး။

အပြောင်းအလဲတွေရှိလာလို့ ပြုပြင်ဖြည့်စွက်ရတဲ့အခါမှာ၊ မမော်မှန်းထားတဲ့ အမှားတစ်ခုကြောင့် မူလအလုပ်လုပ်နေတဲ့ စနစ်ကို ထိခိုက်ပြီး အလုပ်မလုပ်တော့တာမျိုး ဖြစ်တက်ပါတယ်။ ဒါကြောင့် လက်ရှိအလုပ်လုပ်နေတဲ့ အဆင့်တစ်ခုကို Version တစ်ခုဖြစ် သတ်မှတ်ပြီး၊ သီးခြားခွဲခြားသိမ်းဆည်းထားပြီးမှ ဖြည့်စွက်ပြင်ဆင်မှ တွေကို ဆက်လက်ဆောင်ရွက်ရတဲ့ သဘောရှိပါတယ်။ အဲဒီလိုခွဲခြားထားမှုလည်း၊ ဖြည့်စွက် Code တွေကို ရဲ့ရဲ့ ထံ့ထံ့ ဖြည့်စွက်စမ်းသပ်နိုင်မှာဖြစ်ပါတယ်။ အသစ်ဖြည့်စွက်မှုကြောင့် အမှားအယွင်း တစ်စုံတစ်ရာ ရှိခဲ့ရင်လည်း မူလအလုပ်လုပ်နေတဲ့အခြေအနေကို အချိန်မရွေး ပြန်လည်ရရှိနိုင်မှာဖြစ်ပါတယ်။ Version ခွဲခြားမထားရင် မှားယွင်းမှုဖြစ်သွားမှာကို စိုးရိမ်စိတ်နဲ့ ပြင်ဆင်ဖြည့်စွက်မှုတွေကို ရဲ့ရဲ့ မလုပ်ရတော့တဲ့အတွက် အလုပ်မတွင် ဖြစ်တက်ပါတယ်။ ဒီထက်ပိုဆိုးတာက၊ ပြင်ဆင်ဖြည့်စွက်မှုကြောင့် အမှားအယွင်းရှိလာတဲ့အခါ မူလအခြေ အနေ ပြန်ရောက်ဖို့ကို အကောင်လေး အချိန်ပေးပြီး ပြန်လည်ရှင်းလင်းရတက်လို့ အချိန်တွေ ကုန်တက်ပါသေးတယ်။

ဒီလို Version တွေခွဲခြား မှတ်တမ်းတင်တဲ့ လုပ်ငန်းဆောင်ရွက်ရာမှာ၊ Source Code ဖိုင်တွေကို ကိုယ်တိုင် Copy တွေ ကူး၊ နာမည်အမျိုးမျိုးပေး စသဖြင့် Manual လုပ်နေစရာမလိုပဲ၊ စနစ်ကျထိရောက်အောင် ကူးညီပေးနိုင်တာ ကတော့ Version Control System (VCS) တွေပဲဖြစ်ပါတယ်။ Version Control System ကို Revision Control System လိုလည်းခေါ်သလို၊ Source Code Management System (SCM) လိုလည်း ခေါ်တက်ကြပါသေးတယ်။



ပုံ (၅.၁) - Branching in VCS

ပုံ (၅.၁) မှာလေ့လာကြည့်ပါ။ v1, v2, v3 စတဲ့ Version တွေဟာ အဆင့်လိုက်မှတ်တမ်းတင်ထားတဲ့ မူလ Version တွေဖြစ်ပါတယ်။ ဒီမူလ Code Base ကနေ လုပ်ဆောင်ရွက်အနည်းငယ်ကွဲပြားတဲ့ သီးခြား Version တွေဖန်တီးဖို့ အတွက် နောက်ထပ် Project တစ်ခု ဖွင့်ဖို့မလိုပါဘူး။ လက်ရှိ Code Base ကနေပဲ Branch အဖြစ် ခွဲထဲတဲ့လိုက်ပြီး အတူတစ်ကွ ဆက်လက်ဆောင်ရွက်နိုင်စေမှာဖြစ်ပါတယ်။ မူလ Code Base ကို တစ်ချို့ VCS တွေက Trunk လိုခေါ်ပြီး၊ တစ်ချို့ကတော့ master လို့ ခေါ်တက်ကြပါတယ်။

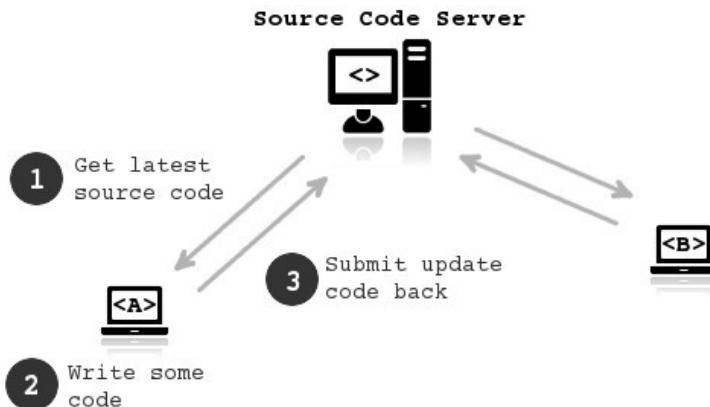
VCS တွက master နဲ့ Branch တွေအကြား အပြန်အလှန် ကူးပြောင်းအလုပ်လုပ်နိုင်အောင် စီမံပေးထားကြပါတယ်။ master ကို ရွေးထားပြီး ပြင်ဆင်မှုတွေပြုလုပ်ရင် ပြုလုပ်လိုက်တဲ့ပြင်ဆင်မှုတွက master Code Base ကိုပဲ သက်ရောက်စေမှာဖြစ်သလို၊ Branch တစ်ခုကိုရွေးထားပြီး ပြင်ဆင်မှုတွေပြုလုပ်ရင်တော့ ရွေးချယ်ထားတဲ့ Branch ကိုပဲ ပြင်ဆင်မှုက သက်ရောက်စေမှာ ဖြစ်ပါတယ်။

Branch တွေကို စမ်းသပ်လုပ်ဆောင်ချက်တွေ ထည့်သွင်းစမ်းသပ်နှိပ်လည်း အသုံးပြုကြပါတယ်။ ဒီနည်းနဲ့ စမ်းသပ်လိုတဲ့ Code တွေက master Code Base ကို မထိခိုက်စေပဲ၊ သီးခြားခွဲထားတဲ့ Development Branch ပေါ်မှာ လုပ်လပ်လပ်လပ် စမ်းသပ်ရေးသားနိုင်စေမှာဖြစ်ပါတယ်။ Development Branch ပေါ်မှာ ရေးသားထားတဲ့ စမ်းသပ် လုပ်ဆောင်ချက်ကို မူလ Version မှာ ပါဝင်သင့်တယ်လို့ အတည်ပြုနိုင်တဲ့အခါမှာ အဲဒီ Development Branch ကို master Code Base နဲ့ ပေါင်းစပ်လိုက်နိုင်ပါတယ်။ ဒီလုပ်ဆောင်ချက်ကို Merge လုပ်တယ်လို့ ခေါ်ပါတယ်။

ံ (၅.၁) မှာ ဖော်ပြထားချက်အရာ၊ အလယ်က v1, v2, v3 စတဲ့ Version တွေဖြစ်ပြီး၊ အပေါ်က v1-free, v1.1-free စတဲ့ Version တွေကတော့ မူလ master Code Base ကနေ လုပ်ဆောင်ချက်တစ်ချို့ ပြောင်းလဲထားတဲ့ သီးခြား Version များအဖြစ် Branch ခွဲယူထားခြင်းဖြစ်ပါတယ်။ အောက်ဘက်က v1.1-dev, v2.1-dev စတဲ့ Version တွေက တော့ စမ်းသပ်လုပ်ဆောင်ချက်တွေ ထည့်သွင်းစမ်းသပ်နှိပ်နှိုးခဲ့ခြားထားတဲ့ Branch ဖြစ်ပြီး အဲဒီ Branch ကို မူလ master Code Base နဲ့ တစ်နေရာမှာ ပြန်လည် Merge လုပ်ထားတာကိုလည်း တွေ့ရမှာဖြစ်ပါတယ်။

VCS တွေက master နဲ့ Branch တွေပေါ်က မှတ်သားထားတဲ့ Version အမှတ်တိုင်းကို ပြန်သွားနိုင်အောင်လည်း စီမံ ထားပေးလို့ အမှားအယွင်းတစ်စုံတစ်ရာကြောင့်ပဲဖြစ်ဖြစ် အခြားအကြောင်း တစ်ခုခုကြောင့်ပဲဖြစ်ဖြစ်လို့ အပ်လာတဲ့အခါ အရင် Version တွေကို အချိန်မရေး ပြန်လည်ရယူနိုင်မှာပဲဖြစ်ပါတယ်။

ဒါတင်မက၊ အဖွဲ့လိုက်ပူးပေါင်းဆောင်ရွက်ရတဲ့အခါမှာလည်း Code Base တစ်ခုတည်းကနေ VCS အကူအညီနဲ့ ရယူရေး သားခြင်းအားဖြင့် Developer တစ်ဦးရေးသားထားတဲ့ Code နဲ့ နောက် Developer တစ်ဦးရေးသားထားတဲ့ Code ကို ပေါင်းစပ်တဲ့လုပ်ငန်းကို Manual လုပ်နေစရာမလိုတော့ပဲ VCS က စနစ်တစ်ကျပေါင်းစပ်ပေးသွားမှာပဲဖြစ်ပါတယ်။ ဒါကြောင့် VCS တွေကို Version တွေစီမံဖို့အတွက်သာမက အဖွဲ့လိုက်ပူးပေါင်းဆောင်ရွက်ရာမှာ အခြေခံ Tool တစ်ခု အဖြစ် အသုံးပြုတက်ကြပါသေးတယ်။ ဒါကြောင့်လည်း VCS တွေကို Source Code Management System (SCM) လို့ခေါ်ခြင်းဖြစ်ပါတယ်။ Version ကိုသာမက၊ Source Code နဲ့ ပက်သက်တဲ့ အခြားစီမံမူလုပ်ငန်းများကိုပါ ဆောင်ရွက်ပေးတဲ့အတွက် ဖြစ်ပါတယ်။



ပုံ (၅.၂) - Centralized VCS Workflow

ဒီလိုအဖွဲ့လိုက်ပူးပေါင်းဆောင်ရွက်နိုင်ဖို့ Centralize Source Code Server လိုအပ်တက်ပါတယ်။ ဒီတော့မှ ပုံ (၅.၂) မှာ ဖော်ပြထားသလို၊ Team မှာပါဝင်တဲ့ Developer တွေက၊ အခြား Developer များရေးသားထဲတဲ့ နောက်ဆုံးရလဒ်ကို Central Server ထံမှရယူခြင်း၊ မိမိတို့ရေးသားဖြည့်စွက်ချက်တွေကို Central Server ထံ ပြန်လည်ပေးပိုခြင်းအားဖြင့် Code Base တစ်ခုတည်းကို Developer အများ ပူးပေါင်းဆောင်ရွက်ခြင်း လုပ်ဆောင်ချက်ကို ရရှိမှာပဲဖြစ်ပါတယ်။

## 5.1 VCS Terminologies

Version Control System တွေအများကြီးရှိပါတယ်။ စနစ်တစ်ခုနဲ့တစ်ခု အသေးစိတ်လုပ်ဆောင်ချက်တွေ မတူကြပေမယ့် အခြေခံရည်ရွယ်ချက်တူပြီး၊ ပါဝင်တဲ့ Function တွေနဲ့ လုပ်ငန်းစဉ် Operation တွေကို ခေါ်တဲ့အခါ အပေါ်တွေလည်း တူတက်ပါတယ်။ VCS တစ်ခုမှာပါဝင်တက်တဲ့ အသုံးအနှံးတွေကို ဖော်ပြပေးလိုက်ပါတယ်။

**revision/version** – Source Code ကို ပြင်ဆင်မှုတစ်ချို့လုပ်ပြီး မှတ်တမ်းတင်သိမ်းဆည်းလိုက်တဲ့ အမှတ်တစ်ခုကို version (သို့မဟုတ်) revision လို့ ခေါ်ပါတယ်။

**branch** – အတိုင်းအတာတစ်ခုထိ ပြီးမြောက်ပြီးသား Software Project တစ်ခုရဲ့ Source Code ကို ပြင်ဆင်ဖြည့်စွက်မှုတွေ ပြုလုပ်လိုတဲ့အခါ၊ မူလ Code Base ပေါ်မှာ တိုက်ရှိက်လုပ်မယ့်အစား သီးခြား branch တစ်ခု အဖြစ် မူဖွားယူပြီးမှ ဆောက်ရွက်ကြလေ့ရှိပါတယ်။ ဒီနည်းနဲ့ စမ်းသပ်မှုကြောင့် ဖြစ်ပေါ်လာတဲ့ မလိုလားအပ်တဲ့ ပြဿနာတွေ မူလ master Code Base ကို သက်ရောက်ခြင်း မရှိတော့မှာပါ။ သီးခြား branch ပေါ်မှာ ဆောင်ရွက်နေတဲ့ ဖြည့်စွက်ပြင်ဆင်ချက်ဟာ လက်ခံနိုင်တဲ့ အတိုင်းအတာတစ်ခု ရောက်ပြီဆိုတော့မှ အဲဒီ branch ကို မူလ master Code Base နဲ့ ပေါင်းစပ်စေခြင်း အားဖြင့် master Code Base ထဲမှာလည်း ဖြည့်စွက်ပြင်ဆင်ချက်အသစ်တွေ ရောက်ရှိပါဝင်သွားစေမှာ ဖြစ်ပါတယ်။

**fork** – fork ဆိုတာ branch လိုပဲ မူလ Code Base ကို မူမားယူခြင်းဖြစ်ပါတယ်။ ဒါပေမယ့် မူလ Code Base နဲ့ ပြန်လည်ပေါင်းစပ်ဖို့မရည်ရွယ်ပဲ သီးခြား Project တစ်ခုအနေနဲ့ ဆောင်ရွက်ဖို့ ရည်ရွယ်ချက်နဲ့ မူမားယူတဲ့ သဘောမျိုးဖြစ်ပါတယ်။

**master/trunk** – နောက်မှ ခွဲယူထားတဲ့ branch မူကဲ့မဟုတ်ပဲ မူလစတင်ကတည်း အခြေဖြေထားတဲ့ မူလ Code Base ကို master branch (သို့မဟုတ်) trunk လိုခေါ်ပါတယ်။

**merge** – branch တစ်ခုနဲ့ ခွဲခြားစီမံနေတဲ့ Code Base ကို master (သို့မဟုတ်) အခြား branch တစ်ခုခဲ့ပေါင်းစပ်စေတဲ့ လုပ်ဆောင်ချက်ဖြစ်ပါတယ်။ branch တစ်ခုတည်းမှာပဲ Developer တစ်ဦးခဲ့ပြင်ဆင်ချက်ကို အခြားတစ်ဦးရဲ့ပြင်ဆင်ချက်နဲ့ ပေါင်းစပ်ခြင်းလုပ်ငန်းကိုလဲ merge လုပ်တယ်လိုပဲ ခေါ်ပါတယ်။

**commit/checkin** – ပြင်ဆင်ဖြည့်စွက်မှုတွေပါဝင်တဲ့ Code ကို Version သစ်တစ်ခုအဖြစ် မှတ်တမ်းတင်လိုက်တဲ့ လုပ်ဆောင်ချက်ကို commit လုပ်တယ် (သို့မဟုတ်) checkin လုပ်တယ်လိုခေါ်ပါတယ်။

**change/diff** – ပြင်ဆင်ထားတဲ့ Code နဲ့ နောက်ဆုံး commit လုပ်ခဲ့တဲ့ Version တို့အကြား ကွာခြားသွားတဲ့ ကွာခြားချက်ကို change (သို့မဟုတ်) diff လိုခေါ်ပါတယ်။ ကွားခြားသွားတယ်ဆိုတာ အသစ်တိုးလာတာ လည်း ဖြစ်နိုင်ပါတယ်၊ မူလ Code ကို ပြင်လိုက်တာလည်း ဖြစ်နိုင်ပါတယ်၊ မူလ Code အချို့ကို ဖယ်ထုတ် လိုက်တာ လည်း ဖြစ်နိုင်ပါတယ်။

**delta compression** – Version အသစ်တစ်ခုအဖြစ် မှတ်တမ်းတင်သိမ်းဆည်းတဲ့အခါ Code Base တစ်ခုလုံး ကို နောက်တစ်ကြိမ် ထပ်မသိမ်းတော့ပဲ ကွားခြားသွားတဲ့ diff ကိုသာ သိမ်းဆည်းတဲ့ လုပ်ဆောင်ချက်ကို delta compression လိုခေါ်ပါတယ်။

**change list/patch** – ဥပမာ – Open Source Project တစ်ခုကို fork လုပ်ယူပြီး ကိုယ်ပိုင်ဖြည့်စွက်မှုတွေ လုပ်တယ် ဆိုပါစို့။ ရရှိလာတဲ့ ပြင်ဆင်ဖြည့်စွက်မှုမှတ်တမ်း Version ကို မူလ Project မှာပါဝင်သွားစေလိုတဲ့အခါ – မိမိ ပြုလုပ်ထားတဲ့ ပြင်ဆင်ဖြည့်စွက်မှုမှတ်တမ်းကို change list (သို့မဟုတ်) patch အဖြစ်ထုတ်ယူရပါတယ်။ ရရှိလာတဲ့ patch ဖိုင်ကိုမူလ Project ရဲ့ တာဝန်ခံထဲ ပေးပို့နိုင်ပါတယ်။ တာဝန်ခံက စီစစ်လေ့လာပြီး ပေါင်းစပ် ပေးမယ်ဆိုရင် မိမိပြုလုပ်ထားတဲ့ ပြင်ဆင်ဖြည့်စွက်မှုတွေလည်း မူလ Project မှာ ပါဝင်သွားမှာဖြစ်ပါတယ်။

**repository** – Version အားလုံးနဲ့ Branch အားလုံးတို့ကို စုစည်းသိမ်းဆည်းထားတဲ့ Code Base ကြီးကို repository လို့ ခေါ်ပါတယ်။

**checkout** – Central Source Server ကနော နောက်ဆုံး Update Version (သို့မဟုတ်) Version တွေ ထဲကတစ်ခု ကို ကိုယ့် Local Machine ထဲ ကူးယူတဲ့ လုပ်ဆောင်ချက်ကို checkout လုပ်တယ်လိုခေါ်ပါတယ်။ တစ်ချို့ VCS တွေမှာ တော့ branch တစ်ခုကနော နောက်တစ်ခုကိုကူးပြောင်းတဲ့ လုပ်ဆောင်ချက်အတွက်လည်း checkout ဆိုတဲ့ အသုံးအနှံးကို ပဲသုံးပါတယ်။

**working copy** – Central Source Server ကနေ checkout လုပ်ယူလိုက်လို ကိုယ့် Local Machine ထဲ ရောက်ရှိလာတဲ့ Version ကို working copy လို့ခေါ်ပါတယ်။

**clone** – clone ဆိုတာကတော့ Central Server ပေါ်က repository ကြီးတစ်ခုလုံးကို ကိုယ့် Local Machine မှာ မူမှား ယူလိုက်တဲ့ လုပ်ဆောင်ချက်ဖြစ်ပါတယ်။ checkout နဲ့မတူတာက၊ checkout က Version တစ်ခုကိုသာ ရယူခြင်းဖြစ်ပြီး၊ clone က Version တွေ branch တွေအားလုံးပါဝင်တဲ့ repository တစ်ခုလုံးကို ရယူခြင်းဖြစ်ပါတယ်။

**conflict** – Local Copy ပေါ်မှာ ပြင်ဆင်ဖြည့်စွက်မှုတစ်ချို့၊ လုပ်လိုက်တယ်ဆိုပါစိုး။ အဲဒီ ပြင်ဆင်မှုကို Central Server ထံမပေးပို့မိ Server က နောက်ဆုံး Version ကို အရင်ရယူရတဲ့ သဘော ရှိပါတယ်။ ဒီလိုရယူလိုက်တဲ့ အခါ ကိုယ်ပြင်ဆင် ထားတဲ့ Code ဖိုင် နဲ့ Server ကနေရရှိလာတဲ့ နောက်ဆုံး Version ရဲ့ Code ဖိုင် တူညီနေတဲ့ အခါ conflict ဖြစ်ပါတယ်။ VCS က ကိုယ်ပြင်ထားတာကို နောက်ဆုံးအနေနဲ့ မှတ်ယူပေးရမလား၊ Server ကနေ ရရှိလာတဲ့ ပြင်ဆင်ချက်ကို နောက ဆုံးအနေနဲ့ မှတ်ယူပေးရမလား မဆုံးဖြတ်နှင့်တဲ့ အတွက် conflict ဖြစ်ရခြင်း ဖြစ်ပါတယ်။ ဒီလို conflict တွေရှိလာတဲ့ အခါ ကိုယ်တိုင်စီစစ်ပြီး ပေါင်းစပ်၍ သော်လည်ကောင်း၊ နှစ်ခုထဲက တစ်ခုကို နောက်ဆုံးအဖြစ် ဆုံးဖြတ်ပေး၍ သော် လည်ကောင်း **resolve** လုပ်ပေးရတက်ပါတယ်။

**head** – HEAD လို့ သုံးကြလေ့ရှိပြီး နောက်ဆုံး commit, နောက်ဆုံး Version ကို မြန်းဆိုတဲ့ Keyword ဖြစ်ပါတယ်။

**tag/label** – commit လုပ်ပြီး မှတ်တမ်းတင်ခဲ့တဲ့ Version ပေါင်းများစွာရှိနိုင်ပါတယ်။ အဲဒီထဲကမှ အရေးပါတဲ့ Version တွေကို အမည်ပေးတဲ့လုပ်ငန်းကို tag (သို့မဟုတ်) label လို့ခေါ်ပါတယ်။ ဥပမာ – အဆင့်ဆင့် သိမ်းဆည်းလာတဲ့ Version တွေထဲက Version အမှတ်တစ်ခုကို 0.1.0 လို့ tag လုပ်ထားနိုင်ပါတယ်။

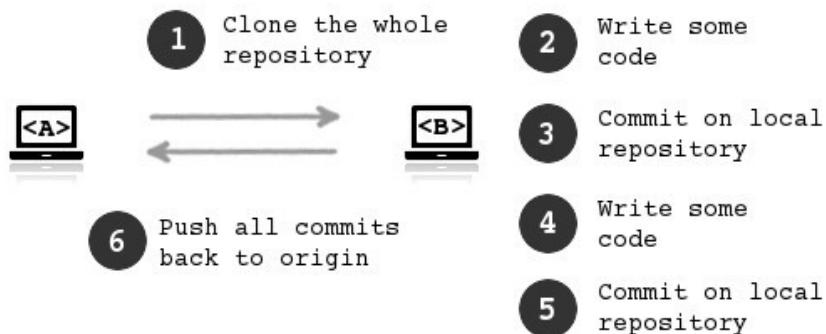
Version နံပါတ် ပေးပို့ပေးနည်းကို မကြောင်မှာ ဆက်လက်ဖော်ပြသားပါမယ်။

## 5.2 – Centralized VCS vs. Distributed VCS

Version Control System တွေမှာ အမျိုးအစားအားဖြင့် နှစ်မျိုးရှိပါတယ်။ တစ်မျိုးကို Centralized Version Control System လို့ခေါ်ပြီး နောက်တစ်မျိုးကိုတော့ Decentralized Version Control System (သို့မဟုတ်) Distributed Version Control System လို့ ခေါ်ပါတယ်။

Centralized VCS တွေမှာ Source Code Repository ကို Central Server တစ်လုံးမှာထားပြီး Developer တွေက ကိုယ့် Local Machine ပေါ်မှာ Working Copy ကို ရယူအလုပ်လုပ်ကြရပါတယ်။ ဆိုလိုတာက၊ Branch တွေနဲ့ Version မှတ်တမ်းဟာ Central Server မှာသာရှိနေပြီး Developer တွေက နောက်ဆုံး Version ကိုသာ ရယူအလုပ်လုပ်ကြခြင်း ဖြစ်ပါတယ်။ Commit လုပ်လိုက်တိုင်းမှာ Update Source Code ကို Central Server ထဲ ပေးပို့ခြင်းအားဖြင့် Version မှတ်တမ်းသစ်ကို Central Server ပေါ်မှာ တိုက်ရှိက်သိမ်းဆည်းသွားမှာ ဖြစ်ပါတယ်။

Distributed VCS တွေမှာတော့ Central Server ရှိဖို့ မလိုပါဘူး။ မည်သူမဆို မိမိတို့ Local Machine ပေါ်မှာ Repository တစ်ခုတည်ဆောက်နိုင်ပါတယ်။ အခြား Developer တွေက အဲဒီ Repository ကို ရယူဖို့လိုတဲ့အခါ Repository တစ်ခုလုံးကို Clone လုပ်ယူနိုင်ပါတယ်။ ဒါကြောင့် Team မှာ ပါဝင်သူအားလုံးမှာ Repository တစ်ခုလုံးရဲ့ မူပွားတွေ ကိုယ်စိရှိနေတဲ့သဘောဖြစ်ပြီး မြှုလုပ်လိုတဲ့ ပြင်ဆင်ဖြည့်စွက်မှုတွေကို မူလ Original Repository နဲ့ ဆက်သွယ်ဖို့မလိုပဲ ကိုယ့်မူပွားပေါ်မှာ ဆက်လက် ဆောင်ရွက်ရပါတယ်။ ပုံ (၅.၃) မှာ လေးလာ ကြည့်ပါ။



ပုံ (၅.၃) - Distributed VCS Workflow

Version တစ်ခုအဖြစ် မှတ်တမ်းတင်ဖို့ Commit လုပ်လိုက်တဲ့အခါမှာ မူလ Original Repository ကို သွားပြီး သက်ရောက် တာမျိုးမဟုတ်ပဲ ကိုယ့် Local Repository ပေါ်မှာပဲ သက်ရောက်မှာဖြစ်ပါတယ်။ Code တွေ ပေါင်းစပ်ဖို့လို အပ်လာတဲ့အခါမှာ တစ်ဦးမှာရှိနေတဲ့ Version တွေနဲ့ အခြားတစ်ဦးထံမှာရှိနေတဲ့ Version တွေ ကို အတွဲလိုက်ပေါင်းစပ်ယူ နိုင်ပါတယ်။

Distributed VCS တွေမှာ Version မှတ်တမ်းများ သိမ်းဆည်းခြင်း၊ Branch များခဲ့ယူခြင်း၊ ပြန်လည် ပေါင်းစပ်ခြင်း၊ စတဲ့ အခြေခံလုပ်ဆောင်ရွက်တွေကို Network Communication မလိုပဲ Local Machine ပေါ်မှာပဲ ဆောင်ရွက်နိုင်တဲ့ အတွက် အများကြီးပိုမြန်ပါတယ်။ အားလုံးမှာ Repository မူပွားကိုယ်စိရှိနေတဲ့အတွက် Server ကွန်ပျိုးတာ ပျက်သွားလို့ Source Code တွေ ဆုံးရှုံးရတယ်ဆိုတာမျိုးလည်း မဖြစ်နိုင်တော့ပါဘူး။ ဒီလို အားသာချက်တွေကြောင့် အခါနာက်ပိုင်းမှာ Distributed VCS တွေကို ပိုပြီး အသုံးများလာကြပါတယ်။

### 5.3 – Git

Git ဟာ Linux Kernel ရဲ့ မူလဖန်တီးရင်ဖြစ်တဲ့ Linus Torvalds ကိုယ်တိုင် (၂၀၀၅) ခုနှစ်မှာ စတင်တိတွင်ခဲ့တဲ့ Distributed VCS ဖြစ်ပါတယ်။ Open Source Software တစ်ခုဖြစ်ပြီး လက်ရှိ လူသုံးအများဆုံး VCS လည်းဖြစ်ပါတယ်။ Linus Torvalds နဲ့ Linux Kernel Developer တွေဟာ မူလက BitKeeper လိုပေါ်တဲ့ DVCS တစ်ခုကို အသုံးပြုပြီး Linux Kernel တစ်ခုလုံးကို ထိမ်းသိမ်းလာခဲ့ကြတာ ဖြစ်ပါတယ်။



BitKeeper ဟာ Open Source Software တစ်ခု မဟုတ်ပါဘူး။ Proprietary Software တစ်ခုပါ။ ဒါပေမယ့် Linux Kernel Developer တွေကို ချင်းချက် အနေနဲ့အခမဲ့အသုံးပြုခွင့် ပေးထားခဲ့ပါတယ်။ ၂၀၀၅ ရောက်တဲ့အခါ မှာ ပြဿနာတစ်ချို့ကြောင့် Bitkeeper ရဲ့ မူလပိုင်ရှင်က Linux Kernel Developer တွေ အတွက် ပေးထားတဲ့ အခမဲ့အသုံးပြုခွင့်ကို ပြန်လည် ရတ သိမ်းလိုက်ပါတယ်။ ဒါကြောင့် Linux Kernel Source Code ကို ဆက်လက်ထိမ်းသိမ်းနိုင်ဖို့အတွက် အခြား VCS တစ်ခုနဲ့ အစားထိုးအသုံး ပြုဖို့ လိုအပ်လာပါတယ်။

Linux Kernel Development ရဲ့ ဦးဆောင်သူဖြစ်တဲ့ Linus Torvalds အနေနဲ့ BitKeeper ရဲ့ Distributed သဘော သဘာဝကို နှစ်သက်သဘောကျပါတယ်။ အခြား VCS တွေကို လေ့လာကြည့်တဲ့အခါမှာတော့ အများစုက Centralize VCS တွေ ဖြစ်နေကြပြီး၊ သူလုပ်ချင်တဲ့ အမြန်နှင့်မရရှိပဲ အလုပ်လုပ်ပုံ နေးကျေးကြတာကို တွေ့ရပါတယ်။ ဒါကြောင့် မြန်ဆန်တိကျော့ အလုပ်လုပ်နိုင်တဲ့ VCS တစ်ခု သူကိုယ်တိုင်ရေးဖို့ ဆုံးဖြတ်ပြီး ၂၀၀၅ ခုနှစ် ဖြောက်တဲ့မှာ Git ကို စတင် ဖန်တီးခဲ့ခြင်းပဲဖြစ်ပါတယ်။ ဒီလိုဖန်တီးရာမှာ အမိက္ခားတည်ချက်သုံးချက် ရှိခဲ့ပါတယ်။

1. Distributed VCS ဖြစ်ရမယ်။
2. အမြန်နှင့်ကောင်းရမယ်။
3. ဘယ်လိုအခြေအနေမှာပဲဖြစ်ဖြစ် Data ပျက်စီးဆုံးရှုံးခြင်း လုံးဝမရှိအောင် အာမခံနိုင်တဲ့ စနစ်တစ်ခု ဖြစ်ရမယ်

- ဆိုတဲ့ အချက်တွေပဲဖြစ်ပါတယ်။

Git Installer ကို <http://git-scm.com/downloads> မှာ ရယူနိုင်ပါတယ်။ Linux, Mac, Windows စတဲ့ အမိက OS အားလုံးမှာအသုံးပြုနိုင်ပါတယ်။ Ubuntu Linux မှာဆိုရင်တော့ အောက်ပါအတိုင်း အလွယ်တစ်ကူ Install လုပ်နိုင်ပါတယ် -

```
$ sudo apt-get install git-core
```

Windows အတွက် Official Installer အား **msysgit** လိုခေါ်တဲ့ Package တစ်ခုကိုလည်း အားထိုး အသုံးပြုနိုင်ပါတယ်။ msysgit က Git သာမက Git Bash, Windows Explorer Integration စတဲ့ ဆက်စပ်ပရိုကရမဲ့တွေကိုပါ ထည့်သွင်းပေးတဲ့အတွက် ပိုမိုပြည့်ဖို့တဲ့ Installer ဖြစ်တယ်လို့ ဆိုနိုင်ပါတယ်။ msysgit ကို အောက်ပါလိပ်စာ မှာ Download ရယူနိုင်ပါတယ်။

<http://msysgit.github.io/>

ဒီနေရာမှာ Install လုပ်ပုံအဆင့်ဆင့်ကို တစ်ဆင့်ချင်း ဖော်ပြမနေတော့ပါဘူး။ အသုံးပြုပုံကိုသာ ဆက်လက်ဖော်ပြပေး သွားမှာဖြစ်ပါတယ်။ ပြီးတော့ Git ဟာ Command Line ပရိုကရမဲ့တစ်ခုဖြစ်ပါတယ်။ Graphical User

Interface နဲ့ အသုံးပြုလိုတယ်ဆိုရင်လည်း UI Client တွေ ထပ်မထည့်သွင်းအသုံးပြု နိုင်ပေမယ့် Command Line ကနေအသုံးပြုခြင်းက ပိုမိုတိရောက်ပြီးအလုပ်တွင်စေနိုင်တဲ့အတွက် Command Line ကနေ အသုံးပြုပုံကို သာ ဖော်ပြသွားမှာပါ။

ဒီအခန်းမှာဖော်ပြသွားမယ့် Command တွေဟာ လက်တွေ့ကူးယူစစ်းသပ်လို့ရတဲ့ Command တွေ ဖြစ်ပါတယ်။ ဒါပေမယ့် Operating System အမျိုးအစား Install လုပ်ထားတဲ့ Git Version, File Permission စတဲ့အချက်တွေပေါ်မှတည်ပြီး အချိုက်လဲမှုတွေ ရှိနိုင်တဲ့အတွက်၊ တိုက်ရှိက်ကူးယူ စစ်းသပ်ခြင်းမပြုပဲ အလုပ်လုပ်ပုံသဘာဝကို အမိကထားလေ့လာသင့်ပါတယ်။

## Configuration

Git ကို Install လုပ်ပြီးနောက်၊ စတင်အသုံးပြုနိုင်ဖို့အတွက် အသုံးပြုမယ့်သူရဲ့ အမည်နဲ့ Email လိပ်စာကို Configuration ဖိုင်ထဲမှာ ဖြည့်သွင်းပြီး သတ်မှတ်ပေးရပါတယ်။ Global Config နဲ့ Local Config ဆိုပြီး Configuration ဖိုင် အမျိုး အစား နှစ်မျိုးရှိပါတယ်။ စက်ထဲမှာရှိတဲ့ Git Repository အားလုံးကို သက်ရောက်စေလိုတဲ့ Setting တွေကို Global Config ထဲမှာသတ်မှတ် ပေးရပါတယ်။ သက်ဆိုင်ရာ Repository တစ်ခုကိုသာ သက်ရောက်စေလိုတဲ့ Setting တွေကို တော့ Local Config ထဲမှာ သတ်မှတ်ထားနိုင်ပါတယ်။

Configuration ဖိုင်ထဲမှာ Setting တွေ သတ်မှတ်တယ်ဆိုပေမယ့် ဖိုင်တွေကို လိုက်ဖွင့်ပြီး သတ်မှတ်နေစရာမလိုပါ ဘူး။ Configuration ဖိုင်တွေရဲ့ တည်နေရာဟာ အသုံးပြုတဲ့ OS နဲ့ Install လုပ်စဉ်က ရွေးချယ်ခဲ့တဲ့ အနေအထားပေါ် မှတည်ပြီး တစ်ယောက်နဲ့တစ်ယောက်မတူပဲ ကဲပြားနိုင်ပါတယ်။ Setting တွေသတ်မှတ်ဖို့ အတွက် git config ဆိုတဲ့ Command ကိုသုံးနိုင်ပါတယ်။ အသုံးပြုမယ့်သူရဲ့ အမည်နဲ့ Email လိပ်စာကို Global Config ဖိုင်ထဲမှာ ထည့်သွင်း သွားစေဖို့ အခုလို Command ကို အသုံးပြုနိုင်ပါတယ်။

```
$ git config --global user.name "John Doe"
$ git config --global user.email "me@johndoe.com"
```

Global Config အဖြစ်သတ်မှတ်လိုတဲ့ Setting တွေကို --global Option နဲ့ တွဲဖက်သတ်မှတ်ရခြင်းဖြစ်ပါတယ်။ --global Option မပါရင်တော့ Local Config အဖြစ် သိမ်းဆည်းမှတ်တမ်းတင်သွားမှာ ဖြစ်ပါတယ်။ အမည် သတ်မှတ်ဖို့အတွက် user.name ကိုသုံးရပြီး Email သတ်မှတ်ဖို့အတွက် user.email ကို သုံးရခြင်း ဖြစ်ပါတယ်။ သတ်မှတ်ထားတဲ့ တန်ဖိုးမှန်မှန် ပြည်လည်စီစစ်လိုရင် အခုလိုပြန်လည် စီစစ်နိုင်ပါတယ်။

```
$ git config --global user.name    # => John Doe
```

Git မှာ user.name နဲ့ user.email တို့လို Setting တွေ အများကြီးရှိပါတယ်။ အောက်ပါ Website လိပ်စာ မှာ အသေးစိတ် လေ့လာနိုင်ပါတယ်။

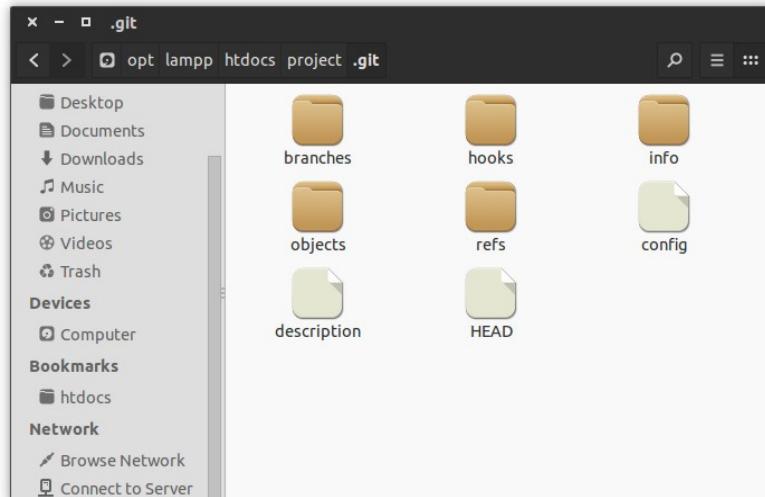
<http://git-scm.com/docs/git-config>

## Initializing a Repository

Git ကိုအသုံးပြုပြီး Version မှတ်တမ်းစီမံနိုင်ဖို့အတွက်၊ ပထမဗီးဆုံးအနေနဲ့ Source Code Directory ကို Git Repository အဖြစ် ကြော်လာတဲ့ မှတ်ပေးရပါတယ်။ ဒီလိုကြော်လာတဲ့မှတ်ဖို့အတွက် git init Command ကို သုံးရပါတယ်။

```
$ git init
Initialized empty Git repository in /path/to/dir/.git/
```

နမူနာမှာပြထားသလို git init ကို Run လိုက်တာနဲ့ .git အမည်နဲ့ Empty Repository တစ်ခုကို Source Code Directory ထဲမှာ တည်ဆောက်လိုက်ကြောင်း Message ပေးလာမှာဖြစ်ပါတယ်။



ပုံ (၅.၄) - Inside .git Directory

.git Directory ကိုဖွင့်ကြည့်လိုက်ရင် ပုံ (၅.၄) မှာဖော်ပြထားသလို တွေ့ရမှာဖြစ်ပါတယ်။ Version မှတ်တမ်း တွေ Branch တွေနဲ့ အခြားလိုအပ်တဲ့ အချက်အလက်တွေကို .git Directory ထဲမှာ သိမ်းဆည်းသွားမှာဖြစ်ပါတယ်။

Linux အပါအဝင် Unix-like OS တွေမှာ Dot နဲ့စတဲ့ File နဲ့ Directory တွေဟာ Hidden File တွေဖြစ်တယ်ဆိတာ ကို သတိပြုပါ။ File Manager နဲ့ ပြန်ကြည့်တဲ့အခါမှာပဲဖြစ်ဖြစ်၊ Terminal ထဲမှာ ကြည့်တဲ့အခါမှာပဲဖြစ်ဖြစ် Hidden File တွေကို ဖော်ပြစေမှ .git Directory ကို တွေရမှာဖြစ်ပါတယ်။

.git Directory ထဲက အရေးကြီးတဲ့ နှစ်ခုကတော့ objects နဲ့ hooks တို့ဖြစ်ပါတယ်။ Git က Version မှတ်တမ်းတွေကို objects Directory ထဲမှာသိမ်းပါတယ်။ Encrypt လုပ်ပြီးသိမ်းမှုဖြစ်တဲ့အတွက် objects Directory ကိုဖွင့်ကြည့်ရင် Hash Value တွေကိုသာ တွေ့ရမှာဖြစ်ပါတယ်။ hooks ကိုတော့ Shell Script တွေ သိမ်း ဖို့သုံးပါတယ်။ ဥပမာ - Version မှတ်တမ်းတစ်ခု သိမ်းဆည်းဖို့အတွက် Commit လုပ်လိုက်တဲ့အခါ တဲ့ဖက်လုပ်ဆောင် သွားစေလိုတဲ့လုပ်ငန်းများ၊ Branch တစ်ခုနဲ့တစ်ခု Merge လုပ်လိုက်တဲ့အခါ ဆောင်ရွက်သွားစေလိုတဲ့ လုပ်ဆာင်ချက်များ စသဖြင့် Shell Script တွေရေးသားပြီး hooks ထဲမှာ သိမ်းဆည်းထားနိုင်ခြင်း ဖြစ်ပါတယ်။

နောက်ထပ်သတိပြုသင့်တဲ့ ဖိုင်တစ်ခုဖြစ်တဲ့ config ကတော့ Local Configuration File ဖြစ်ပါတယ်။ Text Editor နဲ့ ဖွင့်ကြည့်လိုက်ရင် အခုလိုတွေရနိုင်ပါတယ်။

## Config

```
[core]
repositoryformatversion = 0
filemode = true
bare = false
logallrefupdates = true
```

Git နဲ့အလုပ်တစ်ခုလုပ်တိုင်းမှာ ဒါ Configuration ဖိုင်ထဲမှာ ရေးသားထားတဲ့ Setting တွေကို ဦးဆုံးစိစစ် အသုံးပြုပြီး အလုပ်လုပ်ပေးသွားမှုဖြစ်ပါတယ်။ HEAD ဆိတဲ့ ဖိုင်ထဲမှာတော့ လက်ရှိအလုပ်လုပ်နေတဲ့ Branch ကိုညွှန်းထားတဲ့ အညွှန်းလိပ်စာ သိမ်းဆည်းထားပါတယ်။ Version မှတ်တမ်းတစ်ခု သိမ်းဆည်းဖို့ Commit လုပ်လိုက်တိုင်းမှာ HEAD ဖိုင်ထဲက Branch အညွှန်းကိုကြည့်ပြီး Version မှတ်တမ်းကို သိမ်းဆည်းပေးသွားမှုဖြစ်ပါတယ်။

လက်တွေမှာ Version မှတ်တမ်းသိမ်းခြင်း၊ Branch တွေခွဲခြင်းစတဲ့ အခြေခံလုပ်ငန်းတွေလုပ်ဖို့အတွက် .git Directory ကို ပြင်ဖို့မလိုပါဘူး။ အထူးလိုအပ်ချက်ရှိတော့မှာသာ ကိုယ်တိုင်ဖွင့်ပြီးစီမံဖို့လိုမှာပါ။ ဆက်လက်ဖော်ပြုမယ့် Git Command တွေကို Run တဲ့အခါမှာ Source Code Directory ထဲမှာပဲ Run ရပါတယ်။ .git Directory ထဲမှာ Run စရာမလို ပါဘူး။

## Recording and Managing Versions

git init နဲ့ Initialize လုပ်ထားပြီးဖြစ်တဲ့ Git Repository ထဲမှာ လိုအပ်တဲ့ Source Code ဖိုင်နဲ့ Directory တွေ တည်ဆောက်ခြင်း Code ရေးသားခြင်း၊ Compile ပြုလုပ်ခြင်းစတဲ့ လုပ်ငန်းတွေကို လုပ်ရှိုးလုပ်စဉ်အတိုင်း ဆောင်ရွက်နိုင် ပါတယ်။ ရေးသားထားတဲ့ Code တွေကို Version တစ်ခုအဖြစ် မှတ်တမ်းတင်တော့မယ်ဆိုရင် တော့၊ Version မှတ်တမ်းထဲမှာ ပါဝင်စေလိုတဲ့ဖိုင်နဲ့ Directory တွေကို Index ထဲထည့်ပေးရပါတယ်။ နမူနာရစေ

ဖို့အတွက် Git Repository ထဲမှာ ဖိုင်အလွတ် တစ်ချိုက် အခုလိုတည်ဆောက် ကြည့်နိုင်ပါတယ်။

```
$ touch index.html style.css script.js
```

touch Command က File တွေမရှိသေးရင် ဆောက်ပေးတယ်လိုပဲ အလွယ်မှတ်ယူပေးပါ။ touch Command မသုံးချင်ရင်လည်း နှစ်သက်ရာနည်းလမ်းနဲ့ index.html, style.css နဲ့ script.js ဆိုတဲ့ ဖိုင်အလွတ် (၃) ခုကိုတည်ဆောက်ပေးနိုင်ပါတယ်။ ပြီးတဲ့အခါ git status Command ကို Run ကြည့်လိုက်ရင် အခုလို ရလဒ်ကို တွေ့ရမှာဖြစ်ပါတယ်။

```
$ git status

On branch master
Initial commit
Untracked files:
  (use "git add <file>..." to include in what will be committed)

    index.html
    script.js
    style.css

nothing added to commit but untracked files present (use "git add" to track)
```

git status က Repository ရဲ့လက်ရှိအခြေအနေကို လေ့လာဖို့ အသုံးပြုရခြင်းဖြစ်ပါတယ်။ ပြန်ပေးလာတဲ့ Message ရဲ့ နောက်ဆုံးလိုင်းကို လေ့လာကြည့်ရင် Version တစ်ခုအဖြစ် မှတ်တမ်းတင်စရာဘာမှ မရှိသေးတဲ့ အကြောင်း၊ ဒါပေမယ့် Index ထဲ မထည့်ရသေးတဲ့ Untracked files တွေရှိနေတဲ့အတွက် git add Command ကိုသုံးပြီး Track လုပ်ပေးသင့်အကြောင်း ဖော်ပြထားတာကို တွေ့ရမှာဖြစ်ပါတယ်။

git add index.html, git add script.js စသာဖြင့် ဖိုင်တစ်ခုချင်းကို Index ထဲထည့်ပေးသွားလို ရပါတယ်။ ဖိုင်အားလုံးကို Index ထဲ ထည့်လိုရင်တော့ git add . လို့ ပြောလိုက်ရင်ရပါတယ်။ Dot သက်တလေးက လက်ရှိ Directory ထဲက ဖိုင်အားလုံးဆိုတဲ့ အဓိပါယ်ဖြစ်ပါတယ်။

```
$ git add .
$ git status

On branch master
Initial commit
Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

    new file:   index.html
    new file:   script.js
    new file:   style.css
```

git add . နဲ့ ဖိုင်အားလုံး Index ထဲထည့်ပြီးနောက် git status ကို ပြန် Run ကြည့်တဲ့အခါ Version မှတ်တမ်းတင်ဖို့အတွက် ဖိုင်အသစ် (၃) ခု ရှိနေကြောင်း ပြောလာမှာဖြစ်ပါတယ်။

Index ထဲမှာ ဖိုင်အသစ် (၃) ခုရှိနေပြီး Version မှတ်တမ်းတင်ဖို့ အသင့်ဖြစ်နေပြီး git commit Command နဲ့ မှတ်တမ်းတင်နိုင်ပါပြီ။ အဲဒီလိုမှတ်တမ်းတင်ရာမှာ မှတ်ချက် (Comment) ပေးရပါတယ်။ မှတ်ချက်ကို နှစ်သက်သလိုပေး လိုဂျပ်မယ့် လက်ရှုမှတ်တမ်းတင်တော့မယ့် Version မှာဘာတွေတိုးလာသလဲ၊ ဘာတွေပြောင်းသွားသလဲ၊ သတိပြုသင့်တာ တွေက ဘာတွဲလ စသဖိုင် ရှင်းပြထားတဲ့ မှတ်ချက်ဖြစ်သင့်ပါတယ်။

```
$ git commit -m "First commit"

[master (root-commit) 11bf458] First commit
 3 files changed, 0 insertions(+), 0 deletions(-)
  create mode 100644 index.html
  create mode 100644 script.js
  create mode 100644 style.css
```

နဲ့မူနာမှာ git commit Command ကို Run တဲ့အခါ -m Option ကိုသုံးပြီး မှတ်ချက် Comment ကို တစ်ခါတည်း တွဲဖက် ထည့်သွင်းပေးလိုက်ခြင်းဖြစ်ပါတယ်။ အဲဒီလို တစ်ခါတည်းမပေးခဲ့ရင် Git က မှတ်ချက် Comment ပေးဖို့ သီးခြားထပ်မံ တောင်းဆိုလာမှာဖြစ်ပါတယ်။

ဆက်လက်ပြီး၊ နဲ့မူနာအနေနဲ့ index.html ကို ပြင်ပြီး git status ပြန် Run ကြည့်နိုင်ပါတယ်။

```
$ echo 'Hello, world!' > index.html
$ git status

On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

    modified:   index.html

no changes added to commit (use "git add" and/or "git commit -a")
```

index.html ဖိုင်ကို ဖွင့်ပြင်နေမယ့်အစား echo Command ကိုသုံးပြီး 'Hello, world!' ဆိုတဲ့စာကို index.html ထဲမှာ ထည့်သွင်းလိုက်စေခြင်းဖြစ်ပါတယ်။ echo မသုံးချင်ရင်လည်း နှစ်သက်ရာနည်းလမ်းနဲ့ ဖိုင်ကိုဖွင့်ပြင်နိုင်ပါတယ်။

နဲ့မူနာလေ့လာ ကြည့်ရင် git status က index.html ပြင်ထားတာကို သီးခြားပြောင်း ဖော်ပြတာကို တွေ့ရမှာ ဖြစ်ပါတယ်။ ပြန်ပေးလာတဲ့ Message ရဲ့ နောက်ဆုံးလိုင်းကို ကြည့်ရင် git add နဲ့ ပြင်ထားတဲ့ပိုင်ကို Index ထဲ ပြန်ထည့်ပေးဖို့ ပြောထားတာကိုတွေ့ရနိုင်ပါတယ်။ ဒါမှမဟုတ် git commit -a ကို သုံးပြီး တစ်ခါတည်း Commit လုပ်လိုက လုပ်နိုင်ကြောင်းလည်း ပြောထားပါသေးတယ်။ Index ထဲကို တစ်ကိုမဲ့ထည့်

ထားဖူးတဲ့ဖိုင်ကို နောက်တစ်ခါ git add နဲ့ ထပ်ထည့် မနေတော့ပဲ Commit လုပ်တဲ့အခါကျမှ add လုပ်ပေးဖို့ -a Option နဲ့ တွဲပြီး အသုံးပြုနိုင် ခြင်းဖြစ်ပါတယ်။

```
$ git commit -am "Updated index.html"

[master f76103d] Updated index.html
 1 file changed, 1 insertion(+)
```

git commit ကို Run တဲ့အခါ -a Option ရော -m Option ပါနှစ်ခုလုံးပါအောင် -am လို့ တွဲပြီးအသုံးပြုလိုက် ခြင်းဖြစ်ပါတယ်။ Git က ဖိုင် (၁) ခု ပြောင်းသွားကြောင်းနဲ့ Code Line တစ်လိုင်း တိုးသွားကြောင်း ပြန်လည် ဖော်ပြလာတာကို တွေ့ရမှာပဲ ဖြစ်ပါတယ်။ အခါ ဆိုရင် Commit နှစ်ကြိမ်ပြုလုပ်ခဲ့ပြီဖြစ်လို့ ကျွန်ုတ်တို့မှာ Version နှစ်ခုရရှိသွားပြီဖြစ်ပါတယ်။ git log Command နဲ့ Version မှတ်တမ်းကို ပြန်လည်လေ့လာနိုင်ပါတယ်။

```
$ git log

commit f76103d35ad7b2a2641bf16d1baf6de27a5981cf
Author: Ei Maung <eimg@fairwayweb.com>
Date:   Tue Mar 17 15:42:02 2015 +0630

    Updated index.html

commit 11bf458a3a3cc45e65e5c20e16c59d92d66038e7
Author: Ei Maung <eimg@fairwayweb.com>
Date:   Tue Mar 17 12:48:56 2015 +0630

    First commit
```

git log Command က မှတ်တမ်းတင်ခဲ့တဲ့ Version တွေကို စာရင်းနဲ့ဖော်ပြလာခြင်းဖြစ်ပါတယ်။ Version တစ်ခု စီမှာ စာလုံး (၄၀) ပါဝင်တဲ့ Commit Hash ကိုယ်စိုက်ပါတယ်။ Commit Hash ဆိုတာ အဲဒီ Version ရဲ့ ID ပဲဖြစ်ပါ တယ်။ နောင်အဲဒီ Version ကို ပြန်လိုက်ရင် Commit Hash ကို အသုံးပြုပြီး ပြန်လည်ရယူနိုင်ပါတယ်။ Log ထဲမှာ Commit Hash နဲ့အတူ Commit ပြုလုပ်ခဲ့သူ၊ Commit ပြုလုပ်ခဲ့တဲ့အချိန်နဲ့ Commit Comment တို့ကို စာရင်းထဲမှာ ဖော်ပြနေတာကို တွေ့ရမှာဖြစ်ပါတယ်။ git log ကို သပ်ရပ်သွားအောင် --pretty Option နဲ့ တွဲပြီး သုံးနိုင်ပါ တယ်။

```
$ git log --pretty=oneline

f76103d35ad7b2a2641bf16d1baf6de27a5981cf Updated index.html
11bf458a3a3cc45e65e5c20e16c59d92d66038e7 First commit
```

--pretty=oneline လို့ပြောလိုက်တဲ့အတွက် Log ကိုဖော်ပြရမှာ စောစောကလို ရှုပ်ရှက်ခတ်နေအောင် မပြတော့ ပဲ Commit တစ်ခု တစ်ကြောင်းနှင့်နဲ့ ဖော်ပြမှာဖြစ်ပါတယ်။ oneline အစား စိတ်ကြိုက် Format နဲ့

အစားထိုးသုံးနိုင်ပါတယ်။

```
$ git log --pretty=format:"%h - %an, %ar : %s"
f76103d - Ei Maung, 17 minutes ago : Updated index.html
11bf458 - Ei Maung, 3 hours ago : First commit
```

%h Placeholder က Short Commit Hash ဆိုတဲ့အဓိပါယ်ပါ။ Commit Hash ရဲ့ စာလုံး (၄၀) လုံးကို မဖော်ပြတော့ အတိုကောက်အနေနဲ့ ရှေ့ဆုံး (၇) လုံးကိုသာဖော်ပြတော့မှာ ဖြစ်ပါတယ်။ %an က Author Name, %ar က Author Date Relative နဲ့ %s ကတော့ Commit Comment Subject ဆိုတဲ့ အဓိပါယ်ပဲဖြစ်ပါတယ်။ Version တစ်ခုနဲ့တစ်ခု ကူးပြောင်းလိုတဲ့အခါ Short Commit Hash ကို သုံးပြီးတော့လည်း ကူးပြောင်းနိုင်ပါတယ်။ ဥပမာ - ပထမဆုံး Version ကိုပြန်သွားချင်ရင်၊ သူရဲ့ Short Commit Hash က 11bf458 ဖြစ်တဲ့အတွက် အခုလို ပြန်သွားနိုင်ပါတယ်။

```
$ git checkout 11bf458
Note: checking out '11bf458'.
You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by performing another checkout.
If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -b with the checkout command again. Example:
  git checkout -b new_branch_name

HEAD is now at 11bf458... First commit
```

နမူနာကိုလေ့လာကြည့်ရင် Git က လက်ရှိမှာ Commit Hash 11bf485 ကို ပြောင်းလဲအသုံးပြုနေကြောင်း၊ ဒီ Commit ကို Branch အဖြစ် ခွဲထုတ်လိုတယ်ဆိုရင်လည်း git checkout -b နဲ့ ခွဲထုတ်နိုင်ပြောင်းအသိပေးလာတာကို တွေ့ရမှာ ဖြစ်ပါတယ်။

တစ်ချိုအရေးပါတဲ့ Commit တွေကို နောင်ကူးပြောင်းအသုံးပြုရ လွယ်ကူစေဖို့အတွက် Tag လုပ်ထားနိုင်ပါတယ်။

```
$ git tag v1
```

လက်ရှိရောက်ရှိနေတဲ့ Commit ဖြစ်တဲ့ 11bf485 ကို v1 လို Tag လုပ်ပြီး အမည်ပေးလိုက်ခြင်းဖြစ်ပါတယ်။

နောက်ဆုံး Version ကို ပြန်သွားလိုရင် git checkout master ကို Run ပြီး ပြန်သွားနိုင်ပါတယ်။ ဒါကြောင့် git checkout master နဲ့ နောက်ဆုံး Version ကိုပြန်သွားပြီး git log Run ကြည့်ရင် အခုလို တွေ့ရမှာဖြစ်ပါတယ်။

```
$ git checkout master
$ git log --pretty=format:'%h %s %d'

f76103d Updated index.html (HEAD, master)
11bf458 First commit (tag: v1)
```

%d Placeholder က Reference Name ဆိတဲ့အမိပါယ်ပါ။ နေ့နာမှာတွေ့လှုံးရင် Commit 11bf458 ကို v1 လို Tag လုပ်ထားကြောင်းတွေ့နှင့်ပါတယ်။ ဒါကြောင့်အဲဒါ Commit ကိုသွားချင်ရင် အခုလိုအလွယ်တစ်ကူသွားလို့ ရသွားပါတယ်။

```
$ git checkout v1

Note: checking out 'v1'.
You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by performing another checkout.
If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -b with the checkout command again. Example:
  git checkout -b new_branch_name

HEAD is now at 11bf458... First commit
```

ဒီနည်းနဲ့ အရေးပါတဲ့ Commit တွေကို Tag လုပ်ပြီး မှတ်သားထားနိုင်ပါတယ်။ ပေးထားတဲ့ Tag ကို ပြန်ဖျက်ချင်ရင် တော့ git tag Command ကို -d Option နဲ့ တွဲသုံးနိုင်ပါတယ်။

```
$ git checkout master
$ git tag -d v1
$ git log --pretty=format:'%h %s %d'

f76103d Updated index.html (HEAD, master)
11bf458 First commit
```

-d Option က မှတ်တမ်းတင်ထားတဲ့ Commit ကိုဖျက်တာမဟုတ်ပါဘူး။ Commit ကို Reference လုပ်ထားတဲ့ Tag ကို ဖျက်ပေးခြင်းဖြစ်ပါတယ်။

## Working with Branch

VCS တွေမှာ ပါဝင်လေ့ရှိတဲ့ Branch လုပ်ဆောင်ချက်ရဲ့အသုံးဝင်ပုံကို အထက်မှာဖော်ပြခဲ့ပြီးဖြစ်ပါတယ်။ Git မှာ Branch တွေစီမံခွိုင်းအတွက် git branch Command ကို သုံးပါတယ်။

```
$ git branch
* master
```

လောလောဆယ် Branch တွေမရှိသေးတဲ့အတွက် git branch ကို Run ကြည့်တဲ့အခါ master တစ်ခုပဲရှိနေကြောင်း ဖော်ပြခြင်း ဖြစ်ပါတယ်။ လက်ရှိရောက်ရှိနေတဲ့ Version ကို Branch အသစ်တစ်ခုအဖြစ် အခုလို ခွဲထုတ်နိုင်ပါတယ်။

```
$ git branch dev
$ git branch

  dev
* master
```

git branch dev လို့ ပြောလိုက်တဲ့အချိန်မှာ dev ဆိုတဲ့ Branch မရှိသေးတဲ့အတွက် Git က dev Branch ကို အလိုအလျောက် တည်ဆောက်ပေးသွားပါတယ်။ ဒါကြောင့် နောက်တစ်ကြိမ် git branch Run ကြည့်တဲ့ အခါမှာ dev နဲ့ master ဆိုပြီး Branch နှစ်ခု ဖြစ်သွားတာကို တွေ့ရမှာ ဖြစ်ပါတယ်။ master ရဲ့ ရောက Asterisk (\*) က လက်ရှိ master Branch နဲ့ အလုပ်လုပ်နေကြောင်း ဖော်ပြခြင်းဖြစ်ပါတယ်။

အသစ်တည်ဆောက်ထားတဲ့ dev Branch ကို git checkout နဲ့ ကူးပြောင်းနိုင်ပါတယ်။

```
$ git checkout dev

Switched to branch 'dev'

$ git branch

* dev
  master
```

ဒီနည်းနဲ့ Branch တွေ လိုသလိုတည်ဆောက်ပြီး အပြန်အလှန် ကူးပြောင်းအလုပ်လုပ်နိုင်ပါတယ်။ Branch အသစ်တစ်ခု ဆောက်ပြီး ချက်ချင်းကူးပြောင်းသွားစေတဲ့ Short Cut အနေနဲ့ အခုလိုလည်း အသုံးပြုနိုင်ပါတယ်။

```
$ git checkout -b beta

Switched to a new branch 'beta'

$ git branch

* beta
  dev
  master
```

ဒီနည်းနဲ့ beta ဆိုတဲ့ Branch ကို တည်ဆောက်လိုက်ယုံမက တစ်ခါတည်း ကူးပြောင်းပေးသွားပါတယ်။ တည်ဆောက်ထားတဲ့ Branch တစ်ခုကို ပြန်လည်ပယ်ဖျက်စေလိုရင် git branch ကို -d Option နဲ့ တွဲသုံး နိုင်ပါတယ်။

```
$ git branch -d dev
Deleted branch dev (was f76103d) .
```

Branch တစ်ခုကို ကူးပြောင်းပြီးနောက် ပြုလုပ်တဲ့ Commit တွေဟာ လက်ရှိရောက်ရှိနေတဲ့ Branch ကိုသာ သက် ရောက်တော့မှာ ဖြစ်ပါတယ်။

```
$ git checkout beta
$ echo 'body { margin: 0 }' > style.css
$ git commit -am "Updated style.css"

[beta d731a7f] Updated style.css
 1 file changed, 1 insertion(+)
```

```
$ git log --pretty=oneline

d731a7fcdd7dd5a478e2532e1d254a0c69184176 Updated style.css
f76103d35ad7b2a2641bf16d1baf6de27a5981cf Updated index.html
11bf458a3a3cc45e65e5c20e16c59d92d66038e7 First commit
```

beta Branch မှာ style.css ဖိုင်ကိုပြင်ပြီး Commit လုပ်လိုက်ပါတယ်။ ပြီးတော့ Log ပြန်ကြည့်တဲ့အခါ Version သုံးခု ဖြစ်သွားတာကို တွော်မှာဖြစ်ပါတယ်။ master Branch ကို ပြန်သွားပြီး Log ပြန်ကြည့်တဲ့ အခါ မှာတော့ Version နှစ်ခုသာ ရှိတာကို တွော်မှာပဲ ဖြစ်ပါတယ်။

```
$ git checkout master
$ git log -pretty=oneline

f76103d35ad7b2a2641bf16d1baf6de27a5981cf Updated index.html
11bf458a3a3cc45e65e5c20e16c59d92d66038e7 First commit
```

ဒီနည်းနဲ့ Source Code Repository တစ်ခုတည်းကို Branch တွေအပိုးမျိုးခွဲပြီး စီမံဆောင်ရွက်နိုင်မှာပဲဖြစ်ပါတယ်။ Branch တစ်ခုပေါ်မှာ သိမ်းဆည်းထားတဲ့ Version တွေကိုနောက် Branch တစ်ခုနဲ့ ပေါင်းစပ်လိုရင်တော့ git merge ကိုသုံးပြီး ပေါင်းစပ်နိုင်ပါတယ်။

```
$ git checkout master
Switched to branch 'master'

$ git merge beta

Updating f76103d..d731a7f / Fast-forward
  style.css | 1 +
  1 file changed, 1 insertion(+)
```

master Branch ပေါ်မှာ git merge beta ကို Run လိုက်တဲ့အတွက် beta Branch ပေါ်က Version တွေ ဟာ master Branch နဲ့ပေါင်းစပ်သွားမှာပဲဖြစ်ပါတယ်။ ဒါကြောင့် master Branch မှာ Log ကြည့်မယ်ဆိုရင် Version သုံးခုဖြစ်သွားတာကို တွေ့ရမှာပဲဖြစ်ပါတယ်။

```
$ git log --pretty=oneline
d731a7fcdd7dd5a478e2532e1d254a0c69184176 Updated style.css
f76103d35ad7b2a2641bf16d1baf6de27a5981cf Updated index.html
11bf458a3a3cc45e65e5c20e16c59d92d66038e7 First commit
```

## Merge Conflicts

Branch တွေ Merge လုပ်ရမှာ တစ်ခါတစ်ရုံ Conflict တွေရှိတဲ့ပါတယ်။ Merge လုပ်လိုတဲ့ Branch နှစ်ခုလုံးက တူညီ တဲ့ဖိုင်တစ်ခုကို ပြင်ဆင်ထားမိတဲ့အခါမှာ ဖြစ်လေ့ရှုပါတယ်။ စမ်းသပ်ကြည့်နိုင်ဖို့အတွက် master Branch ပေါ်က script.js ကို အခုလို ပြင်ဆင်ပြီး Commit လုပ်လိုက်ပါမယ်။

```
$ git checkout master
Switched to branch 'master'

$ echo 'var app = {}' > script.js
$ git commit -am "Updated script.js in master"

[master 281e6f3] Updated script.js in master
 1 file changed, 1 insertion(+)
```

ပြီးတဲ့အခါ beta Branch ကိုကူးပြီး script.js ကိုပဲ ပြင်ပြီး Commit လုပ်လိုက်ပါမြို့းမယ်။

```
$ git checkout beta
Switched to branch 'beta'

$ echo 'var app = []' > script.js
$ git commit -am "Updated script.js in beta"

[beta 9147870] Updated script.js in beta
 1 file changed, 1 insertion(+)
```

master Branch ကို ပြန်သွားပြီး beta နဲ့ Merge လုပ်ခိုင်းတဲ့အခါ Conflict ပြဿနာကို တွေ့ရမှာပဲ ဖြစ်ပါတယ်။

```
$ git checkout master
Switched to branch 'master'

$ git merge beta
Auto-merging script.js
CONFLICT (content): Merge conflict in script.js
Automatic merge failed; fix conflicts and then commit the result.
```

script.js ကို master ကြောင် beta ကပါ ပြင်ထားတဲ့အတွက် ဘာကိုအတည်ယူရမလဲ Git က မဆုံးဖြတ်နိုင်လို့ ဒီ ပြဿနာတက်ရ ခြင်းဖြစ်ပါတယ်။ script.js ကို ဖွင့်ကြည့်ရင် အခုလို တွေ့ရမှာပါ။

```
<<<<< HEAD
var app = {}
=====
var app = []
>>>>> beta
```

Conflict ညီနေတဲ့နေရာကို နှစ်ပိုင်းခွဲပြီး master script.js ထဲက Code နဲ့ beta script.js ထဲက Code တို့ကို ရောပေးထားပါတယ်။ ကိုယ်အတည်ပြုလိုတဲ့ အနေအထားဖြစ်အောင် Code ဖိုင်ကို စိတ်တိုင်းကျပြင်ပါ။ ပြီးရင် Commit လုပ်ပေးခြင်းအားဖြင့် Conflict ကို Resolve လုပ်ပြီး ဖြစ်သွားပါလိမ့်မယ်။

```
$ git commit -am "Merge with conflict resolve"
[master 08582cf] Merge with conflict resolve
```

ခုနေ Log ခေါ်ကြည့်ရင် အခုလိုတွေ့ရမှာပဲ ဖြစ်ပါတယ်။

```
$ git log --pretty=oneline
08582cf9e218e12b74ff24711692ba2eфеa936dd Merge with conflict resolve
91478702aебe0fe49d667316b11c6fe4bc50062f Updated script.js in beta
281e6f362f37beb9384bf0ebf15605f81f21582e Updated script.js in master
d731a7fcdd7dd5a478e2532e1d254a0c69184176 Updated style.css
f76103d35ad7b2a2641bf16d1baf6de27a5981cf Updated index.html
11bf458a3a3cc45e65e5c20e16c59d92d66038e7 First commit
```

master မှာ script.js ကိုပြင်ပြီး Commit လုပ်ခဲ့တဲ့ Version, beta မှာ script.js ကိုပြင်ပြီး Commit လုပ်ခဲ့တဲ့ Version နဲ့ Conflict ကို Resolve လုပ်ပြီး Commit လုပ်ထားတဲ့ Version အားလုံးကို မှတ်တမ်းတင်ပြီး ဖြစ် နေတာကို တွေ့ရမှာပဲ ဖြစ်ပါတယ်။

## Clone Repository

Repository တစ်ခုကို မူဖားယူဖို့အတွက် git clone ကို သုံးနိုင်ပါတယ်။ git clone နဲ့ ကိုယ့်စက်တဲ့က Repository တစ်ခုကို မူဖားယူနိုင်သလို၊ Network သို့မဟုတ် အင်တာနှင်ပေါ်က Repository ကိုလည်း မူဖားယူ နိုင်ပါတယ်။ git clone နောက်ကနေ မူဖားယူလိုတဲ့ Repository ရဲ့ တည်နေရာလိပ်စာကို ညွှန်းပေးဖို့ပဲ လိုပါ တယ်။ ဥပမာ –

```
$ git clone path/to/repository new_repo
$ git clone smb://192.168.1.100/repository new_repo
$ git clone ssh://server/path/to/repository new_repo
$ git clone https://github.com/user/repository new_repo
```

git clone နဲ့အတူ မူရင်း Repository ရဲ့လိပ်စာပဲပေးမယ်ဆိုရင် မူဖားရဲ့အမည်ကိုလည်း မူရင်း Repository အမည် အတိုင်းထားပေးသွားမှာဖြစ်ပါတယ်။ ဒါမှမဟုတ် နမူနာမှာပြထားသလို git clone နောက်ကနေ မူရင်း Repository လိပ်စာနဲ့အတူ မူဖားအတွက်ပေးလိုတဲ့ အမည်ကိုလည်း တဲ့ဖက်ပေးလိုရင် ပေးနိုင်ပါတယ်။

```
$ cd ~
$ git clone /opt/lampp/htdocs/project

Cloning into 'project'...
done.

$ ls project

index.html  script.js  style.css
```

နမူနာအနေနဲ့ git clone နဲ့ ကျွန်ုတ်တို့ စောင့်ကစမ်းသပ်ထားတဲ့ Repository ကို Home Directory ထဲ မူဖားယူထားပါတယ်။ မူဖားရရှိလာတဲ့ project Directory ထဲက ဖိုင်စာရင်းကိုကြည့်လိုက်တဲ့အခါ ကျွန်ုတ် တို့ တည် ဆောက်ထားခဲ့တဲ့ index.html, script.js နဲ့ style.css တို့ ပါဝင်လာတာကို တွေ့ရမှာဖြစ် ပါတယ်။ ဒုအပြင် မူဖားရရှိလာတဲ့ Repository ထဲမှာ git log Run ကြည့်လိုက်ရင် ကျွန်ုတ်တို့ မှတ်တမ်း တင်ထားခဲ့တဲ့ Version အားလုံးလည်း ပါဝင်လာတာကို တွေ့ရမှာပဲ ဖြစ်ပါတယ်။

```
$ cd project
$ git log --pretty=oneline

08582cf9e218e12b74ff24711692ba2efea936dd Merge with conflict resolve
91478702aebe0fe49d667316b11c6fe4bc50062f Updated script.js in beta
281e6f362f37beb9384bf0ebf15605f81f21582e Updated script.js in master
d731a7fcdd7dd5a478e2532e1d254a0c69184176 Updated style.css
f76103d35ad7b2a2641bf16d1baf6de27a5981cf Updated index.html
11bf458a3a3cc45e65e5c20e16c59d92d66038e7 First commit
```

git branch Run ကြည့်ရင်တော့ master Branch တစ်ခုသာပါဝင်လာတာကို တွေ့ရပါလိမ့်မယ်။ ကျွန်ုင် Branch တွေက မပါလာတာတော့မဟုတ်ပါဘူး။ Reference လုပ်ယုံသာလုပ်ထားတာမို့ အခြား Branch တွေကို လိုချင်ရင် git checkout နဲ့ ထပ်မံရယူဖို့ လိုပါတယ်။

```
$ git branch
* master

$ git branch -a

* master
  remotes/origin/HEAD -> origin/master
  remotes/origin/beta
  remotes/origin/master

$ git checkout beta

Branch beta set up to track remote branch beta from origin.
Switched to a new branch 'beta'

$ git branch

* beta
  master
```

နမူနာမှာ လေ့လာကြည့်ပါ။ ပထမဆုံး git branch Run ကြည့်တော့ master တစ်ခုကိုသာ တွေ့ရပါတယ်။ git branch ကို -a Option နဲ့ ထပ် Run ကြည့်တော့မှ မူလ Repository ရဲ့ Branch တွေကို ညွှန်းထားတဲ့ အညွှန်းတွေ ရှိနေတာ တွေ့ရမှာဖြစ်ပါတယ်။ ဒါကြောင့် git checkout နဲ့ beta Branch ကို ရယူပြီး git branch ပြန် Run ကြည့်တဲ့အခါ မှတော့ beta Branch လည်း Branch စာရင်းထဲမှာ ပါဝင်လာတာကို တွေ့ရ မှာပဲဖြစ်ပါတယ်။ ဒါလို့ မူရင်း Repository ရဲ့ Branch တွေကို ညွှန်းထားတဲ့အတွက်ကြောင့် မူရင်း Repository မှာ အပြောင်းအလဲရှိရင် ရှိတဲ့ အပြောင်းအလဲတွေကို git pull Command နဲ့ အလွယ်တစ်ကူ ရယူနိုင်မှာဖြစ်ပါတယ်။

```
$ cd /opt/lampp/htdocs/project
$ touch app.js
$ git add app.js
$ git commit -m "Added app.js"

[master 81f48dc] Added app.js
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 app.js

$ git log --pretty=oneline -3

81f48dc2faddcd42b0c479f53db522b64dd146ec Added app.js
08582cf9e218e12b74ff24711692ba2eфеa936dd Merge with conflict resolve
91478702aebe0fe49d667316b11c6fe4bc50062f Updated script.js in beta
```

နဲ့မူနာအနေနဲ့ မူရင်း Repository ထဲမှာ ဖိုင်တစ်ခုထပ်တိုးပြီး Commit လုပ်ထားပါတယ်။ Log မှာပြန်ကြည့်တဲ့ အခါ ထပ်တိုးလာတဲ့ Commit ကို တွေ့ရမှာဖြစ်ပါတယ်။ git log မှာတွဲသုံးထားတဲ့ -3 Option က နောက်ဆုံး Commit (၃) ခုပဲကြည့်မယ်လို့ ပြောလိုက်တဲ့သဘောပါ။ -3 အစားအခြားတန်ဖိုးတွေနဲ့ အစားထိုး အသုံးပြုနိုင်ပါတယ်။

မူမွားယူထားတဲ့ Clone Repository ထဲမှာ Log ခေါ်ကြည့်ရင်၊ မူရင်း Repository မှာပြုလုပ်ထားတဲ့ နောက်ဆုံး Commit မပါပဲ မူလ Clone လုပ်ယူစဉ်ကပါဝင်လာတဲ့ Commit တွေသာ ရှိနေတာကို တွေ့ရမှာဖြစ်ပါတယ်။

```
$ cd ~/project
$ git log --pretty=oneline -3

08582cf9e218e12b74ff24711692ba2eфеa936dd Merge with conflict resolve
91478702aebe0fe49d667316b11c6fe4bc50062f Updated script.js in beta
281e6f362f37beb9384bf0ebf15605f81f21582e Updated script.js in master
```

ဒါကြောင့် git pull နဲ့ မူရင်း Repository ထံက နောက်ဆုံး Update ကို ရယူကြည့်ပါမယ်။

```
$ git pull origin master

remote: Counting objects: 1, done.
remote: Total 1 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (1/1), done.
From /opt/lampp/htdocs/project
 * branch            master      -> FETCH_HEAD
   08582cf..81f48dc  master      -> origin/master
Updating 08582cf..81f48dc
Fast-forward
 app.js | 0
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 app.js
```

git pull နဲ့ အတူ origin master ကို တွေ့သုံးထားပါတယ်။ origin အကြောင့် ခကာနေမှာပြောပါမယ်။ master ကတော့ Branch အမည်ဖြစ်တဲ့အတွက် master Branch ပေါ်က Update တွေကို ရယူမယ်လို့ ဆိုလိုက် ခြင်းဖြစ်ပါတယ်။ ဒါကြောင့် ရယူလိုတာက အခြား Branch ဆိုရင်လည်း master အစား၊ ရယူလိုတဲ့ Branch အမည်နဲ့ အစားထိုးအသုံးပြုနိုင်ပါတယ်။

အမှန်တော့၊ မူရင်း Repository က နောက်ဆုံး Update တွေ ရယူဖို့အတွက် git fetch ကို သုံးရတာပါ။ ဒါ ပေမယ့် git fetch က ရယူယုံသက်သက်ပဲ အလုပ်လုပ်ပါတယ်၊ ရရှိလာတဲ့ Update ကို လက်ရှိ Clone Repository နဲ့ပေါင်း စပ်ဖို့အတွက် git merge ကို ထပ်သုံးပေးရပါတယ်။ git pull ကတော့ Update တွေ ကို ရယူပေးယုံမက တစ်ခါတည်းလည်း ပေါင်းစပ်ပေးပါတယ်။ ဒါကြောင့် git pull ဆိုတာ git fetch နဲ့ git merge ကို ပေါင်း စပ်ထားတဲ့သဘောလိုလဲ ဆိုနိုင်ပါတယ်။

နောက်ဆုံး Update ရယူပြီး Log ပြန်ကြည့်တဲ့အခါမှာတော့ Clone Repository ထဲမှာလည်း မူရင်း Repository မှာပြုလုပ်ခဲ့တဲ့ နောက်ဆုံး Commit ပါဝင်လာတာကို တွေ့ရမှာပဲ ဖြစ်ပါတယ်။

```
$ git log --pretty=oneline -3
81f48dc2faddcd42b0c479f53db522b64dd146ec Added app.js
08582cf9e218e12b74ff24711692ba2eфеa936dd Merge with conflict resolve
91478702aебe0fe49d667316b11c6fe4bc50062f Updated script.js in beta
```

git pull နဲ့အတူ တွဲသုံးခဲ့တဲ့ origin အကြောင်းအနည်းငယ်ပြောဖို့ လိုပါမယ်။ origin ဆိုတာ git Command မဟုတ်ပါဘူး။ အညွှန်းအမည်တစ်ခုသာဖြစ်ပါတယ်။ မူရင်း Repository တည်ရှိရာ တည်နေရာကို ညွှန်းထား တဲ့ အညွှန်းတစ်ခု ဖြစ်ပါတယ်။ git clone နဲ့ Repository ကို မူးပွားယူလိုက်စဉ်မှာ၊ Git က Clone Repository အတွက် origin ဆိုတဲ့ အမည်နဲ့ မူရင်း Repository တည်ရှိရာအညွှန်းတစ်ခုကို အလိုအလျောက် ထည့်သွင်းပေးသွား ခြင်းဖြစ်ပါတယ်။ git remote Command ကို Run ကြည့်ရင် မူရင်း Repository အညွှန်း စာရင်းကို တွေ့ရမှာဖြစ် ပါတယ်။

```
$ git remote
origin
```

လောလောဆယ် origin အမည်နဲ့ အညွှန်းတစ်ခုသာ ရှိနေပါတယ်။ origin က ဘယ်ကိုညွှန်းထားတာလဲ သိချင်ရင် တော့ git remote show origin လို့ Run ကြည့်နိုင်ပါတယ်။

```
$ git remote show origin
* remote origin
  Fetch URL: /opt/lampp/htdocs/project
  Push URL: /opt/lampp/htdocs/project
```

နှမူနာအရ origin က /opt/lampp/htdocs/project Repository ကို ညွှန်းထားတာတွေ့ရမှာပါ။

Git ရဲ့ ထူးခြားချက်က Repository တစ်ခုမှာ တစ်ခုထက်ပိုတဲ့ Remote အညွှန်းတွေ ရှိနိုင်ခြင်းဖြစ်ပါတယ်။ ဒါ အချက်က Open Source Project တွေမှာ အတော်လေးအသုံးဝင်ပါတယ်။ ဥပမာ – Open Source Project တစ်ခုကို Team A နဲ့ Team B လို့ ခေါ်တဲ့ အဖွဲ့နှစ်ဖွဲ့က သီးခြားစီ Maintain လုပ်နေတယ်ဆိုပါစို့။ ကျွဲ့နှစ်တော်တို့က Team A ရဲ့ Repository ကို ရော Team B ရဲ့ Repository ကိုပါ Git Remote အညွှန်းများအဖြစ် ညွှန်းထားမယ်ဆို ရင်၊ Team A ရဲ့ Repository မှာရှိနေတဲ့ Update တွေရော Team B ရဲ့ Repository မှာရှိနေတဲ့ Update တွေကိုပါ တစ်နေရာထဲ ကနေ ရယူနေလို့ရနိုင်မှာပဲဖြစ်ပါတယ်။

Git Remote အညွှန်းတစ်ခု ထပ်ထည့်ချင်ရင် git remote add နဲ့ ထည့်သွင်းနိုင်ပါတယ်။

```
$ git remote add teamb /var/www/project
$ git remote

teamb
origin
```

git remote add ကိုသုံးပြီး teamb ဆိုတဲ့ Remote အညွှန်းတစ်ခု ထပ်ထည့်လိုက်ပါတယ်။ ဒါကြောင့် git remote ပြန် Run ကြည့်တဲ့အခါ Remote အညွှန်း နှစ်ခုဖြစ်သွားတာကို တွေ့ရမှာပဲဖြစ်ပါတယ်။ teamb ထံက Update ကိုလိုချင်ရင် -

```
$ git pull teamb master
```

- လို ရယူနိုင်ပြီး origin ထံက Update ကိုလိုချင်ရင်တော့ စောစောကလိုပဲ -

```
$ git pull origin master
```

- လို ရယူနိုင်မှာပဲဖြစ်ပါတယ်။ Remote အညွှန်းတစ်ခုကို ပယ်ဖျက်လိုရင် git remote remove ကို သုံးနိုင်ပါတယ်။

```
$ git remote remove teamb
$ git remote

origin
```

git remote remove နဲ့ teamb အညွှန်းကို ပယ်ဖျက်ပြီး git remote ပြန် Run ကြည့်တဲ့အခါ origin အညွှန်းတစ်ခုပဲကျန်တော့တာကို တွေ့ရမှာပဲဖြစ်ပါတယ်။

## Centralize Repository

Git Repository တစ်ခုကို Centralize Repository တစ်ခုအနေနဲ့ တစ်ဦးထက်ပိုတဲ့ Developer တွေက ပိုင်းဝန်း အသုံး ပြနိုင်ပါတယ်။ အဲဒီလိုအသုံးပြနိုင်ဖို့အတွက် Working Directory မပါပဲ Version တွေ၊ Branch တွေနဲ့ အခြားလုံအပ်တဲ့ အညွှန်းတွေ သာပါဝင်တဲ့ Bare Repository ကို အသုံးပြုရပါတယ်။ နည်းလမ်း (၂) မျိုးနဲ့ Bare Repository တစ်ခုကို ဖန်တီးယူနိုင်ပါတယ်။ ပထမနည်းလမ်းကတော့ Directory အလွတ်တစ်ခု ထဲမှာ git init ကို --bare Option နဲ့တဲ့သုံးပြီး Bare Repository အလွတ်တစ်ခုအဖြစ် ကြော်နိုင်ပါတယ်။

```
$ mkdir rockstar && cd rockstar
$ git init --bare
```

```
Initialized empty Git repository in /opt/lampp/htdocs/rockstar
```

rockstar အမည်နဲ့ Directory အလွတ်တစ်ခုတည်ဆောက်ပြီး အဲဒီ Directory ထဲမှာ git init --bare ကို Run ခြင်းအားဖြင့် rockstar Directory ဟာ Git Bare Repository တစ်ခုဖြစ်သွားပါတယ်။ အဲဒီ Repository ကို ဖွင့်တွေ့ည့်လိုက်ရင် ရှိုးရှိုး Git Repository မှာ ပါဝင်လေ့ရှိတဲ့ .git Directory အတွင်းက objects, hooks, branches, config, HEAD စာတဲ့ ဖိုင်နဲ့ Directory တွေကို တွေ့ရှုမှုပဲဖြစ်ပါတယ်။ တနည်းအားဖြင့် rockstar ဟာ Source Code ဖိုင်တွေသိမ်းဆည်းဖို့ မဟုတ်ပဲ။ Version မှတ်တမ်းတွေကိုသာ သိမ်းဆည်းဖို့ ရည်ရွယ် တည်ဆောက်လိုက်တဲ့ Repository တစ်ခု ဖြစ်သွားခြင်း ဖြစ်ပါတယ်။ ဒါကြောင့် Source Code ဖိုင်တွေကို သူထဲမှာ တိုက်ရှိက် ရေးသားတည်ဆောက်ခြင်း မလုပ်ရတော့ပါဘူး။ သူကို Version မှတ်တမ်း သိမ်းဆည်းပေးတဲ့ Centralize Repository အဖြစ်သာ မှတ်ယူရတော့မှာပါ။

Bare Repository တစ်ခုတည်ဆောက်နည်း နောက်တစ်နည်းကတော့ Clone လုပ်ယူစဉ်မှာ --bare Option ကို အသုံးပြုခြင်းပဲ ဖြစ်ပါတယ်။

```
$ cd ~
$ git clone --bare /opt/lampp/htdocs/project

Cloning into bare repository 'project.git'...
done.
```

ဒီနည်းနဲ့ မူလ Source Code တွေနဲ့ Version မှတ်တမ်းတွေ ရှိနေဖြီးသား Repository တစ်ခုကို Bare Repository တစ်ခုအဖြစ် မူယားယူနိုင်ခြင်းဖြစ်ပါတယ်။ နမူနာမှာလေ့လာကြည့်ရင် Git က မူလ Repository ဖြစ်တဲ့ project ကို project.git အမည် ရှိတဲ့ Bare Repository အဖြစ် Clone လုပ်ပေးသွားခြင်းဖြစ်တာကို တွေ့ရှိနိုင်ပါတယ်။ Git Repository Directory တွေကို အမည်ပေးတဲ့အခါ နောက်ဆုံးက .git ထည့်ပြီး ပေးကြလေ့ရှိပါတယ်။ မဖြစ်မနေပေးဖို့ လိုတာမျိုးမဟုတ်ပေမယ့် အမည်ကို ကြည့်လိုက်ယုံနဲ့ Git Repository ဖြစ်ကြောင်း သိသာစေနိုင်လို့ Git Repository အဖြစ်သုံးမယ့် Directory တွေကို အမည်ပေးတဲ့အခါ အသုံးပြုသင့်တဲ့ နည်းလမ်းတစ်ခုပဲဖြစ်ပါတယ်။

တည်ဆောက်ထားတဲ့ Bare Repository ကို Centralize Repository အဖြစ် Project မှာပါဝင်သူအားလုံး Access လုပ်နိုင်မယ့် တစ်နေရာမှာ ထားဖို့လိုပါတယ်။ Developer တစ်ဦးရဲ့စွဲက်ထဲမှာထားပြီး Network Share အဖြစ် Share ပေးလိုက်ရင်လည်း ရပါတယ်။ ဒါမှမဟုတ် ပိုပြီးစနစ်ကျဖော်ရင် သီးခြား Server ကွန်ပျူတာတစ်လုံး မှာထားပြီး SSH သို့မဟုတ် HTTP တို့ကနေ တစ်ဆင့် ရယူနိုင်အောင်လည်း ထားနိုင်ပါတယ်။ ဒီနေရာမှာတော့ SSH Server နဲ့ HTTP Server မှား Setup လုပ်ပုံတို့ကို ထည့်သွင်းမဖော်ပြနိုင်တော့ပါဘူး။ သဘောသဘာဝပိုင်း ကိုသာ ဆက်လက်ဖော်ပြပေးသွား ပါမယ်။

Project မှာပါဝင်သူတစ်ဦးအနေနဲ့ Code တွေ စတင်ရေးသားနိုင်ဖို့အတွက် ပထမဥုံးဆုံးအနေနဲ့ Centralize Repository အဖြစ်သတ်မှတ်ထားတဲ့ Bare Repository ကို Clone လုပ်ယူဖို့လိုပါတယ်။ Clone လုပ်ပုံလုပ်နည်းက ရှိုးရှိုး Repository ကို Clone လုပ်ပုံနဲ့အတူတူပဲ ဖြစ်ပါတယ်။

```
$ git clone /opt/lampp/htdocs/rockstar
Cloning into 'rockstar'...
warning: You appear to have cloned an empty repository.
done.
```

ဘာမှမရှိသေးတဲ့ Repository အလွတ်ဖြစ်နေတယ်ဆိုရင် Git က နမူနာမှာပြထားသလို Waring တစ်ခုပေးနိုင်ပါတယ်။ Clone ရယူထားတဲ့ Local Repository ထဲမှာ Source Code ဖိုင်နဲ့ Directory တွေတည်ဆောက်ခြင်း၊ Code များရေးသားခြင်း နဲ့ Version မှတ်တမ်းများ Commit လုပ်ခြင်းတို့ကို လုပ်ရှိလုပ်စဉ်အတိုင်း ဆက်လက် ဆောင်ရွက် နိုင်ပါတယ်။

Clone မလုပ်ပဲ Git Repository အလွတ်တစ်ခုတည်ဆောက်ပြီး git remote add နဲ့ Centralize Repository ကို အညွှန်းပေးလိုလည်းရပါတယ်။ ဥပမာ -

```
$ git init
Initialized empty Git repository in /home/eimg/rockstar/.git/
$ git remote add origin /opt/lampp/htdocs/rockstar
```

ကိုယ်ရဲ Local Repository ထဲမှာ မှတ်တမ်းတင်ထားတဲ့ Version မှတ်တမ်းကို Centralize Repository ထဲ ပေးပို့သိမ်းဆည်းလိုတဲ့အခါ git push ကို သုံးရပါတယ်။

```
$ git push origin master
Counting objects: 3, done.
Writing objects: 100% (3/3), 228 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To /opt/lampp/htdocs/rockstar
 * [new branch]      master -> master
```

စောောကြည်ခဲ့တဲ့ git pull မှာလိုပဲ origin ဆိုတဲ့ Remote အညွှန်းနဲ့ master ဆိုတဲ့ ပေးပို့လိုတဲ့ Branch တို့ကို git push နဲ့အတူ တဲ့ဖက်ပေးရခြင်း ဖြစ်ပါတယ်။ ကိုယ်က Version မှတ်တမ်းတွေကို Centralize Repository ထံပေးပို့နေသလို အခြား Developer များကလည်း ပေးပို့နေမှာရို့ Centralize Repository ထံကနေ နောက်ဆုံး Update ကိုလည်း ပြန်လည်ရယူဖို့ လိုပါသေးတယ်။ Centralize Repository မှာ Update ရှိနေတာကို မရယူပဲနဲ့လည်း ကိုယ့်ရဲ့ Version တွေကို ပေးပို့လိုရမှာ မဟုတ်ပါဘူး။

```
$ git push origin master
To /opt/lampp/htdocs/rockstar
 ! [rejected]          master -> master (fetch first)
error: failed to push some refs to '/opt/lampp/htdocs/rockstar'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

နမူနာကိုလေ့လာကြည့်ရင် Centralize Repository က ပေးပို့လာတဲ့ Version တွေကို လက်ခံဖို့ ငြင်းပယ်နေတာ ကို တွေ့ရမှာ ဖြစ်ပါတယ်။ Centralize Repository ထဲမှာ Update တွေရှိနေတဲ့အတွက် အဲဒီ Update တွေကို git pull နဲ့ အရင်ရယူပြီးမှ ပြန်လည်ပေးပို့ဖို့လည်း ပြောထားပါသေးတယ်။

```
$ git pull origin master
remote: Counting objects: 4, done.
remote: Compressing objects: 100% (2/2), done.
Unpacking objects: 100% (3/3), done.
From /opt/lampp/htdocs/rockstar
 * branch            master      -> FETCH_HEAD
   6a05a77..b7ab351  master      -> origin/master
Merge made by the 'recursive' strategy.
 LICENSE | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 LICENSE
```

git pull နဲ့ Centralize Repository က နောက်ဆုံး Update ကိုရယူတဲ့အခါ ကိုယ့် Local Version တွေနဲ့ Centralize Repository ကပြန်ပေးလာတဲ့ Version တွေကို Git က Merge လုပ်ပေးသွားမှာဖြစ်ပါတယ်။ ဒီနေရာ မှာ တစ်ခါတစ်ရုံ Conflict တွေရှိတက်ပါတယ်။ Branch တွေ Merge လုပ်တုန်းကလိုပဲ Local မှာ ကိုယ်ပြင်ထားတဲ့ ဖိုင်နဲ့ Centralize Repository က ပြန်ပေးလာတဲ့ နောက်ဆုံး Version ထဲက ပြင်ထားတဲ့ ဖိုင်ချင်းတိုက်နေရင် Merge Conflict ဖြစ်တက်ခြင်း ဖြစ်ပါတယ်။ အဲဒီလို Conflict တွေရှိလာရင်တော့ Conflict ရှိနေတဲ့ ဖိုင်ကို လို သလိုဖွင့်ပြင်ပြီး Commit ပြန်လုပ်ပေးဖို့လိုမှာဖြစ်ပါတယ်။

နောက်ဆုံး Update ရယူပြီးနောက်ကိုယ့်ရဲ့ Local Version Update ကို Centralize Repository ထဲ ပြန်လည်ပေးပို့လိုက ပေးပို့နိုင်ပြီးဖြစ်ပါတယ်။

```
$ git push origin master
Counting objects: 7, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (5/5), 604 bytes | 0 bytes/s, done.
Total 5 (delta 0), reused 0 (delta 0)
To /opt/lampp/htdocs/rockstar
    b7ab351..72044c5  master -> master
```

ဒီနေရာမှာ တစ်ခုသတိပြုရမှာက၊ Bare Repository တစ်ခုထံကိုသာ Push လုပ်လိုပါတယ်။ ရှိရှိ၏ Repository တစ်ခု ကနေ Clone လုပ်ထားရင်တော့ မူရင်၏ Repository ထံက Update ကို Pull လုပ်ယူလိုပေါ်မယ့်၊ ကိုယ့် Update ကိုတော့ ပြန်လည်ပေးပိုလို ရမှာမဟုတ်ပါဘူး။ ဒါကြောင့် Centralize Repository အဖြစ်ထားရှိလိုတဲ့ အခါတွေမှာ Bare Repository တွေကိုသာ Centralize Repository အဖြစ် ထားရှိအသုံးပြုရခြင်းဖြစ်ပါတယ်။

## Other

Git မှာ အခြားအသုံးဝင်တဲ့ အခြေခံလုပ်ဆောင်ချက် တစ်ချို့ ရှိပါသေးတယ်။ Commit ပြုလုပ်လိုက်ပြီးမှာ အဲဒီ Commit ကို ပြန်လည်ပြင်ဆင်လိုတယ်ဆိုရင် --amend Option ကို သုံးနိုင်ပါတယ်။

```
$ git commit --amend -am "Commit comment"
```

နဲ့မူနာမှာပေးထားသလို Commit ပြုလုပ်စဉ်မှာ --amend Option ကိုထည့်သွင်းအသုံးပြုခဲ့ရင် Commit အသစ် အနေနဲ့ မှတ်တမ်းမတင်တော့ပဲ နောက်ဆုံးပြုလုပ်ခဲ့တဲ့ Commit အစား အစားထိုးမှတ်တမ်းတင်သွားမှာဖြစ်ပါတယ်။

အရင်ပြုလုပ်ခဲ့တဲ့ Commit တွေကို အပြီးဖျက်ပြစ်လိုရင်တော့ git reset ကို သုံးနိုင်ပါတယ်။ ဖျက်လိုတဲ့ Commit ၏ Hash ID ကိုသုံးရမှာဖြစ်လို ပြုလုပ်ခဲ့တဲ့ Log ထဲမှာ Commit ID ကိုအရင်ကြည့်ဖို့တော့လိုပါတယ်။

```
$ git log --pretty=format:"%h %s"
72044c5 Merge branch 'master' of /opt/lampp/htdocs/rockstar
9e4b188 Added index.html
b7ab351 Added LICENSE
6a05a77 Added README
```

```
$ git reset --hard b7ab351
HEAD is now at b7ab351 Added LICENSE
$ git log --pretty=format:"%h %s"
b7ab351 Added LICENSE
6a05a77 Added README
```

နဲ့မူနာကိုလေ့လာကြည့်ရင် မူလက Commit (၅) ခုရှိခဲ့ပေါ်မယ့် git reset --hard b7ab351 လိုပြော လိုက်တဲ့ အချိန်မှာ နောက်ဆုံး Version အဖြစ် b7ab351 ကို သတ်မှတ်လိုက်တဲ့အတွက် b7ab351 နောက်ပိုင်း Commit တွေ အားလုံးပျက်သွားမှာဖြစ်ပါတယ်။ ဒီနည်းလမ်းဟာ Source Code တွေ Loss ဖြစ်နိုင်လို ထူးခြားတဲ့ လိုအပ်ချက်မရှိရင် မသုံးသင့်တဲ့နည်းလမ်းတစ်ခုဖြစ်ပါတယ်။

နောက်ထပ်သတိပြုသင့်တဲ့အချက်တစ်ချက် ရှိပါသေးတယ်။ Windows, Mac, Linux စတဲ့ Operating System တွေမှာ သုံးကြတဲ့ Line Ending နည်းစနစ်တွေ ကဲပြားတက်ပါတယ်။ တစ်ချို့က Carriage Return (CR) သကောက်တကို Line Ending အဖြစ် သုံးကြပါတယ်။ တစ်ချို့က Line Feed (LF) သကောက်တကိုသုံးပါတယ်။ တစ်ချို့ကတော့ နှစ်ခုလုံး ပေါင်းပြီး (CRLF) ဆိုပြီး သုံးပါတယ်။ Developer တစ်ယောက်က Linux နဲ့ရေးထားတဲ့ Source Code ကို နောက် Developer တစ်ယောက်က Windows မှာ ဖွင့်လိုက်တဲ့အခါ Line Ending ပြောင်းသွားနိုင်ပါတယ်။ အဲဒီလို ပြောင်းသွားတဲ့အခါ Git က Source Code ကို ပြင်လိုက်တယ်လို့ ယူဆလိုက်မှာပါ။ တစ်ကယ်တစ်းက ပြင်လိုက်တာမဟုတ်ပဲ OS မတူလို့ Line Ending ပြောင်းသွားတာပါ။ ဒီပြဿနာမျိုး မတက်စေဖို့အတွက် Windows အသုံးပြုသူတွေဟာ ဒီ Setting ကို သတ်မှတ်ထားသင့်ပါတယ်။

```
$ git config --global core.autocrlf true
$ git config --global core.safecrlf true
```

Mac သို့မဟုတ် Linux အသုံးပြုသူတွေကတော့ ဒီလို သတ်မှတ်ထားသင့်ပါတယ်။

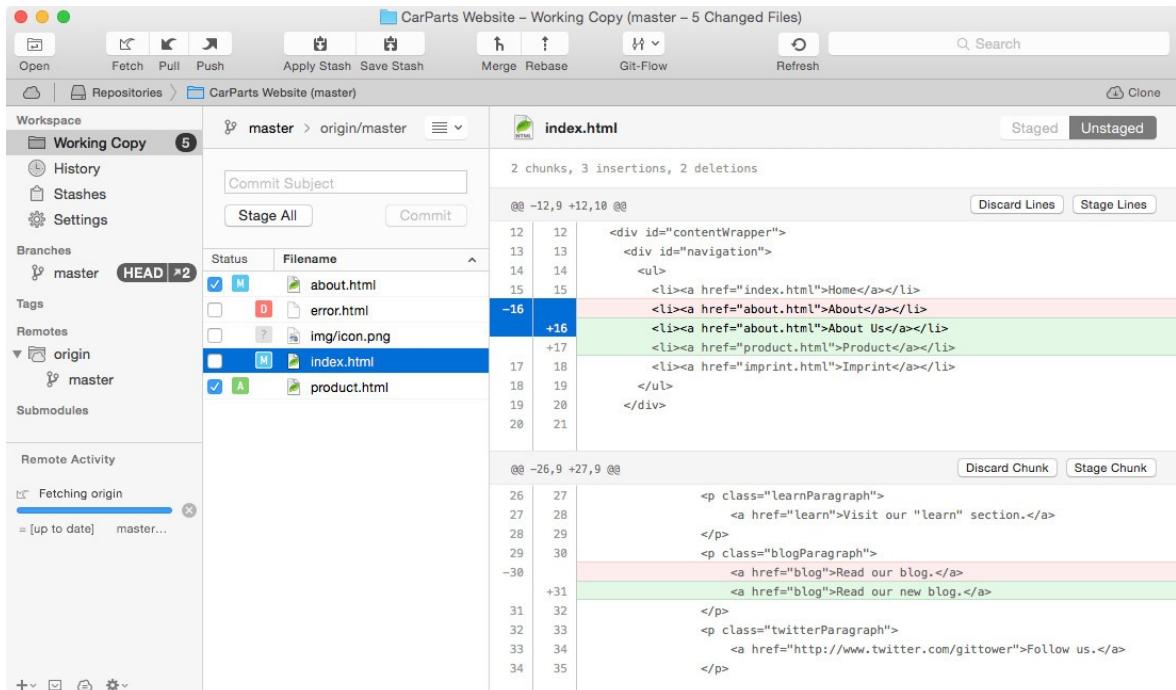
```
$ git config --global core.autocrlf input
$ git config --global core.safecrlf true
```

ဒီတွေမှ Git က Line Ending ကို သင့်တော်အောင် အလိုအလျောက် Convert လုပ်ပြီး အလုပ်လုပ်ပေးသွားမှာဖြစ်လို့ OS မတူလို့ Line Ending ပြောင်းပြီးဖြစ်ရတဲ့ ပြဿနာမဖြစ်တော့မှာပဲဖြစ်ပါတယ်။ Windows Setting မှာ autocrlf true လို့ ပြောထားတဲ့အတွက် Commit လုပ်ချိန်မှာ Line Ending ကို LF ပြောင်းပြီးမှ Commit လုပ်ပေးသွားမှာဖြစ်ပါတယ်။ Checkout လုပ်ချိန်မှာတော့ Windows Line Ending ဖြစ်တဲ့ CRLF ပြောင်းပြီး Checkout လုပ်ပေးသွားမှာပဲဖြစ်ပါတယ်။ Mac နဲ့ Linux အတွက်တော့ autocrlf input လို့ ပြောထားတဲ့အတွက် Checkout မှာ Line Ending Conversion ကို လုပ်မနေတော့ပဲ Commit မှာပဲ Line Ending Conversion ကို လုပ် ပေးသွားမှာဖြစ်ပါတယ်။ safecrlf true Setting ကတော့ Line Ending Conversion လုပ်လိုက်တဲ့အတွက် Source Code ရဲ Integrity ကို ထိခိုက်မသွားအောင်စစ်ဆေးဖို့ သတ်မှတ်ထားခြင်းဖြစ်ပါတယ်။ safecrlf လို့ ပြောထားတဲ့အတွက် Source Code ကို ထိခိုက်နိုင်စရာ ရှိတယ်ဆိုရင် Convert မလုပ်တော့ပဲ Commit တွေကို ငြင်းပယ် သွားမှာဖြစ်ပါတယ်။

နောက်ထပ်သတိပြုစရာတစ်ချက်ကတော့ Indention ဖြစ်ပါတယ်။ Code တွေရေးသားတဲ့အခါ Indent တွေ မဖြစ်မနေ ပါဝင်ကြပါတယ်။ တစ်ချို့က Tab ကို Indent လုပ်ဖို့သုံးပြီး တစ်ချို့ကတော့ Space ကို Indent လုပ်ဖို့သုံးပါတယ်။ ဘာသုံးသုံး တစ်ချို့တည်း သုံးမယ်ဆိုရင်ရပါတယ်။ Tab နဲ့ Space ကို ရောသုံးတဲ့အခါမှာတော့ ပြဿနာရှိတက်ပါတယ်။ Developer တစ်ယောက် က Indent အတွက် Space သုံးထားတယ်။ နောက် Developer တစ်ယောက်က Code ကို ပြင်တာမဟုတ်ပေမယ့် တစ်နေရာမှာ Indent Space အစား Tab ထည့်လိုက်မိတယ်ဆိုရင် Space နဲ့ Indent တွေရော နေပြီး မလိုလားအပ်တဲ့ ပြဿနာတစ်ချို့ ရှိတက်ပါတယ်။ Git ကတော့ Default အနေနဲ့ Tab နဲ့ Space တွေရောပြီးသုံးထားရင် Commit တွေကို လက်မခံပဲ ငြင်းပယ်သွားမှာဖြစ်ပါတယ်။ ဒီပြဿနာကိုရောင်ရှားဖို့အတွင်း Git Setting ကို ပြင်မနေပဲ ရေးသားသူတွေကြားတဲ့မှာ Coding Standard တစ်ခု ထားပြီးတော့ ညီနိုင်းကြဖို့လိုအပ်ပါတယ်။

Git ကို Command Line ကနေမသုံးပဲ User Interface နဲ့ သုံးချင်သူများအနေနဲ့ TortoiseGit, Giggle, Tower စတဲ့ UI Software တွေကို အသုံးပြုနိုင်ပါတယ်။ တစ်ချို့ အခမဲ့ရပြီး တစ်ချို့ လိုင်စင်ဝယ်သုံးရပါတယ်။ အောက်ပါ လိပ်စာများ အသုံးများတဲ့ Git GUI Software စာရင်းကို လေ့လာနိုင်ပါတယ်။

<http://git-scm.com/downloads/guis>



ပုံ (၅.၅) - Git Tower – Git GUI Software on Mac

နောက်ဆုံးတစ်ချက်အနေနဲ့ Git Repository တွေကို အဖွဲ့လိုက်ပူးပေါင်းဆောင်ရွက်နိုင်ဖို့ Centralize Server တစ်ခုနဲ့ Setup လုပ်ရာမှာ Github ([github.com](https://github.com)) နဲ့ BitBucket ([bitbucket.org](https://bitbucket.org)) တို့လို Source Code Hosting Service တွေကိုလည်း လေ့လာအသုံးပြုသင့်ကြောင်း အကြံပြုလိုပါတယ်။ ဒါ Service တွေရဲ့အကူအညီနဲ့ Centralize Server တစ်ခုကို ကိုယ်တိုင် Setup လုပ်နေစရာမလိုပဲ အလွယ်တစ်ကူ ရရှိနိုင်ပါတယ်။

## Software Versioning

ဆက်လက်ဖော်ပြချင်တာကတော့ Software တစ်ခုကို Version နံပါတ်တွေ ပေးပုံပေးနည်းဖြစ်ပါတယ်။ အခြေခံ အားဖြင့် Commit တစ်ခုဟာ Version တစ်ခုဖြစ်တယ်လို့ အကြမ်းဖျင်းအားဖြင့် ပြောနိုင်ပါတယ်။ ဒါပေမယ့် မှတ်သားရေက်တဲ့ Commit Hash တွေကို Version နံပါတ်အဖြစ် သုံးလိုက်တော့ အဆင်မပြေပါဘူး။ အရေးပါတဲ့ Commit တွေကို နားလည်း မှတ်သားရ လွှာယ်တဲ့ Version နံပါတ်တွေ သတ်မှတ်ထားပေးဖို့လိုပါတယ်။ Git ဘဲ Tag လုပ်ဆောင်ချက်ကို ဒီနေရာမှာ အသုံးချိန်ပါတယ်။

Github ကိုပူးတဲ့တည် ထောင်ခဲ့သူတစ်ဦးဖြစ်တဲ့ Tom Preston-Werner ရေးသားတင်ပြထားတဲ့ Semantic

Versioning ဆိုတဲ့ Version နံပါတ်သတ် မှတ်ရာမှာ လိုက်နာသင့်တဲ့ နည်းလမ်းများဆိုတာ ရှုပါတယ်။ သတ်မှတ် ချက်အပြည့်အစုံပါဝင် တဲ့ Version နံပါတ်တစ်ခုဟာ ပဲ (၅.၆) မှာဖော်ပြထားသလို ပုံစံဖြစ်မှာပါ။



ပဲ (၅.၆) - Semantic Version Number

Software တစ်ခု ပထမဆုံးစတင်စမ်းသပ်အသုံးပြုလိုရတဲ့ အဆင့်ကို Version 0.1.0 လိုသတ်မှတ်ရမှာ ဖြစ်ပါတယ်။ အဲဒီအဆင့်မှာ Software က စပြီးစမ်းသုံးလို့ရနေပါပြီ။ ဒါပေမယ့် သတ်မှတ်လုပ်ဆောင်ချက်တွေတော့ စုံလင်အောင် မပါဝင် သေးပါဘူး။ အဲဒီ အဆင့်မှာပဲ အများတစ်ချို့ စတွေလို့ ပြင်ဆင်လိုက်မယ်ဆိုရင် Version 0.1.1, 0.1.2, 0.1.3 စသဖြင့် ပြင်ဆင်ချက်အလိုက် နောက်ဆုံးက PATCH နံပါတ်လည်း လိုက်တိုးလာမှာဖြစ်ပါတယ်။

နောက်ထပ်ဖြည့်စွက်လုပ်ဆောင်ချက်တစ်ချို့ဖြည့်စွက်လိုက်လို့ ပိုပြည့်စုံလာပြီဆိုရင်တော့ Version 0.2.0 အဖြစ် သတ် မှတ်ရမှာဖြစ်ပါတယ်။ 0.2.0 အဆင့်မှာ ပြုလုပ်ခဲ့တဲ့ အများပြင်ဆင်ချက်ကိုတော့ 0.2.1, 0.2.2, 0.2.3 စသဖြင့်အဆင့်လိုက်ပေးသွားရမှာဖြစ်ပါတယ်။ ဒီနည်းနဲ့ 0.3.0, 0.3.2, 0.3.2, 0.4.0, 0.4.1 စသဖြင့် လုပ်ဆောင်ချက် တိုးလာတိုင်း MINOR Version နံပါတ်တိုးလာသလို MINOR Version တစ်ခုချင်းစီမှာ ပြင်ဆင်မှုတွေ လုပ်လိုက် တိုင်း PATCH Version နံပါတ်တွေလည်း တိုးလာမှာပဲဖြစ်ပါတယ်။

ဒီနည်းနဲ့ နောက်ဆုံး အများအသုံးပြနိုင်ဖို့ ကြော်လောက်အောင် ပြည့်စုံလာပြီဆိုရင်တော့ MAJOR Version နံပါတ် တိုးသွားပြီး 1.0.0 ဖြစ်သွားမှာပဲဖြစ်ပါတယ်။ ဒါပေမယ့် အများကိုကြော်တော့မှာမို့ ပြည့်စုံပြီးအများ နည်းစိုးလိုပါတယ်။ အသေအချာမစမ်းသပ်ရသေးတဲ့ အခြေအနေမှာ 1.0.0-pre-alpha လို့ သတ်မှတ်ထားသင့်ပါတယ်။ စမ်းသပ်မှုတွေ စတင်ပြုဆိုရင်တော့ 1.0.0-alpha ကိုယ်တိုင်စမ်းသပ်ယုံမက ပြင်ပ User တွေကိုလည်း စတင်စမ်းသပ်ခွင့်ပြုပြီဆိုရင် 1.0.0-beta စသဖြင့် သတ်မှတ်သွားရမှာဖြစ်ပါတယ်။ အဲဒီ pre-alpha, alpha နဲ့ beta အဆင့်တွေမှာ အများပြင်ဆင်ချက် တွေ ရှုံးခဲ့တယ်ဆိုရင် PRE-RELEASE PATCH နံပါတ်ကို တိုးသွားနိုင်ပါတယ်။ 1.0.0-beta.1, 1.0.0-beta.2 စသဖြင့် ပေးသွားနိုင်ပါတယ်။

စိတ်ကျေနှင်းလောက်အောင် စမ်းသပ်ပြုပြင်မှုတွေဆောင်ရွက်ပြီးလို့ အသင့်ဖြစ်နေပြီဆိုရင်တော့ 1.0.0-rc လို့ သတ်မှတ်နိုင်ပါတယ်။ rc က Release Candidate ဆိုတဲ့အဓိပ္ပာယ်ပါ။ ဒီအတိုင်း Release လုပ်တော့မယ်လို့ ဆုံးဖြတ်လိုက်တဲ့အဆင့်ဖြစ် ပါတယ်။ 1.0.0-rc အဆင့်မှာ ပြဿနာတစ်စုံတစ်ရာမရှိတော့ဘူးဆိုရင်တော့ PRE-RELEASE ကိုဖြူတဲ့လိုက်ပြီး Version 1.0.0 အနေနဲ့ အများ အသုံးပြုနိုင်အောင် ကြော်လှိုင်ပြီးပြုပြီးဖြစ်ပါတယ်။

1.0.0 အဖြစ် ကြော်ပြီးနောက် အသေးစားပြင်ဆင်ချက်တွေကို 1.0.1, 1.0.2 စသဖြင့် ဆက်လက် Release လုပ်သွားနိုင်ပါတယ်။ အသေးစားဖြည့်စွက်ချက်တွေပြုလုပ်ခဲ့ရင်တော့ 1.1.0, 1.2.0 စသဖြင့် ဆက်လက် Release လုပ်သွားနိုင်ပါတယ်။ အဲဒီအဆင့်တွေမှာလည်း 1.1.0-beta, 1.2.0-rc စသဖြင့် PRE RELEASE သတ်မှတ်ချက်တွေ လိုအပ်ရင်တဲ့သုံးနိုင် ပါသေးတယ်။

ပြင်ဆင်ဖြည့်စွက်ချက်တွေ တစ်ဖြည့်ဖြည့်များလာပြီး၊ သိသာမြင်သာတဲ့ နောက်အဆင့်တစ်ခုကို ရောက်လာပြီ ဆိုရင်၊ (သို့မဟုတ်) Software ကို နည်းစနစ်သစ်၊ ပုံစံသစ်တွေနဲ့ ပြန်လည်ဆန်းသစ်တော့မယ်ဆိုရင်တော့ 2.0.0 ဆိုပြီး MAJOR Version နံပါတ်သစ်တစ်ခုနဲ့ ဆက်လက်ဆောင်ရွက်ရမှာဖြစ်ပါတယ်။

Commit Hash (သို့မဟုတ်) Software ကို Build လုပ်လိုက်ရာမှာ ရရှိလာတဲ့ Unique ID (သို့မဟုတ်) Release လုပ်ခဲ့တဲ့ရက်စွဲ စတုအချက်အလက်တွေကို Version နံပါတ်နဲ့တွဲဖော်မယ်ဆိုရင် နောက်ဆုံးကနေ + သက်ကနဲ့တွဲဖော်နိုင် ပါတယ်။ ဒါပေမယ့် အချက်အလက်အနေနဲ့သာ ထည့်သွင်းသင့်ပြီး အဲဒီအချက်အလက်ကို မိုးခိုးအလုပ်လုပ်တာမျိုးတော့ မလုပ်ရပါဘူး။ ဥပမာ – Version 2.5.1-rc.2+72044c5, 1.4.3+20150318 စသဖြင့် တွဲဖော်ပေးနိုင်ပါတယ်။

Software တိုင်းက ဒီနည်းတစ်မျိုးတည်းကို သုံးနေတာတော့ မဟုတ်ပါဘူး။ တစ်ချို့လည်း Version ကို နံပါတ်တင်မက နာမည့်နဲ့လည်း ပေးတက်ကြပါတယ်။ ဥပမာ - Flash MX, Photoshop CS, Windows XP။ တစ်ချို့လည်း ခုနစ်နဲ့ ပေးတက်ကြပါတယ်။ ဥပမာ - Windows 2000, Ubuntu 14.04။ တစ်ချို့လည်း Internal Version နံပါတ်နဲ့ အများ ကို ကြော်တဲ့နံပါတ်ဆိုပြီး နှစ်မျိုးထားတက်ကြပါတယ်။ ဥပမာ - Java7 ရဲ့ တစ်ကယ့် Internal Version နံပါတ်က Java 1.7 ဖြစ်ပါတယ်။ လက်တွေမှာ Version နံပါတ်တွေကို ကိုယ့်နည်းကိုယ့်ဟန်နဲ့ အမျိုးမျိုးပေးတက်ကြပါတယ်။ ဒါကြောင့် အခုဖော်ပြခဲ့တဲ့နည်းအတိုင်းအမြေပေးရမယ်လို့တော့ မဆိုလိုပါဘူး။ ဒါပေမယ့် ဒီနည်းစနစ်ဟာ အများစံထားသုံး နေတဲ့ နည်းစနစ်တစ်ခုဖြစ်လို့ လိုက်နာအသုံးပြုသင့်တဲ့နည်းစနစ်တစ်ခုပဲဖြစ်ပါတယ်။

## Conclusion

Software Project တစ်ခုကို တစ်ဦးတည်းရေးသားသည်ဖြစ်စေ၊ Team နဲ့ ပူးပေါင်းရေးသားသည်ဖြစ်စေ Version Control System တစ်ခုကိုတော့ မဖြစ်မနေ အသုံးပြုသင့်ပါတယ်။ တစ်ချို့လည်း Centralize Code Base လိုအပ်ပြီး အဖွဲ့လိုက်ရေးသားရတဲ့ Project တွေမှာသာ VCS တွေလိုတယ်လို့ အမှတ်မှားကြပါတယ်။ လက်တွေမှာ Version Control System တွေက Centralize Code Base တစ်ခုရရှိအောင် အကူအညီပေးယုံမျှ မက၊ အဆင့်လိုက် မှတ်တမ်းတင်ပေးတဲ့ အတွက် Code Loss မဖြစ်အောင် ကာကွယ်ပေးနိုင်ခြင်းနဲ့ Branch လုပ်ဆောင်ချက်နဲ့အတူ Code Base တစ်ခုတည်းကို သုံးပြီး မတူကွဲပြားတဲ့ Version တွေကို ရေးသားစီမံနိုင်အောင်လည်း ကူညီပေးနိုင်လို့ တစ်ဦးတည်းရေးသားတဲ့ Software Project တွေမှာလည်း အသုံးပြုသင့်ပါတယ်။

နဲ့မူနာအနေနဲ့ လက်ရှိလူသုံးအများဆုံး VCS ဖြစ်တဲ့ Git အကြောင်းကိုဖော်ပြခဲ့ပေယ့် အခြား VCS တွေကို စမ်းသပ် လေ့လာသင့်ပါတယ်။ Subversion (Centralize), Mercurial (Distributed), Bazaar (Distributed) စတဲ့ VCS တွေ ရှိပါသေးတယ်။

Joel Spolsky လိုခေါ်တဲ့ လူသိများထင်ရှားတဲ့ ပညာရှင်တစ်ဦးဖော်ထုတ်ခဲ့တဲ့ "The Joel Test: 12 Steps to Better Code" ဆိုတဲ့ အချက် (၁၂) ချက်ပါဝင်တဲ့ Software Team တစ်ခုရဲ့ အရည်အသွေး တိုင်းတာတဲ့ နည်းလမ်းတစ်ခု ရှိပါတယ်။ အဲဒီ အချက်တွေထဲမှာ နံပါတ် (၁) အချက်က "Do you use source control?" ဆိုတဲ့ အချက်ဖြစ်ပါတယ်။ Source Code တွေစီမံနိုင်တဲ့ စနစ်တစ်ခုကို အသုံးပြုခြင်း၊ မပြုခြင်းဟာ Software Team တစ်ခုရဲ့ Performance ပေါ်မှု အများကြီး သက်ရောက်မှုရှိစေမှာပဲဖြစ်ပါတယ်။

The Joel Test ထဲက ကျွန်ုင်အချက်တွေနဲ့အတူ နောက်ထပ်အရေးကြီးတဲ့ အကြောင်းအရာတစ်ခုဖြစ်တဲ့ Issue Tracking System အကြောင်းကို နောက်အခန်းမှာ ဆက်လက်ဖော်ပြပေးသွားပါမယ်။

ပြင်ရတာမခက်ပေမယ့်၊ သိပ်အရေးမကြီးလို့ နောင်မှ  
ပြင်မထုတိပြီးထားလိုက်တဲ့ Bug ကို၊ မေပြီးမပြင်မိလို့  
အသုံးပြုသူလက်ထဲရောက်မှ အဲဒီ Bug ပြန်ပေါ်တဲ့အတွက်  
ကိုယ့် Software ရဲအရည်အသွေးကိုမေးခွန်းထုတ်စရာ  
ဖြစ်သွားတလောက် နောင်တရဖို့ကောင်းတာ မရှိပါဘူး။

### **Professional Web Developer Course**

HTML5, PHP/MySQL, jQuery/Ajax, Mobile Web စသည်

Professional Web Developer တစ်ဦး သိရှိထားသင့်သည်

နည်းပညာများကို စုစုပေါင်းသင်ကြားခြင်းဖြစ်သည်။

ဆက်သွယ်ရန် - (၀၉) ၇၃၁ ၆၅၄ ၆၂

## အခန်း(၆) – Issue Tracking System

StackOverflow လိုပေါ်တဲ့ Software Developer တိုင်းအားထားရတဲ့ Q&A Platform ကို ယူးတွဲတည်ထောင်သူ တစ်ဦးဖြစ်ပြီး၊ သူ၏ နည်းပညာ ဆောင်းပါးတွေကြောင့် နာမည်ကျော်ကြားတဲ့ Joel Spolsky ဆိုတဲ့ပညာရှင်တစ်ယောက်က Software Development Team တစ်ခုရဲ့ အရည်အသွေးကို အချက် (၁၂) ချက်နဲ့တိုင်းတာနိုင်တယ်လို ဆိုထားပါတယ်။

အဲဒီအချက်တွေကတော့ –

### 1. Do you use source control?

၁။ Source Control (VCS) အသုံးပြုသလား

### 2. Can you make a build in one step?

၂။ Build လုပ်ငန်းကို တစ်ကြိမ်တည်းတစ်ဆင့်တည်းနဲ့ ဆောင်ရွက်နိုင်ရဲ့လား

### 3. Do you make daily builds?

၃။ နေ့စဉ် Build လုပ်ရဲ့လား

### 4. Do you have a bug database?

၄။ Bug Database တစ်ခုထားရှိသလား

### 5. Do you fix bugs before writing new code?

၅။ Code အသစ်မရေးခင် ရှိနေတဲ့ Bug တွေကို ဦးဆုံးရှင်းလင်းတဲ့ အလေ့အထရှိရဲ့လား

### 6. Do you have an up-to-date schedule?

၆။ Up-to-date Schedule ထားရှိသလား

### 7. Do you have a spec?

၇။ Specification တစ်ခု ပြင်ဆင်ရေးသားထားရှိသလား

### 8. Do programmers have quiet working conditions?

၈။ Programmer တွေ အနောင့်အယုက်ကင်းကင်း အလုပ်လုပ်လိုရတဲ့ အခြေအနေရှိရဲ့လား

### 9. Do you use the best tools money can buy?

၉။ အကောင်းဆုံး Development Tool တွေကို အသုံးပြုရဲ့လား

### 10. Do you have testers?

၁၀။ Tester တွေ သီးခြားခန်းထားရဲ့လား

### 11. Do new candidates write code during their interview?

၁၁။ အလုပ်မခန်းခင် လျှောက်ထားသူကို လက်တွေ့ Code ရေးခိုင်းကြည့်ပြီးမှ ခန်းတာလား

### 12. Do you do hallway usability testing?

၁၂။ Hallway Usability Testing လိုပေါ်တဲ့ နည်းစနစ်ကို ကျင့်သုံးရဲ့လား

- စတဲ့ အချက်တွေပြစ်ပါတယ်။ ဒီအချက်တွေထဲက တစ်ချက်ကိုလိုက်နာတိုင်း အဲဒီ Team ရဲ့ စွမ်းဆောင်ရည် ဟာ တစ်ဆင့်တိုး တက်တယ်လို့ ဆိုထားပါတယ်။ အချက်အားလုံးလိုက်နာနိုင်မယ်ဆိုရင်တော့ အရည်အသွေး မြင့် Software Development Team တစ်ခု ဖြစ်မှာပဲပြစ်ပါတယ်။

နံပါတ် (၁) အချက်ဖြစ်တဲ့ **Source Control** ရဲ့အရေးပါမှုအကြောင်းကိုတော့ ပြီးခဲ့တဲ့အခန်းမှာ ဖော်ပြခဲ့ပြီး ဖြစ်ပါတယ်။

နံပါတ် (၂) အချက်ဖြစ်တဲ့ **One-step Build** အကြောင်းကိုတော့ အခန်း (၈) – **Workflow Automation** ရောက်တဲ့ အခါ ဆက်လက်ဖော်ပြပါမယ်။

နံပါတ် (၃) အချက်ဖြစ်တဲ့ **Daily Build** ဆိုတာကဒီလိုပါ။ Version Control System တစ်ခုအကူအညီနဲ့ အဖွဲ့လိုက် ပူးပေါင်းရေးသားကြတဲ့အခါ Developer တစ်ဦးဖြည့်စွက်လိုက်တဲ့ Code က သူအတွက်အဆင်ပြပေ မယ့် အခြား Developer တွေတာဝန်ယူ ဆောင်ရွက်နေတဲ့အပိုင်းကို သွားပြီးထိနိုက်သက်ရောက်နေတက်ပါတယ်။ အဲဒီလိုအခြေအနေမှာ သူပြင်မပေးမချင်း ကျွန်ုတ်သူတွေက ဆက်အလုပ်လုပ်လို့ မရတော့ဘူးဆိုတာမျိုး ဖြစ်နေတက်ပါတယ်။ ဒါကြောင့် စနစ်တစ်ခုလုံး အမြဲအလုပ်လုပ်နေတယ်ဆိုတာ သေချာအောင် နေ့စဉ်စိစစ် ခြင်းလုပ်ငန်းကို ဆောင်ရွက်ဖို့လိုပါတယ်။ တစ်နည်းအားဖြင့် Daily Build အနေနဲ့ နေ့တိုင်း Project ကို Build လုပ်ဖို့လိုအပ်ပါတယ်။ ဒီလိုအပ်ချက်ကို ဖြည့်စည်းဖို့အတွက် VCS ကနေ Commit လုပ်လိုက်တာနဲ့ အလိုအလျောက် Build လုပ်ပေးနိုင်တဲ့ Continues Integration နည်းပညာတွေ ရှိပါတယ်။

နံပါတ် (၄) အချက်ဖြစ်တဲ့ **Bug Database** အကြောင်းကိုတော့ မကြာခင်ဆက်လက်ဖော်ပြပေးသွားပါမယ်။

နံပါတ် (၅) အချက်ဖြစ်တဲ့ **Fix Bugs Before Writing New Code** ဆိုတာက ဒီလိုပါ။ အမှားတစ်ခု (သို့မဟုတ်) Bug တစ်ခုဟာတွေတွေချင်း ပြင်နိုင်လေး ပြင်ရသက်သာလေပါပဲ။ အချိန်ကြာလာတာနဲ့အမှာ လုပ်ဆောင်ချက် တွေများ သထက်များလာပြီး ရှာဖွေပြင်ဆင်ရ ခက်ခဲလာတက်ပါတယ်။ ပြင်ဖို့မပြောနဲ့ အဲဒီအမှားကို ပြန်ရှာရတာတောင် မလွယ် တော့ပဲ ခက်လာတော့မှာပါ။ ရှာတွေပြန်တော့လည်း ပူးပေါင်းနေး လုပ်လက်စတုန်းက

လောက အဲဒီ ပြဿနာရဲ့ရင်းမြစ်နဲ့ သဘာသဘာဝကို သိရှိနိုင်တော့မှာမဟုတ်ပါဘူး။ ဒါကြောင့် လက်တွေမှာ အလုပ်မြန်မြန်ပြီးချင်လို့ ရှိနေတဲ့ အမှားတွေ ကို နောက်မှပြင်မယ်သဘောထားရင် မြန်မြန်မပြီးတဲ့အပြင် ပိုကြာ သွားတက်ပါတယ်။ ဒါကြောင့် လုပ်ဆောင်အချက်အသစ် တွေ ထည့်သွင်းခြင်းမပြီး ရှိနေတဲ့အမှားတွေကို အရင်ပြင်တဲ့အလေ့လအထက ရေရှည်မှာပိုပြီးထိရောက် မြန်ဆန်တဲ့ Productivity ကို ရရှိစေမှာပဲဖြစ်ပါတယ်။

နံပါတ် (၆) အချက်ဖြစ်တဲ့ **Up-to-date Schedule** က နံပါတ် (၅) နဲ့ ဆက်စပ်နေပါတယ်။ Software Project တစ်ခုကို စတင်စဉ်မှာ ပြီးစီးမယ့်အချိန် Release Schedule ကြိုတင်ခန့်မှန်းထားရပါတယ်။ ပြဿနာက၊ Requirement ဆိုတာ အခြေအနေအမျိုးမျိုးကြောင့် ပြောင်းသွားတက်လို့ မူလ Release Schedule က နောင်မှာ လက်တွေမကျတော့ တာမျိုး ဖြစ်လာတက်ပါတယ်။ အဲဒီလိုဖြစ်လာတဲ့အခါ မူလသတ်မှတ်ထားတဲ့အတိုင်း ပြီး လိုအောင် Bug ကို အရင်မပြင်ပဲ သတ်မှတ်ထားတဲ့ လုပ်ဆောင်ချက်တွေကို တစ်ခုပြီးတစ်ခုဆက် ထည့်သွားဖို့ အားထုတ်မိတက်ကြပါတယ်။ အဲဒီအခါမှာ Bug တွေကို နောက်မှ ပြင်ရလို့ ကြာသင့်တာထက် ပိုကြာသွားတက် သလို့ Bug တွေကုန်အောင် အရှင်းမိပဲ Software ကို Release လုပ်မိတက်တာမျိုးလည်း ဖြစ်တက်ပါတယ်။ ဒါကြောင့် လက်တွေကျတဲ့ Up-to-date Schedule ရှိနေမှု Software တစ်ခုရဲ့ အရည်အသွေးကလည်း ကောင်းမွန် မှာ ဖြစ်ပါတယ်။

နံပါတ် (၇) အချက်ဖြစ်တဲ့ **Spec** ကလည်း အရေးပါပါတယ်။ လိုရင်းကတော့ Software Project တစ်ခုကို လက်တန်း မလုပ်ပဲ ကြိုတင်ပြင်ဆင်ထားတဲ့ Specification တစ်ခုပေါ်မှာ အခြေခံပြီး ဆောင်ရွက်ဖို့ပါပဲ။ ဒါတော့ မှ လက်ရှိ Software ရဲ့ အခြေအနေ နဲ့ Specification ကို တိုက်ဆိုင်စစ်ဆေးပြီး ဖြစ်နိုင်ခြေရှိတဲ့ Release Schedule မှန်းဆတဲ့ လုပ်ငန်းတွေ့ Software ကို ဖြန့်ချီရာမှာ ကြိုတင်ပြင်ဆင်သင့်တဲ့ လုပ်ငန်းတွေနဲ့ အခြား အထွေထွေကိစ္စရပ်များကို စနစ်တကျ ဆောင်ရွက်နိုင်မှာဖြစ်ပါတယ်။ တိကျတဲ့ Spec တစ်ခုရှိရှိဖို့ခက်တဲ့ အကြောင်းကို အခန်း (၁) မှာ ဖော်ပြုခဲ့ပြီး ဖြစ်ပါတယ်။ ဒီလိုခက် ခဲလိုဖို့ပြီး Spec လုံးဝမရှိလို့တော့ မရပါဘူး။

နံပါတ် (၈) အချက်ဖြစ်တဲ့ **Working Condition** နဲ့ပက်သက်လိုတော့ နည်းနည်း အငြင်းပွားစရာရှိပါတယ်။ တစ်ချို့ ကလည်း Developer တွေအတွက် ဆိတ်ဖြမ်ပြီး အနောက်အယုက်ကင်းကင်း အလုပ်လုပ်နိုင်တဲ့ Private Office တွေ ဖန်တီးပေးထားမှ ပိုပြီးအလုပ်တွင်တယ်လို့ ဆိုကြပါတယ်။ တစ်ချို့ကလည်း အတူတစ်ကွ ပူးပေါင်း ဆောင်ရွက်နိုင်အောင် Developer တွေကို တစ်နေရာတည်းမှာ စုစုပေါင်းထားမှ ပိုပြီးအလုပ်တွင်တယ်လို့ ဆိုကြပါတယ်။ လိုရင်းကတော့၊ အရည်အသွေးမြင့် အောင်ဆုံးမြင့်မြတ်ရင် Developer တွေအတွက် မလိုလား အပ်တဲ့ အနောက်အယုက်တွေကင်းပြီး၊ ပူးပေါင်းဆောင်ရွက်မှုကိုအားပေးတဲ့ Working Condition တစ်ခု ဖန်တီးထားပေးနိုင်ဖို့ လိုပါတယ်။

နံပါတ် (၉) အချက်ဖြစ်တဲ့ **Best Tools** ဆိုတာလည်း အလုပ်တွင်ဖော်အတွက်ပါပဲ။ လိုင်စင်ဝယ်သုံးရတဲ့ IDE ကောင်း ကောင်းတစ်ခုကို သုံးလိုက်လို့ အလုပ်ပိုတွင်မယ်ဆိုရင် ကုန်ကျစရိတ်ကို နှမာ့မနေပဲ၊ ဝယ်သုံးရပါမယ်။ မြန်မြန်းမြင့် အင်တာနက်ရှိမှ အလုပ်တွင်မယ်ဆိုရင် မြန်မြန်းမြင့် အင်တာနက် တပ်ဆင်ရပါမယ်။ Commercial Database System ဝယ်သုံးလိုက်မှ စွမ်းဆောင်ရည်ပိုကောင်းသွားမယ်ဆိုရင် ဝယ်သုံးရပါမယ်။ ကုန်ကျစရိတ်ကို နှမာ့နေတာနဲ့ ထိရောက်သင့် သလောက်မထိရောက်လို့ အချိန်တွေကြာပြီး စရိတ်တွေ အတက်ခံမယ့်အစား၊ လိုအပ်တဲ့ Tool ကို ဝယ်ယူအသုံးပြုလိုက်တာ က ရေရှည်အတွက် ပိုအကျိုးရှိတယ်ဆိုတဲ့ အမိပါယ်ပဲဖြစ်ပါတယ်။

နံပါတ် (၁၀) အချက်ဖြစ်တဲ့ **Tester** ဆိတာလည်း အရေးပါတဲ့ အချက်တစ်ချက်ပါတယ်။ **အခန်း (၂)** မှာ Test Method အမျိုးမျိုး အကြောင်းကိုဖော်ပြခဲ့ပါတယ်။ Tester တွေရဲတာဝန်ကတော့ System Test လုပ်ငန်းကို ဆောင်ရွက်ဖို့ပဲဖြစ် ပါတယ်။ Software ဟာ Specification က သတ်မှတ်ထားတဲ့ သတ်မှတ်ချက်များ အတိုင်း မှန် မှန် ကန်ကန် အလုပ်လုပ်ရဲ လား၊ သတ်မှတ်ထားတဲ့ လုပ်ဆောင်ချက်တွေ အပြည့်အစုံ ပါရဲလား၊ မရှိသင့်တဲ့ လုပ်ဆောင်ချက်တွေနဲ့ အမှားတွေရှိနေသလား၊ စသဖြင့် ထောင့်စီအောင်စီစစ်ပေးမယ့်သူများ ဖြစ်ပါတယ်။ Tester ဆိတာ Software ရဲ Quality ကို စီစစ်အာမခံ ပေးရမယ့်သူများ ဖြစ်ပါတယ်။ ဒါကြောင့် သူတို့ကို Quality Assurance Engineer များလိုလည်း ခေါ်နိုင်ပါတယ်။ Joel Spolsky ရဲအဆိုအရဆိုရင် Team တစ်ခုမှာ Developer နှစ်ယောက်ကို Tester တစ်ယောက်နှင့်နဲ့ တွဲဖက်ထားရှိသင့် တယ်လို့ ဆိတားပါတယ်။

နံပါတ် (၁၁) အချက်ဖြစ်တဲ့ **Interview** ကတော့ Team မှာ ပါဝင်တဲ့ Developer ဟာ ရှိသင့်တဲ့ အရည်အချင်း တွေရှိနေတာ သေခြားစေဖို့ ဖြစ်ပါတယ်။ Developer တစ်ယောက်ဟာ အမှန်တစ်ကယ် အရည်အချင်းရှိပါတယ်။ အမှန်တစ်ကယ် သူမှာရှိထားတဲ့ ဘွဲ့လက်မှတ်တွေ၊ အောင်လက်မှတ်တွေကို ကြည့်ပြီးမဆုံးဖြတ်သင့်သလို့၊ အမေးအဖြေ သက်သက် အင်တာဗျားနဲ့လည်း မဆုံးဖြတ်သင့်ပါဘူး။ လက်တွေမှာ၊ လေးစားလောက်ဖွယ် Degree တွေကိုင် ထားပေမယ့် အခြေခံ Programming Logic တောင် မပိုင်နိုင်တဲ့သူတွေအများကြီးရှိပါတယ်။ ဒါကြောင့် Team မှာပါဝင်တဲ့ Developer များကို ခန့်ထားဖို့ စီစစ်ရာမှာ လက်တွေ၊ Code ရေးခိုင်းပြီး အမှန်တစ်ကယ် အရည်အချင်း ရှိသူဟုတ်မဟုတ် စီစစ်ဖို့လိုကယ်ဆိုတဲ့ အချက်ကို ထည့်သွင်းထားခြင်းဖြစ်ပါတယ်။

နံပါတ် (၁၂) အချက်ဖြစ်တဲ့ **Hallway Usability Testing** ဆိတာကတော့၊ လမ်းသွားလမ်းလာ Hallway မှာတွေ တဲ့သူကို ဆွဲခေါ်ပြီးကိုယ်ဖန်တီးထားတဲ့ Software ကို စမ်းသုံးခိုင်းကြည့် သင့်တယ်ဆိုတဲ့သဘောပါ။ ကိုယ့် Software အကြောင်း လုံးဝမသိတဲ့ Hallway ထဲမှာတွေတဲ့သူတောင် ချက်ချင်းသုံးနိုင်တယ်ဆိုရင် ကိုယ့် Software က တော်တော်ကို Usable ဖြစ်တယ်လို့ မှတ်ယူနိုင်ပါတယ်။ မသုံးနိုင်ဘူး ဆိုရင်လည်း သူအတွက်ဘာ တွေက အသုံးပြုရခက်နေတာလဲဆိုတဲ့အချက် အပါအဝင် သူဆီကပြန်ရလာတဲ့ Feedback က Software ရဲ Usability ကို မြှင့်တင့်ဖို့ အထောက်အကူ ဖြစ်စေမှာဖြစ်ပါတယ်။

ဒီအချက်တွေပါဝင်တဲ့ Article ဟာ (၂၀၀၀) ခုနှစ်ကတည်းက ရေးသားထားတဲ့ Article တစ်ခုဖြစ်ပေမယ့် ကနေ အချိန်ထိ ခေတ်ရှိနေဆဲဖြစ်ပါတယ်။ အောက်ပါလိပ်စာမှာ မူရင်းအတိုင်းလေ့လာနိုင်ပါတယ်။

<http://www.joelonsoftware.com/articles/fog0000000043.html>

## 6.1 – Bug Database

Joel Spolsky ရဲအချက် (၁၂) ချက်ထဲမှာ နံပါတ် (၄) အချက်က Bug Database ဖြစ်ပါတယ်။ Software တိုင်းမှာ Bug တွေရှိတက်ကြောင်းကို ရှေ့ပိုင်းတွေမှာလည်း ဆွေးနွေးခဲ့ပြီးဖြစ်ပါတယ်။ တစ်ချို့ Bug တွေကို ရေးသားနေ စဉ် ကတည်းက Developer ကိုယ်တိုင် သတိပြုတွေရှိနိုင်ပေမယ့်၊ တစ်ချို့ကိုတော့ စမ်းသပ်တဲ့အဆင့် ရောက်တော့မှ Tester တွေက တွေ့ရှိရတက်ပါတယ်။ ဒါတင်မက တစ်ချို့ Bug တွေဆုံးရင် အများကင်းပြီလို့ Developer တွေရော၊ Tester တွေကပါ လက်ခံယူဆပြီး အများအသုံးပြုနိုင်အောင် Release လုပ်လိုက်ပြီးနောက်မှ တွေ့ရတက်ပါတယ်။ Bug တွေ မရှိချင်လို့တော့မရပါဘူး။ Software မှန်ရင်တော့ Bug တွေ ရှိမှာပဲဖြစ်ပါတယ်။ အဲဒီ Bug တွေကို စနစ်တစ်ကျိုစီမံနိုင်ဖို့ သာ လိုပါတယ်။

ပထမဆုံးအနေနဲ့ အရေးကြီးတာကတော့ ရှိနေတဲ့ Bug တွေကို သိရှိမှတ်မိန့်ပဲဖြစ်ပါတယ်။ Developer အများစုံက ကိုယ်တွေ့ရှိထားတဲ့ Bug တွေကို အမြဲမှတ်မိမယ်လို့ ယူဆထားတက်ကြပါတယ်။ လက်တွေ့မှာ မမှတ်မိနိုင်ပါဘူး။ အထူးသဖြင့် စနစ်တစ်ခုလုံး အလုပ်မလုပ်တော့ပဲ ရပ်တန်သွားလောက်အောင် မဆိုးတဲ့ Minor Bug တွေကို မပြင်ပဲထားတာ အချိန်ကြာ လာတာနဲ့အမျှ မေ့မေ့လျှော့လျှော့ ဖြစ်သွားတက်ပါတယ်။ နောင်တစ်ချိန်ကျတော့မှာ ပြင်းလောက်ခဲတဲ့ Bug တစ်ခုကို မေ့ပြီးမပြင်လိုက်မိပဲ Release လုပ်လိုက်မိလို့ နောင်တရရတက်ပါတယ်။ ဒါကြောင့် တွေ့ရှိသမျှ Bug တိုင်းကို မှတ်တမ်း တင်ထားဖို့လိုပါတယ်။

ပြီးတော့ Software တစ်ခုကို တစ်ယောက်ထက်ပိုတဲ့ Developer တွေက ပူးပေါင်းရေးသားကြတဲ့အခါ၊ တစ်ခြီးတွေ့ရှိထားတဲ့ Bug ကို ကျန် Developer များကလည်း သိရှိနေဖို့ လိုအပ်ပါတယ်။ ဒါကြောင့် But မှတ်တမ်းကို Developer တိုင်း အချိန်မရွေး ကြည့်ရှုနိုင်ဖို့အတွက် Centralize Database တစ်ခုနဲ့ထားရှိသင့်ပါတယ်။ ဒါအပြင် Developer မဟုတ်တဲ့ Manager, Tester နဲ့ User တွေ တွေ့ရှိလာတဲ့ Bug တွေလည်း ရှိနိုင်ပါတယ်။ အဲဒီလို တွေ့ရှိလာတဲ့ Bug တွေကို Developer တွေသိရှိနိုင်ဖို့ Bug Database ထဲကို ထည့်သွင်းပေးနိုင်တဲ့ Bug Report လုပ်ဆောင်ချက်မျိုးတွေ Bug Database မှာ ရှိထားသင့်ပါတယ်။

Bug Database တစ်ခုမှာ Bug တွေကို မှတ်တမ်းတင်ဖို့အတွက် အခြေခံ Field (၅) ခဲ့ အနည်းဆုံး ပါဝင်သင့်ပါတယ်။

- Steps to reproduce** – မှတ်တမ်းတင်ထားတဲ့ Bug ကို ပြန်လည်လေ့လာလိုရင် ဆောင်ရွက်ရမယ့် အဆင့်များ
- Expected behavior** – ရရှိမယ်လို့ မျှော်မှန်းထားတဲ့ ရလဒ် (ရလဒ်အမှန်)
- Buggy behavior** – အမှန်တစ်ကယ်ရရှိလာတဲ့ ရလဒ် (ရလဒ်အမှား)
- Assigned person** – တာဝန်ယူဖြေရှင်းရမယ့်သူ
- Status** – တာဝန်ယူဖြေရှင်းပြီးစီးခြင်း ရှိ၊ မရှိ အခြေအနေ

- တို့ပဲဖြစ်ပါတယ်။

Bug တစ်ခုကို မှတ်တမ်းတင်လိုက်တာနဲ့ အဲဒီ Bug ကို ပြည်လည်လေ့လာလိုတဲ့အခါ ဆောင်ရွက်ရမယ့် **Reproducing Steps** ကို တစ်ပါတည်းတွဲဖော် မှတ်တမ်းတင်ဖို့လိုပါတယ်။ ဘယ်လိုအချိန်ဘယ်လိုအခြေအနေ မှာ Bug ကို တွေ့ရတာလဲ ဆိုတဲ့အဆင့်တွေကို တွဲဖော်မှတ်တမ်းတင်ထားမှ တာဝန်ယူဖြေရှင်းရမယ့်သူက Bug ကို ပြည်လည်လေ့လာပြီး ဘာကြောင့် ဖြစ်နိုင်သလဲ ဆိုတဲ့ပြဿနာရင်းမြစ်နဲ့ အဖြေကို ရှာဖွေနိုင်မှာဖြစ်ပါတယ်။

တစ်ခါတစ်ရုက္ခာတော့ တစ်ကယ်မှားနေတာမဟုတ်ပဲ မှားတယ်ထင်ရတာမျိုးလည်း ရှိတက်ပါတယ်။ ဒါကြောင့် Bug ကို မှတ်တမ်းတင်တဲ့သူက၊ သူမျှော်လင့်ထားတဲ့ရလဒ် **Expected behavior** နဲ့ သူရရှိတဲ့ **Buggy behavior** တို့ကို တွဲဖော်မှတ်တမ်းတင် ပေးဖို့လိုပါတယ်။ ဥပမာ – Expected အနေနဲ့ "Mobile Phone Category တွင် Mobile Phone စာရင်းကိုသာ ဖော်ပြသင့်သည်" လို့ မှတ်တမ်းတင်ပြီး Bug အနေနဲ့ "Mobile Phone Category တွင် Laptop ကွန်ပူးတာများကိုပါ ဖော်ပြနေသည်" လို့ မှတ်တမ်းတင်နိုင်ပါတယ်။ ဒီတော့မှ တာဝန်ယူဖြေရှင်းရမယ့်သူက တစ်ကယ်ပဲမှားပြီး Laptop ကွန်ပူးတာတွေ Mobile Phone စာရင်းထဲမှာဝင်နေတာလား၊ တမင်

ရည်ရွယ်ချက်ရှိရှိ Related Item အနေနဲ့ Mobile Phone စာရင်းနဲ့အတူ Laptop ကွန်ပျူတာတွေကို တဲ့ပြထားတာလား ဆုံးဖြတ်ပြီး၊ လိုအပ်သလို ဆက်လက်ဆောင်ရွက်နိုင်မှာဖြစ်ပါတယ်။ ပေးထားတဲ့နမူနာအရဆိုရင် Bug ကို မှတ်တမ်းတင်သွားမှာ မှားနေတယ်ထင်လို့ မှတ်တမ်းတင်ပေမယ့် လက်တွေမှာ တစ်ကယ့်အမှားမဟုတ်ပဲ တမင်ရည်ရွယ်ထည့်သွင်းထားတဲ့ Feature တစ်ခု ဖြစ်နိုင် ပါတယ်။

Bug တစ်ခုမှတ်တမ်းတင်တာနဲ့ အဲဒီ Bug ကို တာဝန်ယူဖြေရှင်းရမယ့်သူ **Assigned Person** ကို တစ်ခါတည်း တွဲဖက် ထည့်သွင်းသင့်ပါတယ်။ အကယ်၍ Bug ကို မှတ်တမ်းတင်တဲ့သူက ဘယ်သူတာဝန်ယူရမလဲ ဆုံးဖြတ် နိုင်တဲ့သူ မဟုတ်ရင် လည်း၊ ဆုံးဖြတ်နိုင်သူဖြစ်တဲ့ Team Leader ကို Assigned Person အဖြစ် တွဲဖက် ထည့်သွင်းသင့်ပါတယ်။ နောက်တော့ မှ Team Leader က အမှန်တစ်ကယ် တာဝန်ယူရမယ့် Developer ကို လွှာပြောင်းပြီး တာဝန်ပေးသွားပါလိမ့်မယ်။ ဘယ်သူကိုမှ Assigned လုပ်ထားခြင်းမရှိတဲ့ Bug တွေတော့ စာရင်းထဲမှာ မရှိသင့်ပါဘူး။ အနဲ့ဆုံး Team Leader ကိုတော့ Assigned လုပ်ထားပေးသင့်ပါတယ်။

Bug တစ်ခုကို မှတ်တမ်းတင်လိုက်တာနဲ့၊ အဲဒီ Bug ကို စပြင်နေပြုလား၊ မပြင်သေးဘူးလား၊ ပြင်ပြီးသွားပြုလား စတဲ့ အခြေအနေ **Status** အမျိုးမျိုးရှိနိုင်ပါတယ်။ အဲဒီ အခြေအနေကို Bug နဲ့အတူ အမြတ်ဖတ် မှတ်သားထားဖို့ လိုပါတယ်။ ဒီတော့မှ ပြင်ပြီးကြောင်း မှတ်တမ်းတင်ထားတဲ့ Bug ကျေဟာ တစ်ကယ်လိုချင်တဲ့ Behavior ရရှိအောင် ပြင်ဆင်ပြီးပြီ လား ပြန်လည်စမ်းစစ်နိုင်မှာဖြစ်ပါတယ်။ တစ်ချို့ တစ်ကယ်အမှားမဟုတ်လို့ မပြင်ဖို့ ဆုံးဖြတ်တယ်ဆိုရင်လည်း မူလထည့်သွားသူက Status ကိုကြည့်ပြီး ဒါဟာ ရည်ရွယ်ချက်ရှိရှိနဲ့ တမင် ထည့်သွေးထားတဲ့ Feature မှာ မပြုပိုင်ခဲ့ပြီ ကြောင်းကို သိရှိနိုင်မှာဖြစ်ပါတယ်။

ဒီအခြေခံအချက် (၅) ချက်ကတော့ အခြေခံအနေနဲ့ Bug Database တစ်ခုမှာ မဖြစ်မနေပါဝင်သင့်တဲ့ အချက် တွေဖြစ် ပါတယ်။ ပိုပြီးပြည့်စုံလိုတယ်ဆိုရင် Bug မှတ်တမ်းနဲ့ အတူ Screenshot တွေ တွဲဖက်မှတ်တမ်းတင်နိုင် ပါတယ်။ အကယ်၍ အလားတူ Bug ကို ယခင်ကရှိခဲ့ဖူးတယ်ဆိုရင်၊ ယခင် Bug ကိုညွှန်းထားတဲ့ အညွှန်းကို တွဲဖက်မှတ်တမ်းတင်နိုင်ပါ တယ်။ Bug တွေရရှိမှာ အသုံးပြုနေတဲ့ OS Version, Browser Version နဲ့ အခြား ဆက်စပ် Software တွေနဲ့ Version တွေကို တွဲဖက်မှတ်တမ်းတင်မယ်ဆိုရင် ပြင်ဆင်ရမယ့် Developer အတွက် အထောက်အကူ ဖြစ်နိုင်ပါတယ်။ Bug တွေကို အမျိုးအစားအလိုက် Category ခွဲပြီးတော့လည်း မှတ်တမ်းတင်ထားနိုင်ပါတယ်။ Bug ရဲ့ အရေးပါမှာ အဆင့် Priority ကိုလည်း တွဲဖက်မှတ်တမ်းတင်နိုင်ပါတယ်။

Bug Database အနေနဲ့ နှစ်သက်ရာ Tool ကို သုံးနိုင်ပါတယ်။ Project ကသေးပြီး တစ်ဦးတည်းရေးသားခြင်း ဖြစ်လို ရိုးရိုး Text File တစ်ခုနဲ့ မှတ်တမ်းတင်ထားမယ်ဆိုလည်း ထားနိုင်ပါတယ်။ ပိုပြီးစနစ်ကျအောင် Spreadsheet တစ်ခုနဲ့ Column တွေခွဲပြီး အသေအချာ Color-Code လုပ်ထားမယ်ဆို လည်းထားနိုင်ပါတယ်။ Bugzilla, Trac, RedMine, MantisBT စတဲ့ အသင့်သုံးလိုဂရတဲ့ Bug Tracking System တွေ သုံးမယ်ဆိုလည်း သုံးနိုင်ပါတယ်။ Bug Database ထားရှုခြင်းရဲ့ အရေးပါမှုကို သိရှိဖို့ကသာ လိုဂင်းဖြစ်ပြီး လက်တွေသုံးတဲ့ အခါ နှစ်သက်ရာ Tool ကိုသုံးနိုင်ပါတယ်။

## Bug Reporting Best Practices

Bug Database တစ်ခုနဲ့ Bug တွေမှတ်တမ်းတင်ရာမှာ အခြေခံစည်းမျဉ်းလေးတစ်ချို့ သတ်မှတ်ထားဖို့လိုပါတယ်။ ဥပမာ - "Reproducing Steps တွေမဖြစ်မနေထည့်သွင်းရမယ်" ပြီးတော့ "Bug မှတ်တမ်း အသစ်တစ်ခု မထည့်သွင်း မီ အလားတူ Bug ကို ယခင်ကထည့်သွင်းခဲ့ဖူးခြင်းမျိုး သေချာအောင်အရင်ကြည့်ရမယ်" စသဖြင့် အခြေခံ စည်းမျဉ်း လေးတစ်ချို့ သတ်မှတ်ထားရပါတယ်။ ဒီလိုသတ်မှတ်မထားရင် Bug Database ထဲမှာ အမှန် တစ်ကယ်အသုံးမဝင်တဲ့ Bug Report တွေနဲ့ ပြည့်နှက်ပြီး စီမံရ ခက်ခဲသွားတက်ပါတယ်။ အသေးစိတ်စည်းမျဉ်း တွေ ဘယ်လိုသတ်မှတ်မလဲဆိုတာ အသုံးပြုမယ့် Team ရဲ့ စွဲစည်းပုံနဲ့ Project အမျိုးအစားပေါ်မှာ မူတည်ပါတယ်။ ဒီနေရာမှာတော့ လိုက်နာသင့်တဲ့ အခြေခံ စည်းမျဉ်းတစ်ချို့ကို ရွေးထုတ်ဖော်ပြလိုက်ပါတယ်။

၁။ Bug တစ်ခုကိုဖြေရှင်းပြီးတဲ့အခါ ဖြေရှင်းပြီးကြောင်း မှတ်တမ်းတင်ရပါတယ်။ အဲဒါလိမှုမှတ်တမ်းတင်ရမှာ အမှန်တစ်ကယ် ဖြေရှင်းလိုက်လို ပြေလည်သွားတာဆိုရင် Status ကို **Fixed** လို တွဲဖက်မှတ်တမ်းတင်သင့်ပါတယ်။

JII အမှန်တစ်ကယ် Software မှာ ရှိနေတဲ့ပြဿနာကြောင့်မဟုတ်ပဲ အခြားပြဿနာကြောင့်ဖြစ်နေတဲ့အတွက်ဖြေရှင်းလို မရနိုင်ရင် (ဥပမာ - Hardware ကြောင့်ဖြစ်တဲ့ပြဿနာ၊ အခြား Software တစ်ခုကြောင့်ဖြစ်တဲ့ပြဿနာ) Status ကို **Won't Fix** လို တဲ့ဖက်မှတ်တမ်းတင်ထားသင့်ပါတယ်။

၃။ Bug တစ်ခုပါလို့ မှတ်တမ်းတင်ထားပေမယ့် Reproduction Steps တွေမပါရင် (သို့မဟုတ်) စမ်းကြည့်တော့ အဲဒီ ပြဿနာတစ်ကယ်ရှိမနေရင် Status ကို **Not Repro** လို့ တွဲဖက်မှတ်တမ်းတင်သင့်ပါတယ်။ သတိပြုရမှာ က လုံးဝ Reproduce လုပ်လို့မရမှ ဒီလို့သတ်မှတ်ရမှာပါ။ တစ်ခါတစ်ရုံ Reproduce လုပ်လို့ရပြီး၊ တစ်ခါတစ်ရုံ မရဘူးဆိတဲ့ Bug မျိုးကလည်း ရှိတက်ပါသေးတယ်။ ရှာဖွေပြင်ဆင်ရ အခက်ခဲဆုံး Bug အမျိုးအစားဖြစ်ပါတယ်။

၄။ အလားတူ Bug ကိုပဲ တစ်ကြိမ်ထက်ပိုမြို့း ထည့်သွင်းထားမြှင်း ဖြစ်နေရင်တော့ ထပ်နေတဲ့ Bug ရဲ့ Status ကို **Duplicate** လို့ တုံဖက်မှတ်တမ်းတင်သင့်ပါတယ်။

၅။ Bug မဟုတ်ပဲ၊ Feature ကို Bug လိုပုံဆဲပြီး မှားယွင်းထည့်သွင်းထားမိတဲ့ Bug တွေရဲ့ Status ကိုတော့ By Design လို့ တဲ့ဖက်မှတ်တမ်းတင်သင်ပါတယ်။

၆။ Bug တစ်ခုအတွက်ဖြေရှင်းပြီးကြောင်း Status ကိုလက်ခံရရှိပြီဆိုရင် အမှန်တစ်ကယ်ဖြေရှင်းပြီးခြင်းဟုတ်မဟုတ် စိစစ်ပြီး၊ အမှန်တစ်ကယ် ပြေလည်သွားတာဆိုရင် အပြီးပါတ်သိမ်းလိုက်ရပါတယ်။ သတိပြုရမှာ၊ အဲဒီလို ပိတ်သိမ်းခွင့်ကို မူလ Bug မှတ်တမ်း ထည့်သွင်းခဲ့သူကိုပဲ ပိတ်သိမ်းခွင့်ပြုသင့်ပါတယ်။ မူလမှတ်တမ်းတင်ခဲ့သူမှာသာ သူမှတ်တမ်းတင်ခဲ့တဲ့ Bug အမှန်တစ်ကယ် ပြေလည်သွားခြင်းရှိမရှိကို ဆုံးဖြတ်နိုင်မှာဖြစ်ပါတယ်။

Version တွေကို စနစ်တစ်ကျ စီမံပြီး Bug Report တွေ မှတ်တမ်းတင်ရာမှာလည်း ဘယ် Version အတွက်လည်း ဆိုတာကို တဲ့ ဖက်မှတ်တမ်းတင်ထား သင့်ပါတယ်။

၈။ Bug Database ရှိနေရက်နဲ့ ပြဿနာတစ်ခုတွေတဲ့ အခါ Developer ထံ Email သို့မဟုတ် အခြားနည်းလမ်းနဲ့ အဲဒီ Bug အကြောင်းပေးပို့လာရင် လက်မံခံသင့်ပါဘူး။ Developer တွေအနေနဲ့ Bug Database ထဲက Bug က လွှဲရင် အခြား နည်းလမ်းနဲ့ ပေးပို့လာတဲ့ Bug တွေကို တာဝန်ယူဆောင်ရွက်မှာ မဟုတ်ဘူးလို့ ပြတ်ပြတ်သားသား သတ်မှတ်ထားသင့်ပါတယ်။ ဒီတော့မှာ Team မှာပါဝင်သူအားလုံး Bug Database ကို အသုံးပြုဖို့ တွေ့န်းအားတစ်ခု ဖြစ်စေသလို့ Database ထဲမရောက်လို့ မှတ်တမ်းပောက်သွားပြီး မပြင်မိပဲ ကျိုးသွားတယ်ဆိုတဲ့ Bug တွေလည်း မရှိတော့မှာဖြစ်ပါတယ်။

၉။ ကိုယ်တိုင် Bug တစ်ခုတွေလာတဲ့ အခါမှာလည်း အဲဒီ Bug အကြောင်း သက်ဆိုင်ရာ Developer ကို တိုက်ရှိက် မပြောပဲ၊ Bug Database ထဲကိုသာ မှတ်တမ်းအဖြစ် ထည့်သွင်းလိုက်သင့်ပါတယ်။ ဒီတော့မှာ အခြား Developer တွေအနေနဲ့ ရှိနေတဲ့ Bug တွေအကြောင်းသိချင်ရင် Bug Database မှာ အမြဲကြည့်ရမယ်ဆိုတဲ့ တွေ့န်းအားကို ဖြစ်စေမှာဖြစ်ပါတယ်။

၁၀။ Bug တစ်ခုကို တာဝန်ယူဖြေရှင်းဖို့ Developer တစ်ယောက်ကို Assign လုပ်လိုက်တိုင်းမှာ၊ Assign လုပ်ခံရသူထံ Email သို့မဟုတ် နည်းလမ်းတစ်မျိုးမျိုးနဲ့ အလိုအလျောက် Notify လုပ်ဖို့ စီစဉ်ထားသင့်ပါတယ်။ ဒီနည်းနဲ့ မူလက Bug Database ကို ထိရောက်အောင် မသုံးဖြစ်တဲ့ Developer တွေရှိရင် အသုံးပြုဖို့ တွေ့န်းအားဖြစ်စေမှာ ဖြစ်ပါတယ်။ အသင့် သုံး Bug Tracking System တွေမှာတော့ ဒီလိုလုပ်ဆောင်ချက်မျိုး အသင့်ပါပြီး ဖြစ်ပါတယ်။

၁၁။ Bug Database ကို Bug Report အတွက်သာမက Feature Request အတွက်ပါအသုံးပြုနိုင်ပါတယ်။ Project Manager တစ်ယောက်အနေနဲ့ Software မှာ လုပ်ဆောင်ချက်အသစ် ထည့်သွင်းစေလိုတဲ့ အခါ (သို့မဟုတ်) ရှိနေတဲ့ လုပ်ဆောင်ချက်ကို ပြင်ဆင်စေလိုတဲ့ အခါ၊ Bug Database ထဲကိုထည့်သွင်းပြီး တာဝန်ပေးလိုတဲ့ Developer ကို Assign လုပ်နိုင်ပါတယ်။ ဒီနည်းနဲ့ Bug Database ဆိုတာ Bug မှတ်တမ်းသက်သက်မဟုတ်တော့ပဲ Developer တွေ နေစဉ် လက်ခွဲထားအလုပ်လုပ်ရတဲ့ Job Assignment List တစ်ခု ဖြစ်သွားမှာဖြစ်ပါတယ်။

## 6.2 – Bug Database vs. Issue Tracking System

Bug Tracking System (Bug Database) နဲ့ Issue Tracking System ဆိုတာ သဘောတရားအရ အတူတူပါပဲ။ အခြေခံအား ပြင်းပြောရရင် Bug Database က Bug တွေ မှတ်တမ်းတင်ဖို့ အတွက်သုံးပြီး Issue Tracking System ကိုတော့ Bug အပါအဝင် Software မှာရှိနေတဲ့ အထွေထွေပြဿနာ Issue တွေကို မှတ်တမ်းတင်ဖို့ သုံးခြင်း ဖြစ်ပါတယ်။ Issue Tracking System တွေက Bug Database ထက် လုပ်ဆောင်ချက်တွေ ပိုမိုစုံလင်ပြီး ပိုမိုကျယ်ပြန်တက်ပါတယ်။

Bug Database ကို Software Team မှာပါဝင်တဲ့ Project Manager, Developer နဲ့ Tester တွေက အမိကအသုံးပြု ပေမယ့် Issue Tracking System ကိုတော့ အသုံးပြုသူ User တွေ Management Level အရာရှိတွေ

Marketing Staff တွေနဲ့ လုပ်ငန်းတစ်ခုလုံးမှာ ပါဝင်သူတွေက အသုံးပြုနိုင်ဖို့ ရည်ရွယ်ဖန်တီးထားတက်ကြပါတယ်။ ဒီလိုဂုဏ်ရည်ချက် ကျယ်ပြန့်သွားပေမယ့် အလုပ်လုပ်ခုံသောတရားကတော့ Bug Database နဲ့အတူတူပါပဲ။ ရှိနေတဲ့ ပြဿနာ Issue တစ်ခုကို စနစ်မှာမှတ်တမ်းတင်ပြီး၊ အဲဒီပြဿနာကို သက်ဆိုင်ရာ တာဝန်ရှိသူတစ်ဦးကို Assign လုပ်ခြင်း၊ တာဝန်ရှိသူက လိုသလိုဆောင် ရွက်ပြီးတဲ့အခါ ဖြေရှင်းပြီးကြောင်း မှတ်တမ်းတင်ခြင်း၊ အမှန်တစ်ကယ်ဖြေရှင်းပြီးကြောင်း သေချာတဲ့အခါ ပိတ်သိမ်းခြင်း လုပ်ငန်းတွေကို ဆောင်ရွက်နိုင်တဲ့ စနစ်ပဲဖြစ်လို့ Issue Tracking System နဲ့ Bug Database ဆိုတာ အခြေခံအားဖြင့် အတူတူပဲဖြစ်တယ်လို့ ဆိုနိုင်ခြင်း ဖြစ်ပါတယ်။

Issue Tracking System ကို Ticket Management System (သို့မဟုတ်) Customer Relationship Management (CRM) System လိုလည်း ခေါ်တက်ကြပါသေးတယ်။ Software အပါအဝင် ဝန်ဆောင်မှုလုပ်ငန်းတွေမှာ အသုံးပြုသူ User နဲ့ Customer ဘက်က အကြောင်းကြားလတဲ့ သူတို့ရဲ့အက်အခဲပြဿနာ Issue တွေကို လက်ခံထည့်သွင်းခြင်း၊ သက်ဆိုင်ရာ Technician ကို ဖြေရှင်းဖို့တာဝန်ပေးခြင်း၊ ဖြေရှင်းဆောင်ရွက်ပြီးစီးတဲ့ အခါ ပိတ်သိမ်းခြင်း စသဖြင့် အခေါ် အတော် ကွဲသွားပေမယ့် အခြေခံအလုပ်လုပ်ပုံ Workflow ဆင်တူတဲ့ စနစ်တွေဖြစ်ပါတယ်။

Software Project တစ်ခုကို စီမံတဲ့နေရာမှာ Bug တွေသာမက Feature တွေနဲ့ Customer Support Request တွေ ကိုပါ စီမံဖို့လိုအပ်တက်လို့ ရှိုးရှိုး Bug Database တစ်ခုထက် ပိုမိုပြည့်စုံတဲ့ Issue Tracking System တွေ ကို အသုံးပြု ခြင်းအားဖြင့် ပိုမိုထိရောက်အကျိုးရှိစေနိုင်ပါတယ်။ တွေ့ရလေ့ရှိတဲ့ ပြဿနာလေးတစ်ခုကို နမူနာ အနေနဲ့ ဖော်ပြပေး လိုက်ပါတယ်။

**Project Manager** – "Search Result ကို Bookmark လုပ်ထားလိုရတဲ့ လုပ်ဆောင်ချက်က ဘာလို့ မပါသေးတာလဲ။ ပြီးခဲ့တဲ့ Meeting မှာ ဒီလုပ်ဆောင်ချက်ကို ဦးစားပေးထည့်ဖို့ ပြောထားပြီးသား မဟုတ်လား။"

**Team Leader** – "လုပ်ဆောင်ချက်အသစ်တစ်ခု ရှိလာတိုင်း သက်ဆိုင်ရာ Developer တွေကို ချက်ခြင်းတာဝန် သတ်မှတ်ပေးထားပြီးသားပါ။ Search Result Bookmark က ဘယ်သူ တာဝန်လဲ။"

**Developer** – "ကျွန်ုတ်ပေါ်ပါခင်ဗျာ။ လက်ရှိလုပ်လက်စ Newsletter လုပ်ဆောင်ချက်လေး မပြတ် သေးလို့ Search Result Bookmark ဘက်ကို မလှည့်နိုင်သေးတာပါ။"

**Project Manager** – "Newsletter က Minor Feature ပဲ။ သိပ်အရေးမကြီးဘူးလေး။ ပိုပြီး အမိုက ကျွန်ုတ်တဲ့ Search Result Bookmark ကို အရင်လုပ်ရမှာပေါ့။ Team Leader က ဘယ်ဟာ Minor, ဘယ်ဟာ Major သေချာရှင်းပြမထားဘူးလား။"

**Team Leader** – "Major, Minor ခွဲထားပြီးသား Feature List ကို Email နဲ့ Developer အကုန်လုံးကိုပို့ထားပြီးသားပဲ။"

**Developer** - "ကျွန်ုပ်တော် အဲဒီ Email မရပါလား။"

**Team Leader** - "ဘာလို့မရရမှုလဲ၊ အသေအချာပို့လိုက်တာပဲ။"

**Developer** - "မရပါဘူး။ သေချာအောင် နောက်တစ်ခေါက်ထပ်ကြည့်လိုက်မြို့မယ်။ သော်... ဖြစ်ရ မယ်၊ Email က Spam Folder ထဲ ရောက်နေတာကိုး။"

**Project Manager** - "မင်းတို့လွှဲကြတာ အရေးမကြီးဘူး။ ငါက Management ပိုင်းက လူကြီး တွေ ကို ဒီလုပ်ဆောင်ချက် Demo လုပ်မလို Meeting ချိန်းထားပြီးသား။ အခုတော့ ပြစ်ရာမရှိဘူး၊ ပြဿနာတက်ပြီ။ နောင်ကို သေချာအောင်လုပ်ကြကွာ။"

ဒါဟာ Software Project တွေမှာ တွေ့ရလေ့ရှိတဲ့ ပြဿနာတစ်ခုပဲဖြစ်ပါတယ်။ ဒီစာအုပ်မှာ အကြမ်ကြမ်ပြောနေတဲ့ အချက်တစ်ခုက်ရှိပါတယ်။ Specification ဆိတာ တိကျဖို့ခက်ခြား အမြဲပြောင်းလဲနေတက်တဲ့ သဘောရှိပါတယ်။ Specification ပြောင်းလဲတယ်ဆိတာ Software ရဲ့ အလုပ်လုပ်ပုံနဲ့ ပါဝင်ရမယ့် Feature တွေပြောင်းလဲသွားခြင်းပဲ ဖြစ်ပါတယ်။ ဒီလို အမြဲပြောင်းလဲနေတက်တဲ့ Feature တွေကို စီမံဖို့အတွက် Issue Tracking System တစ်ခုကို အသုံးပြုမှသာ ထိရောက်မှုရှိမှာဖြစ်ပါတယ်။

### 6.3 – Standard Issue Tracking System Features

Issuer Tracking System တွေမှာ အခြေခံအနေနဲ့ အောက်ပါလုပ်ဆောင်ချက်တွေ ပါဝင်လေ့ရှိပါတယ်။

**User and Role Management** – Issuer Tracking System တစ်ခုကို Developer တွေ Tester တွေသာမက Manager တွေ Marketing Staff တွေနဲ့ Customer တွေအပါအဝင် အဖွဲ့အစည်းတစ်ခုလုံးမှာ ရှိတဲ့သူအားလုံး အသုံးပြု နိုင်ပါတယ်။ အဲဒီလို အသုံးပြုရောမှာ အရေးကြီးတာကတော့ Role Management ဖြစ်ပါတယ်။ ဥပမာ – Lead Developer မှသာ Issue တွေကို Assign လုပ်လိုရမယ်၊ Customer တွေက Read Only Access ပဲရမယ်၊ QA Engineer မှသာ Issues တွေကို ပိတ်သိမ်းလိုရမယ် စသဖြင့် Role တွေ အမျိုးမျိုး ခွဲခြားစီမံနိုင်ဖို့လိုပါတယ်။ ဘယ် User ကို ဘယ်လို Role ပေးမလဲဆိုတာက သက်ဆိုင်ရာ အဖွဲ့အစည်းရဲ့ Policy ပေါ်မှတည်ပါတယ်။ ဒါ ကြောင့် Issuer Tracking System တစ်ခုဟာ အဲဒီလို User တွေနဲ့ Role တွေကို လုပ်လွပ်လပ်လပ် စီမံလိုရနိုင်တဲ့ လုပ်ဆောင်ချက်တွေ ပါဝင်ရပါတယ်။

**Project Management** - Issue Tracking System ကို Project တစ်ခုတည်းအတွက်သုံးတာဖြစ်နိုင်သလို Project နှစ်ခုသုံးခုအတွက် Issue Tracking System တစ်ခုတည်းသုံးတာလည်း ဖြစ်နိုင်ပါတယ်။ ဒါကြောင့် Project တွေ ခွဲခြား စီမံထိရေး လုပ်ဆောင်ချက်ဟာလည်း Issue Tracking System တစ်ခုမှာ မဖြစ်မနေ ပါဝင်ရ မယ့် လုပ်ဆောင်ချက်ပဲ ဖြစ်ပါတယ်။

Report Issue

Project: [eCommerce Portal](#)

Category: Bug

Severity:  Feature  Minor  Major  Crash

Priority:  Low  Normal  Important  Urgent

Assign To: John Doe

Issue Summary: Drop down menu not working

Detail: Drop down menu in home page not working sometime. Using Firefox 32 on Windows 7 x86.  
Note: The problem is only on home page and it's working fine on other pages.

Steps To Reproduce:

1. Click the menu (Drop down shown)
2. Click gallery view button from photo slider
3. Click the menu again (Drop down not shown)

Screenshot or Reference: [Browse...](#) Screenshot from 2015-03-25 12:08:50.png

Submit Issue

ပုံ (၆.၁) - Example Issue Report Form

**Issue Management** – ဒါကတော့ အမိကလုပ်ဆောင်ချက်ပါ။ Issue Tracking System တစ်ခုဟာ Bug Report နဲ့ Feature Request တွေအပါအဝင် Issue တွေကို Priority အလိုက် မှတ်တမ်းတင်လိုရတဲ့ လုပ်ဆောင်ချက် ပါဝင်ရပါမယ်။ သက်ဆိုင်ရာတာဝန်ရှိသူကို Assign လုပ်လိုရတဲ့ လုပ်ဆောင်ချက်တွေ ပါဝင်ရမယ်။ ပုံ (၆.၁) မှာ ပေးထားတဲ့ Issue Report Form နဲ့နာကို လေ့လာကြည့်ပါ။ အပေါ်နားမှာ လက်ရှိ အလုပ်လုပ်နေတဲ့ Project ကိုဖော်ပြထားပါတယ်။ အမို့ပါယ်က Issue တွေကို သက်ဆိုင်ရာ Project အလိုက် ခွဲခြားစီမံနိုင်တဲ့သဘောပဲဖြစ်ပါတယ်။

Form မှာပါဝင်တဲ့ Field တွေကိုလေ့လာကြည့်မယ်ဆိုရင် ပထမဦးဆုံး အနေနဲ့ Issue Category ကို ဈေးခိုင်းထားပါတယ်။ UI Bug, Text Typo, API Bug, Feature Request စသေဖြင့် ထည့်သွင်းလိုတဲ့ Issue အမျိုးအစားကို ဈေးချယ်ရခြင်းဖြစ်ပါတယ်။ ဒါကြောင့် အဲဒီ Issue Category တွေ စီမံလိုရတဲ့ **Category Management** လုပ်ဆောင်ချက်ဟာလည်း Issue Tracking စနစ်တစ်ခုမှာပါဝင်ရမယ့် အခြေခံလုပ်ဆောင်ချက် တစ်ခုပဲဖြစ်ပါတယ်။

ဆက်လက်ပြီး Issue နဲ့ Severity Level နဲ့ Priority Level တို့ကို ဈေးချယ်ရပါတယ်။ ဒီတော့မှာ နောင် Issue တွေ ကို ဖြေရှင်းတဲ့အခါ အရေးပါမှုအလိုက် ဦးစားပေးဈေးချယ်ဖြေရှင်းနိုင်မှာဖြစ်ပါတယ်။ တစ်လက်စတည်း ဒီ Issue ကို တာဝန် ယူဖြေရှင်းရမယ့်သူကိုလည်း တဲ့ဖက် Assign လုပ်ရပါတယ်။ ပြီးတော့မှ Issue ခေါင်းစဉ်၊ အကြောင်းအရာနဲ့ Reproduction Steps တို့ကို ဖြည့် သွင်းရပါတယ်။ နောက်ဆုံးအနေနဲ့ Screenshot သို့မဟုတ်

ရည်ရွှေးကိုးကားသင့်တဲ့ ဖိုင်တစ်ခုခုရှိရင် တွဲဖက်ပေးရပါတယ်။

ံ (၆.၂) ကိုလေ့လာကြည့်ရင်တော့ မှတ်တမ်းတင်ထားတဲ့ Issue စာရင်းကို တွေ့ရမှာဖြစ်ပါတယ်။ Issue ရဲ့ Severity နဲ့ Priority ပေါ်မှတည်ပြီး ဦးစားပေး Issue ကို ဦးစားပေးဖြစ်ကြောင်း မြင်သာအောင် ဖော်ပြထားပါတယ်။ Category အလိုက်သော်လည်ကောင်း၊ Severity Level အလိုက်သော်လည်ကောင်း၊ Status အလိုက်သော်လည်ကောင်း Sorting စီကြည့်မယ်ဆုံးရင် စီနိုင်ပါတယ်။ ပြီးတော့ သက်ဆုံးရာ Category တစ်ခုပဲကြည့်မယ် သို့မဟုတ် Status တစ်ခုးကိုပဲ ကြည့်မယ်စသဖြင့် Filter လုပ်နိုင်ပါသေးတယ်။ ဒါတွေဟာ Issue Management အနေနဲ့ ပါဝင်ရမယ့် အခြေခံလုပ် ဆောင်ချက်တွေပဲဖြစ်ပါတယ်။

Issues List						Project: <a href="#">eCommerce Portal</a>
View all issues   My Issues			Filter: All category ▾ All severity ▾ All priority ▾ All status ▾			
ID	Category	Summary	Severity	Priority	Status	
 <a href="#">000001</a>	Bug	Drop down menu not working	Minor	Normal	<input checked="" type="checkbox"/> Assigned	
 <a href="#">000002</a>	Text	Mispell in checkout form	Major	Important	<input checked="" type="checkbox"/> WIP	
 <a href="#">000003</a>	Security	Potential XSS vulnerability in order form	Crash	Urgent	<input checked="" type="checkbox"/> WIP	
 <a href="#">000004</a>	Bug	Overlay windows cover only top of the page	Minor	Important	<input checked="" type="checkbox"/> Finished	
 <a href="#">000005</a>	API	Pagination option in item list API	Feature	Normal	<input checked="" type="checkbox"/> Finished	

Showing 5 issues in total 24 « Prev 1 2 3 4 5 Next »

ံ (၆.၂) - Example Issues List

**Log & Notification** – Issue တစ်ခုကို ထည့်သွင်းလိုက်တာနဲ့ ဘယ်သူထည့်သွင်းလိုက်တာလဲ၊ ဘယ်သူ Assign လုပ်လိုက်တာလဲ၊ ဆောင်ရွက်ပြီးစီးကြောင်း မှတ်တမ်းတင်တာဘယ်သူလဲ၊ ဘယ်သူပိတ်သိမ်းလိုက်တာလဲ၊ စတဲ့ Log ကို တွဲဖက်မှတ်တမ်း တင်နိုင်ဖို့လိုပါတယ်။ ဒါအပြင် Assign လုပ်လိုက်တဲ့အခါ Assign လုပ်ခြင်းခံရသူထံ Notify လုပ်ပေးဖို့ လိုသလို၊ အဲဒီ Issue နဲ့ ပက်သက်သူ အမြားသူများကထံ အပြောင်းအလဲတစ်ခုဖြစ်တိုင်း Notify လုပ်ပေးနိုင်ဖို့လည်းလို အပ်ပါတယ်။

ပုံ (၆.၃) - Example Issue Detail

ပုံ (၆.၃) မှာလေ့လာကြည့်ရင် Issue တစ်ခုရဲ့ Detail View နမူနာကို တွေ့နိုင်ပါတယ်။ Issue ရဲ့ အသေးစိတ် အချက် အလက်များနဲ့အတူ Log ကိုလည်း တွဲဖက်ဖော်ပြထားတာကို တွေ့ဖြင့်နိုင်ပါတယ်။

ဒါတွေဟာ Issue Tracking System တစ်ခုမှာ မဖြစ်မနေပါဝင်သင့်တဲ့ အခြေခံလုပ်ဆောင်ချက်တွေဖြစ်ပါတယ်။ ဒီထက် ပိုပြီး ပြည့်စုံတဲ့ System တွေမှာဆိုရင်တော့ အောက်ပါလုပ်ဆောင်ချက်များ ဖြည့်စွက်ပါဝင်တက်ပါသေး တယ်။

**VCS Integration** – Issue Tracking System ကို Version Control System နဲ့တွဲဖက်ထားနိုင်တဲ့ လုပ်ဆောင်ချက် ပါဝင်တက်ပါတယ်။ ဒီနည်းနဲ့ Issue နဲ့ သက်ဆိုင်ရာ Source Code ကို တိုက်ရှိက်တွဲဖက်ပေး ထားနိုင်ခြင်း၊ Issue ပိတ် သိမ်းချိန်မှာ ဖြေရှင်းချက် Commit ID ကို တွဲဖက်မှတ်တမ်းတင်နိုင်ခြင်းစတဲ့ ဖြည့်စွက် လုပ်ဆောင်ချက်တွေကို ရရှိနိုင် ပါတယ်။

**Duplicate Bug Detection** – Bug Report တွေမှာဖြစ်တက်တဲ့ပြဿနာက၊ Bug တစ်ခုကို တစ်ဦးကတွေ့လို မှတ်တမ်း တင်ထားပြီးခြင်းကိုမသိပဲ၊ နောက်တစ်ဦးက ထပ်မံမှတ်တမ်းတင်မိတဲ့အတွက် Duplicate တွေ ဖြစ် တက်ပါတယ်။ တစ်ချို့ Issue Tracking System တွေက Duplicate Bug တွေကို အလိုအလျောက် Detect လုပ် ပေးနိုင်တဲ့ လုပ်ဆောင် ချက်တွေ ပါဝင်တက်ပါတယ်။

**Custom Fields Management** – Bug Database တစ်ခုမှာ အခြေခံအဖြစ် ပါဝင်သင့်တဲ့အချက် (၅) ချက် ရှိကြောင်း ပြောခဲ့ပြီး ဖြစ်ပါတယ်။ Project တစ်ခုနဲ့တစ်ခုပါဝင်သင့်တဲ့ Field တွေ ကဲ့ပြားနိုင်ပါတယ်။ ဒါကြောင့်

Issue မှတ်တမ်း တင်ရာမှာ ဘယ်လိုအချက်တွေပါဝင်သင့်သလဲဆိုတဲ့ကိစ္စကို ပုံသေမထားပဲ လွပ်လပ်စာစီမံနိုင် တဲ့ လုပ်ဆောင်ချက်ပါဝင်မယ် ဆိုရင် Project လိုအပ်ချက်နဲ့အညီ Issue မှတ်တမ်းတင်ရာမှာ ဖြည့်သွင်းရမယ့် Field တွေကို လိုသလို စီမံနိုင်မှာဖြစ်ပါ တယ်။

**Time Tracking and Progress** – Issue တွေမှတ်တမ်းတင်ရာမှာ အခြီးသတ်ဆောင်ရွက်ရမယ့် Due Date တွေ သတ်မှတ်ပြီး Due Date ကျော်နောက်တဲ့ Issue တွေကို Calendar တစ်ခုပေါ်မှာ စုစုည်းလေ့လာနိုင်တဲ့ လုပ်ဆောင်ချက် ကလည်း အသုံးဝင်ပါတယ်။ ဒုံးအပြင် ဆောင်ရွက်ပြီးစီးမှုအခြေအနေ Progress Percentage နဲ့ အတူ စုစုပေါင်း Issue ဘယ်နှစ်ခုရှိနေသလဲ ပြီးခဲ့တဲ့အပါတယ် မှာ Issue ဘယ်နှစ်ခု ပိတ်သိမ်းလိုက်နိုင်သလဲ စ တဲ့ မှတ်သားဖွယ်အချက်အလက် တွေကို Chart များနဲ့ ဖော်ပြနိုင်တဲ့ လုပ်ဆောင်ချက်တွေ ပါဝင်မယ် ဆိုရင်လည်း အသုံးဝင်ပါတယ်။

**File Management** – Issue မှတ်တမ်းနဲ့အတူ ရည်ညွှန်းကိုးကားလိုတဲ့ Document တွေနဲ့ Screenshot တွေ အပါအဝင် ဖိုင်တွေကိုတွဲဖက်မှတ်တမ်းတင်ရလေ့ ရှိပါတယ်။ အဲဒီလို တွဲဖက်ထားတဲ့ ဖိုင်တွေကို သီးခြားစီမံလို ရတဲ့ File Manager လုပ်ဆောင်ချက်လည်း ဖြည့်စွက်ပါဝင်နိုင်ပါတယ်။

**Knowledge Base** – Bug Report ပြုလုပ်ရာတွင် လိုက်နာရမည့် စည်းမျဉ်းများ၊ Issue တစ်ခုပိတ်သိမ်းရာတွင် ဆောင် ရွက်ရမည့် Procedure များ၊ ဖြစ်လေ့ရှိသည့် ပြဿနာများနှင့် ငြင်းတို့အတွက် Solution များ၊ စတဲ့ အချက်အလက်တွေ ကို Knowledge Base တစ်ခုအနေနဲ့ တွဲဖက်မှတ်တမ်းတင်ထားနိုင်မယ်ဆိုရင် ပိုမိုပြည့်စုံသွားမှာဖြစ်ပါတယ်။

**Discussion** – ထည်သွင်းထားတဲ့ Issue အတွက် Assign လုပ်ခံရသူက မူလ Issue ထည်သွင်းသူနဲ့ မရှင်းလင်တဲ့ အချက် အလက်တွေ ပြန်လည်မေးမြန်တိုင်ပင်ခြင်း၊ Team Leader က Issue နဲ့ ပက်သက်တဲ့ မှတ်ချက်နဲ့ ညွှန်ကြားချက်တွေကို တွဲဖက် ပေးခြင်းစတဲ့ လုပ်ငန်းများ ဆောင်ရွက်နိုင်တဲ့ Comment, Forum လုပ်ဆောင်ချက်မျိုးတွေ ပါဝင်မယ်ဆိုရင်လည်း အကျိုးရှိစေမှာ ပဲဖြစ်ပါတယ်။

ဒီလိုလုပ်ဆောင်ချက်တွေ စုံလင်စွာပါဝင်တဲ့ Issue Tracking System တစ်ခုကို တစ်ခုကို Project Management System လို့လည်း ခေါ်ကြပါတယ်။ အမိကရည်ရွယ်ချက်က Issue တွေမှတ်တမ်းတင်ဖို့ဆိုပေမယ့် ဖြည့်စွက် လုပ်ဆောင် ချက်တွေရဲ့ အကူအညီနဲ့ Project တစ်ခုလုံးနဲ့ပက်သက်တဲ့ ကိစ္စတွေကို ဒီစနစ်ကနေတဲ့ဆင့် ပူးပေါင်းဆောင်ရွက်နိုင်မှာ ဖြစ်တဲ့အတွက် Project Management System လို့ ခေါ်ခြင်းလည်းဖြစ်ပါတယ်။

Software Team တစ်ခုအနေနဲ့ ထိရောက်အောင်မြင်တဲ့ ပူးပေါင်းဆောင်ရွက်မှုကို ရရှိလိုတယ်ဆိုရင်၊ Bug တွေ နည်းပြီး အရည်အသွေကောင်းတဲ့ Software တွေ ဖန်တီးလိုတယ်ဆိုရင်၊ Bug Database သို့မဟုတ် Issue Tracking System သို့မဟုတ် Project Management System တစ်ခုကို ရွေးချယ်အသုံးပြုနဲ့ လိုအပ်ပါတယ်။ တစ်ခုသတိပြုရမှာက၊ လုပ်ဆောင်ချက်တွေ များများပါတိုင်းလည်း ကောင်းတာတော့မဟုတ်ပါဘူး။ မလိုအပ်ပဲ ရှုပ်ထွေးပြီး အသုံးပြုရခက်ခဲ နေမယ်ဆိုရင်၊ Team အတွက် အထောက်အကူမဖြစ်တဲ့အပြင် အချိန်ပိုကုန်သွားနိုင်ပါတယ်။ ရိုးရိုးရှင်းရှင်းနဲ့ ကိုယ့်လိုအပ် ချက်ကိုကဲညီမယ့် စနစ်မျိုးကို ရွေးချယ် အသုံးပြု နိုင်ဖို့လိုပါတယ်။

## Conclusion

Software Team တစ်ခုအတွက် Issue Tracking System ကပေးနိုင်တဲ့ အကျိုးကျေးဇူးတွေ အများကြီးရှုပါတယ်။

Issue တွေကို မှတ်တမ်းတင်ရာမှာ သက်ဆိုင်ရာ Issue တစ်ခုချင်းကို တာဝန်ယူရမယ့်သူတွေပါ တစ်ခါတည်းတွဲဖက် မှတ်တမ်း တင်နိုင်တဲ့ အတွက် ဘယ် Issue က ဘယ်သူတာဝန်လဲဆိုတဲ့ ကိုစွဲလုံးဝရှင်းလင်း သွားမှာဖြစ်လို့ တာဝန်ခွဲဝေမှုနဲ့ တာဝန်ယူမှုပိုင်းမှာ ပိုမိုထိရောက်ရှင်းလင်းသွားမှာဖြစ်ပါတယ်။ ဒါအပြင် Issue တွေကို အရေးပါ၍ အဆင့်အလိုက်လည်း မှတ်တမ်းတင်ထား နိုင်တဲ့ အတွက် ပိုအရေးကြီးတဲ့ Issue တွေကို ဦးစားပေးဆောင်ရွက်ခြင်းအားဖြင့် ပိုမိုထိရောက်တဲ့ Time Management နဲ့ Release Schedule Management တိုကို ရရှိနိုင်မှာဖြစ်မှုပါ။

Developer နဲ့ Tester တွေအပြင် အဖွဲ့အစည်းအတွင်းက အခြားတာဝန်ရှိသူတွေကိုပါ Project Development မှာပါဝင် လာအောင်အားပေးတဲ့ အတွက် ဘက်စုံပူးပေါင်းဆောင်ရွက်မှုကို ရရှိနိုင်မှာဖြစ်ပါတယ်။ Issue Tracking System အကူ အညီနဲ့ Software ရဲ့ လက်ရှိ Progress ကို Developer မဟုတ်သူတွေကပါ သိရှိနိုင်လို့ လုပ်ငန်းဆောင်ရွက်ရာမှာ အထောက်အကူဖြစ်စေမှာ ပဲဖြစ်ပါတယ်။

Customer တွေအနေနဲ့လည်း Issue တွေကို လေ့လာခြင်းအားဖြင့် သူတို့သွားချင်တဲ့ Direction ဟုတ်မဟုတ်မှန်းဆန်း မှာဖြစ်တဲ့ အတွက် Feedback တွေကို စောစောစီးစီးပေးနိုင်မှာဖြစ်ပါတယ်။ ဒီနည်းနဲ့ Software Project တွေမှာရှိတက်တဲ့ မတိကျတဲ့ Requirement ပြသာနာကို တစ်ဘက်တစ်လမ်းက ပြေလည်စေနိုင်ပါတယ်။

ဒါကြောင့် အရည်အသွေးကောင်း Software Product တွေဖန်တီးမယ့် စွမ်းဆောင်ရည်မြင့် Software Team တစ်ခု အနေနဲ့ Issue Tracking System တွေကို ထိရောက်အောင် အသုံးချိန်ဖို့လိုပါတယ်။

ထိရောက်တဲ့ပူးပေါင်းဆောင်ရွက်မှု ဖြစ်စေဖို့အတွက်  
စနစ်တစ်ကျသတ်မှတ်ထားတဲ့ လုပ်ငန်းစဉ်နဲ့ ရှင်းလင်းမြင်သာတဲ့  
ဦးတည်ချက်တို့ လိုအပ်ပါတယ်။

### **Rockstar Developer Course**

Project Management, Web Service, Server Architecture

NodeJS အစရိုဘည် ဤစာအုပ်ပါ အကြောင်းအရာများကို  
ဘရေးသူကိုယ်တိုင် သင့်ကြားပေးခြင်းဖြစ်သည်။  
ဆက်သွယ်ရန် - (၀၉) ၇၃၁ ၆၅၄ ၆၂

<http://eimaung.com/courses>

## အခန်း(၇) – Team Collaboration

**အခန်း (၁)** မှာ Agile Principle အကြောင်း ပြောခဲ့တာမှတ်မိန္ဒြီးမှာပါ။ Agile Principle အရ Software Project တစ်ခုကို စီမံရာမှာ အခြေခံထားလိုက်နာရတဲ့ မူ (၅) ချက်ကို ပြန်လည်ဖော်ပြလိုက်ပါတယ်။

1. **Adaptive Planning** – ပုံသေသတ်မှတ်ထားခြင်းမဟုတ်ပဲ လိုအပ်သလို ပြင်လွယ်ပြောင်းလွယ်တဲ့ Plan ပေါ်မှာ အခြေခံရမယ်။
2. **Evolutionary Development** – Product တစ်ခုလုံးကို တစ်ခါတည်း တည်ဆောက်မယ့်အစား အဆင့် လိုက် တည်ဆောက်သွားရမယ်။
3. **Early Delivery** – အသုံးပြုသူက Product တစ်ခုလုံးပြီးစီးအချေသတ်အောင် စောင့်ဖို့မလိုပဲ စောနိုင်သမျှ အစောဆုံး Product ကို စတင်အသုံးပြုနိုင်ရမယ်။
4. **Continuous Improvement** – အဆင့်မြှင့်တင်မှုတွေကို အဆက်မပြတ် ဆောင်ရွက်သွားရမယ်။
5. **Rapid and Flexible Response to Change** – Requirement နဲ့ နည်းပညာ အပြောင်းအလဲရှိလာတိုင်း မှာ လိုအပ်သလိုလက်ခံပြောင်းလဲနိုင်တဲ့ Product တစ်ခုဖြစ်ရမယ်။

ဒီအချက်တွေကိုလေ့လာကြည့်ရင် စိတ်ဝင်စားဖို့ကောင်းသလောက် လက်တွေ့အကောင်အထည်ဖော်ဖို့ ခက်ခဲတဲ့ အချက်တွေ ဖြစ်တယ်ဆိုတာကို တွေ့ရမှာဖြစ်ပါတယ်။ တိကျုတဲ့ Requirement Specification တစ်ခုကိုသာ အခြေခံလို့ရမယ့်ဆိုရင် Software Project တစ်ခုကို စီမံခန့်ခွဲရတာ လွယ်နိုင်ပါတယ်။ Team မှာ ပါဝင်သူအားလုံးက တစ်ယောက်နဲ့တစ်ယောက် စကားတွေ့အများကြီး ပြောနေစရာမလိုပဲ Specification က သတ်မှတ်ချက်တွေအတိုင်း ကိုယ်တာဝန်ကျရာကို လုပ်လိုက် ယုံပါပဲ။

ဒါပေမယ့် လက်တွေ့မှာ တိကျုတဲ့ Specification တစ်ခုရဖို့ဆိုတာ မလွယ်ပါဘူး။ Agile က သတ်မှတ်ထားတဲ့ Adaptive Planning နဲ့ Flexible Response to Change ဆိုတဲ့ သတ်မှတ်ချက်များနဲ့အညီ အခြေအနေအရ လိုအပ်သလို ပြင်လွယ် ပြောင်းလွယ်တဲ့ Up-to-date Specification ပေါ်မှာ အခြေခံနိုင်ဖို့ လိုပါတယ်။ Spec ပြောင်းသွားတိုင်း ပြောင်းသွား တိုင်း၊ ပြောင်းရခြင်း ရည်ရွယ်ချက်ကိုနားလည်အောင်၊ ပြောင်းသွားတဲ့အချက်တွေကို သိရှိအောင်၊ သိရှိတဲ့အတိုင်း ညီညီ ညွတ်ညွတ်နဲ့ Team တစ်ခုလုံးက အမှန်တစ်ကယ်လိုက်နာဆောင်ရွက်အောင် ဆက်သွယ်ညိုနိုင်းရပါတော့မယ်။

Evolutionary Development, Early Delivery နဲ့ Continues Improvement ဆိုတဲ့ သဘောတရားတွေကလည်း ဒီအတိုင်းပါပဲ။ Software ကို တစ်ခါတည်း အစအဆုံးတည်ဆောက်အခြင်းမဟုတ်ပဲ အပိုင်းလိုက်အပိုင်းလိုက် အဆင့်မြှင့် တင်တည်ဆောက်သွားခြင်းဖြစ်လို့ မျက်စိစုံမှတ်ပြီး ငါတာဝန်ယူထားတာ Feature (၁၀) ခဲ့ အဲဒီ (၁၀) ခုကိုပဲ တစ်ခုခြားမှ တစ်ခုလုပ်တယ်ဆိုပြီး ထိုင်လုပ်နေလို့ မရတော့ပါဘူး။ Software ပထမအဆင့်မှာ ဘယ် Feature တွေ ပါဝင်မှာလဲ။ Software တစ်ခုလုံးပြီးအောင် မစောင့်ပဲ အဲဒီ ပထမအဆင့်ကို Release လုပ်နိုင်ဖို့ဆို

ရင် ငါးဘက်က ဘာတွေ လုပ်ပေး မလဲ။ စသဖြင့် တစ်ဦး Developer တွေနဲ့ အဆင့်တိုင်းအဆင့်တိုင်းမှာ ညီညွှန်းပြီး ပူးပေါင်းဆောင်ရွက်ရပါတွေ့မယ်။

ဒါကြောင့် ပူးပေါင်းဆက်သွယ်နိုင်မှု Collaboration နဲ့ ပူးပေါင်းဆောင်ရွက်နိုင်မှု Teamwork ကောင်းမှသာ Agile Principle တွေနဲ့ Software Project တစ်ခုကို စီမံနိုင်မှုဖြစ်ပါတယ်။ Collaboration မကောင်းပဲ Agile Principle တွေ ကိုသုံးဖို့ ကြိုးစားရင်၊ အောင်မြင်ဖို့ထက် တစ်ယောက်တစ်ပေါက်နဲ့ ပုံမရောက်ပီမရောက်ဖြစ်သွားဖို့က ပိုများပါတယ်။

ပြီးခဲ့တဲ့အခန်းမှာ ဖော်ပြခဲ့တဲ့ Issue Tracking System ဟာ Team Collaboration အတွက် များစွာအရေးပါတဲ့ Tool တစ်ခုဖြစ်ပါတယ်။ ဒီအခန်းမှာ နောက်ထပ်အရေးပါတဲ့ Team Collaboration နည်းစနစ်တွေအကြောင်းကို ဆက်လက် ဖော်ပြသွားပါမယ်။

ထိရောက်စွာပူးပေါင်းဆောင်ရွက်နိုင်တဲ့ Agile Software Team တစ်ခုရဲ့ အလုပ်လုပ်ပုံလုပ်ငန်းစဉ်က ဒီလိုဖြစ်ပါလို့မယ်။

1. Software Project တစ်ခု စတင်တွေ့မယ်ဆိုတာနဲ့ Project Manager (သို့မဟုတ်) Product Owner က **Functional Specification** တစ်ခုကို စတင်ရေးသားပါတယ်။
2. Specification ပေါ်အခြေခံပြီး **Project Schedule** ကိုရေးဆွဲထားရပါတယ်။
3. Specification မှာပါဝင်တဲ့ **User Story** တွေကိုထုတ်ယူပြီး **Product Backlog** ဖန်တီးပါတယ်။
4. ပြီးတဲ့အခါ Iteration တစ်ခုစတင်နိုင်ဖို့အတွက် **Sprint Planning Meeting** ခေါ်ပြီး **Sprint Backlog** ကို ဖန်တီးပါတယ်။
5. Development Team က Sprint Backlog မှာပါဝင်တဲ့ User Story တွေပေါ်မှာအခြေခံပြီး Implementation Detail အတွက် **Task List** ကို တည်ဆောက်ပါတယ်။
6. Sprint Backlog နဲ့ Task List ကို အခြေခံပြီး လိုအပ်တဲ့ Code တွေကို စတင်ရေးသားပါတယ်။ ဒီလိုရေးသားရာမှာ Code Quality ကောင်းစေဖို့အတွက် Code Review နည်းစနစ်တစ်ခုဖြစ်တဲ့ Pair Programming ကို အသုံးပြုပါတယ်။
7. တည်ဆောက်ရရှိလာတဲ့ ရလဒ်မှာတွေ့ရှိတဲ့ Bug နဲ့ Feature Request တွေကို **Issue Tracking System** ထဲမှာ မှတ်တမ်းတင်ပါတယ်။
8. Project Manager (သို့မဟုတ်) Product Owner က Issue Tracking System ထဲမှာ ရှိနေတဲ့ Bug နဲ့ Feature Request တွေရဲ့ အခြေအနေနဲ့ချိန်ဆပြီး Specification နဲ့ Schedule ကို Update လုပ်ပါတယ်။
9. **Up-to-date Schedule** နဲ့ချိန်ညီပြီး Product Backlog ကို ပြန်လည်စဉ်ပါတယ်။
10. ပြီးတဲ့အခါ အမှတ်စဉ် (၄) ကို ပြန်သွားပြီး နောက် Iteration တစ်ခုကို ပြန်လည်စတင်ပါတယ်။

လုပ်ငန်းစဉ်ထဲမှာပါဝင်တဲ့ Functional Specification, Project Schedule, User Story, Product Backlog, Sprint Planning Meeting, Sprint Backlog, Task List, Code Review, Pair Programming စာတဲ့ အခေါ်အဝေါ်တွေနဲ့ နည်းစနစ်တွေကို ဆက်လက်ဖော်ပြသွားပါမယ်။

## 7.1 – Specification

Team Collaboration အတွက် ပထမဦးဆုံးအနေဖြင့် အရေးပါတဲ့ အချက်ကတော့ Specification ဖြစ်ပါတယ်။ တိကျတဲ့ Spec တစ်ခုရဖို့ခက်ခဲတက်လို့ Specification ရေးဆွဲတဲ့လုပ်ငန်းအတွက် အလွန်အမင်းအချိန်ကုန်မဲ့သင့် ကြောင်းကို အခန်း (၁) မှာ ပြောခဲ့ပါတယ်။ ဒါပေမယ့် Spec လုံးဝရှိလိုတော့ မရပါဘူး။ လက်တွေမှာ Software မှာ ဘာတွေပါပြီး ဘယ်ပုံဘယ်နည်းအလုပ်လုပ်သင့်တယ်ဆိုတဲ့ Function Specification ကိုတော့ အတိအကျချ ရေးထားသင့်ပါတယ်။ အပြောင်းအလဲတွေကြောင့် မလိုအပ်ပဲ အချိန်တွေ မကုန်စေဖို့အတွက် ဘယ်လိုနည်းပညာတွေနဲ့ ဘယ်လိုတည်ဆောက်မယ် ဆိုတဲ့ Implementation Detail ကိုတော့ Spec ထဲမှာ ထည့်သွင်းမရေးသားပဲ ရှောင်သင့်ပါတယ်။

လက်တွေမှာ၊ Spec ကို လမ်းညွှန်အနေဖြင့်ကိုင်ပြီး Developer တွေက Software ကို ဖန်တီးကြရမှာပါ။ အဲဒါ Spec ကို ကိုင်ပြီးတော့ပဲ Tester တွေက ဒီ Software က အမှန်တစ်ကယ် မော်မှန်းထားတဲ့အတိုင်း အလုပ်လုပ်ခဲ့လား စမ်းသပ်ရမှာပါ။ Spec ရှိထားမှ Software မှာ ပါဝင်ရမယ့် အချက်တွေနဲ့ပက်သက်ပြီး Customer Manager တွေ၊ Tester တွေနဲ့ ပူပိုင်းဆောင်ရွက်ရာမှာ ဘုံးအဖြစ် ကိုကားစရာတစ်ခုကို ရရှိမှာ ဖြစ်ပါတယ်။

Spec ဆိုတာ Software ကို Code တွေရေးသားပြီး တစ်ကယ်မဖန်တီးမဲ့ ရှိုးရှိုးစာနဲ့ ရေးပြီး ကြိုတင်ဖန်တီးလိုက် တဲ့ သဘောဖြစ်လို့ Software ကို အစမ်းတည်ဆောက်လိုက်တာနဲ့တူပါတယ်။ ဒါကြောင့် Spec ရေးပြီးချိန်မှာ တွေကြုံရမယ့် အခက်အခဲတွေကို ကြိုတင်မှန်းဆမိပြီးဖြစ်လို့ အလုပ်ပို့တွင်သွားနိုင်ပါတယ်။

Function Specification တစ်ခုရှိထားခြင်းအားဖြင့် ရရှိလာတဲ့ ဖြည့်စွက်အကျိုးရလဒ်ကတော့၊ နောင်တစ်ချိန် Software ပြီးသွားလို့ Documentation တွေရေးတဲ့အခါမှာ ပြန်လည်အသုံးဝင်လာပါလိမ့်မယ်။ Documentation ကို အစအဆုံး အကုန်ပြန်ရေးနေစရာမလိုတော့ပဲ Spec ကိုပဲ End-user နားလည်တဲ့ အသုံးအနှံးနဲ့ ပြောင်းပေးလိုက်ရင် Documentation ဖြစ်လာနိုင်ပါတယ်။

Function Spec တစ်ခုမှာ ပါဝင်သင့်တဲ့အချက်အချို့ကို ဖော်ပြပေးလိုက်ပါတယ်။

**Overview** – Spec ရေးသားရာမှာ Software ရဲ့ ရည်ရွယ်ချက်နဲ့ ပါဝင်တဲ့ လုပ်ဆောင်ချက် Feature စာရင်းကို လိုရင်းတိုရင်း ဖော်ပြထားတဲ့ Overview နဲ့ အစပြုသင့်ပါတယ်။ ဥပမာ -

"ဤ Software သည် Bug နှင့် Feature များကို မှတ်တမ်းတင်နိုင်သည့် Bug Database အဖြစ် Software Team များတွင် အသုံးပြုရန်ဖြစ်သည်။ ဤ Software တွေ Developer, Tester နှင့် Manager တို့အတွက် Account များ တည်ဆောက်နိုင်သည့် လုပ်ဆောင်ချက်၊ Bug category များ စီမံနိုင်သည့်လုပ် ဆောင်ချက်နှင့် Bug များကို Priority နှင့် Status အလိုက် ခွဲခြားစီမံနိုင်သည့် လုပ်ဆောင်ချက်တို့ ပါဝင်မည်ဖြစ်သည်။"

**Scenarios** – Spec တစ်ခုမှာ ဆက်လက် ပါဝင်သင့်တာကတော့ User Story တွေပဲဖြစ်ပါတယ်။ Software ရဲ့ အလုပ် လုပ်ပုံကို အသုံးပြုသူ User တစ်ဦးချင်းစီရဲ့ ရှုထောင့်ကနေ ရေးသားသတ်မှတ်ထားခြင်း ဖြစ်ပါတယ်။ ဥပမာ -

"Project Manager က သတ်မှတ်ထားသည့် Username/Password ကို သုံး၍ Login ဝင်သည်။ ထိုနောက် Dashboard တွေပေါ်လာသည့် Project များအတွင်းမှ စီမံလိုသည့် Project ကို ရွေးချယ်သည်။ ရွေးချယ်လိုက်သည့် Project အတွက်မှတ်တမ်းတင်ထားသည့် Issue စာရင်းပေါ်လာသည့် အခါ ၏ Status Filter Option မှ Assigned ကို ရွေးချယ်ခြင်းအားဖြင့် Assign လုပ်ယုံသာလုပ်ရသေးပြီး စတင်ဆောင်ရွက်ခြင်းမရှိသေးသည် Issue များကို ရွေးချယ် ကြည့်ရှုသည်။"

စသဖြင့် User Role အမျိုးမျိုးနဲ့ ပါဝင်တဲ့ Feature အမျိုးမျိုးတို့အတွက် User Story တွေကို တက်နိုင်သလောက်ပြည့်စုံအောင် ရေးသားဖော်ပြထားသင့်ပါတယ်။ ပေးထားတဲ့ User Story အရ Login Authentication လုပ်ဆောင်ချက်ပါဝင်ရပါတော့မယ်။ အချက်အလက်တွေကို စုစည်းဖော်ပြပေးတဲ့ Dashboard လုပ်ဆောင်ချက်ပါဝင်ရပါတော့မယ်။ Multiple Project လုပ်ဆောင်ချက် ပါဝင်ရပါတော့မယ်။ Issue List, Issue Filter နဲ့ Issue Status တို့ပါဝင်ရပါတော့ မယ်။

User Story ဆိုတာ ဒါပါပဲ။ ဘာလုပ်ရမယ်လို့ သတ်မှတ်ထားခြင်းမဟုတ်ပဲ ဘယ်လိုလိုချင်တယ်လို့ သတ်မှတ်ထားခြင်းဖြစ် ပါတယ်။ လုပ်ရမယ့်အလုပ်ကို Spec ထဲမှာ အတိအကျသတ်မှတ်ထားခြင်းမဟုတ်ပဲ လိုချင်တဲ့ အနေအထားကိုသာ သတ်မှတ်ထားခြင်းဖြစ်ပါတယ်။

**Non Goals** – Spec မှာ ဖော်ပြသင့်တဲ့ နောက်တစ်ချက်ကတော့ "မလုပ်ဖို့ဆုံးဖြတ်ထားတဲ့အချက်များ" ဖြစ်ပါတယ်။ Spec စတင်မရေးသားမဲ့ တိုင်ပင်ဆွေးနွေးပြီးရလာတဲ့ ရလဒ်အရတစ်ချို့ လုပ်ဆောင်ချက်တွေကို ရည်ရွယ်ချက်ရှိရှိ မလုပ်ပနေဖို့ ဆုံးဖြတ်ထားတက်ကြပါတယ်။ ဥပမာ –

"Internet Explorer Version 8 နှင့် ရှှေ့ပိုင်း Version အဟောင်းတွေကို Support မလုပ်ပါ။"

- စသဖြင့် တစ်ခါတည်းထည့်သွင်းရေးသားထားရပါတယ်။ မလုပ်ရခြင်း အကြောင်း ရင်းကိုလည်း တဲ့ဖက်ဖော်ပြထားနိုင် ရင် ပိုကောင်းပါတယ်။ နောင်လက်တွေ့ Software ကို ရေးသားတော့မှ ဒီလုပ်ဆောင်ချက်လေးထည့်လုပ်လိုက်ရင် ကောင်းမယ်လို့ စိတ်ကူးပေါ်လာပြီး လုပ်မိတဲ့အခါ မလိုအပ်ပဲ အချိန်တွေကုန်တက်ပါတယ်။ မူလကတည်းက အကြောင်းရှိ လို့ မလုပ်ဖို့ ဆုံးဖြတ်ထားတဲ့အချက်တွေကို တစ်ခါတည်းရှင်းအောင် ထည့်ရေးထားရပါတယ်။

**Screen Flow** – Screen Flow Diagram တစ်ခုလဲ Spec မှာပါသင့်ပါသေးတယ်။ Home Screen ကနေ Register ကို နှိပ်လိုက်ရင် ရောက်သွားမယ့် Screen, အဲဒီ Screen ကမှ Register Button ကို နှိပ်လိုက်ရင် ရောက်သွားမယ့် Screen, Login ကို နှိပ်ပြီး Login ဝင်ပြီးနောက်ရောက်သွားမယ့် Screen စသဖြင့် User Interface Screen တွေရဲ့ ဆက်စပ်မှုကို Diagram တစ်ခုနဲ့ ဖော်ပြထားသင့်ပါတယ်။

**Screen by Screen Detail** – ဆက်လက်ပြီး Screen တစ်ခုချင်းစီမှာ ပါဝင်မယ့် အချက်အလက်အသေးစိတ်ကို ဆက်လက်ဖော်ပြရပါတယ်။ ဥပမာ –

"Login Screen မှာ Login ဝင်ဖို့အတွက် Username/Password ထည့်သွင်းနိုင်တဲ့ Form ပါဝင်မယ်။ Password

မြေနေရင် Account Recover လုပ်နိုင်တဲ့လုပ်ဆောင်ချက်ပါဝင်မယ်။ Remember Me Option ပါဝင်မယ်။ Account အသစ်ဆောက်ချင်ရင် Register Page ကို သွားနိုင်တဲ့ လုပ်ဆောင်ချက် ပါဝင်မယ်။"

"Dashboard မှာ Project စာရင်းဖော်ပြုမယ်၊ စုစုပေါင်း Open Issue အရေအတွက်ကို ဖော်ပြုမယ်။ Logout လုပ်ဆောင်ချက် ပါဝင်မယ်။ Account Password ပြောင်းနိုင်တဲ့ Setting လုပ်ဆောင်ချက်ပါဝင်မယ်။"

စသဖြင့် Screen တစ်ခုချင်းမှာ ပါဝင်မယ့် အချက်အလက်တွေကို အသေးစိတ် ဖော်ပြရပါတယ်။ အကြမ်း ဆွဲထားတဲ့ User Interface Prototype တွေရှိရင်လည်း တဲ့ဖော်ထားသင့်ပါတယ်။

**Todo** – တစ်ချို့ ချက်ခြင်းမဆုံးဖြတ်နိုင်သေးတဲ့ အချက်တွေရှိရင် Todo အနေနဲ့ မြင်သာအောင် မှတ်ချက် ထည့်သွင်းထား သင့်ပါတယ်။ တစ်ကယ်တာဝန်ယူရမယ့် Developer (သို့မဟုတ်) သက်ဆုံင်ရာကဏ္ဍမှာ ကျွမ်းကျင်တဲ့ Expert နဲ့ တိုင်ပင် ပြီးမှ ရေးသားဖော်ပြုမယ့်အချက်တွေကို "အသေးစိတ်ဆွေးနွေးပြီးမှ ဖော်ပြရနဲ့" လို့ မြင်သာအောင် မှတ်ချက်ပေးထားရပါမယ်။

**Up-to-date** – နောက်ဆုံးအနေနဲ့ သတိပြုသင့်တာကတော့ Spec ကို ရေးပြီး၊ အဲဒီအတိုင်း မဖြစ်ဖြစ်အောင် လုပ်ရမယ် လို့ ပုံသေမယူဆပဲ ပြောင်းလဲလာတဲ့ Requirement နဲ့ အခြေအနေနဲ့ ဆီလျှော်အောင် အခါအား လျှော်စွာ ပြင်ဆင်ပြီး အမြဲ Update ဖြစ်နေအောင် စီစဉ်ထားသင့်ပါတယ်။

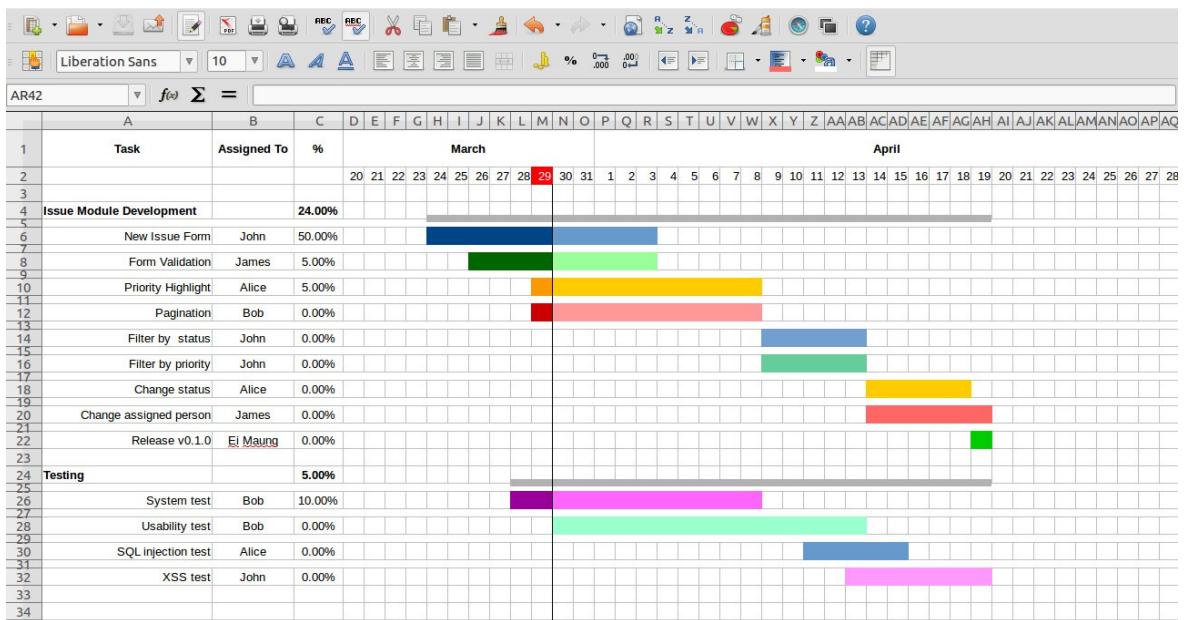
Team နဲ့ အလုပ်လုပ်ရတဲ့အခါ အားလုံးက Developer တွေချည်းမဟုတ်ပါဘူး။ Manager တွေ၊ Tester တွေ၊ Admin Staff တွေ၊ Marketing သမားတွေနဲ့ User တွေလည်း ပါဝင်နိုင်ပါတယ်။ Software မှာပါဝင်တဲ့ Feature တွေနဲ့ ပက်သက်ပြီး ပြောစရာရှိလာတဲ့အခါ Meeting ခန်းတွေထဲမှာ ဟိုတုန်းကပါဟိုလိုပြောခဲ့တယ်၊ ဒီတုန်းကဒီ လိုပြောခဲ့တယ်လို့ အငြင်းအခုံတွေလုပ်ပြီး အချိန်ကုန်မယ့်အစား အသေအချာမှတ်တမ်းတင်ထားတဲ့ Spec တစ်ခုကို ကိုင်ပြီး ဆက်သွယ် ဆောင်ရွက်တာက ပိုပြီးထိရောက်မှာပဲ ဖြစ်ပါတယ်။

## 7.2 – Project Scheduling

ပြီးခဲ့တဲ့အခန်းမှာ ဖော်ပြခဲ့တဲ့ Software Team တစ်ခုရဲ့အရည်အသေးကို တိုင်းတာတဲ့ အချက် (၁၂) ချက်ထဲမှာ Up-to-date Schedule တစ်ခု ထားရှိရှိလားဆိုတဲ့အချက်လည်း တစ်ချက်အပါအဝင်ဖြစ်ပါတယ်။ Requirement တစ်ခုသာ တိတိကျကျရှိ မယ်ဆုံးရင် Software Project တစ်ခုရဲ့ Release Schedule ရေးဆွဲရတာ ဘာမှမခက်ပါဘူး။ ဒါပေမယ့် Requirement ဆိုတာ ပြောင်းလဲတက်တဲ့အတွက် သူပြောင်းတိုင်း Schedule ကလည်း သင့်တော်သလို လိုက်ပြောင်းနိုင်ဖို့ လိုပါတယ်။ Project Schedule တစ်ခုကိုရေးဆွဲတဲ့အခါ Specification မှာ ပါဝင်တဲ့ User Story တွေပေါ်မှာ အခြေခံ ရေးဆွဲရမှာ ဖြစ်ပါတယ်။ တန်ညွှေးအားဖြင့် Specification နဲ့ User Story ပြောင်းသွားရင် Schedule လည်း လိုက်ပြောင်းနိုင်ရမှာပဲ ဖြစ်ပါတယ်။

### The Gantt Chart

Project Schedule ရေးဆွဲရာမှာ အသုံးဝင်တဲ့ နည်းစနစ်ကတော့ The Gantt Chart ဖြစ်ပါတယ်။ Bar Chart တစ်ချိုး ကိုသုံးပြီး Task တစ်ခုချင်းစီရဲ့ စုစုပေါင်းမယ့်အချိန်နဲ့ ဆုံးမယ်အချိန်တို့ကို ဖော်ပြခြင်းအားဖြင့် Project ရဲ့ Progress ကို မြင်သာအောင် သတ်မှတ်တဲ့နည်းလမ်းဖြစ်ပါတယ်။

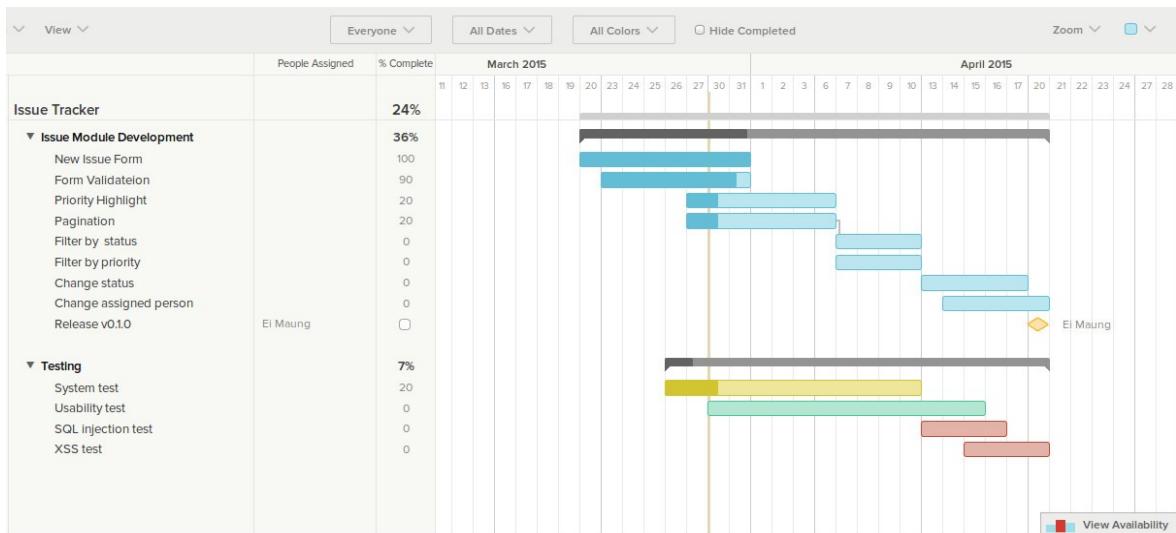


ပုံ (၂.၁) - Example Gantt Chart in Spreadsheet

ပုံ (၇.၁) မှာ လေ့လာကြည့်ရင် Spreadsheet တစ်ခုကိုအသုံးပြုပြီး Project Plan တစ်ခုကို The Gantt Chart အနေနဲ့ ရေးဆွဲထားတာကိုတွေ့ရမှာ ဖြစ်ပါတယ်။ Chart ရဲ့ အတိုင်အဖြစ် Task, Assigned To နဲ့ Progress Percentage တို့ကိုထားပါတယ်။ အတန်းအနေနဲ့ ရက်စွဲကိုထားပါတယ်။ ယနေ့ရက်စွဲကို ဖြင့်သာအောင် ကန်လတ်ဖြတ်လိုင်းတစ်ခုနဲ့ဖော်ပြထားပါသေးတယ်။ Task တစ်ခုချင်းစီအတွက် စမယ့်ရက်နဲ့ ဆုံးမယ့်ရက်ကို Bar နဲ့ဖော်ပြထားသလို ပြီးတိမ် အတွက် အရောင်စွဲပြီး ဖော်ပြထားပါသေးတယ်။ ပြီးတော့ Task တွေကို အမျိုးတူရာစွဲပြီး အုပ်စု ခဲ့ ထားပါသေးတယ်။ နမူနာမှာ Issue Module Development နဲ့ Testing ဆုံးပြုပြီး Task အုပ်စုနှစ်ခု ရှိပါတယ်။

ဒီနည်းနဲ့ Project Deadline အဖြစ် လက်စွဲထားအသုံးပြုနိုင်တဲ့ Up-to-date Schedule တစ်ခုကို ရရှိနိုင်မှာပဲဖြစ်ပါတယ်။ Development Team မှာမပါဝင်တဲ့ Management ပိုင်းကလူတွေ၊ Customer တွေ၊ Marketing သမားတွေနဲ့ ဆက်သွယ်ဆောင်ရွက်ရာမှာလည်း ဒီ Chart ကိုပဲ ကိုးကားအသုံးပြုနိုင်မှာဖြစ်ပါတယ်။

Microsoft Excel တို့ LibreOffice Calc တို့လို့ Spreadsheet Software တစ်ခုကိုသုံးပြီး Gantt Chart Project Schedule ကို အလွယ်တစ်ကူ တည်ဆောက်နိုင်ပါတယ်။ ပို့ပြီးထိရောက်ချင်ရင်တော့ Gantt Chart လုပ်ဆောင်ချက် ပါဝင်တဲ့ Project Management Software တွေကို အသုံးပြုနိုင်ပါတယ်။ အဲဒီ Software တွေက User Story (သို့) Task List ကို ပြုပြင်လိုက်ယုံနဲ့ Project Schedule ကို အလိုအလျောက် Update လုပ်ပေးနိုင်ကြပါတယ်။



ပုံ (၇.၂) - Example Project Schedule on Team Gantt

ပုံ (၇.၂) မှာ နမူနာအနေနဲ့ ဖော်ပြထားတာကတော့ Team Gantt လိုခေါ်တဲ့ Project Management Software တစ်ခု ဖြစ်ပါတယ်။ ဘယ်ဘက်ခြစ်းမှာရှိတဲ့ List ကို Update လုပ်ပေးလိုက်ရင် ညာဘက်ခြစ်းက Project Schedule ကလည်း အလိုအလျောက် Update ဖြစ်သွားမှာပဲဖြစ်ပါတယ်။ ဒီနည်းနဲ့ List ပြောင်းသွားတိုင်း အလိုအလျောက် Up-to-date ဖြစ်သွားတဲ့ Project Schedule ကို ရရှိနိုင်မှာဖြစ်ပါတယ်။ အောက်ပါလိပ်စာမှာ စမ်းသပ်အသုံးပြုနိုင်ပါတယ်။

<http://teamgantt.com/>

တစ်ခုသတိပြုသင့်တာက၊ Software Project တစ်ခုမှာပါဝင်တဲ့ Task တွေ၊ User Story တွေ၊ Feature တွေက အများကြီးရှိမှာပါ။ အဲဒီ Task တွေအားလုံးကို Schedule ထဲမှာ အသေးစိတ်မထည့်သင့်ပါဘူး။ အထူးသဖွင့် Specification ထဲမှာ ဖော်ပြထားတဲ့ User Story တွေကို အခြေခံပြီးတော့သာ အကျဉ်းချုပ်သဘောမျိုး ရေးဆွဲထားသင့်ပါတယ်။

### 7.3 – SCRUM and Backlog

**SCRUM** ဆိုတာ အခန်း (၁) မှာ ဖော်ပြခဲ့တဲ့ Iterative Development Model နဲ့အတူ တဲ့ဖက်သုံးတဲ့ Team Collaboration နည်းစနစ်တစ်မျိုး ဖြစ်ပါတယ်။ SCRUM အကြောင်းပြာတော့မယ်ဆိုရင် SCRUM Team တစ်ခု မှာ ပါဝင်သူတွေရဲ့ တာဝန်ခွဲဝေပုံကနေ စပြောရပါမယ်။ SCRUM Team တစ်ခုမှာ Product Owner, SCRUM Master နဲ့ Development Team ဆိုပြီး သုံးပိုင်းပါဝင်ပါတယ်။

**Product Owner** ဆိုတာ Stakeholder (Customer, Higher Management) တွေကို ကိုယ်စားပြုပြီး Product မှာ ပါဝင်ရမယ့် လုပ်ဆောင်ချက်တွေကို ဆုံးဖြတ်သူဖြစ်ပါတယ်။ Stakeholder တွေနဲ့ညီးပြီး စကားပြောရမယ့် အပိုင်းကို တာဝန်ယူသူလည်းဖြစ်ပါတယ်။ တနည်းအားဖြင့် Product Owner ဆိုတာ Stakeholder နဲ့ Development Team ကြားက ကြားခံဆက်သွယ်ဆောင်ရွက်ပေးသူပဲ ဖြစ်ပါတယ်။ ရုံးရုံး Team တစ်ခုရဲ့ Project

Manager နဲ့ သဘောတရား ဆင်ပေမယ့် Project Manager လို့ Developer တွေကို ဘယ်သူဘာလုပ်ရမယ်လို့ တာဝန်ခွဲခေ ပေးတာတွေဘာတွေ မလုပ်ပါဘူး။ ပါဝင်ရမယ့် Feature (User Story) တွေနဲ့ အဲဒီ Feature တွေရဲ့ ဦးစားပေးအဆင့်ကိုသာ စီမံဆုံးဖြတ်သူ ဖြစ်ပါတယ်။

**Development Team** ကတော့ Software ကို အမှန်တစ်ကယ် ရေးသားမယ့်သူတွေဖြစ်ပါတယ်။ ရီးရိုး Team တွေနဲ့ မတူပဲ ထူးခြားသွားတာကာ မှာပါဝင်သူတွေရဲ့ တာဝန်ကို Product Owner စုံ SCRUM Master တို့က မဆုံးဖြတ်ပဲ သူတို့ကိုယ်တိုင် ဆုံးဖြတ်ပြီး သူတို့ကိုယ်တိုင် တာဝန်ယူသွားမယ့် Self-organized Team ပဲဖြစ်ပါ တယ်။

**SCRUM Master** ရဲတာဝန်ကတော့ Development Team ဟာ SCRUM Practice တွေကို လိုက်နာရဲ့လား။ မလိုက် နာနိုင်တဲ့သူ (သို့မဟုတ်) မလိုက်နာနိုင်လောက်စရာ အကြောင်းတွေရှိနေသလား စတဲ့အချက်တွေကို လေ့လာသုံးသပ်ပြီး SCRUM Practice တွေကို အပြည့်အဝလိုက်နာနိုင်ရေးအတွက် လိုအပ်တဲ့ စီမံဆောင်ရွက်ပေးရသူဖြစ်ပါတယ်။ SCRUM Meeting တွေကို ဦးဆောင်ရပါတယ်၊ လိုအပ်တဲ့ SCRUM Training တွေပေးရပါတယ်၊ Developer တွေ ကိုယ်တိုင်တာဝန်ယူနိုင်မှုကို လမ်းညွှန်အားပေးရပါတယ်။

## Sprint

SCRUM နည်းစနစ်မှာ အဓိကအကျဆုံးတစ်ခုကတော့ Sprint ဖြစ်ပါတယ်။ Iterative Development Model ဆိုတာ Software တစ်ခုလုံးကို အစအဆုံးတည်ဆောက်မယ့်အစား၊ လုပ်ဆောင်ချက်တစ်ခုချင်းကို သုံးလို့ရတဲ့အထိ အပိုင်းလိုက် အပိုင်းလိုက် ခွဲခြားတည်ဆောက်သွားတဲ့ နည်းစနစ်တစ်ခုဖြစ်တယ်လို့ ပြောခဲ့တာကို မှတ်မီးမှာ ပါ။ Sprint ဆိုတာကတော့ အချိန်ကာလတစ်ခု သတ်မှတ်ပြီး အဲဒီကာလအတွင်းမှာ လုပ်ဆောင်ချက်တစ်ခု (သို့မဟုတ်) လုပ်ဆောင်ချက်တစ်ချို့ကို အသုံးပြုလိုရတဲ့အဆင့်ထိ ပြီးမြောက်အောင် ဆောင်ရွက်ဖို့ ကန်သတ်လိုက်တဲ့ အချိန်ကာလသတ်မှတ်ချက်တစ်ခုကို ဆိုလို တာပါ။ အချိန်သတ်မှတ်ချက်ဟာ Team တစ်ခုနဲ့တစ်ခုမ တူပါဘူး။ တစ်ချို့က (၁) ပါတ် သတ်မှတ်တက်ကြပါတယ်။ တစ်ချို့က (၂) ပါတ် သတ်မှတ်တက်ကြပါတယ်။ တစ်ချို့လည်း (၁) လထိ သတ်မှတ်တက်ကြပါတယ်။ (၂) ပါတ်က လက်တွေ့မှာ အသင့်တော်ဆုံးဖြစ်ပါတယ်။

Sprint တစ်ခုကို စတင်တော့မယ်ဆိုတာနဲ့ Team တစ်ခုလုံးပါဝင်တဲ့ Sprint Meeting ခေါ်ရပါတယ်။ Sprint Meeting မှာ Sprint Backlog တစ်ခုကို တည်ဆောက်နိုင်ဖို့ ညီးမြှုပ်းကြရပါတယ်။ Sprint Backlog ဆိုတာ သတ်မှတ်ထားတဲ့ အချိန်ကာလအတွင်း အပြီးဆောင်ရွက်မယ့် Feature စာရင်းပဲ ဖြစ်ပါတယ်။ သတ်မှတ်ကာလ အတွင်းမှာ သတ်မှတ်ထားတဲ့ Backlog ထဲက Feature တွေကို အပြီးဆောင်ရွက်ကြရမှာ ဖြစ်ပါတယ်။ Sprint တစ်ခုပြီးသွားရင် အသုံးချို့ရတဲ့ Feature တစ်ခု အနည်းဆုံး ထွက်ပေါ်လာသင့်ပါတယ်။ Feature တစ်ခု ထွက်ပေါ်လာတယ်ဆိုတာ အဖြစ်ထွက်ပေါ်လာခြင်း မျိုးမဟုတ်ပဲ၊ အမှန်တစ်ကယ် Release လုပ်လို့ရတဲ့ အခြေအနေ ထိ ထွက်ပေါ်လာရမှာ ဖြစ်ပါတယ်။ ဒါပေမယ့် Feature ထွက်ပေါ်မလာလို့ဆိုပြီး Sprint ရဲ့ အချိန်ကို သက်တမ်း မတိုးရပါဘူး။ သတ်မှတ်ရက်ပြည့်တာနဲ့ ရပ်ပြီး Sprint Review Meeting ခေါ်ရပါတယ်။ ထွက်ပေါ်လာတဲ့ရလဒ်ကို လေ့လာပြီး၊ မျှော်မှန်းထားသလို မအောင်မြင်ခဲ့ရင်လည်း မအောင်မြင်ရ ခြင်းအကြောင်းရင်းကို စမ်းစစ်ပြီး၊ နောက် Sprint တစ်ခု စတင်ဖို့ ပြင်ရပါတယ်။

ဒီနည်းနဲ့ Sprint တွေကိုသုံးပြီး Iterative Development Model ကို အသက်ဝင်စေခြင်းဖြစ်ပါတယ်။

Sprint နဲ့ သဘောသဘာဝတူတဲ့ (၂) ခဲ့ရှိပါသေးတယ်။ Spike နဲ့ Tracer Bullet လိုပေါ်ကြပါတယ်။ Sprint လိုပဲ အချိန်ကာလသတ်မှတ်ပြီး ဆောင်ရွက်ပေမယ့် အမှန်တစ်ကယ် Feature တစ်ခုကို Develop လုပ်ခြင်းမဟုတ်ပဲ အကြောင်းအရာတစ်ခုကို စမ်းသပ်လေ့လာဖို့ ရည်ရွယ်ခြင်းကို Spike လိုပေါ်ပါတယ်။ သုတေသနလုပ်ဖို့လိုတဲ့ ကိစ္စတွေနဲ့ လေ့လာသူယူဖို့ လိုတဲ့ ကိစ္စတွေကို Sprint လို အချိန်သတ်မှတ်ပြီး သတ်မှတ်ချိန်အတွင်း အပြီး လေ့လာခြင်းဖြစ်ပါတယ်။

နောက်တစ်ခုကတော့ Sprint လိုပဲ အချိန်ကာလသတ်မှတ်ပြီး ဆောင်ရွက်ပေမယ့် မြင်သာတဲ့ Feature မဟုတ်ပဲ၊ Code Quality ပိုကောင်းအောင် ဆောင်ရွက်တဲ့ ကိစ္စတွေကိုတော့ Tracer Bullet လို ပေါ်ပါတယ်။ Code Refactoring ကို Sprint လို အချိန်သတ်မှတ်ပြီး သတ်မှတ်ချိန်အတွင်း အပြီးဆောင်ရွက်ခြင်းဖြစ်ပါတယ်လို ဆိုနိုင်ပါတယ်။ ပုံမှန်အားဖြင့် Sprint တစ်ခုနဲ့တစ်ခုကြားထဲမှာ လိုအပ်ရင် Spike နဲ့ Tracer Bullet တွေကို ကြားဖြတ်ဆောင်ရွက် တက်ကြပါတယ်။

### Sprint Planning Meeting

Sprint တစ်ခုစတင်တော့မယ်ဆိုတာနဲ့ Sprint Planning Meeting ခေါ်ပါတယ်။ အဲဒီ Meeting မှာ သတ်မှတ်ထားတဲ့ Sprint ကာလအတွင်း ဘာတွေလုပ်မှုလည်း တစ်ခါတည်းဆုံးဖြတ်ရပါတယ်။ ပုံမှန်အားဖြင့် Sprint Planning Meeting ရဲ့ ပထမပိုင်းမှာ Sprint ပြီးသွားတဲ့အခါ ထွက်ပေါ်လာဖို့ မျှော်မှန်းတဲ့ ရလဒ်ကို ဆုံးဖြတ်ရပါတယ်။ အဲဒီရလဒ် ရရှိဖို့အတွက် ဆောင်ရွက်ရမယ့် လုပ်ငန်းတွေကဘာတွေလည်း ဆုံးဖြတ်ရပါတယ်။

Meeting ဒုတိယပိုင်းမှာတော့ သတ်မှတ်ထားတဲ့ ရလဒ်ထွက်ပေါ်အောင် ဆောင်ရွက်ရမယ့် Feature တွေကို ရွေးထုတ်ပြီး Sprint Backlog တစ်ခုတည်ဆောက်ခြင်း၊ Implementation Plan အသေးစိတ်တိုင်ပင်ခြင်း တို့ကို Development Team က ဆက်လက်ဆောင်ရွက်ရပါတယ်။

### Daily SCRUM Meeting

Sprint တစ်ခု စပြီဆိုတာနဲ့ Daily SCRUM Meeting ကို နေ့စဉ်ခေါ်ရပါတယ်။ အကြာကြီး လုပ်စရာမလိုပါဘူး။ (၁၅) မိနစ် Meeting လောက်ဆိုရင်လုံလောက်ပါတယ်။ ထိုင်ပြောမှုလည်းမဟုတ်ပါဘူး။ ပေါ့ပေါ့ပါးပါး မတ်တက်ရပ်ဆွေးနွေး လည်းရပါတယ်။ ဒါပေမယ့် အချိန်နဲ့ဆောရာကိုတော့ ပုံသေသတ်မှတ်ထားဖို့လိုပါတယ်။ အကြောင်းအမျိုးမျိုးနဲ့ တစ်ဦး တစ်ယောက် မပါဝင်နိုင်ရင်လည်း မစောင့်ရပါဘူး။ သတ်မှတ်ထားတဲ့ အချိန်မှာ Meeting ကို ပုံမှန်စတင်ရပါတယ်။ SCRUM Master က ဒီ Meeting ကို တာဝန်ယူ Coordinate လုပ်ပေးရပါတယ်။

Daily SCRUM Meeting မှာ အချက် (၃)ချက်ကို အဓိကထားဆွေးနွေးရပါတယ်။ Sprint Planning Meeting မှာ သတ်မှတ်ခဲ့တဲ့ မျှော်မှန်းချက် အောင်မြင်ဖို့အတွက် မနေ့က ဘာတွေလုပ်ခဲ့သလဲ။ ဒီနေ့ဘာတွေဆက်လုပ်မလဲ။ အခက်အခဲ ဘာရှိနေသလဲ စတဲ့အချက် (၃) ချက်ကို အဓိကဆွေးနွေးရမှာဖြစ်ပါတယ်။

### Sprint Review Meeting

Sprint တစ်ခုအတွက် သတ်မှတ်ချိန်ပြည့်တာနဲ့ လုပ်ငန်းတွေအားလုံး ပြီးသည်ဖြစ်စေ၊ မပြီးသည်ဖြစ်စေ Sprint Review Meeting ခေါ်ရပါတယ်။ Sprint Review Meeting မှာလည်း (၂) ပိုင်း ပါဝင်ပါတယ်။ ပထမတစ်ပိုင်းမှာ

အောင်မြင်ပြီးစီး ခဲ့တဲ့ Feature တွေကာဘတွေလဲ၊ မအောင်မြင်ပဲ ကျန်သွားတာတွေကာဘတွေလည်း Review လုပ်ရပါတယ်။ ပြီးတဲ့အခါ အောင်မြင်ပြီးစီး ခဲ့တဲ့ Feature တွေကို Demo ပြရပါတယ်။

နောက်တစ်ပိုင်းမှာတော့၊ ပြီးခဲ့တဲ့ Sprint နဲ့ပက်သက်ပြီး Developer အားလုံးရဲ့မှတ်ချက်ကို ပေးရပါတယ်။ အောင်မြင်မှာ၊ မအောင်မြင်မှာတွေကို နောက် Sprint တွေမှာ ကိုးကားနိုင်အောင် သင်ခန်းစာထားလေ့လာရပါတယ်။ နောင် Sprint တွေမှာ သတိပြုဆောင်ရွက်သင့်တဲ့ အချက်တွေကို ဈေးနွေးတိုင်ပင်ရပါတယ်။

## Product Backlog and Sprint Backlog

Product Backlog ဆိုတာဟာ Software တစ်ခုလုံးပြီးမောက် အောင်မြင်ဖို့အတွက် ဆောင်ရွက်ရမယ့် User Story တွေကို စုစုပေါင်းထားခြင်းဖြစ်ပါတယ်။ Sprint Backlog ဆိုတာကတော့ Sprint တစ်ခုစတင်တော့မယ်ဆိုတာနဲ့ Sprint မှာ ဆောင်ရွက်မယ့် Feature (User Story) တွေကို Product Backlog ထဲကနေ ဈေးထုတ်ထားတဲ့ စာရင်းဖြစ်ပါတယ်။

Product Backlog ကို တာဝန်ယူစီစဉ်သူကတော့ Product Owner ဖြစ်ပါတယ်။ Product Owner ကပဲ Product Backlog မှာပါဝင်တဲ့ User Story တွေရဲ့ အရေးပါမှုအဆင့် Priority ကိုဆုံးဖြတ်စီစဉ်ပါတယ်။ Product Backlog ထဲက User Story တွေဟာ အမှန်တော့၊ Specification ထဲက User Story တွေပဲဖြစ်ပါတယ်။ ထူးခြားစာကတော့ Spec ထဲ မှာ User Story တွေကို ရေးတဲ့အခါ ပြည့်စုံအောင်ရေးရပေးမယ့်၊ Backlog ထဲရောက်တဲ့အခါ မှာတော့ အတက်နိုင်ဆုံး လိုရင်းတိုရင်း အကျဉ်းချုပ် ရေးသားထားတာမျိုး ဖြစ်နိုင်ပါတယ်။

SCRUM Team တစ်ခုရဲ့ ဆောင်ရွက်ရန် ကိစ္စရပ်များအားလုံးကို အဲဒီ User Story တွေ ပေါ်မှာ အခြေပြုဆောင်ရွက် ကြရပါတယ်။ Team မှာပါတဲ့ မည်သူမဆုံး User Story တွေကို Backlog ထဲ ထည့်လိုရပါတယ်။ ဒါပေမယ့် ဘယ် User Story ကို ဦးစားပေးမလဲ စီစဉ်ဆုံးဖြတ်ခြင်းကိုတော့ Product Owner ကသာ ဆုံးဖြတ်ရပါတယ်။

Backlog ထဲမှာ User Story တွေချည်းပဲရှုရမယ်လိုတော့ မဆိုလိုပါဘူး။ အမြားအချက်တွေဖြစ်တဲ့ Feature Request တွေ၊ Bug Report တွေ၊ Technical Work တွေ၊ Research Task တွေလည်း လိုအပ်ရင် ပါဝင်နိုင်ပါတယ်။ Technical Work ဆိုတာက ဥပမာ - "Server ကို Update လုပ်ရမယ်" ဆိုတဲ့ သတ်မှတ်ချက်မျိုးဖြစ်ပါတယ်။ Research Task ဆိုတာကတော့ ဥပမာ - "မြန်မာနိုင်ငံရှိမြို့များကို စာရင်းပြစ်ရမယ်" ဆိုတဲ့ သတ်မှတ်ချက်မျိုးဖြစ်ပါတယ်။ ဒါတွေဟာ Product အတွက် User Story မဟုတ်ပေမယ့် Backlog ထဲမှာ ပါဝင်နိုင်တဲ့ အကြောင်းအရာတွေဖြစ်ပါတယ်။

Sprint Planning Meeting မှာ လိုအပ်တဲ့အချက်တွေဆုံးဖြတ်ပြီးပြီဆုံးရင် Development Team က မြှော်မှန်းအချက် အတိုင်း အကောင်အထည်ဖော်နိုင်ဖို့အတွက် ဆောင်ရွက်ရမယ့် User Story တွေကို Product Backlog ထဲကနေ Sprint Backlog ထဲကို ဈေးထုတ်ပြောင်းရွေ့ရပါတယ်။ Sprint ကာလမှာ Sprint Backlog ထဲက User Story တွေကိုသာ အဓိကထား ဆောင်ရွက်သွားကြတော့မှာဖြစ်ပါတယ်။ Sprint စပြီဆုံးတာနဲ့ အခြေခံအားဖြင့် Sprint Backlog ထဲကို User Story အသစ်တွေ ထပ်မထည့်ရတော့ပါဘူး။ မဖြစ်မနေလိုအပ်လို ထည့်ရင်လည်း Product Owner နဲ့ တစ်ခြားသူတွေ မထည့်ပဲ Developer တွေကိုယ်တိုင်ကိုပဲ ထည့်ခွင့်ပြရပါတယ်။

## Task Board

Backlog နဲ့ Task တွေကိုစီမံဖို့အတွက် Task Board တွေကိုအသုံးပြုတက်ကြပါတယ်။ Task Board ဆိုတာ ခေါင်းစဉ် တပ်ပြီး Column တွေခဲ့မြားရေးဆွဲထားတဲ့ White Board တစ်ခုသာ ဖြစ်ပါတယ်။ Sprint Board လို လည်းခေါ်ပါတယ်။



ပုံ (၂၃) - Simplest Task Board

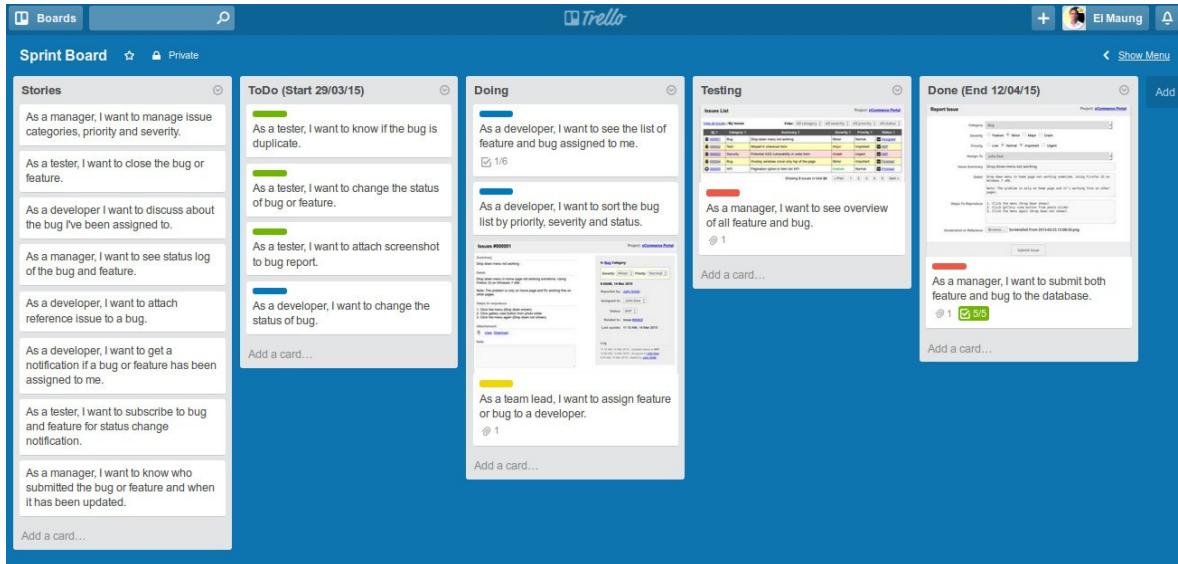
Source – Wikipedia

ပုံ (၇.၃) မှာ နမူနာကြည့်ပါ။ White Board တစ်ခုမှာ ToDo, Doing နဲ့ Done ဆိုပြီး Column (၃) ခုခဲ့ထားပါတယ်။ ဆောင်ရွက်ရန် Task တွေကို ToDo Column ထဲမှာ Sticky Note လေးတွေသုံးပြီး ကပ်ထားပါတယ်။ Sprint တစ်ခုစုံ ဆိုတာနဲ့ Sprint မှာ ပါဝင်တဲ့ User Story တွေအားလုံးကို ToDo Column မှာ စုစည်းထားရပါတယ်။ Category ခွဲတဲ့ အနေနဲ့ အမျိုးအစားမတူတဲ့ User Story တွေအတွက် အရောင်မတူတဲ့ Sticky Note လေးတွေခဲ့ပြီး အသုံးပြုကြလေ့ရှိ ပါတယ်။ User Story တစ်ခုကို စတင်ဆောင်ရွက်နေပြီဆိုရင် ဆောင်ရွက်နေတဲ့ User Story အတွက် Stick Note ကို ToDo ကနေခွာပြီး Doing မှာ ပြောင်းကပ်ပေးရပါတယ်။ ဆောင်ရွက်ပြီးစီးသွားတဲ့ User Story အတွက် Stick Note ကိုတော့ Done Column မှာ ပြောင်းကပ်ထားရပါတယ်။ ဒီနည်းနဲ့ Sprint Board ကို တစ်ချက်ကြည့်လိုက်ယုံနဲ့ ဘယ် User Story တွေပြီးနေပြီး၊ ဘယ် User Story တွေစလုပ်ပြီး User Story ဘယ်လောက် ကျော်နေသေးတယ် စတဲ့ Overview ကို သိရနိုင်ပါတယ်။

နမူနာအနေနဲ့ အရှိုးရှင်းဆုံး Board တစ်ခုကိုပေးထားတာပါ။ ပုံမှန်အားဖြင့် Sprint Board တစ်ခုမှာ Column (၅) ခုရှိနိုင်ပါတယ်။ Stories, ToDo, Doing, Testing, Done တို့ပြုဖြစ်ပါတယ်။

ပုံ (၇.၄) ကိုလေ့လာကြည့်ပါ။ Stories Column က Product Backlog အတွက်ပါ။ Product မှာ ပါဝင်ရမယ့် User Story တွေကို Stories Column မှာစုစည်းထားပါတယ်။ ToDo Column ကတော့ Sprint Backlog အတွက်ပါ။ လက်ရှိ Sprint မှာ ဆောင်ရွက်မယ့် User Story တွေကို Stories ကနေ ToDo ထဲကို ပြောင်းရွှေ့ထားရပါတယ်။ Doing ကတော့ လက်ရှိဆောင်ရွက်နေတဲ့ User Story စာရင်းဖြစ်ပြီး၊ Testing ကတော့ ဆောင်ရွက်ပြီးစီးသွားလို့ သေချာအောင် စမ်းသပ်နေတဲ့ User Story များအတွက်ဖြစ်ပါတယ်။ Testing အဆင့်မှာ စမ်းပြီးလို

ပြီးမြောက်တာ သေချာတဲ့အခါ Done Column ထဲကို ရွှေထားရမှာပဲဖြစ်ပါတယ်။



ဤ (၃၄) - A Sprint Board Example on Trello

နဲ့မှုနာမှာသုံးထားတာက Trello လိုခေါ်တဲ့ Online Tool တစ်ခုဖြစ်ပါတယ်။ Software တစ်ခုဖြစ်သွားတဲ့အတွက် White Board ထက်တော့ လုပ်ဆောင်ချက်နည်းနည်း ပိုစုံသွားပါတယ်။ User Story တွေနဲ့အတူ Screenshot တွေ၊ ဖိုင်တွေကို Attach လုပ်ထားလိုပါတယ်။ User Story တစ်ခုချင်းအတွက် Checklist (Task List) တွေတဲ့ ထားလိုပါတယ်။ Sticky Note တွေကို အရောင်ခွဲသုံးသလိုပဲ သူမှာလည်း User Story တွေကို အရောင် Label လေးတွေနဲ့ တွဲထားလို ရပါတယ်။ User Story တစ်ခုချင်းအလိုက် Comment မှတ်ချက်တွေ ထည့်သွင်းထားလို ရပါတယ်။ ဒါပေမယ့် White Board ကတော့ ပိုပြီးမြင်သာပါတယ်။ ဥပမာ - ရုံးရောက်တာနဲ့ Board ကို တစ်ချက်လမ်းကြည့်လိုက်ယုံနဲ့ အခြေအနေကို ချက်ခြင်းမှန်းဆနိုင်ပါတယ်။ သူအားသာချက်နဲ့သူပဲမို့ နှစ်သက်ရာကို သုံးနိုင်ပါတယ်။ Trello ကို စမ်းသပ်လိုရင်တော့ အောက်ပါလိုပါတယ်။

<http://trello.com/>

SCRUM နဲ့ သဘောတရားဆင်တဲ့ Kanban ဆိုတဲ့ နည်းစနစ်တစ်မျိုးလည်း ရှိပါသေးတယ်။ အခြေအားဖြင့် SCRUM က Development ပိုင်းကို Focus ထားတဲ့ Process Management နည်းစနစ်တစ်ခြားဖြစ်ပြီး Kanban ကတော့ Business Operation တစ်ခုလုံးကို စီမံနိုင် ရည်ရွယ်ထားတဲ့ Process Management နည်းစနစ်ဖြစ်ပါတယ်။ ဒါပေမယ့် Task Board ကို သုံးကြတာချင်း တူပါတယ်။ ကွာသွားတာက Kanban မှာ Development ပိုင်း ကိစ္စတွေသာမက၊ Specification ရေးသားမှုအပိုင်းတွေ၊ Data Analyst ကိစ္စတွေ၊ Deployment ကိစ္စတွေ အပါအဝင် Software Development Work-flow တစ်ခုလုံးကို Task Board သုံးပြီးစီမံတက်ပါတယ်။ ပြီးတော့ Kanban မှာ Sprint လို နည်းစနစ်မျိုးကို မသုံးပါဘူး။ အဲဒီအစား Column တစ်ခုမှာ ပါဝင်ရမယ့် Task အရေအတွက်ကို ကန့်သတ်ပြီး တစ်ခုပြီးတာနဲ့ နောက်တစ်ခုထပ်ထည့်ပြီး အဆက်မပြတ်ဆောင်ရွက်သွားတဲ့ နည်းစနစ်ကို သုံးပါတယ်။ ဥပမာ - Doing Column မှာ User Story (၄) ခုထက် ပိုထည့်ခွင့်မပြုပဲ လက်ရှိလုပ်နေ

တဲ့အလုပ်ပြီးသွားလို Doing Column မှာ User Story တွေလျော့သွားပြီဆိုမှ ထပ်ထည့်ခွင့် ပြုတဲ့စနစ်မျိုးကို သုံးပါတယ်။ Iterative Development ဆိုတာထက် Continues Development ဆိုတဲ့ သဘာဝဘက်ကို အသားပေးတဲ့သဘောဖြစ်ပါတယ်။

အခြေခံအားဖြင့် SCRUM ကို Agile Development Model လိုက်ခြေားပြီး Kanban ကိုတော့ Lean Development Model လို့ ခေါ်က်ကြပါတယ်။ SCRUM နဲ့ Kanban နှစ်ခုပေါ်ပြီး Scrumban ဆိုပြီး နည်းစနစ်သစ်တစ် မျိုးကိုလည်း တိတုင်ထားကြပါသေးတယ်။ အခြား Agile Method တစ်ခုဖြစ်တဲ့ Extreme Programming (XP) ဆိုတာလည်း ရှိပါသေးတယ်။ ဒါတွေအကြောင်းကိုတော့ ဒီနေရာမှာ မဖော်ပြနိုင်တော့ပါဘူး။ စိတ်ဝင်စားရင် ဆက်လက်လေ့လာကြည့် သင့်ပါတယ်။

#### 7.4 – Task List

ပြီးခဲ့တဲ့အခန်းမှာ Issue Tracking System တွေကို Bug Database အနေနဲ့သာမက Feature List အဖြစ်နဲ့ပါ အသုံးပြနိုင် ကြောင်းဖော်ပြခဲ့ပါတယ်။ Feature List ဆိုတဲ့သဘောက Software မှာ ဘယ်လို Features တွေပါဝင်ရမလဲ၊ ဘယ်သူတွေ တာဝန်ယူရမလဲ စတဲ့အချက်တွေကို ဖော်ပြထားပါတယ်။ Feature List ထဲက Feature တစ်ခု ကို လက်တွေ့ရေးသား အကောင်အထည်ဖော်ရာမှာ အသုံးပြုရမယ့် နည်းပညာတွေနဲ့ ဆောင်ရွက်သွားရမယ့် လုပ်ငန်းစဉ် အသေးစိတ်တော့ ပါဝင်မှာမဟုတ်ပါဘူး။ အလားတူပဲ Bug တွေနဲ့ပက်သက်ရင်လည်း ဘယ်လိုအခြေအနေ မှာတွေ့ရတဲ့ Bug လည်း၊ ဘယ်သူတာဝန်ယူဖော်ရှင်းရမှာလဲဆိုတဲ့ အချက်တွေပဲ ပါဝင်မှာပါ။ အဲဒီ Bug ကို ဖော်ရှင်းရာမှာ ဆောင်ရွက် သွားရမယ့် အသေးစိတ်အချက်အလက်တွေတော့ ပါဝင်မှာ မဟုတ်ပါဘူး။

ဒီနေရာမှာ အသုံးဝင်လာတာက Task List ဖြစ်ပါတယ်။ Task List ဆိုတာ သက်ဆိုင်ရာ Feature (သို့မဟုတ်) Bug ကို ဆောင်ရွက်ရာမှာ အသုံးပြုရမယ့် အသေးစိတ် Implementation Detail လိုလည်း ဆိုနိုင်ပါတယ်။

Task List တစ်ခုရရှိဖို့အတွက် Google Keep, Evernote, Remember The Milk စတဲ့ Task List Manager ပရိ ကရမဲ တွေသုံးနိုင်သလို ရိုးရိုး Text ဖိုင်တစ်ခုနဲ့လည်း ဖန်တီးနိုင်ပါတယ်။ ကျွန်ုတ်ကတော့ ရိုးရိုး Text ဖိုင်တစ်ခုကိုသာ အသုံးပြုဖို့ အကြိုပြုလိုပါတယ်။ Text ဖိုင်တစ်ခုနဲ့ မှတ်တမ်းတင်ထားရင် Task List ကို သက်ဆိုင်ရာ Source Code နဲ့ အတူ တဲ့ထားနိုင်မှာဖြစ်ပါတယ်။ ဒါကြောင့် လက်ရှိ Code Base ကို Pull (သို့မဟုတ်) Clone လုပ်ယူတဲ့အခါ Source Code နဲ့အတူ Task List လည်း တစ်ခါတည်း ပါဝင်သွားစေမှာပါ။ တနည်းအားဖြင့် Task List ကိုယ်တိုင်က Source Code ရဲ့ အစိတ်အပိုင်းတစ်ရှည်ဖြစ်သွားပြီး၊ အဲဒီ Task List ကိုလည်း Version Control လုပ်နိုင်စေမှာဖြစ်ပါတယ်။ Text ဖိုင်တစ်ခုနဲ့ မှတ်တမ်းတင်ထားတဲ့ နေ့နာ Task List တစ်ခုကို ဖော်ပြပေးလိုက်ပါတယ် –

## Plain Text

**Design:**

```

# TODO
[-] Two column layout for issue detail page
    Main contents such as category, subject and etc. should be in left column.
[-] Issue list table design
    Issue list table should highlight priority and severity

```

**Coding:**

```

# TODO
[-] Issue form validation @today
    category, subject, reproduction steps, severity and priority are mandatory
[-] SQL Injection filter in issue submission @due (15-03-28)
[-] Issue log in issue details

# DONE
[+] Issue list Sorting (15-03-27 23:35)

# CANCELLED
[x] Issue delete on issue detail (15-03-27 23:46)

```

**Testing:**

```

# TODO
[-] Desktop cross browser test
    We will only support IE 9 and above.

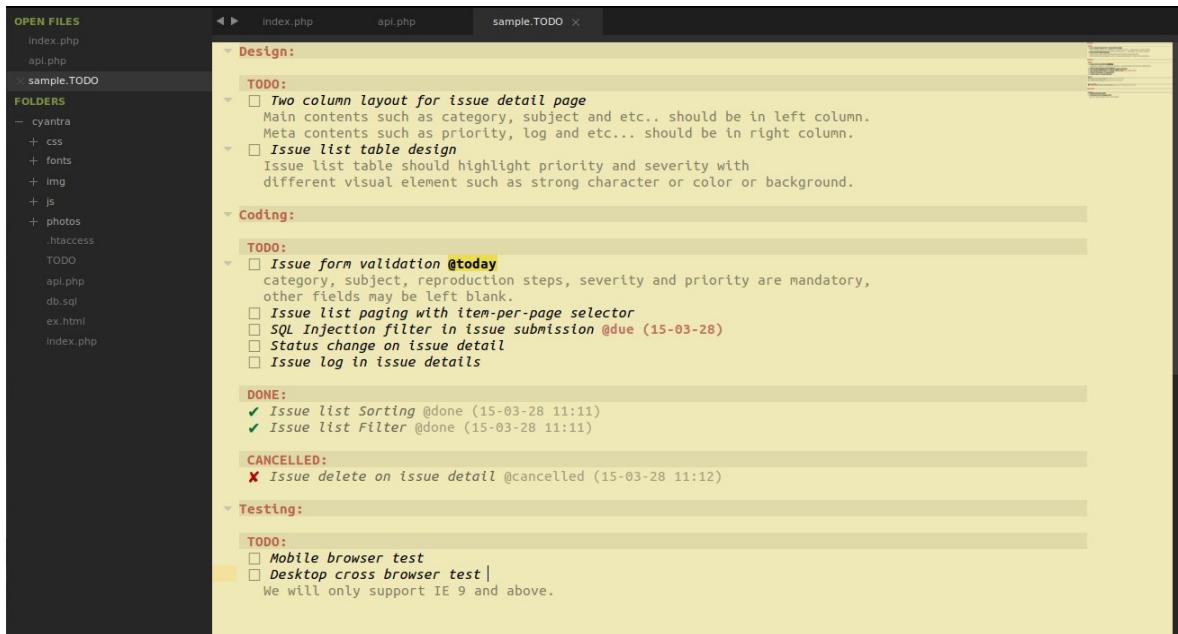
```

Task List တစ်ခုမှာ Task တွေကိုအမျိုးအစားခွဲထားသင့်ပါတယ်။ နမူနာကိုလေ့လာကြည့်ရင် Design, Coding, Testing ဆိုပြီး အမျိုးအစား Category ခွဲထားတာကို တွေ့ရနိုင်ပါတယ်။ နောက်ထပ် ရှိသေးရင်ထပ်ခွဲထားနိုင်ပါတယ်။

သက်ဆိုင်ရာ Category တစ်ခုစီမှာ TODO, DONE နဲ့ CANCELLED ဆိုပြီး အပ်စ (၃) စ ပါဝင်သင့်ပါတယ်။ TODO ကတော့ ဆောင်ရွက်ရန် Task များဖြစ်ပါတယ်။ DONE ကတော့ ဆောင်ရွက်ပြီး Task များဖြစ်ပါတယ်။ CANCELLED ကတော့၊ မူလကဆောင်ရွက်ရန် သတ်မှတ်ခဲ့ပေမယ့် ဆက်လက်မဆောင်ရွက်တော့ဖို့ ဆုံးဖြတ်လိုက်တဲ့ Task များပဲဖြစ်ပါတယ်။ TODO ထဲက Task တစ်ခုကို ဆောင်ရွက်ပြီးစီးသွားတဲ့အခါ DONE ထဲကို ရွှေထားပေးဖို့ လိုပါတယ်။ အဲဒီလို ရွှေလိုက်တဲ့အခါ (နမူနာမှာပြထားသလို) ရွှေလိုက်တဲ့နေ့ရက်ကို တွဲဖက်မှတ်တမ်းတင်ထားသင့်ပါတယ်။ ဒီတော့မှာ ဘယ့်နဲ့ က ဘယ် Task ပြီးသွားတယ်ဆိုတဲ့ မှတ်တမ်းကို ရရှိမှာဖြစ်ပါတယ်။ အလားတဲ့ Task တစ်ခုကို ဆက်လက်မဆောင် ရွက်ဖို့ ဆုံးဖြတ်တယ်ဆိုရင်လည်း CANCELLED အောက်ကို ရွှေပြီး ရက်စွဲနဲ့အတူ မှတ်တမ်းတင်ထားသင့်ပါတယ်။

နမူနာမှာလေ့လာကြည့်ရင် ကနွေအပြီးလုပ်ရမယ့် Task တစ်ခုကို **@today** ဆိုတဲ့ Tag တစ်ခုနဲ့ တွဲဖက်မှတ်တမ်းတင်ထားတာကို တွေ့ရနိုင်ပါတယ်။ သတ်မှတ်ရက်တစ်ခုမှာ အပြီးဆောင်ရွက်ရမယ့် Task တွေကိုလည်း **@due** ဆိုတဲ့ Tag နဲ့ တွဲဖက်မှတ် တမ်းတင်ထားတာကို တွေ့ရမှာပဲဖြစ်ပါတယ်။ ဒါအပြင် အသေးစိတ်မှတ်ချက်ထည့်သွင်းဖို့လိုတဲ့ Task တွေအတွက် အသေးစိတ် မှတ်ချက်တွေကိုလည်း တွဲဖက်ထားပါသေးတယ်။

နဲ့မှနာမှာဖော်ပြထားသလို စာလုံးအရောင်တွေ ခွဲခြားဖော်ပြစွဲဖော်အတွက် အသုံးပြုနေတဲ့ Text Editor (သို့မဟုတ်) IDE ရဲ့ Syntax Highlight လုပ်ဆောင်ချက်ကို အသုံးချိန်ပါတယ်။ ဒါမှမဟုတ် Text Editor နဲ့အတူတွဲဖက်ပါ ဝင်လာတဲ့ Task Management လုပ်ဆောင်ချက်ကိုလည်း အသုံးပြုနိုင်ပါတယ်။



ပုံ (၇.၅) - PlainTasks - Task List Manager in Sublime Text 3

ပုံ (၇.၅) မှာ PlainTasks လိုက်တဲ့ Sublime Text 3 Package တစ်ခုကို နဲ့မှနာအနေနဲ့ ဖော်ပြထားပါတယ်။ PlainTasks က Text ဖိုင်တစ်ခုအနေနဲ့ သိမ်းဆည်းထားတဲ့ Task List ကို Icon တွေ အရောင်တွေနဲ့ Format လုပ်ပြီး သပ်ရပ်သွားအောင်ဖော်ပြပေးနိုင်ပါတယ်။ ဒါအပြင် Task List ကို Manage လုပ်နိုင်တဲ့ Feature တွေလဲ ပါဝင်ပါသေး တယ်။ ဥပမာ - Task တစ်ခုကိုရွေးပြီး Keyboard Shortcut Ctrl + D ကို နှိပ်ခြင်းအား ဖြင့် ဆောင်ချက်ပြီးစီးကြောင်း သတ်မှတ်နိုင်ပါတယ်။ စမ်းသပ် အသုံးပြုလိုရင် အောက်ပါလိပ်စာများ ရယူနိုင်ပါ တယ်။

<https://github.com/aziz/PlainTasks>

Code Comment ထဲမှာ ထည့်သွင်းမှတ်သားခဲ့တဲ့ TODO တွေကို ခွဲထုတ်ပြီး Task List အဖြစ် ဖော်ပြပေးနိုင်တဲ့ လုပ်ဆောင်ချက်တွေလည်း Text Editor နဲ့ IDE တွေမှာ ပါဝင်တာက်ပါတယ်။ ပုံ (၇.၆) မှာတော့ TodoReview လို ခေါ်တဲ့ Source Code ထဲမှာ ထည့်သွင်းရေးသားထားတဲ့ Task တွေကို ခွဲထုတ်ပြီး Task List တစ်ခုအနေနဲ့ Generate လုပ်ပေးနိုင်တဲ့ Sublime Text 3 Package တစ်ခုကို နဲ့မှနာအနေနဲ့ ဖော်ပြပေးထားပါတယ်။

ပုံ (၇.၆) - TodoReview - Task List Extractor in Sublime Text 3

Source Code ထဲမှာ TODO: NOTE: FIXME: CHANGE: စတဲ့ Keyword တွေခံပြီး Comment တွေထည့်သွင်းမှတ်သားထားခဲ့မယ့်ဆိုရင် TodoReview က Code Base တစ်ခုလုံးကို Scan လုပ်ပြီး အဲဒီ Task List တစ်ခုအနေနဲ့ စုစုပေါင်းဖြတ်ပြန်ပါတယ်။ စမ်းသပ်အသုံးပြုလိုရင် အောက်ပါလိပ်စာမှာ ရယူနိုင်ပါတယ်။

<https://github.com/jonathandelgado/SublimeTodoReview>

အကယ်၍ စာဖတ်သူက Sublime Text 3 မသုံးပဲ အခြား Text Editor (သို့မဟုတ်) IDE တစ်ခုခုကို သုံးတယ်ဆိုရင် လည်း အလားတူလုပ်ဆောင်ချက်မျိုးရရှိနိုင်မယ့် နည်းလမ်းတွေကိုလေ့လာအသုံးပြုသင့်ပါတယ်။ တစ်ချို့ IDE တွေမှာ Build-in တစ်ခါတည်း ပါဝင်လာတက်ပေမယ့် တစ်ချို့ IDE တွေမှာတော့ Extension (သို့မဟုတ်) Plugin (သို့မဟုတ်) Package အနေနဲ့ ထပ်မံထည့်သွင်းရနိုင်ပါတယ်။

## 7.5 – Code Review

Team Collaboration နည်းစနစ်တွေထဲမှာ နောက်ဆုံးအနေနဲ့ ဖော်ပြချင်တဲ့ အချက်ကတော့ Code Review ပဲဖြစ်ပါတယ်။ Developer တွေအနေနဲ့ Software Project တွေမှာမြင်ရတဲ့ Functional Feature တွေကို အရင်လုပ်ပြီး နောက်တော့မှ Security, Performance စတဲ့ Non-function တွေကို လိုက်လုပ်တက်ကြလေ့ရှိပါတယ်။ ပြဿနာတစ်ခုရဲ့ အဖြေရရှိရေးကိုသာ အလေးထားတက်ကြတဲ့သဘာဝရှိလို့ အဖြေတစ်ခုထွက်ရေးကိုသာ ဦးစေးပေးတက်ကြပြီး ရေရှည်လုပ်ခြင်း ရေးနဲ့ စွမ်းဆောင်ရည်ကိစ္စတွေကို နောက်မှ လိုက်လုပ်တက်ကြတာပါ။

အဲဒီလိုအလေ့အထာကြား Software Project ကတော့ ပြီးသားပေမယ့် Code ရဲ့ Performance က မကောင်းတာတို့ လုပြုရေးအားနည်းချက်တွေ ရှိနေတာတို့ ဖြစ်တက်ပါတယ်။ Security နဲ့ Performance ဆိုတာ တစ်ကယ့်တော့ စ ကတည်းက စနစ်တစ်ကျိုး ဆောင်ရွက်ရတဲ့လုပ်ငန်းတွေဖြစ်ပါတယ်။ အဲဒီလုပ်ငန်းတွေကို နောင်မှ လိုက်လုပ်တဲ့အခါ စနစ် မကျတာ၊ မပြည့်စုံတော့ပဲကျန်ခဲ့တာတွေ ဖြစ်လာတက်ပါတယ်။ ဒီပြဿနာကို လျှော့ချို့အတွက် Code Review ဆိုတဲ့ နည်းစနစ်ကို သုံးကြပါတယ်။

Code Review ဆိတာ လိုဂ်းကတော့ Developer တစ်ဦးရေးထားတဲ့ Code ကို အခြား Developer တွေက ပိုင်းဝန်း စီစစ်ကြခြင်းဖြစ်ပါတယ်။ တွေ့ရလေ့ရှိတဲ့ အမှားတွေရှိနေသလား၊ မသုံးသင့်တဲ့ Deprecated Syntax တွေသုံးထားသလား၊ Database Connection လို Resource တွေကို သုံးပြီးတဲ့အခါ ပြန်ပိတ်ရဲလား၊ Code Structure ကောင်းမွန်ရဲလား စသဖြင့် ပိုင်းဝန်း စီစစ်ကြခြင်း ဖြစ်ပါတယ်။ တစ်ချို့ပြဿနာတွေကို ဘေးလူကပို မြင်တက်တဲ့သဘာဝရှိလို မူလ Developer သတိမထားမိခဲ့တဲ့ အမှားတွေကို Code Review မှာ ရှာဖွေတွေရှိပြီး ပြင်လိုက်နိုင်မယ်ဆိုရင် Code ရဲ့ အရည်အသွေးဟာ လည်း တိုးတက်လာမှာပဲဖြစ်ပါတယ်။

Code Quality အပြင် ဖြည့်စွက်ရရှိမယ့် အကျိုးကျေးဇူးကတော့ Knowledge Transfer ဖြစ်ပါတယ်။ Developer တစ်ဦးရဲ့ အလေ့အကျင့် ကောင်းတွေကို အခြား Developer တွေက Code Review Session တွေမှာ အပြန်အလှန် နမူနာယူလေ့လာခွင့် ရကြမှာဖြစ်ပါတယ်။ လူညီဖို့တွေ့လိုပါတယ်။ လူမညီရင်တော့ Code Quality ကောင်းစို့ဆိုတဲ့ ရည်ရွယ်ချက်နဲ့လုပ်တဲ့ Code Review ကနေ၊ ကြည့်မရတဲ့ Developer ရဲ့ Code ကို တရားဝင်ပြစ်တင်စေဖန်ခွင့်ပေးလိုက်တဲ့ ရန်တိုက်ပေးတဲ့မွဲဖြစ်သွားနိုင်ပါတယ်။ လိုရင်းဖြစ်တဲ့ Code Quality ကောင်းမွန်တိုးတက်ရေး ဆိုတဲ့ Objective ကို မူတည်ဆောင်ရွက်နိုင်ဖို့လိုပါတယ်။

Code Review နည်းလမ်း (၃) မျိုး ရှိပါတယ်။ Formal Review, Light Review နဲ့ Pair Programming တို့ပဲဖြစ်ပါတယ်။ Formal Review ဆိတာကတော့ အစည်းအဝေးတွေခေါ် Code တွေကို Print ထုတ်၊ စာရွက်ပေါ်မှာ တစ်ကြောင်းချင်း တစ်လိုင်းချင်းဖတ်ပြီး ပိုင်းပြီးစီစစ်ကြခြင်း ဖြစ်ပါတယ်။ Fagan Inspection လိုခေါ်တဲ့ Formal Code Review လုပ်ရာမှာ ကြိုတ်ပြင်ဆင်ပုံတွေ၊ Meeting ခေါ်ပေးတွေနဲ့ Review လုပ်ပုံတွေကို သတ်မှတ်ထားတဲ့ Code Review Procedure တစ်ခုရှိပါတယ်။ Light Review ဆိတာကတော့ အဲဒီလောက် တင်းတင်းကြပ်ကြပ် မဟုတ်ပဲ အလျင်း သင့်သလို တစ်ဦးရေးထားတဲ့ Code ကို နောက်တစ်ဦးက စီစစ်ပေးခြင်း ဖြစ်ပါတယ်။ Light Review အနေနဲ့ Code Review Tool တွေကိုလည်း သုံးတက်ကြပါတယ်။ Code Review Tool ဆိတာ လူတစ်ယောက်က Code ဖိုင်ကို ဖွင့်ပတ်စီစစ်ခြင်းမဟုတ်ပဲ ကွန်ပျူးတာပရိုကရမ်တစ်ခုနဲ့ တွေ့ရလေ့ရှိတဲ့ အမှားတွေကို အလိုအလျောက်စီစစ်ခြင်း ဖြစ်ပါတယ်။ Static Code Analysis လိုလည်းခေါ်ပါတယ်။ JavaScript အတွက်ဆိုရင် JSLint, JSHint စတဲ့ Code Quality Tool တွေရှိသလို Closure Compiler လို Code Analysis Tool တွေ လည်း ရှိပါတယ်။

## Pair Programming

Pair Programming ဆိတာ အခြား Agile Development နည်းစနစ်တစ်ခုဖြစ်တဲ့ Extreme Programming ကနေ လာတဲ့ နည်းစနစ်ဖြစ်ပါတယ်။ ကွန်ပျူးတာတစ်လုံးကို Programmer နှစ်ဦးက တစ်ပြိုင်တည်းသုံးခြင်းအားဖြင့် Code ရေးတဲ့ နည်းစနစ် ဖြစ်ပါတယ်။ တစ်ယောက်က ထိုင်ရေးပြီး၊ နောက်တစ်ယောက်က Review လုပ်တဲ့ သဘောမျိုးနဲ့ ထိုင် ကြည့်ရပါတယ်။ Code ရေးသူကို Driver လိုခေါ်ပြီး၊ ဘေးက Review လုပ်သူကို Navigator လိုခေါ်ပါတယ်။

ရေးတဲ့သူက သတ်မှတ်လုပ်ဆောင်ချက် ရရှိအောင် ရေးသားနေချိန်မှာ၊ Navigator က နောက်ဆက်လုပ်ရမယ့် အဆင့် အတွက် ကြိုတ်စဉ်စဉ်းစားထားနိုင်ပါတယ်၊ လက်ရှိရေးနေတဲ့အပိုင်းကို ပိုကောင်းသွားအောင်လုပ်လိုရမယ့် နည်းလမ်းတွေကို စဉ်းစားနိုင်ပါတယ်။ ဒီနည်းနဲ့ Code ရေးနေတဲ့ Driver က ရေးနေတာကို ရပ်ပြီး နောက်အဆင့်အတွက် စဉ်းစားရ တာတို့၊ သေချာရဲလား တစ်ခေါက်ပြန်စစ်ရတာတို့ လုပ်စရာမလိုတော့ပဲ

လက်ရှိ Code ကို အပြည့်အဝ အာရုံစိုက်နိုင်သွား ပါတယ်။ မှားစရာရှိမယ်ဆိုရင်လည်း မမှားခင်ကတည်းက Navigator က တွေ့ရှိထောက်ပြနိုင်တဲ့အတွက် စမ်းကြည့်ပြီးမှား နေလို့ နောက်တစ်ခေါက် ပြန်ပြင်လိုက်ရတယ် ဆိုတဲ့ အလုပ်အတွက် အချိန်ကုန်စရာမလိုတော့ပါဘူး။ Code Quality တိုးတက်သွားတဲ့အပြင် ပိုပြီးတော့လည်း အလုပ်တွင်သွားပါတယ်။ ကွန်ပျူတာနှစ်လုံးနဲ့ နှစ်ယောက်ခဲ့ ပြီးရေးတာထက် Pair Programming နည်းလမ်း က ရေရှည်မှာ အလုပ်ပိတ်တက်ပါတယ်။

Pair Programming ဟာ အလေ့အကျင့်ကောင်းတစ်ခြဖစ်ပေမယ့် လက်တွေ့အကောင်အထည်ဖော်ဖို့တော့ နည်းနည်း ခက် ပါတယ်။ တဲ့ဖက်ရေးသားမယ့် Developer နှစ်ဦးရဲ့ အရည်အချင်း Level က ညီနှိုင်လိုနိုင်ပါတယ်။ မ ညီရင်မရဘူးတော့ မဟုတ်ပါဘူး။ မညီရင် ပိုတော်တဲ့သူက Driver လုပ်ရမလဲ၊ နောက်တစ်ယောက်က Driver လုပ်ရမလား၊ ဘယ်ဟာ ပို အကျိုးရှိမလဲဆိုတာမျိုးတွေ စဉ်းစားရပါတော့မယ်။ နောက်ပြီးတော့ Developer ဆိုတာမျိုးက တစ်ကိုယ်တော်သမားတွေ ရှိတာက်ပါတယ်။ တစ်ယောက်ထဲ ခေါင်းအေးနဲ့လုပ်ရမှု အာရုံပိုရတယ်ဆိုတာမျိုးလည်း ရှိနိုင်ပါတယ်။ အဲလိုလူမျိုးကို Pair လုပ်ပေးလိုက်ရင် သူအတွက်သက်တောင့်သက်သာ မဖြစ်တက်အတွက် ပိုပြီးအလုပ်မတွင် ဖြစ်သွားတက်ပါသေး တယ်။ အခြေအနေနဲ့ ချိန်ညီပြီး သင်တော်ရင်အသုံးပြုကြဖို့ဖြစ်ပါတယ်။

## Conclusion

ဖော်ပြခဲ့တဲ့ Collaboration နည်းစနစ်တွေကို တစ်ခုမကျန် အကုန်သုံးဖို့တော့ မဟုတ်ပါဘူး။ ကိုယ့် Team ရဲ့ အနေ အထားနဲ့ သင့်တော်သလို ချိန်ညီအသုံးပြုကြရမှာဖြစ်ပါတယ်။ ဥပမာ – နေရာအတည်တစ်ကျမရှိပဲ အင်တာနက်အကူအညီ နဲ့ အလုပ်လုပ်ကြတဲ့ Remote Team တွေမှာဆိုရင် SCRUM တို့ Pair Programming တို့ကို အသုံးချိန်ဖို့ မလွယ် ပါဘူး။ Issue Tracking System ကိုသာ အားကိုးပြီးလုပ်ကြရမှာပါ။ သိပ်ကြီးတဲ့ အဖွဲ့အစည်းတိုးတွေမှာဆိုရင်တော့ ဖော်ပြခဲ့တဲ့ နည်းစနစ်အားလုံးကို တစ်နေရာထဲမှာ စုပေါင်းပေးထားတဲ့ Project Management Software တွေကို အသုံးပြုရနိုင်ပါတယ်။ တစ်ဦးနှစ်ဦးပဲပါတဲ့ Team လေးတွေမှာဆိုရင်တော့ အခြားနည်းစနစ်တွေမလိုပဲ Issue Tracking နဲ့ Task List လောက်နဲ့တင် လုံလောက်နိုင်ပါတယ်။

အဆုံးသတ်အနေနဲ့ ဒီအခန်းရဲ့အစမှာ ဖော်ပြခဲ့တဲ့ Agile Team တစ်ခုရဲ့ လုပ်ငန်းစဉ်ကို နောက်တစ်ခေါက်ပြန်လည်ဖော်ပြ လိုက်ပါတယ်။

1. Software Project တစ်ခု စတင်တော့မယ်ဆိုတာနဲ့ Project Manager (သို့မဟုတ်) Product Owner က **Functional Specification** တစ်ခုကို စတင်ရေးသားပါတယ်။
2. Specification ပေါ်အခြေခံပြီး **Project Schedule** ကိုရေးဆွဲထားရပါတယ်။
3. Specification မှာပါဝင်တဲ့ **User Story** တွေကိုထုတ်ယူပြီး **Product Backlog** ဖန်တီးပါတယ်။
4. ပြီးတဲ့အခါ Iteration တစ်ခုစတင်နိုင်ဖို့အတွက် **Sprint Planning Meeting** ခေါ်ပြီး **Sprint Backlog** ကို ဖန်တီးပါတယ်။
5. Development Team က Sprint Backlog မှာပါဝင်တဲ့ User Story တွေပေါ်မှာအခြေခံပြီး Implementation Detail အတွက် **Task List** ကို တည်ဆောက်ပါတယ်။
6. Sprint Backlog နဲ့ Task List ကို အခြေခံပြီး လိုအပ်တဲ့ Code တွေကို စတင်ရေးသားပါတယ်။ ဒီလိုရေးသားရာမှာ Code Quality ကောင်းစေဖို့အတွက် Code Review နည်းစနစ်တစ်ခုဖြစ်တဲ့ Pair Programming ကို အသုံးပြုပါတယ်။
7. တည်ဆောက်ရရှိလာတဲ့ ရလဒ်မှာတွေ့ဗျာတဲ့ Bug နဲ့ Feature Request တွေကို **Issue Tracking System** ထဲမှာ မှတ်တမ်းတင်ပါတယ်။
8. Project Manager (သို့မဟုတ်) Product Owner က Issue Tracking System ထဲမှာ ရှိနေတဲ့ Bug နဲ့ Feature Request တွေရဲ့အခြေအနေနဲ့ချိန်ဆပြီး Specification နဲ့ Schedule ကို Update လုပ်ပါတယ်။
9. **Up-to-date Schedule** နဲ့ချိန်ညွှန်ပြီး Product Backlog ကို ပြန်လည်စီစဉ်ပါတယ်။
10. ပြီးတဲ့အခါ အမှတ်စဉ် (၄) ကို ပြန်သွားပြီး နောက် Iteration တစ်ခုကို ပြန်လည်စတင်ပါတယ်။

Software Project စီမံရာမှာ အရေးပါတဲ့ နောက်ထပ်နည်းစနစ်တွေကိုတော့ Development Workflow Automation ခေါင်းစဉ်နဲ့ နောက်တစ်ခန်းမှာ ဆက်လက်ဖော်ပြသွားပါမယ်။

Automate လုပ်ထားလို့ရတဲ့နိုင်တဲ့ကိစ္စတစ်ခုကို ထပ်ခါထပ်ခါ  
Manual လုပ်နေရင် အချိန်တွေ၊ လုပ်အားတွေ ဖြန်းတီးတာပါပဲ။  
ထိရောက်အလုပ်တွင်မှာလည်းမဟုတ်ပါဘူး။

### **Professional Web Developer (စာအုပ်)**

Web Standard, jQuery, PHP, MySQL, Ajax, CMS, MVC,  
HTML5, Mobile Web, Web Application Security စသည်  
အကြောင်းအရာများကို ရေးသားဖော်ပြထားသည့်စာအုပ်  
အောက်ပါလိပ်စာတွင် အခမဲ့ Download ရယူနိုင်သည်။

<http://pwdbook.com>

## အခန်း(၈) – Development Workflow Automation

Software Project တစ်ခုမှာ Code ရေးသားခြင်းနဲ့အတူ Developer တွေ ပုံမှန်တွဲဖက်လုပ်ကြရလေ့ရှိတဲ့ လုပ်ငန်း (၄) ခုရှိ ပါတယ်။ အဲဒါတွေကတော့ -

1. Scaffolding
2. Package Management
3. Build နဲ့
4. Deployment – တို့ပဲဖြစ်ပါတယ်။

### 8.1 – Scaffolding

Software Project တစ်ခုကို စတင်ရေးသားတော့မယ်ဆိုတာနဲ့ ပထမဆုံးလုပ်ရတဲ့အလုပ်က၊ လိုအပ်တဲ့ Directory Structure နဲ့ Source Code ဖိုင်တွေ တည်ဆောက်ရခြင်းပဲ ဖြစ်ပါတယ်။ လုပ်ဆောင်ချက် ထွေထွေထူးထူးမပါတဲ့ ရှိုးရှိုး Web Page လေးတစ်ခု ဖန်တီးတော့မယ်ဆိုရင်တောင် အခုလို့ Structure တော့ အနည်းဆုံး လိုအပ်မှာပဲဖြစ်ပါတယ်။

```

WebPage
├── css
│   ├── responsive.css
│   └── style.css
├── img
│   └── logo.png
└── js
    ├── app.js
    └── jquery.js
index.html

```

Directory Structure တည်ဆောက်ပြီးတဲ့အခါ အခြေခံ Code Structure ကို ဆက်လက်တည်ဆောက်ရပါတယ်။ HTML Document တိုင်းမှာ အနည်းဆုံး အခုလို့ Structure တော့ ပါဝင်ရလေ့ရှိပါတယ်။

#### HTML

```

1. <!DOCTYPE html>
2. <html lang="en">
3. <head>
4.   <meta charset="UTF-8">
5.   <title>Document Title</title>
6.   <link rel="stylesheet" href="css/style.css">
7. </head>

```

```

8. <body>
9.
10. <script src="js/jquery.js"></script>
11. <script src="js/app.js"></script>
12. </body>
13. </html>

```

အလားတူပဲ CSS ဖိုင်တစ်ခုမှာလည်း အခုလို Normalize Style က အမြပါရတက်ပါတယ်။

## CSS

```

1. html, body {
2.   margin: 0;
3.   padding: 0;
4. }
5.
6. article, aside, details, figcaption, figure, footer,
7. header, hgroup, main, menu, nav, section, summary {
8.   display: block;
9. }
10.
11. a:active, a:hover {
12.   outline: 0;
13. }
14.
15. img {
16.   border: 0;
17. }
18.
19. table {
20.   border-collapse: collapse;
21.   border-spacing: 0;
22. }

```

ဒီလို အမြပါရတဲ့ အခြေခံ Directory Structure နဲ့ Code Structure တည်ဆောက်ခြင်းလုပ်ငန်းကို Scaffolding လို့ ခေါ်ပါတယ်။ Project အသစ်တစ်ခုစာတိုင်း Scaffolding လုပ်ငန်းကို တစ်ခုချင်း ကိုယ်တိုင်လိုက်လုပ်နေရရင် အချိန်တွေ ကုန်ပါတယ်။ ဒီလိုအချိန်မကုန်ဖေဖို့ဆိုရင် Boilerplate Code တွေကို သုံးနိုင်ပါတယ်။

Boilerplate Code ဆိုတာ အခြေခံအားဖြင့်ပါဝင်လေ့ရှိတဲ့ Code တွေကို ပြောတင်ရေးသားထဲ ၁ Source Code တစ်မျိုးဖြစ်ပါတယ်။ Project တစ်ခုစာတော့မယ်ဆိုရင် Directory Structure နဲ့ အခြေခံ Code Structure ကို ကိုယ်တိုင် ရေးနေမယ့်အစား Boilerplate Code ကို အသင့်ယူအသုံးပြုလိုက်နိုင်တဲ့သော ဖြစ်ပါတယ်။ Web Development အတွက် ထင်ရှားထဲ Boilerplate နှစ်ခုရှိပါတယ်။ HTML5 Boilerplate နဲ့ Google ကပေးထားတဲ့ Web Starter Kit တို့ပဲဖြစ်ပါတယ်။

HTML5 Boilerplate ကို [html5boilerplate.com](http://html5boilerplate.com) မှာ Download ရယူနိုင်ပါတယ်။ Download ရယူထားတဲ့ Archive ဖိုင်ကို Extract လုပ်ကြည့်လိုက်ရင် အခုလို Directory Structure ကို တွေ့ရမှာဖြစ်ပါတယ်။

```

HTML5-boilerplate
└── css
    ├── main.css
    └── normalize.css
└── img
└── js
    ├── vendor
    │   ├── jquery-1.11.2.min.js
    │   └── modernizr-2.8.3.min.js
    ├── main.js
    └── plugins.js
└── .htaccess
└── .gitignore
└── 404.html
└── favicon.ico
└── index.html

```

Web App Project တစ်ခုအတွက် အခြေခံအားဖြင့် လိုအပ်တဲ့ Directory Structure နဲ့ Code ဖိုင်တွေကို အသင့် တည်ဆောက်ထားပေးခြင်းဖြစ်ပါတယ်။ Web App Project တိုင်းမှာ လိုအပ်လေ့ရှိတဲ့ normalize.css, jQuery, Modernizr တို့လို Library တွေသာမက .htaccess, .gitignore စတဲ့ Setting ဖိုင်တွေကိုပါ တစ်ခါတည်း ထည့်သွင်းပေးထားပါတယ်။ index.html ကို ဖွင့်ပြီးလေ့လာကြည့်ရင် အခုလိုတွေရမှာ ဖြစ်ပါတယ်။

## HTML

```

1. <!doctype html>
2. <html class="no-js" lang="">
3.   <head>
4.     <meta charset="utf-8">
5.     <meta http-equiv="X-UA-Compatible" content="IE=edge">
6.     <title></title>
7.     <meta name="viewport" content="width=device-width, initial-scale=1">
8.
9.     <link rel="stylesheet" href="css/normalize.css">
10.    <link rel="stylesheet" href="css/main.css">
11.    <script src="js/vendor/modernizr-2.8.3.min.js"></script>
12.  </head>
13.  <body>
14.    <!--[if lt IE 8]>
15.      <p class="browserupgrade">You are using an <strong>outdated</strong>
16.      browser. Please <a href="http://browsehappy.com/">upgrade your
17.      browser</a> to improve your experience.</p>
18.    <![endif]-->
19.
20.    <p>Hello world! This is HTML5 Boilerplate.</p>
21.
22.    <script src="//ajax.googleapis.com/ajax/libs/jquery.min.js"></script>
23.    <script>window.jQuery || document.write('<script
24.              src="js/vendor/jquery-1.11.2.min.js"></script>')
25.    <script src="js/plugins.js"></script>
26.    <script src="js/main.js"></script>
27.
28.    <script>
29.      (function(b,o,i,l,e,r){b.GoogleAnalyticsObject=l;b[l]|| (b[l]=
30.        function(){(b[l].q=b[l].q||[]).push(arguments)});b[l].l+=new Date();
31.      })(window,window.GoogleAnalyticsObject='ga');
32.    </script>
33.
34.    <script src="js/main.js"></script>
35.
36.  </body>
37. </html>

```

```

27.     e=o.createElement(i);r=o.getElementsByTagName(i)[0];
28.     e.src='//www.google-analytics.com/analytics.js';
29.     r.parentNode.insertBefore(e,r)}(window,document,'script','ga'));
30.     ga('create','UA-XXXXXX-X','auto');ga('send','pageview');
31.   </script>
32. </body>
33. </html>

```

Web App တစ်ခုအတွက် အခြေခံ Code Structure ကို အသင့်ရေးသားပေးခြင်း ဖြစ်ပါတယ်။ ရေးသားတဲ့ Code ကိုလေလာကြည့်ရင် လိုင်းနံပါတ် (၇) မှာ Mobile Device တွေမှာ အဆင်ပြေပြေဖော်ပြန်ဖော်အတွက် လိုအပ်တဲ့ Viewport Meta Element ကို အသင့်ထည့်ထားပေးတာကို တွေ့ရမှာဖြစ်ပါတယ်။ လိုင်းနံပါတ် (၁၄) မှာ Internet Explorer 8 နဲ့ သူရှေ့ပိုင်း Browser အဟောင်းတွေကနေ အသုံးပြုရင် Browser ကို Upgrade လုပ်ဖို့ သတိပေးတဲ့ Message ကိုလည်း တစ်ခါတည်း ထည့်သွင်းပေးထားပါတယ်။ လိုင်းနံပါတ် (၂၀) နဲ့ (၂၁) ကို လေလာကြည့်ရင် jQuery Library ကိုချိတ်ဆက်ရာမှာ Google CDN က jQuery ကို အရင်ချိတ်ဆက်ပြီး မရတော့ မှ Local jQuery ကို ချိတ်ဆက်တဲ့ လုပ်ဆောင်ချက်ကို ရေးသားထားပေးပါတယ်။ လိုင်းနံပါတ် (၂၅) ကနေ (၃၁) ထိမှာ Google Analytic Tracking Code ကိုလည်း တစ်ခါတည်း ထည့်သွင်းထားပေးပါသေးတယ်။ Web App Project တစ်ခုမှာ လိုအပ်တဲ့ အခြေခံ Code ကို တစ်ခါတည်း အသင့်ရေးသားပေးခြင်း ဖြစ်ပါတယ်။

ဒါ index.html တစ်ခုပဲကြည့်ရပါသေးတယ်။ .htaccess ဖိုင်ထဲမှာဆိုရင်လည်း Application Performance နဲ့ လုပ်ချောင်းစွဲအသုံးပြုရတဲ့ Apache Setting တွေ အသင့်ရေးသားထားပေးပါတယ်။ plugin.js ထဲမှာ လည်း Console လုပ်ဆောင်ချက်မပါဝင်တဲ့ Browser တွေမှာ console.log သုံးပြီး Test Code တွေထည့်ရေးမိတဲ့ အခါ Error မတက်အောင် ဖြေရှင်းပေးတဲ့ Code ကို အသင့်ရေးသားထားပေးပါတယ်။ Boilerplate ဆိုတာ ဒါပါပဲ။ အခြေခံ လိုအပ်ချက်ဖြစ်တဲ့ Directory Structure နဲ့ Source Code တွေကို အသင့်ကြိုးတင်ရေးသားခြင်း ပဲဖြစ်ပါတယ်။ ဒါကြောင့် နောင် Web App Project တစ်ခု စတော့မယ်ဆိုတိုင်း Directory Structure တစ်ခုကို ကိုယ်တိုင်ဆောက် မနေပဲ HTML5 Boilerplate ကို Download ရယူခြင်းအားဖြင့် အသင့်ရရှိနိုင်မှာပဲဖြစ်ပါတယ်။

Google Web Starter Kit ကိုတော့ [developers.google.com/web/starter-kit/](https://developers.google.com/web/starter-kit/) မှာ Download ရယူနိုင်ပါတယ်။ ရှုံးလာတဲ့ Archive ဖိုင်ကို Extract လုပ်ကြည့်လိုက်ရင် အခုလုံတွေ့ရမှာဖြစ်ပါတယ်။

```

web-starter-kit
└── app
    ├── fonts
    ├── images
    ├── scripts
    │   └── main.js
    ├── styles
    │   ├── components
    │   │   └── h5bp.scss
    │   └── main.scss
    ├── basic.html
    ├── favicon.ico
    ├── index.html
    ├── manifest.json
    └── manifest.webapp

```

```

└── gulpfile.js
  └── package.json

```

အမှန်တော့ Web Starter Kit က HTML5 Boilerplate ကိုပဲ Mobile Device တွေနဲ့အဆင်ပြေဖော်အတွက် Responsive Design လုပ်ဆောင်ချက် ထပ်မံဖြည့်စွက်ပေးထားခြင်းဖြစ်ပါတယ်။ ဒုအပြင် Performance ကောင်းစေဖို့ အတွက် Google Page Speed လုပ်ဆောင်ချက်၊ ရေးသားထားတဲ့ App ကို စမ်းသပ်ရာမှာ ပိုမို အဆင်ပြေဖော်အတွက် Build-in Server နဲ့ အသင့်သုံးနိုင်တဲ့ CSS Component တွေဖြည့်စွက်ပေးထားပါသေးတယ်။ ပိုမိုအခြေခံကျတဲ့ Web App တစ်ခုတည်ဆောက်လိုရင် HTML5 Boilerplate မူရင်းကို သုံးသင့်ပြီး Mobile Device တွေမှာ အလုပ်လုပ်ဖို့ အမိက ရည်ရွယ်တယ်ဆိုရင်တော့ Web Starter Kit ကို အသုံးပြုသင့်ပါတယ်။

Boilerplate တွေဟာ Software Project တစ်ခုကို အလျင်အမြန်အစပြုနိုင်စေဖို့ ကူညီပေးတဲ့ Tool တွေဖြစ်ပါတယ်။ ဒါပေမယ့် လက်တွေမှာ Boilerplate ရရှိဖို့အတွက် သက်ဆိုင်ရာ Website ကိုသွားပြီး Download လုပ်နေရပါသေးတယ်။ ဒီထက်ပိုပြီး အလုပ်တွင်ဖို့ဆိုရင် Command တစ်ချို့ Run လိုက်တာနဲ့ Boilerplate ကို အလိုအလျောက်တည် ဆောက်ပေးနိုင်တဲ့ Scaffold Generator (သို့မဟုတ်) Code Generator တွေကို အသုံးပြုရမှာ ဖြစ်ပါတယ်။

Code Generator တွေကပေးတဲ့ အားသာချက်ကတော့ Project ကို အစပြုနိုင်တဲ့ Boilerplate ကို တည်ဆောက်ပေးနိုင် ယုံသာမက၊ Code ရေးသားနေရင်း လိုအပ်တဲ့ Directory Structure ကိုလည်း ဖြည့်စွက်တည်ဆောက်ပေးနိုင်ကြသလို၊ အခြားလိုအပ်တဲ့ Code တွေကိုလည်း အလိုအလျောက် ရေးပေးနိုင်ပါသေးတယ်။ ဥပမာ – MVC Pattern ကို အသုံးပြုထားတဲ့ Project တွေမှာ ဆိုရင် လုပ်ဆောင်ချက်တစ်ခု ထည့်သွင်းတိုင်းမှာ Model ဖိုင်၊ View ဖိုင် နဲ့ Controller ဖိုင် တို့ကို တည်ဆောက်ပေးဖို့လိုတက်ပါတယ်။ လုပ်ဆောင်ချက်အသစ်တစ်ခု ထည့်သွင်းလိုတိုင်း ဒီဖိုင်တွေကို ကိုယ်တိုင် တည် ဆောက်နေ့မယ့်အစား Code Generator ကို တည်ဆောက်နိုင်းလိုက်နိုင်ခြင်းဖြစ်ပါတယ်။ ဒါအပြင် Database Table အမည်ပေးလိုက်ယုံနဲ့ အခြေခံ CRUD လုပ်ငန်းကို အလိုအလျောက်ရေးပေးတဲ့ လုပ်ဆောင်ချက်မျိုးတွေလည်း ပါဝင်တက် ပါသေးတယ်။

MVC နဲ့ CRUD အကြောင်းကိုဒီနေရာမှာ အကျယ်မချဲတော့ပါဘူး။ **Professional Web Developer** ရဲ့ အခန်း(၉) နဲ့ အခန်း(၁၃) တို့မှာ လေ့လာနိုင်ပါတယ်။

Ruby on Rails, CakePHP, Django, Symfony, ASP.NET MVC, ExpressJS စိတ် Framework တွေမှာ Scaffold Generator တွေ တစ်ခါတည်း အသင့်ပါဝင်ပါတယ်။ သူတို့မှာပါတဲ့ Generator တွေကတော့ Boilerplate တည်ဆောက် ဖို့ထက်စာရင် MVC Structure နဲ့ CRUD လုပ်ဆောင်ချက်တို့ကို အမိကဆောင်ရွက်ပေးကြခြင်းဖြစ်ပါတယ်။ အခုနောက် ပိုင်း Modern Web App တွေမှာ Yeomen လိုခေါ်တဲ့ Scaffold Generator ကို အသုံးများ ကြပါတယ်။ Yeomen ကို တော့ Boilerplate တည်ဆောက်ဖို့ရော၊ ဖြည့်စွက်လိုအပ်ချက်တွေအတွက်ပါ Code Generator အဖြစ်သုံးနိုင်ပါတယ်။

## 8.2 – Yeomen

Yeomen ဟာ NodeJS ကို အသုံးပြုရေးသားထားတဲ့ Scaffold Generator ဖြစ်ပါတယ်။ JavaScript Web App တွေ အတွက် အသုံးများပေမယ့် Code Generator လိုအပ်တဲ့ မည်သည့် Project အမျိုးအစားအတွက်မဆို အသုံးပြုနိုင်ပါတယ်။ Yeomen ကို အသုံးပြုနိုင်ဖို့အတွက် NodeJS လိုပါတယ်။ NodeJS Installer ကို [nodejs.org](http://nodejs.org) မှာ Download ရယူနိုင်ပါတယ်။ Windows, Mac, Linux မိမိအသုံးပြုလိုရာ OS အတွက် Installer ကို ရွေးချယ် ရယူနိုင်ပါတယ်။ Ubuntu Linux မှာဆိုရင်တော့ ကိုယ်တိုင် Download သွားလုပ်မနေပဲ အခုလို Install လုပ်နိုင်ပါတယ် –

```
$ sudo apt-get install nodejs nodejs-legacy
```

NodeJS နဲ့အတူ NPM လိုခေါ်တဲ့ NodeJS Package Manager တစ်ခု တစ်ခါတည်း တွဲဖက်ပါဝင်ပါတယ်။ NPM ဆိုတာ NodeJS Software Package တွေကို ရှာဖွေခြင်း၊ Install ပြုလုပ်ခြင်း၊ Update ပြုလုပ်ခြင်း စတဲ့လုပ်ငန်း တွေကို ဆောင်ရွက်စီမံပေးတဲ့ Software တစ်မျိုးပဲဖြစ်ပါတယ်။ Yeoman ဟာ NodeJS နဲ့ရေးသားထားတဲ့ Software Package တစ်ခုဖြစ်တဲ့အတွက် NPM ကိုသုံးပြီး Yeoman ကို အခုလို Install လုပ်ယူနိုင်ပါတယ် –

```
$ sudo npm install -g yo
```

NPM ဟာ Command Line Tool တစ်ခုဖြစ်တဲ့အတွက် Terminal (သို့မဟုတ်) Windows Command Line ကနေ တစ်ဆင့် အသုံးပြုရခြင်းဖြစ်ပါတယ်။ Ubuntu Linux မှာသာ ရှေ့ဆုံးက sudo Command ထည့်ရပြီး Windows မှာ ဆိုရင်တော့ sudo ထည့်စရာမလိုပါဘူး။ sudo အကြောင်းအသေးစိတ်သိရှိလိုရင်တော့ Ubuntu – သင့် အတွက် Linux ရဲ့အခန်း (၁၀) မှာ လေ့လာနိုင်ပါတယ်။

npm install yo လိုပြောလိုက်တာနဲ့ NPM က Yeoman Scaffold Generator ကို Download ယူပြီး Install လုပ်ပေးသွားမှာပါ။ အဲဒီလို Install လုပ်တဲ့အခါ လက်ရှိ Directory ထဲမှာ Install လုပ်သွားမှာဖြစ်လို့ လက်ရှိ Directory ထဲမှာပဲ အသုံးပြုလိုရမယ်ဆိုတဲ့ သဘောလည်းဖြစ်ပါတယ်။ နောင်ဘယ်နေရာကနေမဆို အသုံးပြုလိုတယ်ဆိုရင် NodeJS Global Directory ထဲမှာ Install လုပ်သွားဖို့လိုပါတယ်။ -g Option က ဒီရည်ရွယ်ချက်နဲ့ ထည့်သွင်းထားတာပါ။ -g Option သုံးထားတဲ့အတွက် NPM က Yeoman ကို NodeJS Global Directory ထဲမှာ Install လုပ်သွားပြီး၊ နောင် အသုံးပြုလိုရင် ထပ်မံ Install လုပ်စရာမလိုတော့ပဲ အသင့်သုံးနိုင်သွားမှာပါ။

ဒီအခန်းမှာ ဆက်လက်ဖော်ပြုမယ့် Command တွေဟာ လက်တွေ့ကူးယူစစ်းသပ်နိုင်တဲ့ Command တွေ ဖြစ်ပါတယ်။ ဒါပေမယ့် အသုံးပြုထားတဲ့ Operating System, Install လုပ်ထားတဲ့ Tool တွေရဲ့ Version နဲ့ File Permission သတ်မှတ်ထားပဲ အခြေအနေပေါ်မှတည်ပြီး အပြောင်းအလဲ ရှိနိုင်တာကို သတိပြုစေလိုပါတယ်။ လက်တွေ့စစ်းသပ်ရင်း ပြဿနာ တစ်စုံတစ်ရာရှိလာတဲ့အခါ ဖော်ပြုလာတဲ့ Error Message များနဲ့ Instruction များကို ကိုယ်တိုင် လေ့လာပြီး ဖြေရှင်းဖို့လိုအပ်နိုင်ပါတယ်။

လက်တွေ့မှာ Scaffold Generator တစ်ခုတည်းဆိုရင် မပြည့်စုပါဘူး။ Development Workflow တစ်ခုလုံးကို Auto-mate လုပ်နိုင်ဖို့အတွက် Package Manager တွေ Build Tool တွေ လိုပါသေးတယ်။ လက်စနဲ့ လိုအပ်တဲ့ Tool တွေ အကုန်လုံးကို တစ်ခါတည်း Install လုပ်ထားသင့်ပါတယ်။

```
$ sudo npm install -g yo bower grunt-cli
```

yo, bower နဲ့ grunt-cli ဆိုတဲ့ Tool သုံးခုကို တစ်ခါတည်း Install လုပ်လိုက်ခြင်းဖြစ်ပါတယ်။ Bower နဲ့ Grunt အကြောင်းကို မကြာခင်ဆက်လက်ဖော်ပြပါမယ်။ လောလောဆယ် Yeoman အကြောင်းကိုပဲ ဆက်ပါ၌ီးမယ်။ Yeoman ကိုသုံးပြီး App Boilerplate တွေ Generate လုပ်နိုင်ဖို့အတွက် Generator တွေကို ဆက်လက် Install လုပ်ပေးရပါ၌ီးမယ်။ Generator စာရင်းကို [yeoman.io/generators](http://yeoman.io/generators) မှာ လေ့လာနိုင်ပါတယ်။ စာရင်းထဲက အသုံးပြုလိုရာ Generator တစ်ခါ့ကို ကြိုးတင် Install လုပ်ထားသင့်ပါတယ်။ ဥပမာ -

```
$ sudo npm install -q generator-webapp generator-backbone
```

Web App Generator နဲ့ BackboneJS App Generator တို့ကို Install လုပ်လိုက်ခြင်းပဲဖြစ်ပါတယ်။ Generator တွေ Install လုပ်ပြီးပြီဆိုရင် Yeoman ကိုသုံးပြီး Boilerplate တွေ စတင် Generate လုပ်နိုင်ပြီးဖြစ်ပါတယ်။ အခြေခံ yo Command ကို Run ကြည့်ရင် ဒီလိုရလဒ်ကို ရရှိမှာဖြစ်ပါတယ်။

yo Command က Install လုပ်ထားတဲ့ Generator စာရင်းကို ဖော်ပြပေးခြင်းဖြစ်ပါတယ်။ အသုံးပြုလိုတဲ့ Generator ကိုရွေးပြီး Generate လုပ်နိုင်ပါတယ်။ နမူနာအနေနဲ့ Webapp ကိုရွေးပြီး Enter နှင့်လိုက်ရင် အခုလို ရလဒ်ကိုရရှိမှာပါ။

Make sure you are in the directory you want to scaffold into.  
This generator can also be run with: yo webapp



Out of the box I include HTML5 Boilerplate, jQuery, and a Gruntfile.js to build your app.

? What more would you like? (Press <space> to select)

- Bootstrap
- Sass
- Modernizr

Message ရဲ့ ပထမဆုံးလိုင်းမှာ၊ Source Code ဖိုင်တွေ စတင်တည်ဆောက်တော့မှာမို့ Project Directory ကို မှန်အောင်ရွေးထားရဲ့လားလို့ သတိပေးတာကို တွေ့ရမှာဖြစ်ပါတယ်။ အကယ်၍ Project Directory ကို မရွေးရသေးရင် ဒီနေရာတင်ရပ်ပြီး Project Directory ကို cd Command နဲ့ အရင်ရွေးပေးဖို့လိုပါတယ်။

ဆက်လက်ပြီး၊ နောင်တစ်ချိန် Web App Boilerplate တစ်ခု Generate လုပ်လိုရင် yo webapp Command နဲ့ တိုက်ရှိက် Generate လုပ်နိုင်ကြောင်းကိုလည်း Message ထဲမှာ ဖော်ပြထားပါသေးတယ်။ Web App နဲ့အတူ HTML5 Boilerplate, jQuery နဲ့ Setting ဖိုင်တစ်ခုဖြစ်တဲ့ Gruntfile.js တို့ကို တစ်ခါတည်းထည့်ပေးထားကြောင်းလည်း ဖော်ပြထားပါသေးတယ်။ ဆက်လက်ဖြည့်စွက်လိုတဲ့ Library ရှိရင်လည်း ဖော်ပြထားတဲ့ စာရင်းထဲကနေ ရွေးပြီး ထည့်နိုင်ပါ တယ်။ နှမူနာမှာ Bootstrap, Sass နဲ့ Modernizr ဆိုပြီး ရွေးစရာ (၃) ခုပေးထားပါတယ်။

နှမူနာအနေနဲ့ Bootstrap CSS Framework နဲ့ Modernizr JavaScript Library တို့ကို ရွေးထားပါတယ်။ Enter နှိပ်ပြီး နောက်တစ်ဆင့်ကို သွားတဲ့အခါ Yeoman က လိုအပ်တဲ့ Directory Structure နဲ့ Source Code ဖိုင်တွေကို စတင် တည်ဆောက်ပါတော့တယ်။

- ? What more would you like? Bootstrap, Modernizr
  - create Gruntfile.js
  - create package.json
  - create .gitignore
  - create .gitattributes
  - create .bowerrc
  - create bower.json
  - create .jshintrc
  - create app/styles/main.css
  - create app/favicon.ico
  - create app/robots.txt
  - create app/index.html
  - create app/scripts/main.js
  - invoke mocha

ဆက်လက်ပြီး bower install နဲ့ npm install ဆိုတဲ့ Command နှစ်ခုကို အလိုအလျောက် Run ပေးသွားပါ တယ်။

```
I'm all done. Running bower install & npm install for you to install the
required
dependencies. If this fails, try running the command yourself.

create    test/bower.json
create    test/.bowerrc
create    test/spec/test.js
create    test/index.html
bower chai#~1.8.0      cached git://github.com/chaijs/chai.git#1.8.1
bower chai#~1.8.0      validate 1.8.1 against git://github.com/chaijs/chai.git
...
...
```

အမှန်တော့ လိုအပ်တဲ့ Boilerplate ကို တည်ဆောက်ပြီးနေပါပြီ။ အင်တာနက်ကနေ လိုအပ်တဲ့ ဖြည့်စွက် Library ဖိုင်တွေကို ရယူနေတာပါ။ အဲဒီလို Download လုပ်နေတာကို မစောင့်လိုရင် Keyboard ကနေ Ctrl+C နှင့်ပြီးရပ်လိုက်ပါ။ နောက်တော့မှာ bower install နဲ့ npm install Command နှစ်ခုကို ကိုယ်တိုင် Run ပြီး ဆက်လုပ်နိုင်ပါတယ်။ တည်ဆောက်ရရှိထားတဲ့ Code Base ကို လေ့လာကြည့်ရင် အခုလို Structure ကို တွေ့ရမှာပဲ ဖြစ်ပါတယ်။

```
webapp
└── app
    ├── images
    ├── scripts
    │   └── main.js
    ├── styles
    │   └── main.css
    ├── favicon.ico
    ├── index.html
    └── robots.txt
└── test
    ├── bower.json
    ├── .gitignore
    └── Gruntfile.js
...
```

HTML5 Boilerplate ကိုပဲ လိုအပ်သလိုဖြည့်စွက်ထားတဲ့ Boilerplate တစ်ခုဖြစ်ပါတယ်။ ဒါပေမယ့် Website မှာ ကိုယ် တိုင်သွားပြီး Download လုပ်နေစရာမလိုတော့ပါဘူး။ yo webapp လို့ ပြောလိုက်တာနဲ့ Web App Boilerplate တစ်ခုကို ချက်ခြင်းရရှိနိုင်မှာပဲဖြစ်ပါတယ်။ အခြား App အမျိုးအစားတွေ Generate လုပ်လိုရင် လည်း -

```
$ yo backbone
$ yo angular
$ yo php
$ yo wordpress
$ yo laravel
$ yo django
$ yo firefox-os
```

- စသဖြင့် အသင့် Generate လုပ်ယူနိုင်ပါတယ်။ အသုံးပြုလိုတဲ့ Generator တွေကို ကြိုက်တင် Install လုပ်ထားဖို့ သာလို ပါတယ်။

ဒီနည်းနဲ့ Yeoman အကူအညီနဲ့ Scaffolding လုပ်ငန်းကို Automate လုပ်ထားနိုင်ခြင်းပဲ ဖြစ်ပါတယ်။ Generator တွေ အနေနဲ့ အသင့်ရှိနေတဲ့ Generator တွေကိုသုံးနိုင်သလို ကိုယ်တိုင်လည်း Generator တွေ တည်ဆောက် ထားနိုင်ပါတယ်။

Generator တစ်ခု ကိုယ်တိုင်တည်ဆောက် နိုင်ဖို့အတွက် yo generator ကို သုံးနိုင်ပါတယ်။

```
$ sudo npm install -g generator-generator
$ yo generator

  _-----_
  |  o   |
  |-----| Create your own Yeoman
  |  U   |
  |-----| generator with
  |  A   |
  |-----| superpowers!
  |  ~   |
  |-----| Y

? Would you mind telling me your username on GitHub? eimg
? What's the base name of your generator? tinymvc
  create package.json
  create app/index.js
...
...
```

yo generator ၏ GitHub Account နဲ့ Generator အမည်တိုကို မေးလာပါလိမ့်မယ်။ နမူနာမှာ GitHub Account အဖြစ် eimg ကိုပေးထားပြီး Generator အမည်အနေနဲ့ tinymvc လိုပေးထားပါတယ်။ ရရှိလာတဲ့ Directory Structure ကို လေ့လာကြည့်ရင် အခုလိုတွေရမှာ ဖြစ်ပါတယ်။

```
generator-tinymvc
├── app
│   └── templates
│       └── index.js
├── node_modules
└── test
    └── test-app.js
```

```

└── .gitignore
└── .jshintrc
└── package.json
└── ...

```

generator-tinyMVC ဆိတဲ့အမည်နဲ့ Generator Boilerplate တစ်ခုကို တည်ဆောက်ပေးသွားခြင်းဖြစ်ပါတယ်။ ရှုံးလာတဲ့ Structure ထဲမှာ အရေးအကြီးဆုံးက app Directory နဲ့ သူထဲက index.js တို့ပဲ ဖြစ်ပါတယ်။ app Directory ထဲမှာ Source Code Template တွေကို သိမ်းဆည်းရပြီး index.js ကတော့ Generator Script ဖြစ်ပါတယ်။ နမူနာအနေနဲ့ app/template Directory ထဲမှာ index.html နဲ့ css/style.css ဖိုင်တို့ကို ထည့် သွင်းလိုက်ပါမယ်။ Directory Structure အခုလိုဖြစ်သွားမှာပါ။

```

generator-tinyMVC
└── app
    ├── templates
    │   ├── css
    │   │   └── style.css
    │   └── index.html
    └── index.js
└── node_modules
└── test
    └── test-app.js
└── .gitignore
└── .jshintrc
└── package.json
└── ...

```

ပြီးတဲ့အခါ index.js ကိုဖွင့်ပြီး ဒီ Code နဲ့အစားထိုးလိုက်ပါ။

## JavaScript

```

1. var yeoman = require('yeoman-generator');
2.
3. module.exports = yeoman.generators.Base.extend({
4.     writing: {
5.         app: function () {
6.             this.fs.copy(
7.                 this.templatePath('css/style.css'),
8.                 this.destinationPath('css/style.css')
9.             );
10.            this.fs.copy(
11.                this.templatePath('index.html'),
12.                this.destinationPath('index.html')
13.            );
14.        }
15.    }
16. });

```

ဒါဟာ NodeJS နဲ့ ရေးသားထားတဲ့ ပရိုဂရမ်တစ်ခြားဖြစ်ပြီး `require()`, `module.exports` စုတဲ့ Code တွေရဲ့ အဓိပ္ပာယ်ကို အခန်း (၁၀) မှာ NodeJS အကြောင်းဖော်ပြတဲ့အခါ ထည့်သွင်းဖော်ပြပေးပါမယ်။ လိုဂ်းကတော့ Generator ကို Run တဲ့အခါ `app/template` ထဲက `css/style.css` နဲ့ `index.html` တို့ကို ကူးယူပေးပါလို့ သတ်မှတ်ထားလိုက်ခြင်းဖြစ်ပါတယ်။ ကျွန်တော်တို့ကိုယ်ပိုင် Generator တစ်ခုတည်ဆောက် ရရှိသွားဖြိုဖြစ်ပါတယ်။ ရရှိထားတဲ့ Generator ကို NPM Package အနေနဲ့ အခုလို `Install` လုပ်နိုင်ပါတယ်။

```
$ cd generator-tinymvc
$ sudo npm link

/usr/lib/node_modules/generator-tinymvc -> /home/eimg/generator-tinymvc
```

npm link Command က လက်ရှိ Generator Directory ရဲ့ Symbolic Link (Shortcut) ကို NodeJS Package Directory ထဲမှာ ထည့်သွင်းပေးသွားမှာဖြစ်ပါတယ်။ ဒါကြောင့်မို့ လက်ရှိ `generator-tinymvc` ဟာ NodeJS Package တစ်ခုဖြစ်သွားပြီ ဖြစ်ပါတယ်။ Yeoman ကနေတစ်ဆင့် ဒီ Generator ကို စတင်အသုံးပြုနိုင်ဖြစ်ပါတယ်။

```
$ yo tinymvc

  create css/style.css
  create index.html
```

ကျွန်တော်တို့ဖုန်တီးထားတဲ့ `tinymvc` Generator ကိုသုံးပြီး App Boilerplate တစ်ခုတည်ဆောက်လိုက်ခြင်းပဲဖြစ်ပါတယ်။ ရရှိလာတဲ့ Structure ကို လေ့လာကြည့်ရင် အခုလိုတွေ့ရမှာ ဖြစ်ပါတယ်။

```
tinymvc
  └── css
    └── style.css
  └── index.html
```

`css/style.css` နဲ့ `index.html` ဆိတဲ့ Source Code ပိုင်တွေပါဝင်တဲ့ Boilerplate တစ်ခုကို ရရှိသွားပြီပဲဖြစ်ပါတယ်။ Yeoman Generator အကြောင်းအသေးစိတ်ကို အောက်ပါလိပ်စာမှာ ဆက်လက်လေ့လာနိုင်ပါတယ်။

<http://yeoman.io/authoring/>

### 8.3 – Package Management

Package ဆိုတာ အသင့်သုံးနိုင်တဲ့ Software Component တစ်ခု (သို့မဟုတ်) Library တစ်ခုကို ဆိုလိုတာပါ။ အသင့် သုံးလိုရတဲ့ Package တွေရှိနေချိန်မှာ App တစ်ခုအတွက် လိုအပ်တဲ့လုပ်ဆောင်ချက် အားလုံးကို ကိုယ်တိုင်ရေးနေစရာ မလိုပါဘူး။ သင့်တော်တဲ့ Package တွေကို သူနေရာနဲ့သူ ရယူအသုံးချရမှာပဲဖြစ်ပါတယ်။

တစ်ချို့ Package တွေဟာ သီးခြားစီ ရပ်တည်အလုပ်လုပ်ကြပေမယ့် တစ်ချို့ Package တွေကတော့ တစ်ခုနဲ့ တစ်ခု အပြန်အလှန် မိန့်နေကြပါတယ်။ ဥပမာ - jQuery UI ကို အသုံးပြုလိုရင် jQuery ရှိမှုအသုံးပြုလိုရပါမယ်။ BackboneJS က jQuery နဲ့ UnderscoreJS တို့ပါမှ ပြည့်စုံပါတယ်။ ဒီနည်းနဲ့ Package တွေ အပြန်အလှန် မိန့်နေခြင်းကို Dependency လိုခေါ်ပါတယ်။

ပုံမှန်အားဖြင့် Package တစ်ခုကို အသုံးပြုလိုရင် သက်ဆိုင်ရာ Package Author ရဲ့ Website ကနေ Download ရယူရ ပါတယ်။ ဒီတော့ Package တစ်ခု အသုံးပြုလိုတိုင်း အင်တာနက်မှာရှာ၊ တွေ့တဲ့အခါ Download လုပ်၊ ရလာတဲ့ Archive ကို Extract လုပ်၊ ရလာတဲ့ Source Code ကို Project Code Base ထဲ ကူးထည့်၊ စသဖြင့် အဆင့်ဆင့်လုပ်ရပါတယ်။ Package Manager တွေက အဲဒီလို Package တွေ ရှာဖွေခြင်းနဲ့ ရယူအသုံးပြုခြင်းလုပ်ငန်းကို လွယ်ကူအောင် စီမံ ပေးထားကြပါတယ်။

Package Manager တွေက Install လုပ်ထားတဲ့ Package တစ်ခုအတွက် Update Version တွေရှိလာတဲ့အခါမှာ Update Version ကို နောက်တစ်ကြိမ် Download လုပ်တဲ့အလုပ်ကို ကိုယ်တိုင်လုပ်စရာမလိုအောင် စီမံပေးနိုင်ပါတယ်။ Package Dependency တွေကိုလည်း စီမံပေးနိုင်ပါတယ်။ ဥပမာ - jQuery UI ကို Install လုပ်တဲ့အခါ သူရဲ့ Dependency ဖြစ်တဲ့ jQuery ကို အလိုအလျောက် ရယူပေးနိုင်ပါတယ်။ ဆက်လက်ပြီး BackboneJS အတွက် jQuery ထပ်လိုတဲ့အခါမှာတော့ ပထမတစ်ကြိမ် ရယူထားပြီးဖြစ်တဲ့အတွက် အချိန်ကုန်ခံပြီးထပ်ရယူမနေတော့ပဲ ရှိနေတဲ့ jQuery ကိုပဲ အသုံးပြုပေးသွားမှာဖြစ်ပါတယ်။ ဒုဥကြည် Dependency Version ကိုလည်း ညီပေးနိုင်ပါသေးတယ်။ ဥပမာ - jQuery UI 1.11 ကိုအသုံးပြုနိုင်ဖို့ jQuery 2.0 လိုတယ်ဆိုကြပါစို့။ လက်ရှိ ရှိနေတာက jQuery 1.7 ဖြစ်နေရင် Version မကိုက်တဲ့သဘောပဲ ဖြစ်ပါတယ်။ အဲဒီလို အခြေအနေမျိုးတွေကို ဖြေရှင်းနိုင်တဲ့နည်းစနစ်တွေ Package Manager တွေမှာ ပါဝင်တက်ပါတယ်။

Node Package Manager (NPM) ဟာ Package Manager တစ်မျိုးဖြစ်ပါတယ်။ သူအကူအညီနဲ့ NodeJS Package တွေကို အလွယ်တစ်ကူ Install ပြုလုပ်ခြင်း၊ Update ပြုလုပ်ခြင်းတို့ကို ဆောင်ရွက်နိုင်ပါတယ်။ Package တစ်ခုကို Install လုပ်လိုရင် (သို့မဟုတ်) Update ပြုလုပ်လိုရင် npm install Command ကို အခုံ အသုံးပြုနိုင်ပါတယ်။

```
$ sudo npm install -g ionic
```

npm က Install လုပ်လိုတဲ့ Package အတွက် Dependency တွေ ရှိနေရင် တစ်ခါတည်း အလိုအလျောက် ရယူထည့် သွေးပေးသွားမှာ ဖြစ်ပါတယ်။ ဒီနည်းနဲ့ Package တစ်ခုရယူလိုတိုင်း Package Author ရဲ့ Website ကို သွားပြီး ကိုယ်တိုင် Download လုပ်နေစရာမလိုတော့ပဲ အသင့် ရယူအသုံးပြုနိုင်ခြင်းဖြစ်ပါတယ်။ Install လုပ်

ထားပြီးတဲ့ Package စာရင်းကို လေ့လာလိုရင် တော့ npm list Command ကို အသုံးပြု နိုင်ပါတယ်။

```
$ npm list -g

/usr/lib
└── generator-generator@0.6.1
    ├── chalk@1.0.0
    ├── ansi-styles@2.0.1
    ├── escape-string-regexp@1.0.3
    ├── has-ansi@1.0.3
    │   └── ansi-regex@1.1.1
    ├── get-stdin@4.0.1
    ├── strip-ansi@2.0.1
    │   └── ansi-regex@1.1.1
    ├── supports-color@1.3.1
    ├── github@0.2.3
    ├── lodash@2.4.1
    └── npm-name@1.0.4

...
```

Package တစ်ခုကို Uninstall ပြုလုပ်လိုရင်တော့ npm rm (သိမဟုတ်) npm uninstall ကို အသုံးပြုနိုင်ပါတယ်။

```
$ sudo npm uninstall -g generator-tinymvc
unbuild generator-tinymvc@0.0.0
```

ဒီနည်းနဲ့ Package တွေ Install ပြုလုပ်ခြင်း၊ Update ပြုလုပ်ခြင်း၊ Dependency များစီမံခြင်းတို့ကို ကိုယ်တိုင် Package Manager အကူအညီနဲ့ စီမံနှင့်ခြင်းပဲဖြစ်ပါတယ်။

## 8.4 – Bower

NPM ဟာ NodeJS နဲ့အတူ တဲ့ဖက်ပါဝင်လာတဲ့ Package Manager တစ်ခုဖြစ်တဲ့အားလျှော်စွာ၊ NodeJS Package တွေကို စီမံဖို့အတွက်သာအသုံးများပါတယ်။ Web App တွေအတွက် Package တွေကိုစီမံဖို့ Bower လို ခေါ်တဲ့ Package Manager တစ်ခုကို အသုံးပြုနိုင်ပါတယ်။ Yeoman ကို Install လုပ်စဉ်က Bower ကို တစ်ခါတည်း ထည့်သွင်း Install လုပ်ခဲ့ပြီးဖြစ်ပါတယ်။ အကယ်၍ မလုပ်ရသေးရင်လည်း အခုံလို Install လုပ်နိုင်ပါတယ်။

```
$ sudo npm install -g bower
```

Package တွေ Install လုပ်ဖို့အတွက် bower install ကို အသုံးပြုနိုင်ပါတယ်။ ဥပမာ -

```
$ bower install jquery
bower jquery#*                                not-cached git://github.com/jquery/jquery.git#*
bower jquery#*                                resolve git://github.com/jquery/jquery.git#*
bower jquery#*                                download https://github.com/jquery/2.1.3.tar.gz
bower jquery#*                                progress received 0.1MB
bower jquery#*                                progress received 0.2MB
...
bower jquery#~2.1.3                         install jquery#2.1.3
jquery#2.1.3 bower_components/jquery
```

Bower က Install လုပ်လိုက်တဲ့ Package တွေကို bower\_components ဆိုတဲ့ Directory ထဲမှာ သိမ်းဆည်းပေး သွားမှာဖြစ်ပါတယ်။ နမူနာမှာ jQuery ကို Install လုပ်ထားပါတယ်။ Install လုပ်ပြီးနောက် ရရှိလာတဲ့ Directory Structure ကို လေ့လာကြည့်ရင် အခုလိုတွေရမှာပါ။

```
myapp
└── bower_components
    └── jquery
        ├── bower.json
        ├── dist
        │   ├── jquery.js
        │   ├── jquery.min.js
        │   └── jquery.min.map
        └── MIT-LICENSE.txt
    └── src
        └── style.css
└── index.html
```

bower\_components Directory အတွင်းမှာ jQuery ရဲ့ မူလ Source Code တွေရော့ အသင့်သုံးလိုရင်အင် Minified လုပ်ထားပေးတဲ့ Version တို့ကိုပါ Download ရယူထည့်သွင်းပေးထားတာကို တွေ့ရမှာဖြစ်ပါတယ်။ Bower Package စာရင်းကို ဒီနေ့ရာမှာ လေ့လာ နိုင်ပါတယ်။

<http://bower.io/search/>

Bower ရဲတူးခြားချက်က၊ Bower Package တွေကိုသာမက၊ Git Repository တွေနဲ့ Website URL တွေကိုပါ Install လုပ်ပေးနိုင်ခြင်းဖြစ်ပါတယ်။ ဥပမာ – GitHub မှာ Host လုပ်ထားတဲ့ Git Repo တစ်ခုကို အခုလို Install လုပ်နိုင်ပါ တယ်။

```
$ bower install eimg/jquery-voter
...
bower jquery-voter#*                         resolved git://github.com/eimg/jquery-voter.git
bower jquery-voter#*                         install jquery-voter
jquery-voter#143a070c8a bower_components/jquery-voter
```

GitHub Handle လိုခေါ်တဲ့ GitHub Username/Git Repo Name ကို တွဲဖက်ပေးရခြင်းဖြစ်ပါတယ်။ GitHub မဟုတ် တဲ့ တစ်ခြား Website တွေက Package တွေကို Install လုပ်လိုရင်လည်း Package ရှိရာ URL ကိုအန်းပေးခြင်းဖြင့် လည်း Install လုပ်နိုင်ပါသေးတယ်။

```
$ bower install http://eimaung.com/u4urc/.vimrc
...
bower .vimrc#*           install .vimrc#e-tag:1206bc
.vimrc#e-tag:1206bc-5b  bower_components/.vimrc
```

Bower Package တစ်ခုကို ကိုယ်တိုင် ဖန်တီးလိုတယ်ဆိုရင် bower init Command ကို သုံးပြီး ဖန်တီးနိုင်ပါတယ်။ နမူနာအနေနဲ့ Directory တစ်ခုအောက်ပြီး bower install နဲ့ Package တစ်ခု၏ Install လုပ်လိုက်ပါမယ်။

```
$ mkdir mypackage && cd mypackage
$ bower install jquery bootstrap
...
bower jquery#>= 1.9.1           install jquery#2.1.3
bower bootstrap#~3.3.4          install bootstrap#3.3.4
jquery#2.1.3  bower_components/jquery
bootstrap#3.3.4  bower_components/bootstrap
  └── jquery#2.1.3
```

အခုချိန် Directory Structure ကိုလေ့လာကြည့်ရင် အခုလိုတွေရမှာပါ။

```
my package
└── bower_components
    ├── bootstrap
    │   ├── bower.json
    │   ├── dist
    │   ├── fonts
    │   └── js
    └── jquery
        ├── bower.json
        ├── dist
        └── src
```

bower\_components ထဲမှာ bootstrap နဲ့ jquery ဆိုတဲ့ Component နှစ်ခုရှိနေပါတယ်။ ဆက်လက်ပြီး App ရေးသားခြင်းလုပ်ငန်းကို လုပ်ရှိလုပ်စဉ်အတိုင်း ဆက်လက်ဆောင်ရွက်နိုင်ပါတယ်။ နမူနာအနေနဲ့ app.js ဆိုတဲ့ ဖိုင်နဲ့ index.html ဆိုတဲ့ ဖိုင်နှစ်ခု ထည့်သွင်းလိုက်ပါမယ်။ Directory Structure အခုလိုပြောင်းသွားမှာပါ။

```

myapp
└── bower_components
    ├── bootstrap
    │   ├── bower.json
    │   ├── dist
    │   ├── fonts
    │   └── js
    └── jquery
        ├── bower.json
        ├── dist
        └── src
└── app.js
└── index.html

```

ဒါ App ပါ။ jQuery နဲ့ Bootstrap ဆိုတဲ့ Package နှစ်ခုကို အသုံးပြုထားတဲ့ App တစ်ခုဖြစ်ပါတယ်။ App ကို တည်ဆောက် ပြီးစိတဲ့အခါ Bower Package အဖြစ် ပြောင်းဖို့အတွက် bower.json ဆိုတဲ့ Package Information ဖိုင်တစ်ခု ထည့်သွင်းပေးရပါတယ်။ ကိုယ်တိုင် တည်ဆောက်နိုင်သလို bower init ကို Run ပြီး တော့လည်း တည် ဆောက်နိုင်ပါတယ်။

```

$ bower init
? name: mypackage
? version: 0.1.0
? description: Bower package test
? main file: app.js
? authors: Ei Maung <eimg@fairwayweb.com>
? set currently installed components as dependencies?: Yes

{
  name: 'mypackage',
  version: '0.1.0',
  authors: [
    'Ei Maung <eimg@fairwayweb.com>'
  ],
  description: 'Bower package test',
  main: 'app.js',
  keywords: [
    'test'
  ],
  dependencies: {
    bootstrap: '^3.3.4',
    "jquery": '^2.1.3'
  }
}

```

bower init က Package အမည်၊ Version, Author စတဲ့အချက်အလက်တွေကို မေးပါလိမ့်မယ်။ Bower Package တွေဟာ ကျွန်ုပ်တော်တို့ အခန်း (၅) မှာဖော်ပြခဲ့တဲ့ Semantic Versioning ကို လိုက်နာရပါတယ်။ ဒါ ကြောင့် Version နံပါတ်ကို နှစ်သက်သလိုပေးနိုင်ပေမယ့် Semantic Version သတ်မှတ်ချက်နဲ့ကိုက်ညီတဲ့ Version နံပါတ် ဖြစ်ဖို့တော့လိုပါတယ်။

name အတွက်အမည်ပေးရာမှာ အောက်ပါသတ်မှတ်ချက်တွေနဲ့ ကိုက်ညီဖိုလိုပါတယ်။

1. အခြား Package တစ်ခုခုက မပေးရသေးတဲ့ Unique Name ဖြစ်ရပါမယ်။
2. a to z စာလုံးသေးတွေနဲ့ 0 to 9 ဂဏ်နှင့်တွေ အသုံးပြုနိုင်ပါတယ်။ Dash နဲ့ Dot ကိုလည်း သုံးနိုင်ပါတယ်။
3. Dash နဲ့ Dot တွေကို နှစ်ခုတဲ့ သုံးခုတဲ့ မသုံးသင့်ပါဘူး (ဥပမာ my-package လို့ မပေးသင့်ပါဘူး)။
4. စာလုံးရေ (၅၀) အောက်မှာပဲ ပေးရပါတယ်။

main file နေရာမှာ ဒါ Package ရဲ့ အဓိက Source Code ဖိုင် (သို့မဟုတ်) ဖိုင် Array ကို ပေးရပါတယ်။ လက် ရှိ Install လုပ်ထားတဲ့ Package တွေကို Dependency စာရင်းထဲ ထည့်ပေးရမလားဆိုတဲ့ မေးခွန်းကို Yes လို့ ဖြေထား တဲ့အတွက် ကျွန်တော်တို့ Install လုပ်ထားတဲ့ jQeury နဲ့ Bootstrap ကို Dependency စာရင်းထဲမှာ အလိုအလျောက် ထည့်သွင်းပေးသွားမှာ ဖြစ်ပါတယ်။ မေးခွန်းအားလုံး ဖြော်ပြုတဲ့ အခါ bower.json မှာ ပါဝင် မယ့် JSON Code ကို ဖော်ပြပေးပါလိမ့်မယ်။ JSON Code ကိုလေ့လာကြည့်ရင် ကျွန်တော်တို့ ဖြော်ပြုတဲ့ အဖြေ တွေကိုပဲ သူနေရာနဲ့သူ ထည့် သွင်းပေးထားခြင်း ဖြစ်တယ်ဆိုတာ တွေ့ရမှာပါ။

bower init နဲ့ Package Information တွေ သတ်မှတ်ပြီးနောက်မှ ထပ်မံ Install လုပ်တဲ့ Package တွေကို Dependency စာရင်းမှာ အလိုအလျောက် ပါဝင်သွားစေလိုရင် --save Option သုံးပြီး Install လုပ်နိုင်ပါတယ်။

```
$ bower install underscore --save
```

--save Option နဲ့ ထပ်မံ Install လုပ်လိုက်တဲ့ Package တွေကို Dependency စာရင်းထဲမှာ တစ်ခါတည်း ထည့် သွင်းပေးသွားမှာမို့ bower.json ရဲ့ dependencies Section မှာလည်း အခုလို ပြောင်းလဲသွားမှာ ဖြစ်ပါတယ်။

JSON

```
"dependencies": {
  "bootstrap": "~3.3.4",
  "jquery": "~2.1.3",
  "underscore": "~1.8.2"
}
```

နမူနာအရ ဒါ Package အလုပ်လုပ်ဖို့အတွက် Bootstrap 3.3.4, jQuery 2.1.3 နဲ့ UnderscoreJS 1.8.2 တို့ကို Dependency အနေနဲ့ လိုအပ်တယ်လို့ သတ်မှတ်ထားခြင်း ဖြစ်ပါတယ်။ Dependency Package Version နံပါတ် ကို Range နဲ့လည်း သတ်မှတ်နိုင်ပါတယ်။ ဥပမာ >=1.2.9 ဆိုရင် 1.2.9 နှင့်အထက် ဆိုတဲ့အဓိပ္ပာယ်ဖြစ်ပါတယ်။ >1.2.9 <2.0.0 ဆိုရင်တော့ 1.2.9 နဲ့ 2.0.0 ကြားဆိုတဲ့ အဓိပ္ပာယ်ဖြစ်ပါတယ်။ bower.json ဖိုင်မှာ ထည့်သွင်းသတ်မှတ်နိုင်တဲ့ Option အပြည့်အစုံကို အောက်ပါလိပ်စာမှာ လေ့လာနိုင်ပါတယ်။

<https://github.com/bower/bower.json-spec>

ကျွန်တော်တို့ တည်ဆောက်လိုက်တဲ့ Bower Package ကို bower install Command ကနေတစ်ဆင့် Install လုပ်နိုင်ဖို့အတွက် Register လုပ်ပေးဖို့လိုပါတယ်။ Register လုပ်လိုတဲ့ Bower Package ဟာ Git Repo တစ်ခု ဖြစ်သည့်ပြီး Semantic Version သတ်မှတ်ချက်နဲ့အညီပေးထားတဲ့ Git Commit Tag လည်း ပါဝင်ရပါတယ်။ အခုပုံ သတ်မှတ်နိုင်ပါတယ်။

```
$ git init && git add .
$ git commit -m "Initial commit"
$ git tag 0.1.0
```

Git Repo လည်ဖြစ်ပြီ၊ Version Tag လည်းသတ်မှတ်ပြီးပြီဆိုရင် Package ကို အများရယူနိုင်တဲ့ တစ်နေရာမှာ Publish လုပ်ထားပေးရပါတယ်။ GitHub ကို အသုံးပြုကြလေးရှိပါတယ်။ GitHub မှာ Git Repo တစ်ခု Publish လုပ်ပုံကို ဒီနေရာမှာ မပြောတော့ပါဘူး။ [github.com](https://github.com) မှာ သွားပြီး ကိုယ်တိုင်ပဲလေ့လာကြည့်လိုက်ပါ။ GitHub က လိုအပ်တဲ့ Instruction တွေကို အပြည့်အစုံပေးထားပါတယ်။

Register လုပ်ဖို့အတွက် bower register Command ကို အသုံးပြုနိုင်ပါတယ်။ ဥပမာ -

```
$ bower register mypackage https://github.com/eimg/mypackage.git
```

နမူနာမှာပေးထားတဲ့ mypackage ဆိုတာက Package အမည်ဖြစ်ပါတယ်။ သူနောက်က လိပ်စာကတော့ Git Repo တည်ရှိရာကို ညွှန်းထားတဲ့လိပ်စာ ဖြစ်ရမှာဖြစ်ပါတယ်။ Bower က Package Registration အတွက် ဘာ မှ ကန်သတ် မထားပါဘူး။ မည်သူမဆို မိမိတို့ Package ကို Register လုပ်နိုင်တယ်ဆိုတဲ့ သဘောတူဖြစ်ပါတယ်။ Package အမည် ကသာ အရင်ကသူများ Register မလုပ်ဖူးတဲ့ Unique Name ဖြစ်ဖို့လိုပါတယ်။ ဒီလို Register လုပ်ပြီးပြီဆိုရင် bower install နဲ့ မည်သူမဆို ရယူအသုံးပြုနိုင်ပြီဖြစ်ပါတယ်။

bower install က Package တစ်ခုကို Install လုပ်တိုင်းမှာ bower.json ဖိုင်ထဲမှာ ပါဝင်တဲ့ Dependency တွေကို အလုံအလျောက် တစ်ခါတည်းထည့်သွင်း Install လုပ်ပေးသွားမှာဖြစ်တဲ့အတွက် ကျွန်တော်တို့ Package ကို Install လုပ်ရင် Bootstrap, jQuery နဲ့ UnderscoreJS တို့ကိုလည်း Bower က တစ်ခါတည်း Install လုပ်ပေးသွားတော့မှာ ဖြစ်ပါတယ်။

ဒီနည်းနဲ့ ကိုယ့် Project အတွက်လိုအပ်တဲ့ Package တွေကို Bower အကူအညီနဲ့ ရယူနိုင်သလို ကိုယ့် App ကို လည်း အများ ရယူအသုံးပြုနိုင်တဲ့ Bower Package အဖြစ် တည်ဆောက်လိုရင် တည်ဆောက်ထားနိုင်မှာဖြစ်ပါတယ်။

## 8.5 – Build

Software Project တစ်ခုမှာ Development နဲ့ Production ဆိုပြီး အခြေအနေနှစ်ခုအမြှုပါတယ်။ ဥပမာ အားဖြင့် Software ကို တည်ဆောက်ရေးသားနေဆဲ Development Stage မှာ Error Message တွေကို ပြည့်စုံနိုင်သမျှ ပြည့်စုံအောင် ဖော်ပြနိုင်လေ ကောင်းလေပဲဖြစ်ပါတယ်။ ဒီတော့မှ ပြဿနာတစ်ခုတစ်ရာရှိလာတဲ့အခါ Error Message က ပေး တဲ့ အချက်အလက်ကို လေ့လာပြီး ဖြေရှင်းနိုင်မှာဖြစ်ပါတယ်။ ဒါပေမယ့် အများ အသုံးပြုနိုင်ဖို့ ဖြန့်ချုပိလိုက်တဲ့ Production Stage မှာတော့ Error Message တွေ အကုန်ပြနေလို့ မဖြစ်တော့ပါဘူး။ မလွှဲသာမရောင်သာ ဖော်ပြရမယ့် Fatal Error တွေလာက်ကိုသာ ဖော်ပြပြီး၊ Warning တွေ၊ Notice တွေ ကိုတော့ မဖော်ပြသင့်တော့ပါဘူး။

ပြီးတော့ User ထည့်သွင်းတဲ့ အချက်အလက် အစစ်အမှန်မရှိသေးခင်မှာ၊ Dummy Data လိုပေါ်တဲ့ နမူနာ အချက်အလက် တွေကို Development Stage မှာ ခဲ့လှယာယိုသော့နဲ့ ထည့်သွင်းစမ်းသပ်ရတက်ပါတယ်။ Production Stage မှာ အဲဒီယာယို အချက်အလက်တွေကို ပြန်လည်ရှင်းလင်းထားပေးဖို့ လိုပါတယ်။ ဒါအပြင် Development Stage မှာ အများ ရှာရလွယ်အောင် Function တစ်ခုချင်းစီရဲ့ လုပ် ဆောင်ချက်တွေကို မှတ်တမ်း တင်တဲ့ Debug Log တွေ အသုံးပြုရတက် ပါတယ်။ Production Stage မှာ Debug Log ယူနေဖို့ မလိုတော့ပါဘူး။ Unit Test လို Test Code တွေ ထည့်သွင်း ရေးသားထားတယ်ဆိုရင်လည်း Development Stage မှာသာ အဲဒီ Test Code တွေကို Run ဖို့လိုပါတယ်။ Production Stage မှာ Run ဖို့ မလိုတောာပါဘူး။ ဒါဟင် မကသေးပါဘူး။ Development Stage မှာ စမ်းခါနီးတော့မှ ချက်ခြင်း Compile လုပ်ပြီးစမ်းနေတဲ့ ကိစ္စတွေရှိရင်လည်း Production Stage အတွက်ဆိုရင် တစ်ခါတည်း အပြီး Compile လုပ် ထားပေးဖို့ လိုနိုင်ပါတယ်။

လိုရင်းကတော့ Development Stage မှာ Code ရေးရတာ ထိရောက်မြန်ဆန်ရေးကို အသားပေးတဲ့ Setting တွေကို အသုံးပြုရပြီး Production Stage မှာတော့ Software ရဲ့ စမ်းဆောင်ရည် ကောင်းမွန်ရေးကိုအသားပေးတဲ့ Setting တွေကို အသုံးပြုရမှာဖြစ်ပါတယ်။

Build လုပ်တယ်ဆိုတာ Development Code ကို Production ဖြစ်အောင် ပြောင်းယူတဲ့လုပ်ငန်းစဉ်ပဲဖြစ်ပါတယ်။ Software ကို ရေးသားတည်ဆောက်စဉ်မှာ Build လုပ်တဲ့လုပ်ငန်းကို မကြာမကြာလုပ်ရမှာပါ။ နေ့စဉ် လုပ်ရမှာပါ။ ဒါကြောင့် Build လုပ်တဲ့လုပ်ငန်းစဉ်ကို တစ်ခုချင်း ကိုယ်တိုင်လုပ်နေရမယ်ဆိုရင် အလုပ်တွင်မှာ မဟုတ်ပါဘူး။ Build Process မှာပါဝင်တဲ့ လုပ်ငန်းစဉ်အသေးစိတ်ကတော့ Project တစ်ခုနဲ့တစ်ခု တူမှာမဟုတ်ပါဘူး။ Web App တစ်ခုကို Build လုပ်တဲ့အခါမှာတော့ အောက်ပါ လုပ်ငန်းတွေ ပါဝင်ရလေ့ရှိပါတယ်။

1. Run tests
2. Compile preprocessor
3. Generate static template
4. Lint JavaScript
5. Minify JavaScript, CSS and HTML
6. Concatenate JavaScript and CSS
7. Optimize Images
8. Create distribution package

Project ကို Production ပြောင်းတော့မယ်ဆိုရင် **Test** တွေအကုန် Pass ဖြစ်ထားဖို့လိုပါတယ်။ ဒါကြောင့် Build လုပ်ငန်းစဉ်၏ ပထမဆုံးအလုပ်ကတော့ Unit Test အပါအဝင် Test Code တွေကို Run ဖို့ပဲဖြစ်ပါတယ်။ Project မှာ CoffeeScript, LESS စိတ် **Preprocessor** နည်းပညာတွေထံးထားတယ်ဆိုရင် သက်ဆိုင်ရာ Compiler ကို သုံးပြီး JavaScript နဲ့ CSS Code တွေအဖြစ် ပြောင်းယူရပါတယ်။ အကယ်၍ **Template** Language တွေ သုံးထားတယ် ဆိုရင်လည်း Static HTML Template တွေရအောင် Generate လုပ်ထားဖို့လိုပါတယ်။ ဆက်လက်ပြီး JavaScript Code Quality ကောင်းဖို့အတွက် Code Analysis Tool တွေကိုသုံးပြီး **Lint** လုပ်ဖို့လိုပါတယ်။ ပြီးတဲ့ အခါ JavaScript, CSS နဲ့ HTML တွေကို ဖိုင် Size သေးသွားအောင် **Minify** လုပ်ရပါတယ်။ HTTP Request တွေ ကို လျှော့ချိန်ဖို့ Minify လုပ်ထားပြီးသား JavaScript နဲ့ CSS တွေကို **Concatenate** လုပ်ပြီး ပေါင်းစပ်ပေးရပါတယ်။ HTTP Request တွေလျှော့ချွင်းဟာ Performance Optimization အတွက် အဓိကကျတဲ့ ဆောင်ရွက် ချက်တစ်ခုဖြစ်ကြောင်းကို **Professional Web Developer** ရဲ့ အခန်း (၂၀) မှာ ဖော်ပြထားပြီး ဖြစ်ပါတယ်။ **Image** တွေကို အရည်အသွေးမကျ စေပဲ ဖိုင် Size သေးသွားအောင် Optimize လုပ်ပေးရပါတယ်။ အားလုံးပြီး ပို့တော့မှ ရလာတဲ့ရလဒ်ကို အသင့်သုံးနိုင် တဲ့ **Package** ဖြစ်အောင် ပြင်ဆင်ထားပေးရမှာဖြစ်ပါတယ်။

**အခန်း (၆)** မှာဖော်ပြခဲ့တဲ့ Software Team တစ်ခုရဲ့စွမ်းဆောင်ရည်တိုင်းတာတဲ့ အချက် (၁၂) ချက်ထဲမှာလည်း Build နဲ့ပက်သက်ပြီး (၂) ချက်ပါဝင်ပါတယ်။ Build လုပ်ငန်းစဉ်တစ်ခုလုံးကို တစ်နေရာတည်းကနေ တစ်ထိုင်တည်း ဆောင်ရွက်နိုင်ဖို့လိုတယ်ဆိုတဲ့အချက်နဲ့ နောက် Build လုပ်ရမယ်ဆိုတဲ့အချက်တို့ပဲဖြစ်ပါတယ်။ ဒီလို Build လုပ်ငန်းစဉ်တစ်ခုလုံးကို တစ်နေရာတည်းကနေ တစ်ထိုင်တည်း ဆောင်ရွက်နိုင်ပြီး၊ နောက်လည်း Build လုပ်နိုင်ဖို့ဆိုရင် Manual လုပ်နေလို့ အဆင် မပြောတော့ပါဘူး။ Automate လုပ်ထားရပါတော့မယ်။

## 8.6 – Grunt

Grunt ဆိုတာအမြတ်စုံထပ်ခါထပ်ခါလုပ်ရတဲ့ လုပ်ငန်းတွေကို အလိုအလျောက်လုပ်ပေးနိုင်အောင် Automate လုပ်ထား နိုင်တဲ့ Task Runner Tool တစ်ခုပဲဖြစ်ပါတယ်။ NodeJS ကို အသုံးပြုရေးသားထားတဲ့ Tool တစ်ခုဖြစ်ပြီး Yeoman ကို Install လုပ်စဉ်က တစ်ခါတည်းထည့်သွင်း Install လုပ်ခဲ့ပြီး ဖြစ်ပါတယ်။ မလုပ်ရသေးရင် လည်း အခုလို Install လုပ်နိုင် ပါတယ်။

```
§ sudo npm install -g grunt-cli
```

Grunt ကို စတင်အသုံးပြုနိုင်ဖို့ Project Directory ထဲမှာ package.json ဆိုတဲ့ဖိုင်နဲ့ Gruntfile.js ဆိုတဲ့ ဖိုင်တို့ကို တည်ဆောက်ခြင်းအားဖြင့် စတင်အသုံးပြုနိုင်ပါတယ်။ package.json ဖိုင်ဆိုတာ bower.json လိုပဲ Package Information တွေကိုစုစုပေါင်းထားတဲ့ ဖိုင်တစ်ခုဖြစ်ပါတယ်။ Bower Package Manager က Package Information အတွက် bower.json ကို သုံးပြီး NPM Package Manager ကတော့ package.json ကို Package Information အတွက် အသုံးပြုခြင်းဖြစ်ပါတယ်။ ဒါကြောင့် package.json ဖိုင်ရရှိဖို့အတွက် npm init Command နဲ့ တည်ဆောက်ယူနိုင် ပါတယ်။ package.json ဖိုင်ထဲမှာ Dependency များအဖြစ် Grunt နဲ့ Grunt Plugin များကို ကြော်ပေးရလေ့ရှိပါတယ်။ နမှနာ package.json ဖိုင်တစ်ခု ဖော်ပြပေးလိုက်ပါတယ်။

## JSON

```

1. {
2.   "name": "my-project-name",
3.   "version": "0.1.0",
4.   "devDependencies": {
5.     "grunt": "~0.4.5"
6.   }
7. }
```

my-project-name နေရာမှာ နှစ်သက်ရာအမည်တစ်ခုကို ပေးနိုင်ပါတယ်။ devDependencies နေရာမှာ grunt 0.4.5 ကို Dependency အနေနဲ့ ကြော်ပေးထားပါတယ်။ Build လုပ်လိုက်ချိန်မှာ အလိုအလျောက် လုပ် သွားစေလိုတဲ့ Task တွေကို ကိုယ်တိုင်ရေးသားလိုအပ်သလို အသင့်ရေးပေးထားတဲ့ Plugin တွေကိုလည်း အသုံးပြုနိုင်ပါတယ်။ အသုံးပြုလိုတဲ့ Plugin တွေကို ထပ်မံ Install လုပ်ပေးဖို့တော့လိုပါတယ်။ ဥပမာ – JavaScript Code တွေကို Minify လုပ်ပေးနိုင်တဲ့ Grunt Plugin တစ်ခုကို အခုလုံ Install လုပ်နိုင်ပါတယ်။

```
$ sudo npm install grunt-contrib-uglify --save-dev
```

--save-dev Option က package.json ရဲ့ devDependencies စာရင်းထဲမှာ grunt-contrib-uglify ကို အလိုအလျောက်ထည့် သွင်းသွားစေလိုတဲ့အတွက် အသုံးပြုခြင်းဖြစ်ပါတယ်။ package.json ကို ပြန် ဖွင့်ကြည့်ရင် အခုလုံတွေ့ရမှာပါ။

## JSON

```

"devDependencies": {
  "grunt": "^0.4.5",
  "grunt-contrib-uglify": "^0.5.0"
}
```

Gruntfile.js ထဲမှာတော့ Grunt Plugin တွေအတွက် Configuration တွေနဲ့ Build လုပ်ချိန်မှာ အလိုအလျောက် အလုပ်လုပ်သွားစေလိုတဲ့ Task တွေ ပါဝင်မှာဖြစ်ပါတယ်။ နဲ့မှန် Gruntfile.js ကို ဖော်ပြလိုက်ပါတယ်။

## JavaScript

```

1. module.exports = function(grunt) {
2.
3.   grunt.initConfig({
4.     uglify: {
5.       build: {
6.         files: {
7.           'js/app.min.js': ['js/mobile.js', 'js/app.js']
8.         }
9.       }
10.    })
}
```

```

11.  });
12.
13.  grunt.loadNpmTasks('grunt-contrib-uglify');
14.  grunt.registerTask('default', ['uglify']);
15.
16. }

```

နမူနာရဲ့ လိုင်းနံပါတ် (၇) မှာ လေ့လာကြည့်ရင် js/mobile.js နဲ့ js/app.js တို့ကို Minify လုပ်ပြီး js/app.min.js အမည်နဲ့သိမ်းပေးဖို့ သတ်မှတ်ထားခြင်းဖြစ်ပါတယ်။ အခုခံရင် package.json ရေး Gruntfile.js ပါရရှိသွားပြီး Grunt ကိုအသုံးပြုပြီး စတင် Build လုပ်နိုင်ပြီး ဖြစ်ပါတယ်။ ကျွန်တော်တို့ရဲ့ လက်ရှိ Source Code Directory Structure က အခုလိုဖြစ်မှာပါ။

```

my-project
├── js
│   ├── app.js
│   └── mobile.js
└── node_modules
    └── grunt
        └── grunt-contrib-uglify
└── Gruntfile.js
└── package.json

```

js Directory ထဲမှာ app.js နဲ့ mobile.js ဆိုတဲ့ Script ဖိုင်နှစ်ခုရှိနေပါတယ်။ Build လုပ်ဖို့အတွက် grunt Command ကို Run လိုက်ယံပါပဲ။

```

$ grunt

Running "uglify:build" (uglify) task
>> 1 file created.

Done, without errors.

```

grunt Command စာ Gruntfile.js ထဲမှာ သတ်မှတ်ထားတဲ့ Configuration အတိုင်း app.js နဲ့ mobile.js တို့ကို Minify လုပ်ပြီး app.min.js ဆိုတဲ့အမည်နဲ့ Generate လုပ်ပေးသွားမှာဖြစ်ပါတယ်။

```

my-project
├── js
│   ├── app.js
│   ├── app.min.js
│   └── mobile.js
└── node_modules
    └── grunt
        └── grunt-contrib-uglify
└── Gruntfile.js
└── package.json

```

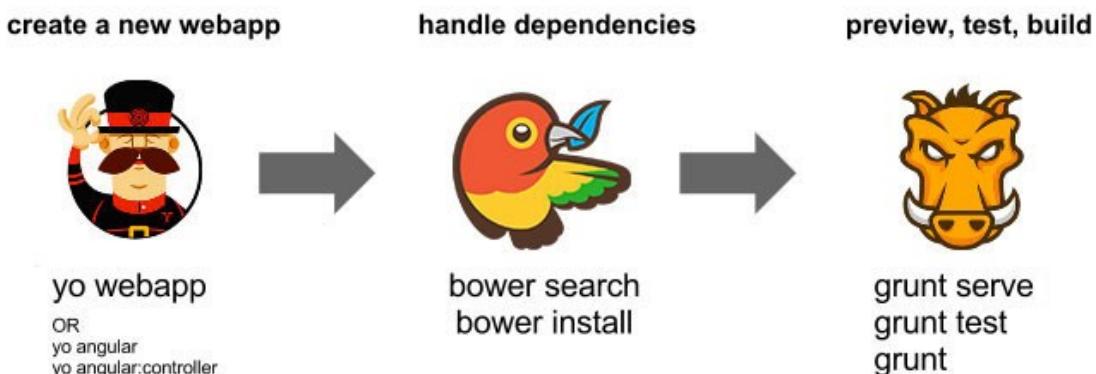
Source Code Directory Structure ကို ပြန်လေ့လာကြည့်ရင် js Directory ထဲမှာ app.min.js ဆိတဲ့ Output ဖိုင်တစ်ခု တိုးလာတာကို တွေ့ရမှာပဲ ဖြစ်ပါတယ်။

ဒီနည်းနဲ့ Test Code တွေ Run တဲ့ကိစ္စတွေ၊ JavaScript Code တွေကို Lint လုပ်တဲ့ကိစ္စတွေ၊ Image Optimize လုပ်တဲ့ ကိစ္စတွေနဲ့ Build လုပ်ငန်းစဉ်မှာလိုအပ်တဲ့ အခြားလုပ်ငန်းတွေကို Gruntfile.js ထဲမှာ ကြော်လုပ်လိုတိုင်း အမြဲတောင် Project ကို Build လုပ်လိုတိုင်း grunt Command တစ်ကြိမ် Run ယုံနဲ့ လိုအပ်တဲ့လုပ်ငန်းစဉ် တစ်ခုလုံး ပြီးမြောက်သွားမှာပဲ ဖြစ်ပါတယ်။ အသုံးဝင်တဲ့ Grunt Plugin တွေကို အောက်ပါလိပ်စာမှာ လေ့လာနိုင်ပါတယ်။

<http://gruntjs.com/plugins>

Yeoman နဲ့ Project တစ်ခုကို Scaffold လုပ်လိုက်တိုင်းမှာ Yeoman က Build Configuration အသင့် ပါဝင်တဲ့ Gruntfile.js ကို တစ်ခါတည်း တည်ဆောက်ပေးသွားလေ့ရှိပါတယ်။ လေ့လာလိုရင် yo webapp Command နဲ့ Web App Boilerplate တစ်ခု Scaffold လုပ်ကြည့်လိုက်ပါ။ ပြီးရင် သူနဲ့အတူပါလာတဲ့ Gruntfile.js ကို လေ့လာကြည့်ရင် Build Process တစ်ခုလုံးအတွက် လိုအပ်တဲ့လုပ်ငန်းအားလုံးကို ကြိုက်တင် Configure လုပ်ပေးထားတယ် ဆိတဲ့တာကို တွေ့ရမှာပဲ ဖြစ်ပါတယ်။

ပုံ (၈.၁) မှာ လေ့လာကြည့်ပါ။ Yeoman, Bower နဲ့ Grunt တို့ကို အသုံးပြုထားတဲ့ Common Workflow ကိုဖော်ပြထားပါတယ်။



ပုံ (၈.၁) - Web App Development Workflow with Yeoman, Bower and Grunt

Source – [yeoman.io](http://yeoman.io)

ပထမဗြိုးဆုံး yo Command ကနေတစ်ခု App Boilerplate ကို တည်ဆောက်ယူရပါတယ်။ ဒါ က အသုံးဝင်တဲ့ Bower Package တွေကိုပါ တစ်ခါတည်းထည့်သွင်းပေးသွားမှာ ဖြစ်သလို Gruntfile.js ကိုလည်း တစ်ခါတည်း တည်ဆောက် ပေးသွားမှာ ဖြစ်ပါတယ်။ ပြီးတဲ့အခါ ဖြည့်စွက်လိုအပ်တဲ့ Package တွေကို bower install နဲ့ ထပ်မံ ဖြည့်စွက်နိုင်ပါတယ်။ ရေးစရာရှိတဲ့ Code တွေရေးပြီးလို့ App ကို စမ်းကြည့်ချင်တယ်ဆို

ရင် grunt serve Command ကို Run ပြီ စမ်းကြည့်နိုင်ပါတယ်။ Build မလုပ်သေးပဲ Test Code တွေကို Run ခေါ်ပေါ်ရင် grunt test Command နဲ့ Test Code တွေ Run နိုင်ပါတယ်။ ဒီအတွက် လိုအပ်တဲ့ Configuration တွေကို ၁၀ က တစ်ခါ တည့်ပေးလိုက်ပြီးသားပါ။ နောက်ခုံး Build လုပ်ချင်တယ်ဆိုရင်တော့ grunt Command ကို Run လိုက်ယုံ ပါပဲ။ JavaScript Lint လုပ်တာတွေ၊ Minify လုပ်တာတွေ၊ Image Optimize လုပ်တာတွေ အားလုံးကို လုပ်ပေးသွားတဲ့ အပြင် ရလာတဲ့ Build Result ရလဒ်ကိုလည်း အသင့်သုံးနိုင်အောင် dist Directory ထဲမှာ ထားပေးမှာပဲဖြစ်ပါတယ်။ ဒါကြောင့် grunt Command Run ပြီးတဲ့နောက် dist Directory ထဲမှာရှိနေတဲ့ Build Result ကိုလက်ရှိ Develop လုပ်နေတဲ့ App ၏ Production Version အဖြစ်မှတ်ယူပြီး လိုအပ်သလို ဆက်လက်ဆောင်ရွက်နိုင်မှာပဲ ဖြစ်ပါတယ်။

## 8.7 – Deployment

Software တစ်ခုကို Deploy လုပ်ရာမှာ အခြေအနေစုစုပေါင်များရှိနိုင်ပါတယ်။ အကယ်၍ Software က အဖွဲ့အစည်း တစ်ခု အတွင်းမှာသုံးဖို့ ရည်ရွယ်တည်ဆောက်ထားတဲ့ Software ဖြစ်တယ်ဆိုရင်၊ Deploy မလုပ်ခင် အဲဒီအဖွဲ့အစည်းမှာ ဒီ Software ကို အသုံးပြုဖို့အတွက် လိုအပ်တဲ့ Infrastructure တစ်ခုရှိရှုလား စမ်းစစ်ဖို့ လိုနိုင်ပါတယ်။ Server, Network နဲ့ System Requirement တွေ ပြည့်စုစုပေါင်မှုံးစိစစ်ပြီး၊ လိုအပ်ရင် Infrastructure Architecture Design ကို ပြင်ဆင် တည်ဆောက်ဖို့လိုပါတယ်။ ကိုယ်တိုင် တည်ဆောက်ရတာမျိုး ဖြစ်နိုင်သလို Enterprise IT Architecture ပိုင်း ဝန်ဆောင် မှုပေးသွေ့ရေးအကူအညီကို ယူရတာမျိုးလဲ ဖြစ်နိုင်ပါတယ်။

အကယ်၍ Software က အဖွဲ့အစည်း တစ်ခု အတွက် ရည်ရွယ်တဲ့လုပ်ငန်းသုံး Software မဟုတ်ပဲ၊ အများသုံး Consumer Software ဆိုရင်တော့ Build လုပ်ပြီးရလာတဲ့ ရလဒ်ကို User တွေထံရောက်ရှုအောင် ဖြန့်ချုပ်ရမယ့် နည်းလမ်းတွေ စဉ်းစားရပါတော့မယ်။ အသုံးများတဲ့ Distribution Method တစ်ခုဖြစ်တဲ့ App Store တွေမှာ တင်ပြီး ဖြန့်ချုပ်ရာလား၊ ကိုယ်ပိုင် Website တစ်ခုနဲ့ Download ပေးမှာလား စသဖြင့် စဉ်းစားရပါမယ်။ Software အများစုက API Back-end ကို အခြေခံကြတဲ့အတွက် Server Environment တစ်ခုလဲ လိုအပ်ပါတယ်။ အဲဒီလို လိုအပ်တဲ့ Server Environment ရရှိဖို့အတွက် Cloud Service Provider တွေထံက လိုအပ်တဲ့ ဝန်ဆောင်မှုတွေ ကို ရယူနိုင်သလို၊ ကိုယ်တိုင်လည်း Production Server တစ်ခုကို Setup လုပ်ယူနိုင်ပါတယ်။

လိုဂ်းကတော့ Deployment & Distribution ဆိုတာဟာ Business Process တစ်ခု Business Decision တစ်ခု ဖြစ်ပါတယ်။ Software Development တစ်ဘက်တည်းကမဟုတ်ပဲ ရှုထောင့်စုံကနေ စဉ်းစားဆုံးဖြတ်ရမယ့် ကိစ္စရပ်တစ်ခု ဖြစ်ပါတယ်။

Business Decision ပိုင်းကိုချုပ်ပြီး Developer တွေနဲ့ သက်ဆိုင်တဲ့အပိုင်းကို ယေဘုယျခြေကြည့်လိုက်ရင် နှစ်ပိုင်း ရှုပါတယ်။ လိုအပ်တဲ့ Server နဲ့ Network Environment ပိုင်း Configure ပြုလုပ်ခြင်းတဲ့ Build လုပ်ပြီး ရရှိလာတဲ့ ရလဒ် ကို အများအသုံးပြုနိုင်ဖို့အတွက် Production Server မှာ Publish လုပ်ခြင်းတို့ပဲဖြစ်ပါတယ်။

Server နဲ့ Network Environment ပိုင်း Configure ပြုလုပ်ခြင်းဟာ အမျိုးကြောင်း အထူးပြုကြောင်းကျင်တဲ့ sysadmin လိုခေါ်တဲ့ System Administrator တွေရဲ့အလုပ်ဖြစ်ပါတယ်။ ဒါပေမယ့်လည်း အခုနောက်ပိုင်းမှာ Full-stack Developer ဆိုတဲ့အသုံးအနှံန်းနဲ့အတူ။ Software ကို တည်ဆောက်နိုင်ယုံသာမက လိုအပ်တဲ့ Server Environment ကိုမှ Configure လုပ်နိုင်တဲ့ ဘက်စုံသုံး Developer တွေ ပေါ်ပေါက်လာခြင်းနဲ့အတူ။ Server

Environment Configuration ဟာလည်း Developer တွေရဲ့ အလုပ်တစ်ခုဖြစ်လာခဲ့ပါတယ်။ ဒီကဏ္ဍမှာ ဖြေစရာတွေ များတဲ့ အတွက် ဒီနေရာမှာ မပြောတော့ပဲ၊ အပိုင်း (၅) ရောက်တော့မှ ဆက်လက်ဖော်ပြပါမယ်။

ဒုတိယတစ်ပိုင်းဖြစ်တဲ့ အသင့် Setup လုပ်ထားပြီးဖြစ်တဲ့ Server Environment မှာ Build ထားတဲ့ Software ကို Publish လုပ်မယ်ဆိုရင်တော့၊ အလွယ်ဆုံးနည်းလမ်းအနေနဲ့ FTP Software တစ်ခုကိုသုံးပြီး Build လုပ်ထားတဲ့ Software ကို Production Server ပေါ်မှာ Copy ကူးပေးလိုက်နိုင်ပါတယ်။ ဒီနည်းက လက်တွေမှာ ပြသနာတစ်ချို့ ပေးတက်ပါတယ်။

၁.) Software မှာပါဝင်တဲ့ ဖိုင်အရေအတွက် အလွန် များပြားတဲ့ အခါ အင်တာနာက်အဆက်အသွယ် ပြတ်တောက် မှုနဲ့ မမျှော်မှန်းနိုင်တဲ့ Connection ပြသနာတစ်ချို့ကြောင့် တစ်ချို့ဖိုင်တွေ မတော်တစ်ဆ ကျွန်းသွားတာမျိုး ဖြစ်နိုင်ပါတယ်။

၂.) Deploy လုပ်တယ်ဆိုတာ ထပ်ခါထပ်ခါလုပ်ရတက်ပါတယ်။ ဥပမာ Version 0.1.0 ကိုတစ်ကြိမ် Deploy လုပ်ပြီး၊ နောက်တော့မှ Version 0.1.1 ထွက်လာလို့ နောက်တစ်ကြိမ် Deploy လုပ်ရတက်ပါတယ်။ အဲဒီအခါမှာ 0.1.0 နဲ့ 0.1.1 အကြား ပြောင်းလဲသွားတဲ့ အပြောင်းအလဲကိုသာမက၊ သေချာအောင် Software တစ်ခုလုံးကို နောက်တစ်ကြိမ် ပြန်ပြီး ကူးတင်ရတက်ပါတယ်။

၃.) လက်တွေမှာ တစ်ချို့ဖိုင်တွေကို Compress လုပ်လိုက်ရင် ဖိုင်အမျိုးအစားနဲ့ Compression Algorithm ပေါ်မှုတည် ပြီး ဖိုင် Size က 200% ကနေ 400% လာက်ထိကျွန်းနိုင်ပါတယ်။ Deployment လုပ်ငန်းမြန်ဆန်ဖို့ဆိုရင် Compress လုပ်ပြီးမှ Server ထံပေးပို့သင့်ပါတယ်။ ဒီအခါမှာ Copy မကူးခင် Compress လုပ်ရတဲ့ ကိစ္စတွေ၊ Copy ကူးပြီးနောက် Decompress လုပ်ရတဲ့ ကိစ္စတွေ ထပ်လိုလာတက်ပါတယ်။

၄.) ဒီလို Manual Deploy လုပ်ရတဲ့ အတွက် နောက်ဆက်တဲ့ ပါလာတဲ့ ပြသနာတစ်ခုလည်း ရှိပါသေးတယ်။ Deploy လုပ်ပြီးမှ အသေးစားပြင်စရာလေးတစ်ခုတွေလာတဲ့ အခါ ကိုယ့်စက်ထဲက မူလ Source Code မှာပြင်ပြီး၊ နောက်တစ်ကြိမ် Deploy လုပ်နေရင်ကြောမှာထိုးလို့ဆိုပြီး Production Server ပေါ်မှာ တိုက်ရိုက်ပြင်လိုက်မိတက်ပါတယ်။ ဒီလို ပြင်ဆင်ချက်လေးတွေ များများလာပြီး၊ အဲဒီပြင်ဆင်ချက်တွေကို Local Code Base နဲ့ ညီညွှန်းဖို့ နောက်ကျတဲ့ အခါ၊ Production Version က Development Version ထက်ရှုံးရောက်နေတဲ့ ပြသနာတွေ ကြောက်ပါတယ်။

Deployment ဆိုတာ Build လုပ်ထားတဲ့ Software ကို Server ပေါ်ကူးလိုက်တဲ့ ကိစ္စလေးပါလို့ ခပ်ပေါ့ပေါ့သဘောထားမိ ရင် ဒီလိုပြသနာတွေ အနည်းနဲ့ အများရှိလာပြီး၊ မလိုအပ်ပေါ်အချိန်တွေပို့ကုန်သွားတက်ပါတယ်။ ဒါကိုဖြေရှင်းဖို့ အတွက် Git ရဲ့ အကူအညီကိုရယူနိုင်ပါတယ်။ ဖြစ်သင့်တဲ့ Workflow က ဒီလိုပါ။

၁.) ပထမဆုံးအနေနဲ့ Production Server ပေါ်မှာ git init --bare နဲ့ Git Repo အလွတ်တစ်ခု သတ်မှတ် ပေးရပါမယ်။

၂.) ကိုယ့် Local Git Repo ရဲ့ Remote အဖြစ် အဲဒီ Production Server ပေါ်က Git Repo ကို ညွှန်းပေးရပါမယ်။

```
$ git remote add prod git://path/to/production/repo
```

၃.) Local Code Base ထဲမှာ Build လုပ်လိုက်လို့ dist Directory ရလာပြီးဆိုရင် အဲဒီ dist Directory ကို git push နဲ့ Production Server ပေါ်ကို ပိုပေးရပါမယ်။ Code Base ကြီးတစ်ခုလုံးကို Push မလုပ်ပဲ dist Directory တစ်ခုတည်းကို ရွေ့ပြီး Push လုပ်လိုရင် Git Sub Tree ဆိုတဲ့ Git Plugin ကို အသုံးပြုနိုင်ပါတယ်။ အောက်ပါလိပ်စာ မှာ ရယူနိုင်ပါတယ် –

<https://github.com/apenwarr/git-subtree>

၄.) Git Sub Tree ကို Install လုပ်ဖို့အတွက် သူနဲ့အတူပါလာတဲ့ လမ်းညွှန်ကိုလေ့လာကြည့်လိုက်ပါ။ Git Sub Tree ကို Install လုပ်ပြီးနောက် dist Directory ကို အောက်ပါအတိုင်း Production Server ထံ Push လုပ်နိုင်ပါတယ်။

```
$ git subtree push --prefix dist prod master
```

--prefix Option အတူ Push လုပ်လိုတဲ့ Directory အမည်ဖြစ်တဲ့ dist ကို တွဲသုံးထားတာကို သတိပြုပါ။ ဒါနည်းနဲ့ Code Base တစ်ခုလုံးကို Push မလုပ်ပဲ dist Directory တစ်ခုတည်းကို Push လုပ်နိုင်ပါတယ်။

ဒီနည်းနဲ့ Deployment ပြုလုပ်လိုတိုင်း git subtree push ကိုသုံးလိုက်ခြင်းအားဖြင့် Build လုပ်ထားတဲ့ Software လည်း Server ထံ ရောက်ရှိသွားမှာပဲ ဖြစ်ပါတယ်။ Git က ဖိုင်တွေကို Compress လုပ်တဲ့ကိစ္စတွေ၊ ဖိုင်မကျွန်းခဲ့အောင်စီစဉ်တဲ့ကိစ္စတွေနဲ့၊ နောင်ပိုင်း Push တွေမှာ Software တစ်ခုလုံးကို ထပ်မပို့တော့ပဲ၊ အပြောင်းအလဲကိုသာ ရွေးချယ်ပေးပို့သွားတဲ့ ကိစ္စတွေအားလုံးကို တာဝန်ယူပေးသွားမှာပဲ ဖြစ်ပါတယ်။ အမှားပြင်စရာရှိရင်လည်း Production Server ပေါ်မှာ ပြင်စရာမလိုပဲ ကိုယ့် Local Code Base ထဲမှာပြင်ပြီး git subtree push နောက်တစ်ကြိမ် Run ပေးလိုက်ယုံပဲဖြစ်ပါတယ်။

## 8.8 – Continues Integration

Development Workflow Automation နဲ့ပက်သက်ပြီး နောက်ဆုံးတစ်ချက်အနေနဲ့ မှတ်သားသင့်တဲ့ နည်းစနစ်ကတော့ Continues Integration (CI) ပဲဖြစ်ပါတယ်။ Continues Integration ဆိုတာကို လိုရင်းပြောရရင် နေစဉ် Build လုပ်တဲ့ Daily Build အဆင့်ကိုကျော်ပြီး၊ VCS ထဲကို Commit တစ်ကြိမ်လုပ်တိုင်း အလိုအလျောက် Build လုပ်သွားအောင် စီမံထားရတဲ့ နည်းစနစ်ဖြစ်ပါတယ်။

ဒီနည်းက ပေးတဲ့အဓိက အကျိုးကျေးဇူးနှစ်ခုရှိပါတယ်။ တစ်ခုကတော့၊ Commit တစ်ခုကြောင့် Build ပြသုနာ ရှိခဲ့ရင် ချက်ခြင်းသိရတဲ့အတွက် ပြင်ဆင်ရလွယ်ကူခြင်းဖြစ်ပါတယ်။ Developer တစ်ယောက်ပြင်ထားတဲ့ Code က သူတာဝန်ယူ တဲ့ အပိုင်းများ မှာမှန်နေလို့ Commit လုပ်လိုက်ပေမယ့် အဲဒီ Commit ကြောင့် အခြား Developer တာဝန်ယူနေတဲ့ အပိုင်းမှာသွားပြီး ပြသုနာဖြစ်နေတယ် ဆိတ်မျိုး ဖြစ်တက်ပါတယ်။ Commit တိုင်းမှာ Build လုပ်သွားတဲ့အတွက် ဒီ ပြသုနာမျိုးရှိခဲ့ရင် ချက်ခြင်းသိနိုင်မှာဖြစ်ပါတယ်။

နောက်ထပ်ရရှိတဲ့ အကျိုးကျေးဇူးကတော့ Software ဟာ အမြတမ်း Production Ready ဖြစ်နေခြင်းပဲဖြစ်ပါတယ်။ Build လုပ်တယ်ဆိတ် လက်ရှိရေးသားနေတဲ့ Code ကို Production Code ပြောင်းတဲ့လုပ်ငန်းဖြစ်ပါတယ်။ Commit တိုင်းကို Build လုပ်နေတဲ့အတွက် ရောက်ရှိနေတဲ့နောက်ဆုံးအဆင့်ဟာ အမြတမ်း Production သွားဖို့ အသင့်ဖြစ်နေမှာ ပဲဖြစ်ပါတယ်။

ဒီနည်းဟာ Agile နည်းစနစ်တစ်ခုဖြစ်တဲ့ Extreme Programming (XP) ကလာတာပါ။ Continues Integration လုပ် နိုင်ဖို့အတွက် Travis CI ([travis-ci.org](https://travis-ci.org)), BuildBot ([buildbot.net](https://buildbot.net)), Hudson CI ([hudson-ci.org](https://hudson-ci.org)) စတဲ့ Tool တွေရှိ ပါတယ်။ အဲဒီ CI Tool တွေရဲ့ အခြေခံ အလုပ်လုပ်ပုံက ဒီလိုပါ -

ပထမဦးဆုံးအနေနဲ့ Build Configuration ဖိုင်တစ်ခုကို အရင် Setup လုပ်ရပါတယ်။ အဲဒီ Configuration တဲ့မှာ လက်ရှိ Software အလုပ်လုပ်ဖို့ ရည်ရွယ်ထားတဲ့ Environment Information အပြည့်အစုံသတ်မှတ်ပေးရတာက် ပါတယ်။ Compiler, Compiler Version, အသုံးပြုထားတဲ့ Component တွေနဲ့ Package တွေစာရင်းကိုသတ်မှတ်ပေးရပါတယ်။ ပြီးတဲ့အခါ Build လုပ်ငန်းစဉ်ဆောင်ရွက်ဖို့အတွက် Production Environment မှာ ရှိရမယ့် အခြေအနေအတိုင်း Build Environment တစ်ခုကို Setup လုပ်ရပါတယ်။ Build လုပ်တဲ့အခါ အဲဒီ Build Environment ပေါ်မှာ လုပ်မှာဖြစ်ပါတယ်။ ဒါကြောင့် Build Script တဲ့မှာ အသုံးပြုထားတဲ့ Compiler Version တွေ၊ အသုံးပြုထားတဲ့ Package တွေကိုဖော်ပြရ ခြင်းဖြစ်ပါတယ်။ Build Environment ကို Setup လုပ်ဖို့ အတွက် အဲဒီအချက်အလက်တွေကို အသုံးပြုသွားမှာဖြစ်ပါတယ်။ Travis CI မှာ အသုံးပြုရလေ့ရှိတဲ့ Build Configuration နမူနာကို ဖော်ပြလိုက်ပါတယ်။

## YAML

```
# File Name: .travis.yml

language: node_js
node_js:
  - 0.10
  - 0.8
  - 0.6

services:
  - mongodb

before_script:
  - npm install
  - bower install

script: npm test
```

```

notifications:
  email:
    - "james@gmail.com"
    - "john@gmail.com"
  
```

Travis CI Build Configuration ကို YAML လိုခေါ်တဲ့ Markup ရေးထံးနဲ့ရေးပေးရပါတယ်။ နမူနာအရ Build Environment ကို NodeJS 0.10, 0.8 နဲ့ 0.6 Version သုံးမျိုးအတွက် Setup လုပ်ပေးသွားမှာဖြစ်ပါတယ်။ အဲဒီလို Setup လုပ်ရာမှာ MongoDB ကို တွဲဖက်ထည့်သွင်းပေးသွားမှာဖြစ်ပြီး Build Script ကို Run မလုပ်ခင် npm install နဲ့ bower install တို့ကို အရင် Run ပေးသွားနိုးမှာဖြစ်ပါတယ်။ Build Script အနေနဲ့ npm test ကို အသုံးပြုဖို့ သတ်မှတ်ထားပါတယ်။ အကယ်၍ Gruntfile.js ကို Build Script အနေနဲ့ သုံးစေချင်ရင် NPM Configuration ဖိုင်ဖြစ်တဲ့ package.json မှာ အခုလိုထည့်သွင်းပေးထားနိုင်ပါ တယ်။

### JSON

```

// File Name: package.json

scripts {
  "test": "grunt"
}
  
```

ဆက်လက်ပြီး Source Code Repo မှာ Commit တစ်ကြိမ်လုပ်တိုင်း Code Base ကို Setup လုပ်ထားတဲ့ သီးခြားစနစ်ပေါ်မှုများယူခြင်းနဲ့ Build Script ကို Run ပေးခြင်းလုပ်ငန်းတွေ အလိုအလျောက် ဆောင်ရွက်ဖို့ သတ်မှတ်ပေးရပါ တယ်။ Travis CI ကတေသာ့ Github ရဲ့ Web Hook ကို သုံးပါတယ်။ ဒါကြောင့် Github မှာ Host လုပ်ထားတဲ့ Repo မှာ Commit လုပ်လိုက်တိုင်း၊ Github က Travis CI ကို အသိပေးသွားမှာဖြစ်ပြီး၊ Travis CI က Build Environment တည်ဆောက်ခြင်းနဲ့ Build Script Run ခြင်းတို့ကို Commit တိုင်းအတွက် လုပ်ပေးသွားမှာဖြစ်ပါတယ်။

Build Script ကို Run ပြီးတဲ့အခါ Fail ဖြစ်ခဲ့ရင် အခြေအနေကို Developer အားလုံးထံး Error Log နဲ့ တစ်ကွ ပေးပို့ ပေးရမှာဖြစ်ပါတယ်။ Pass ဖြစ်ခဲ့ရင်တော့ ရရှိလာတဲ့ Build Result ကို Production Environment ထံ ပေးပို့ခြင်း အားဖြင့် Deployment ကိုပါ တစ်ခါတည်း တွဲလုပ်နိုင်ပါတယ်။

Travis CI Blog Status Help Ei Maung

Search all repositories

My Repositories

- eimg/jquery-voter
- eimg/tinymvc Duration: 17 sec Finished: 9 minutes ago

Current Branches Build History Pull Requests Settings

master Create .travis.yml

# 1 failed Commit 8cf8677 Compare 5a1950c..8cf8677 ran for 17 sec 9 minutes ago

Ei Maung authored and committed

Build Jobs

#	PHP Version	Environment Variables	Duration
1.1	PHP: 5.3	no environment variables set	7 sec
1.2	PHP: 5.4	no environment variables set	6 sec
1.3	PHP: 5.5	no environment variables set	4 sec

### ပုံ (၈.၂) - Travis CI - Build Status

ပုံ (၈.၂) မှာဖော်ပြထားတာကတော့ Build Fail ဖြစ်တဲ့အခါ Travis CI ကဖော်ပြပေးတဲ့ အခြေအနေဖြစ်ပါတယ်။ Build Pass ဖြစ်ခဲ့ရင်တော့ Amazon AWS, Google Cloud Storage, Heroku, Engine Yard, Appfog စတဲ့ Cloud Service တွေပေါ်မှာအလိုအလျောက် Deploy လုပ်သွားအောင် စီမံထားနိုင်ပါသေးတယ်။ အကယ်၍ Git Push နဲ့ Deploy လုပ်စေလိုရင်လည်း Build Configuration ထဲမှာ အခုလို ထည့်သွင်းရေးသားထားနိုင်ပါတယ်။

#### YAML

```
# File Name: .travis.yml

after_success:
- git remote add deploy git:/path/to/repo/
- git push deploy
```

ဒီနည်းနဲ့ VCS ထဲမှာ Commit လုပ်လိုက်တိုင်း၊ အလိုအလျောက် Build လည်းလုပ်၊ Deploy လည်း လုပ်ပေးသွားအောင် စီမံထားခြင်းအားဖြင့် Continuous Integration လုပ်ဆောင်ချက်ကို ရရှိနိုင်မှာပဲဖြစ်ပါတယ်။

## Conclusion

ဒီအခန်းမှာ နမူနာအဖြစ်ဖော်ပြခဲ့တဲ့ Tool တွေဟာ Web App Project တွေအတွက် ရည်ရွယ်ဖန်တီးထားတဲ့ Tool တွေ များပါတယ်။ အခြား Software Project အမျိုးအစားတွေအတွက်လည်း၊ အသုံးပြုထားတဲ့ Language, Framework နဲ့ နည်းပညာတွေ ပေါ်မှုတည်ပြီး Scaffolding, Package Management, Build Tool နဲ့ CI Tool တွေ သူနောရာနဲ့သူ ရှိတက်ပါတယ်။

Ruby on Rails, Django, ASP.NET MVC, Laravel, Symfony, CakePHP စတဲ့ Framework တွေမှာ Scaffolding လုပ်ဆောင်ချက် တဲ့ဖက်ပါဝင်လာပြီးဖြစ်ပါတယ်။ Java Project တွေအတွက်ဆိုရင် Spring Roo လို Tool မျိုးကို Scaffolding အတွက် အသုံးပြုနိုင်ပါတယ်။ ဒုံအပြင် NetBeans, Eclipse, Visual Studio စတဲ့ IDE တွေမှာလည်း

Scaffold နဲ့ အလားသဏ္ဌာန်တူတဲ့ Code Generation လုပ်ဆောင်ချက်တွေ ပါဝင်တက်ပါတယ်။ Plugin Extension အနေနဲ့ ထပ်မံထည့်သွင်းပေးဖို့တော့ လိုကောင်းလိုနိုင်ပါတယ်။

Package Manager အနေနဲ့ဆိုရင်လည်း Microsoft .NET အတွက် NuGet, PHP အတွက် Composer နဲ့ Pear, Python အတွက် pip, Ruby အတွက် Gem နဲ့ Bundler, Java အတွက် Maven စသဖြင့် ကိုယ်စိန္တိကြပါတယ်။ Build Tool အနေနဲ့ဆိုရင်တော့ Ruby အတွက် Rake, Java အတွက် Apache Ant, .Net အတွက် NAnt နဲ့ MSBuild, PHP အတွက် Phing, Python အတွက် SCons နဲ့ Paver စသဖြင့်ကိုယ်စိန္တိကြပါတယ်။ တစ်ချို့ Tool တွေက Command Line Tool တွေ ဖြစ်ပြီး တစ်ချို့ကိုတော့ IDE တွေနဲ့အတူ တွဲဖက်အသုံးပြုလိုရတက်ကြပါတယ်။

(၁၀) ဆအလုပ်ပို့တွင်တဲ့ Rockstar Developer တစ်ဦးအနေနဲ့ ဒီလို Workflow Automation Tool တွေကို လေ့လာပြီး ထိရောက်အောင် အသုံးချတ်ကြပ်နောင်းဟာ လိုအပ်တဲ့အရည်အချင်းတစ်ရပ်ပဲဖြစ်ပါတယ်။

# အပိုင်း (၃)

Modern Development Stack

ပြုပြင်ထိမ်းသိမ်းရလွယ်ကူတဲ့ Code ဖြစ်ဖို့အတွက်  
စနစ်ကျေတဲ့ Code Structure Architecture တစ်ခုကို  
လိုက်နာအသုံးပြုဖို့လိုပါတယ်။

**Ubuntu - သင့်အတွက် Linux (စာအုပ်)**  
Ubuntu Linux အသုံးပြန်လိုးအား Installation မှ စတင်၍ System Configuration နှင့် Development Environment တည်ဆောက်  
ခြင်းအထိ အဆင့်လိုက်ရေးသား ဖော်ပြထားသည့်စာအုပ်  
အောက်ပါလိပ်စာတွင် အခမဲ့ ရယူနိုင်သည်။

## အခန်း(၉) – BackboneJS (Client-Side MVC)

**အခန်း (၃)** မှာ Service Oriented Architecture (SOA) အကြောင်း ဖော်ပြခဲ့စဉ်က Software တစ်ခုကို တည်ဆောက်ရာမှာ သီးခြားစီအလုပ်လုပ်နိုင်ပြီး၊ အပြန်အလုန်လည်း ဆက်သွယ်ဆောင်ရွက်နိုင်တဲ့ ပရိုဂရမ်းယေား အဖြစ် တည်ဆောက်ခြင်းကနေ ရရှိနိုင်တဲ့ အကျိုးကျေးဇူးတွေကို ဖော်ပြခဲ့ပြီးဖြစ်ပါတယ်။ Mobile Device တွေ အသုံးတွင်ကျယ်လာပြီး Mobile ခေတ်ဖြစ်လာတဲ့ ကနောက်ချိန်မှာ Software Application တစ်ခုကို One Core / Multiple Clients ပုံစံ တည်ဆောက်ခြင်းနည်းဟာ တွင်ကျယ်လာတဲ့ နည်းစနစ်တစ်ခု ဖြစ်လာခဲ့ပါတယ်။ App ရဲ့ Core Logic ကို သီးခြားအလုပ်လုပ် နိုင်တဲ့ Service တစ်ခုအနေနဲ့တည် ဆောက်လိုက်ခြင်းအားဖြင့် Device တစ်မီးအတွက် App တစ်ခုစီ သီးခြားထပ်မံတည် ဆောက်နေဖို့မလိုတော့ပဲ၊ UI Service တွေကိုသာ ထပ်မံတည်ဆောက်ဖို့ လိုတဲ့သော ဖြစ်ပါတယ်။

Web Browse အတွင်းမှာ အလုပ်လုပ်တဲ့ Web App တွေဟာလည်း အမိက App Interface အဖြစ်ကနေ Client များစွာထဲက Client တစ်ခုအဖြစ်ကို ရောက်ရှိလာခဲ့ကြပါတယ်။ အရင်က Web Application ဆိုတာ၊ လုပ်ဆောင်ချက်အားလုံးကို Server ပေါ်မှာပဲဆောင်ရွက်ပါတယ်။ အမိက Core Logic တွေကို Process လုပ်တဲ့ အလုပ်တွေ၊ Data အချက် အလက်တွေ သိမ်းဆည်း ရယူတဲ့ လုပ်ငန်းတွေနဲ့ UI Template တွေ ဖန်တီးတဲ့ လုပ်ငန်းတွေ၊ အားလုံးကို Server ပေါ်မှာပဲ လုပ်ပြီး၊ Client ဖြစ်တဲ့ Browser က၊ User Input တွေကို လက်ခံတဲ့ အလုပ်နဲ့၊ Server ပေါ်မှာ အလုပ်လုပ်ပြီး နောက်ဆုံး ရလာတဲ့ ရလဒ်ကို ဖော်ပြပေးတဲ့ အလုပ်ကိုသာ လုပ်ပါတယ်။ တစ်နည်းအားဖြင့် Service တစ်ခုအနေနဲ့ သီးခြားအလုပ် လုပ်တာမျိုးမဟုတ်ပဲ Server ပေါ်မှာ မို့ခို့ အလုပ်လုပ်ကြခြင်းဖြစ်ပါတယ်။ JavaScript ဆိုတာလည်း User Interface ကို ပြည့်စုံအောင် ဖြည့်စွက်တဲ့ ပြည့်စွက်နည်းပညာ တစ်ခုသာဖြစ်ခဲ့ပါတယ်။

နောက်ပိုင်းမှာတော့ Web App တွေလည်း Server ကို မမိုခိုတော့ပဲ သီးခြားအလုပ်လုပ်နိုင်တဲ့ Service များအဖြစ် တည်ဆောက်လာကြပါတယ်။ Server ကို Core Service တစ်ခုအနေနဲ့ လိုအပ်သလို ဆက်သွယ်ဆောင်ရွက်ပေးမယ့်၊ အမိက လုပ်ငန်းအများစုကို Server ကို မိုခိုခြင်းမပြုတော့ပဲ ကိုယ်တိုင်ရပ်တည်နိုင်တဲ့ သီးခြားပရိုဂရမ် အဖြစ် တည်ဆောက်လာကြ ခြင်းဖြစ်ပါတယ်။ JavaScript ဆိုတာလည်း အရင်လို့ User Interface ပိုင်းကို ဖြည့်စွက်ပေးတဲ့ Language သက်သက် မဟုတ်တော့ပဲ Client Service တစ်ခုလုံးကို တည်ဆောက်ရာမှာ အသုံးပြုတဲ့ Programming Language တစ်ခုဖြစ်လာပါ တယ်။ တနည်းအားဖြင့် JavaScript ဆိုတာ မူလက App ရဲ့ UI ပိုင်းကိုသာ စီမံဖို့သုံးခဲ့ပေမယ့်၊ ကနောက်ချိန်မှာတော့ UI ရော Data ကိုပါ စီမံဖို့အသုံးပြုလာကြခြင်းပဲ ဖြစ်ပါတယ်။

Software Development မှာ ရေရှည် Maintenance ကောင်းစေဖို့အတွက် Data ပိုင်းလုပ်ဆောင်ချက်နဲ့ UI ပိုင်းလုပ် ဆောင်ချက်တို့ကို ခွဲခြားရေးသားကြလေ့ရှိပါတယ်။ ဒီတော့မှ Data နဲ့ UI ကို သီးခြားစီမံချွဲပြီး Maintain လုပ်နိုင်တဲ့အတွက် အများနည်းပြီး ပြင်ဆင်ထိမ်းသိမ်းရ လွယ်ကူစေမှာဖြစ်ပါတယ်။ ဒီနေရာမှာ အသုံးများတဲ့

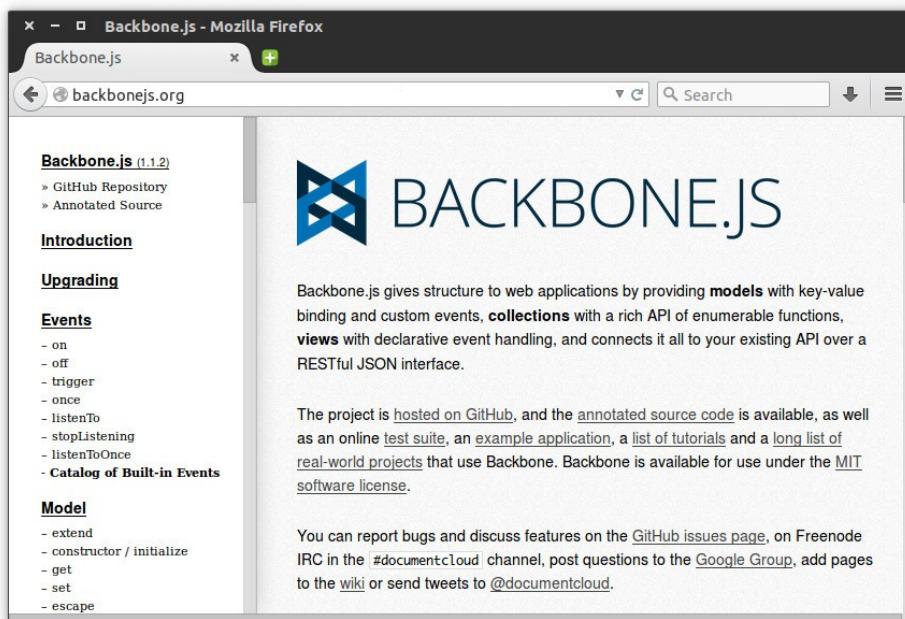
နည်းစနစ်ကတော့ MVC လို့ ခေါ်တဲ့ Model-View-Controller Design Pattern ပဲဖြစ်ပါတယ်။ MVC အကြောင်းကို Professional Web Developer ရဲ့ အခန်း (၁၃) မှာ ဖော်ပြထားပါတယ်။

BackboneJS ဟာ JavaScript Code တွေကို MVC Pattern နဲ့ ရေးသားနိုင်ဖို့ အကူအညီပေးတဲ့ MVC Framework တစ်ခုဖြစ်ပါတယ်။ မူရင်း MVC ရဲ့ အမိပိုယ်က Model-View-Controller ဖြစ်ပါတယ်။ အခြေခံအားဖြင့် Model က Data အချက် အလက်စီမံမှုနဲ့ ပက်သက်တဲ့ အပိုင်းကို တာဝန်ယူပါတယ်။ View ကတော့ User Interface ဖော်ပြနှုန်းကို တာဝန်ယူပါတယ်။ Controller ကတော့ User Input တွေ စီမံမှုအပိုင်းကို တာဝန်ယူရပါတယ်။ Code ရေးသားတဲ့ အခါ Data အချက်အလက် စီမံမှုအပိုင်းတွေကို Model အနေနဲ့ သီးခြားခွဲခြားရေးသားရပြီး၊ UI ဖော်ပြနှုန်းကို Model Code နဲ့ မရောပဲ View အနေနဲ့ သီးခြားရေးသားခြင်းဖြစ်ပါတယ်။ ဘယ်လို့ User Input လက်ခံရရှိရင် ဘာအလုပ်လုပ်ရ မလဲဆိတဲ့ ကိစ္စကို စီမံပေးတဲ့ Code ကို တော့ Model ရော View နဲ့ပါ မရောပဲ Controller အနေနဲ့ သီးခြားရေးသားရ ပါတယ်။

BackboneJS မှာတော့ MVC ဆိုတာ Model-View-Collection ဆိုတဲ့ အမိပိုယ်ဖြစ်ပါတယ်။ Data အချက်အလက် စီမံမှုပိုင်းဖြစ်တဲ့ Model နဲ့ UI ဖော်ပြနှုန်းဖြစ်တဲ့ View တို့ကို ခွဲခြားရေးသားနိုင်အောင် အကူအညီပေးပါတယ်။ Collection ဆိုတာကတော့ Model ကို ထောက်ပံ့ဖြည့်စွက်ပေးတဲ့ Model Collection ဖြစ်ပါတယ်။ Controller သဘော မျိုး သီးခြားလုပ်ဆောင်ချက်မပါဝင်ပဲ၊ ဘယ်လို့ User Event မှာ ဘယ်လုပ်ငန်းလုပ်ရမလဲ စီမံသတ်မှတ်တဲ့ အပိုင်းကို View နဲ့ အတူ တွဲဖော်ရေးသားရပါတယ်။

## 9.1 – Getting BackboneJS

BackboneJS ကို [backbonejs.org](http://backbonejs.org) မှာ ကိုယ်တိုင် Download လုပ်ယူနိုင်သလို့၊ Yeoman ကိုသုံးပြီး တော့လည်း Generate လုပ်ယူနိုင်ပါတယ်။



BackboneJS က Model Collection တွေကို စီမံဖို့အတွက် UnderscoreJS လိုခေါ်တဲ့ JavaScript Library တစ်ခု ကို အသုံးပြုထားပြီး၊ DOM Manipulation အတွက် jQuery ကို အသုံးပြုထားပါတယ်။ တစ်နည်းအားဖြင့် Underscore နဲ့ jQuery တို့ဟာ Backbone အတွက် Dependency တွေဖြစ်ကြပါတယ်။ ဒါကြောင့် Yeoman Generator ကို သုံးပြီး Backbone App Boilerplate တစ်ခုကို Generate လုပ်မယ်ဆိုရင်လိုအပ်တဲ့ Dependency တွေ တစ်ခါတည်းပါဝင်သွားမှာ ဖြစ်ပါတယ်။ Yeoman အတွက် Backbone Generator ကို Install မလုပ်ရသေးရင် အခုလို Install လုပ်နိုင်ပါတယ်။

```
$ sudo npm install -g generator-backbone
```

Backbone Generator ကို Install လုပ်ပြီးပြုဆိုရင်တော့ Backbone App Boilerplate တစ်ခုကို အခုလို တည်ဆောက် ယူနိုင်ပါတယ်။

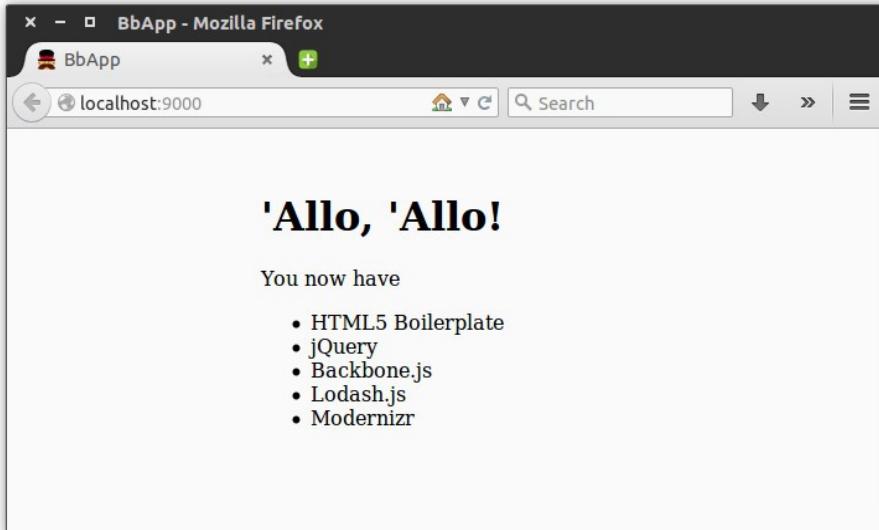
```
$ mkdir bbApp && cd bbApp
$ yo backbone
```

ရရှိလာတဲ့ Directory Structure ကို လေ့လာကြည့်ရင် အခုလိုတွေရမှာ ဖြစ်ပါတယ်။



bower\_components Directory ထဲမှာ backbone, jquery, underscore စသဖြင့် လိုအပ်တဲ့ Dependency အားလုံးပါဝင်လာတာကို တွေ့ရမှာဖြစ်ပါတယ်။ bower\_components ထဲမှာ တွေ့နေရတဲ့ lodash ဆိုတာ Underscore အစား၊ အစားထိုးသုံးနိုင်တဲ့ Underscore နဲ့ လုပ်ဆောင်ချက်တူညီတဲ့ Library တစ်ခုဖြစ်ပါတယ်။

App Boilerplate Directory ဖြစ်တဲ့ BbApp Directory ထဲမှာ grunt serve Command ကို Run ကြည့်ရင် ကျော်စောင်တို့ ရဲ့ App ကို Web Browser နဲ့ အချင်းလို့ ဖော်ပြလာမှာပဲဖြစ်ပါတယ်။



ဗုံး (၃.၂) - Previewing App

Yeoman က Scaffold Generate လုပ်စဉ်မှာ Web Server တစ်ခုကိုပါ တစ်ခါတည်း ထည့်သွင်းပေးသွားတဲ့ အပြင် Browser Sync လို့ခေါ်တဲ့ NodeJS Package တစ်ခုကိုပါ တစ်ခါတည်း Setup လုပ်ပေးသွားတဲ့ အတွက် အချင် App မှာ ပြင်ဆင်မှုတစ်ခုလုပ်လိုက်ရင် Browser မှာဖော်ပြတဲ့ ရလဒ်မှာ တစ်ခါတည်း လိုက်လံပြောင်းလဲသွားမှာ ဖြစ်ပါတယ်။ Development မှာ အလုပ်တွင်ဖော်အတွက် ကူညီပေးတဲ့ အသုံးဝင်တဲ့ လုပ်ဆောင်ချက်များ ဖြစ်ပါတယ်။

ဒီအခန်းမှာ ဖော်ပြထားတဲ့ Code တွေဟာ လက်တွေ့ကူးယူစစ်းသပ်နိုင်တဲ့ Code တွေဖြစ်ပါတယ်။

## 9.2 – Backbone Model

Model ဆိုတာ Data နဲ့ အဲဒီ Data တွေကို စီမံနိုင်တဲ့ Function တွေစုစုပါဝင်တဲ့ Object တစ်ခုဖြစ်ပါတယ်။ Backbone မပါပဲနဲ့ JavaScript Model Object တစ်ခုကို အချင်းလို့ တည်ဆောက်နိုင်ပါတယ်။

### JavaScript

```
1. var Issue = {
2.   data: {
3.     subject: "Drop down menu not working",
4.     reproduce: "...",
5.     priority: 3,
6.     status: 1
7.   },
8. }
```

```

8.  changeStatus: function(value) {
9.      this.data.status = value;
10. }
11. }
```

Issue Object အတွင်းမှာ အချက်အလက်တွေစုစုပေါင်းထည်းထားတဲ့ data နဲ့ changeStatus() လို့ ခေါ်တဲ့ အဲဒီ Data တွေကို စီမံနိုင်တဲ့ Function တို့ပါဝင်ခြင်းဖြစ်ပါတယ်။ ဒီလို JavaScript သက်သက်နဲ့ Model Object တွေ ကို တည်ဆောက်နိုင်ပေမယ့် Backbone Model ကို အသုံးပြုပြီး Model တွေတည်ဆောက်မယ်ဆိုရင် အသုံးဝင် တဲ့ လုပ်ဆောင်ချက်တွေ အသင့်ပါဝင်သွားမှာ ဖြစ်ပါတယ်။ Backbone Model Constructor တစ်ခုကို အခုလို တည်ဆောက်နိုင်ပါတယ်။

### JavaScript

```

1. var Issue = Backbone.Model.extend({
2.
3.     defaults: function() {
4.         return {
5.             subject: "",
6.             reproduce: "",
7.             priority: 2,
8.             status: 1
9.         };
10.    },
11.
12.     changeStatus: function(value) {
13.         this.save({status: value});
14.     }
15.
16. });
17. 
```

Backbone Model Constructor တစ်ခုတည်ဆောက်ဖို့အတွက် Backbone.Model.extend() ကို သုံးရပါ တယ်။ နမူနာအရ Issue ဟာ Model Constructor တစ်ခုဖြစ်သွားပြီဖြစ်ပါတယ်။ ဒါကြောင့် ဒီ Constructor ကို အသုံးပြုပြီး Model Object တွေ တည်ဆောက်နိုင်မှာဖြစ်ပါတယ်။ Model Object တစ်ခု တည်ဆောက်လိုက် တဲ့အခါ Constructor ရဲ့ defaults Property မှာသတ်မှတ်ထားတဲ့ တန်းဖိုးတွေကို Model Object မှာ အလို အလျောက် တွဲဖောက် သတ်မှတ်ပေးသွားမှာ ဖြစ်ပါတယ်။ ဥပမာ - Issue Model Constructor ကိုသုံးပြီး Model Object တွေကို အခုလို တည်ဆောက်နိုင်ပါတယ်။

### JavaScript

```

1. var issueOne = new Issue();
2.
3. var issueTwo = new Issue({
4.     subject: "Dropdown menu not working",
5.     priority: 3,
6.     status: 1
7. });
8. 
```

issueOne လိုခေါ်တဲ့ Model Object တစ်ခုတည်ဆောက်ရာမှာ တန်ဖိုးတွေ ထည့်သွင်းပေးခြင်းမရှိတဲ့အတွက် Constructor ရဲ့ defaults တန်ဖိုးတွေကို အသုံးပြုပေးသွားမှာဖြစ်ပါတယ်။ issueTwo ကိုတည်ဆောက်ရာမှာတော့ subject, priority, status စတဲ့တန်ဖိုးတွေ ထည့်သွင်းသတ်မှတ်ပေးလိုက်တဲ့အတွက် defaults တန်ဖိုး တွေနေရာမှာ သက်ဆိုင်ရာတန်ဖိုးကို အစားထိုးအသုံးပြုပေးသွားမှာဖြစ်ပါတယ်။

Model Object တစ်ခုတည်ဆောက်လိုက်တဲ့အခါ Backbone က ကြိုတင်သတ်မှတ်ထားပေးတဲ့ Function တွေနဲ့ Property တွေကို Object မှာ တစ်ခါတည်း တွဲဖက်ပေးသွားမှာဖြစ်ပါတယ်။ Backbone က Model Object မှာ တွဲဖက်ပေးလိုက်တဲ့ Function နဲ့ Property စာရင်းကို ဖော်ပြလိုက်ပါတယ်။

**get()** – Model ရဲ့ Data တန်ဖိုးတစ်ခုကို ရယူလိုတဲ့အခါ သုံးရပါတယ်။ ဥပမာ –

#### JavaScript

```
issueTwo.get("priority");           // => 3
```

**set()** – Model ရဲ့ Data တန်ဖိုး သတ်မှတ်ရာမှာသုံးရပါတယ်။ ဥပမာ –

#### JavaScript

```
issueTwo.set({priority: 2, status: 1});  
issueTwo.get("priority");           // => 2
```

**escape()** – get() နဲ့ အလုပ်လုပ်ပုံအတူတူပါပဲ။ XSS Filter ပါ တစ်ခါတည်း ပြုလုပ်ပေးပါတယ်။ ဥပမာ –

#### JavaScript

```
issueTwo.set({detail: "<script>alert(1)</script>"});  
issueTwo.escape("detail");          // => &lt;script&ampgtalert(1)&lt;/script&ampgt;
```

**has()** – Model မှာ Data တန်ဖိုး သတ်မှတ်ထားခြင်းရှိမရှိ စစ်ပေးပါတယ်။ ဥပမာ –

#### JavaScript

```
issueTwo.has("assignTo");           // => false
```

**unset()** – Model ရဲ့ Data တန်ဖိုးတစ်ခုကိုဖျက်ပေးပါတယ်။ ဥပမာ –

#### JavaScript

```
issueTwo.unset("subject");
```

**clear()** – Model ရဲ့ Data တန်ဖိုးအားလုံးကို ပယ်ဖျက်ပေးပါတယ်။

**id** – Model Object တိုင်းမှာ Unique ID တစ်ခုရှိရပါတယ်။ အဲဒီ ID ကို Data နဲ့မရောပဲ id Attribute နဲ့ သီးခြား သိမ်းဆည်းပေးထားခြင်းဖြစ်ပါတယ်။

**idAttribute** – Model Data တွေထဲက တန်ဖိုးတစ်ခုကို Model ID အဖြစ် သတ်မှတ်လိုရင် သုံးနိုင်ပါတယ်။

### JavaScript

```

1. var Issue = Backbone.Model.extend({
2.   idAttribute: "issueID",
3.   defaults: {
4.     subject: "",
5.     status: 1
6.   }
7. });
8.
9. var issueThree = new Issue({ issueID: '001', status: 2 });
10. console.log(issueThree.id); // => 001

```

issueThree လိုခေါ်တဲ့ Model Object တစ်ခုတည်ဆောက်ရာမှာ issueID: '001' ဆိုတဲ့တန်ဖိုးတစ်ခုကို တစ်ခါတည်း တွဲဖက်ပေးထားပါတယ်။ Model Constructor မှာ idAttribute: "issueID" လိုသတ်မှတ်ထား တဲ့အတွက် issueID: '001' အနေနဲ့ ပါဝင်လာတဲ့တန်ဖိုးကို Backbone က အလိုအလျောက် Model ID အနေနဲ့ သတ်မှတ်ပေးသွားမှာဖြစ်ပါတယ်။ ဒါကြောင့် issueThree.id ကို ရယူကြည့်တဲ့အခါ 001 ဖြစ်နေခြင်းဖြစ်ပါတယ်။

တစ်ခုသတိပြုပါ Model Object ရဲ့ Data တွေကို ရယူလိုရင် get() Function ကနေတစ်ဆင့် ရယူရပါတယ်။ ဥပမာ - issueThree.get("status")။ ID ကဲ Model Data နဲ့မဆိုင်ပါဘူး။ Model Property ဖြစ်ပါတယ်။ ဒါကြောင့် get() Function မလိုပဲ issueThree.id လို့ တိုက်ရှိက်ရယူခြင်းဖြစ်ပါတယ်။

**cid** – Model Object တစ်ခုတည်ဆောက်လိုက်တာနဲ့ Backbone က အလိုအလျောက် Temporary ID တစ်ခုတည် ဆောက်ပေးပြီး cid Property နဲ့ သိမ်းဆည်းပေးသွားပါတယ်။ တစ်ကယ့် ID မရှိသေးခင် Unique ID တစ်ခုအနေနဲ့ အသုံးပြုဖို့ လိုအပ်တဲ့အခါ cid ကိုယာယီ အသုံးပြုနိုင်ခြင်းဖြစ်ပါတယ်။ ဥပမာ -

### JavaScript

```
console.log(issueTwo.cid); // => c1
```

**toJSON()** – Model Data တွေကို JSON Object တစ်ခုအနေနဲ့ ပြန်ပေးတဲ့ Function ဖြစ်ပါတယ်။ ဥပမာ -

## JavaScript

```
issueTwo.toJSON();
// => { subject: "Dropdown menu not working", priority: 3, status: 1 }
```

**fetch()** – Server ကနေ အချက်အလက်တွေကို ဆက်သွယ်ရယူပြီး လက်ရှိ Model Object ကို Update လုပ်ပေးပါတယ်။ ဆက်သွယ်ရမယ့် Server လိုပ်စာ URL အနေနဲ့ Model ပါဝင်တဲ့ Collection မှာ သတ်မှတ်ထားတဲ့ URL ကို အသုံးပြုသွားမှာ ဖြစ်ပါတယ်။ Request Method အနေနဲ့ GET ကို သုံးပေးသွားမှာပါ။

**save()** – Server ကိုဆက်သွယ်ပြီး လက်ရှိ Model Object မှာပါဝင်တဲ့ အချက်အလက်တွေကို ပေးပို့ပေးပါတယ်။ ဆက်သွယ်ရမယ့် Server လိုပ်စာ URL အနေနဲ့ Model ပါဝင်တဲ့ Collection မှာ သတ်မှတ်ထားတဲ့ URL ကို အသုံးပြု သွားမှာဖြစ်ပါတယ်။ Request Method အနေနဲ့ Server ကို တစ်ကြိမ်မှ မပေးပို့ရသေးရင် POST ကို သုံးပေးသွားမှာဖြစ်ပြီး၊ တစ်ကြိမ်ပေးပို့ဖူးရင်တော့ PUT ကို အသုံးပြုပေးသွားမှာဖြစ်ပါတယ်။

**destroy()** – Server ကိုဆက်သွယ်ပြီး လက်ရှိ Model Object အတွက် သိမ်းဆည်းထားတဲ့ အချက်အလက်တွေကို ပယ်ဖျက်ပေးပါတယ်။ ဆက်သွယ်ရမယ့် Server လိုပ်စာ URL အနေနဲ့ Model ပါဝင်တဲ့ Collection မှာ သတ်မှတ်ထားတဲ့ URL ကို အသုံးပြုသွားမှာ ဖြစ်ပါတယ်။ Request Method အနေနဲ့ DELETE ကို သုံးပေးသွားမှာပါ။

**sync()** – Server ကို ဆက်သွယ်ပြီး ဆက်သွယ်တဲ့ Request Method ပေါ်မှုတည်ပြီး Model Object မှာပါဝင်တဲ့ အချက်အလက်တွေကို ရယူခြင်း၊ သိမ်းဆည်းခြင်း၊ ပယ်ဖျက်ခြင်းတို့ကို ဆောင်ရွက်ပေးပါတယ်။ create, read, update, delete စတဲ့ Option တွေကို Request Method အနေနဲ့ တွေဖက်ပေးရပါတယ်။

**validate()** – Default အနေနဲ့ ဘာမှာသတ်မှတ်မထားပါဘူး။ ဒါပေမယ့် save() Function ကိုခေါ်တိုင်းမှာ Backbone က validate() Function မှာ ရေးသားသတ်မှတ်ထားတဲ့ သတ်မှတ်ချက်တွေကို အလုပ်လုပ်ပေးပါတယ်။ ဒါကြောင့် validate() Function မှာ Model Object ရဲ့ Data တွေကို စီစဉ်တဲ့အလုပ်ကို ရေးသားထားသင့်ပါတယ်။

**validateError** – validate() Function က return ပြန်ပေးတဲ့ Error Message ကို သိမ်းထားပေးပါတယ်။ Error Message ကို တစ်နေရာမှာ ပြန်လည်ဖော်ပြလိုတဲ့အခါ ဒါ Property ထဲကနေ ရယူဖော်ပြန်လိုအသုံးဝင်ပါတယ်။

**isValid()** – validate() Function ကို Run ပေးပါတယ်။

**url()** – လက်ရှိ Model Object ရဲ့ URL ကို ပြန်ပေးပါတယ်။ collection.url/id ဆိတ် Format နဲ့ ပြန်ပေးမှာဖြစ်ပါတယ်။ Model ပါဝင်တဲ့ Collection မှာ သတ်မှတ်ထားတဲ့ URL တန်းဖိုးကိုရှုကထားပြီး Model ID ကို နောက်ကတဲ့ပေးပါတယ်။

**parse()** – Server က ပြန်ပေးတဲ့ JSON String ကို JSON Object ဖြစ်အောင် ပြောင်းပေးပါတယ်။ **save()** Function နဲ့ **fetch()** Function တို့ကို Run တိုင်း **parse()** Function ကို အလိုအလျောက် အလုပ်လုပ်ပေးပါတယ်။

**clone()** – လက်ရှိ Model Object ကို မူးပေးပါတယ်။

**isNew()** – Model Object ဟာ Server မှာ တစ်ကြိမ်မှသွားမသိမ်းရသေးတဲ့ Object အသစ်တစ်ခု ဟုတ်မဟုတ် စီစစ်ပေးပါတယ်။ ID ရှိမရှိကိုကြည့်လိုက်တာပါ။ ID မရှိသေးရင် Model အသစ်၊ ရှိနေရင် Server မှာ သွားသိမ်းထားပြီး သား Model လို ယူဆလိုက်ခြင်း ဖြစ်ပါတယ်။

**hasChanged()** – နောက်ဆုံးတစ်ခေါက် **save()** လုပ်ပြီးနောက်ပိုင်း Model Object မှာပါဝင်တဲ့အချက် အလက်တွေကို ပြင်ဆင်ထားခြင်းရှိမရှိ စီစစ်ပေးပါတယ်။

**changedAttributes()** – နောက်ဆုံးတစ်ခေါက် **save()** ပြီးနောက်ပိုင်း တန်ဖိုးပြောင်းလဲထားတဲ့ Data တွေကို ရွေးချယ်ရယူပေးပါတယ်။

**previous()** – Model Object ရဲ့ တန်ဖိုးတစ်ခုကို **set()** သို့မဟုတ် **save()** နဲ့ ပြောင်းလိုက်ချိန်မှာ ပြောင်းလိုက်တဲ့ တန်ဖိုးမတိုင်ခင်က၊ မူလတန်ဖိုးတစ်ခုကို ပြန်လည်ရရှိလိုရင် အသုံးပြုနိုင်ပါတယ်။

**previousAttributes()** – Model Object ရဲ့ တန်ဖိုးတစ်ခုကို **set()** သို့မဟုတ် **save()** နဲ့ ပြောင်းလိုက်ချိန်မှာ ယခင်မူလတန်ဖိုးအားလုံးကို ပြန်လည်ရရှိလိုတဲ့အခါ အသုံးပြုနိုင်ပါတယ်။

Backbone Model တစ်ခုတည်ဆောက်လိုက်တာနဲ့ ဒီ Function တွေနဲ့ Property တွေ အလိုအလျောက် တွဲဖက်ပါဝင် သွားမှာပဲဖြစ်ပါတယ်။

### 9.3 – Backbone Collection

Collection ဆိုတာဟာ Model တွေကိုစုစုပေါင်းထားတဲ့ Model Object Array တစ်ခု ဖြစ်ပါတယ်။ Object Array တစ်ခု ကို JavaScript သက်သက်နဲ့ ကိုယ်တိုင်လည်း တည်ဆောက်နိုင်ပေမယ့် Backbone Collection ကို အသုံးပြုတည်ဆောက်ရင် အသုံးဝင်တဲ့ လုပ်ဆောင်ချက်တွေကို အသင့်ရရှိနိုင်မှာ ဖြစ်ပါတယ်။ Backbone.Collection.**extend()** ကို အသုံးပြုပြီး Collection Constructor တစ်ခုကိုတည်ဆောက်နိုင်ပါတယ်။ အဲဒါ လိုတည်ဆောက်ရာမှာ ပါဝင်နိုင်တဲ့ Option တွေအများကြီးရှိပေမယ့် မဖြစ်မနေ ပါဝင်သင့်တဲ့ Option နှစ်ခု ကတော့ **model** နဲ့ **url** ဖြစ်ပါတယ်။ ဥပမာ –

#### JavaScript

```
1. var IssueList = Backbone.Collection.extend({
2.   url: 'http://localhost/api/'
3.   model: Issue
```

```
4. } );
```

Model Data တွေကို Database စနစ်တစ်ခုနဲ့ သိမ်းဆည်းလိုတဲ့အခါ Backbone က url မှာ သတ်မှတ်ထားတဲ့ လိပ်စာကို အသုံးပြုပြီး Server နဲ့ ဆက်သွယ်ဆောင်ရွက်ပေးသွားမှာ ဖြစ်ပါတယ်။ model ကတော့ ဒီ Collection မှာ ပါဝင်မယ့် Model အမျိုးအစားကို သတ်မှတ်ပေးဖို့ ဖြစ်ပါတယ်။ Collection ထဲကို ထည့်သွင်းတဲ့ တန်ဖိုးတွေက Model Object တွေမဟုတ်ပဲ ရှိုးရှိုး JavaScript Array တစ်ခုဖြစ်နေရင်လည်း Backbone က model မှာ သတ်မှတ်ထားတဲ့ Model Object အမျိုးအစားဖြစ်အောင် အလိုအလျောက်ဖြောင်းပြီးမှ ထည့်သွင်းပေးသွားမှာ ဖြစ်ပါတယ်။ တည်ဆောက် ရရှိထားတဲ့ Collection Constructor ကိုသုံးပြီး Backbone Collection တစ်ခုကို အခုလို တည်ဆောက်နိုင်ပါတယ်။

### JavaScript

```
var Issues = new IssueList();
```

အဲဒီလို တည်ဆောက်လိုက်တဲ့အခါ Backbone က ဖြည့်စွက်ပေးလိုက်တဲ့အတွက် Collection Object မှာ တဲ့ဖော်ပါဝင် သွားတဲ့ Function နဲ့ Property စာရင်းကို ဖော်ပြပေးလိုက်ပါတယ်။

**toJSON()** – Collection ထဲက Model တွေကို JSON Array အနေနဲ့ ပြန်ပေးပါတယ်။

**add()** – Model တစ်ခု (သို့မဟုတ်) Model Array တစ်ခုကို Collection ထဲကို ထည့်သွင်းလိုတဲ့အခါ သုံးပါတယ်။ ဥပမာ –

### JavaScript

```
Issues.add([issueOne, issueTwo]);
```

**remove()** – Model တစ်ခုကို Collection ကနေ ထုတ်လိုတဲ့အခါသုံးပါတယ်။

**reset()** – Collection ထဲမှာရှိုးတဲ့ လက်ရှိ Model တွေကို ပေးလိုက်တဲ့ Model စာရင်းသစ်နဲ့ အစားထိုးပေးပါတယ်။

**set()** – reset() နဲ့ သဘောတရားဆင်ပါတယ်။ ဒါပေမယ့် သူက စာရင်းတစ်ခုလုံးကို အစားထိုးတာမျိုးမဟုတ်ပဲ၊ မူလ Model စာရင်းနဲ့ ပေးလိုက်တဲ့ Model စာရင်းသစ်ကို တိုက်ကြည့်ပါတယ်။ Model စာရင်းသစ်မှာ ပါပြီး မူလကမရှိ သေးရင် အဲဒီ Model ကို အသစ်အနေနဲ့ ထည့်ပေးပါတယ်။ Model စာရင်းသစ်မှာပါပြီး မူလက လည်း ရှိတယ်ဆိုရင် အဲဒီ Model ကို Update လုပ်ပေးပါတယ်။ Model စာရင်းသစ်မှာမပါပေမယ့် မူလကရှိနေတယ်ဆိုရင် ပယ်ဖျက်ပေးပါတယ်။

**get()** – Collection ထဲကနေ Model တစ်ခုကို ရယူလိုတဲ့အခါသုံးပါတယ်။ Model ရဲ့ id သို့မဟုတ် cid ကို အသုံးပြုရယူရပါတယ်။ ဥပမာ –

## JavaScript

```
Issues.get("c1");
```

**at()** – Collection ထဲကနေ Model တစ်ခုကို ရယူခြင်းပဲဖြစ်ပါတယ်။ ဒါပေမယ့် သူက Model ID ကို မသုံးပဲ Model Index ကို အသုံးပြုရယူပေးပါတယ်။ ဥပမာ –

## JavaScript

```
Issues.at(2);
```

**push()** – Model တစ်ခုကို Collection ရဲ့နောက်ဆုံးကနေ ထည့်ပေးပါတယ်။ `add()` နဲ့ဆင်ပါတယ်။

**pop()** – Collection ထဲက နောက်ဆုံး Model ကို ဖြုတ်ထုတ်ပေးပါတယ်။ `remove()` နဲ့ဆင်ပါတယ်။

**unshift()** – Collection ထဲကို Model တစ်ခု ရွှေ့ဆုံးကနေ ထည့်ပေးပါတယ်။ `add()` နဲ့ ဆင်ပါတယ်။

**shift()** – Collection ထဲက ရွှေ့ဆုံး Model ကို ဖြုတ်ထုတ်ပေးပါတယ်။ `remove()` နဲ့ ဆင်ပါတယ်။

**slice()** – Collection ကို စိတ်ပိုင်းရယူလိုတဲ့အခါ သုံးနိုင်ပါတယ်။ အစ Index နဲ့ အဆုံး Index ပေးရပါတယ်။ ဥပမာ – Model တွေထဲက ထိပ်ဆုံး (၅) ခုကိုလိုချင်တယ်ဆိုရင် အခုလိုရယူနိုင်ပါတယ်။

## JavaScript

```
var topFive = Issues.slice(0, 5);
```

**length** – Collection ထဲမှာရှိတဲ့ Model အရေအတွက်ကို ပေးပါတယ်။

**comparator** – Collection ထဲက Model Object တွေကို Sorting စီတဲ့အခါ အသုံးပြုစေလိုတဲ့ Model Property ကို သတ်မှတ်ပေးနိုင်ပါတယ်။ ဥပမာ –

## JavaScript

```
Issues.comparator = "priority";
```

comparator ကို `priority` လိုသတ်မှတ်လိုက်တဲ့အတွက် Model Object တွေကို `priority` အလိုက် Sorting စီပေးသွားမှာ ဖြစ်ပါတယ်။

**sort()** – Model Object တစ်ခုကို ပြင်ပြီးနောက် Collection ထဲက Model Object တွေကို Sorting ပြန်စီစဉ် လိုက်တဲ့အခါ သုံးရပါတယ်။

**pluck()** – Model Object တွေရဲ့ Property တစ်ခုကို Array အနေနဲ့ လိုချင်တဲ့အခါသုံးရပါတယ်။ ဥပမာ –

### JavaScript

```
Issues.pluck("subject");
```

Model Object တွေအားလုံးရဲ့ subject တွေကိုချဉ်းချုပ်ပြီး Array အနေနဲ့ ပြန်ပေးသွားမှာဖြစ်ပါတယ်။

**where()** – Filter လုပ်ထားတဲ့ Model Object စာရင်းကို ရယူဖို့အတွက် သုံးနိုင်ပါတယ်။ ဥပမာ –

### JavaScript

```
Issues.where({status: 2});
```

Model Object တွေထဲက status: 2 တန်ဖိုးရှိတဲ့ Object စာရင်းကိုပဲ ရွေးထုတ်ပေးသွားမှာဖြစ်ပါတယ်။

**findWhere()** – where() နဲ့သဘောတရား အတူတူပါပဲ။ ဒါပေမယ့် findWhere() က Object စာရင်း ပြန်မပေးပဲ ပထမဆုံးတွေတဲ့ Object တစ်ခုကိုသာ ပြန်ပေးပါတယ်။

**parse()** – Model ရဲ့ parse() နဲ့ သဘောတရားအတူတူပါပဲ။ Server နဲ့ဆက်သွယ်တဲ့လုပ်ငန်းတွေမှာ Server ပြန်ပေးတဲ့ Object String ကို JSON Object ပြောင်းပေးပါတယ်။

**clone()** – လက်ရှိ Collection ကို မူဖွားပေးပါတယ်။

**fetch()** – URL မှာ သတ်မှတ်ထားတဲ့ လိပ်စာကိုအသုံးပြုပြီး Server နဲ့ဆက်သွယ်ပြီး Model စာရင်းကို ရယူပေး ပါတယ်။ ရရှိလာတဲ့အခါ set() Function ကိုသုံးပြီး လက်ရှိ Collection ကို Update လုပ်ပေးပါတယ်။ GET Request Method ကိုသုံးပါတယ်။

**create()** – Model တစ်ခုတည်ဆောက်ပြီး တည်ဆောက်ရရှိထားတဲ့ Model ကို Collection ထဲမှာထည့်သွင်းပေး သလို URL မှာ သတ်မှတ်ထားတဲ့ လိပ်စာကိုအသုံးပြု ပြီး Server ကိုလည်း တစ်ခါတည်း ပေးပို့ပေးပါတယ်။ POST Request Method ကို သုံးပါတယ်။

**sync()** – Model ရဲ့ sync() နဲ့ သဘောသဘာဝတူပါတယ်။ Server နဲ့ဆက်သွယ်ပြီး Collection တွေကို ပေးပို့ခြင်း၊ ရယူခြင်း၊ ပြင်ဆင်ခြင်း၊ ပယ်ဖျက်ခြင်းတို့ကို ဆောင်ရွက်ပေးပါတယ်။ create, read, update, delete တို့

ကို Request Method အနေနဲ့ တွဲဖက်ပေးပြန်ပါတယ်။

ဒီ Function တွေနဲ့ Property တွေအပြင် UnderscoreJS ရဲ့ Array တွေစီမံတဲ့ Function တွေကိုလည်း Collection နဲ့ အတူ တွဲဖက်အသုံးပြန်ပါတယ်။ Underscore Function တွေထဲက မှတ်သားသင့်တာတွေကို ဖော်ပြလိုက်ပါတယ်။

**each()** – Collection တစ်ခုလုံးကို Loop လုပ်စေလိုတဲ့အခါ each() ကို သုံးနိုင်ပါတယ်။ ဥပမာ –

#### JavaScript

```
Issues.each(function(model) {
  console.log( model.get("subject") );
});
```

**map()** – each() လိုပဲ Collection တစ်ခုလုံးကို Loop ပါတ်ပေးပါတယ်။ ကွာသွားတာကတော့၊ map() Function က အဆုံးသတ်ရလဒ် ကို Array အနေနဲ့ ပြန်ပေးမှာ ဖြစ်ပါတယ်။ ဥပမာ –

#### JavaScript

```
var subjectList = Issues.map(function(model) {
  return model.get("subject");
});
```

Collection ကို Loop လုပ်ပြီး Model တွေရဲ့ subject အားလုံးကို Array အနေနဲ့ ပြန်ပေးမှာပဲဖြစ်ပါတယ်။

**find()** – Collection ကို Loop လုပ်ပြီး ပေးလိုက်တဲ့ Condition နဲ့ ကိုက်ညီတဲ့ ပထမဆုံး Model Object ကို ပြန်ပေးမှာဖြစ်ပါတယ်။ ဥပမာ –

#### JavaScript

```
var result = Issues.find(function(model) {
  return model.get("subject") == "Dropdown menu not working";
});
```

Collection ထဲမှာရှိတဲ့ Model တွေထဲက subject တန်ဖိုး Dropdown menu not working ကို ရှာပြီး တွေတယ် ဆိုရင် အဲဒီတန်ဖိုးပါဝင်တဲ့ Model Object ကို ပြန်ပေးမှာဖြစ်ပါတယ်။

**filter()** – find() နဲ့ သဘောတရားဆင်ပါတယ်။ find() က Condition ကိုက်ညီတဲ့ Model Object တစ်ခုကိုသာ ပြန်ပေးပေမယ့် filter() ကတော့ ကိုက်ညီသမျှ Model Object အားလုံးကို Array အနေနဲ့ ပြန်ပေးမှာဖြစ်ပါတယ်။ ဥပမာ –

**JavaScript**

```
var result = Issues.filter(function(model) {
    return model.get("status") == 2;
});
```

Collection တဲ့ status တန်ဖိုး 2 နဲ့ ညီတဲ့ Model အားလုံးကို Array အနေနဲ့ ပြန်ပေးသွားမှာဖြစ်ပါတယ်။

**reject()** – filter() နဲ့ ဆန့်ကျင်ဘက်ဖြစ်ပါတယ်။ Condition နဲ့ မကိုက်ညီတဲ့ Model Object စာရင်းကို ပြန်ပေးမှာ ဖြစ်ပါတယ်။

**every()** – Collection တဲ့ Model Object အားလုံး Condition နဲ့ ကိုက်ညီခြင်းရှိမရှိ စီစဉ်ပေးပါတယ်။ ဥပမာ –

**JavaScript**

```
var result = Issues.every(function(model) {
    return model.get("status") == 4;
});
```

Collection တဲ့ Model Object အားလုံးရဲ့ status တန်ဖိုး 4 ဖြစ်နေရင် true ရလဒ်ကို ရရှိမှာဖြစ်ပါတယ်။

**some()** – Collection တဲ့ Model Object တစ်ခု၏ Condition နဲ့ ကိုက်ညီခဲ့ရင် true ရလဒ်ပြန်ပေးမှာပါ။

**shuffle()** – Collection တဲ့ Model Object တွေကို အစီအစဉ်အတိုင်းမဟုတ်ပဲ Random ပြန်စီပေးပါတယ်။

**toArray()** – Collection တဲ့ Model တွေအားလုံးကို Array အနေနဲ့ ပြန်ပေးပါတယ်။

**first()** – Collection တဲ့ ပထမဆုံး Model ကို ပြန်ပေးပါတယ်။

**last()** – Collection တဲ့ နောက်ဆုံး Model ကို ပြန်ပေးပါတယ်။

**isEmpty()** – Collection စာရင်းက Model တွေမရှိပဲ အလွတ်ဖြစ်နေသလား စစ်ပေးပါတယ်။

Backbone Collection ကို Database စနစ်တစ်ခုရဲ့ Table နဲ့ အလားသဏ္ဌာန်တူတယ်လို့ ယူဆနိုင်ပါတယ်။ Collection ထဲမှာ ပါဝင်တဲ့ Model တွေကတော့ Table Record များဖြစ်ပါတယ်။ RDBMS စနစ်တွေမှာ Table ထဲက Record တွေကို SQL Command တွေနဲ့ စီမံနိုင်သလိုပဲ Backbone Collection ထဲက Model တွေကိုလည်း အချော်ပြုခဲ့တဲ့ Function တွေ ပေါင်းစပ်အသုံးပြုခြင်းအားဖြင့် စီမံနိုင်မှာပဲ ဖြစ်ပါတယ်။

## 9.4 – Backbone View

Backbone မှာ View ဆိုတာ Optional ပါ။ Backbone View ကို မသုံးပါ jQuery (သို့မဟုတ်) အခြားနှစ်သက်ရာ JavaScript UI Library စွဲ Template Engine တွေနဲ့ User Interface ကို တည်ဆောက်နိုင်ပါတယ်။ ဥပမာ – ရုံးရုံး jQuery ကို Backbone Collection နဲ့ဖွေကြပြီး အခုံးပြုနိုင်ပါတယ်။

### JavaScript

```

1. var Issue = Backbone.Model.extend({
2.   defaults: {
3.     subject: "",
4.     status: 1
5.   }
6. });
7.
8. var IssueList = Backbone.Collection.extend({
9.   model: Issue
10. });
11.
12. var Issues = new IssueList();
13.
14. var issueOne = new Issue({ subject: "Issue One", status: 1 });
15. var issueTwo = new Issue({ subject: "Issue Two", status: 1 });
16. var issueThree = new Issue({ subject: "Issue Three", status: 1 });
17.
18. Issues.add([ issueOne, issueTwo, issueThree ]);
19.
20. Issues.each(function(model) {
21.   $("ul").append("<li>" + model.get("subject") + "</li>");
22. });

```

နမူနာကို လေ့လာကြည့်ရင် လိုင်းနံပါတ် (၁) မှာ Issue Model Constructor ကို တည်ဆောက်ထားပါတယ်။ လိုင်းနံပါတ် (၂) မှာ IssueList Collection Constructor ကို တည်ဆောက်ထားပါတယ်။ လိုင်းနံပါတ် (၃) မှာ IssueList Constructor ကို သုံးပြီး Issues လိုခေါ်တဲ့ Collection Object တစ်ခု တည်ဆောက်ထားပါတယ်။ လိုင်းနံပါတ် (၁၄ - ၁၅ - ၁၆) မှာ Issue Constructor ကိုသုံးပြီး Model Object သုံးခု တည်ဆောက်ထားပါတယ်။ တည်ဆောက်ထား တဲ့ Model Object တွေကို add() Function သုံးပြီး လိုင်းနံပါတ် (၁၈) မှာ Issues Collection ထဲ ထည့်သွင်း ထားပါတယ်။ နောက်ဆုံး လိုင်းနံပါတ် (၂၀) မှာတော့ Issues ကို each() နဲ့ Loop လုပ်ပြီး Model တွေရဲ့ subject ကို jQuery အကူအညီနဲ့ List UI အဖြစ် ဖန်တီးတည်ဆောက်လိုက်ခြင်းပဲဖြစ်ပါတယ်။ သဘောသဘာဝကို မြင်သာအောင် ဖော်ပြခြင်းဖြစ်ပါတယ်။ လက်တွေမှာ Backbone View ကပေးနိုင်တဲ့ အားသာချက်တွေရှိပါတယ်။

Backbone.View.extend() ကို အသုံးပြု ပြီး Backbone View Constructor တစ်ခုကို တည်ဆောက်နိုင်ပါတယ်။

## JavaScript

```

1. var IssueView = Backbone.View.extend({
2.   tagName: "li",
3.
4.   events: {
5.     "click .new": "addNew"
6.   },
7.
8.   initialize: function() {
9.     this.listenTo(this.model, 'change', this.render);
10.  },
11.
12.  render: function() {
13.    this.$el.html(this.model.get("status"));
14.    return this;
15.  },
16.
17.  addNew: function() {
18.    ...
19.  }
20. });

```

အဲဒီလို တည်ဆောက်ရာမှာ tagName Property ကိုသုံးပြီး View Template ကို ဖော်ပြစ်စဉ်တဲ့ HTML Element Name ကို သတ်မှတ်နိုင်ပါတယ်။ events Property နဲ့ Event List ကို ကြော်လော်နိုင်ပါတယ်။ နမူနာ လိုင်းနံပါတ် (၅) အရ Class မှာ new လို့ သတ်မှတ်ထားတဲ့ Element ကို Click လုပ်ရင် addNew() Function ကို အလုပ်လုပ်ပေးပါလို့ သတ်မှတ်ထားခြင်း ဖြစ်ပါတယ်။

View Constructor ကိုသုံးပြီး View Object တစ်ခုတည်ဆောက်စဉ်မှာ အလုပ်လုပ်သွားစေလိုတဲ့ လုပ်ဆောင် ချက်တွေကို initialize Property နဲ့ သတ်မှတ်ထားနိုင်ပါတယ်။ နမူနာ လိုင်းနံပါတ် (၉) မှာ listenTo() Function ကို သုံးပြီး Model မှာ တစ်ခုအပြောင်းအလဲဖြစ်တိုင်း render() Function အလုပ်လုပ်စေဖို့ သတ်မှတ်ထားပါတယ်။ ဒီနည်းနဲ့ Model တန်ဖိုးပြောင်းတိုင်း View ရဲဖော်ပြပုလည်း အလိုအလျောက် လိုက်ပြောင်းနေအောင် သတ်မှတ်ထားလို့ ရတဲ့သဘော ဖြစ်ပါတယ်။

listenTo() Function အတွက် Parameter (၃) ခုပေးရပါတယ်။ ပထမတစ်ခုက ပြုပြင်ပြောင်းလဲမှုတွေ ရှိမရှိ စောင့်ကြည့်လိုတဲ့ Model (သို့မဟုတ်) Collection ဖြစ်ပါတယ်။ နမူနာမှာ this.model လို့ပြောထားတဲ့ အတွက် လက်ရှိ View နဲ့ တွဲဖက်ထားတဲ့ Model ကို ညွှန်းထားခြင်း ဖြစ်ပါတယ်။ အခြား Model (သို့မဟုတ်) Collection တစ်ခုကိုလည်း ညွှန်းထားနိုင်ပါတယ်။ ဒုတိယ Parameter အနေနဲ့ Event ကိုပေးရပါတယ်။ နမူနာမှာ change ကို ပေးထားပါတယ်။ အခြားအသုံးပြုလိုရတဲ့ Event စာရင်းကို ခေါ်နေတော့ ဆက်လက်ဖော်ပြပေးပါမယ်။ နောက်ဆုံး Parameter ကတော့ Event ဖြစ်ပေါ်တဲ့အခါ အလုပ်လုပ်စေလိုတဲ့ Function ဖြစ်ပါတယ်။ ဥပမာ - Issues Collection ထဲကို Model တစ်ခုထည့်သွင်းလိုက်တိုင်း အလိုအလျောက် အလုပ်လုပ်သွားစေလိုတဲ့ Function ရှိရင် အခါ လို့ သတ်မှတ်ထားနိုင်ပါတယ်။

## JavaScript

```
this.listenTo(Issues, 'add', this.doSomething());
```

နမူနာရဲ့ render() Function အတွင်းမှာ လက်ရှိ View Element ရဲ့ HTML Content အဖြစ် Model subject ကို ဖော်ပြပေးဖို့သတ်မှတ်ထားပါတယ်။ \$el ဆိတ် လက်ရှိ View Element ကို Select လုပ်လိုက်ခြင်းဖြစ်ပါတယ်။ သူနဲ့ တဲ့ဖက်ပြီး jQuery DOM Function တွေကိုသုံးနိုင်ပါတယ်။ လိုင်းနံပါတ် (၁၃) မှာသုံးထားတဲ့ html() Function ဟာ Backbone Function မဟုတ်ပါဘူး။ jQuery Function တစ်ခုဖြစ်ပါတယ်။ တစ်နည်းအားဖြင့် Backbone View တဲ့မှာ jQuery Function တွေကို လိုအပ်သလို တဲ့ဖက်အသုံးပြုနိုင်ခြင်းဖြစ်ပါတယ်။ jQuery အကြောင်းကို Professional Web Developer ရဲ့ အခန်း (၆) မှာ ဖော်ပြခဲ့ပြီး ဖြစ်ပါတယ်။

နမူနာကို လေ့လာကြည့်ရင် Backbone View ကပေးတဲ့ အမိကအကျခုံး အားသာချက်က၊ Model မှာ အပြောင်းအလဲ တစ်ခုရှိတာနဲ့ View မှာ အလိုအလျောက် လိုက်လံပြောင်းလဲအောင် စီမံထားနိုင်ခြင်းဖြစ်တယ်ဆိုတာကို တွေ့ရနိုင်ပါတယ်။ နောက်ထပ် အားသာချက်တစ်ခုကတော့ View တစ်ခုနဲ့သက်ဆိုင်တဲ့ Event တွေကို တစ်နေရာထဲမှာ စုစုပေါင်းကြော် ထားနိုင်ခြင်းပဲဖြစ်ပါတယ်။ Backbone View နဲ့အတူ တဲ့ဖက်အသုံးပြုနိုင်တဲ့ Function တွေကို ဖော်ပြလိုက်ပါတယ်။

**el** - အထက်ကန်မူနာမှာ tagName ကိုအသုံးပြုပြီး View အတွက် အသုံးပြုရမယ့် HTML Element Name ကို သတ်မှတ် ထားပါတယ်။ အဲဒီလိုမသတ်မှတ်ပဲ ရှိနေပြီးသား HTML Element တစ်ခုကို သုံးစေလိုရင် el Property ကို အသုံးပြုနိုင် ပါတယ်။ ဥပမာ -

### JavaScript

```
var IssueView = Backbone.View.extend({
  el: "#list"
});
```

ID ကို #list လိုသတ်မှတ်ထားတဲ့ HTML Element ကို View Element အနေနဲ့ အသုံးချသွားမှာဖြစ်ပါတယ်။

**\$el** - tagName (သို့မဟုတ်) el နဲ့ ကြော်သတ်မှတ်ထားတဲ့ View Element ကို jQuery Selector သုံးပြီး Select လုပ်ထားပေးခြင်းဖြစ်ပါတယ်။

**attributes** - View Element အတွက် သတ်မှတ်လိုတဲ့ HTML Attribute တွေကို ဒီ Property နဲ့ သတ်မှတ်ထားနိုင်ပါတယ်။ ဥပမာ -

### JavaScript

```
var IssueView = Backbone.View.extend({
  el: "#list",
  attributes: {class: 'issue', title: 'click to edit'}
});
```

**\$(selector)** – View Element အတွင်းက အမြား Element တွေကို Select လုပ်လိုတဲ့အခါ သုံးပါတယ်။

### JavaScript

```
var IssueView = Backbone.View.extend({
  render: function() {
    this.$('.subject').html(this.model.get('subject'));
  }
});
```

View Element အတွင်းက Class .subject သတ်မှတ်ထားတဲ့ Element ကို Select လုပ်ထားခြင်းဖြစ်ပါတယ်။

**events** – View Element ပေါ်မှာ ပြုလုပ်တဲ့ User Event တွေအတွက် သက်ဆိုင်ရာ Function တဲ့ဖက်ကြော်ဖို့ အသုံးပြုပါတယ်။ "event element": "function" ဆိုတဲ့ Format နဲ့ ကြော်ရပါတယ်။ event နေရာမှာ jQuery Event အားလုံး အသုံးပြုနိုင်ပါတယ်။ element နေရာမှာ jQuery Selector ပေးရပါတယ်။ Function ကတော့ မိမိနှစ်သက်သလို ကြော်သတ်မှတ်ထားနိုင်ပါတယ်။ ဥပမာ –

### JavaScript

```
var IssueView = Backbone.View.extend({
  el: "#list",
  events: {
    "click .new": addNew,
    "focus input": getUpdate,
    "change #form select": updateStatus
  },
  addNew: function() { ... },
  getUpdate: function() { ... },
  updateStatus: function() { ... }
});
```

**render()** – Default အနေနဲ့ ဘာအလုပ်မှလုပ်မပေးပါဘူး။ ပါဝင်စေလိုတဲ့လုပ်ဆောင်ချက်တွေကို ကိုယ်တိုင် သတ်မှတ်ပေးရပါတယ်။

**remove()** – View ကို ပယ်ဖျက်စေလိုတဲ့အခါသုံးပါတယ်။

View နဲ့အတူ တဲ့ဖက်အသုံးပြုဖို့အတွက် အရေးပါတဲ့ Function တစ်ခုကတော့ template() Function ပဲဖြစ်ပါ တယ်။ Backbone ရဲ့ Function မဟုတ်ပါဘူး။ Underscore ရဲ့ Function ဖြစ်ပါတယ်။ Format လုပ်ထားတဲ့ HTML String အတွင်းမှာ သက်ဆိုင်ရာ တန်ဖိုးတွေကို အစားထိုးပေးနိုင်ပါတယ်။ ဥပမာ –

### JavaScript

```
var template = _.template("<li><%= subject %> (<%= status %>)</li>");
template({ subject: "Dropdown menu not work", status: 2 });
```

```
// => <li>Dropdown menu not work (2)</li>
```

အရင်ဆုံး `_.template` (Underscore Dot Template) ကို အသုံးပြုပြီး HTML Template Function ကို ကြော်လှပါတယ်။ HTML Template အတွင်းမှာ Output အနေနဲ့ ဖော်ပြစ်လိုတဲ့ တန်ဖိုးတွေကို `<%= %>` Template Tag နဲ့ ထည့်သွင်းသတ်မှတ်နိုင်ပါတယ်။ JavaScript Code တွေ Template ထဲမှာ ထည့်သွင်းလိုရင်တော့ `<% %>` ကို သုံးနိုင် ပါတယ်။ ရရှိလာတဲ့ Template Function ကို Run တဲ့အချင်းမှာ Data ကို ပေးခြင်းအားဖြင့် HTML Template နဲ့ Data တဲ့ဖက်ထားတဲ့ ရလဒ်ကို ရရှိမှာဖြစ်ပါတယ်။

## 9.5 – Backbone Events

Backbone Model နဲ့ Collection တွေမှာ တန်ဖိုးအသစ်ထည့်သွင်းလိုက်ခြင်း၊ ပြုပြင်လိုက်ခြင်း၊ ပယ်ဖျက်လိုက်ခြင်း စတဲ့ လုပ်ဆောင်ချက်တစ်ခုပြုလုပ်တိုင်းမှာ အလိုအလျောက်ဆောင်ရွက်သွားစေလိုတဲ့ Function တွေရှိရင် Event အနေနဲ့ သတ်မှတ်ထားနိုင်ပါတယ်။

ဥပမာ –

### JavaScript

```
1. var Issue = Backbone.Model.extend({
2.   defaults: {
3.     subject: "",
4.     status: 2
5.   },
6.   initialize: function() {
7.     this.on("change", function() { console.log("I've changed") });
8.   }
9. });
10.
11. var issueOne = new Issue();
12. issueOne.set({status: "Dropdown menu not working"});

// => I've changed
```

Model Constructor ရဲ့ `initialize` မှာ `on()` Function ကို သုံးပြီး Model မှာ အပြောင်းအလဲတစ်ခုခုဖြစ်တိုင်း အလိုအလျောက် လုပ်သွားစေလိုတဲ့ Function ကို သတ်မှတ်ထားပါတယ်။ `change` ဆိုတာ အပြောင်းအလဲတစ်ခုရှိရင် ဆိုတဲ့ အဓိပါယ်ပဲဖြစ်ပါတယ်။ `change` အစား သုံးလို့ရတဲ့ Event တွေကိုဖော်ပြလိုက်ပါတယ်။

- **add** – Model ကို Collection ထဲကိုထည့်လိုက်တဲ့အခါ ဖြစ်ပေါ်တဲ့ Event
- **remove** – Model ကို Collection ထဲကနေထုတ်လိုက်တဲ့အခါ ဖြစ်ပေါ်တဲ့ Event
- **reset** – Collection ထဲက Model တွေကို စာရင်းအသစ်နဲ့ Reset လုပ်တဲ့အခါ ဖြစ်ပေါ်တဲ့ Event
- **sort** – Collection ကို Sorting စီလိုက်တဲ့အခါ ဖြစ်ပေါ်တဲ့ Event
- **change** – Model တန်ဖိုးတွေပြောင်းသွားတဲ့အခါ ဖြစ်ပေါ်တဲ့ Event
- **change:attribute** – ဥပမာ `change:status` – Model ထဲက `status` တန်ဖိုးပြောင်းသွားတဲ့

### အခါဖြစ်ပေါ်တဲ့ Event

- **destroy** – Model ကို ပယ်ဖျက်လိုက်တဲ့အခါ ဖြစ်ပေါ်တဲ့ Event
- **request** – Server ကို ဆက်သွယ်တဲ့အခါ ဖြစ်ပေါ်တဲ့ Event
- **sync** – Server နဲ့ ဆက်သွယ်ဆောင်ရွက်တဲ့လုပ်ငန်း အောင်မြင်တဲ့အခါ ဖြစ်ပေါ်တဲ့ Event
- **error** – Server နဲ့ ဆက်သွယ်မှုမအောင်မြင်တဲ့အခါ ဖြစ်ပေါ်တဲ့ Event
- **invalid** – Model Validation မအောင်မြင်တဲ့အခါ ဖြစ်ပေါ်တဲ့ Event
- **all** – အထက် Event တစ်ခုခါ ဖြစ်ပေါ်တိုင်းမှာ ဖြစ်ပေါ်တဲ့ Event

ဘယ် Event မှာ ဘာအလုပ်လုပ်ရမလဲဆိုတာကို အထက်ကန်မှုနာမှာ ဖော်ပြထားသလို `on()` Function နဲ့ ကြော်လော်တဲ့အနေဖြင့်ပါတယ်။ Event အမှန်တစ်ကယ်ဖြစ်ပေါ်ခြင်း မရှိပေမယ့် လိုအပ်လို့ Event Function ကို Run စေ ချင်ရင် `trigger()` Function ကို သုံးနိုင်ပါတယ်။ ဥပမာ –

#### JavaScript

```
IssueOne.trigger("change");
```

Event Function တွေထဲမှာ အမိကကျေတဲ့ Function ကတော့ `listenTo()` ဖြစ်ပါတယ်။ Model (သို့) Collection (သို့) View တစ်ခုမှာဖြစ်ပေါ်တဲ့ Event တွေကို အခြား တစ်နေရာကနေ စောင့်ကြည့်ပေးခြင်းဖြစ်ပါတယ်။ View အတွက် နမူနာဖော်ပြခဲ့စဉ်က `listenTo()` ကိုသုံးပြီး Model မှာ အပြောင်းအလဲတစ်ခုဖြစ်တိုင်း View ရဲ့ `render()` Function ကို နောက်တစ်ကြိမ် ပြန် Run ပေးတဲ့ လုပ်ဆောင်ချက်ကို ထည့်သွင်းဖော်ပြခဲ့ပါတယ်။ နမူနာမှာ change Event ကို နမူနာအနေနဲ့ပေးထားပေမယ့် `add`, `remove`, `reset`, `destroy` စတဲ့ အခြား Event တွေကိုလည်း အသုံးပြုနိုင်ပါတယ်။

အခုခိုရင် Backbone Model, Collection, View နဲ့ Event တို့အကြောင်းကို ဖော်ပြခဲ့ပြီးဖြစ်ပါတယ်။ ထည့်သွင်းမဖော် ပြရသေးတဲ့ လုပ်ဆောင်ချက်ကတော့ Router ပဲဖြစ်ပါတယ်။ [backbonejs.org](http://backbonejs.org) မှာ ဆက်လက်လေ့လာနိုင်ပါတယ်။

## 9.6 – Example Backbone App

အခန်း (၁၅) မှာ Backbone ကို အသုံးပြုပြီး Issue Tracking System တစ်ခုအတွက် Web Client တစ်ခုရေးသားပဲ ကို အပြည့်အစုံဖော်ပြသားမှာပါ။ ဒီနေရာမှာ Issue တွေ မှတ်တမ်းတင်လိုကြပြီး၊ မှတ်တမ်းတင်ထားတဲ့ Issue စာရင်းကို ပြန်လည်ကြည့်ရှုနိုင်တဲ့ App လေးတစ်ခုကို နမူနာအနေနဲ့ ဖန်တီးကြည့်ပါမယ်။

ပထမဥ္ပီးဆုံးအနေနဲ့ Backbone App Boilerplate တစ်ခုကို Generate မလုပ်ရသေးရင် လုပ်ဖို့လိုပါမယ်။

```
$ mkdir bbApp && cd bbApp
$ yo backbone
```

Yeoman က Bootstrap, RequireJS စာတဲ့ Library တွေကိုရော ထည့်သွင်း Install လုပ်လိုသလားမေးပါတိမှု မယ်။ ကျွန်ုတ် တို့နဲ့မှနာမှာ CSS တွေကို ကိုယ်တိုင်ရေးနေစရာမလိုအောင် Bootstrap CSS Framework ကို အသုံးပြုလိုတဲ့အတွက် ထည့်သွင်း Install လုပ်ပေးသင့်ပါတယ်။ အကယ်၍ Backbone နဲ့အတူ တစ်ခါတည်း Install မလုပ်ဖြစ် ရင်လည်း အခုလို သီးခြား Install လုပ်နိုင်ပါတယ်။

```
$ bower install bootstrap
```

Backbone, jQuery, Underscore နဲ့ Bootstrap အားလုံးပါဝင်တဲ့ Boilerplate ရပြီခိုရင် နဲ့မှနာ App ကို စတင်ရေး သားနိုင်ပါပြီ။ Code နဲ့မှနာကို တစ်လုံးမကျွန်ုတ်အကုန် မဖော်ပြတော့ပါဘူး။ အပိုင်းလိုက် ခွဲခြားဖော်ပြသွားပါမယ်။ ပထမဦးး ဆုံးအနေနဲ့ Issue List ဖော်ပြန့်အတွက် Table အလွတ်တစ်ခုကို HTML နဲ့ အခုလို တည်ဆောက်ပေးရပါမယ်။

## HTML

```
1. <table class="table" id="list">
2.   <tr>
3.     <th>Subject</th>
4.     <th>Priority</th>
5.     <th>Severity</th>
6.     <th>Assigned To</th>
7.     <th>Status</th>
8.     <th>#</th>
9.   </tr>
10. </table>
11.
12. <button class="btn btn-primary" data-toggle="modal" data-target="#new">
13.   <span class="glyphicon glyphicon-plus"></span> New Issue
14. </button>
```

#list လို့ ID သတ်မှတ်ပေးထားတဲ့ <table> Element တစ်ခုဖြစ်ပါတယ်။ ခေါင်းစီး Row တစ်ခုပဲပါပြီး Content တွေမပါသေးပါဘူး။ သူအောက်မှာ Issue အသစ်တွေထပ်ထည့်နိုင်တဲ့ New Issue Form ကိုဖွေ့ဖြိုး ဖော်ပြန်ပေးထားပါသေးတယ်။ Bootstrap ရဲ့ Modal Dialog လုပ်ဆောင်ချက်ကို သုံးပြီး New Issue Form ကို Dialog Box နဲ့ ဖော်ပြမှုပါ။ ဒါကြောင့် Dialog Box ရဲ့ ID အဖြစ် သတ်မှတ်မယ့် #new ကို data-target Attribute နဲ့ မွေးသားပေးပါတယ်။ Bootstrap CSS Framework အကြောင်းကို အခန်း (၁၄) မှာ ဆက်လက်ဖော်ပြပါမယ်။

ဆက်လက်ပြီး ID ကို #new လို့ သတ်မှတ်ထားတဲ့ New Issue Form တစ်ခုကို အခုလိုထည့်သွင်းပေးရပါမယ်။

## HTML

```
1. <div class="modal fade" id="new">
2.   <div class="modal-dialog">
3.     <div class="modal-content">
4.
```

```

5.      <div class="modal-header">
6.          <button type="button" class="close" data-dismiss="modal">
7.              <span aria-hidden="true">&times;</span>
8.          </button>
9.          <h4 class="modal-title">New Issue</h4>
10.     </div>
11.
12.     <div class="modal-body">
13.
14.         <div class="form-group">
15.             <input class="form-control" type="text"
16.                   id="subject" placeholder="Subject">
17.         </div>
18.
19.         <div class="form-group">
20.             <select class="form-control" id="priority">
21.                 <option value="-1">-- Priority --</option>
22.                 <option value="0">Normal</option>
23.                 <option value="1">Important</option>
24.                 <option value="2">Critical</option>
25.             </select>
26.         </div>
27.
28.         <div class="form-group">
29.             <select class="form-control" id="severity">
30.                 <option value="-1">-- Severity --</option>
31.                 <option value="0">Minor</option>
32.                 <option value="1">Major</option>
33.                 <option value="2">Crash</option>
34.             </select>
35.         </div>
36.
37.         <div class="form-group">
38.             <input type="text" class="form-control"
39.                   id="assigned-to" placeholder="Assigned To">
40.         </div>
41.
42.         <div class="form-group">
43.             <select class="form-control" id="status">
44.                 <option value="-1">-- Status --</option>
45.                 <option value="0">Assigned</option>
46.                 <option value="1">WIP</option>
47.                 <option value="2">Done</option>
48.                 <option value="3">Close</option>
49.             </select>
50.         </div>
51.
52.         <div class="modal-footer">
53.             <button class="btn btn-primary" id="submit">Submit Issue</button>
54.         </div>
55.
56.     </div>
57. </div>
58. </div>

```

.modal -header, .modal-body, .modal-footer စတဲ့ Element တွေ လိုအပ်တဲ့အတွက် HTML Structure က အဆင့်ဆင့်ဖြစ်နေတာပါ။ လက်တွေ့လို အပ်တာကတော့ #subject, #priority, #severity, #assigned-to နဲ့ #status ဆိုတဲ့ Input Element တွေသာဖြစ်ပါတယ်။ Input Element တွေရဲ့အောက်ဆုံးမှာ <button> Element တစ်ခု ကို #submit ဆိုတဲ့ ID နဲ့ ထည့်သွင်းထားပါသေးတယ်။ Bootstrap က ဒီ Structure တစ်ခုလုံးကို ဖျောက်ထားပြီး #new Button ကို နှိပ်လိုက်တော့မှ Modal Dialog Box တစ်ခုအနေနဲ့ ဖော်ပြပေးသွားမှာဖြစ်ပါတယ်။ #new Button ရဲ့ data-target Attribute မှာ ဒီ Structure ရဲ့ ID ဖြစ်တဲ့ #list ကို ညွှန်းထားတဲ့အတွက် ဖြစ်ပါတယ်။ အခုံဆိုရင် Issue List ဖော်ပြမယ့် Table အလွတ်တစ်ခုနဲ့ Issue အသစ် ထည့်သွင်းလိုရတဲ့ Form အတွက် လိုအပ်တဲ့ HTML တွေ ရရှိသွားပါပြီ။ ဆက်လက်ပြီး Issue တစ်ခုစီကို ဖော်ပြရာမှာ အသုံးပြုဖို့အတွက် Template Structure ကို သတ်မှတ် ပေးရပါမယ်။

## Template

```

1. <script type="text/x-template" id="row">
2.   <td><%= subject %></td>
3.   <td><%= config.priority[priority] %></td>
4.   <td><%= config.severity[severity] %></td>
5.   <td><%= assignedTo %></td>
6.   <td>
7.     <select class='form-control input-sm change-status'>
8.       <% _.each(config.status, function(v, i) { %>
9.         <% if(i == status) { %>
10.           <option value='<%= i %>' selected><%= v %></option>
11.         <% } else { %>
12.           <option value='<%= i %>'><%= v %></option>
13.         <% } %>
14.       <% }) %>
15.     </select>
16.   </td>
17.   <td>
18.     <a href="#" class='del'>
19.       <span class='glyphicon glyphicon-remove'></span>
20.     </a>
21.   </td>
22. </script>

```

ဒီ Structure ကို type မှာ text/x-template လိုသတ်မှတ်ထားတဲ့ <script> Element တစ်ခုအနေနဲ့ ထည့်သွင်းထားပါမယ်။ <script> Element အတွက် type အမှန်က text/javascript ဖြစ်ပါတယ်။ အခုံ type မှာ text/x-template လိုပြောထားတဲ့အတွက် Browser ကဒီ type ကို နားမလည်လို JavaScript အနေနဲ့ အလုပ်လုပ်ပေးမှာမဟုတ်သလို HTML အနေနဲ့လည်း ဖော်ပြပေးမှာမဟုတ်ပါဘူး။

JavaScript ကနေ လိုအပ်သလို ယူသုံးဖို့ ရည်ရွယ်ထည့်သွင်းထားခြင်းဖြစ်ပြီး User တွေကို မပြစေလိုတဲ့ Template Structure တွေကို ဒီနည်းနဲ့ ထည့်သွင်းရေးသားကြလေ့ရှိပါတယ်။ ဒီ Template နဲ့ တွဲဖက်အသုံးပြုရ မယ့် Data တို့ကို ပေးလိုက်ရင် Underscore Template Function က လက်တွေ့သုံးလိုရတဲ့ HTML Output ကို ပြန်ပေးမှာဖြစ်ပါတယ်။

Template Structure ကိုလေ့လာကြည့်ရင် <td> Element တွေနဲ့ subject, priority, severity,

assignedTo, status စာတဲ့ Variable တွေကို သူနေရာနဲ့သူ အစားစိုးအသုံးပြုနိုင်ဖို့ သတ်မှတ်ထားတာကို တွေ့နှုန်ပါတယ်။ priority တို့ severity တို့ကို Model မှာ တန်ဖိုးအနေနဲ့သိမ်းတဲ့အခါ 0, 1, 2 စာတဲ့ နံပါတ် တန်ဖိုးတွေနဲ့ သိမ်းမှာပါ။ ဒါပေမယ့် ပြန်လည်ဖော်ပြတဲ့အခါ Priority: 1, Severity: 2 စသဖြင့် နံပါတ်တွေနဲ့ ပြ ရင်အဆင်မပြေပါဘူး။ ဒါကြောင့် သက်ဆိုင်ရာ 0, 1, 2 နံပါတ်တွေအစား ကြိုတင်သတ်မှတ်ထားတဲ့ တန်ဖိုးတွေ နဲ့ အစားစိုးဖော်ပြန်ဖို့အတွက် config Object ကို တစ်ဆင့်ခံအသုံးပြုထားခြင်းဖြစ်ပါတယ်။ ဥပမာ - 0 ဆုံး ရင် Minor, 1 ဆုံး Major စသဖြင့် ကြိုတင်ကြောထားတဲ့ တန်ဖိုးနဲ့ ဖော်ပြန်ဖို့အတွက်ပါ။ config Object ကို Global Variable တစ်ခုအနေနဲ့ အခုလို ကြောထားရမှာ ဖြစ်ပါတယ်။

### JavaScript

```
1. var config = {
2.     priority: ['Normal', 'Important', 'Urgent'],
3.     severity: ['Minor', 'Major', 'Crash'],
4.     status: ['Assigned', 'Doing', 'Done', 'Close']
5. };
```

Template Structure ရဲ့ လိုင်းနံပါတ် (၇) ကိုလေ့လာကြည့်ရင် config.status ကို Loop လုပ်ပြီး <select> Element တစ်ခုရဲ့ <option> တွေအဖြစ် ထည့်သွင်းထားတာကိုတွေ့ရမှာဖြစ်ပါတယ်။ ဒီတော့မှ User က Issue တစ်ခု ရဲ့ Status ကို ပြောင်းချင်ရင် Select Box ကနေတစ်ခါတည်း ရွေးပြီးပြောင်းနိုင်စေဖို့ ရည်ရွယ်ထည့်သွင်းထားခြင်း ဖြစ်ပါ တယ်။ status Variable ရဲ့ တန်ဖိုး နဲ့ <option> ရဲ့ တန်ဖိုးတူတဲ့ အခါ <option> မှာ selected Attribute ထည့်ပေးဖို့ကိုလည်း လိုင်းနံပါတ် (၉) မှာ Condition စစ်ပြီး သတ်မှတ်ထားပါသေးတယ်။ Underscore Template တွေမှာ Variable တန်ဖိုးတွေ ဖော်ပြန့် <%= %> Template Element ကို အသုံးပြုပြီး JavaScript Code တွေ ရောစပ် ရေးသားဖို့အတွက် <% %> Template Element ကို သုံးကြောင်း အထက်မှာလည်း ဖော်ပြခဲ့ပြီးဖြစ်ပါတယ်။ နောက်ဆုံး <td> အတွင်းမှာ .del Class ပေးထားတဲ့ Element တစ်ခုပါဝင်တာ ကိုလည်း သတ်မှတ်ပါ။ Issue ကို ဖျက်ချင် တဲ့အခါ အဲဒီ Element ကိုနှိပ်ပြီးဖျက်စေနိုင်ဖို့ ရည်ရွယ်ထည့်သွင်းထားခြင်းဖြစ်ပါတယ်။

အခုလိုရင် App ရဲ့ အခြေခံ HTML Structure တော့ပြည့်စုံသွားပါပြီ။ Issue List ဖော်ပြန့် Table တစ်ခုရပါပြီ။ Issue အသစ်ထည့်သွင်းဖို့ Form တစ်ခုရပါပြီ။ Issue တွေကို ဖော်ပြရမယ့် Template တစ်ခု လည်း ရရှိသွားပြီ ဖြစ်ပါတယ်။ ဆက်လက်ပြီး JavaScript Code တွေ ရေးသားပါမယ်။ ပထမဆုံးအနေနဲ့ Issue Model Constructor တစ်ခုကို အခုလို တည်ဆောက်ရပါမယ်။

### JavaScript

```
1. var Issue = Backbone.Model.extend({
2.     defaults: {
3.         subject: 'Issue summary',
4.         priority: 1,
5.         severity: 1,
6.         assignedTo: 'none',
7.         status: 1
8.     },
9.
10.    changeStatus: function(value) {
11.        this.save({status: value});
```

```
12. }
13.});
```

Model ၏ Default Data အနေနဲ့ subject, priority, severity, assignedTo နဲ့ status စုံပါဝင်ပြီး status တန်ဖိုးကို ပြောင်းပေးနိုင်တဲ့ changeStatus() Function လည်းပါဝင်ပါတယ်။ ဆက်လက်ပြီး Collection Constructor တစ်ခုကို အခုလို တည်ဆောက်ပေးရပါမယ်။

### JavaScript

```
1. var IssueList = Backbone.Collection.extend({
2.   url: '#',
3.   model: Issue
4. });
5.
6. var Issues = new IssueList();
```

Collection Constructor မှာ url အနေနဲ့ ဘာမှမသတ်မှတ်လိုတဲ့အတွက် # Sign ကိုပေးထားပါတယ်။ Server နဲ့ ဆက်သွယ်တဲ့အလုပ်တွေ မလုပ်လိုတဲ့အတွက် URL ကို မသတ်မှတ်တော့ခြင်းဖြစ်ပါတယ်။ model အတွက် စောစောက တည်ဆောက်ခဲ့တဲ့ Issue ကို သတ်မှတ်ပေးထားပါတယ်။ Constructor တည်ဆောက်ပြီး Issues အမည်နဲ့ Collection Object တစ်ခုကိုပါ လိုင်းနံပါတ် (၆) မှာ တစ်လက်စတည်း တည်ဆောက်ထားတာကို တွေ့ရမှာဖြစ်ပါတယ်။

ဆက်လက်ပြီး Issue List Table ထဲမှာ ဖော်ပြရမယ့် Issue တစ်ခုစီအတွက် View Constructor ကို ရေးသားပါမယ်။

### JavaScript

```
1. var IssueView = Backbone.View.extend({
2.   tagName: 'tr',
3.
4.   events: {
5.     'change .change-status': 'changeStatus',
6.     'click .del': 'delSelf'
7.   },
8.
9.   template: _.template( $('#row').html() ),
10.
11.   initialize: function() {
12.     this.listenTo(this.model, 'change', this.render),
13.     this.listenTo(this.model, 'destroy', this.remove)
14.   },
15.
16.   render: function() {
17.     var modelData = this.model.toJSON();
18.     modelData.config = config;
19.
20.     this.$el.html( this.template(modelData) );
21.     return this;
22.   },
23. });
```

```

23.
24.     changeStatus: function() {
25.         this.model.changeStatus( this.$('.change-status').val() );
26.     },
27.
28.     delSelf: function() {
29.         this.model.destroy();
30.     }
31. });

```

Issue တွေကို <tr> Element နဲ့ဖော်ပြချင်တဲ့အတွက် tagName အတွက် tr ကို သတ်မှတ်ပေးထားပါတယ်။ Event အနေနဲ့ Status Select Box မှာတန်ဖိုးပြောင်းတဲ့အခါ changeStatus() Function ကိုအလုပ်လုပ်ပေး ဖို့ သတ်မှတ် ပေးထားပါတယ်။ လိုင်းနံပါတ် (၂၂) ကိုလေ့လာကြည့်ရင် ဒါ View က Model ၏ changeStatus() Function ကိုပဲ တစ်ဆင့်ခေါ်ပူးပူးပြေထား တယ်ဆိုတာကို တွေ့ရနိုင်ပါတယ်။ View ထဲ မှာပဲတစ်ခါတည်းရေးလဲ ရပါတယ်။ ဒါပေမယ့် MVC ရေးထုံးအရ View ဟာ User Interface ဖော်ပြမှုပိုင်းကိုသာ တာဝန်ယူသင့်ပြီး၊ Logic ပိုင်း လုပ်ဆောင်ချက် တွေကို Model ကသာတေဝန် ယူဆောင်ရွက်ရပါတယ်။

View Object တစ်ခုတည်ဆောက်တဲ့အခါ တဲ့ဖော်သုံးရမယ့် Model Object ကို သတ်မှတ်ပေးရပါတယ်။ ဥပမာ -

### JavaScript

```

var issueOne = new Issue();
var view = new IssueView({model: issueOne});

```

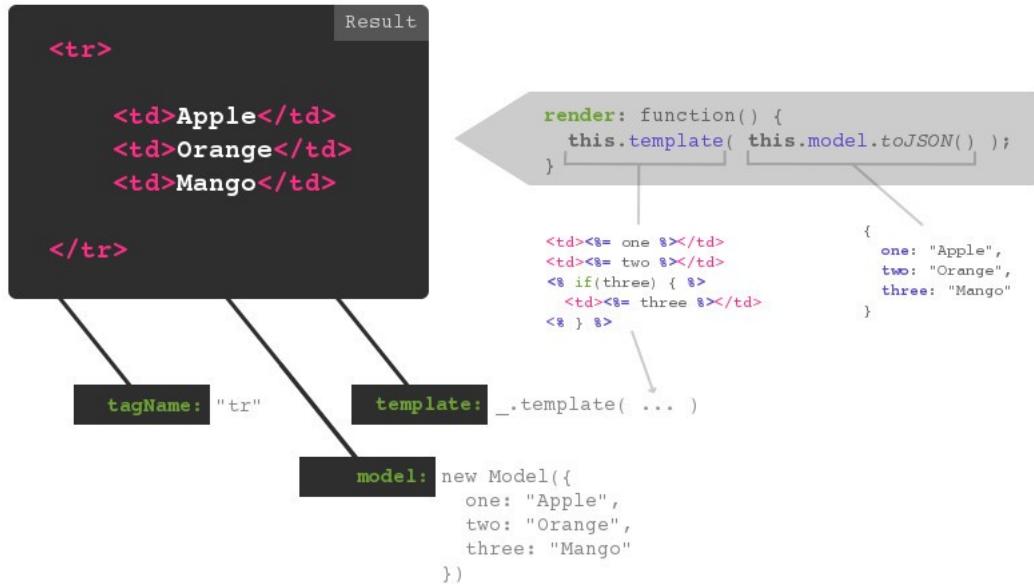
this.model ဆိုတာ အဲဒီလို View နဲ့တဲ့သုံးဖို့ သတ်မှတ်ထားတဲ့ Model Object ကို ပြောတာပါ။

Event အနေနဲ့ .del ကိုနှိပ်ရင် delSelf() Function ကို အလုပ်လုပ်ပေးဖို့လည်း သတ်မှတ်ထားပါတယ်။ လိုင်းနဲ့ပါတ် (၂၉) ကိုလေ့လာကြည့်ပါ။ delSelf() Function ကို this.model ကို ပယ်ဖျက်လိုက်ခြင်းဖြစ်ပါတယ်။

ဆက်လက်ပြီး လိုင်းနံပါတ် (၉) မှာ template Property အတွက် ကြိုတင်ရေးသားထားခဲ့တဲ့ Template ကို Underscore Template Function အဖြစ်သတ်မှတ်ပေးထားပါတယ်။ လိုင်းနံပါတ် (၁၁) က initialize Function ထဲမှာတော့ listenTo() ကိုသုံးပြီး Model မှာ change တစ်ခုချရှိတဲ့ အခါ render() Function ကို Run ပေးဖို့ သတ်မှတ်ထားသလို၊ Model ကို destroy လုပ်လိုက်တဲ့အခါ remove() Function ကို Run ပေးဖို့သတ်မှတ် ထားပါတယ်။

ဒီနေရာမှာ remove() ဆိုတာ jQuery remove() Function ကိုပြောတာပါ။ View Object တစ်ခု တည်ဆောက် လိုက်တာနဲ့ အဲဒီ Object နဲ့အတူ jQuery Function တွေ တစ်ခါတည်းတဲ့ဖော်အသုံးပြုနိုင်အောင် Backbone က စီစဉ်ပေးထားပါတယ်။ ဒါကြောင့် remove() Function ထပ်ရေးစရာမလိုပါဘူး။ jQuery remove() Function က လက်ရှိ View Element ကို DOM Structure ကနေ ဖယ်ထဲတဲ့ပေးသွားမှာဖြစ်ပါတယ်။ change အတွက် သတ်မှတ်ထားတဲ့ render() Function ကိုတော့ ကိုယ်တိုင် ရေးသားရပါတယ်။ လိုင်း

ຳປິຕ່ (ຄົງ) ມີ `render()` Function ກີ່ ວັດທຸກຄະຫຼາດ ພິຕຍົດ|| `render()` ສີ `Template` ເປີດໃຈ ແລ້ວ ອະລຸດ ອຸປະກຸດ ກີ່ ດີ (ຄົງ-2) ມີ ເໝີ້ປຸ່ ພິຕຍົດ|| ເລື່ອງຕັ້ງ `Template` ເປີດໃຈ ແລ້ວ ອະລຸດ ອຸປະກຸດ ກີ່ ດີ (ຄົງ-2) ມີ ເໝີ້ປຸ່ ພິຕຍົດ||



Ü (8.2) - Backbone View Structure

template Property ထဲမှာသိမ်းထားတဲ့ Underscore Template ကို this.template နဲ့ပြန်လည်ရယူ အသုံးပြု ပါတယ်။ Template နဲ့အတူအသုံးပြုရမယ့် Data ကို this.model.toJSON() သုံးပြီးပေးလိုက်တဲ့ အခါ ကျွန်ုတ်တော် တို့ လိုချင်တဲ့ HTML Structure ရလဒ်ကို ရရှိမှာပဲဖြစ်ပါတယ်။

View Constructor Code နမူနာရဲ့ လိုင်းနံပါတ် (၁၈) ကို သွားလေ့လာကြည့်ရင် ထူးခြားချက်အနေနဲ့ ကြိုတင် ကြော်လာသူတဲ့ config ကို Model JSON Object နဲ့ တွဲထည့်ပေးလိုက်တာကို တွေ့ရမှာဖြစ်ပါတယ်။ ဒီနည်းနဲ့ Template ထဲမှာ Model Data တွေသာမက config ကိုပါ အသုံးပြုလို ရရှိသွားခြင်း ဖြစ်ပါတယ်။ Template Code နမူနာမှာ ပြန်လေ့လာကြည့်ပါ။ subject, priority, severity စာတဲ့ Model Data တွေအပြင် config ကိုလည်း အသုံးပြုလာသူတာကို တွေ့ရပါလိမ့်မယ်။ ဆက်လက်ပြီး App View Constructor ကို ဆက်လက်ရေးသားရပါမယ်။

JavaScript

```
1. var AppView = Backbone.View.extend({  
2.     el: 'body',  
3.     events: {  
4.         'click #submit': 'addIssue'  
5.     },  
6.     initialize: function() {  
7.         this.collection = new IssuesCollection();  
8.         this.render();  
9.     },  
10.    addIssue: function() {  
11.        var issue = {  
12.            title: this.$('#title').val(),  
13.            description: this.$('#description').val(),  
14.            priority: this.$('#priority').val()  
15.        };  
16.        this.collection.add(issue);  
17.        this.$('#title').val('');  
18.        this.$('#description').val('');  
19.        this.$('#priority').val('');  
20.    }  
21.});
```

```

9.         this.listenTo(Issues, 'add', this.addRow);
10.    },
11.
12.    addIssue: function() {
13.        var newIssue = new Issue({
14.            subject: $('#subject').val(),
15.            priority: $('#priority').val(),
16.            severity: $('#severity').val(),
17.            assignedTo: $('#assigned-to').val(),
18.            status: $('#status').val()
19.        });
20.
21.        Issues.add(newIssue);
22.    },
23.
24.    addRow: function(issue) {
25.        var view = new IssueView({ model: issue });
26.        $('#list').append( view.render().el );
27.
28.        $('#subject, #assigned-to').val('');
29.        $('#priority, #severity, #status').val(0);
30.        $('#new').modal('hide');
31.    }
32. });

```

ဒါ View အတွက်တော့ Element အသစ်မတည်ဆောက်တော့ပါဘူး။ App တစ်ခုလုံးနဲ့သက်ဆိုင်တဲ့ View ဖြစ်တဲ့ အတွက် <body> Element ကိုပဲ အသုံးပြုပါမယ်။ ဒါကြောင့် tagName Property ကို မသုံးပဲ လိုင်းနဲ့ပါတယ် (၂) မှာ el Property နဲ့ body ကို အောင်နှုန်းထားပေးခြင်းဖြစ်ပါတယ်။ Event အနေနဲ့ Form Dialog Box ထဲက #submit Button ကိုနှိပ်လိုက်ရင် addIssue() Function ကို အလုပ်လုပ်ပေးဖို့ သတ်မှတ်ထားပါတယ်။ #new Button ကို နှိပ်လိုက် ရင် Form Dialog Box ပေါ်လာဖို့ကို ကျွန်တော်တို့ ထပ်ရေးစရာမလိုပါဘူး။ Bootstrap က လုပ်ပေးပါလိမ့်မယ်။

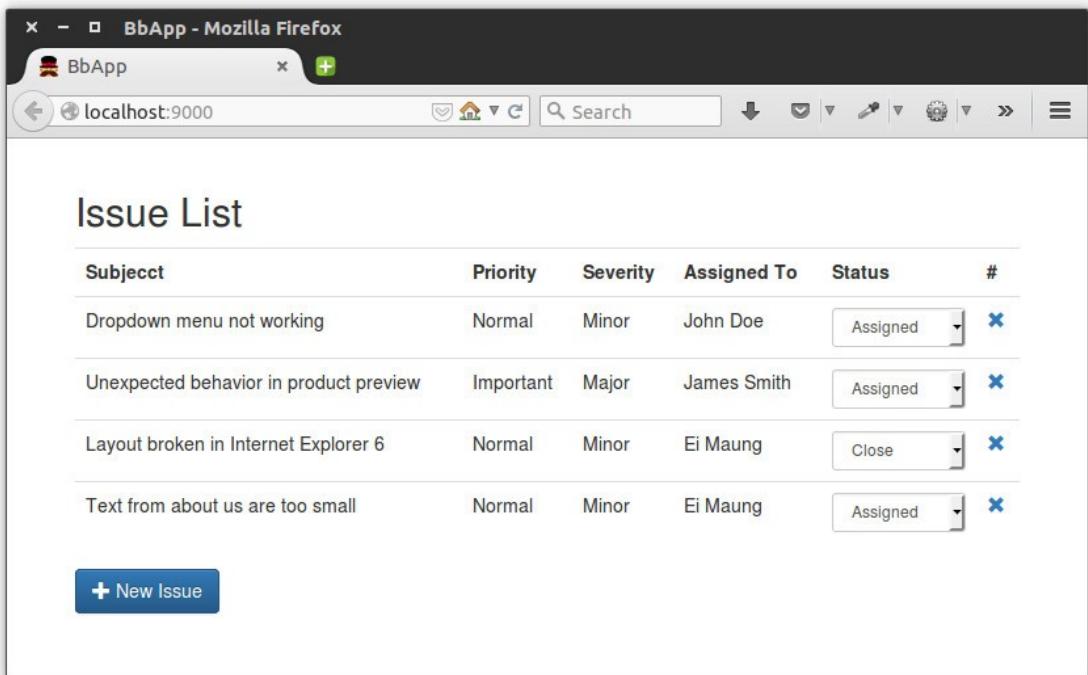
လိုင်းနဲ့ပါတယ် (၁၂) မှာ addIssue() Function ကိုသတ်မှတ်ထားပါတယ်။ Form Input တွေမှာ ရှိက်ထည့်ရွှေးချယ် ထားတဲ့ တန်ဖိုးတွေကို အသုံးပြုပြီး Issue Model တစ်ခုကို အရင်တည်ဆောက်ပါတယ်။ ပြီးတော့မှာ Issues Collection ထဲကို add နဲ့ ထည့်သွင်းပေးလိုက်ခြင်းဖြစ်ပါတယ်။ လိုင်းနဲ့ပါတယ် (၁၃) ကို လေ့လာကြည့်ရင် listenTo() နဲ့ Issues Collection မှာ Model တစ်ခုတိုးလာတိုင်း addRow() Function ကို အလုပ်လုပ်ပေးဖို့ သတ်မှတ်ထား တာကို တွေ့ရမှာဖြစ်ပါတယ်။ လိုင်းနဲ့ပါတယ် (၁၄) က addRow() Function ကိုလေ့လာကြည့်ရင် IssueView Object တစ်ခု တည်ဆောက်ပြီး ရရှိလာတဲ့ရလဒ်ကို jQuery append() Function သုံးပြီး #list Table ထဲမှာ ထည့်သွင်းပေးထားတာကို တွေ့ရမှာ ဖြစ်ပါတယ်။

ဒါတော့ #new ကိုနှိပ်လိုက်ရင် Form Dialog Box ပေါ်လာမယ်။ Input တွေထဲမှာ တန်ဖိုးတွေဖြည့် ပြီး #submit ကို နှိပ်လိုက်ရင် Issues Collection ထဲကို Model အသစ်တစ်ခု ရောက်ရှိသွားပါလိမ့်မယ်။ ဒီလို Model အသစ်တစ်ခု ရောက်တိုင်း IssueView တစ်ခုတည်ဆောက်ပြီး ရရှိလာတဲ့ရလဒ်ကို #list ထဲကို ထည့်သွင်းပေးသွားမှာပဲ ဖြစ်ပါ တယ်။

IssueView မှာ Status Select Box တဲ့က Status တစ်ခုကို ရွေးလိုက်ရင် Model တဲ့က Status တန်ဖိုး လိုက်ပြောင်းပေးတဲ့ လုပ်ဆောင်ချက်က ရေးသားထားပြီးဖြစ်သလို၊ .del ကို နှိပ်လိုက်ရင် သက်ဆိုင်ရာ Model ကိုပယ်ဖျက်ပေးတဲ့ လုပ်ဆောင်ချက်ကလည်း ကြိုတင်ရေးသားထားပြီးဖြစ်ပါတယ်။ ဒါကြောင့် အခုန်ရင် Issue တွေထည့် သွင်းခြင်း၊ ထည့်သွင်းထားတဲ့ Issue တွေ ကို Table တစ်ခုနဲ့ဖော်ပြခြင်း၊ Issue Status ပြောင်းခြင်း၊ Issue တွေကို ပယ်ဖျက်ခြင်း စတဲ့လုပ်ငန်းတွေကို ဆောင်ရွက်ပေး နိုင်တဲ့ App တစ်ခု ရရှိသွားပြီဖြစ်ပါတယ်။ App စတင်အလုပ်လုပ် နှင့်ဖို့အတွက် AppView Object တစ်ခုတည်ဆောက်ပေးဖို့ ကျွန်ုပါသေးတယ်။ အခုလို ရေးသားပေးရမှာ ဖြစ်ပါတယ်။

```
1. $(function() {
2.     new AppView();
3. });
```

အားလုံးပြည့်စုံပြီဖြစ်လို့ grunt serve Command နဲ့ App ကို Run ကြည့်လိုရပါပြီ။ ပုံ (၉.၄) မှာ ပြထားသလို ရလဒ်ကို ရရှိမှာဖြစ်ပါတယ်။ New Issue Button ကို နှိပ်ပြီး Issue တွေ စတင်ထည့်သွင်း စမ်းသပ်နိုင်ပါတယ်။



ပုံ (၉.၄) - Example Backbone App - Issue List

Code တွေကို ရေးသားစမ်းသပ်တဲ့အခါ HTML Document တဲ့မှာ jQuery, Underscore, Backbone, Bootstrap စတဲ့ Library တွေအတွက် လိုအပ်တဲ့ JavaScript နဲ့ CSS ဖိုင်တွေ ချိတ်ဆက်ထားပေးဖို့ လိုတယ်ဆိုတာကို သတိပြုပါ။ နှမူနာ Code အပြည့်အစုံကို အောက်ပါလိပ်စာမှာ ရယူနိုင်ပါတယ်။

<https://github.com/eimg/backbone-example/archive/master.zip>

အပိုင်း (၄) ရောက်တဲ့ အခါ User Registration တွေ၊ Server နဲ့ ချိတ်ဆက်ပြီး အချက်အလက်တွေ သိမ်းဆည်းပေးတဲ့ လုပ်ဆောင်ချက်တွေ ပါဝင်တဲ့ Issue Management System တစ်ခုကို ဆက်လက်တည်ဆောက်သွားမှာ ပဲဖြစ်ပါတယ်။

## Conclusion

Backbone ဟာ လက်ရှိဒီစာရေးသားနေ့ချိန်မှာ ခုတိယမြောက်လူ သုံးအများဆုံး JavaScript MVC Framework တစ်ခု ဖြစ်ပါတယ်။ Backbone ထက် ပိုအသုံးများတာကတော့ AngularJS လို့ ခေါ်တဲ့ Framework ပဲဖြစ်ပါတယ်။ Framework အရွယ်အစားအရ ပိုမိုကြီးမားသလို ပါဝင်တဲ့ လုပ်ဆောင်ချက်တွေလည်း ပိုမိုစုံလင်ပါတယ်။ ဒါပေမယ့် Angular ဟာ Opinionated Framework တစ်ခုဖြစ်ပါတယ်။ Opinionated Framework ဆိတာ Code ရေးသားရာမှာ လိုက်နာသင့်တဲ့ ရေးထုံးတွေ၊ လိုက်နာသင့်တဲ့ နည်းစနစ်တွေကို မဖြစ်မနေလိုက်နာဖို့ သတ်မှတ်ထားတဲ့ Framework အမျိုး၊ အစားဖြစ်ပါတယ်။ Code ဖွဲ့စည်းပုံကောင်းပြီး ရေရှည် Maintenance ကောင်းစေဖို့ ဆိုတဲ့ ရည်ရွယ်ချက်နဲ့ ဖြစ်ပါတယ်။ Backbone ကတော့ Opinionated Framework မဟုတ်ပါဘူး။ MVC Pattern နဲ့ JavaScript Code တွေ ရေးလို့ရ အောင် ပေးထားတဲ့ ရိုးရှင်းတဲ့ အခြေခံ Foundation တစ်ခုသာ ဖြစ်ပါတယ်။ ဒီ Framework ကိုသုံးပြီး လက်တွေ Code တွေ ရေးသားရာမှာ၊ ဘယ်လိုရေးထုံးတွေနဲ့ ရေးသားသူ ဘယ်လိုနည်းစနစ်တွေကို သုံးမလဲဆိုတာကို ရေးသားသူ Developer ကိုယ်တိုင် ဆုံးဖြတ်ရွေးချယ်ရေးသားနိုင်ပါတယ်။

Opinionated Framework တွေဟာ Code အရည်အသွေးကောင်းဖို့ အတွက် အထောက်အကူဖြစ်စေပေမယ့် တစ်ခါ တစ်ရုံမှာ သူရဲ့ ကန့်သတ်ချက်တွေကြောင့် Code ရေးသားရတာ မလွတ်လပ်ပဲ၊ အလုပ်မတွင်ဖြစ်တက်ပါတယ်။ စိတ်ကူး ရှိသလို ရေးသားဖန်တီးဖို့ ထက် သတ်မှတ်ချက်နဲ့ ကိုက်ညီအောင် လိုက်ရေးပေးနေရတာက်ပါတယ်။ ဒါကြောင့် ကျွန်းတော့ တစ်ကိုယ်ရေအနေနဲ့ Opinionated Framework မဟုတ်တဲ့ လွပ်လပ်ရိုးရှင်းတဲ့ Framework တွေကိုသာ ပိုမိုနှစ်သက် ပါတယ်။ မိမိနှစ်သက်တဲ့ ရေးထုံးနည်းစနစ်ကို သုံးနိုင်လို့ အလုပ်ပိုတွင်ပါတယ်။ ပြုပြင်ထိမ်းသိမ်းရလွယ်စေဖို့ ရေးသားသူ Developer ရဲ့ ကျမ်းကျင့်မှုပေါ် မူတည်သွားပေမယ့်၊ SOLID, MVC, TDD စတဲ့ နည်းစနစ်ကောင်းပဲဟုသုတေသန ရှိထား ပြီးသား 10x Programmer တစ်ယောက်အနေနဲ့ မိမိရေးသားတဲ့ Code က ပြုပြင် ထိမ်းသိမ်းရလွယ်အောင် မိမိကိုယ်တိုင် စီမံရေးသားနိုင်မှာပါ။ ဒါကြောင့် ဒီစာအုပ်မှာ JavaScript MVC အတွက် အသုံးအများဆုံးဖြစ်နေတဲ့ Angular ကို မဖော် ပြုပဲ ပိုပြီး ရိုးရှင်းလွပ်လပ်တဲ့ Backbone ကို ရွေးချယ်ဖော်ပြခဲ့ခြင်းဖြစ်ပါတယ်။ စာဖတ်သူ အနေနဲ့ AngularJS, EmberJS, KnockoutJS စတဲ့ အခြား JavaScript MVC တွေကို အခွင့်ရရင်တော့ ဆက်လက်လေ့လာသွားသင့်ကြောင်း အကြံပြုလို ပါတယ်။

Backbone ကို DocumentCloud, Hulu, Wordpress, Foursquare, Bitbucket, Disqus, Delicious, Khan Academy, Basecamp, Airbnb, Pandora, Trello စတဲ့ အသုံးပြုသူ သန်းချိရှိတဲ့ အထင်ကရ App တွေက အသုံးပြုထားတဲ့ အတွက် Scalable ဖြစ်ကြောင့် Proven ဖြစ်ပြီးသား Framework တစ်ခုလည်းဖြစ်ပါတယ်။ ဒါကြောင့် Rockstar Developer တစ်ဦး သိရှိထားသင့်တဲ့ Modern Development Stack ထဲက ပထမဆုံးနည်းပညာတစ်ခုအနေနဲ့ ဖော်ပြပေး လိုက်ရခြင်းပဲ ဖြစ်ပါတယ်။

CPU တစ်ခုဟာ တစ်စက္ကန်မှာ Instruction ပေါင်းသန်းချိ  
အလုပ်လုပ်နိုင်ပေမယ့်၊ Input/Output Device တွေကတော့  
အဲဒီလောက်မြန်မြန် အလုပ်မလုပ်နိုင်ကြပါဘူး။ စွမ်းဆောင်ရည်မြင့်  
ပရီဂရမ်တွေရရှိဖို့အတွက် ဒီအချက်ကိုသတိပြုရပါမယ်။

### **Professional Web Developer Course**

HTML5, PHP/MySQL, jQuery/Ajax, Mobile Web စသည့်  
Professional Web Developer တစ်ဦး သိရှိထားသင့်သည့်  
နည်းပညာများကို စုစည်းသင်ကြားခြင်းဖြစ်သည်။  
ဆက်သွယ်ရန် - (၀၉) ၇၃၀ ၆၅၅ ၆၂

## အခန်း(၁၀) – NodeJS (Server-side JavaScript)

JavaScript ဟာ မူလအစက Web Browser တစ်ခုအတွင်းမှာ အလုပ်လုပ်တဲ့ Client-side Script Language တစ်ခုသာ ဖြစ်ပါတယ်။ Client-side မှာ JavaScript ကို အမိကလုပ်ငန်းနှစ်မျိုးအတွက် အသုံးပြုခဲ့ကြပါတယ်။ Document Object Model (DOM) API အကူအညီနဲ့ HTML Element တွေကို စီမံခြင်းအားဖြင့် Interactive User Interface တွေ ဖန်တီးဖို့နဲ့ XMLHttpRequest (XHR) API အကူအညီနဲ့ Server နဲ့ ဆက်သွယ်ပြီး UI Client တွေ ဖန်တီးဖို့ပဲဖြစ်ပါတယ်။ HTML5 နည်းပညာထွက်ပေါ်လာတဲ့အချင့်မှာတော့ Media API, Web Storage API, Web Socket API, File API, History API, Web GL API စသဖြင့် App Development အတွက် အရေးပါတဲ့ JavaScript API များ ပါဝင်လာခြင်းနဲ့အတူ JavaScript ရဲ့ အခန်းကဏ္ဍာဟာ ပိုပြီးအရေးပါလာခဲ့ပါတယ်။

၂၀၀၉ ခုနှစ်ထဲမှာ Ryan Dahl လို့ခေါ်တဲ့ System Engineer တစ်ဦးက NodeJS ကို တိတွင်လိုက်ချိန်မှာတော့ JavaScript ဟာ Client-side Scripting Language သာမကတော့ပဲ ကဏ္ဍာမှာ အသုံးချလို့ရတဲ့ General Purpose Programming Language တစ်ခုဖြစ်လာခဲ့ပါတယ်။ အဲဒီအချင့်က Web Application ဆိတာ Application Development အတွက် အမိက Platform တစ်ခုဖြစ်နေခဲ့တဲ့အတွက် NodeJS ကို Web Application များအတွက် အနာဂတ် Server-side နည်းပညာအဖြစ် မျော်မှန်းလာခဲ့ကြပါတယ်။ Web Developer တွေဟာ မူလက မည် သည့် Server-side နည်းပညာကို အသုံးချနေသည်ဖြစ်စေ၏ Client-side အတွက် JavaScript ကိုသာ အသုံးပြုကြရတာပါ။ တစ်နည်းအားဖြင့် Server-side အတွက်သုံးရတဲ့ Language က တစ်ခု၊ Client-side အတွက် သုံးရတဲ့ Language ကတစ်ခု နှစ်ခုခဲ့ပြီးအသုံးပြုကြရတဲ့သော ဖြစ်ပါတယ်။ NodeJS တွက်ပေါ်လာတဲ့အတွက် Client-side ရော့ Server-side အတွက်ပါ JavaScript ဆိတာ တစ်မျိုးတည်းကို အသုံးပြုတည်ဆောက်နိုင်တော့မယ်ဆိုတဲ့ အလား အလာဟာ အတော်လေး စိတ်ဝင်စားစရေကောင်းခဲ့ပါတယ်။

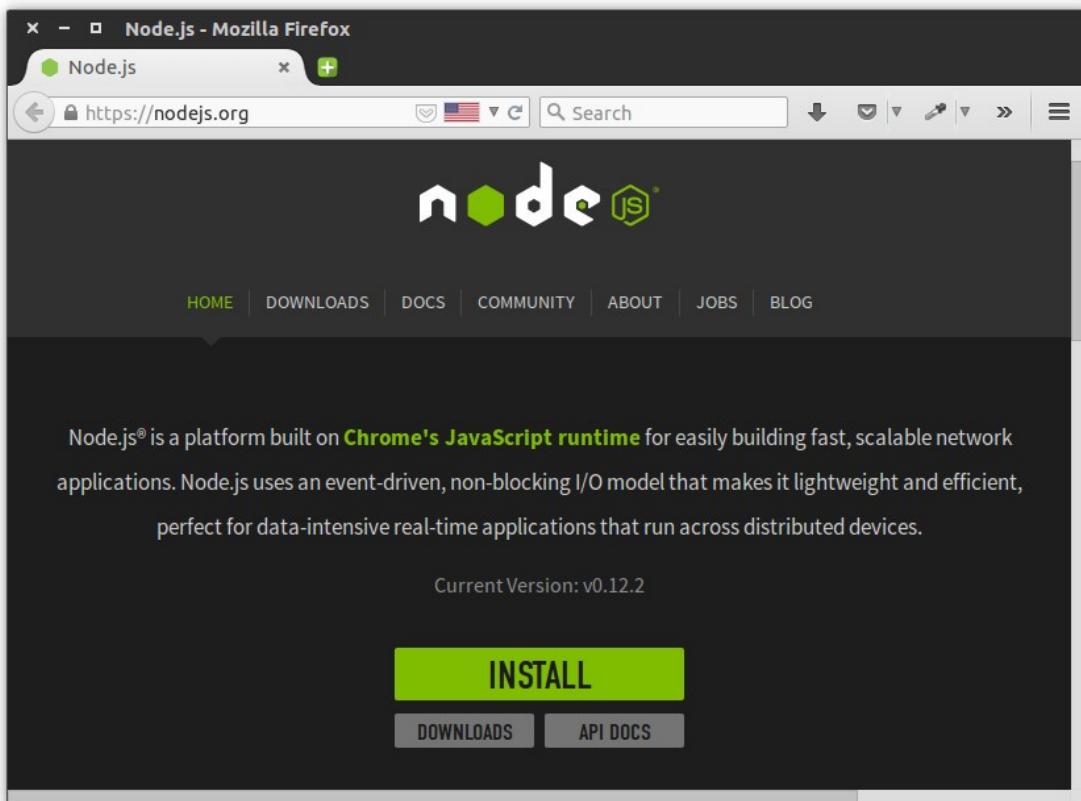
ဒါပေမယ့် လက်တွေမှာ NodeJS ဟာ Server-side နည်းပညာတစ်ခုဆိတာထက် ပိုပါတယ်။ NodeJS ဟာ Event Driven/Non-blocking IO Model နည်းစနစ်ကို အသုံးပြုထားပြီး အခြား Platform တွေနဲ့မတူ တစ်မူထူးခြားတဲ့ နည်းပညာ တစ်ခုဖြစ်ပါတယ်။ Real-time Network App တွေနဲ့ Data ပမာဏများစွာကို ကိုင်တွယ် စီမံရတဲ့ I/O Heavy App တွေ တည်ဆောက်ရာမှာ အထူးကောင်းမွန်တဲ့ နည်းပညာတစ်ခုဖြစ်ပါတယ်။ ဒါအခန်းရဲ့ ခေါင်းစဉ်မှာ NodeJS (Server-side JavaScript) ခေါင်းစဉ်တပ်ထားပေမယ့် Server-side ဆိတာ Web App တွေ အတွက်တင်မကပဲ အခြား App အမျိုးမျိုးတို့အတွက် Back-end Server နဲ့ Socket Server ပရိုဂရမ တွေ ဖန်တီးရာမှာအသုံးပြုနိုင်တဲ့ နည်းပညာ ဖြစ်တယ်ဆိုတဲ့ အမိပါယ်ပဲဖြစ်ပါတယ်။

NodeJS ကို တိတွင်ခဲ့တဲ့ Ryan Dahl ဆိတာ Joyent လို့ခေါ်တဲ့ Cloud နည်းပညာလုပ်ငန်းတစ်ခုမှာ အလုပ်လုပ်နေသူဖြစ်ပါတယ်။ တစ်နည်းအားဖြင့် NodeJS ဟာ Open Source နည်းပညာတစ်ခုဖြစ်ပေမယ့် အမိက Sponsor အဖြစ် Joyent က ပုံပိုးပေးနေခြင်းဖြစ်ပါတယ်။ ဒီစာရေးသားနေချိန်မှာ နောက်ဆုံးထွက်ရှိထားတဲ့

NodeJS Version က 0.12.2 ဖြစ်ပါတယ်။ Version 1.0 လို့ Major Release အနေနဲ့ နံပါတ်မတပ်သေးပေမယ့် NodeJS ပေါ်မှာအခြေခံထားတဲ့ ဆက်စပ်နည်းပညာပေါင်းများစွာ ပေါ်ထွက်နေပြီဖြစ်သလို၊ Microsoft, Paypal, Yahoo, SAP စတဲ့ လုပ်ငန်းကြီးတွေက အစ NodeJS ကို စတင်အသုံးပြုနေကြပြီ ဖြစ်ပါတယ်။ **အခန်း (၈)** မှာ ဖော်ပြခဲ့တဲ့ Yeoman, Bower, Grunt စတဲ့ နည်းပညာတွေဟာလည်း NodeJS ကို အသုံးပြု ဖန်တီးထားခြင်း ဖြစ်ပါတယ်။

ဒီစာရေးသားနေဂျိန်မှာ NodeJS Project ကို ဦးဆောင်နေသူက Ryan Dahl မဟုတ်တော့ပါဘူး။ Timothy J. Fontaine ဆိုသူက ဦးဆောင်နေပါပြီ။ Joyent ဆိုတဲ့ လုပ်ငန်းအဖွဲ့အစည်းက ပုံပိုးပေးနေတဲ့ Project ဖြစ်တဲ့ အတွက် တစ်ချို့ Project နဲ့ ပက်သက်တဲ့ Decision တွေမှာ ဒီအဖွဲ့အစည်း၏ စွက်ဖက်လွမ်းမိုးမှာ အတိုင်းအတာ တစ်ခုထိ ရှုနေပုံ ပါတယ်။ ဒါကြောင့် ၂၀၁၄ ခုနှစ်တဲ့မှာ Fedor Indutny ဆိုသူက NodeJS Project ကို **io.js** အမည်နဲ့ Fork လုပ် ယူပြီး ဆက်လက်ဆောင်ရွက် လာခဲ့တာ အခုန့်ရင် အတော်လေး အရှိန်ရနေပြီဖြစ်ပါတယ်။ ဒါကြောင့် အခုအချိန်မှာ NodeJS နဲ့ **io.js** ဆိုပြီး မူလ Code Base တူတဲ့ Project နှစ်ခု အပြောင်ရှုနေတယ်လို့ ဆိုနိုင်ပါတယ်။

NodeJS ဟာ V8 လိုခေါ်တဲ့ JavaScript Engine ပေါ်မှာ အခြေခံ တည်ဆောက်ထားခြင်းဖြစ်ပါတယ်။ V8 ဆိုတာ Google က Google Chrome Browser အတွက် ရည်ရွယ်တိတောင်ထားတဲ့ စွမ်းဆောင်ရည်မြင့် JavaScript Engine တစ်ခုဖြစ်ပါတယ်။



ဖြည့်စွက်ချက် ။ ။ ဒီစာအုပ်ထွက်ပြီး မကြာခင်မှာပဲ NodeJS Community အတွင်းမှာ ထူးခြားတဲ့ ပြောင်းလဲမှုပေါင်းများစွာ ဖြစ်ပေါ့ပါတယ်။ NodeJS နဲ့ io.js တို့ဟာ ပြန်လည်ပေါင်းစပ် သွားကြပြီး ပထမဆုံး NodeJS Major Version အဖြစ် NodeJS 4.0 ကို ကြော်လှုပါတယ်။ Joyent ကိုလည်း Samsung က ဝယ်ယူလိုက်လို အခုခိုရင် Samsung ပိုင်လှုပ်ငန်း ခဲ့တစ်ခု ဖြစ်သွားပါဖြီ။ ဒီဖြည့်စွက်ချက်ကို ထည့်သွင်းချိန်မှာဆိုရင် NodeJS ဟာ Version 6.2.2 အထိကို ရောက်ရှိသွားခဲ့ပြီ ဖြစ်ပါတယ်။

## 10.1 – Non-blocking I/O

I/O ဆိုတာ Input/Output ကိုပြောတာပါ။ Hard drive ပေါ်က အချက်အလက်တွေကို ဖတ်ရှုခြင်း၊ ရေးသားခြင်း ဟာ I/O လုပ်ငန်းဖြစ်ပါတယ်။ Database Table ထဲက အချက်အလက်တွေကို ရယူခြင်း၊ ထည့်သွင်းခြင်းဟာ I/O လုပ်ငန်း ဖြစ်ပါတယ်။ Server ကို ဆက်သွယ်ပြီး အချက်အလက်တွေ ရယူခြင်း၊ ပေးပိုခြင်းဟာ I/O လုပ်ငန်းဖြစ် ပါတယ်။ Finger Print, GPS, Accelerometer, Gyroscope စတဲ့ Sensor Device တွေကနေ အချက်အလက်တွေ ရယူခြင်းဟာ I/O လုပ်ငန်းဖြစ်ပါတယ်။ ပရိုဂရမ်တစ်ခုမှာ ပါဝင်တဲ့ Instruction တွေကို Processes လုပ်ရတာ ဖြစ်ပါတယ်။ I/O က Process လုပ်ငန်းနဲ့ယှဉ်ရင် အများကြီးနေးကွားပါတယ်။ ပရိုဂရမ်တစ်ခုဟာ အဆင့်လိုက် အလုပ်လုပ်တဲ့ Serialize ပုံစံ အလုပ်လုပ်တယ်ဆိုရင် Processes က I/O ကို စောင့်နေရတဲ့ အတွက် မလိုအပ်ပဲ ပရိုဂရမ်ရွှေ့စွမ်းဆောင်ရည် ကျဆင်းရပါတယ်။

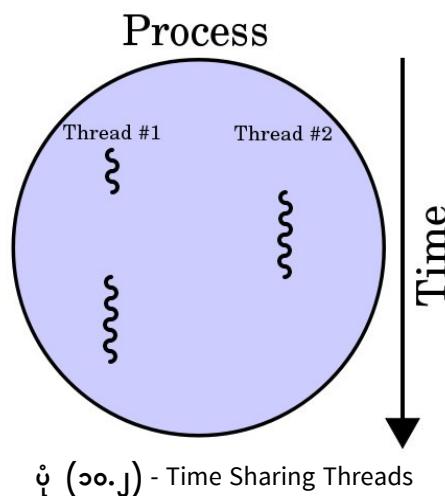
ဥပမာ - ပရိုဂရမ်တစ်ခုဟာ အချက်အလက်တွေအများကြီးပါတဲ့ ဖိုင်တစ်ခုကိုဖွင့်ပြီး အလုပ်လုပ်ဖို့လိုတယ်ဆိုကြပါစို့။ ဖိုင်ကို ဖွင့်ပြီး အချက်အလက်တွေ ဖတ်ယူတဲ့ လုပ်ငန်းမပြီးမချင်း ပရိုဂရမ်က တစ်ခြားအလုပ်တွေကို လုပ်နိုင်မှာ မဟုတ်ပါဘူး။ ဖိုင်ကိုဖွင့်ဖတ်တဲ့ အချိန်ဟာ တစ်စတ္တနဲ့လည်း ဖြစ်နိုင်ပါတယ်။ နှစ်စတ္တနဲ့လည်း ဖြစ်နိုင်ပါတယ်။ စတ္တနဲ့ဆုံးဆယ်လည်း ဖြစ် နိုင်ပါတယ်။ ဖိုင် Size ပေါ်မှုတည်ပါတယ်။ တစ်စတ္တနဲ့မှာ Instruction ပေါင်း (၁၀) သိန်းလောက်ကို အလုပ်လုပ်နိုင်တဲ့ CPU အတွက် တစ်စတ္တနဲ့ နှစ်စတ္တနဲ့ ဆိုတဲ့ အချိန်ဟာ အတော်ကြီးမှာ တဲ့ အဟန်အတားတစ်ခုပါ။

Process တွေက I/O လုပ်ငန်းတွေကို စောင့်နေရာမလိုအောင် စီမံရေးသားတဲ့ နည်းစနစ်ကို Non-blocking I/O လို ခေါ်တာပါ။ Asynchronous I/O လိုလည်း ခေါ်ပါတယ်။ တစ်ခုနဲ့တစ်ခု စောင့်စရာမလိုပဲ ပြုင်တူအလုပ်လုပ်နိုင်တဲ့ အတွက် Concurrency လိုလည်း ခေါ်ပါတယ်။

Non-blocking I/O လုပ်ဆောင်ချက်ရဖို့အတွက် Inter-process Communication နည်းစနစ်ကို တစ်ချို့က သုံးကြပါ တယ်။ ပရိုဂရမ်တစ်ခုကို Run လိုက်တဲ့ အခါ Operating System က Memory Space အပါအဝင် System Resource တွေကို ခွဲဝေပေးထားတဲ့ Process တစ်ခုဖွင့်ပေးပါတယ်။ ပရိုဂရမ်မှာပါတဲ့ Instruction တွေကို အဲဒီ Process ထဲမှာ အစီအစဉ်အတိုင်း အလုပ်လုပ်ပေးသွားမှာဖြစ်ပါတယ်။ Non-blocking I/O လုပ်ဆောင်ချက်ရဖို့ အတွက် Process နှစ်ခု သုံးခုကို အပြိုင်အသုံးပြုနိုင်ပါတယ်။ ဥပမာ - ဖိုင်ကိုဖွင့်ဖတ်ပြီး ပါဝင်တဲ့ အချက်အလက် တွေကို Memory ပေါ်တင်ပေးတဲ့ I/O လုပ်ငန်းကို မူလ Process မှာမလုပ်ပဲ သီးခြား Process တစ်ခုဖွင့်ပြီး အလုပ်လုပ်စေနိုင်ပါတယ်။ ဒါပေမယ့် ပြဿနာ က Process ဆိုတာ သီးခြားအလုပ်လုပ်ကြတာပါ။ အခြေခံ အားဖြင့် တစ်ခုနဲ့တစ်ခု ဆက်စပ်မှုမရှိပါဘူး။ ဒါကြောင့် သီးခြား Process တစ်ခုနဲ့ အလုပ်လုပ်နေတဲ့ I/O လုပ်ငန်းရဲ့ ဆောင်ရွက်မှုအခြေအနေကို မူလ Process က အလိုအလျောက်မသိနိုင်ပါဘူး။ မူလ Process အနေနဲ့ I/O Process ရဲ့ လုပ်ငန်းပြီးမောက်မှုအခြေအနေကို သိရဖို့လိုပါတယ်။ ဥပမာ - ဖိုင်ထဲက အချက်အလက်တွေကို

Memory ပေါ် တင်ပြီးပြဆိုရင်၊ အချက်အလက်တွေရှိနေတဲ့ Memory Space ရဲ့ Reference ကိုသိရမဲ့ မူလ Process က Memory ထဲမှာရောက်ရှိနေပြီးဖြစ်တဲ့ အချက်အလက်တွေကို ပရိုကရမ်မှာ ဖော်ပြပေးတဲ့ လုပ်ငန်းကို လုပ်နိုင်မှာပါ။ ဒီလိမ့်း Process တစ်ခုနဲ့တစ်ခု အပြန်အလုန် ဆက်သွယ်အလုပ်လုပ်နိုင် အောင် စီမံရေးသား ခြင်းကို Inter-process Communication (IPC) လို့ခေါ်တာပါ။ Non-blocking I/O လုပ်ဆောင်ချက်ရရှိ မြင်နည်းစနစ်ကို တစ်ချို့ကသုံးကြပါတယ်။ Message Passing, Message Queue, Socket, Memory Mapped File စာမျက်နှာများ ဖြစ်ပါတယ်။

တစ်ချို့ ပရိုကရမ်တွေကတော့ Non-blocking I/O လုပ်ဆောင်ချက်ရရှိဖို့အတွက် Time Sharing Thread တွေကို သုံးကြပါတယ်။ Thread ဆိုတာ Process တစ်ခုအတွင်းမှာပဲ Instruction တွေကို ခွဲခြားအလုပ်လုပ်စေတဲ့ နည်းစနစ် တစ်မျိုးဖြစ်ပါတယ်။ Process နှစ်ခုသုံးခု သုံးဖို့မလိုတော့ပဲ Process တစ်ခုတည်းနဲ့ Non-blocking I/O လုပ်ဆောင်ချက် ရရှိအောင် စီမံသွားနိုင်ခြင်း ဖြစ်ပါတယ်။ Thread တွေက Process တစ်ခုအတွင်းမှာ အလုပ်လုပ်ကြခြင်းဖြစ်လို့ အချင်းချင်း အပြန်အလုန်ဆက်သွယ် ဆောင်ရွက်နိုင်ကြပါတယ်။ IPC လိုနည်းစနစ် တွေထပ်ပြီး Implement လုပ်နေစရာမလိုပဲ သက်ဆိုင်ရာ Language နဲ့ Platform ကပေးထားတဲ့ Thread API တွေကို သုံးပြီး ရေးသားစီမံနိုင်ပါတယ်။



Source – Wikipedia

ပုံ (၁၀.J) မှာ Time Sharing Thread စနစ်တစ်ခုရဲ့ အလုပ်လုပ်ပုံကို ဖော်ပြထားပါတယ်။ Process တစ်ခု အတွင်းမှာ Thread နှစ်ခုရှိနေပြီး Instruction တွေကို အပြိုင်အလုပ်လုပ်ကြပါတယ်။ ထူးခြားချက်အနေနဲ့ Thread #2 ကို အလုပ် လုပ်နေစဉ်မှာ Thread #1 ကိုခေါ်တွေ့ရပ်နားထားပြီး Thread #1 အလုပ်လုပ်နေစဉ်မှာ Thread #2 ကို ခေါ်တွေ့ရပ်နားထား တဲ့သော့ ဖော်ပြထားတာကို တွေ့နိုင်ပါတယ်။ Process ခဲ့အချိန်ကို ဝေမျှ သုံးစွဲပြီး လုပ်ငန်းနှစ်ခုသုံးခုကို ပြုင်တူအလုပ် လုပ်တဲ့အတွက် Time Sharing Thread လို့ခေါ်တာပါ။ ဒီနည်းနဲ့ I/O လုပ်ငန်းတွေကို Thread တစ်ခုခဲ့ပြီး ဆောင်ရွက် နေစဉ်မှာ ပရိုကရမ်ရဲ့ အခြား Instruction တွေက သီးခြား Thread အနေနဲ့ ဆက်လက်အလုပ်လုပ်နိုင်မှာပဲ ဖြစ်ပါတယ်။ Thread တွေစီမံရတာလည်း လွယ်ကူတဲ့အလုပ်

တော့မဟုတ်ပါဘူး။ Thread တစ်ခုနဲ့တစ်ခု အပြန်အလှန်စောင့်ရင်း ပရီဂရမ်က ရွှေမဆက်နိုင်ပဲ Hang သွားခြင်း၊ Resource တစ်ခုတည်းကို Thread နှစ်ခုက တစ်ပြိုင်တည်း အသုံးပြုမိလို့ မလိုလားအပ်တဲ့ရလဒ်တွေ ထွက်ပေါ်လာခြင်း စတဲ့ပြဿနာတွေ မဖြစ်အောင် စီမံရေးသားနိုင်ဖို့ လိုပါတယ်။

NodeJS မှာတော့ Non-blocking I/O လုပ်ဆောင်ချက်ရရှိဖို့အတွက် Callback နည်းစနစ်ကို အသုံးပြုပါတယ်။ အမှန် တော့ I/O ကို Process တွေက စောင့်ရတယ်ဆိုတာ Process လုပ်ငန်းပြည့်စုံဖို့အတွက် I/O ကပေးတဲ့ ရလဒ်ကို လိုအပ် လိုဖြစ်ပါတယ်။ IPC လို နည်းစနစ်တွေကို အသုံးပြုရတယ်ဆိုတာလည်း I/O Process တစ်ခု ကပေးတဲ့ ရလဒ်ကို မူလ Process က ရယူအသုံးပြုနိုင်ဖို့ လိုအပ်တဲ့အတွက် ဖြစ်ပါတယ်။ NodeJS မှာတော့ I/O လုပ်ဆောင်ချက်ကပေးတဲ့ ရလဒ် ကိုရရှိဖို့အတွက် Process က စောင့်နေစရာမလိုပါဘူး။ I/O လုပ်ဆောင်ချက် တိုင်းနဲ့အတူ Callback Function တစ်ခု တွဲဖက်သတ်မှတ်ပေးနိုင်ပါတယ်။ I/O လုပ်ဆောင်ချက်ပြီးလို့ ရလဒ်ရရှိလာတဲ့အခါ Callback Function က အလိုအလျောက်အလုပ်လုပ်ပေးသွားမှာဖြစ်ပါတယ်။ ဒီနည်းနဲ့ Process တွေက I/O ကို စောင့်စရာမလိုပဲ၊ လုပ်စရာရှိတဲ့အလုပ်ကို အပြုံးလုပ်လို့ ရသွားပါတယ်။ ဥပမာ တစ်ခုဖော်ပြပါမယ် -

### PHP

```
$data = file("bigfile.csv");
print_r($data);

// other processes
```

ပေးထားတဲ့ PHP Code နဲ့နာမာ file() Function ကိုသုံးပြီး bigfile.csv လိုပေါ်တဲ့ ဖိုင်တစ်ခုကိုဖတ်ယူထား ပါတယ်။ ဖိုင်ကိုဖတ်နေတဲ့အချိန်မှာ တစ်ခြား Process တွေကို အလုပ်လုပ်ပေးနိုင်မှာမဟုတ်ပါဘူး။ ဖိုင်ကိုဖတ်တဲ့လုပ်ငန်းပြီးတော့မှာသာ အခြား Process တွေကို အလုပ်လုပ်နိုင်မှာဖြစ်ပါတယ်။

### JavaScript

```
1. var fs = require("fs");
2.
3. fs.readFile("bigfile.csv", "utf8", function(err, data) {
4.   console.log(data)
5. });
6.
7. // other processes
```

ပေးထားတဲ့ NodeJS Code နဲ့နာမာ fs.readFile() Function ကိုသုံးပြီး bigfile.csv ကိုဖတ်ထားပါတယ်။ ဒါပေမယ့် အဲဒီဖိုင်ကိုဖတ်နေတဲ့အချိန်မှာ အခြား Process တွေက စောင့်စရာမလိုပါဘူး ဆက်လက်အလုပ်လုပ်သွားနိုင်ပါတယ်။ ဖိုင်ဖတ်တဲ့လုပ်ငန်းပြီးတဲ့အခါ fs.readFile() ရဲ့ နောက်ဆုံး Parameter အဖြစ် တွဲဖက်ပေးလိုက်တဲ့ Callback Function က အလုပ်လုပ်သွားပြီး ရရှိလာတဲ့ File Content ကို ရှိက်ထုတ်ဖော်ပြပေးသွားမှာဖြစ်ပါတယ်။

ဒီနည်းနဲ့ Callback Function ကိုသုံးပြီး Non-blocking I/O လုပ်ဆောင်ချက် ရရှိသွားတဲ့အတွက် Thread တွေ Process တွေသုံးစရာမလိုတော့ပါဘူး။ Thread တွေအများကြီးဖွင့်စရာမလိုတဲ့အတွက် ရေးသားစီမံရတာ လွယ်ကူသွားတဲ့အပြင် Resource Consumption နည်းသွားပြီး Process ပေါင်းများစွာ၊ Request ပေါင်းများစွာ ကို လက်ခံစီမံဖို့လိုတဲ့ Network ပရိုဂရမ်တွေ ရေးသားဖန်တီးရာမှာ များစွာအထောက်အကျဖြစ်စေမှာပဲဖြစ်ပါတယ်။

## 10.2 – Event Driven Architecture

Web App တွေအပါအဝင် Graphical User Interface (GUI) ကို အခြေခံအလုပ်လုပ်တဲ့ ပရိုဂရမ်တစ်ခု ဖန်တီးတိုးမှာ Event Driven Programming ကို ကျွန်ုတ်တို့ သုံးကြလေ့ရှိပါတယ်။ ပရိုဂရမ်ကို ရေးသားတဲ့အခါ ကြိုတင်သတ်မှတ်ထား တဲ့ အစီအစဉ်အတိုင်း မဟုတ်ပဲ၊ Button ကို Click လုပ်လိုက်ရင် ဘယ်အလုပ်လုပ်ရမယ်၊ Image ကို Load လုပ်လိုက်ရင် ဘယ်အလုပ်လုပ်ရမယ်၊ Menu ကို Mouse နဲ့ ထောက်လိုက်ရင် ဘယ်အလုပ်လုပ်ရမယ်၊ စသဖြင့် UI ပေါ်မှာဖြစ်ပေါ်တဲ့ Action ပေါ်မှုတည်ပြီး အလုပ်လုပ်အောင် စီစဉ်ရေးသားရတဲ့ Programming နည်းစနစ်ဖြစ်ပါတယ်။

Event Driven Architecture ဆိုတာကတော့ Event Driven Programming လိုပဲ ပရိုဂရမ်ကို ကြိုတင်သတ်မှတ်ထားတဲ့ အစီအစဉ်အတိုင်း အလုပ်လုပ်စေခြင်းမဟုတ်ပဲ ဖြစ်ပေါ်တဲ့ Event များပေါ်မှာ မူတည်အလုပ်လုပ်အောင် စီစဉ်ရေးသားခြင်း ပဲဖြစ်ပါတယ်။ User Interface အတွက်သာမကတော့ပဲ စနစ်တစ်ခုလုံးကို Event များပေါ်မှာ အခြေခံအလုပ်လုပ်အောင် စီစဉ်ရေးသားတဲ့ နည်းစနစ်ဖြစ်ပါတယ်။

ပြီးခဲ့တဲ့အခန်းမှာ BackboneJS အကြောင်း လေ့လာခဲ့ကြစဉ်က add, remove, reset, sort, error စသဖြင့် Event တွေဖြစ်ပေါ်တဲ့အခါ ဆောင်ရွက်ရမယ့်လုပ်ငန်းတွေ သတ်မှတ်ထားနိုင်ပုံကို တွေဖြင့်ခဲ့ကြရပြီး ဖြစ်ပါတယ်။ NodeJS နဲ့လည်း အဲဒီလို Custom Event တွေကိုတည်ဆောက်ပြီး Event များကို အခြေခံအလုပ်လုပ်တဲ့ ပရိုဂရမ်အဖြစ် တည်ဆောက် နိုင်ခြင်းဖြစ်ပါတယ်။ ဥပမာ -

### Pseudo-code

```
issue.on('assigned', function() {
    // send notification to assignee
});

issue.on('done', function() {
    // send notification to submitter
});
issue.on('close', function() {
    // archive issue
});
```

UI Event တွေမှာ onClick, onKeyPress, onMouseMove စသဖြင့်ရှိသလိုပဲ ကျွန်ုတ်တို့ပရိုဂရမ်တွေ မှာလည်း နှမူနာမှာပေးထားသလို assigned, done, close စသဖြင့် Event တစ်ခုဖြစ်ပေါ်တဲ့အခါ ဆောင်ရွက်ရမယ့် လုပ်ငန်းတွေကို သတ်မှတ်ထားနိုင်အောင် ရေးသားနိုင်မှာပဲဖြစ်ပါတယ်။

### 10.3 – Running JavaScript with NodeJS

အခန်း (၈) မှာ Yeoman နဲ့ အခြား Tool တွေအသုံးပြုဖို့ NodeJS လိုအပ်တဲ့အတွက် NodeJS Install ပြုလုပ်ပုဂ္ဂိုလ်သွေးပြုလုပ်ရသေးရင်လည်း [nodejs.org](http://nodejs.org) မှာ Installer ကို Download ရယူ နိုင်ပါတယ်။ Ubuntu မှာ ဆိုရင်တော့ ကိုယ်တိုင်သွားပြီး Download လုပ်မနေပဲ အခုလို Install ပြုလုပ်နိုင်ပါတယ်။

```
$ sudo apt-get install nodejs nodejs-legacy
```

Install လုပ်ပြီးပြီဆိုရင် node Command ကနေတစ်ဆင့် NodeJS Console ကိုရရှိနိုင်ပါတယ်။

ဆက်လက်ဖော်ပြုမယ့် Command တွေနဲ့ Code တွေဟာ လက်တွေ့ကူးယူစမ်းသပ်နိုင်တဲ့ Command နဲ့ Code တွေဖြစ်ပါတယ်။

NodeJS Console ထဲမှာ JavaScript Code တွေကို အခုလို တိုက်ရှိက်ရေးသားစမ်းသပ်နိုင်ပါတယ်။

```
$ node
> console.log("Hello, World!");
Hello, World!
undefined

> function area(r) {
... var pi = 3.14;
... return pi * r * r;
... }

undefined

> area(8)
200.96
```

NodeJS Console ကိုမဝင်ပဲ JavaScript Code တွေကို တိုက်ရှိက် Run လိုရင်တော့ -e သို့မဟုတ် -p Option ကို သုံးနိုင်ပါတယ်။

```
$ node -e 'console.log("Hello, World")'
Hello, World
```

-e Option ကို သုံးဖို့တဲ့အမိမိဖြစ်ပြီး -p ကတော့ Evaluate and Print ဆိုတဲ့အမိမိဖြစ်ပါ။ ဖိုင်တစ်ခု

အနေနဲ့ ရေးသားထားတဲ့ JavaScript Code တွေကိုတော့ အခုလို Run နိုင်ပါတယ်။

```
$ node script.js [args]
```

[args] နေရာမှာ script.js ကို Run စဉ် တဲ့ဖက်ပေးလိုတဲ့ Arguments တွေကို ပေးနိုင်ပါတယ်။ Arguments တွေကို NodeJS က process.args ဆိုတဲ့ Object နဲ့လက်ခံထားပေးမှာဖြစ်ပါတယ်။ ဥပမာ – script.js ထဲမှာ အခုလို ရေးသားစစ်းသပ်နိုင်ပါတယ်။

### JavaScript

```
console.log(process.args)
```

process.args မှာရှိတဲ့တန်ဖိုးတွေကို ရိုက်ထုတ်ထားခြင်း ဖြစ်ပါတယ်။ ပြီးတဲ့အခါ script.js ကို Arguments တွေပေးပြီး Run ကြည့်ပါမယ်။

```
$ node script.js yo bower grunt
[ 'node', '/home/eimg/script.js', 'yo', 'bower', 'grunt' ]
```

yo, bower နဲ့ grunt တို့ကို Argument အနေနဲ့ပေးပြီး Run ကြည့်လိုက်တဲ့အခါ process.args ဟာ ကျွန်ုတ်တို့ပေးလိုက်တဲ့ yo, bower နဲ့ grunt တို့ ပါဝင်တဲ့ Array တစ်ခုဖြစ်တယ်ဆိုတာကို တွေ့ရနိုင်ပါတယ်။ ရှေ့ဆုံးက node နဲ့ ဖိုင် Path ဆိုတဲ့ Index နှစ်ခုက process.args မှာ အမြဲ ရှိနေမှာဖြစ်ပါတယ်။ ဒါ ကြောင့် အဲဒီ Index နှစ်ခုကို ရလဒ်မှာမပါဝင်စေလိုရင် အခုလိုပြင်ဆင်ရေးသားနိုင်ပါတယ်။

### JavaScript

```
console.log(process.args.slice(2))
```

ပြင်ဆင်ရေးသားထားတဲ့ script.js ကိုနောက်တစ်ခေါက်စမ်းကြည့်ရင် ရလဒ်က အခုလိုဖြစ်မှာပါ။

```
$ node script.js yo bower grunt
[ 'yo', 'bower', 'grunt' ]
```

script.js ကို Run စဉ်ကပေးလိုက်တဲ့ တန်ဖိုးတွေကို ရရှိတာကိုတွေ့ရမှာဖြစ်ပါတယ်။ ဒီနည်းနဲ့ User Input တွေကို Command Argument အနေနဲ့ လက်ခံအလုပ်လုပ်နိုင်တဲ့ Command Line ပရိုဂရမ်တွေကို NodeJS နဲ့ ရေးသား ဖန်တီးနိုင်မှာဖြစ်ပါတယ်။ Yeoman, Bower, Grunt စတဲ့ Tool တွေတာ ဒီနည်းနဲ့ရေးသားတဲ့ Command Line ပရိုဂရမ်တွေဖြစ်ပါတယ်။

## 10.4 – Debugging in NodeJS

JavaScript ပရိုဂရမ်တစ်ခုကို Debug လုပ်လိုရင် node Command ကို debug Argument နဲ့ တွဲသုံးနိုင်ပါတယ်။

```
$ node debug script.js
```

NodeJS က Debug Console တစ်ခုနဲ့ ပရိုဂရမ်ရဲ့ လိုင်းနံပါတ် (၁) ကနေစပီး တစ်လိုင်းချင်း Debug လုပ်နိုင်အောင် စီစဉ်ပေးထားပါလိမ့်မယ်။ Code ထဲမှာ Break Point တွေ သတ်မှတ်လိုရင်တော့ debugger Statement ကို အသုံးပြု နိုင်ပါတယ်။ ဥပမာ –

### JavaScript

```
1. var pi = 3.14;
2. function area(r) {
3.     var a = pi * r * r;
4.     debugger;
5.     return a;
6. }
7.
8. console.log( area(8) );
9. console.log( area(12) );
```

လိုင်းနံပါတ် (၁) မှာ debugger Statement ကိုသုံးထားတဲ့အတွက် Node Debug က ဒီလိုင်းကိုရောက်တိုင်း Break လုပ်ပေးမှာ ဖြစ်ပါတယ်။ ဒီ Code ကို node debug နဲ့ Run ကြည့်ရင် အခုလုံ တွေ့ရပါလိမ့်မယ်။

```
$ node debug script.js
```

```
< debugger listening on port 5858
connecting... ok
break in script.js:1
1 var pi = 3.14;
2 function area(r) {
3 var a = pi * r * r;

debug>
```

ပရိုဂရမ်ဟာ လိုင်းနံပါတ် (၁) မှာ ရပ်နေပြီး Debug Console ကိုအသုံးပြုတော်ထားပေးပါလိမ့်မယ်။ ပရိုဂရမ်ဆက်အလုပ် လုပ်သွား စေဖို့အတွက် cont (သို့မဟုတ်) c Command ကို ရိုက်ထည့်ပေးရပါတယ်။

```
debug> c
```

```
break in script.js:4
2 function area(r) {
3 var a = pi * r * r;
```

```

4   debugger;
5   return a;
6 }

debug>

```

ပရိုဂရမ်က ဆက်အလုပ်လုပ်သွားပေမယ့် debugger ကိုတွေ့တဲ့အတွက် လိုင်းနံပါတ် (၄) မှာ နောက်တစ်ခေါင် ရပ် သွားပါလိမ့်မယ်။ အဲဒီအချိန်မှာ repl Command ကိုသုံးပြီး JavaScript Code တွေကို Run ကြည့်လို ရပါတယ်။ Debug Console ကနေ repl Console ကိုဖြောင်းသွားပါလိမ့်မယ်။ Variable တွေရဲ့ လက်ရှိတန်ဖိုး ကို စီစစ်ဖို့အသုံးဝင် ပါတယ်။

```

debug> repl

Press Ctrl + C to leave debug repl

> a

200.96

debug>

```

နမူနာမှာ Variable a ထဲက တန်ဖိုးကိုသိချင်တဲ့အတွက် a လို ရိုက်ထည့် ပေးလိုက်တဲ့အခါ လက်ရှိတန်ဖိုး 200.96 ကို ဖော်ပြုလာခြင်းပဲဖြစ်ပါတယ်။ repl Console ကနေထွက်ပြီး Debug Console ကို ပြန်သွားဖို့ အတွက် Ctrl+C ကိုနှိပ် ပေးရပါတယ်။ Debug Console ပြန်ပေါ်လာတဲ့အခါ လက်ရှိ Break လုပ်ထားတဲ့ Function ကနေထွက်ပြီး နောက်တစ် ဆင့်ကို ဆက်လက် Debug လုပ်သွားဖို့အတွက် out (သို့မဟုတ်) o Command ကို သုံးနိုင်ပါတယ်။

```

debug> o

break in script.js:8
  6 }
  7
  8 console.log( area(8) );
  9 console.log( area(12) );
10

debug>

```

Function အပြင်ဘက်က လိုင်းနံပါတ် (၈) ကိုရောက်သွားပါလိမ့်မယ်။ နောက်တစ်လိုင်းကို ဆက်သွားချင်တယ် ဆိုရင် next (သို့မဟုတ်) n Command ကို သုံးနိုင်ပါတယ်။

```
debug> n

< 200.96
break in script.js:9
  7
  8 console.log( area(8) );
  9 console.log( area(12) );
10
11 });

debug>
```

လိုင်းနံပါတ် (၉) ကိုရောက်သွားပါလိမ့်မယ်။ n Command နဲ့တစ်လိုင်းပြီးတစ်လိုင်းဆက်သွားပြီး Debug လုပ်သွားနိုင်ပါတယ်။ တစ်လိုင်းချင်းသွားယုံသာမက၊ ခေါ်ယူအသုံးပြုထားတဲ့ Function ထဲကိုပါ ဝင်လေ့လာတယ် ဆိုရင်တော့ step (သို့မဟုတ်) s Command ကို သုံးနိုင်ပါတယ်။

```
debug> s

break in node.js:205
203  startup.globalConsole = function() {
204    global.__defineGetter__('console', function() {
205      return NativeModule.require('console');
206    });
207  };

debug>
```

လိုင်းနံပါတ် (၉) မှာ console.log() Function ကိုခေါ်ထားတဲ့အတွက် s Command ကိုသုံးလိုက်တဲ့အခါ NodeJS ရဲ့ console.log() Function ရေးသားထားရာနေရာထိ ရောက်ရှိသွားတာကို တွေ့ရမှာဖြစ်ပါတယ်။ အဲဒီနေရာကနေထွက်ပြီး မူလအလုပ်လုပ်နေတဲ့ နေရာကိုပြန်သွားဖို့အတွက် out (သို့မဟုတ်) o Command ကို သုံးနိုင်ပါ တယ်။

```
debug> o

break in script.js:9
  7
  8 console.log( area(8) );
  9 console.log( area(12) );
10
11 });

debug>
```

လက်ရှိပရိုကရမဲ့ လိုင်းနံပါတ် (၉) ကိုပြန်လည်ရောက်ရှိ လာတာကိုတွေ့ရမှာပဲဖြစ်ပါတယ်။ ပရိုကရမဲ့ကို အဆုံးထိ ဆက်လက် အလုပ်လုပ်သွားစေဖို့အတွက် cont (သို့မဟုတ်) c ကို သုံးနိုင်ပါတယ်။

```
debug> c

break in script.js:4
  2 function area(r) {
  3     var a = pi * r * r;
  4     debugger;
  5     return a;
  6 }

debug>
```

ပရိုဂရမ်ဆက်အလုပ်လုပ်တဲ့အခါ area() Function ကို နောက်တစ်ကြိမ်ထပ်ခေါ်ထားတဲ့အတွက် area() Function ထဲကိုရောက်သွားပြန်ပါတယ်။ area() Function ထဲမှာ debugger Statement ရှိနေတဲ့အတွက် debugger Statement ရှိရာလိုင်းမှာ နောက်တစ်ကြိမ် ရပ်နေပေးပါလိမ့်မယ်။ repl Console ကိုဝင်ပြီး Variable a ရဲတန်ဖိုးကို လေ့လာ ကြည့်နိုင်ပါတယ်။

```
debug> repl

Press Ctrl + C to leave debug repl

> a

452.1599999999999997

debug> c

< 452.15999999999997
program terminated

debug>
```

a ရဲတန်ဖိုး စောစောကနဲ့ မတူတော့ပဲ 452.159 ဖြစ်နေတာကို တွေ့ရမှာဖြစ်ပါတယ်။ Debug Mode ကိုပြန်သွားပြီး c Command နဲ့ ပရိုဂရမ်ကို ဆက်လက်အလုပ်လုပ်စေလိုက်တဲ့အခါမှာတော့ ပရိုဂရမ်ရဲလုပ်ငန်းအားလုံးပြီးမြောက်သွားတဲ့ အတွက် program terminated ဆိုတဲ့ Message ကိုပေးပြီး ရပ်သွားမှာပဲဖြစ်ပါတယ်။

နမူနာမှာ အသုံးပြုခဲ့တဲ့ Debug Console Command စာရင်းကို ထပ်မံဖော်ပြလိုက်ပါတယ်။

- **cont, c** - ပရိုဂရမ်ကို ဆက်အလုပ်လုပ်သွားစေလိုတဲ့အခါ သုံးပါတယ်။
- **next, n** - နောက်တစ်လိုင်းကို ဆက်လက်လေ့လာလိုတဲ့အခါ သုံးပါတယ်။
- **step, s** - Step In ဆိုတဲ့အဓိပ္ပာယ်ပါ။ အခြား Platform တွေမှာရှိတဲ့ Debug စနစ်တွေရဲ့ Step In နဲ့ သဘောတရားအတူတူပါပဲ။ လက်ရှိရောက်ရှိနေတဲ့လိုင်းက ခေါ်ယူအသုံးပြုထားတဲ့ Function ကို ဝင်ရောက်လေ့လာလိုတဲ့အခါ သုံးပါတယ်။
- **out, o** - Step Out ဆိုတဲ့အဓိပ္ပာယ်ပါ။ အခြား Platform တွေမှာရှိတဲ့ Debug စနစ်တွေရဲ့ Step Out နဲ့

သဘောတရားအတူတူပါပဲ။ လက်ရှိရောက်ရှိနေတဲ့ Function ထဲကနေထွက်ပြီး နောက်တစ်လိုင်းကို ဆက်လက် Debug လုပ် သွားစေလိုတဲ့ အခါ သုံးပါတယ်။

- **pause** - နမူနာမှာသုံးမထားပါဘူး။ အလုပ်လုပ်နေတဲ့ပရိုကရမဲ့ ကို ရောက်ရာနေရာမှာ ရပ်တန်လိုက် စေလိုတဲ့ အခါ သုံးပါတယ်။

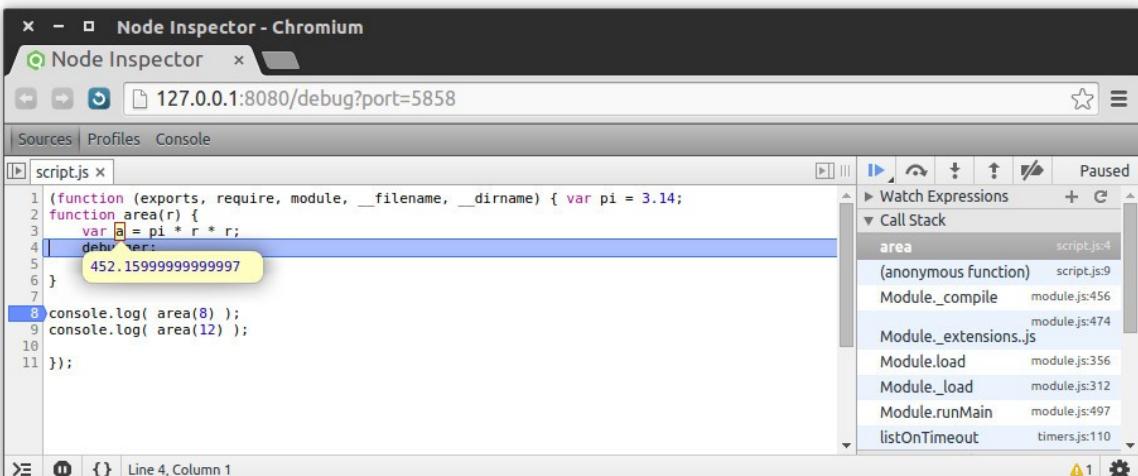
Debug Console ထဲမှာ help Command ကိုသုံးပြီး အသုံးပြုလိုရတဲ့ Command စာရင်းတိုကြည့်ရှုနိုင်ပါတယ်။ ဒီလို Command Line ကနေ Debug လုပ်ရတာ ရှုပ်တယ်ထင်ရင်တော့ Node Inspector ဆိုတဲ့ Package တစ်ခု ကို အသုံးပြု နိုင်ပါတယ်။ NPM နဲ့ အခုလို Install လုပ် နိုင်ပါတယ်။

```
$ npm install -g node-inspector
```

Node Inspector ဆိုတာ အမှန်တော့ Chromium Browser ရဲ့ Dev Tool ပဲဖြစ်ပါတယ်။ Firebug, Chrome Dev Tool စဲတဲ့ Debug Tool တွေနဲ့ ရင်းနှီးပြီးသားသူတွေအတွက် အထူးတလည်ထပ်လေ့လာနေစရာမလိုပဲ အသုံးပြုနိုင်မှာ ဖြစ်ပါတယ်။ Node Inspector နဲ့ JavaScript Code တွေကို Debug လုပ်နိုင်ဖို့ node-debug Command ကို သုံးရပါတယ်။ ဥပမာ -

```
$ node-debug script.js
debugger listening on port 5858
Node Inspector is now available from http://127.0.0.1:8080/debug?port=5858
Debugging `script.js`
```

Node Inspector က script.js ကို Chrome Dev Tool နဲ့ဖော်ပြပေးမှာဖြစ်ပါတယ်။ အကယ်၍ အလိုအလျောက် မဖော်ပြရင်လည်း node-debug Command Message ထဲမှာပါတဲ့ [127.0.0.1:8080/debug?port=5858](http://127.0.0.1:8080/debug?port=5858) ဆိုတဲ့ URL ကို Google Chrome (သို့မဟုတ်) Chromium Browser နဲ့ဖွင့်ပြီး Node Inspector UI ကို ရရှိနိုင်ပါတယ်။



ပုံ (၁၀.၃) မှာလေ့လာကြည့်ပါ။ script.js ကို Node Inspector နဲ့ Debug လုပ်နေခြင်းဖြစ်ပါတယ်။ Continue, Step Into, Step Over, Step Out, Pause စဲတဲ့ Debug Command များအတွက် Button လေးတွေကို ညာဘက် အပေါ်ထောင့်မှာ ပေးထားပါတယ်။ debugger Statement တွေမှာ အလိုအလျောက် Break လုပ်ပေး သလို နောက် ထပ် Break Point တွေ ထပ်ထည့်ချင်ရင်လည်း Line Number ကိုနှိမ်ပြီး Break Point သတ်မှတ်ခြင်း၊ ပထ်ဖျက်ခြင်းကို ဆောင်ရွက်နိုင်ပါတယ်။ Variable တွေကို Mouse နဲ့ ထောက်ကြည့်လိုက်ရင် လက်ရှိ တန်ဖိုးကိုဖော်ပြပေးသလို၊ ညာ ဘက်ခြစ်းက Side-bar ထဲမှာ လက်ရှိ Variable တွေ Function တွေနဲ့ပေါ်သက်တဲ့ အချက်အလက်တွေကို ဖော်ပြပေး ထားတာကို တွေ့ရမှာဖြစ်ပါတယ်။ ပါဝင်တဲ့ Function တွေက NodeJS ရဲ့ Build-in Debug လုပ်ဆောင်ချက်နဲ့ သိပ်မကွာပေးမယ့် Command Line မဟုတ်ပဲ UI နဲ့ ဖြစ်တဲ့အတွက် အသုံးပြုရ ပိုမိုလွယ်ကူပါတယ်။

## 10.5 – NodeJS Module

NodeJS ဆိုတာ JavaScript ပဲမို့ JavaScript ရဲ့ Language Feature တွေကိုတော့ ထပ်ပြောမနေတော့ပါဘူး။ အခန်း (၃) မှာ JavaScript OOP ရဲ့ ထူးခြားချက်တွေကိုဖော်ပြခဲ့ပြီး ဖြစ်ပါတယ်။ ပိုပြီးအခြေခံကျတဲ့ Language Feature တွေကို Professional Web Developer ရဲ့ အခန်း (၄) မှာဖော်ပြထားပါတယ်။ ဒီနေရာမှာ NodeJS က ဖြည့်စွက်ပေးထားတဲ့ ထူးခြားချက်တွေကို အမိကထားဖော်ပြသွားမှာဖြစ်ပါတယ်။ NodeJS က ဖြည့်စွက်ပေးထားတဲ့ ထူးခြားချက်တွေထဲမှာ အမိကကျတဲ့တစ်ချက်ကတော့ Module ဖြစ်ပါတယ်။

အခြေခံအားဖြင့် JavaScript Code ဖိုင်တစ်ခုကနေ အခြား JavaScript Code ဖိုင်တစ်ခုကို ချိတ်ဆက်အသုံးပြုလိုမရပါဘူး။ ဒါကြောင့် Client-side JavaScript မှာ ဆိုရင် JavaScript Module တွေကို စီမံဖို့အတွက် HTML <script> Element ကိုပဲ အားကိုးရပါတယ်။ ဥပမာ – JavaScript ဖိုင် (၃) ခုချိတ်ဆက်အသုံးပြုချင်တဲ့အခါ JavaScript ထဲမှာ ကြော်ချိတ်ဆက်လိုမရပဲ HTML Document တစ်ခုအတွင်းမှာ အခုလိုချိတ်ဆက်ပေးရပါတယ်။

### HTML

```
<script src="circle.js"></script>
<script src="square.js"></script>
<script src="area.js"></script>
```

ဒီလိုအပ်ချက်ကို ဖြည့်စွက်ဖို့အတွက် CommonJS, RequireJS စဲတဲ့ နည်းပညာများကို တိုထွင်ဖန်တီးပြီး အသုံးပြုခဲ့ကြ ရပါတယ်။ NodeJS မှာတော့ JavaScript Code ထဲမှာ အခြား JavaScript Module ဖိုင်တွေကို ချိတ်ဆက်အသုံးပြုနိုင် အောင် စီစဉ်ပေးထားပါတယ်။ require() Function သုံးပြီး ချိတ်ဆက်နိုင်ပါတယ်။

### JavaScript

```
var http = require("http");
var express = require("express");
var measure = require("./circle.js");
```

`require()` Function နဲ့ NodeJS ရဲ့ Build-in Module တွေကို Module အမည်သုံးပြီး တိုက်ရိုက်ချိတ်ဆက်နိုင်ပါ တယ်။ NPM နဲ့ Install လုပ်ထားတဲ့ Package တွေကိုလည်း ချိတ်ဆက်နိုင်ပါတယ်။ ကိုယ်တိုင်ရေးသားထားတဲ့ JavaScript ဖိုင်တွေကိုလည်း ဖိုင်တည်နေရာ Path ပေးပြီး ချိတ်ဆက်နိုင်ပါတယ်။ NodeJS Module တွေကို အခုလို ရေးသားရပါတယ်။

### JavaScript

```
1. exports.circle = function (r) {
2.     var pi = 3.14;
3.     return pi * r * r;
4. }
5.
6. exports.square = function (w, h) {
7.     return w * h;
8. }
```

(သို့မဟုတ်)

### JavaScript

```
1. module.exports = {
2.     circle: function (r) {
3.         var pi = 3.14;
4.         return pi * r * r;
5.     },
6.
7.     square: function (w, h) {
8.         return w * h;
9.     }
10. }
```

`exports Object` မှာ `Function` တွေ `Property` တွေကို တစ်ခုပြီးတစ်ခု တဲ့ဖက်ပေးနိုင်သလို၊ `module.exports Object` မှာ `Function` တွေ `Property` တွေကို အတဲ့လိုက်လည်း သတ်မှတ်ပေးနိုင်ခြင်းဖြစ်ပါတယ်။ `require()` Function ကို အခုလိုရေးသားထားတယ်လို့ မှတ်ယူနိုင်ပါတယ်။

### Pseudo-code

```
var require = function(path) {
    // load module or package or path
    return module.exports;
}
```

`require()` Function က ချိတ်ဆက်အသုံးပြုလိုတဲ့ Module (သို့မဟုတ်) Package (သို့မဟုတ်) ဖိုင်ကိုရယူပြီး အဲဒီ ဖိုင်ထဲက `module.exports Object` ကို `return` ပြန်ပေးခြင်းဖြစ်ပါတယ်။ ဒါကြောင့် `require()` Function က ပြန်ပေးတဲ့တန်ဖိုးကို Variable တစ်ခုနဲ့ လက်ခံပေးဖို့လိုအပ်ပါတယ်။ ဥပမာ -

## JavaScript

```
var area = require("./area.js");
console.log(area.circle(8));           // => 200.96
console.log(area.square(2, 4));        // => 8
```

နဲ့မူနာမှာ area.js Module ကို area Variable နဲ့ လက်ခံချိတ်ဆက်ထားပါတယ်။ ဒါကြောင့် area.js Module ရဲ့ လုပ်ဆောင်ချက်တွေကို အသုံးပြုလိုတဲ့ အခါ area ကနေတစ်ဆင့် အသုံးပြုနိုင်ခြင်းဖြစ်ပါတယ်။

## 10.6 – NodeJS Build-in Modules

NodeJS မှာ Command Line ပရိုဂရမ်တွေ၊ Back-end Service တွေနဲ့ Network ပရိုဂရမ်တွေ ဖန်တီးဖို့အတွက် လိုအပ်တဲ့ အခြေခံ Module တွေ အသင့်ပါဝင်ပါတယ်။ ဒီနေရာမှာ Module အားလုံးကိုတော့ မဖော်ပြနိုင်ပါဘူး။ အမိက (J) ပိုင်းကိုရွေးထုတ်ဖော်ပြလိုပါတယ်။ ဖိုင်တွေစီမံနိုင်တဲ့ File System Module နဲ့ Web Server တစ်ခု အနေနဲ့ အလုပ် လုပ်ပေးနိုင်တဲ့ HTTP Module တို့ပဲဖြစ်ပါတယ်။

### File System Module

File System Module ကို အသုံးပြုနိုင်ဖို့အတွက် require() Function နဲ့ အခုလိုချိတ်ဆက်ပေးဖို့လိုပါတယ်။

## JavaScript

```
var fs = require("fs");
```

File System Object ဖြစ်တဲ့ fs ကိုရရှိပြီးနောက် အဲဒီ Object ကနေတစ်ဆင့် File System Module ရဲ့ လုပ်ဆောင် ချက်တွေကို စတင်အသုံးပြုနိုင်ပြီဖြစ်ပါတယ်။ အသုံးဝင်တဲ့ Function စာရင်းကို ဖော်ပြပေးလိုက်ပါတယ်။

**fs.exists()** – ဖိုင်ရှုမရှိစိစစ်လိုတဲ့ အခါ သုံးရပါတယ်။ ရလဒ်ကို Callback အနေနဲ့ ပြန်ပေးပါတယ်။

## JavaScript

```
fs.exists("path/to/file", function(result) {
  if(result) console.log("File exists!");
});
```

**fs.rename()** – ဖိုင်အမည်ပြောင်းလိုတဲ့ အခါသုံးပါတယ်။ Parameter အနေနဲ့ လက်ရှိအမည်၊ ပြောင်းလိုတဲ့ အမည်နဲ့ Callback တို့ကို ပေးရပါတယ်။ ဥပမာ –

## JavaScript

```
fs.rename("old.csv", "new.csv", function() {
  console.log("Done renaming file.");
});
```

**fs.unlink()** – ဖိုင်ကိုဖျက်လိုတဲ့အခါသုံးပါတယ်။ ဖျက်လိုတဲ့ဖိုင်နဲ့ Callback Function တိုကို Parameter အနေနဲ့ ပေးရပါတယ်။

**fs.readdir()** – Directory တစ်ခုမှာပါဝင်တဲ့ ဖိုင်နဲ့ Sub-directory စာရင်းကိုရယူလိုတဲ့အခါ သုံးပါတယ်။ ဖိုင် စာရင်း Array ကို Callback Parameter အနေနဲ့ ပြန်ပေးပါတယ်။ ဥပမာ –

#### JavaScript

```
fs.readdir("path/to/dir", function(err, files) {
  console.log(files);
});
```

**fs.mkdir()** – Directory တစ်ခုတည်ဆောက်လိုတဲ့အခါသုံးပါတယ်။ တည်ဆောက်လိုတဲ့ Directory အမည်နဲ့ Callback Function တိုကို Parameter အနေနဲ့ ပေးရပါတယ်။

**fs.rmdir()** – Directory တစ်ခုကို ပယ်ဖျက်လိုတဲ့အခါသုံးပါတယ်။ ပယ်ဖျက်လိုတဲ့ Directory အမည်နဲ့ Callback Function တိုကို Parameter အနေနဲ့ ပေးရပါတယ်။

**fs.readFile()** – ဖိုင်တစ်ခုမှာပါဝင်တဲ့ Content ကို ဖတ်ယူလိုတဲ့အခါသုံးပါတယ်။ Content ကို Callback Parameter အနေနဲ့ ပြန်ပေးပါတယ်။ ဥပမာ –

#### JavaScript

```
fs.readFile("users.csv", "utf8", function(err, data) {
  console.log(data);
});
```

ဒုတိယ Parameter အနေနဲ့ Encoding အမျိုးအစားကိုပေးထားတာကို သတိပြုပါ။ နူးနာမှာ utf8 လိုပေးထားပါတယ်။ Read လုပ်လိုတဲ့ဖိုင်မှာသုံးထားတဲ့ Encoding က အခြား Encoding ဆိုရင်လည်း မှန်အောင် သတ်မှတ်ပေးဖို့လိုပါတယ်။

**fs.writeFile()** – ဖိုင်တစ်ခုတည်ဆောက်လိုတဲ့အခါ (သို့မဟုတ်) ဖိုင်တစ်ခုအတွင်းမှာ Content ထွေရေးသားစေ လိုတဲ့အခါသုံးပါတယ်။ ဖိုင်အမည်၊ ရေးသားရမယ့် Content နဲ့ Callback Function တိုကိုပေးရပါမယ်။ ဥပမာ –

#### JavaScript

```
fs.writeFile("fruities.csv", "This is file content.", function() {
  console.log("Done saving data to file.");
});
```

**fs.appendFile()** - လက်ရှိဖိုင်အတွင်းမှာ Content တွေ ဖြည့်စွက်လိုတဲ့အခါ သုံးပါတယ်။ writeFile() မှာလိုပဲ ဖိုင်အမည်၊ ရေးသားရမယ့် Content နဲ့ Callback Function တို့ကိုပေးရပါတယ်။ writeFile() က မူလ Content ရှိနေရင် အစားထိုးသွားမှာဖြစ်ပြီး appendFile() ကတော့ အစားမထိုးပဲ ဖြည့်စွက်ပေးသွားမှာဖြစ်ပါတယ်။

အကြောင်းအမျိုးမျိုးကြောင့် File System Function တွေကို ရိုးရိုး Synchronous ပုံစံအလုပ်လုပ်စေလိုရင် fs.renameSync(), fs.chmodSync(), fs.unlinkSync() စတဲ့ နောက်ကနေ Sync တွဲဖက် အမည်ပေးထားတဲ့ Function တွေကို အစားထိုးသုံးနိုင်ပါတယ်။ ဥပမာ -

### JavaScript

```
var data = fs.readFileSync("users.csv");
console.log(data);
```

File System စီမံရာမှာ အသုံးဝင်တဲ့ နောက်ထပ် Module တစ်ခုကတော့ Path Module ဖြစ်ပါတယ်။ အသုံးပြု နိုင်ဖို့ အတွက် require() နဲ့ အရင်ချိတ်ဆက်ပေးဖို့လိုပါတယ်။

### JavaScript

```
var path = require("path");
```

Path Module ထဲက အဓိကကျတဲ့ Function တွေကိုဖော်ပြလိုက်ပါတယ်။

**path.dirname()** - ပေးလိုက်တဲ့ Path ထဲက Directory အမည်ကိုရယူပေးပါတယ်။ ဥပမာ -

### JavaScript

```
path.dirname("/path/to/file.csv"); // => /path/to
```

**path.basename()** - ပေးလိုက်တဲ့ Path ထဲက ဖိုင်အမည်ကိုရယူပေးပါတယ်။ ဥပမာ -

### JavaScript

```
path.basename("/path/to/file.csv"); // => file.csv
path.basename("/path/to/file.csv", ".csv"); // => file
```

**path.extname()** - ပေးလိုက်တဲ့ Path ထဲက ဖိုင် Extension ကို ရယူပေးပါတယ်။ ဥပမာ -

## JavaScript

```
path.basename("/path/to/file.csv");           // => .csv
path.basename("/path/to/file.csv.gz");         // => .gz
```

**path.parse()** - ပေးလိုက်တဲ့ Path ကို Object အဖြစ်ပြောင်းပေးပါတယ်။ ဥပမာ -

## JavaScript

```
path.parse("/path/to/file.csv");

{
  root: "/",
  dir: "/path/to",
  base: "file.csv",
  ext: ".csv",
  name: "file"
}
```

**path.format()** - ပေးလိုက်တဲ့ Object ကို Path ပြန်ပြောင်းပေးပါတယ်။ ဥပမာ -

## JavaScript

```
path.format({
  root: "/",
  dir: "/path/to",
  base: "file.csv",
  ext: ".csv",
  name: "file"
});

// => /path/to/file.csv
```

အသုံးများနှင့်တဲ့ Function တွေကို ရွေးထုတ်ဖော်ပြခဲ့ခြင်းဖြစ်ပြီး အောက်ပါလိပ်စာများမှာ File System Module နဲ့ Path Module တို့မှာပါဝင်တဲ့ လုပ်ဆောင်ချက် အပြည့်အစုံကို လေ့လာနိုင်ပါတယ်။

<https://nodejs.org/api/fs.html>

<https://nodejs.org/api/path.html>

## HTTP Module

NodeJS ရဲ့ HTTP Module မှာ HTTP Server နဲ့ပက်သက်တဲ့ လုပ်ဆောင်ချက်တွေရှာ HTTP Client နဲ့ပက်သက်တဲ့လုပ်ဆောင်ချက်တွေပါ ပါဝင်ပါတယ်။ တစ်နည်းအားဖြင့် NodeJS နဲ့ HTTP Web Server တစ်ခုတည်ဆောက်နိုင်သလို HTTP Web Client တစ်ခုလည်း တည်ဆောက်နိုင်ပါတယ်။ HTTP Server တစ်ခုကို အခုလိုတည်ဆောက်နိုင်ပါတယ်။

## JavaScript

```

1. var http = require("http");
2.
3. http.createServer(function(req, res) {
4.     res.writeHead(200, { "Content-Type": "text/html" });
5.     res.write("Hello, World!");
6.     res.end();
7. }).listen(8080);

```

HTTP Server တစ်ခုတည်ဆောက်ဖို့အတွက် `http.createServer()` Function ကိုသုံးရပါတယ်။ လက်ခံရရှိတဲ့ Request နဲ့ပက်သက်တဲ့အချက်အလက်တွေကို `createServer()` Callback Function ရဲပထမ Parameter နဲ့ လက်ခံထားပေးမှာဖြစ်ပြီး၊ Response အဖြစ်ဆောင်ရွက်ရမယ့် လုပ်ငန်းတွေကိုတော့ ဒုတိယ Parameter နဲ့ သတ်မှတ်နိုင် ပါတယ်။ နမူနာအရ Request နဲ့ပက်သက်တဲ့အချက်အလက်တွေဟာ `req` Object တဲ့မှာရှိနေမှာဖြစ်ပြီး Response ကို စီမံဖို့အတွက် `res` Object ကို အသုံးပြုရမှာ ဖြစ်ပါတယ်။

HTTP Response တစ်ခုမှာ Response Header နဲ့ Response Body ဆိုပြီးနှစ်ပိုင်းရှိရမှာဖြစ်တဲ့အတွက် လိုင်းနံပါတ် (c) မှာ `res.writeHead()` Function နဲ့ Response Header ကို သတ်မှတ်ပေးထားပါတယ်။ အဲဒီလို သတ်မှတ် ရာမှာ ပထမ Parameter အနေနဲ့ Status Code ကိုပေးရပြီး၊ ဒုတိယ Parameter အနေနဲ့ Header Option တွေပေး ရပါမယ်။ နမူနာမှာ Status Code အနေနဲ့ 200 OK ကို သတ်မှတ်ထားပြီး Response Header Option အနေနဲ့ `content-type: text/html` လို့ သတ်မှတ်ထားပါတယ်။ ပုံမှန်အားဖြင့် Response Header တစ်ခုမှာ Status Code, HTTP Version, Content Type, Content Size, Encoding, Modified Date စာဖြင့် ပါဝင်သင့်တဲ့အချက် တွေအများကြီးရှိပါတယ်။ နမူနာမှာ Content Type တစ်မျိုးတည်းကိုသာ သတ်မှတ်ထားပေမယ့် လက်တွေမှာ Option တွေ အားလုံးကို ထည့်သွင်းပေးသင့်ပါတယ်။

Response Header သတ်မှတ်ပြီးတဲ့အခါမှာ Response Body ကို `res.write()` Function နဲ့သတ်မှတ်ပေးရ ပါ တယ်။ နမူနာအရ Request တစ်ခုဝင်လာတိုင်းမှာ ကျွန်တော်တို့ရဲ့ HTTP Server က Hello, World! ဆိုတဲ့ Response Body ကို အမြဲပြန်ပေးသွားမှာဖြစ်ပါတယ်။ Response Header နဲ့ Body သတ်မှတ်ပြီးလို့ ပြည့်စုံပြီးရင် `res.end()` Function ကို Run ပေးရပါတယ်။ ဒီတော့မှ Response လုပ်ငန်းပြီးပြတ်သွားပြီဖြစ်ကြောင်း Server ရော Request ပြုလုပ်သူ Client ပါ သိရှိနိုင်မှာဖြစ်ပါတယ်။

Server ကို စတင်အလုပ်လုပ်စေပေး `http.listen()` Function ကိုသုံးရပါတယ်။ `listen()` Function နဲ့ အတူ Server အလုပ်လုပ်ရမယ့် Port နံပါတ်ကို တွဲဖက်ပေးရပါတယ်။ နမူနာမှာ 8080 လို့ ပေးထားတဲ့အတွက် အခုနေ ဒီ HTTP Server Code ကို node Command နဲ့ Run လိုက်ရင် HTTP Server တစ်ခုက Port နံပါတ် 8080 မှာ စတင် အလုပ်လုပ် သွားမှာ ဖြစ်ပါတယ်။ Web Browser ကိုဖွင့်ပြီး [localhost:8080](http://localhost:8080) ကို Request လုပ်ကြည့်ရင် Hello, World! ဆိုတဲ့ Response Body ကို လက်ခံရရှိမှာပဲဖြစ်ပါတယ်။

NodeJS HTTP Module ဟာ Low Level Module တစ်ခုဖြစ်ပါတယ်။ ဆိုလိုတာက၊ သူက အခြေခံလုပ်ဆောင်ချက်တွေကိုသာ ပေးထားပြီး၊ Request တစ်ခုလက်ခံရရှိတဲ့အခါ ဆောင်ရွက်သွားစေလိုတဲ့ အသေးစိတ် အလုပ်တွေအားလုံးကို ကိုယ်တိုင်ရေးသား သတ်မှတ်ပေးရမှာဖြစ်ပါတယ်။ HTTP Protocol

အကြောင်းကို သိရှိထားမှသာ NodeJS HTTP Module ကို အသုံးချခိုင်မှာဖြစ်ပါတယ်။ HTTP အကြောင်းကို Professional Web Developer ရဲ့ အခန်း (၁) မှာ ဖော်ပြထားပါတယ်။

Request နဲ့အတူ လက်ခံရရှိတဲ့ URI Path ပေါ်မှုတည်ပြီး HTML Document တွေကို Response ပြန်ပေးနိုင်တဲ့ HTTP Server တစ်ခုကို အခုလိုတည်ဆောက်နိုင်ပါတယ်။

### JavaScript

```

1. var http = require("http");
2. var fs = require("fs");
3. var docroot = "/path/to/docroot";
4.
5. http.createServer(function(req, res) {
6.
7.     var path = docroot + req.url;
8.     var result = fs.existsSync(path);
9.
10.    if (result) {
11.        var body = fs.readFileSync(path, "utf8");
12.        res.writeHead(200, { "Content-Type": "text/html" });
13.        res.write(body);
14.    } else {
15.        res.writeHead(404);
16.        res.write("404 Not found");
17.    }
18.
19.    res.end();
20. }) .listen(8080);
21.
22. console.log("HTTP server running at port 8080...");
```

လိုင်းနံပါတ် (၇) မှာ Request URL နဲ့ Document Root Directory ကိုပေါင်းစပ်ပြီး ဖိုင် Path ကို တည်ဆောက်ထားပါတယ်။ လိုင်းနံပါတ် (၈) မှာတော့ fs.existsSync() ကိုသုံးပြီး ဖိုင် ရှိမရှိ စီစဉ်ထားပါတယ်။ ဖိုင်ရှိတယ်ဆိုရင် Response Body အဖြစ် ဖိုင် Content ကို ပြန်ပေးပြီး ဖိုင်မရှိရင်တော့ 404 Not found ကို ပြန်ပေးထားပါတယ်။ ဒီနည်းနဲ့ Request URL ပေါ်မှုတည်ပြီး ကိုက်ညီတဲ့ HTML Document ကိုပြန်ပေးနိုင်တဲ့ Web Server တစ်ခုကိုရရှိ နိုင်ပါတယ်။ တစ်ကြောင်းချင်း ရှင်းမနေတော့ပါဘူး။ ရေးသားထားတဲ့ Code ကိုသာ လေ့လာကြည့်လိုက်ပါ။

လက်တွေပြည့်စုံဖို့ရင် ဖိုင် Permission စီစဉ်တဲ့ကိစ္စတွေ၊ Directory Index သတ်မှတ်တဲ့ကိစ္စတွေနဲ့ အခြား အထွေထွေ လိုအပ်ချက်တွေရှိသေးတဲ့အတွက် ကိုယ်တိုင်ရေးသားမယ့်အစား အသင့်ရှိနေတဲ့ Package တွေကို အသုံးပြုသင့်ပါတယ်။ Connect နဲ့ Serve Static ဆိုတဲ့ Package တွေကိုအသုံးပြုပြီး Static Web Server တစ်ခု ကို ရရှိနိုင်ပါတယ်။ NPM နဲ့ အခုလို Install လုပ်ယူနိုင်ပါတယ်။

```
$ sudo npm install -g connect serve-static
```

Install လုပ်ပြီးပြဆိုရင် Static Web Server တစ်ခုကို အလွယ်တစ်ကူ ရေးသားတည်ဆောက်နိုင်ပါတယ်။

### JavaScript

```
var connect = require('connect');
var serveStatic = require('serve-static');
connect().use(serveStatic("/path/to/docroot")).listen(8080);
```

HTTP Module ကို တိုက်ရိုက်မသုံးတော့ပဲ Connect Package နဲ့ HTTP Server တစ်ခုတည်ဆောက်လိုက်ခြင်း ဖြစ်ပါတယ်။ နမူနာအရ serveStatic() Function အတွက် Parameter အဖြစ်ပေးလိုက်တဲ့ Directory ကို Document Root အဖြစ် NodeJS Web Server က သုံးပေးသွားမှာဖြစ်ပါတယ်။

NodeJS နဲ့ HTTP Server တစ်ခုကို **Request** ပြုလုပ်လိုတဲ့အခါ http.get() ကို သုံးနိုင်ပါတယ်။ Request ပြုလုပ် လိုတဲ့ URL နဲ့ Callback Function တိုကိုပေးရပါတယ်။ Server က ပြန်ပေးလာတဲ့ Response ကို Callback Parameter အဖြစ် လက်ခံနိုင်ပါတယ်။ jQuery ရဲ့ \$.get() Function နဲ့ သဘောတရား ဆင်တူပါတယ်။ ဥပမာ –

### JavaScript

```
1. http.get("http://www.example.com/", function(res) {
2.   res.setEncoding("utf8");
3.   res.on("data", function (chunk) {
4.     console.log(chunk);
5.   });
6. });
```

Server ပြန်ပေးလာတဲ့ Response ကို res Variable နဲ့ Callback Function မှာ လက်ခံထားပါတယ်။ Response Header ကို လိုချင်ရင် res.headers Property ကနေ ရယူနိုင်ပါတယ်။ Response Status Code ကိုရယူလိုရင် တော့ res.statusCode() Function ကနေ ရယူနိုင်ပါတယ်။ Response Body ကိုတော့ နမူနာလိုင်းနဲ့ပါတ် (၃) မှာ ပေးထားသလို res.on('data') Event ရဲ့ Callback Function ကနေ ရယူနိုင်ပါတယ်။ သတိပြုရမှာက http.get() Function ဟာ Request Method အနေနဲ့ GET ကို အမြဲအသုံးပြုပေးသွားမှာဖြစ်ပါတယ်။ အခြား Request Method တွေနဲ့ Request ပြုလုပ်လိုတယ်ဆိုရင်တော့ http.request() ကိုအသုံးပြုနိုင်ပါတယ်။

### JavaScript

```
1. var options = {
2.   hostname: "www.example.com",
3.   port: 80,
4.   path: "/api",
5.   method: "POST"
6. };
7.
```

```

8. var req = http.request(options, function(res) {
9.     res.setEncoding("utf8");
10.    res.on("data", function (chunk) {
11.        console.log(chunk);
12.    });
13. });
14.
15. req.on("error", function(e) {
16.     console.log(e.message);
17. });
18.
19. req.write("foo=bar");
20. req.end();

```

http.request() ကိုအသုံးပြုဖို့အတွက် Request Option နဲ့ Callback Function ကိုပေးရပါတယ်။ Option အနေနဲ့ Request Header မှာပါဝင်ရမယ့် အချက်အလက်တွေကို ပေးရပါတယ်။ နမူနာမှာ hostname, port, path, method စိတ် Request Header Option တွေကို သတ်မှတ်ပေးထားတာကို တွေ့ရနိုင်ပါတယ်။ Request နဲ့အတူ တွဲဖက်ပေးပို့လိုတဲ့ Query String ရှိခဲ့ရင် နမူနာလိုင်းနံပါတ် (၁၉) မှာဖော်ပြထားသလို req.write() Function ကို အသုံးပြုနိုင်ပါတယ်။ ထူးခြားချက်အနေနဲ့ Request ပြီးစီးတိုင်းမှာ req.end() Function ကို Run ပေးရပါတယ်။ ဒီတော့မှ ဒီ Request ကို လက်ခံမယ့် Server က ဒါပာ Request ရဲ့ အဆုံးသတ်ဖြစ်ကြောင်းကို သိရှိနိုင် မှာဖြစ်ပါတယ်။ http.get() Function ကေတ္တာ req.end() ကို အလိုအလျောက်ထည့်ပေးတဲ့အတွက် သိုးခြား ထပ် Run စရာမလိုပါဘူး။ http.request() သုံးရင်တော့ req.end() ကို ထည့်သွင်းရေးသားပေးရပါတယ်။

HTTP Module နဲ့အတူတွဲဖက်အသုံးပြုဖို့ အသုံးဝင်တာကတော့ URL Module နဲ့ Query String Module တို့ပဲဖြစ်ပါတယ်။ URL Module ရဲ့ parse() နဲ့ format() Function နှစ်ခုက အသုံးဝင်ပါတယ်။

parse() Function က ပေးလိုက်တဲ့ URL ကို Object ပြောင်းပေးပါတယ်။ ဥပမာ -

### JavaScript

```

var url = require("url");
url.parse("http://example.com:8080/foo/bar.html");

{ protocol: 'http:',
  slashes: true,
  auth: null,
  host: 'example.com:8080',
  port: '8080',
  hostname: 'example.com',
  hash: '#two',
  search: '?q=keywords',
  query: 'q=keywords',
  pathname: '/foo/bar.html',
  path: '/foo/bar.html?q=keywords',
  href: 'http://example.com:8080/foo/bar.html?q=keywords#two' }

```

format () Function ကတော့ ပေးလိုက်တဲ့ Object ကို URL ပြန်ပြောင်းပေးပါတယ်။ ဥပမာ -

### JavaScript

```
url.format({
  protocol: 'http',
  hostname:'example.com',
  port: 8080,
  pathname: '/foo/bar.html'
});

'http://example.com:8080/foo/bar.html'
```

Query String Module ကတော့ မူလကတည်းက Global Scope မှာရှိနေပြီးသားမို့ require() တောင်လို စရာ မလိုပဲ querystring Object အနေနဲ့ တန်းသုံးလိုပါတယ်။ querystring.stringify() Function ၏ Object တွေကို Query String ပြောင်းပေးပါတယ်။ ဥပမာ -

### JavaScript

```
querystring.stringify({ foo: 'bar', baz: ['qux', 'quux'] });

'foo=bar&baz=qux&baz=quux'
```

querystring.parse() Function ကတော့ Query String တစ်ခုကို Object ပြောင်းပေးပါတယ်။ ဥပမာ -

### JavaScript

```
querystring.parse('foo=bar&baz=qux&baz=quux')

{ foo: 'bar', baz: ['qux', 'quux'] }
```

## Other Build-in Modules

ဖြည့်စွက်မှတ်သားသင့်တဲ့ NodeJS Build-in Module တစ်ချို့ကို ဈွေးထုတ်ဖော်ပြပေးလိုက်ပါတယ်။

**Crypto** – Hash, Encryption, Description စတဲ့လုပ်ငန်းများအတွက်လိုအပ်တဲ့အခြေခံလုပ်ဆောင်ချက်တွေ ပါဝင်တဲ့ Module တစ်ခုဖြစ်ပါတယ်။ အသုံးများတဲ့ Hash နဲ့ Encryption Algorithm တွေအားလုံး အသင့်ပါဝင်ပါတယ်။ ဥပမာ - String တစ်ခုရဲ့ MD5 (သို့မဟုတ်) SHA1 Hash Summary ကို အခုလိုရယူနိုင်ပါတယ်။

## JavaScript

```
var crypto = require('crypto');

var md5 = crypto.createHash('md5').update('Apple').digest("hex");
// 9f6290f4436e5a2351f12e03b6433c3c

var sha1 = crypto.createHash('sha1').update('Apple').digest("hex");
// 476432a3e85a0aa21c23f5abd2975a89b6820d63
```

Apple နေရာမှာ Hash Summary ရရှိလိုတဲ့ Text နဲ့အစားထိုးပေးနိုင်ပါတယ်။

**DNS** – OS ရဲ့ Domain Name Resolution စနစ်ကို အသုံးပြုပေးတဲ့ Module ဖြစ်ပါတယ်။ Domain Name တစ်ခု ရဲ့ IP address တွေကို ရယူဖို့အတွက် အသုံးပြုနိုင်ပါတယ်။ ဥပမာ –

## JavaScript

```
var dns = require('dns');

dns.lookup('www.example.com', function (err, addresses) {
  console.log('addresses: ' + addresses);
});

// addresses: 93.184.216.34
```

**Net** – Client-server Model နဲ့အလုပ်လုပ်တဲ့ Network ပရိုဂရမ်တွေတည်ဆောက်ရာမှာ အသုံးပြုနိုင်တဲ့ Module ဖြစ်ပါတယ်။ Net Module ကိုသုံးပြီး Server ပရိုဂရမ်တစ်ခုကို အခုလိုတည်ဆောက်နိုင်ပါတယ်။

## JavaScript

```
1. var net = require('net');
2.
3. var server = net.createServer(function(conn) {
4.   console.log('Client connected');
5.
6.   conn.on('end', function() {
7.     console.log('Client disconnected');
8.   });
9.
10.  conn.write('Hello!\n');
11.  conn.pipe(conn);
12. });
13.
14. server.listen(8124, function() {
15.   console.log('Server started');
16. });
```

HTTP Module မှာလိုပဲ createServer() ကိုသုံးပြီး Server တစ်ခုတည်ဆောက်နိုင်ခြင်းဖြစ်ပါတယ်။ Client တွေ ချိတ်ဆက်လာတဲ့အခါ ဆောင်ရွက်ရမယ့်လုပ်ငန်းတွေကို Callback Function အနေနဲ့ သတ်မှတ်ထားနိုင်ပါ

တယ်။ Client တွေ ပေးပို့လိုတဲ့အချက်အလက်တွေကို write() Function နဲ့ပေးပို့နိုင်ပါတယ်။ လိုင်းနံပါတ် (၁၁) မှာ conn.pipe(conn) လိုပေးသားထားတဲ့အတွက် Client ပေးပို့လဲတဲ့ အချက်အလက်တွေကို Client ထံ ပြန်လည် ပေးပို့လိုက်မှာ ဖြစ်ပါတယ်။ ဒါကြောင့် Client က Server ကိုချိတ်ဆက်ပြီး တစ်ခုချေပေးပို့ရင်၊ သူပေးပို့လိုက်တဲ့ အချက် အလက်ကို သူပြန်ရမှာဖြစ်ပါတယ်။ HTTP Module မှာလိုပဲ Server ကိုစတင်ဖို့ အတွက် listen() Function နဲ့အတူ Port နံပါတ်ကို သတ်မှတ်ပေးနိုင်ပါတယ်။

Net Module ကိုသုံးပြီး Client ပရိုကရမ်တစ်ခုကို အခုလိုပေးသားနိုင်ပါတယ်။

### JavaScript

```

1. var net = require('net');
2.
3. var client = net.connect({ port: 8124 }, function() {
4.     console.log('connected to server!');
5.     client.write('Hi!\n');
6. });
7.
8. client.on('data', function(data) {
9.     console.log(data.toString());
10.    client.end();
11. });
12.
13. client.on('end', function() {
14.     console.log('Disconnected from server');
15. });

```

connect() Function ကိုသုံးပြီး Server ပရိုကရမ်ကို ချိတ်ဆက်ရခြင်းဖြစ်ပါတယ်။ Connect လုပ်တဲ့အခါ အသုံးပြုရ မယ့် Port နံပါတ်ကို Option အနေနဲ့ တွဲဖက်ထည့်သွင်းပေးရပါတယ်။ Server ထံပေးပို့လိုတဲ့ အချက်အလက်တွေကို write() Function နဲ့ပေးပို့နိုင်ပါတယ်။ နမူနာလိုင်းနံပါတ် (၈) မှာ data Event နဲ့ Server ပြန်လည်ပေးပို့လဲတဲ့ အချက်အလက်တွေကို လက်ခံရရှိတဲ့အခါ ဆောင်ရွက်ရမယ့် Function ကို သတ်မှတ်ထားပါတယ်။

HTTP ဟာ Pull Protocol တစ်ခုဖြစ်ပါတယ်။ ဒါကြောင့် HTTP Client နဲ့ HTTP Server တို့ဆက်သွယ်တဲ့အခါ Client က Request ပြုလုပ်ခြင်းအားဖြင့် ဆက်သွယ်မှုကို အစပြုရပါတယ်။ Server က သူသဘောနဲ့သူ အချက်အလက်ပေးပို့ တယ်ဆိတ် မရှိပါဘူး။ Client Request ရှိမှာသာ Server Response ရှိမှာပါ။ Net Module နဲ့တည်ဆောက်ထားတဲ့ Network ပရိုကရမ်တွေမှာတော့ Client နဲ့ Server ချိတ်ဆက်ပြီးတဲ့နောက်မှာ၊ Client ကရော Server ကပါ အချက် အလက်တွေကို Full Duplex ဖုန်း အပြန်အလှန် ပေးပို့နိုင်မှာဖြစ်ပါတယ်။

**OS** – CPU Usage, Memory Usage, Server Uptime, Hostname စိတ် Operating System နဲ့ပက်သက်တဲ့ အချက်အလက်တွေကို သိရှိလိုရင် OS Module ကို အသုံးပြုရပါတယ်။

**Punycode** – Unicode Character တွေကို ASCII Character ပြောင်းဖို့သုံးပါတယ်။ Punycode နဲ့ ASCII ပြောင်းထားတဲ့ Character တွေကို Unicode လည်း ပြန်ပြောင်းနိုင်ပါတယ်။ ဥပမာ –

## JavaScript

```
var punycode = require("punycode");
punycode.encode("ဗော်");           // => nid
punycode.decode("nid");           // => ဗော်
```

**Util** – isArray(), isRegExp(), isDate() စသေး Validation Function တစ်ချိုပါဝင်တဲ့ Module တစ်ခုဖြစ်ပါတယ်။ အမိကအသုံးဝင်မှာကတော့ format() Function နဲ့ inherits() Function တို့ဖြစ်ပါတယ်။ format() Function ကို String Format လုပ်ဖို့အတွက် အသုံးပြုနိုင်ပါ တယ်။ ဥပမာ –

## JavaScript

```
var util = require('util');
var id = 1, err = "Error", date = "2015-04-15";
util.format("ID: %d, Msg: %s, Date: %s", id, err, date);

// => ID: 1, Msg: Error, Date: 2015-04-15
```

String ထဲမှာ Placeholder တွေထည့်သွင်းပြီး အစားထိုးအသုံးပြုရမယ့် တန်ဖိုးတွေကို format() Function နဲ့ တွဲဖက်ပေးနိုင်ခြင်းဖြစ်ပါတယ်။ ကိန်းဂဏန်းတွေအတွက် %d Placeholder ကိုသုံးရပြီး Text တွေ အတွက် %s Placeholder ကို သုံးရပါတယ်။ ဒီနည်းနဲ့ Variable တွေကို String နဲ့ တွဲဖော်ပြန့်အတွက် + Operator တွေသုံးပြီး တဲ့ဆက် နေစရာမလိုတော့ပဲ ပိုပြီးစနစ်ကျသွားမှာဖြစ်ပါတယ်။ console.log() Function မှာလည်း format() လိုပဲ Placeholder တွေ ထည့်သွင်း အသုံးပြုနိုင်ပါတယ်။ ဥပမာ –

## JavaScript

```
console.log("ID: %d, Msg: %s, Date: %s", id, err, date);

// => ID: 1, Msg: Error, Date: 2015-04-15
```

inherits() Function ကိုတော့ Object Constructor တွေ Inheritance လုပ်ဖို့အတွက် အသုံးပြုနိုင်ပါတယ်။

## JavaScript

```
var Person = function() {};
var User = function() {};
util.inherits(User, Person);
```

နှမှနာအရ User Constructor ဟာ Person Constructor ကို Inherit လုပ်သွားမှာဖြစ်ပါတယ်။

**zlib** – အချက်အလက်တွေကို Compress/Decompress လုပ်ဖို့သုံးပါတယ်။ Compress လုပ်ဖို့အတွက် Gzip, Deflate စသေး Method တွေကိုအသုံးပြုနိုင်ပါတယ်။ Deflate Method ကိုသုံးပြီး Content တွေကို အခုလို Compress လုပ်နိုင်ပါတယ်။

**JavaScript**

```
var zlib = require("zlib");
var input = "Hello, World!";

zlib.deflate(input, function(err, buffer) {
  console.log( buffer.toString('base64') );
});

// eJzzSM3JyddRCM8vyk1RBAAfngRq
```

Deflate အစား Gzip ကိုသုံးလိုရင် zlib.gzip() ကို အစားထိုးသုံးနိုင်ပါတယ်။ Compress လုပ်ထားတဲ့ Content တွေကို Decompress လုပ်လိုရင်တော့ zlib.unzip() ကိုသုံးနိုင်ပါတယ်။ ဥပမာ -

**JavaScript**

```
var buffer = new Buffer("eJzzSM3JyddRCM8vyk1RBAAfngRq", "base64");

zlib.unzip(buffer, function(err, buffer) {
  console.log(buffer.toString());
});

// Hello, World!
```

Unzip က အသုံးပြုထားတဲ့ Compression Method ကို အလိုအလျောက် Detect လုပ်ပြီး သင့်တော်တဲ့ Decompress Method ကို သုံးပေးသွားမှာဖြစ်ပါတယ်။

Build-in Module တွေထဲမှာ Buffer, Process, Stream, Readline စတဲ့ Performance Optimization အတွက် အရေးပါတဲ့ Low Level Module တွေ ပါဝင်ပါသေးတယ်။ Build-in Module စာရင်းအပြည့်အစုံကို ဒီနေရာမှာ လေ့လာနိုင်ပါတယ်။

<https://nodejs.org/api/>

**Third-party Modules**

Build-in Module တွေအပြင် Third-party Module တွေကိုလည်း NPM နဲ့ Install လုပ်ပြီး အသုံးပြုနိုင်ပါတယ်။ ဥပမာ - MySQL Database ကိုအသုံးပြုထားတဲ့ ပရိုဂရမ်တစ်ခုကို NodeJS နဲ့ တည်ဆောက်လိုတယ်ဆိုရင် Node MySQL Module ကို အသုံးပြုနိုင်ပါတယ်။ NPM နဲ့ အခုလို Install လုပ်နိုင်ပါတယ်။

```
$ npm install mysql
```

-g Option ထည့်သွင်းပြီး Install မလုပ်ထားတာကို သတိပြုပါ။ Command အနေနဲ့ Run လိုတဲ့ Package တွေ ကိုသာ Global Package အနေနဲ့ Install လုပ်သင့်ပြီး require() နဲ့ချိတ်ဆက်အသုံးပြုလိုတဲ့ Module တွေကို

Local Module အဖြစ်သာ Project Directory ထဲမှာ Install လုပ်ထားသင့်ပါတယ်။ ဒါတော့မှ Project အတွက်လိုအပ်တဲ့ Module တွေဟာ တစ်နေရာ တည်းမှာ စုစုပေါင်းဖြီးဖြစ်နေမှာဖြစ်ပါတယ်။ Install လုပ်ပြီးရင် MySQL နဲ့ အခုလို ချိတ်ဆက်ဆောင်ရွက်နိုင်ပါတယ်။

### JavaScript

```

1. var mysql = require('mysql');
2.
3. var conn = mysql.createConnection({
4.   host: 'localhost',
5.   user: 'root',
6.   password: '',
7.   database: 'dbname'
8. });
9.
10. conn.connect();
11.
12. conn.query('SELECT * FROM table', function(err, rows) {
13.   console.log(rows.length);
14. });
15.
16. conn.end();

```

mysql.createConnect() Function ကိုသုံးပြီး Connection တစ်ခုတည်ဆောက်ရပါတယ်။ အဲဒီလိုတည်ဆောက်ရာမှာ Host Name, User Name, Password, Database Name စတဲ့ တန်ဖိုးတွေကို တဲ့ဖက်ပေးရပါတယ်။ တည်ဆောက်ထားတဲ့ Connection Object ကနေတစ်ဆင့် connect() Function နဲ့ MySQL Server ကို ချိတ်ဆက်ရပါတယ်။ Database ပေါ်မှာ အလုပ်လုပ်စေလိုတဲ့ Query တွေကိုတော့ နမူနာလိုင်းနံပါတ် (၁၂) မှာ ဖော်ပြထား သလို query() Function နဲ့ ပေးပို့နိုင်ပါတယ်။ Query က SELECT Query ဆိုရင် ရရှိလာတဲ့ Rows တွေကို Callback Function ရဲ့ Parameter အနေ နဲ့ လက်ခံထားပေးမှာ ဖြစ်ပါတယ်။ Query က INSERT, UPDATE, DELETE Query တွေဆုံးရင်တော့ Result Object ကို Callback Function ရဲ့ Parameter အနေနဲ့ လက်ခံထားပေးမှာ ဖြစ်ပါတယ်။ ဥပမာ -

### JavaScript

```

conn.query('INSERT INTO table (value, value)', function(err, result) {
  console.log(result.insertId);
});

conn.query('DELETE FROM table WHERE id = 123', function (err, result) {
  console.log(result.affectedRows);
})

```

MySQL အကြောင်းကို Professional Web Developer ရဲ့ အခန်း (၉) မှာ ဖော်ပြထားပါတယ်။ Node MySQL ရဲ့ အသုံးပြုပုံ အသေးစိတ်ကို အောက်ပါလိပ်စာမှာ လေ့လာနိုင်ပါတယ်။

<https://www.npmjs.com/package/mysql>

နောက်အခန်းမှာ အခြား Database စနစ်တစ်ခုဖြစ်တဲ့ MongoDB အကြောင်းကို ဆက်လက်ဖော်ပြသွားမှာပါ။ Third-party Module တွေထဲမှာ လက်ရှိပါစာရေးနေဂျိန် အသုံးအများဆုံး Module တွေကတော့ ExpressJS, Grunt, Bower, Yeoman, Cordova, LESS, CoffeeScript, Jade, Karma စာတဲ့ Module တွေပဲဖြစ်ပါတယ်။ အဲဒီထဲက Yeoman, Bower နဲ့ Grunt တို့ အကြောင်းကို ဖော်ပြခဲ့ပြီးဖြစ်ပါတယ်။ အခန်း (၁၂) မှာ ExpressJS အကြောင်းကို ဖော်ပြသွားမှာဖြစ်ပြီး အခန်း (၁၆) မှာ Cordova အကြောင်းကို ဖော်ပြပေးသွားမှာဖြစ်ပါတယ်။

LESS ဟာ CSS Pre-processor နည်းပညာတစ်ခုဖြစ်ပြီး၊ Coffee Script ဟာ JavaScript Pre-processor နည်းပညာဖြစ်ပါတယ်။ Jade ကတော့ HTML Template System တစ်ခုဖြစ်ပါတယ်။ ကျွန်ုတော်အနေနဲ့ HTML/CSS/JavaScript တို့ကို မူလအတိုင်း အသုံးပြုရတာကို ပိုမိုနှစ်သက်ပါတယ်။ Pre-processor တွေ တစ်ဆင့်ခံပြီး ရေးလေ့မရှိ တဲ့အတွက် ထည့်သွင်းမဖော်ပြတော့ပါဘူး။ Pre-processor နည်းပညာတွေက ပိုမို သပ်ရပ်တဲ့ရေးထုံးနဲ့ ပြင်ဆင်ထိမ်းသိမ်း ရလွယ်တဲ့ Code တွေရေးနိုင်အောင် ဖြည့်စွက်ပေးထားတဲ့အတွက် လေ့လာထားမယ်ဆိုရင် အကျိုးရှိနိုင်ပါတယ်။

Karma ကတော့ Test Runner ဖြစ်ပါတယ်။ အခန်း (၂) မှာ TDD အကြောင်းဖော်ပြခဲ့စဉ်က Qunit Test Framework အကြောင်း ဖော်ပြခဲ့ဖူးတာကို မှတ်မိမိုးမှာပါ။ Karma က Qunit လို Test Framework မဟုတ်ပဲ Qunit (သို့မဟုတ်) အခြား Test Framework တစ်ခုနဲ့ ရေးသားထားတဲ့ Test တွေကို Run ပေးတဲ့ Test Runner ဖြစ်ပါတယ်။

Third-party Module တွေ အသုံးပြုပုံဟာ သက်ဆိုင်ရာ Module Author ပေါ်မှာမူတည်တဲ့အတွက် Module Author ပေးထားတဲ့ Documentation တွေကို လေ့လာ အသုံးပြု ကြရမှာပါ။ Third-party Module တွေကို NPM Package အနေနဲ့ [npmjs.org](https://npmjs.org) မှာ လိုအပ်သလို ရှာဖွေအသုံးပြနိုင်ပါတယ်။

## 10.7 – Custom Events

NodeJS မှာ Custom Event တွေဖန်တီးဖို့အတွက် Event Module ရဲ့ EventEmitter လုပ်ဆောင်ချက်ကို အသုံးပြုရပါတယ်။ ဥပမာ - ကျွန်ုတော်တို့က ဖြစ်ပေါ်တဲ့နေအပြောင်းအလဲတွေကို Event အနေနဲ့ Client ကို အသိပေးနိုင်တဲ့ Radio Station Service တစ်ခု တည်ဆောက်လိုတယ်ဆိုကြပါစို့။ အခုလိုရေးသားနိုင်ပါတယ်။

### JavaScript

```

1. var util = require('util');
2. var EventEmitter = require('events').EventEmitter;
3.
4. var Radio = function(station) {
5.     var self = this;
6.
7.     setTimeout(function() {
8.         self.emit('start', station);
9.     }, 0);
10.
11.    setTimeout(function() {
12.        self.emit('next', station);
13.    }, 5000);

```

```

14.
15.     setTimeout(function() {
16.         self.emit('close', station);
17.     }, 10000);
18. };
19.
20. util.inherits(Radio, EventEmitter);
21. module.exports = Radio;

```

လိုင်းနံပါတ် (၄) မှာ Radio Constructor ကိုကြော်လာတဲ့ အခါးများ မှာ သတ်မှတ်ထားပါတယ်။ ဒါ Constructor ကိုသုံးပြီး Object တစ်ခု တည်ဆောက်လိုက်တာနဲ့ start Event ကို Emit လုပ်ပေးဖို့ လိုင်းနံပါတ် (၇) မှာ သတ်မှတ်ထားပါတယ်။ setTimeout() ရဲ့ Timeout တန်ဖိုးကို ၀ လိုပေးထားတဲ့ အတွက် Object တည်ဆောက်လိုက်တာနဲ့ ချက်ခြင်း အလုပ် လုပ်သွားမှာဖြစ်ပါတယ်။

လိုင်းနံပါတ် (၁၁) မှာတော့ အချိန် (၅) စတုန်းကြာတဲ့ အခါး စတုန်းကြာတဲ့ အခါး သတ်မှတ်ထားပါတယ်။ လိုင်းနံပါတ် (၁၃) မှာတော့ အချိန် (၁၀) စတုန်းကြာတဲ့ အခါး close Event ကို Emit လုပ်ပေးဖို့ သတ်မှတ်ပေးထား ပါတယ်။ တစ်ကယ့်လက်တွေ့ ပရိုဂရမ်တွေမှာ၊ အခုလိုပုံသေသတ်မှတ်ထားတဲ့ အချိန်နဲ့ မဟုတ်ပဲပရိုဂရမ်မှာ အမှန်တစ် ကယ် လုပ်ဆောင်ချက်တစ်ခု လုပ်သွားတိုင်းမှာ this.emit() ကိုသုံးပြီး Event တွေ လွှတ်ပေးနိုင်မှာ ဖြစ်ပါတယ်။

လိုင်းနံပါတ် (၂၀) မှာ util.inherits() ကိုသုံးပြီး Radio Constructor က EventEmitter ကို Inherit လုပ်ထားတာကို သတ်မှတ်ပါ။ ဒါလို့ Inherit လုပ်ထားတဲ့ အတွက် Radio Object မှာ emit() Function ကို အသုံးပြု နိုင်ခြင်းဖြစ် ပါတယ်။ ဒါ Module ကိုအသုံးပြုထားတဲ့ ပရိုဂရမ်တစ်ခုကို အခုလို ရေးသားစမ်းသပ်ကြည့် နိုင်ပါတယ်။

### JavaScript

```

1. var Radio = require('./radio.js');
2.
3. var station = {
4.     name: 'City FM Radio',
5. };
6.
7. var radio = new Radio(station);
8.
9. radio.on('start', function(s) {
10.     console.log('%s "%s" STARTED', new Date(), s.name);
11. });
12.
13. radio.on('next', function(s) {
14.     console.log('%s "%s" NEXT SONG', new Date(), s.name);
15. });
16.
17. radio.on('close', function(s) {
18.     console.log('%s "%s" CLOSED', new Date(), s.name);
19. });

```

လိုင်းနံပါတ် (၇) မှာ radio Object တစ်ခုကို Radio Constructor သုံးပြီး တည်ဆောက်ထားပါတယ်။ လိုင်းနံပါတ် (၉) မှာတော့ start Event ဖြစ်ပေါ်တဲ့အခါ လုပ်ရမယ့် Function ကိုသတ်မှတ်ထားပြီး လိုင်းနံပါတ် (၁၃) နဲ့ (၁၇) တို့မှာ next နဲ့ close Event တွေ ဖြစ်ပေါ်တဲ့အခါ လုပ်ရမယ့် Function တွေကို သတ်မှတ်ပေးထားပါတယ်။ တန်လျှော့အားဖြင့် ကျွန်ုတ်တို့ သတ်မှတ်ထားတဲ့ Custom Event တွေကို ဒီ Code က အသုံးချမှု ခြင်းပဲဖြစ်ပါတယ်။

Event Function တွေမှာ သုံးထားတဲ့ s Variable ဟာ Event Emit လုပ်စဉ် Server Module က ပေးလိုက်တဲ့ တန်ဖိုးဖြစ်ပါတယ်။ ဒီနည်းနဲ့ Custom Event တွေကို သတ်မှတ်ယုံသာမက Event ဖြစ်ပေါ်တဲ့အခါ အသုံးပြုရမယ့် တန်ဖိုး တွေကိုပါ Event Emit လုပ်စဉ် တဲ့ဖက်ပေးပို့နိုင်ခြင်း ဖြစ်ပါတယ်။

ဒီ App ကို စမ်းကြည့်ရင် အခုလိုဂလဒ်ကို ရရှိမှာဖြစ်ပါတယ်။

```
$ node app.js
Thu Apr 16 2015 02:09:52 GMT+0630 (MMT) "City FM Radio" STARTED
Thu Apr 16 2015 02:09:57 GMT+0630 (MMT) "City FM Radio" NEXT SONG
Thu Apr 16 2015 02:10:02 GMT+0630 (MMT) "City FM Radio" CLOSED
```

radio Object မှာ on() Function ကိုအသုံးပြုပြီး Event ဖြစ်ပေါ်တဲ့အခါ အလုပ်လုပ်ရမယ် Function ကို သတ် မှတ်ထားတာကို တွေ့ရမှာဖြစ်ပါတယ်။ ကျွန်ုတ်တို့ Radio Constructor ကို တည် ဆောက်စဉ်က on() Function ထည့်သွင်းကြော်မထားပါဘူး။ ဒါကြောင့် on() Function ဟာ EventEmitter ကနေ ဆက်ခံထားတဲ့ Function ဖြစ်ကြောင်းနားလည့်ရမှာ ဖြစ်ပါတယ်။ on() အစား once() Function ကို အစားထိုးသုံးနိုင်ပါတယ်။ on() Function က Event ဖြစ်ပေါ် တိုင်းမှာအလုပ်လုပ်ပြီး once() Function ကတော့ Event ဖြစ်ပေါ်တဲ့ ပထမဆုံး တစ်ကြိမ်သာအလုပ်လုပ်မှာဖြစ်ပါတယ်။

ဒီနည်းနဲ့ Event Emitter ကိုသုံးပြီး NodeJS ပရိုဂရမ်တစ်ခုရဲ့ လုပ်ဆောင်ချက်တွေကို Event ပေါ်အခြေခံပြီး အလုပ် လုပ်တဲ့ Architecture နဲ့ တည်ဆောက်ထားနိုင်မှာပဲဖြစ်ပါတယ်။

## Conclusion

NodeJS ရဲအခိုကအားသာချက် (၃) ချက်ရှိတယ်လို့ ဆိုရပါမယ်။ ပထမတစ်ချက်ကတော့ JavaScript ဖြစ်ခြင်းပါပဲ။

ကျွန်ုတ် တို့အသုံးပြုနေတဲ့ Back-end နည်းပညာတွေ ဘယ်လိုပဲကဲ့ပြားပါစေ။ Client-side Programming အတွက် တော့ JavaScript ကိုပဲသုံးကြရတာပါ။ တန်လျှော့အားဖြင့် JavaScript ကို Developer တိုင်း ရင်းနှီးတက် ကျွမ်းပြီးဖြစ် တယ်လို့ ဆိုနိုင်ပါတယ်။ အဲဒီလို့ ရင်းနှီးကျွမ်းဝင်ပြီးသား Language ကို အသုံးပြုရေးသားနိုင်တဲ့ အတွက် NodeJS ကို အသုံးချမှုနိုင်ဖို့ အချိန်တွေ အများကြီးပေးပြီး ထပ်မံလေ့လာနေစရာမလိုတော့ပါဘူး။

နောက်တစ်ချက်ကတော့ Non-blocking I/O ဖြစ်ပါတယ်။ ကျွန်ုတ်တို့၏ App တွေမှာ Non-blocking I/O လုပ်ဆောင်ချက်ကို Thread တွေ Process တွေစီမံစရာမလိုပဲ အလွယ်တစ်ကူရရှိနိုင်တဲ့အတွက် စွမ်းဆောင်ရည်မြင့် App တွေကို အလွယ်တစ်ကူ တည်ဆောက်နိုင်ခြင်း ဖြစ်တယ်လို့လည်း ဆိုနိုင်ပါတယ်။

နောက်ဆုံးတစ်ချက်ကတော့ Even Driven Architecture ဖြစ်ပါတယ်။ ဒီ Architecture ရဲ့အကူအညီနဲ့ ပိုမြို့ Responsive ဖြစ်တဲ့ Real-time Network App တွေကို NodeJS နဲ့ တစ်ဆောက်နိုင်မှာဖြစ်ပါတယ်။ NodeJS ကို အထူးသဖြင့် I/O Heavy App တွေ၊ JSON API Back-end တွေနဲ့ Real-time App တွေ တည်ဆောက်ရာမှာ အဓိက အသုံးချသင့်ပါတယ်။

ကနေ့ခေတ်မှာ App တွေတည်ဆောက်တဲ့အခါ JSON API Back-end ကို အခြေခံတဲ့ App တွေအဖြစ် တည်ဆောက်ကြခြင်းဟာ Common Practice ဖြစ်နေပါပြီ။ ဒါအပြင် Client နဲ့ Server ကြား၊ အချက်အလက် တွေကို အချိန်နဲ့တစ်ပြီးညီ ပေးပို့နိုင်တဲ့ Real-time လုပ်ဆောင်ချက်ဆိုတာလည်း App တိုင်းလိုလိုအတွက် လိုအပ်ချက်တစ်ခု ဖြစ်လာနေပါပြီ။ ဒါကြောင့် NodeJS ဟာ Rockstar Developer တစ်ဦး မဖြစ်မနေ သိရှိထားရ မယ့် နည်းပညာတစ်ခုပဲဖြစ်ပါတယ်။

အသုံးပြုသူများလာပြီး၊ စနစ်ရဲ့ သက်တမ်း ရည်ကြာလာတာနဲ့အမျှ  
လက်ခံသိမ်းဆည်းပေးရမယ့် Data တွေလည်း များပြားလာမှာပဲဖြစ်ပါတယ်။  
အသုံးပြုထားတဲ့ Database က ရေရှည်မှာ Terabyte ပေါင်းများစွာရှိလာ  
နိုင်တဲ့ Data တွေကို လက်ခံသိမ်းဆည်းနိုင်ဖို့လိုပါတယ်။

### **Rockstar Developer Course**

Project Management, Web Service, Server Architecture  
NodeJS အစရှိသည် ဤစာအုပ်ပါ အကြောင်းအရာများကို  
စာရေးသူကိုယ်တိုင် သင်ကြားပေးခြင်းဖြစ်သည်။  
ဆက်သွယ်ရန် - (၀၉) ၇၃၀ ၆၅၄ ၆၂

<http://eimaung.com/courses>

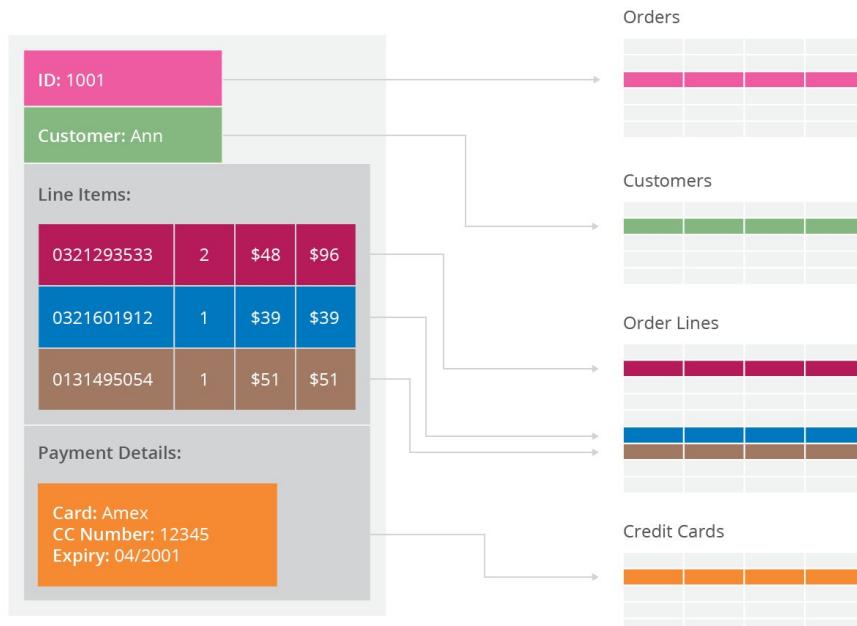
## အခန်း(၁၁) – MongoDB (NoSQL Database)

MongoDB ဟာ Document-oriented, NoSQL Database နည်းပညာတစ်ခုဖြစ်ပါတယ်။ Open Source နည်းပညာ တစ်ခုဖြစ်ပြီး NoSQL Database တွေထဲမှာ လက်ရှိလှသုံးအများဆုံး Database ဖြစ်ပါတယ်။

MongoDB ဟာ အချက်အလက်တွေကိုသိမ်းတဲ့အခါ BSON လို့ခေါ်တဲ့ JSON နဲ့ သဘောသဘာဝတူတဲ့ Data Format ကို အသုံးပြုပြီး၊ အချက်အလက်တွေစိမ်တဲ့ Administration Shell အတွက်လည်း JavaScript ကို အသုံးပြုထားပါတယ်။ Ajax, HTML5, NodeJS စတဲ့ နည်းပညာများနဲ့အတူ Software Development မှာ JavaScript ရဲ့အခန်းကဏ္ဍ၊ ထူးခြား အရေးပါလာတဲ့အချိန်နဲ့ အချိန်ကိုက် တွက်ပေါ်လာတဲ့ Database စနစ်တစ်ခု ဖြစ်ပြီး၊ ခေတ်သစ် Software Development ရဲ့ အရေးပါတဲ့အစိတ်အပိုင်း တစ်ရပ်ပဲဖြစ်ပါတယ်။

### 11.1 – NoSQL

Relational Database Management System (RDBMS) တွေဖြစ်တဲ့ MySQL, Oracle, MSSQL စတဲ့ Database စနစ်တွေမှာ၊ အမိကသဘောသဘာဝ (J) ချက်ရှိတယ်လို့ ဆိုနိုင်ပါတယ်။ ပထမတစ်ချက်က Data Model Schema ဖြစ်ပါတယ်။ Data တွေကို သိမ်းဆည်းနိုင်ဖို့အတွက် သိမ်းဆည်းရမယ့် Data Structure ကို ဦးဆုံး တည်ဆောက်ပေးရ ခြင်းဖြစ်ပါတယ်။



ပုံ (၁၁.၁) - Database Schema – RDBMS

ပုံ (၁၁.၁) ကိုလေ့လာကြည့်ပါ။ RDBMS တွေမှာ အချက်အလက်တွေကို သိမ်းဆည်းတဲ့အခါ Orders, Customers, Credit Cards စသဖြင့် အမျိုးအစားအလိုက် Table ကိုယ်စိန့် ခွဲခြားသိမ်းဆည်းရပါတယ်။ Table တစ်ခုစီမှာ ကိုယ်ပိုင် ဖွဲ့စည်းပုံ Data Structure ကိုယ်စိန့်ကြပါတယ်။ ဥပမာ – Customers Table မှာ Customer Name, Address, Phone စတဲ့ အချက်အလက်တွေ သိမ်းဆည်းထားနိုင်ပြီး၊ Order Lines Table မှာ Item Name, Category, Price စတဲ့အချက်အလက်တွေ သိမ်းဆည်းထားနိုင်ပါတယ်။ ဒီလိုမျိုး ခွဲခြားသိမ်းဆည်းထားတဲ့ Table တွေကနေ လိုချင်တဲ့ အချက်အလက် Data Set ကိုရရှိဖို့အတွက် SQL JOIN Statement တွေသုံးပြီး ရယူရပါတယ်။ အချက်အလက်တွေရဲ့ ဖွဲ့စည်းပုံ Structure နဲ့ ဆက်စပ်မှု Relationship တို့ကို အရင်ဆုံး Design လုပ်ရသလို၊ လက်တွေ့သိမ်းဆည်းတဲ့အခါ မှာလည်း သတ်မှတ်ထားတဲ့ Structure နဲ့အညီသာ သိမ်းဆည်းရမှာ ဖြစ်ပါတယ်။

RDBMS စနစ်တွေမှာ ရှိလေ့ရှိတဲ့ ဒုတိယတစ်ချက်ကတော့ ACID Compliance ဖြစ်ပါတယ်။ ACID ဆိုတာ Atomicity, Consistency, Isolation နဲ့ Durability တို့ကိုဆိုလိုတာပါ။ လိုရင်းကတော့ အချက်အလက် တိကျ သေချာမှုကို အာမခံ ခြင်းပဲဖြစ်ပါတယ်။ Transaction နဲ့ Rollback တို့လိုလုပ်ဆောင်ချက်တွေရဲ့ အကူအညီနဲ့ မမျှော်မှန်နဲ့ ပြဿနာတစ်ခု တစ်ရာကြောင့် အချက်အလက်တွေ အပြည့်အစုံမဟုတ်ပဲ တစ်ဝက်တစ်ပျက် သာ သိမ်းဆည်းမိခြင်းမျိုး မဖြစ်ရအောင် အာမခံပေးနိုင်ပါတယ်။ Foreign Key Relationship လုပ်ဆောင်ချက် အကူအညီနဲ့ တစ်နေရာမှာအသုံးချထားတဲ့ အချက် အလက်ကို သတိမမှုမိပဲပယ ဖျက်မိခြင်းကနေ ကာကွယ်ပေးပါတယ်။ Locking လုပ်ဆောင်ချက်အကူအညီနဲ့ တစ်ချိန် တည်းတစ်ပြိုင်တည်း အချက်အလက်တွေကို Access လုပ်မလို မလိုလားအပ်တဲ့ ဆုံးရှုံးမှုတွေ မဖြစ်ရအောင် အာမခံပေးကြပါတယ်။

RDBMS စနစ်တွေကို ဒီလိုအချက်တွေနဲ့ပြည့်စုံတဲ့အတွက် အခုချိန်ထိလည်း အမိက Database စနစ်များအဖြစ် အားထား အသုံး ပြုနေကြရဆဲဖြစ်ပါတယ်။ ဒါပေမယ့် ခေတ်တွေ ပြောင်းလဲလာတာနဲ့အမှု Data တွေရဲ့ သဘောသဘာဝကလည်း ပြောင်းလဲလာတဲ့အတွက်၊ အခုနောက်ပိုင်းမှာ RDBMS တွေရဲ့ လုပ်ဆောင်ချက်တွေနဲ့ မလုံလောက်တော့တဲ့ အခြေအနေ တွေ ဖြစ်ပေါ်လာပါတယ်။

## Scalability

ကနေ့ခေတ် Software တွေဟာ အင်တာန်ကိုအခြေခံတဲ့ Distributed Software တွေဖြစ်ကြပါတယ်။ အင်တာန်ကို အဆက် အသွယ်ကနေ့တစ်ဆင့် မည်သူမဆိုသုံးလို့ရတဲ့အတွက် တစ်ယောက်စ နှစ်ယောက်စ လူပြောများ လာရာကနေ ရှုတ်တရာ် ထင်ရှားသွားတဲ့ App တစ်ခုဟာ နေ့ချင်းညျှင်း အသုံးပြုသူ သန်းချိရှိသွားနိုင်ပါတယ်။ Database စနစ်က အဲဒီလိုအခြေအနေ ဖြစ်ပေါ်လာတဲ့အခါ လက်ခံအလုပ်လုပ်နိုင်ဖို့လိုပါတယ်။

Database တစ်ခုကို 500GB Storage Capacity ရှိတဲ့ Server တစ်ခုမှာ Setup လုပ်ထားတယ် ဆိုပါစို့။ အချက်အလက်တွေများလာလို 500GB နဲ့ မလောက်တော့လို Storage Capacity ထပ်တိုးရမယ်ဆိုရင် ပုံမှန်အားဖြင့် အဲဒီ Server မှာ Hard Drive တစ်ခုကို သွားတိုးရမယ်သဘောဖြစ်ပါတယ်။ ဒီနည်းကို Vertical Scaling လိုခေါ်ပါတယ်။ Vertical Scaling ဟာ ကုန်ကျစရိတ်များပြီး စီမံခေါ်လဲပေါ်ပါတယ်။

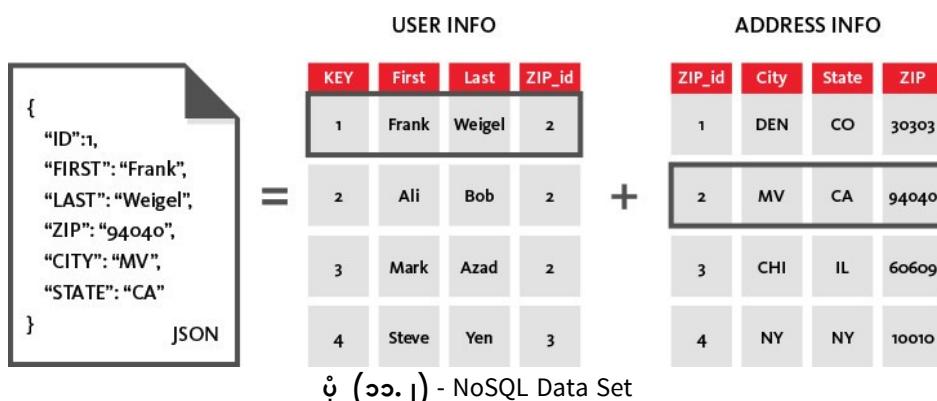
နောက်တစ်နည်းကတော့ Database ကို Network Cluster ပေါ်မှာ Deploy လုပ်နိုင်ပါတယ်။ Capacity တိုးဖို့လိုအပ် လာတဲ့အခါ မူလ Server ကို ထိစရာမလိုပဲ Cluster ထဲမှာ နောက်ထပ် Server Node တစ်ခု ထပ်တိုး

လိုက်ယုံပါပဲ။ ဒီနည်းကို Horizontal Scaling လိုပေါ်ပါတယ်။ အမှန်တစ်ကယ် Scalable ဖြစ်တဲ့ Database တစ်ခုဖြစ်ဖို့ရင် Horizontal Scale လုပ်နိုင်တဲ့ စနစ်ဖြစ်ဖို့လိုအပ်ပါတယ်။

RDBMS တွေမှာလည်း Cluster နဲ့ Replication လုပ်ဆောင်ချက်တွေ ရှိတက်ပေမယ့်၊ လက်တွေမှာ Storage Capacity တိုးစေဖို့ထက် Performance ကောင်းမြှုံးအတွက်သာ သုံးတက်ကြပါတယ်။ နည်းပညာသဘာဝအရ RDBMS တစ်ခုကို Cluster ပေါ်မှာ Deploy လုပ်ရတာ ခက်ခဲသလို၊ အကန်အသတ်တွေလည်း များပါတယ်။ RDBMS ထဲ မှာသမ်းဆည်းထား တဲ့ အချက်အလက်တွေဟာ တစ်ခုနဲ့တစ်ခု အပြန်အလှန်ဆက်စပ်နေကြတာပါ။ Record တစ်သန်းရှိလိုခိုပြီး Node တစ်ခု မှာ ငါးသိန်းထားပြီး နောက်တစ်ခုမှာ ငါးသိန်းထားမယ်လို လွယ်လွယ်ပြောလို မရနိုင်ပါဘူး။ Table တွေတစ်ခုနဲ့ တစ်ခုဟာ အပြန်အလှန် Relationship တွေရှိနေတဲ့အတွက် အဲ ဒီ Relationship Structure တစ်ခုလုံးကို Node တွေမှာ ဖြန့်ထား နိုင်ဖို့ စီစဉ်ရတော့မှာပါ။ လုံးဝမရနိုင်တာမျိုး မဟုတ်ပေမယ့် ခက်ခဲတဲ့ကိစ္စတစ်ခုဖြစ်ပါတယ်။

Scale လုပ်တယ်ဆိုတာ လုပ်လိုဂျာတယ်ဆိုယုံနဲ့ မပြီးပါဘူး။ လွယ်ဖို့လည်း လိုပါတယ်။ အခုမှစပြီး Release လုပ်တဲ့ App တစ်ခုအတွက် လူသန်းချိုံးနိုင်လောက်တဲ့ Infrastructural တစ်ခုကြိုတင်တည်ဆောက်ထားနိုင်ဖို့ မလွယ်ပါဘူး။ အမှန် တစ်ကယ် လူသန်းချိုံးလာပြီဆိုတဲ့အချိန်ရောက်တော့မှ ချက်ခြင်း Scale Up လုပ်လိုက် နိုင်ဖို့လိုပါတယ်။ ပြီးတော့ Scale က အမြဲတန်း Up နေရမှာ မဟုတ်ပါဘူး။ လိုအပ်ရင် ပြန် Down ဖို့လည်း လိုပါတယ်။ ရုတ်တရက် လူပြောများသွားလို တစ်ပါတ်လောက် လူသန်းချိုံးခဲ့ပေမယ့် နောက်ပိုင်းမှာ အသုံးပြုသူ ပြန်နည်းသွားမယ်ဆိုရင် Scale Down ပြန်လုပ်နိုင်ဖို့ လိုပါတယ်။ ဒီလို အလွယ်တစ်ကူ Scale Up / Scale Down လုပ်နိုင်ဖို့အတွက် Cloud နည်းပညာတွေ၊ Visualization နည်းပညာတွေ သုံးနိုင်ပေမယ့် အသုံးပြုတဲ့ Database စနစ်က Network Cluster ပေါ်မှာ Deploy လုပ်ရလွယ်ကူတဲ့ စနစ် ဖြစ်ဖို့လည်း လိုအပ်ပါတယ်။ ဒါဟာ RDBMS စနစ်တွေမှ ရှိနေတက်တဲ့ အကန်အသတ်တစ်ခုပဲဖြစ်ပါတယ်။

NoSQL Database စနစ်တွေကတော့၊ စတင်တိုတွင်ကတည်းက Scalability ကို ဦးစားပေးစဉ်းစားပြီး Horizontal Scale လုပ်နိုင်အောင် ရည်ရွယ်တည်ဆောက်ထားတဲ့ နည်းပညာတွေ ဖြစ်ကြပါတယ်။ NoSQL Database နဲ့ သမ်းဆည်းထားတဲ့ အချက်အလက်တွေဟာ အခြေခံအားဖြင့် သီးခြားစီရပ်တည်ကြပါတယ်။ အပြန်အလှန်ဆက်စပ်မှု Relationship ဆိုတာမရှိပါဘူး။



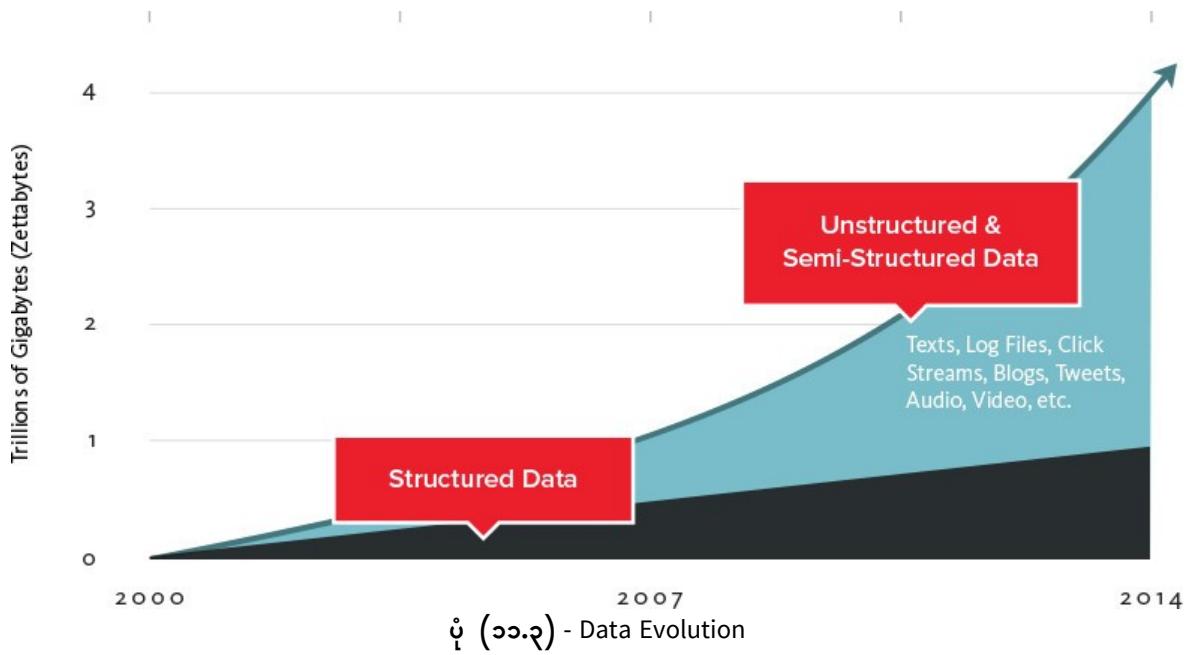
ပုံ (၁၁.၂) ကိုလေ့လာကြည့်ပါ။ ဘယ်ဘက်ခြမ်းက NoSQL ရဲ့ Data သိမ်းဆည်းပုံဖြစ်ပြီး ညာဘက်ခြင်းက RDBMS ရဲ့ Data သိမ်းဆည်းပုံ ဖြစ်ပါတယ်။ RDBMS မှာ User Info ဆိုတဲ့ Table နဲ့ Address Info ဆိုတဲ့ Table နှစ်ခုခွဲခြား သိမ်းဆည်း ထားခြင်းအားဖြင့် User Table မှာ လိပ်စာအပြည့်အစုံကို ထပ်ခါထပ်ခါ သိမ်းနေစရာမ လိုတော့ပဲ သက်ဆိုင်ရာ User ရဲ့ Zip Id ကိုသာ သိမ်းစွဲလိုပါတယ်။ User ရဲ့ လိပ်စာအပြည့်အစုံကို သိချင်ရင် Zip Id ကိုသုံးပြီး Address Info ကနေ ရယူနိုင်မှာပဲဖြစ်ပါတယ်။ NoSQL မှာတော့ အဲဒီလို နှစ်နေရာခွဲသိမ်းမနေပဲ အားလုံးကို စုစုပေါင်းသိမ်းဆည်းပါတယ်။ ဒီလိုစုစုပေါင်းလိုက်တဲ့အတွက် သက်ဆိုင်ရာ User ရဲ့ လိပ်စာ ကိုသိရမို့ နောက်တစ်နေရာကနေ သွားယူနေစရာမလို တော့တဲ့ အားသာချက်ကို ရရှိသွားပါတယ်။ ဒါပေမယ့် Data Duplicate တွေဖြစ်ပြီး Storage နေရာပိုယူသွားတဲ့ အား နည်းချက်တော့ ဖြစ်ပေါ်လာပါတယ်။

NoSQL Database စနစ်တွေက Cluster ပေါ်မှာ Deploy လုပ်နိုင်ဖို့ရည်ရွယ်ထားတဲ့အတွက် Storage နေရာပိုယူတာကို အမှု မထားပါဘူး။ နေရာပိုယူသွားပေမယ့် Relationship မလိုတော့တဲ့အတွက် Process လုပ်ရတာ သက်သာသွားသလို အချက်အလက်တွေကို Node တွေပေါ်မှာ ဖြန့်သိမ်းရတာ ပိုလွယ်သွားပါတယ်။

ဒုံးအပြင် မူလကတည်းက Scalability အတွက် ရည်ရွယ်ဖန်တီးထားတဲ့စနစ်ဖြစ်တဲ့အတွက် Load Balancing လို လုပ် ဆောင်ချက်တွေ Build-in ပါဝင်ကြလေ့ရှိပါတယ်။ Node တစ်ခု Fail ဖြစ်ခဲ့ရင် စနစ်ရပ်တန်မသွားစေပဲ နောက် Node တစ်ခုနဲ့ အလိုအလျောက် အစားထိုးပေးနိုင်တဲ့လုပ်ဆောင်ချက်တွေ Build-in ပါဝင်လေ့ရှုကြပါ တယ်။ RDBMS တွေမှာ တော့ ဒီလိုလုပ်ဆောင်ချက်တွေ လိုချင်ရင် ကိုယ်တိုင်အဆင့်ဆင့် Setup လုပ်ယူကြရမှာ ဖြစ်ပါတယ်။

## Unstructured Data

ကနေ့အချိန်မှာ ကျွန်ုတ်တို့ Software တွေနဲ့အတူ အသုံးပြုရမယ့် Data တွေရဲဖွေစည်းပုံကို ကျွန်ုတ်တို့ ကိုယ်တိုင် အမြဲ သတ်မှတ်ချင်လို့မရတော့ပါဘူး။ Software နဲ့အတူ အသုံးပြုရမယ့် Data တွေဟာ နေရာပေါ်း စုံကလာမှာပါ။ User တွေဆီကလည်းလာပါမယ်။ အဲဒီလိုလာတဲ့အခါ Desktop User ဆီကလာတဲ့ Data ရဲ့ ဖွေစည်းပုံနဲ့ Mobile User ဆီက လာတဲ့ဖွေစည်းပုံ တူချင်မှတူပါလိမ့်မယ်။ Third-party Service တွေထံကလည်း Data တွေကို ရယူအသုံးပြုရနိုင်ပါတယ်။ ဒီလို ရယူအသုံးပြုရောမှာ Service တစ်ခုနဲ့တစ်ခု အသုံးပြုတဲ့ Data Structure တူချင်မှတူပါလိမ့်မယ်။ ပြီးတော့ Data ဆိုတဲ့နေရာမှာ User Input Data တွေချည်းပဲ အသုံးဝင်တာ မဟုတ်ပါဘူး။ Error Log တွေ၊ Traffic Statistics တွေ၊ Usage Behavior တွေ၊ User Location Data တွေဟာ လည်း Software ရဲ့ လုပ်ဆောင်ချက်တွေ ပိုမိုကောင်းမွန်အောင် ဖန်တီးရာမှာ အသုံးဝင်တဲ့ Data တွေပဲ ဖြစ်ပါ တယ်။ ဒီလို နေရာစုံကနေ ပုံစံစုံနဲ့လာတဲ့ Data တွေကို ကြိုတင်သတ်မှတ် ထားတဲ့ Schema တစ်ခုနဲ့ သိမ်းဆည်း ဖို့ဆိုတာ သိပ်အဆင်မပြေတော့ပါဘူး။



ကနေ့အောင်မြင်နေတဲ့ Web App တွေ Mobile App တွေကိုလေ့လာကြည့်ရင် လုပ်ချက်သစ်တွေနဲ့ အမြဲမပြတ် အဆင့် မြှင့်တင်နေကြတာကို တွေ့ရနိုင်ပါတယ်။ နည်းပညာဟာ အမြဲပြောင်းလဲနေသလို ကျွန်ုတော်တို့ရဲ့ Software တွေကို လည်း နည်းပညာသစ်တွေနဲ့ အမြဲအဆင့်မြှင့်တင်နေနိုင်ဖို့လိုပါတယ်။ Software တစ်ခုမှာ လုပ်ဆောင်ချက် အသစ်တိုးလာတဲ့အခါ (သို့) လုပ်ဆောင်ပုံ ပြောင်းလဲသွားတဲ့အခါ နောက်ကွယ်ကနေ အချက်အလက်တွေ သိမ်းဆည်းပေးနေတဲ့ Database ထဲက Data Structure လည်း လိုက်လဲပြောင်းလဲဖို့ လိုအပ် တက်ပါတယ်။ လက်ရှိ Data တွေရှိနေပြီး အလုပ် လုပ်နေတဲ့ Structure တစ်ခုကို ပြောင်းရတယ် ဆိုတာ အတော်စဉ်းစားပြီးမှ လုပ်ရတဲ့အလုပ်တစ်ခုပါ။ အခန့်မသင့်ရင် ရှိနေတဲ့ Data တွေကို ထိခိုက်သွားနိုင်ပါတယ်။ ပြုလုပ်လိုက်တဲ့ Data Structure အပြောင်းအလဲကြောင့် လက်ရှိအလုပ်လုပ်နေတဲ့ လုပ်ဆောင်ချက်တွေကို ထိခိုက်မသွားစေဖို့လိုပါတယ်။ Data Structure ပိုမိုရှုပ်ထွေးလာလေ ပြောင်းလဲမှုပြု လုပ်ဖို့ ခက်ခဲ့လာလေပဲဖြစ်ပါတယ်။

တန်ည်းအားဖြင့်ပြောရရင်၊ ကြိုတင်သတ်မှတ်ထားတဲ့ Structure နဲ့ Relationship ဟာ Iterative Development နဲ့ အံမဝင်ဘူးလို့ ဆိုနိုင်ပါတယ်။ တိကျွေတဲ့ Requirement Specification ရှုဖို့ခက်ခဲတဲ့အတွက် Agile နည်းစနစ်

တွေဖြစ်တဲ့ Iterative Development, SCRUM စတဲ့နည်းစနစ်တွေနဲ့ ဖြေရှင်းနိုင်တယ်လို့ ကျွန်တော်တို့ ပြောခဲ့ရာ မှာ Coding ဆိုတဲ့ ရူထောင့်ကနေပဲ အမိကထား ပြောခဲ့ကြတာပါ။ အမှန်တော့ Data Structure ဆိုတာလည်း Requirement Specification ပေါ်မှာ အခြေခံတည်ဆောက်ရတာပါ။ Coding ကလွှာမှာ တိကျုတဲ့ Spec ရရှိဖို့ခက်တဲ့ပြဿနာကို Iterative Development နည်းစနစ်နဲ့ ဖြေရှင်းမယ်ဆိုရင် တိကျုတဲ့ Data Structure ရရှိဖို့ ခက်ခဲတဲ့ကိစ္စကို ဘယ်လို့ဖြေရှင်းမှာလဲ ဆိုတာ စဉ်းစားစရာဖြစ်လာပါတယ်။ လက်တွေမှာ Coding ပြင်ရခက်တယ်ဆိုတာထက် Data Structure ပြင်ရတာက ပိုခက်နိုင်ပါသေးတယ်။

NoSQL Database တွေက ဒီလိုအပ်ချက်တွေကို ဖြည့်စည်းပေးထားပါတယ်။ NoSQL Database တွေမှာ Data တွေကို စီမံဖို့အတွက် RDBMS တွေမှာလို့ SQL Query တွေကိုမသုံးပေမယ့် Developer တွေ ရင်းနှီးပြီးသားဖြစ်တဲ့ Object-oriented API တွေသုံးပြီး စီမံနိုင်အောင် စီစဉ်ပေးထားတက်ကြပါတယ်။

## 11.2 – Types of NoSQL Database

အမိက NoSQL Database အမျိုးအစား (၄) မျိုးရှိပါတယ်။ အဲဒါတွေကတော့ -

- Key-Value Store,
- Column Families Store,
- Document Store နဲ့
- Graph Database – တို့ပဲဖြစ်ပါတယ်။

**Key-Value Store** ဆိုတာ လက်တွေမှာ အချက်အလက်တွေကို Hasg Array ပုံစံ သိမ်းဆည်းတဲ့အရှိုးရှင်းဆုံး NoSQL နည်းပညာတစ်ခုဖြစ်ပါတယ်။ အချက်အလက်တစ်ခုကို သိမ်းဆည်းလိုတဲ့အခါ Key တစ်ခုနဲ့ သိမ်းဆည်းပြီး၊ ပြန်လည်ရယူ လိုတဲ့အခါ သတ်မှတ်ထားတဲ့ Key ကိုအသုံးပြုပြီး ပြန်လည်ရယူနိုင်ပါတယ်။ ဥပမာ -

### Redis

```
SET foo bar
```

foo ဆိုတဲ့ Key နဲ့ bar ဆိုတဲ့ တန်ဖိုးကိုသိမ်းဆည်းထားတဲ့အတွက်၊ နောင်လိုအပ်လိုအဲဒီတန်ဖိုးကို ပြန်လည်ရယူလိုတဲ့ အခါ အခုလိုရယူနိုင်မှာဖြစ်ပါတယ်။

### Redis

```
GET foo // => bar
```

တစ်နည်းအားဖြင့် ကျွန်တော်တို့ Code တွေရေးသားရာမှာသုံးလေ့ရှိတဲ့ List တွေ Array တွေကိုပဲ Data Storage အတွက် အသုံးပြုလိုက်ခြင်းဖြစ်ပါတယ်။ Programing Language တွေက Array တွေကိုစီမံဖို့အတွက် Function တွေ ပေးထားကြသလို ပဲ၊ Key-Value Store တွေကလည်း အလားတူစီမံနိုင်တဲ့ API တွေပေးထားတက်ကြပါတယ်။

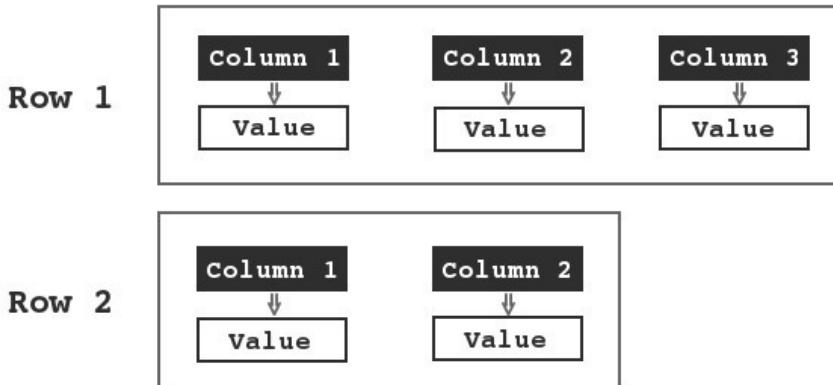
တစ်ချို့ Key-Value Store တွေဟာအချက်အလက်တွေကို Memory ပေါ်မှာပဲ သိမ်းဆည်းအလုပ်လုပ်တဲ့အတွက် အလွန် မြန်ပါတယ်။ လိုအပ်လို့ အချက်အလက်တွေကို File System ပေါ်မှာ အမြဲသိမ်းဆည်းလိုရင်လည်း သိမ်းဆည်း ပေးနိုင်တဲ့ လုပ်ဆောင်ချက်တွေ ပါဝင်ပါတယ်။

ရှုပ်ထွေးတဲ့ဖွဲ့စည်းပုံတွေမရှိပဲ၊ ခပ်ရိုးရိုးပဲ အလုပ်လုပ်တဲ့အတွက် Performance အလွန်ကောင်းကြပါတယ်။ တစ်ချို့ အသုံးများတဲ့ အချက်အလက်တွေကို File System ကနေခဏာခဏသွားယူနေစရာမလိုအောင် Memory ပေါ်မှာ Cache လုပ်ထားဖို့နဲ့ App အလုပ်လုပ်နေစဉ် တောက်လျှောက်မှတ်တမ်းတင်နေဖို့ လိုအပ်တဲ့ State Data တွေ၊ Traffic Statistic လို့ Data တွေ၊ Usage Behavior Data တွေကို သိမ်းဆည်းခြင်းကဲသို့ ကိစ္စရပ်များအတွက် အထူးသင့်တော်ပါတယ်။

ထင်ရှားတဲ့ Key-Value Store တွေကတော့ **Redis**, **DynamoDB** နဲ့ **Raik** တို့ပဲဖြစ်ပါတယ်။

**Column Families Store** တွေကို RDBMS တွေရဲ့ Table တွေနဲ့ ယုံးကြည့်နိုင်ပါတယ်။ RDBMS Table တွေမှာ Table ရဲ့ ဖွဲ့စည်းပုံ Structure ကို Column တွေနဲ့သတ်မှတ်ပြီး၊ အချက်အလက်တွေကို Row အနေနဲ့ သိမ်းဆည်းပါ တယ်။ Column Families Store တွေမှာတော့ အချက်အလက်တွေကို Column အနေနဲ့ တန်းစီသိမ်းဆည်းသွားနိုင်ပြီး၊ အချိုးတရာ အချက်အလက်တွေကို Row အနေနဲ့ စုစုပေါင်မှာ ဖြစ်ပါတယ်။

တစ်နည်းအားဖြင့် တန်ဖိုးတစ်ခုတိုးလိုတဲ့အခါ Row အနေနဲ့မတိုးပဲ၊ Row တစ်ခုရဲ့ Column အနေနဲ့ တိုးသွားရမှာဖြစ် ပါတယ်။



ပုံ (၁၁.၄) – Column Families Store

လက်ရှိထင်ရှားတဲ့ Column Families Store တွေကတော့ **Cassandra** နဲ့ **Hbase** တို့ပဲဖြစ်ပါတယ်။

Cassandra လို့ Column Families Store က Row တစ်ခုမှာရှိတဲ့ Data အားလုံးကို ဖိုင်တစ်ခုအနေနဲ့ သိမ်းဆည်းထား တက်ပါတယ်။ ဒါကြောင့် အတူတကွ အသုံးပြုဖို့လိုတက်တဲ့ အချက်အလက်တွေကို Row တစ်ခုတည်းမှာ စုစုပေါင်းထားခြင်း အားဖြင့်၊ အချက်အလက်တွေ ပြန်လည်ရယူတဲ့အခါ ပိုမိုမြန်ဆန်စေမှာပဲဖြစ်ပါတယ်။

**Document Store** ဆိုတာကတော့ အချက်အလက်တွေကို Table တွေ Row တွေနဲ့မသိမ်းပဲ Document တွေ အနေနဲ့ စုစုည်းသိမ်းဆည်းတဲ့ အမျိုးအစားဖြစ်ပါတယ်။ Document အနေနဲ့ အချက်အလက်တွေကိုသိမ်းဆည်းတဲ့အခါ ကြိုတင် သတ်မှတ်ထားတဲ့ Structure နဲ့ သိမ်းဆည်းဖို့မလိုပေမယ့် အချက်အလက်တွေရဲ့ ဖွံ့စည်းပုံကို JSON, BSON, XML စတဲ့ Format တွေနဲ့ သတ်မှတ်သိမ်းဆည်းနိုင်ပါတယ်။



```

<Key=CustomerID>
{
  "customerid": "fc986e48ca6" ← Key
  "customer": {
    "firstname": "Pramod",
    "lastname": "Sadalage",
    "company": "ThoughtWorks",
    "likes": [ "Biking", "Photography" ]
  }
  "billingaddress": {
    "state": "AK",
    "city": "DILLINGHAM",
    "type": "R"
  }
}

```

ပုံ (၁၁.၅) – Document Store

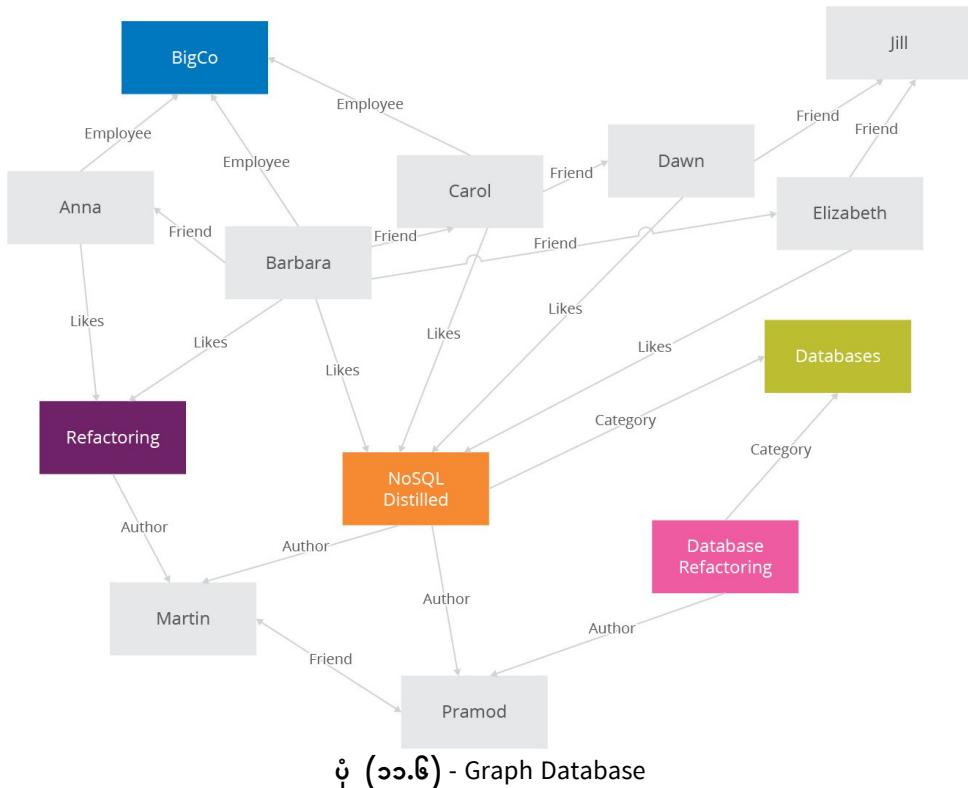
Source – [thoughtworks.com](http://thoughtworks.com)

Document Store နဲ့ Key-Value Store သဘောသဘာဝ သိပ်မကွာပါဘူး။ ကွာသွားတာက Key-Value Store မှာ Value က ရှိုးရှိုးတန်ဖိုးတစ်ခုဖြစ်ပြီး Document Store ရဲ့ Value ကတော့ Key-Value တွေစုစုည်းပါဝင်တဲ့ Document တစ်ခုဖြစ်ခြင်းပဲဖြစ်ပါတယ်။

Document Store တွေထဲမှာ ထင်ရှားတာကတော့ **MongoDB** နဲ့ **CouchDB** တို့ပဲဖြစ်ပါတယ်။

Key-Value Store တစ်ခုမှာ အချက်အလက်တွေကို Key ပေါ်အခြေခံပြီး စီမံရပါတယ်။ Value တစ်ခုကိုလိုချင်ရင် Key ကိုသုံးပြီးရယူရပါတယ်။ ပယ်ဖျက်လိုရင်လည်း Key ကိုသုံးပြီး ပယ်ဖျက်ရပါတယ်။ အခြေခံအားဖြင့် Key-Value Store က Value ကို စိတ်မဝင်စားပါဘူး။ Value ပေါ်အခြေခံပြီး အလုပ်လုပ်မပေးနိုင်ပါဘူး။ Document Store ရဲ့ ထူးမြား ချက်က Key ကိုသာမက Value ကိုပါ အသုံးချဖြီး အချက်အလက်တွေကို စီမံနိုင်ခြင်းပဲဖြစ်ပါတယ်။ ဒါကြောင့် Document Store တွေကို Content အခြေပြု App တွေတည်ဆောက်ရာမှာ အသုံးပြုဖို့ သင့်တော်ပါတယ်။ App အများ စုံသာ Content အခြေပြု App တွေဖြစ်ကြတဲ့အတွက် MongoDB လို Document Store ဟာ NoSQL Database တွေထဲမှာ လူသုံးပိုများတဲ့ အမျိုးအစားဖြစ်နေခြင်းပဲ ဖြစ်ပါတယ်။

**Graph Database** ဆိုတာကတော့ Graph Theory ပေါ်မှာအခြေခံပြီး အချက်အလက်တွေကို သိမ်းဆည်းတဲ့ Database အမျိုးအစားဖြစ်ပါတယ်။ Graph တစ်ခုမှာ အခြေခံအားဖြင့် Node, Property နဲ့ Edge ဆိုပြီး အစိတ်အပိုင်းသုံးရင် ပါဝင်ပါတယ်။ Node ဆိုတာ Record Set တစ်ခုလို့ လွယ်လွယ်မှတ်နိုင်ပါတယ်။ Node တို့မှာ Property နဲ့ Edge ဆိုတဲ့ အချက်အလက်အမျိုး အစားနှစ်မျိုးပါဝင်ပါတယ်။ Property က Node Content ဖြစ်ပြီး Edge ကတော့ Node နဲ့ အခြား Node တစ်ခု ဆက်စပ်ပုံကို သတ်မှတ်ထားတဲ့ အချက်အလက် တွေဖြစ်ပါတယ်။



ပုံ (၁၁.၆) - Graph Database

Source – [thoughtworks.com](http://thoughtworks.com)

ပုံ (၁၁.၈) လွှဲလာကြည့်ရင် ဟိုးသယ်ဘက်အပေါ်ထောင့်က **BigCo** ဆိုတာ Node တစ်ခုဖြစ်ပါတယ်။ သူမှာ Address, Product, Service စိတ် Property တွေပါဝင်နိုင်ပါတယ်။ သူအောက်နားက **Anna** ဆိုတာ နောက်ထပ် Node တစ်ခုဖြစ်ပါတယ်။ အဲဒီ Node မှာလည်း Email, Position, Marital Status စိတ် သူကိုယ်ပိုင် Property တွေ ရှိနိုင်ပါတယ်။ ထူးခြားချက်အနေနဲ့ **Anna** Node မှာ Employee ဆိုတဲ့ Edge တစ်ခုရှိနေပြီး အဲဒီ Edge က **BigCo** ကိုညွှန်းထားပါတယ်။ Employee Edge မှာလည်း Employee Since လို တွဲဖက်အချက်အလက်တွေ ပါဝင်နိုင်ပါ သေးတယ်။ ဒီ Relationship ကို Unidirectional Relationship လိုခေါ်ပါတယ်။ Graph ကိုကြည့်လိုက်တာ နဲ့ **Anna** ဟာ **BigCo** ရဲ့ Employee အနေနဲ့ ဆက်စပ်နေကြောင်းကို သိရနိုင်ပါတယ်။

အောက်နားက **Martin** နဲ့ **Pramod** စိတ် Node တို့ကို လွှဲလာကြည့်ရင် Friend Edge နဲ့ အပြန်အလှန် ဆက်စပ်မှု ရှိနေတာကိုတွေ့ရမှာဖြစ်ပါတယ်။ ဒီလိုဆက်စပ်မှုကိုတော့ Bidirectional Relationship လိုခေါ်ပါတယ်။ လက်တွေမှာ ပုံမှာပြထားသလို Edge တစ်ခုနဲ့ အပြန်အလှန်ချိတ်ဆက်မှာ မဟုတ်ပါဘူး။ **Martin** က Friend Edge နဲ့ **Pramod** ကိုညွှန်းပြီး **Pramod** က သူကိုယ်ပိုင် Friend Edge နဲ့ **Martin** ကို ပြန်ညွှန်းရမှာဖြစ်ပါတယ်။

ဒီတော့ Graph Database တွေမှာ အချက်အလက်တစ်ခုသိမ်းရင် Node အနေနဲ့သိမ်းရပြီး ပါဝင်တဲ့ Content တွေကို Property နဲ့ Edge ဆိုပြီး နှစ်ပိုင်းခွဲသိမ်းပေးရတဲ့သဘောပါ။ RDBMS တွေရဲ့ Table Relationship နဲ့ကွားဘာတာက၊ RDBMS တွေမှာ Relationship က Table ပေါ်မှာ သတ်မှတ်ခြင်းဖြစ်ပါတယ်။ Graph Database မှာ

တော့ Node တိုင်းမှာ ကိုယ်ပိုင် Relationship အချက်အလက်တွေ တွဲဖော်ပါဝင်မှာဖြစ်ပါတယ်။

Graph Database တွေဟာ အချက်အလက်ဆက်စပ်မှုပေါ်မှာ အမိကထားအလုပ်လုပ်တဲ့ App တွေနဲ့ သင့်တော်ပါတယ်။ အမြင်သာဆုံးနဲ့နာကတော့ Social Network ဖြစ်ပါတယ်။ Account တစ်ခုဟာ ဘယ်သူတွေနဲ့ Friend ဖြစ်နေတယ်၊ Follower ဘယ်လောက်ရှိတယ်၊ ဘယ်သူတွေကို Follow လုပ်ထားတယ် စတဲ့ဆက်စပ်မှုတွေကို စီမံရမှာဖြစ်လို့ Social Network လို့ App မျိုးဟာ Graph Database တွေကို အသုံးပြုသင့်တဲ့ App အမျိုးအစားပဲ ဖြစ်ပါတယ်။

Graph Database တွေထဲမှာ ထင်ရှားတာကတော့ **Noe4J** ဖြစ်ပါတယ်။

### 11.3 – MongoDB

MongoDB ဟာ Document Store အမျိုးအစား NoSQL Database တစ်ခုဖြစ်ပါတယ်။ Client-Server ပုံစံအလုပ်လုပ်တဲ့ Database ဖြစ်တဲ့အတွက် အချက်အလက်တွေကို Server မှာစုစည်းသိမ်းဆည်းပြီး Client Software တွေနဲ့ အဲဒီအချက်အလက်တွေကို ရယူစီမံနိုင်မှာဖြစ်ပါတယ်။ MongoDB Installer နောက်ဆုံး Version ကို [mongodb.org](http://mongodb.org) မှာ Download ရယူနိုင်ပါတယ်။ Installer နဲ့အတူ mongod, mongo, mongoexport, mongoimport, mongodump, mongorestore စတဲ့ Command Line Tool တွေ တစ်ခါတည်းပါဝင်လာမှာဖြစ်ပါတယ်။



mongod ဟာ MongoDB Server Administration Tool ဖြစ်ပြီး mongo ကတော့ MongoDB Server ကို ချိတ်ဆက်ပြီး Data တွေ စီမံနိုင်တဲ့ Client Tool တစ်ခုဖြစ်ပါတယ်၊ Mongo Shell လို့ခေါ်ပါတယ်။ Mongo Shell ဟာ Mozilla ရဲ့ JavaScript Engine ဖြစ်တဲ့ SpiderMonkey JavaScript Engine ကို အခြေခံထားခြင်း ဖြစ်ပါတယ်။ ဒါကြောင့် Mongo Shell ဆိုတာ အမှန်တော့ NodeJS Console လို့ JavaScript Console တစ်ခုပဲဖြစ်တယ်လို့ ဆိုနိုင်ပါတယ်။

ဆက်လက်ဖော်ပြုမယ့် Command နဲ့ Mongo Shell နဲ့နာတွေဟာ လက်တွေ့ကူးယူစမ်းသပ်နိုင်တဲ့ နဲ့နာတွေဖြစ်ပါတယ်။ ဒါပေမယ့် MongoDB Version တစ်ခုနဲ့တစ်ခု အချို့လုပ်ဆောင်ချက်တွေ ကဲ့ပြားတက်တာကိုသတိပြုပါ။ နဲ့နာမှာ အသုံးပြုဖော်ပြုမှာကတော့ MongoDB 2.4.9 ဖြစ်ပါတယ်။ စာဖတ်သူရဲ့ MongoDB က အခြား Version ဖြစ်ခဲ့မယ့်ဆိုရင် အချို့ Shell တွေအသုံးပြုပဲ ကဲ့ပြားသွားနိုင်ပါတယ်။

Ubuntu မှာဆိုရင်တော့ MongoDB ကို အခုလို့ Install လုပ်နိုင်ပါတယ်။

```
$ sudo apt-get install mongodb
```

တစ်ခုသတိပြုရမှာက Ubuntu Official Repository ထဲက MongoDB ဟာ Version တစ်ချို့ နောက်ကျနေနိုင်ပါတယ်။ ဒါကြောင့် နောက်ဆုံး Version ကိုမှ လိုချင်တာဆိုရင်တော့ [mongodb.org](http://mongodb.org) ကနေပဲ Download ရယူရမှာ

ဖြစ်ပါတယ်။ ဒီစာရေးနေချိန်မှာ နောက်ဆုံးထွက်ရှိထားတဲ့ MongoDB Version က 3.0.2 ဖြစ်ပြီ။ Ubuntu 14.04 Repository ထဲက MongoDB Version ကတော့ 2.4.9 ပဲ ရှိပါသေးတယ်။

Ubuntu မှာ apt-get နဲ့ MongoDB ကို Install လုပ်ခဲ့မယ်ဆိုရင် Install လုပ်ပြီးနောက် Ubuntu က MongoDB Server ကို အလိုအလျောက် စထားပေးမှာဖြစ်ပါတယ်။ apt-get နဲ့ Install မလုပ်ပဲ Manual Install ပြုလုပ်ခဲ့တာဆုံးရင်တော့ Install လုပ်ပြီးတဲ့အခါ MongoDB Server ကို အခုလို အစပြောပေးနိုင်ပါတယ်။

```
$ sudo service mongod start
```

Ubuntu မှာ MongoDB နောက်ဆုံး Version ကို Manual Install လုပ်နိုင်ဖို့အတွက် Installation Instruction အပြည့်အစုံကို အောက်ပါလိပ်စာမှာ လေ့လာနိုင်ပါတယ်။

<http://docs.mongodb.org/manual/tutorial/install-mongodb-on-ubuntu/>

Windows အတွက် Installation Instruction အပြည့်အစုံကိုတော့ အောက်ပါလိပ်စာမှာ လေ့လာနိုင်ပါတယ်။

<http://docs.mongodb.org/manual/tutorial/install-mongodb-on-windows/>

Install လည်းလုပ်ပြီး MongoDB Server ကိုလည်း Run ပြီးပြီးဆိုရင်တော့ mongo Command နဲ့ Mongo Shell ကို စတင်အသုံးပြုနိုင်ပြီဖြစ်ပါတယ်။ Mongo Shell အတွင်းမှာ help Command ကို Run ကြည့်ရင် အသုံးပြုနိုင်တဲ့ Command စာရင်းကိုတွေ့ရမှာဖြစ်ပါတယ်။

```
$ mongo

MongoDB shell version: 2.4.9
connecting to: test

> help

db.help()                                help on db methods
db.mycoll.help()                           help on collection methods
sh.help()                                 sharding helpers
rs.help()                                 replica set helpers
help admin                               administrative help
help connect                            connecting to a db help
help keys                                key shortcuts
help misc                                misc things to know
help mr                                  mapreduce

show dbs                                 show database names
show collections                         show collections in current database
show users                               show users in current database
show profile                            show most recent system.profile
... ...
```

MongoDB Database Server မှာရှိနေတဲ့ Database စာရင်းကို ကြည့်ရှုလိုရင် show dbs Command ကို သုံးရပါ တယ်။ အဲဒီ Database တွေထဲက အသုံးပြု စီမံလိုတဲ့ Database ကိုရွေးချယ်ဖို့ အတွက် use Command ကို သုံးရပါတယ်။

```
> show dbs
local 0.078125GB
> use mydb
switched to db mydb
> db
mydb
> show dbs
local 0.078125GB
> db.users.insert({ "name": "John Doe" });
> show dbs
local 0.078125GB
mydb 0.203125GB
>
```

နမူနာကိုလွှဲလာကြည့်ပါ။ ပထမတစ်ကြိမ် show dbs နဲ့ လက်ရှိ ရှိနေတဲ့ Database စာရင်းကို ဖော်ပြုစေတဲ့ အခါ local လိုခေါ်တဲ့ Database တစ်ခုရှိနေကြောင်းဖော်ပြုလာပါလိမ့်မယ်။ use mydb Command နဲ့ mydb အမည်ရှိ Database ကို ရွေးခိုင်းလိုက်တဲ့အခါ မူလက mydb Database ရှိမနေတဲ့အတွက်、Mongo Shell က mydb အမည်နဲ့ Database အသစ်တစ်ခုကို တည်ဆောက်ပေးသွားမှာ ဖြစ်ပါတယ်။ တနည်းအားဖြင့် use Command ကို လက်ရှိရှိနေတဲ့ Database တစ်ခုကို ရွေးချယ်ဖို့သုံးနိုင်သလို၊ Database အသစ်တစ်ခု တည်ဆောက်ဖို့လည်း သုံးနိုင်ခြင်းဖြစ်ပါတယ်။

db Command ကိုတော့ လက်ရှိရွေးချယ်ထားတဲ့ Database အမည်ဖော်ပြုစေဖို့ သုံးနိုင်ပါတယ်။ နမူနာမှာ db Command ကို Run ကြည့်တဲ့အခါ mydb လို ဖော်ပြပေးတာကို တွေ့ရမှာဖြစ်ပါတယ်။

show dbs နဲ့ နောက်တစ်ကြိမ် Database စာရင်းကြည့်တဲ့အခါ ကျွန်တော်တို့ တည်ဆောက်ထားတဲ့ mydb မပါဝင်သေးတာကို တွေ့ရမှာ ဖြစ်ပါတယ်။ စာရင်းထဲမှာ ဖော်ပြုစေဖို့အတွက် Database ထဲမှာ အနည်းဆုံး Document တစ်ခုရှိ ရမှာဖြစ်လို့ insert() Function ကိုသုံးပြီး Document တစ်ခု ဆက်လက်ထည့်သွင်းလိုက်ပါတယ်။ db.users. insert() လိုပြောလိုက်တဲ့အတွက် လက်ရှိ Database ဖြစ်တဲ့ mydb ထဲမှာ users အမည်နဲ့ Collection တစ်ခု တည်ဆောက်ပြီး Document တစ်ခုကို ထည့်သွင်းစေလိုက်ခြင်း ဖြစ်ပါတယ်။ နမူနာအရ Document Content အနေနဲ့ { "name": "John Doe" } ကို သတ်မှတ်ပေးထားပါတယ်။

RDBMS တွေမှာ Record တွေကို Table တွေနဲ့ စုစုပေါင်းထားသလိုပဲ MongoDB မှာ Document တွေကို Collection တွေနဲ့ စုစုပေါင်းထားရမှာ ဖြစ်ပါတယ်။

Document တစ်ခုထည့်သွင်းပြီးနောက် `show dbs` ကို ထပ် Run ကြည့်တဲ့အခါ Database စာရင်းထဲမှာ `mydb` ပါဝင်လာတာကို တွေ့ရမှာပဲဖြစ်ပါတယ်။

MongoDB က အချက်အလက်တွေကို Document ထဲမှာ သိမ်းတဲ့အခါ BSON Format ကို သုံးပါတယ်။ BSON Format ဟာ JSON နဲ့ ရေးသားပဲ ရေးထုံးအတူတူပါပဲ။ ကွာသွားတာကတော့ Data Type ဖြစ်ပါတယ်။ JSON ဟာ JavaScript ကနေ လာတဲ့အတွက် JavaScript လိုပဲကိန်းကဏ္ဍး အားလုံးအတွက် Number Data Type ကို သုံးပါတယ်။ ကိန်းပြည့်၊ ဒသမကိန်း ဆိုပြီးခွဲမထားပါဘူး။ BSON ကတော့ ကိန်းပြည့်တွေအတွက် Integer Data Type ကိုသုံးပြီး ဒသမကိန်းတွေအတွက် Double Data Type ကို သုံးပါတယ်။ ဒုအပြင် ဖြည့်စွက် Data Type တွေအဖြစ် Date နဲ့ Byte Array တို့ပါဝင်လာပါတယ်။ Date ကို Unix Timestamps သိမ်းဖို့သုံးပြီး Byte Array ကိုတော့ Binary Data တွေ သိမ်းဖို့သုံးပါတယ်။

လက်ရှိရွေးချယ်ထားတဲ့ Database ကို ပယ်ဖျက်လိုတဲ့အခါ `db.dropDatabase()` ကို အသုံးပြုနိုင်ပါတယ်။

```
> db.dropDatabase()
{
  "dropped" : "mydb",
  "ok" : 1
}
>
```

## 11.4 – CRUD Operations in MongoDB

RDBMS တွေမှာ Record တွေကို Table တွေနဲ့စုစုပေါင်းထားနိုင်သလိုပဲ MongoDB မှာလည်း Document တွေကို Collection တွေနဲ့ စုစုပေါင်းထားနိုင်ပါတယ်။ `db.createCollection()` ကို Collection အသစ်တစ်ခု တည်ဆောက်ဖို့အတွက် သုံးနိုင်ပါတယ်။

```
> use mydb
switched to db mydb
> db.createCollection("issues")
{
  "ok" : 1
}
> show collections
issues
system.indexes
>
```

နဲ့မူနာအရ mydb Database တဲ့မှာ issues အမည်နဲ့ Collection တစ်ခုကို တည်ဆောက်ပေးသွားမှာဖြစ်ပါတယ်။ Collection တစ်ခုကို ကြိုးတင်မထည့်ဆောက်ပဲ Document ထည့်သွင်းစဉ်မှာ တစ်ခါတည်းသတ်မှတ်နိုင်ပါသေးတယ်။

```
> db.users.insert({name: "John Doe"})
> show collections

issues
system.indexes
users

>
```

users အမည်နဲ့ Collection တစ်ခုကို ကြိုးတင်တည်ဆောက်ထားခြင်းမရှိပေမယ့် db.users.insert() နဲ့ users Collection တဲ့မှာ Document တစ်ခု တည်ဆောက်ပေးဖို့ ပြောလိုက်ခြင်းဖြစ်ပါတယ်။ MongoDB က users Collection ကို အလိုအလျောက် တည်ဆောက်ပေးသွားတဲ့အတွက် show collections နဲ့ကြည့်လိုက်တဲ့အခါ Collection စာရင်းတဲ့မှာ users လည်း ပါဝင်နေခြင်းပဲဖြစ်ပါတယ်။

Collection တစ်ခုကို ပယ်ဖျက်ဖို့အတွက် drop() Function ကိုသုံးနိုင်ပါတယ်။ ဥပမာ -

```
> db.users.drop()

true

> show collections

issues
system.indexes

>
```

Collection တဲ့မှာ Document တွေထည့်သွင်းဖို့အတွက် insert() Function ကို အသုံးပြုနိုင်ကြောင်း ပြီးခဲ့တဲ့ နဲ့မူနာမှာ တွေ့ရှိခဲ့ရပြီးဖြစ်ပါတယ်။ နောက်ထပ် insert() နဲ့မူနာတစ်ခု ထပ်မံဖော်ပြပါမယ်။

```
> db.issues.insert({
...   subject: "Dropdown menu not working",
...   priority: {
...     value: 2,
...     label: "Important"
...   },
...   severity: {
...     value: 1,
...     label: "Minor"
...   },
...   status: "Open"
... })
```

```

...     assignedTo: {
...       name: "John Doe",
...       userId: 1
...     },
...     status: {
...       value: 1,
...       label: "Assigned"
...     }
...   }
>

```

issues Collection ထဲကို Document တစ်ခုထည့်သွင်းလိုက်ခြင်းဖြစ်ပါတယ်။ အဲဒီလိုထည့်သွင်းရာမှာ လိုအပ်တဲ့ ဆက်စပ် အချက်အလက် အားလုံးကို ထည့်သွင်းထားတာကို သတိပြုပါ။ ဥပမာ - priority အတွက် value: 2 နဲ့ label: "Import" လို့ တစ်ခါတည်း အပြည့်အစုံ ထည့်သွင်းထားပါတယ်။ RDBMS တွေမှာ ဆိုရင် priority: 2 လို့သာထည့်သွင်းပြီး 2 ရဲ့ Label ကိုသိချင်ရင် Priority Table နဲ့ JOIN လုပ်ယူရမှာဖြစ်ပါတယ်။ ဒီနေရာမှာတော့ အဲဒီလို Relationship ကို ရောင်ချင်တဲ့အတွက် Priority တန်ဖိုး 2 ရော့ Label ဖြစ်တဲ့ Important ကိုပါ တစ်ခါတည်း တွဲဖောက်ထည့်သွင်းထားခြင်းဖြစ်ပါတယ်။ တစ်ပြိုင်တည်း Document နှစ်ခုသုံးရှိထည့်သွင်းလိုရင် ထည့်သွင်းလိုတဲ့ Array အနေနဲ့ ထည့်သွင်းနိုင်ပါတယ်။ ဥပမာ -

```

> db.users.insert([
...   { name: "John Doe", role: "Admin" },
...   { name: "James Smith", role: "Tester" }
... ])
>

```

ထည့်သွင်းထားတဲ့ Document တွေကို ပြန်လည်ကြည့်ရှုလိုရင်တော့ find() ကိုအသုံးပြုနိုင်ပါတယ်။

```

> db.users.find()

{ "_id" : ObjectId("55350aea459cc324c59cf1aa"), "name" : "John Doe", "role" :
"Admin" }
{ "_id" : ObjectId("55350aea459cc324c59cf1ab"), "name" : "James Smith", "role" :
"Tester" }

>

```

ရလာတဲ့ရလဒ်ကိုသတိထားကြည့်ရင် Document တွေအတွက် \_id ဆိုတဲ့ Attribute တစ်ခုပါဝင်နေတာကို တွေ့ရမှာ ဖြစ်ပါတယ်။ Document တိုင်းမှာ \_id Attribute ရှိရပါတယ်။ ကျွန်ုတော်တို့ Document တွေထည့်သွင်းစဉ်၊ \_id ထည့်သွင်းမပေးလိုက်တဲ့အတွက် MongoDB က အလိုအလျောက် Auto Generate လုပ်ပြီး ထည့်သွင်းပေးသွားခြင်းဖြစ်ပါတယ်။ Generate လုပ်လိုက်တဲ့ \_id ဟာ Unique ဖြစ်နေစေဖို့အတွက် MongoDB က အခုလို Formate ကို အသုံးပြုပါတယ်။

- 4 bytes Timestamps
- 3 bytes Machine ID
- 2 bytes Process ID
- 3 bytes Auto-Increment Counter

`_id` ဟာ ဒီတန်ဖိုး (၄) မီးကိုပေါင်းစပ်ထားတဲ့ 12 bytes Hexadecimal String တစ်ခုဖြစ်ပါတယ်။ Document တွေကိုစိမ့်တဲ့ အခါ ဒါ `_id` ကို Primary Key အနေနဲ့ အသုံးပြုစီမံနိုင်စေမှာဖြစ်ပါတယ်။ MongoDB က Generate လုပ်ပေးတဲ့ ID ကို မသုံးပဲ ကိုယ်တိုင် ID တွေသတ်မှတ်အသုံးပြုလိုတယ်ဆိုရင်လည်း Document ထည့်သွင်းစဉ်မှာ `_id` Attribute တွဲဖောက်ထည့်သွင်းပေးလိုက်ယုံပြစ်ပါတယ်။ ပေးလိုက်တဲ့တန်ဖိုးဟာ Unique ဖြစ်ဖို့တော့လိုပါတယ်။

ဤဗုံးခဲ့တဲ့ `find()` ရလဒ်ကို ပြန်လည်လေ့လာကြည့်ရင် `_id` တန်ဖိုးကို `ObjectId()` Function နဲ့ တွဲဖက်ပေးထား တာကို သတိပြုမိနိုင်ပါတယ်။ `ObjectId()` Function နဲ့ တွဲဖက်ထားတဲ့အတွက် ID ကနေ Timestamps ကို ပြန်လည်ထုတ်ယူနိုင်တဲ့ အားသာချက်နဲ့ Sorting စီတဲ့အခါ Timestamps ကိုအခြေခံစီပေးသွားတဲ့အတွက် ပိုမိုမှန်ကန်တဲ့ အားသာချက်တို့ကို ရရှိနိုင်ပါတယ်။ ဥပမာ -

```
> db.users.find()
{
  "_id" : ObjectId("55350aea459cc324c59cf1aa"), "name" : "John Doe", "role" : "Admin"
}
{
  "_id" : ObjectId("55350aea459cc324c59cf1ab"), "name" : "James Smith", "role" : "Tester"
}

> ObjectId("55350aea459cc324c59cf1aa").getTimestamp()
ISODate("2015-04-20T14:19:22Z")

> ObjectId("55350aea459cc324c59cf1aa").str
55350aea459cc324c59cf1aa
>
```

`ObjectId()` Function ကနေတစ်ဆင့် `getTimestamp()` ကိုသုံးပြီး ID ထဲကအချိန်ကို ပြန်လည်ရယူနိုင် ခြင်း ဖြစ်သလို၊ `str` Attribute ကိုသုံးပြီး ID ကို String တန်ဖိုးအဖြစ် ပြောင်းယူနိုင်မှာဖြစ်ပါတယ်။

Collection ထဲက Document တစ်ခုကို မြှုပြင်လိုရင် `save()` ကို အသုံးပြုရပါတယ်။ ဥပမာ -

```
> db.users.find()
{ "_id" : ObjectId("55350aea459cc324c59cf1aa"), "name" : "John Doe", "role" :
"Admin" }
{ "_id" : ObjectId("55350aea459cc324c59cf1ab"), "name" : "James Smith", "role" :
"Tester" }

> db.users.save({
... _id : ObjectId("55350aea459cc324c59cf1aa"),
... name: "John Doe",
... role: "Developer"})

> db.users.find()
{ "_id" : ObjectId("55350aea459cc324c59cf1aa"), "name" : "John Doe", "role" :
"Developer" }
{ "_id" : ObjectId("55350aea459cc324c59cf1ab"), "name" : "James Smith", "role" :
"Tester" }

>
```

နမူနာမှာ User John Doe ရဲ့ `role` ကို Admin တနောက် Developer လို `save()` Function သုံးပြီးပြောင်းထားပါတယ်။ `save()` Function က `_id` ကိုတွက်ပြီး မူလကအဲဒီ ID နဲ့ရှိနေတဲ့ Document ကို Update လုပ်ပေးသွားမှာ ဖြစ်ပါတယ်။

`find()` Function ကပြန်ပေးတဲ့ရလဒ်ကို Format လုပ်ပြီး သပ်သပ်ရပ်ရပ်ဖော်ပြစ်လိုရင် `pretty()` နဲ့ တဲ့ သုံးရပါတယ်။

```
> db.users.find().pretty()
{
  "_id" : ObjectId("55350aea459cc324c59cf1aa"),
  "name" : "John Doe",
  "role" : "Developer"
}
{
  "_id" : ObjectId("55350aea459cc324c59cf1ab"),
  "name" : "James Smith",
  "role" : "Tester"
}
>
```

`find()` Function နဲ့အတူ Query တွေလည်းတွဲဖက် ပေးနိုင်ပါတယ်။ ဥပမာ - `users` Collection ထဲက `role: "Tester"` တွေကို ရရှိလိုတဲ့အခါ အခုလိုအသုံးပြုနိုင်ပါတယ်။

```
> db.users.find({ role: "Tester" })
```

တစ်ချို့တန်ဖိုးတွေက Nested Object အနေနဲ့ Document ထဲမှာသိမ်းဆည်းထားတာပါ။ ကျွန်တော်တို့ insert() နမူနာအနေနဲ့ ထည့်သွင်းခဲ့တဲ့ Document မှာဆိုရင် priority, status စတဲ့ တန်ဖိုးတွေဟာ ရှိုးရှိုးတန်ဖိုးမဟုတ်ပဲ Object တွေဖြစ်ပါတယ်။ ဒီလို Nested Object ထဲက တန်ဖိုးတစ်ခုခုနဲ့လည်း Query လုပ် နိုင်ပါတယ်။ ဥပမာ – Status ရဲ့ Nested Object မှာ Value 1 တန်ဖိုးရှိုးတဲ့ Document တွေကို အချင်း လုပ်နိုင်ပါတယ်။

```
> db.issues.find({ "status.value": 1 })
```

AND ပုံစံနဲ့ တန်ဖိုးတစ်ခုထက်ပိုသုံးပြီးတော့လည်း Query လုပ်နိုင်ပါတယ်။ ဥပမာ –

```
> db.issues.find({ "status.value": 1, "severity.value": 3 })
```

OR ပုံစံ Query လုပ်လိုရင်တော့ အချင်း အသုံးပြုနိုင်ပါတယ်။

```
> db.issues.find({
... $or: [
...   { "status.value": 1 },
...   { "severity.value": 3 }
... ]
... });
>
```

Greater Than, Less Than စတဲ့ Comparison တွေပြုလုပ်လိုရင် အချင်းပြုလုပ်နိုင်ပါတယ်။

```
> db.issue.find({
...   "status.value": { $gt: 1 }
... })
>
```

\$gt က Greater Than ဆိုတဲ့အဓိပ္ပာယ်ဖြစ်ပြီး နမူနာအရ status.value တန်ဖိုး 1 ထက်ကြီးတဲ့ Document တွေကို ရယူပေးမှာ ဖြစ်ပါတယ်။ \$gt အစား: \$lt, \$gte, \$lte, \$ne စို့တို့ အစားထိုးအသုံးပြုနိုင်ပါတယ်။ \$gte က Greater Than or Equal ဖြစ်ပြီး \$ne ကတော့ Not Equal ဖြစ်ပါတယ်။

RDBMS တွေရဲ့ LIKE Statement လိုသဘောမျိုး Filter လိုချင်ရင်တော့ Regular Expression ကိုအသုံးပြုရပါတယ်။

```
> db.issues.find({ subject: /menu/ })
```

နဲ့မှန်အရ subject Attribute ထဲမှာ menu ဆိုတဲ့စာလုံးပါဝင်တဲ့ Document အားလုံးကို ပြန်ပေးသွားမှာဖြစ်ပါတယ်။ Regular Expression အကြောင်းအသေးစိတ်ကိုတော့ ဒီနေရာမှာ မဖော်ပြနိုင်ပါတယ်။ ပိုပြီး အသေးစိတ်တဲ့ Filter တွေ ဆောင်ရွက်နိုင်ဖို့ အတွက် Regular Express ကို လေ့လာပြီး find() Function နဲ့ အတူ တွဲဖက်အသုံးချလို့ရပါတယ်။

RDBMS တွေရဲ့ SELECT Query မှာ Field List ကို သတ်မှတ်နိုင်သလိုပဲ find() Function နဲ့ Query လုပ်တဲ့အခါ Document တစ်ခုလုံးမဟုတ်ပဲ၊ လိုအပ်တဲ့ Attribute တွေကိုသာ ရွေးချယ်ရယူနိုင်ပါတယ်။ ဥပမာ -

```
> db.users.find({}, { name: 1 })
{ "_id" : ObjectId("5537b3036831dbd2140f548e"), "name" : "John Doe" }
{ "_id" : ObjectId("5537ba0b6831dbd2140f548f"), "name" : "James Smith" }
>
```

find() Function ရဲ့ ဒုတိယ Parameter အနေနဲ့ Attribute List ကိုပေးရခြင်းဖြစ်ပါတယ်။ ရွေးချယ်လိုတဲ့ Attribute တန်ဖိုးကို 1 လို သတ်မှတ်ပေးရခြင်းဖြစ်ပါတယ်။ ထူးခြားချက်အနေနဲ့ `_id` က Field List ထဲမှာ အမြပါဝင် နေမှာဖြစ်ပါတယ်။

find() Function ကပြန်ပေးတဲ့ ရလဒ်မှာပါဝင်တဲ့ Document အရေအတွက်ကို ကန်သတ်လိုရင် limit() Function နဲ့ တွဲသုံးနိုင်ပါတယ်။ ဥပမာ -

```
> db.users.find().limit(2)
```

limit(2) လိုပြောထားတဲ့အတွက် find() Function ကပြန်ပေးလာတဲ့ Document တွေထဲက ပထမဆုံး (၂) ခုကိုပဲ ဖော်ပြပေးသွားမှာဖြစ်ပါတယ်။ RDBMS တွေမှာ Offset အနေနဲ့ စတင်ဖော်ပြရမယ့် Index ကိုသတ်မှတ်နိုင်သလိုပဲ MongoDB မှာလည်း သတ်မှတ်နိုင်ပါတယ်။ skip() Function ကို သုံးရပါတယ်။ ဥပမာ -

```
> db.users.find().skip(5).limit(10)
```

နဲ့မှန်အရ users Collection ထဲက Document (၁၀) ခုကိုရယူပေးမှာဖြစ်ပါတယ်။ အဲဒီလိုရယူရာမှာ ပထမဆုံး (၅) ခုကိုကျော်ပြီး (၆) ခုကြောက်ကနေ စတင်ရယူပေးသွားမှာဖြစ်ပါတယ်။ find() Function ကပြန်ပေးတဲ့ရလဒ်ကို Sorting စီပေးစေလိုရင် sort() Function နဲ့ တွဲသုံးနိုင်ပါတယ်။ ဥပမာ -

```
> db.issues.find().sort({ priority: 1, status: 1 })
```

နဲ့မှုနာအရ issues Collection တဲက Document တွေကို priority နဲ့ ငယ်စဉ်ကြီးလိုက်စီပေးသွားမှာဖြစ်ပါတယ်။ priority တူနေတဲ့ Document တွေကိုတော့ status နဲ့စီပေးသွားမှာဖြစ်ပါတယ်။ ကြီးစဉ်ငယ်လိုက်ပြောင်းပြန်စီ စေလိုရင်တော့ 1 အစား -1 ကို သုံးနိုင်ပါတယ်။

Collection တဲက Document တစ်ခုကို ပယ်ဖျက်လိုရင်တော့ remove() Function ကိုသုံးနိုင်ပါတယ်။ ပယ်ဖျက်လိုတဲ့ Document အတွက် Query ထည့်သွေးပေးဖို့လိုပါတယ်။ ဥပမာ - users Collection တဲက role: "Tester" သတ်မှတ်ထားတဲ့ Document အားလုံးကို ပယ်ဖျက်လိုရင် အခုလိုပယ်ဖျက်နိုင်ပါတယ်။

```
> db.users.remove({ role: "Tester" })
```

Query နဲ့ကိုက်ညီတဲ့ Document တွေထဲက ပထမဆုံးတစ်ခုကိုသာ ပယ်ဖျက်လိုရင် အခုလိုပယ်ဖျက်နိုင်ပါတယ်။

```
> db.users.remove({ role: "Tester" }, 1);
```

remove() ကို Query မပေးပဲ Run ရင်တော့ ရှိသမှု Document အားလုံးကို ပယ်ဖျက်သွားမှာဖြစ်ပါတယ်။

save() Function ကိုသုံးပြီး လက်ရှိ Document ကို Overwrite လုပ်နိုင်ပုံကို အထက်မှာဖော်ပြခဲ့ပါတယ်။ အဲဒီလို Overwrite မလုပ်ပဲ တန်ဖိုးတစ်ချို့ကိုသာ ပြင်လိုရင်တော့ update() ကိုသုံးရပါတယ်။ ဥပမာ -

```
> db.users.update(
...   { role: "Developer" },
...   { $set: { role: "Programmer" } }
...   { multi: true }
... )
```

နဲ့မှုနာအရ role: "Developer" သတ်မှတ်ထားတဲ့ Document အားလုံးကို role: "Programmer" တန်ဖိုး နဲ့ Update လုပ်ပေးသွားမှာဖြစ်ပါတယ်။ update() Function အတွက် Parameter (၃) ခုပေးရပါတယ်။ ပထမ တစ်ခုက Query ပါ။ ဒုတိယတစ်ခုက အစားထိုးလိုတဲ့ Value Set ဖြစ်ပြီး၊ နောက်ဆုံးတစ်ခုကတော့ Option ဖြစ်ပါတယ်။ အကယ်၍ Query နဲ့ ကိုက်ညီသမှု Document အားလုံးကို Update မလုပ်စေလိုပဲ၊ ပထမဆုံးတစ်ခုကိုသာ Update ပြုလုပ်စေလိုရင်တော့ နောက်ဆုံးက multi: true Option ကို false ပြောင်းထားနိုင်ပါတယ်။

RDBMS စနစ်တွေမှာ SQL Query တွေကိုသုံးပြီး Create, Read, Update, Delete (CRUD) လုပ်ငန်းတွေကို

ဆောင်ရွက်နိုင်သလိုပဲ MongoDB မှာလည်း `insert()`, `find()`, `update()`, `remove()` Function များကို အသုံးပြုပြီး CRUD လုပ်ငန်းများကို ဆောင်ရွက်နိုင်ခြင်းပဲ ဖြစ်ပါတယ်။

## 11.5 – Indexing in MongoDB

RDBMS စနစ်များမှာ Read Performance ကောင်းမွန်ဖို့အတွက် Index တွေသတ်မှတ်ထားနိုင်သလိုပဲ၊ MongoDB မှာလည်း Index တွေသတ်မှတ်ထားနိုင်ပါတယ်။ `createIndex()` Function ကိုသုံးရပါတယ်။

```
> db.issues.createIndex({ status: 1 })
```

issues Collection ထဲမှာ Index တည်ဆောက်လိုက်ခြင်းဖြစ်ပြီး Index ပြုလုပ်လိုတဲ့ Attribute အဖြစ် status ကို ရွေးချယ်လိုက်ခြင်းဖြစ်ပါတယ်။ ဒါကြောင့် status နဲ့ Query လုပ်တဲ့အခါ Document တစ်ခုစီ လိုက်ရှာနေဖို့မလိုတော့ ပဲ Index ကို ကြည့်ပြီး ရလဒ်ပြန်ပေးနိုင်မှာဖြစ်လို့ Performance ပိုကောင်းသွားမှာပဲဖြစ်ပါတယ်။ Index သတ်မှတ်ရာမှာ Option တစ်ချို့တဲ့ပေးနိုင်ပါတယ်။ ဥပမာ –

```
> db.issues.createIndex({ _id: 1 }, {
  ... name: "PrimaryKey",
  ... unique: true,
  ... sparse: true
})
```

name Option ကိုအသုံးပြုပြီး Index Name သတ်မှတ်နိုင်ပါတယ်။ အကယ်၍ Index Name မသတ်မှတ်ပေးခဲ့ရင်လည်း MongoDB က Index Name တစ်ခု Generate လုပ်ပြီး အလိုအလျောက်သတ်မှတ်ပေးသွားမှာ ဖြစ်ပါတယ်။ unique Option ကို `true` လို့သတ်မှတ်ထားရင်တော့ ရှိုးရှိုး Index မဟုတ်တော့ပဲ Unique Index ဖြစ်သွားမှာ ဖြစ်ပါတယ်။ Index လုပ်လိုတဲ့ Attribute မပါဝင်တဲ့ Document တွေလည်း Collection ထဲမှာ ရှိနေနိုင်ပါတယ်။ အဲဒီလို Document တွေကို Index ထည့်မလုပ်စေလိုရင် `sparse` Option ကို `true` လို့သတ်မှတ်ပေးရပါတယ်။

Index တွေကိုဖြန့်လည်ပယ်ဖျက်စေလိုတဲ့အခါ `dropIndex()` Function နဲ့ပယ်ဖျက်နိုင်ပါတယ်။ ပယ်ဖျက်လိုတဲ့ Index Name (သို့မဟုတ်) Index လုပ်ထားတဲ့ Attribute ကို ပေးရပါတယ်။ ဥပမာ –

```
> db.issues.dropIndex("PrimaryKey")
> db.issues.dropIndex({ status: 1 })
```

ရှိုးသမျှ Index အားလုံးကို ပယ်ဖျက်စေလိုရင်တော့ `dropIndexes()` ကို သုံးနိုင်ပါတယ်။ တည်ဆောက်ထားတဲ့ Index တွေရဲ့စာရင်းကိုတော့ `getIndexes()` နဲ့ ရယူနိုင်ပါတယ်။

```
> db.issues.getIndexes()
```

MySQL MyISAM Storage Engine မှာ Text Search အတွက် Full-text Index လုပ်ဆောင်ချက်ပါဝင်သလိုပဲ MongoDB မှာလည်း Text Index လုပ်ဆောင်ချက်ပါဝင်ပါတယ်။ Text Index ၏ Index ပြုလုပ်ယုံသာမက၊ Search Engine တစ်ခုရဲ့ အခြေခံလုပ်ဆောင်ချက်တွေဖြစ်တဲ့ Noise တွေ၊ Stop Words တွေဖော်ထုတ်ပေးခြင်းနဲ့ Stemming လို့ လုပ်ဆောင်ချက်မျိုး ၏တွေကို တစ်ခါတည်း ဆောင်ရွက်ပေးနိုင်ပါတယ်။ Stemming ဆုံးတာ၊ ဥပမာဏြောရရင် wait လိုရှာရင် waiting, waited, waits စတဲ့စာလုံးတွေကိုပါ အလိုအလျောက် ထည့်သွင်းရှာဖွေပေးတဲ့လုပ်ဆောင်ချက်မျိုး ဖြစ်ပါတယ်။ Text Index တစ်ခုကို အခုလို သတ်မှတ်နိုင်ပါတယ်။

```
> db.issues.createIndex({ subject: "text" })
```

ရုံးရိုး Index အတွက် 1 ကိုအသုံးပြု ရဖြီး Text Index အတွက် text ကို အသုံးပြုရခြင်းဖြစ်ပါတယ်။ အကယ်၍ Text Index သတ်မှတ်ရမှာ { "ok": 0, "errmsg": "text search not enabled" } ဆုံးတဲ့ Error Message ရခဲ့ရင် Text Index လုပ်ဆောင်ချက်ကို အခုလို Enable လုပ်ပေးဖို့လိုပါတယ်။

```
> db.adminCommand({ setParameter: "*", textSearchEnabled: true })
{ "was" : false, "ok" : 1 }
```

Text Search ကိုလည်း Enable လုပ်ပြီးပြီး Text Index လည်းလုပ်ပြီးပြီးဆုံးရင် Text Index ထဲကနေ ရှာဖွေခြင်းလုပ်ငန်းကို runCommand() နဲ့ ဆောင်ရွက်နိုင်မှာ ဖြစ်ပါတယ်။ ဥပမာ -

```
> db.issues.runCommand("text", { search: "menu" })
{
  "queryDebugString" : "menu|||||",
  "language" : "english",
  "results" : [
    {
      "score" : 0.6666666666666666,
      "obj" : {
        "_id" : ObjectId("553508f5459cc324c59cf1a9"),
        "subject" : "Dropdown menu not working",
        "priority" : {
          "value" : 2,
          "label" : "Important"
        },
        "assignedTo" : {
          "name" : "John Doe",
          "id" : 1
        },
        "status" : {
          "value" : 1,
          "label" : "Assigned"
        }
      }
    }
  ]
}
```

```

        }
    }
],
"stats" : {
    "nscanned" : 1,
    "nscannedObjects" : 0,
    "n" : 1,
    "nfound" : 1,
    "timeMicros" : 197
},
"ok" : 1
}
>

```

runCommand() Function ကိုသုံးပြီး menu ဆိုတဲ့ Keyword ကို Text Index ထဲမှာ ရှာကြည့်လိုက်တဲ့အခါ menu Keyword ပါဝင်တဲ့ Document တွေကို ဖော်ပြပေးလာမှာဖြစ်ပါတယ်။ result ထဲက score ကို သတိပြု ကြည့်သင့်ပါတယ်။ score မှာဖော်ပြနေတဲ့တန်ဖိုးဟာ Relevant Value ဖြစ်ပြီး ရှာဖွေလိုတဲ့ Keyword နဲ့ လက်ရှိ Document ဘယ်လောက်ထိ Relevant ဖြစ်သလဲဆိုတာကို ဖော်ပြနေခြင်းဖြစ်ပါတယ်။ ဒါ score ကို သုံးပြီး Sorting စီလိုက်မယ်ဆိုရင် Relevant အဖြစ်ဆုံးရလဒ်ကို အရင်ဖော်ပြပေးနိုင်တဲ့ Search Result ကို ရရှိ မှာပဲဖြစ်ပါတယ်။

ဒီနည်းနဲ့ RDBMS တွေမှာ ရရှိလေ့ရှိတဲ့ Index, Unique Index နဲ့ Full Text Index တို့ကို MongoDB မှာလည်း ရရှိ နိုင်မှာဖြစ်ပါတယ်။

## 11.6 – Access Control in MongoDB

ကျွန်ုတ်တို့ လက်ရှိဆောင်ရွက်နေတဲ့ ကိစ္စတွေအားလုံးအတွက် MongoDB က ဘာ Authentication Credential မှ မတောင်းတာကို သတိပြုမိပါလိမ့်မယ်။ Default အနေနဲ့ MongoDB မှာ Access Control ကို ကန့်သတ်မထားပါဘူး။ Access Control သတ်မှတ်လိုတယ်ဆိုရင် ပထမဦးဆုံးအနေနဲ့ admin User တစ်ဦးကို စတင်သတ်မှတ်သင့်ပါတယ်။ အခုလို သတ်မှတ်နိုင်ပါတယ်။

```

> use admin
switched to db admin
> db.addUser("root", "root")
{
    "user" : "root",
    "readOnly" : false,
    "pwd" : "2a8025f0885adad5a8ce0044070032b3",
    "_id" : ObjectId("5536899f1d331f653a28893e")
}
>

```

ပထမြီးဆုံး admin Database ကိုရွေးထားပေးပါတယ်။ ပြီးတော့မှ db.addUser() နဲ့ Admin User Account တစ်ခုကို တည်ဆောက်နိုင်ခြင်းဖြစ်ပါတယ်။ နမူနာအရ Username: root နဲ့ Password: root လို့ သတ်မှတ်ထားပါတယ်။

သတ်မှတ်ထားတဲ့ Access Control စတင်အလုပ်လုပ်စေဖို့အတွက် /etc/mongodb.conf ဖိုင်ထဲမှာ auth=true ဆိုတဲ့ Setting တစ်ခု သတ်မှတ်ပေးပါတယ်။ ပြီးတဲ့အခါ MongoDB ကို Restart လုပ်ပေးပါတယ်။

```
$ sudo service mongodb restart
```

အခုန် Mongo Shell ကိုပြန်ဝင်ပြီး စမ်းကြည့်ရင် Access Control Error ကိုတွေ့ရမှာဖြစ်ပါတယ်။

```
> use admin
switched to db admin
> show collections
Wed Apr 22 00:11:07.267 error: {
  "$err" : "not authorized for query on admin.system.namespaces",
  "code" : 16550
} at src/mongo/shell/query.js:128
>
```

သတ်မှတ်ထားတဲ့ Username, Password မှန်အောင်ပေးပြီးမှသာ အချက်အလက်တွေကို စတင်စီမံနိုင်တော့မှာဖြစ်ပါတယ်။ db.auth() Function ကိုသုံးပြီး Authenticate လုပ်နိုင်ပါတယ်။

```
> db.auth("root", "root")
1
> show collections
system.indexes
system.users
```

Admin User မဟုတ်တဲ့ ရှိုးရိုး User Account ကိုတော့ သက်ဆိုင်ရာ Database ပေါ်မှာ အခုလိုတည်ဆောက်ပေးနိုင်ပါတယ်။

```

> use mydb
switched to db mydb

> db.addUser("user", "user")

{
  "user" : "user",
  "readOnly" : false,
  "pwd" : "fa26a506aa0f786a447bbd6d1caaa8b5",
  "_id" : ObjectId("55368c6c3e0d2164a31a2ec8")
}
>

```

Admin User နဲ့ Database အားလုံးကိုစီမံနိုင်ပေးမယ့် ရုံးရိုး User ကတော့ သူအတွက်သတ်မှတ်ပေးထားတဲ့ Database ကိုသာ စီမံခွင့်ရမှာဖြစ်ပါတယ်။

```

> use admin
switched to db admin

> db.auth("user", "user")

Error: 18 { code: 18, ok: 0.0, errmsg: "auth fails" }
0

>

```

mydb Database အတွက် တည်ဆောက်ထားတဲ့ User နဲ့ admin Database ကို Authenticate လုပ်ဖို့ကြိုးစားတဲ့ အခါ auth fails Error ကို ရရှိခြင်းဖြစ်ပါတယ်။ Data တွေကို Access လုပ်လိုရပေးမယ့် အသစ်တည်ဆောက်ခွင့် မရှိတဲ့ Readonly User Account တွေလည်းတည်ဆောက်နိုင်ပါတယ်။ addUser() Function ရဲ့ တတိယ Parameter ကို true လိုသတ်မှတ်ပေး ရပါတယ်။

```

> db.addUser("user", "user", true)

{
  "user" : "user",
  "readOnly" : 1,
  "pwd" : "fa26a506aa0f786a447bbd6d1caaa8b5",
  "_id" : ObjectId("55368d46d3bc6702df267302")
}
>

```

Readonly User ဟာ အချက်အလက်တွေကို ရယူနိုင်ပေးမယ့် အသစ်တည်ဆောက်ခွင့် ရမှာမဟုတ်ပါဘူး။

```

> use mydb
switched to db mydb

> db.auth("user", "user")
> show collections

issues
system.indexes
system.users
users

> db.createCollection('comments')

{ "ok" : 0, "errmsg" : "unauthorized" }

>

```

အချေဖော်ပြခဲ့တဲ့ Access Control လုပ်ဆောင်ချက်ဟာ MongoDB Version 2.4 အတွက်ဖြစ်ပါတယ်။ MongoDB Version 2.6 ကနေနောက်ပိုင်းမှာ Access Control သတ်မှတ်ပုံ ပြောင်းလဲသွားပါတယ်။ ဥပမာ - addUser() Function ကိုမသုံးတော့ပဲ User Account တွေရော့၊ Role တွေကိုပါ တစ်ခါတည်း သတ်မှတ်နိုင်တဲ့ createUser() Function ကို ပြောင်းသုံးသွားပါတယ်။ ပြောင်းလဲသွားတဲ့ Access Control သတ်မှတ်ပုံကို အောက်ပါလိုပါတယ်။

<http://docs.mongodb.org/manual/tutorial/add-user-to-database/>

## 11.7 – MongoDB Backup and Restore

MongoDB Database ထဲက အချက်အလက်တွေကို mongodump Tool အသုံးပြုပြီး Backup ပြုလုပ်နိုင်ပါတယ်။ ဥပမာ -

```
$ mongodump -d mydb -o /path/to/output
```

-d Option နဲ့ Backup လုပ်လိုတဲ့ Database ကိုပေးရပြီး -o Option နဲ့ Backup ဖိုင်တွေသိမ်းပေးရမယ့် Directory ကို သတ်မှတ်ပေးရပါတယ်။ -d Option နဲ့ Database မသတ်မှတ်ခဲ့ရင် ရှိသမျှ Database အားလုံးကို Backup လုပ်ပေးသွားမှာ ဖြစ်ပါတယ်။ -o Option နဲ့ Output Directory သတ်မှတ်မပေးရင်တော့ လက်ရှိ Command ကို Run ထဲမှာ Backup ဖိုင်တွေကို သိမ်းပေးသွားမှာဖြစ်ပါတယ်။

Database မှာ Access Control သတ်မှတ်ထားရင် Username, Password တစ်ခါတည်းပေးရပါတယ်။ မပေးရင် auth fails Error Message ကိုပေးပါလိမ့်မယ်။ Username ကို -u Option နဲ့ပေးရပြီး Password ကို -p Option နဲ့ပေးရပါတယ်။ ဥပမာ -

```
$ mongodump -u username -p password -d mydb -o /path/to/output
```

အကယ်၍ Collection တစ်ခုတည်းကိုရွေးပြီး Backup ပြုလုပ်လိုရင်တော့ -c Option ကို သုံးနိုင်ပါတယ်။ ဥပမာ -

```
$ mongodump -d mydb -c issues -o /path/to/output
```

နဲ့မှနာအရ mydb Database ထဲက issues Collection တစ်ခုတည်းကို Backup ထုတ်ယူလိုက်ခြင်းပဲဖြစ်ပါတယ်။

mongodump နဲ့ Backup လုပ်ထားတဲ့ ဖိုင်တွေကို ပြန်ပြီး Restore လုပ်လိုရင်တော့ mongorestore ကို အသုံးပြုရ ပါတယ်။ Backup ဖိုင်တွေရှိနေတဲ့ Directory ထဲမှာ Run ပေးသင့်ပါတယ်။ အကယ်၍ တစ်ခြား Director ထဲမှာ Run မယ်ဆိုရင် Backup ဖိုင်တွေတည်ရှိရောနရာကို တွဲပြီးညွှန်းပေးရပြီး Access Control တွေ ရှိနေရင်လည်း တွဲဖက်ပေးဖို့လို ပါတယ်။

```
$ mongorestore /path/to/backup -u username -p password
```

Restore လုပ်တဲ့အခါ မူလရှိနေတဲ့ Collection တွေကို Drop အရင်လုပ်ပြီးမှ Restore လုပ်စေလိုတဲ့အခါ --drop Option တွဲဖက်သက်မှတ်ပေးရပြီး Collection တွေအတွက် သုံးရမယ့် Database ကို -d Option နဲ့ တွဲဖက်ပေးရပါ တယ်။ ဥပမာ -

```
$ mongorestore /path/to/backup -d mydb --drop
```

နဲ့မှနာအရ Backup ဖိုင်ထဲက Collection တွေကို mydb Database ထဲမှာ Restore လုပ်ပေးသွားမှာဖြစ်ပြီး mydb Database ထဲမှာ မူလကအမည်တဲ့ Collection ရှိနေရင် အဲဒီ Collection ကိုအရင် Drop လုပ်ပြီးမှ Restore လုပ်ပေး သွားမှာဖြစ်ပါတယ်။

Backup လုပ်ဖို့အတွက် mongoexport ကိုလည်း သုံးနိုင်ပါသေးတယ်။ mongodump က အချက်အလက်တွေ ကို Binary Data အနေနဲ့ Backup လုပ်ပေးပြီး mongoexport ကတော့ JSON (သို့မဟုတ်) CSV အနေနဲ့ Backup လုပ်ပေးပါတယ်။

```
$ mongoexport -d mydb -c issues -o export.json
```

နဲ့မှနာအရ mydb Database ထဲက issues Collection ကို JSON Format နဲ့ Export လုပ်ပေးသွားမှာဖြစ်ပါ

တယ်။ –o Option နဲ့ ရရှိလာတဲ့ ရလဒ်ကို export.json ဖိုင်ထဲမှာ ထည့်သွင်းပေးဖို့ သတ်မှတ်ထားပါတယ်။ –o Option သတ်မှတ်မထားရင် ဖိုင်အနေနဲ့မသိမ်းပဲ Terminal မှာ Output အနေနဲ့ ဖော်ပြလာမှာပါ။ CSV Format နဲ့ လိုချင်ရင် တော့ --csv Option နဲ့တဲ့ပြီး Export ထုတ်ယူရပါတယ်။

```
$ mongoexport -d mydb -c issues -f "subject,status" --csv -o export.csv
```

ထူးခြားချက်အနေနဲ့ Export ထုတ်ယူလိုတဲ့ Field List ကို -f Option နဲ့ တဲ့ပေးရပါတယ်။ ထံးစံအတိုင်း Access Control တွေသတ်မှတ်ထားရင်လည်း သက်ဆိုင်ရာ Username, Password ကို -u, -p Option တွေနဲ့ ထည့်သွင်းပေးဖို့လိုပါတယ်။

mongoexport နဲ့ထုတ်ယူထားတဲ့ Backup ဖိုင်ကို ပြန်ပြီး Restore လုပ်လိုရင်တော့ mongoimport ကို သုံးရပါတယ်။

```
$ mongoimport -d mydb -c issues < export.json
```

နှမူနာအရ mydb Database ရဲ့ issues Collection ထဲကို export.json ဖိုင်ထဲက အချက်အလက်တွေနဲ့ Restore လုပ်ပေးသွားမှာ ဖြစ်ပါတယ်။ mongorestore မှာလိုပဲ နိုဂုံရေးနဲ့ Collection ကိုအရင် Drop လုပ်စေလို ရင် --drop Option တဲ့ပေးနိုင်ပါတယ်။

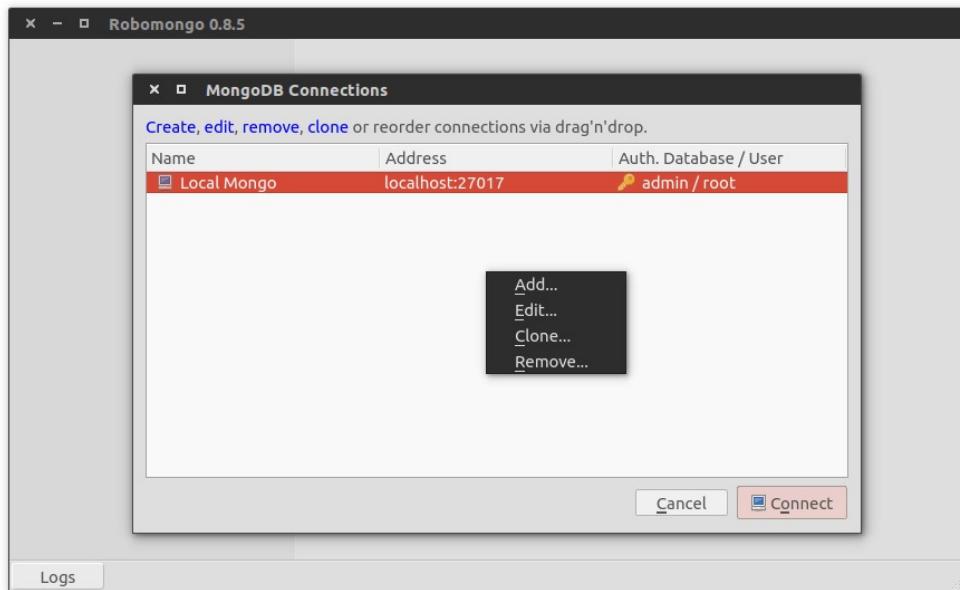
## 11.8 – MongoDB Administration Tool

MongoDB ကို Command Line ကနေစီမံမယ့်အစား Admin Tool တွေသုံးပြီးတော့လည်း စီမံနိုင်ပါတယ်။ အသုံးများ တဲ့ Admin Tool စာရင်းကို အောက်ပါလိုစာမှာ လေ့လာနိုင်ပါတယ်။

<http://docs.mongodb.org/ecosystem/tools/administration-interfaces/>

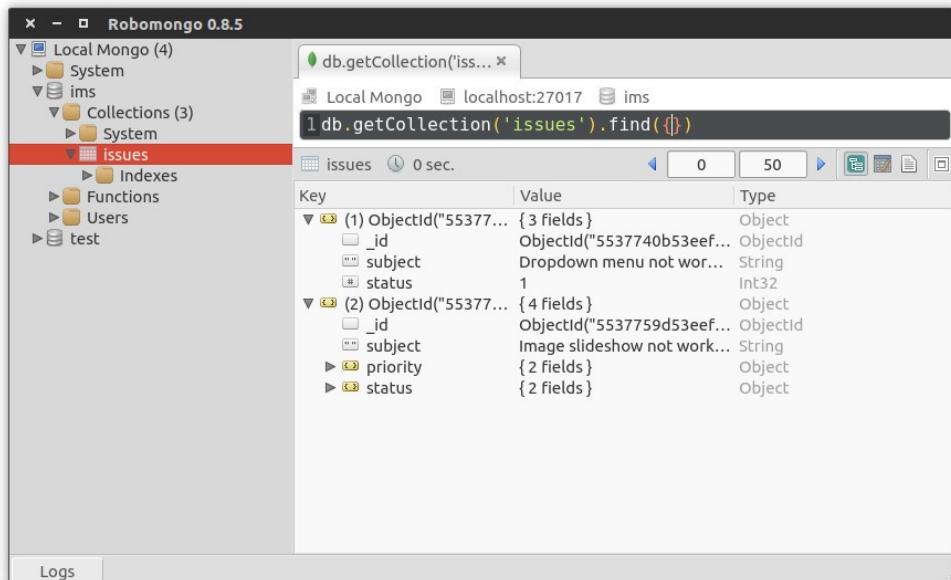
ဒီနေရာမှာတော့ နှမူနာအနေနဲ့ Robomongo လိုပေါ်တဲ့ Admin Tool တစ်ခုကို နှမူနာအနေနဲ့ ဖော်ပြလိုပါတယ်။ Robomongo ဟာ Windows, Linux, Mac စတဲ့ Platform အားလုံးမှာ အလုပ်လုပ်တဲ့ Tool တစ်ခုဖြစ်ပါတယ်။ [robomongo.org](http://robomongo.org) မှာ Installer ကို Download ရယူနိုင်ပါတယ်။

Robomongo ကို Install လုပ်ပြီးတဲ့အခါ ပထမဆုံးအနေနဲ့ Connection Setting သတ်မှတ်ပေးရပါတယ်။ **File** → **Connect** Menu ကို နိုပ်လိုက်ရင် အခုလို Connection Setting Manager ကိုတွေ့ရမှာဖြစ်ပါတယ်။



ပုံ (၁၁.၂) - Robomongo - Connection Manager

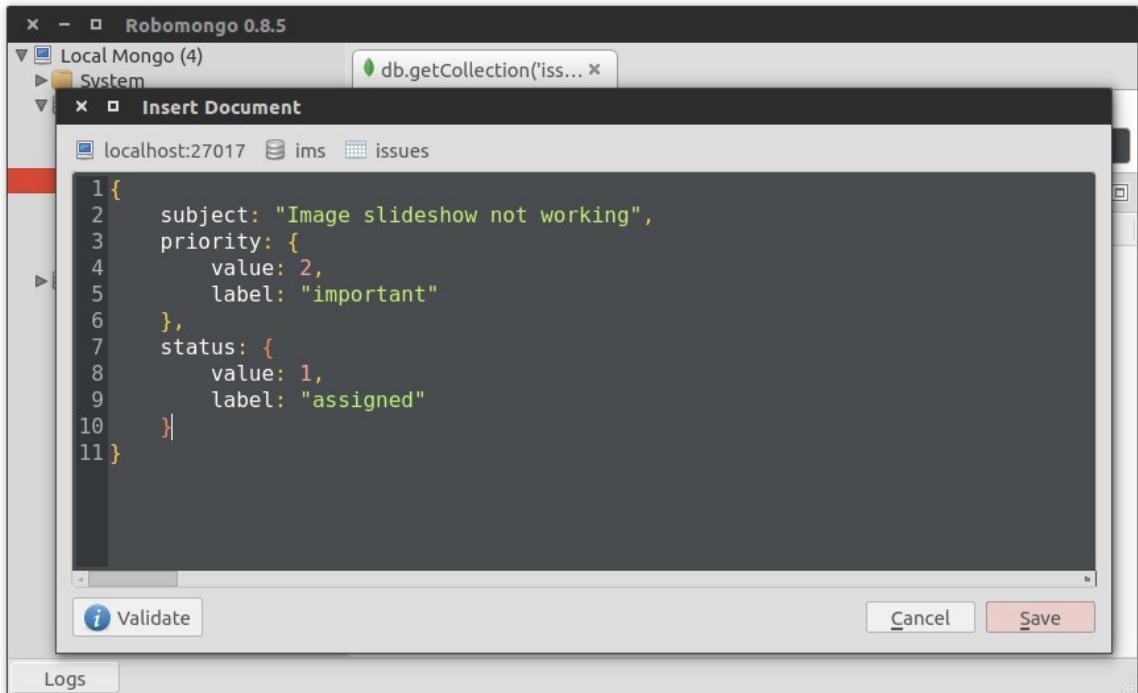
**Right Click → Add** ကို ရွေးပြီး Connection အသစ်တစ်ခုတည်ဆောက်နိုင်ပါတယ်။ ပေါ်လာတဲ့ Dialog Box ထဲမှာ Connection Name အနေနဲ့ နှစ်သက်ရာ အမည်ကိုပေးနိုင်ပါတယ်။ Address နေရာမှာ localhost နဲ့ Port နေရာမှာ 27017 ကိုပေးရပါတယ်။ အကယ်၍ Database မှာ Access Control သတ်မှတ်ထားတယ်ဆိုရင် Username, Password ကိုသတ်မှတ်ပေးရပါတယ်။ လိုအပ်တဲ့အချက်အလက်တွေသတ်မှတ်ပြီး ရရှိလာတဲ့ Connection ကိုရွေးပြီး Connect နိုင်လိုက်ရင် အခုလိုရလဒ်ကို ရရှိမှာဖြစ်ပါတယ်။



ပုံ (၁၁.၃) - Robomongo - Main Interface

ပုံ (၁၁.၈) မှာပြထားသလို ဘယ်ဘက်ခြစ်းမှာ Database Explorer ဖော်ပြစ်ဖို့အတွက် **View → Explorer** ကို ရွေးပေးဖို့လိုနိုင်ပါတယ်။ Database အသစ်ဆောက်ဖို့အတွက် Database Explorer ထဲက Connection ကို Right Click နိုပ်ပြီး Create Database ကိုရွေးခြင်းအားဖြင့် တည်ဆောက်နိုင်ပါတယ်။ Collection အသစ်တည်ဆောက်ဖို့အတွက် သက်ဆိုင်ရာ Database ကို Right Click နိုပ်ပြီး တည်ဆောက်နိုင်ပါတယ်။

ညာဘက်ခြစ်းမှာ လက်ရှိရွေးယားတဲ့ Collection ထဲက Document တွေကိုဖော်ပြပေးမှာဖြစ်ပါတယ်။ Document တွေ ကို Tree View, Table View, Document View စသဖြင့် View အမျိုးမျိုးနဲ့ကြည့်လို့ရပါတယ်။ Toolbar ထဲက Button တွေကို နိုပ်ပြီး View ကို အမျိုးမျိုးပြောင်းကြည့်နိုင်ပါတယ်။



ပုံ (၁၁.၉) - Robomongo - Document Editor

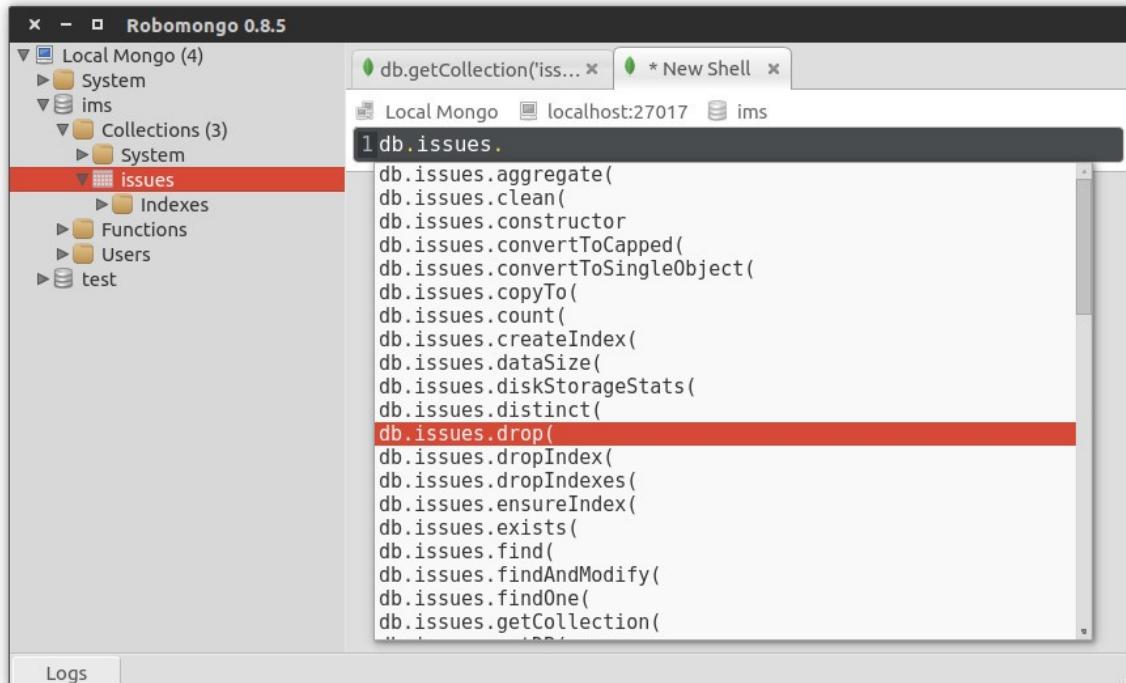
Document တွေကို Create, Read, Update, Delete လုပ်ဖို့အတွက် သက်ဆိုင်ရာ Document ကို Right Click လုပ်ပြီးဆောင် ရွက်နိုင်ပါတယ်။ Document အသစ်တည့်ဖို့အတွက် Document စာရင်းနေရာလွတ်မှာ (သို့မဟုတ်) Database Explorer ထဲက Collection ပေါ်မှာ Right Click နိုပ်ပြီး **Insert Document** ကိုရွေးခြင်းအားဖြင့် တည်ဆောက်နိုင်ပါတယ်။ Robomongo က Document Editor နဲ့ Document အသစ်တစ်ခုထည့်သွင်းနိုင်အောင် စီမံပေးမှာဖြစ်လို့ Command Line Mongo Shell မှာထက် အသုံးပြုရတာ ပိုမိုလွယ်ကူအဆင်ပြေစေမှာပဲဖြစ်ပါတယ်။

Index တွေစီမံဖို့အတွက်လဲ သက်ဆိုင်ရာ Collection အောက်က Index Folder ကို Right Click နိုပ်ပြီးစီမံနိုင်ပါတယ်။ လက်တွေမှာ Robomongo က MongoDB Shell ကိုပဲ အသုံးပြုထားခြင်းဖြစ်ပါတယ်။ ဆိုလိုတာက MongoDB

Shell မှာ လုပ်လို ရသမျှ Robomongo မှာ လည်း ရနိုင်မှာပဲဖြစ်ပါတယ်။

Shell Command တွေ ကိုယ်တိုင်ရေးသားလိုရင် Tab Bar မှာ Right Click လုပ်ပြီး New Shell ကိုရွေးချင်းအားဖြင့် Mongo Shell အသစ်တစ်ခုကို ဖွင့်နိုင်ပါတယ်။ ထူးခြားချက်အနေနဲ့ Robomongo ရဲ့ Shell Window ထဲ မှာ ပုံ (၁၁.၁၀) မှာ ပြထားသလို Auto-complete လုပ်ဆောင်ချက်ကို ရရှိနိုင်မှာပဲဖြစ်ပါတယ်။

ရေးသားထားတဲ့ Shell Command တွေကို Run ဖို့အတွက် Ctrl+R ကိုနိုင်ပြီး Run နိုင်ပါတယ်။ ဒါမှာမဟုတ် Tab ကို Right Click လုပ်ပြီး Re-execute Query ကိုရွေးပြီးတော့လည်း Run နိုင်ပါတယ်။



ပုံ (၁၁.၁၀) - Robomongo - Auto-Complete in Shell

လက်တွေမှာ Mongo Shell Command Line ကနေ စီမံနေ့မယ့်အစား အခုလို Admin Tool တွေကိုအသုံးပြုလိုက်တာ ပိုပြီးအလုပ်တွင်နိုင်ပါတယ်။ NoSQL Database တွေရဲ့ အမိကအားသာချက်ဟာ အလွယ်တစ်ကူ Scale လုပ်နိုင်ခြင်း ဖြစ်ကြောင်းဖော်ပြခဲ့ပါတယ်။ MongoDB ကို Scale လုပ်ရာမှာ လိုအပ်တဲ့နည်းစနစ်တွေဖြစ်တဲ့ Replication, Shard စတုး အကြောင်းအရာတွေကို မဖော်ပြရသေးပါဘူး။ အခန်း (၁၉) ရောက်တော့မှ ဆက်လက်ဖော်ပြပါမယ်။ တစ်ချို့ MongoDB Admin Tool တွေက Database တွေ Collection တွေကိုသာမက Replication နဲ့ Shard တွေကိုပါ စီမံပေးနိုင်ကြပါသေးတယ်။ ဒါကြောင့် အထက်မှာပေးထားတဲ့ Admin Tool စာရင်းလိပ်စာမှာ သွားရောက်လေ့လာပြီး အခြား Admin Tool တွေကိုလဲ လေ့လာစမ်းသပ်ကြည့်သင့်ပါတယ်။

## 11.9 – Using MongoDB with NodeJS

MongoDB ကို NodeJS နဲ့ တွဲဖက်အသုံးပြနိုင်ဖို့အတွက် Official JavaScript Driver ကို အခုလို Install လုပ်ယူနိုင်ပါတယ်။

```
$ npm install mongodb
```

အဲဒီ Official Driver ကိုပဲ Mongo Shell Command တွေနဲ့ အနီးစပ်ဆုံးတူအောင် ဖြည့်စွက်ထားတဲ့ mongojs ဆိုတဲ့ NPM Package တစ်ခုလည်းရှုပါသေးတယ်။ လက်တွေမှာ Mongo Shell တွေနဲ့ ပိုမိုနီးစပ်တဲ့အတွက် mongojs က Official Driver ထက် သုံးရတာပို့အဆင်ပြေပါတယ်။ နမူနာအနေနဲ့ mongojs ကိုအသုံးပြုဖော်ပြပေးသွားပါမယ်။ mongojs ကို အခုလို Install လုပ်နိုင်ပါတယ်။

```
$ npm install mongojs
```

Project မှာအသုံးချလိုတဲ့ NPM Package တွေကို Local Module အဖြစ်နဲ့ Project Director ထဲမှာ Install လုပ်သင့် ကြောင်းကို ပြီးခဲ့တဲ့အခန်းမှာ ဖော်ပြခြေးဖြစ်ပါတယ်။ ဒါကြောင့် mongojs ကို Install လုပ်တဲ့အခါ Local Module အနေနဲ့သာ Install လုပ်ထားတာကို သတိပြုပါ။ mongojs ကိုအသုံးပြုပြီး MongoDB ကို အခုလို ချိတ်ဆက်နိုင်ပါတယ်။

### JavaScript

```
1. var mongojs = require("mongojs");
2. var db = mongojs('mydb', ['issues', 'users']);
3.
4. db.users.find(function(err, data) {
5.   console.log(data);
6.   db.close();
7. });
```

လိုင်းနံပါတ် (၂) မှာ mongojs ကိုသုံးပြီး MongoDB Server ထဲက mydb Database ကိုချိတ်ဆက်ရွေးချယ်ထားပါတယ်။ အဲဒီလိုချိတ်ဆက်ရာမှာ Collection စာရင်းကိုပါ တစ်ခါတည်း Array အနေနဲ့ တွဲပေးရပါတယ်။ နမူနာ အရ issues နဲ့ users ဆိုတဲ့ Collection နှစ်ခုတွဲပေးထားပါတယ်။ ဒီလိုချိတ်ဆက်ပြီးတဲ့နောက်မှာ db ကနေ တစ်ဆင့် Mongo Shell Command တွေကို စတင်အသုံးပြု နိုင်မှာ ဖြစ်ပါတယ်။ လိုင်းနံပါတ် (၄) မှာ users Collection ထဲက Document တွေကို လိုချင်တဲ့အတွက် db.users.find() ကို သုံးထားပါတယ်။ MongoDB ကပြန်ပေးတဲ့ users Collection ထဲက Document တွေဟာ Callback Function ရဲ့ data Variable ထဲမှာ Array အနေနဲ့ ရှိနေမှာဖြစ်ပါတယ်။

လိုင်းနံပါတ် (၆) မှာ db.close() နဲ့ MongoDB Server နဲ့ချိတ်ဆက်ထားမှုကို ပိတ်ပေးထားတာကိုသတိပြုပါ။ Database နဲ့ ဆက်သွယ်ဆောင်ရွက်တဲ့ကိစ္စတွေပြီးတိုင်းမှာ Connection ကို ပြန်ပိတ်ပေးဖို့လိုပါတယ်။ အကယ်၍ Database မှာ Access Control သတ်မှတ်ထားရင်တော့ လိုင်းနံပါတ် (၂) ကို အခုလိုပြင်ဆင်

ချိတ်ဆက်ပေးရမှာပါ။

### JavaScript

```
var db = mongojs('username:password@localhost/mydb', ['issues', 'users']);
```

သက်ဆိုင်ရာ Username, Password ကို MongoDB နဲ့ချိတ်ဆက်စဉ် တစ်ခါတည်း ပေးလိုက်ရတဲ့သဘော ဖြစ်ပါတယ်။

ကျွန်ုတ် Create, Read, Update, Delete လုပ်နောက်အားလုံးဟာ Mongo Shell မှာ အသုံးပြုတဲ့အတိုင်းပဲ ဆက်လက် အသုံးပြု ရမှာဖြစ်ပါတယ်။ ဥပမာ - `find()` Function အတွက် Query ကို အခုလိုပေးနိုင်ပါတယ်။

### JavaScript

```
1. db.issues.find({ status: { $gt: 2 } }, function(err, data) {
2.   console.log(data);
3.   db.close();
4. });
```

ဒါဟာ `status` တန်ဖိုး 2 ထက်ကြီးတဲ့တန်ဖိုးရှိတဲ့ Document များကို Query လုပ်ယူလိုက်ခြင်းဖြစ်ပါတယ်။ Document အသစ်တစ်ခုကို အခုလို ထည့်သွင်းနိုင်ပါတယ်။

### JavaScript

```
1. db.users.insert({
2.   name: "Ei Maung",
3.   role: "Developer"
4. }, function() {
5.   db.close();
6. });
```

Document တစ်ခုကို ပယ်ဖျက်လိုရင် အခုလို ပယ်ဖျက်နိုင်ပါတယ်။

### JavaScript

```
1. db.users.remove({
2.   _id: mongojs.ObjectId("5537b76a292e04f67d2af4c1")
3. }, function() {
4.   db.close();
5. });
```

`ObjectId()` Function ကို တိုက်ရှိက်သုံးလိုမရပဲ mongojs ကနေတစ်ဆင့် သုံးရတာကို သတိပြုပါ။ Document တွေကို Update ပြုလုပ်လိုရင်လည်း အခုလိုပြုလုပ်နိုင်ပါတယ်။

## JavaScript

```

1. db.users.update(
2.   { role: "Admin" },
3.   { $set: { role: "Developer" } },
4.   { multi: true },
5.   function() {
6.     db.close();
7.   }
8. );

```

Callback Function တွေ ပါသားတာကလဲရင် MongoDB အသုံးပြုပဲနဲ့ အားလုံးအတူတူပဲဆိုတာကို တွေ့ရမှာ ဖြစ်ပါ တယ်။ Index တည်ဆောက်ပဲတွေ runCommand() အသုံးပြုပဲတွေလည်း အတူတူပဲဖြစ်ပါတယ်။ ဥပမာ – Text Index တစ်ခုအခါးလို့ တည်ဆောက်နိုင်ပါတယ်။

## JavaScript

```

1. db.issues.createIndex({ subject: "text" }, {}, function() {
2.   db.close();
3. });

```

Text Index ထဲကနေ ပြန်လည်ရှာဖွေလိုတဲ့ အခါး၊ အခါးလို့ ပြည်လည်ရှာဖွေနိုင်ပါတယ်။

## JavaScript

```

1. db.issues.runCommand("text", { search: "menu" }, function(err, result) {
2.   console.log(result);
3.   db.close();
4. });

```

ဒီနည်းအတိုင်း NodeJS ရဲ့ Event Driven, Non-blocking I/O လုပ်ဆောင်ချက်တွေနဲ့ MongoDB ရဲ့ Scalability အားသာ ချက်တိုကို တွဲဖက်ပြီး စွမ်းဆောင်ရည်ဖြင့် Database အခြေပြု App တွေကို ဖန်တီးတည်ဆောက်နိုင် မှာပဲဖြစ် ပါတယ်။

တစ်ချို့လည်း RDBMS တွေနဲ့အတူ ORM Library တွေကို တွဲဖက်အသုံးပြုတက်ကြပါတယ်။ NodeJS အတွက် လည်း mongoose လို့ခေါ်တဲ့ ODM Module တစ်ခုကို လူသုံးများပါတယ်။ Relational Database မဟုတ်တဲ့ အတွက် ORM လို့ မခေါ်ပဲ ODM လို့ခေါ်ခြင်းဖြစ်ပါတယ်။ သဘောသဘာဝကတော့ အတူတူပါပဲ။ ဒီနေရာမှာ ORM နဲ့ mongoose အကြောင်းကို ထည့်သွင်းမဖော်ပြနိုင်တော့ပါဘူး။ mongoose အကြောင်းကို [mongoosejs.com](http://mongoosejs.com) မှာ ဆက်လက်လေ့လာ နိုင်ပါတယ်။

## Conclusion

ဒီအခန်းရဲ့အစမှာ RDBMS တွေမှာ ရှိတက်တဲ့ အခက်အခဲတွေကို ဖြေရှင်းနိုင်ဖို့အတွက် NoSQL Database တွေပေါ်ပေါက်လာတယ်လို့ဖော်ပြခဲ့ပါတယ်။ ဒါပေမယ့် RDBMS တွေမှာရှိနေတဲ့ ACID Compliance ဆိုတဲ့ အချက်အလက်တိကျ သေချာမှု အာမခံအားသာချက်ကိုလည်း မေ့မထားသင့်ပါဘူး။ NoSQL ကို Not Only SQL လို့ ဆိုကြပါတယ်။ RDBMS တွေကို အစားထိုးဖို့ ရည်ရွယ်ချက်နဲ့ ပေါ်လာခြင်းမဟုတ်ပဲ၊ RDBMS တွေနဲ့ အံမဝင်တဲ့ Software အမျိုးအစားတွေမှာ အသုံးချနိုင်ဖို့ဆိုတဲ့ ရည်ရွယ်ချက်နဲ့ ပေါ်ပေါက်လာခြင်းဖြစ်ပါတယ်။ တစ်ချို့အဖွဲ့အစည်းတွေမှာဆိုရင် RDBMS နဲ့ NoSQL Database တွေကို အတူတစ်ကွပ်းတဲ့အသုံးချကြတာတွေလည်း ရှိပါတယ်။ ဥပမာ - အချက်အလက်တိကျ သေချာဖို့လိုတဲ့ အပိုင်းတွေကို RDBMS တွေသုံးပြီး၊ Cache လုပ်ဖို့ Real-time Data, State Data စတဲ့ Unstructured Data တွေကို စီမံဖို့အတွက် NoSQL Database တွေကို တဲ့ဖက်အသုံးပြုကြခြင်းဖြစ်ပါတယ်။

ကိုယ်ရဲ့ Software Project အတွက်ဘယ် Database အမျိုးအစားကို ရွေးချယ်ရမလဲလို့ စဉ်းစားစရာဖြစ်လာတိုင်း၊ ACID Compliance နဲ့ Scalability နှစ်ခုမှာ ဘာပိုအရေးကြီးလဲဆိုတဲ့ အချက်ပေါ်အခြေခံဆုံးဖြတ်ကြရမှာ ဖြစ်ပါတယ်။ အချက် အလက် တိကျသေချာမှုအာမခံချက် မဖြစ်မနေလိုအပ်တဲ့ Software တွေမှာ (ဥပမာ - Financial Solution) RDBMS တွေကို ရွေးချယ်အသုံးပြုရမှာဖြစ်ပြီး၊ အချက်အလက်ပေါင်းများစွာကို မြန်ဆန်ထိရောက်စွာစီမံနိုင်မှုပိုအရေးကြီးတက်တဲ့ Software တွေမှာ (ဥပမာ - Communication App) NoSQL Database တွေကို ရွေးချယ်အသုံးပြုကြရမှာပဲဖြစ်ပါတယ်။

MongoDB ရဲ့ အမိကကျကဲ့လုပ်ဆောင်ချက်နှစ်ခုဖြစ်တဲ့ Replication နဲ့ Sharded Cluster အကြောင်းကို အခန်း(၁၉) မှာ ဆက်လက်ဖော်ပြပေးသွားမှာပဲ ဖြစ်ပါတယ်။

တစ်ခို့ Framework တွေဟာ၊ အစစအရာအရာကြိုတင်

ဆုံးဖြတ်သတ်မှတ်ပေးထားလေ့ရှိတဲ့ Opinionated Framework

တွေဖြစ်ကြပါတယ်။ တစ်ခို့ Framework တွေကတော့ လိုအပ်တဲ့

အခြေခံကိုသာပေးထားပြီး၊ အသေးစိတ်ကအစ ကိုယ်တိုင်ဆုံးဖြတ်

ရေးသားခွင့်ပြုတဲ့ Unopinionated Framework တွေဖြစ်ကြပါတယ်။

သူအားသာချက်နဲ့သူပဲမို့ ရွှေးချယ်မှု မှန်ကန်ဖို့လိုပါတယ်။

## **Professional Web Developer (စာအုပ်)**

Web Standard, jQuery, PHP, MySQL, Ajax, CMS, MVC,

HTML5, Mobile Web, Web Application Security စသည်

အကြောင်းအရာများကို ရေးသားဖော်ပြထားသည့်စာအုပ်

အောက်ပါလိပ်စာတွင် အခမဲ့ Download ရယူနိုင်သည်။

## အခန်း(၁၂) – ExpressJS (REST Framework)

ExpressJS ဟာ NodeJS ပေါ်မှာအခြေခံထားတဲ့ REST Framework တစ်ခုဖြစ်ပါတယ်။ အခြား Full-scaled App Development Framework တွေလို ရှုပ်ထွေးတဲ့လုပ်ဆောင်ချက်တွေ မပါဝင်ပဲ REST API အခြေခြား Service တွေ တည်ဆောက်ရာမှာ လိုအပ်နိုင်တဲ့ အခြေခံလုပ်ဆောင်ချက်များဖြစ်တဲ့ Routing, Request/Response Handling စတဲ့ ကိစ္စတွေသာ အမိကပါဝင်တဲ့ ရိုးရှင်းတဲ့ Unopinionated Framework တစ်ခုဖြစ်ပါတယ်။

**အခန်း (၄)** မှာ Service Oriented Architecture (SOA) အကြောင်းလေ့လာခဲ့ကြစဉ်က၊ HTTP ဟာ REST နှင့် ပညာသတ်မှတ်ချက်နဲ့အညီ ဖန်တီးတိတွင်ထားတဲ့ Protocol တစ်ခုဖြစ်တဲ့အကြောင်း ဖော်ပြခဲ့ပါတယ်။ ဒါ ကြောင့် REST Web Service တွေ တည်ဆောက်တဲ့အခါ HTTP ကို ထိရောက်အောင်အသုံးချမှတ်ဖို့လိုပါတယ်။  
**အခန်း (၁၀)** မှာ NodeJS အကြောင်း ဖော်ပြခဲ့ရာမှာတော့ NodeJS ရဲ့ HTTP Module ဟာ Low Level Module တစ်ခုဖြစ်ပြီး အသေးစိတ်လုပ်ဆောင်ချက်တွေကို ကိုယ်တိုင်ရေးသားရမှာ ဖြစ်တဲ့အကြောင်း ဖော်ပြခဲ့ပါတယ်။ ဒီနေရာမှာ ExpressJS ဝင်လာခြင်းဖြစ်ပြီး NodeJS HTTP Module မှာ မပါဝင်တဲ့ Function တွေကို ကြိုတင်ဖန်တီးပေးထားခြင်းပဲဖြစ်ပါ တယ်။

### 12.1 – Simple ExpressJS App

ExpressJS ဟာ NodeJS Module တစ်ခုဖြစ်တဲ့အတွက် NPM နဲ့ အခုလို Install လုပ်ယူနိုင်ပါတယ်။

```
$ npm install express
```

Install လုပ်ပြီးတဲ့အခါ ExpressJS ရဲ့ `listen()`, `get()` နဲ့ `res.send()` Function တို့ကို အသုံးပြုပြီး Web Service တစ်ခုကို စတင်တည်ဆောက်နိုင်ပြီဖြစ်ပါတယ်။

ဒီအခန်းမှာ ဖော်ပြထားတဲ့ Code တွေဟာ လက်တွေ့ကူးယူစစ်းသပ်နိုင်တဲ့ Code တွေဖြစ်ပါတယ်။

`listen()` Function ကို Web Server Run ဖို့အတွက် အသုံးပြုပြီး `get()` Function ကို Client Request တွေ လက်ခံဖို့သုံးရပါတယ်။ `send()` Function ကိုတော့ Client ထံ Response ပြန်ပေးပို့ဖို့သုံးရပါတယ်။ ဥပမာ -

## JavaScript

```

1. var express = require("express");
2. var app = express();
3.
4. var server = app.listen(3000, function(){
5.     console.log("ExpressJS server running on port 3000");
6.
7.     app.get('/', function(req, res){
8.         res.send("Home Page");
9.     });
10.
11.    app.get('/about', function(req, res){
12.        res.send("About Us Page");
13.    });
14. });

```

နဲ့နာလိုင်းနံပါတ် (၄) မှာ listen() Function ကိုသုံးပြီး Port နံပါတ် 3000 မှာ Web Server တစ်ခုကို Run ထားပါတယ်။ လိုင်းနံပါတ် (၇) မှာ get() Function ကိုသုံးပြီး၊ URI အလွတ်နဲ့ Request တစ်ခုလက်ခံရရှိတဲ့ အခါ res.send() ကိုသုံးပြီး Home Page ဆိုတဲ့ Text Response တစ်ခုကို Respond ပြန်ပေးဖို့ သတ်မှတ် ထားပါ တယ်။ လိုင်းနံပါတ် (၁၁) မှာ တော့ /about URI ကိုလက်ခံရရှိတဲ့အခါ About Us Page ဆိုတဲ့ Text Response တစ်ခုကို ပြန်လည်ပေးပို့ထားပါတယ်။ ဒါ Code ကို server.js အမည်နဲ့ ရေးသားပြီး Run ကြည့် နိုင်ပါတယ်။

```

$ node server.js
ExpressJS server running on port 3000

```

Web Browser ကိုဖွင့်ပြီး localhost:3000 ကိုသွားကြည့်ရင် Home Page ဆိုတဲ့ Response ကို လက်ခံရရှိမှာဖြစ်ပြီး localhost:3000/about ကိုသွားကြည့်ရင်တော့ About Us Page ဆိုတဲ့ Response ကို လက်ခံရရှိမှာပဲဖြစ်ပါတယ်။

URI ပေါ်သာမက Request Method ပေါ်မှုတည်ပြီးတော့လည်း Response တွေကို စီမံနိုင်ပါသေးတယ်။ နဲ့နာ မှာ get() Function ကိုသုံးထားတဲ့အတွက် GET Request Method တွေကို လက်ခံစီမံပေးနေခြင်းဖြစ်ပါတယ်။ POST, PUT, DELETE အစရှိတဲ့ Request Method တွေကိုလက်ခံစီမံဖို့အတွက် get() အစား post(), put(), delete() စတဲ့ Function တွေကို အစားထိုး အသုံးပြုနိုင်ပါတယ်။

ပြီးခဲ့တဲ့နဲ့နာမှာလို့ Text တွေကို Response မပြန်ပဲဖို့တွေကို Response ပြန်ပေးလိုတယ်ဆိုရင် res.send() အစား res.sendFile() ကို အစားထိုးအသုံးပြုနိုင်ပါတယ်။ ဥပမာ -

## JavaScript

```

1. var express = require("express");
2. var app = express();
3.
4. var server = app.listen(3000, function(){
5.     console.log("ExpressJS server running on port 3000");
6.
7.     app.get('/', function(req, res){
8.         res.sendFile( __dirname + "/index.html" );
9.     });
10.
11.    app.get('/about', function(req, res){
12.        res.sendFile( __dirname + "/about.html" );
13.    });
14. });

```

နမူနာအရ URI အလွတ်နဲ့ Request လုပ်လာခဲ့ရင် index.html ကို Response ပြန်ပေးသွားမှာဖြစ်ပြီး /about URI နဲ့ Request လုပ်လာခဲ့ရင်တော့ about.html ကို Response ပြန်ပေးသွားမှာဖြစ်ပါတယ်။ \_\_dirname Constant ကိုလက်ရှိ Directory ရဲ့ Absolute Path ကိုလိုချင်တဲ့အခါ သုံးရပါတယ်။

တစ်ခုသတိပြုရမှာကတော့ res.sendFile() ဟာ Static ဖိုင်တွေကိုသာ Response လုပ်ပေးနိုင်ပါတယ်။ အကယ် ၍ Dynamic Template တွေကို Response လုပ်လိုရင်တော့ res.render() Function ကို အသုံးပြု နိုင်ပါတယ်။ res.render() Function ကို Jade, Ejs, Handlebar စိတ် JavaScript Template Engine တစ်ခုခုနဲ့ တွဲသုံး ပေးရပါတယ်။ နမူနာအနေနဲ့ Ejs ကို အသုံးပြု ဖော်ပြပေးပါမယ်။ Ejs Template Engine ကို အခုလို Install လုပ်နိုင်ပါ တယ်။

```
$ npm install ejs
```

ပြီးတဲ့အခါ စောစောက Code နမူနာကို အခုလို ပြင်ဆင်ဖြည့်စွက်ပေးရပါမယ်။

## JavaScript

```

1. var express = require("express");
2. var app = express();
3.
4. app.set('views', './views');
5. app.engine('html', require('ejs').renderFile);
6.
7. var server = app.listen(3000, function(){
8.     console.log("ExpressJS server running on port 3000");
9.
10.    app.get('/', function(req, res){
11.        res.render("index.html");
12.    });
13.
14.    app.get('/about', function(req, res){
15.        res.render("about.html", { greeting: "Hello, World!" });

```

```
16.      });
17.  );
```

လိုင်းနံပါတ် (၄) မှာ `app.set()` Function ကိုသုံးပြီး HTML Template တွေရှိရာ Directory ကို သတ်မှတ်ပေးထားပါတယ်။ ဒါကြောင့် HTML Template ဖိုင်တွေကို `views` အမည်ရှိ Directory တစ်ခုနဲ့ စုစုပေါင်းထားရမှာဖြစ်ပါတယ်။ လိုင်းနံပါတ် (၅) မှာတော့ `app.engine()` Function ကိုသုံးပြီး `html` Extension နဲ့ ဖိုင်တွေအတွက် Ejs Template Engine ကိုသုံးဖို့ သတ်မှတ်ထားပါတယ်။ လိုင်းနံပါတ် (၁၅) မှာ `res.render()` Function နဲ့ `about.html` ကို Response ပြန်စိုင်းထားပါတယ်။ အဲဒါလို Response ပြန်ရာမှာ တွဲဖက်အသုံးပြုရမယ့် တန်ဖိုးတွေကို JSON Object အနေနဲ့ တွဲဖက်ပေးနိုင်ပါတယ်။

Ejs ဟာ ကျွန်တော်တို့ အခန်း (၆) မှာ BackboneJS အကြောင်းလေ့လာခဲ့စဉ်က ထည့်သွင်းအသုံးပြုခဲ့တဲ့ Underscore ရဲ့ Template နဲ့ ရေးသားပုံ ရေးထုံးဆင်ပါတယ်။ JavaScript Code တွေ HTML ထဲမှာရောရေးဖို့ အတွက် `<% %>` Tag ကို အသုံးပြုပြီး Output တစ်ခါတည်း ရိုက်ထုတ်စေလိုရင်တော့ `<%= %>` Tag ကို သုံးရပါတယ်။ `about.html` ထဲမှာ အခဲလို ရေးသားထားတယ်ဆိုကြပါစိုး။

### Template

```
<h1>About Us</h1>
<p><%= greeting %></p>
```

[localhost:3000/about](http://localhost:3000/about) လို့ Request လုပ်ကြည်ရင် `about.html` ကို Response အနေနဲ့လက်ခံရရှိရမှာဖြစ်ပြီး ရှိခိုးမယ့် မလိုပြုခြင်းနေမှာပါ။

### HTML

```
<h1>About Us</h1>
<p>Hello, World!</p>
```

`greeting` Variable အစား `Hello, World!` ဆိုတဲ့တန်ဖိုးကို Ejs က အစားထိုးထည့်သွင်းပေးလိုက်ခြင်းပဲဖြစ်ပါတယ်။

ဒီနည်းနဲ့ Text တွေ Static ဖိုင်တွေသာမက Dynamic Template တွေကိုပါ Response ပြန်ပေးနိုင်တဲ့ App တွေ တည် ဆောက်နိုင်မှာပဲဖြစ်ပါတယ်။ ဒါပေမယ့် လက်တွေမှာ User Interface နဲ့ Template လုပ်ဆောင်ချက်တွေ ကို ExpressJS နဲ့မလုပ်ပဲ BackboneJS လို့ နည်းပညာမျိုးကိုသုံးပြီး သီးခြား UI Client အနေနဲ့သာ တည်ဆောက်သင့်ပါတယ်။ ExpressJS ကိုတော့ UI Client က ဆက်သွယ်လာတဲ့အခါ လက်ခံဆောင်ရွက်ပေးနိုင်တဲ့ Service တွေ တည်ဆောက်ဖို့ အတွက် အသုံးပြုသင့်ပါတယ်။ BackboneJS နဲ့ ExpressJS တို့ကိုပူးတွဲအသုံးချထားတဲ့ နဲ့ နဲ့ App ကို အပိုင်း (၄) မှာ ဆက်လက်ဖော်ပြပေးသွားပါမယ်။

`res.send()`, `res.sendFile()` နဲ့ `res.render()` တို့အပြင် သုံးလိုရတဲ့အခြား Response Function

တွေ့ရှုပါသေးတယ်။ `res.download()`, `res.json()`, `res.jsonp()` တို့ပြစ်ပါတယ်။ `res.download()` Function က `res.sendFile()` လိုပဲ ဖိုင်ကို Response ပြန်ပေးခြင်းဖြစ်ပါတယ်။ ဒါပေမယ့် ဖော်ပြန့်မဟုတ်ပဲ Client အနေနဲ့ Download လုပ်ယူဖို့ ပေးပို့ခြင်းဖြစ်ပါတယ်။ `res.json()` နဲ့ `res.jsonp()` Function တို့ကတော့ `send()` လိုပဲ Text တွေကို Response ပြန်ပေးခြင်းဖြစ်ပါတယ်။ ဒါပေမယ့် `Response` ရဲ့ Content-Type Header ကို JSON လို့ တစ်ခါတည်း သတ်မှတ်ပေးသွားမှာ ဖြစ်ပါတယ်။

## 12.1 – Routing

REST API အခြေခြား Web Service တွေတည်ဆောက်တဲ့အခါ အခြေခံအကျဆုံးလိုက်နာရမယ့် အချက်နှစ်ချက် ရှိပါတယ်။ HTTP Request Method တွေကို ထိရောက်အောင် အသုံးချဖို့နဲ့ နားလည်ရလွယ်ကူတဲ့ URL တွေကို အသုံးပြုဖို့ပြစ်ပါတယ်။ ExpressJS က အဲဒီလုပ်ဆောင်ချက်တွေရရှိဖို့အတွက် အသုံးပြနိုင်တဲ့ Routing Function တွေကို အသင့်ပေးထားပါတယ်။

ဥပမာအားဖြင့် ကျွန်ုတ်တို့ App အတွက် အခုလို URL Structure ကို ရရှိလိုတယ်ဆိုပါစို့။

Request Method	URL	Purpose
GET	users/	User စာရင်းရယူရန်
GET	users/123	User အမှတ် 123 ၏ အချက်အလက်များရယူရန်
POST	users/	User အသစ်တည်ဆောက်ရန်
PUT	users/123	User အမှတ် 123 ၏ အချက်အလက်များပြင်ဆင်ရန်
DELETE	users/	User အားလုံးအား ပယ်ဖျက်ရန်
DELETE	users/123	User အမှတ် 123 အား ပယ်ဖျက်ရန်

ExpressJS နဲ့ လိုချင်တဲ့ URL Structure ရအောင် အခုလိုသတ်မှတ်ပေးနိုင်ပါတယ်။

### JavaScript

```
app.get("/users", function(req, res) {
  console.log("Getting user list");
});

app.get("/users/:id", function(req, res) {
  var userId = req.params.id;
  console.log("Getting user %s", userId);
});

app.post("/users", function(req, res) {
  console.log("Creating a new user");
});
```

```
app.put("/users/:id", function(req, res) {
  var userId = req.params.id;
  console.log("Updating user %s", userId);
});

app.delete("/users", function(req, res) {
  console.log("Removing all users");
});

app.delete("/users/:id", function(req, res) {
  var userId = req.params.id;
  console.log("Removing user %s", userId);
});
```

/users URL ကို GET Method နဲ့ Request ပြုလုပ်လာခဲ့ရင် app.get("/users") ရဲ့ Callback Function က အလုပ်လုပ်သွားမှာဖြစ်ပြီး /users URL ကိုပဲ POST Method နဲ့ Request ပြုလုပ်ခဲ့ရင် app.post("/users") ရဲ့ Callback Function က အလုပ်လုပ်သွားမှာပဲဖြစ်ပါတယ်။ ဒီနည်းနဲ့ Request Method ပေါ်မှတည်ပြီး အလုပ်လုပ် နှင့်တဲ့ REST API အခြေခြား Web Service တွေကို ExpressJS နဲ့ အလွယ်တစ်ကူ တည်ဆောက်နိုင်မှာ ဖြစ်ပါတယ်။

နမူနာမှာ :id ကို URL Parameter အနေနဲ့အသုံးပြုထားတာကို သတိပြုကြည့်ပါ။ :id ကဲ့သို့ အလားတူ Parameter တွေကို URL Pattern ထဲမှာ လိုအပ်သလို ထည့်သွင်းအသုံးပြုနိုင်ပါတယ်။ ဥပမာ -

### JavaScript

```
app.get("/users/:name/:id", function(req, res) {
  var name = req.params.name;
  var id = req.params.id;
});
```

ဒီလိုသတ်မှတ်ထားတဲ့အတွက် users/john/profile/123 ဆိုတဲ့ URL ကိုလက်ခံရရှိတဲ့အခါ name အတွက် တန်ဖိုး john ကိုလက်ခံရရှိမှာဖြစ်ပြီး id အတွက်တန်ဖိုးအနေနဲ့ 123 ကို လက်ခံရရှိမှာပဲဖြစ်ပါတယ်။ Method ကို မကြည့်ပဲ ဘယ်လို Method နဲ့ပဲ Request ပြုလုပ်ခဲ့သည့်ဖြစ်စေ ဆောင်ရွက်စေလိုတဲ့ လုပ်ငန်းတွေကိုတော့ all() Function နဲ့ သတ်မှတ်ထားနိုင်ပါတယ်။ ဥပမာ -

### JavaScript

```
app.all("/users", function(req, res) {
  console.log("Request received");
  next();
});
```

ပုံမှန်အားဖြင့် all() Function အတွင်းမှာ next() Function ကို ခေါ်ယူပေးရလေ့ရှိပါတယ်။ ဒီတော့မှ အဲဒီ URL အတွက်သတ်မှတ်ထားတဲ့ get(), post() စတဲ့ သီးခြား Function တွေ ရှိနေသေးရင် ဆက်လက် အလုပ်လုပ် ဆောင် ပေးသွားမှာဖြစ်ပါတယ်။

URL တစ်ခုတည်းအတွက် Method အမျိုးမျိုးကို တစ်ခါတည်းအတွဲလိုက် သတ်မှတ်လိုရင်တော့ `route()` Function ကို သုံးနိုင်ပါတယ်။ ဥပမာ –

### JavaScript

```
app.route("users/")
  .get(function(req, res) {
    console.log("Getting user list");
  })
  .delete(function(req, res) {
    console.log("Removing all users");
  });

```

App ရဲ့ အစိတ်အပိုင်းတစ်ခုစီအတွက် Routing Pattern တွေကို သီးခြားစီ ခွဲခြားရေးသားလိုတယ်ဆိုရင် Router Constructor ကို အသုံးပြု နိုင်ပါတယ်။ ဥပမာ – `home-router.js` ဆိုတဲ့ ဖိုင်မှာ အခုလို ရေးသားထားပါမယ်။

### JavaScript

```
1. var express = require("express");
2. var router = express.Router();
3.
4. router.get("/", function(req, res) {
5.   res.send("Home");
6. });
7.
8. router.get("/about", function(req, res) {
9.   res.send("About Us");
10. });
11.
12. module.exports = router;
```

ဆက်လက်ပြီး `user-router.js` ဆိုတဲ့ဖိုင်မှာ အခုလိုရေးသားထားပါမယ်။

### JavaScript

```
1. var express = require("express");
2. var router = express.Router();
3.
4. router.get("/", function(req, res) {
5.   res.send("User List");
6. });
7.
8. router.get("/:id", function(req, res) {
9.   var id = req.params.id;
10.  res.send("View user %s", id);
11. });
12.
13. module.exports = router;
```

ဒီတော့ Home Sub App အတွက် Route Pattern တစ်ခုနဲ့ User Sub App အတွက် Route Pattern တစ်ခု ကိုယ်စိုင်ရှိ ရရှိသွားတဲ့သဘောဖြစ်ပါတယ်။ App ထဲမှာ ဒီ Router နှစ်ခုကို Sub App အနေနဲ့ အခုလို ကြော် အသုံးချခိုင်ပါတယ်။

### JavaScript

```

1. var express = require("express");
2. var homeRouter = require("./home-router.js");
3. var userRouter = require("./user-router.js");
4. var app = express();
5.
6. app.use("/home", homeRouter);
7. app.use("/users", userRouter);
8.
9. var server = app.listen(3000, function(){
10.     console.log("ExpressJS server running on port 3000");
11. });

```

လိုင်နံပါတ် (၆) မှာ /home အတွက် homeRouter ကို Sub App အနေနဲ့သုံးဖို့ သတ်မှတ်ထားပြီး၊ လိုင်နံပါတ် (၇) မှာ /users အတွက် userRouter ကို Sub App အနေနဲ့သုံးဖို့ သတ်မှတ်ထားပါတယ်။ ဒါကြောင့် /home နဲ့ စတဲ့ URL တိုင်းအတွက် home-router.js Sub App က အလုပ်လုပ်ပေးသွားမှာဖြစ်ပြီး၊ /users နဲ့ စတဲ့ URL တိုင်းအတွက် user-router.js Sub App က အလုပ်လုပ်ပေးသွားမှာဖြစ်ပါတယ်။ နမူနာအရ URL က /home/about ဆိုရင် **About Us** ကို Response ပြန်ပေးသွားမှာဖြစ်ပြီး URL က /users/123 ဆိုရင်တော့ **Viewing user 123** ကို Response ပြန်ပေးသွားမှာပါ။

## 12.2 – Serving Statistic Files

NodeJS ရဲ့ HTTP Module အကြောင်းလေ့လာခဲ့စဉ်က Request ပြုလုပ်လာတဲ့ Path လမ်းကြောင်းအတိုင်း Static HTML ဖိုင်တွေကို အလိုအလျောက် Response ပြုလုပ်ပေးပိုကို ဖော်ပြခဲ့ပါတယ်။ အဲဒီလိုဖော်ပြခဲ့ရမှာ NodeJS နဲ့ ကိုယ်တိုင်ရေးသားပုံ နမူနာကိုပေးခဲ့သလို connect နဲ့ serveStatic လိုပေါ်တဲ့ NPM Module တွေကိုအသုံး ပြနိုင်ပုံကိုလည်း ဖော်ပြခဲ့ပါတယ်။ ExpressJS ကိုလည်း အဲဒီလို URL Path လမ်းကြောင်းနဲ့ကိုက်ညီတဲ့ Static ဖိုင်တွေ အလိုအလျောက် Response ပြန်ပေးနိုင်တဲ့ Static Web Server တစ်ခုအနေနဲ့ အသုံးပြနိုင်ပါတယ်။ အခုလို ရေးသား ပေးရမှာဖြစ်ပါတယ်။

### JavaScript

```

1. var express = require("express");
2. var app = express();
3.
4. app.use(express.static("static"));
5.
6. var server = app.listen(3000, function(){
7.     console.log("ExpressJS server running on port 3000");
8. });

```

`express.static()` Function ကိုသုံးပြီး HTML ဖိုင်တွေတည်ရှိရာ Directory ကို သတ်မှတ်ပေးရပါတယ်။ ဒါကြောင့် [localhost:3000/about.html](http://localhost:3000/about.html) ဆိတဲ့ Request ကိုလက်ခံရရှိရင် ExpressJS က static Director ထဲက `about.html` ဆိတဲ့ဖိုင်ကို Response ပြန်ပေးသွားမှာပဲဖြစ်ပါတယ်။

Static ဖိုင် Server လုပ်ဆောင်ချက်ကို Sub App အနေနဲ့ တွဲသုံးလိုတယ်ဆိုရင် လိုင်းနံပါတ် (၄) ကို အခုလို ပြင်ဆင် ရေးသားနိုင်ပါတယ်။

### JavaScript

```
app.use('/static', express.static('static'));
```

ဒီတော့ ExpressJS က `/static` နဲ့စတဲ့ URL တွေကိုလက်ခံရရှိတော့မှာသာ Static ဖိုင် Server လုပ်ဆောင်ချက် ကို အလုပ်လုပ် ပေးသွားတော့မှာဖြစ်ပါတယ်။

### 12.3 – Request Handling

Web Service တစ်ခုအနေနဲ့ Request တွေကိုလက်ခံရရှိတဲ့အခါ၊ အဲဒီ Request တွေနဲ့အတူတဲ့ဖက်ပါဝင်လာတဲ့ Query String တွေ၊ Request Data တွေ၊ ဖိုင်တွေနဲ့ Cookie တွေကို လက်ခံစီမံနိုင်ဖို့လိုပါတယ်။ ExpressJS ရဲ့ အရင် Version တွေမှာ Request Data တွေစီမံနိုင်တဲ့ လုပ်ဆောင်ချက်တွေ တစ်ခါတည်းပါဝင်ပေမယ့်၊ နောက်ပိုင်း Version တွေမှာ Query String တွေလက်ခံစီမံတဲ့ လုပ်ငန်းတစ်ခုသာ Build-in ပါဝင်ဖြီး၊ ကျွန်ုတ်ပုံးတွေ အတွက် သီးခြား Module တွေကို ထပ်မံထည့်သွင်းပေး ရပါတယ်။

Request နဲ့အတူပါဝင်လာတဲ့ Query String Data တွေကို ExpressJS က `req.query` Property ထဲမှာ JSON Object တစ်ခုအနေနဲ့ လက်ခံထားပေးပါတယ်။ ဒါကြောင့် Query String Data တွေကိုစီမံဖို့လိုတဲ့အခါ `req.query` ကနေ အသင့်ရယူအသုံးပြန်စိုင်မှာပဲ ဖြစ်ပါတယ်။ ဥပမာ –

### JavaScript

```
1. var express = require("express");
2. var app = express();
3.
4. var server = app.listen(3000, function() {
5.   console.log("ExpressJS server running on port 3000");
6.
7.   app.get("/", function(req, res) {
8.     res.send(req.query);
9.   });
10. });

// URL => http://localhost:3000/?foo=apple&bar=orange
// Result => {foo: "apple", bar: "orange"}
```

Request နဲ့အတူပါဝင်လာတဲ့ Form Data တွေကို စီမံလိုရင်တော့ `body-parser` ဆိတဲ့ Module တစ်ခုကို

အသုံးပြုရပါတယ်။ အရင်ဆုံး body-parser ကို အခုလို Install လုပ်ထားပေးဖို့လိုပါတယ်။

```
$ npm install body-parser
```

body-parser ၂ req.body ဆိုတဲ့ Property ထဲမှာ Form Data တွေကို JSON Object တစ်ခုအနေနဲ့ အသင့် ထည့်သွင်းပေးထားမှာဖြစ်ပါတယ်။ အခုလို ရေးသားအသုံးပြုရပါတယ်။

### JavaScript

```
1. var express = require('express');
2. var parser = require('body-parser');
3. var app = express();
4.
5. app.use(parser.urlencoded({ extended: true }));
6.
7. app.post('/', function (req, res) {
8.   res.send(req.body);
9. });
```

လိုင်းနံပါတ် (၅) မှာ app.use() Function ကိုသုံးပြီး body-parser ရဲ့ urlencoded() Function ကို အသုံးပြု မယ်လို့ ကြော်လော်မှတ်ထားပါတယ်။ Form Data တွေကို Object ပြောင်းတဲ့အခါ NodeJS Build-in querystring Module ကို အသုံးပြုလိုရင် extended: false Parameter ကိုပေးရပြီး၊ Third-party Module တစ်ခုဖြစ်တဲ့ qs ကို အသုံးပြုလိုရင်တော့ နမူနာမှာဖော်ပြထားသလို extended: true Parameter ကိုပေးရမှာဖြစ်ပါတယ်။

Request နဲ့ပါဝင်လာတဲ့ Data တွေထဲမှာ Form Data အပြင် JSON Data တွေလည်းပါဝင်တက်ပါတယ်။ တစ်နည်းအားဖြင့် Request Body ဟာ Form Data မဟုတ်ပဲ JSON Data ဖြစ်နေနိုင်တဲ့သော်ဖြစ်ပါတယ်။ Request Body ဟာ JSON Data ဖြစ်နေရင်လည်း အလိုအလျောက်ရယူပြီး req.body ထဲမှာ ထည့်သွင်းပေးထားစေနိုင်အတွက် အောက်ပါနမူနာရဲ့ လိုင်းနံပါတ် (၅) မှာဖော်ပြထားသလို ဖြည့်စွက်ကြောပေးရမှာ ဖြစ်ပါတယ်။

### JavaScript

```
1. var express = require('express');
2. var parser = require('body-parser');
3. var app = express();
4.
5. app.use(parser.json());
6. app.use(parser.urlencoded({ extended: true }));
7.
8. app.post('/', function (req, res) {
9.   res.send(req.body);
10.});
```

Request နဲ့အတူ ပါဝင်လာတဲ့ ဖိုင်တွေကို လက်ခံစီမံဖို့အတွက် **multer** လိုခေါ်တဲ့ Module တစ်ခုကို အသုံးပြု ရပါ တယ်။ NPM နဲ့ အခုလုံး Install ပြုလုပ်နိုင်ပါတယ်။

```
$ npm install multer
```

အောက်ပါနမူနာရဲ့ လိုင်းနံပါတ် (၅) မှာဖော်ပြထားသလို multer Module ကို အသုံးပြုမယ့်အကြောင်း ကြော်လာရာမှာ Request နဲ့အတူ ပါဝင်လာတဲ့ ဖိုင်တွေကို သိမ်းဆည်းထားပေးရမယ့် Directory Location ကို တစ်ခါတည်း သတ်မှတ် ပေးရပါတယ်။

### JavaScript

```
1. var express = require("express");
2. var multer = require("multer");
3. var app = express();
4.
5. app.use(multer({ dest: './upload/' }));
6.
7. var server = app.listen(3000, function() {
8.
9.     app.post("/", function(req, res) {
10.         res.send(req.files);
11.     });
12.
13.});
```

နမူနာအရ Request နဲ့အတူ လက်ခံရရှိတဲ့ဖိုင်တွေကို upload ဆိုတဲ့ Directory တစ်ခုနဲ့ စုစုပေါင်းထားမှာဖြစ်ပါ တယ်။ ExpressJS က req.files Property နဲ့လည်း ဖိုင် Information တွေကို အသင့်သုံးလို့ရအောင် သိမ်းထားပေး ပါသေးတယ်။ ဖိုင်တစ်ခုကိုလက်ခံရရှိတဲ့အခါ req.files Property မှာရှိနေမယ့် အချက် အလက်တွေက အခုလုံးပုံစံ ဖြစ်မှုပါ။

### JSON

```
{
  "name": {
    "fieldname": "file",
    "originalname": "express.js",
    "name": "4f4ac9f6ed1ba0e8515f6144a6efb183.js",
    "encoding": "7bit",
    "mimetype": "application/javascript",
    "path": "upload/4f4ac9f6ed1ba0e8515f6144a6efb183.js",
    "extension": "js",
    "size": 471,
    "truncated": false,
    "buffer": null
  }
}
```

ဖိုင်ရဲမှုလအမည်၊ လက်ရှိသိမ်းဆည်းထားတဲ့ Path, ဖိုင် Size, ဖိုင် Type စတဲ့အချက်အလက်တွေကို req.files ထဲမှာ စုစည်းပေးထားခြင်းပဲဖြစ်ပါတယ်။

**Professional Web Developer ရဲ အခန်း (၈)** မှာ PHP နဲ့ Request Data တွေ Upload File တွေ စိမ့်ပုံကို ဖော်ပြထားပါတယ်။ Language မတူနဲ့အတွက် ရေးတုံးကွဲသွားပေမယ့် PHP ရော ExpressJS ကပါ HTTP သတ်မှတ် ချက်များအတိုင်း အလုပ်လုပ်နေခြင်းဖြစ်တဲ့အတွက် အခြေခံသဘောသဘာဝကတော့ အတူတူပဲဖြစ်ပါတယ်။ PHP က Request Data တွေ ကို \$\_GET, \$\_POST စတဲ့ Variable တွေနဲ့လက်ခံပေးပြီး ExpressJS က req.body နဲ့ လက်ခံထားပေးခြင်းဖြစ်ပါတယ်။ PHP က Request နဲ့အတူပါဝင်လာတဲ့ ဖိုင် Information တွေ ကို \$\_FILES Variable နဲ့ လက်ခံထားပေးသလို ExpressJS ကလည်း req.files နဲ့ လက်ခံထားပေးခြင်းပဲဖြစ်ပါတယ်။

အလားတူပဲ PHP က Request နဲ့အတူ ပါဝင်လာတဲ့ Cookie Data တွေကို \$\_COOKIE Variable နဲ့လက်ခံထားပေးသလို ExpressJS က req.cookies နဲ့ လက်ခံထားပေးမှာ ဖြစ်ပါတယ်။ ဒီလိုလက်ခံပေးနိုင်ဖို့အတွက် cookie-parser လိုခေါ်တဲ့ Module တစ်ခုကိုတော့ အသုံးပြုပေးရပါတယ်။ cookie-parser ကို အခုလို Install လုပ်နိုင်ပါတယ်။

```
$ npm install cookie-parser
```

ရေးသားအသုံးပြုပုံ Code နှမူနာက ဒီလိုပါ -

### JavaScript

```
1. var express = require("express");
2. var cookie = require("cookie-parser");
3. var app = express();
4.
5. app.use(cookie());
6.
7. var server = app.listen(3000, function() {
8.
9.     app.get("/", function(req, res) {
10.         res.send(req.cookies);
11.     });
12.
13.});
```

ထုံးစုံအတိုင်း cookie-parser ကိုအသုံးပြုမယ့်အကြောင်း app.use() နဲ့လိုင်းနဲ့ပါတ် (၅) မှာကြော်ပေးထားပါတယ်။ အဲဒီလိုကြော်ပေးလိုက်တာနဲ့ ExpressJS က Request နဲ့အတူပါဝင်လာတဲ့ Cookie Data တွေကို req.cookies ထဲမှာ JSON Object အနေနဲ့ လက်ခံထားပေးမှာပဲဖြစ်ပါတယ်။

Session Data တွေဟာ Request နဲ့အတူ ပါဝင်လာတဲ့ Data တော့မဟုတ်ပါဘူး။ ဒါပေမယ့် ExpressJS မှာ Session Data တွေကို စိမ့်ဖို့အတွက် req.session Property ကိုသုံးပါ။ express-session လိုခေါ်တဲ့

Module နဲ့ တဲ့ သုတေသန လိပါတယ်။ express-session ကို အခုံလို အဲလို Install လုပ်နိုင်ပါတယ်။

```
$ npm install express-session
```

Session တွေစီမံပုံးစီဖော်ပြပေးလိုက်ပါတယ်။

### JavaScript

```
1. var express = require("express");
2. var session = require("express-session");
3. var app = express();
4.
5. app.use(session({
6.   secret: 'secret'
7. }));
8.
9. var server = app.listen(3000, function() {
10.   console.log("ExpressJS server running on port 3000");
11.
12.   app.get("/", function(req, res) {
13.     if(req.session.view) {
14.       req.session.view += 1;
15.     } else {
16.       req.session.view = 1;
17.     }
18.
19.     res.send(req.session);
20.   });
21.
22. });
```

app.use() နဲ့ express-session ကို အသုံးပြုမယ့်အကြောင်း ကြော်လာမှာ secret Option ကို မဖြစ်မနေထည့် သွင်းပေးရပါတယ်။ Value အနေနဲ့နှစ်သက်ရာတန်ဖိုးကို ပေးနိုင်ပါတယ်။ express-session က အဲဒီ Secret Value ကို သုံးပြီး Session Data တွေကို Encrypt လုပ်ပေးသွားမှာဖြစ်ပါတယ်။ နမူနာလိုင်းနဲ့ပါတ် (၁၃) မှာ req.session.view ရှိမရှိစိစစ်ပြီး ရှိရင်တစ်တိုးပေးပါတယ်။ မရှိရင် 1 လို့ သတ်မှတ်ပေးထားပါတယ်။ ဒီနည်းနဲ့ Request တစ်ကြိမ်လုပ်တိုင်း view Session ရဲ့ တန်ဖိုး တစ်တိုးတိုးသွားမှာဖြစ်ပါတယ်။ တန်ဖိုးအားဖြင့် req.session ကို Session Data တွေရယူဖို့သာမက၊ Session တန်ဖိုးတွေ သတ်မှတ်ခြင်း၊ ပြပိုင်ခြင်း စတဲ့ ကိစ္စတွေအတွက်ပါ အသုံးပြုနိုင် ခြင်းဖြစ်ပါတယ်။

Session နဲ့ပက်သက်ရင် တစ်ခုသတိပြုရမှာရှိပါတယ်။ ExpressJS က Session Data တွေကို Memory ပေါ်မှာပဲ သိမ်းပေးပါတယ်။ ဒါကြောင့် Production အတွက် ဒီအတိုင်းသုံးလို့မရပါဘူး။ Request တွေများလာတာနဲ့အမှု Memory ပေါ်မှာ Session Data တွေများလာမှာဖြစ်တဲ့ အတွက် Memory Leak ဖြစ်သွားနိုင်ပါတယ်။ Development အဆင့်မှာ စမ်းသပ်ယုံအတွက်သာ ဒီအတိုင်းအသုံးပြုသင့်ပါတယ်။ လက်တွေမှာ Memory ပေါ်မှာသိမ်းမယ့်အစား Redis တို့ MongoDB တို့လို Database တွေနဲ့ Session Data တွေကိုသိမ်းပေးဖို့ လိပါတယ်။ ဒါမှုမဟုတ်ရင်လည်း PHP က ဒီမံသလို Session Data တွေကို ဖိုင်အနေနဲ့ သိမ်းပေးဖို့လိပါတယ်။ ဒီအတွက်

connect-redis, connect-mongo, session-file-store စတဲ့ Module တွေကို အသုံးပြန်ပါတယ်။ အသေးစိတ်ကို အောက်ပါလိပ်စာမှာ ဆက်လက်လေ့ လာနိုင်ပါတယ်။

<https://github.com/expressjs/session>

နောက်ဆုံးအနေနဲ့ Request တွေစိမ်ရာမှာ ဖြည့်စက်မှတ်သားသင့်တဲ့ Property နှစ်ခုကတော့ req.ip နဲ့ req.xhr တို့ပဲ ဖြစ်ပါတယ်။ ExpressJS က req.ip Property ထဲမှာ Request ပြုလုပ်သူ Client ရဲ့ IP Address ကို သိမ်းဆည်းထားပြီး၊ အကယ်၍ ပြုလုပ်လာတဲ့ Request က Ajax Request ဆိုရင် req.xhr ထဲမှာ true တန်ဖိုးကို သိမ်းဆည်းထားပေးမှာ ဖြစ်ပါတယ်။

## 12.4 – Response Handling

Response တွေ ပြန်လည်ပေးပို့ရာမှာ အသုံးပြုရတဲ့ res.send(), res.sendFile(), res.render(), res.download(), res.json(), res.jsonp() စတဲ့ Function တွေအသုံးပြုပဲ ကို ဖော်ပြခဲ့ပြီးဖြစ်ပါတယ်။ အခုံဆက်လက်ပြီး Response Header တွေ စီမံဖို့အတွက် အသုံးပြုရတဲ့ Function တွေကို ဖော်ပြပေးပါမယ်။

res.send() နဲ့ res.sendFile() Function တွေဟာ Response Body ကို ပေးပို့တဲ့ Function တွေဖြစ်တဲ့ အတွက် သူချဉ်းသက်သက် အသုံးမပြုသင့်ပါဘူး။ res.status() နဲ့ တဲ့ဖက်အသုံးပြုသင့်ပါတယ်။ ဥပမာ -

### JavaScript

```
res.status(200).send("Response Body");
res.status(404).sendFile("/path/to/not-found.png");
```

ဒီတော့မှ Response Body မပေးဖို့ခင် Status Code ကို အရင်ပေးပို့သွားတဲ့အတွက် ပိုပြီးဖြည့်စုံစနစ်ကျသွားမှာ ပဲဖြစ်ပါတယ်။ Response Body မပေးပို့တော့ပဲ Status Code ချဉ်းသက်သက် ပေးပို့လိုတဲ့အခါ res.sendStatus() ကို သုံးနိုင်ပါတယ်။ res.sendStatus() က ပေးလိုက်တဲ့ Status Code ပေါ်မှတည်ပြီး သင့်တော်တဲ့ Status Message ကို Response Body အဖြစ် အလိုအလျောက် တဲ့ဖက်ပေးပို့သွားမှာဖြစ်ပါတယ်။ ဥပမာ -

### JavaScript

```
res.sendStatus(200);           // Response Body => OK
res.sendStatus(403);           // Response Body => Forbidden
res.sendStatus(600);           // Response Body => 600
```

တစ်ကယ်မရှိတဲ့ Status Code တစ်ခုကိုပေးပို့ရင်တော့ Response Body အတွက်လည်း အဲဒီ Code ကိုပဲ ပေးပို့သွားမှာ ဖြစ်ပါတယ်။ Response Header အပြည့်အစုံ သတ်မှတ်လိုရင်တော့ res.set() Function ကို သုံးနိုင်ပါတယ်။

## JavaScript

```
res.set({
  'Content-Type': 'text/plain',
  'Content-Length': '123'
});
```

လက်ရှိ Response Header မှာ နောက်ထပ်တန်ဖိုးတွေ ထပ်တိုးလိုရင်တော့ `res.append()` ကိုသုံးရပါတယ်။ ဥပမာ

## JavaScript

```
res.append('Content-Encoding', 'gzip');
```

Respond နဲ့အတူ Download လုပ်ရမယ့်ဖိုင်တွေ တဲ့ဖက်လိုတဲ့အခါ Content-Disposition: attachment ဆိုတဲ့ Header ကို အသုံးပြုရပါတယ်။ Content-Disposition Header ကို အလိုအလျောက်သတ်မှတ်ပေးစေလိုရင် `res.attachment()` ကို သုံးနိုင်ပါတယ်။ Response နဲ့အတူတဲ့ဖက်ပေးပို့လိုတဲ့ ဖိုင်တည်ရှိရာ Path ကို Parameter အနေနဲ့ပေးရပါတယ်။ ဥပမာ –

## JavaScript

```
res.attachment('path/to/logo.png');

// => Content-Disposition: attachment; filename="logo.png"
// => Content-Type: image/png
```

`res.attachment()` ၏ Content-Disposition Header ကိုသာမက သင့်တော်တဲ့ Content-Type Header ကို ပါ တဲ့ဖက်သတ်မှတ်ပေးသွားမှာပဲဖြစ်ပါတယ်။ Content-Type Header သတ်မှတ်ဖို့ အတွက်လည်း ကိုယ်တိုင် သတ်မှတ်နေမယ့်အစား `res.type()` ကို သုံးနိုင်ပါ တယ်။ ဥပမာ –

## JavaScript

```
res.type('html');           // => Content-Type: text/html
res.type('json');           // => Content-Type: application/json
```

Cookie တန်ဖိုးတွေကို Response Header မှာ တဲ့ဖက်ပေးလိုက်ခြင်းအားဖြင့် Cookie တန်ဖိုးတွေ သတ်မှတ်လို တဲ့ အခါ `res.cookie()` ကိုသုံးနိုင်ပါတယ်။ Cookie Name, Value နဲ့ Expire Time တို့ကိုပေးရပါတယ်။

## JavaScript

```
res.cookie('foo', 'Apple', { expires: new Date().now() + 3600 });
res.cookie('bar', 'Orange', { maxAge: 3600 });
```

Expire Time ကို expires (သို့မဟုတ်) maxAge Option နဲ့ သတ်မှတ်နိုင်ခြင်းဖြစ်ပါတယ်။ နူမူနာမှာ စက်နှင့် ပေါင်း ၃၆၀၀ လို့သတ်မှတ်ထားတဲ့အတွက် Cookie နှစ်ခုလုံးဟာ သက်တမ်း (၁) နာရီကြာသာ တည်ရှိနေမှာ ဖြစ်ပါတယ်။

Cookie တွေပယ်ဖျက်စေလိုရင်တော့ `res.clearCookie()` ကိုသုံးနိုင်ပါတယ်။ ပယ်ဖျက်လိုတဲ့ Cookie အမည်ကို Parameter အနေနဲ့ ပေးရပါတယ်။ ဥပမာ -

#### JavaScript

```
res.clearCookie("foo");
```

နောက်ဆုံးတစ်ခုအနေနဲ့ ဖြည့်စွက်မှတ်သားရမှာကတော့ Redirect ပဲဖြစ်ပါတယ်။ URL တစ်ခုကို အလိုအလျောက် Redirect လုပ်သွားစေလိုတဲ့အခါ `res.redirect()` ကိုသုံးနိုင်ပါတယ်။ ဥပမာ -

#### JavaScript

```
app.get("/about", function(req, res) {
  res.redirect("/home/about-us");
});
```

နူမူနာအရ /about URL နဲ့ပြုလုပ်လာတဲ့ Request တစ်ခုကို လက်ခံရရှိတဲ့အခါ /home/about-us ကို အလိုအလျောက် Redirect ပြုလုပ်ပေးသွားမှာပဲဖြစ်ပါတယ်။

## 12.5 – CRUD Service Example

ExpressJS ဟာ အခြား Full-scaled Framework တွေလို့ MVC Pattern တွေ၊ Database Abstraction တွေ Validation Helper တွေ စတဲ့လုပ်ဆောင်ချက်တွေ တစ်ခါတည်း Build-in မပါဝင်ပါဘူး။ လိုချင်တဲ့ လုပ်ဆောင်ချက်တွေ ကို NPM Module တွေအနေနဲ့ ထည့်သွင်းအသုံးပြုရမှာဖြစ်ပါတယ်။ ဒါကြောင့် MongoDB လို့ Database မျိုးနဲ့ ချိတ်ဆက်အလုပ်လုပ်လိုရင်လည်း နှစ်သက်ရာ MongoDB Module ကို ရွေးချယ်ထည့်သွင်းပြီး အဲဒီ Module က ပေးထားတဲ့ လုပ်ဆောင်ချက်တွေကိုသာ အသုံးပြုရမှာဖြစ်ပါမယ်။ နူမူနာအနေနဲ့ mongojs Module ကိုအသုံးပြုပြီး MongoDB Database မှာ Create, Read, Update, Delete (CRUD) လုပ်ဆောင်ချက်ကို ဆောင်ရွက်ပေးနိုင်တဲ့ Web Service တစ်ခု တည်ဆောက်ပုံကို ဖော်ပြပေးလိုပါတယ်။

ပထမဆုံးအနေနဲ့ Project Directory တစ်ခုတည်ဆောက်ပြီး express, body-parser နဲ့ mongojs စိုးကို Install လုပ်ပါမယ်။

```
$ npm install express body-parser mongojs
```

ပြီးတဲ့အခါ လိုအပ်တဲ့ Directory Structure ရအောင် ဆက်လက်တည်ဆောက်ရပါမယ်။

```

myApp
├── node_modules
│   ├── body-parser
│   ├── express
│   └── mongojs
├── static
│   ├── about.html
│   ├── contact.html
│   └── index.html
└── api.js
└── index.js

```

node\_modules Directory ထဲမှာ NPM နဲ့ Install လုပ်ထားတဲ့ Module တွေရှိနေမှာဖြစ်ပြီ။ static Directory ထဲမှာ Static HTML ဖိုင်တွေ ရှိနေမှာဖြစ်ပါတယ်။ Project Root Folder ထဲမှာတော့ api.js နဲ့ index.js ဆိတဲ့ ဖိုင်နှစ်ခု ရှိနေမှာဖြစ်ပါတယ်။ index.js ဟာ ပင်မ App ဖြစ်ပါတယ်။ api.js ကတော့ Database နဲ့ချိတ်ဆက်ပြီး CRUD လုပ်ဆောင်ချက်များ ဆောင်ရွက်ပေးတဲ့ Sub App တစ်ခုဖြစ်ပါတယ်။

ဆက်လက်ပြီး api.js ထဲမှာ MongoDB နဲ့ချိတ်ဆက်ပြီး CRUD လုပ်နောက်တွေဆောင်ရွက်ပေးတဲ့ လုပ်ဆောင်ချက်ကို အခုလို ရေးသားထည့်သွင်းပါမယ်။

## JavaScript

```

1. var express = require("express");
2. var mongojs = require("mongojs");
3. var router = express.Router();
4.
5. var db = mongojs('username:password@localhost/mydb', ['users']);
6.
7. router.get("/users", function(req, res) {
8.     db.users.find(function(err, users) {
9.         res.status(200).json(users);
10.    });
11. });
12.
13. router.post("/users", function(req, res) {
14.     db.users.insert(req.body, function(err, result) {
15.         res.status(200).send(result._id);
16.    });
17. });
18.
19. router.get("/users/:id", function(req, res) {
20.     var id = req.params.id;
21.     db.users.find({ _id: mongojs.ObjectId(id) }, function(err, user) {
22.         if(user) res.status(200).json(user);
23.         else res.status(404).json({msg: "Not Found"});
24.    });
25. });
26.
27. router.put("/users/:id", function(req, res) {
28.     var id = mongojs.ObjectId(req.params.id);
29.     var data = req.body;
30.

```

```

31.     data._id = id;
32.
33.     db.users.save(data, function(err, result){
34.         res.status(200).send(result.nModified);
35.     });
36. });
37.
38. router.delete("/users/:id", function(req, res) {
39.     var id = mongojs.ObjectId(req.params.id);
40.
41.     db.users.remove({ _id: id }, function(err, result) {
42.         res.status(200).send(result.nRemoved);
43.     });
44. });
45.
46. module.exports = router;

```

အောက်ပါ Request Handler (၅) မျိုးကို ရေးသားသတ်မှတ်ထားခြင်း ဖြစ်ပါတယ်။

- **GET /users** (User စာရင်းရယူရန်)
- **POST /users** (User အသစ်ထည့်သွင်းရန်)
- **GET /users/:id** (:id နှင့်ကိုက်ညီသည့် User ၏ အချက်အလက်များအားရယူရန်)
- **PUT /users/:id** (:id နှင့်ကိုက်ညီသည့် User ၏ အချက်အလက်များပြင်ဆင်ရန်)
- **DELETE /users/:id** (:id နှင့်ကိုက်ညီသည့် User အားပယ်ဖျက်ရန်)

ဆက်လက်ပြီး index.js မှာ အခုလို ရေးသားပါမယ်။

### JavaScript

```

1. var express = require("express");
2. var parser = require("body-parser");
3. var api = require("./api.js");
4. var app = express();
5.
6. app.use(parser.urlencoded({ extended: true }));
7.
8. app.use(express.static("./static"));
9. app.use("/api", api);
10.
11. var server = app.listen(3000, function() {
12.     console.log("ExpressJS server running on port 3000");
13. });

```

လိုင်းနံပါတ် (၃) မှာ api.js ကို require() လုပ်ထားတာကို သတိပြုပါ။ ထူးခြားချက်အနေနဲ့ လိုင်းနံပါတ် (၈) မှာ ပင်မ App အတွက် static Directory ထဲက ဖိုင်တွေကို အသုံးပြုပေးဖို့ သတ်မှတ်ထားပြီး လိုင်းနံပါတ် (၉) မှာ api.js ကို /api Sub App အနေနဲ့ အသုံးပြုပေးဖို့ သတ်မှတ်ထားပါတယ်။ ဒါကြောင့် ဒါ App က Request တွေကို အခုလိုပုံစံ လက်ခံစီမံပေးသွားမှာဖြစ်ပါတယ် –

```

GET localhost:3000/           → static/index.html
GET localhost:3000/about.html  → static/about.html
GET localhost:3000/contact.html → static/contact.html
GET localhost:3000/api/users  → api.js GET /users Handler
GET localhost:3000/api/users/:id → api.js GET /users/:id Handler
PUT localhost:3000/api/users/:id → api.js PUT /users/:id Handler

```

လိုအပ်တဲ့ Code တွေရေးသားပြီးပြုဖြစ်လို့ အောက်ပါ API URL များကို အသုံးပြုပြီးစစ်ဆေးသပ်နိုင်ပြီဖြစ်ပါတယ်။

- **GET** /api/users (User စာရင်းရယူခြင်း)
- **POST** /api/users (User အသစ်ထည့်သွင်းခြင်း)
- **GET** /api/users/:id (:id နှင့်ကိုက်ညီသည့် User ၏ အချက်အလက်များအားရယူခြင်း)
- **PUT** /api/users/:id (:id နှင့်ကိုက်ညီသည့် User ၏ အချက်အလက်များပြင်ဆင်ခြင်း)
- **DELETE** /api/users/:id (:id နှင့်ကိုက်ညီသည့် User အားပယ်ဖျက်ခြင်း)

GET Request တွေကို URL သက်သက်နဲ့ စစ်ဆေးသပ်နိုင်ပေမယ့် POST, PUT, DELETE စတဲ့ Request တွေကိုတော့ URL သက်သက်နဲ့ စစ်ဆေးလို့မရတော့ပါဘူး။ ဒီလို Request တွေပြုလုပ်နိုင်အောင် စီစဉ်ရေးသားထားတဲ့ Client App တွေကနေတစ်ဆင့် စမ်းဆပ်ရမှုဖြစ်ပါတယ်။ API လုပ်ဆောင်ချက်မှန်ကန်မှုရှိမရှိ စစ်ဆေးယုံအတွက် Client App တွေရေးနေ ဖို့မလိုပါဘူး။ cURL လိုပေါ်တဲ့ Command Line Tool ကို အသုံးပြုနိုင်ပါတယ်။ တနည်းအားဖြင့် cURL ဟာ HTTP Client တစ်ခုဖြစ်ပါတယ်။ ဒါကြောင့် Client App တစ်ခု တည်ဆောက်မနေတော့ပဲ cURL နဲ့ပဲ စစ်ဆေးဖြည့်ပါမယ်။ cURL ကို Ubuntu မှာ အခုလို Install လုပ်နိုင်ပါတယ်။

```
$ sudo apt-get install curl
```

Windows အတွက် Installer ကိုတော့ အောက်ပါလိမ့်စာမှာ Download ရယူနိုင်ပါတယ်။

<http://curl.haxx.se/download.html>

cURL ကို Install လုပ်ပြီးပြုဆိုရင် API URL တစ်ခုချင်းကို အခုလို စစ်ဆေးဖြည့်နိုင်ပါပြီ။

```
$ curl -X GET http://localhost:3000/api/users
[{"_id": "553b7f726ab0b99f0664b180", "name": "Jonh Doe", "role": "Developer"}, {"_id": "553b7f7f6ab0b99f0664b181", "name": "Jame Smith", "role": "Tester"}]
```

/api/users URL ကို GET Method နဲ့ Request လုပ်ကြည့်လိုက်ခြင်းဖြစ်ပါတယ်။ /api/users ကို GET Method နဲ့ Request လုပ်လာခဲ့ရင် တာဝန်ယူအလုပ်လုပ်ရမယ့် Handler ကို အခုလိုရေးသားခဲ့ပါတယ်။

```

7. router.get("/users", function(req, res) {
8.   db.users.find(function(err, users) {
9.     res.status(200).json(users);
10.   });
11. });

```

db.users.find() နဲ့ users Collection ထဲကုပ္ပါယ့် Document တွေကို ရယူပြီး res.json() နဲ့ ပြန်ပေးထားတဲ့အတွက် နမူနာမှာ users Collection ထဲမှာ လက်ရှိ ရှိနေတဲ့ Document တွေကို JSON Array အနေနဲ့ ဖော်ပြန်ခြင်းပဲ ဖြစ်ပါတယ်။ နောက်တစ်ခု ထပ်မံစမ်းသပ်ကြည့်ပါမယ်။

```

$ curl -X POST http://localhost:3000/api/users -d
"name=Bob&role=Tester"

"553bd0d215349a0864c5df60"

```

/api/users ကို POST Method နဲ့ Request လုပ်ပြီး name=Bob&role=Tester ဆိုတဲ့ Data ကိုပါ တွဲဖက်ပေးလိုက်ခြင်းဖြစ်ပါတယ်။ /api/users ကို POST နဲ့ Request ပြုလုပ်လာခဲ့ရင် တာဝန်ယူအလုပ်လုပ်မယ့် Handler ကို အခုလို ရေးသားထားပါတယ်။

```

13. router.post("/users", function(req, res) {
14.   db.users.insert(req.body, function(err, result) {
15.     res.status(200).send(result._id);
16.   });
17. });

```

db.users.insert() နဲ့ req.body အဖြစ်ပါဝင်လာတဲ့ Data ကို users Collection ထဲကို ထည့်သွင်းခိုင်းလိုက် ခြင်းဖြစ်ပါတယ်။ ထည့်သွင်းလိုက်တဲ့အခါ ရရှိလာတဲ့ Generated ID ကို res.send() နဲ့ Response ပြန်ပေးဖို့ သတ်မှတ်ထားတဲ့ အတွက် နမူနာမှာ Generated ID ကို ထွေမြင်ရခြင်း ဖြစ်ပါတယ်။ အခုနေ /api/users ကို GET Request နဲ့ ပြန်စမ်းကြည့်ရင် Document တစ်ခုတိုးနေတာကို အခုလို ထွေရမှာ ဖြစ်ပါတယ်။

```

$ curl -X GET http://localhost:3000/api/users

[{"_id": "553b7f726ab0b99f0664b180", "name": "John Doe", "role": "Developer"}, {"_id": "553b7f7f6ab0b99f0664b181", "name": "Jame Smith", "role": "Tester"}, {"name": "Bob", "role": "Tester", "_id": "553bd0d215349a0864c5df60"}]

```

ဆက်လက်ပြီး GET /api/users/:id ကို စမ်းသပ်ကြည့်ပါမယ်။

```
$ curl -X GET http://localhost:3000/api/users/553bd0d215349a0864c5df60
[{"name": "Bob", "role": "Tester", "_id": "553bd0d215349a0864c5df60"}]
```

/api/users/:id ကို GET နဲ့ Request ပြုလုပ်လိုက်တဲ့အခါ :id နဲ့ကိုက်ညီတဲ့ Document တစ်ခုကိုပဲ ပြန်လည် ရရှိတာကို တွေ့ရမှာဖြစ်ပါတယ်။ PUT နဲ့ စမ်းသပ်ကြည့်ပါ၍မယ်။

```
$ curl -X PUT http://localhost:3000/api/users/553bd0d215349a0864c5df60
-d "name=Alice&role=Tester"

$ curl -X GET http://localhost:3000/api/users/553bd0d215349a0864c5df60
[{"name": "Alice", "role": "Tester", "_id": "553bd0d215349a0864c5df60"}]
```

PUT Method ကိုသုံးပြီး စောင့်သွေးသွေးတဲ့ Bob ဆိုတဲ့ User အမည်ကို Alice လို့ Update ပြုလုပ် လိုက်ခြင်း ပဲဖြစ်ပါတယ်။ နောက်ဆုံးတစ်ခုဖြစ်တဲ့ DELETE Method ကို စမ်းသပ်ကြည့်ပါ၍မယ်။

```
$ curl -X DELETE http://localhost:3000/api/users/553bd0d215349a0864c5df60
$ curl -X GET http://localhost:3000/api/users/
[{"_id": "553b7f726ab0b99f0664b180", "name": "Jonh Doe", "role": "Developer"}, {"_id": "553b7f7f6ab0b99f0664b181", "name": "Jame Smith", "role": "Tester"}]
```

/api/users/:id ကို DELETE Method နဲ့ Request လုပ်လိုက်တဲ့အတွက် သက်ဆိုင်ရာ Document ကို ပယ မျက် ပေးသွားခြင်းဖြစ်ပါတယ်။ /api/users ကို GET နဲ့ နောက်တစ်ကြိမ် Request လုပ်ကြည့်တဲ့အခါမှာ ကျွန်ုတ် တို့စမ်းသပ် ပယဖျက်ထားတဲ့ Document ပါဝင်လာခြင်း မရှိတော့တာကို တွေ့ရမှာပဲ ဖြစ်ပါတယ်။

ဒီနည်းနဲ့ CURL ကိုသုံးပြီး ကျွန်ုတ်တို့ဖန်တီးတည်ဆောက်ထားတဲ့ Web Service တွေ မှန်ကန်စွာအလုပ်လုပ် ခြင်း ရှိမရှိ ကို စီစစ်နိုင်မှာပဲ ဖြစ်ပါတယ်။

## 12.6 – ExpressJS Generator

ExpressJS App တစ်ခုအတွက် လိုအပ်တဲ့ Module တွေ၊ Directory Structure တွေနဲ့ Code Structure တွေ အသင့်ပါဝင်တဲ့ Boilerplate တစ်ခုကို Generate လုပ်ပေးနိုင်တဲ့ **express-generator** လိုပေါ်တဲ့ Scaffold Generator တစ်ခုရှိပါတယ်။ NPM နဲ့ အခုလို Install လုပ်ယူနိုင်ပါတယ်။

```
$ sudo npm install -g express-generator
```

Install လုပ်ပြီးတဲ့အခါ express Command ကိုသုံးပြီး ExpressJS Boilerplate ကို Generate လုပ်ယူနိုင်ပါပြီ။

```
$ express myApp

  create : myApp
  create : myApp/package.json
  create : myApp/myApp.js
  ...
  ...
  install dependencies:
    $ cd myApp && npm install

  run the myApp:
    $ DEBUG=myApp:* ./bin/www
  ...
  ...
```

Generate လုပ်ပြီး ရရှိလာတဲ့ App Director ထဲကိုသုံးပြီး npm install Command ကို Run ပေးလိုက်ရင် NPM က လိုအပ်တဲ့ Dependency Package တွေကို အလိုအလျောက် Install လုပ်ပေးသွားပါလိမ့်မယ်။ အားလုံးပြီးစီးသွားတဲ့အခါ ရရှိလာတဲ့ Directory Structure ကို လေ့လာကြည့်ရင် အခုလိုတွေရမှာ ဖြစ်ပါတယ်။

```
myApp
├── bin
│   └── www
├── node_modules
│   ├── body-parser
│   ├── cookie-parser
│   ├── debug
│   ├── express
│   ├── jade
│   ├── morgan
│   └── serve-favicon
├── public
│   ├── images
│   ├── javascripts
│   └── stylesheets
│       └── style.css
├── routes
│   ├── index.js
│   └── users.js
└── views
    ├── error.jade
    ├── index.jade
    └── layout.jade
└── app.js
└── package.json
```

body-parser, cookie-parser စတဲ့ Module တွေကိုတစ်ခါတည်း ထည့်သွင်းပေးထားတာကို တွေ့ရမှာ ဖြစ်သလို၊ Template Engine တစ်ခုဖြစ်တဲ့ jade ကိုလည်း ထည့်သွင်းပေးထားပါတယ်။ အကယ်၍ Template

Engine အနေနဲ့ jade အစား ejs ကို ထည့်သွင်းပေးစေလိုရင် -e Option ကို အသုံးပြနိုင်ပါတယ်။

```
$ express -e myApp
```

-e Option ထည့်သွင်းပေးလိုက်ရင် express-generator က jade အစား ejs ကို Template Engine အနေနဲ့ ထည့်သွင်းပေးသွားမှာဖြစ်ပါတယ်။

express-generator အစား ကျွန်ုတ်တို့ အခန်း (၈) မှာဖော်ပြခဲ့တဲ့ Yeoman ကိုလည်း ExpressJS App Boilerplate တွေ Generate လုပ်ဖို့အသုံးပြနိုင်ပါတယ်။ ဥပမာ -

```
$ sudo npm install -g generator-express
$ yo express
```

Yeoman Generator က လက်တွေ့မှာ express-generator ကိုပဲ ပြန်သုံးထားတာပါ။ ဒါပေမယ့် သူက Boilerplate နဲ့အတူ Bower Config ဖိုင်တွေ၊ Grunt Config ဖိုင်တွေကိုပါ တစ်ခါတည်း ထည့်သွင်းပေးသွားမှာ ဖြစ်တဲ့အတွက် ပိုမို ပြည့်စုံတယ်လို့ ဆိုနိုင်ပါတယ်။ မိမိနှစ်သက်ရာကို အသုံးပြနိုင်ပါတယ်။

## Conclusion

Web Application Development အတွက် အဓိက Development Stack အဖြစ် ဆယ်စုံနှစ်တစ်ခုစာ ထင်ရှားခဲ့တဲ့ နည်းပညာကတော့ **LAMP** Stack ပဲဖြစ်ပါတယ်။ Linux OS, Apache Web Server, MySQL Database, နဲ့ PHP တို့ အတဲ့လိုက် စုစုပေါင်တဲ့ Development Environment တစ်ခုဖြစ်ပါတယ်။ ကျွန်ုတ်ရဲ့ ပထမစာအပ်နှစ် အပ်ဖြစ်တဲ့ **Professional Web Developer** နဲ့ **Ubuntu** – သင့်အတွက် **Linux** တို့ဟာ LAMP Stack တစ်ခုလုံး ကို Covered ဖြစ်စေနဲ့ ရည်ရွယ်ရေးသားခဲ့တဲ့ စာအုပ်နှစ်အုပ် ဖြစ်ပါတယ်။

အခုနောက်ပိုင်းမှာတော့ JavaScript နည်းပညာတွေရဲ့ အခန်းကဏ္ဍာဟာ များစွာတိုးတက်လာခြင်းနဲ့အတူ **MEAN** လို့ ခေါ်တဲ့ Development Stack အသစ်တစ်ခုက LAMP Stack ကို အနာဂတ်မှာ အစားထိုးသွားလိမ့်မယ်ဆိုတဲ့ မျှော်မှန်းချက် တွေ ရှိလာခဲ့ပါတယ်။ MEAN ဆိုတာ MongoDB, ExpressJS, AngularJS နဲ့ NodeJS တို့ စုစုပေါင်တဲ့ Develop-ment Stack ကို ခေါ်တာပါ။

မူလက၊ ဒီစာအုပ်ရဲ့ အပိုင်း (၃) အဖြစ် MEAN Stack အကြောင်းဖော်ပြမယ်လို့ ရည်ရွယ်ထားခဲ့တာပါ။ ဒါပေမယ့် လက်တွေ့မှာ AngularJS အစား ပိုမြေးရိုးရှင်း အခြေခံကျတဲ့ BackboneJS ကိုပဲ ရွေးချယ်ဖော်ပြဖြစ်ခဲ့ပါတယ်။ ဒါကြောင့် ခေါင်းစဉ်ကို MEAN Stack လို့မတပ်တော့ပဲ Modern Development Stack လို့ပဲ တပ်ထားခဲ့တာပါ။ BackboneJS, NodeJS, MongoDB နဲ့ ExpressJS တို့ဟာ JavaScript အခြေခြား အနာဂတ် Development Stack အတွက် အရေးပါတဲ့ အစိတ်အပိုင်းတွေဖြစ်ပါတယ်။

# အပိုင်း (၅)

Building an Issue Tracking System

အကောင်းဆုံး User Experience ရရှိဖို့အတွက် User Interface

နဲ့ပက်သက်တဲ့အပိုင်းတွေကို အသေးစိတ်ကအစ ကိုယ်တိုင်လုပ်နိုင်ရင်  
ကောင်းပေမယ့်၊ Authentication အပါအဝင် အခြေခံခိုင်မာဖို့လိုတဲ့ကိစ္စ  
တွေမှာတော့ Proven ဖြစ်ပြီးသား Component တွေကို အသုံးပြု  
လိုက်တာက ကိုယ်တိုင်လုပ်နေတာထက် ပိုမိုထိရောက်ပြီး  
ပိုမိုစိတ်ချေရဖော့မှာ ဖြစ်ပါတယ်။

## **Ubuntu - သင့်အတွက် Linux (စာအုပ်)**

Ubuntu Linux အသုံးပြုနည်းအား Installation မှ စတင်၍ System Configuration နှင့် Development Environment တည်ဆောက်  
ခြင်းအထိ အဆင့်လိုက်ရေးသား ဖော်ပြထားသည့်စာအုပ်  
အောက်ပါလိပ်စာတွင် အခမဲ့ ရယူနိုင်သည်။

## အခန်း(၁၃) – Building Back-end Service

ပြီးခဲ့တဲ့အပိုင်းမှာ လေ့လာခဲ့တဲ့ BackboneJS, NodeJS, ExpressJS နဲ့ MongoDB တို့နဲ့ပက်သက်ပြီး လက်တွေ့နဲ့ မူနာ တစ်ခုရာသွားစေဖို့ရည်ရွယ်ပြီး ဒီအပိုင်းမှာ Issue Tracking System တစ်ခုကို အဲဒီနည်းပညာတွေသုံးပြီး တည်ဆောက်ပုံ နမူနာကို ဖော်ပြသွားမှာဖြစ်ပါတယ်။

**အခန်း (၄)** မှာ Service-oriented Architecture (SOA) အကြောင်းဖော်ပြုခဲ့စဉ်က ၁၁ Software တစ်ခုကို သီးခြားစီအလုပ်လုပ်နိုင်တဲ့ Service များအဖြစ်တည်ဆောက်ခြင်းအားဖြင့် ရရှိနိုင်တဲ့ အကျိုးရလဒ်တွေကို ဖော်ပြုခဲ့တာကို မှတ်မိမိုးမှာပါ။ ဒီအပိုင်းမှာ Issue Tracking System တစ်ခုကို နမူနာအနေနဲ့ ဖန်တီးရာမှာလည်း Back-end Service နဲ့ Client App တို့ကို Code တွေရောစပ်ရေးသားခြင်းမပြုပဲ သီးခြားစီတည်ဆောက်သွားမှာဖြစ်ပါတယ်။ ExpressJS နဲ့ MongoDB တို့ကိုသုံးပြီး Back-end Service တည်ဆောက်ပုံကို ဦးစွာဖော်ပြချင်ပါတယ်။

ဒီအခန်းမှာ နမူနာဖော်ပြမယ့် Code တွေဟာ လက်တွေ့ကူးယူစွမ်းသပ်လိုက စမ်းသပ်နိုင်တဲ့ Code တွေ ဖြစ်ပါတယ်။ ဒါပေမယ့် တစ်လုံးမကျွန်ကူးယူစွမ်းသပ်မယ့်အစား ရေးသားထားတဲ့ Code ကိုအရင် နားလည်အောင်လေ့လာပြီး၊ ကိုယ်တိုင် ရေးသားဖို့ကို အကြံပြုလိုပါတယ်။ ဒါကြောင့် Code နမူနာ အပြည့်အစုံကို ဖတ်ရှုနိုင်ဖို့အတွက် အောက်ပါလိပ်စာမှာ ကြိုတင် Download ရယူထားသင့်ပါတယ်။

<https://github.com/eimg/issues-api/archive/master.zip>

### 13.1 – Scaffolding

ပထမဗုဒ္ဓားဆုံးအနေနဲ့ issues-api ဆိုတဲ့အမည် (သို့မဟုတ် မိမိနစ်သက်ရာအမည်) နဲ့ Directory တစ်ခုတည်ဆောက်ပြီး အဲဒီ Directory ထဲမှာ လိုအပ်တဲ့ Package တွေကို NPM နဲ့ Install လုပ်ခြင်းအားဖြင့် စတင်ပါမယ်။

```
$ mkdir issues-api && cd issues-api
$ npm install express \
    mongojs \
    express-session \
    body-parser \
    cookie-parser \
    express-validator
    passport \
    passport-local
```

နဲ့မူနာမှာ npm install နဲ့ Package (၈) ခုကို Install လုပ်ထားပါတယ်။ ExpressJS နဲ့ MongoDB တို့ကို အသုံးပြုရေးသားမှာမို့ express နဲ့ mongojs Package တွေကို Install လုပ်ထားခြင်းပါတယ်။ express-session, body-parser နဲ့ cookie-parser တို့ကတော့ User Request တွေလက်ခံခြင်းလုပ်ငန်းနဲ့ Cookie, Session တွေ စီမံခြင်းလုပ်ငန်းအတွက်လိုအပ်လို့ ထည့်သွင်းထားရတာပါ။ User Request တွေကို Validation လဲ စစ်ချင်သေးတဲ့ အတွက် express-validator ကိုလည်း ထည့်သွင်းထားပါတယ်။ User Authentication နဲ့ Authorization လုပ်ငန်းတွေ အတွက် PassportJS လိုပေါ်တဲ့ Framework ကို အသုံးပြုချင်တဲ့ အတွက် passport နဲ့ passport-local ဆိုတဲ့ Package နှစ်ခုကို တွဲဖက် ထည့်သွင်းထားခြင်းဖြစ်ပါတယ်။ ဒီအကြောင်းကို မကြာခင်ဆက်လက်ဖော်ပြပေးပါမယ်။

ဒါ Package တွေထည့်သွင်းပြီးတဲ့ အခါ Directory Structure က အခုလိုရရှိသွားမှာဖြစ်ပါတယ်။

```
issues-api
└── node_modules
    ├── body-parser
    ├── cookie-parser
    ├── express
    ├── express-session
    ├── express-validator
    ├── mongojs
    ├── passport
    └── passport-local
```

ဒါ Structure ထဲမှာ လိုအပ်တဲ့ ဖိုင်တွေကိုအောက်ပါအတိုင်း ဆက်လက်ဖြည့်စွက်ပေးပါ။

```
issues-api
├── app.js
├── auth.js
├── config.js
└── node_modules
    └── routes
        ├── issues.js
        └── users.js
```

app.js, auth.js နဲ့ config.js တို့ကို Project Root Directory အောက်မှာ ထည့်သွင်းပြီး issues.js နဲ့ users.js တို့ကို တော့ routes ဆိုတဲ့ Directory တစ်ခုနဲ့ ဖြည့်စွက်ထည့်သွင်းထားခြင်းဖြစ်ပါတယ်။

### 13.2 – Basic Config – config.js

ဆက်လက်ပြီး config.js အတွင်းမှာ အောက်ပါအတိုင်း ရေးသားရပါမယ်။

#### JavaScript

```

1. exports.role = [
2.   "User",
3.   "Tester",
4.   "Developer",
5.   "Manager"
6. ];
7.
8. exports.status = [
9.   "New",
10.  "Assigned",
11.  "Doing",
12.  "Done",
13.  "Closed"
14. ];
15.
16. exports.type = [
17.   "Bug",
18.   "Feature",
19.   "Other"
20. ];
21.
22. exports.priority = [
23.   "Low",
24.   "High",
25.   "Urgent"
26. ];

```

role, status, type နဲ့ priority တန်ဖိုးတွေကို exports Object ရဲ့ Property တွေအနေနဲ့ ကြိုတင်သတ်မှတ်လိုက်ခြင်း ဖြစ်ပါတယ်။ အခန်း (၁၀) မှာ NodeJS အကြောင်းဖော်ပြခဲ့စဉ်က Module အဖြစ် အခြား Code တွေကနေ ရယူအသုံးပြုလိုတဲ့အခါ module.exports (သို့မဟုတ်) exports Object မှာ အသုံးပြုလိုတဲ့ Function နဲ့ Property တွေကို သတ်မှတ်ပေးရကြောင်း ဖော်ပြခဲ့ပြီး ဖြစ်ပါတယ်။

### 13.3 – Authentication and Authorization Strategies – auth.js

ဆက်လက်ပြီး အဓိကလုပ်ဆောင်ချက်များ မရေးသားမဲ့ PassportJS ကို အသုံးပြုပြီး Authentication နဲ့ပက်သက်တဲ့ အပိုင်းတွေကို အရင်ရေးသားပါမယ်။ auth.js အတွင်းမှာ အခုလိုရေးသားပေးရပါမယ်။

#### JavaScript

```

1. var mongojs = require("mongojs");
2. var crypto = require('crypto');
3. var passport = require("passport");
4. var LocalStrategy = require("passport-local").Strategy;
5.
6. var db = mongojs('mydb', ['users', 'issues']);
7.

```

```

8. passport.serializeUser(function(user, done) {
9.   done(null, user);
10. });
11.
12. passport.deserializeUser(function(user, done) {
13.   done(null, user);
14. });
15.
16. // Authentication strategy
17. passport.use(new LocalStrategy(
18.   function(username, password, done) {
19.     var pwdHash = crypto.createHash('sha1').update(password).digest("hex");
20.
21.     db.users.findOne({
22.       loginId: username,
23.       password: pwdHash
24.     }, function(err, data) {
25.       if(data)
26.         return done(null, data);
27.       else
28.         return done(null, false);
29.     });
30.   }
31. ));
32.
33. exports.login = function() {
34.   return passport.authenticate("local");
35. }
36.
37. exports.ensureAuth = function() {
38.   return function(req, res, next) {
39.     if(req.isAuthenticated())
40.       next()
41.     else res.sendStatus(401);
42.   }
43. }

```

App သို့မဟုတ် API တစ်ခုအတွက် Authentication ပိုင်းစဉ်းစားတဲ့အခါ ဘယ်လို Authentication Strategy ကို အသုံးပြုမလဲဆိတာကို အရင်းဆုံးစဉ်းစားရပါတယ်။ ပုံမှန်အားဖြင့် ရှိုးရှိုး Point to Point API Service တစ်ခု အတွက်ဆိုရင် သိပ်ပြီးရှုပ်ရှုပ်ထွေးထွေးတွေ စဉ်းစားနေစရာမလိုပဲ HTTP Basic Authentication ကို အသုံးပြုနိုင်ပါတယ်။ Point to Point ဆိတာ Client နဲ့ Server တိုက်ရိုက်ဆက်သွယ်ပြီး ကြားခဲ့ Third-party Service တွေ မပါဝင်တဲ့ အခြေအနေ ကို ဆိုလိုတာပါ။

HTTP Basic Authentication ရဲ့အလုပ်လုပ်ပုံက ရှိုးရှင်းပါတယ်။ Request တိုင်းမှာ Access Control Credential ဖြစ်တဲ့ Username နဲ့ Password ပါဝင်ဖို့လိုပြီး၊ Request နဲ့အတူပါဝင်လာတဲ့ Credential မှန်မှသာ Access လုပ်ခွင့် ပြုမှာဖြစ်ပါတယ်။ HTTP ရဲ့ Stateless သဘာဝအတိုင်းသာ အလုပ်လုပ်ပြီး၊ တစ်ကြိမ်ပေးဖူးတဲ့အတွက် နောက် Request တွေမှာ Credential ထပ်မပေးတော့ဘူးလို့ သဘောထားလို့မရတော့ပါဘူး။ Request တိုင်းမှာ Credential ပါရမှာဖြစ်သလို Service ကလည်း Credential ပါလာမှသာ Access လုပ်ခွင့်ပြုမှာပဲဖြစ်ပါတယ်။ ရှိုးရှိုး API Service တွေအတွက် သင့်တော်ပါတယ်။

နောက်အသုံးများတဲ့ Authentication Strategy ကတော့ Session Based Authentication Model ဖြစ်ပါတယ်။ သူကို Local Strategy လိုလည်း ခေါ်ပါတယ်။ အခြေခံအလုပ်လုပ်ပုံကတော့ Access Control Credential ကို တစ်ကြိမ် မှန်အောင်ပေးပြီးရင် Authentication Token ကို Session နဲ့ သိမ်းဆည်းထားမှာဖြစ်ပါတယ်။ ဒါ ကြောင့် နောက် Access လုပ်ဖို့လိုတဲ့အခါ Credential ကို ထပ်ပေးစရာမလိုတော့ပဲ Session Token ကိုကြည့်ပြီး Access လုပ်ခွင့်ပြု မဖြုတ် ဆုံးဖြတ်သွားမှာ ဖြစ်ပါတယ်။ Web App တွေမှာ အမိကအသုံးများတဲ့ Strategy ဖြစ်ပါတယ်။

နောက်တနည်းကတော့ API Key (သို့မဟုတ်) Token ကို အခြေခံတဲ့ Strategy ဖြစ်ပါတယ်။ သူလည်းပဲ လက်တွေမှာ HTTP Basic Authentication နဲ့ သိပ်မကွာပါဘူး။ HTTP Basic Authentication က Username နဲ့ Password ကို Access Control Credential အဖြစ်သုံးပြီး သူကတော့ Generate လုပ်ထားတဲ့ API Key (သို့မဟုတ်) Random Token ကို Access Control Credential အဖြစ် အသုံးပြုခြင်းဖြစ်ပါတယ်။ အခြေခံအားဖြင့်၊ ပထမတစ်ကြိမ်မှာ သတ်မှတ်ထားတဲ့ Access Control Credential တွေကို မှန်ကန်အောင်ပေးရပါတယ်။ Credential မှန်တယ်ဆိုရင် Service က Random Token တစ်ခုကို ပြန်ပေးမှာဖြစ်ပါတယ်။ နောင် Service ကို Access လုပ်ဖို့လိုတဲ့အခါ Credential ကို ထပ်ပေးနေစရာမလိုတော့ပဲ Request နဲ့အတူ Token ကို ထည့်သွင်းပေးပို့ခြင်းအားဖြင့် Access လုပ်နိုင် မှာဖြစ်ပါတယ်။

Token ဆိုတာ ပုံမှန်အားဖြင့် မှန်းဆလိုမရအောင် Random Generate လုပ်ထားတဲ့ တန်ဖိုးတစ်ခုဖြစ်ပါတယ်။ ဒါပေမယ့် အခုနောက်ပိုင်းမှာ Random မဟုတ်တော့ပဲ User Information တွေကို Token ထဲမှာ တစ်ခါတည်းထည့်ပြီး Encrypt လုပ်ပေးနိုင်တဲ့ JSON Web Tokens လို နည်းပညာမျိုးတွေလည်း ပေါ်ပေါက်လာပါတယ်။ Token က Random မဟုတ် တော့ပဲ User Information တွေပါဝင်လာတဲ့အတွက် Token ကနေရှုံးတဲ့ User Information ကိုလိုအပ်သလို အသုံးချ နိုင်မှာဖြစ်တဲ့အတွက် Authorization လုပ်ယုံသက်သက်ထက် ပိုအသုံးဝင် သွားပါတယ်။ ဒါလို Token ထဲမှာ ရှိနေတဲ့ User Information ကို မသမာသူတွေ ကြားဖြတ်ဆုံးယူလို မရစေဖို့ အတွက်တော့ Encrypt လုပ်ခြင်းအားဖြင့် ကာကွယ်ထားတက် ကြပါတယ်။

နောက်တစ်နည်းကတော့ OAuth2 Protocol ကို အသုံးပြုခြင်းဖြစ်ပါတယ်။ ဒီ Strategy ကတော့ Service က သပ်သပ်၊ Client ကသပ်သပ်၊ Authentication Provider သပ်သပ်၊ သီးခြားစီ အလုပ်လုပ်တဲ့ အခြေအနေမျိုးနဲ့ သင့်တော်ပါတယ်။ တစ်နည်းအားဖြင့်၊ Access ပြုလုပ်လိုတာက Service A ဖြစ်ပေမယ့်၊ Access လုပ်ခွင့်ပြုမပြု ဆုံးဖြတ်ပေးမယ့်သူက Service B ဖြစ်နေတယ်ဆိုရင် OAuth2 ကို အသုံးပြုသင့်ပါတယ်။ ဥပမာ – Web App တစ်ခုကို Login ဝင်နိုင်ဖို့အတွက် Account အသစ်မတည်ဆောက်ပဲ Facebook Account ကို အသုံးပြု Login ဝင် နိုင်ပုံကို တွေ့ဖူးကြမှာပါ။ အဲဒီလို အခြေ အနေမှာ အသုံးပြုလိုတဲ့ Web App အတွက် Authentication ကို သီးခြား Service ဖြစ်တဲ့ Facebook က ဆုံးဖြတ်ပေး တဲ့သဘော ဖြစ်ပါတယ်။

ကျွန်ုတ်တို့ရဲ့ နှမုနာမှာတော့၊ နောင်တစ်ချိန်မှာ Web Client နဲ့ တွဲဖက်အလုပ်လုပ်ဖို့ ရည်ရွယ်ထားတဲ့ Service တစ်ခုကို တည်ဆောက်မှာဖြစ်တဲ့အတွက် Session ကို အခြေခံတဲ့ Local Strategy ကို အသုံးပြုသွားမှာဖြစ်ပါတယ်။ ဒါကြောင့် လိုင်းနံပါတ် (၃) နဲ့ (၄) မှာ passport နဲ့ passport-local Package တွေကို အသုံးပြုဖို့ ကြော်ထားခြင်းပဲဖြစ်ပါတယ်။ အကယ်၍ Local Strategy အစား အခြား Strategy ကို အသုံးပြုလိုတယ်ဆိုရင် သက်ဆိုရာ Passport Strategy Package ကို ထည့်သွင်းအသုံးပြုနိုင်ပါတယ်။

အသုံးပြုသေးလာတဲ့ Username နဲ့ Password ကို Database ထဲမှာရှိတဲ့ Record (Document) နဲ့ တိုက်ဆိုင်စစ်ဆေးပြီး မှန်တယ်ဆိုရင် အဲဒီ User ရဲ့ အချက်အလက်တွေအားလုံးကို Session ထဲမှာ သိမ်းပေးခြင်းအားဖြင့် Authorize လုပ်ပေးမှာဖြစ်ပါတယ်။ ဒါကြောင့် လိုင်းနံပါတ် (၆) မှာ mydb လိုပေါ်တဲ့ MongoDB Database ကို mongojs နဲ့ ချိတ်ဆက်ပေးထားပါတယ်။ အကယ်၍ စာဖတ်သူရဲ့ MongoDB Database က Username, Password နဲ့ Protect လုပ်ထားရင် ဒီလိုင်းကို အခုလိုပြင်ပေးရမှာပါ။

### JavaScript

```
6. var db = mongojs('username:password@localhost/mydb', ['users', 'issues']);
```

username နဲ့ password နေရာမှာ မှန်ကန်တဲ့ Database Username နဲ့ Password တို့ကို အစားထိုးထည့်သွင်းပေးရမှာပဲဖြစ်ပါတယ်။ ဒီ Connection String ထဲမှာ users နဲ့ issues ဆိုတဲ့ Collection နှစ်ခုတို့ကို အသုံးပြုမှာဖြစ်ကြောင်း တစ်ခါတည်းထည့်သွင်းပေးထားတာကိုလည်း သတိပြုပါ။

ဆက်လက်ပြီး လိုင်းနံပါတ် (၈) မှာ passport.serializeUser() Function ကို ကြော်လာထားပါတယ်။ ဒီ Function ရဲ့ရည်ရွယ်ချက်က JSON User Data ကို Session ထဲမှာ Text အဖြစ်ပြောင်းပြီး သိမ်းဖို့အတွက် ဖြစ်ပါတယ်။ လိုင်းနံပါတ် (၁၂) က passport.deserializeUser() ကတော့ သူနဲ့ပြောင်းပြန်၏ Session ထဲက User Data ကို JSON ဖြစ်အောင် ပြန်ပြောင်းပေးဖို့အတွက် ဖြစ်ပါတယ်။ တနည်းအားဖြင့် လက်တွေအသုံးပြုတဲ့ အချိန်မှာ User Data တွေကို JSON နဲ့အသုံးပြုနိုင်ပြီး၊ Session ထဲမှာ သိမ်းတဲ့အခါ Text အနေနဲ့ သိမ်းနိုင်ဖို့ ဒီ Function နှစ်ခုကို ကြော်ပေးရခြင်းဖြစ်ပါတယ်။ Serialize လုပ်တဲ့အချိန်နဲ့ Deserialize လုပ်တဲ့အချိန်တို့မှာ ဖြည့်စွက်လုပ်ဆောင် ချက်တွေရှိရင် သက်ဆိုင်ရာ Callback Function ထဲမှာ ရေးသားသတ်မှတ်ထားနိုင်ပါတယ်။ နမူနာမှာတော့ ဘာမှဖြည့်စွက် ရေးသားထားခြင်းမရှိပဲ Serialize/Deserialize လုပ်နေးပြီးစီးကြောင်းကြော်တဲ့ done() Function ကိုပဲ အလုပ်လုပ် စေထားပါတယ်။

လိုင်းနံပါတ် (၁၇) မှာတော့ passport.use() ကိုသုံးပြီး Local Strategy ကို အသုံးပြုမှာဖြစ်ကြောင်းကြော် သတ်မှတ်ထားပါတယ်။ တစ်လက်စတည်း Strategy Definition ကိုလည်း တွဲဖက်သတ်မှတ်ပေးထားပါတယ်။ နမူနာမှာ ပေးထားတဲ့ Strategy Definition အရာ ပေးလာတဲ့ Username, Password နဲ့ ကိုက်ညီတဲ့ User ရှိမရှိ db.users.find() နဲ့ စီစဉ်ထားပါတယ်။ ကိုက်ညီတဲ့ User ရှိတယ်ဆိုရင် done() Function နဲ့အတူ ရရှိလာတဲ့ User Data တွေကို ပြန်ပေးပြီး၊ ကိုက်ညီတဲ့ User မရှိဘူးဆိုရင်တော့ done() Function နဲ့အတူ false တန်ဖိုးကို return ပြန်ပေးထားပါတယ်။

Password တွေကိုသိမ်းတဲ့အခါ SHA1 Hash လုပ်ပြီးမှ သိမ်းဖို့ရည်ရွယ်ထားတဲ့အတွက် ပေးလာတဲ့ Password ကိုလည်း SHA1 Hash လုပ်ပြီးမှ တိုက်ဆိုင်စစ်ဆေးထားတာကိုလည်း သတိပြုပါ။

အသုံးပြုမယ့် Strategy ကြော်သတ်မှတ်ပြီးတဲ့အခါ Authentication နဲ့ Authorization တို့အတွက် သတ်မှတ်ချက်တွေ ကိုဆက်လက်ရေးသားပါတယ်။ Authentication အတွက် လိုင်းနံပါတ် (၃၃) မှာ login() Function ကို ကြော် သတ်မှတ်ထားပါတယ်။ ဒီ Function က ထူးထူးခြား ဘာမှမလုပ်ပဲ passport

.authenticate() Function ကို ခေါ်ယူပေးထားခြင်းဖြစ်ပါတယ်။ ဒီလိုခေါ်ယူရာမှာ Strategy အမည်အဖြစ် local ကိုပေးလိုက်တဲ့အတွက် Request နဲ့အတူပါဝင်လာတဲ့ Username, Password ကိုအသုံးပြုပြီး Credential မှန်မမှန်စိစစ်တဲ့လုပ်ငန်းကို ကျွန်ုတ်တို့ လိုင်းနံပါတ် (၁၇) မှာ ကြော်လွှဲတဲ့ Local Strategy ကို သုံးပြီး အလုပ်လုပ်ပေးသွားမှာဖြစ်ပါတယ်။ Credential မှန်ရင် User Data တွေ ကို Session ထဲမှာ သိမ်းပေးတဲ့ လုပ်ငန်းကို ကျွန်ုတ်တို့ ကိုယ်တိုင်ရေးစရာမလိုပါဘူး။ PassportJS နဲ့ Passport Local တို့က ကျွန်ုတ်တို့ ကိုယ်စား လုပ်ပေးသွားမှာဖြစ်ပါတယ်။ တနည်းအားဖြင့် ကျွန်ုတ်တို့က Strategy Definition ကိုသာ သတ်မှတ်ပေးဖို့လိုပြီး၊ ကျွန်ုတ်တို့က PassportJS က လုပ်ပေးသွားတဲ့သဘော ဖြစ်ပါတယ်။

Authorization အတွက် လိုင်းနဲ့ပါတ် (၃၇) မှာ ensureAuth() ဆိုတဲ့ Function ကိုရေးသားပါတယ်။ လက်ရှိ Request ဟာ Login လုပ်ထားပြီး Access လုပ်ခွင့်ရှိသူ ဟုတ်မဟုတ် စိစစ်တဲ့ Function ဖြစ်ပါတယ်။ ပုံမှန် အားဖြင့် Login လုပ်ထားပြီးသူ ဟုတ်မဟုတ် Session ထဲမှာ ရှိတဲ့ User Data ကို ကြည့်ပြီးစစ်ရမှာပါ။ ဒါပေမယ့် ဒီကိစ္စကိုလည်း ကိုယ်တိုင်လုပ်စရာ မလိုပါဘူး။ PassportJS က လုပ်ပေးတဲ့ အတွက် req.isAuthenticated() Function ကို အသင့်ခေါ်ယူ အသုံးပြုပြီး Authenticated User ဟုတ်မဟုတ် စိစစ်နိုင်မှာဖြစ်ပါတယ်။ နှမူနာအရ Authenticated User ဆိုရင် Next Route ကို ဆက်သွားခွင့်ပြုတဲ့အနေနဲ့ next() Function ကို ခေါ်ယူထားပြီး Authenticated User မဟုတ်ရင်တော့ 401 Status Code ကို ပြန်ပေးဖို့ သတ်မှတ်ထားပါတယ်။

အခြေခံ Authentication နဲ့ Authorization လုပ်ဆောင်ချက်တွေကို ရရှိသွားပြီဖြစ်ပါတယ်။ ဆက်လက်ပြီး User Role နဲ့ ပက်သက်တဲ့ Authorization လုပ်ငန်းတွေ လုပ်ချင်သေးတဲ့အတွက် auth.js ထဲမှာ အခုလို ဆက်လက်ဖြည့်စွက် ပေးပါ။

## JavaScript

```
45. exports.ensureSuper = function() {
46.   return function(req, res, next) {
47.     if (req.isAuthenticated() && req.user.role === 3)
48.       next();
49.     else
50.       res.sendStatus(401);
51.   }
52. };
53.
54. exports.ensureRole = function(role) {
55.   return function(req, res, next) {
56.     if (req.isAuthenticated() && req.user.role >= role)
57.       next();
58.     else
59.       res.sendStatus(401);
60.   }
61. };
62.
63. exports.ensureOwner = function() {
64.   return function(req, res, next) {
65.     if (req.isAuthenticated() && (req.user._id === req.params.id ||
66.       req.user.role === 3))
67.       next();
68.     else
69.       res.sendStatus(401);
70.   }
71. };
72.
73. exports.ensureAdmin = function() {
74.   return function(req, res, next) {
75.     if (req.isAuthenticated() && req.user.role === 3)
76.       next();
77.     else
78.       res.sendStatus(401);
79.   }
80. };
81.
82. exports.ensureOwnerOrAdmin = function() {
83.   return function(req, res, next) {
84.     if (req.isAuthenticated() && (req.user._id === req.params.id ||
85.       req.user.role === 3))
86.       next();
87.     else
88.       res.sendStatus(401);
89.   }
90. };
91.
92. exports.ensureSuperOrAdmin = function() {
93.   return function(req, res, next) {
94.     if (req.isAuthenticated() && (req.user.role === 3))
95.       next();
96.     else
97.       res.sendStatus(401);
98.   }
99. };
100.
```

```

68.     else
69.         res.sendStatus(401);
70.     }
71.   };
72.
73. exports.ensureAssignee = function() {
74.   return function(req, res, next) {
75.     var iid = req.params.id;
76.     db.issues.findOne({ _id: mongojs.ObjectId(iid) }, function(err, data) {
77.       if(err) {
78.         res.sendStatus(401);
79.       } else {
80.         if (req.isAuthenticated() &&
81.             (req.user._id.str==(data.assignedTo && data.assignedTo.str) ||
82.             req.user.role === 3))
83.           next();
84.         else
85.           res.sendStatus(401);
86.       }
87.     });
88.   }
89. };
90.
91. exports.ensureSubmitter = function() {
92.   return function(req, res, next) {
93.     var iid = req.params.id;
94.     db.issues.findOne({ _id: mongojs.ObjectId(iid) }, function(err, data) {
95.       if(err) {
96.         res.sendStatus(401);
97.       } else {
98.         if (req.isAuthenticated() && (req.user._id == data.submittedBy ||
99.             req.user.role === 3))
100.           next();
101.         else
102.           res.sendStatus(401);
103.       }
104.     });
105.   }
106. };

```

PassportJS ၏ Session ထဲမှာ User Data တွေကို ထည့်သွင်းပေးထားယုံသာမက၊ လိုအပ်ရင် အသင့်အသုံးပြုနိုင်အောင် req.user ဆိုတဲ့ Property နဲ့လည်း အသင့်ဖြင့်ဆင်ထားပေးပါတယ်။ ဒါကြောင့် လိုင်းနံပါတ် (၄၅) မှာ req.user ကို အသုံးချဖြီး ensureSuper() Function ကို ကြော်လှုပ်ထားပါတယ်။ သတ်မှတ် ချက်အရ User ဟာ Authenticated User ဖြစ်ယုံသာမက role တန်ဖိုးလည်း 3 ဖြစ်ရမယ်လို့ သတ်မှတ်ထား ခြင်းဖြစ်ပါတယ်။ config.js မှာ ပြန်လည် လေ့လာကြည့်ရင် Role အနေနဲ့ User, Tester, Developer, Manager လို့ အဆင့်ဆင့် သတ်မှတ်ထားပြီး အမြင့်ဆုံး Manager ရဲ့ Index တန်ဖိုးက 3 ဖြစ်ပါတယ်။ ဒါကြောင့် ensureSuper() သတ်မှတ် ချက်အရဆုံး Manager Level User မှာသာ Access လုပ်ခွင့်ရှိမှာဖြစ်ပါတယ်။ ဒါ ဟာ သတ်မှတ်ချက်သက်သက်သာ ဖြစ်ပါတယ်။ ဒီသတ်မှတ်ချက်ကို ဘယ်နေရာမှာသုံးမလဲဆိုတာကို မဆုံးဖြတ်ရသေးပါဘူး။

ဆက်လက်ပြီး (၅၄) မှာ ensureRole() Function ကို ကြော်လာတ်မှတ်ထားပါတယ်။ ဒါ Function ရဲ့ ရည်ရွယ် ချက်ကတော့ Access Right ကို Role နဲ့တွေ့ဖက်သတ်မှတ်ထားဖို့ဖြစ်ပါတယ်။ ဥပမာ – ensureRole(2) လို့ပြော လိုက်ရင် User Role Level တန်ဖိုး 2 ကိုသာ Access လုပ်ခွင့်ပြုမှာဖြစ်ပါတယ်။ သတိပြုသင့်တာကတော့ Manager Level ဖြစ်တဲ့ 3 ဆိုရင်လည်း Access လုပ်ခွင့်ပြုကြောင့်သတ်မှတ်ထားပါတယ်။ ဒါကြောင့် ensureRole(2) လို့ ပြောလိုက်ရင် User Role Level တန်ဖိုး 2 ရှိတဲ့ User (သို့မဟုတ်) Manager Level User တွေကို Access လုပ်ခွင့်ပြု တယ်ဆိုတဲ့ အမိမိပါယ်ဖြစ်ပါတယ်။

လိုင်းနံပါတ် (၆၃) မှာတော့ ensureOwner() Function ကို ကြော်လာတ်မှတ်ထားပါတယ်။ ဒါ Function ရဲ့ ရည်ရွယ်ချက်က လက်ရှိ Access လုပ်ဖို့ကြိုးစားတဲ့ User Information က လက်ရှိ Authenticated ဖြစ်နေတဲ့ User ရဲ့ ကိုယ်ပိုင် Information ဟုတ်သလားစီစဉ်ပေးထားခြင်းဖြစ်ပါတယ်။ ပုံမှန်အားဖြင့် အရေးကြိုးတဲ့ User Information ပြုပြင် တဲ့လုပ်ငန်းတွေကို Manager Level ကိုသာခွင့်ပြုပေးမှာဖြစ်ပါတယ်။ ဒါပေမယ့် User တစ်ယောက်က ကိုယ် Information ကို ကိုယ်တိုင်ပြုပြင်ခြင်းကို ခွင့်ပြုသင့်တဲ့အတွက် ခွင့်ပြုနိုင်အောင် ဒီသတ်မှတ်ချက်ကို ထည့်သွင်းသတ်မှတ်ထားခြင်းဖြစ် ပါတယ်။ Access လုပ်ဖို့ကြိုးစားတဲ့ User ရဲ့ ID နဲ့ Session ထဲမှာ သိမ်းထားတဲ့ လက်ရှိ Authenticated User ရဲ့ ID တို့ တိုက်ဆိုင် စစ်ဆေးခြင်းအားဖြင့် Owner ဟုတ်မဟုတ် သိရှိနိုင်မှာဖြစ်ပါတယ်။

လိုင်းနံပါတ် (၇၃) မှာတော့ ensureAssignee() Function ကို ကြော်လာတ်မှတ်ထားပါတယ်။ ပုံမှန်အားဖြင့် Issue တစ်ခုရဲ့ Status ကို Manager မှသာ ပြောင်းခွင့်ရှုမှာဖြစ်ပါတယ်။ ဒါပေမယ့် Assign လုပ်ခြင်းခံရတဲ့ User ကလည်း Status ကို ပြောင်းလိုတဲ့အခါ ပြောင်းခွင့်ပြုနိုင်ဖို့အတွက် ဒီသတ်မှတ်ချက်ကို ထည့်သွင်းသတ်မှတ်ထားခြင်း ဖြစ်ပါတယ်။ လက်ရှိ ပြင်ဆင်ဖို့ကြိုးစားတဲ့ Issue ရဲ့ Assigned To တန်ဖိုးဟာ၊ Authenticated User ရဲ့ ID နဲ့ တူညီမှုရှိမရှိ တိုက်ဆိုင်စစ်ဆေးခြင်းအားဖြင့် Assign လုပ်ခြင်းခံထားရတဲ့ User ဟုတ်မဟုတ်သိနိုင်ပါတယ်။ Issue ရဲ့ Assigned To တန်ဖိုးကို သိရဖို့အတွက် Database ကနေ သွားရောက်စီစဉ်ဖို့လိုတဲ့အတွက် ဒီနေရာမှာတော့ Database နဲ့ ချိတ်ဆက်အသုံးပြုထားရပါတယ်။

လိုင်းနံပါတ် (၉၁) က ensureSubmitter() ကတော့ ensureAssignee() နဲ့ သဘာသဘာဝဆင်တူ ပါတယ်။ Issue ကို မူလစတင် ထည့်သွင်းခဲ့သူကိုလည်း ပြင်ဆင်ခွင့်ကို ပေးလိုကပေးနိုင်စေဖို့အတွက် ထည့်သွင်းသတ်မှတ် ထားခြင်းပဲဖြစ်ပါတယ်။

အခုခုံရင် Authorization နဲ့ Authentication ပိုင်းဆိုင်ရာ သတ်မှတ်ချက်တွေ ပြည့်စုံသွားပြီးဖြစ်ပါတယ်။

### 13.4 – User Router – `routes/users.js`

ဆက်လက်ပြီး User Login, Register နဲ့ Create, Read, Update, Delete လုပ်ငန်းတွေကို `routes/users.js` ထဲမှာ ဆက်လက်ရေးသားပါမယ်။

## JavaScript

```

1. var mongojs = require("mongojs");
2. var crypto = require('crypto');
3. var express = require("express");
4. var validator = require('express-validator');
5. var router = express.Router();
6.
7. var auth = require("../auth");
8. var config = require("../config");
9.
10. var db = mongojs('mydb', ['users']);
11.
12. router.post("/login", auth.login(), function(req, res) {
13.   if(req.user) res.status(200).json(req.user);
14.   else res.sendStatus(401);
15. });
16.
17. router.delete("/logout", function(req, res) {
18.   // req.logout();
19.   req.session.destroy();
20.   res.status(200).json({"logout": "done"});
21. });
22.
23. router.get("/verify", auth.ensureAuth(), function(req, res) {
24.   res.status(200).json(req.user);
25. });

```

ပထမဆုံးအနေနဲ့ သတိပြုသင့်တာကတော့၊ လိုင်းနံပါတ် (၄) မှာ ချိတ်ဆက်ထားတဲ့ express-validator ပါ။ Request နဲ့ အတူပါဝင်လာတဲ့ Data တွေကို Validate လုပ်ဖို့အတွက် ဒီ Package ကို အသုံးပြုမှုဖြစ်ပါတယ်။ လိုင်းနံပါတ် (၇) နဲ့ (၈) တို့မှာ ပြီးခဲ့တဲ့အဆင့်တွေက ရေးသားထားခဲ့တဲ့ config.js နဲ့ auth.js တို့ ကို ချိတ်ဆက်ထားပါတယ်။ .js Extension က မထည့်လည်းရတဲ့အတွက် နမူနာမှာတော့ .js Extension တွေ ကို ထည့်မထားပါဘူး။

Login လုပ်ငန်း ဆောင်ရွက်နိုင်စေဖို့အတွက် လိုင်းနံပါတ် (၁၂) မှာ /login URI နဲ့ POST Route တစ်ခုကို သတ်မှတ် ထားပါတယ်။ auth.js ထဲမှာရေးသားထားခဲ့တဲ့ login() Function အလုပ်လုပ်သွားစေဖို့ အတွက် auth.login() ကို Route Function အနေနဲ့ သတ်မှတ်ပေးထားပါတယ်။ auth.login() က ပါလာတဲ့ Username, Password စစ်ပြီး မှန်တယ်ဆိုမှ Session ထဲမှာ User Data တွေကို သိမ်းပေးမှုဖြစ်ပါတယ်။ ဒါကြောင့် ဒုတိယ Route Function ထဲမှာ req.user ရှိ မရှိကို စိစစ်ပြီး ရှိတော့မှ 200 Status Code နဲ့ အတူ User Data တွေကို Response ပြန်ပေးပြီး မရှိရင် 401 ကိုသာ Response ပြန်ပေးဖို့ သတ်မှတ်ထားပါတယ်။

လိုင်းနံပါတ် (၁၇) မှာတော့ Logout လုပ်ငန်းအတွက် /logout URI နဲ့ DELETE Route တစ်ခု သတ်မှတ်ထားပါတယ်။ Logout လုပ်ဖို့အတွက်နည်းလမ်းနှစ်မျိုးရှိပါတယ်။ တစ်နည်းကတော့ req.logout() Function ကို ခေါ်ယူ လိုက်ခြင်းဖြစ်ပြီး၊ နောက်တစ်နည်းကတော့ Session ထဲက တန်ဖိုးတွေကို ပယ်ဖျက်လိုက်ခြင်းပဲ ဖြစ်ပါတယ်။ နမူနာမှာ တော့ Session ထဲက တန်ဖိုးတွေကို req.session.destroy() နဲ့ ပယ်ဖျက်ထားပါတယ်။

Authenticated User ဟုတ်မဟုတ်စိစစ်တဲ့လုပ်ငန်းအတွက် /verify URI နဲ့ GET Route တစ်ခုကို လိုင်းနံပါတ် (၂၃) မှာသတ်မှတ်ထားပါတယ်။ ဒါ Route မှာတော့ auth.js ထဲက ensureAuth() Function ကိုအသုံးပြုထား ပါတယ်။ ensureAuth() က Login ဝင်ထားတဲ့အတွက် Session တန်ဖိုးရှိရင် next() Function နဲ့ ရေးဆက် သွားခွင့်ပြုထားတဲ့အတွက် နောက် Route Function ထဲမှာ သတ်မှတ်ထားတဲ့ 200 OK ကို Response လုပ်ပေးသွား မှာဖြစ်ပြီး၊ Login ဝင်ထားတဲ့အတွက် Session ရှိမရှိနေရင် 401 Status Code ကို ပြန်ပေးသွား မှာဖြစ်ပါတယ်။

ဆက်လက်ပြီး User အသစ်ထည့်သွင်းတဲ့ လုပ်ဆောင်ချက်အတွက် အောက်ပါအတိုင်း ဖြည့်စွက်ပေးရပါမယ်။

### JavaScript

```

26. router.post("/", auth.ensureSuper(), function(req, res) {
27.   // Validation
28.   req.checkBody('fullName', 'Invalid Full Name').notEmpty();
29.   req.checkBody('loginId', 'Invalid Login ID').isAlphanumeric();
30.   req.checkBody('email', 'Invalid Email Address').isEmail();
31.   req.checkBody('role', 'Invalid Role').isInt();
32.   req.checkBody('password', 'Password required 6 characters').isLength(6);
33.
34.   var errors = req.validationErrors();
35.   if(errors) {
36.     res.status(400).json(errors);
37.     return false;
38.   }
39.
40.   req.body.password = crypto.createHash('sha1')
41.     .update(req.body.password)
42.     .digest("hex");
43.   req.body.roleLabel = config.role[req.body.role];
44.
45.   db.users.insert(req.body, function(err, data) {
46.     if(err) res.status(500).json(err);
47.     else res.status(200).json(data);
48.   });
49. });

```

User အသစ်ထည့်သွင်းဖို့အတွက် / URL မှာ POST Rout တစ်ခုသတ်မှတ်ထားပါတယ်။ User အသစ်ထည့်သွင်း သူ ဟာ Manager Role ရှိသူဖြစ်သင့်တဲ့အတွက် ensureSuper() Function ကို တစ်ဆင့်ခံပြီး စီစစ်ထားပါတယ်။ Manager မဟုတ်သူက ဒါ Route ကို အသုံးပြုဖို့ကြိုးစားရင် 401 Status Code ပြန်ပေးသွားမှာဖြစ်ပါတယ်။

User တစ်ယောက် ထည့်သွင်းတဲ့အခါ ပါဝင်သင့်တဲ့ အချက်အလက်တွေ အားလုံးပါဝင်တာ သေချာအောင်လို လိုင်းနံပါတ် (၂၄) ကနေ (၂၅) ထိ Validation စစ်ထားပါတယ်။ express-validator က req.checkBody() Function နဲ့ Request Body မှာပါဝင်လာတဲ့ တန်ဖိုးတွေကို Validate စစ်နိုင်အောင် ဆောင်ရွက်ပေးထားတဲ့အတွက် req.checkBody() ကို အသုံးပြု စီစစ်ထားခြင်းပဲ ဖြစ်ပါတယ်။ checkBody() Function အတွက် စီစစ်လိုတဲ့ Field နဲ့ ဖော်ပြ စေလိုတဲ့ Error Message ကိုပေးရပါတယ်။ နဲ့

နာအရ isEmpty() နဲ့ နောက်ဆုံးကနေ စစ်ထားတဲ့အတွက် fullName ပါဝင်လာခြင်းမရှိရင် Invalid Full Name ဆိတဲ့ Error Message ကို ရရှိမှာဖြစ်ပါတယ်။

ဆက်လက်ပြီး loginId ကို Alpha-Numeric ပဲ ဖြစ်ရမယ်လို့ စီစဉ်သတ်မှတ်ထားပါတယ်။ email ကိုတော့ isEmail() နဲ့ Email Format မှန်ရမယ်လို့ စီစဉ်ထားပြီး၊ role ကိုတော့ isInt() နဲ့ Integer ဖြစ်ရမယ်လို့ စီစဉ်ထားပါတယ်။ password အတွက်ကိုတော့ isLength(6) နဲ့ အနည်းဆုံး (၆) လုံးရှိရမယ်လို့ သတ်မှတ်ထားပါ တယ်။

express-validator က စီစဉ်ယုံပဲ စီစဉ်ပေးတာပါ။ ရလာတဲ့လဒ်ပေါ်မှတ်ည်ပြီး ဘာလုပ်ရမလဲဆိတာကို တော့ ကိုယ်တိုင် သတ်မှတ်ပေးရမှာဖြစ်ပါတယ်။ ဒါကြောင့် လိုင်းနံပါတ် (၃၅) မှာ req.validation Errors() Function နဲ့ Error ရှိမရှိ စစ်ဆေးပြီး Error ရှိတဲ့အခါ 400 Status Code နဲ့အတူ ရရှိလာတဲ့ Error Message တွေကို Response ပြန်ပေးဖို့ ဆက်လက်သတ်မှတ်ထားခြင်းဖြစ်ပါတယ်။

ပြီးတဲ့အခါ အချက်အလက်တွေကို Database ထဲမှာ မထည့်သွင်းမဲ့ Password ကို SHA1 Hash ပြောင်းပါတယ်။ ပါဝင်လာတဲ့ role ကိုအသုံးပြုပြီး User, Developer, Manager စတဲ့ roleLabel ကို req.body မှာ တစ်ခါတည်း တွဲပေးပါတယ်။ ပြီးတော့မှ လိုင်းနံပါတ် (၄၅) မှာ တန်ဖိုးတွေကို Database ထဲက users Collection မှာ ထည့်သွင်းပေးလိုက်ခြင်းအားဖြင့် User အသစ် ထည့်သွင်းတဲ့ လုပ်ဆောင်ချက်ကို ရရှိသွားမှာပဲ ဖြစ်ပါတယ်။

ဆက်လက်ပြီး ကျွန်ုတ်လုပ်ငန်းတွေဖြစ်တဲ့ User စာရင်းရယူခြင်း၊ User ကိုပယ်ဖျက်ခြင်း၊ User Role ပြုပြင်ခြင်းစ တဲ့ လုပ်ငန်းတွေအတွက် အခုလို ဆက်လက်ရေးသားပေးရပါမယ်။

### JavaScript

```

51. // Get all users
52. router.get("/", auth.ensureAuth(), function(req, res) {
53.   db.users.find({}, function(err, data) {
54.     res.status(200).json(data);
55.   });
56. });
57.
58. // Get user roles
59. router.get("/roles", auth.ensureAuth(), function(req, res) {
60.   res.status(200).json(config.role);
61. });
62.
63. // Get one user
64. router.get("/:id", auth.ensureAuth(), function(req, res) {
65.   var uid = req.params.id;
66.
67.   // Validation
68.   req.checkParams("id", "Invalid User ID").isMongoId();
69.   var errors = req.validationErrors();
70.   if(errors) {
71.     res.status(400).json(errors);
72.     return false;

```

```
73.     }
74.
75.     db.users.findOne({_id: mongojs.ObjectId(uid)}, function(err, data) {
76.       if(data) res.status(200).json(data);
77.       else res.sendStatus(404);
78.     });
79.   });
80.
81. // Update a user
82. router.put("/:id", auth.ensureOwner(), function(req, res) {
83.   var uid = req.params.id;
84.
85.   // Validation
86.   req.checkParams("id", "Invalid User ID").isMongoId();
87.   req.checkBody('fullName', 'Invalid Full Name').isLength(3);
88.   req.checkBody('loginId', 'Invalid Login ID').isAlphanumeric();
89.   req.checkBody('email', 'Invalid Email Address').isEmail();
90.   req.checkBody('role', 'Invalid Role').isInt();
91.
92.   var errors = req.validationErrors();
93.   if(errors) {
94.     res.status(400).json(errors);
95.     return false;
96.   }
97.
98.   var newData = {
99.     fullName: req.body.fullName,
100.    loginId: req.body.loginId,
101.    email: req.body.email,
102.    role: req.body.role,
103.    roleLabel: config.role[req.body.role]
104.  };
105.
106.  if(req.body.password) {
107.    var pwdHash = crypto.createHash('sha1')
108.      .update(req.body.password)
109.      .digest("hex");
110.    newData.password = pwdHash;
111.  }
112.
113.  db.users.update(
114.    { _id: mongojs.ObjectId(uid) },
115.    { $set: newData },
116.    { multi: false },
117.    function(err, data) {
118.      if(err) res.status(500).json(err);
119.      else res.status(200).json(data);
120.    }
121.  );
122. });
123. });
124.
125. router.delete("/:id", auth.ensureSuper(), function(req, res) {
126.   var uid = req.params.id;
127.
128.   // Validation
129.   req.checkParams("id", "Invalid User ID").isMongoId();
130.   var errors = req.validationErrors();
131.   if(errors) {
132.     res.status(400).json(errors);
```

```

133.     return false;
134.   }
135.
136.   db.users.remove({_id: mongojs.ObjectId(uid)}, function(err, data) {
137.     if(data) res.status(200).json(data);
138.     else res.sendStatus(404);
139.   });
140. });
141.
142. module.exports = router;

```

ဒါ Code တွေကတော့ Create, Read, Update, Delete လုပ်ငန်းတွေသာဖြစ်လို့ ရှင်းပြနေစရာမလိုတော့ဘူးလို့ ထင်ပါတယ်။ ရေးထားတဲ့ Code ကို လေ့လာကြည့်လိုက်ရင် အလုပ်လုပ်ပုဂ္ဂို နားလည်မှာပါ။ ထူးခြားချက်အနေ နဲ့ လိုင်းနဲ့ပါတ် (၆၈) express-validate ရဲ့ req.checkParams() ကို အသုံးပြုပြီး URL Parameter ကို Validation စစ်ထား ချက်တစ်ခုရှုပါတယ်။ Validation Function အနေနဲ့ isMongoId() ကိုအသုံးပြုထားပါတယ်။ URL Parameter အဖြစ်ပါဝင်လာတဲ့ id ဟာ MongoDB ID Format ဟုတ်မှန်ရဲ့လား စစ်ဆေးထားခြင်း ဖြစ်ပါတယ်။

နောက်ဆုံးရရှိမယ့် Route Pattern ကတော့ အခုံလိုဖြစ်မှာပါ –

Method	Route	Description	Access Control
POST	/login	User Login ပြုလုပ်ရန်	မည်သူမဆိုအသုံးပြုနိုင်
DELETE	/logout	User Logout ပြုလုပ်ရန်	မည်သူမဆိုအသုံးပြုနိုင်
GET	/verify	Authenticated User ဟုတ်မဟုတ်စိစစ်ရန်	မည်သူမဆိုအသုံးပြုနိုင်
POST	/	User အသစ်ထည့်သွင်းရန်	Manager Only
GET	/	User စာရင်းရယူရန်	Authenticated User Only
GET	/roles	User Role စာရင်းရယူရန်	Authenticated User Only
GET	/:id	User တစ်ဦးအတွက် Data ရယူရန်	Authenticated User Only
PUT	/:id	User တစ်ဦး၏ Data ကို Update ပြုလုပ်ရန်	Account Owner & Manager
DELETE	/:id	User တစ်ဦးအား ပယ်ဖျက်ရန်	Manager Only

### 13.5 – Issues Router - `routes/issues.js`

ဆက်လက်ပြီး Issue နဲ့ပက်သက်တဲ့ Code တွေကို ဆက်လက်ရေးသားဖို့အတွက် `routes/issues.js` ထဲမှာ အခုလိုရေး သားပေးရပါမယ်။

#### JavaScript

```

1. var mongojs = require("mongojs");
2. var express = require("express");
3. var validator = require('express-validator');
4. var router = express.Router();
5.
6. var auth = require("../auth");
7. var config = require("../config");
8.
9. var db = mongojs('mydb', ['issues']);
10.
11. // Get all issues
12. router.get("/", auth.ensureAuth(), function(req, res) {
13.   db.issues.find({}, function(err, data) {
14.     res.status(200).json(data);
15.   });
16. });
17.
18. // Get one issue
19. router.get("/:id", auth.ensureAuth(), function(req, res) {
20.   var iid = req.params.id;
21.
22.   // Validation
23.   req.checkParams("id", "Invalid Issue ID").isMongoId();
24.   var errors = req.validationErrors();
25.   if(errors) {
26.     res.status(400).json(errors);
27.     return false;
28.   }
29.
30.   db.issues.findOne({_id: mongojs.ObjectId(iid)}, function(err, data) {
31.     if(data) res.status(200).json(data);
32.     else res.sendStatus(404);
33.   });
34. });
35.
36. // Create a new issue
37. router.post("/", auth.ensureRole(1), function(req, res) {
38.   // Validation
39.   req.checkBody('summary', 'Invalid Summary').notEmpty();
40.   req.checkBody('priority', 'Invalid Priority').isInt();
41.   req.checkBody('type', 'Invalid Type').isInt();
42.
43.   var errors = req.validationErrors();
44.   if(errors) {
45.     res.status(400).json(errors);
46.     return false;
47.   }
48.
49.   var newIssue = {
50.     summary: req.body.summary,
51.     detail: req.body.detail,
52.   }

```

```
53.     priority: req.body.priority,
54.     type: req.body.type,
55.     typeLabel: config.type[req.body.type],
56.     priorityLabel: config.priority[req.body.priority],
57.     status: 0,
58.     statusLabel: config.status[0],
59.     assignedTo: null,
60.     assignedToLabel: null,
61.     submittedAt: new Date(),
62.     submittedBy: mongojs.ObjectId(req.user._id),
63.     submittedByLabel: req.user.fullName
64.   }
65.
66.   db.issues.insert(newIssue, function(err, data) {
67.     if(err) res.status(500).json(err);
68.     else res.status(200).json(data);
69.   });
70. });
71.
72. // Update an issue
73. router.put("/:id", auth.ensureRole(1), function(req, res) {
74.   var iid = req.params.id;
75.
76.   // Validation
77.   req.checkParams("id", "Invalid Issue ID").isMongoId();
78.   req.checkBody("summary", "Invalid Summary").notEmpty();
79.
80.   var errors = req.validationErrors();
81.   if(errors) {
82.     res.status(400).json(errors);
83.     return false;
84.   }
85.
86.   var newData = {
87.     summary: req.body.summary,
88.     detail: req.body.detail,
89.     modifiedAt: new Date()
90.   };
91.
92.   db.issues.update(
93.     { _id: mongojs.ObjectId(iid) },
94.     { $set: newData },
95.     { multi: false },
96.     function(err, data) {
97.       if(err) res.status(500).json(err);
98.       else res.status(200).json(data);
99.     }
100.   );
101. });
102.
103. // Delete an issue
104. router.delete("/:id", auth.ensureSubmitter(), function(req, res) {
105.   var iid = req.params.id;
106.
107.   // Validation
108.   req.checkParams("id", "Invalid Issue ID").isMongoId();
109.   var errors = req.validationErrors();
110.   if(errors) {
111.     res.status(400).json(errors);
```

```

112.     return false;
113.   }
114.
115.   db.issues.remove({_id: mongojs.ObjectId(iid)}, function(err, data) {
116.     if(data) res.status(200).json(data);
117.     else res.sendStatus(404);
118.   });
119. });

```

ထူးခြားတဲ့လုပ်ဆောင်ချက် မပါဝင်ပဲ။ issues Collection ပေါ်မှာ Create, Read, Update, Delete လုပ်ငန်းတွေ ဆောင်ရွက်ထားခြင်းသာဖြစ်ပါတယ်။ ဆက်လက်ပြီး Issue တစ်ခုရဲ့ Status, Priority နဲ့ Assignment တို့ကို သီးခြားစီ ပြင်ဆင်နိုင်ဖို့အတွက် အခုလို ရေးသားပေးရပါမယ်။

### JavaScript

```

120. // Update issue type
121. router.patch("/type/:id", auth.ensureRole(1), function(req, res) {
122.   var iid = req.params.id;
123.
124.   // Validation
125.   req.checkParams("id", "Invalid Issue ID").isMongoId();
126.   req.checkBody("type", "Invalid Type").isInt();
127.
128.
129.   var errors = req.validationErrors();
130.   if(errors) {
131.     res.status(400).json(errors);
132.     return false;
133.   }
134.
135.   var newData = {
136.     type: req.body.type,
137.     typeLabel: config.type[req.body.type],
138.     modifiedAt: new Date()
139.   };
140.
141.   db.issues.update(
142.     { _id: mongojs.ObjectId(iid) },
143.     { $set: newData },
144.     { multi: false },
145.     function(err, data) {
146.       if(err) res.status(500).json(err);
147.       else res.status(200).json(data);
148.     }
149.   );
150. });
151.
152. // Update issue priority
153. router.patch("/priority/:id", auth.ensureRole(2), function(req, res) {
154.   var iid = req.params.id;
155.
156.   // Validation
157.   req.checkParams("id", "Invalid Issue ID").isMongoId();
158.   req.checkBody("priority", "Invalid Priority").isInt();
159.

```

```
160.     var errors = req.validationErrors();
161.     if(errors) {
162.         res.status(400).json(errors);
163.         return false;
164.     }
165.
166.     var newData = {
167.         priority: req.body.priority,
168.         priorityLabel: config.priority[req.body.priority],
169.         modifiedAt: new Date()
170.     };
171.
172.     db.issues.update(
173.         { _id: mongojs.ObjectId(iid) },
174.         { $set: newData },
175.         { multi: false },
176.         function(err, data) {
177.             if(err) res.status(500).json(err);
178.             else res.status(200).json(data);
179.         }
180.     );
181. });
182.
183. // Update issue assign to
184. router.patch("/assign/:id", auth.ensureSuper(), function(req, res) {
185.     var iid = req.params.id;
186.
187.     // Validation
188.     req.checkParams("id", "Invalid Issue ID").isMongoId();
189.     req.checkBody("assignedTo", "Invalid User ID").isMongoId();
190.     req.checkBody("assignedToLabel", "Invalid User Name").notEmpty();
191.
192.     var errors = req.validationErrors();
193.     if(errors) {
194.         res.status(400).json(errors);
195.         return false;
196.     }
197.
198.     var newData = {
199.         assignedTo: req.body.assignedTo,
200.         assignedToLabel: req.body.assignedToLabel,
201.         status: 1,
202.         statusLabel: config.status[1],
203.         modifiedAt: new Date()
204.     };
205.
206.     db.issues.update(
207.         { _id: mongojs.ObjectId(iid) },
208.         { $set: newData },
209.         { multi: false },
210.         function(err, data) {
211.             if(err) res.status(500).json(err);
212.             else res.status(200).json(data);
213.         }
214.     );
215. });
216.
217. // Update issue status
```

```
218. router.patch("/status/:id", auth.ensureAssignee(), function(req, res) {  
219.   var iid = req.params.id;  
220.  
221.   // Validation  
222.   req.checkParams("id", "Invalid Issue ID").isMongoId();  
223.   req.checkBody("status", "Invalid Status").isInt();  
224.  
225.   var errors = req.validationErrors();  
226.   if(errors) {  
227.     res.status(400).json(errors);  
228.     return false;  
229.   }  
230.  
231.   var newData = {  
232.     status: req.body.status,  
233.     statusLabel: config.status[req.body.status],  
234.     modifiedAt: new Date()  
235.   };  
236.  
237.   db.issues.update(  
238.     { _id: mongojs.ObjectId(iid) },  
239.     { $set: newData },  
240.     { multi: false },  
241.     function(err, data) {  
242.       if(err) res.status(500).json(err);  
243.       else res.status(200).json(data);  
244.     }  
245.   );  
246. });  
247.  
248. // Close an issue  
249. router.patch("/close/:id", auth.ensureSuper(), function(req, res) {  
250.   var iid = req.params.id;  
251.  
252.   // Validation  
253.   req.checkParams("id", "Invalid Issue ID").isMongoId();  
254.  
255.   var errors = req.validationErrors();  
256.   if(errors) {  
257.     res.status(400).json(errors);  
258.     return false;  
259.   }  
260.  
261.   var newData = {  
262.     status: 4,  
263.     statusLabel: config.status[4],  
264.     modifiedAt: new Date()  
265.   };  
266.  
267.   db.issues.update(  
268.     { _id: mongojs.ObjectId(iid) },  
269.     { $set: newData },  
270.     { multi: false },  
271.     function(err, data) {  
272.       if(err) res.status(500).json(err);  
273.       else res.status(200).json(data);  
274.     }  
275.   );
```

```
276. });

```

ဒီနေရာမှာ သတိပြုသင့်တာကတော့ PATCH Request Method ကို အသုံးပြုထားခြင်းပဲဖြစ်ပါတယ်။ အခြေခံ အားဖြင့် အချက်အလက်တွေ ပြင်ဆင်ဖို့အတွက် PUT Request Method သုံးပါတယ်။ ဒါပေမယ့် လက်တွေမှာ HTTP သတ်မှတ်ချက်အရ PUT ကို အချက်အလက်တစ်ချို့ ပြင်ဆင်ဖို့ဆိုမှာ မသုံးသင့်ပဲ အချက်အလက်အားလုံး ကို အစားထိုးဖို့သာ အသုံးပြုသင့်ပါတယ်။ တန်ညွှေးအားဖြင့် Record Set တိုးတစ်ခုလုံးကို သီးခြား Record Set တစ်ခုနဲ့ အစားထိုးလိုက်တဲ့ ပြင်ဆင်မှုမျိုးအတွက် PUT ကို အသုံးပြုရခြင်းဖြစ်ပြီး၊ Record Set တစ်ခုရဲ့ အစိတ်အပိုင်းတစ်ချို့ကိုသာ ပြင်ဆင်လိုရင်တော့ PATCH Request Method ကို အစားထိုးအသုံးပြုသင့်ပါတယ်။ Issue တစ်ခုရဲ့ Priority ပြင်ဆင်ခြင်း၊ Status ပြင်ဆင် ခြင်း စတဲ့လုပ်ငန်းတွေဟာ Issue တစ်ခုလုံးကို အစားထိုး ပြင်ဆင်ခြင်းမဟုတ်ပဲ အစိတ်အပိုင်းတစ်ခုကိုသာ ပြင်ဆင်ခြင်းဖြစ်တဲ့အတွက် ဒီနေရာမှာ PATCH Request Method ကို အသုံးပြုခြင်းပဲဖြစ်ပါတယ်။

ဆက်လက်ပြီး Issue Comment ထည့်သွင်းခြင်းနဲ့ ပယ်ဖျက်ခြင်းတို့အတွက် အခုလိုရေးသားပေးရပါမယ်။

## JavaScript

```
277. // Comments
278. router.post("/comments", auth.ensureAuth(), function(req, res) {
279.   req.checkBody('issueId', 'Invalid Issue ID').isMongoId();
280.   req.checkBody('authorId', 'Invalid Author ID').isMongoId();
281.   req.checkBody('authorName', 'Invalid Author Name').notEmpty();
282.   req.checkBody('comment', 'Invalid Comment Body').notEmpty();
283.
284.   var errors = req.validationErrors();
285.   if(errors) {
286.     res.status(400).json(errors);
287.     return false;
288.   }
289.
290.   db.issues.findOne({
291.     _id: mongojs.ObjectId(req.body.issueId)
292.   }, function(err, data) {
293.     if(data) {
294.       var comments = data.comments || [];
295.       var comment = {
296.         comment: req.body.comment,
297.         authorId: req.body.authorId,
298.         authorName: req.body.authorName,
299.         submittedAt: new Date()
300.     }
301.
302.     comments.push(comment);
303.
304.     db.issues.update(
305.       { _id: mongojs.ObjectId(req.body.issueId) },
306.       { $set: { comments: comments } },
307.       { multi: false },
308.       function(err, data) {
309.         if(err) res.status(500).json(err);
310.         else res.status(200).json(data);
311.       }
312.     );

```

```

313.     }
314.     else {
315.         res.sendStatus(404);
316.     }
317. });
318. });
319.
320. router.delete("/comments", auth.ensureAuth(), function(req, res) {
321.     req.checkBody('issueId', 'Invalid Issue ID').isMongoId();
322.     req.checkBody('commentId', 'Invalid Comment ID').isInt();
323.
324.     var errors = req.validationErrors();
325.     if(errors) {
326.         res.status(400).json(errors);
327.         return false;
328.     }
329.
330.     db.issues.findOne({
331.         _id: mongojs.ObjectId(req.body.issueId)
332.     }, function(err, data) {
333.         if(data) {
334.             var comments = data.comments || [];
335.             comments.splice(req.body.commentId, 1);
336.
337.             db.issues.update(
338.                 { _id: mongojs.ObjectId(req.body.issueId) },
339.                 { $set: { comments: comments } },
340.                 { multi: false },
341.                 function(err, data) {
342.                     if(err) res.status(500).json(err);
343.                     else res.status(200).json(data);
344.                 }
345.             );
346.         }
347.     else {
348.         res.sendStatus(404);
349.     }
350. });
351. });

```

ဒီနေရာမှာ ထူးခြားချက်အနေနဲ့ Comment တွေကို သီးခြား Collection နဲ့မသုံးပဲ၊ Issue နဲ့အတူ သက်ဆိုင်ရာ Comment တွေကို တစ်ခါတည်း တဲ့ဖက်သိမ်းဆည်းမှာဖြစ်ပါတယ်။ Comment တွေကို သီးခြား Collection နဲ့ သိမ်းမယ် ဆိုရင် issue နဲ့ comment Collection နှစ်ခုအကြား Relationship တစ်ခုရှုံးသွားတော့မှာဖြစ်ပါတယ်။ MongoDB ရဲ့ NoSQL သဘာဝအရ ဖြစ်နိုင်ရင် Relationship ကိုရှေ့ငွေ့သင့်တဲ့အတွက် သီးခြားမသိမ်းပဲ Issue နဲ့အတူ တဲ့ဖက်သိမ်း ဆည်းခြင်းဖြစ်ပါတယ်။ ဒီနည်းနဲ့ Issue တစ်ခုကို ရယူလိုက်ရင်၊ သက်ဆိုင်ရာ Comment တွေကို သီးခြားထပ်မံရယူစရာ မလိုပဲ တစ်ခါတည်းပါဝင်သွားမှာဖြစ်ပါတယ်။ ဒါကြောင့် Comment တည်ဆောက်ခြင်းနဲ့ ပယ်ဖျက်ခြင်းတို့ကိုသာ ရေးသား ထားခြင်းဖြစ်ပါတယ်။ Comment တွေရယူခြင်းကို ထည့်သွင်းရေးသားမထားပါဘူး။ Issue တစ်ခုရယူလိုက်ရင် Comment တွေက တစ်ခါတည်း ပါဝင်သွားမှာပဲ ဖြစ်ပါတယ်။

ဆက်လက်ပြီး Issue Router ရဲ့နောက်ဆုံးပိုင်းအနေနဲ့ Issue Status, Priority နဲ့ Type တို့ကို ရယူနိုင်တဲ့ Route

တွေကို ဆက်လည့်ထုတ္တုံးပါမယ်။

### JavaScript

```

352. // Get statuses
353. router.get("/statuses", auth.ensureAuth(), function(req, res) {
354.   res.status(200).json(config.status);
355. });
356.
357. // Get priorities
358. router.get("/priorities", auth.ensureAuth(), function(req, res) {
359.   res.status(200).json(config.priority);
360. });
361.
362. // Get types
363. router.get("/types", auth.ensureAuth(), function(req, res) {
364.   res.status(200).json(config.type);
365. });
366.
367.
368. module.exports = router;

```

အားလုံးပြီးစီးသွားတဲ့အခါမှာ ရရှိမယ့် Route Pattern ကတော့ အခုလိုဖြစ်မှာပဲဖြစ်ပါတယ်။

Method	URI	Description	Access Control
GET	/	Issue စာရင်းရယူရန်	Authenticated User Only
GET	/:id	Issue တစ်ခုအတွက် Data ရယူရန်	Authenticated User Only
POST	/	Issue အသစ်တစ်ခုထည့်သွင်းရန်	Tester & Above
PUT	/:id	Issue တစ်ခု၏ Data ပြောင်းရန်	Tester & Above
DELETE	/:id	Issue တစ်ခုအား ပယ်ဖျက်ရန်	Submitter & Manager
PATCH	/type/:id	Issue Type ပြောင်းရန်	Tester & Above
PATCH	/priority/:id	Issue Priority ပြောင်းရန်	Developer & Above
PATCH	/assign/:id	Issue အတွက် တာဝန်ကျသူ့ပြောင်းရန်	Manager Only
PATCH	/status/:id	Issue Status ပြောင်းရန်	Assignee & Manager
PATCH	/close/:id	Issue Status ကို Closed ပြောင်းရန်	Manager Only
POST	/comments	Comment တစ်ခုထည့်သွင်းရန်	Authenticated User Only
DELETE	/comments	Comment တစ်ခုပယ်ဖျက်ရန်	Authenticated User Only
GET	/statuses	Issue Status စာရင်းရယူရန်	Authenticated User Only
GET	/priorities	Issue Priority စာရင်းရယူရန်	Authenticated User Only
GET	/types	Issue Type စာရင်းရယူရန်	Authenticated User Only

အခုခိုရင် လိုအပ်တဲ့ လုပ်ဆောင်ချက်တွေ ပြည့်စုံပြီး Main App ကို စတင်ရေးသားနိုင်ပြီဖြစ်ပါတယ်။

### 13.6 – Main App – app.js

ဆက်လက်ပြီး app.js မှာ အခုလိုရေးသားပေးရမှာဖြစ်ပါတယ်။

#### JavaScript

```

1. var express = require("express");
2. var validator = require('express-validator');
3. var cookieParser = require("cookie-parser");
4. var bodyParser = require("body-parser");
5.
6. var issues = require("./routes/issues");
7. var users = require("./routes/users");
8.
9. var app = express();
10.
11. var passport = require("passport");
12.
13. app.use(require('express-session') ({
14.     secret: 'secret',
15.     resave: false,
16.     saveUninitialized: false
17. }));
18.
19. app.use(passport.initialize());
20. app.use(passport.session());
21.
22. app.use(bodyParser.json());
23. app.use(bodyParser.urlencoded({
24.     extended: true
25. }));
26. app.use(cookieParser());
27.
28. app.use(validator());
29.
30. app.use(function(req, res, next) {
31.     res.set("Access-Control-Allow-Origin", "http://localhost:8080/");
32.     res.set("Access-Control-Allow-Credentials", "true");
33.     res.set("Access-Control-Allow-Methods",
34.             "GET, POST, PUT, DELETE, PATCH, OPTIONS");
35.     res.set("Access-Control-Allow-Headers", "X-CSRF-Token, X-Requested-With,
36.             Accept, Accept-Version, Content-Length, Content-MD5,
37.             Content-Type, Date, X-Api-Version");
38.     next();
39. });
40.
41. app.use(express.static("./static"));
42. app.use("/api/issues", issues);
43. app.use("/api/users", users);
44.
45. app.listen(3000, function() {
46.     console.log("API server running at port 3000");
47. });

```

အမိကလုပ်ဆောင်ချက်များ မပါဝင်ပဲ `require()` နဲ့ `app.use()` တို့ကိုသုံးပြီး အသုံးပြုဖို့လိုအပ်တဲ့ Package များကို ချိတ်ဆက်ကြော်ကြော်ထားခြင်းသာ ဖြစ်ပါတယ်။ လိုင်းနံပါတ် (၄၁) မှာ Static File Server အဖြစ် အလုပ်လုပ်ဖို့ သတ်မှတ်ထားတဲ့အတွက် Static ဖိုင်အဖြစ် Response ပြန်ပေးစေလိုတဲ့ဖိုင်တွေကို `./static` Directory မှာထည့် သွင်းထားနိုင်တာကို သတိပြုပါ။ လိုင်းနံပါတ် (၄၂) နဲ့ (၄၃) မှာတော့ ကျွန်ုတ်တို့ ကြိုးတင်ရေးသားထားတဲ့ Issue Router နဲ့ User Router တို့ကို Sub-App အဖြစ် တဲ့ဖက်ပေးထားပါတယ်။ `/api/issues` နဲ့ `/api/users` တို့ကို API URL အဖြစ် အသီးသီး အသုံးပြုရမှုဖြစ်ပါတယ်။

ထူးခြားချက်အနေနဲ့ လိုင်းနံပါတ် (၂၀) မှာ `app.use()` ကိုသုံးပြီး Default Response Header တွေ တန်းစီ သတ်မှတ်ထားပါတယ်။ ExpressJS Server က Response ပြန်ပေးတိုင်းမှာ ဒါ အလိုအလျောက် တဲ့ဖက်ထည့်သွင်းပေးသွားမှာဖြစ်ပါတယ်။

ဥပမာ - Web Client က `example.com` မှာ အလုပ်လုပ်နေဖြီး API Service က `api.example.com` မှာ Host လုပ်ထားတယ်ဆိုပါစို့။ Browser တွေမှာ လုပ်ချိန်များတဲ့ Cross-Origin Policy ဆိုတာရှိပါတယ်။ အဲဒါ ပေါ်လိမ့်အရ Web Browser က `example.com` မှာ အလုပ်လုပ်နေတဲ့ Web App ကနဲ့ `api.example.com` ကို Ajax နဲ့ Request ပြုလုပ်ခြင်းကို ခွင့်ပြုမှုမဟုတ်ပါဘူး။ ဒီပြုသာနာကို ဖြေရှင်းဖို့အတွက် Cross-Origin Resource Sharing (CORS) လိုအပ်တဲ့ နည်းစနစ်ကို အသုံးပြုရပါတယ်။ CORS နည်းစနစ်အရ Server က Response Header ထဲမှာ `Access-Control-Allow-Origin` ဆိုတဲ့ Header ကို ထည့်သွင်းပေးပို့ခြင်းအားဖြင့် Cross-Origin Request တွေကို ခွင့်ပြုကြောင်း ပြောပေးရပါတယ်။

လိုင်းနံပါတ် (၃၁) မှာ `Access-Control-Allow-Origin` အတွက်တန်ဖိုးကို `http://localhost:8080` လိုပြောထားတဲ့ အတွက် ဒါ အသီပေးလက်ခံမယ် လို့ အသီပေးလက်ခံမယ်။ အကယ်၍ Web Client ရဲ့ လိပ်စာက `example.com` ဆိုရင် ဒီနေရာမှာ `http://example.com` လို့ ထည့်သွင်းသတ်မှတ်ပေးရမှာဖြစ်ပါတယ်။ အကယ်၍ မည်သည့် Client ကမဆိုပြုလုပ်တဲ့ Cross-Origin Request တွေကိုလက်ခံလိုရင်တော့ \* သက်တကို အသုံးပြုနိုင်ပါတယ်။ ဥပမာ -

### JavaScript

```
res.set("Access-Control-Allow-Origin", "*");
```

လိုင်းနံပါတ် (၄၂) မှာ `Access-Control-Allow-Credentials` အတွက်တန်ဖိုးကို `true` လို့ သတ်မှတ်ထား ရတဲ့ ရည်ရွယ်ချက်ကတော့၊ Cross-Origin Request တွေနဲ့အတူ Cookie တွေထည့်သွင်းပေးရင်လည်း လက်ခံအလုပ်လုပ်စေလိုတဲ့အတွက် ထည့်သွင်းထားရခြင်းဖြစ်ပါတယ်။ အကယ်၍ ဒါ အသီပေးလက်ခံမယ်။ `Access-Control-Allow-Origin` တန်ဖိုးကို ခွင့်ပြုလိုတဲ့ Origin Domain Name အတိအကျိုး သတ်မှတ်ပေးရပါတယ်။ \*

တစ်ခုသတ်မှတ်ပေးရတဲ့ `Access-Control-Allow-Credentials` တန်ဖိုးကို `true` သတ်မှတ်ထားရင် `Access-Control-Allow-Origin` တန်ဖိုးကို ခွင့်ပြုလိုတဲ့ Origin Domain Name အတိအကျိုး သတ်မှတ်ပေးရပါတယ်။ \*

ကို အသုံးပြုလို မရတော့ပါဘူး။ ဒါကြောင့်လည်း ကျွန်တော်တို့ Code ထဲမှာ Access-Control-Allow-Origin တန်ဖိုးကို \* လိုပေးမထားပဲ <http://localhost:8080> လို့ အတိအကျသတ်မှတ်ပေးထားခြင်းဖြစ်ပါ တယ်။ ဘာ ကြောင့်မရတာလဲဆိတဲ့ နောက်ကွယ်ကအသေးစိတ်ကိုတော့ ဒီနေရာမှာ ထည့်သွင်းမဖော်ပြနိုင်တော့ပါဘူး။ Access-Control-Allow-Credentials တန်ဖိုး `true` သက်မှတ်လိုရင် Access-Control-Allow-Origin တန်ဖိုးကို \* သုံးလို့မရဘူးလို့သာ အလွယ်မှတ်ပေးပါ။

လိုင်းနံပါတ် (၃၃) မှာတော့ Access-Control-Allow-Methods Header နဲ့ Cross-Origin Request တွေနဲ့ အတူ ခွင့်ပြုလိုတဲ့ Request Method စာရင်းကို သတ်မှတ်ပေးထားပါတယ်။ ဒီနေရာမှာလည်း Request Method အားလုံးကိုခွင့်ပြုလိုရင် \* သက်တကို အသုံးပြုနိုင်ပါတယ်။

လိုင်းနံပါတ် (၃၅) မှာတော့ Access-Control-Allow-Headers နဲ့ Cross-Origin Request တွေနဲ့အတူ ထည့်သွင်းပေးပို့ခြင်းကို ခွင့်ပြုမယ့် Header စာရင်းကို သတ်မှတ်ထားပါတယ်။ API Server တွေကို ပေးပို့ဖို့ လိုအပ်တက်လေ့ရှိတဲ့ Header စာရင်းကို ပေးထားခြင်းဖြစ်ပါတယ်။ ဒီနေရာမှာလည်း Header အားလုံးကို ခွင့်ပြုလိုရင် \* သက်တကို အသုံးပြုနိုင်ပါတယ်။

CORS နဲ့ပက်ပြီး အသေးစိတ်ဆက်လက်လေ့လာလိုရင် အောက်ပါလိပ်စာမှာ လေ့လာနိုင်ပါတယ်။

[https://developer.mozilla.org/en-US/docs/Web/HTTP/Access\\_control\\_CORS](https://developer.mozilla.org/en-US/docs/Web/HTTP/Access_control_CORS)

အခုခုံရင် ကျွန်တော်တို့ရဲ့ Issue Tracking System API Service အတွက် လိုအပ်ချက်တွေအားလုံးပြည့်စုံပြီ့မို့စတင် စမ်းသပ်နိုင်ပြီဖြစ်ပါတယ်။

### 13.7 – Testing the API

ရေးသားထားတဲ့ Service ကို စမ်းသပ်နိုင်ဖို့အတွက် ပထမဆုံး User Account တစ်ခုကို mydb Database ရဲ့ users Collection ထဲမှာ ကြိုတင်တည်ဆောက်ထား ပေးရမှာဖြစ်ပါတယ်။ User Account တည်ဆောက်ရာမှာ အသုံးပြုဖို့ လိုအပ်တဲ့အတွက် Password Hash တစ်ခုကို Node နဲ့ အရင် Generate လုပ်ယူပါမယ်။

```
$ node
> var crypto = require("crypto");
> var password =
> crypto.createHash("sha1").update("123456").digest("hex");
> console.log(password);

7c4a8d09ca3762af61e59520943dc26494f8941b
```

123456 ဆိုတဲ့တန်ဖိုးအတွက် SHA1 Hash တစ်ခုကို Generate လုပ်ယူလိုက်ခြင်းဖြစ်ပါတယ်။ ပြီးတဲ့အခါ Mongo Shell ကနေတစ်ဆင့် Manager Account တစ်ခုထည့်သွင်းပါမယ်။

```

$ mongo
> use mydb;

switched to db mydb

> db.users.insert({
...   fullName: "John Doe",
...   loginId: "manager",
...   email: "john@example.com",
...   role: 3,
...   password: "7c4a8d09ca3762af61e59520943dc26494f8941b"
... });
> db.users.find().pretty();

{
  "_id" : ObjectId("55b703a9d2141ae9733ac0eb"),
  "fullName" : "John Doe",
  "loginId" : "manager",
  "email" : "john@example.com",
  "role" : 3,
  "password" : "7c4a8d09ca3762af61e59520943dc26494f8941b"
}

```

loginId ကို manager လိုပေးထားပြီး password ကို 123456 လိုပေးထားတဲ့ User Account တစ်ခုရရှိ သွားပြီဖြစ်ပါတယ်။ Issue Tracking System API ကို စတင်စမ်းသပ်နိုင်ဖို့အတွက် Server ကို Run ပေးရပါမယ်။

```

$ node app.js
API server running at port 3000

```

အခုခံရင် ကျွန်တော်တို့ရေးသားထားတဲ့ Service က Port 3000 မှာ အလုပ်လုပ်နေပြီဖြစ်လို့ CURL နဲ့ ဆက် သွယ်စမ်းသပ်နိုင်ပြီးဖြစ်ပါတယ်။ ပထမဆုံးအနေနဲ့ Verify လုပ်ကြည့်ပါမယ်။

```

$ curl -X GET localhost:3000/api/users/verify
Unauthorized%

```

/users/verify ကို Request လုပ်ကြည့်တဲ့အခါ 401 Unauthorized ကို လက်ခံရရှိခြင်းဖြစ်ပါတယ်။ ဆက်လက် ပြီး Login ဝင်ကြည့်ပါမယ်။

```
$ curl -X POST localhost:3000/api/users/login -d
"username=manager&password=123456"

{"_id":"55b70503d2141ae9733ac0ec","fullName":"John
Doe","loginId":"manager","email":"john@example.com","role":3,"password":"7c4a8
d09ca3762af61e59520943dc26494f8941b"}%
```

```
$ curl -X GET localhost:3000/api/users/verify
```

```
Unauthorized%
```

/users/login ကို POST Request ပြုလုပ်ပြီး Username, Password အမှန်ပေးလိုက်နိုင်တဲ့အတွက် User Data တွေ ကို API Server က Response ပြန်ပေးခြင်းဖြစ်ပါတယ်။ ဒါပေမယ့် /users/verify ကို ပြန်စစ်ကြည့်တဲ့အခါ Unauthorized ပဲဆက်ပြနေတာကို တွေ့ရမှာဖြစ်ပါတယ်။ ပြသာနာက Login Success ဖြစ်ပြီး User Data တွေကို Session ထဲမှာ သိမ်းသွားပေမယ့် CURL က အဲဒီ Session ရဲ့ ID ကို မသိဘူးဖြစ်နေတာပါ။ Session ID က Cookie တစ်ခုနဲ့ တဲ့သိမ်းပေးဖို့လိုပါတယ်။ ဒါကြောင့် အခုလုပ် Login ဝင်မှ မှန်မှာပါ။

```
$ curl -X POST localhost:3000/api/users/login \
-d "username=manager&password=123456" -c cookie.txt
```

```
{"_id":"55b70503d2141ae9733ac0ec","fullName":"John
Doe","loginId":"manager","email":"john@example.com","role":3,"password":"7c4a8
d09ca3762af61e59520943dc26494f8941b"}%
```

နောက်ဆုံးက -c Option နဲ့ Session ID ကို cookie.txt ဖိုင်ထဲမှာ ရေးသားစေလိုက်ခြင်းဖြစ်ပါတယ်။ နောင် Request ပြုလုပ်တဲ့အခါ Username, Password ကို ထပ်ပေးဖို့မလိုတော့ပေမယ့် အဲဒီ Cookie ဖိုင်ကိုတော့ တဲ့ထည့်ပေးဖို့လိုပါတယ်။ -b Option ကို သုံးရပါတယ်။

```
$ curl -X GET localhost:3000/api/users/verify -b cookie.txt
```

```
{"_id":"55b70503d2141ae9733ac0ec","fullName":"John
Doe","loginId":"manager","email":"john@example.com","role":3,"password":"7c4a8
d09ca3762af61e59520943dc26494f8941b"}%
```

နောက်တစ်ကြိမ် /users/verify ကို Request လုပ်တဲ့အခါ -b Option နဲ့ Cookie ဖိုင်ကိုတဲ့ဖက်ပေးလိုက်တဲ့အခါ Authorized ဖြစ်ပြီးဖြစ်တဲ့အတွက် လက်ရှိ Authorized User ရဲ့ အချက်အလက်တွေကို Response အနေနဲ့ လက်ခံရရှိ တာကို တွေ့ရမှာဖြစ်ပါတယ်။

ဆက်လက်ပြီး လက်ရှိရှိနေတဲ့ User စာရင်းကို ရယူကြည့်နိုင်ပါတယ်။

```
$ curl -X GET localhost:3000/api/users -b cookie.txt
[{"_id":"55b70503d2141ae9733ac0ec","fullName":"John Doe","loginId":"manager","email":"john@example.com","role":3,"password":"7c4a8d09ca3762af61e59520943dc26494f8941b"}]%
```

လိုအပ်တဲ့အချက်အလက်တွေ မှန်ကန်အောင်မပေးပဲ User အသစ်တစ်ယောက်ကို တည်ဆောက်ဖို့ကြီးစားရင် တော့ အခုလို ပြန်လည်ရရှိမှာဖြစ်ပါတယ်။

```
$ curl -X POST localhost:3000/api/users -b cookie.txt
[{"param":"fullName","msg":"Invalid Full Name"}, {"param":"loginId","msg":"Invalid Login ID"}, {"param":"email","msg":"Invalid Email Address"}, {"param":"role","msg":"Invalid Role"}, {"param":"password","msg":"Password should have at least 6 characters"}]%
```

express-validate နဲ့ fullName, loginId, email စတဲ့အချက်အလက်တွေပါဝင်ဖို့လိုကြောင်း စစ်ဆေးထားတဲ့အတွက် ဖြစ်ပါတယ်။ ဒါကြောင့် လိုအပ်တဲ့အချက်အလက်အပြည့်အစုံနဲ့ အခုလို Request လုပ်ပေးဖို့ လိုပါတယ်။

```
$ curl -X POST localhost:3000/api/users \
-d "fullName=Bob&loginId=bob&email=bob@gmail.com&role=2&password=123456" \
-b cookie.txt
{"fullName":"Bob","loginId":"bob","email":"bob@gmail.com","role":"2","password":"7c4a8d09ca3762af61e59520943dc26494f8941b","roleLabel":"Developer","_id":"55b70b342468830b671da93b"}%
```

ဒီတစ်ကြိမ်မှာတော့ လိုအပ်တဲ့အချက်အလက်တွေ အပြည့်အစုံပါဝင်တဲ့အတွက် ထည့်သွင်းလိုက်တဲ့ User ရဲ့ အချက် အလက်တွေကို Response အနေဖြင့် ပြည်လည်လက်ခံ ရရှိမှာဖြစ်ပါတယ်။

အသစ်တည်ဆောက်လိုက်တဲ့ User ဖြစ်တဲ့ Bob ရဲ့ Role ကို 2 လိုသတ်မှတ်ထားပါတယ်။ ဒါကြောင့် သူဟာ Developer Role ဖြစ်ပြီး User တွေကိုပယ်ဖျက်လိုပေပါဘူး။ Bob နဲ့ Login ဝင်ပြီး Manager ကို Delete လုပ်ကြည့်ပါမယ်။

```
$ curl -X POST localhost:3000/api/users/login \
-d "username=bob&password=123456" -c cookie.txt
{"fullName":"Bob","loginId":"bob","email":"bob@gmail.com","role":"2","password":"7c4a8d09ca3762af61e59520943dc26494f8941b","roleLabel":"Developer","_id":"55b70b342468830b671da93b"}%
```

```
$ curl -X GET localhost:3000/api/users -b cookie.txt
[{"_id":"55b70503d2141ae9733ac0ec","fullName":"John Doe","loginId":"manager","email":"john@example.com","role":3,"password":"7c4a8d09ca3762af61e59520943dc26494f8941b"}, {"fullName":"Bob","loginId":"bob","email":"bob@gmail.com","role":2,"password":"7c4a8d09ca3762af61e59520943dc26494f8941b","roleLabel":"Developer","_id":"55b70b342468830b671da93b"}]%
$ curl -X DELETE localhost:3000/api/users/55b70503d2141ae9733ac0ec -b cookie.txt
Unauthorized%
```

Bob နဲ့ Login ဝင်ပြီး Manager Account ကိုပယ်ဖျက်ဖို့ DELETE Request ပြုလုပ်တဲ့အခါ ပယ်ဖျက်ခွင့် မရှိတဲ့ အတွက် Unauthorized ကိုပဲ လက်ခံရရှိမှာဖြစ်ပါတယ်။ ဒီနည်းနဲ့ကျန် Rout တွေကို ဆက်လက်စမ်းသပ်ကြည့် နိုင်ပြီဖြစ်ပါတယ်။

Code နမူနာ အပြည့်အစုံကို အောက်ပါလိပ်စာမှာ ရယူနိုင်ပါတယ်။

<https://github.com/eimg/issues-api/archive/master.zip>

## Conclusion

ဖော်ပြခဲ့တဲ့နမူနာဟာ REST API နည်းစနစ်ကို လိုက်နာပြီး HTTP Request Method တွေနဲ့ HTTP Status Code တွေ ကို သူ့နေရာနဲ့သူ အသုံးချထားတဲ့ Back-end Service တစ်ခုဖြစ်ပါတယ်။ နားလည်မှတ်သားရလွယ်တဲ့ URL Routing ကိုလည်း အသုံးပြုထားပါတယ်။ Session Based Authentication စနစ်ကိုလည်း အသုံးပြုထားပါတယ်။ ဒါပေမယ့် စွမ်း ဆောင်ရည်ကောင်းမွန်စေဖို့အတွက် Compression နဲ့ Cache လို လုပ်ဆောင်ချက်တွေ ဖြည့်စွက်ပေးသင့်ပါတယ်။ Response တွေကို Plain Text အနေနဲ့ပြန်ပေးတာမျိုးမဟုတ်ပဲ Compress လုပ်ပြီးမှ Response ပြန်ပေးသင့်ပါတယ်။ တစ်ချို့ Cache လုပ်သင့်တဲ့ Content တွေကို Cache လုပ်စေဖို့နဲ့ Cache မလုပ်သင့်တဲ့ Content တွေကို Cache မလုပ်စေဖို့အတွက် Cache-Control Header တွေ Response နဲ့အတူ မှန်ကန် အောင်သတ်မှတ်ပေးသင့်ပါ သေးတယ်။

ဒီ Back-end Service ကို ကိုယ်တိုင် အသုံးပြယုံးသာမက၊ အများကိုလည်း အသုံးပြုခွင့်ပေးတော့မယ်ဆိုရင် ဖြည့်စွက်စဉ်း စားစရာတွေ ထပ်ရှုလာနိုင်ပါသေးတယ်။ ဥပမာ – API Version ကိစ္စ စဉ်းစားရပါမယ်။ လက်ရှိ Service ကပေးထားတဲ့ လုပ်ဆောင်ချက်တွေကို အသုံးချထားတဲ့ Client တွေရှိနေတယ်ဆိုပါစို့။ အဲဒီအချိန်မှာ ကိုယ့် Service ရဲ့ လုပ်ဆောင်ချက် တွေကို အဆင့်မြှင့်ဖို့ လိုအပ်လာတဲ့အခါ ပြုလုပ်လိုက်တဲ့ အပြောင်းအလဲ ကြောင့် လက်ရှိအသုံးပြုနေဆဲ Client တွေအတွက် အဆင်မပြု မဖြစ်ရအောင် စီစဉ်ထားရမှာဖြစ်ပါတယ်။

ပေးထားတဲ့ Service ကို လိုတာထက်ပိုအသုံးပြုထားတဲ့ Client တွေလည်းရှိနိုင်ပါတယ်။ ဥပမာ – မလိုအပ်ပဲနဲ့ တစ်စက္ကန့် တစ်ခါ Request လုပ်နေတဲ့အတွက် Server Resource တွေ ဖြန်းတီးသလိုဖြစ်နေတဲ့ Client တွေ ရှိနိုင်ပါ တယ်။ အဲဒီလို Client တွေကို ဘယ်လိုကန့်သတ်မလဲဆိုတာ စဉ်းစားဖို့လိုအပ်လာနိုင်ပါတယ်။ ကန့်သတ်ထားတဲ့ Request အရေအတွက် ပြည့်တဲ့အခါ အဲဒီ Client ပြုလုပ်တဲ့ Request တွေကို ထပ်လက်မခံတော့တဲ့ လုပ်ဆောင်ချက်မျိုးတွေ ထည့်သွင်းရေးသားဖို့ လိုနိုင်ပါတယ်။

ပြီးတဲ့အခါ Client ရေးသားသူ Developer တွေအတွက် အဆင်ပြေစေဖို့အတွက် API Documentation တွေ ရေးသားပေးဖို့လည်း လိုအပ်နိုင်ပါသေးတယ်။ လက်တွေ Production အဆင့် သွားနိုင်ဖို့အတွက် ဖြည့်စွက်စဉ်းစားရမှာတွေ ရှိနေပေးမယ့်၊ ဖော်ပြထားတဲ့နမူနာပေါ်မှာအခြားပြီး ကိုယ်တိုင်ဆက်လက်ဖြည့်စွက်သွားဖို့ လမ်းစတစ်ခုကို ရရှိသွားနိုင်လိမ့်မယ်လို့ မျှော်လင့်ပါတယ်။

ပြုပြင်ထိမ်းသိမ်းရလွယ်တဲ့ Maintainable Code

ဖြစ်ဖို့အတွက် သီးခြားစီပြင်ဆင်နိုင်တဲ့ အစိတ်အပိုင်းငယ်များ

အဖြစ် ခွဲခြားရေးသားထားရမှာဖြစ်ပါတယ်။

### **Professional Web Developer Course**

HTML5, PHP/MySQL, jQuery/Ajax, Mobile Web စသည့်

Professional Web Developer တစ်ဦး သိရှိထားသင့်သည့်

နည်းပညာများကို စုစုပေါင်းသင်ကြားခြင်းဖြစ်သည်။

ဆက်သွယ်ရန် - (၁၉) ၇၃၀ ၆၅၉ ၆၂

## အခန်း(၁၄) – Building Clients (HTML Templates)

Issue Tracking System အတွက် Back-end Service ရရှိပြီးနောက်မှာ အဲဒီ Back-end Service နဲ့ဆက်သွယ် အလုပ် လုပ်နိုင်တဲ့ Client App တည်ဆောက်ပုံကို ဆက်လက်ဖော်ပြပါမယ်။ BackboneJS ကို သုံးပြီး Web Client တစ်ခု တည်ဆောက်မှာဖြစ်ပါတယ်။ App တည်ဆောက်ပုံအကြောင်း မပြောခင် BackboneJS နဲ့အတူ တွေဖက်အသုံးပြုမယ့် Bootstrap နဲ့ DataTable တို့အကြောင်းကို အရင်ဖော်ပြချင်ပါတယ်။

### 14.1 – Bootstrap 3

Bootstrap ဟာ လူသုံးများတဲ့ CSS Framework တစ်ခုဖြစ်ပါတယ်။ Device အမျိုးမျိုး၊ Screen Size အမျိုးမျိုးနဲ့ သင့်တော်တဲ့ Layout ကို အလိုက်သင့်ပြောင်းလဲ အလုပ်လုပ်ပေးနိုင်တဲ့ Responsive Layout လုပ်ဆောင်ချက် ပါဝင်ပြီး အသင့်အသုံးပြုနိုင်တဲ့ CSS Style နဲ့ JavaScript Component တွေကို စုစည်းပေးထားတဲ့ Framework တစ်ခုဖြစ်ပါ တယ်။ ဒီစာရေးသားနေစဉ် နောက်ဆုံးထွက်ရှိထားတဲ့ Version ကတော့ Version 3.2.0 ပဲဖြစ်ပါ တယ်။

Bootstrap ကို bower install နဲ့ရယူနိုင်သလို [getbootstrap.com](http://getbootstrap.com) မှာကိုယ်တိုင် Download ရယူလိုလည်းရပါ တယ်။ ဒီနေရာမှာ Bootstrap မှာပါဝင်တဲ့ Style နဲ့ Components အားလုံးကို ဖော်ပြမှာမဟုတ်ပါဘူး။ နမူနာ Client App မှာ အသုံးပြုမယ့်အပိုင်းတွေကို အခိုက်ထား ဖော်ပြသွားမှာဖြစ်ပါတယ်။ Bootstrap ရဲ့ Style နဲ့ Component အားလုံးကိုလေ့လာလိုရင် [getbootstrap.com](http://getbootstrap.com) မှာပဲ တစ်ပါတည်းလေ့လာနိုင်ပါတယ်။ သူပေးထား တဲ့ Document က ရိုးရှင်းနားလည်ရတဲ့အတွက်၊ HTML/CSS နဲ့ JavaScript/jQuery အခြေခံရှိတဲ့ မည်သူမဆို အခက်အခဲမရှိ လေ့လာနိုင် မှာပါ။

### Layout

Bootstrap နဲ့ပက်သက်ပြီး ပထမဆုံးလေ့လာသင့်တာကတော့ Layout ဖြစ်ပါတယ်။ Bootstrap က Column (၁၂) ခုပါ ဝင်တဲ့ Layout ကိုအသုံးပြုပြီး အခြေခံ Layout Structure က အခုလို ဖြစ်မှာပါ။

#### HTML

```

1. <div class="container">
2.   <div class="row">
3.     <div class="col-md-12"></div>
4.   </div>
5.   <div class="row">
6.     <div class="col-md-4"></div>
7.     <div class="col-md-8"></div>
8.   </div>
9. </div>

```

Layout မှာပါဝင်တဲ့ Row တွေကို .row Class သတ်မှတ်ပေးဖိုလိုပြီး .row အားလုံးဟာ .container (သို့) .container-fluid အတွင်းမှာ ရှိရပါတယ်။ .container ဟာ Size ပုံသေသတ်မှတ်ထားတဲ့ Fixed-width Element ဖြစ်ပြီး .container-fluid ဆိုရင်တော့ Screen Size ရှိသလောက်နေရာယူမယ့် Fluid-width Element ဖြစ်မှာဖြစ်ပါတယ်။

Bootstrap ၂ Screen Size (င) မျိုးအတွက် ကြိုတင်သတ်မှတ်ပေးထားပါတယ်။ Extra-Small Screen, Small Screen, Medium Screen နဲ့ Large Screen တို့ဖြစ်ပါတယ်။ အခြေခံအားဖြင့် Extra-Small Screen ဆိုတာ Smart Phone Screen Size ကို ဆိုလိုပြီး Small Screen ကတော့ Tablet Screen Size ကို ဆိုလိုပါတယ်။ Medium Screen ကတော့ Desktop/Laptop Screen Size ဖြစ်ပြီး Large Screen ကတော့ Big Screen TV ကို ဆိုလိုပါတယ်။

.container ရဲ့ Fixed-width Size ဟာ Desktop/Laptop မှာ Width 970px ရှိမှာဖြစ်ပါတယ်။ Screen Size ပြောင်းသွားရင် Container လည်းလိုက်ပြောင်းသွားပြီး Smart Phone Screen မှာဆိုရင် Width Auto ဖြစ်သွားမှာဖြစ်ပါတယ်။ .container-fluid ကတော့ ဘယ် Screen Size မှာမဆို Width Auto နဲ့ အလုပ်လုပ်ပေးမှာဖြစ်ပါတယ်။ အလုပ်လုပ်ပုံကိုသာ ဖော်ပြခြင်းဖြစ်ပြီး လက်တွေမှာ Fixed-Width Layout လိုချင်ရင် .container သုံးရပြီး Fluid-Width Layout လိုချင်ရင် .container-fluid သုံးလိုက်ယုံပါဘူး။ Screen Size အမျိုးမျိုးမှာ အဆင်ပြေပြေ ပြောင်းလဲ အလုပ်လုပ်အောင် Bootstrap က ကြိုတင်သတ်မှတ်ထားပေးပြီးဖြစ်ပါတယ်။

.row အတွင်းမှာတော့ Column တွေကို လိုချင်တဲ့ Layout ရရှိအောင်သတ်မှတ်နိုင်ပါတယ်။ အထက်ကန်မူနာ အရ ပထမ Row ထဲမှာ Size အပြည့် (၁၂) ကွက်စာနေရာယူထားတဲ့ Column တစ်ခုရှိနေပြီး ဒုတိယ Row ထဲမှာ တော့ (၄) ကွက် စာနေရာယူထားတဲ့ Column တစ်ခုနဲ့ (၈) ကွက်စာနေရာယူထားတဲ့ Column တစ်ခု နှစ်ခုရှိနေမှာဖြစ်ပါတယ်။ Column တွေ Layout ပုံစံဖော်ပြစေခဲ့ သီးခြားသတ်မှတ်ပေးစရာမလိုပဲ သတ်မှတ်လိုတဲ့ Column Size ကိုသာ သတ်မှတ်ပေးယုံပဲ ဖြစ်ပါတယ်။ ရရှိမယ်ရလဒ် Layout က အခုလိုဖြစ်မှာပါ။

12

4

8

12

### ပုံ (၁၄.၁) - Bootstrap Layout

အထက်ကန်မူနာမှာ Column တွေကိုသတ်မှတ်ဖို့အတွက် .col-md-[n] ဆိုတဲ့ Format ကို အသုံးပြုခဲ့ပါတယ်။ md ဆိုတာ Medium Device ရဲ့ အတိုကောက်ပါ။ Small Screen Device တွေအတွက် sm ကို အသုံးပြုရပြီး Extra Small Screen Device တွေအတွက် xs ကို အသုံးပြုရပါတယ်။ Large Screen Device တွေအတွက်ဆိုရင်တော့ ၁၅ ကို အသုံးပြုရပါတယ်။ အထက်ကန်မူနာကို အခုလိုပြင်ဆင်ရေးသားနိုင်ပါတယ်။

## HTML

```

1. <div class="container">
2.   <div class="row">
3.     <div class="col-md-12"></div>
4.   </div>
5.   <div class="row">
6.     <div class="col-md-8 col-sm-6 col-xs-12"></div>
7.     <div class="col-md-4 col-sm-6 col-xs-12"></div>
8.   </div>
9. </div>

```

ဒုတိယ Row ထဲက Column မှာသတ်မှတ်ထားတဲ့ col-md-8 ရဲ့အမိပိုယ်က၊ Medium Device ဆိုရင် (၈) ကွက် စာ နေရာယူပါဆိုတဲ့ အမိပိုယ်ဖြစ်ပါတယ်။ col-sm-6 ရဲ့အမိပိုယ်ကတော့ Small Screen Device ဆိုရင် (၆) ကွက်စာ နေရာယူပါဆိုတဲ့ အမိပိုယ်ပါ။ col-xs-12 ရဲ့အမိပိုယ်ကတော့ Extra Small Screen ဆိုရင် (၁၂) ကွက်စာ နေရာယူပါလို့ သတ်မှတ်လိုက်ခြင်းဖြစ်ပါတယ်။ ဒါကြောင့် ဒီ Column ဟာ Screen Size ပေါ်မှတည်ပြီး သတ်မှတ်ထားတဲ့အတိုင်း အလိုအလျောက် လိုက်လုပ်ပြောင်းလဲပေးနိုင်တဲ့ Responsive Column တစ်ခုဖြစ်သွား မှာပဲဖြစ်ပါတယ်။ ဒီနည်းနဲ့ Responsive Layout တွေကို Bootstrap နဲ့အလွယ်တစ်ကူ သတ်မှတ်ထားနိုင်ပါ တယ်။

## Styles

Bootstrap ၂ Basic Element တွေဖြစ်ကြတဲ့ h1-h6, p, a စာတဲ့ Element တွေအတွက် သင့်တော်တဲ့ Default Style တွေ ကြိုတင်သတ်မှတ်ထားပေးပြီးဖြစ်ပါတယ်။ အဲဒီ Element တွေကို Bootstrap Style နဲ့ ဖော်ပြနေဖို့ အတွက် Class တွေ သတ်မှတ်ပေးနေစရာမလိုပါဘူး။ Table လို့ Element မျိုးကို Bootstrap Style နဲ့ ဖော်ပြနေ လိုရင်တော့ .table Class သတ်မှတ်ပေးဖို့လိုပါတယ်။ ဥပမာ -

## HTML

```

1. <table class="table">
2.   <tr>
3.     <th> ... </th>
4.     <th> ... </th>
5.   </tr>
6.   <tr>
7.     <td> ... </td>
8.     <td> ... </td>
9.   </tr>
10. </table>

```

ရရှိမယ့်နမူနာရလဒ်ကတော့ အခုလိုဖြစ်ပါတယ်။

#	First Name	Last Name	Username
1	Mark	Otto	@mdo
2	Jacob	Thornton	@fat
3	Larry	the Bird	@twitter

### ပုံ (၁၄.၂) - Bootstrap Style Table

Table တေးပါတ်လည်မှာ Border ပါဝင်သွားစေလိုရင် .table-bordered ကိုအသုံးပြုနိုင်ပြီး Row တွေကို Zebra Stripping နဲ့ဖော်ပြစ်စေလိုရင် .table-striped ကို အသုံးပြုနိုင်ပါတယ်။ ဥပမာ -

#### HTML

```
1. <table class="table table-bordered table-striped">
2.   <tr>
3.     <th> ... </th>
4.     <th> ... </th>
5.   </tr>
6.   <tr>
7.     <td> ... </td>
8.     <td> ... </td>
9.   </tr>
10. </table>
```

Input, Select, Textarea စုတိ Form Element တွေကို Bootstrap Style နဲ့ ဖော်ပြစ်စေလိုရင်တော့ .form-control ကို သတ်မှတ်ပေးရပါတယ်။ အကယ်၍ Input Element တစ်ခု ကို Label Element နဲ့တွဲဖက်ဖော်ပြ စေလို ရင်တော့ .form-group Element တစ်ခုနဲ့ စုစည်းရေးသားပါရပါတယ်။ ဥပမာ -

#### HTML

```
1. <div class="form-group">
2.   <label for="email">Email address</label>
3.   <input type="email" class="form-control" id="email">
4. </div>
```

Button တစ်ခု (သို့မဟုတ်) Link တစ်ခုကို Bootstrap Button ပုံစံဖော်ပြစ်စေလိုရင် .btn Class ကို အသုံးပြုရပါ တယ်။ .btn နဲ့အတူ .btn-default, .btn-primary, .btn-success, .btn-info, .btn-warning, .btn-danger စုတိ Class တွေကိုလည်း တွဲဖက်အသုံးပြုနိုင်ပါတယ်။ အကယ်၍ Button ကို Link ပုံစံ ဖော်ပြစ်စေလိုတယ်ဆိုရင်လည်း .btn-link ကို သုံးနိုင်ပါသေးတယ်။



### ပုံ (၁၄.၂) - Bootstrap Style Buttons

တစ်ခုတက်ပိတဲ့ Button တွေကို အတဲ့လိုက်စုစည်းထားချင်ရင် .btn-group Class သတ်မှတ်ထားတဲ့ Element တစ်ခုတည်းမှာ ထည့်သွင်းစုစည်းထားနိုင်ပါတယ်။

စာလုံးအရောင်အတွက်လည်း သက်ဆိုင်ရာ Element မှာ .text-primary, .text-success, .text-info, .text-warning, .text-danger စသဖြင့် Bootstrap Text Color တွေကို သတ်မှတ်ပေးနိုင်ပါ တယ်။ အရောင်မိမိနှင့်ဖော်ပြခေါ်လိုတဲ့ Text တွေအတွက်လည်း .text-muted ကို အသုံးပြုနိုင်ပါ တယ်။

အလားတူပဲ Element တစ်ခုရဲ့ Background Color ကို Bootstrap Color နဲ့ သတ်မှတ်လိုရင်လည်း .bg-primary, .bg-success, .bg-info စသဖြင့် သတ်မှတ်ပေးနိုင်ပါသေးတယ်။ ဖြည့်စွက်မှတ်သားသင့် တဲ့ Style ကတော့ .pull-left နဲ့ .pull-right တို့ပဲဖြစ်ပါတယ်။ Element တစ်ခုကို ညာဘက်ကပ်စေလိုရင် (float: right ပြုလုပ်စေလိုရင်) .pull-right ကို သုံးနိုင်ပြီး၊ ဘယ်ဘက်ကပ်စေလိုရင်တော့ .pull-left ကို သုံးနိုင်ပါတယ်။ Drop-down Arrow လေးတစ်ခု (▼) ဖော်ပြခေါ်ရင်တော့ အောက်ပါ အတိုင်းသတ်မှတ်နိုင်ပါတယ်။

## HTML

```
<span class="caret"></span>
```

နောက်ထပ်ထူးခြားချက်အနေနဲ့ Element တွေကို တစ်ချို့ Screen Size တွေမှာဖော်ပြခေါ်လိုပြီး၊ တစ်ချို့ Screen Size တွေမှာ မဖော်ပြခေါ်လိုရင် သတ်မှတ်နိုင်တဲ့ Class တွေလည်းရှုပါသေးတယ်။ ဥပမာ - Element တစ်ခုမှာ .hidden-xs လို့ သတ်မှတ်ထားရင် အဲဒေါ် Element ကို Extra Small Device တွေမှာ မဖော်ပြုပဲဖောက်ထားပေးမှာ ဖြစ်ပါတယ်။ အလားတူ Class တွေဖြစ်တဲ့ .hidden-sm, .hidden-md နဲ့ .hidden-lg တို့ကိုလည်း လိုအပ်သလို အသုံးပြုနိုင်ပါတယ်။

ပြောင်းပြန်အားဖြင့် Element တစ်ခုကို ကျွန်း Screen Size တွေမှာ မဖော်ပြခေါ်လိုပဲ၊ Extra Small Device တွေမှာ သာ ဖော်ပြခေါ်လိုရင် .visible-xs-\* ကို အသုံးပြုနိုင်ပါတယ်။ ဒီလိုသတ်မှတ်ပေးလိုက်ရင် Element ကို Extra Small Device မှာသာဖော်ပြပြီး ကျွန်း Screen Size တွေမှာ ဖောက်ထားပေးမှာဖြစ်ပါတယ်။ အလားတူ Responsive Style Class တွေကို အောက်ပါပေါ်ထားမှာ လေ့လာနိုင်ပါတယ်။

	Extra small devices Phones (<768px)	Small devices Tablets (≥768px)	Medium devices Desktops (≥992px)	Large devices Desktops (≥1200px)
.visible-xs-*	Visible	Hidden	Hidden	Hidden
.visible-sm-*	Hidden	Visible	Hidden	Hidden
.visible-md-*	Hidden	Hidden	Visible	Hidden
.visible-lg-*	Hidden	Hidden	Hidden	Visible
.hidden-xs	Hidden	Visible	Visible	Visible
.hidden-sm	Visible	Hidden	Visible	Visible
.hidden-md	Visible	Visible	Hidden	Visible
.hidden-lg	Visible	Visible	Visible	Hidden

### ပုံ (၁၄.၄) - Bootstrap Responsive Helper Classes

## Components

Bootstrap Component တွေထဲမှာ အခြေခံအကျဆုံးကတော့ Icon ဖြစ်ပါတယ်။ Icon တွေအသုံးပြုဖို့အတွက် သီး၏ Image တွေထည့်သွင်းနေစရာမလိုပဲ Bootstrap နဲ့အတူပါဝင်လာတဲ့ Icon တွေကို အသုံးပြုနိုင်ပါတယ်။ ဥပမာ - Button တစ်ခုမှာ Add Icon ကိုထည့်သွင်းဖော်ပြစေလိုရင် အခုလို သတ်မှတ်နိုင်ပါတယ်။

### HTML

```
1. <button class="btn btn-default">
2.   <span class="glyphicon glyphicon-plus-sign"></span>
3.   Add User
4. </button>
```



Icon ထည့်သွင်းဖို့အတွက် <span> Element ကို အသုံးပြုပြီး .glyphicon Class နဲ့အတူ တွဲဖက်အသုံးပြုလို တဲ့ Icon Type ကို တွဲဖက်ပေးရပါတယ်။ နူးနာမှာ .glyphicon-plus-sign ကို အသုံးပြုထားတဲ့အတွက် ရလဒ်မှာ Plus သက်တဲ့ Icon ပါဝင်လာတာကိုတွေ့ရမှာဖြစ်ပါတယ်။ [getbootstrap.com/components](http://getbootstrap.com/components) မှာ Icon စာရင်အပြည့် အစုံကို ဆက်လက်လေ့လာနိုင်ပါတယ်။

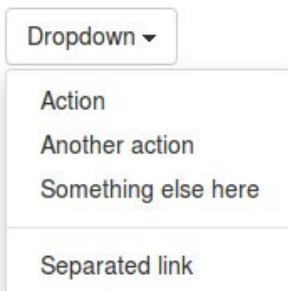
Component တွေထဲမှာ အရေးပါတဲ့ လုပ်ဆောင်ချက်တစ်ခုကတော့ Drop-down Menu ဖြစ်ပါတယ်။ Bootstrap Style Drop-down Menu တစ်ခုရရှိဖို့ အခုလို ရေးသားနိုင်ပါတယ်။

## HTML

```

1. <div class="dropdown">
2.   <button class="btn btn-default" data-toggle="dropdown">
3.     Dropdown <span class="caret"></span>
4.   </button>
5.   <ul class="dropdown-menu">
6.     <li> ... </li>
7.     <li> ... </li>
8.   </ul>
9. </div>

```



Drop-down Trigger လုပ်ပေးမယ့် Link (သို့မဟုတ်) Button တစ်ခုနဲ့ .dropdown-menu Class အသုံးပြုထားတဲ့ <ul> Element တို့ကို Element တစ်ခုအတွင်းမှာ အတူတစ်ကွယ်ညွှန်သွင်းပေးရဖြီး Trigger Link (သို့မဟုတ်) Button မှာ data-toggle Attribute နဲ့ dropdown လိုပြောလိုက်ယုံနဲ့ Bootstrap Style Drop-down Menu တစ်ခုကို ရရှိနိုင်မှာပဲဖြစ်ပါတယ်။ <li> Element တွေထဲက Divider (ကန်လန်ဖြတ်လိုင်း) အဖြစ်အသုံးပြုလို တဲ့ Element မှာ .divider ကို အသုံးပြုနိုင်ပါတယ်။ တစ်ချက်သတိပြုသင့်တာကတော့ Drop-down Menu အပါအဝင် Bootstrap ရဲ့ အချို့လုပ်ဆောင်ချက်တွေက jQuery အသုံးပြု ရေးသားထားခြင်းဖြစ်တဲ့အတွက် jQuery ပါမဲ့ အလုပ်လုပ်မှာဖြစ်ပါတယ်။ နောက်ထပ်အသုံးဝင်တဲ့ Component ကတော့ Navigation Menu Bar ဖြစ်ပါတယ်။ အခုလုံး ထည့်သွင်းနိုင်ပါတယ်။

## HTML

```

1. <nav class="navbar navbar-default">
2.   <div class="container-fluid">
3.     <div class="navbar-header">
4.       <a class="navbar-brand" href="#">Brand</a>
5.     </div>
6.
7.     <ul class="nav navbar-nav">
8.       <li class="active"><a href="#">Link</a></li>
9.       <li><a href="#">Link</a></li>
10.      </ul>
11.
12.    </div>
13.  </nav>

```

Brand    Link    Link

Navigation Bar အတွက် .navbar .navbar-default Class တိုကိုသုံးရပြီး၊ Menu Item တွေအတွက် .nav .navbar-nav Class တို့သတ်မှတ်ထားတဲ့ <ul> Element တစ်ခုကို အသုံးပြုရခြင်းဖြစ်ပါတယ်။ ဖြည့်စွက်ချက်အနေနဲ့ Brand Name ကို Menu Bar ထဲမှာ ထည့်သွင်းဖော်ပြုစေလိုတဲ့အခါ .navbar-header Element ထဲက .navbar-brand Element နဲ့ သတ်မှတ်ပေးနိုင်ပါတယ်။ Menu Item တွေထဲက လက်ရှိရွေးချယ်ထားကြောင်း ပေါ်လွှင်စေလိုတဲ့ Item မှာ .active Class ကို သတ်မှတ်ပေးနိုင်ပါတယ်။

Drop-down Menu တွေ Button တွေနဲ့ Form Element တွေလည်း လိုအပ်ရင် Navbar ထဲမှာ တဲ့ဖက်ထည့်သွင်းအသုံးပြုနိုင်သလို၊ ညာဘက်ကပ်ဖော်ပြုစေလိုတဲ့ Menu ရှိနိုင်လည်း .pull-right Class ကို တဲ့ဖက်အသုံးပြုနိုင်ပါတယ်။ ဒုံးအပြင် Navbar ကိုနူးမှုနားပြထားသလို Light Color နဲ့ မဟုတ်ပဲ Dark Color နဲ့ ဖော်ပြစေလိုရင် .navbar-inverse ကို .navbar-default အစားအစားအသုံးပြုနိုင်ပါတယ်။

အရောင်လေးတွေနဲ့ဖော်ပြတဲ့ Tag Label တွေအသုံးပြုလိုရင် .label Class နဲ့အတူ .label-success, .label-primary, .label-warning, .label-danger စသဖြင့် တဲ့ဖက်အသုံးပြုနိုင်ပါတယ်။

## HTML

```
1. <span class="label label-default">Default</span>
2. <span class="label label-primary">Primary</span>
3. <span class="label label-success">Success</span>
4. <span class="label label-info">Info</span>
5. <span class="label label-warning">Warning</span>
6. <span class="label label-danger">Danger</span>
```

Default    Primary    Success    Info    Warning    Danger

Menu Item (သို့မဟုတ်) အမြားလိုအပ်တဲ့နေရာတွေမှာ Item Count Notification ကို တဲ့ဖက်ဖော်ပြနိုင်အတွက် Badge ကို အသုံးပြုနိုင်ပါတယ်။

## HTML

```
<a href="#">Inbox <span class="badge">42</span></a>
```

Inbox 42

Error Message (သိမဟုတ်) အလားတူ Information တွေဖော်ပြန့်အတွက် .alert ကို အသုံးပြနိုင်ပါတယ်။

### HTML

```
<div class="alert alert-success">...</div>
```

**Well done!** You successfully read this important alert message.

.alert-success အစား .alert-info, .alert-warning, .alert-danger စဲတဲ့ Class ကို လိုအပ်သလိုအစားထိုးအသုံးပြနိုင်ပါတယ်။ Component တွေထဲမှာ နောက်ဆုံးမှတ်သားသင့်တဲ့ Component ကတော့ Panel ဖြစ်ပါတယ်။ အခုလိုရေးသားရပါတယ်။

### HTML

```
1. <div class="panel panel-primary">
2.   <div class="panel-header">Panel Title</div>
3.   <div class="panel-body">Panel Content</div>
4. </div>
```

Panel title

Panel content

နမူနာမှာ .panel နဲ့အတူ .panel-primary ကို တဲ့ဖော်အသုံးပြထားပါတယ်။ ထုံးစံအတိုင်း .panel-primary အစား .panel-success, .panel-info, .panel-warning, .panel-danger စဲ ကို တဲ့ဖော်အသုံးပြနိုင်ပါတယ်။ Panel အတွင်းမှာ .panel-header, .panel-body နဲ့ .panel-footer သတ်မှတ်ထားတဲ့ Element တွေကို အသီးသီးထည့်သွင်းအသုံးပြနိုင်ပါတယ်။

ကျွန်ုတ်တို့ရဲ့ Issue Tracking System Client App ကိုတည်ဆောက်တဲ့အခါ ဒီလုပ်ဆောင်ချက်တွေအားလုံးကို အသုံးပြုသွားမှာဖြစ်ပါတယ်။

## 14.2 – jQuery DataTables

DataTables ဟာ jQuery Plugin တစ်ခုဖြစ်ပြီး ရုံးရုံး HTML Table တစ်ခုကို Sorting, Filter, Paging လုပ်ဆောင် ချက်တွေပါဝင်တဲ့ Data Grid တစ်ခုဖြစ်အောင် ပြောင်းပေးနိုင်ပါတယ်။ bower install နဲ့ ရယူနိုင်သလို [datatables.net](http://datatables.net) မှာလည်း ကိုယ်တိုင် Download ရယူနိုင်ပါတယ်။

Show 10 entries Search:

Name	Position	Office	Age	Start date	Salary
Airi Satou	Accountant	Tokyo	33	2008/11/28	\$162,700
Angelica Ramos	Chief Executive Officer (CEO)	London	47	2009/10/09	\$1,200,000
Ashton Cox	Junior Technical Author	San Francisco	66	2009/01/12	\$86,000
Bradley Greer	Software Engineer	London	41	2012/10/13	\$132,000
Brenden Wagner	Software Engineer	San Francisco	28	2011/06/07	\$206,850
Brielle Williamson	Integration Specialist	New York	61	2012/12/02	\$372,000
Bruno Nash	Software Engineer	London	38	2011/05/03	\$163,500
Caesar Vance	Pre-Sales Support	New York	21	2011/12/12	\$106,450
Cara Stevens	Sales Assistant	New York	46	2011/12/06	\$145,600
Cedric Kelly	Senior Javascript Developer	Edinburgh	22	2012/03/29	\$433,060

Name Position Office Age Start date Salary

Showing 1 to 10 of 57 entries

Previous 1 2 3 4 5 6 Next

### ပုံ (၁၄.၅) - DataTables Example

ပုံ (၁၄.၅) မှာဖော်ပြထားသလို လုပ်ဆောင်ချက်အပြည့်အစုံပါဝင်တဲ့ Data Grid ရရှိဖို့အတွက် DataTables ရဲ့ အကူ အညီနဲ့ အခုလို အလွယ်တစ်ကူရေးသားနိုင်ပါတယ်။

#### JavaScript

```
$( "table" ).dataTables();
```

Data Grid ပြောင်းလိုတဲ့ Table ကို Select လုပ်ပြီး dataTables() Function တဲ့ဖက်ပေးလိုက်ခြင်းဖြစ်ပါတယ်။ DataTables ရဲ့ လုပ်ဆောင်ချက်အပြည့်အစုံကိုတော့ [datatables.net](http://datatables.net) မှာ ကိုယ်တိုင်ပဲ ဆက်လက်လေ့လာလိုက်ပါ။ ဒီနေရာမှာတော့ ကျွန်ုတ်တို့ ထည့်သွင်းအသုံးပြုမယ့် ထူးခြားချက်နှစ်ချက်ကို ဖော်ပြလိုပါတယ်။

ပထမတစ်ချက်ကတော့ Sorting ဖြစ်ပါတယ်။ Sorting စီတဲ့အခါ ဖော်ပြထားတဲ့အတိုင်း ကြီးစဉ်ပေးလိုက် (သို့မဟုတ်) ငယ်စဉ်ကြီးလိုက်စီစဉ်တဲ့အလုပ်ငန်းကို DataTables က အလိုအလျောက်လုပ်ပေးနိုင်ပါတယ်။ ပြဿနာက ဖော်ပြတဲ့ တန်ဖိုးကတစ်ခု၊ အသုံးပြုစီစဉ်စေလိုတဲ့တန်ဖိုးကတစ်ခုဆိုတဲ့ လိုအပ်ချက်မျိုး ရှိတက်ပါတယ်။ ဥပမာ - Issue Status ကို အသုံးပြုသူ ဖတ်ရှုရလွယ်ကူစေဖို့အတွက် New, Assigned, Doing, Done, Close စသဖို့ ဖော်ပြပေမယ့်၊ Sorting စီ တဲ့အခါမှာတော့ ငယ်စဉ်ကြီးလိုက်စီမယ်ဆိုရင် Assigned, Close, Doing, Done, New ဆိုပြီး A-Z စီပေးသွားမှာပါ။ လက်တွေမှာ ဖော်ပြထားတဲ့တန်ဖိုးအတိုင်း စီလိုမရပဲ New (0), Assigned (1), Doing (2) စသဖို့ သက်ဆိုင်ရာ တန်ဖိုးအလိုက်စီပေးဖို့လိုအပ်ပါတယ်။ ဒီပြဿနာကို ပြေလည်စေဖို့အတွက် DataTables က order-value ဆိုတဲ့ Attribute တစ်ခုကို အသုံးပြုပါတယ်။ ဥပမာ -

## HTML

```

1. <table>
2.   <tr><td order-value="1">Assigned</td></tr>
3.   <tr><td order-value="4">Close</td></tr>
4.   <tr><td order-value="0">New</td></tr>
5.   <tr><td order-value="3">Done</td></tr>
6.   <tr><td order-value="2">Doing</td></tr>
7. </table>

```

နမူနာ Table မှာ ဖော်ပြုမယ့်တန်ဖိုးတွေက Assigned, Close, New, Done, Doing စတဲ့တန်ဖိုးတွေဖြစ်ပေးမယ်  
လက်တွေ့ Sorting စိတဲ့အခါမှာတော့ DataTables က အဲဒီတန်ဖိုးတွေနဲ့မစီပဲ order-value မှာပေးထားတဲ့  
တန်ဖိုးကို အသုံးပြု စိပေးသွားမှာပဲဖြစ်ပါတယ်။

နောက်တစ်ခုကတော့ Device အမျိုးမျိုးမှာ အလုပ်လုပ်စေနိုင်တဲ့ Responsive Table Design ဖြစ်ပါတယ်။  
DataTables နဲ့ Bootstrap ဟာ သီးခြား Library တွေဖြစ်ကြတဲ့အတွက် တစ်ခုနဲ့တစ်ခု အသုံးပြုတဲ့ Style တွေမ<sup>တူကြပါဘူး။</sup> ဒါကြောင့် DataTables ရဲ့ ဖော်ပြပုံကို Bootstrap Table Style ကိုသုံးပြီး ဖော်ပြစေခြင်းနဲ့ Screen  
Size ပေါ်မှတည်ပြီး အဆင်ပြေအောင် အလိုက်သင့်ပြောင်းလဲဖော်ပြနိုင်စေခြင်းဟာလည်း လိုအပ်ချက်တစ်ခုဖြစ်  
ပါတယ်။ ဒီပြဿနာပြေလည်း ဖို့အတွက် DataTables Bootstrap 3 Plugin နဲ့ DataTables Responsive  
Plugin တို့ကို အသုံးပြုပေးဖို့လိုအပ်ပါတယ်။

- DataTables Bootstrap - <https://datatables.net/examples/styling/bootstrap.html>
- DataTables Responsive - <https://datatables.net/extensions/responsive/>

Name	Position	Office	Age
 Airi Satou	Accountant	Tokyo	33
 Angelica Ramos	Chief Executive Officer (CEO)	London	47
 Ashton Cox	Junior Technical Author	San Francisco	66
<b>Start date:</b> 2009/01/12			
<b>Salary:</b> \$86,000			
 Bradley Greer	Software Engineer	London	41
 Brenden Wagner	Software Engineer	San Francisco	28
 Brielle Williamson	Integration Specialist	New York	61
 Bruno Nash	Software Engineer	London	38
 Caesar Vance	Pre-Sales Support	New York	21
 Cara Stevens	Sales Assistant	New York	46

ပုံ (၁၄.၆) - DataTables Responsive Example

ဒီ Plugin နှစ်ခုကိုထပ်ည့်သွင်းလိုက်ရင်တော့ DataTables ရဲ ဖော်ပြပုံအသွင်အပြင်ကို Bootstrap Style နဲ့ဖော်ပြပုံအသွင်အပြင်ကို ပေးသွားမှာဖြစ်ပြီး၊ Screen Size သေးသွားတဲ့အခါ ငါ (၁၄.၆) မှာ နမူနာပြထားသလို တစ်ချို့ Table Column တွေကို ဖျောက်ထားပြီး၊ Button တစ်ခုကို နိုင်လိုက်မှသာ ဖော်ပြအောင်ပေးမှာဖြစ်တဲ့အတွက် Screen Size သေးတဲ့ Device တွေမှာလည်း Data Grid က အဆင်ပြေပြေ ဆက်လက်အလုပ်လုပ်ပေးနိုင်မှာပဲဖြစ်ပါတယ်။

ဒီအခန်းမှာ နမူနာဖော်ပြမယ့် Code တွေဟာ မပြည့်စုံသေးတဲ့အတွက် လက်တွေကူးယူ စမ်းသပ်လို့ ရှုံးမှာမဟုတ်ပါဘူး။ နောက်တစ်ခန်းမှာ ဆက်လက်ဖော်ပြတဲ့ Code တွေနဲ့ ပေါင်းစပ်ပြီးမှသာ လက်တွေ စမ်းသပ်လို့ရမှာ ဖြစ်ပါတယ်။ Code နမူနာ အပြည့်အစုံကို အောက်ပါ လိပ်စာမှာ ကြိုတင် Download ရယူထားနိုင်ပါတယ်။

<https://github.com/eimg/issues-backbone/archive/master.zip>

#### 14.3 – Directory Structure for Web Client

အခုခံရင် လိုအပ်တဲ့နည်းပညာနှစ်ခုကို ဖြည့်စွက်ဖော်ပြပြီးပြီး မြတ်နည်းပါပြီး ဒီလို တည်ဆောက်ရာမှာ ထမ္မားချုံးအနေနဲ့ လိုအပ်တဲ့ Directory Structure နဲ့ HTML Template တွေကို အရင် တည်ဆောက်ပါမယ်။ Directory တစ်ခုကို issues-backbone အမည်နဲ့ (သို့) နှစ်သက်ရာအမည်နဲ့ တည်ဆောက် ပြီး၊ လိုအပ်တဲ့ Package တွေကို Bower နဲ့ အခုလိုရယူပါမယ်။

```
$ mkdir issues-backbone && cd issues-backbone
$ bower install backbone jquery underscore bootstrap datatables
datatables-responsive datatables-bootstrap3 bootstrap3-wysiwyg jquery-validate moment
```

အသုံးပြုဖို့လိုအပ်တဲ့ Backbone, jQuery, Underscore, Bootstrap, DataTables, DataTables Responsive Plugin, DataTables Bootstrap Plugin စတဲ့ Library တွေကို ရယူထားပါတယ်။ ဒါအပြင် Rich Text Editor လုပ်ဆောင်ချက် ရရှိနိုင်ဖို့အတွက် Bootstrap3 Wysiwyg, Form Validation ပြုလုပ်နိုင်ဖို့အတွက် jQuery Validate နဲ့ ရက်စွဲတွေကို Format လုပ်ဖော်ပြနိုင်ဖို့အတွက် MomentJS တိုကိုလည်း တဲ့ဖက်ထည့်သွင်းထားပါတယ်။

ရရှိလာတဲ့ Directory Structure က အခုလိုဖြစ်မှာပါ –

```
issues-backbone
└── bower_components
    ├── backbone
    ├── bootstrap
    ├── bootstrap3-wysiwyg
    ├── datatables
    ├── datatables-bootstrap3
    └── datatables-responsive
```

```

  └── jquery
  └── jquery-validate
  └── moment
  └── underscore

```

ဆက်လက်ပြီး လိုအပ်တဲ့ ဖိုင်တွေကို အခုလိုဖြည့်စွက်ပေးရပါမယ်။

```

issues-backbone
  └── app.js
  └── index.html
  └── issues.js
  └── users.js
bower_components
templates
  └── issue-detail.html
  └── issue-edit.html
  └── issue.html
  └── issue-list.html
  └── issue-new.html
  └── login.html
  └── nav.html
  └── user-edit.html
  └── user.html
  └── user-list.html
  └── user-new.html
  └── user-profile.html
views
  └── appView.js
  └── issueDetailView.js
  └── issueEditView.js
  └── issueListView.js
  └── issueNewView.js
  └── loginView.js
  └── navView.js
  └── userEditView.js
  └── userListView.js
  └── userNewView.js
  └── userProfileView.js

```

app.js မှာ Client App အတွက်လိုအပ်တဲ့ အခြေခံ Function တွေကို ရေးသားမှာဖြစ်ပြီး issues.js နဲ့ users.js တို့မှာ သက်ဆိုင်ရာ Model နဲ့ Collection တွေကို ရေးသားမှာဖြစ်ပါတယ်။ index.html ကတော့ Main Template ဖိုင်အဖြစ်လိုအပ်တဲ့ Script နဲ့ Style တွေကိုစုစည်းချက်ဆက်ထားမှာ ဖြစ်ပါတယ်။ views Directory ထဲမှာတော့ လိုအပ်တဲ့ Backbone View တွေကို စုစည်းထားမှာဖြစ်ပြီး templates Directory ထဲမှာ View နဲ့အတူ တဲ့ဖက်အသုံးပြုရမယ့် HTML Template တွေကို ကြိုတင်ရေးသားထားမှာဖြစ်ပါတယ်။ ဒီအခန်းမှာ အဲဒီ Template တွေ ကိုဖော်ပြပြီး နောက်တစ်ခန်းမှာ Model-View-Collection တွေတည်ဆောက်ပုံကို ဆက်လက်ဖော်ပြသွားမှာဖြစ်ပါတယ်။

14.4 – Issue Detail Template – `templates/issue-detail.html`

ဆက်လက်ပြီး Issue တစ်ခုစီရဲ အသေးစိတ်အချက်အလက်တွေကို ဖော်ပြပေးတဲ့ Template ဖြစ်တဲ့ Issue Template တစ်ခုကို templates/issue-detail.html မှာ ရေးသားပါမယ်။

## Template

```
1. <div class="row">
2.   <div class="col-md-8 col-sm-12 col-xs-12">
3.     <div class="panel"
4.       <% if(priority == 0) { %>
5.         panel-info
6.       <% } else if (priority == 1) { %>
7.         panel-warning
8.       <% } else { %>
9.         panel-danger
10.      <% } %>
11.      issue-detail">
12.      <div class="panel-heading">
13.        <h2>
14.          <% if(type == 0) { %>
15.            <span class="glyphicon glyphicon-fire"></span>
16.          <% } else if (type == 1) { %>
17.            <span class="glyphicon glyphicon-heart-empty"></span>
18.          <% } else { %>
19.            <span class="glyphicon glyphicon-asterisk"></span>
20.          <% } %>
21.          <%= summary %>
22.          <span class="pull-right small">#<%= _id %></span>
23.        </h2>
24.      </div>
25.      <div class="panel-body">
26.        <%= detail %>
27.      </div>
28.      <div class="panel-footer clearfix">
29.        <span class="text-muted"><%= moment(submittedAt).fromNow() %></span>
30.        <div class="btn-group pull-right">
31.          <button class="btn btn-default btn-sm" id="close-issue">
32.            <span class="glyphicon glyphicon-ok text-success"></span>
33.            Close
34.          </button>
35.          <button class="btn btn-default btn-sm" id="edit-issue">
36.            <span class="glyphicon glyphicon-edit text-info"></span>
37.            Edit
38.          </button>
39.          <button class="btn btn-default btn-sm" id="delete-issue">
40.            <span class="glyphicon glyphicon-trash text-danger"></span>
41.            Remove
42.          </button>
43.        </div>
44.      </div>
45.    </div>
46.
47.    <hr>
48.    <% if(typeof comments !== "undefined") { %>
49.      <% _.each(comments, function(comment, id) { %>
50.
51.        <div class="panel panel-default small comment">
```

```
52.     <div class="panel-heading">
53.         <span class="glyphicon glyphicon-comment"></span>
54.         <b class="text-primary"><%= comment.authorName %></b>
55.         <time class="text-muted">
56.             <%= moment(commentsubmittedAt).fromNow() %>
57.         </time>
58.
59.         <a href="#" title="" class="pull-right
60.                                         text-danger
61.                                         delete-comment"
62.                                         data-index="<%= id %>">
63.             <span class="glyphicon glyphicon-remove"></span>
64.         </a>
65.
66.         <span class="pull-right text-muted">#<%= id %></span>
67.     </div>
68.     <div class="panel-body">
69.         <p>
70.             <%= comment.comment %>
71.         </p>
72.     </div>
73. </div>
74. <% }) %>
75. <% } %>
76.
77. <div class="comment-form">
78.     <div class="form-group">
79.         <label>Comment</label>
80.         <textarea class="form-control"
81.                     placeholder="Comment" id="comment"></textarea>
82.     </div>
83.     <div class="form-group">
84.         <button type="submit" class="btn btn-default" id="post-comment">
85.             <span class="glyphicon glyphicon-comment"></span>
86.             Add Comment
87.         </button>
88.     </div>
89. </div>
90. </div>
91.
92. <div class="col-md-4 col-sm-12 col-xs-12 sidebar">
93.     <div class="btn-group btn-breadcrumb">
94.         <a href="#" class="btn btn-danger">
95.             <% if(type == 0) { %>
96.                 <span class="glyphicon glyphicon-fire"></span>
97.             <% } else if (type == 1) { %>
98.                 <span class="glyphicon glyphicon-heart-empty"></span>
99.             <% } else { %>
100.                 <span class="glyphicon glyphicon-asterisk"></span>
101.             <% } %>
102.         </a>
103.         <a href="#" class="btn btn-danger">Type</a>
104.
105.         <a href="#" class="btn btn-default" data-toggle="dropdown">
106.             <%= typeLabel %> <span class="caret"></span>
107.         </a>
108.         <ul class="dropdown-menu dropdown-menu-right">
109.             <% _.each(app.issueType, function(label, type) { %>
110.                 <li>
```

```

111.         <a href="#" class="change-type" data-value="<% type %>">
112.             <%= label %>
113.         </a>
114.     </li>
115.     <% }) %>
116.   </ul>
117. </div>
118.
119.
120. <div class="btn-group btn-breadcrumb">
121.   <a href="#" class="btn btn-warning">
122.     <span class="glyphicon glyphicon-sort-by-attributes-alt"></span>
123.   </a>
124.   <a href="#" class="btn btn-warning">Priority</a>
125.   <a href="#" class="btn btn-default" data-toggle="dropdown">
126.     <%= priorityLabel %> <span class="caret"></span>
127.   </a>
128.   <ul class="dropdown-menu dropdown-menu-right">
129.     <% _.each(app.issuePriority, function(label, priority) { %>
130.       <li>
131.         <a href="#" class="change-priority"
132.             data-value="<% priority %>"><%= label %></a>
133.       </li>
134.     <% }) %>
135.   </ul>
136. </div>
137.
138.
139. <hr>
140.
141. <div class="btn-group btn-breadcrumb">
142.   <a href="#" class="btn btn-success">
143.     <span class="glyphicon glyphicon-user"></span>
144.   </a>
145.   <a href="#" class="btn btn-success">Assign</a>
146.   <a href="#" class="btn btn-default" data-toggle="dropdown">
147.     <%= assignedToLabel %> <span class="caret"></span>
148.   </a>
149.   <ul class="dropdown-menu dropdown-menu-right">
150.     <% _.each(users, function(user, id) { %>
151.       <li>
152.         <a href="#" class="change-assign"
153.             data-value="<% user._id %>"><%= user.fullName %></a>
154.       </li>
155.     <% }) %>
156.   </ul>
157. </div>
158.
159. <div class="btn-group btn-breadcrumb">
160.   <a href="#" class="btn btn-info">
161.     <span class="glyphicon glyphicon-arrow-right"></span>
162.   </a>
163.   <a href="#" class="btn btn-info">Status</a>
164.   <a href="#" class="btn btn-default" data-toggle="dropdown">
165.     <%= statusLabel %> <span class="caret"></span>
166.   </a>
167.   <ul class="dropdown-menu dropdown-menu-right">
168.     <% _.each(app.issueStatus, function(label, status) { %>
169.       <li>
170.         <a href="#" class="change-status" data-value="<% status %>">

```

```
171.             <%= label %></a>
172.         </li>
173.     <% } %>
174.   </ul>
175. </div>
176.
177. <hr>
178.
179. <div class="alert text-muted meta">
180.   <p>
181.     Submitted date: <b>
182.       <%= moment(submittedAt).format("DD-MM-YYYY h-mm-ss a") %>
183.     </b>
184.   </p>
185.   <p>Last modified: <b>
186.     <% if(typeof modifiedAt !== 'undefined') { %>
187.       <%= moment(modifiedAt).format("DD-MM-YYYY h-mm-ss a") %>
188.     <% } %>
189.   </b></p>
190.   <p>Submitted by: <b>
191.     <% var submitter = app.userList.get({ _id: submittedBy }) %>
192.     <%= submitter.get("fullName") %>
193.   </b></p>
194. </div>
195. </div>
196. </div>
```

HTML Structure အပြည့်စုထည်သွင်းထားခြင်းမရှိသလို၊ အသုံးပြုလိုတဲ့ Library တွေကို ချိတ်ဆက်ထားခြင်းလည်း မရှိ တာကိုသတိပြုပါ။ ဒါ Template ဟာ သူချေည်းသက်သက် အလုပ်လုပ်တဲ့ Page တစ်ခုမဟုတ်ပဲ နောက်တော့မှ Backbone View ကနေ ရယ်အသုံးပြုနိုင်ဖို့ ကြိုတင်ရေးသားထားပေးတဲ့ Template တစ်ခုဖြစ်ပါတယ်။ Backbone View နဲ့ ချိတ်ဆက်ပြီးရင် ရရှိမယ့် ရလဒ်နှုန်းကို ပဲ (၁၄.၇) မှာလေ့လာနိုင်ပါတယ်။

With Bootstrap 2, we added optional mobile friendly styles for key aspects of the framework.

1. Step 1  
2. Step 2  
3. Step 3

3 days old

Close  Edit  Remove

**John Smith** 2015-05-14 #1 X

With Bootstrap 2, we added optional mobile friendly styles for key aspects of the framework.

**James Doe** 2015-05-14 #2 X

With Bootstrap 2, we added optional mobile friendly styles for key aspects of the framework.

**Comment**

Comment

Add Comment

**Type** Bug X

**Priority** High X

**Assign** John Doe X

**Status** Doing X

Submitted date: 2015-05-11  
Last modified: 2015-05-11  
Submitted by: John Doe

Column နှစ်ခုပါဝင်တဲ့ Layout ဖြစ်ပြီး၊ ဘယ်ဘက် Column ထဲမှာ Bootstrap Panel တစ်ခုနဲ့ Issue Summary နဲ့ Detail တို့ကို ဖော်ပြထားပါတယ်။ Panel ရဲ့ အရောင်ဟာ .panel-info, .panel-warning, .panel-danger စသဖြင့်လက်ရှိ Issue ရဲ့ Priority ပေါ်မှုတည်ပြီး ဖော်ပြအောင် လိုင်းနံပါတ် (၄) မှာ စီစစ် ဖော်ပြထားစေတာကို သတိပြုကြည့်ပါ။

Issue Summary ရှေ့ကဖော်ပြတဲ့ Icon ကိုလည်း Issue Type ပေါ်မှုတည်ပြီးပြောင်းလဲဖော်ပြအောင် လိုင်းနံပါတ် (၁၄) မှာ စီစစ်ထားပါသေးတယ်။ ပြီးတဲ့အခါ Panel Footer ထဲမှာ .btn-group နဲ့ Close, Edit, Delete Button တွေကို တန်းစီထည့်သွင်းထားပါတယ်။

Main Panel ရဲ့အောက်မှာတော့ နောက်ထပ် Panel များနဲ့ Comment တွေကို (ရှိရင်) တန်းစီဖော်ပြသွားမှာဖြစ်ပါတယ်။ အောက်ဆုံးမှာတော့ Comment အသစ်ထည့်သွင်းနိုင်ဖို့အတွက် Textarea တစ်ခုနဲ့ Button တစ်ခုကို ပေးထားပါတယ်။

ညာဘက် Column ထဲမှာတော့ Issue ရဲ့ Type, Priority, Assigned To နဲ့ Status တို့ကို ပြောင်းလဲသတ်မှတ်တိုင် ဖို့ အတွက် Bootstrap Dropdown တွေကို တန်းစီနေရာချထားပါတယ်။ Column နှစ်ခုကို သတ်မှတ်တဲ့အခါ Medium Screen မှာဆိုရင် 8-4 အချိုးနဲ့ဖော်ပြပြီး Small Screen နဲ့ Extra-Small Screen တွေမှာဆိုရင် 12 Column အပြည့် ဖော်ပြဖို့သတ်မှတ်ထားတဲ့အတွက် ဒါ Layout ဟာ Screen Size အမျိုးမျိုးမှာ သင့်တော်အောင် ဖော်ပြပေးနိုင်တဲ့ Responsive Layout တစ်ခုလည်းဖြစ်ပါတယ်။

#### 14.5 – Issue Edit Template – `templates/issue-edit.html`

Issue တစ်ခုကို ပြင်ဆင်တဲ့အခါ အသုံးပြုနိုင်ဖို့အတွက် Form တစ်ခုကို ဆက်လက်ရေးသား ပါမယ်။ `templates/issue-edit.html` မှာ အခုလိုရေးသားပေးပါ။

##### Template

```

1. <form action="#" id="issue-edit-form">
2.   <div class="alert alert-info">
3.     Issue ID: <b><%= _id %></b>
4.   </div>
5.   <div class="form-group">
6.     <label for="">Summary *</label>
7.     <input type="text" class="form-control"
8.       placeholder="Summary" id="summary" value="<%= summary %>">
9.   </div>
10.
11.  <div class="form-group">
12.    <label for="">Detail / Steps to Reproduce</label>
13.    <div class="form-control editor"
14.      placeholder="Detail / Step to reproduce" id="detail">
15.        <%= detail %></div>
16.    </div>
17.
18.  <button type="submit" class="btn btn-primary btn-lg" id="update-issue">
19.    <span class="glyphicon glyphicon-edit"></span>
20.    Update

```

```

21.  </button>
22.  <button class="btn btn-link" id="close-edit">Cancel</button>
23. </form>
24.
25. <script>
26.  $(function() {
27.      $('.editor').wysihtml5({
28.          toolbar: {
29.              "font-styles": false,
30.              "emphasis": true,
31.              "lists": true,
32.              "html": false,
33.              "link": true,
34.              "image": false,
35.              "color": false,
36.              "blockquote": false,
37.              "size": "sm"
38.          }
39.      });
40.  });
41. </script>

```

View နဲ့ချိတ်ဆက်ပြီးတဲ့ အခါ ရရှိလာမယ့် ရလဒ်က အခုလိုဖြစ်မှာပါ။

Issue ID: #55b70503d2141ae9733ac0ec

#### Summary \*

Pagination option in item list API

#### Detail / Steps to Reproduce

**Bold** *Italic* Underline     

With Bootstrap 2, we added optional mobile friendly styles for key aspects of the framework.

1. Step 1
2. Step 2
3. Step 3

 Update

Cancel

#### ပုံ (၁၄.၈) - Issue Edit Template

ဒီ Template မှာတော့ Column တစ်ခုသာရှိပြီး Issue Summary, နဲ့ Issue Detail ပြင်ဆင်ရာမှာအသုံးပြန်ငိုး အတွက် Form Input တွေကို သတ်မှတ်ထားခြင်း ဖြစ်ပါတယ်။ Wysihtml5 Plugin အကူအညီနဲ့ Issue Detail Textarea ကို Rich Text Editor ပြောင်းလဲထားတာကို သတိပြုပါ။

## 14.6 – Issue List Templates – `templates/issue-list.html, issue.html`

Issue List ဖော်ပြတဲ့ Template ကိုတော့ နှစ်ပိုင်းခဲ့ထားပါတယ်။ Issue List Table Template နဲ့ Issue List Item Table Row Template တို့ဖြစ်ပါတယ်။ Issue List Table Template ကို `templates/issue-list.html` မှာ အခုလိုရေးသားရပါမယ်။

### Template

```

1. <table id="issue-list-table"
2.   class="table table-bordered" cellspacing="0" width="100%">
3.   <thead>
4.     <tr>
5.       <th>#</th>
6.       <th width="50%">Summary</th>
7.       <th>Type</th>
8.       <th>Assigned To</th>
9.       <th>Priority</th>
10.      <th>Status</th>
11.    </tr>
12.  </thead>
13.
14.  <tbody>
15.
16.  </tbody>
17. </table>
18.
19. <script>
20. $(function() {
21.   $("#issue-list-table").dataTable({
22.     responsive: true
23.   });
24. });
25. </script>

```

ဘာမှတဲးထူးထွေထွေလုပ်ဆောင်ချက်မပါဝင်တဲ့ Table Template အလွတ်တစ်ခုသာဖြစ်ပါတယ်။ DataTables ကိုသုံးပြီး Table ကို Data Grid ပြောင်းပေးဖို့ကိုတော့ ထည့်သွင်းသတ်မှတ်ထားပါတယ်။ ဆက်လက်ပြီး Issue List Item Table Row Template ကို `templates/issue.html` မှာ အခုလိုရေးသားရပါမယ်။

### Template

```

1. <td><%= index %></td>
2. <td>
3.   <% if (status == 4) { %>
4.     <a href="#" data-id="<%= _id %>" class="issue-summary">
5.       <del><%= summary %></del></a>
6.     <% } else { %>
7.       <a href="#" data-id="<%= _id %>" class="issue-summary">
8.         <%= summary %></a>
9.     <% } %>
10.   </td>
11.   <td data-order="<%= type %>">
12.     <% if(type == 0) { %>
13.       <span class="glyphicon glyphicon-fire"></span>

```

```

14. <% } else if (type == 1) { %>
15.   <span class="glyphicon glyphicon-heart-empty"></span>
16. <% } else { %>
17.   <span class="glyphicon glyphicon-asterisk"></span>
18. <% } %>
19. <%= typeLabel %>
20. </td>
21. <td><%= assignedToLabel %></td>
22. <td data-order="<%= priority %>"><%= priorityLabel %></td>
23. <td data-order="<%= status %>">
24.   <% if(status == 0) { %>
25.     <span class="label label-danger"><%= statusLabel %></span>
26.   <% } else if (status == 1) { %>
27.     <span class="label label-warning"><%= statusLabel %></span>
28.   <% } else if (status == 2) { %>
29.     <span class="label label-info"><%= statusLabel %></span>
30.   <% } else if (status == 3) { %>
31.     <span class="label label-success"><%= statusLabel %></span>
32.   <% } else if (status == 4) { %>
33.     <span class="label label-default"><%= statusLabel %></span>
34.   <% } %>
35. </td>

```

Issue Type အလိုက် သင့်တော်တဲ့ Icon လေးတွေ ဖော်ပြပေးပို့နဲ့ Issue Status ပေါ်မှတည်ပြီး သင့်တော်တဲ့ Label Tag လေးတွေဖော်ပြပေးဖို့ စိစစ်ရေးသားထားတဲ့ Template တစ်ခုဖြစ်ပါတယ်။ သက်ဆိုင်ရာ View နဲ့ ချိတ်ဆက်ပြီးရင် ရရှိလာမယ့်ရလဒ်က ဒီလိုပါ။

Show 10 entries Search:

#	Summary	Type	Assigned To	Priority	Status
1	Potential XSS vulnerability in order form	* Other	John Doe	Urgent	Assigned
2	Pagination option in item list API	♡ Feature	John Doe	Low	Won't Fix
3	Mispell in checkout form	* Other	John Doe	Low	Doing
4	Dropdown menu no longer work after viewing photo slideshow	🔥 Bug	John Doe	Urgent	New
5	Overlay windows cover only top of the page	🔥 Bug	John Doe	High	Done

Showing 1 to 5 of 5 entries Previous 1 Next

### ပုံ (၁၄.၃) - Issue List Template

DataTables ရဲ့ အကူအညီနဲ့ Sorting, Paging နဲ့ Filter လုပ်ဆောင်ချက်တွေ ပါဝင်တဲ့ Issue List Table ကို Bootstrap Table Style နဲ့ ဖော်ပြပေးမှာ ဖြစ်ပါတယ်။

## 14.7 – New Issue Template – `templates/issue-new.html`

ဆက်လက်ပြီး Issue အသစ်ထည့်သွင်းတဲ့အခါမှာ အသုံးပြုရမယ့် Form အတွက် Template တစ်ခုကို `templates/issue-new.html` မှာ အခုံလိုပေးပေးပါ။

### Template

```

1. <form action="#" id="new-issue-form">
2.   <div class="form-group">
3.     <label for="">Summary *</label>
4.     <input type="text" class="form-control"
5.       placeholder="Summary" id="summary">
6.   </div>
7.
8.   <div class="form-group">
9.     <label for="">Detail / Steps to Reproduce</label>
10.    <div class="form-control editor" id="detail"></div>
11.  </div>
12.
13.  <div class="form-group">
14.    <label for="">Category</label>
15.    <select class="form-control" id="type">
16.      <% _.each(types, function(type, val) { %>
17.        <option value="<%= val %>"><%= type %></option>
18.      <% }); %>
19.    </select>
20.  </div>
21.
22.  <div class="form-group">
23.    <label for="">Priority</label>
24.    <select class="form-control" id="priority">
25.      <% _.each(priorities, function(type, val) { %>
26.        <option value="<%= val %>"><%= type %></option>
27.      <% }); %>
28.    </select>
29.  </div>
30.
31.  <button type="submit" class="btn btn-primary btn-lg" id="add-button">
32.    <span class="glyphicon glyphicon-plus-sign"></span>
33.    Add Issue
34.  </button>
35. </form>
36.
37. <script>
38.   $(function() {
39.     $('.editor').wysihtml5({
40.       toolbar: {
41.         "font-styles": false,
42.         "emphasis": true,
43.         "lists": true,
44.         "html": false,
45.         "link": true,
46.         "image": false,
47.         "color": false,
48.         "blockquote": false,
49.         "size": "sm"
50.       }
51.     });
52.   });

```

```

53.  });
54. </script>

```

Issue Summary, Issue Detail နဲ့ Issue Priority တို့ သတ်မှတ်လို့ရတဲ့ Form Template တစ်ခုဖြစ်ပြီး သူလည်းပဲ Issue Edit လိုပဲ Issue Detail Textarea ကို Rich Text Editor အဖြစ် ပြောင်းလဲသတ်မှတ်ထားပါတယ်။

#### 14.8 – Login Form Template – `templates/login.html`

ဆက်လက်တည်ဆောက်မယ့် Template ကတော့ Login Form အတွက်ဖြစ်ပါတယ်။ `teamplates/login.html` မှာ အခုလိုရေးသားရပါမယ်။

##### Template

```

1. <form action="#" id="login-form">
2.   <div class="alert alert-warning" id="login-alert"
3.     style="display: none">
4.       <span class="glyphicon glyphicon-warning-sign"></span>
5.       Login ID or Password incorrect
6.     </div>
7.
8.   <div class="form-group">
9.     <label for="login-id">Login ID</label>
10.    <input type="text" class="form-control" placeholder="Login ID"
11.      id="login-id">
12.    </div>
13.    <div class="form-group" placeholder="Email">
14.      <label for="password">Password</label>
15.      <input type="password" class="form-control" id="password"
16.        placeholder="Password">
17.    </div>
18.
19.    <button type="submit" class="btn btn-primary btn-lg" id="login">
20.      Login
21.      <span class="glyphicon glyphicon-log-in"></span>
22.    </button>
23.  </form>

```

LoginID နဲ့ Password တို့ထည့်သွင်းလို့ရတဲ့ Form Template တစ်ခုဖြစ်ပြီး အပေါ်ဆုံးမှာ Login Error ဖြစ်ရင် ဖော်ပြ ရမယ့် Bootstrap Style Alert Box တစ်ခုကိုပါ ထည့်သွင်းသတ်မှတ်ထားပါတယ်။

## 14.9 – Navigation Menu Template – `templates/nav.html`

Navigation Menu ကိုလည်းသီးခြား Template တစ်ခုနဲ့ ခွဲခြားရေးသားထားပါ၌မယ်။ `templates/nav.html` မှာ အခုလိုရေးသားရပါမယ်။

### Template

```

1. <div class="container-fluid">
2.   <div class="navbar-header">
3.     <button type="button" class="navbar-toggle"
4.       data-toggle="collapse" data-target=".navbar-collapse">
5.       <span class="icon-bar"></span>
6.       <span class="icon-bar"></span>
7.       <span class="icon-bar"></span>
8.     </button>
9.     <a class="navbar-brand" href="#">Issue Tracking System</a>
10.   </div>
11.   <div class="navbar-collapse collapse">
12.     <% if(auth) { %>
13.       <ul class="nav navbar-nav">
14.         <li class="active">
15.           <a href="#/issues" id="issue-list">
16.             All Issues <span class="badge"><%= issueCount %></span>
17.           </a>
18.         </li>
19.         <li><a href="#" id="my-issues">My Issues <span class="badge">
20.           <%= myIssues %></span></a></li>
21.         <li>
22.           <a href="#" id="new-issue">
23.             <span class="text-primary glyphicon glyphicon-plus"></span>
24.             New Issue
25.           </a>
26.         </li>
27.       </ul>
28.       <ul class="nav navbar-nav navbar-right">
29.         <li><a href="#/users" id="user-list">Users</a></li>
30.         <li><a href="#" id="profile"><%= userName %></a></li>
31.         <li><a href="#" id="logout">Logout</a></li>
32.       </ul>
33.     <% } %>
34.   </div>
35. </div>

```

Navbar ရဲ ရှေ့နားမှာ Issue Tracking System ဆိုတဲ့အမည်ကို Brand Name အဖြစ်ထည့်သွင်းထားပြီး၊ Navbar Menu အနေနဲ့ နှစ်ခုပါဝင်ပါတယ်။ ဘယ်ဘက်ခြမ်းက Menu မှာ All Issues, My Issues နဲ့ New Issue တို့ကို သွားလို့ရတဲ့ Button တွေဘာဝင်ပြီး ညာဘက်ခြမ်းက Menu မှာတော့ User List, User Profile နဲ့ Logout Button တို့ပါဝင်ပါတယ်။ Menu Item တွေကို Authenticated User မှသာ ဖော်ပြနိုင်လည်း ထည့်သွင်းစိစစ်ထားပါတယ်။

Navbar ရဲ ရှေ့ဆုံးမှာ Button တစ်ခုကိုထည့်သွင်းထားတဲ့ရည်ရွယ်ချက်ကတော့ Responsive လုပ်ဆောင်ချက်ရရှိနိုင်ဖို့ အတွက်ဖြစ်ပါတယ်။ အခြေခံအလုပ်လုပ်ပုံကတော့ ပုံမှန်ဆုံးရင် Navbar Menu တွေကိုဖော်ပြပြီး Screen သေးလို့ Menu ကို မဖော်ပြနိုင်တဲ့အခါ Menu ကို ဖျောက်လိုက်ပြီး Navbar ရဲ ရှေ့ဆုံးက Toggle Button

ပေါ်လာမှုဖြစ်ပါတယ်။ Button ကို နိုင်လိုက်မှ Menu ကိုဖော်ပြုမှုဖြစ်တဲ့အတွက် Screen Size သေးတဲ့ Device တွေမှာလည်း Menu ရဲ့ အလုပ်လုပ်ပုံက အလိုက်သင့် အဆင်ပြေနေမှာ ဖြစ်ပါတယ်။

Issue Tracking System All Issues 5 My Issues 3 + New Issue Users Profile Logout

፳፭፲፭ - Navbar Template

## 14.10 - User Edit Template - templates/user-edit.html

ဆက်လက်ပြီး User တစ်ဦးချင်းစီရဲ အချက်အလက်တွေနဲ့ Password တို့ကို ပြပြင်လိုပါရ။ User Edit Template တစ်ခုကို တည်ဆောက်ပါမယ်။ [templates/user-edit.html](#) မှာ အခုံလိုရေးသားရပါမယ်။

## Template

```
1. <h2>Edit User</h2>
2.
3. <form action="#" id="edit-user-form">
4.   <div class="form-group">
5.     <label for="">Full Name *</label>
6.     <input type="text" class="form-control"
7.       placeholder="Full Name" id="full-name" value="<%= fullName %>">
8.   </div>
9.   <div class="form-group">
10.    <label for="">Login ID *</label>
11.    <input type="text" class="form-control"
12.      placeholder="Login ID" id="login-id" value="<%= loginId %>">
13.      <span class="help-block">No space, small letters</span>
14.   </div>
15.   <div class="form-group" placeholder="Email">
16.     <label for="">Email *</label>
17.     <input type="email" class="form-control" id="email"
18.       placeholder="Email" value="<%= email %>">
19.   </div>
20.   <div class="form-group">
21.     <label for="">Role *</label>
22.     <select class="form-control" id="role">
23.       <% _.each(app.userRole, function(name, value) { %>
24.         <option value="<%= value %>">
25.           <%= value == role ? 'selected' : '' %><%= name %></option>
26.         <% }) %>
27.       </select>
28.   </div>
29.   <div class="form-group">
30.     <label for="">Password (twice)</label>
31.     <div class="row">
32.       <div class="col-md-6">
33.         <input type="password" class="form-control"
34.           placeholder="Password" id="password">
35.         <span class="help-block">Leave blank if you don't
36.           want to change password</span>
37.       </div>
38.       <div class="col-md-6">
39.         <input type="password" class="form-control"
40.           placeholder="Password Again" id="password-again">
```

```

41.      </div>
42.      </div>
43.      </div>
44.
45.      <button type="submit" class="btn btn-primary btn-lg" id="update-user">
46.          <span class="glyphicon glyphicon-check"></span>
47.          Update User
48.      </button>
49.  </form>

```

User ငဲ့ Full Name, Login ID, Email, Role နဲ့ Password တို့ကို ပြုပြင်လိုက်တဲ့ Form Template တစ်ခုဖြစ်ပါတယ်။

#### 14.11 – User List Templates – `templates/user-list.html, user.html`

User စာရင်းဖော်ပြတဲ့ List Template ကိုတော့ Issue List လိုပဲ နှစ်ရိုင်းခွဲတည်ဆောက်ပါတယ်။ User List Table Template နဲ့ User Row Template တို့ဖြစ်ပါတယ်။ `templates/user-list.html` အခုလိုရေးသားရမှာပါ။

##### Template

```

1. <div class="table-responsive">
2.   <table id="issue-list" class="table table-bordered"
3.     cellspacing="0" width="100%">
4.     <thead>
5.       <tr>
6.         <th>#</th>
7.         <th>Full Name</th>
8.         <th>Login ID</th>
9.         <th>Email</th>
10.        <th>Role</th>
11.        <th width="160">Manage</th>
12.      </tr>
13.    </thead>
14.
15.    <tbody>
16.
17.      </tbody>
18.    </table>
19.  </div>
20.
21. <button class="btn btn-primary" id="new-user">
22.   <span class="glyphicon glyphicon-plus-sign"></span>
23.   New User
24. </button>

```

ထူးခြားတဲ့လုပ်ဆောင်ချက်မပါဝင်ပဲ Table အလွတ်တစ်ခုထည့်သွင်းထားခြင်းဖြစ်ပါတယ်။ `templates/user.html` မှာ အခုလိုဆက်လက်ရေးသားပေးရပါမယ်။

## Template

```

1. <td><%= index %></td>
2. <td><a href="#" class="username"><%= fullName %></a></td>
3. <td><%= loginId %></td>
4. <td><%= email %></td>
5. <td><%= roleLabel %></td>
6. <td>
7.   <a href="#" class="btn btn-default btn-xs edit-user">
8.     <span class="glyphicon glyphicon-edit"></span>
9.     Edit
10.    </a> -
11.    <a href="#" class="btn btn-danger btn-xs delete-user">
12.      <span class="glyphicon glyphicon-trash"></span>
13.      Remove
14.    </a>
15.  </td>

```

Table Row Template တစ်ခုဖြစ်ပြီး သက်ဆိုင်ရာနေရာမှာ Full Name, Login ID, Email စတဲ့တန်ဖိုးတွေကို သူနဲ့ရှာနဲ့သူ အစားထိုးဖော်ပြန့် သတ်မှတ်ထားပါတယ်။

#### 14.12 – User New Template – `templates/user-new.html`

User အသစ်တွေထည့်သွင်းရာမှာ အသုံးပြုနိုင်ဖို့အတွက် New User Form Template တည်ဆောက်ပေးရပါမြို့မယ်။ `templates/user-new.html` မှာ အခုံလုပ်ရေးသားပေးပါမယ်။

## Template

```

1. <h2>New User</h2>
2.
3. <form action="#" id="new-user-form">
4.   <div class="form-group">
5.     <label for="">Full Name *</label>
6.     <input type="text" class="form-control" 
7.       placeholder="Full Name" id="full-name">
8.   </div>
9.   <div class="form-group">
10.    <label for="">Login ID *</label>
11.    <input type="text" class="form-control" 
12.      placeholder="Login ID" id="login-id">
13.    <span class="help-block">No space, small letters</span>
14.  </div>
15.  <div class="form-group" placeholder="Email">
16.    <label for="">Email *</label>
17.    <input type="email" class="form-control" id="email" 
18.      placeholder="Email">
19.  </div>
20.  <div class="form-group">
21.    <label for="">Role *</label>
22.    <select class="form-control" id="role">
23.      <% _.each(app.userRole, function(name, value) { %>
24.        <option value="<%= value %>"><%= name %></option>
25.      <% }) %>
26.    </select>

```

```

27.   </div>
28.   <div class="form-group">
29.     <label for="">Password (twice) *</label>
30.     <div class="row">
31.       <div class="col-md-6">
32.         <input type="password" class="form-control"
33.             placeholder="Password" id="password">
34.       </div>
35.       <div class="col-md-6">
36.         <input type="password" class="form-control"
37.             placeholder="Password Again" id="password-again">
38.       </div>
39.     </div>
40.   </div>
41.
42.   <button type="submit" class="btn btn-primary btn-lg" id="add-user">
43.     <span class="glyphicon glyphicon-plus-sign"></span>
44.     Add User
45.   </button>
46. </form>

```

User Edit မှာလိုပဲ Full Name, Login ID, Email, Role နဲ့ Password တို့ကို ထည့်သွင်းနိုင်တဲ့ Form Template တစ်ခုဖြစ်ပါတယ်။

#### 14.13 – User Profile Template – `templates/user-profile.html`

နောက်ဆုံး Template တစ်ခုအနေနဲ့ User တစ်ယောက်ရဲ့ အချက်အလက်တွေကိုဖော်ပြပေးတဲ့ Profile Template တစ်ခုတည်ဆောက်ဖို့အတွက် `templates/user-profile.html` မှာ အခုလို ရေးသားပေးပါမယ်။

#### Template

```

1.   <div class="row">
2.     <div class="col-md-8 col-sm-12 col-xs-12">
3.       <div class="panel panel-info issue-detail">
4.         <div class="panel-heading">
5.           <h2><span class="glyphicon glyphicon-user"></span>
6.           <%= fullName %></h2>
7.         </div>
8.         <div class="panel-body">
9.           <div class="form-horizontal" role="form">
10.             <div class="form-group">
11.               <label class="col-sm-2 control-label">Login ID</label>
12.               <div class="col-sm-10">
13.                 <p class="form-control-static"><%= loginId %></p>
14.               </div>
15.             </div>
16.             <div class="form-group">
17.               <label class="col-sm-2 control-label">Email</label>
18.               <div class="col-sm-10">
19.                 <p class="form-control-static"><%= email %></p>
20.               </div>
21.             </div>

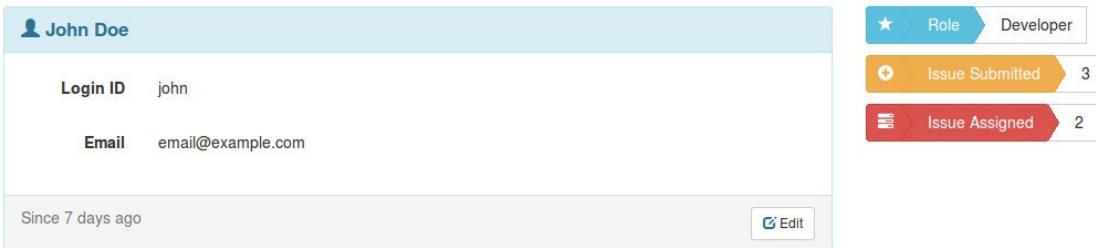
```

```

22.          </div>
23.      </div>
24.      <div class="panel-footer clearfix">
25.          <div class="btn-group pull-right">
26.              <button class="btn btn-default btn-sm" id="edit-user">
27.                  <span class="glyphicon glyphicon-edit text-info"></span>
28.                  Edit
29.              </button>
30.          </div>
31.      </div>
32.  </div>
33. </div>
34.
35. <div class="col-md-4 col-sm-12 col-xs-12 sidebar">
36.     <div class="btn-group btn-breadcrumb">
37.         <a href="#" class="btn btn-info">
38.             <span class="glyphicon glyphicon-star"></span></a>
39.         <a href="#" class="btn btn-info">Role</a>
40.
41.         <a href="#" class="btn btn-default"><%= roleLabel %></a>
42.         <span></span>
43.     </div>
44.
45.     <div class="btn-group btn-breadcrumb">
46.         <a href="#" class="btn btn-warning">
47.             <span class="glyphicon glyphicon-plus-sign"></span></a>
48.         <a href="#" class="btn btn-warning">Issue Submitted</a>
49.
50.         <a href="#" class="btn btn-default">
51.             <%= submitCount %>
52.         </a>
53.         <span></span>
54.     </div>
55.
56.     <div class="btn-group btn-breadcrumb">
57.         <a href="#" class="btn btn-danger">
58.             <span class="glyphicon glyphicon-tasks"></span></a>
59.         <a href="#" class="btn btn-danger">Issue Assigned</a>
60.
61.         <a href="#" class="btn btn-default">
62.             <%= assignCount %>
63.         </a>
64.         <span></span>
65.     </div>
66. </div>
67. </div>

```

သူလည်းက Two Column Layout တစ်ခြဖြစ်ပြီး သက်ဆိုင်ရာ View နဲ့ချိတ်ဆက်ပြီးတဲ့အခါ ရရှိလာမယ့်ရလဒ်ကို ပုံ (၁၄.၁၁) မှာလေ့လာနိုင်ပါတယ်။



John Doe

Login ID: john

Email: email@example.com

Since 7 days ago

Edit

Role: Developer

Issue Submitted: 3

Issue Assigned: 2

### ပုံ (၁၄.၁၁) - User Profile Template

ဘယ်ဘက်ခြမ်းမှာ User ရဲ့ Personal Information ကိုဖော်ပြထားပြီး ညာဘက်ခြမ်းမှာတော့ User Role နဲ့အတူ အဲဒီ User ထည့်သွင်းထားခဲ့တဲ့ Issue စာရင်းနဲ့ အဲဒီ User ကို တာဝန်ပေးအပ်ထားတဲ့ Issue စာရင်းတို့ကို တွေ့ရ မှာပဲဖြစ်ပါ တယ်။

လိုအပ်တဲ့ Template တွေ ပြည့်စုံပြီဖြစ်တဲ့အတွက် BackboneJS နဲ့ လိုအပ်တဲ့ Code တွေကို ဆက်လက်ရေးသား နိုင်ပြီ ဖြစ်ပါတယ်။ နောက်တစ်ခန်းမှာ ဆက်လက်ဖော်ပြပါမယ်။

လက်တမ်းချရေးလိုက်ရင် (၂) နာရီလောက်နဲ့ ပြီးနိုင်တဲ့  
အလုပ်ကို၊ (၄) နာရီလောက် အချိန်ယူပြီး Code Structure  
စနစ်ကျအောင် ပြင်ဆင်တယ်ဆိုတာ၊ ပြုပြင်ထိမ်းသိမ်းရလွယ်တဲ့  
အရည်အသွေးကောင်း Code ရရှိဖို့ဖြစ်ပါတယ်။

## **Rockstar Developer Course**

Project Management, Web Service, Server Architecture

NodeJS အစုံးသည် ဤစာအုပ်ပါ အကြောင်းအရာများတို့

စာရေးသူကိုယ်တိုင် သင့်ကြားပေးခြင်းဖြစ်သည်။

ဆက်သွယ်ရန် - (၀၉) ၇၃၀ ၆၅၅ ၆၂

<http://eimaung.com/courses>

## အခန်း(၁၅) – Building Clients (Web App)

Issue Tracking System Client App တည်ဆောက်ဖို့အတွက် လိုအပ်တဲ့ Directory Structure နဲ့ Template တွေကို ပြီးခဲ့တဲ့အခန်းမှာ တည်ဆောက်ခဲ့ပြီးဖြစ်လို့ လိုအပ်တဲ့ Backbone Model-View-Collection တွေနဲ့ JavaScript Code တွေကို ဆက်လက်ရေးသားပါမယ်။

ဒီအခန်းမှာ ဖော်ပြုမယ့် Code တွေကို ပြီးခဲ့တဲ့အခန်းမှာ ဖော်ပြုပြီးဖြစ်တဲ့ Code တွေနဲ့ တွဲဖက်အသုံး ပြုရမှာဖြစ်ပါတယ်။ အားလုံးပါဝင်တဲ့ Code နမူနာ အပြည့်အစုံကို အောက်ပါလိုပါမယ်။

<https://github.com/eimg/issues-backbone/archive/master.zip>

ပထမဌားဆုံးအနေနဲ့ App တစ်ခုလုံး နဲ့သက်ဆိုင်တဲ့ Login, Logout အပါအဝင် အခြား Helper Function တွေကို app.js ထဲမှာ အခုလို ရေးသားပါမယ်။

### JavaScript

```

1. var app = {
2.   host: "http://localhost:3000/api",
3.
4.   api: function(path) {
5.     return app.host + path;
6.   },
7.
8.   templateCache: {},
9.
10.  loadTemplates: function(names, callback) {
11.    var that = this;
12.    var load = function(index) {
13.      var name = names[index];
14.      console.log('Loading template: ' + name);
15.      $.get('templates/' + name + '.html?cache=false', function(data) {
16.        that.templateCache[name] = data;
17.        index++;
18.        if (index < names.length) {
19.          load(index);
20.        } else {
21.          callback();
22.        }
23.      });
24.    }
25.  }
26.  load(0);

```

```
27. },
28.
29. hookTemplate: function(name, data) {
30.     var tmpl = _.template(this.templateCache[name]);
31.     return tmpl(data);
32. },
33.
34. login: function(user, pass, ok, notok) {
35.     $.ajax({
36.         type: "post",
37.         url: this.api("/users/login"),
38.         data: {
39.             username: user,
40.             password: pass
41.         },
42.         statusCode: {
43.             400: notok,
44.             401: notok,
45.             200: ok
46.         }
47.     });
48. },
49.
50. logout: function(callback) {
51.     $.ajax({
52.         type: "delete",
53.         url: this.api("/users/logout"),
54.         complete: callback
55.     });
56.
57.     localStorage.removeItem("user");
58. },
59.
60. verify: function(ok, notok) {
61.     $.ajax({
62.         url: this.api("/users/verify"),
63.         statusCode: {
64.             401: notok,
65.             200: ok
66.         }
67.     });
68. },
69.
70. saveUserInfo: function(user) {
71.     localStorage.setItem("user", JSON.stringify(user));
72. },
73.
74. getUserInfo: function() {
75.     return JSON.parse(localStorage.getItem("user"));
76. },
77.
78. issueType: [],
79. issuePriority: [],
80. issueStatus: [],
81. userRole: [],
82.
83. loadIssueTypes: function() {
84.     var that = this;
```

```
85.     $.ajax({
86.       url: this.api("/issues/types"),
87.       success: function(data) {
88.         that.issueType = data;
89.       }
90.     });
91.   },
92.
93.   loadIssuePriorities: function() {
94.     var that = this;
95.     $.ajax({
96.       url: this.api("/issues/priorities"),
97.       success: function(data) {
98.         that.issuePriority = data;
99.       }
100.     });
101.   },
102.
103.   loadIssueStatuses: function() {
104.     var that = this;
105.     $.ajax({
106.       url: this.api("/issues/statuses"),
107.       success: function(data) {
108.         that.issueStatus = data;
109.       }
110.     });
111.   },
112.
113.   loadUserRoles: function() {
114.     var that = this;
115.     $.ajax({
116.       url: this.api("/users/roles"),
117.       success: function(data) {
118.         that.userRole = data;
119.       }
120.     });
121.   },
122.
123.   init: function() {
124.     this.loadTemplates(['nav', 'login', 'issue-list',
125.       'issue', 'issue-new', 'issue-detail', 'issue-edit',
126.       'user-list', 'user', 'user-profile', 'user-edit', 'user-new'
127.     ], function() {
128.       new appView();
129.     });
130.   };
131. }
132. }
133.
134. $.ajaxPrefilter( function( options, originalOptions, jqXHR ) {
135.   options.crossDomain = {
136.     crossDomain: true
137.   };
138.
139.   options.xhrFields = {
140.     withCredentials: true
141.   };
142. });
```

လိုင်းနံပါတ် (၂) မှာ API Server Location ကို host Property နဲ့ ကြော်လားပါတယ်။ လိုင်းနံပါတ် (၄) မှာတော့ ပေးလိုက်တဲ့ URI ကို API Server Location နဲ့ တွဲဖက်ပြီးပြန်ပေးတဲ့ api() Function ကိုကြော်လားပါတယ်။ လိုင်းနံပါတ် (၈) က templateCache ဆိတ် Property အလွတ်တစ်ခုကြော်လားပါတယ်။ လိုင်းနံပါတ် (၁၀) က loadTemplates() Function ကတော့ ပြီးခဲ့တဲ့အခန်းမှာ ရေးသားခဲ့တဲ့ Template တွေကိုရယူပြီး template Cache Property ထဲမှာ စုစည်းထားပေးမှာဖြစ်ပါတယ်။ လိုင်းနံပါတ် (၂၉) က hookTemplate() Function ကတော့ ပေးလိုက်တဲ့ Data နဲ့ Template ကို UnderscoreJS Template Function သုံးပြီး Compile လုပ်ပေးမှာ ဖြစ်ပါတယ်။

ဆက်လက်ရေးသားထားတဲ့ login(), logout(), verify() Function တွေကတော့ API Server ထဲ Login, Logout နဲ့ Verify Request တွေ ပေးဖို့အတွက်ဖြစ်ပါတယ်။ Login အောင်မြင်တဲ့အခါ API Server က User Data တွေကို ပြန်လည်ပေးပို့မှာဖြစ်ပါတယ်။ လက်ခံရရှိတဲ့ User Data တွေကို HTML5 LocalStorage နဲ့ သိမ်းဆည်းခြင်း၊ ပြန်လည်ရယူခြင်းလုပ်ငန်းများအတွက် saveUserInfo() နဲ့ getUserInfo() တို့ကို ဆက်လက်ရေးသား ထားပါတယ်။

ဆက်လက်ပြီး လိုင်းနံပါတ် (၈) ကနေစတင်ပြီး loadIssueTypes(), loadIssuePriorities(), loadIssueStatuses() နဲ့ loadUserRoles() Function တို့ကို အသီးသီးသတ်မှတ်ထားပါတယ်။ ဒီ Function တွေက API Server ထဲ Request တွေပေးပို့ပြီး ပြန်လည်လက်ခံရရှိတဲ့ Issue Types, Priorities, Statuses နဲ့ User Roles တို့ကို သက်ဆိုင်ရာ Property ထဲမှာ ထည့်သွင်းသိမ်းဆည်းထားပေးထားမှာ ဖြစ်ပါတယ်။

app Object ရဲ့ နောက်ဆုံး အနေနဲ့ လိုင်းနံပါတ် (၁၂၄) မှာသတ်မှတ်ထားတဲ့ init() Function ကတော့ load Templates() Function အကူအညီနဲ့ Template ဖိုင်တွေအားလုံးကို ရယူပေးပါတယ်။ Template ရယူခြင်း လုပ်ငန်းပြီးစီးတဲ့အခါ appView() Object တစ်ခု တည်ဆောက်ပေးခြင်းအားဖြင့် App ကို စတင်ပေးမှာဖြစ်ပါတယ်။ တနည်းအားဖြင့် App ကို စတင်အလုပ်လုပ်စေဖို့ အတွက် app.init() ကို ခေါ်ယူပေးရမှာဖြစ်ပါတယ်။

ပုံမှန်အားဖြင့် Client App ကနေ Back-end API Service ကို Cross-Domain Request တွေပြုလုပ်တဲ့အခါ CORS Header တွေကို Request နဲ့အတူ မှန်ကန်အောင်ထည့်သွင်းပေးပို့ဖို့ပါတယ်။ ဒါပေမယ့်လက်တွေမှာ ဒီ Header တွေကို တစ်ခုချင်း ကိုယ်တိုင်လိုက်ထည့်နေစရာမလိုပါဘူး။ jQuery ရဲ့ Ajax Options မှာ crossDomain: true နဲ့ withCredentials: true လို့သတ်မှတ်ပေးလိုက်ရင် jQuery က လိုအပ်တဲ့ CORS Header တွေကို အလို အလျောက် ထည့်သွင်းပေးသွားမှာဖြစ်ပါတယ်။ ဒီ Options တွေကို Ajax Request ပြုလုပ်တိုင်း ထည့်ပေးနေစရာ မလိုအောင် \$.ajaxPrefilter() နဲ့ ကြိုတင်သတ်မှတ်ထားနိုင်ပါတယ်။ \$.ajaxPrefilter() မှာ သတ်မှတ် ထားတဲ့ Options တွေကို jQuery က Ajax Request တိုင်းမှာ အလို အလျောက် ထည့်သွင်းပေးသွားမှာဖြစ်ပါတယ်။ ဒါကြောင့် လိုင်းနံပါတ် (၁၃၄) မှာ \$.ajaxPrefilter() Function ကို သတ်မှတ်ပေးထားခြင်းဖြစ်ပါတယ်။

### 15.1 – Issue Model and Collection – `issues.js`

ဆက်လက်ပြီး `issues.js` ထဲမှာ Issue Model နဲ့ Collection တို့ကို အခုလိုရေးသားပေးပါမယ်။

#### JavaScript

```

1. app.issueModel = Backbone.Model.extend({
2.   urlRoot: app.api("/issues"),
3.   defaults: {
4.     "assignedTo": null,
5.     "detail": null,
6.     "priority": 0,
7.     "priorityLabel": "Low",
8.     "status": 0,
9.     "statusLabel": "New",
10.    "summary": null,
11.    "type": 0,
12.    "typeLabel": "Bug"
13.  },
14.
15.  idAttribute: "_id"
16. });
17.
18. app.issueCollection = Backbone.Collection.extend({
19.   model: app.issueModel,
20.   url: app.api("/issues")
21. });
22.
23. app.issueList = new app.issueCollection();

```

`issueModel` ကို `app Object` ရဲ့ `Property` အနေနဲ့ ကြော်လေးတာကို သတိပြုပါ။ အလားတူပဲ လိုင်းနံပါတ် (၁၈) မှာ ကြော်လေးတဲ့ `issueCollection` နဲ့ လိုင်းနံပါတ် (၂၃) မှာ ကြော်လေးတဲ့ `Collection Object` တို့ကိုလည်း `app Object` ရဲ့ `Property` အနေနဲ့ပဲ ကြော်လေးပါတယ်။ ဒီနည်းနဲ့ Namespace တစ်ခုကဲ့သွားတဲ့ အတွက် အခြား တစ်နေရာမှာ `issueModel`, `issueList` စတဲ့ Variable တွေလိုအပ်လို ထပ်မံကြော် အသုံးပြုလိုတဲ့အခါ နာမည်တိုက်နေမှာကို စိတ်ပူစရာမလိုပဲ လွှပ်လွှပ်လပ်လပ် ကြော်အသုံးပြုနိုင်မှာဖြစ်ပါတယ်။

`issueModel` ရဲ့ URL Root ကို `app.api()` Function ရဲ့ အကူအညီနဲ့ `http://localhost:3000/api/issues` လို သတ်မှတ်ပေးထားပါတယ်။ ဒါကြောင့် ဒီ Model ပေါ်မှာ `save()`, `destroy()` စတဲ့လုပ်ငန်း တွေအတွက် BackboneJS က Data Sync Request တွေပေးပိုတဲ့အခါ `http://localhost:3000/api/issues` ကို ပေးပိုသွားမှာ ဖြစ်ပါတယ်။

Model ID အနေနဲ့ `Property` တွေထဲက `_id` ကို အသုံးပြုဖို့ သတ်မှတ်ပေးထားတာကိုလည်းသတိပြုပါ။ သီးခြား ID အသုံးပြုစရာမလိုပဲ Issue တစ်ခုချင်းစီကို MongoDB Database ထဲမှာ သိမ်းလိုက်ရင်ရရှိလာမယ့် `_id` ကို Model ID အဖြစ် အသုံးချလိုက်ခြင်းဖြစ်ပါတယ်။

ဆက်လက်ပြီး လိုင်းနံပါတ် (၁၈) မှာ ကြော်လေးတဲ့ `issueCollection` အတွက် URL ကိုလည်း

app.api() Function အကူအညီနဲ့သတ်မှတ်ထားပါတယ်။ ဒါကြောင့် ဒီ Collection ပေါ်မှာပြုလုပ်တဲ့ fetch(), remove() စဲတဲ့လုပ်ငန်းတွေ အတွက် Data Sync Request တွေပေးပို့တဲ့အခါမှာလည်း <http://localhost:3000/api/issues> ကိုပဲ ပေးပို့သွားမှာ ဖြစ်ပါတယ်။

## 15.2 – User Model and Collection – `users.js`

ဆက်လက်ပြီး `users.js` ထဲမှာ User Model နဲ့ Collection တို့ကို အခုလိုရေးသားပေးပါမယ်။

### JavaScript

```

1. app.userModel = Backbone.Model.extend({
2.   urlRoot: app.api("/users"),
3.   idAttribute: "_id"
4. });
5.
6. app.userCollection = Backbone.Collection.extend({
7.   model: app.userModel,
8.   url: app.api("/users")
9. });
10.
11. app.userList = new app.userCollection();

```

ဒီနေရာမှာလည်း Model နဲ့ Collection တို့အတွက် Back-end Server အဖြစ် <http://localhost:3000/api/users> ကို သတ်မှတ်ပေးထားပါတယ်။

လိုအပ်တဲ့ Model နဲ့ Collection တွေရရှိပြီ့နဲ့ View တွေဆက်လက်တည်ဆောက်ပါမယ်။

## 15.3 – Navigation Menu View – `views/navView.js`

ပထမုဒ္ဒိုးဆုံးအနေနဲ့ Navigation Menu View တစ်ခုတည်ဆောက်ပါမယ်။ Navigation ဟာ ပုံမှန်အားဖြင့် View တစ်ခု ဖြစ်ဖို့မလိုပဲ Static HTML အနေနဲ့ Main Template ထဲမှာ အသေရေးထားလို့ရပါတယ်။ ဒါပေမယ့် ကျွန်တော်တိုက Navigation Menu ထဲမှာ Issue Count လို့ အချက်အလက်မျိုးကို ထည့်သွင်းဖော်ပြလိုတဲ့ အတွက် အသေရေးမထားပဲ အခြေအနေပေါ်မှတည်ပြီး ပြောင်းလဲဖော်ပြနိုင်စေဖို့အတွက် Backbone View အနေနဲ့ တည်ဆောက်ထားမှာဖြစ်ပါတယ်။ `views/navView.js` မှာ အခုလို ရေးသားရပါမယ်။

### JavaScript

```

1. var navView = Backbone.View.extend({
2.   tagName: "div",
3.   className: "navbar navbar-default",
4.
5.   events: {
6.     "click #new-issue": "showNewIssueForm",
7.     "click #issue-list": "showIssueList",
8.     "click #my-issues": "showMyIssues",
9.     "click #user-list": "showUserList",
10.    "click #profile": "showProfile",

```

```
11.     "click #logout": "logout"
12. },
13.
14. render: function() {
15.     var user = app.getuserInfo();
16.     if( user ) {
17.         var myIssues = app.issueList.where({
18.             assignedTo: user._id
19.         });
20.
21.         var data = {
22.             auth: true,
23.             userName: user.fullName,
24.             issueCount: app.issueList.length,
25.             myIssues: myIssues.length
26.         };
27.     } else {
28.         data = {
29.             auth: false
30.         }
31.     }
32.
33.     this.$el.html( app.hookTemplate("nav", data) );
34.     return this;
35. },
36.
37. showNewIssueForm: function(e) {
38.     $(".nav li").removeClass('active');
39.     $(e.currentTarget).parent().addClass('active');
40.
41.     var form = new issueNewView();
42.     $("#main").html( form.render().el );
43. },
44.
45. showIssueList: function(e) {
46.     $(".nav li").removeClass('active');
47.     $(e.currentTarget).parent().addClass('active');
48.
49.     var list = new issueListView();
50.     $("#main").html( list.render().el );
51. },
52.
53. showMyIssues: function(e) {
54.     $(".nav li").removeClass('active');
55.     $(e.currentTarget).parent().addClass('active');
56.
57.     var list = new myIssueListView();
58.     $("#main").html( list.render().el );
59. },
60.
61. showUserList: function(e) {
62.     $(".nav li").removeClass('active');
63.     $(e.currentTarget).parent().addClass('active');
64.
65.     var users = new userListView();
66.     $("#main").html( users.render().el );
67. },
68.
69. showProfile: function(e) {
```

```

70.     $(".nav li").removeClass('active');
71.     $(e.currentTarget).parent().addClass('active');
72.
73.     var user = new app.userModel(app.getUserInfo());
74.     var profile = new userProfileView({ model: user });
75.     $("#main").html( profile.render().el );
76.   },
77.
78.   logout: function() {
79.     var that = this;
80.     app.logout(function() {
81.       var login = new loginView();
82.       $("#main").html( login.render().el );
83.
84.       that.render();
85.     });
86.   }
87. });

```

ဒီ View ရဲ အမိကလုပ်ငန်းတာဝန်ကတော့ နှိပ်လိုက်တဲ့ Menu Item ပေါ်မှုတည်ပြီး သင့်တော်တဲ့ Main Content View ကို ဖော်ပြပေးဖို့ပြစ်ပါတယ်။ All Issue Menu ကို နှိပ်လိုက်ရင် Main Content အနေနဲ့ Issue List View ကို ဖော်ပြပေးပြီး၊ Users Menu ကို နှိပ်လိုက်ရင် User List View ကို ဖော်ပြပေးတဲ့သဘောမျိုးဖြစ်ပါတယ်။

ဒီနေရာမှာ ပိုပြီးသဘာဝကျတဲ့ နည်းလမ်းကတော့ Backbone Router ကိုအသုံးပြုခြင်းဖြစ်ပါတယ်။ Router ရဲ URL Pattern ပေါ်မှ တည်ပြီး သင့်တော်တဲ့ Main Content View ကို ဖော်ပြသင့်ပါတယ်။ ဥပမာ – URL က#/issue-list ဆိုရင် Issue List View ကိုပြပြီး URL က#/user-list ဆိုရင် User List View ကိုပြအောင် Backbone Router နဲ့ရေးသားထားနိုင်ပါတယ်။ ဒါဆိုရင် Menu Item တွေအတွက် သက်ဆိုင်ရာ Link သတ်မှတ်ပေးထားလိုက်ယုံနဲ့ နှိပ်လိုက်တဲ့ Link ပေါ်မှုတည်ပြီး View လည်း ပြောင်းလဲသွားမှုဖြစ်ပါတယ်။ ဒါပေမယ့် ကျွန်ုတ်တို့က Backbone Router ကို ထည့်သွင်း ဖော်ပြမထားတဲ့အတွက် Router မသုံးပဲ Navigation View ရဲ Event အနေနဲ့ပဲ ဘယ် Menu Item ကိုနှိပ်ရင် ဘယ် Main Content View ကို ဖော်ပြရမလဲဆိုတာကို Event အနေနဲ့ သတ်မှတ်ထားခြင်းဖြစ်ပါတယ်။

လိုင်းနံပါတ် (၁၄) က render() ကိုလေ့လာကြည့်ရင်၊ app.getUserInfo() အကူအညီနဲ့ User Information ကို ရယူထားပါတယ်။ ပုံမှန်အားဖြင့် Login ဝင်ထားမှသာ User Information က ရှိမှုပါ။ ဒါကြောင့် User Information ရှိနေရင် Nav View အတွက် Data အနေနဲ့ auth: true တန်ဖိုးကို ထည့်သွင်းပေးထားပါတယ်။ Navigation Menu အတွက် Template ရေးသားစဉ်က auth: true ဖြစ်မှသာ Menu Item တွေ ကို ဖော်ပြို ထည့်သွင်းရေး သားထားပြီးဖြစ်ပါတယ်။ ပြီးခဲ့တဲ့အခန်းမှာ ရေးသားခဲ့တဲ့ tempates/nav.html မှာ ပြန်လည်လေ့လာနိုင်ပါတယ်။

ပြီးတဲ့အခါ Menu ထဲမှာ ထည့်သွင်းဖော်ပြိုလိုတဲ့ User Name, Issue Count, My Issue Count တန်ဖိုးတွေကို လည်း Data အဖြစ် တွဲဖက်ထည့်သွင်းပေးထားပါတယ်။ User Name ကို app.getUserInfo() ကနေရရှိပြီး Issue Count ကိုတော့ Issue Collection ရဲ Length ကို ရယူခြင်းအားဖြင့် ရရှိနိုင်ပါတယ်။ My Issue Count အနေနဲ့ Issue Collection ထဲက Issue တွေရဲ Assigned To Field တန်ဖိုးကို User ID နဲ့ Filter လုပ်ယူခြင်းအားဖြင့် ရရှိ

ပါတယ်။ Filter လုပ်တဲ့ လုပ်ငန်းကို လိုင်းနံပါတ် (၁၇) မှာ ရေးသားထားပါတယ်။

ဒါ View မှာပါဝင်တဲ့ ကျွန်ုင် Function တွေကတော့ သက်ဆိုင်ရာ Main View ကို Render လုပ်ပေးခြင်းဖြစ်ပါတယ်။ အသေးစိတ်ကို ဒီနေရာမှာ ရေးစရာမလိုပါဘူး။ သက်ဆိုင်ရာ View အတွင်းမှာ ရေးသားမှာဖြစ်ပါတယ်။ နဲ့မှုနာအနေနဲ့ လိုင်းနံပါတ် (၄၅) မှာရေးသားထားတဲ့ showIssueList() ကိုလေ့လာကြည့်နိုင်ပါတယ်။ jQuery ကို အသုံးပြုပြီး အမြဲး Menu Item တွေမှာ ရှိနေတဲ့ .active Class ကိုဖြတ်ပြီး လက်ရှိ Menu Item အတွက် .active Class ကို ထည့်သွင်းထားပါတယ်။ ပြီးတဲ့အခါ issueListView() Object တစ်ခု တည်ဆောက်ပြီး၊ အဲဒီ Object ရဲ့ render() Function က ပေးတဲ့ el (Element) ကို #main အတွင်းမှာ app.hook() Function အကူအညီနဲ့ ဖော်ပြခေါ်လိုက်ခြင်းဖြစ်ပါတယ်။ #main ဆိုတာ Main Content View ကို ဖော်ပြန့်အတွက် index.html အတွင်း မှာ ကြော်တင်သတ်မှတ်ထားတဲ့ Element ဖြစ်ပါတယ်။ ဒါ အကြောင်းကို မကြောင်ဆက်လက်ဖော်ပြပါမယ်။

#### 15.4 – Login View – `views/loginView.js`

ဆက်လက်ပြီး Login Form ဖော်ပြပေးနိုင်တဲ့ Login View တစ်ခုတည်ဆောက်ပါမယ်။

##### JavaScript

```

1. var loginView = Backbone.View.extend({
2.   tagName: "div",
3.
4.   events: {
5.     "submit #login-form": "login"
6.   },
7.
8.   initialize: function() {
9.     //
10.  },
11.
12.   render: function() {
13.     this.$el.html( app.hookTemplate("login") );
14.     return this;
15.  },
16.
17.   login: function() {
18.     var loginId = $("#login-id").val();
19.     var password = $("#password").val();
20.
21.     app.login(loginId, password, function(user) {
22.
23.       app.saveUserInfo(user);
24.
25.       app.issueList.fetch({
26.         success: function() {
27.           var list = new issueListView();
28.           $("#main").html( list.render().el );
29.
30.           var nav = new navView();
31.           $("#nav").html( nav.render().el );
32.
33.           app.userList.fetch();

```

```

34.         app.loadIssueTypes();
35.         app.loadIssuePriorities();
36.         app.loadIssueStatuses();
37.         app.loadUserRoles();
38.     }
39. });
40.
41.     }, function() {
42.         $("#login-alert").show();
43.     });
44.
45.     return false;
46. }
47. );

```

Login View ရဲ့ တာဝန်က Login Form ကို ဖော်ပြပုံသာမကပဲ၊ Login လုပ်ငန်းစဉ်ကိုပါ ဆောင်ရွက်ပေးဖို့ဖြစ်ပါ တယ်။ လိုင်းနံပါတ် (၁၂) က render() Function က app.hook() Function အကူအညီနဲ့ ကြိုတင် ရေးသားထားတဲ့ Login Template ကို အသုံးပြုပြီး Login Form ကို ဖော်ပြပေးပါတယ်။ လိုင်းနံပါတ် (၅) မှာ Event အနေနဲ့ Login Button ကို နိုပ်ရင် login() Function ကို အလုပ်လုပ်ပေးဖို့ သတ်မှတ်ထားပါတယ်။

login() Function က Form မှာ ရိုက်ထည့်လိုက်တဲ့ Login ID နဲ့ Password ကို ရယူပြီး ကြိုတင်ရေးသားထားတဲ့ app.login() ကို အသုံးပြုပြီး Login ဝင်ပေးပါတယ်။ Login Success ဖြစ်တဲ့အခါ app.saveUserInfo() နဲ့ User Information ကို သိမ်းပေးပါတယ်။ ပြီးတဲ့အခါ app.issueList Collection ပေါ်မှာ fetch() ကို Run ခြင်းအားဖြင့် API Server က Issue List ကို ရယူပေးပါတယ်။ issueList Collection မှာ URL ကို localhost:3000/api/issues/ လို့ သတ်မှတ်ထားတဲ့အတွက် fetch() Function ကို Run တဲ့အခါ Backbone က localhost:3000/api/issues/ ကို GET Request တစ်ခုပေးပို့သွားမှာဖြစ်ပါတယ်။ ကျွန်ုတ်တို့က API Server မှာ api/issues URI ကို GET Request လက်ခံရရှိတဲ့အခါ Issue List ကို ပြန်ပေးဖို့ ကြိုတင်ရေးသားထားပြီးဖြစ်တဲ့အတွက် API Server က Issue List ကို ပြန်ပေး ပါလိမ့်မယ်။ Backbone က လက်ခံရရှိတဲ့ Issue List ကို Collection ထဲမှာ ထည့်သွင်းပေးသွားမှာဖြစ်ပါတယ်။

ဆုံးလိုတာက Server ကနေ Issue List ကို ရရှိဖို့ Request ပြုလုပ်တဲ့ကိစ္စတွေ၊ ရရှိလာတဲ့ ရလဒ်ကို Format မှန် မ မှန် စီစစ်ပြီး Collection ထဲထည့်သွင်းတဲ့ကိစ္စတွေကို ကိုယ်တိုင်လုပ်စရာမလိုပဲ Collection ပေါ်မှာ fetch() Function ကို Run ပေးလိုက်ယူနဲ့ Server က ပြန်ပေးတဲ့ List က Collection ထဲမှာ အလိုအသွောက်ရောက်ရှိသွားအောင် Backbone က နောက်ကွယ်ကနေ ဆောင်ရွက်ပေးသွားခြင်းပဲဖြစ်ပါတယ်။

Issue List ကို ရရှိပြီးတဲ့အခါ Nav View ကို #nav Element အတွင်းမှာ Render လုပ်ပေးပြီး Issue List View ကို #main Element အတွင်းမှာ Render လုပ်ပေးထားပါတယ်။ ဆက်လက်ပြီး Issue Type, Priority, Status နဲ့ User List, User Role တွေကို Load လုပ်ပြီး သက်ဆိုင်ရာ Property အတွင်းမှာ သိမ်းဆည်းပေးပါသေးတယ်။

ဒီနည်းအတိုင်း Login Success ဖြစ်သွားတာနဲ့ လိုအပ်တဲ့ Data အားလုံးကို တန်းစီရပုံပေးသွားမှာဖြစ်လို့ နောက်အဆင့်တွေမှာ ဒီ Data တွေကို အသင့်အသုံးပြုနိုင်မှာပဲဖြစ်ပါတယ်။ Login Fail ဖြစ်ရင်တော့ Template ထဲမှာ ကြိုတင်ရေး သားထားတဲ့ #login-alert ကို ဖော်ပြပေးမှာဖြစ်ပါတယ်။

## 15.5 – Issue List View – `views/issueListView.js`

ഈ ലാറ്റീൻ പ്രോഗ്രാം ലഭ്യമാണ് ഇവിടെ സൂചിപ്പിച്ചിരിക്കുന്നത്.

### JavaScript

```

1. var issueListView = Backbone.View.extend({
2.   tagName: "div",
3.
4.   initialize: function() {
5.     //
6.   },
7.
8.   render: function() {
9.     this.$el.html( app.hookTemplate("issue-list") );
10.
11.    var that = this;
12.    app.issueList.each(function(issue) {
13.      var view = new issueView({ model: issue });
14.      $("tbody", that.$el).append( view.render().el );
15.    });
16.
17.    return this;
18.  }
19. });
20.
21. var myIssueListView = Backbone.View.extend({
22.   tagName: "div",
23.
24.   initialize: function() {
25.     //
26.   },
27.
28.   render: function() {
29.     this.$el.html( app.hookTemplate("issue-list") );
30.
31.    var that = this;
32.    var issues = app.issueList.where({
33.      assignedTo: app.getUserInfo().__id
34.    });
35.
36.    _.each(issues, function(issue) {
37.      var view = new issueView({ model: issue });
38.      $("tbody", that.$el).append( view.render().el );
39.    });
40.
41.    return this;
42.  }
43. });
44.
45. var issueView = Backbone.View.extend({
46.   tagName: "tr",
47.
48.   events: {
49.     "click .issue-summary": "viewDetail"
50.   },
51.

```

```

52.     initialize: function() {
53.         // 
54.     },
55.
56.     render: function() {
57.         var index = app.issueList.indexOf(this.model);
58.         this.model.set("index", index + 1);
59.
60.         this.$el.html( app.hookTemplate("issue", this.model.toJSON() ) );
61.         return this;
62.     },
63.
64.     viewDetail: function(e) {
65.         var detail = new issueDetailView({model: this.model});
66.         $("#main").html( detail.render().el );
67.     }
68. });

```

ဒီဖိုင်ထဲမှာ View (၃) ခုပါဝင်ပါတယ်။ Issue List Table View, My Issue List Table View နဲ့ Table တွေထဲက Row တစ်ခုချင်းအတွက်သတ်မှတ်ထားတဲ့ Issue View တို့ပဲဖြစ်ပါတယ်။

လိုင်းနဲ့ပါတ် (၄၅) မှာ သတ်မှတ်ထားတဲ့ Issue View မှာ app.hookTemplate() အကူအညီနဲ့ ကြိုတင်ရေးသား ထားတဲ့ Issue Template ကို Render လုပ်ထားပါတယ်။ Event အနေနဲ့ Issue Summary ကိုနိုင်ရင် Issue Detail View ကို ဖော်ပြပေးဖို့ သတ်မှတ်ထားပါတယ်။

Issue List View နဲ့ My Issue List View တို့ကတော့ အလုပ်လုပ်ပုံအတူတူပါပဲ။ Issue List Collection ကို Loop လုပ်ပြီး Table အတွင်းမှာ Issue View တွေကို Row အနေနဲ့ တန်းစီထည့်သွင်းပေးလိုက်ခြင်းပဲဖြစ်ပါတယ်။ Issue List View အတွက် Issue List Collection တစ်ခုလုံးကိုသုံးပြီး၊ My Issue List View အတွက်တော့ Login လုပ်ထားတဲ့ User ရဲ့ ID နဲ့ Issue ရဲ့ Assigned To Field တို့ကိုကိုညီတဲ့ Issue စာရင်းကိုသာ အသုံးပြုထားပါတယ်။

## 15.6 – Issue Detail View – `views/issueDetailView.js`

ဆက်လက်ပြီး Issue Summary ကို နှိပ်လိုက်တဲ့အခါ ဖော်ပြမယ့် Issue Detail အတွက် View တစ်ခုကို တည်ဆောက်ပါမယ်။

### JavaScript

```

1. var issueDetailView = Backbone.View.extend({
2.     tagName: "div",
3.
4.     events: {
5.         "click #delete-issue": "deleteIssue",
6.         "click #close-issue": "closeIssue",
7.         "click #edit-issue": "editIssue",
8.         "click .change-type": "changeType",
9.         "click .change-priority": "changePriority",
10.        "click .change-assign": "changeAssign",

```

```
11.     "click .change-status": "changeStatus",
12.     "click #post-comment": "postComment",
13.     "click .delete-comment": "deleteComment"
14.   },
15.
16.   initialize: function() {
17.     //
18.   },
19.
20.   render: function() {
21.     this.model.set("users", app.userList.toJSON());
22.     this.$el.html(app.hookTemplate("issue-detail",this.model.toJSON()));
23.     return this;
24.   },
25.
26.   deleteIssue: function() {
27.     var urlRoot = this.model.urlRoot;
28.     var id = this.model.id;
29.     this.model.url = function() {
30.       return urlRoot + "/" + id;
31.     };
32.
33.     this.model.destroy({
34.       wait: true,
35.       success: function() {
36.         var list = new issueListView();
37.         $("#main").html( list.render().el );
38.       }
39.     });
40.   },
41.
42.   closeIssue: function() {
43.     var urlRoot = this.model.urlRoot;
44.     var id = this.model.id;
45.     this.model.url = function() {
46.       return urlRoot + "/status/" + id;
47.     };
48.
49.     this.model.save({
50.       status: 4,
51.       statusLabel: app.issueStatus[4]
52.     },
53.     {
54.       patch: true,
55.       wait: true,
56.       success: function() {
57.         var list = new issueListView();
58.         $("#main").html( list.render().el );
59.       }
60.     });
61.   },
62.
63.   editIssue: function() {
64.     var edit = new issueEditView({ model: this.model });
65.     $("#main").html( edit.render().el );
66.   },
67.
68.   changeType: function(e) {
69.     var type = $(e.currentTarget).data("value");
70.     var urlRoot = this.model.urlRoot;
```

```
71.     var id = this.model.id;
72.     var that = this;
73.
74.     this.model.url = function() {
75.         return urlRoot + "/type/" + id;
76.     };
77.
78.     this.model.save({
79.         type: type,
80.         typeLabel: app.issueType[type]
81.     }, {
82.         patch: true,
83.         wait: true,
84.         success: function() {
85.             var detail = new issueDetailView({ model: that.model });
86.             $("#main").html( detail.render().el );
87.         }
88.     });
89. },
90.
91. changePriority: function(e) {
92.     var priority = $(e.currentTarget).data("value");
93.     var urlRoot = this.model.urlRoot;
94.     var id = this.model.id;
95.     var that = this;
96.
97.     this.model.url = function() {
98.         return urlRoot + "/priority/" + id;
99.     };
100.
101.    this.model.save({
102.        priority: priority,
103.        priorityLabel: app.issuePriority[priority]
104.    }, {
105.        patch: true,
106.        wait: true,
107.        success: function() {
108.            var detail = new issueDetailView({ model: that.model });
109.            $("#main").html( detail.render().el );
110.        }
111.    });
112. },
113.
114. changeStatus: function(e) {
115.     var status = $(e.currentTarget).data("value");
116.     var urlRoot = this.model.urlRoot;
117.     var id = this.model.id;
118.     var that = this;
119.
120.     this.model.url = function() {
121.         return urlRoot + "/status/" + id;
122.     };
123.
124.
125.     this.model.save({
126.         status: status,
127.         statusLabel: app.issueStatus[status]
128.     }, {
129.         patch: true,
130.         wait: true,
```

```
131.      success: function() {
132.        var detail = new issueDetailView({ model: that.model });
133.        $("#main").html( detail.render().el );
134.      }
135.    });
136.  },
137.
138.  changeAssign: function(e) {
139.    var assign = $(e.currentTarget).data("value");
140.    var label = $(e.currentTarget).html();
141.
142.    var urlRoot = this.model.urlRoot;
143.    var id = this.model.id;
144.    var that = this;
145.
146.    this.model.url = function() {
147.      return urlRoot + "/assign/" + id;
148.    };
149.
150.    this.model.save({
151.      assignedTo: assign,
152.      assignedToLabel: label
153.    },
154.    {
155.      patch: true,
156.      wait: true,
157.      success: function() {
158.        var detail = new issueDetailView({ model: that.model });
159.        $("#main").html( detail.render().el );
160.      }
161.    });
162.  },
163.  postComment: function() {
164.    var comment = $("#comment").val();
165.    if(!comment) return;
166.
167.    var user = app.getUserInfo();
168.    var authorId = user._id;
169.    var authorName = user.fullName;
170.
171.    var newComment = {
172.      comment: comment,
173.      authorId: authorId,
174.      authorName: authorName,
175.      submittedAt: new Date(),
176.      issueId: this.model.id
177.    }
178.
179.    var comments = this.model.get("comments") || [];
180.    comments.push(newComment);
181.
182.    var urlRoot = this.model.urlRoot;
183.    var that = this;
184.    $.post(urlRoot + "/comments", newComment, function() {
185.      var detail = new issueDetailView({ model: that.model });
186.      $("#main").html( detail.render().el );
187.    });
188.  },
189.
```

```

190. deleteComment: function(e) {
191.     var index = $(e.currentTarget).data("index");
192.
193.     var comments = this.model.get("comments") || [];
194.     comments.splice(index, 1);
195.     this.model.set("comments", comments);
196.
197.     var urlRoot = this.model.urlRoot;
198.     var that = this;
199.     $.ajax({
200.         type: "delete",
201.         url: urlRoot + "/comments",
202.         data: {
203.             issueId: that.model.id,
204.             commentId: index
205.         },
206.         complete: function() {
207.             var detail = new issueDetailView({ model: that.model });
208.             $("#main").html( detail.render().el );
209.         }
210.     });
211. }
212. });

```

ဒါ View မှာတော့ ပါဝင်ဝင်တဲ့လုပ်ဆောင်ချက် နည်းနည်းများပါတယ်။ Issue ကို Delete လုပ်နိုင်တဲ့ လုပ်ဆောင်ချက်တွေ၊ Issue ကို Close လုပ်လိုရတဲ့ လုပ်ဆောင်ချက်တွေ၊ Issue Status, Priority, Assigned To တန်ဖိုးတွေကို ပြောင်းလို ရတဲ့လုပ်ဆောင်ချက်တွေနဲ့ Comments လုပ်ဆောင်ချက်တွေ ပါဝင်ပါတယ်။

လိုင်းနံပါတ် (၂၆) က deleteIssue() ကိုလေ့လာကြည့်ပါ။ Issue Model တည်ဆောက်ခဲ့စဉ်က Model URL ကို ထည့်သွင်းသတ်မှတ်ထားခဲ့ပါတယ်။ ဒါပေမယ့် URL Structure က ဆောင်ရွက်လိုတဲ့ လုပ်ဆောင်ချက်ပေါ် မှတည်၍၊ ပြောင်းလဲမှာဖြစ်တဲ့အတွက် Default URL ကို မသုံးပဲ Model URL တစ်ခုကို အရင်တည်ဆောက်ပါတယ်။ ဥပမာ - Issue Delete လုပ်လိုတဲ့အခါ အသုံးပြုရမယ့် URL က localhost:3000/api/issues/:id ဖြစ်ပေမယ့် Issue Status ကို ပြောင်းလိုတဲ့အခါ အသုံးပြုရမယ့် URL က localhost:3000/api/issues/status/:id ဖြစ်ပါတယ်။ Action အားလုံးအတွက် URL Pattern တစ်ခုတည်းကို သုံးလိုမရတော့ပဲ URL Pattern ကို လိုသလို သတ်မှတ်အသုံးပြုပေးဖို့ လိုတဲ့သဘောဖြစ်ပါတယ်။

Delete လုပ်ဆောင်ချက်ဖြစ်တဲ့အတွက် Model ပေါ်မှာ destroy() Function ကို Run ထားပါတယ်။ Destroy Function က Model ကို ပယ်ဖျက်ပေးလိုက်တဲ့အပြင် Server ထံကိုလည်း DELETE Request ပေးပို့သွားမှာဖြစ်ပါတယ်။ ဒါကြောင့် destroy() Function Run လိုက်တာနဲ့ လက်ရှိ Model Client-side မှာပျက်သွားသလို၊ Server-side မှာလည်း ပယ်ဖျက်ပြီးဖြစ်သွားမှာပဲ ဖြစ်ပါတယ်။

သတိပြုသင့်တာကတော့၊ လိုင်းနံပါတ် (၃၄) က wait: true ဖြစ်ပါတယ်။ ဒီသတ်မှတ်ချက်ကြောင့် Backbone က Model ကို ချက်ခြင်းပယ်ဖျက်မပြစ်ပဲ Server-side ကနေ Success Respond လက်ခံရရှိမှာ ပယ်ဖျက်ပေးမှာဖြစ် ပါတယ်။ တနည်းအားဖြင့် Model က Client-side မှာပျက်သွားပေမယ့် Server-side မှာ အမှားအယွင်းတစ်ခုကြောင့် ကျေန်သွားတယ်ဆိုတာမျိုး၊ မဖြစ်နိုင်တော့ပါဘူး။ Server-side မှာ အမှန်တစ်ကယ်

ပျက်သွားမှ Backbone ၏ Client-side မှာ ပယ်ဖျက်ပေးစေလိုတဲ့အတွက် `wait: true` ကို ထည့်သွင်းထားရ ခြင်းပဲဖြစ်ပါတယ်။

နောက်ထပ်နမူနာအနေနဲ့ လိုင်းနံပါတ် (၄၂) က `closeIssue()` ကို လေ့လာကြည့်စေလိုပါတယ်။ `Issue` ကို Close လုပ်ဖို့ အတွက် Server ကို ပေးပို့ရမယ့် API URL က `localhost:3000/api/issues/status/:id` ဖြစ်တဲ့ အတွက် URL Pattern မှာ `status` ပါဝင်သွားအောင် အရင်ပြုပြင်ထားပါတယ်။ ပြီးတဲ့အခါ လိုအပ်တဲ့တန်ဖိုး တွေ သတ်မှတ်ပြီး Model ပေါ်မှာ `save()` Function ကို Run လိုက်ပါတယ်။

`save()` Function က Model အသစ်ဆိုရင် Server ထံ POST Request တစ်ခုကို ပေးပို့ပေးပါတယ်။ မူလရှိနေ တဲ့ Model ပေါ်မှာဆိုရင်တော့ Model ကို ပြုပြင်ခြင်းဖြစ်တယ်လို့ နားလည်ပြီး PUT Request တစ်ခုကို ပေးပို့ ပေးပါတယ်။ Model အသစ်ဟုတ်မဟုတ်ကို Model မှာ ID Attribute (`_id`) ရှိမရှိနဲ့ဆုံးဖြတ်ပါတယ်။ Model အသစ်ဆိုရင် ID Attribute က ရှိမှာမဟုတ်ပါဘူး။ Server မှာ တစ်ကြိမ်သိမ်းဖူးတဲ့ Model မှသာ ID Attribute က ရှိနေမှာပါ။

လက်ရှိ `save()` Function ကို Run တဲ့ Model က `_id` ရှိပြီး Model ဖြစ်တဲ့အတွက် Backbone ၏ PUT Request တစ်ခုကို Server ထံ ပေးပို့ပေးသွားမှာဖြစ်ပါတယ်။ ဒါတော့မှ Client-side မှာ Model ကို ပြုပြင်လိုက် တဲ့ တန်ဖိုးက Server-side မှာပါ သက်ရောက်သွားစေမှာဖြစ်ပါတယ်။

ပြဿနာက ကျွန်ုတ်တော်တို့၏ API Server မှာ Status ပြောင်းတဲ့လုပ်ငန်းအတွက် PUT Request ကို အသုံးမပြုစေ ပဲ PATCH ကို အသုံးပြုဖို့ သတ်မှတ်ထားပါတယ်။ ဒါကြောင့် Backbone ကို Request Method အနေနဲ့ PATCH ကိုသုံးပေးဖို့ ပြောဖို့လိုပါတယ်။ ဒါကြောင့် လိုင်းနံပါတ် (၅၃) မှာ `patch: true` ဆိုတဲ့တန်ဖိုးကို ထည့်သွင်းပေး ထားခြင်းပဲ ဖြစ်ပါတယ်။

ကျွန်ုတ်ဆောင်ချက်တွေကတော့ သဘာတရားခံပဲဆင်ဆင်တွေပဲမို့ စာဖတ်သူကိုယ်တိုင်ပဲ ရေးသားထားတဲ့ Code ကို ဖတ်ပြီး ဆက်လက်လေ့လာကြည့်လိုက်ပါ။

## 15.7 – App View – `views/appView.js`

ဆက်လက်ပြီး Main App View ကို `views/appView.js` ထဲမှာ ရေးသားပေးပါမယ်။

### JavaScript

```

1. var appView = Backbone.View.extend({
2.   el: "#app",
3.
4.   initialize: function() {
5.     this.render();
6.   },
7.
8.   render: function() {
9.     var nav = new navView();
10.    $("#nav").html( nav.render().el );

```

```

11.     app.verify(function() {
12.       app.issueList.fetch({
13.         success: function() {
14.           var list = new issueListView();
15.           $("#main").html( list.render().el );
16.
17.           var nav = new navView();
18.           $("#nav").html( nav.render().el );
19.
20.
21.           app.userList.fetch();
22.           app.loadIssueTypes();
23.           app.loadIssuePriorities();
24.           app.loadIssueStatuses();
25.           app.loadUserRoles();
26.         }
27.       });
28.     }, function() {
29.       var login = new loginView();
30.       $("#main").html( login.render().el );
31.     });
32.   }
33. });

```

ဒါ View အတွက် Element အနေနဲ့ Main Template ဖြစ်တဲ့ index.html မှာ ကြိုတင်သတ်မှတ်ထားတဲ့ #app Element ကို အသုံးပြုထားပါတယ်။ ထူးခြားချက်အနေနဲ့ initialize() Function အတွင်းမှာ render() Function ကို ခေါ်ယူထားပါတယ်။ အဓိပ္ပာယ်ကာ ဒါ View ကိုသုံးပြီး View Object တစ်ခု တည်ဆောက်လိုက်တာ နဲ့ render() Function က တစ်ခါတည်း အလုပ်လုပ်သွားမယ်ဆိုတဲ့ သဘောပဲဖြစ်ပါတယ်။

render() Function အတွင်းမှာတော့ app.verify() ကိုသုံးပြီး လက်ရှိ User ဟာ Authenticated User ဟုတ် မဟုတ်စိစစ်ထားပါတယ်။ Authenticated User ဆိုရင် Issue List View ကို ဖော်ပြပြီး၊ Authenticated User မဟုတ်ရင် တော့ Login View ကို ဖော်ပြန့် သတ်မှတ်ထားခြင်းဖြစ်ပါတယ်။

### 15.8 – Main Template – `index.html`

တစ်ခြား View တွေလိုအပ်ပေမယ့် App စတင်အလုပ်လုပ်ဖို့အတွက် လိုအပ်တဲ့အခြေခံ View တွေ ပါဝင်သွားပြီးမြှို့ ကျော် View တွေကိုခေါ်ထားပြီး Main Template ကို အရင်တည်ဆောက်ပါမယ်။ index.html မှာ အခုလို ရေးသား ပေးရမှာဖြစ်ပါတယ်။

#### HTML

```

1. <!DOCTYPE html>
2. <html lang="en">
3. <head>
4.   <meta charset="UTF-8">
5.   <title>Issue Tracking System</title>
6.   <link rel="stylesheet" href="bower_components/bootstrap/dist/
      css/bootstrap.min.css">
7.   <link rel="stylesheet" href="bower_components/bootstrap3-wysiwyg/dist/

```

```

bootstrap3-wysihtml5.min.css">
8.  <link rel="stylesheet" href="css/style.css">
9. </head>
10. <body>
11.  <div class="container" id="app">
12.    <div id="nav">
13.
14.    </div>
15.    <div id="main">
16.
17.    </div>
18.  </div>
19.
20.  <script src="bower_components/jquery/dist/jquery.min.js"></script>
21.  <script src="bower_components/bootstrap/dist/js/
22.          bootstrap.min.js"></script>
22.  <script src="bower_components/bootstrap3-wysiwyg/dist/
23.          bootstrap3-wysihtml5.all.min.js"></script>
23.  <script src="bower_components/DataTables/media/js/
24.          jquery.dataTables.min.js"></script>
24.  <script src="bower_components/datatables-responsive/js/
25.          dataTables.responsive.js"></script>
25.
26.  <script src="bower_components/underscore/underscore-min.js"></script>
27.  <script src="bower_components/backbone/backbone-min.js"></script>
28.
29.  <script src="bower_components/moment/min/moment.min.js"></script>
30.
31.  <script src="app.js"></script>
32.  <script src="issues.js"></script>
33.  <script src="users.js"></script>
34.
35.  <script src="views/appView.js"></script>
36.  <script src="views/navView.js"></script>
37.  <script src="views/loginView.js"></script>
38.  <script src="views/issueListView.js"></script>
39.  <script src="views/issueNewView.js"></script>
40.  <script src="views/issueDetailView.js"></script>
41.  <script src="views/issueEditView.js"></script>
42.  <script src="views/userListView.js"></script>
43.  <script src="views/userProfileView.js"></script>
44.  <script src="views/userEditView.js"></script>
45.  <script src="views/userNewView.js"></script>
46.
47.  <script>
48.    app.init();
49.  </script>
50. </body>
51. </html>

```

ထူးခြားတဲ့လုပ်ဆောင်ချက်တွေမပါဝင်ပဲ App မှာအသုံးပြုဖိုလိုတဲ့ Library ဖိုင်တွေ View ဖိုင်တွေနဲ့ Model, Collection ဖိုင်တွေကို တန်းစီချိတ်ဆက်ထားခြင်းသာဖြစ်ပါတယ်။ ထူးခြားချက်အနေနဲ့ #app Element အတွင်းထဲမှာ #nav Element အလွတ်တစ်ခုနဲ့ #main Element အလွတ်တစ်ခုတို့ပါဝင်တာကို သတိပြုပါ။ #nav Element အတွင်းမှာ Nav View ကို Render လုပ်မှာဖြစ်ပြီး #main Element အတွင်းမှာတော့ Main Content View ကို အခြေအနေ ပေါ်မှုတည် ဖော်ပြပေးသွားမှာပဲ ဖြစ်ပါတယ်။

လိုင်းနံပါတ် (၄၈) မှာ app.init() Function ကို ခေါ်ယူခြင်းအားဖြင့် App ကို အစပြုထားပါတယ်။ app.js ထဲမှာ ပြန်လည်လေ့လာကြည့်ရင် app.init() က Template ဖိုင်တွေကို Load လုပ်ပြီး App View တစ်ခု တည်ဆောက်ပေးထားတယ်ဆိုတာကို တွေ့ရမှာဖြစ်ပါတယ်။

ဒီအဆင့်ထိ ရေးသားပြီးစီးပြီးဆိုရင် App ကို စတင်စမ်းသပ်လိုက်ပါပြီ။ စတင်စမ်းသပ်နိုင်ဖို့အတွက် လက်ရှိ ရေးသားနေတဲ့ Client ကို Web Server တစ်ခုနဲ့ Run ပေးဖို့လိုပါတယ်။ ဒါကြောင့် http-server ဆိုတဲ့ NPM Package တစ်ခုကို Install လုပ်ပြီး Project Directory ထဲမှာ http-server Command ကို Run ပေးခြင်းအားဖြင့် စမ်းသပ်နိုင်ပါတယ်။

```
$ sudo npm install -g http-server
$ http-server

Starting up http-server, serving .
Hit CTRL-C to stop the server
```

http-server က Default Port အနေနဲ့ Port နံပါတ် 8080 ကို သုံးပြီးအလုပ်လုပ်ပေးသွားမှာဖြစ်ပါတယ်။ ဒါကြောင့် အခါန် Browser ကိုဖွင့်ပြီး [localhost:8080](http://localhost:8080) ကိုသွားကြည့်လိုက်ရင် လက်ရှိရေးသားနေတဲ့ Client App ကို တွေ့ရမှာပဲ ဖြစ်ပါတယ်။ စစ်ခြင်းမှာ Login View ကိုတွေ့ရမှာ ဖြစ်ပြီး Login ဝင်ပြီးတဲ့အခါမှာတော့ Issue List View ကို ရရှိမှာ ဖြစ်ပါတယ်။

တစ်ခုသတိပြုရမှာကတော့၊ ဒီ App ဟာ ကျွန်တော်တို့ အခန်း (၁၃) မှာ ရေးသားထားခဲ့တဲ့ Back-end Service ကို ဆက်သွယ်အလုပ်လုပ်မှာဖြစ်လို အဲဒီ Service ကိုလည်း Run ထားပေးဖို့လိုပါတယ်။ Back-end Service ကို ရေးသားခဲ့ စဉ်ကလက်ရှိ Client Run နေတဲ့ localhost:8080 ကို Access-Control-Allow-Origin နဲ့ Cross-domain Request ပြုလုပ်ခွင့်ကို ခွင့်ပြုထားခဲ့ပြီးဖြစ်တယ်ဆိုတာကိုလည်း သတိပြုသင့်ပါတယ်။

ကျွန် View တွေကိုတော့ Code ကိုပဲအခိုကဖော်ပြပါတော့မယ်။ တစ်ခုချင်းလိုက်ရှင်းမနေတော့ပါဘူး။ စာဖတ်သူ ကိုယ်တိုင် Code ကိုကြည့်ပြီး အမို့ပါယ်ကို နားလည်နိုင်လိမ့်မယ်လို ယူဆပါတယ်။

### 15.9 – New Issue View – `views/issueNewView.js`

Issue အသစ်ထည့်လိုရတဲ့ New Form ကို ဖော်ပြပေးတဲ့ View ဖြစ်ပါတယ်။ Form ကို Submit လုပ်လိုက်ရင် Issue Model တစ်ခုတည်ဆောက်ပြီး အဲဒီ Model ကို Issue List Collection ထဲမှာထည့်သွင်းထားပါတယ်။ ထူးခြားချက်အနေ နဲ့ လိုင်းနံပါတ် (၉) မှာ listenTo() ကို သုံးပြီး Issue List Collection မှာ Issue တစ်ခု ထည့်သွင်းလိုက်တဲ့အခါ Show Issue List လုပ်ဆောင်ချက်ကို လုပ်ပေးဖို့ သတ်မှတ်ထားတာကို သတိပြုသင့်ပါတယ်။

```
1. var issueNewView = Backbone.View.extend({
2.   tagName: "div",
3.
4.   events: {
5.     "submit #new-issue-form": "create"
6.   },
7.
8.   initialize: function() {
9.     this.listenTo(app.issueList, 'add', this.showIssueList);
10.  },
11.
12.  render: function() {
13.    var data = {
14.      priorities: app.issuePriority,
15.      types: app.issueType
16.    };
17.
18.    this.$el.html( app.hookTemplate("issue-new", data) );
19.    return this;
20.  },
21.
22.  create: function() {
23.    var summaryInput = $("#summary");
24.    if(!summaryInput.val()) {
25.      summaryInput.parent().addClass('has-error');
26.      summaryInput.focus();
27.      return false;
28.    }
29.
30.    var summary = summaryInput.val();
31.    var detail = $("#detail").html();
32.    var type = $("#type").val();
33.    var priority = $("#priority").val();
34.
35.    var model = new app.issueModel({
36.      "detail": detail,
37.      "summary": summary,
38.      "type": type,
39.      "typeLabel": app.issueType[type],
40.      "priority": priority,
41.      "priorityLabel": app.issuePriority[priority]
42.    });
43.
44.    model.save(null, {
45.      wait: true,
46.      success: function(res) {
47.        app.issueList.add(res);
48.      }
49.    });
50.
51.    return false;
52.  },
53.
54.  showIssueList: function(data) {
55.    var list = new issueListView();
56.    $("#main").html( list.render().el );
57.  }
58.});
```

### 15.10 – Edit Issue View – `views/issueEditView.js`

Issue Summary နဲ့ Detail ကို ပြင်ဆင်လိုက်တဲ့ Edit Form ကို ဖော်ပြပေးတဲ့ View ဖြစ်ပါတယ်။ Issue တစ်ခု အတွက် ပါဝင်တဲ့အချက်အလက်အများအပြားရှိပေါ်မယ့် ဒီ Form ကိုတော့ Issue Summary နဲ့ Detail ကို ပြင်ဆင်ဖို့အတွက်သာ အသုံးပြုမှာဖြစ်ပါတယ်။ ကျွန်ုင်အချက်အလက်တွေဖြစ်တဲ့ Priority, Status, Assigned To စတဲ့ အချက်အလက်တွေကို ပြင်ဆင်နိုင်တဲ့ လုပ်ဆောင်ချက်ကိုတော့ Issue Detail မှာ ထည့်သွင်းရေးသားခဲ့ပြီး ဖြစ်ပါတယ်။

```

1. var issueEditView = Backbone.View.extend({
2.   tagName: "div",
3.
4.   events: {
5.     "submit #issue-edit-form": "updateIssue",
6.     "click #close-edit": "showIssueDetail"
7.   },
8.
9.   initialize: function() {
10.     //
11.   },
12.
13.   render: function() {
14.     this.$el.html( app.hookTemplate("issue-edit", this.model.toJSON()) );
15.     return this;
16.   },
17.
18.   updateIssue: function() {
19.     var summaryInput = $("#summary");
20.     if(!summaryInput.val()) {
21.       summaryInput.parent().addClass('has-error');
22.       summaryInput.focus();
23.       return false;
24.     }
25.
26.     var summary = summaryInput.val();
27.     var detail = $("#detail").html();
28.
29.     var that = this;
30.     this.model.save({
31.       "detail": detail,
32.       "summary": summary
33.     }, {
34.       wait: true,
35.       success: function(res) {
36.         that.showIssueDetail();
37.       }
38.     });
39.
40.     return false;
41.   },
42.
43.   showIssueDetail: function(data) {
44.     var detail = new issueDetailView({ model: this.model });
45.     $("#main").html( detail.render().el );
46.   }
47. });

```

### 15.11 – User List View – `views/userListView.js`

User စာရင်းကို ဖော်ပြပေးတဲ့ View ဖြစ်ပါတယ်။ ထူးခြားချက်အနေနဲ့ Delete လုပ်ဆောင်ချက်ကို List နဲ့အတူ တစ်ခါ တည်း တဲ့ဖက်သတ်မှတ်ထားပါတယ်။ ဒါကြောင့် User တစ်ယောက်ကို ပယ်ဖျက်လိုရင် သက်ဆိုင်ရာ User နဲ့အတူ တွဲဖက်ဖော်ပြထားတဲ့ Delete Button ကို နှိပ်ပြီး ပယ်ဖျက်နှင့်မှာ ဖြစ်ပါတယ်။

#### JavaScript

```

1. var userView = Backbone.View.extend({
2.   tagName: "tr",
3.
4.   events: {
5.     "click .username": "viewProfile",
6.     "click .edit-user": "editUser",
7.     "click .delete-user": "deleteUser"
8.   },
9.
10.  render: function() {
11.    var index = app.userList.indexOf(this.model);
12.    this.model.set("index", index + 1);
13.
14.    this.$el.html( app.hookTemplate("user", this.model.toJSON()) );
15.    return this;
16.  },
17.
18.  viewProfile: function() {
19.    var profile = new userProfileView({ model: this.model });
20.    $("#main").html( profile.render().el );
21.  },
22.
23.  editUser: function() {
24.    var user = new userEditView({ model: this.model });
25.    $("#main").html( user.render().el );
26.  },
27.
28.  deleteUser: function() {
29.    this.model.destroy({
30.      wait: true,
31.      success: function() {
32.        var users = new userListView();
33.        $("#main").html( users.render().el );
34.      }
35.    });
36.  }
37. });
38.
39. var userListView = Backbone.View.extend({
40.   tagName: "div",
41.
42.   events: {
43.     "click #new-user": "newUser"
44.   },
45.
46.   render: function() {
47.     this.$el.html( app.hookTemplate("user-list") );
48.
49.     var that = this;

```

```

50.     app.userList.each(function(user) {
51.         var view = new userView({ model: user });
52.         $("tbody", that.$el).append( view.render().el );
53.     });
54.
55.     return this;
56. },
57.
58. newUser: function() {
59.     var view = new userNewView();
60.     $("#main").html( view.render().el );
61. }
62. });

```

### 15.12 – User New View – `views/userNewView.js`

User အသစ်ထည့်သွင်းလိုရတဲ့ Form ကို ဖော်ပြပေးတဲ့ View ဖြစ်ပါတယ်။ New Issue View နဲ့ လုပ်ဆောင်ချက် သိပ် မကွာပါဘူး။ ပြီးခဲ့တဲ့ Form တွေမှာရော ဒီ Form မှာပါ ကျွန်တော်တို့ အလွယ်လုပ်ထားမိတာ တစ်ခုတော့ ရှိပါတယ်။ Form Validation လုပ်ဆောင်ချက်ကို Form နဲ့အတူ တွဲဖက်ထားခြင်းဖြစ်ပါတယ်။ လက်တွေမှာ Validate လို လုပ်ဆောင်ချက်မျိုးကို Model နဲ့တွဲပြီးသတ်မှတ်တာကို ပိုပြီး စနစ်ကျစေမှာဖြစ်ပါတယ်။ BackboneJS ကလည်း ဒီရည် ရွယ်ချက်နဲ့ပဲ validate() Function ကို Model မှာ ထည့်သွင်းပေးထားပါတယ်။ ဒါပေမယ့် နမူနာမှာတော့ Validation လုပ်ငန်းကို View ထဲမှာပဲ ထည့်သွင်းထားပါတယ်။ လက်တွေမှာ Model ထဲမှာ စစ်သင့်တယ်ဆိုတာကို သတိ ပြုစေလိုပါတယ်။

#### JavaScript

```

1. var userNewView = Backbone.View.extend({
2.     tagName: "div",
3.
4.     events: {
5.         "submit #new-user-form": "create"
6.     },
7.
8.     initialize: function() {
9.         this.listenTo(app.userList, 'add', this.showUserList);
10.    },
11.
12.    render: function() {
13.        this.$el.html( app.hookTemplate("user-new") );
14.        return this;
15.    },
16.
17.    create: function() {
18.        var fullName = $("#full-name").val();
19.        var loginId = $("#login-id").val();
20.        var email = $("#email").val();
21.        var role = $("#role").val();
22.        var password = $("#password").val();
23.
24.        var err = 0;
25.        if(!fullName) {
26.            err++;
27.            $("#full-name").parent().addClass('has-error');

```

```

28.      }
29.
30.      if(!loginId) {
31.          err++;
32.          $("#login-id").parent().addClass('has-error');
33.      }
34.
35.      if(!email) {
36.          err++;
37.          $("#email").parent().addClass('has-error');
38.      }
39.
40.      if(!password) {
41.          err++;
42.          $("#password").parent().addClass('has-error');
43.      }
44.
45.      if(password != $("#password-again").val()) {
46.          err++;
47.          $("#password").parent().addClass('has-error');
48.          $("#password-again").parent().addClass('has-error');
49.      }
50.
51.      if(err) return false;
52.
53.      var model = new app.userModel({
54.          "fullName" : fullName,
55.          "loginId" : loginId,
56.          "email" : email,
57.          "role" : role,
58.          "password" : password
59.      );
60.
61.      model.save(null, {
62.          wait: true,
63.          success: function(res) {
64.              app.userList.add(res);
65.          }
66.      });
67.
68.      return false;
69.  },
70.
71.  showUserList: function() {
72.      var list = new userListView();
73.      $("#main").html( list.render().el );
74.  }
75. });

```

### 15.13 – User Edit View – `views/userEditView.js`

User နဲ့ Password အပါအဝင် အချက်အလက်တွေကို ပြင်ဆင်နိုင်တဲ့ Form ကို ဖော်ပြပေးတဲ့ View ဖြစ်ပါတယ်။ ပိုကောင်းစေလိုရင်တော့ Password ပြင်လိုရတဲ့ Form နဲ့ အမြားအချက်အလက်တွေကို ပြင်လိုရတဲ့ Form ခဲ့မြား ထား သင့်ပါတယ်။ နှမူနာမှာတော့ Password အပါအဝင် အချက်အလက်အားလုံးကို ပြင်ဆင်ဖို့အတွက် Form တစ်ခုတည်းကို သာ သုံးထားပါတယ်။

## JavaScript

```
1. var userEditView = Backbone.View.extend({
2.   tagName: "div",
3.
4.   events: {
5.     "submit #edit-user-form": "updateUser"
6.   },
7.
8.   initialize: function() {
9.     //
10.   },
11.
12.   render: function() {
13.     this.$el.html( app.hookTemplate("user-edit", this.model.toJSON()) );
14.     return this;
15.   },
16.
17.   updateUser: function() {
18.     var fullName = $("#full-name").val();
19.     var loginId = $("#login-id").val();
20.     var email = $("#email").val();
21.     var role = $("#role").val();
22.     var password = $("#password").val();
23.
24.     var err = 0;
25.     if(!fullName) {
26.       err++;
27.       $("#full-name").parent().addClass('has-error');
28.     }
29.
30.     if(!loginId) {
31.       err++;
32.       $("#login-id").parent().addClass('has-error');
33.     }
34.
35.     if(!email) {
36.       err++;
37.       $("#email").parent().addClass('has-error');
38.     }
39.
40.     if(err) return false;
41.
42.     var that = this;
43.     this.model.save({
44.       "fullName" : fullName,
45.       "loginId" : loginId,
46.       "email" : email,
47.       "role" : role,
48.       "roleLabel" : app.userRole[role],
49.       "password": password
50.     },
51.     {
52.       wait: true,
53.       success: function() {
54.         that.showUserList();
55.       }
56.     });
57.     return false;
58.   },
59.
```

```

59.
60.     showUserList: function() {
61.         var list = new userListView();
62.         $("#main").html( list.render().el );
63.     }
64. });

```

### 15.14 – User Profile View – `views/userProfileView.js`

User တစ်ဦးရဲ့ အချက်အလက်တွေကို ဖော်ပြပေးတဲ့ Profile View ဖြစ်ပါတယ်။ User ရဲ့ အမည်၊ Email စတဲ့ အချက် အလက်တွေနဲ့အတူ၊ အဲဒီ User ထည့်သွင်းထားတဲ့ Issue အရေအတွက်နဲ့ အဲဒီ User ကို Assigned လုပ် ထားတဲ့ Issue အရေအတွက်ကိုပါ ထည့်သွင်းဖော်ပြပေးမှာ ဖြစ်ပါတယ်။

#### JavaScript

```

1. var userProfileView = Backbone.View.extend({
2.     tagName: "div",
3.
4.     events: {
5.         "click #edit-user": "userEdit"
6.     },
7.
8.     initialize: function() {
9.         //
10.    },
11.
12.     render: function() {
13.         var that = this;
14.         this.model.set("submitCount", app.issueList.where({
15.             "submittedBy": that.model.id
16.         }).length);
17.
18.         this.model.set("assignCount", app.issueList.where({
19.             "assignedTo": that.model.id
20.         }).length);
21.
22.         this.$el.html(app.hookTemplate("user-profile", this.model.toJSON()));
23.         return this;
24.    },
25.
26.     userEdit: function() {
27.         var user = new userEditView({ model: this.model });
28.         $("#main").html( user.render().el );
29.     }
30. });

```

ဒါဟာ ကျွန်တော်တိရဲ့ Issue Tracking System Web Client အတွက် လိုအပ်တဲ့ Code အပြည့်အစုံပြန်ပါတယ်။ အထက်မှာပြောခဲ့သလိုပဲ၊ လက်တွေမှာ တစ်လုံးချင်းကူးယူရေးသားမယ့်အစား၊ ရေးသားထားတဲ့ Code ကိုသာ နားလည် အောင်လေ့လာသင့်ပါတယ်။ သဘောသဘာဝကို နားလည်ပြီဆိုတော့မှုသာ ကိုယ်တိုင်ရေးသားဖန်တီး ကြည့်သင့်ပါတယ်။ Code အပြည့်အစုံကို အောက်ပါလိပ်စာမှာ Download ရယူနိုင်ပါတယ်။

<https://github.com/eimg/issues-backbone/archive/master.zip>

## Conclusion

App တစ်ခုတည်ဆောက်ရာမှာ BackboneJS လို M-V-C Code Architecture Framework တွေကို အသုံးပြုရတဲ့ ရည်ရွယ်ချက်ကတော့ ရေရှည်မှာ ပြပိုင်ထိမ်းသိမ်းရလွယ်ကူစေဖို့ ဖြစ်ပါတယ်။ BackboneJS ရဲ့ အကူအညီနဲ့ App ရဲ့ အစိတ်အပိုင်းတွေကို ကဏ္ဍအလိုက်ခွဲခြားထားနိုင်တဲ့ အတွက် ပြပိုင်ထိမ်းသိမ်းရလွယ်ကူမှုအတွက် အထောက်အကူဖြစ်စေမှာ ဖြစ်ပါတယ်။ ဥပမာ - အသွေးပြုပိုင်ဖော်ပြပု Structure ကို ပြပိုင်လိုရင် သက်ဆိုင်ရာ Template ကိုပြင်နိုင်ပါတယ်။ အခြားလုပ်ဆောင်ချက်တွေ မပါဝင်ပဲ Template သက်သက်ဖြစ်လို ပြပိုင်ဆင်ဖြည့်စွက်ရတာ လွယ်ကူပြီး၊ ပြပိုင်လိုက်တဲ့ အတွက်လည်း အခြားလုပ်ဆောင်ချက်တွေကို သက်ရောက် ထိခိုက်သွားစရာမရှိပါဘူး။ အလားတူပဲ View တစ်ခုအတွက် သတ်မှတ်ထားတဲ့ Event တွေကို ပြပိုင်ဆင်လိုရင် သက်ဆိုင်ရာ View မှာပြပိုင်ပါတယ်။ ဒီလိုပြပိုင်ဆင်မှုကြောင့် အခြား View တွေအတွက် သက်မှတ်ထားတဲ့ Event တွေကို သက်ရောက်ထိခိုက်သွားမှာ မဟုတ်ပါဘူး။

ဖော်ပြခဲ့တဲ့ နမူနာမှာ File Name တွေ၊ Model, View, Collection Name တွေ၊ Function Name တွေနဲ့ Variable Name တွေကိုပေးတဲ့ အခါး၊ သက်ဆိုင်ရာလုပ်ဆောင်ချက်အမိပါယ်ကိုပေါ်လွင်အောင် ပေးထားတာကိုလည်း သတိပြုစေလိုပါ တယ်။ ဒီလိုပေးထားတဲ့ အတွက် File Name (သို့မဟုတ်) Function Name ကို ကြည့်လိုက်ယုံနဲ့ ဒါဘာအတွက်လည်း ဆိုတာကိုနားလည်နိုင်မှာဖြစ်လို ပြပိုင်ဆင်ထိမ်းသိမ်းရလွယ်ကူမှုပိုင်းမှာ အများကြီး အထောက်အကူဖြစ်စေမှာ ဖြစ်ပါတယ်။ ဒါကြောင့် မိမိကိုယ်တိုင် ရေးသားတဲ့ အခါမှာလည်း အမည်ပေးပုံလေး တွေကို အထူးကရပြသသုတေသနပါတယ်။ လုပ်ဆောင်ချက် အမိပါယ်ကို ပေါ်လွင်စေတဲ့ အမည်တွေ ပေထားမှသာ ပြပိုင်ဆင်ထိမ်းသိမ်းရလွယ်ကူတဲ့ Code ဖြစ်မှာ ဖြစ်ပါတယ်။

နမူနာကို တက်နိုင်သမျှရှိုးရင်းစေလိုတဲ့ အတွက် မဖြစ်မနေပါဝင်သင့်တဲ့ လုပ်ဆောင်ချက်တွေကိုသာ ထည့်သွင်းထားပါတယ်။ လက်တွေ့မှာထပ်မံဖြည့်စွက်စရာတွေရှိပါသေးတယ်။ ဥပမာ - Issue တွေကို Status အလိုက် သော်လည်ကောင်း Priority အလိုက် သော်လည်ကောင်း Filter လုပ်ပေးနိုင်တဲ့ လုပ်ဆောင်ချက်တွေ ထည့်သွင်းသင့်ပါတယ်။ Email Notification လို လုပ်ဆောင်ချက်တွေ ထည့်သွင်းသင့်ပါတယ်။ ကျွန်ုတ်တဲ့ အခန်း (၆) မှာဖော်ပြခဲ့တဲ့ Issue Tracking System တစ်ခု မှာပါဝင်သင့်တဲ့ လုပ်ဆောင်ချက်တွေ အပြည့်အစုံပါဝင်အောင် ဆက်လက်ဖြည့်စွက်သင့်ပါတယ်။

စာဖတ်သူအနေနဲ့ နမူနာ Source Code ကိုရယူပြီး လိုအပ်သလို ပြပိုင်ဆင်ဖြည့်စွက်ပြီး စမ်းသပ်ကြည့်စေလိုပါ တယ်။

တစ်ကြိမ်ရေးသားလိုက်ယုံနဲ့ Platform အမျိုးမျိုးမှာအလုပ်  
လုပ်နိုင်တဲ့ Cross-platform လုပ်ဆောင်ချက်ရရှိဖို့ဆိုတာ  
Software Development မှာ ယခင်ကတည်းကရှုခဲ့တဲ့  
အမိကစိန်ခေါ်မှုတစ်ရပ်ဖြစ်ပါတယ်။

### **Professional Web Developer (စာအုပ်)**

Web Standard, jQuery, PHP, MySQL, Ajax, CMS, MVC,  
HTML5, Mobile Web, Web Application Security စသည်  
အကြောင်းအရာများကို ရေးသားဖော်ပြထားသည့်စာအုပ်  
အောက်ပါလိပ်စာတွင် အခမဲ့ Download ရယူနိုင်သည်။

အခန်း(၁၆) – Building Clients (Hybrid Mobile App)

အခန်း (၁၃) မှာ ExpressJS ကိုအသုံးပြုပြီး ကျွန်ုတ်တို့တည်ဆောက်ထားခဲ့တဲ့ Issue Tracking System ဟာ API Service တစ်ခုဖြစ်တဲ့အတွက် HTTP အသုံးပြုဆက်သွယ်နိုင်တဲ့ မည်သည့် Client နဲ့မဆို ချိတ်ဆက်အလုပ်လုပ်နိုင်မှာ ဖြစ် ပါတယ်။ ပြီးခဲ့တဲ့အခန်းမှာ နှမူနာအနေနဲ့ Web Client တစ်ခုကို အဲဒီ API Service နဲ့ချိတ်ဆက်တည်ဆောက်ပုံကို ဖော်ပြ ခဲ့ပါတယ်။ အကယ်၍ အဲဒီ API နဲ့ချိတ်ဆက်ပြီး Andorid, iOS စတဲ့ Mobile Platform တွေမှာအလုပ်လုပ်တဲ့ Mobile Client တွေကို တည်ဆောက်မယ်ဆိုရင်လည်း တည်ဆောက်နိုင်မှာဖြစ်ပါတယ်။

## 16.1 – Types of Mobile Apps

Mobile App တွေကို အပ်စာသုံးစွဲနိုင်ပါတယ်။ Native App, Mobile Web App နဲ့ Hybrid App တို့ပဲဖြစ်ပါတယ်။

Native App ဆိုတာကတော့ သက်ဆိုင်ရာ Mobile Platform အတွက်ပေးထားတဲ့ SDK တွေနဲ့ ဆက်စပ်နည်းပညာတွေ ကို အသုံးပြုတည်ဆောက်ထားတဲ့ App များဖြစ်ပါတယ်။ Mobile Device များမှာ Install လုပ်ထည့်သွင်းထားနိုင်တဲ့ App အမျိုးအစားဖြစ်ပါတယ်။ Native App ရဲ့ ထူးခြားချက်ကတော့ Android အတွက်ဖုန်းထားတဲ့ App ကို Android Device တွေမှာသာ အသုံးပြုနိုင်မှာဖြစ်ပြီး၊ iOS အတွက် ရည်ရွယ်ဖုန်းထားတဲ့ App ကို iOS မှာသာ အသုံးပြုနိုင်မှာ ဖြစ်ပါတယ်။

ဒီနှစ်ခုကို ပေါင်းစပ်အသုံးပြထားတဲ့ နည်းစနစ်ကိုတော့ Hybrid App လိုပေါ်ပါတယ်။ App ကို တည်ဆောက်ရာ မှာ HTML5, CSS3, JavaScript စတဲ့ Web Technology များနဲ့ တည်ဆောက်ရပါတယ်။ Mobile Web Client တွေ ရဲ့ အားသာချက်ဖြစ်တဲ့ တစ်ကြိမ်ရေးလိုက်ယူနဲ့ Mobile Platform အမျိုးမျိုးမှာ အလုပ်လုပ်နိုင်တဲ့ အားသာချက် ကို ဆက်ခံ ရရှိပါတယ်။ ဒီနည်းစနစ်က Web Technology နဲ့ Native Technology ကို ပေါင်းစပ်အသုံးပြခြင်း ဖြစ်လို့ Device ရဲ့ စွမ်းဆောင်ရည်ကိုလည်း အတိုင်းအတာတစ်ခုထိအသုံးချုပ်ပါတယ်။ ဒီနည်းနဲ့ဖော်တီးထားတဲ့ App တွေကို Web Browser ဖွင့်စရာမလိုပဲ အသင့်အသုံးပြုနိုင်အောင်၊ Native App ကဲသို့ Install လုပ်ထားလို့ လည်း ရပါသေးတယ်။

နည်းစနစ် (၃) ပျိုးလုံးမှာ သူ့ကိုယ်ပိုင် အားသာချက်အားနည်းချက်တွေ ကိုယ်စီရိကြပြီး ဒီနေရာမှာတော့ ကျွန်တော်တို့ ပြီးခဲ့တဲ့အခန်းမှာ တည်ဆောက်ခဲ့တဲ့ Web Client ကို Cordova လို့ခေါ်တဲ့ နည်းပညာရဲ့အကူအညီနဲ့ Mobile Device တွေမှာ Install လုပ်အသုံးပြုနိုင်တဲ့ Hybrid App ဖြစ်အောင် တည်ဆောက်ပုံကို ဖော်ပြသွားမှာ ဖြစ်ပါတယ်။

## 16.2 – Apache Cordova

Apache Cordova ဆိုတာဟာ Hybrid App တွေတည်ဆောက်ရာမှာ အသုံးပြုနိုင်တဲ့ Mobile Development Framework တစ်ခုဖြစ်ပါတယ်။ App တည်ဆောက်ဖို့အတွက် HTML5, CSS3, JavaScript စတဲ့ Web Technology များကိုအသုံးပြုရပြီး၊ App ကို သက်ဆိုင်ရာ Mobile Platform နဲ့အတူပါဝင်လာတဲ့ Webview လို့ ခေါ်တဲ့ Embedded Browser ထဲမှာ အလုပ်လုပ်ပေးမှာဖြစ်ပါတယ်။ Mobile Device ရဲ့ Native လုပ်ဆောင်ချက် တွေကို JavaScript နဲ့ ရယူအသုံးပြုနိုင်စေဖို့ API တွေပေးထားတဲ့အတွက် App ကနေ Device ရဲ့ Native လုပ်ဆောင်ချက်တွေကို ရယူအသုံးပြု နိုင်မှာဖြစ်ပါတယ်။

ဒီနေရာမှာ Cordova ရဲ့ လုပ်ဆောင်ချက်အားလုံးကို ဖော်ပြသွားမှာမဟုတ်ပဲ Cordova ကိုအသုံးပြုပြီး ကျွန်တော် တို့ တည်ဆောက်ထားတဲ့ Web Client ကို Hybrid App တစ်ခုဖြစ်အောင် ပြောင်းလဲပုံကိုသာဖော်ပြသွားမှာဖြစ်ပါတယ်။ Cordova အကြောင်းအသေးစိတ်ကို [cordova.apache.org](http://cordova.apache.org) မှာ လေ့လာနိုင်ပါတယ်။

Cordova ကို NPM နဲ့ အခုလို့ Install ပြုလုပ်နိုင်ပါတယ်။

```
$ sudo npm install -g cordova
```

Install လုပ်ပြီးတဲ့အခါ Project တစ်ခုစတင်ဖို့အတွက် cordova create ကို အသုံးပြုနိုင်ပါတယ်။

```
$ cordova create issues com.yourname.issues 'Issue Tracking System'
```

cordova create အတွက် Parameter (၃) ခုပေးလိုက်တာကို တွေ့ရမှာဖြစ်ပါတယ်။ ရှုံးက issues က Project Directory Name ဖြစ်ပါတယ်။ သူ့နောက်က com.yourname.issues ကတော့ Reverse Domain Style ID ဖြစ်ပါတယ်။ Domain Name ကိုပြောင်းပြန်ပေးလိုက်ခြင်းဖြစ်ပြီး Unique Package Name အနေနဲ့ သုံးပါတယ်။ yourname နေရာမှာ ကိုယ့်ရဲ့ကိုယ်ပိုင် Domain Name နဲ့အစားထိုးပေးလို့ရပါတယ်။ နောက်သုံးက Parameter ကတော့ App Title ဖြစ်ပါတယ်။

cordova create က တည်ဆောက်ပေးလိုက်တဲ့ issues Directory ကိုဝင်ကြည့်ရင်အခလိုတွေရမှာဖြစ်ပါတယ်။

```

issues
  config.xml
  hooks
    README.md
  platforms
  plugins
  www
    css
    img
    index.html
    js
  
```

config.xml ထဲမှာ App Setting တွေပါဝင်မှာဖြစ်ပါတယ်။ ဖွင့်ကြည့်လိုက်ရင် အခလို တွေရနိုင်ပါတယ်။

## XML

```

1. <?xml version='1.0' encoding='utf-8'?>
2. <widget id="com.yourname.issues" version="0.0.1"
   xmlns="http://www.w3.org/ns/widgets"
   xmlns:cdv="http://cordova.apache.org/ns/1.0">
3.   <name>Issues Tracking System</name>
4.   <description>
5.     ...
6.   </description>
7.   <author email="dev@cordova.apache.org" href="http://cordova.io">
8.     Apache Cordova Team
9.   </author>
10.  <content src="index.html" />
11.  <plugin name="cordova-plugin-whitelist" version="1" />
12.  <access origin="*" />
13.  <allow-intent href="http:///*/*" />
14.  <allow-intent href="https:///*/*" />
15.  <allow-intent href="tel:*" />
16.  <allow-intent href="sms:*" />
17.  <allow-intent href="mailto:*" />
18.  <allow-intent href="geo:*" />
19.  <platform name="android">
20.    <allow-intent href="market:*" />
21.  </platform>
22.  <platform name="ios">
23.    <allow-intent href="itms:*" />
24.    <allow-intent href="itms-apps:*" />
25.  </platform>
26. </widget>
  
```

id အနေနဲ့ ကျွန်ုတ်တို့ပေးလိုက်တဲ့ Reverse Domain ကို အသုံးပြုထားတာကို တွေ့ရမှာဖြစ်ပြီး <name>, <description>, <author> စာတဲ့ Element တွေရဲ့ တန်ဖိုးကိုလည်း သင့်တော်သလို ပြုပြင်ထားနိုင်ပါတယ်။ ထူးခြားချက်အနေနဲ့ လိုင်းနံပါတ် (၁၂) က <access origin="\*" /> ကို သတိပြုကြည့်ပါ။ ပုံမှန်

ဆိုရင် Web Browser တွက Cross-domain Request တွကို ခွင့်ပြုလေ့မရှိကြပါဘူး။ Cordova ကတော့ Default အနေနဲ့ <access origin="\*"/> လိုသတ်မှတ်ပေးထားတဲ့အတွက် Cross-domain Request တွကို ခွင့်ပြုပေးသွားမှာ ဖြစ်ပါတယ်။ ဒါကြောင့် API Service ကို ဘယ်မှာပဲထားထား ဒါ App ကနေ Access လုပ်အသုံးပြုနိုင်မှာဖြစ်ပါတယ်။

Project Directory ထဲက www Directory ကတော့ App Directory ဖြစ်ပါတယ်။ ကျွန်ုတ်တို့ ရရေးသားပြီးဖြစ်တဲ့ BackboneJS Client Code တွကို အဲဒီ Directory ထဲမှာ ကူးယူထည့်သွင်းပေးထားရမှာ ဖြစ်ပါတယ်။

ဆက်လက်ပြီး ရည်ရွယ်အသုံးပြုလိုတဲ့ Platform ကို cordova add platform နဲ့ထည့်သွင်းပေးနိုင်ပါတယ်။ ဥပမာ -

```
$ cordova platform add android

npm http GET https://registry.npmjs.org/cordova-android/4.0.0
npm http 200 https://registry.npmjs.org/cordova-android/4.0.0
npm http GET https://registry.npmjs.org/cordova-android/-/cordova-android-4.0.0.tgz
npm http 200 https://registry.npmjs.org/cordova-android/-/cordova-android-4.0.0.tgz
Adding android project...
Creating Cordova project for the Android platform:
  Path: platforms/android
  Package: com.eimaung.issues
  Name: Issues Tracking System
  Activity: MainActivity
  Android target: android-22
Copying template files...
Android project created with cordova-android@4.0.0
Discovered plugin "cordova-plugin-whitelist" in config.xml. Installing to the project
Fetching plugin "cordova-plugin-whitelist@1" via npm
npm http GET https://registry.npmjs.org/cordova-plugin-whitelist
npm http 304 https://registry.npmjs.org/cordova-plugin-whitelist
Installing "cordova-plugin-whitelist" for android
```

ဒီလိုထည့်သွင်းနိုင်ဖို့အတွက် စက်ထဲမှာ Android SDK တော့အဆင်သင့်ရှိရပါတယ်။ Cordova က Android SDK နဲ့အတူ ပါဝင်လာတဲ့ Platform Tools တွေကို အသုံးပြုမှာဖြစ်တဲ့အတွက် အကယ်၍ Android SDK မရှိသေးရင် အောက်ပါလိပ်စာမှာ ကြိုတင်ရယူ ထည့်သွင်းထားပေးရမှာ ဖြစ်ပါတယ်။

<http://developer.android.com/sdk/index.html>

Android အပြင် အခြား Platform တွေကို ထည့်သွင်းလိုတယ်ဆိုရင်လည်း အခုလို ထည့်သွင်းပေးနိုင်ပါတယ်။

```
cordova platform add ios
cordova platform add amazon-fireos
cordova platform add blackberry10
cordova platform add firefooxos
cordova platform add wp8
cordova platform add windows
```

ထည့်သွင်းထားတဲ့ Platform တစ်ခုကို ပြန်ထုတ်လိုရင်တော့ cordova platform remove ကို သုံးနိုင်ပါတယ်။

www Directory ထဲမှာ Web Client Source Code ကိုကူးယူထည့်သွင်းပြီးရင် index.html ကို အခုလိုပြုပြင်ပေးဖို့လိုအပ်ပါတယ်။

## HTML

```
1. <!DOCTYPE html>
2. <html lang="en">
3. <head>
...
10. </head>
11. <body onLoad="onLoad()">
...
49. <script>
50.     function onLoad() {
51.         document.addEventListener("deviceready", function() {
52.             app.init();
53.         }, false);
54.     }
55. </script>
56. </body>
57. </html>
```

ပြောင်းလဲဖို့လိုတဲ့အပိုင်းကိုပဲ ရွေးချယ်ဖော်ပြထားခြင်းဖြစ်ပါတယ်။ ကျွန်ုင်အပိုင်းတွေ အပြောင်းအလဲမရှိပါဘူး။ Webview ထဲမှာ JavaScript Code တွေ မှန်မှန်ကန်ကန်အလုပ်လုပ်ဖို့ရင် deviceready Event ထဲမှာ ထည့်သွင်းပေးဖို့လိုတဲ့ အတွက် app.init() ကို deviceready Event ထဲမှာ အလုပ်လုပ်စေခြင်းဖြစ်ပါတယ်။ တနည်းအားဖြင့် ကျွန်ုင်တော်တို့ရဲ့ Web Client Code ကို Device နဲ့ Webview အသင့်ဖြစ်တော့မှ စတင်အလုပ်လုပ်ဖို့ သတ်မှတ်လိုက်ခြင်းပဲ ဖြစ်ပါတယ်။

App ကို Android Device တစ်ခုမှာ Run ကြည့်နိုင်ဖို့အတွက် Device ကို ကွန်ပျူးတာမှာ USB ကြိုးနဲ့ချိတ်ပေးရပါမယ်။ ပြီးရင် Device Setting တွေထဲက USB Debugging ကို ဖွင့်ထားပေးရပါမယ်။ USB ကြိုးချိတ်ပြီး USB Debugging ကို ဖွင့်ပြီးတဲ့အခါ cordova run နဲ့ App ကို စမ်းသပ်နိုင်ပြီဖြစ်ပါတယ်။

```
$ cordova run android
```

ကျွန်တော်တို့၏ App က USB နဲ့ ချိတ်ဆက် ထားတဲ့ Android Device ပေါ်မှာ အလုပ်လုပ်သွားမှာ ဖြစ်ပါတယ်။ cordova run က .apk Installer ဖိုင်တစ်ခုကိုလည်း တစ်ခါတည်းတည်ဆောက်ပေးသွားပါတယ်။ platforms/android/build Directory ထဲမှာ ရှိနေမှာဖြစ်ပါတယ်။ ဒီနည်းနဲ့ Hybrid App တစ်ခုကို Android အတွက်သာမက အခြား Mobile Platform တွေအတွက်ပါ အလွယ်တစ်ကူ တည်ဆောက်ယူနိုင်မှာ ဖြစ်ပါတယ်။

## Conclusion

နမူနာအရ အခြားဖြည့်စွက်မှုတွေမပါဝင်ပဲ ကြိုးတင်ရေးသားထားတဲ့ Web App တစ်ခုကို Install လုပ်လို့ရတဲ့ App ဖြစ် အောင် ပြောင်းလိုက်ခြင်းသာဖြစ်ပါတယ်။ လက်တွေမှာ Device ရဲ့ လုပ်ဆောင်ချက်တွေကို ထိရောက် အောင် ထည့်သွင်း အသုံးချိန်မှသာ Hybrid App ဖြစ်သွားခြင်းရဲ့ အကျိုးရလဒ်ကိုရရှိမှာ ဖြစ်ပါတယ်။

Device ရဲ့ Battery Status, Network Status, Motion Sensor, Orientation, Geolocation, Camera, File, Vibration စတဲ့ Hardware ပိုင်းဆိုင်ရာ လုပ်ဆောင်ချက်တွေနဲ့ Contact, Dialog စတဲ့ Mobile OS ရဲ့ လုပ်ဆောင်ချက် တွေကို App ကနေ ရယူအသုံးပြုလိုတယ်ဆိုရင် Plugin တွေကို အသုံးပြုနိုင်ပါတယ်။ အသုံးပြုနိုင်တဲ့ Cordova Plugin စာရင်းကို အောက်ပါလိပ်စာမှာ ဆက်လက်လေ့လာနိုင်ပါတယ်။

<http://plugins.cordova.io/>

# အပိုင်း (၂)

Deployment and Scaling

Software တစ်ခုရဲ့ လုပ်ဆောင်ချက်တွေ တိုးတက်  
ပြောင်းလဲနေတာနဲ့အမျှ အဲဒီ Software ကို Run ပေးရမယ့်  
Server Environment တလည်း အလိုက်သုတေသန  
လိုက်လုပြောင်းလဲပေးနိုင်ဖို့လိုပါတယ်။

**Ubuntu - သင့်အတွက် Linux (စာအုပ်)**  
Ubuntu Linux အသုံးပြုနည်းအား Installation မှ စတင်၍ System Configuration နှင့် Development Environment တည်ဆောက်  
ခြင်းအထိ အဆင့်လိုက်ရေးသား ဖော်ပြထားသည့်စာအုပ်  
အောက်ပါလိပ်စာတွင် အခမဲ့ ရယူနိုင်သည်။

# အခန်း(၁၇) - Deployment with Docker

Software တစ်ခုအတွက် လိုအပ်တဲ့ Production Server Environment တစ်ခု တည်ဆောက်ရာမှာ အမိက ကြိုတွေ့ရ နိုင်တဲ့ Challenge (၃) ခု ရှိပါတယ်။ အဲဒါတွေကတော့ -

1. လက်ရှိ Development Machine ပေါ်မှာ အလုပ်လုပ်နေတဲ့ Software ဟာ Production Server ပေါ်ရောက်တဲ့အခါ၊ လိုအပ်တဲ့ Package Dependencies မကိုက်ညီတဲ့အတွက် အလုပ်မလုပ်တော့တဲ့ ပြဿနာ ရှိတက်ပါတယ်။
2. Software ကို စဉ်ဆက်မပြတ်အဆင့်ဖြင့်တင်နေတဲ့အတွက် အဲဒီ Software ကို လက်ခံအလုပ်လုပ်ပေးရမယ့် Production Server ကလည်း အလိုက်သင့်လိုက်လုပ်ပြောင်းလဲပေးဖို့ လိုနိုင်ပါတယ်။
3. SOA နည်းစနစ်ကိုသုံးပြီး Software တစ်ခုကို Service ငယ်များအဖြစ် ခဲ့ခြားတည်ဆောက် တဲ့အခါ၊ Service တွေအလုပ်လုပ်ဖို့အတွက် လိုအပ်တဲ့ Environment မတူတဲ့အတွက်၊ Production Server တစ်ခု ပေါ်မှာ Service နှစ်ခုသုံးခု ပူးတွဲအလုပ်မလုပ်နိုင်တဲ့ပြဿနာ ရှိနိုင်ပါတယ်။

## 17.1 – Development Machine vs. Production Server

Software တစ်ခုမှာ လက်ရှိရေးသားစားသပ်နေတဲ့ Development Code နဲ့ အများအသုံးပြုနိုင်ဖို့ Build လုပ်ထားတဲ့ Production Code ဆိုပြီး ရှိတဲ့ကြောင်းကို အပိုင်း (၂) မှာ ဖော်ပြုခြိုးဖြစ်ပါတယ်။ Code မှာ Development နဲ့ Production ဆိုပြီးနှစ်ပိုင်းရှိသလိုပဲ၊ အဲဒီ Code ကို Run မယ့် ကွန်ပျူးတာတွေမှာလည်း Development Machine နဲ့ Production Server ဆိုပြီးနှစ်မျိုးရှိမှာ ဖြစ်ပါတယ်။ Development Machine ဆိုတာ ကျွန်ုတ်တို့ Code ရေးသား ဖို့ နေ့စဉ်အသုံးပြုနေတဲ့ Laptop ကွန်ပျူးတာဖြစ်နိုင်ပါတယ်။ Production Server ဆိုတာကတော့ အများ ဆက်သွယ် ရယူ အသုံးပြုနိုင်အောင် Public IP Address (သို့မဟုတ်) Domain Name တစ်ခုနဲ့ စီစဉ်ပေးထားတဲ့ စွမ်းဆောင်ရည် မြင့် ကွန်ပျူးတာဖြစ်နိုင်ပါတယ်။

ရေးသားထားတဲ့ Software ဟာ ကျွန်ုတ်တို့၏ Development Machine မှာ အလုပ်လုပ်နေသလိုပဲ၊ Production Server မှာလဲ ပုံမှန် ဆက်လက်အလုပ်လုပ်နေဖို့လိုပါတယ်။ တနည်းအားဖြင့် Production Server ကို ကျွန်ုတ်တို့၏ Development Machine နဲ့ တူညီတဲ့ Environment ရှိအောင် စီစဉ်ထားဖို့ လိုအပ်မှာဖြစ်ပါတယ်။ Software အတွက် လိုအပ်တဲ့ Operating System, Web Server, Database Server, Programming Language နဲ့ အခြားလိုအပ်တဲ့ Package တွေကို Production Server ထည့်သွင်းထားဖို့ လိုနိုင်ပါတယ်။

ဒီလိုထည်သွင်းရာမှာ Software တွေ Package တွေရဲ့ Version ကိုလည်း လိုအပ်ချက်နဲ့အညီ အတိအကျ ထည့်သွင်းထားဖို့ လိုနိုင်ပါတယ်။ ဥပမာ - Software ကို ရေးသားစဉ်က MongoDB 2.4 နဲ့ တွေဖက်ရေးသားထားတယ်ဆိုရင် Production Server မှာလဲ MongoDB 2.4 ကို ထည့်သွင်းထားပေးဖို့ လိုအပ်မှာဖြစ်ပါတယ်။

ပိုကောင်းတဲ့နောက်ဆုံး Version မြို့လို့ဆိုပြီး MongoDB 3.0 ကို Production Server မှာထည့်သွင်းထားမယ်ဆိုရင် Software က အဆင် ပြေပြေ အလုပ်လုပ်ဖို့မသေချာတော့ပါဘူး။ Development Machine မှာ အလုပ်လုပ်နေပြီး Production Server ကျတော့မှ အလုပ်မလုပ်တော့ဘူးဆိုတဲ့ ပြဿနာမျိုး ကြိုတွေ့ရနိုင်ပါတယ်။

အမြဲအဆင့်မြှင့်တင်မှုတွေ ပြုလုပ်နေတဲ့ Software တစ်ခုဟာ အသုံးပြုထားတဲ့ Package တွေလည်း အမြဲပြောင်းလဲနေ နိုင်ပါတယ်။ ဥပမာ – လက်ရှိ Software ရဲ့ Version 1.0 ကို NodeJS 0.12.2 ကိုအသုံးပြု တည်ဆောက်ထားတယ် ဆိုပါစို့။ ဒီ Software အဆင်ပြေပြေအလုပ်လုပ်စေဖို့အတွက် Production Server မှာ လည်း NodeJS 0.12.2 ကို အတိ အကျထည့်သွင်းထားဖို့ လိုနိုင်ပါတယ်။ နောက်တော့မှ Software ရဲ့ Version 2.0 ကို NodeJS 1.0 နဲ့ ပြောင်းလဲတည် ဆောက်တယ်ဆိုရင် Production Server မှာလည်း NodeJS 1.0 ကို ပြောင်းလဲထည့်သွင်းထားပေးဖို့ လိုနိုင်ပါတယ်။ ဒါပေမယ့် ချက်ခြင်းထည့်သွင်းလို့ မရသေးပါဘူး။ Version 2.0 ကို Release လုပ်တော့မယ်ဆိုမှ ထည့်သွင်းပေးရမှာပါ။ကာလတစ်ခုထဲ၊ Production Server မှာ Run နေတာက Software ရဲ့ Version 1.0 ဖြစ်ပြီး လက်ရှိ Development Machine ပေါ်မှာ ရောက်ရှိနေတာက Version 2.0 ဖြစ် နေတယ်ဆိုတာမျိုးလည်း ဖြစ်နိုင်ပါသေးတယ်။ ဒီလိုအခြေအနေ တွေကို ထိရောက်အောင် စီမံနိုင်ဖို့ဆိုတာ Challenge တစ်ခုဖြစ်ပါတယ်။

နောက်ပြဿနာတစ်ခုကတော့ အခြေခံလိုအပ်ချက်မတူတဲ့ Service တွေကို Server တစ်ခုတည်းပေါ်မှာ ယုံ့လုပ်တဲ့ အလုပ်လုပ် စေဖို့ကလည်း လိုအပ်နိုင်ပါသေးတယ်။ ဥပမာ – Service တစ်ခုက NodeJS 0.12.2 နဲ့တည်ဆောက် ထားပြီး နောက် Service တစ်ခုက NodeJS 1.0 နဲ့တည်ဆောက်ထားတယ်ဆိုတဲ့အခြေအနေမှာ၊ Service နှစ်ခု လုံးကို Server တစ်လုံး တည်းနဲ့ အလုပ်လုပ်နိုင်စေဖို့ဆိုရင် NodeJS Version နှစ်မျိုး ထည့်သွင်းထားရတော့မယ် သော့ ဖြစ်ပါတယ်။ တစ်ချို့ Package တွေက Version နှစ်မျိုးသုံးမျိုးကို ယုံ့လုပ်တဲ့ထည့်သွင်းထားလို့ ရတက်ပေ မယ်။ Package အများစုံကတော့ အဲဒီ လို ယုံ့လုပ်တဲ့ထည့်သွင်းထားလို့ ရမှာ မဟုတ်ပါဘူး။ ဒါလည်းပဲ နောက်ထပ် Challenge တစ်ခုပဲဖြစ်ပါတယ်။

ဒီ ပြဿနာတွေကို ဖြေရှင်းဖို့အတွက် အသုံးဝင်တဲ့နည်းပညာကတော့ Docker ဆိုတဲ့ နည်းပညာပဲ ဖြစ်ပါတယ်။

## 17.2 – Docker

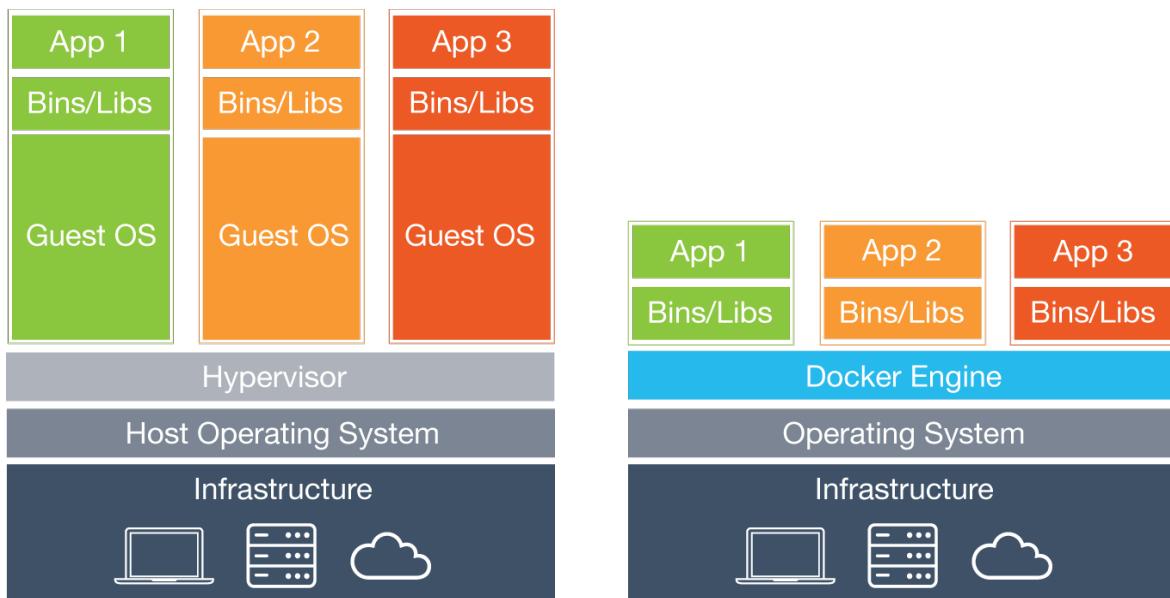
Docker ဟာ လက်ရှိမှာ အများရဲ့ဖို့တ်ဝင်စားမှာကို အထူးရရှိနေတဲ့ Tool တစ်ခုဖြစ်ပါတယ်။ LXC လို အတိုကောက်ခေါ်တဲ့ Linux Container နည်းပညာပေါ်မှာ အခြေခံထားခြင်း ဖြစ်ပါတယ်။ LXC ဆိုတာဟာ Virtualization နည်းပညာတစ်မျိုး ဖြစ်ပါတယ်။ ဒါပေမယ့် ကျွန်ုတ်တို့ အသုံးပြုကြလေ့ရှိတဲ့ VirtualBox, VMWare စတဲ့ Hypervisor အခြေခြား Virtual Machine တွေနဲ့ သဘောသဘာဝ ကဲ့ပြားပါတယ်။

VirtualBox တို့ VMWare တို့လို Hypervisor အခြေခြား Virtual Machine တွေဟာ Host Operating System ပေါ်မှာ Guest Operating System ကို ထပ်မံထည့်သွင်း အသုံးပြုနိုင်တဲ့ နည်းပညာဖြစ်ပါတယ်။ အခြေခံအားဖြင့် Host Operating System ဟာ Linux, Windows, Mac မည်သည့် OS မဆို ဖြစ်နိုင်သလို့ Guest OS အနေနဲ့လည်း မည်သည့် OS မဆိုဖြစ်နိုင်ပါတယ်။ ဥပမာ – Ubuntu OS ပေါ်မှာ VirtualBox နဲ့ Windows 7 ကို ထည့်သွင်းအသုံးပြု နိုင်ပါတယ်။ ဒီအခြေအနေမှာ Ubuntu OS ဟာ Host OS ဖြစ်ပြီး Windows 7 ဟာ Guest OS ဖြစ်ပါတယ်။ ဒီ အကြောင်းကို Ubuntu – သင့်အတွက် Linux ရဲ့ အခန်း (၁၁) မှာ ဖော်ပြထားပါတယ်။ Host OS နဲ့ Guest OS

အမျိုးအစား တူစရာမလိုသလို Host OS တစ်ခု ပေါ်မှာလည်း Guest OS အများအပြားရှိနိုင်ပါတယ်။ ပုံ (၁၇.၁) ရဲ့ ဘယ်ဘက်ခြမ်းမှာ Hypervisor အခြေခြား Virtual Machine တွေ အလုပ်လုပ်ပုံကို ဖော်ပြထားပါတယ်။



Docker Container တွေကတော့ Virtual Machine နဲ့ အနည်းငယ်ကွဲပြားပါတယ်။ Docker ဟာ Linux OS ပေါ်မှာ အလုပ်လုပ်တဲ့ Linux အခြေခြား နည်းပညာ ဖြစ်ပါ တယ်။ Docker ကိုအသုံးပြုပြီး Linux Kernel တစ်ခု ပေါ်မှာ သီးခြား Environment နဲ့အလုပ်လုပ်တဲ့ Linux OS တွေကို ထပ်မံထည့်သွင်းနိုင်ပါတယ်။ Virtual Machine အခေါ်အဝေါ်တွေကို သုံးပြီးပြောရမယ်ဆိုရင် Linux Host OS ပေါ်မှာ နောက်ထပ် Linux Guest OS တွေ ထပ်မံထည့် သွင်းနိုင်တဲ့ သဘော ပါပဲ။ Docker မှာတော့ ထပ်မံထည့် သွင်းထားတဲ့ OS တွေကို Guest OS လို့မခေါ်ပါတယ်။ OS Image တွေ တွေ Run နိုင်တဲ့ လုပ်ဆောင် ချက်ကိုလည်း Virtual Machine လို့မခေါ်ပါဘူး။ Container လို့မခေါ်ပါတယ်။ ပုံ (၁၇.၁) ရဲ့ ညာဘက်ခြမ်းမှာ Docker အခြေခြား Container နည်းပညာရဲ့ အလုပ်လုပ်ပုံကို ဖော်ပြထားပါတယ်။



ပုံ (၁၇.၁) - Hypervisor vs. Docker

Source – [docker.com](https://docker.com)

ပုံ (၁၇.၁) မှာယုဉ်တဲ့ဖော်ပြထားတဲ့ နည်းပညာနှစ်ခုကို လေ့လာကြည့်ရင် ထူးခြားချက်ကို သတိပြုမိမာပါ။ Hypervisor နည်းပညာမှာ၊ Host OS က Kernel, Library နဲ့ လိုအပ်တဲ့အစိတ်အပိုင်း အပြည့်အစုံပါဝင်တဲ့ OS ဖြစ်သလို၊ Guest OS တွေကလည်း Kernel, Library နဲ့ လိုအပ်တဲ့အစိတ်အပိုင်း အပြည့်အစုံပါဝင်တဲ့ OS တွေဖြစ်ကြပါတယ်။ Docker နည်းပညာမှာတော့ Host OS ကသာ Kernel, Library နဲ့ လိုအပ်တဲ့အစိတ်အပိုင်း အပြည့်အစုံပါဝင်တဲ့ OS ဖြစ်ပြီး Container ပေါ်က OS Image တွေကတော့ သီးခြား OS မဟုတ်ပဲ Host OS ကိုပဲ ပြန်လည်အသုံးပြထားခြင်း ဖြစ်ပါ တယ်။ Host OS ကို ပြန်လည်အသုံးပြထားပေမယ့် ကိုယ်ပိုင် Library နဲ့

Configuration တွေကိုယ်စိုက်တဲ့အတွက် မတူကွဲပြားတဲ့ OS များကဲသိ သီးခြား အလုပ်လုပ်ပေးနိုင်ကြခြင်း ဖြစ်ပါတယ်။

Virtual Machine ရဲ့ Guest OS တွေက ကိုယ်ပိုင် Kernel နဲ့ OS တွေဖြစ်ကြတဲ့အတွက် ပိုမိုပြည့်စုံလွှပ်လပ်တယ် လို့ ဆိုနိုင်ပါတယ်။ Host OS နဲ့ Guest OS လည်း အမျိုးအစား တူစရာမလိုပါဘူး။ ဒါပေမယ့် Host OS ကို ထည့်သွင်း ထားတဲ့ ကွန်ပျူးတာကတော့ အတော်လေး စွမ်းဆောင်ရည်မြင်မားဖို့လိုပါတယ်။ စွမ်းဆောင်ရည်မြင်မှသာ Guest OS အများအပြားကို လက်ခံအလုပ်လုပ်ပေးနိုင်မှာ ဖြစ်ပါတယ်။ CPU, Memory အစရှိတဲ့ System Resource တွေကို Host OS ရော့ Guest OS တွေကပါ စိတ်ပိုင်းရယူအသုံးပြုကြမှာဖြစ်ပါတယ်။ ဥပမာ – RAM 4 GB ရှိတဲ့ ကွန်ပျူးတာပေါ်မှာ Host OS အတွက် 2 GB အသုံးပြုမယ်ဆိုရင် RAM 1 GB အသုံးပြုတဲ့ Guest OS နှစ်ခု သာ ထပ်မံထည့်သွင်း အသုံးပြုနိုင်မှာ ဖြစ်ပါတယ်။

Container တွေကတော့ သီးခြား Kernel နဲ့အလုပ်လုပ်တဲ့ OS မဟုတ်တဲ့အတွက် စွမ်းဆောင်ရည်လိုအပ်ချက် သိပ်မရှိပါ ဘူး။ အသင့်အတင့် စွမ်းဆောင်ရည်ရှိတဲ့ Host OS ပေါ်မှာ Container နဲ့ OS Image အမြောက်အများ ထည့်သွင်းအသုံး ပြုနိုင်မှာဖြစ်ပါတယ်။ မတူကွဲပြားတဲ့ Environment လိုချင်ယုံသက်သက်နဲ့ Guest OS ကြီးတစ်ခု ထပ်မံထည့်သွင်းနေစရာ မလိုတော့ပဲ၊ မူလ Host OS ပေါ်မှာပဲ လိုအပ်သလို Setting လုပ်ထားတဲ့ OS Image တွေကို ခွဲခြားအသုံးပြုနိုင်တဲ့ အတွက်ဖြစ်ပါတယ်။ Docker ကို Ubuntu Linux မှာ အခုလို Install ပြုလုပ်နိုင်ပါတယ်။

```
$ sudo apt-get install docker.io
```

boot2docker လိုခေါ်တဲ့ နည်းပညာအကူအညီနဲ့ Windows, Mac နဲ့ အခြား Operating System တွေမှာလည်း Docker ကို Install ပြုလုပ်နိုင်ပါတယ်။ အောက်ပါလိပ်စာမှာ လေ့လာနိုင်ပါတယ်။

<https://docs.docker.com/installation/>

ဆက်လက်ဖော်ပြထားတဲ့ Command နမူနာတွေဟာ လက်တွေ့ စမ်းသပ်လိုရင် ကူးယူ စမ်းသပ်နိုင်တဲ့ နမူနာတွေ ဖြစ်ပါတယ်။

### 17.3 – Getting Docker Images

Docker ကို Install လုပ်ပြီးတဲ့အခါ အသုံးလိုရတဲ့ Docker Image တွေကို Download ရယူထားနိုင်ပါတယ်။ docker pull Command ကို အသုံးပြုရပါတယ်။ Ubuntu Docker Image ကို Install အခုလို ရယူနိုင်ပါတယ်။

```
$ sudo docker pull ubuntu
```

Git နဲ့ ကျွန်ုတ်တို့၏ Source Code ကို Version မှတ်တမ်း တင်ထားနိုင်သလိုပဲ Docker Image တွေကိုလဲ

Version အလိုက် မှတ်တမ်းတင် ထားနိုင်ပါတယ်။ နမူနာအရ Version အမျိုးမျိုးပါဝင်တဲ့ Ubuntu Image ကို ရယူပေးသွားမှာ ဖြစ်ပါတယ်။ သက်ဆိုင်ရာ Version တစ်ခုတည်းကိုပဲလိုချင်ရင် အခုလို ရယူရပါတယ်။

```
$ sudo docker pull ubuntu:latest
$ sudo docker pull ubuntu:14.04
```

နမူနာအရ Ubuntu Image ထဲက latest လို Tag လုပ်ထားတဲ့ Version နဲ့ 14.04 လို Tag လုပ်ထားတဲ့ Version တွေကိုသာ ရယူပေးသွားမှာဖြစ်ပါတယ်။ အသုတေသနရယူနိုင်တဲ့ Docker Image စာရင်းကို [registry.hub.docker.com](https://registry.hub.docker.com) မှာ လေ့လာနိုင်သလို docker search နဲ့လည်း ရှာဖွေနိုင်ပါတယ်။ ဥပမာ -

```
$ sudo docker search mongo
```

NAME	DESCRIPTION	STARS	OFFICIAL	AUTOMATED	mongo
MongoDB document databases provide high av...	753 [OK]				
tutum/mongodb	MongoDB Docker image - listens in port 2...	33		[OK]	
torusware/speedus-mongo	Always updated official MongoDB docker ima...	7		[OK]	
jacksoncage/mongo	Instant MongoDB sharded cluster	6		[OK]	
mongooseim/mongooseim-docker	MongooseIM server the latest stable version	2		[OK]	
knickers/mongo-express	Graphically manage your mongoDB container ...	2		[OK]	
19hz/mongo-container	Mongodb replicaset for coreos	1		[OK]	
...					
asteris/apache2.4-mongo	Apache2.4 + PHP + Mongo + mod_rewrite	0		[OK]	
appertis/hhvm-mongo	Docker image of HHVM with compiled Mongof... DEPRECATED - please use the official mongo...	0		[OK]	
ulexus/mongo	MongoDB base	0		[OK]	
razmo/mongo	PHP and MongoDB (drive)	0		[OK]	
rzani/php-mongo					

နမူနာမှာ docker search နဲ့ mongo Image ကို ရှာဖွေလိုက်တဲ့အခါ MongoDB Install လုပ်ထားပြီး အသင့် အသုံးပြုနိုင်တဲ့ Image စာရင်းကို ဖော်ပြုပေးတာကို တွေ့ရမှာ ဖြစ်ပါတယ်။ OFFICIAL Column မှာ [OK] ပြထားတဲ့ Image ဟာ Docker Team က ဖန်တီးပေးထားတဲ့ Official Image ဖြစ်ပြီး [OK] မပြထားတဲ့ Image တွေ ကတော့ အခြားသူတွေ ဖန်တီးပေးထားတဲ့ Unofficial Image တွေဖြစ်ပါတယ်။ Unofficial Image တွေရဲ့အမည် ကို username/image ပုံစံနဲ့ ဖော်ပြတာကိုလည်းသတိပြုကြည်ပါ။ ကျွန်ုတ်တို့ကိုယ်တိုင်တည်ဆောက်ထားတဲ့ Image တွေကိုလည်း ဆန္ဒရှိရင် DockerHub မှာ တင်ထားပေးနိုင်မှာဖြစ်ပါတယ်။

လက်ရှိ စက်ထဲမှာရှိနေတဲ့ Image စာရင်းကိုရယူလိုရင် docker images ကို အသုံးပြုနိုင်ပါတယ်။

```
$ sudo docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	VIRTUAL SIZE
ubuntu	nginx	906bf9539e9d	4 days ago	206.9 MB
ubuntu	14.04	d2a0ecff6fa	7 days ago	188.4 MB
hello-world	latest	91c95931e552	3 months ago	910 B

နမူနာမှာ ubuntu Image နှစ်ခုရှိနေပြီး hello-world Image တစ်ခုရှိနေတာကို တွေ့ရမှာဖြစ်ပါတယ်။ ubuntu Image နှစ်ခုထဲက တစ်ခုကို nginx လို Tag လုပ်ထားပြီး နောက်တစ်ခုကိုတော့ 14.04 လို Tag လုပ်ထားပါတယ်။

## 17.4 – Running Image in Docker Container

Image တစ်ခုကို Docker Container ထဲမှာ Run လိုရင် docker run ကို အသုံးပြုရပါတယ်။

```
$ sudo docker run ubuntu:14.04 echo 'hello world'
hello world
```

နဲ့မှန်အရာ ubuntu:14.04 Image ကို Docker Container ထဲမှာ Run သွားပြီး echo 'hello world' ဆိတ္တဲ့ Command ကို အဲဒီ Container ထဲမှာ အလုပ်လုပ်စေလိုက်ခြင်း ဖြစ်ပါတယ်။ ရရှိလာတဲ့ hello world Message ဟာ Host OS ပေါ်မှာ အလုပ်လုပ်သွားလို့ ရရှိလာခြင်းမဟုတ်ပဲ Container ထဲမှာ echo Command အလုပ် လုပ်သွားတဲ့အတွက် ရရှိလာခြင်းဖြစ်ပါတယ်။

Container ထဲမှာ Run နေတဲ့ Image ကို Terminal နဲ့ ငင်ရောက် စီမံလိုရင်တော့ -t Option နဲ့ -i Option တို့ကို တွဲဖက် အသုံးပြုရပါတယ်။ -t က Terminal ဆိတ္တဲ့အဓိပ္ပာယ်ဖြစ်ပြီး -i ကတော့ Interactive ဆိတ္တဲ့အဓိပ္ပာယ်ပဲ ဖြစ်ပါ တယ်။ docker run နဲ့အတူ -t -i (သို့မဟုတ်) -ti လို့ တွဲဖက်အသုံးပြုနိုင်ပါတယ်။

```
$ sudo docker run -t -i ubuntu:14.04 /bin/bash
root@090b21537ff8:/#
```

နဲ့မှန်အရာ ubuntu:14.04 Image ကို Container ထဲမှာ Run ပြီး Command တွေ Run လို့ရတဲ့ Prompt တစ်ခုကို Terminal မှာ ဖော်ပြလာမှာပဲဖြစ်ပါတယ်။ နဲ့မှန်အနေနဲ့ NodeJS ကို အခုလုံး Install လုပ်နိုင်ပါတယ်။

```
root@090b21537ff8:/# apt-get update && apt-get install -y nodejs
```

apt-get update နဲ့ Package Repo ကို Update လုပ်လိုက်ပြီး apt-get install နဲ့ NodeJS ကို Install လုပ်လိုက်ခြင်းဖြစ်ပါတယ်။ အခုခုံရင် ကျွန်ုတ်တို့ရဲ့ ubuntu:14.04 Image ဟာ မူလ Image နဲ့ မတူတော့ပဲ NodeJS ဆိတ္တဲ့ Package အသစ်တစ်ခုနဲ့ လိုအပ်တဲ့ Dependency တွေ ဖြည့်စွက်ပါဝင်သွားပြီ ဖြစ်ပါတယ်။

လက်ရှိ အလုပ်လုပ်နေတဲ့ Container စာရင်းကို ရယူလိုရင်တော့ docker ps ကို အသုံးပြုနိုင်ပါတယ်။ နောက် Terminal အသစ်တစ်ခုဖွင့်ပြီး အခုလုံး စမ်းသပ်ကြည့်နိုင်ပါတယ်။

```
$ sudo docker ps
CONTAINER ID  IMAGE          COMMAND
090b21537ff8  ubuntu:14.04  /bin/bash

```

CREATED	STATUS	NAMES
4 minutes ago	Up 4 minutes	berserk_bohr

နှမူနာအရ လက်ရှိ ubuntu:14.04 Image ကို Run နေကြောင်းတွေ့ရမှာဖြစ်ပါတယ်။ ဖော်ပြလာတဲ့စာရင်းရဲ့ ရှေ့ဆုံးမှာ Container Id နဲ့ နောက်ဆုံးမှာ Container Name တို့ပါဝင်လာတာကို သတိပြုပါ။ နှမူနာအရ လက်ရှိ Container ကို berserk\_bohr လို့ Docker က အလိုအလျောက် အမည်တစ်ခုပေးထားပါတယ်။

## 17.5 – Saving Image Update

NodeJS Install လုပ်ထားတဲ့ လက်ရှိ Container ကို သီးခြား Image တစ်ခုအနေနဲ့ သိမ်းထားရှိပါတယ်။ docker commit Command ကို အသုံးပြုရပါတယ်။ Save လုပ်လိုတဲ့ Container ရဲ့ ID သို့မဟုတ် Name ကို စွဲဖက်ပေးရပါတယ်။

```
$ sudo docker commit -m 'Installed NodeJS' -a 'Ei Maung' berserk_bohr ubuntu:node
ecc87910db51819c68414d2400e5a2a88b7090c6efb35eb1e7ee681502f8384a

$ sudo docker images

REPOSITORY          TAG           IMAGE ID            CREATED             VIRTUAL SIZE
ubuntu              node          ecc87910db51   29 seconds ago   221 MB
ubuntu              nginx         906bf9539e9d   4 days ago        206.9 MB
ubuntu              14.04         d2a0ecffe6fa   7 days ago        188.4 MB
hello-world         latest        91c95931e552   3 months ago      910 B
```

အခန်း (၅) မှာ Git အကြောင်းလေ့လာခဲ့စဉ်က Version မှတ်တမ်းတင်ပဲနဲ့ ပေးရပါတယ်။ -m Option နဲ့ Commit Message ကိုပေးရပြီး -a Option နဲ့ Commit ပြုလုပ်သူအမည်ကို ပေးရပါတယ်။ ဆက်လက်ပြီး Container Name (သို့မဟုတ်) ID ကို ပေးရပြီး နောက်ဆုံးက Image အသစ်အတွက် ပေးလိုတဲ့ အမည်ကို ပေးရပါတယ်။

နှမူနာအရ Image ကို ubuntu ဆိုတဲ့အမည်နဲ့သာသိမ်းပြီး နောက်ကနေ node လို့ TAG လုပ်ပေးထားပါတယ်။ ဒါကြောင့် NodeJS Install လုပ်ထားပြီးဖြစ်တဲ့ Ubuntu Image တစ်ခုကို ubuntu:node ဆိုတဲ့အမည်နဲ့ ရရှိသွားမှာ ဖြစ်ပါတယ်။

Image တစ်ခုကို အမည်ပြောင်းလိုရင် (သို့မဟုတ်) Tag ပြောင်းလိုရင် docker tag ကို အသုံးပြုနိုင်ပါတယ်။

```
$ sudo docker tag ubuntu:node ubuntu:nodejs
$ sudo docker images

REPOSITORY          TAG           IMAGE ID            CREATED             VIRTUAL SIZE
ubuntu              node          ecc87910db51   4 minutes ago   221 MB
ubuntu              nodejs         ecc87910db51   4 minutes ago   221 MB
ubuntu              14.04         d2a0ecffe6fa   7 days ago        188.4 MB
hello-world         latest        91c95931e552   3 months ago      910 B
```

နှမူနာအရ ubuntu:node Image ကို ubuntu:nodejs အနေနဲ့ အမည်ပြောင်းပေးထားတာကို တွေ့ရမှာ ဖြစ်ပါတယ်။ ထူးခြားချက်အနေနဲ့ Docker က ubuntu:node ကိုလည်း ဒီအတိုင်း ချုံထားပေးတာကို တွေ့ရ

နိုင်ပါတယ်။ Image တစ်ခုကို ပယ်ဖျက်လိုရင်တော့ docker rmi ကို အသုံးပြုရပါတယ်။

```
$ sudo docker rmi ubuntu:nodejs
Untagged: ubuntu:nodejs

$ sudo docker images

REPOSITORY      TAG      IMAGE ID      CREATED      VIRTUAL SIZE
ubuntu          node     ecc87910db51  6 minutes ago  221 MB
ubuntu          14.04   d2a0ecffe6fa  7 days ago   188.4 MB
hello-world     latest   91c95931e552  3 months ago  910 B
```

## 17.6 – Creating Image with Dockerfile

ဖြည့်စွက်ပြင်ဆင်မှုတွေပြုလုပ်ထားတဲ့ Image တစ်ခုကို docker commit နဲ့ သီးခြား Image အဖြစ် သိမ်းဆည်းနိုင်သလို Docker Image အသစ်တွေ တည်ဆောက်ဖို့အတွက် Dockerfile လိုခေါ်တဲ့ Configuration ဖိုင်တစ်မျိုးကို လည်း အသုံးပြုနိုင်ပါတယ်။ ရေးသားပုံနှုနာက ဒီလိပါ –

### Config

```
1. FROM ubuntu:14.04
2. RUN apt-get update && apt-get install -y mongodb
3. RUN mkdir -p /data/db
4. EXPOSE 27017
5. CMD ['/usr/bin/mongod']
```

FROM Keyword ကိုသုံးပြီး အခြေခံရမယ့် Image ကို သတ်မှတ်ပါတယ်။ နှုနာအရ ubuntu:14.04 Image ပေါ်မှာ Image အသစ်ကို အခြေခံတည်ဆောက်သွားမှာ ဖြစ်ပါတယ်။ RUN Keyword ကို သုံးပြီး Image တည်ဆောက် စဉ်မှာ အလုပ်လုပ်စေလိုတဲ့ Command စာရင်းကို ပေးရပါတယ်။ နှုနာအရ apt-get နဲ့ Package Repository ကို Update လုပ်ပြီး mongodb ကို Install လုပ်ခိုင်းထားပါတယ်။ MongoDB အလုပ်လုပ်ဖို့အတွက် လိုအပ်နိုင်တဲ့ /data/db Directory ကို နောက်ထပ် RUN Keyword တစ်ခုနဲ့ ဆက်လက် တည်ဆောက်ထားပါတယ်။

EXPOSE Keyword ကိုတော့ Container ထဲက Service တွေကို Host OS ကနေ ရယူအသုံးပြုနိုင်အောင် ခွင့်ပြုပေးဖို့ အသုံးပြုရပါတယ်။ နှုနာအရ MongoDB ရဲ့ Default Port ဖြစ်တဲ့ 27017 ကို Host OS ကနေ ရယူအသုံးပြုခွင့် ပြုထားပါတယ်။

CMD Command ကတော့ Image ကို Run တဲ့အခါ အလိုအလျောက် အလုပ်လုပ် သွားစေလိုတဲ့ Command ကို သတ်မှတ်ဖို့ သုံးနိုင်ပါတယ်။ နှုနာအရ MongoDB Server ကို စတင်ပေးနိုင်တဲ့ mongod Command ကို Image အလိုအလျောက် Run ပေးဖို့ သတ်မှတ်ထားပါတယ်။

RUN နဲ့ CMD မတူပါဘူး။ RUN က Image တည်ဆောက်စဉ်မှာ အလုပ်လုပ်စေလိုတဲ့ Command တွေသတ်မှတ်ဖို့

အသုံး ပြုရပြီး CMD ကိုတော့ Image ကို Container ထဲမှာ စတင်စဉ် အလုပ်လုပ်စေလိုတဲ့ Command ကို သတ်မှတ်ရခြင်းဖြစ်ပါတယ်။ ဒါကြောင့် Dockerfile တစ်ခု ထဲမှာ RUN Keyword ကို လိုသလောက်အသုံးပြုနိုင်ပေမယ့် CMD Keyword ကိုတော့ တစ်ကြိမ်သာ အသုံးပြုသင့်ပါတယ်။ CMD Keyword တစ်ခုထက် ပိုသုံးထားရင် Docker က နောက်ဆုံးတစ်ခုကိုသာ အသုံးပြုပေးမှာဖြစ်ပါတယ်။

ရရှိလာတဲ့ Dockerfile ကို အသုံးပြုပြီး Image တည်ဆောက်နိုင်ဖို့အတွက် docker build ကို သုံးရပါတယ်။ -t Option နဲ့ Image ရဲ့ အမည်ကို သတ်မှတ်ပေးနိုင်ပါတယ်။

```
$ sudo docker build -t ubuntu:mongo .
```

နုတေသနမှာ နောက်ဆုံးက Dot ကို သတိပြုပါ။ Dot နေရာမှာ Dockerfile တည်ရှိရာ Directory ကို ပေးရမှာဖြစ်ပါတယ်။ နုတေသနမှာ docker build Command ကို Run တဲ့ Directory ထဲမှာ Dockerfile တည်ရှိတယ်လို ယူဆပြီး Dot ကို Location အနေနဲ့ အသုံးပြုထားခြင်း ဖြစ်ပါတယ်။ Build လုပ်ငန်းစဉ်ပြီးမြောက်သွားတဲ့အခါ MongoDB အသင့် Install လုပ်ထားပြီးဖြစ်တဲ့ ubuntu:mongo ဆိုတဲ့ Image ကို စတင်အသုံးပြုနိုင်ပြီးဖြစ်ပါတယ်။ Dockerfile အတွင်းမှာအသုံးပြုနိုင်တဲ့ ကျွန်ုတ် Keyword တွေကို အောက်ပါလိပ်စာ မှာ ဆက်လက်လေ့လာနိုင်ပါတယ်။

<http://docs.docker.com/reference/builder/>

Image ကို Build လုပ်ယူပြီးနောက်မှ Dockerfile ကို ပြင်ဆင်ဖြည့်စွက်မယ်ဆိုရင် docker build ကို နောက်တစ်ကြိမ် ထပ် Run လိုက်ယုံပါပဲ။ docker build က အစအဆုံး Build ပြန်လုပ်မနေပဲ Dockerfile မှာ ပြုလိုက်လိုက်တဲ့ပြင်ဆင်မှုတွေနဲ့အညီ Image ကို Update လုပ်ပေးသွားမှာဖြစ်ပါတယ်။

## 17.7 – Running Container as Daemon

Image တစ်ခုကို Container ထဲမှာ Run လိုက်ယုံမဟုတ်ပဲ အဲဒီ Container ကို Background Service (Daemon) အနေနဲ့ ဆက်လက် အလုပ်လုပ်စေလိုရင် -d Option ကို အသုံးပြုနိုင်ပါတယ်။ ဥပမာ -

```
$ sudo docker run -d -p 27017:27017 ubuntu:mongo
```

နုတေသနအရ ubuntu:mongo Image ကို Background Service အနေနဲ့ Container ထဲမှာ Run ပေးသွားမှာ ဖြစ်ပါတယ်။ -p Option ကတော့ Port Binding အတွက်ဖြစ်ပါတယ်။ 27017:27017 လိုပေးထားတဲ့အတွက် Host OS ရဲ့ Port 27017 နဲ့ Container ရဲ့ Port 27017 တို့ကို Bind လုပ်ပေးသွားမှာဖြစ်ပါတယ်။ တနည်းအားဖြင့် Container ထဲက Port 27017 မှာအလုပ်လုပ်နေတဲ့ MongoDB Server ကို Host OS ကနေ ရယူအသုံးပြုနိုင်သွားမှာ ဖြစ်ပါတယ်။

လက်ရှိ docker ps နဲ့ လက်ရှိအလုပ်လုပ်နေတဲ့ Container စာရင်းကိုရယူလိုက်တဲ့အခါ Background Service အနေနဲ့ Run ထားတဲ့ Container ပါဝင်လာတာကို တွေ့ရမှာပဲဖြစ်ပါတယ်။ Background Service အနေနဲ့ အလုပ်လုပ်နေတဲ့ Container တစ်ခုကို ရပ်လိုရင်တော့ docker stop ကို အသုံးပြနိုင်ပါတယ်။

```
$ sudo docker ps
CONTAINER ID        IMAGE               COMMAND
959db209f4cb      ubuntu: mongo      /usr/bin/mongod
73f4d6ae4a1a      ubuntu: 14.04      /bin/sh -c 'while tr
$ sudo docker stop silly_archimedes
```

docker stop နဲ့အတူ Container အမည် သို့မဟုတ် ID ကိုပေးပြီး Background Service အဖြစ် Run နေတဲ့ Container ကို ရပ်နိုင်ပါတယ်။ dock stop က Container ကို ရပ်ယုံပဲရပ်လိုက်တာပါ၊ အပြီး ပယ်ဖျက်မပြစ်ပါဘူး။ ဒါကြောင့် လိုအပ်လို ပြန်စလိုရင် docker start နဲ့ ပြန်စနိုင်ပါသေးတယ်။

```
$ sudo docker start silly_archimedes
```

Container တွေထဲက တစ်ခုကို အပြီးဖျက်လိုရင်တော့ docker rm ကို သုံးရမှာဖြစ်ပါတယ်။

```
$ sudo docker stop silly_archimedes
$ sudo docker rm silly_archimedes
```

ဖြည့်စွက်မှတ်သားသင့်တာကတော့ --name Option ဖြစ်ပါတယ်။ Container တစ်ခုကို Run တဲ့အခါ Container အမည်ကို Docker က အလိုအလျောက် သတ်မှတ်ပေးသွားလေ့ ရှိပါတယ်။ အဲဒီလို Docker ကပေးတဲ့ အမည်ကို မသုံးလိုပဲ ကိုယ်တိုင် Container Name သတ်မှတ်လိုရင် --name Option ကို အသုံးပြနိုင်ခြင်းပဲဖြစ်ပါတယ်။ ဥပမာ -

```
$ sudo docker run -ti --name mymongo ubuntu: 14.04 /bin/bash
```

အခုနောက် Terminal တစ်ခုမှာ docker ps ကို Run ကြည့်ရင်၊ ကျွန်ုတ်တို့ သတ်မှတ်ပေးလိုက်တဲ့ mymongo ဆိုတဲ့ အမည်ကို Container အမည်အဖြစ် အသုံးပြုပေးထားတာကို တွေ့ရမှာဖြစ်ပါတယ်။

```
$ sudo docker ps
CONTAINER ID        IMAGE               COMMAND
959db209f4cb      ubuntu: 14.04      /bin/bash
$
```

လက်ရှိ အလုပ်လုပ်နေတဲ့ Container တွေသာ မက Run ခဲ့တဲ့ Container အားလုံးစာရင်းကိုလိုချင်ရင်တော့ docker ps ကို -a Option နဲ့ တွေသုံးနိုင်ပါတယ်။

### 17.8 – Data Volume in Container

Container ထဲမှာ ဖိုင်တွေ Data တွေ သိမ်းဆည်းဖို့အတွက် Volume တွေ သတ်မှတ်ပေးနိုင်ပါတယ်။ ဥပမာ - /data ဆိုတဲ့ Volume တစ်ခု Container ထဲမှာ အသုံးပြုလိုရင် အခုလို သတ်မှတ်နိုင်ပါတယ်။

```
$ sudo docker run -d -v /data ubuntu:14.04
```

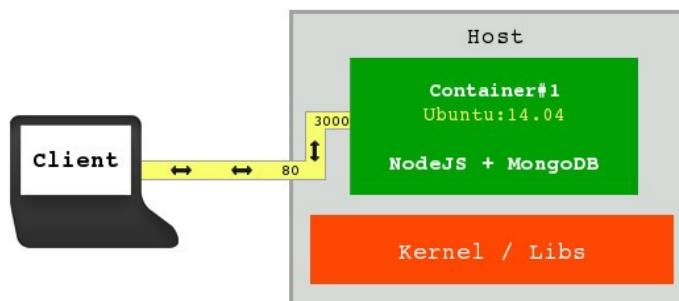
-v Option နဲ့အတူ သတ်မှတ်လိုတဲ့ Volume ကို တွဲဖက်ပေးရခြင်းဖြစ်ပါတယ်။ လက်ရှိ Host OS ထဲက Directory တစ်ခုကို Data Volume အနေနဲ့ အသုံးပြုစေလိုရင်တော့ အခုလို အသုံးပြုနိုင်ပါတယ်။

```
$ sudo docker run -d -v /var/www:/data ubuntu:14.04
root@cef44bf69c38:/# cd data
root@cef44bf69c38:/data# ls
sample.txt logo.png
```

နောက်မှာပေးထားတဲ့ /var/www နေရာမှာ အသုံးပြုလိုတဲ့ Host OS Directory တည်နေရာကို ထည့်သွင်းပေးရ မှာ ဖြစ်ပါတယ်။ နောက်မှာ Host ရဲ့ /var/www ထဲက ဖိုင်တွေဟာ Container ထဲက Image ရဲ့ /data မှာ ပါဝင် သွားမှာဖြစ်ပါတယ်။ ဒီနည်းနဲ့ Build လုပ်ထားတဲ့ Project တည်ရှုရာ Directory ကို Docker Container Run တဲ့ အချင့်မှာ Image နဲ့အတူ တခါတည်း ပါဝင်သွားအောင် စီစဉ်ထားနိုင်စေမှာဖြစ်ပါတယ်။

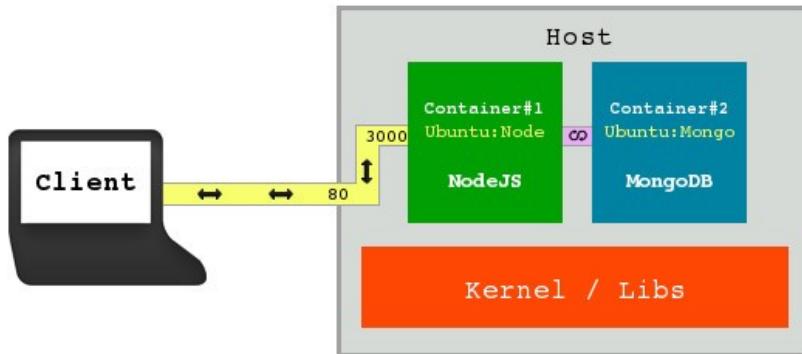
### 17.9 – Linking container

Service တစ်ခုအလုပ်လုပ်ဖို့အတွက် လိုအပ်တဲ့ Package တွေကို Docker Image တစ်ခုအနေနဲ့စုစုပေါင်းထားနိုင်ပါတယ်။ ဥပမာ - NodeJS နဲ့ MongoDB ကို အသုံးပြုထားတဲ့ App တစ်ခုအတွက် NodeJS ရော MongoDB ကိုပါ Image တစ်ခုထဲမှာ တစ်ခါတည်း ထည့်သွင်းထားနိုင်ပါတယ်။



ပုံ (၁၇.၂) - Docker Image with NodeJS + MongoDB

နောက်တစ်နည်းအနေနဲ့ လိုအပ်တဲ့ Package တွေကို သီးခြား Image အနေနဲ့ထားနိုင်ပါတယ်။ ဥပမာ – NodeJS ပါဝင်တဲ့ Image တစ်ခု၊ MongoDB ပါဝင်တဲ့ Image တစ်ခု၊ နှစ်ခုခဲ့ထားပြီး အဲဒီ Image တွေကို Run ပေးတဲ့ Container တွေကို Link လုပ်ပြီး အသုံးပြုနိုင်ခြင်းဖြစ်ပါတယ်။



### ပုံ (၁၇.၃) - Linking Docker Containers

ဥပမာအနေနဲ့ ကျွန်ုင်တော်တို့ အစိုင်း (၄) မှာ ExpressJS နဲ့ တည်ဆောက်ခဲ့တဲ့ Issue Tracking System API အတွက် Image တစ်ခုကို Dockerfile သုံးပြီး အခုလိုတည်ဆောက်ပါမယ်။

#### Config

```
1. FROM ubuntu:14.04
2. RUN apt-get update && apt-get install -y nodejs nodejs-legacy
3. RUN mkdir -p /src
4. COPY . /src
5. EXPOSE 3000
6. CMD ["node", "/src/index.js"]
```

ubuntu:14.04 ကို အခြေခံတဲ့ Image တစ်ခု တည်ဆောက်မှာဖြစ်ပါတယ်။ nodejs နဲ့ nodejs-legacy ဆိုတဲ့ Package နှစ်ခုကို Install လုပ်ထားပါတယ်။ ပြီးတဲ့အခါ /src လိုက်ခေါ်တဲ့ Directory လည်း တည်ဆောက်ထားပါသေးတယ်။ လိုင်းနံပါတ် (၄) မှာ COPY Keyword နဲ့ လက်ရှိ Host Directory ထဲက ပိုင်အားလုံးကို Image ၏ /src Directory ထဲမှာ ကူးယူထည့်သွင်းပေးဖို့ သတ်မှတ်ထားပါတယ်။ ဒါကြောင့် လက်ရှိ Directory ထဲမှာ ကြိုတင် ရေးသားထားတဲ့ Issue Tracking API Source Code ကို ထားပေးရမှာပါ။ Image ကို Build လုပ်စဉ်မှာ Source Code တွေကို Image ထဲ Docker က ကူးယူထည့်သွင်းပေးသွားမှာ ဖြစ်ပါတယ်။

ဆက်လက်ပြီး Port 3000 ကို Expose လုပ်ဖို့သတ်မှတ်ထားပါတယ်။ Image စွဲ Run တဲ့အချိန်မှာ node Command နဲ့ /src/app.js ကို Run ပေးဖို့လဲ သတ်မှတ်ထားပါတယ်။ ဒါ Dockerfile ကို docker build နဲ့ အခုလို Build လုပ်ပေးရပါမယ်။

```
$ sudo docker build -t ubuntu:node .
```

ဒီနည်းနဲ့ NodeJS ကို Install လုပ်ထားယုံသာမက Issue Tracking API Source Code ပါ အတစ်ခါတည်းပါဝင်တဲ့ ubuntu:node Image ကို ရရှိမှာဖြစ်ပါတယ်။ ရရှိလာတဲ့ Image ကို အခုလို Run ပြီး စတင်အသုံးပြနိုင်ပြုဖြစ်ပါတယ်။

```
$ sudo docker run -d -p 80:3000 ubuntu:node
```

ဒါပေမယ့် Issue Tracking API က MongoDB ကို အသုံးပြုထားပါတယ်။ လက်ရှိ Image မှာ MongoDB ကို ထည့်သွင်းမထားပါဘူး။ ဒါကြောင့် ပြီးခဲ့တဲ့နမူနာမှာ MongoDB ကို Install လုပ်ထားတဲ့ Image နဲ့ Link လုပ်ပေးဖို့လို ပါ တယ်။ Link လုပ်နိုင်ဖို့အတွက် MongoDB Image ကို အရင် Run ပေးရပါမယ်။

```
$ sudo docker run -d ubuntu:mongo
$ sudo docker ps
```

CONTAINER ID	IMAGE	COMMAND	PORTS	NAMES
959db209f4cb	ubuntu: mongo	/usr/bin/mongod	27017->27017	silly_archimedes

ရရှိလာတဲ့ Container Name ကိုသုံးပြီး --link Options နဲ့ ချိတ်ဆက်ပေးနိုင်ပါတယ်။

```
$ sudo docker run -d -p 80:3000 --link silly_archimedes ubuntu:node
```

နမူနာမှာ ubuntu:node Image ကို Run တဲ့အခါ --link Option နဲ့ စောင့်ဆောင်ရွက် Run ထားတဲ့ ubuntu: mongo ကို ချိတ်ဆက်ထားပါတယ်။ ဒီနည်းနဲ့ ubuntu:node ထဲက Issue Tracking API က သီးခြား Container ထဲမှာ Run နေတဲ့ MongoDB ကို အသုံးပြု အလုပ်လုပ်ပေးသွားမှာပဲဖြစ်ပါတယ်။

## Conclusion

Production Server Environment တစ်ခုတည်ဆောက်ရာမှာ Challenge (၃) ချိနိုင်တယ်လို့ အစိုင်းမှာ ဖော်ပြုခဲ့ပါ တယ်။ ဒီပြဿနာတွေကို Docker အကူအညီနဲ့ အောက်ပါအတိုင်း ဖြေရှင်းနိုင်ပါတယ်။

1. Software Project အတွက် Development Environment တည်ဆောက်ဖို့အတွက် လိုအပ်တဲ့ Package စာရင်းကို Dockerfile တစ်ခုအနေနဲ့ ကြိုးတင်သတ်မှတ်ထားသင့်ပါတယ်။
2. ရရှိလာတဲ့ Dockerfile ကို သုံးပြီး Image တစ်ခု Build လုပ်ထားနိုင်ပါတယ်။ ဆောင်ရွက်ရမယ့် Development လုပ်ငန်းတွေကို အဲဒီ Image ပေါ်မှာ ဆက်လက်ဆောင်ရွက်ရမှာ ဖြစ်ပါတယ်။
3. လိုအပ်လို့ Package တွေ Upgrade ပြုလုပ်ရမယ် (သို့မဟုတ်) အသစ်တည့်သွင်းရမယ်ဆိုရင်လည်း Manual မထည့်သွင်းပဲ Dockerfile ထဲမှာသာ ဖြည့်စွက်ပြင်ဆင်သင့်ပါတယ်။ ဒီနည်းနဲ့ လက်ရှိ Software Project အတွက် လိုအပ်တဲ့ Package စာရင်းက Dockerfile တစ်ခုအနေနဲ့ အမြဲအသင့်ရှိနေမှာ ဖြစ်ပါတယ်။

4. Development လုပ်ငန်းပြီးလို့ Production Environment တည်ဆောက်ဖို့လိုတဲ့အခါ၊ အဲဒီ Dockerfile ကို Server ပေါ်မှာ Build လုပ်ခြင်းအားဖြင့် Software Project အတွက် လိုအပ်တဲ့ Production Environment ကို တစ်ဆင့်တည်းနဲ့ အဂျယ်တစ်ကျရရှိသွားမှာဖြစ်ပါတယ်။
5. Development ရော Production ကပါ၊ Dockerfile တစ်ခုတည်းပေါ်မှာ အခြေခံထားတဲ့အတွက် Environment မတူတဲ့ပြဿနာ ဖြေလည်သွားမှာဖြစ်ပါတယ်။
6. Development Machine မှာပြုလုပ်တဲ့ Package အပြောင်းအလဲရှိခဲ့ရင်လည်း ရရှိလာတဲ့ Update Dockerfile ကို Production Server မှာ ထပ်မံ Build လုပ်ပေးခြင်းအားဖြင့် Production Environment လည်း လိုအပ် သလို လိုက်လံပြောင်းလဲသွားမှာပဲ ဖြစ်ပါတယ်။

SOA နည်းစနစ်နဲ့ ခွဲခြားတည်ဆောက်ထားတဲ့ Service တွေကိုလည်း၊ ဘယ်လို Environment ပေါ်မှာ အခြေခံ အလုပ်လုပ် သည်ဖြစ်စေ၊ လိုအပ်တဲ့ Package တွေ တစ်ခါတည်းပါဝင်တဲ့ Docker Image များအဖြစ် ဖန်တီးထားခြင်းအားဖြင့် Server တစ်ခုတည်းပေါ်မှာ Environment မတူတဲ့ Service တွေကို အတူတစ်ကွ ပူးတဲ့အလုပ်လုပ်စေနိုင် မှာပဲဖြစ်ပါတယ်။ ဥပမာ – Service တစ်ခုက NodeJS 0.12.2 ပေါ်မှာ အခြေခံအလုပ်လုပ်တယ်ဆိုရင် သူရဲ့ Docker Image ထဲမှာ NodeJS 0.12.2 ကို ထည့်သွင်းထားနိုင်ပါတယ်။ နောက်တစ်ခုက NodeJS 1.0 လိုအပ်တယ်ဆိုရင်လည်း သူရဲ့ Docker Image ထဲမှာ NodeJS 1.0 ကို ထည့်သွင်းထားနိုင်ပါတယ်။ Image နှစ်ခုဟာ သီးခြား Container တွေထဲ မှာ သီးခြားအလုပ်လုပ်ကြတာဖြစ်လို့ ဘာပြဿနာမှ ရှိမှာမဟုတ်ပဲ Server တစ်ခုတည်းပေါ်မှာ ပူးတဲ့ အလုပ်လုပ်နိုင်သွားမှာပဲ ဖြစ်ပါတယ်။

ဒီလိုအားသာချက်တွေကြောင့်ပဲ Docker ဟာ အခုနောက်ပိုင်းမှာ Developer တွေအကြား အထူးထင်ရှားလာခြင်းဖြစ်ပါတယ်။ Docker ရဲအောင်မြင်မှုကြောင့်၊ RedHat, VMWare, Ubuntu, Microsoft စတဲ့ လုပ်ငန်းတွေလည်းပဲ Docker ကဲသို့ LXC အခြေပြု Container နည်းပညာတွေကို လိုက်လံတိတိုင်လာကြပြီဖြစ်တဲ့အတွက် အနာဂတ်မှာ အခုထက်ပိုမို ဖွံ့ဖြိုးတိုးတက်လာပြီး ပိုမိုအသုံးဝင်လာမြီးမယ့် နည်းပညာတစ်ခုပဲဖြစ်ပါတယ်။

ရုတ်တရက်နာမည်ရသွားတဲ့ App တစ်ခုဟာ နေ့ချင်းညာချင်း  
အသုံးပြုသူ ဆယ်ဆလည်းတိုးသွားနိုင်တယ်။ အဆတစ်ရာလည်း  
တိုးသွားနိုင်တယ်။ ဒီလိုအခြေအနေ ကြံးလာခဲ့ရင် လက်ခံအလုပ်လုပ်  
ပေးနိုင်တဲ့ Server Architecture ကို ကြိုတင်  
ပြင်ဆင်ထားရပါမယ်။

### **Professional Web Developer Course**

HTML5, PHP/MySQL, jQuery/Ajax, Mobile Web စသည့်  
Professional Web Developer တစ်ဦး သိရှိထားသင့်သည့်  
နည်းပညာများကို စုစည်းသင်ကြားခြင်းဖြစ်သည်။  
ဆက်သွယ်ရန် - (၁၉) ၇၃၀ ၆၅၉ ၆၂

## အခန်း(၁၈) – Load Balancing with Nginx

ဘယ်လောက်ပဲ Performance ကောင်းတဲ့ Hardware တွေကို အသုံးပြုထားပါစေ၊ Server တစ်လုံးရဲ့ လက်ခံ အလုပ်လုပ် ပေးနိုင်တဲ့ ပမာဏကတော့ အကန်အသတ် ရှိကြတာချည်းပါဝဲ။ Host လုပ်ထားတဲ့ Software Application (သို့မဟုတ်) Service ရဲ့ အနေအထားပေါ်မှုတည်ပြီး တစ်စူးနှင့်ကို Request ပေါင်း ထောင်းလည်း ဖြစ်နိုင်ပါတယ်။ သောင်း ကဏ္ဍားလည်း ဖြစ်နိုင်ပါတယ်။ ကနေ့ခေတ်မှာ ရုတ်တရက် နာမည်ရသွားတဲ့ App တစ်ခုဟာ နေချင်းသွေးပေါ်မှုတဲ့ အသုံးပြုသူ သန်းချိုရရှိသွားနိုင်ပါတယ်။ ဒီလိုအခြေအနေမျိုး ကြိုလာတဲ့အခါ Server က လက်ခံအလုပ်လုပ်ပေးနိုင်တဲ့ ပမာဏကို ကျော် နေတဲ့အတွက် အလုပ်မလုပ်နိုင်တော့ဘူးဆိုတာမျိုး မ ဖြစ်ရလေအောင် App ကို တစ်လုံးထက်ပိုတဲ့ Server တွေမှာခွဲထားပြီး မျှသုံးပေးဖို့လိုအပ်လာမှာဖြစ်ပါတယ်။ ဒီ အခန်းမှာ၊ အဲဒီလို တစ်လုံးထက်ပိုတဲ့ Server တွေကနေ မျှသုံးပြီး အလုပ်လုပ် ပေးနိုင်တဲ့ Load Balance နည်းပညာတစ်ခုဖြစ်တဲ့ Nginx အကြောင်းကို ဖော်ပြပေးသွားမှာပဲဖြစ်ပါတယ်။



Nginx ဟာ လက်ရှိမှာ Apache ပြီးရင် ဒုတိယမြောက် လူသုံးအများဆုံး Web Server ဖြစ်ပါတယ်။ Engine X လို့ အသုံးပြုတွက်ရပါတယ်။ ရရှားလူမျိုး ပညာရှင် တစ်ဦးဖြစ်တဲ့ Igor Sysoev က ၂၀၀၄ ခုနှစ်မှာ စတင်တိတွင်ခဲ့ခြင်းဖြစ်ပြီး Open Source နည်းပညာ တစ်ခုဖြစ်ပါတယ်။ Resource လိုအပ်ချက် နည်းနည်းနဲ့ စွမ်းဆောင်ရည် ကောင်းမွန်မှုပိုင်းကို အမိကရည်ရွယ် ဖန်တီးထားတဲ့ Web Server ဖြစ်တဲ့အတွက် သူကို Web Server အဖြစ်သာမက Reverse Proxy, Load Balancer, Cache Server စတဲ့ လုပ်ငန်းတွေအတွက်ပါ အသုံးပြုကြပါတယ်။

Ubuntu မှာ Nginx ကို အခုလို Install လုပ်နိုင်ပါတယ်။

```
$ sudo apt-get install nginx
```

Source Code (သို့မဟုတ်) Windows အတွက် Nginx Installer ကိုတော့ အောက်ပါလိပ်စာမှာ ရယူနိုင်ပါတယ်။

<http://nginx.org/en/download.html>

Install လုပ်ပြီးတဲ့အခါ Nginx Web Server ကို အခုလို Run နိုင်ပါတယ်။

```
$ sudo nginx
```

Ubuntu မှာ apt-get နဲ့ Install လုပ်ထားတာဆိုရင် Ubuntu က အလိုအလျောက် Run ပေးထားမှာဖြစ်လို

ကိုယ်တိုင် Run ပေးစရာမလိပါဘူး။ Nginx ကို ရပ်စေလိုဂင်တော့ `nginx -s stop` (သိမဟုတ်) `nginx -s quit` ကို အသုံးပြုနိုင်ပါတယ်။ `stop` က Nginx ကို ချက်ခြင်းရပ်လိုက်မှာဖြစ်ပြီး၊ `quit` Signal ကတော့ လက်ရှိအလုပ်လုပ်နေတဲ့ Request တွေ ပြီးစီးအောင် စောင့်ပြီးတော့မှ ရပ်ပေးမှာဖြစ်ပါတယ်။ ဥပမာ –

```
$ sudo nginx -s quit
```

Nginx Server Configuration ဖိုင်တွေကို ပြင်ဆင်ပြီးနောက် Nginx ကို Restart မလုပ်ပဲ Configuration တွေကိုပဲ Reload လုပ်စေလိုဂင်တော့ `reload` Signal ကို အသုံးပြုရပါတယ်။

```
$ sudo nginx -s reload
```

## 18.1 – Nginx Configuration

Nginx Configuration တွေကို Simple Directive နဲ့ Block Directive ဆိုပြီး ပုံစံနှစ်မျိုးနဲ့ ရေးသားနိုင်ပါတယ်။ Simple Directive တွေကို `name` parameter Format နဲ့ရေးရပြီး လိုင်းအဆုံးမှာ Semi-colon နဲ့ ပိတ်ပေးရပါတယ်။ Block Directive တွေကိုတော့ `name` နောက်က တွန်းကွင်း အဖွင့်အပိတ် Block နဲ့ ရေးပေးရပါတယ်။ Block ထဲမှာ လိုအပ်တဲ့ Instruction တွေကို စုစုပေါင်းထည့်သွင်းနိုင်ပါတယ်။ `#` Sign နဲ့စတဲ့လိုင်းတွေကတော့ Comment တွေဖြစ်ပါတယ်။

ဒီအခန်းမှာ ဖော်ပြထားတဲ့ Configuration နှမူနာတွေဟာ တိုက်ရှိကဲးယူ စမ်းသပ်ဖို့မဟုတ်ပါ။ Nginx ရဲ့ သဘောသဘာဝကို ပေါ်လွင်အောင် Simplify လုပ်ထားတဲ့ နှမူနာများဖြစ်ပါတယ်။ ဒါကြောင့် သဘောသဘာဝကိုသာ နားလည်အောင်လေ့လာပြီး လက်တွေ့ကိုတော့ ကိုယ်တိုင်ဆက်လက် လေ့လာ သွားရမှာ ဖြစ်ပါတယ်။

Directive တွေထဲမှာ အခိုက်သတိပြုသင့်တာတွေကတော့ `http`, `server` နဲ့ `location` တို့ပဲဖြစ်ပါတယ်။ ပုံမှန် အားဖြင့်၊ ဒီ Directive (၃) ခုရဲ့ ဖွဲ့စည်းပုံက အခုလိုရှိတက်ပါတယ်။

### Config

```
1. http {
2.   gzip on;
3.   server {
4.     listen 80
5.     server_name example.com
6.     location / {
7.       # Configuration
8.     }
9.   }
10.
```

```
11. server {
12.     listen 80
13.     server_name example.net
14.     location / {
15.         # Configuration
16.     }
17.
18.     location /images {
19.         # Configuration
20.     }
21. }
22. }
```

နှမူနာကိုလေးလာကြည့်ရင် http က Main Block ဖြစ်တယ်ဆိုတာကို တွေ့ရမှာဖြစ်ပါတယ်။ http Block ထဲမှာ အခြား Setting တွေနဲ့အတူ server Block တွေရှိနိုင်ပါတယ်။ server Block တွေတဲ့မှာလည်း အခြား Setting တွေနဲ့အတူ location Block တွေရှိတက်ပါတယ်။ server Block တွေကို Virtual Host တွေ သတ်မှတ်ဖို့အတွက် အသုံးပြုနိုင်ပြီး location Block တွေကိုတော့ Routing အတွက် အသုံးပြုနိုင်ပါတယ်။ Nginx Server တစ်ခုလုံးနဲ့ သတ်ဆိုင်တဲ့ Setting တွေကို http Block ထဲမှာ ရေးရပြီး သက်ဆိုင်ရာ Virtual Web Server အတွက် Setting တွေကိုတော့ server Block ထဲမှာ ရေးပေးရပါတယ်။ အလားတူပဲ location တစ်ခုနဲ့သာ သက်ဆိုင်စေလိုတဲ့ Setting တွေကိုတော့ location Block ထဲမှာ ရေးပေးရပါတယ်။

Ubuntu Linux မှာဆိုရင် Nginx ရဲ့ Main Configuration တွေဟာ /etc/nginx/nginx.conf ဖိုင်ထဲမှာ တည်ရှိနေမှာ ဖြစ်ပါတယ်။ အဲဒီဖိုင်ကို Text Editor နဲ့ ဖွင့်ကြည့်ရင် အခုလိုတွေရနိုင်ပါတယ်။

## Config

```
1. user www-data;
2. worker_processes 4;
3. pid /run/nginx.pid;
4.
5. events {
6.     worker_connections 768;
7.     # multi_accept on;
8. }
9.
10. http {
11.
12.     ##
13.     # Basic Settings
14.     ##
15.
16.     sendfile on;
17.     tcp_nopush on;
18.     tcp_nodelay on;
19.     keepalive_timeout 65;
20.     types_hash_max_size 2048;
21.
22.     include /etc/nginx/mime.types;
23.     default_type application/octet-stream;
24.
25.     access log /var/log/nginx/access.log;
```

```

26. error_log /var/log/nginx/error.log;
27.
28. gzip on;
29. gzip_disable "msie6";
30.
31. include /etc/nginx/conf.d/*.conf;
32. include /etc/nginx/sites-enabled/*;
33. }
```

http Block ရဲ့ အပြင်ဖက်မှာ မဖြစ်မနေပါဝင်ဖို့လိုအပ်တဲ့ အခြေခံ Setting တွေ ရှိပါတယ်။ http Block ထဲက နမူနာလိုင်းနံပါတ် (၃၂) မှာတော့ include Directive နဲ့ /etc/nginx/sites-enabled Directory ထဲက Configuration ဖိုင်တွေကို ချိတ်ဆက်ပေးထားတာကို တွေ့ရမှာဖြစ်ပါတယ်။ အမိပါယ်ကတော့ server Block တွေကို Main Configuration ဖိုင်ထဲမှာ ရောမရေးပဲ sites-enabled Directory ထဲမှာ၊ သီးခြားဖိုင်များအဖြစ် ခွဲရေးထား ခြင်းဖြစ်ပါတယ်။ sites-enabled Directory ၏ default ဖိုင်ကို ဖွင့်ကြည့်ရင်အခါလို တွေ့ရ နိုင်ပါတယ်။

## Config

```

1. server {
2.   listen 80 default_server;
3.   listen [::]:80 default_server ipv6only=on;
4.
5.   root /usr/share/nginx/html;
6.   index index.html index.htm;
7.
8.   server_name localhost;
9.
10.  location / {
11.    try_files $uri $uri/ =404;
12.  }
13. }
```

နမူနာမှာ Comment တွေဖော်ပြုးဖော်ပြထားပါတယ်။ စာဖတ်သူဖွင့်ကြည့်တဲ့အခါမှာတော့ Comment တွေ လည်း ပါဝင်နေနိုင်ပါတယ်။ server Block ထဲက လိုင်းနံပါတ် (၅) မှာ root Directive ကိုသုံးပြီး /usr/share/nginx/html Directory ကို Document Root အဖြစ် သတ်မှတ်ထားတာကို တွေ့ရမှာဖြစ်ပါတယ်။ ဒါ ကြောင့် HTML နဲ့ အခြား Static Document တွေကို /user/share/nginx/html Directory ထဲမှာ ထားပေးရမှာပါ။ လိုင်းနံပါတ် (၆) မှာ index Directive နဲ့ index.html နဲ့ index.htm တို့ကို Directory Index အနေနဲ့ အသုံးပြုပေးဖို့ သတ်မှတ်ထားပါတယ်။

Document Root ကို Location အလိုက် ခွဲခြားသတ်မှတ်လိုရင် အခါလိုသတ်မှတ်နိုင်ပါတယ်။

### Config

```

1. location /static {
2.   root /usr/share/nginx/html
3. }
4.
5. location /images {
6.   root /usr/share/nginx/html/images
7. }
```

URI မှာ /static စတဲ့ Request တွေအတွက် အသုံးပြုရမယ့် Document Root ကတစ်ခုဖြစ်ပြီး URI မှာ /images နဲ့ စတဲ့ Request တွေအတွက် Document Root တစ်ခုကို သီးခြားသတ်မှတ်ထားခြင်းဖြစ်ပါတယ်။

default Configuration ရဲ့ လိုင်းနံပါတ် (၁၁) မှာသတ်မှတ်ထားတဲ့ try\_files \$uri \$uri/ =404 ရဲ့ အဓိပ္ပာယ်ကတော့ Document Root ထဲမှာ URI Path နဲ့ ကိုက်ညီတဲ့ဖိုင် (သို့မဟုတ်) Directory ရှိရင် Response ပြန်ပေးပြီး မရှိနေရင်တော့ 404 ကို Response ပြန်ပေးဖို့ သတ်မှတ်ထားခြင်းဖြစ်ပါတယ်။ တန်ည်းအားဖြင့် Nginx ကို Static File Server တစ်ခုအနေနဲ့ အလုပ်လုပ်စေနိုင်ပါတယ်။

လိုင်းနံပါတ် (၈) မှာတော့ server\_name ကို localhost လို့ သတ်မှတ်ထားပါတယ်။ အကယ်၍ Server မှာ Domain Name ရှိတယ်ဆိုရင် localhost အစား သတ်ဆိုင်ရာ Domain Name ကို သတ်မှတ်ပေးရမှာ ဖြစ်ပါတယ်။ အကယ်၍ Domain Name နှစ်ခုရှိတယ်ဆိုရင် သက်ဆိုင်ရာ Domain Name အတွက် Virtual Host တွေ ကို အခုပ္ပါ ခွဲခြားသတ်မှတ် ပေးနိုင်ပါတယ်။

### Config

```

1. server {
2.   server_name example.com;
3.   root /var/www/example.com;
4.   index index.html index.htm;
5.   location / {
6.     try_files $uri $uri/ =404
7.   }
8. }
9.
10. server {
11.   server_name example.net;
12.   root /var/www/example.net;
13.   index index.html index.htm;
14.   location / {
15.     try_files $uri $uri/ =404
16.   }
17. }
```

ဒီနည်းနဲ့ example.com အတွက် Document Root တစ်ခုနဲ့ example.net အတွက် Document Root တစ်ခု၊ သီးခြားစီ ခွဲခြားသတ်မှတ်ထားနိုင်ပါတယ်။

ကျွန်တော်တိ အခန်း (၁၃) မှာ ExpressJS နဲ့ API Server တည်ဆောက်ခဲ့စဉ်က Cross Origin Resource Sharing နဲ့ပက်သက်တဲ့ Header တွေ သတ်မှတ်ခဲ့ရတာကို မှတ်မိမိုးမှာပါ။ Nginx မှာလည်း အလားတူ Default Header တွေ သတ်မှတ်လိုတယ်ဆိုရင် add\_header Directive ကို အသုံးပြုနိုင်ပါတယ်။

### Config

```

1. location / {
2.   add_header 'Access-Control-Allow-Origin' '*';
3.   add_header 'Access-Control-Allow-Credentials' 'true';
4.
5.   try_files $uri $uri/ =404
6. }
```

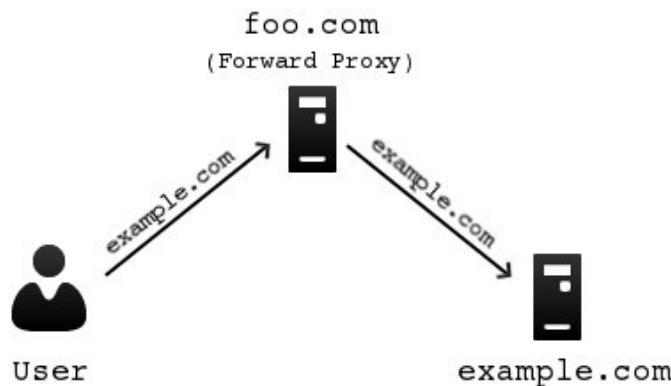
နဲ့မူနာမှာပေးထားသလို location Block ထဲမှာရေးရင်တော့ သက်ဆိုင်ရာ Location အတွက်ပဲ အလုပ်လုပ်မှာပါ။ အကယ်၍ Web Server တစ်ခုလုံးအတွက် အလုပ်လုပ်စေလိုရင်တော့ server Block ထဲမှာ ရေးသားပေးသင့်ပါတယ်။ Nginx Directive တစ်ခုချင်းစီရဲ့ အဓိပ္ပာယ်နဲ့ အသုံးပြုပုံတွေကို အောက်ပါလိပ်စာမှာ ဆက်လက်လေ့လာနိုင်ပါတယ်။

[http://nginx.org/en/docs/http/ngx\\_http\\_core\\_module.html](http://nginx.org/en/docs/http/ngx_http_core_module.html)

## 18.2 – Proxy

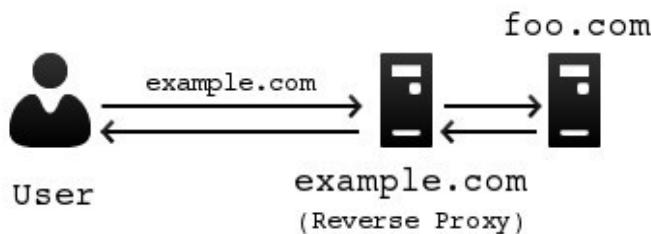
Nginx ရဲ့ လုပ်ဆောင်ချက်တွေထဲက အရေးပါတဲ့ လုပ်ဆောင်ချက်ကတော့ Proxy ဖြစ်ပါတယ်။ Proxy Server တစ်ခုရဲ့ အခြေခံ အလုပ်လုပ်ပုံကတော့၊ အသုံးပြုသူ User တွေပေးပို့တဲ့ Request ကို ကြားထဲကနေလက်ခံပြီး App Server ထံ လက်ဆင့်ကမ်းပေးပို့ခြင်းပဲဖြစ်ပါတယ်။

Proxy မှာ Forward Proxy နဲ့ Reverse Proxy ဆိုပြီး နှစ်မျိုးရှိပါတယ်။ Forward Proxy ဆိုတာ အသုံးပြုသူ "Client ကိုယ်စား" ကြားကနေ တစ်ဆင့်ဆောင်ရွက်ပေးတဲ့ Proxy ဖြစ်ပါတယ်။ ဥပမာ - Client က example.com ကို သွားချင် ပေမယ့် တိုက်ရှိက်မသွားပဲ foo.com ကို တစ်ဆင့်သွားမယ်၊ foo.com က Client ရဲ့ Request ကို example.com ထံ လက်ဆင့်ကမ်းပေးမယ်ဆိုရင် foo.com ကို Forward Proxy လိုအပ်ပါတယ်။ တစ်ကယ်အလုပ်လုပ်တာက example.com ဖြစ်ပြီး foo.com က ကြားခံတစ်ခုသာဖြစ်ပါတယ်။



ပုံ (၁.၁) - Forward Proxy

Reverse Proxy ဆိုတာကတော့ Request ကို လက်ခံအလုပ်လုပ်မယ့် "Server ကိုယ်စား" ကြားကနေ ကြားဖြတ်လက်ခံ ပေးတဲ့ Proxy အမျိုးအစားဖြစ်ပါတယ်။ ဥပမာ - Client က example.com ကို Request လုပ်ပေမယ့် example.com လက်ခံရရှိတဲ့ Request ကို foo.com ထံ လက်ဆင့်ကမ်းပေးလိုက်တယ်ဆိုရင် example.com ဟာ Reverse Proxy ဖြစ် ပါတယ်။ တစ်ကယ်အလုပ်လုပ်တဲ့ App Server က foo.com ဖြစ်ပြီး example.com က ကြားခံတစ်ခုသာ ဖြစ်ပါတယ်။ ထူးခြားတာက၊ User အနေနဲ့ foo.com ရှိနေတာကို သိစရာမလိုပဲ example.com ကိုသာ App Server အမှတ်နဲ့ ဆက်သွယ်အသုံးပြုရမှာဖြစ်ပါတယ်။ ကျွန်ုတ်တို့ ဒီနေရာမှာ ဖော်ပြလိုတာ ကတော့ Reverse Proxy အကြောင်းပဲဖြစ်ပါ တယ်။



ပုံ (၁.၂) - Reverse Proxy

ဒီလို Proxy Server တစ်ခုနဲ့ ကြားကနေ Request တွေကို လက်ခံပေးခြင်းအားဖြင့် ရရှိလာတဲ့ အကျိုးရလဒ် ကတော့ -

- ၁.) လုပ်ခြင်းနဲ့ပက်သက်တဲ့ ကိစ္စတွေကို App Server မှာဝန်မပိုပေး ကြားဖြတ်ဆောင်ရွက်နိုင်ပါတယ်။ ဥပမာ - Access လုပ်ခွင့်မရှိတဲ့ Client တွေက ပြုလုပ်လာတဲ့ Request တွေကို App Server ထံရောက်စရာမလိုပဲ ကြားကနေ Proxy Server က ဖြတ်တောက်လိုက်နိုင်ပါတယ်။

၂။) စွမ်းဆောင်ရည် ပိုမိုကောင်းမွန်စေဖို့အတွက် Proxy Cache အဖြစ် ကြားခံဆောင်ရွက်ပေးနိုင်ပါတယ်။ အကယ်၍ Client လိုချင်တဲ့ Resource က Proxy Cache မှာရှိနေတယ်ဆိုရင် App Server ထံကနေ ရယူနေစရာ မလိုပဲ Proxy Cache ကနေ ပြန်ပေးနိုင်ပါတယ်။ ဒီနည်းနဲ့ ထပ်ခါထပ်ဆဲ ရယူဖို့လိုတဲ့တူညီတဲ့ Content တွေ အတွက်နဲ့ App Server ပေါ်ဝန်ပိုစရာမလိုတော့ပါဘူး။

၃။) App Server အများအပြားရှိဖို့တဲ့က လက်ရှိအချိန်မှာ Request ကို လက်ခံဆောင်ရွက်ဖို့ အဆင်အမြေဆုံးနဲ့ အသင့် တော်ဆုံး Server ထံ လက်ဆင့်ကမ်းပေးပို့ခြင်းအားဖြင့်၊ App Server အများအပြားကို မျှသုံးပေးတဲ့ Load Balance လုပ်ဆောင်ချက်ကို ရရှိနိုင်မှာ ဖြစ်ပါတယ်။

Nginx ရဲ့ Proxy လုပ်ဆောင်ချက်ကို အသုံးပြုဖို့အတွက် proxy\_pass Directive ကို အသုံးပြုနိုင်ပါတယ်။ ဥပမာ -

### Config

```
1. location /api {
2.   proxy_pass http://foo.com:3000/api;
3. }
```

နမူနာအရ /api URI ကို လက်ခံရှိဖို့အခါ Nginx က foo.com:3000/api ကို အလိုအလျောက် လက်ဆင့်ကမ်းပေးသွားမှာဖြစ်ပါတယ်။ အကယ်၍ URI က /api/issues ဖြစ်ခဲ့မယ်ဆိုရင် လက်ဆင့်ကမ်းပေးသွားမယ့် URL က http://foo.com:3000/api/issues ဖြစ်မှာဖြစ်ပါတယ်။

App Server ထံ Request ကို Forward မလုပ်ခင်၊ တစ်ချို့လုပ်ခြင်း စစ်ဆေးမှုတွေ ပြုလုပ်လိုတယ်ဆိုရင် အခုလို ဆောင်ရွက်နိုင်ပါတယ်။

### Config

```
1. if ($http_user_agent ~* (wget|crawler) ) {
2.   return 403;
3. }
4.
5. location /api {
6.   deny 192.168.1.2;
7.   proxy_pass http://foo.com:3000/api;
8. }
```

နမူနာအရ User Agent က Search Engine Crawler (သို့မဟုတ်) Wget ပရိုဂရမ ဖြစ်နေမယ်ဆိုရင် 403 Forbidden ကို Response လုပ်ပေးမှာဖြစ်ပြီး IP Address 192.168.1.2 ကနေ Request ပြုလုပ်လာရင်လည်း Deny လုပ်သွားမှာဖြစ်ပါတယ်။ နမူနာအဖြစ်သာ ဖော်ပြထားခြင်းဖြစ်ပြီး လက်တွေမှာ လုပ်ခြင်းအရ စီစစ်လိုတဲ့ IP Address တွေနဲ့ User Agent တွေကို Black List စာရင်းတစ်ခုလုပ်ထားပြီး အဲဒီစာရင်းနဲ့ တိုက်ဆိုင်စီစစ်သွားရမှာ ဖြစ်ပါတယ်။

Request တွေကို လက်ဆင့်ကမ်းပေးပို့တဲ့အခါ Request Header ကို အနည်းငယ်သတိထားရပါမယ်။ ဥပမာ – example.com မှာ Proxy Server ကို Setup လုပ်ထားပြီး API Server ကို foo.com မှာ Setup လုပ်ထားတယ်ဆို ကြပါစို့။ User က example.com ကို Request ပြုလုပ်လိုက်တဲ့အခါ example.com Proxy Server က foo.com ကို အလိုအလျောက် Request ကို လက်ဆင့်ကမ်းပေးသွားမှာဖြစ်ပါတယ်။ ဒီတော့ စဉ်းစားကြည့်ရင် foo.com ကို အမှန်တစ်ကယ် Request ပြုလုပ်သူက User မဟုတ်တော့ပဲ example.com ဖြစ်နေမှာဖြစ်ပါတယ်။ လိုအပ်ချက်အရ Request ပြုလုပ်သူ စာရင်းကို မှတ်တမ်းလုပ်ထားချင်ရင် မရတော့ပါဘူး။ foo.com ကို Request ပြုလုပ်သူဟာ အမြတ်စွဲ example.com ပဲဖြစ်နေမှာဖြစ်ပါတယ်။ ဒီပြဿနာမျိုးကို ဖြေရှင်းနိုင်ဖို့ အတွက် Proxy Request နဲ့အတွက် Request Header ကို ပြင်ဆင်ပေးပို့ဖို့ လိုတက်ပါတယ်။ ပြင်ဆင်ပေးဖို့ လိုလေ့ ရှိတဲ့ Header အချို့ကို ဖော်ပြလိုက်ပါတယ်။

## Config

```

1. proxy_set_header HOST $host;
2. proxy_set_header X-Forwarded-Proto $scheme;
3. proxy_set_header X-Real-IP $remote_addr;
4. proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
5.
6. location /api {
7.   proxy_pass http://foo.com/api;
8. }
```

Nginx က \$host Variable ထဲမှာ မူလ Request ပြုလုပ်ခဲ့တဲ့ Host အမည်ကို သိမ်းထားပေးပါတယ်။ အထက်က နမူနာအရ \$host Variable ထဲကတန်ဖိုးက example.com ဖြစ်မှာပါ။ Proxy က Request ကို လက်ဆင့်ကမ်း ပေးပို့တဲ့အခါ HOST Header အတွက် proxy\_pass မှာပေးလိုက်တဲ့ Host Name ကို အသုံးပြုပေးမှာပါ။ ဒါကြောင့် HOST Header တန်ဖိုးက Client ပေးပို့စဉ်မှာ example.com ဖြစ်ပေမယ့် Proxy က foo.com လိုပြောင်းပြီး လက်ဆင့်ကမ်းသွားမှာဖြစ်ပါတယ်။ ဒါကြောင့် HOST Header တန်ဖိုးကို မပြောင်းပဲ မူလအတိုင်းပဲ လက်ဆင့်ကမ်းစေလိုတဲ့အခါ နမူနာ လိုင်းနံပါတ် (၁) မှာပေးထားသလို proxy\_set\_header Directive ကိုသုံးပြီး HOST \$host လို့ သတ်မှတ်ပေးနိုင်ပါတယ်။ လိုင်းနံပါတ် (၂) မှာ သတ်မှတ်ထားတဲ့ X-Forwarded-Proto Header ကတော့ Standard HTTP Header မဟုတ်ပဲ ကိုကားနိုင်ဖို့အတွက် ကိုယ့်သဘောနဲ့ကိုယ်ထည့်သွေးပေးလိုက်တဲ့ Custom Header ဖြစ်ပါတယ်။ Nginx က \$scheme Variable ထဲမှာ မူလပြုလုပ်ခဲ့တဲ့ Protocol (HTTP သို့ HTTPS) ကို သိမ်းထားပေးပါတယ်။ Proxy Server နဲ့ App Server တို့ဆက်သွယ်တဲ့အခါ သုံးတဲ့ Protocol နဲ့ Client က အမှန် တစ်ကယ်ဆက်သွယ်စဉ်သုံးတဲ့ Protocol ကွာနိုင်တဲ့အတွက် ဒီလိုတဲ့ဖော်ပေးဖို့ သင့်ခြင်း ဖြစ်ပါတယ်။

လိုင်းနံပါတ် (၃) မှာလည်း X-Real-IP Custom Header ကိုသုံးပြီး Client ရဲ့ IP Address ကို App Server ထဲ တွဲဖက်ပေးပို့ထားပါတယ်။ အကယ်၍များ Proxy က တစ်ဆင့်တည်းမဟုတ်ပဲ၊ နှစ်ဆင့်သုံးဆင့်ဆင့်ပြီး Configure လုပ်ထားရင် Nginx က Proxy Server အဆင့်ဆင့်ရဲ့ IP Address တွေကို \$proxy\_add\_x\_forwarded\_for Variable ထဲမှာ သိမ်းထားပေးပါတယ်။ ဒါကြောင့် လိုင်းနံပါတ် (၄) မှာ အဲဒီ Proxy Server တွေရဲ့ IP စာရင်းကိုလည်း တွဲဖက်ပေးပို့ထားပါသေးတယ်။

ဒီနည်းနဲ့ Request ကို အမှန်တစ်ကယ်တာဝန်ယူဆောင်ရွက်မယ့် App Server သိသုတဲ့ ဖြည့်စွက် အချက်အလက်တွေကို Proxy Server က Header ထဲမှာ ထည့်သွင်းပေးပို့နိုင်မှာဖြစ်ပါတယ်။

## 18.2 – Load Balancing

Nginx Proxy ကို App Server အများအပြားရှိတဲ့ထဲက လက်ရှိမှာ လက်ခံဆောင်ရွက်ဖို့ အဆင်အပြေဆုံး Server ကို အလိုအလျောက်ရွေးချယ်ပြီး Request ကို လက်ဆင့်ကမ်းပေးသွားအောင် စီစဉ်ထားနိုင်ပါတယ်။ upstream Directive ကို အသုံးပြုရပါတယ်။

### Config

```

1. upstream api_servers {
2.   server api.server1.com;
3.   server api.server2.com;
4.   server api.server3.com;
5. }
6.
7. server {
8.   # Other server configs
9.   location /api {
10.     proxy_pass http://api_servers;
11.   }
12. }
```

နမူနာကိုလေ့လာကြည့်ရင် upstream Directive ကိုသုံးပြီး api\_servers အမည်နဲ့ Block တစ်ခုသတ်မှတ်ထားပါတယ်။ Block ထဲမှာ Server စာရင်းကို တန်းစီပြီးထည့်သွင်းပေးထားပါတယ်။ လိုင်းနံပါတ် (၁၀) ကိုလေ့လာကြည့်ရင် proxy\_pass အတွက် Upstream Server စာရင်းအဖြစ် ကြိုတင်သတ်မှတ်ထားတဲ့ api\_servers ကို ပေးထား တာကိုတွေ့ရမှာဖြစ်ပါတယ်။ ဒီလိုပေးလိုက်တဲ့အခါ Nginx က Upstream Server တွေထဲက လက်ရှိ Request ကို လက်ခံဆောင်ရွက်ဖို့ အဆင်အပြေဆုံး Server ကို အလိုအလျောက် ရွေးချယ်ပြီး တော့မှ Request ကို လက်ဆင့်ကမ်းပေး သွားမှာပဲ ဖြစ်ပါတယ်။

Load Balancing Algorithm အမျိုးမျိုးရှိပါတယ်။ Nginx က Default အနေနဲ့ အသုံးပြုပေးမယ့် Algorithm ကတော့ Round Robin Algorithm ပဲဖြစ်ပါတယ်။

**Round Robin** - Round Robin Algorithm ရဲ့ သဘောသဘာဝက ရှင်းရှင်းလေးပါ။ Request တွေကို ပေးထားတဲ့ Server စာရင်းထဲက Server တစ်ခုပြီးတစ်ခု အလှည့်ကျ ပေးပို့ပေးသွားမှာဖြစ်ပါတယ်။ ဥပမာ - Request နှစ်ခုကို လက်ခံရရှိတဲ့အခါ ပထမတစ်ခုကို api.server1.com ထံပေးပို့သွားမှာဖြစ်ပြီး ဒုတိယတစ်ခုကို api.server2.com ထံ ပေးပို့သွားမှာပဲ ဖြစ်ပါတယ်။

**Least Connection** - Nginx က အသုံးပြုပေးနိုင်တဲ့ နောက်ထပ် Load Balancing Algorithm ကတော့ least\_conn ပဲဖြစ်ပါတယ်။ Server တွေထဲလက် လက်ရှိ Active Connection အနည်းဆုံးရှိနေတဲ့ Server ကို ရွေးချယ်ပြီး Request ကို ပေးပို့သွားမှာဖြစ်ပါတယ်။ Round Robin လို ရိုးရိုးအလှည့်ကျ ပေးပို့တာမျိုး မဟုတ်

တော့ Server တွေထဲက လက်ရှိအနည်းဆုံး အလုပ်လုပ်နေရတဲ့ Server ကို ရွေးချယ်ပေးပို့ပေးမယ့် Algorithm ပဲဖြစ်ပါတယ်။

**IP Hash** – နောက် Algorithm တစ်ခုကတော့ ip\_hash ဖြစ်ပါတယ်။ ဒါ Algorithm ကတော့ Client ရဲ့ IP Address ပေါ်မှုတည်ပြီး Server ကို ရွေးချယ်ပေးခြင်းဖြစ်ပါတယ်။ ပုံမှန်အားဖြင့် Client တစ်ခုက Request လုပ်လာတဲ့ အခါ ဘယ် Server ထဲ ရောက်ရှိအလုပ်လုပ်မယ်ဆိုတာ ပုံသေပြောလို့မရပါဘူး။ သင့်တော်ရာ Server ကို Proxy က ရွေးချယ်ပေးပို့သွားမှာဖြစ်တဲ့အတွက် ဖြစ်ပါတယ်။ ip\_hash Algorithm ကတော့ Client ရဲ့ IP ကို မှတ်ထားပြီး တစ်ကြိမ်ပေးပို့ ဖူးတဲ့ Server ထဲကိုသာ နောက်တစ်ကြိမ်ထပ်မံပေးပို့မှာဖြစ်ပါတယ်။

Least Connection Algorithm ကို အသုံးပြုလိုတယ်ဆိုရင် Upstream Setting ကို အခုလို ပြုပြင်နိုင်ပါတယ်။

### Config

```

1. upstream api_servers {
2.
3.   least_conn;
4.
5.   server api.server1.com;
6.   server api.server2.com;
7.   server api.server3.com;
8. }
```

IP Hash ကို အသုံးပြုလိုတယ်ဆိုရင်လည်း အလားတူပဲ သတ်မှတ်နိုင်ပါတယ်။

### Config

```

1. upstream api_servers {
2.
3.   ip_hash;
4.
5.   server api.server1.com;
6.   server api.server2.com;
7.   server api.server3.com;
8. }
```

Server တွေရဲ့ စွမ်းဆောင်ရည် တစ်ခုနဲ့တစ်ခုမတူလို့ တစ်ချို့ Server တွေကို ပိုပြီးအသုံးပြုစေလိုတယ်ဆိုရင် Upstream Setting မှာ Weight ကိုလည်း ထည့်သွင်းသတ်မှတ်နိုင်ပါတယ်။ ဥပမာ –

### Config

```

1. upstream api_servers {
2.   server api.server1.com weight=3;
3.   server api.server2.com;
4.   server api.server3.com;
5. }
```

နမူနာအရ api.server1.com ဟာ တစ်ခြား Server တွေထက် Request တွေကို (၃) ဆပိုပြီး လက်ခံရရှိတော့မှာ ဖြစ်ပါတယ်။ တစ်ခါတစ်ရုံမှာ Server တွေခဲ့ပြီး Load Balance လုပ်နေပေမယ့်၊ အကြောင်းအမျိုးမျိုးကြောင့် Upstream Server တွေထဲက Server တစ်ခုဟာ လက်တစ်လော Request တွေကို လက်ခံဆောင်ရွက်နိုင်ခြင်းမရှိသေးတာမျိုး ဖြစ်တက်ပါတယ်။ ဒီလိုအခြေအနေရှိလာတဲ့အခါ အဆင်ပြေဖော်အတွက် အခုလို သတ်မှတ်ထားနိုင်ပါတယ်။

### Config

```
1. upstream api_servers {
2.   server api.server1.com max_fails=3 fail_timeout=25s;
3.   server api.server2.com;
4.   server api.server3.com;
5. }
```

နမူနာမှာ max\_fails ကို 3 လို သတ်မှတ်ထားတဲ့အတွက် api.server1.com ဟာ Request တွေလက်ခံဆောင်ရွက်ဖို့ (၃) ကြိုင် Fail ဖြစ်ခဲ့တယ်ဆိုရင် fail\_timeout မှာ သတ်မှတ်ထားတဲ့ အချိန်ကာလဖြစ်တဲ့ (၂၅) စွဲဘွဲ့ အဲဒီ Server ကို Nginx Proxy က အသုံးမပြုပဲ၊ အခြား Server တွေကို အစားထိုး အသုံးပြုပေးသွားမှာ ဖြစ်ပါတယ်။

### 18.3 – Proxy Cache

Nginx Proxy ရဲ စွမ်းဆောင်ရည် ပိုမိုကောင်းမွန်ဖော်အတွက် Content တွေကို Cache လုပ်ထားနိုင်ပါတယ်။ Nginx Proxy က တစ်ကြိမ်ရယူထားဖူးတဲ့ Content ကို ခေါ်သိမ်းထားပြီး၊ အလားတူ Request မျိုး ထပ်မံရရှိတဲ့အခါ Upstream Server တွေထဲ လက်ဆင့်ကမ်းမနေတော့ပဲ သူသိမ်းထားတဲ့ Cache Content ကို Response ပြန်ပေးနိုင်တဲ့ လုပ်ဆောင်ချက်ဖြစ်ပါတယ်။ မလိုအပ်ရင် Upstream Server တွေထဲ သွားနေစရာမလိုတော့တဲ့ အတွက် စွမ်းဆောင်ရည် လည်း တိုးတက်လာမှာပဲဖြစ်ပါတယ်။

Cache လုပ်ဆောင်ချက်ကို အသုံးပြုနိုင်ဖို့အတွက်၊ ပထမအဆင့်အနေနဲ့ http Block အတွင်းမှာ အောက်ပါအတိုင်း သတ်မှတ်ပေးဖို့လိုပါတယ်။

### Config

```
1. proxy_cache_path /var/lib/nginx/cache levels=1:2 keys_zone=backcache:8m max_size=50m;
2. proxy_cache_key "$scheme$request_method$host$request_uri$is_args$args";
3. proxy_cache_valid 200 10m;
4. proxy_cache_valid 404 1m;
```

လိုင်းနံပါတ် (၁) မှာ proxy\_cache\_path Directive ကိုအသုံးပြုပြီး Cache Content ကို သိမ်းဆည်းရမယ့် Directory ကို သတ်မှတ်ပေးထားခြင်းဖြစ်ပါတယ်။ Nginx က Cache ကိုသိမ်းတဲ့အခါ Cache Key နဲ့ Cache Data ဆိုပြီး နှစ်ပိုင်းခွဲသိမ်းပါတယ်။ Cache Key အနေနဲ့ Scheme, Request Method, Host Name, Request URI စတဲ့ Request Variable တွေကို အတဲ့လိုက်သုံးဖို့လိုင်းနံပါတ် (၂) မှာ သတ်မှတ်ထားပါတယ်။ ဥပမာ – Request က GET <http://example.com/api> ဆိုရင် Cache Key က <http://example.com/api0> ဖြစ်မှာ ဖြစ်ပါတယ်။ Nginx

က အဲဒီ Key ကို Hash လုပ်ပြီး အသုံးပြုမှုပါ။

လိုင်းနံပါတ် (၁) က levels=1:2 ရဲ့အဓိပ္ပာယ်ကတော့ Cache တွေကို Cache Key Hash ရဲ့ နောက်ဆုံး Character တစ်လုံးကို အမည်အဖြစ်အသုံးပြုထားတဲ့ Directory အတွင်းမှာ Hash ရဲ့ နောက်ဆုံး Character နှစ်လုံးကို အမည်အဖြစ် အသုံးပြုထားတဲ့ Directory အနေနဲ့ စုစည်းသိမ်းဆည်းဖို့ သတ်မှတ်ထားခြင်းဖြစ်ပါတယ်။ ဥပမာ - Key Hash က abcd123ef34abc ဆိုရင် c Directory ထဲမှာ ab Directory နဲ့ Cache Content ကို သိမ်းဆည်းပေးမှာ ဖြစ်ပါတယ်။

keys\_zone=backcache:8m ရဲ့ အဓိပ္ပာယ်ကတော့ Cache Key ကို backcache လိုအပ်တဲ့အမည်နဲ့ သိမ်းဆည်းပြီး Cache Key သိမ်းဆည်းဖို့အတွက် Storage ကို 8 MB ထိ ယူပါလိုသတ်မှတ်ထားခြင်းဖြစ်ပါတယ်။ max\_size=50m ကတော့ Cache Content သိမ်းဆည်းဖို့ အမြင်ဆုံးခွင့်ပြုတဲ့ Storage ပမာဏက 50 MB ဖြစ်တယ် လို့ သတ်မှတ်ထားခြင်းပဲဖြစ်ပါတယ်။

လိုင်းနံပါတ် (၂) နဲ့ (၄) က proxy\_cache\_valid ရဲ့ အဓိပ္ပာယ်ကတော့ Status Code 200 OK နဲ့ လက်ခံရရှိတဲ့ Content တွေကို (၁၀) မိနစ် Cache လုပ်ပြီး Status Code 404 Not Found နဲ့ လက်ခံရရှိတဲ့ Content တွေကို တော့ (၁) မိနစ်သာ Cache လုပ်ပါလို့ သတ်မှတ်ထားခြင်းပဲဖြစ်ပါတယ်။

ဒါ Cache သိမ်းဆည်းရမယ့် Setting ပဲရှိပါသေးတယ်။ သိမ်းဆည်းထားတဲ့ Cache ကို အသုံးပြုဖို့အတွက် သက်ဆိုင်ရာ server (သို့မဟုတ်) location Block တွေထဲမှာ အခုလို သတ်မှတ်ပေးနိုင်ပါတယ်။

## Config

```
1. location /api {
2.     proxy_cache backcache;
3.     proxy_cache_bypass $http_cache_control;
4.
5.     proxy_pass http://api_servers;
6. }
```

proxy\_cache Directive ကိုသုံးပြီး အသုံးပြုရမယ့် Cache Key ကို သတ်မှတ်ပေးရပါတယ်။ အကယ်၍ Client က Cache-Control Header ထည့်သွင်းပေးပို့လာရင် လိုက်နာအသုံးပြုဖို့အတွက် proxy\_cache\_bypass Directive နဲ့ သက်မှတ်ပေးထားပါတယ်။ ဒါကြောင့် Client က Cache ကို မလိုချင်ဘူး၊ App Server ကသွားယူတာကိုပဲ လိုချင်တယ်ဆိုရင် "Cache-Control: no cache" ဆိုတဲ့ Header ကို Request နဲ့အတူ ထည့်သွင်းပေးပို့နိုင်ပါတယ်။ no cache Header ပါဝင်လာရင် Nginx က Cache ကို အသုံးမပြုပဲ Upstream Server ထံ Request ကို လက်ဆင့်ကမ်းပေးသွားမှာဖြစ်ပါတယ်။

Cache ကို အသုံးပြုတဲ့အခါ အတော်လေးသတိထားဖို့တွေ့လိုပါတယ်။ အခြေခံအားဖြင့် အမြော်ပြောင်းလဲနေခြင်း မရှိတဲ့ Image, Icon, Style, JavaScript Library စတဲ့ Static Content တွေကိုသာ Cache လုပ်သင့်ပါဘူး။ ဥပမာ - Database ထဲက User List ကို ပြန်ပေးတဲ့အခါ Cache မလုပ်သင့်ပါဘူး။ Cache လုပ်ထားရင် Database ထဲမှာ လက်တလော Data အသစ်တို့

နေပေမယ့် Nginx က ပြီးခဲ့တဲ့ (၁၀) မိနစ်က Cache ကိုပဲ ပြန်ပေးတဲ့အခါ Client က Data အသစ်ပါဝင်ခြင်းမရှိတဲ့ Content အဟောင်းကိုပဲ လက်ခံရရှိနေမှာဖြစ်ပါတယ်။ ဒီထက်ပိုဆိုးတာက User A Request လုပ်ထားတဲ့ Data ကို Cache လုပ်ထားပြီး User B ရဲ့ Request ကို လက်ခံရရှိတဲ့အခါ Cache တိုးကိုပြန်ပေးနေရင် User B က User A ရဲ့ Content ကို လက်ခံရရှိသွား မှာဖြစ်ပါတယ်။ ဒါကြောင့် ပြောင်းလဲမှာမဟုတ်တဲ့ Static Content တွေ ကိုသာ Cache လုပ်သင့်ပါတယ်။ တစ်ချို့ Response တွေကို Cache မလုပ်စေလိုရင် App Server မှာ အခုလို သတ်မှတ်ထားနိုင်ပါတယ်။

### JavaScript

```
1. app.use(function(req, res, next) {
2.   res.set("expires", "-1");
3.   res.set("Cache-Control", "no store");
4. });


```

ExpressJS App Server က Cache-Control Header နဲ့အတူ no store ကို ထည့်သွင်းပေးတဲ့အခါ Nginx Proxy က အဲဒီ Header ကို အသိအမှတ်ပြုပြီး လက်ခံရရှိတဲ့ Content ကို Cache အနေနဲ့ မသိမ်းဆည်းပဲ ထားပေးမှာပဲ ဖြစ်ပါတယ်။

ဒီနည်းနဲ့ Nginx ကို အသုံးပြုသူအနည်းအများ လိုအပ်ချက်ပေါ်မှတည်ပြီး App Server တွေ အချိန်မရွေး အတိုး အလျှော့ လုပ်နိုင်တဲ့ Load Balancer အနေနဲ့သာမက၊ App ရဲ့ စွမ်းဆောင်ရည် မြင့်တက်စေနိုးအတွက် Cache Proxy အနေနဲ့ပါ အသုံးပြုနိုင်မှာပဲ ဖြစ်ပါတယ်။

## Conclusion

Nginx ကို စတင်ဖန်တီးစဉ်ကတည်းက Reverse Proxy အနေနဲ့ အလုပ်လုပ်ဖို့ ဦးစားပေး ဖန်တီးခဲ့ခြင်း ဖြစ်ပါတယ်။ ဒါကြောင့်လည်း Nginx အတွက် Proxy Setting နဲ့ Load Balancing Setting တွေ သတ်မှတ်ရတာ ရှုပ်ထွေးခြင်းမရှိပဲ ရှိုးရှင်းလွယ်ကူနေခြင်း ဖြစ်ပါတယ်။ Nginx အပြင် အခြားလူသုံးများတဲ့ နည်းပညာတွေ လည်း ရှိပါသေးတယ်။ Load Balancing အတွက် HAProxy လိုခေါ်တဲ့ နည်းပညာကိုလည်း လူသုံးများကြပါတယ်။ ထူးခြားချက်ကတော့၊ HAProxy ဟာ Nginx ကို Application Layer Proxy မဟုတ်ပဲ Network Layer Proxy ဖြစ်ပါတယ်။ Cache Server အနေနဲ့ လည်း Varnish လိုခေါ်တဲ့ နည်းပညာကို လူသုံးများကြပါသေးတယ်။ Nginx ကတော့ Web Server အနေနဲ့သာမက၊ Proxy Server အနေနဲ့ရော့၊ Cache Server အနေနဲ့ပါ အားလုံးကို ဆောင်ရွက်ပေးနိုင်တဲ့အတွက် Nginx ကို ဒီနေရာမှာ ရွေးထုတ် ဖော်ပြလိုက်ရခြင်းပဲ ဖြစ်ပါတယ်။

Database Server Architecture စဉ်းစားတဲ့အခါ Hardware

Failure ကြောင့် Data မဆုံးရှုံးစေဖို့နဲ့ Data ပမာဏများလာတဲ့အခါ

Storage Capacity ကို အလွယ်တစ်ကူ တိုးမြှင့်နိုင်ဖို့ဆိုတဲ့ အချက်

နှစ်ချက်ကို အမိကစဉ်းစားရပါတယ်။

## **Rockstar Developer Course**

Project Management, Web Service, Server Architecture

NodeJS အစရိတ်ဆည်း ဤစာအုပ်ပါ အကြောင်းအရာများတို့

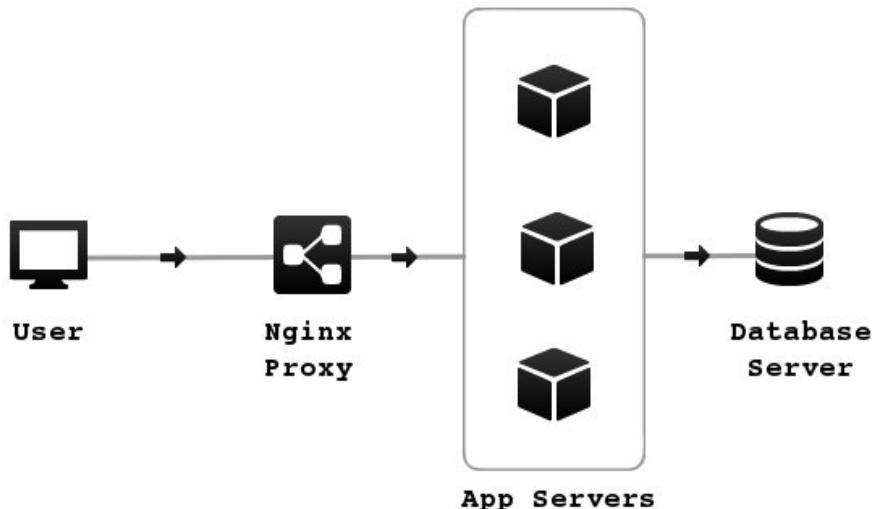
စာရေးသူကိုယ်တိုင် သင်ကြားပေးခြင်းဖြစ်သည်။

ဆက်သွယ်ရန် - (ဝါ) ၇၃၀ ၆၅၉ ၆၂

<http://eimaung.com/courses>

## အခန်း(၁၉) – MongoDB Replication and Sharded Cluster

ပြီးခဲ့တဲ့အခန်းမှာ Nginx ကိုသုံးပြီး Load Balance ပြုလုပ်နိုင်ပုံကို ဖော်ပြခဲ့ပါတယ်။ အဲဒီလို Load Balance ပြုလုပ်ရာမှာ App Server ကိုသာ Balance ပြုလုပ်ခြင်းဖြစ်ပြီး အချက်အလက်တွေ သိမ်းဆည်းထားတဲ့ Database ကိုတော့ ထည့် သွင်း စဉ်းစားထားခြင်း မရှိသေးပါဘူး။ ဒါကြောင့် ဒီနည်းနဲ့ Server Architecture ကို ဖန်တီးထားမယ်ဆိုရင် ရရှိမယ့် ပုံသဏ္ဌာန်က အခုလိုဖြစ်နေမှာပါ။



ပုံ (၁၉.၁) - Server Architecture without Database Load Balancing

အသုံးပြုသူများလာတဲ့ ဒါက်ကိုခံနိုင်အောင် App Server တွေကို Load Balance လုပ်ထားပေမယ့် Database Server ကတော့ တစ်ခုတည်းသာဖြစ်နေဖိုးမှာပါ။ ဒါကြောင့် Database Server မှာ လာပြီး နေးနေတာမျိုးဖြစ်နိုင်ပါတယ်။ Data ပမာဏများလာတဲ့အခါ လက်ခံသိမ်းဆည်းပေးနိုင်ခြင်းမရှိ ဖြစ်လာနိုင်ပါတယ်။

**အခန်း (၁၁)** မှာ MongoDB အကြောင်း လေ့လာခဲ့စဉ်က MongoDB ဟာ ဒီလိုအခြေအနေမျိုးမှာ Scale လုပ်ရ လွယ်အောင်ဖန်တီးထားခြင်း ဖြစ်တယ်ဆိုတဲ့အကြောင်းကိုဖော်ပြခဲ့ပြီးဖြစ်ပါတယ်။ ဒီအခန်းမှာတော့ MongoDB ရဲ့ စွမ်းဆောင်ရည် Performance နဲ့ Availability ကောင်းမွန်စေဖို့အတွက် Replication ပြုလုပ်ပုံနဲ့ Sharded Cluster လုပ်ဆောင် ချက်ကိုသုံးပြီး Scale ပြုလုပ်ပုံတို့အကြောင်းကို ဆက်လက်ဖော်ပြသွားမှာ ဖြစ်ပါတယ်။

## 19.1 – MongoDB Replica Sets

Replication လုပ်ဆောင်ချက်ဟာ Database စနစ်တိုင်းလိုလိမှာပါဝင်တဲ့ လုပ်ဆောင်ချက်တစ်ခုဖြစ်ပါတယ်။ သူ၏ အခြေခံ အလုပ်လုပ်ပုံကတော့ Master Database Server မှာသိမ်းဆည်းထားတဲ့ အချက်အလက်တွေကို Slave Database Server တွေမှာ မူဖွားယူထားခြင်းဖြစ်ပါတယ်။ ဒီလိမ့်မူဖွားယူထားတဲ့အတွက် ရရှိလာတဲ့ အကျိုးရလဒ် တွေကတော့ –

**Read Performance** – အချက်အလက်တွေ ရယူဖို့ Request ပြုလုပ်လာတဲ့အခါ Master Database Server က အမြဲ အကြောင်းပြန်နေဖို့ မလိုအပ်တော့ပဲ Slave Database Server တွေက ကိုယ်စားအကြောင်းပြန်ပေးနိုင်တဲ့ အတွက် Database ရဲ့ Read Performance စွမ်းဆောင်ရည် မြင့်တက်လာမှာပဲဖြစ်ပါတယ်။

**Data Safety** – Master Database Server အကြောင်းအမျိုးမျိုးနဲ့ အလုပ် မလုပ်တော့တဲ့အခါ Slave Database Server တွေမှာ အချက်အလက်တွေ ကျန်ရှိနေမှာဖြစ်တဲ့အတွက် အချက်အလက်လုပ်ခြေရေးကိုလည်း အထောက် အကူဖြစ်စေမှာဖြစ်ပါတယ်။

**Availability** – အကြောင်းအမျိုးမျိုးကြောင့် Master Database Server အလုပ်မလုပ်နိုင်တဲ့အခါ Slave Database Server တစ်ခုက Master အဖြစ် ချက်ခြင်းပြောင်းလဲအလုပ်လုပ်ပေးခြင်းအားဖြင့် လက်ရှိအလုပ်လုပ်နေတဲ့ App ကို ရပ်တန်းသွားစရာမလိုပဲ ဆက်လက်အလုပ်လုပ်နိုင်မှာဖြစ်ပါတယ်။

Replication ရဲ့ ထူးခြားချက်ကတော့ Write လုပ်ငန်းစဉ်အားလုံးဟာ Master Database Server မှာသာ လုပ်ရပါတယ်။ Slave Database Server တွေက Master ရဲ့ Data ကို မူဖွားယူခြင်းမဟုတ်ပဲ၊ Master ရဲ့ Operation Log (oplogs) ကို ဖတ်ပြီး Master မှာလုပ်ခဲ့တဲ့ အလုပ်တွေကို Slave ပေါ်မှာ ထပ်တူလုပ်ပေးခြင်းအားဖြင့် အချက်အလက်တွေကို ကူးယူခြင်းဖြစ်ပါတယ်။ အကယ်၍ Slave တွေမှာပါ Write လုပ်ငန်းစဉ်ကို ခွင့်ပြုလိုက်ရင် မလိုလားအပ်တဲ့ပြဿနာတွေ တက်နိုင်တဲ့အတွက် Write လုပ်ငန်းစဉ်အားလုံးကတော့ Master မှာပဲ လုပ်ရလေ့ရှိပါတယ်။ Read လုပ်ငန်းစဉ်ကိုတော့ Master နဲ့ Slave Server တွေ မျှပြီး ခွဲဝေလက်ခံ ဆောင်ရွက်ကြလေ့ရှိပါတယ်။

MongoDB မှာ Replication လုပ်ဆောင်ချက်ကို Replica Sets လိုခေါ်ပါတယ်။ Master Database Server ကို တော့ Primary Member လိုခေါ်လေ့ရှိပြီး၊ Slave Database Server ကိုတော့ Secondary Member လို ခေါ်လေ့ရှိပါတယ်။ MongoDB Replica Set တစ်ခုမှာ Member (၅၀) ထိ ပါဝင်နိုင်ပါတယ်။

ဒီအခန်းမှာ ဖော်ပြထားတဲ့ Mongo Shell နဲ့နာတွေဟာ တိုက်ရှိကူးယူစမ်းသပ်ဖို့မဟုတ်ပါ။ သဘော သဘာဝ ပေါ်လွင်အောင် Simplify လုပ်ထားတဲ့ နမူနာသက်သက်သာ ဖြစ်ပါတယ်။ ဒါကြောင့် သဘော သဘာဝကိုသာ နားလည်အောင်လေ့လာပြီး အသေးစိတ်ကို ကိုယ်တိုင်ဆက်လက် လေ့လာသွားရမှာ ဖြစ်ပါတယ်။

## 19.2 – Setting Up Replica Sets

MongoDB Replica Sets သတ်မှတ်ပုံ နဲ့မှနာဖော်ပြနိုင်ဖို့အတွက် ကျွန်ုတ်တို့မှာ အောက်ပါ Domain Name များနဲ့ MongoDB ကို အသင့် Install လုပ်ထားပြီးသား Database Server တွေရှိတယ်လို့ မှတ်ယူပေးပါ။

- db1.example.com
- db2.example.com
- db3.example.com

ပထမဦးဆုံးအနေနဲ့ db1.example.com ရဲ့ MongoDB Configuration မှာ replSet = rs0 ဆိုတဲ့ လိုင်းတစ်လိုင်း ထည့်ပေးရပါမယ်။ rs0 ဆိုတာ Replica Sets အမည်ဖြစ်ပြီး၊ အခြားမှတ်ရလွယ်တဲ့ အမည်တစ်ခုခုကို ပေးမယ် ဆိုလည်း ပေးနိုင်ပါတယ်။ Ubuntu မှာဆိုရင် /etc/mongod.conf ဆိုတဲ့အမည်နဲ့ MongoDB Configuration ဖိုင် က တည်ရှိမှာဖြစ်ပါတယ်။

ပြီးတဲ့အခါ MongoDB Server ကို Restart လုပ်ပေးဖို့လိုပါတယ်။

```
$ sudo service mongodb restart
```

ဆက်လက်ပြီး Mongo Shell ကို mongo Command နဲ့ ဝင်ရောက်ပြီး rs.initiate() Function ကို Run ပေးရပါမယ်။

```
$ mongo
> rs.initiate()
```

rs.initiate() Function က လက်ရှိ Server ကို Replica Sets ရဲ့ပထမဦးဆုံး Member အနေနဲ့ ထည့်သွင်းပေးသွားမှာဖြစ်ပါတယ်။ Replica Sets ရဲ့လက်ရှိအခြေအနေကို rs.conf() နဲ့ ခေါ်ယူကြည့်ရှုနိုင်ပါတယ်။

```
{
  "_id" : "rs0",
  "version" : 1,
  "members" : [
    {
      "_id" : 0,
      "host" : "db1.example.com:27017"
    }
  ]
}
```

ဆက်လက်ပြီး ကျွန်ုတ် MongoDB Server တွေကို rs.add() နဲ့ Replication Sets ထဲမှာ ထည့်ပေးနိုင်ပါတယ်။

```
> rs.add("db2.example.com");
> rs.add("db3.example.com");
```

ဒီနည်းနဲ့ Server (၃) လုံးပါဝင်တဲ့ MongoDB Replica Sets တစ်ခုကို အလွယ်တစ်ကူ တည်ဆောက်နိုင်ပါတယ်။ MongoDB Replica Sets တစ်ခုမှာ အနည်းဆုံး Member (၃) လုံး ပါဝင်သင့်ပါတယ်။

Replica Sets ထဲက Member တစ်လုံးကို ဖြန့်ထွက်လိုရင်တော့ rs.remove() ကို အသုံးပြုနိုင်ပါတယ်။

```
> rs.remove("db3.example.com");
```

အကယ်၍ Remove လုပ်လိုက်တဲ့ Member က Primary ဖြစ်နေရင် MongoDB က ကျွန် Member တွေထဲကတစ်ခုကို Primary အဖြစ် အလိုအလျောက် ရွေးချယ်သတ်မှတ်သွားမှာဖြစ်ပါတယ်။

### 19.3 – Arbiter Member in Replica Sets and Member Priority

MongoDB ဟာ Automatic Failover လုပ်ဆောင်ချက်နဲ့ Primary Member အလုပ်မလုပ်တော့အခါ Replica Sets ထဲက Secondary Member တစ်ခုကို အလိုအလျောက် Primary အဖြစ် ပြောင်းပေးသွားမှာဖြစ်ပါတယ်။ အဲဒီလို ပြောင်း ဖို့အတွက် Primary ရွေးချယ်ရေး ရွေးကောက်ပွဲ (Election) ကို ဆောက်ချက်ပါတယ်။ ဒီနေရာမှာ MongoDB Election Process ကို အသေးစိတ်မပြောတော့ပါဘူး။ လိုရင်းအနေနဲ့ မှတ်သားစေလိုတာကတော့ Election လုပ်ငန်းဆောင်ချက်ရာမှာ အထောက်အကြဖြစ်စေဖို့အတွက် Arbiter Member လိုပေါ်တဲ့ MongoDB Server တစ်မျိုးကို Replica Sets ထဲမှာ ထည့်သွင်းပေးနိုင်ပါတယ်။

Arbiter Member ဟာ Replica Sets ရဲ့ အစိတ်အပိုင်းတစ်ခုဖြစ်ပေမယ့် Secondary Member တွေလို Data တွေ ရူးသွေးသွေးတော့မရှိပါဘူး။ အကြောင်းအမျိုးမျိုးနဲ့ Primary Member အလုပ်မလုပ်တော့လို Secondary Member ထဲကတစ်ခုကို Primary ပြောင်းဖို့ ရွေးချယ်ရတဲ့ အချိန်မှာ ဘယ်သူကို Primary ပြောင်းပေးသင့်သလဲဆိုတဲ့ Vote ကို လုပ် ပေးပါတယ်။ Replica Sets ထဲမှာ Arbiter Member တစ်ခုကို အခုလို ထည့်သွင်းနိုင်ပါတယ်။

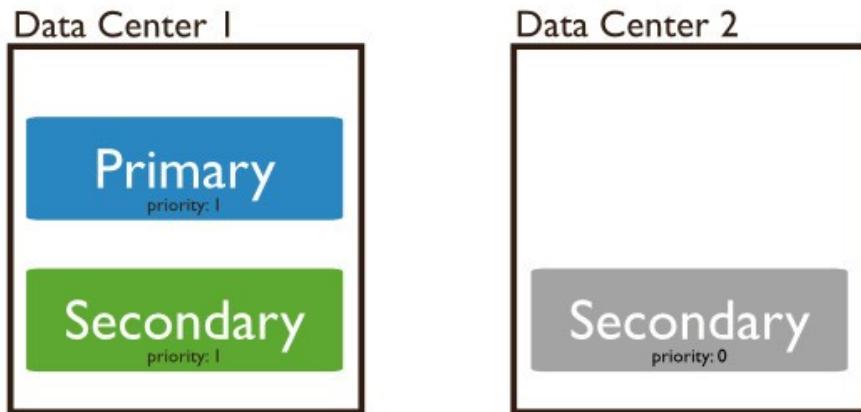
```
> rs.addArb("adb.example.com");
```

အခြေခံအားဖြင့် Replica Set ထဲမှာ Member အရေအတွက် စုံကိန်းဖြစ်နေရင် Arbiter Member ထည့်သွင်းသင့်ပါ တယ်။ မကိန်းဆိုရင်တော့ ထည့်သွင်းစရာမလိုပါဘူး။ ဥပမာ - စုံပေါင်း Member (၄) လုံးဆိုရင် Arbiter ထည့်ပြီး (၅) လုံးဖြစ်သွားအောင် ဆောက်ချက်ပေးသင့်ပါတယ်။ စုံပေါင်း Member (၃) လုံးဆိုရင်တော့ မကိန်းဖြစ်နေတဲ့ အတွက် Arbiter ထပ်ထည့်ဖို့ မလိုအပ်တော့ပါဘူး။

ဖြည့်စွက်သတိပြုသင့်တာကတော့ Replica Sets တစ်ခုမှာ Member (၅) ထိပါဝင်နိုင်ပေမယ့် Election လုပ်တဲ့ အခါ အမြှင့်ဆုံး (၇) လုံးထိသာ Vote လုပ်မှာဖြစ်ပါတယ်။ ဒါကြောင့် Member (၇) လုံးကျော်သွားရင်တော့ စုံပဲ

ဖြစ်ဖစ်၊ မပဲ ဖြစ်ဖစ်၊ Arbiter ကို သီးခြားထပ်ထည့်စရာ မလိုတော့ပါဘူး။

နောက်တစ်ချက်အနေနဲ့ သတိပြုသင့်တာကတော့၊ Primary ဘယ်တော့မှုမဖြစ်စေလိုဘူး၊ Member ကို Priority 0 အဖြစ် သတ်မှတ်ထားနိုင်ပါတယ်။ အထူးသဖြင့် Member Server တွေကို နေရာဒေသမတူတဲ့ Data Center မှာခဲ့ထားတဲ့အခါ သီးခြားဖြစ်နေတဲ့ Server တွေကို Primary အဖြစ် မသတ်မှတ်သင့်ပါဘူး။



ပုံ (၁၉.၂) - Multi-Data Center Replica Sets

source: [mongodb.org](http://mongodb.org)

သီးခြား Data Center မှာ ရှိနေတဲ့ Member က Primary ဖြစ်သွားရင် Replica Sets ရဲ့ အလုပ်လုပ်ပုံ မလိုအပ်ပဲ နေးသွားနိုင်တဲ့ အတွက်ဖြစ်ပါတယ်။ ဒါကြောင့် အဲဒီလို Member မျိုးကို Priority 0 သတ်မှတ်ပေးထားသင့်ပါတယ်။ အခုလို သတ်မှတ်နိုင်ပါတယ်။

```
> rs.conf()

{
  "_id" : "rs0",
  "version" : 1,
  "members" : [
    {
      "_id" : 0,
      "host" : "db1.example.com:27017"
    },
    {
      "_id" : 1,
      "host" : "db2.example.com:27017"
    },
    {
      "_id" : 2,
      "host" : "db3.example.com:27017"
    }
  ]
}

> cfg = rs.conf();
```

```
> cfg.members[2].priority = 0;
> rs.reconfig(cfg);
```

လက်ရှိ Replica Sets Configuration ထဲက Member တစ်ခုရဲ့ Priority တန်ဖိုးကို 0 လို သတ်မှတ်ပေးလိုက်ခြင်း ဖြစ်ပါ တယ်။ ဒါကြောင့် အဲဒီ Member ကို Vote လုပ်ချိန်မှာ Primary အဖြစ်မပြောင်းပဲ ချုန်လုပ်ပေးသွားမှာပဲ ဖြစ်ပါတယ်။

#### 19.4 – Hidden and Delayed Member in Replica Sets

Replica Sets ထဲက တစ်ချို့ Member တွေကို Hidden Member အနေနဲ့ သတ်မှတ်ထားနိုင်ပါတယ်။ Hidden Member တွေဟာ Data တွေကို အခြား Secondary Member များနည်းတူ မူမွားယူပေါ်ယူ။ Client Request တွေကို လက်ခံအလုပ်လုပ်မှာ မဟုတ်ပါဘူး။ Data Backup သက်သက်ပြေလုပ်လိုတဲ့အခါ အသုံးပြုနိုင်ပါတယ်။ ဒါမှမဟုတ် လက်ရှိ Application Data ကို မထိပဲ Statistics နဲ့ Report တွေကို သီးခြားထုတ်ယူချင်တဲ့အခါ Hidden Member ထဲက မူဖွား ယူထားတဲ့ Data တွေကို အသုံးပြုနိုင်မှာဖြစ်ပါတယ်။

Member တစ်ခုကို Hidden Member အဖြစ်အခုလို သတ်မှတ်နိုင်ပါတယ်။

```
> cfg = rs.conf();
> cfg.members[2].priority = 0;
> cfg.members[2].hidden = true;
> rs.reconfig(cfg);
```

Hidden လုပ်လိုတဲ့ Member ရဲ့ hidden ကို true လို သတ်မှတ်ပေးလိုက်ခြင်းဖြစ်ပါတယ်။ သတိပြုသင့်တာ ကတော့ Hidden Member ကို Priority 0 မဖြစ်မနေ သတ်မှတ်ထားပေးဖို့ လိုအပ်ပါတယ်။

Replica Sets ထဲက Member တစ်ခုကို Delay Member အနေနဲ့လည်း သတ်မှတ်ထားနိုင်ပါတယ်။ Delay Member ဆိုတာ မူမွားကူယူတဲ့လုပ်ငန်းကို ချက်ခြင်းမလုပ်ပဲ သတ်မှတ်ထားတဲ့အချိန်တစ်ခုစောင့်ပြီးမှ ကူးယူ တဲ့ Member ဖြစ်ပါ တယ်။ Backup လုပ်ဖို့အတွက် အသုံးဝင်ပါတယ်။ Delay Member က Data တွေကို ချက်ခြင်းကူးမယူတဲ့အတွက် Primary Member ပေါ်မှာ မတော်တဆ အမှားအယွင်းတစ်ခုခုလုပ်မိခဲ့ရင် အဲဒီအမှားက Delay Member ကို ချက်ချင်း မရောက်တဲ့အတွက် အမှားမပါဝင်သေးတဲ့ မူလအချက်အလက်တွေကို Delay Member ကနေ ပြန်လည်ရယူနိုင်မှာပဲ ဖြစ် ပါတယ်။

Delay Member ရဲ့ Priority ကို 0 သတ်မှတ်ထားဖို့လိုသလို Hidden ကိုလည်း True သတ်မှတ်ထားပေးဖို့လိုပါ တယ်။ Member တစ်ခုကို Delay Member အဖြစ် အခုလိုသတ်မှတ်နိုင်ပါတယ်။

```
> cfg = rs.conf();
> cfg.members[2].priority = 0;
> cfg.members[2].hidden = true;
```

```
> cfg.members[2].slaveDelay = 3600;
> rs.reconfig(cfg);
```

Member ရဲ့ slaveDelay Property အတွက် မူးယူမှုမှု စောင့်ရမယ့် အချိန်ကာလစတ္တန်ကို ပေးခြင်း အားဖြင့် Delay Member အဖြစ် သတ်မှတ်နိုင်ခြင်းဖြစ်ပါတယ်။ နမူနာအရ member[2] ဟာ အချိန်စတ္တန် ၃၆၀၀ (၁ နာရီ) စောင့်ဆိုင်းပြီးတော့မှ Primary က အချက်အလက်တွေကို မူးယူမှုမှုဖြစ်ပါတယ်။ တန်ည်း အားဖြင့် member[2] မှာ အမြတမ်း တစ်နာရီနောက်ကျတဲ့ အချက်အလက်တွေ ရှိနေမှုပဲဖြစ်ပါတယ်။

## 19.5 – Forcing a Member to be Primary in Replica Sets

ပုံမှန်အားဖြင့် Replica Sets တစ်ခုမှာ ဘယ်သူ Primary Member ဖြစ်သင့်သလဲဆိုတာကို MongoDB က Election လုပ်ပြီးဆုံးဖြတ်ပါတယ်။ Member တွေထဲက တစ်ခုကို Primary ဖြစ်အောင် ကိုယ်တိုင်သတ်မှတ်လိုရင် တော့ သတ်မှတ်လို တဲ့ Member ရဲ့ Priority ကို ကျွန်း Member များထက် မြင့်အောင်ပေးခြင်းအားဖြင့် သတ်မှတ်နိုင်ပါတယ်။ Priority တစ်ဖိုးကို 0 ကနေ 1000 အထိ ကိန်းပြည့်အနေနဲ့ရော၊ ဒသာမကိန်းအနေနဲ့ပါ ပေးနိုင်ပါတယ်။ Priority ကို 0 သတ်မှတ် ထားရင်တော့ Primary မဖြစ်နိုင်တာကို အထက်မှာ ဖော်ပြခဲ့ပြီးဖြစ်ပါတယ်။

Member တွေထဲက members[2] ကို Primary Member ဖြစ်စေလိုတယ်ဆိုရင် အခုလိုသတ်မှတ်နိုင်ပါတယ်။

```
> cfg = rs.conf();
> cfg.members[0].priority = 0.5;
> cfg.members[1].priority = 0.5;
> cfg.members[2].priority = 1;
> rs.reconfig(cfg);
```

ဒီနည်းနဲ့ members[2] ရဲ့ Priority ကျွန်း Member များထက်မြင့်သွားတဲ့အခါ member[2] ကို Primary အဖြစ် MongoDB က ရွေးချယ်ပေးလိုက်မှုပဲဖြစ်ပါတယ်။

လက်ရှိ ဘယ်သူက Primary ဖြစ်ပြီး ဘယ်သူတွေက Secondary လည်းဆိုတာကို အခုလိုလေ့လာနိုင်ပါတယ်။

```
> db.runCommand("ismaster");
{
  "ismaster" : false,
  "secondary" : true,
  "hosts" : [
    "db1.example.com",
    "db2.example.com"
  ],
  "passives" : [
    "db3.example.com"
  ],
  "arbiters" : [
    "db4.example.com",
  ]
}
```

```

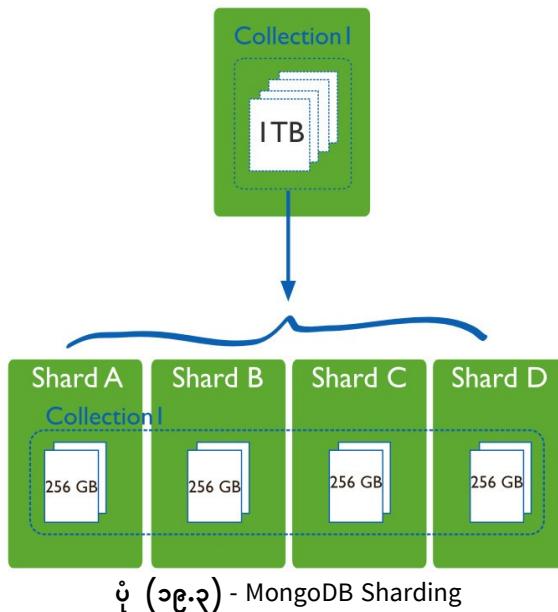
        ]
    "primary" : "db2.example.com",
    "ok" : true
}

```

နဲ့မူနာအရ db2.example.com က Primary ဖြစ်နေတာကို တွေ့ရမှာပါ။ ရလဒ်ထဲမှာ Priority 0 သတ်မှတ်ထားတဲ့ Passive Server စာရင်းနဲ့ Arbiter စာရင်းကိုလည်း ရှိနေရင် ထည့်သွင်းဖော်ပြပေးမှာဖြစ်ပါတယ်။

## 19.6 – Sharding in MongoDB

Replica Sets လုပ်ဆောင်ချက်ဟာ Reliability နဲ့ Read Performance အတွက် အသုံးဝင်ပါတယ်။ ဒါပေမယ့် Data အများအပြားကို Server တွေခဲ့သိမ်းလိုရင်တော့ Sharding လုပ်ဆောင်ချက်ကို အသုံးပြုရမှာဖြစ်ပါတယ်။ အခြေခံအားဖြင့် Sharding ဆိုတာ Data တွေကို တစ်လုံးထက်ပိုတဲ့ Database Server တွေမှာ ခွဲခြားသိမ်းဆည်းပေးခြင်းဖြစ်ပါတယ်။ ဒီနည်းနဲ့ Storage လိုအပ်လာတိုင်း လက်ရှိစနစ်ကို မထိခိုက်စေပဲ Database Server တွေကို လိုအပ်သလို ထပ်မံထည့်သွင်းတဲ့ တိုးချွိန်မှာပဲဖြစ်ပါတယ်။



source: [mongodb.org](http://mongodb.org)

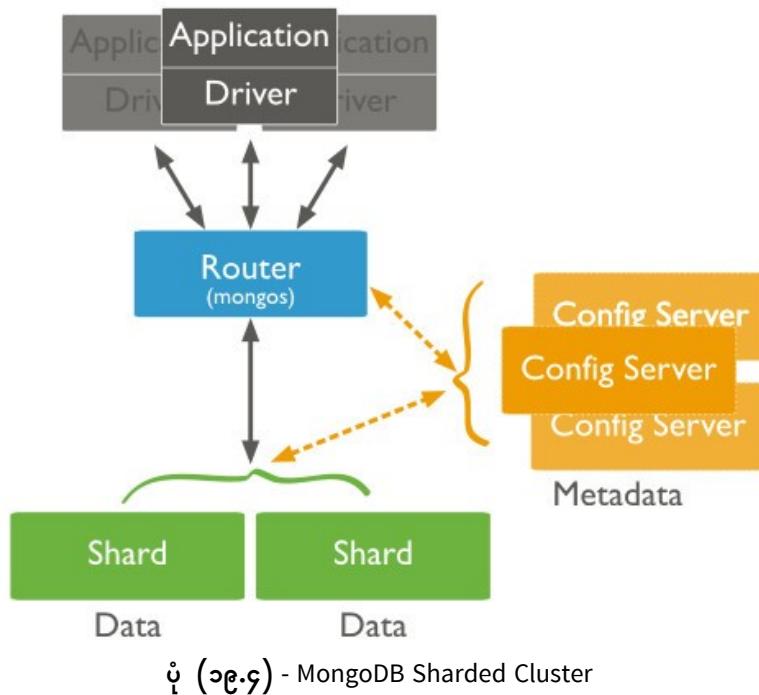
MongoDB ရဲ့ Sharding လုပ်ဆောင်ချက်ကို အသုံးချွိန်ဖို့အတွက် အစိတ်အပိုင်း (၃) ရပ်လိုအပ်ပါတယ်။ Config Servers, Query Routers နဲ့ Shards တို့ပဲဖြစ်ပါတယ်။ အားလုံးစုပေါင်းထားတဲ့ လုပ်ဆောင်ချက်ကို Sharded Cluster လို့ ခေါ်ပါတယ်။

Config Servers ရဲ့တာဝန်ကတော့ Data တွေကို ဘယ်နည်းဖြန့်ကျက်သိမ်းဆည်းထားတဲ့ဆိုတဲ့ အချက်အလက်မှတ်တမ်း ကို သိမ်းဆည်းဖို့ဖြစ်ပါတယ်။ Sharded Cluster တစ်ခုမှာ Config Server (၃) ခု ပါဝင်ရပါတယ်။

Query Routers ရဲ တာဝန်ကတော့ App က ပေးပိုလာတဲ့ Request တွေကိုလက်ခံစွဲဖြစ်ပါတယ်။ Request တွေ လက်ခံရရှိတဲ့အခါ Query Router က Config Server ထံမှာ ရယူလိုတဲ့ Data တွေ ဘယ်လိုသိမ်းထားသလဲ သွားရောက် လေ့လာပြီး၊ Data တွေရယူကာ App ထဲ ပြန်လည်ပေးပိုပေးမှာဖြစ်ပါတယ်။

Shards ရဲတာဝန်ကတော့ Data တွေကို လက်ခံသိမ်းဆည်းဖို့ပဲဖြစ်ပါတယ်။ Shard တစ်ခုဟာ MongoDB Server တစ်ခု ဖြစ်နိုင်သလို၊ Replica Sets တစ်ခုလည်း ဖြစ်နိုင်ပါတယ်။ လက်တွေ Production မှာတော့ Replica Sets ကို Shard အဖြစ် အသုံးပြုရမှာဖြစ်ပါတယ်။

Sharded Cluster တစ်ခုရဲ ဖွဲ့စည်းပုံကို ပုံ (၁၉.၄) မှာ လေ့လာနိုင်ပါတယ်။



source: [mongodb.org](http://mongodb.org)

Sharded Cluster တစ်ခုမှာ တစ်ခုထက်ပိုတဲ့ Query Router တွေ ပါဝင်နိုင်ပါတယ်။ အနည်းဆုံး Query Router တစ်ခု မဖြစ်မနေ ပါဝင်ရပါမယ်။ အချက်အလက်တွေ သိမ်းဆည်းဖို့အတွက် တစ်ခုထက်ပိုတဲ့ Shard တွေပါဝင် နိုင်ပါတယ်။ အနည်းဆုံး Shard (၂) ခုပါဝင်ရပါတယ်။ Config Server ကတော့ အတိအကျ (၃) ခုပါဝင်ရမှာဖြစ်ပါတယ်။

Sharded Cluster သတ်မှတ်ပုံ နမူနာဖော်ပြနိုင်ဖို့အတွက် ကျွန်ုတ်တို့မှာ အောက်ပါ Domain Name များနဲ့ MongoDB ကို Install လုပ်ထားပြီးသား Database Server များရှိတယ်လို့ မှတ်ယူပေးပါ။

- cfg1.example.com (Config Server 1)
- cfg2.example.com (Config Server 2)
- cfg3.example.com (Config Server 3)
- router.example.com (Query Router)
- rs0 (Replica Sets 1)
- rs1 (Replica Sets 2)

Config Server အတွက် Server (၃) ခါ၊ Query Router အတွက် Server တစ်ခုနဲ့ Shard အတွက် Replica Sets (၂) ခုတို့ဖြစ်ပါတယ်။

### 19.7 – Setting Up Config Servers for Sharded Cluster

ပထမဦးဆုံးအနေနဲ့ Config Server ထွက် အရင် Setup လုပ်ရပါမယ်။ သေချာသွားအောင် Config Server အဖြစ် အသုံးပြုမယ့် Server မှာ MongoDB Service ကို Stop လုပ်ပေးသင့်ပါတယ်။

```
$ sudo service mongodb stop
```

ပြီးတဲ့အခါ Configuration Meta Data ထွေသိမ်းဖို့အတွက် Directory တစ်ခုသီးခြားတည်ဆောက်ပေးသင့်ပါတယ်။

```
$ mkdir -p /data/configdb
```

ပြီးတဲ့အခါ MongoDB ကို Install လုပ်စဉ်က တစ်ခါတည်းပါဝင်လာတဲ့ mongod ကိုသုံးပြီး Config Server ကို အခုလို စတင်နိုင်ပါတယ်။

```
$ sudo mongod --configsvr --dbpath /data/configdb --port 27019
```

mongod Command က အခြေခံအားဖြင့် MongoDB Server ကို စတင်ပေးခြင်းဖြစ်ပါတယ်။ အဲဒီလို စတင်တဲ့ အခါ ရိုးရိုး Server အနေနဲ့မဟုတ်ပဲ Config Server အနေနဲ့စတင်ဖို့အတွက် --configsvr Option ကို တွဲဖော်ပေးရခြင်း ဖြစ်ပါတယ်။ Meta Data ထွေ သိမ်းရမယ့် Location ကိုတော့ --dbpath Option နဲ့ တွဲပေးပြီး Config Server Port အဖြစ် 27019 ကို --port Option နဲ့ တွဲဖော်ပေးခြင်းဖြစ်ပါတယ်။ 27019 ဟာ MongoDB Config Server အတွက် Default Port ဖြစ်ပါတယ်။

ဒီနည်းအတိုင်း Config Server အဖြစ် အသုံးပြုမယ့် cfg1.example.com, cfg2.example.com နဲ့ cfg3.example.com Server (၃) ခုစလုံးမှာ ဆောင်ရွက်ပေးရမှာပဲဖြစ်ပါတယ်။

## 19.8 – Setting Up Query Router for Sharded Cluster

ဆက်လက်ပြီး Query Router အဖြစ် အသုံးပြုမယ့် router.example.com Server မှာ MongoDB Install လုပ်စဉ်က ပါဝင်လာတဲ့ mongos ကိုသုံးပြီး Router ကို Run ပေးရမှာဖြစ်ပါတယ်။ ပထမဗီးဆုံးအနေနဲ့ သေချာအောင် မူလ MongoDB ကို Stop လုပ်ထား သင့်ပါတယ်။

```
$ sudo service mongodb stop
$ sudo mongos --configdb cfg1.example.com:27019,cfg2.example.com:27019,cfg3.example.com:27019
```

ပြီးတဲ့အခါ mongos Command အတွက် --configdb Command နဲ့အတူ Config Server စာရင်းကိုတွဲဖက် ပေးရမှာဖြစ်ပါတယ်။ အကယ်၍ Query Router တစ်ခုထက်ပိုသုံးလိုတယ်ဆိုရင်၊ ထပ်မံထည့်သွင်းတဲ့ Query Router Server တွေမှာ mongos Command နဲ့အတူ Config Server စာရင်းကို ပထမ Server မှာ ပေးခဲ့တဲ့ အတိုင်း ရှုနောက်အစီအစဉ် အတိအကျနဲ့ ပေးရမှာဖြစ်ပါတယ်။ ပါဝင်တဲ့ Config Server အရေအတွက် တူညီ ယုံသာမက၊ ပေးလိုက်တဲ့ ရှုနောက်အစီအစဉ်လည်း တူညီဖို့လိုအပ်ပါတယ်။

Query Router Server အတွက် Data Directory သတ်မှတ်ပေးစရာမလိုပါဘူး။ Query Router တစ်ခုကို သီးခြား Server တစ်လုံးနဲ့မထားပဲ Config Server (သို့မဟုတ်) ရှိုးရှိုး MongoDB Database Server နဲ့လဲ တွဲ Run ထားလို ရနိုင်ပါတယ်။ ဒါပေမယ့် လက်တွေမှာ၊ သီးခြား Server တစ်လုံးနဲ့ထားလိုက်တာက စီမံရပိုမိုလွယ်ကူစေမှာ ဖြစ်ပါတယ်။

## 19.9 – Adding Replica Sets as Shard to Sharded Cluster

ဆက်လက်ပြီး MongoDB Server (သို့မဟုတ်) Replica Sets တွေကို အချက်အလက်တွေ အမှန်တစ်ကယ် သိမ်းဆည်းရာမှာ အသုံးပြုမယ့် Shard အဖြစ် Sharded Cluster ထဲမှာ ထည့်သွင်းပေးရမှာဖြစ်ပါတယ်။ ပထမ ဦးဆုံးအနေနဲ့ Query Router ကို Mongo Shell ကနေတစ်ဆင့် ချိတ်ဆက်ပေးရမှာဖြစ်ပါတယ်။

```
$ mongo --host router.example.com --port 27017
```

Query Router ကို Mongo Shell နဲ့ ချိတ်ဆက်ပြီးတဲ့အခါ sh.addShard() Function ကိုသုံးပြီး Shard တွေ စတင်ထည့်သွင်းနိုင်ပြီဖြစ်ပါတယ်။

```
> sh.addShard( "rs0/db1.example.com:27017" );
> sh.addShard( "rs1/db5.example.com:27017" );
```

နဲ့မူနာမှာ rs0 Replica Sets နဲ့ rs1 Replica Sets တို့ကို Sharded Cluster ထဲမှာ ထည့်သွင်းထားပါတယ်။ Replica Sets တစ်ခုမှာ Database Server အများအပြားပါဝင်နိုင်ပေမယ့် အားလုံးကို တစ်ချိုင်းထည့်ပေးစရာမ လိုပါဘူး။ Replica Sets ထဲက Server တစ်လုံးကို ပေးလိုက်ရင် ကျွန်း Server တွေကို MongoDB က အလို အလျောက် ရှာဖွေထည့်သွင်းပေးသွားမှာဖြစ်ပါတယ်။ ဒါကြောင့် နဲ့မူနာမှာ Replica Sets တွေထဲက Server

တစ်ခုစီကိုသာ sh.addShard() နဲ့ ထည့်သွင်းပေးထားခြင်းဖြစ်ပါတယ်။

အကယ်၍ Replica Sets မဟုတ်ပဲ ရှိုးရှိုး MongoDB Server တစ်လုံးကို ထည့်သွင်းချင်တာဆိုရင်တော့ အခုလုံး ထည့်သွင်းနိုင်ပါတယ်။

```
> sh.addShard("db10.example.com:27017");
```

နောက်ဆုံးက Port နံပါတ် 27017 က MongoDB Default Port ဖြစ်ပါတယ်။ မထည့်လည်းရပေမယ့် သေချာ အောင် ထည့်သွင်းပေးထားခြင်းဖြစ်ပါတယ်။

ဆက်လက်ပြီး Shard လုပ်လိုတဲ့ Database ကို အခုလုံး သတ်မှတ်ပေးရပါမယ်။

```
> sh.enableSharding("mydb");
```

နဲ့မူနာအရ mydb Database အတွက် Sharding လုပ်ဆောင်ချက်ကို အသုံးပြုပြီး Data တွေကို Sharded Cluster ပေါ်မှာ ဖြန့်သိမ်းပေးသွားတော့မှာဖြစ်ပါတယ်။

ဘယ် Database တွေမှာ Shard လုပ်ဆောင်ချက် သုံးထားသလဲဆိုတာ သိချင်ရင်တော့ config Database မှာ ကြည့်နိုင်ပါတယ်။ config ဟာ Database Sharding နဲ့ပက်သက်တဲ့ အချက်အလက်တွေ သိမ်းဆည်းဖို့ အတွက် MongoDB အလိုအလျောက် တည်ဆောက်သွားတဲ့ Database တစ်ခုဖြစ်ပါတယ်။ ဥပမာ -

```
> use config;
> db.databases.find();

{ "_id" : "admin", "partitioned" : false, "primary" : "config" }
{ "_id" : "mydb", "partitioned" : true, "primary" : "shard0003" }
```

နဲ့မူနာအရ mydb Database ကို Partition လုပ်ထားတယ် (Shard လုပ်ထားတယ်) လို့ ဖော်ပြထားတာကို တွေ့ရ မှာဖြစ်ပါတယ်။

## 19.10 – Enable Sharding for Collections

Database မှာ Shard လုပ်ဆောင်ချက် Enable လုပ်ပြီးတဲ့အခါ Collections တွေအတွက်လည်း Shard လုပ်ဆောင်ချက် Enable လုပ်ပေးရပြီးမှာပါ။ MongoDB က Collection တဲ့ကအချက်အလက်တွေကို Sharded Cluster ပေါ်မှာ ဖြန့်သိမ်းတဲ့အခါ Shard Key လို့ခေါ်တဲ့အညွှန်းကိုကြည့်ပြီး အုပ်စွဲသိမ်းလေ့ရှိပါတယ်။ ရည်ရွယ်ချက်ကတော့ Data တွေ ရယူတဲ့အခါ မလိုအပ်တဲ့ Shard နှစ်ခုသုံးခကနေ ခွဲပြီးယူမနေပဲ တစ်ခုတည်းကနေ ရယူနိုင်ဖို့အတွက်ဖြစ်ပါတယ်။

ဥပမာ - users Collection အတွက် city Field ကို Shard Key အနေနဲ့ သတ်မှတ်ထားမယ်ဆိုရင်၊ မြို့တူတဲ့ သူတွေကို Shard တစ်ခုတည်းမှာ သိမ်းပေးသွားမှာဖြစ်ပါတယ်။ ဒါကြောင့် မြို့တစ်မြို့က User စာရင်းရယူတဲ့ အခါ Shard နှစ်ခုသုံးခုကနေ သွားယူစရာမလိုပဲ တစ်ခုတည်းကနေ ရယူနိုင်မှာဖြစ်ပါတယ်။

```
> sh.shardCollection("mydb.users", { "city": 1 } );
```

နမူနာအရ city Field ကို Shard Key အနေနဲ့သုံးပြီး mydb Database ရဲ့ users Collection ကို Shard လုပ်ပေး သွားမှာဖြစ်ပါတယ်။ ဒီနည်းရဲ့ အားနည်းချက်အနေနဲ့ User ၉၀% လောက်က မြို့တစ်မြို့တည်းကဖြစ်နေ မယ်ဆိုရင် အချက် အလက် ၉၀% ကို Shard တစ်ခုတည်းမှာသွားသိမ်းမှာဖြစ်လို့ Storage အသုံးပြုမှု မမျှတတာ မျိုးဖြစ်နိုင်ပါတယ်။ ဒီလို့ အခြေအနေမျိုးမှာ Compound Shard Key ကို အသုံးပြုနိုင်ပါတယ်။

```
> sh.shardCollection("mydb.users", { "city": 1, "date_of_birth": 1 } );
```

နမူနာအရ city Field ကို Shard Key အနေနဲ့အသုံးပြုပြီး လိုအပ်လို့ Data တွေကို ထပ်မံတိတ်ပိုင်းပြီး အခြား Shard မှာ သွားသိမ်းဖို့လို့မယ်ဆိုရင် date\_of\_birth ကိုသုံးပါလို့ သတ်မှတ်ထားခြင်းဖြစ်ပါတယ်။ ဒါကြောင့် မြို့တစ်မြို့တည်းက User တွေကို Shard တစ်ခုတည်းမှာ စပြုသိမ်းရလို့ Shard Storage မလောက်တဲ့ အခါ၊ အဲဒီ မြို့က User တွေရဲ့ မွေးနေ့နဲ့ အချက်အလက်တွေကို စိတ်ပိုင်းပြီး အခြား Shard တစ်ခုမှာ ခွဲသိမ်းပေးမှာဖြစ်ပါတယ်။ ဥပမာ - ရန်ကုန် User တွေထဲက ၁၉၈၀ ကနေ ၁၉၉၀ ကြားမွေးသူတွေကို Shard 1 မှာသိမ်းပြီး ၁၉၉၀ နောက်ပိုင်းမွေးသူတွေကို Shard 2 မှာ သိမ်းပေးတဲ့သဘောမျိုး ဖြစ်ပါတယ်။

ဒါကြောင့် MongoDB ရဲ့ လုပ်ဆောင်ချက်မြန်ဆန်စေဖို့အတွက် Shard Key ရွေးချယ်တဲ့အခါ သတိပြုရွေးချယ်ရ မှာဖြစ်ပါတယ်။ အကယ်၍ ဘယ် Field ကို Shard Key အဖြစ် ရွေးချယ်ရမယ်ဆိုတာ မဆုံးဖြတ်နိုင်ရင် \_id Field ကို အသုံးပြု နိုင်ပါတယ်။ \_id ဆိုတာ ဘယ်တော့မှ ပြန်ထပ်မာမဟုတ်လို့ Shard တစ်ခုတည်းမှာ Data တွေ စပြုနေတာမျိုး မဖြစ်စေ တော့ပါဘူး။ ဒါပေမယ့် Data တွေရယူတဲ့အခါ Data Set တစ်ခုရရှိအတွက် Shard နှစ်ခုသုံးခုကနေ ဖြန့်ယူရတာမျိုး ဖြစ်နိုင်တာကိုတော့ နားလည်ထားရမှာဖြစ်ပါတယ်။

```
> sh.shardCollection("mydb.users", { "_id": "hashed" } );
```

နောက်တစ်ချက်အနေနဲ့ သတိပြုသင့်တာကတော့၊ Shard Key အဖြစ် အသုံးပြုမယ့် Field ကို Index လုပ်ထားဖို့လိုပါ တယ်။ အကယ်၍ Collection က Data တွေ မရှိသေးပဲ အလွတ်ဆိုရင် sh.shardCollection() Function က Shard Key အဖြစ်သုံးဖို့ပေးလိုက်တဲ့ Field တွေကို အလိုအလျောက် Index လုပ်ပေးသွားမှာဖြစ်ပါတယ်။ အကယ်၍ မူလက Data တွေ ရှိနေပြီးဖြစ်တယ်ဆိုရင်တော့ ကိုယ်တိုင် Index လုပ်ပေဖို့ လိုနိုင်ပါတယ်။ Index ပြုလုပ်ပုံကို အခန်း (၁၁) မှာ ဖော်ပြခဲ့ပြီးဖြစ်ပါတယ်။

အထက်ကန်မူနာမှာ { "\_id": 1 } လို့ မသုံးပဲ { "\_id": "hashed" } လို့သုံးထားတာကို သတိပြုမိနိုင်ပါတယ်။ အမိပါယ်ကတော့၊ \_id Field ကို Index လုပ်တဲ့အခါ ရှိရှိး Index ကိုမသုံးပဲ Hashed Index ကို သုံးမယ်လို့ ဆို

လိုက်ခြင်းဖြစ်ပါတယ်။ Hashed Index ကို ရှိရှိး Number နဲ့ String တန်ဖိုးမဟုတ်တဲ့တန်ဖိုးတွေ သိမ်းဖွံ့ဖြုံးတဲ့ Field တွေမှာ သတ်မှတ်သင့်ပါတယ်။ MongoDB က Field ထဲက တန်ဖိုးတွေကို Index မလုပ်မိ၊ Hash Summary ပြောင်းပြီးတော့မှ Index လုပ်ပေးမှာဖြစ်ပါတယ်။ `_id` Field ဟာဆိုရင်လည်း ရှိုးရှိုးတန်ဖိုးတွေသိမ်းတဲ့ Field မဟုတ်ပဲ `ObjectId()` Function သုံးပြီးသိမ်းပြီး ID တွေကိုသိမ်းတဲ့ Field ဖြစ်တဲ့အတွက် သူကို Shard Key အနေနဲ့သုံးမယ် ဆိုရင် Hashed Index လုပ်ပြီး အသုံးပြုသင့်တဲ့အတွက် `{ "_id": "hashed" }` ကို အသုံးပြုထားခြင်းပဲဖြစ် ပါတယ်။

လက်ရှိ Sharded Cluster Configuration ကို လေ့လာလိုရင် Query Router ကို Mongo Shell နဲ့ချိတ်ဆက်ပြီး အခုလို လေ့လာနိုင်ပါတယ်။

```
> sh.status();

--- Sharding Status ---
sharding version: { "_id" : 1, "version" : 3 }
shards:
{ "_id" : "shard0000", "host" : "db1.example.com:27017" }
{ "_id" : "shard0001", "host" : "db5.example.com:27017" }
databases:
{ "_id" : "admin", "partitioned" : false, "primary" : "config" }
{ "_id" : "mydb", "partitioned" : true, "primary" : "shard0000" }
mydb.users
  shard key: { "roll_no" : 1 }
  chunks:
    shard0001 2
    shard0000 2
{ "roll_no" : { "$minKey":1 } } --> { "roll_no" : "56000" } on : shard0001 { "t":2, "i":0 }
{ "roll_no" : 57500 } --> { "roll_no" : "58140" } on : shard0001 { "t" : 4, "i" : 0 }
{ "roll_no" : 58140 } --> { "roll_no" : "59000" } on : shard0000 { "t" : 4, "i" : 1 }
{ "roll_no" : 59000 } --> { "roll_no" : { "$maxKey":1 } } on : shard0000 { "t":3, "i":3 }
```

`sh.status()` ကို အသုံးပြုရခြင်းဖြစ်ပြီး လက်ရှိ Data တွေကို Cluster ပေါ်မှာ ဘယ်ပုံဘယ်နည်းသိမ်းထားတယ် ဆိုတဲ့ အချက်အလက်တွေကို ဖော်ပြပေးလာမှာပဲဖြစ်ပါတယ်။

## 19.11 – Removing a Shard from Cluster

Cluster ထဲမှာပါဝင်နေဖြီး လက်ရှိလည်း အချက်အလက်တွေ လက်ခံသိမ်းဆည်းနေတဲ့ Shard တစ်ခုကို အကြောင်းအမျိုးမျိုးကြောင့် ပြန်လည်ဖယ်ထုတ်တော့မယ်ဆိုရင် အဲဒါ Shard မှာသိမ်းထားတဲ့ Data တွေကို ကျွန်း Shard တွေထံပေးပိုတဲ့ လုပ်ငန်းကို အရင်လုပ်ရမှာဖြစ်ပါတယ်။ ပထမဗြိုံးဆုံးအနေနဲ့ Shard ရဲ့ ID ကို သိရ ဖို့အတွက် `sh.status()` ကို အသုံးပြုရပါမယ်။

```
> sh.status();

--- Sharding Status ---
sharding version: { "_id" : 1, "version" : 3 }
shards:
{ "_id" : "shard0000", "host" : "db1.example.com:27017" }
{ "_id" : "shard0001", "host" : "db5.example.com:27017" }
{ "_id" : "shard0002", "host" : "db10.example.com:27017" }
databases:
{ "_id" : "admin", "partitioned" : false, "primary" : "config" }
{ "_id" : "mydb", "partitioned" : true, "primary" : "shard0000" }
... ...
```

နဲ့မူနာအရ shard0000, shard0001 နဲ့ shard0002 ဆိုတဲ့ Shard သုံးခုရှိနေပါတယ်။ အဲဒီထဲက shard0002 ကို ဖယ်ထဲတဲ့လိုရင် Query Router ပေါ်မှာ အခုလို ဆောင်ရွက်နိုင်ပါတယ်။

```
> use admin;
> db.runCommand( { removeShard: "shard0002" } );
{
  "msg" : "draining started successfully",
  "state" : "started",
  "shard" : "mongodb0",
  "ok" : 1
}
```

admin Database ပေါ်မှာ removeShard ကို Run ပေးလိုက်ခြင်းဖြစ်ပါတယ်။ နဲ့မူနာရလဒ်ရဲ့ msg မှာ "draining started successfully" ကိုတွေ့ရမှာဖြစ်ပါတယ်။ Data တွေကို အမြား Shard တွေပေါ်စဉ်နေပြီဆိုတဲ့ အမို့ယိုဖြစ်ပါတယ်။ ရွှေတဲ့လုပ်ငန်း ပြီးမပြီးသိရမိအတွက် removeShard ကို နောက်တစ်ကြိမ် ထပ် Run ကြည့်နိုင်ပါတယ်။

```
> use admin;
> db.runCommand( { removeShard: "shard0002" } );
{
  "msg" : "draining ongoing",
  "state" : "ongoing",
  "remaining" : {
    "chunks" : 42,
    "dbs" : 1
  },
  "ok" : 1
}
```

ဒီတစ်ကြိမ်မှတော့ msg တန်ဖိုးက "drainin ongoing" ဖြစ်သွားပါပြီ။ တန်ည်းအားဖြင့် Data တွေရွေနေဆဲဖြစ်တယ်ဆိုတဲ့အမို့ယိုဖြစ်ပါတယ်။ remaining.chunks မှာလည်း တန်ဖိုးတစ်ခုရှိနေတာကို တွေ့ရမှာဖြစ်ပါတယ်။ အခြေခံ အားဖြင့် remaining.chunks တန်ဖိုး 0 ဖြစ်သွားရင် Data ရွှေတဲ့လုပ်ငန်း ပြီးစီးပြီဖြစ်ပါတယ်။

Database တစ်ခုက Cluster ကနေ ဖယ်ထဲတဲ့ Shard ကို Primary Shard အဖြစ်အသုံးပြုထားတယ်ဆိုရင် အပြီးသတ် မဖယ်ထဲတဲ့ခင် အဲဒီပြဿနာကို ဖြေရှင်းပေးရပါမြို့မယ်။ Primary ဟုတ်မဟုတ် sh.status() ရလဒ်ကို လေ့လာကြည့်ရင် သိရနိုင်ပါတယ်။ sh.status() ရလဒ်မှာ အခုလိုပါဝင်နေနိုင်ပါတယ်။

```
{  "_id" : "somedb",  "partitioned" : true,  "primary" : "shard0002" }
```

အမိပိုယ်က somedb ဆိုတဲ့ Database ဟာ shard0002 ကို Primary Shard အနေနဲ့ အသုံးပြုထားတယ်ဆိုတဲ့ အမိပိုယ်ဖြစ်ပါတယ်။ ဒီလိုအခြေအနေမျိုးမှာ somedb ရဲ့ Primary ကို အရင်ပြောင်းပေးဖို့ လိုအပ်ပါတယ်။

```
> use admin;
> db.runCommand( { movePrimary: "somedb", to: "shard0000" } );
```

admin Database ပေါ်မှာ somedb ရဲ့ Primary အဖြစ် shard0000 ကို ပြောင်းသုံးဖို့ သတ်မှတ်ပေးလိုက်ခြင်း ဖြစ်ပါတယ်။ တစ်ခုသတိပြုရမှာကာ ဒီလုပ်ဆောင်ချက်ကို remaining.chunks တန်ဖိုး 0 ဖြစ်ပြီးမှသာ ဆောင်ရွက်ရမှာဖြစ်ပါတယ်။

Data တွေလည်း အခြား Shard ကိုပြောင်းပြီးပြီး လက်ရှိ Shard ကို Primary အဖြစ်အသုံးပြုထားတဲ့ Database တွေ ကိုလည်း Primary ပြောင်းပြီးပြီးဆိုရင် removeShard ကို နောက်တစ်ကြိမ် Run ခြင်းအားဖြင့် Shard ကို Cluster ထဲကနေ အပြီးသတ် ဖယ်ထုတ်နိုင်ပါတယ်။

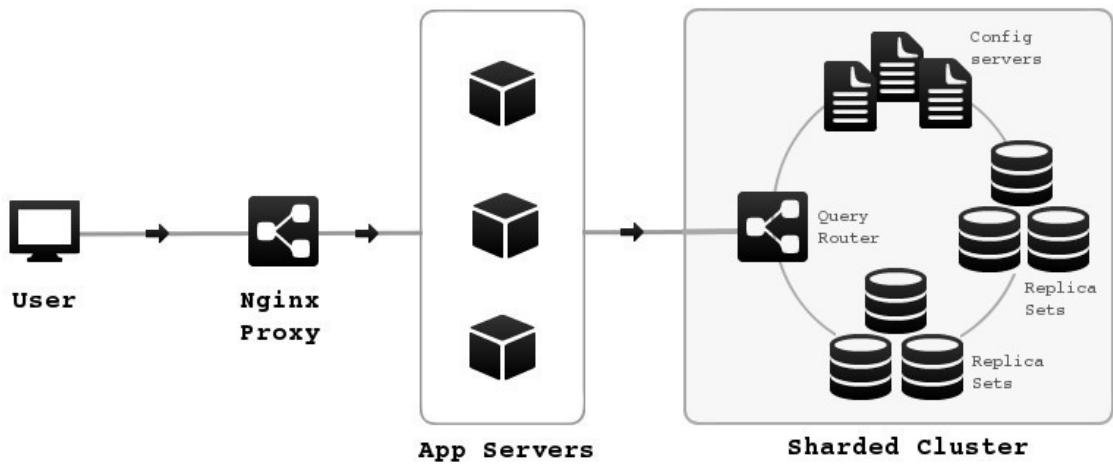
```
> use admin;
> db.runCommand( { removeShard: "shard0002" } );

{
  "msg" : "removeshard completed successfully",
  "state" : "completed",
  "shard" : "shard0002",
  "ok" : 1
}
```

ဒီတစ်ခါရရှိတဲ့ msq ကတော့ removeShard completed successfully ဖြစ်ပါတယ်။ ဒါဆိုရင် Shard ကို Cluster ထဲကနေဖယ်ထုတ်ခြင်းလုပ်ငန်း ပြီးစီးသွားပြီးပြီးဖြောင်းလုပ်ပါတယ်။

## Conclusion

ဒီအခန်းရဲ့အစမှာ Database Replication နဲ့ Sharding မပါဝင်သေးတဲ့အခါ ဖြစ်နိုင်ခြေရှိတဲ့ Architecture ပုံသဏ္ဌာန်ကို ဖော်ပြုခဲ့ပါတယ်။ အဲဒီ Architecture မှာ Replication နဲ့ Sharding ဖြည့်စွက်လိုက်တဲ့အခါမှာ ရရှိ လာနိုင်တဲ့ ပုံသဏ္ဌာန် ကတော့ အခုလိုဖြစ်မှာပါ။



ပုံ (၁၉.၅) - Scalable Architecture with Nginx and Sharded Cluster

ဒီနည်းနဲ့ အသုံးပြုသူများလာလို့ Process Power လိုအပ်လာတဲ့အခါ App Server တွေကို အလွယ်တစ်ကူထပ် ထည့်နိုင်စေမှာဖြစ်သလို၊ Storage Capacity တိုးမြှင့်ဖိုလိုအပ်လာတဲ့အခါ MongoDB Server တွေကို Sharded Cluster ထဲမှာ ထပ်ထည့်ခြင်းအားဖြင့် အလွယ်တစ်ကူ တိုးမြှင့်နိုင်မှာပဲဖြစ်ပါတယ်။ တနည်းအားဖြင့် အသုံးပြုသူများလာရင်များလာသလောက် Resource ပမာဏကို လိုက်လဲတိုးမြှင့်သွားနိုင်တဲ့ Scalable Architecture တစ်ခုကို ရရှိပြီပဲဖြစ်ပါတယ်။

MongoDB Sharded Cluster နဲ့ပက်သက်တဲ့ အကြောင်းအရာအသေးစိတ်ကို အောက်ပါလိပ်စာမှာ ဆက်လက် လေ့လာနိုင်ပါတယ်။

<http://docs.mongodb.org/manual/sharding/>

Front-end, Back-end, UI Design စသဖွင့် Software

Development မှာပါဝင်တဲ့ လုပ်ငန်းအားလုံးကို ဆောင်ရွက်

နိုင်စွမ်းရှိပြီး Server Architecture လို ကိစ္စမျိုးကိုပါနားလည်

တာဝန်ယူနိုင်သူကို Full-stack Developer လို ခေါ်ပါတယ်။

### **Professional Web Developer Course**

HTML5, PHP/MySQL, jQuery/Ajax, Mobile Web စသည့်

Professional Web Developer တစ်ခြီး သိရှိထားသင့်သည့်

နည်းပညာများကို စုစုပေါင်းသင်ကြားခြင်းဖြစ်သည်။

ဆက်သွယ်ရန် - (၁၉) ၇၃၀ ၆၅၉ ၆၂

## အခန်း(၂၀) – Common Server Architectures

ပုံမှန်အားဖြင့် Software တစ်ခု ရေးသားဖန်တီးရာမှာပါဝင်ရသူတွေကို သက်ဆိုင်ရာတာဝန်အလိုက် Programmer, Designer, Software Engineer, Web Master, Front-end Developer, Back-end Developer, Database Administrator, System Administrator, Network Engineer စသဖြင့်အမျိုးမျိုး ခေါ်ကြပါတယ်။ အခုန်ကိုပိုင်းမှာတော့ Full-stack Developer ဆိုတဲ့ အသုံးအနှံးတစ်မျိုး ခေတ်စားလာခဲ့ပါတယ်။

Full-stack Developer ဆိုတာ Software စတင်ရေးသားခြင်းကနေ နောက်ဆုံး အများအသုံးပြုနိုင်အောင် Release လုပ်တဲ့အထိ လိုအပ်တဲ့လုပ်ငန်းစဉ်တိုင်းကို တာဝန်ယူအလုပ်လုပ်နိုင်တဲ့ စွဲယုံရဲ Developer ကို ခေါ်ခြင်းဖြစ်ပါတယ်။ Code ရေးသားခြင်းကို အမိမကျမှုများကျင်သူဖြစ်နိုင်ပေးမယ့်၊ လိုအပ်ရင် UI Design တွေ ရေးဆွဲနိုင်သူလည်းဖြစ်ပါတယ်။ Data Structure နဲ့ System Design တွေကိုလည်း ရေးဆွဲနိုင်သူဖြစ်ပါတယ်။ Client App တွေရေးသားနိုင်သလို၊ Back-end Service တွေလည်း ရေးသားနိုင်သူဖြစ်ပါတယ်။ Server Operating System နဲ့ Server Architecture ကို စီမံတပ်ဆင်နိုင်သူဖြစ်ပါတယ်။ ဒီလို Software Development Process မှာ ပါဝင်တဲ့ လုပ်ငန်းတိုင်းကို ကိုယ်တိုင်ဆောင် ရွက်နိုင်သူကို Full-stack Developer လိုအပ်တယ်။ Programming Language အနေနဲ့လည်း နှစ်မျိုး သုံးမျိုး အသုံးပြု ရေးသားနိုင်ပြီး၊ အခြားဆက်စပ်နည်းပညာတွေနဲ့လည်း ထိုက်သင့်သလောက် ရင်းနှီးကျမှုများဝင် အသုံးချုနိုင်သူဖြစ်ပါတယ်။

သင်ကြားရေးတဲ့ လေ့လာရေးမှာ Full-stack Developer ဖြစ်အောင်လေ့လာခြင်းနဲ့ သက်ဆိုင်ရာဘာသာရပ်တစ်ခုမှာ အထူးပြုကျမှုများကျင်တဲ့ Specialist ဖြစ်အောင်လေ့လာခြင်းတို့ကြား၊ တစ်ဦးနဲ့တစ်ဦး အယူအဆမတူအငြင်းပွားမှုတစ်ခုရှိ ကြပါတယ်။ Software Development Team တွေမှာလဲ၊ Specialist တွေကပိုစိရောက်တယ်၊ Full-stack Developer တွေက ပိုစိရောက်တယ်ဆိုတဲ့ အငြင်းအခံတွေ ရှိကြပါတယ်။ ဒါပေမယ့် အခြေအနေအရပ်ရပ်ကြောင့် ကနောက်မှုမှာ Full-stack Developer တွေ ပိုမိုလိုအပ်လာတယ်ဆိုတဲ့ အချက်ကတော့ ရှောင်လွှဲလိုမရပါဘူး။

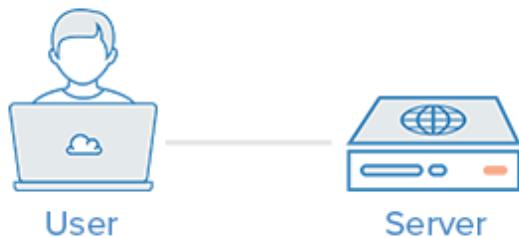
Engineer တွေ၊ Designer တွေနဲ့ Developer အင်အား၊ ရာထောင်ချို့ရှိတဲ့ Software ကုမ္ပဏီကြီးတွေမှာ သူနေရာနဲ့သူ ကျမှုများကျင်တဲ့ Specialist တွေ ခန့်ထားပြီး လုပ်ငန်းဆောင်ရွက်နိုင်ပေးမယ့်၊ Startup နဲ့ Software Team လေးတွေမှာ တော့ ကဏ္ဍတိုင်းအတွက် Specialist တွေကို ခန့်အပ်တာဝန်ပေးနိုင်မှာမဟုတ်ပါဘူး။ ဒါကြောင့် Developer တစ်ဦးအနေ နဲ့ လုပ်ငန်းတာဝန်နှစ်မျိုးသုံးမျိုးကို တာဝန်ယူရတက်ပါတယ်။ ကျွန်ုတော်တို့ဆိုမှာ ပိုဆိုပါတယ်။ သက်ဆိုင်ရာကဏ္ဍမှာ ကျမှုများကျင်တဲ့ Specialist ဆိုတာ အမြဲမရှိနိုင်ပါတယ်။ ၁၀၀% Full-stack Developer မဟုတ်ရင်တောင်မှ၊ Client App ရေား၊ Back-end Service ကိုပါ ရေးနိုင်သူတစ်ဦးဖြစ်ဖို့ လိုအပ်တက်ပါတယ်။ ရေးသားထားတဲ့ Software နဲ့ Service တွေကို Host လုပ်မယ့် Server Architecture ကို Developer ကိုယ်တို့ စီမံဖို့လိုနိုင်ပါတယ်။

ပြီးခဲ့တဲ့အခန်းတွေမှာ Docker, Nginx, Sharded Cluster စတဲ့နည်းပညာတွေကို ဖော်ပြခဲ့ရာမှာလည်း Developer တွေ ကိုယ်တိုင် Server Architecture တွေတည်ဆောက်ရာမှာ အထောက်အကူဖြစ်စေဖို့ဆိုတဲ့ ရည်ရွယ်ချက်နဲ့ပဲ ဖော်ပြခဲ့ခြင်း ဖြစ်ပါတယ်။ ဒါပေမယ့် ဒီနည်းပညာတွေကို အမြဲအသုံးပြုရမယ်ဆိုတဲ့ သဘော တော့လည်း မဟုတ်ပါဘူး။ ကိုယ်ရဲ့ လိုအပ် ချက်နဲ့ချင့်ချိန်အသုံးပြုရမှာဖြစ်ပါတယ်။ ဒီအခန်းမှာတော့ Server Architecture အမျိုးမျိုးနဲ့ သူတို့၏ အားသာချက် အားနည်းချက်တွေကို အကျဉ်းချုပ် ဖော်ပြသွားမှာပဲဖြစ်ပါတယ်။

## 20.1 – Everything on One Server

Server Architecture တွေထဲမှာ အခြေခံအကျဆုံးနဲ့ အရှိုးရှင်းဆုံးနည်းစနစ်ကတော့ လိုအပ်တဲ့ Service အားလုံး ကို Server တစ်ခုတည်းမှာ စုစုပေါင်းစုစုပေါင်းခြင်းပဲဖြစ်ပါတယ်။ ရှုံးခိုင်းမှာ အသုံးပြုဖော်ပြခဲ့တဲ့ နည်းပညာတွေအရ ဆိုရင် Nginx, MongoDB နဲ့ ExpressJS API Server အားလုံးကို Server ကွန်ပျူးတာ တစ်လုံးတည်းမှာ စုစုပေါင်းစုစုပေါင်းခြင်းပဲ ဖြစ်ပါတယ်။

### Single Server



ပုံ (၂၀.၁) - Single Server Setup

source - [digitalocean.com](https://www.digitalocean.com/community/tutorials/computing-basics-1-single-server)

ဒီနည်းကပေးတဲ့ အားသာချက်ကတော့ Setup လုပ်ရတာ အလွန်ရှိုးရှင်းလွယ်ကူခြင်းပဲ ဖြစ်ပါတယ်။ နောက် အားသာချက် တစ်ခုကတော့ ကုန်ကျစရိတ်သက်သာခြင်း ဖြစ်ပါတယ်။ Server ကွန်ပျူးတာတစ်လုံးပဲ လိုအပ်မှာ ဖြစ်လို့ ကိုယ်တိုင် Server Hardware ဝယ်ယူမယ်ဆိုရင်လည်း ကုန်ကျစရိတ်သက်သာသလို့ VPS (သို့မဟုတ်) အခြား Hosting Service တွေ ဝယ်ယူအသုံးပြုမယ်ဆိုရင်လည်း စရိတ်စကသက်သာမှာဖြစ်ပါတယ်။ ဒါပေမယ့် နောင်အသုံးပြုသူများလာတဲ့အခါ ခံနိုင်ရည် ရှိစေဖို့ရည်ရွယ်ပြီး လက်ရှိမလိုအပ်သေးပေမယ့်၊ လိုတာထောက်ပို ကောင်းတဲ့ Hardware Server ကို ကြိုးတင်တပ်ဆင်ထား ရတာမျိုးတော့ ဖြစ်နိုင်ပါတယ်။

ဒီနည်းရဲ့ အဓိကအကျဆုံး အားနည်းချက်ကတော့ Scalable မဖြစ်ခြင်းပဲဖြစ်ပါတယ်။ နောင်အသုံးပြုသူများလာ လို့ RAM (သို့မဟုတ်) Storage Capacity တိုးမြှင့်ဖို့လိုအပ်လာတဲ့အခါ လက်ရှိ Server မှာပဲ RAM နဲ့ HDD တွေ ထပ်တပ် ခြင်းအားဖြင့် လိုအပ်တဲ့ Capacity ရအောင် Upgrade လုပ်ရတော့မှာပါ။ ဒီနည်းကို Scale Up လုပ်တယ်လို့ ခေါ်ပါတယ်။ Vertical Scaling လိုလည်း ခေါ်ပါတယ်။ Scale Up လုပ်တာဟာ အကန်အသတ်ရှိပါတယ်။ ဥပမာ - Hardware ကန်သတ်ချက်အရ 16 GB မေးကွန်ထဲပဲ တိုးလိုရတာမျိုးဖြစ်နိုင်ပါတယ်။ ဒီထက်ပိုလိုလာရင် ထပ်တိုးလို့ ရမှာမဟုတ်တော့ပါဘူး။ အလားတူပဲ Server OS ကန်သတ်ချက်အရ

HDD ပမာဏ အမြင့်ဆုံး 16TB ထို့ လက်ခံ အလုပ်လုပ်ပေးနိုင်တဲ့ အကန့်အသတ်မျိုးတွေ ရှိတက်ပါတယ်။ ဒါ ထက်ပိုလိုအပ်လာရင် အဆင်ပြုမှာမဟုတ်တော့ပါဘူး။

ဒါကြောင့် ကန်သတ်ပမာဏတစ်ခုထက် Resource လိုအပ်ချက် အချမလိုသလို နောင်လည်း လိုလာစရာမရှိဘူး ဆိုတဲ့ App မျိုးတွေမှာသာ ဒီနည်းကိုအသုံးပြုသင့်ပါတယ်။ နောင်တစ်ချိန်မှာ Scale လုပ်ဖို့လိုအပ်နိုင်တဲ့ App တွေမှာဆိုရင်တော့ ဒီနည်းကို မသုံးသင့်ပါဘူး။

## 20.2 – Separate Database Server

နောက်တစ်နည်းကတော့ Database Server ကို သီးခြား Server ကွန်ပျူတာတစ်လုံးနဲ့ ခွဲထုတ်တပ်ဆင်တဲ့နည်း ဖြစ်ပါ တယ်။ ကျွန်ုတ်တို့ ရော့ပိုင်းမှာ ဖော်ပြခဲ့တဲ့ နည်းပညာတွေအရဆိုရင် Nginx Web Server နဲ့ ExpressJS API Server ကို Server တစ်လုံးနဲ့ထားပြီး MongoDB ကို သီးခြား Server တစ်လုံးနဲ့ထားရှိရမှာ ဖြစ်ပါတယ်။

### Separate Database Server



ပုံ (၂၀.၂) - Separate Database Server Setup

source - [digitalocean.com](https://www.digitalocean.com/community/tutorials/separate-database-server)

ဒီနည်းရဲ့ အားသာချက်ကတော့ Resource အသုံးပြုမှုကို Server နှစ်လုံးခဲ့ပြီး မျှလိုက်တဲ့အတွက် Server တစ်လုံး တည်း အသုံးပြုတာထက် ပိုမိုထိရောက် မြန်ဆန်သွားမှာဖြစ်ပါတယ်။ နောင်လိုအပ်လို့ Scale လုပ်ရမယ်ဆိုရင် လည်း App Server ကို သပ်သပ်၊ Database Server ကိုသပ်သပ် Scale Up လုပ်သွားနိုင်မှာဖြစ်ပါတယ်။ Server နှစ်လုံးဖြစ်သွားတာနဲ့အညီ ရရှိနိုင်တဲ့ အမြင့်ဆုံး Capacity လည်း နှစ်ဆမြင့်တက်သွားမှာပဲဖြစ်ပါတယ်။ ဒါ အပြင် Data Storage လိုရင် Database Server ကိုသာ Scale Up လုပ်ပြီး၊ App Server ကို မလိုအပ်ရင်ချိန်ထားနိုင်တဲ့အားသာချက်မျိုးလည်း ရရှိနိုင်ပါသေး တယ်။

နောက်အားသာချက်တစ်ခုကတော့ App Server ကိုသာ Public Internet ကနေ တိုက်ရှိက်အသုံးပြုခွင့်ပေးပြီး Database Server ကိုတော့ Public Internet ကနေ Access လုပ်ခွင့်မပေးပဲ၊ Private Network ထဲမှာသာထားပြီး၊ App Server ကိုသာ Access လုပ်ခွင့် ပြုထားနိုင်တဲ့အတွက် Database Server လုပ်ခြေားအတွက်လည်း ပိုကောင်းသွားမှာ ဖြစ်ပါတယ်။

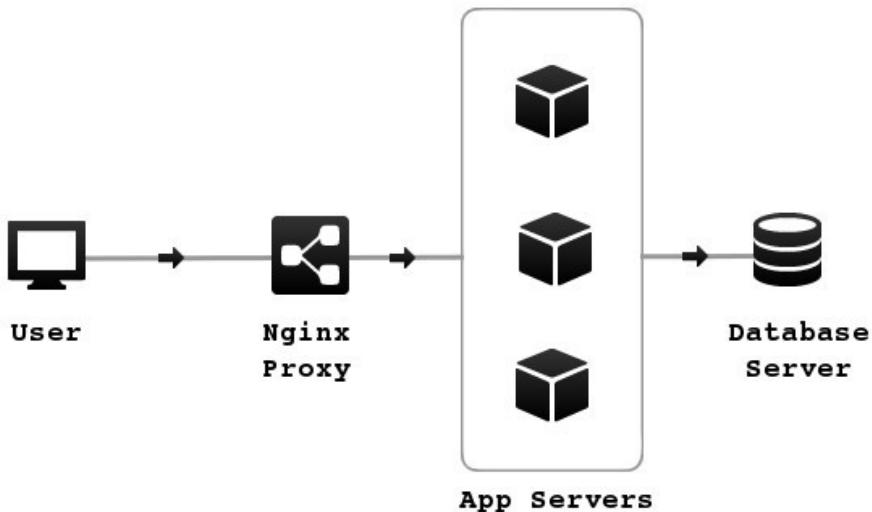
သူလည်းပဲ Setup လုပ်ရတာ ဘာမှမခက်ခဲပါဘူး။ ရိုးရှင်းလွယ်ကူပါတယ်။ ဒါပေမယ့် ဒီနည်းမှာလည်း

Scalability ပိုင်းမှာ အကန်အသတ်ရှိနော်မှာဖြစ်ပါတယ်။ တော်ယုံကန်ယုံ App တွေအတွက် Server တစ်လုံးတည်းထားခြင်းထက် ဒီနည်းနဲ့ Database Server ကို သီးခြားခဲ့ထားခြင်းက ပိုမိုသင့်တော်မှာဖြစ်ပါတယ်။

### 20.3 – Load Balancer (Reverse Proxy)

နောက်တစ်နည်းကတော့ Load Balancer အသုံးပြုခြင်းပဲဖြစ်ပါတယ်။ အခန်း (၁၈) မှာ Nginx ကို သုံးပြီး Load Balance ပြုလုပ်နိုင်ပုံကို ဖော်ပြခဲ့ပြီးဖြစ်ပါတယ်။ Load Balancer အသုံးပြုရခြင်းရည်ရွယ်ချက်ကတော့ App Server တစ်လုံးထက်ပိုပြီး အသုံးပြုနိုင်ဖို့ဖြစ်ပါတယ်။ ဒီနည်းနဲ့ နောင်လိုအပ်လို့ App Server ရဲ့ စွမ်းဆောင်ရည် မြှင့်တင်ရတော့ မယ်ဆိုရင် လက်ရှိ Server မှာ RAM တွေ၊ HDD တွေ သွားတိုးနေစရာမလိုပဲ၊ နောက်ထပ် Server တစ်လုံးထပ်တိုးလိုက် ခြင်းအားဖြင့် Scale Out ပြုလုပ်သွားနိုင်မှာဖြစ်ပါတယ်။ ဒီနည်းကို Horizontal Scaling လိုလည်း ခေါ်ပါတယ်။

ဒီနည်းရဲ့အားသာချက်ကတော့ Scale လုပ်ရလွယ်ခြင်းဖြစ်ပါတယ်။ Scale လုပ်ရလွယ်တဲ့အတွက်အစပထမပိုင်း မှာ Server Hardware တွေကို လိုတာထက်ပိုပြီး တပ်ဆင်ထားရတာမျိုး လုပ်စရာမလိုတော့ပါဘူး။ နောင်လိုအပ်လာတော့မှ လို သလောက်ကိုသာသီးခြား Server အနေနဲ့ ထပ်တိုးနိုင်တဲ့အတွက် လက်ရှိအမှန်တစ်ကယ်လိုအပ်သလောက်ကိုသာ တပ်ဆင်ထားဖို့လိုတဲ့ အားသာချက်ရှိပါတယ်။



ပုံ (၂၀.၃) - Load Balancer Setup

ဒီနည်းကပေးတဲ့ နောက်ထပ်အားသာချက်ကတော့ အခန်း (၁၈) မှာလည်း ဖော်ပြခဲ့ပြီးဖြစ်တဲ့ Security ဖြစ်ပါတယ်။ မလို လားအပ်တဲ့ Request တွေကို App Server ထံရောက်ဖို့မလိုပဲ ကြေားခဲ့ Load Balancer Proxy ကနေ စီစစ်နိုင်တဲ့ အား သာချက်ကို ဖြည့်စွက်ရရှိမှာဖြစ်ပါတယ်။ ဒါအပြင် Cache လုပ်ဆောင်ချက်အကူအညီနဲ့ တစ်ချို့ Request တွေကို App Server ထံရောက်စရာမလိုပဲ Proxy က အကြောင်းပြန်ပေးအောင် စီမံထားနိုင်ပုံကိုလဲ တွေ့မြင်ခဲ့ရပြီးဖြစ်ပါတယ်။

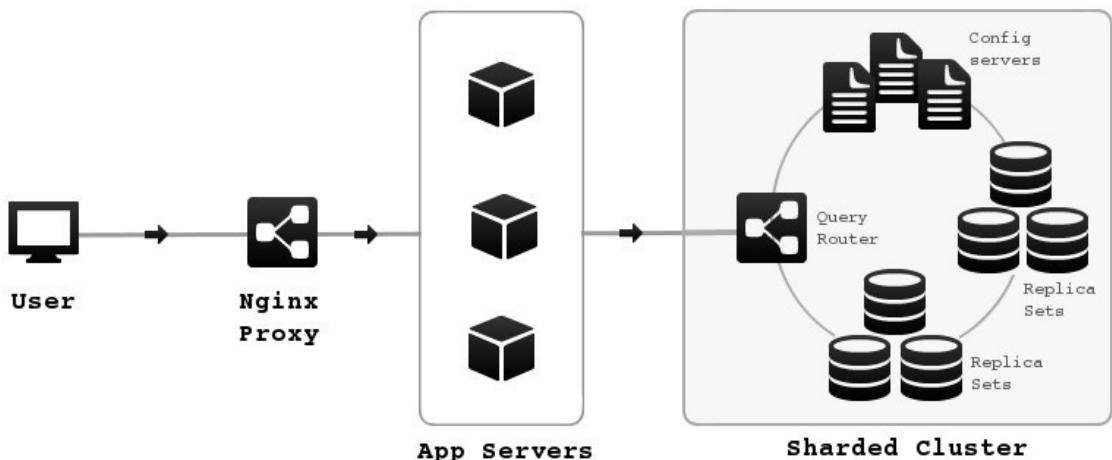
အားနည်းချက်အနေနဲ့ကတော့ တပ်ဆင်စီမံရ အနည်းငယ် ခက်ခဲနိုင်ပါတယ်။ Load Balancer နဲ့ App Server

တွေကြား Session တွေကို ဘယ်လိုစီမံမလဲ၊ SSL Encryption လုပ်ငန်းအဆင်ပြေအောင် ဘယ်လို Setup လုပ်ရမလဲ၊ စတဲ့ ဖြည့်စွက်စဉ်းစားရမှာတွေ၊ ဖြည့်စွက်စီမံရတာတွေ ရှိလာမှာပဲဖြစ်ပါတယ်။ ဒါအပြင် Resource လိုလာတိုင်း Server အသစ်တစ်လုံးထပ်တိုးရတယ်ဆိုတဲ့စရိတ်က လိုအပ်တဲ့ RAM နဲ့ HDD ပဲ ရွေးပြီးတိုးရတဲ့စရိတ်ထက် ပိုများမှာဖြစ်ပါ တယ်။ အကန်းအသတ်မရှိဘဲလောက် Scale လုပ်သွားနိုင်တဲ့ အတွက်ရေရှည်မှာ စရိတ်သတ်သာသွား လိမ့်မယ်လို့ ဆိုနိုင် ပေမယ့် အစပိုင်းမှာတော့ အနည်းငယ် ပိုမိုကုန်ကျမှာဖြစ်ပါတယ်။

အသုံးပြုသူပမာဏ ခန်းမျိုးနိုင်ခြင်းမရှိပဲ လိုအပ်သလို Scale လုပ်ဖို့လိုအပ်တဲ့ App တွေအတွက် ဒီနည်းစနစ်က သင့်တော် ပါတယ်။

## 20.4 – Sharded Cluster

ပြီးခဲ့တဲ့အခန်းမှာ Sharded Cluster အကူအညီနဲ့ Database Server ကိုလည်း Scale လုပ်နိုင်ပုံကို ဖော်ပြုခဲ့ပြီး ဖြစ်ပါ တယ်။ Load Balancer နဲ့ Shard Cluster လုပ်ဆောင်ချက် အကူအညီနဲ့ အပြည့်အဝ Scalable ဖြစ်တဲ့ ပိုမိုပြည့်စုံတဲ့ Server Architecture ကို တပ်ဆင်ထားနိုင်မှာဖြစ်ပါတယ်။



ပုံ (၂၀.၄) - Load Balancer and Sharded Cluster

ဒီနည်းစနစ်ကတော့ App Server ကိုရော့၊ Database Server ကိုပါ Scale Out လုပ်နိုင်မှာ ဖြစ်တဲ့အတွက် ရေရှည်မှာ အပြည့်အဝ Scalable ဖြစ်လိုတယ်ဆိုရင် သုံးရမယ့် နည်းစနစ်ဖြစ်ပါတယ်။ ဒါပေမယ့် တပ်ဆင်စီမံရတာ ရှိရှင်းလွယ်ကူမှာ မဟုတ်ပါဘူး။

Shard Server တွေ တိုးတဲ့လျော့တဲ့အခါး၊ Hardware အပြောင်းအလဲပြုလုပ်လိုတဲ့အခါး Data တွေ ဆုံးရှုံးမှုမ ရှိအောင် Migrate လုပ်တဲ့ လုပ်ငန်းစဉ်ကိုစီမံရပါမယ်။ Shard Server အဖြစ်သုံးထားတဲ့ Replica Sets တစ်ခုလုံး၊ အမျိုးမျိုးကြောင့် အလုပ်မလုပ်တဲ့အခါး ဘယ်လိုလုပ်မလဲဆိုတာ စဉ်းစားရပါမယ်။ Replica Sets တွေကနေ Data Read လုပ်တဲ့အခါး အပြည့်အဝ Sync မလုပ်ရသေးတဲ့ Secondary Server ကနေ Read လုပ်မိလို့ မပြည့်စုံတဲ့ Data ပြန် ရနေတာမျိုး မဖြစ်အောင် စီမံရပါတော့မယ်။ MongoDB က ဒီလိုပြဿနာတွေ

ကို ပြေလည်အောင် ဖြေရှင်းရမယ့် နည်းစနစ်တွေ ပေးထားပေမယ့်၊ အတိုင်းအတာတစ်ခုထိ ခက်ခဲမှုတွေ ကတော့ ရှိလာမှာပဲဖြစ်ပါတယ်။

နောက်အားနည်းချက် တစ်ခုကတော့ Server တွေပိုလိုတဲ့အတွက် စတင်ပေါင်ဆင်ခ စရိတ်ကြီးနှင့်ပါတယ်။ မလိုအပ်ပဲနဲ့ တော့ ဒီနည်းကို မသုံးသင့်ပါဘူး။ အသုံးပြုသူ သိန်းသန်းချီးပြီး ရှိလာနှင့်စရာရှိတဲ့ App တွေမှာဆိုရင် တော့ ဒီနည်းကိုပဲ အသုံးပြုကြရမှာပါ။

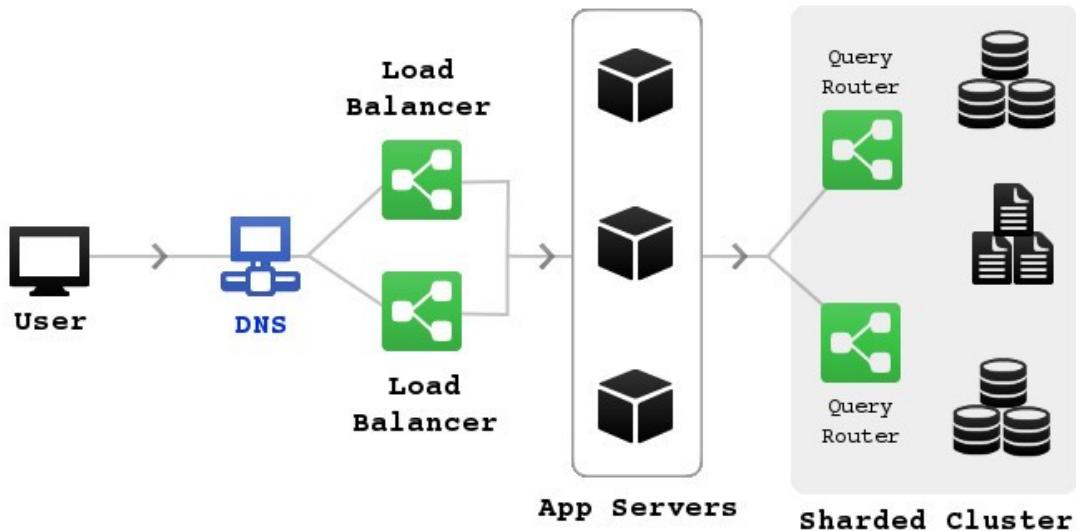
## 20.5 – Multiple Load Balancers and Query Routers

အသုံးပြုသူသိန်းချီးရှိလာတဲ့အခါ ဖြစ်လာနှင့်တဲ့ပြဿနာ ရှိပါသေးတယ်။ အဲဒါကတော့ Load Balancer ကိုသုံးပြီး App ကို Scale လုပ်လိုရအောင် စီမံထားပေမယ့် Load Balancer ကိုယ်တိုင်က များပြားလွန်းလှတဲ့ Request တွေကို လက်ခ စီမံနှင့်ခြင်းမရှိတော့တာမျိုး၊ ဖြစ်လာနှင့်ပါတယ်။ ဒီလိုအခြေအနေမှာတော့ Load Balancer တွေကို ထပ်တိုးဖို့ လိုအပ်လာမှာဖြစ်ပါတယ်။ အလားတူပဲ Sharded Cluster မှာလည်း Data Request များလို့ Query Router တစ်ခု တည်းက လက်ခ စီမံနှင့်ခြင်းမရှိတဲ့ အခြေအနေမှာ Query Router တွေ ထပ်တိုး ပေးဖို့ လိုအပ်မှာဖြစ်ပါတယ်။

Load Balancer တစ်ခုထက်ပိုအသုံးပြုရတဲ့အခါ သူတို့ရဲ့ရှေ့က ထပ်ဆင့် Load Balancer သဘောမျိုး ဝင်ရောက်လာတာ ကတော့ DNS Server ဖြစ်ပါတယ်။ DNS Server ရဲ့ အခြေခံတာဝန်က Domain Name တွေရဲ့ IP Address ကို Resolve လုပ်ပေးနိုင်ဖို့ ဖြစ်ပါတယ်။ ဒါပေမယ့် Domain Name တစ်ခုအတွက် IP Address နှစ်ခုသုံး ခု သတ်မှတ်ထား မယ်ဆိုရင် DNS Server ကလည်း Round Robin Algorithm ကိုသုံးပြီး IP Address တွေကို အလှည့်ကျ ပြန်ပေးမှာ ဖြစ်ပါတယ်။

ဥပမာ – example.com Domain Name အတွက် 123.45.67.89 ဆိုတဲ့ IP Address နဲ့ 123.45.67.99 ဆိုတဲ့ IP Address နှစ်ခုသတ်မှတ်ထားတယ်ဆိုရင်၊ DNS Server က DNS Lookup Request တစ်ခုကို လက်ခ ရရှိတဲ့အခါ ပထမ အကြံမှာ 123.45.67.89, 123.45.67.99 လို့ အကြောင်းပြန်ပေးမှာဖြစ်ပြီး နောက်တစ်ကြံမှာ 123.45.67.99, 123.45.67.89 လို့ အကြောင်းပြန်ပေးမှာဖြစ်ပါတယ်။ သက်ဆိုင်ရာ Domain Name အတွက် IP Address စာရင်းကို အကြောင်းပြန်တဲ့အခါ Round Robin နဲ့ အလှည့်ကျပြီး ပြန်ပေးမှာဖြစ်ပါတယ်။

ဒါကြောင့် Load Balancer တွေကို ကိုယ်ပိုင် IP Address တွေနဲ့ Setup လုပ်ထားပြီး DNS မှာ အဲဒီ IP Address စာရင်းကိုပေးလိုက်မယ်ဆိုရင် DNS Server က Load Balancer တစ်ခုသွေးယူ ရှေ့ကနေ ဆောင်ရွက်ပေးနေမှာပဲ ဖြစ်ပါ တယ်။



ပုံ (၂၀.၅) - Multiple Load Balancers

ဒီနည်းကတော့ ပိုမြီးပြည့်စုံလာတာနဲ့အမျှ တပ်ဆင်စီမံရတာလည်း ပိုမြီးရှုပ်ထွေးခက်ခဲလာဖြိုဖြစ်ပါတယ်။ DNS Server ကို Load Balancer အနေနဲ့အသုံးပြုတဲ့အခါ သတိပြုရမယ့်အချက်တစ်ချို့ ရှိပါတယ်။ DNS Server ဟာ IP Address စာရင်းကို ပေးတဲ့အခါ အရှုံးအတိုင်းပေးမှာဖြစ်ပါတယ်။ သက်ဆိုင်ရာ IP Address မှာ Setup လုပ်ထားတဲ့ Host တွေ အလုပ်မှုလုပ်ရဲလား စစ်ဆေးနေမှာ မဟုတ်ပါဘူး။ Nginx Load Balancer ကတော့ App Server တစ်ခု Down နေရင် အခြား App Server တစ်ခုကို ပြောင်းလဲအလုပ်လုပ်ပေးနိုင်ပေးမယ့်၊ DNS ကတော့ အဲဒီလိုလုပ်ပေးမှာ မဟုတ်ပါဘူး။ သတ်မှတ်ထားတဲ့ IP Address ကိုသာ မူသေပြန်ပေးနေမှာဖြစ်လို့ Load Balancer တွေအားလုံး အလုပ်လုပ်နေကြောင်း သေချာစေဖို့နဲ့ အကြောင်းအမျိုးမျိုးကြောင့် Load Balancer တစ်ခု အလုပ်မလုပ်တဲ့အခါ ဘယ်လိုစီမံမလဲဆိုတဲ့ ကိစ္စမျိုးတွေ ကြိုက်ပြင်ဆင်ထားရမှာ ဖြစ်ပါတယ်။

DNS ရဲနောက်ပြသသနာတစ်ခုကတော့ Cache ဖြစ်ပါတယ်။ Web Browser လို့ Client တွေရော ကြားခဲ့ DNS Resolver တွေကပါ IP Address စာရင်းကို Cache လုပ်ထားတက်ကြပါတယ်။ ဒီနည်းနဲ့ Domain Name တစ်ခုရဲ့ IP Address ကို သိရဖို့အတွက် ထပ်ခါထပ်ခါ DNS Resolve လုပ်နေစရာမလိုတော့ပဲ စွမ်းဆောင်ရည် ပိုကောင်းစေဖို့ဖြစ်ပါတယ်။ ဒါပေမယ့် ဒီနည်းကြောင့် DNS Server မှာ IP Address တွေထပ်မံတိုးပြည့်ခြင်း၊ ပြင်ဆင်ခြင်း လုပ်ငန်းတွေ လုပ်လိုက်ရင် ချက်ခြင်းသက်ရောက်မှာမဟုတ်ပါဘူး။ Browser နဲ့ Resolver တွေက သတ်မှတ်ကာလတစ်ခုအထိ ပြင်ဆင်မှု Update ကို မရယူပဲသူတိမှာရှိတဲ့ Cache ကိုသာဆက်လက်အသုံးပြုနေမှာဖြစ်ပါတယ်။ ဒီအချက်ကိုလည်း ထည့်သွင်း စဉ်းစားရတော့မှာဖြစ်ပါတယ်။

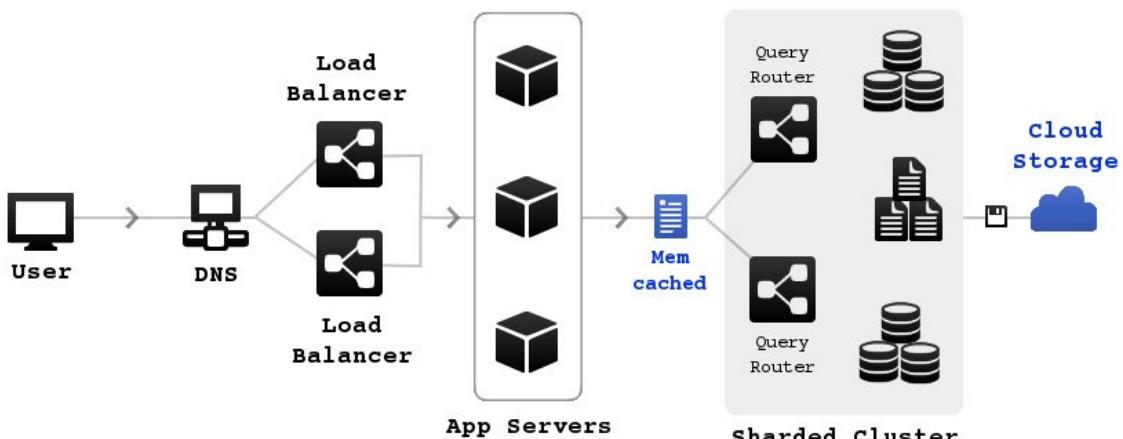
Sharded Cluster မှာ လိုအပ်လို့ Query Router တွေ တိုးတဲ့အခါမှာလည်း အကန်းအသတ်တစ်ခုရှိပါတယ်။ MongoDB ရဲ့ ကန်းသတ်ချက်အရ Client တစ်ခုဟာ Query Router တွေကို အလုပ်ကျသုံးလို့ မရပါဘူး။ Query Router တစ်ခုကိုသာ အမြဲသုံးရမှာဖြစ်ပါတယ်။ ဥပမာ - App Server (၃) ခု ထားတယ်ဆိုရင် App Server A က Query Router 1 ကို အသုံးပြုတယ်ဆိုရင် နောင်ကိုလည်း Query Router 1 ကိုပဲ အသုံးပြုရမှာဖြစ်ပါတယ်။ တနည်းအားဖြင့် Automatic Load Balance လုပ်လို့ အဆင်မပြေတော့ပဲ ဘယ် App Server က ဘယ် Query

Router ကို သုံးရမယ်လို့ ကြိုတင်သတ်မှတ်ထားရမှာဖြစ်ပါတယ်။ ဒါကြောင့် ပြဿနာတစ်စုံတစ်ရာရှိလာတဲ့ အခါ Debug လုပ်အဖြော်ရတာ ထင်သလောက် လွယ်မှာမဟုတ်တော့ပါဘူး။ Code ကြောင့်လည်း ဖြစ်နိုင်ပါတယ်။ Load Balancer ကြောင့်လည်း ဖြစ်နိုင်ပါတယ်။ Query Router ကြောင့်လည်း ဖြစ်နိုင်ပါတယ်။ Sharded Cluster Configuration ကြောင့်လည်း ဖြစ်နိုင်ပါတယ်။ Architecture ရှုပ်ထွေးလာတာနဲ့အမျှ စီမံရတာလည်း လိုက်လဲ ရှုပ်ထွေးခက်ခဲလာမှာပဲ ဖြစ်ပါတယ်။ ဒါကြောင့်လည်း Software Team ကိုတွေမှာ ဒီလို Architecture ပိုင်း စီမံသူတွေကို သီးခြားခန့်အပ်တာဝန်ပေးထားလေ့ ရှိကြတာပဲ ဖြစ်ပါတယ်။

## 20.6 – Memcached & Cloud Storage

Server Architecture စဉ်းစားတဲ့အခါ စီမံရလွယ်ကူမှာ ကုန်ကျစရိတ်နဲ့ ရရှိနိုင်တဲ့ Performance ကို ချိန်ညိုကြရ ပါတယ်။ ပိုကောင်းတဲ့ Performance ကိုလိုချင်တဲ့အခါ စီမံရခက်ခဲလာပြီး ကုန်ကျစရိတ်များလာနိုင်ပါတယ်။ ကုန်ကျစရိတ်သက်သာ ပြီး စီမံရလွယ်ကူတဲ့အခါ အမြင့်ဆုံးစွမ်းဆောင်ရည်ကို ရရှိချင်မှရရှိပါလိမ့်မယ်။

Server Architecture ရဲ့ စွမ်းဆောင်ရည် ပိုကောင်းစေဖို့အတွက် Memcached ကိုလည်း ထည့်သွင်းဖြည့်စွက်နိုင်ပါသေး တယ်။ Memcached ရဲ့ လုပ်ဆောင်ချက်လိုဂုဏ်းကတော့၊ App Server ကို Cache Proxy နဲ့ Cache လုပ်သလိုပဲ Database Server ကို Memcached နဲ့ Cache လုပ်နိုင်ခြင်းဖြစ်ပါတယ်။ အသုံးများတဲ့ Data တွေကို Memcached Server ရဲ့ Memory ပေါ်မှာ တင်ထားလိုက်ခြင်းအားဖြင့် ဆက်သွယ်ရယူရတာ ပိုမြန်သွားသလို၊ အချက်အလက်တစ်ခု လိုချင်တိုင်း Database Server ကို အမြတ်စွာသွားယူနစ်ရာမလိုတော့တဲ့အတွက် စွမ်းဆောင်ရည် ပိုကောင်းသွားမှာပဲ ဖြစ်ပါတယ်။



ပုံ (၂၀.၆) - Memcached and Cloud Backup in Architecture

ပုံ (၂၀.၆) ကိုလေ့လာကြည့်ရင် App Server နဲ့ Sharded Cluster ကြားမှာ Memcached ကို ကြားခံပေးထားတာ ကို တွေ့ရမှာဖြစ်ပါတယ်။

နောက်ထပ်ထပ်မဖြည့်စွက်သင့်တဲ့အချက်ကတော့ Backup ပဲဖြစ်ပါတယ်။ Backup ဆိတာ စနစ်ကြီးတစ်ခုလုံး ကို လုပ်စရာ လိုလေ့မရှိပါဘူး။ Database ထဲက Data တွေကိုသာ Backup လုပ်ကြရတာပါ။ Database ထဲက Data တွေကို ကာလ တစ်ခုသတ်မှတ်ပြီး Export ထုတ်ယူကာ အလိုအလျောက် သိမ်းဆည်းအောင် စီမံထားသင့်

ပါတယ်။ ကာလတစ်ခုဆိုတာ တစ်နောက်ကြိမ်လည်းဖြစ်နိုင်ပါတယ်။ (၄) နာရီခြား တစ်ကြိမ်လည်း ဖြစ်နိုင်ပါတယ်။ ကိုယ့် App ရဲ့ Data အထုတ် အသွင်း ပြုလုပ်မှု Frequency ပေါ်မှုတည်ပြီး ပုံမှန် Backup Routine ကာလတစ်ခုကိုသတ်မှတ်ထားရမှာဖြစ်ပါတယ်။ ဥပမာ - တစ်နောက်ကြိမ် Backup လုပ်ယူတယ်ဆိုရင် အကြောင်း အမျိုးမျိုးကြောင့် Data တွေပျက်စီးဆုံးရတဲ့အခါ နေ့စဉ် Backup Data အမြှုပ်နှံမှာဖြစ်လို ပြီးခဲ့တဲ့တစ်ရက်အခြေအနေနဲ့ လိုအပ်ရင် Restore ပြန်လုပ်နိုင်မှာဖြစ်ပါတယ်။

Backup နဲ့ပေါက်သက်ရင် စဉ်းစားရမယ့် အဓိကအချက်ကတော့ Backup ဖိုင်တွေကို ဘယ်လိုသိမ်းမလဲဆိုတဲ့ အချက် ဖြစ်နိုင် ပါတယ်။ Backup ဖိုင်တွေသိမ်းဆည်းဖို့အတွက် NAS, SAN, Tape Backup စတဲ့ Storage Device တွေနဲ့ သိမ်းဆည်း နိုင်ပါတယ်။ ဒါပေမယ့် လက်တွေမှာ Backup ဆိုတာ လိုရမယ်ရ သေချာအောင် ယူထားတာ ဖြစ်ပါတယ်။ အမြဲ ပြန်သုံး နေဖို့ လိုအပ်တဲ့အရာမဟုတ်ပါဘူး။ တနည်းအားဖြင့် ပြန်လည်ရယူအသုံးပြုရ လွယ်အောင်ဆိုပြီး ကိုယ့် Architecture ထဲမှာ တွဲသိမ်းနေဖို့ မလိုအပ်ပါဘူး။ နောက်တစ်ချက်က သေချာအောင် Backup လုပ်ပြီး ကိုယ့် Architecture နဲ့ တွဲသိမ်း ထားပါပြီတဲ့၊ Datacenter ကြီးမီးလောင်သွားရင် ဘယ်လိုလုပ်မလဲ။ မဖြစ်နိုင်ဘူးလို ပြောလို မရပါဘူး၊ သေချာအောင် Backup ယူတယ်ဆိုပြီး၊ ပြဿနာရှိလာလို Backup ပါရောပါသွားရင်တော့ မဟုတ်သေးပါဘူး။ ဒါကြောင့် Backup ဖိုင်တွေကိုတော့ Google Drive, Amazon S3 စတဲ့ Cloud Storage တွေပေါ်မှာ ခွဲခြားသိမ်းဆည်းထားခြင်းက ပိုပြီး သဘာဝကျမှာဖြစ်ပါတယ်။ ဖိုင်တွေသိမ်းဆည်း ဖို့အတွက် ဈေးကြီးတဲ့ SAN Device တွေ ဝယ်စရာမလိုတော့တဲ့အတွက် စရိတ်အရာသက်သာသွားမှာဖြစ်သလို၊ အကြောင်းအမျိုးမျိုးကြောင့် Server Architecture ထိခိုက်ပျက်စီး ရင်လည်း Backup ကတော့ သီးခြားကျွန်းရှိနေမှာ ဖြစ်ပါတယ်။

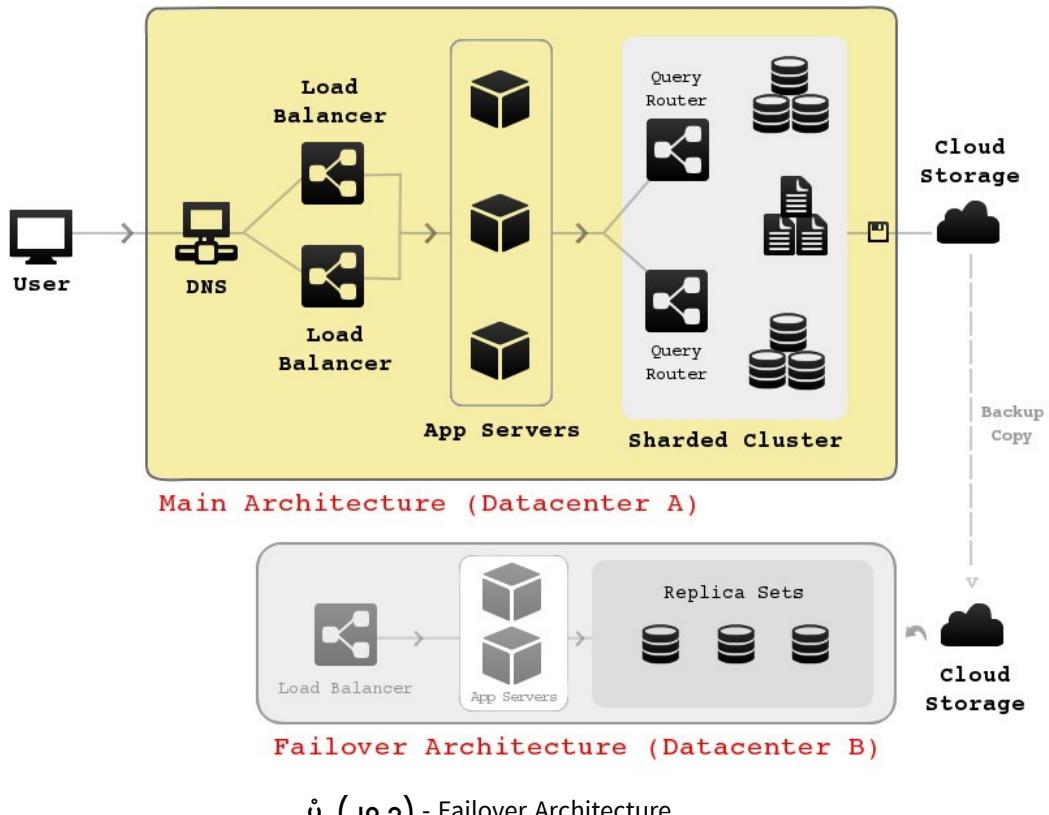
## 20.7 – Failover Architecture to Embrace Disasters

Data Backup ကို ပုံမှန်ပြုလုပ်ခြင်းအားဖြင့် Data ပျက်စီးဆုံးမှုကို ရင်ဆိုင်နိုင်မှာဖြစ်ပေမယ့်၊ အကြောင်းအမျိုးမျိုး ကြောင့် Server Architecture ကြီးတစ်ခုလုံး ပျက်စီးသွားမယ် (သို့မဟုတ်) အလုပ်မလုပ်တော့ဘူးဆိုရင် ဘယ်လိုလုပ်မလဲ ဆိုတာလည်း စဉ်းစားစရာပါ။ ရေးကြီးခြင်း၊ မှန်းတိုင်း၊ ငလျှင် စတဲ့ သဘာဝဘေးအန္တရာယ်တွေကြောင့်လည်း Server Architecture ကို ထိခိုက်နိုင်သလို၊ မသမသူမှားရဲ့ တိုက်ခိုက်မှု ကြောင့်လည်း Server Architecture ကို ထိခိုက်ပျက်စီး စေနိုင်ပါတယ်။ ဒါလိုအခြေအနေမျိုး ကြဲလာရင် ရင်ဆိုင်နိုင်ဖို့အတွက် Failover Architecture ကိုလည်း ထည့်သွင်းစဉ်းစား ထားသင့်ပါတယ်။

Failover Architecture ဟာ ဘယ်လိုဖြစ်သင့်တယ်ဆိုတာကို ပုံသေပြောလိုတော့မရတော့ပါဘူး။ အခြေအနေ တွေက တစ်ခုနဲ့တစ်ခု တူမှာမဟုတ်ပါဘူး။ အခြေခံ Procedure ကတော့ ဒီလိုပါ။

1. Cloud Storage မှာ Data Backup ကို အမြဲပြုလုပ်ထားသင့်ပါတယ်။
2. အဲ့ခဲ့ဒီ Data Backup ကို အခြား Cloud Storage တစ်ခုမှာ ထပ်ဆင့် Backup ပြုလုပ်ထားသင့်ပါတယ်။
3. သီးခြား Datacenter တစ်ခုမှာ လက်ရှိ Architecture လောက် ပြည့်စုံဖို့မလိုအပ်ပေမယ့် လက်ရှိ Architecture နည်းတူ အလုပ်လုပ်နိုင်တဲ့ Failover Architecture တစ်ခုကို Setup လုပ်ထားသင့်ပါတယ်။
4. Failover Architecture မှာပါဝင်တဲ့ App Server တွေ ဟာ အမြဲအလုပ်လုပ်စရာမလိုပဲ Stand-by အနေ အထားနဲ့ အသင့်ရှိနေရမှာ ဖြစ်ပါတယ်။

- Failover Architecture မှာပါဝင်တဲ့ Database Server ထကို ပင်မ Architecture ကနေရရှိလာတဲ့ Data Backup နဲ့ အမြဲ Restore လုပ်ပြီး အသင့်ပြင်ထားသင့်ပါတယ်။
- ဒီနည်းနဲ့ Failover Architecture ထဲက Database Server မှာ Data တွေ အမြဲအသင့်ရှိနေမှာဖြစ်ပါတယ်။



### ပုံ (၂၀.၂) - Failover Architecture

ဒီနည်းနဲ့ အကြောင်းအမျိုးမျိုးကြောင့် ပင်မ Architecture အလုပ်မလုပ်တော့ရင်တော့ Backup Failover Architecture က ဆက်လက်အလုပ်လုပ်ပေးနေနိုင်မှာပဲဖြစ်ပါတယ်။ ဒီနေရာမှာ ထပ်မံစဉ်းစားစရာရရှိလာတာ ကတော့ ပင်မ Architecture ကို ဘယ်ပုံဘယ်နည်း Recover ပြန်လုပ်မလဲဆိုတဲ့ ကိစ္စနဲ့ ပင်မ Architecture ပြန်လည်အလုပ်လုပ်နိုင်တဲ့အခါ Failover နဲ့ ပင်မ Architecture ဘယ်လို့ Migrate ပြန်လုပ်မလဲဆိုတဲ့ ကိစ္စတွေပါစဉ်းစားရတော့မှာဖြစ်ပါတယ်။

## Conclusion

ကနောက်ဆုံးမှာ Full-stack Developer တစ်ဦးဟာ Software ကို ရေးသားဖန်တီးပေးယံ့သာမက Server Architecture ကိုလဲ ကိုယ်တိုင်မဲ့နိုင်ဖို့ လိုအပ်ပါတယ်။ လက်တွေမှာ ကိုယ့်လိုအပ်ချက်နဲ့ ကိုက်ညီအောင် Server Architecture Design ကို ကိုယ်တိုင်ရေးဆွဲသတ်မှတ်ဖို့ လိုနိုင်ပါတယ်။ အဲဒီလို့ ရေးဆွဲရာမှာ အထောက်အကူဖြစ်စေဖို့အတွက် အသုံးများ ပြီး ဖြစ်နိုင်ခြေရှိတဲ့ Architecture များကို ရွေးထုတ်ဖော်ပြုခြင်း

ဖြစ်ပါတယ်။ ဒု့အပြင် ကျွန်တော်တို့ရေးသားတဲ့ Software ဟာလည်း Server Architecture နဲ့ကိုက်ညီဖို့ လိုအပ်ပါတယ်။ ဥပမာ – Memcached သုံးမယ်ဆိုရင် ကျွန်တော်တို့ Software ကလည်း Memcached Server ကနေ Data ကို ရယူကြည့်ပြီး Cache မရှိမှ အမှန်တစ်ကယ် Database Server ကို ဆက်သွယ်အောင် ရေးသားပေးရမှာ ဖြစ်ပါတယ်။ တန်ည်းအားဖြင့် Architecture ကသာ Scalable ဖြစ်ဖို့လိုအပ်ခြင်းမဟုတ်ပဲ၊ ကျွန်တော်တို့ရေးသားတဲ့ Software တွေကလည်း Scalable Architecture ပေါ်မှာ အလုပ်လုပ်နိုင်တဲ့ Scalable Software တွေဖြစ်ဖို့ လိုအပ်ပါတယ်။ ဒါကြောင့် Server Architecture နဲ့ ပက်သက်တဲ့ ဗဟို သုတဟာ ကိုယ်တိုင် Server Architecture တွေတပ်ဆင်စီမံတဲ့နေရာမှာသာမက Scalable Software တွေ ရေးသား ဖန်တီးရမှာလည်း အထောက်အကူဖြစ်စေမှာဖြစ်ကြောင်း ဖော်ပြလိုက်ရပါတယ်။

နိဂုံးချုပ်

Rockstar Developer

Rockstar Developer ဆိုတာ အဖွဲ့လိုက်ပူးပေါင်းဆောင်ရွက်  
နိုင်စွမ်းရှိပြီး၊ ပြုပြင်ထိမ်းသိမ်းရလွယ်ကူတဲ့ Maintainable Code နဲ့  
အခြေခံခိုင်မာတဲ့ Scalable Code တွေကိုရေးသားနိုင်တဲ့အပြင်  
စဉ်ဆက်မပြတ် လေ့လာသင်ယူနေတဲ့သူဖြစ်ပါတယ်။

### **Rockstar Developer Course**

Project Management, Web Service, Server Architecture  
NodeJS အစရိုးသည် ဤစာအုပ်ပါ အကြောင်းအရာများတို့  
စာရေးသူကိုယ်တိုင် သင်ကြားပေးခြင်းဖြစ်သည်။  
ဆက်သွယ်ရန် - (ဝါ) ၇၃၀ ၆၅၉ ၆၂

<http://eimaung.com/courses>

## နိဂုံးချုပ်

ဒီစာအုပ်ဟာ စာဖက်သူကို သာမာန်ထက် (၁၀) ဆအလုပ်ပိုတွင်တဲ့ 10x Programmer (သို့မဟုတ်) Rockstar Developer တစ်ဦးဖြစ်လာစေရန် တစ်ဖက်တစ်လမ်းကနေ အထောက်အကူဖြစ်စေဖို့ ရည်ရွယ်ရေးသားထားခြင်းဖြစ်ပါတယ်။ သာမာန်ထက် (၁၀) ဆအလုပ်ပိုတွင်တဲ့ Rockstar Developer တစ်ဦးမှာ ရှိလေ့ရှိတဲ့ အရည်အသွေးတွေ အများ ကြိုးရှိပါတယ်။ အဲဒီတဲ့က အမိက (၅) ချက်ကို ရွေးထုတ်ဖော်ပြရရင်တော့ -

1. အဖွဲ့လိုက်ပူးပေါင်းဆောင်ရွက်နိုင်သည့် အရည်အသွေးရှိခြင်း
2. ဖတ်ရှုနားလည်ရလွယ်ပြီး၊ ပြင်ဆင်ထိမ်းသိမ်းရလွယ်ကူသည့် Code များ ရေးသားနိုင်ခြင်း
3. သဘောသဘာဝပိုင်း ပိုင်နိုင်၍ ရေရှည်အဆင့်မြှင့်တင်မှုအတွက် အခြေခံနိုင်မာသည့် Code များ ရေးသားနိုင်ခြင်း
4. အလုပ်တွင်စေရန်အတွက်၊ ဆက်စပ်နည်းပညာများကို ထိရောက်အကျိုးရှိအောင် အသုံးချနိုင်ခြင်း
5. အဆက်မပြတ် လေ့လာသင့်ယူတက်သည့် အလေ့အထရှိခြင်း

- တို့ပဲဖြစ်ပါတယ်။

ဒီစာအုပ်မှာ လေ့လာခဲ့တဲ့ Source Code Management System, Issue Tracking System, SCRUM စတဲ့ အကြောင်းအရာတွေကို စာဖက်သူရဲ့ အဖွဲ့လိုက်ပူးပေါင်းဆောင်ရွက်နိုင်မှုကို အထောက်အကူဖြစ်စေလိမ့်မယ်လို့ မျှော်လင့်ပါတယ်။ SOLID, TDD, SOA စတဲ့ သဘောသဘာဝများနဲ့အတူ BackboneJS လို့ Code Architecture နည်းပညာအကူအညီနဲ့ ပြင်ဆင် ထိမ်းသိမ်းရလွယ်ကူတဲ့ Code များရေးသားရာမှာ အထောက်အကူ ဖြစ်စေလိမ့်မယ်လည်း မျှော်လင့်ပါတယ်။ NodeJS, MongoDB, Nginx စတဲ့ နည်းပညာများကိုလည်း ထည့်သွင်းဖော်ပြခဲ့တဲ့အတွက် ရေရှည်အတွက်အခြေခံနိုင်မာပြီး Scalable ဖြစ်တဲ့ Code များ ရေးသားရာမှာလည်း အထောက်အကူ ဖြစ်စေလိမ့်မယ်လို့ မျှော်လင့်ပါတယ်။ Yeoman, NPM, Bower, Grunt, Docker စတဲ့ Tool များကိုလည်း ထည့်သွင်းဖော်ပြခဲ့တဲ့အတွက် လုပ်ငန်းစွမ်းဆောင်ရည် ပိုမိုကောင်းမွန်စေဖို့အတွက် အထောက်အကူဖြစ်စေနိုင်တဲ့ စိတ်ကူးအိုင်ခိုက်ယာများလည်း ရရှိလိမ့်မယ်လို့ မျှော်လင့်ပါတယ်။

ဒီစာအုပ်မှာ Rockstar Developer တစ်ဦး သိရှိသင့်တဲ့ သဘောသဘာဝတွေ၊ နည်းပညာတွေနဲ့ နည်းစနစ်တွေ ကို တက်နိုင်သမျှစုံလင်အောင်စုစည်းဖော်ပြထားပါတယ်။ ဒီလိုပါဝင်တဲ့အကြောင်းအရာများပြားတဲ့ အတွက် ကြောင့် တစ်ဦးအကြောင်းကိုဖော်ပြရာမှာ အခြေခံကျပြီးသတိမှုသင့်တဲ့အပိုင်းကိုသာ ဦးစားပေး ဖော်ပြီး တစ်ဦးအသေးစိတ်တွေကို ချုန်လုပ်ခဲ့ရတာတွေလည်းရှိပါတယ်။ ဒါပေမယ့် ဒီစာအုပ်က သက်ဆိုင်ရာ ကဏ္ဍာကို ကိုယ်တိုင်ဆက်လက်လေ့လာ သွားရာမှာ လမ်းစရေစွဲအတွက် အထောက်အကူပြုလိမ့်လို့မျှော်လင့်ရင်း ဒီနေရာမှာတင် နိဂုံးချုပ်လိုက်ရပါတယ်ခင်ဗျာ။

## ကျေးဇူးတင်လွှာ

ဤစာအပ်အတွက်အမှာစာ ရေးသားပေးခဲ့ကြသော ဆရာတီးထွန်းခိုင်နှင့် ကိုသာထက်တို့အား အထူးကျေးဇူးတင်ရှိပါသည်။ စာမူကြမ်းဖတ်ရှု၍ မှတ်ချက်နှင့် အကြံပြုချက်များပေးခဲ့ကြသော ကိုမိုးလုံးပြည့်လျှံး၊ ကိုညာ၏လင်းထွန်း၊ ကိုနိုင်လင်းအောင်၊ ကိုတင်အောင်လင်း နှင့် ကိုဝင်းထွန်းဟန် တို့ကိုလည်း ကျေးဇူးတင်ရှိပါသည်။ မအားလပ်သည့်ကြားမှ စာမူကြမ်းဖတ်ရှုပေးခဲ့ ကြသော ကိုသန့်သက်၊ ကိုစွမ်းထက်အောင်၊ ကိုသီဟအေးကျော်၊ ကိုရဲလင်းအောင်နှင့် ကိုကောင်းခန့်မင်းထွန်းတို့အားလည်း ကျေးဇူးတင်ရှိပါသည်။ စာအပ်မထွက်မိကပင်၊ ကြိုတင်မှာယူစနစ်ဖြင့် မှာယူအားပေးပြီး ပုံပိုးကူညီခဲ့ကြသူများနှင့် စာအပ်ဖြစ်မြောက်လာရေးအတွက် အဘက်ဘက်မှ ပုံပိုးကူညီခဲ့ကြသူများ အားလုံးကို ကျေးဇူးအထူးတင်ရှိပါသည်။

အီမောင်

Fairway Web

စာဖတ်သူများအားလုံး ကိုယ်စိတ်နှစ်ဖြာ ကျမ်းမာချမ်းသာကြပါစေ။

Website: <http://rsdbook.com>

Author Profile: <http://eimaung.com>

Contact Email: [author@rsdbook.com](mailto:author@rsdbook.com)

Facebook: <http://fb.com/RockstarDeveloperBook>

၂၀၁၅ ခုနှစ်၊ စက်တင်ဘာလ (၂) ရက်နေ့တွင် ရေးသားပြီးစီးသည်။

၂၀၁၆ ခုနှစ်၊ ဂျေလိုင်လ (၅) ရက်နေ့တွင် PDF Ebook အဖြစ် ပြင်ဆင်ခြင်း ပြီးစီးသည်။