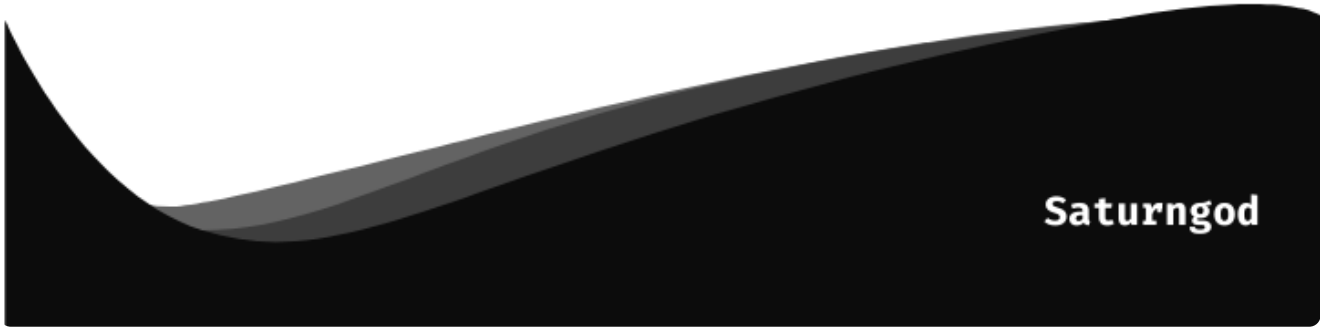
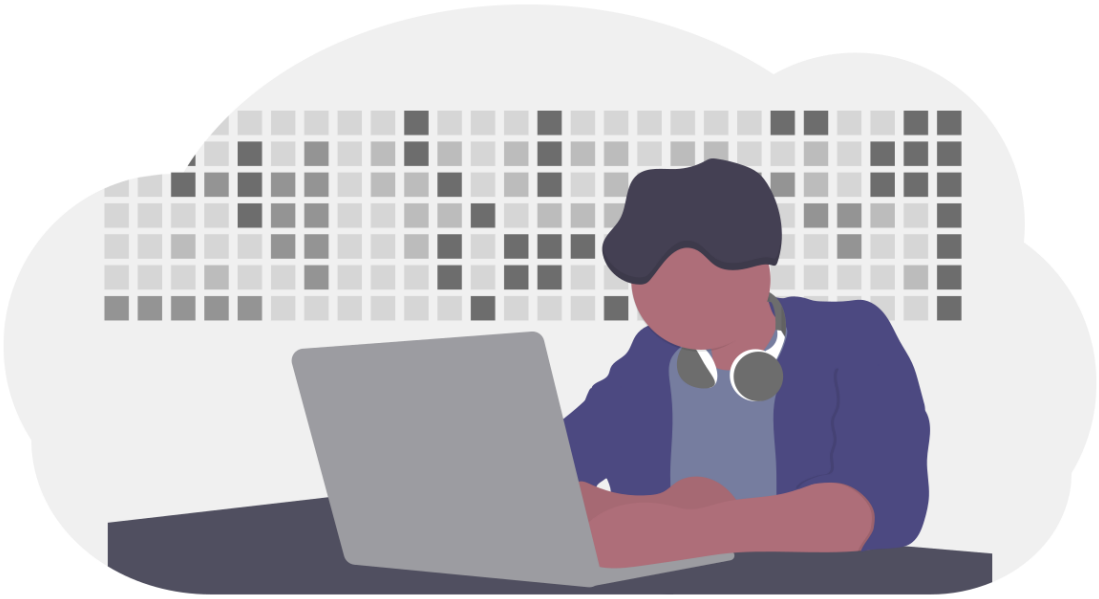
Developer

Intern



မိတ်ဆက်



Developer Intern စာအုပ်ဟာ Developer တစ်ယောက်ဖြစ်ဖို့ အတွက် ဘယ်ကနေ စရမလဲ ဆိုသူတွေ အတွက် Developer တစ်ယောက် ဖြစ်ဖို့ ဘာတွေ လေ့လာရမလဲ လုပ်ငန်းခွင်ဝင်သည့် အခါမှာ အဆင်သင့်ဖြစ်အောင် ဘာတွေ သိထားဖို့လိုလဲ ဘာတွေ ပြင်ဆင်ထားဖို့ လိုလဲ ဆိုသူတွေ အတွက် အဓိက ထားပြီး ရေးထားပါတယ်။ အထူးသဖြင့် ကွန်ပျူတာ တက္ကသိုလ် တက်ပြီးနောက် ဘာတွေ ဆက်လုပ်ရမလဲ ဆိုသူတွေ အတွက် ပိုသင့်သော်ပါလိမ့်မယ်။ ဒီစာအုပ်ကို ဖတ်ဖို့ အတွက် programming , mysql တို့ကို တတ်ကျွမ်းဖို့ လိုအပ်ပြီး web development နဲ့ server အကြောင်းလည်း ကြားဖူးနားဝ ရှိထားဖို့ လိုပါတယ်။

စာအုပ်ကို Digital Version ဖြစ် ဖြန့်ချီပြီး အမှားပြင်ဆင်ချက်တွေ အသစ်ထပ်မံ ဖြည့်သွင်းချက်တွေ ရှိခဲ့ရင် ပုံမှန် update လုပ်ပေးသွားဖို့ အတွက် ပါ။

စာအုပ်နဲ့ ပတ်သက်ပြီး သိလိုသည်များကို https://discord.gg/KUH3Bkmsna တွင် မေးမြန်းနိုင်ပါသည်။

စာအုပ် အတွင်း ပြောင်းလဲချက်များ

အခု စာအုပ်ဟာ Version 1.1.4 ဖြစ်ပါသည်။

Version 2.0.1(08.04.2024)

- SOLID principle ကို update လုပ်ထားသည်။
- Test Driven Development ကို update လုပ်ထားသည်။

Version 2.0.0(04.04.2024)

- Roadmap တွင် လေ့လာရာ လမ်းကြောင်း နှင့် ပတ်သက်ပြီး ပြင်ဆင်ထားသည်။ အကြောင်းအရာ အသစ်များ ထည့်သွင်းထားသည်။
- Resume ပိုင်း လိုတာတွေ ကို ထပ်ပြင်ထားသည်။

Version 1.1.4(13.12.2023) မှာ

- Clean Architecture chapter ထည့်ထားသည်
- Test Driven Development chapter ထည့်ထားသည်

Version 1.1.3(08.10.2022) မှာ

- Resume အပိုင်း အနည်းငယ်ပြင်ထားသည်။
- Interview တွင် လေ့ကျင့်ထားဖို့ အကြောင်း ထပ်ဖြည့်ထားသည်။

Version 1.1.2(05.09.2022) မှာ

- Docker အခန်း အမှားပြင်ထားပါသည်

Version 1.1.1(05.09.2022) မှာ

- စာလုံးပေါင်းပြင်ထားသည်
- အခန်း ၁၇ တွင် Jenkins ကို အသုံးပြုကြောင်း ထပ်ဖြည့်ထားသည်။
- အခန်း ၁၈ တွင် စာ အနည်းငယ်ပြုပြင်ထားသည်။

Version 1.1.0 (27.07.2022) မှာ

- S.T.A.R method ကို အခန်း ၃ တွင် ထပ်ဖြည့်ထားသည်။

Version 1.0.9 (22.07.2022) မှာ

- High Level Diagram အကြောင်း ထပ်ဖြည့်ထားပါသည်။

Version 1.0.8 (22.06.2022) မှာ

- Interview အခန်း ထပ်ဖြည့်ထားပါသည်။

Version 1.0.7 (18.05.2022) မှာ

- Resume အခန်းတွင် Keywords အပိုင်း ထပ်ဖြည့်ထားသည်။
- နိဂုံးချုပ် တွင် စာအနည်း ငယ် ထပ်ဖြည့်ထားသည်။

Version 1.0.6 (08.05.2022) မှာ

- Principles အခန်း ထပ်ဖြည့်ထားသည်။
- Software Development Life Cycle (SDLC) အခန်းဖြည့်ထားသည်။

Version 1.0.5 (16.04.2022) မှာ

- Security အခန်း ထပ်ဖြည့်ထားသည်။
- Principles အခန်း ထပ်ဖြည့်ရန် အတွက် စတင်ရေးသားသည်။

Version 1.0.4 (05.02.2022) မှာ

- Links များနေခြင်းများကို ပြင်ထားခြင်း
- Interview chapter အသစ် ထပ်မံဖြည့်စွက်ထားသည်။

Version 1.0.3 (16.01.2022) မှာ

- Contract ကို အခန်း ၁၄ တွင် ထပ်မံဖြည့်စွက်ထားသည်။

Version 1.0.2 (14.01.2022) မှာ

- စာအုပ် Cover ထည့်သွင်းထားသည်။
- Cron Job ကို အခန်း ၁၁ တွင် ထည့်ထားပါသည်။

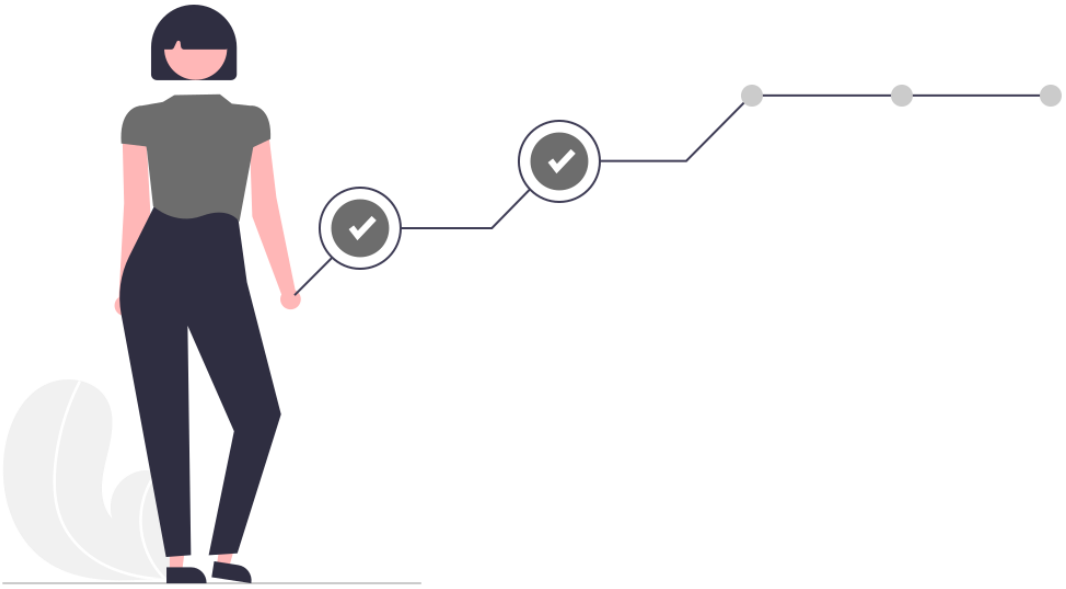
Version 1.0.1 (12.01.2022) မှာ

- စာလုံး အမှား ပြင်ဆင်ခြင်း
- အခန်း ၁၄ တွင် မမှန်သည့် သီအိုရီ စာ ထပ်ဖြည့်ခြင်း ပြုလုပ်ထားပါသည်။

Version 1.0 တွင်

- စာအုပ် အကြမ်းအားဖြင့် အခန်းများ အားလုံး ပါဝင်ခြင်း။
- Online ဖတ်ရန် ဖန်တီးထားခြင်း။

အခန်း ၁ :: Roadmap



Developer တစ်ယောက်မဖြစ်ခင်မှာ ကိုယ်ဘယ်လမ်းကြောင်း ကို သွားမယ်ဆိုတာ အရေးကြီးပါတယ်။ Developer တစ်ယောက် မဖြစ်ခင်မှာ ဦးစွာ လေ့လာဖို့တွေကတော့ Programming ပါ။

ပထမ အဆင့်

ပထမ အဆင့် အနေနဲ့ လေ့လာဖို့ကတော့

- OOP အသုံးပြုသည့် programming basic
- HTML
- CSS
- JavaScript

စသည်တို့ကို ပထမ အဆင့်အနေနဲ့ လေ့လာရပါမယ်။ Web Developer မလုပ်ဘူးဆိုပေမယ့် နည်းပညာလောကထဲမှာ ရှိမယ် ရေရှည်နေမယ် ဆိုရင်တော့ ဒီ ပထမ အဆင့် အကုန် လေ့လာဖို့ လိုတယ်။

ဒုတိယ အဆင့်

ဒုတိယ အဆင့်အနေနဲ့ကတော့

- Object-oriented programming (OOP)
- Database (SQL)
- JSON, XML , YML
- API (RESTful, GraphQL)

လက်ရှိ programming တွေ အားလုံး နဲ့ မကင်းတာကတော့ Object-oriented programming လို့ ခေါ်သည့် OOP ပါ။

အများအားဖြင့် OOP ကို သိရုံ လောက် လေ့လာပြီးတော့ လုပ်ငန်းခွင် ဝင်ကြတာ များပါတယ်။ OOP ကို သေချာ နားလည် နေဖို့လိုတယ်။ လက်ရှိ လုပ်ငန်းခွင်မှာ စီနီယာ တွေဟာ OOP ကို သေသေချာချာ သုံးထားပြီး ကိုယ်နားမလည်ရင် ပြဿနာ တွေဖြစ်ကုန်နိုင်ပါတယ်။ ဒီ code ကို ဘယ်လို ခေါ်လိုက်လို့ ဘယ်လိုဖြစ်သွားတယ် မသိ ဆိုသည့် magic code တွေ တွေ့နေရပါလိမ့်မယ်။

Database ပိုင်းမှာတော့ လေ့လာဖို့တွေက အရမ်းများတယ်။ ဒါပေမယ့် အဓိက ကျသည့် SQL ကိုတော့ မဖြစ်မနေ လေ့လာရပါမယ်။ SQL ကို လေ့လာထားပြီးရင်တော့ MS SQL ဖြစ်ဖြစ် MySQL ဖြစ်ဖြစ် လေ့လာရတာ အဆင်ပြေပါလိမ့်မယ်။ Database ကတော့ data တွေကို ထိန်းသိမ်းပေးပြီး လိုအပ်သည့် data တွေကို ရှာဖွေ ပေးပါတယ်။ Developer တိုင်းမ ဖြစ်မနေ သိရမည့် အပိုင်း တစ်ခုပါ။

```
SELECT name, age from students;
```

အထက်ပါ code က database ရဲ့ students table ထဲက name နှင့် age ကို ဆွဲထုတ်ထားသည့် ပုံစံပါ။

JSON, XML, YML တွေကတော့ programmer တိုင်း မဖြစ် မနေ သိသင့်သည့် text format တွေပါ။

JSON (JavaScript Object Notation) တွေကို Restful API တွေ မှာ မဖြစ်မနေ သုံးကြပါတယ်။ ရိုးရှင်းလွယ်ကူပြီးတော့ programming language တော်တော်များများမှာ ပါဝင်ပြီးသား ဖြစ်လို့ပါ။

```
{
        "name" : "Mg Mg",
        "age" : 23,
        "teacher" : false
}
```

အပေါ်က example မှာ ဆိုရင်တော့ ကြည့်လိုက်တာနဲ့ Mg Mg ရဲ့ information ဆိုတာကို သိနိုင်ပါတယ်။ JSON ကနေ Dictionary ကို programming language တိုင်းမှာ လွယ်လင့်တကူ ပြောင်းနိုင်သည့် အတွက် လူသုံးများပါတယ်။

XML ကတော့ အရင်က JSON popular မဖြစ်ခင် နှင့် Restful popular မဖြစ်ခင်တုန်းက SOAP တွေကို API အတွက် အသုံးများကြပါတယ်။ အဲဒီ အချိန်တုန်းကတော့ XML က မသိမဖြစ်ပါပဲ။

HTML ကတော့ Hypertext Markup Language (HTML) ဖြစ်ပြီး XML ကတော့ Extensible Markup Language (XML) ဖြစ်ပါတယ်။ HTML , XML က ရေးသားသည့် ပုံစံ တူညီ ကြပါတယ်။

```
<person teacher=false>
        <name>Mg Mg</name>
        <age>23</age>
</person>
```

JSON လိုမျိုး ဖြစ်ပေမယ့် XML က attribute , node value စတာတွေ အပြင် Document Object Model (DOM) နဲ့ သုံးရသည့် အတွက်ကြောင့် programming လေ့လာကာစ junior တွေ အတွက် အခက်အခဲ ဖြစ်တတ်ပါတယ်။

YAML ကတော့ နောက်ပိုင်းမှာ config file ပုံစံနဲ့ လူသုံးများလာပါတယ်။ API request response တွေမှာ ထက် config ပုံစံတွေအတွက် ပိုအဆင်ပြေပါတယ်။ YAML ရဲ့ အရှည်ကောက်ကတော့ နည်းနည်းဆန်းတယ်။ YAML Ain't Markup Language က YAML ရဲ့ အရှည်နာမည်ပါ။

```
name: Mg Mg
age: 23
teacher: false
```

တတိယ အဆင့်

တတိယ အဆင့်မှာတော့ ရွေးချယ်ခြင်းပါ။

- Web Development
- Mobile App Development
- App Development for OS (Windows/Linux/Mac)

ဒီအဆင့်ဟာ Developer တွေ အတွက် အရေးကြီး ဆုံးအဆင့်ပါပဲ။ ကိုယ်က ဘာလုပ်ချင်လဲ ဆိုတာ သိဖို့ အရေးကြီးပါတယ်။ ဒီ အပိုင်းတွေထဲကမှ လိုင်းကြောင်း တစ်ခုကို ပိုင်ပိုင် နိုင်နိုင် ရွေးချယ်သွားဖို့က လိုအပ်ပါတယ်။

Web Development

Web Development ပိုင်းကို လေ့လာမယ်ဆိုရင် ၂ ပိုင်း ထပ် ကွဲပါသေးတယ်။

- Frontend (jQuery, Javascript, ReactJS, Angular, VueJS)
- Backend (PHP, Java, NodeJS, Python, Ruby On Rail)
- FullStack (Frontend + Backend)

ဆိုပြီး ၂ ပိုင်း ကွာခြားပါတယ်။ ပုံမှန် အားဖြင့် frontend ကော backend ၂ ခု လုံး လုပ်ကြသည့် developer တွေ အများကြီးရှိပါတယ်။ တချို့တွေကတော့ database ပိုင်းမှာ အားနည်း ပေမယ့် UI/UX လှလှပပ နှင့် လူတွေ လွယ်ကူစွာ အသုံးပြုနိုင်အောင် ဖန်တီးနိုင်ခြင်း ၊ React JS, Angular, Vue JS လိုမျိုး javascript library , javascript framework တွေ နဲ့ ရေးသားရတာ ပို အသားကျခြင်း တို့ကြောင့် front end developer ပိုင်း ကို လုပ်ကြပါတယ်။

Backend ပိုင်းမှာတော့ database query တွေ ရေးခြင်း နှင့် အဓိက system တစ်ခုလုံးရဲ့ logic ပိုင်း ဆိုင်ရာတွေကို ဖန်တီးခြင်း တို့ကို ပြုလုပ်ကြပါတယ်။ Backend ပိုင်းဟာ ရှုပ်ထွေးလှသည့် business logic တွေ နဲ့ database query တွေကို front end အတွက် ထုတ်ပေးရပါတယ်။

FullStack ကတော့ front end, backend မခွဲပဲနှင့်လည်း ရေးကြပါတယ်။ Web Developer တစ်ယောက် အနေနဲ့ ပထမဆုံး အနေနဲ့ front end ပိုင်း ကော backend ပိုင်းကော လေ့လာထားဖို့ လိုအပ်ပါတယ်။ ပြီးမှသာ ဘယ်အပိုင်း ကိုယ်အားသန်တယ် ဆိုတာကို ဆုံးဖြတ်ဖို့ လိုပါတယ်။

Web Development ပိုင်းမှာ ထပ်ပြီး ခွဲရရင် backend ပိုင်းမပါပဲ database တွေ မပါပဲ သာမာန် သမာရိုးကျ company website, အသင်းအဖွဲ့ website ဖန်တီးသည့် website development လည်း ရှိပါတယ်။ အဓိက front end ပိုင်းပဲ ပါပြီးတော့ website ကနေ information ပေးဖို့ အတွက် ရေးဆွဲကြပါတယ်။

Web App တွေကတော့ front end , backend ၂ ခု လုံး ပါဝင် ရေးဆွဲဖို့လိုပါတယ်။ Backend ကနေ Front end လိုသည့် data တွေကို Restful API မှ တဆင့် JSON စတာတွေ နဲ့ ပေါင်းကူးထားပေးပါတယ်။ ဒါကြောင့် Restful ဆိုတာ ဘာလဲ JSON ဆိုတာ ဘာလဲ ဆိုတာကို မဖြစ်မနေ လေ့လာထားသင့်ပါတယ်။

၂၀၂၃ နောက်ပိုင်းမှာတော့ FullStack က ခေတ်စားလာပါတယ်။ Full Stack သမားတွေကို နောက်ပိုင်း ပိုပြီး အလုပ်ခေါ်လာကြပါတယ်။ FullStack သမားတွေဟာ backend ကော frontend ကော ရေးနိုင်သည့် သူတွေပါ။ အရင်က developer ၁၀ ယောက် မှာ ၅ ယောက် backend , ၅ ယောက် frontend team ကနေ ၁၀ ယောက် fullstack team ပုံစံ ပြောင်းလဲနေပါပြီ။ ဒါကြောင့် fullstack developer အနေနဲ့ အလုပ်ရှာမှသာ အလုပ်ရဖို့ အခွင့် အလမ်း များပါမယ်။

Mobile App Development

Mobile App Development ဆိုတာနဲ့ iOS , Android App ဖန်တီးဖို့ကို တွေးမြင်ကြပါလိမ့်မယ်။

iOS App ကို development လုပ်ဖို့ အတွက် Mac OS, Xcode လိုအပ်ပါတယ်။ Language ကတော့ Swift language နဲ့ ရေးသားရပါတယ်။ အရင်တုန်းကတော့ Objective-C နဲ့ ရေးသားပါတယ်။ တချို့ library တွေက အခု ထက်ထိ objective-c နဲ့ ရေးသားထားတာတွေ ရှိပါသေးတယ်။ နောက်ပိုင်းမှာ SwiftUI ကို ပြောင်းလဲ အသုံးပြုလာကြပါပြီ။

Android App အတွက် Kotlin နှင့် နောက်ပိုင်း ရေးသားကြပါတယ်။ အရင်ကတော့ Java နဲ့ ရေးသားပါတယ်။ တချို့ code တွေကတော့ Java နဲ့ ရေးသားထားတာတွေ ရှိပါသေးတယ်။

Cross Platform ဆိုတာကတော့ iOS , Android အတွက် တစ်ခါတည်း ရေးပြီးတော့ platform နှစ်ခုလုံးမှာ run လို့ရအောင် ရေးသားလို့ရပါတယ်။ Cross platform အတွက်က react native နှင့် flutter ကို အသုံးများပါတယ်။ အဓိက native မဟုတ်ပဲ web view နဲ့ ရေးကြတာတွေလည်း ရှိပါတယ်။ Ionic က web developer အတွက် native နဲ့ မဟုတ်ပဲ JavaScript နဲ့ ရေးသားကြပါတယ်။ Web အတွက် အဓိက ပြဿနာကတော့ performance က native လောက် မကောင်းတာပါ။ လက်ရှိ မှာကတော့ flutter က native လိုမျိုး performance ကောင်းပါတယ်။

React Native က React JS တတ်ကျွမ်းသည့် သူတိုင်း လွယ်လင့်တကူ ရေးသားနိုင်ပါတယ်။ React JS သမားတွေ အတွက် Mobile App ကို ရေးချင်သည့် အခါ React Native က ရွေးချယ်စရာ ပါ။ ရေးသားပုံ စဉ်းစားပုံ အတူတူပဲ ဖြစ်သည့်အတွက်ကြောင့် လေ့လာရမှာ လွယ်ကူစေပါတယ်။

Flutter က Dart ကို အသုံးပြုပြီး ရေးသားရပါတယ်။ နောက်ပိုင်းမှာ flutter ကို လူသုံးများလာပါတယ်။ အဓိကတော့ community အားကောင်းပြီးတော့ react native နဲ့ ယှဉ်ရင် performance ပိုကောင်းတယ်လို့ ယုံကြည်ကြလို့ပါ။ Flutter ဟာ React Native နဲ့ ယှဉ်ရင် memory လည်း ပိုမိုသက်သာပါတယ်။

အဓိကတော့ ဘယ် platform က ကိုယ့်အတွက် အဆင်ပြေမယ် ဆိုတာကို ကိုယ်တိုင်လေ့လာပြီး ရွေးချယ်တာ ပိုအဆင်ပြေပါတယ်။

အခြား

နည်းပညာမှာ လေ့လာဖို့က အများကြီးပါ။ ဥပမာ Cloud နှင့် ပတ်သက်ပြီး စိတ်ဝင်စားသည့် အခါမှာ DevOps လိုမျိုး position တွေရှိပါတယ်။ Cloud ပိုင်းမှာလည်း Amazon, Google, Microsoft cloud စသည် တို့ ကွဲပြားသည့် အတွက် လေ့လာစရာ လမ်းကြောင်းတွေ အများကြီး ရှိပါတယ်။

Developer တစ်ယောက် အနေနဲ့က လေ့လာမှုကို ရပ်ထားလိုက်လို့မရပါဘူး။ သို့ပေမယ့်လည်း အကုန်လုံးကို လေ့လာနေဖို့ အချိန်လည်း မရှိပါဘူး။ ဒါကြောင့် ဝါသနာပါသည့် လမ်းကြောင်းကို ကျွမ်းကျင်အောင်သွားပါ။ ပြီးမှ အခြား လမ်းကြောင်းတွေကို အခြေခံ အဆင့် လောက် နားလည်အောင် လေ့လာထားရင် လုံလောက်ပါတယ်။

အခန်း ၂ :: Resume

Resume လို့ ခေါ်သလို CV လို့လည်း ခေါ်ကြပါတယ်။ အလုပ်ရှာတော့မယ် ဆိုရင် မဖြစ်မနေ Resume ကို ပြင်ရပါတယ်။ အလုပ်ခေါ်သည့် အခါမှာ တချို့ တွေက ဘာ experience မှ မပါပဲ အပြင်မှာ CV form ဝယ်ပြီး ဖြည့်ပြီး ပို့လိုက်တာ ရှိသလို လုပ်ခဲ့သမျှ အသေးစိတ်ကို စာမျက်နှာ တွေ အများကြီး နဲ့တင်ထားတာလည်း ရှိပါတယ်။

Resume မှာ အဓိက အားဖြင့် သိချင်တာကတော့ လက်ရှိ လိုအပ်နေသည့် နေရာအတွက် ကိုက်မ ကိုက် အဓိက ကျပါတယ်။ ရုံးမှာ လိုတာက Laravel ဖြစ်ပြီး ကိုယ်က C# လျှောက်ရင်တော့ အလုပ်ရမှာ မဟုတ်တာ သေချာ သလောက်ပါပဲ။ နောက်ပြီး တွေ့သမျှ အကုန် လျှောက်ခဲ့လို့ အလုပ် ရခဲ့ရင်တောင် ကိုယ်ရချင်သည့် အလုပ်မဟုတ်ရင် ပျော်မှာ မဟုတ်ပါဘူး။ ဥပမာ ကိုယ်က Mobile App Development လုပ်ချင်ပေမယ့် ရသည့် အခါမှာတော့ web development ဖြစ်နေ မျိုးပေါ့။

လိုအပ်ချက်

Resume မှာ အဓိက keyword တွေ match ဖြစ်အောင် ဖော်ပြရေးသားထားဖို့ လိုပါတယ်။ နောက်ပိုင်းမှာ Resume ကို AI နဲ့ စစ်တာ ရှိသလို keyword နဲ့ filter လုပ်တာလည်း ရှိပါတယ်။ Resume တစ်ခု မှာ ပါသင့်သည့် အချက်တွေကတော့

- Name
- Date of Birth
- Address
- Phone Number
- Website
- Email Address
- Social Network (Github, LinkedIn)
- Profile/Objectives
- Work Experience (Companies)
- Education
- Projects
- Awards
- Certifications
- Skills
- Hobbies
- Languages

စတာတွေ ပါဝင်ဖို့ လိုအပ်ပါတယ်။ Resume မှာ စာမျက်နှာ အနည်းအများထက်စာရင် job description ထဲမှာ ပါသည့် အချက်အလက်တွေ ကို resume ထဲမှာ ပါနေဖို့က ပို အဓိက ကျပါ တယ်။

Email ကတော့ အရေးကြီးပါတယ်။ မြန်မာနိုင်ငံမှာ လူငယ်တွေ အများစုဟာ email မသုံးကြပါဘူး။ messenger, viber စသည် ဖြင့် အဓိက သုံးပါတယ်။ အလုပ်လျောက်သည့် အခါမှာတော့ Email က အဓိက ကျပါတယ်။ ပုံမှန် email လည်း စစ်ဖို့ လိုအပ်ပါတယ်။

Social Network ဆိုသည့်အပိုင်းမှာလည်း Facebook ထက် Github, LinkedIn လိုမျိုး personal မဟုတ်ပဲ အလုပ်နဲ့ သက်ဆိုင်ရာ ကိုသာ ထည့်သွင်းသင့်ပါတယ်။ အလုပ်ခေါ်သည့် အခါမှာ Github ဆိုရင် code တွေကို လေ့လာကြည့်ဖို့နဲ့ လက်ရှိ skill က ဘယ်လောက်ရှိတယ် ဘာတွေ တတ်သလဲ ဆိုတာကို တစ်ခါတည်း လေ့လာနိုင်ပါတယ်။ LinkedIn ကတော့ ကိုယ့် အလုပ်နဲ့ ပတ်သက်ပြီး ဘယ်သူတွေကို သိသလဲ ဘယ်သူတွေက ကိုယ့်ကို recommend လုပ်ထားလဲ ဆိုတာကို ကြည့်ဖို့ အတွက်ပါ။

Profile/Objectives ကတော့ မိမိ အကြောင်း အကျဉ်းချုပ်သဘောနဲ့ ဘာကြောင့် အခုလျောက်သည့် position ဟာ ကိုယ် နဲ့ သင့်တော်တယ် ဆိုတာကို ရေးသားရသည့် အပိုင်းပါ။

Work Expereince ကတော့ ဘယ်ခုနှစ်ကနေ ဘယ်ခုနှစ် အထိ ဘယ် company တွေမှာ လုပ်ခဲ့လဲ ဆိုတာကို ဖော်ပြဖို့ပါ။ ဘယ် postion မှာ ဘာတွေ လုပ်ခဲ့တယ် ဆိုတာ အကျဉ်းအားဖြင့် ဖော်ပြထားဖို့ လိုပါတယ်။ Work Expereince မှာ အရေးကြီးသည့် နောက်အချက်ကတော့ Responsibilities ပါ။ အဲဒီ အလုပ်မှာ ဘာတွေ လုပ်ခဲ့တယ်။ ဘာတွေ ပြီးမြောက်ခဲ့တယ်။ ဘယ် framework, ဘယ် language တွေ နဲ့ သုံးပြီး ရေးခဲ့တယ် ဆိုတာ မျိုးက အရေးကြီးပါတယ်။

Technical Position ရှာနေတာဖြစ်သည့်အတွက်ကြောင့် Experience ကို ဖော်ပြသည့် အခါမှာ သက်ဆိုင်ရာ နည်းပညာ keyword တွေ ပါအောင် ရေးသားဖို့ လိုပါတယ်။ ဥပမာ

```
- Developed Streaming iOS App (Swift) with UIKit and MVC architecture.
- Implemented Download Manager using NSURLSessionDownloadTask and handling concurrency
with DispatchGroup.
```

အထက်ပါ အချက်အလက်မျိုးတွေကို Work Expereince မှာ ဖော်ပြထားပေးဖို့ လိုပါတယ်။

Education ကတော့ ကိုယ်တက်ခဲ့သည့် တက္ကသိုလ် ရခဲ့သည့် ဘွဲ့တွေကို ဖော်ပြထားဖို့ပါ။

Projects ကတော့ ကိုယ်လုပ်ခဲ့ဖူးသည့် projects တွေပေါ့။ ပုံမှန် အားဖြင့် open sources project တွေ နောက်ပြီး လုပ်ခဲ့သည့် ရုံးက တရားဝင်ဖော်ပြခွင့် ရှိသည့် project တွေသာ ဖော်ပြသင့်ပါတယ်။ ဥပမာ ကိုယ်လုပ်ခဲ့သည့် ရုံးက client အတွက် ရေးသားထားသည့် project ဖြစ်ပြီးတော့ ရုံးကနေ ရေးသားခဲ့သည် ဆိုတာကို ဖော်ပြခွင့် မရှိသည့် project မျိုးတွေကို မရေးသား သင့်ပါဘူး။

Project တွေထဲ ကိုယ်ပိုင် ဖန်တီးထားသည့် project တွေကိုလည်း ထည့်သွင်းနိုင်ပါတယ်။ အလုပ်လျောက်ဖို့အတွက် Project နေရာမှာ ရေးဖို့ မရှိသေးရင် ရှိအောင် အရင် ဖန်တီးပါ။ ကိုယ်ပိုင် Project အသေးလေးတွေ ဖြစ်သည့် MovieDB , Manga App, Video App စသည့် ကိုယ်ပိုင် App လေးတွေ ရေးသားနိုင်ပါတယ်။ နောက်အရေးကြီးဆုံးက ကိုယ်လုပ်ထားသည့် Project က ကိုယ် ကိုယ်တိုင်အတွက် အသုံးဝင်နေဖို့ လိုတယ်။

နောက်တဆင့် freelance project တွေကိုလည်း ထည့်သွင်းနိုင်ပါတယ်။ ထည့်သွင်းသည့် အခါမှာ သတိထားဖို့ လိုပါတယ်။ Client နဲ့ အရင်ညှိဖို့ လိုတယ်။ တချို့တွေက sub contract တွေဖြစ်တတ် သလို အချို့တွေက company ကနေ တဆင့် ပြန်ချသည့် sub contract တွေ ဖြစ်တတ်ပါတယ်။ အဲဒီ အခါမျိုးမှာ ထည့်သွင်းဖို့ အဆင်မပြေပါဘူး။

Awards , Certifications ကတော့ ကိုယ်ရခဲ့ဘူးသည့် ဆုတွေ certificate တွေကို ဖော်ပြဖို့ပါ။ ဥပမာ AWS Certified Solutions Architect လို့ ဖော်ပြထားရင် ကိုယ် AWS ကိုယ်တကယ် သိတယ် ဆိုတာကို ဖော်ပြထားရာလည်း ရောက်ပါတယ်။

Hobbies, Languages တွေကတော့ ရှင်းပြဖို့ မလိုပါဘူး။

Resume အတွက်

- https://rxresu.me
- https://resumake.io

လိုမျိုး template ရှိပြီးသား data ထည့်သွင်း ဖို့ လိုသည့် စနစ်တွေ အသုံးပြုသင့်ပါတယ်။

ပိုပြီး ကောင်းတာကတော့ MS Word ဖြစ်ဖြစ် Google Docs ဖြစ်ဖြစ် ATS (Applicant Tracking System) နဲ့ ကိုက်ညီ သည့် resume ကို ပြင်ဆင်ထားဖို့ လိုပါတယ်။ မဟုတ်သည့် အခါမှာ word,pdf ကို scan ဖတ်သည့် အခါမှာ data တွေက မှန်မှန် ကန်ကန် မဝင်တာ တွေ ဖြစ်တတ်ပါတယ်။

အတွေ့အကြုံမရှိသည့် Junior Developer

Junior Developer တွေမှာ Experience မရှိသည့်အတွက် Work Experience နေရာမှာ ရေးသားတာ ခက်ခဲပါတယ်။ အချို့ကလည်း Intern ခဏ ဝင်လုပ်ဖူးတာတွေပဲ ရှိပါတယ်။ အတွေ့အကြုံမရှိသည့် အခါမှာ Resume ကို projects တွေ စုထားပြီး ပြရပါတယ်။ လုပ်ခဲ့ဖူးသည့် projects တွေ ၊ opensource project တွေ ချရေးထားခြင်းဖြင့် အနည်းဆုံး A4 ၁ မျက်နှာ အပြည့် သို့မဟုတ် ၁ မျက်နှာ ခွဲလောက် ရနိုင်ပါတယ်။ Resume ရေးသားရာမှာ 2023 နောက်ပိုင်းမှာ one page resume တွေထက် ၂ မျက်နှာက နေ ၄ မျက်နှာ ရှိသည့် resume တွေ ကို ပိုပြီး ဖတ် ကြတာ များပါတယ်။ နိုင်ငံ နဲ့ လည်း ဆိုင်ပါတယ်။ recuriter အများစုကတော့ စာမျက်နှာ ၂ ခု လောက် ပါတာကို ပိုပြီး စိတ်ဝင်စားကြတယ် လို့ ပြောကြပါတယ်။

Keywords

အရေးကြီးသည့် အချက်ကတော့ Job Description မှာ ဖော်ပြထားသည့် keywords တွေဟာ ကိုယ့် resume မှာ ပါနေဖို့ လိုတယ်။ ပုံမှန် အားဖြင့် HR သို့မဟုတ် Recruitment team က Resume တွေ အများကြီးကို ကြည့်ရတာကြောင့် keyword နဲ့ filter လုပ်လိုက်တာ များပါတယ်။ Keywords နဲ့ ကိုက်သည့် သူတွေကိုပဲ ခေါ်ပါတယ်။ ဥပမာ Job Title က Project Manager ဖြစ်ပြီး Job Description ထဲမှာ PHP, MySQL လို့ ပါထားတယ်။ ကိုယ်က experience လည်းရှိတယ် ဒါပေမယ့် resume ထဲမှာ Java, Postgres လို့ ဖြစ်နေရင် HR က ပယ်ချမှာပဲ။ အများအားဖြင့် keyword နဲ့ စစ်ထုတ် တတ်ကြပါတယ်။

Resume များ

အလုပ်ခေါင်းစဉ်တစ်ခု အတွက် resume တစ်ခု ဖန်တီးထားဖို့ လိုပါတယ်။ ဥပမာ ကိုယ်က PHP အတွက် လျှောက်မယ်ဆိုရင် PHP နဲ့ ဆိုင်တာတွေ များများ ထည့်ထားပြီးတော့ Node.JS အတွက် ဆိုရင်တော့ javascript, typescript စသည့် project တွေကို အဓိက ဖော်ပြထားဖို့လိုပါတယ်။ Frontend position ဆိုရင်လည်း ReactJS တို့လိုမျိုး frontend နဲ့ ဆိုင်သည့် project တွေ ကို highlight လုပ်ထားဖို့ လိုအပ်ပါတယ်။

ကိုယ်လျှောက်မယ့် position title အတွက် မတူညီသည့် resume တွေ ပြင်ထားဖို့ လိုအပ်ပါတယ်။

အခန်း ၃ :: Interview

အကယ်၍ interview ခေါ်ခံ ရပြီ ဆိုရင်တော့ သင်ဟာ အလုပ်ရဖို့ 50% ရှိသွားပါပြီ။ အလုပ်ခေါ်စာတွေ အများကြီးထဲက မှ အဆင်ပြေနိုင်မယ့်သူတွေ ကို interview ခေါ်ကြပါတယ်။ ဒါကြောင့် interview မှာ ကောင်းမွန်စွာ ဖြေနိုင်ရင် အလုပ်ရဖို့ ရှိပါတယ်။

ပြင်ဆင်ခြင်း

Interview တိုင်းက ပြင်ဆင်ရပါတယ်။ ပြင်ဆင်တယ်ဆိုတာ CV form ပြင်ဆင်ရတာ မဟုတ်ပါဘူး။ မပြင်ဆင်ပဲ လာသည့် junior developer အများအပြားကို interview ခဲ့ဖူးတယ်။ အခုမှ အလုပ်လျောက်သူတွေဟာ Resume ကို ကောင်းမွန်စွာ မရေးထားသလို interview အတွက်လည်း ပြင်ဆင်ထားခြင်း မရှိပါဘူး။

ပထမဆုံး company အကြောင်းကို research လုပ်ရပါတယ်။ ဘယ် technology တွေ သုံးလဲ။ ဘယ်လိုမျိုး app တွေ project တွေ ပြီးထားတယ်။ client တွေက ဘယ်သူလဲ။ CEO က ဘယ်သူလဲ။ company မှာ ဝန်ထမ်း ဘယ်နှစ်ယောက် ရှိလဲ။ စသည့် အချက်အလက်တွေကို company website, facebook , linked in တို့မှာ လေ့လာဖို့ လိုတယ်။ Company အကြောင်းကို လေ့လာပြီးမှ ဒီ company မှာ အလုပ်လုပ်ချင် စိတ်ရှိ မရှိ ကို ပြန်လည် ဆန်းသစ်ရပါတယ်။

အဆင်ပြေတယ် လုပ်မယ် ဆိုရင် နောက်တဆင့် အနေနဲ့ company မှာ လုပ်နေသည့် အထဲမှာ အသိရှိလား ရှာရပါတယ်။ ရှိတယ်ဆိုရင် ရဖို့ အခွင့်အလမ်းများပါတယ်။ သူက ကိုယ့်ကို referral လုပ်ပေးနိုင်သလား ဆိုပြီး မေးကြည့်ဖို့ လိုတယ်။ နောက်ပြီးတော့ ဘယ်လို interview မေးတတ်လဲ။ code test က ဘာတွေ မေးတတ်လဲ ဆိုတာကို သိရနိုင်ပါတယ်။

Code test တွေ မေးတတ်သည့် company တွေ ဆိုရင် code တွေ algorithm တွေကို ပြန်လေ့လာထားဖို့လိုတယ်။ အကောင်းဆုံးကတော့ http://leetcode.com/ မှာ လေ့ကျင့်ဖို့ လိုအပ်ပါတယ်။ အဆင်ပြေတာကတော့ https://neetcode.io/ ပြထားသည့် အဆင့်တိုင်း နေ့စဉ် လေ့ကျင့်ထားရင် interview code questsions တွေကို လွယ်လွယ်လေး ဖြေနိုင်မှာပါ။

Algorithm တွေက အဖြေ တစ်ခါ သိထားပြီးရင် ဖြေရှင်းရလွယ်ပါတယ်။ အဖြေမသိသေးသည့် မေးခွန်းတွေ မေးလာရင် စဉ်းစားရတာ အချိန် အရမ်းကြာနိုင်ပါတယ်။ နောက်ပြီး ရနိုင်ခြေ အခွင့်အလမ်း နည်းပါတယ်။

S.T.A.R Method

Interview တွေ ဖြေသည့် အခါမှာ S.T.A.R method ကို အသုံးပြုပြီး ဖြေကြားသည့် အခါမှာ ပိုမို အဆင်ပြေပါလိမ့်မယ်။

S.T.A.R ဆိုတာကတော့

- S : Situation
- T : Task
- A : Action
- R : Result

မြန်မာလို ရှင်းပြရရင် ဘယ်လို အခြေအနေ ၊ ဘယ်လို task ကို ဘယ်လို action ယူခဲ့တယ်။ အဲဒီ အတွက် ဘယ်လို result ထွက်လာတယ် ဆိုတာကို ဖြေဆိုသည့် စနစ်ပါ။

ဥပမာ။ မင်း pressure ခံနိုင်လား လို့ မေးလာခဲ့ရင် ၊ ဟုတ်ကဲ့ ခံနိုင်ပါတယ်လို့ ဖြေမယ့် အစား

ဘယ် project မှာ ဘယ် task ကို ဆောက်ရွက်ခဲ့ရပါတယ်။ အဲဒီ အခါမှာ pressure တွေ ဘယ်လောက်များပါမယ်။ အဲဒီ အတွက် ဒီလို action တွေ လုပ်ခဲ့သည့် အခါမှာ ဒီ project ဟာ pressure တွေ အောက်မှာ အောင်အောင်မြင်မြင် ပြီးခဲ့ပါတယ် ဆိုပြီး ဖြေဆိုရသည့် ပုံစံပါ။

ဒါဆိုရင် ကိုယ်ဟာ အတွေ့အကြုံရှိခဲ့ဖူးကြောင့် ဖြေရှင်းနိုင်ကြောင်း တာဝန်ယူနိုင်ကြောင်း တို့ကို ဖော်ပြပြီးသား ဖြစ်ပါတယ်။

Round By Round

ပုံမှန် အားဖြင့် interview တွေဟာ တဆင့်ပြီး တဆင့်ခေါ်တာတွေ ရှိပါတယ်။ ပထမ အဆင့် ဖုန်းနဲ့ အင်တာဗျူး ပါတယ်။ Screening interview အဆင့်ပါ။ ပြီးရင် algorithm and problem solving test ကို ဖြေခိုင်းတတ်ပါတယ်။ နောက်တဆင့်ကတော့ platform specific interview ပါ။ ဒီ platform အကြောင်း တကယ်သိမသိ မေးတတ်သည့် interview ပါ။ နောက်ပြီး High Level Diagram တွေ မေးတတ်သည့် interview တွေလည်း ရှိပါတယ်။ သင်ကိုယ်တိုင် ဒီ system ကို ဘယ်လို develop လုပ်မလဲ။ ဘယ်လို ရေးမလဲ ဆိုပြီး မေးတတ်ပါတယ်။ System Analyst skills နဲ့ requirement gathering အား ကောင်းမကောင်း စစ်ဆေးဖို့ အတွက် High Level System Design interview ကို မေးကြတာပါ။

တချို့ interview တွေမှာတော့ personality test တွေ ပါပါတတ်ပါတယ်။ အဆင့်ဆင့် interview တွေဟာ ၁ လ တစ်ခါတစ်လေ ၂ လောက် ကြာတတ်တာတွေ ရှိတတ်ပါတယ်။

ဖိအား ခံနိုင်မှု နှင့် ပြဿနာ ဖြေရှင်းခြင်း

အချို့ company တွေက coding interview ထက် stress ဘယ်လောက် ခံနိုင်လဲ။ ပြဿနာ ကြုံရရင် ဘယ်လို ဖြေရှင်းမလဲ ဆိုတာ သိချင်ကြတယ်။ ဒါကြောင့် ပေါက်ကရ မေးခွန်းတွေ မေးတတ်တယ်။

ဥပမာ။ မနက်ဖြန်ပေးရမယ့် project ဒါပေမယ့် laptop က ပျက်သွားရင် ဘယ်လို လုပ်မလဲ။ မင်း file တွေ မှားဖျက်မိရင် ဘယ်လိုလုပ်မလဲ။ Project က မနက်ဖြန် ပေးရမယ် အခု ထက်ထိ error တက်နေတုန်းဆိုရင် ဘယ်လို ရှင်းမလဲ။

စသည် ဖြင့် စိတ်ထဲ ရှိသည့် မေးခွန်းတွေ တောက်လျှောက်မေးပြီး stress ကို ဘယ်လို handle လုပ်လဲ ပြဿနာကို ဘယ်လို ဖြေရှင်းမလဲ ဆိုတာတွေ မေးတတ်တယ်။

Interview တိုင်းမှာ တည်ငြိမ်ပြီး စကားကို ရှင်းလင်း စွာ ပြောဖို့ လိုတယ်။ စိတ်လှုပ်ရှားတတ်ရင် အလုပ်ရလည်း ဖြစ်တယ် မရလည်း ဖြစ်တယ်လို့ စိတ်ကို ထားပြီး တည်ငြိမ်စွာ ဖြေဖို့ လိုတယ်။ Stress တွေ များလာရင် အမှားတွေ ပြောတတ်တယ်။ ဒေါသတွေ ထွက်လာတတ်တယ်။ ဒါတွေကို သတိထားဖို့ လိုတယ်။ လုပ်ငန်းခွင် ရောက်ရင် မှားရင် ခေါ်ပြီး သတိပေးခံ ရမှာ ဖြစ်သည့် အတွက် stress ကို ထိန်းနိုင်ဖို့ လိုတယ်။

အမှန်တိုင်းပြောပါ

Interview မှာ ကိုယ်မလုပ်ထားသည့် project တွေကို မပြပါနဲ့။ အကယ်၍ တစိတ်တပိုင်း ပဲပါခဲ့ရင် ကိုယ်ပါဝင် လုပ်ဆောင်ခဲ့သည့် အပိုင်းကို ဖော်ပြထားဖို့ လိုပါတယ်။ မလုပ်ထားခဲ့တာတွေကိုတော့ resume ထဲလည်း ထည့်မရေးပါနဲ့။ အခု လိမ်လို့ ရပေမယ့် လုပ်ငန်းခွင် မှာ လိမ်လို့ မရပါဘူး။ အဲဒီအခါ ကိုယ့်လုပ်ဖော်ကိုင်ဖက်တွေက ကိုယ့်ကို အထင်သေးပါလိမ့်မယ်။

လေ့ကျင့်ထားဖို့

Interview တွေက မေးခွန်းမေးရင် တစ်ခုနဲ့ တစ်ခု သိပ်ပြီးကွာခြားမှု မရှိဘူး။ ဒါကြောင့် မေးခွန်းတွေကို ဘယ်လို ဖြေရမလဲဆိုတာကို ရှာပြီး လေ့ကျင့်ထားဖို့ လိုပါတယ်။

မေးတတ်သည့် မေးခွန်းတွေက

- ကိုယ့်ကိုယ်ကို မိတ်ဆက်ပါ
- ဘာကြောင့် ဒီ company ကို ရွေးချယ်ရတာလဲ
- ကိုယ့်ရဲ့ အားသာချက်
- ကိုယ့်ရဲ့ အားနည်းချက်
- နောက် ၅ နှစ်မှာ ဘာဖြစ်ချင်လဲ

ဒီမေးခွန်းတွေက ပုံမှန် မေးနေကြမေးခွန်းပါ။ ဒီမေးခွန်းတွေကို ဘယ်လိုဖြေသင့်သလဲဆိုတာ company တစ်ခု နဲ့ တစ်ခု တူမှာမဟုတ်ဘူး။ ဒါပေမယ့် ကိုယ့်ရဲ့ အဖြေတိုင်းမှာ ကိုယ်လျောက်မယ့် position အတွက် အထောက်အပံ့ ဖြစ်နေဖို့လိုတယ်။

မိတ်ဆက်သည့် အခါမှာ ကိုယ်လျောက်မယ့် position နဲ့ သက်ဆိုင်သည့် project တွေပဲ ပြောတာ မျိုးပေါ့။ အခြားလုပ်ဖူးတာတွေ ပြောနေရင်လည်း စိတ်ဝင်စားမှု ရှိမှာမဟုတ်ပါဘူး။

ကိုယ့်ရဲ့ အားနည်းချက်ကို ဖြေရတာဟာ အခက်အခဲဆုံးပဲ။ အများအားဖြင့် အလုပ်နဲ့ မဆိုင်တာတွေပဲ ဖြေဆိုတတ်ကြတယ်။ ဖြစ်သင့်တာက အလုပ်နဲ့ ဆိုင်တာ မျိုးမှာ အားနည်းချက်ကို သိချင်တာပါ။

ဥပမာ project တစ်ခုပြီးသွားရင် နောက်ထပ် project ကို ချက်ခြင်းမသွားနိုင်တာ ကျွန်တော့်ရဲ့ အားနည်းချက်ပါ။ ပြီးသွားသည့် project မှာ ဘာ bugs တွေ ရှိနေသေးလည်း ဘာတွေ ထပ်ပြင်သင့်တယ်။ code ကို ဘယ်လိုပြန်ပြင် သင့်တယ်ဆိုတာကို ပြန်ပြန်စစ်နေတော့ နောက်ထပ် project တစ်ခုကို မလုပ်နိုင်ဘူး ဖြစ်နေတယ်။ ဒီအားနည်းချက်ကို ပြင်ဆင်နိုင်အောင် ကြိုးစားနေပါတယ်။ အခု လုပ်ငန်းခွင်ထဲမှာ အဲလိုမဖြစ်အောင် ကြိုးစားပါမယ်။ project တစ်ခု ကနေ နောက်တစ်ခု ချက်ခြင်းပြောင်းပြီး လုပ်နိုင်အောင် လုပ်ပါမယ်။

ဒါဆိုရင် အလုပ်နဲ့လည်း သက်ဆိုင်သလို ကိုယ့်အားနည်းချက်က company အတွက် ထိခိုက်မှု ကြီးကြီးမားမား မရှိဘူး ဆိုတာကို ပေါ်လွင်စေပါတယ်။

နောက် ၅ နှစ် မှာ ဘာဖြစ်လဲ ဆိုပြီး မေးခွန်းရဲ့ တကယ့် မေးခွန်း အစစ်ကတော့ ငါတို့ဆီ မှာ ၅ နှစ် လောက် ကြာကြာလုပ်မှာလား လို့ မေးတာပါ။ interview ဖြေနေကြမဟုတ်သည့်သူတွေကတော့ ကိုယ်ဖြစ်တာတွေ ပြောတတ်ပါတယ်။

ဖြေသင့်တာက company မှာ ၅ နှစ်လောက် ရှိနေပါမယ်။ company မှာ senior developer အနေနဲ့ ရှိနေချင်ပြီးတော့ junior တွေကို coach လုပ်ပြီး project တွေကို အရင်ကထက် ပိုပြီး သေချာ သပ်ရပ်အောင် လုပ်နိုင်မယ် လို့ ထင်ပါတယ်။

အခန်း ၄ :: Markdown

Markdown ကတော့ lightweight markup language တစ်ခု ဖြစ်ပြီး John Gruber နှင့် Aaron Swartz က ၂၀၀၄ မှာ ဖန်တီးခဲ့တာပါ။ Markdown ဟာ blog, forum , documentation, readme စတာတွေကို လွယ်လင့်တကူ ရေးသားနိုင်အောင် ကူညီပေးသည့် markup language တစ်ခုပါ။

Developer တစ်ယောက် အနနဲ့ မဖြစ်မနေ Markdown ကို သိထားသင့်တယ်။ Markdown ရဲ့ အားသာချက်က HTML ကို လွယ်လင့် တကူ ပြန်ပြောင်းနိုင်တာပါ။ ပုံမှန် HTML နဲ့ ရေးသည့် အခါမှာ စာတွေ အများကြီး ရေးရပေမယ့် Markdown က ရေးရတာ လျော့ချပေးတယ်။ အခု စာအုပ်ဟာလည်း Markdown နဲ့ ရေးသားထားပြီးတော့ HTML, PDF,Epub တို့ကို ပြန်ထုတ်ထားတာပါ။

Example markdown လေးကို ကြည့်ရအောင်

```
# Header
## Header 2

This is paragrapth with **bold** and *italic*.

[Link](https://www.google.com) to Google.

![Sample](<./images/file.png)
```

အဲဒီ code ကို HTML ပြန်ပြောင်းလိုက်ရင် အောက်ပါ လို မြင်ရပါလိမ့်မယ်။

```
<h1>Header</h1>

<h2>Header 2</h2>

<p>This is paragrapth with <strong>bold</strong> and <em>italic</em>.</p>

<p><a href="https://www.google.com">Link</a> to Google.</p>

<img src="./images/file.png" alt="Sample" />
```

ဆိုပြီး ထွက်လာပါမယ်။ ပုံမှန် HTML ရေးနေတာထက် ပိုပြီး မြန်မြန်ဆန်ဆန် ရေးလို့ရသည့် အတွက် document တွေ ရေးသည့် အခါမှာ အသုံးဝင်ပါတယ်။ Github, Gitlab တို့မှာ README.md ဆိုသည့် text file ကို HTML အနေနဲ့ repo ရဲ့ README အနေနဲ့ ဖော်ပြပေးပါတယ်။ ဒါကြောင့် Git Repo တွေမှာ README.md ကို ထည့်သွင်းပြီး markdown နဲ့ ရေးသားပါတယ်။

Syntax

Header

Markdown မှာ header အတွက်

```
# This is an H1

## This is an H2

### This is an H3

###### This is an H6
```

နောက်ပြီးတော့

```
This is H1
==========

This is H2
----------
```

ဆိုပြီးလည်း သုံးနိုင်ပါတယ်။

Blockquotes

Markdown မှာ blockquote အတွက် `>` ကို အသုံးပြုပြီး ရေးပါတယ်။

```
> This is a blockquote with two paragraphs. Lorem ipsum dolor sit amet,
> consectetuer adipiscing elit. Aliquam hendrerit mi posuere lectus.
> Vestibulum enim wisi, viverra nec, fringilla in, laoreet vitae, risus.
>
> Donec sit amet nisl. Aliquam semper ipsum sit amet velit. Suspendisse
> id sem consectetuer libero luctus adipiscing.
```

အဲဒီလို စာဆိုရင်တော့

> This is a blockquote with two paragraphs. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Aliquam hendrerit mi posuere lectus. Vestibulum enim wisi, viverra nec, fringilla in, laoreet vitae, risus.
>
> Donec sit amet nisl. Aliquam semper ipsum sit amet velit. Suspendisse id sem consectetuer libero luctus adipiscing.

အခုလိုမျိုး paragraph အထဲကို ဝင်သွားသည့် ပုံစံ မျိုးကို မြင်ရပါလိမ့်မယ်။ Multiple level အတွက် `> >` ကို အသုံးပြုနိုင်ပါတယ်။

```
> This is the first level of quoting.
>
> > This is nested blockquote.
>
> Back to the first level.
```

ဆိုရင်တော့

> This is the first level of quoting.
>
> > This is nested blockquote.
>
> Back to the first level.

လိုမျိုး မြင်ရမှာပါ။

List

List အတွက် markdown မှာ လွယ်လွယ်ကူကူ ရေးနိုင်ပါတယ်။ Unorder list အတွက်

```
* Red
* Green
* Blue
```

```
+ Red
+ Green
+ Blue
```

```
- Red
- Green
- Blue
```

`*` , `+` , `-` ကြိုက်တာကို အသုံးပြုနိုင်ပါတယ်။

* Red
* Green
* Blue

ဆိုပြီး unorder list နဲ့ ပြပေးပါလိမ့်မယ်။

Ordered list အတွက်

```
1.  Bird
2.  McHale
3.  Parish
```

ဆိုပြီး ရေးနိုင်ပါတယ်။

```
*   Bird
*   Magic
```

ဆိုသည့် list ဟာ

```
<ul>
<li>Bird</li>
<li>Magic</li>
</ul>
```

Code Blocks

Markdown မှာ အကြိုက်ဆုံး feature တစ်ခုကတော့ code blocks ပါပဲ။ ပုံမှန် code ရေးသည့်အခါ မှာ HTML မှာ pre tag တွေ နဲ့ အသုံးပြုရပေမယ့် markdown မှာတော့ လွယ်ကူပါတယ်။

```
This is normal paragraph:
        This is code block
```

အခုလိုမျိုး tab ကိုထည့်လိုက်ရင်

```
<p>This is a normal paragraph:</p>

<pre><code>This is a code block.
</code></pre>
```

ဆိုပြီး generate လုပ်ပါလိမ့်မယ်။

နောက်တစ်မျိုးကတော့

```
```php
        <?php
                echo "hello";
```

အဲဒီမှာ code အတွက် \` ကို သုံးထားတာ တွေ့နိုင်ပါတယ်။ \`\`\` ဟာ code block ကို ရည်ညွှန်းပါတယ်။

\`\`\`php ဆိုရင်တော့ php code block ဖြစ်တယ် ဆိုပြီး ရည်ညွှန်းထားတာပါ။

အဲဒီ code ကို html ပြောင်းရင်တော့

```
```html
<pre><code class="php">        &lt;?php
            echo &quot;hello&quot;;
</code></pre>
```

ဆိုပြီး ပြောင်းသွားပါမယ်။

အကယ်၍ code ကို `echo` လိုမျိုး inline မှာ သုံးချင်ရင်တော့ \ `echo`\ ဆိုပြီး သုံးနိုင်ပါတယ်။ `<code>echo</code>` အနေနဲ့ generate လုပ်ပေးပါလိမ့်မယ်။

Horizontal Rules

Markdown မှာ `<hr>` ကို အသုံးပြုချင်ရင် `***` ကို အသုံးပြုနိုင်ပါတယ်။ အသုံးပြုနိုင်တာတွေကတော့

```
***
---
```

```
----------
```

စတာတွေ ကို အသုံးပြုနိုင်ပါတယ်။

Links

HTML မှာလိုမျိုး URL ကို ချိတ်ချင်သည့် အခါမှာ `<a href='link'>TEXT</a>` လိုမျိုး အတွက် Markdown မှာ အောက်ပါအတိုင်း ရေးနိုင်ပါတယ်။

```
This is [Text](http://example.com/) inline link.
```

အကယ်၍ tooltip ပေါ်ချင်သည့် အခါမှာ

```
[Text](http://example.com/ "HELLO World")
```

ပုံစံ ရေးနိုင်ပါတယ်။

အဲဒီ အခါမှာတော့

```
<a href="http://example.com/" title="HELLO WORLD">Text</a>
```

Bold and Italic

အထက်မှာ ပြောထားခဲ့ဖူးပါတယ်။ Bold အတွက်ကတော့

```
**bold**
```

ဆိုရင်

```
<b>bold</b>
```

ဆိုပြီး ပြောင်းပေးပါတယ်။

```
_italic_
```

ဆိုရင်

```
<i>italic</i>
```

Images

Markdown မှာ ပုံထည့်မယ် ဆိုရင်အောက်ပါ အတိုင်း ထည့်နိုင် ပါတယ်။

```
![Alt text](</path/to/img.jpg>)

![Alt text](</path/to/img.jpg "Optional title">)
```

ဆိုရင်

```
<img src="/path/to/img.jpg" alt="Alt text"/>

<img src="/path/to/img.jpg" alt="Alt text" title="Optional title" />
```

အကယ်၍ internet က url အသုံးပြုမယ်ဆိုရင်

```
![Alt text](<http://www.mywebsite.com/myimage.jpg>)
```

ဆိုရင်

```
<img src="http://www.mywebsite.com/myimage.jpg" alt="Alt text" />
```

Tool

Markdown ကို ပုံမှန် text editor တစ်ခုခု နဲ့ ရေးရင် ရပါတယ်။ သို့ပေမယ့် syntax highlighitng, (HTML, PDF, Doc) စသည် တို့ကို export လုပ်လို့ ရအောင် editor တွေကိုလည်း အသုံးပြုနိုင်ပါ တယ်။

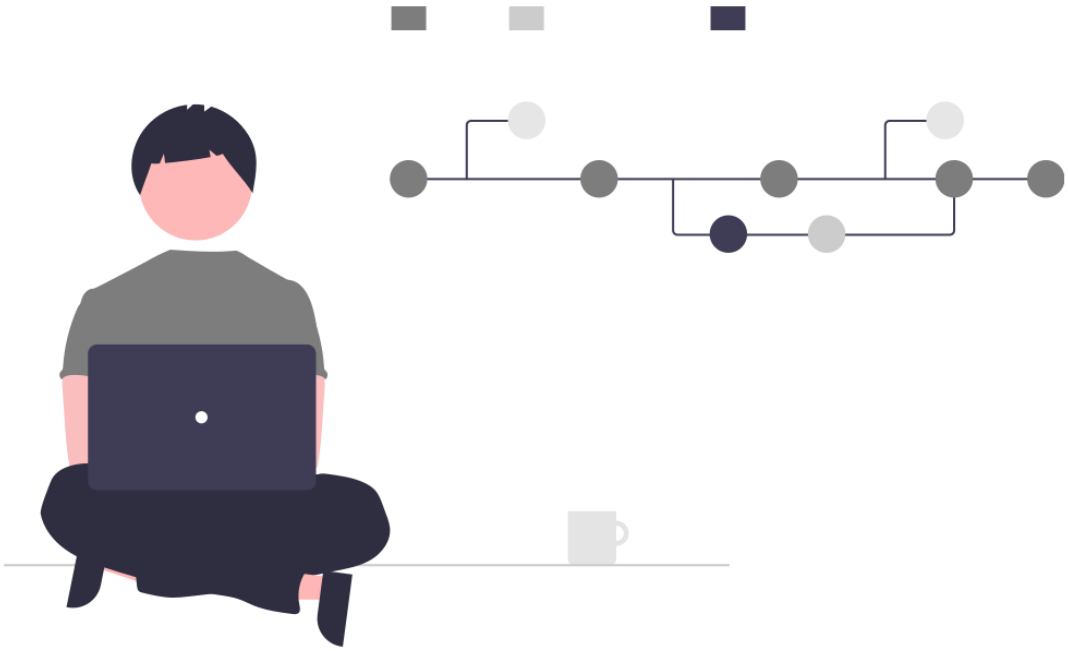
Editor တွေကတော့

Editor	Syntax Highlight	Preview	Export PDF	Export Doc	Export HTML
VS Code	O	O	Plugin	Plugin	Plugin
iA Writer (Mac)	O	O	O	O	O
Typora	O	O	O	O	O
MarkdownPad (Windows)	O	O	O	x	O

markdownmonster (Windows)	O	O	O	x	O
Apostrophe (linux)	O	O	x	x	O

အခု ဆိုရင်တော့ Markdown ကို သုံးပြီးတော့ README.md file ကို ရေးနိုင်ပါလိမ့်မယ်။

အခန်း ၅ :: Git



Developer တိုင်း git ကို မဖြစ်မနေ သိထားသင့်ပါတယ်။ အခုခေတ်မှာ Git မသိရင် developer တစ်ယောက် အနေနဲ့ ရပ်တည်ဖို့က ခက်ပါတယ်။ ဒီစာအုပ်မှာတော့ git setup လုပ်တာတွေ install လုပ်တာတွေ ဖော်ပြမနေတော့ပါဘူး။

Git Hosting

Git အတွက် cloud hosting ဆိုရင်

- Gitlab
- Github
- Bitbucket

စတာတွေ လူသုံးများပါတယ်။ gitlab ကတော့ enterprise အတွက် လူသုံးများပါတယ်။ Free unlimited private repo ရသည့်အတွက် မြန်မာနိုင်ငံက company တော်တော်များများ အသုံးပြုကြပါတယ်။

တကယ်တန်းကတော့ git က hosting မလိုပဲ အလုပ်လုပ်နိုင်ပါတယ်။ Git ကို server မှာ မလိုပဲ share network မှာ တင် ဖန်တီး နိုင်ပြီး အလုပ်လုပ်နိုင်ပါတယ်။ Production deployment တွေ အတွက် Git နဲ့ အတူ CI/CD တွေ git hook နဲ့ တွဲပြီး အသုံးပြုကြသည့် အတွက် local မှာ ထက် Gitlab လိုမျိုး server တွေ မှာ ပို အဆင်ပြေပါတယ်။

Repository အသစ် ဆောက်ခြင်း

Repository ကို repo လို့ အတို ခေါက် ခေါ်ပါတယ်။ Git source code တွေ ထားထားသည့် project နေရာပေါ့။ Gitlab မှာ Repo တစ်ခု ကို ဆောက်ဖို့ လိုပါတယ်။ ပြီးရင် ကိုယ့် စက်ထဲလည်း repo ဆောက်ဖို့ လိုပါတယ်။

စက်ထဲမှာတော့ project path မှာ command line ကနေ

```
git init
```

ဆိုတာနဲ့ ရပါတယ်။

Git Server ဘက်ကနေ repo ဆောက်ပြီး တာနဲ့ git url တစ်ခု ရပါမယ်။

```
git remote add origin [my git https url]
git pull origin master
```

Git ကို စပြီး လေ့လာကာစ သူတွေ အနေနဲ့ HTTPS ကို သုံးစေချင်ပါတယ်။ SSH ဟာ windows အတွက် အဆင်မပြေပါဘူး။

အကယ်၍ git repo တစ်ခုလုံးကို အပေါ်ကလို setup မလုပ်ပဲ git clone လုပ်ရင်လည်း အဆင်ပြေပါတယ်။

```
git clone [my git https url]
```

Branch နှင့် Checkout

Git ဟာ branch နဲ့ ရှိပါတယ်။ Project တစ်ခု ကို ဖန်တီးသည့် အခါမှာ

- Master
- Dev
- Hotfix/task-123
- uat

- staging

စသည် ဖြင့် branch တွေ ခွဲပြီး ရေးကြပါတယ်။

Master ကတော့ ပုံမှန် အားဖြင့် production မှာ ရှိနေသည့် အတိုင်း ရှိနေသည့် code တွေပါ။

Dev ကတော့ နေ့စဉ် development လုပ်ထားသည့် code တွေပါ။

Hotfix/task-123 ကတော့ Hotfix အောက်မှာ urgent bug fix တာတွေကို task နဲ့ ခွဲထုတ်ပြီး သိမ်းထားတာပါ။ ပြီးရင် master , uat စသည်တို့ နဲ့ ပြန် ပြီး merge ပါတယ်။

uat ကတော့ uat server အတွက် ပါ။ client တွေ စမ်းဖို့ ready ဖြစ်သည့် code ပေါ့။

staging ကတော့ staging server အတွက်ပါ။ QA တို့ project manager တို့ စမ်းဖို့ ready ဖြစ်သည့် code တွေ ထားသည့် branch ပါ။

Git ကို အခု လို branch တွေ ခွဲထားသည့် အတွက် code တွေမှာ bugs တစ်ခုခု ရှိခဲ့ရင်လည်း roll back ပြန်လုပ်လို့ရပါတယ်။ Git မသုံးသည့် အခါမှာ code တွေ အတွက် folder တွေ ခွဲ သိမ်းရပါလိမ့်မယ်။ producation မှာ တစ်ခုခု ဖြစ်သွားရင် roll back အတွက်လည်း အတော်လေးကို လက်ဝင်ပါတယ်။

Git မှာ branch ကို checkout လုပ်မယ် ဆိုရင်

```
git branch -a
```

နဲ့ ကြည့်နိုင်ပါတယ်။ အကယ်၍ ကိုယ် သုံးမယ့် branch ကို pull မဆွဲရင် သေးရင်တော့

```
git fetch origin
git branch
```

ဆိုပြီး server မှာ ရှိသည့် branch တွေကို ဆွဲချနိုင်ပါတယ်။
ပြီးရင်တော့

```
git checkout [branch]
```

ဆိုရင် လက်ရှိ အသုံးပြုချင်သည့် branch ကို ရောက်သွားပြီး စတင်ပြီး ရေးလို့ရပါပြီ။

Add Files

Changes တွေပြီးသွားရင် repo ပေါ်ကို ပြန်တင်ဖို့ အတွက်

```
git add .
```

ဆိုရင် changes တွေ အကုန် တင်မယ် လို့ ဆိုပါတယ်။ File ရွေးပြီး တင်ချင်ရင်တော့

```
git add myfile_name
```

ပြီးရင်

```
git commit -m 'commit message'
```

ဆိုပြီး ဘာတွေ ပြင်ထားတယ်။ ဘာအတွက် push လုပ်တယ် ဆိုတာကို commit ထဲမှာ ရေးပါတယ်။

Commit message က အရေးကြီးပါတယ်။ Developer တော်တော်များများက fixed , bugs fixed စသည် ဖြင့် ရေးတတ်ပါတယ်။ ပိုမို ပြည့်စုံသည့် meaing ဖြစ်ဖို့ လိုပါတယ်။ ဥပမာ fixed for login လိုမျိုးပေါ့။

Push code

Commit ပြီးရင်တော့ push လုပ်လို့ရပါပြီ။

```
git push origin [your branch]
```

master branch ကို push လုပ်မယ် ဆိုရင်တော့

```
git push origin master
```

နဲ့ ပြန်ပြီး push လုပ်နိုင်ပါတယ်။ push လုပ်ပြီးသွားရင် git server ပေါ် ရောက်သွားပါပြီ။

Git Merge

လက်ရှိ branch က အခြား branch ကို merge လုပ်မယ်ဆိုရင် git merge ကို သုံးရပါတယ်။

ဥပမာ လက်ရှိ master branch ကနေ uat က code ကို merge လုပ်မယ်ဆိုရင်

```
git merge uat
```

ဆိုပြီး သုံးနိုင်ပါတယ်။

Git Pull

Git commit လုပ်မယ်ဆိုရင် လက်ရှိ repo က နောက်ဆုံး commit မှာ ဖြစ်နေဖို့ လိုပါတယ်။ Server ပေါ်က changes တွေကို local မှာ effect ဖြစ်ဖို့ အတွက် pull လုပ်ပေးရပါတယ်။

```
git pull origin [branch name]
```

လက်ရှိ က master branch ဆိုရင်

```
git pull origin master
```

Tagging

Git မှာ နောက်ထပ် အရေးပါပြီး လူသုံးများသည့် feature ကတော့ tag ပါ။ Production မှာ launch လုပ်ပြီးသွားရင် ပုံမှန် version number နဲ့ tag လုပ်ပါတယ်။

```
git tag -a v1.1 -m 'version 1.1'
```

tag v1.1 ဖြစ်ပြီး message ကိုတော့ version 1.1 လို့ ပေးထားပါတယ်။

push လုပ်သည့် အခါမှာလည်း tags ပါ တွဲပြီး push လုပ်ဖို့ လိုပါတယ်။

```
git push origin master --tags
```

Git Conflict

Git မှာ အရေးကြီးသည့် နောက်တစ်ခုကတော့ conflict ဖြစ်တာပါပဲ။ Git Conflict တွေကို Developer တွေ အနေနဲ့ သေချာရှင်းဖို့ လိုပါတယ်။ Merge Tool တစ်ခုခုကို သုံးပြီး changes တွေကို ပြန်ကြည့်ပါတယ်။ တစ်ခါတစ်လေ ယ်သူ commit လုပ်ထားသည့် code လည်း ဘာ changes တွေ ရှိကုန်တာလဲ ဆိုတာကို commit လုပ်ထားသည့် developer ကို ပြန်မေးပြီးတော့ merge လုပ်ရတာ တွေ ရှိပါတယ်။

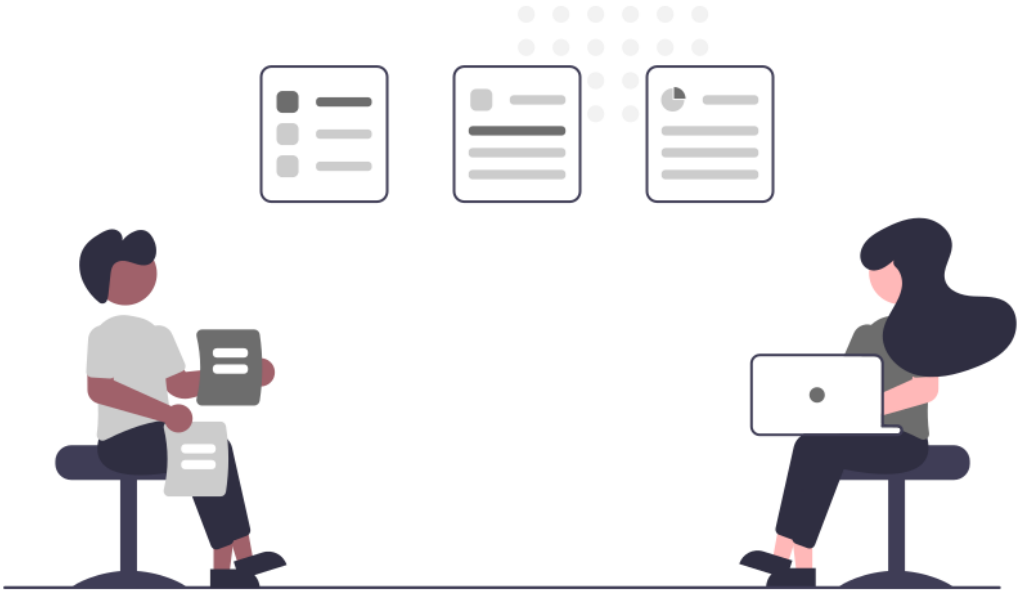
GUI

Git အတွက် GUI ထဲမှာတော့ Source Tree ကို recommend လုပ်ပါတယ်။ အခြား GUI တွေ ရှိပါသေးတယ်။

- GitKraken
- Tower
- Sublime Merge
- GitAhead

စသည် ဖြင့် ရှိပြီး https://git-scm.com/downloads/guis မှာ ကြည့်ရှုနိုင်ပါတယ်။

အခန်း ၆ :: Meeting



အစည်းအဝေး (Meeting) ဆိုတာ နဲ့ developer တော်တော်များများ စိတ်ညစ် ကြပါတယ်။ အဓိကတော့ ကိုယ် နဲ့ မသက်ဆိုင်တာ တွေ အမြဲပါပြီး အချိန် ကုန် သလို ခံစားရတယ်။ အစည်းအဝေး တော်တော်များများဟာ အဖြေထွက် ဖို့ အချိန် အတော်ပေးရတာတွေ ကိုယ့်ဘက် သူ့ဘက် ဆွေးနွေးခြင်းက အချိန်ကြာတတ်ပါတယ်။

ပုံမှန် အားဖြင့် meeting တစ်ခုမှာ

- ဘာအတွက် meeting လဲ
- အချိန် ဘယ်လောက် ပေးရမလဲ
- Agenda က ဘာတွေလဲ
- ဘယ်သူတွေ ပါမှာလဲ
- ပါသည့် သူတွေက ဘယ် department က ဖြစ်ပြီး အခု meeting နဲ့ ဘာတွေ ပတ်သက် လဲ

စတာတွေကို သိထားဖို့ လိုပါမယ်။

Meeting ဟာ တကယ်တန်းတော့ အချိန် အရမ်းကုန်ပါတယ်။ မလိုအပ်သည့် ကိစ္စတွေ တခြား ကိစ္စတွေ ပါလာရင် meeting ရဲ့ အဓိက အချက်အလက်တွေ နဲ့ လွဲကုန်တတ်ပါတယ်။ အထူးသဖြင့် Meeting မှာ ဦးဆောင် ဆွေးနွေးသည့် သူဟာ CEO ဖြစ်နေရင် လုပ်ရမယ့် အဓိက meeting ကနေ နောက်ထပ် project တွေ လုပ်မယ့် အထိ ဖြစ်တတ်သလို ဟိုဘက်ဒီဘက် CEO တွေဖြစ်နေရင် meeting အကြောင်း မပြောရပဲ အ လာပ သ လာပ စကားတွေ နဲ့ အချိန်ကုန်သွားတတ်တာ သတိပြုသင့်ပါတယ်။

Meeting ကို lead လုပ်မယ့်သူဟာ agenda အတိုင်း ဖြစ်အောင် ချိန်ညှိပေးဖို့ လိုတယ်။ ဦးတည် ချက်ပြောင်းသွားရင် ခေါင်းစဉ် ပြောင်းသွားရင် ပြန်ပြီး ဆွေးနွေးနေသည့် ခေါင်းစဉ် ပြန်ရောက် အောင် ခေါ်ဆောင်ရတာတွေလည်း ရှိပါတယ်။

Meeting မှာ အဓိက အားဖြင့်

- ဘယ်သူတွေ ဘာလုပ်နေသလဲ
- ဘာပြဿနာတွေ ဖြစ်နေလဲ
- ဘယ်နားမှာ ရပ်နေလဲ
- ဘယ်လို ဖြေရှင်း မလဲ

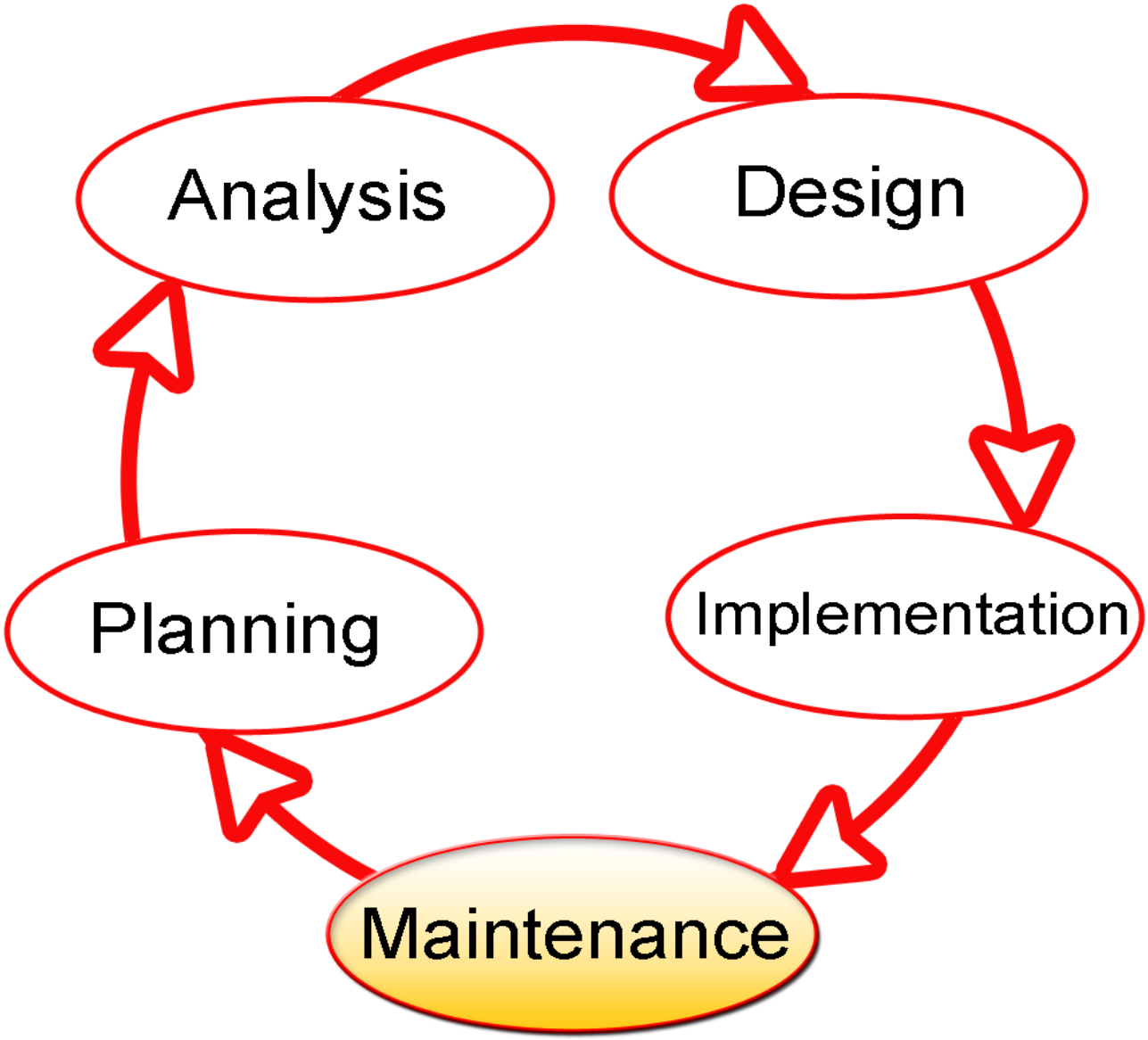
ဆိုပြီး team တစ်လုံး သိအောင် အဖြေရှာရခြင်းပါပဲ။

Meeting တွေ ကို အများအားဖြင့် Daily, Weekly, Monthly စသည်ဖြင့် ရှိတတ်ပါတယ်။ Meetting တော်တော်များများကို Developer တွေက ကိုယ့်အပိုင်း မဟုတ်တော့ရင် စိတ်မပါကြတာများပါ တယ်။ နောက်ပြီး meeting တစ်ခု အတွက် prepare လုပ်ရတာ developer တစ်ယောက်အတွက် code ရေးသည့် အချိန်ကနေ ဖဲ့ပြီး ပေးရသည့် အတွက် timeline က ပြဿနာ ဖြစ်တတ်တယ်။ Developer တွေက tasks တစ်ခု ကြာချိန်ကို ပေးသည့် အခါမှာ အလုပ်လုပ်သည့် အချိန် ပဲ ထည့်ပြီး ပေးတတ်တယ်။ အလုပ်ထဲမှာ meeting တွေ requirement မရှင်းလင်းသည့် အခါမှာ ပြန်ပြီး confirm လုပ်ရတာတွေ ရဲ့ risk ကို ထည့်မတွက်မိကြဘူး။ meeting တွေ များများ တက်ရလေ လေ timeline နဲ့ မကိုက်လေလေ ဖြစ်တတ်တာကို project manager တွေ အနေနဲ့လည်း သဘောပေါက်ဖို့ လိုပါတယ်။

Meeting တစ်ခုကို အများအားဖြင့် ဦးဆောင်ရတာကတော့ project manager တွေ များပါတယ်။ project manager ရဲ့ skills က meeting ကို တိုတို နဲ့ လိုရင်း ဖြတ်နိုင်ဖို့ အရမ်းအရေးပါပါတယ်။ Developer တိုင်းအတွက် အချိန်ဟာ အရမ်းတန်ဖိုး ကြီးပါတယ်။ Developer တွေအနေနဲ့လည်း meeting မရှိပဲနဲ့ project ကို develop လုပ်သည့် အခါမှာလည်း ပြီးမှ requirement လွဲတာတွေ မ ကိုက်တာတွေ ဖြစ်တတ်ပါတယ်။ဒါကြောင့် meeting note ကလည်း အရေးပါပါတယ်။ ဒီ tasks ကို ဘယ်သူက ဘယ် meeting က ပြောခဲ့လို့ လုပ်ခဲ့ရပါတယ်ဆိုသည့် သက်သေ အထောက်အထား ခိုင်မာ ဖို့က meeting note က အဓိက ကျပါတယ်။ မြန်မာနိုင်ငံမှာ meeting note အလေ့အထ နည်းသလို developer တွေ အနေနဲ့လည်း လိုက်မှတ်ထားတာ မရှိသလောက်ပါ ပဲ။

အချုပ်ဆိုရင်သော် meeting ဟာ developer တွေ team တွေ အတွက်လိုအပ်ပါတယ်။ Weekly tasks တွေ assign ချဖို့ meeting တွေက အရေးပါပါတယ်။ အလုပ်လုပ်ကြောင်းပြ ဖို့ Meeting နဲ့ တင် အချိန်မကုန် ဖို့ လိုပါတယ်။ အချို့ project manager တွေက သူတို့ အလုပ်လုပ်ကြောင်း ဖော်ပြဖို့ meeting တွေ အရမ်းလုပ်တာ တွေ ရှိတတ်ပါတယ်။ Project တစ်ခု လုံးပြီးဖို့က team တစ်ခုလုံး နဲ့ သက်ဆိုင်တယ်။ Meeting note မှတ် တတ်သည့် အလေ့အကျင့် နဲ့ အချက်အလက် အထောက်ထားနဲ့ ပြန်လည် ပြောဆိုတတ်သည့် အကျင့်ကို ကျွန်တော်တို့ developer တွေ လေ့ကျင့်သင့်ပါတယ်။

အခန်း ၇ :: Software Development Life Cycle



Developer တိုင်း သိပြီး ဖြစ်ပါလိမ့်မယ်။ Developer တစ်ယောက် ဖြစ်လာပြီဆိုရင် SDLC ကို မ ဖြစ်မနေ သိရပါမယ်။

SDLC မှာ ပါသည့် အဆင့်တွေက

- Planning
- Analysis

- Design
- Implementation
- Testing
- Deployment
- Maintenance

တို့ပဲ ဖြစ်ပါတယ်။

Planning

Planing အဆင့်မှာ ကုန်ကျစရိတ် ဘယ်လောက်ရှိမယ်။ လူ ဘယ်နှစ်ယောက် သုံးရမယ်။ Deadline က ဘယ်တော့လောက် ဖြစ်မယ်။ နောက်ပြီး Team ကို ဘယ် team ကို သုံးမယ်။ Team leader က ဘယ်သူ ဖြစ်မယ်။ စသည်ဖြင့် အစီအစဉ် ဆွဲရပါတယ်။

> "What do we want?"

ဒီမေးခွန်းကို ဒီ အဆင့်မှာ မေးဖို့ လိုပါတယ်။ ဒီအဆင့်မှာ stakeholders, customers, developers, subject matter expers တွေ ရဲ့ feedback တွေကို နားထောင်ပြီး လုပ်နိုင်မယ့် team , လိုအပ်သည့် လူ အရေအတွက် ကြာမယ့် အချိန်တွေကို တွက်လို့ရပါမယ်။

Analysis

Project စတော့မယ်။ မစခင်မှာ Requirement တွေ အကုန်စုံပြီလား။ ငါတို့က ဘာကို လုပ်ချင်တာလဲ ဆိုသည့် မေးခွန်းကို ပြန်မေးဖို့ လိုပါတယ်။ Software အတွက် လိုအပ်ချက် နဲ့ ကုန်ကျစရိတ် တကယ့် အစစ် က စီစဉ်ထားသည့် အတိုင်း မှန်ရဲ့လား စစ်ဆေးရပါတယ်။ Project ရဲ့ requirement တွေကို တစ်ခါတည်း Document ပြုလုပ်သွားနိုင်ရင် ပို ကောင်းမွန်ပါတယ်။

Design

> "How will we get what we want?"

ဒီအဆင့်မှာတော့ ကျွန်တော်တို့ လိုချင်သည့် system/project ကို စတင်ပြီး Design ဆွဲရပါတယ်။ Stakeholders တွေက review ကြည့်မယ်။ ပြီးရင် feedback တွေ ပြန်ပေးမယ်။ အဲဒီ ပေါ်မှာ ပြန်ပြင်ရပါတယ်။ ဒီအဆင့်မှာ Stakeholders တွေ ပါဝင်မှု မရှိဘူး ဆိုရင် project တစ်ခု အောင်မြင်ဖို့ ခက်ပါတယ်။ နောက်ပြီး overrun လုပ်မိလို့ ဒီအဆင့်မှာ ကုန်ကျစရိတ် များပြီး fail လည်း ဖြစ်သွားနိုင်တာ ကို သတိပြုရပါမယ်။ ဒီအဆင့်မှာ အကောင်းဆုံး ကို လိုချင်လို့ ထပ်ခါထပ်ခါ ပြင်ခြင်း နဲ့ features အသစ်တွေ ထပ်ပါလာခြင်းဟာ အချိန်တွေ ပိုကုန်နိုင်ပြီး ကုန်ကျစရိတ်တွေ တိုးစေပါတယ်။

Implementation

> "Let’s create what we want."

တကယ့် development ကို စလုပ်ပါပြီ။ ရှေ့ဘက်မှာ ဆောင်ရွက်ထားသည့် Design , Plan တွေ အတိုင်း developer တွေ လိုက်နာ ဖို့ လိုပါတယ်။ Developer တိုင်းဟာ သက်ဆိုင်ရာ langauge အတွက် သတ်မှတ်ထားသည့် code style guide line ကို မဖြစ်မနေ လိုက်နာ ဆောင်ရွက်သင့်ပါတယ်။

Testing

> "Did we get what we want?"

Task တစ်ခု ပြီးတိုင်း task အတွက် test ကို အမြဲ လုပ်နေဖို့လိုပါတယ်။ Unit Testing တွေ ပါ ထည့်သွင်းရေးသား နိုင်ရင် အကောင်းဆုံးပါပဲ။ ကျွန်တော်တို့ ရဲ့ original sepcifications မရမခြင်း fix လုပ်ဖို့ လိုအပ်ပါတယ်။

Task တစ်ခု အမှန်တကယ်ပြီးမြောက်ဖို့ defination of work done ကို သတ်မှတ်ထားဖို့ လိုပါတယ်။ Testing အဆင့်မှာ အမှန်လိုအပ်သည့် requirement နဲ့ မကိုက်ညီရင် tasks က done ဖြစ်တယ်လို့ မသတ်မှတ်နိုင်ပါဘူး။

Deployment

> "Let's start using what we got"

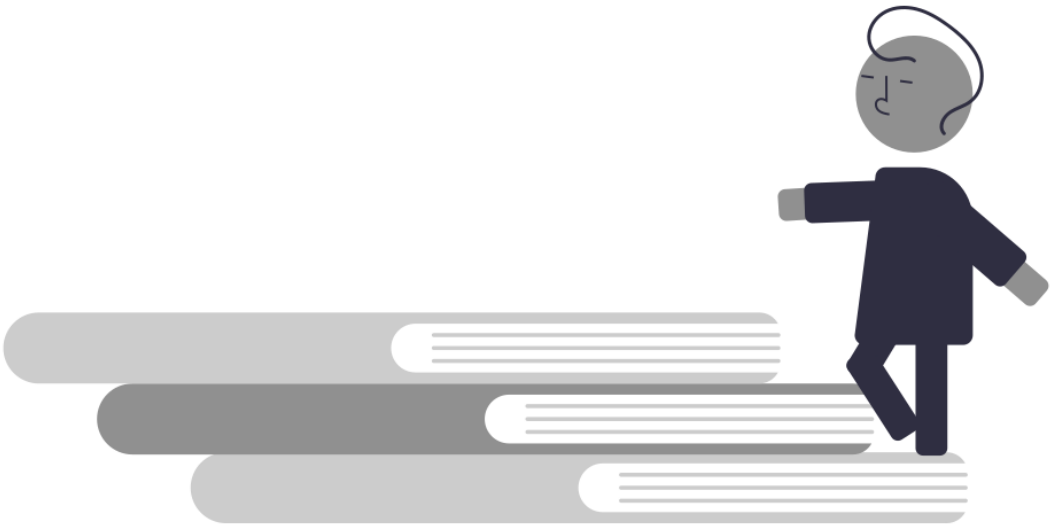
ဒီအဆင့်မှာတော့ production ကို deployment လုပ်ပါမယ်။ Developer တွေက software က launch မလုပ်ပါဘူး။ Deployment ပဲလုပ်ပါတယ်။ Production မှာ deployment လုပ်ပြီး အဆင်ပြေမပြေ ကိုက်ညီမှု ရှိမရှိ stakeholders တွေ စမ်းသပ် သုံးစွဲကြည့်ဖို့ feedback တွေ ရဖို့ လိုအပ်ပါတယ်။ မှားနေတာတွေ ရှိရင် lauch မလုပ်ခင်မှာ ပြန်လည် ပြင်ဆင်ရပါတယ်။

Maintenance

ဘယ် software မဆို maintenance လုပ်ဖို့ လိုအပ်ပါတယ်။ ဥပမာ iOS version အသစ်ထွက်လို့ ပြန်ပြင်ရတာ။ သုံးထားသည့် framework update ဖြစ်လို့ ပြန်ပြင်ရတာ။ Production မှာမှ တွေ့သည့် bugs တွေကို fix ရတာတွေ လုပ်ရပါတယ်။ ဒီဆင့်မှာ ကျွန်တော်တို့တွေဟာ Crash Report system တွေ support tickets တွေ စတာတွေကို အသုံးပြုပြီး maintenance ကို လုပ်ဆောင်ရပါတယ်။

Features အသစ်တွေ ထပ်ဖြည့်တော့မယ်။ အများကြီး အသစ်တွေ ထပ်ဖြည့်မယ်ဆိုရင် maintenance မဟုတ်တော့ဘဲ new major version အတွက် SDLC အစ ကနေ ပြန်စ ပြီး life cycle တစ်ခု ပြန်လည်သွားပါပြီ။

အခန်း ၈ :: Principles



Developer တစ်ယောက် အဖြစ် စတော့မယ် ဆိုရင် Principles တွေကို လိုက်နာခြင်းအားဖြင့် code တွေကို ပိုမို သပ်ရပ်စေပါတယ်။ လူသုံးများသည့် Principles တွေကတော့

- KISS (Keep It Simple, Stupid)
- DRY (Don’t Repeat Yourself)
- YAGNI (You Aren’t Gonna Need It)
- SOLID Principles

ဒီအထဲမှာ အရေးပါဆုံးကတော့ SOLID Princples ပါ။

KISS (Keep It Simple, Stupid)

Software တစ်ခုကို ဖန်တီးရေးသားသည့် အခါမှာ ရိုးရှင်းဖို့ လိုပါတယ်။ ကျွန်တော် junior developer ဘဝ တုန်းက ကိုယ်ရေးထားသည့် code တွေ အခြားသူတွေ နားမလည်ရင် တော်တော်ကောင်းသည့် code လို့ ထင်ဖူးပါတယ်။ ဒါဟာ တကယ်တော့ လုံးဝ မှားယွင်းနေတာပါ။ ကိုယ့် code ကို ဘယ် developer မဆို ဖတ်နိုင်ဖို့ နဲ့ နားလည်လွယ်ကူအောင် အရိုးရှင်းဆုံး ရေးထားမှ ဖြစ်မှာပါ။ သို့ပေမယ့် Code တွေဟာ SOLID principles ကိုတော့ အနည်းဆုံး လိုက်နာ ထားဖို့ လိုပါတယ်။ ကိုယ့်ရဲ့ code တွေဟာ လွယ်ကူစွာ နားလည်ဖို့ လိုတယ် ၊ မလိုအပ်သည့် ရှုပ်ထွေးမှုတွေကို ရှောင်ရှားဖို့ လိုတယ် ၊ လွယ်လွယ်ကူကူ extend လုပ်နိုင်ဖို့ လိုပါတယ်။

DRY (Don’t Repeat Yourself)

ခေါင်းစဥ် ဖတ်လိုက်တာနဲ့ ရှင်းပါတယ်။ ကိုယ်ဟာ junior level မဟုတ်တော့ဘူးလား junior level လား ဆိုတာ ကို စစ်ဖို့ ကိုယ်ရေးထားသည့် project တွေမှာ duplicate code တွေ ရှိနေလား ဆိုပြီး စစ်ကြည့်လိုက်ပါ။ တူညီသည့် code တွေကို ထပ်ခါ ထပ်ခါ မရေးပဲ သက်ဆိုင်ရာ function ဖြစ်စေ class တွေ ဖြစ်စေ ခွဲထုတ်ပြီး ရေးသားထားဖို့ လိုပါတယ်။ The Pragmatic Programmer စာအုပ်ထဲမှာ အောက်ကလို ဖော်ပြထားပါတယ်။

> Every piece of knowledge must have a single, unambiguous, authoritative representation within a system - The Pragmatic Programmer

DRY principle ဟာ code တွေ ကို reusability ဖြစ်ပြီး ထိန်းသိမ်းပြုပြင်ရတာ ပိုမို လွယ်ကူစေပါတယ်။

YAGNI (You Aren’t Gonna Need It)

KISS လိုပါပဲ။ Program တစ်ခုမှာ မလိုအပ်သည့် function ကို မထည့်ပါနဲ့။ တကယ်လိုအပ်တယ် ဆိုမှသာ ထည့်ပါ။ feature တစ်ခု သို့မဟုတ် function တစ်ခု ထည့်ဖို့ တကယ်လိုအပ်ပြီဆိုမှ ထည့်သွင်းဖို့ပါပဲ။

> do the simplest thing that could possibly work

SOLID

Programmer တိုင်း မဖြစ်နေ သိကို သိရမည့် principle ပါ။ အရမ်းကို အသုံးဝင်ပြီး အတတ်နိုင်ဆုံး programmer တိုင်း လိုက်နာကြပါတယ်။ SOLID အရှည်ကောက်ကတော့

- Single Responbility Principle
- Open Close Principle
- Liskov Substitution Principle
- Interface Segregation Principle
- Dependency Inversion Principle

Single Responbility Principle (SRP)

> A class should have one, and only one, reason to change.

Class တစ်ခုဟာ အလုပ်တစ်ခု ကို ပဲ လုပ်သင့်ပါတယ်။ ဥပမာ ပစ္စည်းတစ်ခုမှာ တူ နဲ့ ဝက်အူလှည့် အတူတူ တွဲထားတာ ထက် သီးသန့်ခွဲထားတာ ပို ပြီး အလုပ်ဖြစ်ပါတယ်။ တူ က တူ အလုပ်လုပ်ပြီး ဝက်အူလှည့် က သူ့ အလုပ်သူလုပ်ဖို့ပါပဲ။

ဥပမာ ကြည့်ရအောင်

```
class Book {
        public function save() {
        }

        public function update() {
        }

        public function share() {
        }
}
```

ဒီ code မှာ ဆိုရင် share ဆိုသည့် function ဟာ Book class နဲ့ တိုက်ရိုက် တိုက်ဆိုင် ခြင်း မရှိပါဘူး။ Book Class မှာ share function မလိုအပ်ပါဘူး။

```
class ShareService {
        public function share() {
        }
}
```

share function ကို သီးသန့် ခွဲထုတ် ရေးဖို့ ShareService ကို ခွဲရေးလိုက်ပါမယ်။ ဒါဆိုရင် Book class မှာ စာအုပ် နဲ့ ဆိုင်သည့် responsibility ပဲ ရှိပါတော့တယ်။

Open Close Principle (OCP)

> Software entities (classes, modules, functions, etc.) should be open for extension, but closed for modification

Object ဟာ extension လုပ်ဖို့ အတွက် ဖွင့်ထားပြီးတော့ class ထဲမှာ ရေးထားသည့် code တွေကိုတော့ ပြင်ဆင်ခွင့်မရှိပါဘူး။

Class ထဲမှာ function အသစ်ထပ်ဖြည့်မယ်ဆိုရင် ရေးသားပြီးသား class ကို မပြင်ပဲ class ကို extend လုပ်ပြီး ထည့်ပါ။

ဥပမာ ကြည့်ရအောင်။

```
class Login
{
    public function googleLogin()
    {
```

```
        echo "Login with google";
    }

    public function FacebookLogin()
    {
        echo "Login with Facebook";
    }
}



class LoginController
{
    public function login($type)
    {
        $login = new Login();
        if ($type == "google") {
           $login->googleLogin();
        }
        else if ($type == "facebook") {
            $login->facebookLogin();
        }
    }
}
```

ဒီ code ကို ကြည့်ကြည့်ပါ။ နောက်ထပ် Apple Login ထပ်ဖြည့်ချင်ရင် Login Class ကော LoginController ကော ပြင်ရမှာ ကို တွေ့ရမှာပါ။ ဒါဟာ ကောင်းမွန်သည့် code မဟုတ်ပါဘူး။ Open Close Principle အရ Open for Extension, Close for Modification ဖြစ်ရမှာပါ။ ဒါကြောင့် ကျွန်တော်တို့ ဟာ interface နဲ့ ခွဲထုတ် ပြီး extendable ဖြစ်အောင် လုပ်ပါမယ်။

```
interface LoginInterface
{
    public function login()
}



class GoogleLogin implements LoginInterface
{
    public function login() {
        echo "Google Login";
    }
}

class FacebookLogin implements LoginInterface
{
    public function login() {
        echo "Facebook Login";
    }
}
```

ဒါဆိုရင် LoginInterface ကို LoginController က သိဖို့ပဲ လိုတော့တယ်။ ဘာ Login type လည်း ဆိုတာကို သိဖို့ မလိုတော့ သလို နောက်ပိုင်း Apple Login ရှိလည်း AppleLogin class ကို LoginInterface သုံးပြီး ပြန်ပြင်နိုင်ပါတယ်။

```
class LoginController
{
     public function login(LoginInterface $type) {
          $type->login();
     }
}
```

ဒါဆိုရင် LoginController ကို လှမ်းခေါ်သည့် အခါမှာ

```
$conroller = new LoginController();
$controller->login(new GoogleLogin());
```

ဆိုပြီး ခေါ်နိုင်ပါတယ်။

Liskov substitution principle (LSP)

MIT က Professor Barbara Liskov က စပြီး အဆိုပြုခဲ့သည့် အတွက် LSP ဆိုပြီး ဖြစ်လာတာပါ။

> Objects in a program should be replaceable with instances of their subtypes without altering the correctness of that program

Class တစ်ခု က အခြား class တစ်ခု ရဲ့ အမွေ ဆက်ခံသည့် အခါမှာ အလုပ်လုပ်သည့် ပုံစံ ကို မ ပြောင်းလဲ စေဖို့လိုပါတယ်။

```
class Square extends Rectangle
{
    public function setWidth(int $width): void {
        $this->width = $width;
        $this->height = $width;
    }

    public function setHeight(int $height): void {
        $this->width = $height;
        $this->height = $height;
    }
}
```

Square class က rectangle class ကို extends လုပ်ထားပါတယ်။ setWidth/setHeight ထည့်လိုက်သည့် အခါမှာ width ကော height ကော အတူတူထည့်ထားပါတယ်။

ဒီ code က Liskov substitution principl ကို မလိုက်နာထားတာကို တွေ့ရပါတယ်။ Rectangle မှာ setHeight က height ကို ပဲပြောင်းလဲ သလို setWidth က width ကိုပဲ ပြောင်းလဲတာပါ။ ဒါပေမယ့် လက်ရှိ code မှာ ၂ ခု လုံးကို ပြောင်းလဲ ထားတာ တွေ့နိုင်ပါတယ်။

```
public function testCalculateArea()
{
    $shape = new Rectangle();
    $shape->setWidth(10);
    $shape->setHeight(2);

    $this->assertEquals($shape->calculateArea(), 20);

    $shape->setWidth(5);
    $this->assertEquals($shape->calculateArea(), 10);
}
```

ဒီလို test case မှာ ကြည့်လိုက်ရင် ရှင်းပါမယ်။ Width ပြောင်းသည့် အခါမှာ area ပြောင်းသွားပါတယ်။ ဒါပေမယ့် Square က မပြောင်းနိုင်ပါဘူး။ ဒါကြောင့် code က LSP ကို မလိုက်နာ ထားဘူးလို့ ပြောနိုင်ပါတယ်။

နောက်ထပ် ဥပမာ ကြည့်ရအောင်။

```
<?php

interface PaymentMethod {
    public function processPayment();
}

class CreditCardPayment implements PaymentMethod {
    public function processPayment() {
        // Implement the payment processing logic for credit cards
    }
}

class DebitCardPayment implements PaymentMethod {
    public function processPayment() {
        // Implement the payment processing logic for debit cards
    }
}

class CashPayment implements PaymentMethod {
    public function processPayment() {
        throw new Exception("Cash payments are not supported");
    }
}

class Checkout {
    public function processPayment(PaymentMethod $paymentMethod) {
        try {
            $paymentMethod->processPayment();
        } catch (Exception $e) {
            // Handle the exception for unsupported payment methods
            echo "Error: " . $e->getMessage();
        }
    }
```

```
}

// Example usage
$checkout = new Checkout();
$paymentMethod = new CashPayment(); // Try changing to CreditCardPayment or
DebitCardPayment

$checkout->processPayment($paymentMethod);
```

ဒီ code က LSP ကို လိုက်မနာထားဘူး။ ဘာဖြစ်လို့လည်းဆိုတော့ Cash Payment ကို အသုံးပြုသည့် အခါမှာ Cash Payment မှာ processPayment ဆိုသည့် function က အလုပ်မလုပ်လို့ပဲ။ LSP သဘောတရားက ဘယ် class ပဲ ပြောင်းပြောင်း အလုပ်လုပ်သည့် သဘောတရား တူညီ နေရမယ်။ ဒီ class မှာတော့ မလုပ်ဘူး။ ဒီ class ကို သုံးချင်ရင်တော့ ဒီလို ရေးဆိုပြီး လုပ်လို့မရပါဘူး။ Class အကုန်လုံးက အလုပ်လုပ်ပုံ တူညီနေရမယ်။

Interface segregation principle (ISP)

> Clients should not be forced to depend upon interfaces that they do not use

ရေးထားတာကတော့ ရှင်းပါတယ်။ Client မသုံးသည့် function ကို အတင်းသုံးခိုင်းထားတာ မျိုး မဖြစ်စေရပါဘူး။

```
interface Exportable
{
    public function getPDF();
    public function getCSV();
}
```

Exportable interface မှာ getPDF , getCSV ဆိုပြီး abstract function ရေးထားတာ တွေ့နိုင်ပါတယ်။

```
class Invoice implements Exportable
{
    public function getPDF() {
        // ...
    }
    public function getCSV() {
        // ...
    }
}

class CreditNote implements Exportable
{
    public function getPDF() {
        throw new \NotUsedFeatureException();
    }
    public function getCSV() {
        // ...
```

```
    }
}
```

Invoice အတွက် function ၂ ခု လုံးက အလုပ်လုပ်ပေမယ့် CreditNote အတွက် PDF ထုတ်ဖို့ မလိုဘူး။ ဒါပေမယ့် Exportable interface က getPDF ကို မဖြစ်မနေ function ထည့်ရေးခိုင်းထားသည့် သဘောမျိုး ဖြစ်နေပါတယ်။ ဒါဟာ LSP ကို မလိုက်နာတာပါ။

LSP ကို ဖြေရှင်းဖို့ Interface segration ကို အသုံးပြုနိုင်တယ်။ လိုသည့် function ပဲ interface မှာ ကြေငြာပါ ဆိုသည့် သဘောပဲ။ မကြေငြာထားသည့် function ကို class ထဲမှာ အတင်း မသုံးခိုင်းနဲ့။ ဒါကြောင့် လိုအပ်တာတွေကို interface ခွဲထုတ်ဖြစ်ဖို့ပါ။

```
interface ExportablePdf
{
    public function getPDF();
}

interface ExportableCSV
{
    public function getCSV();
}

class Invoice implements ExportablePdf, ExportableCSV
{
    public function getPDF() {
        //
    }
    public function getCSV() {
        //
    }
}

class CreditNote implements ExportableCSV
{
    public function getCSV() {
        //
    }
}
```

ဒါဆိုရင် မလိုအပ်သည့် function ပါနေဖို့ မလိုတော့ပါဘူး။

Dependency inversion principle (DIP)

> 1. High-level modules should not depend on low-level modules. Both should depend on abstractions.
> 2. Abstractions should not depend upon details. Details should depend upon abstractions.

High-level moduels တွေဟာ low-level modules တွေ အပေါ် depend မလုပ်သင့်ပါဘူး။ အဲဒီအစား abstractions ကို depend လုပ်ဖို့ လိုပါတယ်။

Abstractions တွေဟာ details ပေါ်မှာ depend မလုပ်သင့်ပါဘူး။ Details က သာ abstractions ပေါ်မှာ depend လုပ်ဖို့ လိုအပ်ပါတယ်။

```
class DatabaseLogger
{
    public function logError(string $message)
    {
        // ..
    }
}



class MailerService
{
    private DatabaseLogger $logger;

    public function __construct(DatabaseLogger $logger)
    {
        $this->logger = $logger;
    }

    public function sendEmail()
    {
        try {
            // ..
        } catch (SomeException $exception) {
            $this->logger->logError($exception->getMessage());
        }
    }
}
```

ဒီ class မှာ mail service ဟာ DatabaseLogger ကို တိုက်ရိုက် သုံးထားတယ်။ အကယ်၍ ကျွန်တော်တို့ Database မသုံးပဲ File သုံးမယ် ဒါမှမဟုတ် အခြား thrid party log service သုံးမယ် ဆိုရင် ဘယ်လိုလုပ်မလဲ။ ဒါကြောင့် MailService က DIP ကို ချိုးဖောက်နေပါတယ်။ Log အတွက် သီးသန့် interface တစ်ခု သုံးသင့်ပါတယ်။

```
interface LoggerInterface
{
    public function logError(string $message): void;
}

class DatabaseLogger implements LoggerInterface
{
   public function logError(string $message): void
   {
       // ..
   }
}

class MailerService
{
    private LoggerInterface $logger;

    public function sendEmail()
```

```
    {
        try {
            // ..
        } catch (SomeException $exception) {
            $this->logger->logError($exception->getMessage());
        }
    }
}
```

ဒါဆိုရင် LoggerInterface ဟာ database လည်း ဖြစ်နိုင်သလို File လည်း ဖြစ်နိုင်သလို third party တွေလည်း ဖြစ်လို့ရသွားပါပြီ။

အခု ဆိုရင် SOLID ကို နားလည်သဘောပေါက်ပြီး လက်ရှိ ရေးထားသည့် code တွေကို ပြန်ပြီး review လုပ်ကြည့်ပါ။ အသစ်ရေးမယ့် code တွေကို အတတ်နိုင်ဆုံး follow လုပ်ကြည့်ပါ။

အခန်း ၉ :: Project မစတင်ခင်

Project မစတင်ခင်မှာ လုပ်စရာ အလုပ်တွေ အရမ်းများပါတယ်။ လုပ်ရမယ့် အဆင့်တွေကတော့

၁။ Proposal ၂။ Requirement ၃။ Quotation ၄။ Contract ၅။ Wireframe ၆။ UI/UX ၇။ Human Resources

ဒီအဆင့်တွေက project manager , business development စသည် တို့ အတွက်လို့ ထင်ရပေမယ့် freelance လုပ်မည့် developer တွေအနေနဲ့လည်း အထက်ပါ အဆင့်တွေကို ကိုယ်တိုင် လုပ်ရပါမယ်။

Product development လုပ်သည့် အခါမှာ အထက်ပါ အဆင့်တွေ လိုအပ်မှ လိုအပ်ပါလိမ့်မယ်။ သို့ပေမယ့် product owner နှင့်တော့ အချို့အဆင့်တွေ သဘောတူညီမှု ရှိဖို့ လိုပါတယ်။

Proposal

Project မစတင်မှာ project proposal ရေးသားရပါတယ်။ Project proposal မှာ အဓိက အားဖြင့် လက်ရှိ project နဲ့ ပတ်သက်ပြီး ဘယ်လို အတွေ့အကြုံတွေ ရှိခဲ့တယ် နောက်ပြီး ဘယ်လို project တွေကို အောင်မြင်စွာ launch လုပ်ခဲ့ပြီး ဖြစ်သည့်အတွက် အခု project ကိုလည်း လုပ်နိုင်မည် ဖြစ်ကြောင်းကို ရေးသားဖော်ပြခြင်းဖြစ်ပါတယ်။

Proposal မှာ email နဲ့ တင် ရခြင်း ရှိသလို presentation ပြရသည့် အပိုင်းတွေလည်း ရှိပါတယ်။ Email နဲ့ ပို့ရသည့် အခါမှာ PDF format သည် အကောင်းဆုံး ဖြစ်ပါသည်။

ပါဝင်ရမည့် အချက်တွေကတော့

- About Company
- Summary
    - Objective/Problem
    - Portfolio
    - Solution
- Phases
- Estimate Cost
- Terms Of Payment
- Commencement
- Withdrawal of project
- Responsibilities

စသည့် အချက်အလက်ပါဝင်ပြီး ရေးသားထားဖို့ လိုပါတယ်။

Document အကုန်လုံးမှာ Company Letter Head နဲ့ ရေးသားဖို့ လိုပါတယ်။

အကယ်၍ presentation ပြရမည် ဆိုရင် အထက်အပါ အချက်အလက်တွေ ကို အတိုချုံးပြီး ပြသ ဖို့ လိုပါတယ်။ Guy Kawasaki ရဲ့ 10/20/30 rule က အသုံးဝင်ပါလိမ့်မယ်။

10 slide , 20 minutes , 13 point font size ပါ။ တတ်နိုင်သမျှ slide 10 ခု ထက် မပိုရန်။ မိနစ် ၂၀ အတွင်း အပြီးပြောပါ။ font size ကို 13 လောက် အနည်းဆုံးထားပေးပါ။

Requirement

Project က ဘယ်လို project လဲ ဆိုတာ အရေးကြီးတယ်။ In House Development လား ၊ Client project လား။ In House Development နှင့် Client project ကွာခြားချက်ကတော့ requirement changes ပါပဲ။

Client Project တွေက requirement က project စသည့် အချိန်ကတည်းက သတ်မှတ်ထားပါတယ်။ Changes တွေ အတွက် charges တွေ ရှိတယ်။ ဒါကြောင့် မလိုအပ်ရင် အသစ် ထပ်ဖြည့်တာမျိုး မလုပ်ပါဘူး။ ဒါကြောင့် Project size ပေါ်မှာ မူတည်ပြီး ၃ လ ကနေ ၁ နှစ် လောက် ကြာတတ်ပါတယ်။

In House Development တွေကတော့ company ပေါ်မှာ မူတည်တယ်။ stable ဖြစ်သည့် product တွေဆိုရင် timeline နဲ့ သွားတတ်ပေမယ့် development အဆင့်မှာ တော့ မျိုးစုံ ပြောင်းလဲ တတ်တယ်။ ပြီးသွားသည့် အခါမှာ Directors တွေ review feedback တွေ ပြန်လာရင် ပြန်ပြင် ရတာ တွေ ရှိမယ်။ CEO feedback တွေ ပြန်လာရင် ပြန်ပြင်ရတာ ရှိမယ်။ Launch တစ်ခု ဖြစ်ဖို့ အဆင့်ဆင့် approval တွေ ယူရတာမျိုး ရှိတတ်ပါတယ်။ Project တစ်ခု ထွက်ဖို့ ၆ လ ကနေ ၂ နှစ် လောက် အချိန်ကြာတတ်တယ်။

ဘယ်လို project ဖြစ်ဖြစ် developer အနေနဲ့ ကိုယ်လုပ်ရမယ့် tasks list တွေကို timeline နဲ့ သေချာ မှတ်ထားဖို့ လိုတယ်။ Changes တွေ ရှိလာခဲ့ရင် အချိန် ပို ယူ ဖို့ အတွက် နောက်ကျ နိုင်ကြောင်း ပြသ ဖို့ အတွက် သက်သေ ပြဖို့ လိုတယ်။

Quotation

Quotation က requirement တွေ ရပြီးပြီ။ proposal လည်း approve ဖြစ်ပြီ ဆိုရင် quotation ကို ပို့ဖို့ လိုပါတယ်။ Invoice လိုပါပဲ။ ကျသင့်ငွေ အသေးစိတ်ကို invoice အတိုင်း ဖော်ပြပေးဖို့ပါ။ quotation ကို approve ရမှသာ project အတွက် စတင်ရပါမယ်။ Quotation approval မရခင်မှာ project ရပြီလို့ သတ်မှတ်ထားလို့ မရပါဘူး။

Business Requirement Specification (BRS)

Project မစခင်မှာ အကြမ်းအားဖြင့် ဆွေးနွေးပြီးသွားရင် BRS ကို client ဖြစ်ဖြစ် product owner ဆီက ဖြစ်ဖြစ် တောင်းဖို့ လိုတယ်။ Project Manager ဟာ BRS ထဲမှာ ပါသည့် အချက်အလက်တွေ နဲ့ ဆွေးနွေးထားသည့် requirement အချက်အလက်တွေ ကွာနေလား ဆိုတာကို စစ်ဖို့ လိုတယ်။ BRS က ပုံမှန် အားဖြင့် product owner ဆီ က နေ ရ ပါတယ်။

BRS ဟာ contract မတိုင်ခင်မှာ နှစ်ဖက်ညှိထားသည့် requirement တွေ Business ဘက်က လိုအပ်ချက်တွေ ပါဝင်ပါတယ်။

BRS တစ်ခုမှာ

- Project Name
- Business Project Owner
- Author
- Contributors

- Project Manager

စသည် တို့ ကနေ စတင်ပြီး ပါဝင်ပါတယ်။

ပြီးလျှင် Revision History ကို မဖြစ်မနေ ထည့်သွင်းဖို့လိုပါတယ်။ ဒါမှသာ Vendor ဘက်က သဘောတူထားသည့် Revision number က ဘယ် number ၊ Vendor မသိသေးသည့် Revision number က ဘယ် number ဆိုပြီး ခွဲသိနိုင်ပါလိမ့်မယ်။ ကိုယ့်ဘက် က complain တက်လာရင် လည်း BRS revision number xxx အတိုင်း develop လုပ်ထားပြီး နောက်ထပ် revision number yyy ကို လက်ခံမရရှိကြောင်း ငြင်းလို့ ရမှာပါ။

BRS မှာ ထပ်မံ ပါဝင်သည့် အရာတွေကတော့

- Business Objective
- Current Problems and Limitation
- General Business Requirement
- Business Requirement

စသည့် အချက်အလက်တွေ ပါဝင်ပါတယ်။

BRS က client ရဲ့ company ပေါ်မှာ မူတည်ပြီး ကွဲပြားတတ်ပါတယ်။

Contract

Developer နဲ့ မဆိုင်ပေမယ့် Project တစ်ခုမှာ contract ဟာ အရမ်းအရေးပါတယ်။ တချို့ project တွေက contract မတိုင်ခင်မှာ စပြီး လုပ်နေရင်းနဲ့ contract မချုပ် မိတာ တွေ ရှိပါတယ်။ Project Manager , Business development managers တို့ရဲ့ ဆုံးဖြတ် ချက်တွေက အရမ်းကို အရေးပါပါတယ်။ contract ပြီးမှ project စမယ် ဆိုရင်လည်း အရမ်းကို နောက်ကျ သွားတာတွေ ရှိတတ်တယ်။ Project owner နဲ့ သဘောတူထားတာကတော့ ၃ လ ပဲ ကြာမယ်။ သို့ပေမယ့် contract အ ဆင့်မှာတင် ၃ လ လောက် ကြာသွားတာ ရှိတတ်တယ်။ နားလည်မှု ၊ ယုံကြည်မှုတွေ နဲ့ လုပ်ရပေ မယ့် အကောင်းဆုံးကတော့ contract ပြီးမှသာ လုပ်ဆောင်ခြင်းဟာ အကောင်းဆုံးပါပဲ။ Contract တစ်ခု ကို ပြင်ဆင်ရတာဟာလည်း ၁ ပတ်လောက် ကြာတတ်ပါတယ်။ Contract မပြီးသေးခင် လုပ်ရသည့် အလုပ်ဟာ ပြဿနာဖြစ်ခဲ့ရင် အချက်နဲ့အလက် နဲ့ ပြောဖို့ အတော့်ကို ခက်ပါတယ်။ ဥပမာ Contract ထဲမှာ မပါသည့် Features ကို စကားနဲ့ ပြောထားတယ်။ ကိုယ့်ဘက်ကလည်း develop လုပ်ထားလိုက်တယ်။ တဖက်က ကျွန်တော် ပြောထားပြီးသားလေ။ ပြောထားသည့် စျေးထဲမှာ အဲဒီ feature အပါလို့ ထင်တာ။ စသည် ဖြင့် ညှိနှိုင်းရတာ အလုပ်ရှုပ်ကုန်ပါတယ်။ ဒါ ကြောင့် contract က အရေးကြီးပါတယ်။ contract မထိုးနိုင်ရင်တောင် BRS ကို ရယူထားမှ ဖြစ်ပါ လိမ့်မယ်။

ဒီအချက်အလက်အကြောင်းတွေက စာနဲ့ ရေးသားရတာ လွယ်ပေမယ့် တကယ့် လက်တွေ့မှာ လူတွေ နဲ့ အလုပ်လုပ်ရတာ ခက်ပါတယ်။ လူတစ်ယောက်နဲ့ တစ်ယောက် မတူညီကြသည့်အတွက် ညှိနှိုင်းသည့် အခါမှာ လူပေါ်မှာ မူတည်ပြီး ညှိနှိုင်းပြီး ဆောင်ရွက်တတ်ဖို့ အရေးကြီးပါတယ်။

Wireframe

Team တစ်ခု ကို ရှင်းပြဖို့ client ဘက်က product owner ကို ရှင်းပြဖို့ အတွက် ဖြစ်စေ နောင်တချိန် project document အတွက် project မစချိန်မှာ အကြမ်းအားဖြင့် project အတွင်းပါဝင်ပတ်သက်သူတွေ နားလည် ရှင်းလင်းဖို့ အတွက် wireframe ကို ရေးဆွဲပါတယ်။

wireframe ရေးဆွဲသည့် software တွေ အများကြီး ရှိပါတယ်။ လက်ရှိ ကျွန်တော်ကတော့ https://whimsical.com ကို အသုံးပြုပါတယ်။ အခြား နှစ်သက်ရာ wireframe tool ကို အသုံးပြုနိုင်သလို အလွယ်ကူဆုံးကတော့ လက်နဲ့ပဲ ရေးဆွဲတာတွေလည်း ရှိပါတယ်။

wireframe ကို အခြား UI/UX tool တွေဖြစ်သည့်

- Sketch
- Adobe XD
- Penpot

စသည်တို့မှာ ရေးဆွဲထားရင်တော့ final UI/UX ပြောင်းသည့် အခါမှာ ပိုမိုမြန်ဆန် လွယ်ကူပါတယ်။

Wireframe ဟာ project ရဲ့ size ပေါ်မှာ မူတည်ပြီး ကြာမြင့်နိုင်ပါတယ်။

Wireframe မှာ product တစ်ခုလုံးကို ခြုံပြီး pageတိုင်းပါဖို့လိုပါတယ်။ Page တိုင်းမပါနိုင်ရင်တောင် product owner နဲ့ သဘောတူညီထားသည့် page တွေပါဝင်ထားဖို့ လိုပါတယ်။

Wireframe သေချာသွားမှသာ UI/UX ကို စသင့်ပါတယ်။ ပုံမှန် အားဖြင့် page တစ်ခု ခြင်းစီ ကို confirm လုပ်ပြီး ပြီးသွားသည့် page တွေကို UI/UX အပိုင်း စသင့်ပါတယ်။ Wireframe ကို အချိန်ပေးပြီး လုပ်ဖို့ လိုပါတယ်။ Project timeline နည်းတယ် Project က သေးတယ် ဆိုပြီး wirframe မပါပဲ development လုပ်သည့် အခါမှာ နောက်ပိုင်း ပြဿနာတက်နိုင်ပါတယ်။

Wireframe မှာ project owner ကိုယ်တိုင် စမ်းလို့ရသည့် prototype အဆင့်ထိ ပါ လုပ်ပေးနိုင်ရင် ပိုကောင်းပါမယ်။ ဒါမှသာ ဘာမှ မစခင်မှာ project owner လိုချင်တာ ဘာလဲ ဆိုတာကို သိသာရှင်းလင်းစေပါလိမ့်မယ်။ Project တစ်ခု စတင်သည့် အခါမှာ စမ်းတဝါးဝါး နဲ့ စိတ်ထင်သည့် ပုံစံကို စတတ်ကြပေမယ့် wireframe အဆင့်မှာ ဘာလိုချင်လဲ ဘာဖြစ်ချင်လဲ ဆိုတာကို ပိုပြီးမြင်သာလာပါတယ်။

Wireframe ပြီးသွားရင် project owner နဲ့ သဘောတူညီမှု လက်မှတ် ဖြစ်ဖြစ် ထိုးထားခြင်း ဖြစ်စေ email ကနေ တဆင့် သဘောတူညီကြောင်း ဖြစ်စေ ရယူထားဖို့ လိုအပ်ပါတယ်။ နောက်ပိုင်း changes တွေ ရှိလာခဲ့ရင် new fatures , bug ငြင်းနေသည့် အခါမှာ wireframe ပေါ်မှာ သဘောတူ ညီ ထားချက်ကို ပြန်လည် ကြည့်ရှုဖို့ လိုပါတယ်။

UI/UX

Wireframe မပြီးခင်မှာ UI/UX ကို စတင်ဖို့ အကြံ မပေးချင်ပါဘူး။ Wiframe က ခဏ ခဏ ပြောင်းလဲ တတ်ပါတယ်။ လိုအပ်ချက်တွေက အမြဲ ပြောင်းလဲတတ်ပါတယ်။ Wireframe မှာ confirm ထားပေမယ့် UI/UX ရောက်သည့် အချိန်မှ ပြန်ပြီး ပြောင်းတာတွေလည်း ရှိနိုင်ပါတယ်။

UI/UX ကို အများအားဖြင့်

- Sketch
- Adobe XD
- Figma
- Penpot

စသည့် tool များ ဖြင့် ရေးဆွဲကြပါတယ်။ Designer တွေက ပုံမှန်အားဖြင့် ပုံတွေက export မ ထုတ်ပဲ developer တွေ ကိုယ်တိုင် လိုအပ်သည့် icons တွေ images တွေကို export ထုတ်ရတာ တွေရှိပါတယ်။ ဒါကြောင့် Developer တွေ အနေနဲ့ ကိုယ်တိုင် export လုပ်နိုင်အောင် tool တွေ ကို သုံးတတ်ဖို့ လိုပါတယ်။ အကုန်လုံးကတော့ အခြေခံတွေ တူညီ ကြသည့် အတွက် လေ့လာရတာ ခက်ခဲမှုတော့ မရှိပါဘူး။

Human Resources

Project Contract စကတည်းက project မှာ resources ဘယ်လောက်ပါမလဲ ဆိုတာ ဆုံးဖြတ်ထား ပြီးသားပါ။ အများအားဖြင့် Resources တွေက အခြား project မပြီးသေးသည့် အခါ ဒါမှမဟုတ် bugs fix လုပ်ဖို့ လိုသည့် အခါတွေ မှာ သတ်မှတ်ထားသည့် ရက်မှာ သတ်မှတ်ထားသည့် tasks တွေ လုပ်ဖို့ နောက်ကျ သွားတာ တွေ ရှိပါတယ်။ ဒါကြောင့် အဲဒီလိုမျိုး issue တွေ အတွက် timeline မှာ အချိန် ပိုယူထားဖို့ လိုအပ်ပါတယ်။ Project တစ်ခု ဟာ ပြီးဆုံးသွားပေမယ့် အမြဲ တန်း bugs တွေ features အသေးလေးတွေ ပြန်လာတတ်ပါတယ်။ အဲဒီ အခါမှာတော့ project ကို လုပ်ထားသည့် developer ကိုယ်တိုင် လုပ်ဆောင်ရတာ များပါတယ်။ ဒါကြောင့် project timeline တော်တော်များများမှာ developer တစ်ယောက် ဟာ အချိန်ပြည့် ၁၀၀% လုပ်ဆောင်နိုင်မယ်လို့ တွက်ထားလို့ မရပါဘူး။ ပြဿနာ နောက်တစ်ခုက လက်ရှိ developer မအား လို့ အခြား developer တစ်ယောက်က လွှဲယူရသည့် အခါမှာ တွက်ထားသည့် timeline ထက် ပိုမို ကြာမြင့်နိုင် ပါတယ်။ Project Manager အနေနဲ့ timeline ချ သည့် အခါမှာ Developer တွေရဲ့ timeline ကို လက်ရှိ လုပ်နေဆဲ ကိုင်ထားသည့် project အခြေအနေ ပေါ် ကြည့်ပြီး ရေးဆွဲဖို့ လိုပါတယ်။

အခန်း ၁၀ :: Estimate Task

Developer တစ်ယောက် ဟာ task တစ်ခုကို အချိန် ဘယ်လောက် ယူမလဲ ဆုံးဖြတ်ချက်က အရေးပါပါတယ်။ ပြဿနာက မလုပ်ဖူးသည့် task ဆိုရင် အချိန် ဘယ်လောက်ယူ ရမလဲမသိဘူး။ Project Manager က အဲဒီ အချိန်မှာ အရေးပါတယ်။ တွေ့ကြုံခဲ့သည့် အတွေ့အကြုံရ ဒီ task ကတော့ အချိန်ဘယ်လောက်က အများဆုံးလည်း ဆိုတာ သိဖို့ လိုတယ်။ အဲလို သိဖို့ အတွက်လည်း code တွ အများကြီး ရေးဖူးတာ ဖြစ်ဖြစ် programming မှာ senior တွေ နဲ့ တွဲပြီး အလုပ်တွေ လုပ်ခဲ့ဖူးတာ ဖြစ်ဖြစ် အတွေ့အကြုံတွေ လိုအပ်ပါတယ်။

Developer နဲ့ project manager စကားများ ရသည့် အချက်ထဲမှာ estimate task လည်း ပါပါတယ်။ Developer က ၅ ရက် လောက် လိုမယ် Project Manager က ၃ ရက် နဲ့ ပြီးအောင် လုပ်ရမယ်ဆိုပြီး သတ်မှတ်ထားသည့် အခါမှာ ကတောက်ကဆ ဖြစ်ရပါတယ်။ Developer အနေနဲ့ Project Manager က ဒီ အလုပ်တွေ မလုပ်ခဲ့ဖူးလို့ နားမလည်ဘူး လို့ ထင်လိမ့်မယ်။ Project Manager က လည်း Developer skill အားနည်းလို့ မပြီး နိုင်တာ လို့ ထင်ရတာ အခါမှာ အဖုအထစ်တွေ ဖြစ်တတ်ပါတယ်။

Task တစ်ခုကို အချိန် ဘယ်လောက် ကြာမလဲ လို့ ဆုံးဖြတ်ချက်ချ ခက်နေပြီ ဆိုရင် Project Manager အနေနဲ့ အခြား level တူ developer တစ်ယောက်ကို ခေါ်ပြီး ဒီ task က ဘယ်လောက် ကြာမလဲ ဆိုပြီး ပွင့်ပွင့်လင်းလင်း မေးဖို့ လိုတယ်။ အကယ်၍ project manager က အဲလို မလုပ်ခဲ့ရင်လည်း Developer အနေနဲ့ အခြား တစ်ယောက်ကို ခေါ်ပြီး ဆုံးဖြတ်ဖို့ လိုတယ်။

အဲဒီ အခါမှာ သူက ၄ ရက် ကိုယ်က ၈ ရက်ဖြစ်နေတာ ဖြစ်ဖြစ် ရက်တွေ က အရမ်းကွာနေတာ ဖြစ်နေခဲ့ရင် ဘာကြောင့်ကြာတယ် ဘာကြောင့်မြန်တယ် ဆွေးနွေးပြီး ပြန်ညှိဖို့ လိုအပ်ပါတယ်။ ပြီးသည့် အခါ ၂ ဘက် သဘောတူညီသည့် timeline ကို ရနိုင်သည့် အခါမှာ project manager အတွက်ကော developer အတွက်ပါ အဆင်ပြေပါတယ်။

Unknown

Developer တစ်ယောက် အနေနဲ့ အခက်အခဲ ဆုံး အလုပ်က ဘာလဲဆိုတော့ တစ်ခါမှ မလုပ်ဘူး မ မြင်ဘူး သည့် ကိစ္စကို timeline ထုတ်ပေးရတာပဲ။ ကိုယ်က ဒီ project နဲ့ experience မရှိဘူး။ timeline ဘယ်လောက်ကြာမလဲဆိုတာကို မေးသည့် အခါမှာ ချက်ခြင်း မဖြေသေးဘူး research သေသေချာချာ လုပ်ရပါတယ်။

ဥပမာ Wix ကို mobile app လုပ်မယ်။ အချိန် ဘယ်လောက် လိုမလဲ။ Wix နဲ့လည်း တစ်ခါမှ မ လုပ်ဘူး။ ကြားလည်းမကြားဘူး။ ဘာပြန်ဖြေရမလဲ မသိတွေကတော့ Developer တိုင်း ကြုံရမှာ ပါ။ Senior တော်တော်များများဟာ လုပ်ခဲ့ဘူးသည့် အလုပ်သက်တမ်းကြောင့် တော်တော်များ များ လုပ်ခဲ့ဖူးပါတယ်။ ဒါကြောင့် project တော်တော်များများ timeline ကို junior တွေ ထက် ပို ပြီး မှန်မှန်ကန်ကန် ဆုံးဖြတ်ချက် ချနိုင်တာပါ။ အဓိက working experience က စကားပြောတာ ပါ။

၁ ပတ်စာ Estimate Tasks

Company တော်တော်များများက Agile, Scrum,Kanban စသည် methodology တွေကို အသုံးပြု ကြပါတယ်။ အလုပ် စဝင်ကာစမှာ Agile တို့ SCRUM တို့ကို နားလည်မှာ မဟုတ်ပါဘူး။ Sprint မှာ task တွေကို ဖြည့်ခိုင်းသည့်အခါမှာ ဘာတွေ ဖြည့်ရမလဲ ဆိုတာကို မသိ ဖြစ်တတ်တယ်။ ပုံမှန် အားဖြင့် Monday မှာ မနက်ပိုင်း meeting လုပ်ပြီး ၁ ပတ် စာကို ညှိနှိုင်းပါတယ်။

Project တိုင်းမှာ product owner သို့မဟုတ် project manager ရှိဖို့ လိုပါတယ်။ Tasks တွေကို project manager/product owner က သတ်မှတ်ပြီး assign လုပ်ပေးပါတယ်။ အကယ်၍ team က သေးမယ် သို့မဟုတ် project က သေးခဲ့ရင် developer 2 ယောက် လောက် နဲ့ လုပ်ဆောင်ရတာ တွေ ရှိပါတယ်။

Tasks ကို developer ကိုယ်တိုင် assign လုပ်ရသည် ဖြစ်စေ project manager/product ownwer က assign လုပ်သည် ဖြစ်စေ ကိုယ် ၁ ပတ် လုပ်မယ့် tasks က ကိုယ် ကြာမယ် ထင်ထားသည့် အချိန်နဲ့ ကိုက် မကိုက် ကို ဆုံးဖြတ်ဖို့ လိုတယ်။ အလုပ်ဝင်ကာစမို့ မှားနိုင်ပေမယ့် နောက်ပိုင်းလည်း မှားနေဆဲပါပဲ။ ဒါကြောင့် အမြဲ အချိန် ပိုယူဖို့ test လုပ်ဖို့ အချိန်ပါ ထည့်ပြီး တွက်ဖို့ လိုပါတယ်။

အခန်း ၁၁ :: Teamwork

Teamwork က မြန်မာ တွေ အတွက် အလုပ်ထဲရောက်မှ လေ့လာသင်ယူ ကျရတာ များပါတယ်။ ငယ်စဉ်ကတည်းက အမြဲ တစ်ကိုယ်တည်း ထူးချွန်အောင် ကြိုးစားရတာတွေ များပါတယ်။ မြန်မာ ယဉ်ကျေးမှုက အတန်းထဲမှာ သူများတွေထက် အဆင့်သာ အောင် အခြား ကိစ္စတွေမှာ လည်း ပြိုင်ဆိုင်သည့် အခါမှာ ကိုယ့် သားသမီး ပိုသာ အောင် မိဘတွေက တွန်းအားပေးကြပါ။ အားကစား လုပ်သည့် activity တွေကလည်း နည်းပါတယ်။ နောက်ပိုင်းမှာ mobile games တွေ ကြောင့် team work ကို အရင်ကထက် ပိုနားလည်လာတာ လူငယ်တွေ အတွက် အကျိုးရှိပါ တယ်။

အလုပ်ထဲမှာ ကိုယ်တစ်ယောက်တည်း ထူးပြီးချွန်နေလို့ မရပါဘူး။ team က မည်သည့် အခါမှာ ကတောက်ကဆ တွေ ဖြစ်တတ်တယ်။ နောက်ပိုင်း အချို့ လူငယ်တွေက ငါပိုသိတယ် ပိုတတ် တယ် လို့ ထင်ပြီး အလုပ်ထဲမှာ အဆင်မပြေတာတွေ ရှိတတ်တယ်။

အလုပ်ထဲမှာ team နဲ့ လုပ်ရပါတယ်။ team တစ်ခုမှာ လူ ၂ ယောက်ကနေ လူ ၁၀ ယောက်လောက် ထိ ပါတတ်ပါတယ်။ ပုံမှန် အားဖြင့် team က တစ်ခုကို လူ ၅ ယောက် နဲ့ ဖွဲ့စည်းတတ်ပါတယ်။ အဓိကတော့ ကိုယ့်ရဲ့ task က ဘာတွေ လုပ်ရမလဲ။ ကိုယ့်ရဲ့ tasks ပြီးမှ ဘယ်သူတွေက အလုပ် လုပ်လို့ ရမှာလဲ။ ကိုယ်ကကော ဘယ်သူ့ tasks တွေပြီးမှ အလုပ်စလို့ ရမလဲဆိုတာတွေ သိထားဖို့ လိုတယ်။

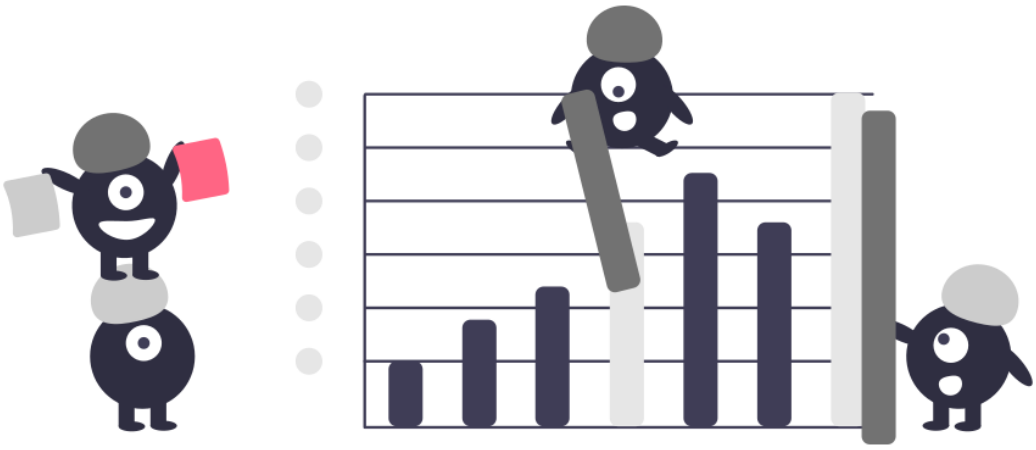
ဥပမာ mobile app ဆိုရင် backend API နဲ့ UI design ၂ ခု ပြီးမှ mobile app ကို စလို့ ရပါလိမ့်မယ်။ ကိုယ့် ရဲ့ App ပြီးမှ QA က စပြီးတော့ စစ်လို့ ရပါမယ်။ team work နဲ့ အလုပ်လုပ်သည့် အခါ ကိုယ်က timeline နဲ့ မကိုက်ခဲ့ရင် အခြားသူတွေရဲ့ timeline တွေကိုပါ effect ဖြစ်နိုင်ပါတယ်။ အဲလိုပဲ အခြားသူတွေက မှန်းထားသည့် အချိန် အတိုင်း မပြီးရင် ကိုယ် plan လုပ်ထားတာတွေ လွဲ ကုန်တယ်။

Skills

အဖွဲ့ထဲမှာ တစ်ယောက် နဲ့ တစ်ယောက် skills မတူညီ ကြပါဘူး။ အများအားဖြင့် ကိုယ်က အရမ်းသာနေသည် ဖြစ်နိုင်သလို ကိုယ်က အဖွဲ့သားရဲ့ skill ထက် အများကြီး နောက်ကျနေတာ ဖြစ်နိုင်ပါတယ်။ Skill ကတော့ အမြဲလေ့လာနေသည့် သူက တော့ တိုးတက်နေမှာပါ။ မတူညီ သည့် skill တွေကြောင့် team မှာ ပြဿနာ တွေ တက်နိုင်ပါတယ်။ တစ်ခါတစ်ရံမှာ junior က ပိုပြီး ရေးသားနိုင်သည့် အခါတွေ မှာ မလေးစားတတ်တာတွေ ရှိတတ်ပါတယ်။ Senior တစ်ယောက် ဟာ programming skills က နောက်ထပ် ဝင်သည့် junior တွေ လောက် strong ဖြစ်ချင်မှ ဖြစ် တော့မယ်။ Programming ဟာ အမြဲတန်း နှစ်စဉ် ပြောင်းလဲနေတာပါ။ ဒါကြောင့် အမြဲ update လုပ်နေဖို့ လိုပါတယ်။ လုပ်ငန်းခွင်ထဲ အကြာကြီးရောက်နေပြီးသား လူတွေက skill ကို upgrade လုပ်ဖို့ အချိန် သီးသန့် မပေးနိုင်ကြပါဘူး။ အလုပ်မှာ ရှိသည့် ပြဿနာတွေ မိသားစု ပြဿနာ တွေ အပြင် အခြား ပြဿနာပေါင်းများစွာ နဲ့ ဘဝ ကို ရုန်းကန်ရသည့် အရွယ်တွေ ဖြစ်ကုန်ကြပါ ပြီ။ ဒါကြောင့် အချို့ senior level တွေက ကိုယ့်ထက် programming skills ပိုကောင်းသည့် junior developers တွေကို ခန့်ကြပါတယ်။

Senior သမားတွေက customer တွေ အများကြီး နဲ့ တွေ့ဖူးတယ်။ Project Manager အများကြီးနဲ့ တွေ့ဖူးသည့် အတွက်ကြောင့် ပြဿနာတွေ လည်း များစုံ ကြုံခဲ့ပြီးသား သူတွေပါ။ Junior တွေမှာ အဲဒီ အတွေ့အကြုံမရှိပါဘူး။ ဒါကြောင့် programming skills တစ်ခု တည်းနဲ့ အလုပ်တစ်ခုဟာ မ ပြီးမြောက်ပါဘူး။ Teamwork ဟာ အရေးပါတယ်။ Senior တွေရဲ့ တာဝန်ယူမှုဟာလည်း အရေး ပါတယ်။ တာဝန်မယူနိုင်သည့် senior တွေက junior တွေရဲ့ လေးစားမှုခံရမှာ မဟုတ်ပါဘူး။

အခန်း ၁၂ :: Project စတင်ခြင်း



Project တစ်ခု စပြီ ဆိုရင် ဘာတွေ လုပ်ရမလဲ။ ဘာတွေ ပြင်ရမလဲ ဆိုတာကို စလုပ်ခါစ Project Manager တွေ အတွက် ခေါင်းစားပါတယ်။ Project တစ်ခု စပြီဆိုရင် အရင်ဆုံး ကျွန်တော်တို့ လိုအပ်တာက overall system design ပါ။

1. User Stories
2. Use Case Diagram (System Flow)
3. ERD
4. Backlog
5. Sprint

စတာတွေကို ပြင်ဖို့ လိုပါတယ်။ Scrum ကို အခြေခံထားပေမယ့် team တော်တော်များများဟာ Scurm အတိုင်း အပြည့်အဝ မ run နိုင်ပါဘူး။ အများအားဖြင့် ပြည်တွင်းမှာ Kaban board နဲ့ သွားကြတာ များပါတယ်။ ဒီစာအုပ်ဟာ Scrum စာအုပ် မဟုတ်သည့် အတွက်ကြောင့် အသေးစိတ်တော့ မရေးထားပါဘူး။ လုပ်ငန်းခွင်မှာ ဘယ်လို အလုပ်လုပ်နေကြတယ် ဆိုတာ သိလောက်အောင်ပဲ ဖြည့်စွက်ရေးထားပါတယ်။

User Stories

Project စတော့မယ် ဆိုရင် အသုံးပြုမယ့်သူတွေ ဘယ်လို အသုံးပြုမလဲ။ ဘယ်လို အသုံးပြုရင် လွယ်မလဲ ဆိုတာကို စပြီး လေ့လာကြပါတယ်။ ဥပမာ Musica App ကို လုပ်ပြီ ဆိုပါဆို့။ ဘယ် အချက်တွေကို ကြည့်ပြီး user က သီချင်း နားထောင်မလဲ ? User တစ်ယောက် သီချင်း တစ်ပုဒ် ကို နားထောင်ဖို့ ဘယ်လို လမ်းကြောင်းတွေ ရှိပြီး ဘယ်လို လုပ်ရင် ရနိုင်မလဲ။

Payment အတွက်ကကော User stories ဘယ်လို ရှိမလဲ။ ဘယ်အချိန် ရောက်ရင် User က ငွေပေး မလဲ။ ငွေပေးဖို့ အဆင့်တွေကတော့ ဘယ်နှစ်ဆင့်လောက် ရှိလဲ။ ငွေပေးပြီးရင် ဘယ်လိုဖြစ်သွား မလဲ။

အချက်အလက်များစွာပေါ်မှာ မူတည်ပြီး User stories ကို တည်ဆောက်ပါတယ်။ Proposal မှာ လုပ်ထားသည့် အချက်အလက်တွေ နဲ့ ကွဲပြားနိုင်တယ်။ proposal wireframe က အပိုင်းနဲ့ မတူ တာတွေကို Product owner နဲ့ ညှိဖို့ လိုတယ်။ ဘယ်အချက်တွေ က ပြောင်းသွားတယ်။ ဘယ် အချက်တွေ သဘောတူတယ် မတူဘူး ဆိုပြီး ပြန်ညှိရပါတယ်။

Use Case Diagram (System Flow)

Use Case Diagram ဟာ team ကို လက်ရှိ လုပ်မယ့် project အတွက် ရှင်းပြလို့ရသည့် အလွယ်ဆုံး diagram ပါပဲ။

Use Case Diagram ကို Computer Science နဲ့ ဘွဲ့ ရထားသည့် ကျောင်းသား တော်တော်များများ ဆွဲ တတ်ပါတယ်။ Unified Modeling Language (UML) သင်သည့် အထဲမှာ Use Case ကိုလည်း ထည့်သင်ပါတယ်။ Computer Science နဲ့ ဘွဲ့ ရထားခဲ့ပြီးရင် Class Diagram, Use Case Diagram, Flow Chart တွေ သိထားပြီးလို့ မှတ်ယူလို့ ရပါတယ်။ အကယ်၍ စာဖတ်သူဟာ Computer တက္ကသိုလ် တစ်ခုခုက ဘွဲ့ရထားခဲ့ပြီး အထက်ပါ diagram တွေ မဆွဲတတ်သေးရင်တော့ ပြန် လေ့လာဖို့ လိုအပ်ပါတယ်။

နောက်အရေးကြီးသည့် အချက်ဟာ document တွေကို သေချာ version ခွဲပြီး သိမ်းထားဖို့ လိုပါ တယ်။ မြန်မာနိုင်ငံမှာ ရှိသည့် project manager တွေက လိုအပ်မှသာ email တွေ ထဲမှာ ပြန်ရှာ တတ်ပါတယ်။ version အလိုက်ခွဲပြီး file တွေကို သိမ်းထားသည့် အကျင့်ကို လုပ်ထားဖို့ လိုပါ တယ်။ ဒါမှသာ developer က flow များသွားရင် သူ့ version နဲ့ ကိုယ့် version မတူတာလား။ ကိုယ်ပြောဖို့ ကျန်ခဲ့တာလည်း စတာတွေကို စီစစ်နိုင်မှာပါ။

Entity Relationship Diagram (ERD)

ERD ကတော့ Project ကို lead လုပ်မယ့် lead developer တွေ ဆွဲကြတာ များပါတယ်။ database ကို ဘယ်လို design ထားသင့်တယ်။ ဘယ် table တွေ လိုအပ်တယ်။ အချို့ data တွေက table ခွဲထုတ်လို့ရပေမယ့် flat ထားသင့်အခါ ထားတာတွေ လည်း ရှိပါတယ်။ ERD diagram ကို ဆွဲ လိုက်ခြင်းအားဖြင့် လက်ရှိ project မှာ ပါဝင်ရမယ့် model တွေ ကို ရှင်းလင်းစွာ မြင်စေပါတယ်။

Team အနေနဲ့လည်း ဘယ် class တွေကို ဖန်တီးသင့်တယ် API design ဘယ်လို ချရမလဲဆိုတာကို ERD ပေါ်မှာ မူတည်ပြီး ဆုံးဖြတ်နိုင်ပါလိမ့်မယ်။ နောက်တချက် သတိထားသင့်တာကတော့ ERD diagram က development လုပ်နေရင်း ပြောင်းလဲ သွားနိုင်ပါတယ်။ တကယ့် plan အတိုင်း မဟုတ်ပဲ sprint တစ်ခု မှာ changes ရှိခဲ့ရင် ပြောင်းလဲ မှုတွေ ဖြစ်နိုင်ပါတယ်။ ဖြစ်နိုင်ရင်တော့ diagram တွေကို version ခွဲပြီး သိမ်းထားရင် အကောင်းဆုံးပါပဲ။

Backlog

ဘာမှ မစခင်မှာ test တွေ အကုန်လုံးကို backlog ထဲကို ထည့်ထားပါ။ ပြီးမှ သက်ဆိုင်ရာ developer ကို assign ချဖို့ လိုပါတယ်။ Backlog မှာ တစ်ခါတည်း estimate task duration ပါ ထည့်ပြီး သတ်မှတ်ထားဖို့ လိုပါတယ်။

Sprint

Backlog တွေ ရရင် actual timeline ကို ရပါပြီ။ လူ ဘယ်နှစ်ယောက် တကယ် သုံးရမယ်။ အချိန် ဘယ်လောက် ကြာမလဲ ဆိုတာ ထွက်လာပါပြီ။ အဲဒီ အခါမှာ estimate လုပ်ပြီး project proposal timeline နဲ့ ကိုက်မကိုက် ကြည့်ရပါတယ်။ မကိုက်ခဲ့ရင် ပြန်ညှိတာတွေ လုပ်ဖို့ လိုတယ်။

နောက်တချက်က အခု project မှာ ပါသည့် developer တွေဟာ အကြောင်းကြောင်းကြောင့် sprint တစ်ခုမှ မပါသည့် အခါ မှာ timeline တွေနောက်ကျနိုင်တယ်။ အဲဒီ အတွက်ပါ ဘယ် လောက် နောက်ကျလို့ ရသည် အထိ ထည့်စဉ်းစားဖို့ လိုတယ်။

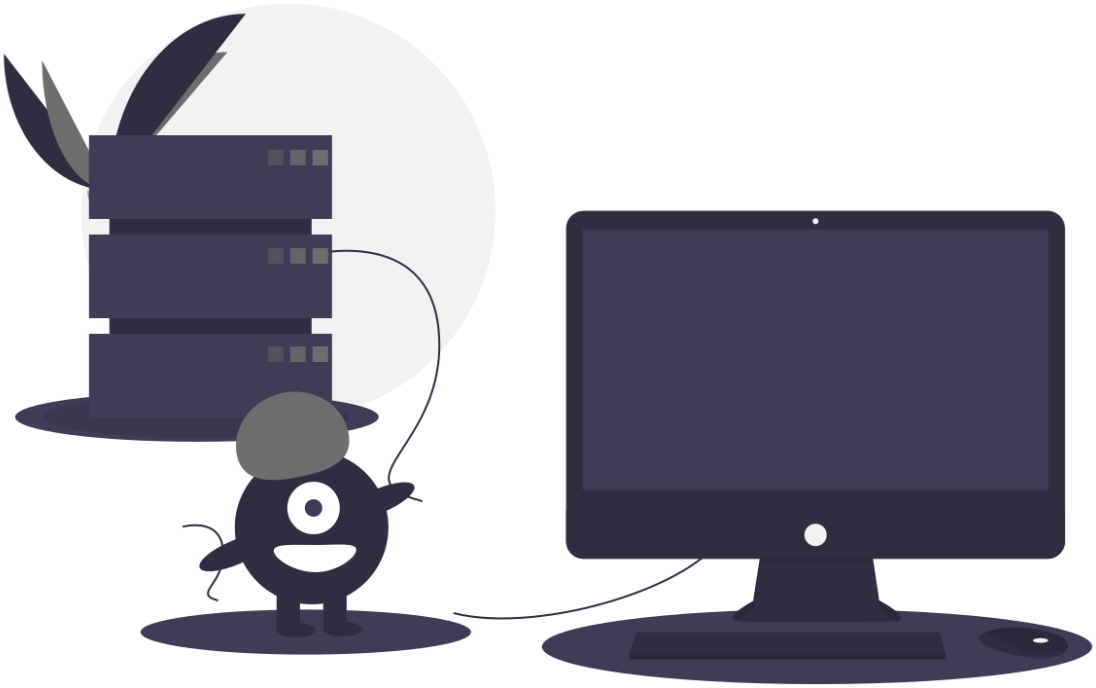
Sprint တွေကို ခွဲလိုက်ရင် စုစုပေါင်း ဘယ်နှစ်ပတ် အတွင်း ပြီးမယ် ဆိုတာ ထွက်လာပါမယ်။ ဒါ ဆိုရင်တော့ project စလို့ ရပါပြီ။ Sprint ခွဲသည့် အခါမှာ work done ကို acual test နဲ့ တွက်ရပါတယ်။ Sprint တစ်ခု ပြီးသွားတယ် ဘာမှ လုပ်မရသေးဘူး ဆိုရင် Sprint ခွဲတာ လွဲနေပါပြီ။ ဥပမာ Sprint 1 ပြီးသွားရင် user login ဝင်ပြီး home screen တွေ ကြည့်လို့ရပြီ။ backend မှာ user တွေ manage လုပ်လို့ရပြီ။ စသည် ဖြင့် acual sprint goal ထားဖို့ လိုပါတယ်။

Agile နှင့် timeline ပြဿနာ

Agile မှာ sprint ခွဲလိုက်ပေမယ့် sprint တစ်ခု ပြီးသွားဖို့ အတော့်ကို ခက်ပါတယ်။ Product owner က sprint ပြီးသည့် အခါမှာ ကြည့်ပြီး design ပြောင်းချင်တာ ပြင်ချင်တာ တွေ ရှိလာနိုင်ပါတယ်။ တစ်ခါတစ်လေ လုပ်ထားသည့် feature တစ်ခု လုံး ဖြုတ်ချပြီး feature နောက်တစ်ခု ထပ်ဖြည့်တာတွေ လုပ်တတ်ပါတယ်။ တစ်ခါတစ်လေ bug တွေ အများကြီး ထွက်လာတာ။ ပြောထားသည့် flow လွဲနေတာ စသည်ဖြင့် sprint တစ်ခု close မလုပ်နိုင်ပဲ အရမ်းကြာသွားနိုင်တာကို Scrum Master က သိဖို့ လိုအပ်ပြီး ဝင်ညှိနိုင်ဖို့ လိုအပ်ပါတယ်။ Scrum Master တွေဟာ feature တွေ လျော့ချပြီး sprint goal ရောက်အောင် product quality မကျအောင် ထိန်းရပါတယ်။ Feature တွေ ထပ်ဖြည့်လို့ product quality ကျသွားလို့ မဖြစ်ဘူး။ Timeline ပို ကြာသွားလို့ မဖြစ်ဘူး။ Project တစ်ခု မှာ Scrum Master ဟာ အရေးပါပါတယ်။

Project တစ်ခု စင်းလုံးချောပြီးဖို့ ဆိုတာ အတော့်ကို ခက်ပါတယ်။ ဒါကြောင့် အရင်စ ထားပေမယ့် Scrum Master, Product Owner ကြားက ညှိနှိုင်းမှု ၊ team ရဲ့ စွမ်းဆောင်ရည် တွေ ပေါ်မူတည်ပြီး project တစ်ခု အောင်မြင်မမြင် ရလဒ် ထွက်လာပါလိမ့်မယ်။

အခန်း ၁၃ :: Backend Infrastructure Design



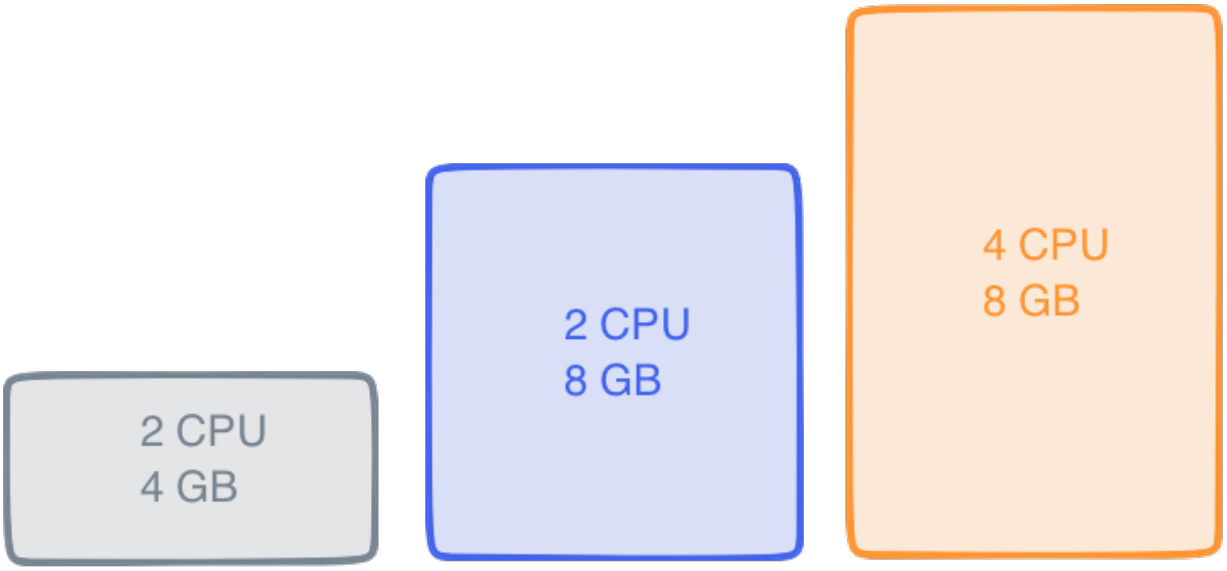
Infra လို့ ပြောလိုက်တာနဲ့ ဦးစွာ ကျွန်တော် မေးနေကျ စကားက scalability ဖြစ်လား မဖြစ်ဘူးလား ဆိုသည့် မေးခွန်းပါပဲ။ Junior Developer တွေဟာ scalability ဆိုတာ ဘာမှန်း မသိတော့ ရေးထားသည့် code တွေဟာလည်း scal လုပ်လို့မရသည့် ပြဿနာတွေ ဖြစ်တတ်တယ်။ Senior တွေဟာလည်း ရှင်းပြဖို့ အချိန်မရှိသည့် အခါမှာ work done ဖြစ်သွားအောင် timleine မှီသွားအောင် junior တွေကို ရေးခိုင်းသည့် အခါမှာတော့ နောက်ပိုင်း scalability issue တက်လာပါတယ်။

Scalability

Scalability လို့ ဆိုလိုတာနဲ့ Horizontal လား Vertical လား ဆိုတာ ကို စပြီး မေးရတာပဲ။ Horizontal scaling သွားမှာလား vertical scaling သွားမှာလား ?

Vertical Scaling

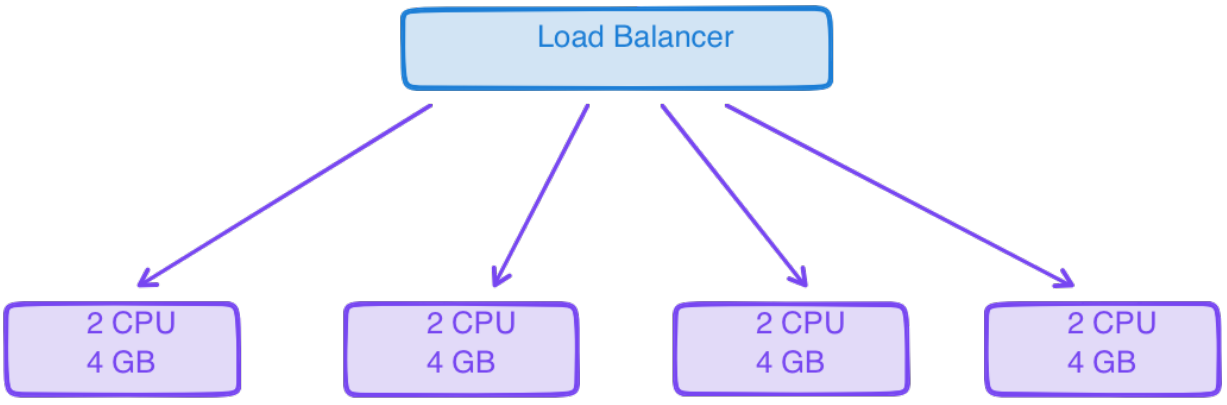
Vertical Scaling ဆိုတာကတော့ မြင်အောင် ပြောရရင် ဒေါင်လိုက်တက်သွားတာပေါ့။ Memory 4GB ကနေ Memory 8GB ကို တိုးလိုက် သလိုမျိုးပေါ့။ Vertical Scaling က အမြန်ဆုံး performance ကို မြှင့်တင်သည့် အခါမှာ အသုံးဝင်ပါတယ်။ Memory မလောက်လို့ Memeory တိုးလိုက်တာ။ CPU usages များနေလို့ တိုးလိုက်တာ။



Vertical Scaling က ကုန်ကျစရိတ် များပါတယ်။ တင်ပြီးသွားရင်လည်း ပြန်လျော့ဖို့ ခက်ပါတယ်။ promotion ကာလမှာ လူသုံးများလာလို့ ၁ လ $20 server ကနေ ၁ လ $80 server ကို ပြောင်းလိုက်တယ်။ promotion လည်းပြီးသွားကော လူသုံးနည်းသည့် အခါမှာတော့ ပြန်ပြီး $20 ကို ပြောင်းဖို့ သိပ်မလွယ်ပါဘူး။

Horizontal Scaling

Horizontal Scaling က တော့ လူသုံးများသည့် Scaling တစ်ခုပါပဲ။ Server အသေးတွေ အများကြီးကို Load Balancer ခံပြီး အသုံးပြုခြင်းပါ။ Horizontal Scaling က Vertical Scaling ထက် အတိုးအလျော့ လုပ်လို့ရပါတယ်။ ဥပမာ Server ရဲ့ Memory ကို 4GB ကနေ 8GB ပြောင်းဖို့ system ကို shutdown လုပ်ပြီးမှ လုပ်လို့ရပါလိမ့်မယ်။ Horizontal Scaling မှာတော့ Server တစ်လုံး ထပ်ဖြည့်လိုက်ရုံပါပဲ။ မလိုအပ်တော့ရင် ပြန်ဖြုတ်လိုက်လို့ရပါတယ်။



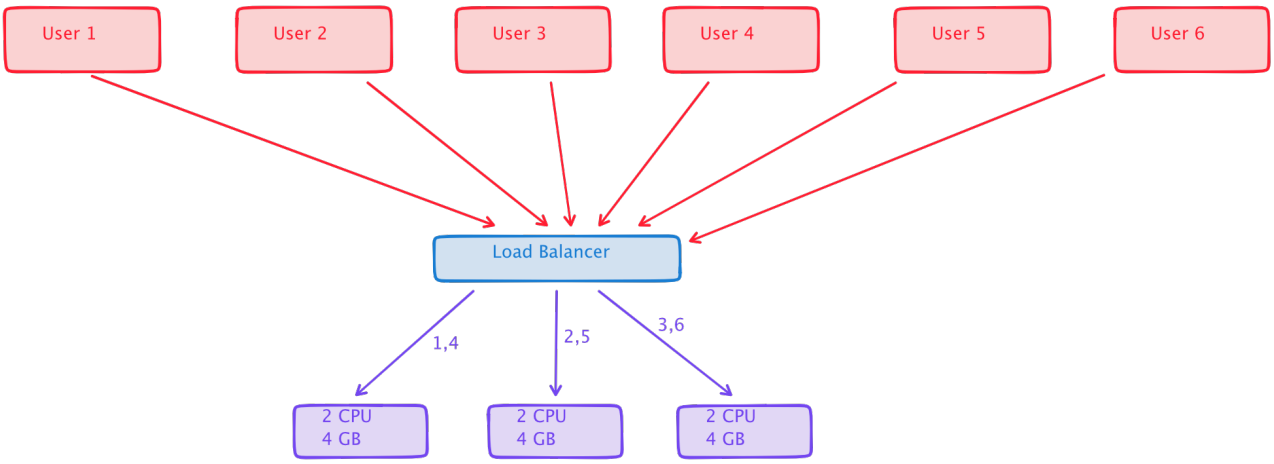
Horizontal Scaling မှာလည်း ပြဿနာတွေ ရှိပါတယ်။ Database ကို vertical မှာလို တနေရာတည်းမှာ ထား လို့ မရတော့ပဲ သီးသန့် ခွဲထုတ်ရပါတယ်။ File Storage ကိုလည်း ဆွဲထုတ်ဖို့ လိုအပ်ပါတယ်။ ခွဲမထုတ်ပဲ server တစ်ခုတည်းမှာထားလိုက်ရင် Server 1 က file ကို server 2 မှာ လာပြပေးမှာ မဟုတ်ပါဘူး။

www.abc.xyz/file1.png လို့ ခေါ်လိုက်ရင် Load Balancer ကနေ server 1 ကို ရောက်လာခဲ့လို့ file ရှိသည့် အခါမှာ ပြပေးပေမယ့် server 2 ကို ရောက်သည့် အခါမှာ file မရှိသည့် အခါမှာ ဖော်ပြ လို့ မရတော့ပါဘူး။

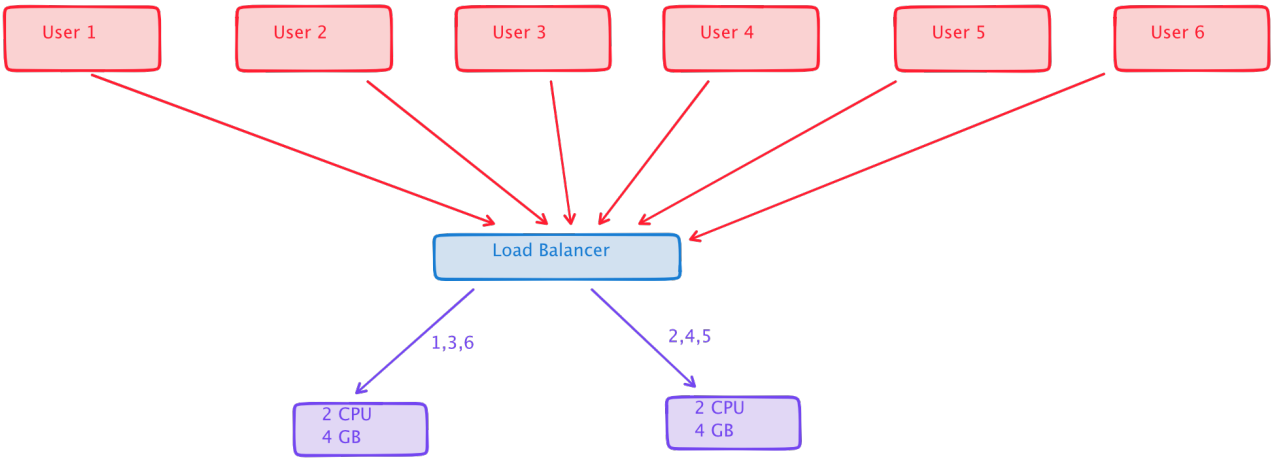
Load Balancer

Load Balancer ကတော့ Server တွေ တစ်ခုထက် မက ရှိသည့် အခါမှာ Public IP တစ်ခုတည်းက နေ ဘယ် server ကို သွားရမလဲ ဆိုတာကို handling လုပ်ပေးပါတယ်။ လူသုံးများတာကတော့ Round Robin Algorithm နဲ့ Least connections ပါ။

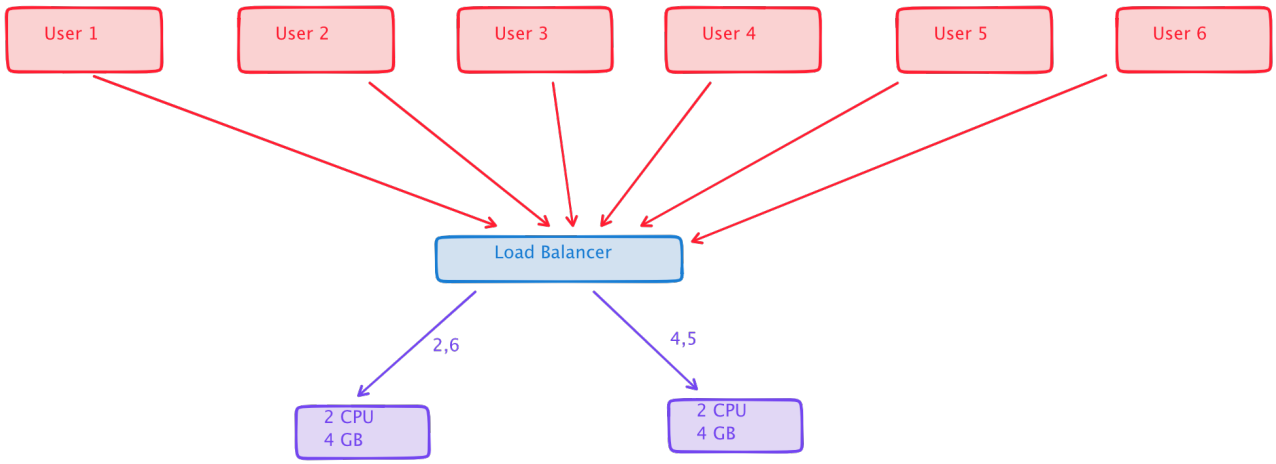
Round Robin ဆိုတာကတော့ အလှည့်ကျ သွားသည့် ပုံစံပါ။ request ကို server တစ်ခုပြီး တစ်ခု ပေါင်းပြီး ခေါ်ပေးသည့် ပုံစံပါ။



Least connections ကတော့ အနည်းဆုံး request ရှိသည့် server ကို ပို့ပေးတာပါ။ ဥပမာ



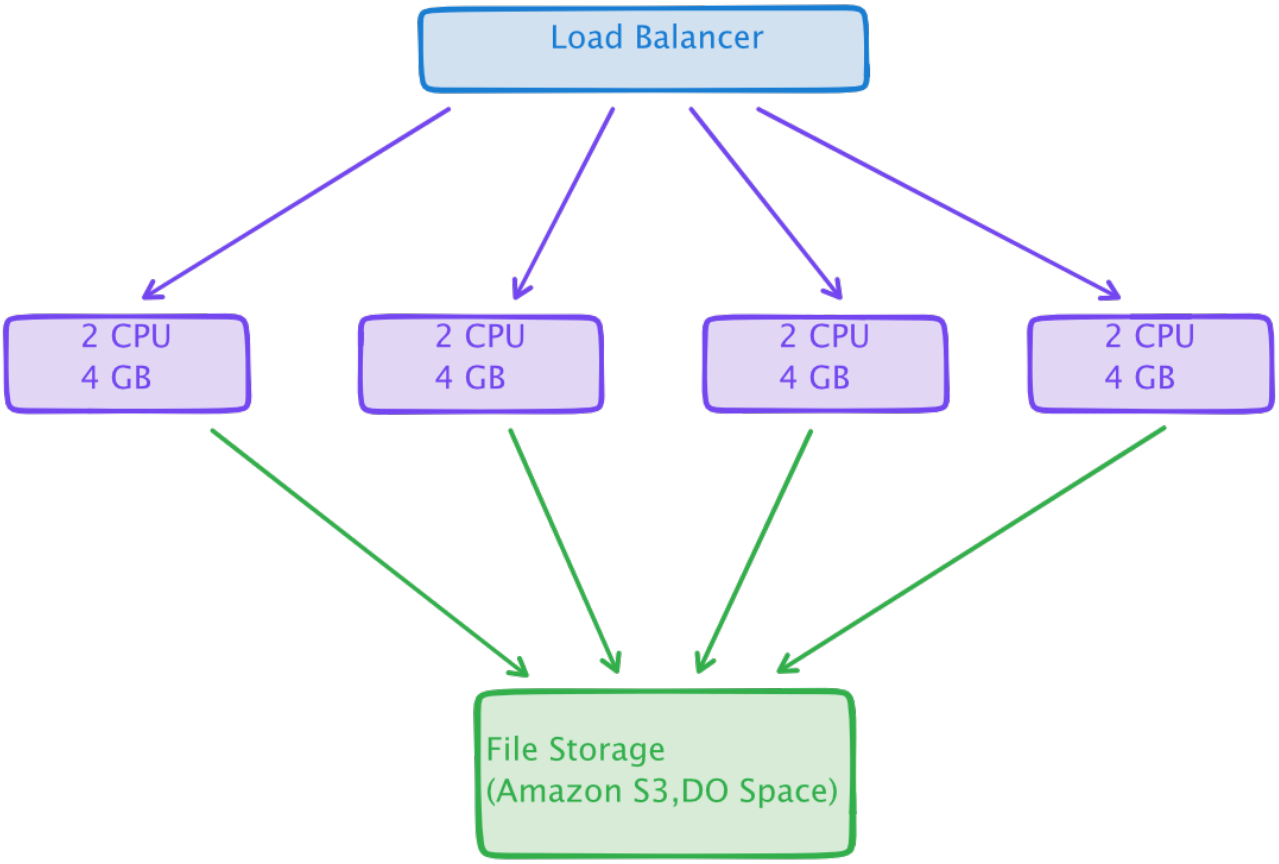
ပထမဆုံး server ၂ ခု မှာ load အပြည့် ရှိတာကနေ 1 နှင့် 3 connection ကို မရှိတော့ဘူး ဆိုပါဆို့။ အဲဒီ အခါမှာ 2 ရဲ့ request တွေက နောက် အခါမှာ server ပြောင်းသွားပါတယ်။



Request တွေက server ရဲ့ load ပေါ်မှာ မူတည်ပြီး ပြောင်းလဲ ပေးတာပါ။ ဒါကြောင့် load balancer ကို အသုံးပြုသည့် အခါမှာ request တွေ ပေါ်မှာ မူတည်ပြီး server အတိုး အချဲ့ လွယ်လင့် တကူ လုပ်လို့ရပါတယ်။

Storage

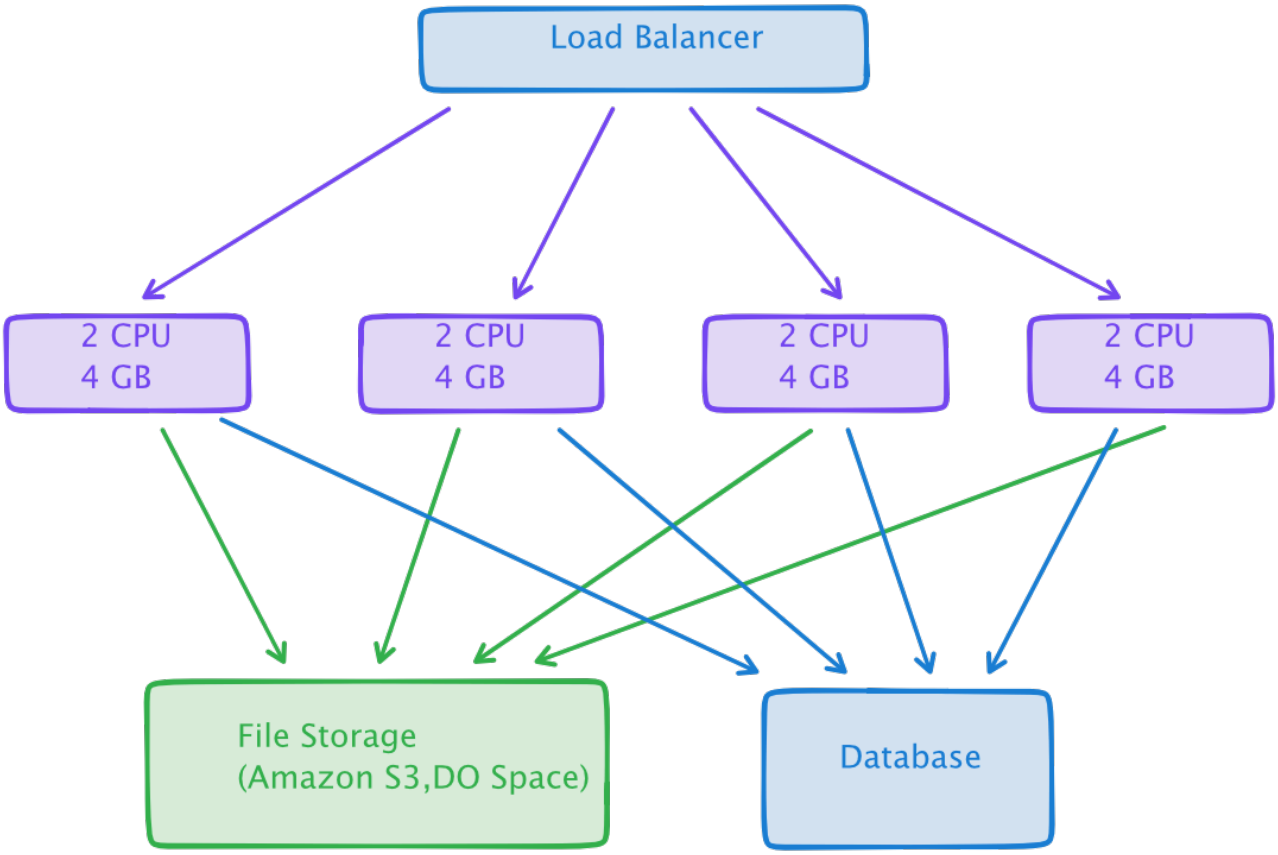
Horizontal Scaling လုပ်သည့် အခါမှာ နောက်ထပ် ပြဿနာ တစ်ခုကတော့ File သိမ်းသည့် ပြဿနာပါပဲ။ Upload လုပ်လိုက်သည့် file ကို ပြန်ပြဖို့ Server တစ်ခုထဲမှာ သိမ်းထားလို့ မရပဲ သီးသန့် ခွဲထုတ်ရပါတယ်။



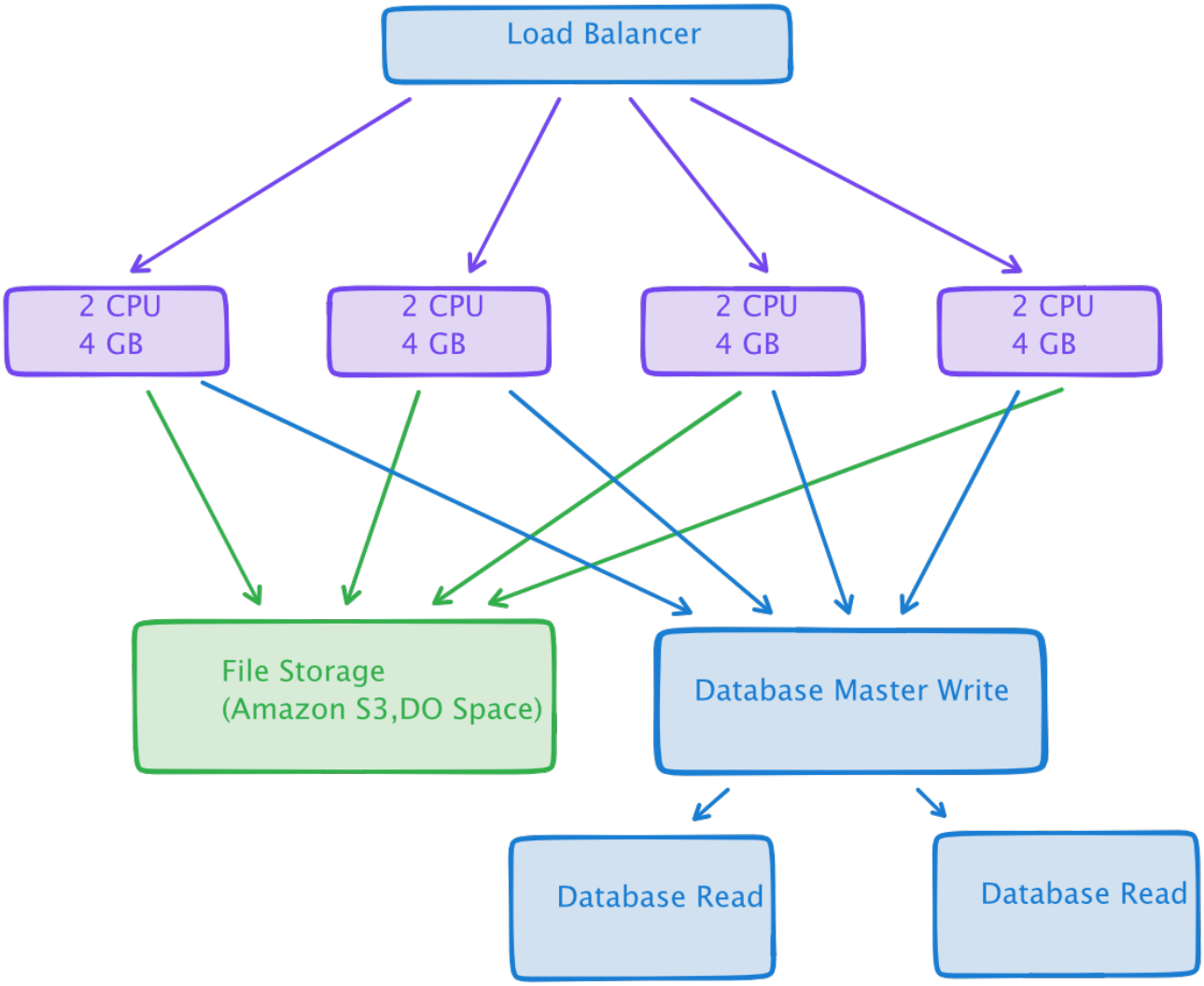
ပုံမှန် အားဖြင့် Amazon S3 , Digital Ocean Space တို့ကို အသုံးပြုပါတယ်။ Cloud File Storage server တွေဟာ ဈေးသက်သာသလို server အကုန်လုံးက file တွေကို API ကနေ တဆင့် သိမ်းဆည်းနိုင်ပါတယ်။

Database (Read/Write)

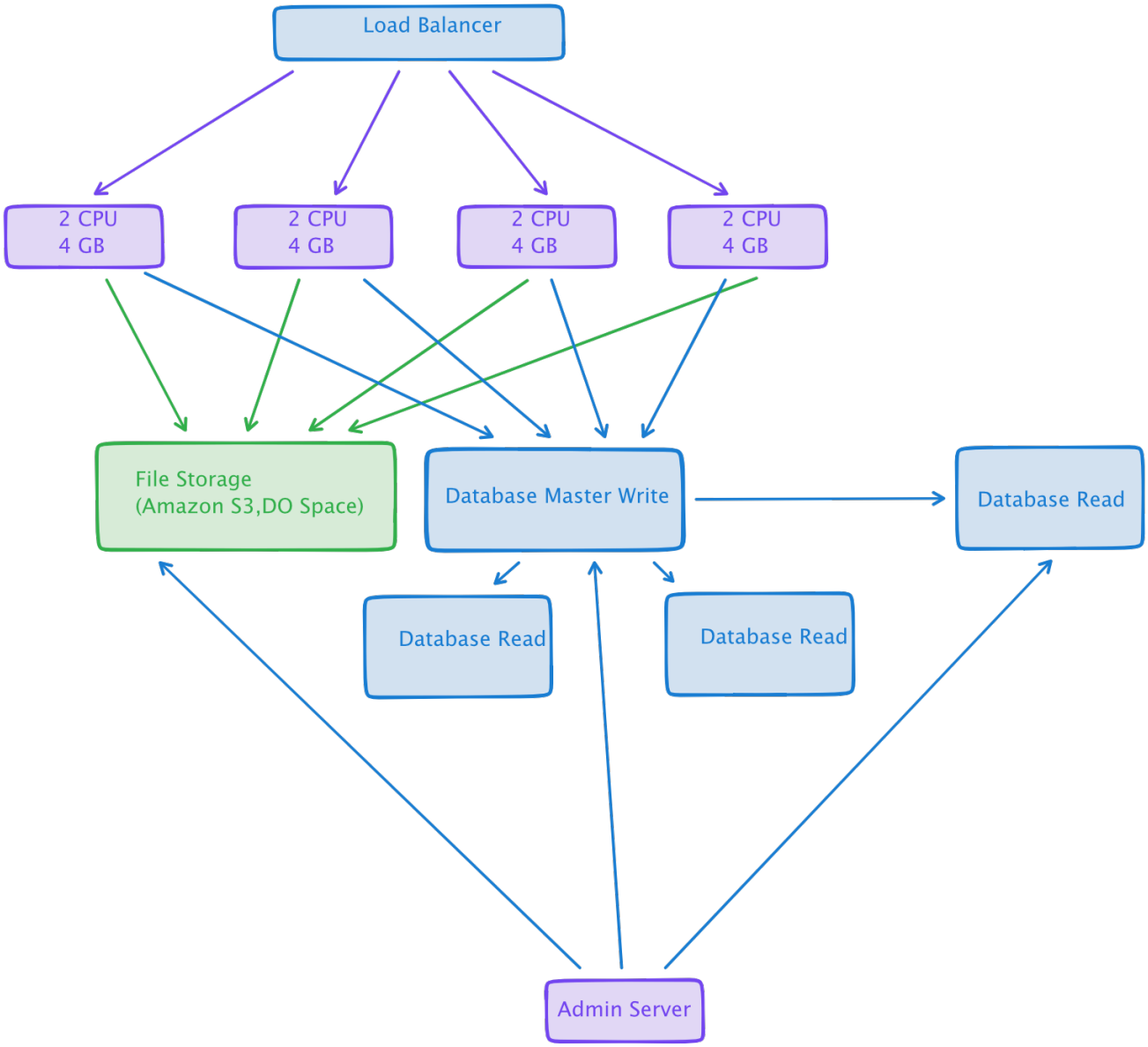
File Storage လိုပဲ database မှာ လည်း တစ်ခုခြင်းဆီမှာ ထားလို့ မရတော့ပါဘူး။ ဒါကြောင့် database ကိုလည်း သီးသန့် ခွဲထုတ်မှသာ အဆင်ပြေပါတယ်။



Database ကို ခွဲထုတ်လိုက်ခြင်းအားဖြင့် server တစ်ခုခြင်းဆီမှာ web server ရဲ့ request တွေပဲ handling လုပ်စရာလိုပါတော့တယ်။ Database ကို load ပေါ်မှာ မူတည်ပြီး vertical scaling တည်ဆောက်လို့ရပါတယ်။

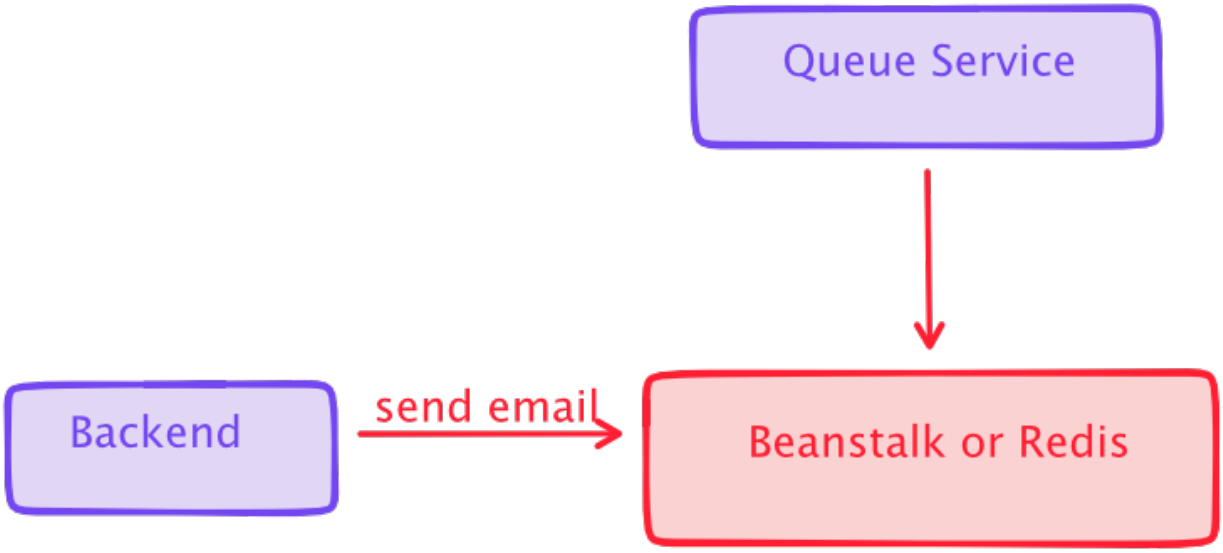


ဒီထက် ပိုကောင်းတာကတော့ Write Server နှင့် Read Server တွေ ခွဲထုတ်လိုက်ခြင်းအားဖြင့် performance ကို ပိုကောင်းစေပါတယ်။ ဥပမာ Admin Panel ကနေ file ထုတ်နေသည့် အချိန်မှာ အဓိက system ကတော့ performance မကျသွားအောင်ပါ။ Admin Panel အတွက် read server တစ်ခုကို သီးသန့် ခွဲထုတ်ပြီး အသုံးပြုကြပါတယ်။



Queue Server

System တွေထဲက data တွေ များလာသည့် အခါမှာ ချက်ချင်း database ထဲကို write ဖို့ မလိုသည့် အခါတွေမှာ queue service တွေကို အသုံးပြုနိုင်ပါတယ်။ ဥပမာ Email ပို့ခြင်း ၊ push notification ပို့ ခြင်းတို့လိုပေါ့။ တစ်ခါတစ်လေ database ကနေ တိုက်ရိုက် export ထုတ်သည့် အခါမှာ လက်ရှိ page က အချိန်အကြာကြီး load ပြီးအောင် စောင့်နေရပါတယ်။ အဲလိုမျိုး ကိစ္စတွေမှာ နောက်ကွယ်ကနေ queue server ကနေ တဆင့် process လုပ်ပြီး email ပို့ပေးခြင်းအားဖြင့် system ကို ပိုပြီး တည်ငြိမ်စေပါတယ်။ Video Encoding တွေလုပ်သည့် အခါမှာ နာရီ နဲ့ ကြာအောင် စောင့်ရတာတွေ ရှိပါတယ်။ Queue Service မပါသည့် အခါမှာ video encoding ကို user တွေ အများကြီးက request လုပ်သည့် အခါမှာ usages တွေ တက်လာပြီး မနိုင်တော့တာတွေ ရှိပါတယ်။ ဒါကြောင့် system design ပေါ်မှာ မူတည်ပြီး queue server ထည့်သွင်း တည်ဆောက်သင့် မသင့် စဉ်းစားရပါတယ်။



Queue အတွက် Redis , Beanstalk အပြင် အခြား နှစ်သက်ရာ service ကို အသုံးပြုနိုင်ပါတယ်။ သတိထားရမှာက redis, beanstalk တို့က data ကို memory ပေါ်မှာ သိမ်းထားတာ ဖြစ်လို့ system restart ချလိုက်ရင် data တွေ ပျောက်သွားနိုင်ပါတယ်။

Cache Server

Developer တစ်ယောက် အနေနဲ့ Cache ကို မဖြစ်မနေ သိထားဖို့လိုတယ်။ Cache ဆိုတာကတော့ တူညီသည့် Process တစ်ခုကို နောက်ထပ် ထပ်လုပ်ဖို့ မလိုပဲ data ကို သိမ်းဆည်းထားပေးတာပါ။

ဥပမာ

```
SELECT * FROM users where id = 5;
```

ဆိုရင် user id 5 ရဲ့ data မပြောင်းမချင်း တူညီသည့် data ပဲ ထွက်လာမှာပါ။ ခဏခဏ မလိုအပ်ပဲ SQL database မှာ သွားပြီး query မလုပ်အောင် cache ခံထားဖို့ လိုပါတယ်။ cache အတွက် **Memcached** , **Redis** စသည့် server တွေ ရှိပါတယ်။ ကိုယ်ရေးမည့် programming နဲ့ အဆင်ပြေမယ့် Cache server ကို အသုံးပြုနိုင်ပါတယ်။

Laravel မှာ ဆိုရင်တော့

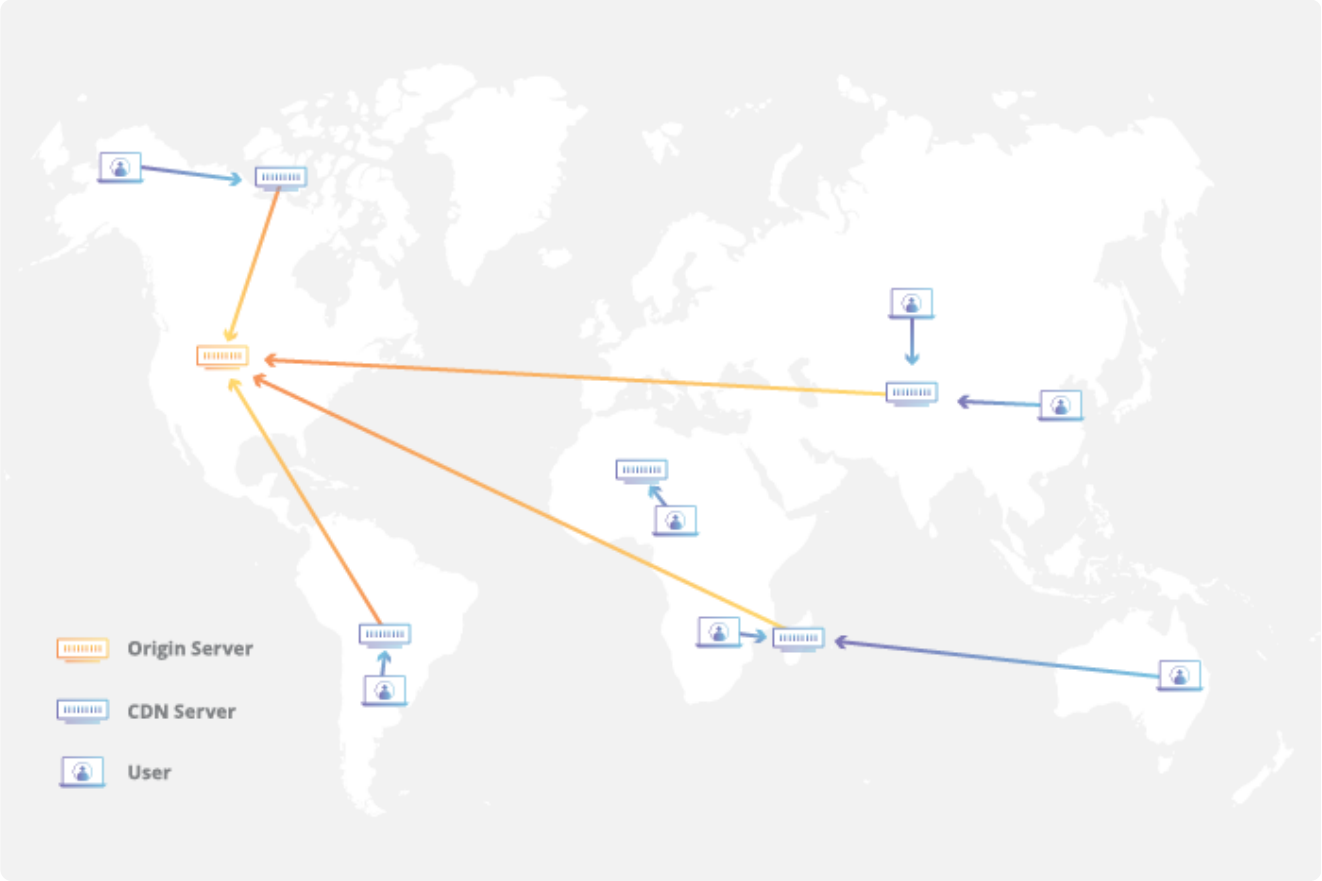
```
$value = Cache::remember('users_5', $seconds, function () {
    return User::where("id",5)->first();
});
```

users_5 ဆိုသည့် cache မရှိတော့မှ database ထဲက query ဆွဲပြီး တစ်ခါတည်း cache ထဲမှာ သိမ်းသွားပါတယ်။

CDN

CDN ဆိုတာ content delivery network ပါ။ ကျွန်တော်တို့တွေ Youtube ကြည့်သည့် အခါမှာ မြန်ပေမယ့် အချို့ video website တွေမှာက နှေးနေတယ် လို့ ခံစားရပါလိမ့်မယ်။ ကိုယ်ပိုင် server နဲ့ video host လုပ်ထားသည့် အခါမှာ တက်လာဖို့က 10 seconds လောက် ကြာပေမယ့် Youtube မှာ 3 seconds လောက်နဲ့ တက်လာတယ်။ ဘာဖြစ်လို့လည်း ဆိုတော့ ကိုယ်ပိုင် server တွေမှာ CDN မသုံးထားကြပါဘူး။ နောက်ပြီး server host ကို ကိုယ်သုံးသည့် နိုင်ငံနဲ့ ဝေးသည့် နေရာမှာထားတတ်လို့ပါ။

ဥပမာ US မှာ server ဈေးသက်သာသည့် အတွက် server location ကို US ရွေးထားတယ်။ ပြည်တွင်း မြန်မာ နိုင်ငံက သူတွေက US server ကနေ data တွေ ပို့သည့် အခါမှာ အတော်လေးကို စောင့်ရပါလိမ့်မယ်။ US server အစား စင်္ကာပူ server ဆို ပိုမြန်တာပေါ့။ ဒါပေမယ့် US က သုံးသည့် သူတွေ အတွက် နှေးသွားပြန်ကော။

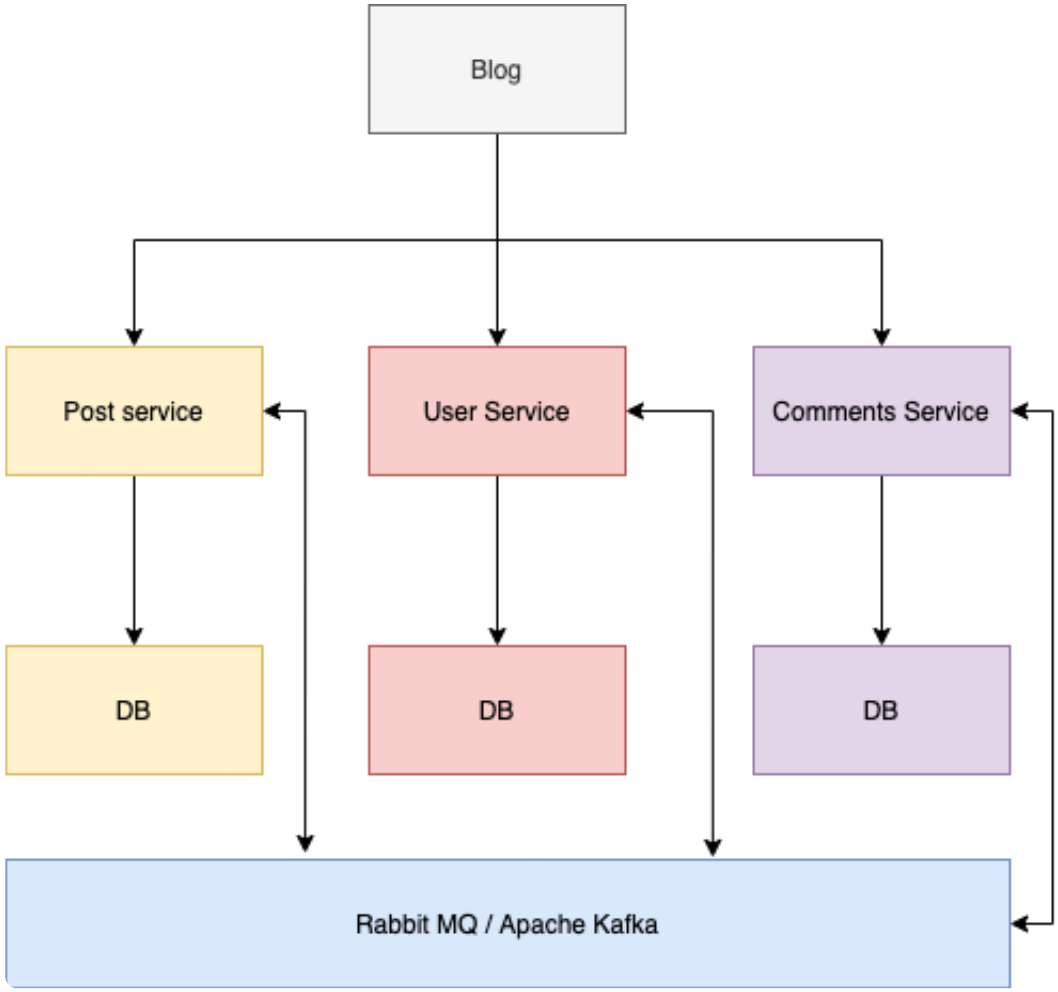


CDN ဟာ static content တွေကို ကမ္ဘာပေါ်က နိုင်ငံ အလိုက် server တွေ မှာ ခွဲထားလိုက်ခြင်းပါပဲ။ ဥပမာ US server က file ကို Bangkok , London , Tokyo စသည့် မြို့တွေ ထဲမှာ clone လုပ်ပြီး သိမ်းထားလိုက်ပါတယ်။ မြန်မာ နိုင်ငံက ဆို ရင် အနီး ဆုံး Bangkok ကို ရောက်သွားပါမယ်။ Korea က သုံးရင် အနီးဆုံး Tokyo server ဆီကနေ data တွေ ရပါလိမ့်မယ်။ Server location နဲ့ အသုံးပြုသူ location နီး စမ်းသွားသည့် အတွက်ကြောင့် load တက်လာတာ မြန်လာပါလိမ့်မယ်။

Microservices

System တွေကြီးလာသည့် အခါမှာ database က record တွေလည်း များလာပါတယ်။ Monolith system မှာ services တွေ အကုန်လုံးက တစုတစည်းထဲ ဖြစ်နေတတ်သလို database ဟာလည်း တစ်ခုတည်းမှာ ရှိနေတတ်ပါတယ်။ Transac†ions ၅ သန်းလောက် ရှိလာသည့် အခါမှာ transaction ထဲမှာ row တစ်ကြောင်းထည့်လိုက်သည့် အခါမှာ အခြား system တွေ ပါ လေး သွား တာ မျိုး ဖြစ်တတ်ပါတယ်။ User က အချို့ အပိုင်း သေးသေးလေး ပြင်သည့် အခါမှာ System တစ်ခုလုံးပြန်ပြီး deploy လုပ်ရပါတယ်။ Monolith ဟာ ကောင်းတာတွေလည်း ရှိသလို အဆင်မ ပြေတာတွေလည်း ရှိတတ်ပါတယ်။

တချို့ system တွေဟာ ကြီးလာသည့် အခါ team ကလည်း ကြီးလာသည့် အခါမှာတော့ Microservices ကို ပြောင်းကြပါတယ်။ Microservices ကို အကြမ်းအား ဖြင့် အောက်မှာ ဖော်ပြပါ ပုံအတိုင်းပါပဲ။



Microservices မှာ ကိုယ်ပိုင် services တွေ အတွက် ကိုယ်ပိုင် database ကို အသုံးပြုပြီး အသေး လေးတွေ ခွဲထုတ်ထားပါတယ်။ Users system ပြင်ဆင်ထားတာတွေကို deploy လုပ်သည့်အခါမှာ အခြား services တွေ သွားမထိတော့ပါဘူး။ User service ကို java နဲ့ ရေးပြီး post service ကို node.js ၊ comment service ကို PHP နဲ့ ရေးလည်း ရပါတယ်။

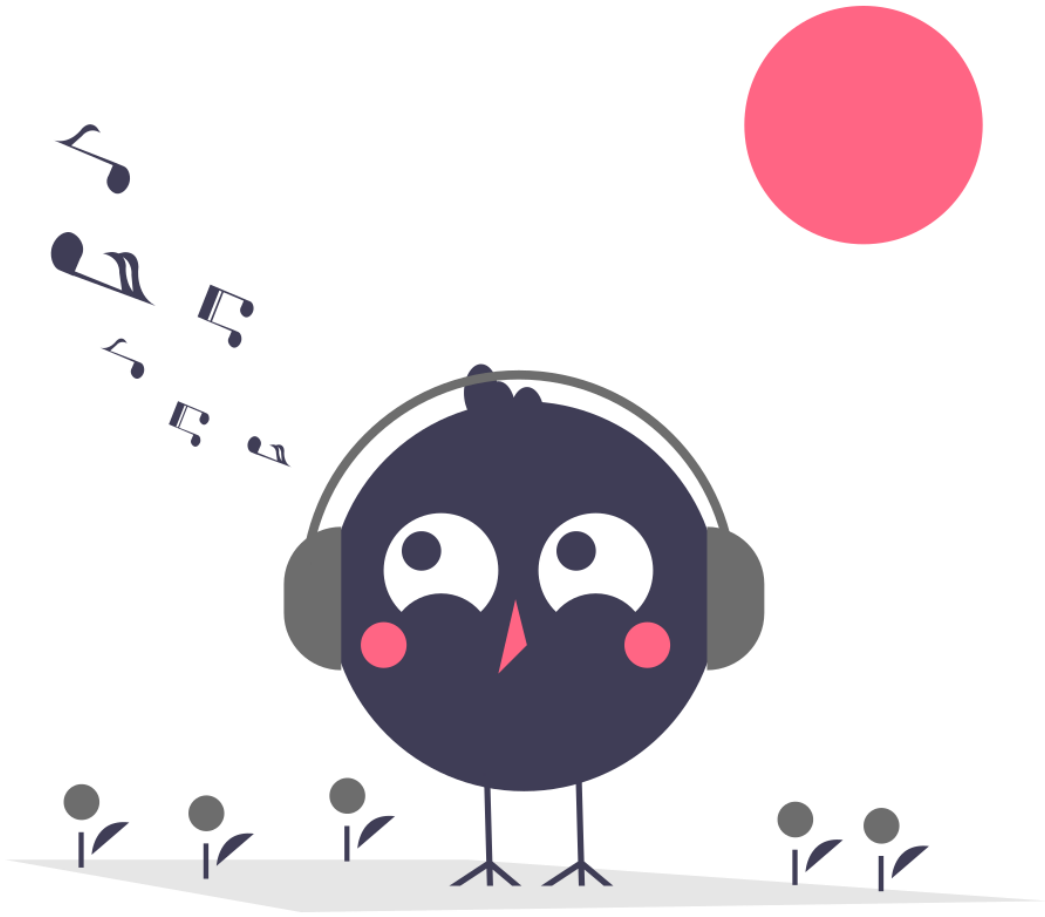
Cloud Server

Cloud Server ဟာ လက်ရှိ သမာရိုးကျ server တွေလို မဟုတ်ပဲ scaling ကို လိုသလို ချက်ခြင်း ဆောင်ရွက်နိုင်တယ်။ Vertical scaling ပဲ ဖြစ်ဖြစ် horizontal scaling ပဲ ဖြစ်ဖြစ် မြန်မြန်ဆန်ဆန် ဆောင်ရွက်နိုင်တယ်။ Database ကိုလည်း performance ကောင်းအောင် ချက်ချင်းပြင်ဆင် ပြောင်းလဲနိုင်တယ်။ အခုခေတ်မှာ လူတိုင်းနီးပါး cloud server တွေ အသုံးပြုနေကြပါပြီ။ AWS ကို မတတ်နိုင်ရင်တောင် Digital Ocean ပေါ်မှာ စမ်းကြည့်လို့ရပါတယ်။

Amazon Web Service လို့ လူသိများသည့် AWS ဟာ အခုခေတ် နေရာတော်တော်များများမှာ အသုံးပြုနေပါပြီ။ AWS အတွက် certificate ကို web developer တွေ အနေနဲ့ ဖြေထားသင့်တယ်။ အလွတ်ကျက်သည့် အစား တကယ့်ကို lab နဲ့ လေ့လာပြီး ဖြေထားဖို့ လိုအပ်တယ်။ AWS က အရေးပါသည့် နေရာကို ရောက်နေပါပြီ။ System တော်တော်များများက AWS ပေါ်မှာ run နေကြ ပါတယ်။ AWS ကို မသိပဲ ကိုယ့်လက်ထဲရောက်လာလို့ manage လုပ်မယ် ဆိုရင် အန္တာရယ် အလွန်များပါတယ်။ မှတ်မှတ်ရရ AWS ကို ကျွန်တော် ပထမဆုံး သုံးသင့် အချိန်က EBS ကို ခွဲ ပြီး မသုံးတတ်သလို configuration လည်း မလုပ်တတ်ပါဘူး။ ရုံးမှာ AWS သုံးကြမယ် AWS ပေါ် တင်ဆိုပြီး ဒီအတိုင်း တင်လိုက်ကြတာ။ Services တွေ down ပြီး internet က မထွက်သည့် အချိန် မှာ instant ကို terminate လုပ်မိလိုက်တယ်။ AWS အကြောင်းမသိတော့ terminate က ecovery လုပ်လို့ မရတာ မသိလိုက်ဘူး။ Production server ဖြစ်လို့ နောက်ဆုံး data တွေ အကုန်ပျက်သွား တာ ဖြစ်ခဲ့ဖူးတယ်။ AWS ပေါ်ကာစ မို့ အချို့ စာတွေကို မလေ့လာပဲ ပုံမှန် linux server လိုပဲ ထင် ပြီး လုပ်ခဲ့မိတာ ကြောင့် အတော်ကို နောင်တရ မိတယ်။ နောက်ပြီး database server ကို ခွဲပြီး သုံး ရမယ်ဆိုတာကို မသိခဲ့တာကြောင့်လည်း ပါတယ်။

AWS မှာ အခြား platform တွေမှာ မရှိသည့် feature တွေ အများကြီး ရှိသလို အမြဲလို update ဖြစ် နေတယ်။ အခုအချိန်မှာ AWS အပြင် Google Cloud , Microsoft Azure တို့ဟာလည်း နေရာတစ် ခု ရလာပါပြီ။ AWS ကို အသုံးပြုမယ်ဆိုရင် certificate ရှိထားရင်တော့ အများကြီး ပိုပြီးသင့် တော်ပါလိမ့်မယ်။ ဒါကြောင့် AWS certificate တွေ အချိန်ရရင် လေ့လာပြီး ဖြေထားသင့်ပါ တယ်။

အခန်း ၁၄ :: Program အသေးလေးများ



ပုံမှန်အားဖြင့် junior developer တွေဟာ programming language တစ်ခုတည်းကို ပဲ လေ့လာကြပါတယ်။ လုပ်ငန်းခွင်ထဲမှာ programming language အပြင် os နဲ့ ပတ်သက်သည့် shell script တွေ python , javascript စသည်ဖြင့် program အသေးလေးတွေ အတွက် ရေးထားတတ်အောင် လေ့လာထားရင် ပိုအဆင်ပြေပါတယ်။

တစ်ခါတစ်လေ program အကြီးကြီး ရေးမယ့်အစား batch shell နဲ့ ပြီးသွားနိုင်ပါတယ်။ ဥပမာ redis ထဲက data တွေကို ထုတ်ဖို့ အတွက် PHP သို့မဟုတ် Python အစား shell script နဲ့ loop ပတ်ပြီး ထုတ်လိုက်ခြင်းက ရေးရတာ ပိုပြီး မြန်ဆန် ပါတယ်။

Shell script တွေကို database backup လုပ်သည့် နေရာတွေမှာ နောက်ပြီးတော့ database ထဲကနေ custom query ထုတ်ပြီးတော့ ftp server ပေါ်တင်ရသည့် နေရာတွေမှာလည်း အသုံးဝင်လှပါတယ်။

နောက်ပြီး personal project တွေမှာလည်း အသုံးဝင်လှပါတယ်။ လိုချင်သည့် website ကနေ data ဆွဲချပြီး လိုအပ်သည့် data ကို ဆွဲထုတ်သည့် အခါမှာ node-js က အတော်လေးကို အသုံးဝင်ပါတယ်။

တစ်ခါတစ်လေမှာ လက်ရှိ အသုံးပြုနေသည့် programming language အတွက် support library မရှိတာ များရှိတတ်ပါတယ်။ ဥပမာ machine learning သုံးပြီး မျက်နှာ ကို analyst လုပ်သည့် system က python နဲ့ပဲ တွဲသုံးလို့ရတယ် ဆိုပါတော့။ လက်ရှိ ရေးနေသည့် php အတွက် မရှိဘူး။ ဒီလို ကိစ္စမျိုးမှာ language ထက် အလုပ်ပြီးမြောက်အောင် python ကို သုံးပြီး service တစ်ခု အနေနဲ့ ခွဲထုတ်ပြီး အသုံးပြုရတာ များရှိပါတယ်။ Tensorflow လိုမျိုး အသုံးပြုဖို့ လိုလာပြီဆိုရင် python ကို နောက်ကွယ်ကနေ အသုံးပြုပြီး ရေးသားရပါတယ်။ web scraping လိုမျိုးဆိုရင် node-js ကို အသုံးပြုပြီး ခွဲထုတ် ရေးထားရတာ များရှိပါတယ်။

ဒါကြောင့် langauge တစ်ခုထဲကို ကိုင်ဆွဲထားမယ့် အစား shell script, javascript တို့ကိုလည်း မဖြစ်မနေ လေ့လာထားသင့်တယ်။ programming အခြေခံကို သဘောပေါက်သွားရင် နောက်ထပ် language တစ်ခုကို လေ့လာရတာ မခက်ခဲတော့ပါဘူး။ ဒါကြောင့် အချိန်ရရင် shell script, node-js (javascript), python စသည်တို့ကို လေ့လာထားစေချင်ပါတယ်။

Cron Job

Cron Job ကတော့ schedule အတွက် အသုံးပြုပါတယ်။ Linux server အသုံးပြုသူတိုင်း မသိ မဖြစ်ပါ။

Cronjob ကို edit လုပ်ဖို့ အတွက်

```
crontab -e
```

ဆိုရင် schedule ပြင်ဖို့ အတွက် တက်လာပါမယ်။ Linux server အသုံးပြုမယ်ဆိုရင် **vim** ကို သိထားရင် ပိုကောင်းပါတယ်။

crontab format က

```
[minute] [hour] [day of month] [month] [day of week] [command]
```

ဘယ်မိနစ် ဘယ်နာရီ ဘယ်ရက် ဘယ်လ ဘယ်နေ့ ဘာလုပ်မယ်

နေ့တိုင်း ည ၁၀ နာရီခွဲ မှာ backup script run မယ် ဆိုရင်

```
30 22 * * * sh backup.sh
```

လတိုင်း ၁ ရက်နေ့မှာ မနက် ၅ နာရီ ၁၅ မှာ system တစ်ခု လုံး backup လုပ်သည့် script run မယ် ဆိုပါဆို့။

```
15 5 1 * * sh wholebackup.sh
```

နှစ်တိုင်း ၁ လ ပိုင်း ၁ ရက်နေ့ မနက် ၉ နာရီ မှာ happy new year စာပို့သည့် script လုပ်မယ် ဆို ရင်

```
0 9 1 1 * sh happynewyear.sh
```

Friday ညနေ ၅ နာရီမှာ မီးတွေ ပိတ်ဖို့ တို့ လိုအပ်သည့် ပစ္စည်းတွေ စစ်ဖို့ အတွက် ရုံးက လူတွေ ကို alram ပေးချင်တယ်။ script ကို သောကြာနေ့ တိုင်း ညနေ ၅ နာရီမှာ run မယ်။

```
0 17 * * 5 sh alert.sh
```

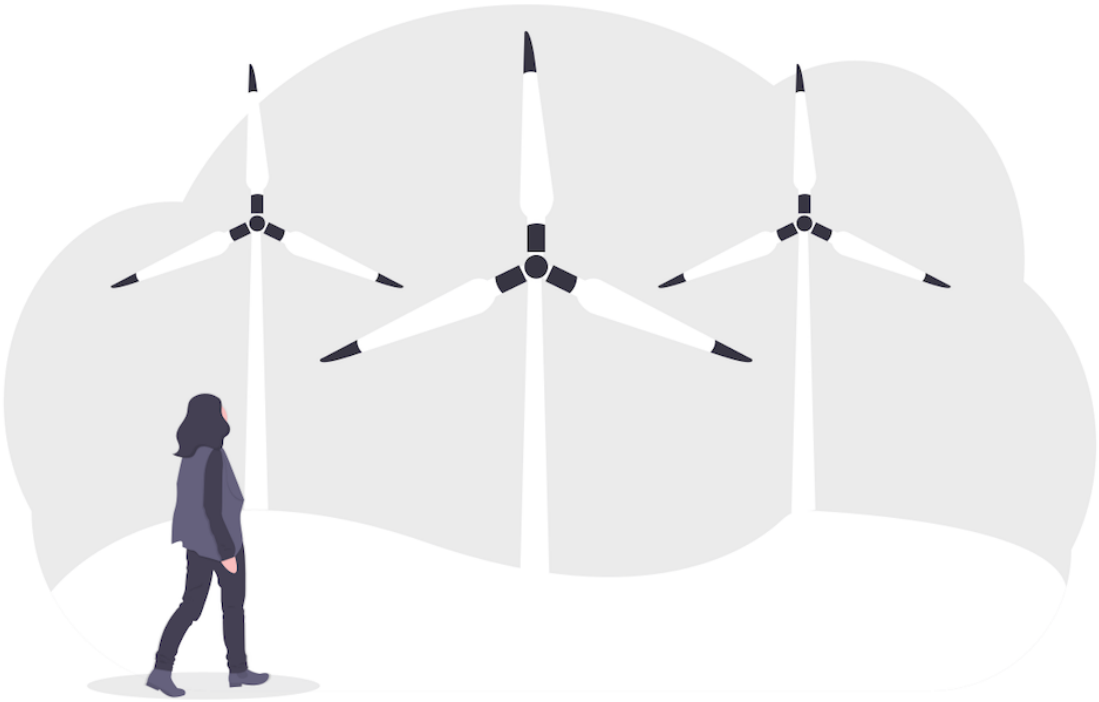
ဒါဆိုရင်တော့ သဘောပေါက်မယ် ထင်ပါတယ်။ shell scriprt တွေ ရေးထားပြီးတော့ cron job နဲ့ အသုံးပြုလို့ ရသလို laravel မှာ command ကို ရေးထားပြီး system ကနေ အလုပ်လုပ်ပေးလည်း ရပါတယ်။

ဥပမာ။။ database က data တွေ တနေ့သာ လူဝင်စာရင်း count လုပ်မယ် အခြား analystic အတွက် ခွဲထုတ်တာတွေကို ညဘက်မှာ လုပ်မှာ ဖြစ်သည့် အတွက် ကြောင့် ဒီလို ရေးထားလို့ရပါ တယ်။

```
1 0 * * * /usr/bin/php /var/www/myproject/artisan run:analystic
```

Programming ရေးသည့် အခါမှာ scheduling က မဖြစ်မနေ ပါဝင်လာပါလိမ့်မယ်။ ဒါကြောင့် cron job က အသုံးဝင်လို့ သိထားသည့်ပါတယ်။

အခန်း ၁၅ :: Clean Architecture



Application တစ်ခု ဖန်တီး သည့် အခါမှာ layers တွေ use case တွေကို ခွဲထုတ်ပြီး ရေးသားကြပါတယ်။ Model View Controller , Model View View Model (MVVM) စသည့် software design pattern တွေ အပြင် Clean Architecture ကိုပါ လေ့လာ ထားဖို့ လိုပါတယ်။ Clean Architecture ဟာ MVC နဲ့ ဖြစ်ဖြစ် MVVM နဲ့ ဖြစ်ဖြစ် အဆင်ပြေပါတယ်။

The Clean Architecture က Clean Code ရေးသည့် Uncle Bob မိတ်ဆက်ထားတာပါ။ နောက်ပိုင်းမှာ Flutter, Android, iOS စသည့် mobile app development တွေ မှာ အသုံးများ သလို server side တွေမှာလည်း အသုံးပြုနိုင်ပါတယ်။ ဒါကြောင့် အခုမှ အလုပ်စဝင်မည့် သူများ အနေနဲ့ မဖြစ်မနေ သိထား လေ့လာသင့်ပါတယ်။ ကိုယ့် senior တွေဟာ ဘာကြောင့် folder တွေ အများကြီး ခွဲထားတယ်။ File တွေ အများကြီး ခွဲထားတယ် ဆိုတာကို နားလည် လာပါလိမ့်မယ်။

Data Flow

1. View (UI) ဟာ ViewModel(Presenter) နဲ့ တွဲပြီး အလုပ်လုပ်ပါတယ်။

2. ViewModel ဟာ Use Case တွေကို အသုံးပြုတယ်။
3. Use Case တွေဟာ Repositories တွေကနေ တဆင့် data တွေ ရပါတယ်။
4. Repository ဟာ Network (API) , Database , Cache စသည့် နေရာက data တွေကို ထိန်းချုပ်ပြီး return ပြန်ပေးတာပါ
5. နောက်ဆုံးမှာတော့ Data တွေကို UI မှာ ပြပေးပါတယ်။

အဲဒီတော့ Layer တွေကို ကြည့်ရအောင်

1. Presentation Layer
2. Domain Layer
3. Data Repositories Layer

ဆိုပြီး ရှိပါတယ်။

တချို့တွေကတော့

1. Presentation Layer
2. Service Layer
3. Domain Layer
4. Integration Layer
5. Data Repositories Layer

ဆိုပြီး ခွဲသုံးကြတာ ရှိပါတယ်။

Presentation Layer

ဒီ Layer ကတော့

- ViewModel
- View (UI)

တို့ ရှိသည့် layer ပါ။

View Model က ရသည့် Data တွေကို View မှာ ပြပေးတာပါ။ View ကို update လုပ်ဖို့ အတွက် အရင်က နည်းလမ်းတွေ အများကြီး ရှိပေမယ့် နောက်ပိုင်း State တွေ ရှိလာသည့်အခါမှာ MVVM ပုံစံနဲ့ ရေးသားခြင်းဟာ မခက်ခဲတော့ပါဘူး။ React, Flutter, SwiftUI, Android Jackpack Compose တို့မှာ တွေ့နိုင်ပါတယ်။

အပေါ်မှာ ပြောခဲ့သလို View ကတော့ Data ပြဖို့ အတွက်ပဲ အသုံးပြုပြီးတော့ View Model က data တွေကို ပြဖို့ ထိန်းချုပ်ပါတယ်။

Domain Layer

Domain Layer ဟာ app တစ်ခု မှာ အရေးကြီးပါတယ်။ Domain Layer မှာ

- Entities (Model)
- Use Cases
- Repositories Interfaces

တွေ ရှိပါတယ်။

Entities ကတော့ Model နဲ့ အတူတူပါပဲ။ Use Case ကတော့ Business logic မှာ အသုံးပြုမယ့် Use Case တွေပါ့။ ဥပမာ Movie App မှာ `SearchMoviesUseCase` , `FetchRecentMoviesUseCase` , `SubscribeUseCase` စသည့် Use Case တွေကို ခွဲထုတ်ထားပါတယ်။ ဒီ layer မှာ Repositories Interface ပဲ သုံးပါတယ်။ Repositories ကို implementation မလုပ်ပဲ business idea တွေ ရှိသည့် function တွေကို interface အနေနဲ့ ရေးပါတယ်။ Domain layer တွေကတော့ business logic တွေ ပဲ ရှိပါမယ်။

Data Repositories Layer

ဒီ layer မှာ နောက်ထပ် layer ထပ်ခွဲပြီး ရေးပါမယ်။

1. Data Layer
2. Infrastructure Layer (Network)

Data Layer

ဒီ Layer မှာ Repositories ကို implement လုပ်ပါတယ်။ Repositories ကတော့ Use Case တွေ နဲ့ တွဲသုံးပါတယ်။ Repositories class တွေကို Use Case တစ်ခု သို့မဟုတ် တစ်ခု ထက် မက ယူသုံးတာ ဖြစ်နိုင်သလို Use Case တစ်ခုကလည်း Repositories နှစ်ခု ကို ခေါ်သုံးတာလည်း ဖြစ်နိုင်တယ်။ ဥပမာ

`MoviesRepository` ဆိုရင် `SearchMoviesUseCase` , `FetchRecentMoviesUseCase` ကပါ ယူ သုံးနိုင်ပါတယ်။

Data Layer မှာ iOS မှာဆိုရင် CoreData Entities လိုမျိုး Java မှာ ဆိုရင် DTO လိုမျိုးတွေ ပါဝင်ပါတယ်။

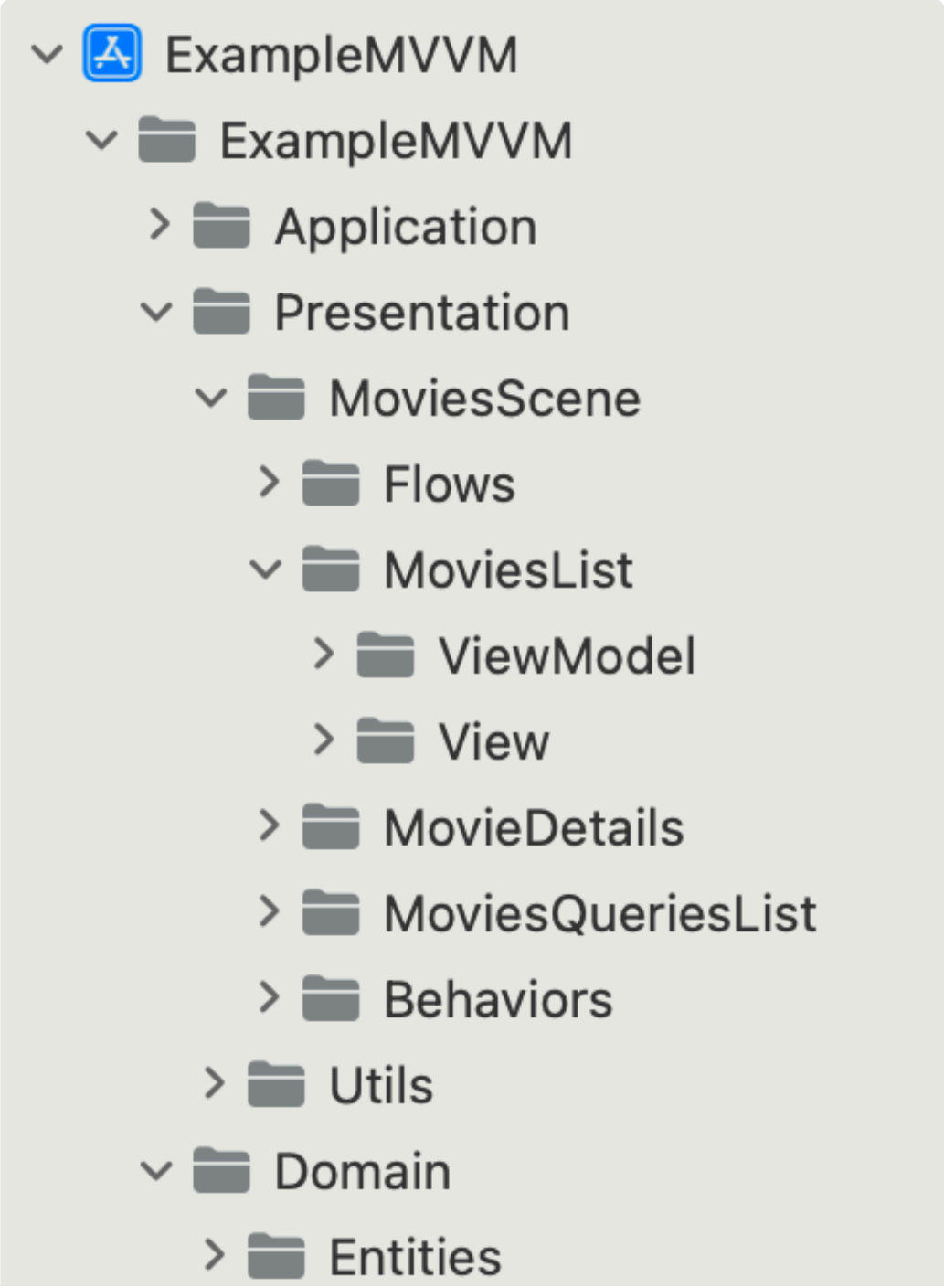
Infrastructure Layer

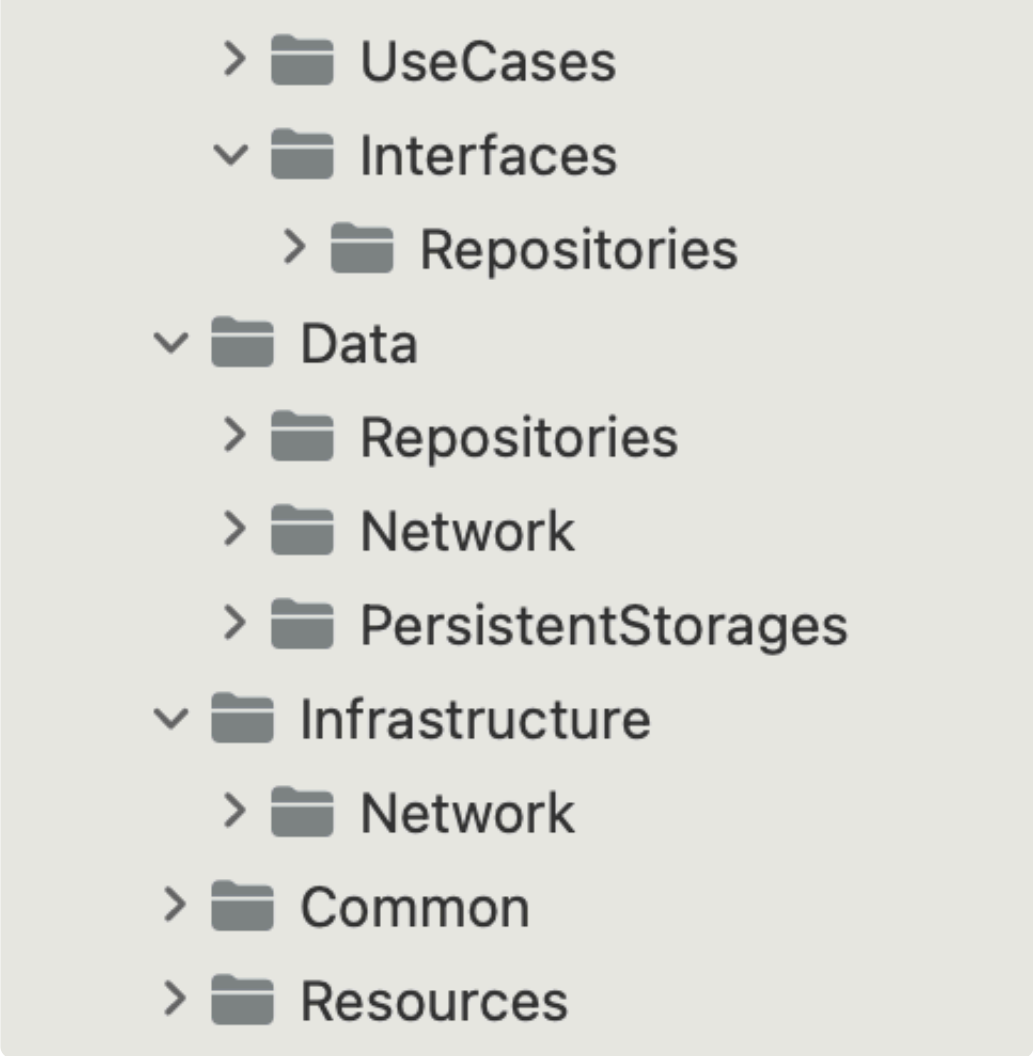
နောက်ပြီး Network (API) call တွေကို လည်း Repositories ကနေ တဆင့် ခေါ်ပြီး ရေးပါတယ်။ ထိုနည်းတူ Database တွေကိုကော ဒီ layer မှာ ပါဝင်ပါတယ်။

ဥပမာ `MoviesRepository` ကို implement လုပ်ပြီးတော့ Movie Search အတွက် Network (API) ကို ခေါ်ပါမယ်။ Server side တွေ ဘက်မှာ ဆိုရင် Database ကနေ query လုပ်တာလည်း ဖြစ်နိုင်ပါတယ်။ တနည်းဆိုရင် Infrastructre Layer က Repositories တွေက လိုအပ်သည့် data တွေ ကို return ပြန်ပေးဖို့ပါ။

Real world

တကယ့် real world မှာ Clean Architecture ကို အသုံးပြုသည့် အခါမှာ project သေးသေးလေး တွေ အတွက် အဆင်မပြေပါဘူး။ Project အကြီးတွေ အတွက်တော့ အဆင်ပြေတယ်။ အထူးသဖြင့် Flutter, React, iOS, Android လိုမျိုး mobile project တွေကို ဖန်တီး သည့် အခါမှာ MVVM နဲ့ တွဲသုံးခြင်းအားဖြင့် နောက်ပိုင်းမှာ project ကို ထိန်းသိမ်း ရတာ အဆင်ပြေစေပါတယ်။





Source code from : https://github.com/kudoleh/iOS-Clean-Architecture-MVVM

Real World project တွေမှာ Screen ၂ ခုပဲ ပါသည့် Project ဖြစ်ခဲ့ရင်တောင် folder တွေ file တွေ အများကြီး ဖြစ်နေတာကို တွေ့နိုင်ပါတယ်။ Project ကြီးလာသည့် အခါမှာတော့ အတော်ကို အဆင်ပြေပါတယ်။ နောက်ပြီးတော့ Clean Architecture က စာအုပ်ထဲကနေ တစ်ပုံတည်း ဖြစ်ချင်မှ ဖြစ်ပါလိမ့်မယ်။ Language တွေ framework တွေ ပေါ်မှာ မူတည်ပြီး ပြောင်းလဲ နိုင်ပါတယ်။

အခန်း ၁၆ :: Test Driven Development

အခုခေတ်မှာ Test Driven Development ကို မဖြစ်မနေ အသုံးပြုနေကြပါပြီ။ Junior Developer တွေဟာ Test Driven Development ကို Unit Test ရေးသားခြင်းလို့ပဲ ထင်မှတ်ထားကြပါတယ်။ TDD ဟာ ဒီထက် ပိုပါတယ်။ ပုံမှန် အားဖြင့် project ရေးပြီးသည့် အခါမှ Uni Test ကို ထည့်သွင်းသည့် အခါမှာ အချိန်ကုန် လူပင်ပန်းတယ်။ အပိုတွေ လို့ ထင်ရပါတယ်။ Project ပြီးမှ Test ရေးခြင်းကတော့ မဖြစ်သင့်ပါဘူး။

TDD ကို နားလည်ဖို့ အတွက် အရင်ဆုံး Application မှာ layers တွေ ခွဲရေးသင့်ကြောင်းကို ပြီးခဲ့သည့် အခန်းမှာ ဖော်ပြခဲ့ပါတယ်။ ကျွန်တော်တို့တွေဟာ Test တွေကို အရေးပါသည့် function တစ်ခု ပြီးတိုင်းမှာ ထည့်သွင်းရေးသားပါတယ်။ ဥပမာ Business Logic တစ်ခု ပြီးတိုင်းမှာ Unit Test ကို ရေးသားပါတယ်။

Blog system တစ်ခုကို backend မှာ ရေးတယ် ဆိုပါဆို့။ Post တွေ အတွက် CRUD ရေးပြီးရင် UI ကို ထပ်ဆောက်။ ပြီးမှ create post တင်လိုက်ရင် database ထဲမှာ ဝင်မဝင် ဆိုပြီး သမာရိုးကျ စစ်ခဲ့ပါတယ်။ Test Driven Development မှာ ဆိုရင် Create Post function ရေးပြီးရင် အဲဒီ အတွက် Unit Test တစ်ခု ရေးပါတယ်။ Deleate အတွက် ရေးပြီးရင် Unit Test တစ်ခု ထပ်ရေးပါတယ်။ Unit Test ဟာ ဖြစ်နိုင်သည့် case တွေ အကုန် ထပ်စစ်ပါတယ်။ ဥပမာ Create Post မှာ success case ရှိမယ်။ validation fail case ရှိမယ်။ duplicate case တွေ ရှိမယ်။ ဖြစ်နိုင်သည့် case တွေကို Unit Test မှာ ထည့်ရေးပြီး စစ်ပါတယ်။\`

```
Feature: Create Blog Post
  Scenario: Successful Post
    Given Post information
    When User has Login
    Then Check Post validation
    When Post has valid
    Then Save in database

  Scenario: Fail User Not Login Post
    Given Post information
    When User has not Login
    Then Show Error

  Scenario: Fail Post not valid
    Given Post information
    When User has Login
    Then Chekc Post Validation
    When Post validation fail
    Then Show Error
```

ဒီလို ဖြစ်နိုင်သည့် case တွေကို Unit Test မှာ စစ်ဆေးတာပါ။ Create Post မှာတင် Unit Test ၃ ခု သို့မဟုတ် ၃ ထက် မက လည်း ပေါ်နိုင်ပါတယ်။

Red/Green/Refactor

Unit Test မှာ အရေးကြီးတာက

1. Red
2. Green
3. Refactor

ဆိုသည့် အပိုင်းပါ။

Red

Red ဆိုသည်မှာ Case တွေကို Red ဖြစ်အောင် အရင် ဖန်တီးရတာပါ။ Code မရေးခင်မှာ ဖြစ်နိုင်သည့် function တွေကို ကြိုစစ်ထားပါတယ်။ function က implement မလုပ်ထားသေးသည့် အတွက် fail ဖြစ်မှာပဲ။

Action: ပါဝင်မယ့် fature တွေကို ကြိုပြီး test တွေ ရေးထားခြင်းပါ။

OutCome: test ကို run သည့် အခါမှာ fail ဖြစ်ပါမယ်။ Red indicator နဲ့ အကုန်ပြနေပါမယ်။ ဒါကြောင့် “Red” လို့ ခေါ်ပါတယ်။

Green

Green ကတော့ ဒုတိယ် အဆင့်ပါ။ ခုနက ရေးထားသည့် function တွေကို pass ဖြစ်အောင် ရေးသားသည့် အပိုင်းပါ။ အဓိကတော့ ရေးထားသည့် function တွေကို implement လုပ်ပြီး pass ဖြစ်အောင် လုပ်ခြင်းပါ။

Action: Test မှာ ရေးထားသားသည့် fail test တွေကို pass ဖြစ်အောင် ရေးသားရပါမည်။

Outcome: Test run ပြီးသည့် အခါမှာတော့ အကုန် pass ဖြစ်ပြီး green ဖြစ်ဖို့ လိုပါတယ်။

Refactor

Test တွေက pass ဖြစ်ပြီ ဆိုရင် Refactor ပြန်လုပ်ဖို့ လိုပါတယ်။ Code ကို clean လုပ်ဖို့ လိုပါတယ်။ Clean လုပ်သည့် အခါမှာ လက်ရှိရှိနေသည့် behavior ကို မပြောင်းလဲ ပဲ refactor လုပ်ရပါမယ်။ Code duplication ဆို remove လုပ်တာ readability ကောင်းအောင် ပြင်တာ design pattern တွေ apply လုပ်တာတွေ ပါဝင်ပါတယ်။ ပြီးသည့် အခါ code ပြန် test လုပ်သည့် အခါ pass ဖြစ်နေဖို့ လိုပါတယ်။

Action: code ကို optimize လုပ်ရန် ၊ clean ဖြစ်ပြီး maintainable code တွေ ဖြစ်အောင် ပြန် လုပ်ဖို့ လိုပါတယ်။

Outcome: Test က pass ဖြစ်ဖို့ လိုပါတယ်။ code ကတော့ ပိုပြီး ကောင်းမွန်သည့် code ဖြစ်လာဖို့ လိုပါတယ်။

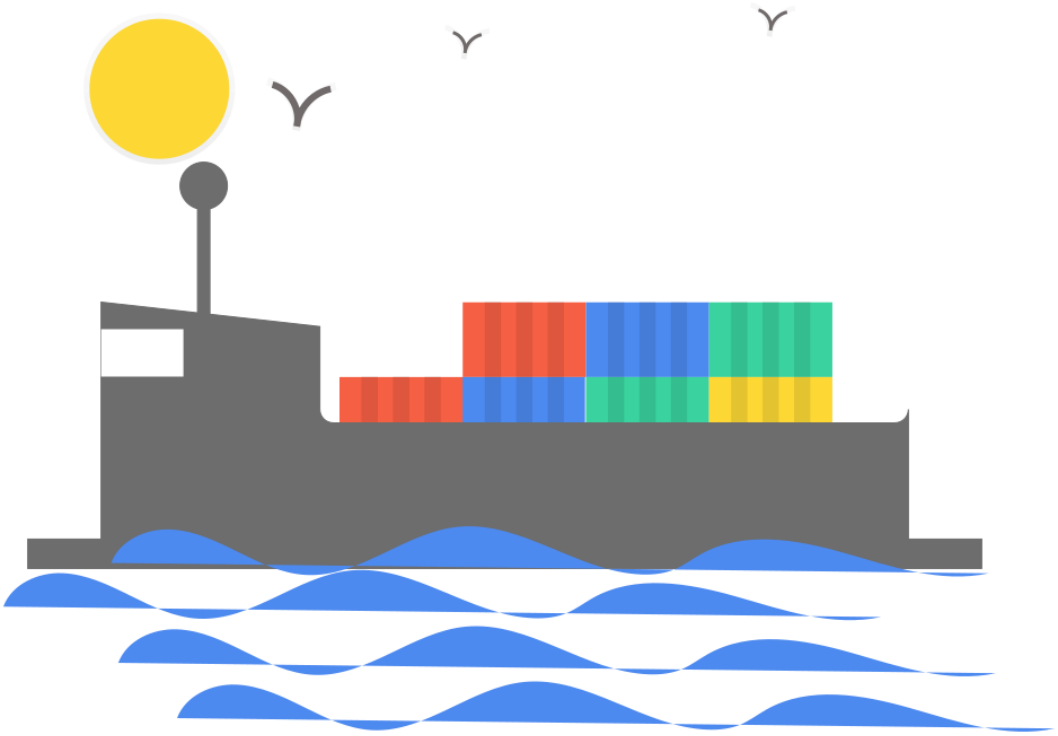
ဘာလို့ Unit Test ရေးရတာလဲ

Unit Test ဟာ အကောင်းဆုံး document တစ်ခုပါ။ Project ဟာ documentation မရေးထားပေမယ့် Test Case တွေ Test Module တွေ ကို သေချာခွဲထုတ်ထားသည့် အခါမှာ ဘယ် function တွေက ဘာလုပ်တယ်။ ဘယ်လိုမျိုး use case တွေ ရှိတယ် ဆိုတာကို တွေ့နိုင်ပါတယ်။ Test Case တွေ ရေးသားထားခြင်းဖြင့် ကိုယ့် ရဲ့ code ကို ပိုပြီး ယုံကြည်မှု ရှိစေပါတယ်။ ဒါပေမယ့် တခြား တစ်ယောက်ယောက်က code ပြင်လိုက်လို့ ကိုယ်ရေးထားသည့် အပိုင်းတွေ ထိသွားမထိသွား သလားဆိုတာကို သိရဖို့ Test Case ကို ပြန် run လိုက်ရုံပါပဲ။ ဒါကြောင့် team တွေနဲ့ ရေးသားသည့် အခါမှာ collaboration ပိုမို ကောင်းမွန် စေပါတယ်။

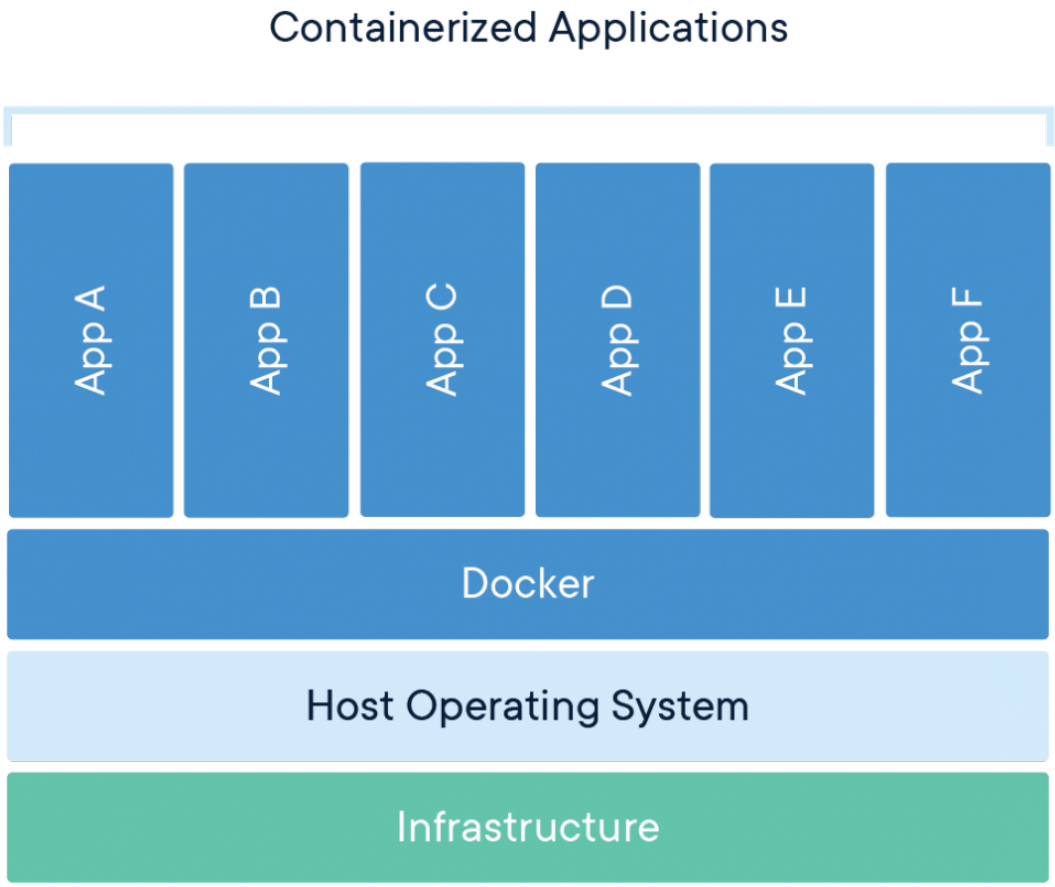
Testable Code

Test Driven Development ကို အသုံးပြုပြီး ရေးမှသာ code တွေကို testable ဖြစ်အောင် ရေးဖြစ်မှာပါ။ Project အစ အဆုံးပြီးမှ code ကို testable ဖြစ်အောင် ရေးသားဖို့ မလွယ်ပါဘူး။ TDD ကြောင့် testable ရေးသားရသာကြောင့် class တွေဟာ lose coupling ဖြစ်စေပါတယ်။ Dependency Injection ကိုလည်း မဖြစ်မနေ အသုံးပြုလာရပါတယ်။ TDD ကြောင့် code တွေကို SOLID principle ကို လိုက်နာအောင် သတိထားပြီး ရေးသားလာနိုင်ပါမယ်။

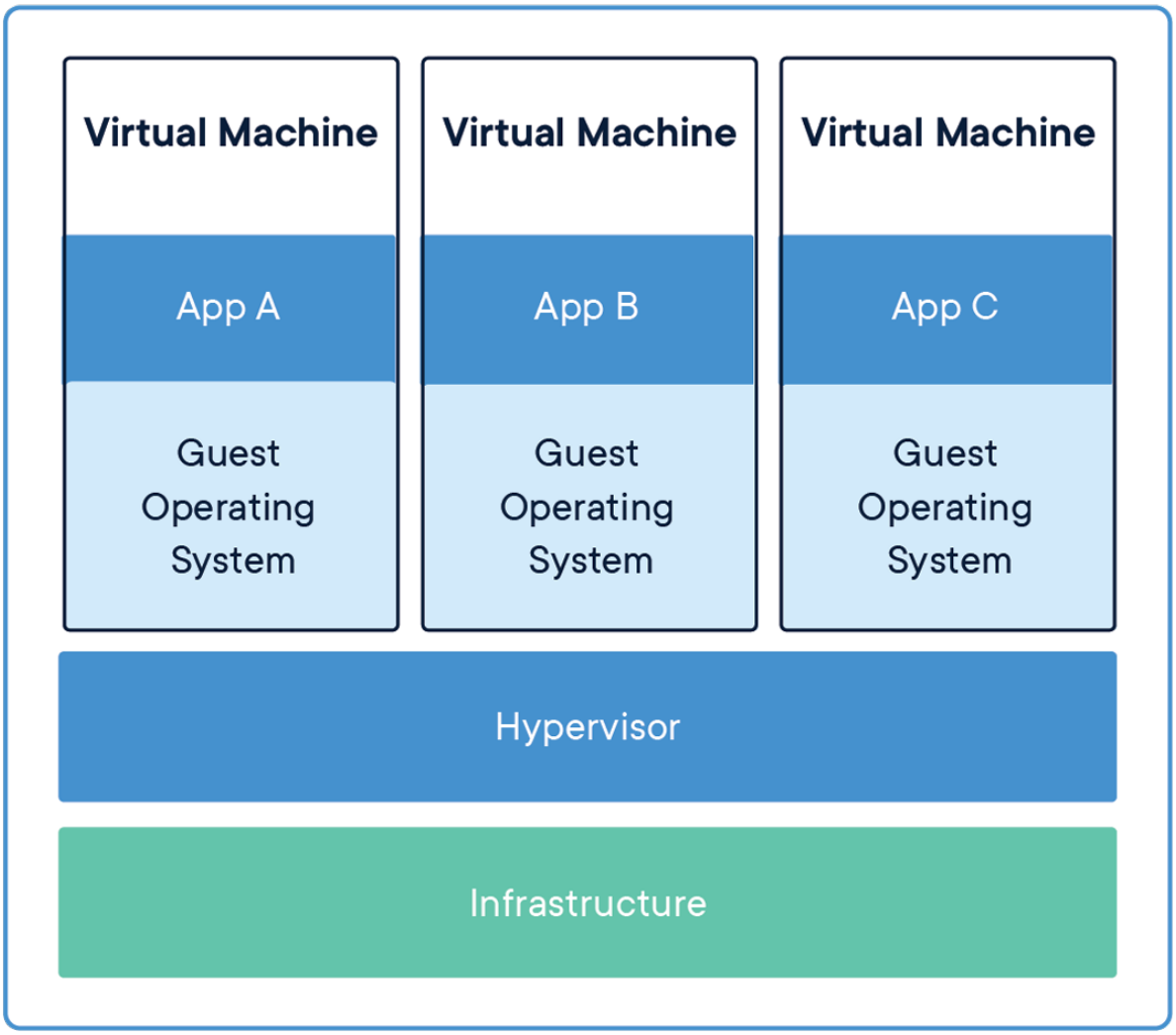
အခန်း ၁၇ :: Docker



အခုနောက်ပိုင်းမှာ developer တွေကို docker ကို အသုံးပြုလာကြပါတယ်။ အထူးသဖြင့် microservices တွေမှာ docker က အရေးပါလာပါတယ်။ server တစ်လုံးထဲ မတူညီ သည့် service တွေ run ဖို့ အတွက် မလွယ်ကူလှပါဘူး။



Docker ဟာ မတူညီသည့် OS နဲ့ services တွေကို လွယ်လင့်တကူ deplopy လုပ်ဖို့ အသင့်တော်ဆုံးပါပဲ။ အထူးသဖြင့် microservices တွေမှာ services တစ်ခုခြင်းဆီကို Container တည်ဆောက်ပြီး docker နဲ့ run ကြပါတယ်။



Virtual Machine နဲ့ run လို့ မရဘူးလား ဆိုတော့ ရပါတယ်။ သို့ပေမယ့် CPU , Memory တွေကို ခွဲပြီး ပေးထားပြီးတော့ အလုပ်လုပ်ပါတယ်။ storage space အစား ခွဲဝေပေးထားသည့် အတွက်ကြောင့် VM အတွက် server တစ်လုံးမှာ လိုအပ်ချက်တွေ အများကြီးရှိပါတယ်။

Container

ကျွန်တော်တို့တွေ အမြဲတန်း ကြုံနေရသည့် ပြဿနာက development စက်မှာ အလုပ်လုပ်တယ်။ production server လည်း ရောက်ရော အလုပ်မလုပ်တော့တာတွေ ကြုံခဲ့ဖူးမှာပါ။ Ubuntu မှာ ရေးထားပေမယ့် တကယ် deploy လုပ်ရသည့် OS ကတော့ CentOS ဖြစ်နေသည့် အခါမှာ လိုချင်သည့် package တွေ မရတာတွေ ဖြစ်ပါလိမ့်မယ်။ တစ်ခါတစ်လေ ubuntu version မတူလို့ package တွေ မတူတာတွေ ရှိတတ်ပါတယ်။

ဒါကြောင့် အခုနောက်ပိုင်းမှာ Docker Image ကို ပြောင်းပြီး အသုံးပြုပါတယ်။ Container ဆိုတာကတော့ Docker Image ကို docker platform ပေါ်မှာ run လို့ ရအောင် ပြောင်းလဲလိုက်တာပါ။ Docker Container မှာ ကျွန်တော်တို့ လိုချင်သည့် လုပ်ဆောင်မှုတွေ ဖြစ်သည့် OS တွေ package တွေ အကုန်ပါဝင်ပါတယ်။ လက်ရှိ OS ကို မထိခိုက်ပဲ သီးသန့် host အနေနဲ့ တည်ရှိနေပါတယ်။ ဒါကြောင့် microservices တွေ အတွက် container ကို အသုံးပြုပြီး server တစ်ခုတည်းမှာလည်း အသုံးပြုနိုင်ပါတယ်။

Install Docker

https://docs.docker.com/engine/install/ubuntu/ မှာ docker သွင်းဖို့ ကို လေ့လာနိုင်ပါတယ်။ Docker ကို Windows/Mac/Linux တို့မှာ သွင်းပြီး အသုံးပြုနိုင်ပါတယ်။

Example

```
docker pull ubuntu
```

ဆိုရင် ubuntu docker image ကို ဆွဲချလိုက်တာပါ။ စက်ထဲမှာ ubuntu image ပဲ ရှိပါသေးတယ်။ container မဖြစ်သေးပါဘူး။

```
docker images
```

အဲဒါဆိုရင်တော့ စက်ထဲမှာ ရှိသည့် images တွေ ဖော်ပြပေးမှာပါ။

Image ကို run ချင်ရင်တော့

```
docker run -dit -v /Users/username/shared:/var/www/sample -p 9091:80 -e LANG=C.UTF-8 -e
LC_ALL=C.UTF-8 ubuntu
```

ဒါဆိုရင် ubuntu image ကို container တစ်ခု ဆောက်လိုက်ပါပြီ။

```
-d | docker container ကို နောက်ကွယ်မှာ run ပြီး container ID ထုတ်ပြရန်
-i | Keep STDIN open even if not attached
-t | Allocate a pseudo-TTY
```

လက်ရှိ စက်က file ကို docker မှာ attach တွဲချင်သည့် အခါမှာ -v ကို သုံးပါတယ်။

```
-v [host file path]:[docker file path]
```

```
-v 9091:80
```

ဆိုရင် docker ထဲက port 80 ကနေ port localhost 9091 ကို ပြန် forward လုပ်ထားတာပါ။

docker port ကို လက်ရှိ os က port ကို forward လုပ်ချင်သည့် အခါမှာ -p ကို အသုံးပြုပါတယ်။

\-e ကတော့ environment ပါ။

```
-e LANG=C.UTF-8 -e LC_ALL=C.UTF-8
```

အဲဒါ မပါလာခဲ့ရင် တစ်ခါတစ်လေ encoding issue တွေ ဖြစ်တတ်လို့ ထည့်ထားတာပါ။

```
docker ps
```

ဆိုရင် run ထားသည့် docker container ကို တွေ့ရမှာပါ။

```
➜  shared docker run -dit -v /Users/htainlinshwe/shared:/var/www/sample -p 9091:80 -e LANG=C.UTF-8 -e LC_ALL=C.UTF-8 ubuntu
74fc52ec5c5b9dd1c9ed60f7f29cb2585eae92df514f9e87e928eecf420e809c
➜  shared docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED         STATUS         PORTS                   NAMES
74fc52ec5c5b   ubuntu    "bash"    8 seconds ago   Up 7 seconds   0.0.0.0:9091->80/tcp    sleepy_tu
```

Shell

`docker ps` နဲ့ ဆိုရင် container ID ရလာပြီ။ အဲဒါဆိုရင် shell ထဲ ဝင်လို့ရပါပြီ။

```
docker exec -it [containerID] /bin/bash
```

Container မှ image သို့

docker container တစ်ခုထဲ shell script နဲ့ ဝင်ပြီး လိုအပ်သည့် software များ ထည့်သွင်းပြီးပြီ ဆိုရင် ကျွန်တော်တို့ container ကို image ပြောင်းပါမယ်။

```
docker ps
```

နဲ့ container ID ကို အရင်ကြည့်ပါ။ ပြီးရင်

```
docker commit [containerID]
```

```
→  ~ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED         STATUS          PORTS                    NAMES
74fc52ec5c5b   ubuntu    "bash"    22 hours ago    Up 12 minutes   0.0.0.0:9091->80/tcp     sleepy_tu
→  ~ docker commit 74fc52ec5c5b
sha256:31ac4941991bea0a9f4e9cd4394f7643abca5f7faa98ab99e7edc6e3383a59ff
→  ~ docker images
REPOSITORY                          TAG       IMAGE ID       CREATED          SIZE
<none>                              <none>    31ac4941991b   3 seconds ago    274MB
```

```
docker images
```

ခေါ်လိုက်သည့် အခါမှာတော့ REPOSITORY Tag နဲ့ image ထွက်လာပါမယ်။

```
docker tag [IMAGE ID] [NEW NAME]
```

```
→  ~ docker tag 31ac4941991b hi_vim
→  ~ docker images
REPOSITORY                          TAG       IMAGE ID       CREATED              SIZE
hi_vim                              latest    31ac4941991b   About a minute ago   274MB
registry.gitlab.com/saturngod/tesan latest    cd3db666e310   4 weeks ago          871MB
postgres                            latest    577410342f45   6 weeks ago          374MB
ubuntu                              latest    ba6acccedd29   2 months ago         72.8MB
```

အခု ပုံမှာတော့ hi_vim လို့ နာမည်ပေးထားလိုက်ပါတယ်။

```
→  ~ docker stop 74fc52ec5c5b
74fc52ec5c5b
→  ~ docker rm 74fc52ec5c5b
74fc52ec5c5b
→  ~
```

```
docker stop [Container ID]
```

လက်ရှိ run နေသည့် container ကို ရပ်ဖို့ပါ။

```
docker rm [Container ID]
```

လက်ရှိ run ထားသည့် container ကို ဖျက်လိုက်ပါပြီ။

```
→  ~ docker run -dit -v /Users/htainlinshwe/shared:/var/www/sample -p 9091:80 -e LANG=C.UTF-8 -e LC_ALL=C.UTF-8 hi_vim
ce9c4ace33b88b365fb8319665a0a2c3bb31ef52c8b6fb1374c9e3a3623ebc0c
→  ~ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED          STATUS          PORTS                    NAMES
ce9c4ace33b8   hi_vim    "bash"    4 seconds ago    Up 3 seconds    0.0.0.0:9091->80/tcp     ecstatic_jemison
→  ~
```

```
docker run -dit -v /Users/username/shared:/var/www/sample -p 9091:80 -e LANG=C.UTF-8 -e
LC_ALL=C.UTF-8 hi_vim
```

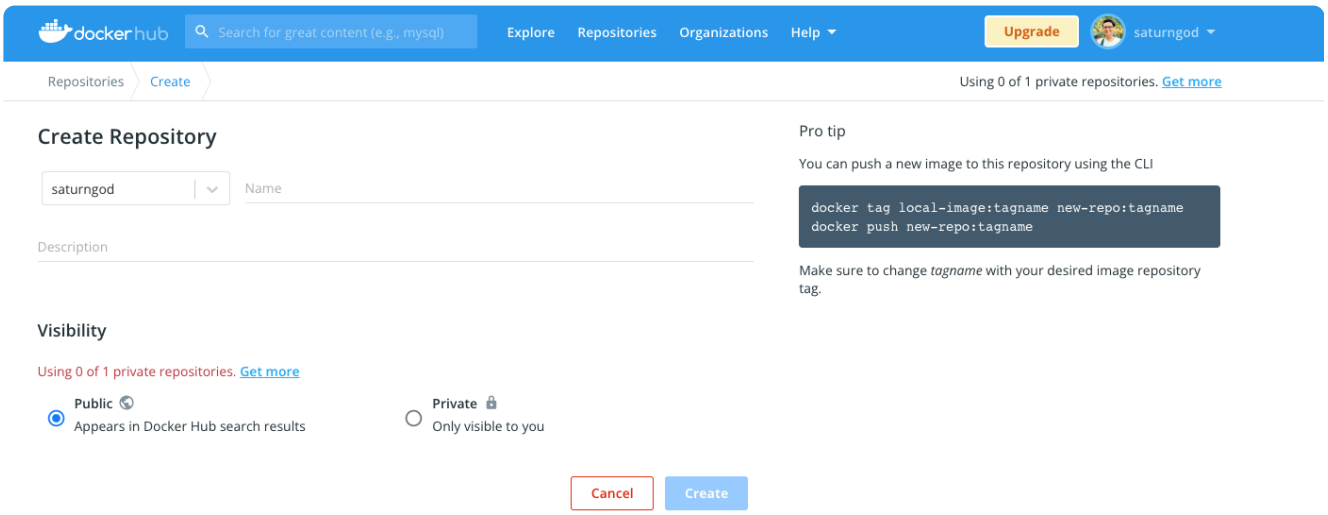
docker run တာကတော့ အရင်အတိုင်းပါပဲ။ ပြောင်းလဲ သွားတာကတော့ image name ပါ။ အခု image `hi_vim` နဲ့ run ထားလိုက်ပါတယ်။

```
→  ~ docker exec -it ce9c4ace33b8 /bin/bash
root@ce9c4ace33b8:/#
```

ပြီးရင် `docker ps` ကနေ container ID ထုတ်ပါ။ `docker exec -it [container id] /bin/bash` နဲ့ ဝင်ပြီး သွင်းထားသည့် software တွေ package တွေ ရှိရဲ့လား စစ်ကြည့်ပါ။

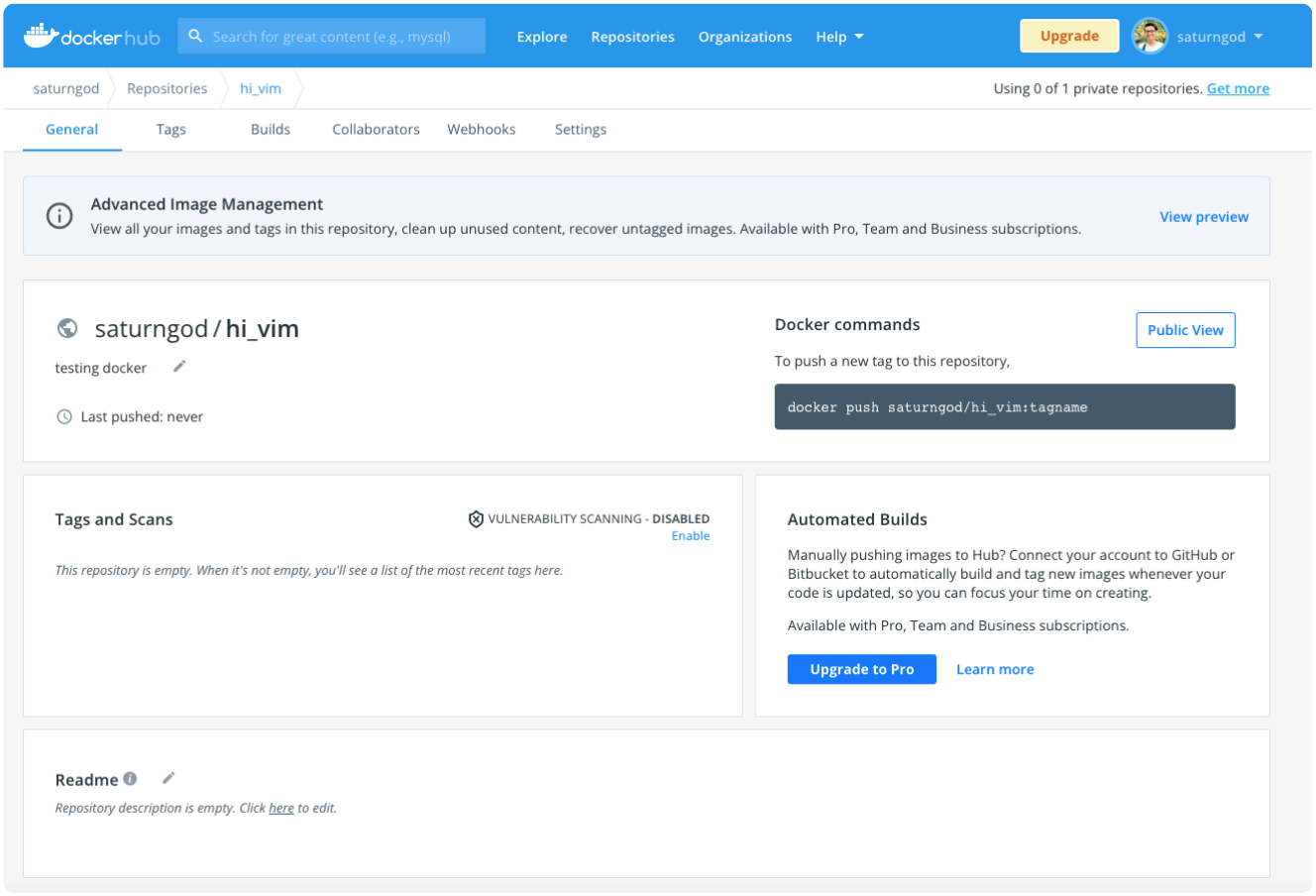
Publish to Docker Hub

Image ကို လူတိုင်း သုံးနိုင်အောင် publish လုပ်ထားလို့ရပါတယ်။ နောက်ပြီး server ကနေ ဆွဲချလို့ရအောင် publish အရင် လုပ်လို့ရပါတယ်။ Docker hub မသုံးချင်ပဲ private repo အတွက် ဆိုရင် gitlab က repo ကို အသုံးပြုနိုင်ပါတယ်။



https://hub.docker.com မှာ login ဝင်ပြီး Create Repository လုပ်ပါ။

Repo name ထည့်ပြီး Create လုပ်လိုက်ပါ။



အခု ကျွန်တော်က `saturngod/hi_vim` repo ကို create လုပ်ထားပါတယ်။

ပြီးလျှင်

```
docker login
```

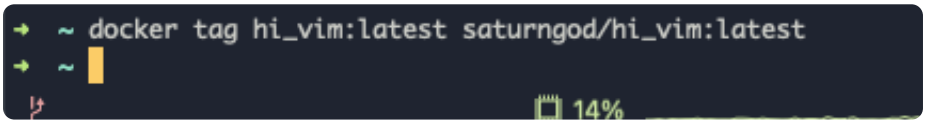
ဖြင့် docker hub ကို command line ကနေ login အရင် ဝင်ထားဖို့ လိုပါတယ်။ သတိထားရမှာက password အစား access token နဲ့ ဝင်ဖို့ recommand လုပ်ပါတယ်။ Access Token ကို ဘယ်လိုရနိုင်သလဲ ကို

https://docs.docker.com/go/access-tokens/ မှာ ရေးထားပေးပါတယ်။

အကောင့် ပြောင်းချင်ရင်တော့

```
docker logout
docker login
```

နဲ့ ပြောင်းနိုင်ပါတယ်။



```
docker tag [local image]:[Tag] [Repo]:[Tag]
```

အခု hi_vim ကို saturngod/hi_vim repo tag ပေါင်းထည့်လိုက်ပါပြီ။

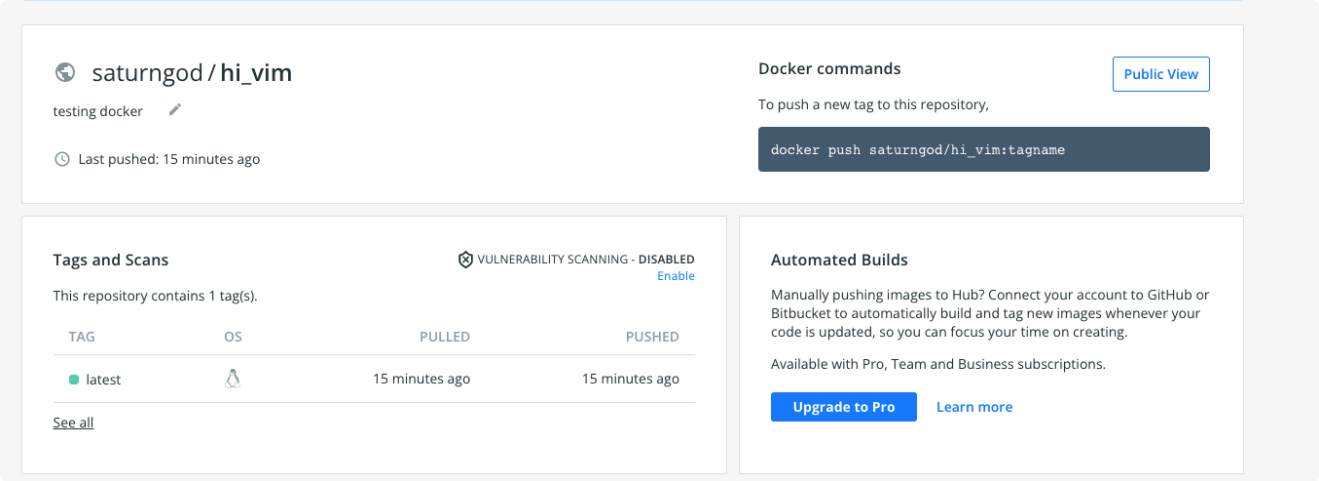
```
➜  ~ docker tag hi_vim:latest saturngod/hi_vim:latest
➜  ~ docker images
REPOSITORY                    TAG       IMAGE ID       CREATED          SIZE
hi_vim                        latest    31ac4941991b   22 minutes ago   274MB
saturngod/hi_vim              latest    31ac4941991b   22 minutes ago   274MB
```

docker images မှာလည်း saturngod/hi_vim ကို မြင်ရပါလိမ့်မယ်။

```
➜  ~ docker push saturngod/hi_vim:latest
The push refers to repository [docker.io/saturngod/hi_vim]
822cfcece7b7: Pushed
9f54eef41275: Mounted from library/ubuntu
latest: digest: sha256:49bcc06fbdfb83271c2ee479661c8b6c01fefd0d475f565af9e812e19bb2a021 size: 741
```

```
docker push saturngod/hi_vim:latest
```

ဒါကတော့ ကျွန်တော့် image ကို docker hub ပေါ် push တင်လိုက်တာပါ။



Docker hub website မှာ image တက်လာတာ တွေ့ရပါလိမ့်မယ်။

ဒါဆိုရင် ဘယ်သူမဆို

```
docker pull saturngod/hi_vim
```

ဆိုပြီး ကျွန်တော် ဖန်တီးထားသည့် docker image ကို ယူသုံးလို့ရပါပြီ။

Docker File

ပုံမှန် အားဖြင့် project တစ်ခုလုံးကို depoly လုပ်ချင်သည့် အခါမှာ ကျွန်တော်တို့တွေ Docker File ကို အသုံးပြုပါတယ်။ ကျွန်တော်တို့ project ရှိသည့်နေရာမှာ Dockerfile ဆိုပြီး empty file တစ်ခု ဖန်တီးထားပါ။

```
→  store ls -a
.               .editorconfig   .gitattributes Dockerfile      artisan         composer.lock  package.json   resources       storage         webpack.mix.js
..              .env            .gitignore     README.md       bootstrap       config         phpunit.xml    routes          tests
.docker         .env.example    .styleci.yml   app             composer.json   database       public         server.php      vendor
→  store
```

Dockerfile ထဲမှာ

```
FROM ubuntu:latest

WORKDIR /var/www/html

RUN apt update

RUN apt install -y tzdata
ENV TZ Asia/Yangon

RUN apt install -y software-properties-common
RUN add-apt-repository ppa:ondrej/php
RUN apt update

RUN apt install -y \
        apache2 \
        zip \
        curl \
        vim \
        unzip \
        git

RUN apt install -y php8.0 libapache2-mod-php8.0
RUN apt install -y php8.0-mysql php8.0-cli php8.0-gd php8.0-curl php8.0-xml php8.0-
mbstring

RUN curl -sS https://getcomposer.org/installer | php -- --install-dir=/usr/local/bin --
filename=composer

COPY . /var/www/html/
COPY .docker/production.env /var/www/html/.env
COPY .docker/vhost.conf /etc/apache2/sites-available/000-default.conf

RUN cd /var/www/html/ && composer install

RUN chown -R www-data:www-data /var/www/html && a2enmod rewrite

RUN service apache2 restart

EXPOSE 80 443

CMD ["apachectl", "-D", "FOREGROUND"]
```

Ubuntu image ကို ယူထားပြီး ubuntu မှာ laravel run ဖို့ လိုသည့် အဆင့်အတိုင်း ထည့်ထားပါတယ်။ Port 80 နဲ့ 443 ပဲ access လုပ်ခွင့်ပေးထားပါတယ်။ WORKDIR ကတော့ `docker exec` ကနေ run ရင် ရောက်နေမယ့် directory ပါ။ ဥပမာ `docker exec -it <containerID> ls` ဆိုရင် directory list ပြလာပါမယ်။

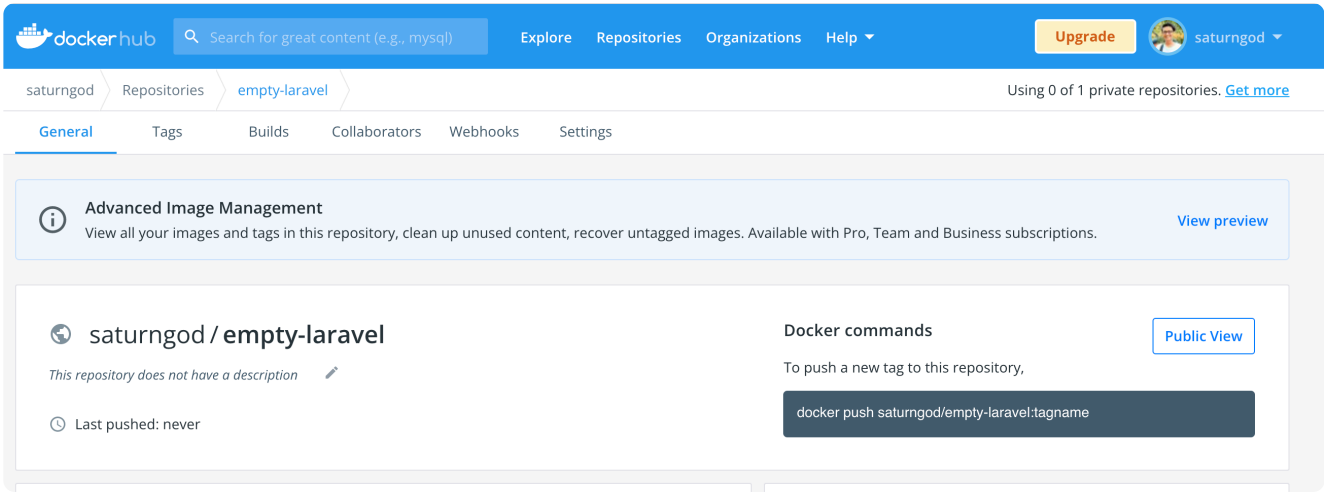
```
→  store docker exec -it store-laravel-app ls
Dockerfile  composer.json       index.html    routes      webpack.mix.js
README.md   composer.lock       package.json  server.php
app         config              phpunit.xml   storage
artisan     database            public        tests
bootstrap   docker-compose.yml  resources     vendor
→  store
```

`.docker/vhost.conf` က

```
<VirtualHost *:80>
    DocumentRoot /var/www/html/public

    <Directory "/var/www/html/public">
        AllowOverride all
        Require all granted
    </Directory>

</VirtualHost>
```

production.env ကတော့ laravel က .env file ပါ။ production အတွက် config တွေ ထည့်ထားတာပါ။

အခု DockerFile ရပြီ ဆိုရင် docker image ဆောက်လို့ရပါပြီ။ Docker Hub မှာ repo တစ်ခု ဆောက်ပါ။ ပြီးရင်




```
docker build -t saturngod/empty-laravel:1.0 .
```

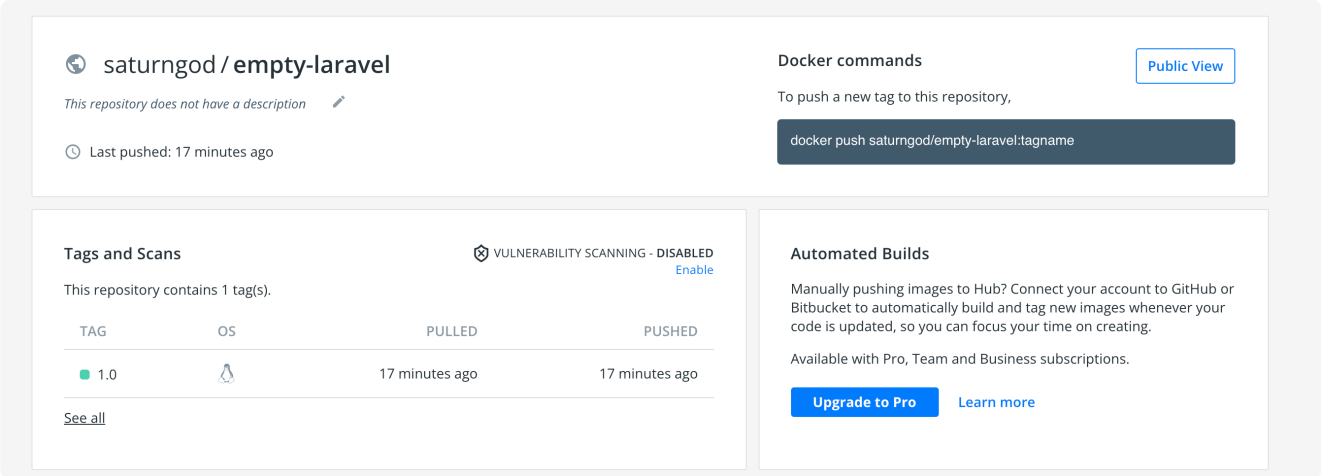
```
→  store docker build -t saturngod/empty-laravel:1.0 .
[+] Building 16.2s (22/22) FINISHED
 => [internal] load build definition from Dockerfile                                                0.0s
 => => transferring dockerfile: 889B                                                                0.0s
 => [internal] load .dockerignore                                                                   0.0s
 => => transferring context: 2B                                                                     0.0s
 => [internal] load metadata for docker.io/library/ubuntu:latest                                    0.0s
 => [internal] load build context                                                                   1.9s
 => => transferring context: 37.42MB                                                                1.8s
 => [ 1/17] FROM docker.io/library/ubuntu:latest                                                    0.0s
 => CACHED [ 2/17] WORKDIR /var/www/html                                                            0.0s
 => CACHED [ 3/17] RUN apt update                                                                   0.0s
 => CACHED [ 4/17] RUN apt install -y tzdata                                                        0.0s
 => CACHED [ 5/17] RUN apt install -y software-properties-common                                    0.0s
 => CACHED [ 6/17] RUN add-apt-repository ppa:ondrej/php                                            0.0s
 => CACHED [ 7/17] RUN apt update                                                                   0.0s
 => CACHED [ 8/17] RUN apt install -y  apache2  zip  curl  unzip  git                               0.0s
```

ကျွန်တော်က empty-laravel ဆိုပြီး repo ဆောက်ထားပါတယ်။ အခု version 1.0 ကို တင်ပါမယ်။

```
docker push saturngod/empty-laravel:1.0
```

```
→  store docker push saturngod/empty-laravel:1.0
The push refers to repository [docker.io/saturngod/empty-laravel]
f4e7423aa540: Pushing [==================================================>]  6.656kB
642fb36e78d2: Pushing [==================================================>]  38.73MB
46925c052240: Pushing [==================================================>]   1.55MB
2f70e5f2311f: Pushing [==================================================>]  3.584kB
741e1ca4f842: Pushing [==================================================>]  4.096kB
2b1787df603f: Waiting
e573681614ad: Waiting
99e8aee85b53: Waiting
01070993db9d: Waiting
eaa87bfd29cb: Waiting
ddcb0df2743e: Waiting
c6d2639776b4: Waiting
87486a46955b: Waiting
```

ဒါဆိုရင် laravel အတွက် docker image တစ်ခု ရသွားပါပြီ။ proudction မှာ image ကို pull ဆွဲပြီး အသုံးပြုရုံပါပဲ။



Docker Compose

Docker compose ဟာ docker file ကို testing လုပ်တာ ဖြစ်ဖြစ် docker run ကို ခဏ ခဏ ပြန်ရေးနေမယ့် အစား `docker-compose.yml` ဖြင့် လိုချင်သည့် docker တွေ ကို တစ်ခါတည်း တည်ဆောက်နိုင်ပါတယ်။

အခု `docker-compose.yml` ဖန်တီးပါမယ်။

```
➜  store touch docker-compose.yml
➜  store ls -a
.                .env               .styleci.yml      artisan           config            phpunit.xml       server.php        webpack.mix.js
..               .env.example       Dockerfile        bootstrap         database          public            storage
.docker          .gitattributes     README.md         composer.json     docker-compose.yml resources        tests
.editorconfig    .gitignore         app               composer.lock     package.json      routes            vendor
➜  store
```

```
version: '3.8'

services:
    app:
        build:
            context: .
            dockerfile: Dockerfile
        image: 'store.dev/laravel'
        container_name: store-laravel-app
        ports:
            - "9980:80"
        networks:
            - laravel-store

networks:
    laravel-store:
        driver: bridge
```

docker compose version 3.8 standard ကို လိုက်နာထားပြီးတော့ docker image ကို `Dockerfile` ကို သုံးထားတယ်။ image name ကို `store.dev/laravel` ပေးထားတယ်။ container_name ကို `store-laravel-app` ပေးထားပါတယ်။ Docker Container က Port 80 ကို port 9980 ကနေ access လုပ်ခွင့်ပေးထားတယ်။ network ကို bridge သုံးထားပါတယ်။

Dockerfile ကို မသုံးပဲ image ကိုလည်း အသုံးပြုနိုင်ပါတယ်။

```
version: '3.8'

services:
    app:
        build:
            context: .
            dockerfile: Dockerfile
        image: 'store.dev/laravel'
        container_name: store-laravel-app
        ports:
            - "9980:80"
        networks:
            - laravel-store
        depends_on:
            - mysql
    mysql:
        image: 'mariadb:latest'
        container_name: store-laravel-db
        restart: unless-stopped
```

```
        ports:
            - "3307:3306"
        environment:
            MYSQL_DATABASE: ${DB_DATABASE}
            MYSQL_ROOT_PASSWORD: ${DB_PASSWORD}
            MYSQL_PASSWORD: ${DB_PASSWORD}
            MYSQL_USER: ${DB_USERNAME}
            MYSQL_ALLOW_EMPTY_PASSWORD: 'yes'
        volumes:
            - ./database/dbdata:/var/lib/mysql
        networks:
            - laravel-store

networks:
    laravel-store:
        driver: bridge
```

ဒါဆိုရင် dependency က mysql ဖြစ်သည့်အတွက်ကြောင့် mariadb container ပါ ပါဝင်လာမှာဖြစ်ပါတယ်။

volumes ကတော့ docker နဲ့ ကိုယ့် စက် နဲ့ ချိတ်ထားသည့် သဘောပါ။ mysql နဲ့ app ကို network အတူတူ သုံးထားသည့် အတွက်ကြောင့် internal network ထဲမှာလို ဖြစ်ပါလိမ့်မယ်။ app ကနေ database container ကို လမ်းပြီး ခေါ်လို့ရမှာပါ။

app container ကနေ mysql container ကို port 3306 ပဲ သုံရမှာပါ။

docker run နေသည့် host ကနေ mysql container က mysql ကို သုံးချင်ရင်တော့ port 3307 နဲ့ ချိတ်ရမှာပါ။

`.env` ထဲမှာ

```
DB_CONNECTION=mysql
DB_HOST=mysql
DB_PORT=3306
DB_DATABASE=laravel
DB_USERNAME=root
DB_PASSWORD=
```

လို့ update လုပ်ပေးလိုက်ပါ။ production.env မှာလည်း update လုပ်ပေးဖို့ လိုပါမယ်။

```
docker exec -it store-laravel-app php artisan migrate
```

```
→  store docker exec -it store-laravel-app php artisan migrate
Migration table created successfully.
Migrating: 2014_10_12_000000_create_users_table
Migrated:  2014_10_12_000000_create_users_table (74.11ms)
Migrating: 2014_10_12_100000_create_password_resets_table
Migrated:  2014_10_12_100000_create_password_resets_table (45.97ms)
Migrating: 2019_08_19_000000_create_failed_jobs_table
Migrated:  2019_08_19_000000_create_failed_jobs_table (55.36ms)
Migrating: 2019_12_14_000001_create_personal_access_tokens_table
Migrated:  2019_12_14_000001_create_personal_access_tokens_table (89.92ms)
→  store
```

store-laravel-app ကတော့ container name ပါ။ laravel ရဲ့ database migration code ကို run လိုက်တာပါ။

အခုဆိုရင် docker အကြောင်းအနည်းငယ်သိလောက်ပြီး ဘာလို့ အသုံးပြုလဲ ဆိုတာ သဘောပေါက်လောက်ပါပြီ။

Orchestration

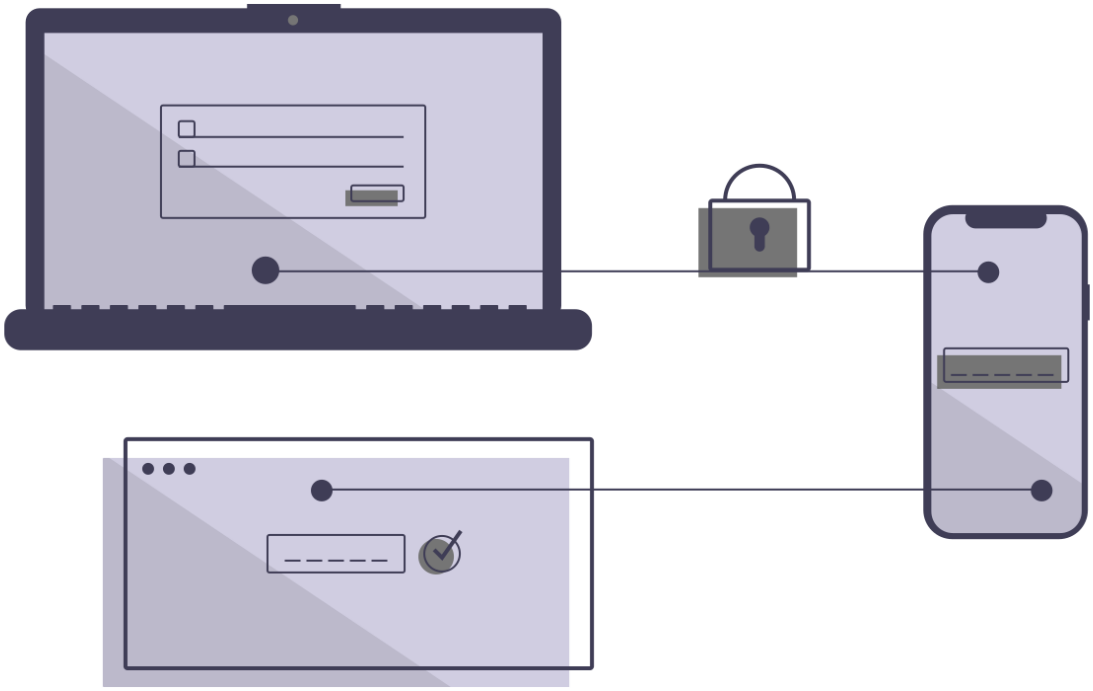
Docker ကို production မှာ run ဖို့ Orchestration platform လိုပါတယ်။ လူသုံးများတာကတော့ Kubernetes ပါ။ K8s လို့လဲ အတိုရေးပါတယ်။ Kubernetes အပြင် Docker Swarm ရှိပါတယ်။

Kubernetes ကို အသုံးပြုဖို့ run ထားသည့် platform ရှိဖို့လိုပါတယ်။ လက်ရှိ လူသုံးများတာတွေကတော့

- Digital Ocean Kubernetes
- Google Kubernetes Engine (GKE)
- Amazon Elastic Kubernetes Service (EKS)
- Azure Kubernetes Service (AKS)

Kubernetes ဟာ Auto Scaling အပြင် zero downtime အတွက်ပါ အဆင်ပြေပါတယ်။ Kubernetes အကြောင်းဖော်ပြဖို့ အတွက် သီးသန့် စာအုပ်တစ်အုပ်စာလောက် ရှိသာဖြစ်လို့ နောက်ပိုင်း လိုအပ်ရင် ကိုယ်တိုင် ဆက်လေ့လာဖို့ လိုပါတယ်။ Kubernetes ဟာ configuration တွေက docker လိုမျိုး အများကြီး လေ့လာဖို့ လိုအပ်သလို production သွားဖို့အတွက် သီးသန့် platform လိုအပ်ပါတယ်။ အခု အခါမှာ Kubernetes ကို ပြည်တွင်းက company တွေ အသုံးပြုမှုကတော့ နည်းပါးပါသေးတယ်။ Digital Ocean မှာ Auto Scaling လိုအပ်ရင်တော့ Kubernetes ကို အသုံးပြုသင့်ပါတယ်။

အခန်း ၁၈ :: Security



Web Developer ဖြစ်စေ mobile app developer ဖြစ်စေ security နှင့် ပတ်သက်ပြီးတော့ အခြေခံတွေကို နားလည် သိထားဖို့ လိုပါတယ်။ Mobile app ဖြစ်ပေမယ့် အသုံးပြုထားသည့် API call တွေဟာ အများအားဖြင့် web app နဲ့ ဖန်တီးထားတာတွေပါ။

SQL Injection

SQL Injection ကို လူတိုင်းကြားဖူးပါလိမ့်မယ်။ Website ဖန်တီးသည့် အခါ ဖြစ်ဖြစ် API ဖန်တီးသည့် အခါဖြစ်ဖြစ် SQL Injection ကို သိရှိထားဖို့ လိုပါတယ်။

```
http://www.example.com/search?q=hello
```

ဆိုသည့် URL က database ထဲမှာ hello ကို ရှာပေးပါသည်။

```
SELECT * from contents where text like "%hello%";
```

ဆိုပြီး SQL code ကို ပြန်ပြောင်းရေးထားသည်။

အကယ်၍သာ

```
http://www.example.com/search?q=";"SELECT * FROM contents";--
```

ဆိုပြီး ပြောင်းလိုက်မယ် ဆိုရင်

```
SELECT * from contents where text like "%";"SELECT * FROM contents";--hello%";
```

ဆိုပြီး ဖြစ်သွားပါမယ်။ -- ကတော့ comment ဖြစ်သည့်အတွက် --hello%"; က အလုပ်လုပ်မှာ မဟုတ်ပါဘူး။ ရှာမယ့် text အစား contents တွေ အကုန်ထွက်လာတာ သို့မဟုတ် နှစ်သက်ရာ SQL ကို နောက်မှာ အစားထိုးပြီး ရေးလို့ရသွားတာကို တွေ့ရပါလိမ့်မယ်။

ပုံမှန် အားဖြင့် PHP နဲ့ database ထဲက data ကို ဆွဲထုတ်မယ်ဆိုရင် အောက်ကလို ရေးကြပါတယ်။

```
$sql = "SELECT first_name, last_name FROM employees WHERE first_name =
'".$_GET['name']."' OR last_name ='" . $_GET['name'] . "' LIMIT 5";

$result = $conn->query($sql);
```

Get Query string က နေ name နဲ့ ရှာထားတာပါ။

```
http://localhost/sqlinjection/index.php?name=Georgi&submit=Search
```

ဆိုပြီး ခေါ်ထားပါတယ်။ Database ထဲက အမည်ရှာပြီး ဆွဲထုတ်ပေးပါတယ်။ သို့ပေမယ့် query ကို အောက်ကလို ပြင်လိုက်မယ် ဆိုရင်

```
http://localhost/sqlinjection/index.php?name='%20OR%201=1%20LIMIT%2010;%20--
%20&submit=Search
```

LIMIT 5 အစား LIMIT 10 ဖြစ်သွားပါလိမ့်မယ်။ URL Decode လုပ်ကြည့်လိုက်ရင် name က

```
' OR 1=1 LIMIT 10; --
```

ဆိုပြီး ထည့် လုပ်ထားတာပါ။

အဲဒီအခါ query က

```
SELECT first_name, last_name FROM employees WHERE first_name = '' OR 1=1 LIMIT 10; --
```

ပုံစံ ဖြစ်သွားပါတယ်။

ဒါကြောင့် query တွေ ကို ရေးသည့် အခါမှာ parameter binding ကို အသုံးပြုနိုင်ပါတယ်။

```
 $sql = "SELECT first_name, last_name FROM employees WHERE first_name = ? OR last_name =
? LIMIT 5";

    $stmt = $conn->prepare($sql);

    $stmt->bind_param("ss",$name,$name);

    $stmt->execute();

    $res = $stmt->get_result();
```

SQL Injection တွေကို ကာကွယ်ဖို့ အတွက် programming language တိုင်းမှာ parameter နှင့် passing လုပ်တာ ဒါမှမဟုတ် framework တွေကို အသုံးပြုတာ ဖြစ်ဖြစ် ကာကွယ်နိုင်ပါတယ်။

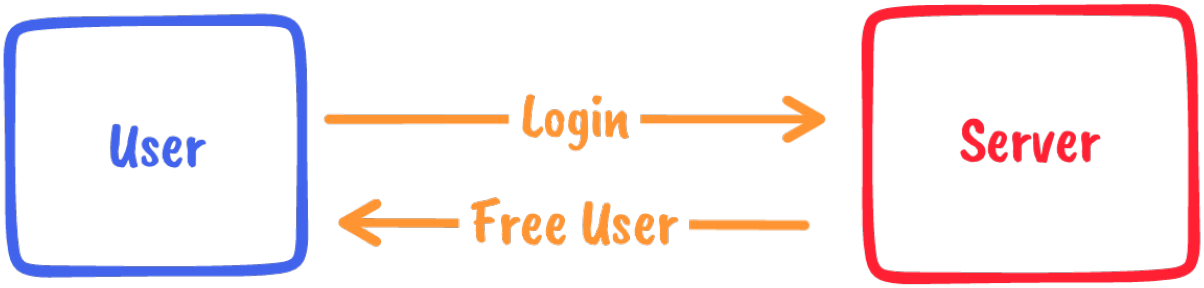
SQL Injection အတွက် sqlmap (http://sqlmap.org) ကို အသုံးပြုပြီး လွယ်လင့်တကူ တိုက်ခိုက်နိုင်ပါတယ်။

```
 python sqlmap.py -u "http://localhost/sqlinjection/index.php?name=Georgi&submit=Search"
--dbs
```

SQL Injection ရှိသည့် website တွေကို SQL Map ကို အသုံးပြုပြီးတော့ website ကိုသာမက system ထဲ အထိတောင် တိုက်ခိုင်နိုင်ပါတယ်။

Man-in-the-middle attack

ပုံမှန် အားဖြင့် Web App, Mobile App တွေမှာ API ဖြင့် သွားလာပါတယ်။



ပုံမှန် သွားနေကြသည့် ပုံစံပါ။

ကြားက တစ်ယောက်က ဝင်ပြီးတော့ server ဘက်က ပို့လိုက်သည့် data ကော server ဘက်က ပြန်လာသည့် data ကို ပြင်လိုက်လို့ရပါတယ်။ Man In Middle attack.ကို

Android OS 10 နောက်ပိုင်းမှာတော့ device ကို root မဖောက်ပဲနှင့် https certificate ကို ပြောင်းလဲလို့မရပါဘူး။ ကြားက data တွေကို ဖတ်လို့ရအောင် ပြင်လို့ရအောင် အတွက် ကြားခံ certificate ကိုပြင်ရပါတယ်။

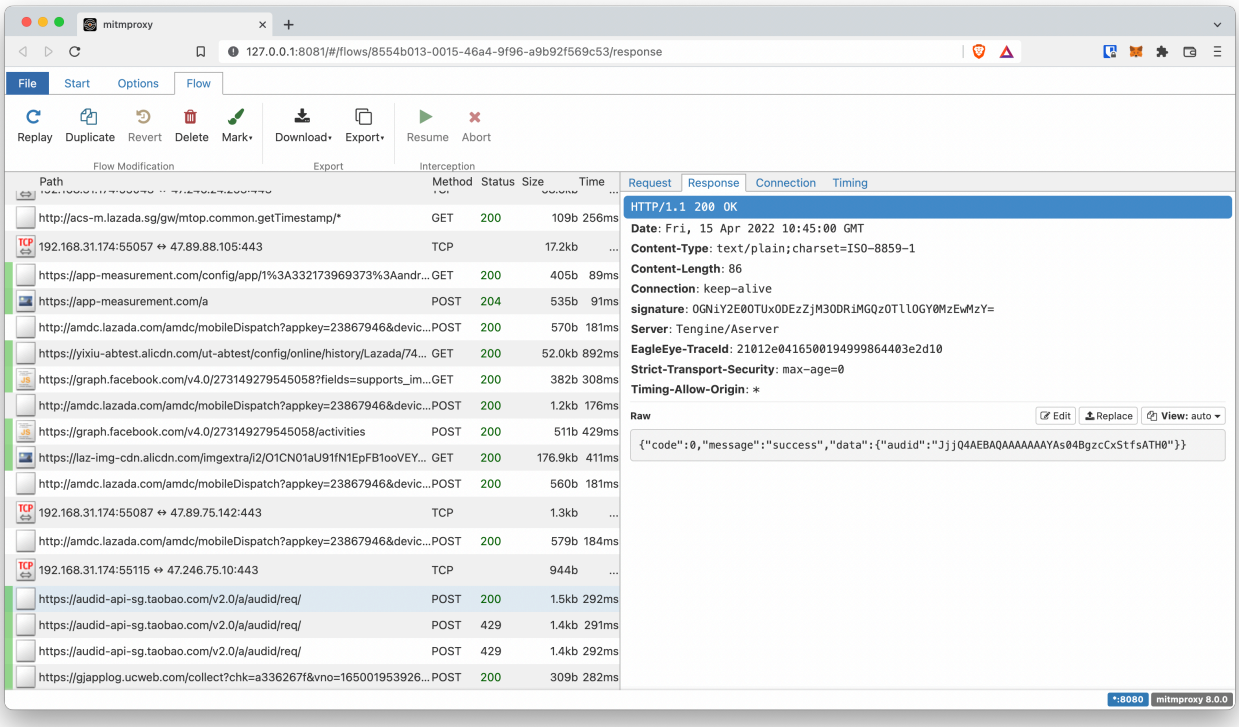
Man In Attack ကို စမ်းသပ်ဖို့ proxy server တွေဖြစ်သည့်

- BurpSuite
- Charles
- Fiddler
- mitmproxy

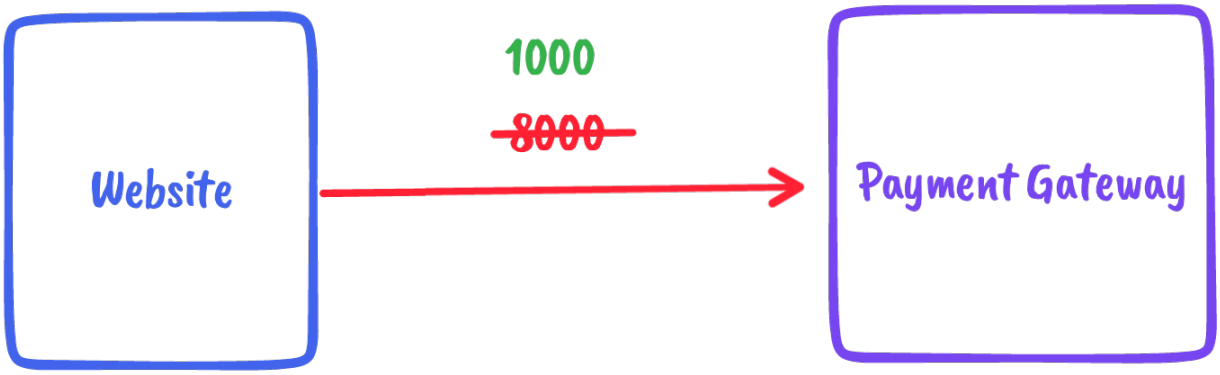
အပြင် နှစ်သက်ရာ proxy နဲ့ စမ်းသပ်နိုင်ပါတယ်။ Mac မှာဆိုရင်တော့ Charles app ကို အသုံးပြုနိုင်ပါတယ်။

Website အတွက် ဆိုရင် https://requestly.io/ ကို recommend ပေးပါတယ်။ ရိုးရှင်းလွယ်ကူသည့် အတွက် စမ်းသပ်ရသာ အဆင်ပြေပါတယ်။

ကျွန်တော်ကတော့ BurpSuite ကို အသုံးပြုပါတယ်။ Proxy တွေ ခံထားလိုက်ခြင်းအားဖြင့် App တွေ website တွေ နောက်က အသုံးပြုနေသည့် Request/Response တွေ အကုန်လုံးကို မြင်ရပါမယ်။

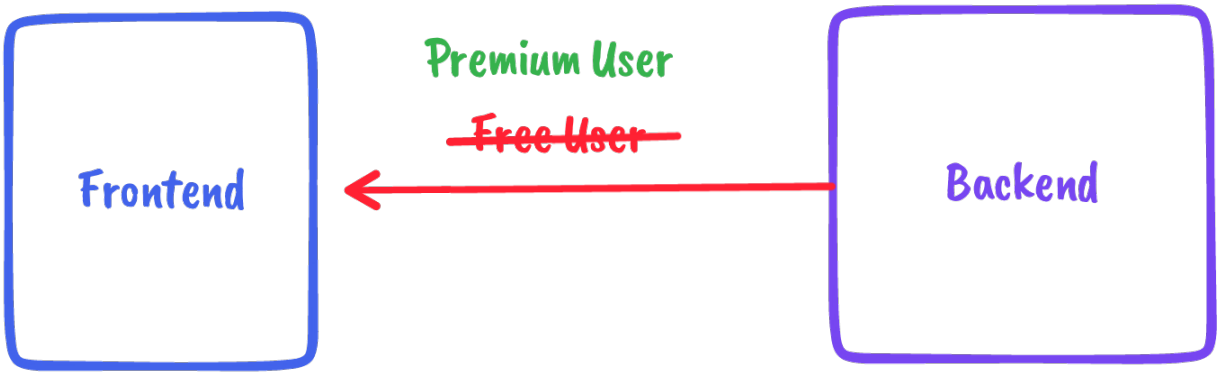


Proxy ခံထားခြင်းအားဖြင့် နှစ်သက်ရာ request ကို ပြန်ပြီး ပြင်နိုင်ပါတယ်။ ပြန်လာသည့် response ကိုလည်း ပြင်နိုင်ပါတယ်။ ဥပမာ ECommerence website သို့မဟုတ် APP ဆိုပါဆို့။ 8,000 MMK ကျသင့်တာကို server side ကို ပို့လိုက်သည့် အချိန်မှာ 1,000 MMK နဲ့ လို့ ပြောင်းပြီး ပို့လိုက်တယ်။ Payment Gateway ဘက်ကို ရောက်သည့် အခါမှာ 1,000 MMK နဲ့ ငွေရှင်းရပြီး success ဖြစ်သွားပါတယ်။



Customer ကို ပစ္စည်းမပေးရင် မင်းတို့ system က အလိုလို ဖြစ်သွားတာ ဆိုပြီး ပြန်ငြင်းနိုင်သည့် အတွက်ကြောင့် မလိုအပ်ပဲ ရှင်းနေရပါတယ်။ ဒါကြောင့် request တွေမှာ hashing နှင့် ဖြစ်နိုင်ရင် server side ဘက်မှာပဲ processing လုပ်ဖို့လိုပါတယ်။

တစ်ခါတစ်လေ ကျွန်တော်တို့တွေဟာ response တွေပေါ်မှာ မူတည်ပြီး စစ်ထားတာတွေ ရှိတတ်ပါတယ်။ ဥပမာ product ကို premium user တွေပဲ access ရမယ်။ Premium user ဟုတ်မဟုတ် ကို user profile response ကနေ ထိန်းထားတယ်။ အဲဒီ အခါမှာ response ကို ပြင်လိုက်ရုံနဲ့ product ကို access လုပ်လို့ရသွားနိုင်ပါတယ်။



ဒါကြောင့် response တွေကို အပြည့် မယုံကြည်ပဲ server side နဲ့ ပြန်ပြီး စစ်ဆေးဖို့ လုိပါတယ်။ ဥပမာ download ချသည့် အခါမှာ link ကို server side က နေ user ကို စစ်ပြီးမှ link ချပေးတာမျိုး ကို ထည့်သွင်း စဉ်းစားထားဖို့ လိုပါတယ်။

Encryption, Encoding, Hashing

Developer တစ်ယောက် အနေနဲ့ မဖြစ်မနေ

- Hashing
- Encoding
- Encryption

ကို သိဖို့ လိုပါတယ်။ ဒါဟာ အရမ်းအခြေခံ ကျပါတယ်။ Junior Developer တွေ အနေနဲ့ အမြဲ ဒီ ၃ ခု ကို မှားယွင်းတတ် ရောထွေးနေတတ်ပါတယ်။

Encryption

Data ကို တစ်စုံတစ်ယောက်က ဖတ်လို့ မရအောင် ပြုလုပ်ချင်သည့် အခါမှာ Encryption ကို အသုံးပြုပါတယ်။ ဥပမာ သော့အိမ် နဲ့ သော့ လိုပါပဲ။ သေတ္တာကို သော့ နဲ့ ခတ်လိုက်တယ်။ သေတ္တာထဲမှာ ဘာရှိလဲ သိဖို့ အတွက် သော့ နဲ့ ပြန်ဖွင့် မှ ရတယ်။ ပုံမှန် အားဖြင့် Encryption ကို AES-128 Method နဲ့ အသုံးပြုကြတာ များပါတယ်။ AES-128 နဲ့ AES-256 မှာ အဓိက key ကွာပါတယ်။ 128 မှာ key က 16 လုံး ရှိရပါမယ်။ Vector လည်း ၁၆ လုံး ရှိဖို့ လိုတယ်။ AES-256 မှာတော့ key က 32 လုံး ရှိရပါမယ်။ Vector က ၁၆ လုံး ရှိရပါမယ်။ 128 , 256 ဆိုတာကိုတော့ bit ပါ။ ဒါကြောင့် key size က 8 နဲ့စားပြီးတော့ ဘယ်လောက် byte လည်း ဆိုတာ တွက် လို့ ရပါတယ်။ AES မှာ CFB နဲ့ CBC Mode ဆိုတာ ထပ်ရှိပါသေးတယ်။ AES ကို ပြန်ဖြည်မယ်ဆိုရင် ဘာ Mode လဲ ? CFB လား CBC လား ။ key က 16 လုံးလား 32 လုံးလား။ Vector ကကော ဘာလဲ စတာတွေ သိဖို့ လိုပါတယ်။ Encryption Method တွေက အများကြီးရှိပါတယ်။

AES ကတော့ Key , Vector သိရင်တော့ encryption က ဖြည်လို့ ရပါတယ်။ တနည်းပြောရင် Dictionary Attack နဲ့ ဖြည်ရင်တော့ ဖြစ်နိုင်ချေရှိပါတယ်။

Encryption မှာ **symmetric** နှင့် **asymmetric** encryption ၂ မျိုး ရှိပါတယ်။

Symmetric

Symmetric ကတော့ encrypt လုပ်ထားသည့် key နှင့် information တွေ အတိုင်း decrypt ပြန်လုပ်ရတယ်။ သော့ အိမ် တစ်ခု နဲ့ သော့ တစ်ချောင်း နဲ့ ပိတ်ပြီး အဲဒီ သော့ နဲ့ ပဲ ပြန်ဖွင့် သလိုပေါ့။ Symmetric မှာ AES, DES, 3DES, RC4 စတာတွေ ပါဝင်ပါတယ်။

Asymmetric

Asymmetric ကတော့ အနည်းငယ် ကွဲပြားပါတယ်။ သူက key pair နဲ့ အလုပ်လုပ်ပါတယ်။ Key Pair ဆိုတာကတော့ private key တစ်ခု public key တစ်ခု ပါသည့် key တစ်စုံပါ။

ဘာနဲ့ တူသလဲ ဆိုတော့ မောင်မောင် နဲ့ အောင်အောင် စာ အလဲအလှယ် လုပ်ကြတယ်။ မောင်မောင် မှာ အောင်အောင် ပေးထားသည့် သော့ဂလောက် ရှိတယ်။ အောင်အောင်မှာ မောင်မောင် ပေးထားသည့် သော့ဂလောက် ရှိတယ်။ မောင်မောင်မှာပဲ သူ့ သော့ဂလောက်ကို ဖြည်ဖို့ သော့ ရှိတယ်။ အောင်အောင်လည်း ထို့အတူပဲ။

မောင်မောင် က စာ ကို သေတ္တာထဲထည့်။ အောင်အောင်ပေးထားသည့် သော့ဂလောက် နဲ့ ပိတ်။ ပြီးတော့ ကြားလူကနေ အောင်အောင် ဆီကို ပို့ ။

အဲဒီ သော့ ကို အောင်အောင် မှာပဲ သော့ ရှိသည့် အတွက် ဖွင့်လို့ရ လိမ့်မယ်။ ပို့လိုက်သည့် မောင်မောင် ကိုယ်တိုင် ပြန်ဖွင့် လို့ မရတော့ဘူး။ အောင်အောင် ဆီ ရောက်လာမှ သူ့မှာ ရှိသည့် သော့ နဲ့ ဖွင့်။ ပြီးရင် မောင်မောင် သော့ဂလောက် နဲ့ ပြန် ပိတ် ပြီး ကြားလူကနေ မောင်မောင် ဆီ ကို စာပြန်ပို့ ။ အဲဒါမှ မောင်မောင် တစ်ယောက်ပဲ သော့ ဂလောက် ကို ဖွင့် ပြီး စာပြန် ဖတ်လို့ ရပါလိမ့်မယ်။

Private/Public Key Pair ဆိုတာ အထက်ပါ အတိုင်းပါပဲ။ ကျွန်တော်တို့က တစ်ဖက် ကို ပို့ချင်ရင် သူ့ဘက်က public key နဲ့ encrypt လုပ်ပြီး ကျွန်တော် တို့ ဆီ ဘက်ကို စာရောက်လာရင် ကျွန်တော် တို့ private key နဲ့ ပြန် decrypt လုပ်ပါတယ်။

Symmetric ထက် ပို လုံခြုံပါတယ်။ private key ကို ရမှ decrypt လုပ်နိုင်မှာ ဖြစ်ပြီး public key ကို ရမှ encrypt လုပ်နိုင်ပါလိမ့်မယ်။

Asymmetric အတွက် အများအားဖြင့် RSA ကို အသုံးပြုကြပါတယ်။ Apple pay မှာ token ကို elliptic curve cryptography (ECC) အသုံးပြုထားတာကိုလည်း တွေ့နိုင်ပါတယ်။

Encoding

Encoding ကတော့ data format တစ်ခု ကနေ နောက် တစ်ခုကို ကူးပြောင်းတာပါ။ မြင်အောင် ပြောရ ရင် byte data ကို base64 ပြောင်းသလိုမျိုးပေါ့။ Encoding ကို ကျွန်တော်တို့တွေ data transfer အဆင်ပြေဖို့ အဓိက အသုံးပြုပါတယ်။ ဥပမာ image upload တင်မယ် ဆိုရင် image ကို base64 ပြောင်းပြီး server ပေါ်ကို API နဲ့ တင်လိုက်တယ်။ server မှာ ရလာသည့် base64 ကို image format ပြန်ပြောင်းပြီး သိမ်းပါတယ်။ ပုံမှန် bytes array ကို API မှာ transfer လုပ်ဖို့ နည်းနည်း ရှုပ်ထွေးပါလိမ့်မယ်။ Encoding နဲ့ အတူ Decoding ကို ပါနားလည် ဖို့ လိုပါတယ်။ Encode ဆိုတာ ကတော့ data တစ်ခု ကနေ နောက်တစ်ခု ပြောင်းလဲ လိုက်တာပါ။ Decode ကတော့ ပြောင်းလဲထားသည့် data ကို နဂို data ပြန်ရအောင် လုပ်သလိုမျိုးပေါ့။

တစ်ခါတစ်လေ image အသေးလေးတွေကို database ထဲမှာ သိမ်းချင်တယ်။ BLOB သိမ်းနိုင်သလို base64 ပြောင်းပြီး text အနေနဲ့လည်း သိမ်းနိုင်ပါတယ်။ နောက်ပြီးတော့ encrypt လုပ်သည့် အခါမှာ ထွက်လာသည့် bytes data အစား base64 enocoding လုပ်ပြီး data ကို တစ်နေရာကနေ တစ်နေရာ transfer လုပ်နိုင်အောင် အသုံးပြုကြပါတယ်။ decrypt မဖြည်ခင်မှာ base64 ကို deocde ပြန်လုပ်ဖို့ လိုပါတယ်။

```
$data = openssl_encrypt("a","AES-128-
CBC","aes1279ksamja89c",OPENSSL_RAW_DATA,"89729acedfa4eafa");
echo base64_encode($data);
```

ဒီ code ကို ကြည့်လိုက်ရင် AES-128-CBC နဲ့ encrypt လုပ်ထားပြီးတော့ base64 encode ပြန်လုပ်ထားပါတယ်။ encrypt result က RAW data ဖြစ်နေသည့် အတွက်ကြောင့် ဘယ် language နဲ့ မဆို အဆင်ပြေအောင် base 64 ပြန်ပြောင်းထားပါတယ်။ PHP က ထွက်သည့် RAW format နဲ့ .NET က ထွက်သည့် bytes format တွေက မတူညီပါဘူး။ ဒါကြောင့် အများအားဖြင့် base64 encoding ကို အသုံးပြုကြပါတယ်။

Hashing

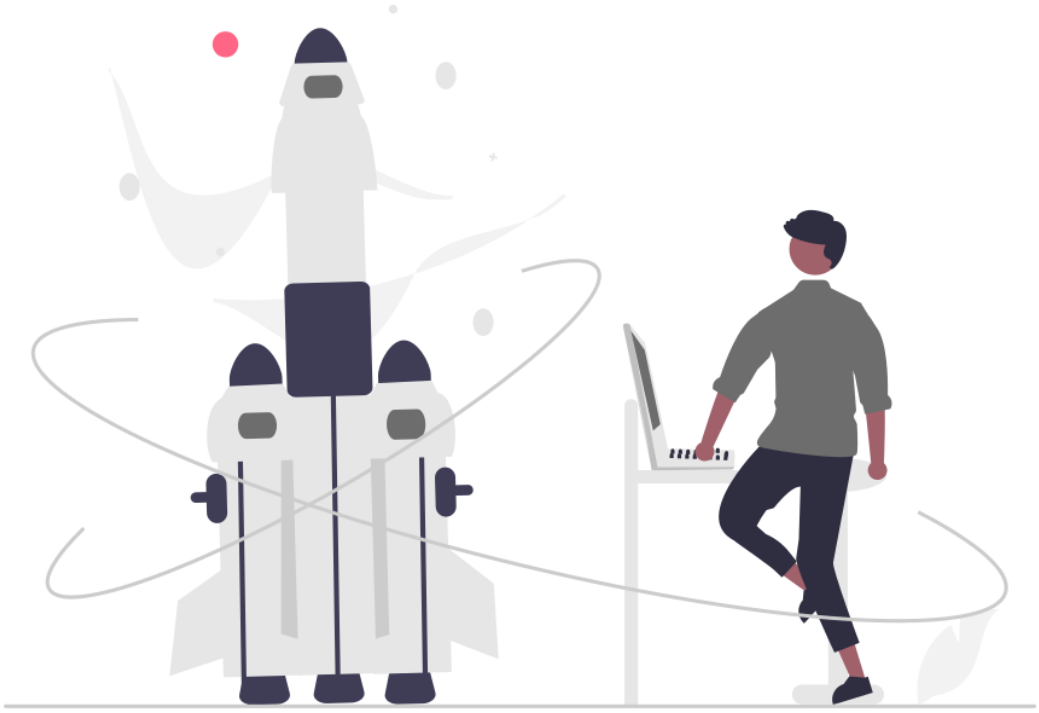
Encrypt/Decrypt, Encoding/Decoding နဲ့မတူတာကတော့ hashing က မူရင်း data ကို ပြန်မရနိုင်ပါဘူး။ Hashing ကို မူရင်း data မသိမ်းထားပဲ data တိုက်ဖို့ အတွက် အသုံးပြုကြပါတယ်။ ဥပမာ Passowrd တွေကို MD5 နဲ့ သိမ်းထားပြီး user က data ပြန်ထည့်သည့် အခါမှာတော့ user ထည့်လိုက်သည့် password ကို MD5 ပြောင်း။ ပြီးမှ database မှာ သိမ်းထားသည့် MD5 နဲ့ တူသလား စစ်။ Hashing က Data ကို Hash လုပ်သည့်အခါမှာ စာလုံးအရေအတွက် စာလုံး မပြောင်းလဲပါဘူး။ ဥပမာ md5("a") ဟာ အမြဲတန်း `0cc175b9c0f1b6a831c399e269772661` ပါပဲ။ ကျွန်တော်တို့တွေ a ကို database မှာ သိမ်းမည့် အစား `0cc175b9c0f1b6a831c399e269772661` ဆိုပြီး သိမ်းထားပါမယ်။ User က a ထည့်သည့်အခါမှသာ MD5 နဲ့ Hash လုပ်ပြီးတော့ user ထည့်တာ မှန်မမှန် ပြန်စစ်သည့် သဘောပါ။ database ထဲမှာ hash ပဲ ထည့်ထားသည့် အတွက်ကြောင့် user password ကို developer ကိုယ်တိုင်မသိနိုင်ပါဘူး။ datbase ပါသွားခဲ့ရင်တောင် user ရဲ့ password ကို သိဖို့ ခက်ပါတယ်။

```
md5("a") == "0cc175b9c0f1b6a831c399e269772661"
sha1("a") == "86f7e437faa5a7fce15d1ddcb9eaeaea377667b8"
```

MD5 hashing ထက် ကျွန်တော်တို့တွေ SHA1 , SHA256 စတာတွေကိုလည်း အသုံးပြုနိုင်ပါတယ်။ Hashing မှာ key ပါထည့်သွင်းရသည့် HMAC hash နဲ့ Key မရှိသည့် MD5, SHA1, SHA256 လိုမျိုး alogrithm တွေ ရှိပါတယ်။

အများအားဖြင့် Request တစ်ခု ပို့လိုက်သည့် အခါမှာ data တွေ transfer လုပ်သည့် အခါမှာ ပို့လိုက်သည့် data က ကြားမှာ loss ဖြစ်သွားလား။ တစ်ယောက်ယောက်က ပြောင်းလိုက်သလား သိနိုင်ဖို့ Hashing ကို အသုံးပြုကြပါတယ်။ အသုံးများတာကတော့ HMAC နဲ့ secrect key ကို အသုံးပြုပြီး Hash လုပ်ပါတယ်။ အထူးသဖြင့် API call တွေမှာ Hashing ကို အသုံးပြုကြပါတယ်။ အဓိက ကတော့ data ကို safe ဖြစ်အောင် အတွက် အသုံးပြုကြပါတယ်။

အခန်း ၁၉ :: Deployment

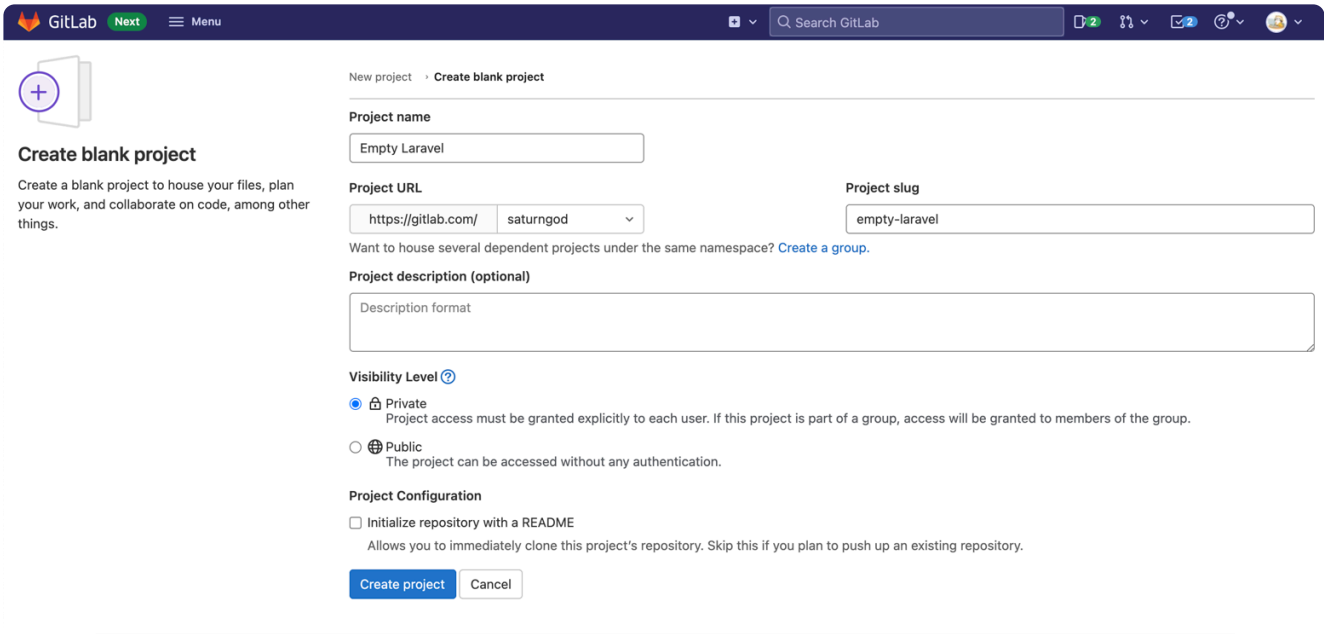


ကျွန်တော်တို့ အလုပ်စလုပ်ချိန်မှာ git ဟာ လူသုံးမများသေးပါဘူး။ Github မှာ 2008 မှာ စပေါ်လာပြီး 2010 မှာ လူသုံးများလာပါတယ်။ Github လူသုံးများလာတာနဲ့ အမျှ Git အကြောင်းလူတွေ သိလာကြပါတယ်။ Git နဲ့ Github ဟာ မတူပါဘူး။ Git ကို Linus Torvalds က 2005 မှာ စတင်ဖန်တီးခဲ့ပါတယ်။ အဲဒီ မတိုင်ခင်မှာ CVS ကို အသုံးပြုပါတယ်။ CVS ကို production deployment အတွက် အသုံးမပြုပဲ code backup , version backup တွေ အတွက် အသုံးပြုကြတာများတယ်။

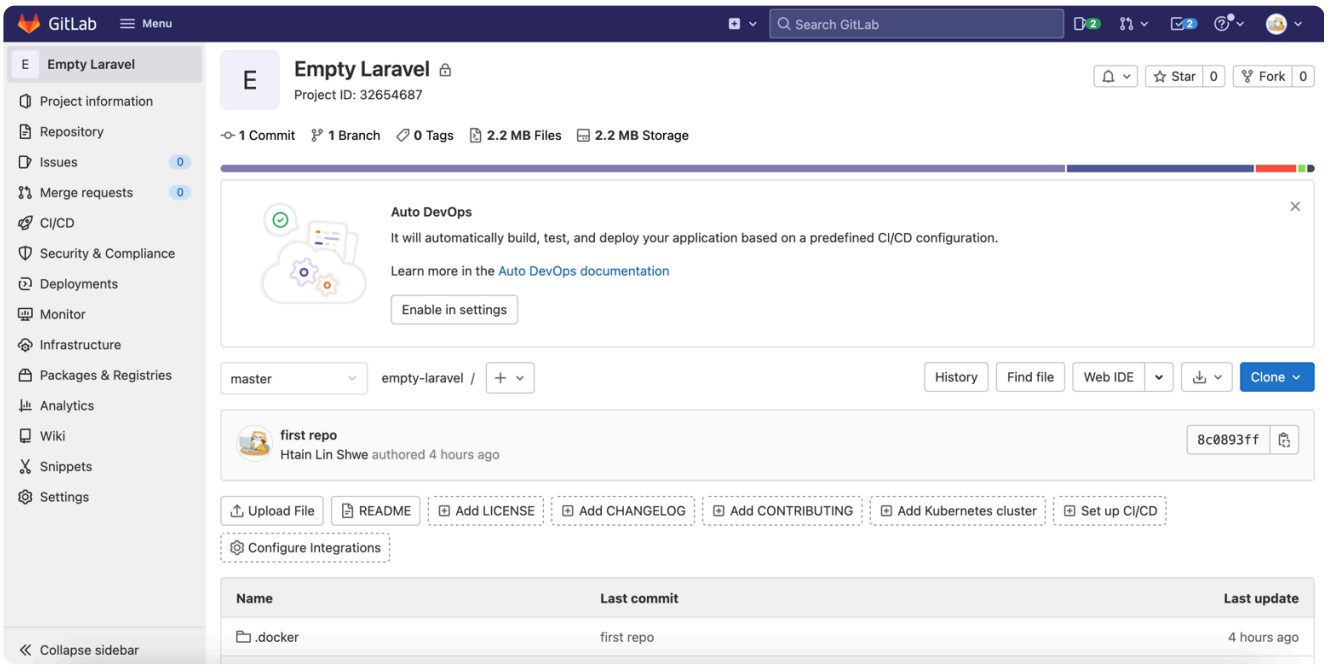
Git Repo

Deployment လုပ်သည့် အခါမှာ manual file တွေ ရွေ့တာကနေ Git ကို ပြောင်းလဲလာတယ်။ နောက်ပိုင်း CI/CD တွေ အသုံးပြုလာကြပါတယ်။ အခုခေတ်မှာတော့ company တိုင်း နီးပါက CI/CD ကို အသုံးပြုနေပါပြီ။ မျက်စိထဲ မြင်အောင်ပြောရရင် production branch ကို push လုပ်လိုက်တာနဲ့ တစ်ခါတည်း production ပေါ် ရောက်သွားတာမျိုးပေါ့။ CI/CD အတွက် platform ပေါ်မူတည်ပြီး script တွေ ကွာပါတယ်။ တချို့ Company တွေကတော့ Jenkins ကို self host အနေနဲ့ အသုံးပြုကြပါတယ်။ ကျွန်တော်ကတော့ Gitlab ကို အသုံးပြုပါတယ်။

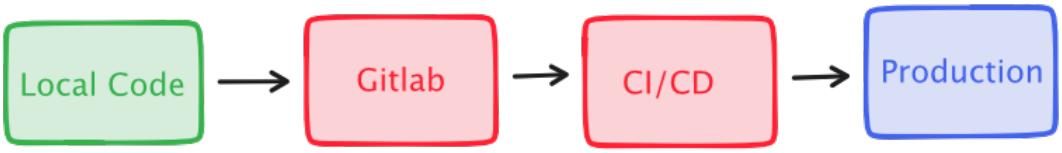
ပထမဆုံး gitlab.com မှာ account ရှိထားဖို့ လိုပြီး project အတွက် git repo တစ်ခု ရှိထားဖို့ လိုပါတယ်။



Code တွေကို အရင် push တင်ထားလိုက်ပါ။



ကျွန်တော်တို့ သွားမည့် အဆင့်က အောက်ပါ အတိုင်းဖြစ်ပါတယ်။



အရင်ဆုံး Code တွေကို Gitlab ကို push လုပ်မယ်။ Gitlab က နေ auto deploy လုပ်ပြီး production server ပေါ်တင်ပါမယ်။

Setup Key

အရင်ဆုံး Production server ကို SSH နဲ့ ဝင်ခွင့်ရအောင် Key အသစ် ဖန်တီးပါမယ်။ Terminal မှာ

```
ssh-keygen -t rsa
```

နဲ့ key အသစ်ဖန်တီးလိုက်ပါ။ Password ကို empty ထားပါ။

```
→  servertestkey ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/Users/htainlinshwe/.ssh/id_rsa): storekey
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in storekey
Your public key has been saved in storekey.pub
The key fingerprint is:
SHA256:XrdDFYEqz8/GDzimgbzVe7tp63N+rxT1ckk7h2kQC1Y htainlinshwe@Htains-MBP
The key's randomart image is:
+---[RSA 3072]----+
|          o.E.o. |
|         . ..o . |
|           .o ...|
|        . .  o.++|
|        S+. o *++|
|     . o oo+ o +o|
|      o + =++ .  |
|       o + oO=. .|
|      . . .=*O+oo|
+----[SHA256]-----+
→  servertestkey ls
storekey     storekey.pub
```

```
cat storekey.pub
```

ပြီးရင် public key ကို copy ကူးလိုက်ပါ။

```
→  servertestkey cat storekey.pub
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABgQCsxsKPIG4lJlRkGhiNNPMxFc2FUKNi4/Dk08CnzNOS
ouVz+MNY3FocpZjIe1uPmYlQBedhYihpCLtEg6RkmrCvO5vD/0GmYYWFnRZTNGouEt6+tDKS/s9rWPeD
wLXXx9xvivPUauh+3FMzY6ihMQCh61l/2nlkDVe7A2qTsS/YA4IuGgQTmkGJY2jrbA5v7xAn6mdlF/2q
```

အခု public key ကို server ရဲ့ `~/.ssh/authorized_keys` ထဲမှာ ထည့်ထားပါ။

ဒါဆိုရင်

```
ssh [username]@[yourip] -i [your private key]
```

ဥပမာ

```
ssh root@123.123.123 -i storekey
```

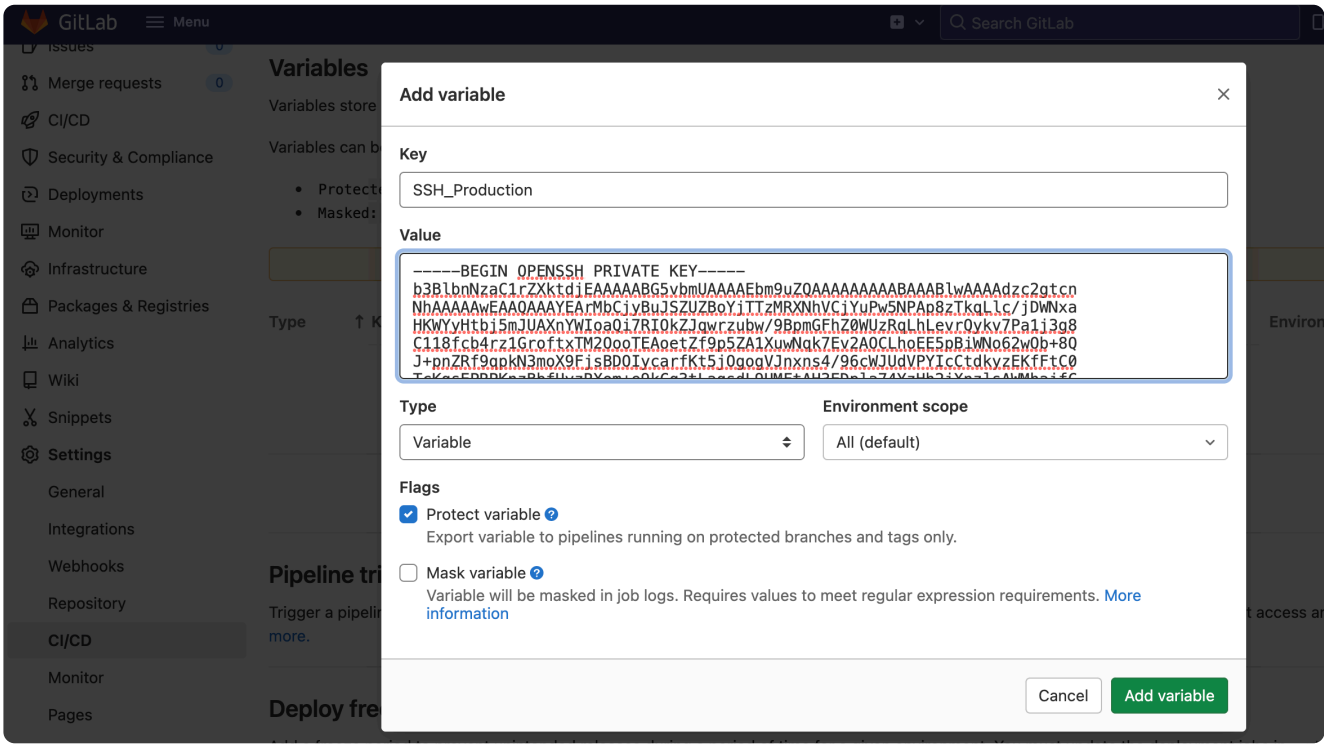
ဒါဆိုရင် တစ်ခါတည်း server ထဲ ဝင်လို့ရသွားပါမယ်။ Key က အရေးကြီးပါတယ်။

အကောင်းဆုံးကတော့ Linux Admin က deployment အတွက် သီးသန့် user နဲ့ deployment အတွက် permission ပေးထားတာ ပိုကောင်းပါတယ်။

Private key ကို ဖွင့်ပြီး copy ကူးယူပါ။

```
cat [your private key]
```

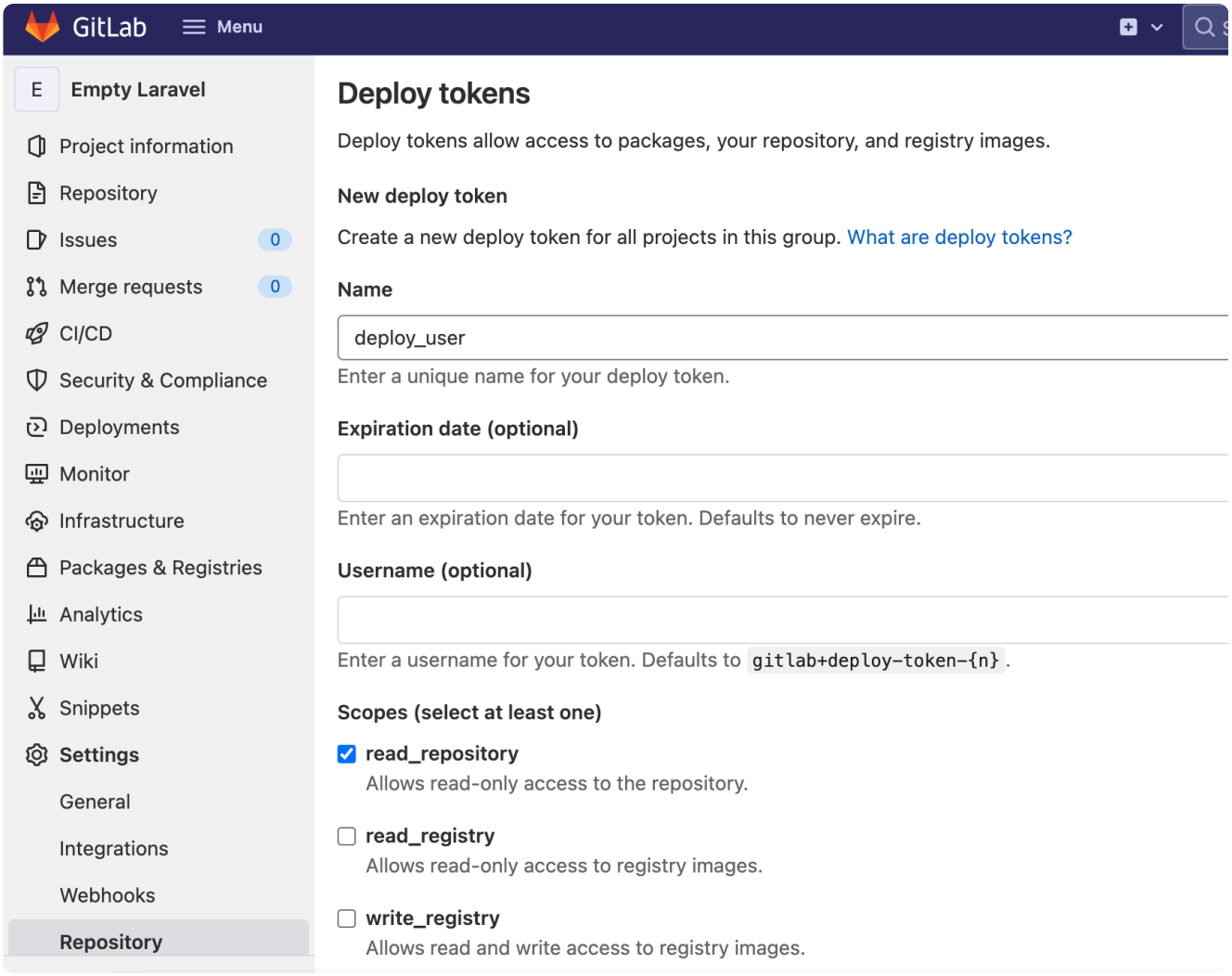
Gitlab Repo ရဲ့ Setting > CI/CD > Variable မှာ private key ကို variable အနေဖြင့် ထည့်ပါမည်။



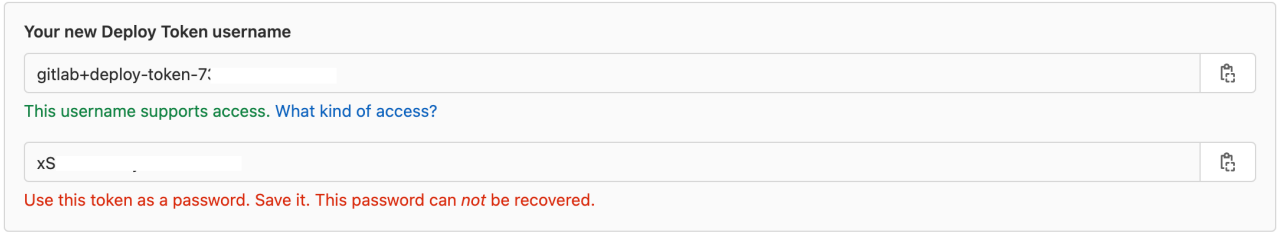
ပေထားသည့် variable name `SSH_Production` ကို နောက်ပိုင်း .gitlab-ci.yml မှာ ပြန် အသုံးပြုပါမယ်။

Readonly Account

Gitlab ရဲ့ Repo အောက်မှာ Setting > Repository > Deploy tokens မှာ read_repository တစ်ခုပဲ on ပြီးတော့ token generate လုပ်ပါမယ်။



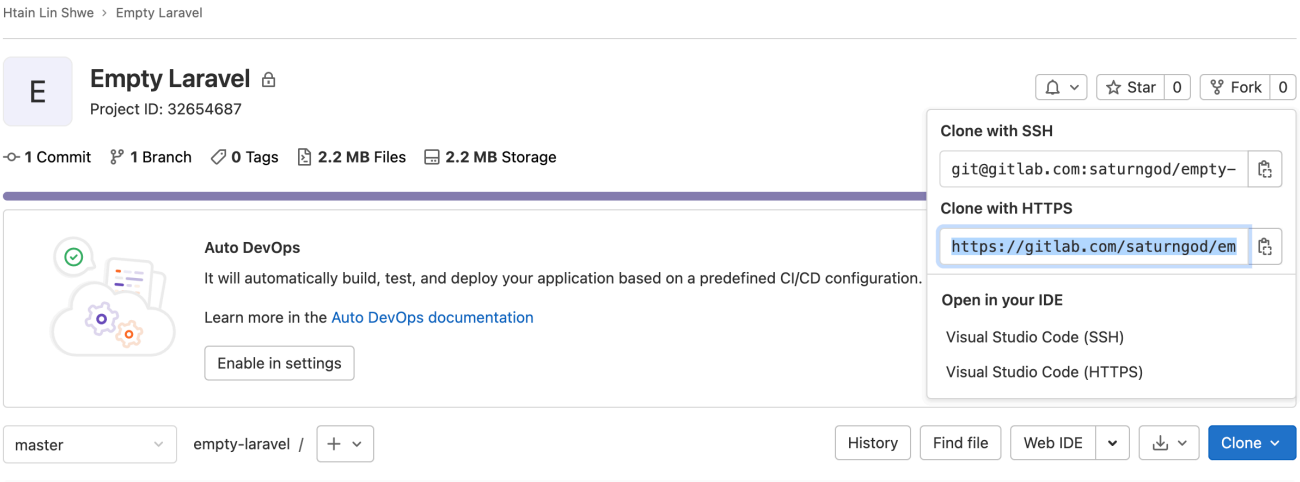
Create လုပ်ပြီးသွားရင် username နဲ့ token ထွက်လာပါမယ်။ note မှာ ဖြစ်ဖြစ် အဆင်ပြေသည့် နေရာမှာ သေချာ မှတ်ထားဖို့ လိုပါတယ်။ password က recover ပြန်မရနိုင်ပါဘူး။



အခု production server မှာ git အတွက် ရရှိလာသည့် token နဲ့ တည်ဆောက်လို့ရပါပြီ။

```
cd /var/www/mysample
git init
```

Git init ပြီးသွားရင် အခု Repo URL ကို setup လုပ်ပါမယ်။ Repo URL ကို Gitlab ကနေ https ကို အသုံးပြုဖို့ လိုပါတယ်။




```
git remote add origin https://[username]:[token]@[url]
```

\[username\] ကတော့ generate လုပ်ထားသည့် username ပါ။

\[token\] ကတော့ generate လုပ်ထာသည့် အချိန်က ရလာသည့် token ပါ။

\[url\] ကတော့ gitlab.com/xxxx/xxx ပါ။ gitlab.com က copy ကူးလာသည့် url ပါ။

```
git pull origin master
```

ဆိုပြီး pull ဆွဲရင် password မလိုတော့ပဲ pull ဆွဲလို့ရနေပါပြီ။ ပုံမှန်အားဖြင့် read only ပဲ ထားပါတယ်။ server ပေါ်မှာ changes လုပ်ပြီး local မှာ changes မလုပ်ခဲ့ရင် မေ့သွားခဲ့ရင် နောက်တစ်ခါ deployment လုပ်သည့် အချိန်မှာ အကုန်ပျက်ကုန်ပါတယ်။

.gitlab-ci.yml

Project ရဲ့ အောက်မှာပဲ `.gitlab-ci.yml` ဆောက်ပါ။

```
→  store git:(master) touch .gitlab-ci.yml
→  store git:(master) ✗ ls -a
.                   .gitattributes      bootstrap           public
..                  .gitignore          composer.json       resources
.DS_Store           .gitlab-ci.yml      composer.lock       routes
.docker             .styleci.yml        config              server.php
.editorconfig       Dockerfile          database            storage
.env                README.md           docker-compose.yml  tests
.env.example        app                 package.json        vendor
.git                artisan             phpunit.xml         webpack.mix.js
```

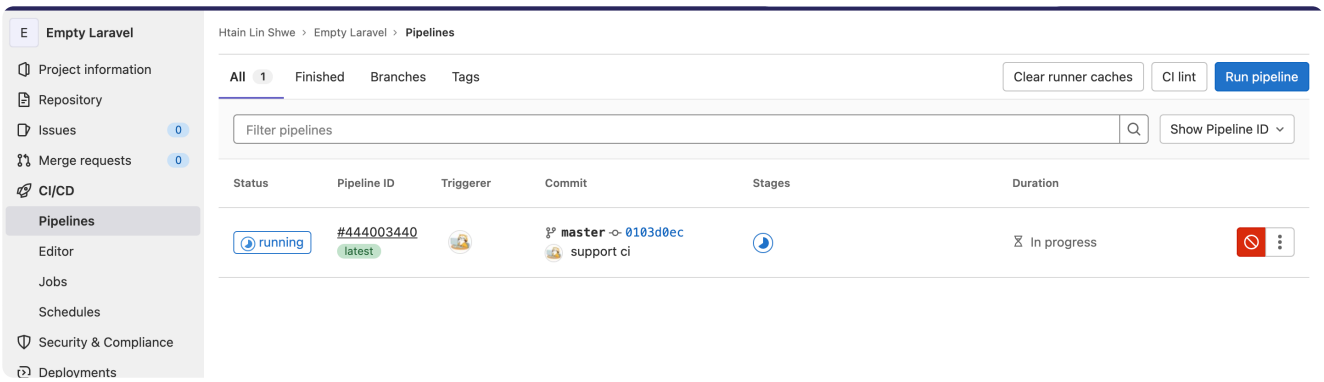
`.gitlab-ci.yml` ထဲမှာ အောက်ပါ အတိုင်းထည့်လိုက်ပါ။

```
image: ubuntu:latest

deploy:
  stage: deploy
  only:
    - master
  before_script:
    - 'which ssh-agent || ( apt-get update -y && apt-get install openssh-client -y )'
    - eval $(ssh-agent -s)
    - ssh-add <(echo "$SSH_Production")
    - mkdir -p ~/.ssh
    - echo -e "Host *\n\tStrictHostKeyChecking no\n\n" > ~/.ssh/config

  script:
    - ssh root@[your_server_IP] "cd '[your_path]'; git pull origin master;composer
install;"
```

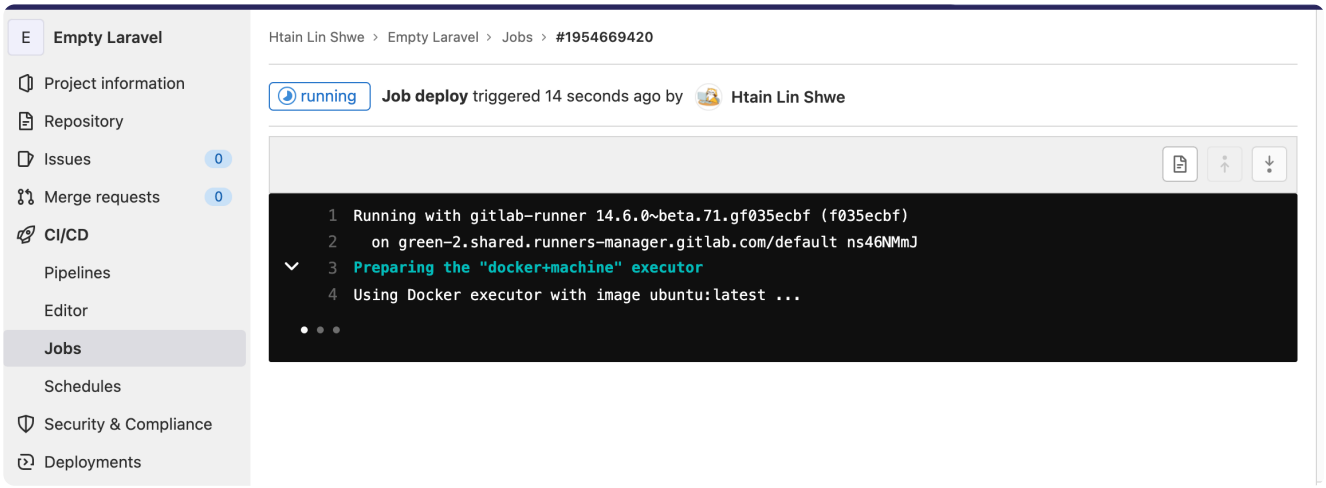
\[your_server_IP\] က သင့်ရဲ့ server IP ပါ။

\[your_path\] ကတော့ သင့်ရဲ့ server ပေါ်မှာ ရှိသည့် file path ပါ။ ဥပမာ /var/www/sample/

`.gitlab-ci.yml` ကို update လုပ်ပြီးသွားရင် git master branch ကို commit လုပ်ပြီး ပြန် push လိုက်ပါ။



CI/CD > Pipeline မှာ run နေတာကို တွေ့ရမှာပါ။



running ကို နှိပ်လိုက်ရင် သက်ဆိုင်ရာ job ကို ရောက်သွားပြီး processing လုပ်နေတာတွေကို တွေ့ရမှာပါ။

ပြဿနာမရှိဘူး ဆိုရင် Job Success နဲ့ ပြီးသွားတာကို တွေ့နိုင်ပါတယ်။

`.gitlab-ci.yml` ထဲမှာ ဘာတွေ ရေးထားလဲ လေ့လာကြည့်ရအောင်။ Gitlab က CI က တကယ် တန်းတော့ docker image ပေါ်မှာ run တာပါပဲ။

`image: ubuntu:latest` ဆိုတာကတော့ Ubuntu နောက်ဆုံး version image ကို ယူပါမယ်။

```
only:
    - master
```

master branch ကို push လုပ်မှ script က စ အလုပ်လုပ်မယ် လို့ ကြေငြာထားတာပါ။

```
only:
  - tags
```

အကယ်၍ အပေါ်က အတိုင်း ပြောင်းရေးထားမယ် ဆိုရင်တော့ push လုပ်သည့် အထဲမှာ tags ပါလာမှသာ အလုပ်လုပ်မယ် လို့ ဆိုလိုတာပါ။

```
before_script:
    - 'which ssh-agent || ( apt-get update -y && apt-get install openssh-client -y )'
    - eval $(ssh-agent -s)
    - ssh-add <(echo "$SSH_Production")
    - mkdir -p ~/.ssh
    - echo -e "Host *\n\tStrictHostKeyChecking no\n\n" > ~/.ssh/config
```

script မ run ခင်မှာ SSH key ကို docker image ထဲမှာ ထည့်ထားလိုက်တာပါ။ အပေါ်မှာ ကြေငြာထားခဲ့သည့် SSH_Production ကို သုံးထားတာ ကိုတွေ့နိုင်ပါတယ်။

```
script:
    - ssh root@[your_server_IP] "cd '[your_path]'; git pull origin master;composer
install;"
```

Server ကို SSH နဲ့ ဝင်ပြီး git pull ဆွဲလိုက်တာပါ။

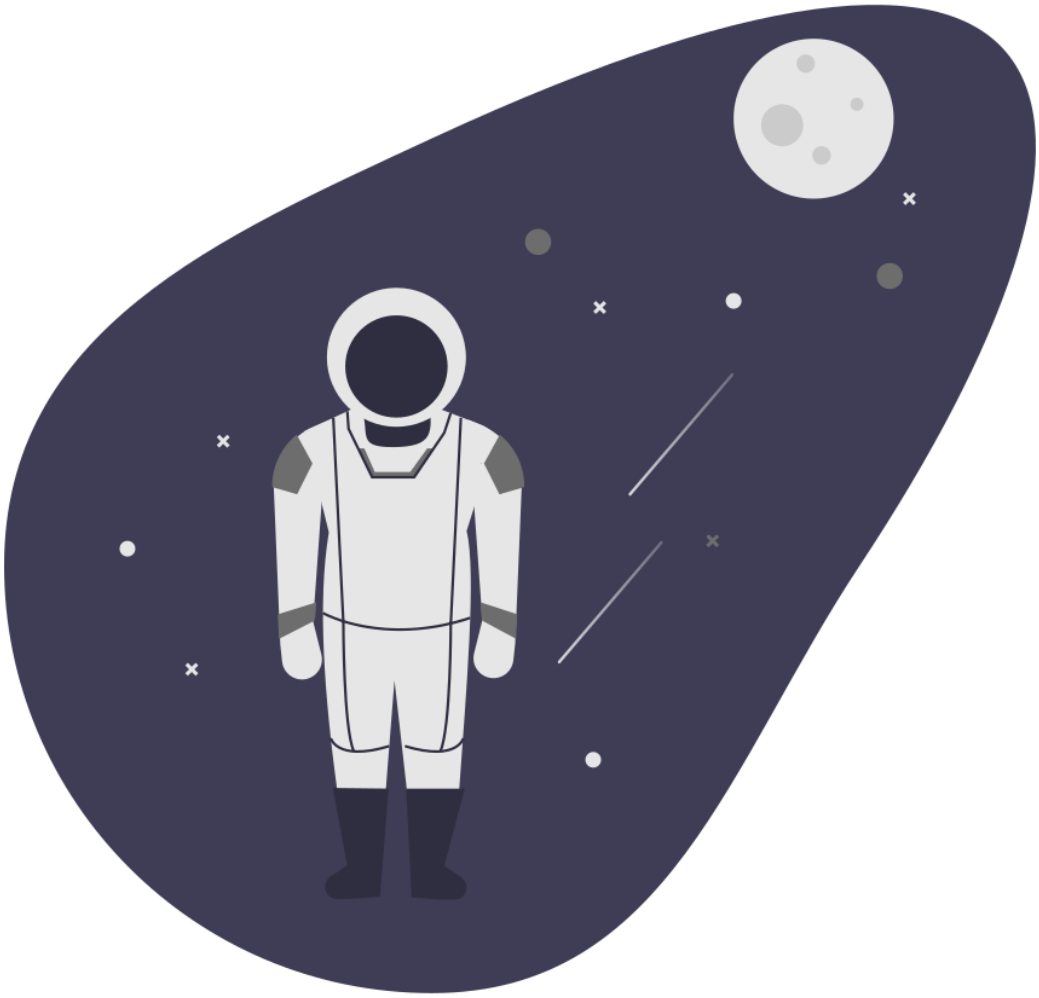
အကယ်၍ unit testing တွေပါ ထည့်ချင်ရင် အောက်ပါ အတိုင်း ပြင်နိုင်ပါတယ်။

```
stages:
  - build
  - test
  - deploy

job1:
  stage: build
  script:
    - echo "This job compiles code."

job2:
  stage: test
  script:
    - echo "This job tests the compiled code. It runs when the build stage completes."

job3:
  script:
    - echo "This job also runs in the test stage".

job4:
  stage: deploy
  script:
    - echo "This job deploys the code. It runs when the test stage completes."
```

Gitlab CI/CD က အရမ်းကို အသုံးဝင်ပါတယ်။ Multiple server တွေ micro services deployment တွေ docker deployment တွေ အတွက် အသုံးဝင်လှပါတယ်။

အခုနောက်ပိုင်း Github Action ကို စမ်းကြည့်ဖို့ အကြံပေးလိုပါတယ်။ Github Action အတွက် CI/CD က သီးသန့် ပြန်ရေးသားရပါတယ်။ Gitlab နဲ့ မတူပါဘူး။ ဒါကြောင့် CI/CD ကို သဘောတရား နားလည်ဖို့ အဓိက ဖြစ်ပြီး ကိုယ်သုံးမယ့် CI/CD ပိုင်းကို လေ့လာ ဖို့ လိုပါတယ်။

အခန်း ၂၀ :: Company တစ်ခု တည်ထောင်ခြင်း



Developer အများစုရဲ့ အိမ်မက်ကတော့ ကိုယ်ပိုင် company တစ်ခု တည်ထောင်လိုချင်းပါပဲ။ ချမ်းသာချင်တာ ထက် ကိုယ်ပိုင် အယူအဆ ကိုယ်ပိုင်ဟန် နဲ့ company တစ်ခု ရှိချင်ကြတာပါ။ Company ထောင်လိုက်တာနဲ့ ချမ်းသာသွားသည့် သူတွေက အနည်းစု လို့ ဆိုရပါမယ်။

Company တစ်ခု ထောင်မယ်ဆိုရင် မြန်မာနိုင်ငံမှာ ဆိုရင်တော့ https://www.myco.dica.gov.mm/ မှာ register လုပ်လိုက်ရင် ရပါပြီ။ Company Registration Number ရရင် သင် Company ထောင်ပြီးပါပြီ။

ကျွန်တော်ကိုင်တိုင် COMQUAS Co.,ltd ဆိုပြီး တည်ထောင်ထားပါတယ်။ အဓိက mobile app ကို customize ရေးပေးသည့် company တစ်ခုပါ။ ဒီ အခန်း မှာ ကျွန်တော့် အတွေ့အကြုံကို အခြေခံပြီး ရေးသားထားခြင်း ဖြစ်ပါသည်။

ချမ်းသာချင်လို့ Company ထောင်တယ် ဆိုရင်တော့ မမှားပေမယ့် အရမ်းမှန်တယ်လို့ မထင်ပါဘူး။ Company မထောင်ထားပဲ investment တွေ နဲ့ ကြွယ်ဝ ချမ်းသာနေသူ အခြားသူမှာ လုပ်နေမယ့် ရာထူးကြီးကြီး နဲ့ လခ များများ ရနေသူတွေ အများကြီး ရှိပါတယ်။ ရသည့် လခ ထဲကမှ ပို သည့် ပိုက်ဆံ နဲ့ အခြား investment တွေ လုပ်ရင်းနဲ့ ချမ်းသာနေသည့် သူတွေ လည်း ရှိပါတယ်။ ချမ်းသာဖို့ အတွက် company ထောင်ချင်းက first priority မဟုတ်ဘူးလို့ အကြံပေးချင်ပါတယ်။ ကျွန်တော့် ကိုယ်ပိုင် အမြင်ကတော့ Investment က ပို အရေးပါတယ်လို့ မြင်ပါတယ်။

နောက်ပြီး အရေးကြီးသည့် အချက်က ကိုယ်ပိုင် company ထောင်ပြီး management ပိုင်းကို ရောက်သွားသည့် အခါမှာတော့ coding တွေ နဲ့ အလှမ်းဝေးသွားပါတယ်။ အကယ်၍ company မ အောင်မြင်ခဲ့လို့ အလုပ်ပြန်လုပ်မယ်ဆိုရင် developer အနေနဲ့ အလုပ်ရချင်မှ ရပါလိမ့်မယ်။ နိုင်ငံတကာမှာ နည်းပညာ တွေ က နှစ်စဉ်ပြောင်းလဲ နေသည့် အတွက် သက်ဆိုင်ရာ နယ်ပယ်ထဲမှာ မဟုတ်ခဲ့ရင် ၆ လ လောက် အဆက်အသွယ်ပြတ်သွားရင် အတော်လေးကို ပြန်လိုက်မှ ရပါလိမ့်မယ်။ ဒါကြောင့် ယုံကြည်ချက် အပြည့်ရှိတယ် risk လည်း ခံနိုင်တယ် ဆိုမှသာ ကိုယ်ပိုင် startup လေး တစ်ခု စဖို့ အကြံပြုလိုပါတယ်။

Business Plan

Company တစ်ခုကို စ လုပ်တော့မယ်ဆိုရင် ဘာတွေ service ပေးမှာလဲ ကို ပထမဆုံး မေးခွန်းပါ။ Product ရောင်းမှာလား service ရောင်းမှာလား။ Product ရောင်းမယ့် plan နဲ့ service ရောင်းသည့် plan က မတူညီ သည့် အတွက် ဦးစွာ ဆုံးဖြတ် ဖို့ လိုပါတယ်။ Product ရောင်းရင် service လုပ်လည်း ဖြစ်သလို service ရောင်းရင်း product လုပ်လည်း ဖြစ်ပါတယ်။ သို့ပေမယ့် အဓိက ဦးတည်ချက်ကို ထားဖို့ လိုပါတယ်။

ကိုယ့်ရဲ့ Milestone တွေက ဘာလဲ ? Business Plan က ဘယ်လို ရှိလဲ။ စတာတွေကို စဉ်းစားရပါတယ်။ လုပ်ရင်း တတ်သွားပါမယ် ဆိုပေမယ့် plan သေချာဆွဲထားသည့် အခါမှာ အကျသက်သာပါတယ်။ ခံ ရတာ သက်သာပါတယ်။

Burning Rate

ကိုယ့် Product အတွက် ၁ လ ဘယ်လောက်ကုန်မလဲ။ Product မထွက်ခင်မှာ ဝန်ထမ်း ဘယ်နှစ်ယောက် လိုမလဲ ? သူတို့ အတွက် လခ ဘယ်လောက်ကုန်မလဲ ? Product ထွက်ပြီးရင် ဝန်ထမ်း ဘယ်နှစ်ယောက်ပဲ လိုတော့မလဲ ? ဒါမှမဟုတ် ပိုလိုလာမလား။ အခန်းခ ဘယ်လောက်ကုန်မလဲ။

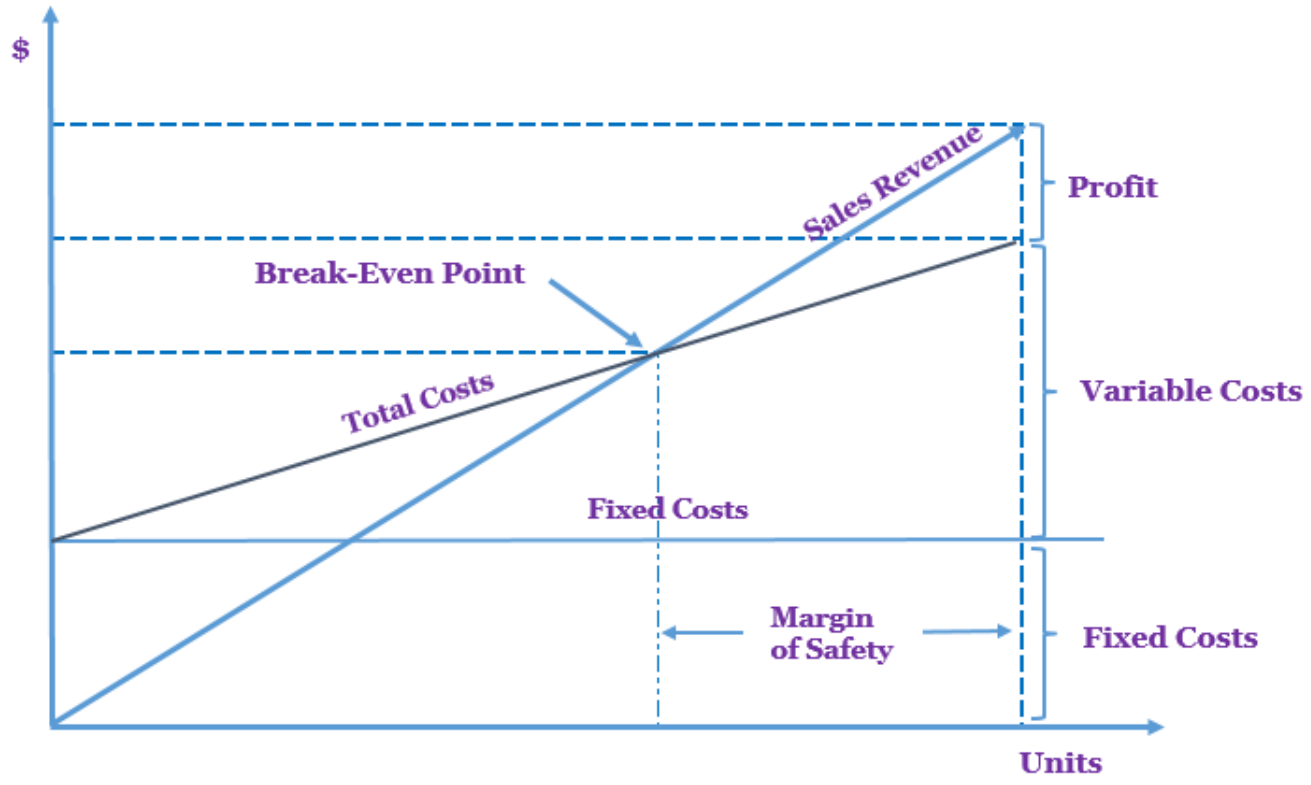
စသည်ဖြင့် လစဉ် ကုန်ကျစရိတ်ကို ၂ နှစ်စာလောက် တွက်ထားဖို့ လိုပါတယ်။

Service လုပ်မယ်ဆိုရင် တွက်ဖို့ က သိပ်မလိုပါဘူး။ သို့ပေမယ့် project ဘယ်လို ရှာရမလဲ။ Project ရခဲ့ရင် လူဘယ်လိုရှာရမလဲ ဆိုတာက ခေါင်းစားပါတယ်။ ဝန်ထမ်းမရှိပဲ နဲ့ project ရဖို့ ခက်ပါတယ်။ ဒါကြောင့် အနည်းဆုံး ဝန်ထမ်း ဘယ်လောက်နဲ့ စမယ်။ project မရခင်ကာလခ ဘာတွေ ခိုင်းထားမယ်။ project ကုန်သွားသည့် အခါမှာ ဘာတွေ ခိုင်းမယ် ဆိုတာက Service ပေးသည့် company တွေ အတွက် challenge ပါပဲ။

၁ လစာ Burning Rate ကို တွက်ထားဖို့က အရေးကြီးပါတယ်။ ကိုယ့်မှာ အနည်းဆုံး ၂ နှစ်စာ လုံလုံလောက်လောက် ရှိထားမှသာ စလို့ ကောင်းပါတယ်။

Break Event

Company တစ်ခုဟာ လစဉ် ဝင်ငွေ ရှိဖို့ လိုအပ်ပါတယ်။ နောက်တချက်က လစဉ် ဝင်ငွေ တိုးနေဖို့ လည်း လိုအပ်ပါတယ်။ Burning Rate က အချိန်တစ်ခု ရောက်ရင် များစွာ မြင့်တက်တော့မှာ မဟုတ်ပါဘူး။



ဒါကြောင့် company မတင်ခင်မှာ ဘယ်အချိန်ရောက်ရင်တော့ Break Event ဖြစ်ရမယ် ဆိုပြီး milestone သတ်မှတ်ထားဖို့ လိုပါတယ်။

တစ်လခြင်း ခန့်မှန်း ကုန်ကျစရိတ် ဝင်ငွေ တွေကို excel မှာ ထည့်ပြီး break event ဖြစ်မည့် chart ကို ဆွဲထားခြင်းဖြင့် ကိုယ့် company အခြေအနေဟာ milestone ကို ထိ နိုင်မလား ကျ နေပြီလား နောင်အချိန်မှာ သိနိုင်ပါတယ်။

Account

Company အသုံးစရိတ် အသေးသုံး စရိတ် တွေမှာ စတည်ထောင် ကတည်းက ပုံမှန် ထည့်သွင်း ထားဖို့ လိုပါတယ်။ ဒါဟာ အရေးကြီးပြီး အကျင့် တစ်ခု လို ဖြစ်နေဖို့ လိုပါတယ်။ Developer အများစုဟာ အသုံးစရိတ် တွေကို ခေါင်းထဲမှာ မှတ်ထားပြီး နောက်မှ ပြန်ထည့်မယ် ဆိုပြီး ခဏခဏ လုပ်တတ်သည့် အကျင့်ရှိပါတယ်။ စာရင်း မကိုင်နိုင်ရင် စာရင်းကိုင်တစ်ယောက် ကို part time ဖြစ်ဖြစ် ငှားထားဖို့ လိုပါတယ်။

Audit Firm

နှစ်စဉ် အခွန်အတွက် စာရင်းရေးဆွဲ ခြင်း ကို ဆောင်ရွက်ရပါတယ်။ ဒါကြောင့် ကိုယ် နဲ့ အဆင်ပြေမယ့် Audit Firm တစ်ခုနဲ့ ဆောင်ရွက်ရန် လိုအပ်ပါသည်။ ၁ နှစ် စာ bill , invoices စ တာတွေ အကုန်လုံးကို ပုံမှန် စာရင်း ပြုစုထားခြင်းအားဖြင့် Audit အတွက် စာရင်းရေးဆွဲရာမှာ ပိုမို မြန်ဆန် အဆင်ပြေစေပါတယ်။

Contract

ဘယ်သူနဲ့ မဆို အလုပ်လုပ်သည့် အခါမှာ Contract ကို မဖြစ်မနေ ထိုးဖို့ လိုပါတယ်။ Contract ရှိ ထားခြင်းအားဖြင့် ကိုယ်လုပ်ဆောင်ရမယ့် အချက်တွေ တဖက် customer/client က လုပ်ဆောင်ရ မယ့် အချက်တွေကို ရှင်းလင်းစေပါတယ်။ နောက်ပြီး ငွေချေရန် နောက်ကျ ခဲ့ရင် ထပ်တိုး ကောက်ခံမှုတွေ ထည့်သွင်းထားဖို့ လိုပါတယ်။ ပုံမှန် အားဖြင့် client/customer တွေ ဆီက ငွေရ သည့် အခါ ပုံမှန် ထက် အမြဲ နောက်ကျတတ်ပါတယ်။ Contract ဖြင့် လုပ်ဆောင်ခြင်းအားဖြင့် တဖက်နဲ့ တဖက် နဲ့ ပိုမို ယုံကြည်စိတ်ချစွာ အလုပ်လုပ်ကိုင် နိုင်ပါတယ်။

Quotation

Project တစ်ခု မစတင်ခင်မှာ ခန့်မှန်း ကုန်ကျစရိတ်ကို တင်ရပါတယ်။ Quotation ကို ကြည့်ပြီး ဈေးပြန်ညှိခြင်း လိုတာတွေ ထပ်ဖြည့်ခြင်း စတာတွေ ကို လုပ်ရပါတယ်။ Quotation ဆိုတာ Invoice လိုပါပဲ။ Invoice ကတော့ ပိုက်ဆံ ရရန် ရှိသည့် အခါမှာ ပေးပို့ပြီး Quotation ကတော့ ခန့် မှန်းကုန်ကျ ငွေကို ပေးပို့ခြင်းဖြစ်ပါတယ်။

Risk

Service ပေးရသည့် အခါမှာ အခက်အခဲဆုံး က အချိန် ဘယ်လောက်လို မလဲ ဘယ်လောက်ကြာ မလဲ ဆိုတာပါပဲ။ တစ်ခါတစ်လေ Developement ရှေ့ဆက်ဖို့ client ရဲ့ feedback လိုသည့် အခါ မှာ ခရီးထွက်သွားတာတို့ အခြား ကိစ္စကြောင့် အချိန်မပေးနိုင်တာတို့ ကြောင့် timeline က နောက်ကျ ကုန်သည့် အခါမှာ ဝန်ထမ်း လခ တွေ စိုက်ပေးထားရတာတွေ ရှိတတ်ပါတယ်။

တစ်ခါတစ်လေ ထင်ထားသည့် အတိုင်းမ ဖြစ်လာပဲ လုပ်ရတာ ကြာလာတာမျိုးတွေ လုပ်နေရင်း version changes ကြောင့် ပြင်ရတာ ကြာလာတာမျိုးတွေ ကြောင့် ထည့်မတွက်ထားသည့် ကုန်ကျစရိတ်တွေ ရှိလာနိုင်ပါတယ်။ ဥပမာ Google Play Store တင်ဖို့ Android SDK 30 ကို target ထားလိုက်သည့်အခါမှာ အချို့ permission အပြောင်းအလဲ ကြောင့် ပြန်ပြင်ရတာတွေ။ Framework ကို update လုပ်မိလို့ changes တွေကြောင့် ပြန်ပြင်ရတာတွေ အပြင် အခြား ပြဿနာများစွာ လုပ်နေရင်း ကြုံရတတ်ပါတယ်။

တစ်ခါတစ်လေ ထည့်မတွက်ထားသည့် ပြဿနာတွေလည်း ကြုံရပါတယ်။ ဥပမာ product မပြီးခင်မှာ တာဝန်ယူထားသည့် developer အလုပ်ထွက်သွားတာ မျိုးပေါ့ ။ developer အသစ် မရ ခင် ကြားကာလ မှာ timeline မှီဖို့ ခက်ခဲသလို team တစ်ခုလုံးလည်း ခက်ခဲပါတယ်။

Business Development Manager

ကိုယ်တိုင် proposal တွေ မရေးနိုင်ရင် ရေးဖို့ အချိန် မရှိရင် ဒါမှမဟုတ် presentation တွေ ပြင်ဖို့ အချိန် မရှိရင် Business Development Manager တစ်ယောက် ခန့်ထားဖို့ လိုပါတယ်။ Client နဲ့ contract ကိစ္စတွေ proposal တွေ presentation စတာတွေကို လုပန်ဖို့ အတွက်ပါ။ နောက်ပြီး client အဟောင်းတွေ နဲ့ အဆက်အသွယ် မပြတ်သွားအောင် client နဲ့ relationship ကောင်းအောင် လုပ်ဖို့က Business Development Manager တာဝန်တွေ ဖြစ်လို့ ကိုယ် မလုပ်နိုင်သည့် အလုပ်တွေ လွှဲထားဖို့ လိုပါတယ်။

မပြီးဆုံးနိုင်သည့် Project များ

အချို့ project တွေက မပြီးနိုင်ဘူးထင်ရလောက်အောင် bugs တွေ ပြဿနာတွေ featurs တွေ နဲ့ ရှိနေတတ်ပါတယ်။ Client ပေါ်မှာ မူတည်ပြီး ပြဿနာတွေ ဖြေရှင်းရပုံ မတူပါဘူး။ Project ကြီးလာသည့် အခါမှာ bug များတာကတော့ ပုံမှန်ပါပဲ။

တစ်ခါတစ်လေ ပြီးပြီထင်သည့် အခါမှ ထပ်ရောက်လာသည့် feature တွေ design changes တွေကြောင့် မပြီးနိုင်အောင်ပဲ ရှိတတ်တယ်။ Product Owner အနေနဲ့ project ကို ပြီးမြောက်အောင် ပါဝင် ကူညီနိုင်ဖို့ လိုအပ်တယ်။

တချို့ project တွေကတော့ production ရောက်သည့် အခါမှသာ ထွက်လာသည့် ပြဿနာတွေ အခြား department က ပါလာသည့် အခါမှ ထွက်လာသည့် ပြဿနာတွေလည်း ရှိပါတယ်။ ဥပမာ ထွက်လာသည့် Report က Account ဘက်က လိုချင်သည့် ပုံစံ မဟုတ်လို့ ပြန်ပြင်ခိုင်းထား Report အသစ်တွေ ထပ်ပါလာတာ။ အစက ပေးထားသည့် export format နဲ့ မတူပဲ လုံးဝ ပြောင်းလဲတာတွေ ရှိတတ်ပါတယ်။

အတက်အကျ

Company တစ်ခု တိုးတက်မှုဟာ break event chart ဆွဲထားသလို linear ဖြစ်မနေပါဘူး။ တက်လိုက် ကြလိုက်နဲ့ ရုန်းကန် ရပါတယ်။ လူတွေ တိုးခန့်တာ ရှိသလို ပြန်ဖြုတ်ရတာတွေ လည်း ရှိတယ်။ ကိုယ့် company ဝင်ငွေဟာ ကိုယ့်ကို ငွေပေးနေသည့် customer တွေ client တွေ ပေါ်မှာ မူတည်ပါတယ်။ ကိုယ့် customer တွေ ငွေမပေးနိုင်တော့သည့် အခါမှာ revenue က ကျဆင်းပါတယ်။ လစဉ် ရရှိသည့် revenue ကနေ ဝန်ထမ်းတွေ ကို လခ ပေးပြီး လည်ပတ်နေရတာပါ။ Startup တွေမှာ တည်ထောင်သူတွေဟာ လခ မယူပဲ အလုပ်လုပ်နေကြပါတယ်။ အနည်းဆုံး ၃ နှစ် ၄ နှစ်လောက်မှ စပြီး လခ ယူကြပါတယ်။ တည်ထောင်သူတွေ လခ ထည့်ယူဖို့ အတွက် company ဝင်ငွေဟာ သူတို့အတွက် ပါ ပေးနိုင်သည့် အဆင့် ထိ အရင် ရောက်ဖို့ လိုအပ်ပါတယ်။

မမှန်သည့် သီအိုရီများ

တစ်ခါတစ်လေ စာတွေ ဖတ်ပြီး company ထောင်သည့် အခါမှာ စာထဲက အတိုင်း လိုက်လုပ်ဖို့ ကြိုးစားတာတွေ ရှိတတ်တယ်။ အများစု startup စာအုပ်တွေက အနောက်နိုင်ငံကို အခြေခံထားတယ် စာရေးသူတွေ လုပ်ခဲ့သည့် နိုင်ငံတွေကို ပဲ အခြေခံထားတာပါ။ နိုင်ငံတစ်နိုင်ငံနဲ့ တစ်နိုင်ငံ မတူသည့် အတွက် မြန်မာနိုင်ငံ အတွက် အရမ်းအသုံးဝင်တာ မဟုတ်ပါဘူး။ ဥပမာ နိုင်ငံခြား company တွေ ပြည်တွင်းမှာ လာလုပ်ရင် ဝေလ ငါးကြီး ချောင်းထဲ လာသလိုပဲ ဆိုပြီး ဂျက်မ ပြောပေမယ့် တရုတ်နိုင်ငံ အတွက် မှန်ကောင်းမှန်လိမ့်မယ်။ မြန်မာနိုင်ငံ မျိုးအတွက် ဆိုရင် နိုင်ငံတကာ company တွေ နဲ့ ယှဉ်ပြိုင်ရတာ ခက်ပါတယ်။ အထူးသဖြင့် Quality , Document , လစာ အပိုင်းတွေ ပြည်တွင်း company အများစုက အားနည်းကြတယ်။ ကိုယ့် ဝန်ထမ်းတွေက လစာ အများကြီးပေးသည့် နိုင်ငံ ခြား company တွေ ဆီ ရောက်သွားတတ်ပါတယ်။ ဒီလိုပဲ နာမည်ကြီးသည့် လူတွေ ပြောသည့် စကားတွေဟာ သူတို့ အတွက် မှန်ပေမယ့် ကိုယ့်အတွက် မှန်ချင် မှ မှန်ပါလိမ့်မယ်။ ဘဝ အတွေ့အကြုံကနေပဲ လေ့လာပြီး သင်ယူသွားရမှာပါ။

Network

Network က အလုပ်လုပ်သည့် အခါမှာ အရေးပါပါတယ်။ ဘာလုပ်လုပ် အမျိုးတော်မှ ရပါတယ်။ အမျိုးတော်တယ် ဆိုတာ ကိုယ့်ကိုယ် ယုံကြည်ဖို့ recommend ပေး နိုင်မယ့် သူ တွေ နေရာတိုင်းမှာ ရှိနေဖို့ပါ။ အလုပ် အတော်တော်များများဟာ အချင်းချင်း ချိတ်ဆက်ပြီး ဆောင်ရွက်နေရတာပါ။ လုပ်ရင်း ကိုင်ရင်း နဲ့ network တွေ တည်ဆောက်သွားရတာပါ။

မပြီးနိုင်သည့် နေ့ရက်တွေ

အရင်က သူများမှာ လုပ်သည့် အခါမှာ မပြီးနိုင်ဘူး ထင်ရပေမယ့် နားရက်တွေ ရှိပါတယ်။ ကိုယ်ပိုင် စလုပ်သည့် အချိန်မှာ နားရက် ဆိုတာ မရှိသလောက် အလုပ်ရှုပ်နိုင်တယ်။ Weekend တွေ holiday တွေ မရှိသည့် နေ့ရက်တွေလည်း ရှိနိုင်တယ်။ အရင်ကလို အချိန်တန် လခ လည်း မ ရသလို အလုပ်တွေ လည်း ပိုရှုပ်လာပြီး နားရက် မရှိဘူး။ ရသမျှတွေကလည်း ဝန်ထမ်း လခ company က သုံးစရိတ်တွေ နဲ့ပဲ လည်ပတ်ပြီး ကိုယ့် အတွက် လခ မရ သလောက် ရှိနိုင်ပါတယ်။ တစ်ခါတစ်လေ လုပ်နေတာတွေ မှန်ပါ့မလား ဆိုပြီး တွေဝေသည့် အချိန်တွေလည်း ရှိလာနိုင်ပါ တယ်။

ကိုယ်ပိုင် company တည်ထောင် လုပ်သည့် အတွက်ကြောင့် ချက်ချင်း ချမ်းသာသွားတာ မဟုတ် ပဲ အခြေကျအောင် အနည်းဆုံး ၅ နှစ်လောက် အချိန်ပေးရတယ်။ အချို့ company တွဆိုရင် ၁၀ နှစ်ကြာမှ profit များများ ရလာတာတွေလည်း ရှိတတ်တယ်။

စပြီးသည့် အခါမှာ မလုပ်ချင်တော့လို့ ရပ်လိုက်ဖို့ကလည်း ခက်ခဲပါတယ်။ ခန့်ထားသည့် ဝန်ထမ်းတွေ ကုန်သွားသည့် စရိတ်တွေ ရုံးငှား စရိတ်တွေကို ပြန်ကြည့်လိုက်ပြီး ရှေ့ဆက်ဖို့ အားတင်းထားရမယ့် နေ့ရက်တွေ အများကြီး ကြုံလာရနိုင်ပါတယ်။

နိဂုံးချုပ်

အခု ရေးသားသည့် စာအုပ်ကို နာမည်မျိုးစုံ ပေးရင်း နောက်ဆုံး အလုပ်သင် Developer တွေ အတွက် ပို အဆင်ပြေမယ့် စာအုပ် ဖြစ်တာကြောင့် စာအုပ်ကို အလုပ်သင် Developer ဆိုပြီး ပေးခဲ့တာပါ။ ဒီစာအုပ်ဟာ ကျွန်တော့် အတွေ့အကြုံတွေပေါ်မှာ အခြေခံပြီး ရေးသားထားချက်တွေ ဖြစ်လို့ တစ်ယောက် နဲ့ တစ်ယောက် မတူညီနိုင်တာကို သတိပြုစေလိုပါတယ်။ သို့ပေမယ့် အချက်အလက်တွေ ကျွန်တော့် အတွေ့အကြုံတွေက လုပ်ငန်းခွင် ဝင် မည့် developer တွေကို အထောက်အပံ့ ပေးနိုင်မယ်လို့ မျှော်လင့်ပါတယ်။

စာအုပ်ဟာ junior တွေ အတွက် ရည်ရွယ်ပြီး ရေးထားသည့် အတွက် information overload မ ဖြစ်အောင် တတ်နိုင်သမျှ လျော့ချထားပါတယ်။

Developer တစ်ယောက်ဖြစ်လာပြီ ဆိုရင် လေ့လာဖို့ က မပြီးနိုင်တော့ပါဘူး။ အမြဲပြောင်းလဲ နေပါတယ်။ ရှေ့ဆက်ပြီးတော့

- AWS
- Serverless
- Kubernetes
- Machine Learning
- NoSQL Database (Mongodb)
- Web3

စသည်တို့ကို လေ့လာစေချင်ပါတယ်။

စာအုပ်တွေ အနေနဲ့

- Clean Code: A Handbook of Agile Software Craftsmanship
- Clean Architecture: A Craftsman's Guide to Software Structure and Design
- The Pragmatic Programmer: Your Journey To Mastery

စသည့် စာအုပ်တွေ ကို ဖတ်သင့်ပါတယ်။

Programming Algorithm တွေ ကို လေ့လာချင်ရင် လေ့ကျင့်ချင်ရင် https://leetcode.com/ မှာ လေ့ကျင့်နိုင်ပါတယ်။

ကိုးကား

- All the images are using from Undraw.
- Use containers to Build, Share and Run your applications (https://www.docker.com/resources/what-container)
- https://devops-myanmar.gitbook.io/docker-quick-start/
- https://docs.gitlab.com/ee/ci/yaml/
- SOLID principles in PHP By Michał Romańczuk
- What is SDLC
- Clean Architecture and MVVM on iOS | by Oleh Kudinov | OLX Engineering

စာရေးသူ

စာရေးသူ အမည်ရင်းမှာ ထိန်လင်းရွှေ ဖြစ်ပြီး saturngod အမည်ဖြစ် blog , twitter တို့တွင် စာရေးသားလေ့ ရှိပါတယ်။

https://blog.saturngod.net တွင် စာရေးသူ၏ စာများကို ဖတ်ရှုနိုင်ပြီး
https://twitter.com/saturngod တွင် စာရေးသူအား follow လုပ်ထားနိုင်ပါသည်။ Discord Channel
https://discord.gg/KUH3Bkmsna တွင် သိချင်တာများ မေးမြန်းနိုင်ပါတယ်။