MVC with Laravel

MVC with Laravel

Professional Web Developer (တတိယအကြိမ်) မှ ကောက်နှုတ်ဖော်ပြသည်။

M-V-C (Model-View-Controller)

MVC လို့ အတိုကောက်ခေါ်ကြတဲ့ Model-View-Controller ဟာ Application Project မှာပါဝင်တဲ့ Code တွေ ကို လုပ်ဆောင်ချက် ကဏ္ဍအလိုက် ခွဲခြားစုစည်းရာမှာအသုံးပြုတဲ့ Pattern တစ်ခုဖြစ်ပါတယ်။ MVC Pattern က Code တွေ ကို Model, View နဲ့ Controller ဆိုပြီး အုပ်စုသုံးစု ခွဲခြားစုစည်းဖို့အကြံပြုထားပါတယ်။ အကြံပြုတယ်လို့ သုံးရခြင်းက MVC ဟာ သဘောတရားတစ်ခုသာဖြစ်လို့၊ သတ်မှတ်ချက်အတိုင်း တိတိကျကျလိုက်နာခြင်း မလိုက်နာခြင်းက ကျွန်တော် တို့ရေးသားသူအပေါ် မူတည်နေတဲ့အတွက် ဖြစ်ပါတယ်။ တစ်ချို့လည်း တည်ဆောက်နေတဲ့ Software ရဲ့သဘာဝလိုအပ် ချက်အရ မူလ MVC သတ်မှတ် ချက်များကို သင့်တော်သလိုပြုပြင်ပြီးမှ အသုံးချကြတာတွေလည်းရှိပါတယ်။

MVC ဟာ ၁၉၇၀ ကာလများကတည်းက Smalltalk Programming Language နဲ့အတူ စတင်အသုံးပြုလာခဲ့တဲ့ Pattern တစ်ခု ဖြစ်ပါတယ်။ General Purpose Pattern တစ်ခုဖြစ်ပြီး Software အမျိုးအစား အားလုံးအတွက် အသုံး ပြုနိုင်ပါတယ်။ ဒါပေမယ့် Web Application Development အတွက် ပိုပြီးအသုံးများပါတယ်။

MVC Workflow

MVC ရဲ့အဓိကရည်ရွယ်ချက်က Model နဲ့ View ကိုခွဲခြားထားလိုခြင်းဖြစ်ပါတယ်။ Application ရဲ့ Domain Logic နဲ့ အဓိက လုပ်ဆောင်ချက်တွေကို **Model** အနေနဲ့စုစည်းရေးသားထားရမှာ ဖြစ်ပါတယ်။ Model ဟာ Application ရဲ့ ဦးနှောက်လို့လည်း ဆိုနိုင် ပါတယ်။ ပေးလာတဲ့ Input တန်ဖိုးကို လက်ခံပြီးဆောင်ရွက်ရန်ရှိတဲ့ လုပ်ငန်းတွေကို Model က ဆောင်ရွက်ပါတယ်။ ပြီးရင် ရရှိလာ တဲ့ ရလဒ်ကိုပြန်ထုတ်ပေးနိုင်ပါတယ်။ ဒါပေမယ့် အဲဒီရလဒ်များကို ဘယ်လိုပုံစံ၊ ဘယ်လိုအပြင်အဆင်မျိုးနဲ့ ဖော်ပြရမလဲဆိုတဲ့ကိစ္စကို Model က ဆောင်ရွက်မပေးပါဘူး။ ဒါက View ရဲ့အလုပ်ဖြစ်ပါ တယ်။

View ကတော့ Model နဲ့ဆန့်ကျင်ဘက်ဖြစ်ပါတယ်။ လုပ်ငန်းပိုင်းကိုစိတ်မဝင်စားပဲ ပေးလာတဲ့အချက်အလက် များကိုသုံး ပြီး သတ်မှတ်ထားတဲ့ Presentation ပုံစံနဲ့ ဖော်ပြပေးတဲ့အလုပ်ကိုသာ ဆောင်ရွက်ပေးမှာဖြစ်ပါတယ်။ ဒီတော့ တစ်ကယ့် Logic က Model ထဲမှာရှိနေပြီး ဖော်ပြရမယ့် Presentation ပိုင်းကိစ္စတွေက View ထဲမှာ သီးခြားရှိနေစေမှာ ဖြစ်ပါတယ်။ Decouple လုပ် လိုက်ခြင်းတစ်မျိုးဖြစ်ပါတယ်။ ဒီလိုမျိုး Model နဲ့ View တို့က သူ့ အလုပ်ကိုသူ သီးခြားစီဆောင် ရွက်နေတဲ့အခါ Maintenance အတွက်အဆင်ပြေသွားစေမှာဖြစ်ပါတယ်။ ဖော်ပြပုံ အပြင်အဆင်ပိုင်းသက်သက်ပြုပြင် လိုရင် View ကို ပြင်နိုင်ပါတယ်။ ဒီလိုပြင်လိုက်လို့ Logic ပိုင်းကိုထိခိုက်သွားမှာပူစရာမလိုတော့ပါဘူး။ အလားတူပဲ Logic ပိုင်း အလုပ်လုပ်ပုံကို ပြုပြင်လိုတာဆိုရင် Model ကိုပြင်နိုင် ပါတယ်။ ဖော်ပြပုံ Presentation ကိုထိခိုက်မှာ ပူစရာမလို တော့ပါဘူး။

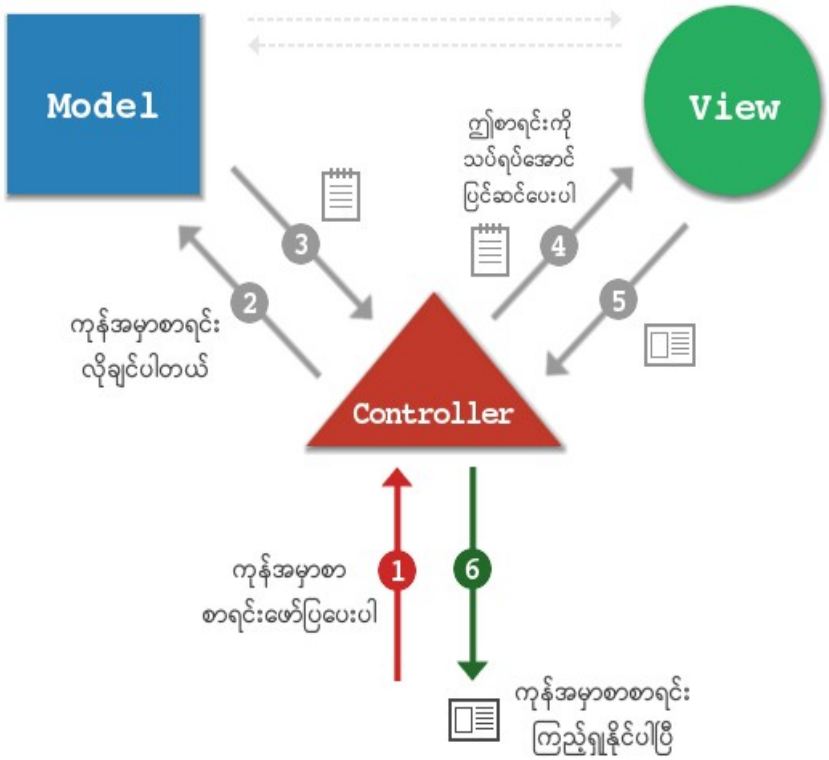
အဓိကလိုရင်းက View နဲ့ Model ကိုခွဲခြားလိုခြင်းဖြစ်ပေမယ့် လက်တွေ့မှာဒီနှစ်ခုကို ပေါင်းကူးဆက်စပ်ပေးနိုင်မယ့် အစိတ်အပိုင်းတစ် ခု လိုအပ်လာပါတယ်။ ဒီတာဝန်ကို **Controller** က ယူပေးပါတယ်။ Controller က ကြားပွဲစား ဆိုကြပါစို့။ သူကိုယ်တိုင်က တစ်ကယ့် လုပ်ဆောင်ချက်တွေ ဘာမှမလုပ်ပေမယ့် အသုံးပြုသူနဲ့တိုက်ရိုက်ဆက်ဆံရတာက Controller ဖြစ်ပါတယ်။ အသုံးပြုသူ ပြုလုပ်တဲ့ Action ကိုကြည့်ပြီးတော့ လိုအပ်တဲ့လုပ်ဆောင်ချက်တွေကို Model ကို ဆောင်ရွက်စေပါတယ်။ Model ပြန်ပေးလာတဲ့ ရလဒ်ကို၊ View ထံ လက်ဆင့်ကမ်းပေးပို့ပြီး သင့်တော်တဲ့ ဖော်ပြမှုပုံစံနဲ့ ဖော်ပြ ပေးဖို့ တောင်းဆိုပေးပါတယ်။ View ကပြန်လည် ပေးပို့လာတဲ့ အပြီးသတ်ပြင် ဆင်ပြီးရလဒ်ကို အသုံးပြုသူထံ ပြန်ပေးနိုင် ပါတယ်။

တနည်းပြောရရင် Controller က Input များကိုလက်ခံစီမံပြီး Output များပြန်လည်ပေးပို့ခြင်းအပိုင်းကို Model နဲ့ View ကြားထဲ ကနေဝင်ပြီး ဆောင်ရွက်ပေးသွားတဲ့သဘောဖြစ်ပါတယ်။ Web Application တွေမှာတော့ Input ဆိုတာ Request များဖြစ်ပြီး၊

Output ဆိုတာ Respond များပဲဖြစ်ပါတယ်။ ဒီတော့ Web Application တွေမှာ Controller က Request နဲ့ Respond စီမံခြင်းကဏ္ဍကိုတာဝန်ယူသူဖြစ်တယ်လို့ ဆိုနိုင်ပါတယ်။

MVC ဟာ နည်းစနစ်တစ်ခုမျှသာဖြစ်ပါတယ်။ အတိအကျလိုက်နာလိုက လိုက်နာနိုင်ပြီး မလိုက်နာလိုရင် မလိုက်နာပဲလည်း နေနိုင်ပါတယ်။ ဒါပေမယ့် သူ့ရဲ့အားသာချက်ကို အပြည့်အဝရရှိဖို့ဆိုရင် Model, View နဲ့ Controller တို့ရဲ့ သက်ဆိုင်ရာ တာဝန်တွေကို မှန်ကန်အောင် သတ်မှတ်ဖို့လိုအပ်ပါတယ်။ View ထဲမှာ Logic နဲ့ပက်သက်တဲ့ Code တွေရေးထားရင် Program ကတော့ အလုပ်လုပ်နေမှာပါပဲ။ အလားတူပဲ Controller ထဲမှာ Logic တွေ ရော ရေးထားမယ်၊ Model ထဲမှာ ဖော်ပြရမယ့်ပုံစံကို သတ်မှတ်ထားမိမယ်ဆိုရင်လည်း Program က အလုပ်လုပ်နေမှာပါပဲ။ ဒါပေမယ့် ကျွန်တော်တို့ရဲ့ Logic နဲ့ Presentation ကိုခွဲထားလိုခြင်းရည်ရွယ်ချက်ကို ထိခိုက်နေပါပြီ။ MVC လို့ပြောနေပေမယ့် MVC ရဲ့ Pattern ပျက်နေလို့ ထိရောက်သင့်သလောက် ထိရောက်မှာ မဟုတ်တော့ပါဘူး။

MVC ရဲ့ အခြေခံအလုပ်လုပ်ပုံကို ပိုမိုမြင်သာအောင် အောက်ပါပုံမှာ လေ့လာနိုင်ပါတယ်။



ပုံမှာ Order List ဖော်ပြခြင်းလုပ်ငန်းစဉ်ကို ဥပမာအနေနဲ့ဖော်ပြထားပါတယ်။ Order List ရယူလိုကြောင်း Request ဝင်ရောက်လာတဲ့အခါ Controller ကလက်ခံပြီး Model ထံကနေ အမှာစာစာရင်းကိုတောင်းယူပါတယ်။ ရရှိလာတဲ့ Order List ကို View ထံလက်ဆင့်ကမ်းပေးလိုက်ပါတယ်။ View က Format လုပ်ထားတဲ့ Layout တစ်ခုအနေနဲ့ Order List ကို ပြန်လည်ပေးပို့တော့မှ အဲဒီ Format လုပ်ထားပြီးရလဒ်ကို Respond အနေနဲ့ပြန် လည်ပေးပို့လိုက်ခြင်းဖြစ်ပါတယ်။

MVC with Laravel

ဆက်လက်ပြီး Laravel PHP Framework ကို အသုံးပြုပြီး MVC ပုံစံ ရေးသားပုံကို ဖော်ပြသွားမှာ ဖြစ်ပါတယ်။ Laravel ဟာ လက်ရှိအချိန်မှာတော့ လူသုံးအများဆုံး PHP Framework တွေထဲကတစ်ခုဖြစ်ပါတယ်။ ဒီစာရေးသားနေချိန်မှာ နောက်ဆုံးထွက်ရှိထားတဲ့ Version ကတော့ Laravel 5.1 ဖြစ်ပါတယ်။

Laravel ဟာ သူ့ရဲ့ သပ်ရပ်ရှင်းလင်းပြီး ဖွဲ့စည်းပုံစနစ်ကျတဲ့ Code ပေါ်မှာ အလွန်ဂုဏ်ယူပြီး၊ Laravel ကို အများ နှစ်သက်ရခြင်း

အဓိကအကြောင်းရင်းကလည်း၊ အဲ့ဒီလို သပ်ရပ်ရှင်းလင်းပြီး ဖွဲ့စည်းပုံ စနစ်ကျတဲ့ Code ကြောင့်ပဲ ဖြစ်ပါတယ်။ PHP ဟာ Language Design ပိုင်း အားနည်းမှုတွေရှိတယ်လို့ ဝေဖန်ခံရလေ့ရှိတဲ့ Language တစ်ခုဖြစ်ပါတယ်။ အခုနောက်ပိုင်းမှာတော့၊ PHP က Language Design အားနည်းတယ်လို့ပြောချင်ရင် Laravel ကို ကြည့်ပြီးမှ ပြောပါလို့ ဆိုရလောက်အောင် သူ့ရဲ့ စနစ်ကျသပ်ရပ်မှုက စံပြဖြစ်လာခဲ့ပါတယ်။

Laravel ရဲ့ အခြား ထူးခြားအသုံးဝင်တဲ့ အချက်တွေကတော့၊ နောက်ပိုင်း PHP Version တွေမှာ ပါဝင်လာတဲ့ Namespace, PHP Development Server စတဲ့ လုပ်ဆောင်ချက်တွေကို ထိရောက်အောင်အသုံးချထားခြင်း၊ Composer လို့ခေါ်တဲ့ အများနှစ်သက်တဲ့ Package Management စနစ်တစ်ခု အသုံးချထားခြင်း၊ Database ဆိုင်ရာ လုပ်ငန်းတွေ စနစ်ကျ လွယ်ကူစေဖို့အတွက် Eloquent လို့ခေါ်တဲ့ ORM (Object Relational Mapping) နည်းပညာကို အသုံးချထားခြင်း၊ Blade လို့ ခေါ်တဲ့ Template Engine တစ်ခုကို အသုံးပြုထားခြင်း၊ Authentication စနစ် တစ်ခါတည်း ပါဝင်ခြင်း၊ Development Environment တည်ဆောက်ခြင်းနဲ့ Deployment လွယ်ကူစေဖို့အတွက် Homestead လို့ ခေါ်တဲ့ Virtualization အခြေပြု နည်းပညာ ဖန်တီးပေးထားခြင်း၊ စသဖြင့် ထူးခြားချက်ပေါင်းများစွာ ပါဝင်ပါတယ်။ ဒီအကြောင်း အရာအားလုံးကိုတော့ ဒီနေရာမှာ ဖော်ပြနိုင်မှာ မဟုတ်ပါဘူး။ ဒီနေရာမှာ Laravel ကို အသုံးပြုပြီး MVC Code တွေ ဘယ်လိုရေးနိုင်သလဲဆိုတဲ့ အပိုင်းကိုသာ ဖော်ပြပေးသွားမှာဖြစ်ပါတယ်။ စတင်လေ့လာသူတစ်ဦးအနေနဲ့ ချက်ခြင်း အလုပ် ဖြစ်စေဖို့အတွက် လိုအပ်တဲ့အပိုင်းတွေကို ရွေးထုတ်ဖော်ပြပြီး ကျန်နည်းပညာတွေကိုတော့ ကိုယ်တိုင် ဆက်လက်လေ့လာသွားနိုင်စေဖို့ ရည်ရွယ်ခြင်းဖြစ်ပါတယ်။

14.1 – Installing Laravel

Laravel Framework ကို အသုံးပြုနိုင်ဖို့အတွက် PHP 5.5.9 နဲ့အထက်ရှိဖို့ လိုအပ်ပါတယ်။ PDO, Mbstring, Tokenizar စတဲ့ PHP Extension တွေလည်း Requirement အနေနဲ့ လိုအပ်ပေမယ့် XAMPP မှာ အဲ့ဒီ Package တွေက ပါဝင်ပြီးဖြစ်ပါတယ်။ ဒါကြောင့် XAMPP နောက်ဆုံး Version ကို ရယူထည့်သွင်းလိုက်ရင် Laravel အတွက် လိုအပ်တဲ့ Environment ပြည့်စုံပြီလို့ ဆိုနိုင်ပါတယ်။ XAMPP နောက်ဆုံး Version ကို အောက်ပါလိပ်စာမှာ Download ရယူနိုင်ပါတယ်။

http://apachefriends.org

Laravel ကို Install လုပ်ဖို့အတွက် နည်းလမ်းမှန်ကတော့ Composer လို့ခေါ်တဲ့ Package Manager ကနေတစ်ဆင့် Install လုပ်ခြင်းပဲ ဖြစ်ပါတယ်။ ဒါပေမယ့် ဒီနေရာမှာ Composer အကြောင်း အကျယ်မချဲ့ချင်တဲ့အတွက် တိုက်ရိုက်ပဲ Download ရယူတဲ့နည်းလမ်းကိုပဲ အသုံးပြုသွားမှာ ဖြစ်ပါတယ်။ ဒါဟာ နည်းလမ်းမှန် မဟုတ်ပေမယ့်၊ ကျွန်တော်တို့လို မြန်နှုန်းမြင့် အင်တာနက်မရှိသူတွေအတွက် ပိုလွယ်ပါတယ်။ အင်တာနက်မရှိလဲ ဖိုင်ကို ကူးယူအသုံးပြုနိုင်တဲ့အတွက် ဖြစ်ပါတယ်။ Composer ကနေ Install လုပ်ရင်တော့ အင်တာနက် အဆက်အသွယ်လိုသလို သင့်တင့်တဲ့ မြန်နှုန်းနဲ့ တည်ငြိမ်မှု လည်းလိုနိုင်ပါတယ်။ အင်တာနက်မငြိမ်လို့ လိုအပ်တဲ့ Package တွေ မစုံရင် အခက်အခဲတွေနဲ့ ကြုံတွေ့ရမှာ ဖြစ်ပါတယ်။

Laravel နဲ့ ဆက်စပ်လိုအပ်တဲ့ Package အားလုံးကို စုစည်းပေးထားတဲ့ Larapack လို့ခေါ်တဲ့ Project တစ်ခုရှိပါတယ်။ အဲ့ဒီ Larapack ကို အောက်ပါလိပ်စာမှာ Download ရယူနိုင်ပါတယ်။

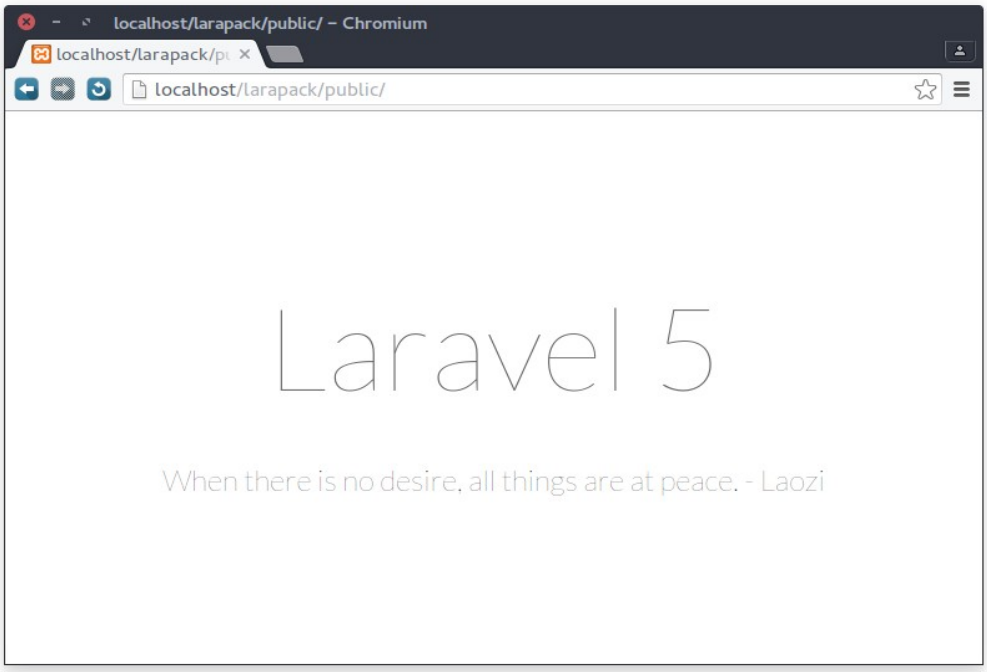
http://fian.my.id/larapack/

Download ရရှိလာတဲ့ Zip ဖိုင်ကိုဖြည်ကြည့်လိုက်ရင် `larapack` ဆိုတဲ့ Directory တစ်ခုထဲမှာ အခုလို ဖိုင်နဲ့ Directory တွေ ပါဝင်လာတာကို တွေ့ရမှာ ဖြစ်ပါတယ်။

```
larapack
├── app
│   ├── Commands
│   ├── Http
│   ├── Models
│   ├── Services
│   └── User.php
├── artisan
├── bootstrap
├── composer.phar
├── config
│   ├── app.php
│   ├── auth.php
```

```
│   ├── database.php
│   ├── session.php
│   └── view.php
├── database
├── public
│   ├── css
│   └── index.php
├── resources
│   └── views
├── server.php
├── storage
├── tests
└── vendor
    ├── autoload.php
    ├── composer
    ├── doctrine
    ├── laravel
    ├── monolog
    ├── phpdocumentor
    ├── phpspec
    ├── phpunit
    ├── swiftmailer
    └── symfony
```

ပါတဲ့ဖိုင်တွေအရမ်းများလို့ လျှော့ပြီး ပြထားတာပါ။ လက်တွေ့ကြည့်လိုက်ရင် ဒီထက်ပို များပါလိမ့်ဦးမယ်။ ရရှိလာတဲ့ `larapack` Directory ကို XAMPP Document Root (`htdocs`) အောက်မှာ ထားလိုက်ပြီး Laravel Framework ကို စတင်အသုံးပြုဖို့ အတွက် အသင့်ဖြစ်ပြီဖြစ်ပါတယ်။ Web Browser ဖွင့်ပြီး **`localhost/larapack/public`** လို့ ထည့်သွင်းကြည့်လိုက်ရင် အခုလို တွေ့ရမှာဖြစ်ပါတယ်။



Laravel စတင် အလုပ်လုပ်နေပြီဆိုတဲ့သဘော ဖြစ်ပါတယ်။ URL မှာ `public` Directory ကို ညွှန်းထားတာကို သတိပြုပါ။ ကျန်ဖိုင် နဲ့ Folder တွေက App Source Code နဲ့ လိုအပ်တဲ့ ဆက်စပ် Package တွေဖြစ်ပြီး၊ နောက်ဆုံးရလဒ်ကိုတော့ `public` Directory ကနေ ရရှိမှာဖြစ်ပါတယ်။

ရှေ့ဆက်မသွားခင် သတိပြုသင့်တဲ့ ဖိုင်နဲ့ Directory တွေအကြောင်းကို အရင်လေ့လာချင်ပါတယ်။

/app	ကိုယ့် Application ရဲ့ Source Code အများစုဟာ ဒီ Directory ထဲမှာ ရေးသားရမှာဖြစ်ပါတယ်။ app အတွင်းမှာ Console, Events, Http စတဲ့ အခြေခံ လုပ်ဆောင်ချက်တွေကို ကြိုတင်ရေးသားပေးထားတဲ့ Directory တွေ ပါဝင်ပါသေးတယ်။ တည်ဆောက်မယ့် Application ရဲ့ Model Code တွေကို ဒီ Directory အတွင်းမှာ သက်ဆိုင်ရာ Directory တွေ ထပ်ခွဲပြီး ထည့်သွင်းရေးသားနိုင်ပါတယ်။
/app/Http/Controllers	MVC မှာ Controller ရဲ့ တာဝန်ဟာ Requests/Responses တွေ စီမံဖြစ်ဖြစ်ကြောင်း ဖော်ပြခဲ့ပြီး ဖြစ်ပါတယ်။ Controller Code တွေကို ဒီ Directory ထဲမှာ စုစည်း ရေးသားရမှာ ဖြစ်ပါတယ်။
/app/Http/routes.php	URI လမ်းကြောင်းတွေ စီမံတဲ့လုပ်ငန်းကို ဒီဖိုင်မှာ သတ်မှတ်ရမှာ ဖြစ်ပါတယ်။ Controller နဲ့ တွဲဖက်အလုပ်လုပ်မှာ ဖြစ်ပါတယ်။ ဘယ် URI ကို လက်ခံရရှိရင် ဘယ် Controller နဲ့ Action ကို အလုပ်လုပ်ပေးရမလဲ ဆိုတဲ့ လမ်းကြောင်းတွေကို သတ်မှတ်ပေးရမှာ ဖြစ်ပါတယ်။
/config	Framework Setting တွေကို ဖိုင်အမည်အလိုက် ဒီ Directory ထဲမှာ စုစည်းပေးထားပါတယ်။
/config/app.php	URL, Timezone, Locale စတဲ့ အခြေခံ Setting တွေကို ဒီဖိုင်ထဲမှာ သတ်မှတ်ထားခြင်း ဖြစ်ပါတယ်။ လိုအပ်သလို ပြင်ဆင်သတ်မှတ်နိုင်ပါတယ်။
/config/database.php	အသုံးပြုလိုတဲ့ Database အမျိုးအစား၊ Database အမည်၊ Username, Password စတဲ့ Setting တွေကို ဒီဖိုင်ထဲမှာ သတ်မှတ်ထားခြင်း ဖြစ်ပါတယ်။ လိုအပ်သလို ပြင်ဆင်သတ်မှတ်နိုင်ပါတယ်။
/config/auth.php	Laravel နဲ့အတူ တစ်ခါတည်း တွဲဖက်ပါဝင်လာတဲ့ Authentication စနစ်နဲ့ ပက်သက်တဲ့ Setting တွေကို ဒီဖိုင်ထဲမှာ သတ်မှတ်ထားပါတယ်။
/public	Static Resources တွေဖြစ်တဲ့ CSS ဖိုင်တွေ၊ JavaScript ဖိုင်တွေ၊ Font တွေနဲ့ Logo လို Image တွေကို ဒီ Directory ထဲမှာ စုစည်းသတ်မှတ်ထား ရမှာ ဖြစ်ပါတယ်။
/resources	HTML Template, CoffeeScript, LESS, SASS စတဲ့ Pre-processor နည်းပညာတွေနဲ့အတူ တွဲဖက်အသုံးပြုရမယ့် Source Code တွေကို ဒီ Directory ထဲမှာ စုစည်းထားရပါတယ်။
/resources/views	View Template တွေကို Blade လို့ခေါ်တဲ့ Template Engine သုံးပြီး ရေးသားရတဲ့အတွက် View Code တွေကို ဒီ Directory ထဲမှာ စုစည်း ရေးသားပေးရပါတယ်။
/database	SQLite လို Standard Alone Database ဖိုင်တွေနဲ့ Database Structure ကြေငြာသတ်မှတ်ထားတဲ့ Migration ဖိုင်တွေကို ဒီ Directory နဲ့ စုစည်းထားရပါတယ်။
/storage	Framework က Generate လုပ်လိုက်တဲ့ Session Data ဖိုင်တွေ၊ Cache ဖိုင်တွေနဲ့ Log ဖိုင်တွေကို ဒီ Directory ထဲမှာ သိမ်းပါတယ်။ ဒါကြောင့် Linux System မှာဆိုရင် ဒီ Directory ကို Write Permission မှန်အောင် ပေးထားဖို့ လိုအပ်မှာ ဖြစ်ပါတယ်။
/vendor	Laravel Source Code နဲ့ Package ဖိုင်တွေအားလုံးကို ဒီ Directory ထဲမှာ စုစည်းပေးထားမှာ ဖြစ်ပါတယ်။
/tests	Unit Testing လို လုပ်ဆောင်ချက်မျိုးကို အသုံးပြုလိုရင်တော့ Unit Test Code တွေကို ဒီ Directory ထဲမှာ စုစည်းရေးသားနိုင်ပါတယ်။
/bootstrap	Application ကို Run တဲ့အခါ လိုအပ်တဲ့ ဖိုင်တွေကို Include လုပ်ခြင်း Load

	လုပ်ခြင်း စတဲ့ လုပ်ငန်းတွေကို လုပ်ပေးတဲ့ ဖိုင်တွေကို စုစည်းပေးထားတဲ့ Directory ဖြစ်ပါတယ်။
artisan	Laravel ရဲ့ Command Line Tool ဖြစ်ပါတယ်။ Project ကို Run ဖို့၊ Database Migration တွေ Run ဖို့နဲ့ Model ဖိုင်တွေ၊ Controller ဖိုင်တွေ တည်ဆောက်ဖို့အတွက် အသုံးပြုနိုင်ပါတယ်။
.env	Environment Setting တွေကို ဒီ ဖိုင်ထဲမှာ ရေးသားသတ်မှတ် ပေးရပါတယ်။ ဥပမာ – Local မှာ Develop လုပ်နေစဉ် သုံးရမယ့် Database Username, Password ကဘာ၊ Server မှာ Distribute လုပ်လိုက်တဲ့အခါ သုံးရမယ့် Database Username, Password ကဘာ စသဖြင့် Environment အလိုက် ခွဲခြားသတ်မှတ်လိုတဲ့ Setting တွေကို သတ်မှတ်ပေးနိုင်ခြင်း ဖြစ်ပါတယ်။

Laravel Router

Application တစ်ခုကို စတင်တော့မယ်ဆိုရင် ပထမဆုံးလုပ်သင့်တဲ့ အလုပ်ကတော့ URI လမ်းကြောင်းတွေ သတ်မှတ်ခြင်း ပဲဖြစ်ပါ တယ်။ ဒီလုပ်ဆောင်ချက်ကို Routing လို့ခေါ်ပြီး `/app/Http/routes.php` မှာ Routing သတ်မှတ်ချက်တွေကို ရေးသား သတ်မှတ် ပေးရပါတယ်။ ရေးသားပုံကတော့ အခုလို ဖြစ်ပါတယ်။

```
Route::get('/user', function () {
     return 'Hello User';
});
```

`/user` URI နဲ့ Request ကို လက်ခံရရှိရင် ဆောင်ရွက်ရမယ့် Function ကို သတ်မှတ်ပေးထားခြင်း ဖြစ်ပါတယ်။ ဒီနေ ရာမှာ JavaScript Nameless Function လိုမျိုး Nameless Function တစ်ခုကို အသုံးပြုထားတာကို တွေ့ရမှာ ဖြစ်ပါ တယ်။ Nameless Function ဆိုတာ PHP 5.1, 5.2 စတဲ့ အရင် PHP Version တွေမှာ မပါဝင်ပေမယ့် နောင်ပိုင်း PHP Version တွေမှာ ပါဝင်လာတဲ့ လုပ်ဆောင်ချက် ဖြစ်ပါတယ်။ နမူနာအရ `/user` URI နဲ့ Request ကို လက်ခံရရှိရင် Hello User ဆိုတဲ့ Message ကို Response အဖြစ် ပြန်ပေးသွားမှာ ဖြစ်ပါတယ်။

ဒီလို Function တစ်ခု သတ်မှတ်မယ့်အစား Controller နဲ့ Action ကိုလဲ သတ်မှတ်ပေးနိုင်ပါတယ်။ ဥပမာ –

```
Rout::get('/user', 'UserController@index');
```

အဓိပ္ပါယ်ကတော့ `/user` URI နဲ့ Request ကို လက်ခံရရှိရင် `UserController` ရဲ့ `index` Action ကို အလုပ်လုပ်ပေးပါလို့ သတ်မှတ်လိုက်ခြင်းဖြစ်ပါတယ်။ ဒီနေရာမှာ `get()` Function ကို အသုံးပြုထားတဲ့အတွက် Request Method က GET ဖြစ်မှ အလုပ်လုပ်မှာပါ။ အကယ်၍ Request Method POST အတွက် Route ကို သတ်မှတ်လိုရင် တော့ `get()` Function အစား `post()` Function ကို အစားထိုး အသုံးပြုနိုင်ပါတယ်။ အလားတူပဲ အခြား Request Method တွေဖြစ်တဲ့ PUT နဲ့ DELETE တို့ အတွက်လည်း `put()` Function နဲ့ `delete()` Function တို့ကို အသုံးပြုနိုင်ပါ တယ်။ Request Method ကိုမကြည့်ပဲ အလုပ် လုပ်စေလိုရင်တော့ `all()` Function ကို အသုံးပြုနိုင်ပါတယ်။

Variable Parameter တွေကိုလည်း Route URI အတွက် သတ်မှတ်ပေးနိုင်ပါတယ်။ ဥပမာ –

```
Route::post('/user/update/{id}', 'UserController@update');
```

နမူနာအရ `/user/update/123` ဆိုတဲ့ URI ကို လက်ခံရရှိရင် `{id}` Parameter ရဲ့တန်ဖိုးက 123 ဖြစ်တဲ့အတွက် အလုပ် လုပ်တဲ့အခါ `$id` ဆိုတဲ့ Variable တစ်ခုကို Laravel ကအသင့်ကြေငြာပေးထားမှာ ဖြစ်ပြီး `123` ကို တန်ဖိုးအဖြစ် ထည့်သွင်းပေး ထားမှာ ဖြစ်ပါတယ်။ `123` နေရာမှာ အခြားမည်သည့်တန်ဖိုးကိုမဆို အစားထိုးအသုံးပြုနိုင်ပါတယ်။

Laravel Controllers and Views

Laravel မှာ Controller ဖိုင်တွေကို /app/Http/Controllers/ Directory ထဲမှာ ရေးသားပေးရပါတယ်။ Controller.php ဆိုတဲ့ Class တစ်ခုဟာ မူလကတည်းက ရှိနေမှာဖြစ်ပြီး ကျွန်တော်တို့ ဖြည့်စွက်ရေးသားတဲ့ Controller တွေဟာ အဲဒီ Class ကနေ ဆက်ခံရေးသားပေးရမှာဖြစ်ပါတယ်။ ဥပမာ – index နဲ့ update ဆိုတဲ့ Action နှစ်ခု ပါဝင် တဲ့ UserController တစ်ခုကို ရေးသားလိုတဲ့ဆိုရင် UserController.php လို့ ဖိုင်အမည်ကိုပေးရမှာဖြစ်ပြီး ရေးသားရမယ့် Code နမူနာကတော့ အခုလို ဖြစ်မှာဖြစ်ပါတယ်။

```
<?php

namespace App\Http\Controllers;

use App\Http\Controllers\Controller;



class UserController extends Controller
{
      function index() {
            return 'I am Index';
      }

      function update($id) {
            return "I am updating $id";
      }
}
```

namespace အကြောင်းကို ဒီနေရာမှာ အကျယ်ချဲ့ပြီးတော့ မရှင်းတော့ပါဘူး။ လိုရင်းအနေနဲ့ နားလည်ရမှာကတော့ namespace ကို အမျိုးတူရာ Class တွေကို စုစည်းဖို့ သုံးတယ်လို့ နားလည်ရပါမယ်။ Laravel Controller Class အားလုံးဟာ App\Http\Controllers ဆိုတဲ့ namespace အတွင်းမှာ ပါဝင်ရမှာဖြစ်ပါတယ်။ ဒါကြောင့် နမူနာ Code ရဲ့ ထိပ်ဆုံးမှာ namespace App\Http\Controllers လို့ ကြေငြာထည့်သွင်းပေးထားခြင်း ဖြစ်ပါတယ်။

ပြီးတဲ့အခါ ပင်မ Controller Class ကို use Keyword နဲ့ ချိတ်ဆက်ထည့်သွင်းထားပါတယ်။ ပင်မ Controller Class ဟာလည်း App\Http\Controllers Namespace အောက်မှာပဲပါဝင်တဲ့အတွက် use App\Http\Controllers\Controller လို့ ရည်ညွှန်းချိတ်ဆက်ထားခြင်းပဲ ဖြစ်ပါတယ်။

ပြီးတဲ့အခါ ကျွန်တော်တို့ရဲ့ Controller ဖြစ်တဲ့ UserController ကို ကြေငြာတဲ့အခါ extends Keyword နဲ့ ပင်မ Controller ကို Inherit လုပ်ယူကြေငြာထားတာကို တွေ့ရမှာ ဖြစ်ပါတယ်။ ဒါဟာ Controller Class တစ်ခုရေးသားဖို့ အတွက် မဖြစ်မနေ ပါဝင်ရမယ့် အခြေခံ Structure ပဲ ဖြစ်ပါတယ်။

UserController မှာ index နဲ့ update ဆိုတဲ့ Action နှစ်ခုပါဝင်စေလိုတဲ့အတွက် index() နဲ့ update() ဆိုတဲ့ Function နှစ်ခုကို ကြေငြာထားတာကိုလည်း တွေ့ရနိုင်ပါတယ်။ update() Function မှာ $id Variable ကို အသုံးပြုထားပြီး အဲဒီ $id Variable ထဲမှာ URI Route ရဲ့ {id} Parameter နေရာမှာ ပါဝင်လာတဲ့တန်ဖိုး အသင့်ရှိနေမှာပဲ ဖြစ်ပါတယ်။

လက်ရှိ index နဲ့ update Action တွေက ရိုးရိုး Text Message တွေကို Response ပြန်ပေးနေခြင်း ဖြစ်ပါတယ်။ အဲဒီလို Text Message တွေကို Response မပြန်ပဲ HTML Template တွေကို Response ပြန်လိုရင်တော့ View ကို အသုံးပြုရမှာ ဖြစ်ပါတယ်။

View Template တွေကို Blade Template Engine အသုံးပြုရေးသားပြီး /resources/views/ Directory အောက်မှာ သိမ်းပေးရပါတယ်။ Blade Template Engine ကို မသုံးတက်ရင်လည်း ရိုးရိုး HTML နဲ့ PHP ကိုပဲ အသုံးပြုနိုင်ပါတယ်။ ဥပမာ – /resources/views/ Directory အောက်မှာ users ဆိုတဲ့ Directory တစ်ခုဆောက်ပြီး hello.blade.php ဆိုတဲ့ Template တစ်ခုကို အခုလို ရေးသားတည်ဆောက်ထားနိုင်ပါတယ်။

```
<!DOCTYPE html>
<html lang="en">
<head>
      <title>Laravel Example</title>
</head>
<body>
      <h1>Index</h1>
      <p>Hello World!</p>
</body>
</html>
```

ရိုးရိုး HTML Code တွေသာ ဖြစ်တယ်ဆိုတာကို တွေ့နိုင်ပါတယ်။ အဲ့ဒီ Template ကို UserController ရဲ့ `index` Action ကနေ ရယူပြီး Response ပြန်လိုတဲ့အခါ အခုလို ရေးသားနိုင်ပါတယ်။

```
function index() {
      return view("user/hello");
}
```

`view()` Function ကို Response ပြန်ပေးလိုက်ခြင်းဖြစ်ပြီး အသုံးပြုရမယ့် Template အနေနဲ့ `user` Directory ထဲက `hello.blade.php` ကို အသုံးပြုပါလို့ ပြောလိုက်ခြင်း ဖြစ်ပါတယ်။ `view()` Function ထဲမှာ File Extension ဖြစ်တဲ့ `blade.php` ကို ထည့်ပေးစရာ မလိုပါဘူး။

View Template ထဲမှာ အသုံးပြုရမယ့် Data တွေကိုလည်း Template ကို `view()` Function နဲ့ ခေါ်ယူစဉ်မှာ ထည့်သွင်း ပေးနိုင်ပါသေးတယ်။ ဥပမာ – `index` Action ကို အခုလို ပြင်ဆင်ရေးသားလိုက်တယ် ဆိုပါစို့။

```
function index() {
      return view("user/hello", array(
            "name" => "John Doe"
      ));
}
```

Array တစ်ခုကို `view()` Function ရဲ့ ဒုတိယ Parameter အဖြစ် ထည့်သွင်းပေးလိုက်ခြင်းဖြစ်ပါတယ်။ Array Index `name` ကို Template ထဲမှာ `$name` ဆိုတဲ့ Variable အဖြစ် အသင့်သုံးနိုင်အောင် Laravel က စီစဉ်ပေးထားမှာဖြစ်လို့ Template ကို အခုလို ပြင်ဆင်ရေးသားနိုင်ပါတယ်။

```
<!DOCTYPE html>
<html lang="en">
<head>
      <title>Laravel Example</title>
</head>
<body>
      <h1>Index</h1>
      <p>Hello <?= $name ?>!</p>
</body>
</html>
```

`$name` Variable ကို PHP နဲ့ ရိုက်ထုတ် အသုံးပြုထားခြင်းပဲ ဖြစ်ပါတယ်။

Master View

လက်တွေ့မှာ View တိုင်းအတွက် Template အပြည့်အစုံလိုက်ရေးပေးနေရရင် သဘာဝမကျပါဘူး။ မည်သည့် App မဆို Main Content ကသာ ပြောင်းလဲလေ့ရှိပြီး Menu, Logo, Title, Footer စတဲ့ Component တွေက View တစ်ခုနဲ့ တစ်ခု တူညီကြလေ့ရှိပါတယ်။ ဒီပြဿနာပြေလည်စေဖို့အတွက် Master View ကို သုံးကြလေ့ရှိပါတယ်။

ပထမဦးဆုံးအနေနဲ့ Master View တစ်ခုကို အခုလိုတည်ဆောက်နိုင်ပါတယ်။ ဖိုင်အမည်ကို `master.blade.php` လို့ ပေးထားတယ်ဆိုကြပါစို့။

```
<!DOCTYPE html>
<html lang="en">
<head>
      <meta charset="UTF-8">
      <title>Master View</title>
</head>
<body>
      <h1>Hello, World!</h1>
      <div class="container">
            @yield("main")
      </div>
</body>
</html>
```

Class ကို `container` လို့သတ်မှတ်ထားတဲ့ `<div>` အတွင်းမှာ `@yield("main")` လို့ ရေးသားထားတာကို သတိပြုရမှာ ဖြစ်ပါတယ်။ ဤနေရာတွင် Main Section ကို ဖော်ပြပါ ဆိုတဲ့ အဓိပ္ပါယ်ပါ။ Master View ကို အခုလို သတ်မှတ်ပြီးပြီဆိုရင် အခြား View တွေကို အခုလို သတ်မှတ်နိုင်ပါတယ်။ ဖိုင်အမည်ကို `index.blade.php` လို့ ပေးထားတယ် ဆိုပါစို့။

```
@extends("master")

@section("main")
<h2>Index</h2>
<p>Content</p>
@stop
```

`@extends("master")` လို့ ကြေငြာထားတဲ့အတွက် `master.blade.php` ဆိုတဲ့ Template ထဲမှာ ဖော်ပြပေးပါဆိုတဲ့

အဓိပ္ပါယ် ဖြစ်သွားပါတယ်။ Master View ရဲ့ ဖိုင်အမည်ကို app.blade.php လို့ ပေးထားမယ်ဆိုရင် @edtends("app") လို့ ရေးသားပေးရမှာ ဖြစ်ပါတယ်။

ဖော်ပြရမယ့် Content တွေကိုတော့ @section("main") နဲ့ @stop အကြားမှာ ရေးသား ပေးရမှာပဲဖြစ်ပါတယ်။ Master View မှာ @yield("main") လို့ သတ်မှတ်ထားလို့ ဒီနေရာမှာလည်း @section("main") လို့ သုံးထားခြင်းဖြစ်ပါတယ်။ အခြားအမည်နဲ့ သုံးလိုကလည်း သုံးနိုင်ပါတယ်။

ဒီလို Master View နဲ့ View ရေးသားထားပြီးပြီဆိုရင် Controller ထဲမှာ ပုံမှန်ရေးရိုးရေးစဉ်အတိုင်း ဆက်လက်ရေးသား နိုင်ပါတယ်။

```
function index() {
     return view("index");
}
```

ဒီလိုရေးသားထားတဲ့အတွက် Response အနေနဲ့ Index View ကို ပြန်ပေးဖို့ ကြိုးစားတဲ့အခါ Index View ဟာ Master View ကို Extend လုပ်ထားကြောင်း Laravel နဲ့ Blade Template Engine တို့ကသိပြီး Master View နဲ့ Index View တို့ကို ပေါင်းစပ်ပြီးမှ Response ပြန်ပေးသွားမှာပဲ ဖြစ်ပါတယ်။

Redirect

လိုအပ်တဲ့လုပ်ငန်းတွေကို ဆောင်ရွက်ပြီးတဲ့အခါ View တစ်ခုကို Response မပြန်ပဲ အခြားလုပ်ဆောင်ချက် (သို့မဟုတ်) Route ထံ Redirect လုပ်တဲ့အလုပ်ဟာလည်း အခြေခံလိုအပ်ချက်ဖြစ်ပါတယ်။ ဒီလုပ်ဆောင်ချက် ရရှိဖို့အတွက် Laravel မှာ redirect() Function ကို အသုံးပြုနိုင်ပါတယ်။ Redirect လုပ်တဲ့အခါ Route တစ်ခုကို ညွှန်းနိုင်သလို၊ Controller တစ်ခုရဲ့ Action ကိုလည်း ညွှန်းနိုင်ပါတယ်။

Route တစ်ခုကို အခုလို Redirect လုပ်နိုင်ပါတယ်။

```
return redirect('user/list');
```

နမူနာအရ user/list ဆိုတဲ့ Route ကို Redirect ပြုလုပ်ပေးသွားမှာ ဖြစ်ပါတယ်။ Action တစ်ခုကို Redirect လုပ်လိုရင်တော့ အခုလို ပြုလုပ်နိုင်ပါတယ်။

```
return redirect('UserController@new');
```

နမူနာအရ UserController ရဲ့ new Action ကို Redirect ပြုလုပ်ပေးသွားမှာပဲ ဖြစ်ပါတယ်။ အဲ့ဒီလို Redirect ပြုလုပ်တဲ့အခါ Error Message တွေ Information Message တွေ တွဲဖက်ထည့်သွင်း ပေးနိုင်ပါတယ်။ ဥပမာ –

```
return redirect('user/profile')->with('info', 'Login Successful');
```

with() Function ကို သုံးပြီး ပေးပို့ထားခြင်းဖြစ်ပါတယ်။ Message အမည်ကို info လို့ ပေးထားပြီး၊ Message Content အနေနဲ့ Login Successful လို့ သတ်မှတ်ပေးထားခြင်း ဖြစ်ပါတယ်။ ဒီ Message ကို View Template ထဲမှာ ဖော်ပြလိုရင်တော့ အခုလို ဖော်ပြနိုင်ပါတယ်။

```
<? if( session('info') ): ?>
<div class="info">
     <?= session('info') ?>
</div>
<? endif; ?>
```

`with()` Function က Message ကို SESSION ထဲမှာ သိမ်းပြီး ထည့်ပေးလိုက်တာဖြစ်လို့၊ အဲဒီ Message ကို ပြန်လည် ရယူလိုတဲ့အခါ `session()` Function ကနေတစ်ဆင့် ရယူရခြင်း ဖြစ်ပါတယ်။ နမူနာအရ `info` ဆိုတဲ့ Message ရှိမရှိ စစ်ပြီး ရှိရင် Class မှာ `info` လို့ သတ်မှတ်ထားတဲ့ `<div>` တစ်ခုနဲ့ ဖော်ပြထားစေခြင်းပဲ ဖြစ်ပါတယ်။ ဒီနည်းနဲ့ လိုအပ်တဲ့ မည်သည့် Message ကို မဆို Redirect နဲ့အတူ တွဲဖက် ပေးပို့နိုင်ခြင်းပဲ ဖြစ်ပါတယ်။

Validation

User Input Data တွေ မှန်ကန်မှုရှိမရှိ စီစစ်တဲ့ Validation လုပ်ဆောင်ချက်ဟာလည်း မည်သည့် App မဆိုလိုအပ်တဲ့ အခြေခံ လုပ်ဆောင်ချက် ဖြစ်ပါတယ်။ Laravel မှာ Validation Class တစ်ခု တစ်ခါတည်း ပါဝင်ပြီး အဲဒီ Class ရဲ့ အကူ အညီနဲ့ Validation လုပ်ငန်းကို ဆောင်ရွက်နိုင်ပါတယ်။ Validation Class ကို စတင်အသုံးပြုနိုင်ဖို့အတွက် `use` Keyword နဲ့ ကြေငြာပေးဖို့ လိုနိုင်ပါတယ်။

```
use Validation;
```

Validation စစ်တယ်ဆိုတာ အခြေခံအားဖြင့် User Input Data ကို စစ်တာဖြစ်တဲ့အတွက် User Input Data ကို လက်ခံနိုင်ပေးနိုင်တဲ့ Request Class ကိုလည်း တွဲဖက်အသုံးပြုဖို့ လိုနိုင်ပါတယ်။

```
use Request;
```

Request Class ရဲ့ `all()` Function ကို အသုံးပြုပြီး Request နဲ့အတူပါဝင်လာတဲ့ Input Data တွေကို ရယူနိုင်ပါတယ်။ ရရှိလာတဲ့ Input Data တွေမှာ –

1. `name` မဖြစ်မနေပါဝင်ရမယ်
2. `email` မဖြစ်မနေပါဝင်ပြီး Email Format မှန်ရမယ်
3. `password` မဖြစ်မနေပါဝင်ပြီး အနည်းဆုံး (၆) လုံးရှိရမယ်
4. `password_again` ပါဝင်ပြီး `password` နဲ့ တူညီရမယ်

– လို့ Validation စစ်ချင်ရင် အခုလို စစ်နိုင်ပါတယ်။

```
$input = Request::all();

$validator = Validator::make($input, array(
      "name" => "required",
      "email" => "required|email",
      "password" => "required|min:6",
      "password_again" => "same:password"
));
```

ပထမဦးဆုံး `Request::all()` နဲ့ Input Data အားလုံးကို ရယူပါတယ်။ ပြီးတဲ့အခါ `Validator::make()` Function ရဲ့ ပထမ Parameter အဖြစ် Input Data ကိုပေးလိုက်ပြီး ဒုတိယ Parameter အဖြစ် Validation Rule တွေကို Array အနေနဲ့ ပေးလိုက်ခြင်း ဖြစ်ပါတယ်။

`name` အတွက် `required` သတ်မှတ်ထားပြီး `email` အတွက် `required` နဲ့ `email` လို့ သတ်မှတ်ထားပါတယ်။ `password` ကို `min:6` ဖြစ်ရမယ်လို့ သတ်မှတ်ထားပြီး `password_again` ကိုတော့ `password` နဲ့ တူညီရမယ်လို့ `same` ကို သုံးပြီး သတ်မှတ်ထားခြင်းဖြစ်ပါတယ်။ ဒီလိုစီစစ်လိုက်တဲ့ ရလဒ် Pass ဖြစ်မဖြစ် သိလိုရင်တော့ `passes()` Function ကို အသုံးပြု စစ်ဆေးနိုင်ပါတယ်။ ဥပမာ –

```
if( $validator->passes() ) {
     redirect ('user/login')->with('info', 'Success');
} else {
     redirect ('user/register')->withErrors( $validator );
}
```

ဒီနေရာမှာ သတိပြုရမှာက Validation Passes မဖြစ်လို့ Redirect လုပ်တဲ့အခါ Validation Error Message တွေကို `withErrors()` Function နဲ့အတူတွဲဖက်ပေးထားခြင်းပဲ ဖြစ်ပါတယ်။ ဒီလိုပေးပို့လိုက်တဲ့ Validation Error တွေဟာ View Template ထဲမှာ `$errors` ဆိုတဲ့ Object အနေနဲ့ တည်ရှိနေမှာဖြစ်ပါတယ်။ ဒါကြောင့် View Template ထဲ Validation Error Message တွေကို ဖော်ပြလိုရင် အခုလို ဖော်ပြနိုင်ပါတယ်။

```
<? if( count($errors) ): ?>
     <ul class='errors'>
          <? foreach( $errors->all() as $err ): ?>
               <li><?= $err ?></li>
          <? endforeach; ?>
     </ul>
<? endif; ?>
```

ပထမဦးဆုံး `$errors` မှာ Message ရှိမရှိ `count()` Function နဲ့ စီစစ်ပြီး ရှိတယ်ဆိုရင် `<ul>` Element တစ်ခုနဲ့ Error Message တွေကို Loop လုပ်ပြီး ဖော်ပြထားခြင်းပဲ ဖြစ်ပါတယ်။ `$errors` Object မှာပါဝင်တဲ့ Error Message အားလုံးကို Array အနေနဲ့ ရရှိဖို့အတွက် `all()` Function ကို အသုံးပြုရပြီး၊ ရရှိလာတဲ့ Array ကို `foreach` နဲ့ Loop လုပ်ထားခြင်း ဖြစ်ပါတယ်။

Validation စစ်ရာမှာ အသုံးပြုနိုင်တဲ့ `required`, `email`, `min`, `same` စတဲ့ Option အပြည့်အစုံကိုတော့ အောက်ပါလိပ်စာမှာ ဆက်လက်လေ့လာနိုင်ပါတယ်။

http://laravel.com/docs/5.1/validation

Authentication

Laravel မှာ User Authentication လုပ်ဆောင်ချက် တစ်ခါတည်း ပါဝင်လာပြီးဖြစ်ပါတယ်။ စတင်အသုံးပြုနိုင်ဖို့အတွက် `use` Keyword နဲ့ Authenticate Class ကို အသုံးပြုမယ့်အကြောင်း ကြေငြာပေးဖို့ လိုအပ်နိုင်ပါတယ်။

```
use Auth;
```

Laravel ရဲ့ Authentication လုပ်ဆောင်ချက်ကို အသုံးပြုနိုင်ဖို့အတွက် လိုအပ်ချက် (၃) ချက် ရှိပါတယ်။

ပထမဦးဆုံးလိုအပ်ချက်ကတော့ `users` ဆိုတဲ့ အမည်နဲ့ Table တစ်ခုရှိဖို့ပဲ ဖြစ်ပါတယ်။ Laravel က `users` Table မှာ သွားရောက်စစ်ဆေးခြင်းအားဖြင့် User တစ်ယောက်ကို Authenticate ပြုလုပ်ပေးသင့် မသင့် ဆုံးဖြတ်မှာဖြစ်ပါတယ်။ `users` Table ရဲ့ ဖွဲ့စည်းပုံကို မိမိ App ရဲ့ လိုအပ်ချက်အတိုင်း တည်ဆောက်နိုင်ပါတယ်။ ဒါပေမယ့် `password` ဆိုတဲ့ Field တစ်ခုနဲ့ `remember_token` ဆိုတဲ့ Field တို့တော့ မဖြစ်မနေ ပါဝင်သင့်ပါတယ်။

ဒုတိယလိုအပ်ချက်ကတော့ `/config/auth.php` မှာ Authentication Driver သတ်မှတ်ပေးရမှာ ဖြစ်ပါတယ်။ `/config/auth.php` ကို ဖွင့်ကြည့်လိုက်ရင် Default အနေနဲ့ အခုလို Setting တစ်ခု ပါဝင်နိုင်ပါတယ်။

```
'driver' => 'eloquent'
```

Eloquent ORM ကို မသုံးပဲ Authentication လုပ်ငန်းကို ဆောင်ရွက်လိုရင် အခုလို ပြင်ဆင်ပေးရမှာ ဖြစ်ပါတယ်။

```
'driver' => 'database'
```

တတိယတစ်ချက်ကတော့ Login Form မှာ CSRF Token တစ်ခု ပါဝင်ရမှာဖြစ်ပါတယ်။ CSRF အကြောင်းကို Professional Web Developer – **အခန်း (၁၉) Web Application Security** မှာ လေ့လာနိုင်ပါတယ်။ Login Form ရဲ့ ဖွဲ့စည်းပုံဟာ အခုလို ဖြစ်သင့်ပါတယ်။

```
<form method="post" action="<?= URL::to('user/login') ?>">
     <input type="hidden" name="_token" value="<?= csrf_token() ?>">

     <input type="email" name="email" required placeholder="Your Email">
     <input type="password" name="password" required placeholder="Password">

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

Form ရဲ့ `action` Attribute အတွက် လိပ်စာသတ်မှတ်တဲ့အခါ Route တစ်ခုကို ညွှန်းချင်တဲ့အတွက် `URL::to()` ကို အသုံးပြုထားတာကို သတိပြုသင့်ပါတယ်။ မဖြစ်မနေ ပါဝင်ရမယ့် အချက်ကတော့ `name` ကို `_token` လို့ သတ်မှတ်ထားတဲ့ Hidden Input တစ်ခုပဲဖြစ်ပါတယ်။ အဲ့ဒီ Hidden Input ရဲ့ `value` အနေနဲ့ `csrf_token()` Function ကပေးတဲ့ တန်ဖိုးကို သတ်မှတ်ပေးရမှာပဲဖြစ်ပါတယ်။

ဒါဟာ Cross-Site Request Forgery လို့ခေါ်တဲ့ လုံခြုံရေးပြဿနာကို ကာကွယ်ပေးတဲ့ နည်းစနစ်ဖြစ်ပြီး Laravel မှာ Login Form သာမက Form အားလုံး ဒီလုပ်ဆောင်ချက်ကို မဖြစ်မနေ အသုံးပြုရမှာ ဖြစ်ပါတယ်။

Login Form ကနေ ပေးပို့လာတဲ့ တန်ဖိုးကို အသုံးပြုပြီး Authenticate လုပ်တဲ့လုပ်ငန်းကို အခုလို ဆောင်ရွက်နိုင်ပါတယ်။

```
$input = Request::all();

if(Auth::attempt( array(
     'email' => $input['email'],
     'password' => $input['password']
))) {
     return redirect('user/profile');
} else {
     return redirect('user/login')->withErrors(
           array('Login failed! Try again.')
     );
}
```

ပထမဦးဆုံးအနေနဲ့ Login Form က ပေးပို့လာတဲ့ Data ကို `Request::all()` နဲ့ရယူပါတယ်။ ပြီးတဲ့အခါ `Auth::attempt()` Function ကိုသုံးပြီး Login Form ကပေးပို့လာတဲ့ Email နဲ့ Password ဟာ `users` Table ထဲမှာ သိမ်းဆည်းထားတဲ့ `email`, `password` နဲ့ ကိုက်ညီခြင်းရှိမရှိ စီစစ်ထားခြင်းဖြစ်ပါတယ်။ ကိုက်ညီတယ်ဆိုရင် Login အောင်မြင်မှာဖြစ်ပါတယ်။ Login အောင်မြင်ပြီး Authenticate ဖြစ်နေခြင်းရှိမရှိကို `Auth::check()` Function နဲ့ စီစစ်နိုင်ပါတယ်။

```
if( !Auth::check() ) {
     return redirect('user/login')->with('info', 'Please login first!');
}
```

နမူနာအရ Authenticate ဖြစ်မနေရင် `user/login` ကို Redirect လုပ်ပေးသွားမှာ ဖြစ်ပါတယ်။ Authenticate ဖြစ်ထားတဲ့ User ရဲ့ အချက်အလက်တွေကို လိုချင်ရင်တော့ `Auth::user()` ကို အသုံးပြု ရယူနိုင်ပါတယ်။

```
$user = Auth::user();
```

Authenticate ဖြစ်ထားတဲ့ User ကို Logout ပြုလုပ်လိုရင်တော့ `Auth::logout()` ကို အသုံးပြုနိုင်ပါတယ်။

Authentication နဲ့အတူ တွဲဖက်မှတ်သားရမှာကတော့ Password Hash ဖြစ်ပါတယ်။ Password တွေကို `users` Table ထဲမှာ သိမ်းတဲ့အခါ Plain Text အနေနဲ့ မသိမ်းသင့်ပဲ Hash ပြုလုပ်ပြီးမှ သိမ်းသင့်ပါတယ်။ အဲ့ဒီလို Hash ပြုလုပ်နိုင်ဖို့ အတွက် Laravel ရဲ့ Hash Class ကို အသုံးပြုနိုင်ပါတယ်။

```
use Hash;
```

`Hash::make()` Function ကို Password တွေ Hash ပြုလုပ်ဖို့ သုံးနိုင်ပါတယ်။

```
$password = 'mypassword';
$hashed_password = Hash::make($password);
```

`Hash::make()` က ပြန်ပေးတဲ့ Hash လုပ်ပြီးသား Password ကိုသာ `users` Table ထဲမှာ သိမ်းဆည်းသင့်ပါတယ်။ `Auth::attempt()` Function နဲ့ Login ဝင်ဖို့ ကြိုးစားတဲ့အခါမှာလည်း Laravel က User ပေးလိုက်တဲ့ Password ကို Hash လုပ်ပြီးမှသာ `users` Table ထဲက Password Hash နဲ့ တိုက်ဆိုင် စစ်ဆေးမှာပဲ ဖြစ်ပါတယ်။ Password Hash ရဲ့ လုံခြုံရေးမှာ အရေးပါပုံကို Professional Web Developer – **အခန်း (၁၉) Web Application Security** မှာ ဆက်လက် လေ့လာနိုင်ပါတယ်။

Database

Laravel မှာ Database နဲ့ပက်သက်တဲ့ လုပ်ငန်းတွေအတွက် Eloquent လို့ခေါ်တဲ့ ORM နည်းပညာတစ်မျိုး ပါဝင်ပါတယ်။ အဲ့ဒီနည်းပညာကို ဒီနေရာမှာ ထည့်သွင်းမဖော်ပြတော့ပဲ၊ ရှေ့ပိုင်းအခန်းတွေမှာ လေ့လာခဲ့ပြီးဖြစ်တဲ့ Database Query တွေကိုသာ ဆက်လက် အသုံးပြုသွားမှာ ဖြစ်ပါတယ်။ Laravel မှာ Database နဲ့ပက်သက်တဲ့ လုပ်ငန်းတွေအတွက် DB ဆိုတဲ့ Class တစ်ခုဟာ တစ်ခါတည်း ပါဝင်လာပြီးဖြစ်ပါတယ်။

```
use DB;

class User
{
    static function create() {
        $name = 'John Doe';
        $email = 'john@example.com';
        $result = DB::insert("INSERT INTO users (name, email)
                              VALUES ('$name', '$email'));

        return $result;
    }
}
```

နမူနာကိုလေ့လာကြည့်ရင် DB Class ကို အသုံးပြုမယ့်အကြောင်း `use` Keyword နဲ့ ကြေငြာထားတာကို တွေ့ရမှာဖြစ် ပါတယ်။ ပြီးတဲ့အခါ `User` အမည်နဲ့ Class တစ်ခုတည်ဆောက်ပြီး `create()` Function ကို ထည့်သွင်းရေးသားထားပါတယ်။ `create()` Function အတွင်းမှာ `DB::insert()` ကို သုံးပြီး INSERT Query တစ်ခုကို Run ထားခြင်းဖြစ်ပါတယ်။

ဒီနေရာမှာ သတိပြုရမှာက၊ `create()` Function ကို `static` Function အဖြစ် ကြေငြာပဲဖြစ်ပါတယ်။ ဒါကြောင့် `create()` Function ကို အသုံးပြုလိုတဲ့အခါ Object တွေတည်ဆောက်ဖို့မလိုပဲ Class Name ဖြစ်တဲ့ `User` ကနေ အခုလို တိုက်ရိုက်အသုံးပြုနိုင်မှာ ဖြစ်ပါတယ်။

```
User::create();
```

Query တွေရေးတဲ့အခါ ပိုပြီးစနစ်ကျတဲ့ ရေးနည်းက အခုလိုဖြစ်မှာပါ။

```
static function create() {
    $name = 'John Doe';
    $email = 'john@example.com';
    $result = DB::insert("INSERT INTO users (name, email) VALUES (?, ?),
        array($name, $email)
    );

    return $result;
}
```

Query Statement ထဲမှာ Data Value တွေဖြစ်တဲ့ `$name` နဲ့ `$email` ကို ထည့်မရေးပဲ `DB::insert()` Function ရဲ့ ဒုတိယ Parameter အဖြစ် ထည့်သွင်းပေးထားခြင်း ဖြစ်ပါတယ်။ ဒီလို Query Statement နဲ့ Data ကို ခွဲခြားရေးသား ထားခြင်း ဟာ လုံခြုံရေးအတွက် ပိုကောင်းတဲ့ နည်းလမ်းဖြစ်ပါတယ်။ ဒီသဘောသဘာဝကို Professional Web Developer – **အခန်း (၁၉) Web Application Security** မှာ ဆက်လက်လေ့လာနိုင်ပါတယ်။ အမှန်တော့ Laravel ကလည်း Database လုပ်ငန်းတွေအတွက် PDO ကို အသုံးပြုထားခြင်းဖြစ် ပါတယ်။

INSERT Query အတွက် `DB::insert()` ကို အသုံးပြုသလို SELECT Query အတွက် `DB::select()` ကို အသုံးပြုနိုင်ပါ တယ်။

```
$result = DB::select('SELECT * FROM users');
```

`DB::select()` က ရရှိလာတဲ့ ရလဒ်ကို Array တစ်ခုအနေနဲ့ ပြန်ပေးမှာဖြစ်ပါတယ်။ UPDATE Query အတွက် `DB:update()` ကို သုံးနိုင်ပြီး DELETE Query အတွက် `DB::delete()` တို့ကို အသုံးပြုနိုင်ပါတယ်။ ဒီ Function တွေ ကတော့ UPDATE နဲ့ DELETE Query တွေ Run လိုက်တဲ့အတွက် သက်ရောက်သွားတဲ့ Record အရေအတွက်ကို ပြန်ပေးမှာဖြစ်ပါ တယ်။

ဒီလို Database နဲ့ဆက်သွယ်အလုပ်လုပ်တဲ့ Class ကို Model အဖြစ် အလုပ်လုပ်စေလိုရင် `/app/` Directory အောက်မှာ ရေးသား သိမ်းဆည်း ပေးရမှာဖြစ်ပြီး `App\Models` Namespace ထဲမှာလည်း ထည့်သွင်းပေးရမှာဖြစ်ပါတယ်။ Code ဖိုင်ရဲ့ အပေါ်ဆုံးမှာ အခုလိုထည့်သွင်းပေးခြင်းအားဖြင့် Class ကို Model Namespace မှာ ထည့်သွင်းနိုင်ပါတယ်။

```
namespace App\Models;
```

User Registration System Example

ဆက်လက်ပြီး လက်တွေ့နမူနာအနေနဲ့ စမ်းသပ်နိုင်စေဖို့အတွက် User Registration System တစ်ခု ရေးသားပုံကိုဖော်ပြ ပေးသွားမှာ ဖြစ်ပါတယ်။ Code နမူနာကို အဓိကဖော်ပြတော့မှာဖြစ်ပြီး Code ရဲ့ အလုပ်လုပ်ပုံ အဓိပ္ပါယ်ကိုတော့ ရှေ့ပိုင်းမှာ ဖော်ပြခဲ့ပြီးဖြစ်တဲ့ အချက်တွေပေါ်အခြေခံပြီး ကိုယ်တိုင်နားလည်အောင် ဖတ်ယူရမှာ ဖြစ်ပါတယ်။

ပထမဦးဆုံးအနေနဲ့ လိုအပ်တဲ့ M-V-C ဖိုင်တွေ တည်ဆောက်ရပါမယ်။ အောက်ပါ Directory Structure ကို လေ့လာ ကြည့်ပါ။

```
larapack
├── app
│   ├── …
│   ├── Http
│   │   ├── Controllers
│   │   │   ├── …
│   │   │   └── UserController.php (new)
│   │   ├── …
│   │   └── routes.php
```

```
│   ├── Models (new)
│   │   └── User.php (new)
│   └── …
├── config
│   ├── auth.php
│   ├── …
│   └── database.php
├── …
├── public
│   ├── css
│   │   └── app.css
│   ├── …
│   └── index.php
├── resources
│   ├── …
│   └── views
│       ├── …
│       ├── template.blade.php (new)
│       └── user (new)
│           ├── login.blade.php (new)
│           ├── profile.blade.php (new)
│           └── register.blade.php (new)
└── …
```

1. `/app/Http/Controllers/` ထဲမှာ `UserController.php` ကို တည်ဆောက်ပေးရပါမယ်။
2. `/app/` ထဲမှာ `Models` အမည်နဲ့ Directory တစ်ခုတည်ဆောက်ပေးရပါမယ်။
3. `/app/Models/` ထဲမှာ `User.php` ကို တည်ဆောက်ပေးရပါမယ်။
4. `/resources/views/` ထဲမှာ `template.blade.php` ဆိုတဲ့ Master View ကို တည်ဆောက်ပေးရပါမယ်။
5. `/resources/views/` ထဲမှာ user ဆိုတဲ့ Directory တစ်ခု တည်ဆောက်ပေးရပါမယ်။
6. `/resources/views/` ထဲမှာ အောက်ပါ View ဖိုင်တွေကို တည်ဆောက်ရပါမယ်။
   - `login.blade.php`
   - `profile.blade.php`
   - `register.blade.php`

လိုအပ်တဲ့ ဖိုင်တွေ တည်ဆောက်ပြီးရင် `/app/Http/routes.php` မှာ ရှိနေတဲ့ Code ကို အောက်ပါ Code နဲ့ အစားထိုးရပါမယ်။

```
<?php

Route::get('/', 'UserController@index');
Route::get('/user/login', 'UserController@index');
Route::post('/user/login', 'UserController@login');

Route::get('/user/register', 'UserController@register');
Route::post('/user/register', 'UserController@create');

Route::get('/user/profile', 'UserController@profile');
Route::post('/user/profile', 'UserController@update');

Route::get('/user/logout', 'UserController@logout');

Route::get('/user/delete/{id}', 'UserController@delete');
```

- ပေးထားတဲ့ Code အရ Root URI အတွက် GET Request တစ်ခု လက်ခံရရှိရင် UserController ရဲ့ `index` Action ကို အလုပ်လုပ်မှာ ဖြစ်ပါတယ်၊
- `/user/login` URI ကို GET Request တစ်ခု လက်ခံရရှိရင်လည်း UserController ရဲ့ `index` Action ကိုပဲ အလုပ်လုပ် မှာ ဖြစ်ပါတယ်။
- `/user/login` URL ကိုပဲ POST Request တစ်ခု လက်ခံရရှိရင်တော့ UserController ရဲ့ `login` Action ကို အလုပ်လုပ်ပေးမှာ ဖြစ်ပါတယ်။

ကျန် Route တွေကိုတော့ ကိုယ်တိုင် ဆက်လက်လေ့လာကြည့်လိုက်ပါ။ ဆက်လက်ပြီး /app/Models/user.php ထဲမှာ အောက်ပါ Code ကို ရေးသားပေးပါ။

```
<?php

namespace App\Models;

use DB;
use Hash;

class User {

      static function insert($input) {
            $data = array(
                  $input['name'],
                  $input['email'],
                  Hash::make($input['password'])
            );

            $result = DB::insert("INSERT INTO users (
                                    name, email, password
                              ) VALUES (?, ?, ?)", $data);

            return $result;
      }

      static function update($input) {
            if($input['password']) {
                  $data = array(
                        $input['name'],
                        $input['email'],
                        Hash::make($input['password']),
                        $input['phone'],
                        $input['address']
                  );

                  $result = DB::update("UPDATE users SET name = ?, email = ?,
                                          password = ?, phone = ?,
                                          address = ?", $data);

            } else {
                  $data = array(
                        $input['name'],
                        $input['email'],
                        $input['phone'],
                        $input['address']
                  );

                  $result = DB::update("UPDATE users SET name = ?, email = ?,
                                          phone = ?, address = ?", $data);
            }

            return $result;
      }

      static function delete($user_id) {
            $result = DB::delete("DELETE FROM users WHERE id = ?",
                                    array($user_id));

            return $result;
      }

}
```

ဒီဖိုင်မှာပါဝင်တဲ့ User Class ကို Model Class ဖြစ်စေဖို့အတွက် App\Models Namespace မှာ ထည့်သွင်းထားပါတယ်။ ပြီးတဲ့အခါ Class ထဲမှာ insert(), update() နဲ့ delete() Function (၃) ခုကို ရေးသားထားပါတယ်။

insert() Function က ပေးလာတဲ့ Input Data ကို users Table ထဲမှာ ထည့်ပေးပါတယ်။ password ကို Hash လုပ်ပြီးမှာ ထည့်သွင်းထားတာကို သတိပြုပါ။ update() Function ကတော့ ပေးလာတဲ့ Data ကို users Table မှာ Update လုပ်ပေးပါတယ်။ Input Data မှာ password ပါ မပါ စီစစ်ပြီး ပါရင် password ကို ထည့်ပြီး Update လုပ်ပေမယ့် မပါရင်တော့ password ကို ချန်ပြီး Update လုပ်ထားခြင်းဖြစ်ပါတယ်။ delete() Function ကတော့ ပေးလာတဲ့ User ID နဲ့ ကိုက်ညီတဲ့ Record ကို users Table ထဲကနေ ပယ်ဖျက်တဲ့အလုပ်ကို လုပ်ထားပါတယ်။ Model တစ်ခုရဲ့ တာဝန်ဖြစ်တဲ့ Data တွေ Create, Read, Update, Delete လုပ်တဲ့ လုပ်ငန်းတွေကို ရေးသားသတ်မှတ်ထားခြင်းဖြစ်ပါတယ်။

ဆက်လက်ပြီး /app/Http/Controllers/UserController.php မှာ အောက်ပါ Code ကို ရေးသားပေးပါ။

```
<?php
namespace App\Http\Controllers;

use App\Http\Requests;
use App\Http\Controllers\Controller;

use Request;
use Validator;
use Auth;

use App\Models\User;

class UserController extends Controller {

	function index() {
		if(Auth::check())
			return redirect("user/profile");

		return view("user/login");
	}

	function login() {
		$input = Request::all();

		if(Auth::attempt([
			'email' => $input['email'],
			'password' => $input['password']
		])) {
			return redirect('user/profile');
		} else {
			return redirect('user/login')
				->withErrors(array('Login failed! Try again.'));
		}
	}

	function register() {
		if(Auth::check())
			return redirect("user/profile");

		return view("user/register");
	}

	function create() {
		$input = Request::all();

		$validator = Validator::make($input, array(
			"name" => "required",
			"email" => "required|email|unique:users",
			"password" => "required|min:6",
			"password_again" => "same:password"
		));

		if($validator->passes()) {
			User::insert($input);
			return redirect('user/login')->with('info', 'Register success!');
		} else {
```

```
                return redirect('user/register')->withErrors($validator);
        }
    }

    function profile() {
        if(!Auth::check())
            return redirect("user/login");

        $user = Auth::user();
        return view('user/profile', array(
            'id' => $user->id,
            'name' => $user->name,
            'email' => $user->email,
            'address' => $user->address,
            'phone' => $user->phone
        ));
    }

    function update() {
        $input = Request::all();
        User::update($input);

        return redirect("user/profile")
                ->with("info", "Profile Updated!");
    }

    function logout() {
        Auth::logout();
        return redirect('user/login');
    }

    function delete($user_id) {
        Auth::logout();
        User::delete($user_id);

        return redirect('user/register');
    }

}
```

ရေးသားထားတဲ့ Class ဟာ Controller Class ဖြစ်စေဖို့အတွက် `App\Http\Controllers` Namespace ထဲကို ထည့်သွင်းထားပါတယ်။ ပြီးတဲ့အခါ User Request တွေကို လက်ခံနိုင်ဖို့အတွက် `App\Http\Requests` နဲ့ ပင်မ Controller ဖြစ်တဲ့ `App\Http\Controller` တို့ကို `use` Keyword နဲ့ ကြေငြာရယူထားပါတယ်။ ဆက်လက်ပြီး `Request`, `Validator`, `Auth` စတဲ့ Class တွေကိုလည်း အသုံးပြုမယ့်အကြောင်း ကြေငြာထားပါတယ်။ နောက်ဆုံးအနေနဲ့ ကျွန်တော်တို့ ကြိုတင်ရေးစားထားတဲ့ User Model ကို အသုံးပြုမယ့်အကြောင်း ကြေငြာထားပါတယ်။

- **index** Action က Authenticate ဖြစ်မဖြစ်စီစစ်ပြီး Authenticate ဖြစ်ပြီးသားဆိုရင် `user/profile` Route ကို Redirect လုပ်ထားပါတယ်။ Authenticate မဖြစ်သေးရင်တော့ `user/login` View ကို ဖော်ပြစေထားပါတယ်။
- **login** Action ကတော့ Login Form က ပေးပို့လာတဲ့ Input Data ကို ယူပြီး `Auth::attempt()` နဲ့ Authenticate လုပ်စေပါတယ်။ Authenticate လုပ်ငန်းအောင်မြင်ရင် `user/profile` Route ကို Redirect လုပ်စေပြီး၊ မအောင်မြင်ရင်တော့ `user/login` Route ကို Error Message နဲ့အတူ Redirect လုပ်စေပါတယ်။
- **register** Action က Authenticate ဖြစ်မဖြစ်စီစစ်ပြီး Authenticate ဖြစ်ပြီးသားဆိုရင် `user/profile` Route ကို Redirect လုပ်ထားပါတယ်။ Authenticate မဖြစ်သေးရင်တော့ `user/register` View ကို ဖော်ပြစေထားပါတယ်။
- **create** Action ကတော့ Register Form က ပေးပို့လာတဲ့ Input Data ကို Validate စစ်ပါတယ်။ `email` Field အတွက် `unique` ကို ထည့်စစ်ထားတာကို သတိပြုပါ။ `unique:users` ရဲ့ အဓိပ္ပါယ်က `users` Table ထဲမှာ အရင်က `email` ရှိပြီးသားလားဆိုတာကို စစ်ခိုင်းလိုက်တဲ့သဘောဖြစ်ပါတယ်။ Validation Pass ဖြစ်ရင် Model ရဲ့ `insert()` Function ကို သုံးပြီး User Data ကို `users` Table ထဲမှာ ထည့်သွင်းစေပါတယ်။ ပြီးတဲ့အခါ `user/login` Route ကို Redirect လုပ်ပါတယ်။ Validation Fail ဖြစ်ရင်တော့ `user/register` Route ကို Error Message နဲ့တစ်ကွ Redirect လုပ်ထားပါတယ်။

- **profile** Action ကတော့ Authenticate ဖြစ်မဖြစ်စီစစ်ပြီး Authenticate မဖြစ်သေးရင် `user/login` Route ကို Redirect လုပ်ထားပါတယ်။ Authenticate ဖြစ်ပြီးသားဆိုရင်တော့ `user/profile` View ကို ဖော်ပြစေပြီး၊ View နဲ့ အတူ အသုံးပြုနိုင်ဖို့အတွက် လက်ရှိ Login ဝင်ထားသူ User ရဲ့ အချက်အလက်တွေကို တွဲဖက်ထည့်သွင်း ပေးထားပါတယ်။

ကျန် `update, logout` နဲ့ `delete` Action တွေကိုတော့ ကိုယ်တိုင်ပဲ ဆက်လက်လေ့လာ ကြည့်လိုက်ပါ။

လိုအပ်တဲ့ Route, Model နဲ့ Controller တို့ ရရှိသွားပြီဖြစ်လို့ View Template တွေ ဆက်လက်တည်ဆောက်ရပါမယ်။ Master View ဖြစ်တဲ့ `/resources/views/template.blade.php` မှာ အောက်ပါအတိုင်း ရေးသားပေးပါ။

```
<!DOCTYPE html>
<html lang="en">
<head>
	<meta charset="UTF-8">
	<title>Laravel Example</title>

	<link href="<?= URL::asset('css/app.css') ?>" rel="stylesheet">
</head>
<body>
	<h1>Laravel Example</h1>
	<div class="container">
		@yield("main")
	</div>
</body>
</html>
```

CSS ဖိုင်ချိတ်ဆက်ဖို့အတွက် Path လမ်းကြောင်းကို `URL::asset()` အကူအညီနဲ့ ရယူထားတာကို သတိပြုပါ။ ဆက်လက်ပြီး `/resources/views/user/login.blade.php` မှာ အောက်ပါအတိုင်း ရေးသားပေးပါ။

```
@extends("template")

@section("main")
<h2>Login</h2>

<? if( count($errors) ): ?>
<div class="errors">
	<ul>
		<? foreach($errors->all() as $err): ?>
			<li><?= $err ?></li>
		<? endforeach; ?>
	</ul>
</div>
<? endif; ?>

<? if( session('info') ): ?>
<div class="info">
	<?= session('info') ?>
</div>
<? endif; ?>

<form method="post" action="<?= URL::to('user/login') ?>">
	<input type="hidden" name="_token" value="<?= csrf_token() ?>">

	<input type="email" name="email" required placeholder="Your Email"><br>
	<input type="password" name="password" required placeholder="Password">

	<br><br>

	<input type="submit" value="Login">
	<a href='<?= URL::to('user/register') ?>'>Register</a>
</form>
@stop
```

Login Form တစ်ခုဖြစ်ပြီး Error နဲ့ Message တွေရှိရင် ဖော်ပြဖို့ တစ်ခါတည်း သတ်မှတ်ထားပါတယ်။ အောက်နားမှာ `user/register` Route ကို ညွှန်းထားတဲ့ Link တစ်ခုလည်း ထည့်ထားပါသေးတယ်။ ဆက်လက်ပြီး Register Form အတွက် `/resources/views/user/register.blade.php` မှာ အောက်ပါအတိုင်း ရေးသားပါ။

```
@extends("template")

@section("main")
<h2>Register</h2>

<? if( count($errors) ): ?>
<div class="errors">
      <ul>
            <? foreach($errors->all() as $err): ?>
                  <li><?= $err ?></li>
            <? endforeach; ?>
      </ul>
</div>
<? endif; ?>

<form action="<?= URL::to('user/register') ?>" method="post">

      <input type="hidden" name="_token" value="<?= csrf_token() ?>">

      <input type="text" name="name" required placeholder="Your Name"><br>
      <input type="email" name="email" required placeholder="Your Email"><br>
      <input type="password" name="password" required
               placeholder="Password"><br>
      <input type="password" name="password_again" required
               placeholder="Password Again">

      <br><br>

      <input type="submit" value="Register">
      <a href='<?= URL::to('user/login') ?>'>Login</a>
</form>
@stop
```

ဆက်လက်ပြီး User Profile ဖော်ပြဖို့အတွက် `/resources/views/user/profile.blade.php` မှာ အောက်ပါအတိုင်း ရေးသားပေးပါ။

```
@extends("template")

@section("main")
<? if( session('info') ): ?>
<div class="info">
      <?= session('info') ?>
</div>
<? endif; ?>

<form action="<?= URL::to('user/profile') ?>" method="post">

      <input type="hidden" name="_token" value="<?= csrf_token() ?>">

      <input type="text" name="name" required value="<?= $name ?>">
      Name<br>
      <input type="email" name="email" required value="<?= $email ?>">
      Email<br><br>
      <input type="text" name="phone" value="<?= $phone ?>">
      Phone<br>

      <input type="text" name="address" value="<?= $address ?>">
      Address

      <br><br>
      <input type="password" name="password">
```

```
      Password
      <br><br>

      <input type="submit" value="Update">

      <br><br>

      <a href="<?= URL::to("user/logout") ?>">Logout</a> |
      <a href="<?= URL::to("user/delete/$id") ?>">Delete Account</a>
</form>
@stop
```

လိုအပ်တဲ့ **M-V-C Code** တွေတော့ ပြည့်စုံပါပြီ။ **Style** အချို့ထည့်သွင်းလိုတဲ့အတွက် `/public/css/app.css` မှာရှိနေတဲ့ မူလ **CSS** တွေကို အောက်ပါ **CSS Code** နဲ့ အစားထိုးလိုက်ပါ။

```
html, body {
      margin: 0;
      padding: 0;
      font-family: arial, sans-serif;
}
html {
      background: #efefef;
}

body {
      width: 600px;
      margin: 20px auto;
      background: #fff;
      padding: 20px;
      border: 4px solid #ddd;
      border-radius: 5px;
}

h1 {
      margin: 0 0 20px 0;
      padding: 0 0 10px 0;
      border-bottom: 1px solid #ddd;
      font-size: 21px;
}

h2 {
      color: #555;
      padding: 0;
      margin: 0 0 10px 0;
      font-size: 16px;
}

.errors {
      background: #fee;
      border: 1px solid #900;
      font-size: 13px;
      margin: 10px 0;
}

.info {
      background: #def;
      border: 1px solid #09f;
      font-size: 13px;
      padding: 10px;
      margin: 10px 0;
}
```

လိုအပ်တဲ့ **Code** တွေတော့ စုံပါပြီး။ **Database Table** တစ်ခု တည်ဆောက်ပေးရပါဦးမယ်။ **phpMyAdmin** ကိုဖွင့်ပြီး `mydb` ဆိုတဲ့ အမည်နဲ့ **Database** တစ်ခု တည်ဆောက်ပါ။ ပြီးတဲ့အခါ `users` ဆိုတဲ့ အမည်နဲ့ **Table** တစ်ခုကို အောက်ပါ **Field** တွေနဲ့ တည်ဆောက်ပေးပါ။

users Table

```
id - INT - Auto Increment / Primary Key
name - VARCHAR (255)
email - VARCHAR (255)
phone - VARCHAR (255)
address - TEXT
password - VARCHAR (255)
remember_token - VARCHAR (255)
```

ပြီးတဲ့အခါ စတင်စမ်းသပ်နိုင်ဖို့အတွက် Setting လေးတစ်ချို့တော့ ပြင်ပေးရပါဦးမယ်။ ကျွန်တော်တို့က Eloquent ORM ကို မသုံးထားတဲ့အတွက် /config/auth.php မှာ -

```
'driver' => 'eloquent'
```

ကို အောက်ပါအတိုင်း ပြောင်းပေးဖို့ လိုပါတယ်။

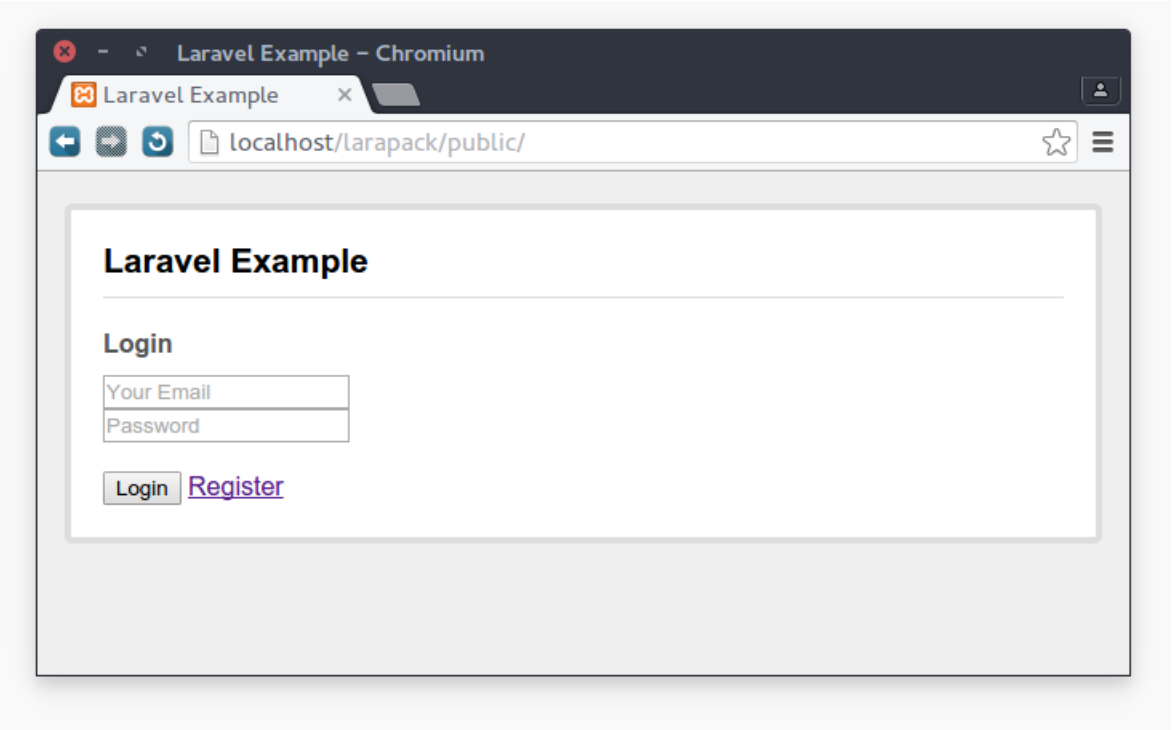
```
'driver' => 'database'
```

ပြီးတဲ့အခါ /config/database.php မှာ Database Setting တွေကို အခုလို ပြင်ဆင်ပေးရပါမယ်။

```
'mysql' => [
      'driver'    => 'mysql',
      'host'      => 'localhost',
      'database'  => 'mydb',
      'username'  => 'root',
      'password'  => '',
      'charset'   => 'utf8',
      'collation' => 'utf8_unicode_ci',
      'prefix'    => '',
      'strict'    => false,
]
```

database, username နဲ့ password တို့ကို သင့်တော်သလို ပြင်ဆင်ပေးရမှာ ဖြစ်ပါတယ်။ အားလုံးပြည့်စုံပြီမို့ အခုနေ Browser မှာ localhost/larapack/public ကို ညွှန်းလိုက်ရင် Login Form တစ်ခုကို တွေ့မြင်ရမှာပဲ ဖြစ်ပါတယ်။ ကျွန်တော်တို့က Route မှာ Root URI နဲ့ Request လုပ်လာခဲ့ရင် UserController ရဲ့ index Action ကို အလုပ်လုပ်ပေးဖို့ သတ်မှတ်ထားတဲ့အတွက် ဖြစ်ပါတယ်။ index Action က user/login View ကို ဖော်ပြဖို့ ဆက်လက် သတ်မှတ်ထားတဲ့အတွက် Login Form ကို မြင်တွေ့ရခြင်းဖြစ်ပါတယ်။

Register ပြုလုပ်ကြည့်ခြင်း၊ Login ပြုလုပ်ကြည့်ခြင်းအားဖြင့် စတင်စမ်းသပ်အသုံးပြုနိုင်ပြီ ဖြစ်ပါတယ်။



ရေးသားထားတဲ့နမူနာ Code ကို အောက်ပါလိပ်စာမှာ Download ရယူနိုင်ပါတယ်။

https://github.com/eimg/laravel-example.git

Conclusion

Laravel ဟာ ထူးခြားအသုံးဝင်တဲ့ လုပ်ဆောင်ချက်တွေ အများအပြားပါဝင်တဲ့ Framework တစ်ခုဖြစ်ပါတယ်။ ဒီနေရာမှာတော့ Laravel ကို အသုံးပြုပြီး M-V-C Code တွေ ရေးသားနိုင်ပုံကိုသာ ဖော်ပြခဲ့ခြင်းဖြစ်ပါတယ်။ Laravel ရဲ့ အားသာချက်တွေကို ထိထိရောက်ရောက် အသုံးချနိုင်စေဖို့အတွက် အောက်ပါနည်းပညာများကို အဆင့်လိုက် ဆက်လက်လေ့လာသွားသင့်ပါတယ်။

Composer – https://getcomposer.org/
Database Migration – http://laravel.com/docs/5.1/migrations
Eloquent ORM – http://laravel.com/docs/5.1/eloquent
Blade Template Engine – http://laravel.com/docs/5.1/blade
Artisan CLI – http://laravel.com/docs/5.1/artisan
Middleware – http://laravel.com/docs/5.1/middleware
Homestead – http://laravel.com/docs/5.1/homestead

အိမောင် (Fairway Web)
၂၂ ဒီဇင်ဘာ၊ ၂၀၁၅

Professional Web Developer (တတိယအကြိမ်) မှ ကောက်နှုတ်ဖော်ပြသည်။
http://pwdbook.com