

Basic Laravel Course

အရင်ဦးဆုံး Laravel Framework အသုံးမပြုမီ မိမိစက်ထဲတွင် PHP 7.1.3 နှင့် အထက်ရှိပြီး Composer ခေါ် PHP Package Manager လွှဲသင်းထားရမှာပါ။

လယ်ကူအောင် PHP ကိုတော့ Xampp ကို Download ဆဲပြီး သင်းလိုက်လို့ ရပါတယ်။

<https://www.apachefriends.org/index.html>



The screenshot shows the XAMPP website homepage. The header is dark blue with navigation links: Apache Friends, Download, Add-ons, Hosting, Community, About, a search bar, and a language selector (EN). The main content area has a large heading "XAMPP Apache + MariaDB + PHP + Perl" with an orange icon. Below this, a section titled "What is XAMPP?" explains that it is the most popular PHP development environment, free, easy to install, and contains MariaDB, PHP, and Perl. To the right of this text is a large grey box with a document icon. At the bottom, there are four buttons: a green "Download" button with a link to other versions, and three grey buttons for "XAMPP for Windows 7.3.5 (PHP 7.3.5)", "XAMPP for Linux 7.3.5 (PHP 7.3.5)", and "XAMPP for OS X 7.3.5 (PHP 7.3.5)".

Xampp နဲ့ PHP Install ပြီးတော့မှ Composer ကို သင်းလို့ရမှာပါ။

Composer ကို <https://getcomposer.org/> မှ Download ယူပြီး Install လုပ်နိုင်ပါတယ်။

Composer သင်းတဲ့ အချိန်မှာ Internet ချိတ်ထားရမှာ ဖြစ်ပါတယ်။



Dependency Manager for PHP

Latest: v1.8.5

[Getting Started](#)[Download](#)[Documentation](#)[Browse Packages](#)[Issues](#)[GitHub](#)

Composer ဆိုတာ PHP အတွက် Package Manager လေးတစ်ခုပါ။ Javascript အတွက် NPM နဲ့ အလားတူလေးပါ။ Composer သင်းပြီးမှ ကျွန်တော်တို့ အသုံးပြုတဲ့ Laravel Framework ကို Download ဆဲလို့ရမှာပါ။

Laravel အကြောင်းကို <https://laravel.com> မှာ Documentation မှ လေ့လာနိုင်ပြီးတော့ Install လုပ်နည်းလည်း အပြည့်အစုံ ရှိပါတယ်။ Composer အသုံးပြုပြီး Laravel installation လေး စတင်ပြုလုပ်လိုက်ကြရ အောင်။ Windows မှာ Command Prompt Linux မှာဆို terminal ဖွင့်ပြီးတော့ အရင်ဦးဆုံး Laravel/Installer ကို Composer နဲ့ Download ရယူရမှာ ဖြစ်ပါတယ်။

composer global require laravel/installer

Laravel/Installer Download ရယူပြီးတဲ့အချိန်မှာ မိမိနှစ်သက်ရာ Directory သို့သွားပြီး

laravel new project-name ဟု ရိုက်ပြီး Laravel Project လေးတစ်ခု တည်ဆောက်လိုက် လို့ရပါတယ်။ (Laravel project အသစ်တည်ဆောက်ချင်တိုင်း Internet ချိတ်ထားရန်လိုအပ်ပါတယ်။)

မိမိတည်ဆောက်ပြီးသော Project Directory ထဲကို **cd project-name** ဖြင့်သွားပြီးတော့ **php artisan serve** ဟုရိုက်ပြီး web server လေးတစ်ခု စ Run လို့ရပါပြီ။

composer global require laravel/installer

```
~/Desktop$ mkdir basic-laravel-course  
~/Desktop$ cd basic-laravel-course/  
~/Desktop/basic-laravel-course$ laravel new blog
```

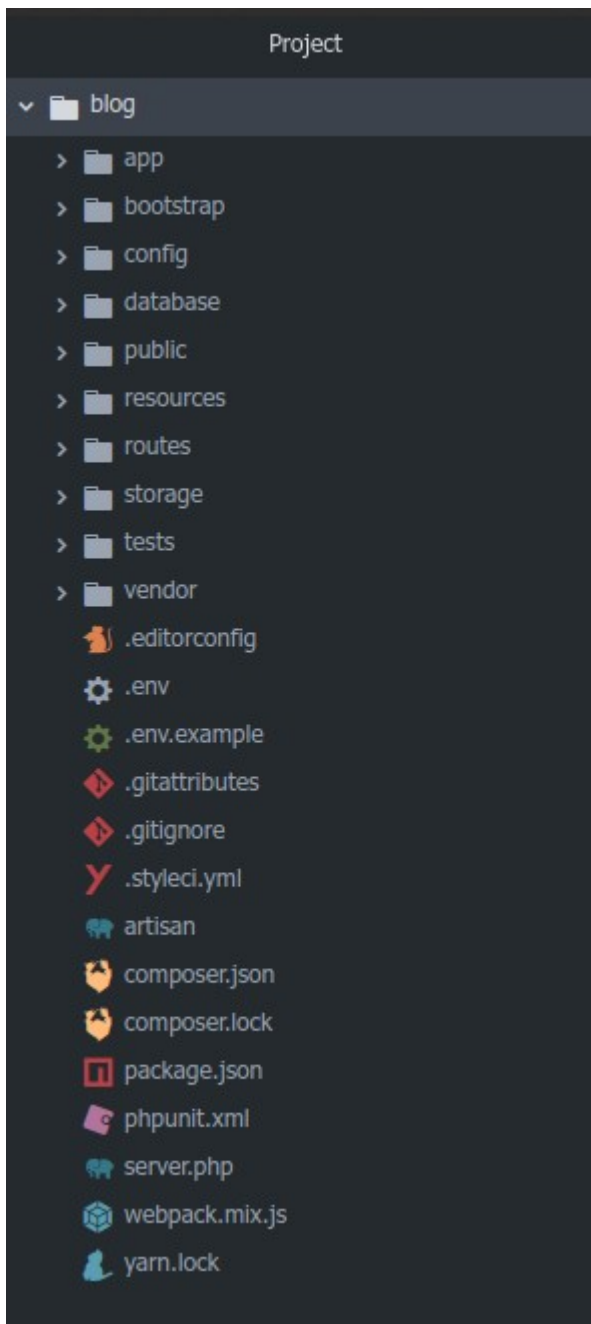
```
kksmiles@kksmiles-TM1701:~/Desktop/basic-laravel-course$ cd blog  
kksmiles@kksmiles-TM1701:~/Desktop/basic-laravel-course/blog$ php artisan serve  
Laravel development server started: <http://127.0.0.1:8000>
```



မိမိနှစ်သက်ရာ Browser မှာ 127.0.0.1:8000/ ဟုရိုက်ပြီး မိမိ website ဝင်ကြည့်လို့ရပါပြီ။

Code Editor အနေနဲ့ တော့ Sublime text ဒါမှမဟုတ် Atom ကို ကျွန်တော် Recommend ပေးပါတယ်။

ကျွန်တော်တို့ တည်ဆောက်ခဲ့တဲ့ blog ဆိုတဲ့ Project လေးကို မိမိနှစ်သက်ရာ Code Editor ထဲမှာ ဖွင့်ပြီး စလိုက်ရ အောင်။

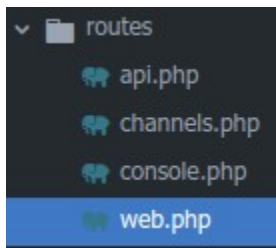


File ငွေတ များလို့ မ ကြောက်ပါနဲ့။ ကျွန်တော် တို့ Basic လေ့လာပြီး Project တစ်ခု တည် ဆောက်ရန် လေ့လာရန် လိုအပ်တဲ့ File ဇ လးငွေတက နည်းနည်းလေးပါ။

Step by Step လေ့လာကြတာပေါ့။

Note : ကျွန်တော်တို့ Website ကို စမ်းသပ်ချင်တိုင်း မိမိ project directory ထဲမှာ Command line မှ **php artisan serve** ဆိုပြီး ရိုက်ရမှာ ဖြစ်ပါတယ်။

Topic 1 : Basic routing and views

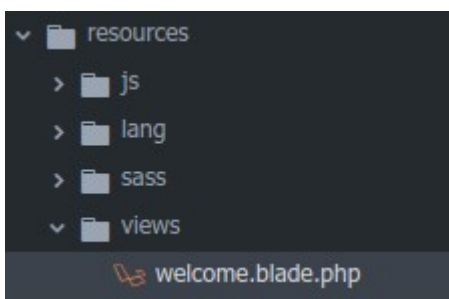


Routes ဆိုတဲ့ Folder ထဲက web.php ဆိုတဲ့ File လေးကို ဖွင့်ကြည့် လိုက်ရ အောင်။

routes/web.php

```
14 Route::get('/', function () {  
15     return view('welcome');  
16 });
```

ဒီမှာကြည့်လိုက်မယ်ဆိုရင် / က မိမိ Website Route မှာ ဝင်လာရင် welcome ဆိုတဲ့ View လေးတစ်ခု return ပြန်မယ်ဆိုတဲ့ သဘော ရပါတယ်။ ဒီတော့ welcome ဆိုတဲ့ View လေးကိုသားရှာကြည့်ရ အောင်။



Resources folder ထဲက views folder ထဲမှာ မိမိ တင်ပြမဲ့ web page ရဲ့ front-end view ဖွဲ့တ ရေး သားရမှာဖြစ် ပါတယ်။

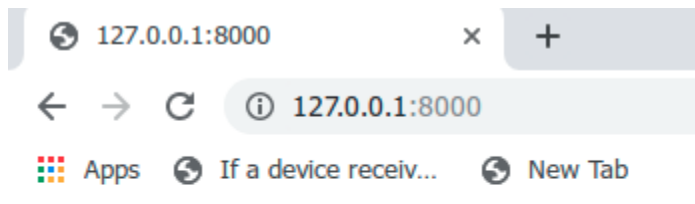
welcome.blade.php မှာ blade.php ဆိုတာ Laravel ရဲ့ Template Engine လေးပါ။

Template Engine အကြောင်းကိုတော့ နောက် Topic ကျမှ အွတ်တင်းကျကျ ရှင်းပြပါတော့မယ်။

resources/views/welcome.blade.php

```
<body>
    <h1>Basic Laravel Course</h1>
</body>
```

ဒီ simple h1 tag လေးနဲ့ Code လေးရေးပြီး Run ကြည့်တာပေါ့။



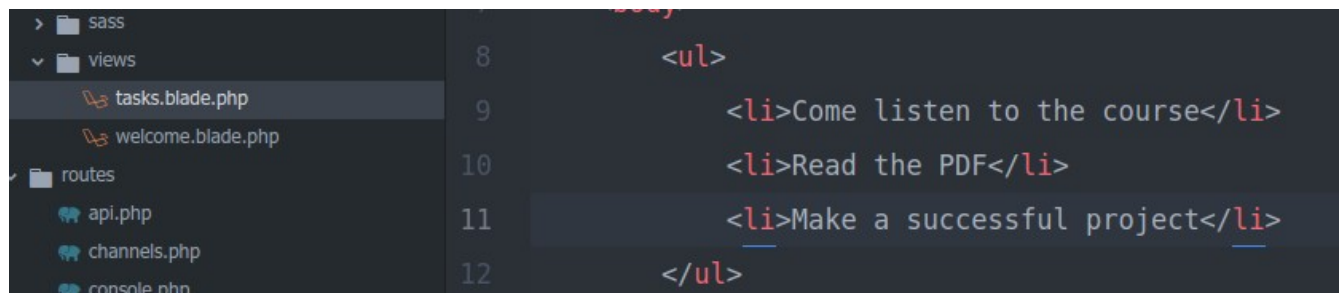
Workflow လေးကို ကျွန်တော် သေချာ ပြောပြပါမယ်။ ကျွန်တော်တို့ Website မှာ / ဒါမှမဟုတ် အွလတ် လာတဲ့ အခါ

Route ထဲက web.php code file

က နေ resources/views ထဲက

welcome.blade.php ဆိုတဲ့ View လေးကို ပိုမိုပေးလိုက်တာပါ။

ကျွန်တော် တို့ exercise အ နေနဲ့ ကိုယ့်ဟာကို view အသစ်လေးဆောက်ပြီး ၎င်း View ကို Route လေးရေးပြီးချိတ်လိုက်ကြရ အောင်။



resources/views ထဲမှာ tasks.blade.php ဆိုပြီး view လေးတစ်ဆောက်လုပ်ပါတယ်။

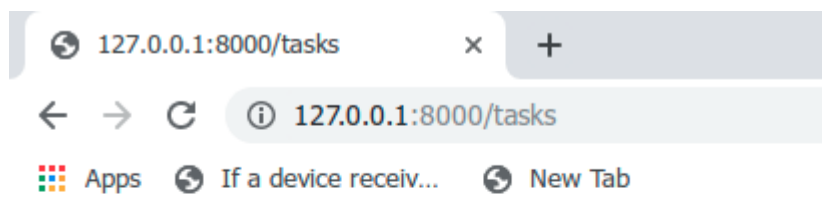
ထို code file ထဲမှာ html နဲ့ Unordered list လေးတစ်ခုရေးလိုက်ပါတယ်။

ကျွန်တော်တို့ ရေးခဲ့တဲ့ View လေးကို ချိတ်ဖို့ Route လေးတစ်ခု ရေးလိုက်ကြရအောင်။

routes/web.php

```
Route::get('/tasks', function () {  
    return view('tasks');  
});
```

ဒီ code လေး အလုပ်လုပ်ပုံလေး
က တော့ /tasks ဆိုပြီး ဝင်လာရင်
tasks ဆိုတဲ့ view လေးကို
return ပြန်ဆိုတဲ့ သဘောပါ။



ကျွန်တော်တို့ Browser ထဲမှာ
/tasks ကို သွားပြီး စမ်းသပ်လို
က်တဲ့အခါ အောင်မြင်စွာ အလုပ်
လုပ်တာကို ဖွဲ့တုရပါတယ်။

- Come listen to the course
- Read the PDF
- Make a successful project

ကျွန်တော်တို့ အလုပ်လုပ်ပုံ သားတဲ့ Workflow ကို ပြန် Recap ရှင်းပြမယ်။

- မိမိ ဖော်ပြချင်တဲ့ View လေးကို resources/views ထဲမှာ view-name.blade.php ဆိုပြီး တည်ဆောက်တယ်။
- routes/web.php ထဲမှာ ထို view ကိုသားဖို့ အွတ်ကဲ Route က လေးသတ်မှတ်တယ်။
- နောက်ဆုံး ထို Route အတိုင်း Browser မှာ စမ်းသပ်ကြည့်တယ်။

Topic 2 Blade Template Engine

ကျွန်တော်တို့ Website တစ်ခု ရေးတဲ့အချိန်မှာ page ငွတ အများကြီး ပါဝင်မှာဖြစ်ပါတယ်။

Website တစ်ခုရဲ့ Structure မှာ ပုံမှန်အားဖြင့် Header နဲ့ Footer ပါဝင်တတ်ပါတယ်။

facebook ကဲ့သို့ သော website ကို ဥပမာ အားဖြင့် ကြည့်ရ အောင်။ သင် newsfeed မှာပဲ ဖစ်ဖြစ်၊ Page တစ်ခုမှာပဲဖြစ်ဖြစ်၊ Profile တစ်ခုမှာပဲဖြစ်ဖြစ် Header ငွတ Footer ငွတ အကုန် တူတူပါဘဲ။ သင်သာ Page အကုန်လုံးမှာ တစ်ခုချင်းစီ လိုက်ရေးထားမယ်သာ ဆိုရင် အပြောင်းအလဲတစ်ခုသာ ပြုပြင်ချင်တဲ့အချိန်မှာ Page ငွတ အကုန်လုံးဆီမှာ လိုက်ပြင်နေရမယ် ဆိုရင် အဆင်မပြေ လှပါဘူး။ အဲ့ဒီ ပြဿနာ ကို Laravel ရဲ့ Blade template engine က ဖြေရှင်းပေးသားမှာဖြစ်ပါတယ်။

ကယ် ဒီတော့ စ လေ့လာလိုက်ကြရ အောင်။

Blade template engine မှာ အဓိက အသုံးပြုမဲ့ keyword လေးခုကေတာ့ extends, include, yield နဲ့ section ဖြစ်ပါတယ်။

မိမိ website ရဲ့ layout ပုံစံချတဲ့အခါ yield ကို အဓိက အသုံးပြုပြီး အခြား page များက layout ကို ခေါ်သုံးတဲ့အခါ extends နဲ့ section များကို အသုံးပြုကြပါတယ်။

လက်ငွတ လေ့လာလိုက်ကြည့်ရ အောင်။

ကျွန်တော်တို့ အရင်ဦးစာ Page တစ်ခုမှ တစ်ခု ကူးလို့ ရ အောင် a tag လေးတွေနဲ့ navigation panel တစ်ခုတည်ဆောက်ကြမယ်။ ၎င်း navigation panel ကို ကျွန်တော်တို့ ရေးပြီးသား webpage ဖွဲ့တရဲ့ အ ပေါ် လေးတွေမှာ ရေးလိုက်လို့ ရတယ်ဆိုပေမဲ့ Page နောက်တစ်ခုထပ်တိုးလို့ Navigation panel မှာ ထပ်တိုးချင်ရင် Page တိုင်းမှာ လိုက်ပြင်လို့ အဆင်မပြေပါဘူး။ အဲဒီအတွက် ကျွန်တော်တို့ navpanel.blade.php ဆိုပြီး navigation panel ကို file တစ်ခု အနေနဲ့ သီးသန့် ရေးသား တည်ဆောက်လိုက်ပါတယ်။

navpanel.blade.php

```
<div>
    <a href="/tasks"> Tasks </a>
    <a href="/"> Home </a>
</div>
```

ဒီ Code File လေးကို အခြား webpage ဖွဲ့တရဲ့ code file မှာ include လိုက်ရင် ရတယ်ဆိုပေမယ့် တဖန် အ ပေါ် မှ အောက်ပြောင်းရွှေ့ချင်ခဲ့သည်ရှိသော် code file တိုင်းမှာ ပြုပြင်ရမယ်ဆို အဆင်မပြေ ပါဘူး။ အဲဒီ အတွက် layout ပုံစံ ချရန် layout.blade.php ကို တည်ဆောက်ကြမှာဖြစ်ပါတယ်။ css file javascript များ Page များအားလုံးကိုယ်စား ချိတ်ဆက်တဲ့ အခါမှာလည်း အဆင် ပြေသွားတာပေါ့။ တချို့ Project အကြီးတွေမှာဆို CSS တို့ javascript တို့ ချိတ်ဖို့အတွက် code file သက်သက်ဆီ ရေးကြပါတယ်။ ဒီမှာတော့ ရိုးရှင်းအောင် မလုပ်ပြ တော့ပါဘူး။

layout.blade.php

```
<!DOCTYPE html>
<html lang="en" dir="ltr">
  <head>
    <meta charset="utf-8">
    <title>@yield('title')</title>
  </head>
  <body>
    @include('navpanel')
    @yield('content')

    <script type="text/javascript">

    </script>
  </body>
</html>
```

Yield ဆိုတာ သူ့ကိုလာ extends လုပ်တဲ့ page ရဲ့ section ကိုထုတ်ပြတာပါ။

ဒီမှာ ရေးထားပုံကို လေ့လာကြည့်မယ်ဆိုရင် title tag မှာ title ဆိုတဲ့ section လေးကို ထုတ်ပြမယ် page ထိပ်ဆုံးမှာ Navigation panel လေးတစ်ခုဖော်ပြမယ်။ ပြီးရင် Content ဆိုတဲ့ section လေးကို ထုတ်ပြမယ်ဆိုတဲ့ သဘောပါ။

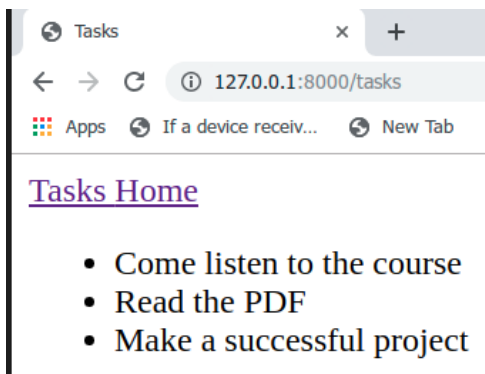
ကျွန်တော့်တို့ ယခင်က ရေးသားခဲ့တဲ့ tasks.blade.php နဲ့ welcome.blade.php ကို သန့်ရှင်းအောင် ပြုပြင်ရေးသားကြစို့။

tasks.blade.php

```
@extends('layout')

@section('title')
    Tasks
@endsection

@section('content')
    <ul>
        <li>Come listen to the course</li>
        <li>Read the PDF</li>
        <li>Make a successful project</li>
    </ul>
@endsection
```



အရင်ဦးစာ layout.blade.php code file လေးကို extends('layout') နဲ့ extend လုပ်လိုက်ပါတယ်။

@section('name') @endsection ဟာ yield('name') နေရာမှာ ထုတ်ပြန်ဖို့အတွက် ရေးခြင်းဖြစ်ပါတယ်။ ယခင် layout file ရဲ့ yield ထားတဲ့ နေရာမှာ ယခုရေးထားတဲ့ section လေးတွေက နေရာဝင်ယူသားမှဖြစ်ပါတယ်။

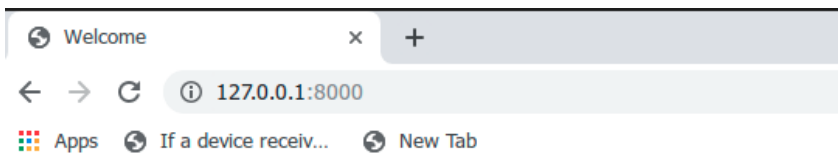
welcome.blade.php

```
@extends('layout')

@section('title')
    Welcome
@endsection

@section('content')
    <h1>Basic Laravel Course</h1>
    <p>
        This is basic laravel course. Please listen carefully.
    </p>
@endsection
```

ဒီမှာလည်း အလားတူပါဘဲ။ layout ကို extends လုပ်တယ်။ layout file ရဲ့ yield ထားတဲ့ နေရာတွေမှာ နေရာယူဖို့ Section ကြေညာပြီး ရေးတယ်။



[Tasks Home](#)

Basic Laravel Course

This is basic laravel course. Please listen carefully.

Blade ရဲ့ template engine

အသုံးဝင်ပုံကို ရေးရမယ့် page

များလာလေ သိသာလာလေ ဖြစ်ပါတယ်။

Topic 3 Sending data to your views

ကျွန်တော်တို့ အ ပေါ်က tasks.blade.php ကို ပြန်ဖွင့်ကြည့်လိုက်ပါ။ unordered list ထဲမှာ List item လေးတွေကို Hard code ရေးထားတာတွေ့ရပါမယ်။ တကယ့် လက်တွေ့ Project မှာ ဆို ဒါ လက်တွေ့မဆန်ပါဘူး။ database ထဲက နေ Data ထုတ်ပြီး list ထဲမှာ ထည့်သင်းဖော်ပြရမှာဖြစ်ပါတယ်။ ဒါကြောင့် ဒီ သင်ခန်းစာမှာ Database မပါသေးဘူးဆိုပေမယ့် View ကို Data ပို့ ခြင်းကစပြီး လေ့လာသားမှာဖြစ်ပါတယ်။ Step by Step ဝေပေါ့။

routes/web.php

```
Route::get('/tasks', function () {  
    return view('tasks', [  
        'foo'=>'bar'  
    ]);  
});
```

ဒီက Code လေးကို လေ့လာကြည့်မယ်ဆိုရင် View ရဲ့ ဒုတိယ Parameter မှာ 'foo'=>'bar' ဆိုပြီး JSON Format လေးနဲ့ data ပို့ထားတာကို တွေ့ရပါမယ်။

tasks.blade.php

```
1 @extends('layout')  
2  
3 @section('title')  
4     Tasks  
5 @endsection  
6  
7 @section('content')  
8     {{ $foo }}  
9 @endsection  
10
```

← → ↻ ⓘ 127.0.0.1:8000/tasks

🌐 Apps 📶 If a device receiv... 🆕 New Tab

Tasks Home About us
bar

tasks.blade.php မှာ ယခင်က ရေးသားထားတဲ့ list လေးကို ဖျက်ပြီး ကျွန်တော်တို့ route ဘက်က ပေးပို့ထား

တဲ့ foo ဆိုတဲ့ Variable လေးကို ထုတ်ကြည့်လိုက်ရ အောင် {{ \$foo }} ဆိုတာ php echo နဲ့ အလားတူလေးပါ။ php echo နဲ့ လုံးဝ ဆင်တူလားဆိုတော့ မဟုတ်ပါဘူး။

php echo က Javascript ထုတ်ရင် (eg. <script> alert('hello') </script>) ၎င်း script ကို ကီးကို run သွားမှာပါ။ ကိုယ့်ဟာကိုယ် စမ်းကြည့်လိုက်ပါ

'foo' => '<script> alert('hello') </script>' ဆိုပြီး echo \$foo နဲ့ တစ်ခါထုတ်ကြည့် {{ \$foo }} နဲ့ တစ်ခါထုတ်ကြည့်လိုက်ပါ။ echo နဲ့တုန်းက ၎င်း Javascript run သွားပြီး {{ \$foo }} မှာကြ တော့ <script> အတိုင်း display ထွက်သားမှာ ကိုတွေ့ရပါတယ်။ အကယ်၍ script ကို run စေချင်တယ်ဆိုရင်တော့ {!! \$foo !!} ဆိုပြီးရေးရမှာဖြစ်ပါတယ်။ ဒါတွေ ဘာကြောင့် သိဖို့ လိုလဲဆိုတော့ မိမိ website မှာ request ပေးပို့တဲ့ အခါ Malicious script ဖွဲ့တဆီက နေ ကွက်ကယ်ဖို့ အွတ်ကံ Laravel က automatic လုပ်ပေးထားတဲ့ စနစ် က လေးကို အသုံးပြုပုံနဲ့ echo နဲ့ ကာခြားပုံနဲ့ အသုံးပြုပုံတွေ သိဖို့လိုအပ်ပါတယ်။

ခုနုကအတိုင်း Data ပို့တာကို list ထုတ်ဖို့အွတ်က tasks array လေးတစ်ခု ပေးပို့လိုက်ရအောင်။

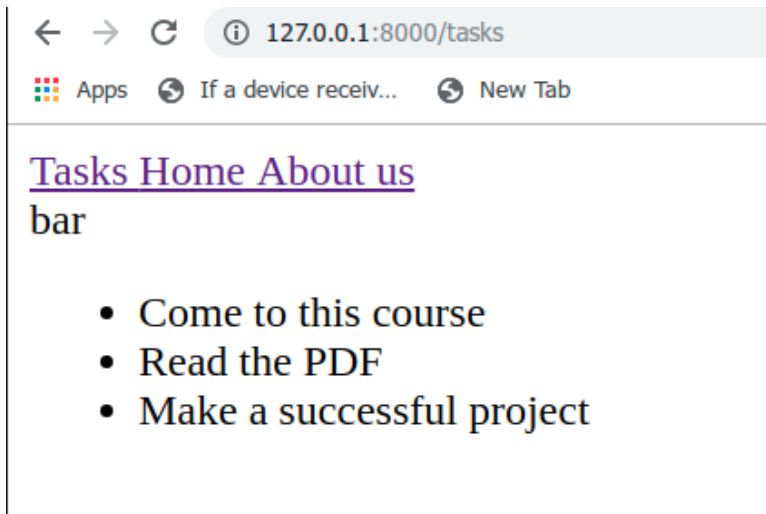
route/web.php

```
Route::get('/tasks', function () {  
    $tasks = [  
        "Come to this course",  
        "Read the PDF",  
        "Make a successful project"  
    ];  
    return view('tasks', [  
        'foo' => 'bar',  
        'tasks' => $tasks  
    ]);  
});
```

ဘာမှ ထူးထူးခြားခြား မဟုတ်တဲ့ အွတ်က အွတ်ကင်းကျကျ မရှင်းပြ တော့ပါဘူး။ \$tasks ဆိုတဲ့ array လေးကြည့်ပြီး ပို့လိုက်တဲ့သ ဘောပါဘဲ။

tasks.blade.php

```
@section('content')  
    {{ $foo }}  
    <ul>  
        @foreach ($tasks as $task)  
            <li>{{ $task }}</li>  
        @endforeach  
    </ul>  
  
@endsection
```

ယခု ပြလိုက်တဲ့ data ပို့နည်းထပ်ပိုပြီးသန့်ပြီး လယ်ကူတဲ့ Data ပို့နည်းလေးပြချင်ပါတယ်။

```
Route::get('/tasks', function () {
    $tasks = [
        "Come to this course",
        "Read the PDF",
        "Make a successful project"
    ];
    $foo = 'bar';
    return view('tasks', compact('tasks', 'foo'));
});
```

compact ဆိုပြီး data လေးပို့တာပါ။ ယခင်ကဟာနဲ့ အလားတူပါပဲ။ \$tasks ကို view ဘက် မှ \$tasks ဆိုပြီး ခေါ်သုံးလိုရ အောင်ပါ။ result ကလည်းတူတူပါဘဲ။ အခြား withTasks(\$tasks) လိုကဲ့သို့သော အခြားdata ပို့နည်းတွေလည်းရှိပါတယ်။ ရှုပ်လည်းရှုပ် ကျွန်တော်ကိုယ်တိုင်လည်း လက်တွေ့ မသုံးတဲ့အွတ်ကံ မ ဖော်ပြ တော့ပါဘူး။ ယခု ပြထား တဲ့ Data ပို့နည်းနှစ်ခုနဲ့ဆို Project တစ်ခုတည်ဆောက်ဖို့အွတ်ကံ လံလောက်ပါတယ်။ သိချင်ရင်တော့ ကိုယ့်ဟာကိုယ်လေ့လာကြည့်လို့ရပါတယ်။

Topic 4 Working with controllers

ကျွန်တော်တို့ ခုနက route/web.php file ကိုကြည့်လိုက်ရင် ရှုပ်ပွဲ နေတာကိုတွေ့ရပါလိမ့်မယ်။ ကျွန်တော်တို့ Data နှင့် processing နှစ်ခုလုံး လုပ်သမျှ function တွေကို route/web.php file မှာလုပ်နေရမှာ မသင့်တော်ပါဘူး။ အဲဒီတော့ ကျွန်တော်တို့ MVC architecture က Controller က ဒီနေရာမှာ စတင် အလုပ်လုပ်ပါတော့မယ်။

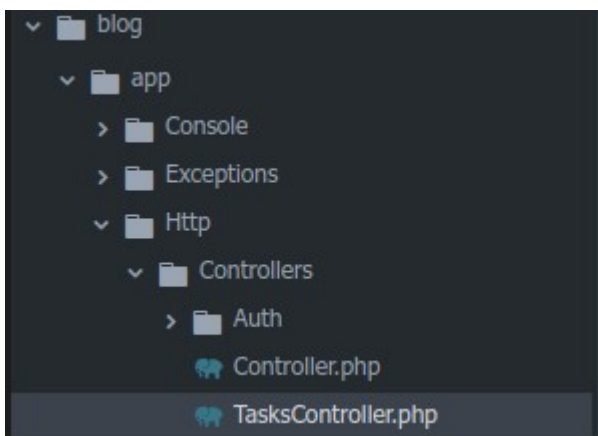
Controller တစ်ခု တည်ဆောက်လိုက်ကြရအောင်။

ကျွန်တော်တို့ အရင်ဦးစား မိမိ Project directory ဆီသို့ Cmd line နဲ့ သွားမယ်။

php artisan ဆိုပြီး ရိုက်ကြည့်လိုက်ရင် အသုံးပြုရတဲ့ artisan command တွေ တွေ့ရမှာပါ။ အဲမှာ make ထဲက make:controller ကိုအသုံးပြုရမှာဖြစ်ပါတယ်။

php artisan help make:controller ဆိုပြီးရိုက်လိုက်ရင် အသုံးပြုပုံလေးကို ပြထားမှာဖြစ်ပါတယ်။ အဲဒီတော့ ကျွန်တော်တို့ ဦးစား Simple အဖြစ်ဆုံးနဲ့ TasksController လေးတစ်ခု တည်ဆောက်လိုက်ကြရအောင်။

```
blog$ php artisan make:controller TasksController
```



app/http/controllers ထဲမှာ
TasksController.php ဆိုပြီး Automatic
Boiler Plate လေးတည်ဆောက်ပြီးသား ဖြစ်သွားမှာဖြစ်ပါတယ်။

TasksController ထဲမှာ ကျွန်တော်တို့ အရှေ့က routes/web.php မှာရေးသားခဲ့တဲ့ ဟာလေးတွေကို Copy ကူးလိုက်ရုံပါဘဲ။

app/http/controller/TasksController.php

```
class TasksController extends Controller
{
    public function show(){
        $tasks = [
            "Come to this course",
            "Read the PDF",
            "Make a successful project"
        ];
        $foo = 'bar';
        return view('tasks', compact('tasks', 'foo'));
    }
}
```

routes/web.php

```
Route::get('/tasks', 'TasksController@show');
```

Route file မှာ Controller ထဲက function လေးကို အထက်ပါအတိုင်း reference လုပ်ပေးလိုက်ရုံပါဘဲ။ စမ်းကြည့်လိုက်ရင် Result ကတူတူပါဘဲ။ Code ပို သန့် သားတာပဲရှိတာပါ။

ကယ် Controller အသုံးပြုပုံက တော့ ဒီလောက်ပါဘဲ။ ကျွန်တော်တို့ အခု file ငွေတ ခုန့်ကူးကတာ များလာပြီဆိုတော့ File location မသိတာငွေတဖြစ်နိုင်တဲ့အတွက် trick လေးတစ်ခု ဟပြပါမယ်။ Atom နဲ့ Sublime text မှာ shortcut ctrl+p နဲ့ မိမိသားချင်တဲ့ File ဆီကို file name ရိုက်ပြီးသားလို့ရပါတယ်။

Topic 5 Databases and migrations

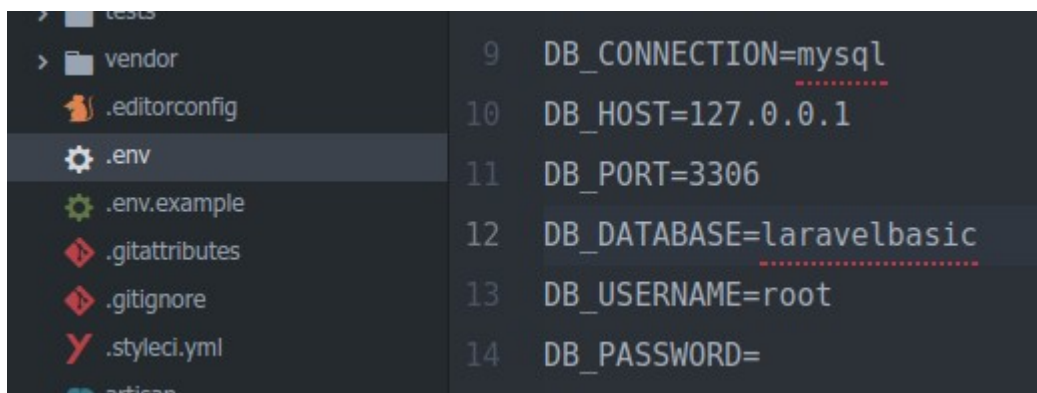
ကျွန်တော်တို့ အခုလုပ်သားခဲ့သမျှ ငွတ အားလုံးဟာ ကိုယ့်လက်ရှိ code file ထဲမှာဘဲ variable ကြေညာ data ထည့်၊ data ပို့ နဲ့လုပ်ကိုင်သားတာဖြစ်တယ်။ တကယ့် လက်ငွတ၊ Project ငွတမှာ Database ထဲက data ငွတထုတ်ပြီးလုပ်ရမှာဖြစ်တဲ့အွတက် ယခု သင်ခန်းစာမှာ Database တည်ဆောက်နည်းလေး လေ့လာကြတာပေါ့။

ကျွန်တော်တို့ တစ်ဦးတည်းသား Project တစ်ခုလုပ်မယ်ဆိုရင် phpmyadmin မှာ database ငွတ database table ငွတ တခါတည်း ကိုယ့်ဟာကိုယ်ဆောက်တာ အ ဆင်ပြေ ကောင်း ပြေ ပပါလိမ့်မယ်။ ဒါပေမယ့် Production server ပေါ်တင်တာပဲဖြစ်ဖြစ်၊ Project team members အများကြီးနဲ့ အလုပ်လုပ်တဲ့အချိန်၊ Git ကဲ့သို့သော version control system ငွတ အသုံးပြုလာတဲ့အချိန်ငွတ မှာ ဒီနည်းလမ်းဟာ အဆင်မ ပြေ လက်ငွတ၊ မကျပါဘူး။

ဘာလို့လဲဆိုတာ သင်ခန်းစာအဆုံးမှာ ရှင်းပြပါမယ်။

Laravel မှာ ဒီပြဿနာကို ဖြေရှင်းဖို့အွတက် migration ဆိုတဲ့ နည်းပညာလေးတစ်ခု အသုံးချဖို့ အဆင်သင့်ရှိပါတယ်။

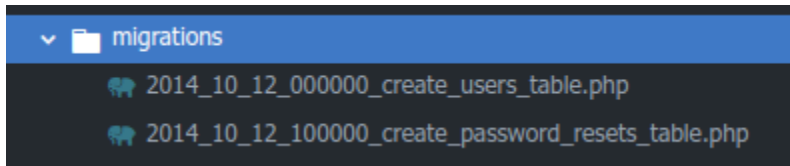
ကယ်အရင်ဦးစာ မိမိစမ်းသပ်ဖို့အွတက် နှစ်သက်ရာ database နဲ့ ချိတ်နည်းလေးပြပါမယ်။



```
9 DB_CONNECTION=mysql
10 DB_HOST=127.0.0.1
11 DB_PORT=3306
12 DB_DATABASE=laravelbasic
13 DB_USERNAME=root
14 DB_PASSWORD=
```

.env file လေးထဲမှာ မိမိချိတ်မဲ့ database config လေး အရင်ဦးစာ ဖြည့်သင်းရပါမယ်။

Note : php artisan လို့ command line မှာ ရိုက်ပြီး artisan မှ run လို့ရတဲ့ Command ဖွဲ့တစ်ခုထုတ်ကြည့်လို့ရပါတယ်။ php artisan help <command> နဲ့ ၎င်း command ကို အသုံးပြုနည်းကို ထုတ်ကြည့်လို့ရပါတယ်။



ကျွန်တော်တို့ အခုလက်ရှိ database/migrations directory ထဲကို သွားကြည့်လိုက်မယ်ဆိုရင် create_users_table.php နဲ့ create_password_resets_table.php ကိုမြင်ဖွဲ့ရမှာပါ။ ကျွန်တော်တို့ ဘာမှပြုပြင်ခြင်းမလုပ်ဘဲ php artisan migrate ဆိုပြီး command line မှာ ရိုက်ကြည့်လိုက်ကြရအောင်။

```
kksmiles@kksmiles-TM1701:~/Desktop/basic-laravel-course/blog$ php artisan migrate
Migration table created successfully.
Migrating: 2014_10_12_000000_create_users_table
Migrated: 2014_10_12_000000_create_users_table
Migrating: 2014_10_12_100000_create_password_resets_table
Migrated: 2014_10_12_100000_create_password_resets_table
```

** တချို့ Laravel version ဖွဲ့တော့ ပြဿနာ error တက်တတ်ပါတယ်။ ဒီတိုင်းကိုယ့်ဟာကိုယ် Google ခေါက် database table ဖွဲ့တစ်ခု drop ပြီး ပြန်လည်စမ်းသပ်ပြီး ဖြေရှင်းလိုက်ပါ။**

Table နှစ်ခု migrate လုပ်တာ အောင်မြင်သွားတာကို ဖွဲ့ရနိုင်ပါတယ်။

Table	Action
<input type="checkbox"/> migrations	★ Browse Structure Search Insert Empty Drop
<input type="checkbox"/> password_resets	★ Browse Structure Search Insert Empty Drop
<input type="checkbox"/> users	★ Browse Structure Search Insert Empty Drop
3 tables	Sum

Database ထဲမှာ table 3 ခုတည်ဆောက်သွားတာကို ဖွဲ့ရပါလိမ့်မယ်။ Migrations ဆိုတဲ့ table က တော့ Laravel ကသူ့ဟာသူ migration log သိမ်းဆည်းဖို့အတွက် auto တည်

ဆွက်သားတဲ့ table ဖြစ်တယ်။ ဂရုစိုက်စရာမလိုပါဘူး။ ကျွန်တော်တို့ ဒီမှာ သေချာ လေ့လာကြည့်မယ်ဆိုရင် create_users_table.php ဆိုတဲ့ Migration file လေးဟာ users ဆိုတဲ့ table ကိုတည်ဆွက်သားပါတယ်။ ကျွန်တော်တို့ ကိုယ်တိုင် Migration မတည်ဆောက်ခင် အရင်ရှိပြီးသား Migration လေးကို ကြည့်လိုက်ရ အောင်။

```
public function up()
{
    Schema::create('users', function (Blueprint $table) {
        $table->bigIncrements('id');
        $table->string('name');
        $table->string('email')->unique();
        $table->timestamp('email_verified_at')->nullable();
        $table->string('password');
        $table->rememberToken();
        $table->timestamps();
    });
}
```

ဒါလေးဖတ်ကြည့်မယ်ဆို နားလည်ရ လွယ်ကူပါတယ်။ id ဟာ autoincrement big integer လေး တစ်ခုဖြစ်မယ်။ name ဟာ string လေးတစ်ခုဖြစ်မယ်။ email ဟာ string ဖြစ်ပြီး unique ပါဖြစ်ရမယ်။ email_verified_at ဟာ အချိန်မှတ်သားတဲ့ timestamp လေးတစ်ခုဖြစ်မယ်။ password ဟာ string လေးတစ်ခုဖြစ်မယ်။ အောက်ဆုံးကနစ်ခုကေတော့ data query တစ်ခုနဲ့ ဝင်လာတဲ့ အချိန်သတ်မှတ်ဖို့ မှတ်တဲ့ field လေးတွေပါ။

ကယ် ကျွန်တော်တို့ ကိုယ်တိုင် database table migration လေးတည်ဆောက်လိုက်ရ နေအောင်။

```
php artisan make:migration create_projects_table
Created Migration: 2019_06_07_131916_create_projects_table
```

projects table တည်ဆောက်ချင်တာဖြစ်တဲ့အတွက် သူ့ စံပုံစံအတိုင်းဘဲ create_projects_table ဆိုပြီး migration လေးတစ်ခုတည်ဆောက်လိုက်ပါတယ်။

```
public function up()
{
    Schema::create('projects', function (Blueprint $table) {
        $table->bigIncrements('id');
        $table->string('title');
        $table->text('description');
        $table->timestamps();
    });
}
```

အခုကျွန်တော်တို့ လိုချင်တဲ့ data field title နဲ့ description လေးထည့်ပြီး migrate လုပ်လိုက်ကြရအောင်။

```
kksmiles@kksmiles-TM1701:~/Desktop/basic-laravel-course/blog$ php artisan migrate
Migrating: 2019_06_07_131916_create_projects_table
Migrated: 2019_06_07_131916 create projects table
```

```
SELECT * FROM `projects`
```

id	title	description	created_at	updated_at
----	-------	-------------	------------	------------

ကျွန်တော်တို့ လိုချင်တဲ့ Projects table လေး တည်ဆောက်ပြီးသားဖြစ်တာတွေတပါမယ်။

php artisan migrate:fresh ဆိုတဲ့ command run မယ်ဆိုရင်တော့ table နှုတ် အကုန်
drop ပြီး အစက migration အကုန်ပြန်လုပ်မှာဖြစ်ပါတယ်။ အခြား migration command
နှုတ် အကြောင်းလေ့လာချင်ရင်တော့ documentation သားဖတ်လို့ရပါတယ်။

ဒီနည်းပညာလေးက ဆိုရင်ဖြင့် ကျွန်တော်တို့ project team members နှုတ် အများကြီးနဲ့
Git ကဲ့သို့သော version control system လေး အသုံးပြုတဲ့အခါ ဘာကြောင့်အသုံးဝင်လဲဆို
တာ ကို ရှင်းပြပါမယ်။

ကျွန်တော်တို့ ပုံမှန် လမ်းစဉ်အတိုင်းသာ database ကိုယ့်ဟာကိုယ် တည်ဆောက်ပြီး team
member တစ်ဦးရှိကို လွှဲပြောင်းတဲ့ အခါကျ database ကို export လုပ်ကတယ်။ export
လုပ်ထားတာလေးကို ပေးကတယ်။ ဟိုဘက်ကလည်း တဖန် ပြန်လည် import ကတယ်။ အ
လုပ်ရှုပ်ပါတယ်။ ဒါကြောင့် ဒီ migration file လေးနှုတ်သော git က နေပဲဖြစ်ဖြစ်၊ ဒီတိုင်းယူလို
က်တာပဲဖြစ်ဖြစ် ပြီးတာနဲ့ migrate လုပ်လိုက်ရုံနဲ့ဘဲ project ကူးပြောင်း ပေါင်းစပ်ကတာ
အလွန် လွယ်ကူတာကို သိသိသာသာ နှုတ် ရမှာဖြစ်ပါတယ်။

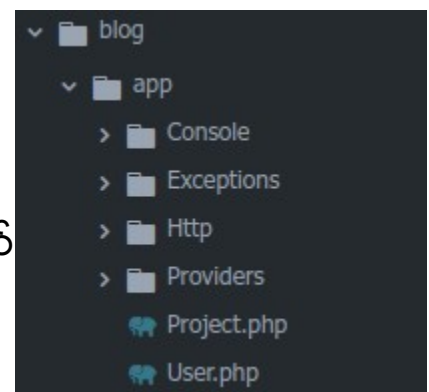
Topic 6 Model and eloquent

ကျွန်တော်တို့ Database နဲ့ အလုပ်လုပ် တော့မယ်ဆိုရင် Laravel မှာ eloquent နဲ့ အလုပ်လုပ်ပါတယ်။ ဘယ်လိုမျိုးလဲဆိုတော့ projects ဆိုတဲ့ database table ထဲက row တစ်ခုထုတ်မယ်ဆိုရင် Project Model ပုံစံနဲ့ Project object လေးတစ်ခု အနေနဲ့ အလုပ်လုပ်သွားမှာပါ။

ကယ်ဒါဆို လက်တွေ့ ကြည့်ကြတာပေါ့။ အရင်ဦးစွာ Project ဆိုတဲ့ model လေးတစ်ခု တည်ဆောက်မယ်။

```
$ php artisan make:model Project  
Model created successfully.
```

app directory ထဲမှာ Project.php ဆိုတဲ့ Model လေးထွက်လာမှာဖြစ်ပါတယ်။



ကျွန်တော်တို့ web page တွေမှာ လက်တွေ့ မ run ခင် အရင်ဦးဆုံး php artisan tinker ဇလးနဲ့ စမ်းကြည့်ရ အောင်။ Php artisan tinker ဆိုတာ မိမိ project ရဲ့ model တွေ စမ်းချင်တာပဲဖြစ်ဖြစ်၊ php function တွေ စမ်းချင်တာပဲဖြစ်ဖြစ်၊ query ဖွဲ့အစည်း run ချင်တဲ့အခါဖြစ်ဖြစ် ဆော့ကစားလို့ရတဲ့ php ကစားကွင်း လေးလိုမျိုးမြင်ကြည့်လိုက်ပါ။

ကယ်စမ်းကြတာပေါ့။

အရင်ဦးစွာ php artisan tinker ဆိုတဲ့ command နဲ့ ကစားကွင်းထဲဝင်မယ်။

```

kksmiles@kksmiles-TM1701:~/Desktop/basic-laravel-course/blog$ php artisan tinker
Psy Shell v0.9.9 (PHP 7.2.19-0ubuntu0.19.04.1 - cli) by Justin Hileman
>>> App\Project::all();
=> Illuminate\Database\Eloquent\Collection {#3197
    all: [],
}
>>> App\Project::first();
=> null
>>> App\Project::latest()->first();
=> null
>>> $project = new App\Project;
=> App\Project {#3192}
>>> $project
=> App\Project {#3192}
>>> $project->title = 'My First Project';
=> "My First Project"
>>> $project->description = 'some description';
=> "some description"
>>> $project->save();
=> true
>>>

```

ကျွန်တော်တို့ တည်ဆောက်ခဲ့တဲ့ App directory ထဲမှာ ရှိတဲ့ Project class (Model) လေးကို App\Project နဲ့ သူ့အထဲက function ဖွဲ့တ attribute ဖွဲ့တ ထုတ်သုံးလို့ရမှာဖြစ်ပါတယ်။ App\Project::all() ဆိုပြီး eloquent model ဖွဲ့တ မှာ default ပါတဲ့ all function လေးနဲ့ ထုတ်ကြည့် လိုက်တဲ့အခါ ကျွန်တော်တို့ database table ထဲမှာ ဘာမှမရှိသေးတဲ့အတွက် null ဆိုပြီး ထက်လာမှာဖြစ်ပါတယ်။ ဒီမှာတော့ ကျွန်တော် common ဖြစ်တဲ့ first() တို့ latest() တို့ run ပြထားတယ်။ ဒါပေမယ့် table အလတ်ကြီးဖြစ်တဲ့ အတွက် result က တော့ null ဘဲ။ ကယ် အခု database ထဲ data ထည့်လိုက်ကြရ အောင်။ \$project = new App\Project ဆိုပြီး Project model နဲ့ object တစ်ခုတည်ဆောက်လိုက်တယ်။ ပြီးတော့ project ရဲ့ title ဖွဲ့တ description ဖွဲ့တထည့်ပြီး \$project->save() ဆိုပြီး data သင်းလိုက်တယ်။ true ဆိုပြီး အောင်မြင်ကြောင်း ပြန်ပြတယ်။ ကျွန်တော်တို့ database table ထဲသားကြည့်တဲ့အခါ data ဝင်ကြောင်း မြင်ဖွဲ့တ နိုင်ပါတယ်။

+ Options

	id	title	description	created_at	updated_at
1		My First Project	some description	2019-06-08 09:47:57	2019-06-08 09:47:57

```

>>> App\Project::first();
=> App\Project {#3204
    id: 1,
    title: "My First Project",
    description: "some description",
    created_at: "2019-06-08 09:47:57",
    updated_at: "2019-06-08 09:47:57",
}
>>> App\Project::first()->title;
=> "My First Project"
>>> App\Project::first()->description
=> "some description"

```

ကျွန်တော်တို့ tinker နဲ့ ပြန်ပြီး ပုံပါအတိုင်း data ပြန်ထုတ်ကြည့်လိုရပါတယ်။

knowledge အ နေနဲ့ သိထားသင့်တာက တော့ App\Project::all() ဆိုရင် Project object ဇွတ်အားလုံး စုထားတဲ့ collection လေးထက်လာမှာကိုပါဘဲ။

collection ကဘာလဲဆိုတော့ နည်းနည်းပိုမိုက်တဲ့ array လို့ဘဲ မှတ်ထားလိုက်ပါ။

Eloquent model ဇွတ် collection ဇွတ်မှာ run လို့ရတဲ့ function ဇွတ်ကို official documentation မှာသားရောက် လေ့လာ စမ်းသပ်ကြည့်လိုရပါတယ်။

ကျွန်တော်တို့ အခု Project model တစ်ခုရပြီဆိုပေမယ့် သူကိုထိန်းမဲ့ Controller မ ဆောက်ရ သေးပါဘူး။ အဲဒီအတွက် php artisan make:controller ProjectsController ဆိုပြီး တည်ဆောက်မယ်။

```

namespace App\Http\Controllers;
use Illuminate\Http\Request;
use App\Project;

class ProjectsController extends Controller
{
    public function index()
    {
        $projects = Project::all();
        return $projects;
        return view('projects.index');
    }
}

```

ကျွန်တော်တို့ ဒီ ProjectsController ရဲ့ code ကိုလေ့လာကြည့်မယ်ဆိုရင် ကျွန်တော် page ထိပ်ဆုံးမှာ **use App\Project** ဆိုတာကို ကြေညာထားတာတွေ ရပါမယ်။ ဒါကိုရှင်းပြရမယ်ဆိုရင်တော့ အပေါ်က namespace App\Http\Controllers ကြောင့်ပါ။ အကယ်၍ ကျွန်တော်တို့ အောက်မှာသာ `$projects = App\Projects::all()` ဆိုပြီး tinker တုန်းကအတိုင်း အသုံးပြုမယ်ဆိုရင် **App\Http\Controllers\App\Projects::all()** ဆိုပြီး သွား run မှာပါ။ အဲ့ဒီအခါ ၎င်း directory မှာ Project model မရှိတဲ့အတွက် error တက်ပါလိမ့်မယ်။ ထိုပြဿနာကို ရှောင်ချင်ရင် **`$projects = \App\Projects::all()`** ဆိုပြီး ရေးသားလို့ရပါတယ်။ ဒါပေမယ့် အလုပ်ရှုပ်တဲ့ အတွက် **use App\Project;** ဆိုပြီး အလွယ်တကူ ရှောင်ရှားလိုက်တာပါ။ **use App\Project as Project** လို့ အဓိပ္ပာယ်ရောက်ပါတယ်။ Project လို့ ဒီ code file ထဲမှာ နေရာရင် App\Project ကိုသွား ရှာပြီးအလုပ်လုပ်ပါလို့ အဓိပ္ပာယ်ရပါတယ်။

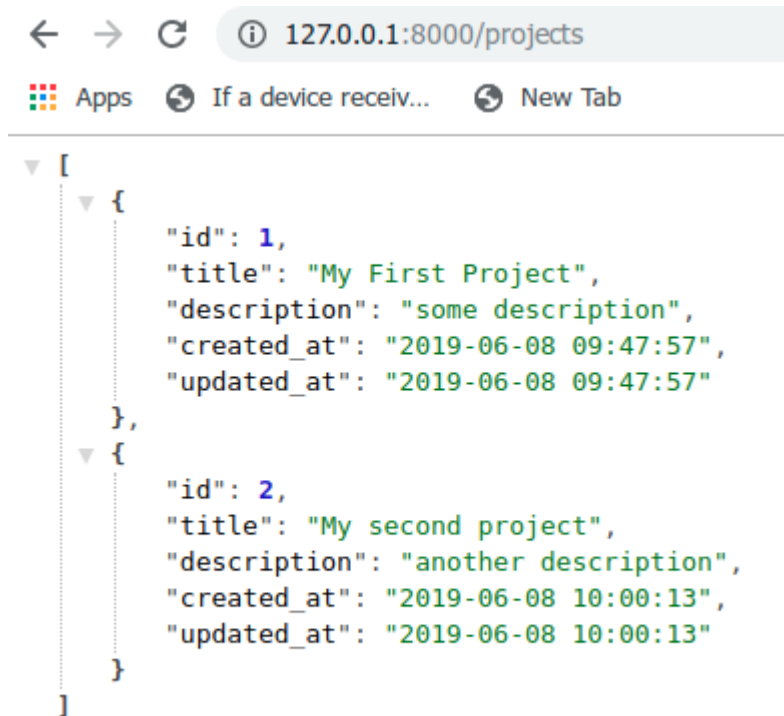
use App\Project as something ဆိုရင်တော့ something လို့ နေရာရင် App\Project ကို ရှာအလုပ်လုပ်သွားမှာပါ။

ကျွန်တော်တို့ ဒီဟာတွေအတွက် route မ ရေးရ သေးတဲ့အတွက် routeလေး ရေးလိုက်ကြ တာပေါ့။

routes/web.php

```
Route::get('/projects', 'ProjectsController@index');
```

run လိုက်မယ်ဆိုရင် return \$projects လို့ ပြန်ထားတဲ့အတွက် projects ငွေအကုန် json format နဲ့ မြင်တွေ့ရမှာဖြစ်ပါတယ်။

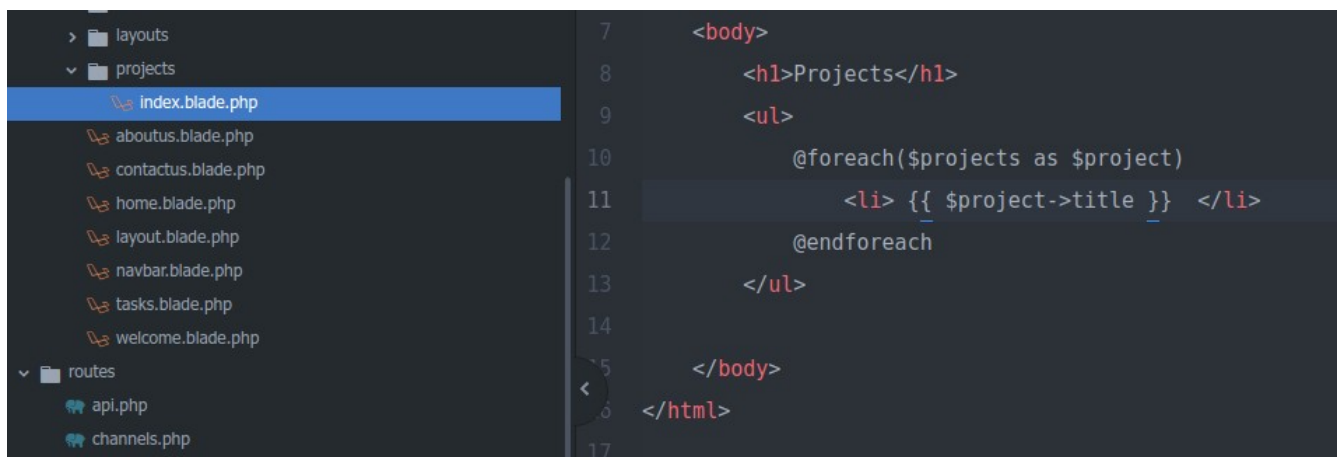


app\http\controllers\ProjectsController.php

```
class ProjectsController extends Controller
{
    public function index()
    {
        $projects = Project::all();
        return view('projects.index', compact('projects'));
    }
}
```

ကျွန်တော်တို့ resources/views/projects/index.blade.php ဆီကို projects ဆိုတဲ့ data လေးကို ပို့လိုက်မယ်။

view ဆောက်မယ်။ ဒါငွေတအားလုံး အ ရှေ့ သင်ခန်းစာငွေတ မှာရှင်းပြပြီးသား ဖြစ်တဲ့အွတ်ကံ အွတ်ကံကျကျမရှင်းပြ တော့ပါဘူး။



```
7 <body>
8 <h1>Projects</h1>
9 <ul>
10 @foreach($projects as $project)
11 <li> {{ $project->title }} </li>
12 @endforeach
13 </ul>
14
15 </body>
16 </html>
17
```

Projects

- My First Project
- My second project

အလုပ်လုပ်သွားတဲ့ workflow လေးပြန်ရှင်းပြမယ်။

1. route/web.php မှာ /projects လို့ ဝင်လာရင် ProjectsController.php ထဲက index function ကို run မယ်။
2. index function မှာ Project model အ နေနဲ့ project object ဖွဲ့တကို collection တစ်ခု အ နေနဲ့ ထုတ်ယူပြီး views/projects/index.blade.php ကို compact နဲ့ data ပို့တယ်။
3. index view file က နောက်ဆုံး ရရှိလာတဲ့ data ဖွဲ့တကို foreach loop လေးနဲ့ data ထုတ်ပြန်တယ်။

Topic 7 Form handling

ကျွန်တော်တို့ အခု Eloquent နဲ့ data ထုတ်ကြည့်လို့ ရသွားပါပြီ။ ဒါပေမယ့် data ထည့်ဖို့ ကျွန်တော်တို့မှာ web page မရှိသေးပါဘူး။ Project အသစ်လေးတည်ဆောက်ဖို့ အွတက် create.blade.php ဆိုပြီး form လေးတစ်ခုတည်ဆောက်ကြတာပေါ့။

```
public function create()
{
    return view('projects.create');
}
```

```
Route::get('/projects/create', 'ProjectsController@create');
```

ကျွန်တော်တို့ အရင်အတိုင်းပဲ Controller မှာ create ဆိုတဲ့ function နဲ့ view return ပြန်တယ်။ Route က ယင်း controller function ကို လှမ်းချိတ်တယ်။

views/projects/create.blade.php

```
<h1>Create a new project</h1>
<form action="/projects" method="post">
    Enter Project Title : <br> <input type="text" name="title"><br><br>
    Enter Project Description : <br> <textarea name="description"></textarea><br>
    <button type="submit">Create Project</button>
</form>
```

ဒီမှာကြည့်မယ်ဆိုရင် /projects ဆိုတဲ့ route ဆီကို data ပို့ထားတာဖြစ်ပါတယ်။ ဟာ /projects ဆိုတာ projects ငွတထုတ်ပြထားတာ ရှိပြီးသားကြီးလေလို့ တွေးနေနိုင်ပါတယ်။ ဒါပေမယ့်ဒီမှာ သတိထားရမှာက Get နဲ့ Post ကွာခြားချက်ပါ။ ကျွန်တော်တို့ project data ထုတ်ပြတုန်းက Get request နဲ့ /projects ဆီကိုသွားတာပါ။ ဒီမှာက တော့ Post request နဲ့ /projects ဆီကိုသွားတာပါ။ အဲ့ဒီတော့ route မှာလည်း post ဖြစ်သွားပါလိမ့်မယ်။


```
Route::get('/projects', 'ProjectsController@index');  
Route::get('/projects/create', 'ProjectsController@create');  
Route::post('/projects', 'ProjectsController@store');
```

function name တွေ route တွေကို မိမိနှစ်သက်ရာ ပေးလိုရတယ်ဆိုပေမယ့် ကျွန်တော်ပြထားတဲ့ အတိုင်း ထားတာကောင်းပါတယ်။ create ရယ် store ရယ် index ရယ် ဒါတွေကို ဧဟတာပါ။ ကျွန်တော်တို့ Team member တွေ အလုပ်လုပ်ရင် လူတိုင်း အသုံးပြုနေတဲ့ RESTful approach ကို သုံးသင့်ပါတယ်။ ဒါမှ အလုပ်ဝင်တဲ့အခါ အဆင်ပြေတာပေါ့။ REST stands for representational state transfer. REST အကြောင်းနောက်သင်ခန်းစာ မှာ အတင်းကျကျရှင်းပြပါမယ်။

အခုလောလောဆယ် store function မှာ ဘာမှ ထူးထူးခြားခြား မ ရေးဘဲ dd('stored') ဆိုပြီး output လေးထုတ်လိုက်ရ အောင် dd ဆိုတာ die and dump ပါ။ အလုပ်လုပ်တာတွေ အကုန်ရပ်ပြီး stored ဆိုတာလေးကို ထုတ်ပြပါဆိုတဲ့ အဓိပ္ပာယ်ဖြစ်ပါတယ်။ dd ကို error debug လုပ်တဲ့နေရာတွေမှာ သုံးပါတယ်။

ProjectsController.php

```
public function store()  
{  
    dd("stored");  
}
```

ကျွန်တော်တို့ createproject form လေးမှာ createproject ဆိုတဲ့ button ကိုနှိပ်ကြည့်လိုက်ပါ။

Create a new project

127.0.0.1:8000/projects

Enter Project Title :

Some title

Enter Project Description :

Some description

Create Project

419 | Page Expired

419 page expired ဆိုတဲ့ error လေးဖြစ်ပါတယ်။ ဒါက တော့ laravel မှာ security vulnerability တွေကို လျော့ချဖို့အတွက် မဖြစ်မနေ သုံးရမယ့် `{ csrf_field() }` ဒါမှမဟုတ် အတိုခေါက် အနေနဲ့ `@csrf` ဆိုတဲ့ ဟာလေး ကျွန်တော်တို့ form မှာမပါလို့ပါ။ ကျွန်တော်တို့ laravel မှာ form တစ်ခုတည်ဆောက်တိုင်း `@csrf` ဆိုတာမဖြစ်မနေထည့်ရမှာပါ။

csrf ဆိုတာ Cross-Site Request Forgery ဖြစ်ပါတယ်။ သူဟာဘာလဲဆိုရင် authenticated (login ဝင်ပြီးသား) လူကို တစ်ဖက်ကမ်းက နေလှမ်းပြီး မလိုလားအပ်တဲ့ request တွေ ပေးပို့ အောင် ပြုလုပ်တဲ့ တိုက်ခိုက်နည်း တစ်မျိုးဖြစ်ပါတယ်။ ဒါကို ကာကွယ်ဖို့အတွက် laravel မှာ `csrf_field()` နဲ့ ကွဲကယ်ပါတယ်။

ကွဲကယ်ပုံကို အကြမ်းပြောရမယ်ဆိုရင် `@csrf` လို့ ထည့်လိုက်တဲ့ နေရာမှာ hidden input field လေးနဲ့ token လေးတစ်ခု ထည့်သွင်းပေးလိုက်ခြင်းဖြစ်ပါတယ်။ ကိုယ့်ဟာကိုယ် မျက်နှာမင်ကြည့်ချင်ရင်တော့ မိမိ webpage မှာ `ctrl + u` နှိပ်ပြီး page source က နေကြည့်နိုင်ပါတယ်။

```
<form action="/projects" method="post">
```

```
@csrf
```

```
Enter Project Title : <br> <input type="text" name="title"><br><br>
```

```
Enter Project Description : <br> <textarea name="description"></textarea><br>
```

```
<button type="submit">Create Project</button>
```

```
</form>
```

```
<form action="/projects" method="post">
```

```
<input type="hidden" name="_token" value="UCT9Nb7c0jP0uwWhmcXYybfkqktKCAhzM6Gc9Ks4">
```

```
Enter Project Description : <br> <textarea name="description"></textarea><br>
```

အခုအခါကျမှ create project နှိပ်ပြီး request submit လိုက်တဲ့အချိန်မှာ ကျွန်တော်တို့ dd လုပ်ထားတာလေးကို မြင်ရမှဖြစ်ပါတယ်။

← → ↻ ⓘ 127.0.0.1:8000/projects

📱 Apps 📶 If a device receiv... 🆕 New Tab

"stored"

ယခုအခါ ကျွန်တော်တို့ store() function ထဲကို ရောက်ပြီဆိုတာသိတော့ ကျွန်တော်တို့ တကယ် store လုပ်တာလေး လုပ်လိုက်ကြတာပေါ့။

```
public function store()
```

```
{
```

```
    $project = new Project();
```

```
    $project->title = request('title');
```

```
    $project->description = request('description');
```

```
    $project->save();
```

```
    return redirect('/projects');
```

```
}
```

php artisan tinker နဲ့ တုန်းကလုပ်သလိုမျိုးပါဘဲ။

Project object လေးတစ်ခုရ အောင်ကြည့်တယ်။ attribute နှစ်ခုကို request ထဲက နှုတ်ယူပြီး ထည့်သွင်းတယ်။ save လိုက်တယ်။ နောက်ဆုံးကြမှ /projects ဆီကို ပန်ပြီး redirect လုပ်လိုက်တယ်။ redirect ရဲ့ default method ဟာ get ဖြစ်ပါတယ်။ ကျွန်တော်တို့ စမ်းကြည့်တာပေါ့။

127.0.0.1:8000/projects/create

Apps If a device receiv... New Tab

Create a new project

Enter Project Title :
My third project

Enter Project Description :
Third project's Description

Create Project

127.0.0.1:8000/projects

Apps If a device receiv... New Tab

Projects

- My First Project
- My second project
- My third project

+ Options

id title description created_at updated_at

<input type="checkbox"/>	Edit Copy Delete	1	My First Project	some description	2019-06-08 09:47:57	2019-06-08 09:47:57
<input type="checkbox"/>	Edit Copy Delete	2	My second project	another description	2019-06-08 10:00:13	2019-06-08 10:00:13
<input type="checkbox"/>	Edit Copy Delete	3	My third project	Third project's Description	2019-06-08 11:53:39	2019-06-08 11:53:39

အောင်မြင်သွားပါပြီ။

Topic 8 Restful Approach

ကျွန်တော် အရှေ့မှာတုန်းက ပြောခဲ့တဲ့ REST (Representational state transfer) အကြောင်း ရှေ့မဆက်မီပြောချင်ပါတယ်။ REST ဟာ မည်သည့် Framework ဖွဲ့တမှာမဆို အသုံးများတဲ့ data လဲလှယ်နည်း၊ api ရေးနည်းလေးပါ။ ဒါကြောင့် project team အနေနဲ့လုပ်တဲ့အခါမှာ စံပုံစံအတိုင်း standard ဖြစ်အောင်ရယ်၊ နောင်ကိုယ်ဒီ project ကို မကိုင်တော့တဲ့အချိန်မှာ အခြား project member ဖွဲ့တ လွယ်လွယ်ကူကူ ဆက်လက် maintain လို့ရ အောင် တစ်ကမ္ဘာလုံး စံပုံစံ အနေနဲ့ အသုံးပြုနေကြတဲ့ REST ကို ကျွန်တော်တို့ လေ့လာကြတာပေါ့။ function name ဖွဲ့တကို ဖော်ပြတဲ့အတိုင်း စံပုံစံနဲ့ ဘဲ အလုပ်လုပ်သင့်ပါတယ်။

method route function

GET /projects (index)	projects အကုန်ထုတ်ကြည့်ရန်။
GET /projects/{id} (show)	project တစ်ခုချင်းဆီ details ထုတ်ကြည့်ရန်။
POST /projects (store)	project အသစ် ထပ်တိုး သိမ်းဆည်းရန်။
PATCH /projects/{id} (update)	project တစ်ခုကို ပြုပြင်ပြီးသိမ်းဆည်းရန်။
DELETE /projects/{id} (destroy)	project တစ်ခုကို ဖြတ်ရန်။
DELETE /projects (destoryall)	projects အားလုံးကို ဖြတ်ရန်။

ကျွန်တော်တို့ တကယ့် REST approach မှာဆို PUT ဆိုတာလေးပါပါသေးတယ်။ သူက PATCH နဲ့ တူတူပါဘဲ။ လူတွေက ဒီအတိုင်းပဲ သုံးတာပါ။ data ကို တစ်ခုလုံးပြောင်းလဲပြီး သိမ်းဆည်းရင် PUT ၊ တစ်စိတ်တစ်ပိုင်းပဲ ပြောင်းလဲတယ်ဆိုရင် PATCH ဆိုပြီး ခွဲသုံးကြတာပါ။ ဒီမှာတော့ အရမ်းရှုပ်ထွေးသားခြင်းမရှိအောင် PATCH ပဲထည့်ထားပါတယ်။

ကျွန်တော်တို့ အထက်မှာ ပြထားတာဟာ data သွားပြင်တာပဲရှိပါသေးတယ်။ data ကိုပြင်ဖို့ အွတ်ကံ form ငွတလိုအပ်တာဖြစ်တဲ့အွတ်ကံ form အွတ်ကံroute လေးငွတလည်း အလားတူ ရှိဖို့လိုပါတယ်။

method route function

GET /projects/create (create) Project အသစ်တည်ဆောက်ရန် form ခေါ်ခြင်း

GET /projects/{id}/edit (edit) Project တစ်ခု ပြုပြင်ရန် form ခေါ်ခြင်း

ကျွန်တော်တို့ route မှာ လိုအပ်တာတွေ ဖြည့်ဆည်းလိုက်ကြတာပေါ့။

```
Route::get('/projects', 'ProjectsController@index');
Route::get('/projects/create', 'ProjectsController@create');
Route::get('/projects/{project}', 'ProjectsController@show');
Route::post('/projects', 'ProjectsController@store');
Route::get('/projects/{project}/edit', 'ProjectsController@edit');
Route::patch('/projects/{project}', 'ProjectsController@update');
Route::delete('/projects/{project}', 'ProjectsController@destroy');
```

ကျွန်တော်တို့ ရေးသားထားတဲ့ route ငွတကို **php artisan route:list** သားစစ်ကြည့်လို့ရပါတယ်။


```
→ project php artisan route:list
```

Domain	Method	URI	Name	Action	Middleware
	GET HEAD	/		Closure	web
	GET HEAD	api/user		Closure	api,auth:api
	GET HEAD	projects		App\Http\Controllers\ProjectsController@index	web
	POST	projects		App\Http\Controllers\ProjectsController@store	web
	GET HEAD	projects/create		App\Http\Controllers\ProjectsController@create	web
	GET HEAD	projects/{project}		App\Http\Controllers\ProjectsController@show	web
	PATCH	projects/{project}		App\Http\Controllers\ProjectsController@update	web
	DELETE	projects/{project}		App\Http\Controllers\ProjectsController@destroy	web
	GET HEAD	projects/{project}/edit		App\Http\Controllers\ProjectsController@edit	web

ကျွန်တော်တို့ အပေါ်က route နှစ်ခု ကိုယ်တိုင်မရေးဘဲ Laravel ကို autogenerate လုပ်ခိုင်းလို့ရပါတယ်။

```
Route::resource('projects', 'ProjectsController');
```

ဒီတိုင်းလေးရေးလိုက်ရုံနဲ့ ကျွန်တော်တို့ အလားတူ route နှစ်ခု ရရှိမှာဖြစ်ပါတယ်။

ကျွန်တော်တို့ ဒီ projects အတွက် လိုအပ်သမျှ model နှစ်ခု migration နှစ်ခု controller နှစ်ခုကို အဆင့်ဆင့် လေ့လာခဲ့တာဖြစ်တဲ့အတွက် ဖြတ်လမ်းနည်းလေးတွေ မပြုဖြစ်ခဲ့ပါဘူး။ အခု အကုန်ပြီးသားပြီဖြစ်တဲ့အတွက် ဖြတ်လမ်းနည်းလေး လေ့လာကြည့်ပါမယ်။

ကျွန်တော်တို့ php artisan help make:controller လို့ရိုက်ကြည့်ရင် Options ဆိုပြီး ထည့်လို့ရတဲ့ ဟာတွေ အများကြီးထပ်ပြပါတယ်။ ကျွန်တော်တို့သာ controller တစ်ခုဆောက်တဲ့အချိန်မှာ -m (model) -r (resources) ထည့်လိုက်မယ်ဆိုရင်

php artisan make:controller ProjectsController -r -m Post

projects အတွက် controller နှစ်ခုအတူ model ၊ resourceful routes နှစ်ခု ProjectsController ထဲမှာ resources နှစ်ခုကို ထိန်းချုပ်ဖို့ function နှစ်ခု တစ်ခါတည်း ထည့်သင်းပေးသားမှာဖြစ်ပါတယ်။

Topic 9 Faking PATCH and DELETE Requests

ကျွန်တော်တို့ html form method attribute မှာ request method အနေနဲ့ GET နဲ့ POST နှစ်ခုရတယ်ဆိုတာကို php basics မှာ သိပြီးသားဖြစ်မှာပါ။ ဒါပေမယ့် PATCH နဲ့ DELETE ဟာ အခုလက်ရှိ HTML5 မှာ မရသေးပါဘူး။ ဒါကြောင့် ကျွန်တော်တို့ ၎င်း ပြဿနာကို ဖြေရှင်းကြည့်ကြတာပေါ့။

အရှေ့မှာ route ဖွဲ့စည်း အကုန်ဖြစ်ပြီး

Route::resource('projects', 'ProjectsController'); ဆိုပြီးရေးထားတော့ အောက်မှာ အလုပ်လုပ်တဲ့ workflow မမြင်မှာဆိုးလို့ route လေးပြထားပါမယ်။

```
Route::get('/projects', 'ProjectsController@index');
Route::get('/projects/create', 'ProjectsController@create');
Route::get('/projects/{project}', 'ProjectsController@show');
Route::post('/projects', 'ProjectsController@store');
Route::get('/projects/{project}/edit', 'ProjectsController@edit');
Route::patch('/projects/{project}', 'ProjectsController@update');
Route::delete('/projects/{project}', 'ProjectsController@destroy');
```

ကျွန်တော်တို့ အခုရရှိစိုက်ကမှာက /projects/{id}/edit ကိုလာရင် ProjectsController ထဲက edit function နဲ့ form ထုတ်ပြမယ်၊ form က /projects/{id} ဆီကို PATCH Request နဲ့ submit လိုက်မယ်ဆိုရင် update function ကို run ပြီး data ဖွဲ့စည်း update လုပ်ပြီး ပြပြင်ပြီးသားဖြစ်သွားမယ်။ /projects/{id} ကိုသွားရင် show function run ပါမယ်။

အောက်မှာ URL ဖွဲ့စည်း နားလည်ရခက်ပြီး မမြင်ရင် ဒီ စာမျက်နှာကိုလာပြီး route က လေး ဖွဲ့စည်း ပြန်ကြည့်လို့ ရပါတယ်။

ကျွန်တော်တို့ အရင်ဦးစွာ ပုံမှန်အတိုင်းဘဲ ကျွန်တော်တို့ create လုပ်ပြီးသား project လေးတွေကို ပြုပြင်လို့ရမယ့် Form လေးတစ်ခုဆောက်လိုက်ကြတာပေါ့။

ဒီ code file (edit.blade.php) ကို ProjectsController.php က edit function မှာ အရင်ဦးစွာ data လေးပို့ပြီး view လေး ထုတ်ပြတယ်။

ProjectsController.php

```
public function edit($id)
{
    $project = Project::find($id);
    return view('projects.edit', compact('project'));
}
```

\$id ဆိုတာ /projects/{ \$id } ဆိုပြီး route ကလာတာကို argument အနေနဲ့ လက်ခံထားတာပါ။ \$project ထဲကို Project ဖွဲ့တထဲက \$id နဲ့ ညှိတဲ့ဟာကို လက်ခံတယ်။ compact နဲ့ data ပေးပို့လိုက်တယ်။

edit.blade.php

```
<form action="/projects/{{ $project->id }}" method="patch">
    @csrf
    Enter title : <br> <input type="text" name="title" value="{{ $project->title }}"> <br><br>
    Enter Description : <br> <textarea name="description"> {{ $project->description }} </textarea><br>
    <button type="submit" name="button">Edit Project</button>
</form>
```

/project/1/edit ကို Browser မှာသားပြီး edit project button ကိုနှိပ်လိုက်ရင်

127.0.0.1:8000/projects/1?_token=wWtTHqSWtZlXSN40rxYva0VWtyunad3qrRe8G7yV&title=My+First+Project&description=+some+description+&button=

ဒီတိုင်းဘာမှမဖြစ်ဘဲ GET request method နဲ့ဘဲ request ပေးပို့တာကို ဖွဲ့တ.ရမှပါ။ ဒါက ဘာလို့လဲဆိုတော့ Form method မှာ patch လို့ ပေးလိုက်တာကို Browser က မသိတဲ့အတွက် default get method နဲ့ဘဲ request ပေးပို့သားတာပါ။ ဒီပြဿနာကို ရှင်းဖို့အတွက် form method ကို POST ဘဲပြောင်းပြီး FORM အတွင်းမှာ {{ method_field('PATCH') }} ဒါမှမဟုတ် အတိုခေါက် @method('PATCH') ဆိုပြီး ထည့်ပေးကပါတယ်။ အလုပ်လုပ်ပုံက @csrf တုန်းကလိုမျိုးပါဘဲ။ HIDDEN INPUT လေးတိုးတိုး တိတ်တိတ်လေး ဒီ REQUEST ဟာ PATCH ပါဆိုပြီးပြောလိုက်တာပါ။

edit.blade.php

```
<form action="/projects/{{ $project->id }}" method="post">
    @method('PATCH')
    @csrf
    Enter title : <br> <input type="text" name="title" value="{{ $project->title }}"> <br><br>
    Enter Description : <br> <textarea name="description"> {{ $project->description }} </textarea><br>
    <button type="submit" name="button">Edit Project</button>
</form>
```

action က တော့ *projects/{ \$id }* (ဒီ case မှာက *projects/1*) ကို patch request နဲ့ form က data ဖွဲ့တ ပေးပို့လိုက်ပါလို့ အဓိပ္ပာယ်ရောက်ပါတယ်။

ကျွန်တော်တို့ route မှာတုန်းက resource နဲ့ ပေးခဲ့တာဖြစ်တဲ့အတွက် ကျွန်တော်ပေးတဲ့ standard name ဖွဲ့တ function ဖွဲ့တအတိုင်းသာရေးမယ်ဆိုရင် route မှာသားပြင်စရာ ဘာညာ မလိုတော့ပါဘူး။

အခု /project/{ $\$id$ } (/projects/1) ကို PATCH REQUEST နဲ့ data ဖွဲ့တ ပေးလိုက်ပြီဆိုတော့ update function ကို run မှာဖြစ်ပါတယ်။ အဲ့ဒီအတွက် update function လေးရေးလိုက်ကြရအောင်။

ProjectsController.php

```
public function update( $\$id$ )
{
     $\$project$  = Project::find( $\$id$ );

     $\$project$ ->title = request('title');
     $\$project$ ->description = request('description');
     $\$project$ ->save();
    return redirect('/projects');
}
```

ထူထူးခြားခြားမပါပါဘူး။ $\$id$ နဲ့ project ရှာတယ်။ ရှာဖွေတော့ project ရဲ့ title ဖွဲ့တ description ဖွဲ့တ ပြောင်းပြီး redirect လုပ်လိုက်တယ်။ အရင်ကကျွန်တော်တို့ သိပြီးသားဟာ ဖွဲ့တပါဘဲ။

စမ်းကြည့်ကြတာပေါ့။

← → ↻ ⓘ 127.0.0.1:8000/projects/1/edit

Apps If a device receiv... New Tab

Enter title :

Enter Description :

← → ↻ ⓘ 127.0.0.1:8000/projects

Apps If a device receiv... New Tab

Projects

- My First Project Changed
- My second project
- My third project

ကျွန်တော်က ဒီမှာတစ်ခါတည်း edit လုပ်လို့ရ အောင် link လေးလုပ်ထားတာပါ။

index.blade.php

```
@foreach($projects as $project)
    <li>
        <a href="/projects/{ {{ $project->id }} " style="text-decoration: none;"> {{ $project->title }} </a>
        <a href="/projects/{ {{ $project->id }} /edit"><button>Edit</button></a>

    </li>
@endforeach
```

edit button ကိုနှိပ်ရင် /projects/{id}/edit ကို ရောက်ပြီး form ထုတ်မယ်။

Project name လေးကို နှိပ်ရင် /projects/{id} ကို ရောက်ပြီး show method run မယ်။

ProjectsController.php

```
public function show($id)
{
    $project = Project::find($id);
    return view('projects.show', compact('project'));
}
```

show.blade.php


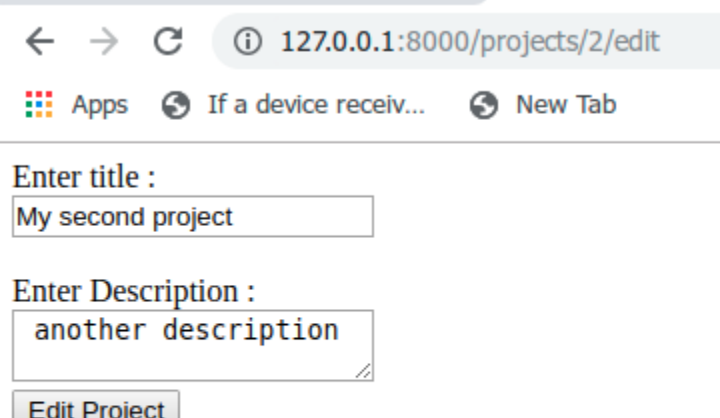
```
<body>
    <h1>{{ $project->title }}</h1>
    <p> {{ $project->description }} </p>
</body>
```

ထူးထူးခြားခြားမဟုတ်တော့ မရှင်းပြ တော့ပါဘူး။ ကိုယ့်ဟာကိုယ် link လေးငွေတ နှိပ်ပြီးစမ်းကြ ကည့်လိုက်ကြပါ။

index.blade.php (/projects) မှာ

project name ကို နှိပ်ရင်

edit ခလုတ်ကို နှိပ်ရင်

127.0.0.1:8000/projects/2	127.0.0.1:8000/projects/2/edit
	

ကျွန်တော်တို့ အခု delete လုပ်နည်းလေးကြည့်လိုက်ကြရ အောင်။

show.blade.php

```
<body>
  <h1>{{ $project->title }}</h1>
  <p> {{ $project->description }} </p>
  <a href="/projects/{{ $project->id }}/edit"><button>Edit</button></a>
  <form action="/projects/{{ $project->id }}" method="post">
    @method('DELETE')
    @csrf
    <button type="submit">Delete</button>
  </form>
</body>
```

ဒီမှာ form လေးတစ်ခု နဲ့ @method('delete') လေးနဲ့ /projects/{id} ဆီကို button နှိပ်လိုက်ချိန်မှာ delete request လေးပေးပို့သွားမှာပါ။ route မှာက /projects/id ကို delete request ဝင်လာရင် ProjectsController ရဲ့ destroy method ကိုအလုပ်လုပ်မှာဖြစ်တဲ့အတွက် destroy method လေးရေးကြတာပေါ့။

ProjectsController.php

```
public function destroy($id)
{
    Project::find($id)->delete();
    return redirect('/projects');
}
```

\$id လေးနဲ့ project ရှာတယ်။

ဖြတ်လိုက်တယ်။ /projects

ကို ပြန် redirect လိုက်တယ်။

ဒါပါဘဲ။

ကျွန်တော်တို့ Project အသစ်တစ်ခု တည်ဆောက်လိုရပြီ (Create)၊ Project ဖွဲ့တကို အားလုံး ၊ တစ်ခုချင်းဆီ ထုတ်ကြည့်လိုရပြီ (Retrieve) ၊ ရှိပြီးသား Project ကို ပြုပြင်လိုရပြီ (Update)၊ ဖြတ်လိုရပြီ (Delete)။

ကယ်ဒါဆို CRUD function လေး အကုန်ဆွဲသားပါပြီ။

အခုပြချင်တာက တော့ လိုအပ်လားဆိုတော့ မလိုပါဘူး။ ဒါပေမယ့် code လေးပိုသန့်ပြီး လယ်ကူတဲ့ နည်းလမ်းလေး ပြချင်တာပါ။ ကျွန်တော်တို့ ဥပမာအ နေနဲ့ show function လေးကို ကြည့်ကြတာပေါ့။

```
public function show($id)
{
    $project = Project::find($id);
    return view('projects.show', compact('project'));
}
```

\$id လက်ခံတယ်
\$id နဲ့ရှာတယ်။ အ
လုပ်ရှုပ်ပါတယ်။

ကျွန်တော်တို့ typehinting ဆိုတဲ့ နည်းပညာအသုံးပြုပြီး ဒီလိုသုံးလိုရပါတယ်။

```
public function show(Project $project)
{
    return view('projects.show', compact('project'));
}
```

မိမိနှစ်သက်ရာ နည်းလမ်းအသုံးပြုပါ။

Topic 10 Mass assignment protection and Two layers of Form validation

အခုလက်ရှိ create နဲ့ update မှာ textbox အလတ်ထားပြီး submit ကြည့်လိုက်ရင် Error တက်ပါလိမ့်မယ်။

Create a new project

Enter Project Title :

Enter Project Description :

Create Project

Illuminate \ Database \ **QueryException** (23000)

SQLSTATE[23000]: Integrity constraint violation: 1048 Column 'title' cannot be null (SQL: insert into `projects` (`title`, `description`, `updated_at`, `created_at`) values (?, ?, 2019-06-09 07:59:25, 2019-06-09 07:59:25))

Previous exceptions

Application frames (2) All frames (66)

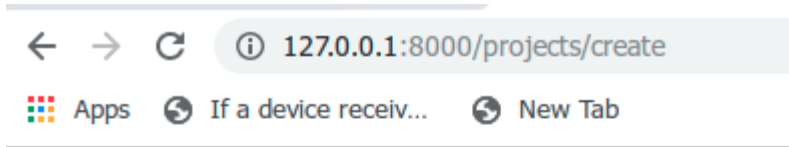
ဤကဲ့သို့ error ဖွတ် မတက်အောင်ရယ် မလိုလားအပ်တဲ့ data ဖွတ် ကျွန်တော်တို့ database အတွင်း မဝင်လာနိုင်အောင်ရယ် ကွဲကယ်ဖို့အတွက် ဒီသင်ခန်းစမှာ ဆက်လေ့လာကြတာပေါ့။

အရင်ဦးစာ mass assignment protection လေး လေ့လာကြတာပေါ့။

ကျွန်တော်တို့ store function လေးကို ဒီပုံစံလေးပြင်လိုက်ကြရ အောင်။

```
public function store()
{
    Project::create([
        'title' => request('title'),
        'description' => request('description')
    ]);
    return redirect('/projects');
}
```


ကျွန်တော်တို့ အခု Project အသစ်တည်ဆောက်တဲ့အခါ error တက်ပါလိမ့်မယ်။



Create a new project

Enter Project Title :

Enter Project Description :

Illuminate \ Database \ Eloquent \ **MassAssignmentException**

Add [title] to fillable property to allow mass assignment on [App\Project].



COPY

ဒါက security ပိုင်းအရ database ထဲကို ဘာလာထည့်ထည့် လက်မခံဘဲကွက်ယထားတာ ပါ။ အဲ့ဒီတော့ ကျွန်တော်တို့ Project model မှာ ဘာကို လက်ခံမလဲဆိုတာကို ကြေညာရပါမယ်။

app/Project.php

```
class Project extends Model
{
    protected $fillable = [
        'title', 'description'
    ];

    protected $guarded = ['id'];
}
```

ကျွန်တော် ဒီမှာ နှစ်ခု ကြေညာပြထားပါတယ်။ တကယ်တော့ တစ်ခုဘဲလုပ်ကမှာပါ။ မိမိနှစ်သက်ရာသုံးပေါ့။

\$fillable ကြေညာတာဟာ ဒီကြေညာထားတဲ့ field နှစ်ခုထည့်ခွင့်ရှိပါတယ်ဆိုတဲ့သဘောပါ။

\$guarded ကြေညာတာဟာ ဒီ field နှစ်ခု ပေးမထည့်ဘူးဆိုတဲ့သဘောပါ။

ကျွန်တော်တို့ \$fillable ဒါမှမဟုတ် \$guarded တစ်ခုခု ထည့်ပြီးပြီဆိုရင် ရသွားမှာပါ။

ဒါဆို Mass assignment protection လုပ်ပြီးသားပါပြီ။ ကျွန်တော် ဥပမာ အ နေနဲ့ ပြောရမယ်ဆိုရင် ကျွန်တော်တို့ website မှာ subscription plan ရှိခဲ့သည်ရှိသော်။ subscribe လုပ်ထားလား မလုပ်ထားလား field ကို လာပြီး assign လုပ်လို့မရ အောင် ကာကွယ်တဲ့ အခါမျိုးတွေမှာ သုံးပါတယ်။

ကျွန်တော်တို့ update ကိုလည်း သန်အောင် ပြုပြင်လိုက်ရ အောင်။

```
public function update(Project $project)
{
    $project->update(request(['title', 'description']));
    return redirect('/projects');
}
```

ကယ် အခု Form validation လေး စလိုက်ကြရ အောင်။

ကျွန်တော်တို့ form ကို ပုံမှန်အရဆို validation ကို javascript, JQuery နဲ့ validate လုပ်ပြီးမှ submit လို့ရအောင် လုပ်လို့ရပါတယ်။ basic HTML နဲ့တောင် input tag နောက်ဆုံးမှာ required ဆိုပြီး ထည့်လို့ရပါတယ်။ ဒါပေမယ့် script load မဖြစ်တာဘဲဖြစ်ဖြစ် user က နေ script disable ပြီး input tag က ဟာကို edit as html နဲ့ ဖြတ်ပြီး form submit မယ်ဆိုရင် error တက်သွားမှာဖြစ်ပါတယ်။

အဲ့လို ပြဿနာ တွေကို ရှင်းဖို့အတွက် form validation ကို နောက်ထပ် layer လေးတစ်ခု ထပ်တိုးပြီး laravel ဘက်က နေပြီး ထပ်ပြီး validation လေးလုပ်မှာဖြစ်ပါတယ်။

လုပ်ပါသည်။ ProjectsController ရဲ့ store မှာ အောက်ပါအတိုင်း rules လေးတွေ သတ်မှတ်လိုက်ရပါမည်။

ProjectsController.php

```
public function store()
{
    $attributes = request()->validate([
        'title' => ['required', 'min:3', 'max:255'],
        'description' => ['required', 'min:3', 'max:255']
    ]);
    Project::create($attributes);
    return redirect('/projects');
}
```

ကျွန်တော်တို့ Error တက်ရင် ထုတ်ပြဖို့ဆိုရင် \$errors ဆိုတဲ့ Variable လေး laravel မှာရှိပါတယ်။ error ရှိခဲ့ရင် background အနီ ရောင်လေးနဲ့ ထုတ်ပြသွားမှာပါ။

```
@if ($errors->any())
    <div style="background-color:red">
        @foreach($errors->all() as $error)
            <li>{{ $error }}</li>
        @endforeach
    </div>
@endif
```

စမ်းသပ်ကြည့်ကြတာပေါ့။

Create a new project

Enter Project Title :

Enter Project Description :

Create Project

- The title field is required.
- The description field is required.

Create project button နှိပ်လိုက်တဲ့အခါ ကျွန်တော်တို့ validation မှာ ချမှတ်ထားတဲ့ rules နဲ့ မကိုက်ညီတဲ့ error မှာ ထုတ်ပြသွားမှာဖြစ်ပါတယ်။