Laravel Controllers

Laravel controllers are another essential feature in a Laravel framework just like the route feature we saw on the last session.

Initially, we were handling the request logic in the form of closures in route files; now, in place of using closures in route files, we use controller classes.

Controllers are used to handle the request logic within the single class, and the controllers are defined in the "`app/http/Controllers`" directory. Laravel framework follows the MVC (Model View Controller) architecture in which controllers act as moving the traffic back and forth between model and views.

The default file of controller is available in the app/http/Controllers directory.

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Foundation\Auth\Access\AuthorizesRequests;
use Illuminate\Foundation\Bus\DispatchesJobs;
use Illuminate\Foundation\Validation\ValidatesRequests;
use Illuminate\Routing\Controller as BaseController;

class Controller extends BaseController
{
   use AuthorizesRequests, DispatchesJobs, ValidatesRequests;
}
```

In the above code, the namespace is used as it allows you to use the same function names and classes in the different parts of the same application. For example,

```
namespace App\Http\functions1;
namespace App\Http\functions2;
```

Suppose we have to run the function having the name, i.e., RunQuery(). They are available in different directories functions1 and functions2, so we can say that namespace avoids the collision between the same function names.

`'use'` is used to import the class to the current file.

Let's see how to create the controller through terminal Window.

Step 1: Open the Terminal and type the command

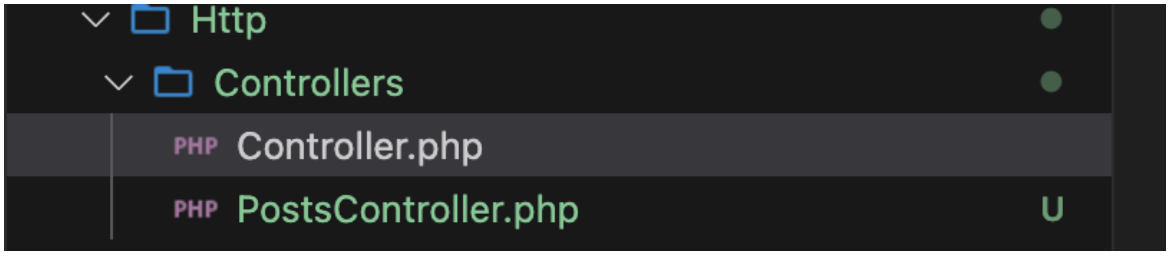
```
php artisan make:Controller PostsController
```

in Terminal to create the Controller.

```
@PC training-class-contents % php artisan make:Controller PostsController

INFO  Controller [app/Http/Controllers/PostsController.php] created successfully.
```

The above screen shows that the controller named as `PostsController` has been created successfully.

Step 2: Now move to your project and see whether the `PostsController` file has been created or not.



The above screen shows that the `PostsController` file is created.

The default code of `PostsController.php` file is given below:

```php
<?php
namespace App\Http\Controllers;
use Illuminate\Http\Request;

class PostsController extends Controller
{

}
```

The above code contains the class that extends the Controller class, but this class does not contain the functions such as create, update, or delete. Now we will see how to create the controller which contains some default functions.

To create the Controller, we will first delete the `PostsController.php` from the project, which we have created in the previous step.

```
php artisan make:controller PostController --resource
```

this command is used to create the controller.

The default code of the PostController.php after creating it by –resource option like above

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class PostController extends Controller
{
   /**
    * Display a listing of the resource.
    *
    * @return \Illuminate\Http\Response
    */
   public function index()
   {
       //
   }
   /**
    * Show the form for creating a new resource.
    *
    * @return \Illuminate\Http\Response
    */
   public function create()
   {
       //
   }

   /**
    * Store a newly created resource in storage.
    *
```

```
     * @param  \Illuminate\Http\Request  $request
     * @return \Illuminate\Http\Response
     */
    public function store(Request $request)
    {
        //
    }
    /**
     * Display the specified resource.
     *
     * @param  int  $id
     * @return \Illuminate\Http\Response
     */
    public function show($id)
    {
        //
    }
    /**
     * Show the form for editing the specified resource.
     *
     * @param  int  $id
     * @return \Illuminate\Http\Response
     */
    public function edit($id)
    {
        //
    }
    /**
     * Update the specified resource in storage.
     *
     * @param  \Illuminate\Http\Request  $request
     * @param  int  $id
     * @return \Illuminate\Http\Response
     */
    public function update(Request $request, $id)
    {
        //
    }
    /**
     * Remove the specified resource from storage.
     *
     * @param  int  $id
     * @return \Illuminate\Http\Response
     */
    public function destroy($id)
    {
        //
    }
```

```
}
```

The above code contains the functions which are used to perform the various operations on the resources such as:

- `create()`: It is used to create a new resource.
- `store()`: It is used to store the specified resource.
- `update()`: It is used to update the specified resource in the storage.
- `destroy()`: It is used to remove the specified resources from the storage.

Routing Controllers

Routing controllers allow you to create the controller classes with methods used to handle the requests.

Step 1: First, we need to create a controller. We already created the controller named as '`PostController`' in the previous topic.

Step 2: Open the web.php file and write the following code:

```
Route::get('/post','\App\Http\Controllers\PostController@index');
```

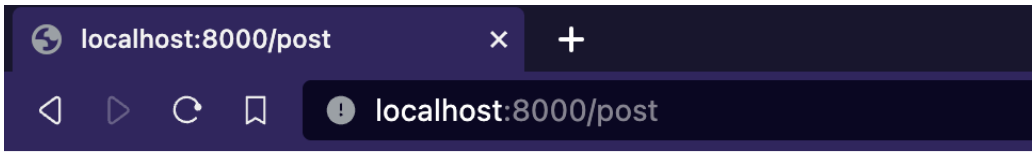
In the above code, '`/post`' is the URL that we want to access, and `PostController` is the name of the controller. The `'index'` is the name of the method available in the `PostController.php` file, and `@index` indicates that the `index()` method should be hit when we access the `'/post`' url.

Step 3: replace the index method from PostController with the below code

```
public function index()
{
  return "Hello Everyone";

}
```

Step 4: lets browse for `/post`

Output



Hello Everyone

Passing data to the Controller

Step 1: Open the web.php file and add the following code:

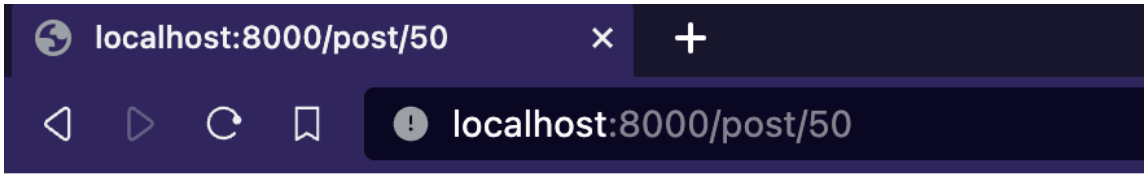
```
Route::get('/post/{id}','\App\Http\Controllers\PostController@show');
```

The above code contains the `'id'` parameter in the `'/post'` url.

Step 2: Edit the PostController.php file and update the `show(id)` method

```
public function show($id)
{
    return "ID is :". $id;
}
```

Step 3: lets see it in the browser by browsing to `/post/50`



ID is :50

Laravel Resource Controllers

Laravel resource controllers provide the CRUD routes to the controller in a single line of code. A resource controller is used to create a controller that handles all the http requests by your application.

The resource() is a static function like get() method that gives access to multiple routes that we can use in a controller.

Syntax of resource() method:

```
Route::resource('posts','\App\Http\Controllers\PostController');
```

In the above syntax, `'posts'` contains all the routes, and `'PostController'` is the name of the controller. In this case, we do not need to specify the method name such as `@index` as we did in `get()` method because `create(), store(), destroy()` methods are already available in the `PostController` class.

Step 1: Create the controller by using the command given below:

```
php artisan make:controller PostController --resource
```

The above command will create the Controller at the app/Http/Controllers/PostController.php directory. The PostController class contains the methods for each resource operations.

Step 2: Now, we need to register the resourceful route to the Controller, and which can be done as follows:

```
Route::resource('posts','\App\Http\Controllers\PostController');
```

Open the Terminal, and enter the command

```
php artisan route:list
```

This command produces the following output:

```
GET|HEAD        post ..... PostController@index
GET|HEAD        post/{id} ..... PostController@show
GET|HEAD        posts ..... posts.index › PostController@index
POST            posts ..... posts.store › PostController@store
GET|HEAD        posts/create ..... posts.create › PostController@create
GET|HEAD        posts/{post} ..... posts.show › PostController@show
PUT|PATCH       posts/{post} ..... posts.update › PostController@update
DELETE          posts/{post} ..... posts.destroy › PostController@destroy
GET|HEAD        posts/{post}/edit ..... posts.edit › PostController@edit
```

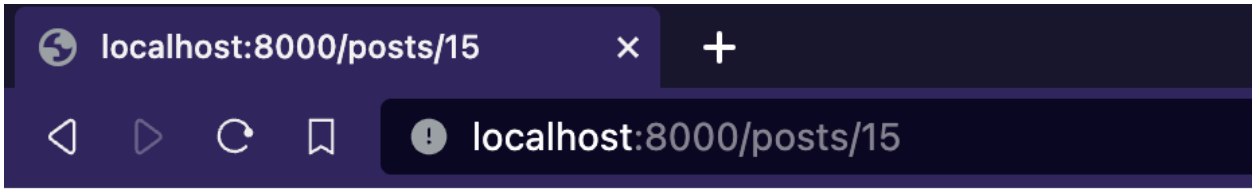
The post parameter in the `resource()` method produces the names or resources shown in the above output, and its corresponding methods.

Accessing the show() method of PostController class

Suppose we want to call the show() method of `PostController.php` file. To do so, add the code in `show()` method. I added the following code in `show()` method:

```
public function show($id)
 {
     return "show method is called and ID is : ". $id;
 }
```

Lets See this with `/posts/15`



ID is :15

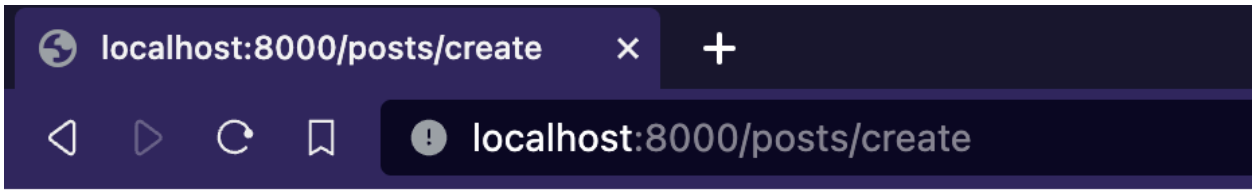
Accessing the create() method of PostController class

Step 1: First, we need to add the code in create() method. I added the following code:

```
public function create()
 {
     return "This is the create method";
 }
```

As we know that the URI of the `posts.create` is `posts/create`, so the URL to access the `create()` method would be '`/posts/create`'

Step 2: Enter the URL '`/posts/create`' to the browser, then the output should be:



This is the create method

Registering routes for multiple controllers

We can register the routes for multiple controllers by passing an array to the resources() method.
Suppose I want to register the routes for two controllers, such as PostController and StudentController.

Following are the steps to achieve this:

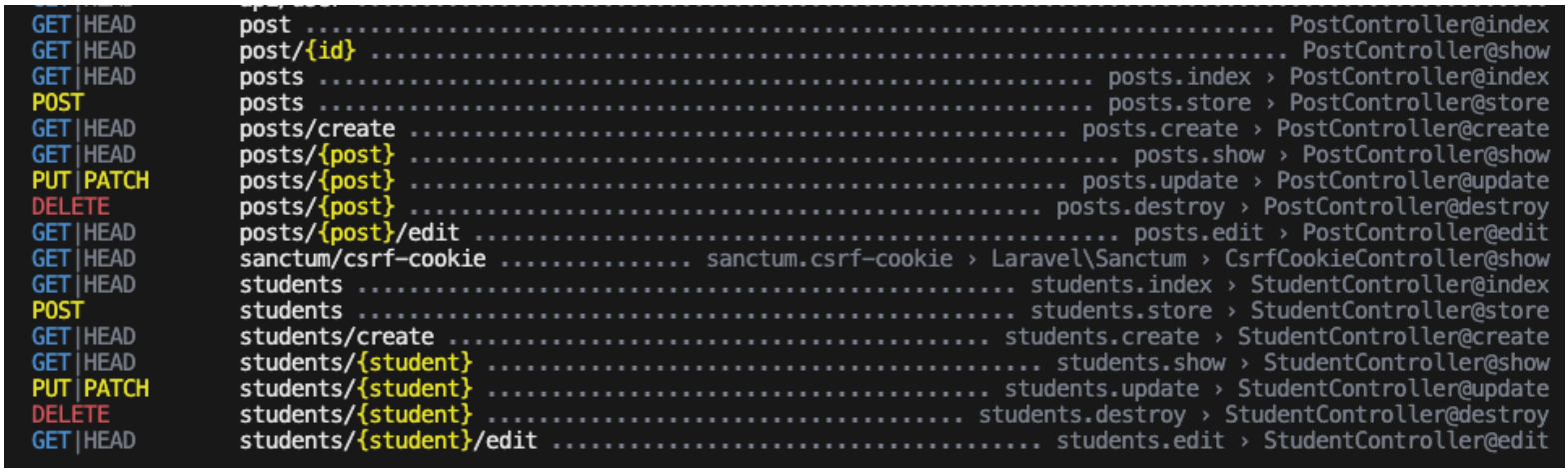
Step 1: First, you need to create the `PostController` and `StudentController` by using the following commands:

```
php artisan make:controller StudentController --resource
php artisan make:controller PostController --resource
```

Step 2: Add the code given below in web.php file to register routes:

```
route::resources(
        [
        'posts'=>'\App\Http\Controllers\PostController',
        'students'=>'\App\Http\Controllers\StudentController'
        ]
   );
```

Step 3: Enter the command `php artisan route:list` on Terminal.



The above screen shows that routes of both the `PostController` and `StudentController` are registered.

Partial Resource Routes

When we do not want to register the routes for all the methods, then we can do so by specifying only those functions that the controller can handle.

Steps to create the Partial Resource Routes:

Step 1: First, we create the StudentController by using the below command:

```
php artisan make:controller StudentController --resource
```

Step 2: Now, we add the following command in web.php file to create the Partial resource routes

```
Route::resource('student','\App\Http\Controllers\StudentController',
[
   'only' => ['create','show']
]);
```

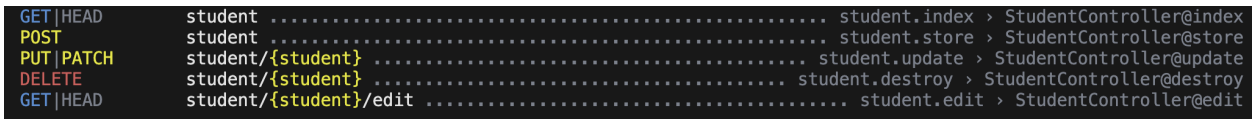
Step 3: To verify whether the above code has registered the routes for the specified methods or not, type the command '`php artisan route:list`' on Terminal.

The above screen shows that the routes for `create()` and `show()` methods have been generated.

We Can Also Exclude the controller methods using `except`

```
Route::resource('student','\App\Http\Controllers\StudentController',
[
    'except' => ['create','show']
]);
```

To verify whether the above code has registered the routes and excluded the specified create and show methods , type the command '`php artisan route:list`' on Terminal.



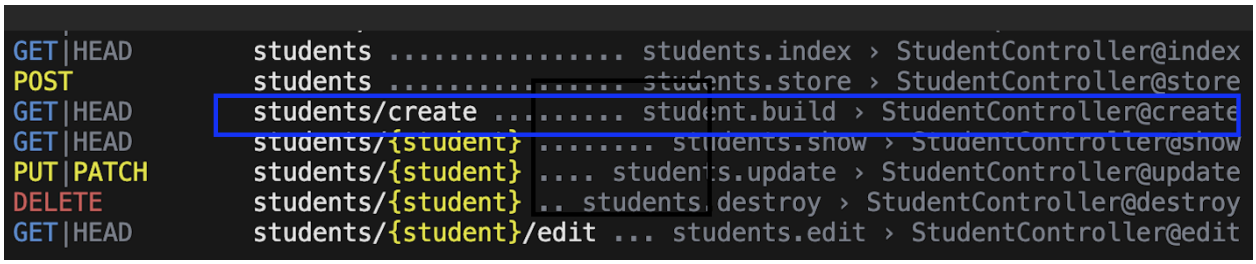
Naming Resource Routes

All the methods of the controller have a default route name, but Laravel allows you to override the route names by passing name array. Name array contains the name of the routes that you want to specify of your choice.

We can add the below code in web.php file to name the resource routes.

```
Route::resource('students', '\App\Http\Controllers\StudentController',
[
   'names' => ['create' =>'student.build']
]);
```

Now, enter the command `php artisan route:list` on Terminal.



The above screen shows that the route name of the `create()` method has been renamed as `students.build`, and its default name was `students.create`.

Controller method that returns different responses

A laravel controller can contain different methods. Each method can have $request argument as a default function parameter. This is basically a request object that contains your form variables, session data or cookie information.

A controller method can respond via different response type some of them are mentioned as below:

- Plain Text
- Json Array
- blade view
- File download
- PDF or Image output on browser directly etc…

Lets First Create a new controller to show the different return values

```
php artisan make:controller TestController
```

Render Plain Text

We have already worked by returning text from controller until now.

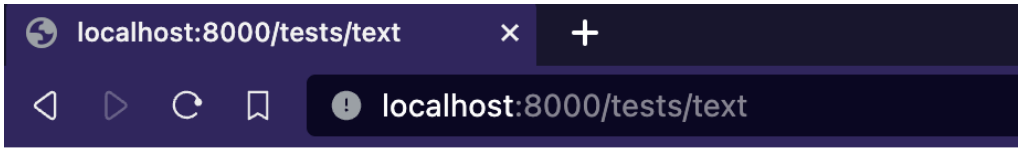
Lets add a new method inside Testcontroller to return plain text

```
function text()
{
    return "welcome To Our Website";
}
```

Lets add a route definition inside route/web.php

```
Route::get('/tests/text','\App\Http\Controllers\TestController@text');
```

And lets see the response on our browser

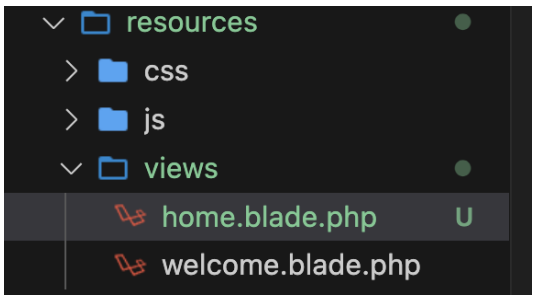


welcome To Our Website

Return a blade view

Lets add a new method inside Testcontroller to return a blade view from `resources/views` folder

Lets first create a new blade file inside `resources/view` folder and lets call it `home.blade.php` and add the below code inside it



Add this in the `home.blade.php` file

```
<!doctype html>
<html lang="en">
 <head>
   <!-- Required meta tags -->
   <meta charset="utf-8">
   <meta name="viewport" content="width=device-width, initial-scale=1">

   <!-- Bootstrap CSS -->
   <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesheet"
integrity="sha384-1BmE4kWBq78iYhFldvKuhfTAU6auU8tT94WrHftjDbrCEXSU1oBoqyl2QvZ6jIW3" crossorigin="anonymous">

   <title>Home Page</title>
 </head>
 <body>
   <nav class="navbar navbar-expand-lg bg-body-tertiary bg-dark border-bottom border-body"
data-bs-theme="dark">
       <div class="container-fluid">
         <a class="navbar-brand" href="#">Navbar</a>
```

```
        <button class="navbar-toggler" type="button" data-bs-toggle="collapse"
data-bs-target="#navbarSupportedContent" aria-controls="navbarSupportedContent" aria-expanded="false"
aria-label="Toggle navigation">
          <span class="navbar-toggler-icon"></span>
        </button>
        <div class="collapse navbar-collapse" id="navbarSupportedContent">
          <ul class="navbar-nav me-auto mb-2 mb-lg-0">
            <li class="nav-item">
              <a class="nav-link active" aria-current="page" href="#">Home</a>
            </li>
            <li class="nav-item">
              <a class="nav-link" href="#">Link</a>
            </li>
            <li class="nav-item dropdown">
              <a class="nav-link dropdown-toggle" href="#" role="button" data-bs-toggle="dropdown"
aria-expanded="false">
                Dropdown
              </a>
              <ul class="dropdown-menu">
                <li><a class="dropdown-item" href="#">Action</a></li>
                <li><a class="dropdown-item" href="#">Another action</a></li>
                <li><hr class="dropdown-divider"></li>
                <li><a class="dropdown-item" href="#">Something else here</a></li>
              </ul>
            </li>
            <li class="nav-item">
              <a class="nav-link disabled" aria-disabled="true">Disabled</a>
            </li>
          </ul>
          <form class="d-flex" role="search">
            <input class="form-control me-2" type="search" placeholder="Search" aria-label="Search">
            <button class="btn btn-outline-success" type="submit">Search</button>
          </form>
        </div>
      </div>
    </nav>



    <div class="jumbotron">
      <h1 class="display-4">Hello, world!</h1>
      <p class="lead">This is a simple hero unit, a simple jumbotron-style component for calling extra
attention to featured content or information.</p>
      <hr class="my-4">
      <p>It uses utility classes for typography and spacing to space content out within the larger
container.</p>
      <a class="btn btn-primary btn-lg" href="#" role="button">Learn more</a>
    </div>
```

```
   <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js"
integrity="sha384-ka7Sk0Gln4gmtz2MlQnikT1wXgYsOg+OMhuP+IlRH9sENBO0LRn5q+8nbTov4+1p"
crossorigin="anonymous"></script>


 </body>
</html>
```

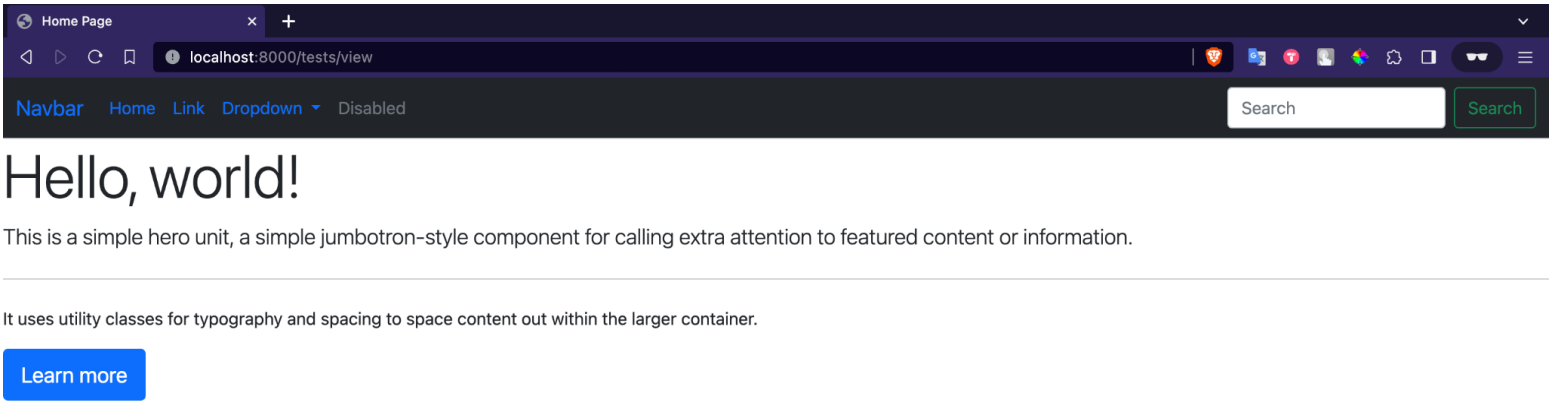
Lets add a new method TestController to return this new home.blade.php view

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

Lets add a route definition inside route/web.php

```
Route::get('/tests/view','\App\Http\Controllers\TestController@view');
```

And lets see the response on our browser at `/tests/view`



Return a Json Array

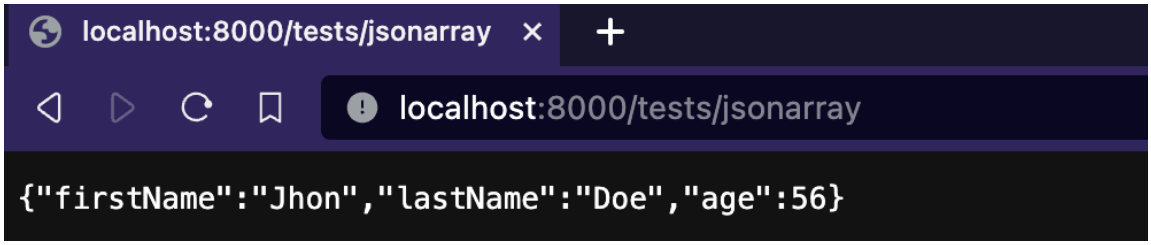
Lets add a new method inside Testcontroller to return a Json Array

```
function jsonarray()
{

    return response()->json([
        "firstName" => "Jhon",
        "lastName" => "Doe",
        "age" => 56
    ]);
}
```

Lets add a route definition inside route/web.php

```
Route::get('/tests/jsonarray','\App\Http\Controllers\TestController@jsonarray');
```

And lets see the response on our browser



Return an Array

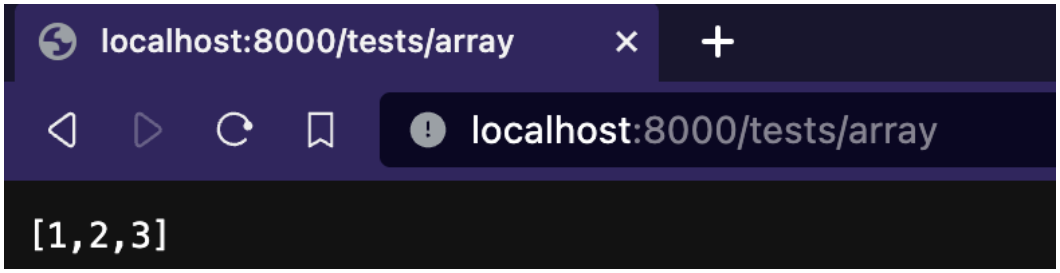
Lets add a new method inside Testcontroller to return an Array

```
function array()
{
    return [1, 2, 3];
}
```

Lets add a route definition inside route/web.php

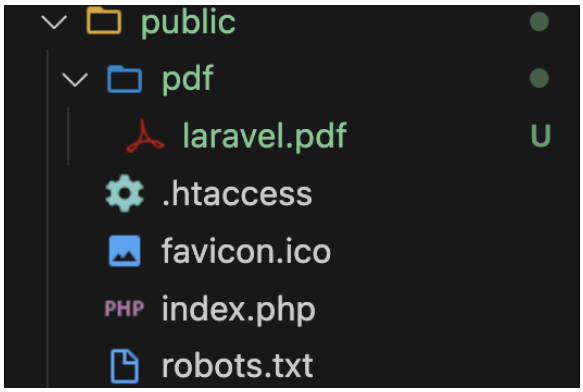
```
Route::get('/tests/array','\App\Http\Controllers\TestController@array');
```

And lets see the response on our browser



Render a pdf on browser

Lets first add a pdf file inside public folder under a pdf folder



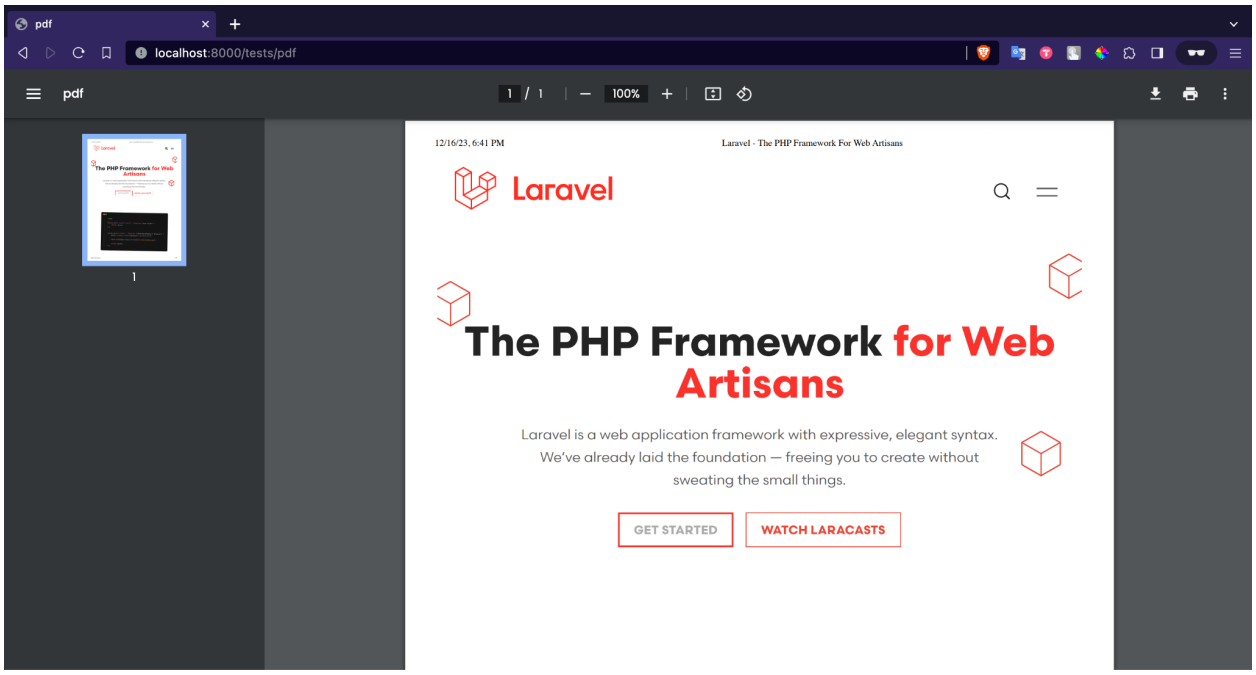
Lets add a new method inside Testcontroller to render a pdf in browser

```
function pdf(){

    return response()->file(
     public_path('pdf/laravel.pdf'),
     ['content-type'=>'application/pdf']);
}
```

Lets add a route definition inside route/web.php

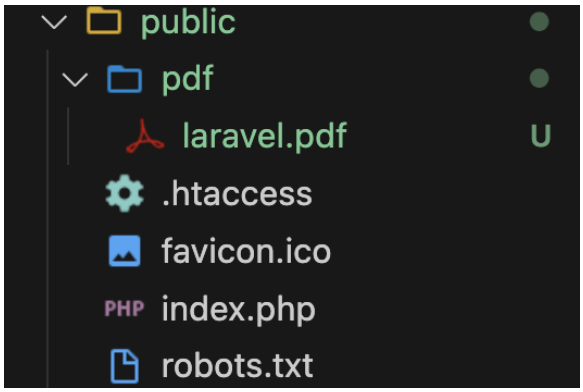
```
Route::get('/tests/pdf','\App\Http\Controllers\TestController@pdf');
```

And lets see the response on our browser at `/tests/pdf`



Return a pdf for Direct Download

Lets first add a pdf file inside public folder under a pdf folder



Lets add a new method inside Testcontroller to return a pdf for direct download

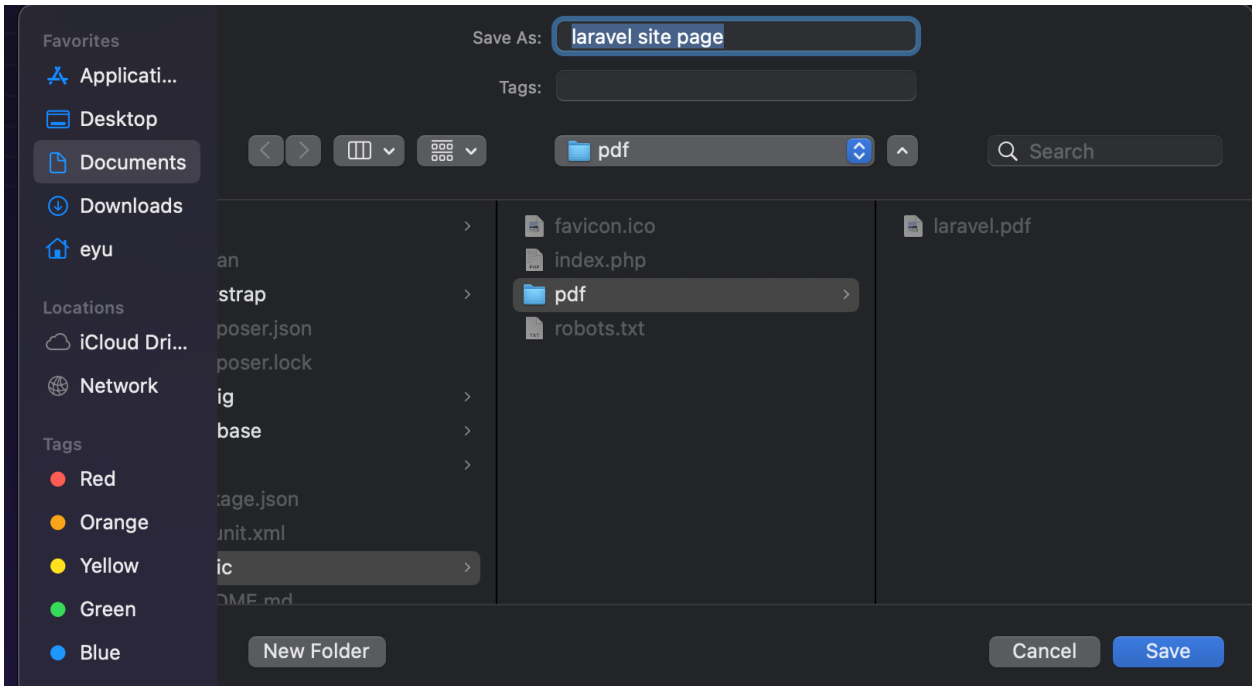
```
function downloadpdf(){

    // download a file with given path with the given name
    return response()->download(public_path('pdf/laravel.pdf'), 'laravel site page');

}
```

Lets add a route definition inside route/web.php

```
Route::get('/tests/downloadpdf','\App\Http\Controllers\TestController@downloadpdf');
```

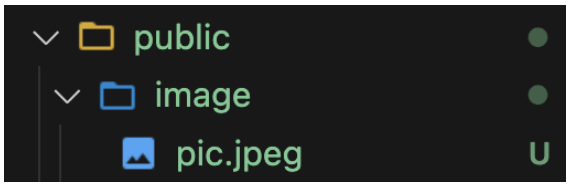
And lets see the response on our browser at `/tests/downloadpdf`

It should directly show you a download and save option with the name shown in the controller action method



Render a Image on browser

Lets first add a image file inside public folder under a image folder



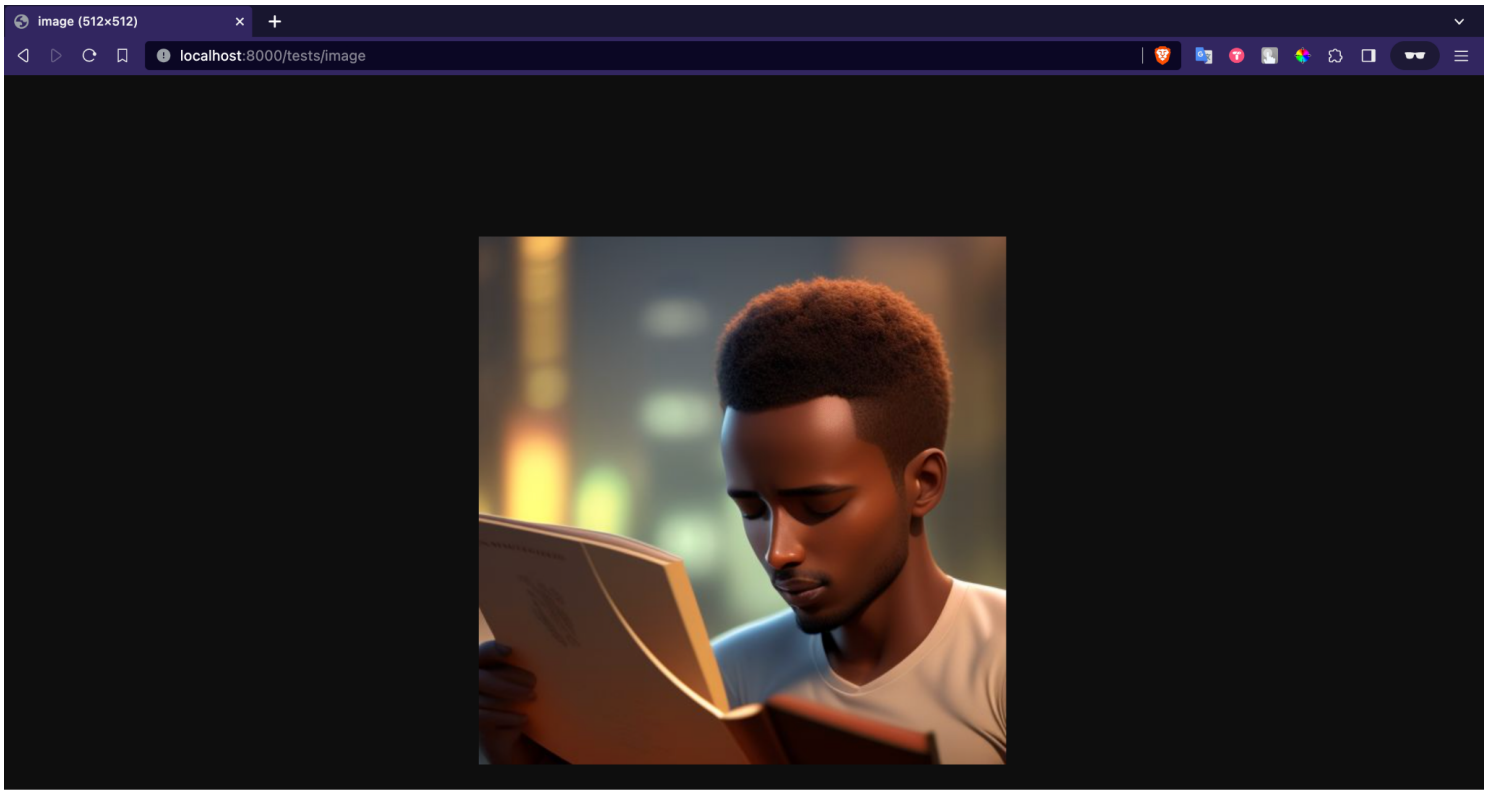
Lets add a new method inside Testcontroller to render an image in browser

```
    function image(){

        return response()->file(
         public_path('image/pic.jpeg'));
    }
```

Lets add a route definition inside route/web.php

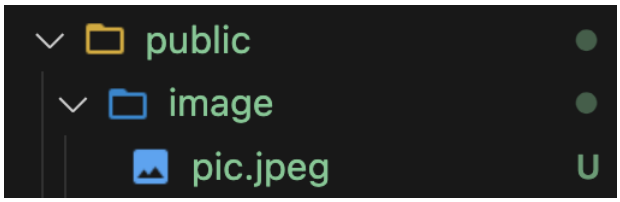
```
Route::get('/tests/image','\App\Http\Controllers\TestController@image');
```

And lets see the response on our browser at `/tests/image`



Return an image for Direct Download

Lets first add an image file inside public folder under a image folder



Lets add a new method inside Testcontroller to return an image for direct download

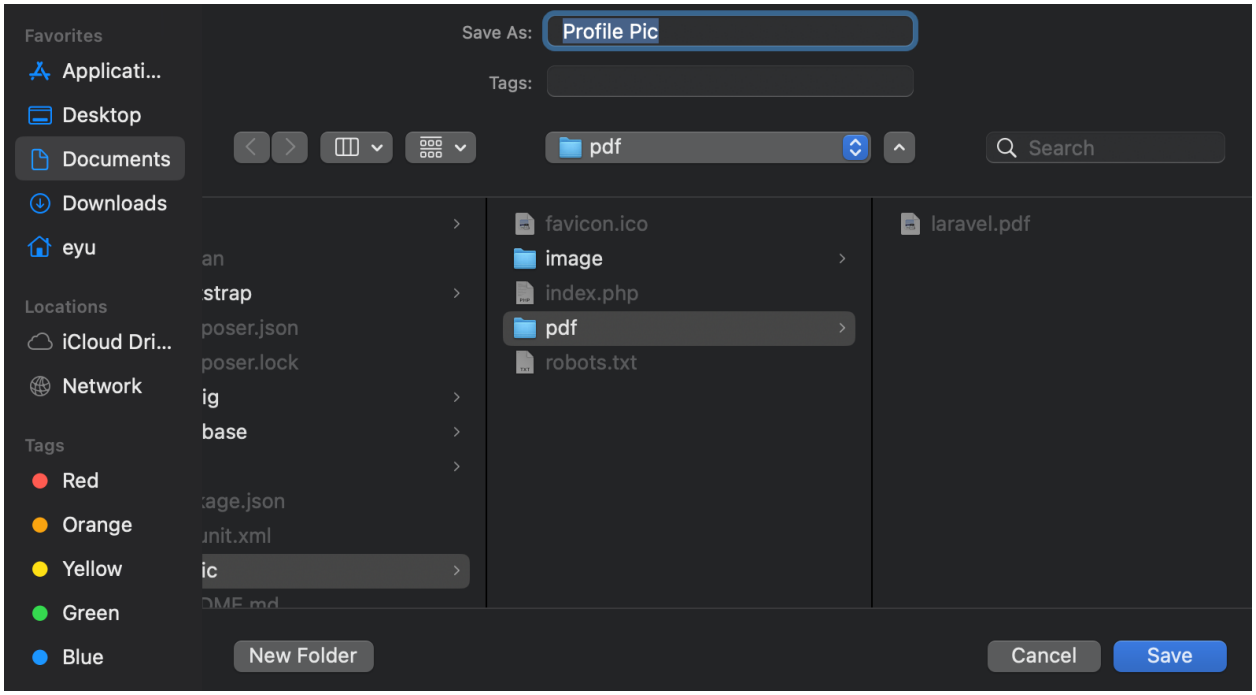
```php
function downloadimage(){

    return response()->download(public_path('image/pic.jpeg'), 'Profile Pic');

}
```

Lets add a route definition inside route/web.php

```php
Route::get('/tests/downloadimage','\App\Http\Controllers\TestController@downloadimage');
```

And lets see the response on our browser at `/tests/downloadimage`

It should directly show you a download and save option with the name shown in the controller action method



Render HTML

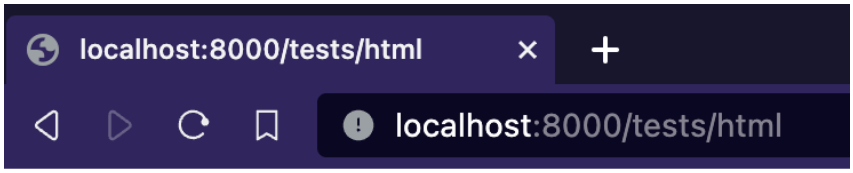
Lets add a new method inside Testcontroller to return an HTML

```
function html() {

    return "
            <table>
                <tr>
                    <th>Month</th>
                    <th>Savings</th>
                </tr>
                <tr>
                    <td>January</td>
                    <td>$100</td>
                </tr>
                <tr>
                    <td>February</td>
                    <td>$250</td>
                </tr>
                <tr>
                    <td>March</td>
                    <td>$200</td>
                </tr>
            </table>
        ";
}
```

Lets add a route definition inside route/web.php

```
Route::get('/tests/html','\App\Http\Controllers\TestController@html');
```

And lets see the response on our browser at `/tests/html`



Month	Savings
January	$100
February	$250
March	$200

Redirect to internal or external urls

```php
function redirects(){

        // redirect to tests/view page
        return redirect('tests/view');

        // redirect with named route
        return redirect()->route('posts');

        // redirect with named route by passing variables
        return redirect()->route('posts', ['id' => 1]);


        // redirect to specific controller method
        return redirect()->action('TestController@view');


        // redirect to external url
        return redirect()->away('https://www.google.com');

}
```

Our Online Shopping Project