UECS3294 ADVANCED WEB APPLICATION DEVELOPMENT

Practical 2 : Route, Controller and View

In this lab, you will use Laravel Framework to learn expound the concept of routing, comtroller and view.

1. Route.

Routing in Laravel simply means mapping the Laravel page with a specific URL. In Figure 1, we can see a few examples of routing.

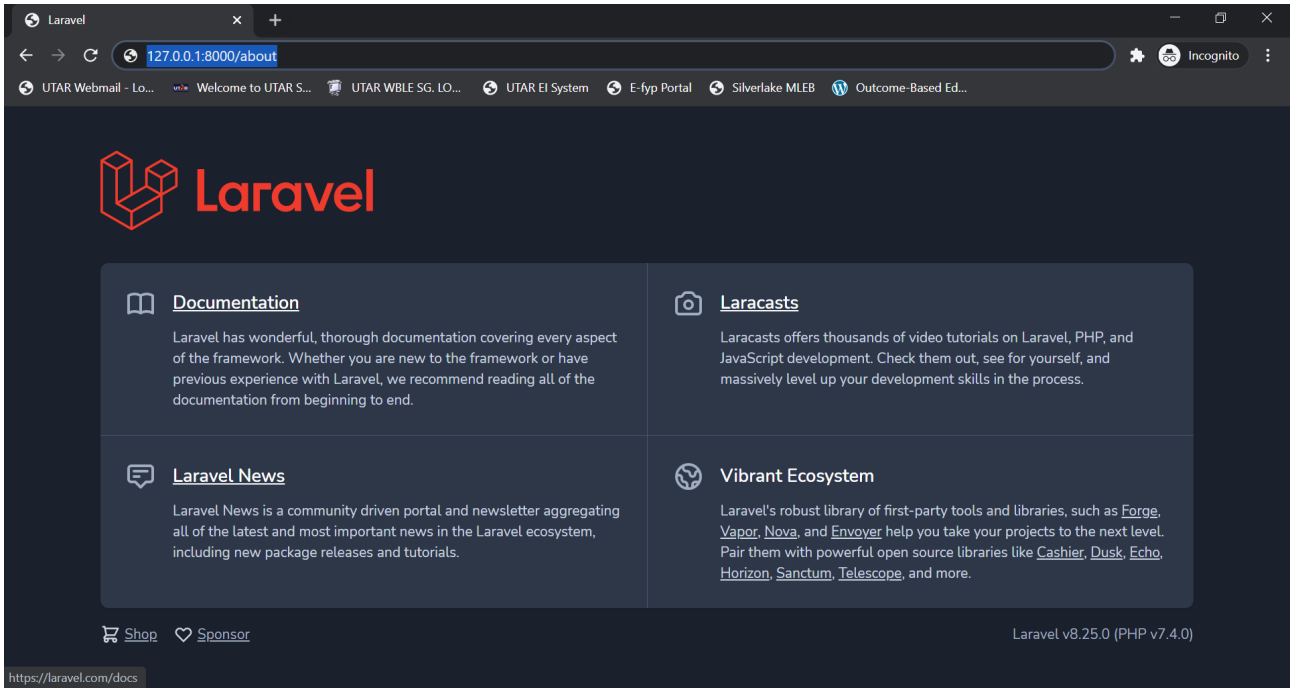

Figure 1: Specific URL for routing to different pages.

Based on previous lab, we learn that within Laravel web application project files, the first page is found in resources >> views >> welcome.blade.php
As for the route, is found in routes >> web.php
Within web.php, the scripts

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

implicate that this route is built to fetch the page from the url of “root directory” using “get” method. If we would like to change the url to “/home”, that means the access to the page on server will have to change as well. Example is as shown in Figure 2.

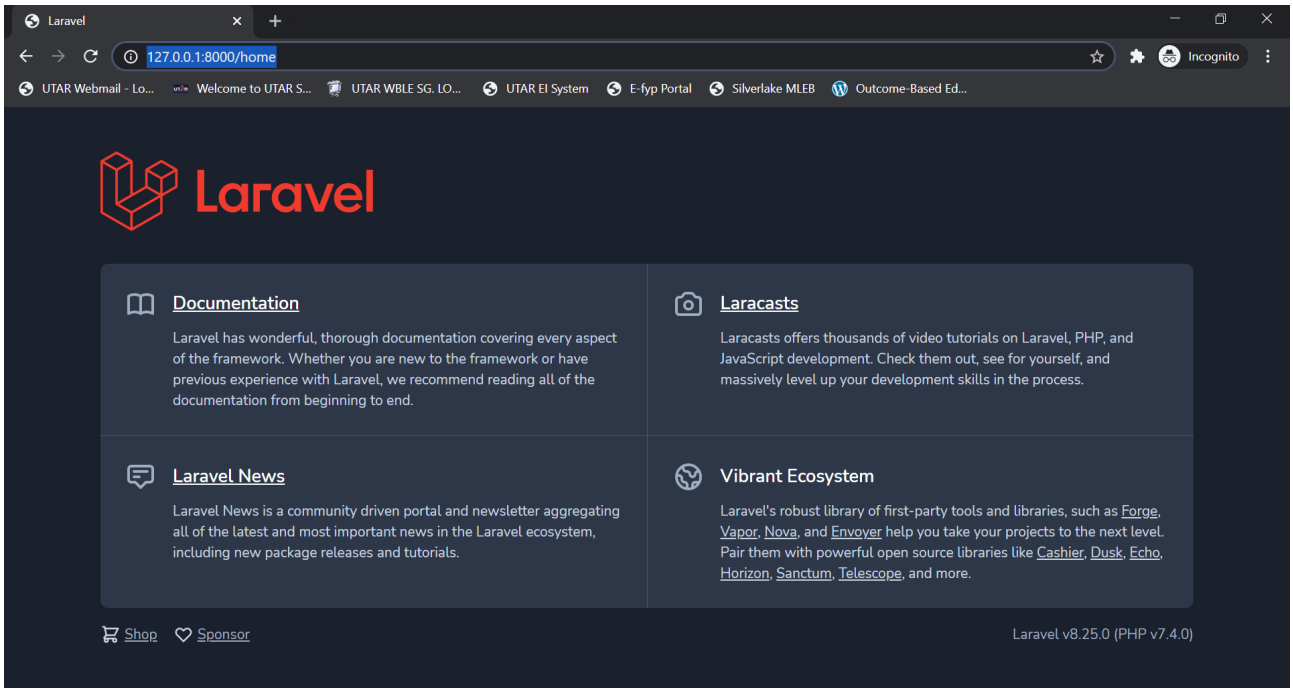

Figure 2: Change of landing page URL.

Exercise:

Create two more pages for routing: “About Us” and “Contact Us” pages. Then, create two routes to route to the pages created.

Webpages have parameters / data such as username within a url link which could be passed to a page using route. To do so, parameters of within the link need to be passed to the returning view as shown in Figure 3.

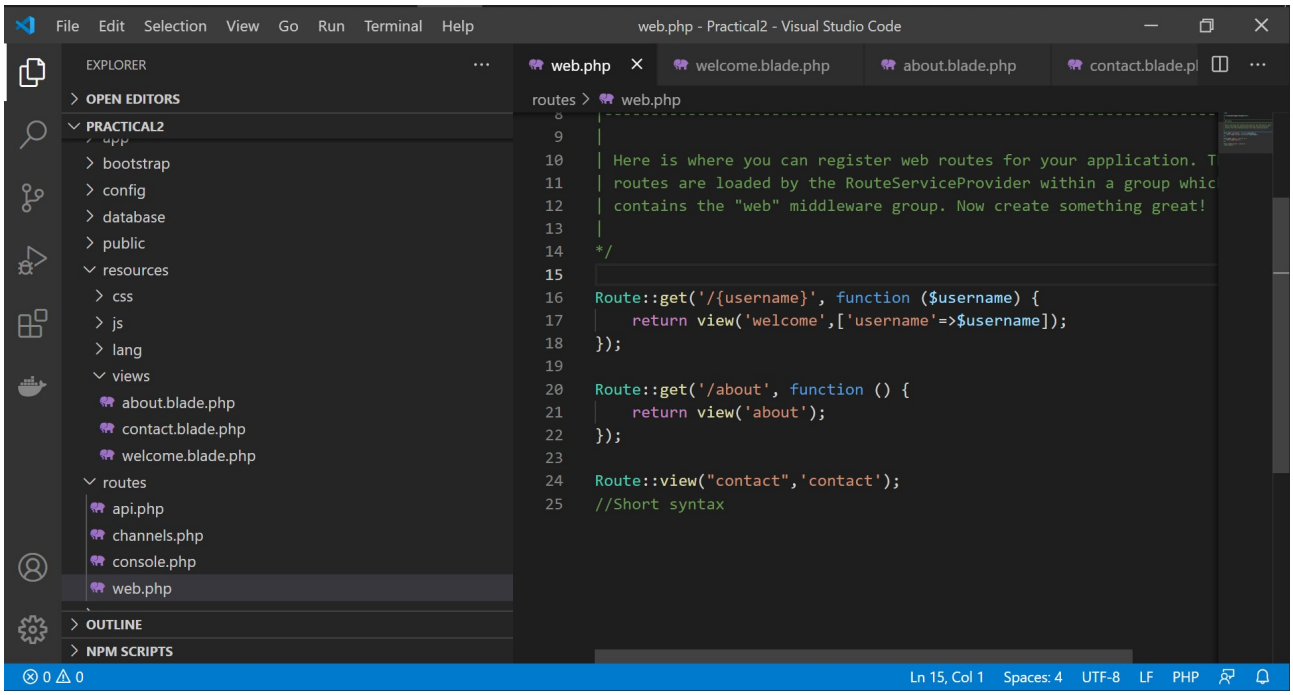

Figure 3: Pass data from URL into routed page.

Then, make sure to have the routed page to echo out / ouput the passed parameter / data as shown in Figure 4.

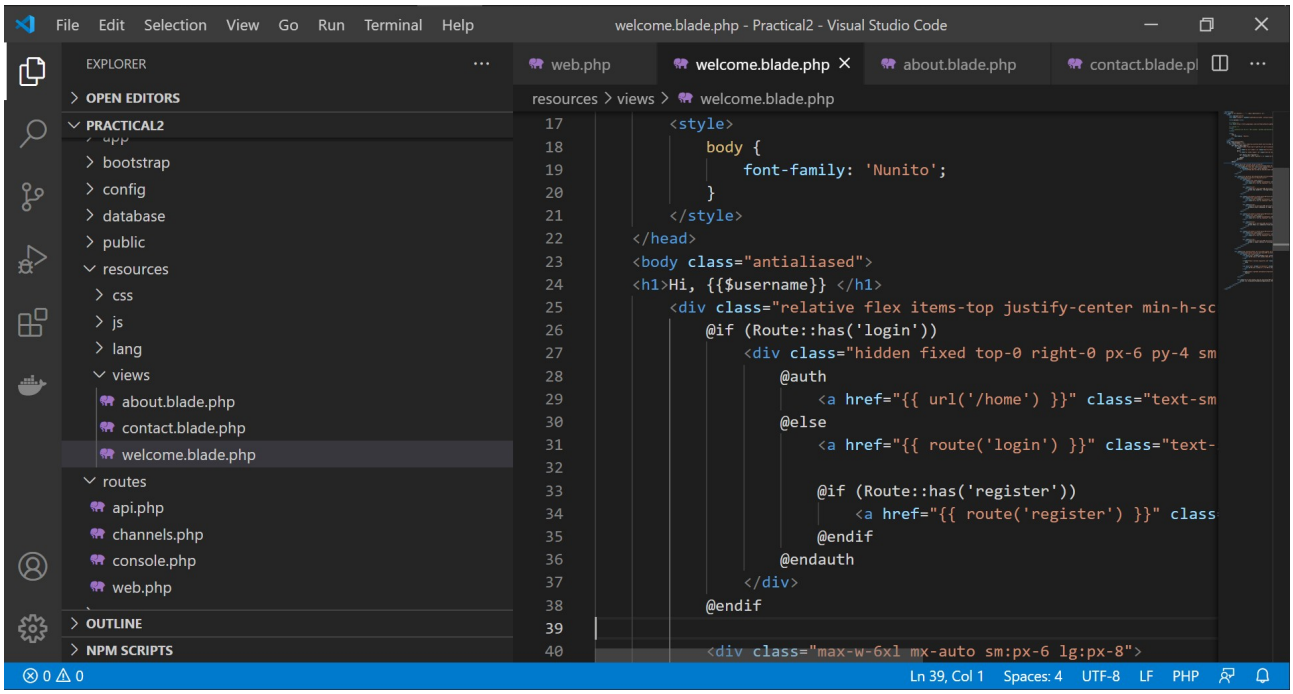

Figure 4: Page modification to output data passed from route.

Save all the modifications and try to let your route pass data to your page as shown in Figure 5.

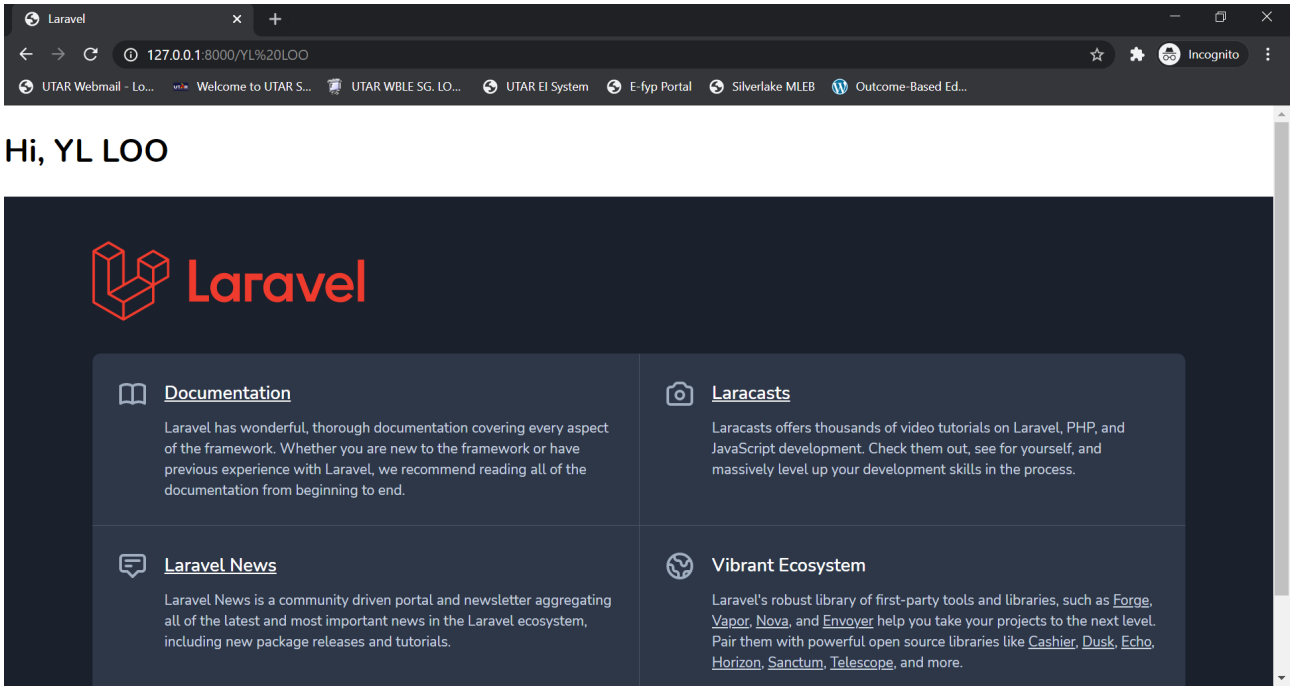

Figure 5: Pass data with route.

Anchor tags within the welcome landing page can be easily created in routes.

Exercise:
Prior to modifying the routes to do so, create anchor tags within your “About Us” and “Contact Us” pages so that each of them will have anchor tags to one another and to the original Laravel landing page.

Web developer can do automatic page redirects when another page is under maintenance using the "redirect" command in route.
In the following example, we look into redirecting the user to About Us page, whenever the Welcome page is accessed.

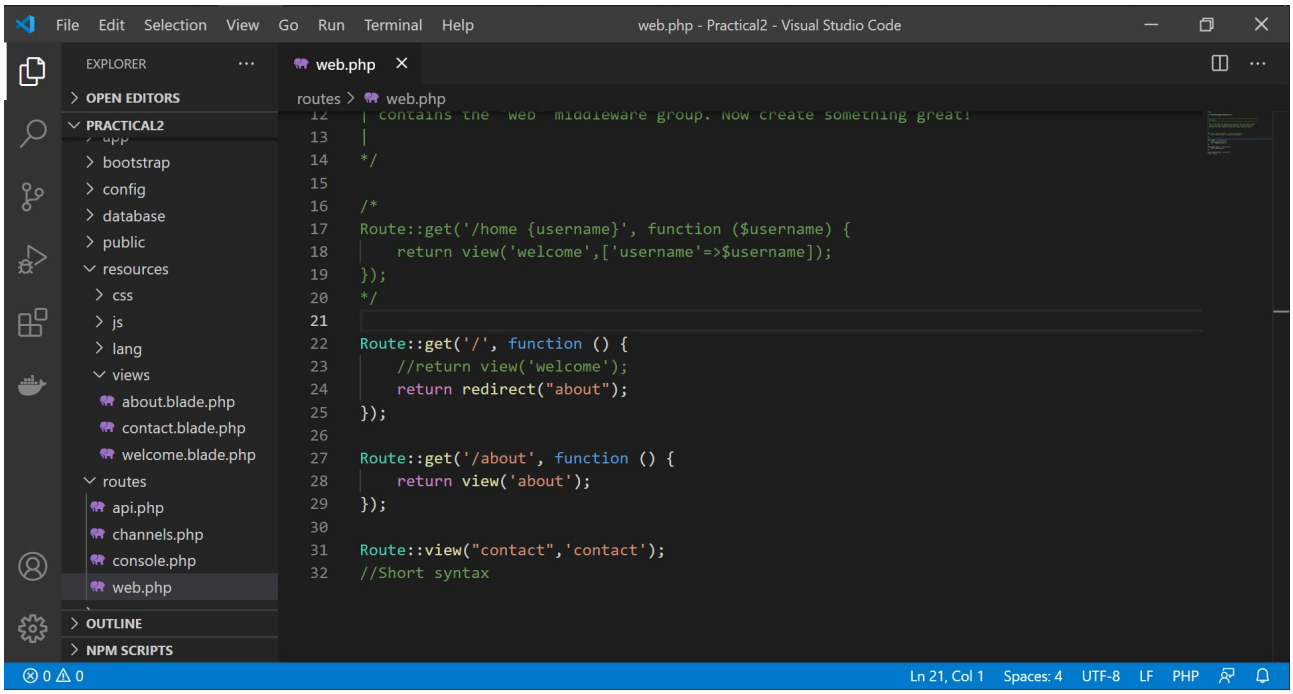

Figure 6: Using redirect in Route.

That's all for Route in Laravel. The following practical session is exploration of Controller.

2. Controller.

Controller is the central unit of any MVC architecture. Controller basically fetch data from Model and send to the View. All logical part of Web Programming is in the Controller and all routes are directly linked to the Controller.

How to create a controller in Laravel web application? There are two ways for doing so.

(a) Through the Artisan CLI "php artisan make:controller". Figure 7 shows the example of creating "Users" controller.

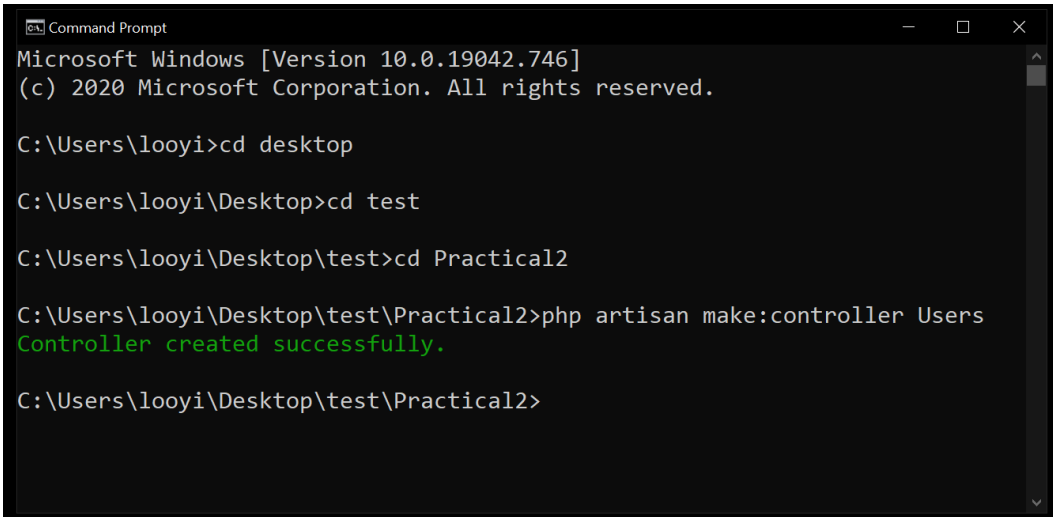

Figure 7: Using Artisan CLI to create Users controller.

(b) Through manual “New File” creation within web application project folder.
Within the newly created Users controller, create a simple echoing function as illustrated in Figure 8.

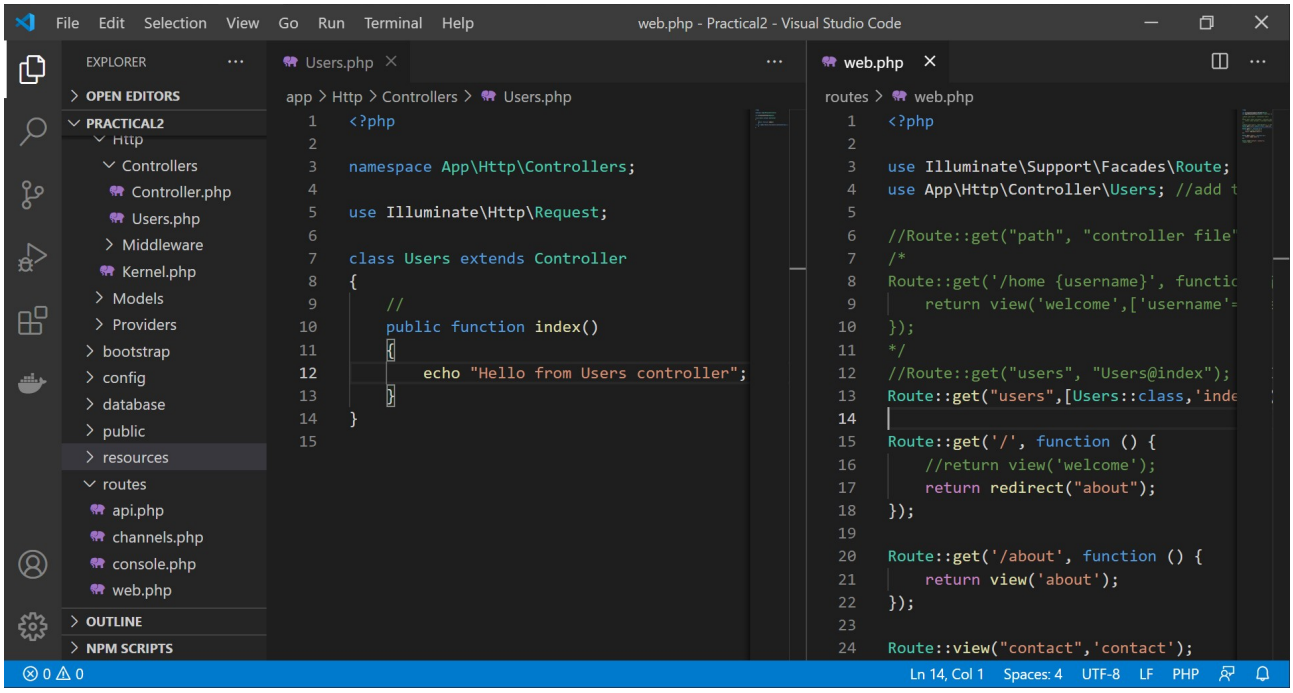

Figure 8: An echoing function in Users Controller.

In order to instruct the route to call the newly created Users Controller, a route need to be created.

***Take note that there is a slight change of syntax for route creation for controllers between older versions of Laravel compared to Laravel 8.*
In older versions of Laravel, a route to controller can be created as such:
`Route::get("controller_path", "controller_name@function_name");`
Specific to this example will be:
`Route::get("users", "Users@index");`
In Laravel 8, the controller need to be imported first, then only be referenced as an array returning to the route that calls it as shown in Figure 9.

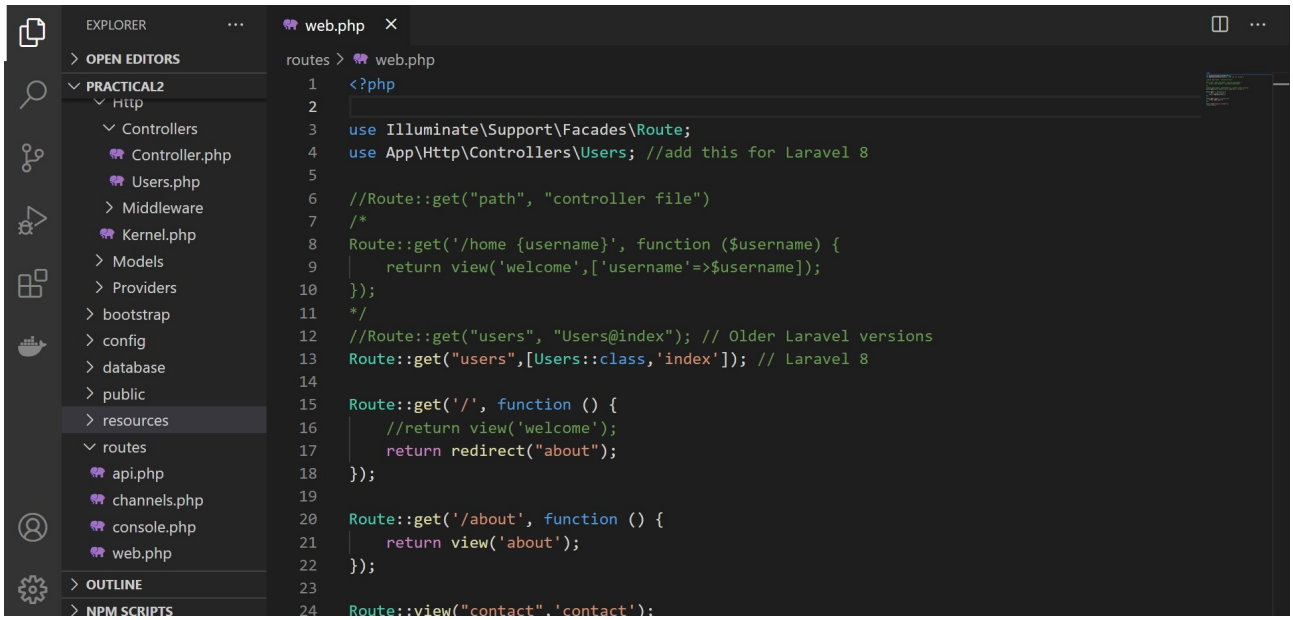

Figure 9: Routing to Controller in Laravel 8.

Similar to route, passing a parameter / data from URL to the controller needs to have the route and controller file modified with an additional parameter / data as shown in Figure 10.

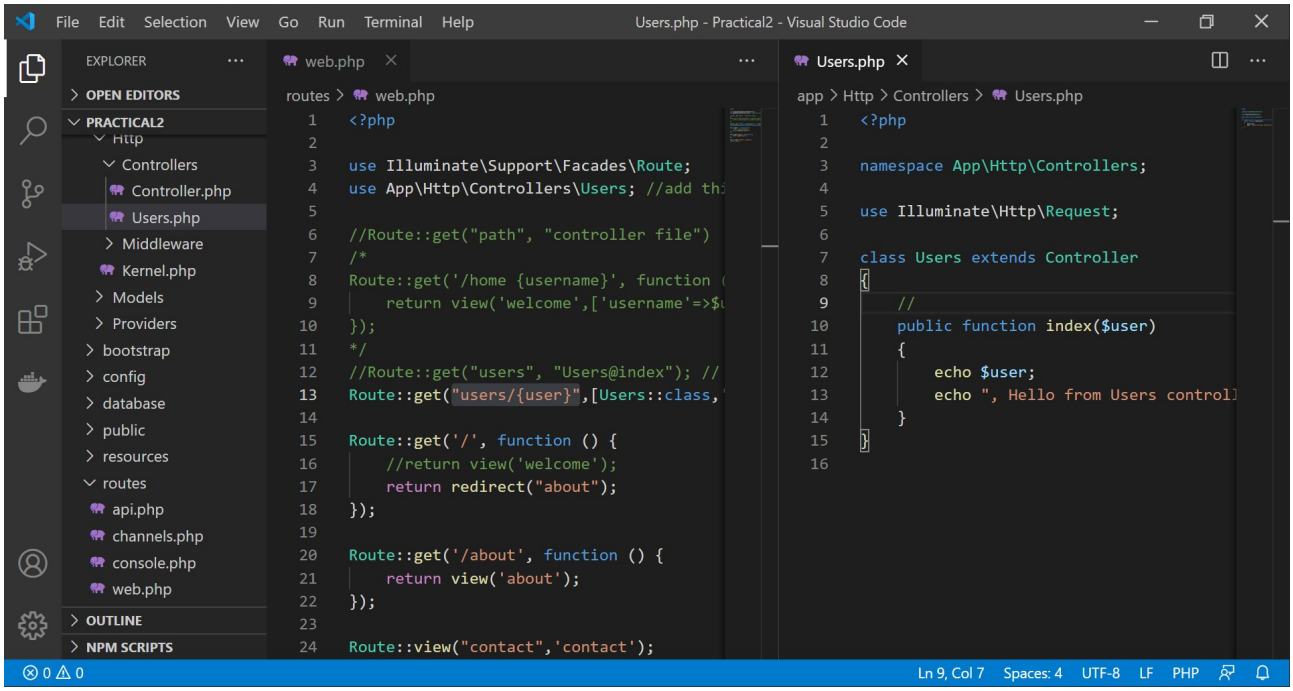

Figure 10: Additional parameters for passing data from URL.

Controller could also be used as an API. Figure 11 shows an example of writing API in a controller.

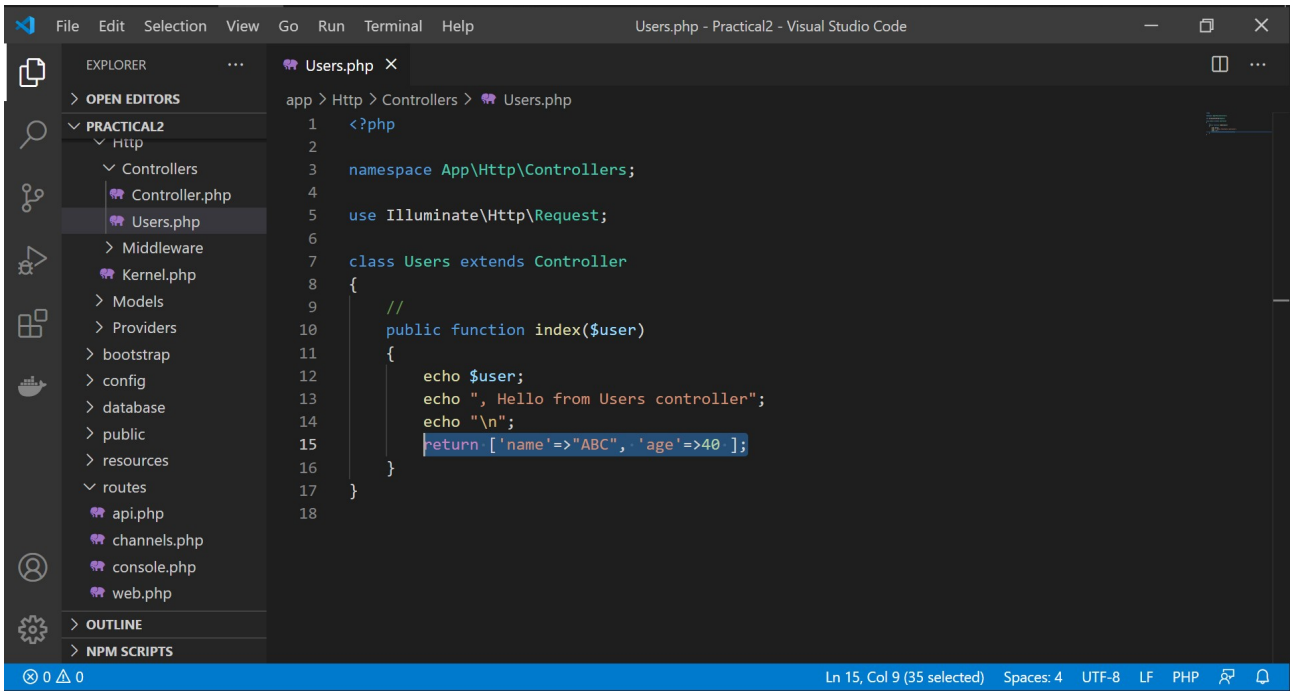

Figure 11: Using Controller as an API.

That's all for Controller in Laravel Frmaework. The following practical session, we expound View.

3. View.

The "view" entity or component of MVC architecture had been explored previously in the earlier practical session. Thus, let's have an exercise to refresh our memories of previously learnt "route" and "view" concepts.

Exercise.

Create a "user" view and a route to users. Specify the route so that parameter of username in the URL could be passed to "user" view.

We have known about creating view, calling view from route, passing data from URL to view using route. Now, let's explore the calling of view from controller. Using previously created "users" controller, create a loadView() function to call "user" view from "users" controller when the controller is routed to as shown in Figure 12.

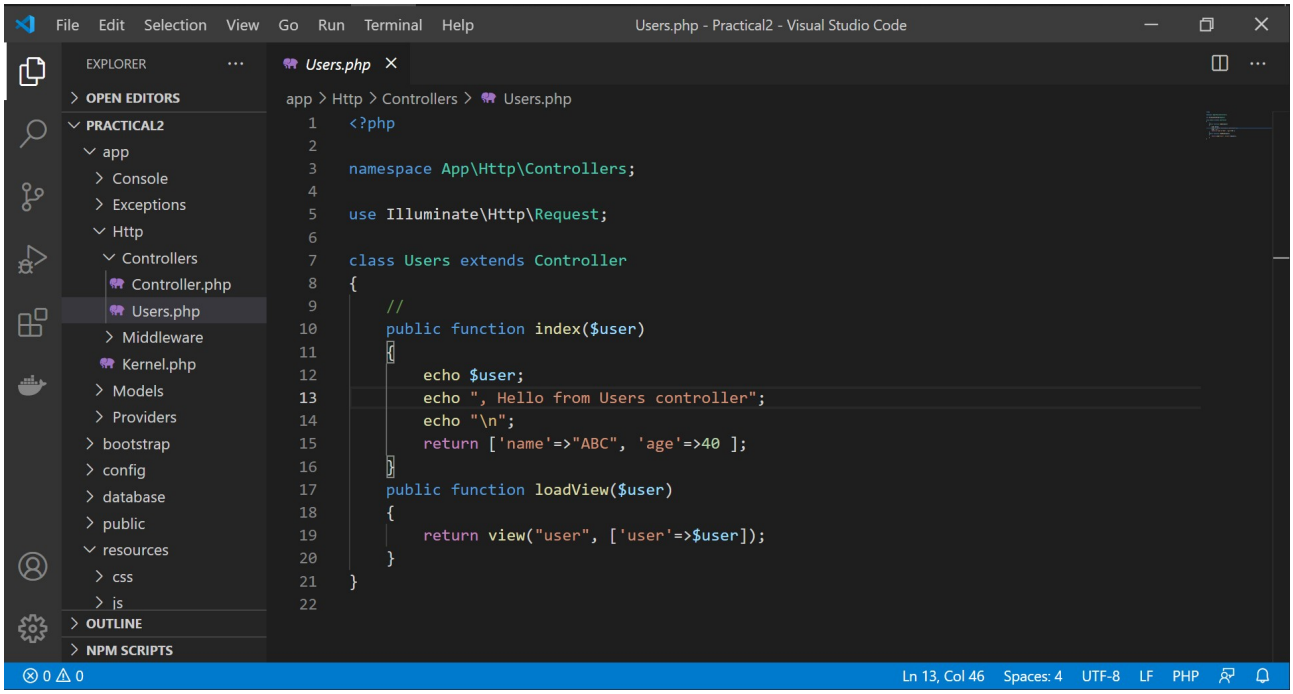

Figure 12: Users controller to call for user view.

Exercise.

Create a "users" controller route which allow a username parameter in the URL to be passed to "user" view.

Using Laravel Component to format View.

In Laravel, a component can be reused in view files in order to avoid copy and pasting similar scripts in different view files. This modularization also promote code reuse. In view, parts or divisions which could be reused could be “header” and “footer”. Let’s create a “header” component using the Artisan CLI “php artisan make:component Header” to create our first component as shown in Figure 13.

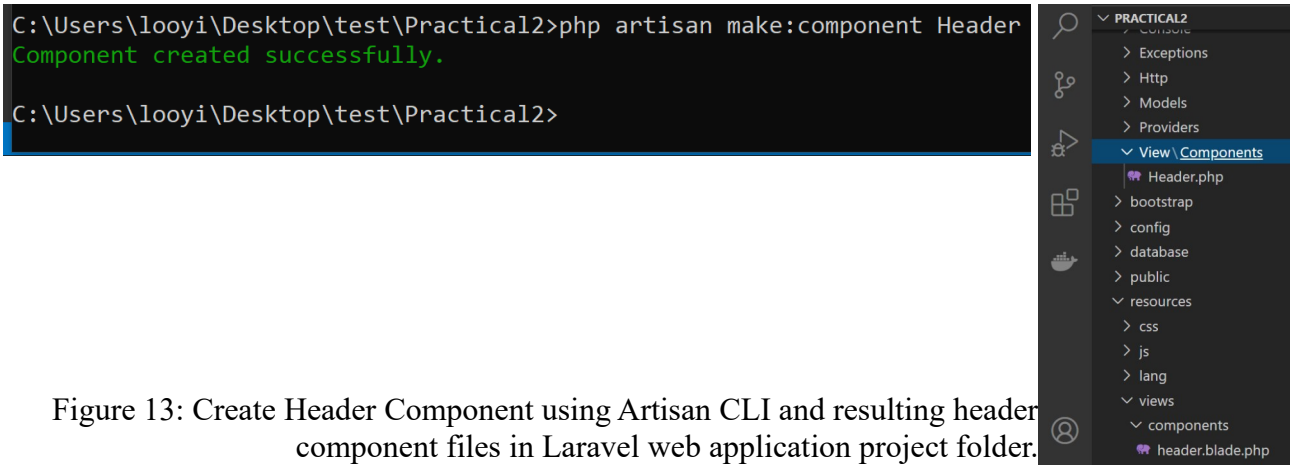

Figure 13: Create Header Component using Artisan CLI and resulting header component files in Laravel web application project folder.

Once a Laravel Component is created, a php file is generated in “app” folder and a blade template HTML is generated in resources >> views.
Stylize the header.blade.php file in order to create a visible output for any view file that will use the component as shown in Figure 14.

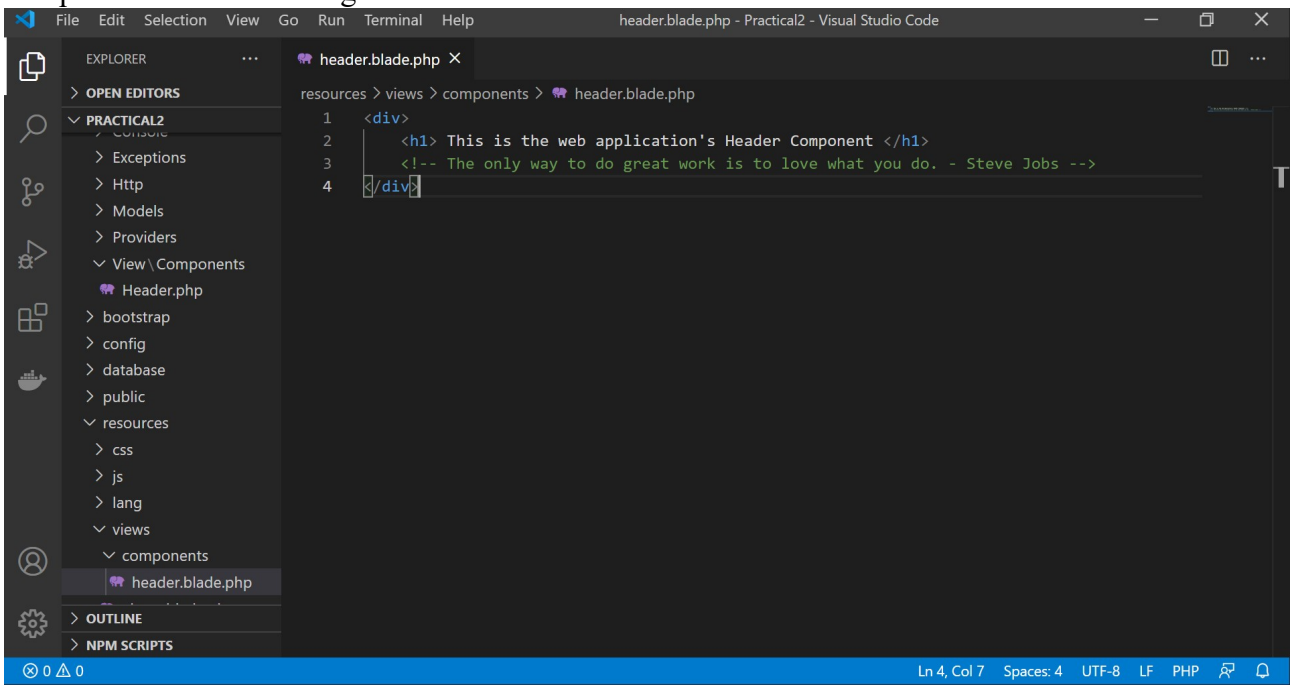

Figure 14: Stylize Header component.

After styling the Header component, let About Us and Contact Us view use header component with the <x-header > script as shown in Figure 15.

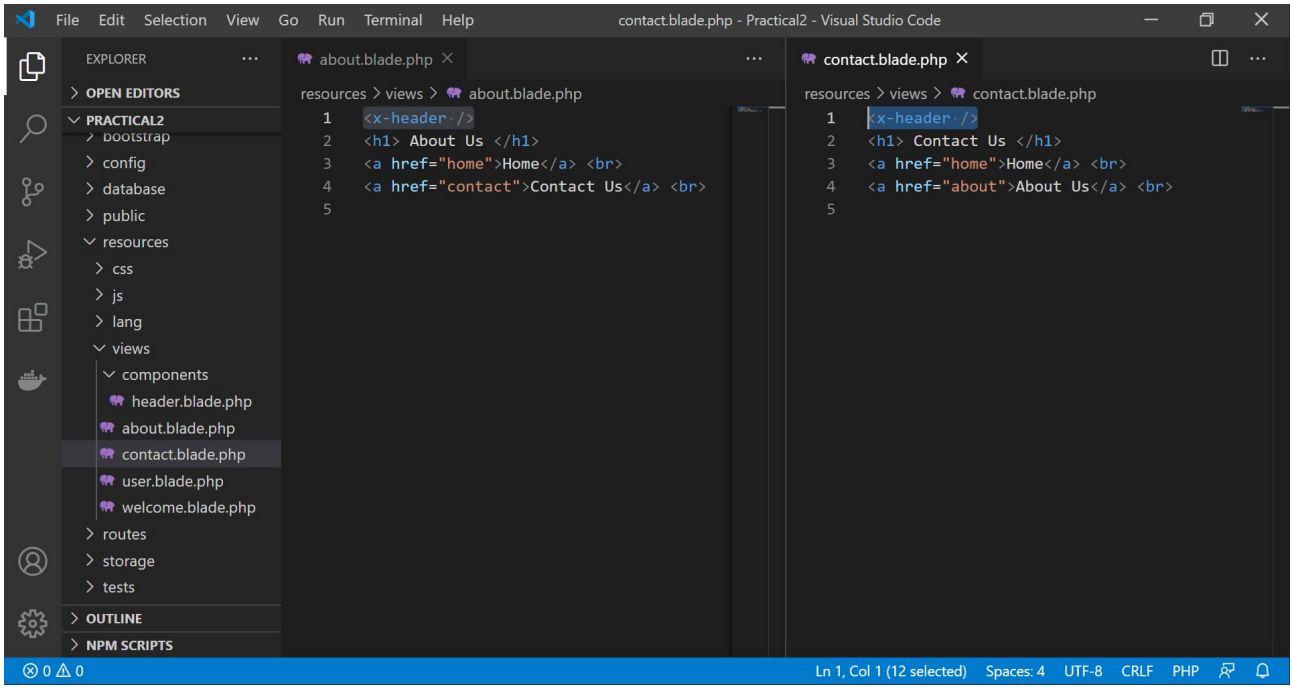

Figure 15: Using Header component in View.

If we would like to format the Header Component to be sepcifically for About Us and Contact Us; we need to create functions in the component's PHP file. In order to create a page-specific header, as the first step, a parameter / data need to be passed to the header component as shown in Figure 16.

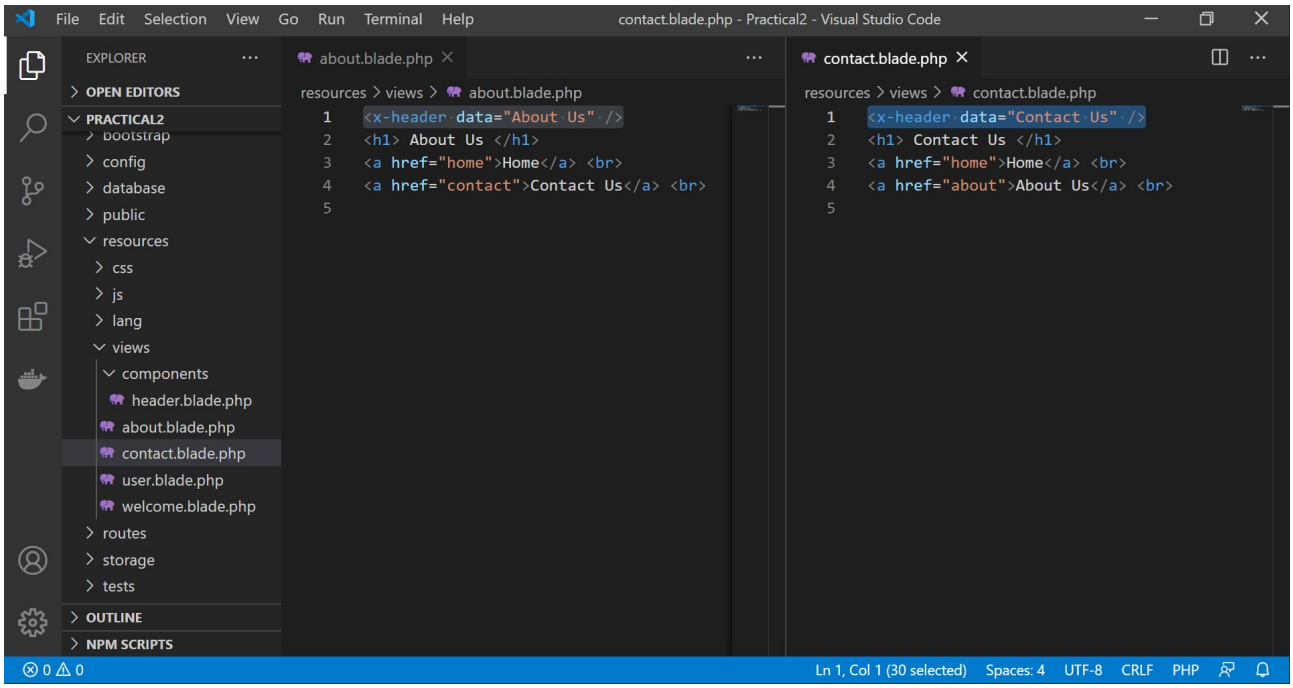

Figure 16: Passing data from View to Component.

Then, we need to manipulate the component's PHP and HTML file as shown in Figure 17.

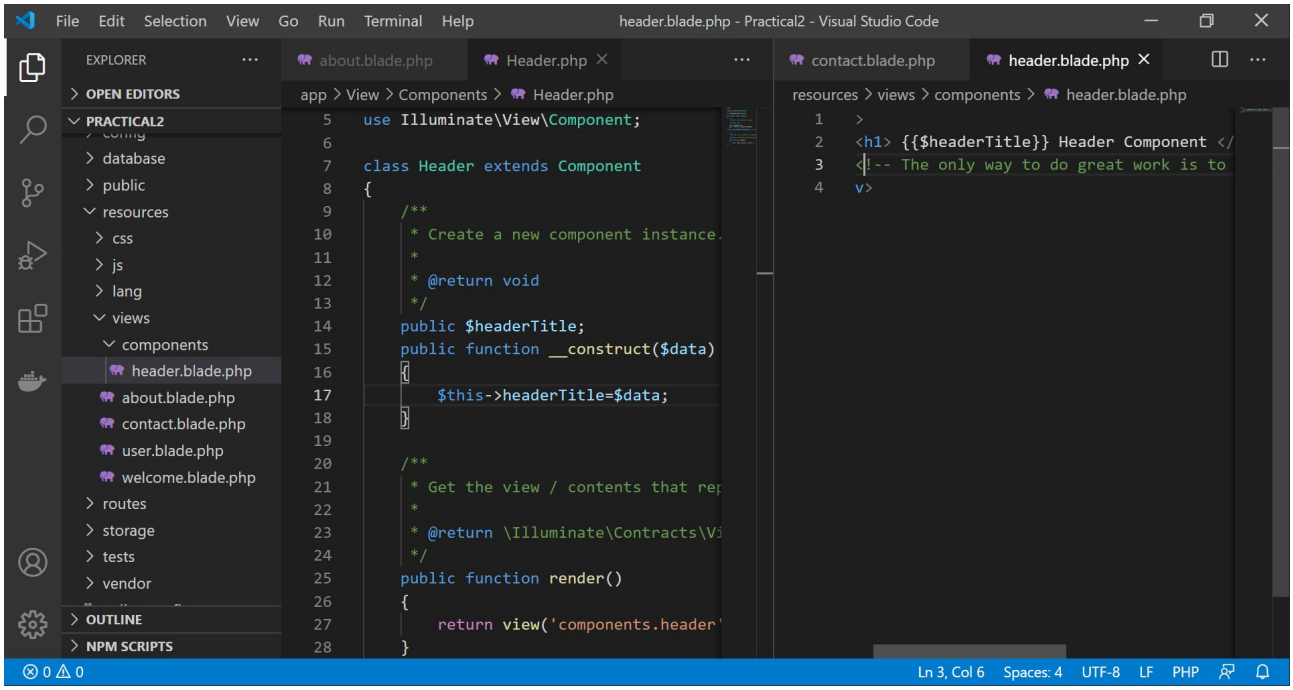

Figure 17: Processing data passed from View to Component.

Using Blade Template to format View.

Blade templating is a feature offered in Laravel framework for specific ways of having PHP scripts in view files. Previously, we've known that blade templates recognize double curly braces "{ { } }" as equivalent to "<?php ?>". In this session, we will look into some of the common functions scripting using blade templating. The first one we will look into, is conditional functions.

Looking into user view and users controller, create a condition to allow only three users to be known user for the web application, while others to be unknown. An example is shown in Figure 18.

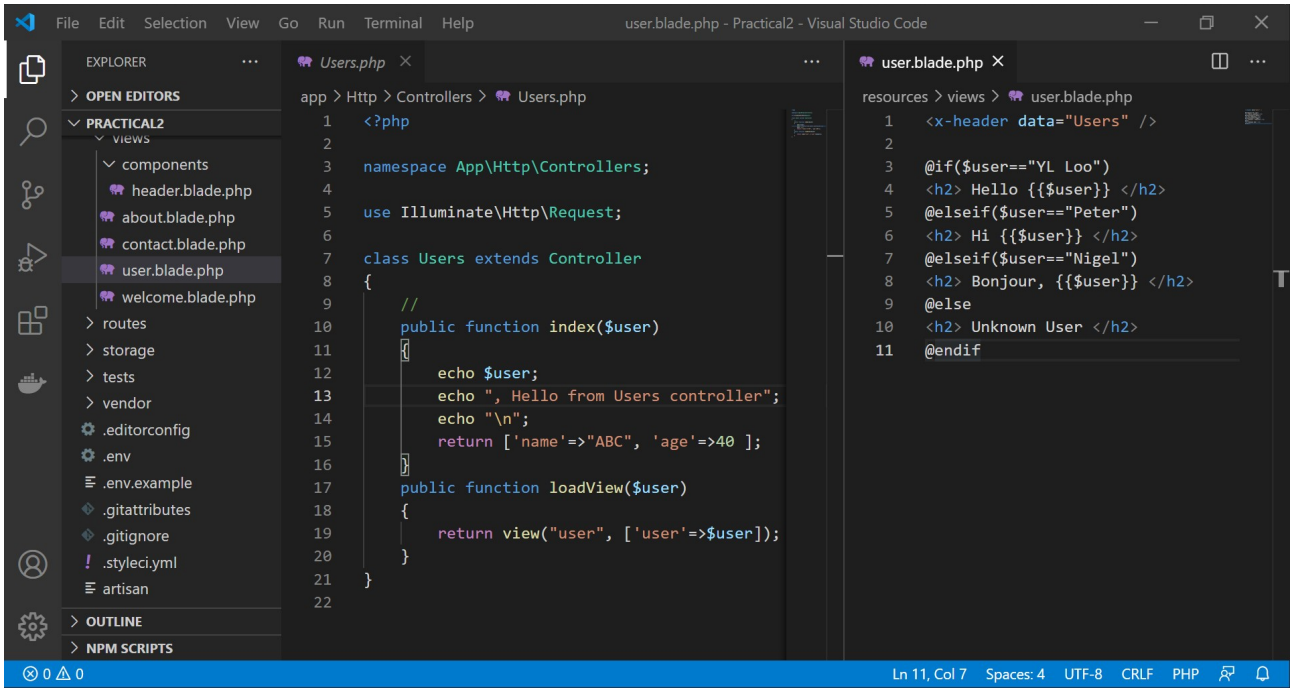

Figure 18: Conditional Function in Blade Template.

Then, let's create a Loop Function which will pass an array of users from the controller to view as shown in Figure 19.

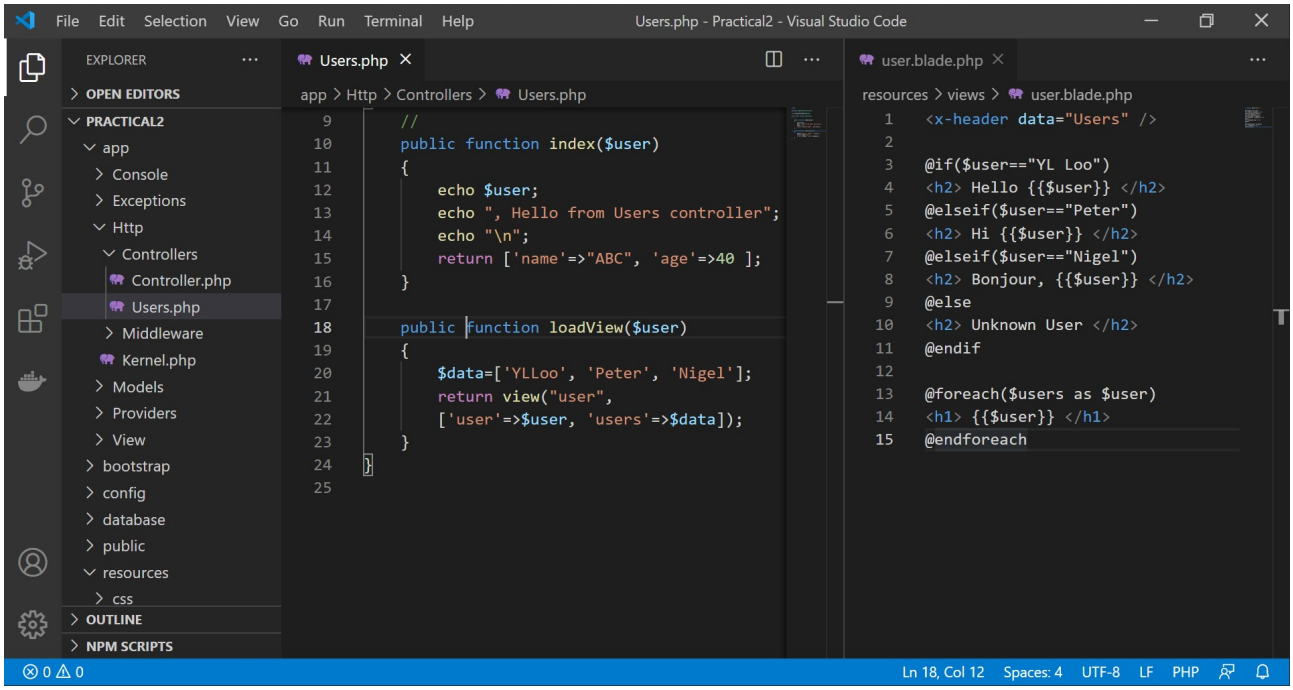

Figure 19: Loop Function in in Blade Template.

Using Blade Template to Create View in View.

Blade templating in Laravel allows putting a view within a view if a web application do have inner view designs. In order to explore that, create an userInner view and simply call userInner view in user view using @include script as shown in Figure 20.

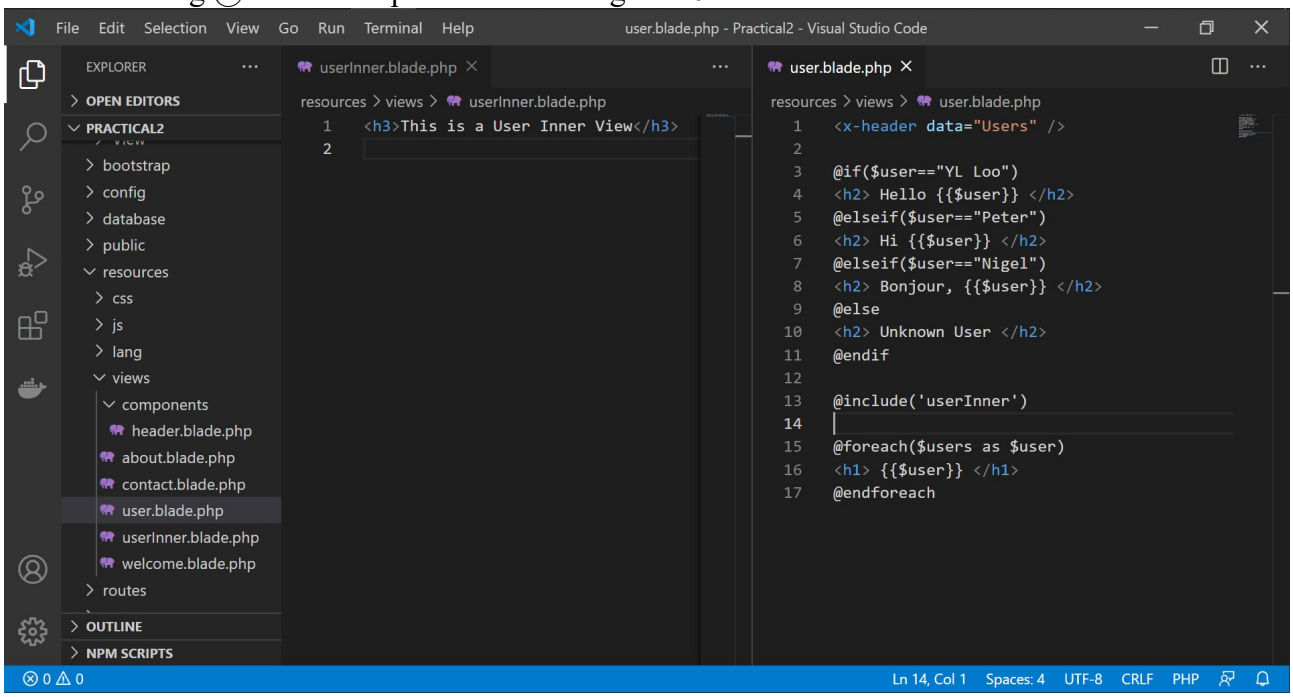

Figure 20: Using Blade Template for View in View.

Using Blade Template with Javascript and csrf Token.

Javascripts can be included in a blade template and a csrf can be transmitted within Blade Template. A sample of the usages are shown in Figure 21.

***Press F12 in browser to interact with console and csrf token.*

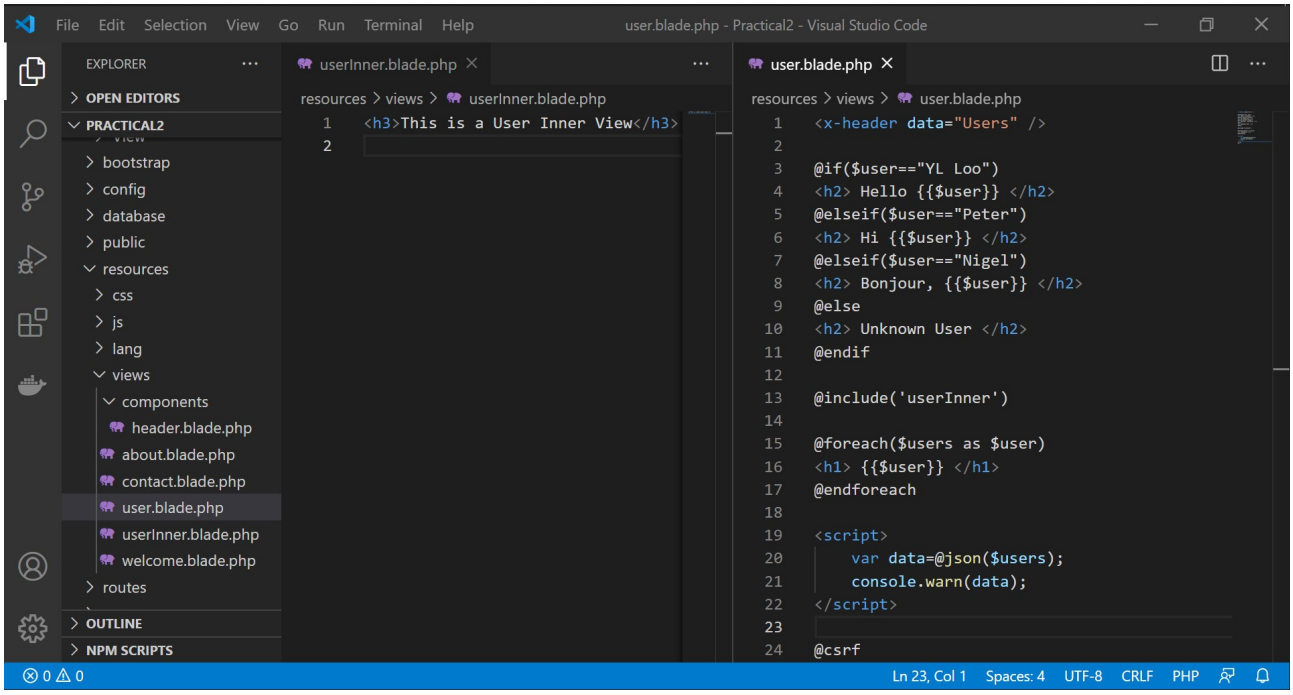

Figure 21: Using Blade Template for JS and csrf call.