

Practical 2: Route, Controller and View

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

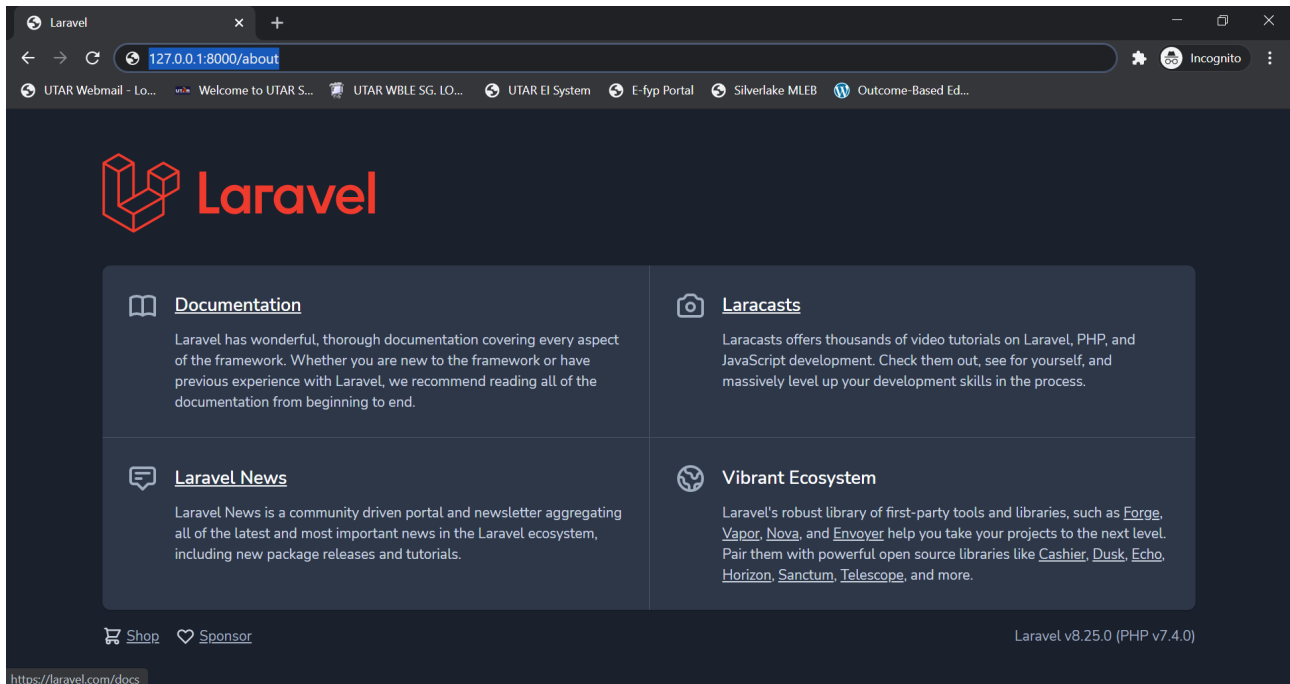


Figure 1: Specific URL for routing to different pages.

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');

});
```

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

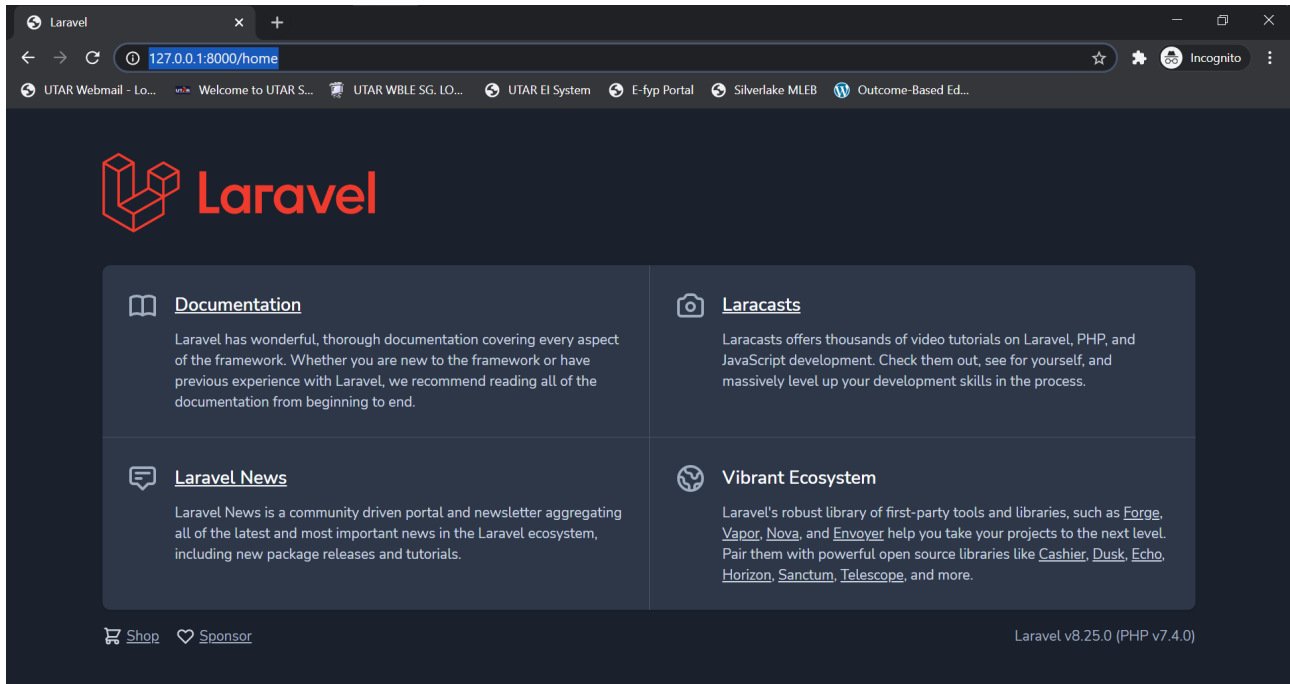


Figure 2: Change of landing page URL.

Exercise 1: Create 2 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.

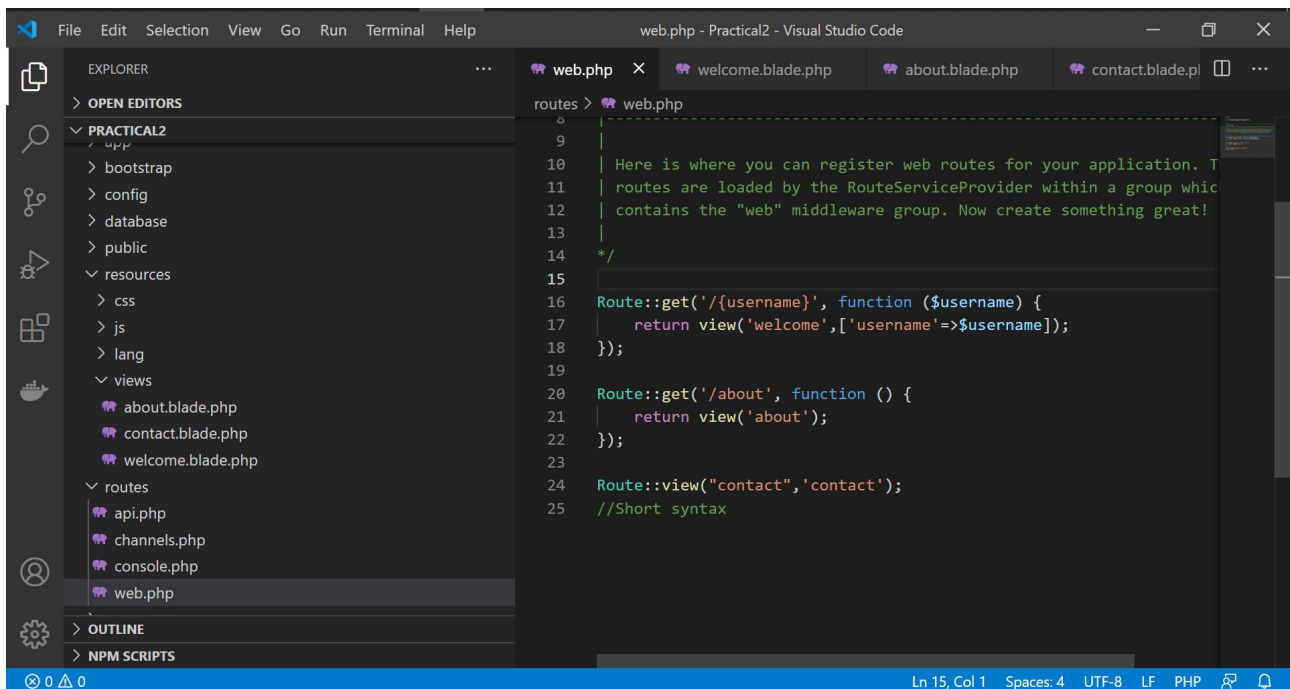
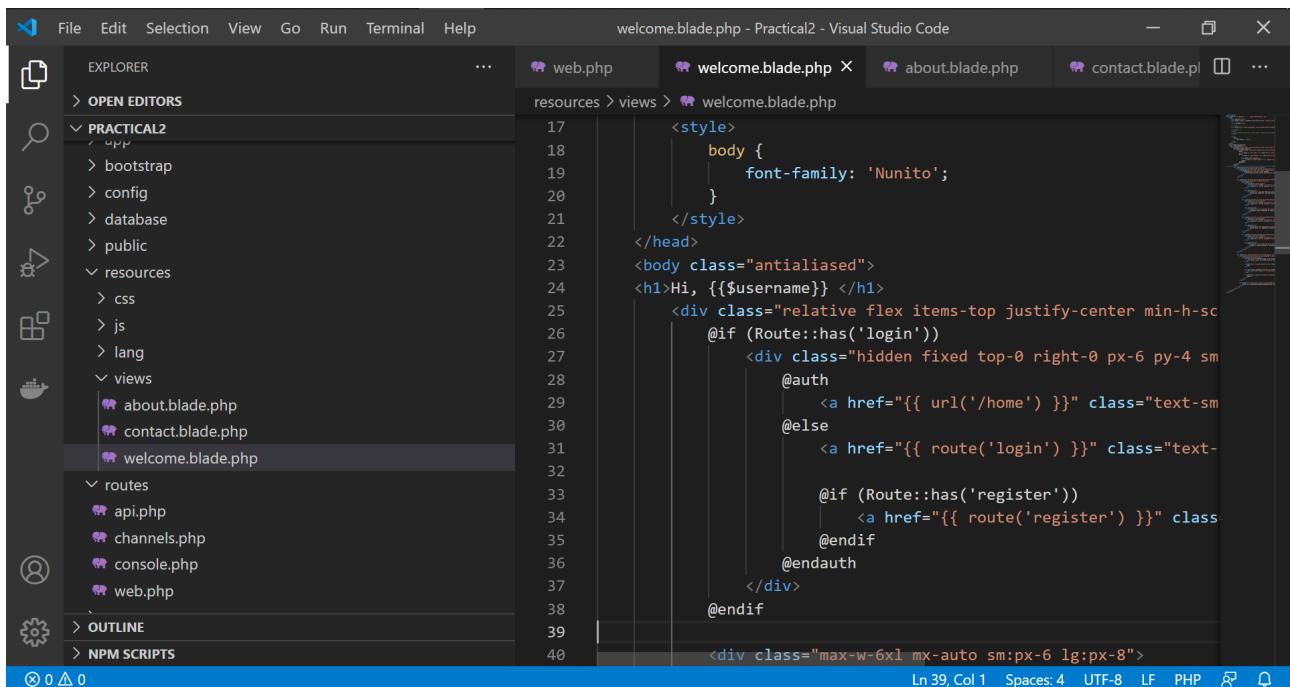


Figure 3: Pass data from URL into routed page.

UECS3294 ADVANCED WEB APPLICATION DEVELOPMENT

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



The screenshot shows the Visual Studio Code editor with the file `welcome.blade.php` open. The file is located in the `resources > views` directory. The code is as follows:

```
17 <style>
18     body {
19         font-family: 'Nunito';
20     }
21 </style>
22 </head>
23 <body class="antialiased">
24 <h1>Hi, {{ $username }} </h1>
25 <div class="relative flex items-top justify-center min-h-screen">
26     @if (Route::has('login'))
27         <div class="hidden fixed top-0 right-0 px-6 py-4 sm:px-4 sm:py-5">
28             @auth
29                 <a href="{{ url('/home') }}" class="text-sm text-gray-600 hover:text-gray-900">Home</a>
30             @else
31                 <a href="{{ route('login') }}" class="text-sm text-gray-600 hover:text-gray-900">Log in</a>
32             @endif
33             @if (Route::has('register'))
34                 <a href="{{ route('register') }}" class="text-sm text-gray-600 hover:text-gray-900">Register</a>
35             @endif
36         @endauth
37     @endif
38 </div>
39
40 <div class="max-w-6xl mx-auto sm:px-6 lg:px-8">
```

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.

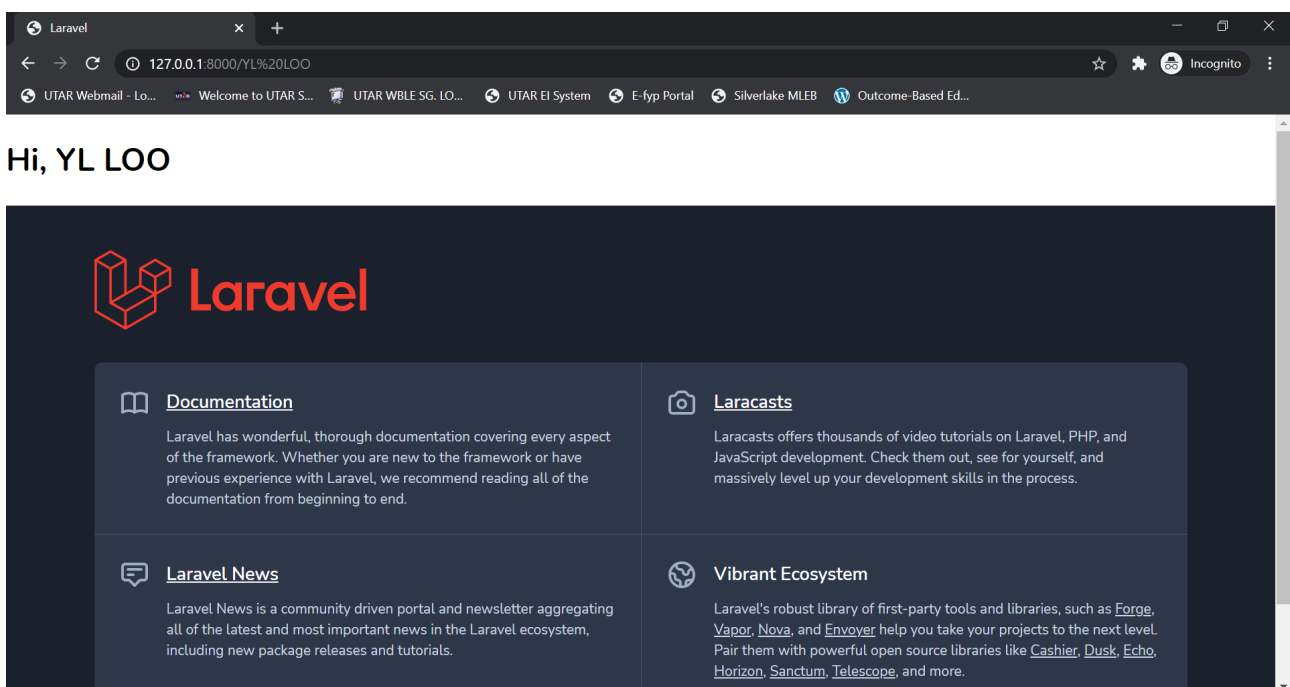


Figure 5: Pass data with route.

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

UECS3294 ADVANCED WEB APPLICATION DEVELOPMENT

Exercise 2: 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 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.

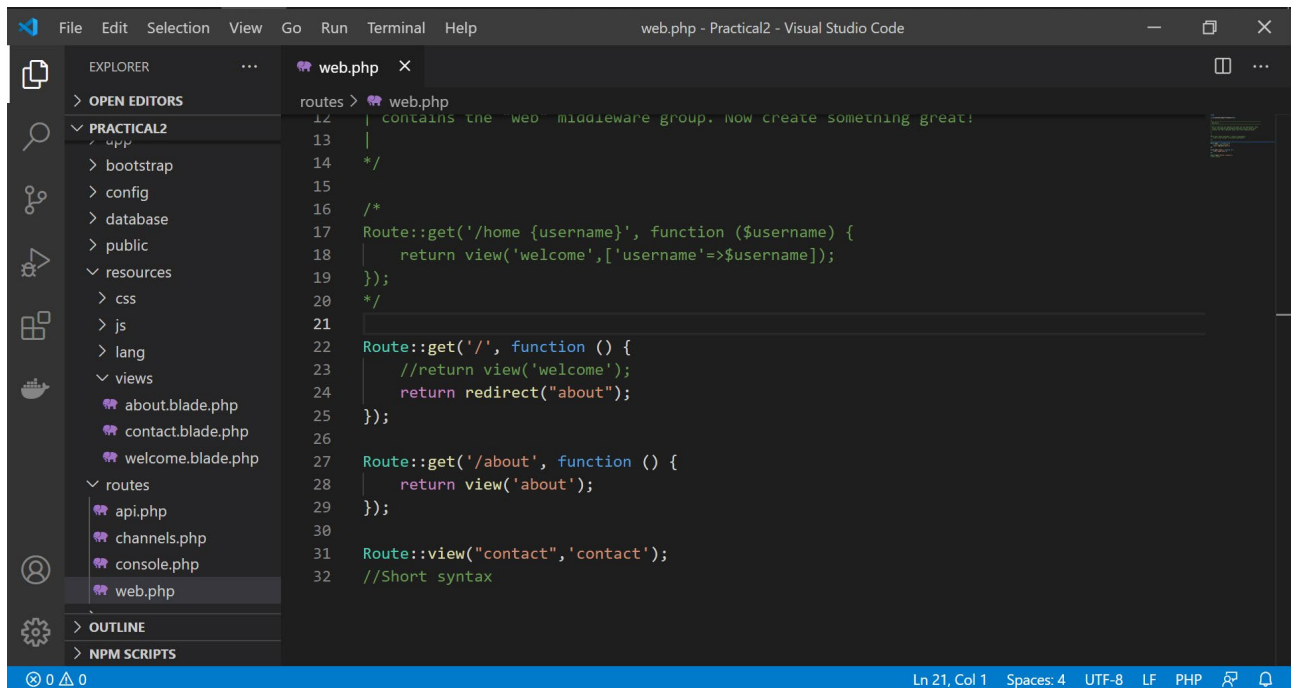


Figure 6: Using redirect in Route.

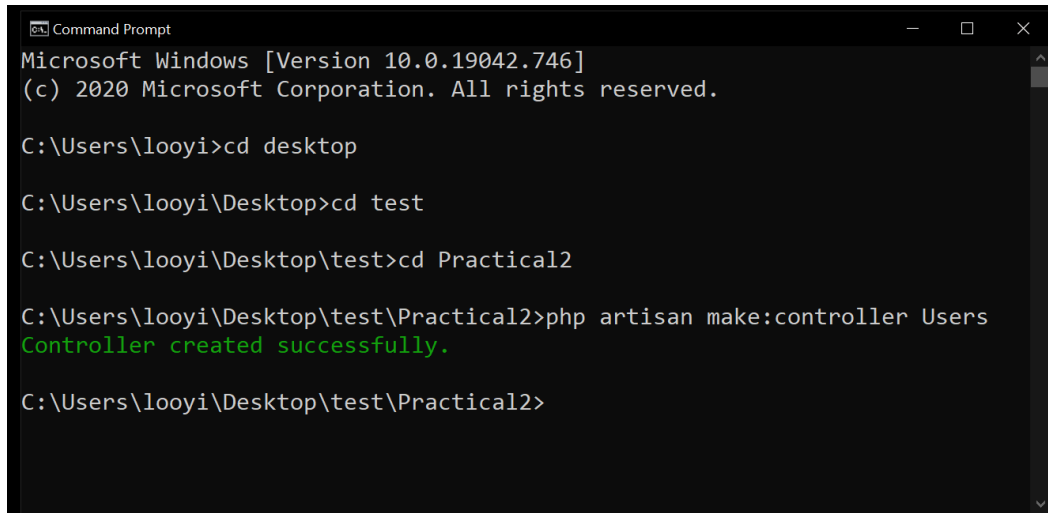
That’s all for Route in Laravel. The following practical session is exploration of Controller.

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.

To create a controller in Laravel web application, there are two ways:

- 1) Through the Artisan CLI “**php artisan make:controller**”. Figure 7 shows the example of creating “Users” controller.



```

Microsoft Windows [Version 10.0.19042.746]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Users\looyi>cd desktop

C:\Users\looyi\Desktop>cd test

C:\Users\looyi\Desktop\test>cd Practical2

C:\Users\looyi\Desktop\test\Practical2>php artisan make:controller Users
Controller created successfully.

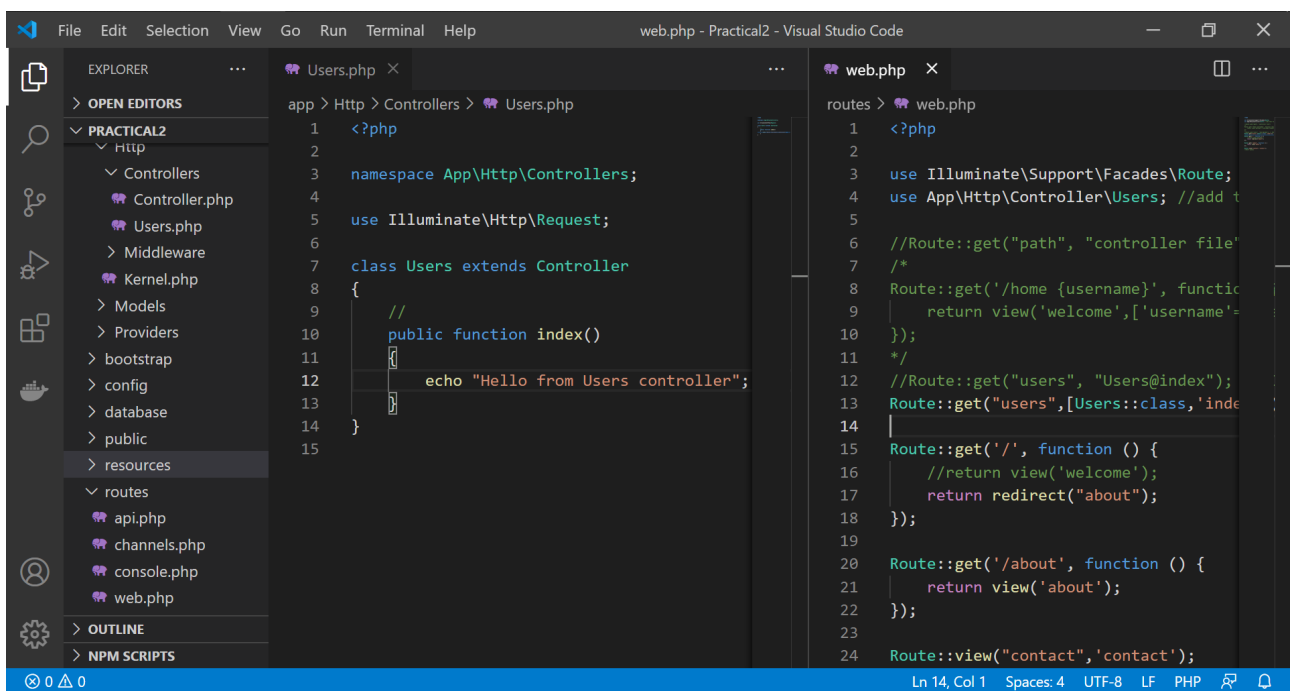
C:\Users\looyi\Desktop\test\Practical2>

```

Figure 7: Using Artisan CLI to create Users controller.

- 2) 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.



```

web.php - Practical2 - Visual Studio Code

EXPLORER
  PRACTICAL2
    Http
      Controllers
        Controller.php
        Users.php
      Middleware
      Kernel.php
    Models
    Providers
    bootstrap
    config
    database
    public
    resources
      routes
        api.php
        channels.php
        console.php
        web.php
    OUTLINE
    NPM SCRIPTS

app > Http > Controllers > Users.php
1  <?php
2
3  namespace App\Http\Controllers;
4
5  use Illuminate\Http\Request;
6
7  class Users extends Controller
8  {
9      //
10     public function index()
11     {
12         echo "Hello from Users controller";
13     }
14 }
15

routes > web.php
1  <?php
2
3  use Illuminate\Support\Facades\Route;
4  use App\Http\Controller\Users; //add t
5
6  //Route::get("path", "controller file"
7  /*
8  Route::get('/home {username}', functio
9  |   return view('welcome',['username'=
10  });
11  */
12  //Route::get("users", "Users@index");
13  Route::get("users",[Users::class,'inde
14  |
15  Route::get('/', function () {
16  |   //return view('welcome');
17  |   return redirect("about");
18  });
19
20  Route::get('/about', function () {
21  |   return view('about');
22  });
23
24  Route::view("contact",'contact');

```

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:

UECS3294 ADVANCED WEB APPLICATION DEVELOPMENT

```
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.

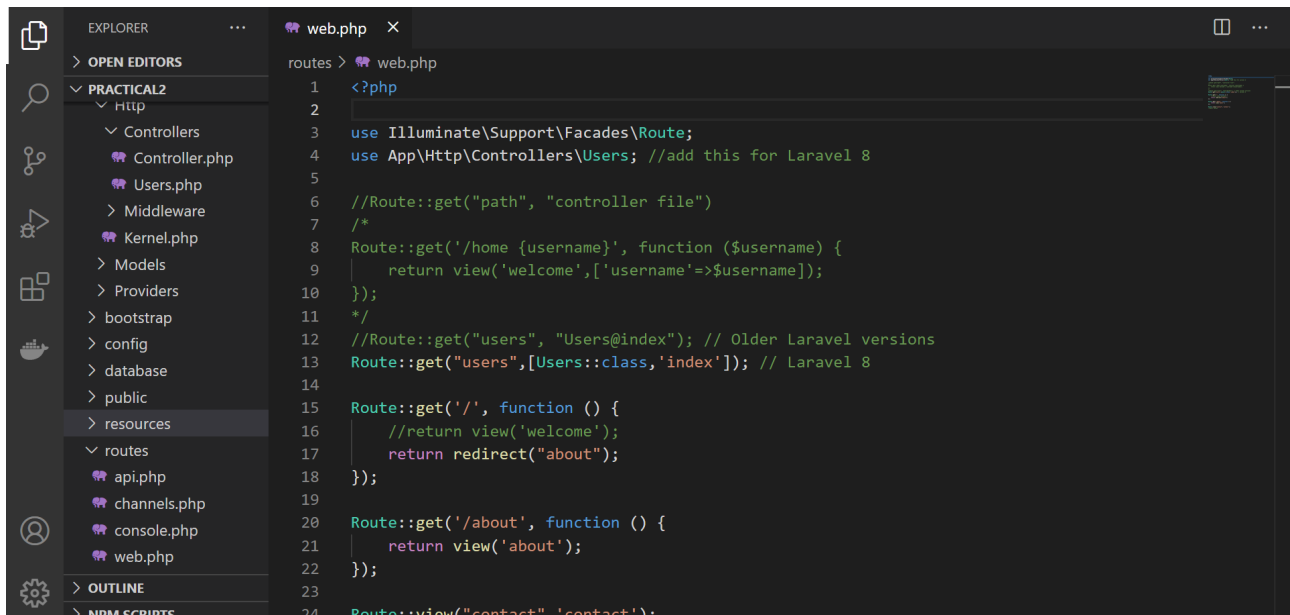


Figure 9: Routing to Controller in Laravel 8.

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.

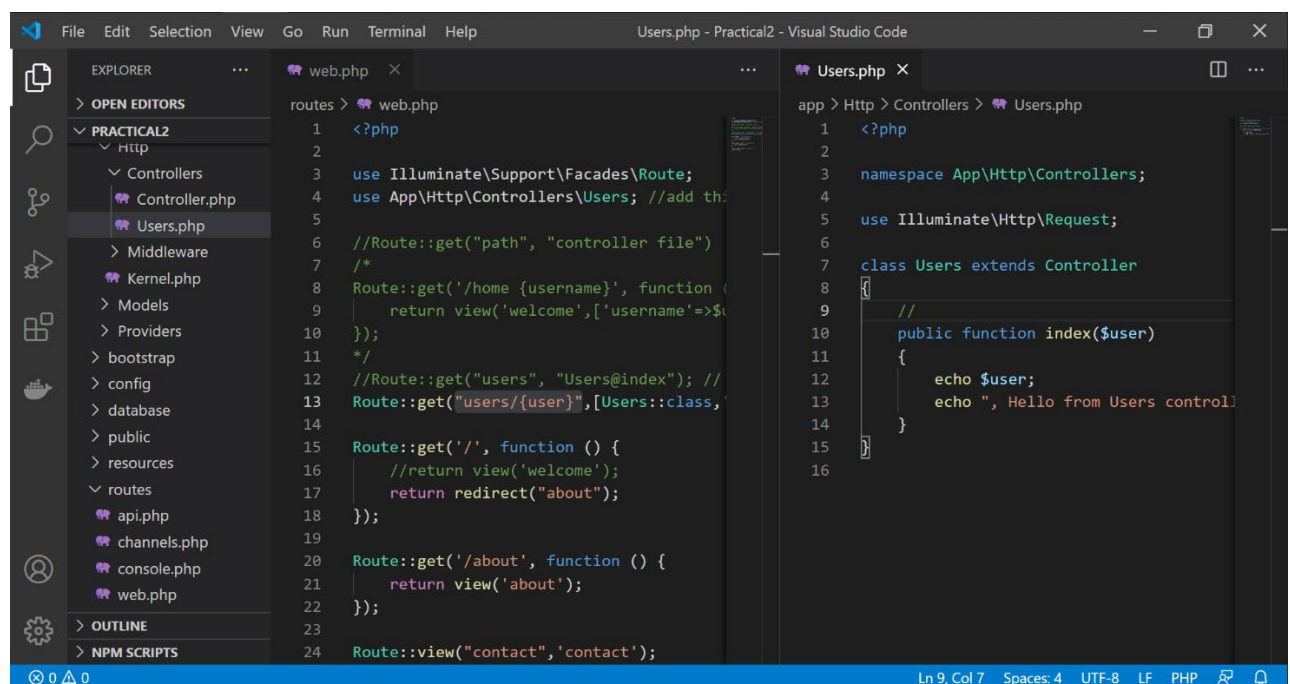


Figure 10: Additional parameters for passing data from URL.

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

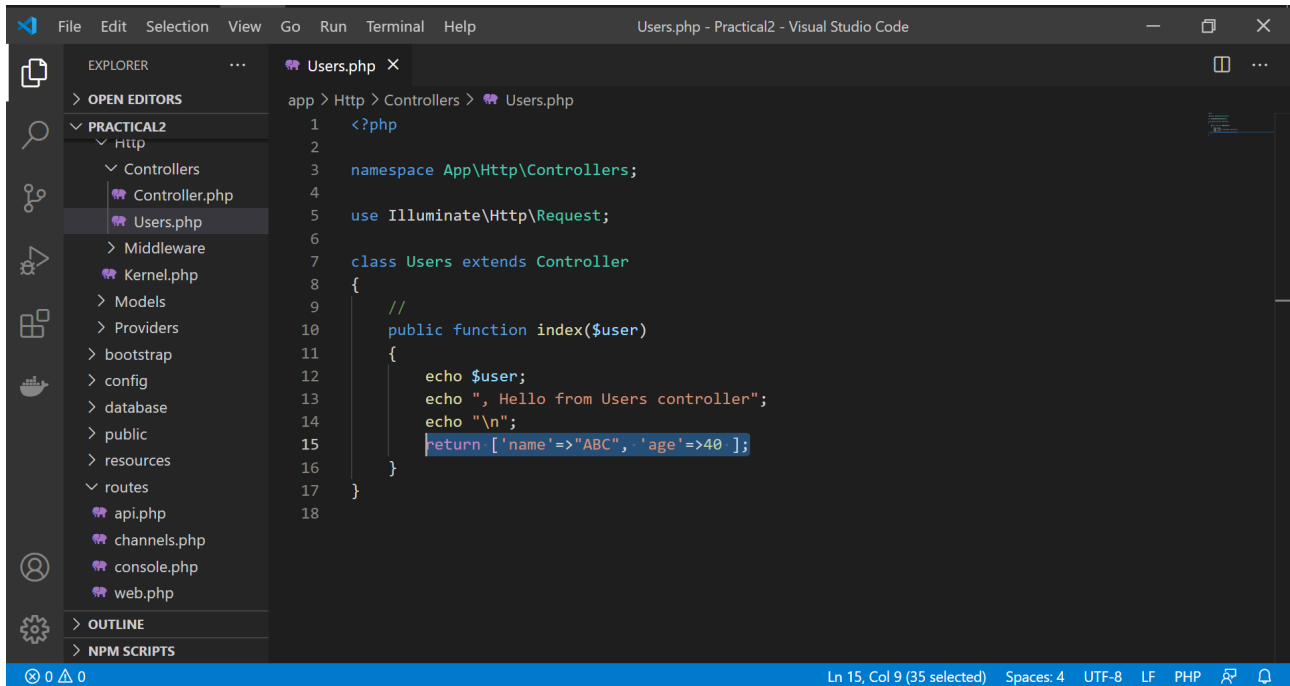


Figure 11: Using Controller as an API.

That's all for Controller in Laravel Framework. 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 3: 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.

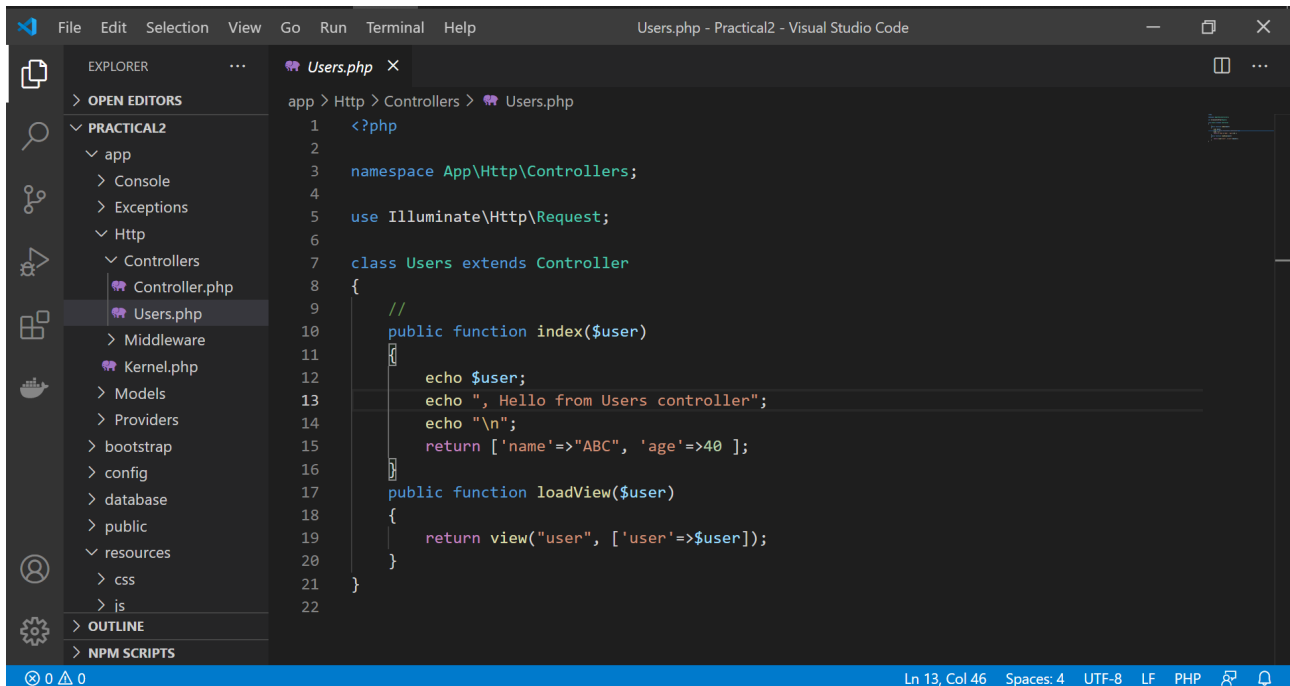


Figure 12: Users controller to call for user view.

Exercise 4: Create a “users” controller route which allow a username parameter in the URL to be passed to “user” view.

3.1. 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.

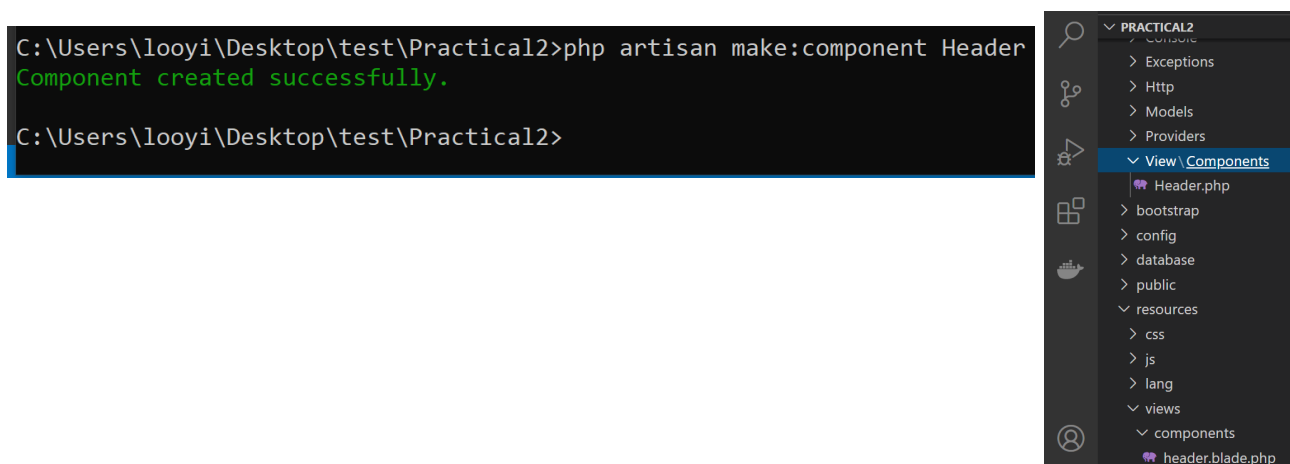
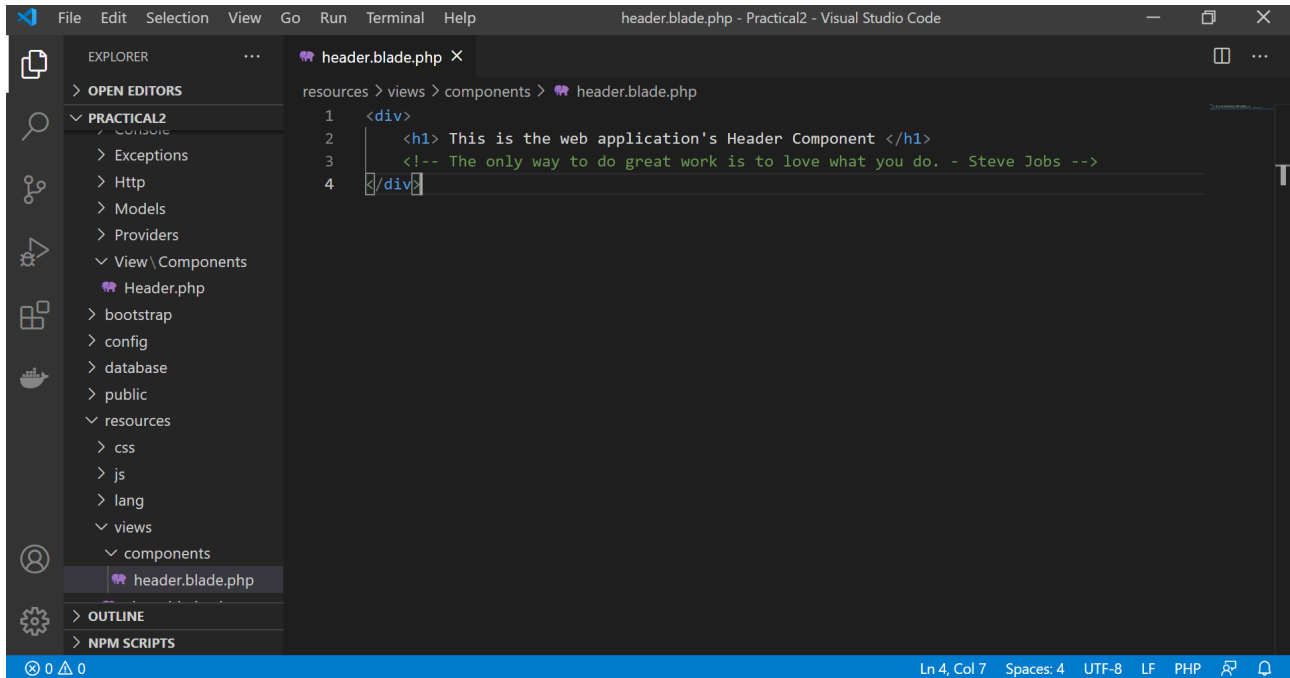


Figure 13: Create Header Component using Artisan CLI and resulting header component files in Laravel web application project folder.

UECS3294 ADVANCED WEB APPLICATION DEVELOPMENT

Once a Laravel Component is created, a php file is generated in “app” folder and a blade template HTML is generated in resources >> views.

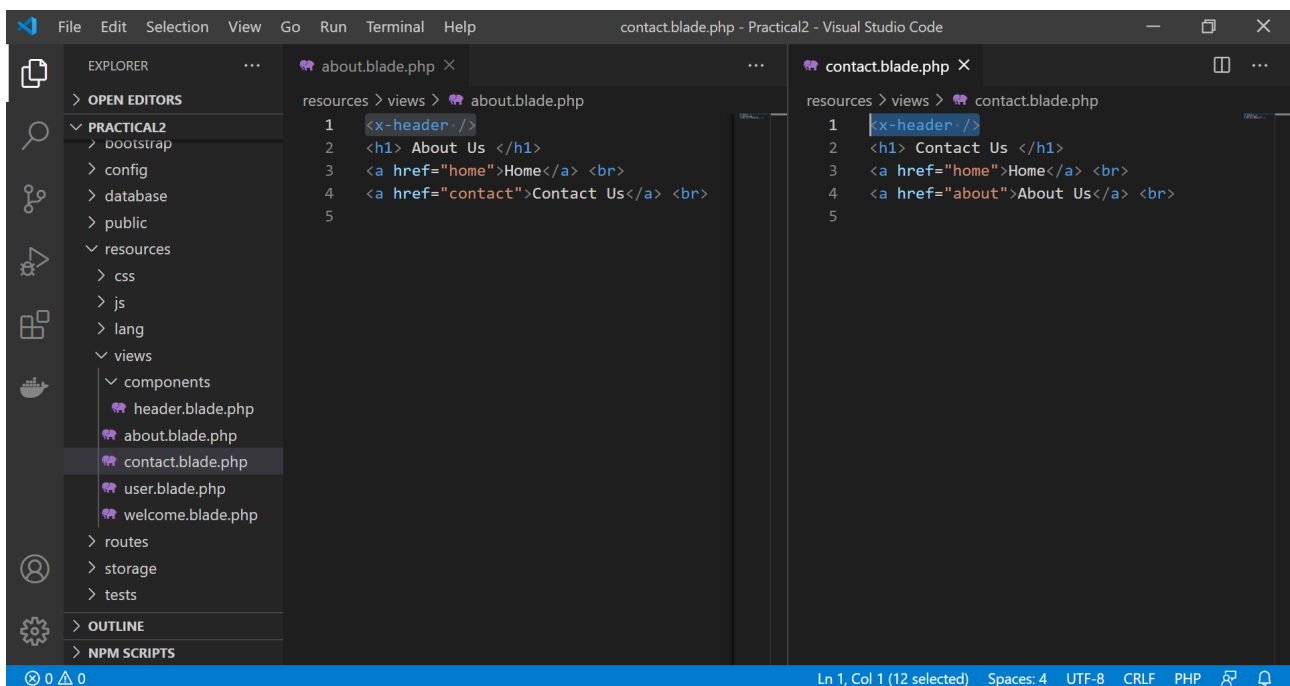
Stylize the header.blade.php file in order to create a visible output for any view file that will use the component as shown in Figure 14.



```
1 <div>
2   <h1> This is the web application's Header Component </h1>
3   <!-- The only way to do great work is to love what you do. - Steve Jobs -->
4 </div>
```

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.

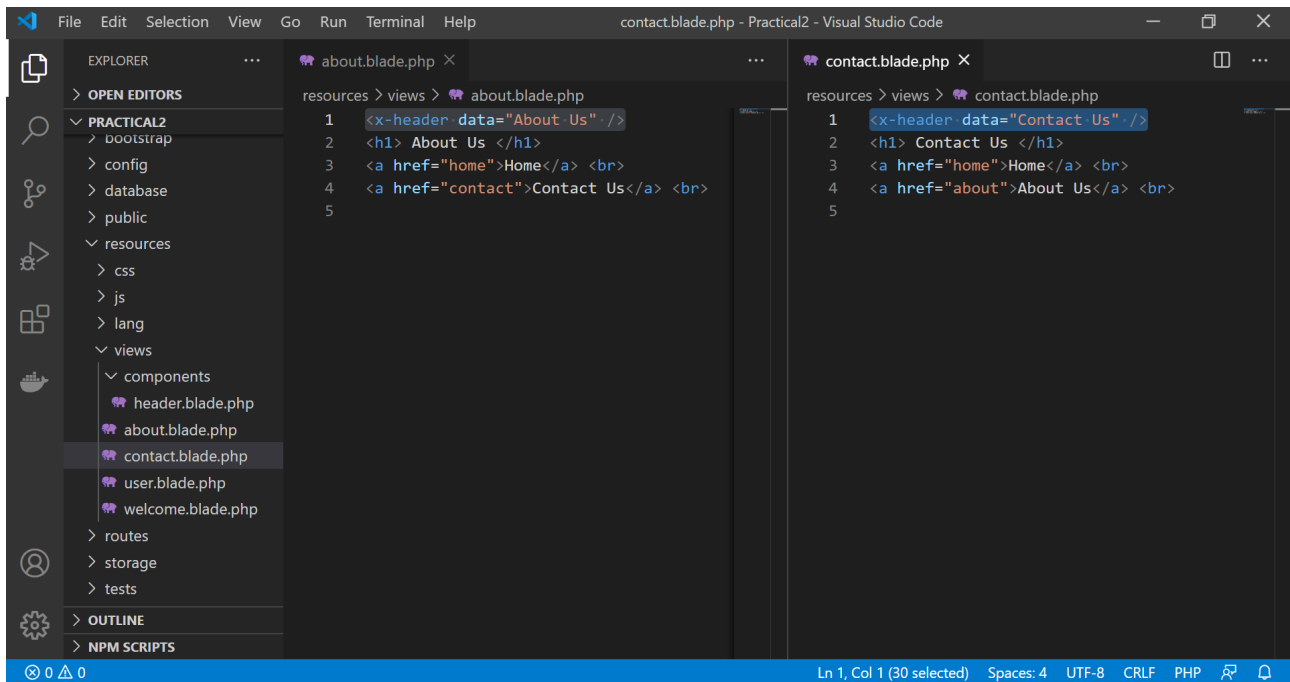


```
1 <x-header />
2 <h1> About Us </h1>
3 <a href="home">Home</a> <br>
4 <a href="contact">Contact Us</a> <br>
5
```

Figure 15: Using Header component in View.

UECS3294 ADVANCED WEB APPLICATION DEVELOPMENT

If we would like to format the Header Component to be specifically 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.



```
File Edit Selection View Go Run Terminal Help contact.blade.php - Practical2 - Visual Studio Code
```

EXPLORER

OPEN EDITORS

PRACTICAL2

- bootstrap
- config
- database
- public
- resources
 - css
 - js
 - lang
 - views
 - components
 - header.blade.php
 - about.blade.php
 - contact.blade.php
 - user.blade.php
 - welcome.blade.php
- routes
- storage
- tests

OUTLINE

NPM SCRIPTS

resources > views > about.blade.php

```
1 <x-header data="About Us" />
2 <h1> About Us </h1>
3 <a href="home">Home</a> <br>
4 <a href="contact">Contact Us</a> <br>
5
```

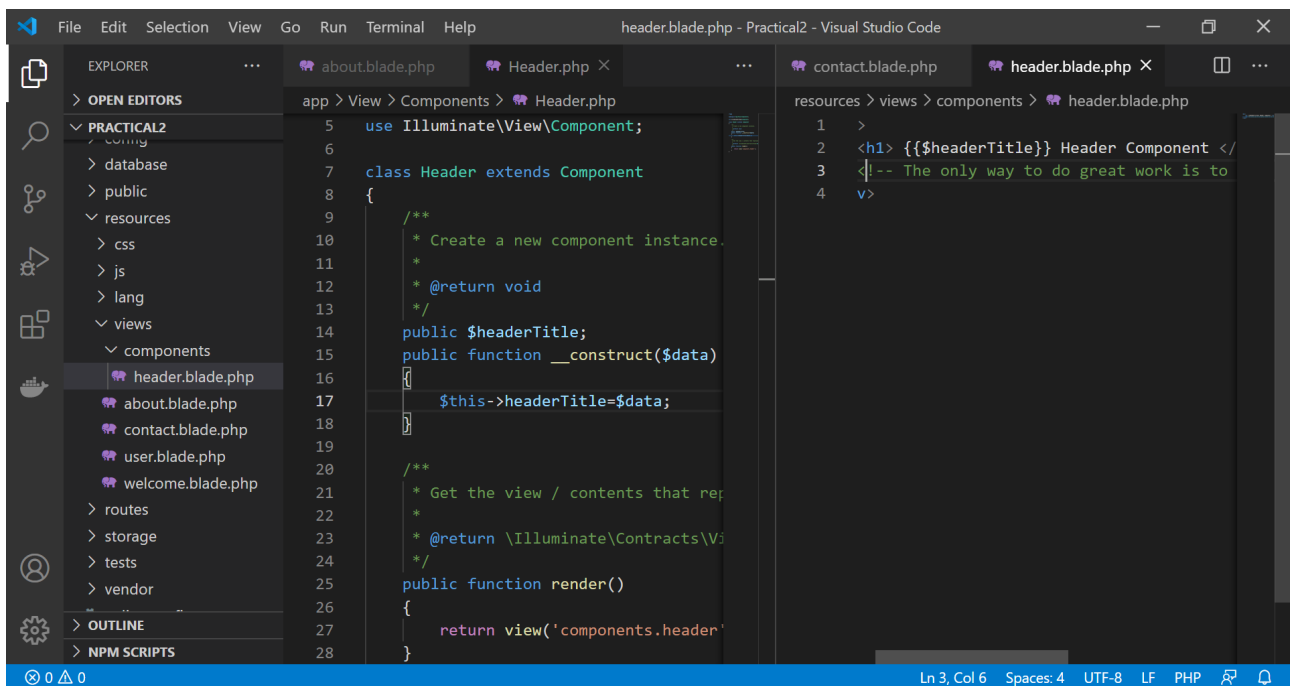
resources > views > contact.blade.php

```
1 <x-header data="Contact Us" />
2 <h1> Contact Us </h1>
3 <a href="home">Home</a> <br>
4 <a href="about">About Us</a> <br>
5
```

Ln 1, Col 1 (30 selected) Spaces: 4 UTF-8 CRLF PHP

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.



```
File Edit Selection View Go Run Terminal Help header.blade.php - Practical2 - Visual Studio Code
```

EXPLORER

OPEN EDITORS

PRACTICAL2

- bootstrap
- database
- public
- resources
 - css
 - js
 - lang
 - views
 - components
 - header.blade.php
 - about.blade.php
 - contact.blade.php
 - user.blade.php
 - welcome.blade.php
- routes
- storage
- tests
- vendor

OUTLINE

NPM SCRIPTS

app > View > Components > Header.php

```
5 use Illuminate\View\Component;
6
7 class Header extends Component
8 {
9     /**
10      * Create a new component instance.
11      *
12      * @return void
13      */
14     public $headerTitle;
15     public function __construct($data)
16     {
17         $this->headerTitle=$data;
18     }
19
20     /**
21      * Get the view / contents that represent the component.
22      *
23      * @return \Illuminate\Contracts\View\Factory|\Illuminate\View\View
24      */
25     public function render()
26     {
27         return view('components.header');
28     }
29 }
```

resources > views > components > header.blade.php

```
1 <
2 <h1> {{{headerTitle}}} Header Component </
3 <!-- The only way to do great work is to
4 v>
```

Ln 3, Col 6 Spaces: 4 UTF-8 LF PHP

Figure 17: Processing data passed from View to Component.

3.2. 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.

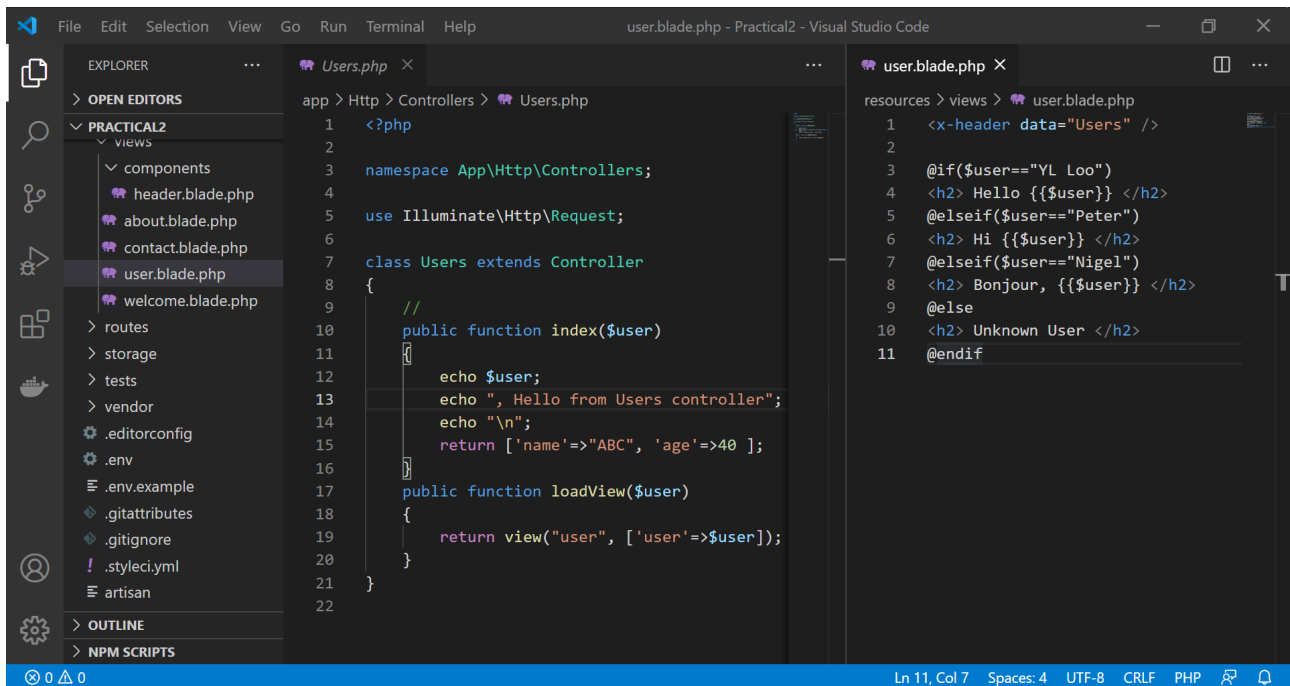


Figure 18: Conditional Function in Blade Template.

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

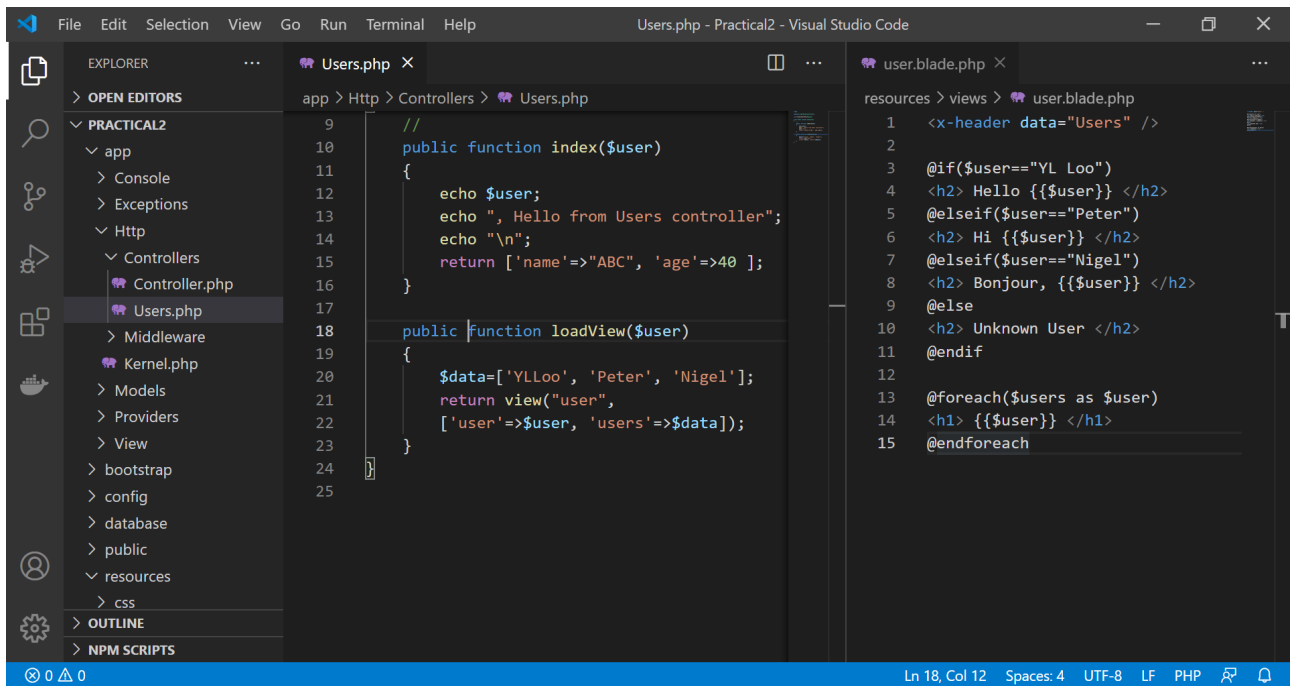


Figure 19: Loop Function in in Blade Template.

3.3. 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.

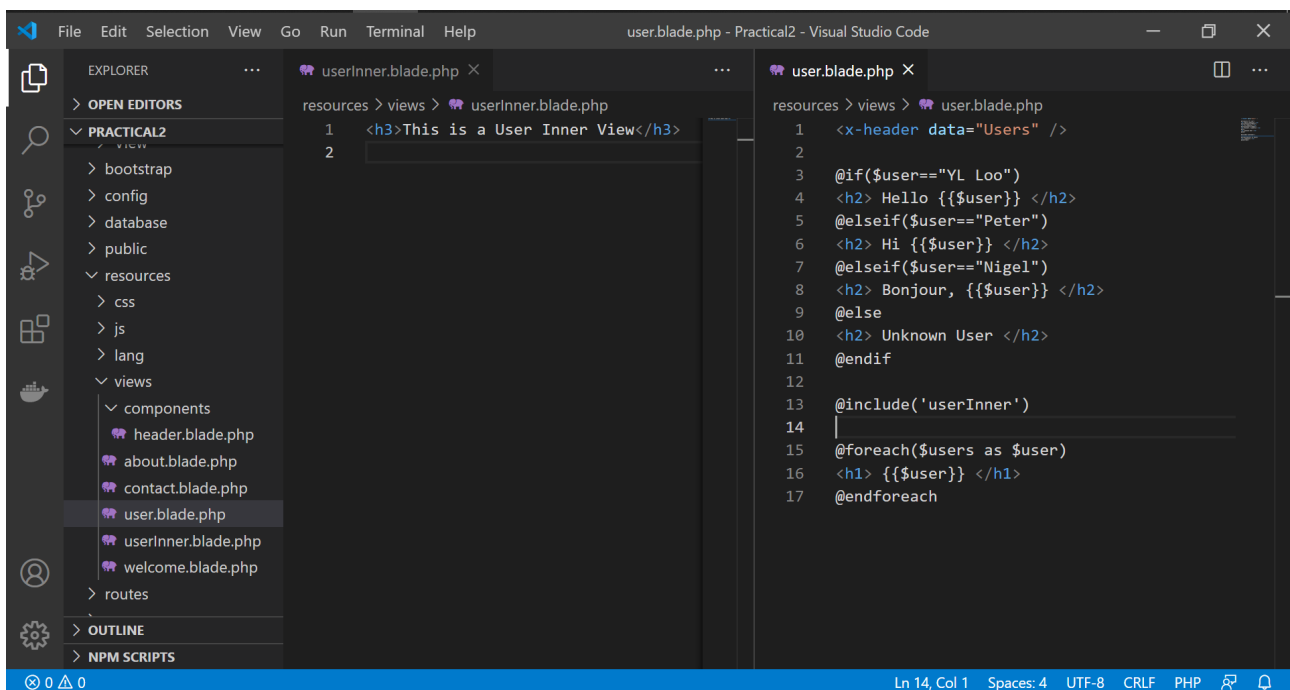
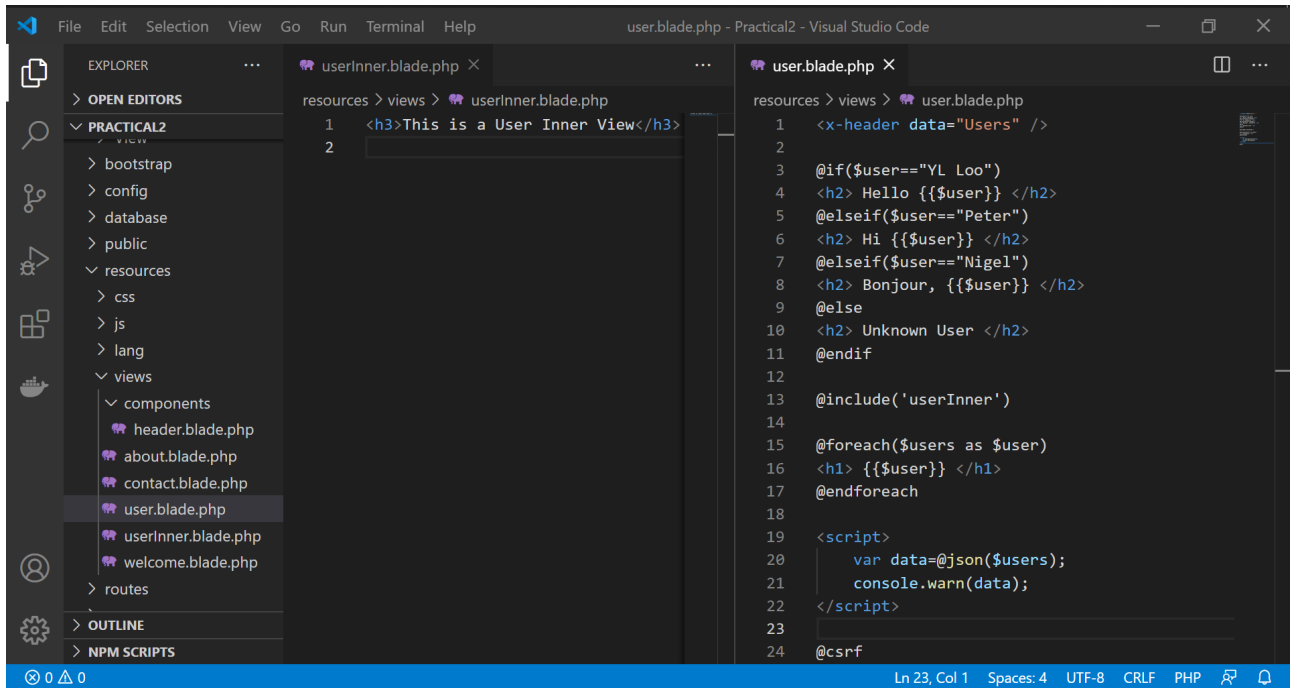


Figure 20: Using Blade Template for View in View.

3.4. 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.*



```
File Edit Selection View Go Run Terminal Help user.blade.php - Practical2 - Visual Studio Code

EXPLORER
OPEN EDITORS
PRACTICAL2
  bootstrap
  config
  database
  public
  resources
    css
    js
    lang
    views
      components
        header.blade.php
        about.blade.php
        contact.blade.php
        user.blade.php
        userInner.blade.php
        welcome.blade.php
    routes
  OUTLINE
  NPM SCRIPTS

resources > views > userInner.blade.php
1 <h3>This is a User Inner View</h3>
2

resources > views > user.blade.php
1 <x-header data="Users" />
2
3 @if($user=="YL Loo")
4 <h2> Hello {{$user}} </h2>
5 @elseif($user=="Peter")
6 <h2> Hi {{$user}} </h2>
7 @elseif($user=="Nigel")
8 <h2> Bonjour, {{$user}} </h2>
9 @else
10 <h2> Unknown User </h2>
11 @endif
12
13 @include('userInner')
14
15 @foreach($users as $user)
16 <h1> {{$user}} </h1>
17 @endforeach
18
19 <script>
20   var data=@json($users);
21   console.warn(data);
22 </script>
23
24 @csrf
```

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