



Outlines

€Routing in Laravel

€Controllers in Laravel

€View in Laravel

€Blade Template

€Components in laravel

```
└─ app/Http/Controllers/ChirpController.php

<?php

class ChirpController extends Controller
{
    public function store(Request $request)
    {
        $validated = $request->validate([
            'message' => 'required|string|max:255',
        ]);

        $request->user()->chirps()->create($validated);

        return redirect(route('chirps.index'));
    }
}
```



Routing in laravel

Routing in Laravel

1. Defining Routes

In Laravel, routes are defined in the `routes/web.php` file. Routes usually specify which URL should be handled by which controller or closure.

Example: Basic Route Definition

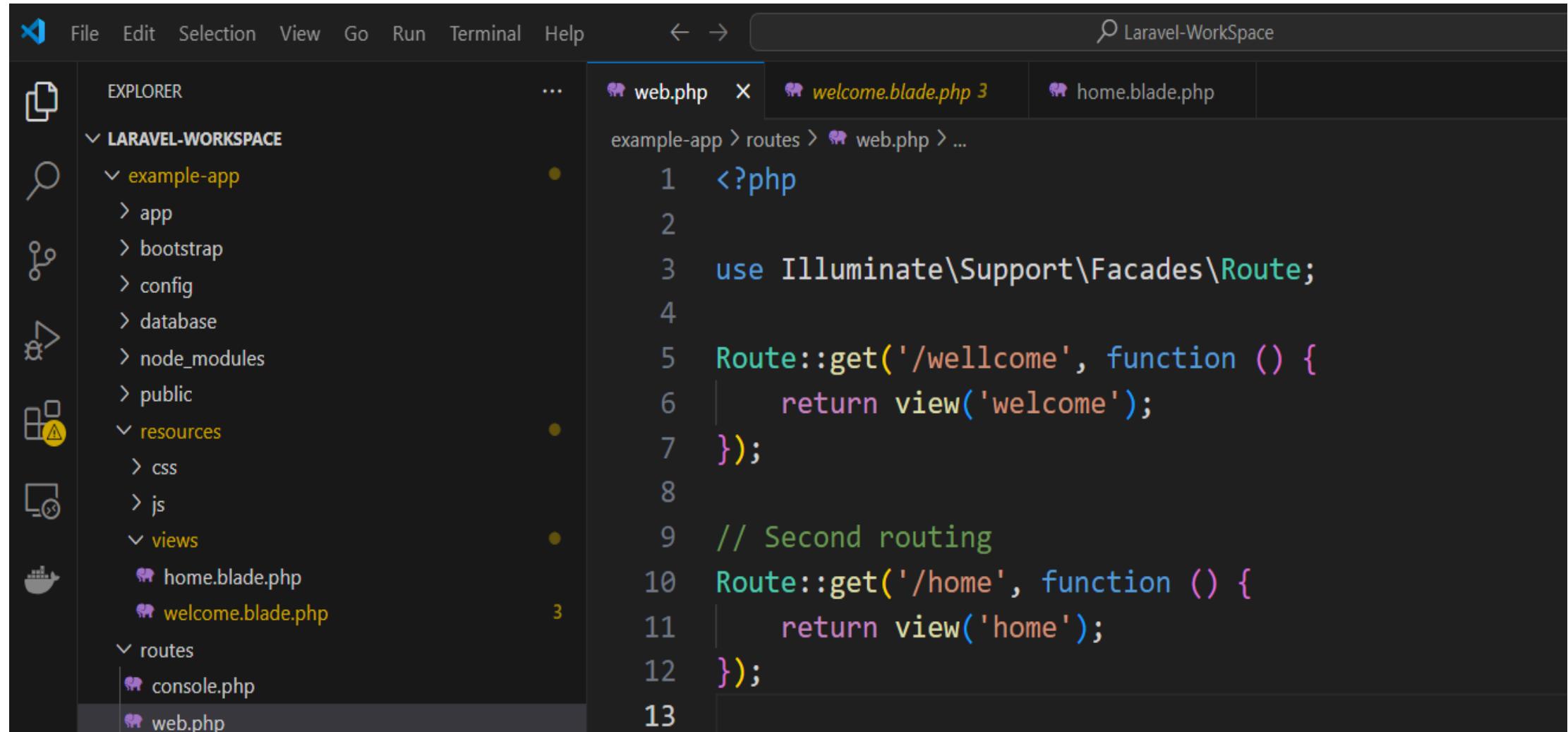
php

 Copy code

```
Route::get('/welcome', function () {
    return 'Welcome to Laravel!';
});
```

- The above code defines a route that responds to a GET request to the `/welcome` URL and returns the text "Welcome to Laravel!".

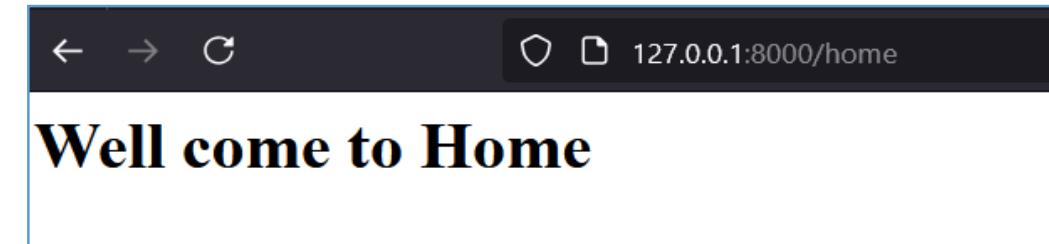
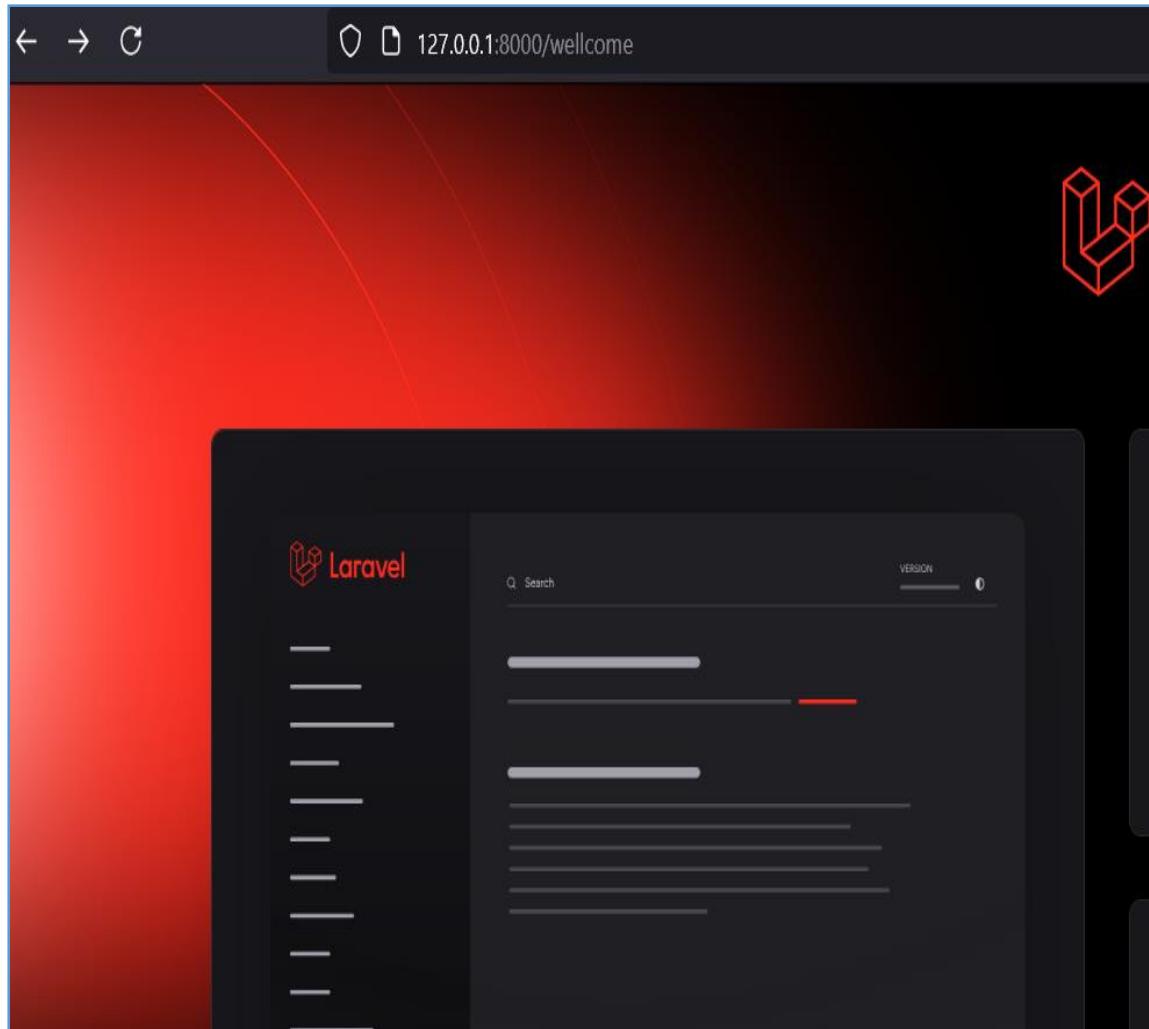
Example



The screenshot shows a Visual Studio Code interface with an orange border. The left sidebar is the Explorer view, showing a tree structure of a Laravel workspace named 'example-app'. The 'routes' folder contains 'web.php'. The 'views' folder contains 'welcome.blade.php' and 'home.blade.php'. The top right shows tabs for 'web.php', 'welcome.blade.php 3', and 'home.blade.php'. The main editor area displays the contents of 'web.php'.

```
1 <?php
2
3 use Illuminate\Support\Facades\Route;
4
5 Route::get('/wellcome', function () {
6     return view('welcome');
7 });
8
9 // Second routing
10 Route::get('/home', function () {
11     return view('home');
12 });
13
```

Cont. ...



Cont. ...

2. Route Methods

Laravel provides different HTTP methods (GET, POST, PUT, DELETE) to define routes for specific types of requests. These can be used to handle actions such as form submissions, data updates, and deletions.

Example: Handling GET and POST requests

```
php

// GET route to display a form
Route::get('/form', function () {
    return view('form');
});

// POST route to handle form submission
Route::post('/submit', function () {
    // Logic to handle submitted form data
    return 'Form submitted!';
});
```

Contact Us

Name:

Email:

Message:

Submit

- The first route displays a form, while the second route handles form submissions using the POST method.

Cont. ...

Example: PUT and DELETE methods

php

 Copy code

```
// PUT route to update a resource
Route::put('/update/{id}', function ($id) {
    // Logic to update a resource with the given ID
    return "Updated resource with ID: $id";
});

// DELETE route to delete a resource
Route::delete('/delete/{id}', function ($id) {
    // Logic to delete a resource with the given ID
    return "Deleted resource with ID: $id";
});
```

- These routes handle updating and deleting resources based on their unique ID.

Cont. ...

3. Route Parameters

Laravel allows you to define dynamic routes with parameters. These parameters are passed to the route's function or controller.

Example: Route with Required Parameter

```
php
```

 Copy code

```
Route::get('/user/{id}', function ($id) {
    return "User ID: " . $id;
});
```

← → ⌂



127.0.0.1:8000/user/12

User ID: 12

- This route captures a dynamic parameter `id`, which can be any value. For example, accessing `/user/5` will output "User ID: 5".

Cont. ...

Example: Route with Optional Parameter

php

 Copy code

```
Route::get('/profile/{name?}', function ($name = 'Guest') {  
    return "Profile Name: " . $name;  
});
```

- In this example, the parameter `name` is optional. If no name is provided, it defaults to "Guest".

Cont. ...

Example: Defining a Named Route

```
php
```

```
Route::get('/dashboard', function () {
    return 'Dashboard';
})->name('dashboard');
```

 Copy code

- You can now refer to this route using the route's name (`dashboard`) anywhere in the application, for example, when generating links.

Example: Generating URLs Using Named Routes

```
php
```

```
// In a Blade template or controller
$url = route('dashboard'); // Generates the URL for the dashboard route
```

 Copy code

Cont. ...

5. Route Groups

Route groups allow you to apply common attributes like middleware, namespaces, or prefixes to multiple routes.

Example: Grouping Routes with a Common Prefix

php

 Copy code

```
Route::prefix('admin')->group(function () {
    Route::get('/users', function () {
        return 'Admin Users';
    });

    Route::get('/settings', function () {
        return 'Admin Settings';
    });
});
```

- This defines a group of routes with the prefix `/admin`. The resulting URLs would be `/admin/users` and `/admin/settings`.

Cont. ...

6. Route Redirects and Route Views

Laravel makes it easy to quickly define routes that either redirect to another URL or return a view directly.

Example: Route Redirect

php

 Copy code

```
Route::redirect('/home', '/dashboard');
```

- This redirects any request to `/home` to `/dashboard`.

Cont. ...

Example: Returning a View from a Route

php

 Copy code

```
Route::view('/welcome', 'welcome');
```

- This route directly returns the `welcome.blade.php` view when the `/welcome` URL is accessed.

Cont. ...

7. Route Middleware

Middleware is used to filter requests entering your application, like checking if a user is authenticated before allowing access to certain routes.

Example: Applying Middleware to Routes

php

 Copy code

```
Route::get('/dashboard', function () {
    return 'Dashboard';
})->middleware('auth');
```

- This route is only accessible to authenticated users, as enforced by the `auth` middleware.

Cont. ...

8. Resourceful Routing

Laravel provides a simple way to create a controller that handles all CRUD operations with a single line of code using resourceful routes.

Example: Defining a Resource Controller

php

 Copy code

```
Route::resource('posts', 'PostController');
```

- This automatically sets up routes for all the common CRUD actions (index, create, store, show, edit, update, destroy) in the `PostController`.

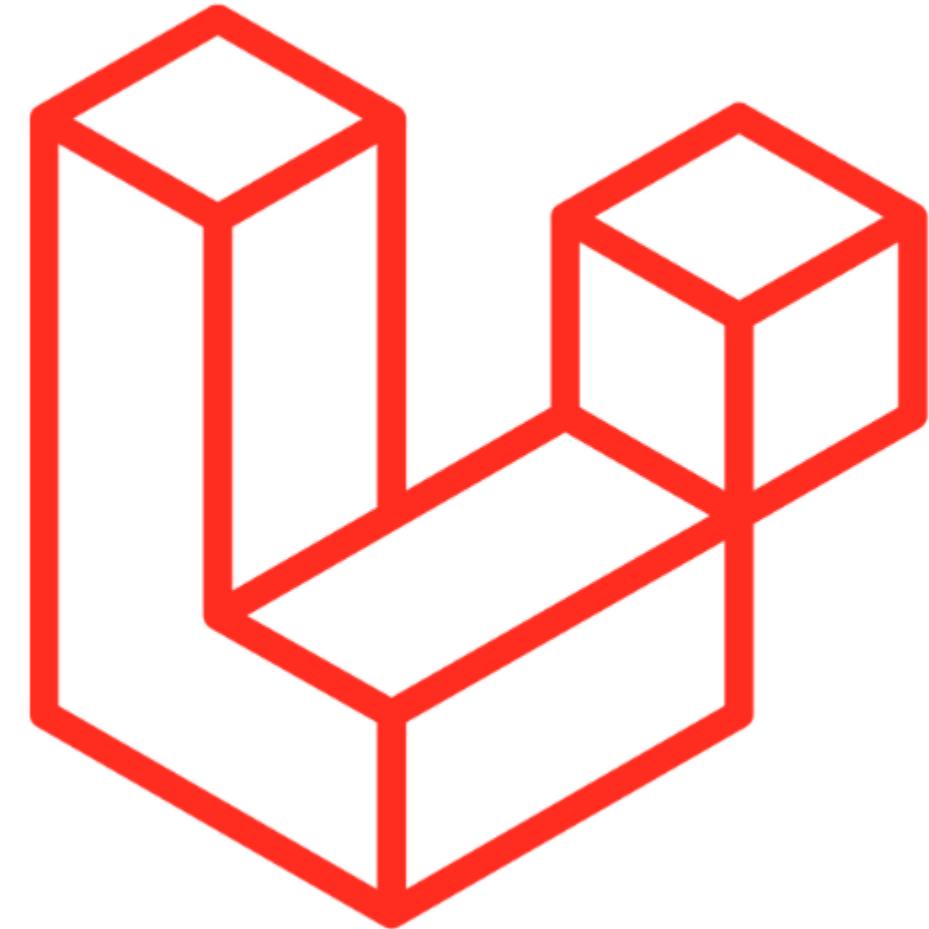
Cont. ...

Summary of Key Routing Concepts:

- **Defining Routes:** Basic routes are created in `routes/web.php`, with support for GET, POST, PUT, DELETE methods.
- **Route Parameters:** Dynamic values are captured in routes, and parameters can be required or optional.
- **Named Routes:** Simplify route management by using names instead of hardcoded URLs.
- **Route Groups:** Apply common behavior like middleware or prefixes to multiple routes.
- **Middleware:** Routes can be filtered through middleware for tasks like authentication.
- **Redirects and Views:** Easily redirect users or return views directly from routes.
- **Resourceful Routing:** Handle all CRUD operations in a controller using resource routes.

LARAVEL

Controllers



Controllers in Laravel

- In Laravel, controllers are a vital part of the MVC (Model-View-Controller) architecture, which separates the **application logic** from the user **interface**.
- Controllers handle the **incoming HTTP requests**, **process data** (usually with the help of models), and return responses (typically views or JSON data).
- Controllers allow you to organize your application logic more cleanly compared to defining logic in the route files.

Creating Controller

1. Creating a Controller

To create a controller, you can use Laravel's Artisan command-line tool. You can define controllers to handle various HTTP requests and organize your application code.

Example: Creating a Basic Controller

bash

 Copy code

```
php artisan make:controller UserController
```

- This command creates a new `UserController` inside the `app/Http/Controllers` directory.

Cont. ...

The screenshot shows the Visual Studio Code interface with the title bar "Laravel-WorkSpace". The left sidebar (EXPLORER) displays the project structure of "example-app" under "LARAVEL-WORKSPACE". The "app" folder contains "Http\Controllers", "Controller.php", "UserController.php", and other files like "Models", "Providers", "bootstrap", "config", "database", "node_modules", "public", "resources", "routes", "storage", "tests", "vendor", and configuration files (.editorconfig, .env, .env.example, .gitattributes, .gitignore). The "TERMINAL" tab at the bottom shows the command \$ php artisan make:controller UserController being run, with a success message indicating the controller was created successfully.

```
<?php
namespace App\Http\Controllers;
use Illuminate\Http\Request;
class UserController extends Controller
{
    //
}
```

TERMINAL

```
yitayew@DESKTOP-NPGGF04 MINGW64 ~/Desktop/Laravel-WorkSpace/example-app
$ php artisan make:controller UserController
INFO Controller [C:\Users\yitayew\Desktop\Laravel-WorkSpace\example-app\app\Http\Controllers\UserController.php] created successfully.
```

Cont. ...

2. Routing to a Controller

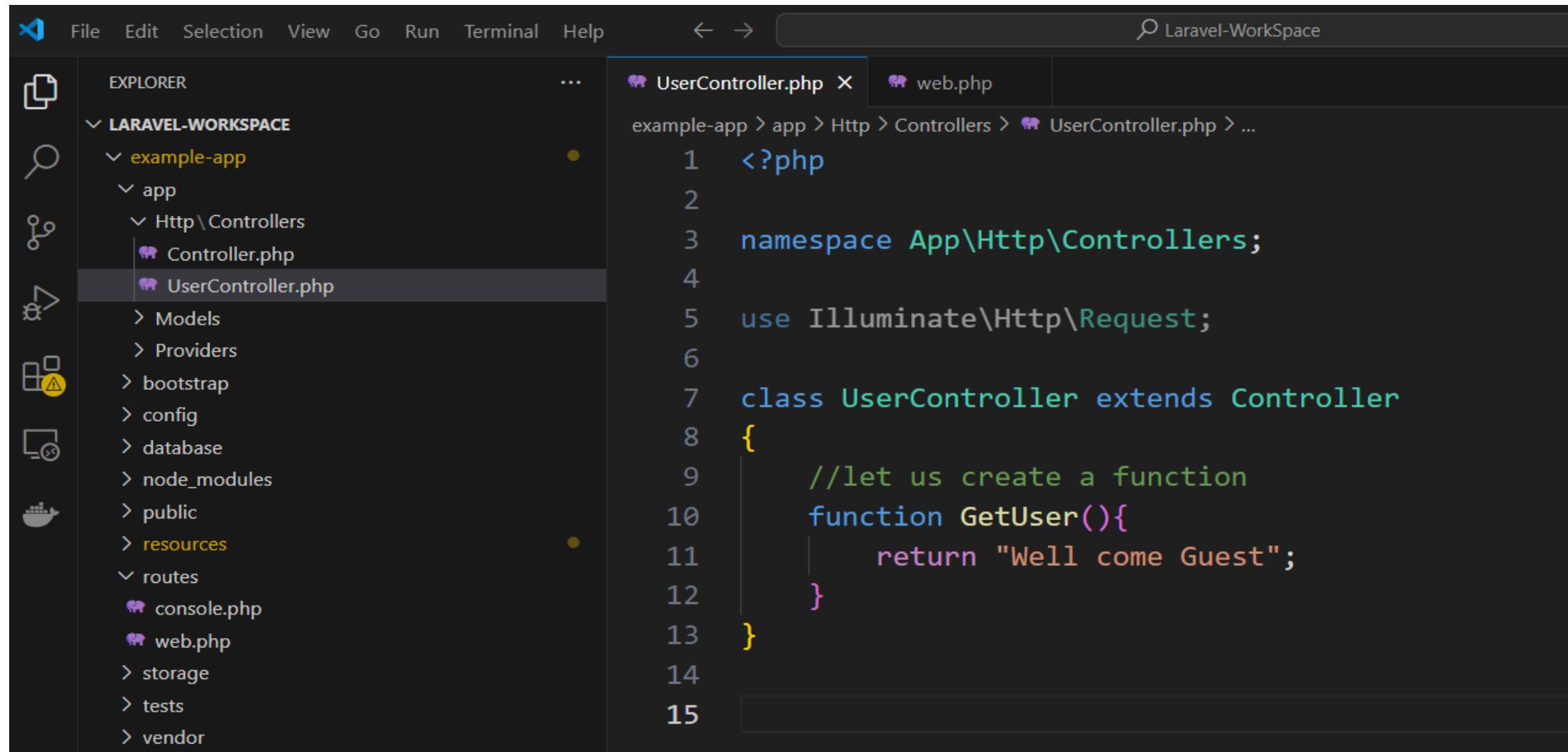
Once the controller is created, you can define routes that map to the controller's methods. This allows your application to handle various URLs and HTTP methods using controller actions.

Example: Defining Routes for Controller Methods

```
php Copy code
// Defining routes for the UserController
Route::get('/users', [UserController::class, 'index']);
Route::get('/users/{id}', [UserController::class, 'show']);
```

- When accessing `/users`, the `index` method of the `UserController` is executed.
- When accessing `/users/{id}`, the `show` method is executed, displaying the user's ID.

Defining Controller (User Controller)



The screenshot shows a code editor interface with a dark theme. On the left is the Explorer sidebar, which lists the project structure under 'LARAVEL-WORKSPACE'. The 'UserController.php' file in the 'Http\Controllers' directory is currently selected and highlighted with a grey background. The main editor area displays the code for this controller:

```
1 <?php
2
3 namespace App\Http\Controllers;
4
5 use Illuminate\Http\Request;
6
7 class UserController extends Controller
8 {
9     //let us create a function
10    function GetUser(){
11        return "Well come Guest";
12    }
13 }
14
15 
```

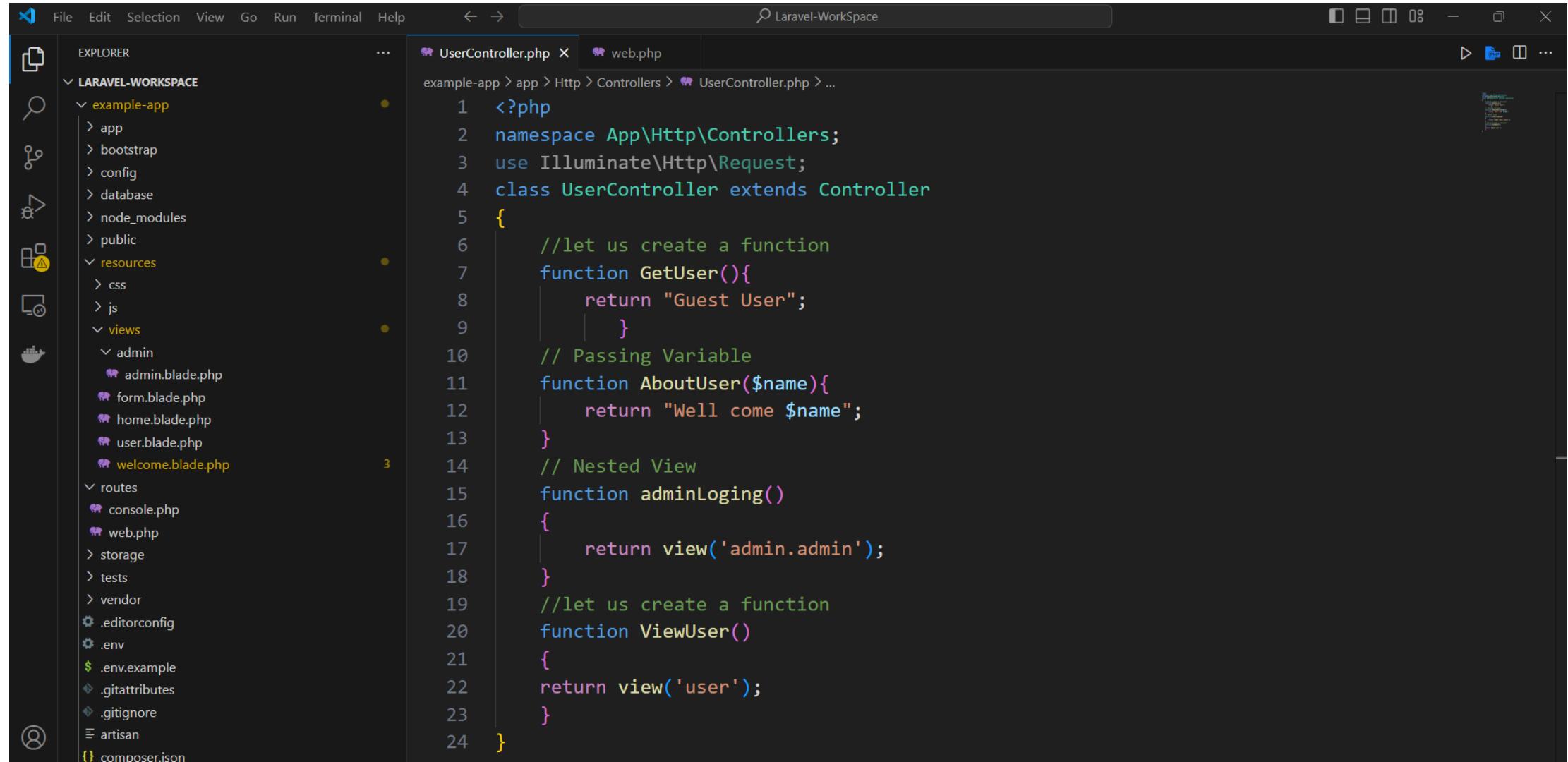
Creating Route for UserController

The screenshot shows a Visual Studio Code interface with an orange header bar. The title bar says "Laravel-WorkSpace". The left sidebar is the Explorer view, showing a project structure under "LARAVEL-WORKSPACE": "example-app" (with "app", "Http\Controllers", "Models", "Providers", "bootstrap", "config", "database", "node_modules", "public", "resources", "routes"), "routes" (with "console.php" and "web.php"), and a ".gitignore" file. The main editor area shows "UserController.php" and "web.php". The "web.php" tab is active, displaying the following code:

```
35
36
37 // Acesssing controllers, first we have to import controller
38
39 // use app\Http\Controllers\UserController; or
40 use App\Http\Controllers\UserController;
41
42 Route::get('user',[UserController::class,' GetUser']);
43
44
45
```

Below the editor, a terminal window shows the command "127.0.0.1:8000/user" and the output "Well come Guest".

Calling From View



The screenshot shows a Visual Studio Code interface with the title bar "Laravel-WorkSpace". The left sidebar displays the file structure of a Laravel application named "example-app". The "EXPLORER" view shows files like app, bootstrap, config, database, node_modules, public, resources (css, js, views), routes, storage, tests, vendor, .editorconfig, .env, .env.example, .gitattributes, .gitignore, artisan, and composer.json. The "LARAVEL-WORKSPACE" section highlights the "resources/views" directory, which contains admin, form.blade.php, home.blade.php, user.blade.php, and welcome.blade.php. The main editor area shows two files: "UserController.php" and "web.php". "UserController.php" contains the following code:

```
1 <?php
2 namespace App\Http\Controllers;
3 use Illuminate\Http\Request;
4 class UserController extends Controller
5 {
6     //let us create a function
7     function GetUser(){
8         return "Guest User";
9     }
10    // Passing Variable
11    function AboutUser($name){
12        return "Well come $name";
13    }
14    // Nested View
15    function adminLogging()
16    {
17        return view('admin.admin');
18    }
19    //let us create a function
20    function ViewUser()
21    {
22        return view('user');
23    }
24 }
```

"web.php" contains the following code:

```
1 <?php
2 require __DIR__ . '/../vendor/autoload.php';
3 $app = require_once __DIR__ . '/../bootstrap/app.php';
4 $app->run();
```

Cont. ...

```
37 // Acesssing controllers, first we have to import controller
38
39 // use app\Http\Controllers\UserController; or
40 use App\Http\Controllers\UserController;
41
42 Route::get('user',[UserController::class,'ViewUser']);
43 Route::get('about/{name}',[UserController::class,'AboutUser']);
44 Route::get('admin-login',[UserController::class,'adminLoging']);
```



Cont. ...

3. Resource Controllers

Laravel provides a powerful way to manage CRUD (Create, Read, Update, Delete) operations using **Resource Controllers**. A resource controller automatically creates routes for all standard CRUD actions (index, create, store, show, edit, update, destroy).

Example: Creating a Resource Controller

bash

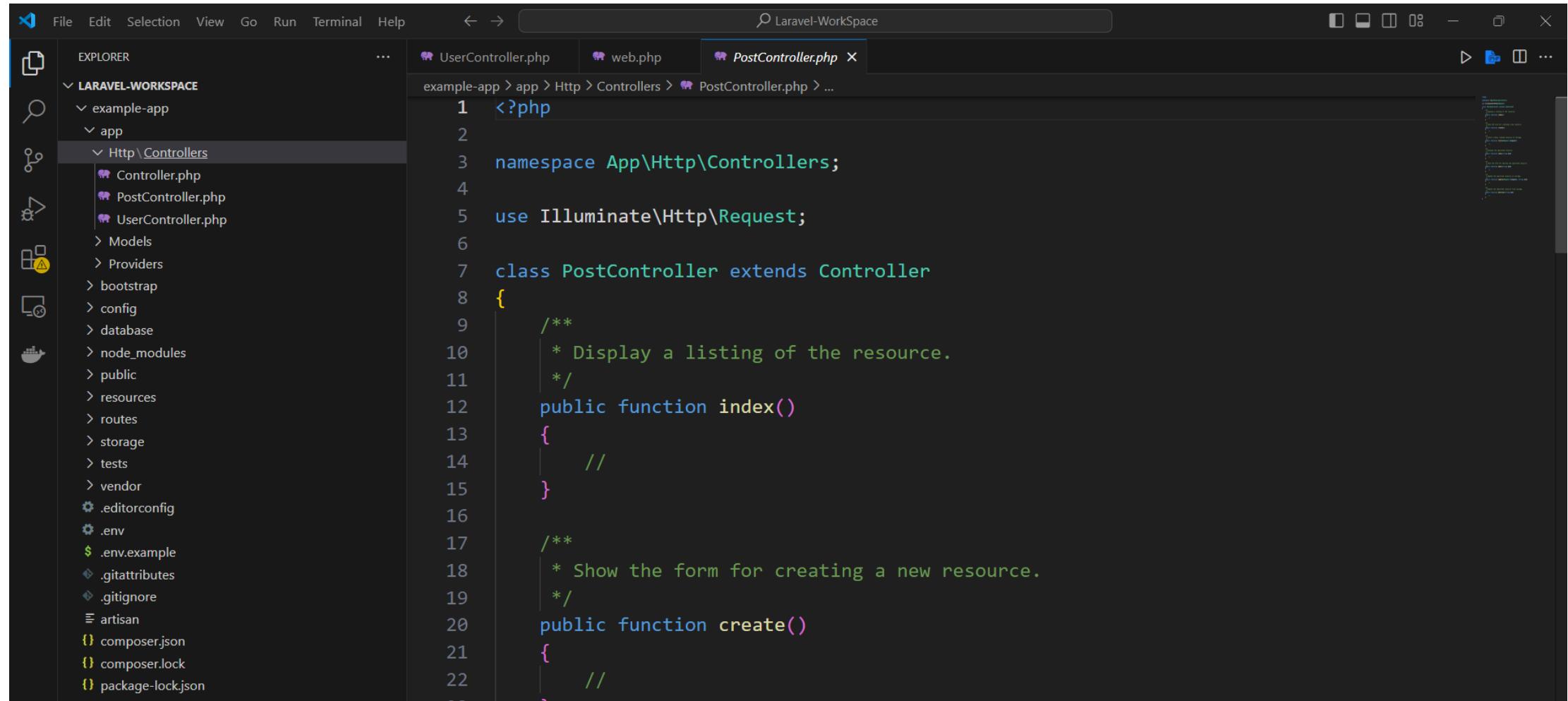
 Copy code

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

Cont. ...

- This command creates a `PostController` with pre-defined methods for handling CRUD operations:
 - `index()` - List all posts.
 - `create()` - Show form for creating a new post.
 - `store()` - Save the new post to the database.
 - `show($id)` - Show a specific post.
 - `edit($id)` - Show form to edit a post.
 - `update($id)` - Update a specific post.
 - `destroy($id)` - Delete a specific post.

Cont. ...



The screenshot shows a code editor interface with the title "Laravel-WorkSpace". The left sidebar is an "EXPLORER" view showing the file structure of a Laravel application named "example-app". Under "app", there is a "Http\Controllers" folder containing "Controller.php", "PostController.php", and "UserController.php". Other visible files include "Models", "Providers", "bootstrap", "config", "database", "node_modules", "public", "resources", "routes", "storage", "tests", "vendor", ".editorconfig", ".env", ".env.example", ".gitattributes", ".gitignore", ".artisan", "composer.json", "composer.lock", and "package-lock.json". The main editor area displays the content of "PostController.php".

```
1 <?php
2
3 namespace App\Http\Controllers;
4
5 use Illuminate\Http\Request;
6
7 class PostController extends Controller
8 {
9     /**
10      * Display a listing of the resource.
11      */
12     public function index()
13     {
14         //
15     }
16
17     /**
18      * Show the form for creating a new resource.
19      */
20     public function create()
21     {
22         //
23     }
24 }
```

Cont. ...

Example: Defining a Resource Route

php

 Copy code

```
Route::resource('posts', PostController::class);
```

- This single line generates all the necessary routes for the above CRUD actions (index, show, store, update, etc.).

Example: Generated Routes

bash

 Copy code

GET	/posts	→ index
GET	/posts/create	→ create
POST	/posts	→ store
GET	/posts/{id}	→ show
GET	/posts/{id}/edit	→ edit
PUT/PATCH	/posts/{id}	→ update
DELETE	/posts/{id}	→ destroy

Cont. ...

4. Single Action Controllers

In cases where you need a controller with only a single action, Laravel allows the creation of **invokable controllers**. These controllers use the `__invoke()` method and can be defined with a single method.

Example: Creating an Invokable Controller

bash

 Copy code

```
php artisan make:controller ContactController --invokable
```

- This creates a controller with just one method `__invoke()`:

Cont. ...

- This creates a controller with just one method `__invoke()`:

```
php Copy code  
  
<?php  
  
namespace App\Http\Controllers;  
  
use Illuminate\Http\Request;  
  
class ContactController extends Controller  
{  
    public function __invoke()  
    {  
        return view('contact');  
    }  
}
```

Cont. ...

- Define the route like this:

php

 Copy code

```
Route::get('/contact', ContactController::class);
```

- Here, the controller is directly mapped to a route without specifying a method.

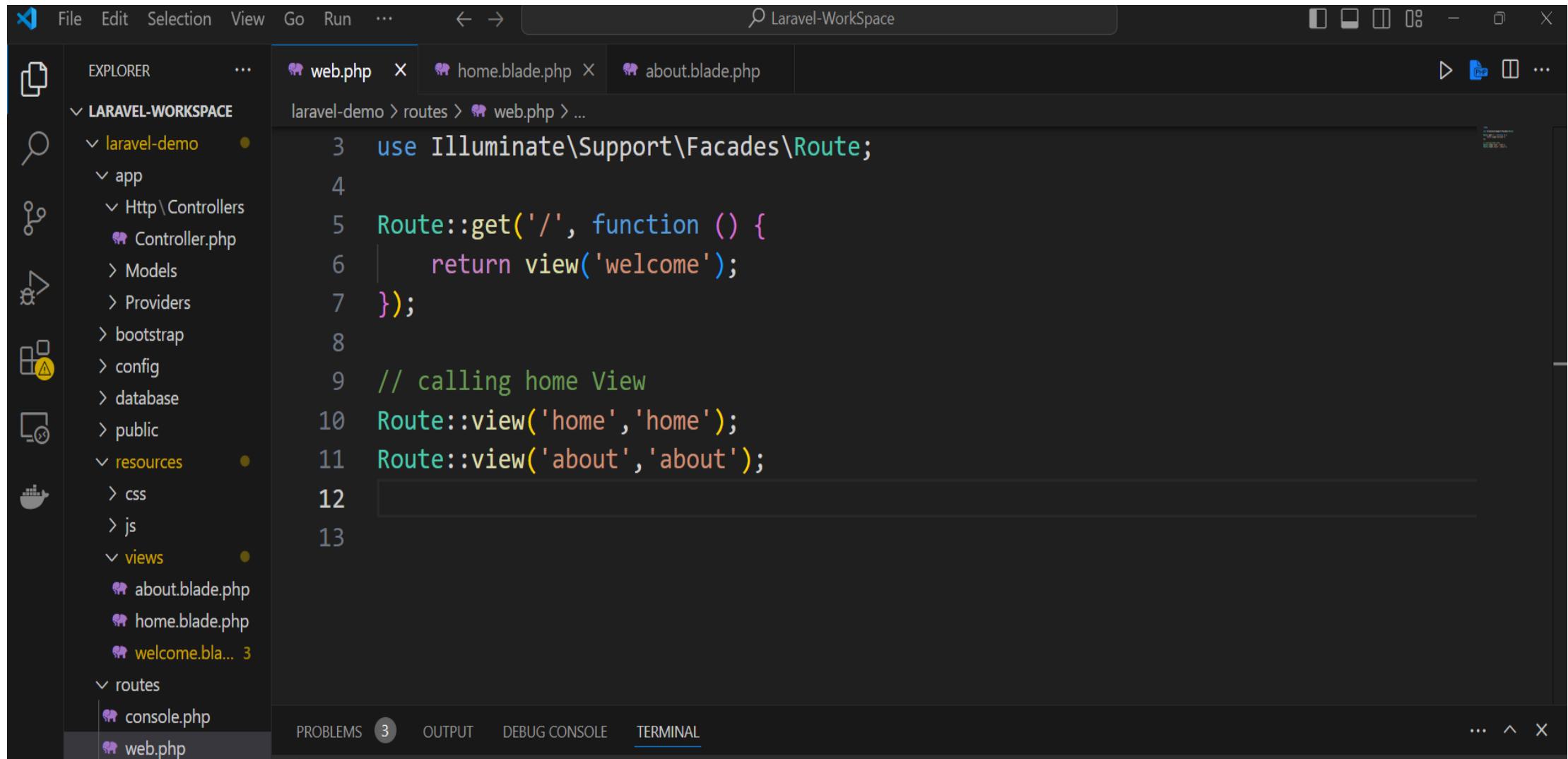


Laravel Views

View in Laravel

- In Laravel, views are used to separate the **application logic** from the **presentation layer**. They allow developers to create HTML output in organized templates, making it easier to build dynamic content.
- Views are stored as **.blade.php** files in the **resources/views** directory, where Blade templating enables efficient layouts, reusability, and simplified syntax.
- Here's how to work with views in Laravel:

Cont. ...



The screenshot shows the Visual Studio Code interface with a Laravel project workspace titled "Laravel-WorkSpace". The left sidebar displays the file structure of the "laravel-demo" application, including the "routes" directory which contains "web.php". The main editor tab is "web.php", showing the following PHP code:

```
use Illuminate\Support\Facades\Route;

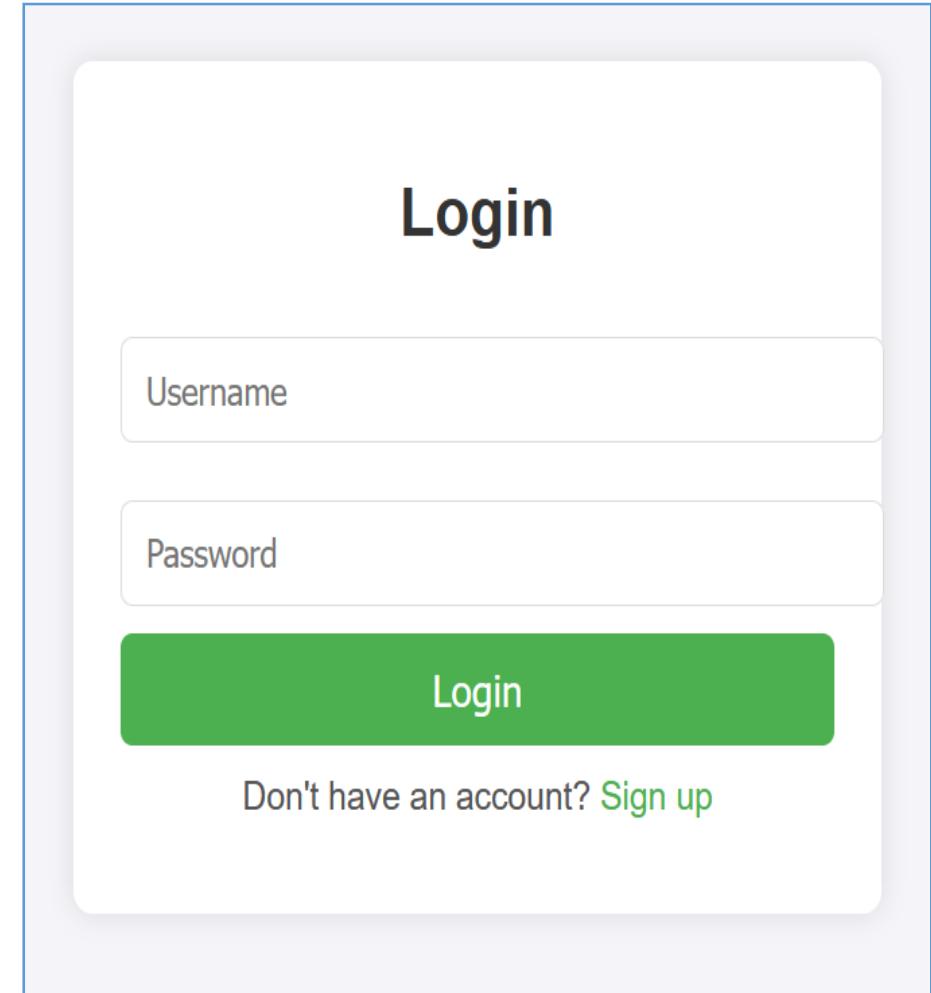
Route::get('/', function () {
    return view('welcome');
});

// calling home View
Route::view('home','home');
Route::view('about','about');
```

The code defines a route for the root path ('/') that returns the 'welcome' view. It also defines two routes using the `Route::view` method to call the 'home' and 'about' views respectively.

Cont. ...

```
web.php X home.blade.php X about.blade.php  
laravel-demo > routes > web.php > ...  
4  
5 Route::get('/', function () {  
6     return view('welcome');  
7 });  
8  
9 // calling home View  
10 Route::view('home', 'home');
```



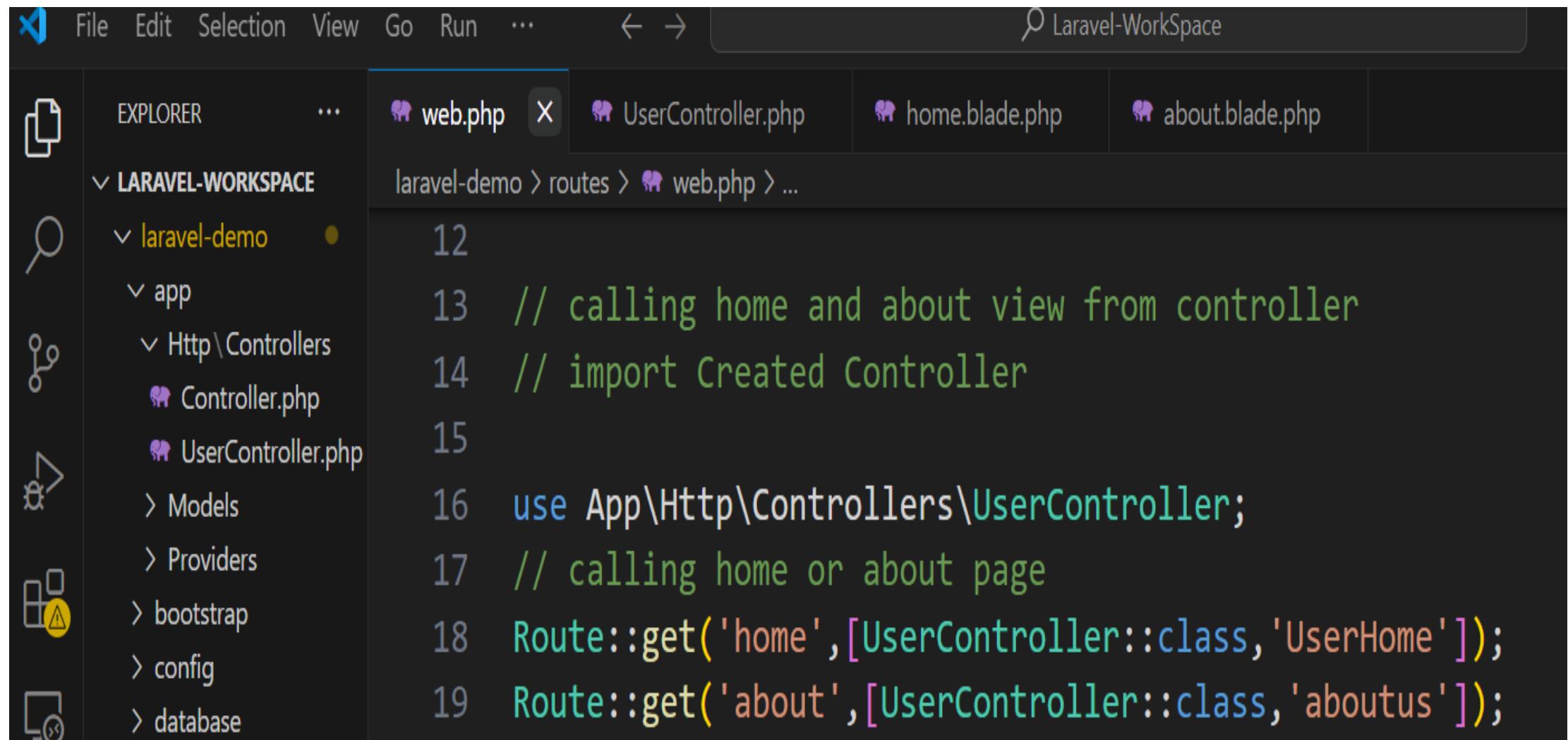
Calling View From Controller

The screenshot shows a Visual Studio Code interface with the title bar "Laravel-WorkSpace". The left sidebar displays the project structure under "EXPLORER" with the "UserController.php" file selected. The main editor area shows the code for the UserController.php file:

```
1 <?php
2
3 namespace App\Http\Controllers;
4
5 use Illuminate\Http\Request;
6
7 class UserController extends Controller
8 {
9     //
10    function UserHome(){
11        return view('home');
12    }
13
14    function aboutus(){
15        return view('about');
16    }
17}
18}
```

At the bottom of the code editor, there is a terminal window showing the command: `$ php artisan make:controller UserController`. Below the terminal, a message indicates the controller was created successfully.

Cont. ...



The screenshot shows a code editor interface with the title "Laravel-WorkSpace". The left sidebar contains icons for file operations like Explorer, Search, Open, and Issues. The Explorer section shows a project structure under "LARAVEL-WORKSPACE": "laravel-demo" (selected), which contains "app", "Http\Controllers", "Controller.php", "UserController.php", "Models", "Providers", "bootstrap", "config", and "database". The main editor area has tabs for "web.php", "UserController.php", "home.blade.php", and "about.blade.php". The "web.php" tab is active, displaying the following PHP code:

```
12
13 // calling home and about view from controller
14 // import Created Controller
15
16 use App\Http\Controllers\UserController;
17 // calling home or about page
18 Route::get('home',[UserController::class,'UserHome']);
19 Route::get('about',[UserController::class,'aboutus']);
```

Nesting View

The screenshot shows a code editor interface with a sidebar on the left displaying a file tree and a main panel on the right showing terminal output.

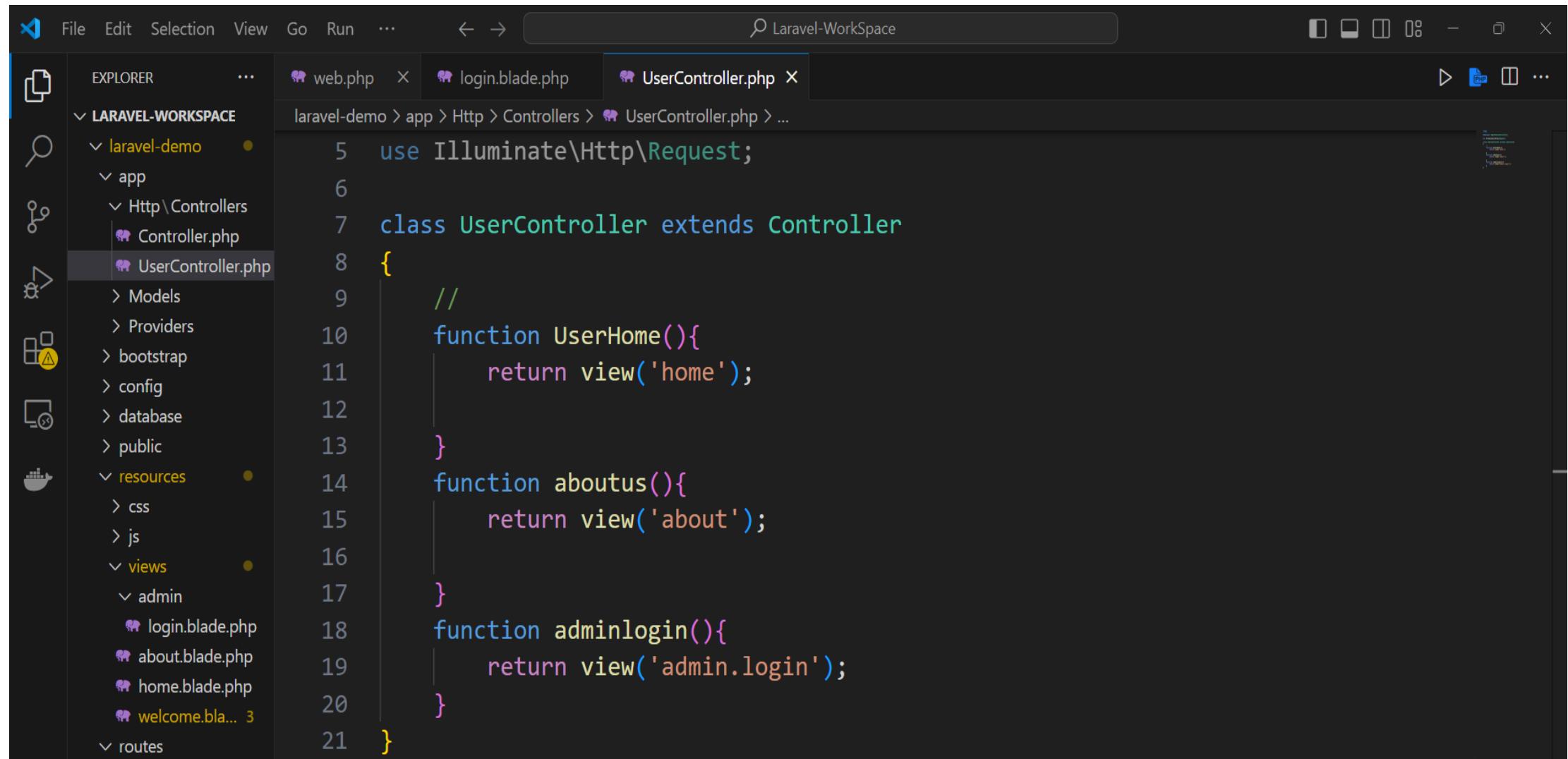
File Tree (Sidebar):

- views
 - admin
 - login.blade.php
 - about.blade.php
 - home.blade.php
 - welcome.blade.php
- routes
 - console.php
 - web.php

Terminal Output (Main Panel):

```
$ php artisan make:view admin.login
INFO View [C:\Users\yitayew\Desktop\Laravel-WorkSpace\laravel-demo\resources\views\admin\login.blade.php] created successfully.
```

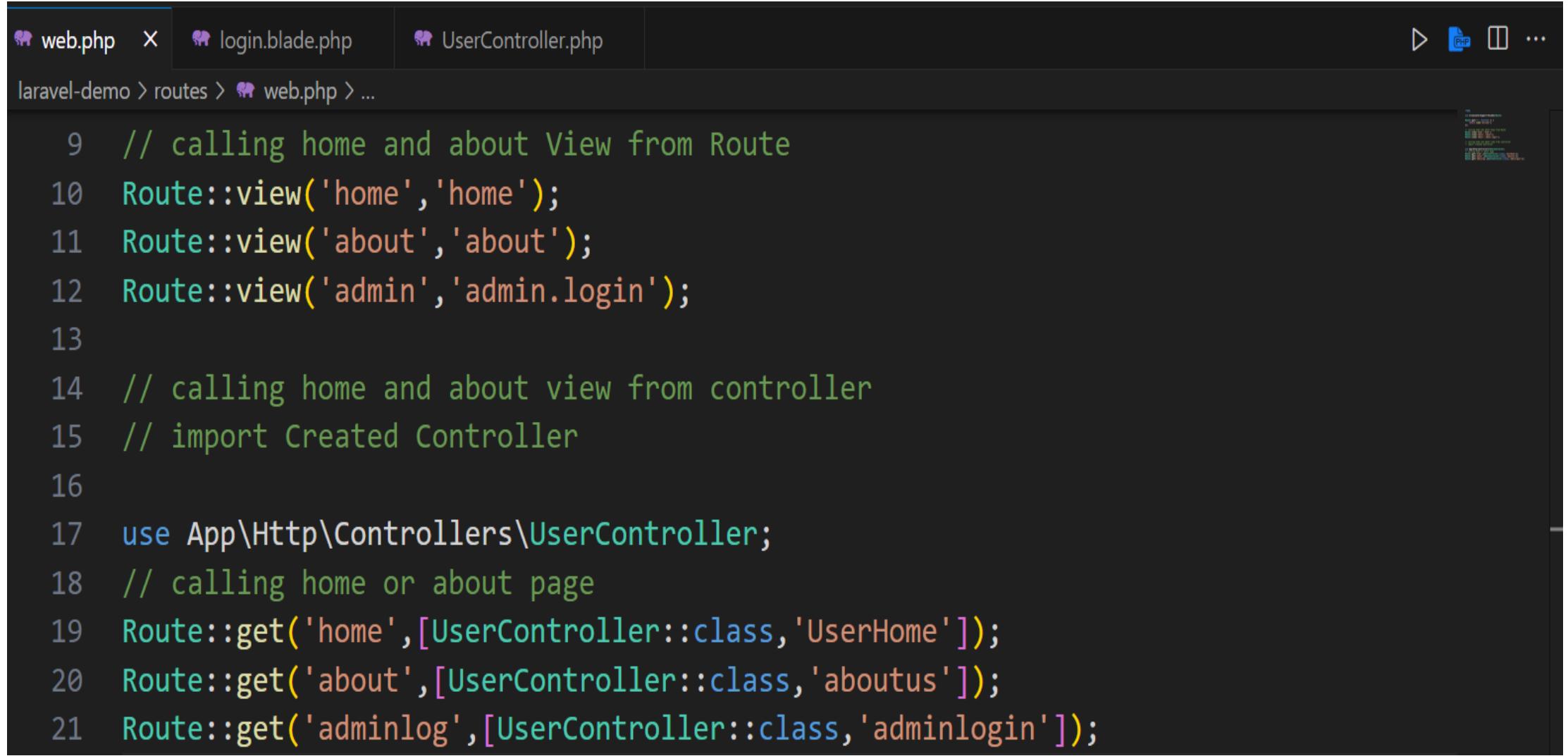
Cont. ...



The screenshot shows a code editor interface with a dark theme. On the left is the Explorer sidebar, which lists the project structure of a Laravel application named "laravel-demo". The "UserController.php" file is selected in the sidebar and is also open in the main editor area. The main editor area displays the following PHP code:

```
5  use Illuminate\Http\Request;
6
7  class UserController extends Controller
8  {
9      //
10     function UserHome(){
11         return view('home');
12     }
13
14     function aboutus(){
15         return view('about');
16     }
17
18     function adminlogin(){
19         return view('admin.login');
20     }
21 }
```

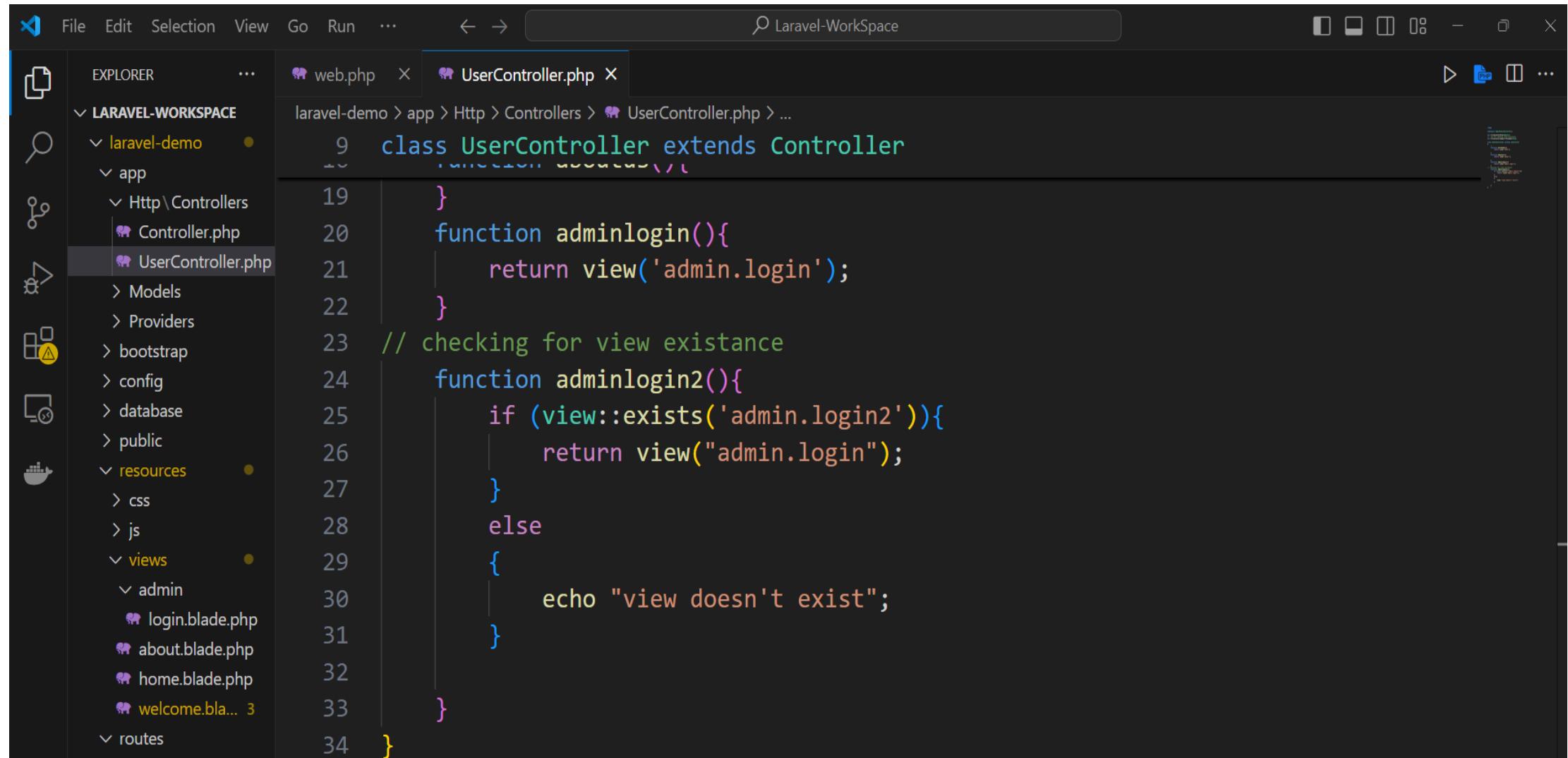
Cont. ...



The screenshot shows a code editor interface with a dark theme. At the top, there are three tabs: 'web.php' (selected), 'login.blade.php', and 'UserController.php'. Below the tabs, the file path 'laravel-demo > routes > web.php > ...' is displayed. The main area contains the following PHP code:

```
9 // calling home and about View from Route
10 Route::view('home','home');
11 Route::view('about','about');
12 Route::view('admin','admin.login');
13
14 // calling home and about view from controller
15 // import Created Controller
16
17 use App\Http\Controllers\UserController;
18 // calling home or about page
19 Route::get('home',[UserController::class,'UserHome']);
20 Route::get('about',[UserController::class,'aboutus']);
21 Route::get('adminlog',[UserController::class,'adminlogin']);
```

Checking view Existence



The screenshot shows a dark-themed interface of Visual Studio Code (VS Code) with an orange header bar. The title bar says "Laravel-WorkSpace". The left sidebar (Explorer) shows a file tree for a "laravel-demo" workspace, including "app", "Http\Controllers", "Controller.php", "UserController.php", "Models", "Providers", "bootstrap", "config", "database", "public", "resources", "css", "js", "views", "admin", "login.blade.php", "about.blade.php", "home.blade.php", "welcome.blade.php", and "routes". The "UserController.php" file is currently open in the main editor area. The code is as follows:

```
class UserController extends Controller
{
    public function adminlogin()
    {
        return view('admin.login');
    }

    // checking for view existence
    public function adminlogin2()
    {
        if (view::exists('admin.login2')){
            return view("admin.login");
        }
        else
        {
            echo "view doesn't exist";
        }
    }
}
```

Cont. ...

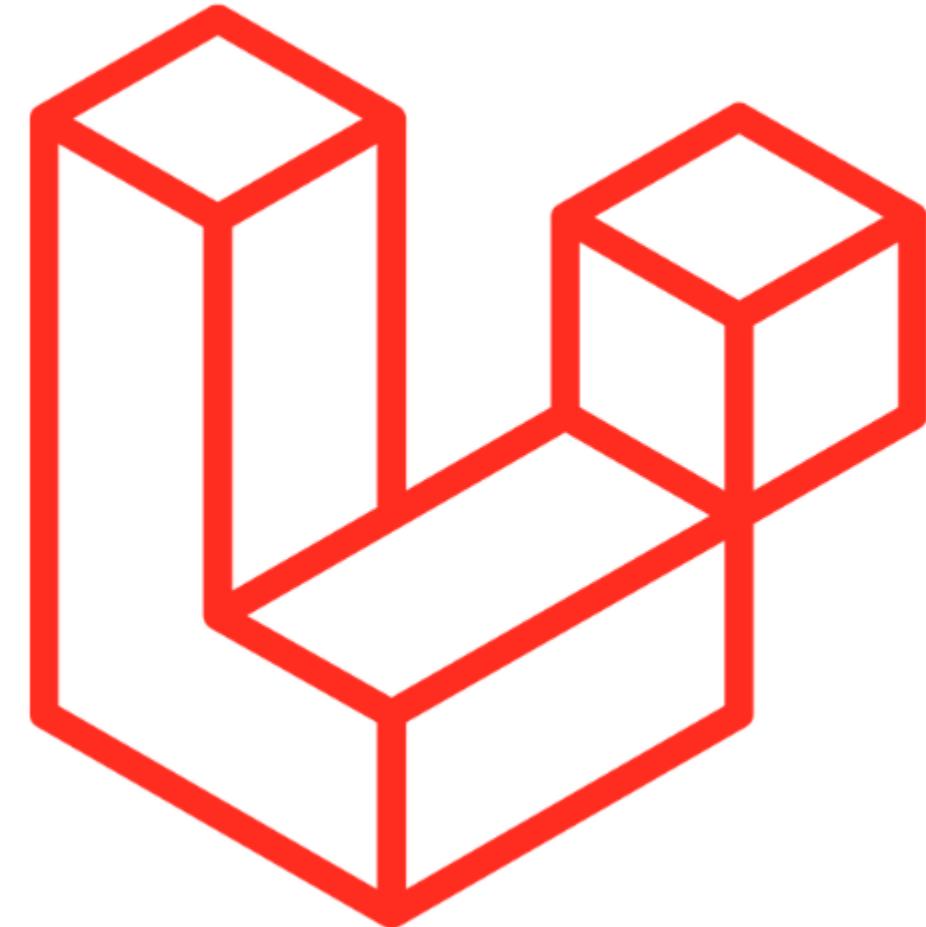
The screenshot shows a code editor and a browser window. The code editor has two tabs: 'web.php' and 'UserController.php'. The 'routes/web.php' file contains the following code:

```
8
9 // calling home and about View from Route
10 Route::view('home', 'home');
11 Route::view('about', 'about');
12 Route::view('admin', 'admin.login');
13
14 // calling home and about view from controller
15 // import Created Controller
16
17 use App\Http\Controllers\UserController;
18 // calling home or about page
19 Route::get('home',[UserController::class,'UserHome']);
20 Route::get('about',[UserController::class,'aboutus']);
21 Route::get('adminlog',[UserController::class,'adminlogin']);
22 Route::get('adminlog2',[UserController::class,'adminlogin2']);
```

The browser window shows a 404 error message: "view doesn't exist" at the URL `127.0.0.1:8000/adminlog2`.

LARAVEL

Blade Templates



Blade Template

- In Laravel, Blade is a simple yet powerful templating engine that provides an easy way to create **dynamic HTML pages**.
- It allows you to **structure your application's layout** and inject data from controllers, making it one of the key tools for Laravel developers.

Cont. ...

- Key Features of Blade:
- ✓ **Template Inheritance:** Enables layout reuse with sections for flexible content.
 - ✓ **Echoing Data:** Simple syntax for outputting variables and HTML.
 - ✓ **Conditional Statements and Loops:** Control structures for dynamic content.
 - ✓ **Components and Slots:** Reusable and customizable template fragments.

Example

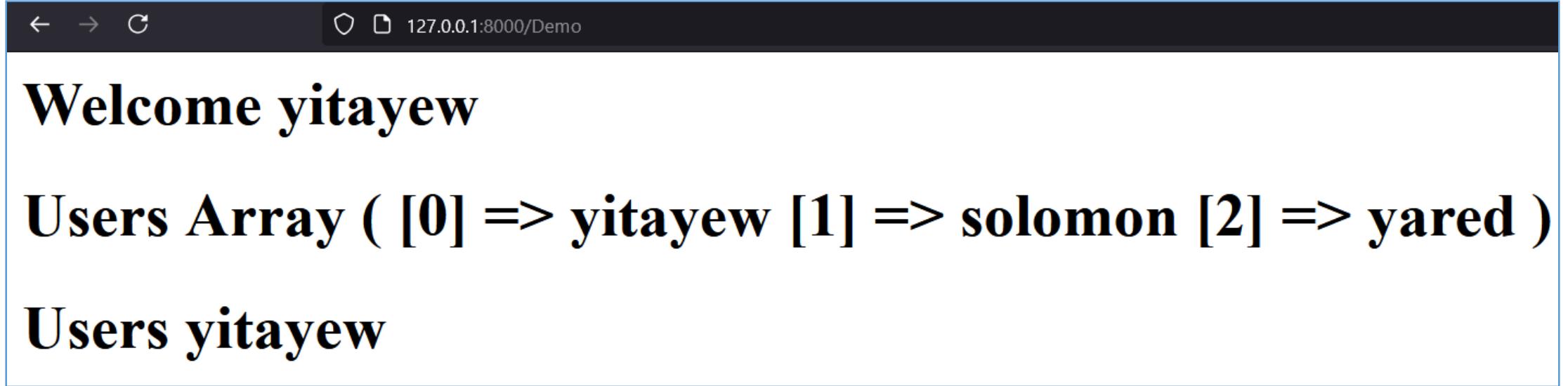
The screenshot shows a code editor interface with the title bar "Laravel-WorkSpace". The left sidebar is the "EXPLORER" view, showing the project structure:

- LARAVEL-WORKSPACE
- laravel-demo
 - app
 - Http\Controllers
 - Controller.php
 - UserController.php
 - Models
 - Providers
 - bootstrap
 - config
 - database
 - public
 - resources
 - css
 - js
 - views
 - admin
 - login.blade.php
 - about.blade.php
 - Demo.blade.php
 - home.blade.php
 - welcome.blade.php

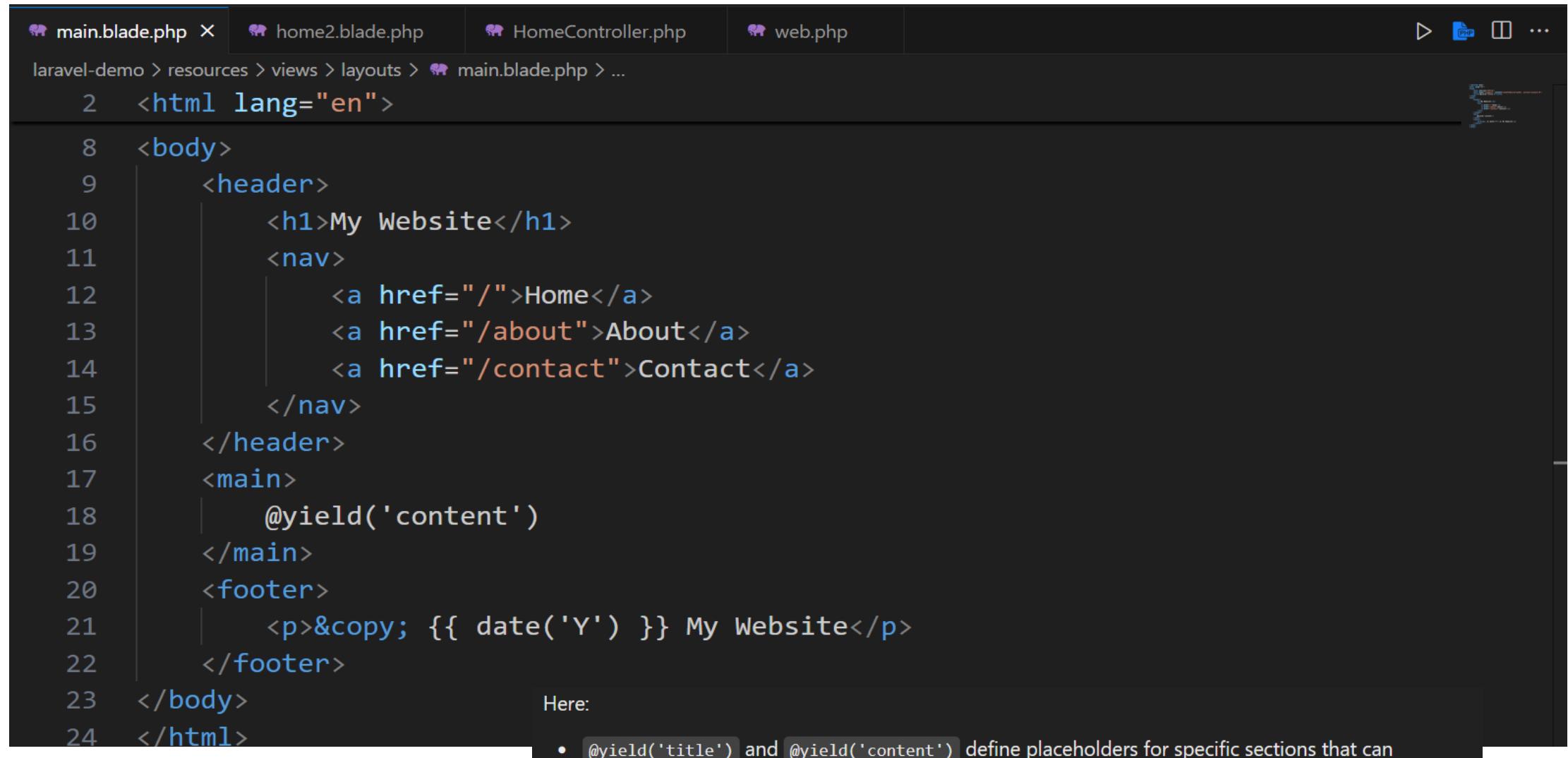
```
9  class UserController extends Controller
24      function adminlogin2(){
33          }
34      function demo(){
35          $Name = "yitayew";
36          $users = ['yitayew','solomon','yared'];
37          return view('Demo',[ "name"=>$Name,"family"=>$users]);
38      }
39 }
```

```
40
41      laravel-demo > resources > views > Demo.blade.php > ...
42      1  <h1>Welcome {{$name}}</h1>
43      2  <h1>Users {{print_r($family)}}</h1>
44      3  <h1>Users {{$family[0]}}</h1>
45      ^
```

Cont. ...



Example:



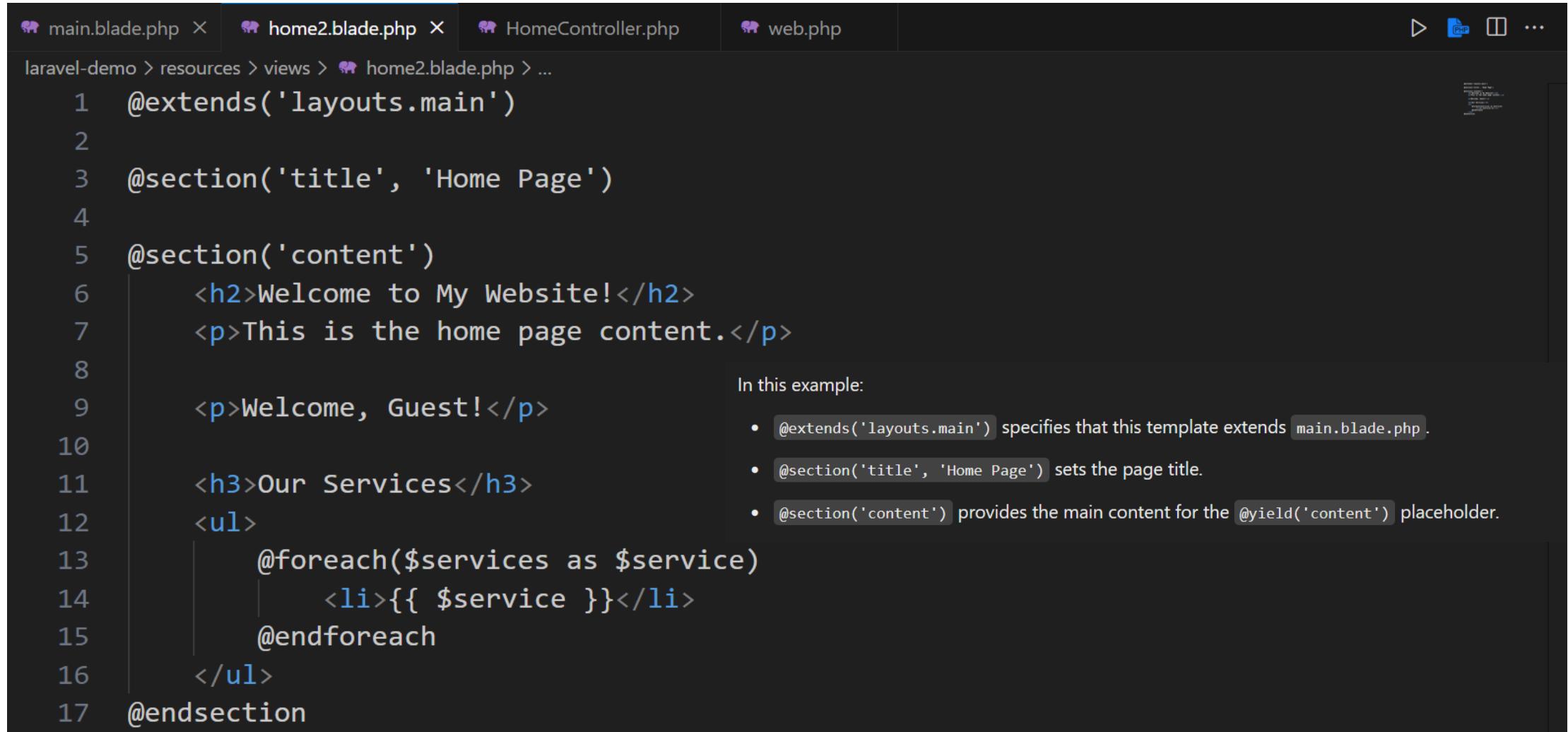
The screenshot shows a code editor with a dark theme. The file being edited is `main.blade.php`, which is a layout file for a Laravel application. The code includes sections for header, navigation, main content, and footer, with placeholder yield statements for title and content.

```
1<html lang="en">
2  <body>
3    <header>
4      <h1>My Website</h1>
5      <nav>
6        <a href="/">Home</a>
7        <a href="/about">About</a>
8        <a href="/contact">Contact</a>
9      </nav>
10     </header>
11     <main>
12       @yield('content')
13     </main>
14     <footer>
15       <p>&copy; {{ date('Y') }} My Website</p>
16     </footer>
17   </body>
18 </html>
```

Here:

- `@yield('title')` and `@yield('content')` define placeholders for specific sections that can be populated in child templates.

Cont. ...



The screenshot shows a code editor with several tabs at the top: 'main.blade.php', 'home2.blade.php', 'HomeController.php', and 'web.php'. The current file is 'home2.blade.php'. The code in the editor is:

```
1 @extends('layouts.main')
2
3 @section('title', 'Home Page')
4
5 @section('content')
6     <h2>Welcome to My Website!</h2>
7     <p>This is the home page content.</p>
8
9     <p>Welcome, Guest!</p>
10
11    <h3>Our Services</h3>
12    <ul>
13        @foreach($services as $service)
14            <li>{{ $service }}</li>
15        @endforeach
16    </ul>
17 @endsection
```

In this example:

- `@extends('layouts.main')` specifies that this template extends `main.blade.php`.
- `@section('title', 'Home Page')` sets the page title.
- `@section('content')` provides the main content for the `@yield('content')` placeholder.

Cont. ...



The screenshot shows a code editor interface with several tabs at the top: 'main.blade.php' (closed), 'home2.blade.php' (closed), 'HomeController.php' (active), and 'web.php' (closed). Below the tabs, a breadcrumb navigation bar indicates the file structure: 'laravel-demo > app > Http > Controllers > HomeController.php > ...'. The main content area displays the following PHP code:

```
1 <?php
2
3 namespace App\Http\Controllers;
4
5 use Illuminate\Http\Request;
6
7 class HomeController extends Controller
8 {
9     public function index()
10    {
11        $services = ['Web Development', 'SEO', 'Consulting'];
12
13        return view('home2', compact('services'));
14    }
15 }
```

Cont. ...

A screenshot of a web browser window displaying a local website at `127.0.0.1:8000/home2`. The page title is **My Website**. Below the title are three navigation links: Home, About, and Contact. The main content area features a large heading **Welcome to My Website!** followed by the text "This is the home page content." A greeting message "Welcome, Guest!" is also present. Under the heading **Our Services**, there is a bulleted list: • Web Development, • SEO, and • Consulting. At the bottom left, a copyright notice reads © 2024 My Website. A code snippet at the bottom right shows a Laravel route definition: `Route::get('home2', [HomeController::class, 'index']);`.

127.0.0.1:8000/home2

My Website

[Home](#) [About](#) [Contact](#)

Welcome to My Website!

This is the home page content.

Welcome, Guest!

Our Services

- Web Development
- SEO
- Consulting

© 2024 My Website

```
Route::get('home2', [HomeController::class, 'index']);
```

Blade Commands in Laravel 11

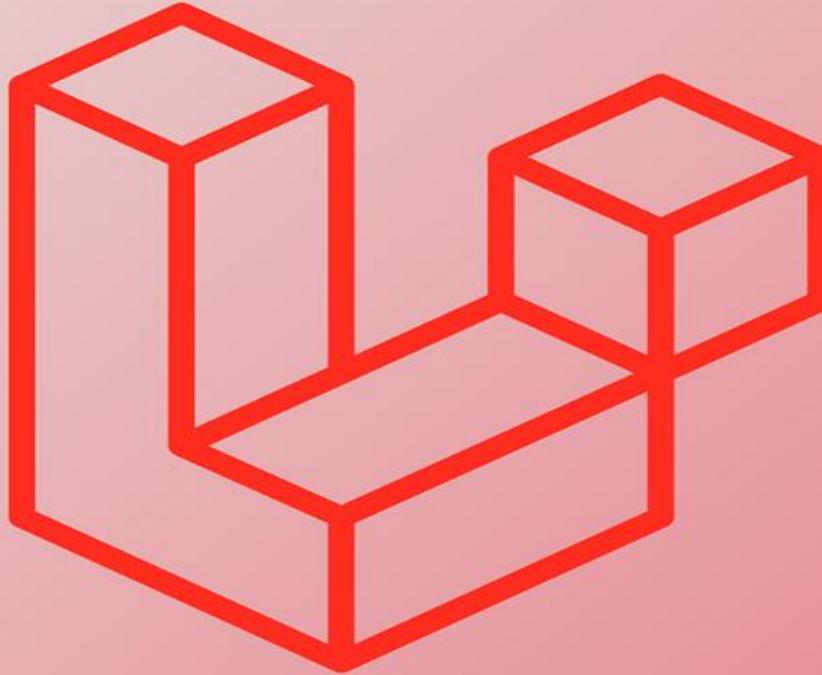
Category	Blade Command	Description/Example
Displaying Data	<code>{{ \$variable }}</code>	Echoes variable content with automatic HTML escaping.
	<code>{!! \$variable !!}</code>	Echoes variable content without HTML escaping.
Conditionals	<code>@if(condition) ... @endif</code>	Executes code if the condition is true.
	<code>@elseif(condition) / @else</code>	Adds additional conditions or an else block.
	<code>@unless(condition) ... @endunless</code>	Executes code if the condition is false.
Loops	<code>@for(init; condition; increment)</code>	Standard for-loop.
	<code>@foreach(\$items as \$item)</code>	Loops through a collection of items.
	<code>@forelse(\$items as \$item) ... @empty</code>	Loops through items; shows alternative content if empty.
	<code>@while(condition)</code>	Executes a while loop.

Cont. ...

Including Files	<code>@include('view')</code>	Includes a Blade file.
	<code>@includeIf('view')</code>	Includes a view only if it exists.
	<code>@includeWhen(condition, 'view')</code>	Includes a view based on a condition.
Layouts & Sections	<code>@extends('layout')</code>	Extends a layout file.
	<code>@section('name') ... @endsection</code>	Defines a section to inject content.
	<code>@yield('name')</code>	Displays the content of a section.
Components	<code><x-component-name /></code>	Renders a Blade component.

Cont. ...

Forms	<code>@csrf</code>	Adds a CSRF token field.
	<code>@method('PUT')</code>	Specifies an HTTP method (e.g., PUT, DELETE) in a form.
	<code>@error('field') ... @enderror</code>	Displays validation error messages for a specific field.
Authentication	<code>@auth ... @endauth</code>	Displays content if the user is authenticated.
	<code>@guest ... @endguest</code>	Displays content if the user is not authenticated.
Miscellaneous	<code>@php ... @endphp</code>	Executes raw PHP code inside Blade.
	<code>@dump(\$variable)</code>	Dumps a variable for debugging.
	<code>@switch(condition) ... @case</code>	Implements switch-case logic.



Components

Laravel 11

Components in laravel

- In Laravel, components are **reusable pieces of UI elements** designed to improve code organization, reusability, and modularity in Blade templates.
- They allow you to create custom HTML components that can be used throughout your application, making your views more **manageable** and **your code DRY** (Don't Repeat Yourself).

Cont. ...

- There are two main types of components in Laravel 11:
 - ❖ **Class-Based Components:** Components backed by a PHP class file, which allows you to add logic to the component.
 - ❖ **Anonymous Components:** Components without a backing class file, designed for simpler, static components.

Cont. ...

Creating a Component in Laravel 11

To create a component, Laravel offers an Artisan command:

bash

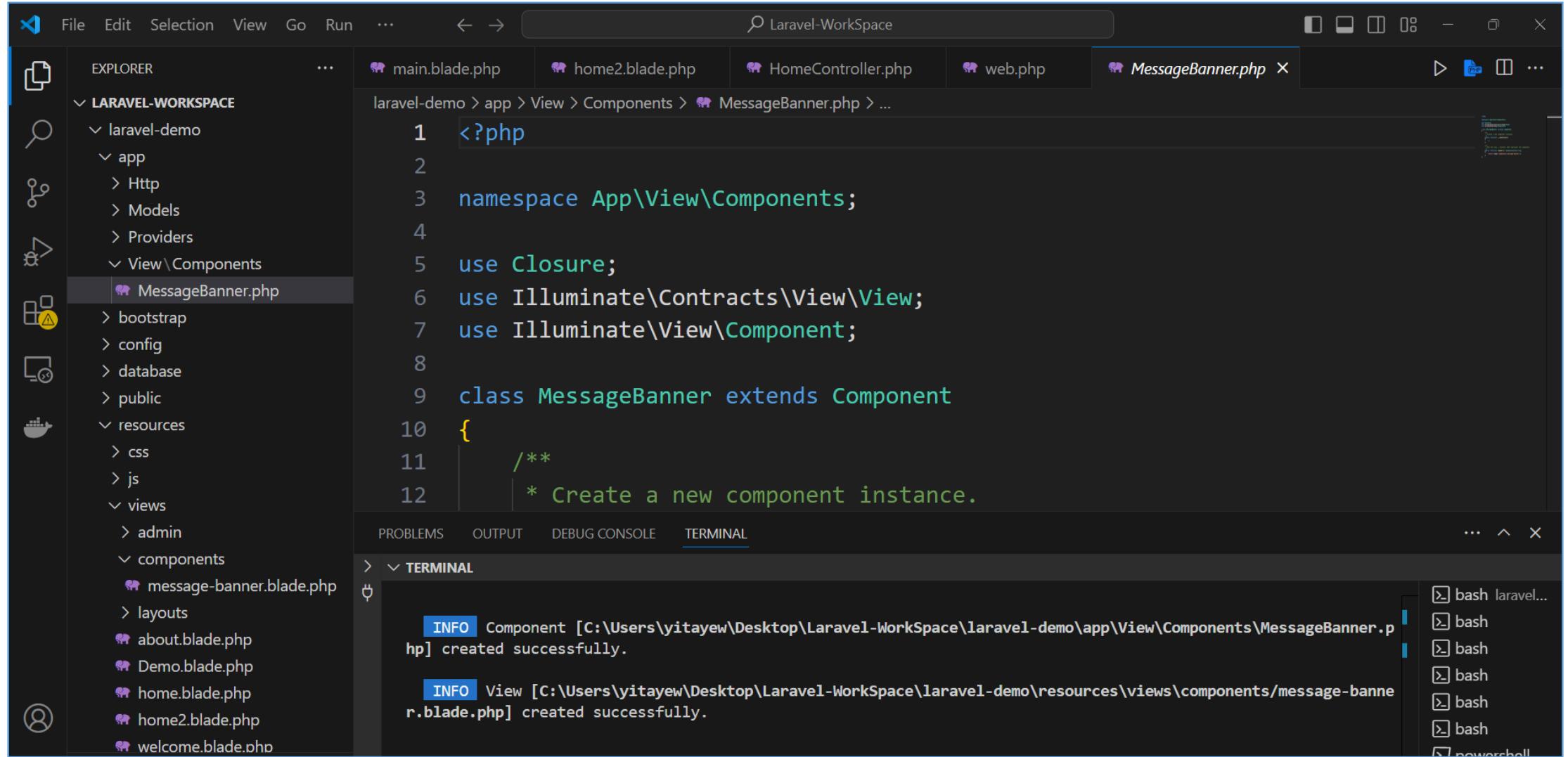
 Copy code

```
php artisan make:component Alert
```

This will create:

- A class file in `app/View/Components/Alert.php`.
- A Blade view file in `resources/views/components/alert.blade.php`.

Cont. ...



The screenshot shows a Visual Studio Code (VS Code) interface with the title bar "Laravel-WorkSpace". The left sidebar (EXPLORER) displays the project structure of "laravel-demo" under "LARAVEL-WORKSPACE". The "View\Components" folder contains several files: main.blade.php, home2.blade.php, HomeController.php, web.php, and MessageBanner.php (which is currently selected). The main editor area shows the code for "MessageBanner.php":

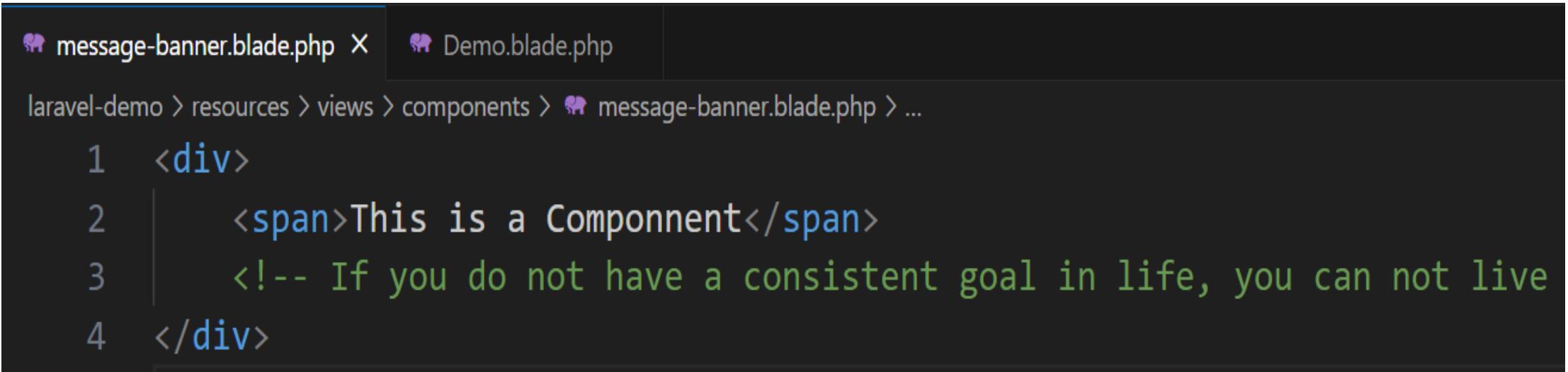
```
1 <?php
2
3 namespace App\View\Components;
4
5 use Closure;
6 use Illuminate\Contracts\View\View;
7 use Illuminate\View\Component;
8
9 class MessageBanner extends Component
10 {
11     /**
12      * Create a new component instance.
```

Below the editor, the "TERMINAL" tab is active, showing two informational messages:

```
INFO Component [C:\Users\yitayew\Desktop\Laravel-WorkSpace\laravel-demo\app\View\Components\MessageBanner.php] created successfully.
INFO View [C:\Users\yitayew\Desktop\Laravel-WorkSpace\laravel-demo\resources\views\components\message-banner.blade.php] created successfully.
```

On the right side, there is a sidebar with multiple terminal sessions, each labeled "bash" or "powershell".

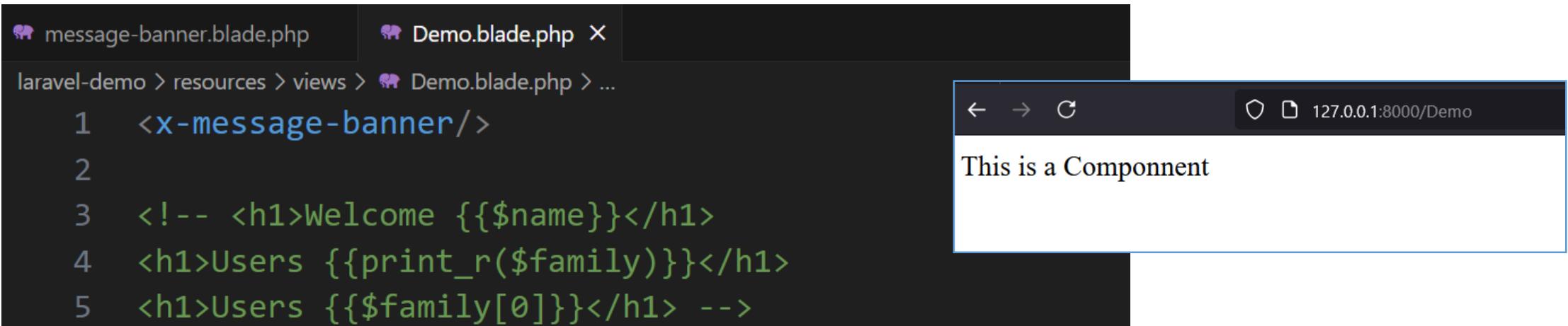
How to use the Component



The screenshot shows a code editor with two tabs open: "message-banner.blade.php" and "Demo.blade.php". The file "message-banner.blade.php" contains the following code:

```
1 <div>
2   <span>This is a Componnent</span>
3   <!-- If you do not have a consistent goal in life, you can not live -->
4 </div>
```

The file "Demo.blade.php" is currently active. The browser's address bar shows "laravel-demo > resources > views > components > message-banner.blade.php > ...". The browser window displays the output of the component: "This is a Componnent".

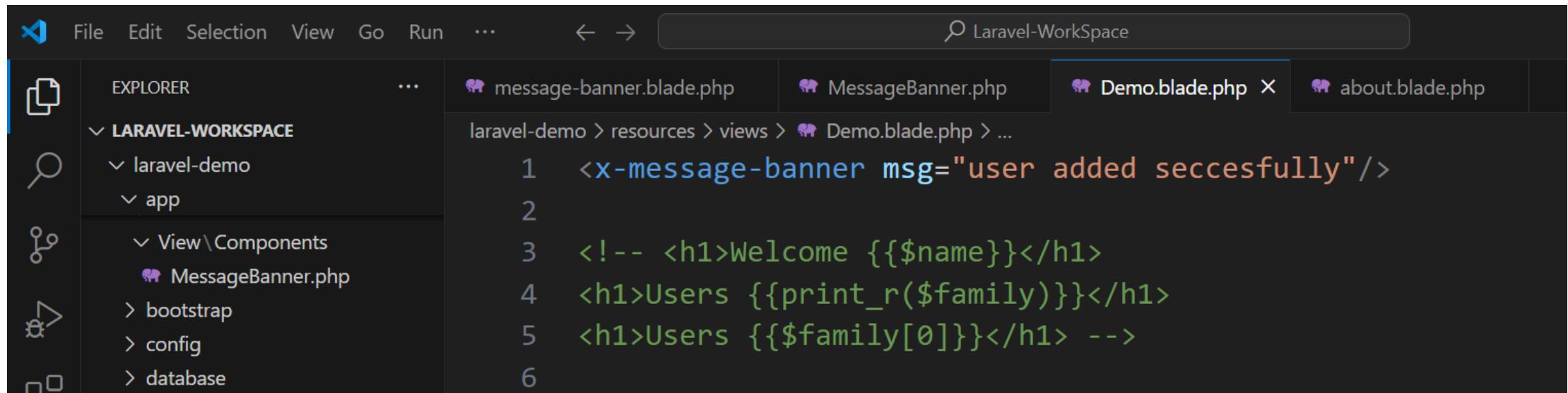


The screenshot shows a code editor with two tabs open: "message-banner.blade.php" and "Demo.blade.php". The file "message-banner.blade.php" contains the following code:

```
1 <x-message-banner/>
2
3 <!-- <h1>Welcome {{ $name }}</h1>
4 <h1>Users {{ print_r($family) }}</h1>
5 <h1>Users {{ $family[0] }}</h1> -->
```

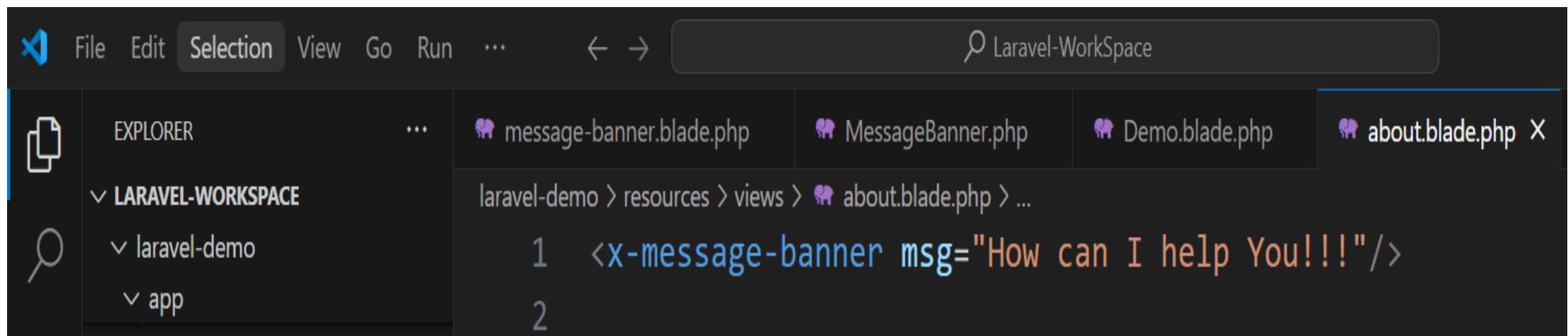
The file "Demo.blade.php" is currently active. The browser's address bar shows "laravel-demo > resources > views > Demo.blade.php > ...". The browser window displays the output of the component: "This is a Componnent".

Dynamic Component



The screenshot shows the VS Code interface with the title bar "Laravel-WorkSpace". The Explorer sidebar on the left shows a project structure under "LARAVEL-WORKSPACE": laravel-demo > resources > views > Demo.blade.php. The main editor area displays the following Blade template code:

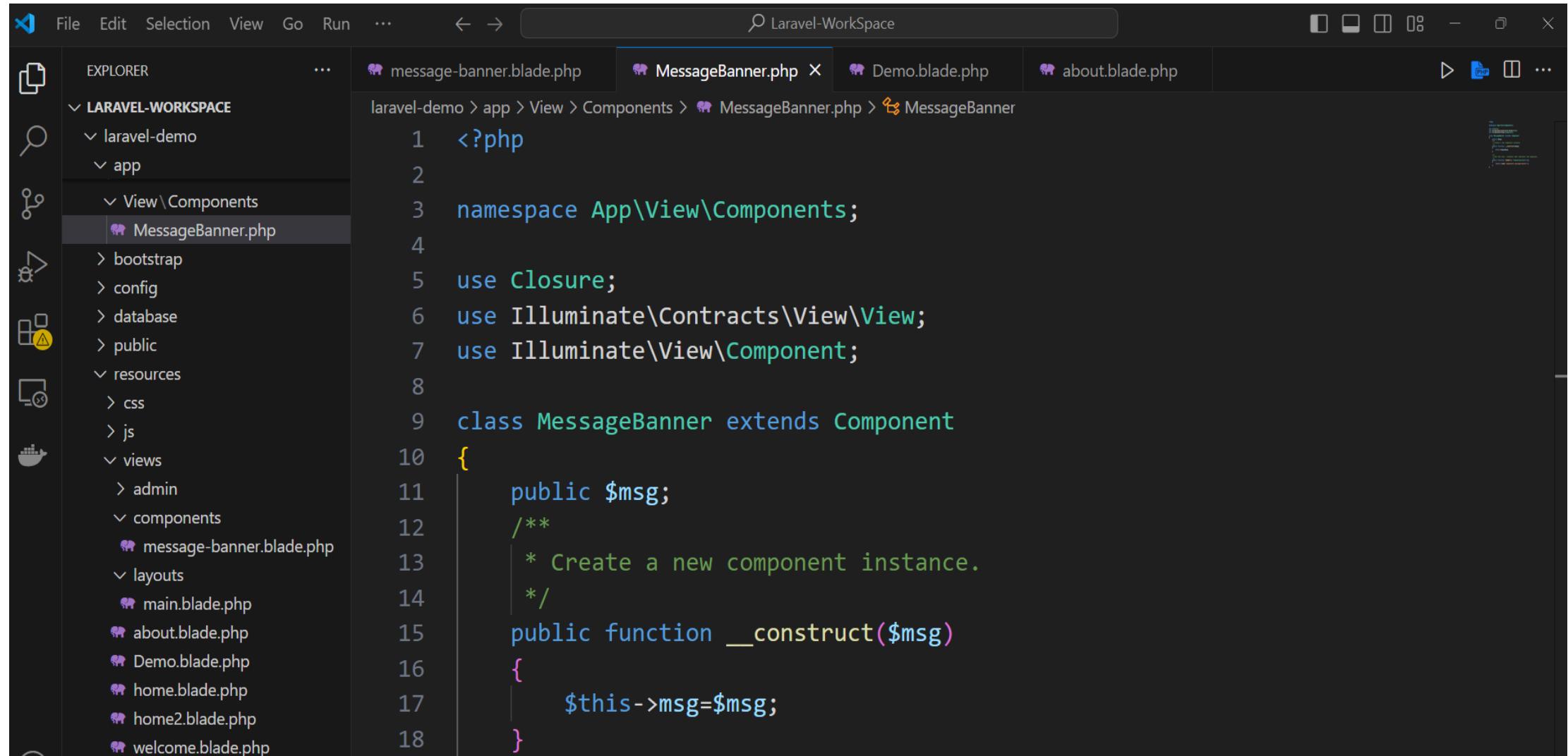
```
1 <x-message-banner msg="user added seccesfully"/>
2
3 <!-- <h1>Welcome {{$name}}</h1>
4 <h1>Users {{print_r($family)}}</h1>
5 <h1>Users {{$family[0]}}</h1> -->
6
```



The screenshot shows the VS Code interface with the title bar "Laravel-WorkSpace". The Explorer sidebar on the left shows a project structure under "LARAVEL-WORKSPACE": laravel-demo > resources > views > about.blade.php. The main editor area displays the following Blade template code:

```
1 <x-message-banner msg="How can I help You!!!"/>
2
```

Cont. ...



Laravel-WorkSpace

File Edit Selection View Go Run ... ← → 🔍 Laravel-WorkSpace

EXPLORER LARAVEL-WORKSPACE laravel-demo > app > View > Components > MessageBanner.php > MessageBanner

message-banner.blade.php MessageBanner.php Demo.blade.php about.blade.php

MessageBanner.php

```
1 <?php
2
3 namespace App\View\Components;
4
5 use Closure;
6 use Illuminate\Contracts\View\View;
7 use Illuminate\View\Component;
8
9 class MessageBanner extends Component
10 {
11     public $msg;
12     /**
13      * Create a new component instance.
14     */
15     public function __construct($msg)
16     {
17         $this->msg=$msg;
18     }
}
```

Cont. ...

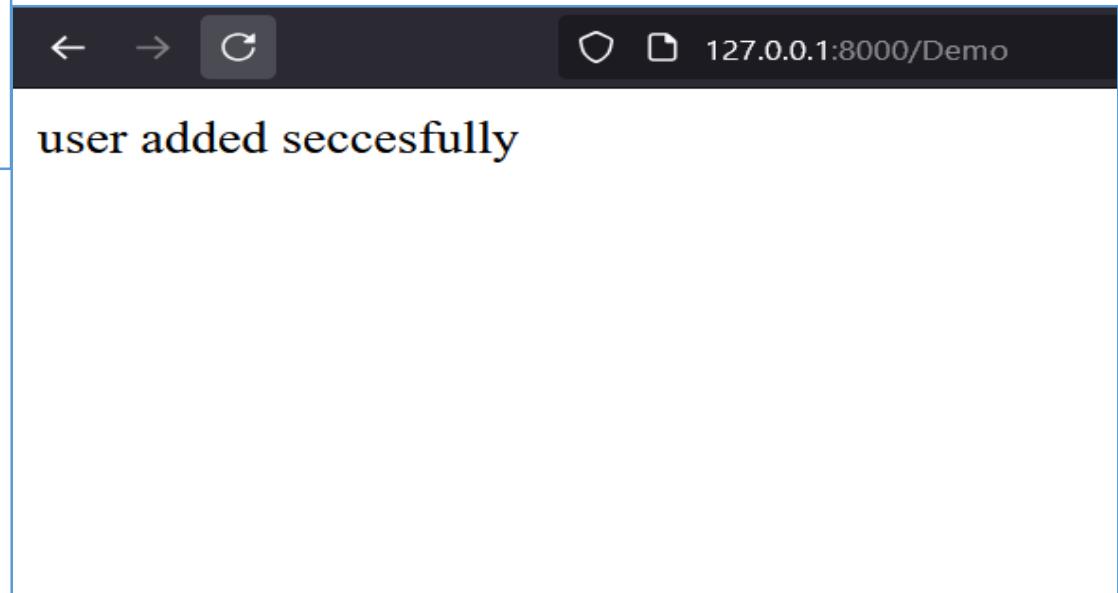
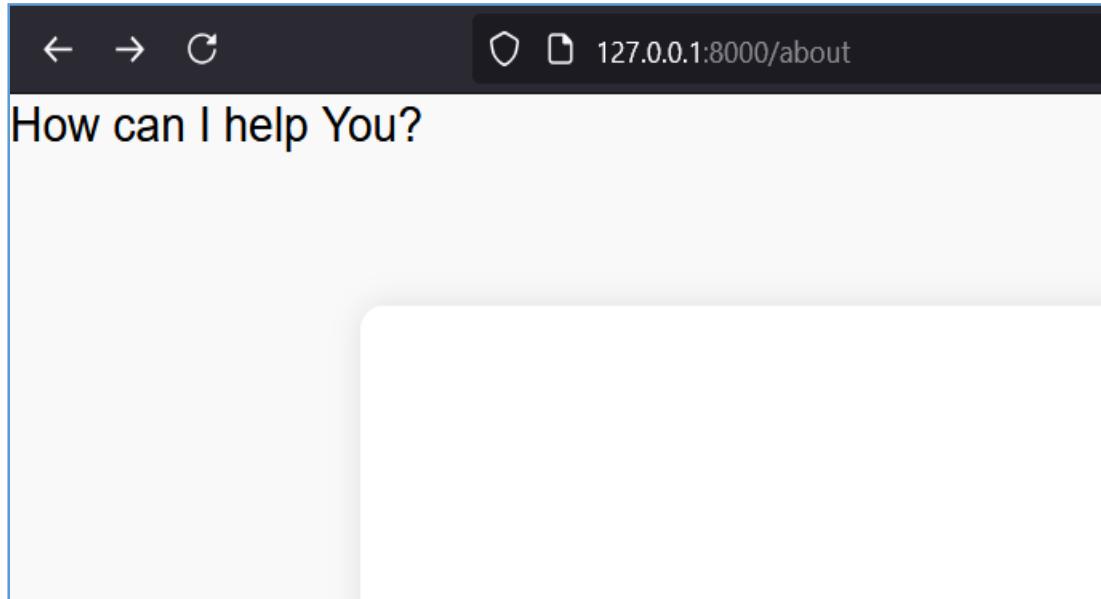
The screenshot shows a dark-themed interface of Visual Studio Code (VS Code) with an orange border. On the left is the Explorer sidebar, which lists the project structure under 'LARAVEL-WORKSPACE'. The 'resources/views/components' folder contains 'message-banner.blade.php', 'MessageBanner.php', 'Demo.blade.php', and 'about.blade.php'. The 'MessageBanner.php' file is currently selected in the editor tab bar. The code editor displays the following Blade template:

```
1 <div>
2 <span>{$msg}</span>
3 |   <!-- If you do not have a consistent goal in life, you
4 &lt;/div&gt;
5 </pre>

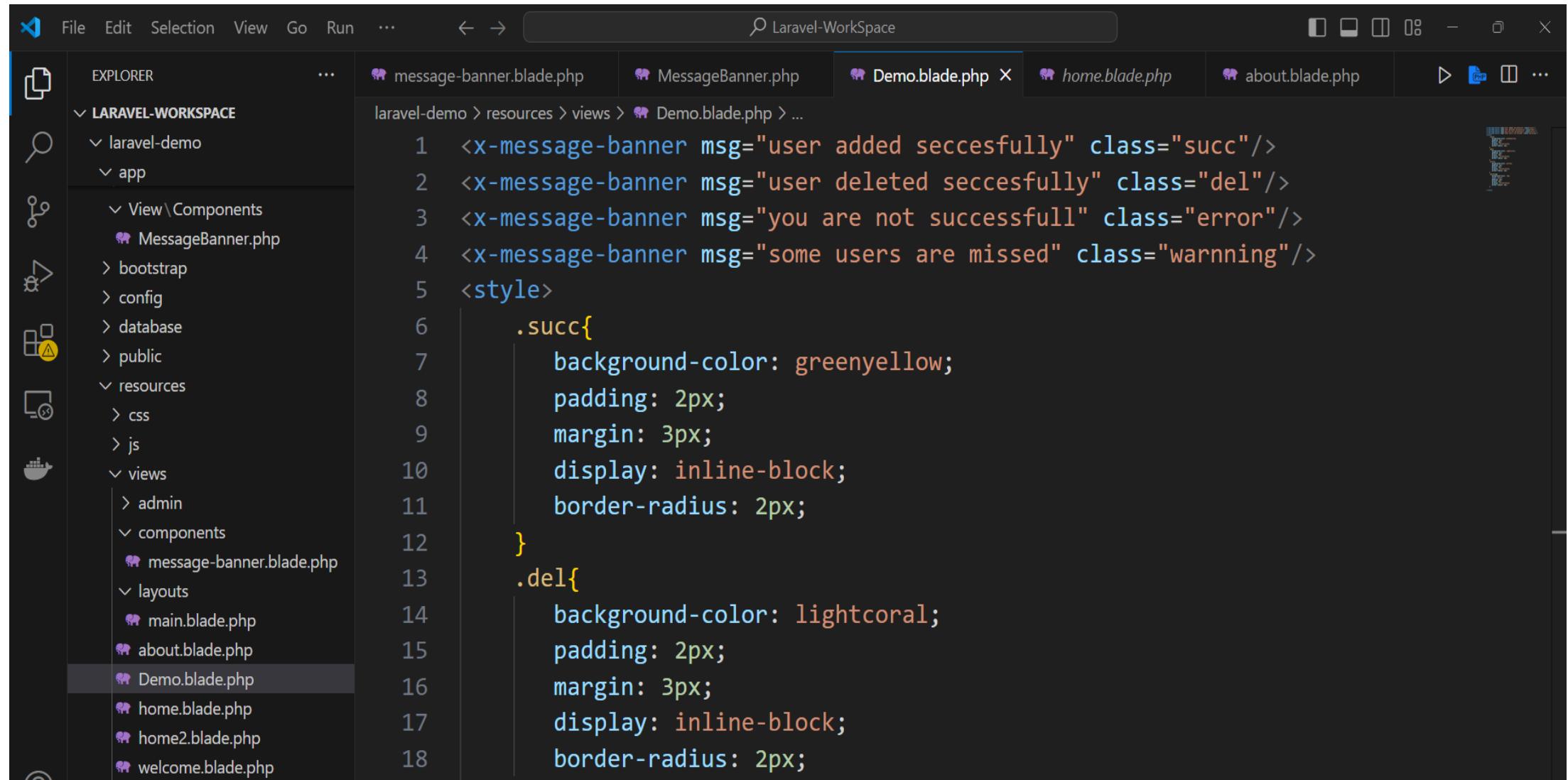
Below the editor are two browser preview panes. The left pane shows the 'about' page at 127.0.0.1:8000/about, displaying the placeholder text 'How can I help You?'. The right pane shows the 'Demo' page at 127.0.0.1:8000/Demo, displaying the message 'user added seccesfully'.


```

Cont. ...



Applying Common Style



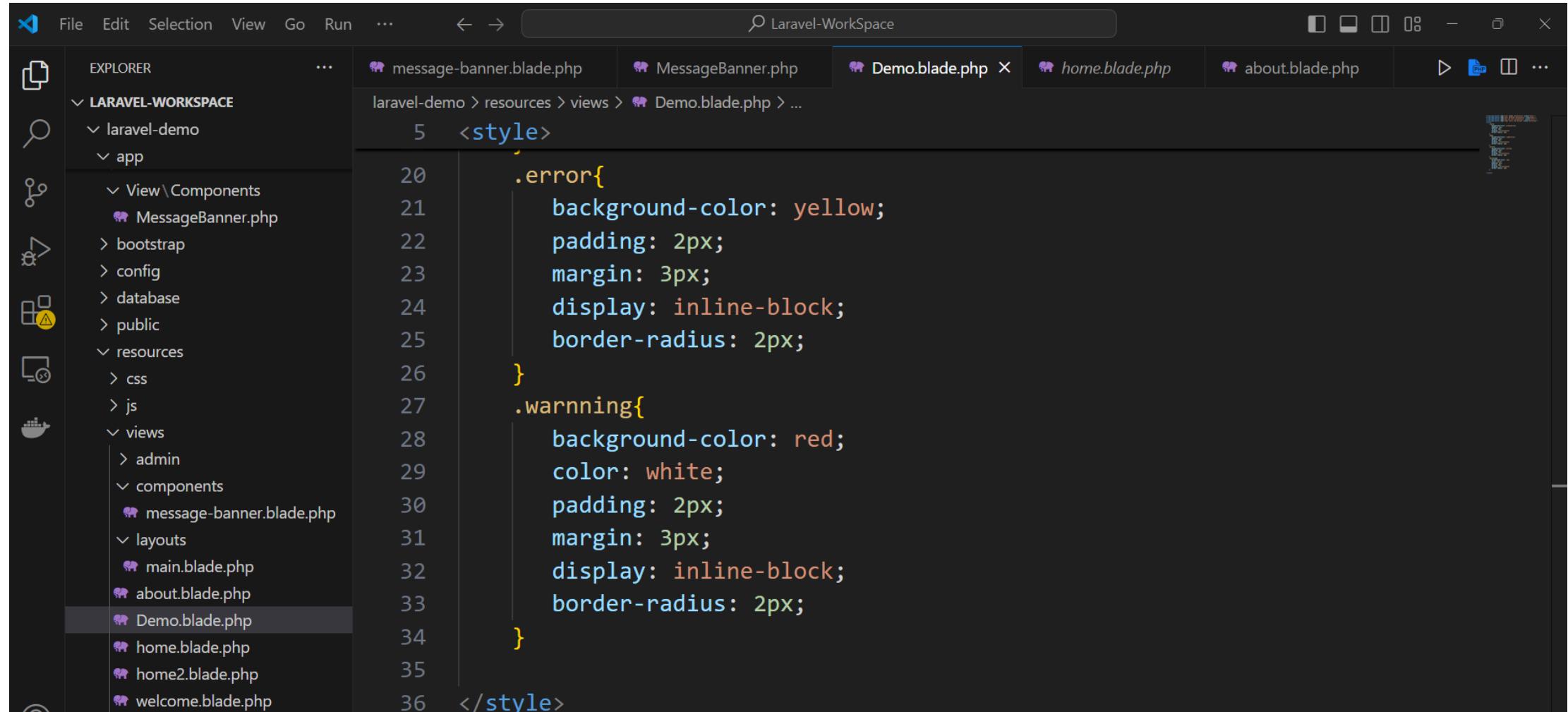
The screenshot shows a dark-themed interface of Visual Studio Code (VS Code) with the title bar "Laravel-WorkSpace". The left sidebar is the "EXPLORER" view, showing the file structure of a Laravel project named "laravel-demo". The "resources/views" folder contains several blade files: "message-banner.blade.php", "MessageBanner.php", "Demo.blade.php" (which is currently selected), "home.blade.php", "about.blade.php", and others. The main editor area displays the contents of "Demo.blade.php". The code uses inline CSS classes ("succ", "del") to apply styles to message banners based on their message content.

```
<x-message-banner msg="user added successfully" class="succ"/>
<x-message-banner msg="user deleted successfully" class="del"/>
<x-message-banner msg="you are not successful" class="error"/>
<x-message-banner msg="some users are missed" class="warning"/>

<style>
    .succ{
        background-color: greenyellow;
        padding: 2px;
        margin: 3px;
        display: inline-block;
        border-radius: 2px;
    }
    .del{
        background-color: lightcoral;
        padding: 2px;
        margin: 3px;
        display: inline-block;
        border-radius: 2px;
    }

```

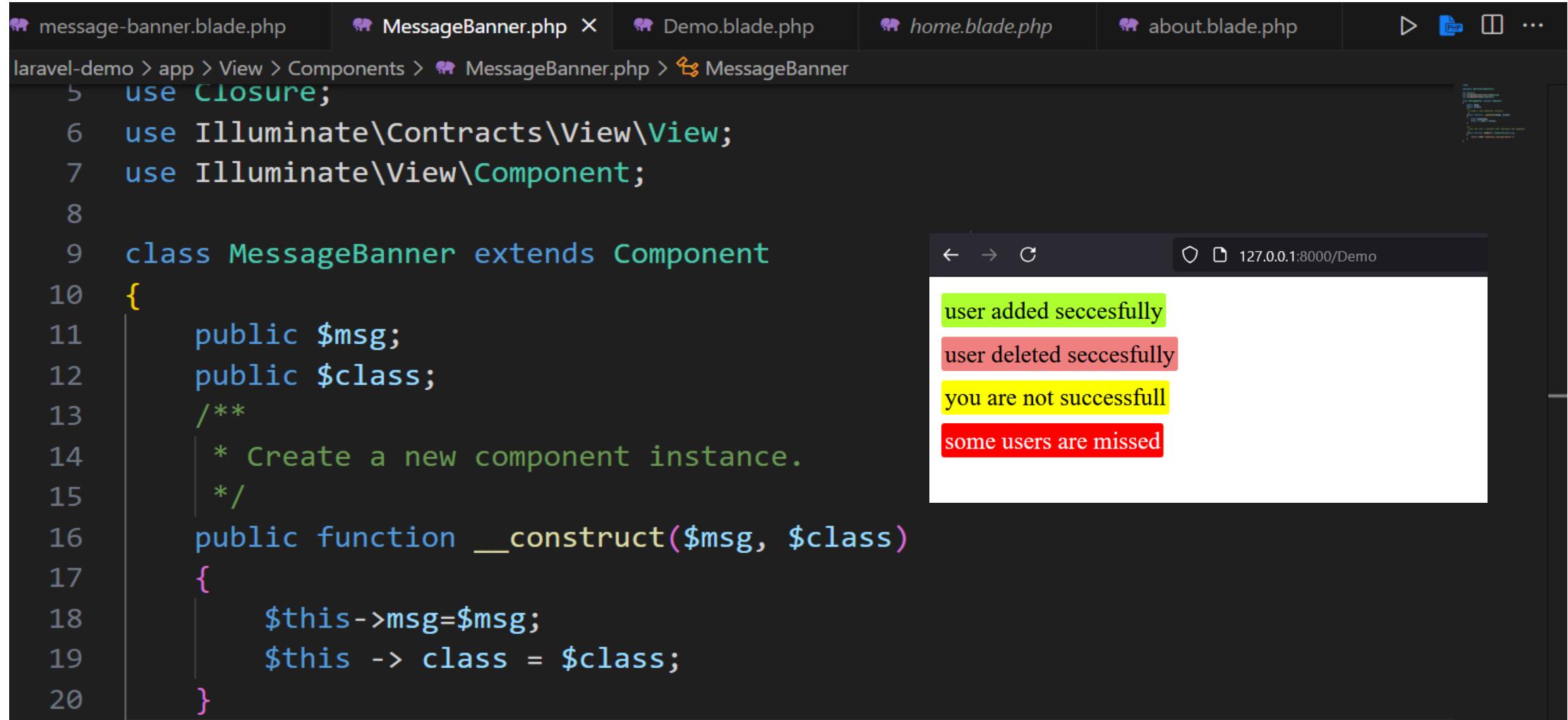
Cont. ...



The screenshot shows a code editor interface for a Laravel project named "laravel-demo". The "EXPLORER" sidebar on the left displays the project structure, including "LARAVEL-WORKSPACE", "laravel-demo", "app", "View\Components", "MessageBanner.php", "bootstrap", "config", "database", "public", "resources", "css", "js", "views", "admin", "components", "message-banner.blade.php", "layouts", "main.blade.php", "about.blade.php", "Demo.blade.php" (which is currently selected), "home.blade.php", "home2.blade.php", and "welcome.blade.php". The main editor area shows the contents of "Demo.blade.php".

```
5  <style>
20 .error{
21     background-color: yellow;
22     padding: 2px;
23     margin: 3px;
24     display: inline-block;
25     border-radius: 2px;
26 }
27 .warning{
28     background-color: red;
29     color: white;
30     padding: 2px;
31     margin: 3px;
32     display: inline-block;
33     border-radius: 2px;
34 }
35 </style>
```

Cont. ...



The image shows a code editor and a browser window side-by-side.

Code Editor (MessageBanner.php):

```
message-banner.blade.php MessageBanner.php X Demo.blade.php home.blade.php about.blade.php > PHP ...  
laravel-demo > app > View > Components > MessageBanner.php > MessageBanner  
5 use Closure;  
6 use Illuminate\Contracts\View\View;  
7 use Illuminate\View\Component;  
8  
9 class MessageBanner extends Component  
10 {  
11     public $msg;  
12     public $class;  
13     /**  
14      * Create a new component instance.  
15     */  
16     public function __construct($msg, $class)  
17     {  
18         $this->msg=$msg;  
19         $this -> class = $class;  
20     }  
}
```

Browser Screenshot:

127.0.0.1:8000/Demo

- user added seccesfully
- user deleted seccesfully
- you are not successfull
- some users are missed

Thank you!

Appreciate your action.