



Outlines of Discussion

- ❖ Authentication in Laravel
- ❖ Authorization in Laravel
- ❖ Roles and Permissions
- ❖ Roles and Permissions Example

Authentication In Laravel

- Laravel provides a powerful, out-of-the-box system for **Authentication** (identifying users) and **Authorization** (restricting or allowing access to certain resources). Below is a comprehensive, step-by-step explanation:

Authentication vs. Authorization

1. Understanding Authentication vs. Authorization

Authentication:

- It verifies the user's identity (e.g., login).
- Determines *who* the user is.

Authorization:

- It decides what actions or resources a user can access based on their roles or permissions.
- Determines *what* the user can do.

Setting Up Authentication

2. Setting Up Authentication

Step 1: Install Laravel

- Make sure you have Laravel installed.
- Use the command:

bash

 Copy code

```
composer create-project laravel/laravel example-app
cd example-app
```

Cont. ...

Step 2: Add Authentication Scaffolding

Laravel offers multiple ways to implement authentication scaffolding. Choose one based on your project requirements:

1. **Laravel Breeze** (lightweight):

bash

 Copy code

```
composer require laravel/breeze --dev  
php artisan breeze:install  
npm install && npm run dev  
php artisan migrate
```

Cont. ...

2. Laravel Jetstream (advanced features like 2FA):

bash

 Copy code

```
composer require laravel/jetstream
php artisan jetstream:install livewire
npm install && npm run dev
php artisan migrate
```

3. Laravel Fortify (backend-only):

bash

 Copy code

```
composer require laravel/fortify
php artisan vendor:publish --provider="Laravel\Fortify\FortifyServiceProvider"
php artisan migrate
```



Authentication In laravel (Creating New Project)

The screenshot shows a Microsoft Visual Studio Code (VS Code) interface with a dark theme. The left sidebar contains icons for file operations like Open, Save, Find, and Delete. The Explorer sidebar on the left lists the project structure:

- File
- Edit
- Selection
- View
- Go
- Run
- Terminal
- Help

Search bar: permission

Explorer sidebar:

- EXPLORER
- PERMISSION (selected)
- app
- bootstrap
- config
- database
- public
- resources
- routes (selected)
- console.php
- web.php (selected)
- storage
- tests
- vendor
- .editorconfig
- .env
- .env.example
- .gitattributes
- .gitignore
- artisan
- composer.json
- composer.lock
- package.json
- phpunit.xml
- postcss.config.js
- README.md
- tailwind.config.js
- vite.config.js

Code editor:

```
routes > web.php > ...
1 <?php
2
3 use Illuminate\Support\Facades\Route;
4
5 Route::get('/', function () {
6     return view('welcome');
7 });
8
```

Bottom navigation tabs: PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL (selected)

Terminal:

```
yitayew@DESKTOP-NPGGF04 MINGW64 ~/Desktop/Laravel-Workspace/permission
$ ls
app/    bootstrap/   composer.lock  database/   phpunit.xml  public/   resources/  storage/   tests/   vite.config.js
artisan*  composer.json config/      package.json postcss.config.js README.md routes/   tailwind.config.js vendor/
yitayew@DESKTOP-NPGGF04 MINGW64 ~/Desktop/Laravel-Workspace/permission
$ php artisan serve
forking is not supported on this platform

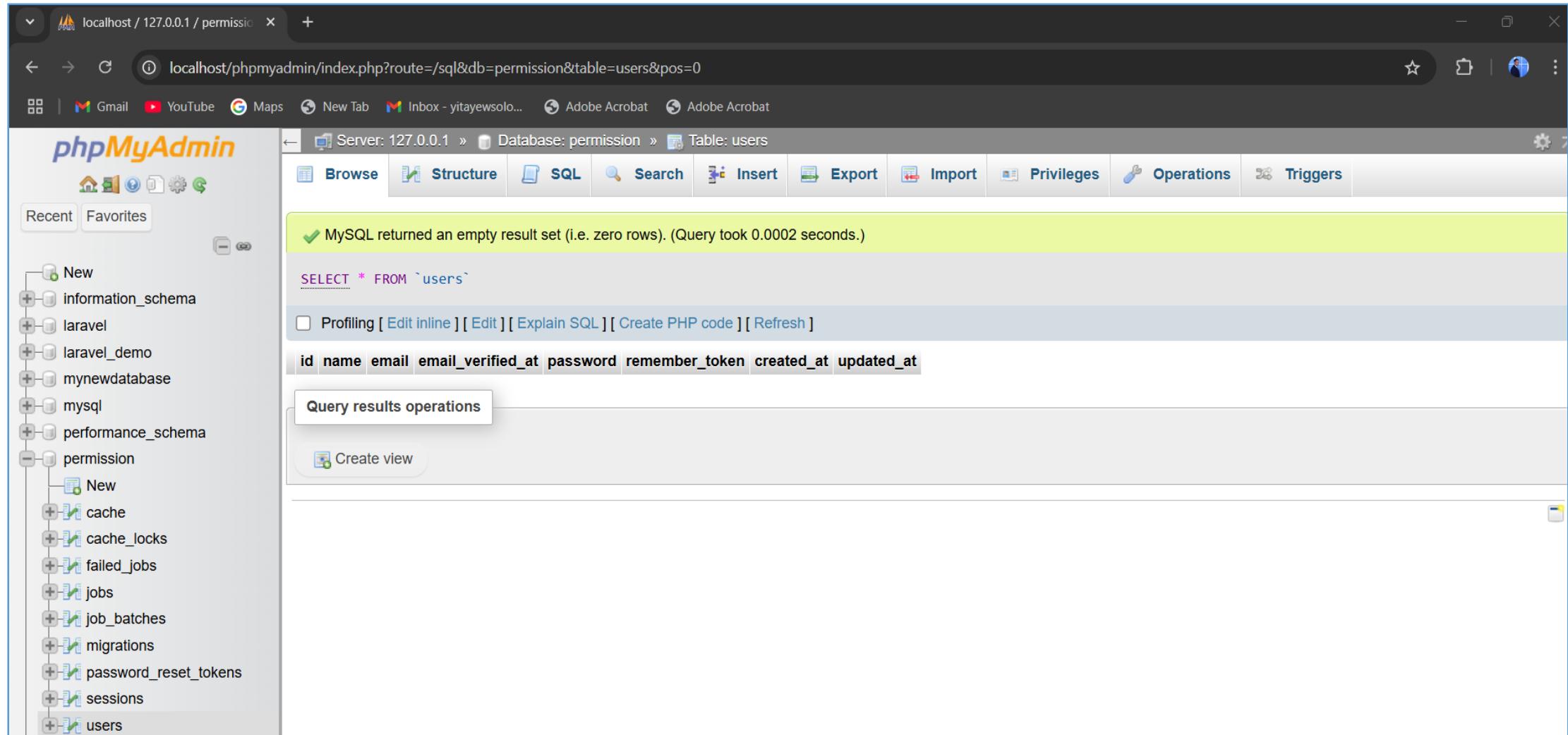
INFO Server running on [http://127.0.0.1:8000].
```

Terminal dropdown menu:

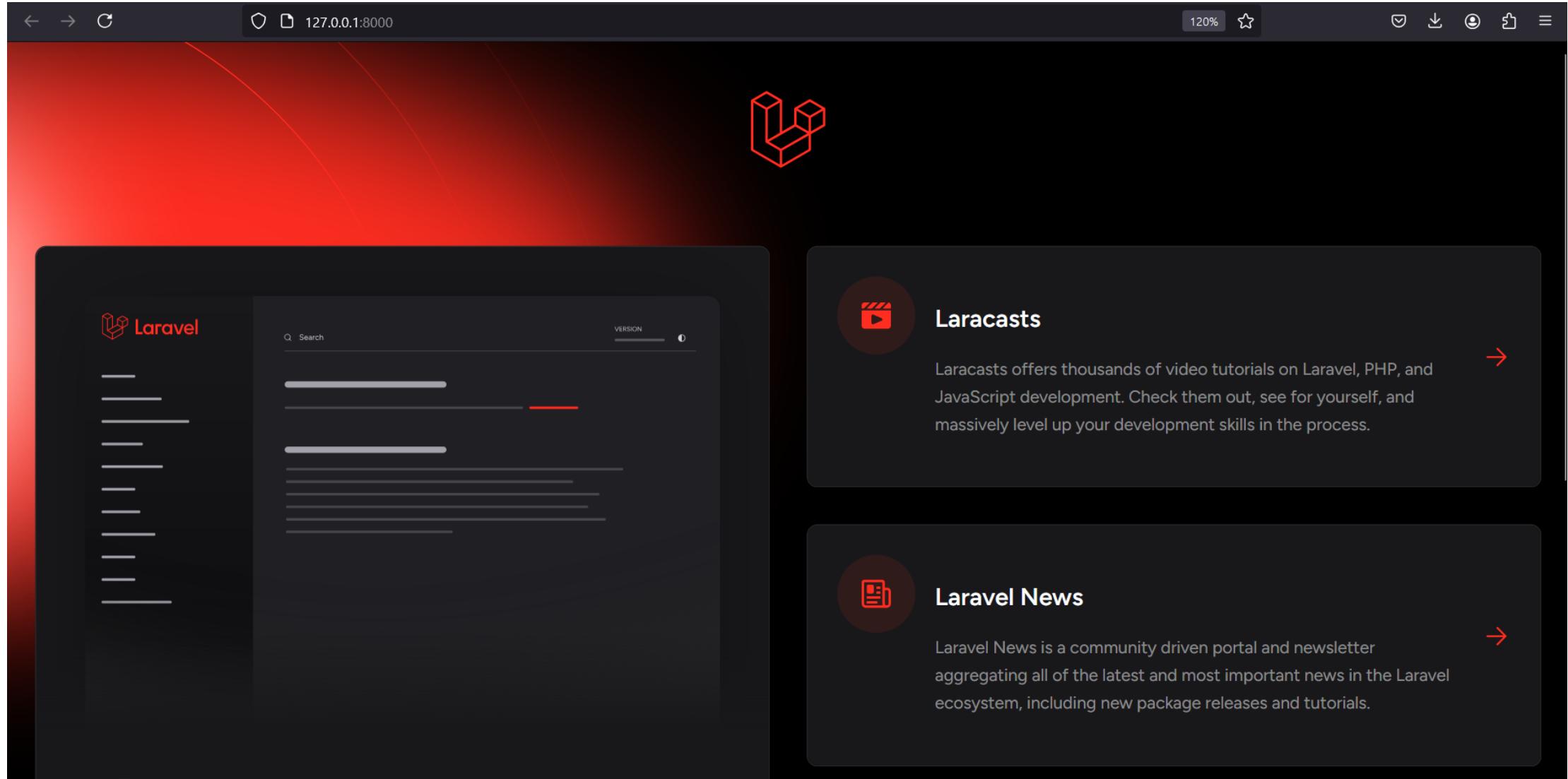
- powershell
- bash

Bottom status bar: Press Ctrl+C to stop the server

Connecting to Database



Configuring Laravel Breeze



Laravel Breeze

- Laravel Breeze is a simple, lightweight starter kit for Laravel that provides a basic authentication system. It's ideal for quickly scaffolding authentication functionality in your Laravel application, with minimal setup and clean implementation. Laravel Breeze comes with views built using Blade templates, along with an option for Inertia.js and Vue or React if you prefer modern frontend tools.

Features of Laravel Breeze

Features of Laravel Breeze

1. Authentication:

- User registration
- Login/logout
- Password reset
- Email verification

2. Frontend:

- Blade templates for server-side rendering.
- Optionally, Vue.js or React for client-side interactivity via Inertia.js.

3. Minimal dependencies to keep the implementation simple and lightweight.

4. Tailwind CSS integration for quick and beautiful styling.

Installing Breeze

Installing Laravel Breeze

Step 1: Install Laravel

Ensure you have a Laravel application installed:

```
bash
```

 Copy code

```
composer create-project laravel/laravel my-app  
cd my-app
```

Step 2: Require Breeze Package

Install the Laravel Breeze package:

```
bash
```

 Copy code

```
composer require laravel/breeze --dev
```

Cont. ...

Step 3: Install Breeze

Run the Breeze installation command:

bash

 Copy code

```
php artisan breeze:install
```

By default, Breeze installs with Blade templates. If you want to use a frontend framework like Vue or React, you can specify it:

- For Vue:

bash

 Copy code

```
php artisan breeze:install vue
```

Cont. ...

Step 4: Install Dependencies

After installing Breeze, install the frontend dependencies:

```
bash
```

 Copy code

```
npm install  
npm run dev
```

Step 5: Run Migrations

Breeze comes with the required migrations for users. Run the migrations to create the necessary database tables:

```
bash
```

 Copy code

```
php artisan migrate
```

Cont. ...

Using Laravel Breeze

1. Login and Registration:

- Breeze provides routes, controllers, and views for authentication:
 - `/register`
 - `/login`
 - `/forgot-password`
 - `/reset-password`
 - `/email/verify`

Cont. ...

2. Route Protection:

Use the `auth` middleware to protect routes:

php

 Copy code

```
Route::get('/dashboard', function () {
    return view('dashboard');
})->middleware(['auth'])->name('dashboard');
```

3. Customization:

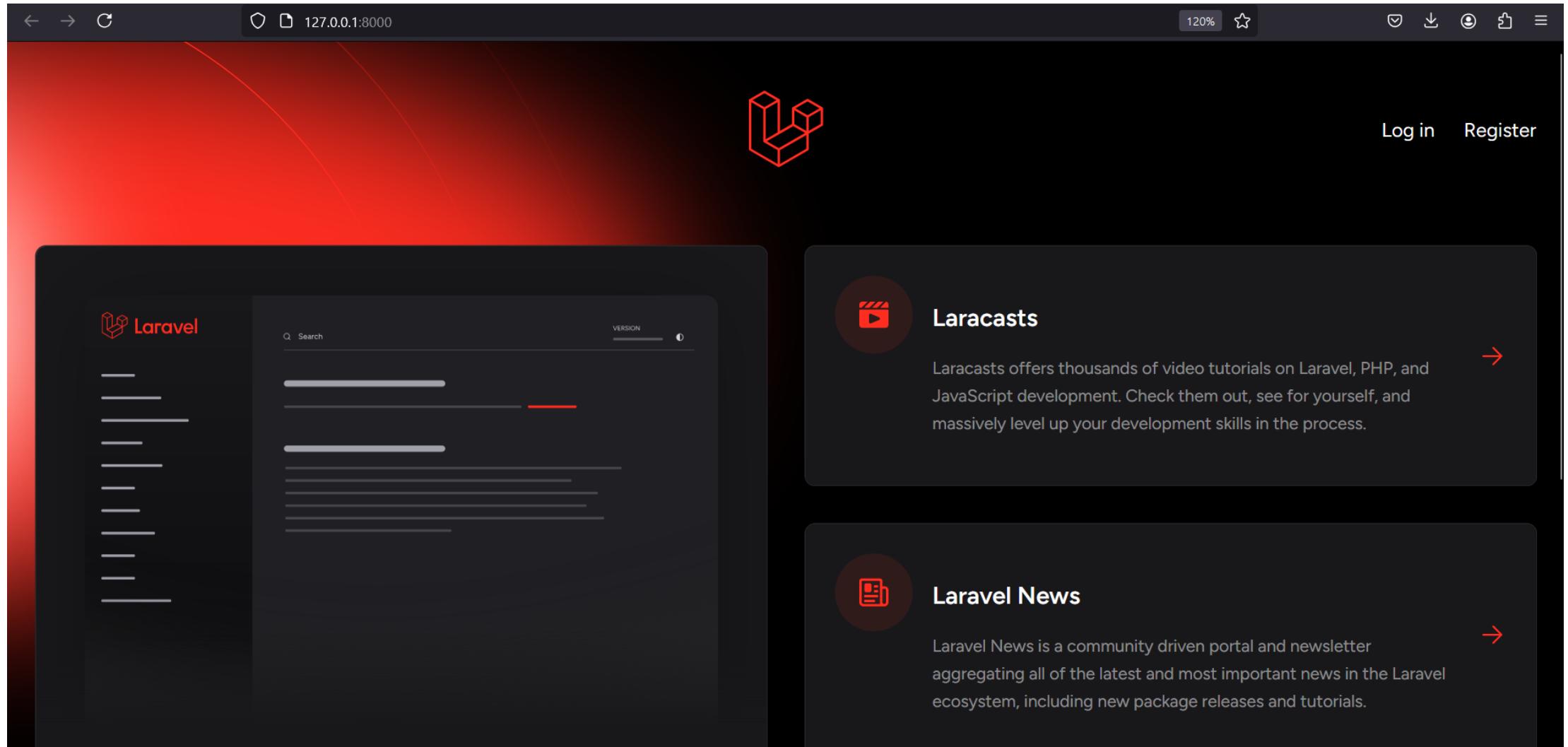
- Edit the authentication views in `resources/views/auth`.
- Modify the controller logic in `app/Http/Controllers/Auth`.

4. Frontend Customization:

- Breeze uses Tailwind CSS for styling, which you can extend by editing the `tailwind.config.js` file.



Output (After Breeze Installation)



Database Tables After Migration

The screenshot shows the phpMyAdmin interface with the following details:

- Server:** 127.0.0.1
- Database:** permission
- Table:** users

The interface includes the following sections:

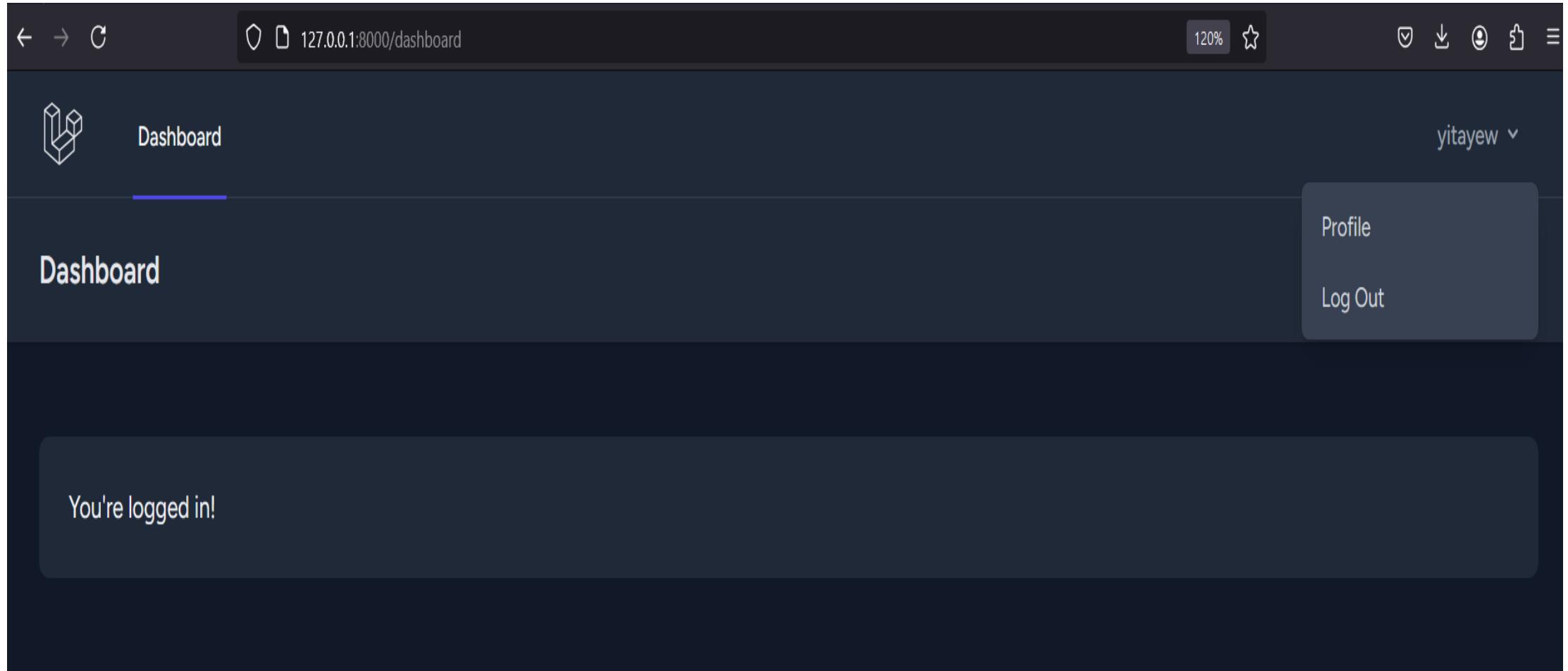
- Left sidebar:** Shows the database structure with tables: New, information_schema, laravel, laravel_demo, mynewdatabase, mysql, performance_schema, permission, and their respective sub-tables like cache, failed_jobs, etc.
- Top navigation:** Includes tabs for Browse, Structure, SQL, Search, Insert, Export, Import, Privileges, Operations, and Triggers.
- Message bar:** A green message box states: "MySQL returned an empty result set (i.e. zero rows). (Query took 0.0001 seconds.)"
- SQL query area:** Displays the query: "SELECT * FROM `users`". Below it are options for Profiling, Edit inline, Edit, Explain SQL, Create PHP code, and Refresh.
- Result preview:** Shows the columns: id, name, email, email_verified_at, password, remember_token, created_at, updated_at.
- Operations:** A "Query results operations" section with a "Create view" button.

Registration Page (<http://127.0.0.1:8000/register>)

The screenshot shows a registration form on a dark-themed web page. At the top, the URL bar displays "127.0.0.1:8000/register". The page features a large, stylized "Laravel" logo in white. Below the logo is a dark rectangular form containing four input fields: "Name", "Email", "Password", and "Confirm Password". Each field has a corresponding placeholder text area below it. At the bottom right of the form is a "REGISTER" button, and to its left is a link "Already registered?".

	id	name	email	email_verified_at	password	remember_token	cre
<input type="checkbox"/>	1	yitayew	yitayewsolomon3@gmail.com	NULL	\$2y\$12\$CEs2jfiMRPR1HqJGQqYHPOJD5/lbPHt5m09h/UySorB...	2020-09-	

User Loge in (<http://127.0.0.1:8000/dashboard>)



Login(<http://127.0.0.1:8000/login>)

The screenshot shows a web browser window with a dark-themed login interface. At the top, the address bar displays the URL `127.0.0.1:8000/login`. The main content area features a large, stylized white 'U' logo centered above a dark gray rectangular form. The form contains fields for 'Email' and 'Password', each with a corresponding input field. Below these fields is a 'Remember me' checkbox. At the bottom right of the form are two buttons: a blue 'Forgot your password?' link and a white 'LOG IN' button with a blue outline.

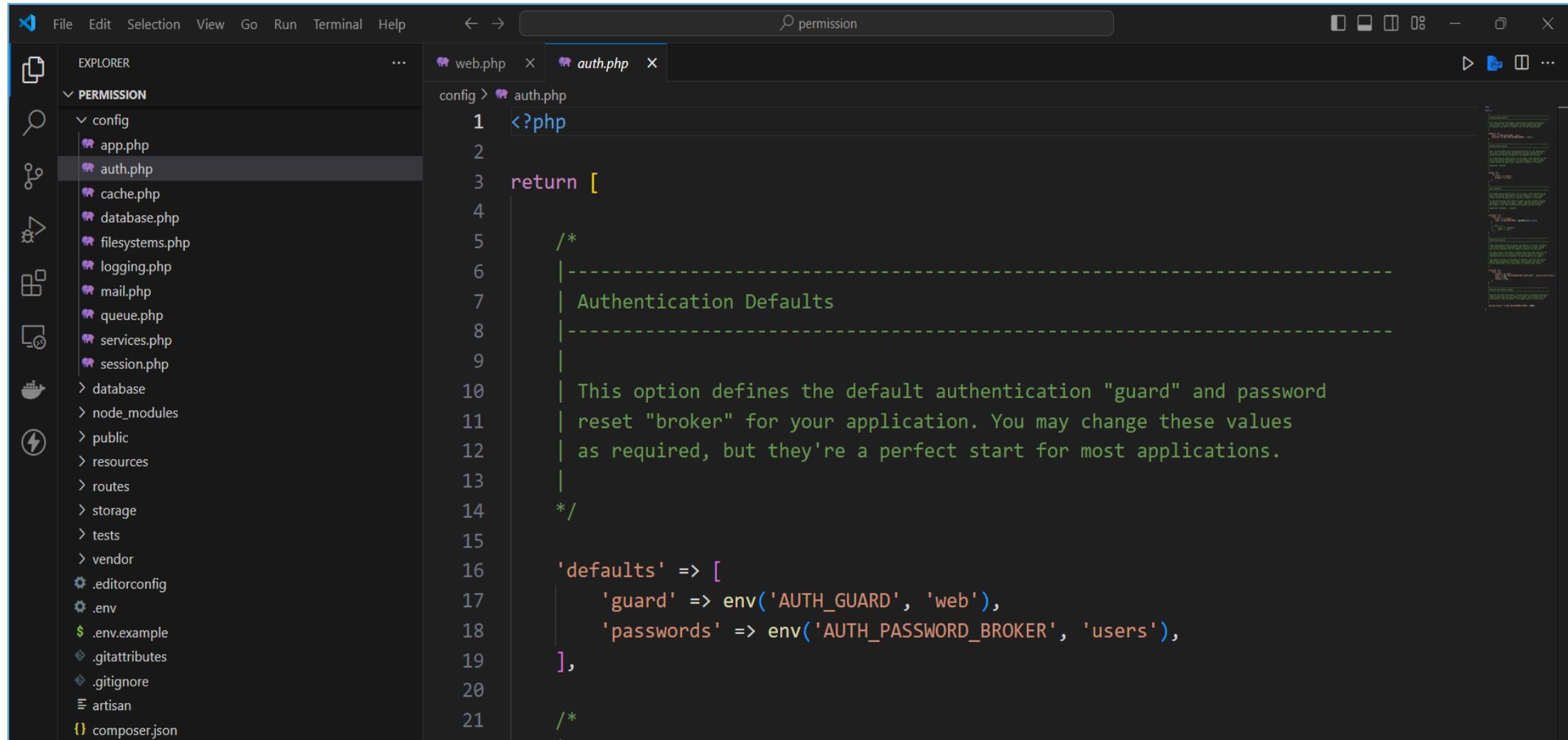
Project Structure (After Installing Breeze)

The screenshot shows a code editor interface with the following details:

- File Menu:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Search Bar:** permission
- Toolbars:** Standard window controls (minimize, maximize, close) and a toolbar with icons for file operations.
- Left Sidebar (EXPLORER):** Shows the project structure:
 - PERMISSION
 - app
 - Http
 - Controllers
 - Auth
 - AuthenticatedSessionController.php
 - ConfirmablePasswordController.php
 - EmailVerificationNotificationController.php
 - EmailVerificationPromptController.php
 - NewPasswordController.php
 - PasswordController.php
 - PasswordResetLinkController.php
 - RegisteredUserController.php
 - VerifyEmailController.php
 - Controller.php
 - ProfileController.php
 - Requests
 - Models
 - Providers
 - View
 - bootstrap
 - config
 - database
 - node_modules
 - public
 - resources
 - css
 - js
- Central Area:** Displays the content of `AuthenticatedSessionController.php`. The file is located at `app > Http > Controllers > Auth > AuthenticatedSessionController.php`.
- Code Content:**

```
5  use App\Http\Controllers\Controller;
6  use App\Http\Requests\Auth\LoginRequest;
7  use Illuminate\Http\RedirectResponse;
8  use Illuminate\Http\Request;
9  use Illuminate\Support\Facades\Auth;
10 use Illuminate\View\View;
11
12 class AuthenticatedSessionController extends Controller
13 {
14     /**
15      * Display the login view.
16      */
17    public function create(): View
18    {
19        return view('auth.login');
20    }
21
22    /**
23     * Handle an incoming authentication request.
24     */
25    public function store(LoginRequest $request): RedirectResponse
```

Config

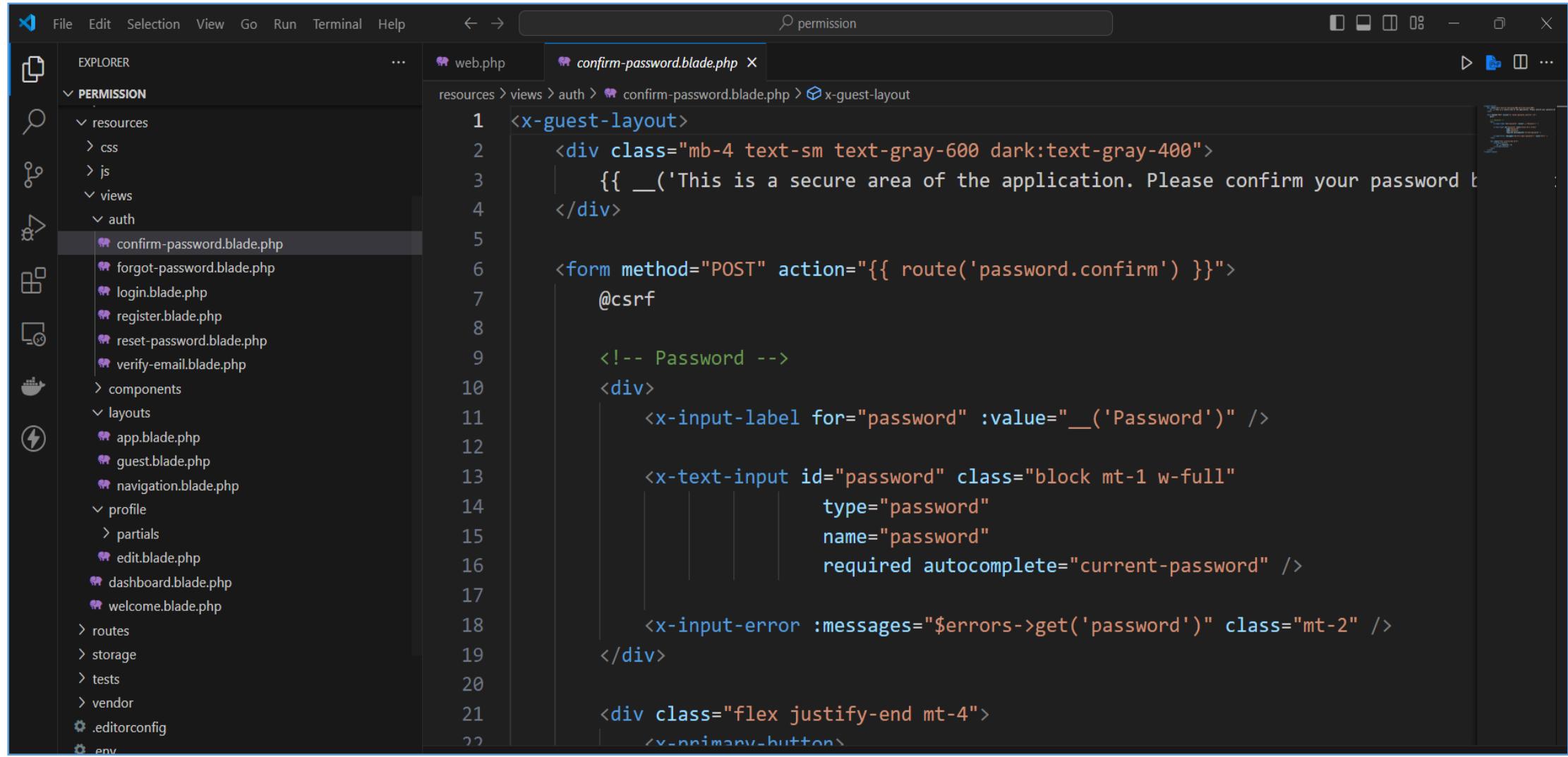


A screenshot of a code editor interface, likely Visual Studio Code, displaying the `auth.php` file from a Laravel application's configuration directory. The editor shows the following code:

```
1 <?php
2
3 return [
4
5     /*
6     | -----
7     | Authentication Defaults
8     | -----
9     |
10    | This option defines the default authentication "guard" and password
11    | reset "broker" for your application. You may change these values
12    | as required, but they're a perfect start for most applications.
13    |
14 */
15
16 'defaults' => [
17     'guard' => env('AUTH_GUARD', 'web'),
18     'passwords' => env('AUTH_PASSWORD_BROKER', 'users'),
19 ],
20
21 /*
```

The code defines the default authentication guard as 'web' and the password broker as 'users'. It also includes a comment explaining the purpose of these settings.

Resource



A screenshot of a code editor interface, likely VS Code, displaying a Blade PHP file named `confirm-password.blade.php`. The file is located in the `resources > views > auth` directory. The code implements a password confirmation form using X-Blade components.

```
<x-guest-layout>
    <div class="mb-4 text-sm text-gray-600 dark:text-gray-400">
        {{ __('This is a secure area of the application. Please confirm your password to continue.') }}
    </div>

    <form method="POST" action="{{ route('password.confirm') }}">
        @csrf

        <!-- Password -->
        <div>
            <x-input-label for="password" :value="__('Password')"/>

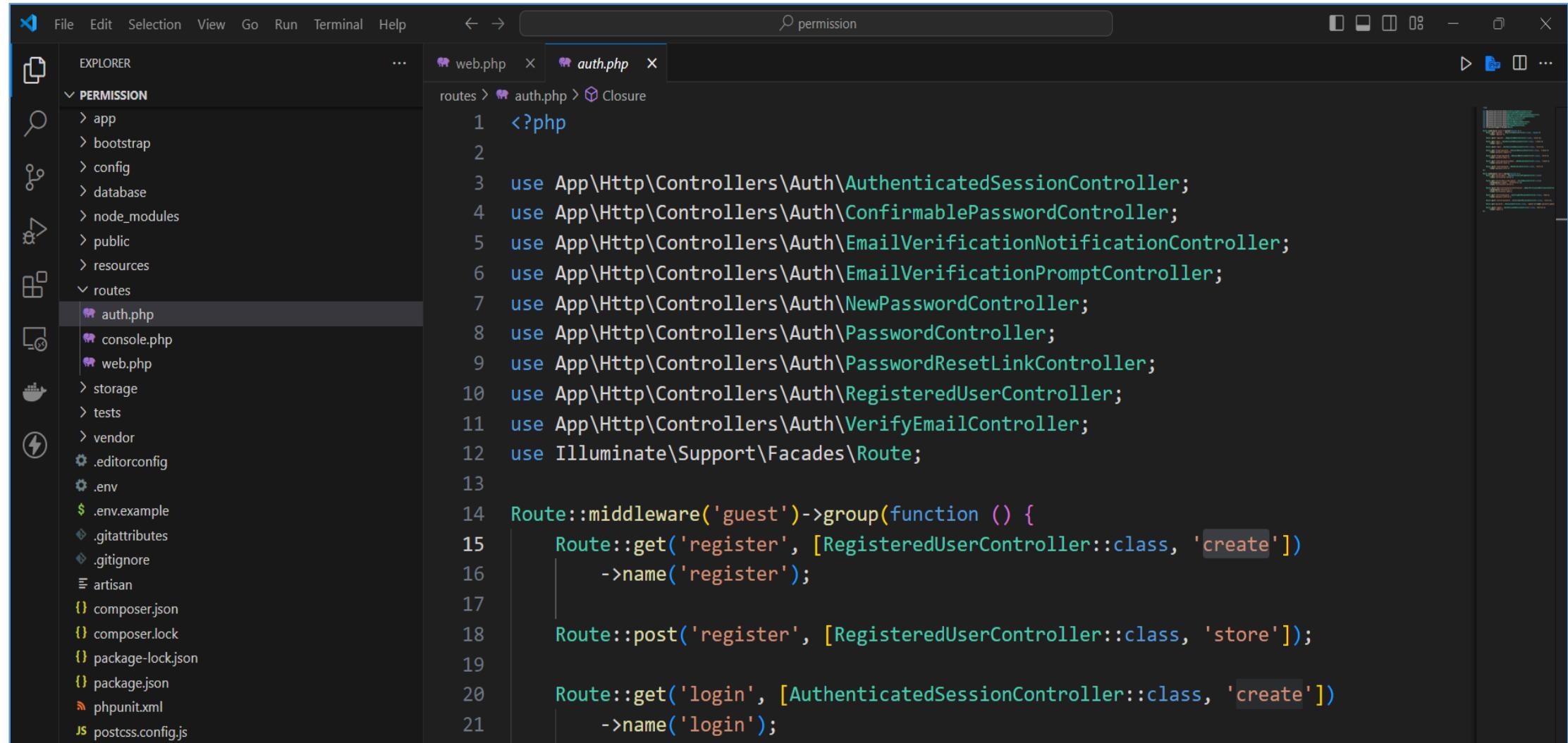
            <x-text-input id="password" class="block mt-1 w-full" type="password" name="password" required autocomplete="current-password" />

            <x-input-error :messages="$errors->get('password')" class="mt-2" />
        </div>

        <div class="flex justify-end mt-4">
            <x-primary-button>
        </x-primary-button>
    </div>

```

Route



A screenshot of a code editor (Visual Studio Code) displaying PHP code for routes. The file is named `auth.php` and is located in the `routes` directory. The code defines several routes using the `Route::middleware` and `Route::group` methods to handle user authentication.

```
<?php

use App\Http\Controllers\Auth\AuthenticatedSessionController;
use App\Http\Controllers\Auth\ConfirmablePasswordController;
use App\Http\Controllers\Auth>EmailVerificationNotificationController;
use App\Http\Controllers\Auth\EmailVerificationPromptController;
use App\Http\Controllers\Auth\NewPasswordController;
use App\Http\Controllers\Auth>PasswordController;
use App\Http\Controllers\Auth>PasswordResetLinkController;
use App\Http\Controllers\Auth\RegisteredUserController;
use App\Http\Controllers\Auth\VerifyEmailController;
use Illuminate\Support\Facades\Route;

Route::middleware('guest')->group(function () {
    Route::get('register', [RegisteredUserController::class, 'create'])
        ->name('register');

    Route::post('register', [RegisteredUserController::class, 'store']);

    Route::get('login', [AuthenticatedSessionController::class, 'create'])
        ->name('login');
});
```

Authorization (Roles and Permissions)

- Roles and Permissions in Laravel allow you to manage **user access control** efficiently. Laravel doesn't include a built-in role and permission system, but popular packages like **Spatie Laravel Permission** make it straightforward to implement.

Key Concepts of Roles and Permissions

Key Concepts of Roles and Permissions

1. **Role:** A named group of permissions, e.g., *Admin, Editor, User*.
2. **Permission:** A specific action that a role or user can perform, e.g., *edit articles, delete users, view dashboard*.
3. **Role-Based Access Control (RBAC):** Assigning roles to users to determine what actions they can perform.
4. **Direct Permissions:** Assign permissions directly to users or associate them with roles.

Laravel 11 ACL - Roles and Permissions Example

Step for Laravel 11 ACL - Roles and Permissions Example

- **Step 1:** Install Laravel 11
- **Step 2:** Install spatie/laravel-permission Package
- **Step 3:** Create Product Migration
- **Step 4:** Create Models
- **Step 5:** Add Middleware
- **Step 6:** Create Authentication
- **Step 7:** Create Routes
- **Step 8:** Add Controllers
- **Step 9:** Add Blade Files
- **Step 10:** Create Seeder For Permissions and AdminUser
- **Run Laravel App**

Step 1 (Installing Laravel)

Step 1: Install Laravel 11

First of all, we need to get a fresh Laravel 11 version application using the command below because we are starting from scratch. So, open your terminal or command prompt and run the command below:

```
composer create-project laravel/laravel example-app
```

Step two (Install Spatie)

I Step 2: Install spatie/laravel-permission Package

Now, we require to install the Spatie package for ACL; that way, we can use its method. Also, we will install the form collection package. So, open your terminal and run the below command.

```
composer require spatie/laravel-permission
```

We can also customize changes on the Spatie package, so if you also want to make changes, you can fire the command below and get the config file in config/permission.php and migration files.

```
php artisan vendor:publish --provider="Spatie\Permission\PermissionServiceProvider"
```

Now you can see the permission.php file and one migration. So you can run the migration using the following command:

```
php artisan migrate
```

Step 3 (Creating Migrations)

Step 3: Create Product Migration

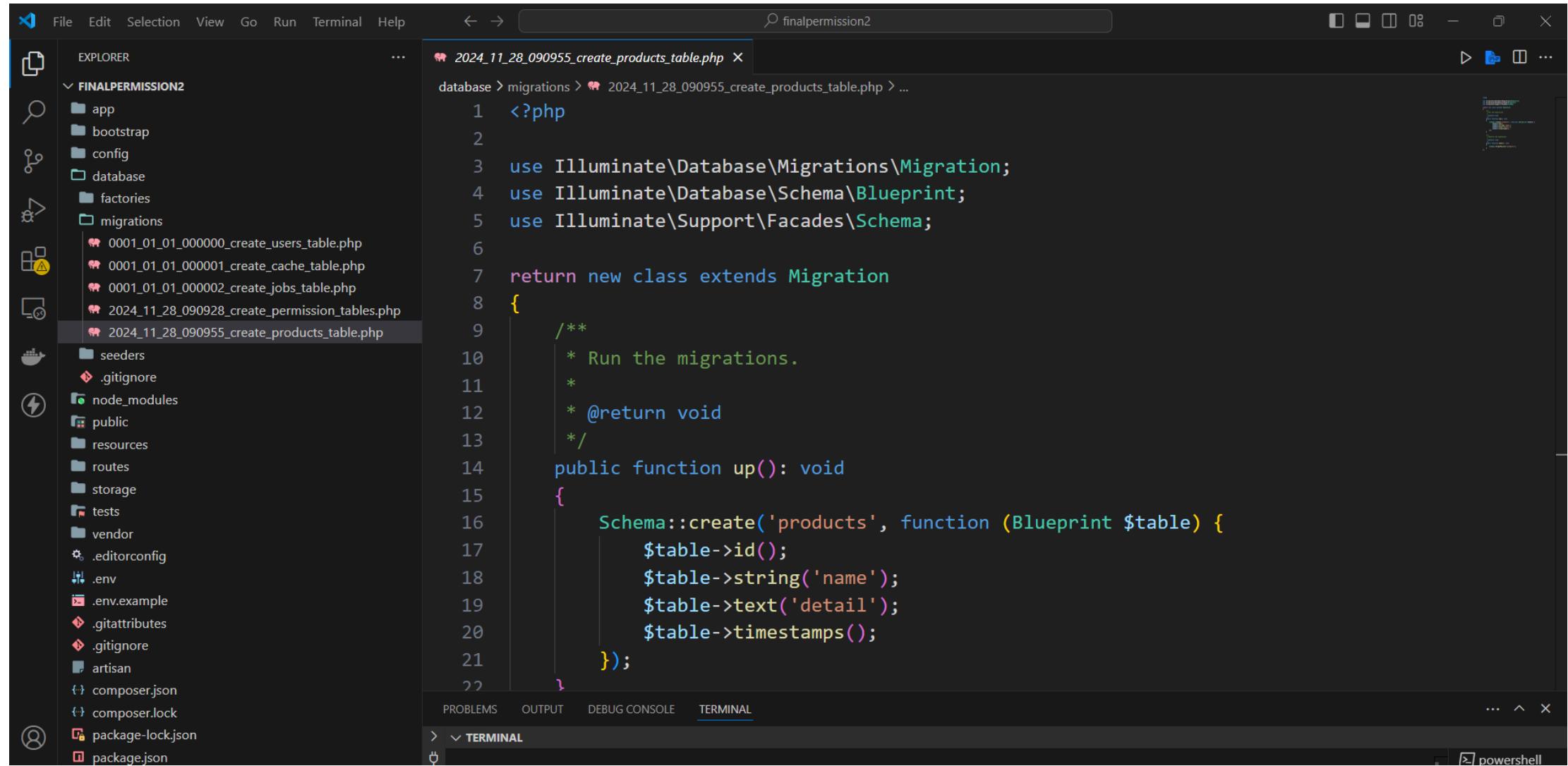
In this step, we have to create three migrations for the products table using the below command:

```
php artisan make:migration create_products_table
```

Now, run migration:

```
php artisan migrate
```

Cont. ...



The screenshot shows a Microsoft Visual Studio Code (VS Code) interface with an orange border. The left sidebar is the Explorer view, showing a project structure for 'FINALPERMISSION2'. The 'migrations' folder contains several files, with '2024_11_28_090955_create_products_table.php' selected and highlighted in grey. The main editor area displays the PHP code for this migration. The code defines a new database table named 'products' with columns for 'name' (string), 'detail' (text), and timestamps. The code uses the Illuminate\Database\Migrations\Migration and Blueprint classes.

```
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up(): void
    {
        Schema::create('products', function (Blueprint $table) {
            $table->id();
            $table->string('name');
            $table->text('detail');
            $table->timestamps();
        });
    }
}
```

At the bottom, the terminal tab is active, showing a powershell prompt. The status bar indicates the file is open in a 'finalpermission2' workspace.

Cont. ...

The screenshot shows the phpMyAdmin interface for a MySQL database named 'finalpermission2'. The 'products' table is selected. A green message bar at the top indicates that MySQL returned an empty result set (i.e. zero rows). Below the message, a SQL query is displayed: 'SELECT * FROM `products`'. The results section shows a single row with columns: id, name, detail, created_at, and updated_at. The 'id' column value is 1, 'name' is 'Laptop', 'detail' is 'Dell XPS 15', 'created_at' is '2024-01-15 10:00:00', and 'updated_at' is '2024-01-15 10:00:00'. There are also 'Query results operations' and 'Create view' buttons.

localhost / 127.0.0.1 / finalperm

localhost/phpmyadmin/index.php?route=/sql&pos=0&db=finalpermission2&table=products

phpMyAdmin

Server: 127.0.0.1 » Database: finalpermission2 » Table: products

Browse Structure SQL Search Insert Export Import Privileges Operations Triggers

MySQL returned an empty result set (i.e. zero rows). (Query took 0.0002 seconds.)

SELECT * FROM `products`

id name detail created_at updated_at

Create view

Recent Favorites

finalpermission finalpermission2

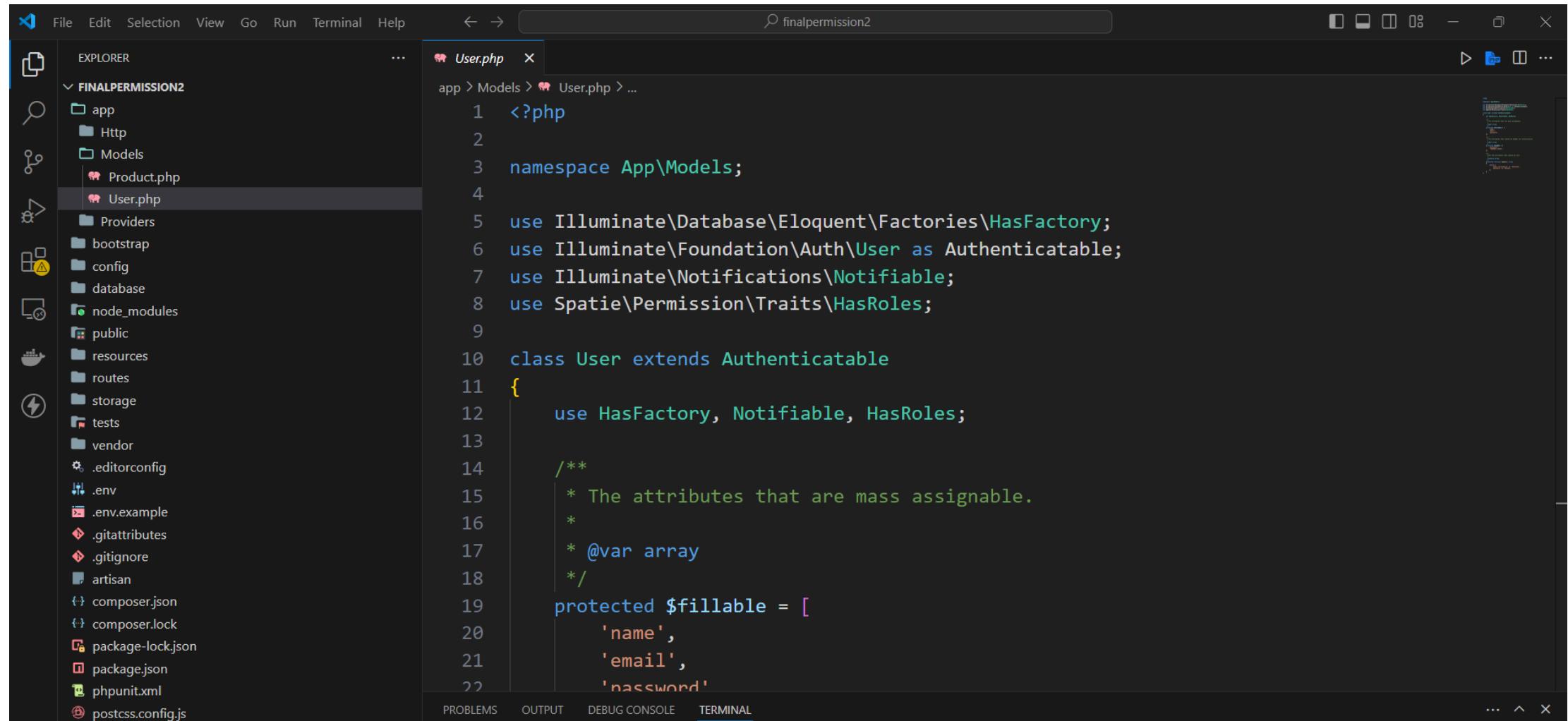
- New
- cache
- cache_locks
- failed_jobs
- jobs
- job_batches
- migrations
- model_has_permissions
- model_has_roles
- password_reset_tokens
- permissions
- products
- roles
- role_has_permissions
- sessions
- users

Step 4 (Creating Models)

Step 4: Create Models

In this step, we have to create a model for the User and Product tables. If you get a fresh project, you will have a User model, so just replace the code, and for the other, you should create one.

Model (User.php)

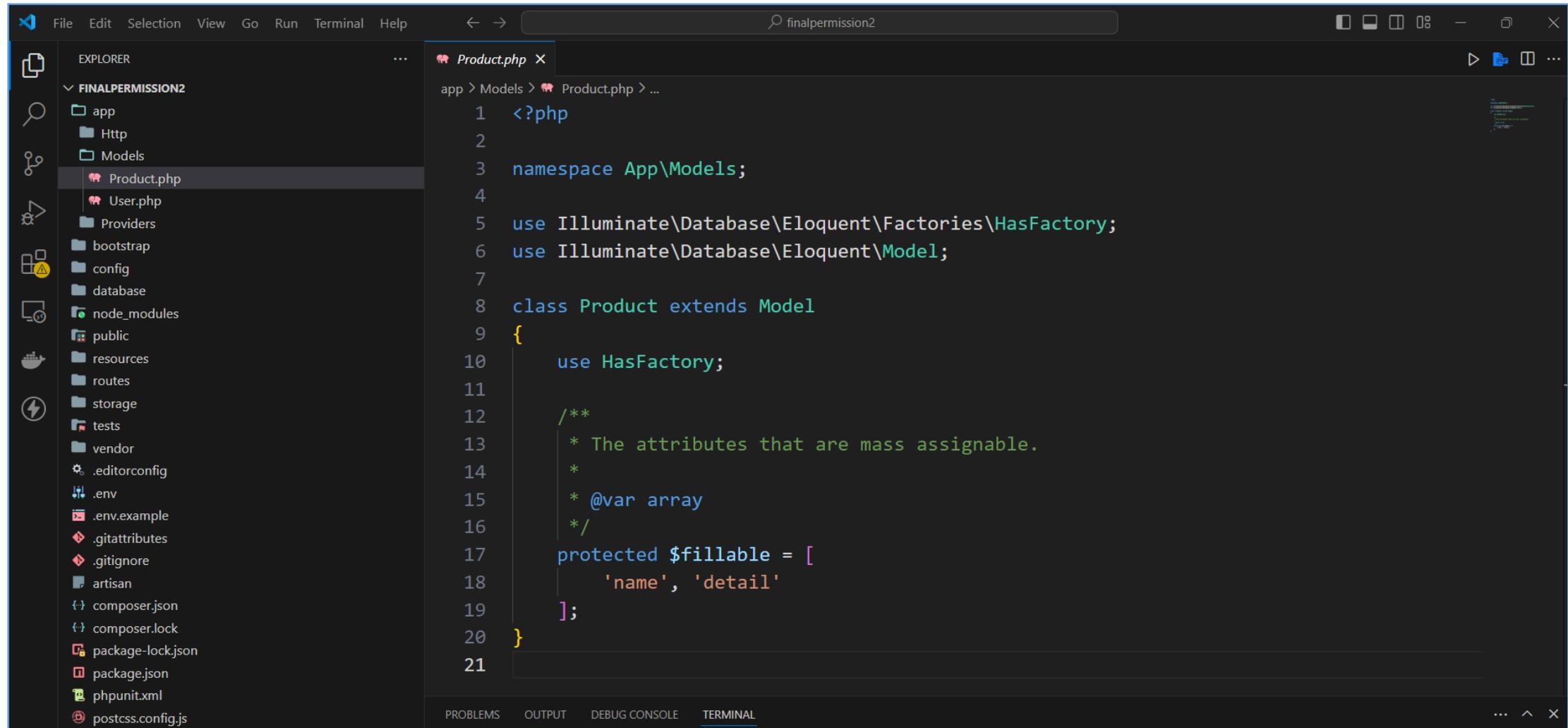


The screenshot shows a Microsoft Visual Studio Code (VS Code) interface with an orange border. The left sidebar is the Explorer view, showing the project structure of 'FINALPERMISSION2'. The 'Models' folder contains 'Product.php' and 'User.php', with 'User.php' currently selected. The main editor area displays the code for 'User.php':

```
<?php  
namespace App\Models;  
  
use Illuminate\Database\Eloquent\Factories\HasFactory;  
use Illuminate\Foundation\Auth\User as Authenticatable;  
use Illuminate\Notifications\Notifiable;  
use Spatie\Permission\Traits\HasRoles;  
  
class User extends Authenticatable  
{  
    use HasFactory, Notifiable, HasRoles;  
  
    /**  
     * The attributes that are mass assignable.  
     *  
     * @var array  
     */  
    protected $fillable = [  
        'name',  
        'email',  
        'password'  
    ];  
}
```

At the bottom of the screen, there are tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, and TERMINAL. The TERMINAL tab is currently active.

Model (Product.php)



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 'FINALPERMISSION2'. The 'Models' folder contains 'Product.php' and 'User.php'. Other visible files include 'app', 'Http', 'Providers', 'bootstrap', 'config', 'database', 'node_modules', 'public', 'resources', 'routes', 'storage', 'tests', 'vendor', '.editorconfig', '.env', '.env.example', '.gitattributes', '.gitignore', 'artisan', 'composer.json', 'composer.lock', 'package-lock.json', 'package.json', 'phpunit.xml', and 'postcss.config.js'. The current file being edited is 'Product.php', which is highlighted in the Explorer and the main editor area. The editor shows the following PHP code:

```
<?php  
namespace App\Models;  
  
use Illuminate\Database\Eloquent\Factories\HasFactory;  
use Illuminate\Database\Eloquent\Model;  
  
class Product extends Model  
{  
    use HasFactory;  
  
    /**  
     * The attributes that are mass assignable.  
     *  
     * @var array  
     */  
    protected $fillable = [  
        'name', 'detail'  
    ];
```

The status bar at the bottom of the editor shows tabs for 'PROBLEMS', 'OUTPUT', 'DEBUG CONSOLE', and 'TERMINAL'. There are also standard window control buttons (minimize, maximize, close) in the top right corner.

Step 5 (Adding Middleware)

Step 5: Add Middleware

Spatie package provides its in-built middleware, which we can use simply and is displayed as below:

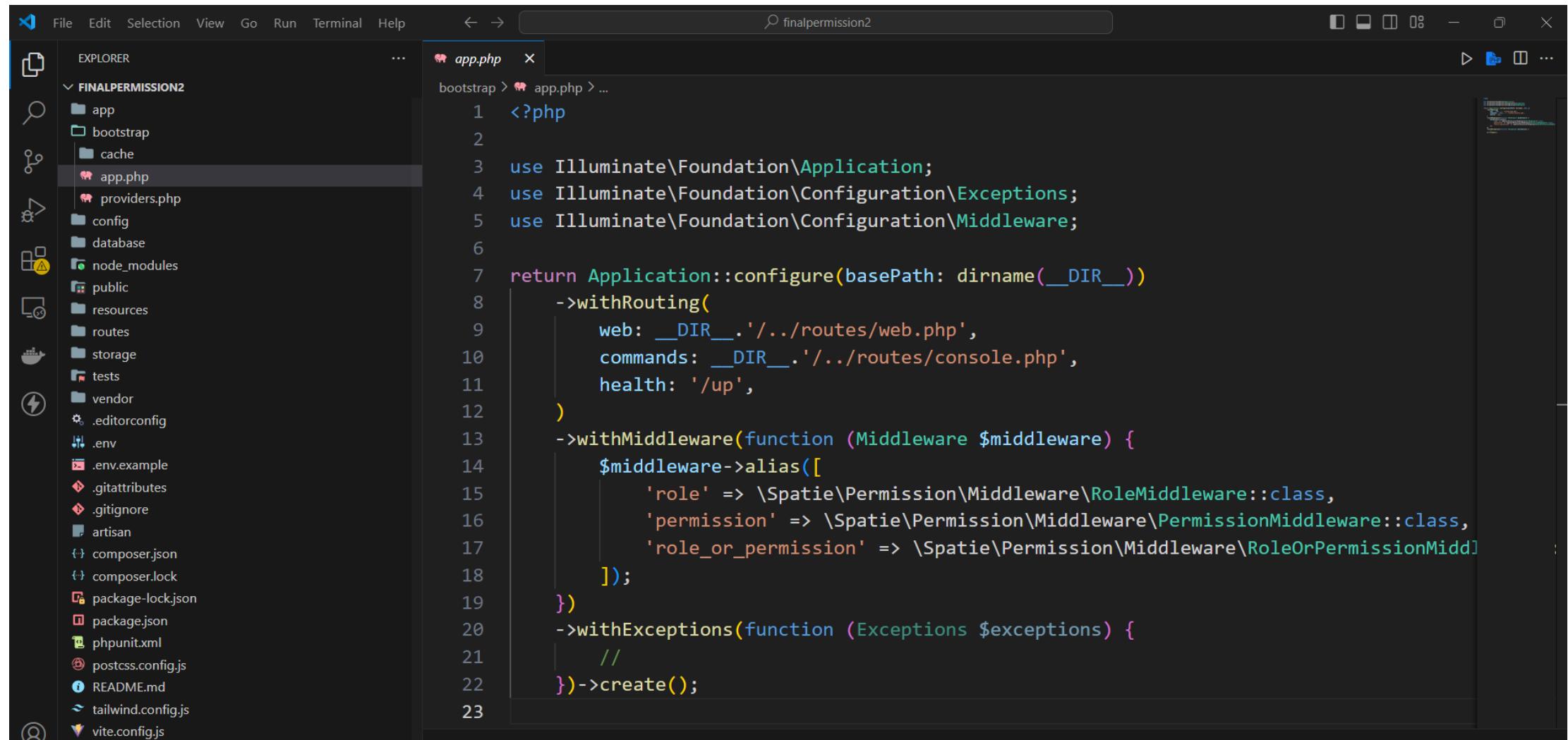
role

permission

So, we have to add middleware in the app.php file this way:

bootstrap/app.php

Cont. ...



The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer (Left):** Shows the project structure for "FINALPERMISSION2". The "app" folder is expanded, showing "bootstrap", "cache", "app.php", "providers.php", "config", "database", "node_modules", "public", "resources", "routes", "storage", "tests", "vendor", ".editorconfig", ".env", ".env.example", ".gitattributes", ".gitignore", "artisan", "composer.json", "composer.lock", "package-lock.json", "package.json", "phpunit.xml", "postcss.config.js", "README.md", "tailwind.config.js", and "vite.config.js".
- Editor (Center):** The file "app.php" is open in the editor. The code is as follows:

```
1 <?php
2
3 use Illuminate\Foundation\Application;
4 use Illuminate\Foundation\Configuration\Exceptions;
5 use Illuminate\Foundation\Configuration\Middleware;
6
7 return Application::configure(basePath: dirname(__DIR__))
8     ->withRouting(
9         web: __DIR__.'/../routes/web.php',
10        commands: __DIR__.'/../routes/console.php',
11        health: '/up',
12    )
13    ->withMiddleware(function (Middleware $middleware) {
14        $middleware->alias([
15            'role' => \Spatie\Permission\Middleware\RoleMiddleware::class,
16            'permission' => \Spatie\Permission\Middleware\PermissionMiddleware::class,
17            'role_or_permission' => \Spatie\Permission\Middleware\RoleOrPermissionMiddl
18        ]);
19    })
20    ->withExceptions(function (Exceptions $exceptions) {
21        //
22    })->create();
23
```

- Search Bar (Top):** Contains the text "finalpermission2".
- Terminal (Bottom):** Not visible in the screenshot.

Step 6 (Creating Authentication Using laravel/ui)

Step 6: Create Authentication

You have to follow a few steps to make auth in your Laravel 11 application.

First, you need to install the laravel/ui package as shown below:

```
composer require laravel/ui
```

Here, we need to generate auth scaffolding in Laravel 11 using Laravel UI command. So, let's generate it by the below command:

```
php artisan ui bootstrap --auth
```

Now you need to run the npm command; otherwise, you cannot see the better layout of the login and register page.

Install NPM:

```
npm install
```

Run NPM:

```
npm run build
```

npm run dev

- ***npm run dev*** is a command used in Laravel projects (and other JavaScript projects) to compile frontend assets, such as CSS and JavaScript, in development mode.
- It leverages the Vite build tool in Laravel (replacing Webpack Mix in earlier versions) to bundle and serve assets efficiently.

How to run npm

Full Syntax of `npm run dev`

1. Run in Project Root: You must run the command in the root directory of your Laravel project, where the `package.json` file is located.

bash

 Copy code

```
npm run dev
```

Cont. ...

ChatGPT ▾

Share

2. What Happens Under the Hood:

- `npm run dev` executes the `dev` script defined in the `package.json` file.
- In a Laravel project using Vite, the `dev` script typically looks like this:

json

Copy code

```
"scripts": {  
    "dev": "vite",  
    "build": "vite build"  
}
```

- The `vite` command starts the development server for asset compilation.

Output (<http://localhost:5173/>)

A screenshot of a web browser window displaying a dark-themed page. The address bar shows 'localhost:5173'. The page itself is a Vite development server interface for a Laravel application. It features two logos at the top: a red 'V' icon followed by a plus sign and a blue and yellow lightning bolt icon. Below the logos, text reads: 'This is the Vite development server that provides Hot Module Replacement for your Laravel application.' Further down, it says: 'To access your Laravel application, you will need to run a local development server.' Two sections are listed at the bottom: 'Artisan Serve' (described as Laravel's local development server powered by PHP's built-in web server) and 'Laravel Sail' (described as a light-weight command-line interface for interacting with Laravel's default Docker development environment). The browser interface includes standard navigation buttons (back, forward, search), a zoom level of 130%, and various window control icons.

This is the Vite development server that provides Hot Module Replacement for your Laravel application.

To access your Laravel application, you will need to run a local development server.

Artisan Serve

Laravel's local development server powered by PHP's built-in web server.

Laravel Sail

A light-weight command-line interface for interacting with Laravel's default Docker development environment.

Output (<http://127.0.0.1:8000/>)

A screenshot of a web browser window displaying the Laravel homepage at <http://127.0.0.1:8000>. The page features a dark background with a red-to-black gradient header. The Laravel logo is visible in the top right corner. Below the header, there's a large image of the Laravel UI builder interface. To the right of the image, there are two cards: one for Laracasts and one for Laravel News.

The browser's address bar shows the URL <http://127.0.0.1:8000>. The page includes standard navigation icons like back, forward, and refresh, along with zoom controls (120%) and a star icon for bookmarks.

Laravel

Laravel offers thousands of video tutorials on Laravel, PHP, and JavaScript development. Check them out, see for yourself, and massively level up your development skills in the process.

Laravel News

Laravel News is a community driven portal and newsletter aggregating all of the latest and most important news in the Laravel ecosystem, including new package releases and tutorials.

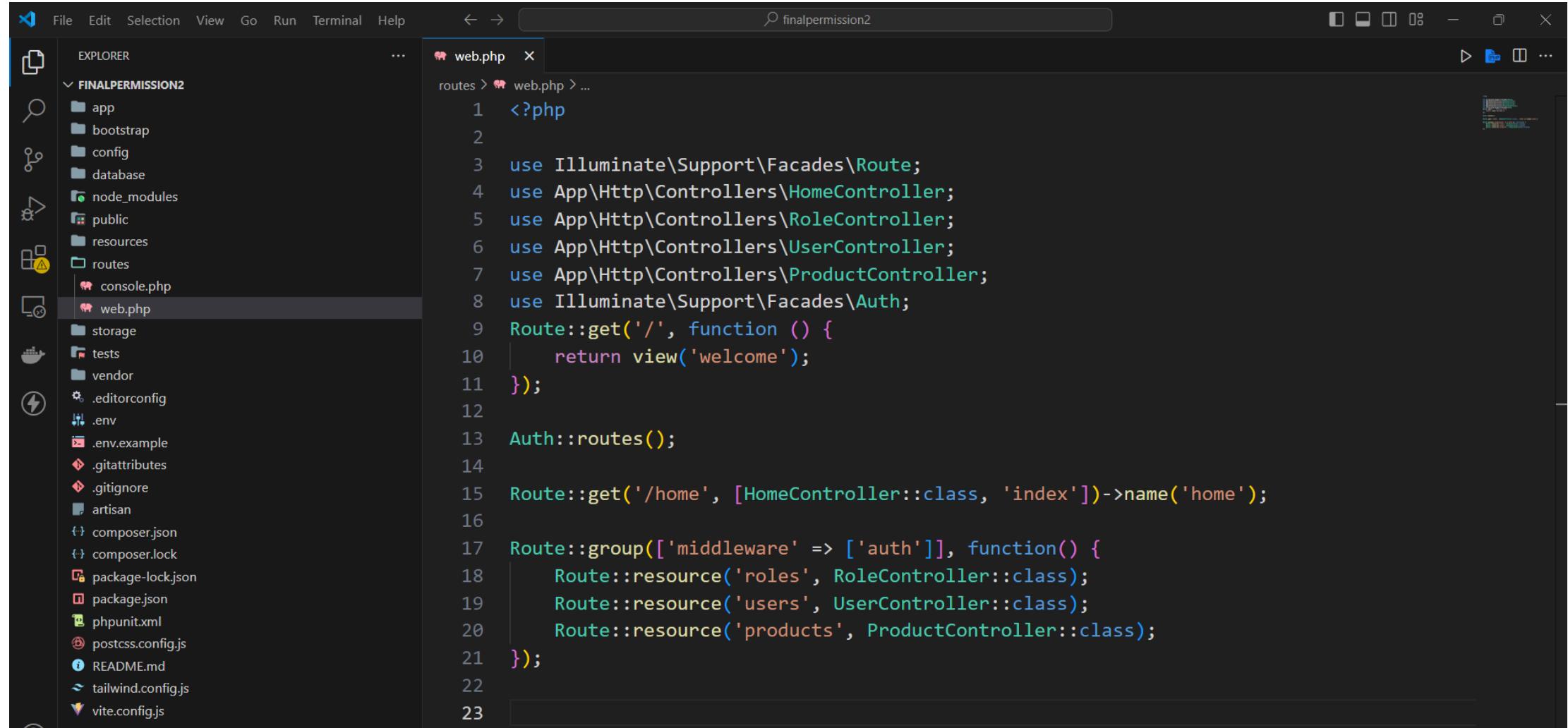
Step 7 (Creating Routes)

Step 7: Create Routes

We need to add a number of routes for the users module, products module, and roles module. In these routes, I also use middleware with permissions for the roles and products routes, so add routes this way:

`routes/web.php`

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 'FINALPERMISSION2'. The 'routes' folder contains 'console.php' and 'web.php', with 'web.php' currently selected. The main editor area displays the 'web.php' file content:

```
<?php  
use Illuminate\Support\Facades\Route;  
use App\Http\Controllers\HomeController;  
use App\Http\Controllers\RoleController;  
use App\Http\Controllers\UserController;  
use App\Http\Controllers\ProductController;  
use Illuminate\Support\Facades\Auth;  
Route::get('/', function () {  
    return view('welcome');  
});  
Auth::routes();  
Route::get('/home', [HomeController::class, 'index'])->name('home');  
Route::group(['middleware' => ['auth']], function() {  
    Route::resource('roles', RoleController::class);  
    Route::resource('users', UserController::class);  
    Route::resource('products', ProductController::class);  
});
```

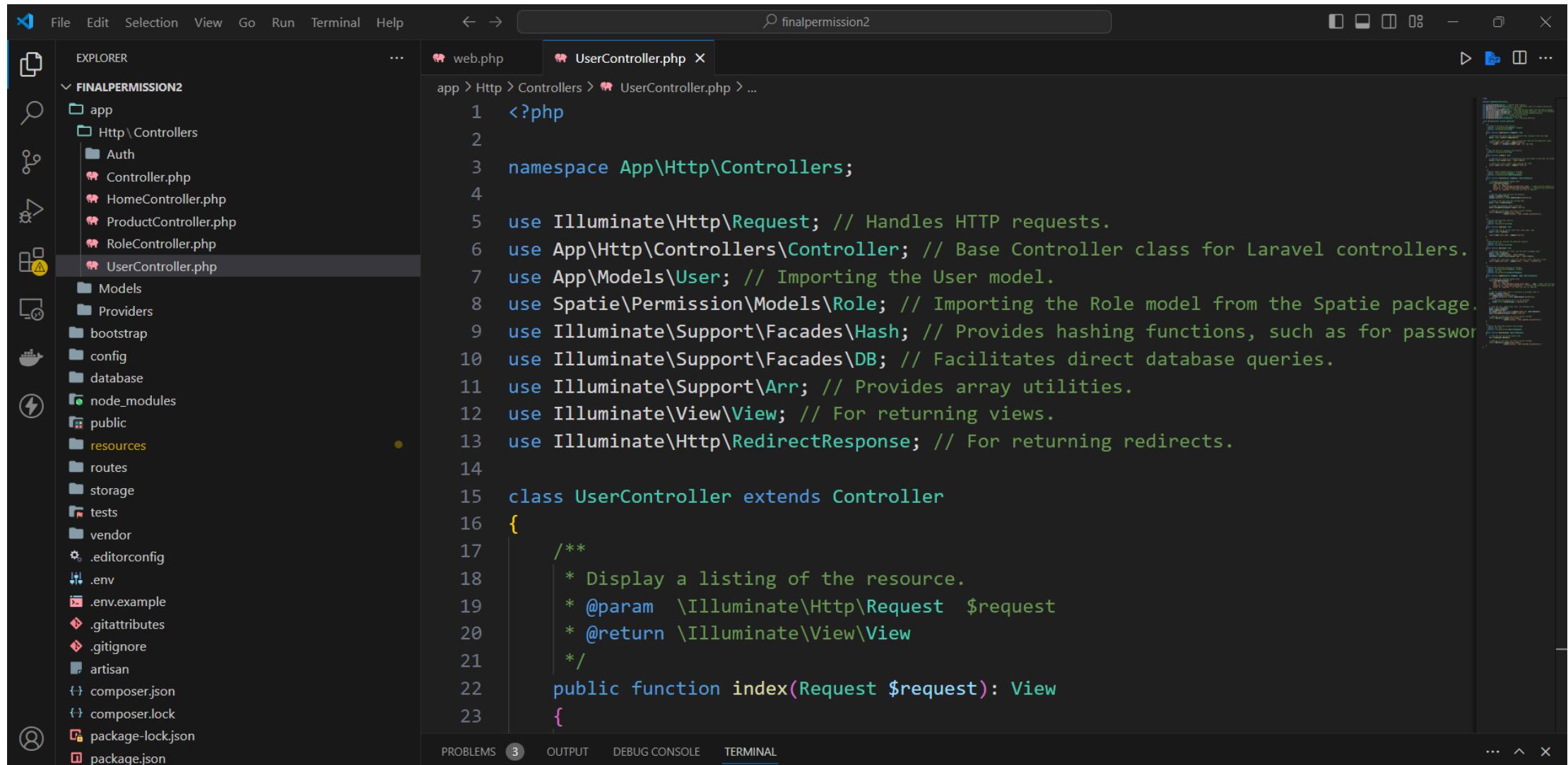
Step 8 (Add Controllers)

Step 8: Add Controllers

In this step, we have added three controllers for the users module, products module, and roles module. So, you can create three controllers as below:

`app/Http/Controllers/UserController.php`

app/Http/Controllers/UserController.php



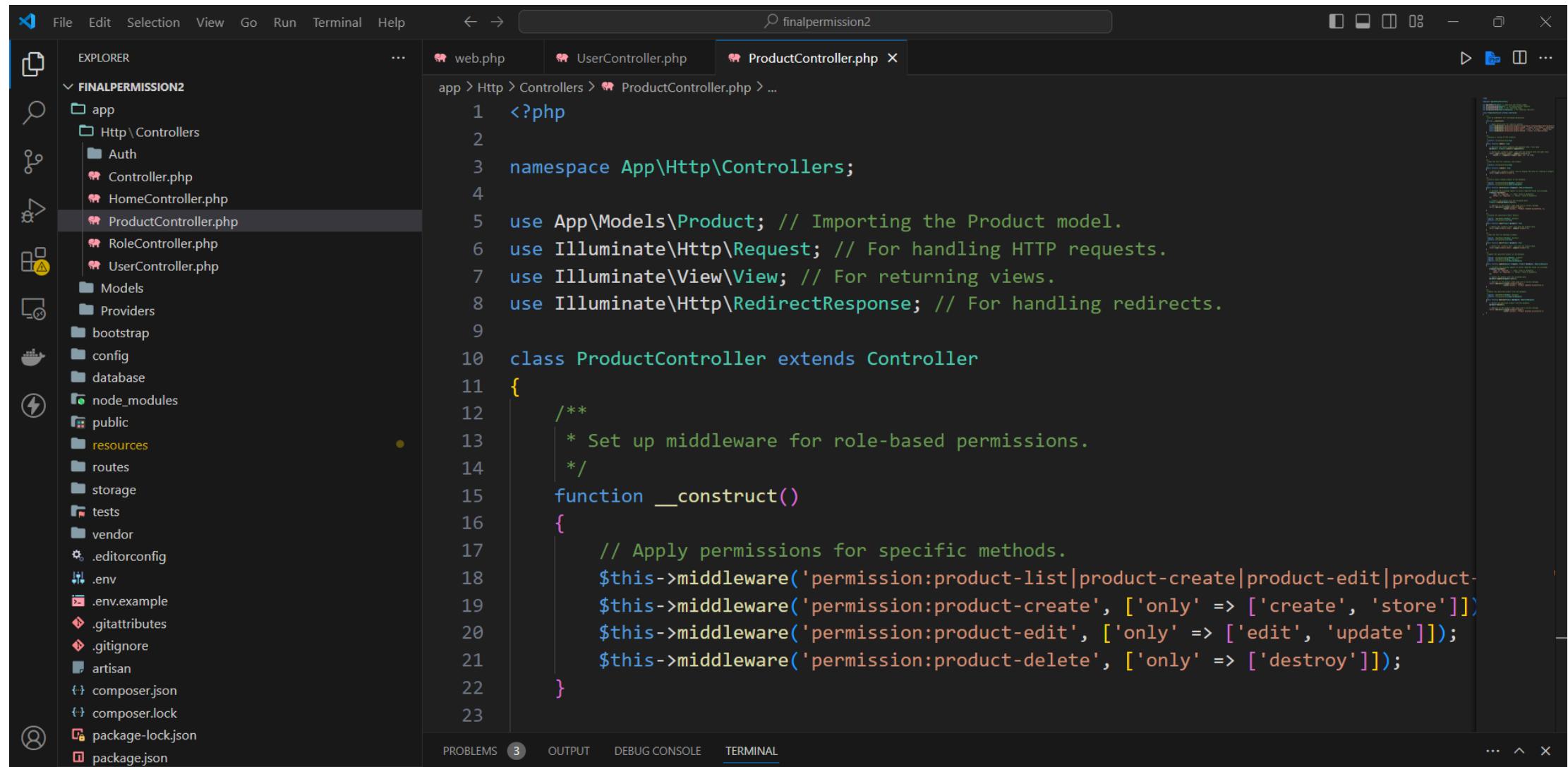
The screenshot shows a Microsoft Visual Studio Code (VS Code) interface with the following details:

- File Explorer (Left):** Shows the project structure under the folder "FINALPERMISSION2". The "Http\Controllers" folder contains "Auth", "Controller.php", "HomeController.php", "ProductController.php", "RoleController.php", and "UserController.php".
- Editor (Center):** The file "UserController.php" is open. The code is a Laravel controller:

```
<?php  
namespace App\Http\Controllers;  
  
use Illuminate\Http\Request; // Handles HTTP requests.  
use App\Http\Controllers\Controller; // Base Controller class for Laravel controllers.  
use App\Models\User; // Importing the User model.  
use Spatie\Permission\Models\Role; // Importing the Role model from the Spatie package.  
use Illuminate\Support\Facades\Hash; // Provides hashing functions, such as for password hashing.  
use Illuminate\Support\Facades\DB; // Facilitates direct database queries.  
use Illuminate\Support\Arr; // Provides array utilities.  
use Illuminate\View\View; // For returning views.  
use Illuminate\Http\RedirectResponse; // For returning redirects.  
  
class UserController extends Controller  
{  
    /**  
     * Display a listing of the resource.  
     * @param \Illuminate\Http\Request $request  
     * @return \Illuminate\View\View  
    */  
    public function index(Request $request): View  
    {
```

- Search Bar (Top):** Contains the text "finalpermission2".
- Bottom Navigation:** Includes tabs for PROBLEMS (3), OUTPUT, DEBUG CONSOLE, and TERMINAL.

app/Http/Controllers/ProductController.php

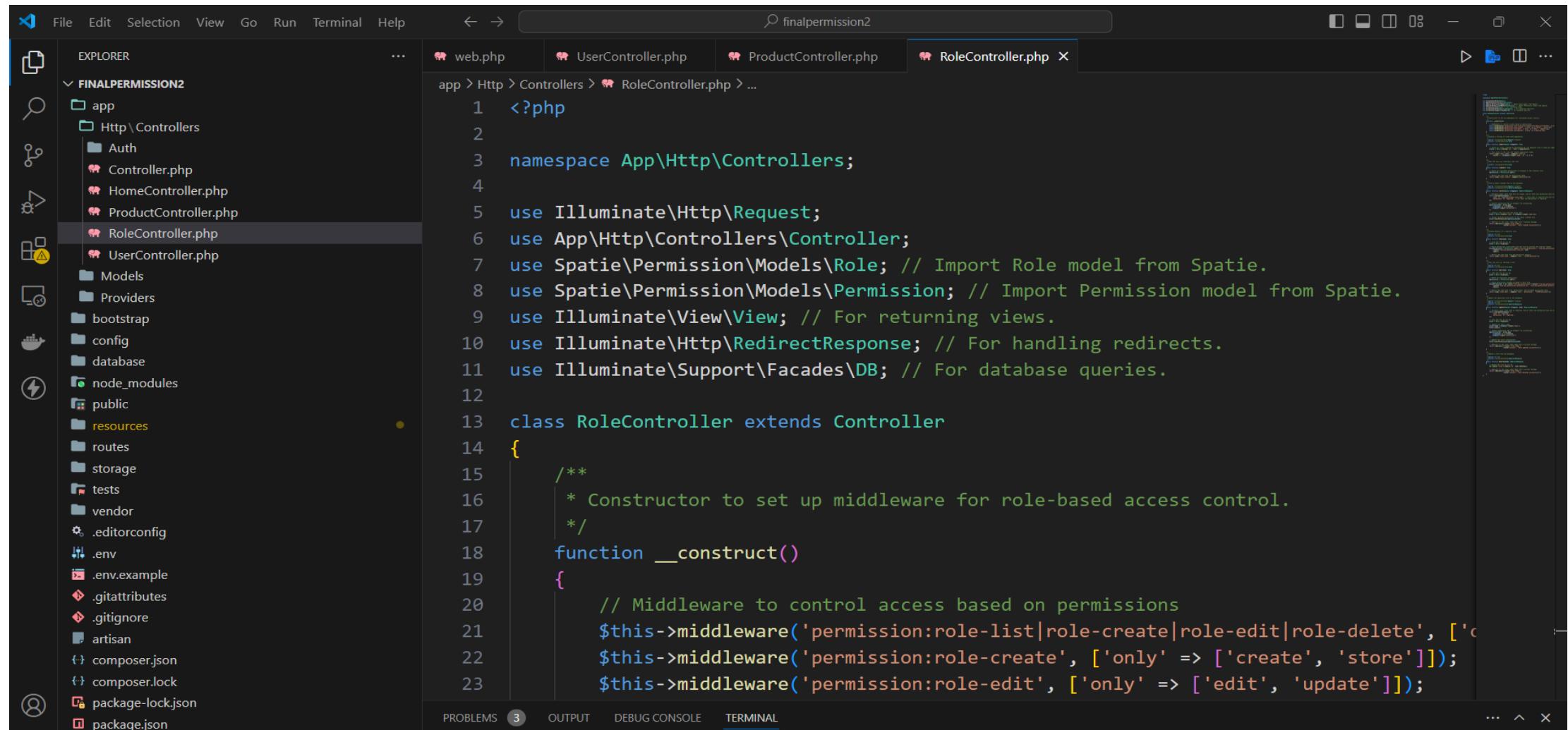


The screenshot shows a code editor interface with a dark theme. The left sidebar displays a file tree for a Laravel application named 'FINALPERMISSION2'. The 'app' directory contains 'Http\Controllers' which includes 'Auth', 'Controller.php', 'HomeController.php', and 'ProductController.php'. The 'ProductController.php' file is currently selected and open in the main editor area. The code implements role-based permissions using middleware.

```
1 <?php
2
3 namespace App\Http\Controllers;
4
5 use App\Models\Product; // Importing the Product model.
6 use Illuminate\Http\Request; // For handling HTTP requests.
7 use Illuminate\View\View; // For returning views.
8 use Illuminate\Http\RedirectResponse; // For handling redirects.
9
10 class ProductController extends Controller
11 {
12     /**
13      * Set up middleware for role-based permissions.
14      */
15     function __construct()
16     {
17         // Apply permissions for specific methods.
18         $this->middleware('permission:product-list|product-create|product-edit|product-
19         $this->middleware('permission:product-create', ['only' => ['create', 'store']]);
20         $this->middleware('permission:product-edit', ['only' => ['edit', 'update']]);
21         $this->middleware('permission:product-delete', ['only' => ['destroy']]);
22     }
23 }
```

The status bar at the bottom shows 'PROBLEMS 3', 'OUTPUT', 'DEBUG CONSOLE', and 'TERMINAL'.

app/Http/Controllers/RoleController.php



The screenshot shows a Microsoft Visual Studio Code (VS Code) interface with the following details:

- File Explorer (Left):** Shows the project structure under the folder "FINALPERMISSION2". The "Http\Controllers" folder is expanded, and "RoleController.php" is selected.
- Editor (Center):** Displays the contents of the "RoleController.php" file. The code is written in PHP and defines a controller for managing roles.
- Bottom Status Bar:** Shows tabs for "PROBLEMS" (with 3), "OUTPUT", "DEBUG CONSOLE", and "TERMINAL".

```
<?php
namespace App\Http\Controllers;
use Illuminate\Http\Request;
use App\Http\Controllers\Controller;
use Spatie\Permission\Models\Role; // Import Role model from Spatie.
use Spatie\Permission\Models\Permission; // Import Permission model from Spatie.
use Illuminate\View\View; // For returning views.
use Illuminate\Http\RedirectResponse; // For handling redirects.
use Illuminate\Support\Facades\DB; // For database queries.

class RoleController extends Controller
{
    /**
     * Constructor to set up middleware for role-based access control.
     */
    function __construct()
    {
        // Middleware to control access based on permissions
        $this->middleware('permission:role-list|role-create|role-edit|role-delete', [
            $this->middleware('permission:role-create', ['only' => ['create', 'store']]);
            $this->middleware('permission:role-edit', ['only' => ['edit', 'update']]);
        ]);
    }
}
```

Step 9 (Adding Blade Files)

Step 9: Add Blade Files

In this step, we need to create the following files as listed below:

Theme Layout

app.blade.php

Users Module

index.blade.php create.blade.php edit.blade.php show.blade.php

Roles Module

index.blade.php create.blade.php edit.blade.php show.blade.php

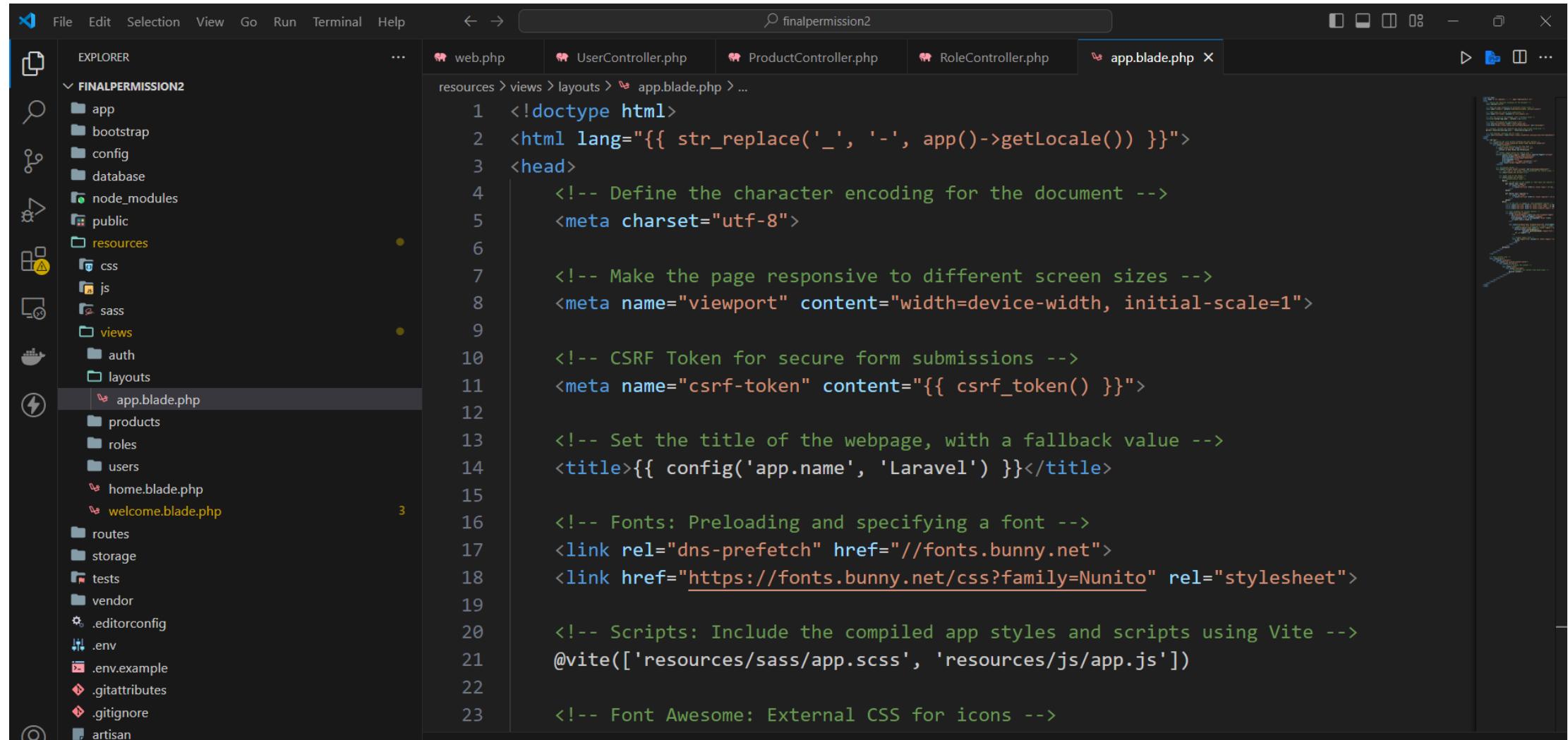
Product Module

index.blade.php create.blade.php edit.blade.php show.blade.php

So, let's create following files:

[resources/views/layouts/app.blade.php](#)

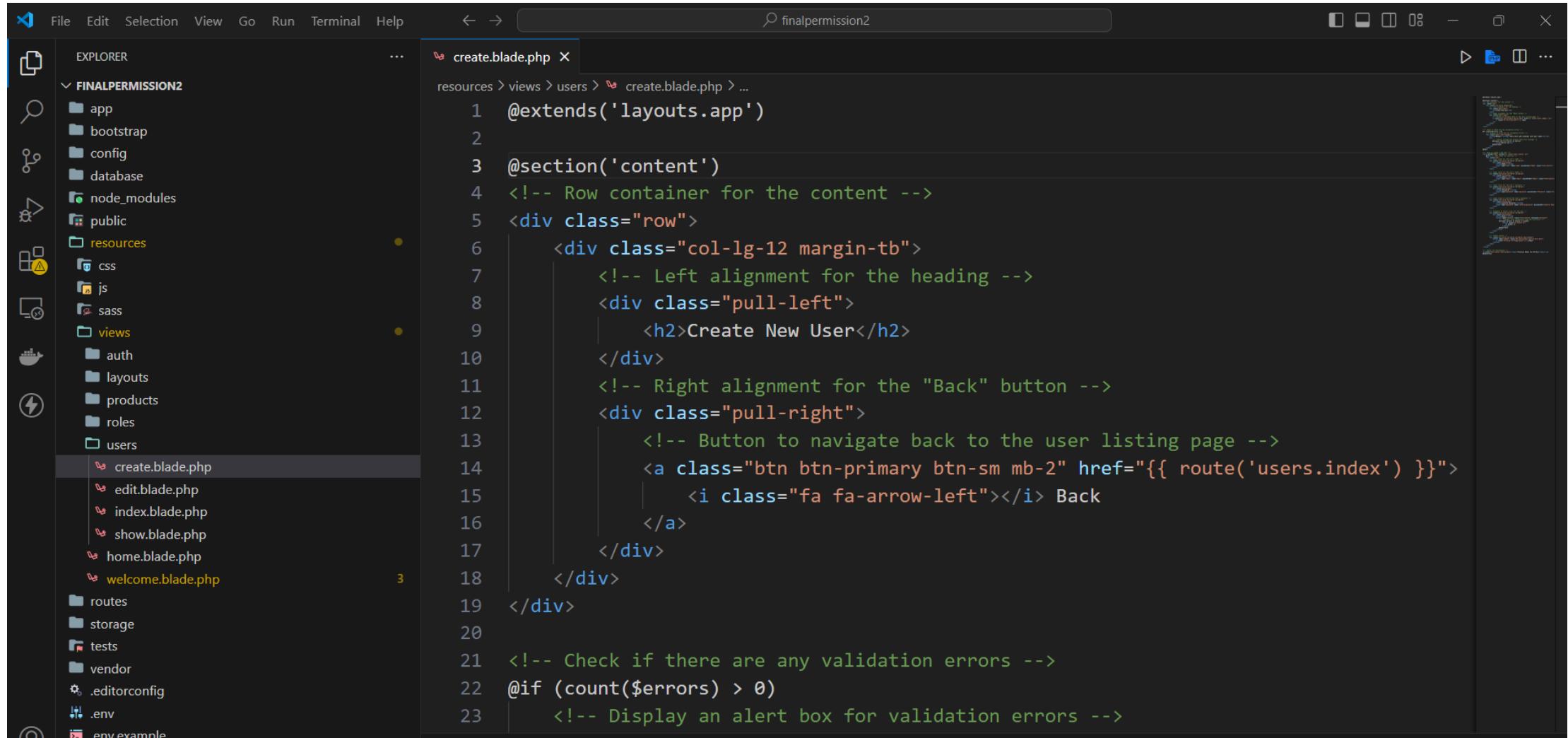
resources/views/layouts/app.blade.php



The screenshot shows a code editor interface with the following details:

- File Menu:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Search Bar:** finalpermission2
- Tab Bar:** web.php, UserController.php, ProductController.php, RoleController.php, app.blade.php (active tab).
- Explorer:** Shows the project structure under FINALPERMISSION2:
 - app
 - bootstrap
 - config
 - database
 - node_modules
 - public
 - resources
 - css
 - js
 - sass
 - views
 - auth
 - layouts
 - app.blade.php
 - products
 - roles
 - users
 - home.blade.php
 - welcome.blade.php
 - routes
 - storage
 - tests
 - vendor
 - .editorconfig
 - .env
 - .env.example
 - .gitattributes
 - .gitignore
 - artisan
- Code Area:** Displays the contents of the app.blade.php file. The code includes standard HTML headers, meta tags for responsiveness and CSRF protection, a title tag, font loading for Nunito, and a Vite script inclusion directive. Line numbers 1 through 23 are visible on the left.

resources/views/users/index.blade.php



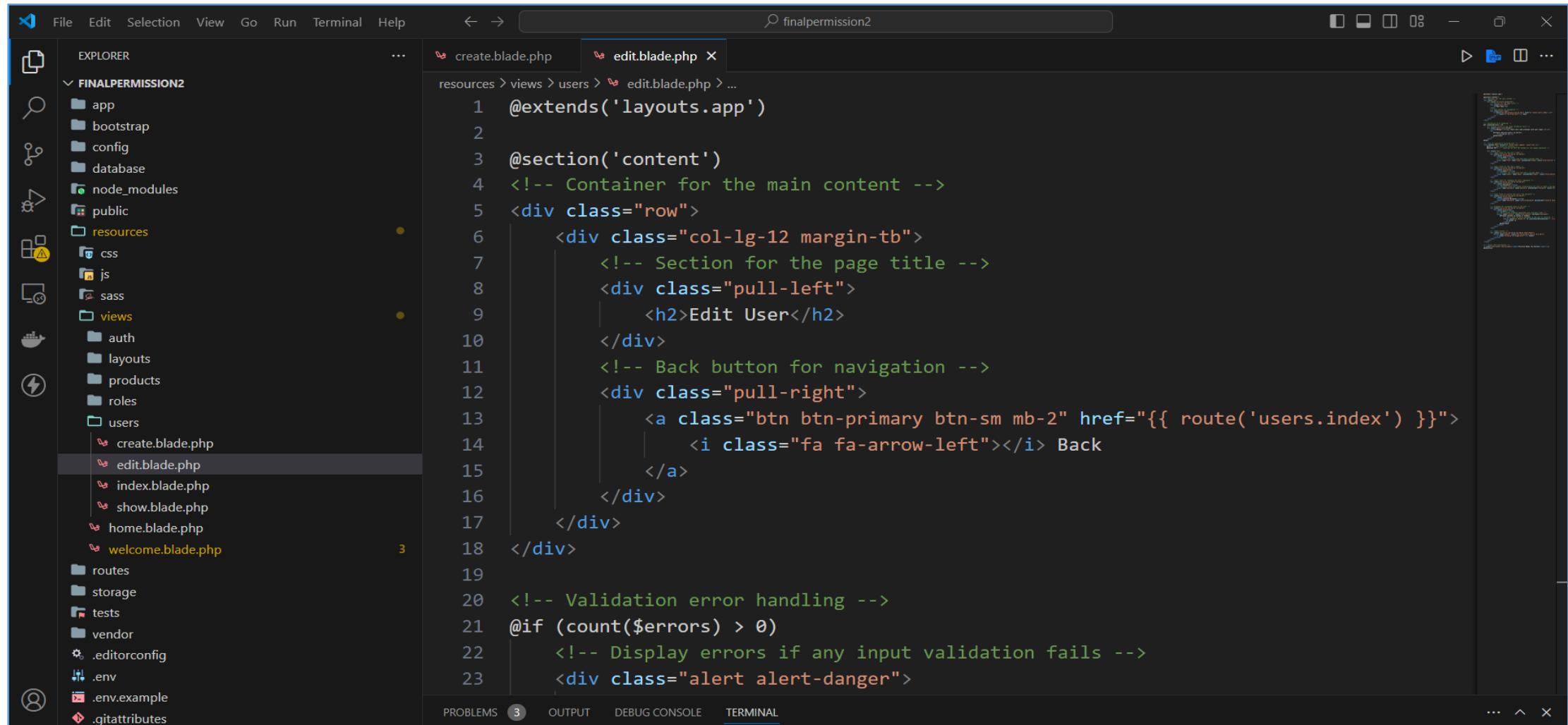
The screenshot shows a code editor interface with a dark theme. The left sidebar is an 'EXPLORER' view showing the project structure:

- FINALPERMISSION2
 - app
 - bootstrap
 - config
 - database
 - node_modules
 - public
 - resources
 - css
 - js
 - sass
 - views
 - auth
 - layouts
 - products
 - roles
 - users
 - create.blade.php
 - edit.blade.php
 - index.blade.php
 - show.blade.php
 - home.blade.php
 - welcome.blade.php
 - routes
 - storage
 - tests
 - vendor
 - .editorconfig
 - .env
 - env.example

The main editor area displays the contents of 'index.blade.php':

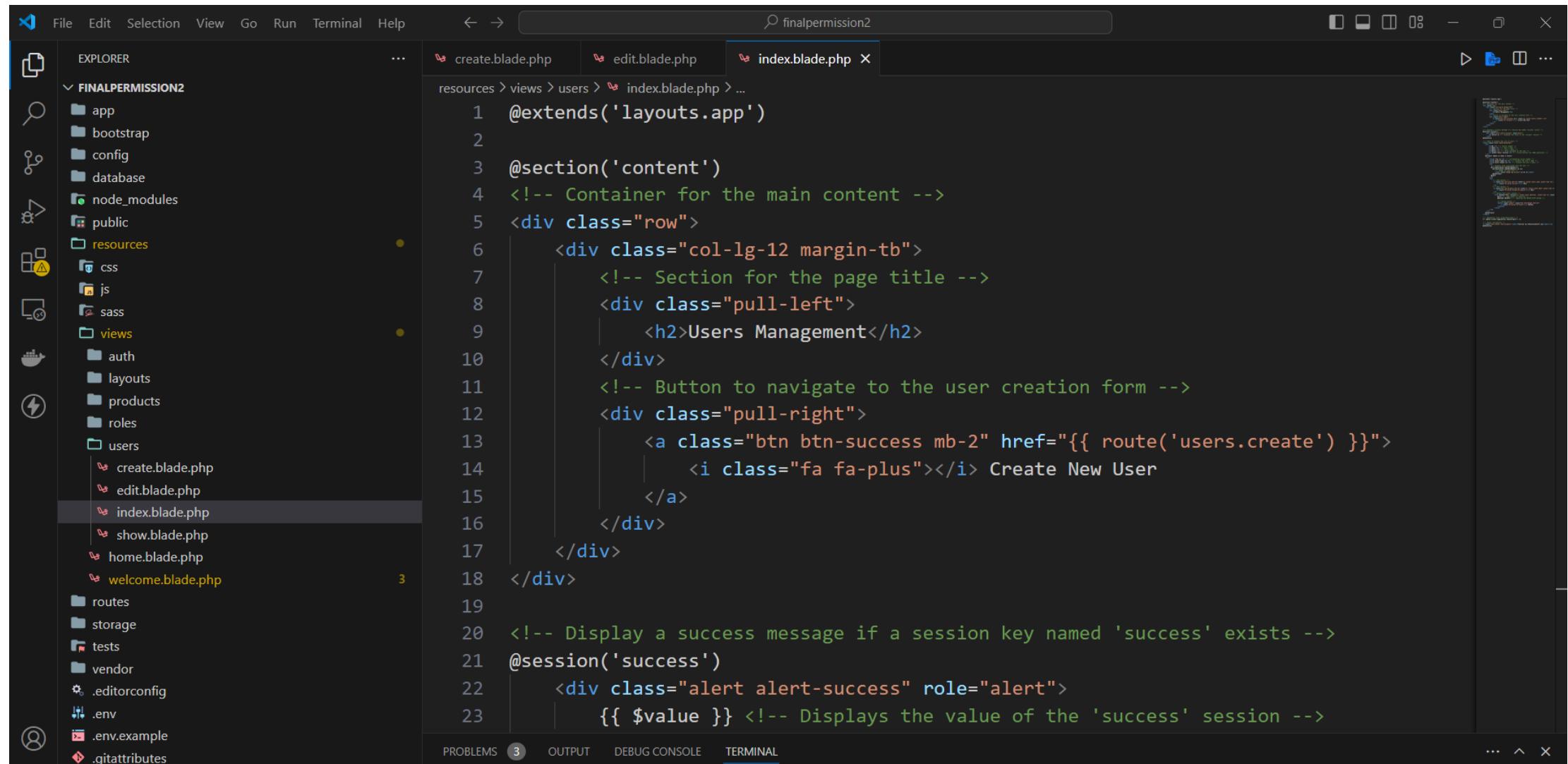
```
1 @extends('layouts.app')
2
3 @section('content')
4 <!-- Row container for the content -->
5 <div class="row">
6     <div class="col-lg-12 margin-tb">
7         <!-- Left alignment for the heading -->
8         <div class="pull-left">
9             <h2>Create New User</h2>
10        </div>
11        <!-- Right alignment for the "Back" button -->
12        <div class="pull-right">
13            <!-- Button to navigate back to the user listing page -->
14            <a class="btn btn-primary btn-sm mb-2" href="{{ route('users.index') }}">
15                <i class="fa fa-arrow-left"></i> Back
16            </a>
17        </div>
18    </div>
19 </div>
20
21 <!-- Check if there are any validation errors -->
22 @if (count($errors) > 0)
23     <!-- Display an alert box for validation errors -->
```

resources/views/users/edit.blade.php



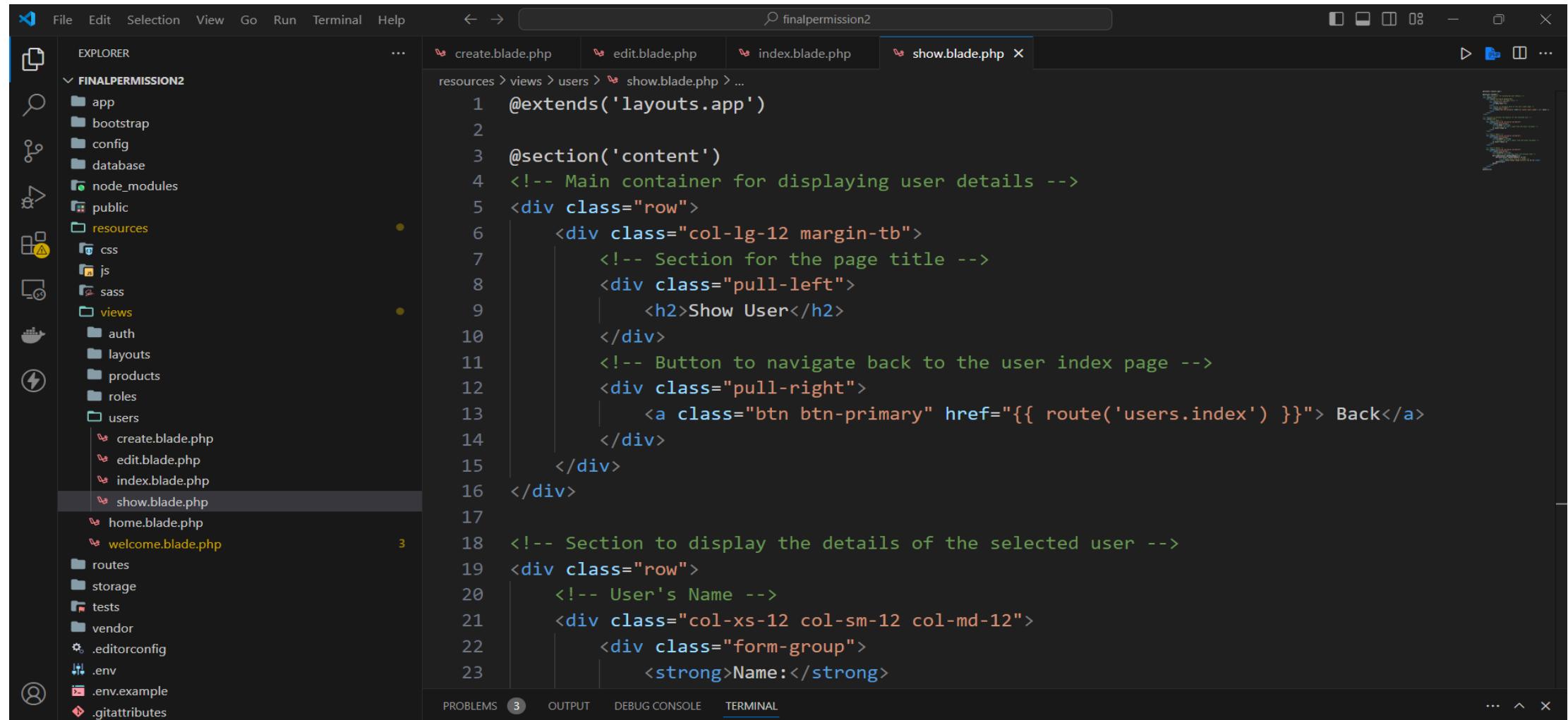
```
1 @extends('layouts.app')
2
3 @section('content')
4 <!-- Container for the main content -->
5 <div class="row">
6     <div class="col-lg-12 margin-tb">
7         <!-- Section for the page title -->
8         <div class="pull-left">
9             <h2>Edit User</h2>
10        </div>
11        <!-- Back button for navigation -->
12        <div class="pull-right">
13            <a class="btn btn-primary btn-sm mb-2" href="{{ route('users.index') }}">
14                <i class="fa fa-arrow-left"></i> Back
15            </a>
16        </div>
17    </div>
18 </div>
19
20 <!-- Validation error handling -->
21@if (count($errors) > 0)
22    <!-- Display errors if any input validation fails -->
23    <div class="alert alert-danger">
```

resources/views/users/index.blade.php



```
File Edit Selection View Go Run Terminal Help ← → 🔍 finalpermission2
EXPLORER ... resources > views > users > index.blade.php > ...
FINALPERMISSION2
  app
  bootstrap
  config
  database
  node_modules
  public
  resources
    css
    js
    sass
  views
    auth
    layouts
    products
    roles
    users
      create.blade.php
      edit.blade.php
      index.blade.php
      show.blade.php
      home.blade.php
      welcome.blade.php
    routes
    storage
    tests
    vendor
    .editorconfig
    .env
    .env.example
    .gitattributes
  PROBLEMS 3 OUTPUT DEBUG CONSOLE TERMINAL
1 @extends('layouts.app')
2
3 @section('content')
4 <!-- Container for the main content -->
5 <div class="row">
6   <div class="col-lg-12 margin-tb">
7     <!-- Section for the page title -->
8     <div class="pull-left">
9       <h2>Users Management</h2>
10    </div>
11    <!-- Button to navigate to the user creation form -->
12    <div class="pull-right">
13      <a class="btn btn-success mb-2" href="{{ route('users.create') }}">
14        <i class="fa fa-plus"></i> Create New User
15      </a>
16    </div>
17  </div>
18 </div>
19
20 <!-- Display a success message if a session key named 'success' exists -->
21 @session('success')
22   <div class="alert alert-success" role="alert">
23     {{ $value }} <!-- Displays the value of the 'success' session -->
```

resources/views/users/show.blade.php



```
File Edit Selection View Go Run Terminal Help ← → ⌂ finalpermission2 EXPLORER FINALPERMISSION2 app bootstrap config database node_modules public resources css js sass views auth layouts products roles users create.blade.php edit.blade.php index.blade.php show.blade.php resources > views > users > show.blade.php > ... 1 @extends('layouts.app') 2 3 @section('content') 4 <!-- Main container for displaying user details --> 5 <div class="row"> 6     <div class="col-lg-12 margin-tb"> 7         <!-- Section for the page title --> 8         <div class="pull-left"> 9             <h2>Show User</h2> 10        </div> 11        <!-- Button to navigate back to the user index page --> 12        <div class="pull-right"> 13            <a class="btn btn-primary" href="{{ route('users.index') }}> Back</a> 14        </div> 15    </div> 16 </div> 17 18 <!-- Section to display the details of the selected user --> 19 <div class="row"> 20     <!-- User's Name --> 21     <div class="col-xs-12 col-sm-12 col-md-12"> 22         <div class="form-group"> 23             <strong>Name:</strong>
```

For Product and User Roles

The screenshot shows a Microsoft Visual Studio Code (VS Code) interface with the following details:

- File Explorer (Left):** Shows the project structure under the folder "FINALPERMISSION2". The "views" folder contains subfolders for "products", "roles", and "users", each with their respective blade files (e.g., "create.blade.php").
- Code Editor (Center):** Displays the contents of the "create.blade.php" file located in the "products" view folder. The code is written in Blade templating language, which is a superset of PHP.
- Search Bar (Top):** Contains the text "finalpermission2".
- Terminal (Bottom):** Shows the command "laravel serve" being run, indicating the application is running locally.

```
resources > views > products > create.blade.php > p.text-center.text-primary > small
1 @extends('layouts.app')
2
3 @section('content')
4 <!-- Main container for the "Add New Product" page -->
5 <div class="row">
6     <div class="col-lg-12 margin-tb">
7         <!-- Left-aligned header displaying the page title -->
8         <div class="pull-left">
9             <h2>Add New Product</h2>
10        </div>
11        <!-- Right-aligned "Back" button to navigate back to the products list -->
12        <div class="pull-right">
13            <a class="btn btn-primary btn-sm" href="{{ route('products.index') }}"><i </a>
14        </div>
15    </div>
16 </div>
17
18 <!-- Display validation error messages if there are any -->
19 @if ($errors->any())
20     <div class="alert alert-danger">
21         <strong>Whoops!</strong> There were some problems with your input.<br><br>
22         <ul>
23             <!-- Loop through each error message and display it as a list item -->
```

Step 10 (Creating Seeder)

Step 10: Create Seeder For Permissions and AdminUser

In this step, we will create a seeder for permissions. Right now, we have fixed permissions, so we'll create them using a seeder as listed below. However, if you want, you can add more permissions as you wish.

1.role-list

2.role-create

3.role-edit

4.role-delete

5.product-list

6.product-create

7.product-edit

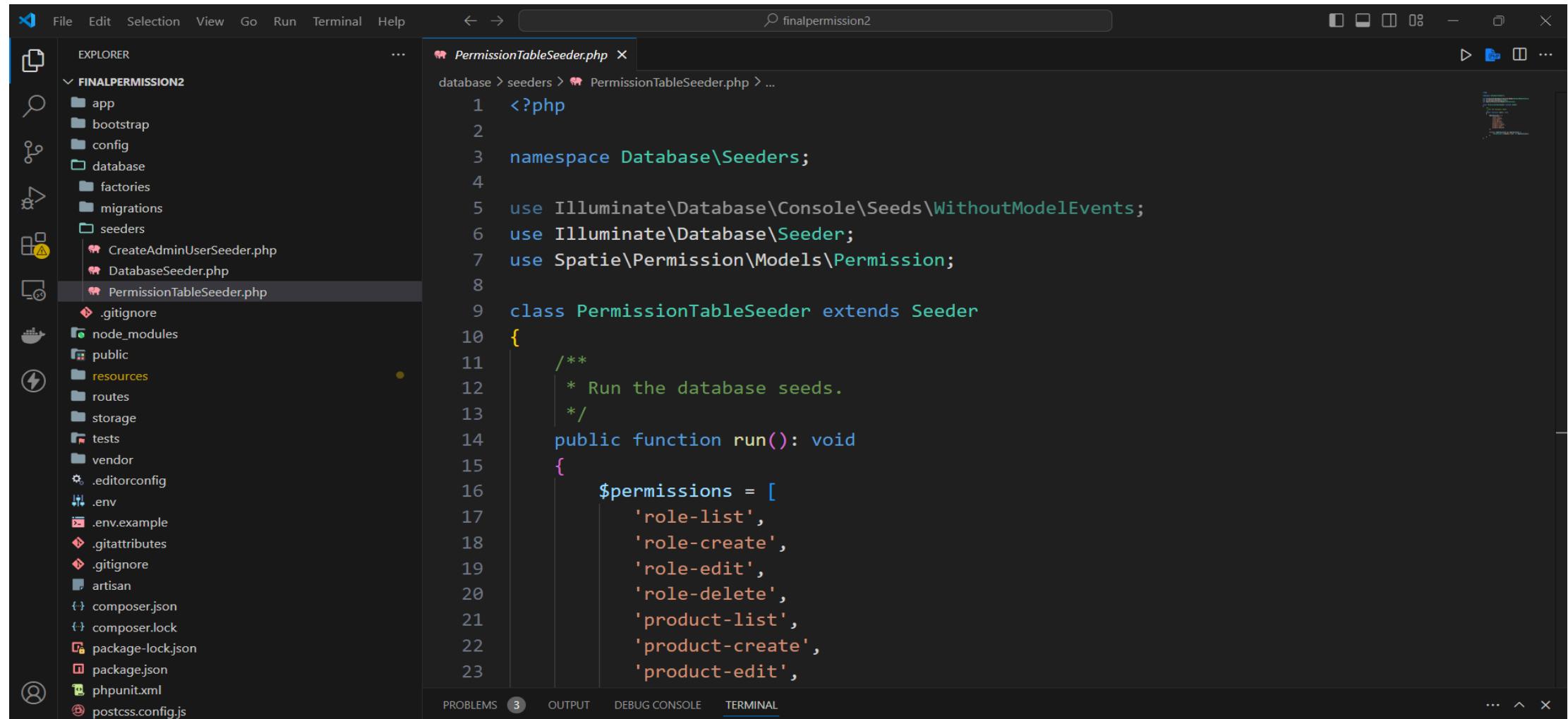
8.product-delete

So, first create a seeder using the below command:

```
php artisan make:seeder PermissionTableSeeder
```

And put bellow code in PermissionTableSeeder seeder this way:

database/seeders/PermissionTableSeeder.php



The screenshot shows a code editor interface with a dark theme. On the left is the Explorer sidebar, which lists the project structure of 'FINALPERMISSION2'. The 'seeders' folder contains three files: CreateAdminUserSeeder.php, DatabaseSeeder.php, and the currently selected file, PermissionTableSeeder.php. The main editor area displays the PHP code for the PermissionTableSeeder.php file:

```
<?php  
namespace Database\Seeders;  
  
use Illuminate\Database\Console\Seeds\WithoutModelEvents;  
use Illuminate\Database\Seeder;  
use Spatie\Permission\Models\Permission;  
  
class PermissionTableSeeder extends Seeder  
{  
    /**  
     * Run the database seeds.  
     */  
    public function run(): void  
    {  
        $permissions = [  
            'role-list',  
            'role-create',  
            'role-edit',  
            'role-delete',  
            'product-list',  
            'product-create',  
            'product-edit',  
        ];  
    }  
}
```

The code defines a class 'PermissionTableSeeder' that extends the 'Seeder' class from the Illuminate\Database\Console\Seeds namespace. It uses the 'WithoutModelEvents' trait and the 'Spatie\Permission\Models\Permission' model. The 'run' method creates an array of permission names: 'role-list', 'role-create', 'role-edit', 'role-delete', 'product-list', 'product-create', and 'product-edit'.

Cont. ...

After this, we have to run the below command to execute the PermissionTableSeeder seeder:

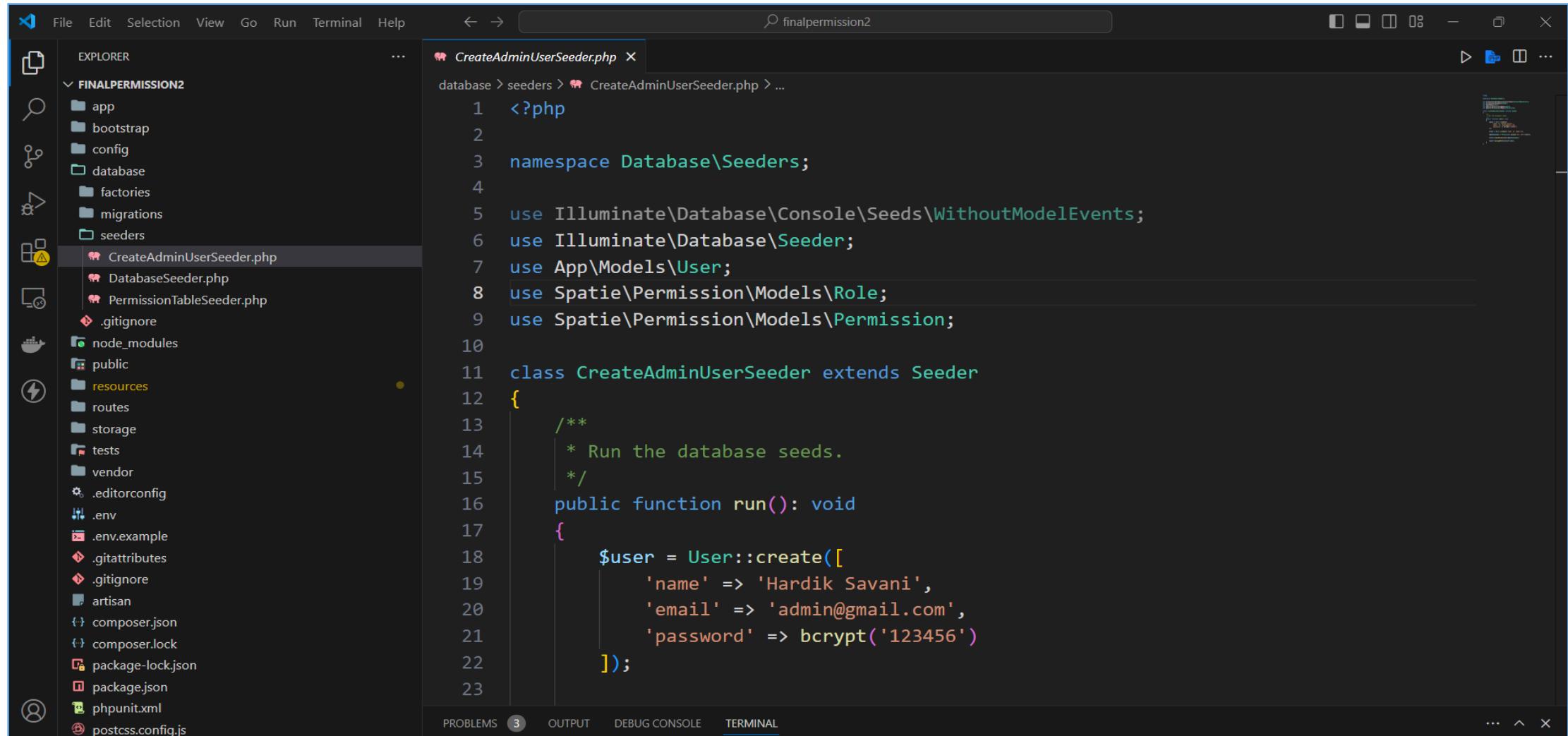
```
php artisan db:seed --class=PermissionTableSeeder
```

Now let's create a new seeder for creating an admin user.

```
php artisan make:seeder CreateAdminUserSeeder
```

database/seeders/CreateAdminUserSeeder.php

database/seeders/CreateAdminUserSeeder.php



The screenshot shows a dark-themed interface of the Visual Studio Code code editor. On the left is the Explorer sidebar, which lists the project structure under 'FINALPERMISSION2'. The 'seeders' folder contains three files: 'CreateAdminUserSeeder.php' (which is currently selected), 'DatabaseSeeder.php', and 'PermissionTableSeeder.php'. Other visible files include 'app', 'bootstrap', 'config', 'database', 'factories', 'migrations', '.gitignore', 'node_modules', 'public', 'resources', 'routes', 'storage', 'tests', 'vendor', '.editorconfig', '.env', '.env.example', '.gitattributes', '.gitignore', 'artisan', 'composer.json', 'composer.lock', 'package-lock.json', 'package.json', 'phpunit.xml', and 'postcss.config.js'. The main editor area displays the PHP code for 'CreateAdminUserSeeder.php'. The code defines a class 'CreateAdminUserSeeder' that extends ' Seeder'. It uses the 'User' model to create a new user with the name 'Hardik Savani', email 'admin@gmail.com', and password hashed with bcrypt('123456'). The code is well-formatted with color-coded syntax highlighting.

```
<?php  
namespace Database\Seeders;  
  
use Illuminate\Database\Console\Seeds\WithoutModelEvents;  
use Illuminate\Database\Seeder;  
use App\Models\User;  
use Spatie\Permission\Models\Role;  
use Spatie\Permission\Models\Permission;  
  
class CreateAdminUserSeeder extends Seeder  
{  
    /**  
     * Run the database seeds.  
     */  
    public function run(): void  
    {  
        $user = User::create([  
            'name' => 'Hardik Savani',  
            'email' => 'admin@gmail.com',  
            'password' => bcrypt('123456')  
        ]);
```

Cont. ...

Then, run seeder using the following command:

```
php artisan db:seed --class=CreateAdminUserSeeder
```

Run Laravel App:

All the required steps have been done, now you have to type the given below command and hit enter to run the Laravel app:

```
php artisan serve
```

Now, Go to your web browser, type the given URL and view the app output:

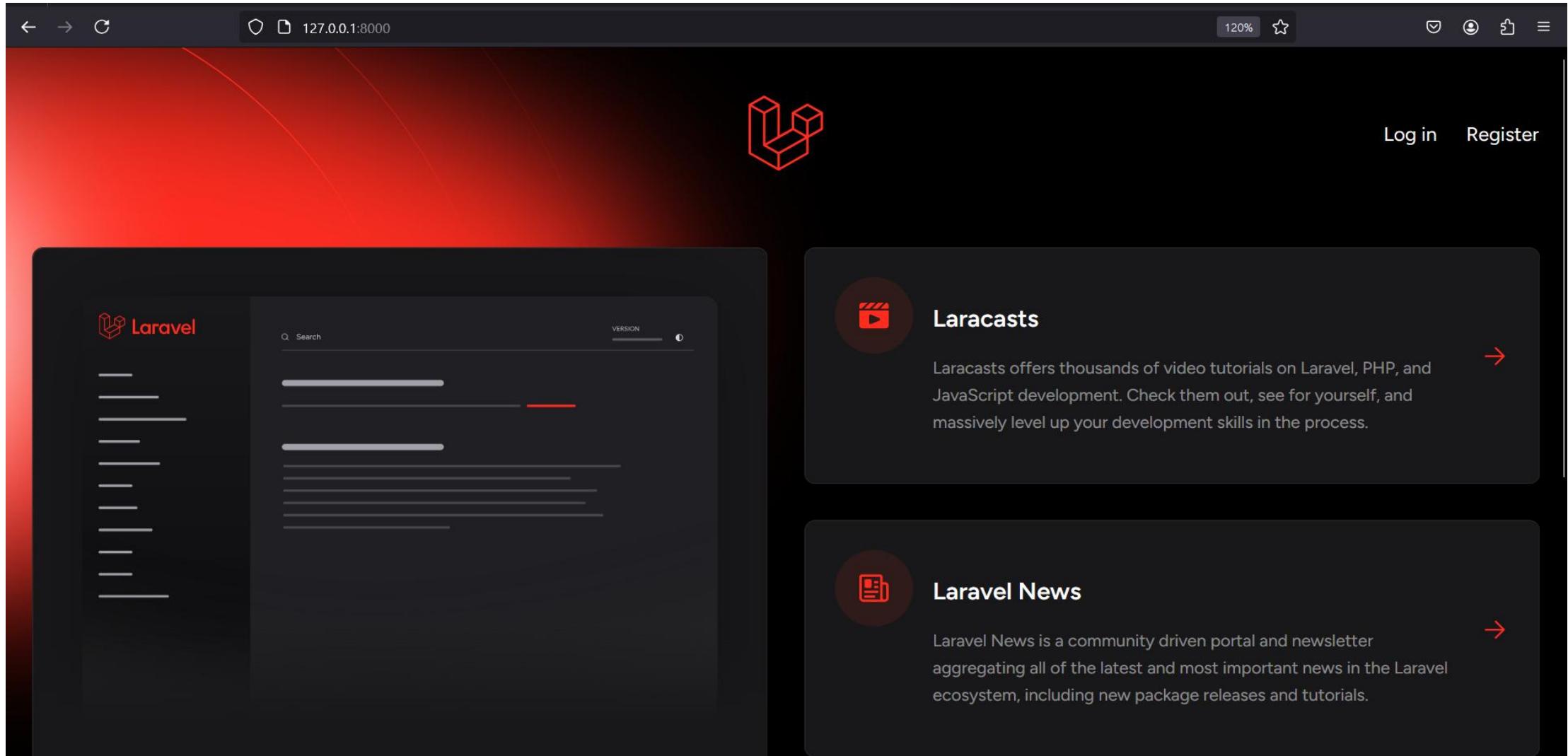
```
http://localhost:8000
```

Now you can login with following credential:

```
Email: admin@gmail.com  
Password: 123456
```

Now you can run and check.

Run The Application (<http://127.0.0.1:8000/>)



http://127.0.0.1:8000/login

The screenshot shows a web browser window with the URL `127.0.0.1:8000/login` in the address bar. The page title is "Laravel 11 User Roles and Permissions". On the right side of the header, there are "Login" and "Register" links. The main content area contains a "Login" form. The form has fields for "Email Address" and "Password", both represented by input boxes. Below these fields is a "Remember Me" checkbox. At the bottom of the form are two buttons: a blue "Login" button and a link "Forgot Your Password?".

Laravel 11 User Roles and Permissions

Login Register

Login

Email Address

Password

Remember Me

[Login](#) [Forgot Your Password?](#)

http://127.0.0.1:8000/login

The screenshot shows a web browser window with the URL `127.0.0.1:8000/login` in the address bar. The page title is "Laravel 11 User Roles and Permissions". On the right, there are "Login" and "Register" links. The main content is a "Login" form. It has fields for "Email Address" containing `admin@gmail.com` and "Password" containing five masked dots. There is a "Remember Me" checkbox and a "Login" button in a blue box. A link for "Forgot Your Password?" is also present.

Laravel 11 User Roles and Permissions

Login Register

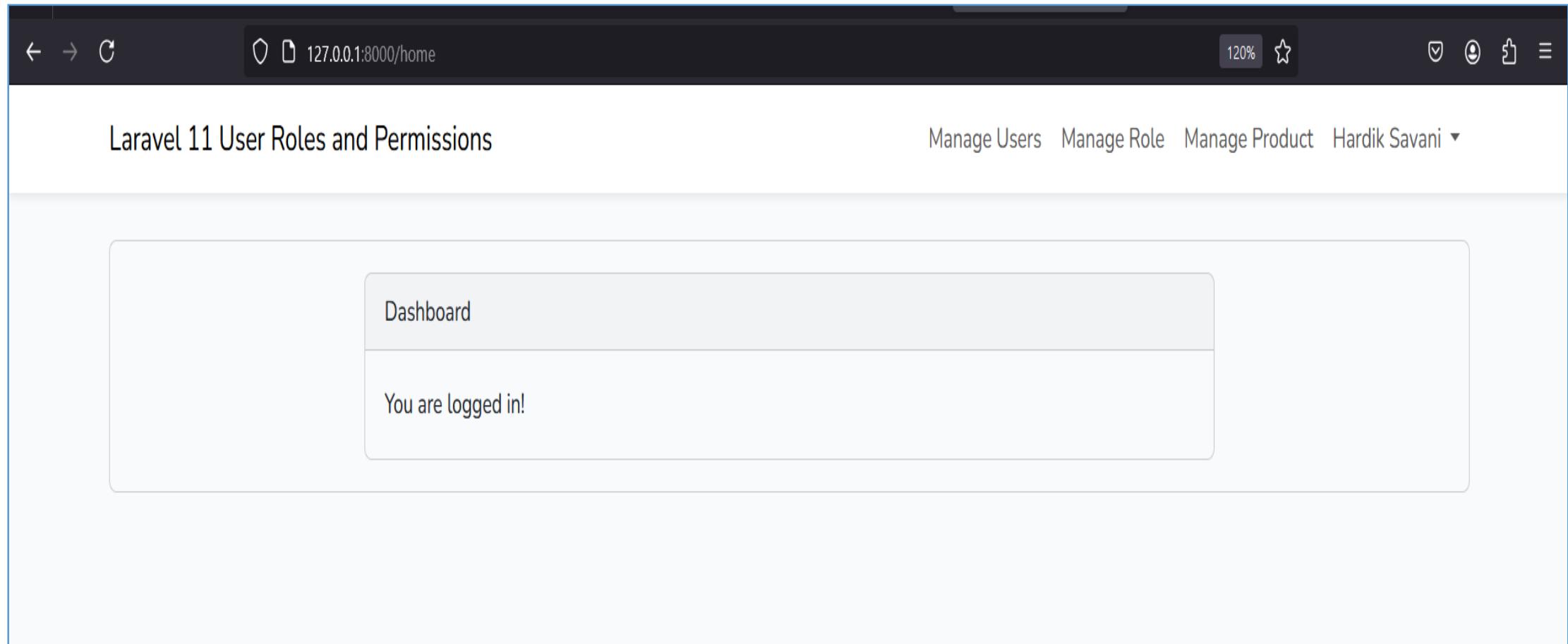
Email Address admin@gmail.com

Password •••••

Remember Me

Login [Forgot Your Password?](#)

http://127.0.0.1:8000/home



http://127.0.0.1:8000/users

The screenshot shows a web browser window with the URL `127.0.0.1:8000/users` in the address bar. The page title is "Laravel 11 User Roles and Permissions". The main content area is titled "Users Management" and features a green button labeled "+ Create New User". Below this is a table with two rows of user data:

No	Name	Email	Roles	Action
1	yitayew	yitayewsolomon3@gmail.com	Admin	Show Edit Delete
2	Hardik Savani	admin@gmail.com	Admin	Show Edit Delete

At the bottom of the page, the text "Practice Makes You Perfect" is displayed.

http://127.0.0.1:8000/users/create

The screenshot shows a web browser window with the URL `127.0.0.1:8000/users/create` in the address bar. The page title is "Laravel 11 User Roles and Permissions". The top navigation bar includes links for "Manage Users", "Manage Role", "Manage Product", and a dropdown for "Hardik Savani". The main content area is titled "Create New User" and contains the following fields:

- Name: dumyuser
- Email: user@gmail.com
- Password: *****
- Confirm Password: *****
- Role: Admin

A blue "Submit" button is located at the bottom right of the form.

http://127.0.0.1:8000/roles

Laravel 11 User Roles and Permissions

Manage Users Manage Role Manage Product Hardik Savani ▾

Role Management

+ Create New Role

No	Name	Action
1	supper Admin	Show Edit Delete
2	Admin	Show Edit Delete

Practice Makes You Perfect

http://127.0.0.1:8000/roles/create

The screenshot shows a web browser window with the URL `127.0.0.1:8000/roles/create` in the address bar. The page title is "Laravel 11 User Roles and Permissions". The top navigation bar includes links for "Manage Users", "Manage Role", "Manage Product", and a user profile "Hardik Savani". The main content area is titled "Create New Role" and contains a "Back" button. It has two input fields: "Name" (placeholder "Name") and "Permission" (checkboxes for various role permissions). A "Submit" button is at the bottom right, and a footer message "Practice Makes You Perfect" is at the bottom center.

Laravel 11 User Roles and Permissions

Manage Users Manage Role Manage Product Hardik Savani ▾

Create New Role

[← Back](#)

Name:

Permission:

role-list
 role-create
 role-edit
 role-delete
 product-list
 product-create
 product-edit
 product-delete

[Submit](#)

Practice Makes You Perfect

http://127.0.0.1:8000/products/create

The screenshot shows a web browser window with the URL `127.0.0.1:8000/products/create` in the address bar. The page title is "Laravel 11 User Roles and Permissions". In the top right, there are links for "Manage Users", "Manage Role", "Manage Product", and a dropdown for "Hardik Savani". The main content area has a heading "Add New Product" and a "Back" button. It contains two input fields: one for "Name" and one for "Detail". A "Submit" button is at the bottom. A footer message "Practice Makes You Perfect" is visible.

Laravel 11 User Roles and Permissions

Manage Users Manage Role Manage Product Hardik Savani

Add New Product

[← Back](#)

Name:
Name

Detail:
Detail

Submit

Practice Makes You Perfect

http://127.0.0.1:8000/products

The screenshot shows a web browser window with the URL `127.0.0.1:8000/products` in the address bar. The page title is "Laravel 11 User Roles and Permissions". On the right, there are navigation links: "Manage Users", "Manage Role", "Manage Product", and a dropdown menu for "Hardik Savani". The main content area has a title "Products" and a green button "+ Create New Product". Below is a table with one row:

No	Name	Details	Action
1	Omen Laptop	The OMEN brand is a series of high-performance gaming laptops and desktops manufactured by HP (Hewlett-Packard). These devices are specifically designed for gamers and offer features like powerful processors, high refresh rate displays, advanced graphics, and customizable RGB lighting. The OMEN laptops are known for their sleek design, cutting-edge performance, and specialized gaming capabilities.	Show Edit Delete

At the bottom center, there is a footer message: "Practice Makes You Perfect".

Database Tables

			id	name	email	email_verified_at	password	remember_token
<input type="checkbox"/>	 Edit	 Copy	 Delete	1	Hardik Savani	admin@gmail.com	NULL	\$2y\$12\$h/bMZixuwWlw6AGnrvPip.XKtlaq2FoHG54KC92hxM7... NULL
<input type="checkbox"/>	 Edit	 Copy	 Delete	2	yitayew	yitayewsolomon3@gmail.com	NULL	\$2y\$12\$G6Qb8anJ0Zf1uH5565drnumrhcdH8HXJ6HH5oGPKkpA... NULL
<input type="checkbox"/>	 Edit	 Copy	 Delete	3	dumyuser	user@gmail.com	NULL	\$2y\$12\$HXUJ9auzkbaKWzvdm/ft7ONAssegDN3cD1UfHD2B.HrH... NULL

			id	name	guard_name	created_at	updated_at	
<input type="checkbox"/>	 Edit	 Copy	 Delete	1	Admin	web	2024-11-28 09:31:00	2024-11-28 09:31:00
<input type="checkbox"/>	 Edit	 Copy	 Delete	2	supper Admin	web	2024-11-28 13:25:45	2024-11-28 13:25:45

[Up](#) Check all With selected: [Edit](#) [Copy](#) [Delete](#) [Export](#)

			id	name	detail	created_at	updated_at	
<input type="checkbox"/>	 Edit	 Copy	 Delete	1	Omen Laptop	The OMEN brand is a series of high-performance gam...	2024-11-28 13:27:43	2024-11-28 13:27:43

[Up](#) Check all With selected: [Edit](#) [Copy](#) [Delete](#) [Export](#)

Thank you!

Appreciate your action.