CSE 3100: Web Programming Laboratory

Lab 9: Laravel MiddleWare

# Create a Laravel Project

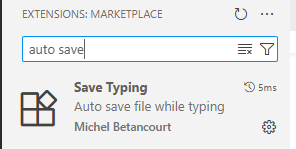
* Open Terminal in your VS Code . Shortcut: *Ctrl+`*  or View>Terminal
* Go to the htdocs folder in the terminal : ***cd C:\xampp\htdocs***
* Create a Laravel project: *composer create-project laravel/laravel test-app* **
* Open the project in VS Code:

***cd test-app***

***code .***

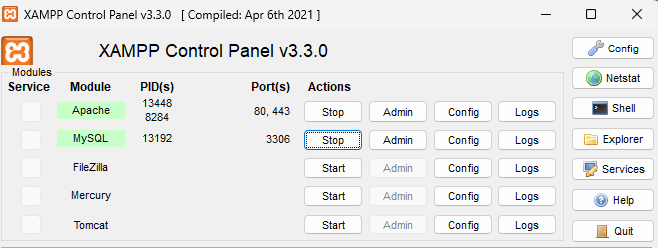
**

* Start Laravel built-in server: ***php artisan serve***
* Suggestion: Add any autosave extension in your VS code project



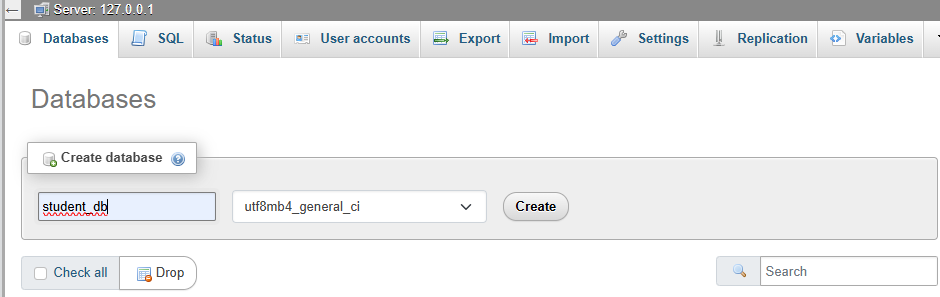
# Apache & MySQL

First open your XAMPP Control Panel and turn MySQL services on along with the Apache server.



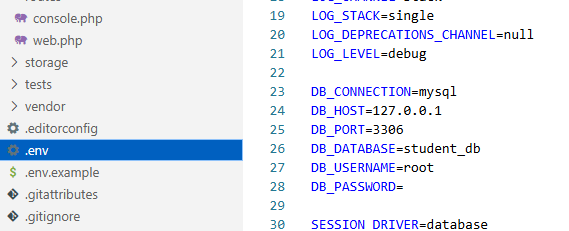
# Create Database

Paste *http://localhost/phpmyadmin* URL in your browser to open phpMyAdmin. Create a database with any name i.e. student\_db. We do not need to add any table manually. We will add tables through artisan commands. First we need to connect our database to the project.



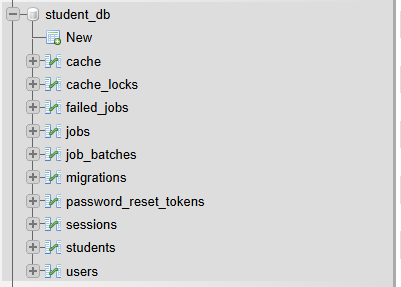
# Connect Laravel App with Database

Now, copy your database name and go back to your project in VS Code and open the .env file . Change the DB\_CONNECTION to your RDBMS name i.e. mysql,sqlite,mongodb. Then comment out the other variables and change DB\_DATABASE to your created database’s name. Then create a new terminal in VS Code and write the following artisan command: *php artisan migrate*





If you go to the database in phpmyadmin, you will find some newly created tables. These are the default tables of Laravel. Now we will create our own table students or today’s lab task.

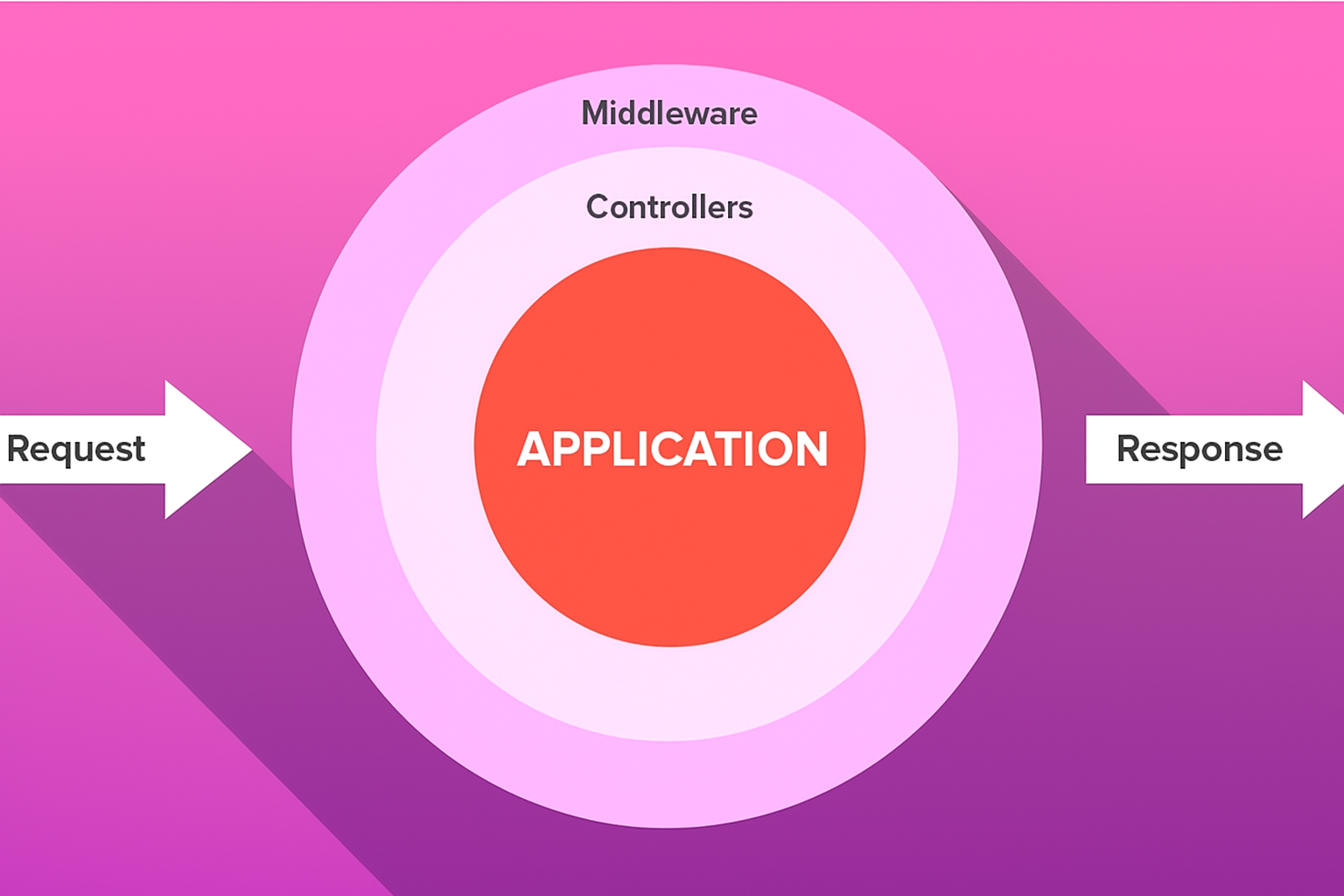


# Laravel Middleware

Laravel middleware provides a mechanism for inspecting and filtering HTTP requests entering a Laravel application. It functions as a series of "layers" that requests pass through before reaching the application's core logic. Each middleware layer can perform specific tasks, such as **authentication, logging, or CSRF protection**, and can either allow the request to proceed or redirect/terminate it based on defined conditions. Its sole purpose is to perform actions on HTTP requests before they reach the application's routes or controllers, or on responses before they are sent back to the client.

The conditions can be-

* Authentication and authorization: Whether one is an authenticated user or not
* Role or permission-based access control: Is Admin or not
* Has ValidAge or not
* Logging user activity



# Types of Middleware

1. Route Middleware: Applied to specific routes. Condition checking on individual routes
2. Global Middleware: Applied to every request. Condition checking on all routes
3. Middleware Groups: Group multiple middleware under one key. Multiple conditions will be checked or a single route

# 

# 1. Resource Middleware:

By default, Laravel’s users table only stores basic information such as *id*, *name, email, and password*. We need another column for role

Add new column to users table:

1. Add new column named “role” ->

***php artisan make:migration add\_role\_to\_users\_table --table=users***

1. Open the file (something like ***xxxx\_xx\_xx\_xxxxxx\_add\_role\_to\_users\_table.php***) and add your new column inside the schema.

public function up(): void

{

Schema::table('users', function (Blueprint $table) {

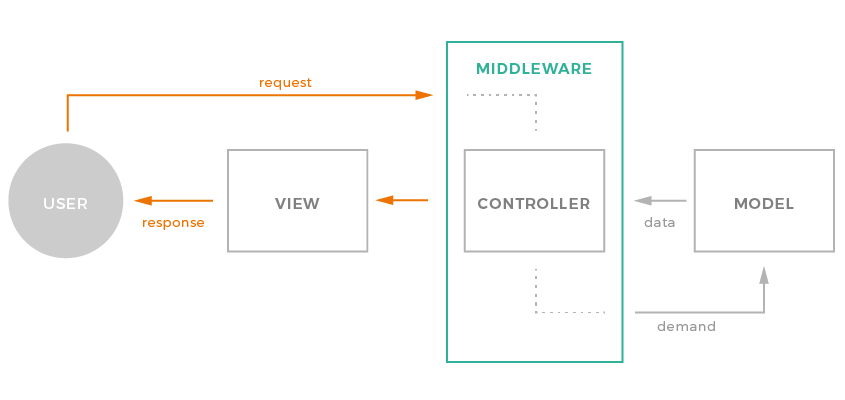
//

$table->string('role')->default('user');

});

}

1. Run the Migration using ***php artisan migrate***



Create Middleware

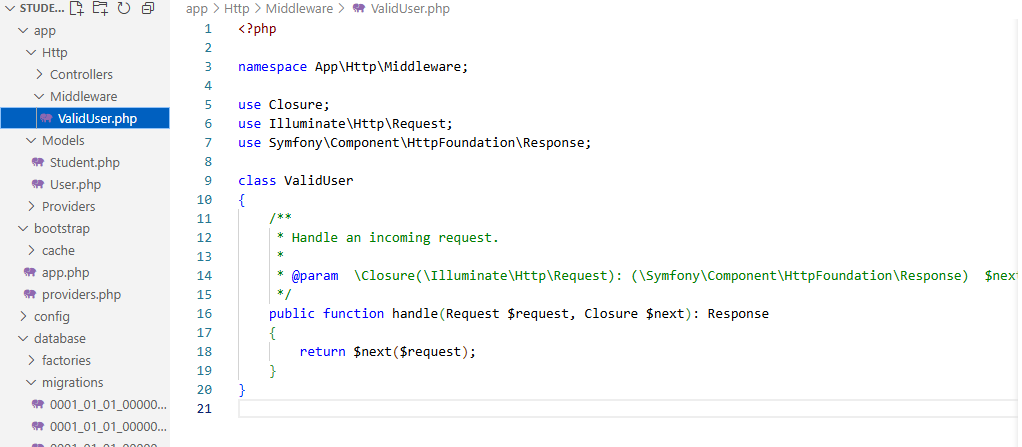
1. A middleware can be generated using the Artisan command

*php artisan make:middleware <MiddlewareName>*

Then you can see your newly created middleware classes in the *app/Http/Middleware* directory.

Example:

*php artisan make:middleware ValidUser***The following file will be created**

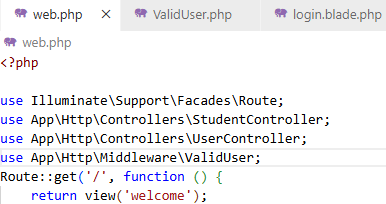


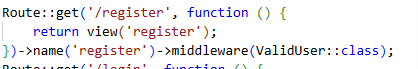
Write simple echo “ValidUser Middleware has been called” to check the middleware

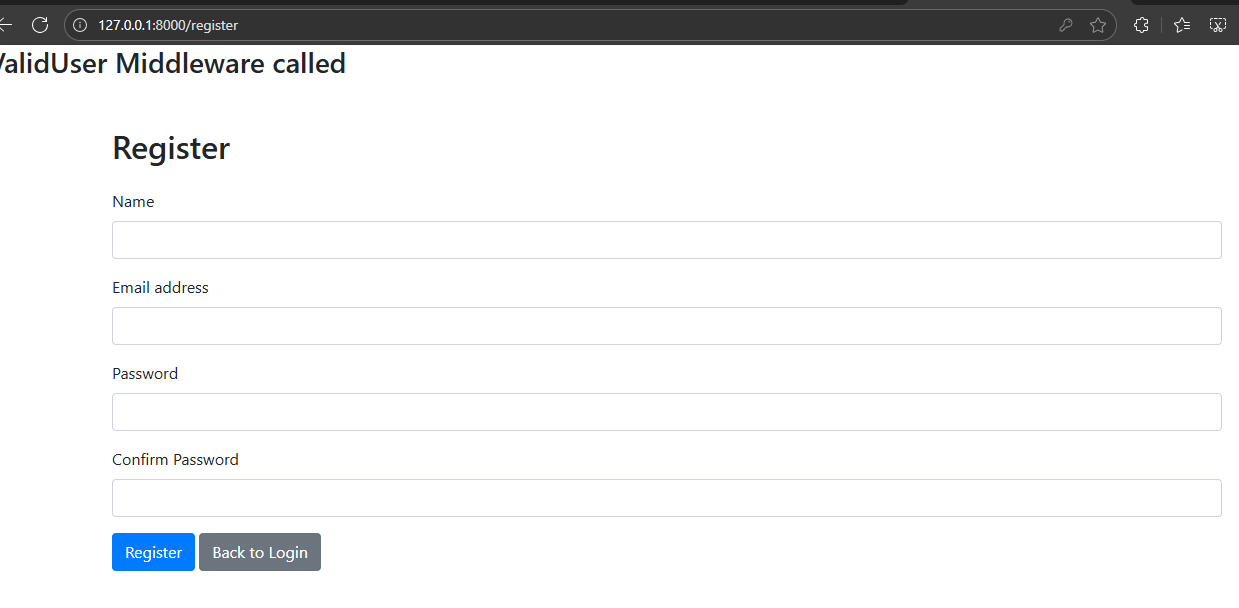
*echo "<h3>ValidUser Middleware called</h3>";*

1. Assigning middlewares to routes: <https://laravel.com/docs/12.x/middleware#assigning-middleware-to-routes>

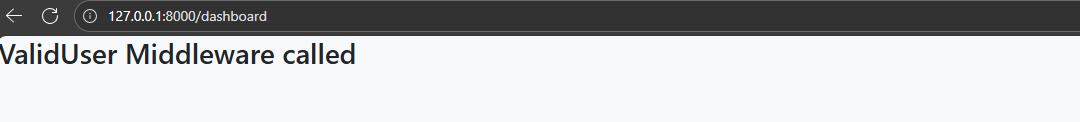
Now go to the web.php file to mention the routes. Firstly, we need to import the middleware in our web.php and then write the route







Now do the same or dashboard page so that when the page loads following message shows:



# Middleware with Authentication Check

1. Modify the handle function of ValidUser.php middleware

Go to App\Http\Middleware\ValidUser.php

Import class:- ***use Illuminate\Support\Facades\Auth;***

Do a Authentication check through middleware

public function handle(Request $request, Closure $next): Response

{

echo "<h3>ValidUser Middleware called</h3>";

if (Auth::check()) {

return $next($request);

}

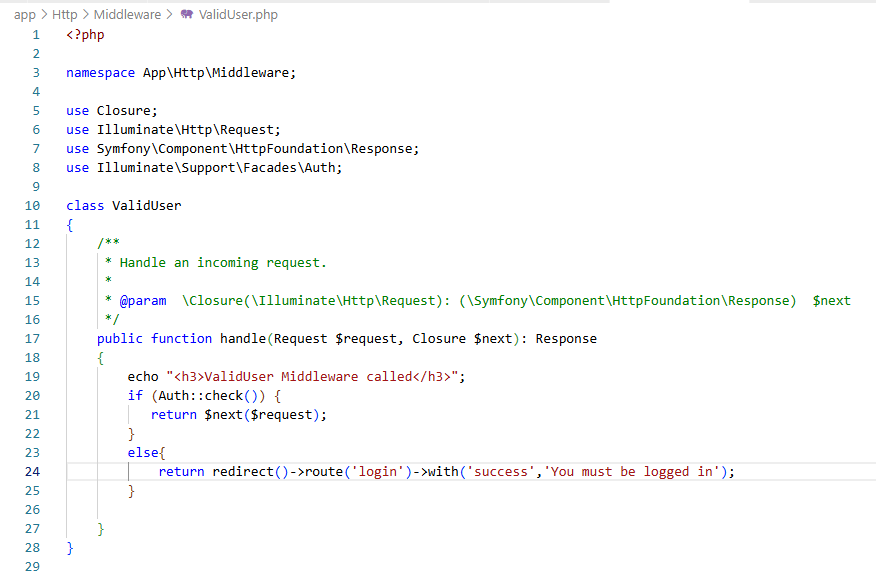
else{

return redirect()->route('login')->with('success','You must be logged in');

}

}





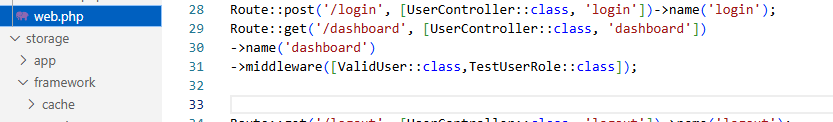
Create Another Middleware

1. Create another middleware named TestUserRole

*php artisan make:middleware TestUserRole*

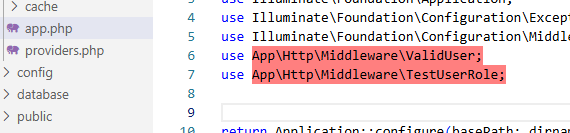
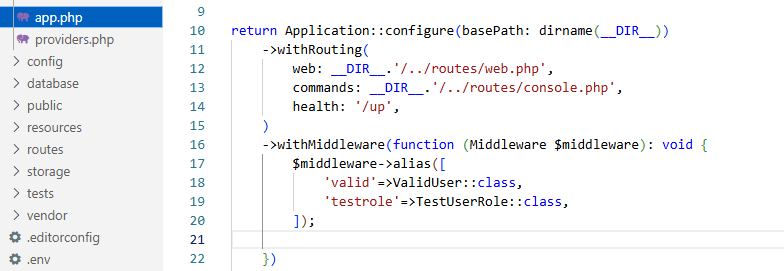
1. We can pass the middlewares by creating an array of middleware classes. Here, we are using long names to address each middleware classes.





Instead of the long names, you can use [Middleware Aliases](https://laravel.com/docs/12.x/middleware#middleware-aliases)

You may assign aliases to middleware in your application's *bootstrap/app.php file*. Middleware aliases allow you to define a short alias for a given middleware class, which can be especially useful for middleware with long class names. To do this-

1. Go to *bootstrap/app.php file*
2. First import your middleware classes 
3. Then inside *withMiddleware(function (Middleware $middleware): void {})* write your aliases
4. Now we can modify our routes/web.php

# 2. [Middleware Groups:](https://laravel.com/docs/12.x/middleware#middleware-groups)

Sometimes you may want to group several middleware under a single key to make them easier to assign to routes. You may accomplish this using the ***appendToGroup method*** within your application's ***bootstrap/app.php*** file:

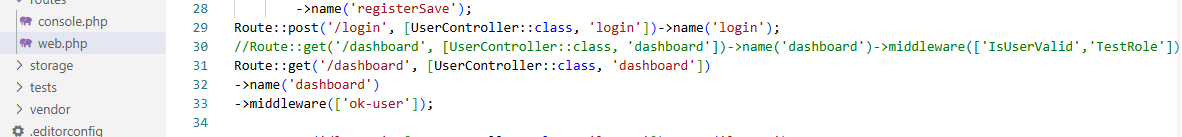
1. Go to ***bootstrap/app.php*** file:
2. inside *withMiddleware(function (Middleware $middleware): void {})* where we described alias, now we will define the middleware groups using appendToGroup, give the group any name

$middleware->appendToGroup('ok-user', [

ValidUser::class,

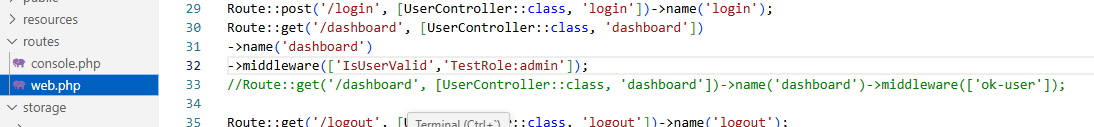
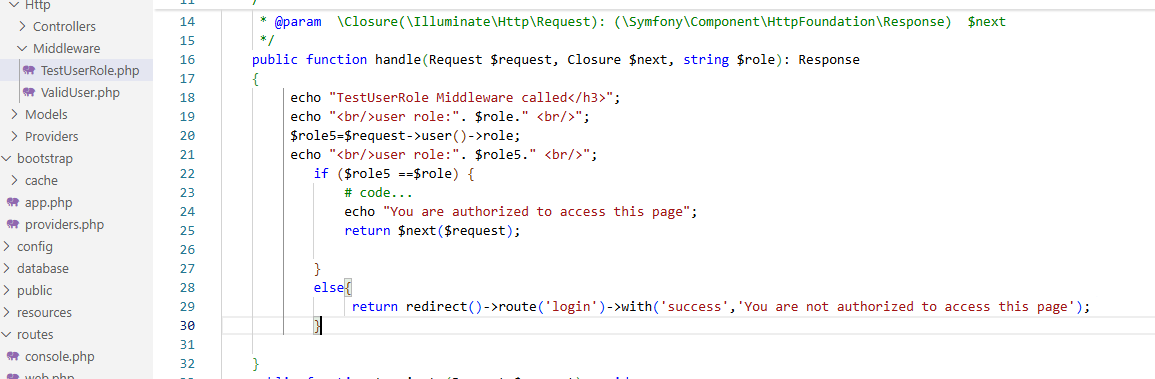
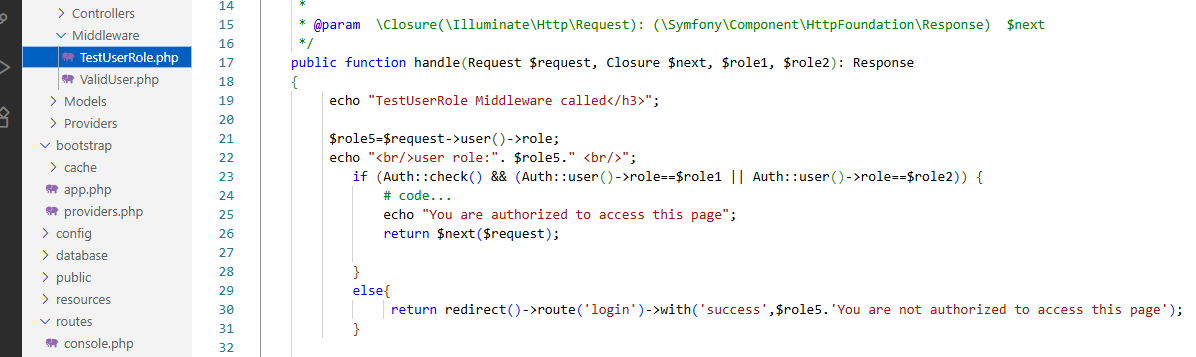
TestUserRole::class,

]);

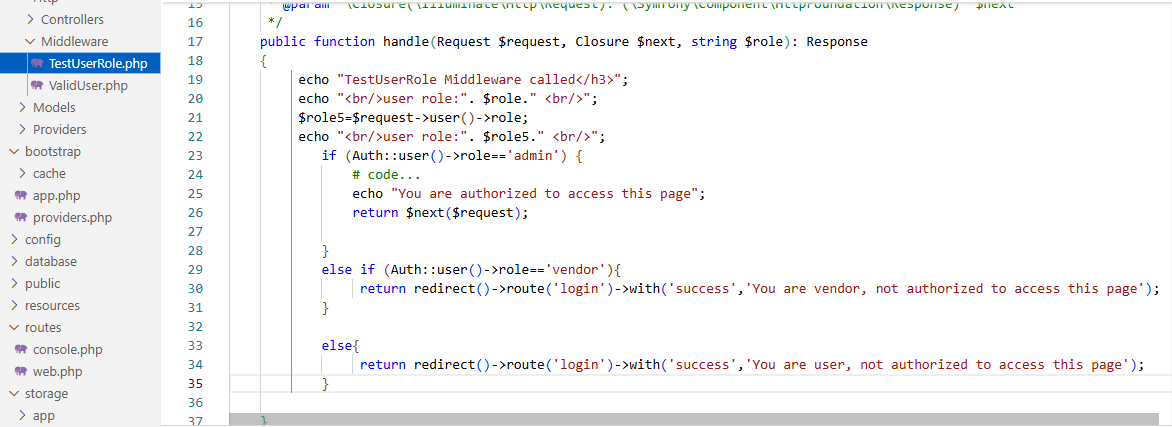
1. Now, assign middleware group “ok-user” to the route

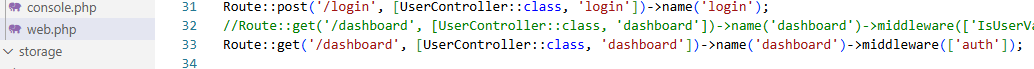
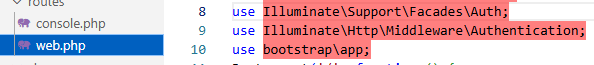
# [Validate Role via Middleware](https://laravel.com/docs/12.x/middleware#middleware-parameters)

Middleware can also receive additional parameters. For example, if your application needs to verify that the authenticated user has a given "role" before performing a given action, you could create a TestUserRole middleware that receives a role name as an additional argument.

1. Go to Middleware\TestUserRole.php. Additional middleware parameters will be passed to the middleware after the *$next argument* and go routes/web.php to update middleware assignment including the roleYou can **multiple roles** in the route in following mannerAlso update the Middleware\TestUserRole
2. We can also use Auth.user. Lets import the class first

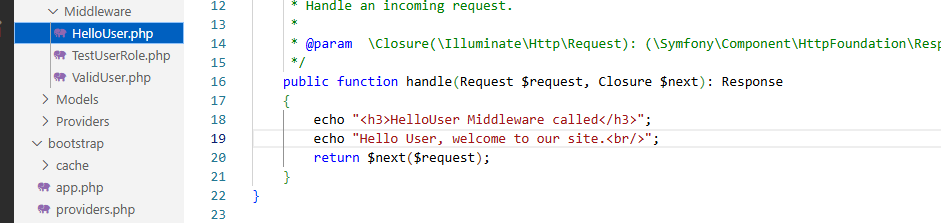
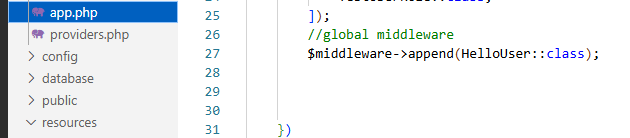
***use Illuminate\Support\Facades\Auth;***

****For convenience, some of Laravel's built-in middleware are aliased by default. For example, the auth middleware is an alias for the **Illuminate\Auth\Middleware\Authenticate middleware**. It can be used to retrieve all the authenticated users instead of using check function all the time

****

To learn more about it: <https://laravel.com/docs/12.x/middleware#middleware-aliases>

# 3.[Global Middleware](https://laravel.com/docs/12.x/middleware#global-middleware)

1. You may using already existing middleware or create any new middleware, example *php artisan make:middleware HelloUser*
2. Go to ***bootstrap/app.php*** file
3. Or global middleware we will use append function 

php artisan serve and navigate through any page to see the message from global middleware.

# Prerequisites

* PHP version 8.2 or higher
* Composer (Dependency Manager for PHP)
* MySQL
* Laravel Authentication: <https://laravel.com/docs/12.x/authentication#authentication-quickstart>
* Authenticating User: <https://laravel.com/docs/12.x/authentication#authenticating-users>
* Seeding: <https://laravel.com/docs/12.x/seeding>

# Resources:

* <https://medium.com/@zulfikarditya/model-attribute-casting-in-laravel-complete-guide-to-accessors-and-mutators-b8e67e8c00df>
* <https://laravel.com/docs/master/eloquent#updates>
* <https://www.cloudways.com/blog/routing-in-laravel/>
* <https://laravel.com/docs/12.x/routing>
* <https://laravel.com/docs/12.x/controllers>
* <https://laravel.com/docs/master/eloquent#retrieving-models>
* <https://laravel.com/docs/master/eloquent#deleting-models>
* <https://laravel.com/docs/master/queries>
* <https://laravel.com/docs/master/eloquent#mass-assignment>
* <https://laravel.com/docs/12.x/csrf>
* <https://laravel.com/docs/11.x/redirects#redirecting-with-flash-data>
* <https://laravel.com/docs/11.x/redirects#redirecting-with-flashed-session-data>
* <https://laravel.com/docs/12.x/validation#quick-writing-the-validation-logic>
* Repopulating forms:<https://laravel.com/docs/12.x/validation#repopulating-forms>
* Session <https://laravel.com/docs/12.x/session#main-content>
* <https://laravel.com/docs/12.x/authentication#authentication-quickstart>
* <https://laravel.com/docs/12.x/authentication#authenticating-users>
* <https://walkerspider.com/cours/laravel/middlewares/>