UECS3294 ADVANCED WEB APPLICATION DEVELOPMENT

Practical 6 : Laravel Authorization and RBAC

In this lab, we will explore the concept of Authorization in Laravel. We will explore the concept of authenticated user with several roles to imitate the role based access control (RBAC) with the previously created laravelAuth web application. In this case, we will explore allowing admin user to only 'delete' post and author to 'create' and 'update' post.

Authorization through Gates

Gates are most applicable to actions which are not related to any model or resource, such as viewing an administrator dashboard. In the following lab session, we will explore on creating gates for user's authorization.

1. Add Roles to User

Create a database migration to add role to user database table through the Artisan CLI.

```
php artisan make:migration add_role_column_to_users_table
```

Within the migration table, modify as shown in Figure 1 in order to add enum (choices) of three different roles to a user.

Figure 1: Adding role column with three choices of roles to user table.

```
public function up()
    {
        Schema::table('users', function (Blueprint $table) {
            $table->enum('role',  ['user', 'author', 'admin'])->default('user');
        });
    }
```

2. Database Migrate

Migrate the created database migration through the Artisan CLI.

```
php artisan migrate
```

3. Create Post Model

Create a new model called Post using the Artisan CLI.

```
php artisan make:model Post
```

4. Create Post Controller

Create a new model called Post using the Artisan CLI.

```
php artisan make:controller PostController
```

5. Define Gates

Define custom gates for admin and author in relation to actions that both user is allowed to do within app/Providers/AuthServiceProvider.php file as shown in Figure 2.

Figure 2: Define gates to define abilities for different users.

```
public function boot()
    {
        $this->registerPolicies();

        /* define an administrator user role */
        Gate::define('isAdmin', function($user) {
            return $user->role == 'admin';
         });

         /* define an author user role */
         Gate::define('isAuthor', function($user) {
             return $user->role == 'author';
         });

         /* define a user role */
         Gate::define('isUser', function($user) {
             return $user->role == 'user';
         });
    }
```

6. Authorize Actions in Blade Template

Within home.blade.php, authorize the actions or abilities for different user that logged in as shown in Figure 3.

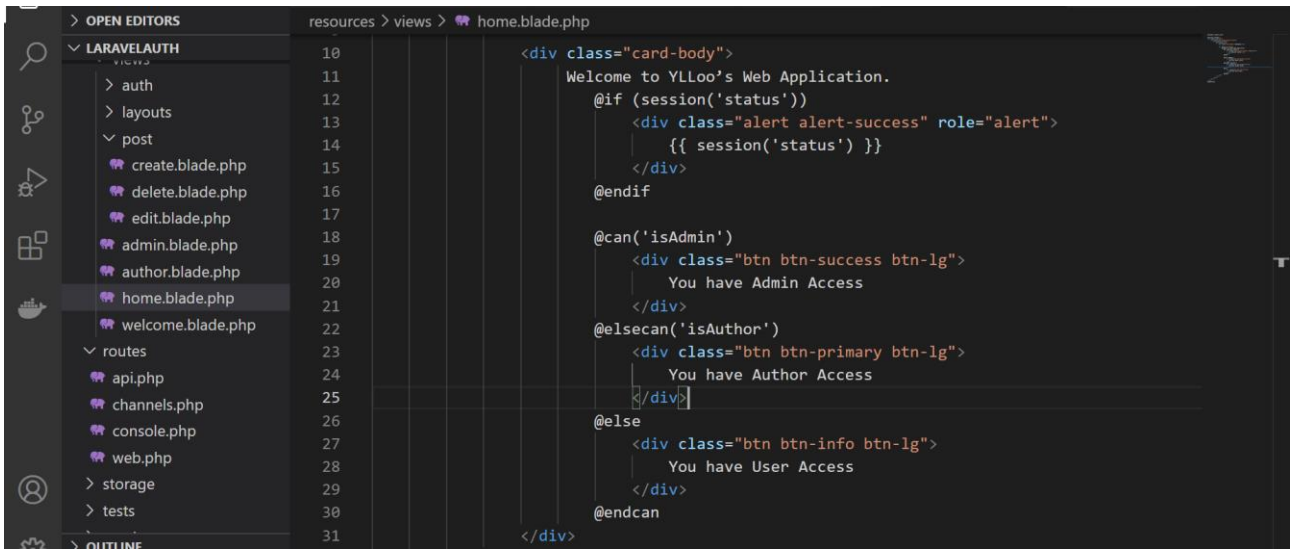

Figure 3: Authorize actions for different users in Blade.

```
<div class="card-body">
                    Welcome to YLLoo's Web Application.
                        @if (session('status'))
                            <div class="alert alert-success" role="alert">
                                {{ session('status') }}
                            </div>
                        @endif

                        @can('isAdmin')
                            <div class="btn btn-success btn-lg">
                                You have Admin Access
                            </div>
                        @elsecan('isAuthor')
                            <div class="btn btn-primary btn-lg">
                                You have Author Access
                            </div>
                        @else
                            <div class="btn btn-info btn-lg">
                                You have User Access
                            </div>
                        @endcan
                </div>
```

7. Create Users Testing Data

Host the web application and try to register a few users using the laravel/ui common user registration form and modify the users into different roles as shown in Figure 4.

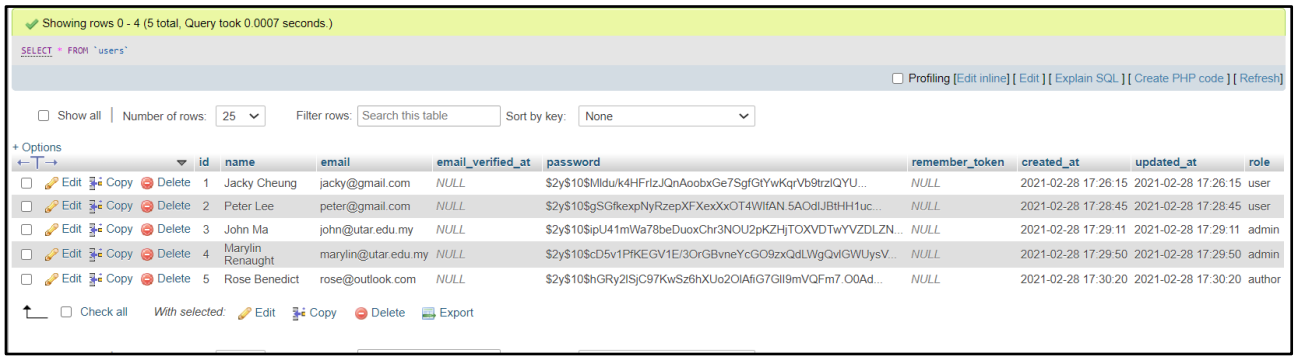

Figure 4: Insert dummy data into users database table.

8. Test the Web Application

Login to see the Gates used in Blade to authorize actions for different users' role as shown in Figure 5.

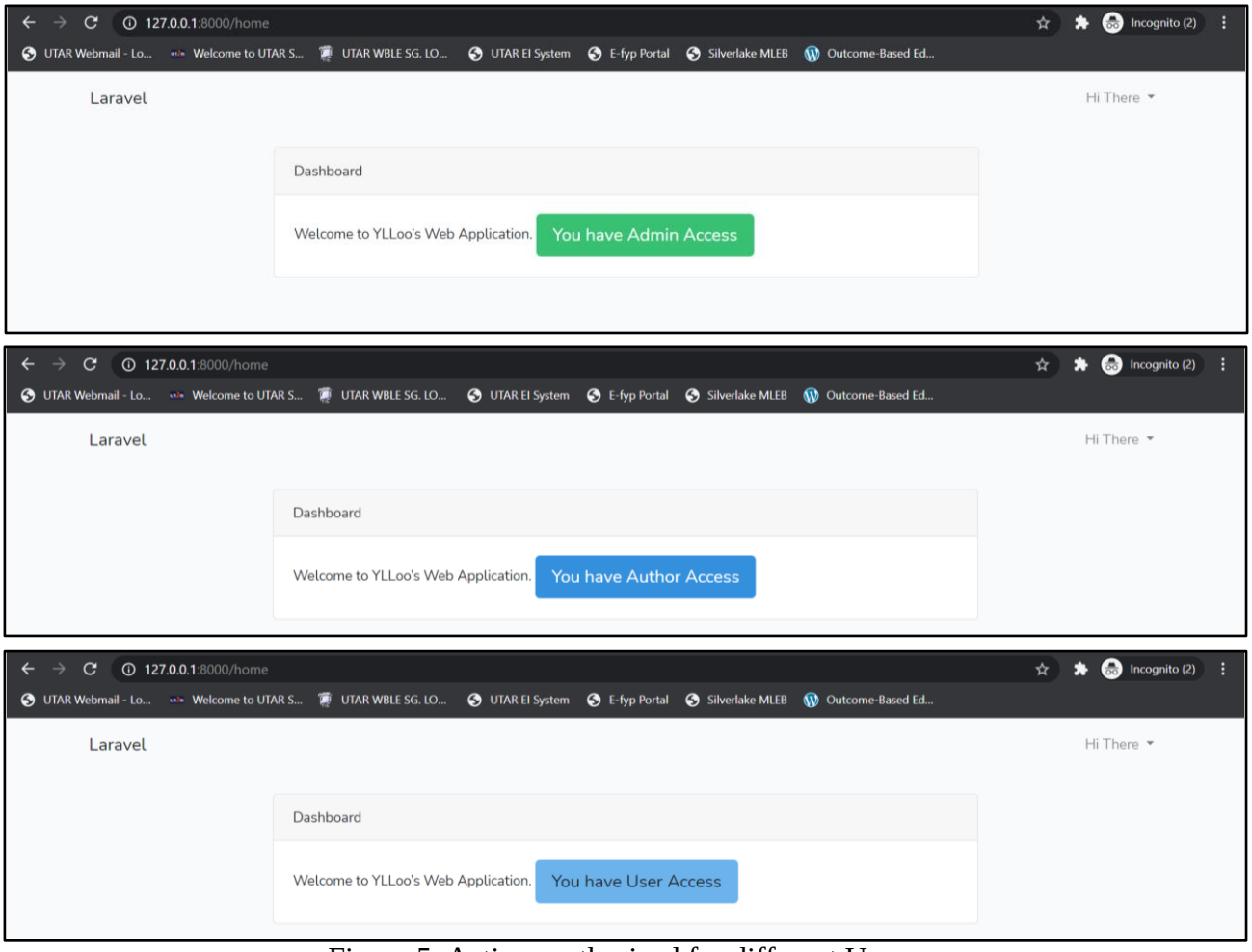

Figure 5: Actions authorized for different Users.

9. Authorize Actions in PostController

User's actions can be authorized through Gates in a Controller as shown in Figure 6. As a testing, let's create three different actions to be "allowed" for different users; "create" and "edit" for authors while "delete" for admins.

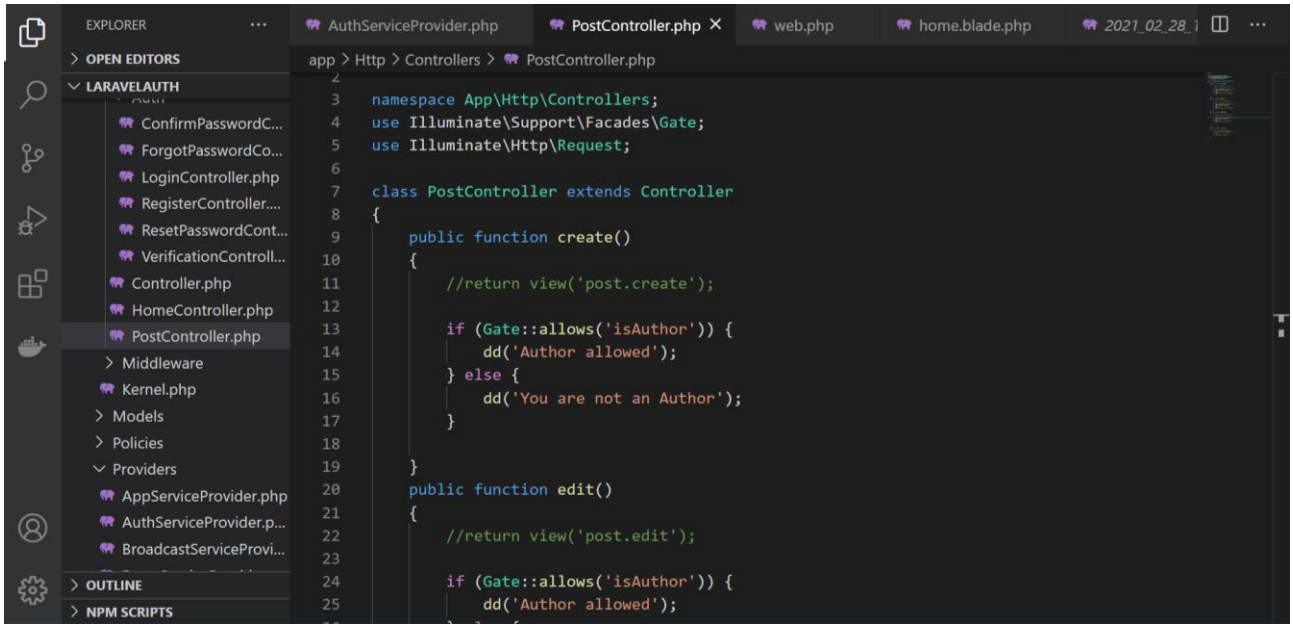

Figure 6: Authorizing actions in Controller.

```php
<?php

namespace App\Http\Controllers;
use Illuminate\Support\Facades\Gate;
use Illuminate\Http\Request;

class PostController extends Controller
{
    public function create()
    {
        if (Gate::allows('isAuthor')) {
            dd('Author allowed');
        } else {
            dd('You are not an Author');
        }

    }
    public function edit()
    {
        if (Gate::allows('isAuthor')) {
            dd('Author allowed');
        } else {
            dd('You are not an Author');
        }
    }
    public function delete()
    {
        if (Gate::allows('isAdmin')) {
            dd('Admin allowed');
        } else {
            dd('You are not Admin');
        }
```

```
    }
}
```

10. Create Routes for Post Controller

In order to test the Gates in Post Controller, create routes for all the controller as shown in Figure 7. Test the routes by accessing them after a login of a user.

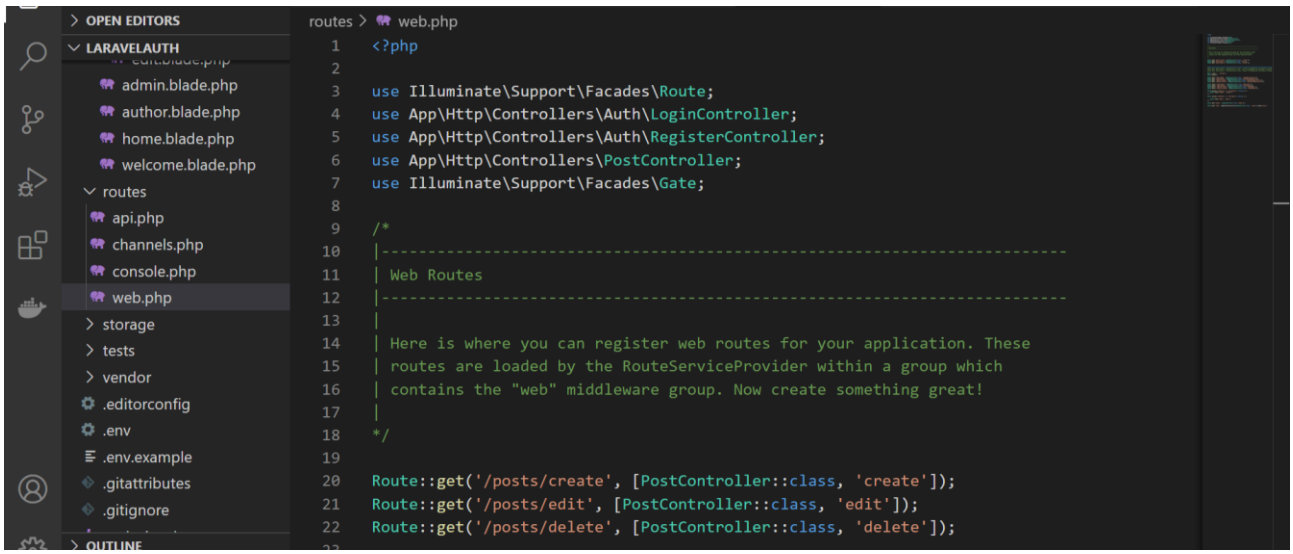

Figure 7: Create routes for all PostController functions.

```
use App\Http\Controllers\PostController;

Route::get('/posts/create', [PostController::class, 'create']);
Route::get('/posts/edit', [PostController::class, 'edit']);
Route::get('/posts/delete', [PostController::class, 'delete']);
```

11. Authorize Actions in Routes

User's actions can be authorized through Gates in routes as shown in Figure 8. After creating the routes, test in order to see the difference between authorizing actions in Controllers and in Routes.

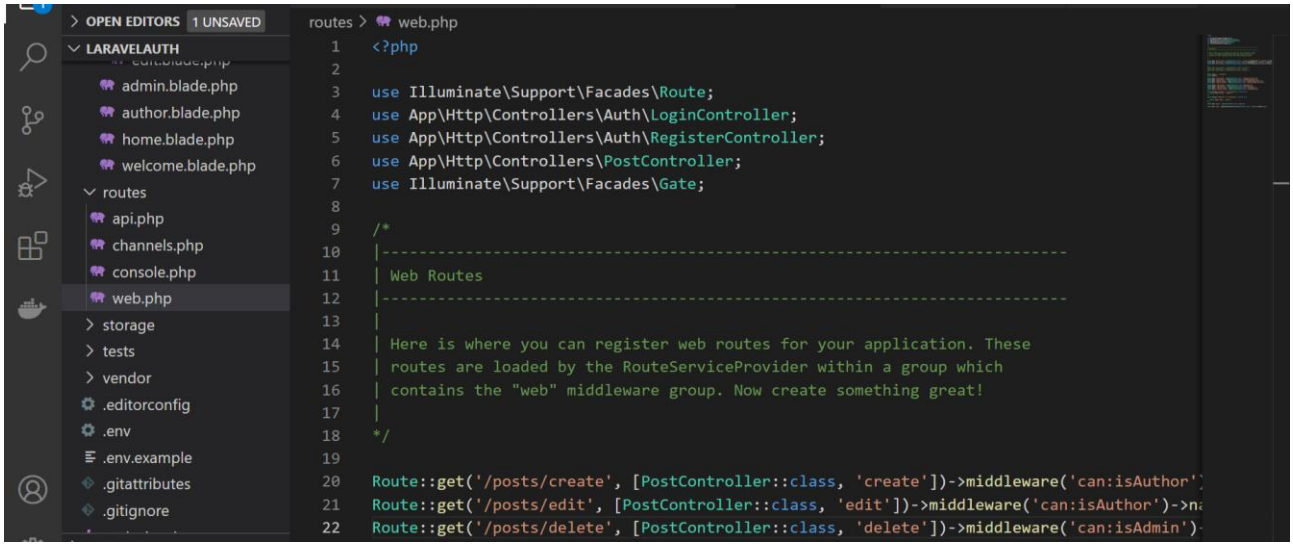

Figure 8: Authorizing actions in Routes.

```
use Illuminate\Support\Facades\Gate;

Route::get('/posts/create',          [PostController::class,          'create'])-
>middleware('can:isAuthor')->name('post.create');
Route::get('/posts/edit',            [PostController::class,            'edit'])-
>middleware('can:isAuthor')->name('post.edit');
Route::get('/posts/delete',          [PostController::class,          'delete'])-
>middleware('can:isAdmin')->name('post.delete');
```

Authorization through Policies

In the following session of this lab, we will explore about policy. Using policy, a model can be authorized.

In Laravel, Policies are classes that organize authorization logic around a particular model or resource. For example, if the web application is a blog, a Post model and a corresponding PostPolicy may be needed to authorize user actions such as creating or updating posts.

1. Create Policy

Create policy using Artisan CLI for Post model.

```
php artisan make:policy PostPolicy --model=Post
```

2. Register the Policies

Policy need to be registered before usage. Policy has to be registered in app/Providers/AuthServiceProvider.php as shown in Figure 9.

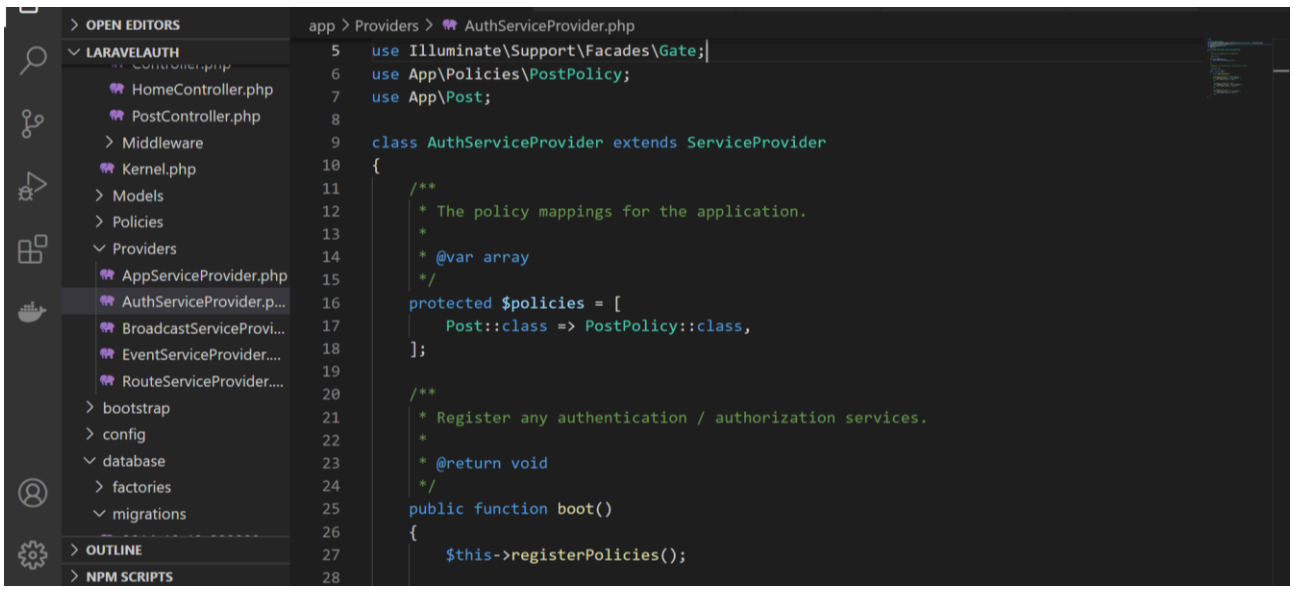

Figure 9: Registering Policy.

```
use App\Policies\PostPolicy;
use App\Post;

protected $policies = [
        Post::class => PostPolicy::class,
    ];
```

3. Define the Policies

Once the policy has been registered, add methods for each action it authorizes in app/Policies/PostPolicy.php file as shown in Figure 10.

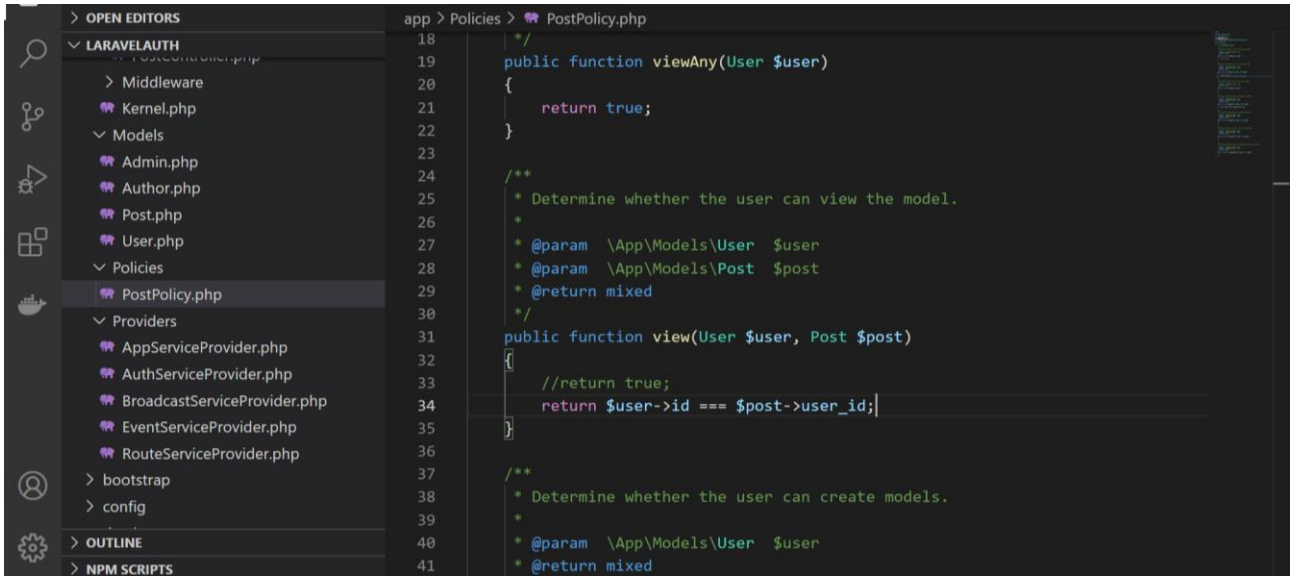

Figure 10: Defining Policy for every method.

***Take note: the model controller methods will be mapped to their corresponding policy method as the following.*

Controller Method	Policy Method
index	viewAny
show	view
create	create
store	create
edit	update
update	update
destroy	delete

4. Controllers Mapping to Policy

Once methods for each action it authorizes in app/Policies/PostPolicy.php file been defined, create methods in PostController which will be mapped to Policy Method as shown in Figure 11.

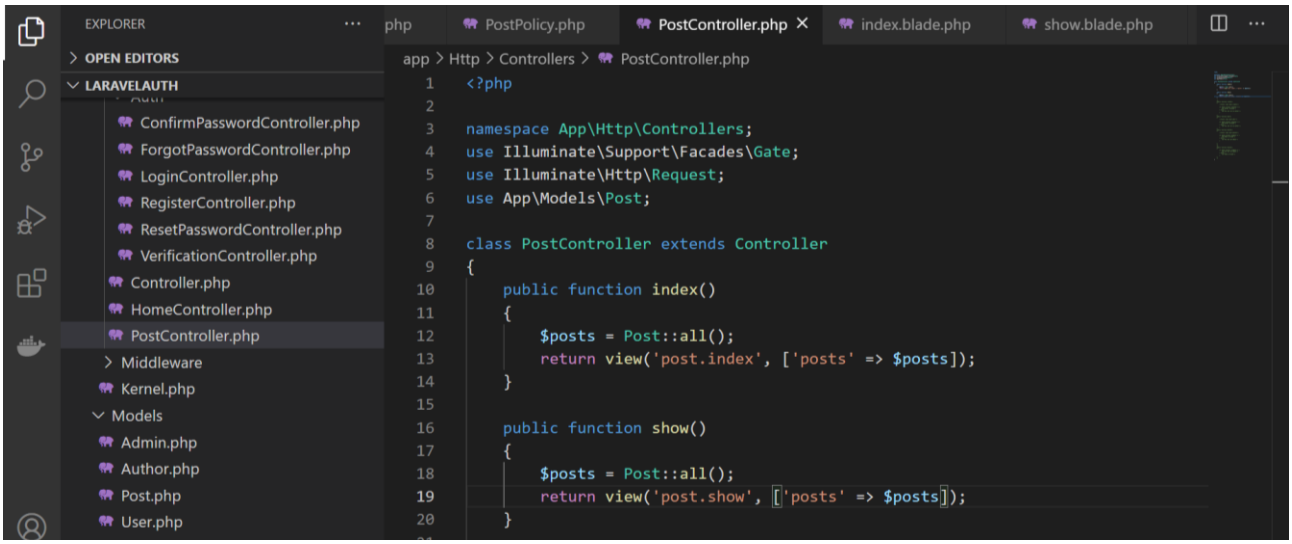

Figure 11: Defining Matching Controller Methods.

4. Defining Policy in Blade

Once methods in PostController mapped to Policy Method, map the policy into Blade Templates as well as shown below.

```
@can('viewAny', $post)
      //view all posts
@endcan

@can('view', $post)
      //view posts only by user
@endcan
```

Exercise:

With all the knowledge and understanding thus far, try to modify and create necessary files in order to have outputs as shown in Figure 12, 13 and 14.

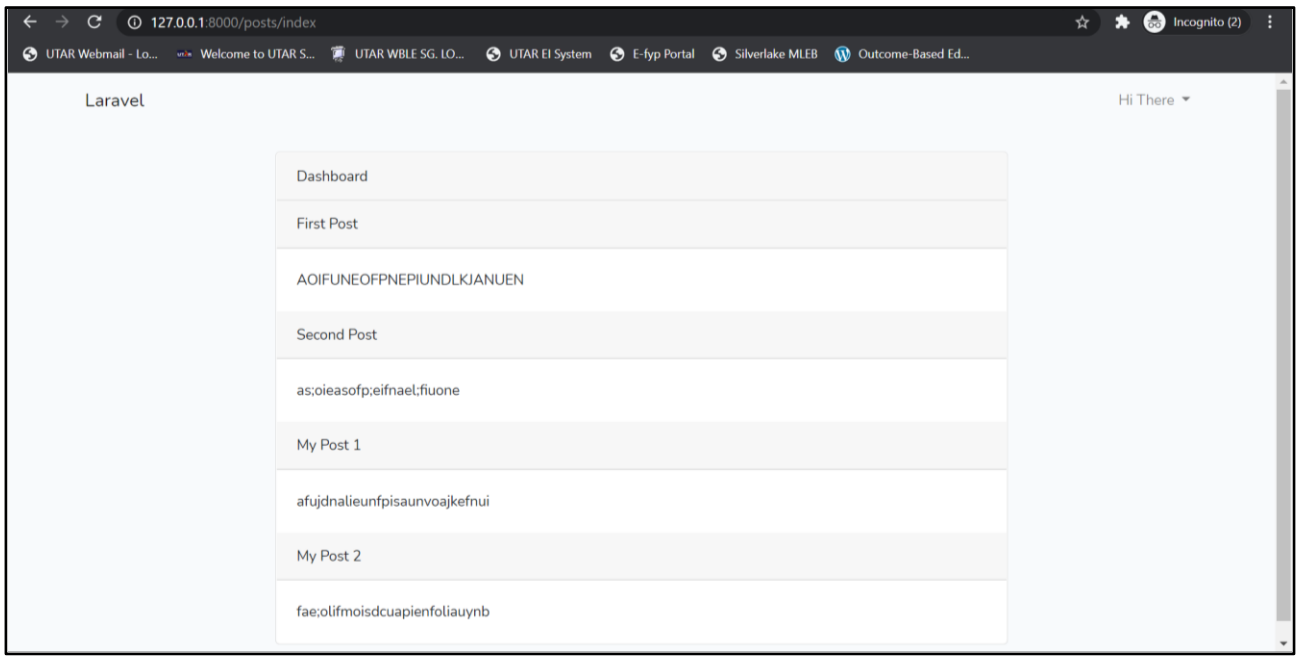

Figure 12: Showing all Posts with /posts/index route.

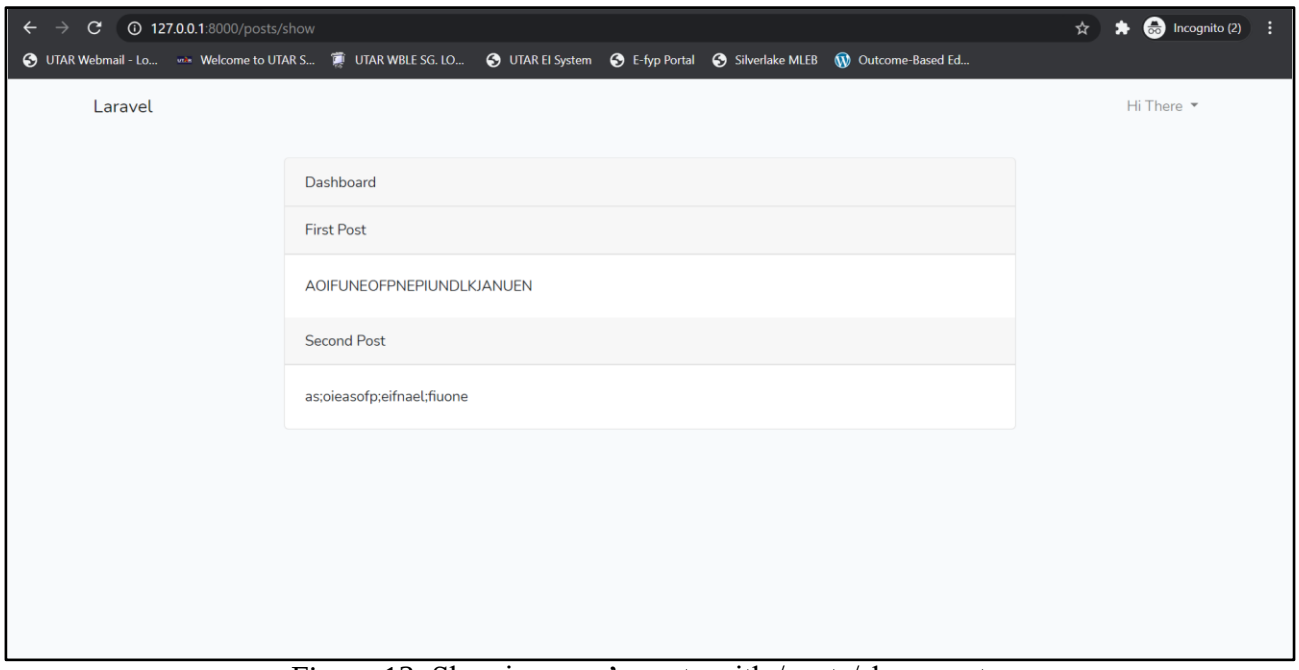

Figure 13: Showing user's posts with /posts/show route.

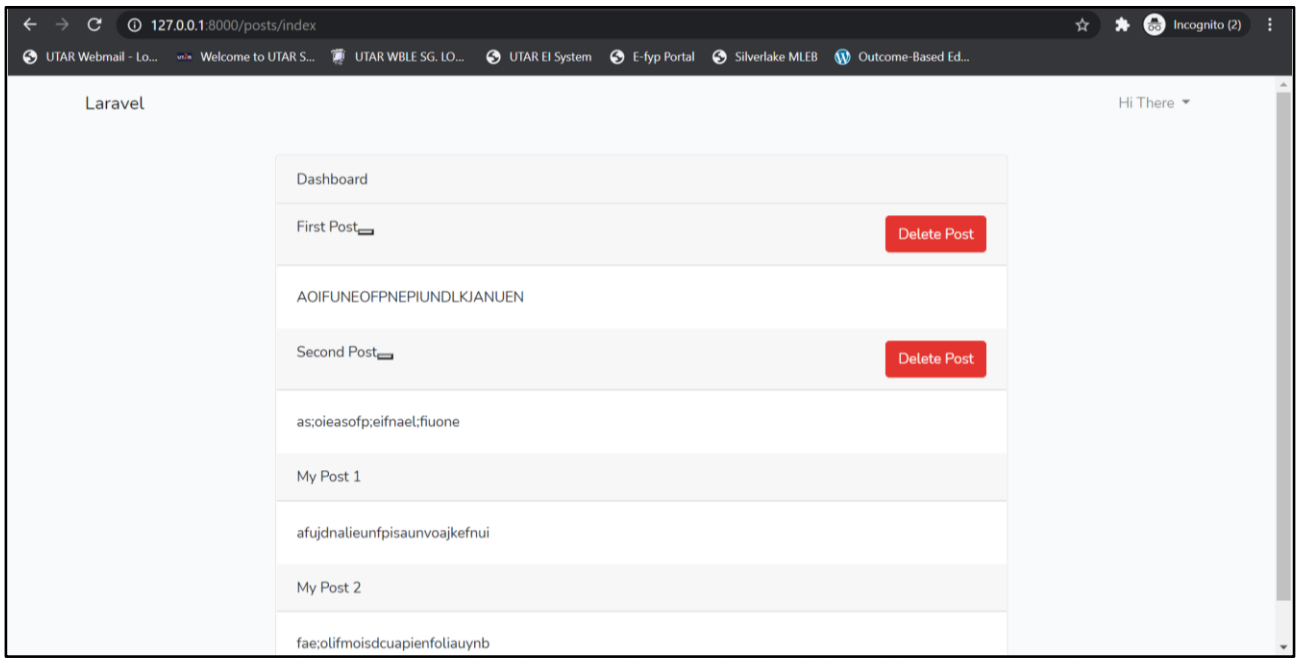

Figure 14: Showing posts by the user that could be deleted by pressing a button.