Laravel - Authorization

Topics : Laravel
Written on December 25, 2023

Authorization in Laravel involves determining if a user has the right permissions to perform a certain action. Laravel provides a simple and expressive way to handle authorization through policies and gates. Here's a guide on how to use authorization in Laravel:

1. Defining Policies:

Policies are classes that organize the authorization logic for a particular model or resource. You can create a policy using the `make:policy` Artisan command.

```
php artisan make:policy PostPolicy
```

This will generate a `PostPolicy` class in the `app/Policies` directory.

2. Registering Policies:

You need to register your policies in the `AuthServiceProvider` class.

```
// app/Providers/AuthServiceProvider.php

protected $policies = [
    'App\Models\Post' => 'App\Policies\PostPolicy',
];
```

3. Defining Policy Methods:

In the generated policy class, you can define methods that correspond to specific actions.

```
// app/Policies/PostPolicy.php

public function view(User $user, Post $post)
{
    return $user->id === $post->user_id;
}
```

4. Using Policies in Controllers:

In your controllers, you can authorize actions using the `authorize` method.

```
public function show(Post $post)
{
    $this->authorize('view', $post);

    // The user is authorized to view the post
    return view('posts.show', compact('post'));
}
```

5. Using Policies in Blade Views:

You can also use policies directly in Blade views.

```
@can('view', $post)
    <!-- The user is authorized to view the post -->
    {{ $post->title }}
@endcan
```

6. Gates:

Gates provide a more general way to define authorization logic. You can define gates in the `AuthServiceProvider` or by using the `Gate` facade.

```
// app/Providers/AuthServiceProvider.php

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

    Gate::define('update-post', function (User $user, Post $post) {
        return $user->id === $post->user_id;
    });
}
```

7. Using Gates:

In controllers or other parts of your application, you can use gates using the `gate` method.

```
public function update(Post $post)
{
    if (Gate::allows('update-post', $post)) {
        // The user is authorized to update the post
    }
}
```

8. Using Middleware for Authorization:

You can apply authorization logic using middleware in your routes.

```
Route::put('/posts/{post}', 'PostController@update')->middleware('can:update-
```

```
post,post');
```

9. Policy Filters:

Laravel provides a convenient `@can` Blade directive for authorizing actions in Blade views.

```
@can('update', $post)
    <!-- The user is authorized to update the post -->
@endcan
```

10. Policy Responses:

You can customize the response when a user is not authorized by defining a `deny` method in your policy.

```
// app/Policies/PostPolicy.php

public function deny(User $user)
{
    return response('You are not authorized.', 403);
}
```

11. Implicit Controller Policies:

You can implicitly authorize actions based on the controller and method names.

```
public function __construct()
{
    $this->authorizeResource(Post::class, 'post');
}
```

12. Defining Abilities in User Model:

You can define abilities directly in your `User` model.

```
public function canEditPost(Post $post)
{
    return $this->id === $post->user_id;
}
```

13. Using Policies in Controllers:

You can use policies directly in controllers.

```
public function edit(Post $post)
{
    $this->authorize('edit-post', $post);
    // The user is authorized to edit the post
```

```
}
```

14. Middleware for Gates:

You can use middleware to authorize actions using gates.

```
Route::put('/posts/{post}', 'PostController@update')->middleware('can:update-
post,post');
```