Many Ways for Multi-authentication in Laravel

BY: engineer Ali-Ali

The main idea of Role-Based Access Control (RBAC) in the context of multi-authentication in Laravel is to manage access to resources and actions within an application based on the roles assigned to users.

How to implement (RBAC)??????

Using Guards

Guards manage authentication in Laravel applications. They determine how user credentials are verified and where the application should look for these credentials. Laravel provides multiple guard drivers, such as "web" for browser sessions and "api" for API token authentication. Guards are configured in the `config/auth.php` file, specifying the driver and provider for each guard

The steps for using the guard as follow 😊

1. Create Separate User Models: Generate separate models for each type of user you want to authenticate. For example, for regular users and administrators.
2. Configure Authentication Providers: In `config/auth.php`, define providers for each user model:
3. Define Guards: Also in `config/auth.php`, define multiple guards, each associated with a user model:
4. Implement Authentication Logic: Use the `auth()` helper method to authenticate users based on the selected guard:
5. Protect Routes: Use middleware to protect routes based on the selected guard. For example, apply the `auth:web` middleware for regular users and `auth:admin` for administrators.

Gates and Policies

- Gates are used for defining and checking authorization policies. They allow for fine-grained access control rules for different actions or operations within the application. Gates are defined in the AuthServiceProvider class using the Gate facade. They are particularly useful for actions not directly related to a model or resource.
- Policies provide a structured way to manage authorization rules specific to your application's models. They are used to define authorization policies for specific models, making it easier to manage rules for CRUD operations related to those models. Policies are most applicable when you wish to authorize an action for a particular model or resource

The steps for using the Gates and polices as follow: 😊

1. Define Gates: In the AuthServiceProvider class, define gates using the Gate facade.
2. Create Policies: Use the artisan command to create a policy for a model
3. define the authorization rules for the model's actions in the policy class.
4. Use Policies in Controllers: In your controller, use the authorize method to check if a user is authorized to perform an action

Using Role-Based Access Control (RBAC) BACKAGES

RBAC is a security model that governs access to resources based on predefined roles and permissions. In Laravel, roles categorize users based on their responsibilities or privileges, while permissions determine the actions or operations that users with specific roles can perform. Laravel offers a robust foundation for implementing RBAC through its built-in features for permissions and roles, including middleware, policies, and gates. This allows for efficient management of user access, streamlining

authorization workflows, and fortifying applications against unauthorized access Achieve Role-Based Access Control (RBAC) using various packages, each offering different features and levels of customization. and

below is simple steps for implement each package for achieve the RBAC

1. Laravel RBAC Package by Itstructure

1. Installation: First, you need to install the package via Composer. Run the following command in your terminal

```
composer require itstructure/laravel-rbac
```

2. Publish Configuration: Publish the package's configuration file to your Laravel application's config directory

```
php artisan vendor:publish --provider="Itstructure\RbacModule\RbacModuleServiceProvider"
```

3. Migrate Database: Run the migrations to create the necessary tables for roles, permissions, and user-role assignments

```
php artisan migrate
```

4. Usage: After installation and configuration, you can use the package's facades to manage roles and permissions. For example, to create a role and assign permissions to it

```
use Itstructure\RbacModule\Models\Role;
use Itstructure\RbacModule\Models\Permission;
$role = Role::create(['name' => 'writer']);
$permission = Permission::create(['name' => 'edit articles']);
$role->permissions()->attach($permission);
```

2. Bouncer

1. Installation: Install the package via Composer

```
composer require silber/bouncer
```

2. Publish Configuration: Publish the package's configuration file

```
php artisan vendor:publish --provider="Silber\Bouncer\BouncerServiceProvider"
```

3. Migrate Database: Run the migrations to create the necessary tables

```
php artisan migrate
```

4. Usage: Use the package's facades to manage roles and permissions. For example, to allow a role to perform an action

```
use Silber\Bouncer\Bouncer;
// Allow a role to perform an action
Bouncer::allow('admin')->to('ban-users');
```

3. Wnikk Laravel Access Rules

- Create a Laravel Application

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

- Install Packages

```
composer require wnikk/laravel-access-rules
composer require wnikk/laravel-access-ui
```

- Publish Configuration and Migration Files

```
php artisan vendor:publish --provider="Wnikk\\LaravelAccessRules\\AccessRulesServiceProvider"
php artisan vendor:publish --provider="Wnikk\\LaravelAccessUi\\AccessUiServiceProvider"
```

- Migrate Database

```
php artisan migrate
```

- Create Rules

```
php artisan make:seeder CreateRulesSeeder
```

- edit the CreateRulesSeeder file to define your rules. Here's an example of how to define various rules:

```
<?php
namespace Database\Seeders;

use Illuminate\Database\Seeder;
use Wnikk\LaravelAccessRules\AccessRules;

class CreateRulesSeeder extends Seeder
{
public function run(): void
{
// Example rules
AccessRules::newRule('example1.viewAny', 'View all users on example1');
AccessRules::newRule('example2.view', 'View data of user on example2');
// Add more rules as needed
}
}
```

- Run the Seeder

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

- Implement Access Control

```
if (AccessRules::can('example1.viewAny')) {
// User has permission to view all users on example1
}
```

- Customize Access UI Edit the config/accessUi.php file to turn off the standard routes:

```
'register' => false,
```

4. Laravel Spatie backage Roles and Permissions

Simplify Role-based Access Control with Laravel Spatie Roles and Permissions | by InkExchanger | Medium

5. Using custom Middleware

How to Create Multiple Authentication in Laravel 9 App | by Bayram EKER | Medium .

Ref:

Laravel 10 Multi Authentication with Guards - WebJourney

Multiple role-based authentication in Laravel - Mastering Backend

How use Access Control Rules and GRUD in Laravel 10 (Tutorial step by step) - DEV Community

Simplify Role-based Access Control with Laravel Spatie Roles and Permissions | by InkExchanger | Medium