**Laravel Roles & Permissions with Authentication & Authorization**

**1. User Authentication (Login, Registration)**

Laravel provides built-in authentication using Laravel Breeze, Jetstream, or custom authentication.

**Install Laravel Breeze (Recommended)**

composer require laravel/breeze --dev

php artisan breeze:install

php artisan migrate

npm install && npm run dev

This sets up authentication (login, registration, password reset) using Blade, Vue, or React.

**2. Roles & Permissions using Spatie Package**

Instead of manually handling roles, the **Spatie Laravel Permission** package makes it easy to manage roles & permissions.

**Step 1: Install Spatie Package**

composer require spatie/laravel-permission

**Step 2: Publish & Run Migrations**

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

php artisan migrate

This will create the following tables:

* roles
* permissions
* model\_has\_roles
* model\_has\_permissions
* role\_has\_permissions

**Step 3: Add Traits to User Model**

Modify app/Models/User.php:

use Spatie\Permission\Traits\HasRoles;

class User extends Authenticatable

{

use HasRoles;

}

**3. Creating Roles and Permissions**

**Step 1: Create Roles & Permissions in a Seeder**

Create a new seeder:

php artisan make:seeder RoleSeeder

Modify database/seeders/RoleSeeder.php:

use Illuminate\Database\Seeder;

use Spatie\Permission\Models\Role;

use Spatie\Permission\Models\Permission;

class RoleSeeder extends Seeder

{

public function run()

{

// Creating Roles

$admin = Role::create(['name' => 'admin']);

$user = Role::create(['name' => 'user']);

// Creating Permissions

$manageUsers = Permission::create(['name' => 'manage users']);

$editPosts = Permission::create(['name' => 'edit posts']);

// Assign permissions to roles

$admin->givePermissionTo([$manageUsers, $editPosts]);

$user->givePermissionTo($editPosts);

}

}

Run the seeder:

php artisan db:seed --class=RoleSeeder

**4. Assign Roles & Permissions to Users**

You can assign roles in a controller or tinker:

use App\Models\User;

use Spatie\Permission\Models\Role;

// Assign role to a user

$user = User::find(1);

$user->assignRole('admin');

// Check user role

if ($user->hasRole('admin')) {

echo "User is an Admin";

}

// Give user a permission directly

$user->givePermissionTo('edit posts');

// Check permission

if ($user->can('edit posts')) {

echo "User can edit posts";

}

**5. Middleware for Role-Based Access Control**

**Create Middleware**

php artisan make:middleware RoleMiddleware

Modify app/Http/Middleware/RoleMiddleware.php:

use Closure;

use Illuminate\Http\Request;

class RoleMiddleware

{

public function handle(Request $request, Closure $next, $role)

{

if (!$request->user() || !$request->user()->hasRole($role)) {

abort(403, 'Unauthorized');

}

return $next($request);

}

}

Register middleware in app/Http/Kernel.php:

protected $routeMiddleware = [

'role' => \App\Http\Middleware\RoleMiddleware::class,

];

**Protect Routes Using Middleware**

Modify routes/web.php:

use App\Http\Controllers\AdminController;

Route::middleware(['auth', 'role:admin'])->group(function () {

Route::get('/admin', [AdminController::class, 'index']);

});

**6. Gate-Based Authorization (Alternative)**

Gates provide another way to authorize actions: Modify app/Providers/AuthServiceProvider.php:

use Illuminate\Support\Facades\Gate;

public function boot()

{

Gate::define('edit-posts', function ($user) {

return $user->hasPermissionTo('edit posts');

});

}

Use in Controller:

if (Gate::allows('edit-posts')) {

echo "User can edit posts";

} else {

abort(403);

}

**Conclusion**

Now your Laravel app supports: ✅ Authentication (Login, Registration)  
✅ Role-Based Access Control (Admin/User)  
✅ Middleware to Restrict Routes  
✅ Gates for Custom Authorization

Let me know if you need any modifications or explanations! 🚀