1v0 Ship v1.0 Faster

Laravel Spatie Permissions: Step-by-Step Installation & Setup

Managing user roles and permissions is a critical part of any web application. Instead of reinventing the wheel, the community-standard package for this in Laravel is **Spatie Laravel-Permission**. It provides a flexible, production-ready way to assign roles and permissions to users, check access in controllers, routes, and views, and even manage everything through a user-friendly UI.

In this tutorial, we'll walk through **installing and setting up Spatie Permissions in Laravel 12**. You'll learn how to configure the package, run migrations, assign roles and permissions to users, and protect routes or views accordingly. We'll also add a simple UI for managing roles and permissions.

1 – Install the Package

First, install the package via Composer:

```
composer require spatie/laravel-permissionCode language: Bash (bash)
```

This pulls the package into your Laravel 12 app. The package integrates seamlessly with Laravel's gate and policy features, extending them with role and permission helpers.

2 – Publish Config & Migration Files

Run the vendor publish command to copy the config and migration files into your project:

```
php artisan vendor:publish --
provider="Spatie\Permission\PermissionServiceProvider"Code language: Bash (bash)
```

This command publishes:

- `config/permission.php`
- Migrations for `roles`, `permissions`, and pivot tables

Now run the migrations:

```
php artisan migrateCode language: Bash (bash)
```

This creates the necessary database tables for roles and permissions.

3 – Add Trait to User Model

The package works by adding a trait to your `User` model. This gives each user instance methods like `assignRole`, `givePermissionTo`, and `hasRole`.

```
// app/Models/User.php
namespace App\Models;

use Illuminate\Foundation\Auth\User as Authenticatable;
use Spatie\Permission\Traits\HasRoles;

class User extends Authenticatable
{
```

```
    use HasRoles;

    // ...
}Code language: PHP (php)
```

Now every user in your system can have roles and permissions assigned to them.

4 – Assign Roles & Permissions

You can seed some initial roles and permissions for testing:

```
// database/seeders/RolesAndPermissionsSeeder.php
namespace Database\Seeders;

use Illuminate\Database\Seeder;
use Spatie\Permission\Models\Role;
use Spatie\Permission\Models\Permission;

class RolesAndPermissionsSeeder extends Seeder
{
    public function run()
    {
        $admin = Role::create(['name' => 'admin']);
        $editor = Role::create(['name' => 'editor']);

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

        $admin->givePermissionTo([$manageUsers, $publishArticles]);
        $editor->givePermissionTo($publishArticles);
    }
```

```
}Code language: PHP (php)
```

Run the seeder to populate roles and permissions:

```
php artisan db:seed --class=RolesAndPermissionsSeederCode language: Bash (bash)
```

Now you can assign roles and permissions to users like this:

```
$user->assignRole('admin');
$user->givePermissionTo('publish articles');Code language: PHP (php)
```

5 – Protect Routes & Controllers

Once users have roles and permissions, you can guard routes using middleware:

```
// routes/web.php
Route::get('/admin', function () {
    return view('admin.dashboard');
})->middleware('role:admin');

Route::get('/articles', function () {
    return view('articles.index');
})->middleware('permission:publish articles');Code language: PHP (php)
```

This ensures only admins see the admin dashboard, and only users with the `publish articles` permission can view the articles page.

6 – Build a Simple UI

To make things user-friendly, let's add a simple interface where admins can assign roles to users. Create a controller and view:

```
// app/Http/Controllers/Admin/UserRoleController.php
namespace App\Http\Controllers\Admin;

use App\Http\Controllers\Controller;
use App\Models\User;
use Illuminate\Http\Request;
use Spatie\Permission\Models\Role;

class UserRoleController extends Controller
{
    public function edit(User $user)
    {
        $roles = Role::all();
        return view('admin.users.edit-roles',
compact('user','roles'));
    }

    public function update(Request $request, User $user)
    {
        $user->syncRoles($request->roles);
        return redirect()->back()->with('status','Roles updated!');
    }
}Code language: PHP (php)
```

Blade view (`resources/views/admin/users/edit-roles.blade.php`):

```
@extends('layouts.app')
```

```
@section('content')
<div class="container">
  <h2>Manage Roles for {{ $user->name }}</h2>

  <form method="POST" action="{{
route('admin.users.roles.update',$user) }}">
    @csrf
    @method('PUT')

    @foreach($roles as $role)
      <div class="form-check">
        <input class="form-check-input" type="checkbox" name="roles[]"
value="{{ $role->name }}"
          {{ $user->hasRole($role->name) ? 'checked' : '' }}>
        <label class="form-check-label">{{ $role->name }}</label>
      </div>
    @endforeach

    <button type="submit" class="btn btn-primary mt-3">Save</button>
  </form>
</div>
@endsectionCode language: HTML, XML (xml)
```

This gives admins a simple checkbox interface to assign or remove roles for a given user.

Wrapping Up

We installed and configured **Spatie Permissions in Laravel 12**, created roles and permissions, assigned them to users, protected routes, and built a simple UI for role management. This package is robust and flexible, letting you scale from simple apps to complex enterprise permission systems with ease.

What’s Next

- How to Assign Roles to Users Dynamically in Laravel — automate role assignment.
- Creating a User-Friendly Roles & Permissions UI in Laravel — polish the admin interface.
- How to Give and Revoke Permissions to Users in Laravel — manage permissions directly.