# Building a Role-Based Admin Panel in Laravel 12

One of the most powerful ways to manage user access in a modern web application is by creating an **Admin Panel with role-based permissions**. This ensures that administrators, editors, and regular users only see and control what they're allowed to. In **Laravel 12**, we can achieve this quickly and cleanly using the **Spatie Laravel-Permission** package.

In this step-by-step guide, you'll learn how to build a **Role-Based Admin Panel in Laravel 12**. We'll cover everything from installing and configuring Spatie Permissions, creating roles and permissions, assigning them to users, setting up route protection, and even building a simple admin interface to manage everything visually.

## 1 – Install Laravel & Spatie Permissions

Start with a fresh Laravel 12 project (or use an existing one). Install Laravel using Composer:

```
composer create-project laravel/laravel role-based-admin
```

Now install the Spatie Permission package:

```
cd role-based-admin
composer require spatie/laravel-permissionCode language: JavaScript (javascript)
```

This package gives you ready-to-use models and methods for roles and permissions.

## 2 – Publish Config & Run Migrations

Next, publish the configuration and migration files for the package:

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

This will create a `config/permission.php` file and migration files for `roles`, `permissions`, and `model_has_roles`.

Run the migrations to apply them:

```
php artisan migrate
```

This adds the required database tables for role and permission management.

## 3 – Update the User Model

Add the `HasRoles` trait to your `User` model so users can be assigned roles and permissions:

```
// 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;

    protected $fillable = [
        'name',
```

```
        'email',
        'password',
    ];
}Code language: PHP (php)
```

Now, you can use methods like `$user->assignRole('admin')` or `$user->givePermissionTo('edit articles')`.

## 4 – Seeding Roles & Permissions

Let's create initial roles and permissions so your admin panel has a base to work with.

```
php artisan make:seeder RolesAndPermissionsSeederCode language: CSS (css)
```

```
// 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()
    {
        // Create roles
        $admin = Role::create(['name' => 'admin']);
        $editor = Role::create(['name' => 'editor']);
        $user = Role::create(['name' => 'user']);

        // Create permissions
        $manageUsers = Permission::create(['name' => 'manage users']);
        $publishPosts = Permission::create(['name' => 'publish
```

```
posts']);
        $viewReports = Permission::create(['name' => 'view reports']);

        // Assign permissions
        $admin->givePermissionTo([$manageUsers, $publishPosts,
$viewReports]);
        $editor->givePermissionTo($publishPosts);
        $user->givePermissionTo($viewReports);
    }
}Code language: PHP (php)
```

This seeder creates three roles (admin, editor, user) and three permissions. Each role is assigned different levels of access. Run the seeder:

```
php artisan db:seed --class=RolesAndPermissionsSeederCode language:
JavaScript (javascript)
```

## 5 – Protect Routes with Middleware

With roles and permissions created, you can now restrict access to certain pages. Laravel's `role` and `permission` middleware comes from Spatie.

```
// routes/web.php

use App\Http\Controllers\Admin\DashboardController;

Route::middleware(['auth','role:admin'])->group(function () {
    Route::get('/admin/dashboard', [DashboardController::class,
'index'])->name('admin.dashboard');
});Code language: PHP (php)
```

Now, only users with the `admin` role can access the admin dashboard. Others will receive a `403 Forbidden` response.

## 6 – Create an Admin Dashboard

Let's build a simple admin dashboard for demonstration:

```
php artisan make:controller Admin/DashboardController
```

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

use App\Http\Controllers\Controller;

class DashboardController extends Controller
{
    public function index()
    {
        return view('admin.dashboard');
    }
}Code language: PHP (php)
```

```
<!-- resources/views/admin/dashboard.blade.php -->
@extends('layouts.app')

@section('content')
<div class="container">
  <h1>Admin Dashboard</h1>
  <p>Welcome, {{ auth()->user()->name }}. You are logged in as an
admin.</p>
</div>
@endsectionCode language: HTML, XML (xml)
```

## 7 – Admin UI for Managing Users

Next, we'll create an interface for admins to assign or revoke roles for users directly in the admin panel.

```
php artisan make:controller Admin/UserManagementController
```

```
// app/Http/Controllers/Admin/UserManagementController.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 UserManagementController extends Controller
{
    public function index()
    {
        $users = User::with('roles')->get();
        return view('admin.users.index', compact('users'));
    }

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

    public function update(Request $request, User $user)
    {
        $user->syncRoles($request->roles ?? []);
        return
```

```
redirect()->route('admin.users.index')->with('status','Roles
updated!');
    }
}Code language: PHP (php)
```

```
<!-- resources/views/admin/users/index.blade.php -->
@extends('layouts.app')

@section('content')
<div class="container">
  <h2>User Management</h2>
  <table class="table table-striped">
    <thead>
      <tr>
        <th>Name</th>
        <th>Email</th>
        <th>Roles</th>
        <th>Action</th>
      </tr>
    </thead>
    <tbody>
      @foreach($users as $user)
      <tr>
        <td>{{ $user->name }}</td>
        <td>{{ $user->email }}</td>
        <td>{{ implode(', ', $user->getRoleNames()->toArray()) }}</td>
        <td><a href="{{ route('admin.users.edit',$user) }}" class="btn
btn-sm btn-outline-primary">Edit</a></td>
      </tr>
      @endforeach
    </tbody>
  </table>
</div>
@endsectionCode language: HTML, XML (xml)
```

```
<!-- resources/views/admin/users/edit.blade.php -->
@extends('layouts.app')

@section('content')
```

```
<div class="container">
  <h2>Edit Roles for {{ $user->name }}</h2>
  <form method="POST" action="{{ route('admin.users.update',$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">{{ ucfirst($role->name)
}}</label>
      </div>
    @endforeach
    <button type="submit" class="btn btn-success mt-3">Update</button>
  </form>
</div>
@endsectionCode language: HTML, XML (xml)
```

## Wrapping Up

We've built a complete **role-based admin panel in Laravel 12**. Starting from installing Spatie Permissions, creating roles and permissions, assigning them to users, and protecting routes — all the way to creating an admin interface for managing user roles. This setup provides a strong foundation for secure, scalable user management in any application.

If you prefer a ready-to-use implementation with authentication, roles, permissions, and a prebuilt UI, check out the Laravel Authentication & User Management Kit from 1v0.net.

## What’s Next

- How to Give and Revoke Permissions to Users in Laravel
- Creating a Role-Specific Dashboard in Laravel 12
- Laravel Middleware for Role-Based Route Protection