1v0 Ship v1.0 Faster

# How to Manage Permissions in Laravel Without Coding

Managing permissions directly in code works fine for developers, but in real projects, you'll often want **non-developers (like team managers or admins)** to control who can do what. Instead of hardcoding permissions, Laravel + Spatie Permissions lets you store them in the database and even manage them through a user-friendly interface.

In this guide, we'll build a **permissions management system in Laravel 12** that doesn't require writing PHP code. Admins will be able to create new permissions, assign them to roles, and give or revoke permissions from users directly from the UI.

## 1 – Prerequisites

Before continuing, make sure you have followed our Spatie Permissions setup guide. You should already have the required migrations, `User` model updated with the `HasRoles` trait, and some roles seeded.

## 2 – Storing Permissions in the Database

With Spatie, permissions are stored in the `permissions` table. This means you don't need to update your codebase when you want to add new ones — you just insert rows into the table.

```
INSERT INTO permissions (name, guard_name, created_at, updated_at)
VALUES ('approve comments', 'web', NOW(), NOW());Code language: JavaScript
```

(javascript)

This creates a new permission named `approve comments`. Once created, you can assign it to roles or users dynamically without modifying your PHP files.

## 3 – Building a Permissions Management UI

Let's create a simple UI where admins can add new permissions, assign them to roles, and manage them without coding.

```
// routes/web.php
use App\Http\Controllers\Admin\PermissionController;

Route::middleware(['auth','role:admin'])->group(function () {
    Route::resource('/admin/permissions',
PermissionController::class);
});Code language: PHP (php)
```

Generate the controller:

```
php artisan make:controller Admin/PermissionController --resource
```

Controller logic (`app/Http/Controllers/Admin/PermissionController.php`):

```
namespace App\Http\Controllers\Admin;

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

class PermissionController extends Controller
{
    public function index()
```

```
    {
        $permissions = Permission::all();
        return view('admin.permissions.index',
compact('permissions'));
    }

    public function create()
    {
        return view('admin.permissions.create');
    }

    public function store(Request $request)
    {
        $request->validate([
            'name' => 'required|unique:permissions,name'
        ]);

        Permission::create(['name' => $request->name]);
        return
redirect()->route('permissions.index')->with('status','Permission
created!');
    }

    public function edit(Permission $permission)
    {
        return view('admin.permissions.edit', compact('permission'));
    }

    public function update(Request $request, Permission $permission)
    {
        $request->validate([
            'name' =>
'required|unique:permissions,name,'.$permission->id
        ]);

        $permission->update(['name' => $request->name]);
        return
redirect()->route('permissions.index')->with('status','Permission
```

```
updated!');
    }

    public function destroy(Permission $permission)
    {
        $permission->delete();
        return
redirect()->route('permissions.index')->with('status','Permission
deleted!');
    }
}Code language: PHP (php)
```

Example Blade view (`resources/views/admin/permissions/index.blade.php`):

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

@section('content')
<div class="container">
  <h2>Permissions</h2>
  <a href="{{ route('permissions.create') }}" class="btn btn-primary
mb-3">Add Permission</a>

  <table class="table table-bordered">
    <thead>
      <tr><th>Name</th><th>Actions</th></tr>
    </thead>
    <tbody>
      @foreach($permissions as $permission)
      <tr>
        <td>{{ $permission->name }}</td>
        <td>
          <a href="{{ route('permissions.edit',$permission) }}"
class="btn btn-sm btn-outline-secondary">Edit</a>
          <form action="{{ route('permissions.destroy',$permission)
}}" method="POST" style="display:inline">
            @csrf
            @method('DELETE')
            <button type="submit" class="btn btn-sm btn-outline-
```

```
danger">Delete</button>
          </form>
        </td>
      </tr>
      @endforeach
    </tbody>
  </table>
</div>
@endsectionCode language: HTML, XML (xml)
```

This interface allows an admin to add, edit, and delete permissions directly in the browser — no code required.

## 4 – Assign Permissions to Roles from the UI

You can expand the UI to let admins attach permissions to roles. Example controller snippet:

```
// In RoleController@edit
$permissions = Permission::all();
return view('admin.roles.edit', compact('role','permissions'));Code language: PHP (php)
```

Then in your view, simply render checkboxes for each permission, and use `$role->syncPermissions($request->permissions)` in your controller to save updates.

## 5 – Enforcing Permissions on Controllers (Without Coding)

Creating permissions is only half the story — we must apply them to specific pages or actions. To keep this “no-code”, we’ll store permission-to-route mappings in the database and enforce them with a middleware.

First, create a migration for a new table to store route-permission pairs:

```
php artisan make:migration create_route_permissions_tableCode language: CSS (css)
```

```
// database/migrations/xxxx_xx_xx_create_route_permissions_table.php
use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration {
    public function up(): void {
        Schema::create('route_permissions', function (Blueprint $table) {
            $table->id();
            $table->string('route_name');
            $table->string('permission_name');
            $table->timestamps();
        });
    }

    public function down(): void {
        Schema::dropIfExists('route_permissions');
    }
};Code language: PHP (php)
```

This table will store which permission is required for which named route.

## 6 – Create a Middleware to Enforce DB Permissions

Next, build a middleware that checks the current route against the database and ensures the logged-in user has the required permission.

```
php artisan make:middleware DynamicPermissionMiddlewareCode language: CSS (css)
```

```
// app/Http/Middleware/DynamicPermissionMiddleware.php
namespace App\Http\Middleware;

use Closure;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Route;
use App\Models\RoutePermission;

class DynamicPermissionMiddleware
{
    public function handle(Request $request, Closure $next)
    {
        $routeName = Route::currentRouteName();

        $mapping = RoutePermission::where('route_name',
$routeName)->first();

        if ($mapping && !
$request->user()?->can($mapping->permission_name)) {
            abort(403, 'You do not have permission to access this
page.');
        }

        return $next($request);
    }
```

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

Register this middleware in `app/Http/Kernel.php` under `$routeMiddleware` as `'dynamic.permission'`.

Now, any route using this middleware will check its required permission dynamically from the database.

## 7 – Admin UI to Map Routes to Permissions

Finally, let's give admins the power to assign which permission protects which route — all from the UI.

```
// routes/web.php
use App\Http\Controllers\Admin\RoutePermissionController;

Route::middleware(['auth','role:admin'])->group(function () {
    Route::resource('/admin/route-permissions',
RoutePermissionController::class);
});Code language: PHP (php)
```

Controller (`app/Http/Controllers/Admin/RoutePermissionController.php`):

```
namespace App\Http\Controllers\Admin;

use App\Http\Controllers\Controller;
use Illuminate\Http\Request;
use App\Models\RoutePermission;
use Spatie\Permission\Models\Permission;
use Illuminate\Support\Facades\Route;

class RoutePermissionController extends Controller
{
```

```
    public function index()
    {
        $routes =
collect(Route::getRoutes())->pluck('action.as')->filter();
        $permissions = Permission::all();
        $mappings = RoutePermission::all();

        return view('admin.route-permissions.index',
compact('routes','permissions','mappings'));
    }

    public function store(Request $request)
    {
        RoutePermission::updateOrCreate(
            ['route_name' => $request->route_name],
            ['permission_name' => $request->permission_name]
        );

        return redirect()->back()->with('status','Mapping saved!');
    }
}Code language: PHP (php)
```

Blade view (`resources/views/admin/route-permissions/index.blade.php`):

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

@section('content')
<div class="container">
  <h2>Route-Permission Mappings</h2>
  <form method="POST" action="{{ route('route-permissions.store') }}"
class="mb-4">
    @csrf
    <div class="mb-3">
      <label>Route</label>
      <select name="route_name" class="form-select">
        @foreach($routes as $route)
          <option value="{{ $route }}">{{ $route }}</option>
        @endforeach
```

```
      </select>
    </div>

    <div class="mb-3">
      <label>Permission</label>
      <select name="permission_name" class="form-select">
        @foreach($permissions as $permission)
          <option value="{{ $permission->name }}">{{ $permission->name
}}</option>
        @endforeach
      </select>
    </div>

    <button type="submit" class="btn btn-primary">Save
Mapping</button>
  </form>

  <h3>Existing Mappings</h3>
  <ul>
    @foreach($mappings as $map)
      <li>Route: {{ $map->route_name }} → Permission: {{
$map->permission_name }}</li>
    @endforeach
  </ul>
</div>
@endsectionCode language: HTML, XML (xml)
```

Now, an admin can log into the UI, select a route (by name), and choose which permission is required. The middleware enforces it automatically — without writing any PHP code.

## Wrapping Up

We built a **permissions management UI in Laravel 12** that lets admins create, update, and delete permissions directly from the browser. By storing permissions in the database and exposing them via an admin panel, you empower non-developers to control application access without touching code. This makes your app far more flexible and team-friendly.

## What's Next

- How to Give and Revoke Permissions to Users in Laravel
- Creating a User-Friendly Roles & Permissions UI in Laravel
- Laravel Middleware for Role-Based Route Protection