1v0 Ship v1.0 Faster

Laravel Middleware for Role-Based Route Protection

In every web application, certain routes should only be accessible by specific users. For example, an **admin dashboard** should be restricted to admins, while editors may only manage content. In **Laravel 12**, this is best achieved using **middleware** — small classes that filter requests before they reach controllers.

In this guide, you'll learn how to use middleware to **protect routes by role and permission**. We'll cover Spatie's built-in middleware, how to create your own custom middleware, and how to apply them effectively in your application.

1 – Why Middleware for Roles?

Middleware acts like a security checkpoint. When a request enters your app, middleware decides if it should continue. With **role-based middleware**, you can make sure that only users with the right roles/permissions can access sensitive pages.

2 – Spatie's Role & Permission Middleware

When you install Spatie Permissions, it registers two middleware out of the box:

- `role:<role>` — restricts access to a role
- `permission:<permission>` — restricts access to a permission

Example usage:

```
// routes/web.php

Route::middleware(['auth','role:admin'])->group(function () {
    Route::get('/admin/dashboard', function () {
        return 'Welcome Admin';
    });
});

Route::middleware(['auth','permission:publish posts'])->group(function
() {
    Route::get('/editor/posts', function () {
        return 'Editor Post Management';
    });
});Code language: PHP (php)
```

Now only admins can access `/admin/dashboard`, and only users with the `publish posts` permission can access `/editor/posts`.

3 – Multiple Roles & Permissions

You can also allow multiple roles or permissions in a single middleware declaration by separating them with a pipe (`|`):

```
// routes/web.php

Route::middleware(['auth','role:admin|manager'])->group(function () {
    Route::get('/reports', function () {
        return 'Reports Page';
    });
});Code language: PHP (php)
```

In this example, both `admin` and `manager` roles can access the `/reports` route.

4 – Creating Custom Middleware

Sometimes, Spatie's built-in middleware isn't enough. For example, maybe you want to restrict access based on **both a role and a specific condition**. In that case, you can create your own middleware.

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

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

use Closure;
use Illuminate\Http\Request;

class CheckEditorApproval
{
    public function handle(Request $request, Closure $next)
    {
        if (! $request->user()->hasRole('editor') || !
$request->user()->approved) {
            abort(403, 'Access denied.');
        }

        return $next($request);
    }
}Code language: PHP (php)
```

Register your middleware in `app/Http/Kernel.php` under `$routeMiddleware` and apply it like this:

```
Route::middleware(['auth','check.editor'])->group(function () {
    Route::get('/editor/dashboard', function () {
        return 'Editor Dashboard';
```

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

This ensures that only approved editors can access the dashboard.

5 – Applying Middleware in Controllers

You don’t have to declare middleware only in routes. You can also apply them directly inside controllers:

```
// app/Http/Controllers/Admin/DashboardController.php

class DashboardController extends Controller
{
    public function __construct()
    {
        $this->middleware(['auth','role:admin']);
    }

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

This way, any route pointing to `DashboardController` will automatically be protected by the `admin` role requirement.

6 – Using HasMiddleware in Laravel 12

Laravel 12 introduces a modern way to assign middleware directly in controllers using the `HasMiddleware` interface and a static `middleware()` method. This is cleaner than the old `$this->middleware()` approach and doesn't require constructors.

 Important: In older Laravel versions, your base controller extended `Illuminate\Routing\Controller` (aliased as `BaseController`). That class contains an instance `middleware()` method, which **conflicts** with the new static `middleware()` method. In Laravel 12, controllers no longer need to extend `Illuminate\Routing\Controller`. You should update your `App\Http\Controllers\Controller` to remove the inheritance.

```
// Before (default older style)
use Illuminate\Routing\Controller as BaseController;

class Controller extends BaseController
{
    use AuthorizesRequests, ValidatesRequests;
}

// After (Laravel 12 style – no BaseController needed)
class Controller
{
    use AuthorizesRequests, ValidatesRequests;
}Code language: PHP (php)
```

Now, you can safely implement `HasMiddleware` in your controllers without conflicts:

```
// app/Http/Controllers/Admin/DashboardController.php

namespace App\Http\Controllers\Admin;

use App\Http\Controllers\Controller;
use Illuminate\Routing\Controllers\HasMiddleware;
use Illuminate\Routing\Controllers\Middleware;

class DashboardController extends Controller implements HasMiddleware
```

```
{
    public static function middleware(): array
    {
        return [
            new Middleware('auth'),
            new Middleware('role:admin'),
        ];
    }

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

With this setup, all actions in the `DashboardController` are automatically protected by the `auth` and `role:admin` middleware. You can also restrict specific actions using `only` or `except` options.

Wrapping Up

Middleware is the most effective way to enforce role-based security in Laravel 12. With Spatie's built-in middleware, you can restrict routes by role or permission in seconds. For advanced scenarios, you can write custom middleware to check additional conditions. By combining these approaches, you can secure every part of your app's routes with precision.

1v0 Ship v1.0 Faster

What’s Next

- Building a Role-Based Admin Panel in Laravel 12 — complete role & permission management.
- Creating a Role-Specific Dashboard in Laravel 12 — tailor dashboards by role.
- How to Create a Multi-Level Role & Permission System in Laravel — learn advanced role hierarchies.