1v0 Ship v1.0 Faster

How to Restrict Page Access by Role in Laravel 12

When building modern web applications, controlling **who can see what** is just as important as authentication. Not all users should have the same level of access — for example, an admin may need to manage users, while regular users should only see their own content. This is where **Role-Based Access Control (RBAC)** comes in.

In this tutorial, you'll learn **how to restrict page access by role in Laravel 12**. We'll cover what roles and permissions are, how to create a simple role system, how to assign roles to users, and how to protect routes and pages so that only authorized users can see them.

1 – What are Roles and Permissions?

Role: A role represents a group of permissions that can be assigned to a user. Examples include `admin`, `editor`, or `user`.

Permission: A permission defines what an account is allowed to do, such as `edit-post` or `delete-user`. Roles usually contain multiple permissions.

By combining them, you can control what different users can access in your Laravel 12 app. In this guide, we'll implement a basic version without external packages, so you understand the inner workings.

2 - Prerequisites

- A fresh **Laravel 12** project (see our setup guide)
- Basic authentication already working (see our Laravel auth guide)
- A database with the `users` table ready

3 - Update the Users Table with a Role Column

We'll keep things simple by adding a `role` column directly to the `users` table. Later, you can expand this into a full permissions table if needed.

```bash
php artisan make:migration add_role_to_users_table --table=users
```
Code language: Bash (bash)

```php
// database/migrations/xxxx_xx_xx_xxxxxx_add_role_to_users_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::table('users', function (Blueprint $table) {
            $table->string('role')->default('user');
        });
    }

    public function down(): void {
        Schema::table('users', function (Blueprint $table) {
            $table->dropColumn('role');
        });
    }
};
```
Code language: PHP (php)

This migration adds a new `role` field with a default value of `user`. That means anyone who registers gets the "user" role automatically, unless you assign them another one (like "admin").

4 – Middleware for Role Protection

To restrict access, we'll create a custom middleware that checks if the current user has the required role.

```
php artisan make:middleware RoleMiddlewareCode language: Bash (bash)
```

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

use Closure;
use Illuminate\Support\Facades\Auth;

class RoleMiddleware
{
    public function handle($request, Closure $next, $role)
    {
        if (!Auth::check() || Auth::user()->role !== $role) {
            abort(403, 'Unauthorized');
        }
        return $next($request);
    }
}Code language: PHP (php)
```

This middleware compares the user's `role` column with the role required. If it doesn't match, we abort with a 403 (Forbidden).

Now register the middleware in `app/Http/Kernel.php`:

```
// app/Http/Kernel.php
protected $routeMiddleware = [
    // ...
    'role' => \App\Http\Middleware\RoleMiddleware::class,
];
```
Code language: PHP (php)

5 – Protecting Routes by Role

Now that the middleware is registered, apply it to routes. For example, let's protect an admin dashboard so only users with the `admin` role can access it.

```
// routes/web.php
Route::get('/admin', function () {
    return view('admin.dashboard');
})->middleware(['auth','verified','role:admin']);
```
Code language: PHP (php)

Here, we stacked three middleware: `auth` (user must be logged in), `verified` (user must have confirmed email), and `role:admin` (user must be an admin). Any unauthorized user will see a 403 error automatically.

6 – Adding Role Selection in the UI

If you want to let an admin assign roles via the UI, add a simple form. For example, let's create a form to update a user's role:

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

```
@section('content')
<div class="container">
  <h2>Edit User Role</h2>
  <form method="POST" action="{{ route('admin.users.update', $user)
}}">
    @csrf
    @method('PUT')
    <div class="mb-3">
      <label class="form-label">Role</label>
      <select name="role" class="form-control">
        <option value="user" {{ $user->role==='user'?'selected':''
}}>User</option>
        <option value="admin" {{ $user->role==='admin'?'selected':''
}}>Admin</option>
      </select>
    </div>
    <button class="btn btn-primary">Update Role</button>
  </form>
</div>
@endsectionCode language: PHP (php)
```

This gives administrators a dropdown to assign `user` or `admin`. You can extend this idea with a dedicated `roles` table for more complex scenarios.

7 - Common Errors & Fixes

- **403 error even for admins:** Ensure the `role` column has the correct value and your middleware is spelled correctly.
- **Middleware not working:** Double-check that you registered the middleware alias in `Kernel.php`.
- **New users can't access pages:** Remember that the default `role` is `user`. If you want new admins, set their role after registration.

Wrapping Up

You’ve successfully restricted pages by role in Laravel 12. We started with a simple role column, created a custom `RoleMiddleware`, and applied it to routes. We also built a small UI for admins to change user roles. This approach is simple and great for beginners; as your app grows, you might switch to a package like Spatie Permissions for more advanced control.

What’s Next

- Laravel Spatie Permissions: Step-by-Step Installation & Setup — learn the professional way of handling roles.
- Creating a User-Friendly Roles & Permissions UI in Laravel — manage roles and permissions directly in the dashboard.
- How to Give and Revoke Permissions to Users in Laravel — control access dynamically.