How to Create a Multi-Level Role & Permission System in Laravel

As applications grow, a simple “Admin vs User” setup is rarely enough. You might need multiple levels of access: **Super Admins** who control everything, **Managers** who oversee teams, and **Users** who only manage their own data. In **Laravel 12**, you can build a flexible **multi-level role and permission system** using the **Spatie Permissions** package.

In this guide, we’ll walk through creating hierarchical roles and assigning permissions at different levels. We’ll also add a UI so admins can manage roles dynamically, without writing code.

1 – Setting Up Spatie Permissions

Make sure you’ve installed and configured Spatie Permissions. If not, follow our installation guide. Once set up, add the `HasRoles` trait to your `User` model and run the migrations.

2 – Defining Multi-Level Roles

Let’s create three levels of roles: **Super Admin**, **Manager**, and **User**. Super Admins have all permissions, Managers have elevated access, and Users have limited access.

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

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

class MultiLevelRolesSeeder extends Seeder
{
    public function run()
    {
        $superAdmin = Role::create(['name' => 'super-admin']);
        $manager = Role::create(['name' => 'manager']);
        $user = Role::create(['name' => 'user']);

        // Permissions
        $manageUsers = Permission::create(['name' => 'manage users']);
        $viewReports = Permission::create(['name' => 'view reports']);
        $editProfile = Permission::create(['name' => 'edit profile']);

        // Assign permissions
        $superAdmin->givePermissionTo([$manageUsers, $viewReports,
$editProfile]);
        $manager->givePermissionTo([$viewReports, $editProfile]);
        $user->givePermissionTo([$editProfile]);
    }
}Code language: PHP (php)
```

This creates three distinct access levels. Each level includes the permissions of the one below it.

3 – Middleware Protection

Now, protect routes based on roles or permissions:

```
// routes/web.php
```

```
// Super Admin only
Route::middleware(['auth','role:super-admin'])->group(function () {
    Route::get('/admin/settings', function () {
        return 'Settings Page';
    });
});

// Managers and above
Route::middleware(['auth','role:manager|super-admin'])->group(function
() {
    Route::get('/reports', function () {
        return 'Reports Page';
    });
});

// All authenticated users
Route::middleware(['auth','role:user|manager|super-
admin'])->group(function () {
    Route::get('/profile', function () {
        return 'Profile Page';
    });
});Code language: PHP (php)
```

Using multiple roles in the middleware ensures a hierarchy of access — higher-level roles automatically get lower-level privileges.

4 – Role Inheritance with Gates

If you want a more formal hierarchy, you can extend roles via custom logic. For example, treat `super-admin` as having all permissions without assigning them manually.

```
// app/Providers/AuthServiceProvider.php
use Illuminate\Support\Facades\Gate;
```

```
public function boot()
{
    $this->registerPolicies();

    Gate::before(function ($user, $ability) {
        return $user->hasRole('super-admin') ? true : null;
    });
}Code language: PHP (php)
```

This ensures that super-admins can do anything, regardless of specific permissions.

5 – Admin UI for Multi-Level Roles

Finally, let's create a simple admin panel where Super Admins can assign roles to users. This UI helps non-technical admins manage multi-level access.

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

Route::middleware(['auth','role:super-admin'])->group(function () {
    Route::get('/admin/users/{user}/roles',
[UserRoleController::class, 'edit'])->name('admin.users.roles.edit');
    Route::put('/admin/users/{user}/roles',
[UserRoleController::class,
'update'])->name('admin.users.roles.update');
});Code language: PHP (php)
```

```
// 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
successfully');
    }
}Code language: PHP (php)
```

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

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

@section('content')
<div class="container">
  <h2>Assign Roles to {{ $user->name }}</h2>
  <form method="POST" action="{{
route('admin.users.roles.update',$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">{{ ucfirst($role->name)
}}</label>
      </div>
```

```
    @endforeach
    <button type="submit" class="btn btn-primary mt-3">Save</button>
  </form>
</div>
@endsectionCode language: JavaScript (javascript)
```

This lets Super Admins manage roles across all levels, ensuring fine-grained access control in your application.

Wrapping Up

We built a **multi-level role and permission system in Laravel 12** using Spatie Permissions. Roles like Super Admin, Manager, and User provide hierarchical access, with middleware and Gates ensuring proper control. By adding an admin UI, you empower administrators to manage roles dynamically, without writing code.

What's Next

- Laravel Middleware for Role-Based Route Protection — secure routes effectively.
- Creating a Role-Specific Dashboard in Laravel 12 — show different dashboards per role.
- How to Give and Revoke Permissions to Users in Laravel — manage permissions dynamically.