Building a Team Management System with Laravel Roles & Permissions

Many modern SaaS applications need **team-based access control**. For example, a user might be an `admin` in one team but just a `member` in another. In **Laravel 12**, you can achieve this by enabling **team support in Spatie Permissions** and building a management UI for teams, members, and roles.

In this guide, we'll create a **Team Management System** where users can belong to multiple teams, each with its own roles and permissions. We'll enable Spatie's team mode, update our models, enforce team roles with middleware, and build a UI to manage team members.

1 – Enabling Teams in Spatie Permissions

The Spatie Laravel Permission package supports **teams** (multi-tenancy). By default it's off, but you can enable it in the config file. This ensures that all roles and permissions are scoped to a specific team.

Fresh installation: open `config/permission.php` and enable teams:

```
'teams' => true,
'team_foreign_key' => 'team_id',Code language: PHP (php)
```

Then run the migrations. Spatie will automatically add `team_id` columns to its tables.

If Spatie was already installed without teams, you can still enable it:

- Set `'teams' => true` in `config/permission.php`.
- Create a migration to add `team_id` to the `model_has_roles` and `model_has_permissions` tables.

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

```
// database/migrations/xxxx_xx_xx_add_team_id_to_permission_tables.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('model_has_roles', function (Blueprint $table) {
$table->unsignedBigInteger('team_id')->nullable()->index();
        });

        Schema::table('model_has_permissions', function (Blueprint
$table) {
$table->unsignedBigInteger('team_id')->nullable()->index();
        });
    }

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

Once migrated, all role and permission checks will require a `team` context parameter.

2 – Updating Models

Create a `Team` model and link it with users. Users can belong to multiple teams with different roles.

```
php artisan make:model Team -mCode language: CSS (css)
```

```
// app/Models/User.php
namespace App\Models;

use Illuminate\Foundation\Auth\User as Authenticatable;
use Illuminate\Database\Eloquent\Relations\BelongsToMany;

class User extends Authenticatable
{
    public function teams(): BelongsToMany
    {
        return $this->belongsToMany(Team::class)
                    ->withPivot('role')
                    ->withTimestamps();
    }
}Code language: PHP (php)
```

```
// app/Models/Team.php
namespace App\Models;

use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Relations\BelongsToMany;

class Team extends Model
{
    protected $fillable = ['name'];
```

```
    public function users(): BelongsToMany
    {
        return $this->belongsToMany(User::class)
                    ->withPivot('role')
                    ->withTimestamps();
    }
}Code language: PHP (php)
```

3 – Assigning Roles with Team Context

When teams are enabled, you must always pass the team when assigning or checking roles and permissions. For example:

```
$team = Team::find(1);
$user = User::find(10);

// Assign role in the context of a team
$user->assignRole('admin', $team);

// Check role in a team
if ($user->hasRole('admin', $team)) {
    // user is an admin of this team
}

// Check permission in a team
if ($user->can('edit posts', $team)) {
    // user can edit posts, but only within this team
}Code language: PHP (php)
```

Without passing the team parameter, the check will fail, because roles and permissions are always scoped by team once this mode is enabled.

4 – Middleware for Team Access

To enforce team-based roles, we can create middleware that validates the user's role in the current team context.

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

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

use Closure;
use Illuminate\Http\Request;

class CheckTeamRole
{
    public function handle(Request $request, Closure $next, $role)
    {
        $team = $request->route('team');

        if (! $request->user()->hasRole($role, $team)) {
            abort(403, 'Unauthorized team access.');
        }

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

Now in your routes:

```
Route::middleware(['auth','team.role:admin'])->group(function () {
    Route::get('/teams/{team}/settings', function ($team) {
        return "Team Settings for team #{$team->id}";
    });
```

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

Only users with the `admin` role in that specific team can access the settings.

5 – UI for Managing Teams & Members

Finally, let's add a management UI where team admins can invite members and assign them roles.

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

Route::middleware(['auth'])->group(function () {
    Route::resource('teams', TeamController::class);
});Code language: PHP (php)
```

```
// app/Http/Controllers/TeamController.php
namespace App\Http\Controllers;

use App\Models\Team;
use App\Models\User;
use Illuminate\Http\Request;

class TeamController extends Controller
{
    public function show(Team $team)
    {
        $users = $team->users;
        return view('teams.show', compact('team','users'));
    }

    public function addMember(Request $request, Team $team)
    {
```

```
        $user = User::where('email',$request->email)->firstOrFail();
        $user->assignRole('member', $team);
        return back()->with('status','Member added!');
    }

    public function updateMemberRole(Request $request, Team $team,
User $user)
    {
        $user->syncRoles([$request->role], $team);
        return back()->with('status','Role updated!');
    }
}Code language: PHP (php)
```

Example Blade view (`resources/views/teams/show.blade.php`):

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

@section('content')
<div class="container">
  <h2>{{ $team->name }} Members</h2>
  <ul>
    @foreach($users as $user)
      <li>
        {{ $user->name }} - Role: {{
$user->roles->pluck('name')->first() }}
        <form method="POST" action="{{ route('teams.updateMemberRole',
[$team, $user]) }}">
          @csrf
          @method('PUT')
          <select name="role">
            <option value="member">Member</option>
            <option value="admin">Admin</option>
          </select>
          <button type="submit" class="btn btn-sm btn-
primary">Update</button>
        </form>
      </li>
    @endforeach
```

```
  </ul>
</div>
@endsectionCode language: HTML, XML (xml)
```

This UI allows admins to add members and change their team-specific roles easily.

Wrapping Up

We built a **Team Management System in Laravel 12** using Spatie's team feature. We enabled team support, added migrations, scoped role checks to teams, created middleware for team access, and built a UI for managing team members and their roles. With this approach, users can belong to multiple teams with different roles in each — a must for SaaS applications.

What's Next

- How to Create a Multi-Level Role & Permission System in Laravel
- Laravel Middleware for Role-Based Route Protection
- Creating a Role-Specific Dashboard in Laravel 12