How to Give and Revoke Permissions to Users in Laravel

In a real-world Laravel app, you won't always grant access through roles alone. Sometimes a user needs a **specific permission** (e.g., "publish posts") without changing their whole role. In **Laravel 12** with **Spatie Permissions**, you can give and revoke permissions directly to users, inherit them via roles, and manage everything from a simple admin UI.

In this guide, you'll learn how to **grant and revoke permissions** in multiple ways: via code, via roles, and from a user-friendly admin interface. We'll also cover best practices, caching, and how to confirm permissions at runtime.

1 – Prerequisites

Make sure you've installed Spatie Laravel Permission, run its migrations, and added the `HasRoles` trait to your `User` model. If you're building a multi-tenant app, enable teams first so permissions are scoped per team.

If you need a quick seed for demo permissions:

```
// database/seeders/PermissionsSeeder.php
use Illuminate\Database\Seeder;
use Spatie\Permission\Models\Permission;

class PermissionsSeeder extends Seeder {
    public function run(): void {
        foreach (['create posts','edit posts','publish posts','delete
posts'] as $name) {
            Permission::firstOrCreate(['name' => $name]);
        }
    }
```

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

Run with: `php artisan db:seed --class=PermissionsSeeder`

2 – Granting & Revoking Permissions in Code

You can assign permissions *directly* to a user, or let users inherit permissions via **roles**. Direct assignment is useful for one-off exceptions; roles are best for groups of users.

```
// 2.1 — grant a permission directly
$user->givePermissionTo('publish posts');

// 2.2 — revoke a direct permission
$user->revokePermissionTo('publish posts');

// 2.3 — check permission
if ($user->can('publish posts')) {
    // allowed
}

// 2.4 — via roles
$role = \Spatie\Permission\Models\Role::firstOrCreate(['name' =>
'editor']);
$role->givePermissionTo(['create posts','edit posts','publish
posts']);
$user->assignRole('editor'); // user now inherits all role
permissionsCode language: PHP (php)
```

Notes: `givePermissionTo` and `revokePermissionTo` affect only the user's *direct* permissions. If a permission still appears, it may be inherited from a role. Use `syncPermissions([...])` to replace all direct permissions in one go.

Teams: If you've enabled teams in Spatie, always pass the team context:

```
$team = Team::find(1);

$user->givePermissionTo('publish posts', $team);
$user->revokePermissionTo('publish posts', $team);

if ($user->can('publish posts', $team)) {
    // allowed only within this team
}Code language: PHP (php)
```

3 – Route & Controller Protection (Using Permissions)

Use Spatie's `permission:<name>` middleware to protect routes. You can combine it with `auth` and roles as needed.

```
// routes/web.php

use App\Http\Controllers\PostPublishController;

Route::middleware(['auth','permission:publish posts'])->group(function
() {
    Route::post('/posts/{post}/publish',
PostPublishController::class)->name('posts.publish');
});Code language: PHP (php)
```

In controllers, you can add additional checks (`$this->authorize()` for policies, or `Gate::allows()`) if needed. With Laravel 12's `HasMiddleware` approach, you can even declare the permission at the controller level (static method).

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

use Illuminate\Http\Request;
use Illuminate\Routing\Controllers\HasMiddleware;
```

```
use Illuminate\Routing\Controllers\Middleware;

class PostPublishController extends Controller implements
HasMiddleware
{
    public static function middleware(): array
    {
        return [
            new Middleware('auth'),
            new Middleware('permission:publish posts'),
        ];
    }

    public function __invoke(Request $request)
    {
        // publish logic...
        return back()->with('status','Post published');
    }
}Code language: PHP (php)
```

With teams enabled, prefer a team-aware approach (e.g., custom middleware that checks `$user->can('publish posts', $team)` using the route's team parameter).

4 – Building a Permissions Management UI

Let's build an admin interface to give and revoke user permissions without writing code. We'll create routes, a controller, and a Blade view with checkboxes. This UI complements a roles UI (often both are needed).

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

Route::middleware(['auth','role:admin'])->group(function () {
```

```
    Route::get('/admin/users/{user}/permissions',
[UserPermissionController::class,
'edit'])->name('admin.users.permissions.edit');
    Route::put('/admin/users/{user}/permissions',
[UserPermissionController::class,
'update'])->name('admin.users.permissions.update');
});Code language: PHP (php)
```

Controller:

```
// app/Http/Controllers/Admin/UserPermissionController.php
namespace App\Http\Controllers\Admin;

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

class UserPermissionController extends Controller
{
    public function edit(User $user)
    {
        $permissions = Permission::orderBy('name')->get();
        $direct = $user->permissions->pluck('name')->toArray(); //
direct only
        $viaRoles =
$user->getPermissionsViaRoles()->pluck('name')->toArray();

        return view('admin.users.permissions',
compact('user','permissions','direct','viaRoles'));
    }

    public function update(Request $request, User $user)
    {
        $selected = $request->input('permissions', []); // array of
names

        // Replace all direct permissions with the submitted set
```

```
        $user->syncPermissions($selected);

        // Clear permission cache to reflect changes immediately
app()->make(\Spatie\Permission\PermissionRegistrar::class)->forgetCach
edPermissions();

        return back()->with('status','Permissions updated.');
    }
}Code language: PHP (php)
```

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

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

@section('content')
<div class="container">
  <h2 class="mb-3">Manage Permissions for {{ $user->name }}</h2>

  <div class="alert alert-info">
    <strong>Note:</strong> Permissions shown with a lock (🔒) are
inherited via roles. You can only change direct permissions here.
  </div>

  <form method="POST" action="{{
route('admin.users.permissions.update', $user) }}">
    @csrf
    @method('PUT')

    <div class="row">
      @foreach($permissions as $perm)
        @php
          $isDirect = in_array($perm->name, $direct);
          $isViaRole = in_array($perm->name, $viaRoles);
        @endphp

        <div class="col-md-4 mb-2">
          <div class="form-check">
            <input class="form-check-input"
```

```
                    type="checkbox"
                    name="permissions[]"
                    value="{{ $perm->name }}"
                    id="perm_{{ $perm->id }}"
                    {{ $isDirect ? 'checked' : '' }}
                    {{ $isViaRole ? 'disabled' : '' }}>

            <label class="form-check-label" for="perm_{{ $perm->id
}}">
              {{ $perm->name }} {!! $isViaRole ? '' : '' !!}
            </label>
          </div>
        </div>
      @endforeach
    </div>

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

This screen distinguishes between **direct permissions** (checkboxes you can toggle) and **role-inherited permissions** (locked). That way, admins don't accidentally remove permissions that come from roles.

Teams: When teams are enabled, pass the team explicitly (hidden input or resolved from the route) and use `$user->syncPermissions($selected, $team)`, `$user->permissions($team)`, etc., to scope everything per team.

5 – Common Gotchas & Troubleshooting

- **Permission cache:** Spatie caches permissions. After updates, run: `php artisan permission:cache-reset` or call the registrar's `forgetCachedPermissions()` as

shown.

- **Role vs direct:** If a permission persists after revoking, it might be inherited via a role. Use the UI to show which permissions are coming from roles.
- **Teams mode:** Always pass the `$team` parameter to `givePermissionTo`, `revokePermissionTo`, `can()`, `syncPermissions()` when `'teams' => true` is enabled.
- **Authorization in Blade:** Use `@can('publish posts')` and `@cannot` to show/hide actions.
- **Audit/logging:** Consider logging who granted/revoked what (Model events, Activity Log) for compliance.

Wrapping Up

You now have a complete workflow to **give and revoke permissions** in Laravel 12 using Spatie: via code for automation, and via a clean **admin UI** for non-developers. You learned how to protect routes with permission middleware, handle team-scoped permissions, and avoid common caching pitfalls. This approach scales from small apps to large, role-heavy systems.

What’s Next

- Laravel Spatie Permissions: Step-by-Step Installation & Setup
- Building a Role-Based Admin Panel in Laravel 12
- Creating a User-Friendly Roles & Permissions UI in Laravel