Laravel Authentication: How to Redirect Users After Login & Logout

After setting up authentication, one of the most common requirements is controlling **where users are redirected** after they log in or log out. By default, Laravel redirects users to `/home` after login and back to `/` after logout. But in real applications, you'll often want more control — such as sending admins to a dashboard, regular users to their profile page, or guests back to a landing page.

In this guide, we'll explore **how to customize Laravel 12 authentication redirects**. We'll cover login redirects, logout redirects, role-based redirection, and even protecting against infinite redirect loops. You'll see code samples and explanations to ensure you can implement this feature confidently.

1 – The Default Redirect (Login)

Laravel uses the `RedirectsUsers` trait inside the built-in `LoginController` to decide where to send users after login. By default, it points to `/home`.

```
// app/Http/Controllers/Auth/LoginController.php

protected $redirectTo = '/home';Code language: PHP (php)
```

You can change this to any route or path, for example:

```
protected $redirectTo = '/dashboard';Code language: PHP (php)
```

This is the simplest way to control post-login redirection globally for all users.

2 – Dynamic Redirects (Role-Based or Conditional)

Often, different users should land on different pages depending on their role. Instead of a static `$redirectTo`, you can override the `redirectTo()` method:

```
// app/Http/Controllers/Auth/LoginController.php

protected function redirectTo()
{
    if (auth()->user()->hasRole('admin')) {
        return '/admin/dashboard';
    }

    return '/user/profile';
}Code language: PHP (php)
```

Here, admins are redirected to an admin dashboard, while regular users land on their profile page. You can customize this based on roles, permissions, or even subscription status.

3 – Redirecting After Logout

By default, Laravel redirects to `/` after logout. To change this, you can override the `loggedOut` method in the `LoginController`:

```
// app/Http/Controllers/Auth/LoginController.php
```

```
protected function loggedOut(Request $request)
{
    return redirect('/goodbye');
}Code language: PHP (php)
```

This example sends users to a `/goodbye` page after logout. You can point this to a landing page, marketing page, or even a “logged out successfully” message.

4 – Redirecting Intended Users

Laravel includes a built-in “intended” feature that remembers where a guest tried to go before being redirected to the login page. After login, they’ll automatically go back there unless you override it. Example:

```
// app/Http/Controllers/Auth/LoginController.php

protected function authenticated(Request $request, $user)
{
    return redirect()->intended('/dashboard');
}Code language: PHP (php)
```

If no intended page exists, users will land on `/dashboard` instead. This ensures seamless UX — especially when protecting routes with `auth` middleware.

5 – Protecting Against Redirect Loops

Be careful not to redirect users to a route that itself requires authentication. For example, if you redirect logged-out users back to `/dashboard`, they'll just hit the login screen again in a loop.

Always ensure your logout redirect points to a public page, like `/`, `/goodbye`, or `/thanks`.

Wrapping Up

You now know how to **customize Laravel 12 authentication redirects** for both login and logout. We saw how to set a global redirect, create dynamic role-based redirects, override logout behavior, and use Laravel's intended feature for smarter flows. With these techniques, you can tailor the authentication flow to match your app's UX perfectly.

What's Next

- How to Restrict Page Access by Role in Laravel 12 — redirect users away from pages they can't access.
- Implementing Passwordless Authentication in Laravel 12 — modernize login UX even further.
- Implementing Two-Factor Authentication in Laravel — add stronger login security.