# Laravel Fortify 2FA Example: Enable, Challenge, Recovery Codes (Step by Step)

Laravel Fortify provides a headless authentication backend, including built-in Two-Factor Authentication (2FA) with time-based one-time passwords (TOTP). In this guide, you'll install and configure Fortify, enable 2FA, build minimal Blade views for enabling/disabling 2FA, display QR codes and recovery codes, handle the two-factor challenge at login, wire useful events, and test the flow end-to-end.

## Install & Register Laravel Fortify

```bash
composer require laravel/fortify
```
Code language: Bash (bash)

This installs Fortify into your Laravel app. Fortify exposes authentication routes and actions (login, logout, 2FA enable/disable, challenges) without generating UI scaffolding.

```bash
php artisan vendor:publish --provider="Laravel\Fortify\FortifyServiceProvider"
```
Code language: Bash (bash)

Publishing copies the Fortify configuration file, migrations, and language lines to your project so you can customize them (including the 2FA-related columns).

```php
// config/app.php (ensure provider is registered if not auto-discovered)
'providers' => [
    // ...
    App\Providers\FortifyServiceProvider::class,
],
```
Code language: PHP (php)

Fortify is typically registered via your own `App\Providers\FortifyServiceProvider` so you can define views and behaviors. If you don't have it, create and register it as above.

```
php artisan migrate
```
Code language: Bash (bash)

Run migrations to ensure 2FA columns exist on the users table. The published migration adds `two_factor_secret`, `two_factor_recovery_codes`, and timestamps needed for 2FA.

## Enable Two-Factor Authentication in Fortify

```
// config/fortify.php
use Laravel\Fortify\Features;

return [
    // ...
    'features' => [
        Features::registration(),
        Features::resetPasswords(),
        Features::emailVerification(),
        Features::twoFactorAuthentication([
            'confirmPassword' => true,
        ]),
    ],
];
```
Code language: PHP (php)

Enabling `Features::twoFactorAuthentication()` activates Fortify's 2FA endpoints: enabling/disabling 2FA, generating recovery codes, and challenging users during login when 2FA is active.

```
// app/Providers/FortifyServiceProvider.php
namespace App\Providers;

use Illuminate\Support\ServiceProvider;
use Laravel\Fortify\Fortify;
```

```
class FortifyServiceProvider extends ServiceProvider
{
    public function boot(): void
    {
        // Point Fortify to your custom Blade views:
        Fortify::loginView(fn() => view('auth.login')); // your
existing login
        Fortify::twoFactorChallengeView(fn() => view('auth.two-factor-
challenge'));
        // You can set other views (register, reset, etc.) as needed.
    }
}
```
Code language: PHP (php)

Fortify is “headless”, so you must provide the login and two-factor challenge views. We will create a minimal set of views next.

## Profile UI: Enable / Disable 2FA + Show QR & Recovery Codes

Fortify exposes signed-in endpoints for enabling/disabling 2FA and regenerating recovery codes. Here’s a simple Blade “Profile Security” section to manage 2FA on the frontend.

```
<!-- resources/views/profile/security.blade.php -->
@extends('layouts.app')

@section('content')
  <h2>Two-Factor Authentication</h2>

  @if (! auth()->user()->two_factor_secret)
    <form method="POST" action="/user/two-factor-authentication">
      @csrf
      <button type="submit">Enable 2FA</button>
    </form>
  @else
    <p>2FA is enabled on your account.</p>

    <h3>Scan this QR code in your authenticator app</h3>
    {!! auth()->user()->twoFactorQrCodeSvg() !!}

    <h3 class="mt-3">Recovery Codes</h3>
    <ul>
      @foreach (auth()->user()->recoveryCodes() as $code)
        <li><code>{{ $code }}</code></li>
      @endforeach
    </ul>

    <form method="POST" action="/user/two-factor-recovery-codes">
      @csrf
      <button type="submit">Regenerate Recovery Codes</button>
    </form>

    <form method="POST" action="/user/two-factor-authentication">
      @csrf
      @method('DELETE')
      <button type="submit" class="mt-3">Disable 2FA</button>
    </form>
  @endif
@endsectionCode language: PHP (php)
```

When 2FA is disabled, the form posts to `/user/two-factor-authentication` to enable it. Once enabled, users see a QR SVG (scan with Google Authenticator, 1Password, Authy, etc.) and recovery codes. They can regenerate codes or disable 2FA via the provided forms.

## Two-Factor Challenge View (Login Step)

After a successful password login for a user with 2FA enabled, Fortify redirects to a challenge page to enter the TOTP code or a recovery code. Create this Blade view and wire it in your `FortifyServiceProvider` as shown earlier.

```
<!-- resources/views/auth/two-factor-challenge.blade.php -->
@extends('layouts.guest')

@section('content')
  <h1>Two-Factor Challenge</h1>

  <form method="POST" action="/two-factor-challenge">
    @csrf

    <div>
      <label>Authentication Code</label>
      <input type="text" name="code" inputmode="numeric"
autocomplete="one-time-code">
    </div>

    <p>Or use a recovery code:</p>

    <div>
      <label>Recovery Code</label>
      <input type="text" name="recovery_code">
    </div>

    <button type="submit">Verify</button>

    @error('code') <p class="text-danger">{{ $message }}</p> @enderror
    @error('recovery_code') <p class="text-danger">{{ $message }}</p>
@enderror
  </form>
@endsectionCode language: PHP (php)
```

Posting to `/two-factor-challenge` tells Fortify to validate either the 6-digit code from the authenticator app or a recovery code, completing the login flow.

## Useful Events: Email Users When 2FA Changes

Fortify fires events when users enable/disable 2FA or regenerate recovery codes. You can listen to these and notify users for security awareness.

```
// app/Providers/EventServiceProvider.php
protected $listen = [
    \Laravel\Fortify\Events\TwoFactorAuthenticationEnabled::class => [
        \App\Listeners\SendTwoFactorEnabledNotification::class,
    ],
    \Laravel\Fortify\Events\TwoFactorAuthenticationDisabled::class =>
[
        \App\Listeners\SendTwoFactorDisabledNotification::class,
    ],
    \Laravel\Fortify\Events\RecoveryCodesGenerated::class => [
\App\Listeners\SendRecoveryCodesRegeneratedNotification::class,
    ],
];
```
Code language: PHP (php)

Registering listeners lets you send mail, Slack/Log notifications, or audit events whenever 2FA settings change.

```
// app/Listeners/SendTwoFactorEnabledNotification.php
namespace App\Listeners;

use Illuminate\Support\Facades\Mail;
use Laravel\Fortify\Events\TwoFactorAuthenticationEnabled;

class SendTwoFactorEnabledNotification
{
    public function handle(TwoFactorAuthenticationEnabled $event):
void
    {
        $user = $event->user;
        Mail::raw('Two-Factor Authentication was enabled on your 
account.', function ($m) use ($user) {
            $m->to($user->email)->subject('2FA Enabled');
        });
    }
}
```
Code language: PHP (php)

This simple listener sends an email whenever a user enables 2FA. You can create similar listeners for disabled and regenerated codes to keep users informed.

## Controller Integration: Protect Critical Actions with Password/2FA

Even with 2FA enabled, you might want to require recent password confirmation (and therefore 2FA at login) before sensitive actions (like deleting an account). Fortify ships a password confirmation route you can require via middleware.

```
// routes/web.php
Route::middleware(['auth', 'password.confirm'])->group(function () {
    Route::delete('/account', 
[\App\Http\Controllers\AccountController::class, 'destroy'])
        ->name('account.destroy');
});
```
Code language: PHP (php)

Using `password.confirm` ensures the user recently re-entered their password (and has passed 2FA on login). You can also build a custom flow to ask for a fresh TOTP if you prefer a second check before a destructive action.

## Feature Test: Happy Path for 2FA Challenge

This example shows how to simulate a user with 2FA enabled and verify that the two-factor challenge gate works. In practice, you can stub the verification logic or seed a valid TOTP using a known secret.

```
// tests/Feature/TwoFactorLoginTest.php
namespace Tests\Feature;

use Tests\TestCase;
use App\Models\User;
use Illuminate\Foundation\Testing\RefreshDatabase;

class TwoFactorLoginTest extends TestCase
{
    use RefreshDatabase;

    public function test_user_with_2fa_is_redirected_to_challenge():
void
    {
        $user = User::factory()->create([
            // Pretend 2FA is enabled by seeding secret/recovery
fields:
            'two_factor_secret' => encrypt('TESTSECRET'),
            'two_factor_recovery_codes' =>
encrypt(json_encode(['recovery-code-1'])),
        ]);

        // First step: password login (simulate posting valid
credentials)
        $response = $this->post('/login', [
            'email' => $user->email,
            'password' => 'password', // matches default factory
        ]);

        $response->assertRedirect('/two-factor-challenge');

        // Second step: submit a recovery code (bypassing TOTP for
test)
        $challenge = $this->post('/two-factor-challenge', [
            'recovery_code' => 'recovery-code-1',
        ]);

        $challenge->assertRedirect('/home'); // or your intended
location
        $this->assertAuthenticatedAs($user);
    }
}Code language: PHP (php)
```

The test verifies that a user with 2FA enabled is redirected to the challenge after password login, and that providing a valid recovery code authenticates them fully.

## Troubleshooting & Notes

- **QR not showing?** Ensure you've run the vendor publish & migrations, and that your user has a generated secret after enabling 2FA. The `twoFactorQrCodeSvg()` helper renders only when 2FA is enabled.
- **Time drift errors?** TOTP is time-based: make sure your server clock is accurate (use NTP) so codes match authenticator apps.
- **Lost device?** Users can sign in with a recovery code and immediately regenerate new recovery codes from the profile screen.
- **Security hardening:** Consider emailing users on 2FA changes (examples above) and auditing those events.

These tips help ensure a smooth 2FA experience and keep your app's authentication flow secure and user-friendly.

## Wrapping Up

With Laravel Fortify, adding 2FA is straightforward: enable the feature, provide minimal UI for enabling/disabling and challenges, and wire event listeners for better security hygiene. The built-in helpers for QR codes and recovery codes make UX smooth, while middleware like `password.confirm` protects sensitive operations. You now have a production-ready baseline for strong, user-friendly 2FA in Laravel.

## What's Next

Keep strengthening your auth stack with these related guides:

- Implementing Two-Factor Authentication in Laravel
- How to Build Email Verification in Laravel 12 (Step by Step)
- Implementing Password Reset in Laravel 12 Without Packages