**Opción 1: Implementación Manual**

1. **Crear el sistema de autenticación**:
   * Crea las rutas necesarias en routes/web.php:

php

Copiar código

use App\Http\Controllers\Auth\LoginController;

use App\Http\Controllers\Auth\RegisterController;

Route::get('/login', [LoginController::class, 'showLoginForm'])->name('login');

Route::post('/login', [LoginController::class, 'login']);

Route::post('/logout', [LoginController::class, 'logout'])->name('logout');

Route::get('/register', [RegisterController::class, 'showRegistrationForm'])->name('register');

Route::post('/register', [RegisterController::class, 'register']);

* + Crea los controladores LoginController y RegisterController si no existen ya:

bash

Copiar código

php artisan make:controller Auth/LoginController

php artisan make:controller Auth/RegisterController

* + En LoginController, implementa las funciones showLoginForm(), login(), y logout():

php

Copiar código

namespace App\Http\Controllers\Auth;

use App\Http\Controllers\Controller;

use Illuminate\Http\Request;

use Illuminate\Support\Facades\Auth;

class LoginController extends Controller

{

public function showLoginForm()

{

return view('auth.login');

}

public function login(Request $request)

{

$credentials = $request->only('email', 'password');

if (Auth::attempt($credentials)) {

return redirect()->intended('/');

}

return back()->withErrors([

'email' => 'Las credenciales no coinciden.',

]);

}

public function logout()

{

Auth::logout();

return redirect('/login');

}

}

* + En RegisterController, implementa las funciones showRegistrationForm() y register():

php

Copiar código

namespace App\Http\Controllers\Auth;

use App\Http\Controllers\Controller;

use App\Models\User;

use Illuminate\Http\Request;

use Illuminate\Support\Facades\Hash;

class RegisterController extends Controller

{

public function showRegistrationForm()

{

return view('auth.register');

}

public function register(Request $request)

{

$request->validate([

'name' => 'required|string|max:255',

'email' => 'required|string|email|max:255|unique:users',

'password' => 'required|string|min:8|confirmed',

]);

User::create([

'name' => $request->name,

'email' => $request->email,

'password' => Hash::make($request->password),

]);

return redirect('/login');

}

}

1. **Crear las vistas personalizadas**:
   * Crea las vistas de login y registro en resources/views/auth/login.blade.php y resources/views/auth/register.blade.php utilizando tu diseño actual con Tailwind CSS.
2. **Configurar la autenticación en Laravel**:
   * Asegúrate de que las migraciones para los usuarios estén realizadas:

bash

Copiar código

php artisan migrate

**Opción 2: Usar Laravel Fortify**

Si prefieres un enfoque basado en paquetes sin tocar tu diseño, puedes usar **Laravel Fortify**. Este paquete ofrece autenticación sin proporcionar interfaces prediseñadas, lo que te permite controlar completamente la UI.

1. **Instalar Laravel Fortify**:

bash

Copiar código

composer require laravel/fortify

1. **Publicar la configuración**:

bash

Copiar código

php artisan vendor:publish --provider="Laravel\Fortify\FortifyServiceProvider"

1. **Habilitar Fortify en config/fortify.php**:

php

Copiar código

'features' => [

Features::registration(),

Features::resetPasswords(),

Features::emailVerification(),

Features::updateProfileInformation(),

Features::updatePasswords(),

Features::twoFactorAuthentication(),

],

1. **Registrar Fortify en App\Providers\FortifyServiceProvider**:

php

Copiar código

public function boot()

{

Fortify::loginView(function () {

return view('auth.login');

});

Fortify::registerView(function () {

return view('auth.register');

});

}

1. **Crear las vistas de autenticación**:
   * Crea las vistas auth.login y auth.register en resources/views/auth/, utilizando tu propio CSS y componentes Livewire.

Esto te permitirá implementar un sistema de autenticación sin afectar tu CSS de Tailwind ni la configuración de Livewire.