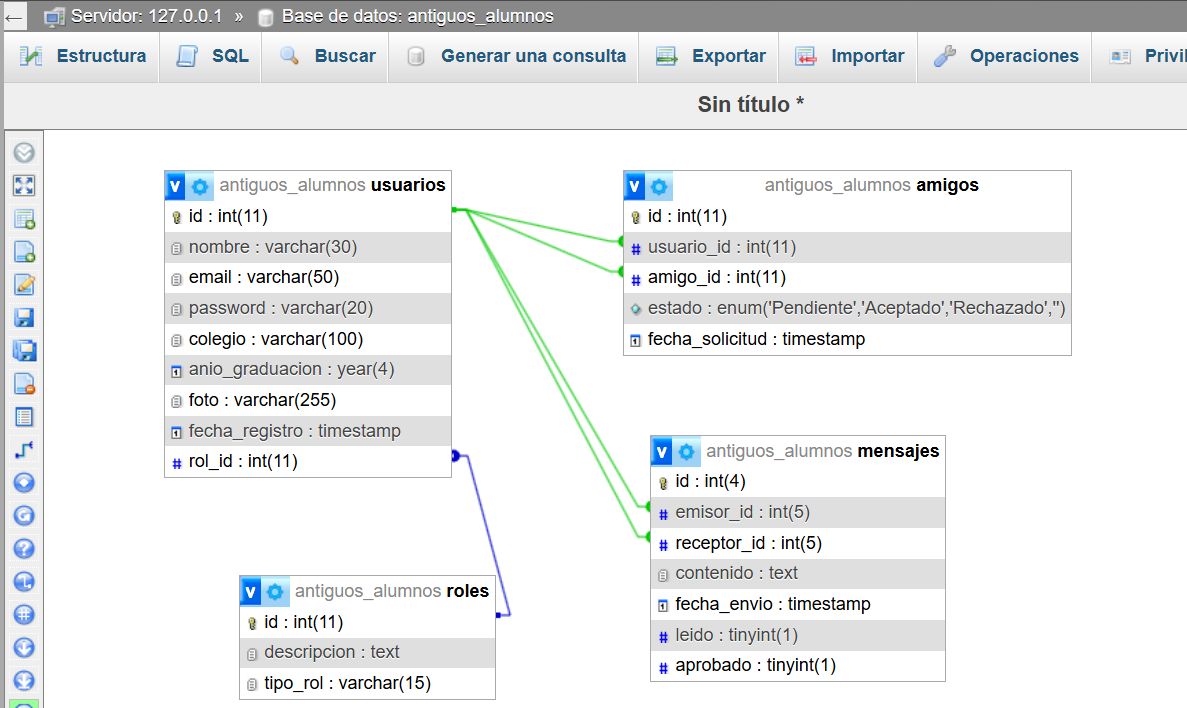
PROYECTO ANTIGUOS ALUMNOS

# Primeros pasos en el diseño

1. Diseño de la base de datos



Comienzo el proyecto del tfg consistente en el desarrollo y diseño de una aplicación web, a través de la cual se puedan encontrar compañeros de colegios e institutos.

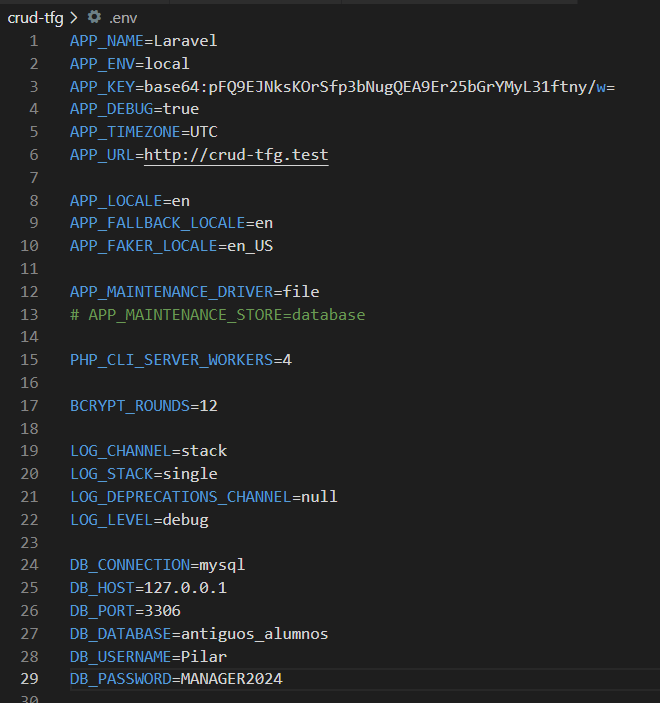
1. Entorno de desarrollo

El proyecto lo voy a realizar usando el framework Laravel. Laravel Herd es una herramienta de entorno de desarrollo creada por el equipo oficial de Laravel, que admite el cambio de versiones de PHP y Node. Después de instalar Herd, sirve a todos los sitios de tu máquina a través de dominios \*.test: es como Laravel Valet, pero no tiene dependencias y no necesita Homebrew. Viene con sus propios binarios precompilados, lo que hace que sea increíblemente rápido de instalar y usar.

Instalé Herd y comencé a trabajar con VSC desde sus archivos.

## Rutas

Los valores de configuración importantes en Laravel se definen mediante el archivo .env que existe en la raíz de nuestra aplicación. Así lo hice y quedó de la siguiente manera.



Donde entre otras cosas, señalé la conexión a la bbdd, datos de usuario y pasword.

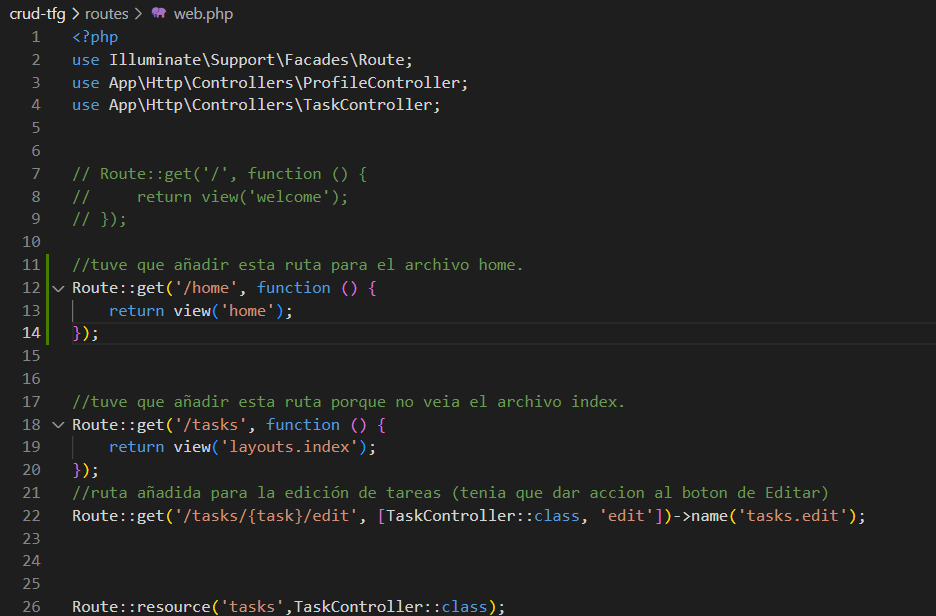
Una vez que ya tuve configurada la base de datos en MySql, pude ejecutar las migraciones de base de datos de mi aplicación. Que se harán mediante el código -php artisan migrate-

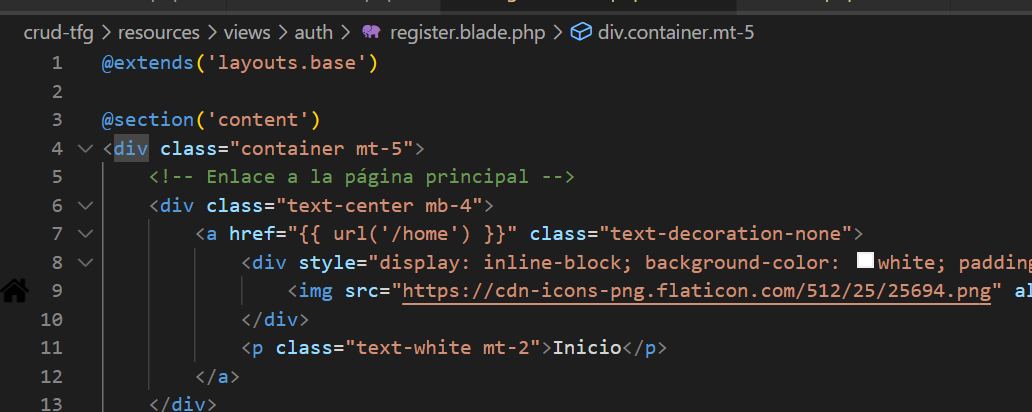
Lo primero que tuve que familiarizarme fue con el diseño de las rutas en Laravel. Todas las rutas de Laravel se definen en tus archivos de ruta, que se encuentran en el directorio routes. Estos archivos se cargan automáticamente mediante el App\Providers\RouteServiceProvider de tu aplicación. El archivo routes/web.php define rutas para tu interfaz web. Estas rutas se asignan al grupo de middleware web, que proporciona funciones como el estado de sesión y la protección CSRF.

Para la mayoría de las aplicaciones, comenzarás definiendo rutas en tu archivo routes/web.php. Las rutas definidas en routes/web.php se pueden acceder ingresando la URL definida de la ruta en tu navegador.

Nota: Cuando defines varias rutas que comparten la misma URI, las rutas que utilizan los métodos get, post, put, patch, delete y options deben definirse antes que las rutas que utilizan los métodos any, match y redirect. Esto asegura que la solicitud entrante coincida con la ruta correcta.

Comencé a definir las primeras rutas tal y como detallo en la imagen siguiente:

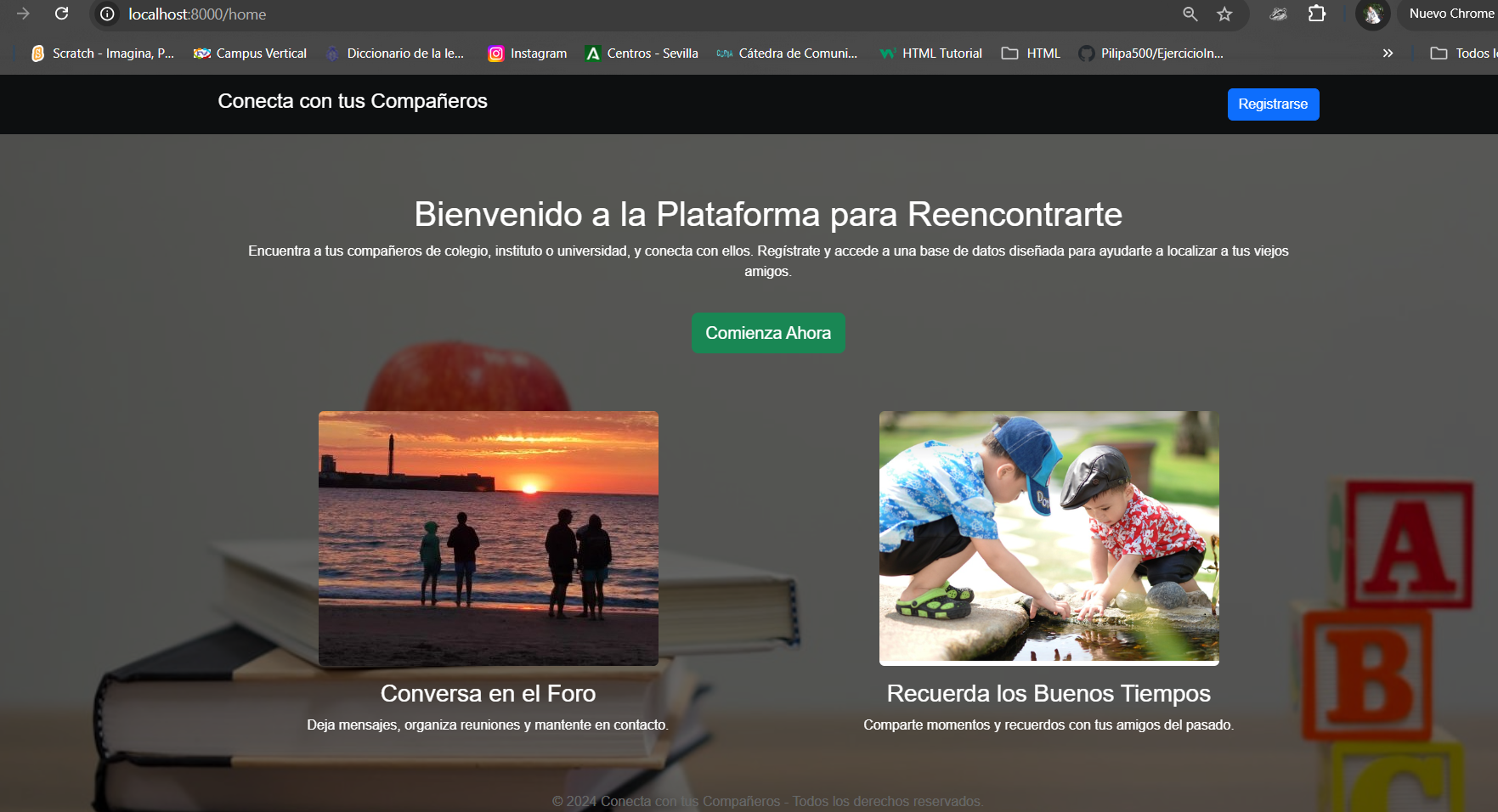


En Laravel se puede usar la directiva de Blade @extends para especificar qué diseño debe “heredar” la vista secundaria. En mi caso diseñé un archivo base.blade.php que me servía como de plantilla para el resto de vistas. Las vistas que extienden un diseño de Blade pueden inyectar contenido en las secciones del diseño mediante directivas @section.

Y realicé el archivo que sería mi pantalla inicial de la aplicación [home.blade.php](http://home.blade.php).

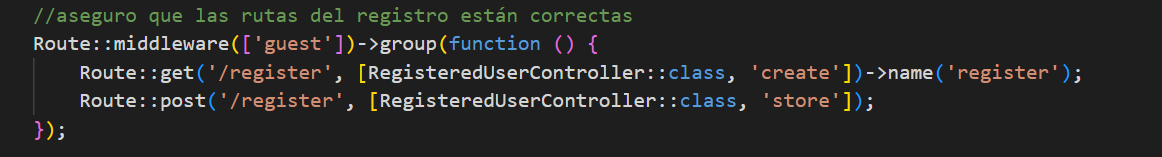
Blade es el motor de plantillas simple pero potente incluido con Laravel. A diferencia de algunos motores de plantillas PHP, Blade no te restringe a usar código PHP simple en tus plantillas. De hecho, todas las plantillas de Blade se compilan en código PHP simple y se almacenan en caché hasta que se modifican, lo que significa que Blade agrega prácticamente cero sobrecarga a tu aplicación. Los archivos de plantilla de Blade usan la extensión de archivo .blade.php y generalmente se almacenan en el directorio resources/views.

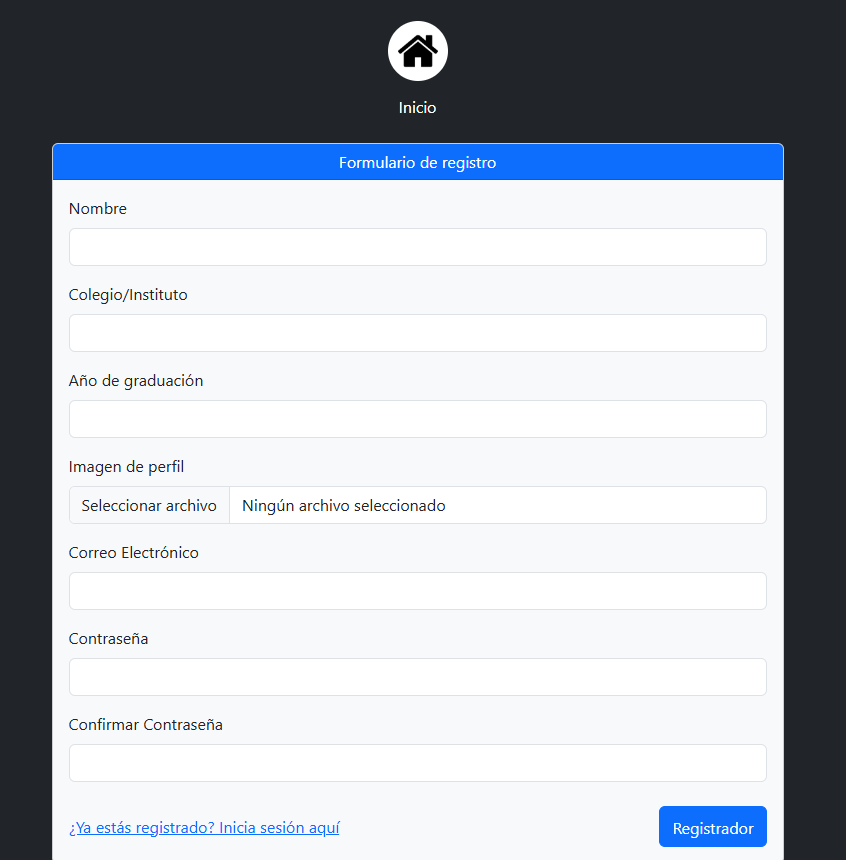
En esa dirección generé el archivo [home.blade.php](http://home.blade.php) donde se detallaba el “porqué” de esta web.



En esta primera página se describe brevemente como desde esta app podremos encontrar a los compañeros con lo que hubiéramos estados unidos en el pasado, por medio de nuestros centros de estudios. Con un registro inicial, estaremos insertándonos en la base de datos, desde la que podremos formar parte de esta comunidad de amigos.

Los dos botones que hemos puesto aquí, tanto el de “Registrarse” como el de “Comienza Ahora”, dirigen al formulario de registro, donde se iniciará el proceso de acceso.

Antes de editar el formulario de registro, nos vamos al archivo de web.php donde, como ya hemos explicado más arriba detallamos las rutas de todo nuestro proyecto 

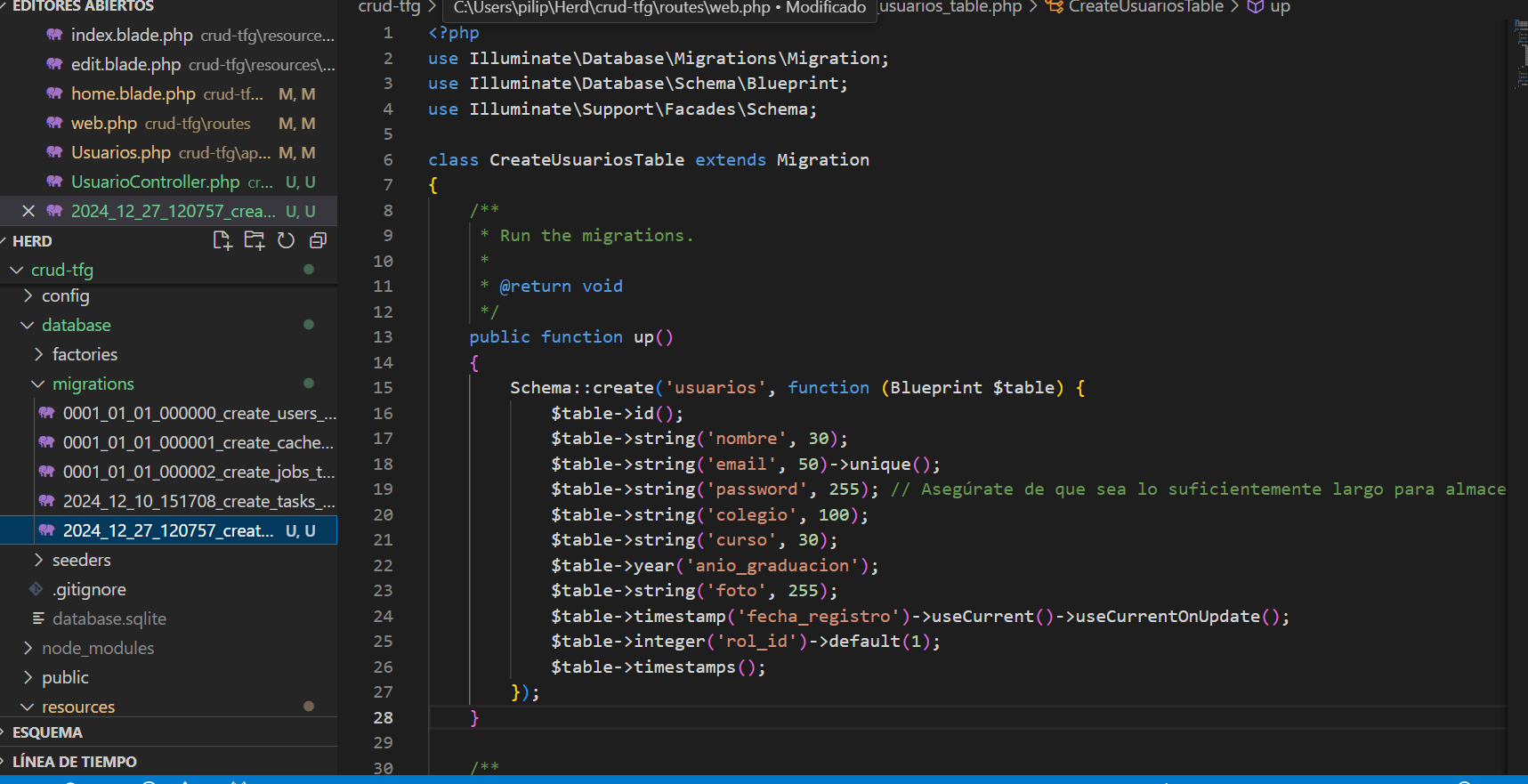
En este caso damos las rutas para que un usuario invitado pueda crear y almacenar su registro, rellenando el formulario al que se dirige desde el home. 

Como se pudo observar en la imagen del diseño de la bbdd que tiene este proyecto, había creado la “tabla usuarios” con la que alineamos las entradas del formulario, para que se insertaran correctamente.

Me aseguré de que la migración de la tabla usuarios, estuviera correctamente definida en Laravel. Y cree una migración con el comando php artisan make: migration create\_usuarios\_table

Abrí el archivo de migración recién creado en database/migrations/.

Y dentro del método up, definí las columnas según la estructura de la base de datos. Aquí dejo el ejemplo basado en la imagen del diseño de la bbdd:

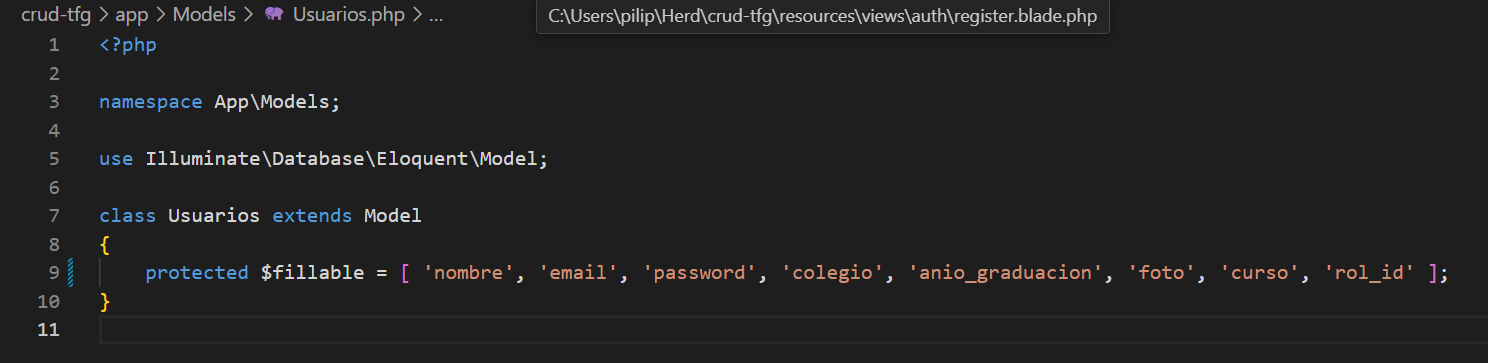


y definí las columnas según la estructura de la base de datos en App/Models creando el archivo usuarios.php.

**Ejecutar la Migración**:

Una vez que he definido las columnas, ejecuto la migración con el siguiente comando: php artisan migrate

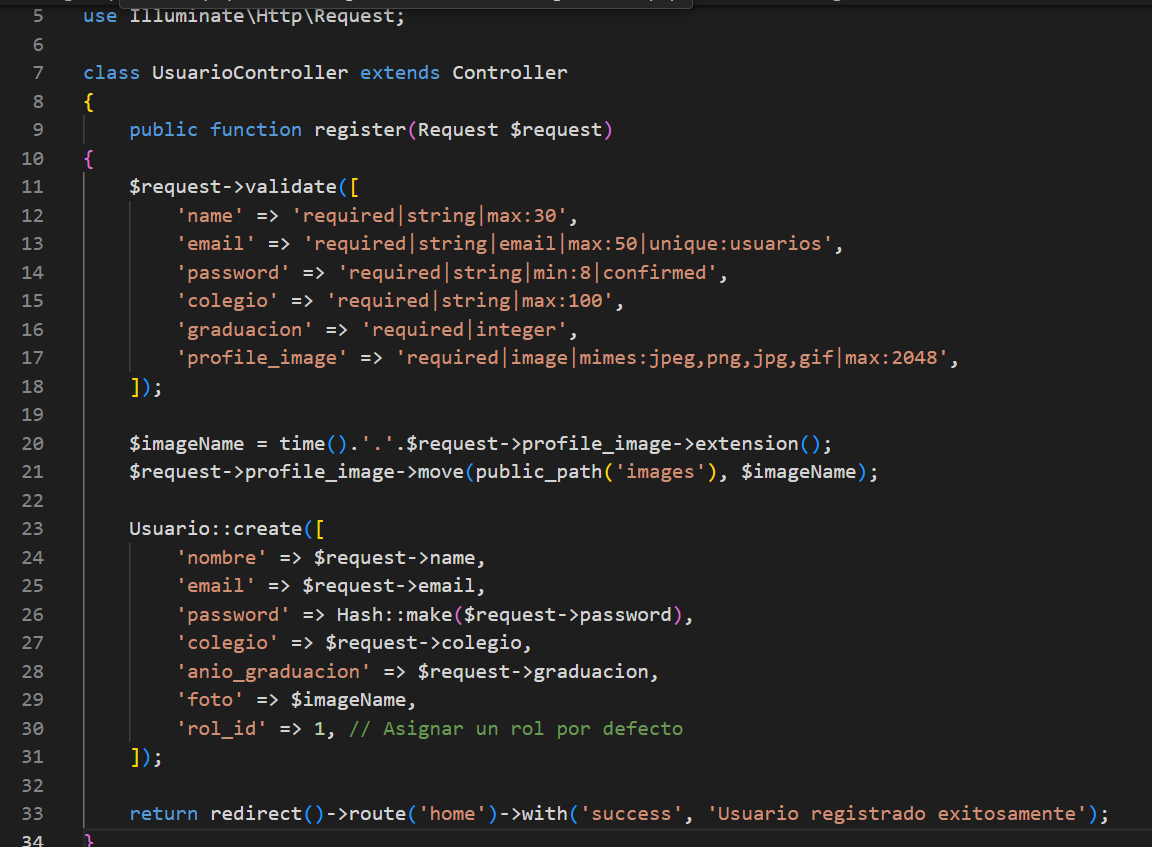
Esto creará la tabla usuarios en la base de datos con las columnas especificadas.



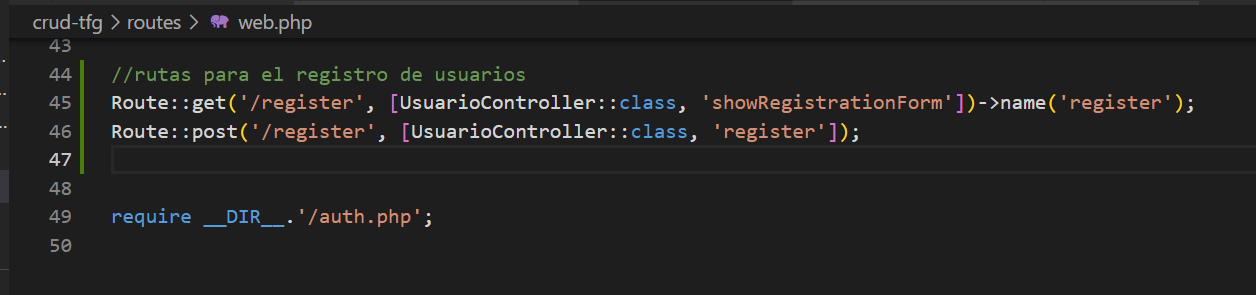
De igual manera, tendré que hacer lo mismo para las demás tablas que vaya a usar. Como serán las que se ven en la bbdd de más arriba. Como son: Amigos, roles y mensajes. Así que voy a generar ya los archivos con el comando php artisan make: migration create\_ejemplotabla\_table y los tendré ya escritos para cuando los vaya a usar.

Controlador:

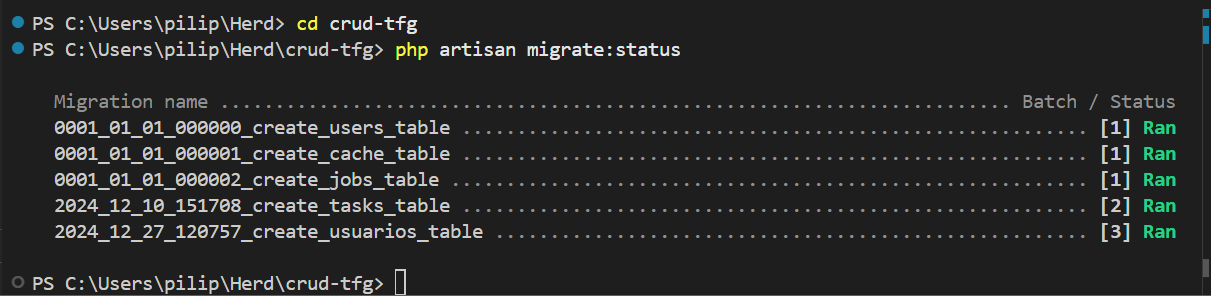
A continuación creamos un controlador para manejar el registro de usuarios con el comando php artisan make:controller UsuarioController. En el método de registro, valida y guarda los datos del formulario:



A continuación se definen las rutas necesarias en web.php

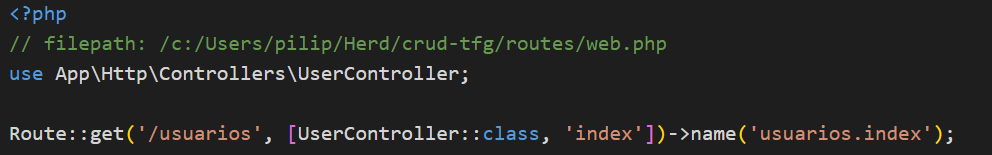


Al añadir usuarios registrados, observé que “aparentemente”, parecían guardarse, pero no lo veía añadido en la bbdd de MySQl. Así que seguí, comprobando que la conexión con la bbdd.

 Que aunque aparecía “corriendo” en el terminal, no lo veía claro. No salían los nuevos usuarios en la tabla de MySQL.

Así que decidí, **crear una nueva vista (nuevo archivo también)** para listar los usuarios que se van añadiendo a la plataforma de AAA.

1. Creé una ruta para la vista de usuarios:

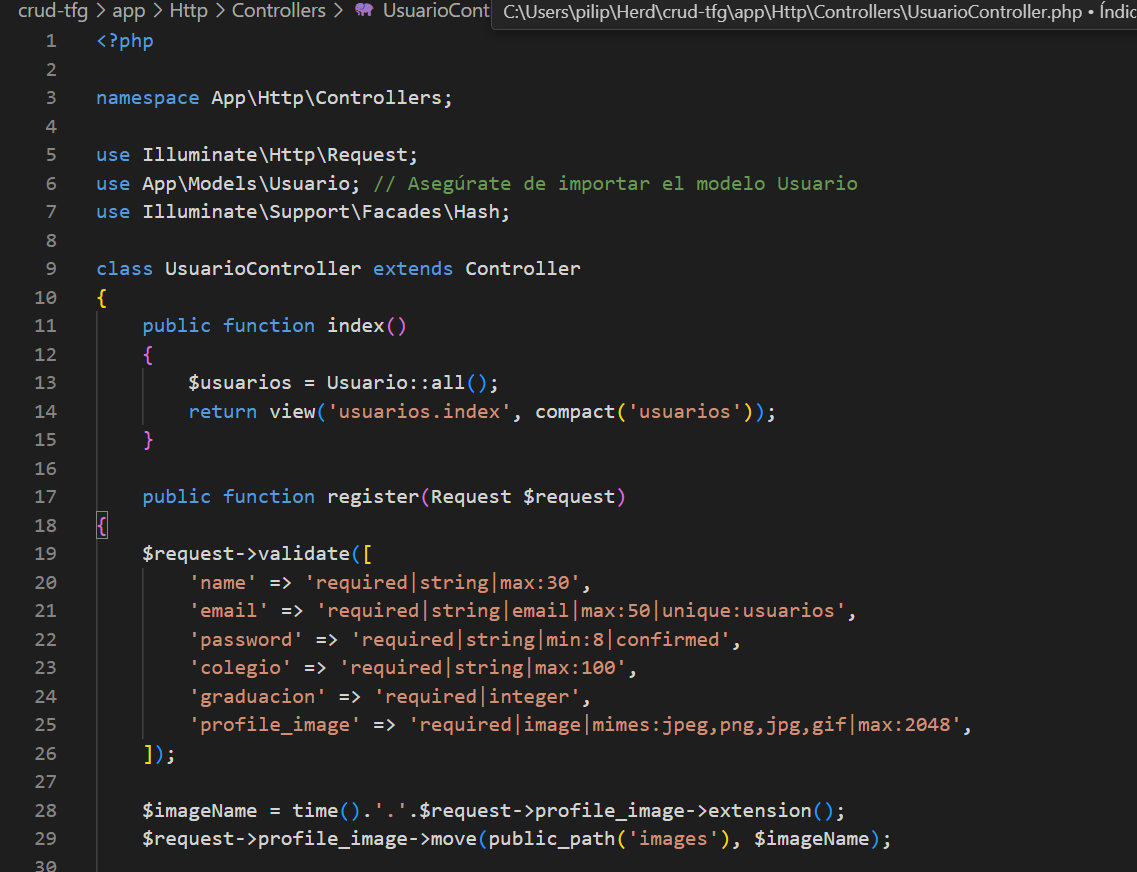


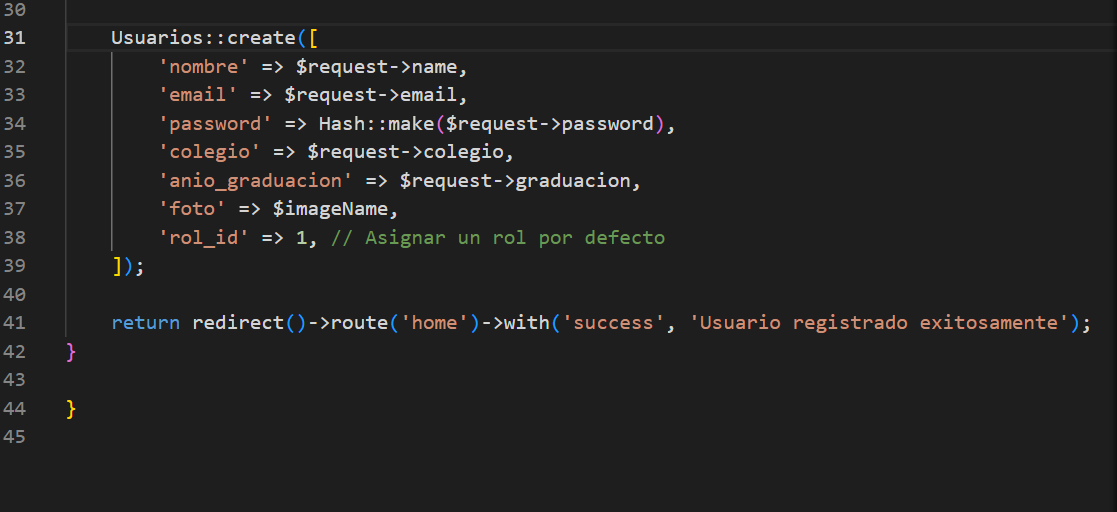
El archivo completo está así, actualmente.



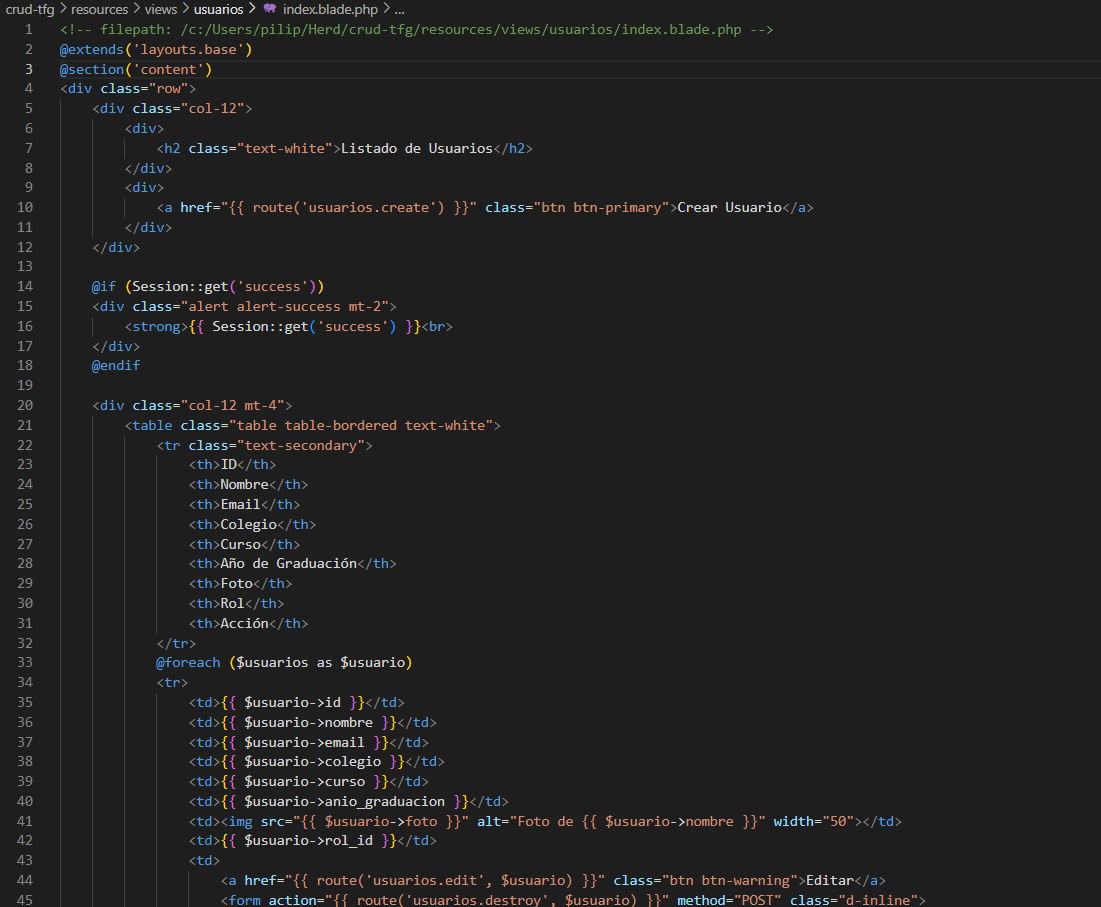


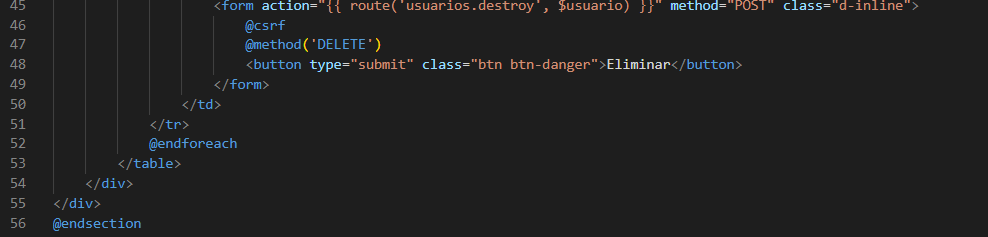
1. Utilicé el mismo `UsuarioController` para añadir el código necesario para listar los usuarios. Solo necesite añadir el método `index` y asegurarme de que el modelo `Usuario` esté importado correctamente. Aquí puse como lo hice:





Crear la vista para listar los usuarios. Hice un nuevo archivo de vista index.blade.php en la carpeta resources/views/usuarios.

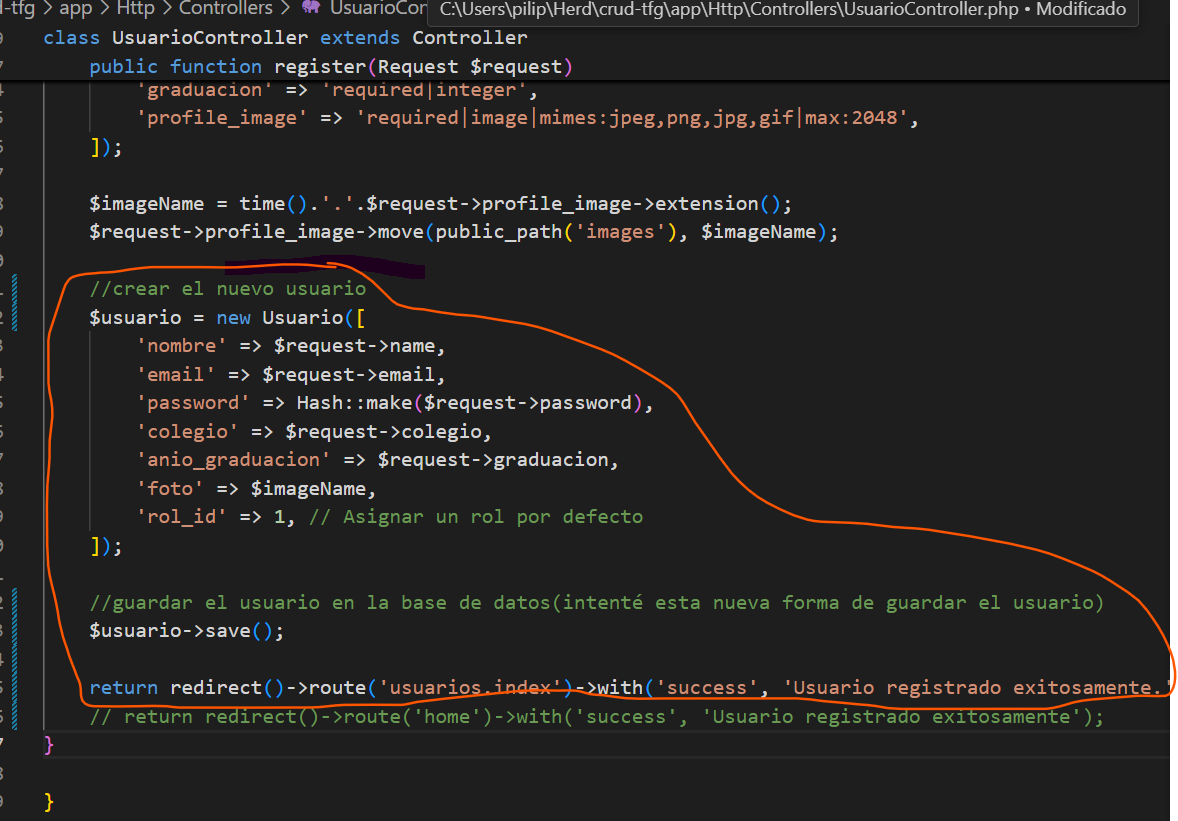




Ahora debería poder acceder a la vista de usuarios visitando la URL /usuarios en mi navegador.

PERO NO ES ASÍ!!! ALGO ESTÁ PASANDO Y NO SÉ QUE ES!!!

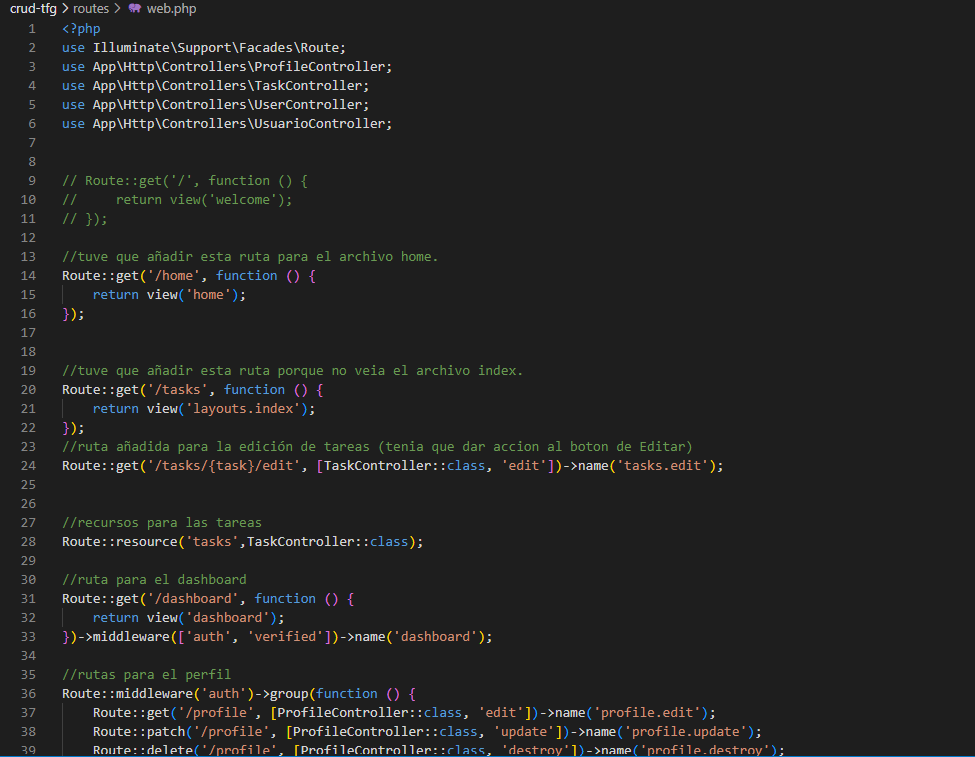
Se están guardando los usuarios en la tabla user de la bbdd y no en la de usuarios, que es la que destiné para el registro. A pesar de que está corriendo y activada, según el terminal. Pero al intentar abrir la url /usuarios no se abre el listado que he hecho.

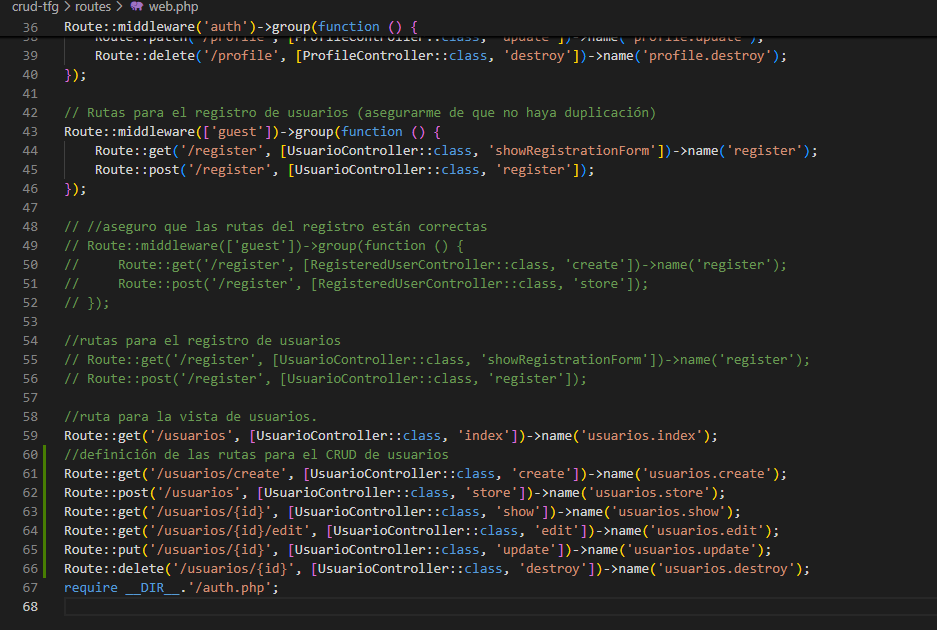
Cambié el controlador en esta parte del código

Y volví activar el servidor local. La realidad es que a cada poco debo hacerlo. En el terminal ejecuto en la dirección del proyecto

php -S localhost:8000 -t public y reactivo el servidor.

Pensé que para poder ver la vista de usuarios, debía añadir en las dirección web, también su vista y toda la definición de las rutas para el CRUD de usuarios, por el administrador. Con lo que el archivo quedó así en C:\Users\pilip\Herd\crud-tfg\routes\web.php:





Al pensar hacer un CRUD, parecido al de “task” guardé las posibles rutas que iría creando, para el perfil del “administrador” de la plataforma.

Ahora modifiqué el archivo de UsuarioController.php. Donde tuve que añadir los métodos correspondientes para manejar las rutas, que acababa de crear:

namespace App\Http\Controllers;

use Illuminate\Http\Request;

use App\Models\Usuarios;

use Illuminate\Support\Facades\Hash;

class UsuarioController extends Controller

{

public function index()

{

$usuarios = Usuarios::all();

return view('usuarios.index', compact('usuarios'));

}

public function create()

{

return view('usuarios.create');

}

public function store(Request $request)

{

$request->validate([

'name' => 'required|string|max:30',

'email' => 'required|string|email|max:50|unique:usuarios',

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

'colegio' => 'required|string|max:100',

'graduacion' => 'required|integer',

'profile\_image' => 'required|image|mimes:jpeg,png,jpg,gif|max:2048',

]);

$imageName = time().'.'.$request->profile\_image->extension();

$request->profile\_image->move(public\_path('images'), $imageName);

$usuario = new Usuarios([

'nombre' => $request->name,

'email' => $request->email,

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

'colegio' => $request->colegio,

'anio\_graduacion' => $request->graduacion,

'foto' => $imageName,

'rol\_id' => 1,

]);

$usuario->save();

return redirect()->route('usuarios.index')->with('success', 'Usuario registrado exitosamente.');

}

public function show($id)

{

$usuario = Usuarios::findOrFail($id);

return view('usuarios.show', compact('usuario'));

}

public function edit($id)

{

$usuario = Usuarios::findOrFail($id);

return view('usuarios.edit', compact('usuario'));

}

public function update(Request $request, $id)

{

$request->validate([

'name' => 'required|string|max:30',

'email' => 'required|string|email|max:50|unique:usuarios,email,'.$id,

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

'colegio' => 'required|string|max:100',

'graduacion' => 'required|integer',

'profile\_image' => 'nullable|image|mimes:jpeg,png,jpg,gif|max:2048',

]);

$usuario = Usuarios::findOrFail($id);

if ($request->hasFile('profile\_image')) {

$imageName = time().'.'.$request->profile\_image->extension();

$request->profile\_image->move(public\_path('images'), $imageName);

$usuario->foto = $imageName;

}

$usuario->nombre = $request->name;

$usuario->email = $request->email;

if ($request->password) {

$usuario->password = Hash::make($request->password);

}

$usuario->colegio = $request->colegio;

$usuario->anio\_graduacion = $request->graduacion;

$usuario->save();

return redirect()->route('usuarios.index')->with('success', 'Usuario actualizado exitosamente.');

}

public function destroy($id)

{

$usuario = Usuarios::findOrFail($id);

$usuario->delete();

return redirect()->route('usuarios.index')->with('success', 'Usuario eliminado exitosamente.');

}

}