

10/01/2026 - 20:18 a 21:58

Instalación de laragon

creación de un proyecto con laragon :

The screenshot shows the Laragon application window. At the top, it displays "Laragon 2025 v8.4.0 251210 [default] php-8.3.28 [TS] 192.168.56.1 C:\laragon". Below the title bar, there's a menu icon and some status information: "Menú", "[Unlicensed] [default]", and icons for help and settings. The main area shows services: Apache 2.4.62 (status: 80), MySQL 8.4.3 (status: 3306), and Mailpit 1.22.3 (status: 1025/8025). A "Recargar" button is also present. A dropdown menu is open under "www", showing options like "Laragon", "Apache", "MySQL", and "Mailpit". A sub-menu for "Apache" is open, showing "Blank", "WordPress", "Laravel" (which is selected), "Laravel CLI", and "Symfony". Below this, a modal dialog titled "Creación rápida de sitio web" asks for the "Nombre del Proyecto [Laravel]:" and has a text input field containing "Foodsynk". It also has "OK" and "Cancel" buttons. At the bottom, a terminal window shows the command "C:\laragon\www\Foodsynk>code ." and the message "y abre VScode".

***** NOTE: Now, you can use pretty url for your awesome project :) *****

(Laragon) Project path: C:/laragon/www/Foodsynk
(Laragon) Pretty url: http://Foodsynk.test

C:\laragon\www\Foodsynk>code .
y abre VScode

Para que me instale los paquetes necesito :

```
C:\laragon\www\Foodsynk>npm install .
```

```
added 83 packages, and audited 84 packages in 15s  
21 packages are looking for funding  
  run `npm fund` for details  
  
found 0 vulnerabilities
```

Instalación de los paquetes de composer, que son los paquetes necesarios para el uso de laravel.

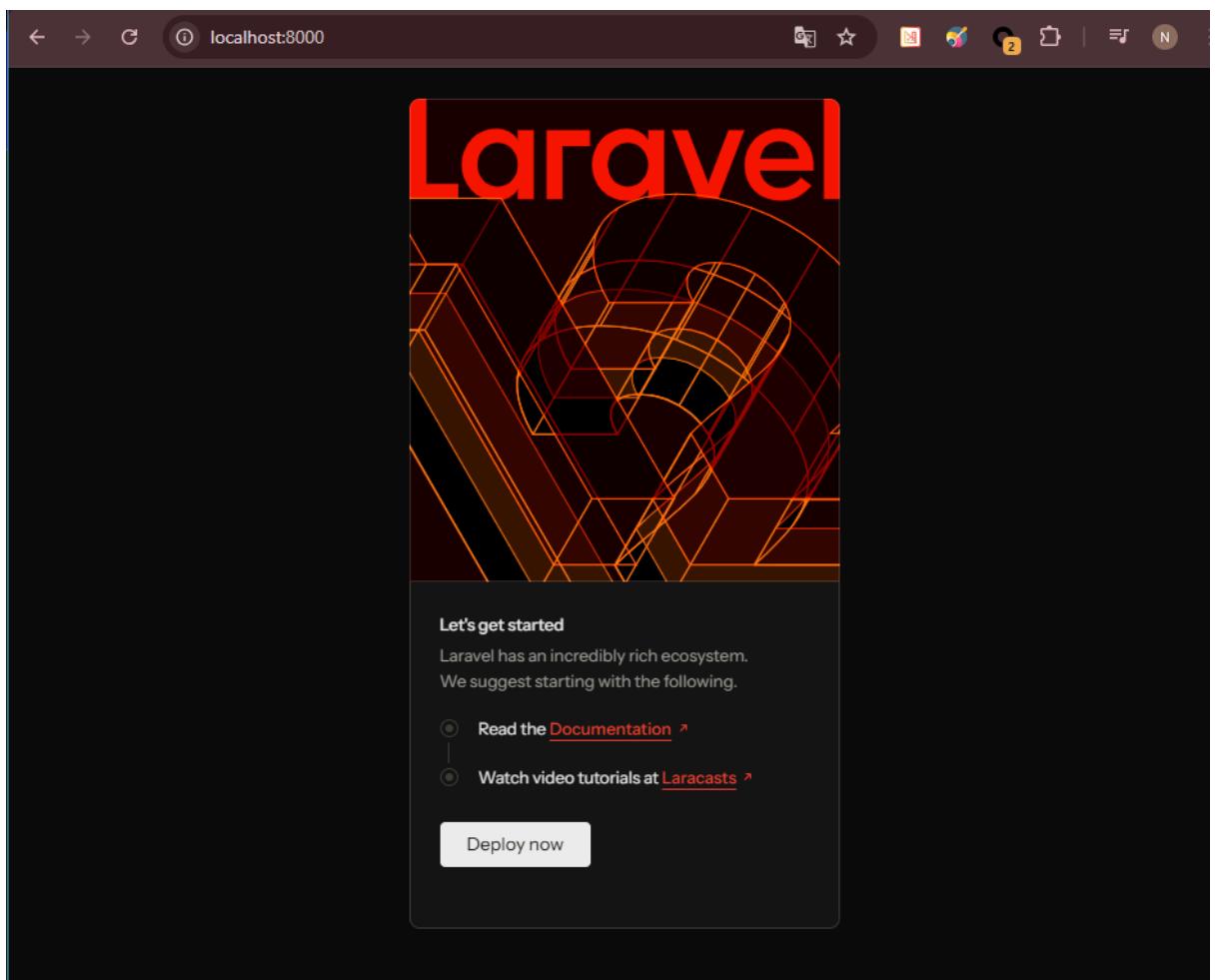
```
C:\laragon\www\Foodsynk>composer require  
lo dejamos vacío  
C:\laragon\www\Foodsynk>composer require  
Search for a package:  
The package you required is recommended to be placed in require-dev.  
Do you want to re-run the command with --dev? [yes]? yes  
php artisan package:discover --ansi  
INFO Discovering packages.  
  
laravel/pail .. DONE  
laravel/sail .. DONE  
laravel/tinker .. DONE  
nesbot/carbon .. DONE  
nunomaduro/collision .. DONE  
nunomaduro/termwind .. DONE
```

Necesitamos una clave para levantar el proyecto.

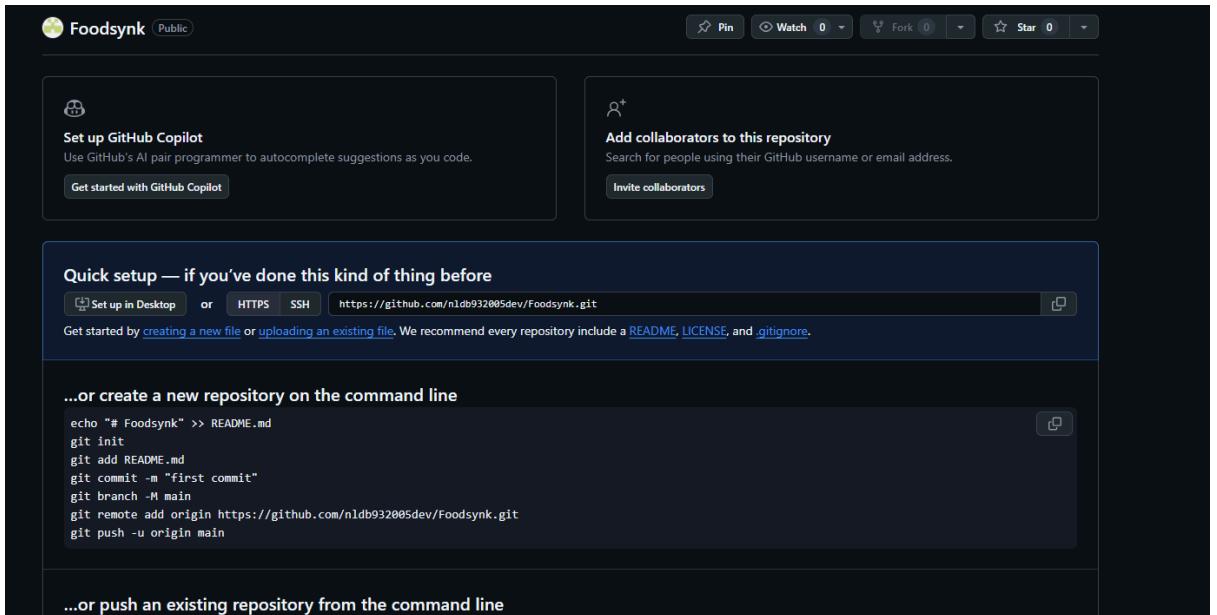
```
C:\laragon\www\Foodsynk>php artisan key:generate  
INFO Application key set successfully.
```

levantar el proyecto:

```
C:\laragon\www\Foodsynk>php artisan serve  
INFO Server running on [http://127.0.0.1:8000].  
Press Ctrl+C to stop the server
```



Creamos el repositorio de git hub



dentro de mi carpeta de proyecto en local, ejecuto:

```
PS C:\laragon\www\Foodsynk> git init
Initialized empty Git repository in C:/laragon/www/Foodsynk/.git/
```

para iniciar el proyecto.

```
PS C:\laragon\www\Foodsynk> git commit -m "subir proyecto a git"
```

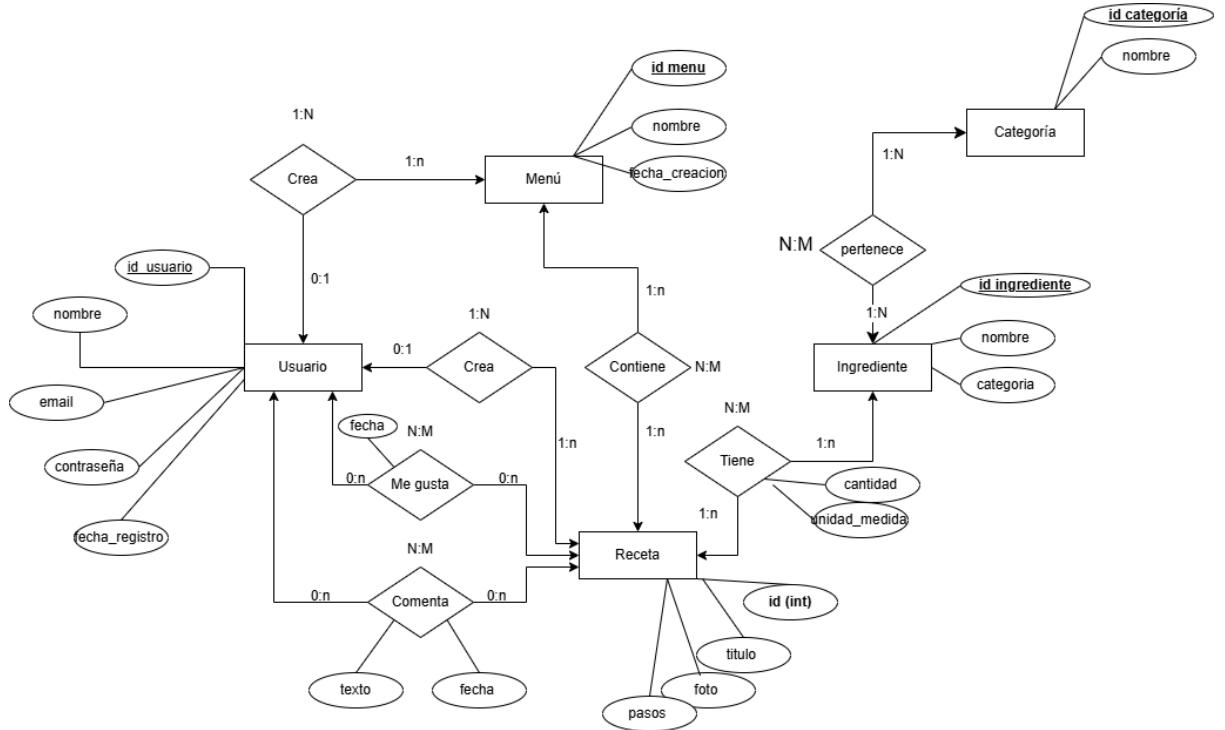
un primer commit para orientarnos.

Vamos a conectar el proyecto con una base de datos. Para eso, nos vamos al ".env" del proyecto y descomentamos las líneas de la bd.

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=laravel
DB_USERNAME=root
DB_PASSWORD=
```

Revisé la BD y me di cuenta de que había un atributo que había que cambiar a entidad. Esto se verá reflejado en el documento oficial del proyecto dónde se habla de la bd.

Pero en resumen es qué ingredientes tenía categorías, pero como queremos que sea escalable nos viene mejor una entidad.



Las tablas son:

Usuario(ID_Usuario(**PK**), nombre, email (**UQ**), contraseña, fecha_registro)

Receta(ID_Receta(**PK**), título, foto, pasos, ID_Usuario(**FK**))

Ingrediente(ID_Ingrediente(**PK**), nombre)

Categoría (ID_Categoría(**PK**), nombre)

Menú(ID_Menu(**PK**), nombre, fecha_creación, ID_Usuario(**FK**))

Receta_Ingrediente(cantidad, unidad_medida, ID_Receta(**FK**)(**PK**), ID_Ingrediente(**FK**)(**PK**))

Menu_Receta(ID_Menu(**FK**)(**PK**), ID_Receta(**FK**)(**PK**))

Me_gusta(fecha, ID_Usuario(**FK**)(**PK**), ID_Receta(**FK**)(**PK**))

Comentario(texto, fecha, ID_Usuario(**FK**)(**PK**), ID_Receta(**FK**)(**PK**))

Ingrediente_Categoría(ID_categoría(**PK**)(**FK**), ID_categoría(**PK**)(**FK**))

Y el orden a crear las migraciones es importantes, por eso el orden de las tablas debe ser:

Usuario(ID_Usuario(**PK**), nombre, email (UQ), contraseña, fecha_registro)
Ingrediente(ID_Ingrediente(**PK**), nombre)
Categoría (ID_Categoría(**PK**),nombre)
Receta(ID_Receta(**PK**), título, foto, pasos, ID_Usuario(**FK**))
Menú(ID_Menu(**PK**), nombre, fecha_creación, ID_Usuario(**FK**))
Receta_Ingrediente(cantidad, unidad_medida, ID_Receta(**FK**)(**PK**),
ID_Ingrediente(**FK**)(**PK**))
Menu_Receta(ID_Menu(**FK**)(**PK**), ID_Receta(**FK**)(**PK**))
Me_gusta(fecha, ID_Usuario(**FK**)(**PK**), ID_Receta(**FK**)(**PK**))
Comentario(texto, fecha, ID_Usuario(**FK**)(**PK**), ID_Receta(**FK**)(**PK**))
Ingrediente_Categoría(ID_categoría(**PK**)(**FK**),ID_categoría(**PK**)(**FK**))

Y ahora vamos a probar a hacer una migración que creé una tabla:

php artisan make:migration create_X_table

```
C:\laragon\www\Foodsynk>php artisan make:migration create_ingredients_table
INFO Migration [C:\laragon\www\Foodsynk\database\migrations\2026_01_10_195858_create_ingredients_table.php] created successfully.
```

Tenemos que ver la migración en nuestro proyecto, dentro de la carpeta database > migrations. Y añadirle los campos que queramos. Para esto me ayudé de la IA.

```
database      ●
  > factories
  < migrations ●
    ▾ 0001_01_01_000000...
    ▾ 0001_01_01_000001...
    ▾ 0001_01_01_000002...
    ▾ 2026_01_10_1... U
  > seeders
  .gitignore
  database.sqlite
  > node_modules
  > public
  > resources
4   use Illuminate\Database\Schema\Blueprint;
5   use Illuminate\Support\Facades\Schema;
6
7   return new class extends Migration
8   {
9     /**
10      * Run the migrations.
11      */
12     public function up(): void
13     {
14       Schema::create('ingredients', function (Blueprint $table): void {
15         $table->id();
16         $table->string('nombre')->unique();
17         $table->timestamps();
18       });

```

ahora para que impacte en la bd tenemos que :

```
:\\laragon\\www\\Foodsynk>php artisan migrate
WARN The database 'laravel' does not exist on the 'mysql' connection.
Would you like to create it? (yes/no) [yes]
yes
INFO Preparing database.
Creating migration table ..... 24.59ms DONE
INFO Running migrations.

0001_01_01_000000_create_users_table ..... 56.45ms DONE
0001_01_01_000001_create_cache_table ..... 15.17ms DONE
0001_01_01_000002_create_jobs_table ..... 50.16ms DONE
2026_01_10_195858_create_ingredients_table ..... 7.99ms DONE

:\\laragon\\www\\Foodsynk>
```

Ahora vamos a instalar laravel breeze en el repositorio para usar sanctum para proteger nuestras apis y ayudarnos con la tabla usuarios y lo que sería en un futuro la gestión de usuarios.

Para ello vamos a:

composer require laravel/breeze --dev (*dev es importante para dejar claro que es una herramienta únicamente de desarrollo y que cuando se poné en producción la aplicación no se instala el paquete) (esto en la cmd de laragon)*

```
C:\laragon\www\Foodsynk>composer require laravel/breeze --dev
/composer.json has been updated
running composer update laravel/breeze
Loading composer repositories with package information
Updating dependencies
Lock file operations: 1 install, 0 updates, 0 removals
- Locking laravel/breeze (v2.3.8)
Writing lock file
Installing dependencies from lock file (including require-dev)
Package operations: 1 install, 0 updates, 0 removals
- Downloading laravel/breeze (v2.3.8)
- Installing laravel/breeze (v2.3.8): Extracting archive
Generating optimized autoload files
-> Illuminate\Foundation\ComposerScripts::postAutoloadDump
-> @php artisan package:discover --ansi
```

```
C:\laragon\www\Foodsynk>php artisan breeze:install
Which Breeze stack would you like to install?
Blade with Alpine ..... blade
Livewire (Volt Class API) with Alpine ..... livewire
Livewire (Volt Functional API) with Alpine ..... livewire-functional
React with Inertia ..... react
Vue with Inertia ..... vue
API only ..... api
```

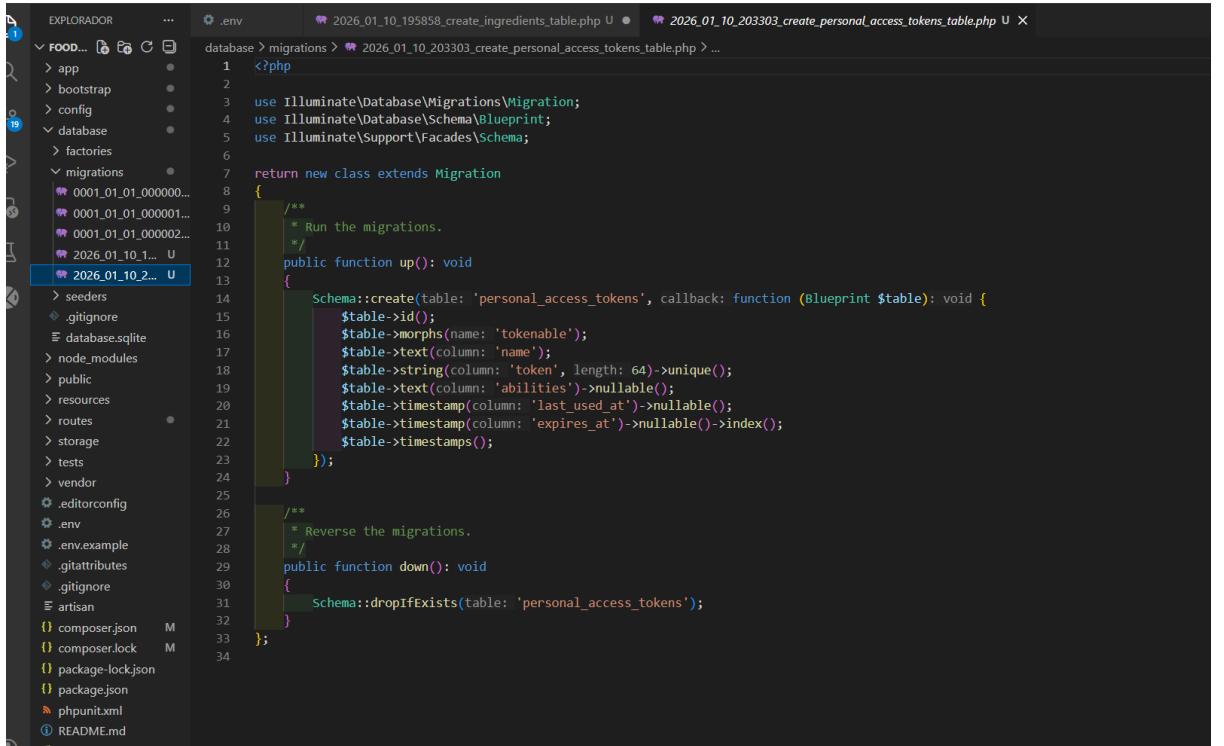
Aquí te hace unas preguntas:

```
C:\laragon\www\Foodsynk>php artisan breeze:install
Which Breeze stack would you like to install?
Blade with Alpine ..... blade
Livewire (Volt Class API) with Alpine ..... livewire
Livewire (Volt Functional API) with Alpine ..... livewire-functional
React with Inertia ..... react
Vue with Inertia ..... vue
API only ..... api
api
Which testing framework do you prefer? [Pest]
Pest ..... 0
PHPUnit ..... 1
0
0
```

Elegí en este caso la api, porque nosotros vamos a usar únicamente laravel para el back y react para el front.

Con esto debería habernos creado en el proyecto una nueva migración que es importante para la seguridad de nuestras rutas, porque esto es lo que generará el

token de autenticación.



The screenshot shows a code editor with a file named `2026_01_10_203303_create_personal_access_tokens_table.php` open. The file contains PHP code for a database migration. It uses the `Schema::create` method to define a table named `personal_access_tokens`. The table has columns for `id`, `tokenable` (morphed), `name`, `token` (unique and nullable), `abilities` (text), `last_used_at` (timestamp), and `expires_at` (timestamp). The `down` method is also defined to drop the table if it exists.

```
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create('personal_access_tokens', function (Blueprint $table): void {
            $table->id();
            $table->morphs('name: "tokenable"');
            $table->text('name');
            $table->string('token', 64)->unique();
            $table->text('abilities')->nullable();
            $table->timestamp('last_used_at')->nullable();
            $table->timestamp('expires_at')->nullable()->index();
            $table->timestampt();
        });
    }

    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        Schema::dropIfExists('personal_access_tokens');
    }
};
```

Y también te va a generar un archivo llamado api dentro de la carpeta routes.



The screenshot shows a file named `api.php` with the following content:

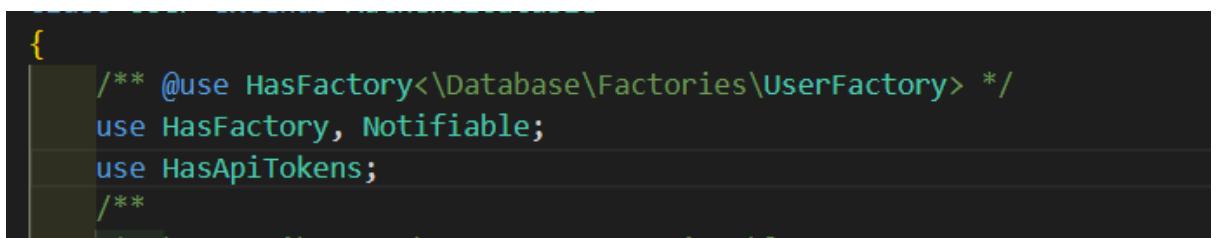
```
<?php

use Illuminate\Http\Request;
use Illuminate\Support\Facades\Route;

Route::middleware(['auth:sanctum'])->get('/user', function (Request $request): mixed {
    return $request->user();
});
```

Dentro del modelo users se guarda toda la información relacionada con la entidad usuarios de la BD.(app>models> user.php) Dentro del archivo tenemos que añadir la línea :

```
use HasApiTokens;
```



The screenshot shows a file named `User.php` with the following code:

```
{
    /**
     * @use HasFactory<\Database\Factories\UserFactory> */
    use HasFactory, Notifiable;
    use HasApiTokens;
    /**
```

para que vincule los tokens con los usuarios.

Por último tenemos que ejecutar la migración de los tokens que se nos han creado.

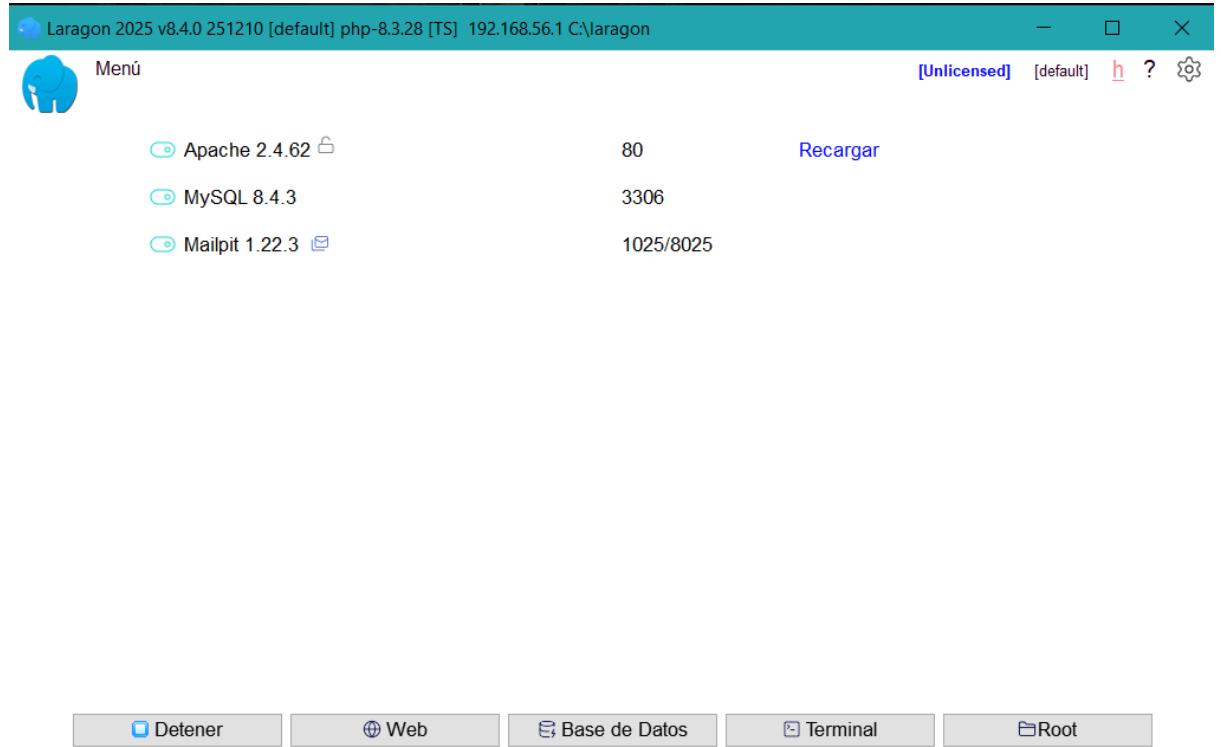
```
C:\laragon\www\Foodsynk>php artisan migrate
[INFO] Nothing to migrate.

C:\laragon\www\Foodsynk>php artisan migrate:status
Migration name ..... Batch / Status
0001_01_01_000001_create_users_table ..... [1] Ran
0001_01_01_000001_create_cache_table ..... [1] Ran
0001_01_01_000002_create_jobs_table ..... [1] Ran
2026_01_10_195858_create_ingredients_table ..... [1] Ran
2026_01_10_203303_create_personal_access_tokens_table ..... [2] Ran

C:\laragon\www\Foodsynk>
```

Y acabamos el día subiendo los cambios a nuestro repositorio de github.

12/1/26 19:40-22:03



```
C:\laragon\www
λ cd Foodsynk\  
  
C:\laragon\www\Foodsynk(main -> origin)
λ php artisan serve  
  
INFO Server running on [http://127.0.0.1:8000].  
  
Press Ctrl+C to stop the server
```

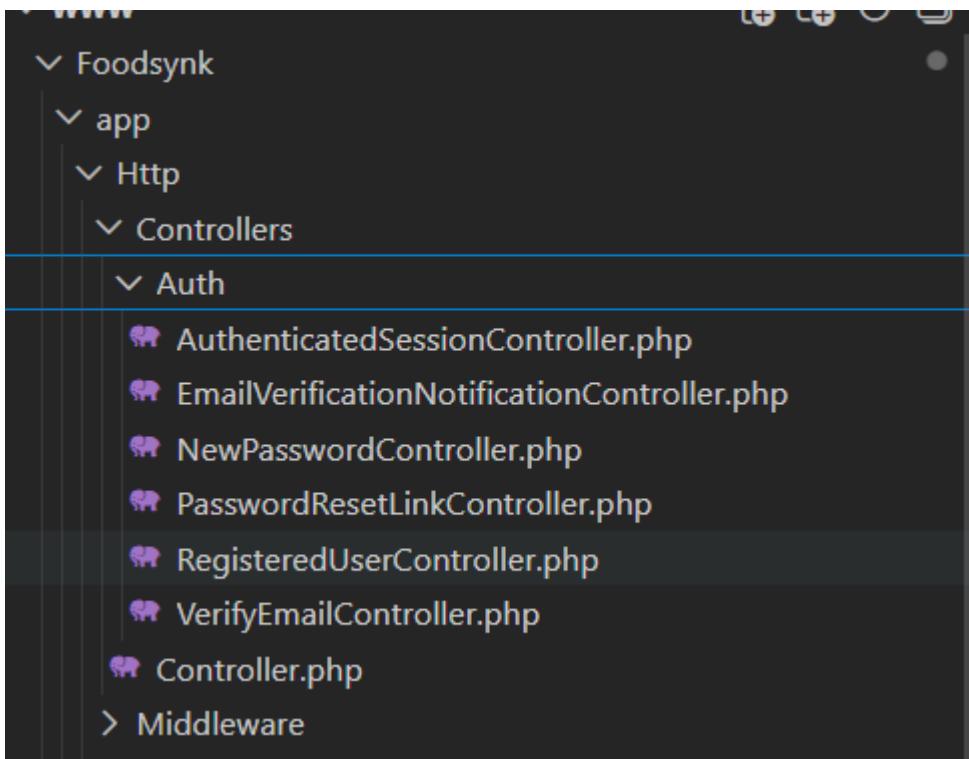
php artisan serve para levantar el proyecto.

php artisan breeze:install api para instalar la api para que los endpoints nos lo devuelva en formato json.

```
C:\laragon\www\Foodsynk(main -> origin)
λ php artisan breeze:install api
./composer.json has been updated
    Running composer update laravel/sanctum
Loading composer repositories with package information
    Updating dependencies
Nothing to modify in lock file
Writing lock file
Installing dependencies from lock file (including require-dev)
Nothing to install, update or remove
Generating optimized autoload files
    > Illuminate\Foundation\ComposerScripts::postAutoloadDump
    > @php artisan package:discover --ansi

INFO Discovering packages.
INFO laravel/breeze ..... DONE
INFO laravel/pail ..... DONE
INFO laravel/sail ..... DONE
INFO laravel/sanctum ..... DONE
INFO laravel/tinker ..... DONE
INFO nesbot/carbon ..... DONE
INFO nunomaduro/collision ..... DONE
INFO nunomaduro/termwind ..... DONE
INFO pestphp/pest-plugin-laravel ..... DONE
```

Y esto nos genera estos archivos para la autenticación



para ver que todo se ha generado de manera correcta vamos a comprobar si los endpoints están funcionando para eso vamos a utilizar este comando.

php artisan route:list

```
C:\laragon\www\Foodsynk(main -> origin)
λ php artisan route:list

GET|HEAD / ...
GET|HEAD api/user ...
POST email/verification-notification ... verification.send > Auth\EmailVerificationNotificationController@store
POST forgot-password ...
POST login ...
POST logout ...
POST register ...
POST reset-password ...
GET|HEAD sanctum/csrf-cookie ...
GET|HEAD storage/{path} ...
GET|HEAD up ...
GET|HEAD verify-email/{id}/{hash} ...

Showing [12] routes
λ php artisan breeze install api
```

Estos son todos los endpoint que nos ha generado y que podemos utilizar.

Después de horas de prueba con la autenticación de breeze y el csrf token desistí y encontré otra solución porque no era capaz de securizar las rutas y elegí cambiar al bearer token que nos va a ser más sencillo con eso consumir la api desde el front.

Para empezar tenemos que crear un controlador de autenticación nuevo, pero que es un controlador? Es una clase que se encarga de recibir las peticiones HTTP, procesarlas y devolver una respuesta. Osea recibe los datos, los valida, ejecuta la lógica y devuelve un json (en modo api), para crearlo usamos el comando:

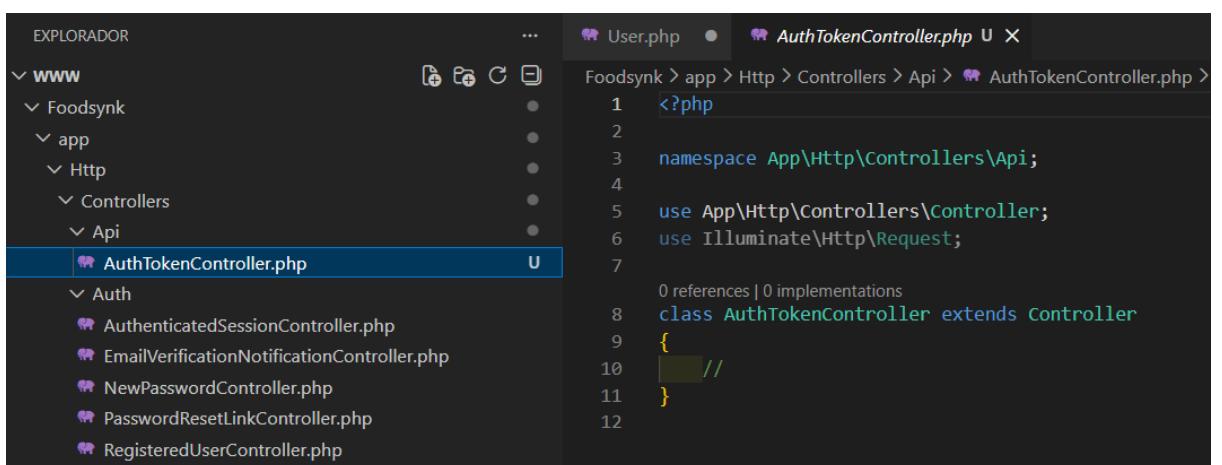
php artisan make:controller Api/AuthTokenController

Lo que hace ese comando es crearte un controlador llamado AAuth token controller dentro de la carpeta api.

```
C:\laragon\www\Foodsynk(main -> origin)
λ php artisan make:controller Api/AuthTokenController
[INFO] Controller [C:\laragon\www\Foodsynk\app\Http\Controllers\Api\AuthTokenController.php] created successfully.

para empezar tenemos que crear el controlador de la autenticación, pero que es un controlador? Es una clase que se encarga de
C:\laragon\www\Foodsynk(main -> origin) los, los valida, ejecuta la lógica y devuelve un json (en modo api), para crearlo usamos
λ |
```

Ahora está vacío.



```
EXPLORADOR ... User.php ● AuthTokenController.php U X
▼ www
  ▼ Foodsynk
    ▼ app
      ▼ Http
        ▼ Controllers
          ▼ Api
            AuthTokenController.php
            └── Auth
              └── AuthenticatedSessionController.php
              └── EmailVerificationNotificationController.php
              └── NewPasswordController.php
              └── PasswordResetLinkController.php
              └── RegisteredUserController.php
FoodSync > app > Http > Controllers > Api > AuthTokenController.php >
1  <?php
2
3  namespace App\Http\Controllers\Api;
4
5  use App\Http\Controllers\Controller;
6  use Illuminate\Http\Request;
7
8  0 references | 0 implementations
9  class AuthTokenController extends Controller
10 {
11
12 }
```

Esto lo cambiamos así:

```
<?php

namespace App\Http\Controllers\Api;

use App\Http\Controllers\Controller;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Hash;
use Illuminate\Validation\ValidationException;
use App\Models\User;

class AuthTokenController extends Controller
{
    public function login(Request $request)
    {
```

```

// 1 Validar los datos que llegan
$request->validate([
    'email' => ['required', 'email'],
    'password' => ['required'],
    'device_name' => ['nullable', 'string'],
]);

// 2 Buscar el usuario por email
$user = User::where('email', $request->email)->first();

// 3 Comprobar que el usuario existe y la contraseña es correcta
if (! $user || ! Hash::check($request->password,
$user->password)) {
    throw ValidationException::withMessages([
        'email' => ['Credenciales incorrectas.'],
    ]);
}

// 4 Nombre del dispositivo (Postman, navegador, etc.)
$deviceName = $request->device_name ?? 'api-client';

// 5 Crear token y devolverlo
return response()->json([
    'token' =>
$user->createToken($deviceName)->plainTextToken,
    'user' => $user,
]);
}
}

```

*explicar para qué es esto y que esto suele ser así en casi todos los proyectos, así que nos podría ser útil para próximos proyectos futuros.

Tenemos la funcionalidad de la autenticación pero no tenemos un endpoint para probarlo, por lo que tenemos que crearlo, en el archivo Api.php dentro de la carpeta routes.

```
<?php
```

```

use Illuminate\Support\Facades\Route;
use Illuminate\Http\Request;
use App\Http\Controllers\Api\AuthTokenController;

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

Route::middleware('auth:sanctum')->group(function () {
    Route::get('/me', function (Request $request) {
        return $request->user();
    });
});

//      Route::post('/logout', [AuthTokenController::class, 'logout']);  lo crearemos más adelante.
});;

```

ahora para confirmar las dos rutas, utilizamos este comando:

php artisan route:list --path=api

que es el mismo que usamos antes pero que contiene un filtro por el que se nos va a mostrar solo aquellas que estén dentro del api.

```

C:\laragon\www\Foodsy\main->origin> php artisan route:list --path=api
λ php artisan route:list --path=api
POST    api/login ..... Api\AuthTokenController@login
GET|HEAD api/me .....
```

Showing [2] routes

```

Route::post('/logout', [AuthTokenController::class, 'logout']); // lo crearemos en el siguiente paso
C:\laragon\www\Foodsy\main->origin>
```

Ahora vamos a crear un usuario por tinker, tinker es como una consola donde podemos ejecutar código de las clases/modelos de nuestro proyecto:

php artisan tinker

```
C:\laragon\www\Foodsynk(main -> origin)
λ php artisan tinker --connection=mysql
Psy Shell v0.12.18 (PHP 8.3.28 - cli) by Justin Hileman
New PHP manual is available (latest: 3.0.1). Update with `doc --update-manual`
> \App\Models\User::create([
    'name' => 'Nuria',
    'email' => 'nuria@test.com',
    'password' => bcrypt('password'),
])
> Illuminate\Database\Eloquent\Model::find(1)
Illuminate\Database\QueryException SQLSTATE[HY000] [1045] Access denied for user 'root'@'localhost' (using password: NO) (Connection: mysql, Host: 127.0.0.1, Port: 3306, Database: laravel, SQL: insert into `users` (`name`, `email`, `password`, `updated_at`, `created_at`) values ('Nuria', 'nuria@test.com', '$2y$12$HrRKAj45tdCFSQ90MMpXI..V8aVaX1pkp.1hEStoYQ8M0ePEVpjMO', 2026-01-12 19:12:25, 2026-01-12 19:12:25)).
```

Aquí tuve un problema:

```
Route::post('/logout')|Auth\TokenController::class, 'logout']); // lo crearemos en el paso 3
C:\laragon\www\Foodsynk(main -> origin)
λ php artisan config:clear
INFO Configuration cache cleared successfully.

hora podemos hacer la prueba de crear un usuario con tinker, tinker es como una consola de desarrollo que nos permite interactuar con la base de datos y el modelo de datos de Laravel.

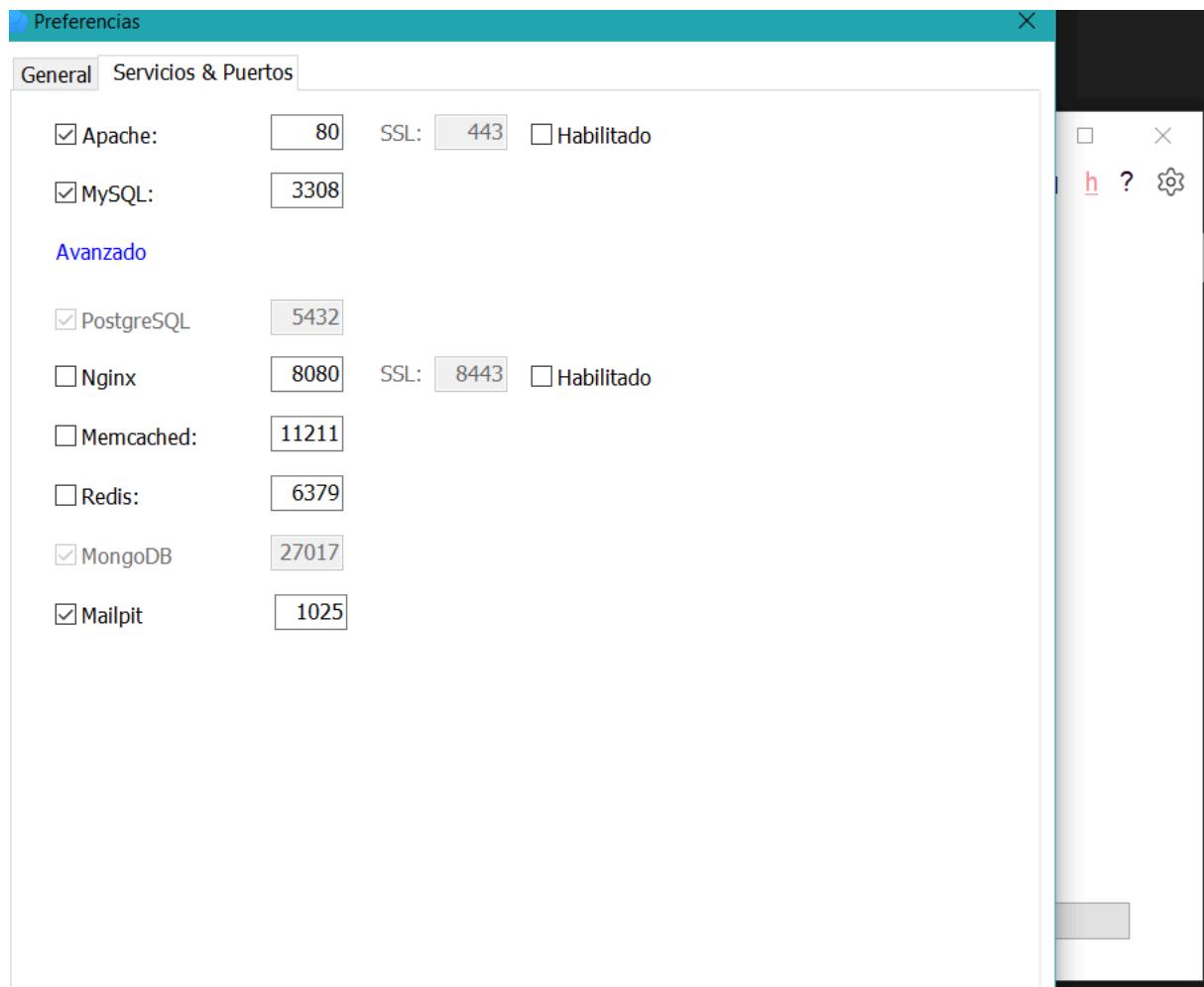
C:\laragon\www\Foodsynk(main -> origin)
λ php artisan cache:clear
INFO Application cache cleared successfully.

C:\laragon\www\Foodsynk(main -> origin)
λ php artisan tinker
Psy Shell v0.12.18 (PHP 8.3.28 - cli) by Justin Hileman
New PHP manual is available (latest: 3.0.1). Update with `doc --update-manual`
> config('database.connections.mysql.host');
= "127.0.0.1"

> config('database.connections.mysql.username');
= "root"

> config('database.connections.mysql.password');
= ""

>
```



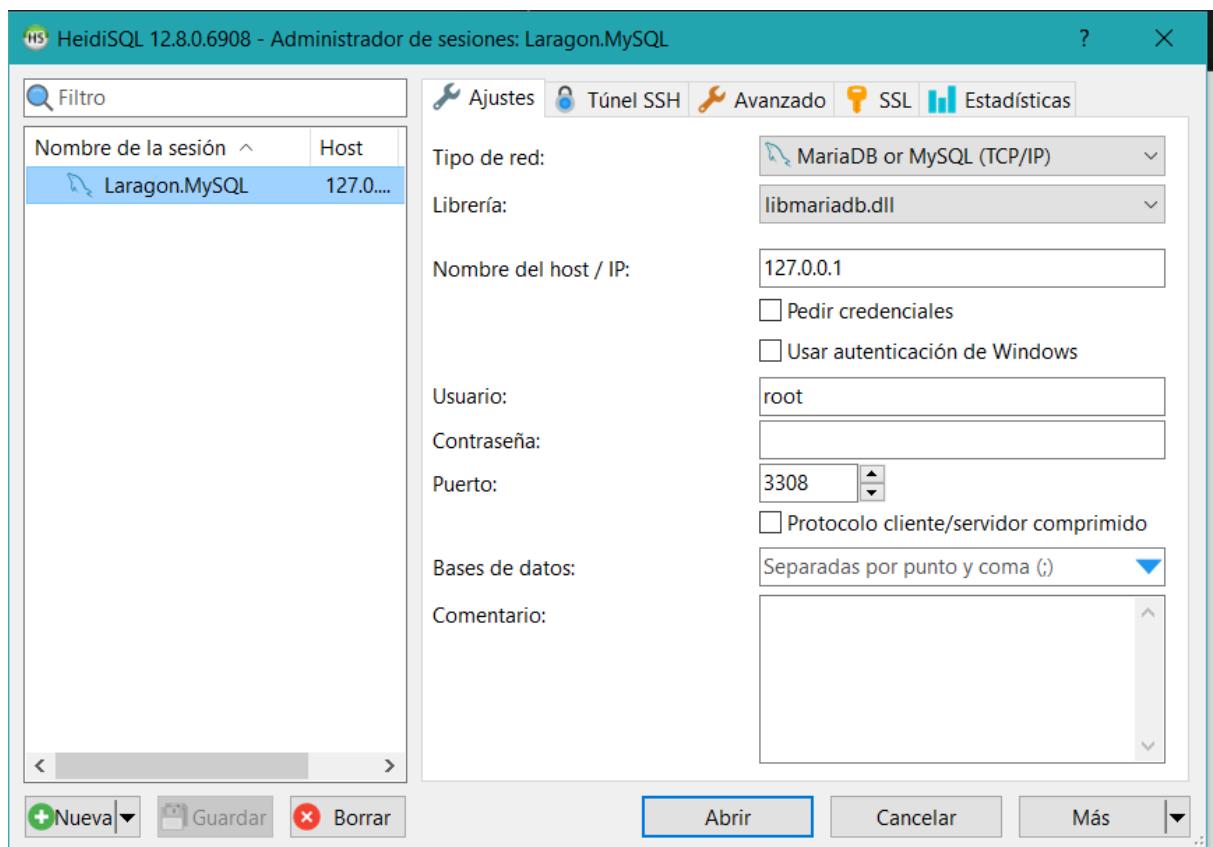
The screenshot shows the 'my.ini' configuration file in Notepad++. The file contains the following settings:

```
[client]
#password=your_password
port=3308
socket=/tmp/mysql.sock

[mysqld]
datadir="C:/laragon/data/mysql-8.4"
port= 3308
socket=/tmp/mysql.sock
key_buffer_size=256M
max_allowed_packet=512M
table_open_cache=256
sort_buffer_size=1M
read_buffer_size=1M
read_rnd_buffer_size=4M
myisam_sort_buffer_size=64M
thread_cache_size=8
secure-file-priv=""
explicit_defaults_for_timestamp=1

[mysqldump]
=quick
max_allowed_packet=512M
```

The Notepad++ status bar at the bottom indicates: MS ini file | length: 445 lines: 24 | Ln: 3 Col: 10 Pos: 45 | Windows (CR LF) | UTF-8 | INS | :|



Conseguimos añadir un user a nuestra base de datos para probar el login.

```
C:\laragon\www\Foodsync(main -> origin)
λ php artisan tinker DB_CONNECTION=mysql
Psy Shell v0.12.18 (PHP 8.3.28 - cli) by Justin Hileman
New PHP manual is available (latest: 3.0.1). Update with `doc --update-manual`
> \App\Models\User::create([
    'name' => 'Nuria',
    'email' => 'nuria@test.com',
    'password' => bcrypt('password'),
])
= \App\Models\User {#5875
    name: "Nuria",
    email: "nuria@test.com",
    #password: "$2y$12$t7SIRVnv3GC1G7cW8bpPD.NeEFD13ldUZ7ZJuZG0BAmKH580AV/Fe",
    updated_at: "2026-01-12 20:00:19",
    created_at: "2026-01-12 20:00:19",
    id: 1,
}
37 SESSION_DOMAIN=null
38
39 BROADCAST_CONNECTION=log
```

Ahora desde postman, creé una nueva colección y le he añadido una nueva petición de post que es:

The screenshot shows the Postman interface with a POST request to `localhost:8000/api/login`. The Body tab is selected, showing a raw JSON payload:

```
1 [{}  
2 ...."email":"nuria@test.com",  
3 ...."password":"password"  
4 ]
```

The response status is 200 OK, with a response body containing a token and user information:

```
1 [{"token": "4|9ySFqrPJZ3ajk2G1EazlNZ1WVD0xwniFxPd5oYope6700433", "user": {"id": 1, "name": "Nuria", "email": "nuria@test.com", "email_verified_at": null, "created_at": "2026-01-12T20:00:19.000000Z", "updated_at": "2026-01-12T20:00:19.000000Z"}]}
```

Below the response, a note explains the difference between GET and POST requests regarding entity representation.

En el body hay que añadir en las opciones que sea `raw` y que esté en formato JSON. Por otro lado nosotros tenemos que añadir las credenciales de nuestro usuario también en formato json.

También es importante es que headers que

The Headers section shows two entries:

- `Content-Type` set to `application/json`
- `Accept` set to `application/json`

y debería salir así:

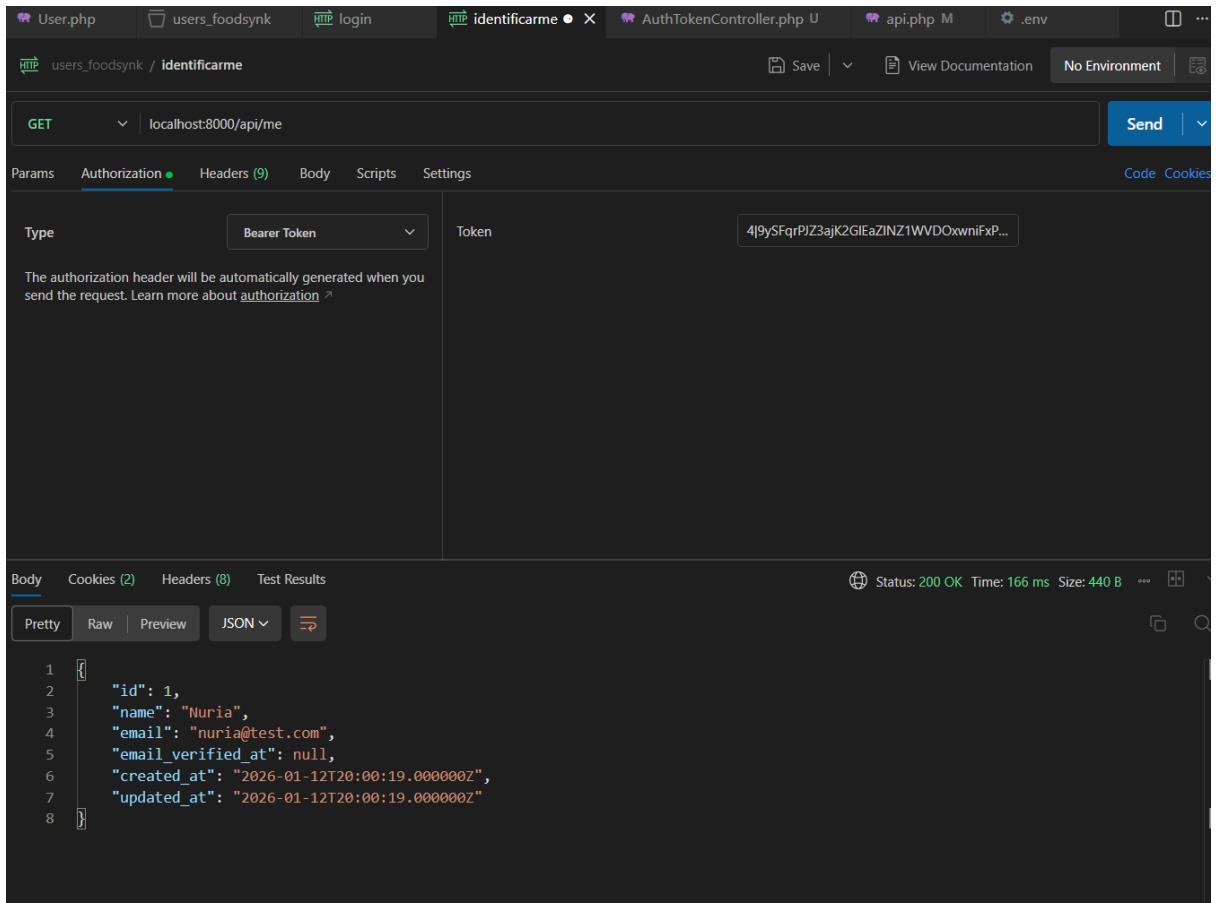
The response body is identical to the previous one:

```
1 [{"token": "4|9ySFqrPJZ3ajk2G1EazlNZ1WVD0xwniFxPd5oYope6700433", "user": {"id": 1, "name": "Nuria", "email": "nuria@test.com", "email_verified_at": null, "created_at": "2026-01-12T20:00:19.000000Z", "updated_at": "2026-01-12T20:00:19.000000Z"}]}
```

Lo importante de esto es que siempre vamos a usar este token para autenticarnos desde el front colocándolo en los headers de las llamadas.

Para explicar esto y que nos quede claro. Si yo ahora hago una nueva petición get, en vez de login a me, que sería mi usuario y lo mando, la petición se quedaría colgada.

Pero si yo cojo, me voy a autorización, busco que sea del tipo bearer porque es una forma sencilla y segura y le añadimos el token que nos dio en la anterior petición, nos sale nuestro usuario.



The screenshot shows the Postman interface with the following details:

- Header: Authorization: Bearer 4J9ySFqrPIZ3ajK2GIEaZINZ1WVD0xwniFxP...
- Body: JSON response (Pretty) showing user data:

```
1  {
2    "id": 1,
3    "name": "Nuria",
4    "email": "nuria@test.com",
5    "email_verified_at": null,
6    "created_at": "2026-01-12T20:00:19.000000Z",
7    "updated_at": "2026-01-12T20:00:19.000000Z"
8 }
```
- Status: 200 OK

Por si acaso también lo probamos con curl.

```
C:\laragon\www\Foodsynk(main -> origin)
λ curl -X POST http://localhost:8000/api/login ^
¿Más? -H "Accept: application/json" ^
¿Más? -H "Content-Type: application/json" ^
¿Más? -d "{\"email\":\"nuria@test.com\",\"password\":\"password\"}"
{"token": "3|N4Y9ZPwt3ziDdh1HvxM36Nb0thXHjgEjSCfaNjtd8ddbd96", "user": {"id": 1, "name": "Nuria", "email": "nuria@test.com", "email_verified_at": null, "created_at": "2026-01-12T20:00:19.000000Z", "updated_at": "2026-01-12T20:00:19.000000Z"}}
C:\laragon\www\Foodsynk(main -> origin)
```

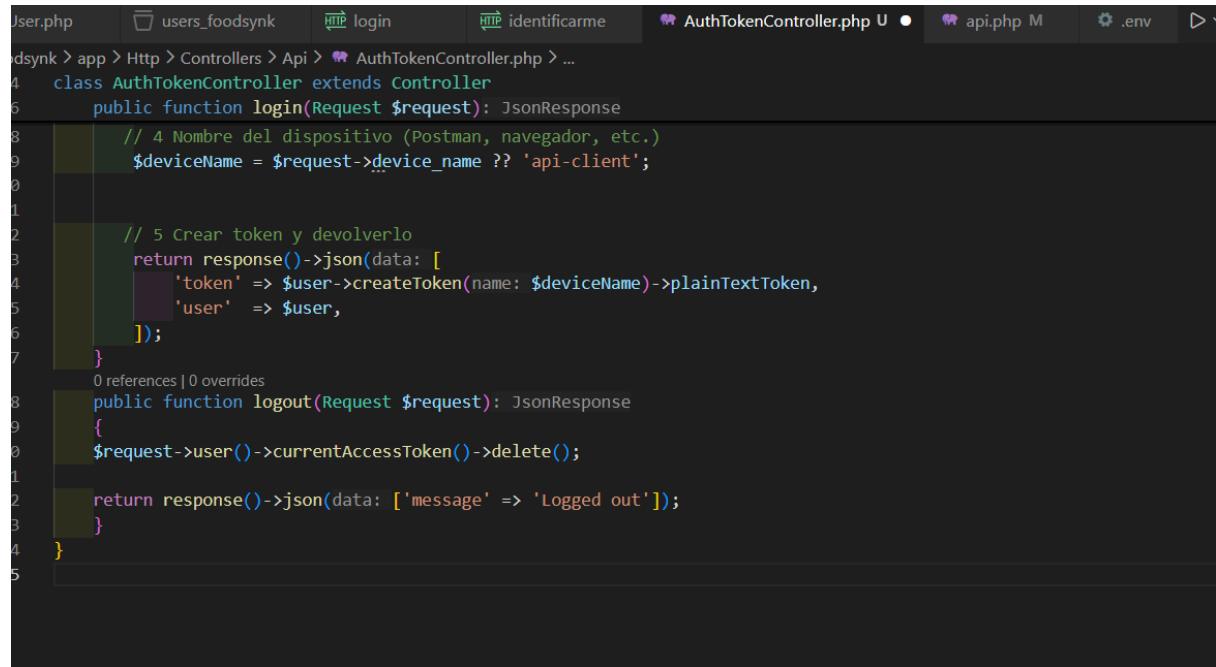
Por si acaso lo subí a git dónde también tuve un problemilla de cuentas.

Ahora vamos a crear el logout.

Para ello, nos vamos al controlador que hemos creado anteriormente, el authTokenController.php y añadimos la función:

```
public function logout(Request $request)
{
    $request->user()->currentAccessToken()->delete();

    return response()->json(['message' => 'Logged out']);
}
```



```
User.php      users_foodsynk      login      identificarme      AuthTokenController.php U •      api.php M      .env
User.php      users_foodsynk      login      identificarme      AuthTokenController.php U •      api.php M      .env
1 4dsynk > app > Http > Controllers > Api > AuthTokenController.php > ...
2 4   class AuthTokenController extends Controller
3 5       public function login(Request $request): JsonResponse
4 6           // 4 Nombre del dispositivo (Postman, navegador, etc.)
5 7           $deviceName = $request->device_name ?? 'api-client';
6
7
8           // 5 Crear token y devolverlo
9           return response()->json(data: [
10               'token' => $user->createToken(name: $deviceName)->plainTextToken,
11               'user'  => $user,
12           ]);
13
14       }
15
16       0 references | 0 overrides
17       public function logout(Request $request): JsonResponse
18       {
19           $request->user()->currentAccessToken()->delete();
20
21           return response()->json(data: ['message' => 'Logged out']);
22       }
23
24   }
25
```

La mecánica siempre es la misma, creamos la función que queremos en el controlador correspondiente y creamos una ruta en el routes/api.php la llamada a esa función :

```
Route::middleware('auth:sanctum')->group(function () {
    Route::get('/me', fn (Request $request) => $request->user());
    Route::post('/logout', [AuthTokenController::class, 'logout']);
});
```

The screenshot shows a code editor with several tabs at the top: User.php, users_foodsynk, login, identificarme, AuthTokenController.php, api.php, .env, and others. The api.php tab is active, displaying the following PHP code:

```
1 <?php
2
3 /**
4  * use Illuminate\Support\Facades\Route;
5  * use Illuminate\Http\Request;
6  * use App\Http\Controllers\Api\AuthTokenController;
7
8
9 Route::post('login', [AuthTokenController::class, 'login']);
10
11
12 Route::middleware('auth:sanctum')->group(function () {
13     Route::get('/me', [Request $request]: mixed => $request->user());
14     Route::post('logout', [AuthTokenController::class, 'logout']);
15 });

```

Ahora volvemos a postman y :

The screenshot shows the Postman interface. The URL is set to `users_foodsynk / logout`. The method is `POST` and the target is `localhost:8000/api/logout`. The `Authorization` tab is selected, showing a `Bearer Token` type with the value `4j9ySFqrPJZ3ajK2GiEaZINZ1WVD0xwniFxP...`. The response status is `200 OK`, time `249 ms`, and size `305 B`. The response body is a JSON object with the message `"Logged out"`.

Es importante que si da error tener `Accept application/json`.

13/01/26- 18:27-20:29 20:39-

Vamos a instalar Orion, que es un paquete de laravel que nos crea endpoints automáticos, basados en controladores con un montón de ayudas como paginación, filtros, relaciones...

Para instalar Orion usamos:

composer require tailflow/laravel-orion

```
V C:\laragon\www\Foodsynk(main -> origin)
v λ composer require tailflow/laravel-orion
c   ./composer.json has been updated
c     Running composer update tailflow/laravel-orion
c       Loading composer repositories with package information
c         Updating dependencies
N           Lock file operations: 3 installs, 0 updates, 0 removals
P             - Locking doctrine/dbal (4.4.1)
P             - Locking psr/cache (3.0.0)
P             - Locking tailflow/laravel-orion (2.23.0)
I   Writing lock file
N     Installing dependencies from lock file (including require-dev)
S       Package operations: 3 installs, 0 updates, 0 removals
S         - Downloading psr/cache (3.0.0)
S         - Downloading doctrine/dbal (4.4.1)
N       - Downloading tailflow/laravel-orion (2.23.0)
+         - Installing psr/cache (3.0.0): Extracting archive
```

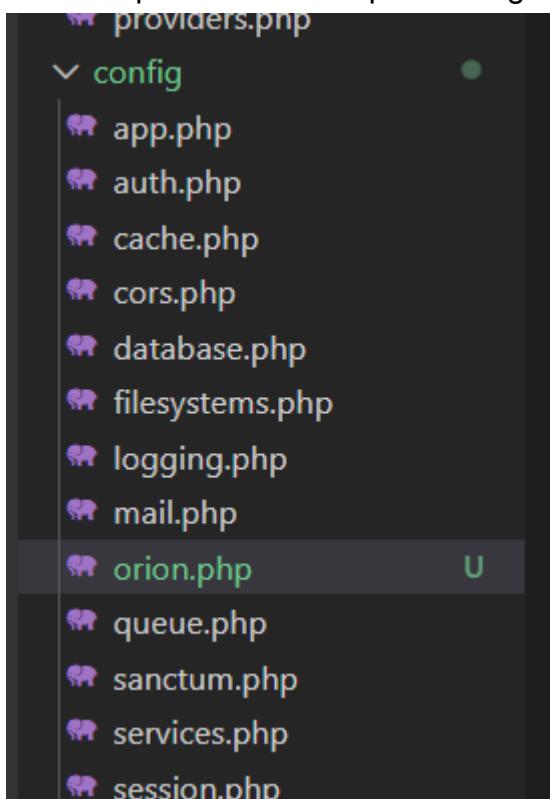
Ahora hay que publicar el archivo de configuración de Orion para poder modificarlo.Para eso usamos:

php artisan vendor:publish --tag=orion-config

(con artisan siempre va php o sail delante)

```
C:\laragon\www\Foodsynk(main -> origin)
λ php artisan vendor:publish --tag=orion-config
INFO Publishing [orion-config] assets.
stoy contento de mas o menos lo que te he montado
Copying file [C:\laragon\www\Foodsynk\vendor\tailflow\laravel-orion\config\orion.php] to [C:\laragon\www\Foodsynk\conf
ig\orion.php] DONE
```

Esto nos publica en la carpeta config un archivo llamado orion.php



Dentro de este archivo tenemos que cambiar el *api* que aparece al lado de guard, en auth por sanctum.

```
config > orion.php
1  <?php
2
3  return [
4      'namespaces' => [
5          'models' => 'App\\Models\\',
6          'controllers' => 'App\\Http\\Controllers\\',
7      ],
8      'auth' => [
9          'guard' => 'sanctum',
10     ],
11 ]
```

Este cambio sirve para decirle a orion que vamos a usar sanctum para la autenticación de las rutas.

Vamos a crear la entidad recetas dentro de la bd.

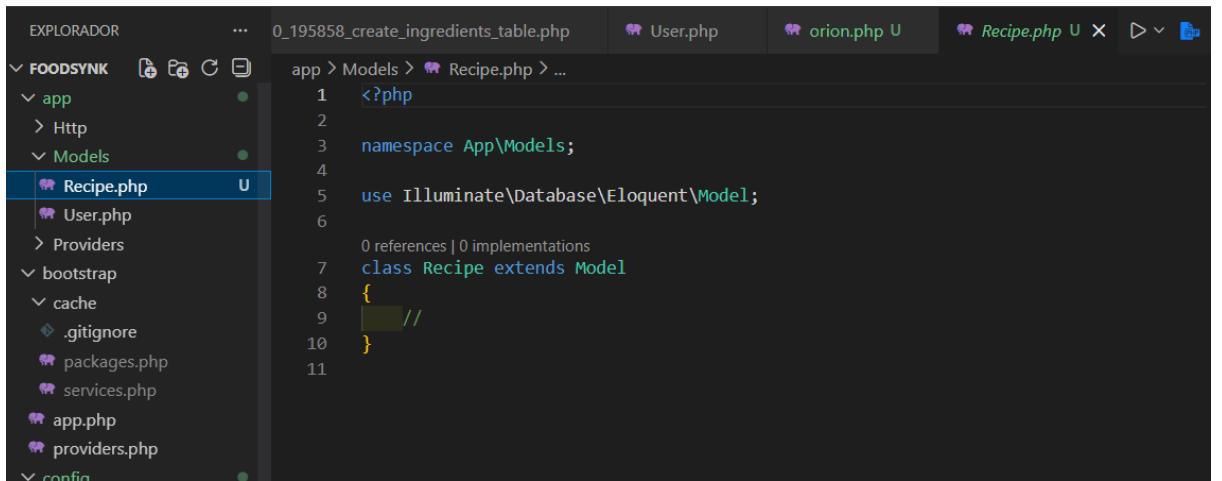
php artisan make:model Recipe -m

Ese comando te genera el modelo recetas y el -m del final te genera una migración para que impacte directamente en la bd.

```
C:\laragon\www\Foodsynk(main -> origin)
λ php artisan make:model Recipe -m
[INFO] Model [C:\laragon\www\Foodsynk\app\Models\Recipe.php] created successfully.
[INFO] Migration [C:\laragon\www\Foodsynk\database\migrations\2026_01_13_174610_create_recipes_table.php] created successfully.

Unos es un paquete de laravel que nos crea endpoints automaticos, basados en controladores con un montón de ayudas como pa
```

Te crea las tabla Recetas, en plural.



```
EXPLORADOR      ...  0_195858_create_ingredients_table.php | User.php | orion.php U | Recipe.php U X D v E
└─ FOODSYNK  ⌂ ⌃ C ⌚  app > Models > Recipe.php > ...
    └─ app
        > Http
        < Models
            └─ Recipe.php U
                User.php
                > Providers
            < bootstrap
                < cache
                    .gitignore
                packages.php
                services.php
            app.php
            providers.php
        config
```

```

1 <?php
2
3 namespace App\Models;
4
5 use Illuminate\Database\Eloquent\Model;
6
7 class Recipe extends Model
8 {
9     //
10 }
```

Esto lo modificamos para que se adapte a nuestra bd.

Ahora tenemos que llenar la migración con:

A screenshot of a code editor showing a PHP migration file named `2026_01_13_174610_create_recipes_table.php`. The file is part of a Laravel project, as indicated by the file paths in the top bar: `database > migrations > 2026_01_13_174610_create_recipes_table.php`. The code defines a new database table named `recipes` with the following columns:

```
1 <?php
2
3 use Illuminate\Database\Migrations\Migration;
4 use Illuminate\Database\Schema\Blueprint;
5 use Illuminate\Support\Facades\Schema;
6
7 return new class extends Migration {
8     public function up(): void
9     {
10         Schema::create('recipes', function (Blueprint $table): void
11         {
12             $table->id();
13             $table->string('titulo');
14             $table->string('foto')->nullable();
15             $table->text('pasos');
16             $table->timestamps();
17         });
18     }
19
20     public function down(): void
21     {
22         Schema::dropIfExists('recipes');
23     }
24 };
```

ahora tenemos que hacer un migrate para que impacte los cambios en la bd

```
C:\laragon\www\Foodsy wholek (main -> origin)
λ php artisan migrate

[INFO] Running migrations.

2026_01_13_174610_create_recipes_table ..... 58.51ms DONE

C:\laragon\www\Foodsy wholek (main -> origin)
λ
```

ahora tenemos que hacer el controlador de la receta:

```
php artisan make:controller Api/RecipeController --api
```

```
C:\laragon\www\Foodsy wholek(main -> origin)
\ php artisan make:controller Api/RecipeController --api

[INFO] Controller [C:\laragon\www\Foodsy wholek\app\Http\Controllers\Api\RecipeController.php] created successfully.

C:\laragon\www\Foodsy wholek(main -> origin)
```

De manera automática se genera este archivo con todos los métodos para que los implementemos pero no nos hace falta porque Orion lo hace por nosotros:

The screenshot shows a code editor with a sidebar navigation on the left and the main code area on the right.

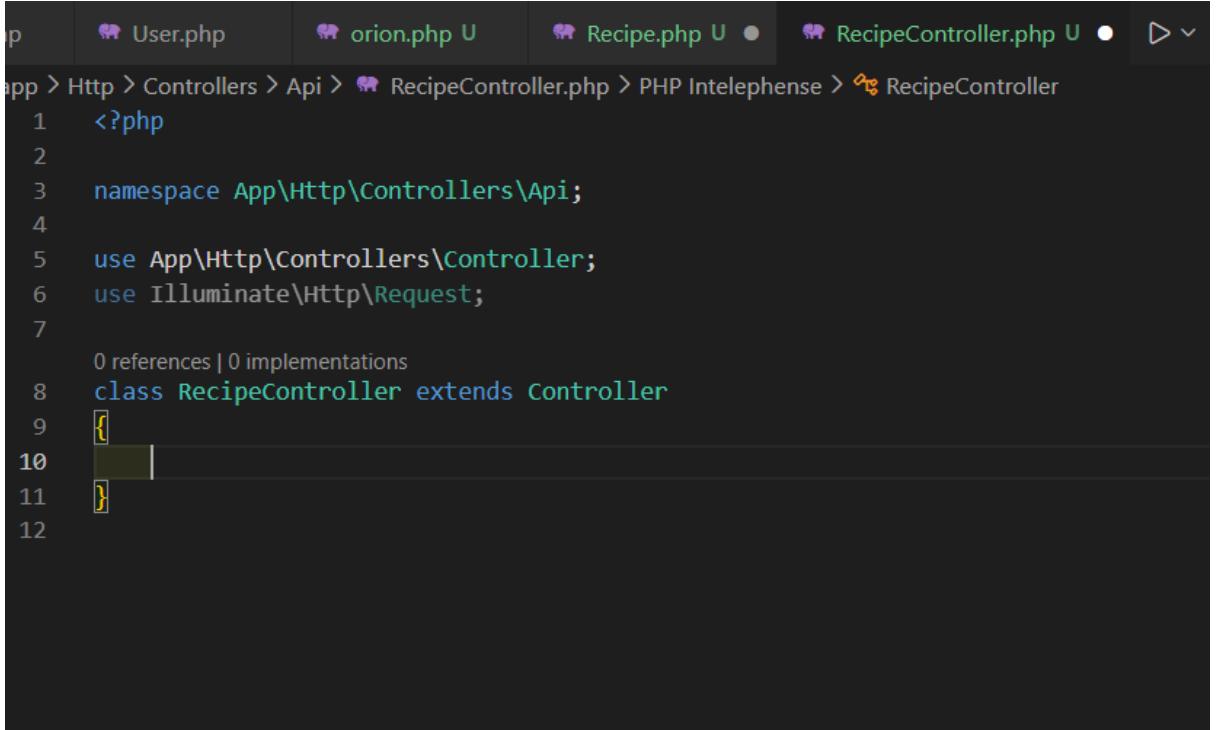
Left Sidebar (File Explorer):

- EXPLORADOR
- ...
- e.php
- User.php
- orion.php U
- Recipe.php U
- RecipeController.php U X
- ...

FOODSYNK

- app
- Http
- Controllers
 - Api
 - AuthTokenController.php
 - RecipeController.... U
- Auth
- Controller.php
- Middleware
- Requests
- Models
- Providers
- bootstrap
- cache
 - .gitignore
- packages.php
- services.php
- app.php
- providers.php
- config
 - app.php
 - auth.php
 - cache.php
 - cors.php
 - database.php
 - filesystems.php
 - logging.php
 - mail.php
 - orion.php U
 - queue.php
 - sanctum.php
 - services.php

Por lo tanto el relleno no sirve de nada.

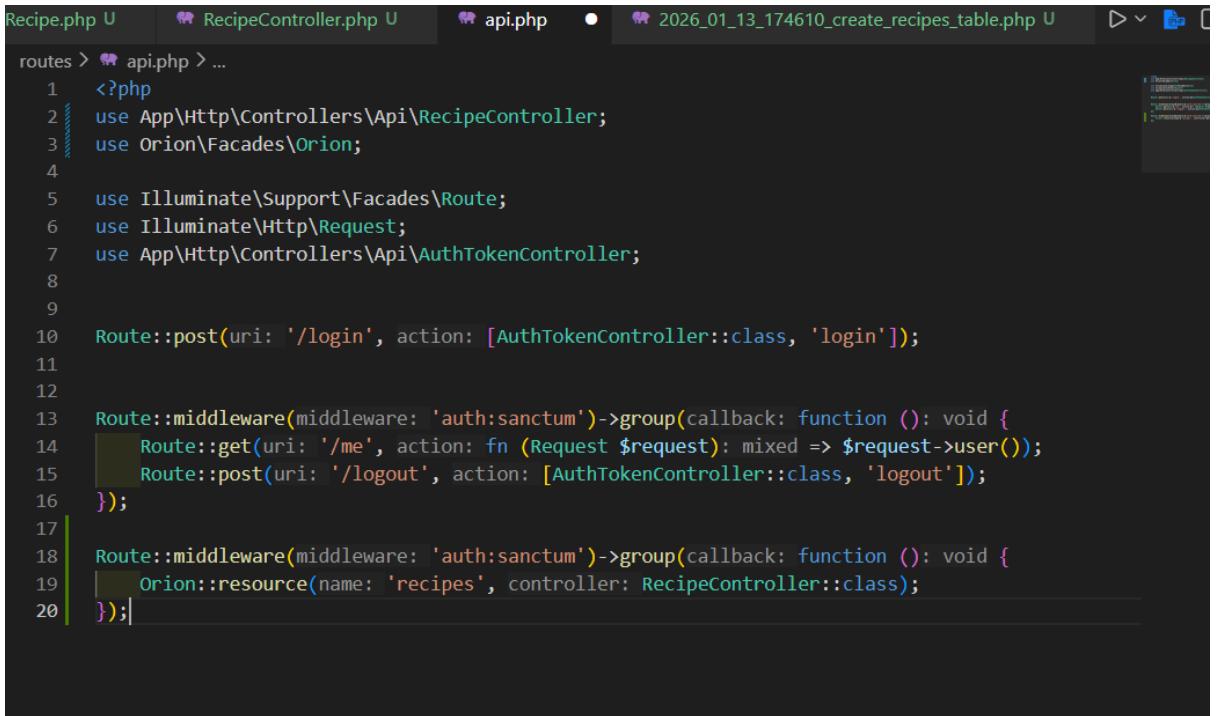


The screenshot shows a code editor with several tabs at the top: User.php, orion.php U, Recipe.php U, and RecipeController.php U. The RecipeController.php tab is active. The code in the editor is:

```
1 <?php
2
3 namespace App\Http\Controllers\Api;
4
5 use App\Http\Controllers\Controller;
6 use Illuminate\Http\Request;
7
8 class RecipeController extends Controller
9 {
10
11 }
```

Y le añadimos esta línea que llama al modelo recipe.

Con esto Orión prácticamente está funcionando y lo que nos queda por hacer es añadir las rutas de los recipe al api.php del routes:



The screenshot shows a code editor with tabs: routes, RecipeController.php U, api.php, and 2026_01_13_174610_create_recipes_table.php U. The api.php tab is active. The code in the editor is:

```
routes > api.php > ...
1 <?php
2 use App\Http\Controllers\Api\RecipeController;
3 use Orion\Facades\Orion;
4
5 use Illuminate\Support\Facades\Route;
6 use Illuminate\Http\Request;
7 use App\Http\Controllers\Api\AuthTokenController;
8
9
10 Route::post(uri: '/login', action: [AuthTokenController::class, 'login']);
11
12
13 Route::middleware(middleware: 'auth:sanctum')->group(callback: function () {
14     Route::get(uri: '/me', action: fn (Request $request): mixed => $request->user());
15     Route::post(uri: '/logout', action: [AuthTokenController::class, 'logout']);
16 });
17
18 Route::middleware(middleware: 'auth:sanctum')->group(callback: function () {
19     Orion::resource(name: 'recipes', controller: RecipeController::class);
20 });
```

Vamos a ver los endpoints que tenemos con: **php artisan route:list**

```
C:\laragon\www\Foodsynk(main -> origin) | auth:token [auth:token:controller:login, "login"] |
λ php artisan route:list

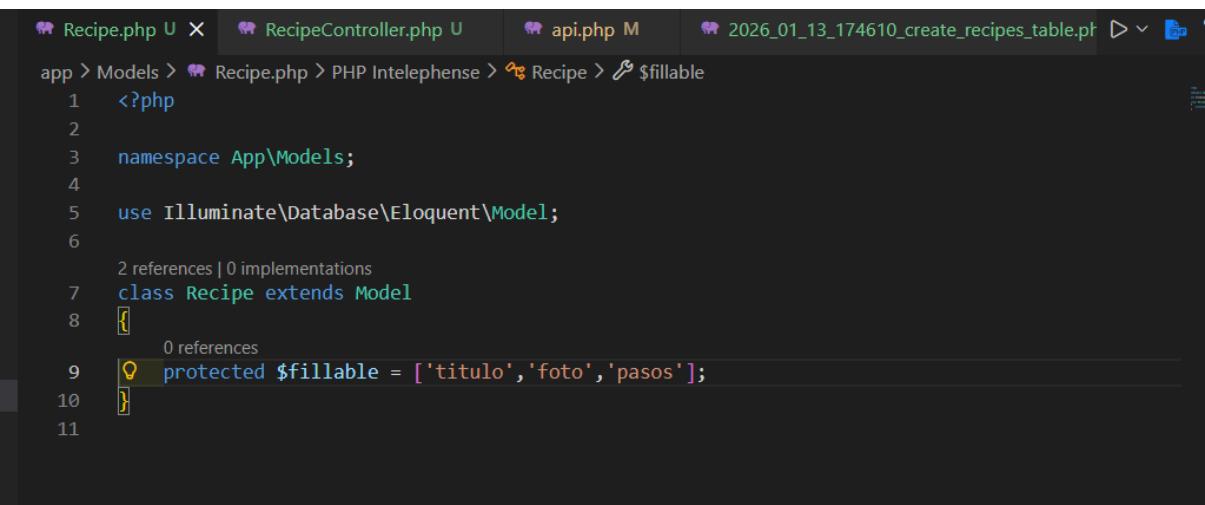
GET|HEAD / ..... .
POST api/login ..... Api\AuthTokenController@login
POST api/logout ..... Api\AuthTokenController@logout
GET|HEAD api/me ..... .
GET|HEAD api/recipes ..... recipes.index > Api\RecipeController@index
POST api/recipes ..... recipes.store > Api\RecipeController@store
POST api/recipes/batch ..... recipes.batchStore > Api\RecipeController@batchStore
PATCH api/recipes/batch ..... recipes.batchUpdate > Api\RecipeController@batchUpdate
DELETE api/recipes/batch ..... recipes.batchDestroy > Api\RecipeController@batchDestroy
POST api/recipes/search ..... recipes.search > Api\RecipeController@search
GET|HEAD api/recipes/{recipe} ..... recipes.show > Api\RecipeController@show
PUT|PATCH api/recipes/{recipe} ..... recipes.update > Api\RecipeController@update
DELETE api/recipes/{recipe} ..... recipes.destroy > Api\RecipeController@destroy
POST email/verification-notification verification.send > Auth\EmailVerificationNotificationController@st...
POST forgot-password ..... password.email > Auth\PasswordResetLinkController@store
POST login ..... login > Auth\AuthenticatedSessionController@store
POST logout ..... logout > Auth\AuthenticatedSessionController@destroy
POST register ..... register > Auth\RegisteredUserController@store
POST reset-password ..... password.store > Auth\NewPasswordController@store
GET|HEAD sanctum/csrf-cookie ..... sanctum.csrf-cookie > Laravel\Sanctum > CsrfCookieController@show
GET|HEAD storage/{path} ..... storage.local
GET|HEAD up ..... .
GET|HEAD verify-email/{id}/{hash} ..... verification.verify > Auth\VerifyEmailController

Showing [23] routes
```

Si nos fijamos, nos crea varios endpoints.

```
GET|HEAD 14 api/recipes ..... recipes.index > Api\RecipeController@index
POST 15 api/recipes ..... recipes.store > Api\RecipeController@store
POST 16 api/recipes/batch ..... recipes.batchStore > Api\RecipeController@batchStore
PATCH 17 api/recipes/batch ..... recipes.batchUpdate > Api\RecipeController@batchUpdate
DELETE 18 api/recipes/batch ..... recipes.batchDestroy > Api\RecipeController@batchDestroy
POST 19 api/recipes/search ..... recipes.search > Api\RecipeController@search
GET|HEAD 20 api/recipes/{recipe} ..... recipes.show > Api\RecipeController@show
PUT|PATCH api/recipes/{recipe} ..... recipes.update > Api\RecipeController@update
DELETE api/recipes/{recipe} ..... recipes.destroy > Api\RecipeController@destroy
```

Para poder crear una receta, tenemos que crear una entrada de datos desde el front o desde postman, por lo que tenemos que validar estos datos. Para eso, tenemos que añadir la línea: `$fillable`, en el modelo.

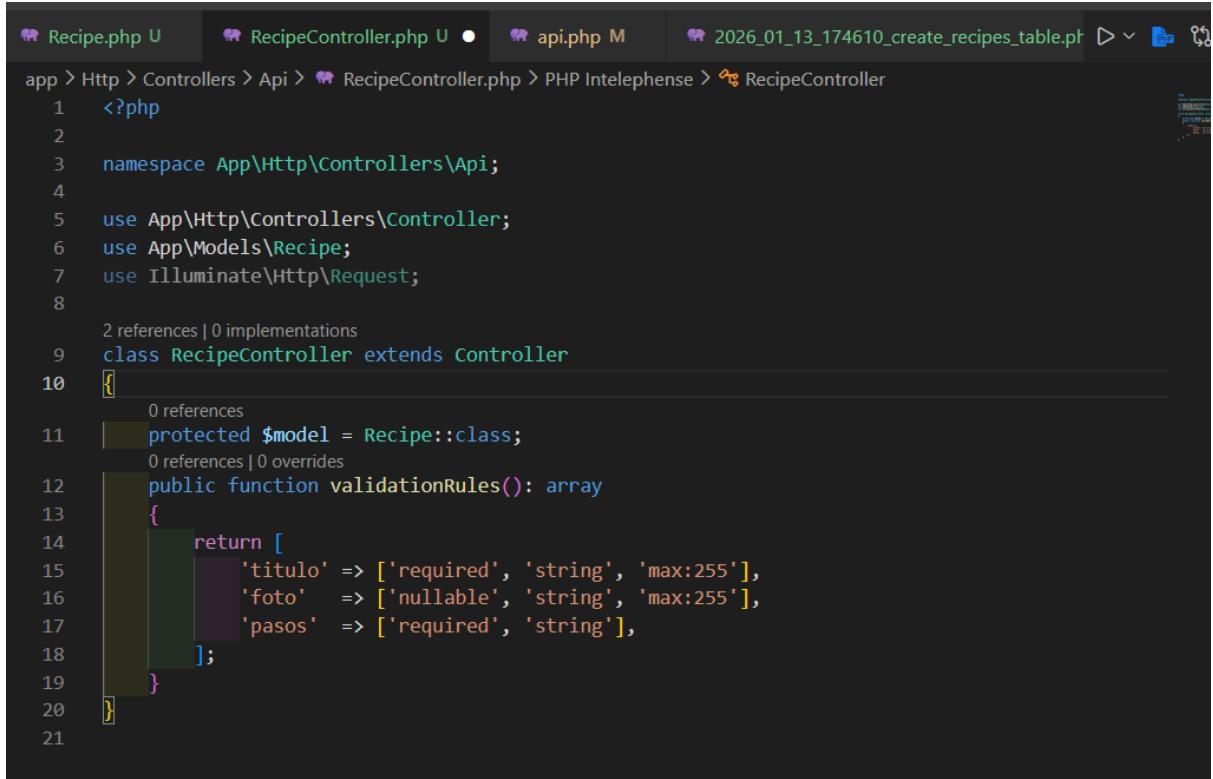


The screenshot shows a code editor with two tabs open: `Recipe.php` and `RecipeController.php`. The `Recipe.php` tab is active, displaying the following code:

```
<?php
namespace App\Models;
use Illuminate\Database\Eloquent\Model;
class Recipe extends Model
{
    protected $fillable = ['titulo', 'foto', 'pasos'];
}
```

The `$fillable` array contains three fields: 'titulo', 'foto', and 'pasos'. The `RecipeController.php` tab is also visible in the background.

Luego, en el controlador tenemos que añadir una función que valide los datos que el modelo acepte coger.

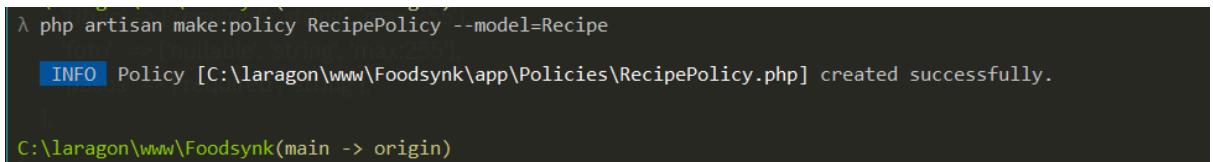


```
app > Http > Controllers > Api > RecipeController.php U ● api.php M 2026_01_13_174610_create_recipes_table.php D ▾ RecipeController.php
1 <?php
2
3 namespace App\Http\Controllers\Api;
4
5 use App\Http\Controllers\Controller;
6 use App\Models\Recipe;
7 use Illuminate\Http\Request;
8
9 class RecipeController extends Controller
10 {
11     protected $model = Recipe::class;
12
13     public function validationRules(): array
14     {
15         return [
16             'titulo' => ['required', 'string', 'max:255'],
17             'foto' => ['nullable', 'string', 'max:255'],
18             'pasos' => ['required', 'string'],
19         ];
20     }
21 }
```

Por ahora tampoco nos dejaría meter datos porque Orion necesita un archivo de políticas que define lo que puede o no hacer el usuario.

Para crear ese archivo de políticas para la receta tenemos que hacerlo:

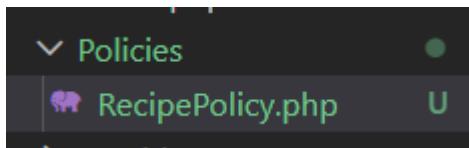
```
php artisan make:policy RecipePolicy --model=Recipe
```



```
λ php artisan make:policy RecipePolicy --model=Recipe
INFO Policy [C:\laragon\www\Foodsynk\app\Policies\RecipePolicy.php] created successfully.

C:\laragon\www\Foodsynk(main -> origin)
```

con esto te lo crea:



Con la configuración inicial el usuario puede hacer de todo pero como por ahora no tenemos usuario, no pasaría nada:

```
<?php

namespace App\Policies;

use App\Models\Recipe;
use App\Models\User;

class RecipePolicy
{
    public function viewAny(User $user): bool
    {
        return true;
    }

    public function view(User $user, Recipe $recipe): bool
    {
        return true;
    }

    public function create(User $user): bool
    {
        return true;
    }

    public function update(User $user, Recipe $recipe): bool
    {
        return true;
    }

    public function delete(User $user, Recipe $recipe): bool
    {
        return true;
    }
}
```

```
}
```

Como ya sabemos hay relaciones diferentes en las bases de datos pero Orion nos facilita mucho ese trabajo. Lo primero que hay que hacer es una migración para modificar la tabla de Recipes y ponerle el id del user para que cada usuario pueda identificar sus recetas, para ello usamos el comando:

```
php artisan make:migration add_user_id_to_recipes_table  
--table=recipes
```

```
C:\laragon\www\Foodsynk(main -> origin)  
λ php artisan make:migration add_user_id_to_recipes_table --table=recipes  
  
[INFO] Migration [C:\laragon\www\Foodsynk\database\migrations\2026_01_13_182521_add_user_id_to_recipes_table.php] created successfully.  
  
Como ya sabemos hay relaciones diferentes en las bases de datos pero Orion nos facilita mucho ese trabajo. Lo primero que hay que hacer es una migración para modificar la tabla de Recipes y ponerle el id del user para que cada usuario pueda identificar sus recetas, para ello usamos el comando:  
C:\laragon\www\Foodsynk(main -> origin)  
λ |
```

Dentro de la nueva migración tenemos que añadir esto:

```
<?php  
  
  
use Illuminate\Database\Migrations\Migration;  
use Illuminate\Database\Schema\Blueprint;  
use Illuminate\Support\Facades\Schema;  
  
  
return new class extends Migration  
{  
    /*Run the migrations.*/  
    public function up(): void  
    {  
        Schema::table('recipes', function (Blueprint $table) {  
            $table->foreignId('user_id')  
                ->constrained()  
                ->cascadeOnDelete();  
        });  
    }  
  
    /*Reverse the migrations.*/  
    public function down(): void  
    {  
        Schema::table('recipes', function (Blueprint $table) {  
            $table->dropForeign(['user_id']);  
            $table->dropColumn('user_id');  
        });  
    }  
}
```

```
    } ) ;
}

} ;
```

ahora hacemos un migrate:

```
C:\laragon\www\Foodsy whole file
n(\main -> origin)
λ php artisan migrate

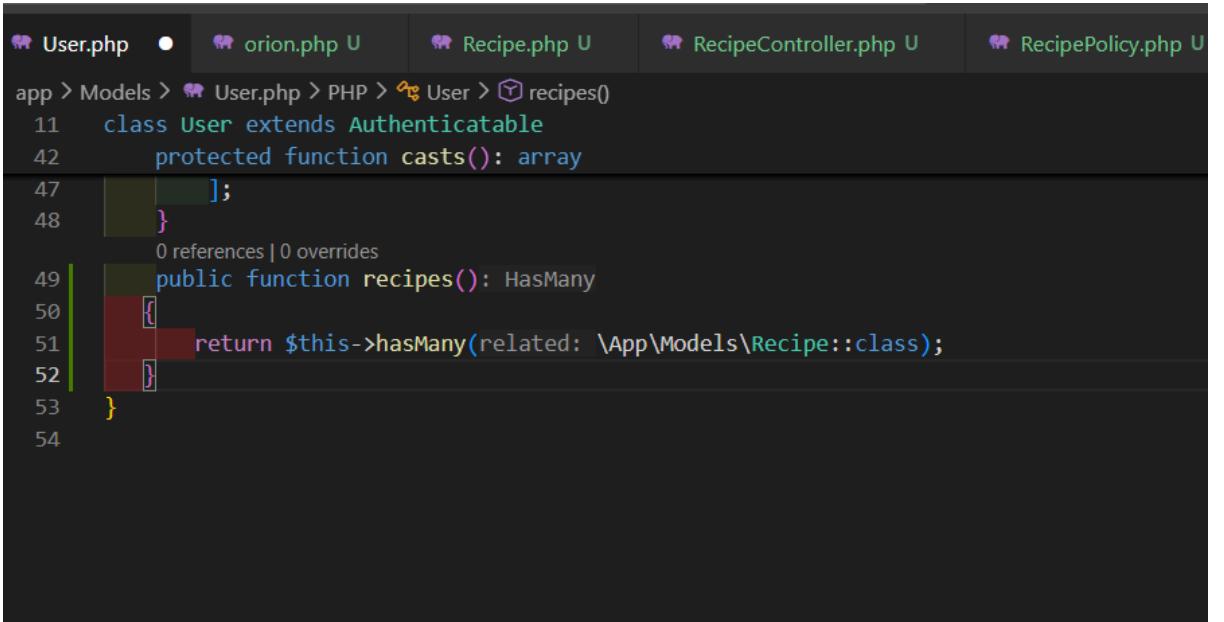
[INFO] Running migrations.

2026_01_13_182521_add_user_id_to_recipes_table ..... 64.94ms DONE

C:\laragon\www\Foodsy whole file
λ
```

Tras tener la migración tenemos que relacionar los modelos, decirle que tipo de relación tienen, esto lo tenemos que hacer siempre para que Orion devuelva los datos correspondientes.

Primero vamos al modelo user, un usuario puede tener muchas recipes y una recipe pertenece a un usuario por lo que la relación es 1 N y la FK de user va a posts, por lo que añadimos lo siguiente:



```
User.php ● orion.php U Recipe.php U RecipeController.php U RecipePolicy.php U
app > Models > User.php > PHP > User > recipes()
11 class User extends Authenticatable
42     protected function casts(): array
47         [
48     ]
49     public function recipes(): HasMany
50     {
51         return $this->hasMany(related: \App\Models\Recipe::class);
52     }
53 }
54
```

Y dentro de recipe.php:

```

User.php M orion.php U Recipe.php U RecipeController.php U
app > Models > Recipe.php > PHP Intelephense > Recipe > user
7 class Recipe extends Model
8     0 references
9     protected $fillable = ['titulo', 'foto', 'pasos'];
10    0 references | 0 overrides
11    public function user(): BelongsTo
12    {
13        return $this->belongsTo(related: \App\Models\User::class);
14    }
15

```

Ahora para que todo sea más seguro y el back se encargue totalmente de la base de datos lo que tenemos que hacer es una función que recoja el user del bearer token y lo asigne a esa receta, eso lo hacemos dentro del RecipeController:

```

User.php M orion.php U Recipe.php U RecipeController.php U X Auth
app > Http > Controllers > Api > RecipeController.php > PHP Intelephense > RecipeController > before
9 class RecipeController extends Controller
10    public function validationRules(): array
11    {
12        0 references | 0 overrides
13        protected function beforeStore(Request $request, $recipe): void
14        {
15            $recipe->user_id = $request->user()->id;
16        }
17    }
18

```

Ahora con esto establecido, tenemos que crear un endpoint para que un usuario pueda ver sus recetas para eso lo primero es ir al api.php dentro de routes y hacer el endpoint:

```

Route::middleware('auth:sanctum')->get('/my-recipes',
[AuthTokenController::class, 'myRecipes']);

```

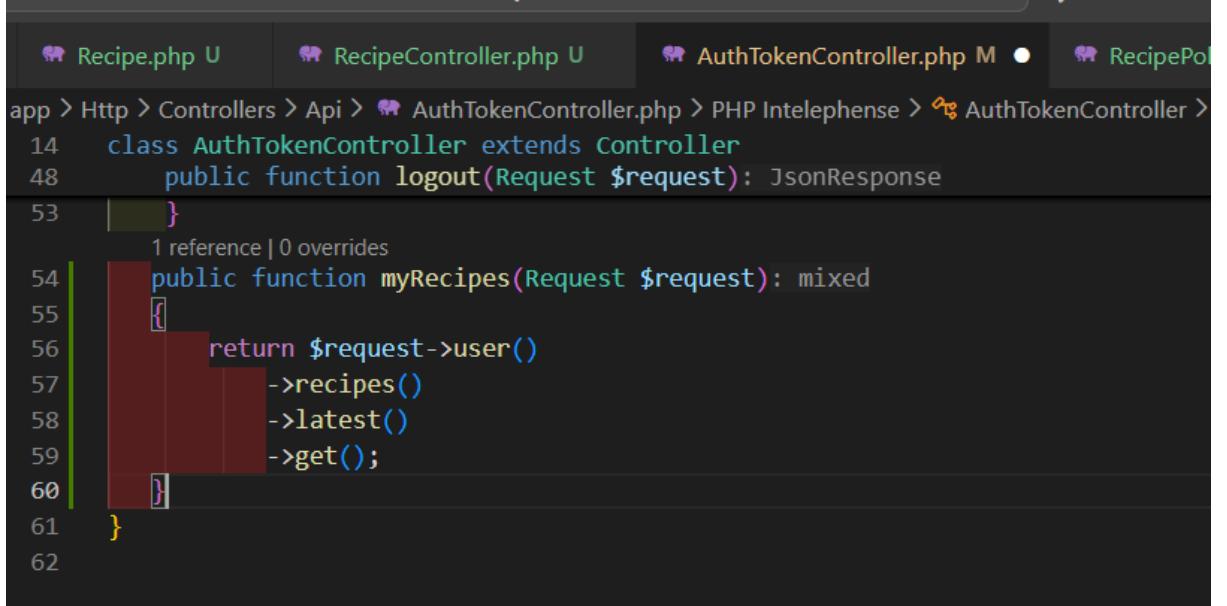
O puedes meterlo con el resto de nuestros propios endpoints:

```

Route::middleware('auth:sanctum')->group(function () {
    Route::get('/me', fn (Request $request) => $request->user());
    Route::post('/logout', [AuthTokenController::class,
    'logout']);
    Route::get('/my-recipes', [AuthTokenController::class,
    'myRecipes']);
});

```

Como vimos anteriormente estamos llamando al método myRecipes pero no lo tenemos, tenemos que implementarlo, pero esta vez dentro del AuthTokenController, porque hay que acceder a la request:



```
app > Http > Controllers > Api > AuthTokenController.php > PHP Intelephense > AuthTokenController >
14     class AuthTokenController extends Controller
48         public function logout(Request $request): JsonResponse
53     }
54     1 reference | 0 overrides
54     public function myRecipes(Request $request): mixed
55     {
56         return $request->user()
57             ->recipes()
58             ->latest()
59             ->get();
60     }
61 }
62
```

Para quedar mejor podemos añadir una última cosa y es cambiar el get() por paginate(10) por ejemplo, así ya tenemos los resultados paginados para el front.

```
public function myRecipes(Request $request)
{
    return $request->user()
        ->recipes()
        ->latest()
        ->paginate(10);
}
```

Por último tenemos que ajustar las policies para que cada usuario solo pueda borrar o editar sus propias recetas. Para eso tenemos que cambiar el update y delete del archivo:

```
public function update(User $user, Recipe $recipe): bool
{
    return $recipe->user_id === $user->id;
}

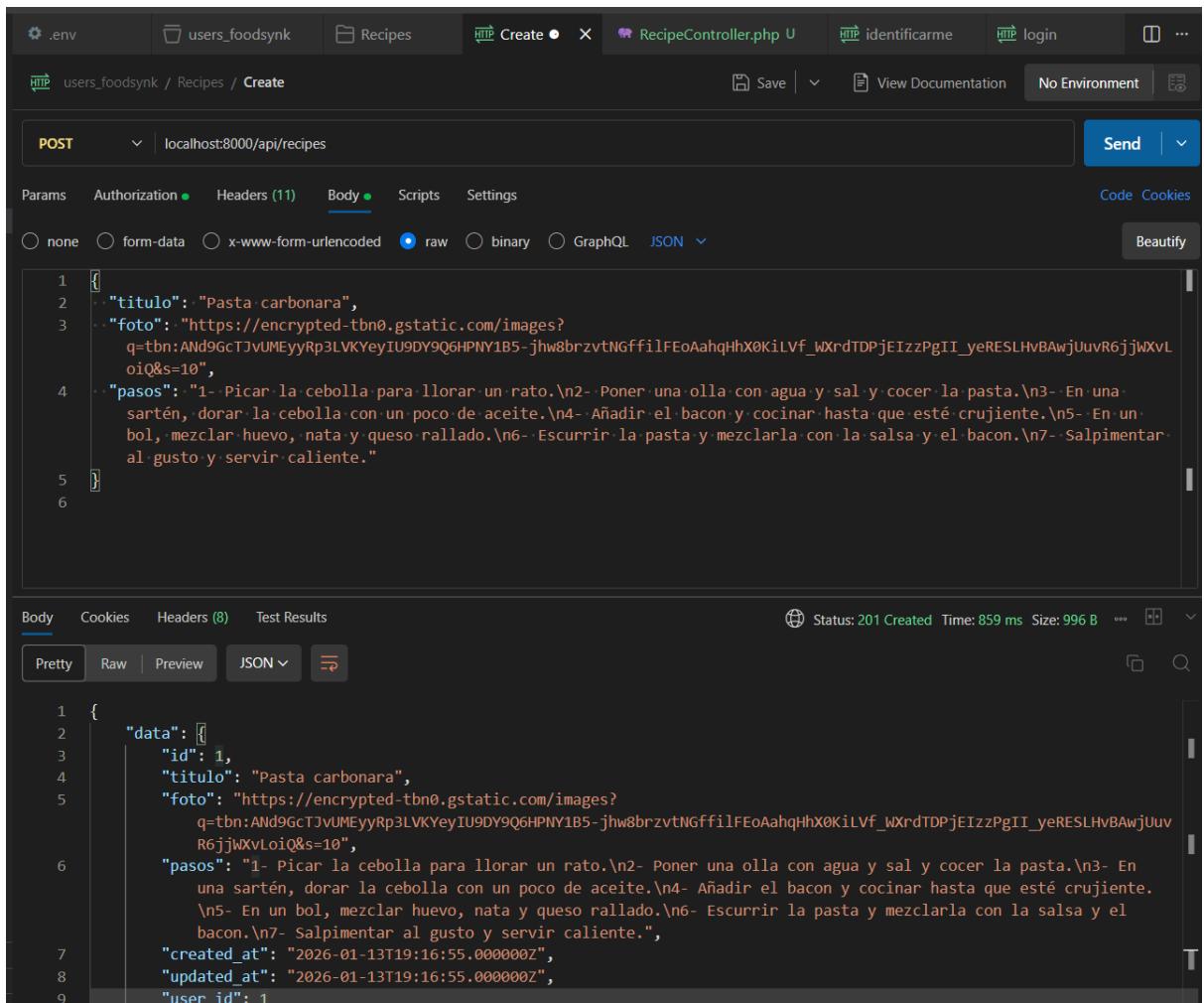
public function delete(User $user, Recipe $recipe): bool
{
    return $recipe->user_id === $user->id;
}
```

Vamos ahora con las pruebas de postman para verificar todo lo que hemos hecho. Para ello, volvemos a abrir Tinker y creamos un nuevo usuario:

```
\App\Models\User::create(['name' => 'pablo', 'email' => 'pablo@test.com',  
'password' => bcrypt('password'), ]);
```

```
C:\laragon\www\Foodsynk(main -> origin)  
λ php artisan tinker  
Psy Shell v0.12.18 (PHP 8.3.28 - cli) by Justin Hileman  
New PHP manual is available (latest: 3.0.1). Update with `doc --update-manual`  
> \App\Models\User::create(['name' => 'pablo', 'email' => 'pablo@test.com', 'password' => bcrypt('password'), ]);  
= \App\Models\User {#6585  
  +name: "pablo",  
  +email: "pablo@test.com",  
  +password: "$2y$12$f2cJYJ.uo/8A2xdXL/LbOZG0./wmYbC39m8u74x/Le0MOACvfDhe",  
  +updated_at: "2026-01-13 19:06:39",  
  +created_at: "2026-01-13 19:06:39",  
  +id: 2,  
  +sub: function logout(Request $request)  
}>
```

Ahora nos vamos a postman y yo cree dentro de mi colección una carpeta para los endpoints de recipes. Y creamos una nueva request con post para el create.



The screenshot shows the Postman interface with the following details:

- Collection:** users_foodsynk / Recipes
- Request Type:** POST
- URL:** localhost:8000/api/recipes
- Body:** Raw JSON (selected)
- JSON Content:**

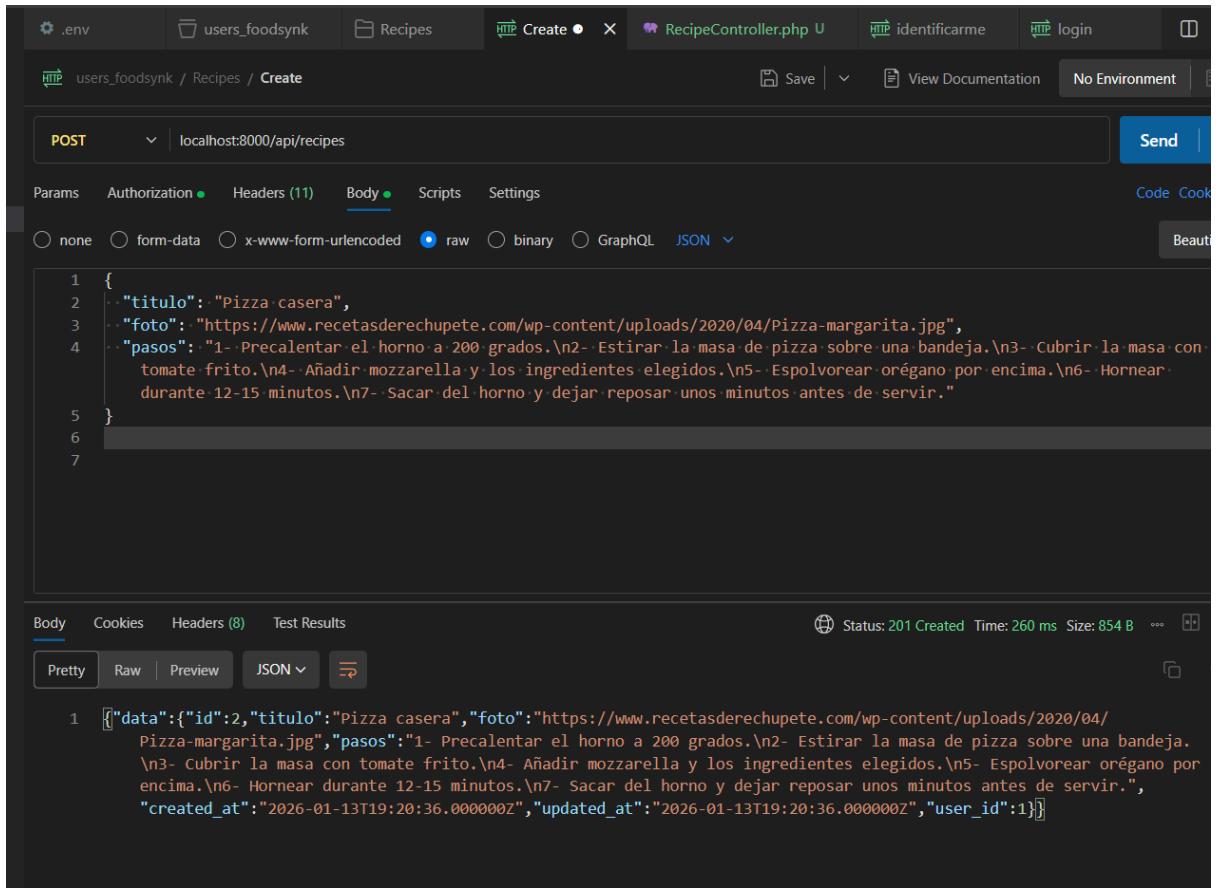
```
1 {  
2   "titulo": "Pasta carbonara",  
3   "foto": "https://encrypted-tbn0.gstatic.com/images?  
        q=tbn:ANd9GcTjvUMEyyRp3LVKYeyIU9DY9Q6HPNY1B5-jhw8brzvtNGffilFEoAahqHhx0KilVf_wXrdTDPjEIZzPgII_yeRESLhvBAwjUuvR6jjWxVL  
        oijQ&s=10",  
4   "pasos": "1- Picar la cebolla para llorar un rato.\n2- Poner una olla con agua y sal y cocer la pasta.\n3- En una  
        sartén, dorar la cebolla con un poco de aceite.\n4- Añadir el bacon y cocinar hasta que esté crujiente.\n5- En un  
        bol, mezclar huevo, nata y queso rallado.\n6- Escurrir la pasta y mezclarla con la salsa y el bacon.\n7- Salpicar  
        al gusto y servir caliente."  
5 }  
6 }
```
- Response Headers:** Status: 201 Created Time: 859 ms Size: 996 B
- Response Body (Pretty JSON):**

```
1 {  
2   "data": [  
3     "id": 1,  
4     "titulo": "Pasta carbonara",  
5     "foto": "https://encrypted-tbn0.gstatic.com/images?  
            q=tbn:ANd9GcTjvUMEyyRp3LVKYeyIU9DY9Q6HPNY1B5-jhw8brzvtNGffilFEoAahqHhx0KilVf_wXrdTDPjEIZzPgII_yeRESLhvBAwjUuv  
            R6jjWxVlojQ&s=10",  
6     "pasos": "1- Picar la cebolla para llorar un rato.\n2- Poner una olla con agua y sal y cocer la pasta.\n3- En  
            una sartén, dorar la cebolla con un poco de aceite.\n4- Añadir el bacon y cocinar hasta que esté crujiente.  
            \n5- En un bol, mezclar huevo, nata y queso rallado.\n6- Escurrir la pasta y mezclarla con la salsa y el  
            bacon.\n7- Salpicar al gusto y servir caliente.",  
7     "created_at": "2026-01-13T19:16:55.000000Z",  
8     "updated_at": "2026-01-13T19:16:55.000000Z",  
9     "user_id": 1
```

El controller de recipes tiene que apuntar a orion **Importante**.

Para la petición tenemos que meter el token del usuario que va a cambiar la receta, el header con accept application/json y el body con raw.

Vamos a meter 2 recetas con este usuario y otras dos con el nuevo.



The screenshot shows a Postman interface with the following details:

- Request URL:** `localhost:8000/api/recipes`
- Method:** POST
- Body Content:**

```
1 {
2   "titulo": "Pizza casera",
3   "foto": "https://www.recetasderechupete.com/wp-content/uploads/2020/04/Pizza-margarita.jpg",
4   "pasos": "1- Precalentar el horno a 200 grados.\n2- Estirar la masa de pizza sobre una bandeja.\n3- Cubrir la masa con tomate frito.\n4- Añadir mozzarella y los ingredientes elegidos.\n5- Espolvorear orégano por encima.\n6- Hornear durante 12-15 minutos.\n7- Sacar del horno y dejar reposar unos minutos antes de servir."
5 }
6
7 }
```
- Response Status:** 201 Created
- Response Time:** 260 ms
- Response Size:** 854 B
- Response Body (Pretty JSON):**

```
[{"data":{"id":2,"titulo":"Pizza casera","foto":"https://www.recetasderechupete.com/wp-content/uploads/2020/04/Pizza-margarita.jpg","pasos":"1- Precalentar el horno a 200 grados.\n2- Estirar la masa de pizza sobre una bandeja.\n3- Cubrir la masa con tomate frito.\n4- Añadir mozzarella y los ingredientes elegidos.\n5- Espolvorear orégano por encima.\n6- Hornear durante 12-15 minutos.\n7- Sacar del horno y dejar reposar unos minutos antes de servir.", "created_at":"2026-01-13T19:20:36.000000Z", "updated_at":"2026-01-13T19:20:36.000000Z", "user_id":1}]
```

A la hora de meter las recetas con el otro usuario. Tenemos que usar el endpoint de logout y luego loguearnos con el nuevo usuario.

The screenshot shows a Postman interface with the following details:

- Request URL:** `localhost:8000/api/recipes`
- Method:** POST
- Body Content:**

```
1 {
2   "titulo": "Tacos de pollo",
3   "foto": "https://www.comedera.com/wp-content/uploads/2022/05/tacos-de-pollo.jpg",
4   "pasos": "1- Cortar el pollo en tiras pequeñas.\n2- Salpimentar y añadir especias al gusto.\n3- Saltear el pollo en una sartén con aceite.\n4- Calentar las tortillas de trigo o maíz.\n5- Rellenar las tortillas con el pollo.\n6- Añadir lechuga, tomate y salsa.\n7- Servir calientes."
5 }
6
7 }
```
- Response Status:** 201 Created
- Response Time:** 267 ms
- Response Size:** 821 B

Como me equivoqué añadiendo una receta de más, nos viene que ni pintado porque vamos a probar el endpoint para borrar. Para ello creamos una nueva request delete, le metemos el token de nuestro usuario pq sino no va a funcionar y le metemos en la url la receta a borrar.

The screenshot shows a Postman interface with the following details:

- Request URL:** `localhost:8000/api/recipes/1`
- Method:** `DELETE`
- Authorization:** `Bearer Token` (selected)
- Token:** `6j9ESs98ID0ikbqpxRfAbu4uYJhpo2x1r70...`
- Status:** `403 Forbidden`
- Time:** `276 ms`
- Size:** `13.96 KB`
- Response Body (Pretty JSON):**

```
1 [ { "message": "This action is unauthorized.", "exception": "Symfony\\Component\\HttpKernel\\Exception\\AccessDeniedHttpException", "file": "C:\\laragon\\www\\Foodsynk\\vendor\\laravel\\framework\\src\\Illuminate\\Foundation\\Exceptions\\Handler.php", "line": 672, "trace": [ { "file": "C:\\laragon\\www\\Foodsynk\\vendor\\laravel\\framework\\src\\Illuminate\\Foundation\\Exceptions\\Handler.php", "line": 616, "function": "prepareException", "class": "Illuminate\\Foundation\\Exceptions\\Handler" } ] } ]
```

En esta captura se ve que no nos deja borrar recetas que no sean nuestras.

The screenshot shows the Postman interface with the following details:

- Header Bar:** Shows various API endpoints like "Create", "All_recipes", "Delete_recipes", "logout", "RecipeController.php U", "identificarme", "login", and environment settings.
- Request URL:** users_foodsync / Delete_recipes
- Method:** DELETE
- URL:** localhost:8000/api/recipes/4
- Authorization:** Type is set to "Bearer Token" and the token value is Gj9ESs98lD0ikbqpxRfABu4uYJhpo2x1r70...
- Body:** Headers (8) tab is selected. Body tab shows JSON response.
- Response Headers:** Status: 200 OK, Time: 188 ms, Size: 869 B
- Response Body (Pretty JSON):**

```
1 [ { 2   "data": { 3     "id": 4, 4       "titulo": "Hamburguesa casera", 5       "foto": "https://www.cocinacaserayfacil.net/wp-content/uploads/2018/01/Hamburguesa-casera.jpg", 6       "pasos": "1- Salpimentar la carne picada y formar las hamburguesas.\n2- Calentar una sartén o plancha con un poco de aceite.\n3- Cocinar la hamburguesa por ambos lados.\n4- Tostar ligeramente el pan.\n5- Colocar la hamburguesa en el pan y añadir queso.\n6- Añadir lechuga, tomate y salsas al gusto.\n7- Cerrar el pan y servir.", 7     "created_at": "2026-01-13T19:22:06.000000Z", 8     "updated_at": "2026-01-13T19:22:06.000000Z", 9     "user_id": 2 10   } 11 }
```

Por último hoy, hice lo del antigravity que tendrá su documento aparte.
Y subí el proyecto a github.

15/01/26 18:48-19:13 20:17-

Hoy vamos a enfocar nos en hacer la parte del front del login, logout y el registro.

Vamos a crear el proyecto con:

npm create vite@latest Foodsynk-front -- --template react

```
C:\Users\Nuria\Desktop\DAW\segundo\proyecto\proyecto_front>npm create vite@latest Foodsynk-front -- --template react
> npx
> create-vite Foodsynk-front --template react
|
|   o Package name:
|     foodsynk
|
|   o Use rolldown-vite (Experimental)?:
|     No
|
|   o Install with npm and start now?
|     Yes
|
|   o Scaffolding project in C:\Users\Nuria\Desktop\DAW\segundo\proyecto\proyecto_front\Foodsynk-front...
|
|   o Installing dependencies with npm...
|
|
```

no funciona el nombre con mayúsculas

Vamos a empezar instalando axios , que es el que nos va a permitir realizar las peticiones HTTP en entornos js o node. Además de instalar react router Dom que es una librería esencial para un proyecto de React, nos permite gestionar la navegación entre diferentes componentes o páginas sin recargar la página completa.

npm i axios react-router-dom

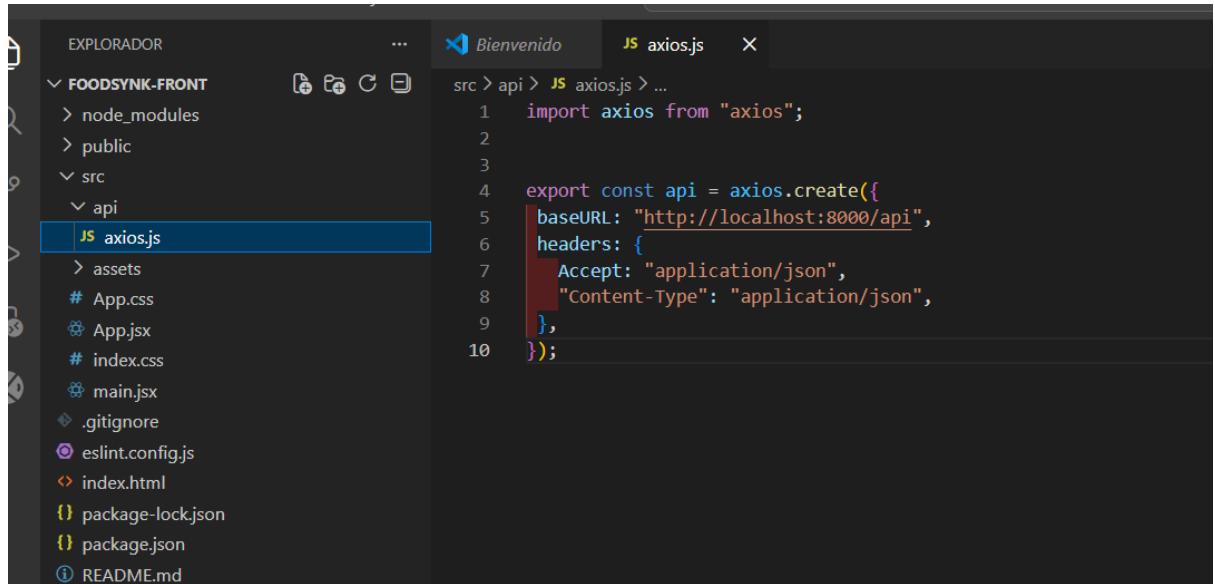
```
C:\Users\Nuria>cd C:\Users\Nuria\Desktop\DAW\segundo\proyecto\proyecto_front\Foodsynk-front
C:\Users\Nuria\Desktop\DAW\segundo\proyecto\proyecto_front\Foodsynk-front>npm i axios react-router-dom
added 27 packages, and audited 185 packages in 2s

40 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities

C:\Users\Nuria\Desktop\DAW\segundo\proyecto\proyecto_front\Foodsynk-front>
```

Ahora viene lo importante vamos a configurar axios, lo primero dentro de la carpeta src vamos a crear una carpeta llamada api y dentro un archivo axios.js , aquí vamos a centralizar toda la lógica :



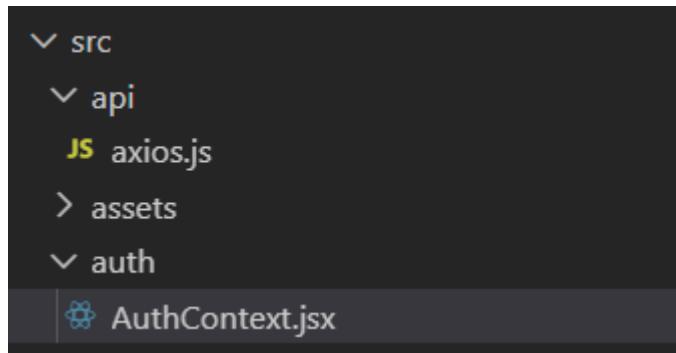
The screenshot shows the VS Code interface. On the left, the Explorer sidebar displays the project structure under 'FOODSYNK-FRONT': node_modules, public, src (which contains api and axios.js), assets, App.css, App.jsx, index.css, main.jsx, .gitignore, eslint.config.js, index.html, package-lock.json, package.json, and README.md. The 'api' folder and 'axios.js' file are selected. On the right, the code editor shows the content of 'axios.js':

```
src > api > JS axios.js > ...
1 import axios from "axios";
2
3
4 export const api = axios.create({
5   baseURL: "http://localhost:8000/api",
6   headers: {
7     Accept: "application/json",
8     "Content-Type": "application/json",
9   },
10 });

Vale ahora toca crear un context , esto es super importante para poder guardar el token de manera segura en la memoria, no lo queremos dentro del localStorage, para ello usaremos hooks de react, como useState, useMemo..
```

Vale ahora toca crear un context , esto es super importante para poder guardar el token de manera segura en la memoria, no lo queremos dentro del localStorage, para ello usaremos hooks de react, como useState, useMemo..

Como todo esto va a ir sobre la autenticación crearemos una carpeta llamada auth dentro de src y dentro tres archivos, El primero será el contexto en si, necesitamos un contexto para que guarde las variables que queremos (user y token), y que cambiemos de componente sin perder esa información. Se llamará AuthContext.jsx:



Este se rellena con esto:

```
import { createContext } from "react";

export const AuthContext = createContext(null);
```

El segundo tendrá toda la lógica (cerebro) para setear en el contexto el token y el usuario al con el que nos estamos logueando, se llamará AuthProvider :

```
import { useMemo, useState } from "react";
import { AuthContext } from "./AuthContext";

export function AuthProvider({ children }) {
  const [token, setToken] = useState(null);
  const [user, setUser] = useState(null);

  const value = useMemo(
    () => ({ token, setToken, user, setUser }),
    [token, user]
  );

  return <AuthContext.Provider
    value={value}>{children}</AuthContext.Provider>;
}
```

Por último tendremos el hook useAuth.js que es nuestro hook personalizado para usar el contexto:

```
import { useContext } from "react";
import { AuthContext } from "./AuthContext";

export function useAuth() {
  const ctx = useContext(AuthContext);
  if (!ctx) throw new Error("useAuth must be used inside <AuthProvider>");
  return ctx;
}
```

COMPONENTES

Ahora vamos a empezar a crear los componentes, lo primero que vamos a hacer es crear una carpeta pages para tener todo bien organizado, dentro vamos a crear el Login.jsx :

```
import { useState } from "react";
import { api } from "../api/axios";
import { useAuth } from "../auth/useAuth";
import styles from "./Login.module.css";

export default function Login() {
  const { setToken, setUser } = useAuth();

  const [email, setEmail] = useState("");
  const [password, setPassword] = useState("");
  const [error, setError] = useState("");
  const [loading, setLoading] = useState(false);

  async function handleSubmit(e) {
    e.preventDefault();
    setError("");
    setLoading(true);

    try {
      const res = await api.post("/login", {
        email,
        password,
        device_name: "react",
      });
      setToken(res.data.token);
    } catch (err) {
      setError("Error al iniciar sesión");
    }
  }

  return (
    <div>
      <h1>Iniciar Sesión</h1>
      <form>
        <input type="text" value={email} onChange={(e) => setEmail(e.target.value)} placeholder="Email" />
        <input type="password" value={password} onChange={(e) => setPassword(e.target.value)} placeholder="Contraseña" />
        <button type="button" onClick={handleSubmit}>Iniciar Sesión</button>
      </form>
      {error ? <p>{error}</p> : null}
    </div>
  );
}
```

```
        setUser(res.data.user);

        alert("Login OK: " + res.data.user.email);
    } catch (err) {
        const msg =
            err?.response?.data?.message
        JSON.stringify(err?.response?.data)

        "Error al iniciar sesión";
        setError(msg);
    } finally {
        setLoading(false);
    }
}

return (
    <div className={styles.container}>
        <h2 className={styles.title}>Login</h2>

        <form onSubmit={handleSubmit} className={styles.form}>
            <label className={styles.label}>
                Email
                <input
                    className={styles.input}
                    value={email}
                    onChange={(e) => setEmail(e.target.value)}
                    autoComplete="email"
                />
            </label>

            <label className={styles.label}>
```

```

        Password

<input
    className={styles.input}
    value={password}
    onChange={(e) => setPassword(e.target.value)}
    type="password"
    autoComplete="current-password"
/>
</label>

<button className={styles.button} disabled={loading}>
    {loading ? "Entrando..." : "Entrar"}
</button>

{error && <div className={styles.error}>{error}</div>}
</form>
</div>
);
}

```

Ahora por mucho que lo tengamos hecho no lo estamos llamando en ningún sitio, tenemos que meterlo en el main.jsx de react para que se vea:

```

import React from "react";
import ReactDOM from "react-dom/client";
import { BrowserRouter, Routes, Route } from "react-router-dom";
import {AuthProvider} from "./auth/AuthProvider";
import Login from "./pages/Login";

function App() {
  return (
    <Routes>
      <Route path="/" element={<Login />} />
    </Routes>
  );
}


```

```

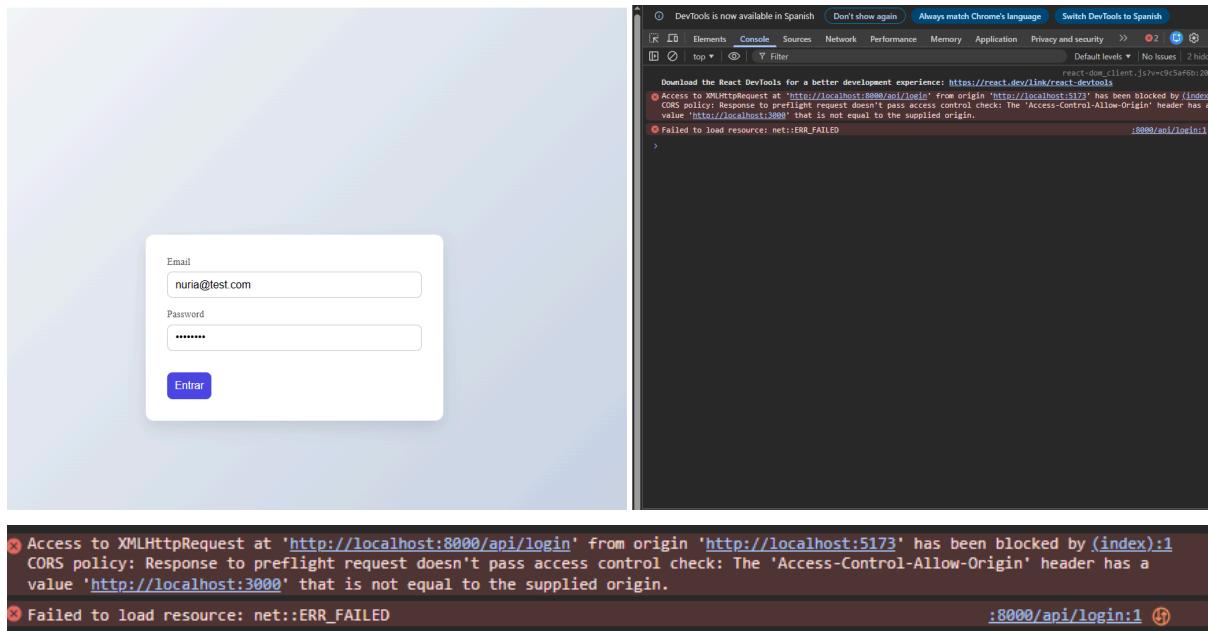
ReactDOM.createRoot(document.getElementById("root")).render(
  <React.StrictMode>
    <AuthProvider>
      <BrowserRouter>
        <App />
      </BrowserRouter>
    </AuthProvider>
  </React.StrictMode>
);

```

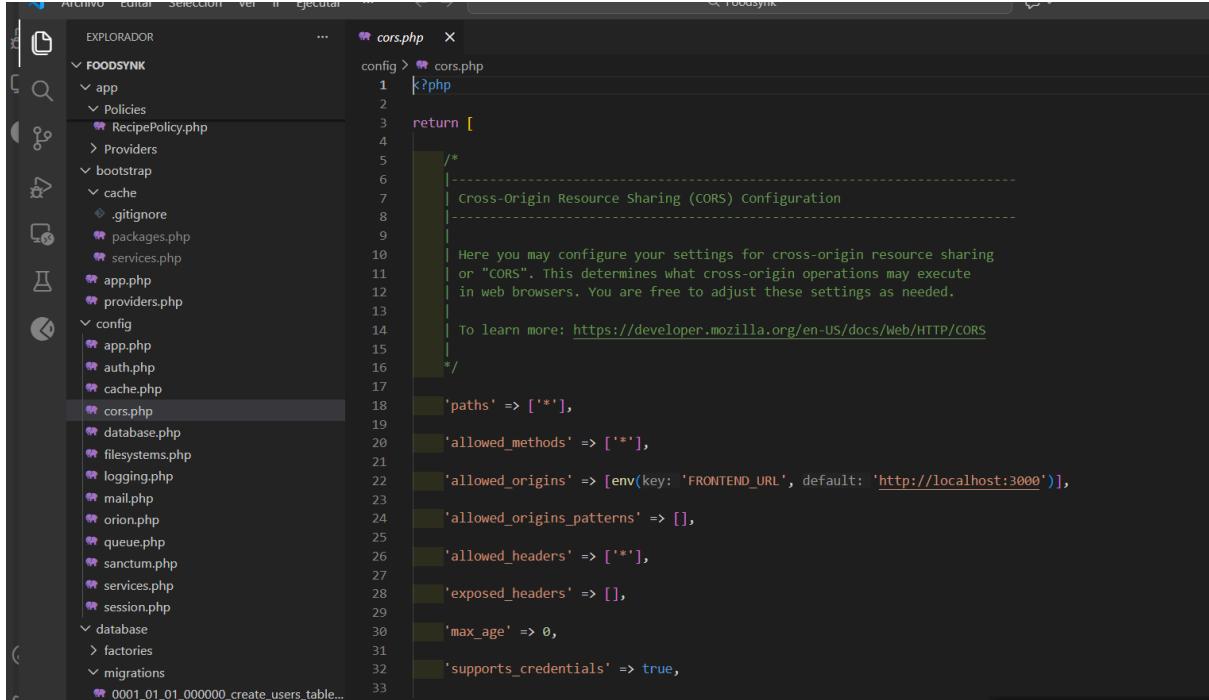
Tenemos que crearle también una hoja de estilos al login, dentro de pages que se llame Login.module.css

El funcionamiento es complejo pero se entiende, el Authprovider con su contexto contiene dentro las rutas que hemos creado, en este caso solo tenemos el login, y dentro la app.

Vale si todo sale bien, como dice el código deberíamos tener un alert con los datos del usuario pero nos enfrentamos a un problema muy común que nos vamos a encontrar, el error de CORS :



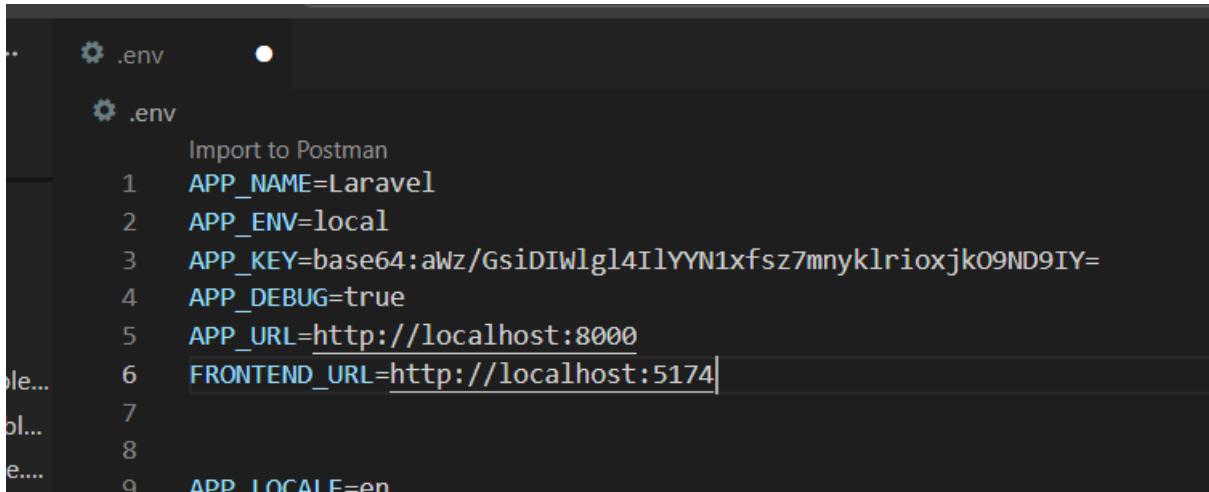
El back no permite la entrada a ese puerto, tenemos que darselo desde el back, para ello tenemos que ir a la carpeta config y buscar el archivo cors.php:



The screenshot shows a code editor with the file `cors.php` open in the `config` directory of a Laravel application named `FOODSYNK`. The file contains the following PHP code:

```
1  //?php
2
3  return [
4      /*
5      |--------------------------------------------------------------------------
6      | Cross-Origin Resource Sharing (CORS) Configuration
7      |--------------------------------------------------------------------------
8
9      |
10     | Here you may configure your settings for cross-origin resource sharing
11     | or "CORS". This determines what cross-origin operations may execute
12     | in web browsers. You are free to adjust these settings as needed.
13
14     | To learn more: https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS
15
16     */
17
18     'paths' => ['*'],
19
20     'allowed_methods' => ['*'],
21
22     'allowed_origins' => [env(key: 'FRONTEND_URL', default: 'http://localhost:3000')],
23
24     'allowed_origins_patterns' => [],
25
26     'allowed_headers' => ['*'],
27
28     'exposed_headers' => [],
29
30     'max_age' => 0,
31
32     'supports_credentials' => true,
33 ]
```

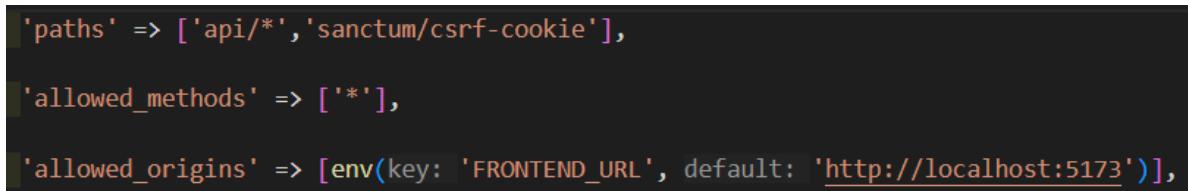
Vamos al `.env` y cambiamos la url del frontend :



The screenshot shows the `.env` file for a Laravel application. It contains the following environment variables:

```
APP_NAME=Laravel
APP_ENV=local
APP_KEY=base64:awz/GsiDIWlg14IlYYN1xfsz7mnyklrioxjk09ND9IY=
APP_DEBUG=true
APP_URL=http://localhost:8000
FRONTEND_URL=http://localhost:5174
APP_LOCALE=en
```

Ahora vamos al `cors.php` y cambiamos el paths y el frontend url en el 5173



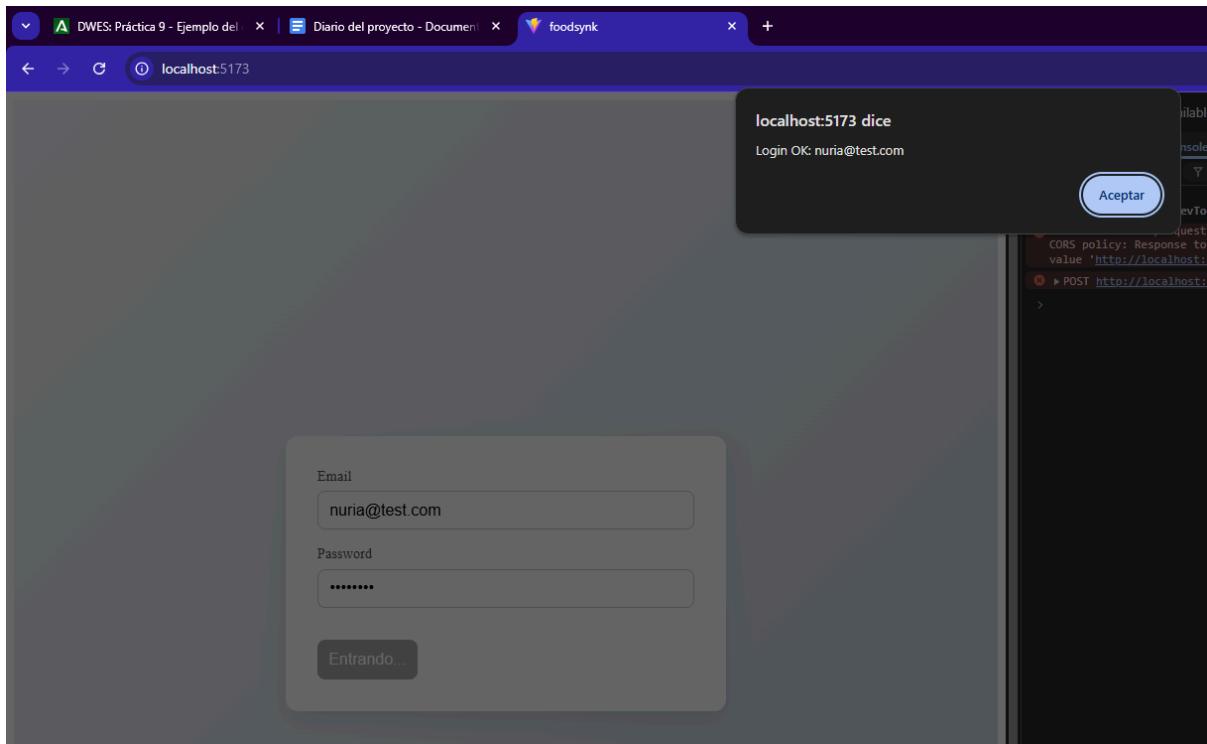
```
'paths' => ['api/*', 'sanctum/csrf-cookie'],

'allowed_methods' => ['*'],

'allowed_origins' => [env(key: 'FRONTEND_URL', default: 'http://localhost:5173')],
```

Ahora hay que reiniciar el servicio de back porque cuando tu cambias en el `.env`, tienes que reiniciar para que se carguen los nuevos datos.

Las personas solo tendran acceso desde el front y solo tendrá acceso a las rutas de la api.



Ahora tenemos que buscar la persistencia de ese token, meterlo en el contexto para ello usaremos sessionStorage porque es más seguro que localStorage y suficiente para una SPA con autenticación por Bearer token, en el AuthProvider:

```
import { useMemo, useState, useEffect } from "react";
import { AuthContext } from "./AuthContext";

export function AuthProvider({ children }) {
  const [token, setToken] = useState(null);
  const [user, setUser] = useState(null);

  useEffect(() => {
    if (token) {
      sessionStorage.setItem("token", token);
    } else {
      sessionStorage.removeItem("token");
    }
  }, [token]);

  const value = useMemo(
    () => ({ token, setToken, user, setUser }),
    [token, user]
);
```

```
    return <AuthContext.Provider  
value={value}>{children}</AuthContext.Provider>;  
}
```

Ahora tenemos que cambiar el login, para que nos setee las variables y nos lleve a la página home donde vamos a ver todas las recetas de la app:

```
import { useState } from "react";
import { api } from "../api/axios";
import { useAuth } from "../auth/useAuth";
import { useNavigate } from "react-router-dom";
import styles from "./Login.module.css";

export default function Login() {
  const { setToken, setUser } = useAuth();
  const navigate = useNavigate();

  const [email, setEmail] = useState("");
  const [password, setPassword] = useState("");
  const [error, setError] = useState("");
  const [loading, setLoading] = useState(false);

  async function handleSubmit(e) {
    e.preventDefault();
    setError("");
    setLoading(true);

    try {
      const res = await api.post("/login", { email, password });

      setToken(res.data.token);
      setUser(res.data.user);

      navigate("/home");
    } catch (err) {
      console.error("LOGIN ERROR:", err);
    }
  }
}
```

```
const msg =
  err?.response?.data?.message ||
  (typeof err?.response?.data === "string" ? err.response.data :
 "") ||

  (err?.response?.data ? JSON.stringify(err.response.data) : "")

||

  err?.message ||
  "Error al iniciar sesión";

  setError(msg);
} finally {
  setLoading(false);
}
}

return (
<div className={styles.container}>
  <h2 className={styles.title}>Login</h2>

  <form onSubmit={handleSubmit} className={styles.form}>
    <label className={styles.label}>
      Email
      <input
        className={styles.input}
        value={email}
        onChange={(e) => setEmail(e.target.value)}
        autoComplete="email"
      />
    </label>

    <label className={styles.label}>
      Password
      <input
        className={styles.input}
        value={password}
        onChange={(e) => setPassword(e.target.value)}
        type="password"
      />
    </label>
  </form>
</div>
)
```

```

        autoComplete="current-password"
      />
    </label>

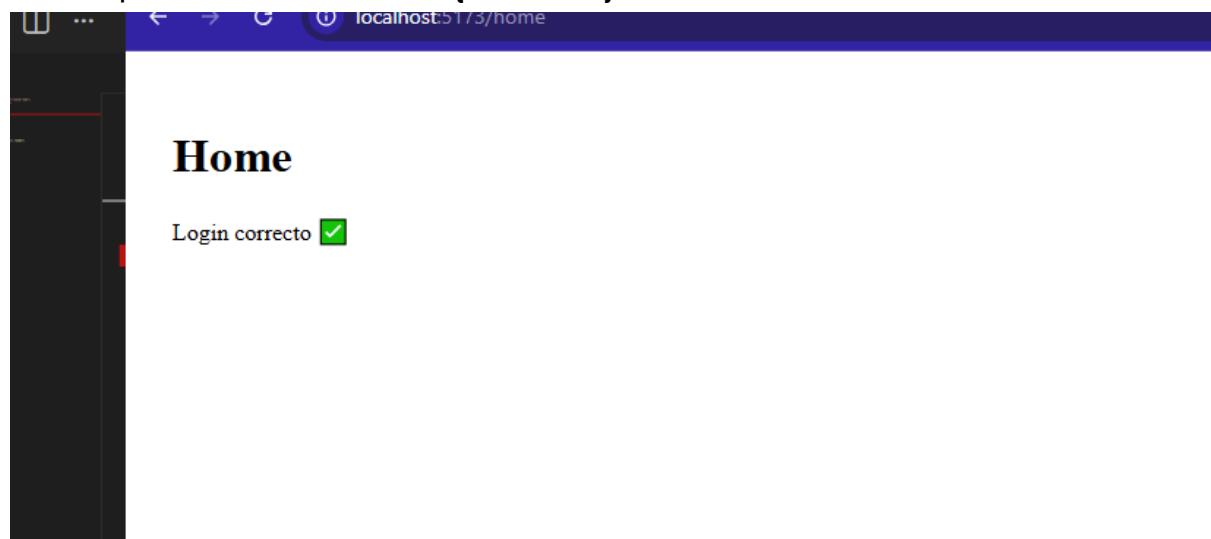
    <button className={styles.button} disabled={loading}>
      {loading ? "Entrando..." : "Entrar"}
    </button>

    {error && <div className={styles.error}>{error}</div>}
  </form>
</div>
);
}

```

Crearemos un componente Home.jsx provisional y lo importamos en el main añadiendo en routes:

```
<Route path="/home" element={<Home />} />
```



Ahora tenemos que hacer que las rutas sean seguras

Para hacer las rutas seguras vamos a cambiar el AuthProvider, para que quede de las siguiente manera:

```

import { useMemo, useState, useEffect } from "react";
import { AuthContext } from "./AuthContext";

export function AuthProvider({ children }) {
  const [token, setToken] = useState(null);
  const [user, setUser] = useState(null);
}

```

```

useEffect(() => {
  const savedToken = sessionStorage.getItem("token");
  if (savedToken) setToken(savedToken);
}, []);

useEffect(() => {
  if (token) {
    sessionStorage.setItem("token", token);
  } else {
    sessionStorage.removeItem("token");
  }
}, [token]);

const value = useMemo(
  () => ({ token, setToken, user, setUser }),
  [token, user]
);

return <AuthContext.Provider
value={value}>{children}</AuthContext.Provider>;
}

```

Lo que hace es que cuando se abre la app está null, y después cada vez que nos movamos consultara el session.storage Ya con esto tenemos que crear un archivo que nos proteja las rutas, si no hay token nos redireccionará al login.

src/auth/ProtectedRoute.jsx

```

import { Navigate, Outlet } from "react-router-dom";
import { useAuth } from "./useAuth";

export default function ProtectedRoute() {
  const { token } = useAuth();
  if (!token) return <Navigate to="/" replace />;
  return <Outlet />;
}

```

Si no hay token redirige a login, si no redirige a donde toca.

Ahora en el main.jsx ponemos las rutas que queremos que estén protegidas:

```
function App() {
  return (
    <Routes>
      <Route path="/" element={<Login />} />

      {/* Rutas protegidas */}
      <Route element={<ProtectedRoute />} >
        <Route path="/home" element={<Home />} />
      </Route>
    </Routes>
  );
}
```

Importante importar el ProtectedRoute en el main. Y podemos probarlo.

Ahora puede pasar algo que no está bien en cuanto a funcionamiento, ahora una persona con token puede ver el login, y no debería ser así, tenemos que ponerle que si hay token le redireccione a home, para eso vamos al archivo del login y añadimos:

```
const { token, setToken, setUser } = useAuth();
```

y tras todos los const:

```
if (token) return <Navigate to="/home" replace />;
```

y en el import con useNavigate añadimos Navigate

Para poder consumir la api de manera más cómodo con sus endpoints podemos hacer un navbar pero eso ya sería un componente no una page por lo que tendríamos la carpeta components

```
import { useNavigate } from "react-router-dom";
import { api } from "../api/axios";
import { useAuth } from "../auth/useAuth";

export default function Navbar() {
  const { setToken, setUser } = useAuth();
  const navigate = useNavigate();
```

```

    async function handleLogout() {
      try {
        await api.post("/logout");
      } catch (e) {
        console.warn("Error en logout backend");
      } finally {
        setToken(null);
        setUser(null);
        navigate("/", { replace: true });
      }
    }

    async function goToMyRecipes() {
      navigate("/my-Recipes");
    }

    return (
      <nav
        style={{ 
          display: "flex",
          gap: 12,
          padding: 12,
          borderBottom: "1px solid #ddd",
        }} 
      >
        <button onClick={() => navigate("/home")}>Home</button>
        <button onClick={goToMyRecipes}>Mis Recetas</button>
        <button onClick={handleLogout}>Logout</button>
      </nav>
    );
}

```

Ahora para que se vean solo en las rutas protegidas podemos hacer un layout con ellas únicamente, dentro del carpeta layouts:
Y creamos un archivo ProtectedLayout.jsx:

```

import Navbar from "../components/Navbar";
import { Outlet } from "react-router-dom";

```

```

export default function ProtectedLayout() {
  return (
    <>
      <Navbar />
      <Outlet />
    </>
  );
}

```

Y ahora nos vamos al main y cambiamos nuestro Routes por esto:

```

<Routes>
  {/* Pública */}
  <Route path="/" element={<Login />} />

  {/* Protegidas */}
  <Route element={<ProtectedRoute />}>
    <Route element={<ProtectedLayout />}>
      <Route path="/home" element={<Home />} />
      <Route path="/my-recipes" element={<MyRecipes />} />
    </Route>
  </Route>
</Routes>

```

Y tenemos que importar el ProtectedLayout.

La lógica de esto es , el protected route protege las rutas , el protectedlayout pinta el navbar. Ahora tenemos que hacer la página myRecipes

```

import { useEffect, useState } from "react";
import { api } from "../api/axios";
import "./MyRecipes.css";

export default function MyRecipes() {
  const [recipes, setRecipes] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState("");

  useEffect(() => {
    async function loadRecipes() {
      try {
        setLoading(true);
        setError("");
      }

```

```
        const res = await api.get("/my-recipes");

        // Si tu API es paginada (Laravel Resource/Pagination)
        const data = res.data?.data ?? res.data;

        setRecipes(data);
    } catch (err) {
        console.error(err);
        setError("No se pudieron cargar tus recetas.");
    } finally {
        setLoading(false);
    }
}

loadRecipes();
}, []);

return (
<div className="my-recipes">
    <h1 className="my-recipes__title">Mis recetas</h1>

    {loading && <p className="my-recipes__info">Cargando...</p>}

    {!loading && error && <p
className="my-recipes__error">{error}</p>}

    {!loading && !error && recipes.length === 0 && (
        <p className="my-recipes__info">No tienes recetas todavía.</p>
    )}
}

<ul className="my-recipes__list">
    {recipes.map((recipe) => (
        <li className="my-recipes__item" key={recipe.id}>
            <div className="my-recipes__card">
                {recipe.foto && (
                    <img
                        className="my-recipes__img"
                        src={recipe.foto}
                        alt={recipe.titulo ?? "Receta"}
                        loading="lazy"
                    />
                )}

            
```

```

        <div className="my-recipes__content">
          <h2 className="my-recipes__recipe-title">
            {recipe.titulo ?? "Sin título"}
          </h2>

          {recipe.pasos && (
            <p className="my-recipes__steps">
              {String(recipe.pasos).slice(0, 140)}
              {String(recipe.pasos).length > 140 ? "..." : ""}
            </p>
          )}

        </div>
      </li>
    ))}
  </ul>
</div>
);
}

```

Tal y como está nos va a dar error de autorizacion porque no estamos mandando el bearer token, tenemos que ir al archivo axios dentro de api :

 ▶ GET <http://localhost:8000/api/my-recipes> 401 (Unauthorized)

Le programamos un interceptor , que si hay token nos lo añade a la autorización, en axios.js:

```

import axios from "axios";

export const api = axios.create({
  baseURL: "http://localhost:8000/api",
  headers: {
    Accept: "application/json",
    "Content-Type": "application/json",
  },
});

api.interceptors.request.use((config) => {
  const token = sessionStorage.getItem("token");

  if (token) {
    config.headers.Authorization = `Bearer ${token}`;
  }

  return config;
});

```

```
}

return config;
}) ;
```

Por último queremos que el home sea la vista de todos las recetas:

```
import { useEffect, useState } from "react";
import { api } from "../api/axios";

export default function Home() {
  const [recipes, setRecipes] = useState([]);

  useEffect(() => {
    async function load() {
      const res = await api.get("/recipes");

      // Si /recipes viene paginado (Laravel):
      setRecipes(res.data.data);

      // Si alguna vez viniera sin paginar, podrías usar:
      // const payload = res.data;
      // setRecipes(Array.isArray(payload) ? payload : payload.data ?? []
      []);
    }

    load();
  }, []);

  return (
    <div style={{ padding: 24 }}>
      <h1>Todas las recetas</h1>

      {recipes.length === 0 && <p>No hay recetas todavía</p>}

      <ul>
        {recipes.map((recipe) => (
          <li key={recipe.id} style={{ marginBottom: 16 }}>
            <strong>{recipe.titulo}</strong>

            {recipe.foto && (
              <div style={{ marginTop: 8 }}>
                <img
```

```

        src={recipe.foto}
        alt={recipe.titulo}
        style={{ width: 220, height: 140, objectFit: "cover",
borderRadius: 8 }}
      />
    </div>
  ) }

  <div style={{ marginTop: 8, whiteSpace: "pre-line" }}>
    {recipe.pasos}
  </div>
</li>
) )
</ul>
</div>
);
}

```

Vamos a hacer los siders:

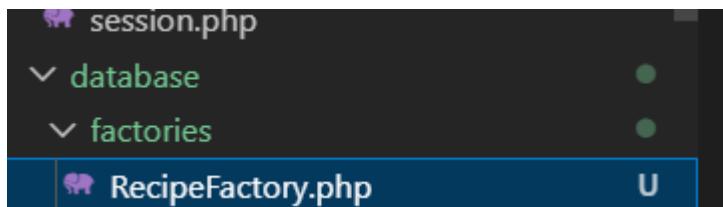
```

C:\laragon\www\Foodsy wholek(main -> origin)
λ php artisan make:factory RecipeFactory --model=Recipe
[INFO] Factory [C:\laragon\www\Foodsy wholek\database\factories\RecipeFactory.php] created successfully.

C:\laragon\www\Foodsy wholek(main -> origin)

```

esto crea esto:



Y le metemos esto:

```

<?php

namespace Database\Factories;

use Illuminate\Database\Eloquent\Factories\Factory;

class RecipeFactory extends Factory
{
    public function definition(): array
    {
        return [
            'titulo' => $this->faker->sentence(3),

```

```

        'foto' => 'recipes/default.jpg',
        'pasos' => $this->faker->paragraphs(3, true),
    ];
}
}

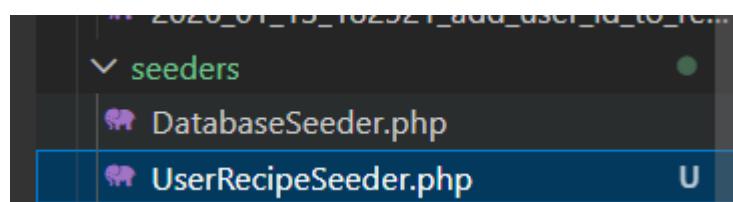
```

C:\laragon\www\Foodsynk(main -> origin) RecipeFactory.php

```

\ php artisan make:seeder UserRecipeSeeder
INFO Seeder [C:\laragon\www\Foodsynk\database\seeders\UserRecipeSeeder.php] created successfully.

```



Y en este archivo metemos esto:

```

<?php

namespace Database\Seeders;

use Illuminate\Database\Seeder;
use App\Models\User;
use App\Models\Recipe;
use Illuminate\Support\Facades\Hash;

class UserRecipeSeeder extends Seeder
{
    public function run(): void
    {
        // Usuario Nuria
        $nuria = User::firstOrCreate(
            ['email' => 'nuria@test.com'],
            [
                'name' => 'nuria',
                'password' => Hash::make('password'),
            ]
        );

        Recipe::factory()
            ->count(5)
            ->for($nuria)
            ->create();
    }
}

```

```

    // Usuario Pablo
    $pablo = User::firstOrCreate(
        ['email' => 'pablo@test.com'],
        [
            'name' => 'pablo',
            'password' => Hash::make('password'),
        ]
    );

    Recipe::factory()
        ->count(5)
        ->for($pablo)
        ->create();
    }
}

```

Y en el database seeder metemos esto:

```

<?php

namespace Database\Seeders;

use Illuminate\Database\Seeder;

class DatabaseSeeder extends Seeder
{
    public function run(): void
    {
        $this->call([
            UserRecipeSeeder::class,
        ]);
    }
}

```

```

10 references | 0 implementations
3   class Recipe extends Model
4   {
5     use HasFactory;
6   }
7   0 references

```

```
C:\laragon\www\Foodsynk(main -> origin) | en el Database seeder metemos esto:
λ php artisan migrate:fresh --seed

Dropping all tables ..... 74.16ms DONE
INFO Preparing database.

Creating migration table ..... 12.23ms DONE
INFO Running migrations.

0001_01_01_000000_create_users_table ..... 53.84ms DONE
0001_01_01_000001_create_cache_table ..... 14.77ms DONE
0001_01_01_000002_create_jobs_table ..... 52.74ms DONE
2026_01_10_195858_create_ingredients_table ..... 15.72ms DONE
2026_01_10_203303_create_personal_access_tokens_table ..... 29.78ms DONE
2026_01_13_174610_create_recipes_table ..... 7.11ms DONE
2026_01_13_182521_add_user_id_to_recipes_table ..... 34.09ms DONE

INFO Seeding database.

Database\Seeder\UserRecipeSeeder ..... RUNNING
Database\Seeder\UserRecipeSeeder ..... 1,054 ms DONE

C:\laragon\www\Foodsynk(main -> origin)
```

09/02/2026 19:38-

Instalación de Tailwind

Desde la consola, nos vamos a la carpeta de trabajo que tengamos el proyecto y ejecutamos:

```
npm install -D tailwindcss postcss autoprefixer
```

Comando

```
npm install -D tailwindcss postcss autoprefixer
```

Sintaxis y significado

- npm: gestor de paquetes de Node.
- install: instala dependencias en el proyecto actual.
- -D (o --save-dev): guarda los paquetes en devDependencies, es decir, solo se usan en desarrollo/build, no en producción.
- tailwindcss: framework de utilidades CSS.
- postcss: herramienta que procesa CSS con plugins (Tailwind usa PostCSS).
- autoprefixer: plugin de PostCSS que añade prefijos CSS para compatibilidad con navegadores.

Por qué lo usamos

- Tailwind necesita postcss y autoprefixer para generar el CSS final.
- Son herramientas de build, por eso se guardan en devDependencies (-D).

```
C:\Users\Nuria>cd C:\Users\Nuria\Desktop\DAW\segundo\proyecto\proyecto_front\Foodsynk-front
C:\Users\Nuria\Desktop\DAW\segundo\proyecto\proyecto_front\Foodsynk-front>npm install -D tailwindcss postcss autoprefixer
```
added 4 packages, changed 1 package, and audited 189 packages in 3s
42 packages are looking for funding
 run `npm fund` for details
1 high severity vulnerability

To address all issues, run:
 npm audit fix

Run `npm audit` for details.
C:\Users\Nuria\Desktop\DAW\segundo\proyecto\proyecto_front\Foodsynk-front>
```

Al ejecutar `npm install -D tailwindcss postcss autoprefixer` se instalaron los paquetes correctamente, pero `npm` mostró un aviso de “1 high severity vulnerability” y sugirió ejecutar `npm audit fix` para corregirlo; al revisar el detalle vimos que el problema era de `axios`, así que lo actualizamos y luego `npm audit` ya no mostró vulnerabilidades.

```
C:\Users\Nuria\Desktop\DAW\segundo\ proyecto\ proyecto_front\Foodsynk-front>npm install axios@latest
changed 1 package, and audited 189 packages in 1s
42 packages are looking for funding
 run `npm fund` for details
found 0 vulnerabilities

C:\Users\Nuria\Desktop\DAW\segundo\ proyecto\ proyecto_front\Foodsynk-front>
C:\Users\Nuria\Desktop\DAW\segundo\ proyecto\ proyecto_front\Foodsynk-front>npm audit fix
up to date, audited 189 packages in 1s
42 packages are looking for funding
 run `npm fund` for details
found 0 vulnerabilities
```

Y comprobamos

```
C:\Users\Nuria\Desktop\DAW\segundo\ proyecto\ proyecto_front\Foodsynk-front>npm audit
found 0 vulnerabilities
```

Mientras, he tenido problemas con codex, he seguido con las actualizaciones del antigravity.

Para empezar, vamos a crear una rama en git, de la main por si acaso para trabajar sin miedo a romper el código. Para ello primero, guardamos los cambios que tuvieramos y luego ejecutamos en consola lo siguiente:

**git checkout -b Security**

Esto sirve para crear una nueva rama:

```
PS C:\laragon\www\Foodsynk> git checkout -b Security
```

Luego nos movemos a dicha rama con :

**git branch**

```
PS C:\laragon\www\Foodsynk> git branch
* Security
 main
PS C:\laragon\www\Foodsynk>
```

Tras esto le pregunto por el punto de la primera fase: “**Rate Limiting**: Configurar **Throttle** en ruta de login para evitar fuerza bruta (A07).”

Mi prompt fue: en la fase uno, hablame del rate limiting, configurar throttle y dime como hacerlo paso a paso

Esto me genera el siguiente archivo : Implementation Plan

## Plan de Implementación - Configuración de Límite de Velocidad (Rate Limiting)

Este plan describe los pasos para configurar el límite de velocidad (throttling) para los endpoints de la API en la aplicación Laravel.

### Revisión del Usuario Requerida

#### NOTE

Configuraré un límite predeterminado de **60 peticiones por minuto** para la API. Esto se puede ajustar en `App\Providers\AppServiceProvider.php`.

### Cambios Propuestos

#### Configuración

[MODIFICAR] `AppServiceProvider.php`

- Agregar el facade `RateLimiter`.
- Definir un limitador global o específico en el método `boot`.
- Configuración de ejemplo: Limitar a 60 peticiones por minuto por dirección IP.

#### Rutas

[MODIFICAR] `api.php`

- Aplicar el middleware `throttle` a los grupos de rutas relevantes.

### Plan de Verificación

#### Verificación Manual

- Crearé un script simple de PHP o usaré `curl` para enviar > 60 peticiones en un minuto a un endpoint (ej. `/api/login` o una ruta de prueba).
- Verificar que la petición 61 devuelva un código de estado `429 Too Many Requests`.
- Verificar los encabezados `X-RateLimit-Limit` y `X-RateLimit-Remaining`.

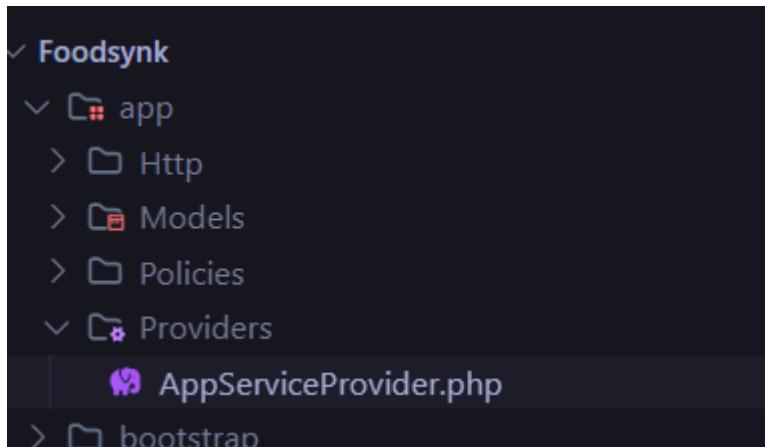
Le pedí que me explicara paso a paso .

Queremos limitar el número de peticiones que un usuario puede hacer a la API en un minuto (por defecto, 60 peticiones). Si superan este límite, recibirán un error 429 Too Many Requests.

El archivo clave es

**app/Providers/AppServiceProvider.php**

Aquí es donde definimos las reglas del limitador.



1. Lo abrimos
2. Importamos las clases necesarias al principio del archivo:

```
use Illuminate\Cache\RateLimiting\Limit;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\RateLimiter;
```

```

App > Providers > AppServiceProvider.php > ...
1 <?php
2
3 namespace App\Providers;
4
5 use Illuminate\Auth\Notifications\ResetPassword;
6 use Illuminate\Cache\RateLimiting\Limit;
7 use Illuminate\Http\Request;
8 use Illuminate\Support\Facades\RateLimiter;
9 use Illuminate\Support\ServiceProvider;
10
11 1 reference | 0 implementations
12 class AppServiceProvider extends ServiceProvider
13 {
14 /**
15 * Register any application services.
16 */
17
18 0 references | 0 overrides
19 public function register(): void
20 {
21 //
22 }
23
24 /**
25 * Bootstrap any application services.
26 */

```

En el método boot(), agregamos la definición del limitador llamado api  
**RateLimiter::for('api', function (Request \$request) {**  
 **return Limit::perMinute(60)->by(\$request->user()?->id ?: \$request->ip());**  
**});**

Limit::perMinute(60): Permite 60 peticiones por minuto.

by(...): Aplica el límite por usuario autenticado (ID) o por dirección IP si no está autenticado.

```

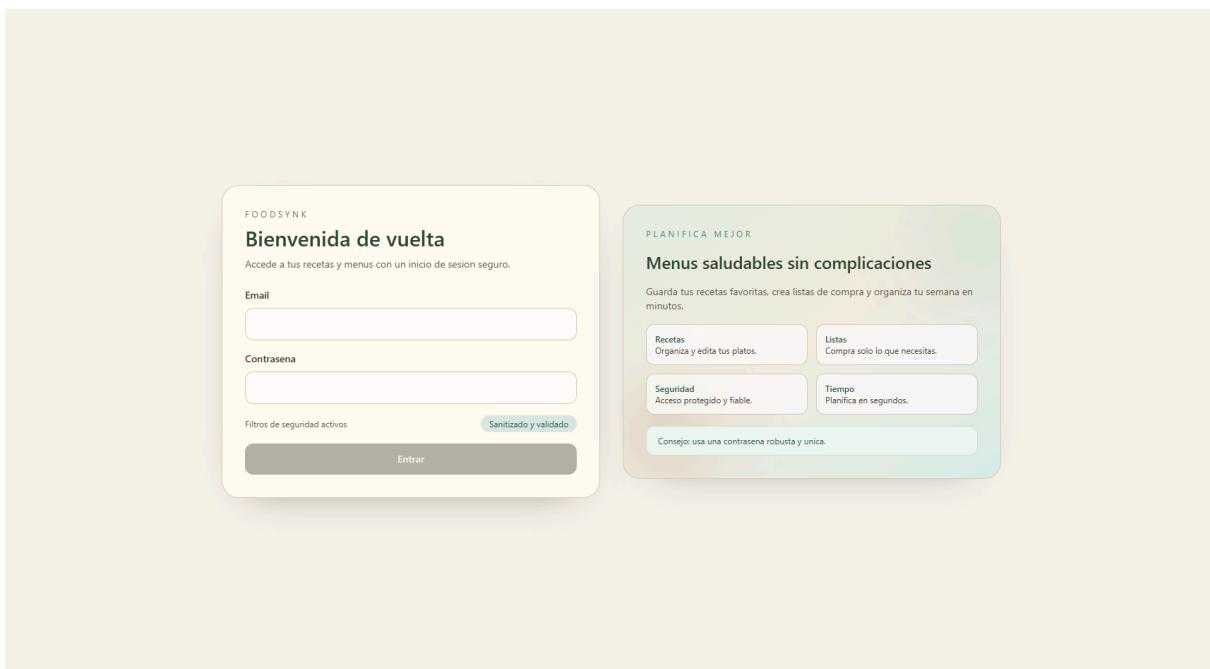
public function boot(): void
{
 RateLimiter::for('api', function (Request $request) {
 return Limit::perMinute(60)->by($request->user()?->id ?: $request->ip());
 });

 ResetPassword::createUrlUsing(function (object $notifiable, string $token) {
 return config('app.frontend_url') . "/password-reset/$token?email={$notifiable->getEmailForPassw";
 });
}

```

En Laravel 11, las rutas definidas en `routes/api.php` ya tienen asignado automáticamente el grupo de middleware `api`. Este grupo incluye por defecto el middleware `throttle:api`.

- Como hemos definido un limitador llamado `api` en `AppServiceProvider`, Laravel usará nuestra configuración (60 peticiones/minuto) para todas las rutas en `routes/api.php`.
- No es necesario editar `routes/api.php` a menos que quieras límites específicos para ciertas rutas.



## Menús saludables sin complicaciones

Guarda tus recetas favoritas, crea listas de compra y organiza tu semana en minutos.

### Recetas

Organiza y edita tus platos.

### Listas

Compra solo lo que necesitas.

### Seguridad

Acceso protegido y fiable.

### Tiempo

Planifica en segundos.

Consejo: usa una contraseña robusta y única.

FOODSYNK

## Bienvenida de vuelta

Accede a tus recetas y menús con un inicio de sesión seguro.

Email

Contraseña

Filtros de seguridad activos

Sanitizado y validado

Entrar

10/02/26

Vamos a terminar de completar la gestión de usuarios.

Como sigo viendo por otro lado lo del security he vuelto a crear una rama de security para asegurarme de trabajar desde los cambios.

```
PS C:\laragon\www\Foodsynk> git checkout -b Registro
Switched to a new branch 'Registro'
PS C:\laragon\www\Foodsynk> git branch
* Registro
 Security
 main
PS C:\laragon\www\Foodsynk>
```

Como bearer no nos funciona, decidimos crear desde Controllers/Api un controlador para el registro.

En este archivo he creado un controlador para gestionar el registro de usuarios desde la API. Lo primero que se hace es definir el namespace App\Http\Controllers\Api, que sirve para indicar que este controlador pertenece a la parte de la API y para que Laravel pueda localizar correctamente la clase dentro de la estructura del proyecto.

Después se importan todas las clases que se van a necesitar. Se usa Controller como clase base de la que va a heredar el controlador. También se importa el modelo User, ya que es el que representa a los usuarios en la base de datos y se utiliza para crear nuevos registros. Se incluye el evento Registered para lanzar el evento de registro de Laravel cuando se crea un usuario. Además, se importa Request para poder acceder a los datos enviados en la petición, Hash para encriptar la contraseña antes de guardarla y Rules para aplicar las reglas de validación de contraseñas por defecto de Laravel.

A continuación se define la clase RegisteredUserController, que extiende de Controller. Esta clase se encarga exclusivamente de manejar el proceso de registro de usuarios mediante la API.

Dentro de la clase se crea el método store, que recibe un objeto Request. Este método se ejecuta cuando se hace una petición de tipo POST para registrar un nuevo usuario. Lo primero que se hace dentro del método es validar los datos recibidos. Se comprueba que el nombre sea obligatorio, de tipo texto y con un máximo de 255 caracteres. El email también es obligatorio, debe estar en formato válido, en minúsculas, no superar los 255 caracteres y no existir previamente en la tabla de usuarios. La contraseña es obligatoria, debe coincidir con su confirmación y cumplir las reglas de seguridad por defecto de Laravel. Por último, el campo device\_name es opcional y solo se valida que sea una cadena de texto.

Una vez que los datos pasan la validación, se crea el usuario en la base de datos utilizando el modelo User. Se guardan el nombre y el email directamente desde la petición y la contraseña se guarda encriptada usando Hash::make, para garantizar la seguridad y evitar almacenar contraseñas en texto plano.

Después de crear el usuario, se lanza el evento Registered, lo que permite que Laravel ejecute acciones automáticas asociadas al registro, como el envío de correos de verificación si están configurados.

A continuación se define el nombre del dispositivo desde el que se crea el token. Si el cliente ha enviado el campo device\_name, se utiliza ese valor; si no, se asigna por defecto el valor api-client.

Por último, se devuelve una respuesta en formato JSON con un código HTTP 201, indicando que el usuario se ha creado correctamente. En la respuesta se incluye un token de autenticación generado con Sanctum, que el usuario utilizará para autenticarse en futuras peticiones a la API, junto con los datos del usuario recién creado.

Al finalizar terminamos con esto:

```
<?php
```

```
namespace App\Http\Controllers\Api;

use App\Http\Controllers\Controller;
use App\Models\User;
use Illuminate\Auth\Events\Registered;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Hash;
use Illuminate\Validation\Rules;

class RegisteredUserController extends Controller
{
 /**
 * Handle an incoming API registration request.
 *
 * @throws \Illuminate\Validation\ValidationException
 */
 public function store(Request $request)
 {
 $request->validate([
 'name' => ['required', 'string', 'max:255'],
 'email' => ['required', 'string', 'lowercase',
'email', 'max:255', 'unique:' . User::class],
 'password' => ['required', 'confirmed',
Rules>Password::defaults()],
 'device_name' => ['nullable', 'string'],
]);

 $user = User::create([
 'name' => $request->name,
 'email' => $request->email,
 'password' =>
Hash::make($request->string('password')),
]);
 }
}
```

```

 event(new Registered($user));

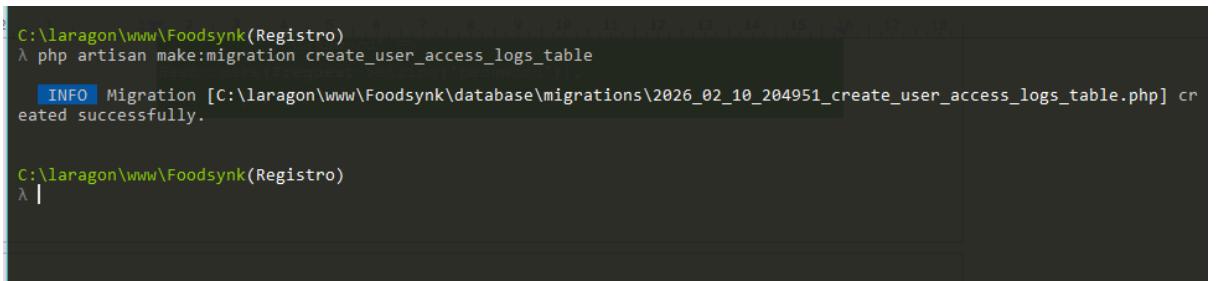
 $deviceName = $request->device_name ?? 'api-client';

 return response()->json([
 'token' =>
 $user->createToken($deviceName)->plainTextToken,
 'user' => $user,
], 201);
 }
}

```

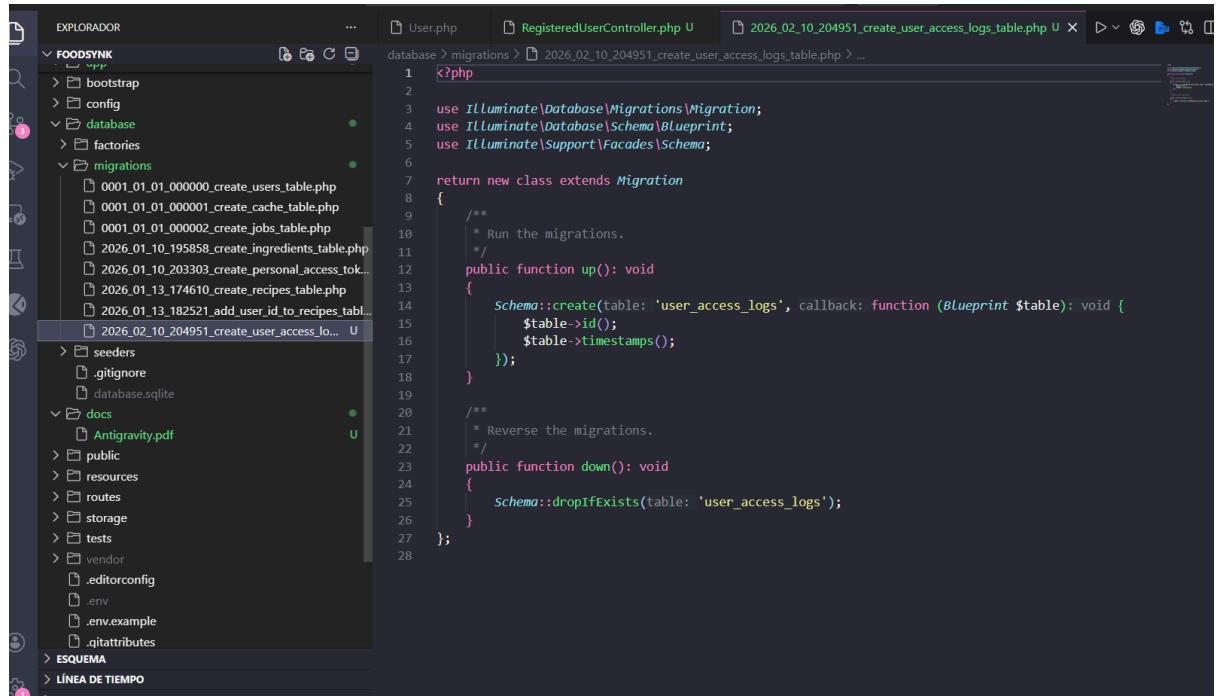
Antes de seguir vamos a crear tabla de auditoria de accesos , para ello vamos a crear la migración con :

```
php artisan make:migration create_user_access_logs_table
```



```
C:\laragon\www\Foodsynk(Registro)
λ php artisan make:migration create_user_access_logs_table
[INFO] Migration [C:\laragon\www\Foodsynk\database\migrations\2026_02_10_204951_create_user_access_logs_table.php] created successfully.
```

Esto crea el siguiente archivo



```

EXPLORADOR User.php RegisteredUserController.php U 2026_02_10_204951_create_user_access_logs_table.php U
FOODSYNK
 <-- app
 > bootstrap
 > config
 > database
 > migrations
 0001_01_01_000000_create_users_table.php
 0001_01_01_000001_create_cache_table.php
 0001_01_01_000002_create_jobs_table.php
 2026_01_10_195858_create_ingredients_table.php
 2026_01_10_203303_create_personal_access_tokens_table.php
 2026_01_13_174610_create_recipes_table.php
 2026_01_13_182521_add_user_id_to_recipes_table.php
 2026_02_10_204951_create_user_access_logs_table.php
 > seeders
 .gitignore
 database.sqlite
 > docs
 Antigravity.pdf
 > public
 > resources
 > routes
 > storage
 > tests
 > vendor
 .editorconfig
 .env
 .env.example
 .gitattributes
 > ESQUEMA
 > LÍNEA DE TIEMPO
 MAGOSSADEV.COM

User.php
class User extends Model
{
 protected $fillable = [
 'name',
 'email',
 'password',
 'device_name',
 'remember_token',
 'created_at',
 'updated_at',
];
}

class Registered implements ShouldQueue
{
 use Dispatchable, InteractsWithSockets, Queueable, Serializable;

 public function handle()
 {
 event(new Registered($user));
 }
}

class RegisteredUserController extends Controller
{
 public function __construct()
 {
 $this->middleware('auth');
 }

 public function index()
 {
 $user = auth();
 $deviceName = $request->device_name ?? 'api-client';

 return response()->json([
 'token' =>
 $user->createToken($deviceName)->plainTextToken,
 'user' => $user,
], 201);
 }
}

class CreateUserAccessLogsTable extends Migration
{
 use Schema;

 /**
 * Run the migrations.
 */
 public function up(): void
 {
 Schema::create('user_access_logs', function (Blueprint $table) {
 $table->id();
 $table->timestamps();
 });
 }

 /**
 * Reverse the migrations.
 */
 public function down(): void
 {
 Schema::dropIfExists('user_access_logs');
 }
}

```

Ahora hay que editar esa migración para añadir los campos de auditoría.

La tabla debe incluir:

- user\_id (nullable, FK a users)
- action (string, ej: login, logout, register)
- success (boolean)
- ip (string, 45 para IPv6)
- user\_agent (string, 512)
- timestamps

Por eso vamos a sustituir el bloque `Schema::create` por esto:

```
/*
public function up(): void
{
 Schema::create('user_access_logs', callback: function (Blueprint $table): void {
 $table->id();
 $table->foreignId(column: 'user_id')->nullable()->constrained()->nullOnDelete();
 $table->string(column: 'action', length: 50);
 $table->boolean(column: 'success')->default(value: true);
 $table->string(column: 'ip', length: 45)->nullable();
 $table->string(column: 'user_agent', length: 512)->nullable();
 $table->timestamps();

 $table->index(columns: ['user_id', 'action']);
 });
}

/**
 * Reverse the migrations.
 */
```

Ahora ejecutamos la migración con : **php artisan migrate**

```
C:\laragon\www\Foodsy whole (Registro)
λ php artisan migrate

INFO Running migrations.
2026_02_10_204951_create_user_access_logs_table 167.84ms DONE
```

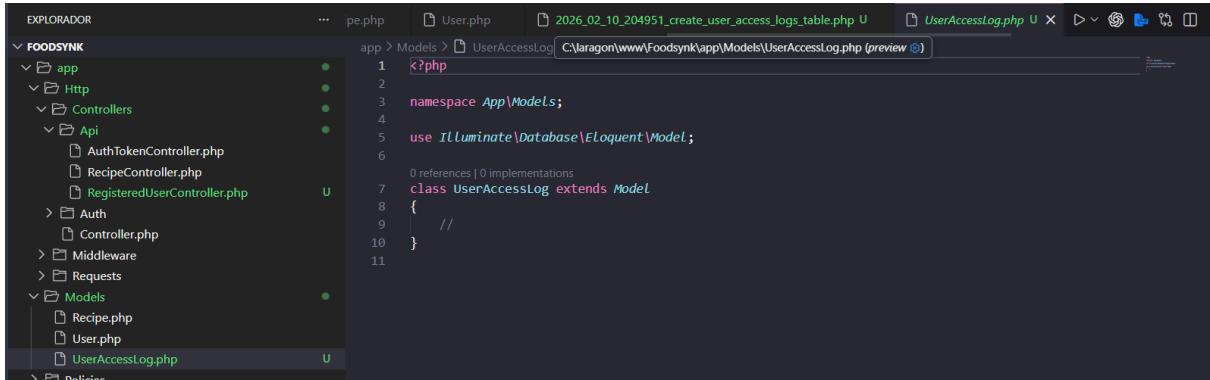
Esto creó la tabla `user_access_logs`.

Ahora crearemos el modelo `UserAccessLog`.

Necesitamos ejecutar primero **php artisan make:model UserAccessLog**

```
C:\laragon\www\Foodsy whole (Registro)
λ php artisan make:model UserAccessLog
INFO Model [C:\laragon\www\Foodsy whole\app\Models\UserAccessLog.php] created successfully.
```

Esto nos debe crear el archivo UserAccessLog.php



The screenshot shows a code editor with a sidebar labeled 'EXPLORADOR' containing a tree view of a Laravel application structure. The 'Models' folder is expanded, showing 'User.php', 'UserAccessLog.php', and '2026\_02\_10\_204951\_create\_user\_access\_logs\_table.php'. The main editor area displays the contents of 'UserAccessLog.php':

```
<?php
namespace App\Models;
use Illuminate\Database\Eloquent\Model;

class UserAccessLog extends Model
{
 //
}
```

Una vez creado el modelo, se modificó su contenido para adaptarlo a las necesidades del proyecto. La clase UserAccessLog extiende de Model, lo que le da acceso a todas las funcionalidades de Eloquent. También se utiliza el trait HasFactory, que permite crear registros de prueba mediante factories si fuese necesario más adelante.

Se añadió la propiedad `$fillable` para definir qué campos pueden ser asignados de forma masiva. Esto es importante porque Laravel, por seguridad, no permite guardar datos en masa si no se especifica explícitamente qué campos están permitidos. En este caso se han incluido los campos `user_id`, `action`, `success`, `ip` y `user_agent`, que son los datos que se quieren guardar cada vez que se registre una acción o acceso de un usuario.

Además, se añadió el método `user()`, que define una relación de tipo `belongsTo` con el modelo User. Esto indica que cada registro de acceso pertenece a un único usuario, identificado por el campo `user_id`. Gracias a esta relación, se puede acceder fácilmente al usuario asociado a un registro de acceso usando Eloquent, lo que facilita consultas y el manejo de los datos.

Este modelo se ha configurado así para poder registrar de forma estructurada la actividad de los usuarios y relacionarla directamente con la tabla de usuarios, manteniendo una base de datos más organizada y preparada para futuras funcionalidades relacionadas con seguridad o control de accesos.

Tenemos que crear la relación en user, así que nos vamos a **User.php**

Al final añadimos la siguiente función:

```
public function accessLogs(): HasMany
{
 return $this->hasMany(related: \App\Models\UserAccessLog::class);
}
```

Este código define la relación entre el modelo User y los registros de acceso. El método `public function accessLogs()` crea un método público dentro del modelo User, que es la forma que utiliza Laravel para definir relaciones entre modelos y poder acceder a ellas de manera sencilla.

Dentro del método se devuelve `$this->hasMany(...)`, lo que indica que un usuario puede tener muchos registros de acceso asociados. Es decir, un mismo usuario puede generar múltiples entradas en la tabla de logs a lo largo del tiempo.

El modelo relacionado es `\App\Models\UserAccessLog::class`. Al definir esta relación, Laravel asume automáticamente que la clave foránea utilizada en la tabla `user_access_logs` es `user_id`, que es justo el campo que se creó previamente en el modelo y en la base de datos.

Gracias a esta relación, ahora es posible obtener todos los registros de acceso de un usuario utilizando `$user->accessLogs`. También permite crear nuevos registros de forma directa y vinculada al usuario mediante `$user->accessLogs()->create([...])`, lo que simplifica mucho el registro de la actividad del usuario en la aplicación.

Ahora vamos a cambiar AuthTokenController para configurar los accesos sin revelar información sensible .

En este archivo hacemos los siguientes cambios:

```
^ 8 unmodified lines
 9 use Illuminate\Support\Facades\Hash;
 10 use Illuminate\Validation\ValidationException;
 11 use App\Models\User;
12 use App\Models\UserAccessLog;
 13
 14
 15 class AuthTokenController extends Controller
◊ 14 unmodified lines
 30
 31 // 3 Comprobar que el usuario existe y la contraseña es correcta
 32 if (! $user || ! Hash::check($request->password, $user->password)) {
 33 UserAccessLog::create([
 34 'user_id' => $user?>id,
 35 'action' => 'login',
 36 'success' => false,
 37 'ip' => $request->ip(),
 38 'user_agent' => $request-> userAgent(),
 39]);
 40 throw ValidationException::withMessages([
 41 'email' => ['Credenciales incorrectas.'],
 42]);
◊ 5 unmodified lines
 48
 49
```

Cambios exactos:

- Se importa UserAccessLog.
- En login fallido: se crea log con success = false.
- En login exitoso: se crea log con success = true y se reutiliza \$token.
- En logout: se crea log con action = 'logout'.

Ahora toca registrar el acceso en el registro API (Api\RegisteredUserController)

Por eso tenemos que importar UserAccessLog, y luego crear el usuario y token, guardando un log.

El archivo quedaría así:

```
<?php

namespace App\Http\Controllers\Api;

use App\Http\Controllers\Controller;
use App\Models\User;
use App\Models\UserAccessLog;
use Illuminate\Auth\Events\Registered;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Hash;
use Illuminate\Validation\Rules;

class RegisteredUserController extends Controller
{
 /**
 * Handle an incoming API registration request.
 *
 * @throws \Illuminate\Validation\ValidationException
 */
 public function store(Request $request)
 {
 $request->validate([
 'name' => ['required', 'string', 'max:255'],
 'email' => ['required', 'string', 'lowercase', 'email',
'max:255', 'unique:' . User::class],
 'password' => ['required', 'confirmed',
Rules\Password::defaults()],
 'device_name' => ['nullable', 'string'],
]);
 }
}
```

```
$user = User::create([
 'name' => $request->name,
 'email' => $request->email,
 'password' => Hash::make($request->string('password')) ,
]) ;

event(new Registered($user));

$deviceName = $request->device_name ?? 'api-client';

$token = $user->createToken($deviceName)->plainTextToken;

UserAccessLog::create([
 'user_id' => $user->id,
 'action' => 'register',
 'success' => true,
 'ip' => $request->ip(),
 'user_agent' => $request->userAgent(),
]) ;

return response()->json([
 'token' => $token,
 'user' => $user,
], 201);
}

}
```

16/02/26

Nos dirigimos a routes/api.php e importamos RegisteredUserController para poder usarlo en una nueva ruta POST /api/register y registrar usuarios desde la API (sin pasar por el flujo web).

Para ello añadimos:

use App\Http\Controllers\Api\RegisteredUserController;

Ahora vamos a endurecer el login y añadir el register con throttle estricto para mitigar la fuerza bruta y abuso según OWASP A07.

Para eso sustituimos la ruta actual de login y añadimos register así:

```
Route::middleware(['throttle:5,1'])->post('/login', [AuthUserController::class, 'login']);
Route::middleware(['throttle:5,1'])->post('/register', [RegisteredUserController::class, 'store']);
```

```
7 use App\Http\Controllers\Api\AuthTokenController;
8 use App\Http\Controllers\Api\RegisteredUserController;
9
10 Route::middleware(['throttle:5,1'])->post('/login', action: [AuthTokenController::class]);
11 Route::middleware(['throttle:5,1'])->post('/register', action: [RegisteredUserController::class]);
```

Por último vamos a verificar las rutas para confirmar que Laravel registró endpoints y middleware.

Nos vamos a la terminal de laragon y ejecutamos

```
C:\laragon\www\fodsync\main> php artisan route:list --path=api
+----+----+-----+
| POST | api/login | ApiAuthTokenController@login
+----+----+-----+
| POST | api/logout | ApiAuthTokenController@logout
+----+----+-----+
| GET|HEAD| api/me | ApiUserController@me
+----+----+-----+
| GET|HEAD| api/recipes | ApiRecipeController@index
+----+----+-----+
| POST | api/recipes | ApiRecipeController@store
+----+----+-----+
| POST | api/recipes/batch | ApiRecipeController@batchstore
+----+----+-----+
| PATCH|PUT| api/recipes/batch | ApiRecipeController@batchstore
+----+----+-----+
| DELETE| api/recipes/batch | ApiRecipeController@batchdestroy
+----+----+-----+
| POST | api/recipes/search | ApiRecipeController@search
+----+----+-----+
| DELETE|PUT| api/recipes/{recipe} | ApiRecipeController@update
+----+----+-----+
| DELETE| api/recipes/{recipe} | ApiRecipeController@destroy
+----+----+-----+
| POST | api/register | ApiRegisterUserController@register
+----+----+-----+

```

Vamos a probar que funciona en postman.

The screenshot shows the Postman interface with a successful API call. The left sidebar lists collections like 'users\_foodsynk' and 'usuarios'. The main area shows a 'New HTTP Request' with a 'POST' method to 'http://localhost/Foodsynk/public/api/register'. The 'Body' tab is selected, showing a raw JSON payload:

```
1 {
2 "name": "Paul",
3 "email": "paul_test@example.com",
4 "password": "Password123!",
5 "password_confirmation": "Password123!",
6 "device_name": "postman"
7 }
8
```

The response status is 201 Created, with a timestamp of 650 ms and a size of 644 B. The response body is:

```
1 [{"token": "2|kV0Bq3EmrZCDIhrUH8MbsXsbakCvvLKciR9IaVcc61c25a7", "user": {"name": "Paul", "email": "paul_test@example.com", "updated_at": "2026-02-16T22:22:58.000000Z", "created_at": "2026-02-16T22:22:58.000000Z", "id": 3}}]
```

Hicimos varias pruebas: registro, login, logout, me y el rate limit.

Todas salieron correctamente y por último, hemos comprobado user\_access\_log

```
C:\laragon\www\FoodSync(main -> origin)
λ php artisan tinker
Psy Shell v0.12.18 (PHP 8.3.30 - cli) by Justin Hileman
New PHP manual is available (latest: 3.0.1). Update with `doc --update-manual`
> \App\Models\UserAccessLog::latest()->take(20)->get();
= Illuminate\Database\Eloquent\Collection {#6543
 all: [as que añades al documento
 App\Models\UserAccessLog {#6522
 id: 8,
 user_id: 3,
 action: "login",
 success: 0,
 ip: "127.0.0.1",
 user_agent: "PostmanRuntime/7.39.1",
 created_at: "2026-02-16 22:45:12",
 updated_at: "2026-02-16 22:45:12",
 },
 App\Models\UserAccessLog {#6521
 id: 7,
 user_id: 3,
 action: "login",
 success: 0,
 ip: "127.0.0.1",
 user_agent: "PostmanRuntime/7.39.1",
 created_at: "2026-02-16 22:45:11",
 updated_at: "2026-02-16 22:45:11",
 },
 App\Models\UserAccessLog {#6520
 id: 6,
 user_id: 3,
 action: "login",
 success: 0,
 ip: "127.0.0.1",
 user_agent: "PostmanRuntime/7.39.1",
 created_at: "2026-02-16 22:45:10",
 updated_at: "2026-02-16 22:45:10",
 },
 App\Models\UserAccessLog {#6519
```

middleware.  
Nos vamos a la terminal

Vamos a probar que func

**18/02/26**