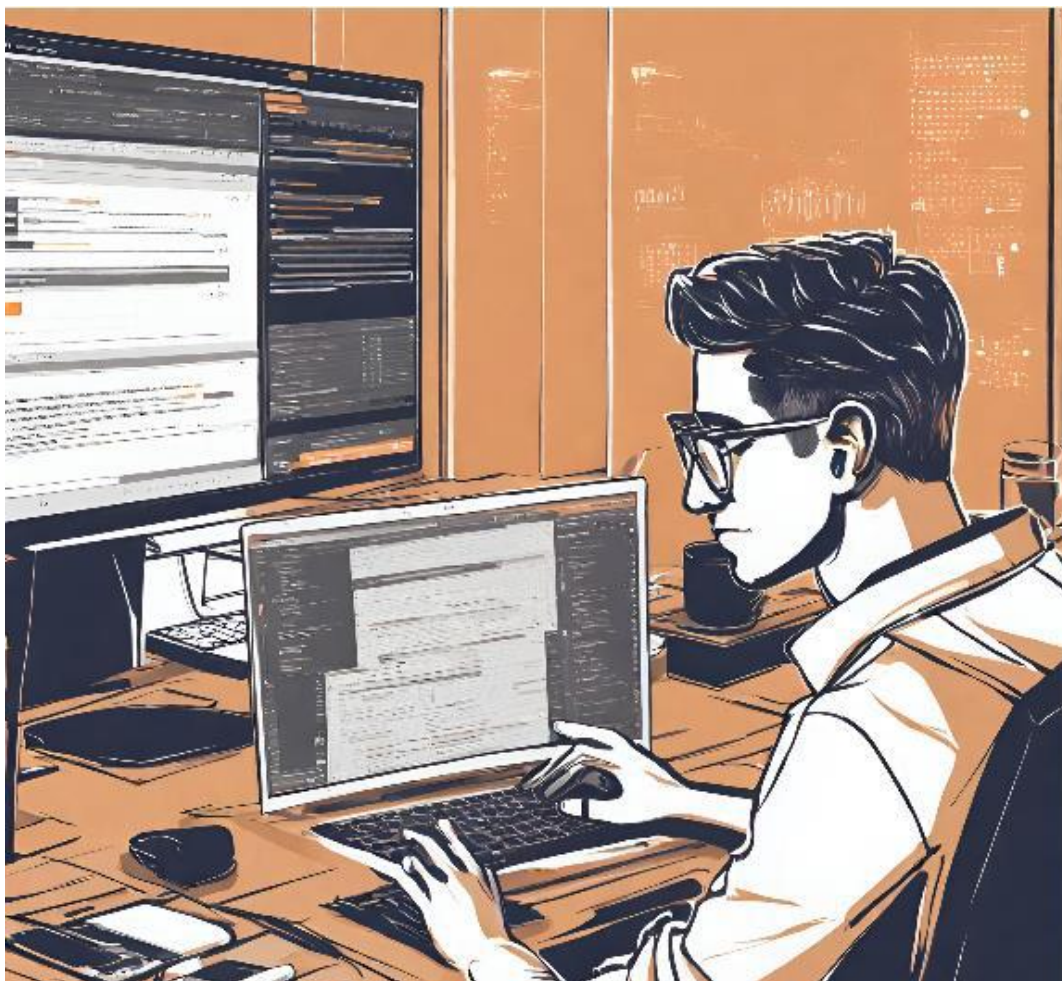


# Laravel

*Instalación de un servidor Laravel y utilización de Api Rest*

*Fco Javier Gallego Fernández*



## Índice

1. Creación del primer proyecto
2. Creación de la tabla tareas
  - a. Introducción de datos en la tabla
3. Creación del modelo Tarea
  - a. Maquetación TareaController
  - b. Creación TareaRequest
  - c. Creación TareaResource
4. Creación de la tabla etiquetas
  - a. Introducción de datos en la tabla
5. Creación del modelo Etiqueta
  - a. Maquetación EtiquetaController
  - b. Creación EtiquetaRequest
  - c. Creación EtiquetaResource
6. Relacional entre Tareas y Etiquetas
7. Route de la Api
8. Comprobando con Postman
9. Creación Login API REST
  - a. Controlador AuthController
  - b. Route api AuthController
  - c. Comprobando Auth con Postman
10. Implementación de test

## Creación del primer proyecto

Hacemos nuestro primer proyecto en laravel, para ello vamos a utilizar el siguiente comando:

**composer create-project --prefer-dist laravel/laravel apitareas**

```
83 packages you are using are looking for funding.
Use the 'composer fund' command to find out more!
> @php artisan vendor:publish --tag=laravel-assets --ansi --force
INFO: No publishable resources for tag [laravel-assets].
> @php artisan key:generate --ansi
INFO: Application key set successfully.
javi@Javi:/mnt/c/Users/Javier/Desktop/Estudios/2ºDAW/DWESE/Docker/www$ composer create-project --prefer-dist laravel/laravel apitareas
```

Si no tenemos el composer instalado deberemos ejecutar el comando “**composer install**”

Ahora nos debemos mover a nuestro proyecto con el comando ‘**cd apitareas**’

```
javi@Javi:/$ cd mnt/c/Users/Javier/Desktop/Estudios/2ºDAW/DWESE/Docker/www/apitareas/
javi@Javi:/mnt/c/Users/Javier/Desktop/Estudios/2ºDAW/DWESE/Docker/www/apitareas$ |
```

El siguiente paso que tendremos que hacer es instalar Laravel Sail utilizando Artisan con el comando:

**‘php artisan sail:install’**

Al introducir el comando nos preguntara con qué tipo de base de datos queremos trabajar en nuestro proyecto, en nuestro caso utilizaremos mysql.

```
javi@Javi:/mnt/c/Users/Javier/Desktop/Estudios/2ºDAW/DWESE/Docker/www/apitareas$ php artisan sail:install
Which services would you like to install?
> ■ mysql
  □ pgsql
  □ mariadb
  □ redis
  □ memcached
1 selected
```

Este proceso entre otras cosas nos va a generar el Docker-compose.yml con el que podremos abrir el contenedor posteriormente.

Cuando obtengamos el siguiente mensaje quiere decir que todo ha ido bien y está listo para lanzar.

```
Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to fix them
INFO: Sail scaffolding installed successfully. You may run your Docker containers using Sail's "up" command.
-> ./vendor/bin/sail up
WARN: A database service was installed. Run "artisan migrate" to prepare your database:
-> ./vendor/bin/sail artisan migrate
javi@Javi:/mnt/c/Users/Javier/Desktop/Estudios/2ºDAW/DWESE/Docker/www/apitareas$ |
```

El siguiente paso es levantar el contenedor Docker, para ello tendremos que utilizar el comando ‘**./vendor/bin/sail up -d**’ pero como sail es algo que vamos a tener que estar utilizando le vamos a crear un alias (El cual según distribución puede ser temporal y deberemos de hacer cada vez que arranquemos nuestra máquina Linux) con el siguiente comando:

**‘alias sail=./vendor/bin/sail’**

```
javi@Javi:/mnt/c/Users/Javier/Desktop/Estudios/2ºDAW/DWESE/Docker/www/apitareas$ alias sail=./vendor/bin/sail
javi@Javi:/mnt/c/Users/Javier/Desktop/Estudios/2ºDAW/DWESE/Docker/www/apitareas$
```

Con el alias conseguimos poder escribir 'sail' en lugar de './vendor/bin/sail'.

Ahora sí, vamos a levantar el contenedor con el comando 'sail up -d', recuerda tener antes abierto el Docker desktop abierto:

```
javi@Javi:/mnt/c/Users/Javier/Desktop/Estudios/2ºDAW/DWESE/Docker/www/apitareas$ sail up -d
[+] Running 4/4
 # Network apitareas_sail          Created
 # Volume "apitareas_sail-mysql"   Created
 # Container apitareas-mysql-1     Started
 # Container apitareas-laravel.test-1 Started
javi@Javi:/mnt/c/Users/Javier/Desktop/Estudios/2ºDAW/DWESE/Docker/www/apitareas$
```

Si vamos a nuestro Docker podemos comprobar que el contenedor está funcionando:

|                                     |                                |                        |               |                    |                |  |
|-------------------------------------|--------------------------------|------------------------|---------------|--------------------|----------------|--|
| <input checked="" type="checkbox"/> | apitareas                      | -                      | Running (2/2) |                    |                |  |
| <input type="checkbox"/>            | mysql-1<br>cb378aea99b0        | mysql/mysql-server:8.0 | Running       | 3306:3306          | 40 seconds ago |  |
| <input type="checkbox"/>            | laravel.test-1<br>beabf73bf9f5 | sail-8.3/app:latest    | Running       | 5173:5173<br>80:80 | 38 seconds ago |  |

## Creación de la tabla tareas

Ahora vamos a crear la tabla tareas, para ello deberemos ejecutar el siguiente comando:  
'sail artisan make:migration create\_tareas\_table'

```
javi@Javi:/mnt/c/Users/Javier/Desktop/Estudios/2ºDAW/DWESE/Docker/www/apitareas$ sail artisan make:migration create_tareas_table
INFO Migration [database/migrations/2024_02_14_122254_create_tareas_table.php] created successfully.
```

Esto nos crea un archivo en nuestro proyecto **apitareas->database->migrations->%fecha%create\_tareas\_table.php**

Aquí tenemos que definir la estructura de nuestra tabla, en nuestro caso vamos a definir que tenga un id, titulo y descripción.

```
2024_02_14_122254_create_tareas_table.php U
database > migrations > 2024_02_14_122254_create_tareas_table.php
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 {
9     /**
10      * Run the migrations.
11      */
12     public function up(): void
13     {
14         Schema::create('tareas', function (Blueprint $table) {
15             $table->id();
16             $table->string('nombre', 40);
17             $table->string('descripcion')->nullable();
18             $table->timestamps();
19         });
20     }
21
22     /**
23      * Reverse the migrations.
24      */
25     public function down(): void
26     {
27         Schema::dropIfExists('tareas');
28     }
29 }
```

Para crear la tabla deberemos ejecutar el comando **'sail artisan migrate'**

```
javi@Javi:/mnt/c/Users/Javier/Desktop/Estudios/2ºDAW/DWESE/Docker/www/apitareas$ sail artisan migrate
[INFO] Preparing database.
Creating migration table ..... 131ms DONE
[INFO] Running migrations.
2014_10_12_000000_create_users_table ..... 130ms DONE
2014_10_12_100000_create_password_reset_tokens_table ..... 41ms DONE
2019_08_19_000000_create_failed_jobs_table ..... 135ms DONE
2019_12_14_000001_create_personal_access_tokens_table ..... 149ms DONE
2024_02_14_122254_create_tareas_table ..... 50ms DONE
```

## Introducción de datos en la tabla

Ahora vamos a introducir los datos de nuestra tabla, para ello debemos crear el seeder con el siguiente comando:

**'sail artisan make:seeder TareaSeeder'**

```
javi@Javi:/mnt/c/Users/Javier/Desktop/Estudios/2ºDAW/DWESE/Docker/www/apitareas$ sail artisan make:seeder TareaSeeder
[INFO] Seeder [database/seeder/TareaSeeder.php] created successfully.
```

Como bien nos dice el mensaje que devuelve esto nos crea el archivo en nuestro proyecto en **database/seeder/TareaSeeder.php**

En dicho archivo deberemos importar el archivo DB para ello al inicio incluimos lo siguiente:

**'use Illuminate\Support\Facades\DB;'**

En el archivo vamos a incluir las rows que queramos en nuestra tabla, el archivo debería quedar de la siguiente manera:

```
TareaSeeder.php U X
database > seeders > TareaSeeder.php > Database\Seeders\TareaSeeder
1  <?php
2
3  namespace Database\Seeders;
4
5  use Illuminate\Support\Facades\DB;
6  use Illuminate\Database\Console\Seeds\WithoutModelEvents;
7  use Illuminate\Database\Seeder;
8
9  1 reference | 0 implementations
10 class TareaSeeder extends Seeder
11 {
12     /**
13      * Run the database seeds.
14      */
15     0 references | 0 overrides
16     public function run(): void
17     {
18         DB::table('tareas')->insert([
19             'nombre'=>'Hacer proyecto',
20             'descripcion'=>'hay que terminar el proyecto',
21         ]);
22         DB::table('tareas')->insert([
23             'nombre'=>'Entregar',
24             'descripcion'=>'Hay que entregar el proyecto',
25         ]);
26         DB::table('tareas')->insert([
27             'nombre'=>'Descansar',
28             'descripcion'=>'Tomate un descansooo',
29         ]);
30         DB::table('tareas')->insert([
31             'nombre'=>'OotraTarea',
32             'descripcion'=>'Tomate un descansooo',
33         ]);
34     }
35 }
```

Por último en DatabaseSeeder.php debemos añadir '\$this->call([TareaSeeder::class]);'

```
DatabaseSeeder.php M X
database > seeders > DatabaseSeeder.php > ...
1  <?php
2
3  namespace Database\Seeders;
4
5  // use Illuminate\Database\Console\Seeds\WithoutModelEvents;
6  use Illuminate\Database\Seeder;
7
8  0 references | 0 implementations
9  class DatabaseSeeder extends Seeder
10 {
11     /**
12      * Seed the application's database.
13      */
14     0 references | 0 overrides
15     public function run(): void
16     {
17         $this->call([TareaSeeder::class]);
18     }
19 }
```

Para hacer los inserts en nuestra DB que hemos incluido en nuestro archivo deberemos ejecutar el comando 'sail artisan db:seed'

```
javi@Javi:/mnt/c/Users/Javier/Desktop/Estudios/2ºDAW/DWEESE/Docker/www/apitareas$ sail artisan db:seed
INFO Seeding database.
```

## Creación del modelo Tarea

Ahora vamos a crear el modelo de tareas, para ello vamos a utilizar el comando

'sail artisan make:model Tarea -cr'

```
javi@Javi:/mnt/c/Users/Javier/Desktop/Estudios/2ºDAW/DWEESE/Docker/www/apitareas$ sail artisan make:model Tarea -cr
INFO Model [app/Models/Tarea.php] created successfully.
INFO Controller [app/Http/Controllers/TareaController.php] created successfully.
```

Con esto creamos el modelo Tarea.php y el controller.

Vamos a empezar a maquetar el modelo Tarea.php. En nuestro caso simplemente le diremos qué puede modificar y los ocultos que se van a rellenar solo.

```
Tareas.php U X
app > Models > Tareas.php > ...
1  <?php
2
3  namespace App\Models;
4
5  use Illuminate\Database\Eloquent\Factories\HasFactory;
6  use Illuminate\Database\Eloquent\Model;
7
8  5 references | 0 implementations
9  class Tareas extends Model
10 {
11     use HasFactory;
12     // No deja modificar todos los que estén dentro de corchetes, si esta vacío deja modificar todo
13     0 references
14     protected $guarded = [];
15
16     0 references
17     protected $hidden = ['created_at', 'updated_at'];
18 }
```

Antes de modificar el controller, vamos a crear el request, el resource y la tabla etiquetas para no tener que volver a modificar el controller.

En el caso que queramos comprobar que todo va correcto, podemos hacerlo añadiendo lo siguiente en TareaController.php

```
0 references | 0 overrides
public function index()
{
    $tareas = Tarea::all();

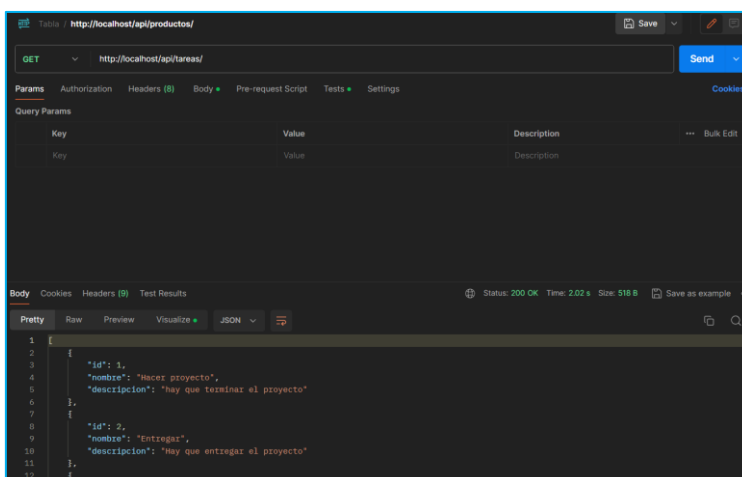
    return response()->json($tareas, 200); //Con el 200 controlamos el OK
}
```

Y completar nuestra api.php añadiendo la siguiente línea:

**'Route::resource('/tareas', TareaController::class);'**

```
api.php M X
routes > api.php
1 <?php
2
3 use App\Http\Controllers\TareaController;
4 use Illuminate\Http\Request;
5 use Illuminate\Support\Facades\Route;
6
7 /*
8 |-----
9 | API Routes
10 |-----
11 |
12 | Here is where you can register API routes for your application. These
13 | routes are loaded by the RouteServiceProvider and all of them will
14 | be assigned to the "api" middleware group. Make something great!
15 |
16 */
17
18 Route::middleware('auth:sanctum')->get('/user', function (Request $request) {
19     return $request->user();
20 });
21
22 Route::resource('/tareas', TareaController::class);
```

Si accedemos al postman con el método get en localhost/api/tareas comprobamos que nos devuelve correctamente los datos.



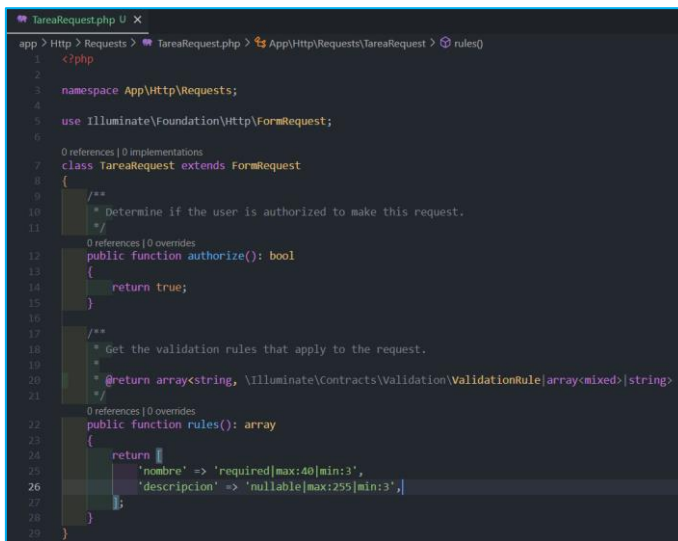
## Creación TareaRequest

Vamos a crear nuestro request, para ello vamos a utilizar el comando **'sail artisan make:request TareaRequest'**

```
javi@Javi:/mnt/c/Users/Javier/Desktop/Estudios/2ºDAW/DWESE/Docker/www/apitareas$ sail artisan make:request TareaRequest
INFO Request [app/Http/Requests/TareaRequest.php] created successfully.
```

Esto nos crea las requests en **TareaRequest.php**.

En dicho requests le debemos poner en true la función authorize y añadimos las reglas que queremos que deba tener los request



```

1 <?php
2
3 namespace App\Http\Requests;
4
5 use Illuminate\Foundation\Http\FormRequest;
6
7 class TareaRequest extends FormRequest
8 {
9     /**
10      * Determine if the user is authorized to make this request.
11      */
12     public function authorize(): bool
13     {
14         return true;
15     }
16
17     /**
18      * Get the validation rules that apply to the request.
19      *
20      * @return array<string, \Illuminate\Contracts\Validation\ValidationRule|array<mixed>|string>
21      */
22     public function rules(): array
23     {
24         return [
25             'nombre' => 'required|max:40|min:3',
26             'descripcion' => 'nullable|max:255|min:3',
27         ];
28     }
29 }

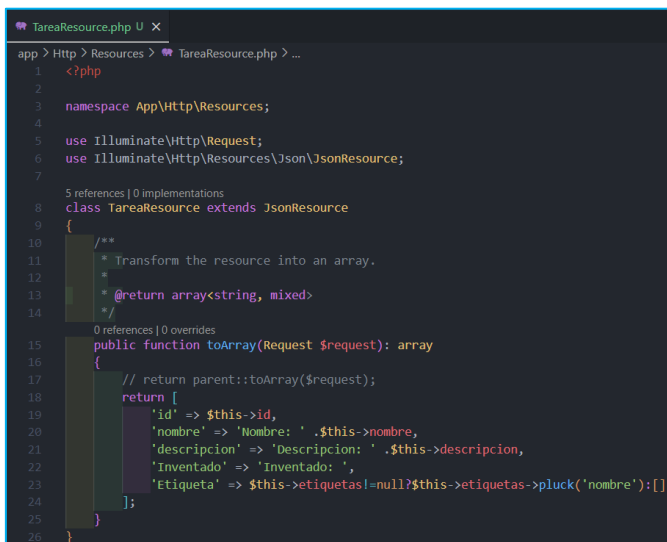
```

## Creación TareaResource

Ahora vamos a crear nuestro resource, para ello utilizamos el comando **'sail artisan make:resource TareaResource'**

```
javi@Javi:/mnt/c/Users/Javier/Desktop/Estudios/2ºDAW/DWESE/Docker/www/apitareas$ sail artisan make:resource TareaResource
```

Esto nos crea el archivo TareaResource. En dicho archivo vamos a decirle la estructura que queremos.



```

1 <?php
2
3 namespace App\Http\Resources;
4
5 use Illuminate\Http\Request;
6 use Illuminate\Http\Resources\Json\JsonResource;
7
8 class TareaResource extends JsonResource
9 {
10     /**
11      * Transform the resource into an array.
12      *
13      * @return array<string, mixed>
14      */
15     public function toArray(Request $request): array
16     {
17         // return parent::toArray($request);
18         return [
19             'id' => $this->id,
20             'nombre' => 'Nombre: ' . $this->nombre,
21             'descripcion' => 'Descripcion: ' . $this->descripcion,
22             'inventado' => 'Inventado: ',
23             'etiqueta' => $this->etiquetas != null ? $this->etiquetas->pluck('nombre') : []
24         ];
25     }
26 }

```



## Creación de la tabla Etiquetas

Ahora vamos a crear la tabla etiquetas, para ello vamos a utilizar el comando **'sail artisan make:migration create\_etiquetas\_table'**

```
javi@Javi:/mnt/c/Users/Javier/Desktop/Estudios/2ºDAW/DWESE/Docker/www/apitareas$ sail artisan make:migration create_etiquetas_table
INFO Migration [database/migrations/2024_02_19_155801_create_etiquetas_table.php] created successfully.
javi@Javi:/mnt/c/Users/Javier/Desktop/Estudios/2ºDAW/DWESE/Docker/www/apitareas$
```

Y ahora maquetamos la estructura de la tabla:

```
2024_02_19_155801_create_etiquetas_table.php U X
database > migrations > 2024_02_19_155801_create_etiquetas_table.php
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  {
9      /**
10       * Run the migrations.
11       */
12       public function up(): void
13       {
14           Schema::create('etiquetas', function (Blueprint $table) {
15               $table->id();
16               $table->string('nombre', 15);
17               $table->timestamps();
18           });
19       }
20
21       /**
22        * Reverse the migrations.
23        */
24       public function down(): void
25       {
26           Schema::dropIfExists('etiquetas');
27       }
28     };
```

Y para crear las tablas usaremos el comando **'sail artisan migrate'**

```
javi@Javi:/mnt/c/Users/Javier/Desktop/Estudios/2ºDAW/DWESE/Docker/www/apitareas$ sail artisan migrate
INFO Running migrations.
2024_02_19_155801_create_etiquetas_table ..... 84ms DONE
```

## Introducción de datos en la tabla Etiquetas

Ahora vamos a introducir los datos de nuestra tabla, para ello debemos crear el seeder con el siguiente comando:

**'sail artisan make:seeder EtiquetaSeeder'**

```
javi@Javi:/mnt/c/Users/Javier/Desktop/Estudios/2ºDAW/DWESE/Docker/www/apitareas$ sail artisan make:seeder EtiquetaSeeder
INFO Seeder [database/seeds/EtiquetaSeeder.php] created successfully.
```

Como bien nos dice el mensaje que devuelte esto nos crea el archivo en nuestro proyecto en **database/seeds/EtiquetaSeeder.php**

En dicho archivo deberemos importar el archivo DB para ello al inicio incluimos el lo siguiente:

**'use Illuminate\Support\Facades\DB;'**

En el archivo vamos a incluir las rows que queramos en nuestra tabla, el archivo debería quedar de la siguiente manera:

```
EtiquetaSeeder.php U X
database > seeders > EtiquetaSeeder.php > ...
1  <?php
2
3  namespace Database\Seeders;
4
5  use Illuminate\Support\Facades\DB;
6  use Illuminate\Database\Console\Seeds\WithoutModelEvents;
7  use Illuminate\Database\Seeder;
8
9  0 references | 0 implementations
10 class EtiquetaSeeder extends Seeder
11 {
12     /**
13      * Run the database seeds.
14      */
15     0 references | 0 overrides
16     public function run(): void
17     {
18         DB::table('etiquetas')->insert([
19             'nombre'=>'Clase',
20         ]);
21         DB::table('etiquetas')->insert([
22             'nombre'=>'Otros',
23         ]);
24     }
25 }
```

Por último en DatabaseSeeder.php debemos añadir **'EtiquetaSeeder::class'** de manera que nos quede **'\$this->call([TareaSeeder::class,EtiquetaSeeder::class]);'**

```
DatabaseSeeder.php M X
database > seeders > DatabaseSeeder.php > Database\Seeders\DatabaseSeeder
1  <?php
2
3  namespace Database\Seeders;
4
5  // use Illuminate\Database\Console\Seeds\WithoutModelEvents;
6  use Illuminate\Database\Seeder;
7
8  0 references | 0 implementations
9  class DatabaseSeeder extends Seeder
10 {
11     /**
12      * Seed the application's database.
13      */
14     0 references | 0 overrides
15     public function run(): void
16     {
17         $this->call([TareaSeeder::class,EtiquetaSeeder::class]);
18     }
19 }
```

Para hacer los inserts en nuestra DB que hemos incluido en nuestro archivo deberemos ejecutar el comando **'sail artisan db:seed'**

```
javi@Javi:/mnt/c/Users/Javier/Desktop/Estudios/2ºDAW/DWESE/Docker/www/apitareas$ sail artisan db:seed
INFO Seeding database.
```

## Creación del modelo Etiqueta

Ahora vamos a crear el modelo de etiquetas, para ello vamos a utilizar el comando

**'sail artisan make:model Etiqueta -cr'**

```
javi@Javi:/mnt/c/Users/Javier/Desktop/Estudios/2ºDAW/DWSE/Docker/www/apitareas$ sail artisan make:model Etiqueta -cr
[INFO] Model [app/Models/Etiqueta.php] created successfully.
[INFO] Controller [app/Http/Controllers/EtiquetaController.php] created successfully.
```

Con esto creamos el modelo Etiqueta.php y el controller.

Vamos a empezar a maquetar el modelo Etiqueta.php. En nuestro caso simplemente le diremos qué puede modificar y los ocultos que se van a rellenar solo.

```
Etiqueta.php U
app > Models > Etiqueta.php > App\Models\Etiqueta
1  <?php
2
3  namespace App\Models;
4
5  use Illuminate\Database\Eloquent\Factories\HasFactory;
6  use Illuminate\Database\Eloquent\Model;
7
8  5 references | 0 implementations
9  class Etiqueta extends Model
10 {
11     use HasFactory;
12
13     0 references
14     protected $guarded = [];
15
16     0 references
17     protected $hidden = ["created_at", "updated_at"];
18 }
```

## Relacional entre Tareas y Etiquetas

Ahora vamos a crear la tabla referencial, para ello vamos a utilizar el comando **'sail artisan make:migration create\_tarea\_etiqueta\_table'**

```
javi@Javi:/mnt/c/Users/Javier/Desktop/Estudios/2ºDAW/DWSE/Docker/www/apitareas$ sail artisan make:migration create_tarea_etiqueta_table
[INFO] Migration [database/migrations/2024_02_19_162126_create_tarea_etiqueta_table.php] created successfully.
```

Ahora maquetamos la estructura de la tabla:

```

2024_02_19_162126_create_tarea_etiqueta_table.php U X Tarea.php U
database > migrations > 2024_02_19_162126_create_tarea_etiqueta_table.php
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  {
9      /**
10       * Run the migrations.
11       */
12       public function up(): void
13       {
14           Schema::create('tarea_etiqueta', function (Blueprint $table) {
15               $table->id();
16               $table->foreignId('tarea_id')->constrained()->onDelete('cascade');
17               $table->foreignId('etiqueta_id')->constrained()->onDelete('cascade');
18               $table->timestamps();
19           });
20       }
21
22       /**
23        * Reverse the migrations.
24        */
25       public function down(): void
26       {
27           Schema::dropIfExists('tarea_etiqueta');
28       }
29 };

```

En el modelo Tarea.php debemos crear la relacional, para ello añadimos **“Function categorías(): BelongsToMany”** y especificamos tabla relacional, el id de la primera y el id de la segunda

```

Tarea.php U X
app > Models > Tarea.php > App\Models\Tarea > categorías()
1  <?php
2
3  namespace App\Models;
4
5  use Illuminate\Database\Eloquent\Factories\HasFactory;
6  use Illuminate\Database\Eloquent\Model;
7  use Illuminate\Database\Eloquent\Relations\BelongsToMany;
8
9  class Tarea extends Model
10 {
11     use HasFactory;
12
13     // No deja modificar todos los que estén dentro de corchetes, si esta vacío deja modificar todo
14     protected $guarded = [];
15
16     protected $hidden = ['created_at', 'updated_at'];
17
18     public function categorías(): BelongsToMany
19     {
20         return $this->belongsToMany(Etiqueta::class, 'tarea_etiqueta', 'tarea_id', 'etiqueta_id');
21     }
22 }

```

## Maquetación TareaController

Ahora si podemos completar nuestro TareaController.php:

Function index:

```

0 references | 0 overrides
public function index(): JsonResponse
{
    $tareas = Tarea::all();

    // return response()->json($tareas, 200); //Con el 200 controlamos el OK
    return TareaResource::collection($tareas); // 200 es para que todo ha ido bien
}

```

Function store:

```
0 references | 0 overrides
public function store(Request $request)
{
    $tarea = new Tarea();
    $tarea->nombre = $request->nombre;
    $tarea->descripcion = $request->descripcion;
    $tarea->save();

    $tarea->tareas()->attach($request->tareas); // Asociar tareas al tarea (relacion muchos a muchos)

    return new TareaResource($tarea); // 200 es para que todo ha ido bien
}
```

Function show:

```
0 references | 0 overrides
public function show($id):JsonResource
{
    $tarea = Tarea::find($id);
    return new TareaResource($tarea); // 200 es para que todo ha ido bien
}
```

Function update:

```
0 references | 0 overrides
public function update(Request $request, $id):JsonResource
{
    $tarea = Tarea::find($id);
    $tarea->nombre = $request->nombre;
    $tarea->descripcion = $request->descripcion;
    $tarea->categorias()->detach(); // Eliminar categorias del tarea (relacion muchos a muchos)

    $tarea->categorias()->attach($request->etiquetas);
    $tarea->save();

    return new TareaResource($tarea); // 200 es para que todo ha ido bienx
}
```

Function destroy:

```
0 references | 0 overrides
public function destroy(Tarea $tarea)
{
    $tarea->delete();
    return response()->json(['success' => true], 200); // 200 Para que devuelva OK
}
```

Ahora vamos a crear EtiquetaRequest y EtiquetaResource. Para ello seguiremos los pasos igual que antes con Tarea.

## Creación EtiquetaRequest

Para crear nuestro request vamos a utilizar el comando 'sail artisan make:request EtiquetaRequest'

```
javi@Javi:/mnt/c/Users/Javier/Desktop/Estudios/2ºDAW/DWSE/Docker/www/apitareas$ sail artisan make:request EtiquetaRequest
INFO Request [app/Http/Requests/EtiquetaRequest.php] created successfully.
```

Esto nos crea las requests en EtiquetaRequest.php.

En dicho requests le debemos poner en true la función authorize y añadimos las reglas que queremos que deba tener los request

```

EtiquetaRequest.php U X
app > Http > Requests > EtiquetaRequest.php > App\Http\Requests\EtiquetaRequest > rules()
1  <?php
2
3  namespace App\Http\Requests;
4
5  use Illuminate\Foundation\Http\FormRequest;
6
7  class EtiquetaRequest extends FormRequest
8  {
9      /**
10       * Determine if the user is authorized to make this request.
11       */
12      public function authorize(): bool
13      {
14          return true;
15      }
16
17      /**
18       * Get the validation rules that apply to the request.
19       */
20      @return array<string, \Illuminate\Contracts\Validation\ValidationRule|array<mixed>|string>
21      */
22      public function rules(): array
23      {
24          return [
25              'nombre' => 'required|max:40|min:3',
26          ];
27      }
28  }

```

## Creación EtiquetaResource

Ahora vamos a crear nuestro resource, para ello utilizamos el comando **'sail artisan make:resource EtiquetaResource'**

```

javi@Javi:/mnt/c/Users/Javier/Desktop/Estudios/2ºDAW/DWESE/Docker/www/apitareas$ sail artisan make:resource EtiquetaResource
INFO Resource [app/Http/Resources/EtiquetaResource.php] created successfully.

```

Esto nos crea el archivo EtiquetaResource. En dicho archivo vamos a decirle la estructura que queremos.

```

EtiquetaResource.php U X
app > Http > Resources > EtiquetaResource.php > ...
1  <?php
2
3  namespace App\Http\Resources;
4
5  use Illuminate\Http\Request;
6  use Illuminate\Http\Resources\Json\JsonResource;
7
8  class EtiquetaResource extends JsonResource
9  {
10     /**
11      * Transform the resource into an array.
12      *
13      * @return array<string, mixed>
14      */
15     public function toArray(Request $request): array
16     {
17         // return parent::toArray($request);
18         return [
19             'id' => $this->id,
20             'nombre' => 'Nombre: ' . $this->nombre,
21             'inventado' => 'Inventado: ',
22             'Tarea' => $this->tarea !== null ? $this->tarea->pluck('nombre') : []
23         ];
24     }
25 }

```

## Maquetación EtiquetaController

Por último, vamos a maquetar EtiquetaController.php al igual que con TareaController.

Function index:

```
0 references | 0 overrides
public function index()
{
    $etiquetas = Etiqueta::all();

    return EtiquetaResource::collection($etiquetas);
}
```

Function store:

```
public function store(EtiquetaRequest $request)
{
    $etiqueta = new Etiqueta();
    $etiqueta->nombre = $request->nombre;
    $etiqueta->save();

    return new EtiquetaResource($etiqueta); // 200 es para que todo ha ido bien
}
```

Function show:

```
public function show($id):JsonResource
{
    $etiqueta = Etiqueta::find($id);
    return new EtiquetaResource($etiqueta);
}
```

Function update:

```
public function update(Request $request, $id):JsonResource
{
    $etiqueta = Etiqueta::find($id);
    $etiqueta->nombre = $request->nombre;

    $etiqueta->save();

    return new EtiquetaResource($etiqueta); // 200 es para que todo ha ido bien
}
```

Function destroy:

```
0 references | 0 overrides
public function destroy(Etiqueta $etiqueta)
{
    $etiqueta->delete();
    return response()->json(['success' => true], 200); // 200 Para que devuelva OK
}
```

## Route Api

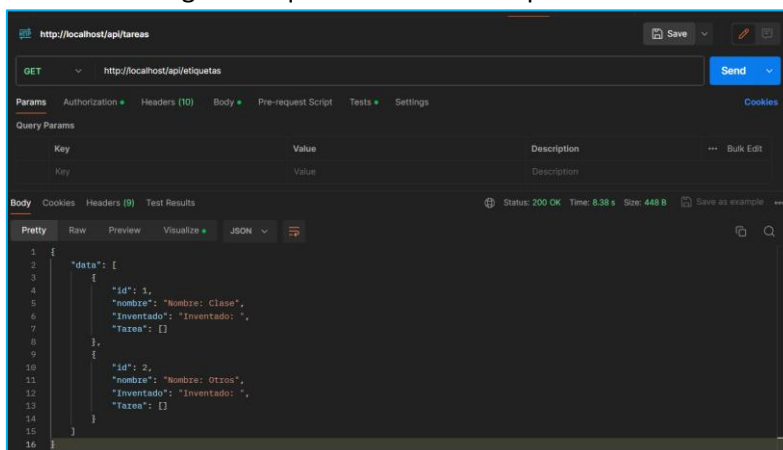
Por último, añadimos en api.php el route para etiquetas al igual que con tareas

```
api.php M X
routes > api.php
1 <?php
2
3 use App\Http\Controllers\AuthController;
4 use App\Http\Controllers\TareaController;
5 use App\Http\Controllers\EtiquetaController;
6 use Illuminate\Http\Request;
7 use Illuminate\Support\Facades\Route;
8
9 /*
10 |-----
11 | API Routes
12 |-----
13 |
14 | Here is where you can register API routes for your application. These
15 | routes are loaded by the RouteServiceProvider and all of them will
16 | be assigned to the "api" middleware group. Make something great!
17 |
18 */
19
20 Route::middleware('auth:sanctum')->get('/user', function (Request $request) {
21     return $request->user();
22 });
23
24 Route::resource('/tareas', TareaController::class);
25 Route::resource('/etiquetas', EtiquetaController::class);
26
27 Route::post('register', [AuthController::class, 'register']);
28 Route::post('login', [AuthController::class, 'login']);
29
30 Route::middleware('auth:sanctum')->group(function() {
31     Route::get('logout', [AuthController::class, 'logout']);
32 });
```

## Comprobando con Postman

Vamos a comprobar que todo funcione con el postman.

Si hacemos un get a etiquetas nos debería aparecer los datos en JSON:

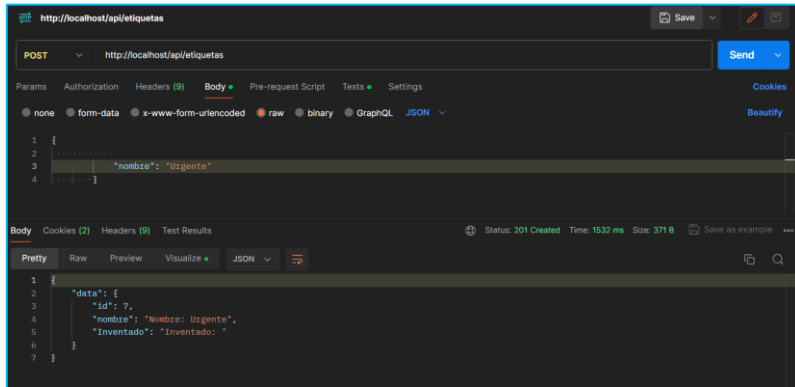


The screenshot shows the Postman interface with a GET request to `http://localhost/api/etiquetas`. The response is a JSON array of two objects, each representing an item with an ID, name, inventory status, and tasks.

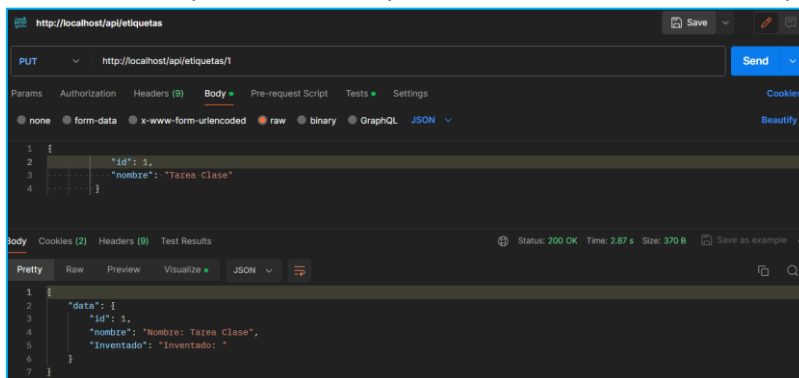
```
1 {
2   "data": [
3     {
4       "id": 1,
5       "nombre": "Nombre: Clase",
6       "inventado": "Inventado: ",
7       "tarea": []
8     },
9     {
10      "id": 2,
11      "nombre": "Nombre: Otros",
12      "inventado": "Inventado: ",
13      "tarea": []
14    }
15  ]
16 }
```



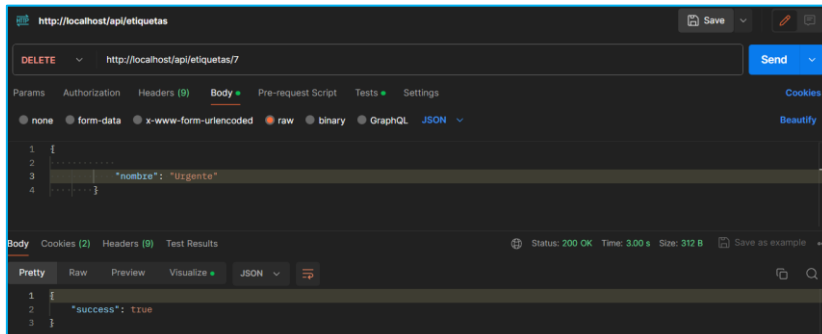
Para añadir etiquetas podemos hacer un post con los datos y se agregará a la base de datos:



Para hacer un update, en el body con estilo raw le escribimos el update:



Y si queremos borrar una etiqueta simplemente le pasaremos el id en la url:



## Creación login API REST

Para crear nuestro login con api rest debemos realizar el laravel/sactum, para ello debemos ejecutar el comando:

**'sail composer require laravel/sactum'**

```
javi@Javi:/mnt/c/Users/Javier/Desktop/Estudios/2ºDAW/DWESE/Docker/www/apitareas$ sail composer require laravel/sanctum
./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.

laravel/sail ..... DONE
laravel/sanctum ..... DONE
laravel/tinker ..... DONE
nesbot/carbon ..... DONE
nunomaduro/collision ..... DONE
nunomaduro/termwind ..... DONE
spatie/laravel-ignition ..... DONE

83 packages you are using are looking for funding.
Use the 'composer fund' command to find out more!
> @php artisan vendor:publish --tag=laravel-assets --ansi --force

 INFO  No publishable resources for tag [laravel-assets].

No security vulnerability advisories found.
Using version ^3.3 for laravel/sanctum
```

Ahora vamos a tomar la configuración de Laravel Sactum en nuestro proyecto, para ello deberemos ejecutar el siguiente comando

***sail artisan vendor:publish --provider="Laravel\Sanctum\SanctumServiceProvider"***

```
javi@Javi:/mnt/c/Users/Javier/Desktop/Estudios/2ºDAW/DWESE/Docker/www/apitareas$ sail artisan vendor:publish --provider="Laravel\Sanctum\SanctumServiceProv
ider"

 INFO  Publishing assets.

Copying directory [vendor/laravel/sanctum/database/migrations] to [database/migrations] ..... DONE
File [config/sanctum.php] already exists ..... SKIPPED
```

## Controlador AuthController

Ahora vamos a hacer un controlador para el login, logout y register, para ello deberemos ejecutar el comando **'sail artisan make:controller AuthController'**

```
javi@Javi:/mnt/c/Users/Javier/Desktop/Estudios/2ºDAW/DWESE/Docker/www/apitareas$ sail artisan make:controller AuthController

 INFO  Controller [app/Http/Controllers/AuthController.php] created successfully.
```

Esto nos crea un archivo un archivo en Controllers llamado **AuthController.php**

En dicho archivo introduciremos las funciones de login, logout y register. Para ello vamos a completar el archivo de la siguiente manera:

```

AuthController.php U X api.php M HasApiTokens.php
app > Http > Controllers > AuthController.php > App\Http\Controllers\AuthController > register()
1 <?php
2
3 namespace App\Http\Controllers;
4
5 use Illuminate\Http\Request;
6 use App\Models\User;
7 use Illuminate\Support\Facades\Auth;
8 use Illuminate\Support\Facades\Hash;
9
10 4 references | 0 implementations
11 class AuthController extends Controller
12 {
13     1 reference | 0 overrides
14     public function register(Request $request){
15         $user = User::create([
16             'name'=>$request->name,
17             'email'=>$request->email,
18             'password'=>Hash::make($request->password)
19         ]);
20         $token = $user->createToken('auth_token')->plainTextToken;
21         return response()->json(['data'=>$user,'access_token'=>$token,'token_type'=>'Bearer']);
22     }
23     1 reference | 0 overrides
24     public function logout(){
25         auth()->user()->tokens()->delete();
26         return ['message' => 'Sesión cerrada correctamente'];
27     }
28     1 reference | 0 overrides
29     public function login(Request $request){
30         $user = User::where('email', $request->email)->firstOrFail();
31         if(Hash::check($request->password, $user->password)){
32             return response()->json(['message'=>'Credenciales incorrectas']);
33         }
34         $token = $user->createToken('auth_token')->plainTextToken;
35         return response()->json(['message'=>'Hola ' . $user->name, 'access_token'=> $token, 'token_type'=>'Bearer']);
36     }
37 }
38

```

## Route api AuthController

Ahora en api.php debemos añadir los route para register, login y logout:

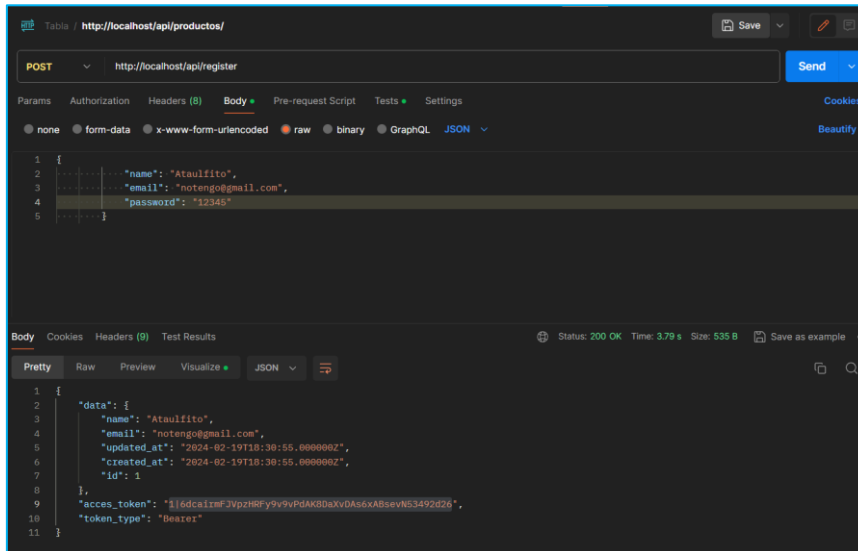
```

AuthController.php U api.php M X HasApiTokens.php
routes > api.php
1 <?php
2
3 use App\Http\Controllers\AuthController;
4 use App\Http\Controllers\TareaController;
5 use Illuminate\Http\Request;
6 use Illuminate\Support\Facades\Route;
7
8 /*
9 |-----
10 | API Routes
11 |-----
12 |
13 | Here is where you can register API routes for your application. These
14 | routes are loaded by the RouteServiceProvider and all of them will
15 | be assigned to the "api" middleware group. Make something great!
16 |
17 */
18
19 Route::middleware('auth:sanctum')->get('/user', function (Request $request) {
20     return $request->user();
21 });
22
23 Route::resource('/tarear', TareaController::class);
24
25 Route::post('register', [AuthController::class, 'register']);
26 Route::post('login', [AuthController::class, 'login']);
27
28 Route::middleware('auth:sanctum')->group(function() {
29     Route::get('logout', [AuthController::class, 'logout']);
30 });
31
32

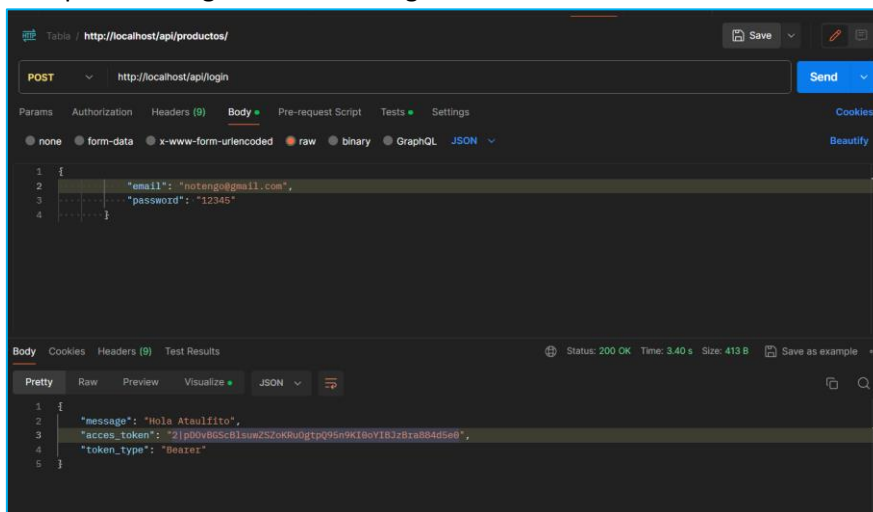
```

## Comprobando Auth con postman

Para comprobar que funciona correctamente vamos al postman, haciendo un post al register de la siguiente manera nos devolvera el acces\_token

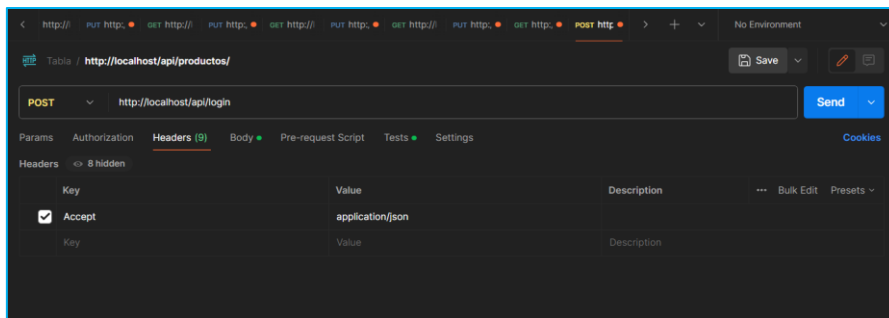


Para probar el login hacemos lo siguiente:

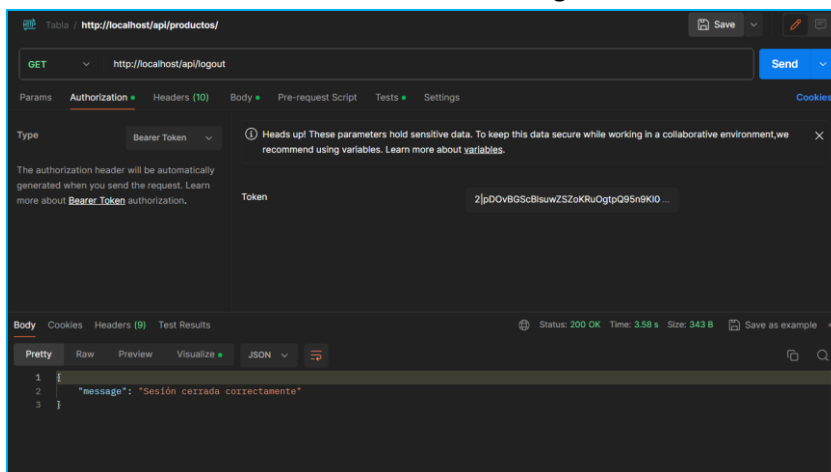


Y nos debe devolver el token.

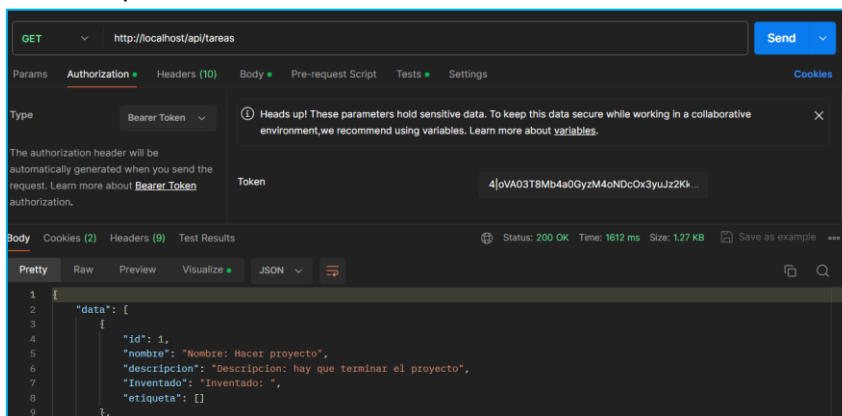
Recordar en Headers tener lo siguiente:



Para cerrar sesion haremos la consulta de la siguiente manera:



Si implementamos que sea requerido estar logueado para atacar a los endpoints debemos recordar que debemos meter nuestro token en authorization:



## Implementación de test

Vamos a implementar test para comprobar el correcto funcionamiento de nuestro servidor. Para ello vamos a crear nuestro modelo test con sus diferentes funciones. Debemos ejecutar el siguiente comando:

**'sail artisan make:test TareasTest'**

```
javi@Javi:/mnt/c/Users/Javier/Desktop/Estudios/2ºDAW/DWEESE/Docker/www/apitareas$ sail artisan make:test TareasTest
INFO Test [tests/Feature/TareasTest.php] created successfully.
```

Esto nos creará un archivo donde podremos implementar funciones que se ejecutaran cuando hagamos los test en nuestro servidor. En nuestro caso, para 'TareasTest' vamos a implementar test para comprobar que las funciones como 'get', 'update' o 'delete' funcionen correctamente.

Antes de nada, saber que si tenemos obligatorio el login y queremos hacer los test, tendremos que añadir las siguientes líneas en nuestras funciones:

Primero creamos la variable de user.

```
$user = User::factory()->create();
```

`"$user = User::factory()->create();"`

Y adaptamos nuestras funciones para que actúe como si fuera ese usuario.

```
$response = $this->actingAs($user)->withSession(['banned' => false])->getJson('api/tareas');
```

`"$response = $this->actingAs($user)->withSession(['banned' => false])->getJson('api/tareas');"`

E incluir los objetos Tarea y Users:

`"use App\Models\Tarea;"`

`"use App\Models\User;"`

```
namespace Tests\Feature;

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

Además deberemos usar la función **'use RefreshDatabase'** para que pueda hacer las pruebas simulando nuestra base de datos.

Nuestras funciones del test serán las siguientes:

Función test\_get:

```
class TareasTest extends TestCase
{
    use RefreshDatabase;
    // references | 0 overrides
    public function test_get_tareas()
    {
        $user = User::factory()->create();

        $tareas = new Tarea();
        $tareas->nombre = "TestTitulo";
        $tareas->descripcion = "TestDescripcion";
        $tareas->save();

        $response = $this->actingAs($user)->withSession(['banned' => false])->getJson('api/tareas');
        $response->assertStatus(200);

        $response->assertJsonFragment([
            'id' => $tareas->id,
            'nombre' => 'Nombre: ' . $tareas->nombre,
            'descripcion' => 'Descripcion: ' . $tareas->descripcion,
            'inventado' => 'Inventado: ',
            'etiqueta' => $tareas->etiquetas!=null?$tareas->etiquetas->pluck('nombre'):[]
        ]);
    }
}
```

Funcion test\_insert:

```
0 references | 0 overrides
public function test_insert_tarea()
{
    $user = User::factory()->create();

    // Datos de la nueva tarea
    $newTareaData = [
        'nombre' => 'NuevoTitulo',
        'descripcion' => 'NuevaDescripcion',
    ];

    // Enviar una solicitud de inserción a la API
    $response = $this->actingAs($user)->withSession(['banned' => false])->postJson('api/tareas', $newTareaData);
    $response->assertStatus(201); // Verificar que la tarea ha sido creada exitosamente

    // Verificar que la tarea ha sido insertada en la base de datos
    $this->assertDatabaseHas('tareas', [
        'nombre' => $newTareaData['nombre'],
        'descripcion' => $newTareaData['descripcion'],
    ]);
}
```

Función test\_update:

```
public function test_update_tarea()
{
    $user = User::factory()->create();

    // Crear una tarea para ser actualizada
    $tarea = new Tarea();
    $tarea->nombre = "TestTitulo";
    $tarea->descripcion = "TestDescripcion";
    $tarea->save();

    // Datos de actualización
    $updatedData = [
        'nombre' => 'NuevoTitulo',
        'descripcion' => 'NuevaDescripcion',
    ];

    // Enviar una solicitud de actualización a la API
    $response = $this->actingAs($user)->withSession(['banned' => false])->putJson("api/tareas/{$tarea->id}", $updatedData);
    $response->assertStatus(200);

    // Refrescar el modelo desde la base de datos
    $tarea->refresh();

    // Verificar que la tarea ha sido actualizada en la base de datos
    $this->assertEquals($updatedData['nombre'], $tarea->nombre);
    $this->assertEquals($updatedData['descripcion'], $tarea->descripcion);
}
```

## Función test\_delete:

```

public function test_delete_tarea()
{
    $user = User::factory()->create();

    // Crear una tarea para ser eliminada
    $tarea = new Tarea();
    $tarea->nombre = "TestTitulo";
    $tarea->descripcion = "TestDescripcion";
    $tarea->save();

    // Enviar una solicitud de eliminación a la API
    $response = $this->actingAs($user)->withSession(['banned' => false])->deleteJson("api/tareas/{$tarea->id}");
    $response->assertStatus(200);

    // Verificar que la tarea ha sido eliminada de la base de datos
    $this->assertNull(Tarea::find($tarea->id));
}

```

Para realizar las pruebas de test simplemente tendremos que ejecutar el comando

## ‘sail artisan test’:

```

javi@Javi:/mnt/c/Users/Javier/Desktop/Estudios/2ºDAW/DWSE/Docker/www/apitareas$ sail artisan test

PASS Tests\Unit\ExampleTest
✓ that true is true 0.31s

PASS Tests\Feature\ExampleTest
✓ the application returns a successful response 5.48s

PASS Tests\Feature\TareasTest
✓ get tareas 4.45s
✓ update tarea 0.11s
✓ delete tarea 0.15s
✓ insert tarea 0.58s

Tests: 6 passed (15 assertions)
Duration: 13.60s

```

Y comprobamos que todo ha ido correctamente.

Vamos a crear ahora un test para las etiquetas con la misma estructura de funciones. Para ello repetiremos el proceso que ya hemos realizado con el testareas usando el siguiente comando:

## ‘sail artisan make:test EtiquetasTest’

## ‘Recordar usar los import de los modelos y el uso del RefreshDataBase’

## Función test\_get\_etiquetas:

```

public function test_get_tareas()
{
    $user = User::factory()->create();

    $tareas = new Tarea();
    $tareas->nombre = "TestTitulo";
    $tareas->descripcion = "TestDescripcion";
    $tareas->save();

    $response = $this->actingAs($user)->withSession(['banned' => false])->getJson('api/tareas');
    $response->assertStatus(200);

    $response->assertJsonFragment([
        'id' => $tareas->id,
        'nombre' => 'Nombre: ' . $tareas->nombre,
        'descripcion' => 'Descripcion: ' . $tareas->descripcion,
        'inventado' => 'Inventado: ',
        'etiqueta' => $tareas->etiquetas!=null?$tareas->etiquetas->pluck('nombre'):[]
    ]);
}

```



## Función test\_insert\_etiquetas:

```

public function test_insert_tarea()
{
    $user = User::factory()->create();

    // Datos de la nueva tarea
    $newTareaData = [
        'nombre' => 'NuevoTitulo',
        'descripcion' => 'NuevaDescripcion',
    ];

    // Enviar una solicitud de inserción a la API
    $response = $this->actingAs($user)->withSession(['banned' => false])->postJson('api/tareas', $newTareaData);
    $response->assertStatus(201); // Verificar que la tarea ha sido creada exitosamente

    // Verificar que la tarea ha sido insertada en la base de datos
    $this->assertDatabaseHas('tareas', [
        'nombre' => $newTareaData['nombre'],
        'descripcion' => $newTareaData['descripcion'],
    ]);
}

```

## Función test\_update\_etiquetas:

```

public function test_update_tarea()
{
    $user = User::factory()->create();

    // Crear una tarea para ser actualizada
    $tarea = new Tarea();
    $tarea->nombre = "TestTitulo";
    $tarea->descripcion = "TestDescripcion";
    $tarea->save();

    // Datos de actualización
    $updatedData = [
        'nombre' => 'NuevoTitulo',
        'descripcion' => 'NuevaDescripcion',
    ];

    // Enviar una solicitud de actualización a la API
    $response = $this->actingAs($user)->withSession(['banned' => false])->putJson("api/tareas/{$tarea->id}", $updatedData);
    $response->assertStatus(200);

    // Refrescar el modelo desde la base de datos
    $tarea->refresh();

    // Verificar que la tarea ha sido actualizada en la base de datos
    $this->assertEquals($updatedData['nombre'], $tarea->nombre);
    $this->assertEquals($updatedData['descripcion'], $tarea->descripcion);
}

```

## Función test\_delete\_etiquetas:

```

public function test_delete_tarea()
{
    $user = User::factory()->create();

    // Crear una tarea para ser eliminada
    $tarea = new Tarea();
    $tarea->nombre = "TestTitulo";
    $tarea->descripcion = "TestDescripcion";
    $tarea->save();

    // Enviar una solicitud de eliminación a la API
    $response = $this->actingAs($user)->withSession(['banned' => false])->deleteJson("api/tareas/{$tarea->id}");
    $response->assertStatus(200);

    // Verificar que la tarea ha sido eliminada de la base de datos
    $this->assertNull(Tarea::find($tarea->id));
}

```

Y si volvemos a ejecutar el comando **“sail artisan test”** podemos volver a ejecutar los test, esta vez también para las etiquetas:

```

levi@Javi:/mnt/c/Users/Javier/Desktop/Estudios/2ºDAW/DWSE/Docker/www/apitareas$ sail artisan test

PASS: Tests\Unit\ExampleTest
✓ that true is true 0.11s

PASS: Tests\Feature\EtiquetasTest
✓ get etiquetas 9.51s
✓ update tarea 0.08s
✓ delete tarea 0.12s
✓ insert tarea 0.34s

PASS: Tests\Feature\ExampleTest
✓ the application returns a successful response 0.60s

PASS: Tests\Feature\TareasTest
✓ get tareas 0.18s
✓ update tarea 0.09s
✓ delete tarea 0.08s
✓ insert tarea 0.09s

Tests: 10 passed (25 assertions)
Duration: 12.76s

```

Vamos a crear ahora un test para el login con funciones para comprobar el login, register y logout. Para ello repetiremos el proceso que ya hemos realizado con el testareas usando el siguiente comando:

**'sail artisan make:test LoginTest'**

```
javi@Javi: /mnt/c/Users/Javier/Desktop/Estudios/2ºDAW/DWESE/Docker/www/apitareas$ sail artisan make:test LoginTest
INFO Test [tests/Feature/LoginTest.php] created successfully.
```

**'Recordar usar los import de los modelos y el uso del RefreshDataBase'**

Función test\_register\_user:

```
public function test_register_user()
{
    $userData = [
        'name' => 'John Doe',
        'email' => 'john@example.com',
        'password' => 'password123',
    ];

    $response = $this->postJson('api/register', $userData);
    $response->assertStatus(200)
        ->assertJsonStructure(['data', 'acces_token', 'token_type']);
}
```

Función test\_login\_user:

```
public function test_login_user()
{
    $user = User::factory()->create();

    $loginData = [
        'email' => $user->email,
        'password' => 'password',
    ];

    $response = $this->postJson('api/login', $loginData);
    $response->assertStatus(200)
        ->assertJsonStructure(['message', 'acces_token', 'token_type']);
}
```

Función test\_logout\_user:

```
public function test_logout_user()
{
    $user = User::factory()->create();
    $token = $user->createToken('auth_token')->plainTextToken;

    $this->actingAs($user)->assertAuthenticated();

    $response = $this->actingAs($user)->withToken($token)->getJson('api/logout');
    $response->assertStatus(200)
        ->assertJson(['message' => 'Sesión cerrada correctamente']);
}
```

Y si volvemos a ejecutar el comando **“sail artisan test”** podemos volver a ejecutar los test, esta vez también para las funciones de login:

```
javi@Javi:/mnt/c/Users/Javier/Desktop/Estudios/2ºDAW/DWEESE/Docker/www/apitareas$ sail artisan test

PASS Tests\Unit\ExampleTest
✓ that true is true 0.12s

PASS Tests\Feature\EtiquetasTest
✓ get etiquetas 7.24s
✓ update tarea 0.11s
✓ delete tarea 0.10s
✓ insert tarea 0.39s

PASS Tests\Feature\ExampleTest
✓ the application returns a successful response 0.58s

PASS Tests\Feature\LoginTest
✓ register user 0.16s
✓ login user 0.05s
✓ logout user 0.05s

PASS Tests\Feature\TareasTest
✓ get tareas 0.12s
✓ update tarea 0.06s
✓ delete tarea 0.06s
✓ insert tarea 0.08s

Tests: 13 passed (36 assertions)
Duration: 10.74s
```

Con esto ya tendríamos los test principales para comprobar el correcto funcionamiento en nuestros endpoints que hemos configurado anteriormente.