

PROYECTO LARAVEL



Índice

1. Instalación Laravel
2. Migration
3. Seeders
4. Models
5. Controllers
6. Resource and resource
7. Usuario
8. Postman
9. Tests

1. Instalación Laravel

En esta guía verás un paso a paso, de como instalar Laravel Sail, con sus dependencias, y a su vez la creación de una Api Rest. Veremos cómo crear, mostrar, actualizar y borrar tareas de una base de datos, todo ello se controlará a través de un login, registro y logout de usuario.

Empezaremos creando un proyecto laravel, utilizando el siguiente comando:

- composer create-project --prefer-dist laravel/laravel apiTaskLA

```
arkangel@Arkangel Bloque6_Laravel % composer create-project --prefer-dist laravel/laravel apiTaskLA
Creating a "laravel/laravel" project at "./apiTaskLA"
Installing laravel/laravel (v10.3.3)
  - Downloading laravel/laravel (v10.3.3)
    - Installing laravel/laravel (v10.3.3): Extracting archive
Created project in /Users/arkangel/GradoDAWMAC/2DAW/DWServidor/jmoybae/Ejercicios/docker/www/Bloque6_Laravel/apiTaskLA
> @php -r "file_exists('.env') || copy('.env.example', '.env');"
Loading composer repositories with package information
Updating dependencies
Lock file operations: 111 installs, 0 updates, 0 removals
  - Locking brick/math (0.11.0)
  - Locking carbonnbo/carbon-doctrine-types (2.1.0)
```

Seguidamente instalamos el sail, y la instalación del npm:

- **php artisan sail:install**

```
arkangel@Arkangel apiTaskLA % php artisan sail:install
Which services would you like to install? _____
  mysql

[+] Pulling 1/1
✓ mysql Pulled
[+] Building 197.6s (17/17) FINISHED
  => [laravel.test internal] load build definition from Dockerfile
  => => transferring dockerfile: 3.21kB
  => [laravel.test internal] load metadata for docker.io/library/ubuntu:22.04
  => [laravel.test auth] library/ubuntu:pull token for registry-1.docker.io
  => [laravel.test internal] load .dockerignore
  => => transferring context: 2B
```

2.4s	docker:desktop-linux
0.0s	0.0s
0.0s	1.9s
0.0s	0.0s
0.0s	0.0s

- **npm install**

```
arkangel@Arkangel Bloque6_Laravel % cd apiTaskLA
arkangel@Arkangel apiTaskLA % npm install
added 23 packages, and audited 24 packages in 1s
5 packages are looking for funding
  run `npm fund` for details
found 0 vulnerabilities
```

Cuando se terminen de ejecutar los comandos anteriores, seguimos con el levantamiento del contenedor docker.

- Primero hacemos un alias, para facilitar el trabajo, a la hora de ejecutar comandos:
- alias sail=./vendor/bin/sail
- sail up -d

```
arkangel@Arkangel apiTaskLA % alias sail=./vendor/bin/sail
arkangel@Arkangel apiTaskLA % sail up -d
[+] Running 2/2
  ✓ Container apitaskla-mysql-1      Started
  ✓ Container apitaskla-laravel.test-1 Started
```

2. Migration

Ahora toca crear las tablas de la base de datos con migration, usando los comandos:

- sail artisan make:migration create_tareas_table
- sail artisan make:migration create_etiquetas_table
- sail artisan make:migration create_tarea_etiqueta_table

```
arkangel@Arkangel apiTaskLA % sail artisan make:migration create_tareas_table && sail artisan make:migration create_etiquetas_table && sail artisan make:migration create_tarea_etiqueta_table
INFO Migration [database/migrations/2024_02_22_155656_create_tareas_table.php] created successfully.

INFO Migration [database/migrations/2024_02_22_155656_create_etiquetas_table.php] created successfully.

INFO Migration [database/migrations/2024_02_22_155657_create_tarea_etiqueta_table.php] created successfully.
```

- Tendremos que modificar los archivos que acabamos de crear, a nuestro gusto, en mi caso esta de esta manera:

```
database > migrations > 2024_02_22_164302_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      public function up(): void
10     {
11         Schema::create('tareas', function (Blueprint $table) {
12             $table->id();
13             $table->string('titulo', 40);
14             $table->string('descripcion')->nullable();
15             $table->timestamps();
16         });
17     }
18
19     public function down(): void
20     {
21         Schema::dropIfExists('tareas');
22     }
23 };
24 |
```

```
database > migrations > 2024_02_22_164303_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     public function up(): void
11     {
12         Schema::create('etiquetas', function (Blueprint $table) {
13             $table->id();
14             $table->string('nombre', 20);
15             $table->timestamps();
16         });
17     }
18
19     public function down(): void
20     {
21         Schema::dropIfExists('etiquetas');
22     }
23 };
24 |
```

```
database > migrations > 2024_02_22_164304_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     public function up(): void
11     {
12         Schema::create('tarea_etiqueta', function (Blueprint $table) {
13             $table->id();
14             $table->foreignId("tarea_id")->constrained('tareas');
15             $table->foreignId("etiqueta_id")->constrained('etiquetas');
16             $table->timestamps();
17         });
18     }
19
20     public function down(): void
21     {
22         Schema::dropIfExists('tarea_etiqueta');
23     }
24 };
25
```

3. Seeders

Una vez creados para agregar a la base de datos las tablas que acabamos de configurar tendríamos que usar los siguientes comandos:

- Para crear las tablas: **sail artisan migrate**
- Si queremos borrar las tablas y volverlas a crear: **sail artisan migrate:refresh**
- Y para borrar una tabla específica o hacer un reseteo de la misma sería:
sail artisan migrate:rollback --step=3 (en este caso borraría la fila 3, podrías borrar la que quieras)

Ahora crearemos los ficheros en los que configuraremos lo que va dentro de las tablas que acabamos de crear de la base de datos:

- Para crear los ficheros:
 - **sail artisan make:seeder TareasSeeder**
 - **sail artisan make:seeder EtiquetasSeeder**

```
arkangel@Arkangel apiTaskLA % sail artisan make:seeder TareaSeeder && sail artisan make:seeder EtiquetasSeeder
[INFO] Seeder [database/seeders/TareaSeeder.php] created successfully.

[INFO] Seeder [database/seeders/EtiquetasSeeder.php] created successfully.
```

- Justo después tendríamos que añadir lo que quisiéramos dentro de las tablas tareas y etiquetas, junto con el fichero DatabaseSeeder.

```
database > seeders >  TareaSeeder.php >  TareaSeeder >  run
1  <?php
2
3  namespace Database\Seeders;
4
5  use Illuminate\Support\Facades\DB;
6  use Illuminate\Database\Seeder;
7
8  class TareaSeeder extends Seeder
9  {
10     /**
11      * Run the database seeds.
12      */
13     public function run(): void
14     {
15         DB::table('tareas')->insert([
16             [
17                 'titulo' => 'Compra del mes',
18                 'descripcion' => 'Lacteos: leche, yogurts, queso;
19                                         Hidratos: pan, pasta, arroz;
20                                         Proteinas: pollo, ternera, cerdo.',
21             ],
22             [
23                 'titulo' => 'Limpiar habitaciones',
24                 'descripcion' => 'Limpiar exhaustivamente el WC y poner lavadora.',
25             ],
26             [
27                 'titulo' => 'Llevar a la abuela al medico',
28                 'descripcion' => 'Hay que recogerla del canto porque a las 5pm tiene traumatologo.',
29             ],
30             [
31                 'titulo' => 'Proyecto Laravel',
32                 'descripcion' => 'Hay que realizar el proyecto laravel para el dia 4 de Marzo.',
33             ]
34         ]);
35     }
}
```

```
database > seeders >  EtiquetasSeeder.php > ...
1  <?php
2
3  namespace Database\Seeders;
4
5  use Illuminate\Support\Facades\DB;
6  use Illuminate\Database\Seeder;
7
8  class EtiquetasSeeder extends Seeder
9  {
10     /**
11      * Run the database seeds.
12      */
13     public function run(): void
14     {
15         DB::table('etiquetas')->insert([
16             [
17                 'nombre' => 'Home',
18             ],
19             [
20                 'nombre' => 'Familia',
21             ],
22             [
23                 'nombre' => 'Trabajo',
24             ]
25         ]);
26     }
27 }
28 |
```

```
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  class DatabaseSeeder extends Seeder
9  {
10     /**
11      * Seed the application's database.
12      */
13     public function run(): void
14     {
15         $this->call([TareaSeeder::class,EtiquetasSeeder::class]);
16     }
17 }
18 |
```

- Para introducir los datos en la base de datos tendríamos que ejecutar el siguiente comando: **sail artisan db:seed**.

```
arkangel@Arkangel apitaskLA % sail artisan db:seed
[INFO] Seeding database.
Database\Seeders\TareaSeeder ..... RUNNING
Database\Seeders\TareaSeeder ..... 32 ms DONE
Database\Seeders\EtiquetasSeeder ..... RUNNING
Database\Seeders\EtiquetasSeeder ..... 4 ms DONE
```

Después de terminar de modificar la base de datos yo lo que haría sería, ejecutar los siguientes comandos para confirmar que se ha creado todo correctamente y añadidos los valores a las tablas:

- **sail artisan migrate:refresh && sail artisan db:seed**

4. MODELS

Seguimos configurando nuestra API REST para ello continuamos con los Models, en este caso con los modelos Etiquetas y Tareas, añadiendo al final de cada comando la terminación ‘-cr’, nos ayuda a la hora de crearnos también los controladores y rellenar cada una de las funciones necesarias para su correcto funcionamiento:

- **sail artisan make:model Etiquetas -cr && sail artisan make:model Tareas -cr**
- Lo mismo que hemos estado haciendo, vamos configurando los ficheros recién creados:

```
app > Models > Tarea.php > ...
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     protected $guarded = [];
14     protected $hidden = ['created_at', 'updated_at'];
15
16     public function etiquetas(): BelongsToMany
17     {
18         return $this->belongsToMany(Etiqueta::class, 'tarea_etiqueta',
19             'tarea_id', 'etiqueta_id');
20     }
21 }
22 }
```

5. CONTROLLERS

Ya que hemos usado la terminación ‘-cr’, ahora vamos a llenar esas funciones creadas:

```
public function index():JsonResource
{
    $tareas = Tarea::all();
    return TareaResource::collection($tareas);
}
```

```
public function index():JsonResource
{
    $etiquetas = Etiqueta::all();
    return EtiquetaResource::collection($etiquetas);
}
```

```
public function store(TareaRequest $request):JsonResource
{
    $tarea=new Tarea();
    $tarea->titulo = $request->titulo;
    $tarea->descripcion = $request->descripcion;
    $tarea->save();

    $tarea->etiquetas()->attach($request->etiquetas);
    return new TareaResource($tarea);
}
```

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

    $etiqueta->tareas()->attach($request->tareas);
    return new EtiquetaResource($etiqueta);
}
```

```
public function show($id):JsonResource
{
    $tarea = Tarea::find($id);
    return new TareaResource($tarea);
}
```

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

```
public function update(TareaRequest $request, $id):JsonResource
{
    $tarea = Tarea::find($id);
    $tarea->titulo = $request->titulo;
    $tarea->descripcion = $request->descripcion;
    $tarea->etiquetas()->detach();

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

    return new TareaResource($tarea);
}
```

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

    $etiqueta->tareas()->attach($request->tareas) ;
    $etiqueta->save();

    return new EtiquetaResource($etiqueta);
}
```

```
public function destroy($id)
{
    Tarea::find($id)->delete();
    return response()->json(['success' => true],200);
}
```

6. REQUEST AND RESOURCE

Nos toca ahora crear los ficheros que nos van a ayudar a recopilar la información y valores que nos va a ir llegando a través de los endpoints, y para mostrar los valores que nos pidan, tendremos que crear los ficheros resource, creándolos con los comandos:

- **sail artisan make:request EtiquetaRequest && sail artisan make:resource EtiquetaResource**

```
● arkangel@Arkangel apiTaskLA % sail artisan make:request EtiquetaRequest
  INFO Request [app/Http/Requests/EtiquetaRequest.php] created successfully.

● arkangel@Arkangel apiTaskLA % sail artisan make:resource EtiquetaResource
  INFO Resource [app/Http/Resources/EtiquetaResource.php] created successfully.
```

- **sail artisan make:request TareaRequest && sail artisan make:resource TareaResource**

```
arkangel@Arkangel apiTaskLA % sail artisan make:request TareaRequest
  INFO Request [app/Http/Requests/TareaRequest.php] created successfully.

arkangel@Arkangel apiTaskLA % sail artisan make:resource TareaResource
  INFO Resource [app/Http/Resources/TareaResource.php] created successfully.
```

- Despues de crearlos tendríamos que añadir las reglas que queramos que tenga cada uno de los valores recibidos

```
public function toArray(Request $request): array
{
    return [
        'id' => $this->id,
        'titulo' => 'Titulo: ' . $this->titulo,
        'descripcion' => 'Descripcion: ' . $this->descripcion,
        'etiquetas' => 'Etiqueta: ' . $this->etiquetas
    ];
}

public function toArray(Request $request): array
{
    return [
        'id' => $this->id,
        'nombre' => 'Nombre: ' . $this->nombre
    ];
}

public function rules(): array
{
    return [
        'titulo'=> 'required|max:40|min:3',
        'descripcion'=> 'nullable|max:255|min:3'
    ];
}

public function rules(): array
{
    return [
        'nombre'=> 'required|max:40|min:3'
    ];
}
```

7. USUARIO

Nos faltaría por configurar el login, registro y logout de los usuarios. Empezamos ejecutando el comando, para controlar los paquetes usando tokens:

- **sail composer require laravel/sanctum**

```
arkangel@Arkangel apiTaskLA % 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
Class App\Http\Controllers\EtiquetasController located in ./app/Http/Controllers/EtiquetasController.php does not comply with psr-4 autoloading standard. Skipping.
Class App\Http\Controllers\TareasController located in ./app/Http/Controllers/TareasController.php does not comply with psr-4 autoloading standard. Skipping.
> 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
```

Seguimos con el comando, que terminará de instalar las dependencias de Sanctum:

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

```
arkangel@Arkangel apiTaskLA % sail artisan vendor:publish --provider="Laravel\Sanctum\SanctumServiceProvider"
[INFO] No publishable resources for tag [].
```

Para controlar los paquetes, de login, registro y logout, tendremos que crear el controlador, AuthController:

- **sail artisan make:controller AuthController**

```
arkangel@Arkangel apiTaskLA % sail artisan make:controller AuthController
[INFO] Controller [app/Http/Controllers/AuthController.php] created successfully.
```

- Dentro de él vamos a crear las funciones login, registro y logout

```
public function register(Request $request)
{
    $user = User::create([
        'name' => $request->name,
        'email' => $request->email,
        'password' => Hash::make($request->password),
    ]);

    $token = $user->createToken('auth_token')->plainTextToken;

    return response()->json([
        'data' => $user,
        'access_token' => $token,
        'token_type' => 'Bearer',
    ]);
}

public function login(Request $request)
{
    $user = User::where('email', $request->email)->firstOrFail();

    if (!Hash::check($request->password, $user->password)) {
        return response()->json(
            [
                'message' => 'Contraseña incorrecta'
            ],
            401
        );
    }

    $token = $user->createToken('auth_token')->plainTextToken;

    return response()->json([
        'message' => 'Hola' . $user->name,
        'user' => $user,
        'access_token' => $token,
        'token_type' => 'Bearer'
    ]);
}

public function logout()
{
    auth()->user()->tokens()->delete();
    return [
        'message' => 'Session cerrada correctamente'
    ];
}
```

Ahora nos tocaría modificar el fichero api.php, que será el encargado de proporcionar las rutas de autenticación de los usuarios, con las operaciones CRUD (CREATE, READ, UPDATE, DELETE).

```
routes >  api.php > ...
1  <?php
2
3
4  use App\Http\Controllers\AuthController;
5  use Illuminate\Http\Request;
6  use Illuminate\Support\Facades\Route;
7  use App\Http\Controllers\TareaController;
8  use App\Http\Controllers\EtiquetaController;
9
10 Route::middleware('auth:sanctum')->get('/user', function (Request $request) {
11     return $request->user();
12 });
13
14 Route::post('register', [AuthController::class, 'register']);
15 Route::post('login', [AuthController::class, 'login']);
16
17 Route::middleware('auth:sanctum')->group( function () {
18     Route::get('logout', [AuthController::class, 'logout']);
19
20     Route::resource('/tareas', TareaController::class);
21
22     Route::resource('/etiquetas', EtiquetaController::class);
23 });
```

8. POSTMAN

Usaremos la plataforma POSTMAN para ejecutar los endpoints a nuestra api:

- Registro: Usando el método POST, indicamos la url, seleccionamos body, he introducimos el nombre, email y password.

```

POST http://localhost/api/register
http://localhost/api/register

POST http://localhost/api/register
Params Authorization Headers (9) Body Pre-request Script Tests Settings
none form-data x-www-form-urlencoded raw binary GraphQL JSON
1 {
2   "name": "test",
3   "email": "test@test.com",
4   "password": "1234"
5 }

Body Cookies Headers (9) Test Results
Pretty Raw Preview Visualize JSON
1 {
2   "data": {
3     "name": "test",
4     "email": "test@test.com",
5     "updated_at": "2024-02-26T17:10:28.000000Z",
6     "created_at": "2024-02-26T17:10:28.000000Z",
7     "id": 1
8   },
9   "access_token": "1|4Blea2gq987n5jN5fVfRgCi7oq8ESJ6B8tVmQ0lHf4688581",
10  "token_type": "Bearer"
11 }
    
```

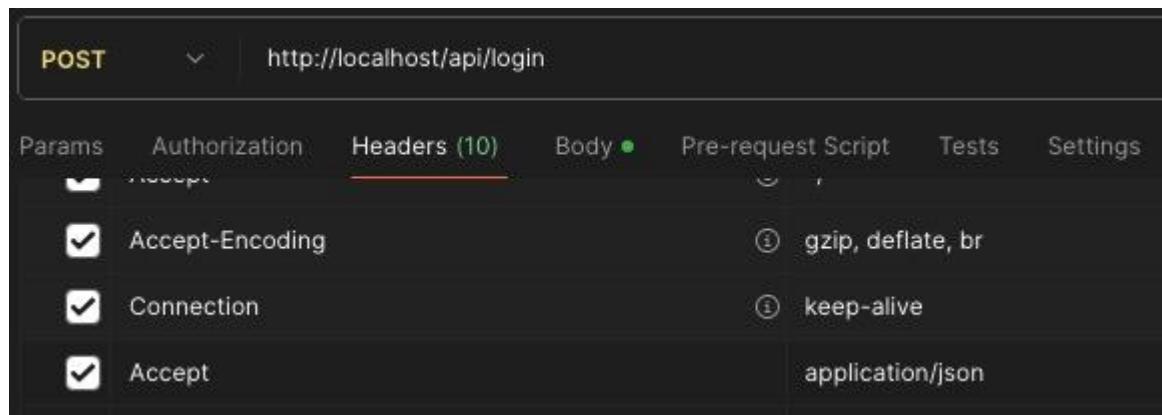
- Login: Lo mismo que en registro, usando POST, indicando la url, y seleccionamos body, he introducimos el email y la password, también en el header tendríamos que añadir Accept – application/json:

```

POST http://localhost/api/login
http://localhost/api/login

POST http://localhost/api/login
Params Authorization Headers (10) Body Pre-request Script Tests Settings
none form-data x-www-form-urlencoded raw binary GraphQL JSON
1 {
2   "email": "test@test.com",
3   "password": "1234"
4 }

Body Cookies Headers (9) Test Results
Pretty Raw Preview Visualize JSON
1 {
2   "message": "Holatest",
3   "user": [
4     {
5       "id": 1,
6       "name": "test",
7       "email": "test@test.com",
8       "email_verified_at": null,
9       "created_at": "2024-02-26T17:10:28.000000Z",
10      "updated_at": "2024-02-26T17:10:28.000000Z"
11    },
12    {
13      "access_token": "2|T5d0jYiZnvKNdS1j1qwNFQlueCl90zQ6S6WcqMNb188e277",
14      "token_type": "Bearer"
15    }
16  ]
    
```



- Para ver todas las tareas, tendríamos que: usar el método GET, y añadir el mismo header que login, y la autenticación el token que nos ha salido a la hora de logearnos:

The screenshot shows the Postman interface with the following details:

- Method:** GET
- URL:** http://localhost/api/tareas
- Auth:** Bearer Token (selected)
- Token:** 2|0KkV4NnDtTeqn0WvB61fcc2L...
- Body:** JSON (Pretty) - Displays the response data as follows:

```
1 {  
2   "data": [  
3     {  
4       "id": 1,  
5       "titulo": "Titulo: Compra del mes",  
6       "descripcion": "Descripcion: Lacteos: leche, yogurts, queso;  
7           \n          Hidratos: pan, pasta, arroz;  
8           \n          Proteinas: pollo, ternera, cerdo.",  
9       "etiquetas": "Etiqueta: []"  
10      },  
11      {  
12        "id": 2,  
13        "titulo": "Titulo: Limpiar habitaciones",  
14        "descripcion": "Descripcion: Limpiar exhaustivamente el WC  
15           \n           y poner lavadora.",  
16      }  
17    ]  
18  }  
19 }
```

- Para ver una sola tarea, hay que usar la misma configuración que para ver todas las tareas pero cambiando la url y añadiendo una barra con el id concreto a seleccionar:

The screenshot shows the Postman interface with a GET request to `http://localhost/api/tareas/2`. The 'Auth' tab is active, set to 'Bearer Token' with the value `2|qj8WYtWPvxkRm6GbGJjWavC...`. The response is a 200 OK status with the following JSON data:

```

1 {
2   "data": {
3     "id": 2,
4     "titulo": "Titulo: Limpiar habitaciones",
5     "descripcion": "Descripcion: Limpiar exhaustivamente el WC\n\ny poner lavadora.",
6     "etiquetas": "Etiqueta: []"
7   }
8 }

```

- El logout del usuario: usamos el método GET, la url, y añadimos la autenticación con el token.

The screenshot shows the Postman interface with a GET request to `http://localhost/api/logout`. The 'Authorization' tab is active, set to 'Bearer Token' with the value `2|T5d0jYiZnvVkJNdsIJ1qwNFQlueCi9OzQ6S...`. The response is a 200 OK status with the following JSON data:

```

1 {
2   "message": "Session cerrada correctamente"
3 }

```

9. TEST

Por últimos tendríamos que verificar que todo ha salido correctamente, para ellos crearemos los test necesarios, para hacer la comprobación:

- Primero TareaTest: **sail artisan make:test TareaTest**

```
public function test_get_tareas_all(): void
{
    $user = User::factory()->create();
    $tarea = new Tarea();
    $tarea->titulo = "testTitulo";
    $tarea->descripcion = "testDescripcion";
    $tarea->save();

    $response = $this->actingAs($user)->getJson('api/tareas');
    $response->assertStatus(200);

    $response->assertJsonFragment([
        'id' => $tarea->id,
        'titulo' => 'Titulo: ' . $tarea->titulo,
        'descripcion' => 'Descripcion: ' . $tarea->descripcion,
        'etiquetas' => 'Etiqueta: []'
    ]);
}
```

```
public function test_get_una_tarea(): void
{
    $user = User::factory()->create();
    $tarea = new Tarea();
    $tarea->titulo = "testTitulo";
    $tarea->descripcion = "testDescripcion";
    $tarea->save();

    $response = $this->actingAs($user)->getJson('api/tareas/' . $tarea->id);
    $response->assertStatus(200);

    $response->assertJsonFragment([
        'id' => $tarea->id,
        'titulo' => 'Titulo: ' . $tarea->titulo,
        'descripcion' => 'Descripcion: ' . $tarea->descripcion,
        'etiquetas' => 'Etiqueta: []'
    ]);
}
```

- Luego el de EtiquetaTest: **sail artisan make:test EtiquetaTest**

```
public function test_get_etiquetas_all(): void
{
    $user = User::factory()->create();
    $etiqueta = new Etiqueta();
    $etiqueta->nombre = "testNombre";
    $etiqueta->save();

    $response = $this->actingAs($user)->getJson('api/etiquetas');
    $response->assertStatus(200);

    $response->assertJsonFragment([
        'id' => $etiqueta->id,
        'nombre' => 'Nombre: ' . $etiqueta->nombre
    ]);
}

public function test_get_una_etiqueta(): void
{
    $user = User::factory()->create();
    $etiqueta = new Etiqueta();
    $etiqueta->nombre = "testNombre";
    $etiqueta->save();

    $response = $this->actingAs($user)->getJson('api/etiquetas/' . $etiqueta->id);
    $response->assertStatus(200);

    $response->assertJsonFragment([
        'id' => $etiqueta->id,
        'nombre' => 'Nombre: ' . $etiqueta->nombre
    ]);
}
```

- Y por último UsuarioTest

```
public function test_registro(): void
{
    $user = [
        'name' => 'test',
        'email' => 'test@test',
        'password' => 'test'
    ];

    $response = $this->postJson('api/registro', $user);
    $response->assertStatus(200)->assertJsonStructure([
        'data',
        'access_token',
        'token_type'
    ]);
}
```

```
public function test_login(): void
{
    $user = User::factory()->create();
    $login = [
        'email' => $user->email,
        'password' => 'password'
    ];

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

```
0 references | 0 overrides | Codeium: Refactor | Explain | Generate Function Comment | X
public function test_logout(): void
{
    $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);
}
```

Para ejecutar los test usamos el siguiente comando:

- sail test

```
arkangel@Arkangel apiTaskLA % sail test

PASS Tests\Feature\EtiquetaTest
✓ get etiquetas all                                0.81s
✓ get una etiqueta                                0.02s

PASS Tests\Feature\ExampleTest
✓ the application returns a successful response    0.04s

PASS Tests\Feature\TareaTest
✓ get tareas all                                    0.02s
✓ get una tarea                                    0.01s

FAIL Tests\Feature\UsuarioTest
✗ registro                                         0.07s
✓ login                                            0.05s
✓ logout                                           0.01s
```