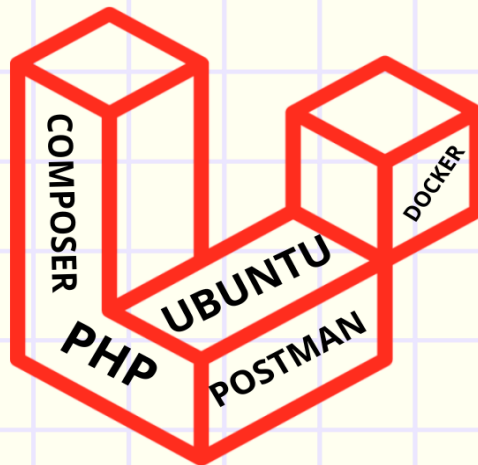


PROYECTO DE TAREAS

EN LARAVEL



Alumno: Steven Cubillos García

Profesora: Alicia Vega Moreno

Módulo: Desarrollo web en entorno servidor



ÍNDICE

1. Introducción	pg 3
2. Creación del proyecto	pg 3
3. Creación de las tablas y Seeders	pg 7
4. Creación de los modelos y controladores	pg 13
5. Creación de Requests	pg 20
6. Creación de Resources	pg 21
7. Creación de login con API y controlador AuthController	pg 23
8. Uso de Postman	pg 25
9. Creación de tests	pg 40



1.Introducción

Para empezar, Laravel es un framework que sirve para desarrollar apps/webs con PHP, esto hace que sea más fácil crear aplicaciones web al proporcionar herramientas que simplifican el proceso. Esto ayuda a los desarrolladores a escribir código de manera ordenada y fácil de manejar. Gracias a estas ventajas, Laravel se ha vuelto muy popular, atrayendo a muchos desarrolladores que lo prefieren como su marco de trabajo principal para un desarrollo eficiente, sobre todo para el campo del backend. Es por eso que esta guía va a mostrar el proceso de un proyecto de gestión de tareas que usaremos a través de la API Postman.

2.Creación del proyecto

Para empezar abrimos nuestra consola de Ubuntu y la app de Docker, dentro de la consola Ubuntu, nos vamos a dirigir a nuestra carpeta donde están el resto de proyectos que se levanten con el Docker, en nuestro caso, `www`

```
steven@Revision-PC: /mnt/c/ × + v
Welcome to Ubuntu 22.04.3 LTS (GNU/Linux 5.15.133.1-microsoft-standard-WSL2 x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/pro

 * Strictly confined Kubernetes makes edge and IoT secure. Learn how MicroK8s
   just raised the bar for easy, resilient and secure K8s cluster deployment.

https://ubuntu.com/engage/secure-kubernetes-at-the-edge

This message is shown once a day. To disable it please create the
/home/steven/.hushlogin file.
steven@Revision-PC:~$ cd ..
steven@Revision-PC:/home$ ls -l
total 4
drwxr-x--- 9 steven steven 4096 Feb  1 17:34 steven
steven@Revision-PC:/home$ cd
steven@Revision-PC:~$ ls -l
total 4
drwxr-xr-x 2 steven steven 4096 Oct  3 19:40 ue
steven@Revision-PC:~$ cd ..
steven@Revision-PC:/home$ ls -l
total 4
drwxr-x--- 9 steven steven 4096 Feb  1 17:34 steven
steven@Revision-PC:/home$ cd ..
steven@Revision-PC:/$ ls -l
```

```
steven@Revision-PC:/$ cd mnt
steven@Revision-PC:/mnt$ cd c
steven@Revision-PC:/mnt/c$ Users
```

```
steven@Revision-PC:~$ cd ..
steven@Revision-PC:/home$ cd ..
steven@Revision-PC:/$ cd mnt
steven@Revision-PC:/mnt$ cd c/Users/Cubillos
steven@Revision-PC:/mnt/c/Users/Cubillos$ cd Desktop/servidor
steven@Revision-PC:/mnt/c/Users/Cubillos/Desktop/servidor$ cd "docker compose"
steven@Revision-PC:/mnt/c/Users/Cubillos/Desktop/servidor/docker compose$ cd www
steven@Revision-PC:/mnt/c/Users/Cubillos/Desktop/servidor/docker compose/www/api_tarea$ composer create-project --prefer-dist laravel/laravel apiTarea
```

Dentro de la carpeta, usamos el comando **composer create-project --prefer-dist laravel/laravel apiTarea** para crear nuestro proyecto generando una serie de carpetas.

Después de la instalacion, usamos los comandos **php artisan sail:install** y

npm install para instalar composer y npm dentro de la carpeta del proyecto.

```
- Installing spatie/laravel-ignition (2.4.2): Extracting archive
48 package suggestions were added by new dependencies, use 'composer suggest' to see details.
Generating optimized autoload files
> Illuminate\Foundation\ComposerScripts::postAutoloadDump
> @php artisan package:discover --ansi

  INFO  Discovering packages.

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

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.

steven@Revision-PC:/mnt/c/Users/Cubillos/Desktop/servidor/docker compose/www/api_tarea$ |
```

Ya hecho el proceso, vamos a ejecutar estos comandos: **alias sail=./vendor/bin/sail**, crea un alias que apunta a un ejecutable, y **sail up -d** que crea(en caso de que no lo está) y ejecuta el contenedor de Docker necesario para correr nuestro proyecto.

```
steven@Revision-PC:/mnt/c/Users/Cubillos/Desktop/servidor/docker compose/www/apiTarea$ alias sail=./vendor/bin/sail
steven@Revision-PC:/mnt/c/Users/Cubillos/Desktop/servidor/docker compose/www/apiTarea$ sail up -d
[+] Running 4/4
 ✓ Network apitarea_sail          Created
 ✓ Volume "apitarea_sail-mysql"   Created
 ✓ Container apitarea-mysql-1     Started
 ✓ Container apitarea-laravel.test-1 Started
steven@Revision-PC:/mnt/c/Users/Cubillos/Desktop/servidor/docker compose/www/apiTarea$ |
```

Y si vamos a nuestro programa de Docker, comprobamos que está en funcionamiento:

BETA

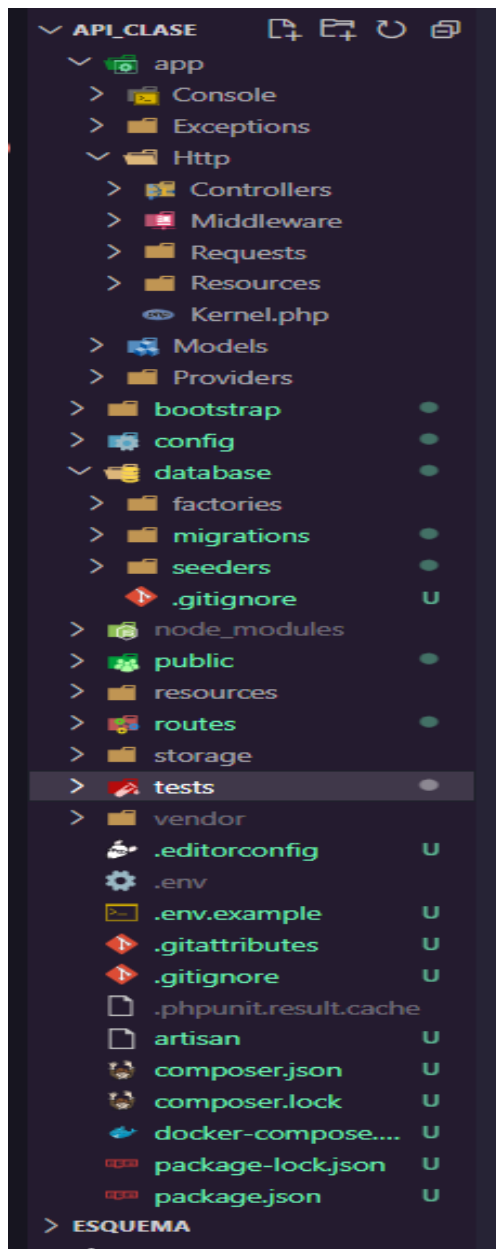
Search

Only show running containers

	Name	Image	Status	CPU (%)	Port(s)	Last started	Actions
<input type="checkbox"/>	apitarea		Running (2/2)	0.56%		1 hour ago	<input type="checkbox"/> ⋮ <input type="trash"/>
<input type="checkbox"/>	laravel.test b02afb9e484	sail-8.3/app	Running	0.07%	5173:5173 ↗ Show all ports (2)	1 hour ago	<input type="checkbox"/> ⋮ <input type="trash"/>
<input type="checkbox"/>	mysql-1 55d499c3e6f	mysql/mysql-server:8.0	Running	0.49%	3306:3306 ↗	1 hour ago	<input type="checkbox"/> ⋮ <input type="trash"/>

Showing 5 items

Por otra, si miramos la carpeta de **www** (nuestra ubicación para que se cree nuestro proyecto) nos mostrará el proyecto de Laravel, de forma que se nos muestra estructurado de la siguiente forma:



3.Creación de las tablas y seeders

Aquí vamos a proceder a crear nuestras tablas de tareas y etiquetas, para ello, usamos los siguientes comandos:

```
steven@Revision-PC: /mnt/c/Users/Cubillos/Desktop/servidor/docker compose/www/apiTarea$ sail artisan make:migration create_tareas_table
INFO Migration [database/migrations/2024_02_27_231022_create_tareas_table.php] created successfully.
steven@Revision-PC: /mnt/c/Users/Cubillos/Desktop/servidor/docker compose/www/apiTarea$ sail artisan make:migration create_etiquetas_table
INFO Migration [database/migrations/2024_02_27_231106_create_etiquetas_table.php] created successfully.
```

se nos crea unos archivos ubicado en `apiTareas/database/migrations` donde nos establece una base para definir nuestra tarea, en este caso, de una **id**, **título** y **descripción**:

```
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('tareas', function (Blueprint $table) {
            $table->id();
            $table->string('nombre',40);
            $table->string('descripcion')->nullable();
            $table->timestamps();
        });
    }

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

Y para redefinirla, usamos el comando **sail artisan migrate** en la consola.

Ahora si queremos introducir datos, tenemos que usar un seeder con el comando **sail artisan make:seeder TareaSeeder**

```
steven@Revision-PC:/mnt/c/Users/Cubillos/desktop/servidor/docker compose/www/apiTarea$ sail artisan make:seeder TareaSeeder
```

En el **Seeder** vamos a incluir datos a inyectar:


```

namespace Database\Seeders;

use Illuminate\Database\Console\Seeds\WithoutModelEvents;
use Illuminate\Support\Facades\DB;
use Illuminate\Database\Seeder;

class TareaSeeder extends Seeder
{
    /**
     * Run the database seeds.
     */
    public function run(): void
    {
        DB::table('tareas')->insert([
            'nombre'=>'Quimica',
            'descripcion'=>'Estudiar para examen final'
        ]);

        DB::table('tareas')->insert([
            'nombre'=>'Lengua',
            'descripcion'=>'Terminar seis analisis morfologicos'
        ]);

        DB::table('tareas')->insert([
            'nombre'=>'Perro',
            'descripcion'=>'Sacar a pasear al perro'
        ]);

        DB::table('tareas')->insert([
            'nombre'=>'Cumpleaños',
            'descripcion'=>'Darle regalo a mi amigo en casa'
        ]);
    }
}

```

Por otra parte, en el caso de **etiquetas** quedaría algo similar a esto:

```

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('etiquetas', function (Blueprint $table) {
            $table->id();
            $table->string('nombre' ,15);
            $table->timestamps();
        });
    }

    public function down(): void
    {
        Schema::dropIfExists('etiquetas');
    }
};

```

Y para su inserción de datos usamos el comando el comando **sail artisan make:seeder EtiquetaSeeder**

```
e/www/apiTarea$ sail artisan make:seeder EtiquetaSeeder
```

Incluyendo los siguientes datos:

```

use Illuminate\Database\Console\Seeds\WithoutModelEvents;
use Illuminate\Database\Seeder;
use Illuminate\Support\Facades\DB;

class EtiquetaSeeder extends Seeder
{
    /**
     * Run the database seeds.
     */
    public function run(): void
    {
        DB::table('etiquetas')->insert([
            'nombre'=>'Instituto',
        ]);

        DB::table('etiquetas')->insert([
            'nombre'=>'Casa',
        ]);
    }
}

```

Llamamos a ambos seeders en **DatabaseSeeder** para que funcione a la hora de migrar datos

```

<?php

namespace Database\Seeders;

// use Illuminate\Database\Console\Seeds\WithoutModelEvents;
use Illuminate\Database\Seeder;

class DatabaseSeeder extends Seeder
{
    /**
     * Seed the application's database.
     */
    public function run(): void
    {
        $this->call([TareaSeeder::class, EtiquetaSeeder::class]);
    }
}

```

En nuestro caso, la relación que hay entre tarea y etiqueta es muchos a muchos hace que se cree una nueva tabla llamada tareas_etiquetas donde se almacena las ids de ambas, usaremos el comando **sail artisan make:migration create_tareas_etiquetas_table**

```
'apiTarea$ sail artisan make:migration create_tareas_etiquetas_table|
```

Maquetando de esta forma:

```
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('tareas_etiquetas', function (Blueprint $table) {
            $table->id();
            $table->foreignId('tarea_id')->constrained();
            $table->foreignId('etiqueta_id')->constrained();
            $table->timestamps();
        });
    }

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

Y recuerda que en el archivo tarea.php le debemos especificar la relacion muchos a muchos con etiquetas a través de `public function etiquetas(): BelongsToMany`

```
public function etiquetas(): BelongsToMany{

    return $this->belongsToMany(Etiqueta::class, 'tareas_etiquetas', 'tarea_id', 'etiqueta_id');

}
```

Y ya para la migración de datos que hemos puesto usamos el comando **sail artisan db:seed**.

```
steven@Revision-PC:/mnt/c/Users/Cubillos/Desktop/servidor/docker compose/www/apiTarea$ sail artisan db:seed

INFO Seeding database.
Database\Seeders\TareaSeeder ..... RUNNING
Database\Seeders\TareaSeeder ..... 270 ms DONE
Database\Seeders\EtiquetaSeeder ..... RUNNING
Database\Seeders\EtiquetaSeeder ..... 15 ms DONE
steven@Revision-PC:/mnt/c/Users/Cubillos/Desktop/servidor/docker compose/www/apiTarea$ |
```

4.Creación de los modelos y controladores

Después de crear las tablas, vamos a crear sus modelos respectivos, lo crearemos con los comandos **sail artisan make:model Tarea/Etiqueta (una de las dos opciones) -cr**. A su vez, crearemos sus controladores gracias a **-cr** en una misma línea.

```
steven@Revision-PC:/mnt/c/Users/Cubillos/Desktop/servidor/docker compose/www/apiTarea$ sail artisan make:model Tarea -cr
INFO Model [app/Models/Tarea.php] created successfully.
INFO Controller [app/Http/Controllers/TareaController.php] created successfully.
steven@Revision-PC:/mnt/c/Users/Cubillos/Desktop/servidor/docker compose/www/apiTarea$ sail artisan make:model Etiqueta -cr
INFO Model [app/Models/Etiqueta.php] created successfully.
INFO Controller [app/Http/Controllers/EtiquetaController.php] created successfully.
steven@Revision-PC:/mnt/c/Users/Cubillos/Desktop/servidor/docker compose/www/apiTarea$ |
```

Si vamos a nuestro archivo Tarea.php, lo podemos moldear de esta forma, que el array oculto se llene con **created_at** y **updated_at** y con la propiedad **guarded**, además de asignarle una función para definir la relación muchos a muchos con etiqueta.

```
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     /*
15     protected $fillable = ["nombre", "descripcion"]; */
16
17     protected $hidden = ["created_at", "updated_at"];
18     // protected $table = 'tareas';
19
20     public function etiquetas(): BelongsToMany{
21         //categoria belongs to many porque un producto
22         return $this->belongsToMany(Etiqueta::class, 'tareas_etiquetas', 'tarea_id', 'etiqueta_id');
23     }
24 }
25 }
26
```

En cuanto a su controlador quedaría de esta forma, definiendo su index, store, almacenamiento, entre otros

```

namespace App\Http\Controllers;

use App\Models\Tarea;
use Illuminate\Http\Request;
use Illuminate\Http\Resources\Json\JsonResource;
use App\Http\Resources\TareaResource;
use App\Http\Requests\TareaRequest;

class TareaController extends Controller
{
    /**
     * Display a listing of the resource.
     */
    public function index():JsonResource
    {
        $tarea = Tarea::all();
        return TareaResource::collection($tarea);
    }

    /**
     * Show the form for creating a new resource.
     */
    public function create()
    {
        //
    }

    /**
     * Store a newly created resource in storage.
     */
    public function store(TareaRequest $request):JsonResource
    {
        $tarea=new Tarea();
        $tarea->nombre = $request->nombre;
        $tarea->descripcion = $request->descripcion;
        $tarea->save();
    }
}

```

```

        $tarea->etiquetas()->attach($etiquetas);
        return response()->json($tarea,201);
    }

    /**
     * Display the specified resource.
     */
    public function show(Tarea $tarea)
    {
        $tarea = Producto::find($id);

        //return response()->json($producto,200);
        return new TareaResource($tarea);
    }

    /**
     * Show the form for editing the specified resource.
     */
    // public function edit(Tarea $tarea)
    // {
    //     //
    // }

    /**
     * Update the specified resource in storage.
     */
    public function update(TareaRequest $request, $id)
    {
        $tarea = Tarea::find($id);
        $tarea->nombre = $request->nombre;
        $tarea->descripcion = $request->descripcion;

        $tarea->save();

        //return response()->json($producto,200);
        $tarea->etiquetas()->attach($request->etiquetas);
        $tarea->save();
        return new TareaResource($tarea);
    }

```

```

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

```

Y para que funcione, es necesario añadir la ruta de nuestro controlador en el archivo **api** con esta línea:

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


```
19
20 Route::resource('/tareas', TareaController::class);
```

Ahora, vamos a hacer lo mismo que hicimos en el modelo de tareas pero aplicando a **Etiquetas**, moldeamos el modelo de forma que quede así:

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Etiqueta extends Model
{
    use HasFactory;

    protected $guarded = [];
    protected $hidden = ["created_at", "updated_at"];
}
```

Y su controlador lo definimos de manera similar como el de tareas,

```
namespace App\Http\Controllers;

use App\Http\Resources\EtiquetaResource;
use App\Http\Requests\EtiquetaRequest;

use App\Models\Etiqueta;
use Illuminate\Http\Request;
use Illuminate\Http\Resources\Json\JsonResource;

class EtiquetaController extends Controller
{
    /**
     * Display a listing of the resource.
     */
    public function index()
    {
        $etiq=Etiqueta::all();
        return EtiquetaResource::collection($etiq);
    }

    /**
     * Show the form for creating a new resource.
     */
    public function create()
    {
        //
    }

    /**
     * Store a newly created resource in storage.
     */
    public function store(EtiquetaRequest $request)
    {
        $etiq = new Etiqueta();
        $etiq->nombre = $request->nombre;
        $etiq->save();
    }
}
```

```

        return new EtiquetaResource($etiq);
    }

    /**
     * Display the specified resource.
     */
    public function show($id):JsonResource
    {
        $etiq = Etiqueta::find($id);
        return new EtiquetaResource($etiq);
    }

    /**
     * Show the form for editing the specified resource.
     */
    public function edit(Etiqueta $etiqueta)
    {
        //
    }

    /**
     * Update the specified resource in storage.
     */
    public function update(Request $request, $id):JsonResource
    {
        $etiq = Etiqueta::find($id);
        $etiq->nombre = $request->nombre;
        $etiq->save();

        return new EtiquetaResource($etiq);
    }

```

```

    public function destroy(Etiqueta $etiqueta)
    {
        $etiqueta->delete();
        return response()->json(['success' => true], 200);
    }

```

Su ruta en la **api**:

```

Route::resource('/etiquetas', EtiquetaController::class);

```

5.Creación de Requests

En esta parte vamos a crear requests, que son controladores en el que recibimos un objeto como Request de parámetro para acceder a datos de una solicitud, así que para empezar usamos el comando **sail artisan make:request TareaRequest**, así lo mismo para etiqueta

```
steven@Revision-PC:/mnt/c/Users/Cubillos/Desktop/servidor/docker compose/www/apiTarea$ sail artisan make:request TareaRequest
INFO Request [app/Http/Requests/TareaRequest.php] created successfully.
steven@Revision-PC:/mnt/c/Users/Cubillos/Desktop/servidor/docker compose/www/apiTarea$ sail artisan make:request EtiquetaRequest
INFO Request [app/Http/Requests/EtiquetaRequest.php] created successfully.
steven@Revision-PC:/mnt/c/Users/Cubillos/Desktop/servidor/docker compose/www/apiTarea$ sail artisan make:request UsuarioRequest
INFO Request [app/Http/Requests/UsuarioRequest.php] created successfully.
steven@Revision-PC:/mnt/c/Users/Cubillos/Desktop/servidor/docker compose/www/apiTarea$ |
```

Dentro de **TareaRequest**, tenemos que autorizar la función `authorize` en `true` y añadimos lo que se pedirá en el request.

```
namespace App\Http\Requests;

use Illuminate\Foundation\Http\FormRequest;

class TareaRequest extends FormRequest
{
    /**
     * Determine if the user is authorized to make this request
     */
    public function authorize(): bool
    {
        return true;
    }

    /**
     * Get the validation rules that apply to the request.
     *
     * @return array<string, \Illuminate\Contracts\Validation\Rule>
     */
    public function rules(): array
    {
        return [
            'nombre' => 'required|max:40|min:3',
            'descripcion' => 'nullable|max:255|min:3'
        ];
    }
}
```

Y en **EtiquetaRequest** quedaría así:

```

namespace App\Http\Requests;

use Illuminate\Foundation\Http\FormRequest;

class EtiquetaRequest extends FormRequest
{
    /**
     * Determine if the user is authorized to make this request.
     */
    public function authorize(): bool
    {
        return true;
    }

    /**
     * Get the validation rules that apply to the request.
     *
     * @return array<string, \Illuminate\Contracts\Validation\Rule>
     */
    public function rules(): array
    {
        return [
            'nombre' => 'required|max:40|min:3',
        ];
    }
}

```

6.Creación de Resources

Para crear nuestros archivos **Resource** usamos el comando **sail artisan make:resource TareaResource** (o **EtiquetaResource** en el caso de las etiquetas).

```
apiTarea$ sail artisan make:resource TareaResource
```

```
apiTarea$ sail artisan make:resource EtiquetaResource
```

Nos crearía estos archivos y lo definiremos como se muestra a continuación:

```

<?php

namespace App\Http\Resources;

use Illuminate\Http\Request;
use Illuminate\Http\Resources\Json\JsonResource;

class TareaResource extends JsonResource
{
    /**
     * Transform the resource into an array.
     *
     * @return array<string, mixed>
     */
    public function toArray(Request $request): array
    {
        // return parent::toArray($request);

        return [
            'id' => $this->id,
            'nombre' => 'Nombre: ' . $this->nombre,
            'descripcion' => 'Desc: ' . $this->descripcion,
            'algo' => 'tortilla',
            'etiqueta' => $this->etiquetas != null && $this->etiquetas->isNotEmpty() ?
                $this->etiquetas->pluck('nombre') : 'No hay etiquetas asociadas'
        ];
    }
}

```

```

namespace App\Http\Resources;

use Illuminate\Http\Request;
use Illuminate\Http\Resources\Json\JsonResource;

class EtiquetaResource extends JsonResource
{
    /**
     * Transform the resource into an array.
     *
     * @return array<string, mixed>
     */
    public function toArray(Request $request): array
    {
        // return parent::toArray($request);

        return [
            'id' => $this->id,
            'nombre' => 'Nombre: ' . $this->nombre,
            'Pizza' => 'Pineapple: ',
        ];
    }
}

```

7.Creación de login con API y controlador AuthController

Ahora nos toca crear nuestro login para cargar nuestra cuenta, para ello, vamos a usar **laravel/sactum** con el comando **sail composer require laravel/sactum**

```
steven@Revision-PC:/mnt/c/Users/Cubillos/Desktop/servidor/docker compose/www/apiTarea$ 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
steven@Revision-PC:/mnt/c/Users/Cubillos/Desktop/servidor/docker compose/www/apiTarea$
```

Después lo configuramos para nuestro proyecto en concreto con el comando **sail artisan vendor:publish --provider="Laravel\Sanctum\SanctumServiceProvider"**

```
steven@Revision-PC:/mnt/c/Users/Cubillos/Desktop/servidor/docker compose/www/apiTarea$ sail artisan vendor:publish --provider="Laravel\Sanctum\SanctumService
eProvider"

 INFO  Publishing assets.

Copying directory [vendor/laravel/sanctum/database/migrations] to [database/migrations] ..... DONE
File [config/sanctum.php] already exists ..... SKIPPED
steven@Revision-PC:/mnt/c/Users/Cubillos/Desktop/servidor/docker compose/www/apiTarea$
```

Ya hecho esta parte, vamos a crearnos un controlador para el login, registro y cierre de sesión, con el comando **sail artisan make:controller AuthController**, creandonos un archivo que se llame **AuthController.php**, debería quedar definido de esta forma:

```

use Illuminate\Http\Request;
use App\Models\User;
use Illuminate\Support\Facades\Auth;
use Illuminate\Support\Facades\Hash;

class AuthController extends Controller
{
    //
    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 logout(){
        auth()->user()->tokens()->delete();
        return ['message' =>'La sesión se ha cerrado correctamente'];
    }

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

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

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

```

Así como su ruta en **api.php** para su funcionamiento:


```
Route::resource('/tareass', TareaController::class);
Route::resource('/etiquetas', EtiquetaController::class);

Route::post('register', [AuthController::class, 'register']);
Route::post('login', [AuthController::class, 'login']);

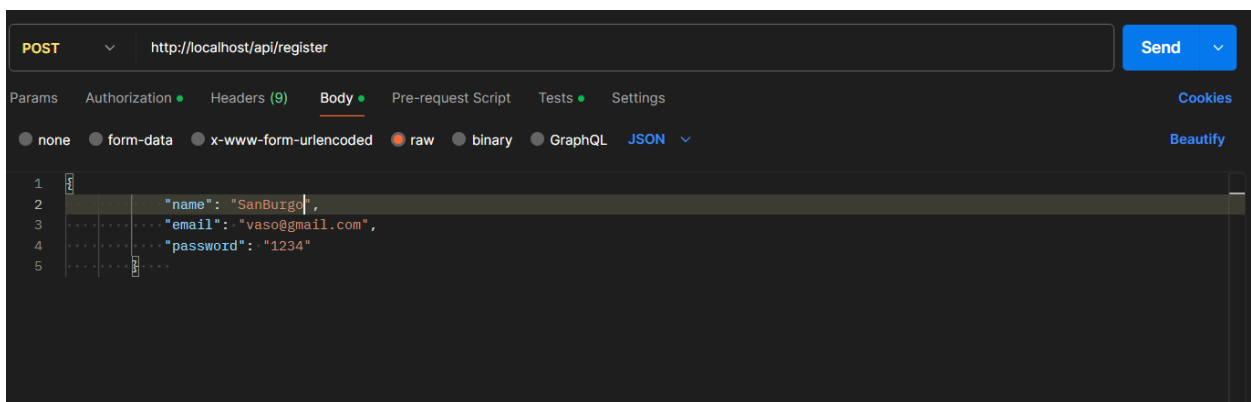
Route::middleware('auth:sanctum')->group(function(){
    Route::get('logout', [AuthController::class, 'logout']);
});
```

8. Uso de Postman

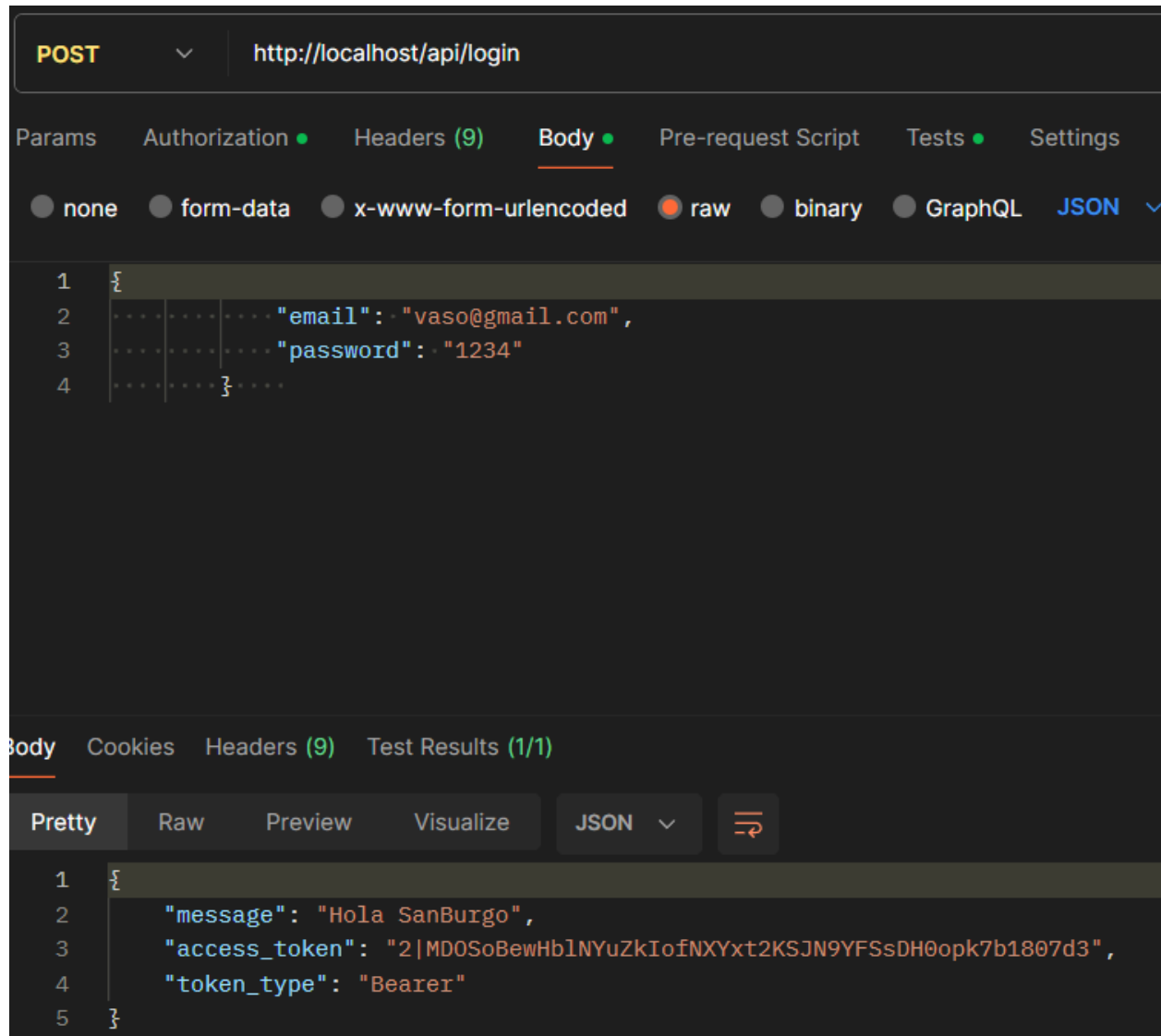
ya para comprobar que nuestro proyecto en laravel funcione, vamos a usar **Postman** como herramienta, actuando como una API.

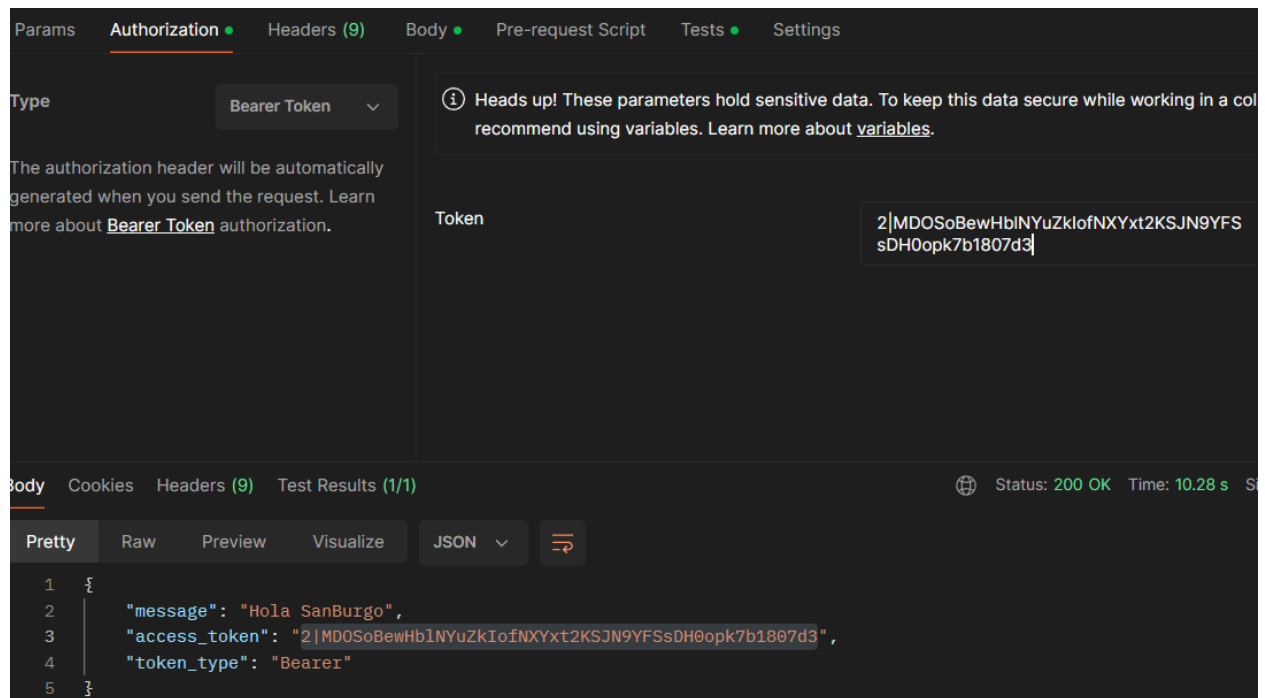
Dentro de ella, vamos a hacer un post en el register para registrar nuestro usuario:

Registro



Hacemos lo mismo pero con el login, introduciendo nuestro email ya través de post también, nos dará un access_token que debemos introducirlo en Authorization





A partir de aquí se hará una serie de acciones según la tabla de **tareas** o **etiquetas**, para manipular datos:

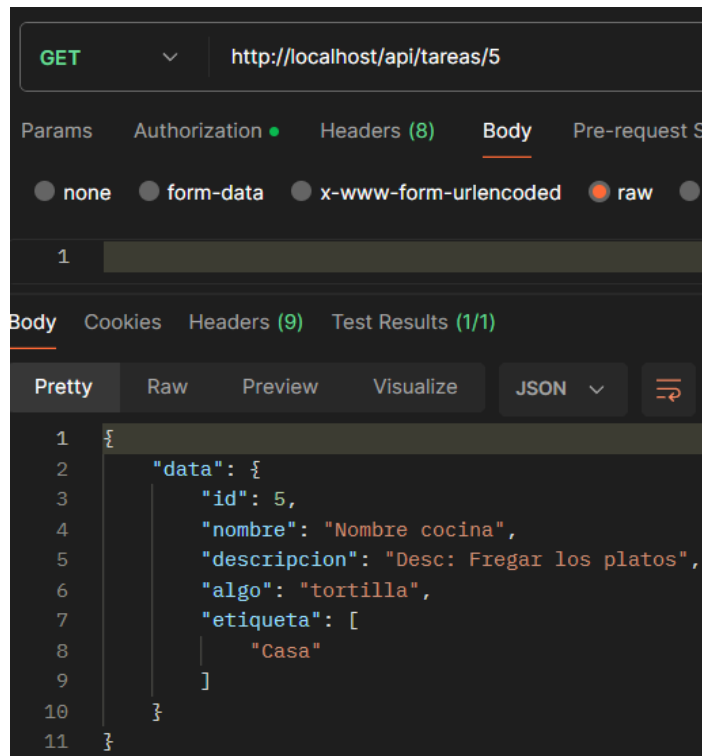
TAREAS:

Obtener tareas:

The screenshot shows a REST client interface with a GET request to `http://localhost/api/tareas`. The 'Body' tab is selected, showing a 'raw' body type. The response is displayed in the 'Pretty' view as JSON. The JSON structure is as follows:

```
1 {
2   "data": [
3     {
4       "id": 1,
5       "nombre": "Nombre Quimica",
6       "descripcion": "Desc: Estudiar para examen final",
7       "algo": "tortilla",
8       "etiqueta": "No hay etiquetas asociadas"
9     },
10    {
11      "id": 2,
12      "nombre": "Nombre Lengua",
13      "descripcion": "Desc: Terminar seis analisis morfologicos",
14      "algo": "tortilla",
15      "etiqueta": "No hay etiquetas asociadas"
16    },
17  ]
18 }
```

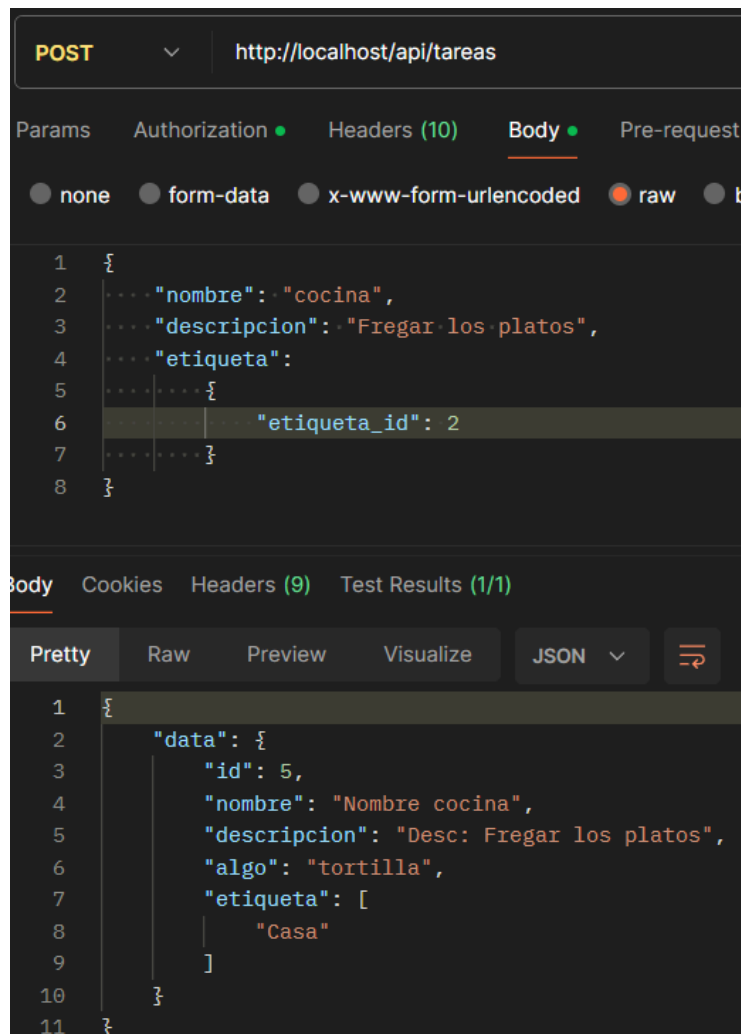
Obtener una tarea en concreto:



The screenshot shows a REST client interface. At the top, a GET request is configured for the URL `http://localhost/api/tareas/5`. Below the URL bar, there are tabs for Params, Authorization, Headers (8), Body, and Pre-request S. The Body tab is selected, and the format is set to raw. The response body is displayed in the Pretty view, showing a JSON object with the following structure:

```
1 {
2   "data": {
3     "id": 5,
4     "nombre": "Nombre cocina",
5     "descripcion": "Desc: Fregar los platos",
6     "algo": "tortilla",
7     "etiqueta": [
8       "Casa"
9     ]
10  }
11 }
```

Crear tarea:



Modificar tarea:

```
PUT http://localhost/api/tareas/2

Body
none form-data x-www-form-urlencoded raw

1 {
2   .... "id": 2,
3   .... "nombre": "Mates",
4   .... "descripcion": "Hacer fracciones",
5   .... "algo": "tortilla",
6   .... "etiqueta": []
7   .... {
8   ....   .... "etiquetas_id": 1
9   .... }
10 }
11 }
```

```
PUT http://localhost/api/tareas/2

Body
none form-data x-www-form-urlencoded raw

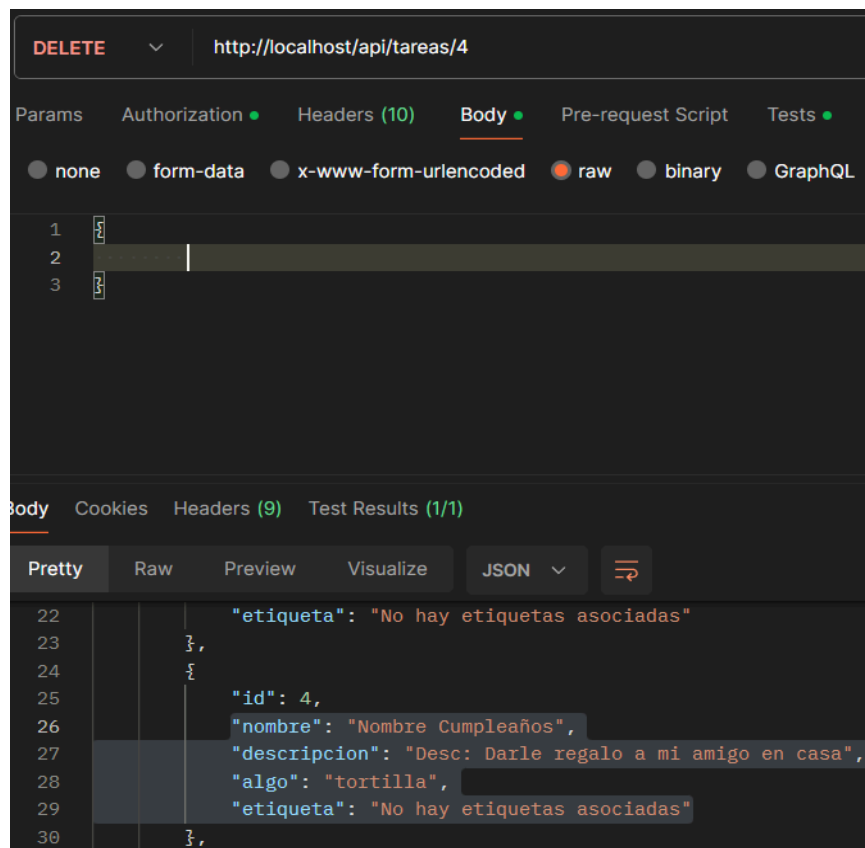
1 {
2   .... "id": 2,
3   .... "nombre": "Mates",
4   .... "descripcion": "Hacer fracciones",
5   .... "algo": "tortilla",
6   .... "etiqueta": [
7   ....   .... {
8   ....     .... "etiquetas_id": 1
9   ....   }
10  ]
11 }
```

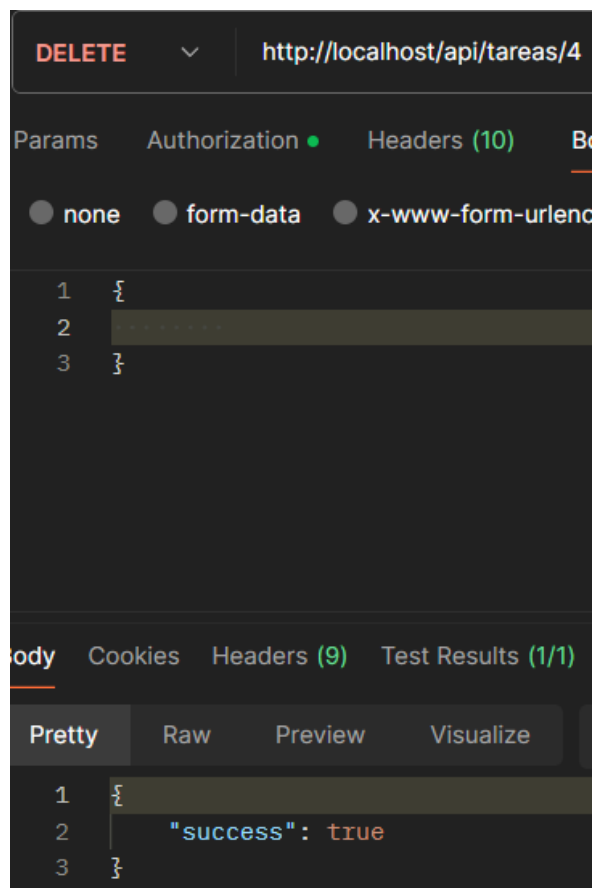
```
Body Cookies Headers (9) Test Results (1/1)

Pretty Raw Preview Visualize JSON

1 {
2   "data": {
3     "id": 2,
4     "nombre": "Nombre Mates",
5     "descripcion": "Desc: Hacer fracciones",
6     "algo": "tortilla",
7     "etiqueta": "No hay etiquetas asociadas"
8   }
9 }
```

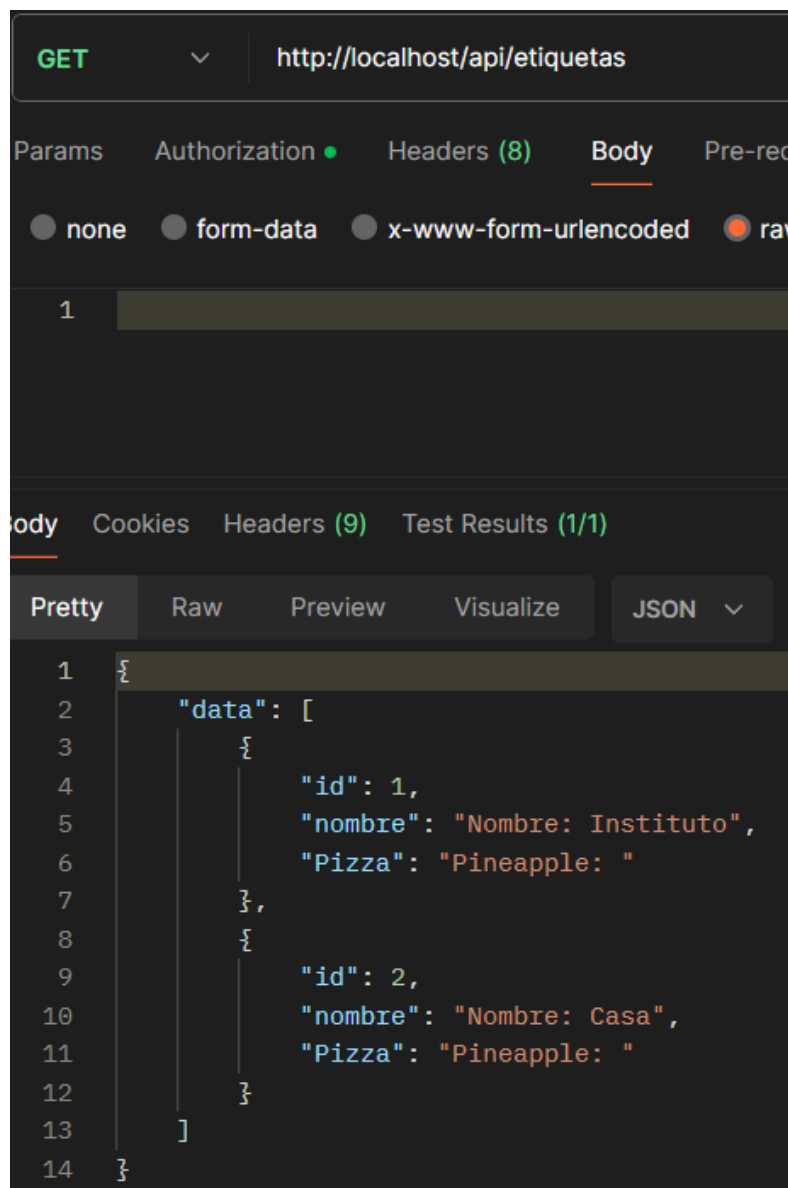
Borrar una tarea:



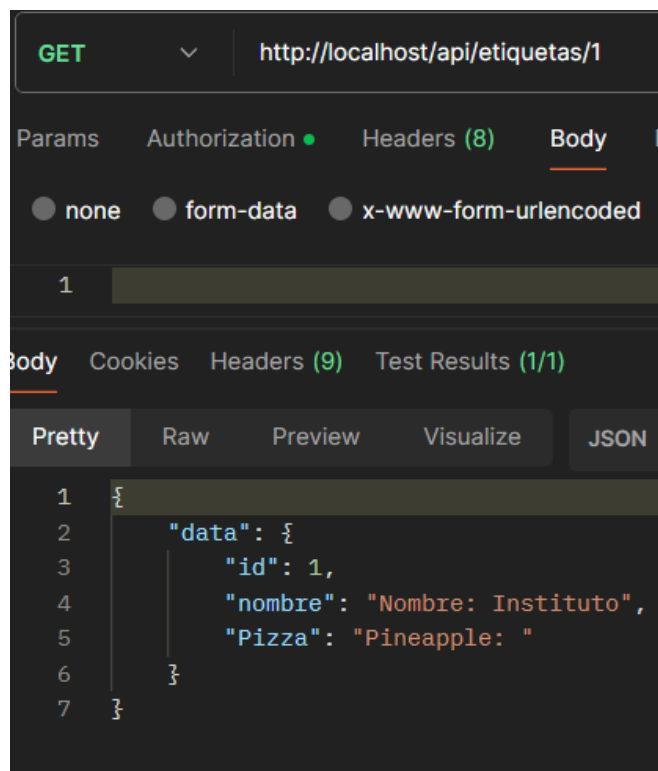


ETIQUETAS:

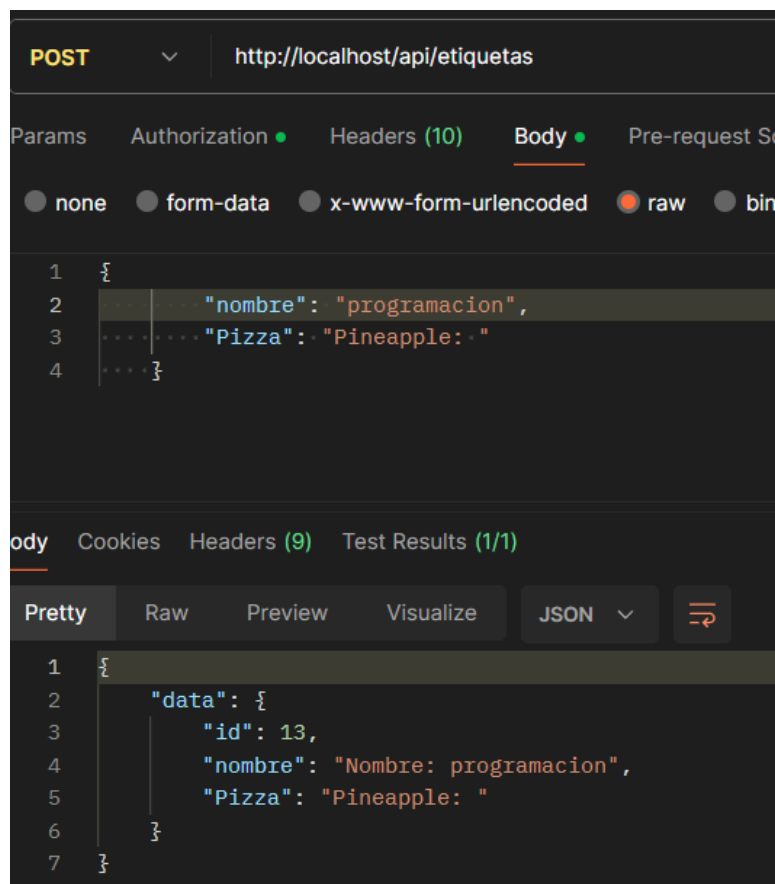
Obtener etiquetas:



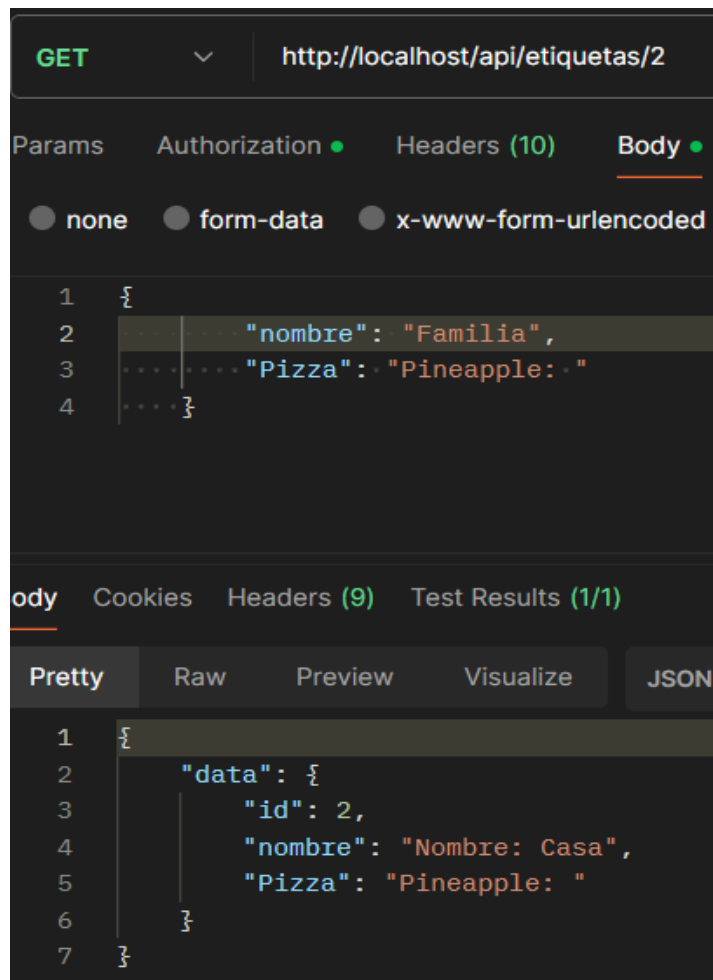
Obtener etiqueta en concreto:

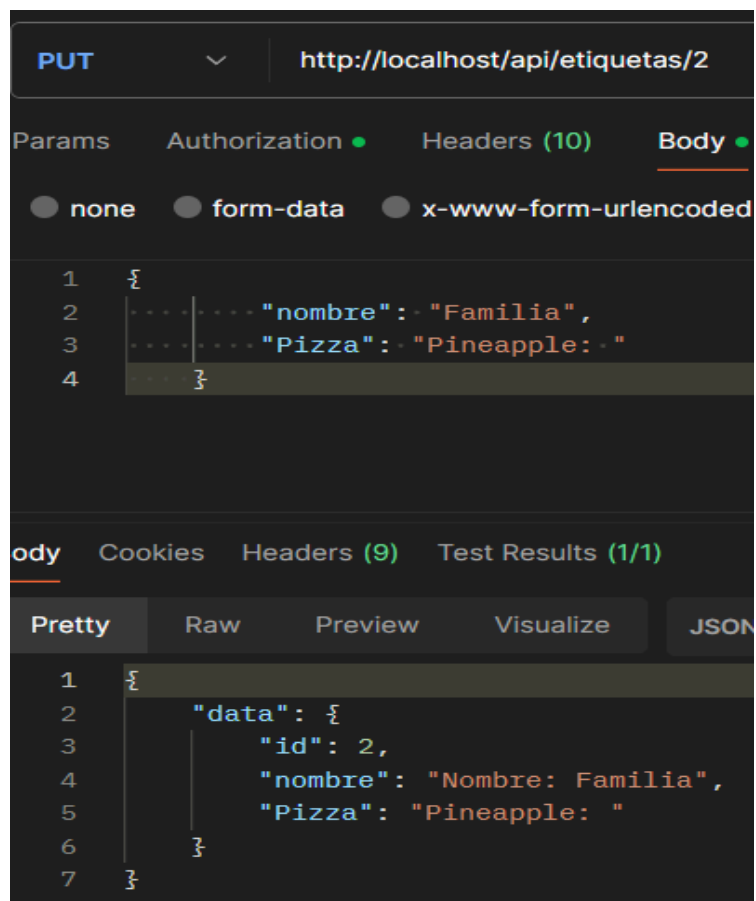


Crear etiqueta:

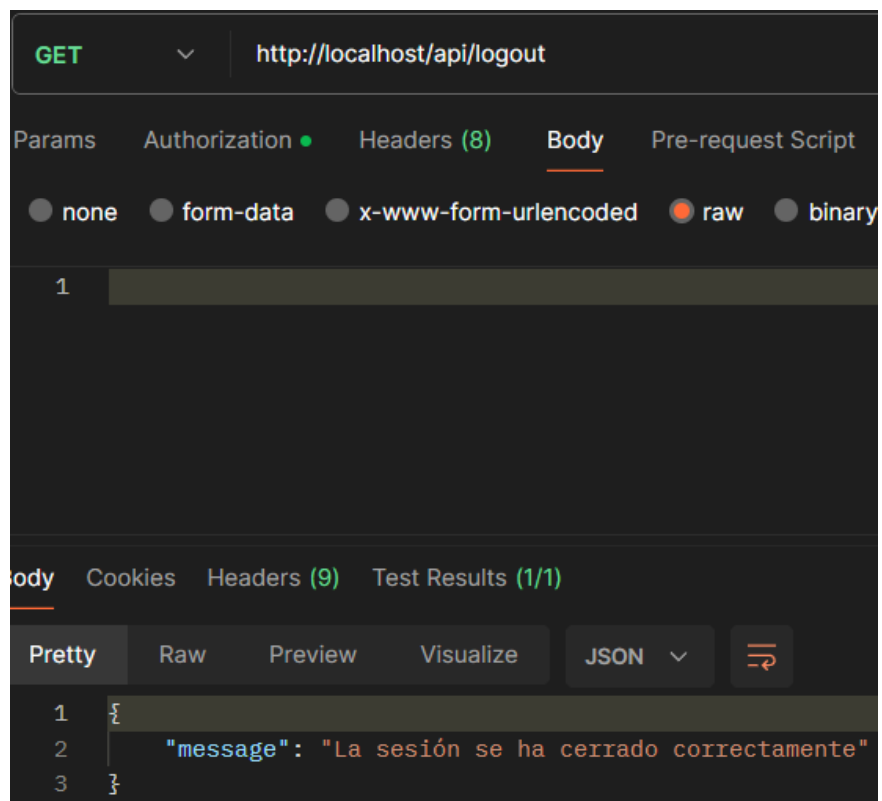


Modificar etiqueta:





Cerrar sesión:



9.Creación de tests

Ya definido nuestras tablas, modelos y otros elementos, vamos a crear nuestros test unitarios.

Primero, crearemos el de tareas con **sail artisan make:test tareasTest**

```
steven@Revision-PC:/mnt/c/Users/Cubillos/desktop/servidor/docker compose/www/apiTarea$ sail artisan make:test TareasTest
INFO Test [tests/Feature/TareasTest.php] created successfully.
steven@Revision-PC:/mnt/c/Users/Cubillos/desktop/servidor/docker compose/www/apiTarea$ |
```

Como dice el nombre, un test (dentro del contexto de Laravel), son pruebas que verifica el funcionamiento de diversas funciones, en el caso de tareas, he probado sus funciones de obtención, actualización, añadir, borrado,

El patron común que tienen es que el usuario lo creamos con **UserFactory**, un archivo con datos de modelo de usuario para generar datos de prueba ficticios, creamos una tarea con datos y realizamos una solicitud **HTTP GET** a la **API**, verificando también que el contenido sea igual en formato **JSON**.

Test:


```

class TareasTest extends TestCase
{
    use RefreshDatabase;
    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'):[]
        ]);
    }

    public function test_update_tarea()
    {

```

```

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

            // Creamos una tarea para que se actualize
            $tarea = new Tarea();
            $tarea->nombre = "TestTitulo";
            $tarea->descripcion = "TestDescripcion";
            $tarea->save();

            $updatedData = [
                'nombre' => 'NuevoTitulo',
                'descripcion' => 'NuevaDescripcion',
            ];

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

            $tarea->refresh();

            // Verificamos la actualizacion de la tarea en la BBDD
            $this->assertEquals($updatedData['nombre'], $tarea->nombre);
            $this->assertEquals($updatedData['descripcion'], $tarea->descripcion);
        }
    }
}

```

```

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

    // Creamos 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);

    // Verificamos la eliminacion de la tarea
    $this->assertNull(Tarea::find($tarea->id));
}

```

```

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

    $newTareaData = [
        'nombre' => 'NuevoTitl',
        'descripcion' => 'NuevaDescp',
    ];

    // Enviamos la solicitud de inserción
    $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'],
    ]);
}

```

En la consola, ejecutamos el test con `sail artisan test`, y vemos que ha pasado la prueba con éxito, en caso de que hubiese fallado la prueba, se nos notificará el posible “origen” del error dentro del código

```

steven@Revision-PC:/mnt/c/Users/Cubillos/Desktop/servidor/Docker compose/www/apiTarea$ sail artisan test

PASS Tests\Unit\ExampleTest
✓ that true is true

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

PASS Tests\Feature\TareasTest
✓ get tareas
✓ update tarea
✓ delete tarea
✓ insert tarea

Tests: 6 passed (11 assertions)
Duration: 12.25s

steven@Revision-PC:/mnt/c/Users/Cubillos/Desktop/servidor/Docker compose/www/apiTarea$ |

```

Login:

```

steven@Revision-PC:/mnt/c/Users/Cubillos/Desktop/servidor/Docker compose/www/apiTarea$ sail artisan make:test LoginTest

INFO Test [tests/Feature/LoginTest.php] created successfully.

steven@Revision-PC:/mnt/c/Users/Cubillos/Desktop/servidor/Docker compose/www/apiTarea$ |

```

```

namespace Tests\Feature;

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

class LoginTest extends TestCase
{
    use RefreshDatabase;
    /**
     * A basic feature test example.
     */
    public function test_register()
    {
        $user = [
            'name' => 'Santiago',
            'email' => 'vigo@gmail.com',
            'password' => '2222',
        ];

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

```

```

public function test_login()
{
    $user = User::create([
        'name' => 'michi',
        'email' => 'michigan@gmail.com',
        'password' => bcrypt('password'),
    ]);

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

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

```

Ejecutamos con sail artisan test

```

steven@Revision-PC:/mnt/c/Users/Cubillos/Desktop/servidor/Docker compose/www/apiTarea$ sail artisan test

PASS Tests\Unit\ExampleTest
✓ that true is true

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

PASS Tests\Feature\LoginTest
✓ register
✓ login
✓ logout

PASS Tests\Feature\TareasTest
✓ get tareas
✓ update tarea
✓ delete tarea
✓ insert tarea

Tests: 9 passed (24 assertions)
Duration: 10.68s

steven@Revision-PC:/mnt/c/Users/Cubillos/Desktop/servidor/Docker compose/www/apiTarea$ |

```

Etiqueta:

```
steven@Revision-PC:/mnt/c/Users/Cubillos/Desktop/servidor/Docker compose/www/apiTarea$ sail artisan make:test EtiquetaTest
INFO Test [tests/Feature/EtiquetaTest.php] created successfully.
steven@Revision-PC:/mnt/c/Users/Cubillos/Desktop/servidor/Docker compose/www/apiTarea$ |
```

```
<?php

namespace Tests\Feature;

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

class EtiquetaTest extends TestCase
{
    use RefreshDatabase;
    /**
     * A basic feature test example.
     */

    public function test_get_labels()
    {
        $user = User::create([
            'name' => 'algo',
            'email' => 'texas@gmail.com',
            'password' => bcrypt('password'),
        ]);

        $response = $this->actingAs($user)->get('/api/etiquetas');

        $response->assertStatus(200);
    }
}
```

```

public function test_make_label()
{
    $user = User::create([
        'name' => 'maria',
        'email' => 'francia@gmail.com',
        'password' => bcrypt('password'),
    ]);

    $this->actingAs($user);

    $datosEtiqueta = [
        'nombre' => 'Club',
    ];

    $response = $this->post('/api/etiquetas', $datosEtiqueta);

    $this->assertDatabaseHas('etiquetas', [
        'nombre' => 'Club',
    ]);

    $response->assertStatus(201);
}

public function test_get_one_label()

```

```

public function test_get_one_label()
{
    $user = User::create([
        'name' => 'test',
        'email' => 'test@gmail.com',
        'password' => bcrypt('password'),
    ]);

    $etiqueta = Etiqueta::create([
        'nombre' => 'Nombre',
    ]);

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

    $response->assertStatus(200);

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

```

```
public function test_update_label()
{
    $user = User::create([
        'name' => 'test',
        'email' => 'test@gmail.com',
        'password' => bcrypt('password'),
    ]);

    $etiqueta = Etiqueta::create([
        'nombre' => 'Nombre',
    ]);

    $datosActualizados = [
        'nombre' => 'Nuevo',
    ];

    $response = $this->actingAs($user)->put('/api/etiquetas/' . $etiqueta->id, $datosActualizados);
    $response->assertStatus(200);
    $etiqueta->refresh();
    $this->assertEquals('Nuevo', $etiqueta->nombre);
}
```

```
public function test_delete_label()
{
    $user = User::create([
        'name' => 'paco',
        'email' => 'torre@gmail.com',
        'password' => bcrypt('password'),
    ]);

    $etiqueta = Etiqueta::create([
        'nombre' => 'Nombre',
    ]);

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

    $response->assertStatus(200);

    $this->assertDatabaseMissing('etiquetas', [
        'id' => $etiqueta->id,
    ]);
}
```

Ejecutamos el test **con sail artisan test**.

```
steven@Revision-PC:/mnt/c/Users/Cubillos/Desktop/servidor/Docker compose/www/apiTarea$ sail artisan test

PASS Tests\Unit\ExampleTest
✓ that true is true

PASS Tests\Feature\EtiquetaTest
✓ get labels
✓ make label
✓ get one label
✓ update label
✓ delete label

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

PASS Tests\Feature\LoginTest
✓ register
✓ login
✓ logout

PASS Tests\Feature\TareasTest
✓ get tareas
✓ update tarea
✓ delete tarea
✓ insert tarea

Tests: 14 passed (34 assertions)
Duration: 10.88s

steven@Revision-PC:/mnt/c/Users/Cubillos/Desktop/servidor/Docker compose/www/apiTarea$ |
```