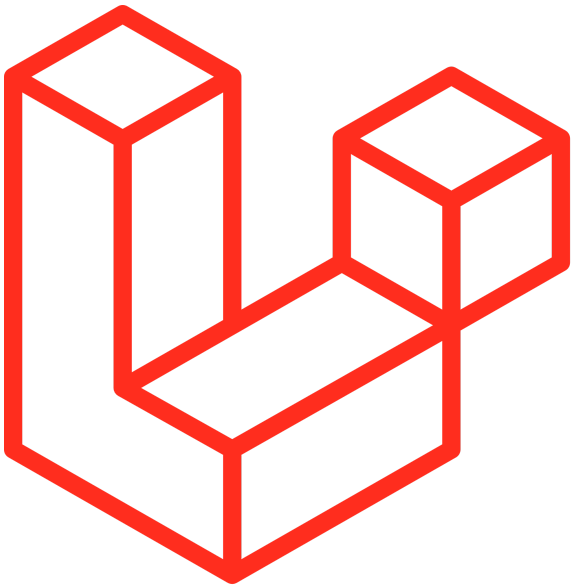
PROYECTO LARAVEL
APICLASE



CRISTINA CONDE SERRANO
2º DAW - TARDE
2023/2024

ÍNDICE

1.- CREACIÓN PROYECTO EN LARAVEL

Para iniciar con la creación del proyecto en primer lugar vamos a elegir el nombre, en nuestro caso será ApiTask, y para poder crearlo vamos a usar el siguiente comando:

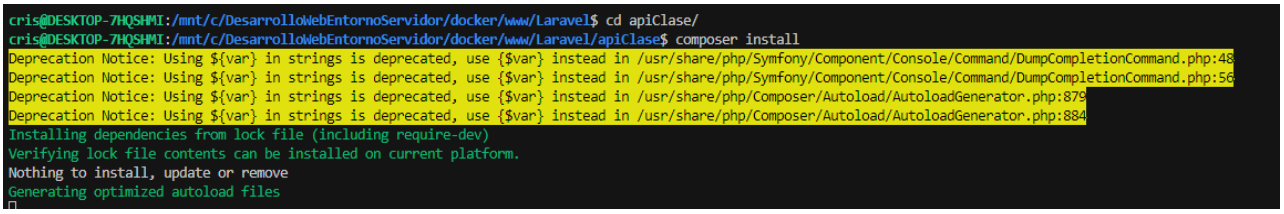
- **composer create-project - -prefer-dist laravel/laravel apiClase**

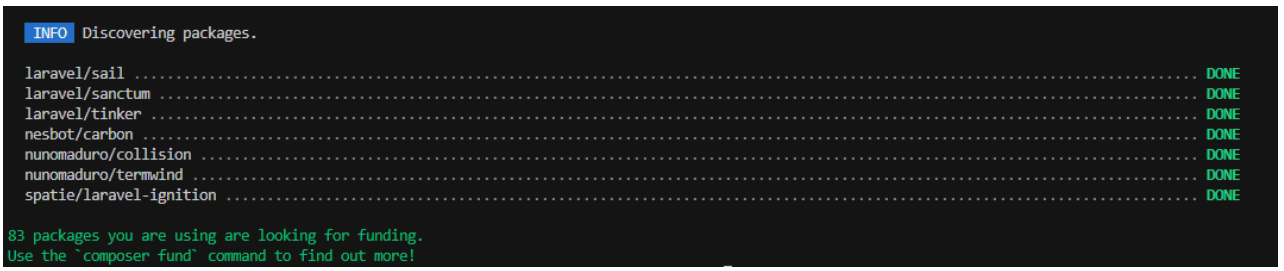
Una vez creado el proyecto vamos a entrar dentro de la carpeta para poder realizar la instalación de todos los ficheros que son necesarios para poder realizar la api.

- Entrar en carpeta: **cd apiClase**

Con el comando de composer vamos a instalar todas las dependencias y herramientas necesarias para que nuestro proyecto funcione correctamente.

- **composer install**





Una vez tenemos esto realizado vamos a indicar a nuestro entorno que se desarrolle en local y para ello hay que configurarlo con el siguiente comando:

- **php artisan sail:install**

```
cris@DESKTOP-7HQSHMI:/mnt/c/DesarrolloWebEntornoServidor/docker/www/Laravel/apiClase$ php artisan sail:install

  Which services would you like to install?
  mysql

   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
```

Con este comando vemos que nos pide elegir el tipo de lenguaje que queremos usar para nuestra base de datos, para este proyecto vamos a usar MYSQL.

Además vamos a instalar las librerías necesarias para nuestro proyecto y para ello usaremos el comando:

- **npm install**

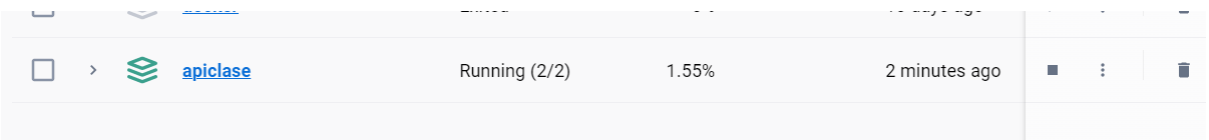
```
cris@DESKTOP-7HQSHMI:/mnt/c/DesarrolloWebEntornoServidor/docker/www/Laravel/apiClase$ npm install

added 23 packages, and audited 24 packages in 18s

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

found 0 vulnerabilities
npm notice
npm notice New minor version of npm available! 10.2.4 -> 10.4.0
npm notice Changelog: https://github.com/npm/cli/releases/tag/v10.4.0
npm notice Run npm install -g npm@10.4.0 to update!
npm notice
```

Por último vamos a llevar nuestro proyecto a docker para que podamos trabajar a nivel local con el siguiente comando:

- **./vendor/bin/sail up -d**

```
cris@DESKTOP-7HQSHMI:/mnt/c/DesarrolloWebEntornoServidor/docker/www/Laravel/apiClase$ ./vendor/bin/sail up
[+] Building 0.0s (0/0)
[+] Running 3/3
 ✓ Network apiclase_sail              Created
 ✓ Container apiclase-mysql-1         Created
 ✓ Container apiclase-laravel.test-1  Created
Attaching to apiclase-laravel.test-1, apiclase-mysql-1
```

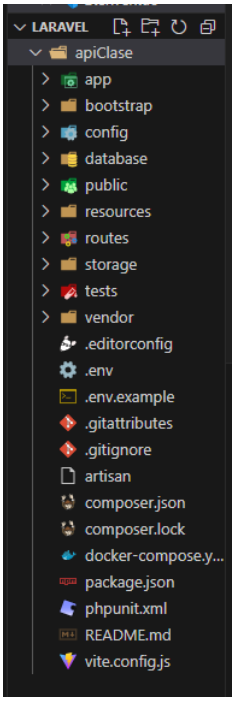
DOCKER



Para continuar con nuestro proyecto vamos a tener que usar varias veces el comando ./vendor/bin/sail y para no tener que escribirlo tantas veces y poder ahorrar código vamos a crear un alias y para eso usamos:

- **alias sail=./vendor/bin/sail**

Con todo lo hecho anteriormente, ya tenemos nuestra estructura de carpetas creada para poder trabajar en ellas.



2.- CREACIÓN DE TABLAS PARA BASE DE DATOS

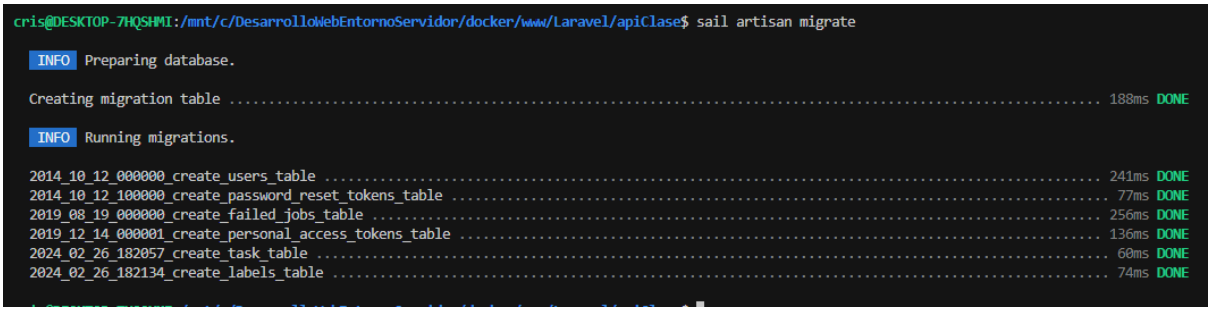
En nuestra base de datos tenemos que tener dos tablas: tareas (task) y etiquetas (labels) para ellos vamos a hacerlo mediante los siguientes comandos:

- **sail artisan make:migration create_task_table**
- **sail artisan make:migration create_labels_table**

```
cris@DESKTOP-7HQSHMI:/mnt/c/DesarrolloWebEntornoServidor/docker/www/Laravel/apiClase$ sail artisan make:migration create_task_table

  INFO  Migration [database/migrations/2024_02_26_182057_create_task_table.php] created successfully.

cris@DESKTOP-7HQSHMI:/mnt/c/DesarrolloWebEntornoServidor/docker/www/Laravel/apiClase$ sail artisan make:migration create_labels_table

  INFO  Migration [database/migrations/2024_02_26_182134_create_labels_table.php] created successfully.
```

Después hay que realizar una actualización para vincular todos los cambios que se han ido creando, esto se llama migrate, por tanto, cada vez que realicemos un cambio habrá que realizar un migrate para vincular todos los cambios. El comando que usaremos es el siguiente:

- sail artisan migrate

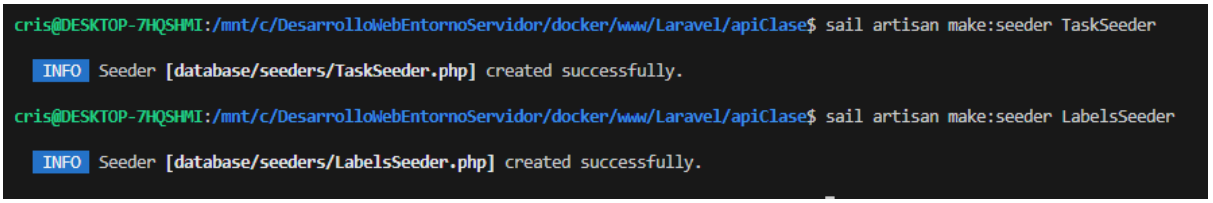


Nota: En caso de que nos hayamos confundido a la hora de crear la tabla, podremos revertir los cambios o borrarla con los comandos:

- sail artisan migrate:refresh
- sail artisa migrate:rollback - - step=<número de fila> (para borrar)

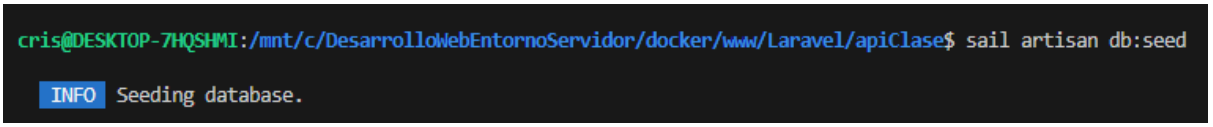
A continuación vamos a crear los seeder, que son archivos de PHP que contienen el código para insertar registros en la base de datos, usaremos el siguiente comando:

- sail artisan make:seeder TaskSeeder
- sail artisan make:seeder LabelsSeeder



Una vez tenemos esto realizado hay que ingestarlo en nuestra base de datos para poder añadir todas las tareas y etiquetas que necesitemos, usaremos:

- sail artisan db:seed



Tras esto hay que crear el controlar y el modelo que nos ayudarán a:

1. Modelo: Representa los datos de la aplicación y define la interacción con la base de datos.
2. Controlador: Es el responsable de manejar las solicitudes HTTP entrantes y actuar como un intermediario entre las vistas y los modelos.

Para crear el modelo usaremos:

- sail artisan make:model Task
- sail artisan make:model Labels

```
cris@DESKTOP-7HQSHMI:/mnt/c/DesarrolloWebEntornoServidor/docker/www/Laravel/apiClase$ sail artisan make:model Task

  INFO  Model [app/Models/Task.php] created successfully.

cris@DESKTOP-7HQSHMI:/mnt/c/DesarrolloWebEntornoServidor/docker/www/Laravel/apiClase$ sail artisan make:model Labels

  INFO  Model [app/Models/Labels.php] created successfully.
```

Y para crear los controladores usaremos:

- sail artisan make:controller AuthController
- sail artisan make:controller Task
- sail artisan make:controller Labels

```
cris@DESKTOP-7HQSHMI:/mnt/c/DesarrolloWebEntornoServidor/docker/www/Laravel/apiClase$ sail artisan make:controller AuthController

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

cris@DESKTOP-7HQSHMI:/mnt/c/DesarrolloWebEntornoServidor/docker/www/Laravel/apiClase$ sail artisan make:controller Task

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

cris@DESKTOP-7HQSHMI:/mnt/c/DesarrolloWebEntornoServidor/docker/www/Laravel/apiClase$ sail artisan make:controller Labels

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

Tenemos que crear una solicitud de validación en Laravel para facilitar la implementación de la validación de formularios en nuestra aplicación, lo haremos con estos comandos:

- sail artisan make:request TaskRequest
- sail artisan make:request LabelsRequest

```
cris@DESKTOP-7HQSHMI:/mnt/c/DesarrolloWebEntornoServidor/docker/www/Laravel/apiClase$ sail artisan make:request TaskRequest

  INFO  Request [app/Http/Requests/TaskRequest.php] created successfully.

cris@DESKTOP-7HQSHMI:/mnt/c/DesarrolloWebEntornoServidor/docker/www/Laravel/apiClase$ sail artisan make:request LabelsRequest

  INFO  Request [app/Http/Requests/LabelsRequest.php] created successfully.
```

Vamos a crear además recursos para facilitar la creación de clases en Laravel, lo que nos permitirá definir fácilmente la forma en que se presentan los datos devueltos por nuestra API web, usaremos:

- sail artisan make:resource TaskResource
- sail artisan make:resource LabelsResource

```
cris@DESKTOP-7HQSHMI:/mnt/c/DesarrolloWebEntornoServidor/docker/www/Laravel/apiClase$ sail artisan make:resource TaskResource

  INFO  Resource [app/Http/Resources/TaskResource.php] created successfully.

cris@DESKTOP-7HQSHMI:/mnt/c/DesarrolloWebEntornoServidor/docker/www/Laravel/apiClase$ sail artisan make:resource LabelsResource

  INFO  Resource [app/Http/Resources/LabelsResource.php] created successfully.
```

Para poder ver realizar pruebas para verificar si está o no correcto, podemos realizar test ya que nos permiten escribir y ejecutar pruebas automatizadas para verificar el correcto funcionamiento de la api, para ello usaremos:

- sail artisan make:test TaskTest
- sail artisan make:test LabelsTest
- sail artisan make:test LoginTest

```
cris@DESKTOP-7HQSHMI:/mnt/c/DesarrolloWebEntornoServidor/docker/www/Laravel/apiClase$ sail artisan make:test TaskTest

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

cris@DESKTOP-7HQSHMI:/mnt/c/DesarrolloWebEntornoServidor/docker/www/Laravel/apiClase$ sail artisan make:test LabelsTest

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

cris@DESKTOP-7HQSHMI:/mnt/c/DesarrolloWebEntornoServidor/docker/www/Laravel/apiClase$ sail artisan make:test LoginTest

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

Ahora podemos hacer una comprobación de lo que tenemos hasta ahora creado, que serán test de prueba, para ello usaremos el comando:

- sail test

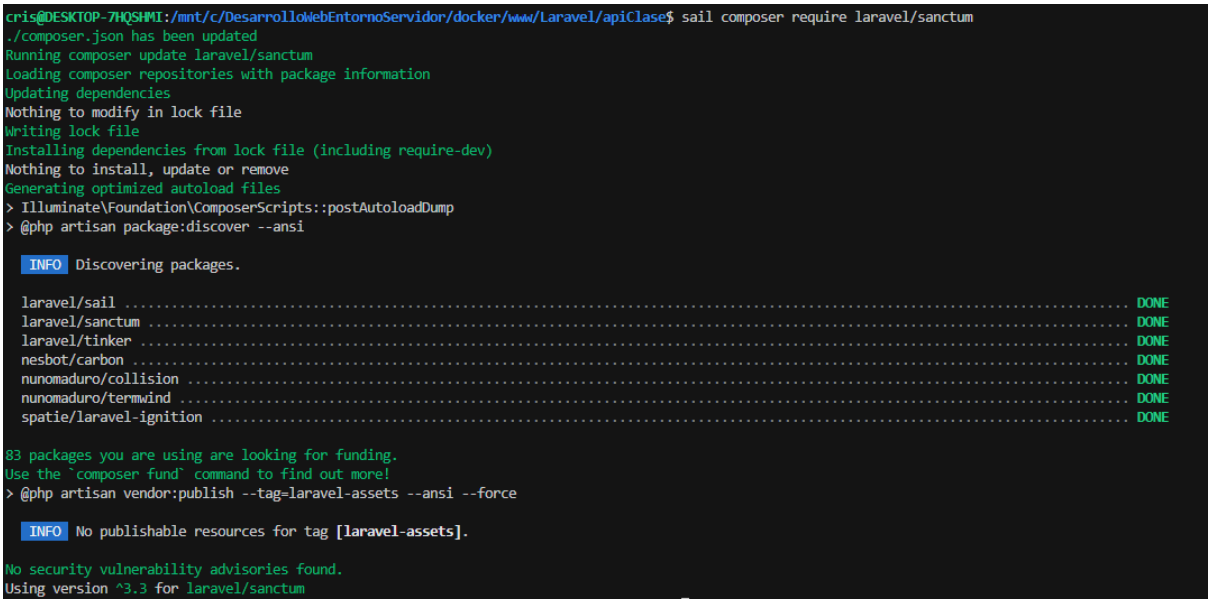
```
cris@DESKTOP-7HQSHMI:/mnt/c/DesarrolloWebEntornoServidor/docker/www/Laravel/apiClase$ sail test

  PASS  Tests\Unit\ExampleTest
  ✓ that true is true

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

  PASS  Tests\Feature\LabelsTest
  ✓ example

  PASS  Tests\Feature\LoginTest
  ✓ example

  PASS  Tests\Feature\TaskTest
  ✓ example

  Tests:    5 passed (5 assertions)
  Duration: 19.10s
```

Adicionalmente para lanzar los test usaremos este mismo comando.

3.- INSTALACIÓN DE SANCTUM

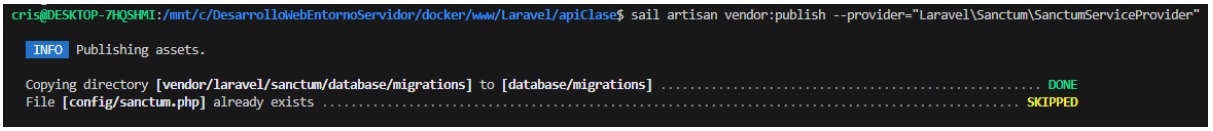
Instalaremos Sanctum ya que es una herramienta que proporciona autenticación API simple y segura para nuestra aplicación en Laravel, esto nos dará tokens de acceso, ya que es una forma común de autenticación utilizada, para ello usaremos estos comandos de instalación:

- sail composer require laravel/sanctum

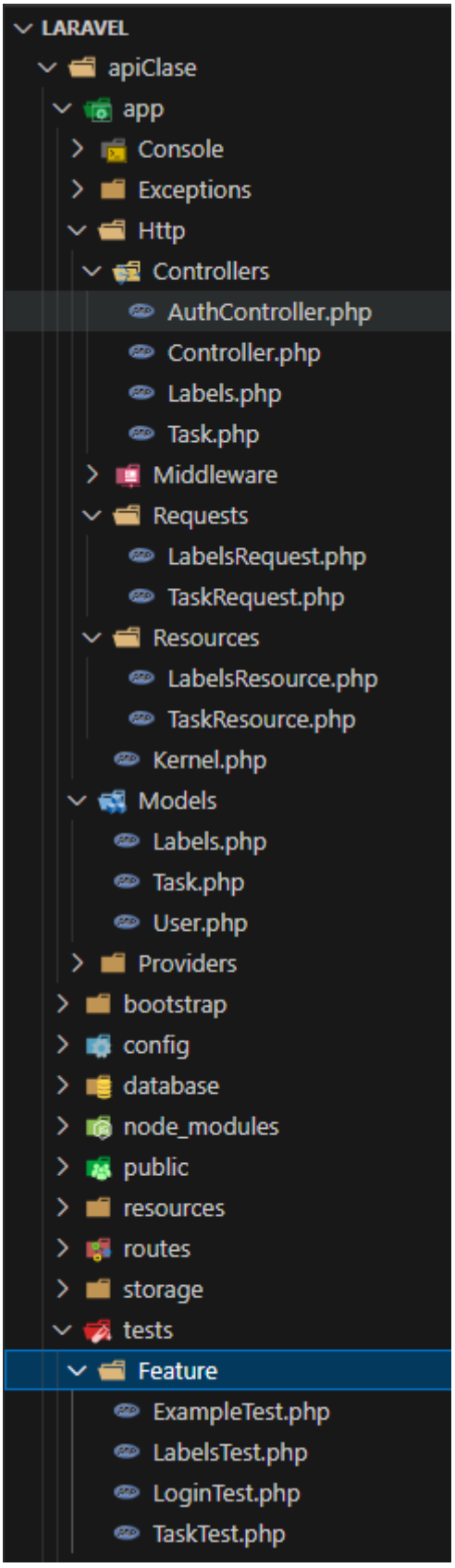


Ahora tenemos que copiar todos los archivos y recursos necesarios desde el paquete de Laravel Sanctum a nuestro directorio de la aplicación, esto nos sirve para poder personalizar y modificar la configuración según nuestras necesidades, para ellos usaremos:

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



Tras crear todo esto nuestra nueva estructura de carpetas es esta:



4.- CREACIÓN DE CONTROLADORES

4.1- AuthControllers

```php
<?php

namespace App\Http\Controllers;

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

class AuthController extends Controller
{
    /**
     * Display a listing of the resource.
     */
    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, 'acces_token'=>
$token, 'token_type'=> 'Bearer']);
    }
    public function logout(){
        auth()->user()->tokens()->delete();
        return ['message'=> 'Sesión cerrada correctamente'];
    }

    public function login(Request $request)
{
    $user = User::where('email', $request->email)->firstOrFail();
    $token = $user->createToken('auth_token')->plainTextToken;
    return response()->json([
        'message' => '¡Hola, ' . $user->name . '!',
        'access_token' => $token,
```

```
        'token_type' => 'Bearer'
    ]);
}
}
```

4.2.- LabelsController

```
<?php

namespace App\Http\Controllers;

use App\Models\Labels;

use App\Http\Requests\LabelsRequest;
use App\Http\Resources\LabelsResource;

use Illuminate\Http\Resources\Json\JsonResource;


class LabelsController extends Controller
{
    /**
     * Display a listing of the resource.
     */
    public function index():JsonResource
    {
        $Labelss = Labels::all();

        return LabelsResource::collection($Labelss);
    }

    /**
     * Store a newly created resource in storage.
     */
    public function store(Request $request)
{
    $validatedData = $request->validate([
        'nombre' => 'required|string|max:255',
    ]);
```

```
    $label = Labels::create($validatedData);

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



    /**
     * Display the specified resource.
     */

    public function show($id):JsonResource

    {
        $Labels = Labels::find($id);


return new LabelsResource($Labels);
    }



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

    }

    /**
     * Update the specified resource in storage.
     */
    public function update(LabelsRequest $request, $id): JsonResource
    {
        try {
            $Labels = Labels::find($id);

            if (!$Labels) {
                return response()->json(['error' => 'La Labels no se
encontró'], 404);
            }

            $Labels->update($request->all());
```

```
            return new LabelsResource($Labels);
        } catch (\Exception $e) {
            return response()->json(['message' => 'No se pudo
actualizar', 'error' => $e->getMessage()], 500);
        }
    }



    public function destroy($id)
    {
        $label = Labels::findOrFail($id);
        // Desvincula todas las relaciones de las tareas antes de
eliminar la etiqueta
        $label->tasks()->detach();
        $label->delete();
        return response()->json(['message' => 'Etiqueta eliminada
correctamente'], 200);
    }
}
```

4.3.-TaskController

```
<?php

namespace App\Http\Controllers;

use App\Models\Task;

use App\Http\Requests\TaskRequest;
use App\Http\Resources\TaskResource;

use Illuminate\Http\Resources\Json\JsonResource;

class TaskController extends Controller
{
    /**
     * Display a listing of the resource.
     */
    public function index(): JsonResource
```

```
    {
        $Tasks = Task::all();
        // return response()->json($productos,200);
        return TaskResource::collection($Tasks);
    }

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

    /**
     * Store a newly created resource in storage.
     */
    public function store(TaskRequest $request)
    {
        $etiquetas = $request->labels;
        $params = $request->all();
        unset($params["labels"]);
        $Task = Task::create($params);
        $Task->labels()->attach($etiquetas);

        return response()->json($Task, 201);

    }

    /**
     * Display the specified resource.
     */
    public function show($id): JsonResource
    {
        $Task = Task::find($id);
        // return response()->json($producto,200);

        return new TaskResource($Task);
    }

    /**
     * Show the form for editing the specified resource.
     */
```

```
    public function edit(Task $Task)
    {

    }

    /**
     * Update the specified resource in storage.
     */
    public function update(TaskRequest $request, $id)
{
    $task = Task::findOrFail($id);
    $task->labels()->detach();
    $task->titulo = $request->titulo;
    $task->descripcion = $request->descripcion;
    $task->save();
    $task->labels()->attach($request->labels);
    return new TaskResource($task);
}



    /**
     * Remove the specified resource from storage.
     */
    public function destroy(Task $Task)
    {
        $Task->labels()->detach();
        $Task->delete();
        return response()->json($Task, 200);
    }
}
```

5.- REQUEST

5.1.- LabelsRequest

```php
<?php
namespace App\Http\Requests;

use Illuminate\Foundation\Http\FormRequest;

class LabelsRequest extends FormRequest
{
    public function authorize(): bool
    {
        return true;
    }
    public function rules(): array
    {
        return [
            'nombre' => 'required|max:17|min:5',
        ];
    }
}
```

5.2. TaskRequest

```php
<?php
namespace App\Http\Requests;

use Illuminate\Foundation\Http\FormRequest;

class TaskRequest extends FormRequest
{
    public function authorize(): bool
    {
        return true;
    }
    public function rules(): array
    {
        return [
            'titulo' => 'required|max:30|min:3',
            'descripcion' => 'nullable|max:30|min:3',
        ];
    }
```

```
}
```

6.- RESOURCES

6.1.- LabsResources

```php
<?php
namespace App\Http\Resources;

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

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

6.2.- TaskResources

```php
<?php
namespace App\Http\Resources;

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

class TaskResource extends JsonResource
{
    public function toArray(Request $request): array
    {
        return [
            'id' => $this->id,
            'titulo' => 'Título: ' . $this->titulo,
            'descripcion' => 'Descripción: ' . $this->descripcion,
            'etiquetas' => $this->etiquetas->pluck('nombre')
        ];
    }
```

```
}
```

7.- MODELS

7.1.- Labels

```php
<?php
namespace App\Models;

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

class Labels extends Model
{
    use HasFactory;
    protected $fillable = ["nombre"];
    protected $hidden = ["created_at", "updated_at"];
    public function task(): BelongsToMany
    {
        return $this->belongsToMany(
            Task::class,
            'tarea_etiqueta',
            'etiqueta_id',
            'tarea_id'
        );
    }
};
```

7.2.-Task

```php
<?php
namespace App\Models;

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

class Task extends Model
{
    use HasFactory;
    protected $fillable = ["titulo", "descripcion"];
```

```
    protected $hidden = ["created_at", "updated_at"];
    public function labels(): BelongsToMany
    {
        return $this->belongsToMany(
            Labels::class,
            'tarea_etiqueta',
            'tarea_id',
            'etiqueta_id'
        );
    }
}
```

8.- USER

```
<?php

namespace App\Models;

// use Illuminate\Contracts\Auth\MustVerifyEmail;
use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Foundation\Auth\User as Authenticatable;
use Illuminate\Notifications\Notifiable;
use Laravel\Sanctum\HasApiTokens;

class User extends Authenticatable
{
    use HasApiTokens, HasFactory, Notifiable;

    /**
     * The attributes that are mass assignable.
     *
     * @var array<int, string>
     */
    protected $fillable = [
        'name',
        'email',
        'password',
    ];

    /**
     * The attributes that should be hidden for serialization.
     *
     * @var array<int, string>
```

```
     */
    protected $hidden = [
        'password',
        'remember_token',
    ];

    /**
     * The attributes that should be cast.
     *
     * @var array<string, string>
     */
    protected $casts = [
        'email_verified_at' => 'datetime',
        'password' => 'hashed',
    ];
}
```

9.- MIGRATIONS

9.1.- Labels

```
<?php
use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration {
    public function up(): void
    {
        Schema::create('labels', function (Blueprint $table) {
            Schema::dropIfExists('labels');
            $table->id();
            $table->string('nombre', 15);
            $table->timestamps();
        });
    }
    public function down(): void
    {
        Schema::dropIfExists('labels');
    }
};
```

9.2.- Task

```php
<?php
use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration {
    public function up(): void
    {
        Schema::dropIfExists('tasks');
        Schema::create('tasks', function (Blueprint $table) {
            $table->id();
            $table->string("titulo", 40);
            $table->string("descripcion")->nullable();
            $table->timestamps();
        });
    }
    public function down(): void
    {
        Schema::dropIfExists('tasks');
    }
};
```

9.3.- Task_Labels

```
cris@DESKTOP-7HQSHMI:/mnt/c/DesarrolloWebEntornoServidor/docker/www/Laravel/apiClase$ sail artisan make:migration create_task_labels_table

  INFO  Migration [database/migrations/2024_02_26_195653_create_task_labels_table.php] created successfully.
```

```php
<?php

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

return new class extends Migration {
    public function up(): void
    {
        Schema::create('tarea_etiqueta', function (Blueprint $table) {
            $table->id();
            $table->foreignId("tarea_id")->constrained('tasks'); //
Cambiando 'tareas' por 'tasks'
```

```
            $table->foreignId("etiqueta_id")->constrained('labels'); //
Asumiendo que 'etiquetas' es el nombre correcto de la tabla
            $table->timestamps();
        });
    }
    public function down(): void
    {
        Schema::dropIfExists('tarea_etiqueta');
    }
};
```

10.- SEEDERS

10.1.- DatabaseSeeder

```
<?php
namespace Database\Seeders;

use Illuminate\Database\Seeder;

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

10.2.- TaskSeeder

```
<?php
namespace Database\Seeders;

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

class TaskSeeder extends Seeder
{
    public function run(): void
    {
```

```
        DB::table('tareas')->insert([
            'titulo' => 'PHP project',
            'descripcion' => 'Proyecto en Laravel'
        ]);
        DB::table('tareas')->insert([
            'titulo' => 'JavaScrip',
            'descripcion' => 'Realización carrito de comprar'
        ]);
        DB::table('tareas')->insert([
            'titulo' => 'Figma',
            'descripcion' => 'Creación prototipo página web'
        ]);
    }
}
```

10.3.-LabelsSeeder

```
<?php
namespace Database\Seeders;

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

class EtiquetaSeeder extends Seeder
{
    public function run(): void
    {
        DB::table('etiquetas')->insert([
            'nombre' => 'Diseño de interfaz'
        ]);
        DB::table('etiquetas')->insert([
            'nombre' => 'Desarrollo de aplicacion web entorno servidor'
        ]);
        DB::table('etiquetas')->insert([
            'nombre' => 'Despliegue de aplicaciones web'
        ]);
        DB::table('etiquetas')->insert([
            'nombre' => 'Desarrollo de aplicacion web entorno clientes'
        ]);
    }
}
```

11.- ROUTES

11.1.- API

```php
<?php

use Illuminate\Http\Request;
use Illuminate\Support\Facades\Route;
use App\Http\Controllers\AuthController;
use App\Http\Controllers\TaskController;
use App\Http\Controllers\LabelsController;

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

Route::post('/register', [AuthController::class, 'register']);
Route::post('/login', [AuthController::class, 'login']);
Route::post('/labels', [LabelsController::class, 'store']);


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

12.- TEST

12.1.- LabelsTest

```php
<?php
namespace Tests\Feature;

use App\Models\Labels;
use Illuminate\Foundation\Testing\RefreshDatabase;
use Tests\TestCase;

class LabelsTest extends TestCase
{
    use RefreshDatabase;
    public function test_get_Labels()
    {
        $task = new Labels();
        $task->nombre = "testNombre";
        $task->save();
        $response = $this->getJson('api/Labels');
        $response->assertStatus(200);
        $response->assertJsonFragment([
            'id' => $task->id,
            'nombre' => 'Nombre: ' . $task->nombre
        ]);
    }
    public function test_create_Labels()
    {
        $data = [
            'nombre' => 'Geografia'
        ];
        $response = $this->postJson('api/Labels', $data);
        $response->assertStatus(201);
        $this->assertDatabaseHas('Labels', [
            'nombre' => $data['nombre']
        ]);
        $response->assertJson([
            'nombre' => $data['nombre']
        ]);
    }
    public function test_delete_Labels()
    {
        $Labels = new Labels();
```

```
        $Labels->nombre = "Geografia";
        $Labels->save();
        $response = $this->deleteJson('api/Labels/' . $Labels->id);
        $response->assertStatus(200);
        $this->assertDatabaseMissing('Labels', ['id' => $Labels]);
    }
    public function test_show_Labels()
    { {
            $Labels = Labels::create([
                'nombre' => 'Finalizado',
            ]);
            $response = $this->getJson("api/Labels/{$Labels->id}");
            $response->assertStatus(200);
            $response->assertJson([
                'data' => [
                    'id' => $Labels->id,
                    'nombre' => 'Nombre: ' . $Labels->nombre
                ]
            ]);
        }
    }
    public function test_update_Labels()
    {
        $Labels = new Labels();
        $Labels->nombre = "testNombre";
        $Labels->save();
        $Labels->nombre = "nuevoNombre";
        $updateResponse = $this->putJson('api/Labels/' . $Labels->id, [
            'nombre' => $Labels->nombre,
        ]);
        $updateResponse->assertStatus(200);
        $Labels = Labels::find($Labels->id);
        $updateResponse->assertJsonFragment([
            'id' => $Labels->id,
            'nombre' => 'Nombre: ' . $Labels->nombre,
        ]);
        $updateResponse->assertJsonMissing([
            'nombre' => 'Nombre: ' . "testNombre",
        ]);
    }
}
```

12.2.- TaskTest

```php
<?php
namespace Tests\Feature;

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

class TaskTest extends TestCase
{
    use RefreshDatabase;
    public function test_store_Task()
    {
        $user = User::factory()->create();
        $Labels = Labels::create([
            'nombre' => 'Hola'
        ]);
        $data = [
            'titulo' => 'Título de prueba',
            'descripcion' => 'Descripción de prueba',
            'Labelss' => [$Labels->id]
        ];
        $Task = Task::create([
            'titulo' => $data['titulo'],
            'descripcion' => $data['descripcion']
        ]);
        $Task->Labelss()->sync($data['Labelss']);
        $response = $this->actingAs($user)
            ->withSession(['banned' => false])
            ->postJson("api/Task", $data);
        $response->assertStatus(201);
    }
    public function test_get_Tasks()
    {
        $user = User::factory()->create();
        $task = new Task();
        $task->titulo = "testTitulo";
        $task->descripcion = "testDescripcion";
        $task->save();
```

```
        $response = $this->actingAs($user)
            ->withSession(['banned' => false])
            ->getJson('api/Task');
        $response->assertStatus(200);
    }
    public function test_update_Task()
    {
        $user = User::factory()->create();
        $task = new Task();
        $task->titulo = "testTitulo";
        $task->descripcion = "testDescripcion";
        $task->save();
        $task->descripcion = "nuevoDescripcion";
        $updateResponse = $this->actingAs($user)
            ->withSession(['banned' => false])
            ->putJson('api/Task/' . $task->id, [
                'titulo' => $task->titulo,
                'descripcion' => $task->descripcion
            ]);
        $updateResponse->assertStatus(200);
        $task = Task::find($task->id);
        $updateResponse->assertJsonFragment([
            'id' => $task->id,
            'titulo' => 'Título: ' . $task->titulo,
            'descripcion' => 'Descripción: ' . $task->descripcion,
            'Labelss' => []
        ]);
        $updateResponse->assertJsonMissing([
            'titulo' => 'Titulo: ' . "testTitulo",
            'descripcion' => 'Descripcion: ' . 'testDescripcion'
        ]);
    }
    public function test_delete_Task()
    {
        $user = User::factory()->create();
        $Labels = Labels::create([
            'nombre' => 'Hola'
        ]);
        $data = [
            'titulo' => 'Título de prueba',
            'descripcion' => 'Descripción de prueba',
            'Labelss' => [$Labels->id]
        ];
```

```
        $Task = Task::create([
            'titulo' => $data['titulo'],
            'descripcion' => $data['descripcion']
        ]);
        $Task->Labelss()->sync($data['Labelss']);
        $response = $this->actingAs($user)
            ->withSession(['banned' => false])
            ->deleteJson("api/Task/{$Task->id}");
        $response->assertStatus(200);
        $this->assertDatabaseMissing('Tasks', ['id' => $Task->id]);
    }
    public function test_show_Task()
    {
        $user = User::factory()->create();
        $Task = Task::create([
            'titulo' => 'Título de prueba',
            'descripcion' => 'Descripción de prueba'
        ]);
        $response = $this->actingAs($user)
            ->withSession(['banned' => false])
            ->getJson("api/Task/{$Task->id}");
        $response->assertStatus(200);
        $response->assertJson([
            'data' => [
                'id' => $Task->id,
                'titulo' => 'Título: ' . $Task->titulo,
                'descripcion' => 'Descripción: ' . $Task->descripcion,
            ]
        ]);
    }
}
```

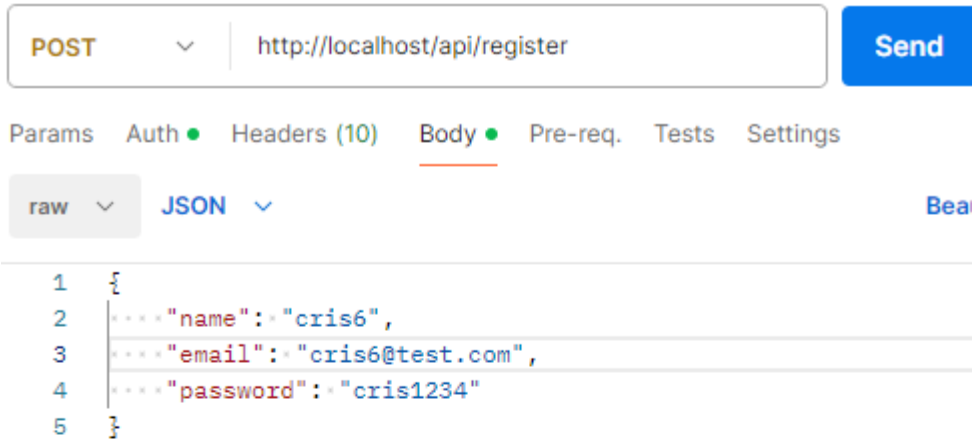
12.3.-LoginTest

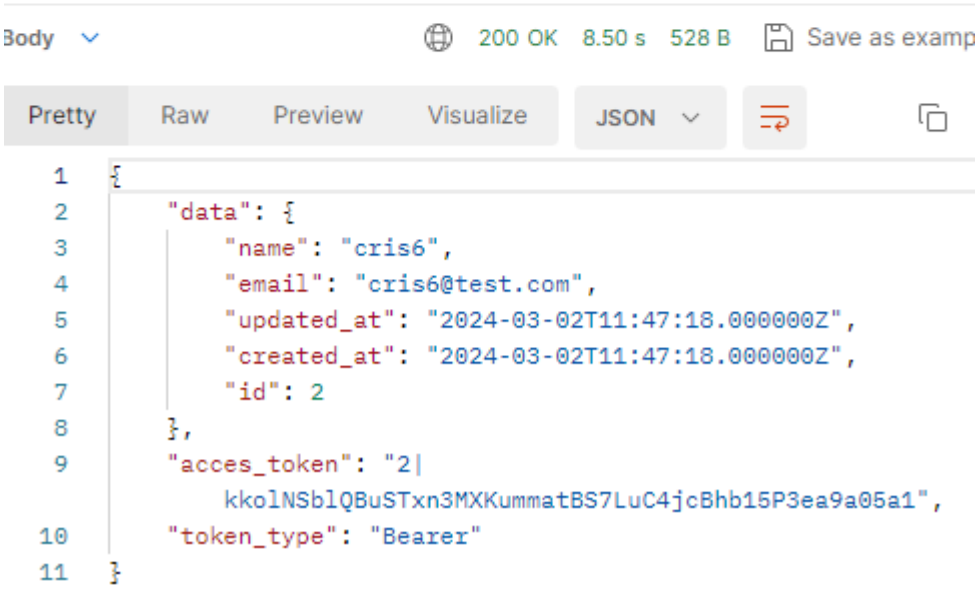
```php
<?php
namespace Tests\Feature;

use Illuminate\Foundation\Testing\RefreshDatabase;
use Tests\TestCase;
use App\Models\User;
use Illuminate\Support\Facades\Hash;
use Laravel\Sanctum\Sanctum;

class LoginTest extends TestCase
{
    use RefreshDatabase;
    public function testUserRegistration()
    {
        $userData = [
            'name' => 'Antonio',
            'email' => 'anton@pokemon.com',
            'password' => 'pokemon123',
        ];
        $response = $this->postJson('api/register', $userData);
        $response->assertStatus(200)
            ->assertJsonStructure([
                'data' => [
                    'id',
                    'name',
                    'email',
                    'created_at',
                    'updated_at'
                ],
                'acces_token',
                'token_type'
            ]);
        $this->assertDatabaseHas('users', [
            'name' => $userData['name'],
            'email' => $userData['email']
        ]);
    }
    public function test_login()
    {
        $user = User::factory()->create([
            'email' => 'anton@pokemon.com',
```

```
            'password' => Hash::make('pokemon123'),
        ]);
        $response = $this->postJson('/api/login', [
            'email' => $user->email,
            'password' => 'pokemon123',
        ]);
        $response->assertStatus(200);
        $response->assertJsonStructure(['acces_token']);
    }
    public function testLogout()
    {
        $user = User::factory()->create();
        Sanctum::actingAs($user);
        $response = $this->postJson('/api/logout');
        $response->assertStatus(200);
        $this->assertCount(0, $user->tokens);
    }
}
```

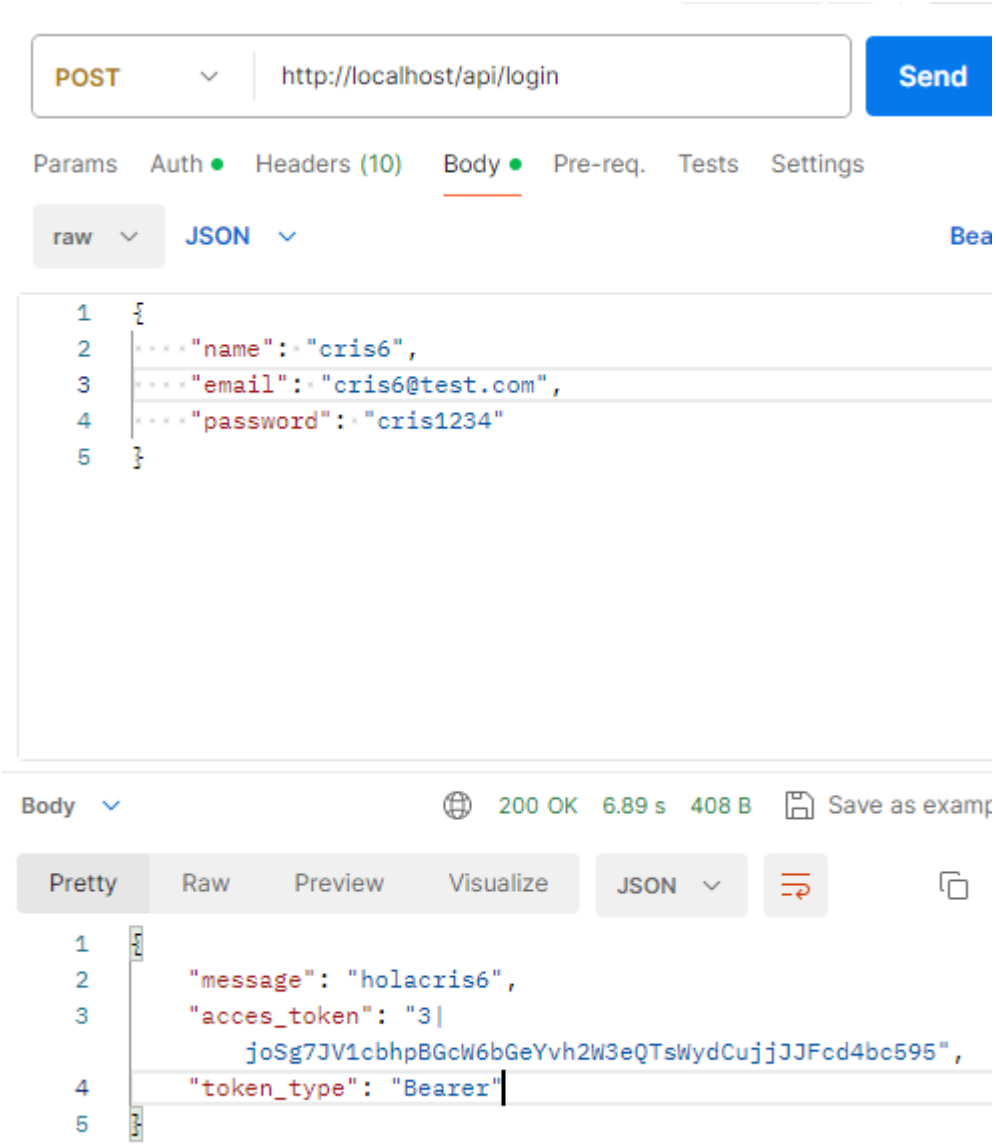
13.- TESTEO DEL CÓDIGO. COMPROBACIÓN

13.1.- REGISTRO

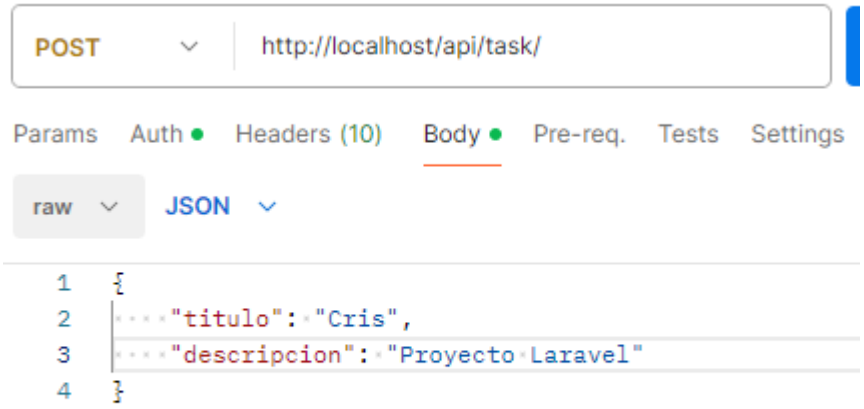


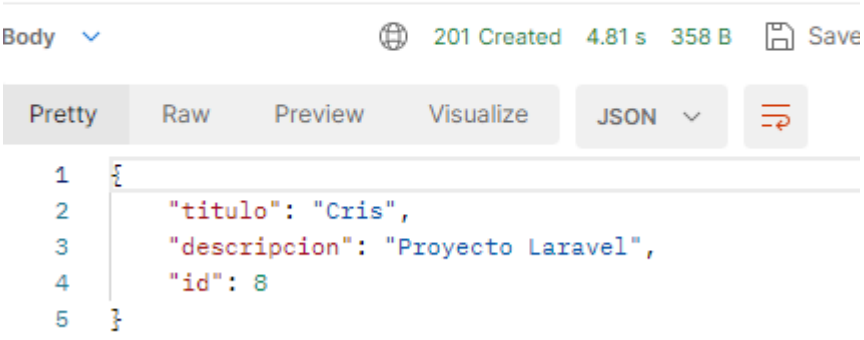


13.2.- LOGIN

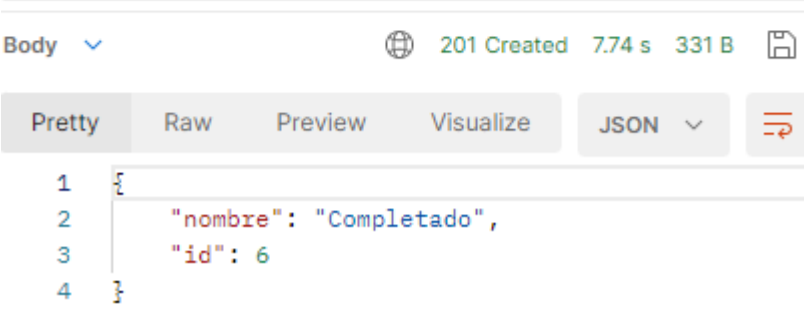

POST
http://localhost/api/login
Send
Params
Auth
Headers (10)
Body
Pre-req.
Tests
Settings
raw
JSON
{
"name": "cris6",
"email": "cris6@test.com",
"password": "cris1234"
}
Body
200 OK
6.89 s
408 B
Save as examp
Pretty
Raw
Preview
Visualize
JSON
{
"message": "holacris6",
"acces_token": "3|
joSg7JV1cbhpBGcW6bGeYvh2W3eQTsWydCujjJJFcd4bc595",
"token_type": "Bearer"
}

13.3.- PUBLICAR TAREA

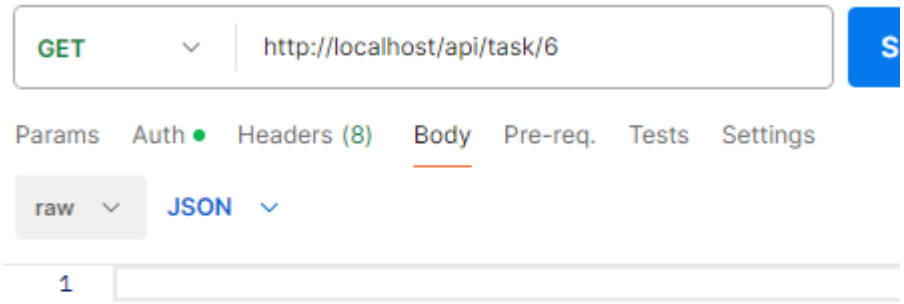


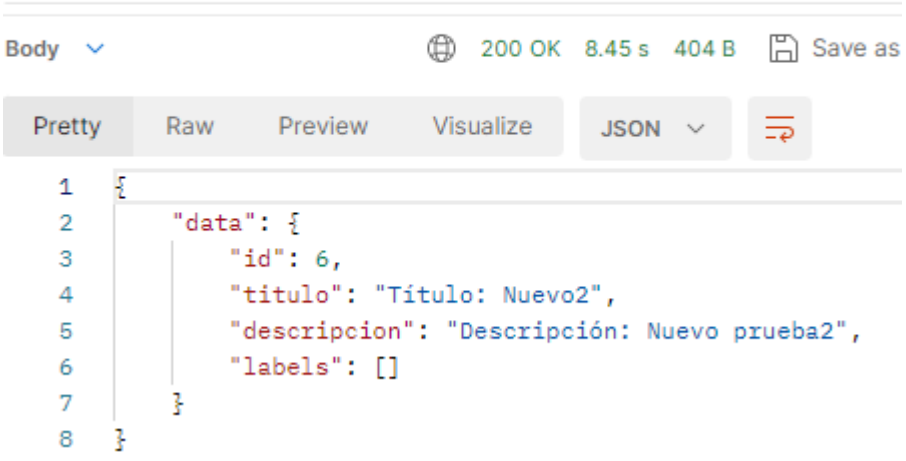


13.4.- PUBLICAR ETIQUETA

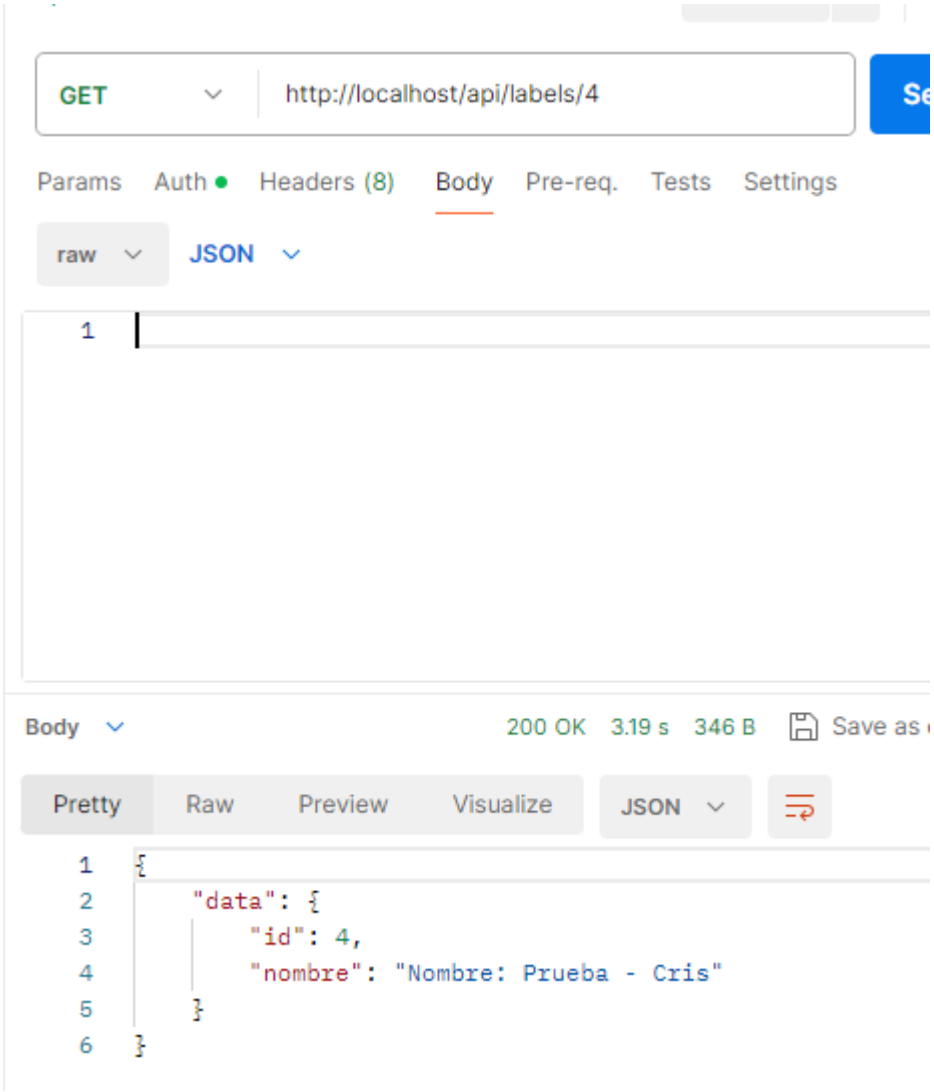


13.5.- VER TAREA CONCRETA A PARTIR DE UN ID

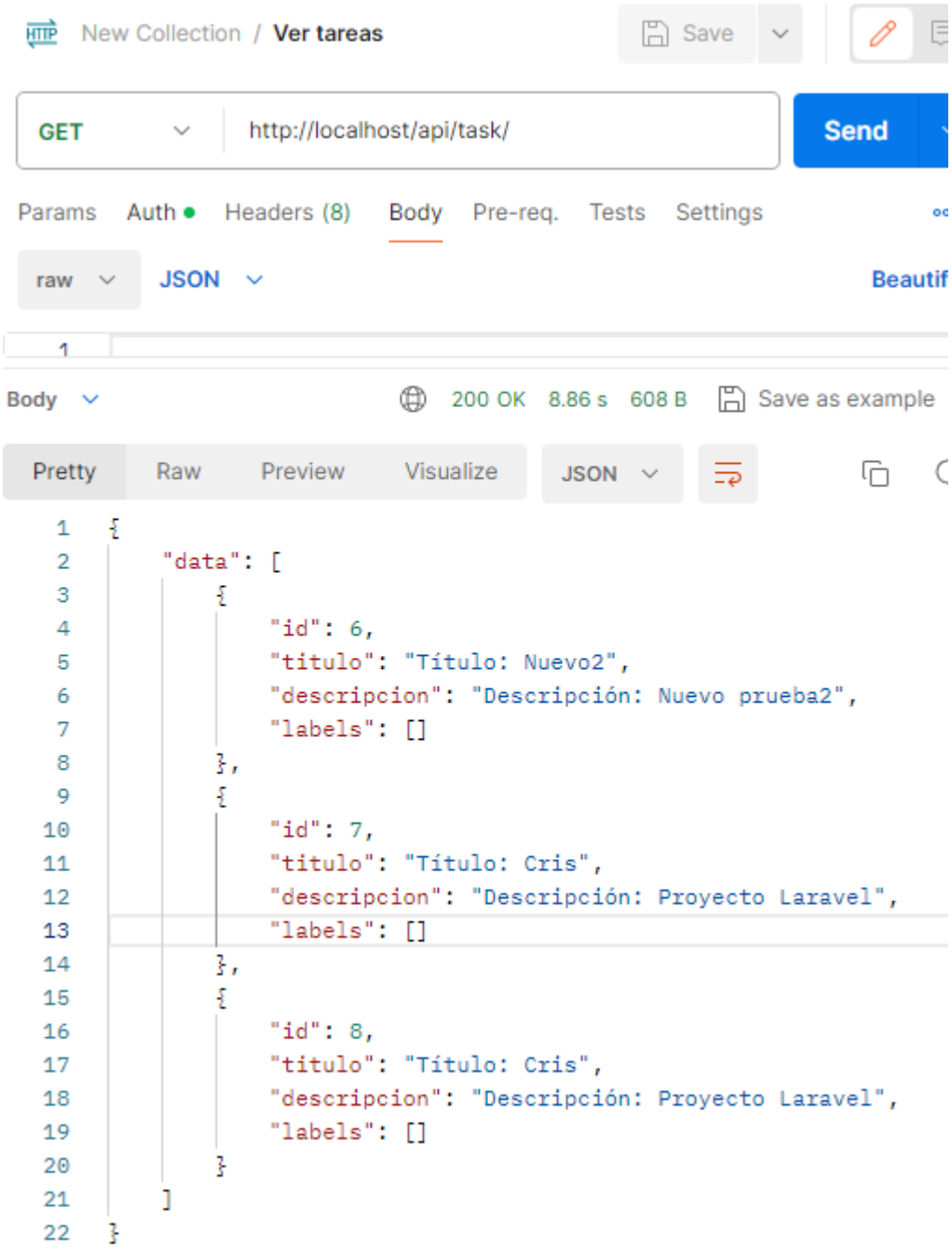




13.6.- VER ETIQUETA CONCRETA A PARTIR DE UN ID

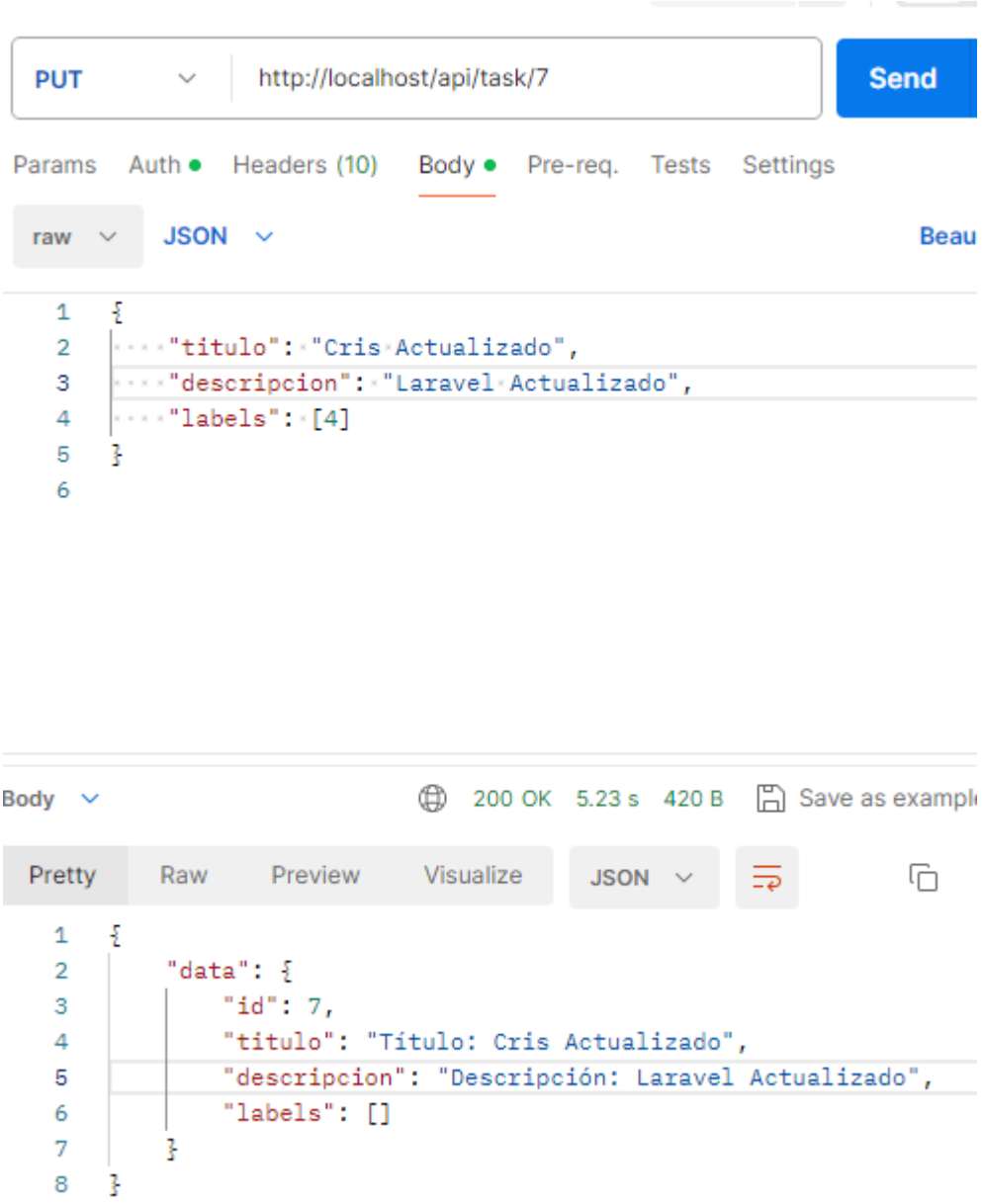


13.7.- VER LISTADO DE TAREAS

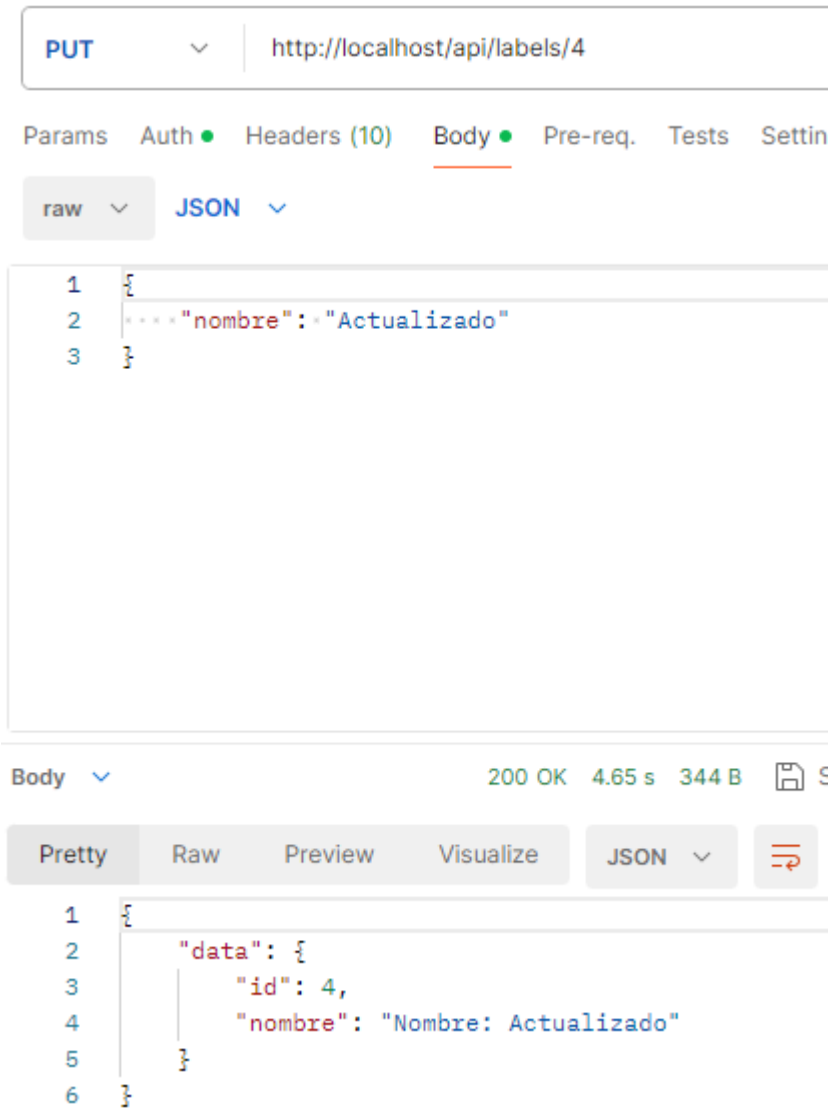


13.8.- VER LISTADO DE ETIQUETAS

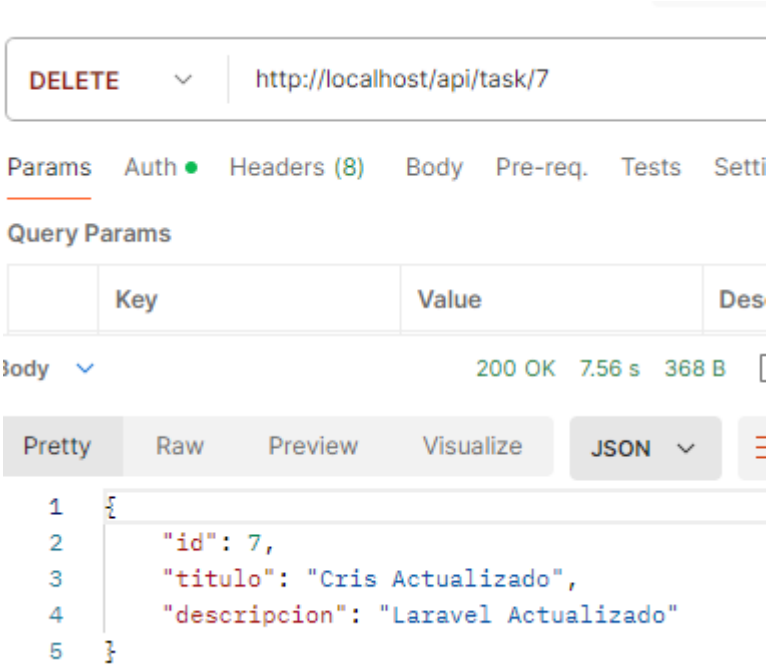
13.9.- ACTUALIZAR TAREA A PARTIR DE UN ID



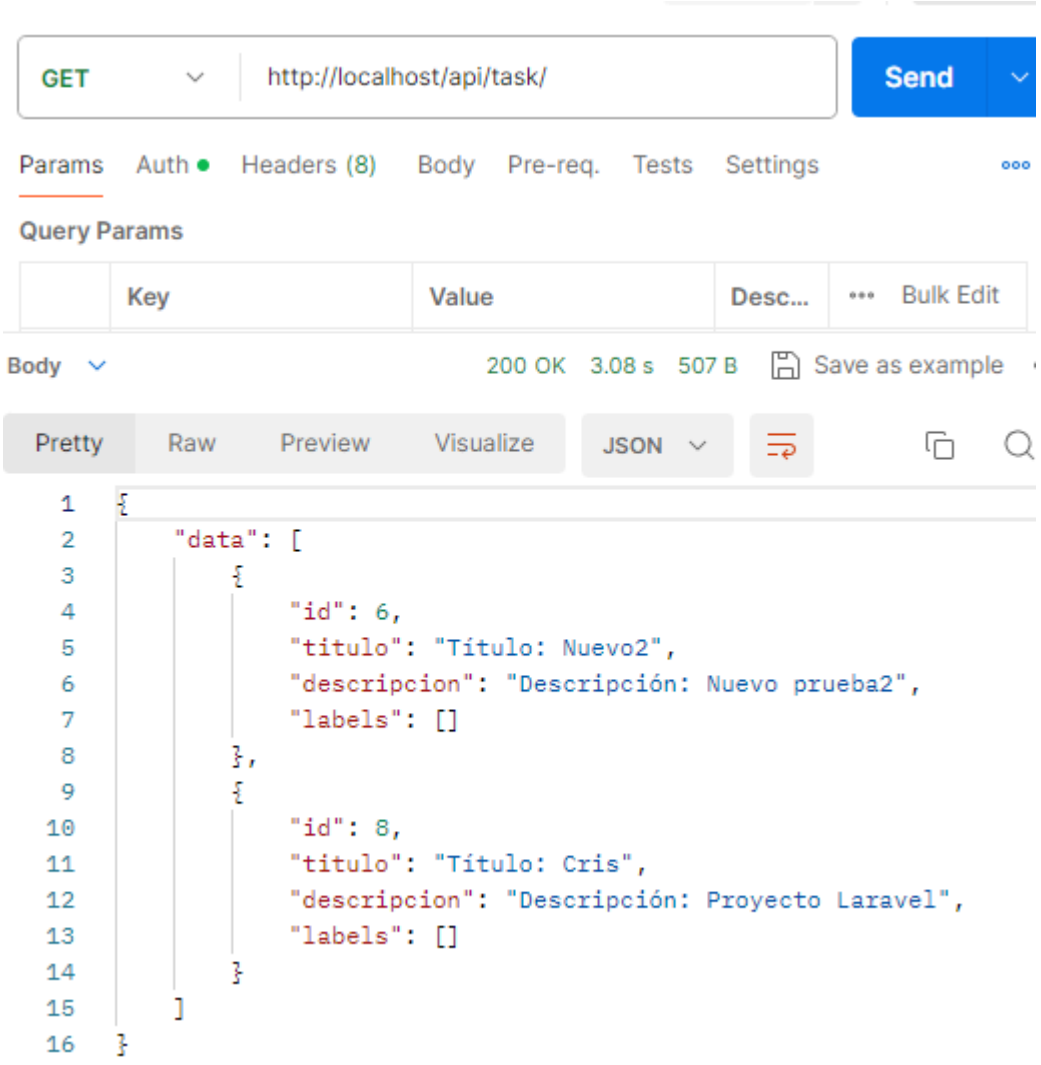
13.10.- ACTUALIZAR ETIQUETA PARTIR DE UN ID



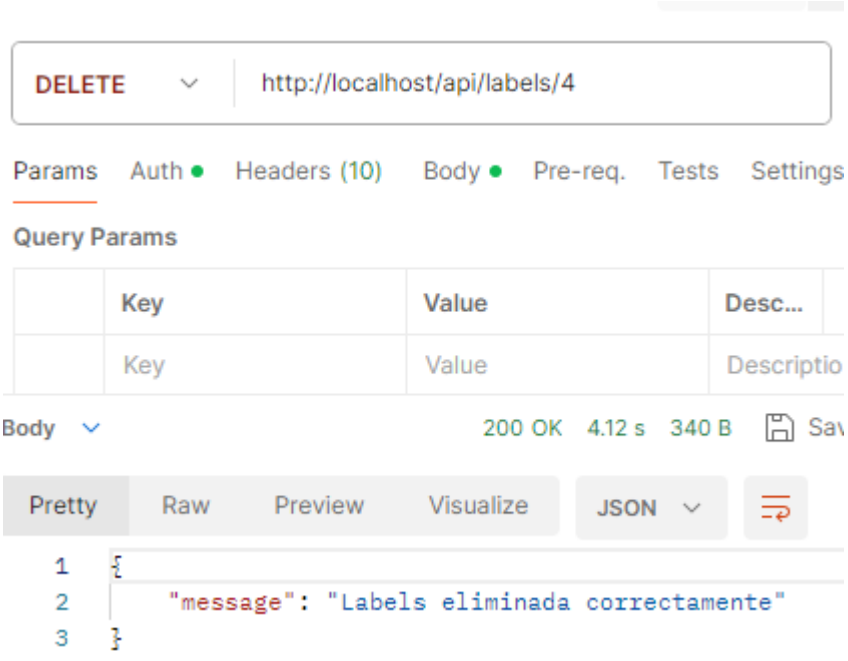
13.11.- BORRAR TAREAS A PARTIR DE UN ID



En el apartado 13.7 podíamos ver todas las tareas que tenemos, inclusive la tarea 7, ahora tras el borrado aparece así.



13.12.- BORRAR ETIQUETA PARTIR DE UN ID



En el apartado 13.8 podíamos ver un listado de etiquetas, inclusive la etiqueta 4, ahora tras el borrado queda así: