

Alumno: SAFAA FEQRI

01/06/2024

2.º DAW, TARDE

**Asignatura :Desarrollo Web en Entorno
Servidor**

Proyecto Tudespacho.net

**DOCUMENTACIÓN
BACK-END
LARAVEL**

ÍNDICE

- 1. Introducción
 - 1.1 Diagrama UML
- 2. Crear nuevo proyecto en laravel
 - 2.1 Estructura del proyecto
- 3. Laravel cheat sheet
- 4. Migraciones
 - 4.1 Ejecutar migraciones
 - 4.2 Productos
 - 4.3 Categorías
 - 4.4 Compras
- 5. Seeders
 - 5.1 DatabaseSeeder.php
 - 5.2 CategoriaSeeder.php
- 6. Models
 - 6.1 Producto.php
 - 6.2 Categoria.php
 - 6.3 Compra.php
 - 6.4 User.php
- 7. Controllers
 - 7.1 ProductoController.php
 - 7.2 CateogoriaController.php
 - 7.2 CompraController.php
 - 7.3 AuthController.php
- 8. Request
 - 8.1 AuthRequest.php
 - 8.2 ProductoRequest.php
- 9. Factories
 - 9.1 CategoriaFactory.php
- 10. Mail
 - 10.1 FormEmail.php
 - 10.2 SendEmail.php
- 11. Resources
 - View
 - 11.1 email.blade.php
 - 11.2 form.blade.php
- 12. Routes
 - 12.1 Api

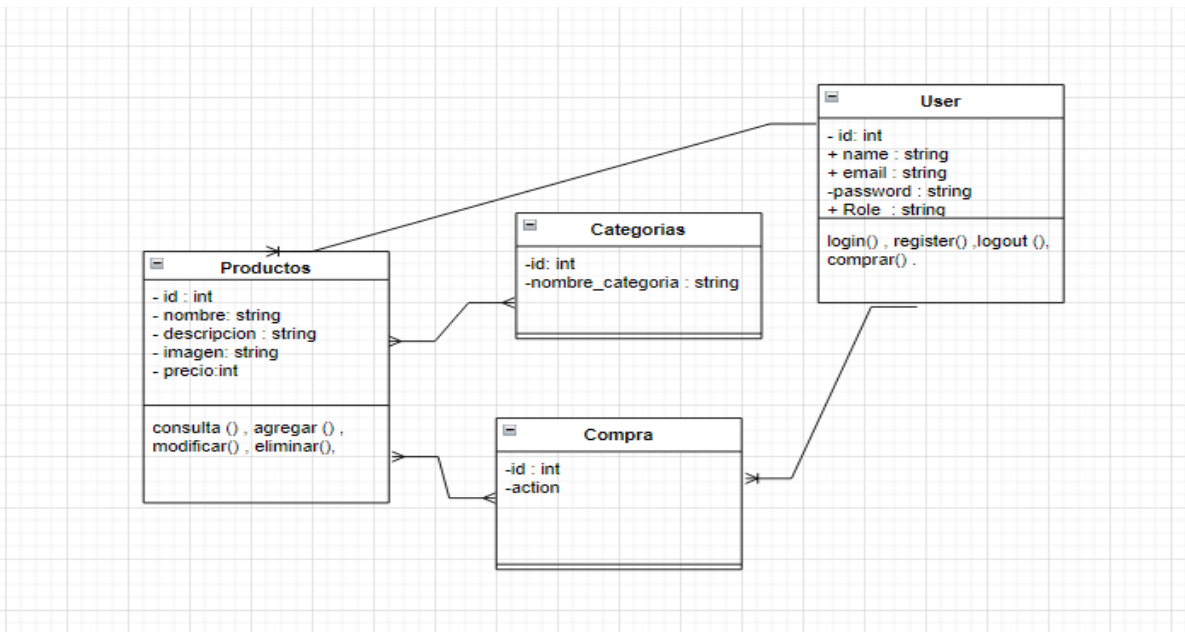
- 13. Tests
 - 13.1 ProductoTest.php
 - 13.2 LoginTest.php
- 14. Postman
 - 14.1 Register Post
 - 14.2 Login Post
 - 14.3 Crear Producto POST
 - 14.5 Modificar Producto PUT
 - 14.6 Obtener todos los productos GET
 - 14.7 Obtener un producto con su id
 - 14.8 Obtener Categorías GET
 - 14.9 Eliminar Producto Delete
 - 14.10 hacer una compra

1. INTRODUCCIÓN

Sudespacho.net es una aplicacion web para contratar servicios de diseño web, marketing, publicidad

En la parte de back-end tenemos dos tipos de usuarios si es un usuario admin si puede crear , modificar,borrar productos mediante los endpoint y si es un usuario normal se puede registrar loguear y ver solamente los productos o categorías o hacer una compra

Diagrama



2.Crear nuevo proyecto en laravel

Laravel es un framework PHP popular y poderoso para el desarrollo web, aqui los pasos necesarios para crear un nuevo proyecto ApiProbar en Laravel desde la línea de comandos

Crearemos un nuevo proyecto de Laravel utilizando tanto Composer como el instalador de Laravel: Instalación con Composer:

- Paso 1: Abrimos el terminal y navegamos hasta el directorio donde deseamos crear el proyecto Laravel.
- Paso 2: Ejecutamos el siguiente comando para crear un nuevo proyecto de Laravel

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

- Paso 3: Una vez que Composer haya terminado de instalar las dependencias, navegamos hasta el directorio del proyecto recién creado con el siguiente comando:

cd apiProbar

cuando estamos dentro del directorio **apiProbar** ejecutamos este comando
composer install

Este comando configura el proyecto para usar Sail como el entorno de desarrollo local. Esto implica que configura tu proyecto para que pueda ser ejecutado dentro de contenedores Docker administrados por Sail.

php artisan sail:install

El proceso de instalación nos preguntará qué tipo de base de datos vamos a utilizar para nuestro proyecto, así que seleccionamos mysql

npm install

Este comando crea un alias, ayudará de mucho para que escribamos los comandos mucho más cortos

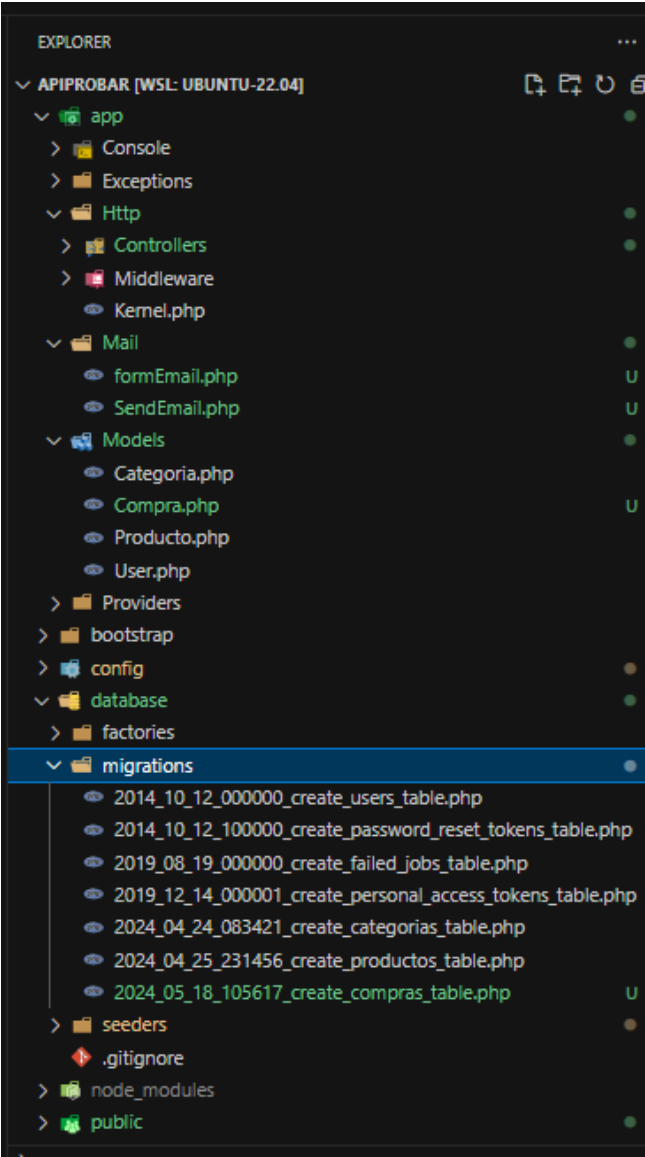
alias sail=./vendor/bin/sail

luego ejecutamos Este comando levantará el docker

sail up -d

2.1 Estructura del proyecto

Asi sera nuestra estructura y nuestros directorios y archivos más importantes para que nos ayuden a entender más el funcionamiento del framework



3.Laravel cheat sheet

Laravel generará automáticamente los archivos necesarios con la estructura básica y estos son los comandos utilizados en nuestra **ApiProbar**

alias sail=../vendor/bin/sail

sail up -d

Migrations

- sail artisan make:migration create_productos_table
- sail artisan make:migration create_categorias_table
- sail artisan make:migration create_compras_table
- sail artisan migrate

Seeders

- sail artisan make:seeder CategoriaSeeder
- sail artisan db:seed

Models

```
sail artisan make:model Producto
sail artisan make:model Categoria
sail artisan make:model Compra
```

Controllers

```
sail artisan make:controller ProductoController
sail artisan make:controller CategoriaController
sail artisan make:controller CompraController
sail artisan make:controller AuthController
```

Requests

```
sail artisan make:request ProductoRequest
sail artisan make:request AuthRequest
```

Tests

```
sail artisan make:test ProductoTest
sail artisan make:test LoginTest
sail test
```

4.Migraciones

Creamos las migración se debe de utilizar el siguiente comando de Artisan y en nuestro caso vamos a crear nuestras tablas ejecutamos estos comandos

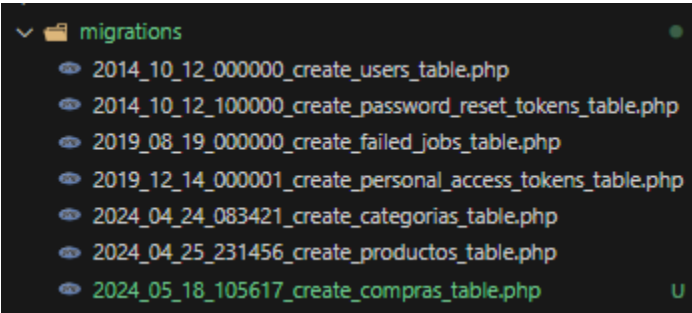
```
sail artisan make:migration create_productos_table
sail artisan make:migration create_categorias_table
sail artisan make:migration create_compras_table
```

4.1 Ejecutar las migraciones

Una vez creado y llenado nuestras migraciones, debemos ejecutar para poder generar el esquema de nuestra base de datos, para esto utilizamos el siguiente **comando de Artisan**.

sail artisan migration

Comienza el proceso de generar todas las migraciones creadas.



php artisan migrate : refresh

Si, por otro lado, queremos refrescar las migraciones

4.2 Categorias.php

Crea una migración que crea la tabla 'categorias' con dos columnas ('id' y 'nombre_categoria') y lo elimina si existe

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

return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::dropIfExists('categorias');

        Schema::create('categorias', function (Blueprint $table) {
            $table->id();

            $table->string('nombre_categoria'); // Agrega una columna para el nombre de la categoría

            $table->timestamps();//registrar la fecha y hora*/
        });

    }

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

4.3Productos.php

Creamos nuestra tabla con las características necesarias id -imagen- nombre..... y creamos tambien

una columna 'id_categoria' de tipo unsigned BigInteger que puede ser nula

Definimos una clave foránea en 'id_categoria' que referencia a 'id' en la tabla 'categorias'

Si una categoría es eliminada, también se eliminarán los productos asociados (onDelete('cascade')) y si la tabla productos existe lo eliminamos

```
<?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('productos', function (Blueprint $table) {
            $table->id();
            $table->string('imagen');
            $table->string('nombre');
            $table->text('descripcion');
            $table->unsignedBigInteger('id_categoria')->nullable();
            $table->integer('precio');
            $table->timestamps();

            $table->foreign('id_categoria')->references('id')->on('categorias')->onDelete('cascade');
        });
    }

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


4.4 Compras.php

Crea una migración que crea la tabla 'compras' para establecer la relación muchos a muchos entre las tablas 'user_id' y 'product_id'

// Crear una columna 'user_id' de tipo **unsignedBigInteger** para almacenar el ID del usuario y otra 'product_id' y Crear una columna 'action' de tipo string con un valor por defecto 'terminado'

// Definir la clave foránea 'user_id' que referencia la columna 'id' de la tabla 'users' y otra que referencia 'product_id'

```
<?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('compras', function (Blueprint $table) {
            $table->id();
            $table->unsignedBigInteger('user_id');
            $table->unsignedBigInteger('product_id');
            $table->string('action')->default('terminado');
            $table->foreign('user_id')->references('id')->on('users');
            $table->foreign('product_id')->references('id')->on('productos');
            $table->timestamps();
        });
    }

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

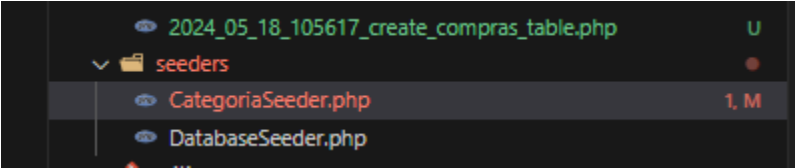
5. Seeders

Los seeders nos proporcionan un mecanismo para rellenar de datos las tablas de la base DATOS ejecutamos los siguientes comandos para crear los seeders

sail artisan make:seeder CategoriaSeeder

luego ejecutamos `sail artisan db:seed`

Al ejecutar este comando se generará los archivo dentro del directorio `database/seeds`.



5.1 DatabaseSeeder.php

Database Seeder Se encarga de llamar a otros seeders para poblar las tablas de la base de datos con datos iniciales e Importamos el modelo **Categoría** y luego genera tres instancias del modelo **Categoría**.

- crear y guardar estas instancias en la base de datos.

```
<?php
namespace Database\Seeders;
use App\Models\Categoria;
use Illuminate\Database\Seeder;
class DatabaseSeeder extends Seeder
{
    public function run(): void
    {
        Categoria::factory(3)->create();
    }
}
```

5.2 CategoriaSeeder.php

Debemos importar `\Illuminate\Support\Facades\DB` al principio del archivo Insertar datos iniciales en la tabla 'categorias' de la base de datos

```
<?php

namespace Database\Seeders;
use Illuminate\Database\Console\Seeds\WithoutModelEvents;
use Illuminate\Database\Seeder;
use App\Models\Categoria;
class CategoriaSeeder extends Seeder
{
    public function run(): void
    {
```

```
        Categoria::factory(3)->create();
    }
}

]);
}
```

6.Models

Ejecutamos los comandos siguientes para crear modelos

```
sail artisan make:model Producto
sail artisan make:model Categoria
sail artisan make:model Compra
```

6.1 Producto.php

Crear el modelo Producto definirlo con un atributo \$fillable, definimos la relación "belongsTo" entre los productos y las categorías sera muchos a muchos . Indica que un producto pertenece a una categoría.. y la relación "hasMany" entre los productos y las compras. uno a mucho

Indica que un producto puede tener varias compras asociadas.

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

class Producto extends Model
{
    use HasFactory;

    protected $fillable = [
        'imagen',
        'nombre',
        'descripcion',
        'id_categoria',
        'precio',
    ];

    public function categorias()
    {
```

```

        return $this->belongsTo(Categoria::class, 'id_categoria');
    }

    // Definir the relationship with Compra
    public function compras()

    {

        return $this->hasMany(Compra::class, 'product_id');
    }
}

```

6.2 Categoria.php

Definir la relación entre el producto y categoria que sera uno a mucho

```

<?php

namespace App\Models;

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

class Categoria extends Model
{
    use HasFactory;

    protected $fillable = [
        'nombre_categoria', // Agrega 'nombre_categoria' a la lista de atributos
        // llenables
    ];

    // Define una relación uno a muchos con el modelo Producto
    public function productos()
    {
        return $this->hasMany(Producto::class, 'id_categoria');
    }
}

```

6.3 Compra.php

Definir la relación entre el producto y la compra que sera mucho a mucho

```
<?php
namespace App\Models;

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

class Compra extends Model
{
    use HasFactory;
    // Definir the relationship with Producto

    public function producto()
    {
        return $this->belongsTo(Producto::class, 'product_id');
    }
}
```

6.4 User.php

Lo he modificado role en el modelo User

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

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

7. Controllers

Creamos los controladores ejecutamos el comando

```
sail artisan make:controller ProductoController
```

```
sail artisan make:controller CategoriaController
```

```
sail artisan make:controller ComprasController
```

sail artisan make:controller AuthController
sail artisan migrate

7.1 ProductoController.php

Este controlador maneja las operaciones CRUD (store, show, update, delete) y otras funcionalidades relacionadas con nuestro modelo Producto

Function index

Recupera todos los productos de la base de datos, incluyendo sus categorías asociadas, y devuelve los datos en formato JSON 200

Function store

Primero validar los datos del formulario antes de procesarlos desde el ProductoRequest.

luego se verifica si el usuario que realiza la solicitud tiene el rol de 'admin'. Si no lo tiene, se devuelve una respuesta JSON con un mensaje de error indicando que el usuario no está autorizado para realizar esta acción,

luego procesa manejar la creación de nuevos productos en la aplicación, asegurando que el usuario que realiza la solicitud tenga los permisos adecuados y manejando posibles errores durante el proceso de creación. si todo bien devuelve 201 y si hay un error devuelve el 500

Function update

Verificamos si el usuario que realiza la solicitud es un role admin. Luego, buscamos el producto por su ID en la base de datos. Si lo encuentra, procesa cualquier imagen proporcionada y actualiza los detalles del producto. Si todo va bien, devuelve una respuesta con el producto actualizado y un mensaje de éxito 200. En caso de errores, devuelve un mensaje de error junto con el código de estado correspondiente 500 Y SI EL PRODUCTO NO EXISTE HTTP 404 (Not Found).

Function Destory

Eliminar un producto de la base de datos.

Primero, valida si el usuario es admin. Luego, busca el producto por su ID. Si lo encuentra, lo elimina y devuelve un mensaje de éxito. Si no encuentra el producto, devuelve un mensaje de error. Si hay algún error durante el proceso, devuelve un mensaje de error interno del servidor

Function show

Recuperar los productos, incluyendo su categoría asociada, y devuelve los datos en formato JSON. Si el producto no se encuentra, devuelve un mensaje de error con un código de estado 404.

```

<?php

namespace App\Http\Controllers;
use App\Models\Producto;
use App\Http\Requests\ProductoRequest;
use Illuminate\Support\Facades\Auth;

use Illuminate\Http\Request;
class ProductoController extends Controller
{
    public function index()
    {
        $productos = Producto::with('categorias')->get();
        return response()->json($productos, 200);
    }

    public function store(ProductoRequest $request)
    {
        // Verificar si el usuario autenticado es un admin

        try {
            if(Auth::user()->role != 'admin' ){
                return response()->json(['error'=>'Usuario no autorizado']);
            }
            // Manejar la subida de imagen si se proporciona una imagen
            if ($request->hasFile('imagen')) {
                $imagen = $request->file('imagen');
                $imageName = time() . '.' .
$imagen->getClientOriginalExtension();
                $imagen->move(public_path('images'), $imageName);
            } else {
                $imageName = '';
            }
            // Crear un nuevo producto con los datos proporcionados

            $producto = Producto::create([
                'imagen' => $imageName,
                'nombre' => $request->nombre,
                'id_categoria' => $request->id_categoria,
                'descripcion' => $request->descripcion,
                'precio' => $request->precio
            ]);

```

```

        return response()->json($producto, 201);
    } catch (Exception $e) {
        return response()->json(['error' => $e->getMessage()], 500);
    }
}

public function update(ProductoRequest $request, $id)
{
    try {

        if(Auth()->user()->role != 'admin'){
            return response()->json(['error'=>'Usuario no
autorizado'],401);
        }

        // Buscar el producto por su ID

        $producto = Producto::find($id);

        if (!$producto) {
            return response()->json(['error' => 'Producto not found'],
404);
        }

        $imagenName = $producto->imagen; // Mantener la imagen existente si
no se proporciona una nueva

        if ($request->hasFile('imagen')) {
            $imagen = $request->file('imagen');
            $imagenName = time() . '.' .
$imagen->getClientOriginalExtension();
            $imagen->move(public_path('images'), $imagenName);
        }

        // Actualizar el producto con los nuevos datos

        $producto->update([
            'nombre' => $request->nombre,
            'id_categoria' => $request->id_categoria,
            'descripcion' => $request->descripcion,
            'precio' => $request->precio,
            'imagen' => $imagenName
        ]);

        return response()->json(['producto' => $producto, 'message' =>
'Producto modificado'], 200);
    } catch (\Exception $e) {

```



```

        return response()->json(['error' => $e->getMessage()], 500);
    }
}

public function destroy($id)
{
    try {

        if(Auth()->user()->role != 'admin')
        {
            return response()->json(['error'=>'Usuario no
autorizado'],401);
        }

        $producto = Producto::find($id);
        if ($producto)
        {
            $producto->delete();
            return response()->json(['message' => 'Producto Eliminado'],
200);
        } else
        {
            return response()->json(['error' => 'Producto not found'],
404);
        }
    } catch (\Exception $e) {
        return response()->json(['error' => $e->getMessage()], 500);
    }
}

public function show($id)
{
    $producto = Producto::with('categorias')->find($id);

    if (!$producto)
    {
        return response()->json(['message' => 'Producto not found'], 404);
    }

    return response()->json([
        'producto' => $producto,
        'nombre_categoria' => $producto->categorias->nombre_categoria,
    ]);
}
}

```

7.2 CategoriaController.php

// Se importa el modelo Categoria

Recuperar todas las categorías de la base de datos utilizando el método **all()** del modelo **Categoria**

```
<?php

namespace App\Http\Controllers;
use App\Models\Categoria;

use Illuminate\Http\Request;

class CategoriaController extends Controller
{
    public function index(){
        try {

            $categorias = Categoria::all();

            return response()->json(['categoria' => $categorias], 200);
        } catch (\Exception $e) {
            return response()->json(['message' => 'Error fetching categorias',
'error' => $e->getMessage()], 500);
        }
    }
}
```

7.3 CompraController.php

Primero importamos los namespaces necesarios para acceder a las clases relevantes en el código del controlador.

Esto incluye los modelos **User**, **Compra**, **Producto**, y las clases de correo electrónico **SendEmail** y **formEmail**. que estan la carpeta Mail

aqui en el controlador de compra

Obtener los datos de la solicitud, incluido el ID de usuario

// (\$user_id) y los IDs de los productos (\$producto_ids)

// buscamos el usuario en la base de datos con su ID

//luego verificar si es solamente un producto se crea la compra con id user y id producto en un atributo compra en un array asociativo

Luego Se envía un correo electrónico al usuario con detalles sobre el producto comprado he manejado [mailtrop.io](https://mailtrap.io)

y si son varios productos se crea y guarda las compras correspondientes a múltiples productos seleccionados en la solicitud, además de recuperar los detalles completos de cada producto seleccionado y almacenarlos en el array \$productos. luego guarda la compra y mande un correo al usuario

```
<?php

namespace App\Http\Controllers;
use App\Models\User;
use App\Models\Compra;
use App\Models\Producto;
use App\Mail\SendEmail;
use App\Mail\formEmail;
use Illuminate\Support\Facades\Mail;

use Illuminate\Http\Request;

class CompraController extends Controller
{
    public function store(Request $request)
    {

        try{
            //obtener los datos de la solicitud, incluido el ID de usuario
            // ($user_id) y los IDs de los productos ($producto_ids).
            $user_id = $request->user_id;
            $producto_ids = $request->producto_ids;

            // busca el usuario en la base de datos utilizando su ID
            $user = User::find($user_id);
            $email = $user->email;
            $name = $user->name;

            //verificar si es solamente un producto se crea la compra con id
            user y id producto en atributo de compra en un array asociativo
            if($producto_ids == 1){
                $compra = new Compra([
```

```

        'user_id'=>$user_id,
        'product_id'=>$producto_ids,
    ]);
    $productos = Producto::find($producto_id);

    // Se envía un correo electrónico al usuario con detalles sobre
    el producto comprado. Para esto, se utiliza la clase Mail de Laravel para enviar
    el correo electrónico y se pasa
    //al constructor de la clase sendEmail el nombre del usuario y
    los detalles del producto.

    Mail::to($email)->send(new sendEmail($name, $productos));
    return response()->json(['message' => 'productos añadidos '],
200);
}

}else{
    //inicializa un array vacío
    $productos = [];
    // itera sobre cada ID de producto en el array $producto_ids
    foreach ($producto_ids as $producto_id) {
        $producto = Producto::find($producto_id);
        $productos[] = $producto;
    }

    //iteración para crear y guardar las compras
    foreach ($producto_ids as $producto_id) {
        $compra = new Compra();
        $compra->user_id = $user_id;
        $compra->product_id = $producto_id;
        $compra->save();
    }

    // envía un correo electrónico al usuario con los detalles de
    los productos añadidos
    Mail::to($email)->send(new sendEmail($name, $productos));
    return response()->json(['message' => 'productos añadidos'],
200);
}

}

}catch(\Exception $e){

    return response()->json(['message' => $e->getMessage()], 500);

}

}

public function sendEmail(Request $request)
{

```

```

try {
    // Obtener datos de la solicitud
    $data =[
        'name'=>$request->name,
        'email'=>$request->email,
        'message'=>$request->mensaje,
        'categoria'=>$request->categoria,
        'tel'=>$request->tel,

    ];
    // Se utiliza la clase Mail de Laravel para enviar el correo electrónico

    Mail::to('tudespacho@gmail.com')->send(new formEmail($data));
    return response()->json(['message' => 'Email enviado correctamente']);
} catch (\Exception $e) {
    return response()->json(['message' => 'Error al enviar email', 'error'
=> $e->getMessage()], 500);
}
}
}

```

7.4 AuthController.php

sail composer require laravel/sanctum . Este comando utiliza Composer para instalar el paquete laravel/sanctum, que es el paquete que proporciona la funcionalidad de autenticación de tokens API en Laravel.

sail composer require laravel/sanctum

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

sail artisan make:controller AuthController

en la parte AuthController crearemos el registro el login y logout con los requisitos adecuados a cada endpoint

```

<?php

namespace App\Http\Controllers;
use App\Http\Requests\AuthRequest;

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

```

```

class AuthController extends Controller
{
    public function register(Request $request)
    {
        // Validar los datos de entrada

        //Crear un nuevo usuario
        $user = new User([
            'name' => $request->input('nombre_completo'),
            'email' => $request->input('correo_electronico'),
            'password' => Hash::make($request->input('contraseña')),
        ]);

        $user->save();

        return response()->json(['message' => 'Usuario registrado
 exitosadamente', 'user' => $user], 200);
    }

    public function login(Request $request)
    {
        try{

            $loginData = $request->validate([
                'email' => 'required|string|email',
                'password' => 'required|string',
            ]);

            if (!auth()->attempt($loginData)) {
                return response()->json(['message' => 'Credenciales inválidas'],
401);
            }

            $accessToken =
auth()->user()->createToken('authToken')->plainTextToken;

            return response()->json([
                'access_token' => $accessToken,
                'status' => true,
                'message' => 'Inicio de sesión válido',
                'user'=>auth()->user()
            ], 200);
        }
        catch (\Exception $e) {
            return response()->json(['error' => $e->getMessage()], 500);
        }
    }
}

```

```

    }

    // Validar los datos de entrada para el inicio de sesión

    public function logout() {
        auth()->user()->tokens()->delete();
        return response()->json(['message'=>'successfully logged out'],200);
    }

}

```

8.Requests

Crear los request para validar datos ejecutamos el comando siguiente
sail artisan make:request ProductoRequest

8.1 ProductoRequest.php

Aqui determina si el usuario tiene autorización para realizar la solicitud. En este caso, siempre devuelve true, y asegura que los datos proporcionados en la solicitud cumplan con ciertos requisitos antes de ser procesados por la aplicación

```

<?php
namespace App\Http\Requests;
use App\Http\Requests\ProductoRequest;
use Illuminate\Foundation\Http\FormRequest;
class ProductoRequest extends FormRequest
{
    public function authorize(): bool
    {
        return true;
    }

    * @return array<string,
    \Illuminate\Contracts\Validation\ValidationRule|array<mixed>|string
    */
    public function rules(): array
    {
        //cumplir los requisitos
        return [
            'imagen' => 'required',
            'nombre' => 'required|string|max:255',
            'descripcion' => 'required|string',
            'id_categoria' => 'required|integer',
            'precio' => 'required|integer|min:0',
        ];
    }
}

```

```
}  
}
```

8.2 AuthRequest.php

Aquí determina si el usuario tiene autorización para realizar la solicitud. En este caso, siempre devuelve true, validar los datos de registro y login con estos requisitos

```
<?php  
namespace App\Http\Requests;  
use Illuminate\Foundation\Http\FormRequest;  
  
class AuthRequest extends FormRequest  
{  
    public function authorize(): bool  
    {  
        return true;  
    }  
  
    * @return array<string,  
\Illuminate\Contracts\Validation\ValidationRule|array<mixed>|string>  
    */  
    public function rules(): array  
    {  
        return [  
            'nombre_completo' => 'required|string',  
            'correo_electronico' => 'required|email|unique:users,email',  
            'contraseña' => 'required|string|min:6',  
            'email' => 'required|string|email',  
            'password' => 'required|string',  
        ];  
    }  
}
```

10.Mail

php artisan make:mail FormEmail.php

php artisan make:mail SendEmail.php

primero lo configuramos el correo desde env.


```
MAIL_MAILER=smtp
MAIL_HOST=sandbox.smtp.mailtrap.io
MAIL_PORT=587
MAIL_USERNAME=3ef8ef971258f9
MAIL_PASSWORD=7515627d1043e6
MAIL_FROM_NAME="tudespacho"
MAIL_FROM_ADDRESS=tudespacho@gmail.com
```

10.1 formEmail.php

```
<?php

namespace App\Mail;

use Illuminate\Bus\Queueable;
use Illuminate\Contracts\Queue\ShouldQueue;
use Illuminate\Mail\Mailable;
use Illuminate\Mail\Mailables\Content;
use Illuminate\Mail\Mailables\Envelope;
use Illuminate\Queue\SerializesModels;

class formEmail extends Mailable
{
    use Queueable, SerializesModels;

    protected $data;

    /**
     * Create a new message instance.
     */
    public function __construct($data)
    {
        $this->data = $data;
    }

    public function envelope(): Envelope
    {
        return new Envelope(
            subject: 'Form Email',
        );
    }

    /**
     * Get the message content definition.
     */
    public function content(): Content
```

```

    {
        return new Content(
            view: 'form',
            with:[
                'data'=> $this->data,
            ]
        );
    }
}

    attachments for the message.
    *
    * @return array<int, \Illuminate\Mail\Mailables\Attachment>
    */
    public function attachments(): array
    {
        return [];
    }
}

```

10.2 sendEmail.php

```

<?php

namespace App\Mail;

use Illuminate\Bus\Queueable;
use Illuminate\Contracts\Queue\ShouldQueue;
use Illuminate\Mail\Mailable;
use Illuminate\Mail\Mailables\Content;
use Illuminate\Mail\Mailables\Envelope;
use Illuminate\Queue\SerializesModels;

class SendEmail extends Mailable
{
    use Queueable, SerializesModels;

    public $name;
    public $productos;
    public function __construct($name, $productos)
    {
        $this->name = $name;
        $this->productos = $productos;
    }
}

```

```
public function envelope(): Envelope
{
    return new Envelope(
        subject: 'Send Email',
    );
}

public function content(): Content
{
    return new Content(
        view: 'email',
        with:[
            'name'=>$this->name,
            'productos'=>$this->productos
        ]
    );
}

*
* @return array<int, \Illuminate\Mail\Mailables\Attachment>
*/
public function attachments(): array
{
    return [];
}
}
```

11 Resources

Esta parte cuando el usuario contrata un servicio le llega un correo con los datos del producto

Views

11.1 email.blade.php

al recibir el correo se muestra el producto contratado y sus detalles

```
<table cellpadding="0" cellspacing="0" align="left" class="es-left">
    <tbody>

@foreach($productos as $product)

    <tr>
```

```

                                <td
width="270" class="esd-container-frame es-m-p30b" align="center" valign="top">
                                <table
cellpadding="0" cellspacing="0" width="100%">

<tbody>

<tr>

<td align="center" class="esd-block-image" style="font-size: 0px;"><a
target="_blank"></a></td>

</tr>

<tr>

<td align="left" class="esd-block-text es-p10t es-p5b es-m-txt-1">

<h3>{{ $product->nombre }}</h3>

</td>

</tr>

<tr>

<td align="left" class="esd-block-text es-p5t es-p5b">

<p>{{ $product->descripcion }}</p>

</td>

</tr>

</tbody>

</table>

                                </td>
                                </tr>
                                @endforeach
                                </tbody>
                                </table>
                                </td>

```

11.2 form.blade.php

el formulario de pedir un presupuesto

```
<body>
  <h2>Pedir presupuesto</h2>
  <table>
    <tr>
      <th>Campo</th>
      <th>Datos</th>
    </tr>
    <tr>
      <td><strong>Nombre:</strong></td>
      <td>{{ $data['name'] }}</td>
    </tr>
    <tr>
      <td><strong>Email:</strong></td>
      <td>{{ $data['email'] }}</td>
    </tr>
    <tr>
      <td><strong>Phone:</strong></td>
      <td>{{ $data['tel'] }}</td>
    </tr>
    <tr>
      <td><strong>Categoría:</strong></td>
      <td>{{ $data['categoria'] }}</td>
    </tr>
    <tr>
      <td><strong>Mensaje:</strong></td>
      <td>{{ $data['message'] }}</td>
    </tr>
  </table>
```

12 .Routes

12.1 Api

En API definimos las Rutas de Registro y Inicio de Sesión cuando se envía una solicitud POST son Rutas Protegidas por Autenticación Sanctum

Este grupo de rutas está protegido por el middleware auth:sanctum, lo que significa que solo los usuarios autenticados pueden acceder a ellas.

Importamos tambien los controladores

```
<?php

use Illuminate\Http\Request;
use Illuminate\Support\Facades\Route;
use App\Http\Controllers\ProductoController;
use App\Http\Controllers\CategoriaController;
use App\Http\Controllers\CompraController;

use App\Http\Controllers\AuthController;

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

Route::post('/login', [AuthController::class, 'login']);
Route::post('/register', [AuthController::class,
'register'])->name('api.register');

Route::middleware('auth:sanctum')->group(function() {
    Route::post('/logout', [AuthController::class, 'logout']);
    Route::post('/productos', [ProductoController::class, 'store']);
    Route::put('/productos/{id}', [ProductoController::class, 'update']);
    Route::delete('/productos/{id}', [ProductoController::class, 'destroy']);
});

Route::get('/productos', [ProductoController::class, 'index']);
Route::get('/producto/{id}', [ProductoController::class, 'show']);

Route::get('/categorias', [CategoriaController::class, 'index']);

Route::post('/compras', [CompraController::class, 'store']);
Route::post('/sendEmail', [CompraController::class, 'sendEmail']);

Route::get('/not-authorized', function() {
    return response()->json(['error'=>'Usuario no autorizado'], 401);
})->name('not-authorized');
```

13.Tests

LOS TEST nos permiten verificar que nuestro código funciona según lo esperado. Verificar la funcionalidad Y Detectar errores para probarlo
primero creamos los test con el comando

sail artisan make:test LoginTest

sail artisan make:test ProductoTest

13.1 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 = [
            'nombre_completo' => 'John Doe',
            'correo_electronico' => 'john@example.com',
            'contraseña' => 'password123',
        ];

        // Enviar una solicitud POST a la ruta 'api/register' con los datos del
        usuario

        $response = $this->postJson('api/register', $userData);

        // Verificar que la respuesta tenga un estado 200 (OK)

        $response->assertStatus(200)
            ->assertJsonStructure([
                'user' => [
                    'id',
                    'name',
                    'email',
                    'created_at',
                    'updated_at',
```

```

        ],
        'message',
    ));

    // Verificar que la base de datos tenga un registro en la tabla 'users'
    con los datos proporcionados

    $this->assertDatabaseHas('users', [
        'name' => $userData['nombre_completo'],
        'email' => $userData['correo_electronico'],
    ]);
}

public function test_login()
{
    // Crear un usuario utilizando una fábrica con un correo
    electrónico y contraseña específicos

    $user = User::factory()->create([
        'email' => 'john@example.com',
        'password' => Hash::make('password123'),
    ]);

    $response = $this->postJson('/api/login', [
        'email' => $user->email,
        'password' => 'password123',
    ]);
    $response->assertStatus(200)
        ->assertJsonStructure([
            'user' => [
                'id',
                'name',
                'email',
                'created_at',
                'updated_at',
            ],
            'message',
            'access_token',
            'status',
        ]);
}

public function testLogout()
{
    // Crear un usuario utilizando una fábrica

```



```

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

        // (autenticar al usuario)
        Sanctum::actingAs($user);

        $response = $this->postJson('/api/logout');
        $response->assertStatus(200);

        // Verificar que el usuario no tenga tokens activos en la base
de datos

        $this->assertCount(0, $user->tokens);
    }

}

```

13.2ProductoTest.php

```

<?php

namespace Tests\Feature;

use Illuminate\Foundation\Testing\RefreshDatabase;
use Illuminate\Http\UploadedFile;
use Illuminate\Support\Facades\Storage;
use Tests\TestCase;
use App\Models\Producto;
use App\Models\Categoria;
use App\Models\User;
use Database\Factories\ProductoFactory;

class ProductoTest extends TestCase
{
    use RefreshDatabase;

    public function test_store_producto()
    {
        // Crear un usuario con el rol de 'admin' usando una factory
        $user = User::factory()->create();
        $user->role = 'admin';

        // Crear una categoría usando una factory
        $categoria = Categoria::factory()->create();
    }
}

```

```

        // Generar un token para el usuario
        $token = $user->createToken('TestToken')->plainTextToken;

        // Preparar los datos del producto
        $data = [
            'nombre' => 'Producto de prueba',
            'id_categoria' => $categoria->id,
            'descripcion' => 'Descripción de prueba',
            'precio' => 100,
            'imagen' => UploadedFile::fake()->image('producto.jpg')
        ];

        // Enviar una solicitud POST al endpoint api/productos con los datos del
        producto, incluyendo el token de autenticación

        $response = $this->actingAs($user)->postJson('api/productos', $data, [
            'Authorization'=>'Bearer'.$token
        ]);

        $response->assertStatus(201);

        $this->assertDatabaseHas('productos', [
            'nombre' => 'Producto de prueba',
            'id_categoria' => $categoria->id,
            'descripcion' => 'Descripción de prueba',
            'precio' => 100,
        ]);

        $producto = Producto::first();

        // Asegurar que la respuesta coincida con el producto creado
        $response->assertJson([
            'id' => $producto->id,
            'imagen' => $producto->imagen,
            'nombre' => 'Producto de prueba',
            'id_categoria' => $categoria->id,
            'descripcion' => 'Descripción de prueba',
            'precio' => 100
        ]);
    }

    public function test_update_producto() {

        $categoria = Categoria::factory()->create();

```

```

$user = User::factory()->create();
$user->role = 'admin';
$token = $user->createToken('TestToken')->plainTextToken;

$producto = Producto::create([
    'nombre' => 'Producto de prueba',
    'id_categoria' => $categoria->id,
    'descripcion' => 'Descripción de prueba',
    'precio' => 100,
    'imagen' => UploadedFile::fake()->image('producto.jpg')
]);

$test =[
    'nombre' => 'New Product Name',
    'id_categoria' => $categoria->id,
    'descripcion' => 'New Description',
    'precio' => 200,
    'imagen' => "test.png"
];

$response= $this->actingAs($user)->putJson('api/productos/'.
$producto->id,$test,[
    'Authorization'=>'Bearer'.$token
]);

$response->assertStatus(200);

}

public function test_destroy_producto()
{
    // Crear un producto
    $categoria = Categoria::factory()->create();
    // Preparar los datos del producto
    $user = User::factory()->create();
    $user->role = 'admin';
    $token = $user->createToken('TestToken')->plainTextToken;

    $producto = Producto::create([
        'nombre' => 'Producto de prueba',
        'id_categoria' => $categoria->id,
        'descripcion' => 'Descripción de prueba',
        'precio' => 100,
        'imagen' => UploadedFile::fake()->image('producto.jpg')
    ]);

```

```

        // Enviar una solicitud DELETE al endpoint para eliminar el producto

$response= $this->actingAs($user)->delete('api/productos/'. $producto->id,[
    'Authorization'=>'Bearer'.'.$token
]);

        // Verificar que la solicitud fue exitosa (código de estado 200)
$response->assertStatus(200);
    }

    public function test_show_producto()
    {
        $categoria = Categoria::factory()->create();
        $producto = Producto::create([
            'nombre' => 'Producto de prueba',
            'id_categoria' => $categoria->id,
            'descripcion' => 'Descripción de prueba',
            'precio' => 100,
            'imagen' => UploadedFile::fake()->image('producto.jpg')
        ]);

        $response = $this->getJson('/api/producto/'.$producto->id);

        // Verificar que la solicitud fue exitosa
$response->assertStatus(200);
    }

}

```

Prueba de los test

```
• safaa@DESKTOP-HGU6CRC:~/apiprobar$ sail test

PASS Tests\Unit\ExampleTest
✓ that true is true

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

PASS Tests\Feature\LoginTest
✓ user registration
✓ login
✓ logout

PASS Tests\Feature\ProductoTest
✓ store producto
✓ update producto
✓ destroy producto
✓ show producto

Tests: 9 passed (29 assertions)
Duration: 1.42s
```

14.Postamnn los Endpoint 14.1 Register Post

Despacho / register

POST

http://localhost/api/register

Params

Authorization

Headers (10)

Body

Scripts

Tests

Se

☐ none

☐ form-data

☐ x-www-form-urlencoded

☒ raw

☐ binary

☐ Gra

1 {

2 | "nombre_completo": "safaa-feqri",

3 | "correo_electronico": "safaa@gmail.com",

4 | "contraseña": "usuario"

5 |

6 } }

Body

Cookies

Headers (9)

Test Results (1/1)

Pretty

Raw

Preview

Visualize

JSON

1 {

2 | "message": "Usuario registrado exitosamente",

3 | "user": {

4 | "name": "safaa feqri",

5 | "email": "safaa@gmail.com",

6 | "updated_at": "2024-06-05T07:08:04.000000Z",


7 | "created_at": "2024-06-05T07:08:04.000000Z",

8 | "id": 12

9 | }

10 } }

Registro admin

 Despacho / register

POST

▼

http://localhost/api/register

Params

Authorization ●

Headers (10)

Body ●

Scripts ●

Tests ●

☐ none

☐ form-data

☐ x-www-form-urlencoded

☒ raw

☐ binary

☐ C

1 {

2 "nombre_completo": "administrador",

3 "correo_electronico": "admin@admin.com",

4 "contraseña": "adminadmin"

5 }

6 }

Body

Cookies

Headers (9)

Test Results (1/1)


Pretty

Raw

Preview

Visualize

JSON ▼



1 {

2 "message": "Usuario registrado exitosamente",

3 "user": {

4 "name": "administrador",

5 "email": "admin@admin.com",

6 "updated_at": "2024-06-05T07:11:13.000000Z",

7 "created_at": "2024-06-05T07:11:13.000000Z",

8 "id": 13

9 }

10 }

id	name	email	email_verified_at	password	role	remember_tok
12	safaa feqri	safaa@gmail.com	NULL	\$2y\$12\$...	member	NULL
13	administrador	admin@admin.com	NULL	\$2y\$12\$...	admin	NULL

14.2 Login Post

Verificación de usuario y contraseña

-Usuario o contraseña incorrecta control de errores en el caso Falta pj

.com al correo

38

POST

http://localhost/api/login

ParamsAuthorizationHeaders (11)BodyScriptsTest

☐ none

☐ form-data

☐ x-www-form-urlencoded

☒ raw

☐ binary

1

{

2

"email": "admin@admin",

3

"password": "adminadmin"

4

}

5

BodyCookies (2)Headers (9)Test Results (0/1)

PrettyRawPreviewVisualizeJSON

1

{

2

"message": "Usuario o contraseña Incorrecto"

3

}

Caso Todo correcto

Despacho / login

POST

http://localhost/api/login

ParamsAuthorizationHeaders (11)BodyScriptsTests

☐ none

☐ form-data

☐ x-www-form-urlencoded

☒ raw

☐ binary

1

{

2

"email": "admin@admin.com",

3

"password": "adminadmin"

4

}

5

BodyCookies (2)Headers (9)Test Results (1/1)

PrettyRawPreviewVisualizeJSON

1

{

2

"access_token": "36|di0l9cihy7gZhHiatk17xZ034QDE5k",

3

"status": true,

4

"message": "Inicio de sesión válido",

5

"user": {

6

"id": 13,

7

"name": "administrador",

8

"email": "admin@admin.com",

9

"email_verified_at": null,

10

"role": "admin",

11

"created_at": "2024-06-05T07:11:13.000000Z",

12

"updated_at": "2024-06-05T07:11:13.000000Z"

13

}

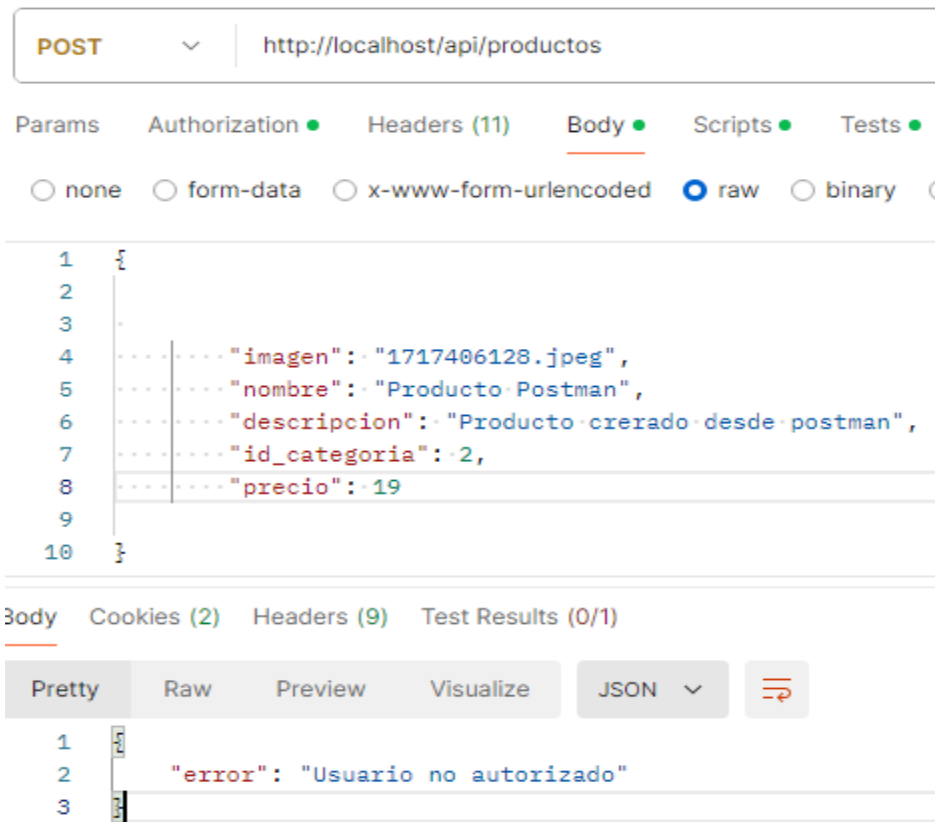
}

Una vez el usuario ya esta logueado vamos a crear un producto metiendo el token en authorization y seleccionar Bearer Token Y PEGAR el token que se ha generado en login adjunto capturas de pantalla

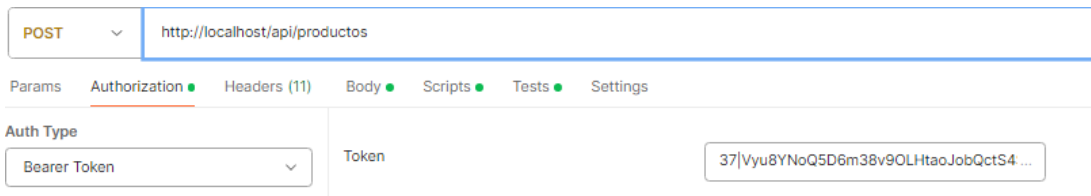
14.3 Crear producto POST

Al crear un producto si no es un admin autorizado mediante el token no se puede crear y modificar y borrar un producto

Caso No autorizado



Caso autorizado ahora cogemos el token de admin



POST

▼

http://localhost/api/productos

ParamsAuthorization ●Headers (11)Body ●Scripts ●Tests ●Settings

☐ none

☐ form-data

☐ x-www-form-urlencoded

☒ raw

☐ binary

☐ GraphQL

JSON

```
1 {
2
3
4     .... "imagen": "imagenPostman.jpeg",
5     .... "nombre": "productos desde postman",
6     .... "descripcion": "Incluimos productos desde postman",
7     .... "id_categoria": 1,
8     .... "precio": 20
9     ....
10 }
```

BodyCookies (2)Headers (9)Test Results (1/1)

PrettyRawPreviewVisualizeJSON ▼



```
1 {
2     "imagen": "",
3     "nombre": "productos desde postman",
4     "id_categoria": 1,
5     "descripcion": "Incluimos productos desde postman",
6     "precio": 20,
7     "updated_at": "2024-06-05T08:49:57.000000Z",
8     "created_at": "2024-06-05T08:49:57.000000Z",
9     "id": 13
10 }
```

14.4 Modificar Producto PUT

PUT

http://localhost/api/productos/12

Params

Authorization

Headers (11)

Body

Scripts

Tests

Settings

none

form-data

x-www-form-urlencoded

raw

binary

GraphQL

1

{

2

-

3

....."imagen": "ImagenPostmanModificado.jpeg",

4

....."nombre": "Producto MODIFICADO Postman",

5

....."descripcion": "Producto creado desde postman test MODIFI

6

....."id_categoria": 1,

7

....."precio": 20

8

}

9

Body

Cookies (2)

Headers (9)

Test Results (1/1)

Pretty

Raw

Preview

Visualize

JSON

1

{

2

"producto": {

3

"id": 12,

4

"imagen": "",

5

"nombre": "Producto MODIFICADO Postman",

6

"descripcion": "Producto creado desde postman test MODIFI

7

"id_categoria": 1,

8

"precio": 20,

9

"created_at": "2024-06-05T08:07:10.000000Z",

10

"updated_at": "2024-06-05T08:08:07.000000Z"

11

},

12

"message": "Producto modificado"

13

}

1.

14.4Obtener todas las categorías

GET

http://localhost/api/categorias/

Params

Authorization

Headers (11)

Body

Scripts

Tests

Setti

none

form-data

x-www-form-urlencoded

raw

binary

Grapt

1

{

2

3

...

4

.....

5

}

body

Cookies (2)

Headers (9)

Test Results (1/1)

Pretty

Raw

Preview

Visualize

JSON

1

{

2

"categoria": [

3

{

4

"id": 1,

5

"nombre_categoria": "Diseño Web",

6

"created_at": "2024-06-02T12:34:24.000000Z",

7

"updated_at": "2024-06-02T12:34:24.000000Z"

8

},

9

{

10

"id": 2,

11

"nombre_categoria": "Marketing y Publicidad",

12

"created_at": "2024-06-02T12:34:24.000000Z",

13

"updated_at": "2024-06-02T12:34:24.000000Z"

14

},

15

{

16

"id": 3,

17

"nombre_categoria": "Seo y Posicionamiento",

18

"created_at": "2024-06-02T12:34:24.000000Z",

19

"updated_at": "2024-06-02T12:34:24.000000Z"

20

}

21

]

22

}

14.6 Obtener todos los Productos GET

GET

http://localhost/api/productos/

Params

Authorization

Headers (11)

Body

Scripts

Tests

Settings

Body

Cookies (2)

Headers (9)

Test Results (1/1)

Pretty

Raw

Preview

Visualize

JSON

```
48     }
49   },
50   {
51     "id": 6,
52     "imagen": "",
53     "nombre": "test",
54     "descripcion": "Incluimos admin.",
55     "id_categoria": 1,
56     "precio": 20,
57     "created_at": "2024-06-03T21:37:56.000000Z",
58     "updated_at": "2024-06-03T21:37:56.000000Z",
59     "categorias": {
60       "id": 1,
61       "nombre_categoria": "Diseño Web",
62       "created_at": "2024-06-02T12:34:24.000000Z",
63       "updated_at": "2024-06-02T12:34:24.000000Z"
64     }
65   },
66   {
67     "id": 12,
68     "imagen": "",
69     "nombre": "Producto MODIFICADO Postman",
70     "descripcion": "Producto creado desde postman test MODIFICADO",
71     "id_categoria": 1,
72     "precio": 20,
73     "created_at": "2024-06-05T08:07:10.000000Z",
74     "updated_at": "2024-06-05T08:08:07.000000Z",
75     "categorias": {
76       "id": 1,
77       "nombre_categoria": "Diseño Web",
78       "created_at": "2024-06-02T12:34:24.000000Z",
79       "updated_at": "2024-06-02T12:34:24.000000Z"
80     }
81   }
82 ]
```

14.7Obtener un Producto por su ID

GET

▼

http://localhost/api/producto/2

Params

Authorization

Headers (9)

Body ●

Scripts

Tests

Settings

none

form-data

x-www-form-urlencoded

raw

binary

GraphQL

1

2

Body

Cookies (2)

Headers (9)

Test Results

Pretty

Raw

Preview

Visualize

JSON ▼

1

{

2

"producto": {

3

"id": 2,

4

"imagen": "1717406128.jpeg",

5

"nombre": "Diseño Web modificado",

6

"descripcion": "Incluimos hasta 20 páginas en el diseño de tu negocio y servicios.",

7

"id_categoria": 1,

8

"precio": 19,

9

"created_at": "2024-06-03T08:30:31.000000Z",

10

"updated_at": "2024-06-04T16:11:05.000000Z",

11

"categorias": {

12

"id": 1,

13

"nombre_categoria": "Diseño Web",

14

"created_at": "2024-06-02T12:34:24.000000Z",

15

"updated_at": "2024-06-02T12:34:24.000000Z"

16

}

17

},

18

"nombre_categoria": "Diseño Web"

14.8 Eliminar Producto Delete

Ahora borramos el producto creado con id 12

DELETE

http://localhost/api/productos/12

Params

Authorization

Headers (11)

Body

Scripts

Tests

Settings

none

form-data

x-www-form-urlencoded

raw

binary

GraphQL

1 {

2 -

3

4

5 }

Body

Cookies (2)

Headers (9)

Test Results (1/1)

Pretty

Raw

Preview

Visualize

JSON

1 {

2 "message": "Producto Eliminado"

3 }

Ahora Cerramos la sesión de admin

POST

http://localhost/api/logout

Params

Authorization

Headers (10)

Body

Scripts

Tests

Setti

none

form-data

x-www-form-urlencoded

raw

binary

Gra

1

body

Cookies (2)

Headers (9)

Test Results (1/1)

Pretty

Raw

Preview

Visualize

JSON

1 {

2 "message": "Sesion Cerrada"

3 }

Conectamos con el login usuario

caso incorrecto

POST

http://localhost/api/login

Params

Authorization

Headers (11)

Body

Scripts

Tests

none

form-data

x-www-form-urlencoded

raw

binary

1

{

2

3

... "email": "safaa@gmail.com",

4

... "password": "usuario1"

5

}

body

Cookies (2)

Headers (9)

Test Results (0/1)

Pretty

Raw

Preview

Visualize

JSON

1

{

2

"message": "Usuario o contraseña Incorrecto"

3

}

caso correcto

POST

http://localhost/api/login

Params

Authorization

Headers (11)

Body

Scripts

Tests

Settings

none

form-data

x-www-form-urlencoded

raw

binary

GraphQL

JSON

1

{

2

3

"email": "safaa@gmail.com",

4

"password": "usuario"

5

}

Body

Cookies (2)

Headers (9)

Test Results (1/1)

Pretty

Raw

Preview

Visualize

JSON

1

{

2

"access_token": "38|sbWZP1Fi0qn3sbcQ5MS0yAOLB5sksi2o1Qbvwem820732014",

3

"status": true,

4

"message": "Inicio de sesión válido",

5

"user": {

6

"id": 12,

7

"name": "safaa feqri",

8

"email": "safaa@gmail.com",

9

"email_verified_at": null,

10

"role": "member",

11

"created_at": "2024-06-05T07:08:04.000000Z",

12

"updated_at": "2024-06-05T07:08:04.000000Z"

13

}

14

}

GET

http://localhost/api/productos

Params

Authorization

Headers (11)

Body

Scripts

Tests

Settings

Auth Type

Bearer Token

Token

38|sbWZP1Fi0qn3sbcQ5MS0yAOLB5sksi2...

pasamos el token luego hacemos un get al producto eso sin o con token si puede ver

GET

http://localhost/api/productos

Params

Authorization

Headers (11)

Body

Scripts

Tests

Settings

Auth Type

Bearer Token

Token

3B|sbWZP1FiOqn3sbcQ5MSOyAOLB5sksi2...

The authorization header will be automatically generated when you send the request. Learn more about [Bearer Token](#) authorization.

body

Cookies (2)

Headers (9)

Test Results (1/1)

Status: 200 OK

Time: 38 ms

Pretty

Raw

Preview

Visualize

JSON

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

"imagen": "1717406128.jpeg",

"nombre": "Diseño Web modificado",

"descripcion": "Incluimos hasta 20 páginas en el diseño de su sitio web, asegurando suficiente espacio para presentar

"id_categoria": 1,

"precio": 19,

"created_at": "2024-06-03T08:30:31.000000Z",

"updated_at": "2024-06-04T16:11:05.000000Z",

"categorias": {

"id": 1,

"nombre_categoria": "Diseño Web",

"created_at": "2024-06-02T12:34:24.000000Z",

"updated_at": "2024-06-02T12:34:24.000000Z"

}

},

{

"id": 3,

"imagen": "1717406137.jpg",

"nombre": "Publicidad y Marketing",

"descripcion": "Aumenta tus ventas informando al comprador sobre qué productos necesita en su carrito para acceder a u

"id_categoria": 3,

"precio": 19,

"created_at": "2024-06-03T08:33:19.000000Z",

"updated_at": "2024-06-03T09:41:18.000000Z",

}

ahora a ver con el token intentar crear o modificar o eliminar un producto mediante un usuario miembro no es un admin

POST

http://localhost/api/productos

Params

Authorization

Headers (11)

Body

Scripts

Tests

Settings

none

form-data

x-www-form-urlencoded

raw

binary

GraphQL

JSON

1

2

3

4

5

6

7

8

9

{

...

"imagen": "usuario.jpeg",

"nombre": "Diseño Web modificado",

"descripcion": "Incluimos hasta 20 páginas en el diseño de su sitio web, a

"id_categoria": 1,

"precio": 19

}

body

Cookies (2)

Headers (9)

Test Results (1/1)

Pretty

Raw

Preview

Visualize

JSON

1

2

3

{

"error": "Usuario no autorizado"

}

PUT

▼

http://localhost/api/productos/2

Params

Authorization ●

Headers (11)

Body ●

Scripts ●

Tests ●

Setti

☐ none

☐ form-data

☐ x-www-form-urlencoded

☒ raw

☐ binary

☐ Grapt

1

{

2

...

3

"imagen": "usuariomodificar.jpeg",

4

"nombre": "Diseño Web modificado",

5

"descripcion": "Incluimos hasta 20 páginas en el diseño

6

"id_categoria": 1,

7

"precio": 19

8

}

9

ody

Cookies (2)

Headers (9)

Test Results (0/1)

Pretty

Raw

Preview

Visualize

JSON ▼

⌵

1

{

2

"error": "Usuario no autorizado"

3

}

DELETE

▼

http://localhost/api/productos/2

Params

Authorization ●

Headers (11)

Body ●

Scripts ●

Tes

☐ none

☐ form-data

☐ x-www-form-urlencoded

☒ raw

☐ binai

1

{

2

...

3

"imagen": "usuariomodificar.jpeg",

4

"nombre": "Diseño Web modificado",

5

"descripcion": "Incluimos hasta 20 páginas en

6

"id_categoria": 1,

7

"precio": 19

8

}

9

ody

Cookies (2)

Headers (9)

Test Results (0/1)

Pretty

Raw

Preview

Visualize

JSON ▼

⌵

1

{


2

"error": "Usuario no autorizado"

3

}

14.9Añadir Producto para comprar

 http://localhost/api/compras

POST

▼

http://localhost/api/compras

Params

Authorization

Headers (9)

Body ●

Scripts

Tests

Settings

☐ none

☐ form-data

☐ x-www-form-urlencoded

☒ raw

☐ binary

☐ GraphQL

```
1  {
2    ... "user_id": "12",
3    ... "producto_ids": [3]
4  }
5
```

Body

Cookies (2)

Headers (9)

Test Results


Pretty

Raw

Preview

Visualize

JSON ▼



```
1  {
2    "message": "productos añadidos"
3  }
```