

Desarrollo y Configuración de los Microservicios para “Perfulandia”

Integrantes : Janet Huaylla Huayllas
Diego Hidalgo

Profesor: Ruben Badilla Aquino

Seccion: 09D

Fecha de entrega: 25/05/2025

ÍNDICE DE CONTENIDO

Introducción.....	3
1.1 Planificación del backend.....	3
1.2 Estructura del proyecto.....	4
Este archivo define la configuración y dependencias del proyecto Spring Boot:.....	5
1. Entidad: Producto.java.....	6
2. Repositorio: ProductoRepository.java.....	6
Interfaz que permite acceder a la base de datos usando JPA.....	6
3. Servicio: ProductoService.java.....	6
4. Controlador: ProductoController.java.....	7
2.2 Configuración del archivo application.properties.....	11
2.3 Configuración de la Base de Datos MySQL.....	11
2.4 Peticiones Postman – CRUD de Productos.....	12
1. Crear Producto.....	12
2. Obtener Todos los Productos.....	13
3. Obtener Producto por ID.....	14
Enlace al repositorio GitHub.....	18
Enlace al repositorio: https://github.com/janet0u0/backend-perfulandia.git.....	18
Tabla de integrantes:.....	18
Conclusión.....	20
Bibliografía.....	21

Introducción

Este informe presenta el desarrollo de un sistema backend funcional basado en Spring Boot y Maven, como parte del proyecto grupal para Perfulandia SPA. El objetivo es aplicar conocimientos sobre microservicios, arquitectura REST, persistencia de datos con JPA, y pruebas con Postman, cumpliendo con los criterios definidos en la rúbrica oficial.

A continuación se describen el diseño del proyecto, la configuración técnica, el uso del control de versiones, el desarrollo colaborativo y las pruebas funcionales de los microservicios desarrollados.

1. Diseño del proyecto y línea base de trabajo.

Para abordar los problemas identificados en el sistema monolítico de Perfulandia SPA, se inició el desarrollo de un nuevo backend utilizando el framework Spring Boot y el sistema de construcción Maven. Este enfoque permite organizar el código de forma modular, escalable y alineado con una arquitectura basada en microservicios.

1.1 Planificación del backend

Se definió una planificación preliminar del proyecto dividida en cinco etapas:

- Inicialización del proyecto: configuración del proyecto con Spring Boot y Maven.
- Diseño de entidades y base de datos: definición de clases de modelo y relaciones.
- Desarrollo de operaciones CRUD: implementación de servicios RESTful para entidades clave (Cliente, Producto, Pedido).
- Pruebas con Postman: validación de endpoints para asegurar su correcto funcionamiento.
- Documentación e informe: consolidación de pruebas, capturas y explicación técnica.

Además, como línea base de trabajo, se estableció una estructura modular basada en paquetes separados por capas (controller, service, repository, model). Se definieron convenciones de nombres uniformes, el uso obligatorio de anotaciones de Spring para delimitar responsabilidades, y se optó por utilizar Spring Boot 3.5.0 con Java 21 como estándar del proyecto. Las dependencias se gestionaron exclusivamente con Maven, asegurando la trazabilidad del proyecto desde su inicio.

1.2 Estructura del proyecto

El backend de Perfulandia SPA fue generado a partir de Spring Initializr con las siguientes configuraciones:

- **Spring Web:** Permite crear aplicaciones web y API REST usando Spring MVC. Incluye servidor Tomcat embebido.
- **Spring Data JPA:** Facilita el acceso a bases de datos SQL usando JPA y Hibernate para mapear objetos Java a tablas.
- **MySQL Driver:** Controlador JDBC para conectar tu aplicación con bases de datos MySQL.
- **Lombok:** Biblioteca que reduce el código repetitivo como getters, setters y constructores con anotaciones.
- **Spring Boot DevTools:** Herramientas para mejorar el desarrollo, como recarga automática de la app al guardar cambios

Dependencies

ADD DEPENDENCIES... CTRL + B

Spring Web WEB

Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

Lombok DEVELOPER TOOLS

Java annotation library which helps to reduce boilerplate code.

Spring Data JPA SQL

Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.

MySQL Driver SQL

MySQL JDBC driver.

Spring Boot DevTools DEVELOPER TOOLS

Provides fast application restarts, LiveReload, and configurations for enhanced development experience.

El backend fue estructurado respetando las buenas prácticas de desarrollo con Spring Boot, aplicando una separación clara de responsabilidades mediante los siguientes paquetes:

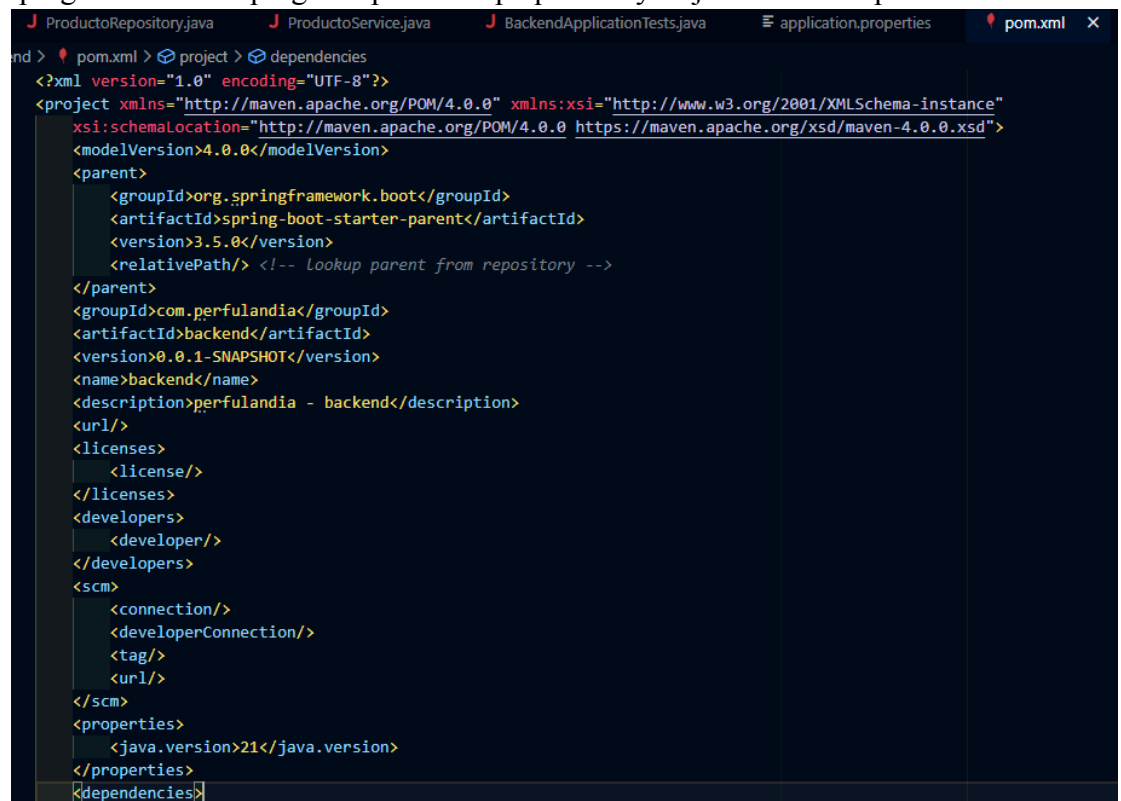
- **controller:** contiene las clases responsables de recibir y responder a las solicitudes HTTP.
- **service:** encapsula la lógica de negocio.
- **repository:** gestiona la interacción con la base de datos mediante Spring Data JPA.
- **model:** incluye las entidades del dominio de Perfulandia (como Cliente, Producto, Pedido, etc.).

Configuración del archivo pom.xml (Maven)

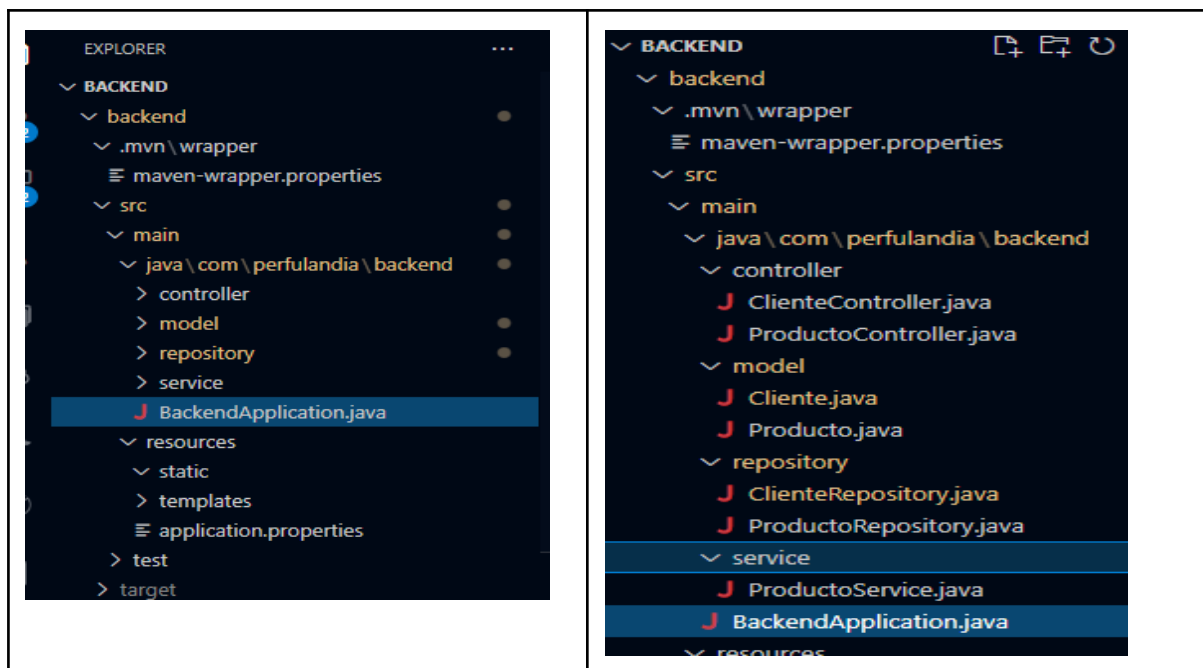
Este archivo define la configuración y dependencias del proyecto Spring Boot:

- **Parent:** Usa como base spring-boot-starter-parent versión 3.5.0.
- **Proyecto:** Grupo com.perfulandia, artefacto backend, versión 0.0.1-SNAPSHOT.
- **Java:** Usa la versión 21.
- **Dependencias:**
 - spring-boot-starter-data-jpa: para operaciones con base de datos.
 - Spring-boot-starter-web: para crear APIs REST.
 - mysql-connector-j: para conectarse a una base de datos MySQL.
 - lombok: para simplificar el código eliminando la necesidad de escribir getters, setters, etc.
 - spring-boot-devtools: para recarga automática en desarrollo.
 - spring-boot-starter-test: para pruebas.
- **Plugins:**

- maven-compiler-plugin: para compilar el proyecto y procesar anotaciones de Lombok.
- spring-boot-maven-plugin: para empaquetar y ejecutar la aplicación.+



```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>4.0.0</version>
        <relativePath/> <!-- Lookup parent from repository -->
    </parent>
    <groupId>com.perfulandia</groupId>
    <artifactId>backend</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>backend</name>
    <description>perfulandia - backend</description>
    <url/>
    <licenses>
        <license/>
    </licenses>
    <developers>
        <developer/>
    </developers>
    <scm>
        <connection/>
        <developerConnection/>
        <tag/>
        <url/>
    </scm>
    <properties>
        <java.version>21</java.version>
    </properties>
    <dependencies>
```



1. **Entidad:** Producto.java

Define la estructura de los datos que se almacenarán en la base de datos.

- Anotada con `@Entity` (JPA).
- Atributos: id, nombre, descripcion, precio, stock.
- Usa `@Data` de Lombok para generar automáticamente getters, setters, `toString()` y otros métodos útiles.

Objetivo: Representar un producto como una tabla en la base de datos.

2. **Repositorio:** ProductoRepository.java

Interfaz que permite acceder a la base de datos usando JPA

- Extiende `JpaRepository<Producto, Long>`.
- Spring Boot genera automáticamente métodos como `findAll()`, `save()`, `findById()`, y `deleteById()`.

Objetivo: Acceder y manipular datos sin escribir código SQL manual.

3. **Servicio:** ProductoService.java

Contiene la lógica de negocio del backend.

- Métodos: `listarProductos()`, `obtenerProductoPorId()`, `guardarProducto()`, `eliminarProducto()`.
- Utiliza el repositorio para acceder a la base de datos.
- Centraliza las operaciones de producto.

Objetivo: Desacoplar la lógica del negocio del controlador y organizar el backend.

4. **Controlador:** ProductoController.java

Define los endpoints REST para consumir el servicio desde Postman o frontend.

- Ruta base: `/api/productos`.
- Métodos:
 - GET: Listar todos o buscar por ID.
 - POST: Crear un nuevo producto.
 - PUT: Actualizar un producto existente.
 - DELETE: Eliminar un producto por ID.

Objetivo: Exponer los servicios CRUD como una API REST.

```
nd > src > main > java > com > perfulandia > backend > model > J Producto.java > ...
package com.perfulandia.backend.model;

import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import lombok.Data;

@Entity
@Data // Esta anotación crea automáticamente getters, setters, toString,
public class Producto {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String nombre;
    private String descripcion;
    private double precio;
    private int stock;
}
```



```
package com.perfulandia.backend.controller;

import com.perfulandia.backend.model.Producto;
import com.perfulandia.backend.service.ProductoService;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/api/productos")
public class ProductoController {

    private final ProductoService productoService;

    public ProductoController(ProductoService productoService) {
        this.productoService = productoService;
    }

    // Obtener todos los productos
    @GetMapping
    public List<Producto> getAllProductos() {
        return productoService.listarProductos();
    }

    // Obtener producto por ID
    @GetMapping("/{id}")
    public ResponseEntity<Producto> getProductoById(@PathVariable Long id) {
        return productoService.obtenerProductoPorId(id)
            .map(ResponseEntity::ok)
            .orElse(ResponseEntity.notFound().build());
    }
}
```

```
public class ProductoController {

    // Crear un nuevo producto
    @PostMapping
    public Producto createProducto(@RequestBody Producto producto) {
        return productoService.guardarProducto(producto);
    }

    // Actualizar un producto existente
    @PutMapping("/{id}")
    public ResponseEntity<Producto> updateProducto(@PathVariable Long id, @RequestBody Producto productoDetails) {
        return productoService.obtenerProductoPorId(id)
            .map(producto -> {
                producto.setNombre(productoDetails.getNombre());
                producto.setDescripcion(productoDetails.getDescripcion());
                producto.setPrecio(productoDetails.getPrecio());
                producto.setStock(productoDetails.getStock());
                Producto actualizado = productoService.guardarProducto(producto);
                return ResponseEntity.ok(actualizado);
            })
            .orElse(ResponseEntity.notFound().build());
    }

    // Eliminar un producto
    @DeleteMapping("/{id}")
    public ResponseEntity<Void> deleteProducto(@PathVariable Long id) {
        if (productoService.obtenerProductoPorId(id).isPresent()) {
            productoService.eliminarProducto(id);
            return ResponseEntity.noContent().build();
        }
        return ResponseEntity.notFound().build();
    }
}
```

```
BackendApplication.java  ProductoController.java  Producto.java  ProductoRepository.java
nd > src > main > java > com > perfulandia > backend > repository > ProductoRepository.java > Language
package com.perfulandia.backend.repository;

import com.perfulandia.backend.model.Producto;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface ProductoRepository extends JpaRepository<Producto, Long> {}
```

```
package com.perfulandia.backend.service;

import com.perfulandia.backend.model.Producto;
import com.perfulandia.backend.repository.ProductoRepository;
import org.springframework.stereotype.Service;

import java.util.List;
import java.util.Optional;

@Service
public class ProductoService {

    private final ProductoRepository productoRepository;

    public ProductoService(ProductoRepository productoRepository) {
        this.productoRepository = productoRepository;
    }

    // Obtener todos los productos
    public List<Producto> listarProductos() {
        return productoRepository.findAll();
    }

    // Buscar producto por id
    public Optional<Producto> obtenerProductoPorId(Long id) {
        return productoRepository.findById(id);
    }

    // Guardar o actualizar producto
    public Producto guardarProducto(Producto producto) {
        return productoRepository.save(producto);
    }
}
```

```
public class ProductoService {
    public Optional<Producto> obtenerProductoPorId(Long id) {
        // ...
    }

    // Guardar o actualizar producto
    public Producto guardarProducto(Producto producto) {
        return productoRepository.save(producto);
    }

    // Eliminar producto por id
    public void eliminarProducto(Long id) {
        productoRepository.deleteById(id);
    }
}
```

Configuración técnica y uso de control de versiones

2.2 Configuración del archivo application.properties

Este archivo define la configuración principal del proyecto:

- **Nombre del servicio:** backend.
- **Base de datos:** conexión a MySQL local (perfulandia) con usuario root.
- **JPA/Hibernate:** actualización automática de tablas (ddl-auto=update) y muestra de SQL en consola.
- **Puerto del servidor:** 8080.



```
spring.application.name=backend

# URL de conexión a la base de datos MySQL local
spring.datasource.url=jdbc:mysql://localhost:3306/perfulandia?useSSL=false&serverTimezone=UTC

# Usuario y contraseña de la base de datos (ajusta si tienes contraseña)
spring.datasource.username=root
spring.datasource.password=

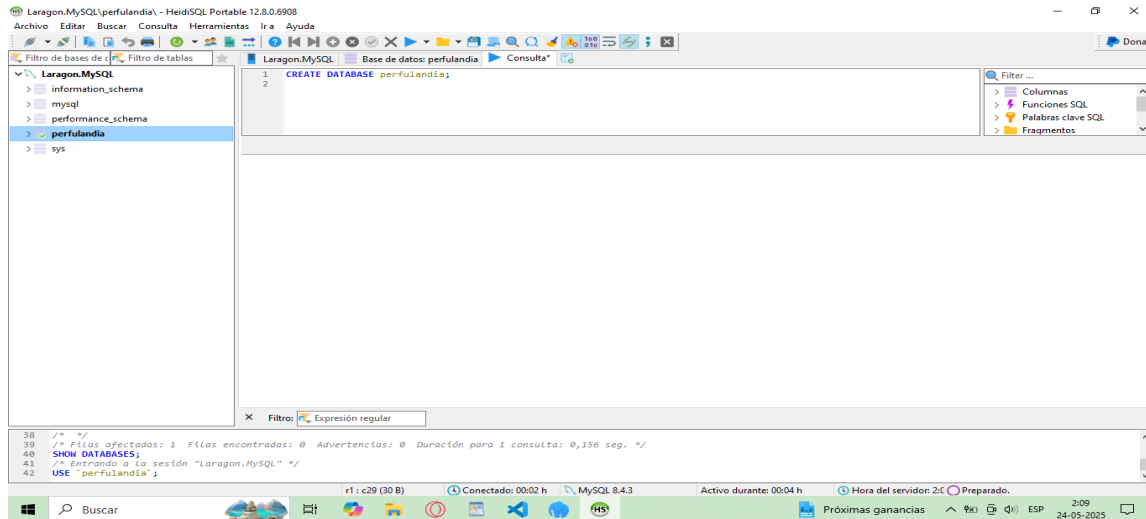
# Configuraciones de JPA/Hibernate
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect

# Puerto por defecto (opcional)
server.port=8080
```

2.3 Configuración de la Base de Datos MySQL

Para el proyecto, utilizamos **MySQL** como sistema gestor de base de datos relacional, debido a su estabilidad, popularidad y fácil integración con Spring Boot.

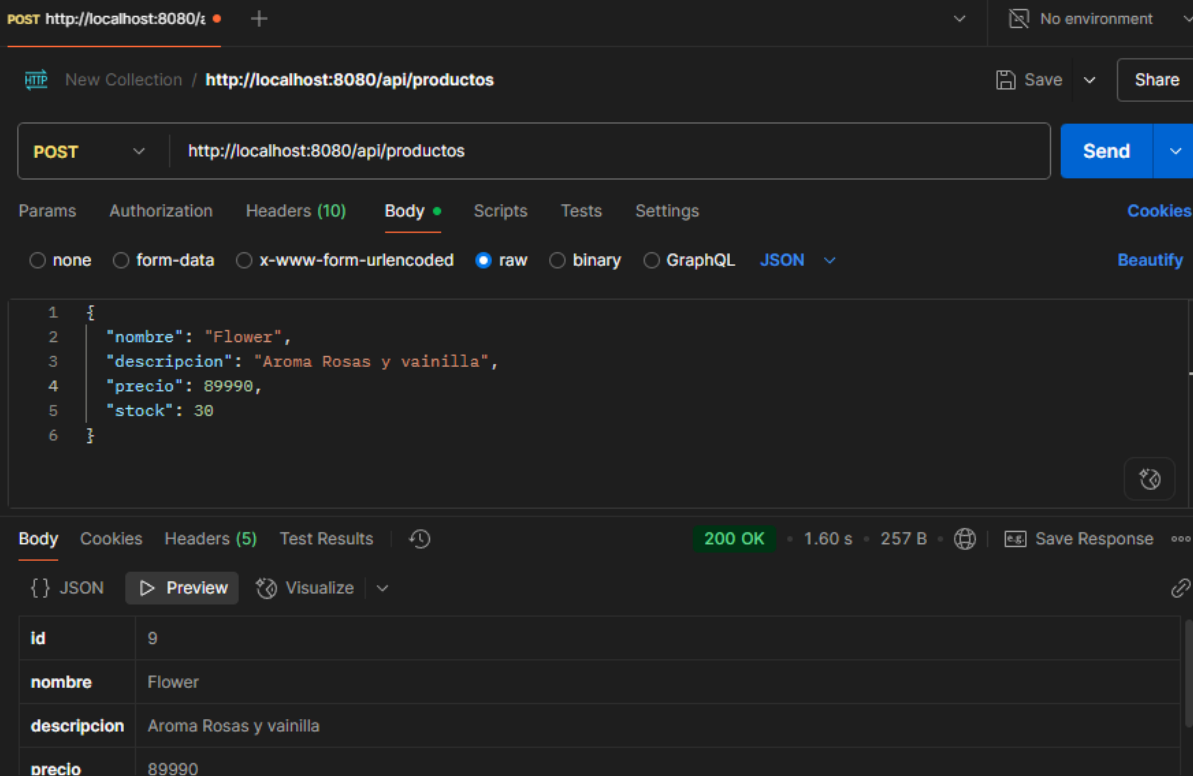
- **Base de datos:** Se creó una base llamada perfulandia donde se almacenan las tablas necesarias para la aplicación.
- **Conexión:** La aplicación se conecta a MySQL usando la URL de conexión configurada en application.properties:



2.4 Peticiones Postman – CRUD de Productos

Las pruebas se realizaron en Postman validando cada operación CRUD. Se usaron datos simulados en formato JSON, y se comprobó el correcto funcionamiento de los endpoints con respuestas HTTP apropiadas.

1. Crear Producto



POST `http://localhost:8080/api/productos`

Body (raw):

```

1 {
2   "nombre": "Flower",
3   "descripcion": "Aroma Rosas y vainilla",
4   "precio": 89990,
5   "stock": 30
6 }

```

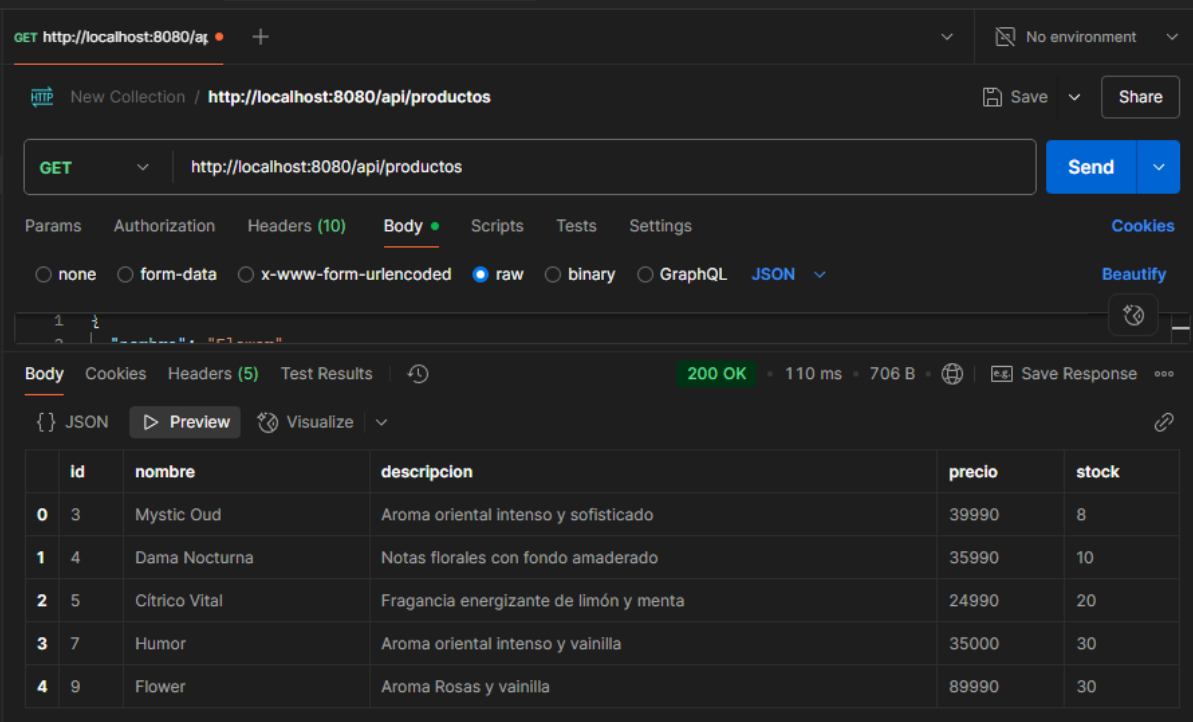
Response (JSON):

```

{
  "id": 9,
  "nombre": "Flower",
  "descripcion": "Aroma Rosas y vainilla",
  "precio": 89990
}

```

2. Obtener Todos los Productos



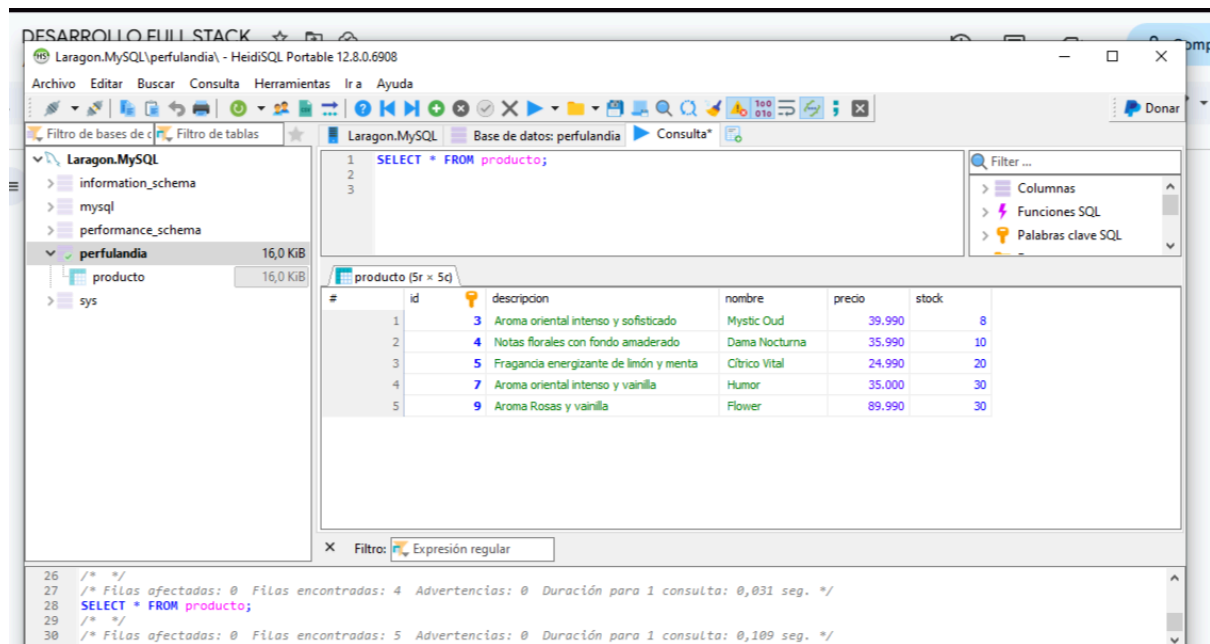
GET `http://localhost:8080/api/productos`

Response (JSON):

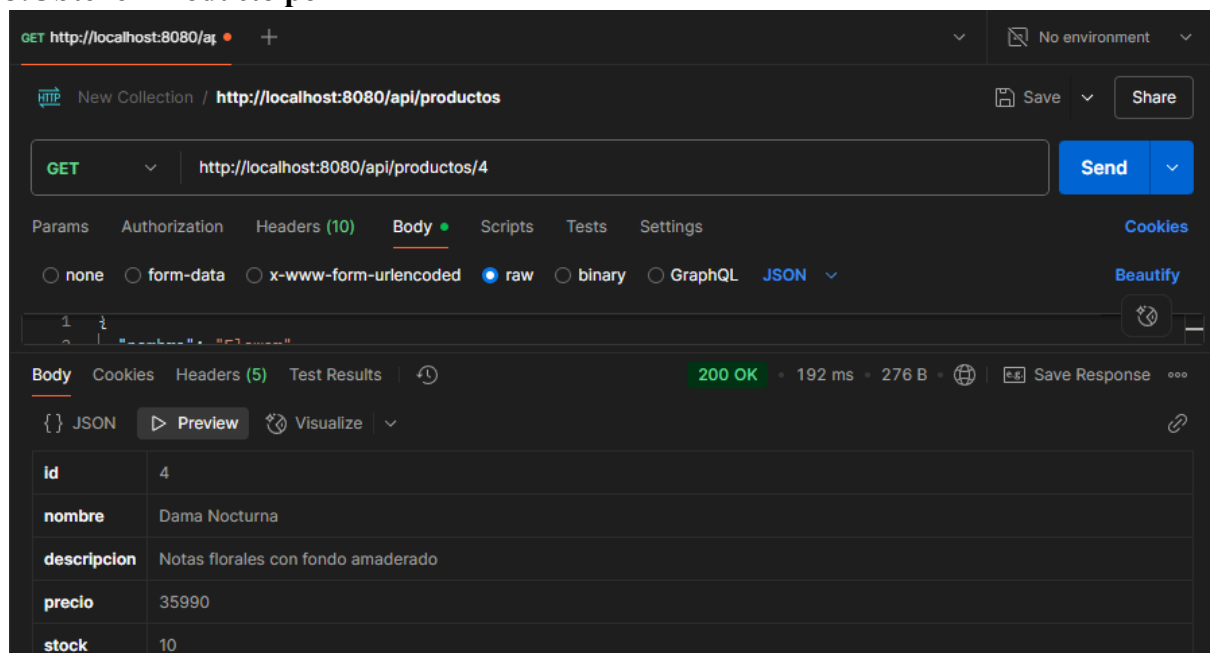
```

[
  {
    "id": 3,
    "nombre": "Mystic Oud",
    "descripcion": "Aroma oriental intenso y sofisticado",
    "precio": 39990,
    "stock": 8
  },
  {
    "id": 4,
    "nombre": "Dama Nocturna",
    "descripcion": "Notas florales con fondo amaderado",
    "precio": 35990,
    "stock": 10
  },
  {
    "id": 5,
    "nombre": "Cítrico Vital",
    "descripcion": "Fragancia energizante de limón y menta",
    "precio": 24990,
    "stock": 20
  },
  {
    "id": 7,
    "nombre": "Humor",
    "descripcion": "Aroma oriental intenso y vainilla",
    "precio": 35000,
    "stock": 30
  },
  {
    "id": 9,
    "nombre": "Flower",
    "descripcion": "Aroma Rosas y vainilla",
    "precio": 89990,
    "stock": 30
  }
]

```



3. Obtener Producto por ID



4. Actualizar Producto

PUT <http://localhost:8080/api> + No environment

New Collection / <http://localhost:8080/api/productos> Save Share

PUT <http://localhost:8080/api/productos/4> Send

Params Authorization Headers (10) **Body** Scripts Tests Settings Cookies

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL ☐ JSON Beautify

```

1 {
2   "id": 4,
3   "nombre": "Dama Nocturna",
4   "descripcion": "Notas florales con fondo amaderado",
5   "precio": 25000,
6   "stock": 10
7 }

```

Body Cookies Headers (5) Test Results 200 OK 107 ms 276 B Save Response

{} JSON Preview Visualize

nombre	Dama Nocturna
descripcion	Notas florales con fondo amaderado
precio	25000
stock	10

Laragon.MySQL\perfulandia - HeidiSQL Portable 12.8.0.6908 — □ ×

Archivo Editar Buscar Consulta Herramientas Ir a Ayuda

Filtro de bases de datos Filtro de tablas

Laragon.MySQL Base de datos: perfulandia Consulta

> information_schema
 > mysql
 > performance_schema
 > **perfulandia** 16,0 KiB
 producto 16,0 KiB
 > sys

```

1 SELECT * FROM producto;
2
3

```

Filter ...

> Columnas
 > Funciones SQL
 > Palabras clave SQL

#	id	descripcion	nombre	precio	stock
1	3	Aroma oriental intenso y sofisticado	Mystic Oud	39.990	8
2	4	Notas florales con fondo amaderado	Dama Nocturna	25.000	10
3	5	Fragancia energizante de limón y menta	Cítrico Vital	24.990	20
4	7	Aroma oriental intenso y vainilla	Humor	35.000	30
5	9	Aroma Rosas y vainilla	Flower	89.990	30

5. Eliminar Producto

DEL http://localhost:8080/api/... No environment

New Collection / http://localhost:8080/api/productos Save Share

DELETE ▼ http://localhost:8080/api/productos/9 Send ▼

Params Authorization Headers (10) **Body** • Scripts Tests Settings Cookies

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL **JSON** ▼ Beautify

```

1 {
2   "id": 4,
3   "nombre": "Dama Nocturna",
4   "descripcion": "Notas florales con fondo amaderado",
5   "precio": 25000,
6   "stock": 10
7 }

```

Body Cookies Headers (3) Test Results 🕒 **204 No Content** • 248 ms • 112 B • 🌐 Save Response ⋮

Raw Preview Visualize ▼ 🔗

Laragon.MySQL Base de datos: perfulandia Consulta* 📄

```

1 SELECT * FROM producto;
2
3

```

Filter ...

- > Columnas
- > Funciones SQL
- > Palabras clave SQL

producto (3r x 5c)

#	id	descripcion	nombre	precio	stock
1	3	Aroma oriental intenso y sofisticado	Mystic Oud	39.990	8
2	4	Notas florales con fondo amaderado	Dama Nocturna	25.000	10
3	5	Fragancia energizante de limón y menta	Cítrico Vital	24.990	20

```
Dar formato al texto ☒

[
  {
    "id": 3,
    "nombre": "Mystic Oud",
    "descripcion": "Aroma oriental intenso y sofisticado",
    "precio": 39990,
    "stock": 8
  },
  {
    "id": 4,
    "nombre": "Dama Nocturna",
    "descripcion": "Notas florales con fondo amaderado",
    "precio": 25000,
    "stock": 10
  },
  {
    "id": 5,
    "nombre": "Cítrico Vital",
    "descripcion": "Fragancia energizante de limón y menta",
    "precio": 24990,
    "stock": 20
  }
]
```

Desarrollo de componentes backend y colaboración

Durante el desarrollo, se utilizaron ramas por funcionalidad (**feature-producto**, **feature-pedido**) y se realizaron merges hacia la rama principal (**main**) una vez probados los cambios. Los conflictos fueron gestionados colaborativamente mediante pull requests y revisión de código previa a la integración.

También se aplicaron mensajes de commit descriptivos para cada funcionalidad agregada, y se hizo seguimiento de versiones para revertir cambios si era necesario. Esto garantizó orden, trazabilidad y trabajo sincrónico entre los integrantes.

```

MINGW64:/c/Users/diego/OneDrive/Desktop/Perfulandia
diego@diegopc MINGW64 ~/OneDrive/Desktop/Perfulandia (main)
$ git init
Reinitialized existing Git repository in C:/Users/diego/OneDrive/Desktop/Perfulandia/.git/

diego@diegopc MINGW64 ~/OneDrive/Desktop/Perfulandia (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   .idea/vcs.xml

diego@diegopc MINGW64 ~/OneDrive/Desktop/Perfulandia (main)
$

```

git init: Crea un nuevo repositorio Git en la carpeta donde estás, para empezar a controlar versiones.

git status: Muestra el estado actual de los archivos (qué cambios hay, qué archivos están listos para hacer commit, etc.).

```

diego@diegopc MINGW64 ~/OneDrive/Desktop/Perfulandia (main)
$ git add .

diego@diegopc MINGW64 ~/OneDrive/Desktop/Perfulandia (main)

```

Agregar todos los archivos modificados, nuevos o eliminados en la carpeta actual (y sus subcarpetas) al área de preparación (staging area), para que estén listos para el próximo commit.

```

diego@diegopc MINGW64 ~/OneDrive/Desktop/Perfulandia (main)
$ git commit -m "Se agrega microservicio pedido-service"
[main ded486d] Se agrega microservicio pedido-service
1 file changed, 6 insertions(+)
 create mode 100644 .idea/vcs.xml

```

git commit -m "mensaje": Guarda (confirma) los cambios que has agregado con git add, y les pones un mensaje descriptivo entre comillas para saber qué cambios hiciste.

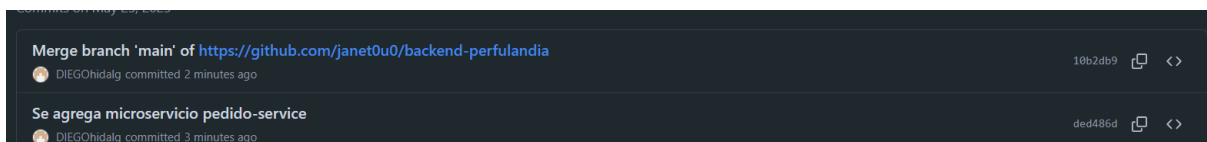
```
diego@diegopc MINGW64 ~/OneDrive/Desktop/Perfulandia (main)
$ git pull origin main --allow-unrelated-histories
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 2 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
Unpacking objects: 100% (2/2), 931 bytes | 77.00 KiB/s, done.
From https://github.com/janet0u0/backend-perfulandia
* branch      main      -> FETCH_HEAD
   301836f..8e28637  main      -> origin/main
Deletion of directory 'pedidos-service/.mvn/wrapper' failed. Should I try again?
(y/n) |
```

- `git pull origin main` — Trae y fusiona la rama main del repositorio remoto origin.
- `--allow-unrelated-histories` — Permite fusionar dos repositorios o ramas que no comparten historial común..






```
diego@diegopc MINGW64 ~/OneDrive/Desktop/Perfulandia (main)
$ git push origin main
Enumerating objects: 10, done.
Counting objects: 100% (9/9), done.
Delta compression using up to 16 threads
Compressing objects: 100% (6/6), done.
Writing objects: 100% (6/6), 856 bytes | 285.00 KiB/s, done.
Total 6 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/janet0u0/backend-perfulandia.git
   8e28637..10b2db9  main -> main
```

`git push origin main` significa:

- Subir (enviar) los commits que tienes localmente en tu rama main hacia la rama main del repositorio remoto llamado origin.



En esta parte **hice un merge** para unir los cambios que hice en una rama a la rama principal (main). Esto se hace para subir y combinar esos cambios en la rama principal del proyecto y que estén disponibles para todos.

 DIEGOHidalg Nuevo commit para agregar microservicio	9fd5822 · 1 minute ago	 10 Commits
 .idea	Se agrega microservicio pedido-service	8 minutes ago
 pedidos-service	Nuevo commit para agregar microservicio	1 minute ago
 productos-service	Proyecto de microservicios	31 minutes ago

Como se puede apreciar, los cambios en GitHub se han subido correctamente. De esta manera, utilizamos Git y GitHub para trabajar de forma colaborativa, gestionando el proyecto mediante ramas, commits, pull requests y revisiones.

Conclusión

Tras el análisis, se puede deducir que el desarrollo backend del sistema para Perfulandia SPA logró cumplir con los objetivos técnicos propuestos. Tal y como se pudo comprobar, se aplicaron correctamente los conceptos de arquitectura en capas con Spring Boot, la gestión de dependencias con Maven, y el control de versiones con Git.

Gracias a todo lo anterior, es posible interpretar que el trabajo colaborativo y el uso de buenas prácticas permitieron construir un backend funcional, escalable y validado mediante pruebas exitosas en Postman.

Como reflexión final, destacamos que el uso de buenas prácticas de diseño como la separación de capas, el uso de anotaciones específicas de Spring, la modularización del código y la correcta gestión de versiones en Git contribuyeron a lograr un backend sólido, mantenible y escalable. Este enfoque no solo permitió cumplir los requerimientos del proyecto, sino que también sentó las bases para una futura integración con un frontend full stack.

Enlace al repositorio GitHub

Enlace al repositorio: <https://github.com/janet0u0/backend-perfulandia.git>

Tabla de integrantes:

Nombre completo	Usuario GitHub	Funciones/Responsabilidades principales
Janet Huaylla	https://github.com/janet0u0	Configuré el proyecto con Spring Boot y Maven. Creé el CRUD de Producto (modelo, repositorio, servicio y controlador). Realicé commits: uno con la estructura y otro con el CRUD. Probé todo en Postman y con base de datos MySQL.
Diego Hidalgo	https://github.com/DIEGOhidalgo	Configuré el microservicio Pedido con Spring Boot y Maven. Implementé el CRUD de cupones (modelo, repositorio, servicio y controlador), lo probé en Postman y lo conecté a una base de datos MySQL. Subí los cambios al repositorio backend-perfulandia usando Git y gestioné los commits y ramas correctamente.

Bibliografía

Cómo realizar una introducción de un informe: ejemplo. (2023, junio 30). *Educación Activa*.
<https://educacionactiva.org/como-realizar-una-introduccion-de-un-informe-ejemplo>

Farías, G. (s/f). *Conclusión - Qué es y cómo hacer una conclusión*. Recuperado el 27 de junio de 2024, de <https://concepto.de/conclusion/>

Porto, J. P., & Gardey, A. (2012, febrero 20). *VIP - Qué es, definición y concepto*. Definición.de; Definicion.de. <https://definicion.de/vip/>

Puedes usar: <https://www.bibguru.com/es/c/generador-citas-apa/>

