Caso 1: Perfulandia

SPA

Integrantes: Gabriel Avendaño – Jeremy Cárcamo – Camila Pino

Asignatura: Desarrollo FullStack I

Profesor: Eduardo Baeza

Sección: 010D

Carrera: Analista Programador Computacional

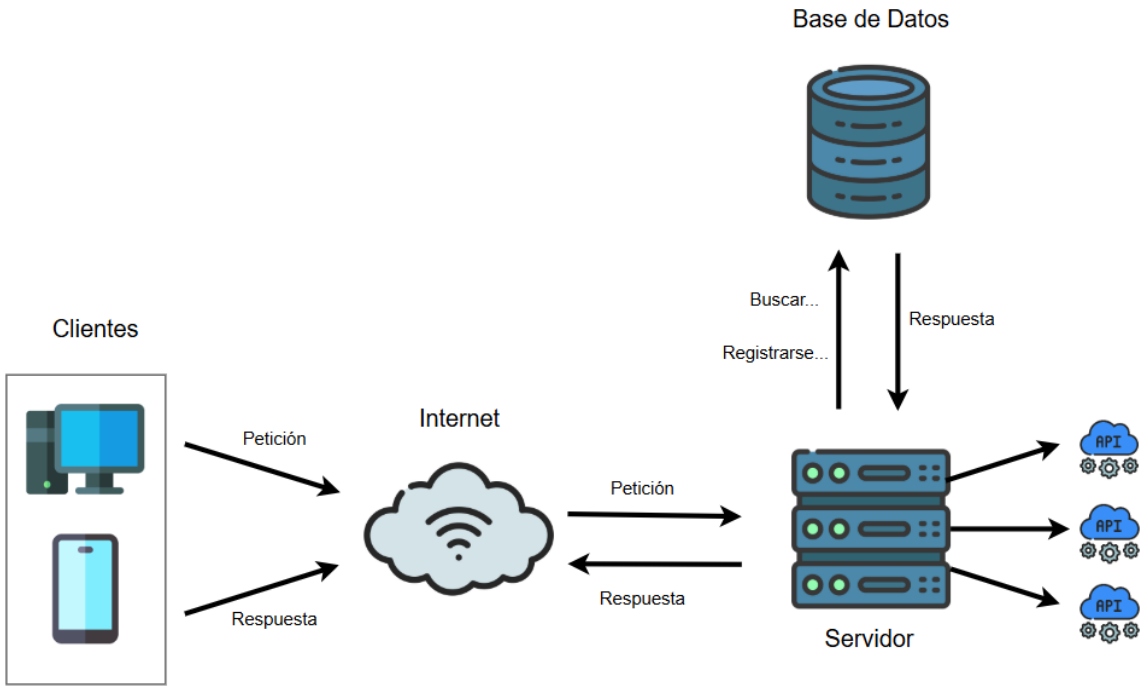
Fecha de entrega: 26 de Mayo de 2025

Contenido

Introducción

El presente informe documenta el desarrollo y la implementación de una arquitectura basada en microservicios para el sistema de gestión de Perfulandia SPA, una empresa en proceso de expansión que requiere una solución tecnológica escalable, eficiente y moderna. Con el fin de resolver las limitaciones del sistema monolítico anterior, se adoptó una estructura modular apoyada en Spring Boot y MySQL, que permite gestionar de manera independiente funcionalidades clave como la administración de productos, usuarios y boletas. Este enfoque facilita el mantenimiento, mejora la escalabilidad y permite que cada componente del sistema pueda ser desarrollado, probado y desplegado de forma separada. Además, se detallan las pruebas realizadas en Postman, la conexión con base de datos, y el uso de herramientas de control de versiones como Git y GitHub para respaldar el flujo de trabajo del desarrollo.

1. Diagrama de arquitectura de microservicios



2. Estructura del proyecto

El proyecto fue estructurado utilizando Spring Boot como framework principal, utilizando dependencias como:

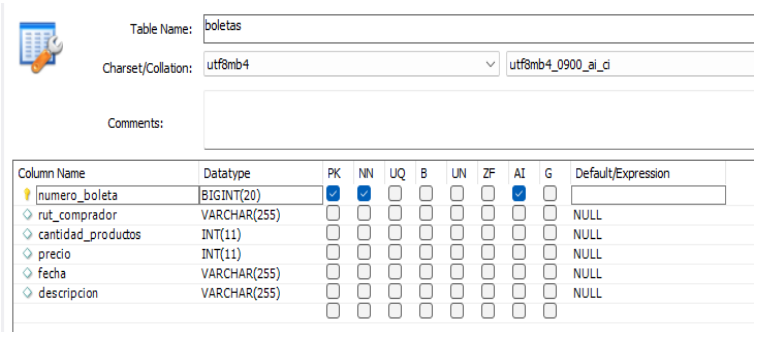
- Spring Boot DevTools: Facilita el desarrollo con recarga automática de la aplicación al guardar cambios.
- Spring Web: Permite crear servicios RESTful y manejar peticiones HTTP con Spring MVC.
- MySQL Driver: Driver JDBC necesario para conectar la aplicación con una base de datos.
- Spring Data JPA: Habilita el uso de JPA e Hibernate para la persistencia de datos en bases relacionales.

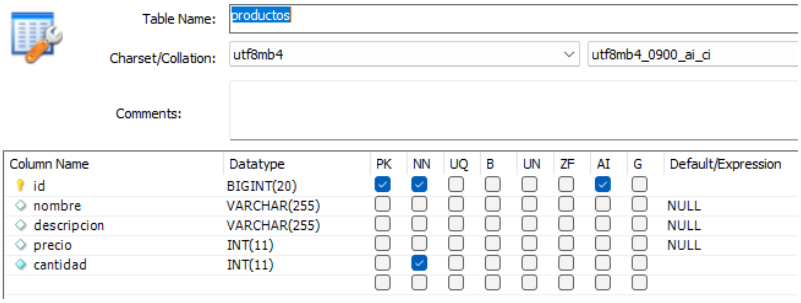
En la arquitectura del proyecto también se implementaron componentes tales como:

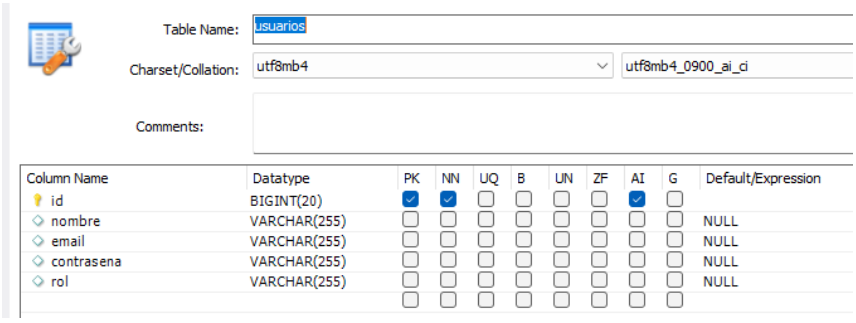
- Controllers: Manejan las peticiones HTTP (como GET, POST, PUT, DELETE) y están anotados con @RestController. Reciben datos del cliente, llaman a los servicios correspondientes y devuelven respuestas. Un ejemplo de esto es ProductController.
- Entities: Representan las tablas de la base de datos y se implementan como clases de Java anotadas con @Entities, @Id, @Table, etc.
- Repository: Son interfaces que extienden CrudRepository o JpaRepository, lo que permite acceder a la base de datos sin necesidad de escribir SQL manual.
- Services: Realiza las operaciones principales del sistema, como guardar, buscar o eliminar datos. Están organizados en interfaces como ProductService y clases que las implementan, como ProductServiceImpl. Usan la anotación @Service para indicar que son parte de la lógica del programa y @Transactional para asegurarse de que las acciones sobre la base de datos se realicen correctamente.

3. Base de Datos

El proyecto utilizará 1 Base de datos, usando MySQL como su motor, esta contiene tablas como productos, usuarios y boletas, que almacenan la información principal del sistema. Cada tabla está representada por una clase entidad en Java, mapeada con anotaciones como @Entity y @Id. A continuación, se presenta una imagen con la estructura de las tablas utilizadas.







4. Implementación de Servicios

Con el objetivo de distribuir de manera eficiente las responsabilidades del sistema y mejorar la escalabilidad de este, se crearon 3 servicios independientes, en el que cada uno de ellos se encarga de una funcionalidad específica del sistema. Estos servicios fueron implementados usando Spring Boot, persistencia con con JPA y conexión con una base de datos MySQL.

4.1 Servicio de Productos

Este servicio tiene como objetivo gestionar todo lo relacionado con los productos disponibles en el catálogo de Perfulandia SPA. Permite registrar productos nuevos, obtener productos generales o por ID, modificar la información de productos existentes y eliminarlos. Cada producto contiene atributos de nombre, descripción, precio y cantidad en stock.

Este servicio está compuesto por entities, el cual es Producto, que representa cada producto en la base de datos, también cuenta con un repository, el que permite acceder a productos almacenados, ya sea para acceder a una lista, buscar un producto específico por ID, guardarlo o eliminarlo. Su interface define las acciones que este puede realizar, y su implementación las ejecuta, y por último el Controller, que recibe las solicitudes de Postman que permite enviar, modificar, obtener o eliminar productos.

4.2 Servicio de Boletas

El servicio de Boletas se encarga de gestionar la emisión y consulta de comprobantes de venta. Cada boleta contiene un número identificador, RUT del cliente, detalle de los productos comprados, el total pagado, y la fecha de transacción. Siendo este importante para el control de ventas realizadas en el sistema.

Compuesto por Entities, que es Boleta, que representa cada comprobante generado con su respectiva información. Un Repository, que permite guardar nuevas boletas, buscar boletas anteriores o eliminarlas, y es gestionado automáticamente por Spring para conectarse con la base de datos. El Interface define las operaciones a realizar, como registrar boletas o buscarla por su número, y su clase implementada las ejecuta. El controlador correspondiente recibe las solicitudes desde herramientas como Postman y permite consultar o registrar boletas fácilmente.

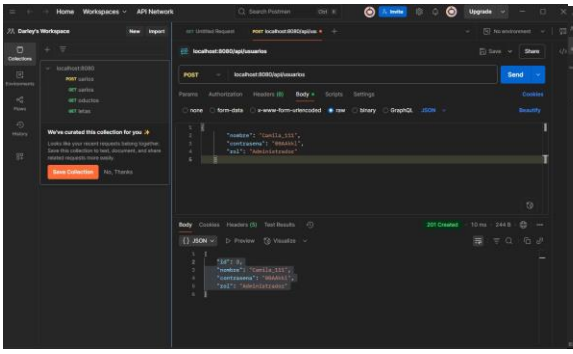
4.3 Servicio de Usuarios

Este servicio está encargado de administrar todas las cuentas de usuario que accedan al sistema, incluye registrar usuarios nuevos, consultar su información, actualizar sus datos y eliminar registros en casos necesarios.

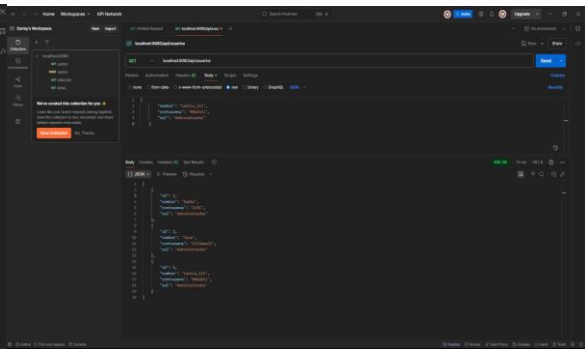
Construido por Entities, que es Usuario, que contiene los datos básicos de cada persona registrada. Su Repository permite guardar nuevos usuarios, buscar usuarios existentes, modificarlos o eliminarlos. El Interface define las funciones disponibles para trabajar con los datos de cada cuenta, mientras que su implementación las aplica correctamente. Finalmente, su controlador gestiona las peticiones de registrar un nuevo usuario, obtener la información de uno existente o eliminarlo, haciendo que el sistema responda adecuadamente.

Cada servicio fue probado en postman y siendo verificado por cada petición, logrando interactuar con la base de datos:

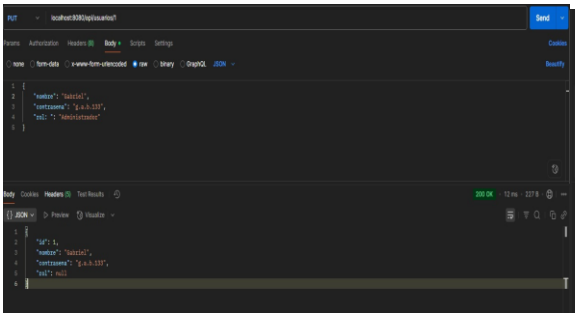
POST



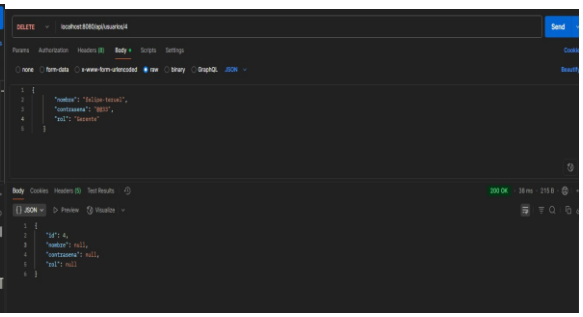
GET



PUT

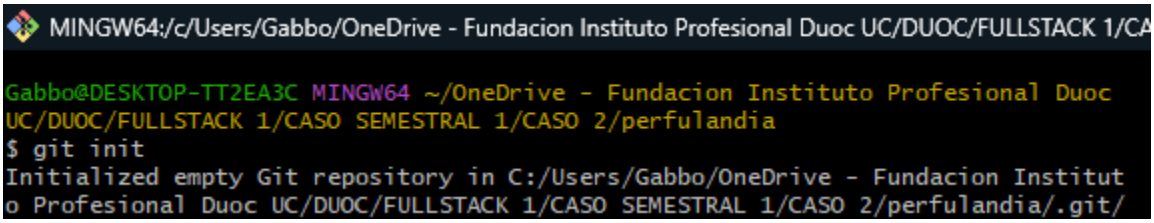


DELETE

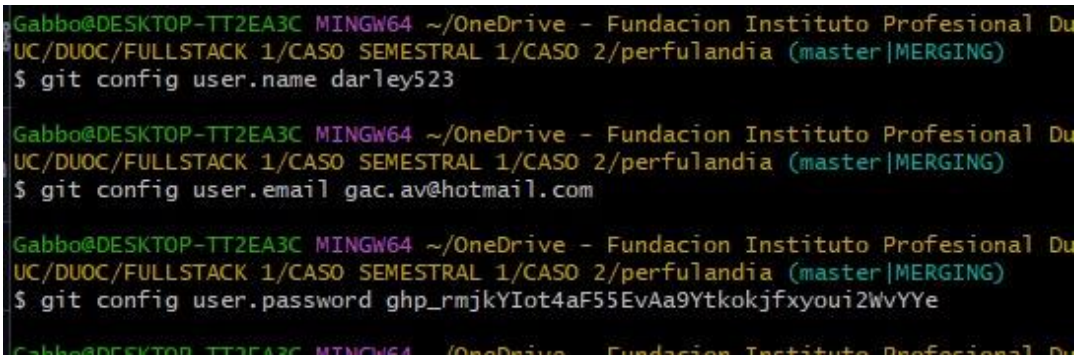


5. GitHub

Comando para inicializar un repositorio de git en local



Comando para identificarse al momento de hacer los commit, agregando una contraseña para el repositorio online



Comando para añadir todos los archivos creados o modificados al repositorio

Comando para guardar una “versión/snapshot” del repositorio (local)

```
Gabbo@DESKTOP-TT2EA3C MINGW64 ~/OneDrive - Fundacion Instituto Profesional Duoc UC/DUOC/FULLSTACK 1/CASO SEMESTRAL 1/CASO 2/perfulandia (master)
$ git commit -m "Correo añadido a usuario"
[master 7e3fa4b] Correo añadido a usuario
 2 files changed, 16 insertions(+), 1 deletion(-)
```

Comando para comprobar que cosas no han sido guardadas en el repositorio

```
$ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   src/main/java/com/perfulandia/perfulandia/controllers/UsuarioController.java
        modified:   src/main/java/com/perfulandia/perfulandia/entities/Usuario.java

no changes added to commit (use "git add" and/or "git commit -a")
```

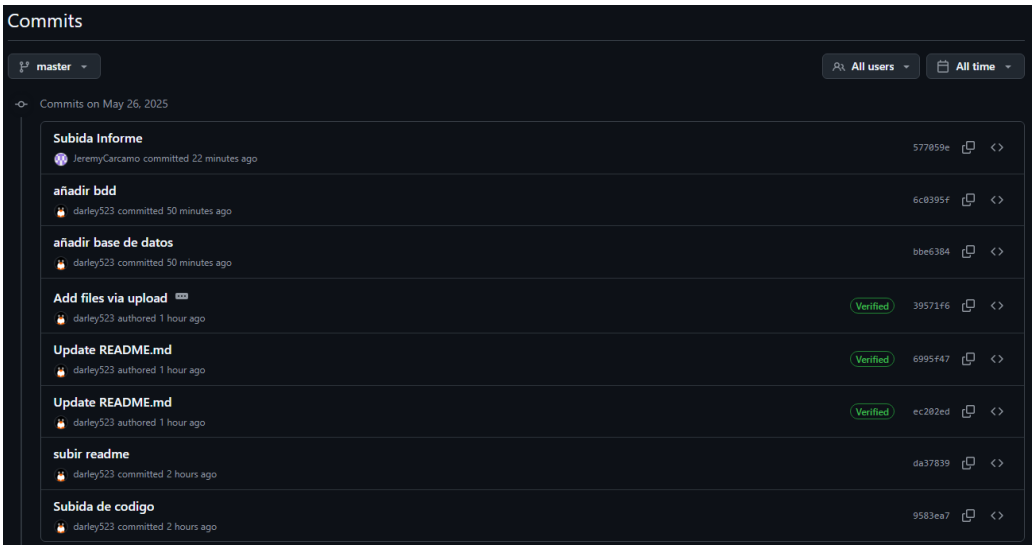
Comando para vincular el repositorio local con uno online

```
$ git remote add origin https://github.com/darley523/perfulandia_2
error: remote origin already exists.
```

Comando para subir los archivos al repositorio online (sube todos los commit)

```
$ git push -u origin master
Enumerating objects: 36, done.
Counting objects: 100% (26/26), done.
Delta compression using up to 12 threads
Compressing objects: 100% (11/11), done.
Writing objects: 100% (14/14), 1.13 KiB | 289.00 KiB/s, done.
Total 14 (delta 7), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (7/7), completed with 6 local objects.
To https://github.com/darley523/perfulandia_2.git
   2e90d2a..f9bb160  master -> master
branch 'master' set up to track 'origin/master'.
```

Historial de commits colaborativos durante el desarrollo del proyecto



Conclusión

La migración hacia una arquitectura de microservicios ha representado una mejora significativa para el sistema de Perfulandia SPA, otorgando una base sólida para escalar y mantener sus operaciones de forma eficiente. La implementación de servicios independientes, junto con la utilización de tecnologías como Spring Boot, JPA y MySQL, ha permitido distribuir responsabilidades entre los distintos módulos del sistema, facilitando su desarrollo y prueba. Gracias al uso de Postman, fue posible verificar el correcto funcionamiento de cada servicio a través de las peticiones HTTP (GET, POST, PUT y DELETE), confirmando su integración con la base de datos.