Transformación Digital de "Perfulandia SPA"

Integrantes: Brayan Ahumada Rojas

Cristian Ormazabal

Sección: 011D

Profesor: Eduardo Baeza

Índice

[**1. Introducción 3**](#_2et92p0)

[**1.1. Objetivo del Caso 3**](#_a97nsu2x0yo)

[**2. Diagramas de arquitectura de microservicios 3**](#_d20w3h2pj0d4)

[**2.1. Diagrama Usuario 3**](#_2gsms42lcxdi)

[**2.2. Diagrama Inventario 3**](#_yl3e2ah6mnp2)

[**2.3. Diagrama Envío 3**](#_nwyo12oyfd6q)

[**3. Estructura del proyecto 3**](#_udnfkmbw8vki)

[**3.1. Descripción de carpetas y paquetes principales. 3**](#_hh0xru32qndl)

[**3.2. Imagen del árbol de directorios. 4**](#_olcmaalqdfbw)

[**3.3. Dependencias más importantes del pom.xml 5**](#_hqn5uis4m0g1)

[**3.4. Descripción de componentes implementados (controladores, servicios, repositorios, modelos). 5**](#_nidyj78lznh)

[**4. Base de datos 5**](#_cy96g0opq9xp)

[**4.1. Motor de base de datos utilizado (ej. H2, MySQL). 5**](#_dbasjdujcbgf)

[**4.2. Imagen del esquema de base de datos. 6**](#_qnpm4rq2hoqw)

[**4.3. Breve descripción de tablas y relaciones. 6**](#_lwmsxaevh17m)

[**5. Implementación de los servicios (API REST) 7**](#_buhacrky18cn)

[**5.1. Nombre y propósito de cada servicio. 7**](#_4ov65muphidh)

[**5.2. Imágenes de ejecución de peticiones en Postman. 8**](#_kobn8hpckrta)

[**5.3. Fragmentos del código relevante si es necesario. 16**](#_i3x19lhk1cvu)

[**5.4. Confirmar interacción con la base de datos. 17**](#_521haojulm1x)

[**6. Vistas para petición GET 17**](#_va1e0yf69hdf)

[**6.1. Captura de pantalla de alguna interfaz simple donde se muestran los datos consultados desde los servicios. 17**](#_vey4pya9ehva)

[**6.2. Breve explicación de cómo se conecta esa vista con el backend. 17**](#_9jo7onqula0s)

[**7. Git – GitHub 18**](#_p0g8c9dwicyo)

[**7.1. Explicación de comandos Git usados (git init, git add, git commit, git push, git clone, etc.). 18**](#_l71zk0e0m1t4)

[**7.2. Capturas del terminal con estos comandos en uso. 18**](#_einrqkjzjgmh)

[**7.3. Imagen del repositorio en GitHub. 20**](#_myf523oh19xz)

[**8. Conclusión 20**](#_3ln4ec9sx7fi)

[**8.1. Reflexión del equipo sobre el trabajo realizado. 20**](#_339bghlh46ig)

[**8.2. Dificultades enfrentadas y cómo se solucionaron. 20**](#_rfud6pr2s1im)

[**8.3. Importancia del uso de microservicios en el desarrollo moderno. 20**](#_c2rzvdoa69zd)

# 

## **Introducción**

Perfulandia SPA es una empresa chilena emergente, que se ha destacado por ofrecer productos de alta calidad a precios competitivos. Inicialmente, la empresa comenzó con una sucursal en el Barrio Meiggs en Santiago, pero su éxito en ventas tanto al por menor como al por mayor ha llevado a la apertura de nuevas sucursales en Concepción y Viña del Mar. La empresa ahora planea continuar su expansión debido a su crecimiento exponencial y el aumento de nuevos clientes a nivel nacional. Sin embargo, este rápido crecimiento ha revelado las limitaciones de su actual sistema de software monolítico. El sistema ha comenzado a fallar, presentando problemas de rendimiento y disponibilidad que ponen en riesgo las operaciones diarias y la satisfacción del cliente.

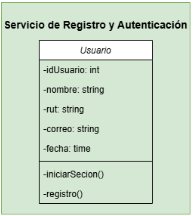
## **Objetivo del Caso**

El objetivo de este caso de estudio es analizar y desarrollar una solución al problema que presenta la empresa, pasando por el análisis, diseño, e implementación de una solución tecnológica que permita a Perfulandia SPA superar las limitaciones de su sistema actual y soportar su continuo crecimiento por medio de la arquitectura de microservicios utilizando un motor de BD MySql.

El proyecto se desarrollará en tres partes, a lo largo de las tres evaluaciones parciales del semestre, culminando en una presentación final donde deberá defenderse la solución propuesta.

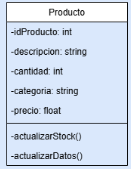
## **Diagramas de arquitectura de microservicios**

## **Diagrama Usuario**

* + 1. **Diagrama de Clases y Casos de Uso**

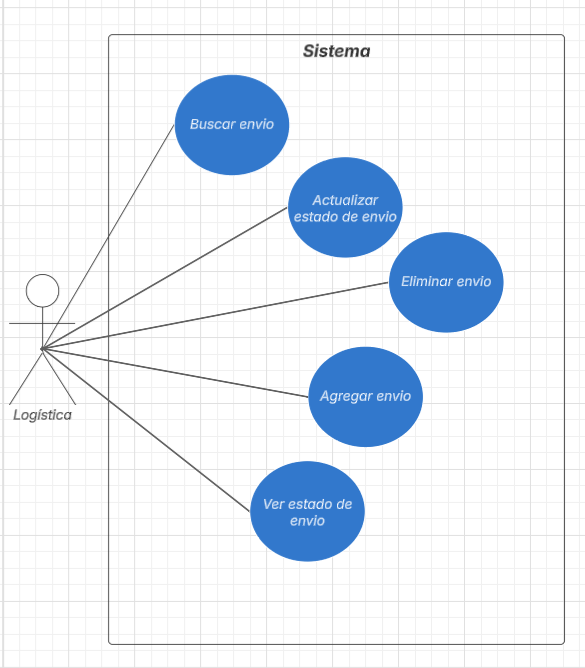
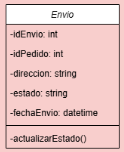
## **Diagrama Inventario**

* + 1. Diagrama de Clases y Casos de Uso

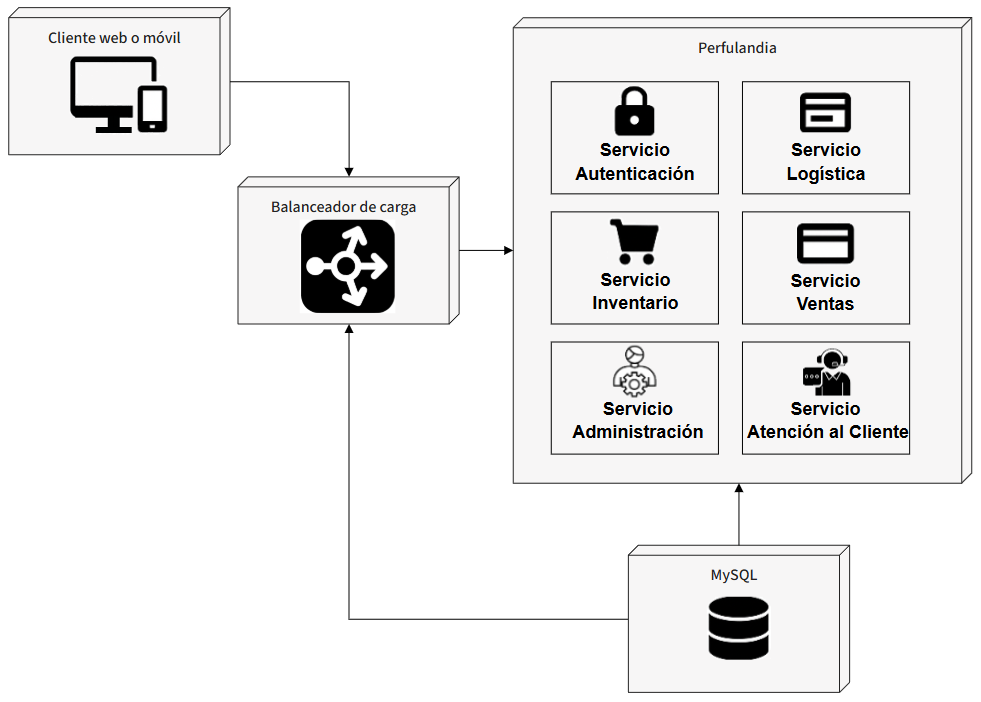


## **Diagrama Envío**

* + 1. Diagrama de Clases y Casos de Uso



* 1. **Diagrama de Despliegue**

****

## **Estructura del proyecto**

## **Descripción de carpetas y paquetes principales.**

El proyecto de perfulandia se está desarrollando en el framework Spring Boot, implementando el sistema de Maven. La estructura de este proyecto mantiene un patrón usualmente común de una aplicación basada en microservicios, donde se dividen responsabilidades en carpetas como por ejemplo: controller, entities, services y repositories.

## **Imagen del árbol de directorios.**



## **Dependencias más importantes del pom.xml**

Dentro de las dependencias que se utilizaron al momento de crear el proyecto y que se encuentran en el pom.xml, se encuentran las siguientes:

* **spring-boot-starter-web**: Permite crear servicios REST y controladores web.
* **spring-boot-starter-data-jpa**: Habilita el acceso a base de datos mediante JPA (ORM).
* **spring-boot-starter-thymeleaf**: Integración con el motor de plantillas Thymeleaf para vistas HTML (opcional).
* **mysql-connector-j**: Driver JDBC para conectar con base de datos MySQL.
* **spring-boot-devtools**: Mejora la experiencia de desarrollo con recarga automática.
* **spring-boot-starter-test**: Dependencias necesarias para pruebas unitarias y de integración.

## **Descripción de componentes implementados (controladores, servicios, repositorios, modelos).**

**@RestController**: Convierte la clase correspondiente en un controlador “REST”, quiere decir que manejará solicitudes HTTP.

**@RequestMapping**: “Mapea” urls y solicitudes HTTP a controladores, esencial para lograr un funcionamiento correcto al solicitar “GET, POST, PUT, DELETE”.

**@Autowired**: Le pide a Spring que use el componente que encuentre pertinente.

**@GetMapping**: Es una versión simplificada para solicitudes GET, devuelve todos (o solo uno indicado por id) los resultados de la base de datos a través de un URL mapeado anteriormente.

**@PostMapping**: Es una versión simplificada para solicitudes POST, crea un nuevo dato en la base de datos.

**@PutMapping**: Es una versión simplificada para solicitudes PUT, modifica un dato existente en la base de datos.

**@DeleteMapping**: Es una versión simplificada para solicitudes DELETE, elimina un dato existente en la base de datos dejando sus valores en “null”.

**@PathVariable**: Obtiene valores directamente de la URL, usado para obtener un dato a través de un id o código.

**@RequestBody**: Vincula la solicitud HTTP a un objeto Java, si se envían datos JSON, Spring los convierte en un objeto Java que recibe el método.

**@Entity**: Marca una clase como una “entidad” que representa una tabla en la base de datos.

**@Table**: Configura los detalles de la tabla que representa una “entidad”.

**@Id**: Marca el campo que será la “PK” (Primary Key) de la tabla en la base de datos.

**@GeneratedValue**: El campo marcado con @Id será generado automáticamente por la base de datos, se puede usar con diferentes “estrategías”, en este caso se usa “GenerationType.IDENTITY la cual usa la columna AI (autoincremental) de la base de datos.

**@Service**: Marca la clase como servicio, facilitando la inyección de dependencias y organiza el código separándolos de controladores o repositorios.

**@Override:** Indica que un método está sobrescribiendo un método de una clase o interfaz.

**@Transactional**: Marca que el método o clase debe ejecutarse en una “transacción”, quiere decir que las operaciones se ejecutan todas o ninguna, en este caso se usa “readOnly = true”, lo cual indica que la operación será solo de lectura.

## **Base de datos**

## **Motor de base de datos utilizado.**

La base de datos utilizada en este proyecto es MySQL, utilizando XAMPP como herramienta intermediaria para montar el servidor de manera local y así poder probar las aplicaciones web antes de publicarlas. Además, se utilizó MySQL WorkBench que es una herramienta visual de diseño de bases de datos, gestión y mantenimiento para el sistema de base de datos MySQL.

## **Imagen del esquema de base de datos.**

## 

## **Breve descripción de tablas y relaciones.**

Dentro de la base de datos “db\_perfulandia” se encuentran tres tablas principales que representan a las entidades del sistema: Usuarios, inventario, Envíos.

Table Usuario.

| **Columna** | **Descripción** |
| --- | --- |
| id | Identificador único del usuario |
| nombre | Nombre completo del usuario |
| correo | Correo electrónico del usuario |
| direccion\_envio | Dirección para los envíos |

Table Inventario.

| **Columna** | **Descripción** |
| --- | --- |
| id | Identificador único del producto |
| nombre | Nombre de producto |
| descripcion | Detalles del producto |
| precio | Precio del producto |

Table Envío.

| **Columna** | **Descripción** |
| --- | --- |
| id | Identificador único del envío |
| dirección | Dirección de destino del envío |
| fecha\_envio | Fecha programada del envío |
| nombre | Nombre del destinatario |

Inicialmente en el modelo de base de datos no se encuentra una relación entre tablas, sin embargo estas relaciones pueden ser implementadas en las versiones futuras de este proyecto como por ejemplo: un usuario puede estar asociado a múltiples envíos (relación 1:n.)

## **Implementación de los servicios**

## **Nombre y propósito de cada servicio.**

El propósito de los microservicios es aligerar la carga de un sistema grande y complejo en pequeños servicios independientes y fáciles de desplegar. Dentro de los microservicios implementados en esta primera etapa del proyecto se encuentran:

Usuarios: Su propósito es manipular la información de los usuarios de manera correcta, ya sea crear, eliminar o modificar datos ingresados. Inventario: Su propósito es manipular la información de los productos de manera correcta, ya sea crear, ver, eliminar o modificar los distintos productos actuales en el inventario.

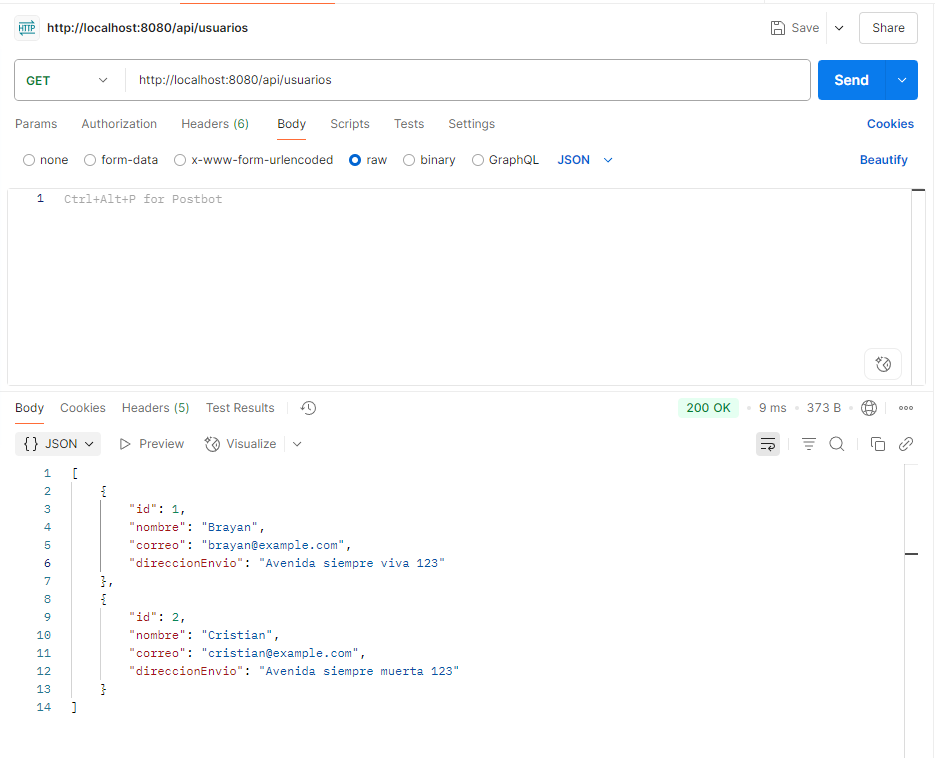
Envíos: Su propósito es manipular la información de los envíos programados de manera correcta, ya sea crear, ver, modificar o eliminar una solicitud.

Algunos de los componentes que estos micro servicios tienes estan **@Entity**: Modelo de datos, **@Repository**: Interfaz que extiende CrudRepository para manejar la base de datos, **@RestControlle**r Controlador con endpoints para CRUD, application.properties o .yml Configuración de puerto, conexión a DB, etc. Además, tiene los Método **GET** (Listar todos o por id si se especifica), **POST**(Crear), **PUT** (Actualizar), **DELETE** (Eliminar).

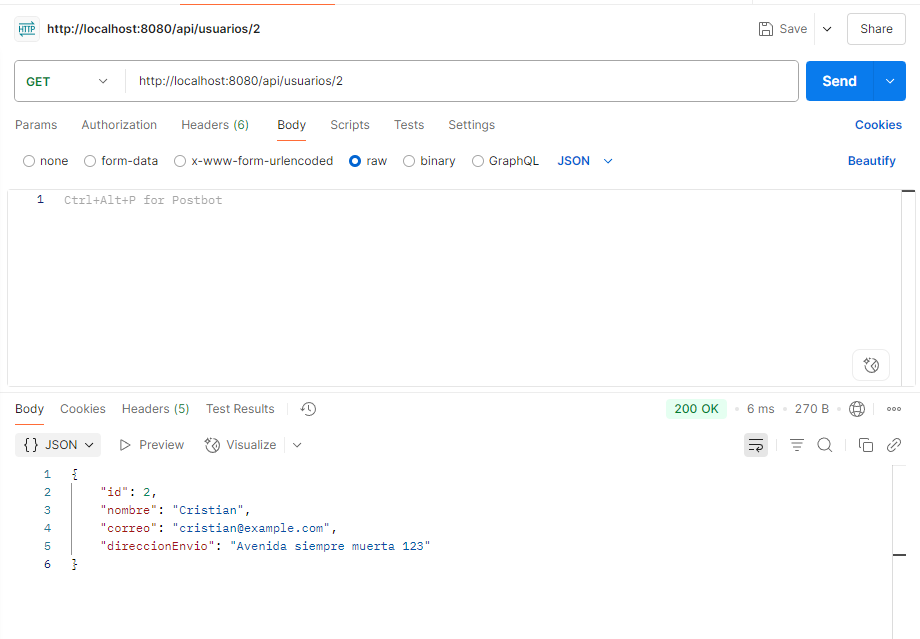
## **Imágenes de ejecución de peticiones en Postman.**

**Usuarios**

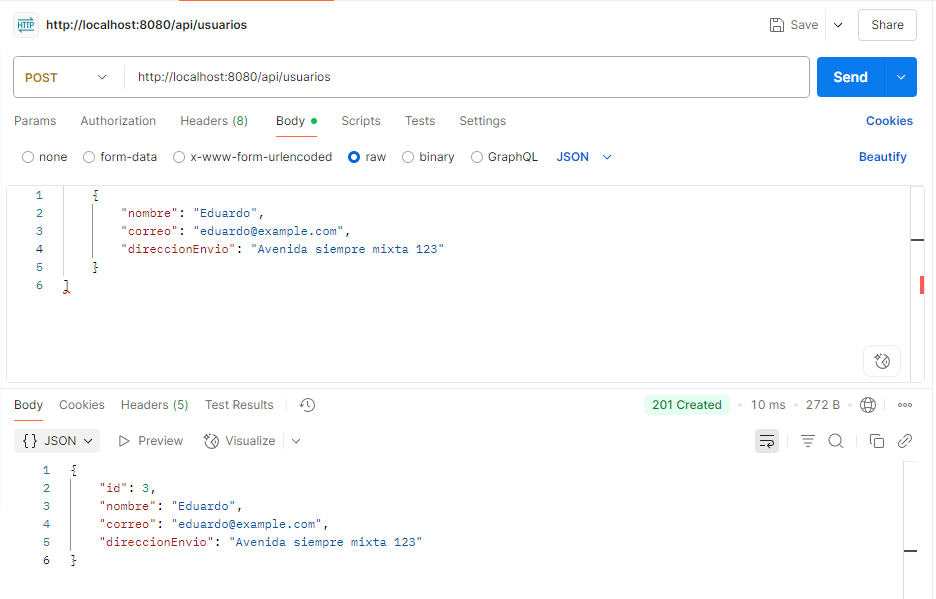
* **Get**



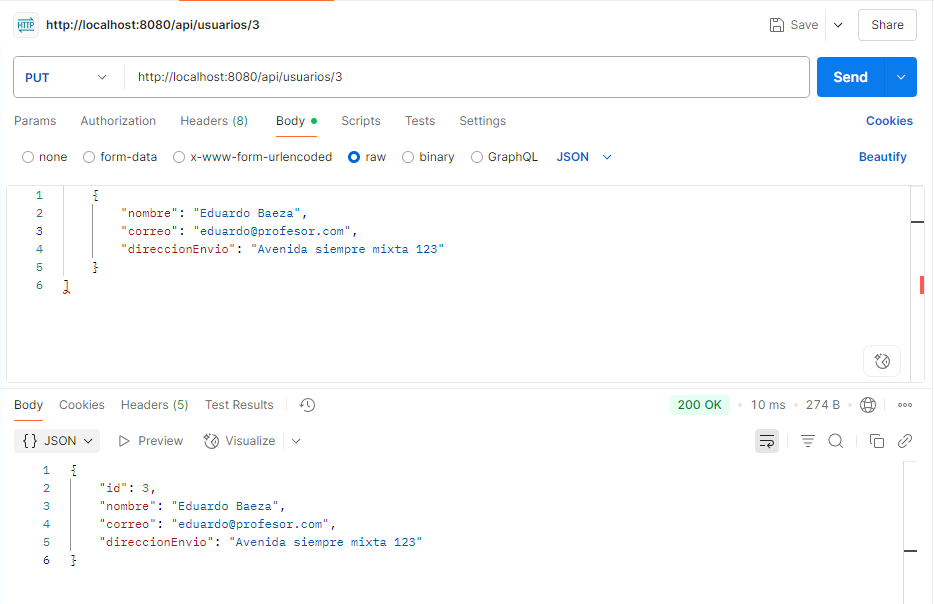
* **Get (id)**



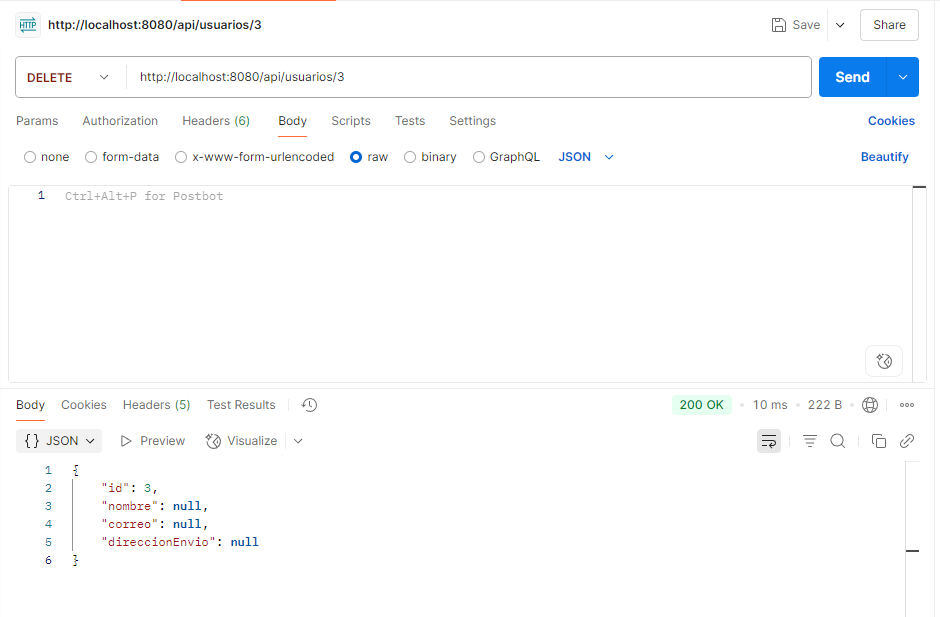
* **Post**



* **Put**

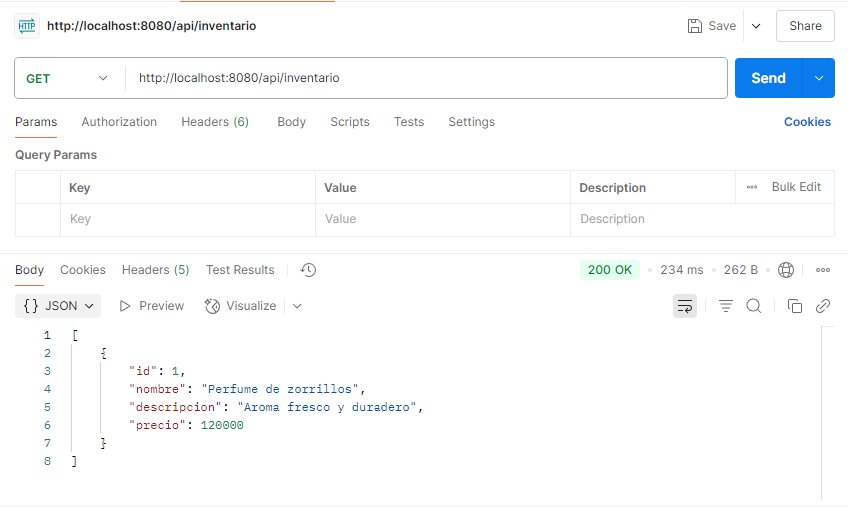


* **Delete**

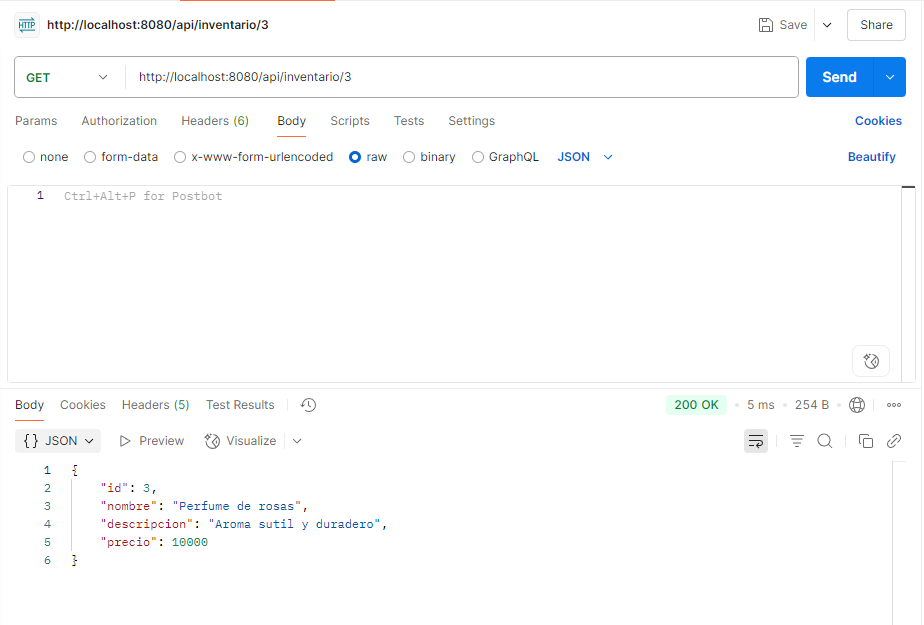


**Inventario:**

* **Get**



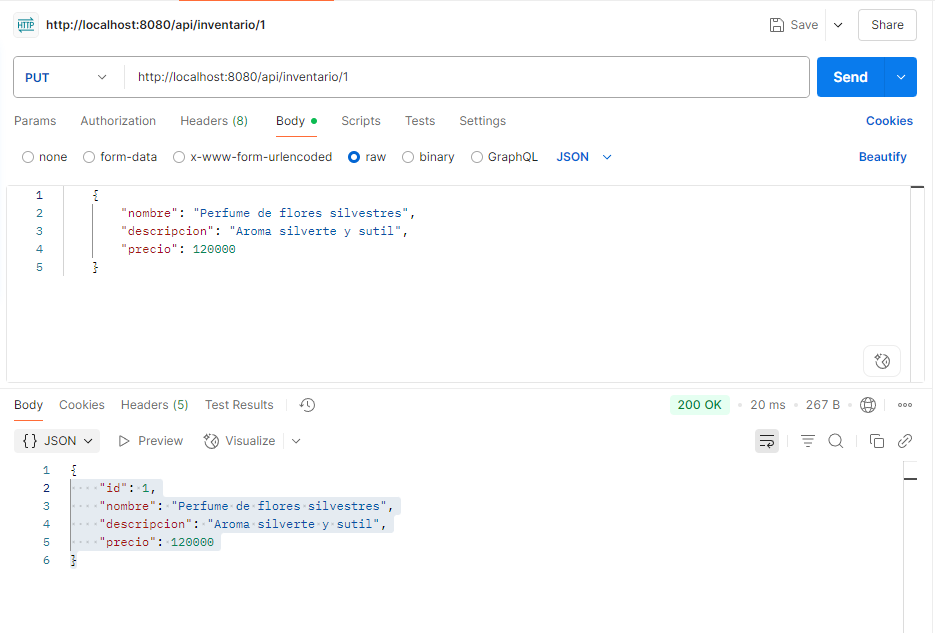
* **Get (id)**



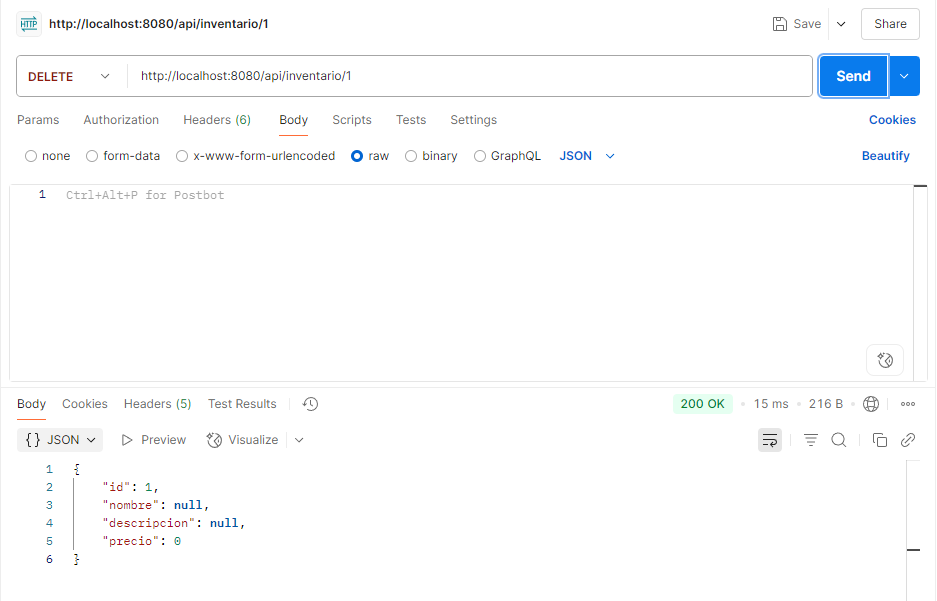
* **Post**

## 

* **Put**

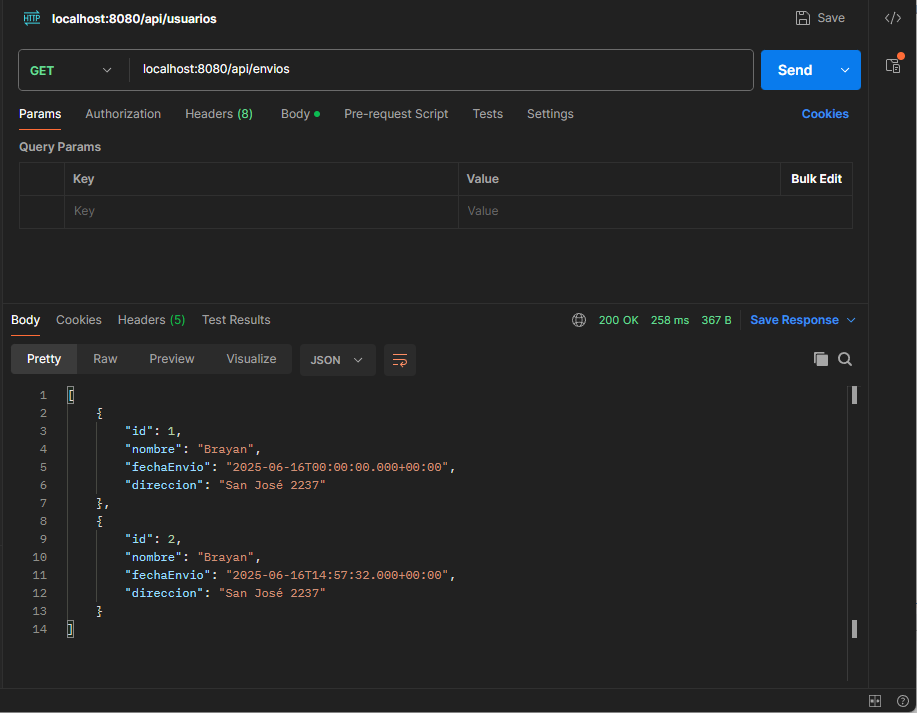


* **Delete**

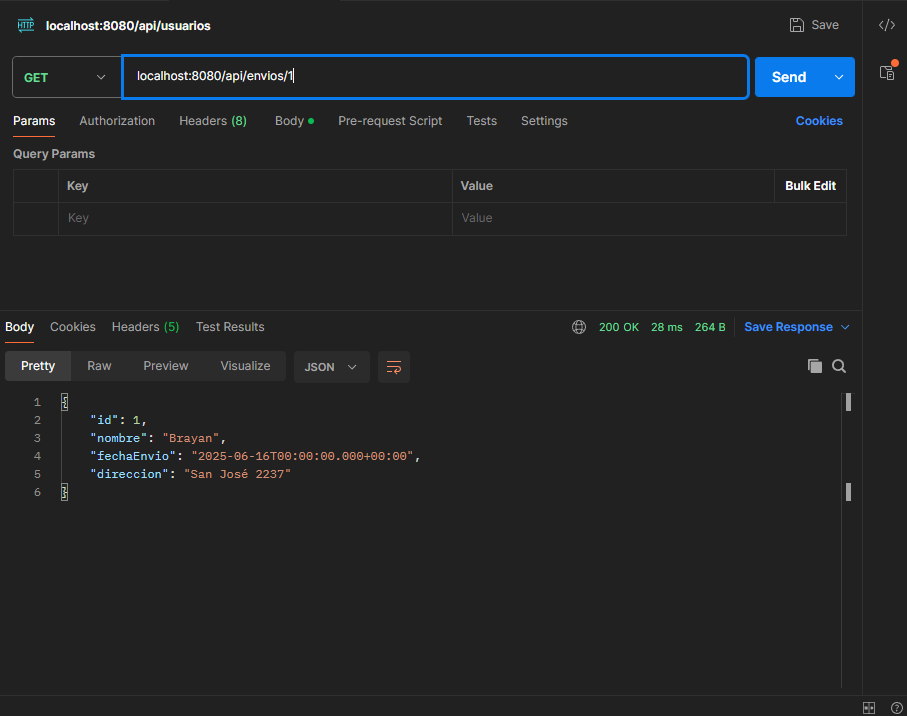


**Envíos**

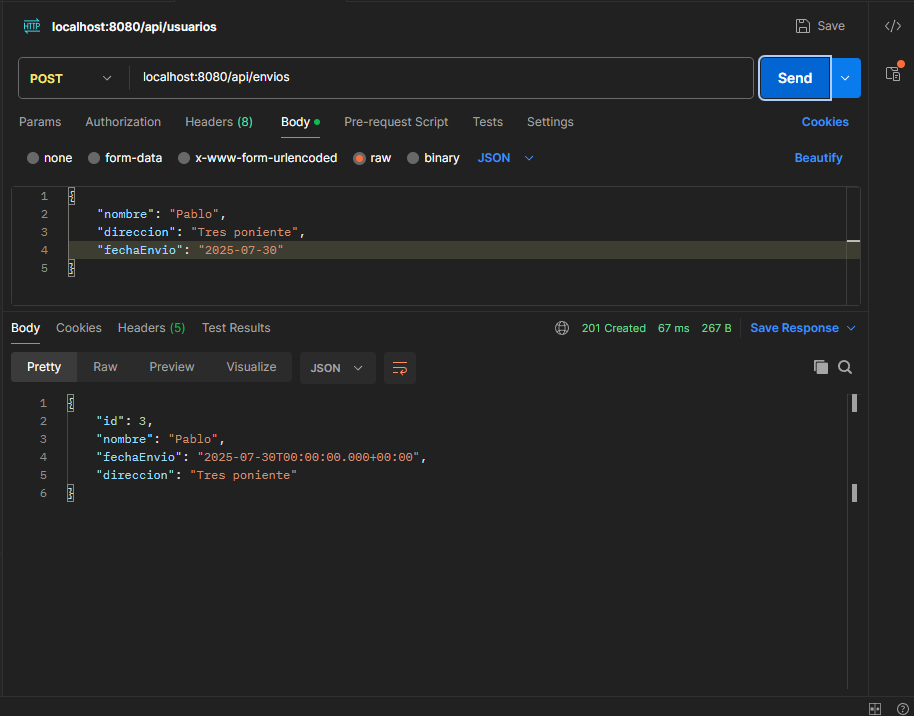
* **Get**



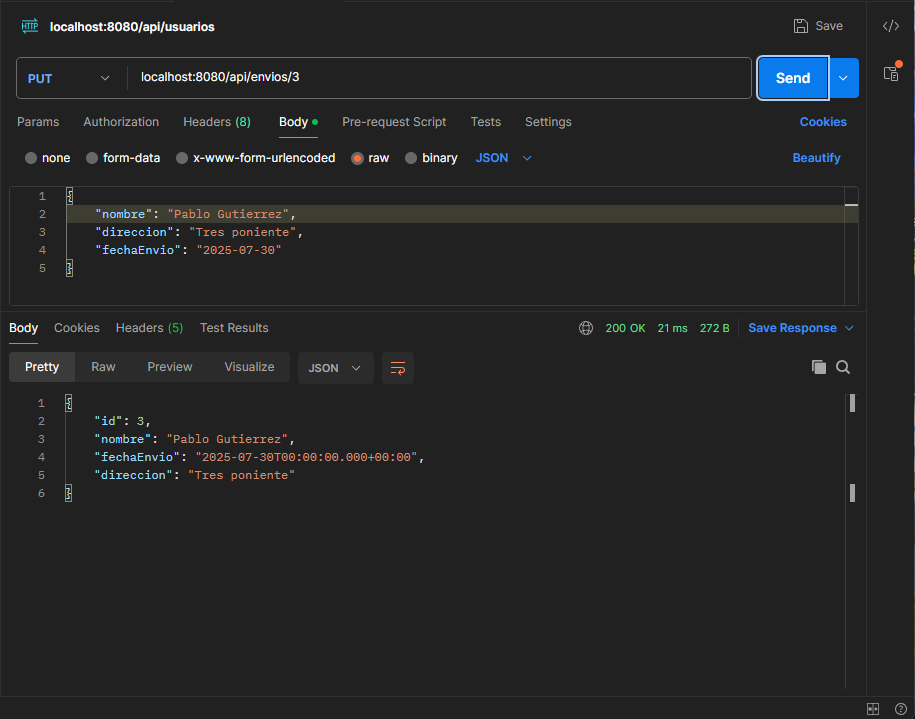
* **Get (id)**



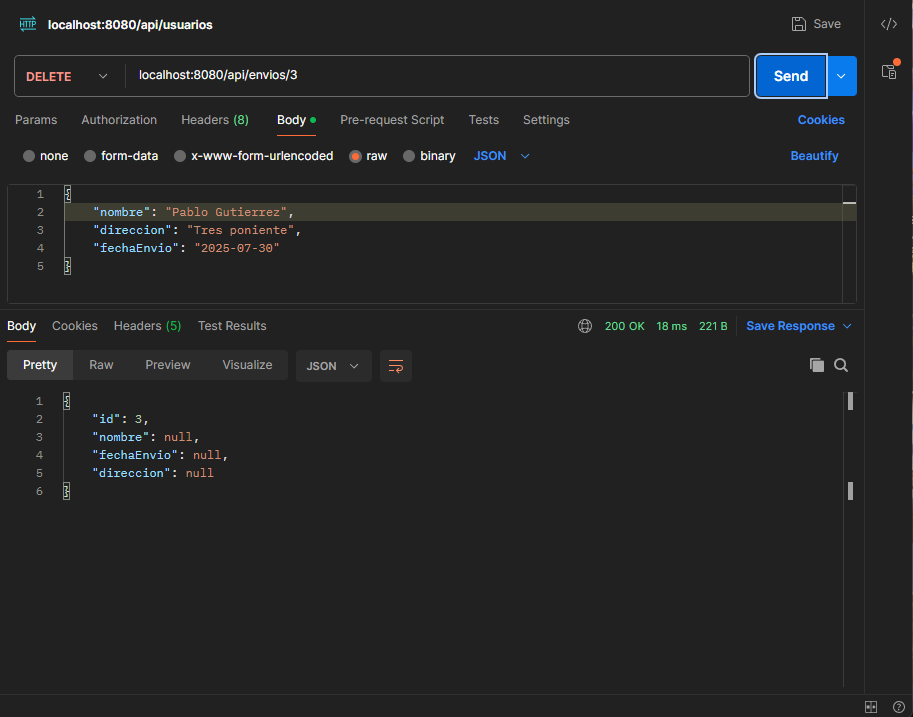
* **Post**



* **Put**



* **Delete**



## **Fragmentos del código relevantes.**

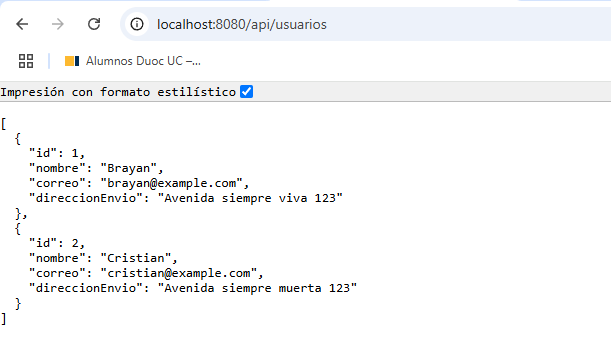




## **Confirmar interacción con la base de datos.**

## **Vistas para petición GET**

## **Captura de pantalla de alguna interfaz simple donde se muestran los datos consultados desde los servicios.**



## **Breve explicación de cómo se conecta esa vista con el backend.**

La vista envía peticiones HTTP al backend para pedir o enviar datos, el backend tiene controladores y componentes que reciben estas peticiones (como @RestController) , luego el servicio usa repositorios para interactuar con la base de datos (como CrudRepository) para finalizar guardando los datos entregados o leyendo los datos pedidos.

## **Git – GitHub**

## **Explicación de comandos Git usados (git init, git add, git commit, git push, git clone, etc.).**

$ git init: Inicializa un nuevo repositorio Git en la carpeta seleccionada.

$ git add: Agrega archivos a la “fase de preparación”, esto quiere decir que los prepara para subirlos, pero no los sube a un repositorio.

$ git commit: Guarda en el historial de Git todos los cambios previamente añadidos con $ git add, además de asociar un mensaje que indique qué cambios existen en ese “commit”.

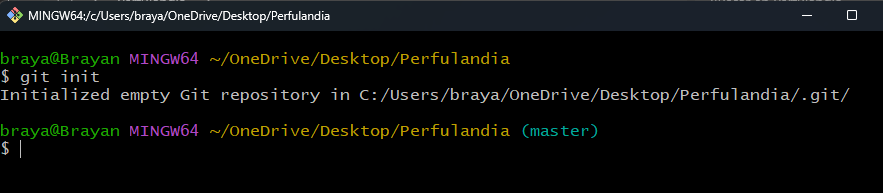
$ git push: Sube los “commits” previamente hechos con $ git commit hacia el repositorio en GitHub.

$ git pull: Actualización la versión “local” del proyecto con la versión actual en GitHub.

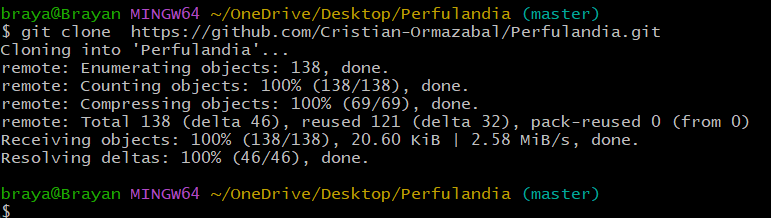
$ git clone: Clona el repositorio hacia una carpeta local, se necesita el url del repositorio de GitHub.

## **Capturas del terminal con estos comandos en uso.**

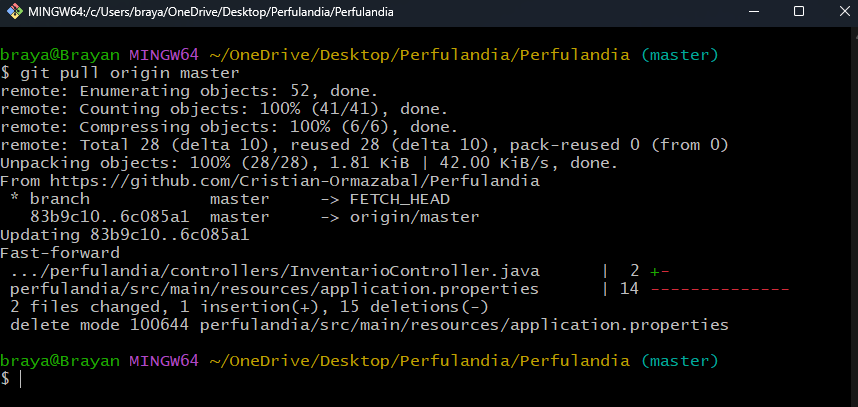
* + 1. **$ git init**

****

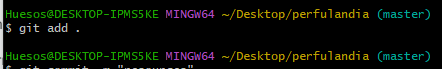
* + 1. **$ git clone**

****

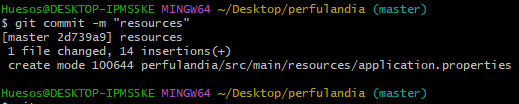
* + 1. **$ git pull**

****

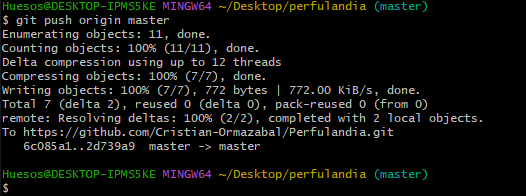
* + 1. **$ git add**



* + 1. **$ git commit**



* + 1. **$ git push**



## **Imagen del repositorio en GitHub.**

## **Conclusión**

## **Reflexión del equipo sobre el trabajo realizado.**

Abordar la transformación de un sistema monolítico hacia una arquitectura de microservicios fue un desafío ya que había varios elementos a tener en consideración como lo sería entender el funcionamiento de cada componente utilizado, esto es importante porque cualquier mal uso de estos puede ocasionar el mal funcionamiento del código, por otra parte, es importante la utilización de distintas herramientos para verificar la correcta ejecución del programa, como lo sería “Postman”, por último, la utilización de git fue una experiencia enriquecedora e interesante que sin duda se le dará uso en un futuro.

## **Dificultades enfrentadas y cómo se solucionaron.**

Si bien no existieron una gran variedad de dificultades, si hubieron algunas que fueron estresantes. El archivo se creó en un principio en una versión de Springboot 3.4.5, sin embargo, al trabajar de manera local en nuestros computadores, nos tiraba error en el archivo pom.xml, y no sabíamos porqué, luego entendimos que la versión instalada en nuestros computadores era 3.4.6, y la versión que tenía el archivo estaba desactualizada, esto se arregló modificando el archivo pom.xml, cambiando la versión que indicaba ahí, con la versión que teníamos, por otra parte, a un inicio tiró error la conexión a la base de datos, si bien la conexión resultaba bien dentro de los computadores del DUOC, en nuestras casas no funcionaba, y era por la misma razón que mencioné del problema anterior, nuestras versiones de mysql eran distintas, buscando soluciones y probando qué funcionaba, logramos hacerlo funcionar poniendo en la dependencia de mysql, una versión de 9.1.0, luego de esto se realizó correctamente la conexión a la base de datos, después de estos dos problemas no hubo grandes dificultades, más que una letra mal escrita, o un punto y coma que se nos pasó poner.

## **Importancia del uso de microservicios en el desarrollo moderno.**

Es importante el uso de microservicios ya que esta permite desarrollar, desplegar y escalar componentes de manera independiente, logrando así una mejor mantención y disponibilidad del sistema, además de facilitar la integración de nuevas funcionalidades que sean requeridas en un futuro. Asimismo, los microservicios promueven la resiliencia del sistema, esto quiere decir que si falla un servicio, no afecta al resto de los servicios. En definitiva, migrar hacia microservicios no solo responde a una necesidad técnica, sino que también representa una inversión estratégica en la sostenibilidad y evolución tecnológica de la empresa.