

**Informe caso de estudio**

**Perfulandia**

Desarrollo Full Stack 010D

Integrantes:

* Tomas Heise
* Eugenio Parada
* Joaquín Zenteno

Índice:

Análisis del sistema actual:  
 Descripción del sistema monolítico…………………………  
 Fallas y sobrecargas del sistema…………………………………

Análisis de requerimientos:  
 Requisitos funcionales………………………………………………  
 Requisitos no funcionales…………………………………………  
 Entrevistas a usuarios………………………………………………

Diseño de la nueva arquitectura:  
 Arquitectura de microservicios propuesta…………………………  
 Diagrama de caso de uso…………………………………………  
 Diagrama de clases…………………………………………………  
 Diagrama de microservicios……………………………………….

Planificación de la migración:  
 Respaldo del sistema……………………………………………….  
 Preparación del nuevo entorno…………………………………….  
 Transferencia de datos………………………………………………  
 Revisión final y pruebas…………………………………………….  
 Cambio de DNS………………………………………………………  
 Etapa post-migración…………………………………………………

Gestión de riesgos y mitigación:  
 Identificación de riesgos…………………………………………….  
 Estrategias de mitigación…………………………………………….

Estructura del proyecto:  
 Herramientas utilizadas……………………………………………..  
 Componentes del proyecto………………………………………….  
 Explicación de cada parte del código………………………………

Base de datos:  
 Motor utilizado…………………………………………………………  
 Tablas y relaciones……………………………………………………  
 Imagen del modelo…………………………………………………….

Implementación de servicios REST:

Servicios desarrollados………………………………………………..  
 Operaciones CRUD por servicio………………………………………  
 Pruebas con Postman…………………………………………………..

Gestión con Git y GitHub:  
 Comandos utilizados  
 Capturas de pantalla  
 Rutas documentadas en el archivo README………………..

Conclusión…………………………………………………………………

Análisis del sistema actual

Descripción del Sistema Monolítico:

El sistema descrito en el caso de Perfulandia SPA es una aplicación monolítica que gestiona las operaciones de la empresa, incluyendo ventas, inventario, envíos, facturación y administración de usuarios. Toda la lógica de negocio, la base de datos y la interfaz de usuario se encuentran integradas en una única aplicación desplegada en un servidor principal desarrollada en un único bloque de código extenso.

El sistema descrito previamente presentaba ventajas para Perfulandia, tales como que el desarrollo y la gestión del código era más sencillo, la administración de la seguridad era más simple y el despliegue era más fácil de ejecutar, entre otros. Todo se vio envuelto en un cambio drástico en el momento en el que Perfulandia comenzó a crecer y el mismo sistema que en un momento les sirvió para mantener su lógica de negocio, ahora se presenta como uno con severos problemas de sobrecargas y fallas de software, entre estás podemos encontrar:

* Cuellos de botella en la base de datos: todas las solicitudes de los distintos usuarios pasan por una misma base de datos central, lo que genera alta concurrencia y tiempos de respuesta elevados.
* Altos tiempos de carga: con el crecimiento del volumen de transacciones, las consultas y procesamiento de datos se vuelven más lentos, afectando la experiencia de usuario.
* Fallas en la Disponibilidad: cuando algún módulo del sistema fallaba, toda la aplicación se veía afectada, causando interrupciones en las operaciones de venta, inventario y logística.
* Dificultad en la integración con nuevas sucursales: Cada vez que se añada una nueva sucursal, el sistema deberá ser ajustado y actualizado manualmente, lo que al final generará tiempos de inactividad.
* Errores en la sincronización de datos: La replicación de datos entre sucursales y la central no siempre será inmediata, causando inconsistencias en inventario y pedidos.
* Limitaciones en el crecimiento del negocio: La rigidez del sistema impide la incorporación rápida de nuevas funcionalidades sin afectar el rendimiento general.

Análisis de requerimientos:

Requisitos funcionales:

Con respecto a la gestión de Usuarios:

* El sistema debe permitir al Administrador crear, actualizar, desactivar y eliminar cuentas de usuario.
* El Administrador debe poder asignar y modificar permisos de acceso a diferentes módulos y funciones del sistema.

Para la gestión de Inventario:

* El Administrador debe poder agregar, actualizar y eliminar productos del inventario.
* El sistema debe permitir ajustar las cantidades de stock en tiempo real.

En cuanto a la generación de reportes:

* El Administrador podrá generar reportes de ventas
* El sistema debe permitir exportar estos reportes en formatos como CSV y PDF

En cuanto a las acciones de los clientes:

* Los clientes deben poder crear cuentas, iniciar sesión, y gestionar su perfil.
* El sistema debe permitir a los clientes buscar productos, agregar productos al carrito, y completar el proceso de compra.
* Los clientes deben poder consultar su historial de pedidos y realizar solicitudes de soporte en línea.
* Los clientes podrán dejar reseñas y calificaciones de productos.

Para las seguridad y respaldo de datos:

* El sistema debe permitir realizar copias de seguridad periódicas y restaurar los datos en caso de pérdida.
* El sistema debe contar con mecanismos de autenticación segura para usuarios y administradores.

Requisitos No funcionales

En el campo de rendimiento:

* El sistema debe ser capaz de manejar un número elevado de transacciones.
* El tiempo de respuesta para la búsqueda de productos y el proceso de compra no debe superar los 2 segundos.
* El sistema debe ser capaz de procesar al menos 1000 transacciones por minuto sin degradar el rendimiento.

En la disponibilidad:

* El sistema debe estar disponible el 99.9% del tiempo

Para la escalabilidad:

* El sistema debe ser escalable para permitir el crecimiento futuro sin necesidad de modificaciones mayores en la infraestructura del código.
* Debe ser capaz de soportar un aumento en el número de usuarios concurrentes y transacciones a medida que la empresa crece.

En cuanto a la usabilidad del sistema:

* El sistema debe ser intuitivo y fácil de usar para todos los usuarios
* Debe contar con una interfaz de usuario amigable y accesible tanto en dispositivos de escritorio como móviles.
* El proceso de compra debe ser sencillo, con pasos claros y opciones para modificar la orden antes de finalizar.

Pensando en la seguridad

* El sistema debe proveer seguridad y confidencialidad en los datos entregados por el cliente

Con el mantenimiento:

* El sistema debe permitir actualizaciones sin interrupciones significativas en el servicio.

En el caso de la compatibilidad:

* El sistema debe ser compatible con los principales navegadores web y debe funcionar en dispositivos móviles con sistemas operativos Android e iOS.
* Las actualizaciones del sistema deben ser compatibles con las versiones anteriores, garantizando que no se pierdan datos ni funciones.

Por último en función del soporte:

* El sistema debe ofrecer soporte técnico 24/7 para resolver problemas de operación de la plataforma.
* Los usuarios deben tener acceso a una base de conocimientos y un sistema de tickets o preguntas frecuentes para resolver dudas y problemas.

Entrevista

1) Administrador del Sistema.

¿Cuáles son las tareas más críticas que realiza actualmente como Administrador del Sistema?

* Mi tarea principal es gestionar los usuarios del sistema, como crear, actualizar y desactivar cuentas. Además, configurar los permisos de acceso para asegurar que cada usuario tenga acceso solo a las funciones necesarias para su rol. También me encargo de monitorear el estado del sistema, recibir alertas sobre posibles fallos y realizar copias de seguridad periódicas. Con el crecimiento de la empresa, es crucial que podamos restaurar los datos rápidamente en caso de un fallo.

¿Qué dificultades está experimentando actualmente con el sistema?

* El sistema actual tiene muchos problemas de rendimiento. Durante las horas pico, el acceso a los módulos de gestión y reportes se vuelve lento, lo que retrasa nuestras operaciones. Esto es especialmente preocupante cuando se realizan actualizaciones o cuando hay cambios en los permisos de los usuarios.

¿Qué funcionalidades considera esenciales en la nueva versión del sistema?

* Me gustaría que la monitorización del sistema sea más eficiente, permitiendo visualizar el estado en tiempo real y recibir alertas más específicas. Además, es necesario contar con una función de respaldo y restauración más ágil, para poder recuperar rápidamente los datos si ocurre un problema. También sería importante una mejor gestión de los permisos, con un control más específico sobre las funciones a las que cada usuario puede acceder.

¿Cómo te gustaría que se mejorará la seguridad del sistema?

* La seguridad es una prioridad, por lo que me gustaría que el sistema tenga una forma de autentificar a los usuarios con distintos tipos de roles. Además, el sistema debe proteger toda la información sensible, especialmente la relacionada con las transacciones de los clientes, y cumplir con las normativas de protección de datos.

3) Cliente

¿Cómo describirías tu experiencia general al utilizar la plataforma web de Perfulandia SPA?

* En general, la experiencia es bastante buena, pero podría ser mejor en algunas cosas. Navegar por los productos es fácil, pero a veces el sistema tarda en cargar las páginas, especialmente cuando busco productos específicos. Además, cuando quiero hacer una compra, el proceso de pago puede ser un poco largo si el sistema está lento.

¿Cuáles son las funcionalidades más importantes para ti como cliente?

* Me gusta poder ver el catálogo de productos y poder buscar lo que quiero rápidamente. También es fundamental que el proceso de compra sea rápido y seguro, con múltiples opciones de pago. Otra funcionalidad que considero muy importante es poder consultar mi historial de pedidos. Además, me gustaría tener la opción de agregar cupones de descuento de manera sencilla y ver la aplicación en el total de la compra.

¿Qué problemas has experimentado al usar la plataforma de Perfulandia SPA?

* El sistema se pone lento, lo que me genera dudas si el pedido se procesó correctamente. También he tenido problemas con la visibilidad del estado de mis pedidos, ya que en ocasiones no se actualiza el estado rápidamente. Me gustaría que el sistema me notificara en tiempo real cuando mi pedido ha sido procesado.

¿Qué características adicionales te gustaría ver en la nueva versión del sistema?

* Sería genial que el sitio web fuera más rápido y más accesible en dispositivos móviles. Además, me gustaría tener un sistema de notificaciones que me avise cuando haya ofertas o descuentos especiales.

Diseño de la nueva arquitectura

La nueva arquitectura será basada en microservicios en lugar de un sistema monolítico, dividiremos la aplicación en funciones independientes cada uno encargado de ejecutar una acción especifica.

Los microservicios que serán utilizados en esta arquitectura serán los siguientes

* Autenticación: esta se encargará de la autenticación de los usuarios, el control de sesiones y los permisos, manejara el registro de contraseña
* Catálogo de productos: Permite gestionar los productos disponibles en la tienda esto incluye nombres, descripciones, imágenes los precios y categorías la cual responderá consultas sobre la disponibilidad de los productos.
* Pedidos y facturación: Administra el proceso de compra, confirmando pagos, generando las facturas y notificando a clientes sobre el estado de su pedido.
* Intenvario: este controla el stock y la disponibilidad de los productos, manteniendo actualizada el stock de los productos almacenados, este reducirá automáticamente el inventario cuando se lleve a cabo una compra y avisará en caso de bajo stock
* Notificaciones: Enviará correos y mensajes a los clientes, gestiona las notificaciones enviadas a clientes y a los administradores, como confirmación de pedidos, alertas de stock y los mensajes promocionales
* Análisis y reportes: Gestión de reportes de ventas y el desempeño, procesando datos históricos a tiempo real para generar reportes sobre ventas y el comportamiento de los clientes y tendencias de productos más vendidos ayudando a tomar decisiones estratégicas para la empresa

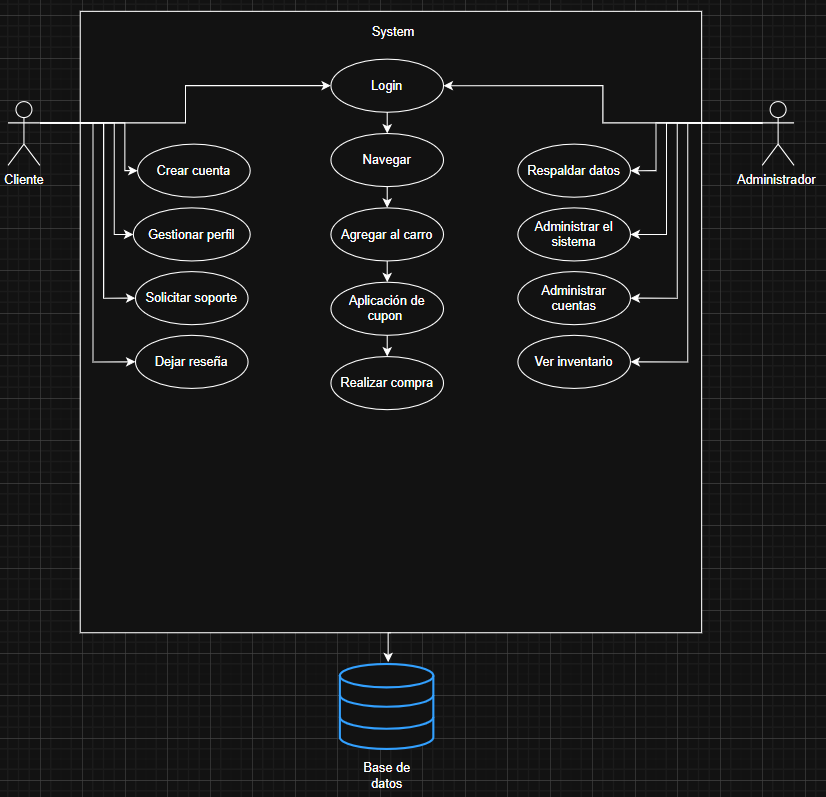
Diagrama de causa de uso 

Diagrama de clase

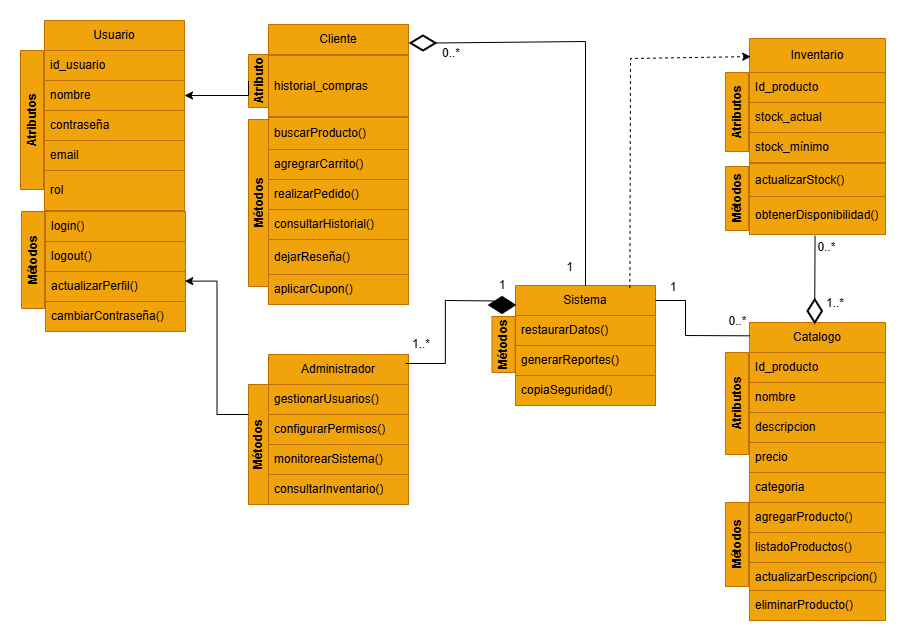
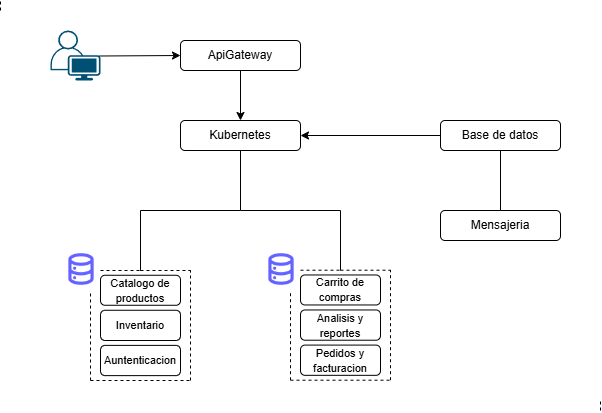
****

Diagrama de Microservicios



Plan de migración

1.- Hacer un respaldo del sitio web:

- Copia de seguridad de archivos

- Exportación de la base de datos

Con esto se podrá evitar la pérdida de información de tanto la base de datos como el del sitio web anterior.

2.- Preparará el nuevo sitio:

- Creación de la base de datos

- Instalación de lo que sea necesario

- Verificación de compatibilidad

de tal manera se proporcionará una transición fluida evitando así complejidades en la transferencia de datos.

3.- Transferir los datos:

- Subir los archivos

- Importar la base de datos

así se garantizará que toda la información del sitio web anterior esté disponible en el nuevo servidor sin pérdida de datos ni inconsistencias.

4.- Últimas revisiones:

- Revisión del enlace, base de datos y carga de archivos

- Analizar errores y corrección de últimos fallos

Esto va a permitir detectar y solucionar posibles errores antes del lanzamiento, asegurando que el sitio web funcione correctamente.

5.- Cambio de DNS:

- Actualizar los registros DNS

- Verificar el acceso al sitio

- Realizar pruebas finales

De este modo, los usuarios podrán acceder al sitio desde su dominio habitual sin interrupciones, y se garantizará una correcta propagación del DNS.

6.- post migración:

- Revisión de rendimiento

- Educación de los empleados

Con este paso, se asegurará que el sitio mantenga un buen desempeño y que el equipo responsable esté capacitado para gestionar el nuevo entorno.

**Riesgos y plan de mitigación**

Pérdida de datos durante la migración

Riesgo: en transferencia de los datos del sistema monolítico al nuevo sistema de microservicios, existe la posibilidad de pérdida de información o corrupción de datos

Mitigación:

* realizar copias de seguridad completas antes de la migración
* implementar pruebas de migración en un entorno de prueba antes de la migración real
* verificar la integridad de los datos después de la migración mediante pruebas de consistencia

Problemas de rendimiento en la nueva arquitectura

Riesgo: una mala optimización de los microservicios puede generar tiempos de respuesta elevados o fallos en momentos de alta demanda.

Mitigación:

* monitorear constantemente el rendimiento y ajustar recursos según demanda
* realizar pruebas de carga y estrés antes de la puesta en producción

Errores de compatibilidad

Riesgo: diferencias entre tecnologías, versiones de software o esquemas de datos pueden generar fallos en la interacción entre componentes.

Mitigación:

* garantizar compatibilidad con versiones anteriores
* implementar pruebas de integración entre microservicios antes del lanzamiento
* mantener documentación actualizada sobre dependencias y protocolos de comunicación

Vulnerabilidades de seguridad

Riesgo: los datos pueden quedar vulnerables por algún error de parte de los programadores y así comprometiendo el proyecto.

Mitigación:

* implementar autenticación y autorización robustas
* realizar auditorías de seguridad y pruebas de penetración regularmente
* aplicar principios de seguridad en el desarrollo

Resistencia al cambio por parte de los empleados

Riesgo: los empleados tienen dificultades para adaptarse al nuevo sistema, lo que va a afectar la productividad y generar errores en el uso de esta.

Mitigación:

* capacitar al personal sobre el nuevo uso del sistema antes de la migración
* incluir a los empleados en el proceso de transición para reducir la resistencia
* proporcionar documentación clara y guías de usuario

Costos elevados de mantenimiento y operación

Riesgo: la administración de los microservicios van a generar un aumento en los costos operativos y de infraestructura.

Mitigación:

* utilizar arquitecturas escalables que optimicen los recursos
* evaluar modelos de pago por uso en la nube para evitar gastos innecesarios
* optimizar el código y eliminar servicios innecesarios para reducir costos

Fallos en la integridad de los datos

Riesgo: los datos pueden migrar de manera incorrecta, generando inconsistencias o registros duplicados.

Mitigación:

* comparar datos antes y después de la migración con herramientas de verificación
* diseñar un plan de recuperación en caso de inconsistencias

Problemas de Desempeño en el Nuevo Sistema

Riesgo: el nuevo sistema puede no responder adecuadamente a las altas demandas o transacciones simultáneas.

Mitigación:

* utilizar herramientas de monitoreo para detectar cuellos de botella
* implementar balanceo de carga entre los microservicios
* optimizar consultas a bases de datos y usar caché donde sea necesario

Costos No Previstos

Riesgo: puede haber costos adicionales por infraestructura, licencias o servicios no considerados en la planificación.

Mitigación:

* definir un presupuesto detallado con un margen de contingencia
* evaluar previamente los costos de infraestructura y servicios en la nube
* realizar un análisis de costo-beneficio antes de cada implementación

Impacto en la Experiencia del Usuario

Riesgo: los clientes pueden encontrar problemas con el nuevo sistema, como dificultades en las compras o errores en la interfaz.

Mitigación:

* realizar pruebas de usabilidad con usuarios antes del lanzamiento
* recoger retroalimentación continua y mejorar la interfaz de usuario
* optimizar el rendimiento para tiempos de carga mínimos

Problemas con la Configuración de DNS

Riesgo: si la configuración de DNS no se actualiza correctamente, los usuarios podrían no acceder a la página después de la migración.

Mitigación:

* planificar la actualización de DNS en un horario de menor usuarios
* mantener el sistema antiguo operativo hasta que la transición sea completa

Estructura del proyecto:

Qué herramientas se ocuparon:

* Visual Studio Code: Donde se creó toda la estructura del proyecto.
* MySQL: Donde se crea y gestiona la base de datos.
* Spring Boot: Framework que nos permite crear de forma más fácil el desarrollo de aplicaciones en Java.
* Javascript: El lenguaje de programación que se utilizó en el proyecto.
* Maven: Herramienta que nos ayuda a gestionar y construir proyectos en Java**.**

Qué componentes tiene el proyecto:

entities: Catalogo, Inventario, Usuario

repository: CatalogoController, CatalogoRepository, InventarioController, InventarioRepository, UsuarioController,

UsuarioRepository

services: CatalogoService, CatalogoServiceImpl, InventarioService, InventarioServiceImpl, UsuarioService, UsuarioServiceImpl

application.properties: Archivo donde se configura la conexión a la base de datos.

pom.xml: Archivo que utiliza Maven para definir todas las dependencias del proyecto.

Qué hace cada parte:

entities: Contiene las clases que representan las tablas de la base de datos como serían Catalogo, Inventario y Autenticación, cada clase define los atributos que se guardan en la base de datos.

repository: Aquí estan las interfaces que se conectan con la base de datos lo cual permiten hacer operaciones como guardar, buscar, actualizar y eliminar datos sin escribir consultas SQL manuales.

service: Contiene la lógica del negocio como por ejemplo, si se necesita validar datos antes de guardar o aplicar ciertas reglas, aquí es donde se implementa.

controller: Son los puntos de entrada para las peticiones HTTP como GET, POST, PUT y DELETE donde cada controlador recibe las solicitudes del cliente por ejemplo, desde Postman o el navegador lo que permite comunicarse con el servicio para ejecutar la lógica correspondiente.

application.properties: Aquí se definen la conexión a la base de datos, necesario para que la aplicación funcione correctamente.

pom.xml: Archivo que usa Maven para gestionar las dependencias del proyecto como las librerías y frameworks lo que permite automatizar la compilación y construcción del proyecto.

#### 

#### 

#### 

#### 

#### Base de datos:

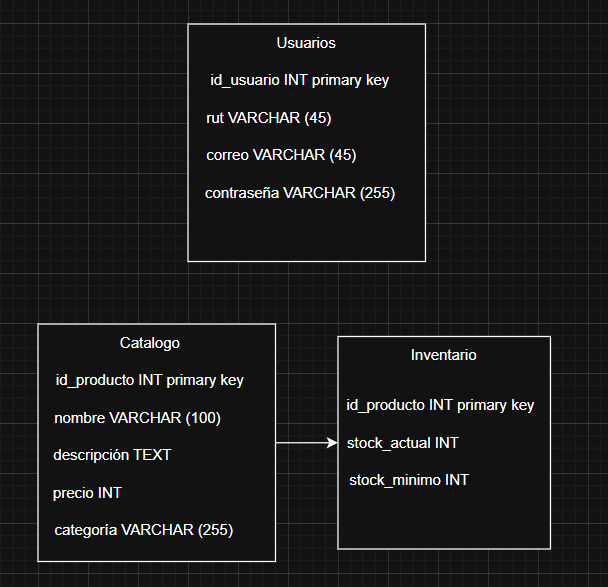
Motos de base de datos utilizado:

El motor de base de datos utilizado para esta etapa del proyecto es el de MySQL WorkBench que brinda una fácil utilización y conexión.

Tablas de la base de datos:

Las tablas utilizadas en la base de datos serian las siguientes:

* Inventario
* Catálogo
* Usuarios

Imagen de la base de datos: 

#### 

#### Implementación de los servicios

Servicios creados:

Los servicios que sean creado se rian los siguientes:

* Inventario: Mostrar el stock de los productos.
* Catálogo de producto: Mostrar la lista de producto con sus detalles y permitir búsqueda.
* Usuario: Registrar nuevos usuarios, permitir inicio de sesión.

Qué operaciones puede hacer cada servicio:

Catálogo:

Este servicio permite gestionar los productos disponibles en la tienda.

* POST: Agregar un nuevo producto al catálogo.
* GET: Obtener la lista de todos los productos.
* PUT:Actualizar los datos de un producto.
* DELETE: Eliminar un producto del catálogo.

Inventario:

Este servicio permite controlar la cantidad de productos disponibles en stock.

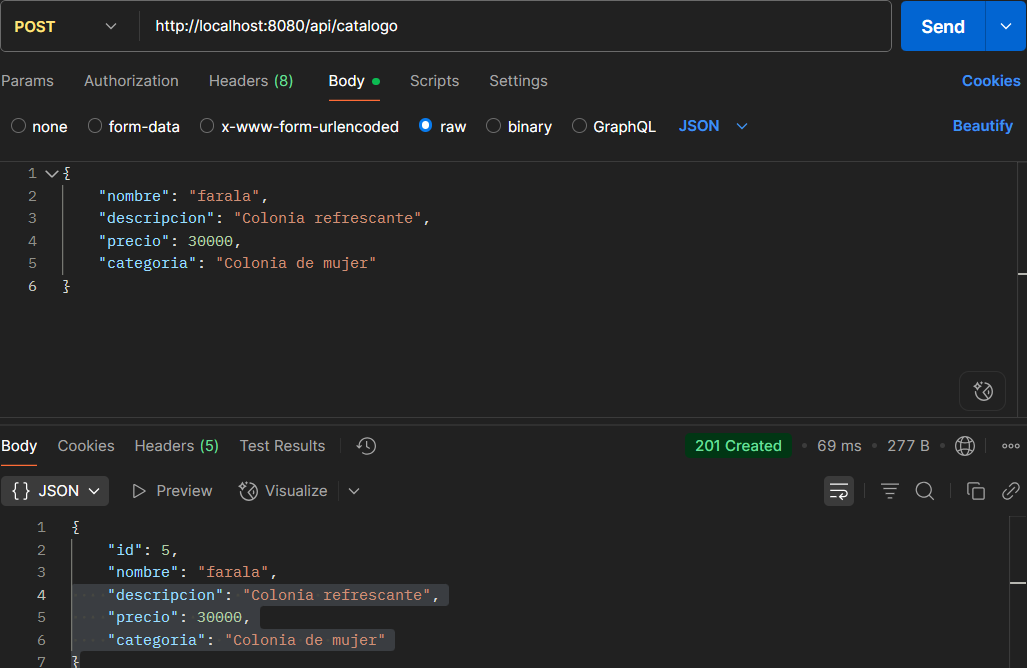
* POST: Agregar un nuevo stock de un producto al catálogo.
* GET: Obtener la lista de stock de todos los productos.
* PUT: Actualizar los datos de stock de un producto.
* DELETE: Eliminar el stock de un producto del catálogo.

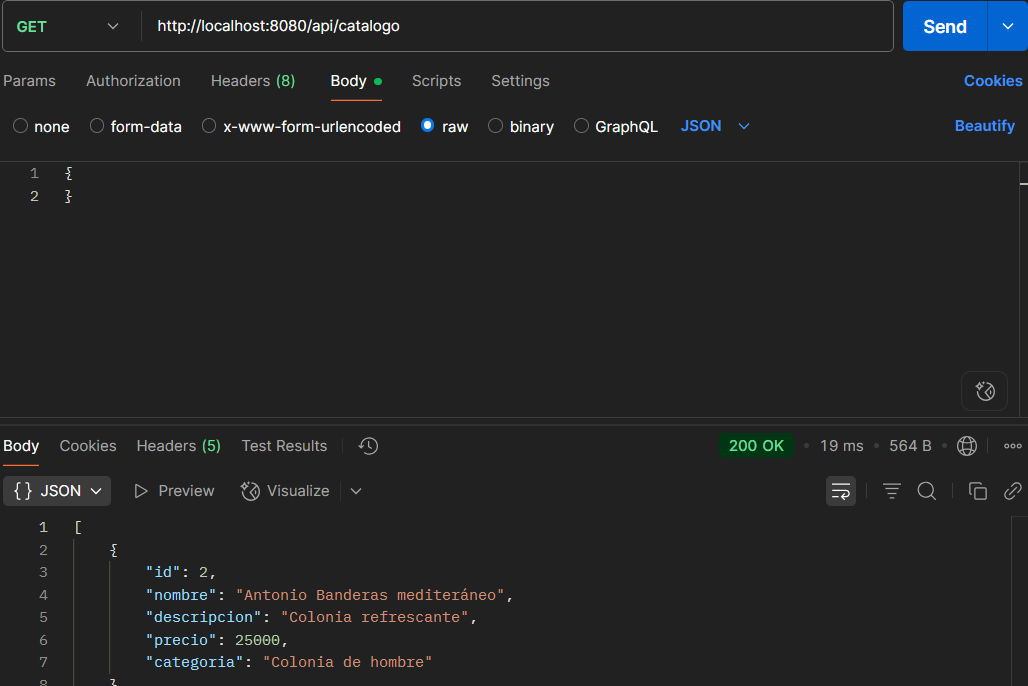
Usuario:

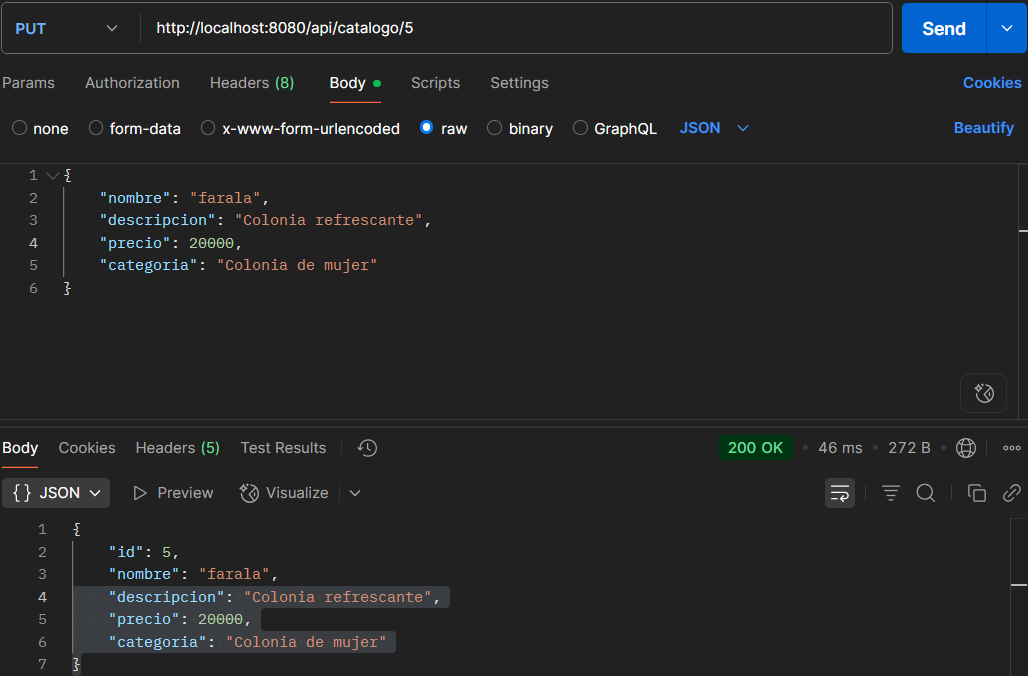
Este servicio permite gestionar los usuarios.

* POST: Agregar un nuevo producto al catálogo.
* GET: Obtener la lista de todos los productos.
* PUT:Actualizar los datos de un producto.
* DELETE: Eliminar un producto del catálogo.

Imagen catálogo:

POST:

GET:

PUT:

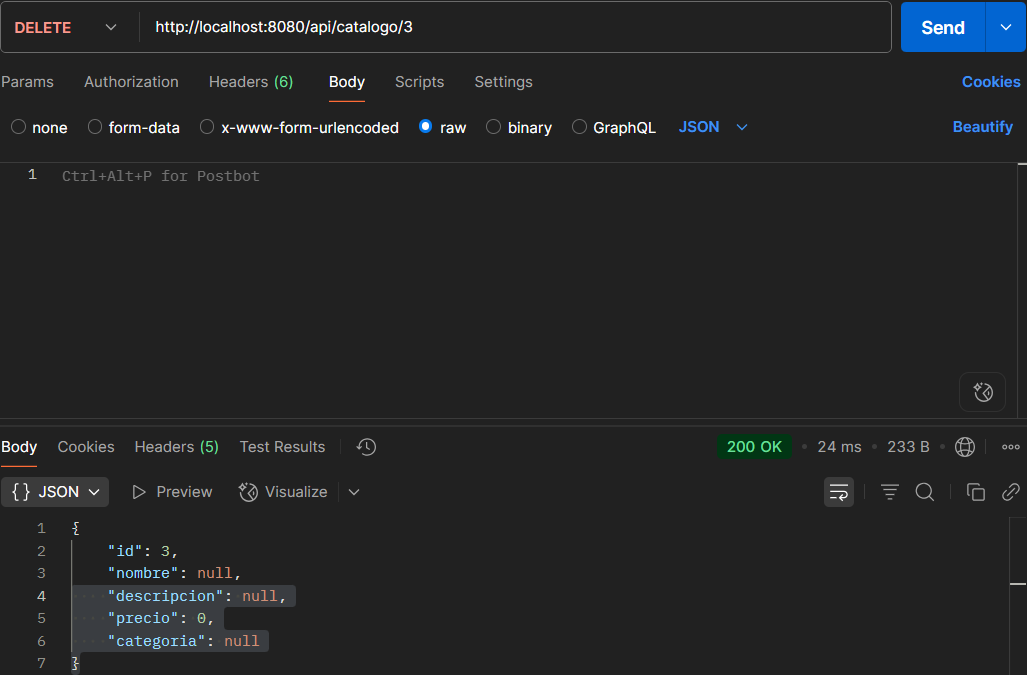
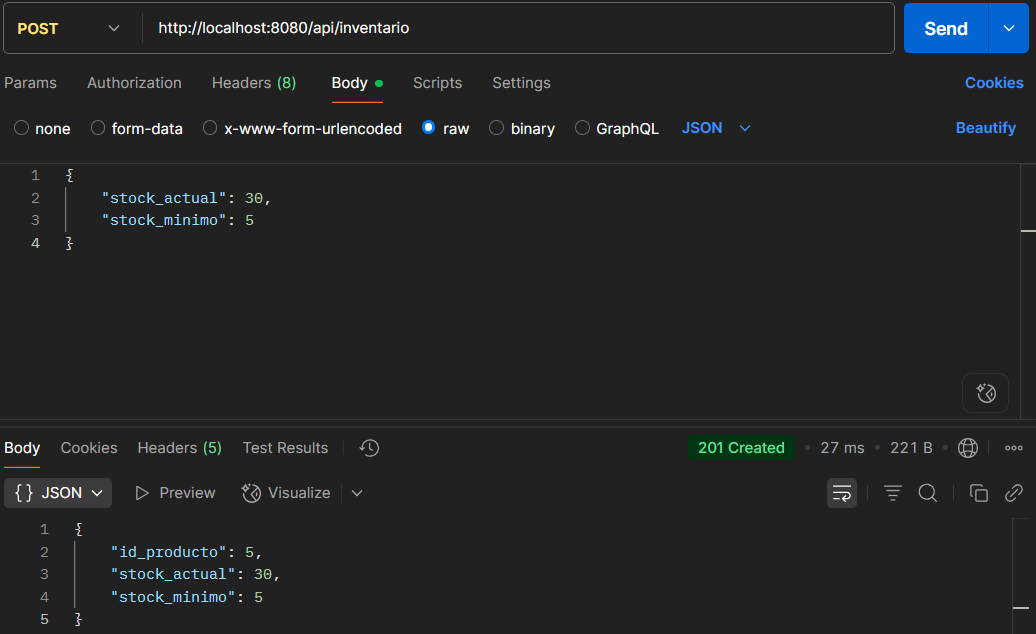
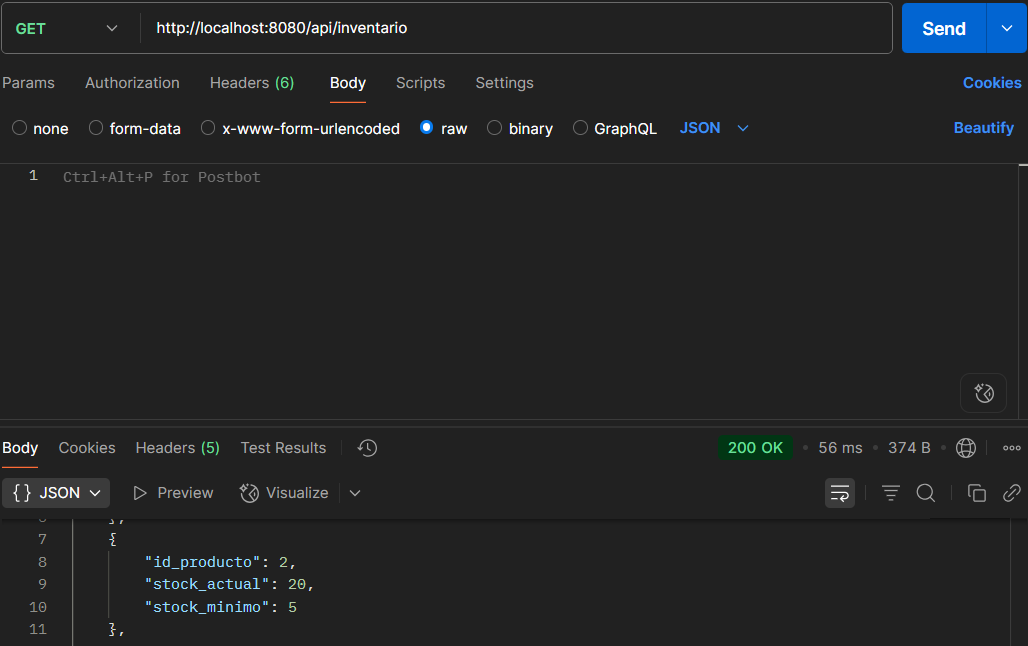
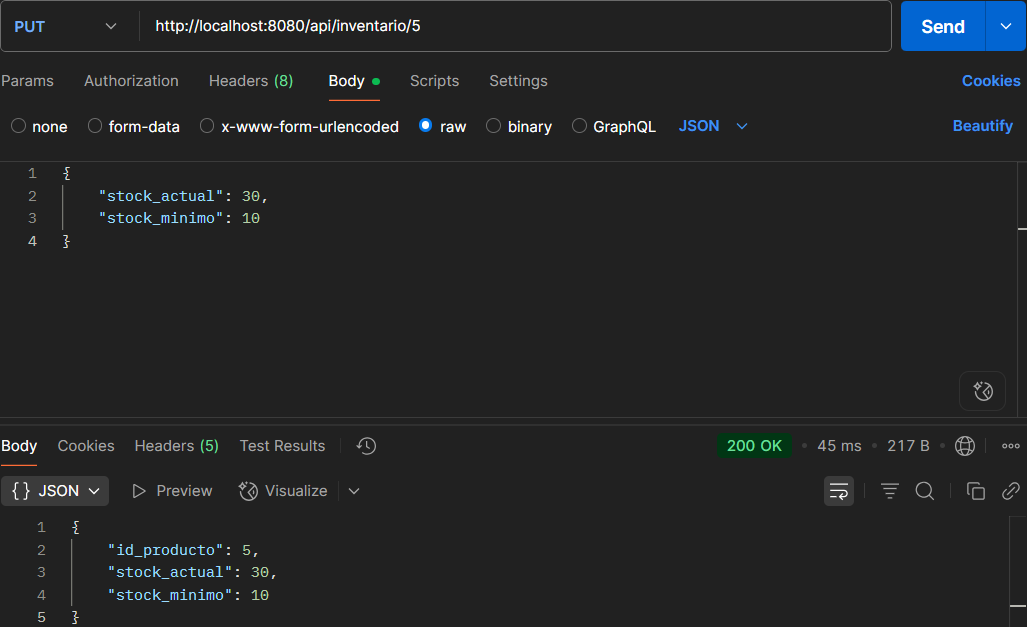
DELETE:

Imagen Inventario:

POST:

GET:

PUT:

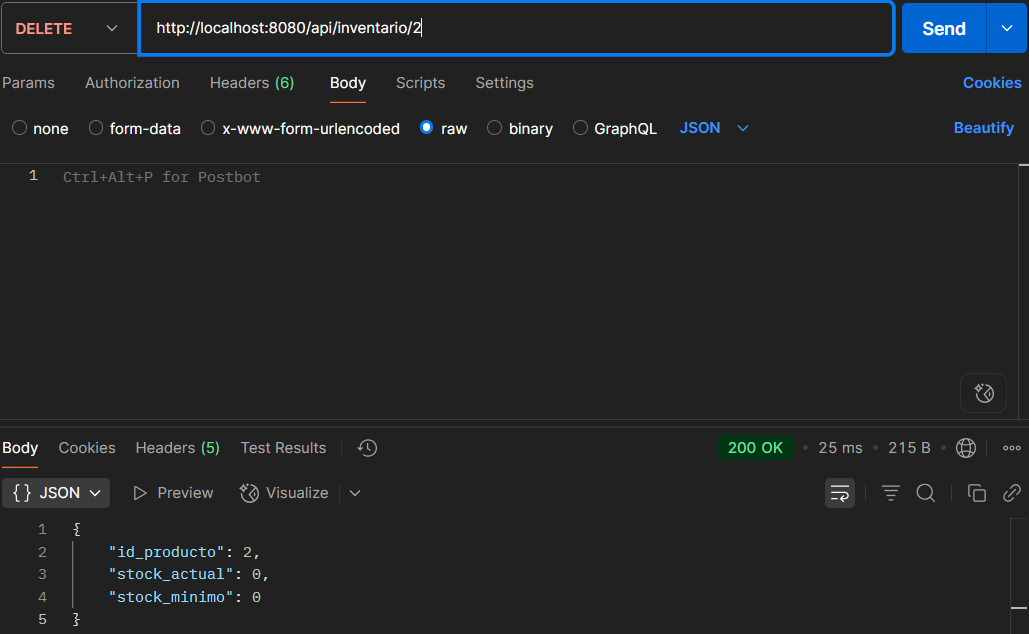
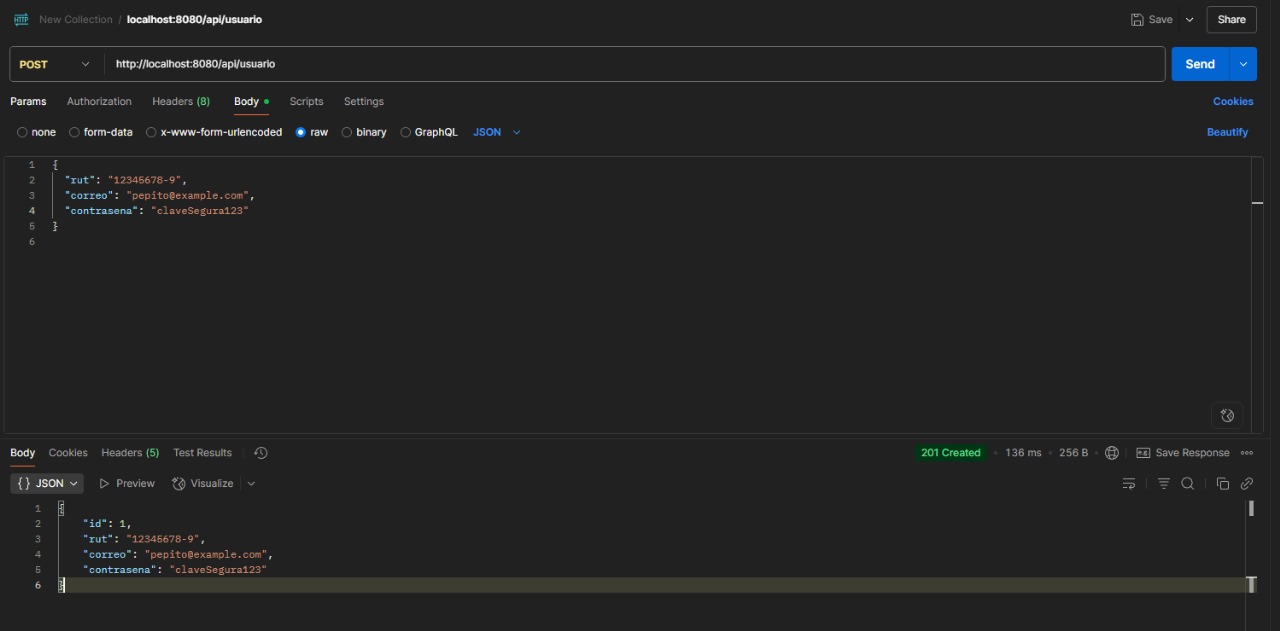
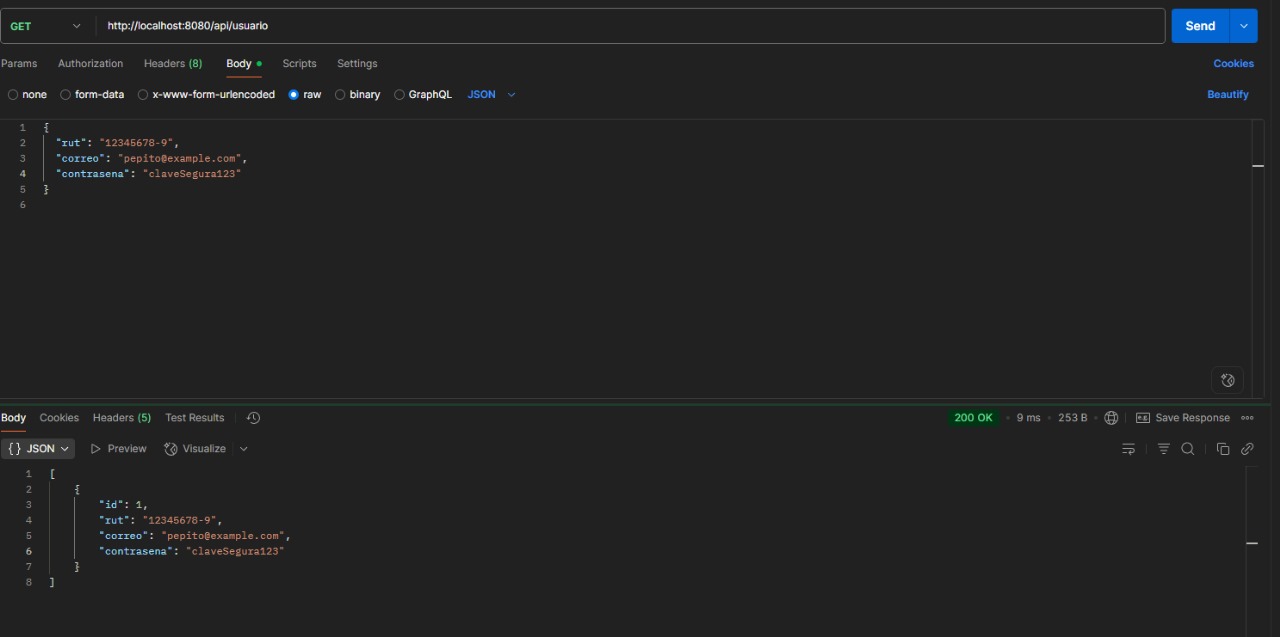
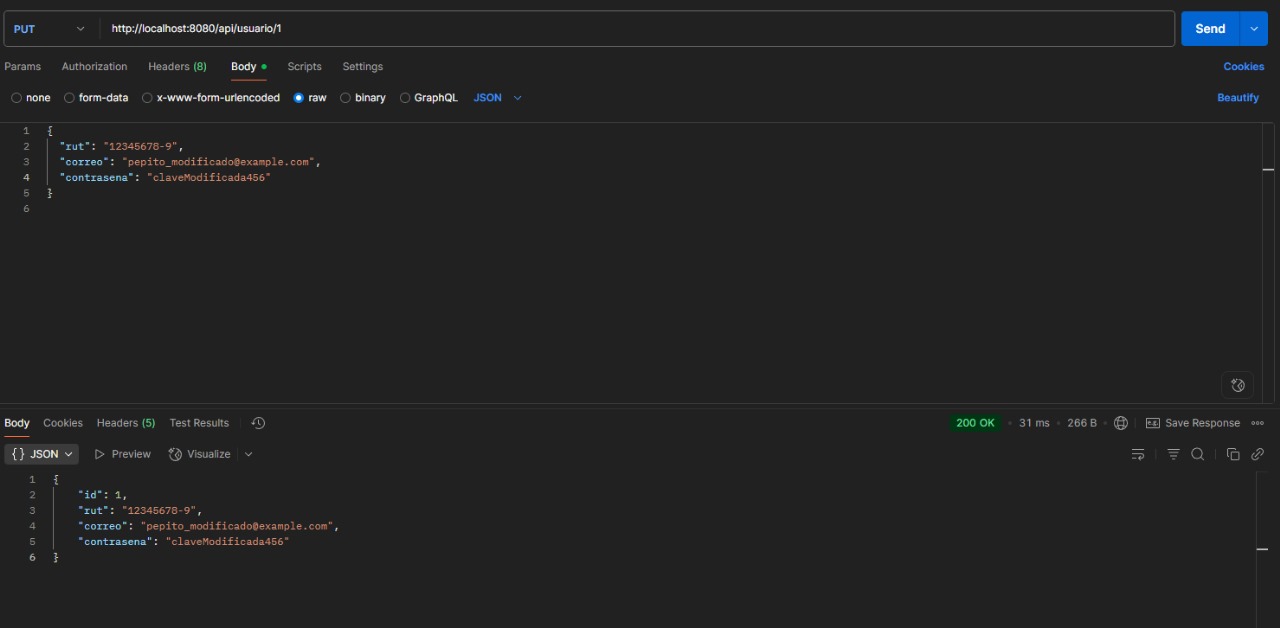
DELETE:

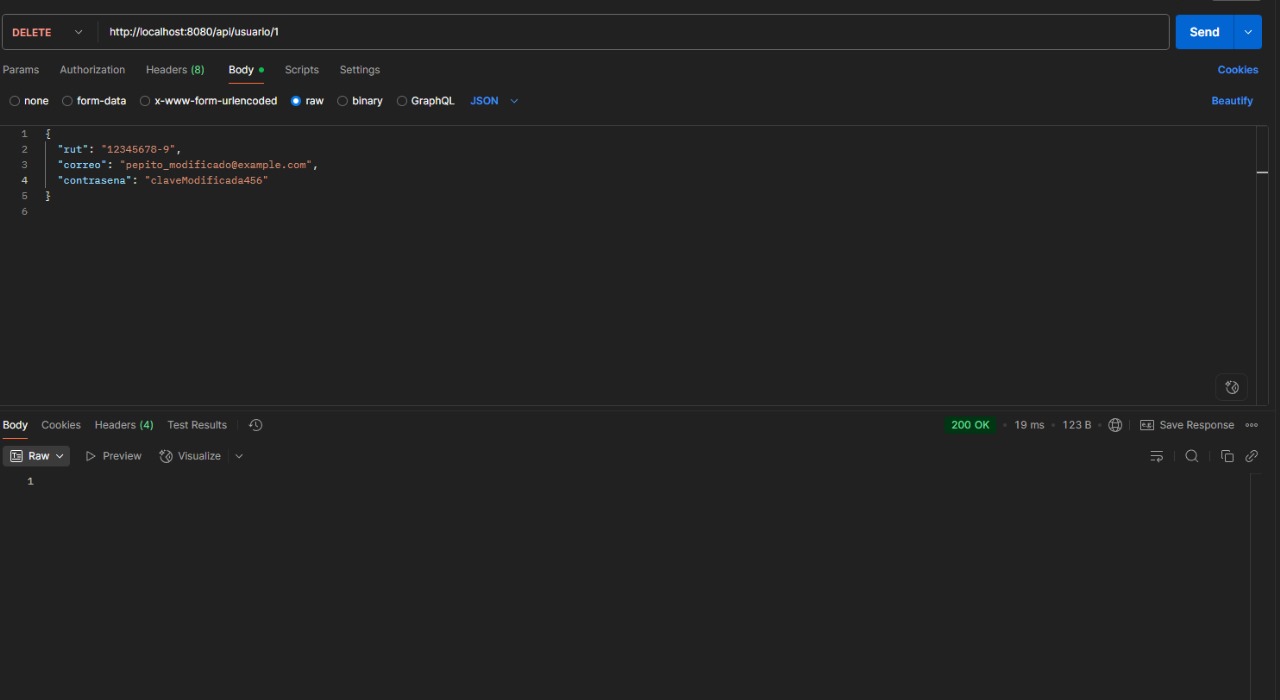
Imagen Usuario:

POST:

GET:

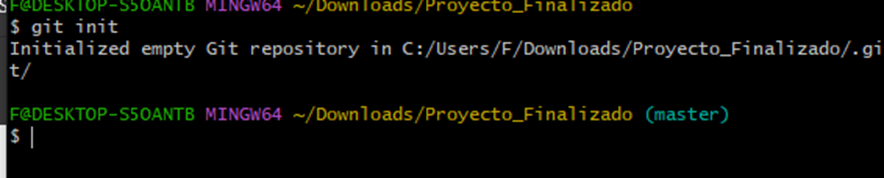
PUT:

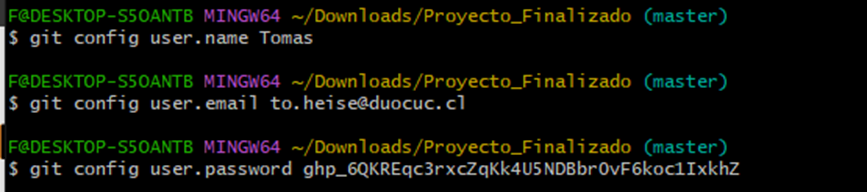
DELETE:



#### 

#### Git y GitHub



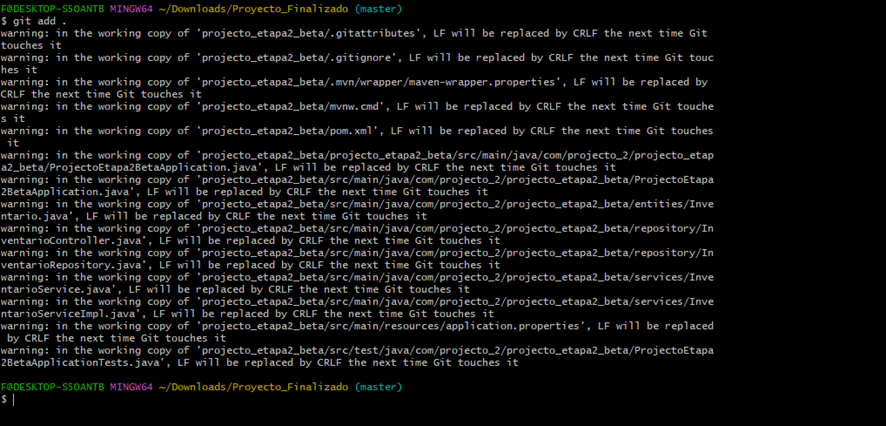


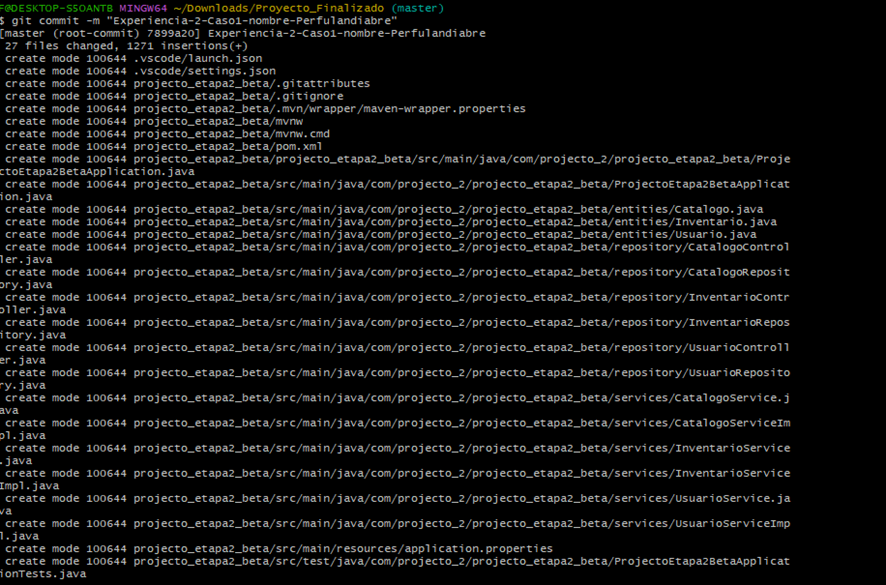
#### 

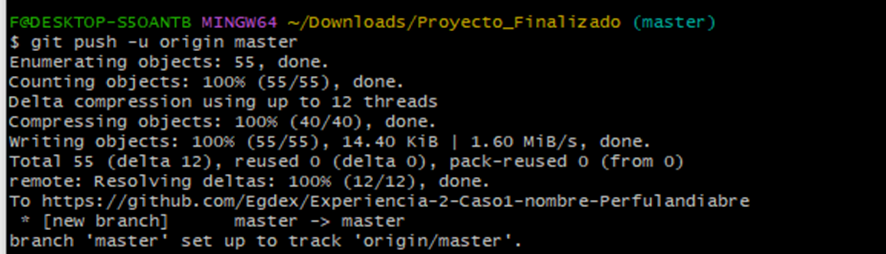
#### 

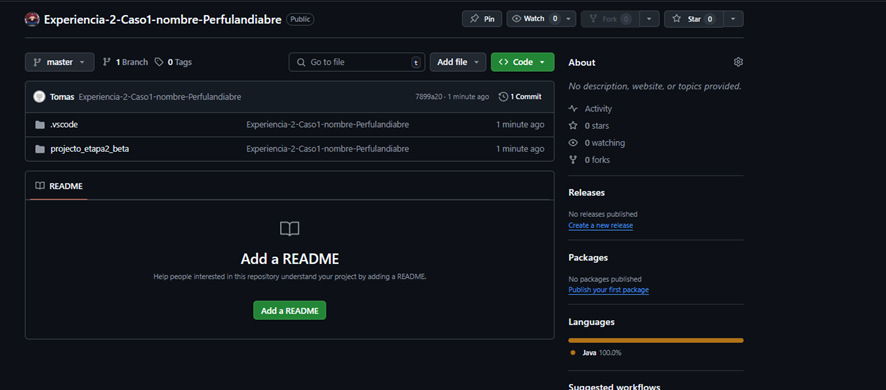
#### 











Conclusión:

Con este proyecto hemos aprendido a desarrollar servicios utilizando Spring Boot y Maven, aplicando la arquitectura de microservicios para reemplazar un sistema monolítico que presentaba varios problemas.

Se pudo lograr implementar de forma correcta los servicios de Usuario, catálogo y inventario, todos con operaciones CRUD y pruebas en Postman.

También usamos Git y GitHub para trabajar en equipo y gestionar el código.

A pesar de algunas dificultades técnicas, como la conexión a la base de datos y la configuración de los servicios, se pudo resolver y completar con éxito los objetivos del proyecto.