UCAB Universidad Católica ANDRÉS BELLO

Informe Proyecto APIS con Python

Marketplace Backend haciendo uso de la arquitectura Microservicios con FastAPI

Prof. Jose Gregorio Castillo Pacheco

Blanco, Alfonso

De Freitas, Yeniree

Gamboa, Paul

Serrano, Manuel

10/01/2024

Ingeniería Informática

Descripción General

Este proyecto es una plataforma de comercio electrónico con diseño fundamentado en la arquitectura de microservicios teniendo en cuenta la escalabilidad, la capacidad de mantenimiento y la modularidad. Aprovecha la arquitectura hexagonal para garantizar una clara separación de preocupaciones y adopta los principios de programación orientada a aspectos para gestionar las preocupaciones transversales de manera eficiente.

La plataforma incorpora las mejores prácticas modernas, como el uso de Optional class para eliminar referencias nulas y de Result class para gestionar los resultados de las operaciones de manera consistente. Estas opciones contribuyen a una base de código sólida y fácil de usar para los desarrolladores, lo que facilita las mejoras futuras y reduce la deuda técnica.

Arquitectura hexagonal

Este proyecto sigue la arquitectura hexagonal (también conocida como arquitectura de puertos y adaptadores) para la construcción de cada microservicio. Este enfoque promueve un diseño desacoplado, lo que garantiza que la lógica empresarial esté aislada de las dependencias externas, como bases de datos, API o marcos.

Principios clave:

Independencia de Frameworks: la lógica empresarial no depende de ningún framework específico.

Facilidad de Testeo: los casos de uso y la lógica central se pueden probar sin depender de una infraestructura externa.

Adaptabilidad: es sencillo cambiar las dependencias externas (por ejemplo, migrar de una base de datos SQL a una base de datos NoSQL).

Aplicación en el proyecto:

Capa de dominio: contiene la lógica empresarial básica y los casos de uso.

Capa de aplicación: coordina los casos de uso y actúa como intermediario entre la capa de dominio y las interfaces externas. Esta capa administra los servicios de la aplicación y organiza las operaciones.

Capa de infraestructura: contiene implementaciones concretas de dependencias externas, como bases de datos, API externas y adaptadores específicos. Esta capa interactúa directamente con bibliotecas y marcos externos.

Puertos: interfaces que definen cómo interactúan las capas externas con la lógica empresarial.

Adaptadores: implementaciones concretas de puertos, como controladores HTTP, repositorios de bases de datos o integraciones de API externas.

Esta arquitectura garantiza que los microservicios sean altamente mantenibles, escalables y fáciles de ampliar.

Mejores prácticas implementadas:

Uso de **Optional class**: para evitar valores nulos y manejar explícitamente la ausencia de datos.

Uso de **Result class**: para representar claramente el éxito o el fracaso de las operaciones, mejorando la legibilidad y el manejo de errores.

Estas prácticas garantizan un código más sólido, legible y fácil de mantener.

Link del repositorio del proyecto

Arquitectura

La Aplicación está fundamentada en una Arquitectura de microservicios, la cual cuenta con 3 servicios los cuales son:

Auth Service: Se encarga de la lógica de manejo de usuarios de todo tipo, al igual que la autenticación de los mismos.

Product Service: Se encarga de la lógica de manejo de productos y el inventario de los mismos.

Orders Service: Se encarga de la lógica de manejo de órdenes, manejo de carrito de compras y de los reportes a partir de las órdenes.

Cada servicio fue construido bajo la Arquitectura Hexagonal, haciendo uso del lenguaje Python y el Framework FastAPI. Cabe destacar que cada servicio cuenta con su propia Base de Datos (PostgreSQL fue el MBDR usado en los 3 servicios).

Estos servicios interactúan asíncronamente entre ellos haciendo uso de eventos y colas de mensajes, al igual que conexión http en casos donde la comunicación debía ser síncrona.

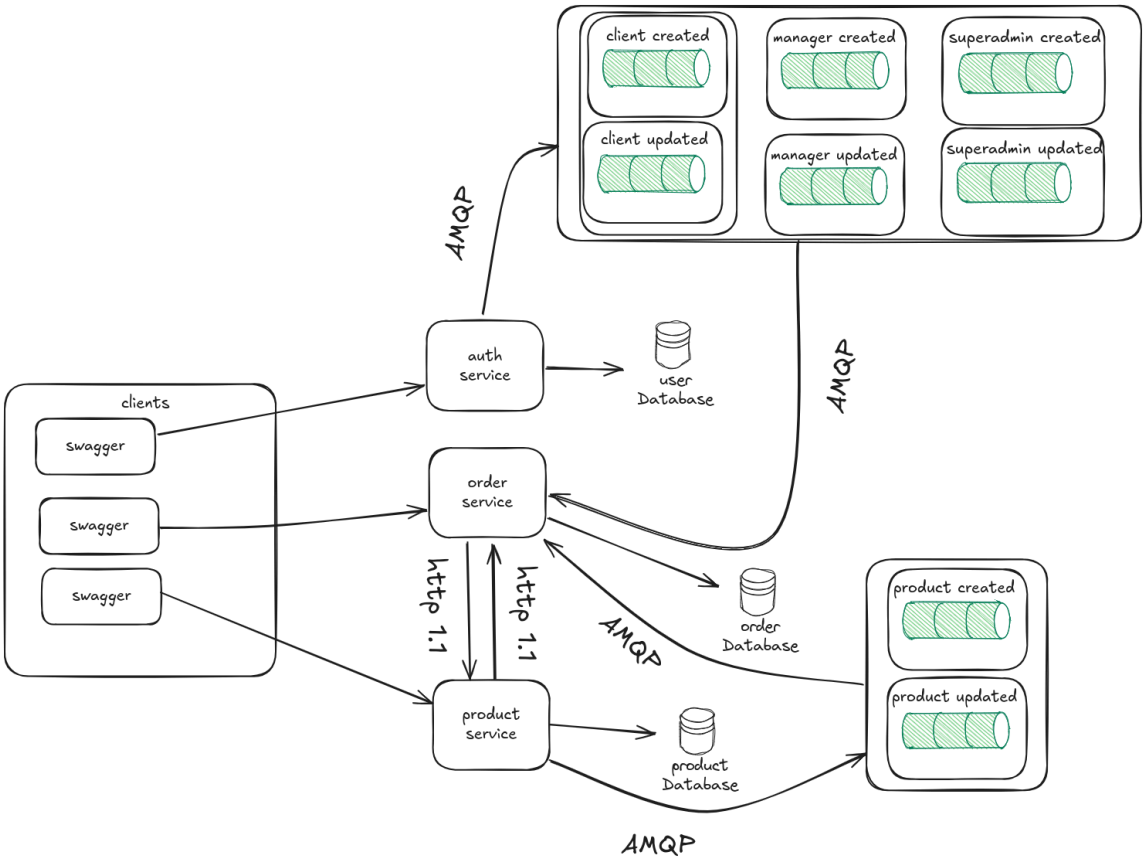
Los servicios de Auth y Product emiten eventos hacia la cola de mensaje, de la cual Orders consume dichos eventos y registra los cambios y/o acciones necesarias.

El servicio de Órdenes a través de HTTP se comunica con el servicio de Product para coordinar la disponibilidad y reintegración de inventario.

La comunicación asíncrona está apoyada por RabbitMQ el cual hace uso del Protocolo AMQP para la transmisión de mensajes.

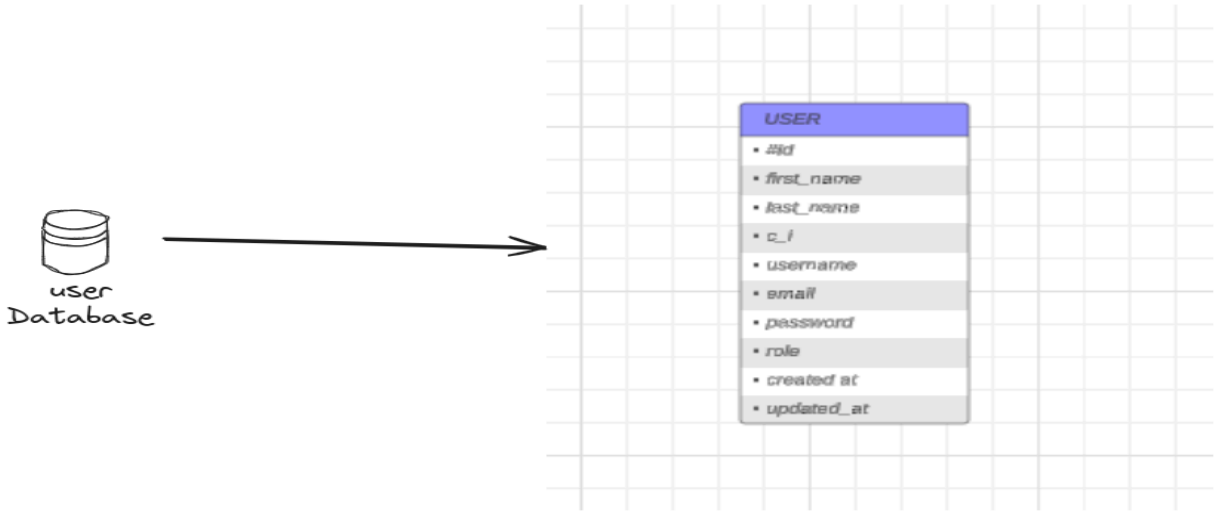
Diagramas

1. Diagrama de la aplicación

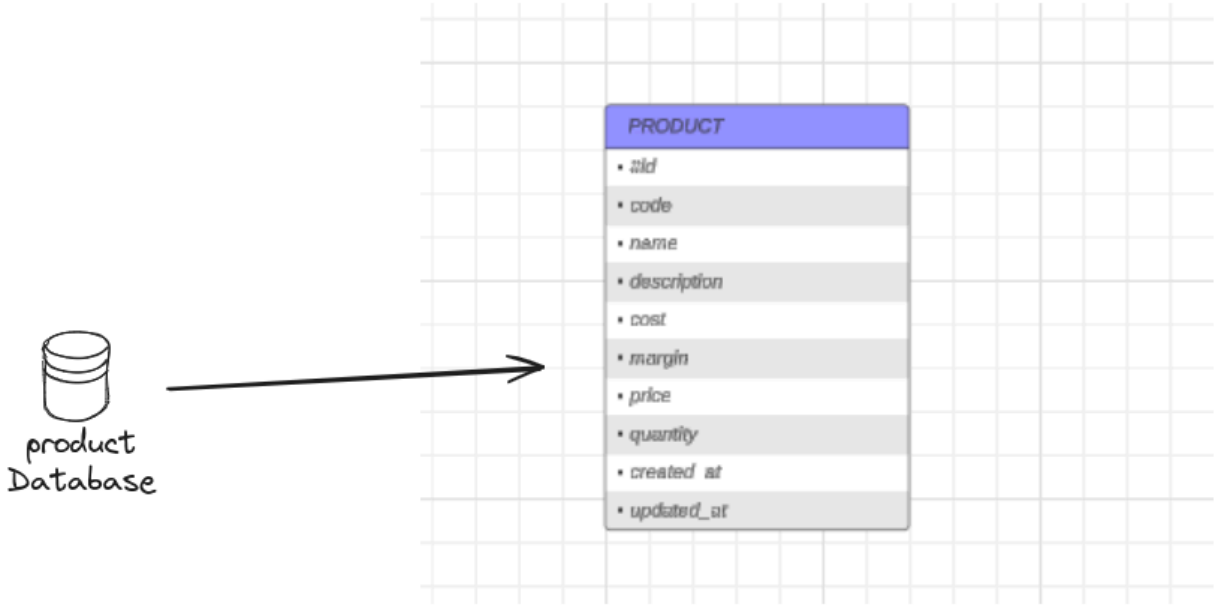


2.

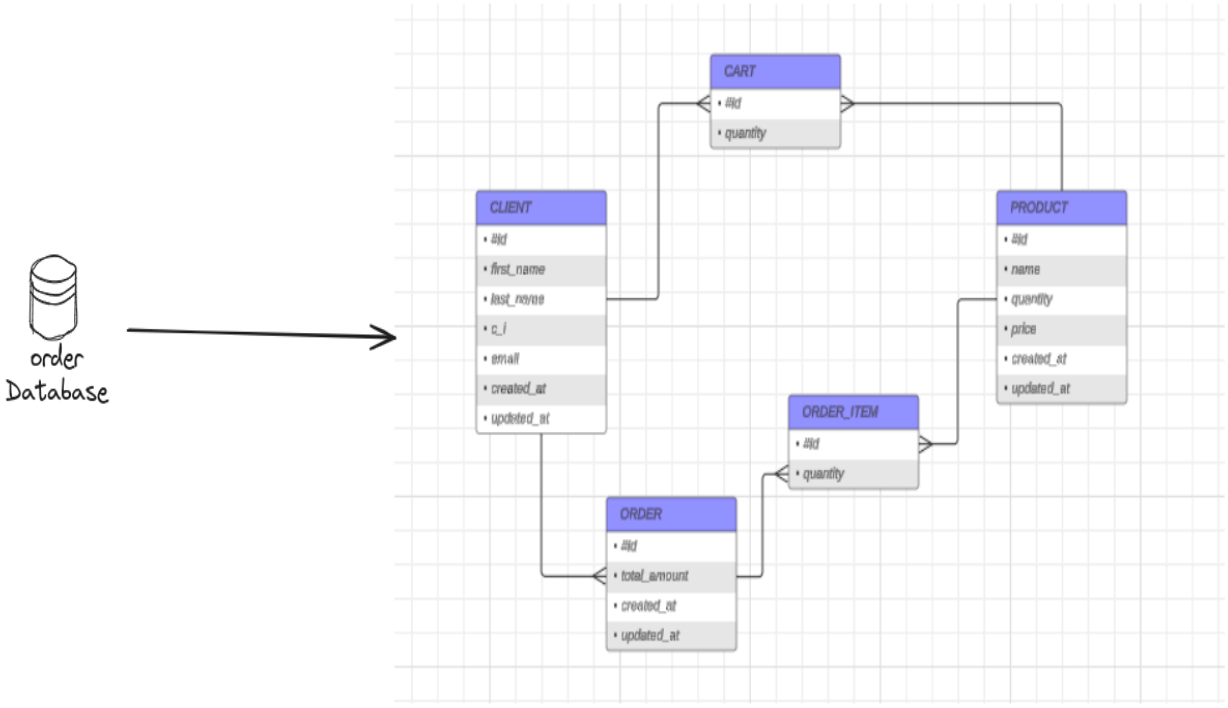
3. Diagrama ER de Auth



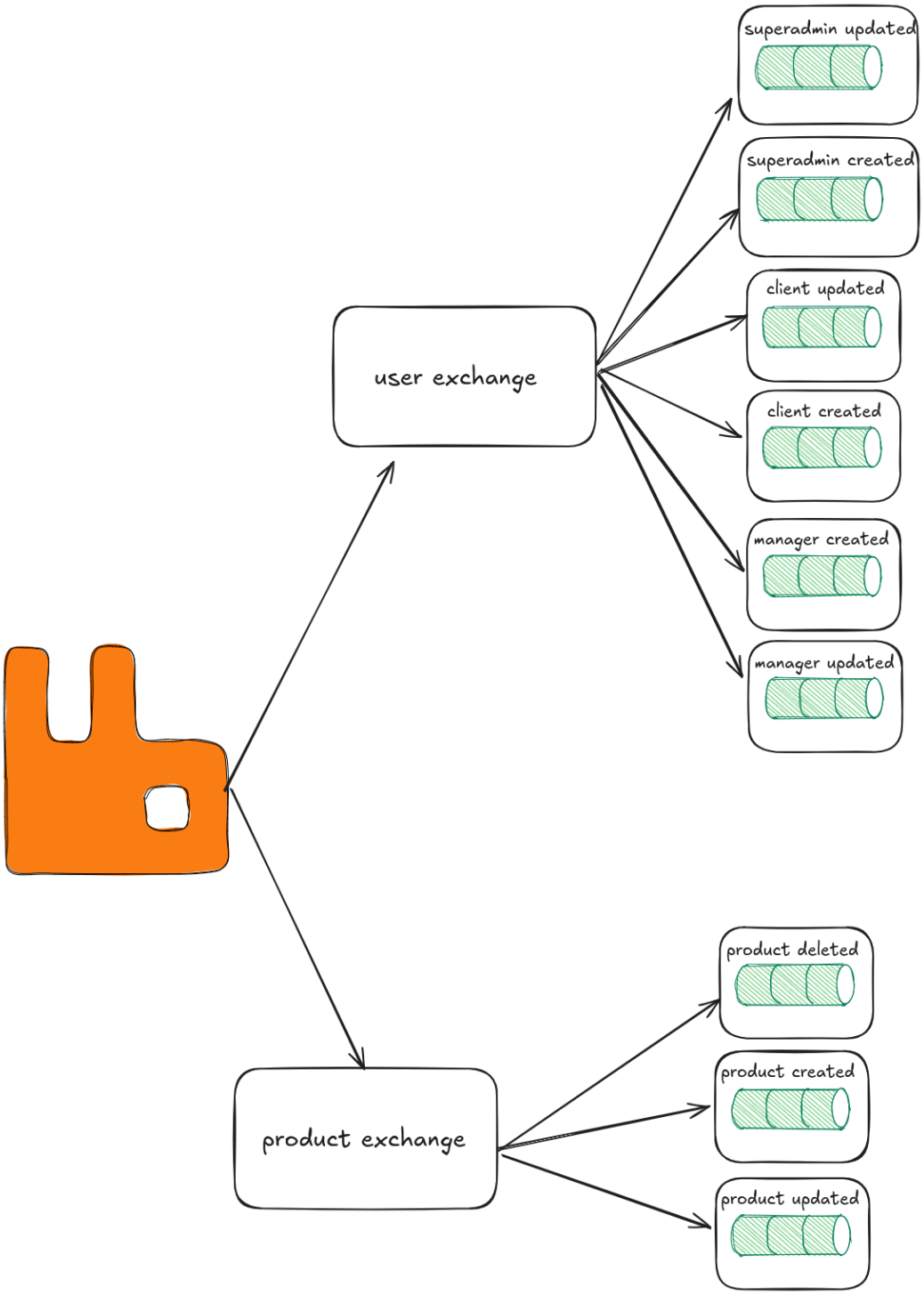
4. Diagrama ER de Product



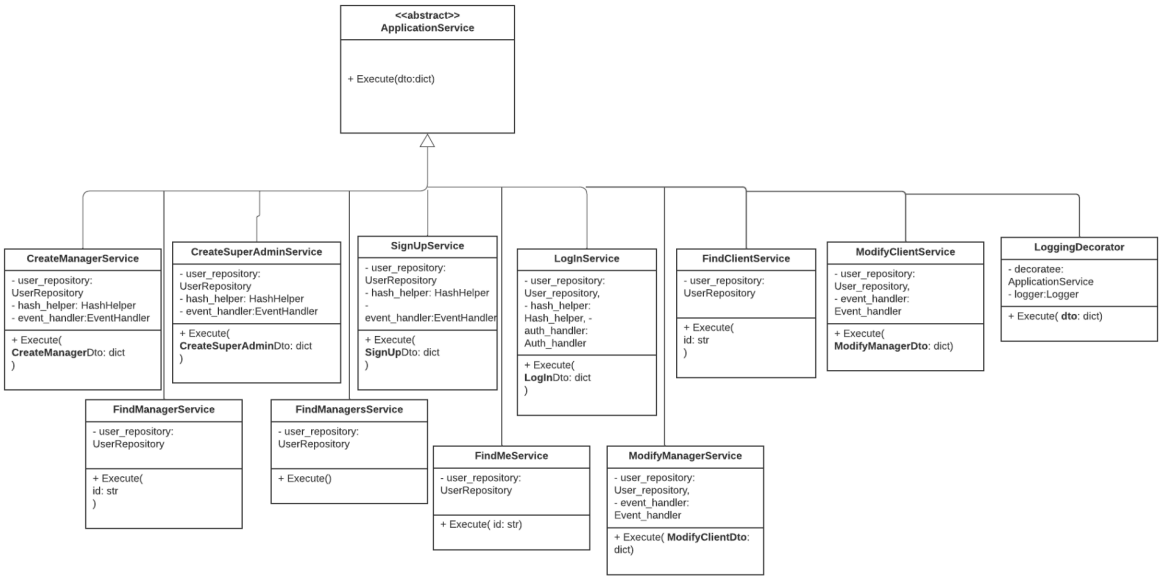
5. Diagrama ER de Orders

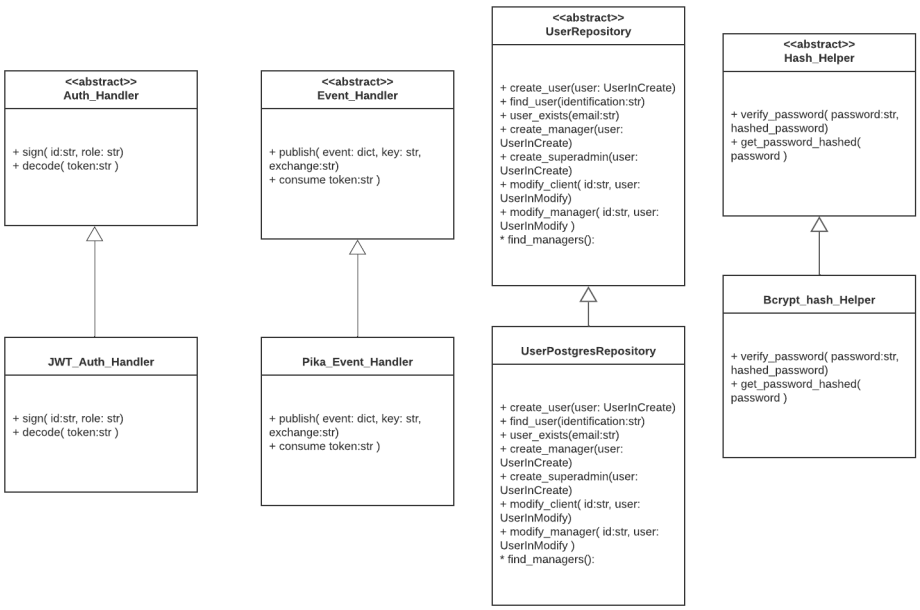


6. Diagrama de Eventos con Rabbitmq



7. Diagrama de clases Auth service





Instalación

Para correr la aplicación haciendo uso de docker:

1- Clonarse la rama development en repositorio

2- Crear un archivo .env a partir del .env example que se encuentra en la raíz del proyecto y rellenar las variables de entorno tomando en cuenta que direcciones para comunicación entre contenedores deben referirse hacia ellos con su nombre respectivo en el Docker Compose como host .

3- Utilizar “docker-compose up --build ” para construir y arrancar cada uno de los contenedores.

Impresiones

Este proyecto basado en microservicios (Aunque no era el objetivo del proyecto) fue una experiencia de aprendizaje sumamente completa y enriquecedora. Nos permitió profundizar en la arquitectura hexagonal y las mejores prácticas de desarrollo moderno haciendo uso de un lenguaje tan intuitivo como lo es Python y un framework tan cómodo para trabajar como lo es FastAPI .

Sin duda, la parte más desafiante fue la comunicación entre los servicios. Coordinar la interacción asíncrona a través de RabbitMQ y la comunicación síncrona vía HTTP requirió un entendimiento profundo de ambos mecanismos y una cuidadosa planificación.

La dockerización también presentó un reto significativo debido a la cantidad de contenedores involucrados y la necesidad de orquestar sus dependencias. Lograr que los servicios de Auth, Product y Orders funcionaran en armonía dentro del

entorno Dockerizado fue una tarea compleja pero con mucho aprendizaje.

En resumen, el proyecto fue un gran reto completado, nos permitió adquirir habilidades valiosas en el diseño y desarrollo de sistemas escalables y mantenibles. La implementación de la arquitectura de microservicios, junto con la arquitectura hexagonal y las mejores prácticas adoptadas, resultó en una base de código bastante atractiva para cada uno de nuestros portafolios personales.