UNIVERSIDAD CENTRAL DEL ECUADOR

FACULTAD DE INGENIERIAS Y CIENCIAS APLICADAS

INGENIERIA EN SISTEMAS DE INFORMACIÓN

Arquitectura de Software

Tema: Manual de Implementación

Integrantes

Emily Guerrón

Paralelo: SI7-001

Fecha: 18-02-2026

QUITO – ECUADOR

Índice

1. Introduccion

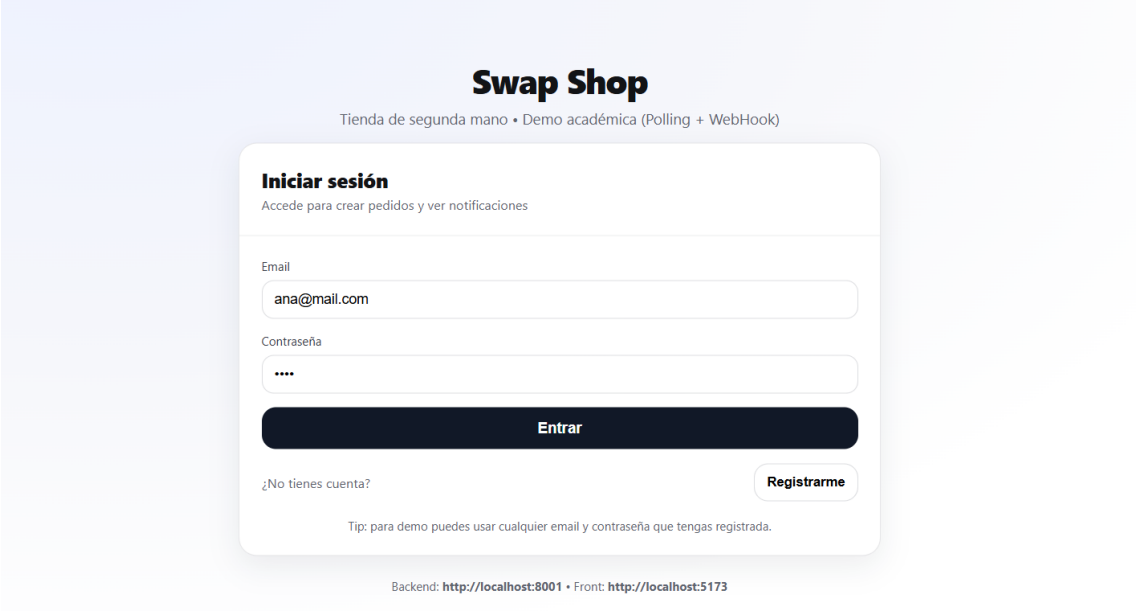
El sistema Swap Shop es una aplicación web desarrollada bajo una arquitectura de microservicios, diseñada para simular la gestión de pedidos en una tienda de segunda mano. La solución integra tecnologías modernas como FastAPI, MySQL, Redis, Docker y React, permitiendo demostrar conceptos de persistencia, caché, comunicación entre servicios y pruebas de carga.

El sistema implementa un flujo completo de negocio que incluye autenticación, creación de pedidos, almacenamiento en base de datos, procesamiento automático mediante polling y envío de notificaciones a través de WebHooks.

Se priorizó una arquitectura desacoplada y escalable, donde cada servicio cumple una función específica, favoreciendo mantenibilidad y separación de responsabilidades.

Además, se incorporó Redis como mecanismo de optimización para reducir la carga sobre la base de datos en consultas repetitivas.

Finalmente, se realizaron pruebas de carga con k6 para validar el comportamiento del sistema bajo múltiples usuarios concurrentes.



2. Objetivos

2.1 Objetivo General

Diseñar e implementar un sistema distribuido basado en microservicios que gestione pedidos de una tienda virtual, integrando persistencia relacional, caché en memoria y comunicación interservicios mediante polling y WebHooks.

2.2 Objetivos Específicos

1. Implementar una API REST en FastAPI para manejar autenticación y pedidos.
2. Diseñar una base de datos relacional en MySQL para almacenar información estructurada.
3. Integrar Redis como sistema de caché para optimizar consultas frecuentes.
4. Implementar un servicio de polling que procese pedidos pendientes automáticamente.
5. Desarrollar un frontend interactivo en React para interacción del usuario.

Cada objetivo responde a una capa específica de la arquitectura, garantizando modularidad y claridad estructural.

3. Alcance del Sistema

El sistema permite a los usuarios registrarse e iniciar sesión mediante un mecanismo de autenticación basado en tokens. Una vez autenticado, el usuario puede crear pedidos compuestos por múltiples items, los cuales son almacenados en la base de datos.

El sistema permite consultar pedidos pendientes, visualizar detalles específicos por ID y observar notificaciones generadas tras el procesamiento automático de pedidos.

No se incluyen funcionalidades avanzadas como pasarela de pagos real, sistema de inventario ni panel administrativo completo.

El alcance está orientado principalmente a demostrar integración tecnológica y comunicación entre microservicios.

El proyecto tiene fines académicos y de demostración técnica

4. Arquitectura del Sistema

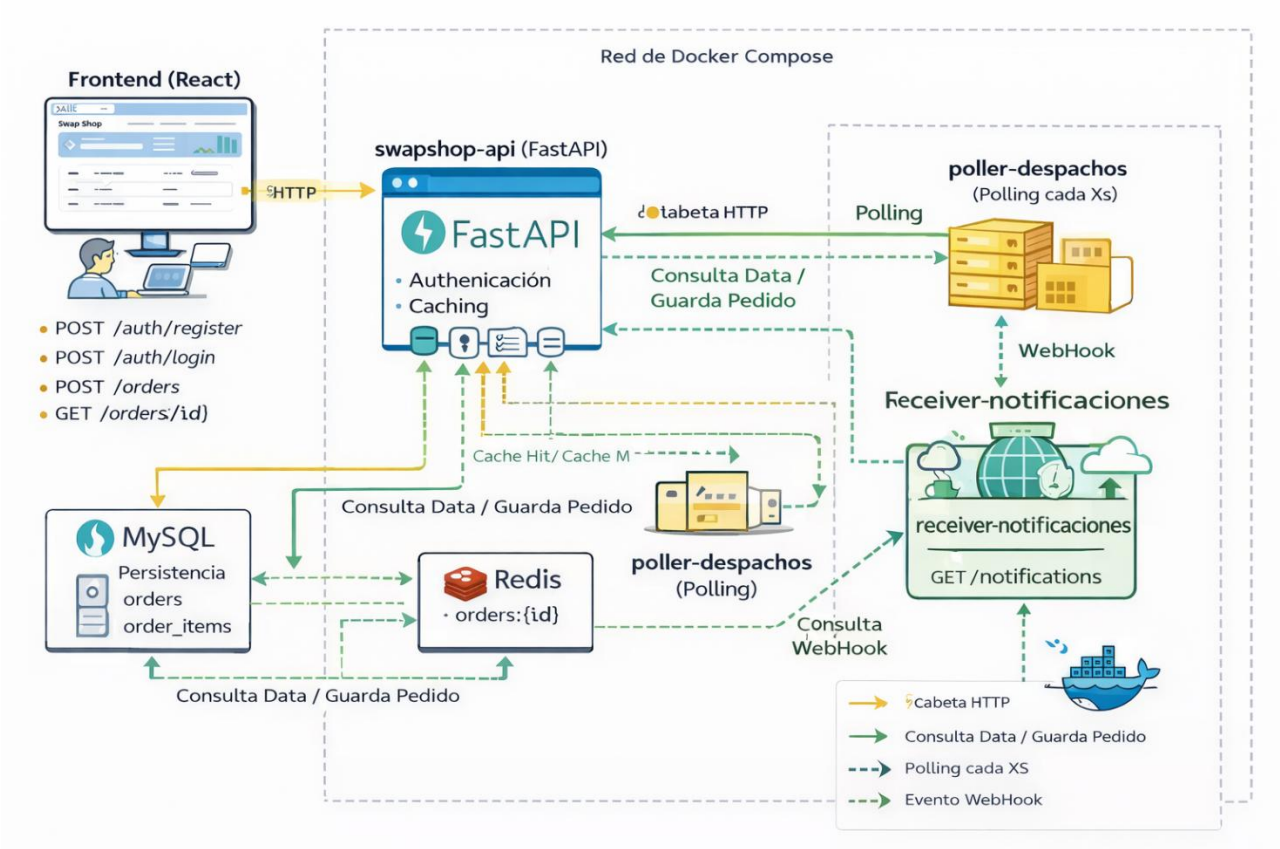
El sistema está compuesto por múltiples servicios independientes que se comunican entre sí dentro de una red Docker.

La API principal (swapshop-api) gestiona autenticación y lógica de negocio. MySQL se encarga del almacenamiento persistente, mientras que Redis funciona como capa de caché para acelerar consultas específicas.

El servicio poller-despachos consulta periódicamente pedidos pendientes y ejecuta procesamiento automático. El receiver-notificaciones recibe eventos mediante WebHook.

El frontend desarrollado en React actúa como cliente, permitiendo interacción del usuario con los servicios backend.

Esta arquitectura favorece escalabilidad horizontal y desacoplamiento entre componentes.



5. Flujo de Funcionamiento

5.1 Flujo de Autenticación

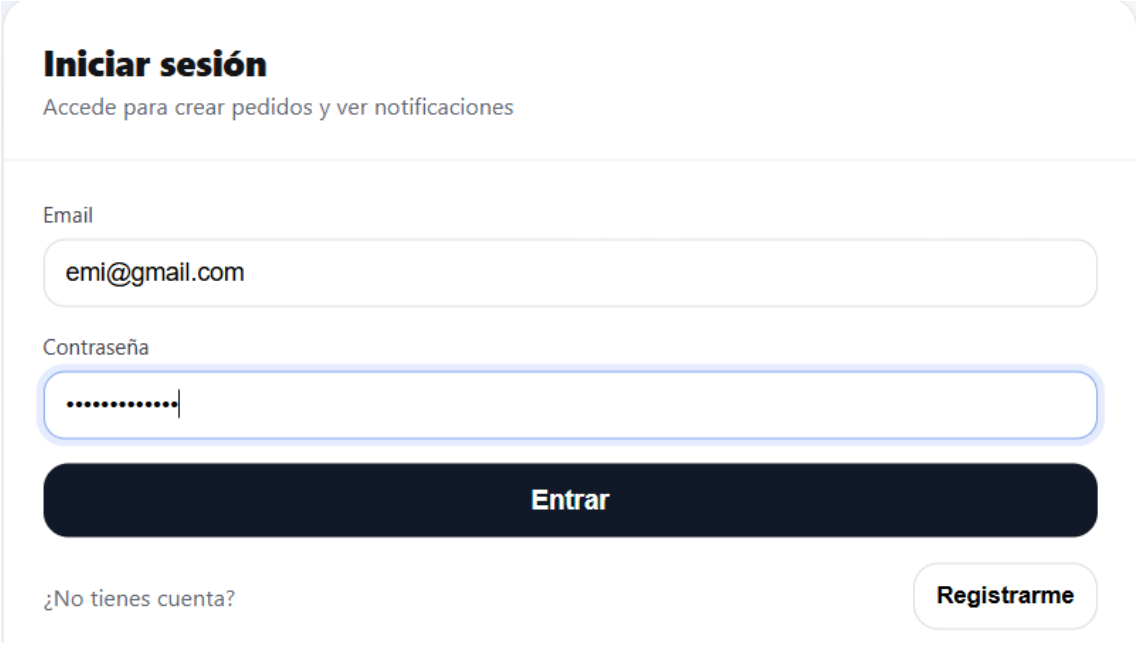
El usuario accede al sistema mediante un formulario de registro o inicio de sesión. Las credenciales son enviadas a la API, la cual valida los datos y genera un token de autenticación.

Este token es almacenado en el navegador y se utiliza en cada solicitud protegida mediante el header Authorization.

La autenticación permite restringir el acceso a endpoints sensibles como la creación de pedidos.

Este mecanismo garantiza seguridad básica y separación entre usuarios autenticados y no autenticados.

El uso de tokens permite una arquitectura stateless en el backend.



5.2 Flujo de Creación de Pedido

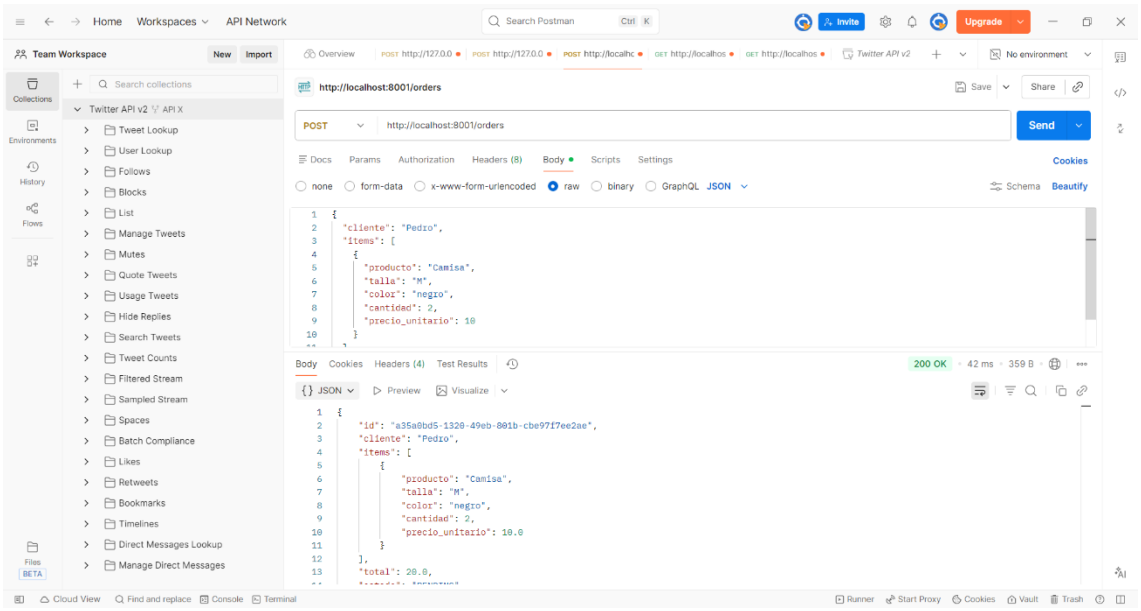
Cuando el usuario crea un pedido, el frontend envía la información estructurada hacia la API.

La API calcula el total automáticamente, genera un identificador único (UUID) y almacena los datos en MySQL.

Posteriormente, guarda una copia del pedido en Redis para optimizar futuras consultas.

El sistema devuelve el pedido creado al frontend, confirmando la operación.

Este flujo demuestra la interacción coordinada entre persistencia relacional y caché en memoria.



5.3 Flujo de Consulta con Caché (Redis)

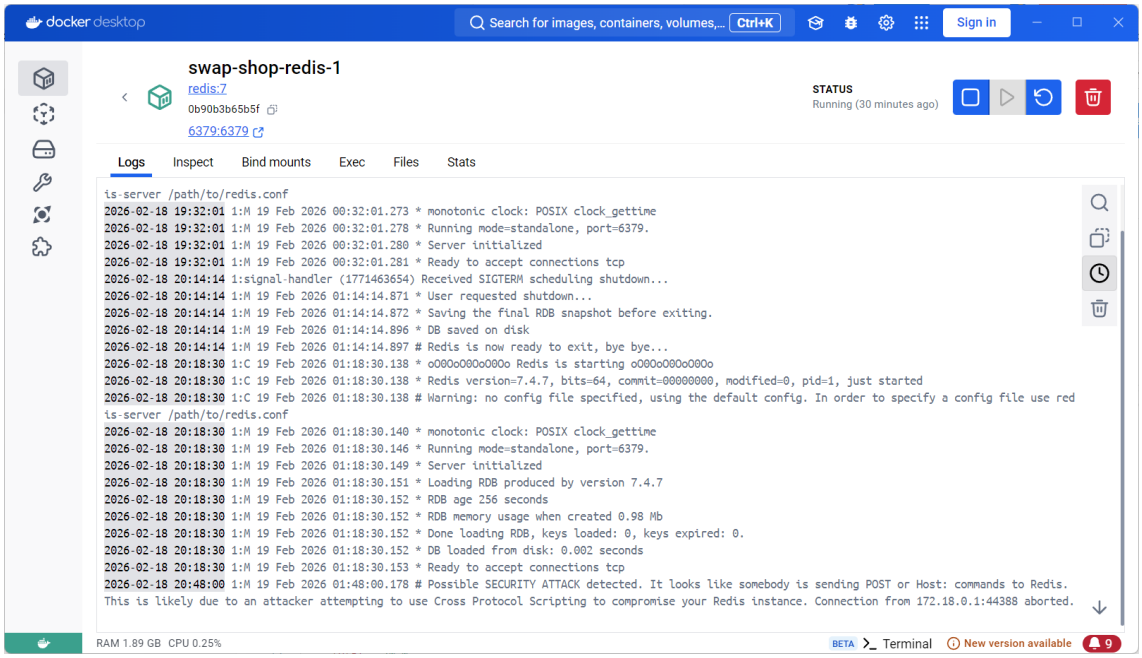
Cuando se consulta un pedido por ID, el sistema primero verifica si existe en Redis.

Si el dato está en caché (CACHE HIT), la respuesta es inmediata y no se consulta MySQL.

Si no existe (CACHE MISS), se consulta la base de datos y luego se almacena el resultado en Redis.

Este mecanismo reduce carga en la base de datos y mejora rendimiento bajo alta concurrencia.

Redis funciona como acelerador de lectura para operaciones frecuentes.



5.4 Flujo Polling + WebHook

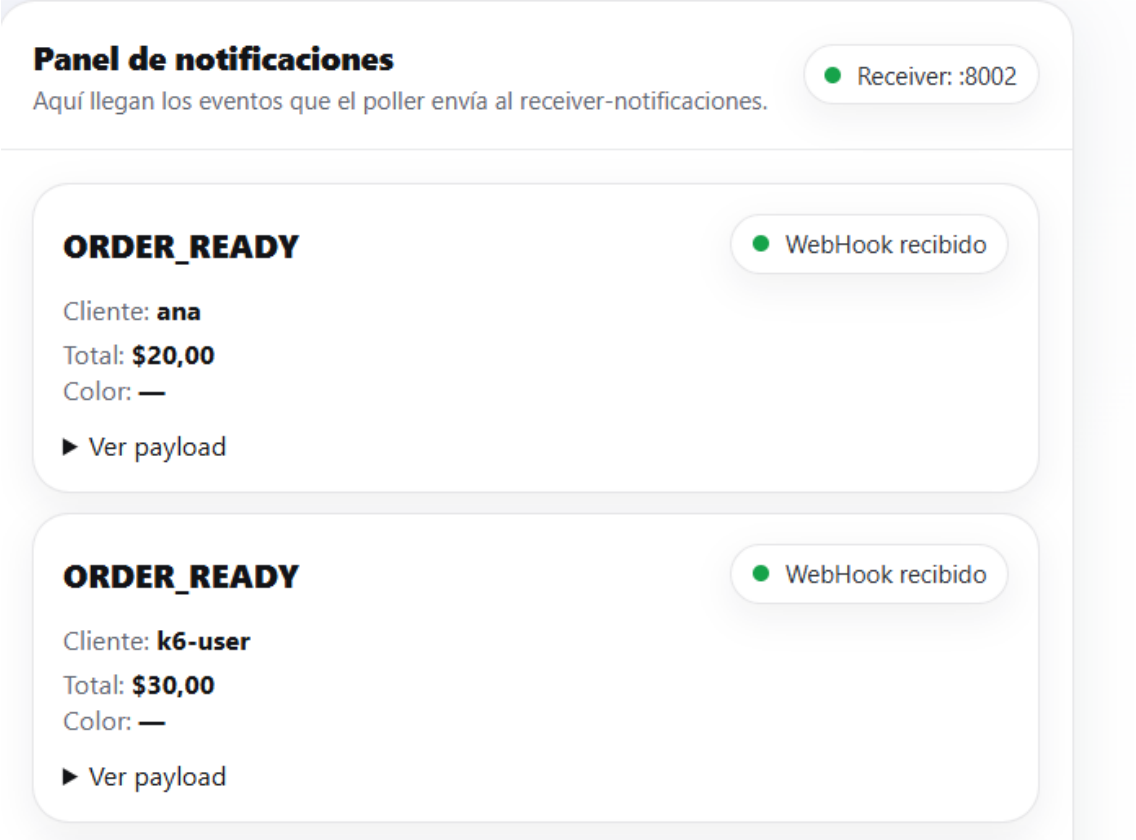
El servicio poller ejecuta consultas periódicas a /orders/pending.

Cuando detecta pedidos pendientes, puede marcarlos como procesados.

Luego envía un evento mediante WebHook al servicio receiver-notificaciones.

El receiver almacena la notificación y la expone al frontend.

Este mecanismo simula comunicación asíncrona entre microservicios.



6. Base de Datos

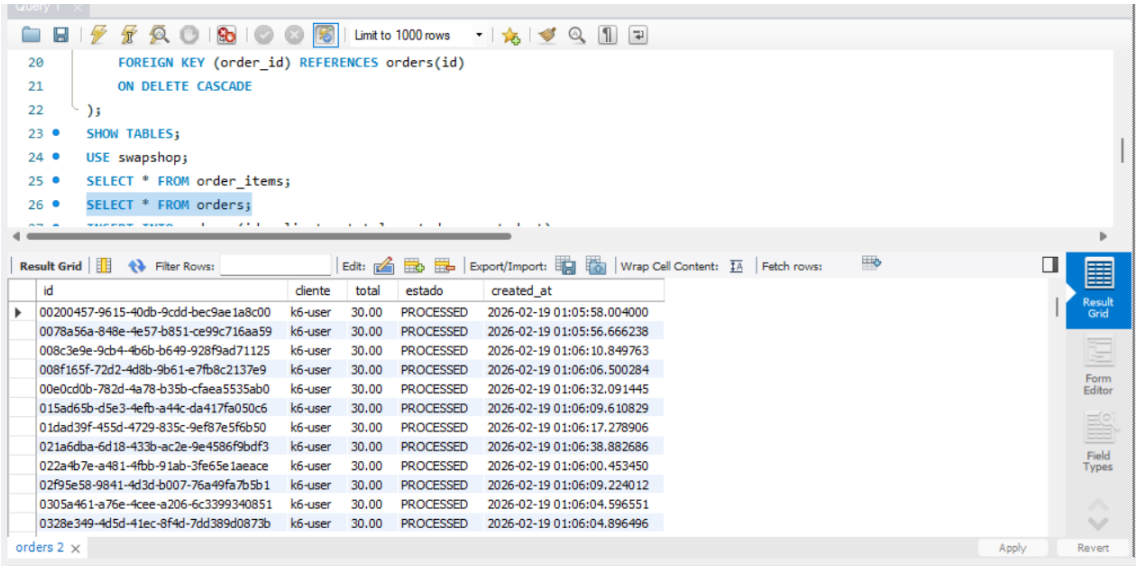
La base de datos MySQL almacena información estructurada en dos tablas principales: orders y order_items.

La tabla orders contiene la cabecera del pedido, incluyendo estado y fecha de creación.

La tabla order_items almacena el detalle de productos asociados a cada pedido mediante una clave foránea.

Este diseño relacional garantiza integridad referencial y normalización básica.

MySQL proporciona persistencia confiable y transacciones seguras.



7. Pruebas de Carga (k6)

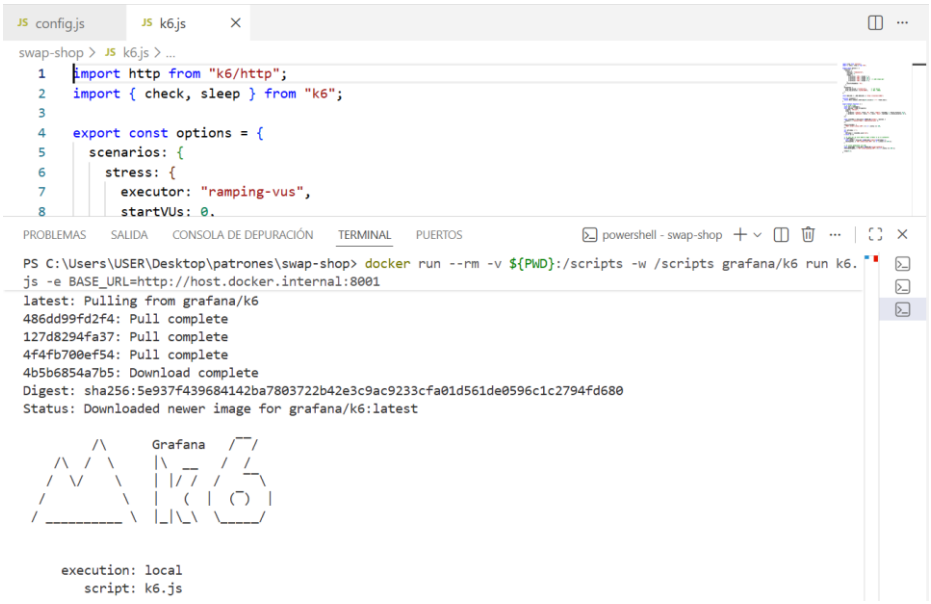
Se utilizó la herramienta k6 para simular múltiples usuarios concurrentes realizando peticiones al sistema.

La prueba permitió evaluar tiempos de respuesta, estabilidad de la API y comportamiento bajo estrés.

Se verificó que Redis mejora significativamente el rendimiento en consultas repetidas.

Las pruebas demostraron que la arquitectura soporta múltiples conexiones simultáneas sin colapsar.

El uso de pruebas de carga es fundamental para validar escalabilidad.



ORDER_READY

WebHook recibido

Cliente: **k6-user**

Total: **$30,00**

Color: —

▶ Ver payload

ORDER_READY

WebHook recibido

Cliente: **k6-user**

Total: **$30,00**

Color: —

▶ Ver payload

ORDER_READY

WebHook recibido

Cliente: **k6-user**

Total: **$30,00**

Color: —

▶ Ver payload

8. Problemas Encontrados y Soluciones

- Durante el desarrollo surgieron errores relacionados con resolución de servicios en Docker, autenticación MySQL y validación de respuestas.
- Se solucionaron ajustando redes Docker, agregando dependencias necesarias y corrigiendo modelos de respuesta en FastAPI.
- También se presentó un problema de pantalla en blanco en el frontend, resuelto mediante control de errores y validación de sesión.
- Estos incidentes permitieron fortalecer la estabilidad general del sistema.
- Cada error representó una oportunidad de aprendizaje técnico.

9. Repositorio Git

https://github.com/Emilianah/PROYECTO_FINAL.git

10. Conclusiones

- El sistema cumple con los objetivos planteados, integrando múltiples tecnologías modernas en una arquitectura funcional.
- La combinación de MySQL y Redis demuestra cómo se puede equilibrar persistencia y rendimiento.
- El uso de microservicios permite modularidad y fácil mantenimiento.
- Las pruebas de carga validaron estabilidad bajo condiciones exigentes.
- El proyecto evidencia dominio de backend, frontend y despliegue con contenedores.

11. Recomendaciones

- Se recomienda implementar persistencia de notificaciones en base de datos para evitar pérdida de información al reiniciar servicios.
- Agregar monitoreo con herramientas como Prometheus o Grafana mejoraría la observabilidad.
- Implementar expiración de tokens y refresh tokens reforzaría la seguridad.
- Optimizar manejo de errores en frontend evitaría bloqueos de interfaz.
- Escalar horizontalmente servicios críticos permitiría mayor rendimiento en producción.