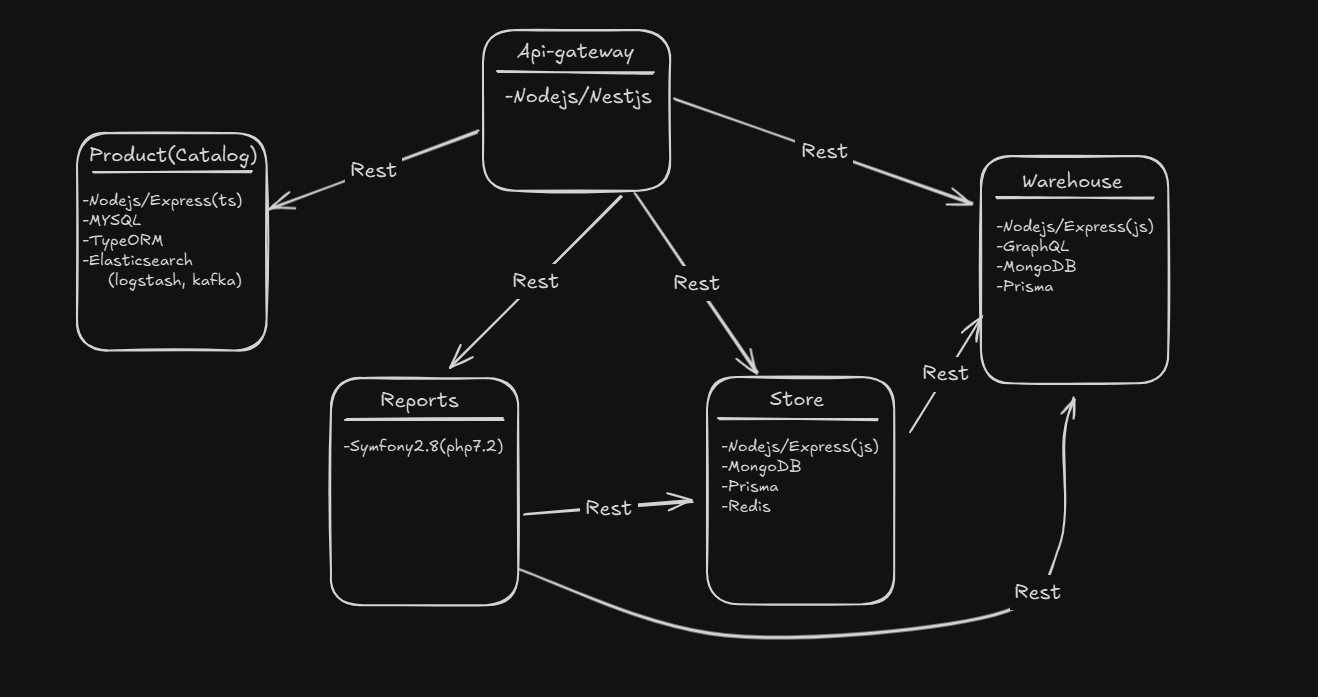
1. Cómo resolvió la prueba.

Para abordar la prueba decidí implementar los cuatro microservicios solicitados y, adicionalmente, desarrollar un API Gateway. Este gateway actúa como punto de entrada único al sistema, centralizando las solicitudes externas y distribuyéndolas hacia el microservicio correspondiente según la funcionalidad requerida. Con esta estrategia se logra una arquitectura más organizada, escalable y fácil de mantener, garantizando además una experiencia de consumo más clara y consistente para los clientes.



Decisiones mencionables por microservicios:

Microservicio Catálogo:

- En el punto referente al endpoint para listar productos con filtros por categoría y nombre, opté por utilizar **Elasticsearch** como motor de búsqueda. Esta decisión se debe a que la búsqueda avanzada era el único requerimiento en esta parte, y Elasticsearch ofrece un rendimiento óptimo y flexible para este tipo de consultas, garantizando resultados rápidos y escalables.

Microservicio Almacén:

* Decidí separar completamente el concepto de **almacén y tienda** del de **producto**, almacenando únicamente el productSku como referencia. Además, no implementé validaciones sobre los movimientos, ya que no estaban especificadas en los requisitos. Esta separación incrementa la independencia entre microservicios, permitiendo que, incluso si el servicio de productos presenta algún problema, sea posible registrar movimientos en el almacén y posteriormente crear el producto correspondiente. De esta forma, se favorece la **desacoplación** y la **resiliencia** de la arquitectura.

Microservicio Tienda:

* En este caso, decidí que al realizar un pedido la respuesta inicial fuera “orden en proceso”. Esto se logra gracias a que el pedido se envía primero a una cola en Redis, lo que permite manejarlo de forma asíncrona. Posteriormente, un verificador revisa la disponibilidad de los productos:

1. Si los productos existen y cuentan con la cantidad requerida o superior, la orden se genera automáticamente como confirmada.
2. En caso contrario, se registra de manera automática como pendiente.

De esta forma, se garantiza un flujo más eficiente y desacoplado, optimizando el rendimiento y la experiencia del cliente.

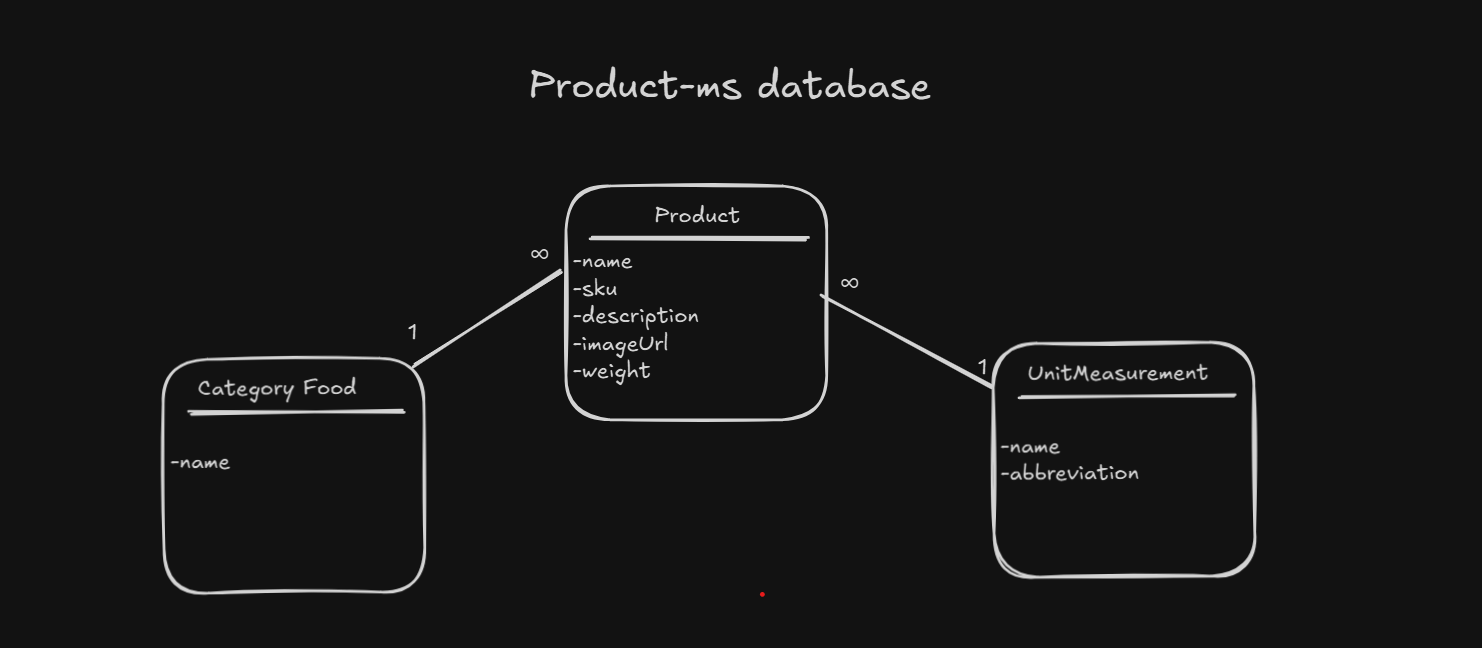
Microservicio de Reportes:

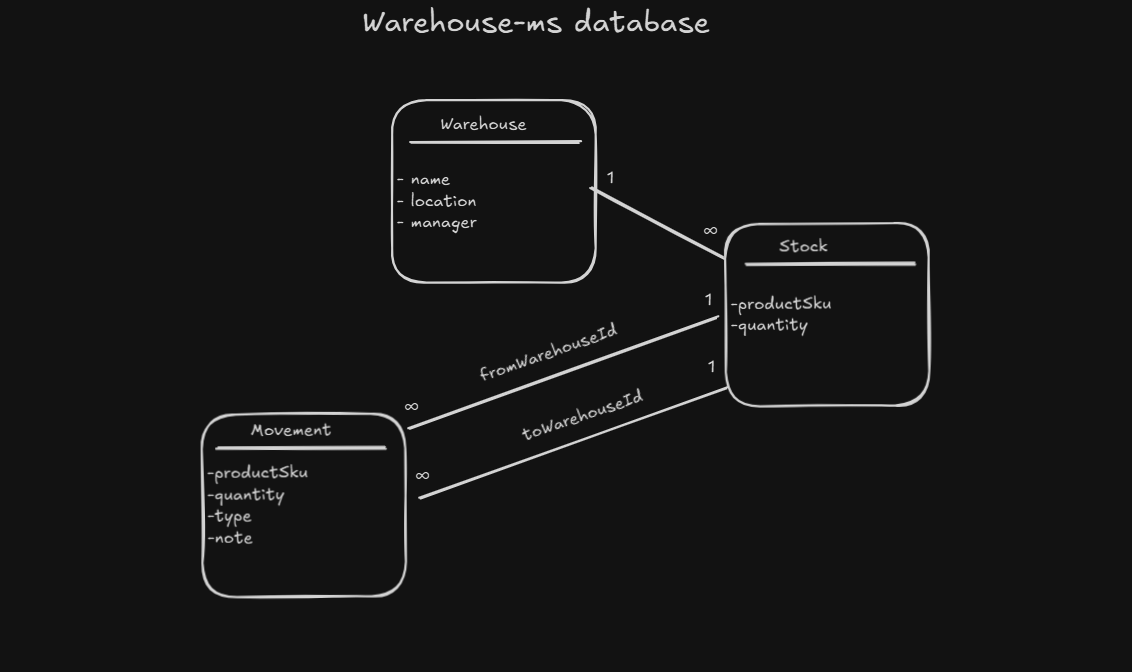
* Punto: Stock total en almacén y tienda  
  Para resolver este requerimiento decidí obtener la información consolidada a partir de dos fuentes:

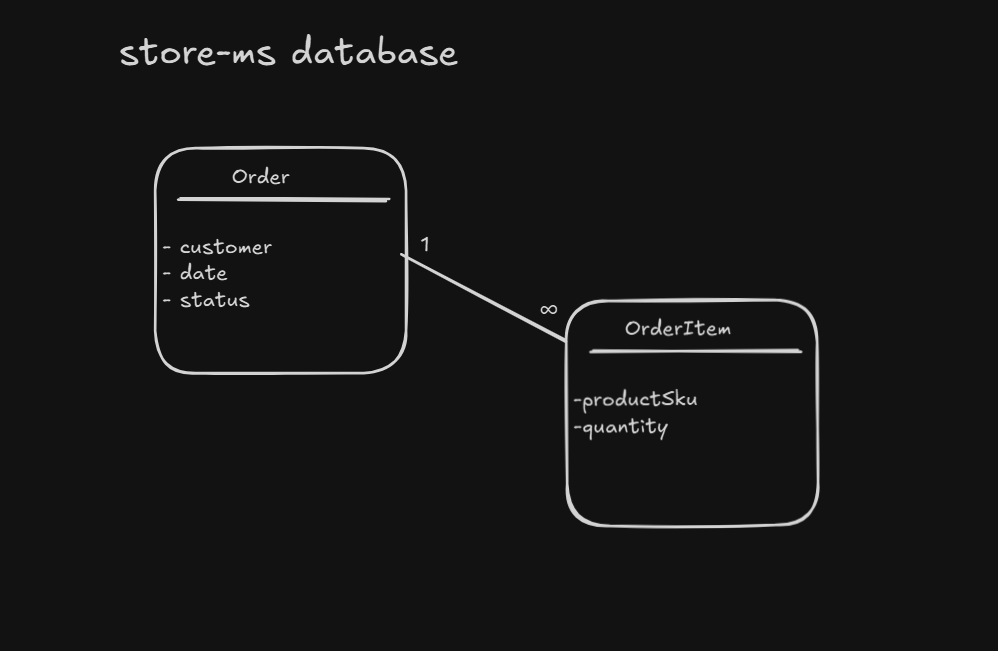
1. Almacén: recuperando las cantidades registradas por producto y por almacén.
2. Pedidos: consultando las órdenes existentes para identificar las unidades comprometidas.

De esta forma, por cada producto se devuelve tanto la cantidad disponible en almacén como la cantidad en pedidos, ofreciendo una visión completa del stock total y facilitando la gestión de inventario

1. Diseño de base de datos.







1. Herramientas de IA que utilizó durante el desarrollo

Durante el desarrollo del proyecto integré herramientas de inteligencia artificial para optimizar el flujo de trabajo y mejorar la calidad del resultado. En particular:

1. Utilicé GitHub Copilot como asistente de desarrollo para recibir sugerencias de código en tiempo real, lo que permitió agilizar la escritura de funciones y estructuras repetitivas.
2. Hice uso de ChatGPT para resolver dudas conceptuales y técnicas, obtener ejemplos de código a partir de prompts específicos y comprender mejor la interpretación de ciertos puntos del enunciado de la prueba.
3. Gracias a estas herramientas fue posible encontrar soluciones a errores más rápidamente, validar enfoques de implementación y generar explicaciones claras que facilitaron la toma de decisiones.
4. Además, empleé ChatGPT para refinar y mejorar la redacción de los textos técnicos aquí presentados, logrando una documentación más clara, estructurada y profesional.
5. Retos encontrados y cómo los resolvió.
6. Uso de GraphQL

* Reto: No tenía experiencia previa y al inicio me costó entender la diferencia con REST.
* Solución: Revisé documentación oficial, usé Copilot para sugerencias rápidas y ChatGPT para profundizar dudas. Finalmente logré montar resolvers y schema correctamente.
* Aprendizaje: GraphQL me terminó gustando por la flexibilidad en las consultas y la reducción de overfetching.

1. Error con Docker (max depth exceeded)

* Reto: El error se debía a acumulación de volúmenes e imágenes viejas que estaban causando inconsistencias.
* Solución: Aunque al principio no quería limpiar volúmenes, terminé eliminándolos (docker system prune -a --volumes) y reconstruyendo los contenedores desde cero.
* Aprendizaje: Mantener limpio el entorno Docker ayuda a evitar problemas difíciles de diagnosticar.