**Prueba Técnica Backend**

# Objetivo

Crear una solución compuesta por dos microservicios independientes que interactúen entre sí, utilizando JSON API como estándar para la comunicación.

# Requisitos Técnicos

* Usa el lenguaje de la vacante aplicada.
* Implementa JSON API [(https://jsonapi.org/](https://jsonapi.org/)) para todas las respuestas.
* Usa Docker para containerizar los servicios.
* Elige entre base de datos SQL, NoSQL o en memoria (justifica tu elección).

**Microservicio 1: Productos**

* Gestionará un recurso llamado productos con los campos: id, nombre y precio.
* El microservicio permitirá:
  1. Crear un nuevo producto.

○ Obtener un producto específico por ID.

○ Actualizar un producto por ID.

○ Eliminar un producto por ID.

○ Listar todos los productos con paginación simple.

**Microservicio 2: Inventario**

* Gestionará un recurso llamado inventarios con los campos: producto\_id y cantidad.
* Permitirá:
  1. Consultar la cantidad disponible de un producto específico por ID (obtiene la información del producto llamando al microservicio de productos).

○ Actualizar la cantidad disponible de un producto específico tras una compra.

○ Emitir un evento simple cuando el inventario cambia (implementación básica puede ser un mensaje en consola).

# Comunicación

* La comunicación entre microservicios debe ser mediante peticiones HTTP usando JSON API.
* Implementa un mecanismo básico de autenticación entre servicios (API keys). ● Implementa manejo de timeout y reintentos básicos.

**Documentación**

* Documenta los endpoints con Swagger/OpenAPI.

# Requisitos de Testing

* Implementa pruebas unitarias que cubran:
  1. Creación y actualización de productos.

○ Comunicación entre microservicios.

○ Manejo de errores (producto no encontrado, errores de comunicación).

* Incluye al menos una prueba de integración por microservicio.

**Expectativas según Seniority.**

* **Junior:** 
  1. Implementación básica funcional de ambos microservicios.

○ Docker básico para cada servicio.

○ Pruebas unitarias básicas (mínimo 40% cobertura).

* **Mid-level:** 
  1. Implementación estructurada siguiendo buenas prácticas.

○ Docker Compose para orquestar ambos servicios.

○ Manejo adecuado de errores y logs básicos.

○ Pruebas unitarias con buena cobertura (mínimo 60%).

○ **Documentación básica de API.**

○ Incluye un diagrama simple de la arquitectura e interacción entre servicios.

* **Senior:** 
  1. **Solución robusta con arquitectura clara.**

○ Implementación completa de Docker con configuración optimizada.

○ Sistema de logs estructurados y health checks.

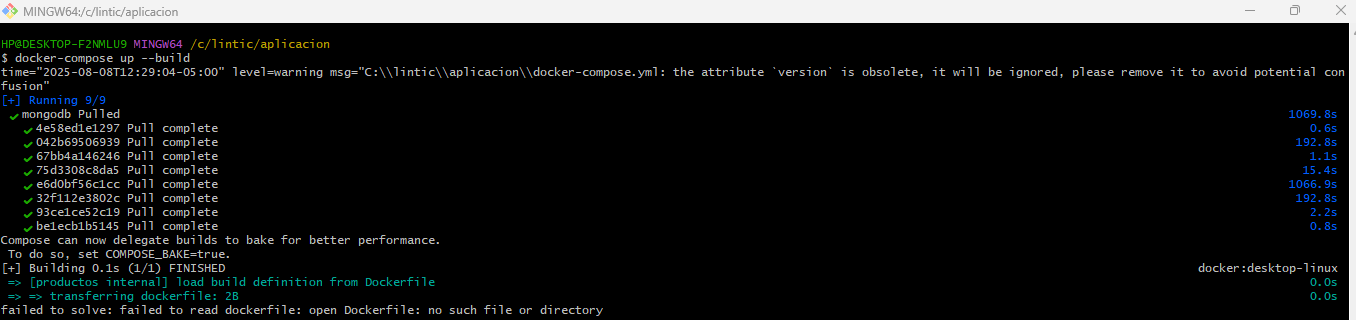
**○ Alta cobertura en pruebas (mínimo 80%).**

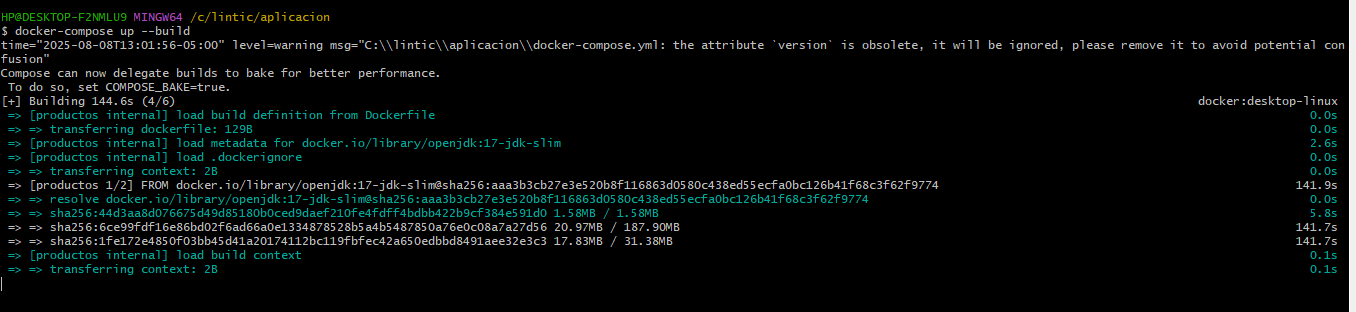
○ Implementación de autenticación entre servicios.

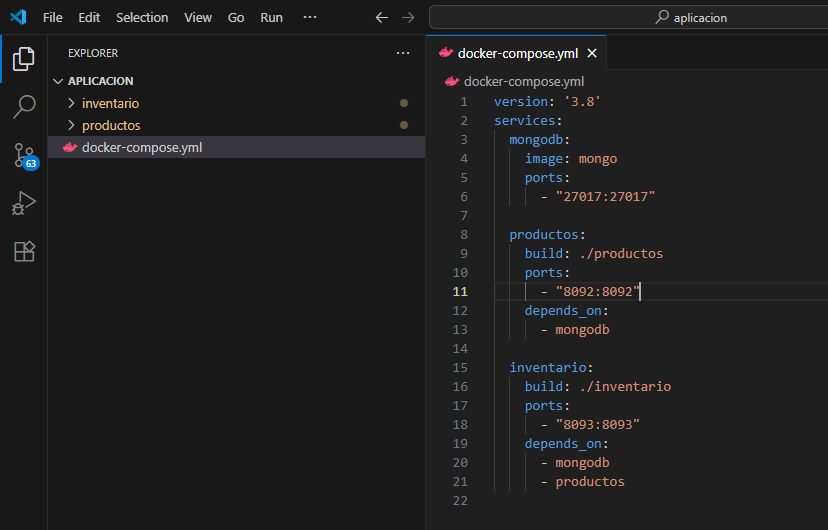
○ **Documentación completa y diagrama de arquitectura.**

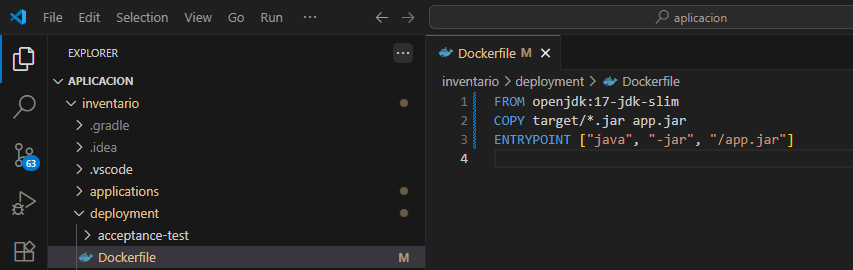
* **Líder Técnico:**

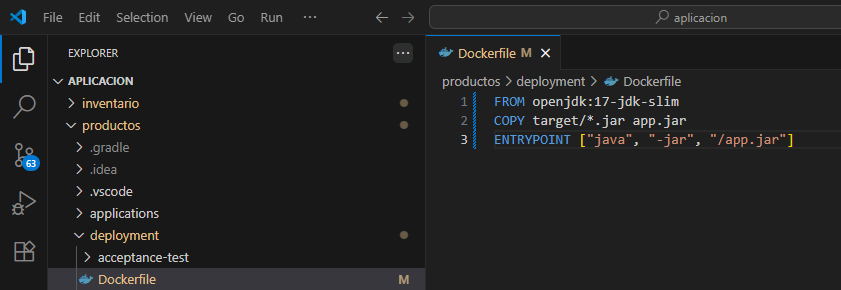
**○ Docker Compose para orquestar ambos servicios**

****

****

****

****

****

**Microservicio 1: Productos**

* Gestionará un recurso llamado productos con los campos: id, nombre y precio.
* El microservicio permitirá:
  1. Crear un nuevo producto.

○ Obtener un producto específico por ID.

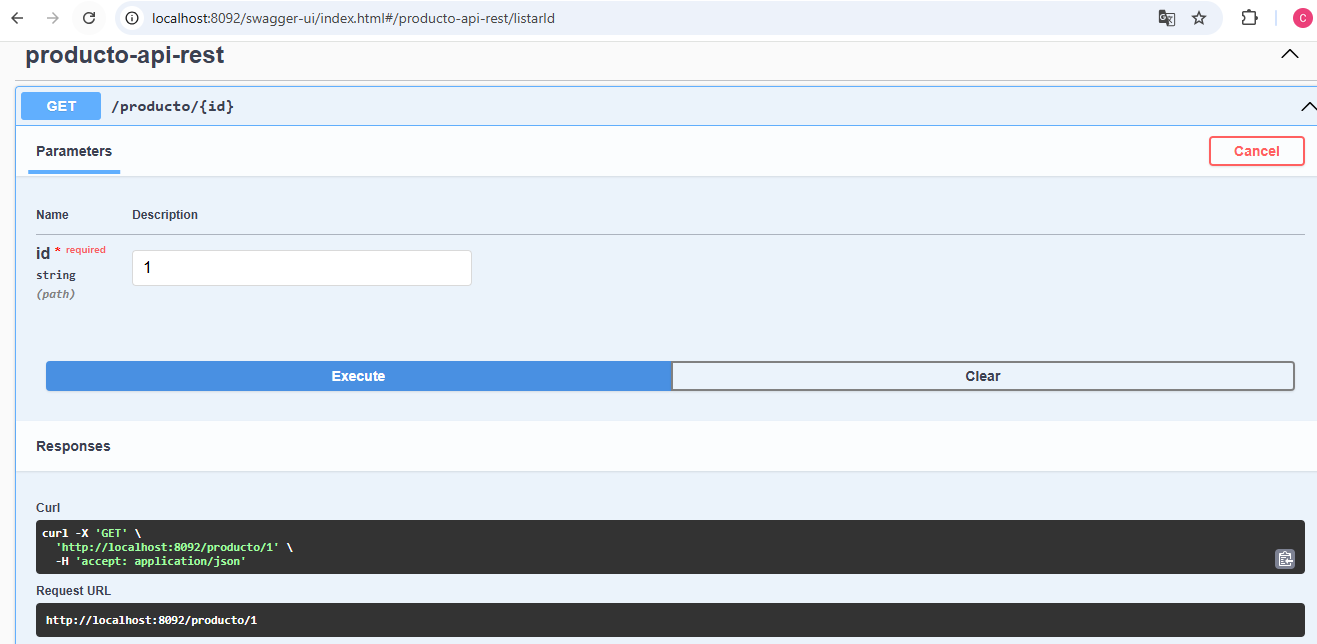
○ Actualizar un producto por ID.

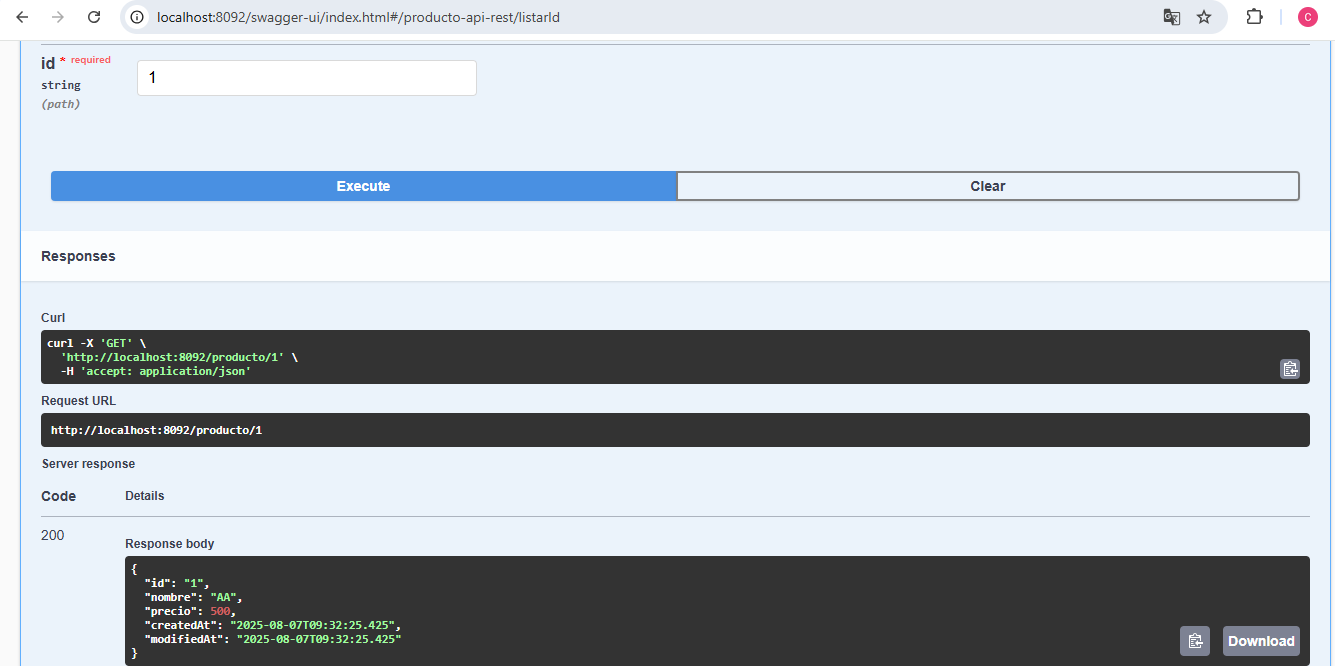
○ Eliminar un producto por ID.

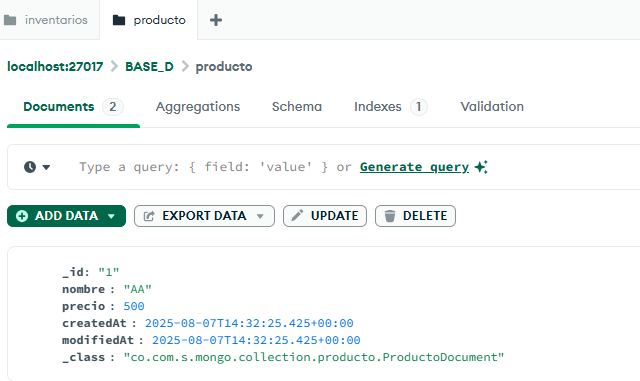
○ Listar todos los productos con paginación simple.

**Documentación API Swagger**

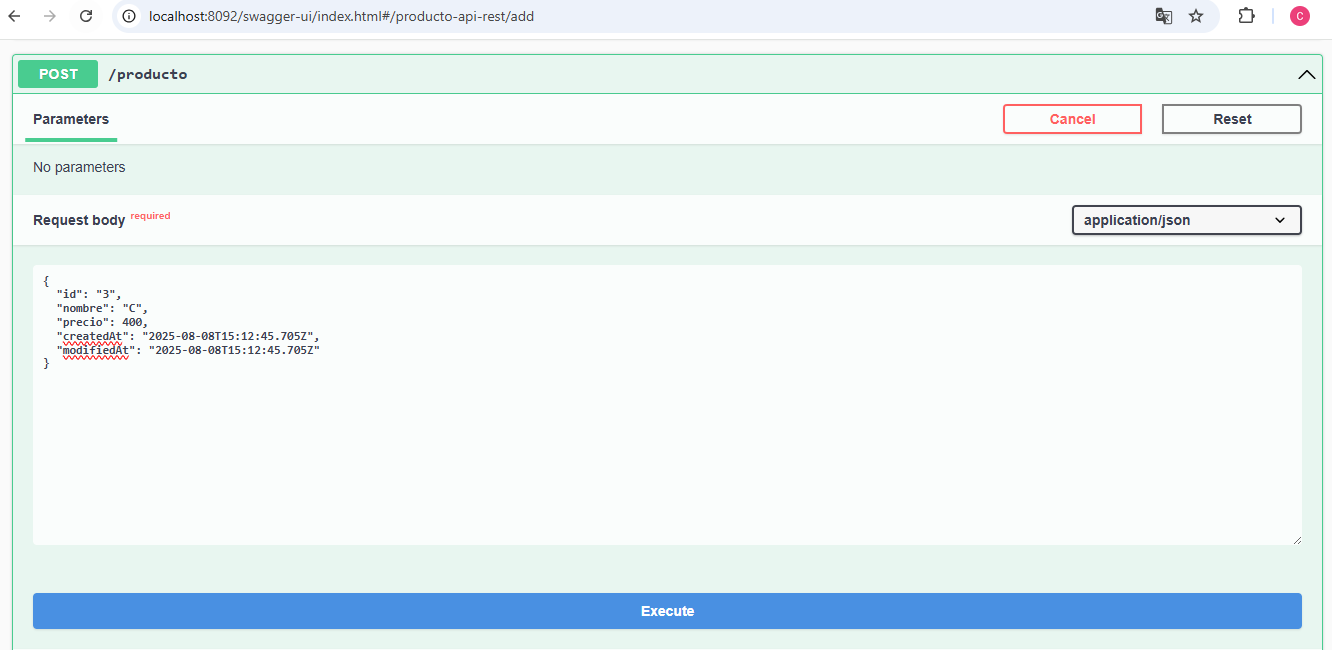
Obtener un producto específico por ID

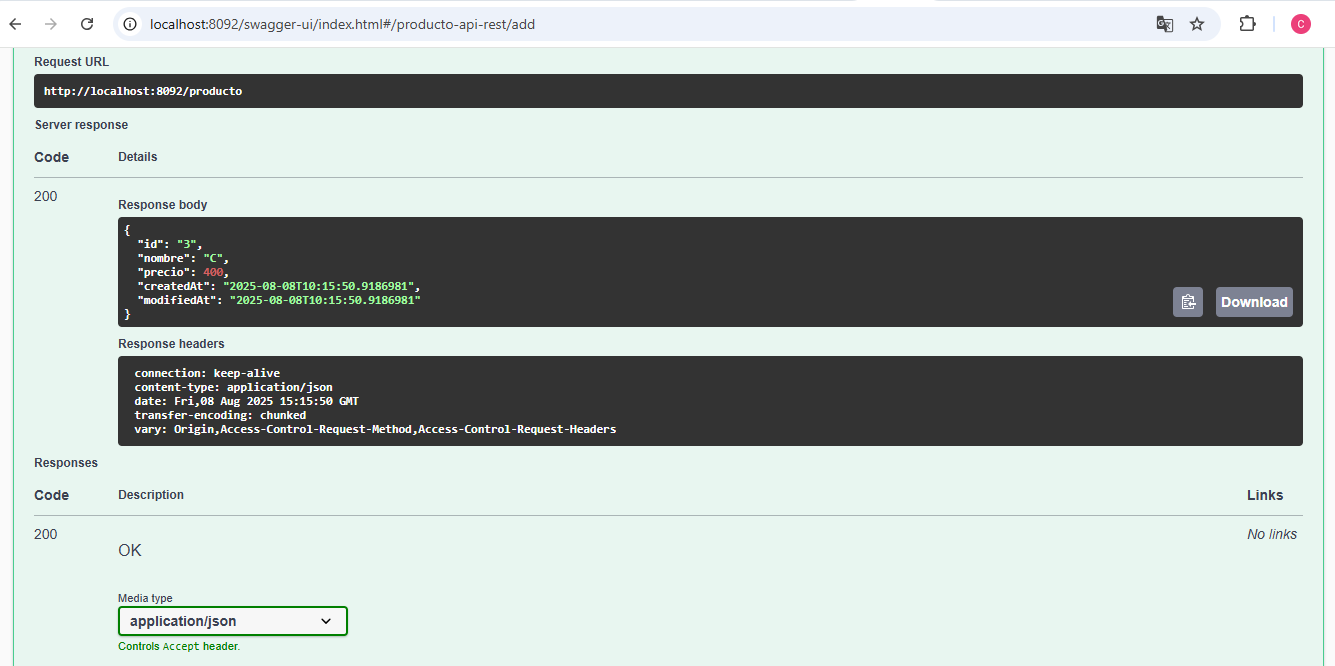






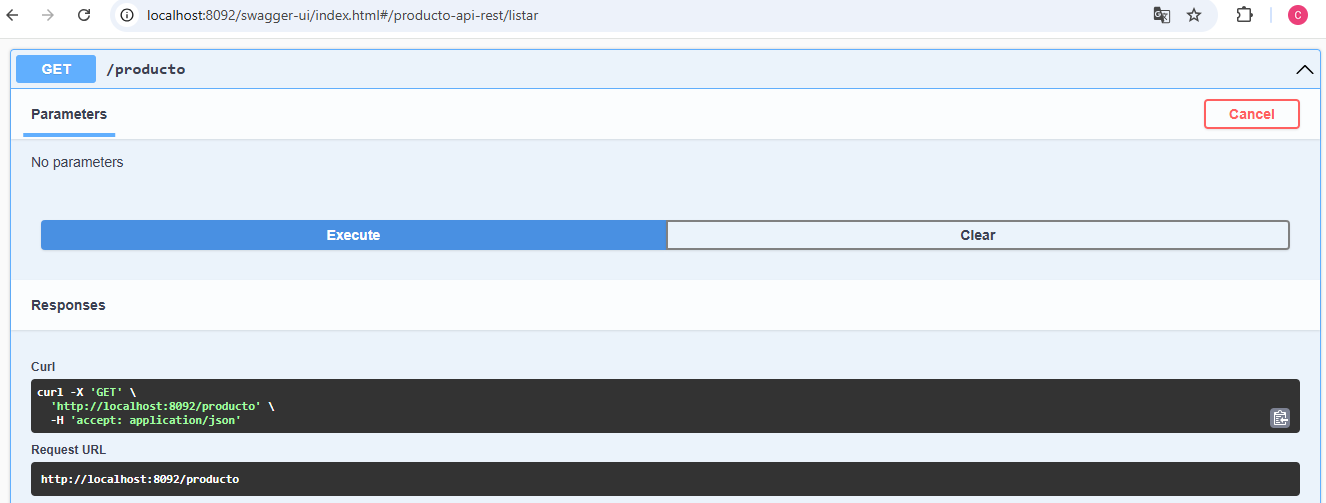
Crear un nuevo producto

****

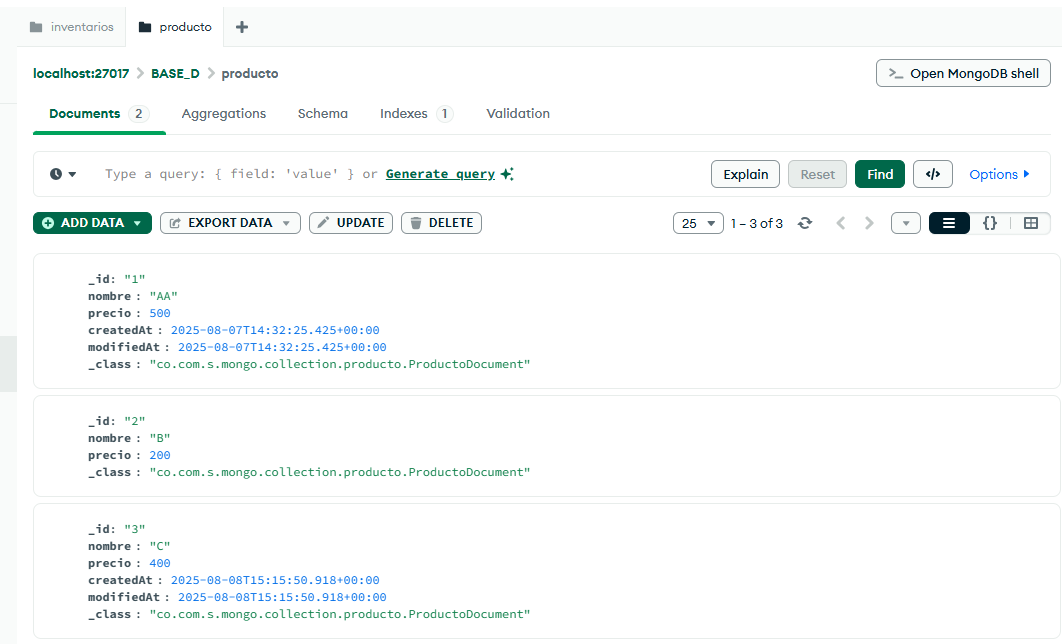
****

****

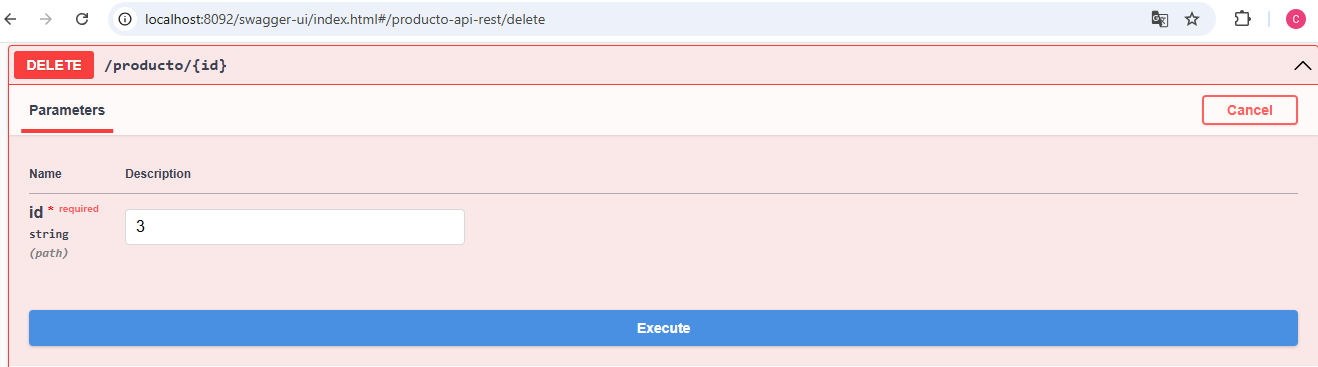
○ Listar todos los productos

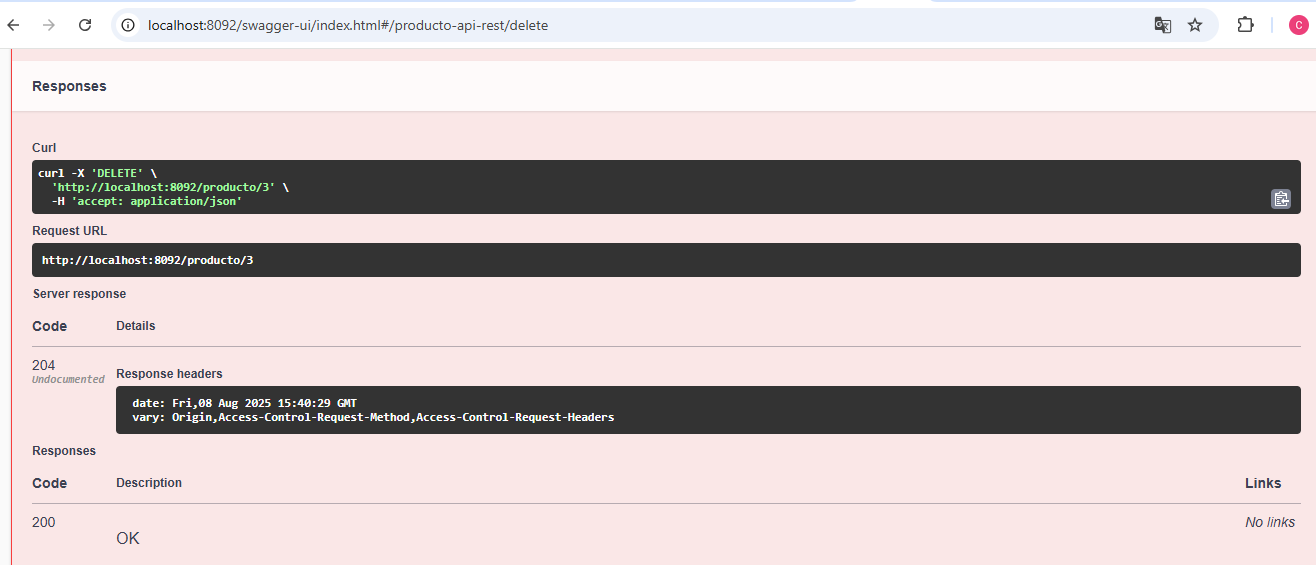
****

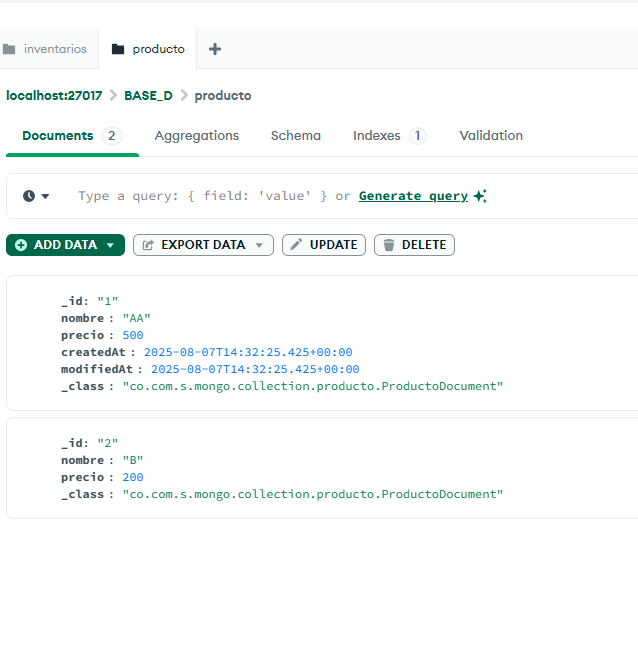
****

****

Eliminar un producto por ID

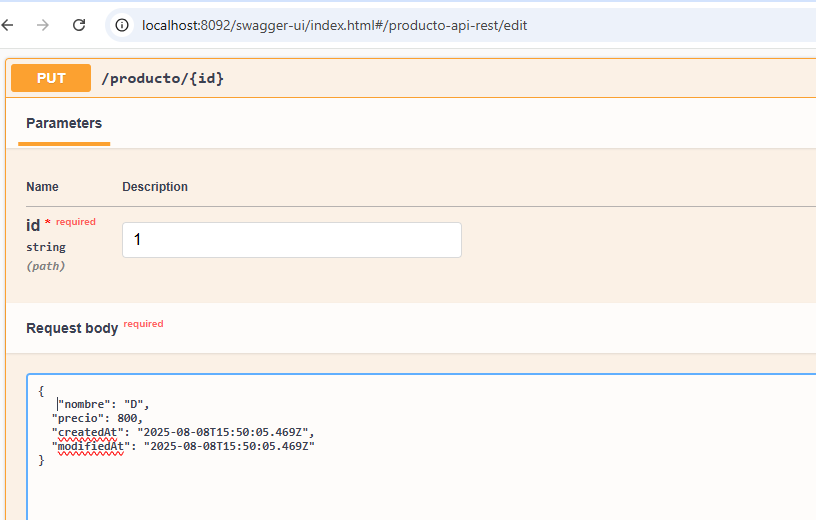
****

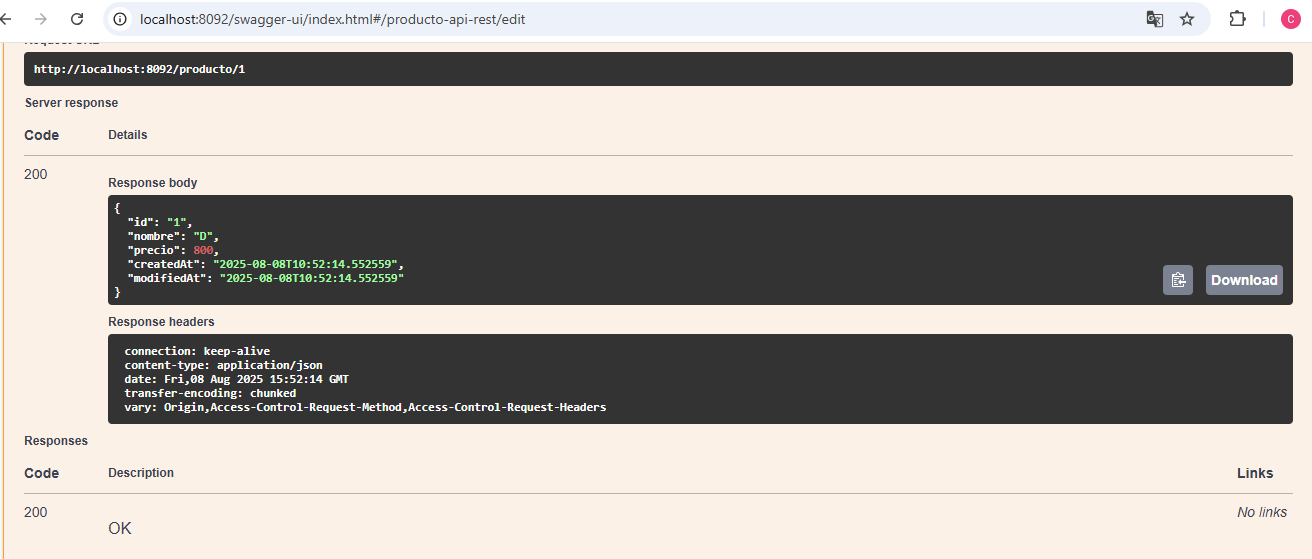
****

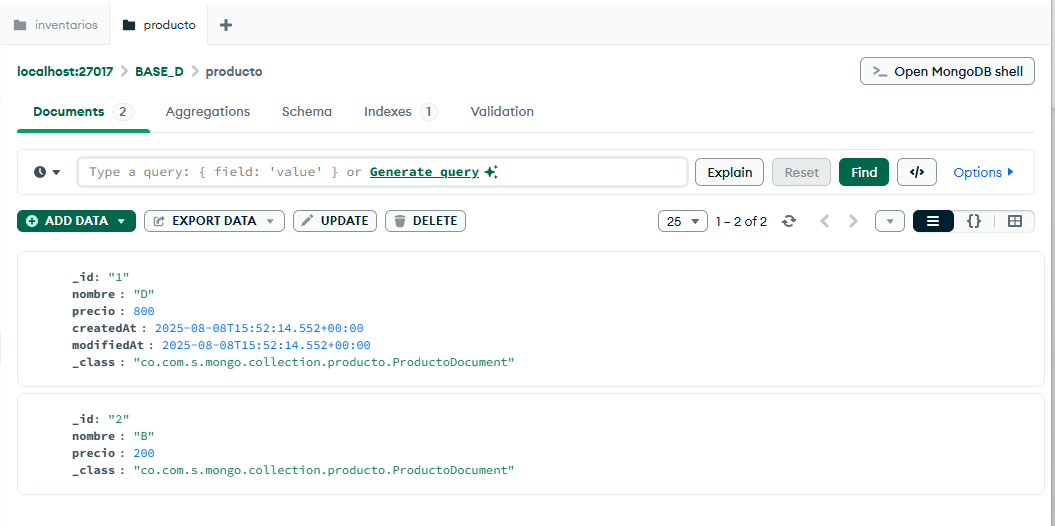
****

○ Actualizar un producto por ID.

Actualizar un producto por ID

****

****

****

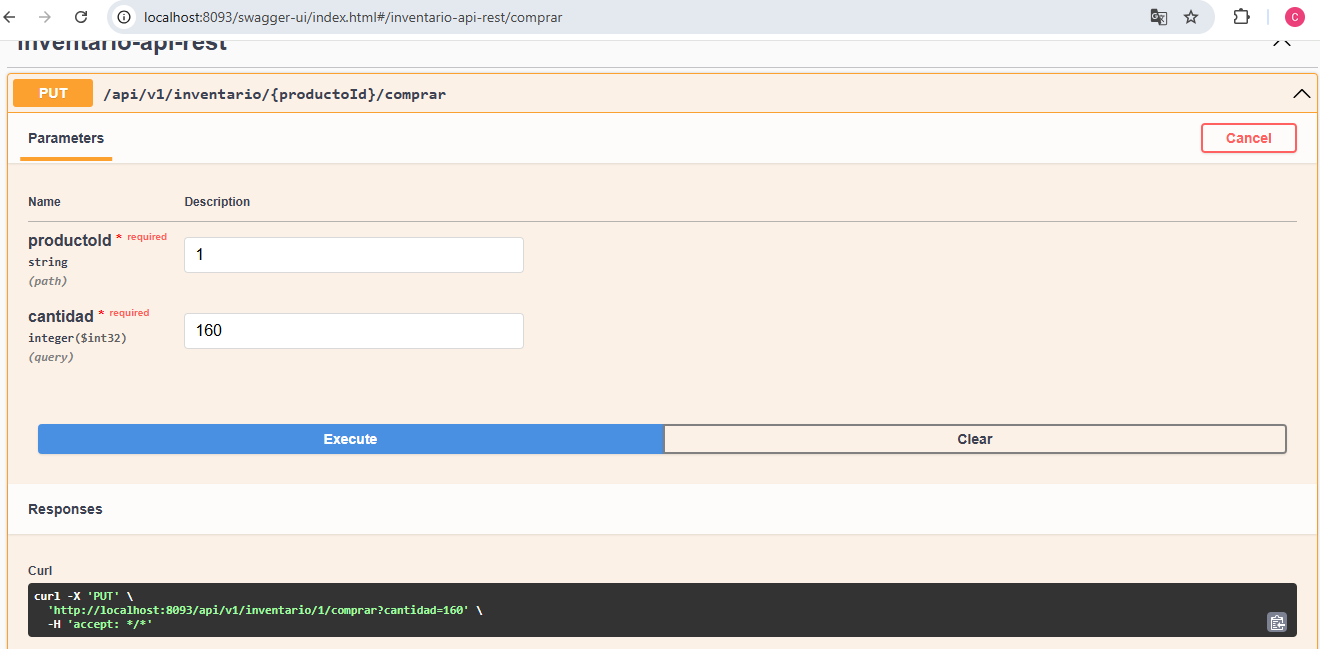
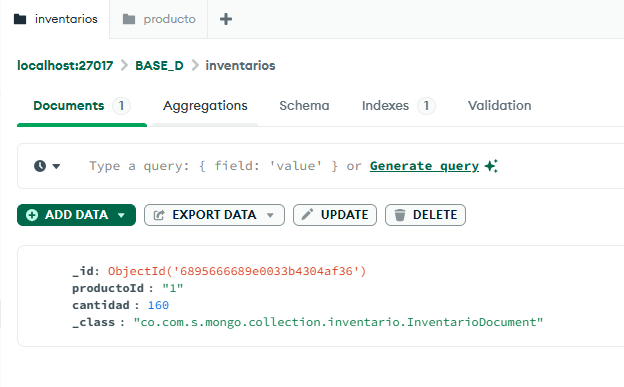
**Microservicio 2: Inventario**

* Gestionará un recurso llamado inventarios con los campos: producto\_id y cantidad.
* Permitirá:
  1. Consultar la cantidad disponible de un producto específico por ID (obtiene la información del producto llamando al microservicio de productos).

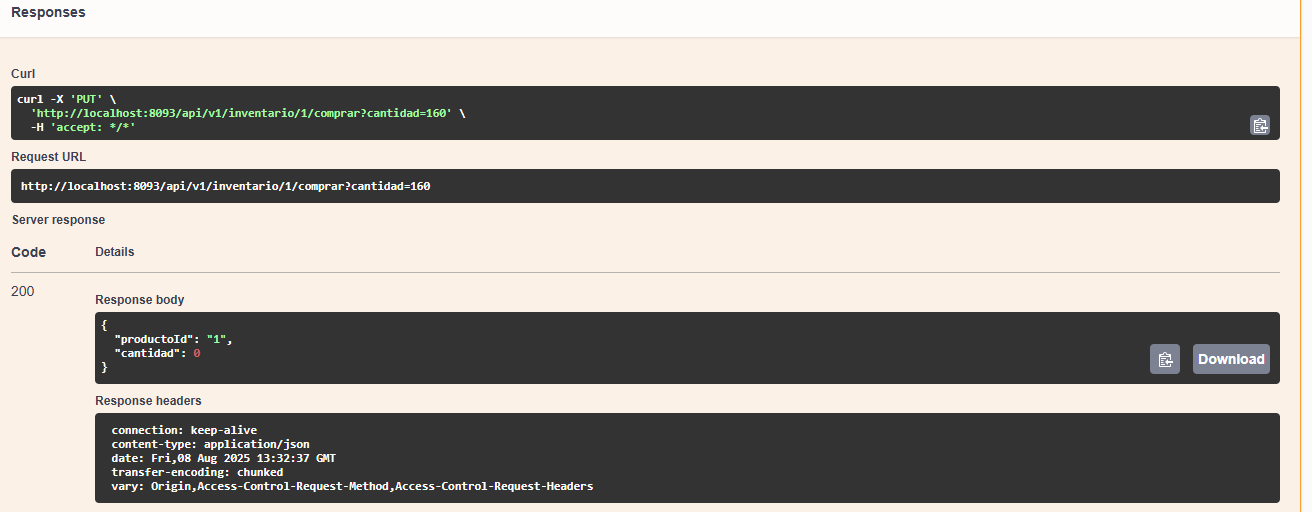
○ Actualizar la cantidad disponible de un producto específico tras una compra.

○ Emitir un evento simple cuando el inventario cambia (implementación básica puede ser un mensaje en consola).

**Actualizar la cantidad disponible de un producto específico tras una compra**

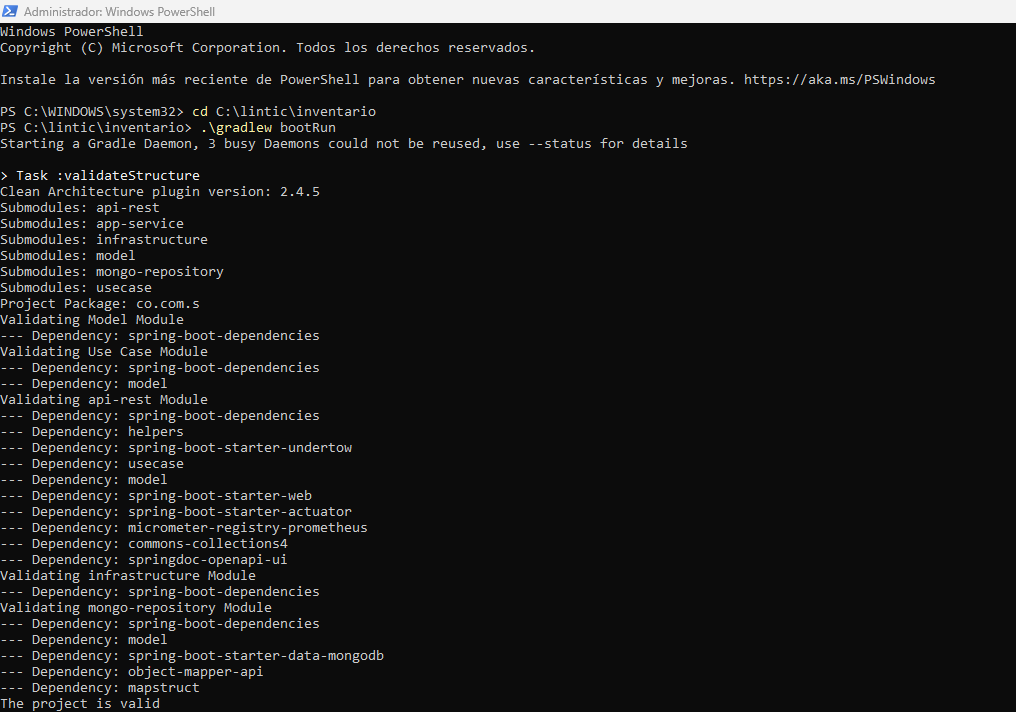
****

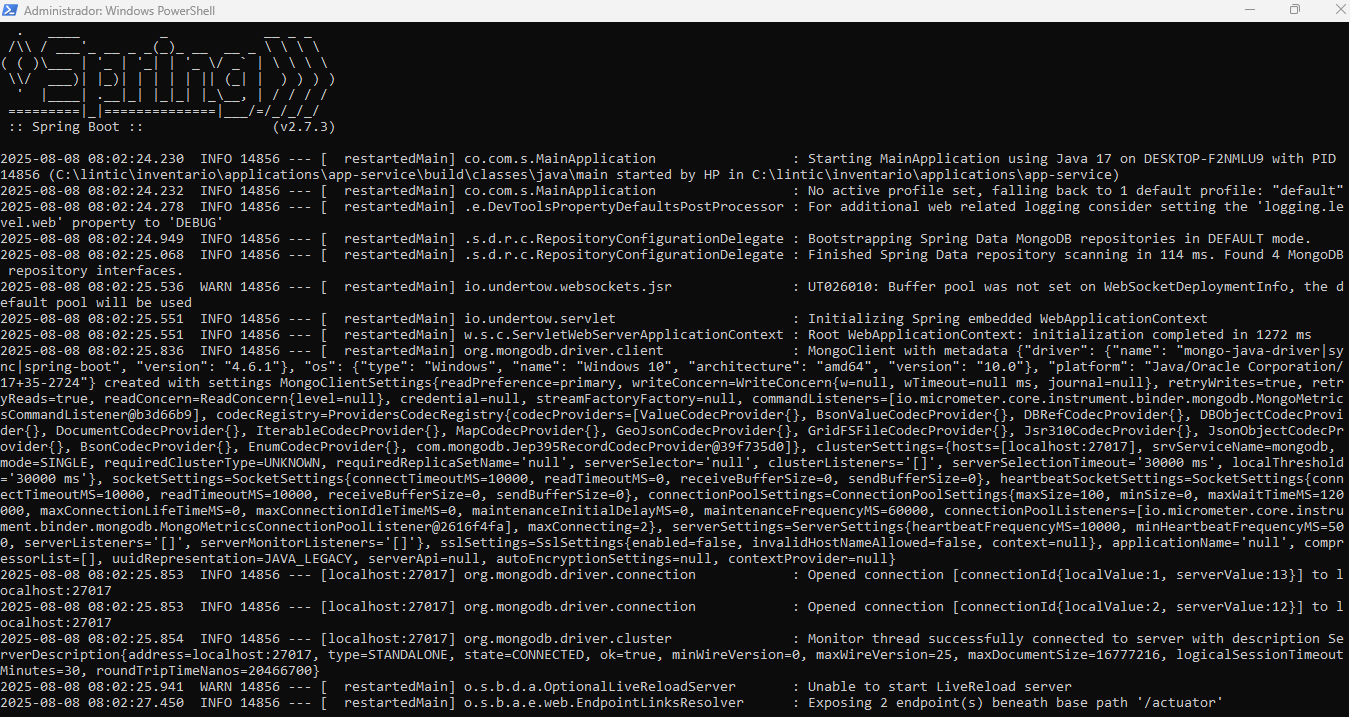
1. **Consultar la cantidad disponible de un producto específico por ID (obtiene la información del producto llamando al microservicio de productos).**

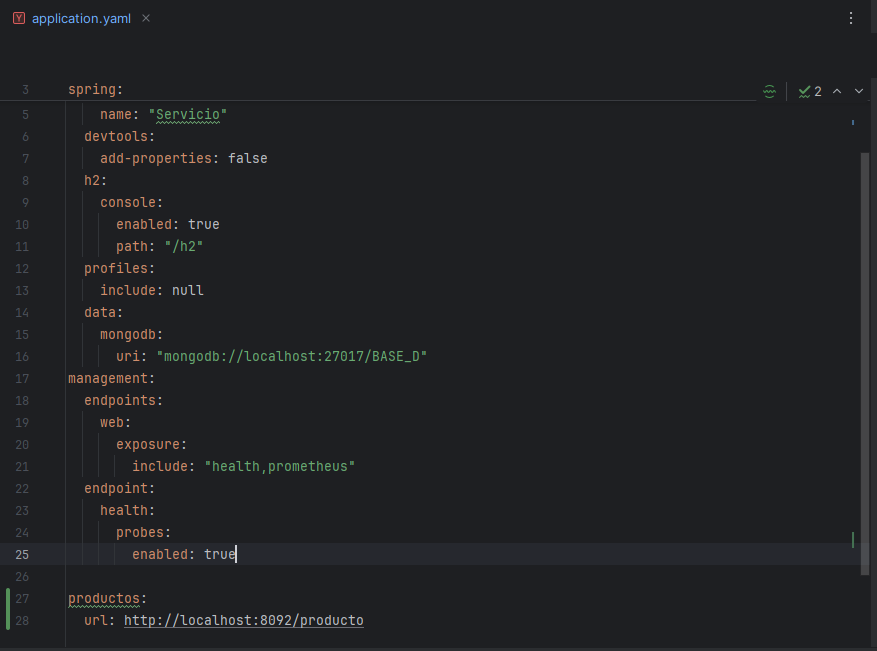


* El microservicio de inventario **recibió la solicitud** desde Swagger.
* Se conectó correctamente a MongoDB.
* Encontró el inventario del producto con ID 1.
* Restó la cantidad solicitada (160).
* El inventario quedó en 0 unidades.
* Se guardó exitosamente en la base de datos.

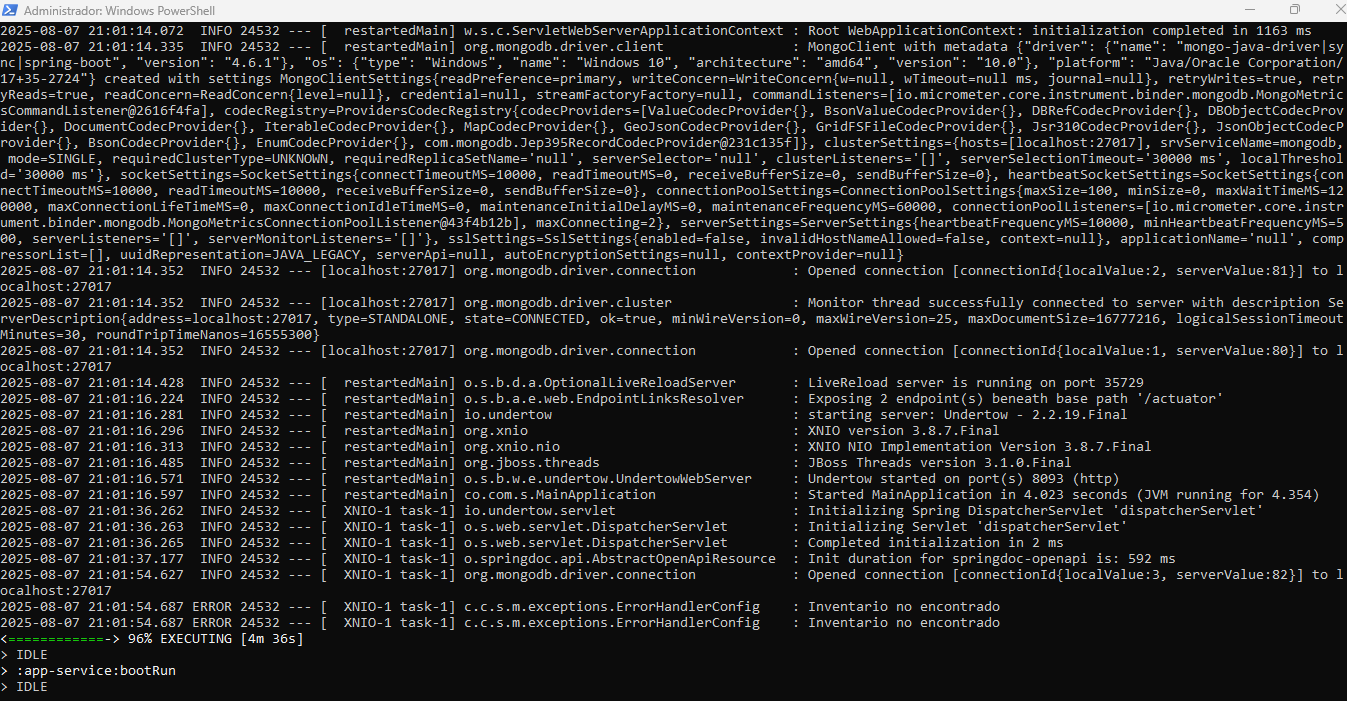
**Ejecución del Microservicio inventario**



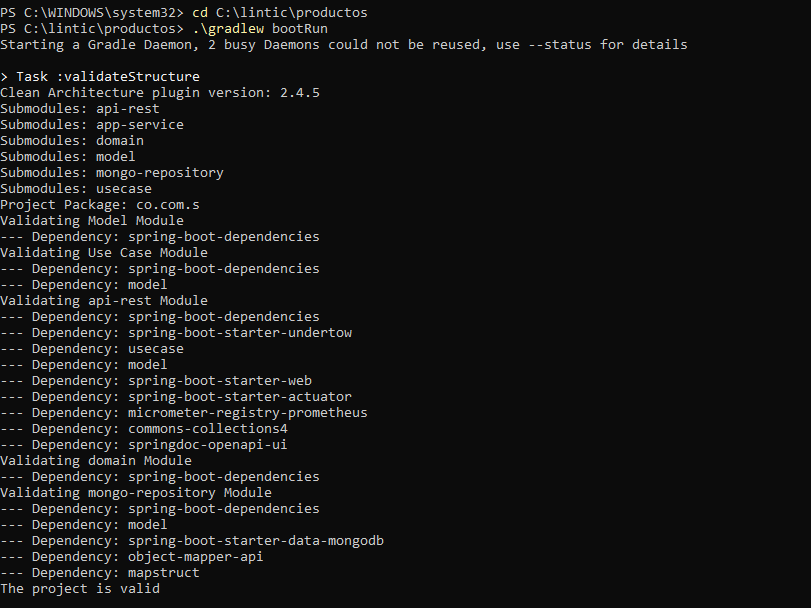


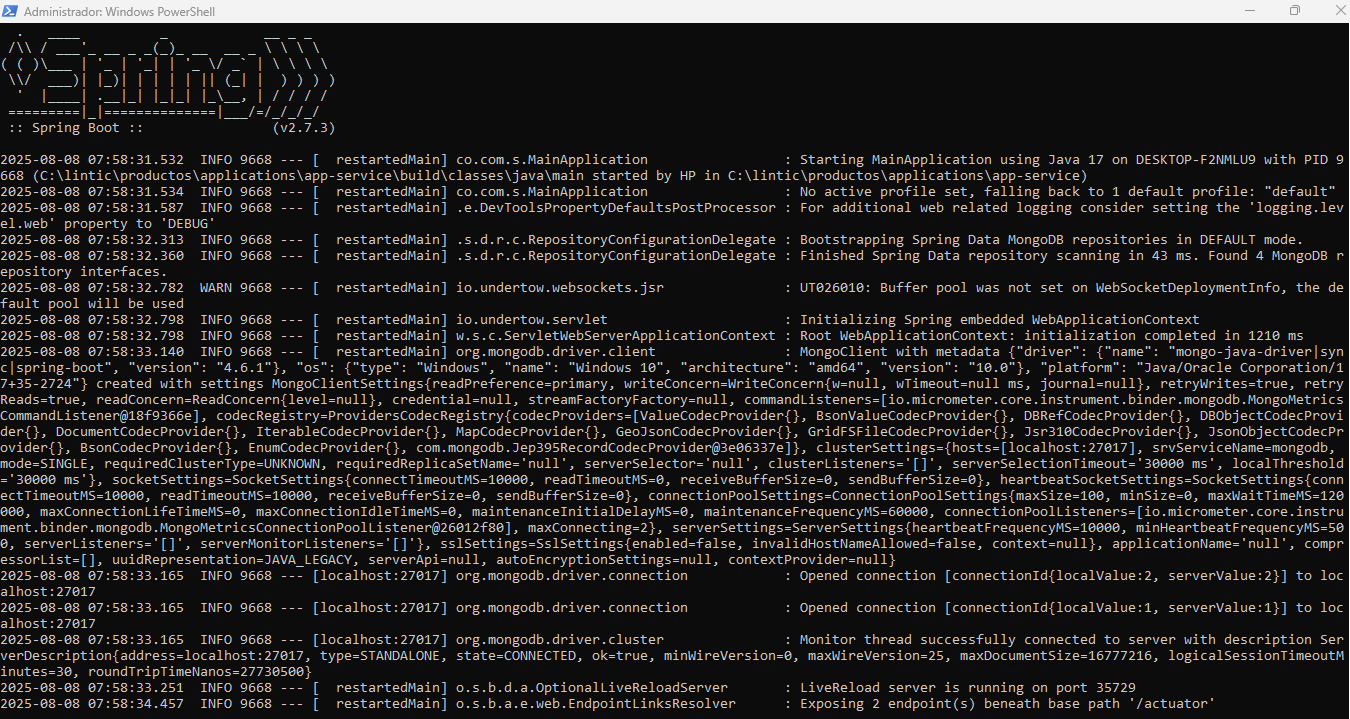
****

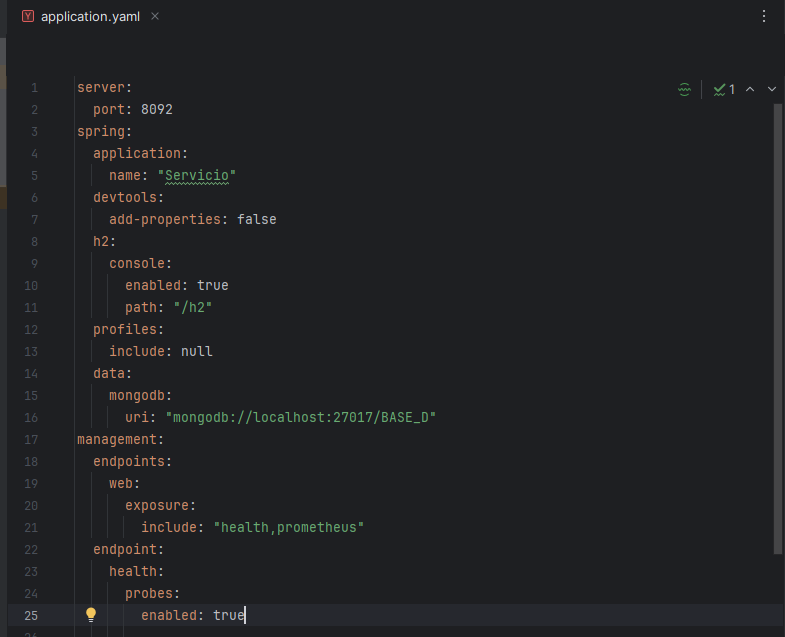
**Emitir un evento simple cuando el inventario cambia (implementación básica puede ser un mensaje en consola).**



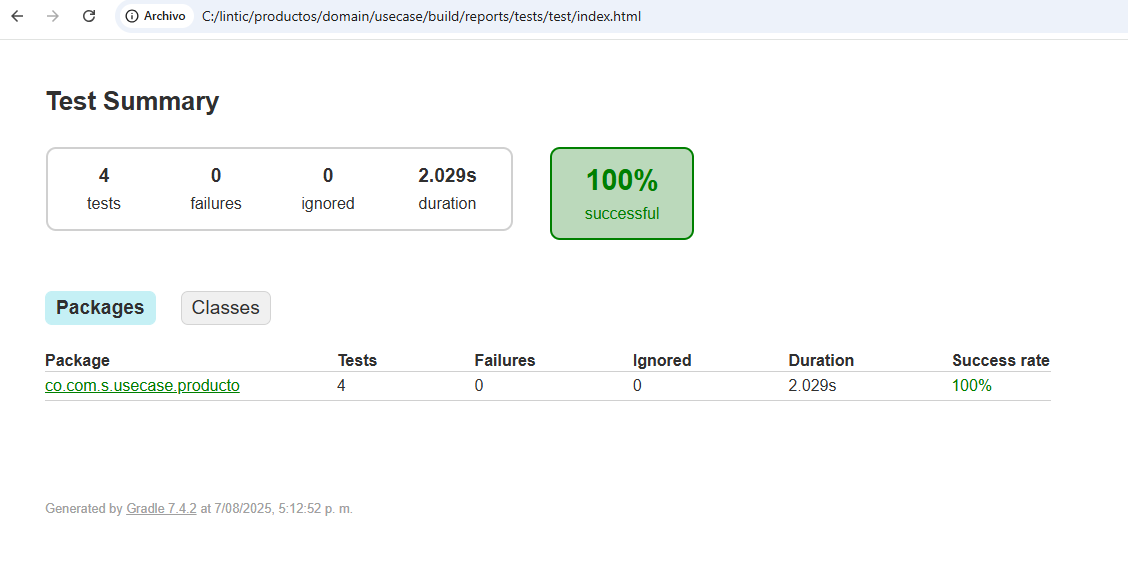
**Ejecución del Microservicio productos**

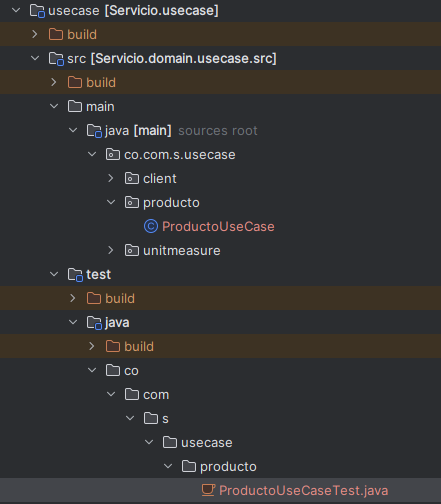


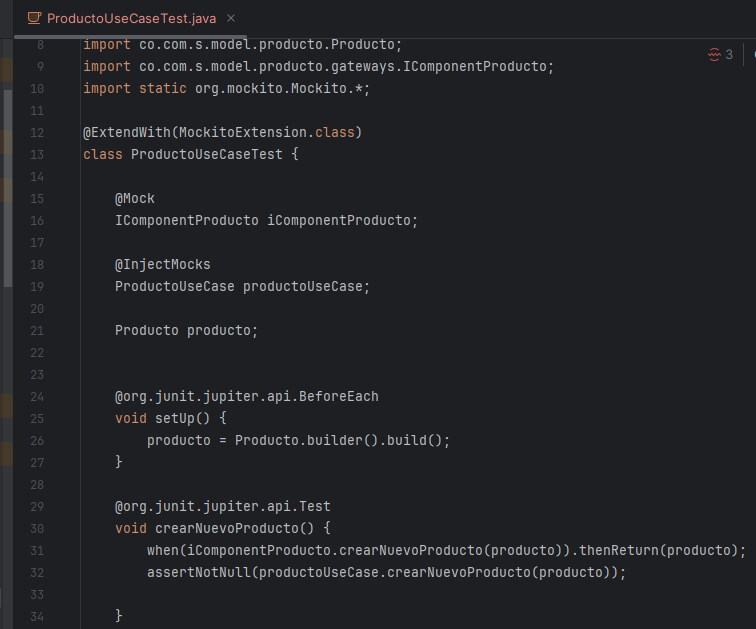


****

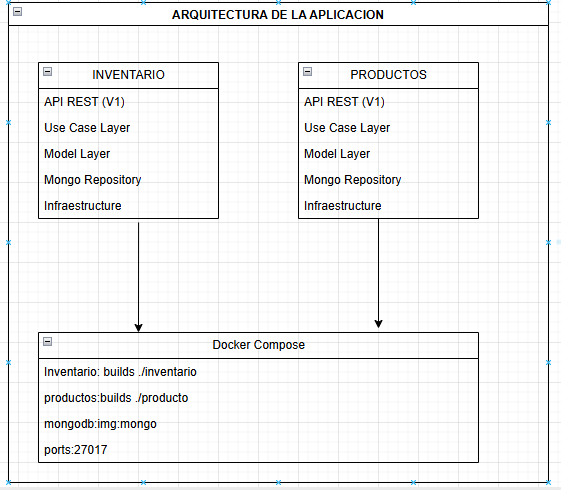
○ Alta cobertura en pruebas (mínimo 80%).



****

****

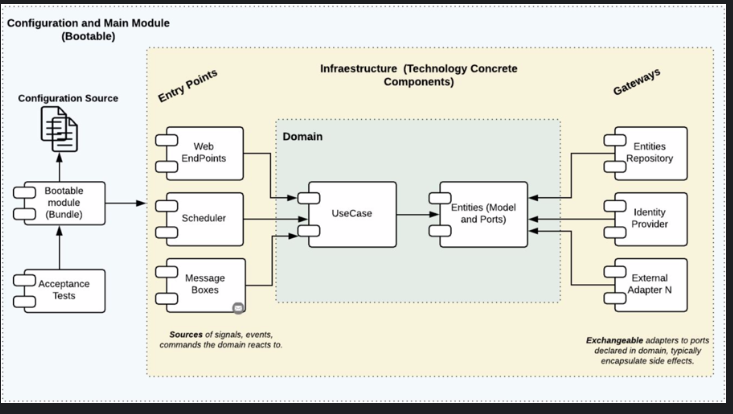
○ Documentación completa y diagrama de arquitectura.

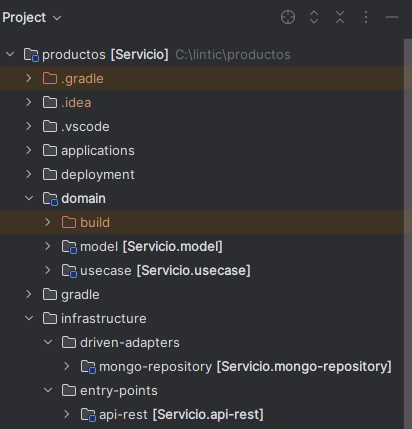
****

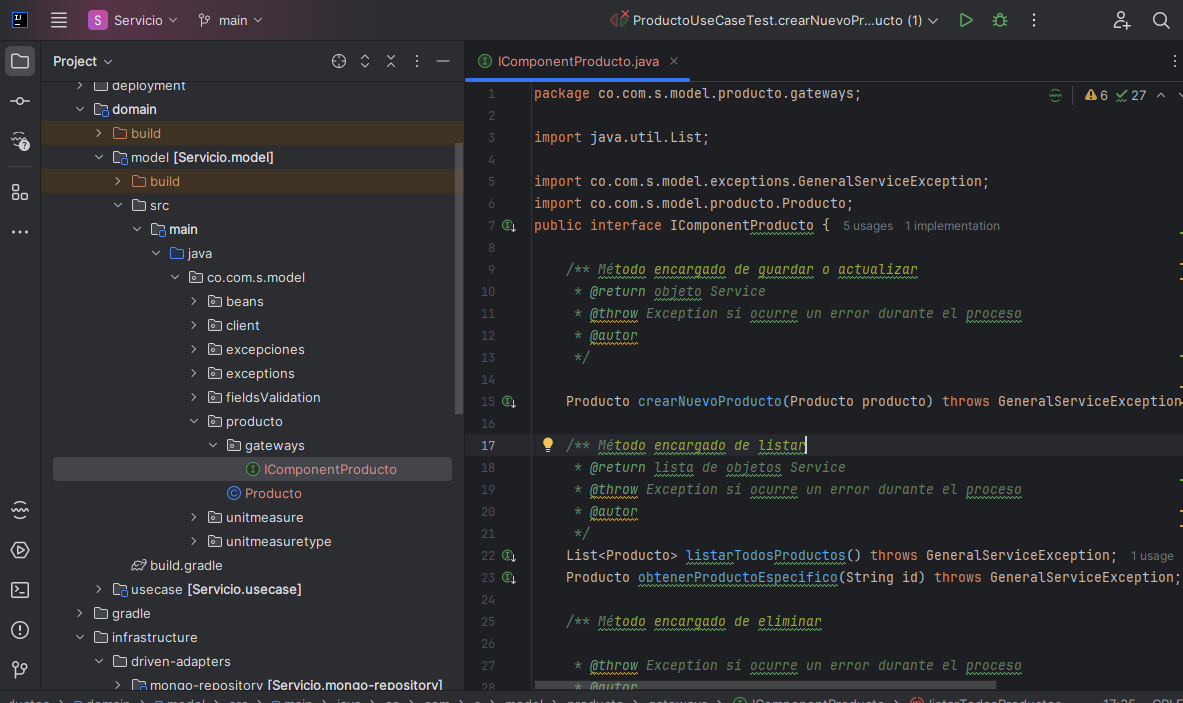
○ **Solución robusta con arquitectura clara.**

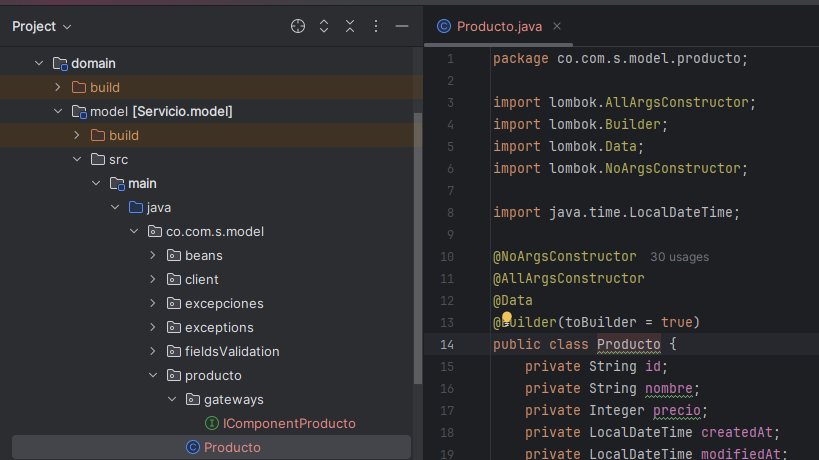
○ **Patrones de diseño claramente implementados y documentados.**

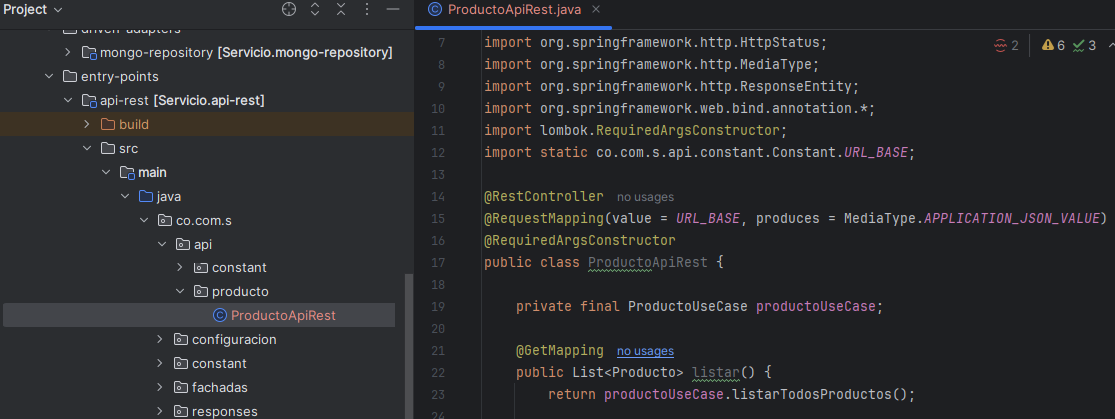
**Arquitectura limpia (Clean Architecture)**

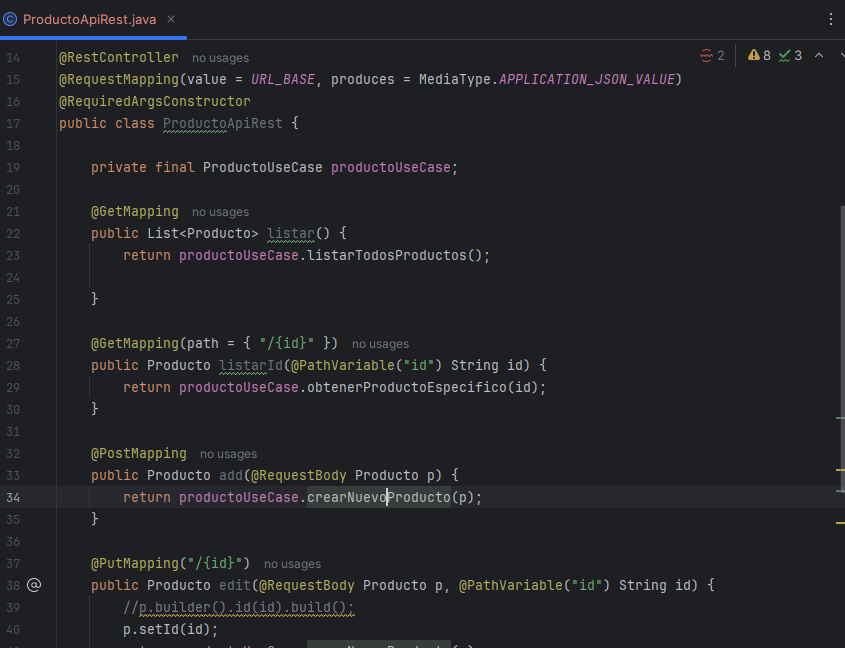


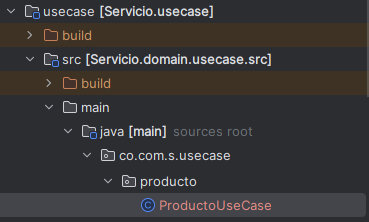


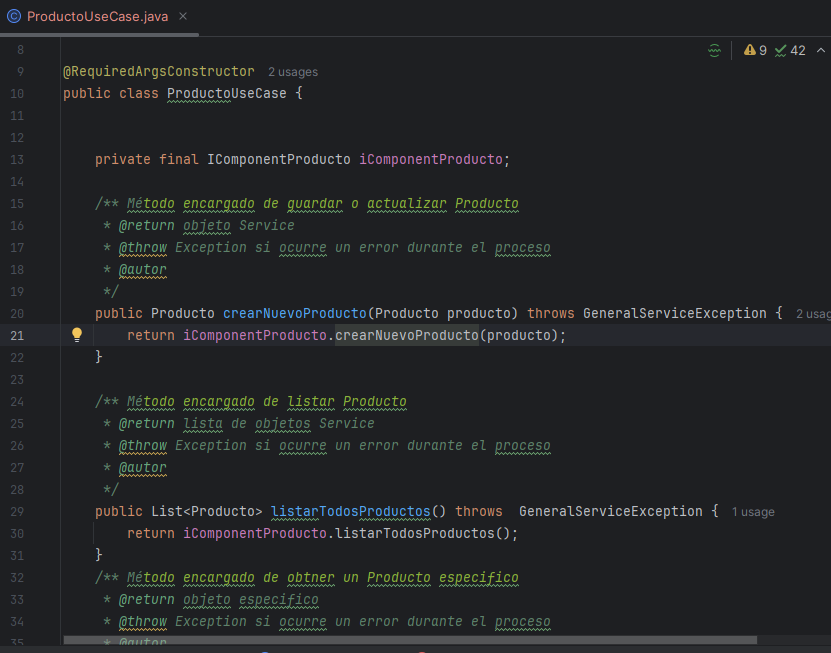


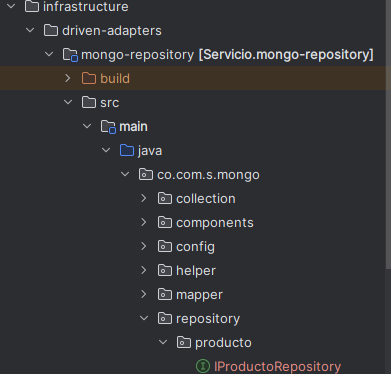


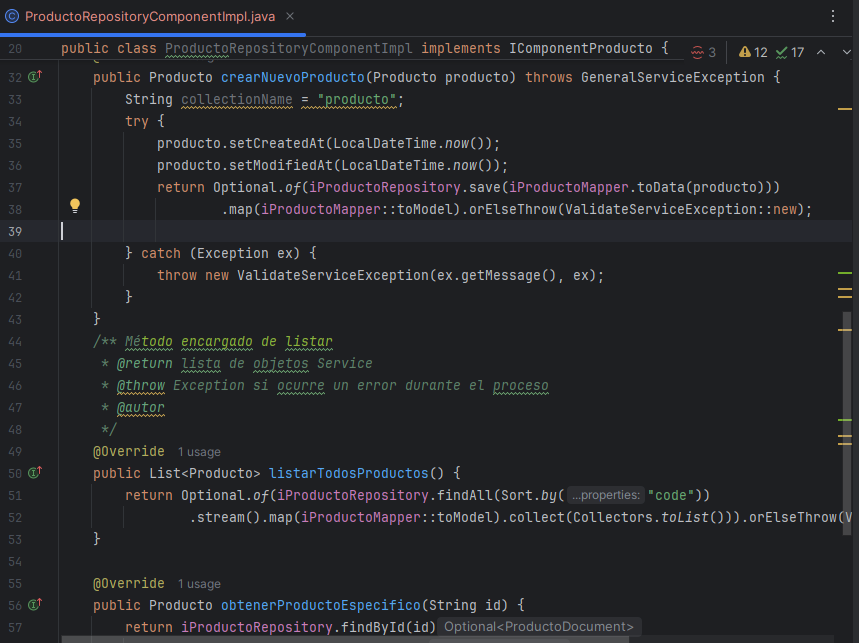


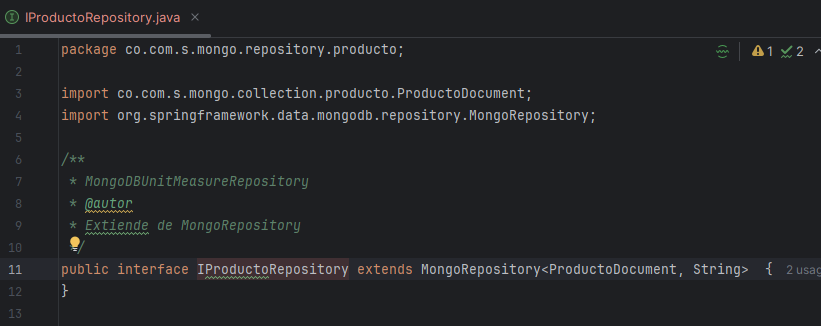


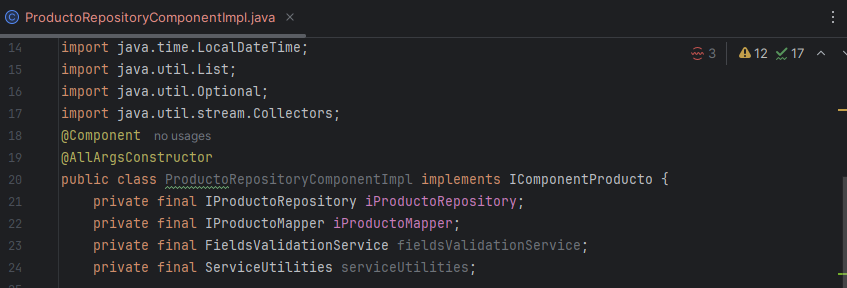


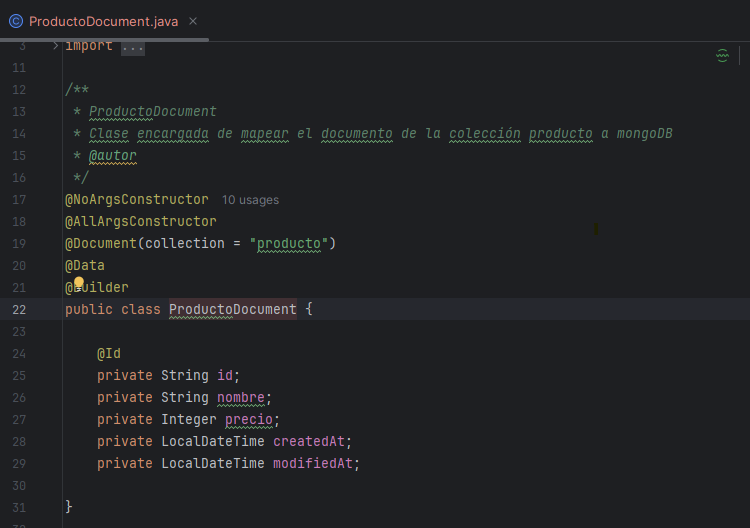


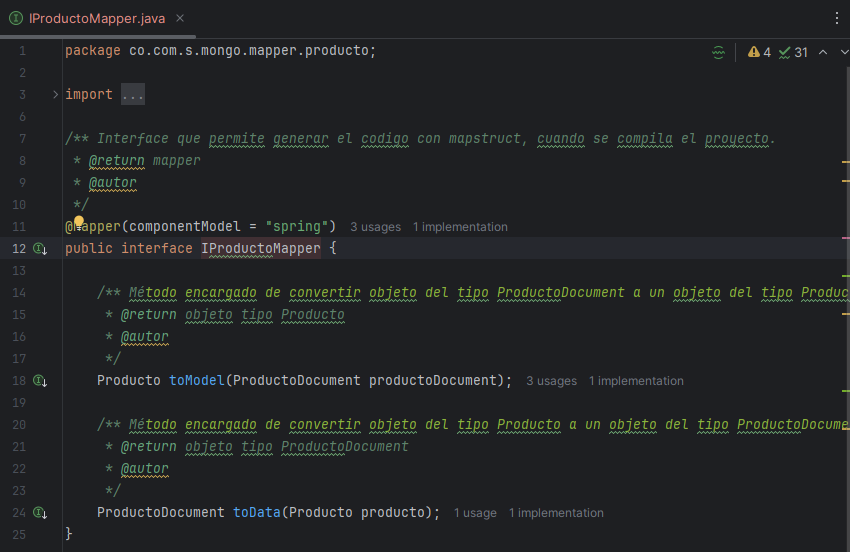


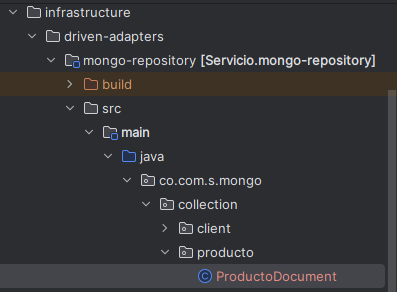








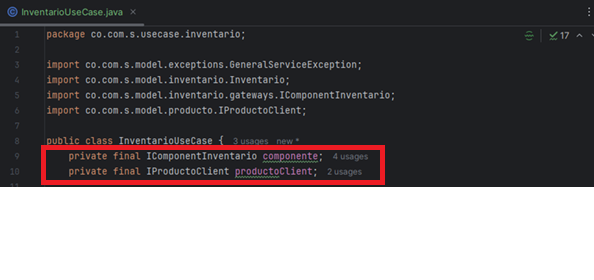




**Dependency Injection**

Usas Spring para inyectar dependencias como:

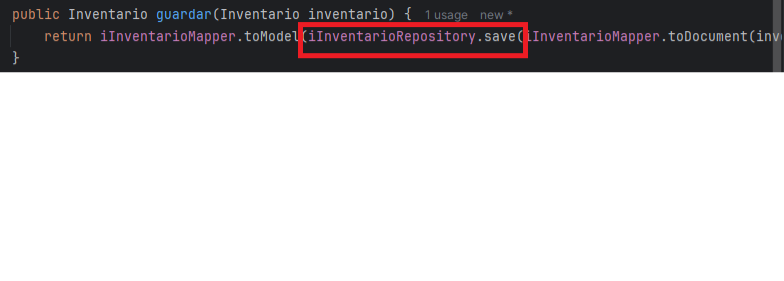
permitiendo desacoplar la lógica del caso de uso de su implementación concreta.

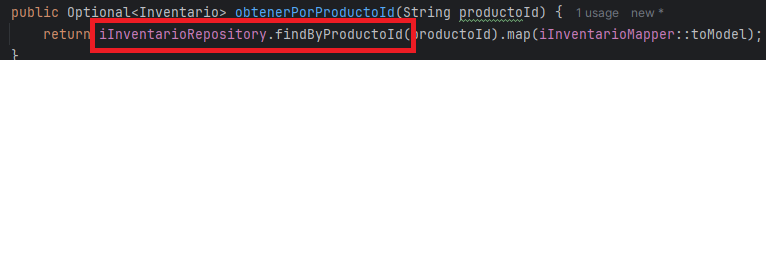


**Repository**

clase InventarioRepositoryComponentImpl actúa como un repositorio que encapsula el acceso a MongoDB

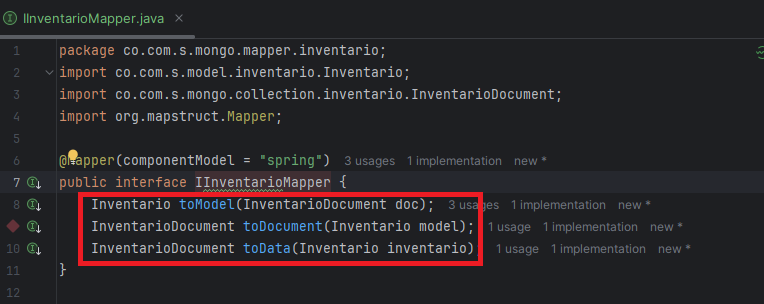
Esto separa la lógica de persistencia de la lógica de negocio.





**Mapper**

IInventarioMapper para convertir entre entidades de dominio y documentos de base de datos, útil para mantener la independencia entre capas:

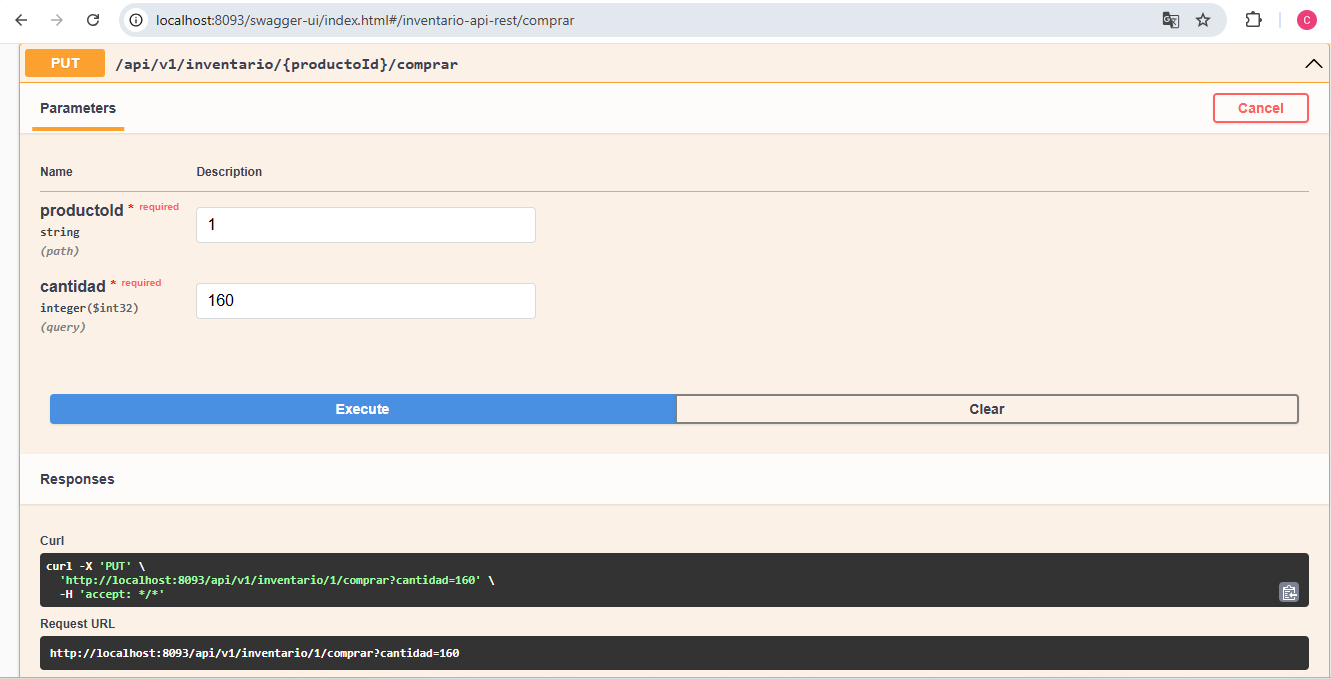


**Strategy**

ValidateServiceException, lo que sugiere un patrón de manejo de errores centralizado



**○ Estrategia de versionado de API.**



Esto indica que estoy usando versionado en la ruta:

**○ Guía de implementación para futuros desarrolladores.**

**Guía de Implementación: Microservicios Inventarios y Productos**

**Estructura general del proyecto**

Ambos microservicios siguen los principios de **Clean Architecture**, con los siguientes módulos:

* model: entidades de dominio y contratos (interfaces)
* usecase: lógica de negocio (casos de uso)
* mongo-repository: acceso a datos con MongoDB
* infrastructure: componentes auxiliares (helpers, clientes HTTP)
* api-rest: entrada HTTP (controladores, Swagger)
* app-service: punto de arranque (MainApplication)

**Requisitos previos**

* Java 17
* MongoDB corriendo en localhost:27017
* Gradle instalado
* Puertos disponibles: 8092 (productos), 8093 (inventario)

**Patrones de diseño aplicados**

| **Patrón** | **Aplicación en el proyecto** |
| --- | --- |
| Clean Architecture | Separación en módulos por responsabilidad |
| Repository | IInventarioRepository, InventarioRepositoryComponentImpl |
| Mapper | IInventarioMapper para conversión entre capas |
| Dependency Injection | Uso de @Component, @Autowired, @Value |
| API Versioning | Ruta base /api/v1/controladores REST |
| Exception Handling | ValidateServiceException |

**Microservicio: Productos**

* Propósito

Valida la existencia de productos mediante un endpoint REST.

**Componentes clave**

* ProductoApiRest: expone /producto/{id}
* ProductoCliente: cliente HTTP usado por Inventario
* application.yaml: define server.port y configuración MongoDB

**Endpoints**

* http

GET /api/v1/productos/{id}

* Devuelve 200 OK si el producto existe, o 404 si no.

**Microservicio: Inventario**

**🔹 Propósito**

Gestiona el stock de productos, permite consultar, actualizar y crear inventarios.

**🔹 Componentes clave**

* InventarioUseCase: lógica de negocio
* InventarioApiRest: controlador REST
* InventarioRepositoryComponentImpl: acceso a MongoDB
* ProductoCliente: verifica existencia de producto vía HTTP

**🔹 Endpoints**

* http
* GET /api/v1/inventario/{id}
* POST /api/v1/inventario // crear nuevo inventario
* PUT /api/v1/inventario/{id}/comprar?cantidad= #

**Comunicación entre microservicios**

* Inventario llama a Productos vía ProductoCliente usando WebClient.
* La URL se configura en application.yaml:

yaml

* productos:
* url: http://localhost:8092/api/v1/productos

**Pruebas**

* Usa Swagger en http://localhost:8092/swagger-ui.html y http://localhost:8093/swagger-ui.html
* Verifica que MongoDB tenga datos de prueba para productos e inventarios.
* Usa Postman para simular compras y validar actualizaciones de stock.

**○ Propuesta de mejoras y escalabilidad futura.**

* **Buenas prácticas para nuevos desarrolladores**
* Mantén la separación de capas: no mezcles lógica de negocio en controladores.
* Usa Optional para evitar null y lanzar excepciones controladas.
* Documenta cada endpoint con Swagger (@Operation, @Tag).
* Versiona tus APIs desde el inicio (/api/v1/).
* Usa @Value para URLs configurables y evita hardcoding.
* Implementa validaciones antes de modificar inventario (stock suficiente).
* Agrega pruebas unitarias para cada caso de uso.

# Criterios de evaluación

* Cumplimiento con el estándar JSON API.
* Claridad y simplicidad en la definición e interacción de microservicios.
* Calidad del código y buenas prácticas
* Cobertura y calidad de las pruebas.
* Uso correcto de Git (commits descriptivos, ramas adecuadas).
* Calidad de la documentación.

# Entrega

* Plazo máximo: 2 días desde la recepción de la prueba.
* **Comparte el enlace a un repositorio público con tu solución:**

**Ruta para Clonar el código:**

https://github.com/alovba1/pruebaslinktick.git

* **Incluye instrucciones claras para ejecutar los servicios y pruebas:**

**Pruebas**

* Usa Swagger en http://localhost:8092/swagger-ui.html y http://localhost:8093/swagger-ui.html
* Verifica que MongoDB tenga datos de prueba para productos e inventarios.
* Usa Postman para simular compras y validar actualizaciones de stock.
* El README debe incluir:

○ Instrucciones de instalación y ejecución.

○ Descripción breve de la arquitectura.

○ Decisiones técnicas y justificaciones.

○ Diagrama de interacción entre servicios.





