

# Caso: Sistema de Inventario

## 1. Contexto

Una empresa desea optimizar su sistema de gestión de inventarios para facilitar el control de productos en su almacén, permitiendo realizar operaciones como agregar, actualizar, eliminar y consultar información de productos de manera eficiente, segura y escalable.

Para abordar estos desafíos, se propone el desarrollo de dos microservicios independientes pero sincronizados mediante eventos:

1. **Microservicio de Productos:** Este servicio RESTful será responsable de gestionar la información relacionada con los productos, permitiendo realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar) sobre los datos de los productos. Deberá incluir medidas para garantizar la integridad y validez de los datos, así como controlar el acceso mediante autenticación y autorización.
2. **Microservicio de Inventario:** Este servicio se encargará de gestionar el stock de los productos en el almacén, incluyendo operaciones como el ajuste de cantidades disponibles, seguimiento de movimientos de inventario (entradas y salidas) y consultas sobre el estado del inventario. Este microservicio deberá suscribirse a eventos generados por el Microservicio de Productos (como la creación o eliminación de un producto) para mantenerse sincronizado, y a su vez, podrá emitir eventos relacionados con cambios en el inventario.

Ambos servicios estarán diseñados para trabajar de forma desacoplada, comunicándose entre sí mediante un sistema de mensajería basado en eventos, lo que asegura escalabilidad y resiliencia frente a futuros cambios en los procesos o requisitos del negocio.

## 2. Requerimientos funcionales

### 2.1. Operaciones CRUD en el Microservicio de Productos

1. **Crear un nuevo producto:** Permitir a los usuarios con permisos crear un producto con detalles como nombre, descripción, precio, categoría, y SKU.
2. **Obtener todos los productos existentes:** Retornar una lista de productos con sus detalles completos.
3. **Obtener productos por categoría específica:** Permitir filtrar productos por su categoría.
4. **Obtener un producto específico por su identificador:** Proveer información completa de un producto dado su ID.
5. **Actualizar un producto existente:** Modificar detalles como precio, nombre, descripción, o categoría de un producto existente.
6. **Eliminar un producto por su identificador:** Borrar un producto de la base de datos.

### 2.2. Operaciones CRUD en el Microservicio de Inventario

1. **Aumentar o reducir la cantidad de inventario de un producto:** Permitir registrar entradas o salidas de inventario, ajustando las cantidades.
2. **Consultar stock disponible de un producto:** Obtener la cantidad actual de un producto específico.
3. **Consultar historial de movimientos de inventario:** Proveer un registro de todos los cambios realizados en el stock de un producto.

### 2.3. Consultas Específicas

1. **Conversión de moneda en consultas de precios:** Los endpoints deben aceptar un parámetro opcional de moneda y devolver los precios convertidos según las tasas actuales. VER NOTAS
2. **Historial de precios:** Registrar y permitir la consulta del historial de precios de cada producto.

### 2.4. Autenticación y Autorización

1. **Autenticación:** Implementar autenticación mediante JWT para validar que solo usuarios autorizados puedan acceder al sistema.
2. **Roles de usuario:**
3. **Admin:** Permisos completos para crear, actualizar, y eliminar productos o realizar ajustes de inventario.
4. **User:** Acceso limitado a la consulta de productos e inventarios.
5. **Restricción de acciones sensibles:** Solo los usuarios con rol de administrador pueden modificar o eliminar productos, o realizar ajustes de inventario.

### 2.5. Caché

1. **Caché para consultas frecuentes:**
  - Consultas de productos (lista general y filtrada por categoría).
  - Consultas de precios y tasas de conversión.
  - Consultas de stock disponible.

### 2.6. Eventos y Sincronización

1. **Gestión de eventos:**
  - Notificar al Microservicio de Inventario cuando se crea, actualiza o elimina un producto.
  - Emitir eventos cuando se realicen ajustes de inventario.
2. **Sistema de eventos asíncrono:** Usar una cola de eventos como Apache Kafka o RabbitMQ para manejar los eventos, asegurando la sincronización entre servicios.
3. **Idempotencia en eventos:** Garantizar que los eventos procesados sean idempotentes para evitar inconsistencias en los datos en caso de reintentos.

## 3. Entregables

El candidato debe realizar las siguientes entregas mediante un enlace a un repositorio de control de versiones alojado en la plataforma de su preferencia (por ejemplo, GitHub, GitLab, Bitbucket):

### 1. Código Fuente Completo:

- Código fuente del proyecto que implemente los microservicios.
- Archivos de configuración relevantes (como `.env`, configuración y scripts de base de datos, etc.).

### 2. Documentación Detallada

- **Descripción de la solución:** Explicación de cómo se implementaron los microservicios y su interacción mediante eventos.
- **Archivo de Docker Compose:** Configuración para levantar el ecosistema completo, incluyendo servicios de base de datos, caché, y cola de eventos. VER NOTAS
- **Consideraciones de diseño o implementación:** Justificaciones de decisiones clave, como elección de tecnologías, patrones aplicados, y manejo de eventos.

### 3. Diagramas de Arquitectura y Flujo



- **Diagrama de arquitectura general:** Representación de los microservicios, bases de datos, caché, sistema de mensajería, y cómo interactúan entre sí.



### 4. Pruebas

- **Pruebas unitarias:** Implementación de pruebas unitarias para las funciones principales de ambos microservicios.
- **Informe de cobertura:** Archivo que muestre el porcentaje de cobertura alcanzado por las pruebas

**Todo el código fuente y la documentación deben ser entregados a través de un enlace a un repositorio de control de versiones (GitLab, GitHub, u otra plataforma similar). No se aceptarán archivos adjuntos por correo. La documentación y diagramas puede ser alojada en servicios en línea.**

## 4. Notas

-  Para la consulta de tasas, necesarias para conversión de monedas se puede usar API externas como [www.exchangerate-api.com](http://www.exchangerate-api.com).
-  Es de suma importancia que el entregable incluya el archivo de docker compose funcional, con el que se pueda levantar toda la infraestructura de la solución. Esto incluye contenedores, configuraciones y migración de base de datos (de ser necesario).

-  Puedes utilizar cualquier biblioteca o herramienta adicional que consideres necesaria para cumplir con los requisitos mencionados, siempre y cuando la solución siga siendo eficiente, escalable y segura.
-  Aunque no hay restricciones en cuanto a los lenguajes y frameworks de programación, se recomienda utilizar frameworks especializados en la creación de aplicaciones modernas, como NestJs, Quarkus y ASP.NET Core.

## 5. Referencias útiles

- [Mermaid.js](https://mermaid.js.org/) - Herramienta de generación de diagramas y gráficos a partir de texto.
- [Exchangerate-api.com](https://exchangerate-api.com/) - API gratuita para consultar tasas de conversión de monedas.
- [Nestjs.com](https://nestjs.com/) - Framework para construir aplicaciones del lado del servidor en JavaScript o TypeScript.
- [Quarkus.io](https://quarkus.io/) - Framework de Java para construir aplicaciones modernas y optimizadas para la nube.