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/) 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:
    - 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:
    - 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:
    - 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:**
    - Implementación básica funcional de ambos microservicios.
    - Docker básico para cada servicio.
    - Pruebas unitarias básicas (mínimo 40% cobertura).
- **Mid-level:**
    - 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:**
    - 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:**
    - Todo lo anterior más:
    - Patrones de diseño claramente implementados y documentados.
    - Estrategia de versionado de API.
    - Guía de implementación para futuros desarrolladores.
    - Propuesta de mejoras y escalabilidad futura.

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.
- Incluye instrucciones claras para ejecutar los servicios y pruebas.
- El README debe incluir:
    - Instrucciones de instalación y ejecución.
    - Descripción breve de la arquitectura.
    - Decisiones técnicas y justificaciones.
    - Diagrama de interacción entre servicios.

Nota importante

Prioriza la calidad sobre la cantidad de funcionalidades. Es mejor entregar menos características pero bien implementadas que muchas incompletas o con errores.