tenpo

Challenge Backend

API REST en Spring Boot

Informacion General:

Desarrollar una API REST en **Spring Boot (Java 21)** con las siguientes funcionalidades.

Funcionalidades principales

1. **Cálculo con porcentaje dinámico:**
   - Un endpoint que reciba num1 y num2, los sume y aplique un porcentaje adicional obtenido de un servicio externo (puede ser un mock con valor fijo).
2. **Caché del porcentaje:**
   - El porcentaje obtenido debe almacenarse en memoria durante **30 minutos**.
   - Si el servicio externo falla, se usa el último valor almacenado; si no hay, se devuelve un error.
3. **Historial de llamadas:**
   - Un endpoint que devuelva el historial de llamadas (fecha, endpoint, parámetros, respuesta o error).
   - El registro debe ser **asíncrono** para no afectar el rendimiento.

Requerimientos técnicos

1. **Base de datos:**
   - Usar **PostgreSQL** (en Docker) solo para almacenar el historial de llamadas.
2. **Despliegue:**
   - Ejecutar la API en un **contenedor Docker** con un docker-compose.yml para levantar la API y la base de datos.
3. **Documentación:**
   - Usar **Swagger** o **Postman**.
   - Incluir un **README.md** con instrucciones claras para ejecutar el servicio.
4. **Tests:**
   - Implementar **tests unitarios** para los cálculos y manejo de caché.

tenpo

Entrega

- Código en un **repositorio público (GitHub o similar)**.
- Imagen publicada en **Docker Hub** o docker-compose para levantar el proyecto.

Checklist de Evaluación

1. Funcionalidad Correcta

- **Cálculo con porcentaje dinámico** (suma + porcentaje del servicio externo o caché).
- **Caché del porcentaje** (almacenado 30 min, usado si el servicio externo falla).
- **Historial de llamadas** (fecha, endpoint, parámetros, respuesta/error, con paginación y registro asíncrono).

2. Calidad del Código

- **Estructura limpia** (capas bien definidas: Controller, Service, Repository).
- **Manejo de errores adecuado** (@ExceptionHandler, códigos HTTP correctos).
- **Buenas prácticas en Java** (código claro, nombres descriptivos, separación de responsabilidades).

3. Requerimientos Técnicos

- **Base de datos PostgreSQL** (corre en Docker, almacena historial de llamadas).}
- **Despliegue en Docker** (docker-compose.yml funcional, imagen en Docker Hub si aplica).
- **Documentación clara** (Swagger/Postman, instrucciones en README.md).
- **Pruebas unitarias** (JUnit, Mockito, simulación de fallos del servicio externo).