



## Challenge:

### API REST en Spring Boot

#### Información General:

Primero que todo, agradecemos tu interés en ser parte de Tenpo!

Nos entusiasma contar contigo en este proceso de selección para el rol de **Technical Lead (TL)**.

**Como parte de nuestro proceso, te invitamos a realizar un desafío técnico que nos permitirá conocer más a fondo tu experiencia, habilidades, y cómo abordan retos clave dentro de un entorno dinámico y colaborativo.**

En este desafío tendrás la oportunidad de demostrar tus habilidades, las cuales son fundamentales para el crecimiento y éxito de Tenpo.

#### Challenge:

Desarrolla una **API REST** en **Spring Boot** utilizando **Java 21**, que cumpla con las siguientes funcionalidades y requerimientos.

#### Funcionalidades principales:

##### 1. Cálculo con porcentaje dinámico:

0 Implementa un endpoint REST que reciba **dos números** (**num1** y **num2**).

- El servicio debe sumar ambos números y aplicar un porcentaje adicional obtenido de un servicio externo (por ejemplo, si recibe **num1=5** y **num2=5** y el servicio externo retorna un 10%, el resultado será  $(5 + 5) + 10\% = 11$ ).
- El servicio externo puede ser un **mock** que retorne un valor de porcentaje fijo (por ejemplo, 10%).

##### 2. Caché del porcentaje:

0 El porcentaje obtenido del servicio externo debe almacenarse en memoria (caché) y considerarse válido durante **30 minutos**.

- Si el servicio externo falla, se debe usar el último valor almacenado en caché.
- Si no hay un valor previamente almacenado en caché, la API debe responder con un error HTTP adecuado.

##### 3. Reintentos ante fallos del servicio externo:



- Si el servicio externo falla, se debe implementar una lógica de **reintento** con un máximo de **3 intentos** antes de devolver un error o usar el valor en caché.

#### 4. Historial de llamadas:

- Implementa un endpoint para consultar un historial de todas las llamadas realizadas a los endpoints de la API.
- El historial debe incluir detalles como:
  - Fecha y hora de la llamada.
  - Endpoint invocado.
  - Parámetros recibidos.
  - Respuesta (en caso de éxito) o error retornado.
- La consulta del historial debe soportar **paginación**.
  - **Nota:** El registro de las llamadas debe ser **asíncrono** para no afectar el tiempo de respuesta del servicio principal.
- Si el registro falla, no debe impactar la ejecución del endpoint invocado.

#### 5. Control de tasas (Rate Limiting):

- La API debe soportar un máximo de **3 RPM (requests por minuto)**.
    - Si se excede este umbral, debe responder con un error HTTP adecuado (por ejemplo, **429 Too Many Requests**) y un mensaje descriptivo.
- #### 6. Manejo de errores HTTP:
- Implementa manejo adecuado de errores HTTP para las series **4XX** y **5XX**. Incluye mensajes descriptivos para ayudar a los clientes a entender el problema.

### Requerimientos técnicos:

#### 1. Base de datos:

- Utiliza **PostgreSQL** para almacenar el historial de llamadas.
  - La base de datos debe correr en un contenedor Docker y ser configurada mediante Docker Compose.

#### 2. Despliegue:

- El servicio debe ejecutarse en un contenedor Docker.
- Publica la imagen en un repositorio público de **Docker Hub**.
  - Proporciona un archivo **docker-compose.yml** para levantar tanto la API como la base de datos fácilmente.

#### 3. Documentación:

- Genera documentación para la API utilizando **Swagger** o una colección de **Postman**.
- Incluye instrucciones claras en el repositorio sobre cómo:
  - Desplegar el servicio localmente.
  - Probar los endpoints.

#### 4. Tests:



0 Cubre la funcionalidad con **tests unitarios** para asegurar el correcto funcionamiento del servicio.

- Agregar tests para simular casos de error (fallo del servicio externo, límite de RPM excedido, etc.).

#### 5. Escalabilidad:

0 Diseña la aplicación para que pueda ejecutarse en un entorno con múltiples réplicas.

- Maneja adecuadamente el uso de caché compartida o soluciones distribuidas (por ejemplo, Redis) si es necesario.

#### 6. Bonus (Plus):

0 Si implementas el servicio utilizando **Spring WebFlux** o programación reactiva, se considerará un punto extra.

- Incluye un análisis en el README del repositorio justificando las decisiones técnicas tomadas.

### Entrega:

- **Repositorio público:** Sube el código a un repositorio público en **GitHub** o similar.
- **Instrucciones:** Proporciona un archivo **README.md** con:
  - Descripción del proyecto.
  - Instrucciones para ejecutar el servicio y la base de datos localmente.
  - Detalles sobre cómo interactuar con la API.
- **Docker Hub:** Comparte el enlace a la imagen publicada o docker-compose que permita levantar el proyecto.