FullStack Challenge

1. *Objetivo del Challenge*

Diseñar y desarrollar una aplicación web con un frontend en React y un backend en Java con Spring Boot, siguiendo las mejores prácticas de desarrollo de software. El objetivo es evaluar tus habilidades técnicas para implementar una solución escalable, bien documentada y mantenible.

2. Requerimientos Funcionales

Funcionalidad Principal:

- Crear una aplicación que permita registrar transacciones a una cuenta de un Tenpista.
- Para esto la aplicación debe permitir:
    - Crear nuevas transacciones, las cuales deben contener los siguientes campos:
        - Id transacción (int)
        - Monto transacción en pesos (int)
        - Giro o comercio de transacción (varchar)
        - Nombre de Tenpista (varchar)
        - Fecha de transacción (datetime)
    - Editar una transacción
    - Eliminar una transacción

Restricciones:

- Cada cliente puede tener un máximo de 100 transacciones.
- Las transacciones no pueden tener montos negativos.
- La fecha de transacción no puede ser superior a la fecha y hora actual

3. Requerimientos Técnicos

Backend:

- **Spring Boot:**
    - Implementar un API REST para manejar la funcionalidad descrita.
    - Endpoints sugeridos:
        - /transaction (con metodos GET/PUT/POST/DELETE).
- **Base de Datos:**
    - Utilizar PostgreSQL como base de datos relacional.
    - Diseñar una estructura adecuada para el almacenamiento de las transacciones.
- **Rate Limiting:**

- Implementar un límite de 3 request por minuto por cliente para evitar abusos del sistema.
- **Pruebas Unitarias:**
  - Incluir pruebas unitarias para los servicios, repositorios y controladores.
  - Será un plus utilizar mocks para pruebas.
- **Manejo de Errores:**
  - Implementar un manejador global de errores HTTP para devolver respuestas estructuradas y claras.
  - Ejemplo: Para un error de servidor implementar el código HTTP 500.
- **Documentación:**
  - Documentar los endpoints utilizando Swagger/OpenAPI.
  - Generar una UI de Swagger accesible en /swagger-ui.

Frontend

- **React:**
  - Crear una interfaz web moderna y responsiva.
  - Consumir los endpoints del backend:
    - El fetching debe ser realizado utilizando axios.
    - Será un plus el uso de react-query y el uso de caché de información cuando sea necesario para economizar los requests.
  - Componentes sugeridos:
    - Panel de cliente con listado de transacciones.
    - Formulario para agregar/editar transacciones.
- **Validación de Formularios:**
  - Validar los formularios en el frontend antes de enviar solicitudes al backend.

Escalabilidad y Despliegue

- Docker:
  - Crear contenedores para:
    - El backend.
    - La base de datos PostgreSQL.
    - El frontend.
  - Proporcionar un archivo docker-compose.yml para orquestar los servicios.
  - Publicación de Docker Images:
  - Publicar la imagen del backend en Docker Hub o cualquier otro registro público.

4. Criterios de Evaluación

- Correctitud:
  - Cumplimiento de los requerimientos funcionales y técnicos.
- Calidad del Código:

    - Legibilidad, organización, uso de patrones y mejores prácticas.
- Pruebas:
    - Cobertura y calidad de las pruebas unitarias.
- Documentación:
    - Claridad y completitud del README.md y Swagger.
- Uso de Docker:
    - Correcta configuración y funcionalidad del entorno Docker.
- Escalabilidad y Eficiencia:
    - Implementación adecuada de rate limit y manejo de errores.

5. Entregables:

- 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.

6. Tiempo para entrega

Se estima que este desafío debería completarse en un plazo de 7 días. Sin embargo, puedes organizar tu tiempo como prefieras. Si tienes alguna consulta, no dudes en contactarnos.

¡Buena suerte! 😊🚀