Prueba Técnica Agricapital SAS

Diseño y Desarrollo de APIs

El objetivo de esta prueba es evaluar la experiencia del candidato con los conceptos relacionados con las APIs, base angular de las funcionalidades implementadas en Agricapital.

- **Evaluación conceptual**

1. ¿Cuál es tu enfoque para manejar el versionado de una API?

2. ¿Cómo implementarías un sistema de limitación de tasa (rate limiting) en una API para manejar el abuso o tráfico excesivo?

3. Un cliente reporta un comportamiento inconsistente en las respuestas de tu API. ¿Cómo lo diagnosticarías y resolverías?

4. ¿Qué son los índices y cómo afectan el rendimiento de la base de datos?

5. ¿Cómo abordarías la refactorización de código heredado mientras minimizas las interrupciones?

Solución evaluación conceptual:

1. ¿Cuál es tu enfoque para manejar el versionado de una API?

El enfoque que utilizo para exponer diferentes versiones de una API es principalmente en la URL, ejemplo, https://www.nombresitio.com/v1/libros, pero este enfoque puede volverse complicado al manejar muchas versiones, sin embargo es fácil de identificar y se puede apuntar a una versión especifica sin dificultad. El otro enfoque también es muy parecido al anterior pero es más basado en recursos, ejemplo, https://www.nombresitio.com/libros/v1, aquí se tiene más claridad para versiones especificas de un solo recurso. También sé que existe el versionado por subdominios de la forma https://v1.api.nombresitio.com/libros, que aunque es mas organizado para equipos que manejan múltiples versiones también la configuración puede ser más compleja.

2. ¿Cómo implementarías un sistema de limitación de tasa (rate limiting) en una API para manejar el abuso o tráfico excesivo?

En Spring Boot por ejemplo se puede agregar una dependencia de la biblioteca Bucket4j, en el archivo pom.xml, luego necesitamos crear una clase que implemente la interface Filter y que especifique las reglas de limitación como 10 solicitudes por minuto o el valor que indiquemos. Para identificar el cliente se puede usar un token o identificador único. La librería Bucket4j permite la solicitud si el bucket tiene capacidad, sino devuelve un error HTTP 429. Redis también es muy utilizado o las bases de datos para almacenar los buckets en caso de necesitar escalabilidad o datos persistentes.

3. Un cliente reporta un comportamiento inconsistente en las respuestas de tu API. ¿Cómo lo diagnosticarías y resolverías?

Primeramente, pido al cliente que me dé la mayor y mas detallada información que pueda, como a qué se refiere con inconsistente, si es que que no está recibiendo la respuesta correcta, o si es lentitud, cuando sucede, qué cliente está usando, por ejemplo, si es postman o alguna biblioteca HTTP, si tiene alguna configuración especifica. Luego intento reproducir el problema usando los ejemplos del cliente, pero en mi entorno y con herramientas como postman. Si el problema no ocurre en mi entorno entonces trataría de conectar desde una red similar a la del cliente o emular las condiciones de su entorno. Algo que es muy importante a tener en cuenta es habilitar el modo debug en el sistema, por ejmplo cuando algo así me pasa desarrollando alguna API con springboot, lo que hago es ir al archivo application.properties y agregar la configuración logging.level.org.springframework.web=DEBUG, esto para ver los logs de una manera mas detallada y poder dar solución a la inconsistencia. También puede pasar que se puede deber a la latencia del endpoint afectado, uso de recursos o tiempos de respuesta para solicitudes recientes y esto en los casos que se tenga la API en AWS, se puede monitorear con AWS CloudWatch.

4. ¿Qué son los índices y cómo afectan el rendimiento de la base de datos?

Un índice es una estructura de datos como árboles o tablas hash que se crea en una tabla para acelerar la búsqueda de datos, es similar al índice de un libro donde si se quiere encontrar un tema en concreto no se busca página por página, sino que mediante el índice nos podemos dirigir directamente a la página donde está el tema que buscamos. Los índices pueden afectar tanto positiva como negativamente. Los efectos positivos serían que los índices permiten localizar rápidamente las filas relevantes, en lugar de recorrer una tabla que tal vez podría tener millones de registros, también mejoran las operaciones con las cláusulas where, join, order by, group by en caso de que las columnas involucradas tengan índices. Por otro lado, los efectos negativos serían que cada vez que se inserta, actualiza o elimina una fila, los índices relacionados se deben actualizar y esto puede hacer mas lento las operaciones de escritura especialmente en tablas con muchos índices. Como los índices son estructuras adicionales que ocupan espacio, entonces en tablas grandes o con muchos índices, esto podría ser muy significativo y afectar el rendimiento. El mal uso también puede

ser contraproducente para tablas pequeñas o columnas poco utilizadas lo cual generaría una sobrecarga sin mejorar el rendimiento.

5. ¿Cómo abordarías la refactorización de código heredado mientras minimizas las interrupciones?

La forma en que abordaría la refactorización de código heredado reduciendo al mínimo las interrupciones sería primero entendiendo el código existente, para lo cual es necesario leer la documentación en caso que tenga o comentarios en el código, ver cuáles son las dependencias del código como bases de datos, servicios externos, módulos internos y en lo posible mapear los flujos principales del sistema y de cómo encajan sus distintas partes. También es importante escribir pruebas unitarias paras funciones o clases muy importantes y desarrollar pruebas de integración para garantizar que los módulos trabajen correctamente, aunque hay veces que ya existen pruebas entonces lo ideal sería asegurarse que sean confiables ejecutándolas. Aquí aplica mucho lo de divide y vencerás, por tanto es bueno identificar módulos, o funciones, o componentes, es decir, pequeñas secciones de código para refactorizar, garantizando que todo siga funcionando después de cada cambio para minimizar el riesgo de interrupciones y facilitar mejor la identificación de problemas. Como el objetivo de refactorizar es mejorar la estructura del código y no cambiar la funcionalidad, entonces hay que buscar renombrar funciones o variables para que sean mas claras, eliminar el código duplicado y hacer el código mas módular o legible. También es lo recomendable tener el código original en funcionamiento mientras se desarrolla la versión refactorizada en paralelo, para que cuando esté lista y probada reemplace la versión antigua. Aquí es muy útil git para en una rama separada ir creando la versión refactorizada y tener un control de los cambios. También tengo entendido que mientras se hacen los cambios es bueno informar y coordinar los equipos que puedan verse afectados por la refactorización como clientes internos por ejmplo. Por último, sería bueno tener un plan de rollback para revertir los cambios rápidamente en caso de que algo no funcionara como lo esperado.