Optimización en sistemas financieros:

1. Imagina que trabajas en una aplicación bancaria que debe procesar transacciones en tiempo real. ¿Qué estrategias aplicarías para mejorar el rendimiento y la escalabilidad? Considera aspectos como concurrencia, caché y patrones de diseño.

Respuesta: (use GPT para mejorar la respuesta pero la idea general si es mia)

* **Uso de métodos asincrónicos:** Implementar llamadas asincrónicas en la integración de APIs para reducir los tiempos de espera y mejorar la velocidad de respuesta del sistema. Esto permite procesar múltiples solicitudes simultáneamente sin bloquear el flujo principal.
* **Consultas bajo demanda:** Ejecutar únicamente las consultas necesarias en el momento en que se requieren, evitando cargas innecesarias sobre la base de datos y reduciendo el consumo de recursos.
* **Diseño modular:** Organizar el sistema de forma modular, de modo que solo se actualicen y procesen las partes estrictamente necesarias de la aplicación, lo que mejora la eficiencia y facilita la escalabilidad.
* **Caché de datos:** Utilizar mecanismos de caché para almacenar temporalmente información frecuentemente solicitada, reduciendo la carga en la base de datos y acelerando las respuestas a los usuarios.
* **Concurrencia y paralelismo:** Implementar colas de trabajo y procesamiento concurrente de transacciones para manejar altos volúmenes de operaciones en tiempo real sin comprometer la consistencia ni la disponibilidad.

2. Seguridad en APIs financieras

a. Explica cómo protegerías una API que maneja información sensible de cuentas bancarias contra ataques como inyección SQL, CSRF, XSS y otros ataques comunes.

Respuesta: (use GPT para mejorar la respuesta pero la idea general si es mia)

**Prevención de inyección SQL:** Utilizar **parametrización de consultas** y **procedimientos almacenados** para asegurar que los datos ingresados por el usuario no puedan alterar las consultas SQL de manera maliciosa.

**Validación de datos:** Implementar **validaciones tanto en el frontend como en el backend** para asegurar que los campos reciban solo datos del formato esperado, evitando inyecciones de código y datos corruptos.

**Protección contra CSRF (Cross-Site Request Forgery):** Usar **tokens CSRF** y mecanismos de autenticación robustos para garantizar que las solicitudes provengan de usuarios legítimos.

**Otras medidas de seguridad:**

* Autenticación y autorización sólida (JWT, OAuth2).
* Uso de HTTPS para cifrar la comunicación.

**3. Transacciones en sistemas distribuidos:**

a. En un sistema bancario distribuido, ¿cómo implementarías la consistencia y el manejo de errores en una API que procesa transferencias entre cuentas en diferentes servicios?

Respuesta: (user GPT para mejorar la respuesta pero la idea general si es mia)

**Estructura estandarizada de respuesta:** Definir un **JSON uniforme** para todas las APIs, que indique claramente si la operación fue exitosa o si hubo errores, incluyendo códigos de error y mensajes descriptivos. Esto facilita la validación y el manejo de fallos por parte de los servicios consumidores.

4. Pruebas unitarias y de integración:

a. Describe cómo diseñarías una suite de pruebas que asegure la correcta operación de una API bancaria, considerando tanto pruebas unitarias como de integración. ¿Qué herramientas utilizarías?

Respuesta: (user GPT para mejorar la respuesta pero la idea general si es mia)

**Pruebas unitarias:** Desarrollar tests que verifiquen el correcto funcionamiento de cada componente de forma aislada, asegurando que las funciones de procesamiento de transacciones, validación de datos y lógica de negocio se comporten como se espera. Herramientas recomendadas: **xUnit, JUnit, NUnit** o **Jest**, según el stack tecnológico.

**Pruebas de integración:** Verificar que los diferentes módulos y servicios de la API interactúen correctamente, incluyendo bases de datos, servicios externos y microservicios. Esto asegura que las transferencias, consultas y actualizaciones de cuentas funcionen de manera consistente. Herramientas recomendadas: **Postman, REST Assured, Supertest**.

**Automatización y flujo de CI/CD:** Integrar los tests en pipelines de **CI/CD** para ejecutar las pruebas automáticamente en cada despliegue, garantizando calidad continua.

5. Front-end:

a. En una aplicación bancaria que muestra el saldo de las cuentas, ¿cómo gestionarías el estado y la autenticación en el front-end, garantizando la seguridad y la coherencia de los datos?

Respuesta: (user GPT para mejorar la respuesta pero la idea general si es mia)

**Servicio de autenticación centralizado:** Crear un **servicio de autenticación** que maneje el login, la renovación de tokens (JWT o similares) y la verificación del estado de sesión, asegurando que solo usuarios autenticados puedan acceder a la información sensible.

6. SPRING BOOT (No soy tan bueno en java lo que hice lo hice con una noche de estudiar para la prueba con ganas de seguir aprendiendo)

a. Que pasa en una aplicación internamente cuando usas la anotación

@SpringBootApplication y cómo afecta el arranque de una aplicación?

b. ¿Cómo funciona el ciclo de vida de un beans en Spring y como podrías

intervenir en él?

c. ¿Cómo personalizarías el comportamiento de la auto-configuración en Spring

Boot sin romper la filosofía de 'convención sobre configuración'?

Nota:

“No tengo experiencia avanzada en Java ni fuerte es .Net pero la lógica de programación ya la tengo, pero pude desarrollar funcionalidades básicas en Spring Boot estudiando intensamente para la prueba. Esto me permitió comprender conceptos esenciales como controladores REST, inyección de dependencias y manejo de servicios. Estoy motivado a seguir profundizando mis conocimientos en Java y Spring Boot para poder implementar soluciones más complejas y optimizadas en proyectos reales.”