**Parte Teórica (30%)**

1. **Optimización en sistemas financieros**:  
   a. 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.

Aplicaría un enfoque integral basado en programación asíncrona y control de concurrencia, utilizando técnicas como colas de mensajes para desacoplar procesos y garantizar un procesamiento eficiente de múltiples transacciones simultáneas; además, implementaría soluciones de caché en memoria, como Redis, para reducir la latencia en accesos frecuentes, y diseñaría la arquitectura mediante microservicios, aplicando patrones como CQRS y event-driven para lograr una escalabilidad horizontal robusta, complementado con una optimización de la base de datos (indexación, particionamiento y replicación) y un monitoreo continuo para anticipar y mitigar cuellos de botella.

1. **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.

Para proteger una API financiera que maneja información sensible, adoptaría una estrategia de defensa en profundidad. Esto incluiría el uso de autenticación y autorización robustas (por ejemplo, mediante OAuth2 o tokens JWT) y la implementación de HTTPS para cifrar la comunicación. Para mitigar ataques como la inyección SQL, se utilizarían consultas parametrizadas u ORMs que validen y saneticen todas las entradas, evitando la ejecución de código malicioso en la base de datos. Además, para prevenir ataques CSRF se incorporarían tokens específicos y se configurarían cabeceras de seguridad (como SameSite en cookies), mientras que la validación, sanitización y uso de políticas de seguridad de contenido (CSP) ayudarían a evitar vulnerabilidades XSS. Complementariamente, la implementación de un firewall de aplicaciones web (WAF), junto con auditorías de seguridad y pruebas de penetración, aseguraría una protección integral frente a otras amenazas comunes.

1. **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?

Implementaría un patrón de transacciones distribuidas, como el Saga, para garantizar la consistencia en la transferencia entre cuentas de diferentes servicios. Con este enfoque, cada operación local se ejecuta de manera secuencial y, en caso de fallo en alguno de los pasos, se activa una transacción compensatoria que revierte los cambios realizados hasta el momento, asegurando que la transferencia se complete de forma atómica. Paralelamente, establecería mecanismos de manejo de errores robustos, como reintentos controlados, circuit breakers y un monitoreo centralizado, para detectar y actuar de inmediato ante cualquier anomalía en el flujo. Esta estrategia integral permite mantener la integridad de los datos y gestionar eficazmente los errores en un entorno distribuido.

1. **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?

Diseñaría una suite de pruebas integral que combine pruebas unitarias y de integración para cubrir tanto la lógica interna como la interacción entre componentes. En el ámbito de las pruebas unitarias, utilizaría frameworks como JUnit o NUnit junto con herramientas de mocking (por ejemplo, Mockito o sinon) para aislar cada componente y validar el comportamiento de funciones y servicios específicos, garantizando que cada módulo responda correctamente a distintos escenarios, incluidos los casos límite y errores esperados. Por otro lado, para las pruebas de integración implementaría herramientas como REST Assured o Supertest para simular peticiones HTTP reales y verificar la correcta interacción entre la API, la base de datos y otros servicios externos; además, emplearía entornos de prueba basados en contenedores (como Docker) para replicar configuraciones de producción y asegurar la coherencia en la ejecución. Finalmente, integraría estas pruebas en un pipeline de CI/CD (utilizando Jenkins, GitLab CI, etc.) y complementaría con herramientas de análisis de cobertura (como JaCoCo o Istanbul) para monitorear y mejorar la calidad del código.

1. **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?

Gestionaría el estado de la aplicación utilizando un enfoque centralizado, por ejemplo, mediante Redux o Context API en frameworks como React, para asegurar que todas las vistas y componentes accedan a un estado coherente y actualizado; mientras que para la autenticación implementaría un sistema basado en tokens (como JWT), almacenados de manera segura en cookies con atributos HttpOnly y SameSite, lo que previene vulnerabilidades XSS y CSRF. Además, establecería mecanismos de renovación automática de tokens y validaciones en cada interacción con el backend a través de HTTPS, garantizando que los datos sensibles, como los saldos de las cuentas, se mantengan sincronizados y protegidos a lo largo de toda la sesión.