Ejercicio Práctico

Nombre: Jonathan Santos P.

Cargo: Arquitecto de Soluciones

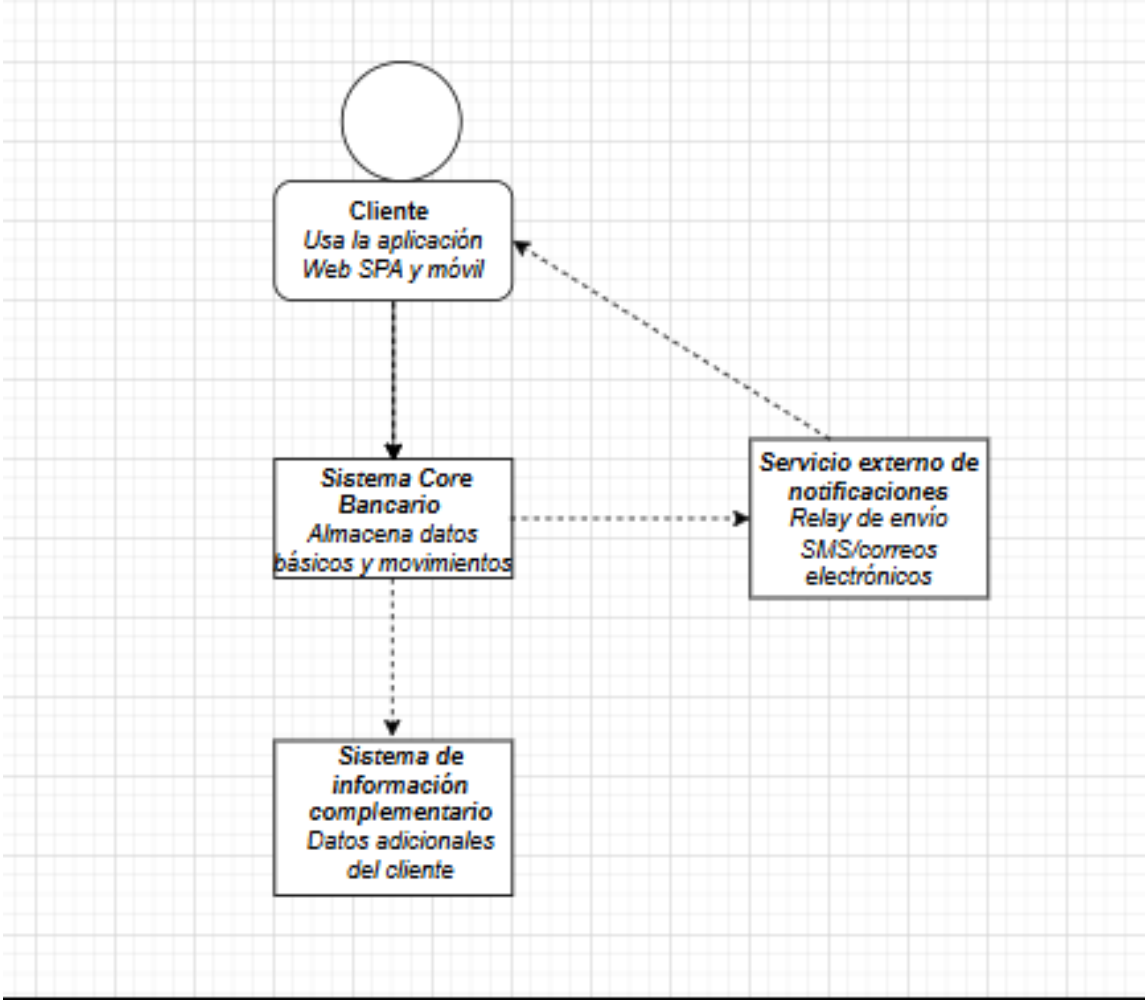
Fecha: 24/03/2024

Resolución:

1) **Diagramas**

- **Diagrama de Contexto:**

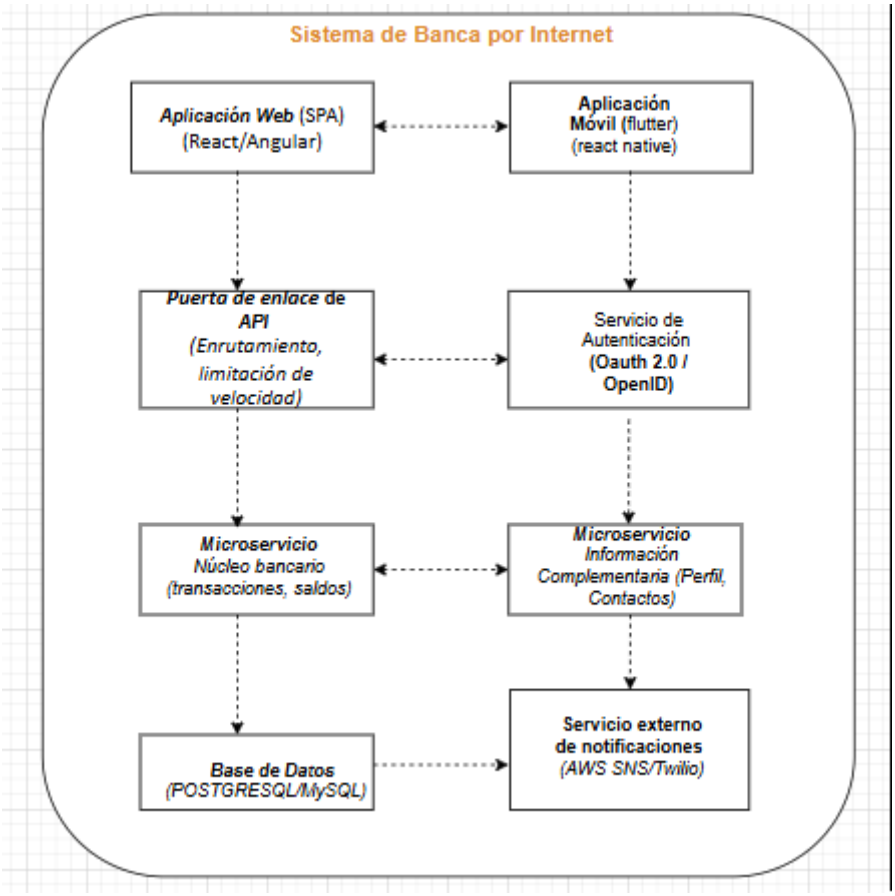
Descripción. - Muestra las interacciones entre el sistema de Banca por Internet y los actores externos.



Justificación. – Este diagrama delimita el alcance del sistema y sus interacciones externas, es clave para entender dependencias y flujos de información.

- **Diagrama de Contenedores:**

Descripción. – Representa las tecnologías y plataforma donde se ejecuta el sistema.



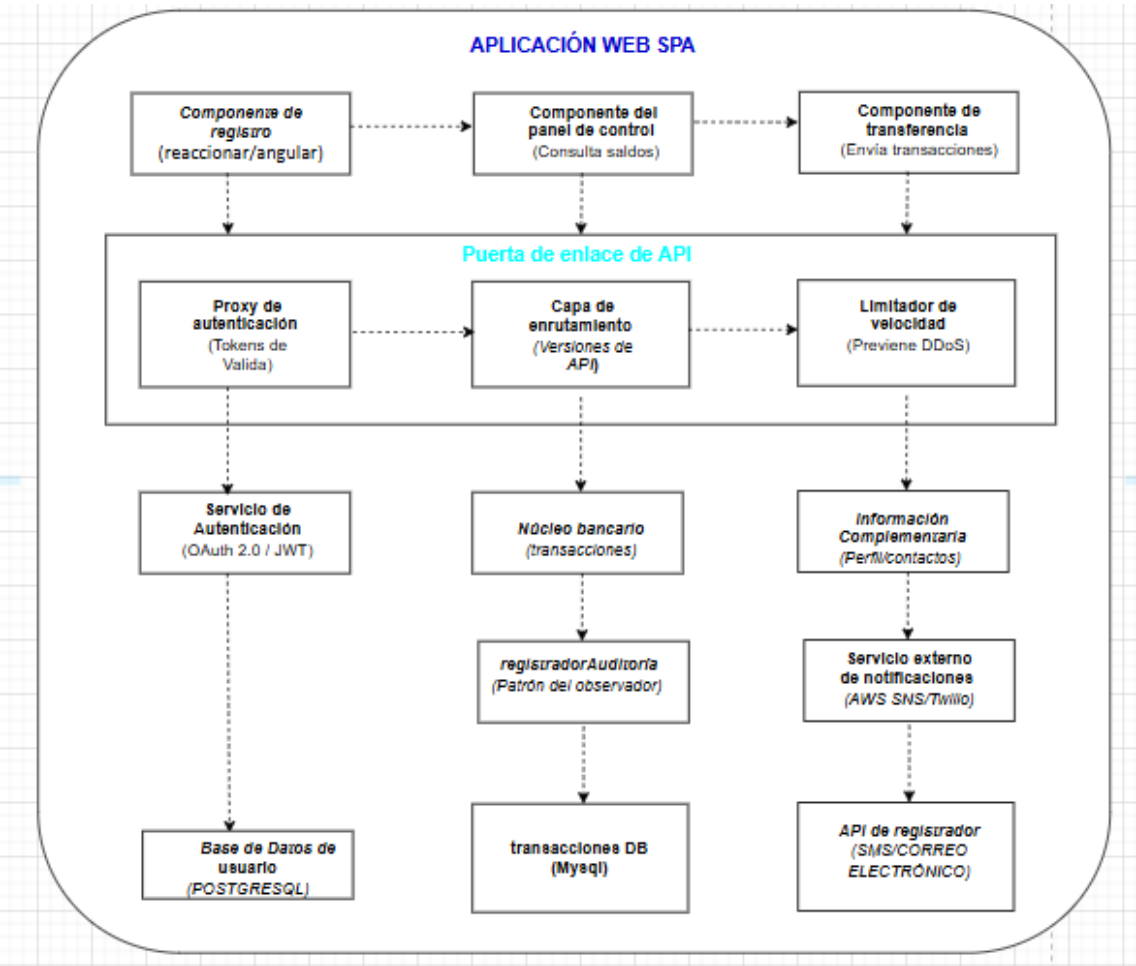
Justificación. –

Flutter (alto rendimiento) /React Native (flexibilidad en desarrollo): Son frameworks multiplataforma que reducen costos de desarrollo y mantienen consistencia en IOS/Android

Oauth 2.0: Es el estándar para autenticación en banca, seguro y escalable. Alternativas: OpenID Connect (para identidad) o SAML (empresarial)

- **Diagrama de Componentes:**

Descripción. – Detalla los módulos internos y sus relaciones.



Justificación. –

1. Frontend (aplicación web y móvil)
   . Componente de registro
   - Justificación: encapsula la autenticación mediante OAuth 2.0, permitiendo integración con proveedores (google, apple, bancos federados)
   - Alternativas: Webauthn para autenticación sin contraseña (huella/rostro)

   . Componente del panel de control
   - Justificación: muestra información en tiempo real mediante llamadas al core banking, siguiendo el principio de single responsibility (SOLID)

   . Componente de transferencia
   - Justificación: implementa el patrón command para deshacer transacciones (ej:reverso por error)

2. API Gateway
   . Proxy de autenticación
   - Justificación: Centraliza la validación de tokens JWT, evitando redundancia en cada microservicio (Gateway offloading).

. Capa de enrutamiento
    - Justificación: permite versionado de APIs para compatibilidad hacia atras

. Limitador de velocidad
    - Justificación: Protege contra ataques ddos y abuso de APIs.

3. Microservicios backend

. Servicio de autenticación
    - Justificación: OAuth 2.0 + openid connect es el estándar en Banca por su seguridad y escalabilidad
    - Alternativas: SAML para integración con sistemas empresariales, ej: Active directory

. Core bancario
    - Justificación: separación clara de responsabilidades (domain-driven design)

. Info complementaria
    - Justificación: almacena datos no críticos (ej. Preferencias de contacto), permitiendo escalamiento independiente

4. Servicios auxiliares

. Registrador auditoría
    - Patron de observador:
        - Ventaja: desacopla el registro de auditoría de la lógica principal.
        - Alternativa: Decorator Pattern para envolver operaciones con registro
    - Servicio de notificación:
        - Justificación: Usa AWS SNS (para SMS/email) por su alta disponibilidad
        - Alternativa: Websockets para notificaciones en tiempo real

5. Bases de datos

. Base de datos de usuario (PostgreSQL)
    - Justificación: ACID-compliance para operaciones críticas (ej. Bloqueo de cuentas)

. Base de datos transaccional (MySQL)
    - Justificación: optimizado para escrituras rápidas (transacciones/secuencia)

. APIs externas (Twilio/AWS SNS)
    - Justificación: delegación de envíos masivos a servicios especializados.

2) Conclusión y recomendación

Los diagramas realizados en este ejercicio proporcionan una visión de la arquitectura destacando lo siguiente:

- Para poder obtener un frontend unificado de la aplicación se debe utilizar frameworks multiplataforma flutter/react native
- Para poder obtener una autenticación robusta se aconseja configurar Oauth 2.0 + MFA (autenticación multifactor)
- Para poder tener una auditoría trazable se recomienda implementar un patrón de observador
- Se recomienda la integración con sistemas legacy (core banking) mediante APIs
- Para tener un escenario de alta disponibilidad, contingencia, replicación y restauración se sugiere contar con las aplicaciones de nuestro sistema en un escenario de sitios alternos, ya sea en un DCP- DCA para escenarios on premise, o en escenarios de contingencia replicados hacia la nube ya sea en Azure, AWS o GCP
- Para poder contar con bajas latencias en el acceso a los aplicativos web se recomiendan tener enlaces óptimos entre origen y destino o si están estos publicados en la nube mediante un express route ya que se cuenta con el presupuesto.
- Se recomienda aplicar hardening en los servidores/servicios/aplicativos y demás elementos que intervengan en el uso adecuado de nuestro sistema, para tener un escenario seguro y libre de ataques
- Se recomiendan el uso y adecuación de herramientas de monitoreo y sus respectivas alertas, como por ejemplo utilizando azure sentinel y la configuración de los diferentes umbrales de las diferentes alertas
- Todas las aplicaciones que se encuentren en ambiente on-premise o de ser el caso las aplicaciones de nuestro sistema para este ejercicio se recomienda realizar un tema de optimización de estas aplicaciones para que en un futuro muy cercano puedan ser migradas hacia la nube (independientemente de ella) y con esto optimizar temas de costos y demás
- Se recomienda tener herramientas de finops para poder optimizar los costos operativos que puedan ocacionar nuestra aplicación de este ejercicio y el resto de aplicativos de nuestro core de negocio.