Problema:

Usted ha sido contratado por una entidad llamada BP como arquitecto de soluciones para diseñar un sistema de banca por internet, en este sistema los usuarios podrán acceder al histórico de sus movimientos, realizar transferencias y pagos entre cuentas propias e interbancarias.

Toda la información referente al cliente se tomará de 2 sistemas, una plataforma Core que contiene información básica de cliente, movimientos, productos y un sistema independiente que complementa la información del cliente cuando los datos se requieren en detalle.

Debido a que la norma exige que los usuarios sean notificados sobre los movimientos realizados, el sistema utilizará sistemas externos o propios de envío de notificaciones, mínimo 2.

Este sistema contará con 2 aplicaciones en el Front, una SPA y una Aplicación móvil desarrollada en un Framework multiplataforma. (Mencione 2 opciones y justifique el porqué de su elección).

Ambas aplicaciones autenticarán a los usuarios mediante un servicio que usa el estándar Oauth2.0, para el cual no requiere implementar toda la lógica, ya que la compañía cuenta con un producto que puede ser configurado para este fin; sin embargo, debe dar recomendaciones sobre cuál es el mejor flujo de autenticación que se debería usar según el estándar.

Tenga en cuenta que el sistema de Onboarding para nuevos clientes en la aplicación móvil usa reconocimiento facial, por tanto, su arquitectura deberá considerarlo como parte del flujo de autorización y autenticación, a partir del Onboarding el nuevo usuario podrá ingresar al sistema mediante usuario y clave, huella o algún otro método especifique alguno de los anteriores dentro de su arquitectura, también puede recomendar herramientas de industria que realicen estas tareas y robustezca su aplicación.

El sistema utiliza una base de datos de auditoría que registra todas las acciones del cliente y cuenta con un mecanismo de persistencia de información para clientes frecuentes, para este caso proponga una alternativa basada en patrones de diseño que relacione los componentes que deberían interactuar para conseguir el objetivo.

Para obtener los datos del cliente el sistema pasa por una capa de integración compuesta por un api Gateway y consume los servicios necesarios de acuerdo con el tipo de transacción, inicialmente usted cuenta con 3 servicios principales, consulta de datos básicos, consulta de movimientos y transferencias que realiza llamados a servicios externos dependiendo del tipo, si considera que debería agregar más servicios para mejorar el rendimiento de su arquitectura o agregar más servicios para mejorar la repuesta de información a sus clientes, es libre de hacerlo.

Respuesta Completa para la Arquitectura del Sistema de Banca por Internet

A continuación, presentaré una arquitectura detallada que responde a los puntos mencionados en el enunciado. Utilizaré patrones de diseño, tecnologías adecuadas y buenas prácticas para cumplir con todos los requisitos del sistema de banca por internet.

---

1. Aplicaciones Front-End: SPA y Aplicación Móvil

Para las aplicaciones front-end, se proponen las siguientes opciones para el desarrollo de la **Aplicación Móvil**:

Opciones de Framework para la Aplicación Móvil:

1. **Flutter (Google)**
    - **Justificación**: Flutter es un framework de desarrollo de aplicaciones móviles multiplataforma. Su ventaja principal es la capacidad de compilar en nativas para iOS y Android, lo que proporciona una excelente performance. Además, permite una experiencia visual consistente y permite usar un único código base para ambas plataformas, reduciendo costos y tiempos de desarrollo.
    - **Escalabilidad y Mantenimiento**: Flutter es ideal para aplicaciones que necesitan mantener una experiencia fluida y escalabilidad en ambas plataformas sin perder el rendimiento.
2. **React Native (Facebook)**
    - **Justificación**: React Native es otro framework de desarrollo multiplataforma que permite desarrollar aplicaciones nativas para iOS y Android. Su ventaja es la fuerte integración con JavaScript y React, lo que permite a los equipos de desarrollo web con experiencia en JavaScript adaptarse rápidamente. Además, es compatible con plugins nativos, lo que lo hace muy flexible.
    - **Escalabilidad y Mantenimiento**: También tiene una gran comunidad y soporte a largo plazo, con una extensa cantidad de paquetes y librerías que permiten ampliar las capacidades de la aplicación.

---

2. Flujo de Autenticación y Autorización (OAuth2.0)

Para la autenticación de los usuarios mediante OAuth2.0, recomiendo implementar el siguiente flujo:

Flujo de Autenticación recomendado:

1. **Flujo de Autenticación Code Grant con PKCE (Proof Key for Code Exchange)**:
    - **Justificación**: Este flujo es el más seguro para aplicaciones móviles y SPA, ya que evita exponer el token de acceso en el cliente, lo cual es importante para proteger la seguridad de los usuarios.
    - **Proceso**:
        1. El cliente (ya sea SPA o la aplicación móvil) solicita el código de autorización con un code_challenge (hash seguro).
        2. El servidor de autorización devuelve un código de autorización.
        3. El cliente intercambia ese código por un access_token enviando el code_verifier.
        4. El servidor de autorización verifica el code_verifier y emite el token.
2. **Integración con biometría (Reconocimiento Facial y Huella)**:

- Durante el **Onboarding**, la aplicación debe integrar un flujo de autenticación biométrica. Se recomienda usar APIs como **Apple Face ID**, **Google Face Recognition**, **Android Biometric API** y **Microsoft Azure Face API**.
- Después del onboarding, el sistema debe permitir la autenticación con huella dactilar o reconocimiento facial (dependiendo de lo que el usuario prefiera). Estos métodos son muy adecuados para dispositivos móviles.

---

3. Arquitectura de Integración y API Gateway

En esta arquitectura, la capa de integración estará conformada por un **API Gateway** que orquesta las solicitudes a los servicios backend según el tipo de transacción. Dado el requerimiento, la arquitectura podría incluir los siguientes componentes:

Componentes Principales de la Arquitectura:

1. **API Gateway:**
   - El **API Gateway** maneja todas las solicitudes de los usuarios, proporcionando una única entrada al sistema. Este se encargará de enrutar las solicitudes hacia los servicios correspondientes (consultar datos, movimientos, transferencias).
   - Además, el API Gateway se encargará de manejar la seguridad (OAuth2.0) y balanceo de carga para mejorar la performance y disponibilidad.
2. **Microservicios de Back-End**:
   - **Consulta de Datos Básicos**: Servicio que consulta la base de datos de clientes y productos del Core Banking.
   - **Consulta de Movimientos**: Servicio que consulta los movimientos históricos y saldos de las cuentas del cliente.
   - **Servicio de Transferencias**: Servicio que gestiona las transferencias, tanto entre cuentas propias como interbancarias. Este servicio hará llamadas a APIs externas (bancos de destino).
   - **Notificaciones**: Servicio para enviar notificaciones por diferentes canales (email, SMS, push notifications).
   - **Auditoría**: Servicio que gestiona la base de datos de auditoría. Este servicio registra todas las acciones realizadas por el cliente.
3. **API de Integración Externa (Bancos de Destino)**:
   - Servicios para ejecutar transferencias interbancarias, pagos de servicios, etc.

Modelo de Componentes C4:

1. **Diagrama de Contexto C4**:
   En este diagrama, los usuarios (clientes) interactúan con la aplicación móvil y la SPA. Ambas aplicaciones están conectadas a un API Gateway, que se comunica con los microservicios de backend para realizar las consultas necesarias (movimientos, datos

básicos, etc.). Además, el sistema interactúa con servicios de terceros para realizar transferencias interbancarias.

    - **Usuarios ↔ SPA ↔ API Gateway ↔ Microservicios ↔ Core Banking / Servicios Externos**

2. **Diagrama de Contenedores C4**:
    - **SPA y App Móvil**: Contienen la lógica de negocio en el cliente (con React o Flutter) y las interacciones de la interfaz de usuario.
    - **API Gateway**: Enrutador central para las peticiones hacia los servicios backend.
    - **Microservicios de Backend**: Servicios especializados en gestionar datos de clientes, movimientos, transferencias y auditoría.
    - **Base de Datos de Auditoría**: Se registra toda acción relevante realizada por el cliente.
    - **Servicios Externos (Interbancarios)**: APIs externas para manejar transferencias a otras entidades bancarias.

---

4. Persistencia y Auditoría

Patrón de Diseño:

1. **Patrón Repository**: Utilizar este patrón para abstraer el acceso a los datos, haciendo que los servicios puedan interactuar con la base de datos de manera más flexible y manteniendo la lógica de negocios limpia.
    - **Ejemplo**: Crear un TransactionRepository que se encargue de la persistencia de las transacciones del cliente y otro para manejar los movimientos históricos de las cuentas.
2. **Patrón Singleton**: Utilizar este patrón para los servicios de auditoría, ya que las acciones de los usuarios deben ser registradas de manera centralizada y uniforme.
3. **Patrón Event Sourcing**: Considerar usar event sourcing para el sistema de auditoría, lo que permitirá registrar todas las acciones como eventos, con lo que obtendrás un historial completo y detallado de las interacciones del usuario con el sistema.

---

5. Recomendaciones Adicionales para la Arquitectura

1. **Cacheo de Datos**: Implementar un sistema de cacheo para mejorar el rendimiento, especialmente para consultas de movimientos históricos. Herramientas como **Redis** pueden ser útiles.
2. **Escalabilidad**: Considerar el uso de **microservicios** para permitir que el sistema crezca y escale con facilidad. Usar Kubernetes para la orquestación de contenedores y asegurar la disponibilidad.

3. **Manejo de Errores y Resiliencia**: Implementar patrones como **Circuit Breaker** y **Retry** para asegurar la resiliencia en los microservicios, especialmente cuando se consumen APIs externas.
4. **Seguridad**: Asegurarse de que todas las comunicaciones entre el front-end, API Gateway y los servicios backend sean cifradas mediante HTTPS/TLS. Además, se deben emplear mecanismos de autenticación y autorización robustos (OAuth2.0, MFA).

Conclusión

Con este enfoque, estaríamos cubriendo tanto los aspectos técnicos de la arquitectura (microservicios, API Gateway, OAuth2.0, etc.) como los aspectos de seguridad y experiencia de usuario (autenticación biométrica, escalabilidad, etc.). Esto proporciona un sistema de banca por internet robusto, seguro y escalable que cumple con los requisitos de la entidad BP y las normativas legales aplicables.