JOSE LUIS ALVEAR
ARQUITECTO DE SOLUCIONES

Información general - Propuesta Solución de Sistema Bancario.

Tabla de contenido

ANTECENDETES:

Una entidad bancaria llamada BP quiere contratar un servicio 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.

TOBE SOLUTION. -

Se diseña una aplicación web para gestión bancaria que se conectara con 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. Tendrá un front-end SPA y un front-end para dispositivos Móvil.

Arquitectura en la NUBE:

Dado que el sistema tendrá alta transaccionalidad por minuto, se considera escalable al ser una APP nueva y altamente complejo por volumen de transacciones y datos de cuentas y montos, incluso puede llegar a evolucionar según regulaciones de gobiernos y sistema financiero bancario, entonces se considera una Arquitectura en la NUBE se propone las siguientes opciones:

1. AWS Lambda: Configurado en Amazon que permite ejecutar código en respuesta a eventos, solicitudes de API o cambios en la base de datos. Por ser altamente escalable y ofrecer una alta disponibilidad y tolerancia a fallos, también se podrá hacer de integraciones con otros servicios de AWS.

2. Azure Application Gateway: SE puede aprovechar el servicio de balanceo de carga de Microsoft para distribuir el tráfico de red entre múltiples instancias de aplicaciones. También nos ofrece una alta disponibilidad y tolerancia a fallos, y podemos utilizar opciones avanzadas de seguridad, como la protección contra ataques DDoS.

Frameworks Multiplataforma:

Dado que tendrá 2 aplicaciones en el Front, una SPA y una Aplicación móvil. Se recomienda usar Frameworks multiplataforma que nos ofrezcan más componentes y ventajas sobre las arquitecturas seleccionadas arriba, entonces se propone los siguientes:

1. React Native (AWS): Nos permitirá desarrollar aplicaciones móviles para Android y iOS utilizando JavaScript y React. Nos puede ofrecer alta velocidad de desarrollo, gran cantidad de componentes y bibliotecas disponibles que podemos usar y en la comunidad podemos solventar issues.

2. Xamarin (Azure): Nos permitirá desarrollar aplicaciones móviles para Android, iOS y Windows utilizando C# y .NET. Xamarin. Podemos tomar ventaja de una alta calidad y rendimiento de las aplicaciones, también una cantidad importante de herramientas y bibliotecas disponibles en la comunidad.

Servicios de Envió Notificaciones:

Siguiendo las Arquitecturas seleccionadas de Plataforma NUBE y framewroks se recomienda utilizar servicios externos para envío de Notificaciones. Se sugiere los siguientes:

1. Amazon Simple Notification Service (SNS): Al tener un servicio de mensajería en la nube de Amazon que permite enviar notificaciones a dispositivos móviles, correo electrónico y otros servicios de AWS. Podemos usar SNS altamente escalable y con alta disponibilidad, también podemos usar opciones avanzadas de filtrado y enrutamiento de mensajes.

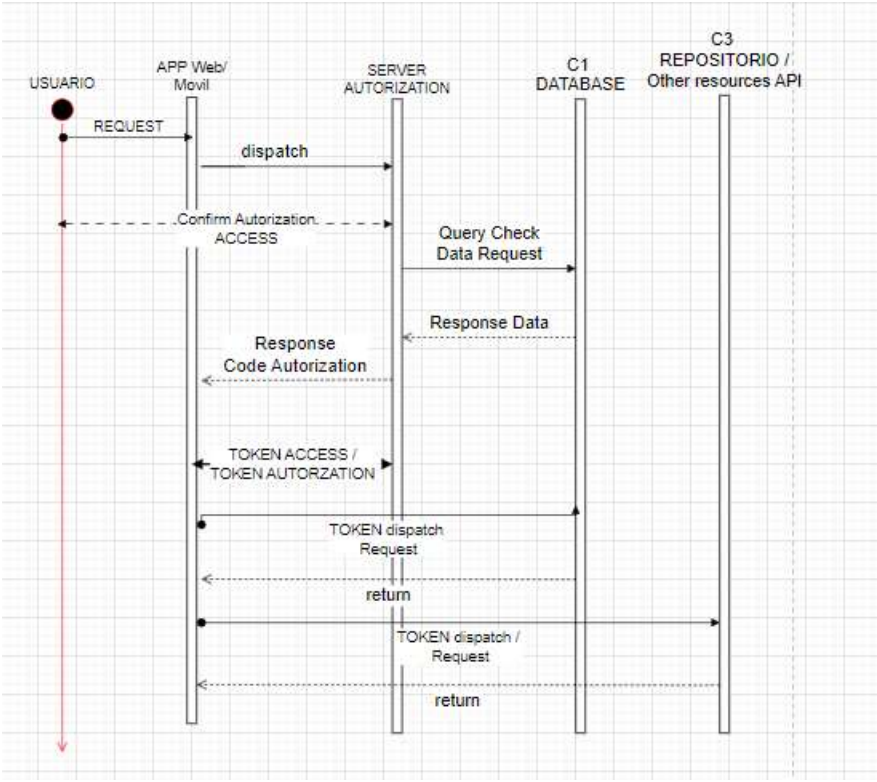
2. Firebase Cloud Messaging (FCM): Por el lado de Microsoft podemos usar un servicio de mensajería en la nube de Google fácil de integrar que envía notificaciones a dispositivos móviles Android, iOS y web. FCM es y ofrece una alta tasa de entrega de mensajes, así como opciones avanzadas de segmentación y personalización de mensajes.

Se podría utilizar una propio de la entidad Financiera SI es fácil de integrar y ofrece las mismas características de las opciones mencionadas, esto sería para abaratar costos de la solución.

- Estos elementos de infraestructura nos garantizan baja latencia, alta disponibilidad y tolerancia a fallos, seguridad y monitoreo

Autenticación y Autorización:

Debido a que ambas aplicaciones (SPA y móvil) integraran la autenticación mediante servicio que utiliza el estándar OAuth 2.0 que es una forma segura y eficiente de permitir que las aplicaciones accedan a los datos de los usuarios sin compartir sus credenciales de inicio de sesión. Se propone el recomienda utilizar el flujo de autorización de código de autorización (Authorization Code Grant) de OAuth 2.0. Se propone implementar los siguientes pasos:



1. El usuario inicia sesión en la APP de banca.

2. Para autenticarse La aplicación redirige al usuario a un servidor de autorización.

3. El servidor de autorización solicita al usuario que autorice la APP de banca para que pueda a acceder a sus datos.

4. Si el usuario autoriza, entonces el servicio de autorización emite un código de autorización a la APP.

5. Entonces la APP intercambia el código de autorización por un token de acceso y un token de actualización con el servidor de autorización.

6. Finalmente la aplicación de banca utiliza el token de acceso para acceder a los recursos protegidos en nombre del usuario.

NOTA: Si bien es cierto en la actualizada han proliferado ataques de Hacking y Phishing se recomendaría la implementación de OAuth 2.0 de forma metódica y siguiendo las mejores prácticas de seguridad para proteger los datos de los usuarios. De esta forma se reduciría la

probabilidad de hacking porque el token de acceso solo se comparte entre la aplicación bancaria SPA y el servidor de autorización, y no se comparte con el usuario o con otros servicios. También se puede usar el token de actualización para obtener un nuevo token de acceso sin necesidad de que el usuario inicie sesión nuevamente.

DIAGRAMAR MODELO C4 DEL SISTEMA. -

Diagrama de Contexto

El diagrama de contexto muestra el sistema como un todo -se compone de una aplicación móvil y una API que se comunica con los servicios externos de notificaciones y banca.

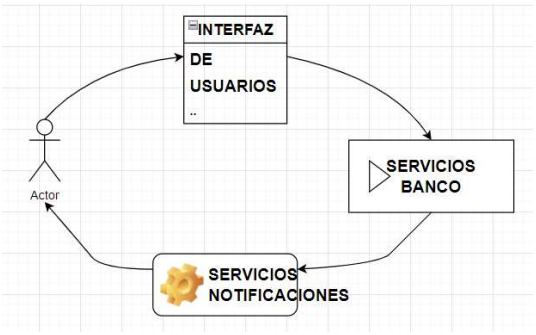


Diagrama de Contenedores

El diagrama de contenedores muestra los componentes principales del sistema y sus relaciones. El sistema se compone de una aplicación móvil, una API y una base de datos. La aplicación móvil se comunica con la API a través de una interfaz de usuario, y la API se comunica con la base de datos a través de una capa de persistencia.

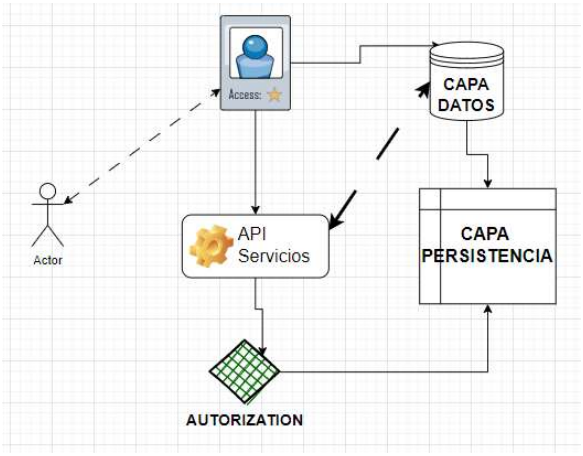
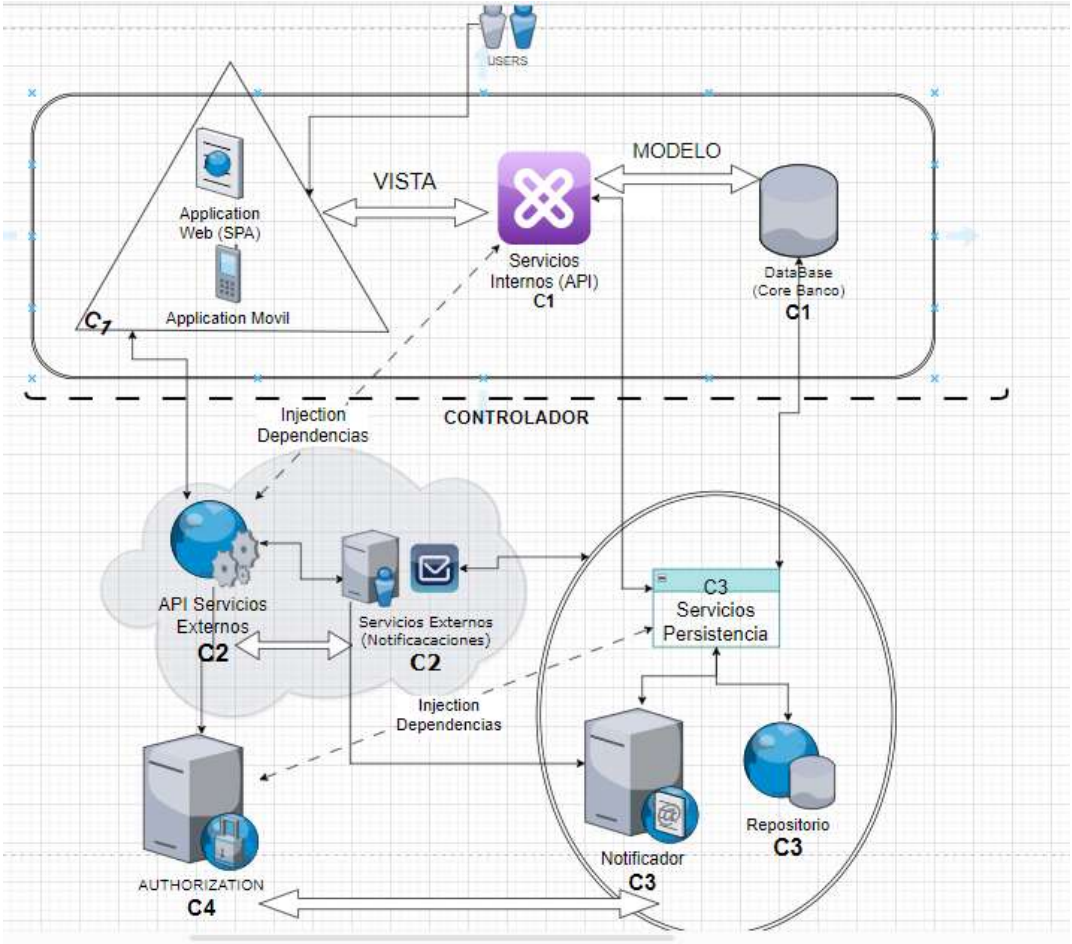


Diagrama de Componentes

El diagrama de componentes muestra los componentes internos de la API y su relación con la capa de persistencia. La API se compone de varios componentes, incluyendo un controlador de autenticación, un controlador de notificaciones, un controlador de banca y un controlador de

usuarios. Cada controlador se comunica con la capa de persistencia a través de un repositorio correspondiente.



Leyenda:

- C1 representa el contenedor de Aplicación principal, servicios internos y la base de datos.
- C2 representa el contenedor de la API servicios externos.
- C3 representa el contenedor de los servicios persistencia.
- C4 representa el controlador de autenticación en la API.
- La aplicación móvil y los servicios externos se representan como usuarios externos en el diagrama de contexto (no tienen componentes asociados).

Patrones de Arquitectura

- **Patrón Modelo-Vista-Controlador (MVC)*:** Se utilizará para separar la lógica de la aplicación móvil en tres componentes principales: el modelo, la vista y el controlador. El modelo representa los datos y la lógica de negocio, la vista representa la interfaz de usuario y el controlador maneja la interacción entre el modelo y la vista.
- **Patrón de Inyección de Dependencias (DI)*:** Se utiliza para permitir la creación de objetos con sus dependencias resueltas en tiempo de ejecución. Esto permite una mayor flexibilidad y modularidad en el diseño de la API.

ONBOARDING:

En la misma línea base utilizando el estándar OAuth 2.0 para la autenticación, se ara integración con sistema de OnBoarding para nuevos usuarios que permita registrarse y autenticarse utilizando métodos como usuario y contraseña, preguntas de perfil BIOMETRICO y reconocimiento facial (para el proceso de reconocimiento facial del OnBoarding incluirá la verificación de la identidad del usuario mediante la solicitud de documentos de identificación, como una cédula o pasaporte, y la validación de la información proporcionada por el usuario por correo electrónico).

Una vez que el usuario ha sido verificado y registrado, se permitirá autenticarse con el servicio de autorización y el estándar de autenticación OAuth 2.0. Debido a que la seguridad es fundamental en una aplicación bancaria, se deben tener en cuenta para implementar medidas de seguridad adicionales y proteger los datos de los usuarios y prevenir el fraude, como la autenticación de dos factores (2FA) o la detección de comportamiento sospechoso.

AUDITORIA y PERSISTENCIA

Siguiendo las directrices planteadas la solución tendrá una base de datos de auditoría como medida de seguridad adicional. Se utilizarán patrones de diseño para la base de auditoria, así como el patrón de arquitectura de microservicios para el servicio de auditoria y persistencia.

Utilizando patrón de arquitectura de microservicios, los servicios (incluido Auditoria) se dividirá en pequeños servicios independientes, se encargará de registrar todas las acciones del cliente y almacenarlas en una base de datos dedicada. Este servicio estará comunicado con otros servicios de la aplicación entre sí a través de una interfaz API para obtener información relevante, como el ID del cliente o la acción realizada.

Además, se utilizará un patrón de diseño de persistencia de información para clientes frecuentes, como el patrón de caché de datos. Nos permitirá guardará en caché los datos de los clientes frecuentes en memoria o en una base de datos en memoria, lo que permite acceder a ellos de forma más rápida y reducir la carga en la base de datos principal.

Utilizando patrones de diseño como la arquitectura de microservicios y el patrón de caché de datos nos permiten una mayor escalabilidad, flexibilidad y seguridad en la aplicación.

MODELO DE CAPAS DE ARQUITECTURA:

El modelo de arquitectura para esta aplicación bancaria con 2 front-end (Web/Móvil) incluirá una capa de integración con una API Gateway y servicios externos para obtener datos del cliente:

1. Capa de presentación: Esta capa se encarga de la interfaz de usuario de la aplicación bancaria para la parte del front-end, que se comunica con la capa de integración a través de una API.

2. Capa de integración: Esta capa se compone de una API Gateway que actúa como punto de entrada para la aplicación bancaria y se encarga de enrutar las solicitudes a los servicios correspondientes. Los servicios principales son:

- Servicio de consulta de datos básicos: Este servicio se encarga de obtener los datos básicos del cliente, como su nombre, dirección y número de identificación, a partir de un servicio externo de identificación.

- Servicio de consulta de movimientos: Este servicio se encarga de obtener los movimientos del cliente, como sus transacciones y saldos, a partir de un servicio externo de banca.

- Servicio de transferencias: Este servicio se encarga de realizar transferencias de fondos entre cuentas del cliente, utilizando un servicio externo de pagos.

3. Capa de servicios externos: Esta capa se compone de los servicios externos de identificación, banca y pagos, que se encargan de proporcionar los datos necesarios para los servicios de la capa de integración.

4. Capa de persistencia: Esta capa se encarga de almacenar los datos de la aplicación bancaria en una base de datos, como los datos de los clientes y las transacciones realizadas.

Este modelo de arquitectura permite una mayor flexibilidad y escalabilidad en la aplicación bancaria con 2 front-end (Web/Móvil), ya que los servicios se pueden actualizar y escalar de forma independiente. Además, la capa de integración con la API Gateway proporciona una capa de seguridad adicional para proteger los datos del cliente y prevenir el acceso no autorizado.

MANEJO DE COSTOS:

Al lanzar un sistema de estas características y envergadura hay que tomar en cuenta las perspectivas de Retorno de Inversión y el alcance de usuarios objetivo que va a tener nuestra aplicación. Para esta propuesta de sistema voy a manejar un alcance para 0.5 millones de usuarios como para una FASE 1ra. Dando a conocer la aplicación al público. Con este objetivo en mente el costo de lanzar un sistema con capacidad puede variar significativamente dependiendo de otros varios factores, como la complejidad del sistema, la cantidad de funcionalidades, la elección de la infraestructura en la nube, la cantidad de desarrolladores involucrados y el tiempo de desarrollo.

A modo de referencia, se puedo proporcionar un presupuesto aproximado basado en algunos supuestos (aproximaciones):

- Desarrollo de la aplicación móvil: $50,000
- Desarrollo de la API y la capa de integración: $30,000
- Desarrollo de la capa de persistencia: $25,000
- Infraestructura en la nube (AWS o Azure): $10,000 por mes
- Costos adicionales (monitoreo, seguridad, pruebas, etc.): $20,000

Los costos reales pueden variar significativamente dependiendo de los factores mencionados anteriormente. Es importante tener en cuenta que el costo de lanzar un sistema no es el único factor para considerar, ya que también se deben tener en cuenta los costos de mantenimiento y actualización a largo plazo.

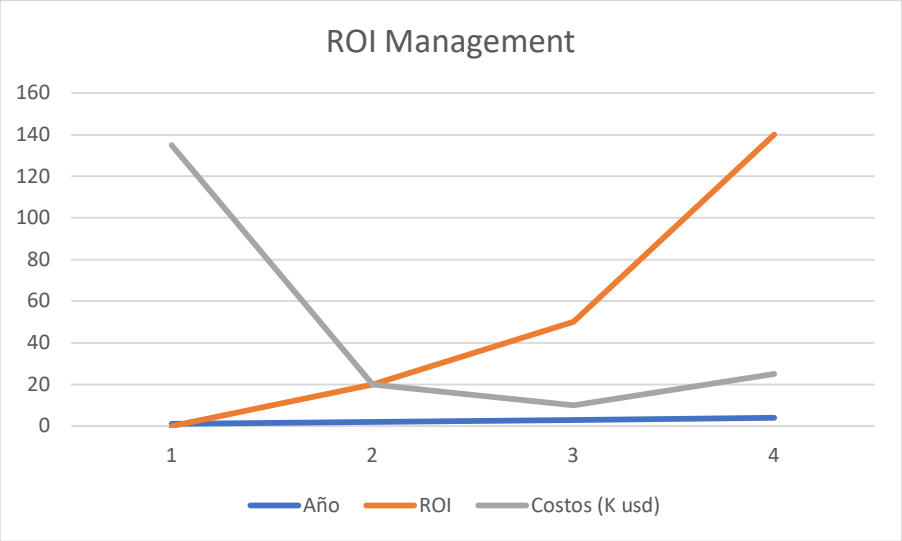
Cronograma:

Se esperaría que la implementación del sistema no lleve mas de 11 meses a partir de la firma y autorización de la proforma y pago de anticipo por adelantado.

Tarea N#	WorkBreak Dwon (high level)	Meses	Predecesoras
1	Infraestructura en la nube (AWS o Azure)	4	
2	Desarrollo de la aplicación móvil / Web	6	1
3	Desarrollo de la API y la capa de integración	6	1
4	Desarrollo de la capa de persistencia	5	2

ROI:

Se espera que el ROI pueda justificar los costos de la aplicación y refleje ingresos en aproximadamente 2 años.



Ciclo de Vida del Producto:

Este tipo de sistemas bancarios necesita un tiempo para la aceptación del mayor numero de usuarios promedio, seguido de eso se podrán identificar oportunidades de mejora a partir del 4to./5to. Año para implementar una nueva versión con características mejoradas.