Requerimiento

Banca Por Internet para BP

Versión 1.0

IDEA PRESENTADA: Banca Por Internet

Autor Aníbal Albán

DEVSU

Marzo / 2024

TABLA DE CONTENIDO

Entidad	Nombres y Apellidos	Cargo	Fecha
BP	NA	Líder de Tribu Canales	Marzo /2024
BP	NA	Product Owner Canal Banca Web.	Marzo /2024
BP	NA	Product Owner Canal Banca Móvil.	Marzo /2024
BP	Aníbal Albán	Arquitecto de Soluciones	Marzo /2024
Devsu	NA	Líder de Arquitectura	Marzo /2024

1. DATOS INFORMATIVOS

Requerimiento:	Solución Arquitectura BP
Originador del Requerimiento:	BP

2. ALCANCE DE LA SOLUCIÓN

En el sistema de Banca por Internet se requiere que los usuarios puedan acceder al histórico de movimientos, realizar transferencias y pagos entre cuentas propias e internet. Por otro lado, el sistema debe notificar a los usuarios de las transacciones mediante notificaciones mínimo 2 medios.

Es sistema tiene 2 aplicaciones un SPA y banca móvil. Las aplicaciones se autentican mediante Oauth 2.0.

El sistema cuenta con Onboarding y debe considerar el ingreso con usuario y clave, huella.

El sistema cuenta con una base de datos de auditoria para todas las acciones del cliente y debe persitir información para clientes frecuentes.

2.1. PROCESOS DE NEGOCIOS AFECTADOS

Procesos de Negocio	Detalle del impacto en el proceso
Gestión de Canales y Servicios	El área de canales debe generar valor al negocio con las nuevas características de la banca por internet

2.2. CANALES AFECTADOS

Canales	Detalle del impacto en el canal
Banca Movil	Se realizan modificaciones al proceso de consulta de movimientos, transferencia y pagos interbancoario en la Banca Movil.

2.3. APLICACIONES AFECTADAS

Aplicaciones	Detalle del impacto en la aplicación
Banca Web	Modificación para consulta de movimientos, pagos y transferencias entre cuantas propias e interbancarias.
Banca Móvil	Modificación para consulta de movimientos, pagos y transferencias entre cuantas propias e interbancarias.

2.4. AREAS AFECTADAS

Empresa	Área	Detalle del impacto en Áreas
BP	Canales	Preparación comunicaciones y campañas hacia el cliente
BP	Canales	Capacitación del especialista en el nuevo proceso.
BP	Operaciones	Capacitación del nuevo proceso

2.5. FUERA DE ALCANCE Y SUPOSICIONES

Se consideran los siguientes supuestos:

.

- ✓ De ser necesarios el acceso a aplicaciones del BP se deberá gestionar los permisos correspondientes
- ✓ BP participará con acompañamiento directo durante la fase de pruebas.

2.6. RESTRICCIONES

No.	Detalle de Restricción
	No se considera restricciones para la presente solución

3. SOLUCIÓN PROPUESTA

3.1. MARCOS DE TRABAJO TOGAF Y BIAN

Bajo la premisa de optimizar la arquitectura empresarial, al considerar BIAN, marco de referencia para la arquitectura empresarial de la industria financiera, se desacopla en diferentes capas.

Capa de Aplicaciones
Capa de procesos de negocio
Capa de eventos y reglas de negocio
Capa de actividades robóticas
Capa de APIS
Capa de servicios (SOA)
Capa de Documentos Electrónicos
Capa de Componentes

BIAN está basado en la arquitectura orienta a servicios y es un marco de referencia para la banca y apoya directamente a TOGAF. A continuación, se presenta el desacople de manera macro para Banca por Internet

Capa de Aplicaciones:

SPA, App Mobile

Capa de Procesos de Negocio:

Para el caso de transferencias interbancarias se requiere un proceso de autorización expos para montos mayores de un valor parametrizable, para lo cual se usuaria un BPM.

Capa de Eventos y Reglas de Negocio:

Servicios que manejan las reglas de negocio

Capa de Actividades Robóticas:

Para el caso de uso no se visión RPAs

Capa de Apis:

Plataformas de gestión de APIs Azure API Management.
Frameworks de desarrollo de APIs como Spring Boot.

Capa de Servicios (SOA - Service-Oriented Architecture/ Microservicios):

Plataformas de gestión de servicios IBM Integration Bus para acceso a los servicios CORE y satélites.
Frameworks de desarrollo de servicios como Spring Framework para microservicios

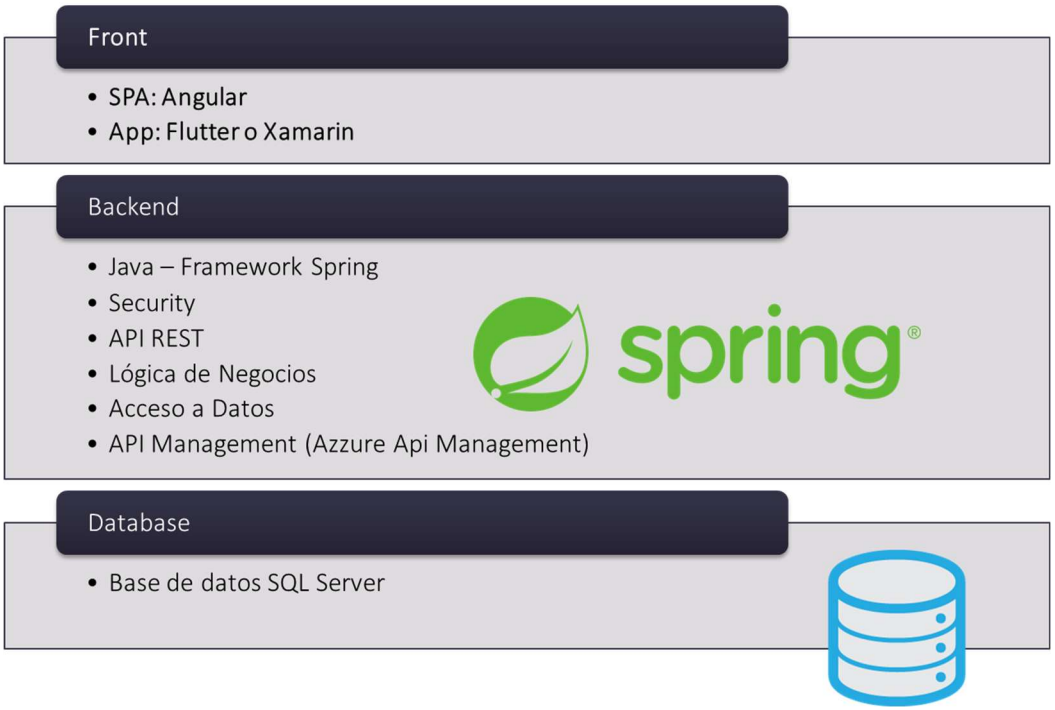
Capa de Documentos Electrónicos:

Para el caso de uso no se tiene

Capa de Componentes:

En esta capa están los procedimientos almacenados que se aplicarán sobre la Base de Datos SQLServer.
Podremos tener Core Banking, El sistema de Trasferencias, Sistemas de Autenticación IAM. Sistema de Correo Electrónico, Sistema Notificación SMS, Sistema de Autenticación Biométrico, Sistema Datos Adicionales, Bases de Datos.

3.2. INFORMACION TÉCNICA



A continuación se indica de manera general los criterios para considerar las tecnologías para el front-end, el back-end y la gestión de la base de datos, así como aspectos relacionados con la seguridad y la gestión de APIs.

Front-End

Para el desarrollo del front-end de la aplicación bancaria, se recomienda utilizar una arquitectura de una sola página (SPA), ya que proporciona una experiencia de usuario fluida y rápida al cargar solo una página web, lo que mejora la velocidad y la capacidad de respuesta de la aplicación. Angular es una excelente opción para implementar una SPA debido a su robustez, alto rendimiento y amplio soporte de la comunidad. Algunas de las razones para elegir Angular son:

- **Framework sólido y maduro:** Angular es un framework bien establecido respaldado por Google, que ofrece un amplio conjunto de características y herramientas para el desarrollo de aplicaciones web complejas.
- **TypeScript:** Angular se basa en TypeScript, un superconjunto de JavaScript que agrega características de programación orientada a objetos y detección de errores estáticos, lo que mejora la calidad y la mantenibilidad del código.
- **Soporte de la comunidad:** Angular cuenta con una gran comunidad de desarrolladores y una abundante cantidad de recursos, documentación y bibliotecas disponibles para facilitar el desarrollo y la resolución de problemas.

Aplicación Móvil

Para el desarrollo de la aplicación móvil, se consideran dos opciones: Flutter y Xamarin.

- **Flutter:** Flutter es un framework de código abierto desarrollado por Google para construir aplicaciones móviles nativas para iOS y Android desde una única base de código. Algunas razones para elegir Flutter son su rendimiento, su capacidad de personalización y su eficiencia en el desarrollo.
- **Xamarin:** Xamarin es un framework de desarrollo de aplicaciones móviles multiplataforma que permite desarrollar aplicaciones nativas para iOS y Android utilizando el lenguaje de programación C#. Es una excelente opción para empresas que ya tienen experiencia en el ecosistema Microsoft y prefieren utilizar C#.

Back-End

Para el desarrollo del back-end de la aplicación bancaria, se recomienda utilizar Java en combinación con el framework Spring.

- **Java:** Java es un lenguaje de programación robusto y ampliamente utilizado en la industria, conocido por su estabilidad, seguridad y portabilidad. Es una opción confiable para aplicaciones empresariales de alto rendimiento.
- **Framework Spring:** Spring es un framework de desarrollo de aplicaciones Java ampliamente adoptado en la industria debido a su modularidad, escalabilidad y facilidad de integración. Spring Boot, una extensión de Spring, simplifica el desarrollo de aplicaciones web y ofrece características como la configuración automática y el empaquetado de aplicaciones listas para producción.

Seguridad

Para garantizar la seguridad de la aplicación bancaria, se recomienda implementar una arquitectura de seguridad en capas que incluya:

- **Autenticación y autorización basadas en tokens JWT:** Utilizando bibliotecas como Spring Security en el back-end para gestionar la autenticación y autorización de usuarios.
- **Cifrado de datos sensibles:** Se deben cifrar los datos sensibles, como las contraseñas de los usuarios, antes de almacenarlos en la base de datos.
- **TLS/SSL:** Utilizar conexiones seguras a través de HTTPS para proteger la comunicación entre el cliente y el servidor.

- **Protección contra ataques de seguridad conocidos:** Implementar medidas de seguridad para protegerse contra vulnerabilidades comunes, como ataques de inyección SQL, XSS y CSRF.
- **Identity Access Managemet:** Un gestor de accesos e identidades cumplen con las normas de seguridad requeridas para la industria Bancaria podría utilizarse el de WSO2, Single Sing On de RedHat o el IAM de Azure.
- **Almacen de secretos:** los desarrolladores pueden almacenar de forma segura los secretos en Azure Key Vault

Gestión de la Base de Datos

Para la gestión de la base de datos, se recomienda utilizar SQL Server debido a su robustez, escalabilidad y soporte para transacciones ACID (Atomicidad, Consistencia, Aislamiento y Durabilidad).

Gestión de APIs

Para la gestión de APIs, se sugiere utilizar Azure API Management, una solución de Microsoft que proporciona herramientas para crear, publicar, gestionar y supervisar APIs de forma segura y escalable. Azure API Management ofrece características como la seguridad integrada, la escalabilidad automática y la supervisión detallada del rendimiento de las APIs.

3.3. NORMATIVAS Y ESTANDARES

En referencia a marcos de arquitectura de seguridad, Uso de OAuth2.0 con flujo de autorización y código de autorización debe considerarse dentro de la aplicación de Banca por Internet.

Si bien la arquitectura de seguridad de la información implica en su mayoría conceptos técnicos, no lograría su fin si no se tienen todos aquellos tópicos administrativos mencionados anteriormente y allí radica la diferencia y el punto clave para que funcione, no sirve poseer todas las condiciones técnicas necesarias sin la gestión y mantenimiento en el tiempo y esto sólo se consigue con una dinámica de seguridad continua que para este caso puede ser a través de la definición de un sistema de gestión de la seguridad de la información (SGSI), propuesto por la serie 27000 de la ISO (Organización Internacional para la Estandarización) con su principal norma ISO 27001, la cual define cómo organizar la seguridad de la información en cualquier tipo de organización, con o sin fines de lucro, privada o pública, pequeña o grande, y ofrece un estándar internacional que proporciona un modelo para establecer, implementar, utilizar, monitorizar, revisar, mantener y mejorar un SGSI, el hecho de implementar actividades de monitoreo y revisión garantiza que el sistema se mantenga y además se mejore con lo observado y analizado, incidiendo de esta forma en la disponibilidad de los servicios tecnológicos y el cumplimiento de objetivos misionales

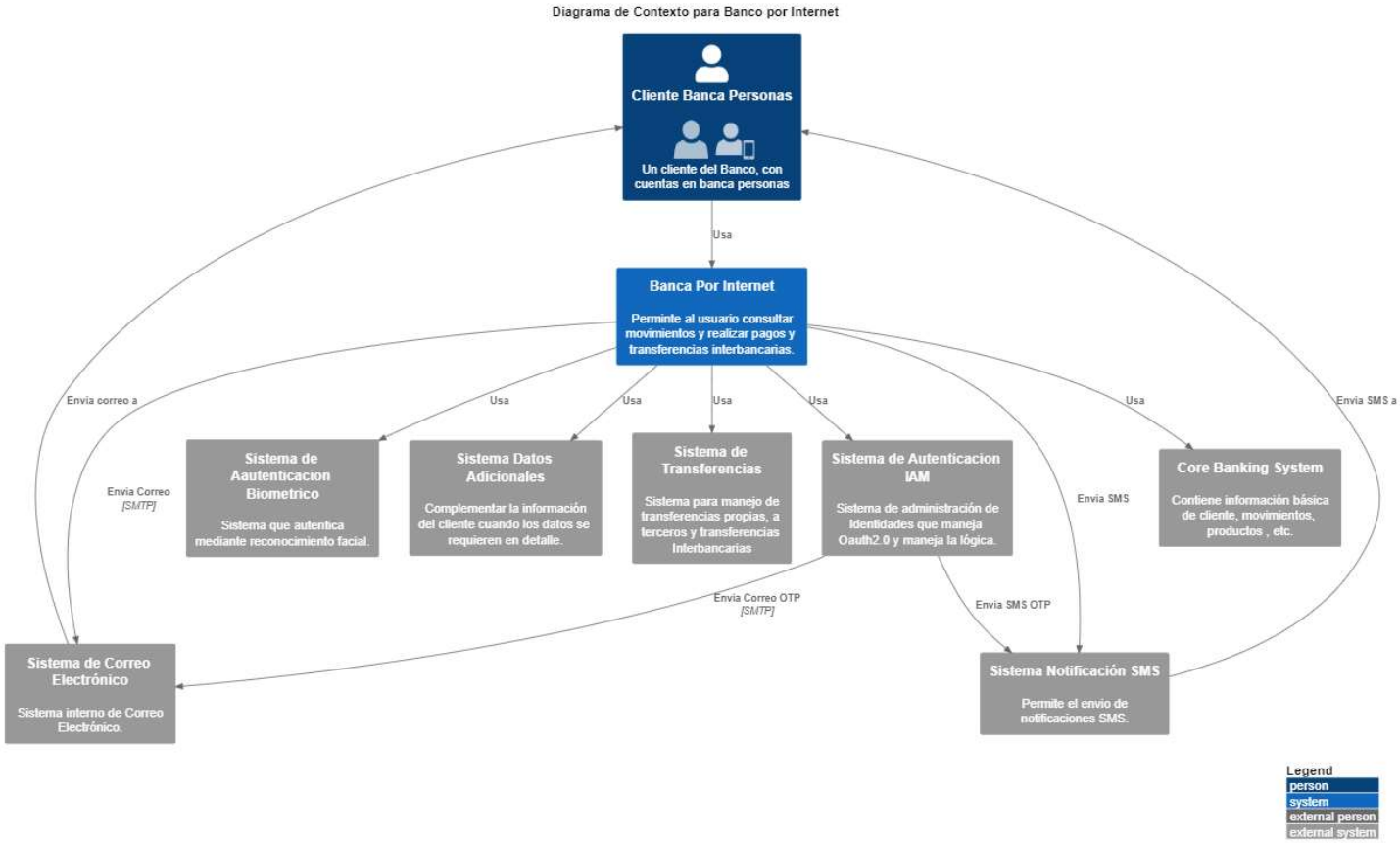
En el Registro Oficial Suplemento No. 459 del 26 de mayo de 2021 se publicó la Ley Orgánica de Protección de Datos Personales. Su objeto es garantizar el derecho a la protección de datos personales. Esto incluye el acceso y la decisión sobre la información y datos de este carácter, así como su correspondiente protección.

Por otro, lado la ubicación de los datos bancarios de una persona está sujeta regulaciones y consideraciones legales, por lo cual los datos bancarios de una persona no se pueden colocar en una nube fuera del Ecuador.

.

3.4. MODELO – C4 DE ARQUITECTURA DE SOFTWARE

DIAGRAMA DE CONTEXTO



Los clientes de BP utilizan el sistema de banca por Internet para ver información sobre el histórico de movimientos de sus cuentas bancarias y realizar pagos y transferencias Interbancarias. El sistema de banca por Internet en su capa de aplicaciones tiene dos canales, Banca Web (SPA) y Banca Móvil.

Core Banking System

Los datos de los clientes (datos básicos) de cuentas, productos y transacciones se en encuentra en Core Banking System, en este Core se encuentran los movimientos de las transacciones y se pueden realizar operaciones de débito crédito para realizar transferencias entre cuentas propias y cuentas de terceros.

Sistema de Autenticación Biométrica

Incluye el reconocimiento Facial, este sistema hace uso de la nube de Azure para realizar la comparación de la foto en vivo con la foto que se registra en la creación del cliente.

Sistema de Datos Adicionales

Es un sistema externo que complementa la información del cliente cuando los datos se requieren en detalle.

Sistema de Transferencias

Para realizar transferencias interbancarias se tiene un sistema externo de transferencias que permite validar si la trasferencia es del mismo banco o va hacia otros bancos, adicionalmente permite realizar transferencias interbancarias Pago Directo hacia los Bancos con convenio con BANRED o Transferencias Banco Central. En este sistema se encuentra encapsuladas las reglas de negocio para el manejo de comisiones, así como los montos máximos y mínimos.

Sistema Autenticación IAM

Para el sistema de autenticación se tiene un IAM, Identity and Acces Managemet. Por lo que se recomienda Uso de flujo Aouth2.0 de autorización implícita para aplicaciones SPA y la aplicación móvil que incluya autorización del código de seguridad o one time password. El IAM genera los códigos de seguridad y envia via correo electrónico y SMS.

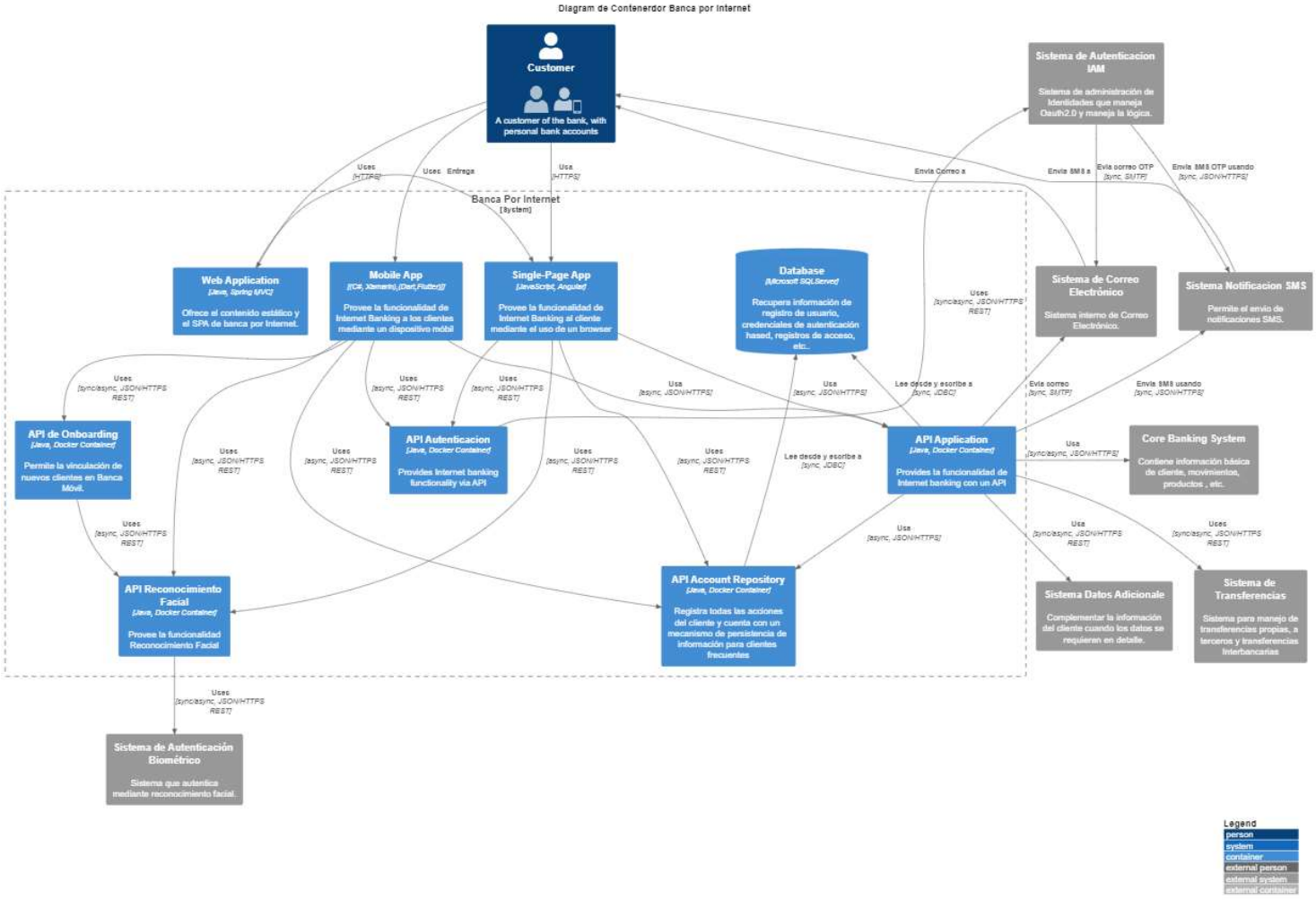
Sistema de Correo Electrónico

El sistema permite notificar a los clientes mediante el envío de notificaciones al correo electrónico.

Sistema Notificación SMS

El sistema permite notificar a los clientes mediante el envío de notificaciones vía SMA.

DIAGRAM DE CONTENEDOR



Web Aplication

La aplicación web es un sistema Java/Spring MVC que ofrece contenido estático (HTML, CSS y JavaScript), incluido el contenido que constituye la aplicación de página única (SPA).

Single-Page App (SPA)

La aplicación de página única es una aplicación JavaScript y Angular que se ejecuta en el navegador web del cliente y proporciona todas las funciones de la banca por Internet.

En la experiencia con un proyecto decidimos utilizar React. React es una biblioteca de JavaScript desarrollada por Facebook que se utiliza para construir interfaces de usuario interactivas y reutilizables sin embargo no pudimos lograr formar el equipo de desarrolladores por la falta de estos en el mercado.

Luego de mi revisión de varios Core System Banking open source pude observa que la mayoría empezó a utilizar Angular en su Front End, Angular es un framework de desarrollo web desarrollado por Google que permite la creación de aplicaciones web modernas y escalables.

Mobile App:

Una aplicación móvil que ofrece la funcionalidad de banca por internet a los clientes a través de dispositivos móviles, ésta es una aplicación móvil multiplataforma creada en C# Xamarin o Dart en Fluter

En referencia al uso de framework multiplataforma, en mi experiencia he utilizado varias herramientas para construir aplicaciones APK, por ejemplo, Android Studio con el uso de Kotlin, Android Studio con el uso de Java, sin embargo, estas aplicaciones son nativas únicamente para Andriod. No se tenía ningún problema hasta cuando decidimos hacer una aplicación móvil de pagos que también pueda ser instalada en iOS, para lo cual decidimos utilizar Flutter con Dart, herramienta que genera códigos nativos para iOS y Android con el mismo código fuente, incluso generar aplicaciones para Windows stand alone y aplicaciones web con el mismo código fuente.

Es importante notar la existencia de los ingenieros de software especialistas en cada herramienta

En base a lo anterior se propone utilizar Flutter o Xamarin para la aplicación móvil

En referencia al uso de Fluter se justifica de la siguiente manera.

Desarrollo Rápido: Flutter utiliza un hot reload que permite realizar cambios en tiempo real, acelerando el proceso de desarrollo.
Interfaz de Usuario Personalizable: Flutter ofrece un conjunto de widgets personalizables que facilita la creación de interfaces de usuario atractivas y consistentes.

Alto Rendimiento: Flutter compila a código nativo, lo que garantiza un rendimiento óptimo en todas las plataformas.

Unificación de Código: Flutter permite escribir un solo código base que funciona en ambas plataformas (iOS y Android), lo que simplifica el mantenimiento.
Flutter/Dart Community: La comunidad activa de Flutter y el soporte de Google aseguran actualizaciones frecuentes y recursos disponibles.

Arquitectura de la Aplicación Móvil con Fluter:

Flutter Widgets: Utiliza widgets específicos de Flutter para construir la interfaz de usuario y gestionar la lógica de presentación.

Comunicación con el Backend: Implementa llamadas a API RESTful para la comunicación eficiente con el backend.

Gestión de Estado: Utiliza el manejo de estado incorporado en Flutter o puede integrar soluciones externas como Provider o Riverpod para gestionar el estado de la aplicación.
Integración con Servicios de Notificación Externos: Desarrolla módulos de notificaciones que se conecten con los servicios externos de notificación de correo electrónico y SMS.

Acceso a Funciones Nativas: Flutter permite acceder a funciones nativas del dispositivo cuando sea necesario a través de canales de plataforma.

Herramientas Recomendadas:

Flutter: Para la construcción de la interfaz de usuario y la lógica de la aplicación.
Dart: Lenguaje de programación utilizado por Flutter.
Provider o Riverpod: Para la gestión eficiente del estado de la aplicación.
Dio: Biblioteca para realizar peticiones HTTP al backend.

En referencia a uso de Xamarin se puede considerar lo siguiente:

Reutilización de Código: Xamarin permite compartir una gran cantidad de código entre las plataformas iOS y Android, lo que facilita el mantenimiento y la consistencia.

Desarrollo Rápido: Al utilizar C# y .NET, los desarrolladores pueden aprovechar la eficiencia de desarrollo y las características avanzadas de este lenguaje.

Acceso a Funcionalidades Nativas: Xamarin permite acceder a las API nativas de las plataformas, garantizando un rendimiento óptimo y la posibilidad de aprovechar las características específicas de cada dispositivo.
Integración con Visual Studio: Xamarin se integra perfectamente con Microsoft Visual Studio, proporcionando un entorno de desarrollo robusto y familiar para los desarrolladores.

Soporte de la Comunidad y Documentación: Xamarin cuenta con una comunidad activa y una amplia documentación, lo que facilita la resolución de problemas y la mejora continua del desarrollo.

Arquitectura de la Aplicación Móvil con Xamarin:

UI/UX Consistente: Utiliza Xamarin.Forms para garantizar una interfaz de usuario consistente en ambas plataformas.
Comunicación con Servicios Backend: Utiliza servicios RESTful para la comunicación eficiente con los servicios backend.

Acceso a Funciones Nativas: Implementa Xamarin Native para acceder a funciones específicas de iOS y Android cuando sea necesario.

Flujo de Onboarding con Reconocimiento Facial: Integra bibliotecas de reconocimiento facial específicas de Xamarin para el flujo de Onboarding.

Herramientas Recomendadas:

Xamarin.Forms: Para crear la interfaz de usuario compartida entre iOS y Android.

Xamarin.Essentials: Para acceder a características nativas del dispositivo de forma sencilla.

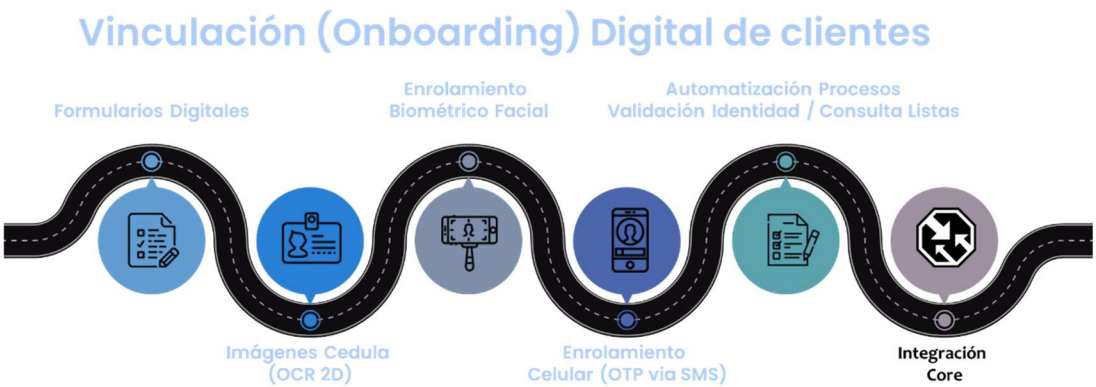
Xamarin.Auth: Para facilitar la implementación de OAuth2.0 en el servicio de autenticación.

Bibliotecas de Reconocimiento Facial de Xamarin: Para el flujo de Onboarding con reconocimiento facial.

.

Api de Onboarding:

En la aplicación móvil cuenta con Sistema de Onboarding que considera el uso módulo de Reconocimiento Facial, Registro de Usuario y Autenticación Post-Onboarding.



El flujo para el onboarding se debe considerar ley del uso de Datos personales, por lo que se emite el acuerdo de términos y condiciones incluido con el acuerdo de uso de datos personales. Para la aceptación se generar un código de seguridad para validar que el celular del cliente es a quien pertenece y se registra la aceptación en el log como una confirmación digital. Luego de lo cual se realiza el ingreso de los datos mínimos del cliente para una mejor experiencia de usuario, para lo cual el formulario digital solicita la Identificación del cliente, correo.

Con estos datos el Banco realiza la validación con la identificación para determinar si es ya un cliente del banco, en caso de no serlo consulta datos alternos en el banco, de otro modo, se realiza una llamada al Registro civil para traer sus datos biométricos, luego se solicita el enrolamiento biométrico facial, lo cual se puede utilizar el Sistema de Autenticación Biométrica. Se procede con las validaciones de listas negras y listas de observados, así como las consideradas en las políticas internas del Banco. Finalmente se procede con la creación del cliente, creación del producto de ser el caso y finalmente se emiten los habilitantes para el servicio.

Database:

Una base de datos Microsoft SQLServer 2019 que almacena información como registros de usuario, credenciales de autenticación hasheadas y registros de acceso.

API Application:

Una aplicación de API que proporciona la funcionalidad de banca por internet a través de una interfaz de programación de aplicaciones (API), utilizando Java y Docker, está instalado en Azure Api Management.

La aplicación API también se comunica con el Core Banking, que hace uso un bus de datos utilizando una interfaz JSON/HTTTPS, en el bus se encuentra los microservicios que accedan a los servicios del CORE para obtener información de histórico de cuentas bancarias, realizar transacciones débito crédito para transferencias, y consulta de datos básicos del cliente. La aplicación API también utiliza el sistema de correo electrónico existente si necesita enviar correos electrónicos a los clientes, así como el sistema de notificaciones SMS.

La Api Application, se comunica con el sistema de datos adicionales o complementarios para consultad datos adicionales de clientes, así como el sistema de transferencias.

API Reconocimiento Facial:

Un servicio de reconocimiento facial que proporciona la funcionalidad de reconocimiento biométrico a través de una interfaz de programación de aplicaciones (API), utilizando Java y

Docker, está instalado en Azure Api Management, esta Api hace uso de un microservicios que interactúa con la Api de reconocimiento Facial de Azure .

API Autenticación: Un servicio de autenticación que proporciona la funcionalidad de autenticación para el sistema de banca por internet, a través de una interfaz de programación de aplicaciones (API), utilizando Java y Docker, está instalado en Azure Api Management y hace las interfaces con los microservicios expuestos en IAM del Banco, el IAM permite generar códigos de seguridad para cumplir con OAuth2.0.

API Auditoria: Un servicio de auditoría que registra todas las acciones del cliente y cuenta con un mecanismo de persistencia de información para clientes frecuentes, el patrón de diseño Patrón de repositorio, este patrón se utiliza para separar la lógica de acceso a datos de la lógica empresarial. En este caso, tenemos un repositorio dedicado para manejar la persistencia de los datos de los clientes frecuentes. Este repositorio proporcionaría métodos específicos para acceder y manipular datos relacionados con clientes frecuentes, aislando así esta funcionalidad.

Patrones de Diseño

1. **Arquitectura de Microservicios**: El sistema está dividido en varios contenedores que representan servicios independientes, como la aplicación web, la aplicación móvil, las APIs de autenticación y reconocimiento facial, entre otros. Esto refleja la arquitectura de microservicios, que permite la escalabilidad y la independencia de implementación de cada componente.

2. **Patrón Modelo-Vista-Controlador (MVC)**: Se puede observar una separación de responsabilidades entre los contenedores que representan la vista (aplicaciones web y móviles), el controlador (APIs de autenticación y otras APIs) y el modelo (base de datos y repositorio de cuentas).

3. **Patrón Repositorio**: un repositorio de datos que registra todas las acciones del cliente y cuenta con un mecanismo de persistencia de información para clientes frecuentes. Este patrón se utiliza para separar la lógica de acceso a datos de la lógica de negocio.

4. **Patrón de Autenticación**: Se utiliza un patrón de autenticación basado en APIs para manejar la autenticación de los usuarios, con contenedores dedicados para las APIs de autenticación y reconocimiento facial. Esto permite una separación de preocupaciones y una fácil escalabilidad de la autenticación.

5. **Patrón de Comunicación Asíncrona**: el uso de comunicación asíncrona entre los diferentes componentes, como las aplicaciones web y móviles que consumen APIs a través de HTTPS y las APIs que interactúan entre sí utilizando comunicación asíncrona JSON/HTTPS REST.

6. **Patrón de Mensajería**: Se utilizan sistemas externos de correo electrónico y notificación SMS para enviar mensajes a los clientes. Esto refleja un patrón de mensajería para la comunicación con los usuarios.

Integracion con Servicio Externos:

A continacion se describe las características de integració a realizarse sobre servicios Externos.

Integración con APIs HTTP/HTTPS: Se observa que la aplicación web, la aplicación de una sola página (SPA) y la aplicación móvil consumen la API de backend utilizando comunicación

asíncrona JSON/HTTPS. Esta integración se realiza a través de solicitudes HTTP/HTTPS asincrónicas para acceder a los servicios proporcionados por el backend.

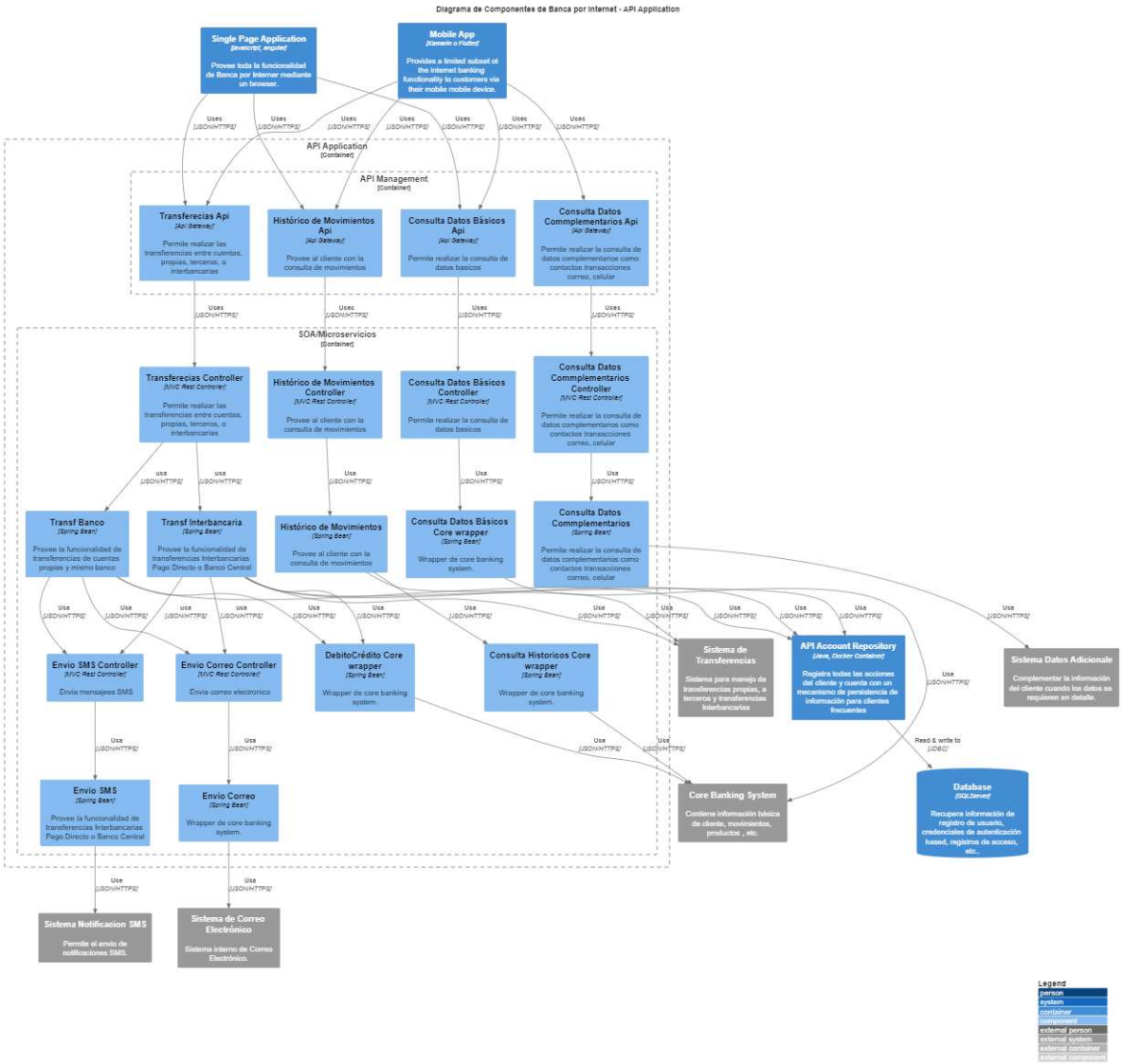
Integración con Bases de Datos usando JDBC: El contenedor de Api Aplication se integra con la base de datos utilizando el protocolo JDBC (Java Database Connectivity). JDBC se utiliza para realizar operaciones de lectura y escritura en la base de datos Microsoft SQLServer, lo que permite al sistema acceder y manipular los datos almacenados.

Integración con Sistemas de Correo Electrónico y SMS: La aplicación interactúa con sistemas externos de correo electrónico y notificación SMS para enviar mensajes a los usuarios. Esto se logra mediante la comunicación síncrona utilizando SMTP (Simple Mail Transfer Protocol) para el sistema de correo electrónico y JSON/HTTPS para el sistema de notificación SMS.

Integración con APIs RESTful: las APIs de autenticación, reconocimiento facial, y otras APIs (como la de Onboarding) se exponen como servicios RESTful y son utilizadas por la aplicación web y la aplicación móvil. Esta integración se realiza mediante solicitudes asincrónicas JSON/HTTPS REST.

Integración con Sistemas de Autenticación IAM: La API de autenticación utiliza un sistema de autenticación IAM (Identity and Access Management) para manejar la autenticación de los usuarios. Esta integración se realiza a través de solicitudes asincrónicas JSON/HTTPS REST, permitiendo que la API de autenticación interactúe con el sistema IAM para autenticar a los usuarios y generar tokens de acceso.

Diagrama de Componentes de Banca por Internet - API Application



El diagrama presenta la estructura de componentes para la aplicación de API en el sistema de Banca por Internet. La aplicación de API proporciona funcionalidades bancarias a través de una interfaz de programación de aplicaciones (API) que permite la comunicación con otros sistemas y clientes.

Componentes Principales

1. **Single Page Application (SPA):** Este componente representa la aplicación de una sola página que ofrece la funcionalidad completa de Banca por Internet a través de un navegador web. Está desarrollada en JavaScript y Angular.
2. **Mobile App :** Esta aplicación proporciona un subconjunto limitado de la funcionalidad de Banca por Internet a los clientes a través de sus dispositivos móviles. Se desarrolla utilizando tecnologías como Xamarin o Flutter.
3. **Database :** Este componente representa la base de datos SQLServer utilizada para almacenar información relacionada con los usuarios, credenciales de autenticación, registros de acceso, etc.
4. **API Account Repository:** Este componente es responsable de registrar todas las acciones del cliente y cuenta con un mecanismo de persistencia de información para clientes frecuentes. Se implementa como una API en un contenedor de Docker utilizando Java.

Integraciones Externas

- **Core Banking System :** Este sistema contiene información básica de cliente, movimientos, productos, etc. Se integra con la API Account Repository para proporcionar información sobre los clientes.
- **Sistema de Transferencias :** Proporciona funcionalidades para manejar transferencias propias, a terceros e interbancarias.
- **Sistema de Correo Electrónico y Sistema de Notificación SMS:** Se utilizan para enviar notificaciones por correo electrónico y mensajes SMS respectivamente.
- **Sistema de Datos Adicionales :** Se utiliza para complementar la información del cliente cuando se requieren detalles adicionales.

Arquitectura de Componentes

La arquitectura de componentes se organiza en dos contenedores principales:

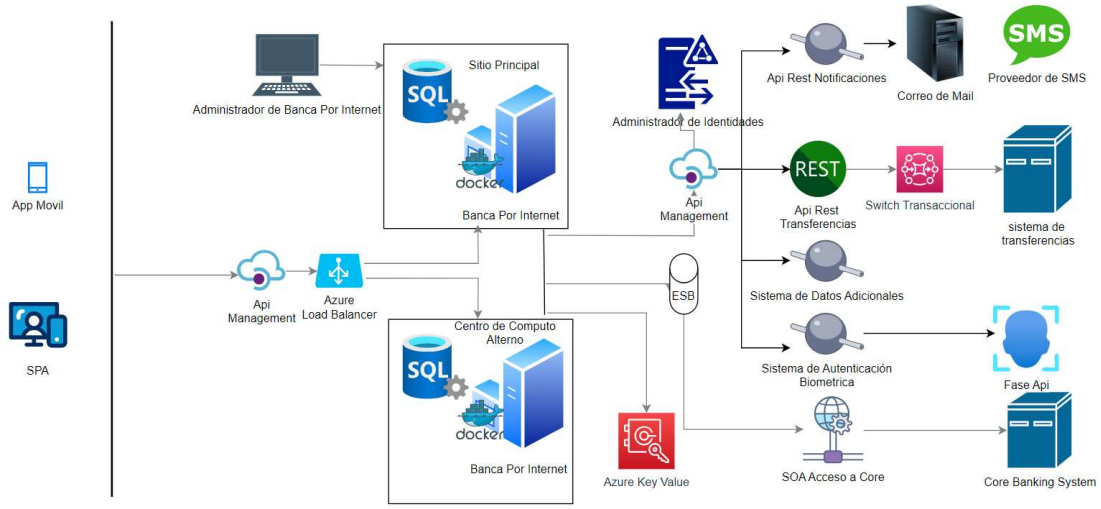
- **API Application:** Contiene los componentes relacionados con la lógica de la aplicación API, incluyendo controladores para diferentes operaciones bancarias, envío de correos electrónicos y SMS, así como wrappers para acceder al Core Banking System.
- **API Management:** Contiene los componentes relacionados con la gestión de API, incluyendo un API Gateway para enrutar las solicitudes de manera apropiada y componentes para proporcionar acceso a diferentes servicios mediante interfaces API.

Relaciones entre Componentes

- La API Account Repository utiliza la base de datos para leer y escribir datos relacionados con las acciones del cliente y la persistencia de información para clientes frecuentes.

- Los componentes de la API Application y API Management interactúan entre sí y con los sistemas externos para proporcionar funcionalidades bancarias a través de interfaces API.

Diagrama de Arquitectura General de la Plantaforma



Los canales de atención se comunican a Azure Api Management, utilizando https, SSL/TLS, usando un api que hace uso de JWT, El Api Management esta frente a un Azure Load Balance que distribuye la carga de los servidores de Backend. Los servidores de backend usan contenedores Docker para mayor disponibilidad y el uso de Devsecop.

En el diagrama se observa tener un sitio principal y un sitio alterno, con lo cual se implementación de redundancia de sitio y servidores críticos. El uso de los servidores en la nube de Azure para garantizar escalabilidad.

Respecto de uso de servidores de datos en la nube se debe considerar la ley que considera que se deben almacenar datos de confidenciales y de ciudadanos en el pais.

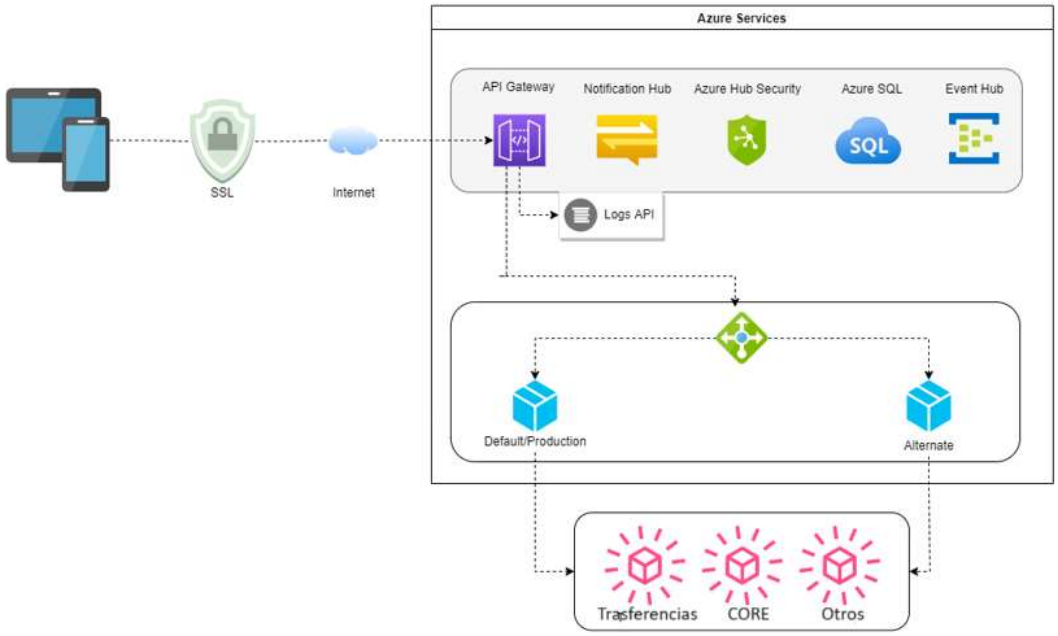
Para la autenticacion se usa un IAM que incluye las funcionalidades de Oauth2.0, mediante el uso de usuario y clave y un segundo factor de autenticación que es el código de seguridad, o tambien pueden autorizarse el ingreso con el usos de Biometria con el uso de Face Api. Utiliza servicios de reconocimiento facial como Azure Cognitive Services para integrar el reconocimiento facial en el flujo de autenticación.

El servidor de correo esta detras de un microservicio que permite en el envio de notificaciones SMTP, mientras que para las notificaciones de SMS, se usa un microservicio que llama al access point del proveedor de envio SMS, en esto puede considerarse directamente el uso de empresas telefonicas como Movistar, Claro o hacer el uso de distribuidores de este servicio.

Banca Por internet tambien accede a los servicios de transferencias mediante el uso de api rest, quien se encarga de conecctarse al switch transacciones para transferencias.

Para las peticiones de CORE Bancking, se hace el llamado a microservicios que estan en un BUS de Datos de IBM, quien se encarga de llamar a los servidios requeridos en el CORE.

Diagrama de Arquitectura de Infraestructura General en Azure



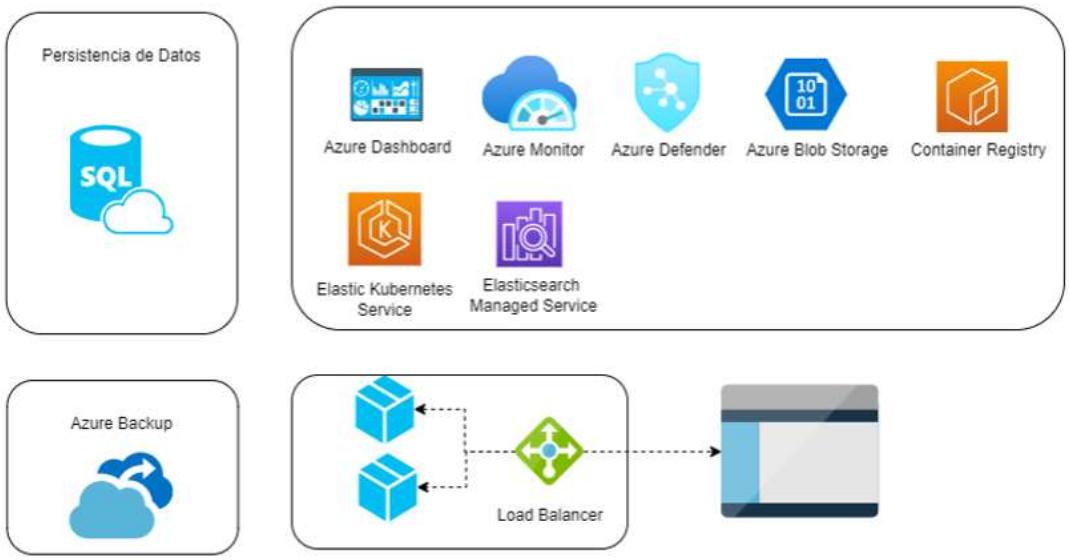
El diagrama presenta componentes adicionales:

- **Alta Disponibilidad**:
  - Implementación de redundancia en servidores críticos, sitio alterno y sitio principal.
  - Uso de servicios en la nube para garantizar escalabilidad.

- **Capa de Integración**:
  - **Azure API Management**: Configurado para manejar solicitudes de clientes y dirigirlas a los servicios correspondientes.
- **Notificaciones**:
  - **Azure Notification Hub**: Envía notificaciones a los usuarios sobre movimientos realizados, integrándose con sistemas externos si es necesario.
- **Seguridad**:
  - **Azure Hub Security**: Proporciona medidas de seguridad adicionales para proteger los datos del cliente y las transacciones financieras.
  - **Azure Key Vault**: Almacena y gestiona claves de API y otros secretos sensibles.
- **Monitoreo y Gestión**:
  - **Azure Application Insights**: Monitorea el rendimiento de las aplicaciones y ofrece insights sobre el comportamiento del usuario.

- **Azure Monitor**: Supervisa la salud de los servicios en tiempo real y proporciona alertas ante cualquier anomalía.

Diagrama de Arquitectura de Infraestructura Sitio Principal en Azure



El diagrama presenta:

- **Almacenamiento y Auditoría**:
  - Se utiliza Azure Blob Storage para almacenar registros de auditoría y otros datos no estructurados.
  - La base de datos de auditoría registra todas las acciones del cliente para cumplir con los requisitos normativos.
  - La base de Datos tiene Microsoft SQLServer 2019
- **Despliegue y Orquestación**:
  - Elastic Kubernetes Service (EKS) administra y orquesta los contenedores Docker para garantizar escalabilidad y alta disponibilidad.
- **Monitoreo y Seguridad**:
  - Azure Dashboard proporciona una visión centralizada del estado del sistema.
  - Azure Defender protege los recursos en la nube contra amenazas.
  - Elasticsearch Managed Service almacena y analiza registros para monitorear la salud y el rendimiento del sistema.

4. ANEXOS

A continuación se anexa el codigo PlantUML para los diagramas C4.

```
@startuml
!include <c4/C4_Context.puml>

'ref http://plantuml.com/stdlib
!include <office/Users/user.puml>
!include <office/Users/mobile_user.puml>

LAYOUT_WITH_LEGEND()

title Diagrama de Contexto para Banco por Internet

Person(customer  , Cliente Banca Personas , "<$user> <$mobile_user>\n Un
cliente del Banco, con cuentas en banca personas" )

System(banking_system, "Banca Por Internet", "Perminte al usuario
consultar movimientos y realizar pagos y transferencias interbancarias.")

System_Ext(mail_system, "Sistema de Correo Electrónico", "Sistema interno
de Correo Electrónico.")
System_Ext(sms_system, "Sistema Notificación SMS", "Permite el envio de
notificaciones SMS.")
System_Ext(mainframe, "Core Banking System", "Contiene información básica
de cliente, movimientos, productos , etc.")
'System_Ext(Onboarding, "Sistema de Onboarding", "Permite la vinculación
de nuevos clientes en Banca Móvil.")
System_Ext(Biometrico, "Sistema de Aautenticacion Biometrico", "Sistema
que autentica mediante reconocimiento facial.")
System_Ext(Complementarios, "Sistema Datos Adicionales", "Complementar la
información del cliente cuando los datos se requieren en detalle.")
System_Ext(trf_ext, "Sistema de Transferencias", "Sistema para manejo de
transferencias propias, a terceros y transferencias Interbancarias")
System_Ext(aim, "Sistema de Autenticacion IAM", "Sistema de
administración de Identidades que maneja Oauth2.0 y maneja la lógica.")

Rel(customer, banking_system, "Usa")
Rel_Back(customer, mail_system, "Envia correo a")
Rel_Back(customer, sms_system, "Envia SMS a")
Rel(banking_system, mail_system, "Envia Correo", "SMTP")
Rel(banking_system, sms_system, "Envia SMS")
Rel(aim, mail_system, "Envia Correo OTP", "SMTP")
Rel(aim, sms_system, "Envia SMS OTP")
Rel(banking_system, mainframe, "Usa")
'Rel(banking_system, Onboarding, "Usa")
Rel(banking_system, Biometrico, "Usa")
Rel(banking_system, trf_ext, "Usa")
```

```
Rel(banking_system, aim, "Usa")
Rel(banking_system, Complementarios, "Usa")
@enduml
```

```
@startuml
'!includeurl https://raw.githubusercontent.com/RicardoNiepel/C4-
PlantUML/master/C4_Container.puml
!include <c4/C4_Container.puml>

'ref http://plantuml.com/stdlib
!include <office/Users/user.puml>
!include <office/Users/mobile_user.puml>

LAYOUT_WITH_LEGEND()


title Diagram de Contenerdor Banca por Internet

Person(customer  , Customer , "<$user> <$mobile_user>\n A customer of the
bank, with personal bank accounts" )

System_Boundary(c1, "Banca Por Internet") {
    Container(web_app, "Web Application", "Java, Spring MVC", "Ofrece el
contenido estático y el SPA de banca por Internet.")
    Container(spa, "Single-Page App", "JavaScript, Angular", "Provee la
funcionalidad de Internet Banking al cliente mediante el uso de un
browser")
    Container(mobile_app, "Mobile App", "(C#, Xamarin),(Dart,Flutter)]",
"Provee la funcionalidad de Internet Banking a los clientes mediante un
dispositivo móbil")
    ContainerDb(database, "Database", "Microsoft SQLServer", "Recupera
información de registro de usuario, credenciales de autenticación hased,
registros de acceso, etc..")
    Container(backend_api, "API Application", "Java, Docker Container",
"Provides la funcionalidad de Internet banking con un API")
    Container(facial_api, "API Reconocimiento Facial", "Java, Docker
Container", "Provee la funcionalidad Reconocimiento Facial")
    Container(autenticacion_api, "API Autenticacion", "Java, Docker
Container", "Provides Internet banking functionality via API")
    Container(ApiAccountRepository , "API Account Repository", "Java,
Docker Container", "Registra todas las acciones del cliente y cuenta con
un mecanismo de persistencia de información para clientes frecuentes")
    Container(Onboarding, "API de Onboarding","Java, Docker Container",
"Permite la vinculación de nuevos clientes en Banca Móvil.")

}

System_Ext(email_system, "Sistema de Correo Electrónico", "Sistema
interno de Correo Electrónico.")
System_Ext(sms_system, "Sistema Notificacion SMS", "Permite el envio de
notificaciones SMS.")
```

```
System_Ext(banking_system, "Core Banking System", "Contiene información básica de cliente, movimientos, productos , etc.")
'System_Ext(Onboarding, "Sistema de Onboarding", "Permite la vinculación de nuevos clientes en Banca Móvil.")
System_Ext(Biometrico, "Sistema de Autenticación Biométrico", "Sistema que autentica mediante reconocimiento facial.")
System_Ext(Complementarios, "Sistema Datos Adicionale", "Complementar la información del cliente cuando los datos se requieren en detalle.")
System_Ext(trf_ext, "Sistema de Transferencias", "Sistema para manejo de transferencias propias, a terceros y transferencias Interbancarias")
System_Ext(iam, "Sistema de Autenticacion IAM", "Sistema de administración de Identidades que maneja Oauth2.0 y maneja la lógica.")

Rel(customer, web_app, "Uses", "HTTPS")
Rel(customer, spa, "Usa", "HTTPS")
Rel(customer, mobile_app, "Uses")

Rel_Neighbor(web_app, spa, "Entrega")
Rel(spa, backend_api, "Usa", "async, JSON/HTTPS")
Rel(mobile_app, backend_api, "Usa", "async, JSON/HTTPS")
Rel(backend_api,ApiAccountRepository ,  "Usa", "async, JSON/HTTPS")
Rel_Back(database, backend_api, "Lee desde y escribe a", "sync, JDBC")
Rel_Back(database, ApiAccountRepository , "Lee desde y escribe a", "sync, JDBC")

Rel(spa, autenticacion_api, "Uses", "async, JSON/HTTPS REST")
Rel(mobile_app, autenticacion_api, "Uses", "async, JSON/HTTPS REST")

Rel(spa, ApiAccountRepository , "Uses", "async, JSON/HTTPS REST")
Rel(mobile_app, ApiAccountRepository , "Uses", "async, JSON/HTTPS REST")


Rel(Onboarding, facial_api, "Uses", "async, JSON/HTTPS REST")
Rel(spa, facial_api, "Uses", "async, JSON/HTTPS REST")
Rel(mobile_app, facial_api, "Uses", "async, JSON/HTTPS REST")



Rel_Back(customer, email_system, "Envia Correo a")
Rel_Back(email_system, backend_api, "Evia correo ", "sync, SMTP")
Rel_Neighbor(backend_api, banking_system, "Usa", "sync/async, JSON/HTTPS")
Rel(backend_api, Complementarios, "Usa", "sync/async, JSON/HTTPS REST")
Rel(backend_api, trf_ext, "Uses", "sync/async, JSON/HTTPS REST")
Rel_Back(customer, sms_system, "Envia SMS a")
Rel_Back(sms_system, backend_api, "Envia SMS usando", "sync, JSON/HTTPS")

Rel(mobile_app, Onboarding, "Uses", "sync/async, JSON/HTTPS REST")
Rel(facial_api, Biometrico, "Uses", "sync/async, JSON/HTTPS REST")
Rel(autenticacion_api, iam, "Uses", "sync/async, JSON/HTTPS REST")
Rel(iam, email_system, "Evia correo OTP", "sync, SMTP")
Rel(iam, sms_system, "Envia SMS OTP usando", "sync, JSON/HTTPS")
```

```
@enduml
```

```
@startuml

'!includeurl https://raw.githubusercontent.com/RicardoNiepel/C4-PlantUML/master/C4_Component.puml
!include <c4/C4_Component.puml>


LAYOUT_WITH_LEGEND()


title Diagrama de Componentes de Banca por Internet - API Application

Container(spa, "Single Page Application", "javascript, angular", "Provee toda la funcionalidad de Banca por Interner mediante un browser.")
Container(ma, "Mobile App", "Xamarin o Flutter", "Provides a limited subset ot the internet banking functionality to customers via their mobile mobile device.")
ContainerDb(db, "Database", "SQLServer", "Recupera información de registro de usuario, credenciales de autenticación hased, registros de acceso, etc..")
Container(ApiAccountRepository , "API Account Repository", "Java, Docker Container", "Registra todas las acciones del cliente y cuenta con un mecanismo de persistencia de información para clientes frecuentes")
System_Ext(mbs, "Core Banking System", "Contiene información básica de cliente, movimientos, productos , etc.")
System_Ext(trf, "Sistema de Transferencias", "Sistema para manejo de transferencias propias, a terceros y transferencias Interbancarias")
System_Ext(email_system, "Sistema de Correo Electrónico", "Sistema interno de Correo Electrónico.")
System_Ext(sms_system, "Sistema Notificacion SMS", "Permite el envio de notificaciones SMS.")
System_Ext(Complementarios, "Sistema Datos Adicionale", "Complementar la información del cliente cuando los datos se requieren en detalle.")

Container_Boundary(api, "API Application") {

Container_Boundary(Gateway, "API Management") {
    Component(trf_ag, "Transferecias Api", "Api Gateway", "Permite realizar las transferencias entre cuentas, propias, terceros, o interbancarias")
    Component(DatosB_ag, "Consulta Datos Básicos Api", "Api Gateway", "Permite realizar la consulta de datos basicos")
    Component(DatosC_ag, "Consulta Datos Commplementarios Api", "Api Gateway", "Permite realizar la consulta de datos complementarios como contactos transacciones correo, celular")
```

```
    Component(accounts_ag, "Histórico de Movimientos Api", "Api Gateway",
"Provee al cliente con la consulta de movimientos")
}
Container_Boundary(EBS, "SOA/Microservicios") {
    Component(mbsfacade, "Consulta Historicos Core wrapper", "Spring
Bean", "Wrapper de core banking system.")
    Component(mbsDebitoCredito, "DebitoCrédito Core wrapper", "Spring
Bean", "Wrapper de core banking system.")
    Component(mbsDBasicos, "Consulta Datos Básicos Core wrapper", "Spring
Bean", "Wrapper de core banking system.")
    Component(trf_c, "Transferecias Controller", "MVC Rest Controller",
"Permite realizar las transferencias entre cuentas, propias, terceros, o
interbancarias")
    Component(DatosB_c, "Consulta Datos Básicos Controller", "MVC Rest
Controller", "Permite realizar la consulta de datos basicos")
    Component(DatosC_c, "Consulta Datos Commplementarios Controller",
"MVC Rest Controller", "Permite realizar la consulta de datos
complementarios como contactos transacciones correo, celular")
    Component(accounts_c, "Histórico de Movimientos Controller", "MVC
Rest Controller", "Provee al cliente con la consulta de movimientos")
    Component(accounts, "Histórico de Movimientos ", "Spring Bean",
"Provee al cliente con la consulta de movimientos")
    Component(trfbanco, "Transf Banco", "Spring Bean", "Provee la
funcionalidad de transferencias de cuentas propias y mismo banco")
    Component(DatosC, "Consulta Datos Commplementarios ", "Spring Bean",
"Permite realizar la consulta de datos complementarios como contactos
transacciones correo, celular")
    Component(trfInter, "Transf Interbancaria", "Spring Bean", "Provee la
funcionalidad de transferencias Interbancarias Pago Directo o Banco
Central")
    Component(email_c, "Envio Correo Controller", "MVC Rest Controller",
"Envia correo electronico")
    Component(sms_c, "Envio SMS Controller", "MVC Rest Controller",
"Envia mensajees SMS")
    Component(email, "Envio Correo", "Spring Bean", "Wrapper de core
banking system.")
    Component(sms, "Envio SMS", "Spring Bean", "Provee la funcionalidad
de transferencias Interbancarias Pago Directo o Banco Central")
}



    Rel(accounts_c, accounts, "Usa","JSON/HTTPS")
    Rel(accounts, mbsfacade, "Usa","JSON/HTTPS")
    Rel(accounts, ApiAccountRepository, "Usa","JSON/HTTPS")
    Rel(trf_c, trfbanco, "usa", "JSON/HTTPS")
    Rel(trf_c, trfInter, "usa", "JSON/HTTPS")
    Rel(DatosC, Complementarios, "Usa","JSON/HTTPS")
    Rel(trfbanco, mbsDebitoCredito, "Usa","JSON/HTTPS")
    Rel(trfbanco, email_c, "Usa","JSON/HTTPS")
    Rel(trfbanco, sms_c, "Usa","JSON/HTTPS")
```

```
    Rel(trfbanco, ApiAccountRepository, "Usa","JSON/HTTPS")
    Rel(trfInter, email_c, "usa", "JSON/HTTPS")
    Rel(trfInter, sms_c, "usa", "JSON/HTTPS")
    Rel(trfInter, ApiAccountRepository, "Usa","JSON/HTTPS")
    Rel(trfInter, trf, "Usa","JSON/HTTPS")
    Rel(mbsfacade, mbs, "Use", "JSON/HTTPS")
    Rel(trfInter, trf, "Usa","JSON/HTTPS")
    Rel(trfInter, mbsDebitoCredito, "Usa","JSON/HTTPS")
    Rel(DatosB_c, mbsDBasicos, "Usa","JSON/HTTPS")

    Rel(email_c, email, "Usa","JSON/HTTPS")
    Rel(sms_c, sms, "Usa","JSON/HTTPS")

    Rel(DatosC_c, DatosC, "Usa","JSON/HTTPS")
 }
Rel_Neighbor(spa, accounts_ag, "Uses", "JSON/HTTPS")

Rel(mbsDBasicos, mbs, "Use", "JSON/HTTPS")
Rel(mbsDebitoCredito, mbs, "Use", "JSON/HTTPS")

Rel(ma, trf_ag, "Uses", "JSON/HTTPS")
Rel(ma, accounts_ag, "Uses", "JSON/HTTPS")
Rel(spa, DatosB_ag, "Uses", "JSON/HTTPS")
Rel(spa, trf_ag, "Uses", "JSON/HTTPS")
Rel(ma, DatosB_ag, "Uses", "JSON/HTTPS")
Rel(ma, DatosC_ag, "Uses", "JSON/HTTPS")
Rel(DatosC_ag, DatosC_c, "Uses", "JSON/HTTPS")
Rel(DatosB_ag, DatosB_c, "Uses", "JSON/HTTPS")
Rel(trf_ag, trf_c, "Uses", "JSON/HTTPS")
Rel(accounts_ag, accounts_c, "Uses", "JSON/HTTPS")
Rel(email, email_system, "Usa","JSON/HTTPS")
Rel(sms, sms_system, "Use", "JSON/HTTPS")
Rel(ApiAccountRepository , db, "Read & write to", "JDBC")
@enduml
```