Manual de Arquitectura para el Banco BP

Diseño y Desarrollo de un Banco en Línea Seguro y Eficiente

ELABORADO POR

RUBEN MARTINEZ OLIVA

MAYO 2024

INDICE DE CONTENIDO

1. Introducción

Objetivos del manual:

El objetivo de este manual es proporcionar una guía detallada sobre la arquitectura de software del sistema de banco en línea. Se pretende ofrecer una visión general de la estructura del sistema, los componentes principales, los patrones arquitectónicos utilizados y las decisiones de diseño clave.

Alcance del manual:

Este manual aborda la arquitectura de software específicamente del sistema de banco en línea. No cubre aspectos operativos, de seguridad o de negocio del banco en línea, los cuales pueden ser abordados en documentos separados.

Audiencia prevista:

Este manual está dirigido a arquitectos de software, desarrolladores, gerentes de proyecto y otros miembros del equipo involucrados en el diseño, desarrollo, implementación y mantenimiento del sistema de banco en línea. También puede ser útil para partes interesadas externas que deseen comprender la arquitectura del sistema.

2. Descripción general del sistema

Visión general del banco en línea:

El sistema de banco en línea es una plataforma digital que permite a los clientes acceder y gestionar sus cuentas bancarias de manera remota a través de Internet. Proporciona una amplia gama de servicios financieros, como consultas de saldo, transferencias de fondos, pago de facturas, solicitud de préstamos, entre otros.

El objetivo principal del sistema es ofrecer a los clientes una experiencia bancaria conveniente y segura desde cualquier lugar y en cualquier momento.

Funcionalidades principales:

Autenticación de usuarios: Permite a los clientes iniciar sesión de forma segura en sus cuentas bancarias utilizando credenciales únicas, como nombre de usuario y contraseña.

Consultas de saldo y transacciones: Los clientes pueden verificar el saldo de sus cuentas, revisar el historial de transacciones y obtener detalles sobre depósitos, retiros y transferencias.

Transferencias de fondos: Facilita la transferencia de dinero entre cuentas bancarias internas y externas, así como pagos a terceros mediante transferencias bancarias.

Pago de facturas: Permite a los clientes pagar facturas de servicios públicos, tarjetas de crédito y otros servicios mediante el sistema de banco en línea.

Solicitud y gestión de préstamos: Ofrece a los clientes la posibilidad de solicitar préstamos en línea, verificar el estado de las solicitudes y gestionar los pagos de préstamos existentes.

Alertas y notificaciones: Proporciona alertas y notificaciones personalizadas sobre actividades importantes en las cuentas de los clientes, como transacciones realizadas, saldos bajos, vencimiento de facturas, etc.

3. Arquitectura de alto nivel del sistema

Modelo C4: Nivel 1: Contexto
En el nivel de contexto, se ofrece una vista panorámica del sistema de banco en línea y su interacción con los usuarios y otros sistemas externos en el ecosistema financiero.

[System Context] Sistema bancario por internet

Cliente del Banco: Representa los usuarios finales que interactúan con el sistema de banco en línea para realizar transacciones financieras y gestionar sus cuentas bancarias.

Sistema Bancario: Consiste en el núcleo del sistema, que gestiona la lógica de negocio y las operaciones financieras, incluyendo la autorización de transacciones, el procesamiento de pagos y la gestión de cuentas.

Sistema de Notificaciones: Es un componente adicional que proporciona funcionalidad de notificación para informar a los clientes sobre eventos importantes, como transacciones realizadas, alertas de seguridad y actualizaciones de cuenta.

Esta descripción y diagrama de arquitectura presentan una visión integral y detallada del sistema de banco en línea y su interacción con su entorno, proporcionando un contexto claro para comprender su funcionamiento y sus relaciones con los usuarios y otros sistemas externos en el ecosistema financiero.

Modelo C4: Nivel 2: Contenedores

El diagrama de contenedores muestra la forma de alto nivel de la arquitectura de software y cómo se distribuyen las responsabilidades en ella. Describe la estructura de alto nivel de la arquitectura de software del sistema bancario, centrándose en la distribución de responsabilidades entre los diferentes contenedores.

[Container] Internet Banking System

Web Application: Es una interfaz de usuario basada en web que proporciona acceso a los usuarios al sistema bancario a través de un navegador web.

Mobile App: Es una aplicación móvil que permite a los usuarios acceder al sistema bancario desde dispositivos móviles.

Account: Gestiona los productos financieros que tiene un usuario, como cuentas corrientes, de ahorro, etc. También muestra los movimientos del último mes y proporciona certificados bancarios. Cuenta con información básica del cliente, La data completa de clientes se encuentra en el servicio de **Client**.

CashFlow: Gestiona las transacciones financieras realizadas por los usuarios, incluyendo depósitos, retiros, transferencias, etc.

SingUP: Manejo el onboarding de clientes integra la biometría para fortalecer la seguridad y la verificación de identidad.

User: Gestiona la información de los usuarios del banco, incluyendo la autenticación y autorización a través de tokens OAuth 2.0.

Client : Gestiona datos críticos de los clientes de un banco, como información personal y de contacto. Este contenedor asegura que todos los datos sensibles se almacenan de forma encriptada, manteniendo la integridad y confidencialidad de la información conforme a las normativas legales. En Ecuador, esta práctica está regulada por la Ley Orgánica de Protección de Datos Personales.

Data: Gestiona toda la ingeniería de datos de la organización, incluyendo la recopilación, almacenamiento y análisis de datos para respaldar las decisiones estratégicas del negocio.

Integration: Se encarga de la comunicación con sistemas externos al banco, como cajeros automáticos (ATM) y pasarelas de pago.

Bouncer : Gestiona el saldo de los clientes y los movimientos generados por intereses, como créditos y débitos.

Core : Gestiona el dinero disponible en el banco a través de cuentas concentradoras, asegurando la liquidez y la disponibilidad de fondos.

Notification: Gestiona el envío de notificaciones a los usuarios a través de diferentes canales, como correo electrónico y mensajes de texto (SMS).

AuditGuard: Gestiona los registros de auditoría de las acciones de los usuarios en el sistema, garantizando la integridad y seguridad de los datos. También valida que no se estén produciendo actividades fraudulentas por parte de usuarios malintencionados.

Kafka: Gestiona la comunicación entre los diferentes microservicios del sistema bancario, proporcionando una plataforma de mensajería distribuida y escalable.

Este diagrama permite comprender cómo están organizados y comunicados los diferentes componentes del sistema bancario, facilitando la gestión y el mantenimiento de la arquitectura de software.

3.1 Arquitectura del Frontend

Web Application: Es una interfaz de usuario basada en web que proporciona acceso a los usuarios al sistema bancario a través de un navegador web.
Para el desarrollo de la SPA se validaron dos tecnologías (Next.js y Angular)
Características de **Next.js**:

- **Renderizado del lado del servidor (SSR)**: Next.js permite el renderizado del lado del servidor, lo que puede mejorar significativamente el tiempo de carga inicial y el SEO.
- **Generación de sitios estáticos (SSG)**: Puede pre-renderizar páginas para cargar rápidamente contenido estático.
- **Optimización de imágenes:** Incluye herramientas automáticas para optimizar imágenes, lo cual es útil para mejorar el rendimiento.
- **Enrutamiento basado en archivos:** El sistema de enrutamiento se basa en la estructura de archivos del proyecto.
- **Soporte para TypeScript**: Aunque Next.js está construido sobre React, ofrece soporte completo para TypeScript.

Características de **Angular:**

- **Arquitectura MVC/MVVM**: Angular utiliza un patrón Modelo-Vista-Controlador/Modelo-Vista-VistaModelo, lo que facilita la separación de la lógica de la interfaz de usuario.
- **Enlace bidireccional de datos:** Permite una sincronización automática entre el modelo y la vista, lo que puede simplificar el desarrollo.
- **Uso de TypeScript**: Angular está construido sobre TypeScript, lo que proporciona tipado estático y herramientas avanzadas de desarrollo.
- **Inyección de dependencias:** Facilita la gestión de dependencias y la reutilización de código.
- **Herramientas robustas:** Viene con una suite de herramientas para pruebas, animaciones, y más.

Después de analizar a detalles estas 2 tecnología se seleccionó **Next.js**:

- **Rendimiento:** Next.js puede ofrecer un mejor rendimiento inicial debido a su capacidad de SSR y SSG. Angular también es rápido, pero puede requerir más tiempo para la carga inicial debido a su enfoque de SPA tradicional.

- **SEO**: Next.js tiene una ventaja en SEO gracias al SSR, lo que es importante para un banco que quiere ser visible en las búsquedas en línea.

- **Desarrollo**: Next.js puede ser más fácil de empezar para desarrolladores con experiencia en React. Angular tiene una curva de aprendizaje más pronunciada.

3.2 Arquitectura Mobile

Mobile App: Es una aplicación móvil que permite a los usuarios acceder al sistema bancario desde dispositivos móviles.

Para el desarrollo de la App Movil se validaron dos tecnologías (React Native y Flutter)

React Native:

- Desarrollado por Facebook, permite crear aplicaciones móviles nativas usando JavaScript y React.

- Reutilización de código entre plataformas iOS y Android.

- Gran comunidad de desarrolladores y acceso a numerosas bibliotecas y herramientas.

- Actualizaciones en vivo y recarga en caliente, lo que facilita la iteración rápida durante el desarrollo.

- Enfoque en la interfaz de usuario, con una experiencia cercana a la nativa.

Flutter:

- Desarrollado por Google, utiliza el lenguaje Dart para construir aplicaciones nativas compiladas.

- Widgets personalizables que permiten crear interfaces de usuario complejas y atractivas.

- Rendimiento cercano al nativo gracias a la compilación en código de máquina.

- Amplia documentación y soporte de Google.

- Motor de renderizado propio, lo que significa que la aplicación se ve igual en todas las plataformas.

Después de analizar a detalles estas 2 tecnología se seleccionó

Flutter: Basados en el rendimiento y la personalización de la interfaz de usuario más profunda que permite la implantación de una arquitectura **Server-Driven UI**

Server-Driven UI (SDUI) puede ser particularmente beneficioso para un banco por varias razones:

- **Flexibilidad y Iteración Rápida:** Con SDUI, las actualizaciones de la interfaz de usuario se realizan en el servidor, lo que elimina la necesidad de actualizar la aplicación del cliente cada vez. Esto permite a los bancos desplegar cambios rápidamente, como ajustes de diseño o nuevas características, sin pasar por el proceso de aprobación de las tiendas de aplicaciones.

- **Personalización en Tiempo Real:** Los bancos pueden personalizar la experiencia del usuario en tiempo real, mostrando ofertas, servicios o información relevante basada en el comportamiento del usuario o eventos específicos.

- **Mantenimiento Simplificado:** Al centralizar la lógica de la interfaz de usuario en el servidor, se reduce la complejidad del mantenimiento de la aplicación y se facilita la gestión de múltiples versiones de la aplicación.

- **Consistencia en Múltiples Plataformas:** SDUI asegura una experiencia de usuario consistente en diferentes dispositivos y plataformas, ya que el servidor controla la presentación de la UI.

- **Reducción de Carga en el Cliente:** Al manejar más lógica en el lado del servidor, se reduce la carga computacional en el dispositivo del cliente, lo que puede mejorar el rendimiento de la aplicación.

- **Seguridad Mejorada:** Los bancos pueden implementar y actualizar medidas de seguridad de manera centralizada, lo que es crucial para proteger la información sensible de los clientes.

En resumen, SDUI nos va ayudar a ofrecer una experiencia de usuario más rica y personalizada, al tiempo que simplifica el mantenimiento y mejora la seguridad. Es una solución eficiente para adaptarse rápidamente a las cambiantes demandas del mercado y las expectativas de los usuarios.

3.3 Arquitectura del Backend

Implementación de una arquitectura de Microservicios

La arquitectura de microservicios ofrece múltiples beneficios a los bancos, especialmente en términos de escalabilidad, mantenimiento, y agilidad en el desarrollo:

- **Escalabilidad:** Los microservicios permiten que las aplicaciones bancarias manejen un gran número de clientes y solicitudes de manera eficiente. Pueden escalar servicios individuales según la demanda sin afectar a todo el sistema.

- **Alta Disponibilidad y Tolerancia a Fallos:** En la banca digital, es crucial ofrecer servicios sin demoras. La arquitectura de microservicios puede aumentar la disponibilidad y es más tolerante a fallos, ya que un problema en un servicio no necesariamente afecta a los demás.

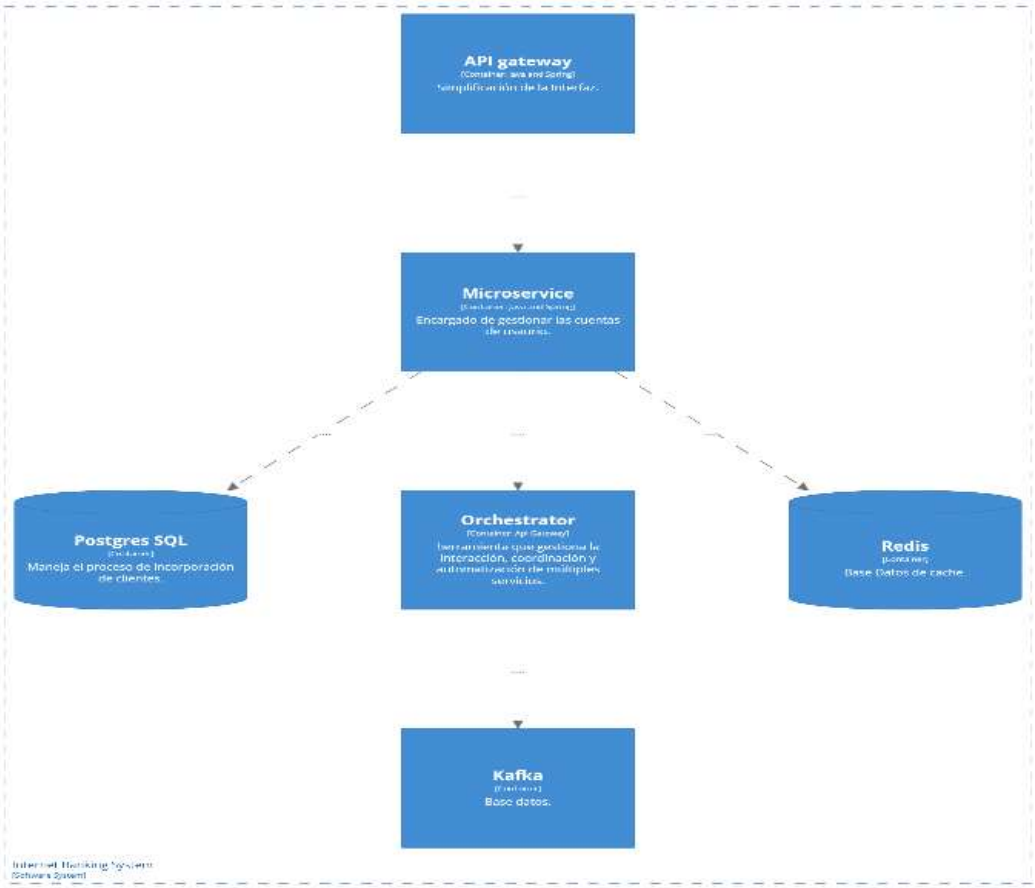
- **Facilidad de Mantenimiento:** Los microservicios son más fáciles de mantener y actualizar. Los equipos pueden implementar y probar nuevas funciones rápidamente y revertirlas si es necesario, lo que acelera el tiempo de comercialización de nuevas características.

- **Seguridad y Cumplimiento:** Los bancos pueden implementar medidas de seguridad y cumplimiento de manera más efectiva en microservicios individuales, lo que facilita la gestión de la seguridad y el cumplimiento normativo.

- **Flexibilidad:** Los microservicios permiten a los bancos ser más flexibles en su desarrollo, utilizando diferentes tecnologías y frameworks que se adaptan mejor a cada servicio.

Estructura de los Contenedores de los subsistemas

- **Account**
- **CashFlow**
- **Client**
- **Data**:
- **Integration**
- **Bouncer**
- **Core**
- **Notification**
- **AuditGuard**



[Container] Internet Banking System

API Gateway: es un componente crucial en la arquitectura de microservicios, y ofrece varios beneficios para los bancos, especialmente en términos de gestión de interfaces de programación de aplicaciones (APIs).

Beneficios

- **Simplificación de la Interfaz**: Actúa como un punto único de entrada para todas las APIs, simplificando la interacción entre los clientes y los servicios.

- **Seguridad Mejorada:** El API Gateway puede manejar la autenticación y autorización, así como la encriptación y el aseguramiento de las APIs, lo que es fundamental para la seguridad bancaria.

- **Monitoreo y Analítica:** Permite a los bancos monitorear el uso de las APIs, detectar patrones de tráfico anormales y realizar análisis de datos para mejorar los servicios.

- **Gestión de Tráfico**: Ayuda a gestionar la carga y distribuir el tráfico de manera eficiente, lo que puede ser crucial durante picos de demanda.

- **Facilita la Transformación Digital:** Al exponer sus APIs, los bancos pueden crear nuevas líneas de negocio y mejorar la experiencia del cliente, ofreciendo servicios como conciliación bancaria automática y pagos por transferencia automatizados.

Microservice: Microservicios necesarios para la implementación de los subsistemas bancarios.

Orchestrator: dedicado a la gestión de envíos y peticiones de eventos a Apache Kafka actúa como un coordinador centralizado para el flujo de mensajes entre productores y consumidores. Aquí hay una descripción de sus responsabilidades y capacidades:

Coordinación de Eventos: El Orchestrator se encarga de dirigir los eventos entrantes a los tópicos correctos de Kafka y de asegurar que los consumidores reciban los mensajes adecuados.

Balanceo de Carga: Distribuye la carga entre diferentes productores y consumidores para optimizar el rendimiento y la eficiencia del sistema.

Resiliencia: Implementa mecanismos de recuperación y reintento para manejar fallos en la comunicación o en los servicios individuales.

Monitoreo: Supervisa el estado de los flujos de eventos y proporciona alertas y métricas para garantizar la salud del sistema.

Seguridad: Gestiona la autenticación y autorización de los servicios que interactúan con Kafka, protegiendo la integridad y la confidencialidad de los datos.
Este microservicio Orchestrator es esencial para mantener un sistema de mensajería robusto y eficiente, permitiendo que un banco maneje grandes volúmenes de datos en tiempo real con confiabilidad y seguridad.

Redis: Avanzada base de datos en memoria, utilizada como sistema de caché para optimizar la recuperación de datos en aplicaciones de alto rendimiento. En el contexto de un banco, Redis se implementa para que las solicitudes de datos se procesen inicialmente a través de la caché de Redis. Si los datos solicitados no están disponibles en la caché, la búsqueda se redirige a la base de datos PostgreSQL subyacente. Esta estrategia garantiza una respuesta rápida y eficiente, especialmente para operaciones de lectura frecuentes.

Beneficios:

- **Rendimiento Mejorado:** Redis proporciona tiempos de respuesta de submilisegundos, lo que acelera significativamente el acceso a los datos críticos.
- **Alta Disponibilidad:** Con un SLA de tiempo de actividad del 99.999%, Redis asegura un acceso constante a los datos, lo que es esencial para las operaciones bancarias.
- **Escalabilidad:** Mantiene un rendimiento óptimo incluso con un volumen de operaciones que alcanza los 200 millones por segundo, lo que permite al banco escalar según las necesidades del negocio.
- **Costo-Eficiencia:** Redis es rentable para grandes conjuntos de datos, lo que ayuda a los bancos a gestionar los costos operativos mientras manejan grandes volúmenes de transacciones.
- **Resiliencia:** La arquitectura de Redis está diseñada para ser resistente y escalable, lo que garantiza la fiabilidad en entornos de misión crítica.

La implementación de Redis como una capa de caché en la arquitectura de datos de un banco proporciona una solución robusta que mejora la experiencia del usuario, la eficiencia operativa y la capacidad de respuesta del sistema frente a la demanda fluctuante.

Postgres SQL: Base de datos relacional de código abierto que ofrece numerosos beneficios para su uso en instituciones bancarias, destacándose por su confiabilidad, seguridad y rendimiento. Aquí están algunos de los beneficios clave:

- **Confiabilidad:** PostgreSQL es conocido por su estabilidad y confiabilidad, con más de dos décadas de desarrollo activo y mejoras continuas.
- **Cumplimiento ACID:** Cumple completamente con las propiedades ACID (Atomicidad, Consistencia, Aislamiento, Durabilidad), lo que garantiza transacciones seguras y confiables, esenciales para operaciones bancarias.
- **Escalabilidad:** Ofrece gran escalabilidad, permitiendo ajustar la configuración según el hardware disponible para manejar una mayor cantidad de peticiones simultáneas.
- **Seguridad:** Proporciona múltiples funciones para una mejor seguridad, como TDE (Transparent Data Encryption) y Data Masking, protegiendo así los datos valiosos del banco.

- **Extensibilidad:** Permite la creación y uso de una amplia variedad de extensiones, lo que aumenta su funcionalidad y adaptabilidad a diferentes necesidades.

- **Soporte para SQL Estándar:** Implementa casi todas las funcionalidades del estándar ISO/IEC 9075:2011, facilitando la realización de consultas y la inclusión de scripts de otros motores de bases de datos.

Estos beneficios hacen de PostgreSQL una opción robusta y segura para las necesidades de gestión de datos de un banco, proporcionando una plataforma sólida para el almacenamiento y la manipulación de información financiera crítica.

3.4 Arquitectura de desarrollo de los microservicios

La arquitectura hexagonal, también conocida como arquitectura de puertos y adaptadores, es un patrón de diseño de software que busca separar las responsabilidades de cada componente de un sistema. A través de este enfoque, se pueden obtener varios beneficios que ayudan a garantizar la calidad del software y su escalabilidad.

Separación de responsabilidades:

- Claramente separa las responsabilidades de la lógica de negocio de las responsabilidades de la infraestructura.

- Facilita el cambio de dependencias sin afectar la lógica de negocio.

Mantenibilidad:

- Al separar la aplicación de otras dependencias (como un framework web), se facilita el cambio de dependencias sin afectar la lógica de negocio.

Calidad desde el diseño:

- La arquitectura hexagonal guía la construcción de software de calidad mediante directrices y reglas.

En resumen, la arquitectura hexagonal es una práctica recomendada para construir sistemas escalables, flexibles y mantenibles. Al separar claramente las responsabilidades, se logra una mayor flexibilidad y facilidad de mantenimiento a lo largo del tiempo.

4 Consideraciones de Seguridad

- **Cifrado de Datos:** Todos los datos sensibles, como la información personal del cliente, las transacciones financieras y los detalles de la cuenta, deben estar protegidos mediante cifrado. Se debe implementar un cifrado sólido tanto en reposo como en tránsito para garantizar la confidencialidad de la información.

- **Autenticación y Autorización Robustas:** Es fundamental implementar un sistema de autenticación sólido para verificar la identidad de los usuarios. Además, se deben establecer políticas de autorización claras y granulares para controlar el acceso a diferentes funcionalidades y datos dentro del sistema bancario en línea.

- **Prevención de Amenazas Externas e Internas: Se deben implementar medidas de seguridad para** proteger el sistema contra amenazas externas, como ataques de denegación de servicio (DDoS), inyección de código SQL y ataques de phishing. Además, es importante monitorear de cerca las actividades de los usuarios internos para detectar y prevenir posibles actividades fraudulentas o maliciosas.

- **Gestión de Identidades y Accesos:** La gestión adecuada de identidades y accesos es esencial para garantizar que solo los usuarios autorizados tengan acceso a las funcionalidades y datos pertinentes. Esto incluye la gestión de contraseñas seguras, la implementación de políticas de bloqueo de cuentas y la capacidad de revocar los accesos de los usuarios en caso de que sea necesario.

- **Auditoría y Registro de Eventos:** Se deben implementar mecanismos de auditoría y registro de eventos para realizar un seguimiento de todas las actividades dentro del sistema bancario en línea. Esto ayuda a detectar y responder rápidamente a posibles incidentes de seguridad, así como a cumplir con los requisitos de cumplimiento normativo.

- **Resiliencia y Continuidad del Negocio:** Para garantizar la disponibilidad y la continuidad del servicio, se deben implementar medidas de resiliencia, como la redundancia de servidores, la copia de seguridad de datos y la planificación de la continuidad del negocio en caso de desastres o interrupciones inesperadas.

Estas son solo algunas consideraciones clave en materia de seguridad para un banco en línea. Es fundamental diseñar y desarrollar el sistema con un enfoque centrado en la seguridad desde el principio y mantener una postura proactiva en la gestión de riesgos y amenazas en curso.
La mejor estrategia que un banco debe implementar para la gestión de sus tokens implica una combinación de seguridad, flexibilidad y cumplimiento normativo.

Tokenización Segura: Utilizar la tokenización para proteger la información sensible del cliente, como los detalles de la tarjeta de crédito, reemplazándolos con tokens no descriptivos que son inútiles fuera del sistema bancario.

Autenticación Robusta: Combinar tokens con métodos de autenticación fuertes, como la autenticación multifactor (MFA) y la biometría, para asegurar que solo los usuarios autorizados puedan acceder a los servicios.

Gestión Centralizada de Tokens: Mantener un sistema centralizado para la emisión, renovación y revocación de tokens, lo que permite un control más estricto y una visión general de todos los tokens emitidos.

Cumplimiento Normativo: Asegurarse de que la gestión de tokens cumpla con las regulaciones financieras locales e internacionales, como el Estándar de Seguridad de Datos para la Industria de Tarjeta de Pago (PCI DSS) y el Reglamento General de Protección de Datos (GDPR).

Monitoreo y Análisis: Implementar sistemas de monitoreo y análisis para detectar y responder a actividades sospechosas o fraudulentas en tiempo real.

Educación del Usuario: Proporcionar educación y recursos a los clientes sobre la seguridad de los tokens y cómo proteger su información personal y financiera.

5 Proceso de onboarding

En el proceso de onboarding de un banco en línea, Amazon Rekognition puede desempeñar un papel crucial en la verificación de identidad de los clientes mediante el reconocimiento facial. Aquí hay una descripción a nivel arquitectónico de cómo podría integrarse Amazon Rekognition en este proceso:

- **Captura de imágenes faciales:** En el flujo de registro, se solicitaría a los clientes que proporcionen una fotografía de su rostro. Esta imagen podría capturarse mediante una cámara web integrada en la aplicación móvil o el sitio web del banco, o a través de la carga de una imagen existente desde el dispositivo del usuario.

- **Envío de la imagen a Amazon Rekognition:** Una vez que se ha capturado la imagen facial del cliente, se enviaría a Amazon Rekognition a través de una llamada a la API de Amazon Rekognition desde el backend del sistema bancario. La imagen se transmitiría de forma segura utilizando HTTPS para garantizar la privacidad y la seguridad de los datos del cliente.

- **Procesamiento de la imagen:** En el lado de Amazon Rekognition, el servicio utilizaría sus algoritmos de aprendizaje profundo para analizar la imagen facial y extraer características únicas, como la forma de la cara, la posición de los ojos, la nariz y la boca, así como otras características faciales distintivas.

- **Comparación con la base de datos de clientes:** Una vez que Amazon Rekognition ha procesado la imagen facial del cliente, el servicio puede compararla con una base de datos de imágenes faciales almacenadas de clientes existentes. Esta base de datos podría mantenerse en un repositorio seguro en la nube o en una base de datos local del sistema bancario.

- **Verificación de la identidad:** Basándose en la comparación de la imagen facial del cliente con las imágenes almacenadas en la base de datos, Amazon Rekognition puede proporcionar una puntuación de similitud que indica cuánto se parece la imagen del cliente a las imágenes existentes en la base de datos. Si la puntuación de similitud supera un umbral predefinido, se puede considerar que la identidad del cliente ha sido verificada exitosamente.

- **Notificación del resultado al sistema bancario:** Una vez completada la verificación de la identidad por parte de Amazon Rekognition, se enviaría un mensaje o respuesta de vuelta al sistema bancario indicando el resultado de la comparación facial. Esto permitiría al sistema bancario continuar con el proceso de registro del cliente o solicitar una verificación adicional si es necesario.

En resumen, Amazon Rekognition puede integrarse de manera efectiva en el proceso de onboarding de un banco en línea para proporcionar una autenticación biométrica segura y precisa utilizando el reconocimiento facial. Esto ayuda a mejorar la seguridad y la experiencia del cliente al mismo tiempo que simplifica el proceso de registro y verificación de identidad.

6 Escalabilidad y Disponibilidad

La escalabilidad y la disponibilidad son aspectos críticos en la arquitectura de un banco en línea, ya que deben garantizar que el sistema pueda manejar un gran volumen de transacciones y estar disponible en todo momento para los clientes. Aquí hay una explicación más detallada de estos conceptos en el contexto de un banco:

Escalabilidad:

Vertical: La escalabilidad vertical implica aumentar la capacidad de los recursos individuales, como agregar más memoria, CPU o almacenamiento a un servidor existente. En un banco en línea, esto podría significar actualizar los servidores de base de datos para manejar un mayor número de consultas o mejorar los servidores web para manejar más solicitudes de clientes simultáneas.

Horizontal: La escalabilidad horizontal implica agregar más instancias de servidores o servicios para distribuir la carga entre ellos. En un banco en línea, esto podría implicar la implementación de clústeres de servidores web y bases de datos distribuidas que pueden escalar automáticamente según la demanda. También puede incluir la implementación de microservicios para desacoplar las funciones y permitir la escalabilidad independiente de cada servicio.

Disponibilidad:

Redundancia: Para garantizar la disponibilidad, es fundamental tener redundancia en todos los niveles del sistema, desde la infraestructura física hasta la lógica de la aplicación. Esto incluye implementar múltiples servidores, centros de datos geográficamente dispersos y replicación de datos en tiempo real para evitar puntos únicos de falla.

Balanceo de carga: El balanceo de carga distribuye el tráfico de red entre múltiples servidores para evitar la congestión y garantizar un rendimiento óptimo. Esto se logra mediante el uso de dispositivos de equilibrio de carga o software de equilibrio de carga que distribuye las solicitudes de los clientes entre los servidores disponibles de manera equitativa.

Failover automático: Implementar mecanismos de failover automático garantiza que si un servidor o servicio falla, la carga se redirija automáticamente a un servidor o servicio secundario sin interrupción del servicio para los usuarios finales. Esto puede lograrse mediante la configuración de clústeres de alta disponibilidad y utilizando herramientas de orquestación y monitoreo automatizado.

En resumen, la escalabilidad y la disponibilidad en un banco en línea implican asegurar que el sistema pueda manejar un crecimiento orgánico del negocio y mantenerse disponible en todo momento para satisfacer las necesidades de los clientes. Esto se logra mediante la implementación de estrategias de escalabilidad vertical y horizontal, así como redundancia, balanceo de carga y failover automático para garantizar un servicio continuo y confiable.

7 Atributos de calidad

En el diseño arquitectónico de un banco en línea, la priorización de los atributos de calidad, como seguridad, mantenibilidad y eficiencia de ejecución, es fundamental para garantizar un servicio confiable y de alto rendimiento para los clientes. A continuación, se detallan las razones por las cuales se priorizan estos atributos y las ventajas que ofrecen:

Seguridad:

Razón de la priorización: La seguridad es de suma importancia en un entorno financiero, donde se manejan datos confidenciales y transacciones monetarias. La priorización de la seguridad garantiza la protección de la información del cliente, previene el fraude y asegura el cumplimiento de las regulaciones gubernamentales.

Ventajas:

- Protección de datos confidenciales del cliente.
- Prevención de accesos no autorizados y actividades maliciosas.
- Cumplimiento de regulaciones de seguridad y privacidad, como GDPR y PCI DSS.
- Generación de confianza y fidelidad del cliente

Mantenibilidad:

Razón de la priorización: La mantenibilidad se refiere a la facilidad con la que el sistema puede ser modificado, actualizado y mantenido a lo largo del tiempo. En un entorno bancario, donde se realizan constantes cambios regulatorios y de negocio, la mantenibilidad es crucial para garantizar la adaptabilidad y la evolución continua del sistema.

Ventajas:

- Facilidad para realizar cambios y mejoras en el sistema sin afectar su estabilidad.
- Reducción del tiempo y costos asociados con el mantenimiento y la resolución de problemas.
- Mejora en la capacidad de respuesta a los cambios regulatorios y de mercado.
- Facilita la incorporación de nuevas funcionalidades y tecnologías emergentes.

Eficiencia de ejecución:

Razón de la priorización: La eficiencia de ejecución se refiere a la capacidad del sistema para utilizar los recursos de manera óptima y lograr un alto rendimiento en términos de velocidad y capacidad de respuesta. En un entorno bancario en línea, donde se manejan grandes volúmenes de transacciones y se requiere una respuesta rápida a las consultas de los clientes, la eficiencia de ejecución es esencial para garantizar una experiencia de usuario fluida.

Ventajas:

- Mejora en el rendimiento del sistema y tiempos de respuesta más rápidos.
- Optimización de recursos, como CPU, memoria y ancho de banda de red.
- Reducción de los costos operativos al minimizar el uso de recursos de infraestructura.
- Mejora en la capacidad de escalar vertical y horizontalmente para manejar picos de carga.

8 Patrones de diseño

La priorización de los patrones de arquitectura, como Event Driven, Microservices, CQRS, Hexagonal y Domain Driven Design (DDD), en el diseño de la arquitectura de un banco se basa en varias razones fundamentales que abordan aspectos clave del sistema y proporcionan ventajas significativas en términos de escalabilidad, mantenibilidad, flexibilidad y rendimiento. A continuación, se explican las razones por las que se priorizaron estos patrones:

Event Driven:

Razón de la priorización: El patrón **Event Driven** permite la creación de sistemas altamente escalables y flexibles al permitir que los componentes del sistema reaccionen a eventos y se comuniquen de manera asíncrona. En un entorno bancario, donde se generan eventos constantemente, como transacciones, pagos y notificaciones, este enfoque permite una gestión eficiente de la información y una arquitectura más adaptable a cambios y nuevas funcionalidades.

Ventajas:

- Desacoplamiento entre componentes del sistema.
- Escalabilidad horizontal al permitir la distribución de cargas de trabajo.
- Mejora en la capacidad de respuesta y la eficiencia del sistema.
- Facilita la integración con sistemas externos y la implementación de procesos asincrónicos.

Microservices:

Razón de la priorización: El enfoque de **Microservices** divide la aplicación en un conjunto de servicios independientes y autónomos, cada uno con su propio contexto de negocio y funcionalidad. En el contexto bancario, donde se manejan diversas operaciones y funcionalidades complejas, los Microservices permiten una mayor modularidad, escalabilidad y evolución independiente de cada componente del sistema.

Ventajas:

- Flexibilidad para desarrollar, desplegar y escalar servicios de forma independiente.
- Facilita la implementación de equipos ágiles y la adopción de metodologías DevOps.
- Mejora en la resiliencia y tolerancia a fallos al aislar funcionalidades críticas.

Permite una gestión más eficiente de recursos y una mayor agilidad en el desarrollo y la entrega de software.

CQRS (Command Query Responsibility Segregation):
Razón de la priorización: **CQRS** separa la lógica de escritura (comandos) de la lógica de lectura (consultas), lo que permite optimizar cada una de ellas de manera independiente. En un banco en línea, donde se realizan muchas consultas de datos por parte de los clientes y operaciones de escritura en el sistema, este enfoque mejora el rendimiento, la escalabilidad y la consistencia de la información.

Ventajas:

- Mejora en el rendimiento al optimizar consultas y operaciones de escritura por separado.
- Flexibilidad para escalar y ajustar cada componente según las necesidades del negocio.
- Facilita la implementación de modelos de dominio complejos y la gestión de transacciones distribuidas.
- Mejora en la capacidad de adaptarse a cambios y evolucionar el sistema con el tiempo.

Hexagonal Architecture:

Razón de la priorización: La **arquitectura hexagonal**, también conocida como Puertos y Adaptadores, se centra en la separación de las preocupaciones y la creación de sistemas más mantenibles y testables. En un banco en línea, donde se requiere una arquitectura modular y flexible, este enfoque facilita la integración con sistemas externos, la implementación de lógica de negocio y la gestión de la interfaz de usuario de manera independiente.

Ventajas:

- Desacoplamiento entre la lógica de negocio y la infraestructura técnica.
- Facilita la prueba unitaria y la implementación de pruebas automatizadas.
- Permite la reutilización de componentes y la sustitución de tecnologías sin afectar la lógica de negocio.
- Mejora en la mantenibilidad, la escalabilidad y la evolución del sistema a largo plazo.

Domain Driven Design (DDD):

Razón de la priorización: **DDD** se centra en el diseño de software alrededor del dominio del negocio, identificando y modelando entidades, agregados, servicios y contextos delimitados. En un banco en línea, donde el conocimiento del negocio es fundamental, DDD ayuda a alinear la arquitectura del sistema con los conceptos y procesos del dominio bancario, lo que facilita la comprensión, la colaboración y la evolución del sistema.

Ventajas:

- Mejora en la comunicación y la colaboración entre el equipo de desarrollo y los expertos del dominio.
  Mayor alineación entre el diseño del software y los conceptos del negocio.
- Facilita la identificación y modelado de conceptos complejos y relaciones dentro del dominio.
- Permite una evolución más natural y adaptativa del sistema a medida que cambian las necesidades del negocio.

9 Costos estimados

Team Account

Cargo	Salario promedio
Mobile Developer (Flutter)	3000
Mobile Developer (Flutter)	3000
Frontend Developer (React)	3000
Backend Developer	3300

Backend Developer	3300
Backend Developer	3300
Product Owner	3500
Tech Lead	3800

Team SingUp

Cargo	Salario promedio
Mobile Developer (Flutter)	3000
Mobile Developer (Flutter)	3000
Frontend Developer (React)	3000
Backend Developer	3300
Backend Developer	3300
Backend Developer	3300
Product Owner	3500
Tech Lead	3800

Team CashFlow

Cargo	Salario promedio
Mobile Developer (Flutter)	3000
Mobile Developer (Flutter)	3000
Frontend Developer (React)	3000
Backend Developer	3300
Backend Developer	3300
Backend Developer	3300
Product Owner	3500
Tech Lead	3800

Team User

Cargo	Salario promedio
Frontend Developer (React)	3000
Backend Developer	3300
Backend Developer	3300
Backend Developer	3300
Product Owner	3500
Tech Lead	3800

Team Client

Cargo	Salario promedio
Frontend Developer (React)	3000

Backend Developer	3300
Backend Developer	3300
Backend Developer	3300
Product Owner	3500
Tech Lead	3800

Team Data

Cargo	Salario promedio
Frontend Developer (React)	3000
Backend Developer	3300
Backend Developer	3300
Backend Developer	3300
Product Owner	3500
Tech Lead	3800

Team Integration

Cargo	Salario promedio
Frontend Developer (React)	3000
Backend Developer	3300
Backend Developer	3300
Backend Developer	3300
Product Owner	3500
Tech Lead	3800

Team Bouncer

Cargo	Salario promedio
Frontend Developer (React)	3000
Backend Developer	3300
Backend Developer	3300
Backend Developer	3300
Product Owner	3500
Tech Lead	3800

Team Core

Cargo	Salario promedio
Frontend Developer (React)	3000
Backend Developer	3300
Backend Developer	3300
Backend Developer	3300

Product Owner	3500
Tech Lead	3800

Team Notification

Cargo	Salario promedio
Frontend Developer (React)	3000
Backend Developer	3300
Backend Developer	3300
Backend Developer	3300
Product Owner	3500
Tech Lead	3800

Team AuditGuard

Cargo	Salario promedio
Frontend Developer (React)	3000
Backend Developer	3300
Backend Developer	3300
Backend Developer	3300
Product Owner	3500
Tech Lead	3800

Team General

Cargo	Salario promedio
Architect	4000
Architect	4000
Architect	4000
DevOps Engineer	3800
DevOps Engineer	3800
DevOps Engineer	3800

El gasto promedio de equipo de desarrollo es 244 000 al mes y 3 660 000 al Año.

Costos de infraestructura

Como arquitecto de software, comprendo la importancia de proporcionar estimaciones precisas y basadas en datos concretos. En este caso, no puedo ofrecer un presupuesto sin información específica sobre la infraestructura y las métricas relevantes. Sin embargo, puedo destacar algunos puntos clave:

Documento de Despliegue de Infraestructura:

El documento de despliegue de infraestructura es fundamental para comprender la arquitectura, los componentes y los recursos necesarios para implementar el sistema.Debería incluir detalles sobre servidores, redes, bases de datos, servicios en la nube y cualquier otro recurso relevante.

Métricas de Rendimiento:
Para proporcionar un presupuesto preciso, necesitamos métricas concretas:

Usuarios del Sistema: Cuántos usuarios se espera que utilicen el sistema. Esto afecta la capacidad de los servidores y la escalabilidad.

Carga Promedio: La cantidad de solicitudes o transacciones que el sistema debe manejar en un período de tiempo determinado (por ejemplo, solicitudes por segundo).

Patrones de Uso: ¿Hay horas pico o momentos de mayor actividad? Esto afecta la planificación de recursos.

Escalabilidad y Elasticidad:
La infraestructura debe diseñarse para manejar cargas variables. La escalabilidad vertical (más recursos en una sola máquina) y la escalabilidad horizontal (más máquinas) son consideraciones importantes. La elasticidad permite ajustar automáticamente los recursos según la demanda.

Costos Asociados:
El presupuesto también debe considerar los costos operativos, como licencias, mantenimiento, alojamiento y servicios en la nube.