Arquitectura del Sistema

De Banca por Internet

Cristian Patricio Armijo Vaca

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

Tabla de contenido

Arquitectura del Sistema de Banca por Internet

Introducción

Este sistema tiene como objetivo proporcionar a los clientes de BP una plataforma segura y eficiente para gestionar sus cuentas bancarias en línea, incluyendo consultas de saldos, transferencias y pagos

Propósito

Garantizar una arquitectura, alta disponibilidad (HA), tolerancia a fallos, recuperación ante desastres (DR), Seguridad y Monitoreo, Excelencia operativa y auto-healing.

Alcance

Consulta de saldos: Permitir a los usuarios verificar el saldo de sus cuentas en tiempo real.

Histórico de transacciones: Mostrar un registro detallado de todas las transacciones realizadas.

Transferencias: Facilitar transferencias entre cuentas propias y a terceros.

Pagos: Permitir el pago de servicios, compras en línea, etc.

Recargas: Facilitar la recarga de teléfonos móviles y otras tarjetas prepago.

Solicitudes de productos: Permitir a los usuarios solicitar nuevos productos o servicios (tarjetas de crédito, préstamos).

Configuración de perfiles: Permitir a los usuarios actualizar su información personal y configurar preferencias.

Tecnología y Plataformas

Aplicación Front-end

Justificación Angular es robusto y adecuado para aplicaciones empresariales complejas, mientras que React Native permite desarrollo multiplataforma eficiente para iOS y Android

- SPA:Angular
- Aplicación móvil:React Native

Autenticación

Se recomienda usar el flujo de "Authorization Code" con PKCE (Proof Key for Code Exchange) de OAuth 2.0. Este flujo es seguro para aplicaciones móviles y SPAs, y previene ataques de intercepción de código.

Seguridad y Cumplimiento Normativa

Onboarding yAutenticación Biométrica

Usar AWS Rekognition o Azure Face API para el reconocimiento facial.

Implementar autenticación biométrica (huella digital) usando bibliotecas nativas de iOS (Touch ID) y Android (Fingerprint API).

Arquitectura de Microservicios

API Gateway: AWS API Gateway o Azure API Management

Servicios: Implementados como contenedores Docker orquestados con Kubernetes

Base de Datos de Auditoría: Amazon Aurora o Azure SQL Database

Caché: Amazon ElastiCache (Redis) o Azure Cache for Redis

Patrones de Diseño

Patrón Repository para acceso a datos

Patrón Adapter para integración con sistemas externos

Patrón Circuit Breaker para manejar fallos en servicios externos

Alta Disponibilidad y Tolerancia a Fallos

Despliegue en múltiples zonas de disponibilidad

Uso de balanceadores de carga

Implementación de auto-scaling para servicios

Replicación de bases de datos

Recuperación ante Desastres:

Replicación de datos entre regiones

Implementación de backups regulares

Plan de failover automatizado

Seguridad

Encriptación de datos en tránsito y en reposo

Implementación de WAF (Web Application Firewall)

Uso de VPCs y Security Groups

Implementación de políticas de least privilege

Monitoreo

Uso de Amazon CloudWatch o Azure Monitor

Implementación de logging centralizado con ELK stack

Uso de APM (Application Performance Monitoring) como New Relic o Datadog

Consideraciones Normativas

GDPR o leyes locales de protección de datos personales

PCI DSS para manejo de información financiera

Normativas específicas del sector bancario del país

Integración y Desacoplamiento

Uso de colas de mensajes (Amazon SQS o Azure Service Bus) para comunicación asíncrona entre servicios

Implementación de API Gateway para desacoplar clientes de servicios backend

Uso de eventos (Amazon EventBridge o Azure Event Grid) para notificaciones y actualizaciones en tiempo real

Costos

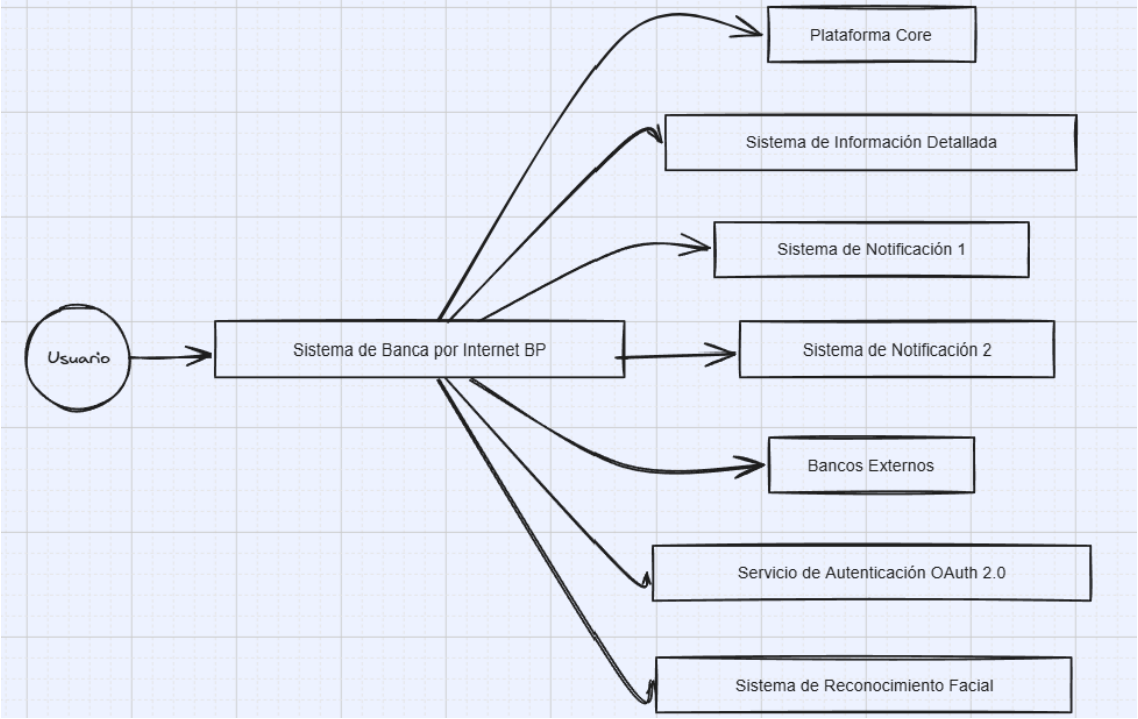
Uso de instancias reservadas para servicios base

Implementación de auto-scaling para optimizar costos en horas pico

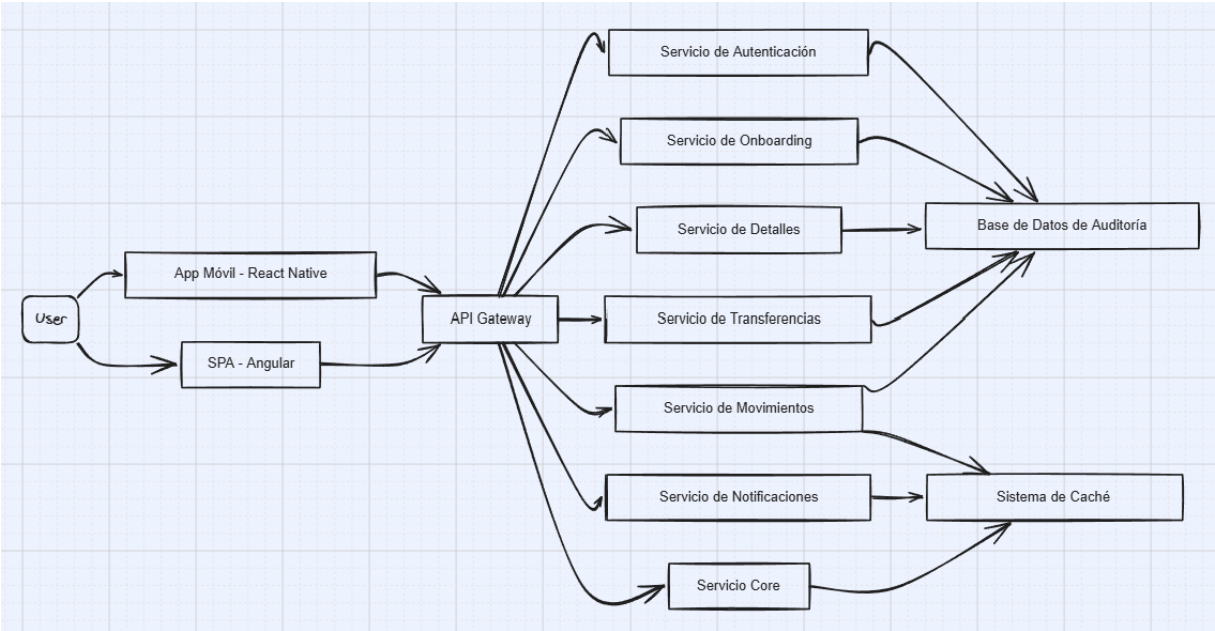
Uso de servicios serverless donde sea posible para reducir costos operativos

Diagramas

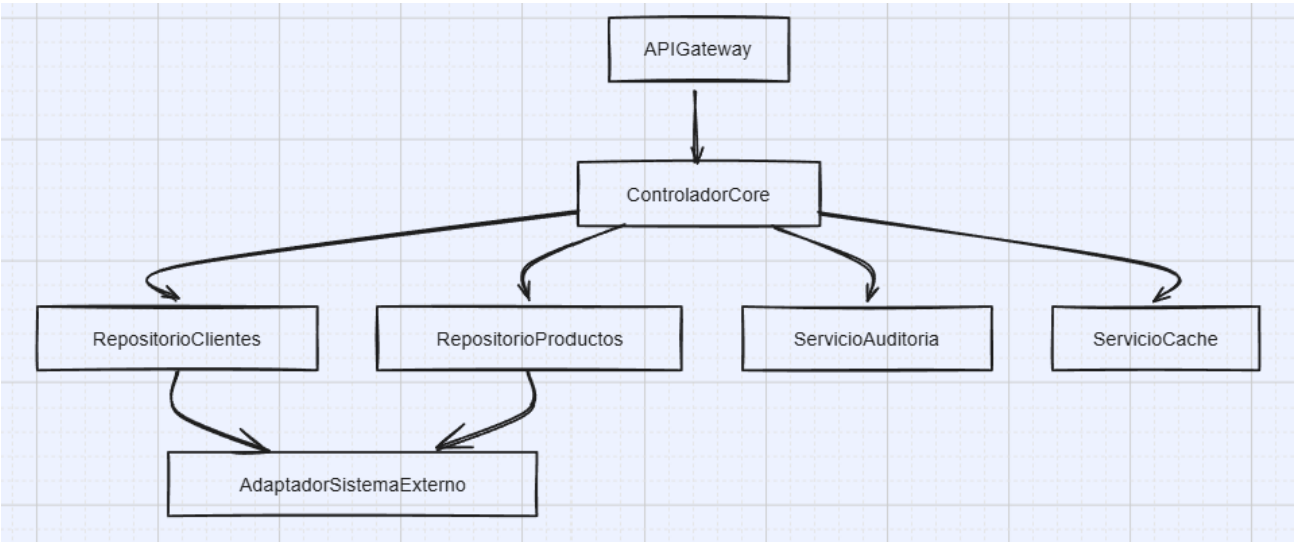
Modelo de Contexto



Modelo de Componentes



Modelo de Componentes



Criterio de calificación

Que la solución satisfaga los requerimientos La arquitectura propuesta cubre todos los requerimientos especificados, incluyendo el acceso a histórico de movimientos, realización de transferencias y pagos, integración con sistemas externos, notificaciones, aplicaciones front-end (SPA y móvil), autenticación OAuth 2.0, reconocimiento facial para onboarding, y base de datos de auditoría. Vista de Implementación

Calidad y profundidad de los diagramas (Contexto, Contenedores y Componentes):

Se proporcionaron tres diagramas detallados utilizando el modelo C4

Diagrama de Contexto: Muestra la interacción del sistema con usuarios y sistemas externos.

Diagrama de Contenedores: Detalla los principales componentes del sistema y sus interacciones.

Diagrama de Componentes: Se enfoca en el Servicio Core, mostrando sus componentes internos.

Segmentación de Responsabilidades y Desacoplamiento:

La arquitectura utiliza microservicios (CoreService, MovementService, TransferService, DetailService) para segmentar responsabilidades. El uso de un API Gateway desacopla los clientes de los servicios backend. La implementación de colas de mensajes y eventos para comunicación asíncrona mejora aún más el desacoplamiento.

Uso de patrones de arquitectura:

Se mencionan varios patrones de diseño, incluyendo Repository, Adapter, Circuit Breaker, y CQRS. Además, la arquitectura general sigue el patrón de microservicios.

Integración con servicios externos:

La solución integra varios servicios externos, incluyendo la Plataforma Core, el Sistema de Información Detallada, sistemas de notificación, y bancos externos para transferencias interbancarias.

Calidad de arquitectura de aplicación front-end y móvil:

Se propone usar Angular para la SPA y React Native para la aplicación móvil, justificando estas elecciones. La arquitectura considera la interacción de estas aplicaciones con el backend a través del API Gateway.

Arquitectura de acceso a datos:

Se utiliza el patrón Repository para el acceso a datos, con servicios específicos (CoreService, MovementService, etc.) manejando diferentes tipos de datos. También se implementa un sistema de caché (Redis) para mejorar el rendimiento.

Conocimientos de Nube (AWS o Azure):

La solución propone el uso de servicios específicos de AWS o Azure, como AWS API Gateway/Azure API Management, Amazon Aurora/Azure SQL Database, y Amazon ElastiCache/Azure Cache for Redis, demostrando conocimiento de ambas plataformas.

Manejo de costos:

Se mencionan estrategias para optimizar costos, incluyendo el uso de instancias reservadas, auto-scaling, y servicios serverless donde sea posible.

Arquitectura de Autenticación:

Se recomienda el flujo de "Authorization Code" con PKCE de OAuth 2.0, explicando por qué es apropiado para este caso.

Arquitectura de Integración con Onboarding:

Se propone el uso de servicios como AWS Rekognition o Azure Face API para el reconocimiento facial en el proceso de onboarding, integrándolo con el flujo de autenticación.

Diseño de Solución de Auditoría:

Se incluye una Base de Datos de Auditoría específica, con todos los servicios registrando acciones en ella. Esto permite un seguimiento completo de las actividades del usuario.

Conocimientos de regulaciones bancarias y estándares de seguridad:

Se mencionan consideraciones normativas como GDPR, PCI DSS, y la necesidad de cumplir con regulaciones específicas del sector bancario.

Implementación de Alta Disponibilidad y Tolerancia a Fallos:

La arquitectura incluye despliegue en múltiples zonas de disponibilidad, uso de balanceadores de carga, auto-scaling, y replicación de bases de datos. También se menciona un plan de recuperación ante desastres.

Implementación de Monitoreo:

Se propone el uso de herramientas como Amazon CloudWatch o Azure Monitor, logging centralizado con ELK stack, y APM con herramientas como New Relic o Datadog.

Propuesta (opcional)

Servicio de Análisis Predictivo:

- *Este servicio utilizará machine learning para analizar los patrones de gasto y comportamiento financiero de los clientes, ofreciendo recomendaciones personalizadas y alertas proactivas.*

Servicio de Agregación de Cuentas:

- Este servicio permitirá a los clientes ver todas sus cuentas bancarias, incluso de otros bancos, en una sola interfaz, mejorando la experiencia del usuario.

Servicio de Chatbot:

Un servicio de chatbot basado en IA para proporcionar soporte al cliente 24/7, responder preguntas frecuentes y guiar a los usuarios en el uso de la plataforma.

Servicio de Optimización de Rendimiento:

Este servicio se encargará de optimizar dinámicamente el rendimiento de la aplicación, ajustando la caché, las consultas a la base de datos y la distribución de carga en tiempo real.

Servicio de Gestión de Consentimientos

Este servicio gestionará los consentimientos de los usuarios para el uso de sus datos, cumpliendo con las regulaciones de protección de datos y permitiendo a los usuarios controlar qué información comparten.

Repositorio

https://github.com/arvac/Arquitectura-del-Sistema-para-banco-Cristian-Armijo.git

Referencias