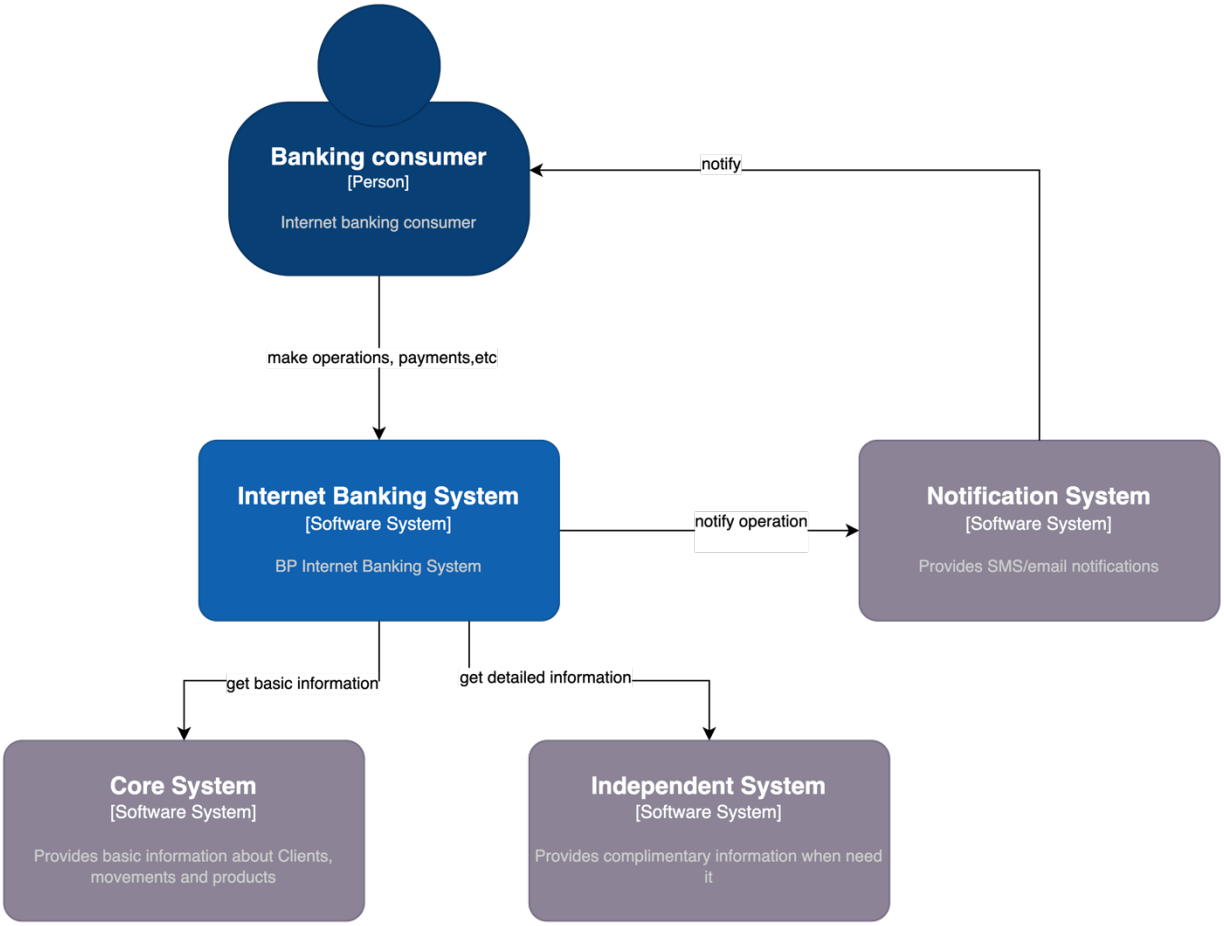
EJERCICIO PRACTICO

LUIS TORRES ZAMBRANO

Diseño de Arquitectura Banco BP

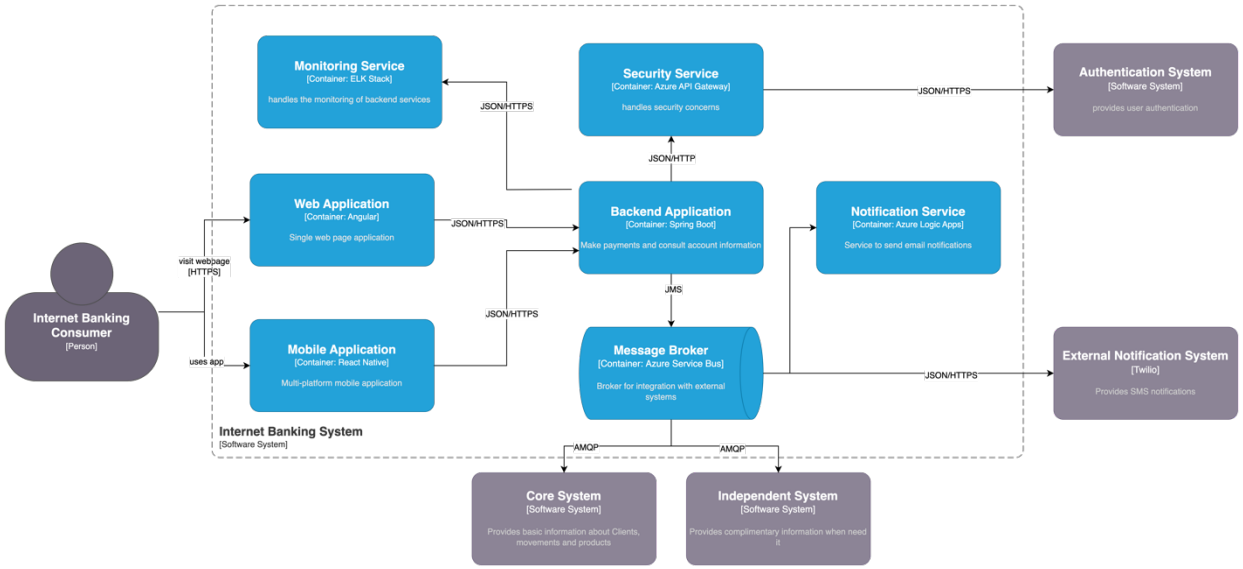
Diagrama de contexto-C1

Se muestra el sistema que soporta los requerimientos funcionales y no funcionales.



[System Context] Internet Baking System

Diagrama de contenedores- C2



Se describe cada contenedor:

1. Web Application

Este contenedor, que es una SPA, será el front-end a la que accedan los clientes de Banca por internet. Esta desarrollado usando el framework Angular debido a que ofrece un buen rendimiento (aunque no el mejor comparado con otros frameworks) y es el que ofrece más ventajas en modularidad (p.e. su arquitectura está basada en componentes lo que promueve la reutilización de componentes UI) y facilidad de desarrollo (p.e. una CLI para crear proyectos y componentes).

2. Mobile Application

Este contenedor es una aplicación móvil instalada en los smartphones de los clientes del Banco. Para decidir que tecnología usar para desarrollar este contenedor se van a comparar 3 frameworks multi-plataforma que utilizan TypeScript (con el objetivo de aprovechar las capacidades del equipo). Tenemos React Native, NativeScript y Ionic.

Feature	React Native	NativeScript	Ionic
Performance	Rendimiento alto, cercano al nativo Utiliza componentes nativos Adecuado para aplicaciones	Alto, similar a React Native Acceso directo a API nativas Ideal para aplicaciones que	Inferior a React Native y NativeScript: usa WebView para renderizar la interfaz de usuario optimizado para aplicaciones menos

	complejas y de alto rendimiento	requieren rendimiento nativo	críticas para el rendimiento
Ecosistema	Ecosistema maduro con una amplia gama de bibliotecas de terceros fuerte integración con servicios de back-end y herramientas de terceros	Ecosistema en crecimiento con soporte para varios complementos fuerte integración con Angular y Vue.js	Amplio ecosistema, especialmente para tecnologías web Componentes para mejora las capacidades de integración nativas
Comunidad	Comunidad grande y activa Fuerte apoyo de Meta (Facebook) Amplia documentación y tutoriales disponibles	Comunidad activa pero más pequeña que la de React Native. Buena documentación y apoyo de la comunidad	Comunidad grande y activa Fuerte apoyo del equipo de Ionic Amplios recursos y mercado para componentes y complementos
Funcionalidades nativas	Alto: conecta JavaScript con API nativas. requiere algunos conocimientos de código nativo para funciones complejas	Alto: Acceso directo a API nativas Admite componentes de interfaz de usuario nativos e integración de hardware de dispositivos	Moderado:Limitado por WebView para funciones nativas más complejas

De la tabla anterior, la herramienta que destaca en los 4 criterios es React Native, principalmente en la parte de funcionalidades nativas ya que la aplicación necesita usar recursos nativos como la cámara u otra interfaz biométrica para el proceso de Onboarding. Además, se usará el lenguaje TypeScript que ayuda a que el código sea mas limpio y ordenado, lo cual impacta positivamente en la mantenibilidad de nuestra solución.

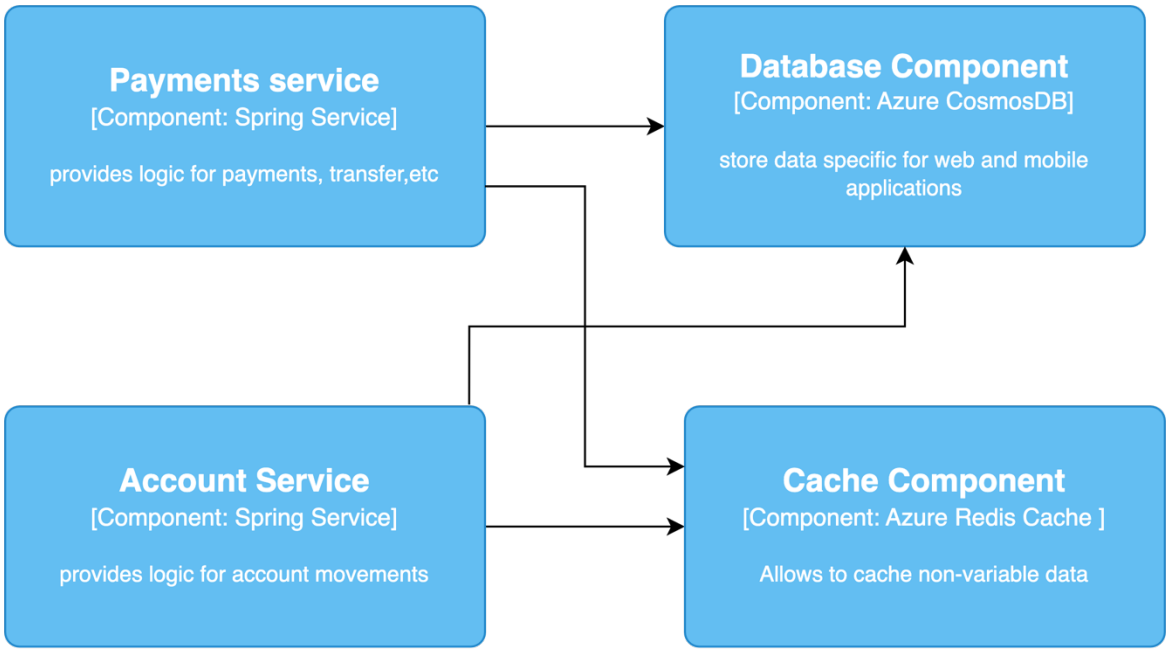
3. Backend Application

Este contenedor contiene la lógica de negocio de la solución, ejecutar pagos y transferencias y consultar movimientos. Además, se comunica con los 2 sistemas actuales(core platform y independent system) y con los sistemas de notificación de operaciones.

A pesar de que el sistema no tiene aún muchas funcionalidades, estas se irán incrementando debido a la gran cantidad de clientes del banco. Además, debido a la gran cantidad y variedad de productos bancarios, cada uno con su propia lógica de negocio, se trata de un sistema complejo. Por tanto, se usa una arquitectura de Microservicios, que nos brinda flexibilidad y escalabilidad.

Se usa como framework backend Spring boot ya que provee muchas ventajas en una arquitectura de Microservicios como configuración distribuida, service discovery, etc.

a. Diagrama de componentes C3



[Components] Backend Service
provides business logic

Para soportar las 3 funcionalidades del sistema: acceder al histórico de movimientos, realizar transferencias y realizar pagos se crearon 2 servicios:

- Account Service que maneja las información de las cuentas del usuario
- Payment Service que maneja la operaciones de pago, como transferencias y pagos.
- Estos servicios usaran un componente de base de datos para almacenar información especifica de los aplicativos web y móvil, ya que la información restante se obtendrá de los sistemas existentes.
- Además, se usa un componente de cache para incrementar la performance de las aplicaciones.

4. Message Broker

Este contenedor sirve para desacoplar la comunicación entre el backend y los sistemas externos. Se usa Azure Service Bus debido a sus múltiples opciones de integración(p.e. AMQP) además de las opciones avanzadas de mensajería (p.e. dead letter queues)

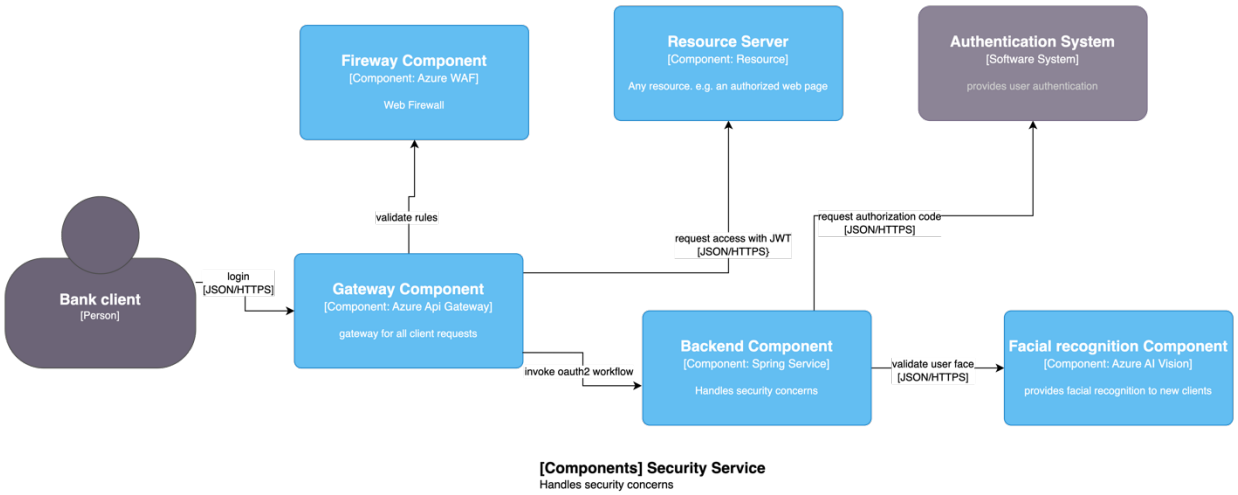
5. Security Service

Este contenedor sirve para manejar la autenticación y autorización de los usuarios. Ya que el banco cuenta con sistema propio de autenticación, solo implementaremos el flujo de autorización usando Oauth2. De todos los flujos (grant type) que maneja este framework usaremos el más seguro de ellos, ya que nuestro sistema maneja el dinero de los clientes. Además, lo ideal es usar un mecanismo que proteja frente a ataques como code injection (p.e. CSRF). Por tanto, vamos a usar una extensión al grant type Authorization Code llamado PKCE.

En este flujo nuestra aplicación recibirá un authorization code desde el servicio de autenticación(externo) que será cambiado por un token JWT, que será usado para autorizar a los clientes dependiendo de los claims que este maneje.

En el caso de la aplicación móvil, se accede a las APIs nativas del dispositivo para realizar la autenticación biométrica. Luego se continua con el flujo de autorización, haciendo que la aplicación backend obtenga el authorization code.

a. Diagrama de componentes -C3



b. Proceso de Onboarding

El proceso de onboarding de clientes nuevos requiere de autenticación biométrica, el cual solo se puede realizar en la aplicación móvil. Se podría delegar este proceso a los dispositivos móviles pero cada marca maneja un proceso diferente de autenticación biométrica lo cual dificulta tener un proceso standard en nuestro onboarding. Por ello, se integrará el actual

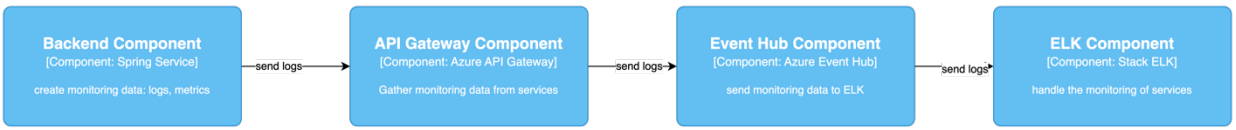
componente de autenticación Oauth2 con el servicio Azure AI Vision, el cual provee reconocimiento de imágenes(p.e. reconocimiento facial). Los pasos son:

- El usuario accede al formulario de registro
- Se invoca al Authentication Server para que el usuario se registre con su id y password
- Al ser la primera vez que se registra, el backend redirecciona hacia la url para hacer la verificación facial.
- Azure AI Vision SDK ayuda a la aplicación a iniciar la cámara y guiar al usuario a colocarse en la posición correcta.
- El cliente toma la foto y la aplicación la envía al servicio de reconocimiento de Azure para detectar si la foto es válida.
- Si la imagen es válida, el backend la registra asociada al id del usuario para utilizarla la siguiente vez que el usuario se autentique.

6. Monitoring Service

La estrategia de monitoreo se implementara a nivel del contenedor backend, el cual incluye Logging de eventos, Tracing de invocaciones a servicios y Metrics de funcionamiento de los servicios. Todos los logs serán centralizados en el API gateway usando un componente llamado Logger. Posteriormente los logs serán enviados al stack ELK(Elastic Search, Logstash y KIbana), el cual será usado como herramienta de Monitoreo.

a. Diagrama de componentes - C3



[Components] Monitoring
Monitoring of services

7. Notification Service

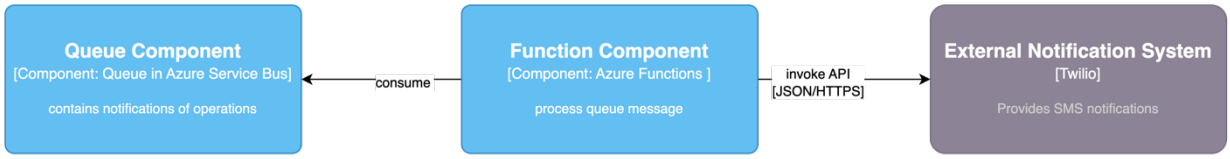
Se usan notificaciones por email y SMS para asegurar el lectura de los mensajes enviados. Además se usan 2 proveedores diferentes de notificaciones como estrategia de contingencia en caso uno de los proveedores presente problemas.

Para el envió por email se usara un componente interno como Azure Logic Apps

Para el envió de SMS's se usara un sistema externo como Twilio.

Ambos métodos de notificación se comunican con el backend usando el Message broker.

a. Diagrama de componentes - C3



[Components] External Notification System
notifies by SMS

Para el envío de SMS usando el sistema externo Twilio se usara un Azure function para leer el mensaje del topic en el service bus y enviarlo a Twilio.

Requerimientos transversales

Estrategia de Disaster Recovery DRP

Para DRP tenemos 4 posibles estrategias:

• Backup and Restore (RPO horas, RTO horas)

• Pilot Light (RPO minutos, RTO horas)

• Warm Standby (RPO segundos, RTO minutos)

• Activo-Activo (RPO segundo/instantáneo, RTO segundo)

Debido a que el tiempo de recuperación RTO de horas es alto para una aplicación de la que dependen usuarios para administrar su dinero, Backup and Restore queda descartado.

La estrategia Pilot Light implica mantener una versión mínima en la nube , con la data critica replicada. Sin embargo, el RTO sigue siendo alto debido a que vamos a necesitar mover los datos de la "plataforma core" y el "sistema independiente" de on-premise a la nube. Dependiendo de la cantidad de data almacenada este proceso puedo tardar horas. Por ello, también se descarta.

La estrategia Warm Standby implica mantener siempre ejecutándose una versión reducida del ambiente completo(on-premise y on-cloud) en la nube. En caso de un desastre, rápidamente se puede escalar para manejar la carga usual de trabajo. En este caso el RPO es mínimo y el RTO de minutos es aceptable para una entidad bancaria. Por ello, se usara esta estrategia debido a que es costo-efectivo y cumple con las regulaciones estatales.

La solución incluye el uso de los siguientes servicios:

- Azure Site Recovery (ASR): Para replicación de las cargas de trabajo on-premises y on-cloud.
- Azure Traffic Manager: Para el switch automático hacia el ambiente reducido.
- Azure Virtual Machines with Auto-scaling: Para mantener las instancias mínimas ejecutándose en el ambiente reducido de contingencia.
- Geo-Redundant Storage (GRS): Para asegurar disponibilidad de los recursos en diferentes regiones.

Estrategia de Seguridad de datos

1. Cifrado de Datos

- Cifrado en Tránsito: Asegura que los datos están cifrados mientras se mueven entre clientes, aplicaciones y servidores utilizando protocolos como TLS (Transport Layer Security).
- Cifrado en Reposo: Cifra los datos almacenados en la nube. En nuestro caso, se almacenaran datos encriptados en Azure CosmosDB y Azure Redis Cache.
- Gestión de Claves: Implementa una gestión segura de claves con soluciones como Azure Key Vault.

2. Clasificación y Etiquetado de Datos

- Clasificación de Datos Sensibles: Identifica y clasifica los datos sensibles, aplicando políticas de protección de datos basadas en su sensibilidad.
- Etiquetado de Datos: Implementa etiquetado automático de datos para aplicar controles específicos según el tipo de información (por ejemplo, datos de tarjetas de crédito, información personal).

3. Ofuscación de logs

La ofuscación de logs consiste en transformar o enmascarar partes de la información contenida en los registros para que no sean legibles o identificables directamente, manteniendo al mismo tiempo la utilidad del log para tareas de monitoreo, análisis, y solución de problemas.

4. Cifrado de contenido de APIs

El principal caso de uso del cifrado de contenido es garantizar la confidencialidad de los datos de extremo a extremo, a través de plataformas intermedias como API Gateways que, de otro modo, podrían permitir la captura, el registro o la inspección de datos en tránsito. Ejemplos de datos de contenido de APIs que deberían cifrarse son números de tarjetas de crédito, direcciones de correo, teléfono, etc.