EJERCICIO PRÁCTICO ARQUITECTURA DE SOLUCIONES

Autor: Paúl Jérez

Fecha de Elaboración: DMQ, 15 de septiembre de 2025

Contenido

Descripción del Ejercicio Sistema Bancario BP

Diseñe un sistema para una entidad bancaria que permita a los usuarios:

- ✓ Consultar su historial de transacciones.
- ✓ Realizar transferencias.
- ✓ Efectuar pagos entre cuentas propias e interbancarias.

Toda la información referente al cliente del sistema está compuesta por dos subsistemas:

- ✓ Una plataforma Core que contiene información básica del cliente (movimientos , productos).
- ✓ Sistema independiente que complementa la información del cliente cuando os datos requieren en detalle.

Debido a que la norma exige que los usuarios sean notificados sobre los movimientos realizados, el sistema utilizará sistemas externos o propios de envío de notificaciones, mínimo 2.

Este sistema contará con 2 aplicaciones en el Front, una SPA y una Aplicación móvil desarrollada en un Framework multiplataforma. (Mencione 2 opciones y justifique el porqué de su elección).

Ambas aplicaciones autenticarán a los usuarios mediante un servicio que usa el estándar Oauth2.0, para el cual no requiere implementar toda la lógica, ya que la compañía cuenta con un producto que puede ser configurado para este fin; sin embargo, debe dar recomendaciones sobre cuál es el mejor flujo de autenticación que se debería usar según el estándar.

Tenga en cuenta que el sistema de Onboarding para nuevos clientes en la aplicación móvil usa reconocimiento facial, por tanto, su arquitectura deberá considerarlo como parte del flujo de autorización y autenticación, a partir del Onboarding el nuevo usuario podrá ingresar al sistema mediante usuario y clave, huella o algún otro método especifique alguno de los anteriores dentro de su arquitectura, también puede recomendar herramientas de industria que realicen estas tareas y robustezca su aplicación.

El sistema cuenta con una base de datos de auditoría que registra todas las acciones realizadas por los clientes. Además, se implementa un mecanismo para persistir información de clientes frecuentes, optimizando el acceso y la experiencia.

Se sugiere utilizar un patrón de diseño alternativo que permita relacionar componentes que deben interactuar para cumplir un objetivo común.

La obtención de datos del cliente se realiza mediante una capa de integración que utiliza un API Gateway, el cual consume los servicios necesarios según el tipo de transacción.

Inicialmente, se utilizan tres servicios principales:

- Servicio de datos básicos
- Servicio de consulta de movimientos
- Servicio de transferencias

Se contempla la posibilidad de agregar más servicios para mejorar el rendimiento o enriquecer la información disponible para los clientes.

Consideraciones

Para este reto, mencione aquellos elementos normativos que considere importantes a tener en cuenta para entidades financieras.
Ejemplo: ley de protección de datos personales, seguridad, etc.

Garantice en su arquitectura, Alta disponibilidad (HA), Tolerancia a fallos, Recuperación ante desastres (DR), Seguridad y monitoreo, Excelencia operativa
Capacidades de auto-healing.

Si lo considera necesario, en su arquitectura puede incluir elementos de infraestructura en la nube, como Azure o AWS, garantizando Baja latencia, cuenta con presupuesto para esto.

En lo posible, plantee una arquitectura desacoplada, con componentes reusables y cohesionados, que permitan la incorporación de nuevos elementos en el futuro.

El modelo debe desarrollarse bajo el modelo C4: Modelo de Contexto, Modelo de Contenedores (Aplicación), de Componentes, de Código, describa hasta el modelo de componentes, la infraestructura la puede modelas como usted lo considere usando la herramienta de su preferencia.

1. Solución del Sistema Bancario BP

El sistema bancario BP ,lo hemos dividido en requerimientos Funcionales y No Funcionales el cual detallamos a continuación:

Funcionales:

- ✓ Consulta de historial de las transacciones.

  Justificación:

  Permite a los usuarios visualizar todas sus operaciones financieras, lo que facilita la toma de decisiones, la verificación de cargos y la detección de posibles fraudes. Esta funcionalidad también cumple con requerimientos regulatorios de transparencia bancaria.

- ✓ Realización de transferencias entre cuentas propias e interbancarias

  Justificación:

  El sistema debe permitir transferencias internas (entre productos del mismo cliente) y externas (a otros bancos), cumpliendo los tiempos y estándares definidos por la regulación financiera, como ISO 20022.

- ✓ Efectuar pagos electrónicos.

  Justificación:

  El sistema debe soportar pagos internos y externos, garantizando cumplimiento con la regulación financiera local y asegurando eficiencia y trazabilidad en las operaciones.

- ✓ Onboarding con reconocimiento facial.

  Justificación:

  El registro digital mediante biometría reduce fricción en la adquisición de clientes y mejora la experiencia de usuario, cumpliendo con las normas KYC (Know Your Customer) y estándares de seguridad.

- ✓ Notificaciones por lo menos dos canales.

  Justificación:

  El sistema debe informar a los usuarios sobre movimientos sensibles a través de al menos dos canales (por ejemplo, correo electrónico y SMS), cumpliendo regulaciones de seguridad y fortaleciendo la confianza del cliente.

- ✓ Acceso desde SPA web y app móvil.

  Justificación:

  Permite que los usuarios accedan a sus productos financieros desde cualquier dispositivo y en cualquier momento. La SPA (Single Page Application) asegura rapidez y eficiencia en desktop, mientras que la app móvil garantiza portabilidad y conveniencia para el uso diario.

No Funcionales:

- ✓ Alta disponibilidad, tolerancia a fallos, DRP.

  Justificación:

  El sistema debe estar disponible 24/7, soportar errores sin interrumpir el servicio y contar con un plan de recuperación ante incidentes, protegiendo la reputación del banco y evitando pérdidas económicas.

- ✓ Seguridad (OAuth2.0, biometría, cifrado, GDPR).

  Justificación:

  Se manejan datos personales y financieros altamente sensibles. Es obligatorio aplicar prácticas y estándares de seguridad como el cifrado en tránsito y en reposo, autenticación segura (OAuth2.0) y cumplir con normativas como GDPR, Habeas Data, ISO 27001, entre otros.

- ✓ Escalabilidad, bajo acoplamiento, alta cohesión.

  Justificación:

  La arquitectura debe soportar crecimiento en usuarios y funcionalidades sin necesidad de rediseños. El bajo acoplamiento permite actualizaciones independientes y la alta cohesión facilita mantenimiento y calidad del sistema.

- ✓ Observabilidad, trazabilidad, auditoría.

  Justificación:

  La trazabilidad es esencial para cumplimiento normativo (auditoría de acciones de usuarios) y para monitorear la salud de la aplicación. Permite detectar problemas, anomalías y realizar análisis post-mortem de incidentes.

2. Arquitectura C4

2.1.Diagrama de Contexto (Nivel 1)

Primeramente, vamos a detallar una matriz del diagrama de Contexto mediante las siguientes columnas con los nombres: Actor/Sistema, Rol Funcional , Tipo de interacción y las notas estrategias

Actor / Sistema	Rol Funcional	Tipo de Interacción	Notas Estratégicas
Cliente Bancario	Usuario final del sistema.	Interacción directa vía interfaz digital	Experiencia del usuario debe ser fluida y segura; clave para fidelización.
Servicio OAuth2.0	Autenticación y autorización.	Validación de credenciales	Seguridad robusta; permite escalabilidad y cumplimiento normativo.
Sistema Bancario Digital	Núcleo de operaciones digitales transaccionales.	Procesamiento de solicitudes del cliente.	Debe ser modular, escalable y altamente disponible.
Core Bancario	Consulta de datos básicos y sensibles del cliente.	Comunicación interna vía API	Integración eficiente; fuente confiable de datos esenciales.
Sistema Complementario	Consulta de datos detallados del cliente.	Comunicación interna vía API	Complementa al Core; permite personalización de servicios.
Servicio de SMS	Envío de notificaciones por mensaje de texto.	Comunicación unidireccional	Ideal para alertas críticas; útil en zonas con baja conectividad.
Servicio de Push	Envío de notificaciones push.	Comunicación unidireccional	Mejora la interacción en tiempo real; útil para marketing y alertas.
API Gateway	Orquestador de servicios externos	Enrutamiento y consumo de APIs	Centraliza la gestión de servicios; facilita monitoreo y seguridad.

Ilustración 1:Matriz del Diagrama de Contexto.

Beneficios para la Entidad Bancaria

- **Seguridad**: La autenticación vía OAuth2.0 es clave para proteger los datos del cliente y cumplir con regulaciones como GDPR o PCI-DSS.
- **Escalabilidad**: El uso de API Gateway permite agregar nuevos servicios sin afectar la arquitectura principal.
- **Experiencia del Usuario**: Las notificaciones (SMS y Push) deben ser oportunas y relevantes para mantener al cliente informado y comprometido.

- **Modularidad**: Separar el Core Bancario del Sistema Complementario permite una evolución independiente de cada componente.

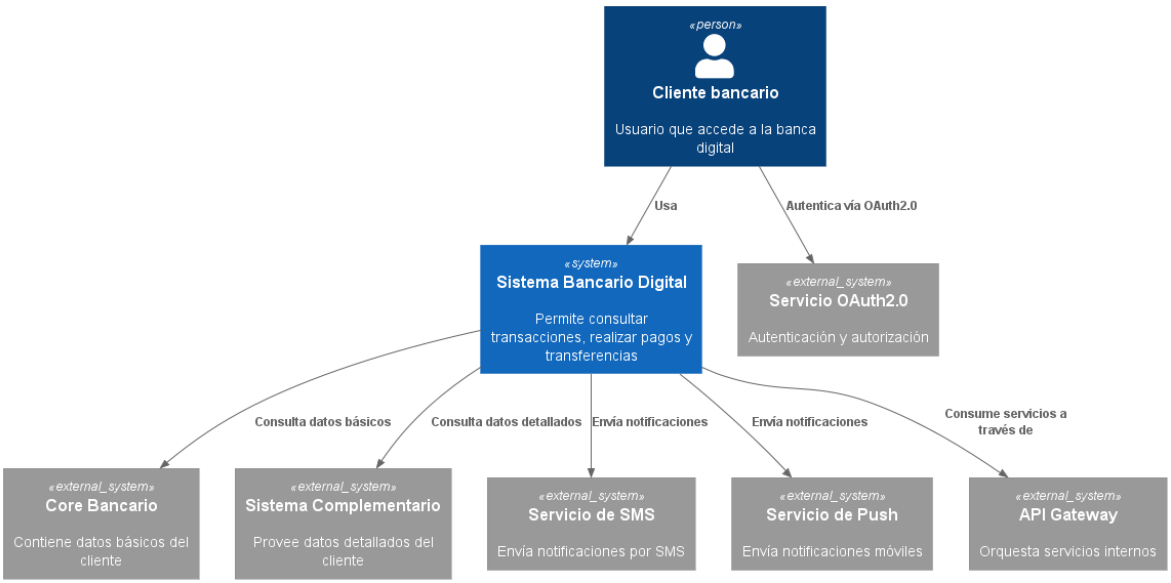


Ilustración 2.:Diagrama de Contexto BP

Este diagrama muestra cómo un cliente bancario interactúa con el Sistema Bancario Digital para realizar operaciones como pagos, transferencias y consultas. El cliente se autentica mediante un servicio seguro llamado OAuth2.0, que garantiza que solo usuarios autorizados accedan al sistema.

Una vez autenticado, el sistema se conecta con varios servicios:

- ✓ Core Bancario: Proporciona datos esenciales como saldos, movimientos y productos financieros.
- ✓ Sistema Complementario: Ofrece información adicional del cliente, como perfil, preferencias o historial.
- ✓ Servicio de SMS y Servicio de Push: Envían notificaciones al cliente sobre sus transacciones.
- ✓ API Gateway: Actúa como punto de entrada para consumir servicios internos de forma segura y organizada.

Justificación Teórica 1: Uso de OAuth2.0 para Autenticación

¿Por qué se eligió OAuth2?

- ✓ Estándar internacional: OAuth2.0 es ampliamente adoptado por sistemas bancarios, fintechs y plataformas digitales por su robustez y compatibilidad.
- ✓ Separación de responsabilidades: Permite delegar la autenticación a un servicio especializado, reduciendo riesgos en el sistema principal.

¿Qué opciones se evaluaron?

- ✓ Autenticación básica (usuario/contraseña): descartada por ser menos segura y no escalable.
- ✓ JWT sin OAuth: viable, pero menos flexible para delegar autenticación a terceros como Google o Apple.

Justificación Teórica 2: Uso de API Gateway como punto de entrada

¿Por qué se eligió un API Gateway?

- ✓ Centralización del acceso: Permite controlar, monitorear y asegurar todas las llamadas a servicios internos desde un único punto.
- ✓ Escalabilidad y mantenimiento: Facilita la gestión de múltiples microservicios sin exponer directamente sus endpoints.

  ¿Qué opciones se evaluaron?
- ✓ Acceso directo a microservicios: descartado por falta de control y seguridad.
- ✓ Bus de servicios (ESB): considerado, pero menos ágil para arquitecturas modernas basadas en microservicios.

2.2.Diagrama de Contenedores (Nivel 2)

La Matriz de Contenedores (Aplicaciones) permite visualizar la arquitectura del sistema como un conjunto de bloques funcionales independientes pero interconectados. Esta estructura facilita la escalabilidad, la seguridad, el mantenimiento y la evolución tecnológica del sistema. A continuación, se presenta la justificación por categoría, junto con las herramientas típicas que se emplean en cada una.

Categoría	Rol Funcional	Herramientas	Justificación Estratégica
Cliente / Frontend	Interfaces de usuario(UI) para acceso al sistema.	- React / Angular (SPA Web) - Flutter / React Native (App Móvil)	Interfaces modernas y responsivas que mejoran la experiencia del cliente y reducen fricción.
Backend / API	Procesa lógica de negocio y gestiona solicitudes.	- Node.js / Spring Boot - Express / NestJS - REST	Desacopla el frontend del backend; permite escalar servicios y aplicar lógica empresarial.
Seguridad / Autenticación	Verificación de identidad y registro biométrico	- OAuth2.0 / OpenID Connect. - Face Recognition SDKs - Auth0	Protege el acceso al sistema; cumple con normativas como PCI-DSS y GDPR.
Gestión de Datos	Almacena y optimiza acceso a datos de clientes y transacciones	- RDS PostgreSQL	Bases de datos seguras y rápidas; caché mejora el rendimiento en consultas frecuentes.
Auditoría / Registro	Registra todas las acciones del sistema,	- DynamoDM - MKS(Managed Streaming for Kafka)	Base de datos NoSQL orientada a documentos, ideal para datos flexibles y semiestructurados y MKS plataforma de mensajería distribuida para flujos de eventos en tiempo real.
Orquestación / Integración	Coordina y enruta servicios externos e internos	- AWS API Gateway	Centraliza el control de tráfico, seguridad y monitoreo de servicios distribuidos.

Ilustración 3:Matriz del Diagrama de Contenedores.

Valor Estratégico para la Entidad Bancaria

- **Alta disponibilidad y escalabilidad automática** sin necesidad de gestionar servidores.
- **Costos optimizados** por uso real (pay-per-use), ideal para cargas variables.
- **Seguridad multicapa** con control de acceso, cifrado y monitoreo continuo.
- **Interoperabilidad** con sistemas legacy y proveedores externos.
- **Cumplimiento normativo** y trazabilidad para auditorías y reguladores.

Beneficios Estratégicos del Enfoque por Contenedores:

- **Modularidad**: Cada contenedor puede evolucionar de forma independiente, facilitando actualizaciones sin afectar el sistema completo.
- **Escalabilidad Horizontal**: Permite distribuir la carga entre múltiples instancias de cada servicio.
- **Seguridad Granular**: Se pueden aplicar políticas específicas por contenedor (por ejemplo, autenticación reforzada en el backend).
- **Observabilidad**: Herramientas como Cloudwatch, Grafana permiten monitorear el sistema en tiempo real.
- **Automatización y DevOps**: Facilita la integración con pipelines CI/CD y despliegues en contenedores mediante ECS Fargate.

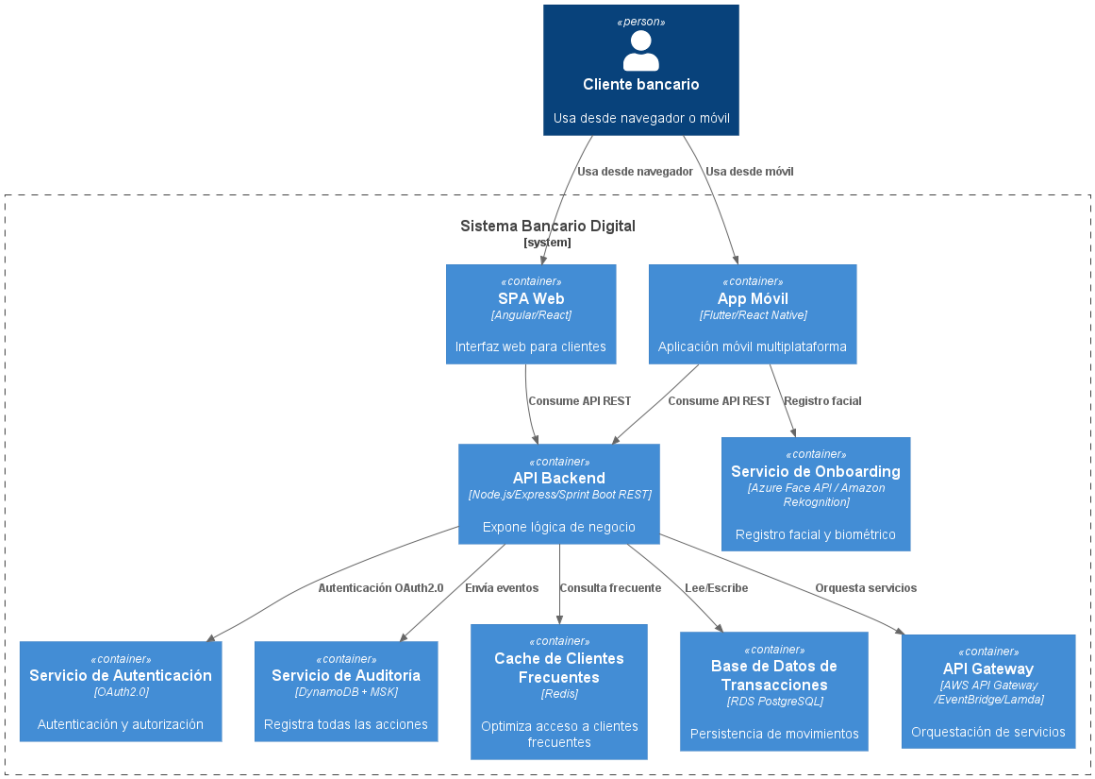


Ilustración 4:Diagrama de Contenedores BP

Este diagrama de contenedores representa la arquitectura de un Sistema Bancario Digital, estructurado en capas y contenedores que interactúan entre sí:

Cliente Bancario:

- ✓ Accede al sistema desde navegador web o dispositivo móvil.

Contenedores de presentación:

- ✓ **SPA Web:** Aplicación web de una sola página, rápida y dinámica.
- ✓ **App Móvil:** Aplicación multiplataforma para Android/iOS.

Ambos consumen servicios del backend a través de REST APIs.

Protocolo: HTTPS + OAuth2 para autenticación segura

Contenedor de lógica de negocio:

API Backend: Expone la lógica del negocio y orquesta las operaciones, que expone endpoints RESTful. Se comunica con:

- ✓ Servicio de Onboarding: Registro de nuevos clientes y captura biométrica.
- ✓ Servicio de Autenticación: Maneja login, tokens y permisos.
- ✓ Servicio de Auditoría: Registra todas las acciones para trazabilidad.
- ✓ Cache de Clientes Frecuentes: Mejora el rendimiento en consultas repetidas.
- ✓ Base de Datos de Transacciones: Persistencia de operaciones bancarias.
- ✓ **Tecnología sugerida:** Node.js / Express/Spring Boot .
- ✓ **Patrón:** Microservicios desacoplados, con control de acceso vía API Gateway

• API Gateway: Punto de entrada para servicios internos, controla acceso y seguridad.

Cloud: AWS como proveedor principal, con servicios gestionados para escalabilidad, alta disponibilidad y cumplimiento normativo.

2.3.Diagrama de Componentes (Nivel 3)

En este diagrama de Componentes del sistema bancario BP , tenemos la siguiente matriz que contempla la capa, elemento técnico, Rol Funcional y Notas estratégicas.

Capa	Elemento Técnico	Rol Funcional	Notas Estratégicas
Controlador (Presentación)	Controlador de Transacciones (REST Controller)	Gestiona solicitudes REST para pagos y transferencias.	Punto de entrada del cliente; debe ser seguro, rápido y cumplir estándares de interoperabilidad.
Capa de Servicios (Negocio)	Servicio de Movimientos	Consulta y entrega historial de transacciones.	Clave para transparencia y confianza del cliente; debe ser eficiente y auditable.
	Servicio de Transferencias	Ejecuta transferencias entre cuentas.	Debe garantizar integridad, validación antifraude y trazabilidad.
	Servicio de Pagos	Procesa pagos interbancarios y propios.	Alta disponibilidad y cumplimiento normativo (ej. compensación bancaria).
Capa de Persistencia (Datos)	Repositorio de Clientes (DAO)	Accede a datos del cliente desde la base de datos.	Debe proteger datos sensibles y cumplir con regulaciones como GDPR o PCI-DSS.
	Repositorio de Auditoría (DAO)	Registra eventos y acciones del sistema.	Fundamental para trazabilidad, cumplimiento legal y análisis de comportamiento.
Adaptadores / Integraciones	Servicio de Notificaciones (Adapter)	Envía notificaciones vía SMS y Push.	Mejora la experiencia del cliente; útil para alertas de seguridad y confirmaciones de operación.

Ilustración 5:Capas de los Componentes del Sistema Bancario.

Capa / Componente	Elemento Técnico	Herramientas Recomendadas
Controlador (Presentación)	Controlador de Transacciones (REST Controller)	- Spring Boot (Java) - Express.js (Node.js) - OpenAPI / Swagger
Capa de Servicios (Negocio)	Servicio de Movimientos	- Java / Node.js - Spring Service Layer - NestJS (TypeScript)
	Servicio de Transferencias	- Motor de reglas (Drools) - Validaciones antifraude con ML (TensorFlow, Scikit-learn)
	Servicio de Pagos	- ISO 20022 / SEPA / ACH APIs - Integración con sistemas de compensación bancaria
Capa de Persistencia (Datos)	Repositorio de Clientes (DAO)	- Hibernate / JPA (Java) - Sequelize (Node.js) - PostgreSQL
	Repositorio de Auditoría (DAO)	- AWS MKS - DynamoDB
Adaptadores / Integraciones	Servicio de Notificaciones (Adapter)	- AWS SNS (Amazon Simple Notification Service) es un servicio de mensajería totalmente gestionado que permite enviar notificaciones de forma rápida, escalable y desacoplada entre aplicaciones o hacia usuarios finales,

Ilustración 6:Herramientas del Diagrama de Componentes.

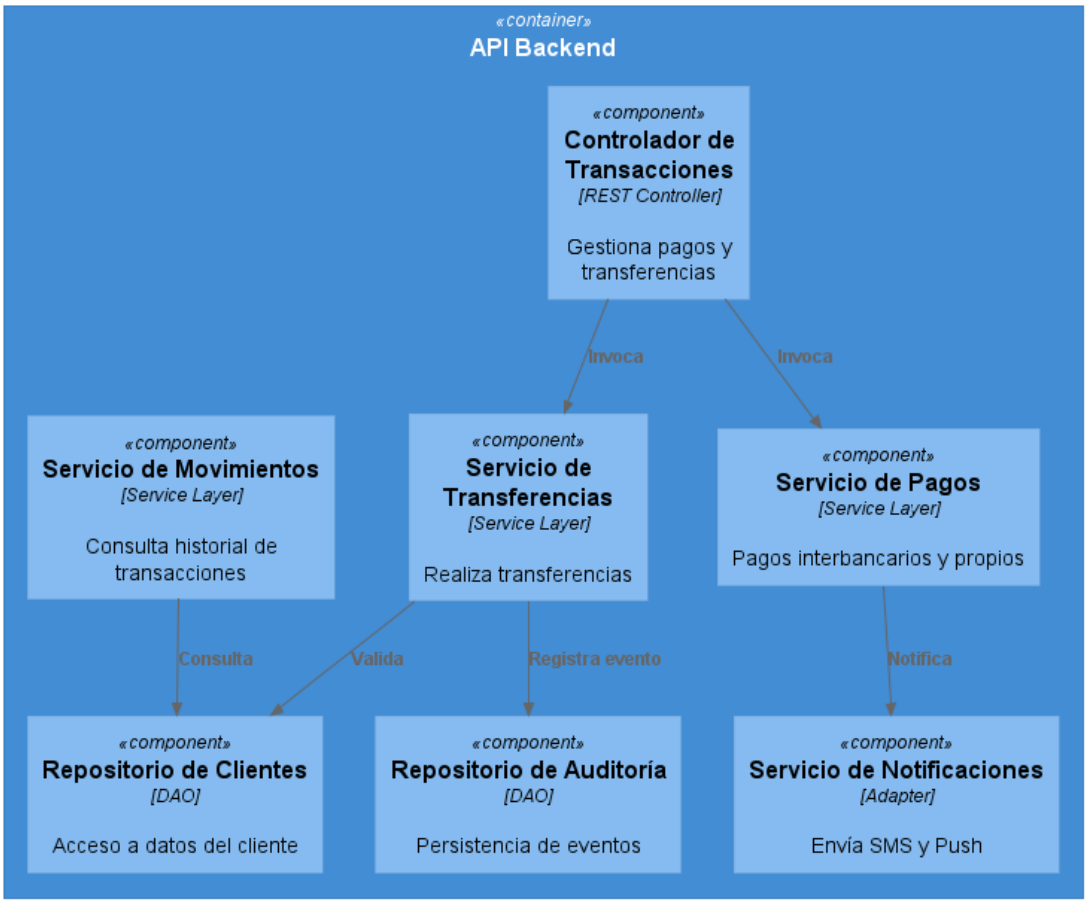


Ilustración 7:Diagrama de Componentes.

Contenedor Principal: API Backend

- **Responsabilidad:** Exponer endpoints RESTful para operaciones bancarias (transferencias, pagos, movimientos).
- **Tecnología sugerida:** Spring Boot / Node.js / .NET Core
- **Protocolo:** HTTPS + OAuth2 (seguridad en capa de transporte + autenticación federada)
- **Patrón arquitectónico:** Microservicio independiente, desacoplado del front-end y otros servicios

Controlador de Transacciones [REST Controller]

- **Función:** Orquesta las operaciones de pagos y transferencias.
- **Invoca:** Servicios de dominio según el tipo de operación.
- **Seguridad:** Validación de token OAuth2, control de acceso por roles.

Servicios de Dominio (Service Layer)

Servicio	Función técnica	Interacciones clave
Servicio de Movimientos	Consulta de historial de transacciones	Repositorio de Clientes
Servicio de Transferencias	Validación de cliente y ejecución de transferencias	Repositorio de Clientes + Auditoría
Servicio de Pagos	Pagos interbancarios + notificaciones	Servicio de Notificaciones

- ✓ **Patrón:** DDD (Domain-Driven Design), con separación clara entre lógica de negocio y persistencia
- ✓ **Protocolo interno:** Llamadas síncronas vía HTTP o asincronía con colas (ej. SQS/Kafka si se extiende)

Adaptadores y Repositorios (DAO / Adapter Layer)

Componente	Rol técnico	Tecnología sugerida
Repositorio de Clientes	Acceso a datos de clientes	RDS PostgreSQL / DynamoDB
Repositorio de Auditoría	Registro de eventos críticos	CloudWatch Logs
Servicio de Notificaciones	Envío de SMS y Push	AWS SNS / Pinpoint

- ✓ **Seguridad:** Logs cifrados, acceso controlado por IAM roles.
- ✓ **Cloud:** Uso de servicios gestionados en AWS para persistencia, mensajería y monitoreo.

Componentes Cloud Destacados

Componente	Servicio Cloud	Justificación
Persistencia	Amazon RDS / DynamoDB	Alta disponibilidad, backups automáticos
Auditoría	CloudWatch	Observabilidad, trazabilidad normativa
Notificaciones	SNS / Pinpoint	Escalabilidad en mensajería multicanal
Seguridad	AWS Cognito / IAM / Security Hub/ GuardDuty	Autenticación federada, gestión de claves

Justificación 1: Separación por capas + microservicios

- **Motivo:** Facilita mantenimiento, pruebas unitarias, escalabilidad por dominio
- **Evaluación:** Se comparó con monolitos y arquitecturas hexagonales. Se eligió microservicios con DDD para permitir evolución independiente de cada servicio (ej. pagos vs transferencias).
- **Beneficio:** Permite escalar servicios críticos sin afectar el resto del sistema. Mejora la trazabilidad y el control normativo.

Justificación 2: Uso de adaptadores para notificaciones y auditoría

- **Motivo:** Desacoplar lógica de negocio de servicios externos (SMS, logs, push)
- **Evaluación:** Se consideró integración directa, pero se optó por adaptadores para facilitar pruebas, mocking y cambios de proveedor.
- **Beneficio:** Flexibilidad para cambiar de Pinpoint, o de CloudWatch sin afectar el core del sistema.

3. Canales Digitales Frontend SPA y Móvil

- **SPA Web**: Se usará los framewoks Angular o React.

  Angular si se busca estructura rígida y control empresarial.

  React si se prioriza flexibilidad y velocidad de desarrollo.

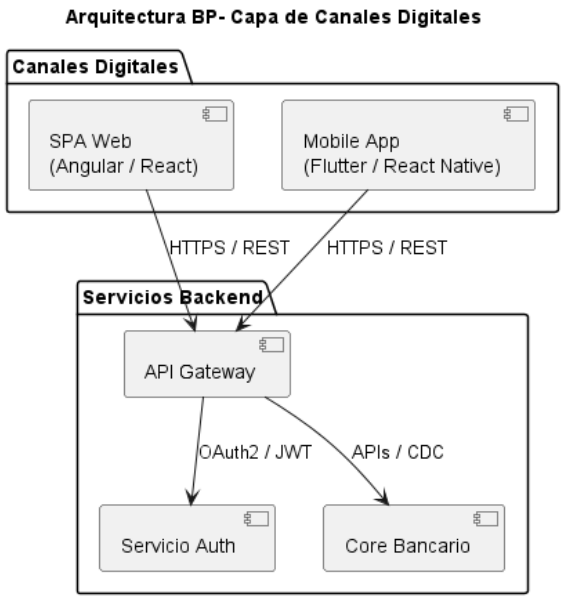
Criterio	Justificación
Ecosistema maduro	Ambos frameworks cuentan con amplio soporte empresarial, documentación robusta y comunidades activas. Angular es respaldado por Google y React por Meta, lo que garantiza continuidad y evolución.
Integración con OAuth2.0 y API REST	Angular y React permiten integrar fácilmente flujos de autenticación como Authorization Code Flow con PKCE, usando bibliotecas como angular-oauth2-oidc o react-oauth2. Además, su arquitectura basada en componentes facilita el consumo de APIs RESTful.
Manejo de estados y rutas	Angular ofrece un sistema de rutas integrado y gestión de estados con NgRx. React permite usar React Router y bibliotecas como Redux o Zustand para manejar estados complejos. Esto es clave para flujos como transferencias, pagos y consultas.
Escalabilidad y mantenibilidad	Angular promueve una arquitectura modular con servicios inyectables, ideal para proyectos bancarios. React, con hooks y componentes reutilizables, permite construir interfaces dinámicas y desacopladas.
Seguridad y cumplimiento	Ambos permiten implementar CSP, sanitización de inputs, y protección contra XSS/CSRF, alineándose con normativas como OWASP y PCI-DSS.

- App Móvil: Flutter o React Native.

Flutter si se prioriza rendimiento gráfico y control visual.

React Native si se busca integración rápida con SDKs existentes y ecosistema JS compartido con el SPA.

Criterio	Justificación
Multiplataforma (iOS/Android)	Ambos frameworks permiten desarrollar una sola base de código para múltiples plataformas, reduciendo costos y tiempos de entrega.
Integración con biometría y SDKs nativos	Flutter y React Native permiten integrar biometría (huella, rostro) mediante plugins como local_auth (Flutter) o react-native-fingerprint-scanner. También permiten integrar SDKs bancarios, notificaciones push y cámaras para onboarding facial.
Rendimiento y experiencia de usuario	Flutter compila a código nativo, ofreciendo alto rendimiento gráfico. React Native usa el puente nativo, con buen rendimiento y acceso a componentes nativos. Ambos permiten animaciones fluidas y UX moderna.
Comunidad y soporte	Flutter tiene fuerte respaldo de Google y una comunidad creciente. React Native tiene una comunidad consolidada y gran cantidad de librerías disponibles.
Seguridad y cumplimiento	Permiten cifrado local, almacenamiento seguro (Secure Storage / Keychain), y autenticación biométrica, alineándose con estándares como ISO/IEC 27001 y normativas locales.



Integración con Backend y Seguridad

- Ambas soluciones consumen servicios vía API Gateway con tokens OAuth2.0.
- Se recomienda usar **Authorization Code Flow con PKCE** para evitar exposición de credenciales en clientes públicos.
- Se puede implementar **SSO** si la entidad bancaria lo requiere, usando Azure AD B2C o Auth0.

Valor estratégico

- Reducción de costos operativos por reutilización de código.
- Experiencia de usuario consistente en todos los canales.
- Flexibilidad para incorporar nuevas funcionalidades como onboarding facial, notificaciones, o pagos QR.
- Cumplimiento normativo y alineación con estándares bancarios.

4. Autenticación y Onboarding

Flujo de Autenticación: OAuth2.0 con Authorization Code Flow + PKCE

Justificación:

- **Authorization Code Flow con PKCE** es el flujo recomendado para clientes públicos como SPA y apps móviles, ya que:

✓ No requiere almacenamiento de secretos en el cliente.
✓ Protege contra ataques de interceptación de tokens.
✓ Compatible con navegadores y dispositivos móviles.

Seguridad y compatibilidad

- **PKCE (Proof Key for Code Exchange**) agrega una capa de seguridad adicional al flujo estándar.

Compatible con proveedores como:

✓ IdentityServer (auto-hosted, extensible).
✓ Auth0 (SaaS, rápido de implementar).
✓ Azure AD B2C (integración con ecosistemas Microsoft, escalabilidad empresarial).

Flujo resumido:

1. Cliente inicia login → redirige a proveedor OAuth2.0.
2. Usuario se autentica → se genera código de autorización.
3. Cliente envía código + PKCE verifier → recibe token de acceso.
4. Token se usa para consumir APIs protegidas.

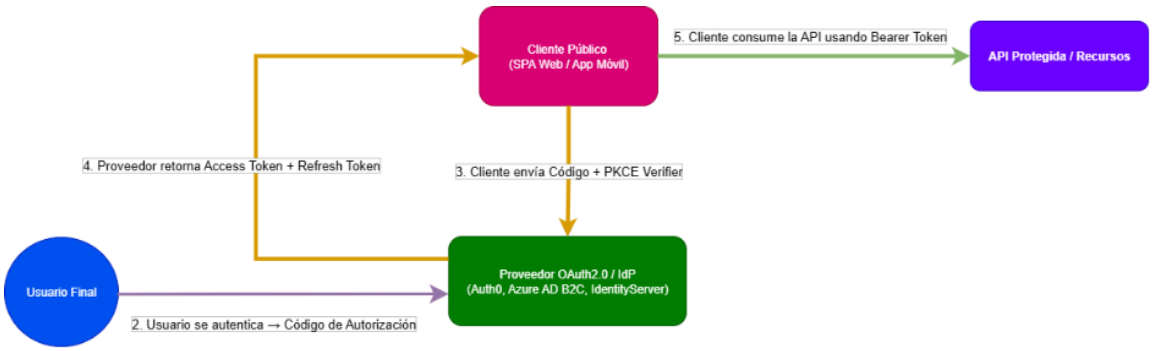


Ilustración 8:Flujo de Autenticación.

Onboarding Biométrico con Reconocimiento Facial

Mejora la experiencia de usuario y reduce fricción en el registro.

- ✓ Fortalece la seguridad desde el primer contacto.
- ✓ Cumple con estándares de autenticación biométrica (FIDO2, ISO/IEC 30107).

Herramienta	Características
Azure Face API	Detección facial, verificación, reconocimiento. Integración con Azure AD B2C.
Amazon Rekognition	Análisis facial, comparación de rostros, integración con Cognito y Lambda.

Flujo de Onboarding

1. Usuario instala app móvil y accede al flujo de registro.
2. Captura facial → se valida contra base de datos o patrón biométrico.
3. Si es exitoso → se generan credenciales (usuario/clave).
4. Se habilita autenticación futura por:
   - Usuario + clave
   - Huella digital (Android Biometric API)
   - Reconocimiento facial (iOS FaceID / Android Face Unlock).

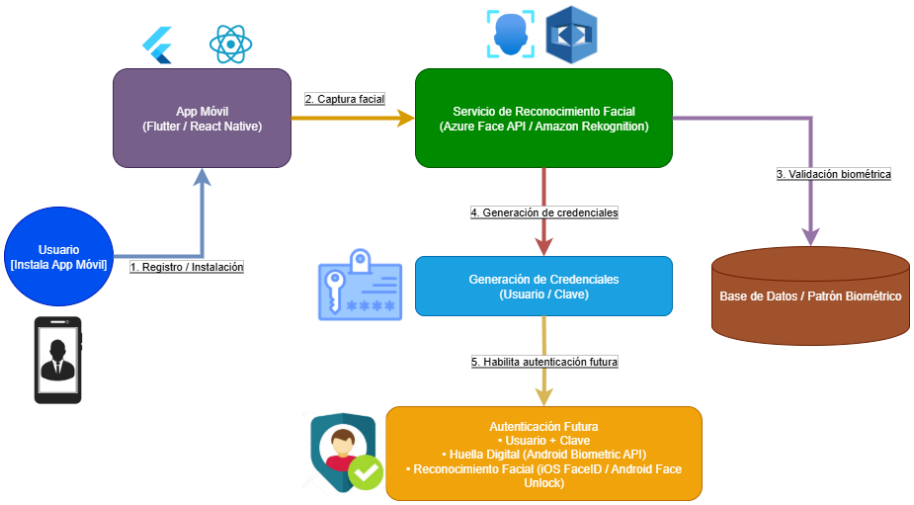


Ilustración 9:Flujo de Reconocimiento Facial BP

Justificación del flujo de reconocimiento Facial:

Cuando de se realiza la instalación y el registro del usuario , este permite que se descargue la app y se registre, sentando las bases para una experiencia personalizada y segura. Es el punto de entrada al ecosistema de autenticación biométrica.

Se utilizar frameworks multiplataforma como Flutter o React Native permite desarrollar una única base de código para iOS y Android, reduciendo costos y tiempo de desarrollo. La captura facial es el primer paso para vincular la identidad biométrica del usuario con su cuenta.

La validación Biométrica con Azure Face API / Amazon Rekognition, ofrecen algoritmos avanzados y confiables para verificar la identidad del usuario. Al delegar esta tarea a servicios cloud, se garantiza escalabilidad, precisión y cumplimiento con estándares de seguridad internacionales.

Continuando con la generación de Credenciales (Usuario / Clave),aunque el sistema es biométrico, generar credenciales tradicionales permite compatibilidad con otros sistemas y ofrece una capa adicional de autenticación en caso de fallos biométricos o accesos desde dispositivos no compatibles.

En cuestión del almacenamiento en Base de Datos / Patrón Biométrico, permite guardar el patrón biométrico (de forma segura y cifrada) permite futuras comparaciones para autenticación. Este paso es crítico para mantener la persistencia de la identidad del usuario.

En la habilitación de Autenticación Futura, una vez registrado, el usuario puede autenticarse en el futuro sin repetir todo el proceso. Esto mejora la experiencia de usuario y reduce fricción en el acceso.

Podemos presentar algunas Opciones de Autenticación Futura:

Usuario + Clave: Método tradicional como respaldo.

Huella Digital (Android Biometric API): Aprovecha sensores nativos para autenticación rápida y segura.

Reconocimiento Facial (iOS FaceID / Android Face Unlock): Utiliza tecnologías nativas del sistema operativo para una experiencia fluida y segura.

Beneficios Generales del Flujo:

Seguridad: Combina biometría con credenciales tradicionales.

- ✓ Escalabilidad: Uso de servicios cloud como Azure y AWS.
- ✓ Compatibilidad: Funciona en múltiples plataformas y dispositivos.
- ✓ Experiencia de Usuario: Flujo intuitivo y rápido para el usuario final.

Integración con arquitectura

- El servicio de onboarding se desacopla como microservicio independiente.
- Se comunica con el servicio de autenticación OAuth2.0 para registrar nuevos usuarios.
- Se integra con el servicio de auditoría para registrar eventos críticos (registro, autenticación, fallos).
- Se conecta con el servicio de notificaciones para confirmar al usuario su registro exitoso.

5. Patrones de diseño recomendados

A. CQRS + Event Sourcing : Permite la separación de lectura/escritura y trazabilidad.

Command Query Responsibility Segregation separa las operaciones de lectura (queries) de las de escritura (commands).

• En lugar de tener un único modelo para todo, se usan dos modelos especializados:

- ✓ Command Model: para modificar el estado.
- ✓ Query Model: para consultar el estado.

¿Qué es Event Sourcing?

• En lugar de guardar el estado actual, se guarda una secuencia de eventos que llevaron a ese estado.

• Cada acción (ej. transferencia realizada) se registra como un evento inmutable.

Aplicación en el sistema bancario BP;

- ✓ **Transferencias**: cada vez que un usuario realiza una transferencia, se genera un evento .
- ✓ **Historial de transacciones:** se reconstruye desde los eventos, garantizando trazabilidad completa.
- ✓ **Auditoría:** los eventos son perfectos para cumplir con normativas como ISO 27001 o PCI-DSS.

Beneficios:

- ✓ Trazabilidad total.
- ✓ Escalabilidad: puedes optimizar las consultas sin afectar la lógica de negocio.
- ✓ Integración con MSK para persistencia de eventos.

B. **Saga Pattern**: Este patrón permite manejar transacciones distribuidas en microservicios , como por ejemplo para orquestación de transferencias interbancarias.

- Cada paso de la transacción es un servicio independiente que emite eventos y escucha respuestas.
- Si algo falla, se ejecutan acciones compensatorias.

Ejemplo:

Aplicación en transferencias interbancarias

1. Servicio A inicia transferencia → evento "TransferenciaIniciada"
2. Servicio B debita cuenta origen → evento "CuentaDebitada"
3. Servicio C acredita cuenta destino → evento "CuentaAcreditada"
4. Si falla el paso 3 → se emite "ReversarDebito"

Beneficios

- Evita bloqueos de recursos.
- Alta disponibilidad y resiliencia.
- Compatible con arquitecturas basadas en eventos (Kafka, RabbitMQ).

C. **Adapter Pattern:** Este patrón permite desacoplar servicios de notificación y que el sistema interactúe con múltiples servicios externos sin depender directamente de ellos, se crea una interfaz común y cada servicio externo tiene su propio adaptador.

Aplicación en notificaciones

```
interface Notificador {
  enviar(mensaje, destino)
}

class NotificadorSMS implements Notificador {
  enviar(mensaje, destino) // usa Pinpoint
}

class NotificadorPush implements Notificador {
  enviar(mensaje, destino) //usa Firebase
```

```
}
```

Beneficios

- Puedes cambiar de proveedor sin modificar la lógica de negocio.
- Facilita pruebas unitarias y simulaciones.
- Mejora la mantenibilidad y extensibilidad.

D. **Factory Pattern** : Este patrón es para creación de clientes en onboarding y encapsula la lógica de creación de objetos en una clase especializada.

- Ideal cuando el proceso de creación depende de múltiples condiciones o configuraciones.

Aplicación en onboarding de clientes

```
class ClienteFactory {
 crearCliente(tipo, datos) {
   if (tipo == "facial") //crea cliente con biometría
   if (tipo == "manual") // crea cliente con usuario/clave
 }
}
```

Beneficios:

- Centraliza la lógica de creación.
- Permite validar y enriquecer datos antes de instanciar el objeto.
- Facilita la incorporación de nuevos tipos de clientes (ej. corporativos, VIP).

Integración en la Arquitectura bancaria de BP

Patrón	Ubicación en C4	Propósito
CQRS + Event Sourcing	Backend API + Auditoría	Separación de responsabilidades y trazabilidad.
Saga Pattern	Servicio de Transferencias	Orquestación distribuida.
Adapter Pattern	Servicio de Notificaciones	Desacoplamiento de proveedores.
Factory Pattern	Servicio de Onboarding	Creación flexible de clientes.

De los 4 patrones mencionados, anteriormente se prefiere usar el **Saga Pattern** este patrón está diseñado para coordinar múltiples microservicios o componentes que deben ejecutar una serie de pasos para completar una transacción distribuida. Cada paso puede ser ejecutado por un componente distinto, y el patrón se encarga de orquestar o coreografiar esas interacciones.

Características clave:

- ✓ Desacopla componentes, pero los mantiene sincronizados mediante eventos o comandos.
- ✓ Gestiona compensaciones en caso de fallos, lo que permite mantener la consistencia sin bloqueos.
- ✓ Ideal para flujos de negocio complejos, donde varios servicios deben colaborar

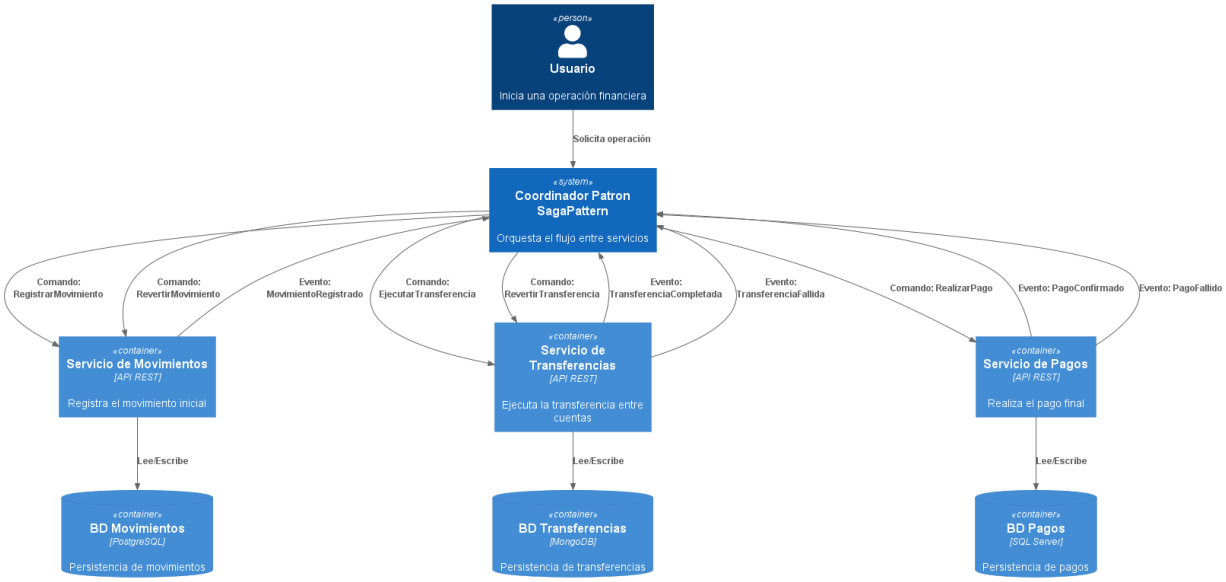


Ilustración 10: Diagrama se Patrón Saga Pattern de Servicios.

En este Diagrama del patrón de diseño **Saga Pattern** permite manejar transacciones distribuidas en arquitecturas de microservicios. En lugar de usar una transacción global (como en bases de datos monolíticas), se divide en una serie de pasos locales coordinados por un orquestador central. Cada paso tiene su propia lógica de compensación en caso de fallo.

Componentes del Diagrama

Rol	Servicio	Función	Comando	Evento
Usuario	—	Inicia la operación financiera	—	—
Coordinador Saga	Orquestador	Controla el flujo de la transacción	Envía comandos	Recibe eventos
Servicio de Movimientos	Microservicio	Registra el movimiento inicial	RegistrarMovimiento	MovimientoInicialRegistrado
Servicio de Transferencias	Microservicio	Ejecuta la transferencia	EjecutarTransferencia	TransferenciaEjecutada
Servicio de Pagos	Microservicio	Finaliza el pago	RealizarPagoFinal	PagoConfirmado

Flujo Orquestado Paso a Paso:

1. **Inicio**: El usuario solicita una operación (ej. transferencia).
2. **Orquestador (Saga)**:
   - Envía `RegistrarMovimiento` al Servicio de Movimientos.
   - Espera el evento `MovimientoInicialRegistrado`.
   - Luego envía `EjecutarTransferencia` al Servicio de Transferencias.
   - Espera `TransferenciaEjecutada`.
   - Finalmente envía `RealizarPagoFinal` al Servicio de Pagos.
   - Espera `PagoConfirmado`.
3. **Finalización**: Si todos los eventos son exitosos, la operación se considera completada.

¿Qué pasa si algo falla?

El orquestador puede activar comandos de compensación (no mostrados en el diagrama) como:

- Revertir el movimiento inicial.
- Cancelar la transferencia.
- Anular el pago.

6. Infraestructura en la nube (AWS)

Se tiene la siguiente matriz donde se detalla la justificación técnica y estratégica:

Requisito	Solución	Justificación
Alta disponibilidad (HA)	AWS ECS Fargate con múltiples zonas	Permiten desplegar contenedores o aplicaciones web en múltiples zonas de disponibilidad. Esto garantiza que si una zona falla, el tráfico se redirige automáticamente a otra. Ideal para microservicios desacoplados.
Tolerancia a fallos	AWS Lamda Auto Scaling + Load Balancer	Lamda es Auto Scaling ajusta la cantidad de instancias según la carga. El Load Balancer distribuye el tráfico entre instancias saludables. Esto evita caídas por sobrecarga y asegura continuidad operativa.
Recuperación ante desastres (DR)	Backups en S3 + replicación geográfica	S3 permiten backups automáticos y replicación entre regiones. Esto protege contra pérdida de datos por fallos físicos o ciberataques. Se puede configurar recuperación punto en el tiempo.
Seguridad	IAM Roles, AWS Secrets Manager, WAF, TLS, AWS Shield Advanced	IAM controla acceso granular a recursos. Secrets Manager protegen credenciales y claves. WAF (Web Application Firewall) bloquea ataques como XSS y SQLi. TLS cifra la comunicación. AWS Shield Advanced es un servicio administrado de protección contra ataques DDoS (Denegación de Servicio Distribuido) Todo esto cumple con PCI-DSS, ISO 27001 y normativas locales.
Monitoreo	CloudWatch + Grafana	CloudWatch recolectan métricas, logs y alertas. Grafana permite visualización avanzada y dashboards personalizados. Esto facilita la observabilidad, análisis de rendimiento y respuesta proactiva.
Auto-healing	Health checks + reinicio automático	Las instancias se monitorean constantemente. Si una falla, se reinicia automáticamente o se reemplaza. Esto reduce el tiempo de inactividad y mejora la resiliencia del sistema.

Arquitectura recomendada en la nube de AWS

- **Frontend SPA**: Deploy en S3 + CloudFront (CDN).

✓ **S3 + CloudFront**: Hosting estático + CDN para baja latencia global.
✓ **WAF + Shield**: Protección contra ataques DDoS y amenazas web.
✓ **Route 53**: DNS resiliente y balanceo geográfico si se requiere.

- **Seguridad**: IAM, Secrets Manager, WAF, Shield, GuardDuty.

✓ **IAM**: Control de acceso granular.
✓ **Secrets Manager**: Gestión segura de credenciales.

- ✓ **GuardDuty**: Detección de amenazas.
- ✓ **AWS Config + Security Hub**: Cumplimiento normativo continuo.

- **App móvil**: Backend en ECS Fargate + API Gateway.
- **ECS Fargate:** Ejecuta la lógica de negocio de la aplicación móvil , en este caso validaciones, procesamiento de pagos, reglas de negocio, también, expone endpoints REST que la app móvil consume y se comunica con servicios como bases de datos, sistemas de auditoría, APIs internas.

  Permite escalar automáticamente según la carga (más usuarios, más contenedores, y aísla cada microservicio como las transferencias, pagos, e historial.

- **API Gateway + Lamda (Serverless):** este va a contemplar los diferentes API que se mencionan a continuación desacoplados cada uno con Lamda .

  - ✓ Servicio API de Clientes.
  - ✓ Servicio API de Movimientos.
  - ✓ Servicio API de Transferencias.
  - ✓ Servicio API de Transferencias.
  - ✓ Servicio API de Pagos.

Integración con **Amazon SNS** y **Amazon Pinpoint** para SMS, email, push.

- **Base de datos**: RDS PostgreSQL con replicación multi-AZ.

  - ✓ **RDS PostgreSQL Multi-AZ**: Transaccional, alta disponibilidad.
  - ✓ **Amazon DynamoDB (MongoDB compatible)**: Auditoría flexible.

- **Auditoría**: DynamoDB + eventos en Kafka (MSK).
- **Monitoreo**: CloudWatch + Grafana + SNS para alertas.

Valor estratégico:

- **Escalabilidad automática**: Se adapta a la demanda sin intervención manual.
- **Resiliencia operativa**: Minimiza el impacto de fallos y ataques.
- **Cumplimiento normativo**: Compatible con estándares bancarios y regulatorios.
- **Costos controlados**: Pago por uso, con posibilidad de optimización por horarios o tráfico.
- **Desacoplamiento total**: Cada componente puede evolucionar sin afectar al resto.

La arquitectura está diseñada bajo principios de modularidad, seguridad, escalabilidad y trazabilidad, alineada con estándares internacionales y buenas prácticas de AWS. Cada VPC representa un dominio funcional independiente, lo que permite aislamiento lógico, control granular de accesos y facilidad de mantenimiento.

Relación entre Componentes de la Arquitectura de nube AWS

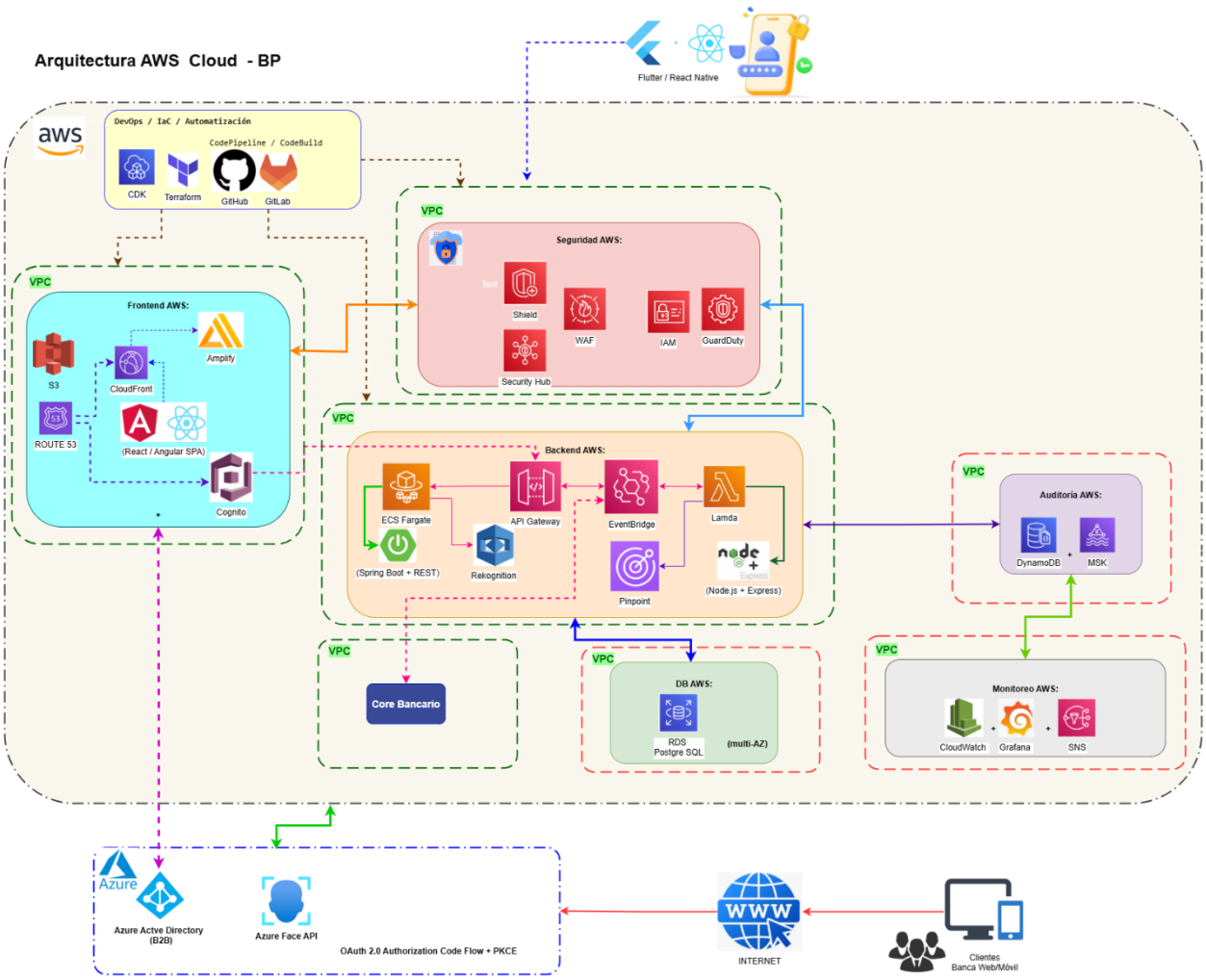


Ilustración 11:Arquitectura BP en AWS.

VPC / Dominio	Servicios Clave	Propósito Funcional	Buenas Prácticas
Frontend AWS	Route 53, CloudFront, WAF, Cognito, S3, Amplify, API Gateway	Acceso público, entrega de contenido, autenticación y hosting web	- Usar WAF para protección contra OWASP Top 10 - Cognito para federación segura - Amplify para CI/CD frontend
Seguridad AWS	GuardDuty, Security Hub, IAM, Config	Monitoreo, gestión de identidades y cumplimiento normativo	- Activar GuardDuty y Security Hub para detección proactiva - Config + CloudTrail para auditoría continua
Backend AWS	EC2, RDS, ElastiCache, S3, AppConfig,	Lógica de negocio, persistencia de datos y configuración centralizada	- Usar AppConfig para despliegues seguros - Parameter Store cifrado para secretos - RDS con backups automáticos

Serverless AWS	Lambda, EventBridge, SQS, SNS, DynamoDB	Procesamiento sin servidor, eventos y mensajería desacoplada	- Lambda con IAM mínimo necesario - SQS/SNS para resiliencia
Data & Control	CloudWatch, CloudFormation, CodePipeline, CodeBuild, CodeDeploy	Observabilidad, trazabilidad y automatización CI/CD	- CloudWatch con alarmas por VPC- CodePipeline con validaciones
Audit AWS	CloudTrail, Config,	Auditoría, cumplimiento y protección de datos sensibles	- Macie para detección de PII - Config para reglas normativas - CloudTrail centralizado en S3 cifrado
Integraciones Externas	Azure AD, Azure DevOps, GitHub, Internet	Identidad federada, control de versiones y conectividad externa	- Azure AD para SSO empresarial - GitHub con escaneo SAST/SCA - Control de egress con NAT/WAF
One Branch	Conexión entre VPCs y servicios	Punto de control o acceso unificado entre dominios	- Usar Transit Gateway o VPC Peering - Control de tráfico con Security Groups y NACLs

Buenas Prácticas Aplicadas

- **Segmentación por VPC**: Aísla funciones críticas (auth, base de datos, backend) para minimizar riesgos y facilitar auditorías.
- **Principio de menor privilegio (IAM)**: Cada servicio accede solo a lo que necesita, reduciendo superficie de ataque.
- **Desacoplamiento con SQS/SNS**: Mejora resiliencia y escalabilidad. Si un componente falla, los mensajes no se pierden.
- **Serverless (Lambda)**: Reduce costos operativos y permite escalar automáticamente según demanda.
- **Federación de identidad (Azure AD + Cognito)**: Mejora experiencia de usuario y seguridad con autenticación centralizada.

- **Monitoreo proactivo (CloudWatch + X-Ray)**: Permite detectar anomalías antes de que impacten al cliente.
- **Uso de servicios gestionados (RDS, DynamoDB, Amplify)**: Minimiza mantenimiento y mejora disponibilidad.
- **PKCE en OAuth2**: Previene ataques de interceptación en flujos móviles o públicos.

La solución está diseñada para garantizar alta disponibilidad, seguridad y trazabilidad en un entorno bancario. Cada componente cumple una función específica dentro de un ecosistema desacoplado y monitoreado, donde la autenticación, la lógica de negocio y la persistencia de datos están aisladas pero interconectadas estratégicamente. Se aplican principios de arquitectura moderna como **serverless**, federación de identidad y segmentación por dominios funcionales, lo que permite escalar, auditar y evolucionar la solución sin comprometer la seguridad ni la experiencia del usuario.

7. Equipo del proyecto propuesto.

Nuestro equipo está compuesto por profesionales altamente calificados y con experiencia probada en la modernización de plataformas empresariales. Los perfiles clave y su dedicación al proyecto son:

1 Arquitecto de Soluciones / Líder Técnico: responsable del diseño de la arquitectura y la guía técnica del equipo.

2 Desarrolladores Senior de Microservicios: Expertos en Sprint Boot, Flutter ,Java Reactive, APIs REST y buenas prácticas de codificación.

1 especialista en DevOps: Responsable de la implementación del pipeline de CI/CD y la infraestructura de AWS ECS Fargate.

1 Analista de Calidad (QA): Encargado de la estrategia de pruebas, la automatización y el reporte de calidad.

1 Product Owner (opcional, dependiendo del alcance): A cargo de la gestión del cronograma, los recursos y la comunicación con el Banco.

8. Cronograma de entrega (Estimación aproximada 4 meses).

Dado el alcance funcional, normativo y técnico del sistema bancario propuesto, se plantea un cronograma extendido que permita una ejecución rigurosa, con entregables progresivos y validaciones continuas. Esta planificación considera la complejidad de los servicios legacy, la integración con sistemas externos, la implementación de seguridad avanzada y la excelencia operativa exigida.

La siguiente es una línea de tiempo estimada:

Fase 1 (Semana 1–2):

Análisis y Discovery detallado de los servicios legacy, inventario de componentes, definición de dependencias críticas y evaluación normativa.

- Entregable: Documento de descubrimiento técnico y normativo.

Fase 2 (Semana 3–4):

Diseño de arquitectura de microservicios, definición de contenedores y componentes (modelo C4), estrategia de desacoplamiento e integración.

- Entregable: Diagramas C4, estrategia de implementación y justificación técnica.

Fase 3 (Semana 5–10):

Desarrollo de microservicios principales (datos básicos, movimientos, transferencias), pruebas unitarias, integración continua, validación de contratos.

- Entregable: Servicios funcionales, pruebas automatizadas, documentación técnica.

Fase 4 (Semana 11–12):

Implementación de pipelines CI/CD, configuración de monitoreo (CloudWatch, Grafana), alertas, auto-healing, despliegue en entornos de QA.

- **Entregable**: Infraestructura operativa, monitoreo activo, entornos listos para validación.

Fase 5 (Semana 13–14):

Integración de autenticación Oauth2.0, onboarding biométrico, MFA, pruebas de seguridad (SAST/DAST), validación normativa.

- **Entregable**: Módulos de autenticación seguros, cumplimiento normativo validado.

Fase 6 (Semana 15–16):

Despliegue final en producción, validación funcional, DR, HA, monitoreo activo, documentación ejecutiva.

- Entregable: Sistema en producción, informe de validación, presentación ejecutiva.

Nota: Este cronograma es una estimación sujeta a ajustes según el número de servicios a realizarse, la complejidad de los sistemas legacy y los hallazgos en la fase de Discovery. Se recomienda mantener entregas parciales cada 2 semanas para facilitar la retroalimentación continua y la validación temprana de componentes críticos.

Plan de Trabajo y Cronograma (Gantt)

Fases	Actividad	Responsable	Duración (Semanas)	Semana 1	Semana 2	Semana 3	Semana 4	Semana 5	Semana 6	Semana 7	Semana 8	Semana 9	Semana 10	Semana 11	Semana 12	Semana 13	Semana 14	Semana 15	Semana 16
1	Análisis y Discovery	Arq. / Dev.	3	X	X	X													
2	Diseño y Documentos	Arq. / Dev.	3				X	X	X										
3	Desarrollo	Desarrollador	6							X	X	X	X	X	X				
4	Pruebas y Calidad	QA / Dev.	4								X	X	X	X					
5	Despliegue en Producción	DevOps	3													X	X	X	X

Ilustración 12:Plan de Trabajo y Cronograma.

Para mayor detalle se encuentra adjuntado el archivo **Plan de Trabajo y Cronograma_ Banco BP.xlsx** , en los Anexos.

9. Proyección Financiera

Vamos a detallar los indicadores de la proyección financiera de la Propuesta económica a continuación:

Número de Microservicios a Entregar: Tomamos como referencia 15 microservicios, en el cual nos basamos en el alcance según definido en el proyecto. Es la cantidad total de entregables.

Costo Total Proyectado: Es el costo estimado de desarrollar los microservicios. Incluye mano de obra, infraestructura, licencias, etc.

Tenemos un costo Total proyectado es que el costo interno unitario **≈ 8,136.25 por microservicio**.

Precio Unitario (por Microservicio): Es el precio de venta por cada microservicio. Se calcula para lograr una ganancia neta del 40%.

Precio Unitario (con margen 40%): 13,560.42

- La fórmula es: $\boldsymbol{Precio\ Unitario = \frac{Costo\ Unitario}{1-Margen}}$

$$\boldsymbol{Precio\ Unitario = \frac{8{,}136.25}{1-0{,}40} = 13{,}560.42}$$

Nota: "En caso de requerir un mayor o menor número de microservicios, el precio total se ajustará multiplicando la cantidad de unidades por el valor unitario de **USD 13,560.42**."

Revenue Total Proyectado: Es el ingreso total esperado: Precio Unitario × Número de Microservicios.

Margen de Ganancia (%): Se calcula como:

$$Margen = \frac{Revenue\ Total\ Proyectado - Costo\ Total}{Revenue\ Total} \approx 40\%$$

Ganancia Neta Proyectada: Es la utilidad esperada después de cubrir todos los costos.

Plazo Máximo de Entrega: 15 días hábiles. Es el tiempo límite para entregar los 10 microservicios.

Período de Gracia: 2 días hábiles, margen adicional para imprevistos antes de incurrir en penalizaciones.

Resumen Ejecutivo y Proyección Financiera

Indicador	Valor	Fórmula/Comentarios
Número de Microservicios a Entregar	1	Basado en el análisis inicial del alcance.
Costo Total Proyectado	**$8.136,25**	Se obtiene el Costo Unitario Total * Número de Microservicios
Precio Unitario (por Microservicio)	**$ 13.560,42**	Cálculo basado en el Costo Unitario Total / (1- 0,40) que es el margen del 40%.
Revenue Total Proyectado	$13.560,42	Precio Unitario * Número de Microservicios , Este es el ingreso total esperado.
Margen de Ganancia (%)	40%	Debe ser >= 40%. Calcula (Revenue Total - Costo Total) neta dividida por el revenue total
Ganancia Neta Proyectada	**$5.424,17**	Ganancia final del proyecto.
Plazo Máximo de Entrega	15 días hábiles	
Período de Gracia	2 días hábiles	

Ilustración 13:Proyección Financiera.

Para mayor detalle se encuentra adjuntado el archivo **Proyección y Estimación de Costos RFP_ Banco BP.xlsx** , en los Anexos.

Estimación de Costos por Unidad de Servicio

Perfil	Rol	Tarifa/Hora	Horas Estimadas por Servicio	Costo por Servicio
Arquitecto de Soluciones	Diseño y Estrategia	$75	20	$1.500,00
Desarrollador Senior	Desarrollo y Pruebas	$60	60	$3.600,00
Especialista DevOps	CI/CD y despliegue	$65	15	$975,00
Analista QA	Pruebas y Calidad	$50	20	$1.000,00
Subtotal Costo de Personal				**$7.075,00**
Costo de Herramientas	GitHub Copilot			$100
Costos Indirectos (15% sobre personal)	Administración y Gestión			$1.061,25
Costo Unitario Total				**$8.136,25**

Tabla 1: Estimación de Costo por Unidad de Servicio.

Para mayor detalle se encuentra adjuntado el archivo **Proyección y Estimación de Costos RFP_ Banco BP.xlsx** , en los Anexos.

En la *tabla2* muestra cómo se distribuye el esfuerzo del equipo en términos de horas y porcentaje de dedicación.

Descripción de las columnas:

- Perfil / Rol en el Proyecto
- Tarifa/Hora
- Unidades de microservicios.

- Horas Totales Estimadas: Por ejemplo, 10 unidades × 20 horas = 200 horas.
- % Dedicación: Es la proporción del total de horas que representa cada perfil

Equipo Propuesto y Dedicación

- *Tabla 2:Equipo Propuesto y Dedicación.*

Perfil	Rol en el Proyecto	Tarifa/Hora	Horas Estimadas	Unidades de Microservicios	Horas Totales Estimadas	% de Dedicación
Arquitecto de Soluciones	Liderazgo y Diseño	$75	20	1	20	17,39%
Desarrollador Senior	Programación y Refactorización	$60	60	1	60	52,17%
Especialista DevOps	CI/CD y Automatización	$65	15	1	15	13,04%
Analista QA	Calidad y Pruebas	$50	20	10	20	17,39%

- Para mayor detalle se encuentra adjuntado el archivo **Equipo Propuesto y Dedicación_BancoBP.xlsx** , en los Anexos.

10.Regulaciones bancarias y estándares

Normativas Aplicadas

En la arquitectura propuesta de solución de banco BP , se considera el cumplimiento de las siguientes normativas y estándares relevantes para el sector financiero:

- Ley Orgánica de Protección de Datos Personales (LOPDP – Ecuador)
- Reglamento General de Protección de Datos (GDPR – si aplica por jurisdicción de usuarios)
- PCI DSS (Payment Card Industry Data Security Standard)
- ISO/IEC 27001: Gestión de Seguridad de la Información
- OWASP ASVS (Application Security Verification Standard)
- Ley de Servicios Financieros y normativa de la Superintendencia de Bancos (Ecuador).

Justificación de Cumplimiento

La arquitectura en AWS garantiza el cumplimiento normativo mediante los siguientes mecanismos:

Requisito Normativo	Mecanismo de Cumplimiento en la Arquitectura
Protección de datos personales	Uso de AWS Cognito para autenticación segura, cifrado en tránsito (TLS) y en reposo (KMS), segmentación por VPCs.
Trazabilidad y auditoría	Implementación de CloudTrail y CloudWatch Logs para registrar acciones, accesos y eventos críticos.
Seguridad de aplicaciones	Integración de AWS WAF, Shield, y validaciones OWASP en el backend (API Gateway + Lambda).
Alta disponibilidad y recuperación ante desastres (DR)	Arquitectura distribuida en múltiples zonas de disponibilidad, uso de Amazon S3 y RDS Multi-AZ.
Autenticación robusta	Uso de OAuth2.0 con Cognito, flujos PKCE para móviles, integración con biometría (ej. FaceTec).
Monitoreo y respuesta ante incidentes	Activación de GuardDuty, Config, y alertas en CloudWatch para detección temprana de amenazas.
Segregación de ambientes y funciones	Separación por VPCs: Frontend, Backend, Seguridad, Datos, Core Bancario, con roles IAM específicos.

Riesgos Mitigados

La arquitectura propuesta mitiga los siguientes riesgos críticos para una entidad financiera:

Riesgo	Mitigación
Acceso no autorizado a datos sensibles	IAM con políticas granulares, autenticación multifactor, cifrado de datos.
Ataques DDoS o inyecciones web	Protección con AWS Shield, WAF, validaciones en API Gateway.
Pérdida de trazabilidad o auditoría	Registro completo de eventos con CloudTrail, retención en S3.
Fallas en disponibilidad del servicio	Arquitectura HA con balanceo de carga, replicación en RDS, auto-healing.
Incumplimiento normativo	Diseño alineado a estándares internacionales y locales, documentación técnica y legal.
Exposición de credenciales o tokens	Uso de flujos seguros OAuth2.0, almacenamiento cifrado, rotación de secretos.

11.Anexos.

A continuación, se adjunta los siguientes anexos de la propuesta de Banco BP:

URL del repositorio de GitHub:

https://github.com/pvjerez/Devsu_Ejercicio_Practico_Arquitecto_Soluciones.git

Plan de Trabajo y Cronograma_ Banco BP.xlsx

Plan de Trabajo y Cronograma_ Bancc

Proyección y Estimación de Costos RFP_ Banco BP.xlsx

Proyección y Estimacion de Costc

Equipo Propuesto y Dedicación_BancoBP.xlsx

Equipo Propuesto y Dedicación_BancoBI

Herramientas usadas: https://draw.io , https://plantuml.com/es/