Arquitectura para un sistema bancario

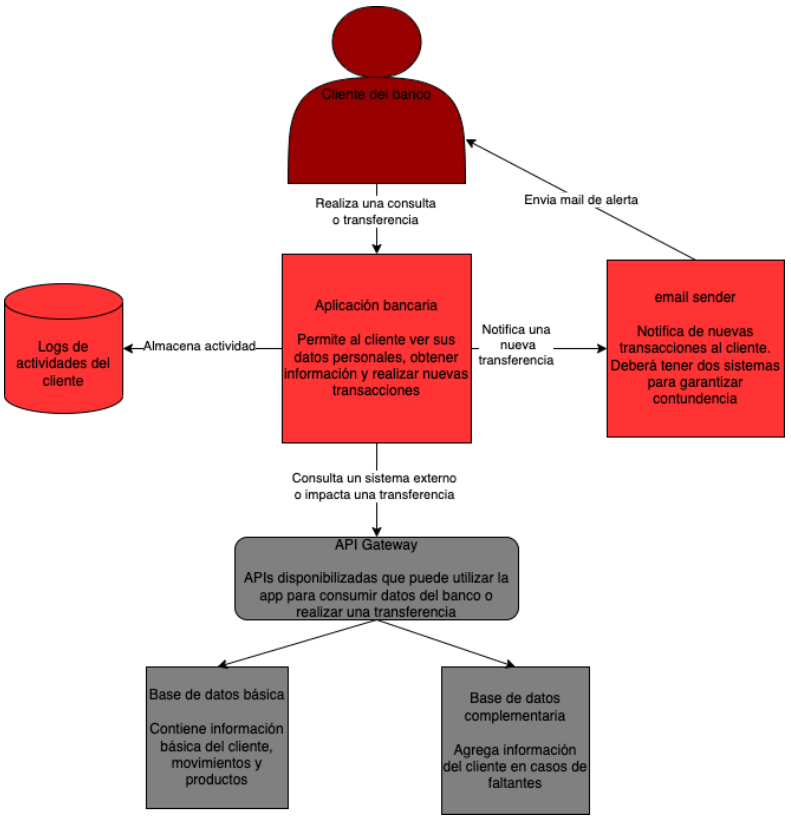
Contexto

Esta solución consiste en una app bancaria que le permitirá al cliente, ya sea desde una app desde cualquier celular con Android o iOS o un navegador:

- Consultar sus datos
- Consultar sus transacciones
- Realizar nuevas transferencias

Para ello, la app consumirá los servicios necesarios vía API que impactan en dos bases del banco, desde donde se pueden obtener informaciones básicas del cliente o más detalladas.

System Diagram Context



Análisis general

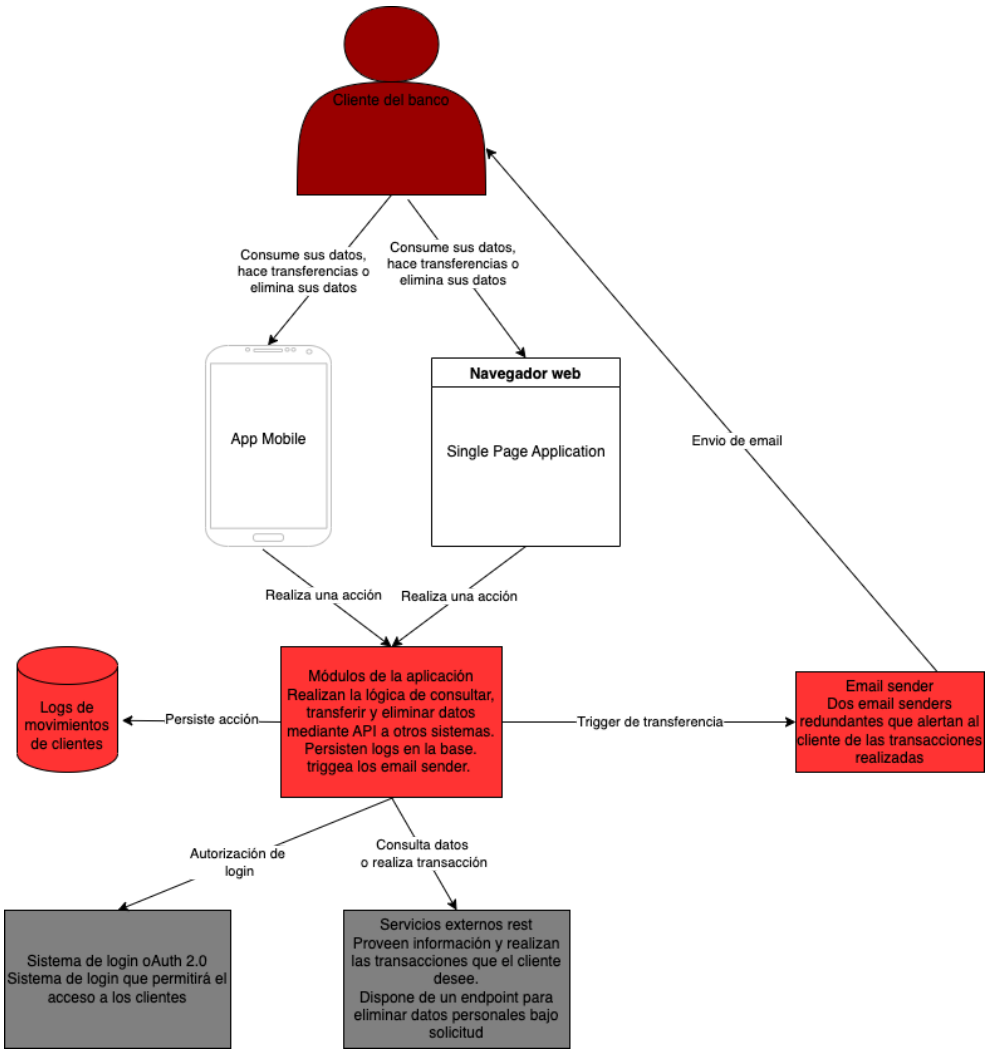
Esta app bancaria deberá atender los requerimientos mencionados con la premisa de alta disponibilidad. Para mitigar la posibilidad de una falla total en el sistema, se trabajará con un producto dividido en microservicios. Cada microservicio será el encargado de ejecutar una

tarea distinta: logueo, consultar datos básicos de clientes, consultar histórico de transacciones, realizar una nueva transacción o persistir los datos del cliente. Cada módulo será desarrollado de manera independiente: en caso de haber algún error en un release, este sólo impactará a un microservicio y no así a todo el sistema, lo que permitirá al cliente seguir operando en el resto de los módulos en el caso más extremo. Más allá de esto, se requerirá la aprobación de los cambios del equipo de QA y su jefe inmediato, acompañado de un monitoreo exhaustivo en las primeras 24hs desde su deploy en la actividad de clientes y recursos consumidos. A la primera detección de un comportamiento anormal, se debe realizar un rollback y analizar sus causales, se pueda demostrar en primera instancia o no la relación entre la modificación y la incidencia.
En cuanto al hardware, para garantizar la alta disponibilidad del sistema la solución correrá sobre AWS. Cada request de los clientes será tomada por uno de los “n” elastic load balancer de AWS encargado de distribuir la carga entre los “m” targets que estarán corriendo a los distintos módulos de nuestra aplicación. Al cada uno de ellos estar modulado en un microservicio distinto, el escalamiento según la necesidad será por módulo en base al requerimiento de los clientes: no es necesario escalar toda la solución por tener un microservicio sobreexigido. Al llegar a un límite de exigencia del servidor, el microservicio en AWS escalará automáticamente y alertará al equipo correspondiente. Cuando el equipo a cargo del seguimiento y soporte de la app detecte el porqué se debió escalar y reduzca el consumo, podrá desescalar el servicio. De esta forma, se reduce la posibilidad de incidentes por hardware. Si a esto se le suma la disponibilidad de 99.99% de AWS, la alta disponibilidad por hardware está garantizada.
Por el lado de las autenticaciones, se entiende que el desarrollo OAuth 2.0 ya está implementado en un servicio del cliente. Para intermediar, existirá un microservicio de autenticación que obtendrá el código de autorización del cliente y lo intercambiará por el código de acceso. Se presume que se autenticará mediante una concesión de código de autorización, que es lo que mejor se adapta a una plataforma con dos fronts como la que se tiene en este escenario.
Adicionalmente, el desarrollo requerirá almacenar los logs de los movimientos que hagan todos sus clientes allí. Para ello, se tendrá un microservicio encargado de almacenar todos estos movimientos independiente al resto de los módulos, que estos últimos solamente alertarán de los cambios.
Las tres principales funcionalidades del sistema estarán abarcadas cada una en un microservicio distinto, responsable de:

- Consultar los datos básicos del cliente
- Consultar sus transacciones
- Realizar nuevas transferencias.

En este punto es importante aclarar que según la ley argentina de tratado de datos personales exige que el dueño de ellos pueda eliminarlos con tan solo solicitarlo. Por otro lado, las transacciones no pueden ser eliminadas ni modificadas ya que es un registro de los movimientos hechos por el cliente. Por ello, el módulo de consultas de datos básicos de clientes será también el encargado de eliminar sus datos personales en caso que el cliente así lo desee.
Finalmente, el sistema contará con dos fronts. La tecnología en la que se desarrollará cada uno será explicada en el siguiente apartado

Container Diagram

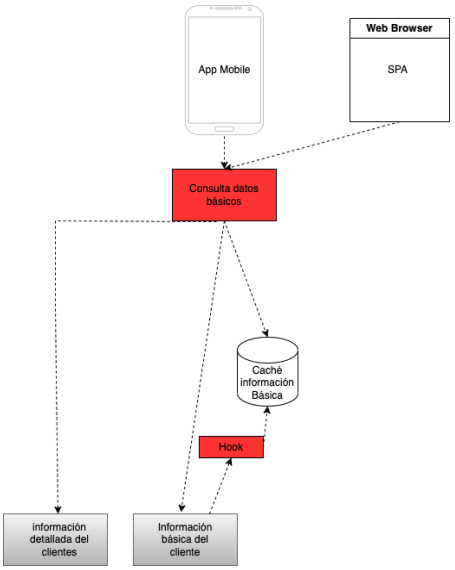


Análisis de cada módulo

Este sistema bancario contará con dos fronts. En lo que respecta a esta primera parte, el trabajo de análisis se basará en una fluida experiencia de usuario en su front end. Debido al avance de las tecnologías, los usuarios cada vez son menos tolerantes a los tiempos de carga para acceder al contenido.Por ello, para el SPA como primera opción se recomienda la utilización de Gatsby JS. Esta tecnología permite tener las partes del front estáticas renderizadas en la CDN y los datos dinámicos serán solicitados mediante API por el navegador. Al ser una SPA que realizará pocas acciones que se verán reflejadas en el front, esta es una buena opción para tener una experiencia de usuario fluida. Por otro lado, se debe realizar el front de una app multiplataforma. Si se elige a Gatsby JS para el desarrollo del front web, la mejor opción que se puede tener para mobile es React Native. Al ser lenguajes de programación similares, se pueden emplear los mismos desarrolladores front end para ambas partes del proyecto: ambas tienen como base a JS. Este framework de trabajo dará un rendimiento similar a los de componentes nativos de las distintas plataformas mobile.

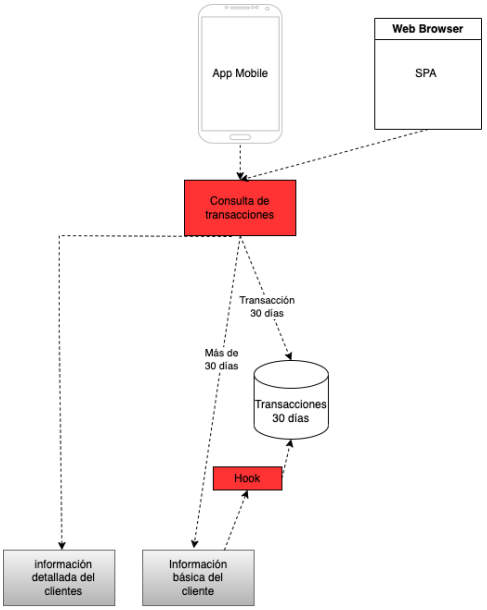
Ambos fronts interactuarán mediante API con las tres funcionalidades básicas de la plataforma, que serán un microservicio de nuestra solución:

- Consulta de datos básicos: cuando se requiera esta información, se realizará un get desde el front hacia el módulo de procesamiento de datos básicos con el ID del cliente autenticado. Este módulo, para la primera request que se realice del cliente, tomará la request y le solicitará los datos al sistema core del banco que posee la información básica de todos los clientes. Para evitar un cuello de botella en el sistema core del banco, almacenará una caché de esta información en una base no relacional. Ante una nueva consulta a esta API, está primero irá a verificar si los datos de este cliente existen en la base cacheada y recién luego impactará nuevamente al sistema bancario. Al ser una base de datos no relacional, su tiempo de respuesta será acotado y mejorará además la performance de la app. Entendiendo que los cambios en los datos básicos del cliente suelen ser pocos, existirá un hook que escuchará los cambios en este sistema bancario sólo para los clientes que hayan hecho alguna vez alguna consulta durante los últimos 30 días sobre sus datos personales y actualizará los cambios. Para clientes mayores a 30 días sin consultas, se removerá la entrada de la base. Si en algún caso se requiere información detallada, se consultará mediante API al sistema bancario destinado para dicho servicio. Finalmente, este módulo será el encargado de eliminar los datos del cliente en caso de que este así lo desee. Para ello, impactará dos requests vía API con un delete a los servicios externos que poseen los datos simplificados del cliente y los detallados. También así, eliminará de la caché los datos del cliente almacenados, garantizando el cumplimiento de la legislación argentina. El diagrama de componentes simplificado para esta funcionalidad quedaría de la siguiente manera:

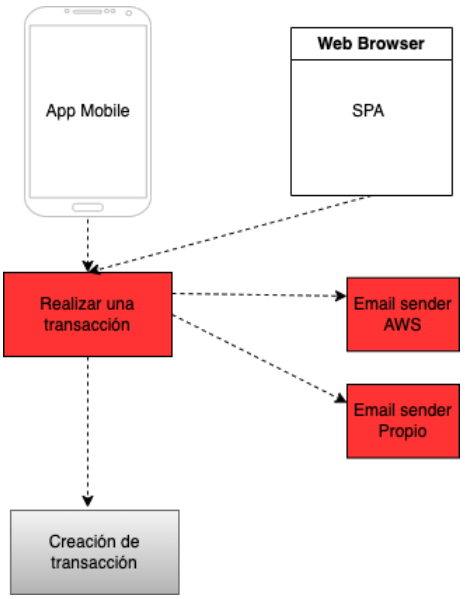


- Consulta de transacciones: Ante una consulta de transacción, el front realizará un get pasando en forma encriptada el ID del cliente autenticado y, por parámetro, el rango de fechas. El módulo encargado de este tratamiento consultará a una base no relacional que contiene las transacciones de los últimos 30 días de los clientes del banco. En el caso de que el rango de tiempo sea mayor, consultará al sistema bancario externo con información detallada del cliente. Para mantener este sistema sincronizado, entendiendo que una transacción nunca puede ser eliminada del sistema, existirá un hook que estará escuchando al sistema bancario e irá

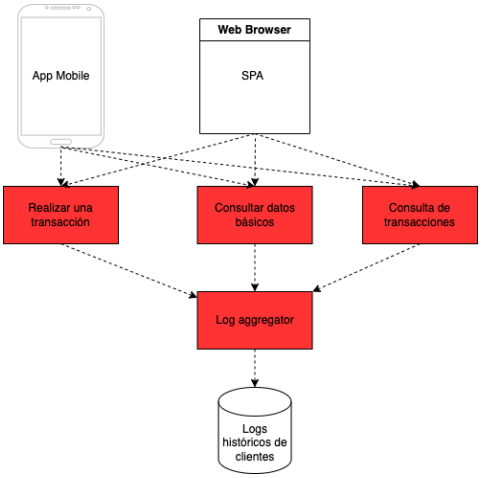
actualizando las nuevas transacciones ocurridas en la base. Todos los días a la medianoche, la base eliminará transacciones mayores a 30 días. Si en algún caso se requiere información detallada, se consultará mediante API al sistema bancario destinado para dicho servicio. El diagrama de componentes simplificado para esta funcionalidad quedaría de la siguiente manera:



- Realización de transacciones: al querer realizar una transacción, el front disparará un post con el ID del cliente autenticado. En el body de la request, colocará el CBU o alias de la persona de la que se requiere realizar la transacción. En caso de ser una primera transferencia a un nuevo cliente de la lista, se deberá sumar encriptado en el body el token correspondiente para aceptar la transferencia. El módulo encargado de dicho trabajo obtendrá la request, analizará la veracidad de los datos y requerirá al sistema externo mediante un post la realización de esta nueva transferencia. A partir de la respuesta del sistema externo de transferencias, en caso de que haya sido exitosa, el módulo le dará aviso al email sender de AWS que debe enviar la notificación al cliente sobre la transacción. Si pasados 5 minutos no se obtiene respuesta o esta no es satisfactoria, se realiza una request al sistema on premise de email sender para enviarlo. En este punto, se toma como premisa que los sistemas externos tendrán la información de las transacciones actualizadas sin nuestra intervención, por lo que el hook actualizará el estado de las transacciones al recibir una notificación de la base. En caso de que alguno de los datos sea incorrecto y esto se detecte ya sea en el módulo propio o en el externo, brindando algún tipo de error, el módulo propio hará un wrapper de él y mostrará un mensaje al cliente en caso de que corresponda para que corrija y reintente. El diagrama de componentes simplificado para esta funcionalidad quedaría de la siguiente manera:



Como bien se enuncia, es necesario poder almacenar logs de todos los movimientos que los clientes realizan en nuestros sistemas. Para ello, se utilizará un cuarto módulo con el patron de diseño log aggregation: Cada aplicación deberá informar al módulo de log de movimientos de clientes cada acción que se realiza desde la app. Con ello, este módulo (que conformará a un cuarto microservicio) almacenará los logs de información en la base correspondiente unificada. El diagrama de componentes simplificado para esta funcionalidad quedaría de la siguiente manera:



Finalmente, el módulo de autenticación y registro en el sistema tomará los datos de registro del cliente e interactuará con el sistema ya desarrollado para logueo de clientes, validando que el cliente esté autenticado. Una vez con esa información, utilizará el ID client desde los diversos microservicios para poder realizar las peticiones mediante API. Caso contrario, si el acceso es denegado se alertará al cliente. Luego de tres intentos, el usuario se bloqueará y deberá volver a generar su token en el cajero automático del banco.