Diseño de una arquitectura basada en microservicios para una Wallet Electrónica

Byron Giovanny Cholca Campués

Maestría en ingeniería en Software, Universidad Nacional de Loja, Loja, Ecuador

Resumen.- **Los medios de pago electrónicos han presentado una gran acogida por parte de los agentes económicos en Ecuador. Las empresas de servicios financieros de pago (Fintech) PayPhone y PeiGo, publicaron sus nuevas billeteras virtuales en marzo del 2022 y la billetera de Google empezó a operar en Ecuador en febrero 2023.**

El presente artículo tiene como objetivo proponer un diseño arquitectura de software para una Wallet Electrónica. El enfoque teórico en el que se fundamentó el diseño de la arquitectura fue el concepto de microservicios con patrones arquitectónicos que permitan mejorar aspectos como la escalabilidad y seguridad, y mediante pruebas de concepto evaluar el rendimiento en un ambiente Cloud con un patrón arquitectónico API Gateway.

Para lograr el cumplimiento de este objetivo, se utilizó la metodología Incremental, dividiendo el trabajo en 3 incrementos, el primero fue realizar una revisión sistemática de la literatura en librerías científicas, seleccionando artículos que dan respuestas a las preguntas de investigación previamente definidas, y de estos estudios primarios seleccionados, obtener el conocimiento teórico en patrones arquitectónicos y arquitectura basada en microservicios tales como Api Gateway, Servicios Lambda, Base de Datos Por Servicio, Circuit Breaker y Bus de Mensajes. El segundo objetivo fue proponer la infraestructura de software de la arquitectura en una solución Cloud para ser implementada en AWS, se modelo la arquitectura del software fue expresado mediante diagramas de despliegue y de componentes que dan respuesta a los requerimientos funcionales y no funcionales establecidos. El tercer objetivo fue evaluar el rendimiento mediante pruebas de concepto y validar el funcionamiento del patrón arquitectónico Api Gateway junto con APIs publicadas en un ambiente Cloud con servicios Lambda de AWS.

Palabras clave.- **Wallet electrónica, arquitectura de software, microservicios, Cloud, AWS, Api Gateway, patrones arquitectónicos.**

INTRODUCCIÓN

El inicio de la pandemia del Covid-19 en 2020 provocó un aumento en la demanda de servicios digitales por parte de los latinoamericanos, lo que impulsó a las empresas a proporcionar sus servicios de manera diferente. Un ejemplo es el uso de pagos digitales por parte de los gobiernos para proveer alivio financiero a los grupos más vulnerables. De la misma forma, los propios ciudadanos buscaron maneras de realizar transacciones en formatos seguros para cumplir las reglas de distanciamiento social [1]. Aunque la pandemia ha llevado a un aumento en la tenencia de efectivo por parte de los agentes económicos, también ha permitido una aceleración en la adopción de medios de pago electrónicos y la transformación digital en los servicios financieros. Esto ha contribuido a la inclusión financiera de personas con ingresos medios y bajos, según estudios realizados por el Banco Interamericano de Desarrollo y The Economist [2]. En Latinoamérica, el uso de billeteras móviles está en aumento, incluyendo en Ecuador. Según el informe "The Global Payments Report", los puntos de venta (almacenes, locales comerciales o tiendas) representan el 8% de los pagos totales pero se espera que el uso de billeteras digitales se duplique para 2025 en la región [3].

La banca en Ecuador está promoviendo un sistema financiero electrónico que representa un gran avance tecnológico. Para lograr esta innovación, las Fintech están trabajando en Billeteras Electrónica que permitan realizar pagos y transferencias de manera sencilla a través de recargas electrónicas desde el celular[4]. y la arquitectura de software debe permitir integrarse a servicios de medios de pago y procesar un mayor número de transacciones, pues una aplicación de este tipo tiene que ser altamente transaccional y tiene que soportar mucha carga de peticiones y como resultado permitir bancarizar de cierta forma a grupos sociales antes rezagados [5].

Por lo expuesto, este artículo tuvo como objetivo general Proponer el diseño de una arquitectura de software basada en microservicios para una Wallet Electrónica la cual permita una adecuada integración con otras aplicaciones y servicios de clientes, proveedores y empresas afiliadas que deseen utilizar este medio de pago electrónico.

En consecuencia el presente artículo se divide en varias secciones, empezando por la sección de Materiales y Métodos donde se describe el área de estudio, los procedimientos y los recursos empleados. Luego la sección de Resultados se presenta de manera detallada las evidencias obtenidas durante el cumplimiento de cada objetivo planteado. A continuación, en la sección de Resultados, se presentan detalladamente las evidencias obtenidas durante el desarrollo de cada uno de los objetivos planteados. En la sección de Discusión, se realiza un análisis de los resultados obtenidos desde el punto de vista del autor. Finalmente, en la sección de Conclusiones, se presentan los hechos más relevantes tras completar el trabajo planificado.

MATERIALES Y METODOS

Para llevar a cabo el trabajo propuesto, se utilizaron los siguientes recursos:

Recursos científicos

Investigación bibliográfica

Se realizó la investigación sobre las tecnologías predominantes utilizadas en el diseño de la arquitectura basada en microservicios, realizando una revisión sistemática de la literatura, que consistió en la recopilación de documentación y artículos, con arquitecturas basadas en microservicios.

Recursos técnicos

Entrevista

Aplicando la técnica de Entrevista no Estructurada de preguntas y respuestas se identificó del estado arquitectónico de la empresa Quikly

RESULTADOS

En esta sección se presentan los resultados obtenidos para cumplir con el presente trabajo.

A. Identificar la tecnología, metodología y arquitectura de software basada en microservicios, que se emplea, apoyado en documentación, reuniones agendadas a los técnicos desarrolladores, para conocer el modelo de desarrollo de aplicaciones que utilizan, así como indagar en los patrones de diseño a utilizar.

La finalidad de este objetivo consiste en identificar el estado de la arquitectura actual, en base a un conjunto de técnicas (entrevista no estructurada) al personal técnico, de la empresa Quikly, definir la arquitecturas de software, los requerimientos funcionales y no funcionales, actores y sus respectivos roles que interactúan con el aplicativo junto con los respectivos casos de uso, seguidamente se realizó una revisión sistemática, para identificar las tecnologías predominantes en la arquitectura de microservicios, y patrones que en ella se utilizan, y finalmente; obtener el modelo de requerimientos.

Como resultado de la entrevista se llegó a conocer que la empresa Quikly se encuentra en un proceso de transformación digital, migrando las aplicaciones con arquitecturas monolíticas a microservicios, y así entendiendo el estado actual de la arquitectura de software. El equipo de desarrollo no dispone de una arquitectura de software definida, las aplicaciones se implementan bajo un criterio establecido en lineamientos tradicionales y por propia experiencia, siendo empíricos. La falta de una arquitectura basada en microservicios puede limitar la escalabilidad, la flexibilidad y la capacidad de respuesta de la Wallet Electrónica, lo que puede tener un impacto negativo en la satisfacción del usuario y la eficiencia de la empresa, adicionalmente se definen tambien los requerimientos funcionales Tabla 1 y no funcionales Tabla 2.

Código	Descripción	Proceso
RF-01	Creación de cuenta mediante registro de información personal.	Autenticación
RF-02	Creación de cuenta mediante integración con cuentas de Google o Facebook.	Autenticación
RF-03	Servicios para el registro y verificación de tarjetas	Autenticación
RF-04	Integración y consumo del API del proveedor Spreedly para registro de información sensible del usuario	Transacción
RF-05	Cobro de micro transacciones por registro de tarjeta integrando el API del proveedor Fintech D-Local	Transacción
RF-06	Registrar de valores cobrados	Transacción
RF-07	Verificar montos para registro de tarjetas	Transacción
RF-08	Generación de link de pagos mediante código QR.	Integración
RF-09	Búsqueda de usuarios para generar el pago.	Integración
RF-10	Obtener Token de la tarjeta registrada.	Integración
RF-11	Consumir el API del proveedor Spreedly para registro del pago con el token obtenido.	Integración
RF-12	El monto de la transacción se suma al saldo del beneficiario.	Integración
RF-13	Verificación de información de tarjeta para recibir pagos y ventas.	Integración
RF-14	Integración con servicio de reconocimiento facial del proveedor OnDato.	Integración
RF-15	Servicios para catálogos	Integración

Tabla 1 Requerimientos funcionales.

Código	Categoría	Descripción
RNF-01	Seguridad, Compatibilidad	Las integraciones deben ser transparentes y seguro para el usuario.
RNF-02	Seguridad	Los servicios deben tener autenticación y autorización mediante encriptación JWT.
RNF-03	Seguridad	La información de tarjetas, datos personales debe ser segura y no expuesta a terceros.
RNF-04	Disponibilidad	Los microservicios deben estar disponible en todo momento, ser tolerantes a fallos y ser capaces de recuperarse rápidamente en caso de que ocurra una falla en algún servicio.
RNF-05	Escalabilidad	Ser capaz de manejar grandes volúmenes de transacciones simultáneas sin sufrir una reducción en su rendimiento y que puedan ser replicados y distribuidos en diferentes servidores.
RNF-06	Accesibilidad	La Wallet debe ser fácil de usar y entender para el usuario promedio.
RNF-07	Compatibilidad	La Wallet debe ser compatible con varios dispositivos y sistemas operativos.
RNF-8	Interoperabilidad	Los microservicios deben ser diseñados para ser compatibles con diferentes lenguajes de programación, protocolos de comunicación y tecnologías de integración.
RNF-12	Rendimiento	Los microservicios deben manejar grandes volúmenes de datos y transacciones, y ser capaces de procesar solicitudes en un tiempo mínimo.
RNF-9	Facilidad de mantenimiento	Los microservicios deben ser diseñados para ser independientes y desacoplados, y mantenido de manera independiente sin afectar el resto del sistema.
RNF-10	Compatibilidad	La aplicación debe ser compatible con estándares de desarrollo web HTML, CSS y JS.
RNF-11	Adaptabilidad	El proceso de desarrollo debe ser usado con metodologías ágiles que sean lo suficientemente flexibles para adaptarse a cambios en los requerimientos.

Tabla 2 Requerimientos no funcionales

Como resultado de usar la revisión sistemática de la literatura aplicada a arquitectura de microservicios se identifican documentos y artículos llamados ahora estudios primarios que contienen información sobre: buenas prácticas,

conceptos sobre patrones arquitectónicos y técnicas asociadas con la arquitectura de software basada en microservicios. Con el fin de comparar los resultados de los diferentes artículos, se ha organizado la información recopilada en categorías temáticas comunes. En la Tabla 3 se detallan las bases teóricas y de esta manera, se simplifica la síntesis de los hallazgos y se obtiene una visión más clara de los desafíos actuales.

Proceso	Funcionalidad	Descripción
Despliegue	Contenedores	Los contenedores son esenciales para implementar y orquestar microservicios de manera eficiente, Docker o Lambdas.
	Orquestación de contenedores	Herramientas como Kubernetes y Docker Swarm permiten gestionar y orquestar contenedores en producción.
	Un servicio por host	Este método consiste en desplegar los microservicios en una única maquina física.
	Múltiples servicios por host	En maquina se implementan varios microservicios utilizando máquinas virtuales o contenedores.
Comunicación	Servicios web	Los servicios web RestFull para implementar microservicios, usando tecnologías como HTTP, JSON y XML.
	Balanceo de carga	Las herramientas de balanceo de carga, como NGINX, permiten distribuir la carga entre múltiples instancias de microservicios.
	API-Gateway	La comunicación con el exterior se da a través de un microservicio que funciona como un único punto de acceso.
DevOps	Integración y entrega continuas	La práctica de integración y entrega continua son fundamentales para la implementación exitosa de microservicios. Jenkins es útil para implementar estas prácticas.
Control de ejecución	Orquestación.	Este método consiste en utilizar un microservicio principal para controlar la ejecución de los demás
	Coreografía.	El control de la ejecución es descentralizado, cada microservicio reacciona en base a eventos.
Almacenamiento de datos	Bases de datos no relacionales.	Son una buena opción para implementar microservicios debido a su escalabilidad y flexibilidad, como MongoDB.
	Base de datos compartida.	Todos los microservicios acceden a la misma base de datos, sus datos se separan en diferentes esquemas
	Base de datos por servicio.	Se implementa para cada uno de los microservicios una base de datos diferente.

Tabla 3 Bases teóricas

A continuación en la se definen los actores e identifican los actores y respectivo rol que cumplen en la aplicación.

Actor	Rol
Usuario	Usuario de la Wallet
Facebook	Red social para integrar al login
Gmail	Red social de correo para integrar al login
OnDato	Empresa proveedora del servicio de reconocimiento facial.
Operadora Móvil	Operadora móvil para envió de SMS
Spreedly	Empresa proveedora de servicio de cobro de tarjetas.
DLocal	Empresa proveedora de servicio de almacenamiento de información bancaria.

Tabla 4 Actores y roles

En la Fig. 1 presenta el diagrama de casos de uso correspondiente al proceso de Autenticación.

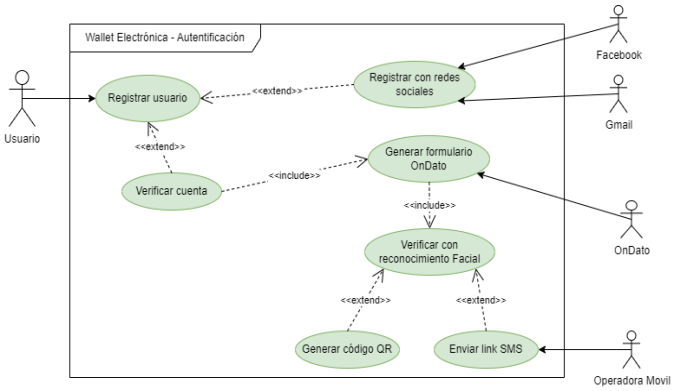


Fig. 1 Caso de uso Autenticación

Mediante este proceso de investigación y las técnicas mencionadas se logró cumplir con el primer objetivo específico

B. Proponer una infraestructura de software aplicando una arquitectura basada en microservicios en una solución Cloud que presente las capacidades óptimas de administración necesarias para una Wallet electrónica.

A continuación, se indica el diagrama arquitectónico propuesto en base a los requerimientos funcionales y no funcionales, como se detalla el la Fig. 2

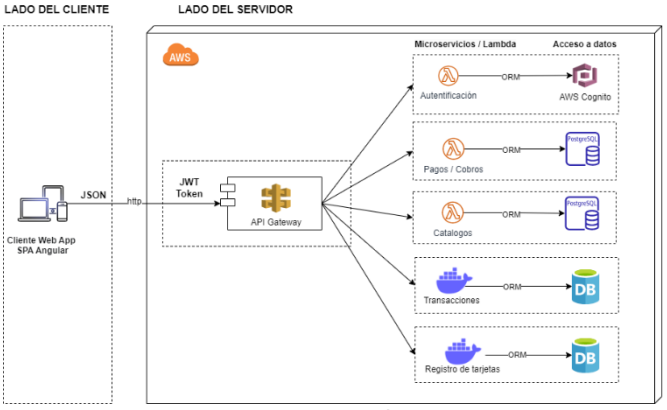


Fig. 2 Diagrama arquitectónico propuesto

En la Fig. 2 se muestran los componentes que formarán parte de la arquitectura de software basada en microservicios juntos con los patrones y estilos arquitectónicos en una solución Cloud. Este diagrama arquitectónico para una aplicación basada en microservicios incluye: el cliente web, la capa de API Gateway, los microservicios con su respectiva base de datos. Cada uno de estos componentes debe ser independiente y escalable para garantizar la disponibilidad y el rendimiento de la aplicación. El diagrama arquitectónico para Wallet Electrónica basada en microservicios posee la siguiente estructura:

Cliente: Los clientes son aquellos que consumen los recursos que ofrece el microservicio. Para poder acceder a estos recursos, los clientes deben autenticarse mediante un proceso de autenticación. Las peticiones Response y Request son en formato JSON

JWT Token: es el estándar de seguridad utilizado en la aplicación, permite el acceso a las APIs implementadas en los microservicios. El cliente se autentifica mediante el servicio de

Autenticación si el usuario y contraseña son correctos el servicio retorna un token único, de esta manera, se puede verificar que el cliente está autorizado o denegado para acceder a los recursos de los microservicios. El token de comunicación debe ser incluido en el Header de cada petición, como se describe en la Fig. 3

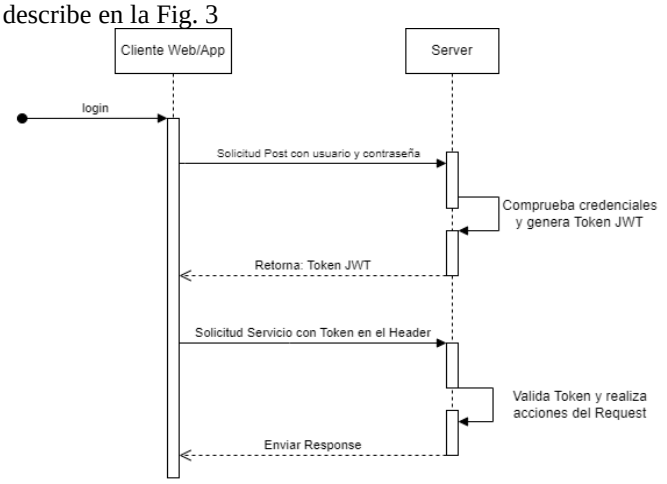


Fig. 3 Diagrama Secuencia del proceso de Autenticación con JWT

Auth2: El protocolo Auth2 permitirá delegar la autenticación y la autorización de los usuarios en servicios de terceros como Google o Facebook. En la Fig. 4 se describe el proceso de este protocolo.

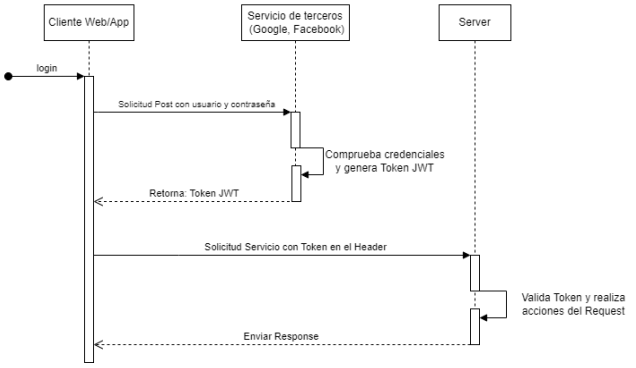


Fig. 4 Diagrama de Secuencia de Protocolo Auth2

Capa de API Gateway: Este componente debe actuar como un punto de entrada para la aplicación enrutando las solicitudes de los clientes a los servicios apropiados.[6] La elección de este patrón es resolver el problema de accesos a los microservicios y aporta otra ventaja como trabajar como un proxy. El cliente realiza una solicitud de un recurso al API Gateway, el cual reenvía la solicitud al servidor adecuado, espera la respuesta y envía la respuesta de regreso al cliente, adicionalmente en algunos casos el cliente tenga que comunicarse con varios servicios para obtener una respuesta, entonces el API Gateway expondrá otro servicio orquestador, y reenvía la solicitud del cliente a los servicios correspondientes y finalmente agrupando las respuestas obtenidas en una única respuesta para el cliente.

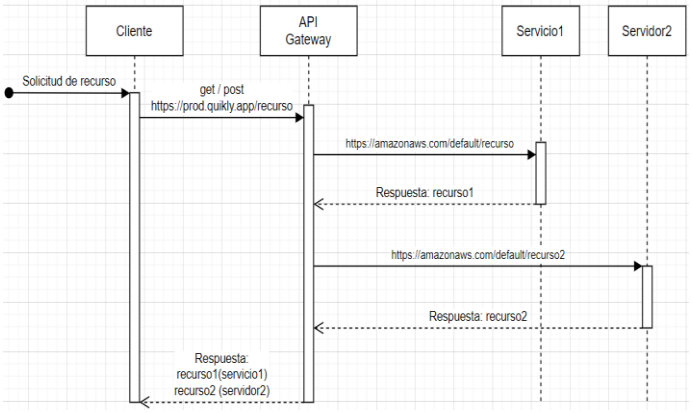


Fig. 5 Arquitectura Api Gateway - Solicitud de servicios.

Autenticación: Es el encargado de: (a) gestionar el acceso a la aplicación a través de los roles y permisos configurados en ella y (b) generar un token de autorización a los recursos a través del estándar JWT. En la Fig. 6 de detalle el diagrama de arquitectónico y el proceso de autorización JWT a la Wallet.

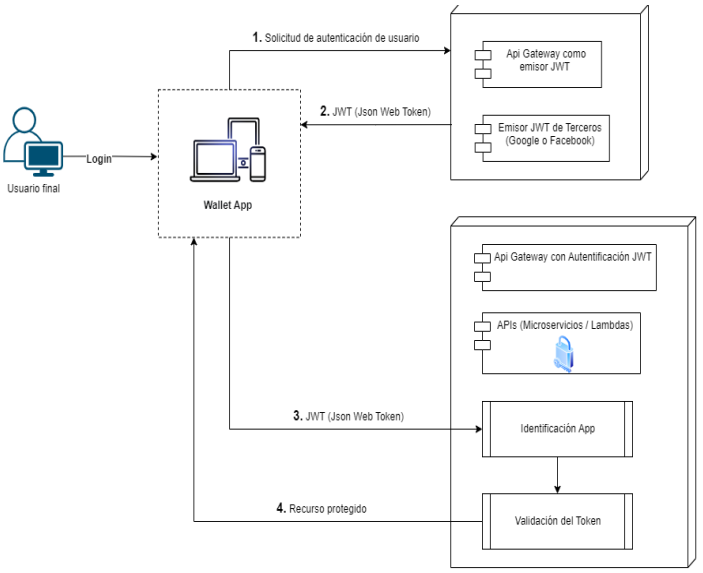


Fig. 6 Flujo de trabajo de autorización de JWT

1. Autenticación del usuario: El usuario proporciona sus credenciales de inicio de sesión al servicio de autenticación del Api Gateway o utiliza el emisor JWT de terceros para autenticarse mediante redes sociales. **2. Generación y Envío del JWT:** Si las credenciales son válidas, el servicio de autenticación genera un JWT firmado digitalmente que contiene información sobre la autenticación del usuario. El Api Gateway envía el JWT al cliente, con la respuesta de inicio de sesión. **3. Inclusión del JWT en las solicitudes posteriores:** El cliente incluye el JWT en las solicitudes posteriores al Api Gateway, en la Cabecera Authorization con el tipo de autenticación Bearer. **4. Validación del JWT:** El Api Gateway verifica la validez del JWT y autoriza o deniega el acceso según corresponda. **Estructura del token:** Después de que el cliente se autentique y autorice el acceso correctamente, la aplicación recibirá un token de acceso. La estructura del Token se puede visualizar en la Fig. 7, La respuesta incluye el accessToken, refreshToken, idToken.

```
{
    "code": 0,
    "description": "Success",
    "detail": {
        "parentId": "2023/04/08/[$LATEST]9b4a7f0974ea4df987a1b4f4dc8b2bc6",
        "localId": "584fc1ab-f866-438f-9d74-b181bc43517b",
        "accessToken": "eyJraWQiOiJaNUc2R0ZvellUbHZJU3oxdURPOVRYMVwvdGhteEFjel
        "refreshToken": "eyJjdHkiOiJKV1QiLCJlbmMiOiJBMjU2R0NNIiwiYWxnIjoiUlNBL
        "idToken": "eyJraWQiOiI3XC9XdVNrMlJQUWQxMWtRKzFsY054UnJsMEpBbEh3MkZMNk
        "client_data": {
            "name": "Byron",
            "family_name": "Cholca",
            "email": "byroncholca@gmail.com",
            "user_id": "1259b2e8-f2a4-4291-a537-1258708d5007"
        }
    }
}
```


Fig. 7 Estructura del Token en formato JSON

Comunicación de microservicios: Cuando el cliente requiere hacer una petición a varios recursos, se propone en la arquitectura desarrollar un API Orquestador [7] según como el negocio requiera. La orquestación, es donde uno de los microservicios actúa como si fuera el director de una En orquesta y le indica a cada microservicio qué debe hacer en cada momento, y tomando la información requerida para remitir al cliente en una sola respuesta como se describe a continuación:

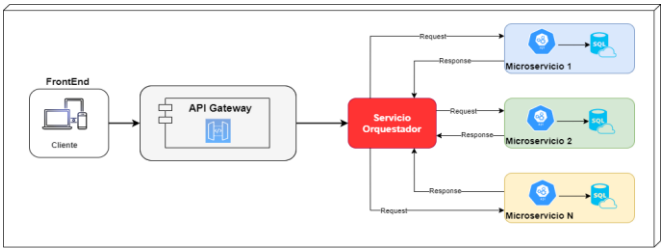


Fig. 8 Diagrama arquitectónico Servicio Orquestador

Sistema de mensajería: Para facilitar la comunicación entre los microservicios del sistema, se propone utilizar la técnica de integración a través de mensajería. Esto implica que un microservicio no se comunica directamente con otro microservicio en forma síncrona, para instruir una acción, sino que cada microservicio está observando su entorno y actúa en consecuencia de los eventos autónomos. La implementación del bus de eventos con **RabbitMQ** [8] permite que los microservicios se suscriban a eventos, los publiquen y los reciban. Siendo RabbitMQ quien controle la distribución, y el intermediario entre el publicador de mensajes y los suscriptores, como se describe en la Fig. 9

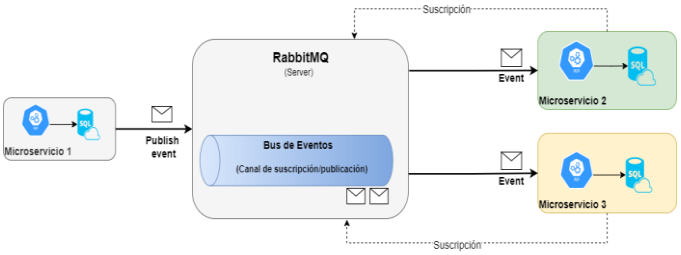


Fig. 9 Diagrama arquitectónico Bus de Eventos.

Implementación de los microservicios: Se propone usar contenedores en procesos que requieran mayor lógica de negocio y Lambdas para el caso AWS. De este modo, cada microservicio se empaqueta junto con las aplicaciones y bibliotecas necesarias para su funcionamiento.

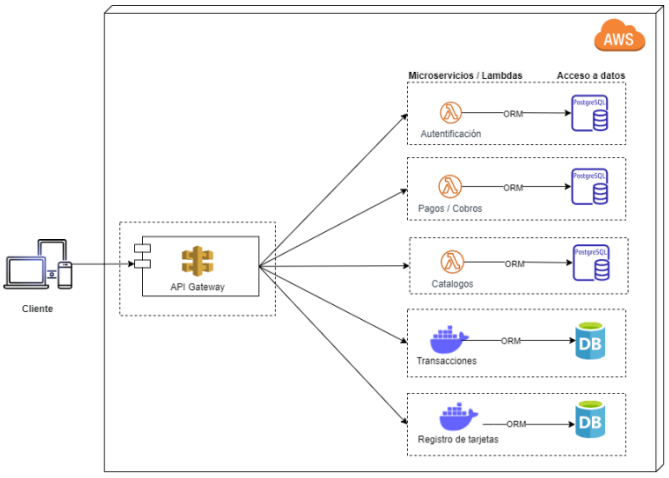


Fig. 10 Diagrama arquitectónico servicios en Docker y Lambdas

La elección entre contenedores y arquitecturas serverless para los nuevos microservicios de la Wallet dependerá de los requisitos específicos de la aplicación, como la escalabilidad, el tiempo de ejecución, la seguridad y los costes. Se debe evaluar cuidadosamente cada opción antes de tomar una decisión.

Base de datos: Este componente usa el patrón Base de Datos por Servicio la cual cada microservicio tiene su propia base de datos.

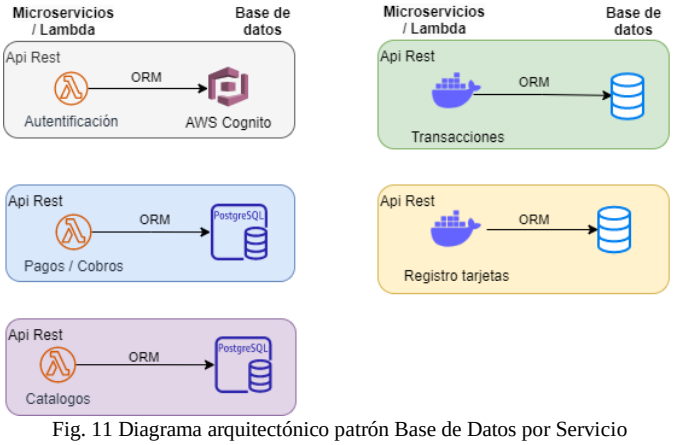


Fig. 11 Diagrama arquitectónico patrón Base de Datos por Servicio

Tolerancia a fallas: Con el objetivo de que los microservicios de la aplicación sean tolerantes a fallas, se propone implementar el patrón Circuit Breaker [9] en cada microservicio. Este patrón se basa en la idea de cerrar temporalmente la comunicación con un servicio cuando se detecta que está fallando, con el fin de evitar que el fallo se propague por todo el sistema. Con esta propuesta se logra que no saturemos a los servicios de la Wallet y así poder tener una acción de respuesta.

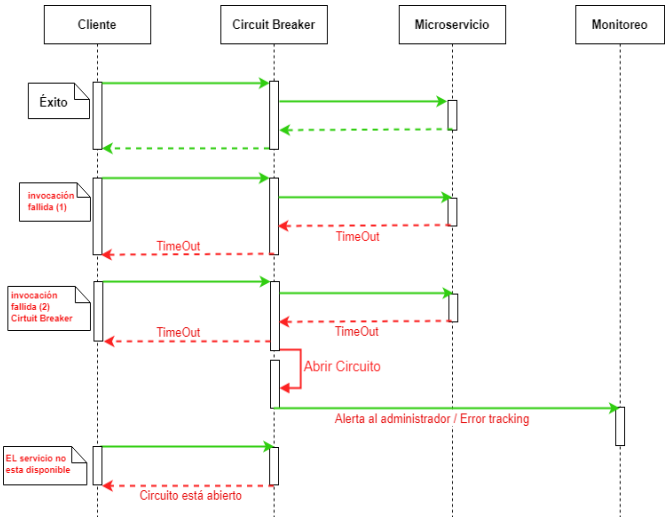


Fig. 12 Diagrama de Secuencia patrón Circuit Breaker

C. Evaluar el rendimiento y validar el funcionamiento de la solución basada en microservicios en la Cloud mediante cargas de estrés

En esta sección se realizaron pruebas de concepto en diferentes escenarios para validar la comunicación de las Apis que ha sido publicadas en un ambiente Cloud AWS. Se valida un Api para la autenticación y autorización de un usuario junto con la generación de un Token JWT para consumir otros microservicios, se analiza que los códigos y formatos de respuesta sean correctos y se medición de los tiempos de respuesta. Los Casos de prueba se realizaron en los siguientes microservicios:

Microservicio	Caso de prueba
Autenticación	Comprobar que el microservicio realiza la autenticación y autorización de manera adecuada utilizando el protocolo JWT, y que las respuestas y errores del microservicio son coherentes y acordes a lo esperado.
Usuario	Comprobar que el microservicio retorne la información del cliente registrado en la Wallet
Transactions	Comprobar la seguridad del microservicio, enviar al Api en el Header Authorization el Token de Autenticación para obtener el Response, caso contrario el Api retorna que no tiene autorización.

Tabla 5 Caso de prueba

Resultados de las pruebas de concepto:

Las pruebas de concepto con ayuda de la herramienta Postman fue consumir un API de autentificación para un usuario registrado y generar un JSON Web Token junto con la información del usuario, seguido consumir otra API con seguridad de autentificación y con el Token generado anteriormente tener acceso al recurso, y si el Token es incorrecto el API devuelve un mensaje de No Autorizado, la tercera prueba de concepto consistió en realizar cargas de estrés al microservicio de catálogos de la aplicación con pruebas automatizadas. En cada una de las pruebas se validaron que las respuestas sean correctas en base a los requerimientos y se midió que los tiempos de respuesta sean óptimos.

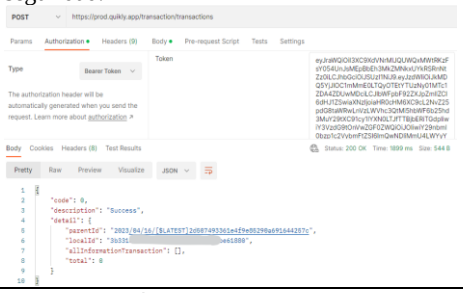
Microservicio	Respuesta caso de prueba
Autenticación	El Response del Api retorna la descripción del cliente con el respectivo el accessToken, refreshToken y idToken. La herramienta nos da información sobre el código, el tiempo y el peso de la respuesta que son satisfactorios. 
Usuario	Se observa una alerta porque el microservicio se demora más de 200ms, la cual no generar ningún error en el proceso y con esto se evalúa que el Api Gateway cumple las pruebas de concepto. 
Transactions	Los escenarios de prueba fueron correctos al usar el primer Api de Autenticación y luego con el Token obtenido consumir otro servicio con seguridad. 

Tabla 6 Evaluación casos de prueba

Con estas pruebas de concepto se puede demostrar que la arquitectura propuesta basada en microservicios implementada en AWS cumple con las características esenciales de escalabilidad e integración de componentes o módulos y sobre todo con un mínimo acoplamiento y alta cohesión con el resto de los servicios.

DISCUSION

El uso de la Investigación Proyectiva para la recopilación de información fue de gran aporte, las técnicas de Entrevistas no Estructurada y la Recopilación de Documentación, logró definir los requerimientos funcionales, no funcionales y casos de uso con sus respectivas interacciones con el aplicativo, esto ayudó a determinar el estado arquitectónico actual de la empresa ya que se encuentra en un proceso de transformación digital. Por

medio de una revisión sistemática de la literatura de los artículos de revistas científicas ayudó a tener una noción general y bases teóricas sobre arquitectura de software, patrones arquitectónicos y arquitectura basada en microservicios.

El conjunto de los componentes y patrones arquitectónicos para el diseño de la arquitectura basada en microservicios permite realizar cambios en cualquier microservicio de manera eficiente, ya sea para corregir fallos o para realizar actualizaciones y mejoras. Estos cambios pueden realizarse de forma transparente para los clientes que utilizan los servicios, lo que permite un escalado rápido y efectivo de la aplicación.

Para el despliegue de los microservicios y patrones arquitectónicos se ha escogido al proveedor Amazon Web Service ya que es una plataforma que facilita la implementación de aplicaciones nativas en la nube gracias a su atractiva relación coste-beneficio y el apoyo técnico que brinda. El uso de AWS para la arquitectura de microservicios utilizando API Gateway, contenedores y lambdas ofrece una solución escalable, flexible, fácil de usar y segura para las aplicaciones de microservicios. Al aprovechar los servicios de AWS, los desarrolladores pueden centrarse en la creación de aplicaciones de alta calidad en lugar de preocuparse por la infraestructura y el hardware.

Las API están publicadas en funciones Lambda de AWS se realizaron 3 pruebas de concepto con los microservicios (1) el microservicio de autenticación: al consumir este servicio y enviando el mail del usuario existente en el Request se obtiene en el Response el JWT junto con la información básica del cliente, obteniendo el resultado esperado con tiempos de respuesta fueran óptimos. (2) el microservicio cliente, la prueba de concepto consistió en realizar cargas de estrés y con ayuda de la herramienta Postman se realizan pruebas automatizadas para realizar peticiones seguidas y midiendo el tiempo y el código de respuesta del Api y que retorne la información del usuario registrado. Las pruebas fueron satisfactorias, cumpliendo el formato requerido, pero la herramienta indica una advertencia que el tiempo de respuesta de Api no es menor a 2ms, teniendo como promedio de 0.23s a 1.75s de un total de 20 peticiones en pruebas de monitoreo, como se visualiza en la Fig. 13

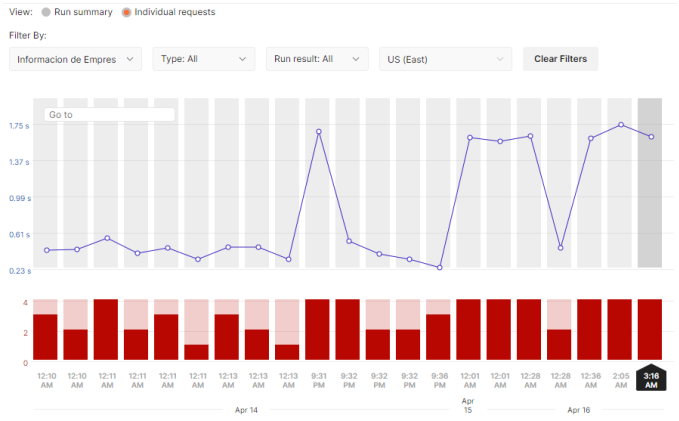


Fig. 13 Pruebas monitoreo – microservicio usuario

Para validar los requerimientos no funcionales con la última prueba de concepto (3) el microservicio Transactions tiene autenticación de seguridad, una vez generado el JWT en el microservicio de Autenticación y configurado la solicitud HTTP en Postman con el Token inyectado en el Bearer Token del microservicio se obtiene la respuesta esperada y al no enviar el token el Api no permitió obtener la información y este escenario da como resultado que la prueba de concepto sea exitosa.

La evaluación del rendimiento y la validación del funcionamiento de las APIs en microservicios en la nube de AWS es fundamental para garantizar la calidad y la disponibilidad del aplicativo. Al realizar las pruebas de las APIs con Postman es de gran aporte y ayuda para obtener una idea del rendimiento básico de la arquitectura propuesta.

CONCLUSIONES

Las evidencias presentadas en el marco teórico sobre la Billeteras Electrónicas en el Ecuador destacan la necesidad de implementar una Política Nacional de Inclusión Financiera para crear Sistemas Financieros Inclusivos. Estas políticas deben ser planificadas y desarrolladas en colaboración entre el Estado, el sector privado y la sociedad civil, para lograr objetivos a largo plazo.

Actualmente no existe un estilo arquitectónico específico, ni tampoco un lenguaje descriptivo asociado a los microservicios. Esto se debe a las características particulares de este tipo de arquitectura, se divide en múltiples servicios independientes y autónomos, lo cual resulta complejo tener una visión global de este tipo de arquitectura. La propuesta arquitectónica modelada se enfoca en las prácticas más relevantes, pero existen múltiples formas de implementación. La infraestructura puede ser adaptada según las necesidades y alcance de la solución, lo que la hace altamente flexible

El patrón arquitectónico API Gateway es una solución efectiva para acceder a los microservicios publicados en servicios Lambda de AWS. Al implementar este patrón, se obtiene una capa de abstracción que permite una mayor flexibilidad y escalabilidad en el diseño de la arquitectura de microservicios.

En conjunto con los servicios Lambda de AWS, el patrón API Gateway permite la implementación eficiente de microservicios y la gestión efectiva de las solicitudes de la API, lo que resulta en una arquitectura escalable y de alto rendimiento en AWS. Las implementaciones de funciones lambda son muy atractivas para las organizaciones que buscan soluciones basadas en microservicios a través de ambientes Cloud. Los servicios basados en lambda se encargan de ejecutar las funciones de forma independiente, lo que lo convierte en un entorno sin servidor.

El uso de JWT (Json Web Token) para la autenticación y autorización de usuarios en una API, proporciona una solución segura y escalable para proteger los recursos de la aplicación. JWT es fácilmente integrable con numerosas bibliotecas y

herramientas, lo que lo hace una opción popular para los desarrolladores.

La implementación de una arquitectura basada en microservicios para que sea exitosa y obtener los resultados esperados la persona encargada (arquitecto de software) debe hacer un análisis de ventajas y desafíos que existen y ser considerados antes de su implementación. Al abordar estas consideraciones, se puede garantizar una implementación exitosa de esta arquitectura y obtener los resultados esperados.

REFERENCIAS

[1] "Del dinero en efectivo al pago digital en pandemia." https://www.bancomundial.org/es/news/feature/2022/02/04/dinero-en-efectivo-pago-digital-pandemia-america-latina (accessed Apr. 19, 2023).

[2] J. Rubio, J. Jiménez, and D. Acosta, "Evolución de los medios de pago del Ecuador en el contexto de pandemia Covid-19." Accessed: Apr. 09, 2023. [Online]. Available: https://contenido.bce.fin.ec/documentos/Administracion/snp-estadistica-2.pdf

[3] L. Zambrano, "Las billeteras digitales crecen y amplian los servicios de pagos," Feb. 09, 2023. https://www.expreso.ec/actualidad/economia/billeteras-digitales-crecen-amplian-servicios-pagos-150003.html (accessed Apr. 20, 2023).

[4] V. René, E. Encarnación, S. Caridad, R. Quesada, O. Máximina, and E. Merchán, "Billetera electrónica móvil: una alternativa de pago del sistema financiero ecuatoriano," *Contabilidad y Negocios*, vol. 15, no. 30, pp. 24–42, Dec. 2020, doi: 10.18800/CONTABILIDAD.202002.002.

[5] "DSpace de Uniandes: Perspectivas de los taxistas de Santo Domingo sobre el uso del dinero electrónico como medio de pago." https://dspace.uniandes.edu.ec/handle/123456789/10589 (accessed Apr. 13, 2023).

[6] "Amazon API Gateway | API Management | Amazon Web Services." https://aws.amazon.com/es/api-gateway/ (accessed Apr. 14, 2023).

[7] "Microservices vs. service-oriented architecture – O'Reilly." https://www.oreilly.com/radar/microservices-vs-service-oriented-architecture/ (accessed Apr. 14, 2023).

[8] "What can RabbitMQ do for you? — RabbitMQ." https://www.rabbitmq.com/features.html (accessed Apr. 14, 2023).

[9] S. Fix, "Michael T. Nygard-Release It!_ Design and Deploy Production-Ready Software-Pragmatic Bookshelf (2018)", Accessed: Apr. 16, 2023. [Online]. Available: https://pragprog.com/titles/mnee2/release-it-second-edition