Universidad Nacional de Loja

Facultad de la Energía, las Industrias y los de Recursos Naturales No Renovables

Maestría en Ingeniería en Software

Diseño de una arquitectura basada en microservicios para una Wallet Electrónica

Trabajo de Titulación previo a la obtención del título de Magíster en Ingeniería en Software

AUTOR:
Byron Giovanny Cholca Campués

DIRECTOR:
Ing. Wilman Patricio Chamba Zaragocín, Mg Sc.

Loja - Ecuador
2023

Certificación

Loja, 21 de abril de 2023

Ing. Wilman Patricio Chamba Zaragocín, Mg. Sc.
DIRECTOR DEL TRABAJO DE TITULACIÓN

CERTIFICO:

Que he revisado y orientado todo proceso de la elaboración del Trabajo de Titulación denominado: **Diseño de una arquitectura basada en microservicios para una Wallet Electrónica**, previo a la obtención del título de **Magíster en Ingeniería en Software**, de autoría del estudiante **Byron Giovanny Cholca Campués**, con **cédula de identidad Nro. 1715851406**, una vez que el trabajo cumple con todos los requisitos exigidos por la Universidad Nacional de Loja para el efecto, autorizo la presentación para la respectiva sustentación y defensa.

Firmado electrónicamente por:
WILMAN PATRICIO
CHAMBA ZARAGOCIN

Ing. Wilman Patricio Chamba Zaragocín, Mg.Sc.
DIRECTOR DEL TRABAJO DE TITULACIÓN

Autoría

Yo, **Byron Giovanny Cholca Campués**, declaro ser autor del Trabajo de Titulación y eximo expresamente a la Universidad Nacional de Loja y a sus representantes jurídicos de posibles reclamos y acciones legales, por el contenido del mismo. Adicionalmente acepto y autorizo a la Universidad Nacional de Loja la publicación del Trabajo de Titulación en el Repositorio Digital Institucional – Biblioteca Virtual.

Firma: BYRON GIOVANNY CHOLCA CAMPUES Firmado digitalmente por BYRON GIOVANNY CHOLCA CAMPUES Fecha: 2023.05.03 21:21:55 -05'00'

Cédula de Identidad: 1715851406

Fecha: 03/05/2023

Correo electrónico: byron.cholca@unl.edu.ec

Teléfono: 099 4910 959

Carta de autorización por parte del autor, para consulta, reproducción parcial o total y/o publicación electrónica de texto completo, del Trabajo de Titulación

Yo, **Byron Giovanny Cholca Campués**, declaro ser autor del Trabajo de Titulación denominado: **Diseño de una arquitectura basada en microservicios para una Wallet Electrónica** como requisito para optar el título de **Magíster en Ingeniería en Software**, autorizo al sistema Bibliotecario de la Universidad Nacional de Loja para que con fines académicos muestre la producción intelectual de la Universidad, a través de la visibilidad de su contenido de la siguiente manera en el Repositorio Institucional.

Los usuarios pueden consultar el contenido de este trabajo en el Repositorio Institucional, en las redes de información del país y del exterior con las cuales tenga convenio la Universidad.

La Universidad Nacional de Loja, no se responsabiliza por el plagio o copia del Trabajo de Titulación que realice un tercero.

Para constancia de esta autorización, en la ciudad de Loja, a los tres días del mes de mayo de dos mil veintitrés.

Firma: BYRON GIOVANNY CHOLCA CAMPUES Firmado digitalmente por BYRON GIOVANNY CHOLCA CAMPUES Fecha: 2023.05.03 21:22:23 -05'00'

Autor: Byron Giovanny Cholca Campués
Cédula de identidad: 1715851406
Dirección: Quito, Comité del Pueblo, calle Máximo Gómez
Correo electrónico: byron.cholca@unl.edu.ec
Teléfono: 099 4910 959

DATOS COMPLEMENTARIOS:
Director del Trabajo de Titulación: Ing. Wilman Patricio Chamba Zaragocín Mg. Sc.

Dedicatoria

En primer lugar, dedico este trabajo a Dios, gracias a su fortaleza, provisión y sabiduría en toda la carrera de la maestría para el desarrollo de este gran logro, y que siempre me recuerda en su Palabra en Mateo 6:33 *Buscar primeramente el reino de Dios y su justicia y las demás cosas serán añadidas.*

A mi padre Julio y mi madre María Antonia ya que su esfuerzo y amor hacia mí, me han impulsado cada día a salir adelante, este logro universitario es para honrar y agradecer su esfuerzo como padres.

A mi amada novia Cesia, mi apoyo e inspiración para seguir adelante, cumpliendo nuestro propósito como pareja y futura familia para glorificar a Dios en todo. Este logro académico también es tuyo, te amo y sé que este Trabajo de Titulación es un reflejo de nuestro compromiso y apoyo mutuo en nuestra relación.

Byron Giovanny Cholca Campués

Agradecimiento

Doy gracias a Dios por su fidelidad, ya que de Él son todos mis logros.

A mis familiares y amigos, en especial a mis Padres y mi futura esposa, quienes han estado presente en todo este caminar académico profesional

A la Universidad Nacional de Loja por la oportunidad otorgada para mi crecimiento profesional, y al cuerpo de docentes, especialmente al Ing. Wilman Chamba Zaragocín por su compromiso y apoyo en el desarrollo del presente Trabajo de Titulación.

Byron Giovanny Cholca Campués

Índice de contenidos

Índice de tablas:

Índice de figuras:

Índice de anexos:

1. Título

Diseño de una arquitectura basada en microservicios para una Wallet Electrónica.

2. 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 Trabajo de Titulación 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 estudios primarios seleccionados para 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 expresados mediante diagramas de despliegue y de componentes que den 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 claves: *Wallet electrónica, arquitectura de software, microservicios, Cloud, AWS, Api Gateway, patrones arquitectónicos.*

Abstract

Electronic means of payment have been very well received by economic agents in Ecuador. The financial payment services (Fintech) companies PayPhone and PeiGo, published their new virtual wallets in March 2022 and the Google wallet began operating in Ecuador in February 2023. The present dissertation aims to propose a software architecture design for an e-Wallet. The theoretical approach on which the architecture design was based was the concept of microservices with architectural patterns that allow improving aspects such as scalability and security, and through proofs of concept to evaluate performance in a Cloud environment with an API Gateway architectural pattern. An incremental methodology was used to achieve the aims of this dissertation. The work was divided into 3 installments. The first was to perform a systematic literature review in scientific libraries, selecting articles, which give answers to the previously defined research questions, and from selected primary studies to obtain the theoretical knowledge in architectural patterns and architecture based on microservices such as API Gateway, Lambda Services, Database Per Service, Circuit Breaker, and Message Bus. The second step was to propose the software infrastructure of the architecture in a Cloud solution to be implemented in AWS, the software architecture was modeled and expressed through deployment and component diagrams that respond to the functional and non-functional requirements established. Finally, the third step was to evaluate the performance through proofs of concept and validate the operation of the API Gateway architectural pattern together with published APIs in a Cloud environment with AWS Lambda services.

Keywords*: e-Wallet, software architecture, microservices, Cloud, AWS, Api Gateway, architectural patterns.*

3. 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 Ecuador, la pandemia también ha llevado a un aumento en la confianza de los ciudadanos en los medios de pago electrónicos disponibles, principalmente debido a un crecimiento del 35% en el número de transacciones por pagos interbancarios realizados por los clientes del sistema financiero entre mayo y septiembre de 2020 [3].

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 [4].

La banca de Ecuador está impulsando un sistema financiero electrónico que representa un importante avance tecnológico. Para llevar a cabo esta innovación, es necesario que una aplicación de Billetera Electrónica permita realizar pagos de transferencias de forma sencilla a través de recargas electrónicas desde el celular.

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 [5] 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 [6].

Por lo expuesto el presente Trabajo de Titulación, se presentó con el fin de dar respuesta a la pregunta de investigación: "¿Una arquitectura basada en microservicios en la nube para una Wallet Electrónica permitirá mejorar los tiempos de respuesta de las transacciones y la integración con otros servicios?" Planteando el 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" y para conseguirlo se desarrollaron tres objetivos específicos: "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.", "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.", "Evaluar el rendimiento y validar el funcionamiento de la solución basada en microservicios en la Cloud mediante cargas de estrés."

En lo que respecta al desarrollo de los objetivos específicos, aplicando la técnica de Entrevista no Estructurada de preguntas y respuestas se identificó del estado arquitectónico de la empresa Quikly, y mediante la investigación y revisión sistemática de la literatura en librerías científicas, se seleccionaron artículos primarios que respondían a las preguntas de investigación. En el objetivo dos, mediante modelado con diagramas UML se propuso una arquitectura de microservicios detallando cada uno de sus componentes con patrones arquitectónicos de seguridad y tolerancia a fallos. En el objetivo tres se realizaron pruebas de concepto sobre un prototipo de arquitectura de Software en un ambiente Cloud AWS utilizando servicios Lambdas con el patrón API Gateway, sobre este prototipo se realizaron pruebas de estrés logrando determinar que la arquitectura propuesta cumple con los requisitos identificados.

En cuanto a la estructura de este informe, se compone de la siguiente manera: el **Marco teórico**, expone antecedentes conceptos con bases teorías para la comprensión del tema principal; la **Metodología** presenta el área de estudio y los procedimientos para el desarrollo del presente Trabajo de Titulación, en la sección de **Resultados** se detalla cada componentes que da solución al objetivo planteado; en la **Discusión** se analiza como en base a la teoría se llevó a cado los resultados, desde el punto de vista del autor y finalmente se detalla las **Conclusiones y Recomendaciones** donde se exponen los aspectos más relevantes del proyecto y se brindan sugerencias para su mejora.

4. Marco teórico

En esta sección se abordan conceptos preliminares con el fin de construir una base teórica para el desempeño del presente Trabajo de Titulación (TT).

El marco teórico, que se desarrolla a continuación, permitirá situar el TT dentro de un conjunto de conocimientos, además ofrecer una conceptualización adecuada de términos y conceptos necesarios para su entendimiento.

4.1 Antecedentes

4.1.1 Medios de pago

Los medios de pago son activos que utilizan los agentes económicos para cumplir con obligaciones derivadas de una transacción. Existen medios de pago físicos, como el dinero en efectivo y los cheques, y medios de pago electrónicos, como las transferencias por medios electrónicos o digitales, las tarjetas de crédito y débito y otros de similar naturaleza [7].

En la actualidad los medios de pago son provistos por las instituciones financieras y reguladas por los Bancos Centrales, su provisión eficaz y satisfactoria depende del grado de confianza que el usuario mantenga sobre el proveedor para ejecutar las transacciones correctamente, con la seguridad suficiente y en el plazo acordado [8].

4.1.2 Billetera electrónica.

El Banco Central del Ecuador, [9] define **Dinero Electrónico o Billetera Móvil** a la manera de presentar monedas virtuales en sustitución a los pagos comúnmente usados para cualquier arquetipo de transacciones comerciales o de servicios. Este procedimiento fue fundado desde septiembre del 2014.

En este contexto, Ecuador incluye el sistema de pago electrónico, nombrado Sistema de Dinero Electrónico por el Banco Central del Ecuador y conocido como Billetera Móvil o Billetera de Cooperativas por las entidades financieras privadas del país, según el Banco Central del Ecuador [9].

4.2 Arquitectura de Software

En el contexto de los sistemas informáticos, la arquitectura de software se puede definir como el diseño de alto nivel de un conjunto de estructuras que incluyen elementos de software, sus relaciones y propiedades, con el fin de satisfacer los requisitos del negocio. Esta arquitectura proporciona un marco de referencia para guiar la construcción de sistemas informáticos mediante el uso de patrones y abstracciones. Al seguir esta arquitectura, programadores, analistas y otros ingenieros pueden compartir una línea de trabajo común. Dentro de los conceptos de arquitectura de software se puede recalcar, los estilos y los patrones arquitectónicos, que se definen como una colección de decisiones de diseño arquitectónico aplicable a un contexto de desarrollo y a problemas de diseño recurrentes [10].

4.2.1 Microservicios

Existen varias definiciones y descripciones de microservicios encontradas en artículos y publicaciones en la web y cada una es propuestas por diferentes autores, organizaciones y comunidades de desarrollo de software. Algunas de estas definiciones pueden diferir en ciertos detalles, pero en general, todas se refieren a un enfoque arquitectónico en el que una aplicación se descompone en un conjunto de servicios independientes, altamente cohesivos y con una comunicación de bajo acoplamiento entre ellos. A continuación, se describe las más relevantes:

En su definición, Fowler y Lewis explican que un microservicio es una unidad de software que se enfoca en resolver un problema específico de negocio y que se puede desarrollar, desplegar y escalar de manera independiente. Los microservicios son un enfoque para construir sistemas de software que se basa en el desarrollo de pequeños servicios autónomos e independientes que trabajan juntos para proporcionar una funcionalidad completa del sistema [11].

La plataforma de Amazon Web Services proporciona diversos servicios en la nube, incluyendo una plataforma para los elementos clave de una arquitectura basada en Microservicios. En su sitio web oficial, Amazon Web Services explica la siguiente definición: “Los microservicios son un enfoque arquitectónico y organizativo para el desarrollo de software donde el software está compuesto por pequeños servicios independientes que se comunican a través de API bien definidas. Los propietarios de

estos servicios son equipos pequeños independientes. Las arquitecturas de microservicios hacen que las aplicaciones sean más fáciles de escalar y más rápidas de desarrollar. Esto permite la innovación y acelera el tiempo de comercialización de las nuevas características" [12].

De acuerdo con [13] que citó a [14] "la arquitectura de microservicios es un cambio hacia la transformación de servicios y/o componentes en microservicios: servicios más detallados y autónomos que aíslan funcionalidades comerciales detalladas por límites e interactúan a través de interfaces estandarizadas".

Los microservicios cooperan para proveer complejidad y agregar funcionalidades al sistema, esto resalta la importancia de que cada uno de ellos sea independiente y su función dentro del sistema se encuentre claramente definida. Sin estos dos aspectos no sería posible lograr la mantenibilidad del sistema mientras ´este evoluciona mediante la adición de nuevas características. Independencia, en este contexto, implica la posibilidad de establecer para cada servicio su propio entorno de ejecución ´on (arquitectura, plataforma, etc.), ciclo de desarrollo y que su despliegue y operación no dependa de los demás microservicios que componen el sistema [15].

Al ser la independencia, modularidad y cohesión las principales características de los microservicios, es importante que la plataforma e infraestructura que alberga el software sea flexible, mientras enfrenta los principales retos asociados a este tipo de arquitectura: la seguridad y el desempeño, heredados de SOA, debido al intercambio de información a través de la red [16].

4.2.2 Arquitectura basada en microservicios

Fowler y Lewis definen el estilo arquitectónico de microservicios como un en-foque para el desarrollo de una aplicación integrada por un conjunto de pequeños servicios, cada uno ejecutándose independientemente y comunicándose con mecanismos ligeros, siendo en la mayoría de las veces un API de recursos HTTP.

En una arquitectura de microservicios, cada microservicio es responsable de una función de negocio específica y se comunica con otros microservicios mediante interfaces bien definidas y abiertas [11]. Cada microservicio se desarrolla y se despliega de forma independiente, lo que permite una mayor flexibilidad y escalabilidad en la construcción

de aplicaciones. Además, los microservicios pueden ser escritos en diferentes lenguajes de programación y utilizar diferentes tecnologías, lo que les da una gran flexibilidad y los hace adecuados para aplicaciones grandes y complejas.

Provee cohesión en funciones de negocio mediante la división del sistema en pequeños servicios, facilitando el diseño de sistemas complejos [16].

Los sistemas con este tipo de arquitectura son más ágiles, resilientes, escalables a la vez que simplifica su implementación, mantenimiento y despliegue respecto a sistemas con arquitecturas monolíticas [17].

4.2.2.1 Características de los microservicios

Según Lewis y Fowler [11] la arquitectura basada en microservicios cuenta con las siguientes características:

- **Componentización a través de servicios:** Los servicios en una arquitectura de microservicios son componentes de software independientes, autónomos y altamente cohesivos, diseñados para ser reutilizables.
- **Organizado en torno a las capacidades comerciales:** En lugar de estar organizado en torno a la estructura tecnológica o de la organización, una arquitectura de microservicios se enfoca en organizar los servicios en torno a las capacidades de negocio o funcionalidades.
- **Productos no proyectos:** Los servicios de una arquitectura de microservicios se desarrollan y despliegan como productos de software en lugar de proyectos, lo que significa que son mantenidos y evolucionados a lo largo del tiempo en lugar de ser abandonados después de su implementación inicial.
- **Terminales inteligentes y canalizaciones tontas:** Los servicios de una arquitectura de microservicios se comunican a través de interfaces bien definidas y abiertas, utilizando terminales inteligentes que gestionan el flujo de datos y canalizaciones tontas que no procesan ni transforman los datos.
- **Gobernanza descentralizada:** Una arquitectura de microservicios se enfoca en la descentralización de la gobernanza, lo que significa que cada equipo o servicio es responsable de su propio conjunto de reglas y políticas.

- **Gestión de datos descentralizada:** Los datos de una arquitectura de microservicios se almacenan y se gestionan de manera descentralizada, lo que significa que cada servicio es responsable de su propio conjunto de datos.
- **Automatización de infraestructura:** Una arquitectura de microservicios se enfoca en la automatización de la infraestructura y la implementación, lo que significa que los servicios se pueden desplegar y actualizar rápidamente.
- **Diseño para el fracaso:** Los servicios de una arquitectura de microservicios están diseñados para fallar y recuperarse rápidamente de posibles errores o interrupciones.
- **Diseño evolutivo:** Una arquitectura de microservicios está diseñada para ser evolutiva y adaptable a medida que cambian las necesidades de negocio y tecnológicas.

4.2.2.2 Ventajas de los microservicios

- Equipo de trabajo mínimo.
- Escalabilidad.
- Funcionalidad modular, módulos independientes.
- Libertad del desarrollador de desarrollar y desplegar servicios de forma independiente.
- Uso de contenedores permitiendo el despliegue y el desarrollo de la aplicación rápidamente

4.2.2.3 Desventajas de los microservicios

- Alto consumo de memoria
- Necesidad de tiempo para poder fragmentar distintos microservicios
- Complejidad de gestión de un gran número de servicios
- Necesidad de desarrolladores para la solución de problemas como latencia en la red o balanceo de cargas.
- Pruebas o testeos complicados al despliegue distribuido.

4.2.3 Arquitecturas Monolítica

En la arquitectura de microservicios, la aplicación monolítica se descompone en múltiples servicios pequeños, granulares, aislados, independientes y distribuibles [18].

El hecho de que estos servicios se desacoplan por separado es trascendental, pues permite priorizar los recursos escasos a los microservicios más relevantes, sobre los demás [19].

4.2.4 Arquitectura monolítica en comparación con la arquitectura de microservicios

Con las arquitecturas monolíticas, todos los procesos están estrechamente asociados y se ejecutan como un solo servicio. Esto significa que, si un proceso de una aplicación experimenta un pico de demanda, se debe escalar toda la arquitectura. Agregar o mejorar las características de una aplicación monolítica se vuelve más complejo a medida que crece la base de código. Esta complejidad limita la experimentación y dificulta la implementación de nuevas ideas. Las arquitecturas monolíticas aumentan el riesgo de la disponibilidad de la aplicación porque muchos procesos dependientes y estrechamente vinculados aumentan el impacto del error de un proceso.

Con una arquitectura de microservicios, una aplicación se crea con componentes independientes que ejecutan cada proceso de la aplicación como un servicio. Estos servicios se comunican a través de una interfaz bien definida mediante API ligeras, como se observa en la Fig. 1, los servicios se crean para las capacidades empresariales y cada servicio desempeña una sola función. Debido a que se ejecutan de forma independiente, cada servicio se puede actualizar, implementar y escalar para satisfacer la demanda de funciones específicas de una aplicación.

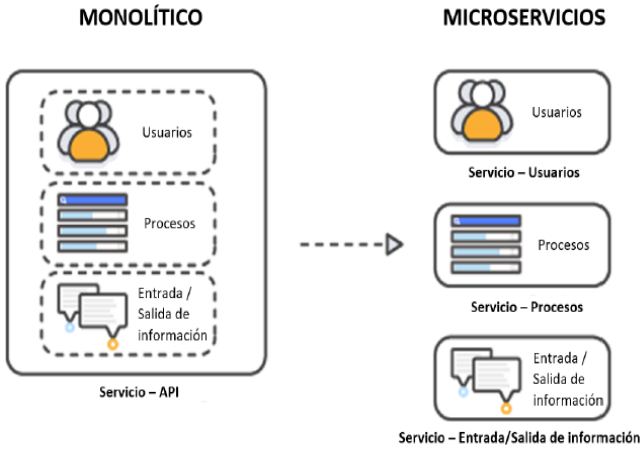


Fig. 1. Dividir una aplicación monolítica en microservicios [20].

4.3 Patrones Arquitectónicos

Es un conjunto de decisiones de diseño arquitectónico específicas que es aplicable a un problema de diseño recurrente, y parametrizado para considerar distintos contextos de desarrollo software en los que se presenta el mismo problema [10].

Por consiguiente, un patrón de arquitectura proporciona soluciones de diseño reutilizables y efectivas para un problema específico en un contexto validado, verificado y aceptado por la comunidad. Estas soluciones describen la relación entre los elementos y pueden ser clasificadas según el tipo de elementos arquitectónicos que utilizan. En los siguientes subapartados, se presentan las definiciones de varios patrones de diseño.

Las arquitecturas de microservicios están compuestas por múltiples elementos que desempeñan funciones específicas como, por ejemplo: aumentar la cohesión, reducción del acoplamiento y aplazamiento del tiempo de enlace. Por esta razón, es crucial administrar la comunicación con los servicios de manera efectiva para no comprometer el rendimiento de la aplicación, estos patrones pueden ayudar a mejorar la escalabilidad, la tolerancia a fallos y la eficiencia del sistema.

Afortunadamente, existen diversos modelos arquitectónicos probados que permiten aprovechar al máximo estas funciones.

4.3.1 API Gateway

Un servidor que actúa como el único punto de entrada en el sistema. Encapsula la arquitectura interna del sistema y proporciona una API que se adapta a cada cliente. Puede tener otras responsabilidades, como autenticación, monitoreo, balanceo de carga, almacenamiento en caché, configuración y gestión de solicitudes, y manejo de respuesta estática. Es responsable del enrutamiento de solicitudes, la composición y la traducción del protocolo. Todas las solicitudes de los clientes pasan primero por la API Gateway. Luego, enruta las solicitudes al microservicio apropiado. La API Gateway a menudo manejará una solicitud invocando múltiples microservicios y agregando los resultados.

4.3.2 Amazon API Gateway

Amazon API Gateway es un servicio completamente administrado que facilita a los desarrolladores la creación, la publicación, el mantenimiento, el monitoreo y la protección de API a cualquier escala. Las API actúan como la "puerta de entrada" para que las aplicaciones accedan a los datos, la lógica empresarial o la funcionalidad de sus servicios de backend. Con API Gateway, puede crear API RestFull y API WebSocket que permiten aplicaciones de comunicación bidireccional en tiempo real. API Gateway admite cargas de trabajo en contenedores y sin servidor, así como aplicaciones web [21].

4.3.3 Mensajería - RabbitMQ

Técnica que consiste en intercambiar mensajes relevantes entre los servicios mediante comandos o eventos de integración. Si se implementa correctamente, la mensajería logra un alto grado de desacoplamiento. Además, ofrece una flexibilidad total en cuanto a las opciones tecnológicas, siempre y cuando se puedan comunicar adecuadamente mediante el transporte utilizado.

RabbitMQ es un agente de mensajería, un intermediario para la mensajería. Les brinda a sus aplicaciones una plataforma común para enviar y recibir mensajes, y a sus mensajes un lugar seguro para vivir hasta que los reciba [22].

4.3.4 Patrón Circuit Breaker

El patrón Circuit Breaker, popularizado por Michael Nygard en su libro Release It!, [23] puede impedir que una aplicación intente repetidamente ejecutar una operación que tenga probabilidad de dar error. Ello le permite continuar sin esperar a corregir el error ni desperdiciar ciclos de CPU mientras se determina si el error continuará durante mucho tiempo. El patrón Circuit Breaker también permite a una aplicación detectar si el error se ha resuelto. Si el problema parece haberse corregido, la aplicación puede intentar invocar la operación [24].

Los estados del patrón son los siguientes:

- **Closed:** El circuito está cerrado y el flujo fluye ininterrumpidamente. Este es el estado inicial, todo funciona bien, la aplicación funciona de la manera esperada y la llamada al recurso/servicio se realiza de manera normal.

- **Open:** El circuito está abierto y el flujo interrumpido. En este estado todas las llamadas al recurso/servicio fallan inmediatamente, es decir no se realizan, devolviendo la última excepción conocida a la aplicación.
- **Half-Open:** El circuito está medio abierto (o medio cerrado) dando una oportunidad al flujo para su restauración. En este estado la aplicación volverá a intentar realizar la petición al servicio/recurso que fallaba.

4.3.5 Orquestación

Los microservicios tienen la capacidad de ejecutar sus propias funciones, pero cuando necesitan interactuar con otros servicios, requieren la orquestación de procesos. El término orquestación de servicios se refiere a la coordinación de múltiples servicios a través de un mediador centralizado, como un consumidor de servicios o un centro de integración, se llama orquestación debido a que al igual que una orquesta un número de músicos están tocando diferentes instrumentos y todos ellos son coordinados por el conductor central [25].

4.3.6 Base de datos por servicio

Este enfoque está basado en que cada microservicio tenga su propia base de datos privada, esto tiene como objetivo separar por completo los servicios. Otorga mayor escalabilidad, mecanismos de seguridad y despliegue independiente [26].

4.4 Microservicios y contenedores en entornos Cloud

La adopción de patrones de arquitectura que incrementen los beneficios de la computación cloud se hizo necesaria conforme esta última se posiciona como uno de los principales entornos para el desarrollo de software, debido a su versatilidad y flexibilidad en el despliegue y mantenimiento de las aplicaciones [17].

4.5 Cloud Computing

Los proveedores de servicios en la nube utilizan diferentes términos y conceptos para describir los servicios que ofrecen. A continuación, se describen algunos de los conceptos de computación en la nube más comunes según los proveedores Cloud:

- **Amazon Web Services (AWS):** La computación en la nube es un término genérico para todo aquello que implique proporcionar recursos de computación a través de Internet. Los usuarios pueden acceder a grandes cantidades de potencia de computación bajo demanda. Pueden adquirirla por minutos o por horas, y utilizar la cantidad que necesiten para sus actividades de computación [27].

- **Google Cloud Platform (GCP):** La computación en la nube es la disponibilidad a pedido de los recursos de procesamiento como los servicios por Internet. Elimina la necesidad de que las empresas obtengan, configuren o administren recursos por su cuenta; de esta forma, solo paguen por lo que usan [28].

- **IBM:** Computación en la nube es el acceso bajo demanda, a través de Internet, a recursos informáticos: aplicaciones, servidores (servidores físicos y Servidores virtuales), almacenamiento de datos, herramientas de desarrollo, capacidades de red y más, hospedadas de forma remota (centro de datos) gestionado por un servicio en la nube (proveedor CSP). El CSP ofrece estos recursos en un plan de suscripción mensual o los factura según el uso [29].

El Instituto Nacional de Estándares y Tecnología (NIST, por sus siglas en inglés) define modelos de implementación en la nube, los cuales especifican la ubicación de los servicios implementados y quién los administra. Estos modelos son los siguientes:

- **Nube pública:** Este modelo de servicio se define como una infraestructura de TI que se proporciona a través de internet a cualquier persona que lo necesite. Según el (NIST) [30], la nube pública se define como "una infraestructura de computación que se comparte entre varias organizaciones y que soporta un modelo de servicio de autoservicio bajo demanda para entregar recursos informáticos compartidos (por ejemplo, redes, servidores, almacenamiento, aplicaciones y servicios) que se pueden configurar y liberar con un mínimo esfuerzo de gestión o interacción con el proveedor del servicio en la nube".

- **Nube privada:** Este modelo de servicio se define como una infraestructura de TI que se proporciona y se utiliza exclusivamente por una única organización o empresa. Según el NIST [30], la nube privada se define como "una infraestructura de computación dedicada que es utilizada exclusivamente por una única organización y

que soporta un modelo de servicio de autoservicio bajo demanda para entregar recursos informáticos compartidos (por ejemplo, redes, servidores, almacenamiento, aplicaciones y servicios) que se pueden configurar y liberar con un mínimo esfuerzo de gestión o interacción con el proveedor del servicio en la nube".

- **Nube híbrida:** Este modelo de servicio combina la nube pública y la nube privada. Los recursos informáticos se pueden mover entre la nube pública y la privada según las necesidades de la organización. Según el NIST [30], la nube híbrida se define como "una infraestructura de computación que combina dos o más nubes (públicas, privadas o comunitarias) que permanecen como entidades distintas, pero se unen por tecnologías estandarizadas o propietarias que permiten la portabilidad de datos y aplicaciones".

4.6 Virtualización

La virtualización es un proceso que permite una utilización más eficiente del hardware físico de la computadora y es la base de la computación en la nube. El software de virtualización crea una capa de abstracción sobre el hardware de la computadora que permite que los elementos de hardware de una sola computadora (procesadores, memoria, almacenamiento y más) se dividan en múltiples computadoras virtuales, comúnmente llamadas máquinas virtuales (VM). Cada máquina virtual ejecuta su propio sistema operativo (SO) y se comporta como una computadora independiente [31].

4.6.1 Virtualización en la nube

Como se mencionó previamente, el modelo de computación en la nube se basa en la virtualización. Al virtualizar servidores, almacenamiento y otros recursos físicos del centro de datos, los proveedores de servicios en la nube pueden ofrecer una amplia gama de servicios a sus clientes, como los siguientes [31]:

- **Infraestructura como servicio (IaaS)**: recursos de servidor, almacenamiento y red virtualizados que puede configurar en función de sus requisitos.

- **Plataforma como servicio (PaaS)**: herramientas de desarrollo virtualizadas, bases de datos y otros servicios basados en la nube que puede utilizar para crear sus propias aplicaciones y soluciones basadas en la nube.

- **Software como servicio (SaaS):** aplicaciones de software que utiliza en la nube. SaaS es el servicio basado en la nube que más se abstrae del hardware.

4.6.2 Contenedores

Un contenedor es "Un sistema operativo liviano que se ejecuta dentro del sistema host, ejecuta instrucciones nativas de la CPU central, eliminando la necesidad de emulación a nivel de instrucción o compilación just in time. Los contenedores proporcionan ahorros en el consumo de recursos sin la sobrecarga de la virtualización, al tiempo que proporcionan aislamiento" [32].

En base a la definición anterior Los contenedores surgen como una alternativa ligera a las máquinas virtualizadas en el contexto de computación Cloud, pero orientados al despliegue de aplicaciones mediante el modelo PaaS (Platform as a Service), haciendo uso del concepto de "contenedores", es decir, la implementación de una aplicación o sus componentes en contenedores [33].

Los contenedores proporcionan a los sistemas dos características fundamentales para trabajar con componentes: portabilidad y modularización [34].

4.6.3 Computación sin servidor con la nube de AWS

La computación sin servidor hace referencia al desarrollo de aplicaciones con infraestructura de servidor subyacente, administrado externamente. Los servicios sin servidor, como AWS Lambda, ofrecen escalado automático, alta disponibilidad integrada y un modelo de facturación de pago en función del valor.

La computación sin servidor es una manera de describir los servicios, las prácticas y las estrategias que permiten a las compañías de desarrollo de software innovar y responder más rápido a los cambios. Los equipos pueden publicar aplicaciones con rapidez, obtener comentarios y mejorar su software mediante la eliminación de las cargas de trabajo operativas [27].

4.6.4 AWS lambda

AWS Lambda es un servicio informático que permite ejecutar código sin aprovisionar ni administrar servidores. Lambda ejecuta el código en una infraestructura de computación de alta disponibilidad y realiza todas las tareas de administración de los recursos de

computación, incluido el mantenimiento del servidor y del sistema operativo, el aprovisionamiento de capacidad y el escalado automático, así como las funciones de registro. Con Lambda, puede ejecutar código para prácticamente cualquier tipo de aplicación o servicio de backend. Lo único que tiene que hacer es suministrar el código en uno de los lenguajes que admite Lambda [35].

Lambda ejecuta la función solo cuando es necesario y escala de manera automática, desde unas pocas solicitudes por día hasta miles por segundo. Lambda ejecuta instancias de la función para procesar eventos. Puede invocar una función directamente mediante la API de Lambda o configurar un servicio o recurso de AWS para invocarla.

- **Función:** Una función es un recurso que se puede invocar para ejecutar el código en Lambda. Una función tiene código para procesar los eventos que pasa a la función o que otros AWS servicios envían a la función.

- **Evento** Un evento es un documento con formato JSON como se indica en la Fig. 2 que contiene datos para que una función de Lambda los procese. El tiempo de ejecución convierte el evento en un objeto y lo pasa al código de la función. Cuando se invoca una función, se determina la estructura y el contenido del evento. Puede invocar funciones de Lambda directamente con un punto de conexión HTTP(S) para la URL de función.

```
{
  "nombre":"Giovanny Cholca",
  "profesion":"Desarrollador",
  "edad":37,
  "lenguajes":["Java","Javascript","CSharp"],
  "disponibilidadParaViajar":true,
  "rangoProfesional": {
      "aniosDeExperiencia": 10,
      "nivel": "Senior"
  }
}
```

Fig. 2. Ejemplo evento: Datos Profesionales.

4.6.5 Docker

Es una herramienta para la gestión de imágenes (Docker File) para desplegar aplicaciones en contenedores, permite la creación de microservicios utilizando la virtualización para segmentar los recursos de una sola maquina física.

Docker es una plataforma abierta para desarrollar, enviar y ejecutar aplicaciones. Docker le permite separar sus aplicaciones de su infraestructura para que pueda entregar software rápidamente. Brinda la capacidad de empaquetar y ejecutar una aplicación en un entorno poco aislado llamado contenedor. El aislamiento y la seguridad le permiten ejecutar muchos contenedores simultáneamente en un host determinado. Los contenedores son livianos y contienen todo lo necesario para ejecutar la aplicación, por lo que no necesita depender de lo que está instalado actualmente en el host [36].

4.7 DevOps

DevOps (Desarrollo y Operaciones), es un enfoque basado en principios ágiles en que los stakeholders y los departamentos de desarrollo, operaciones y control de calidad colaboran para entregar de manera continua, lo que permite a la empresa aprovechar rápidamente las oportunidades del mercado y reducir la carga de trabajo. Para entender mejor el significado DevOps primero es necesario comprender los roles de Dev(desarrolladores) y de Ops (operaciones). Operadores (Ops) está compuesto en parte por los sysadmins (administradores de sistemas), los cuales tienen la misión del mantenimiento de los sistemas y su normal funcionamiento, hacen los despliegues y los rollbacks de las versiones de aplicaciones. Mientras que desarrollo Dev, es su responsabilidad mantener integro el entorno de producción. Los sysadmins tienen que ejecutar las aplicaciones, supervisar la operación, funcionamiento, evaluar y proponer mejoras para mantener las aplicaciones con rapidez y disponibilidad [37].

4.8 Microservicios y DevOps

Es importante destacar que la arquitectura de microservicios no se trata solamente de diseñar servicios más pequeños. Si se simplifica demasiado de esta manera, se puede terminar con una arquitectura más compleja, más lenta y de menor calidad. Para evitar este error, es necesario diseñar servicios autónomos, simples y con un mínimo acoplamiento y alta cohesión con el resto de los servicios.

Uno de los objetivos del uso de microservicios es ofrecer mayor agilidad, calidad y eficiencia en la entrega y despliegue continuo de software, lo que contribuye a mejorar el time-to-market, un requisito cada vez más exigido por los departamentos de negocios de las empresas. Debido a esto, los microservicios desempeñan un papel clave en la implementación de la filosofía DevOps.

5. Metodología

En este capítulo se expondrán los pasos, procedimientos, metodologías y técnicas empleadas para abordar el problema planteado previamente, en función de sus características, y brindar una solución efectiva.

El presente Trabajo de Titulación (TT) pertenece al tipo de Investigación Proyectiva ya que tiene como objetivo: Diseñar o crear propuestas dirigidas a resolver determinadas situaciones y siempre que estén sustentados en un proceso de investigación [38] como es el caso para la propuesta de un diseño arquitectónica basado en microservicios.

Para la realización del presente Trabajo de Titulación (TT) se utiliza la metodología de desarrollo de software Incremental, la cual es un modelo de ciclo de vida del software en el que se construye y se entrega una versión del software en diferentes incrementos, cada uno con funcionalidades y características adicionales. De manera que el software se va construyendo de forma progresiva, con cada incremento construido sobre la base del anterior [37]. Usando la metodología Incremental cada incremento corresponde a los 3 objetivos aprobados en el presente TT, como se describe en la Fig. 3

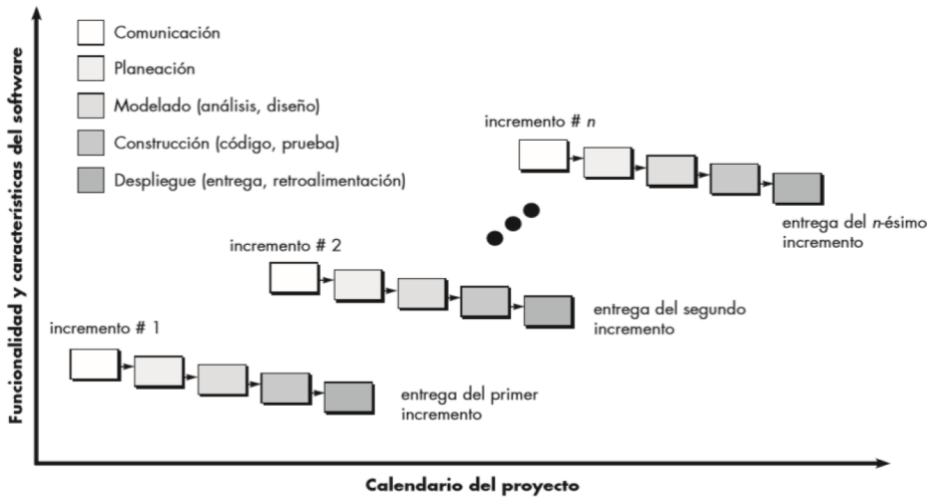


Fig. 3. Modelo incremental

5.1 Área de estudio

El presente Trabajo de Titulación (TT) se llevó a cabo para la empresa Quikly SAS, la cual es un ecosistema e-commerce y Fintech, en la cual desarrollan tecnología acorde a las necesidades de las PYMES y pequeños negocios, están enfocados en democratizar el comercio electrónico en Latinoamérica. Quikly SAS se encuentra ubicado en la cuidad de Quito en la dirección: La Pinta E4-432 y Amazonas, y cuyo representante legal es el Ing. Byron Rodríguez.

En este contexto se realizó el acercamiento al representante legal y se planteó el presente TT, cuyo objetivo es proponer "el diseño de una arquitectura basada en microservicios para una Wallet Electrónica", donde se formuló tres objetivos específicos los mismos que se detallan en la siguiente sección.

5.2 Procedimiento

El proceso de la investigación se detalla a continuación, mencionando cada uno de los objetivos con sus actividades:

5.2.1 Objetivo 1: 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.

- Se identificó el estado actual de la empresa en el uso las tecnologías y arquitectura de software, aplicando la técnica de Entrevista no Estructura de preguntas y respuestas al equipo de desarrollo de Quikly.
- 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.
- Se identificaron los requerimientos funcionales y no funcionales.
- Se definió los actores y roles junto con los casos de uso y las interacciones entre el usuario y el aplicativo.

5.2.2 Objetivo 2: 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.

- Se investigó los documentos primarios y se analizaron teóricamente patrones de diseño y componentes específicos para la arquitectura en microservicios, y se aplicó una técnica para los requerimientos no funcionales de seguridad.
- Se diseñó un diagrama de componentes detallando la arquitectura propuesta basada en microservicios para la Wallet Electrónica. Este diagrama de componentes es una representación visual de la arquitectura de la aplicación, y permitió una comprensión clara de cómo los diferentes microservicios interactúan entre sí, y se cumplió con los requerimientos de manera eficiente y escalable.

5.2.3 Objetivo 3: Evaluar el rendimiento y validar el funcionamiento de la solución basada en microservicios en la Cloud mediante cargas de estrés.

La finalidad de este objetivo tres fue realizar una Prueba de Concepto de Comunicación, consumiendo las Api Rest de algunos microservicios, y logró demostrar el funcionamiento en un entorno de infraestructura con AWS Cloud, para lo cual se estableció lo siguiente:

- Se Definieron los casos de pruebas que permitieron validar las mensajes y tiempos de respuestas. Los microservicios que se probaron son los siguientes:

  Autenticación: El microservicio realizó la autenticación y autorización de manera adecuada, se utilizó el protocolo JWT, y que las respuestas y errores del microservicio fueron correctos y acordes a lo esperado.

  Usuario: Con la herramienta Postman se consumió el Api para consultar información del usuario autenticado en la Wallet y se validó el servicio para obtener del Saldo de la cuenta del usuario.

  Catálogos: Se generaron las consultas de Ciudades, Provincias, Estado de Transacción, por medio de las APIs y se obtuvo la información requerida en formato Json.

- Mediante Postman se validaron los casos de prueba y se logró consumir las APIs, se aseguró que el microservicio funcione correctamente y cumplió con los requisitos definidos, se analizó el tiempo y códigos de respuesta de los servicios en las cargas de estrés.

- Se validó y documentó los resultados que dieron las pruebas, se logró evaluar los resultados de la prueba de concepto, y se documentó la estructura del Api Gateway y los resultados de las pruebas.

6. Resultados

En esta sección se describen las actividades llevadas a cabo durante el proceso de investigación y los resultados obtenidos en cada una de las fases definidas anteriormente en el apartado de procedimiento metodológico.

Aplicando la metodología Incremental basada en cumplimiento de objetivos; en el primer objetivo se obtuvo el estado arquitectónico actual de la empresa Quicky las definiciones sobre 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. En el segundo objetivo se diseñará la propuesta de arquitectura basada en microservicios y finalmente en el tercer objetivo mediante pruebas de concepto se evaluará el rendimiento de Apis seleccionadas en un ambiente Cloud.

6.1 Objetivo 1: 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 consistió 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, seguidamente con una revisión sistemática, se identificó las tecnologías predominantes en la arquitectura de microservicios, y patrones que en ella se utilizan, y finalmente; se obtuvo el modelo de requerimientos.

6.1.1 Estado y arquitectura actual de la empresa Quicky.

Al aplicar la técnica de Entrevista No Estructura vía online al desarrollador encargado de Quikly, la recolección de los datos es flexible ya que se determina más a una conversación entre dos personas (el entrevistador y entrevistado).

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, a través de la definición de requerimientos. En el **Anexo A** se detalla el guion utilizado en la entrevista junto con la evidencia en la herramienta Slack

El personal técnico labora en modalidad de teletrabajo, para los desarrollos usan metodología ágil Scrum, apoyándose de la herramienta Slack para compartir recursos, mensajes y tener reuniones online.

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.

6.1.2 Bases teóricas y tecnologías predominantes utilizadas en el diseño de microservicios.

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, el protocolo aplicado es el siguiente:

6.1.2.1 Definición de las cadenas de búsqueda

- **Término principal:**
  - software architecture base on microservices / arquitectura software basada en microservicios
- **Término alterno:** Software Architecture / Patrones arquitectónicos de software, estilos arquitectónicos.

6.1.2.2 Sitios de búsqueda

Las librerías digitales indexadas que se usaron para este estudio son:

- Google Académico: https://scholar.google.es/
- IEEEXplore: https://ieeexplore.ieee.org/
- Dialnet, de la Universidad de La Rioja: https://dialnet.unirioja.es/
- ACM Digital Library: https://dl.acm.org/

6.1.2.3 Criterios de exclusión

- Artículos que el título no tenga relación con el objeto de investigación.
- Artículos de contenido similar, aceptando solo artículos de contenido más completo.
- Artículos que solo mencionen microservicios sin considerar aspectos arquitectónicos.
- Artículos publicados antes del año 2015

6.1.2.4 Criterios de inclusión

- Artículos que el objeto de estudio trate de manera explícita el estilo arquitectónico basado y orientado a microservicios.
- Investigaciones que abarquen los microservicios desde el punto de vista arquitectónico.
- Material investigativo que ha sido publicado en español e inglés.

6.1.2.5 Selección de artículos

Paso 1: Se realizó la respectiva búsqueda de los artículos con la cadena de texto en las librerías digitales ya indicadas. En el **Anexo 2** se evidencia los resultados obtenidos en cada una.

Paso 2: De la lista de artículos descargados se aplicó los criterios de exclusión e inclusión ya indicados.

Paso 3: Del total de artículos incluidos y omitiendo los repetidos, se procede a la investigación, estudio y análisis sobre las tecnologías predominantes utilizadas en la construcción de microservicios. En el **Anexo 3** se describen los Criterios de investigación y motivación que se emplearon en el estudio de los artículos seleccionados.

Cade recalcar que los criterios de inclusión/exclusión se consideró el estado del arte de las publicaciones de los últimos 7 años, a excepción de los conceptos y definiciones sobre arquitectura de software considerando que, a pesar de la evolución constante de las tecnologías y herramientas, aún existen principios que siguen siendo relevantes en la actualidad, la TABLA I se detalla el número de artículos que se filtraron en uno de los tres pasos mencionados.

TABLA I
NÚMERO DE ARTÍCULOS ENCONTRADOS, EXCLUIDOS E INCLUIDOS

Librería digital	Artículos encontrados Paso 1	Paso 2	Paso 3
Google Académico	35	10	8
IEEEXplore	15	6	3
Dialnet	10	5	4
ACM	25	10	5
Total	60	31	20

- **Documentos encontrados:** 60
- **Documentos eliminados:** 40
- **Documentos que se utilizaron:** 20

Se concluyó que las arquitecturas de microservicios se basan principalmente en los conceptos de sistemas modulares, ya que su objetivo es dividir problemas complejos en servicios funcionales independientes entre sí.

Con el fin de comparar los resultados de los diferentes artículos, se ha organizado la información recopilada en categorías temáticas comunes. En la TABLA II 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.

TABLA II
BASES TEÓRICAS

Proceso	Funcionalidad	Descripción
Despliegue	Contenedores	Los contenedores, como Docker, son esenciales para implementar y orquestar microservicios de manera eficiente.
	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 cada uno de los microservicios en una única maquina física.
	Múltiples servicios por host	En una sola maquina se implementan varios microservicios utilizando máquinas virtuales o contenedores.

Comunicación	Servicios web	Los servicios web RestFull son una forma común de implementar microservicios, por lo que es importante conocer las tecnologías que los habilitan, 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.
	Descubrimiento de servicios	El cliente o el servidor posee una lista dinámica en donde aparecerán los servicios que se encuentran suscritos, esta lista puede ser implementada en el FrontEnd o en el BackEnd
DevOps	Integración y entrega continuas	las prácticas de integración y entrega continuas son fundamentales para la implementación exitosa de microservicios. Herramientas como Jenkins y Travis CI son útiles 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. Algunas tecnologías populares incluyen MongoDB y Cassandra.
	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.
	Base de datos en clúster.	Consiste en tener una base de datos distribuida en múltiples nodos para garantizar el acceso a los datos.

Se utiliza la herramienta Mendeley para gestionar las referencias permitió el almacenamiento y organización de los estudios primarios ubicando en directorios los artículos excluidos e incluidos como se visualiza en el **Anexo 4.**

6.1.3 Identificación de los requerimientos de la aplicación.

Aplicando la técnica de entrevista no estructurada, que se realizó con los técnicos desarrolladores de la empresa y justo con el análisis de la revisión documental, se definieron los requerimientos funcionales (TABLA III) y no funcionales (TABLA IV), que debe cumplir la Wallet Electrónica, junto con los diagramas de casos de uso donde se describen gráficamente las interacciones del usuario con el aplicativo y otros servicios de terceros.

6.1.4 Requerimientos funcionales

TABLA III
REQUERIMIENTOS FUNCIONALES.

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 (Es una bóveda de información del detalle del registro de pago de la tarjeta) 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 (permite emitir y recibir pagos, gestionar pagos para Marketplace y emitir tarjetas locales a través de una API directa, una plataforma y un contrato)	Transacción
RF-06	Registrar de valores cobrados	Transacción
RF-07	Verificar de montos para registro de tarjetas	Transacción
RF-08	Generación de link de pagos mediante código QR.	Integración
RF-09	Búsqueda de usuarios en Marketplace para generar el pago transaccional.	Integración
RF-10	Obtener Token de la tarjeta registrada para obtener información transaccional del pago.	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 (aplica comisión) se suma al saldo del beneficiario.	Integración
RF-13	Verificación de información de tarjeta para recibir pagos y ventas.	Integración
RF-14	Integración con reconocimiento facial servicios del proveedor OnDato el cual solicita documentos de identidad, escaneo facial mediante Inteligencia Artificial para verificación de información y prueba de vida.	Integración
RF-15	Servicios para catálogos	Integración

6.1.5 Requerimientos no funcionales

TABLA IV
REQUERIMIENTOS NO FUNCIONALES

Código	Categoría	Descripción
RNF-01	Seguridad, Portabilidad, Compatibilidad	Las integraciones con terceros y/o proveedores debe ser transparente y seguro para el usuario.
RNF-02	Seguridad	Los servicios deben proteger la información del usuario y los fondos almacenados mediante medidas de seguridad sólidas, como encriptación y autenticación de usuario.
RNF-03	Seguridad	La Wallet debe ser segura y resistente a ataques externos, lo que implica que los microservicios deben ser diseñados con mecanismos de seguridad apropiados, como autenticación, autorización, cifrado y monitoreo de seguridad.
RNF-04	Seguridad	La información de tarjetas, datos personales debe ser segura y no expuesta a terceros.
RNF-05	Disponibilidad	La Wallet debe estar disponible en todo momento, lo que significa que los microservicios deben estar diseñados para ser tolerantes a fallos y ser capaces de recuperarse rápidamente en caso de que ocurra una falla en algún servicio.
RNF-06	Escalabilidad	La Wallet debe ser capaz de manejar grandes volúmenes de transacciones simultáneas sin sufrir una reducción en su rendimiento, lo que requiere que los microservicios sean

		diseñados de manera escalable y que puedan ser fácilmente replicados y distribuidos en diferentes servidores.
RNF-07	Fiabilidad	La Wallet debe ser fiable y no generar errores inesperados.
RNF-08	Accesibilidad	La Wallet debe ser fácil de usar y entender para el usuario promedio.
RNF-09	Compatibilidad	La Wallet debe ser compatible con varios dispositivos y sistemas operativos.
RNF-10	Interoperabilidad	debe ser interoperable con diferentes plataformas y servicios, lo que significa que los microservicios deben ser diseñados para ser compatibles con diferentes lenguajes de programación, protocolos de comunicación y tecnologías de integración.
RNF-12	Rendimiento	debe tener un rendimiento rápido y eficiente, lo que implica que los microservicios deben estar diseñados para manejar grandes volúmenes de datos y transacciones, y ser capaces de procesar solicitudes en un tiempo mínimo.
RNF-13	Facilidad de mantenimiento	La Wallet debe ser fácil de mantener y actualizar, lo que significa que los microservicios deben ser diseñados para ser independientes y desacoplados, y que cada uno de ellos pueda ser actualizado y mantenido de manera independiente sin afectar el resto del sistema.
RNF-14	Compatibilidad	Para el desarrollo de una aplicación, se debe asegurar que el framework sea compatible con el servidor web utilizado, y que cumpla con los estándares de desarrollo web establecidos por la industria, como HTML, CSS y JavaScript.
RNF-15	Adaptabilidad	El proceso de desarrollo debe ser usado con metodologías ágiles que sean lo suficientemente flexibles para adaptarse a cambios en los requisitos del software, en el equipo de desarrollo o en el entorno de trabajo.

6.1.6 Actores, roles y casos de uso que interactúan en la aplicación.

A través de los diagramas de casos uso se realizó el modelado de los requerimientos funcionales, resumiendo las interacciones que se pueden dar entre los distintos usuarios y el sistema.

TABLA V
ACTORES Y ROLES

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.

Después de identificar los actores y respectivo rol que cumplen en la aplicación en la TABLA V, se procedió a elaborar los casos de uso correspondientes.

Casos de uso para el proceso de Autenticación. En la Fig. 4, se presenta el diagrama de casos de uso del sistema propuesto correspondiente al proceso de Autenticación.

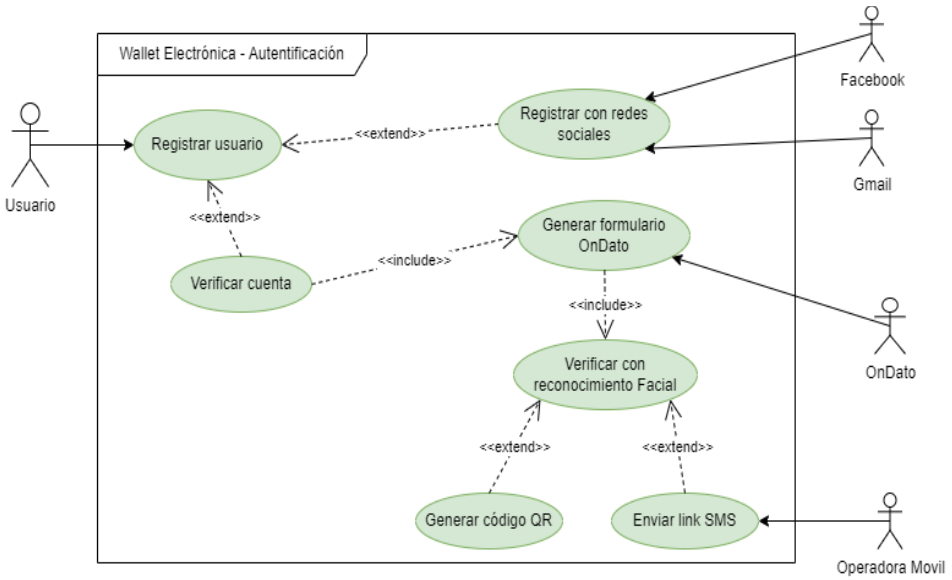


Fig. 4. Caso de uso Autenticación

Casos de uso para el proceso de Registro de tarjetas. En la Fig. 5, se presenta el diagrama de casos de uso del sistema propuesto correspondiente al proceso de Registro de tarjetas.

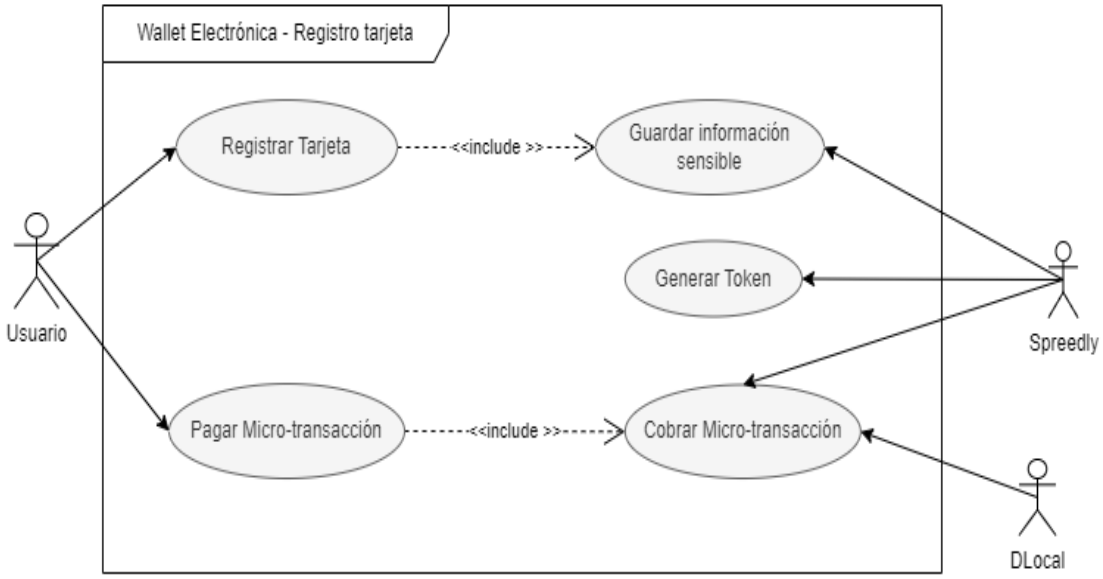


Fig. 5. Caso de uso Registro Tarjetas

Casos de uso para el proceso de Transacciones. En la Fig. 6, se presenta el diagrama de casos de uso del sistema propuesto correspondiente al proceso de Transacciones.

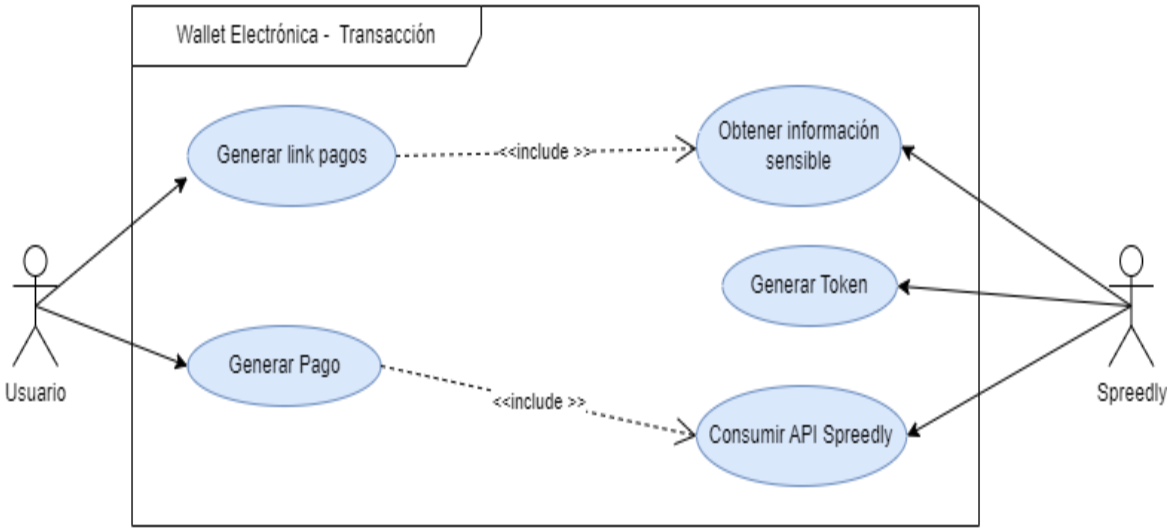


Fig. 6. Caso de uso Transacciones

Casos de uso para el proceso de Pagos. En la Fig. 7, se presenta el diagrama de casos de uso del sistema propuesto correspondiente al proceso de Pagos.

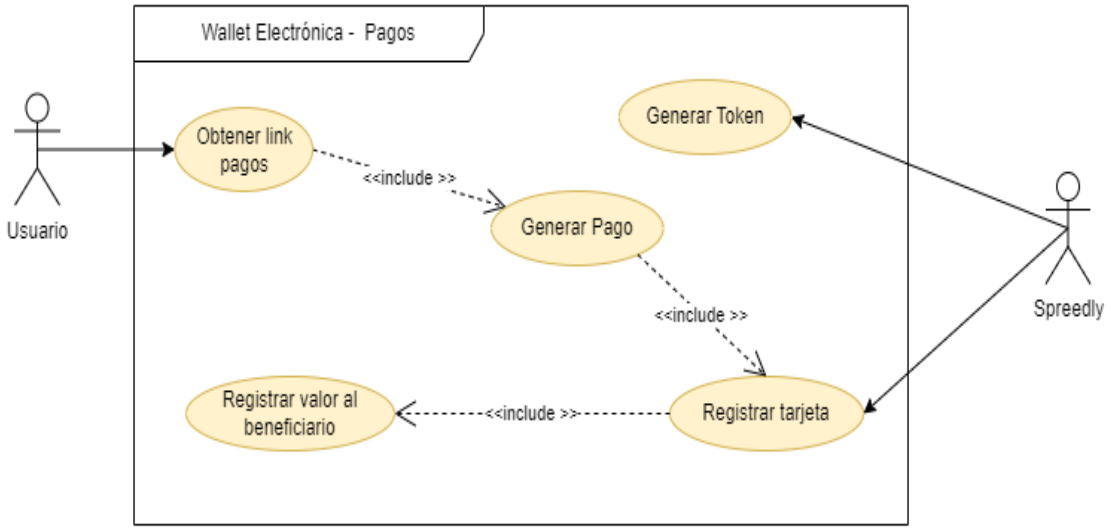


Fig. 7. Caso de uso Pagos

Casos de uso para el proceso de Cobros. En la Fig. 8, se presenta el diagrama de casos de uso del sistema propuesto correspondiente al proceso de Cobros.

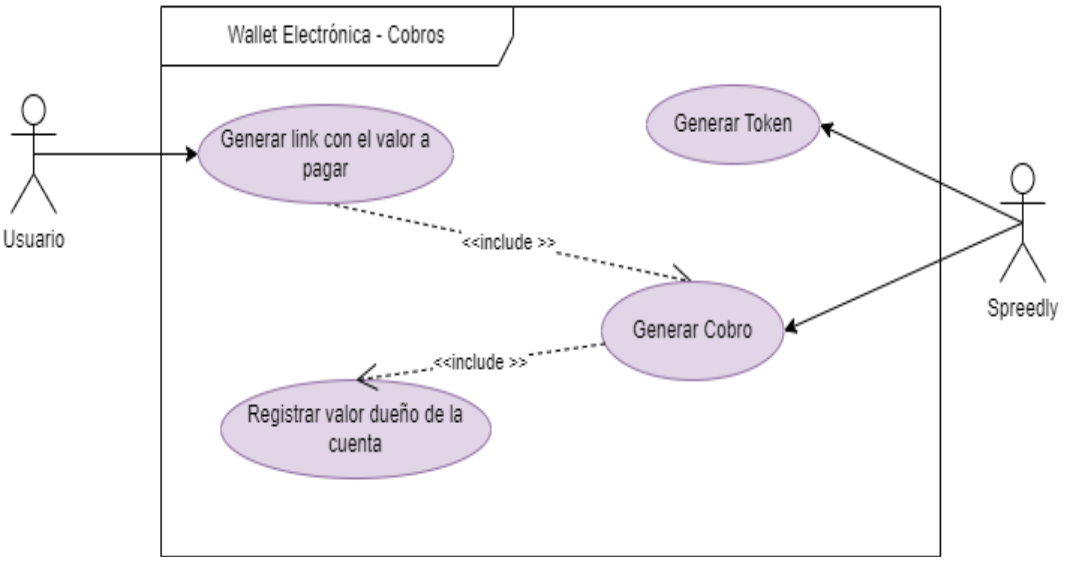


Fig. 8. Caso de uso Cobros

Casos de uso para la consulta de Catálogos. En la Fig. 9, se presenta el diagrama de casos de uso del sistema propuesto correspondiente al proceso de consulta de Catálogos.

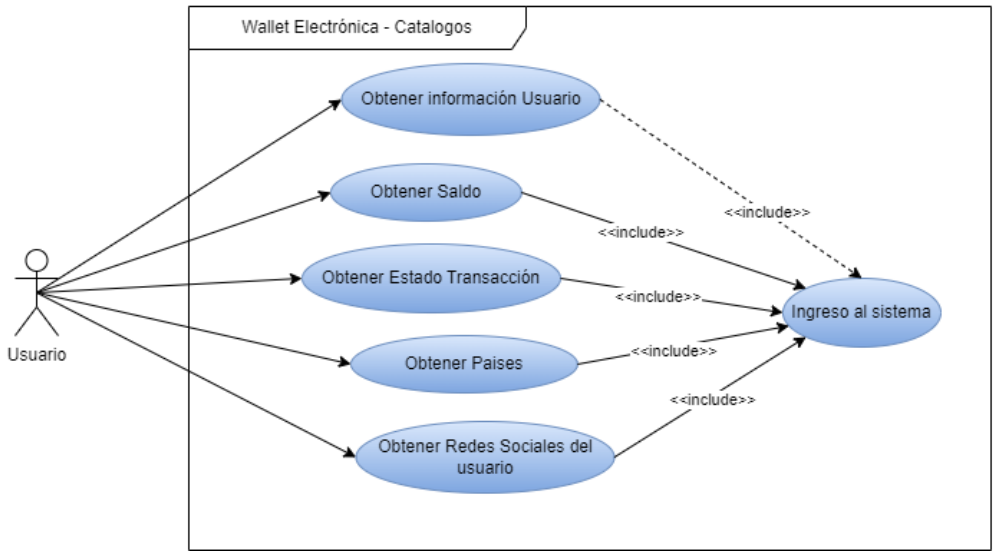


Fig. 9. Caso de uso Consulta Catálogos.

Mediante este proceso de investigación y las técnicas mencionadas se logró cumplir con el primer objetivo específico, el cual se refiere a llevar a cabo un estudio para identificar la tecnología, metodología y arquitectura que se emplea junto con los principios y patrones relevantes para el diseño de una arquitectura basada en microservicios.

Empleando la técnica de revisión documental para el primer objetivo se realiza un estudio a partir de la información recopilada sobre diseño de arquitecturas basadas en microservicios: conceptos, ventajas y desventajas, buenas prácticas, patrones arquitectónicos, metodologías, entre otros aspectos relevantes, que son relevantes al momento de diseñar la arquitectura propuesta.

La investigación fue fundamental para poder proponer un modelo de arquitectura que cumpliera con los requisitos establecidos, ya que proporcionó la base teórica necesaria. Como resultado, se obtuvo información relevante sobre los patrones de diseño más utilizados en este tipo de arquitecturas, así como las diferencias existentes entre ellos.

6.2 Objetivo 2: 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.

Una vez determinados los requerimientos que debe cumplir la aplicación, a continuación, en el incremento 2, se detallan los resultados obtenidos en las fases anteriores dentro del apartado del proceso metodológico.

Para identificar los microservicios que conforman la aplicación, se utilizó como punto de partida los Casos de Uso previamente presentados. Esto se debe a que los Casos de Uso describen los requisitos funcionales que la aplicación debe cumplir para satisfacer las necesidades del usuario.

A continuación, se indica el diagrama arquitectónico propuesto en base a los requerimientos funcionales y no funcionales, como se detalla el la Fig. 10

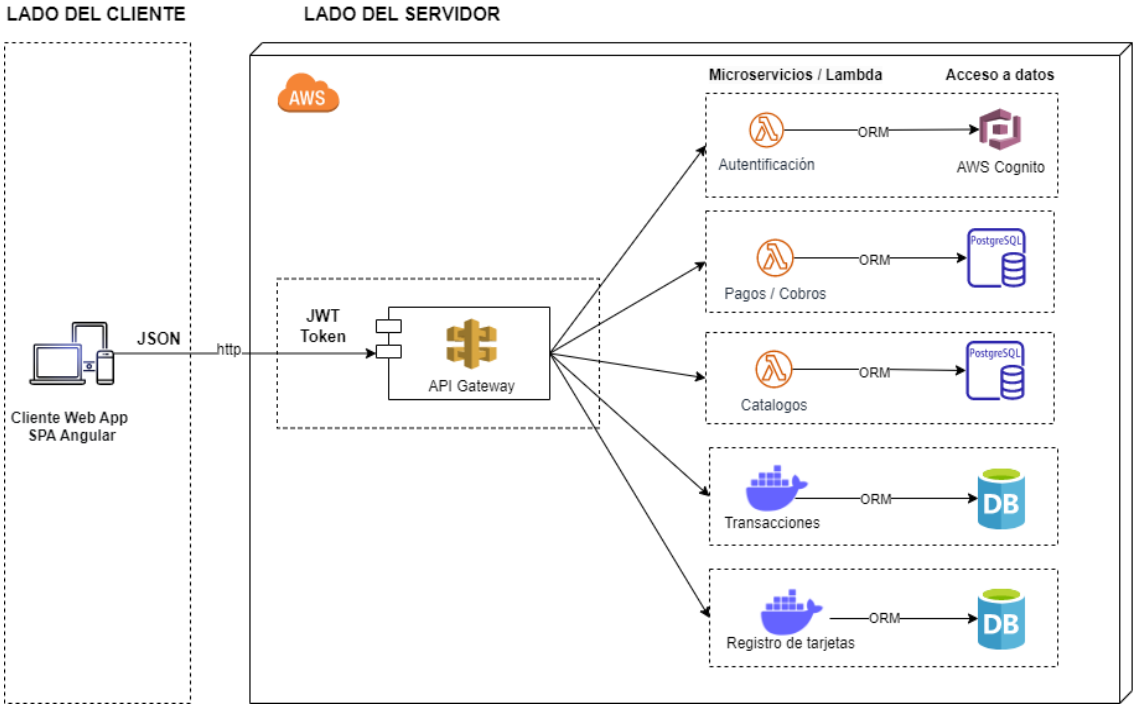


Fig. 10. Diagrama arquitectónico propuesto

En la Fig. 10 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: la capa de presentación, la capa de API Gateway, los servicios, la base de datos, los servicios de infraestructura, el sistema de mensajería. 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:

Lado del cliente:

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. Una vez que se ha autenticado al cliente, se genera un token de comunicación que se utiliza en cada una de las peticiones que se realizan desde el cliente hacia el microservicio. 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. El token de comunicación debe ser incluido en el Header de cada petición **Fig. 11**

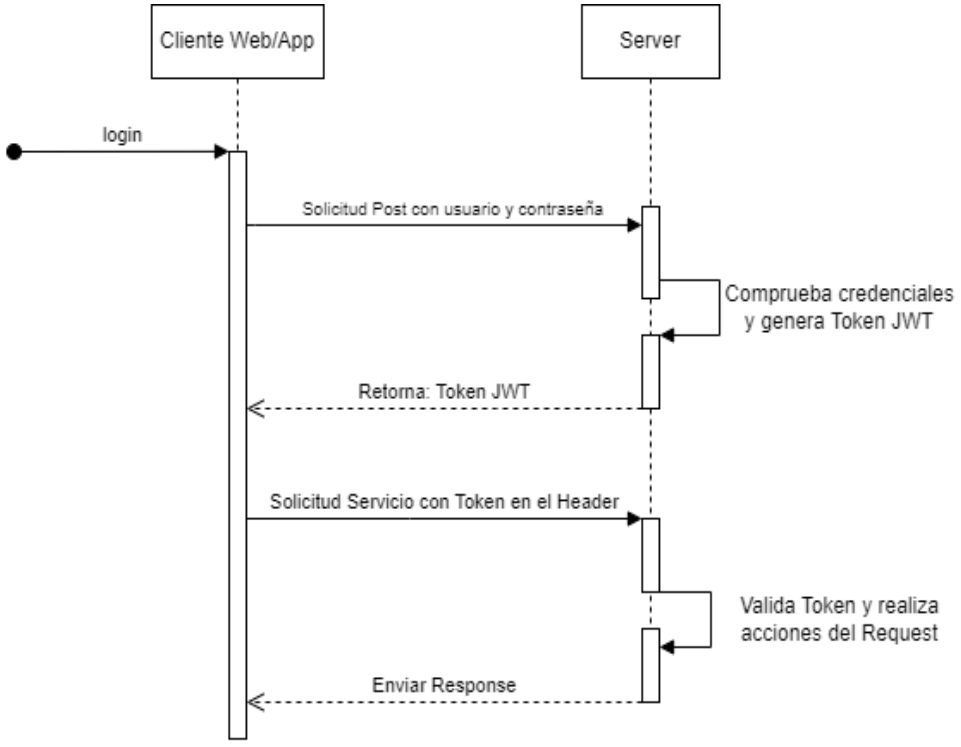


Fig. 11. Diagrama Secuencia del proceso de Autenticación con JWT

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. 12 se describe el proceso de este protocolo.

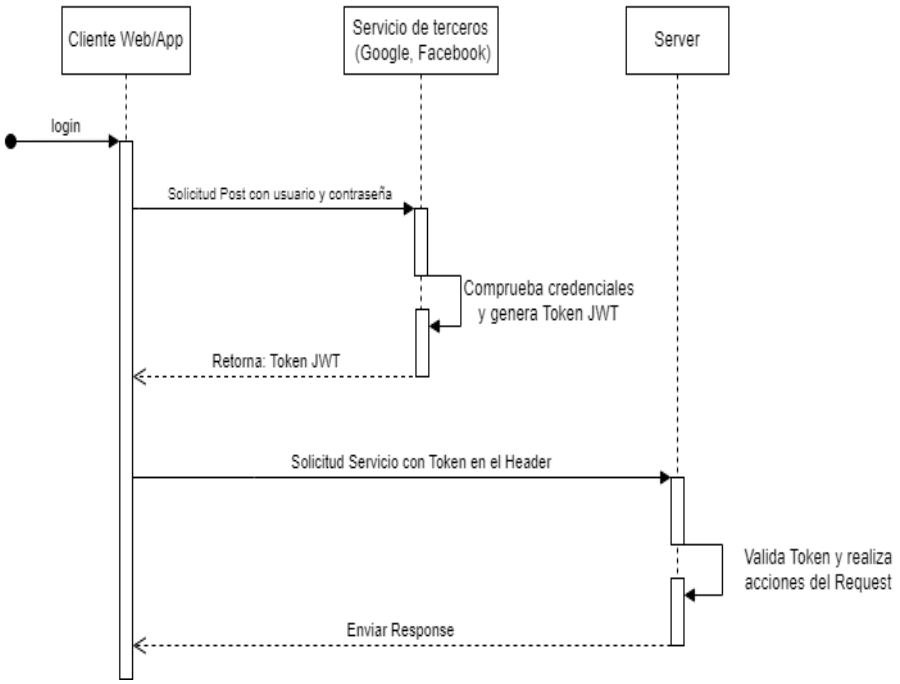


Fig. 12. Diagrama de Secuencia de Protocolo Auth2

Lado del servidor:

Capa de API Gateway: En este componente debe actuar como un punto de entrada para la aplicación enrutando las solicitudes de los clientes a los servicios apropiados. La elección de este patrón es resolver el problema de accesos a los microservicios y aporta otra ventaja como trabajar como un proxy.

En la Fig. 13 se muestra el funcionamiento del API Gateway en la arquitectura del Wallet, donde 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.

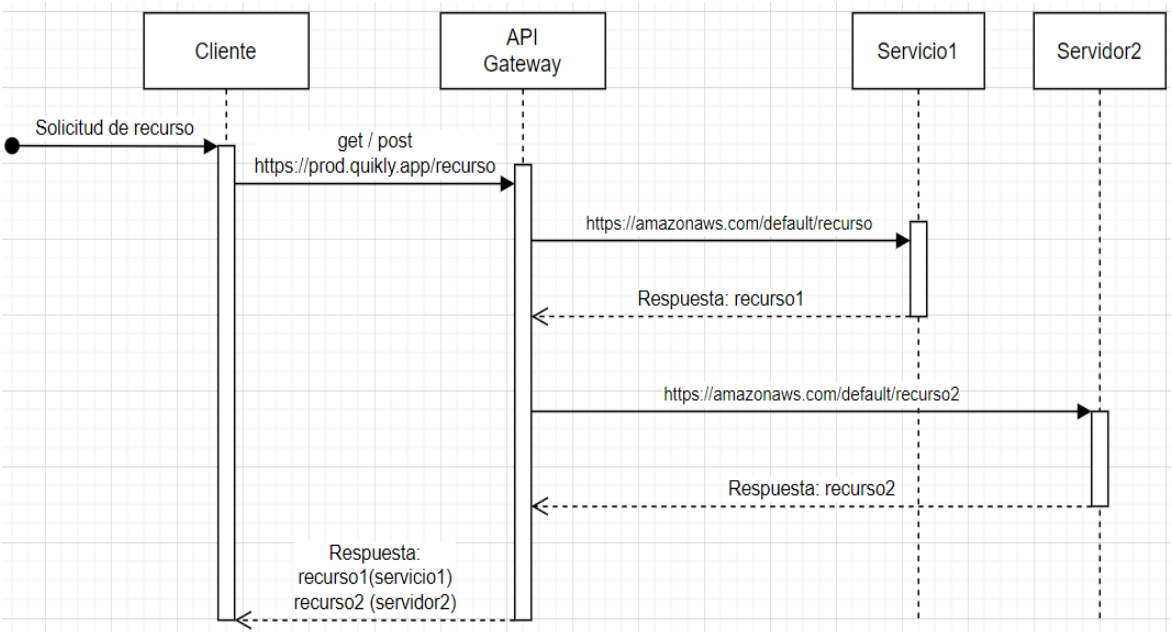


Fig. 13. Arquitectura Api Gateway - Solicitud de servicios.

Autenticación: Este componente 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. 14 de detalle el diagrama de arquitectónico y el proceso de autorización JWT a la Wallet Electrónica:

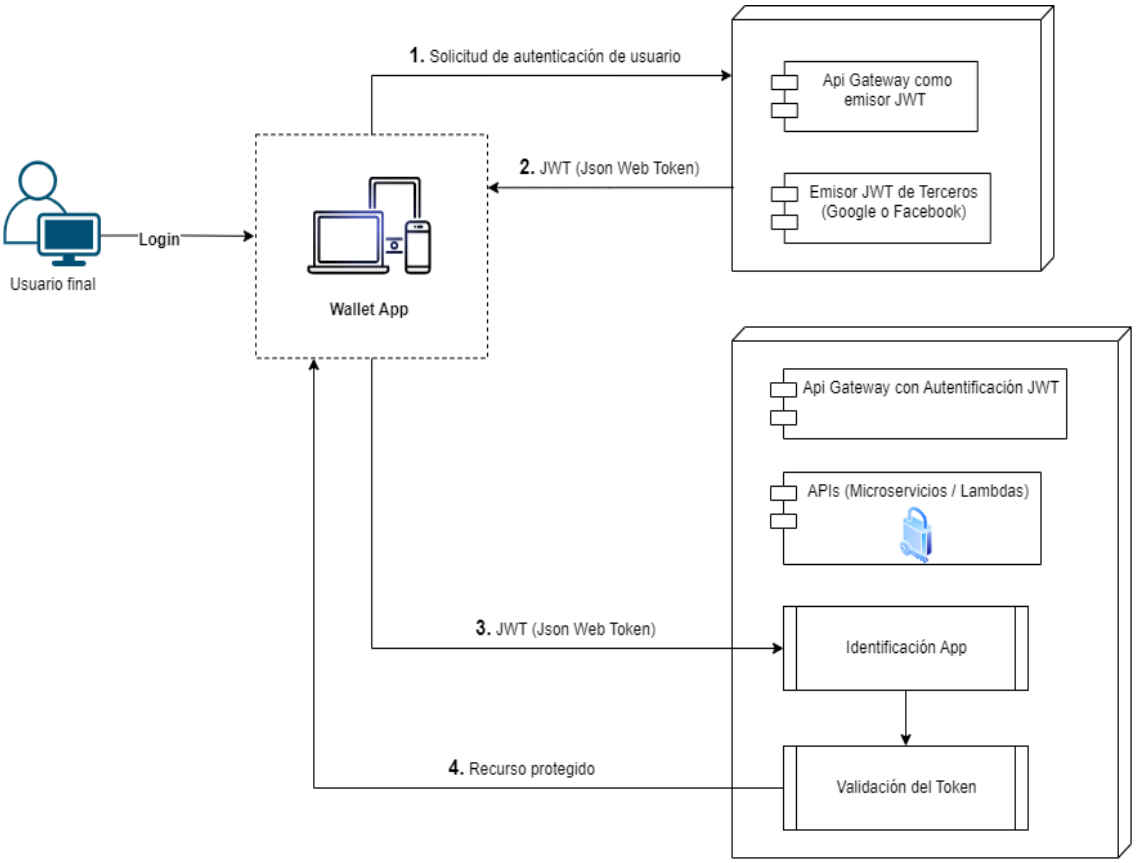


Fig. 14. Flujo de trabajo de autorización de JWT

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. 15, La respuesta incluye el accessToken, refreshToken, idToken. La aplicación puede usar el accessToken Se utiliza para autorizar el acceso a las APIs. El idToken contendrá la información necesaria del usuario para que el servidor pueda identificarlo, así como información adicional que pueda resultar útil (por ejemplo, roles o permisos). Además, el token tendrá una fecha de vencimiento, lo que significa que una vez que el tiempo de validez haya expirado, el servidor no permitirá que se acceda a los recursos mediante el mismo token. En este caso, el usuario tendrá que obtener un nuevo token de acceso ya sea volviéndose a autenticar o utilizando algún método adicional, como un token de actualización (refreshToken).

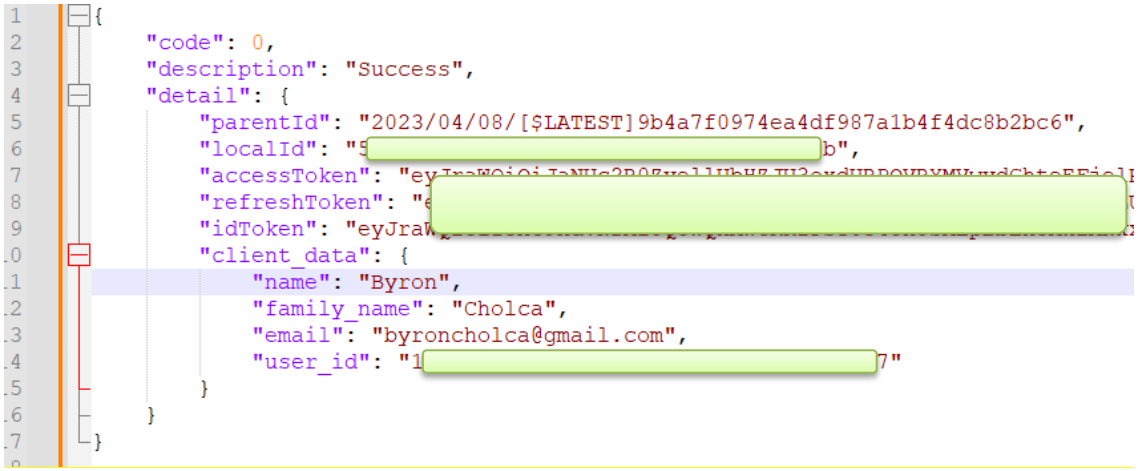


Fig. 15. Estructura del Token en formato JSON

En la Fig. 16 usando una herramienta web para decodificar el token se visualiza la información del usuario en el accessToken.

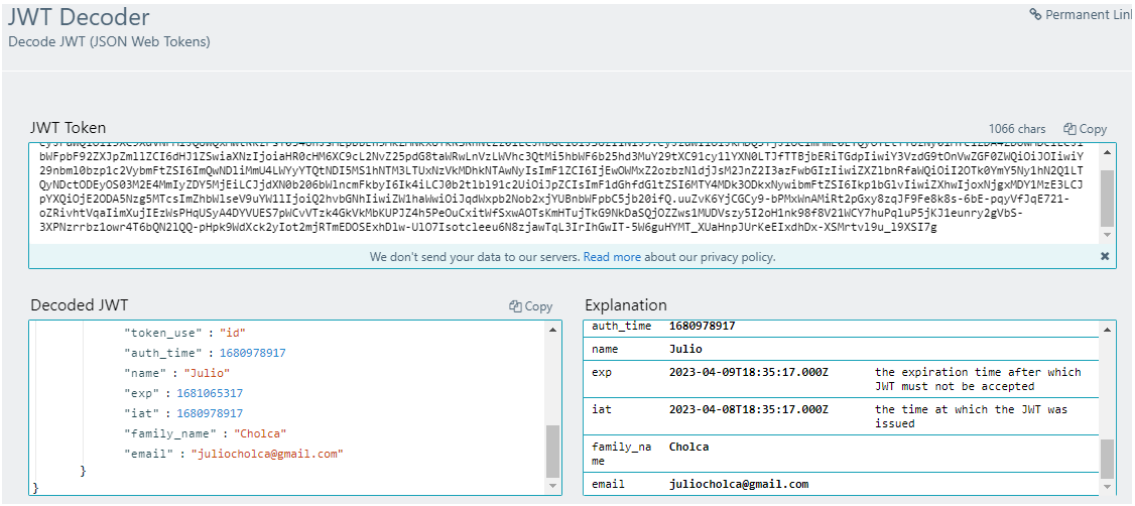


Fig. 16. Información decodificada del JWT Access token

Comunicación de microservicios: Como se indicó en la arquitectura del Api Gateway cuando el cliente requiere hacer una petición a varios recursos, se propone en la arquitectura desarrollar un API Orquestador 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:

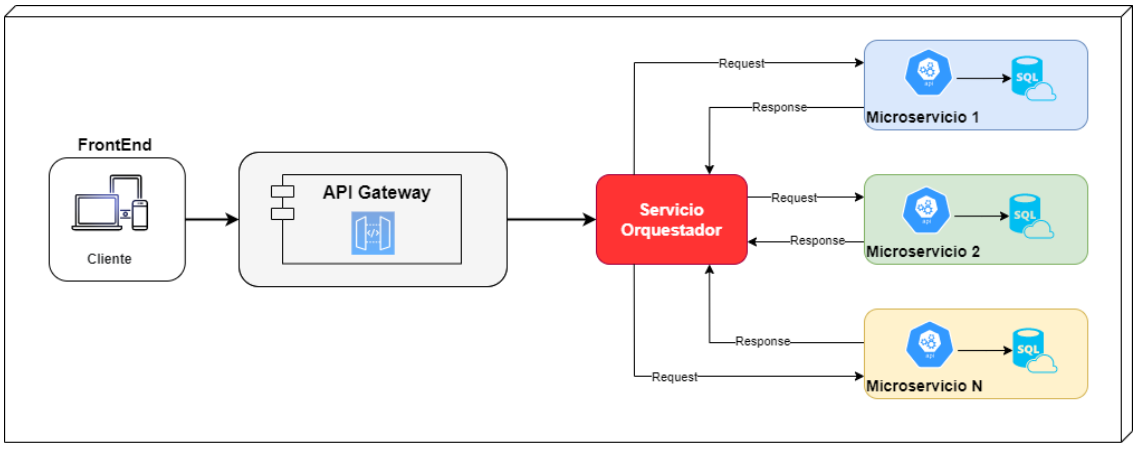


Fig. 17. Diagrama arquitectónico Servicio Orquestador

Protocolo de comunicación: Cada microservicio usa el protocolo de comunicación RestFull, por lo tanto, se exponen en API's que utilizan protocolos de comunicación estándares como HTTP/HTTPS, bajo estas tecnologías Rest el intercambio de datos es formato JSON.

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. Resumiendo, los microservicios están conectados a un bus de mensajes y canales de suscripción que están interesados. Este patrón arquitectónico se conoce como Arquitectura Dirigida por Eventos.

Al utilizar este patrón, se facilita la implementación de nuevos microservicios a la aplicación, ya que es necesario asegurarse de que los eventos a los que el nuevo microservicio estará suscrito están siendo emitidos por otros microservicios. Además, es posible agregar nuevos eventos sin afectar la lógica existente, lo que garantiza la independencia de cada microservicio.

La implementación del bus de eventos con RabbitMQ permite que los microservicios se suscriban a eventos, los publiquen y los reciban. Siendo RabbitMQ quien controle la distribución, siendo el intermediario entre el publicador de mensajes y los suscriptores, como se describe en la Fig. 18

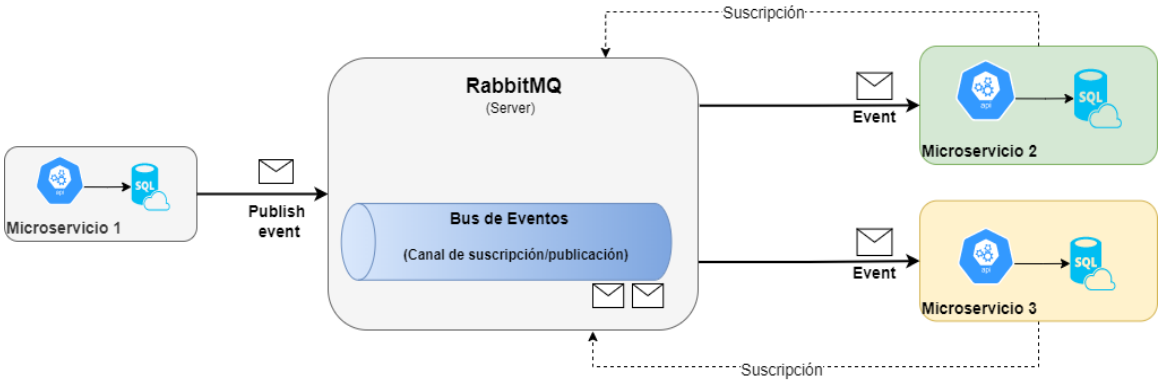


Fig. 18. Diagrama arquitectónico Bus de Eventos.

Implementación de los microservicios: En cuanto a la implementación de los microservicios, se propone usar contenedores en procesos que requieran mayor lógica de negocio y Lambdas para el caso AWS. Se propone seguir usando Lambdas para funciones pequeñas y autónomas que se ejecutan en respuesta a eventos específicos. Por ejemplo, en los servicios que manejan pagos en línea, una función lambda podría ser responsable de recibir y procesar pagos, y otra función lambda podría ser responsable de enviar confirmaciones de pago por correo electrónico. De este modo, cada microservicio se empaqueta junto con las aplicaciones y bibliotecas necesarias para su funcionamiento, lo que maximiza su portabilidad y facilita los procesos de implementación y escalabilidad. En la Fig. 19 se muestra el diagrama de cómo pueden convivir en una arquitectura servicios con Docker y Lambdas, según sea la necesidad del negocio.

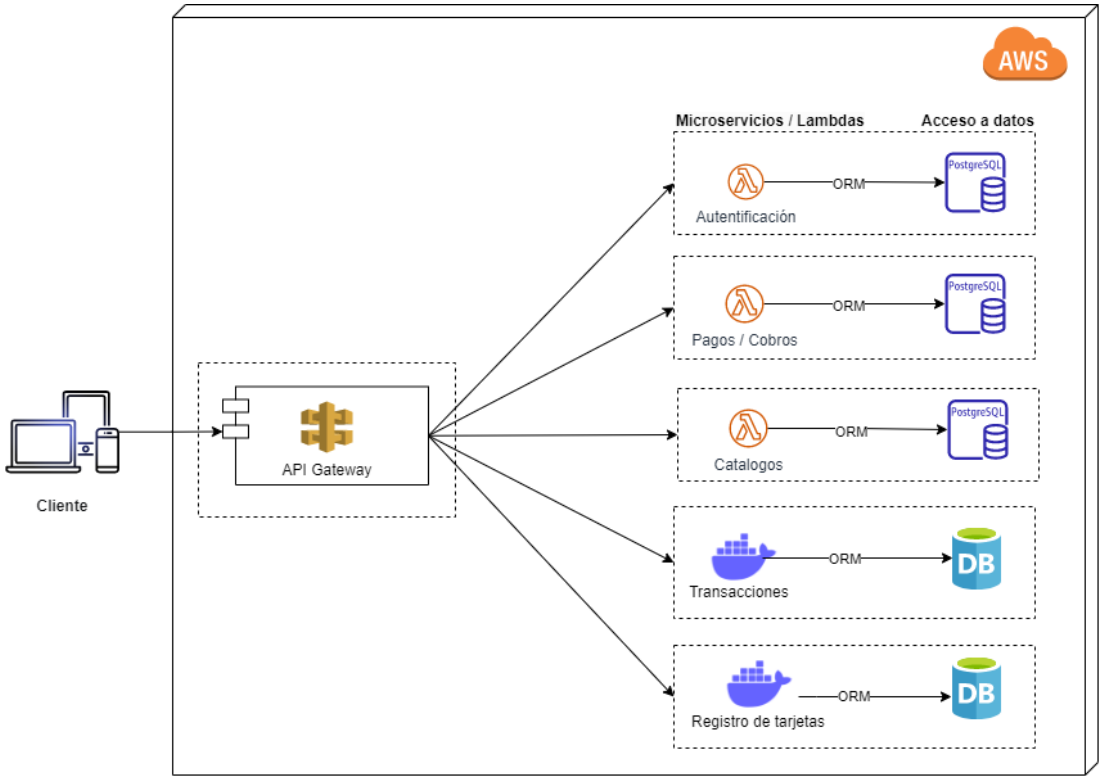


Fig. 19. Diagrama arquitectónico servicios en Docker y Lambdas

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.

De acuerdo con la arquitectura propuesta, cada microservicio y el API Gateway deben estar en contenedores diferentes para permitir la escalabilidad independiente de cada

microservicio. Al aplicar políticas adecuadas, se puede establecer que, durante períodos de alta actividad de un microservicio en particular, se incrementen de manera incremental los recursos disponibles. Posteriormente, cuando la actividad disminuya, se pueden reducir los recursos para optimizar los costos.

Tolerancia a fallas: Con el objetivo de que los microservicios de la aplicación sean tolerantes a fallas, se propone implementar el patrón Circuit Breaker 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. En lugar de intentar una y otra vez comunicarse con un servicio que está fallando, el patrón Circuit Breaker detiene temporalmente la comunicación y lo reporta como un error para buscar una solución, como se indica en la Fig. 20. Con esta propuesta se logra que no saturemos a los servicios de la Wallet y así poder tener una acción de respuesta.

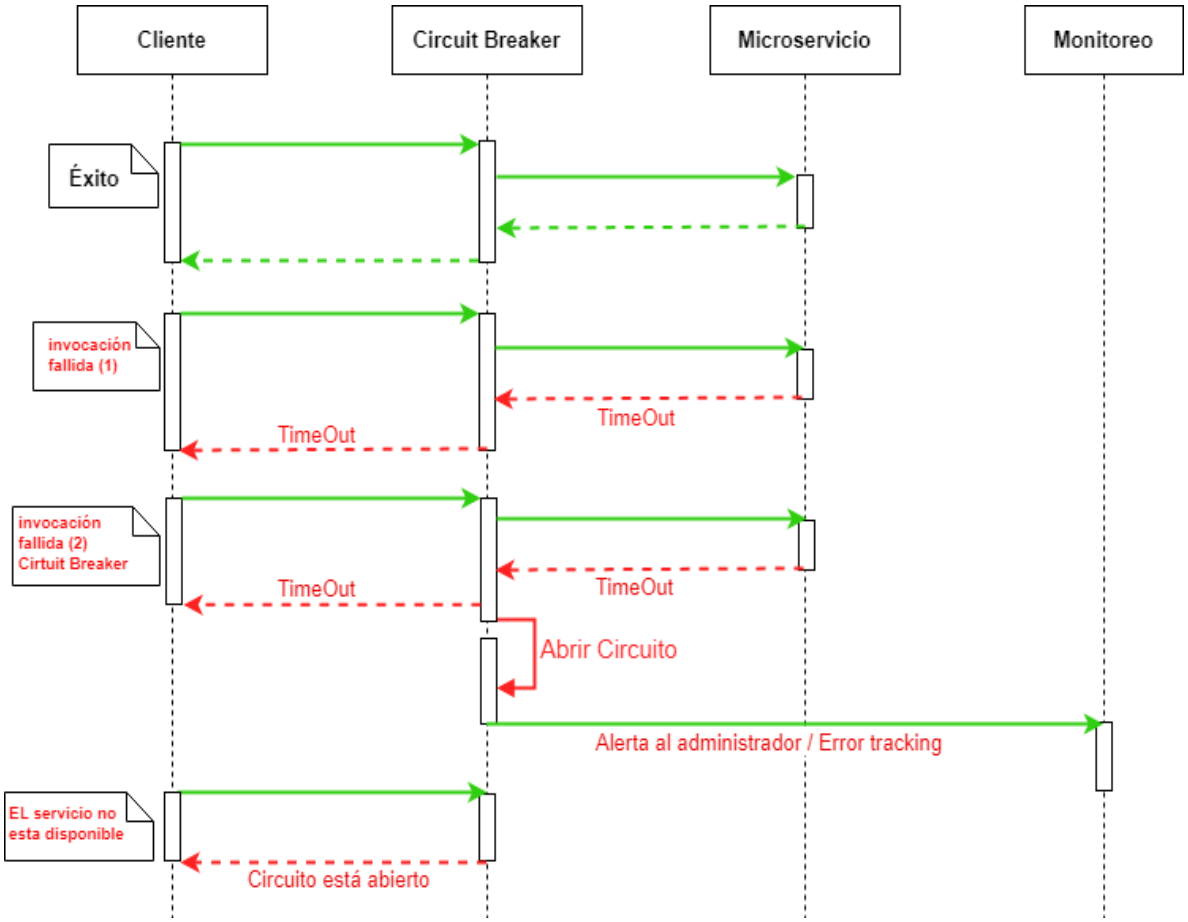


Fig. 20. Diagrama de Secuencia patrón Circuit Breaker

Base de datos: Este componente usa el patrón Base de Datos por Servicio la cual cada microservicio tiene su propia base de datos. Los datos son almacenados en una base de datos específica para el servicio y se accede a ellos a través de la API del servicio. Cada base de datos es independiente y puede ser escalada de manera individual. En la Fig. 21 se muestra parte del diagrama arquitectónico con el patrón Base de datos por servicio.

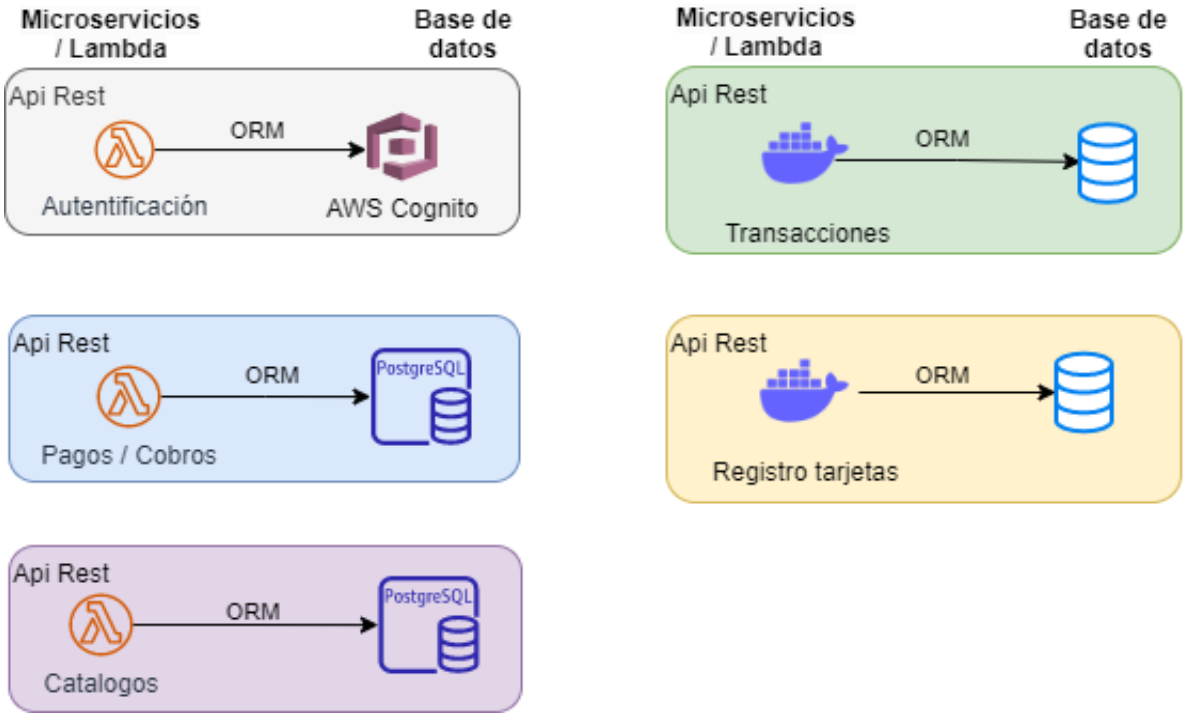


Fig. 21. Diagrama arquitectónico patrón Base de Datos por Servicio

6.3 Objetivo 3: 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.

6.3.1 Definición de casos de prueba

En la TABLA VI, TABLA VII y TABLA VIII, se definen tres casos de prueba que se detallan a continuación:

TABLA VI
CASO DE PRUEBA 1 – API AUTENTICACIÓN

Caso de prueba 1	
Microservicio:	Autenticación
Dominio del Api Gateway:	https://prod.quikly.app
API:	https://prod.quikly.app/login/login
Método:	POST
Descripción:	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.
Herramienta a utilizar:	Postman
Resultado esperado:	JWT (accessToken, refreshToken y idToken)
Procedimiento:	Enviar la clave de usuario y contraseña al API

TABLA VII
CASO DE PRUEBA 2 – API CONSULTA USUARIO

Caso de prueba 2	
Microservicio:	Usuario
Dominio del Api Gateway:	https://prod.quikly.app
API:	https://prod.quikly.app/user/getAccountPreferencesResource
Método:	GET
Descripción:	Comprobar que el microservicio retorne la información del cliente registrado en la Wallet
Herramienta a utilizar:	Postman
Resultado esperado:	Objeto Usuario en formato JSON
Procedimiento:	Enviar en el Request del Api el userId y verificar: - Estado de la Response OK (200) - Tiempo de respuesta menor a 2s. - El Response en formato JSON contenga el atributo “accountPreferences” donde se almacena la información del usuario.

TABLA VIII
CASO DE PRUEBA 3 – API TRANSACTIONS CON AUTORIZACIÓN

Caso de prueba 3	
Microservicio:	Transactions
Dominio del Api Gateway:	https://prod.quikly.app
API:	https://prod.quikly.app/transaction/transactions
Método:	POST
Descripción:	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.
Herramienta a utilizar:	Postman
Resultado esperado:	Obtener el Response correspondiente cuando se llame al Api con la respectiva autorización del token
Procedimiento:	- Usar el Api Autenticación (caso de prueba 1) - Obtener el JWT - Copiar el Token obtenido en el Authorization Header del microservicio Transactions - Si el Token es correcto se obtiene el Response correspondiente. - Caso contrario el Api devuelve un mensaje de "Unauthorized"

6.3.2 Evaluación de resultados de los casos de prueba

En la TABLA IX, TABLA X, TABLA XI, se detallan los resultados de los casos de prueba:

TABLA IX
EVALUACIÓN CASO DE PRUEBA 1 – API AUTENTICACIÓN

Evaluación caso de prueba 1	
Microservicio:	Autenticación
Dominio del Api Gateway:	https://prod.quikly.app
API:	https://prod.quikly.app/login/login
Método	POST
Cloud:	AWS
Descripción:	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
Herramienta a utilizar:	Postman

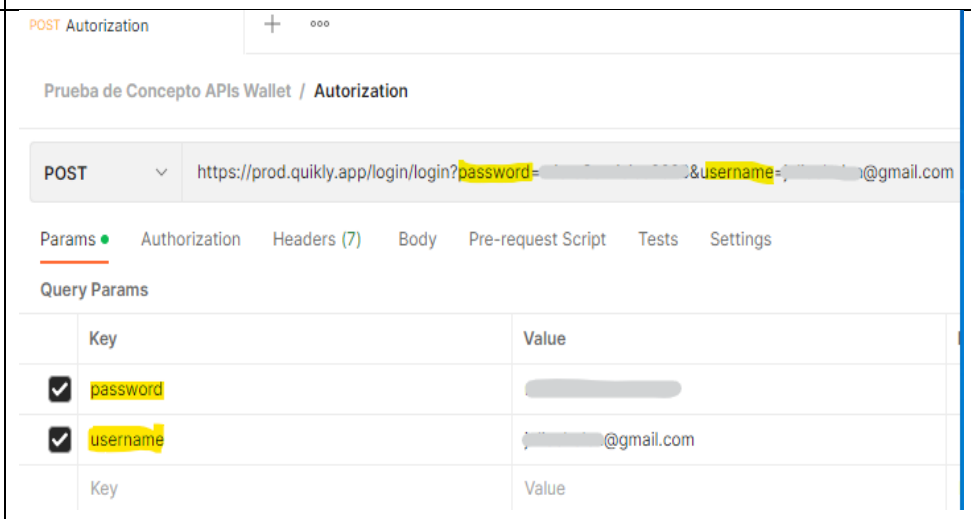
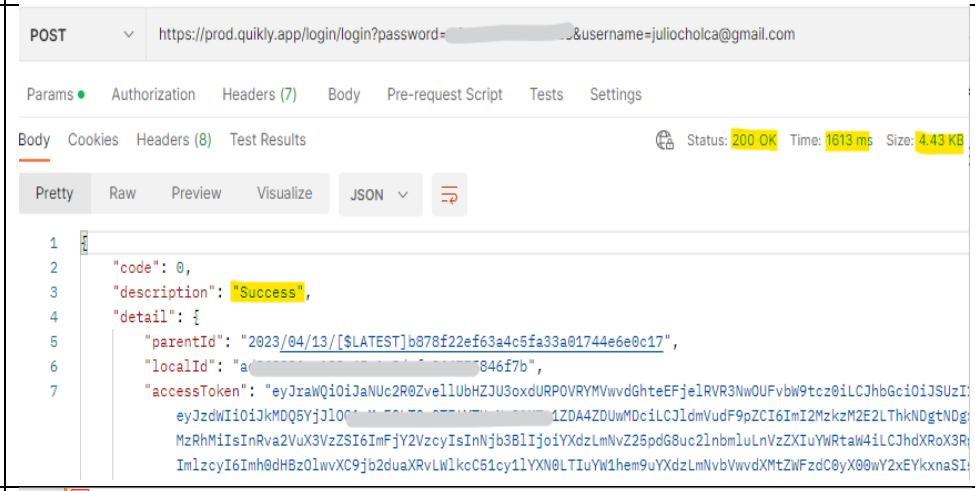
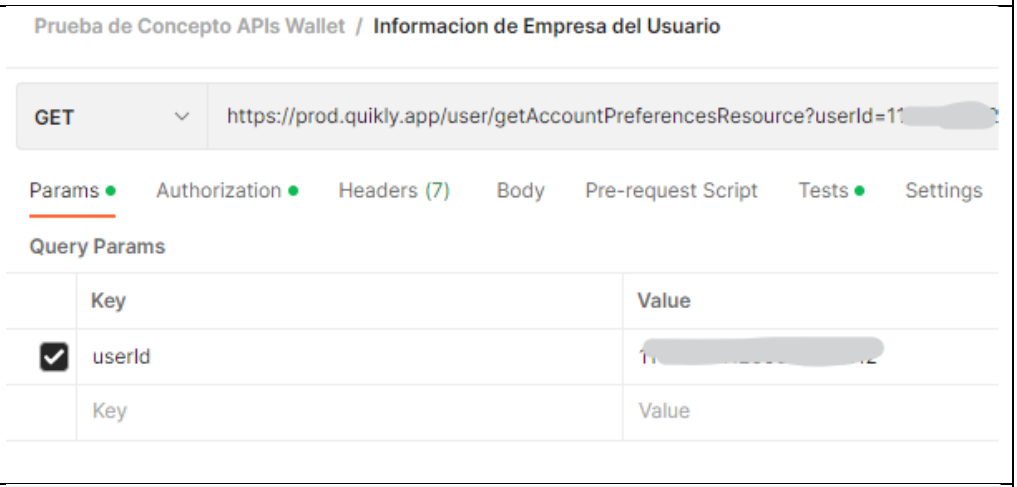
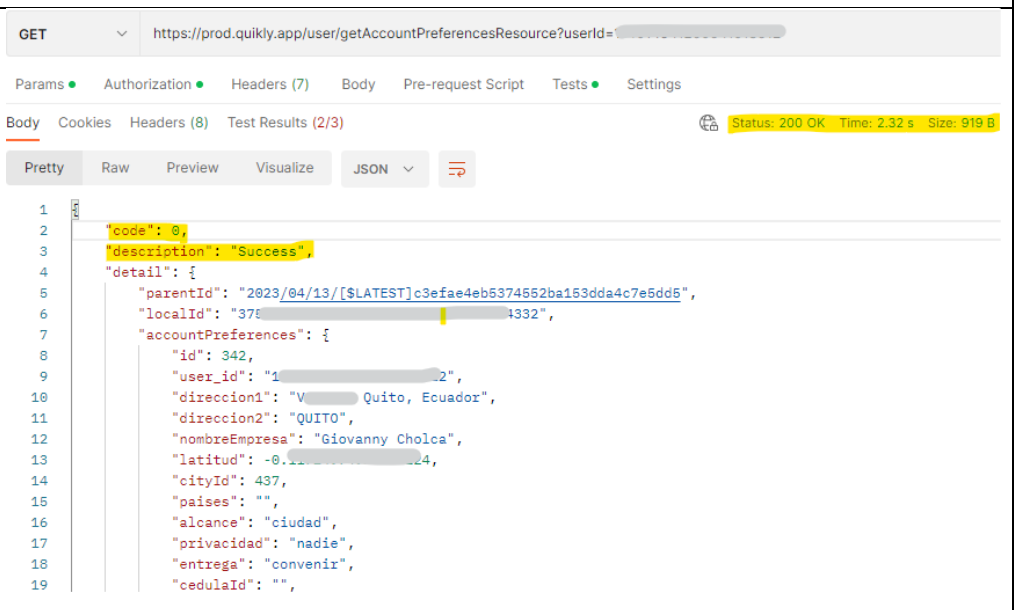
Resultado esperado:	JWT (accessToken, refreshToken y idToken)
Procedimiento:	Enviar la clave de usuario y contraseña en la ruta indicada del API.
Ejecución en la herramienta:	
Respuesta obtenida:	
Formato de Respuesta en JSON:	
Estado:	Aprobado.
Observación:	El Response del Api retorna la descripción 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.

TABLA X

EVALUACIÓN CASO DE PRUEBA 2 – API CONSULTA USUARIO

Evaluación caso de prueba 2	
Microservicio:	Usuario
Dominio del Api Gateway:	https://prod.quikly.app
API:	https://prod.quikly.app/user/getAccountPreferencesResource
Método:	GET
Descripción:	Comprobar que el microservicio retorne la información del cliente registrado en la Wallet
Herramienta a utilizar:	Postman
Resultado esperado:	Objeto Usuario en formato JSON
Procedimiento:	Enviar en el Request del Api el userId y verificar: - Estado de la Response OK (200) - Tiempo de respuesta menor a 2s. - El Response en formato JSON contenga el atributo "accountPreferences" donde se almacena la información del usuario.
Ejecución en la herramienta:	
Respuesta obtenida:	

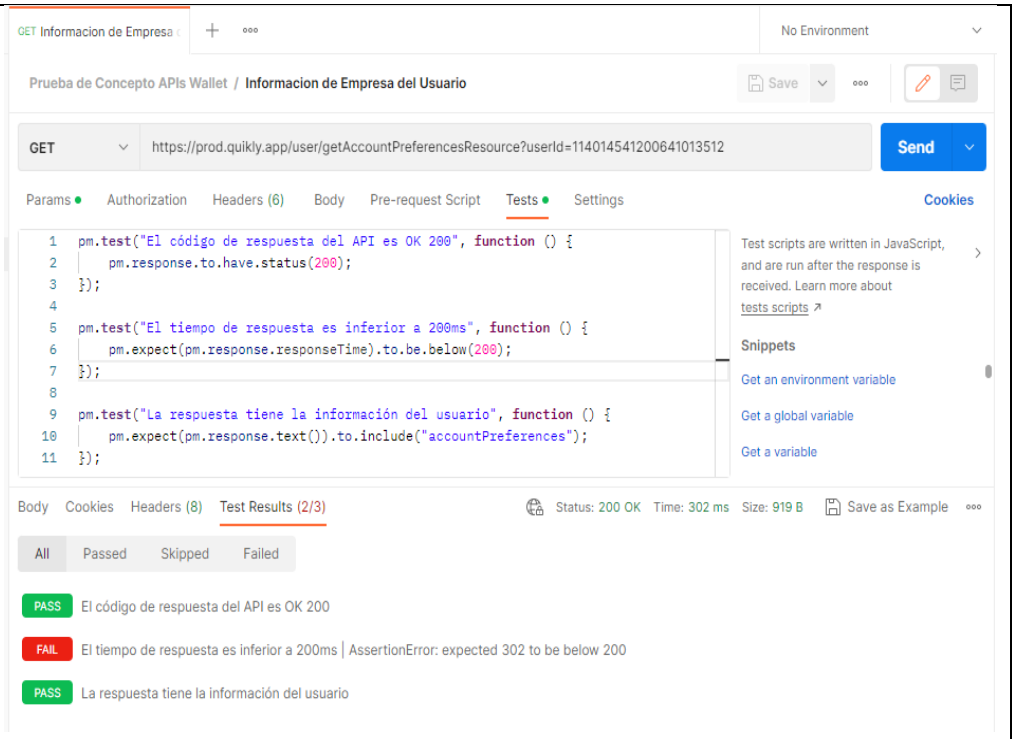
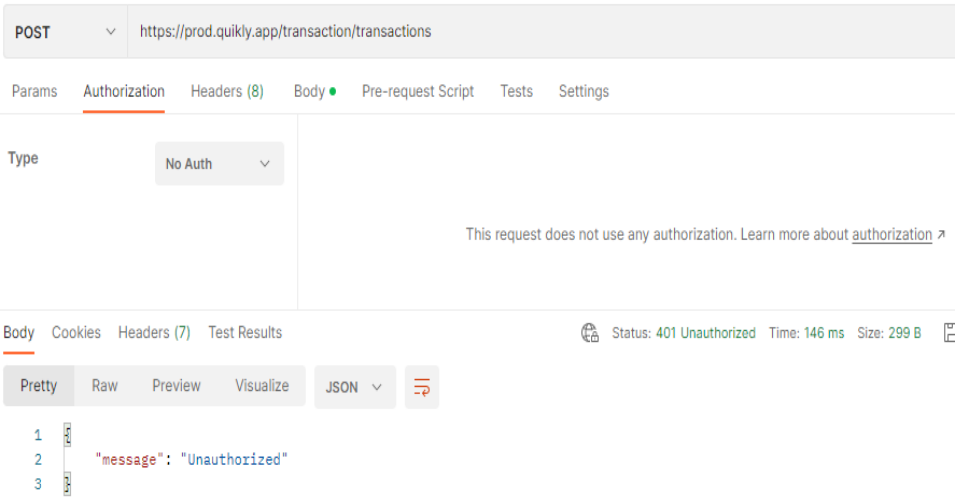
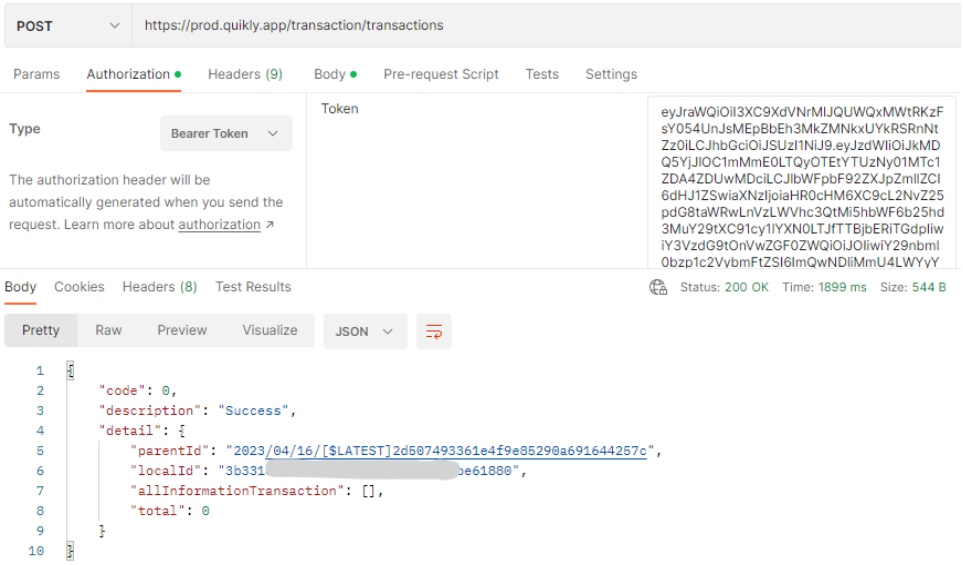
Respuesta pruebas automatizadas	
Estado:	- Estado del Response OK (200): CUMPLE - Tiempo de repuesta menor a 200ms: CUMPLE PARCIALMENTE. - El Response en formato JSON contenga el atributo “accountPreferences” donde se almacena la información del usuario: CUMPLE
Observación:	En el test el Tiempo de repuesta menor a 200ms al aplicarlo 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.

TABLA XI

EVALUACIÓN CASO DE PRUEBA 3 – API TRANSACTIONS

Evaluación Caso de prueba 3	
Microservicio:	Transactions
Dominio del Api Gateway:	https://prod.quikly.app
API:	https://prod.quikly.app/transaction/transactions
Método:	POST
Descripción:	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.
Herramienta a utilizar:	Postman
Resultado esperado:	Obtener el Response correspondiente cuando se llame al Api con la respectiva autorización del token
Procedimiento:	- Usar el Api Autenticación (caso de prueba 1) - Obtener el JWT

<table>
<tr><td></td><td>- Copiar el Token obtenido en el Authorization Header del microservicio Transactions
- Si el Token es correcto se obtiene el Response correspondiente.
- Caso contrario el Api devuelve un mensaje de “Unauthorized”</td></tr>
<tr><td>Respuesta obtenida:</td><td>- Consumir Api sin token de autorización:
 

- Consumir el Api con token de autorización:
 
</td></tr>
<tr><td>Estado:</td><td>- No permitir consumir el Api sin el Token de Autenticación: CUMPLE
- Permitir obtener el Response con el Token de Autenticación: CUMPLE</td></tr>
<tr><td>Observación:</td><td>- Los dos escenarios de prueba fueron correctos al usar el primer Api de Autenticación y luego con el Token obtenido realizar el caso de prueba.</td></tr>
</table>

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.

7. Discusión

El desarrollo de la propuesta de una arquitectura basada en microservicios para la empresa Quikly, se realizó aplicando la Investigación Proyectiva para recopilar información sobre las prácticas de desarrollo y la arquitectura actual de la empresa, junto con artículos y documentación de revistas científicas sobre arquitecturas de software basadas en microservicios. Adicionalmente se aplicó la metodología Incremental para el desarrollo de cada uno de los objetivos por cuanto esta metodología es incremental.

7.1 Objetivo 1: 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.

En esta sección 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.

Este permite tener empatía y entender la necesidad que tiene la empresa para proponer soluciones tecnologías y tener claro el objetivo 1 permite tener los insumos para poder analizar y realizar el diseño de una arquitectura basada en microservicios.

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.

Las bases teóricas ayudaron a demostrar que las nuevas tendencias de desarrollo de software requieren un enfoque colaborativo y abierto, independiente de los lenguajes de programación utilizados y de las plataformas tecnológicas en las que se desplieguen.

Los microservicios ofrecen muchas capacidades eficaces, como la implementación independiente, los límites de subsistemas seguros y la diversidad de tecnología. No

obstante, también plantean numerosos desafíos nuevos en cuanto al desarrollo de aplicaciones distribuidas y la comunicación entre ellos.

Estos aspectos conllevan un nivel de complejidad mucho mayor que una aplicación monolítica tradicional. Como consecuencia, las aplicaciones basadas en microservicios son adecuadas solo para escenarios específicos. Estos incluyen aplicaciones grandes y complejas con múltiples subsistemas en evolución. En tales casos, invertir en una arquitectura de software más compleja vale la pena, ya que a largo plazo la aplicación será más ágil y su mantenimiento será mejor.

7.2 Objetivo 2: 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.

Para el segundo incremento (objetivo 2) se inició con el estudio de los resultados de la revisión sistemática de la literatura, que son las bases teóricas de los artículos seleccionados y junto con el análisis de la definición de los requerimientos que es uno de los pasos críticos en el proceso de desarrollo de software, ya que la propuesta arquitectónica debe satisfacer los requerimientos funcionales y no funcionales establecidos.

Pero primeramente considerando el estado actual de la empresa Quicky al encontrarse en un proceso de transformación digital y en una etapa de migración de su arquitectura monolítica a microservicios se tiene como limitante el factor económico para alguna nueva inversión en personal o infraestructura tecnológica. El equipo de desarrollo trabaja de forma remota y necesitan seguir un protocolo para el manejo de tiempos y presupuestos con los proveedores y otros equipos. Y antes del análisis y diseño de la propuesta arquitectónica que toma en cuenta la Ley de Conway que nos dice *que cualquier organización que diseña aplicaciones, producirá un diseño cuya estructura es una copia de la estructura de la organización*.

Por lo tanto, Quicky al ser una empresa Fintech que desea una aplicación y diseño arquitectónico de software bien diseñado deberá examinar cuidadosamente su estructura y asegurarse de que fomente una comunicación efectiva y colaborativa entre los equipos.

El diseño propuesto de la arquitectura basada en microservicios junto con los componentes y los patrones de arquitectónicos que lo conforman satisface los requerimientos establecidos, los principales patrones y componentes propuestos dentro del diseño arquitectónico son:

- Api Gateway
- Base de datos por servicio.
- Mensajería - Bus de eventos con RabbitMQ.
- Autenticación JWT y Auth2
- Servicio Orquestador
- Implementación de los microservicios con Docker y Lambdas

El conjunto de estos 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.

Escalabilidad: es una de las principales ventajas de esta propuesta arquitectónica. Cada servicio puede ser escalado individualmente, lo que significa que se puede escalar solo el servicio que se necesita sin afectar el rendimiento de otros servicios.

Modularidad: esta propuesta de arquitectura permite que los servicios sean desarrollados y desplegados de forma independiente, tambien se ha elegido una base de datos separada para cada servicio para evitar conflictos de datos, lo que facilita la implementación de nuevas funcionalidades.

Flexibilidad que proporciona. Al dividir la aplicación en microservicios más pequeños, se puede desarrollar cada uno de ellos en diferentes tecnologías y lenguajes de programación. Esto permite a los equipos de desarrollo elegir la tecnología más adecuada para cada servicio y evitar las limitaciones impuestas por una tecnología única.

Implementación de microservicios autónomos hace que cada servicio tenga su propio ciclo de vida y, por lo tanto, no dependa de otros servicios para funcionar correctamente. El uso de contenedores resulta altamente beneficioso para la implementación de aplicaciones nativas en la nube, ya que representan una solución ligera y portable

compatible con diversas plataformas. Además, permite desacoplar la aplicación en pequeñas partes, lo que granula la arquitectura y facilita su distribución en múltiples plataformas.

Sin embargo, la propuesta de esta arquitectura basada en microservicios también presenta algunos desafíos. Uno de los principales desafíos es:

La complejidad de la gestión de múltiples microservicios: Cada servicio tiene su propia base de datos y es responsable de una funcionalidad específica, lo que significa que la coordinación entre los servicios puede ser difícil de administrar.

Se necesita una infraestructura sólida de comunicación: para garantizar que los servicios se comuniquen correctamente y que no haya pérdida de datos.

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.

7.3 Objetivo 3: Evaluar el rendimiento y validar el funcionamiento de la solución basada en microservicios en la Cloud mediante cargas de estrés.

Para el último objetivo, tiene la finalidad de realizar pruebas de concepto para consumir las Apis de Autenticación, catálogos de la aplicación, y un Api de consulta con seguridad de autenticación, que se han publicado en un ambiente Cloud AWS. Y con el uso de la herramienta Postman que es un cliente para consumir Api Rest, realizado peticiones a los microservicios, verificando que las respuestas de cada petición sean la correctas y en formato Json y midiendo los tiempos de respuesta

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. 22**

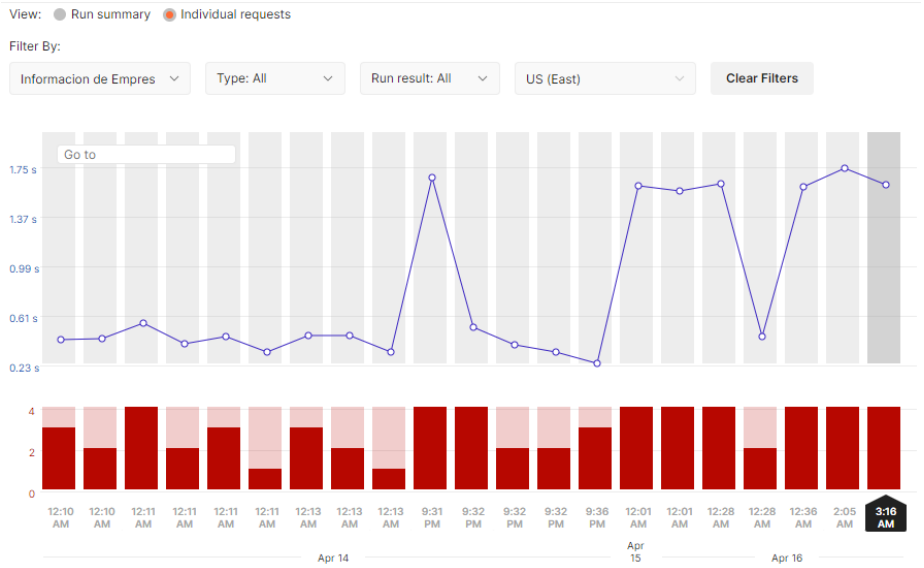


Fig. 22. Pruebas monitoreo – microservicio usuario

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.

8. Conclusiones

Según los objetivos planteados para esta investigación, se han obtenido varias conclusiones que se detallan a continuación.

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. En cuanto a las comunicaciones, es posible definirlas mediante solicitudes HTTP, lo cual puede ser realizado directamente a través del balanceador de carga, utilizando el servicio de AWS API Gateway con REST API.

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. El API Gateway actúa como un proxy para las solicitudes de la API, lo que permite la centralización del control de acceso y la implementación de políticas de seguridad. Además, permite una fácil gestión y monitoreo de las solicitudes entrantes y salientes. 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.

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. Los tokens son autocontenidos, lo que significa que la información de autenticación y autorización se almacena directamente en el token, reduciendo la necesidad de buscar información adicional en una base de datos o sistema de almacenamiento de sesión. Además, JWT es fácilmente integrable con numerosas bibliotecas y herramientas, lo que lo hace una opción popular para los desarrolladores. Sin embargo, es importante tener en cuenta sus limitaciones y usarlo en conjunto con otras medidas de seguridad para garantizar la protección adecuada de los datos y recursos de la aplicación.

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. Estas funciones son llamadas por el servicio de API Gateway, lo que permite centrarse solo en el código, sin la necesidad de suministrar infraestructura, lo que resulta en un ahorro en costos para las organizaciones.

La implementación de esta arquitectura tendrá un impacto significativo en la organización. Será necesario un mayor compromiso y colaboración entre los equipos de desarrollo y operaciones para garantizar el éxito de la implementación. Además, puede ser necesario proporcionar más recursos y capacitar al personal en la implementación y gestión de esta arquitectura en ambientes Cloud.

Para validar el funcionamiento del patrón arquitectónico Api Gateway en AWS las pruebas de concepto fueron de vital importancia, se comprobó la comunicación con las APIs, el funcionamiento de los microservicios y midiendo los tiempos de respuesta.

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.

9. Recomendaciones

Un arquitecto de software debe enfatizar en el análisis de limitaciones y costes de las tecnologías y nuevas aplicaciones a desarrollar, para no sobrepasar la diversidad tecnológica y que se convierta en un problema que a la larga que puede generar una deuda técnica.

Para implementar servicios en la nube de manera efectiva en soluciones financieras, es necesario contar con un ecosistema que ofrezca las mejores características. Con la presencia de múltiples proveedores de servicios en la nube, es importante identificar a los líderes del mercado que ofrezcan ofertas estandarizadas, servicios de virtualización y automatización, y servicios adecuados para la adopción de estrategias innovadoras. Además, estos proveedores deben ofrecer una amplia gama de servicios para cubrir diversos casos de uso y destacarse en muchos de ellos.

Para utilizar contenedores y microservicios, las organizaciones a menudo necesitan implementar múltiples herramientas, lo que puede conducir a una mayor complejidad. Es importante tener en cuenta que, aunque los microservicios y contenedores ofrecen numerosos beneficios, también requieren de una gestión y mantenimiento cuidadosos. Para ello, se necesita una automatización efectiva para los contenedores, y es fundamental tener una visibilidad clara del estado de todos los contenedores (Docker o Lambdas), siendo crucial que se administren cuidadosamente las dependencias entre los contenedores para evitar problemas en el funcionamiento de la aplicación en su conjunto.

Para trabajos futuros se recomienda hacer un plan de automatización DevOps, los equipos de desarrollo pueden trabajar en componentes débilmente acoplados de la aplicación y elegir la tecnología más adecuada para sus necesidades. Esto permite que los nuevos miembros del equipo se integren y desarrollen más rápidamente, ya que solo necesitan familiarizarse con el servicio en el que están trabajando.

Las empresas que deseen adoptar los principios de cultura, organización, procesos y herramientas de la filosofía DevOps pueden beneficiarse del uso de microservicios, ya que pueden aumentar la agilidad en los despliegues, mejorar la calidad del software entregado, facilitar la creación de equipos autónomos y eliminar barreras interdepartamentales. Los microservicios se han convertido en un componente clave para lograr estos objetivos en la implementación de una filosofía DevOps.

10. Bibliografía

[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 (accedido 19 de abril de 2023).

[2] J. Rubio, J. Jiménez, y D. Acosta, «Evolución de los medios de pago del Ecuador en el contexto de pandemia Covid-19». Accedido: 9 de abril de 2023. [En línea]. Disponible en: https://contenido.bce.fin.ec/documentos/Administracion/snp-estadistica-2.pdf

[3] «Los medios de pago electrónico crecen durante la pandemia - Banco Central del Ecuador». https://www.bce.fin.ec/boletines-de-prensa-archivo/los-medios-de-pago-electronico-crecen-durante-la-pandemia (accedido 13 de abril de 2023).

[4] L. Zambrano, «Las billeteras digitales crecen y amplian los servicios de pagos», 9 de febrero de 2023. https://www.expreso.ec/actualidad/economia/billeteras-digitales-crecen-amplian-servicios-pagos-150003.html (accedido 20 de abril de 2023).

[5] V. René, E. Encarnación, S. Caridad, R. Quesada, O. Máximina, y E. Merchán, «Billetera electrónica móvil: una alternativa de pago del sistema financiero ecuatoriano», *Contabilidad y Negocios*, vol. 15, n.º 30, pp. 24-42, dic. 2020, doi: 10.18800/CONTABILIDAD.202002.002.

[6] «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 (accedido 13 de abril de 2023).

[7] J. Rubio, B. Pérez, D. Acosta, y J. Arroyo, «Preferencias en el uso de pagos electrónicos en el Ecuador», *Cuestiones Económicas*, vol. 31, n.º 1, p. Jeniffer Rubio-Dayana Acosta y John Arroyo, jun. 2021, doi: 10.47550/RCE/31.1.3.

[8] M. R. Llácer Matacás, «El cliente de servicios de pago: contratación de tarjetas y responsabilidad en el sistema de pagos», 2007.

[9] J. M. Lara y M. Reis, «Un Análisis Inicial del Dinero Electrónico en Ecuador y su Impacto en la Inclusión Financiera», *Cuestiones Económicas*, vol. 25, n.º 1, p. Jorge Moncayo Lara-Jorge Moncayo Lara, 2015, Accedido: 13 de abril de 2023. [En línea]. Disponible en: https://estudioseconomicos.bce.fin.ec/index.php/RevistaCE/article/view/77

[10] «Software Architecture: Foundations, Theory, and Practice - Richard N. Taylor, Nenad Medvidovic, Eric Dashofy - Google Libros». https://books.google.com.ec/books?id=j9pdGQAACAAJ&printsec=frontcover&hl=es&source=gbs_ge_summary_r&cad=0#v=onepage&q&f=false (accedido 14 de abril de 2023).

[11] M. Fowler y J. Lewis, «Microservices», 25 de marzo de 2014. https://martinfowler.com/articles/microservices.html (accedido 14 de abril de 2023).

[12] «¿Qué son los microservicios? | AWS». https://aws.amazon.com/es/microservices/ (accedido 14 de abril de 2023).

[13] S. Hassan, N. Ali, y R. Bahsoon, «Microservice Ambients: An Architectural Meta-Modelling Approach for Microservice Granularity», *Proceedings - 2017 IEEE International Conference on Software Architecture, ICSA 2017*, pp. 1-10, may 2017, doi: 10.1109/ICSA.2017.32.

[14] S. Hassan y R. Bahsoon, «Microservices and their design trade-offs: A self-adaptive roadmap», *Proceedings - 2016 IEEE International Conference on Services Computing, SCC 2016*, pp. 813-818, ago. 2016, doi: 10.1109/SCC.2016.113.

[15] Y. Yale, H. Silveira, y M. Sundaram, «A microservice based reference architecture model in the context of enterprise architecture», *2016 IEEE Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC)*, pp. 1856-1860, feb. 2016, doi: 10.1109/IMCEC.2016.7867539.

[16] N. Dragoni *et al.*, «Microservices: Yesterday, today, and tomorrow», *Present and Ulterior Software Engineering*, pp. 195-216, nov. 2017, doi: 10.1007/978-3-319-67425-4_12/COVER.

[17] L. De Lauretis, «From monolithic architecture to microservices architecture», *Proceedings - 2019 IEEE 30th International Symposium on Software Reliability Engineering Workshops, ISSREW 2019*, pp. 93-96, oct. 2019, doi: 10.1109/ISSREW.2019.00050.

[18] J. I. P. Velasco, A. I. R. Ruiz, y H. A. P. Alvira, «Arquitectura basada en microservicios para aplicaciones web», *Tecnología Investigación y Academia*, vol. 7, n.º 2, pp. 12-20, ene. 2020, Accedido: 13 de abril de 2023. [En línea]. Disponible en: https://revistas.udistrital.edu.co/index.php/tia/article/view/13364

[19] J. GUERRERO, «La arquitectura de microservicios: qué es y cómo funciona - Asesoftware», 19 de enero de 2019. https://asesoftware.com/la-arquitectura-de-microservicios/ (accedido 13 de abril de 2023).

[20] «¿Qué son los microservicios? | AWS». https://aws.amazon.com/es/microservices/ (accedido 14 de abril de 2023).

[21] «Amazon API Gateway | API Management | Amazon Web Services». https://aws.amazon.com/es/api-gateway/ (accedido 14 de abril de 2023).

[22] «What can RabbitMQ do for you? — RabbitMQ». https://www.rabbitmq.com/features.html (accedido 14 de abril de 2023).

[23] S. Fix, «Michael T. Nygard-Release It!_ Design and Deploy Production-Ready Software-Pragmatic Bookshelf (2018)», Accedido: 16 de abril de 2023. [En línea]. Disponible en: https://pragprog.com/titles/mnee2/release-it-second-edition

[24] «Patrón de disyuntor - Azure Architecture Center | Microsoft Learn». https://learn.microsoft.com/es-es/azure/architecture/patterns/circuit-breaker (accedido 16 de abril de 2023).

[25] «Microservices vs. service-oriented architecture – O'Reilly». https://www.oreilly.com/radar/microservices-vs-service-oriented-architecture/ (accedido 14 de abril de 2023).

[26] D. Taibi, V. Lenarduzzi, y C. Pahl, «Architectural Patterns for Microservices: A Systematic Mapping Study», *CLOSER 2018 - Proceedings of the 8th International Conference on Cloud Computing and Services Science*, vol. 2018-January, pp. 221-232, 2018, doi: 10.5220/0006798302210232.

[27] «¿Qué es la computación? - Explicación de la computación en la nube para empresas - AWS». https://aws.amazon.com/es/what-is/compute/ (accedido 14 de abril de 2023).

[28] «¿Qué es la computación en la nube? | Google Cloud | Google Cloud». https://cloud.google.com/learn/what-is-cloud-computing?hl=es-419 (accedido 14 de abril de 2023).

[29] «¿Qué es la computación en la nube? IBM». https://www.ibm.com/mx-es/topics/cloud-computing (accedido 14 de abril de 2023).

[30] P. Mell y T. Grance, «The NIST Definition of Cloud Computing», sep. 2011, doi: 10.6028/NIST.SP.800-145.

[31] «¿Qué es la virtualización? | IBM». https://www.ibm.com/mx-es/topics/virtualization (accedido 14 de abril de 2023).

[32] R. Dua, A. R. Raja, y D. Kakadia, «Virtualization vs Containerization to Support PaaS», *2014 IEEE International Conference on Cloud Engineering*, pp. 610-614, sep. 2014, doi: 10.1109/IC2E.2014.41.

[33] C. Pahl, «Containerization and the PaaS Cloud», *IEEE Cloud Computing*, vol. 2, n.º 3, pp. 24-31, may 2015, doi: 10.1109/MCC.2015.51.

[34] M. Koskinen, T. Mikkonen, y P. Abrahamsson, «Containers in Software Development: A Systematic Mapping Study», *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture*

Notes in Bioinformatics), vol. 11915 LNCS, pp. 176-191, 2019, doi: 10.1007/978-3-030-35333-9_13.

[35] «¿Qué es AWS Lambda? - AWS Lambda». https://docs.aws.amazon.com/es_es/lambda/latest/dg/welcome.html (accedido 14 de abril de 2023).

[36] «Docker overview | Docker Documentation». https://docs.docker.com/get-started/overview/ (accedido 14 de abril de 2023).

[37] A. S. R. Molina, M. G. AGUIRRE, R. F. H. ALARCON, E. S. CARMONA, y V. A. HILARIO, «¿Qué es DevOps? Definición y características», *Revista Innova Ingeniería*, vol. 1, n.º 7, pp. 8-8, 2022, Accedido: 14 de abril de 2023. [En línea]. Disponible en: https://innovaingenieria.uagro.mx/innova/index.php/innova/article/view/134

[38] «Hurtado, Guia Para La Comprension Holistica De La Ciencia Unidad III». https://www.calameo.com/read/00441616639f9029c29f4 (accedido 14 de abril de 2023).

11. Anexos

Anexo 1. Guión de la entrevista no estructurada.

1. ¿En qué lenguaje de programación están desarrollador los servicios Rest?
2. ¿Con que otros sistemas interactúa la Wallet Electrónica?
3. ¿Cuántos usuario interactúan en la aplicación?
4. ¿Cómo se registra en la aplicación?
5. ¿Cuál es el proceso para realizar un pago?
6. ¿Cuál es el proceso para realizar un cobro?
7. ¿Cuál es el proceso para realizar la verificación con el reconocimiento facial?
8. ¿Cómo se registra una tarjeta?
9. ¿Cómo se genera un link de pagos?
10. ¿Qué metodología de desarrollo de software utilizan en el equipo?
11. ¿Con que proveedor en la nube se van a publicar los servicios Rest?
12. ¿Utilizan algún patrón arquitectónico?

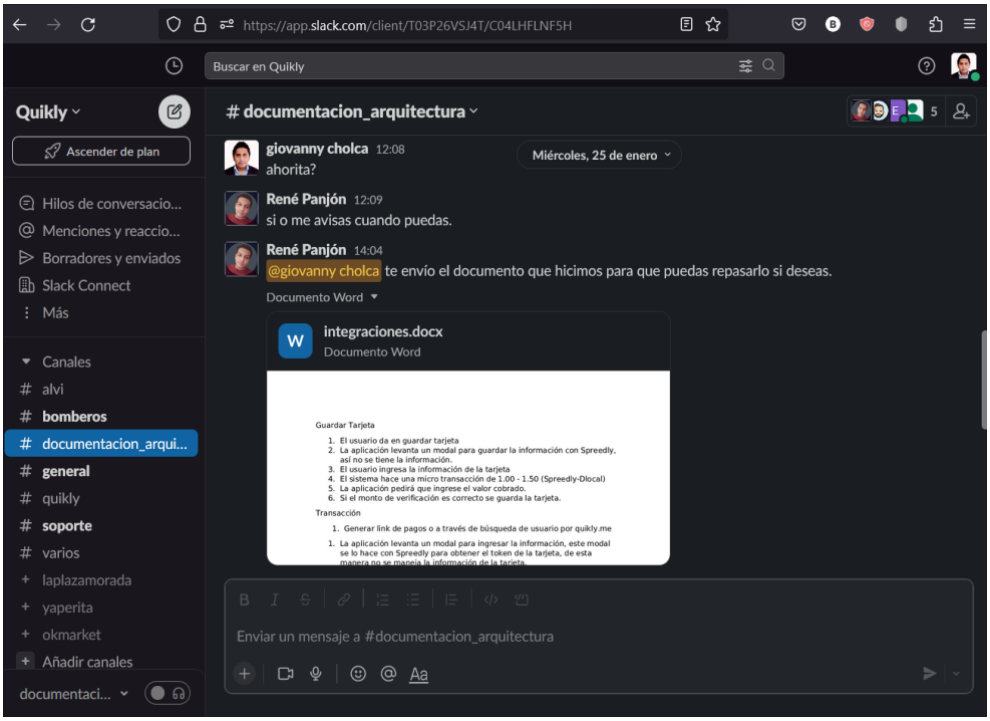


FIG. 23 ENTREVISTAS CON LA HERRAMIENTA SLACK

Anexo 2. Búsqueda en librerías científicas

- IEEEXplore: https://ieeexplore.ieee.org/

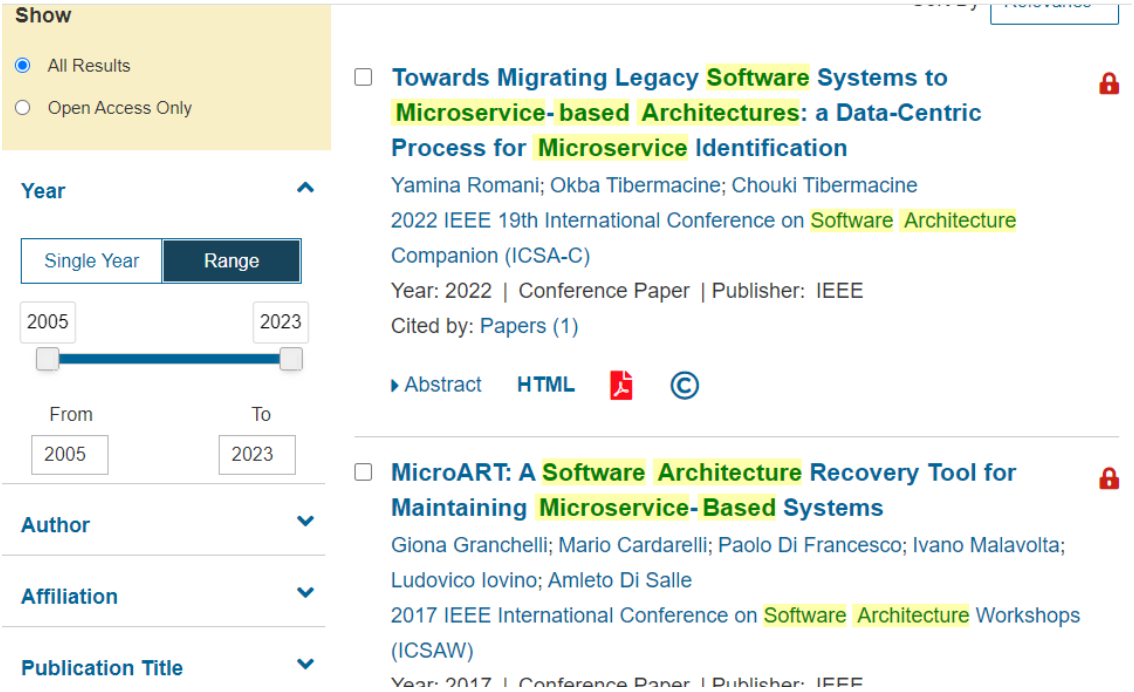


Fig. 24. Consulta artículos en IEEEXplore

- Google Académico: https://scholar.google.es/

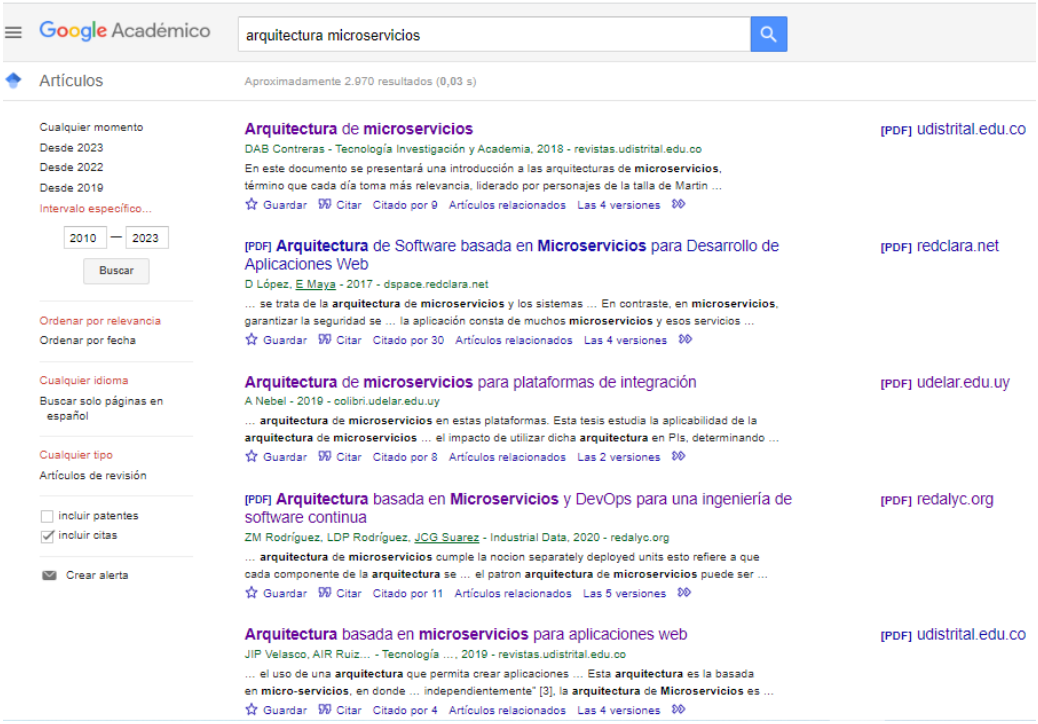


Fig. 25. Consulta artículos en Google Académico

- Dialnet, de la Universidad de La Rioja: https://dialnet.unirioja.es/

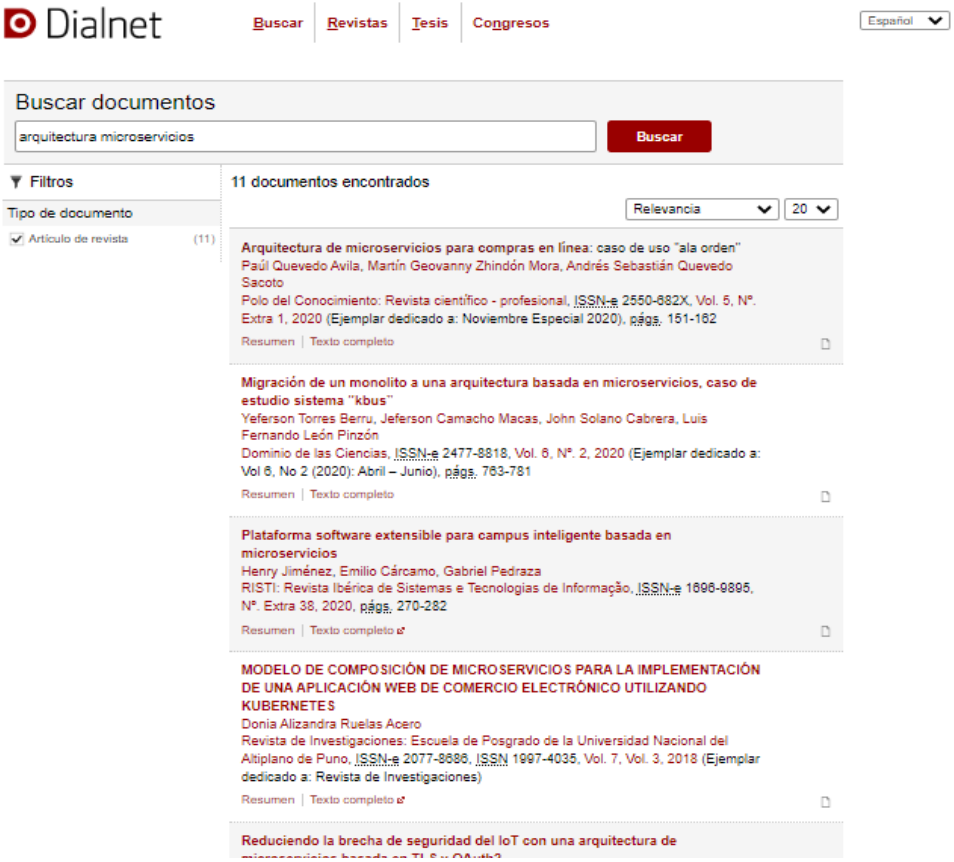


Fig. 26. Consulta articulos en Dialnet

- ACM Digital Library: https://dl.acm.org/

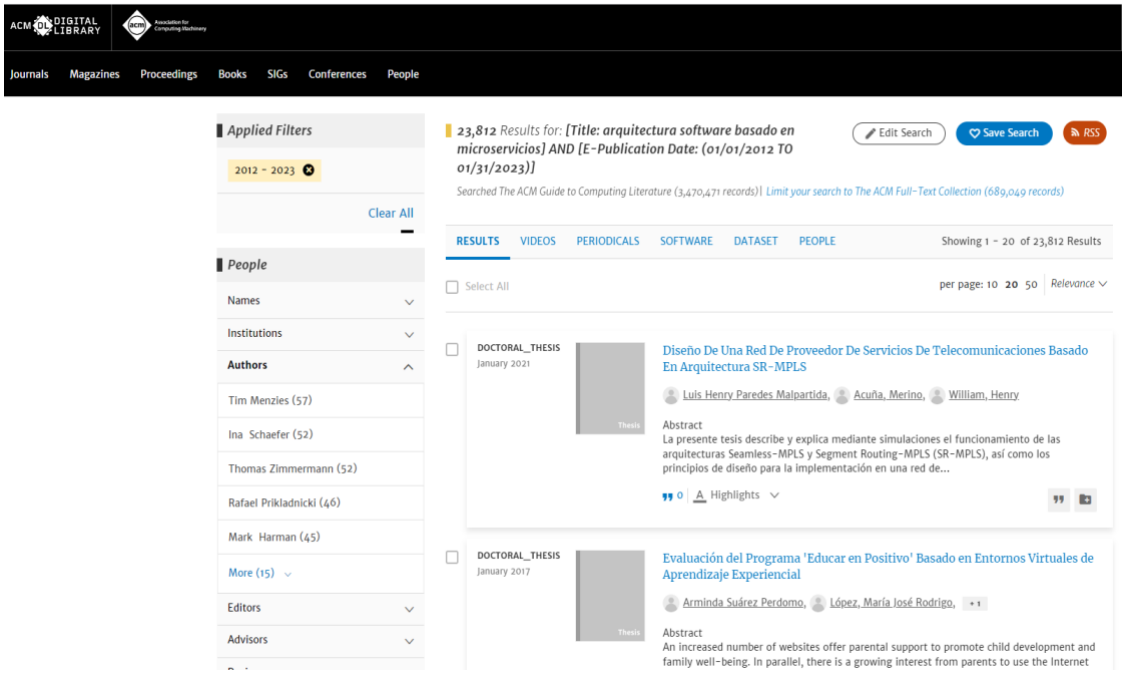


Fig. 27. Consulta artículos en Dialnet

Anexo 3. Criterios de investigación y motivación

TABLA XII
CRITERIOS DE INVESTIGACIÓN Y MOTIVACIÓN

ID	Criterios de selección	Motivación
1	¿Qué son las arquitecturas basadas en microservicios y cuáles son sus ventajas y desventajas?	Comprender el concepto de arquitecturas basadas en microservicios, así como sus fortalezas y debilidades.
2	¿Diferencias entre arquitecturas basadas en microservicios y arquitecturas monolíticas?	Analizar las diferencias entre las arquitecturas basadas en microservicios y las arquitecturas tradicionales, y entender cuándo es apropiado utilizar cada una.
3	¿Cuáles son las herramientas y tecnologías clave utilizadas en las arquitecturas basadas en microservicios, como contenedores, orquestadores, API Gateways, ¿entre otras?	Conocer las tecnologías que son esenciales para implementar una arquitectura basada en microservicios, y entender cómo se interrelacionan entre sí.
4	¿Cuáles son las mejores prácticas para diseñar e implementar una arquitectura basada en microservicios?	Conocer las prácticas recomendadas para crear una arquitectura basada en microservicios, que permitan alcanzar los beneficios esperados.
5	¿Cómo se pueden integrar y orquestar servicios de microservicios en una arquitectura más amplia, como una nube híbrida o multicloud?	Entender cómo los microservicios se pueden utilizar en un contexto más amplio, integrándose con la Nube híbrida o multicloud
6	¿Cómo se puede garantizar la escalabilidad, seguridad y alta disponibilidad en una arquitectura basada en microservicios?	Conocer las prácticas recomendadas para asegurar que una arquitectura basada en microservicios pueda escalar, ser segura y estar disponible en todo momento.

Anexo 4. Criterios de investigación y motivación

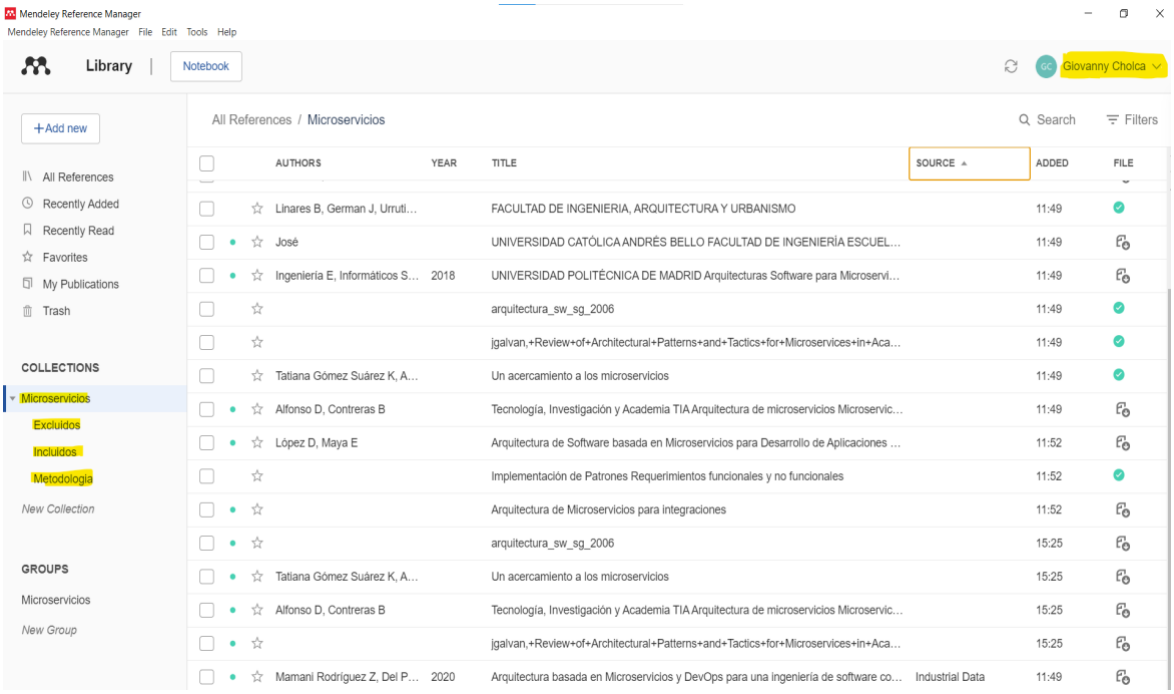


Fig. 28. Gestión de artículos mediante la herramienta Mendeley

Anexo 5. Certificado traducción al idioma inglés

Quito, 24 de abril de 2023

CERTIFICADO

Yo, Ing. Kevin Haro Msg. graduado en La Universidad de Manchester en el Reino Unido, con certificación en el idioma inglés en Cambridge y docente del idioma inglés en La Unidad Educativa Adventista Gedeón, certifico que la traducción al idioma inglés del resumen del trabajo de titulación: **Diseño de una arquitectura basada en microservicios para una Wallet Electrónica**, perteneciente al Ing. Byron Giovanny Cholca Campués, con C.I. 1715851406, está traducido en su integridad, manteniendo el mismo mensaje al texto original en español.

Es todo lo que puedo certificar en honor a la verdad.

Atentamente.
Ing. Kevin Haro Mgs.
C.I. 1724157084

Registro Senescyt N°: 1036-2022-2559613 & 8261152167