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 en la nube para mejorar la escalabilidad y agilidad del aplicativo Ktaxi de la empresa Kradac Cia. Ltda.

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

AUTOR:
Manuel Stalin Armijos Ordóñez

DIRECTOR:
Ing. Roberth Gustavo Figueroa Díaz, Mg. Sc.

Loja - Ecuador
2023

Certificación

Loja, 22 de abril de 2023

Ing. Roberth Gustavo Figueroa Díaz, 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 en la nube para mejorar la escalabilidad y agilidad del aplicativo Ktaxi de la empresa Kradac cia. Ltda**, previo a la obtención del título de **Magíster en Ingeniería en Software**, de autoría del estudiante **Manuel Stalin Armijos Ordóñez**, con cédula de identidad Nro. **1105593238**, 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.

Ing. Roberth Gustavo Figueroa Díaz, Mg. Sc.
DIRECTOR DEL TRABAJO DE TITULACIÓN

Autoría

Yo, **Manuel Stalin Armijos Ordóñez**, 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: 

Cédula de Identidad: 1105593238

Fecha: 03/05/2023

Correo electrónico: manuel.s.armijos@unl.edu.ec

Teléfono: 0991189175

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, **Manuel Stalin Armijos Ordóñez**, declaro ser autor del Trabajo de Titulación denominado: **Diseño de una arquitectura basada en microservicios en la nube para mejorar la escalabilidad y agilidad del aplicativo Ktaxi de la empresa Kradac Cia. Ltda.,** 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: Firmado electrónicamente por: MANUEL STALIN ARMIJOS ORDONEZ

Autor: Manuel Stalin Armijos Ordóñez
Cédula de identidad: 1105593238
Dirección: Av. Ángel felicísimo rojas y Quinta San Luis
Correo electrónico: manuel.s.armijos@unl.edu.ec
Teléfono: 0991189175

DATOS COMPLEMENTARIOS:
Director del Trabajo de Titulación: Ing. Roberth Gustavo Figueroa Díaz Mg. Sc

Dedicatoria

El presente Trabajo de Titulación lo dedico a mis queridos padres **ANA** quien ha sido mi sostén en cada momento de mi vida, por su amor incondicional y por creer en mí, incluso cuando yo mismo dudaba de mis capacidades, mi padre biológico **MANUEL** quien me ha apoyado incondicionalmente en todo momento brindándome su paciencia y confianza en todo momento, mi padrastro **WILLAN**, por ser mi inspiración y por enseñarme el valor del esfuerzo y la dedicación, también a mi querida novia **CELENA** quien me supo dar ánimos y alientos en los momentos más difíciles para poder continuar y no ceder ante las adversidades, a mí abuelita ARCELIA por brindarme su apoyo absoluto cuando más lo he necesitado, a mis hermanos **EDINSON** y **VERÓNICA**, a mi sobrinos **MISHELL, SOFIA, ARIEL** y **MARLON**, a mis compañeros de trabajo y en general a toda mi familia que hicieron posible el poder alcanzar tan prestigiosa meta en mi vida.

Manuel Stalin Armijos Ordóñez

Agradecimiento

Quiero dejar constancia de mi agradecimiento sincero a mis padres, novia, familiares, compañeros y amigos que siempre me ofrecieron su apoyo incondicional que hoy se ven reflejados en este logro personal.

Agradezco infinitamente a Dios por brindarme salud, bienestar y la capacidad imprescindible para poder adquirir los conocimientos necesarios para desarrollar con éxito el presente Trabajo de Titulación.

De manera especial, un agradecimiento sincero y profundo a mis compañeros de trabajo del departamento de Tecnologías de la Información de la empresa Kradac Cia. Ltda., que gracias a su experiencia supieron guiarme para cumplir con mis metas.

Finalmente, un agradecimiento especial a mi director de Trabajo de Titulación Ing. Roberth Gustavo Figueroa Díaz Mg. Sc por su sabia dirección y excelente asesoría durante todo el desarrollo de este.

Manuel Stalin Armijos Ordóñez

Índice de contenidos

Índice de tablas:

Índice de figuras:

Índice de anexos:

1. Título

Diseño de una arquitectura basada en microservicios en la nube para mejorar la escalabilidad y agilidad del aplicativo Ktaxi de la empresa Kradac Cia. Ltda.

2. Resumen

En la última década, se ha producido un incremento en el uso de las arquitecturas de microservicio, motivado por diversos factores, entre los que destacan la necesidad de lograr una mayor escalabilidad y flexibilidad, así como la posibilidad de implementar tecnologías más avanzadas. Además, las arquitecturas de microservicio permiten a los desarrolladores construir y desplegar servicios de forma independiente, lo que favorece la colaboración y mejora continua en el proceso de desarrollo. Por otra parte, la implementación de microservicios también proporciona una mayor facilidad de mantenimiento y actualización de las aplicaciones, al centrarse cada servicio en una tarea específica.

El presente trabajo de titulación (TT) se desarrolló en base a la información de la empresa Kradac Cia. Ltda. y como caso de estudio se realizó un prototipo funcional, para ello, se obtuvo información de la arquitectura actual del aplicativo Ktaxi dando como resultado el diseño de una arquitectura monolítica que permitió conocer los componentes que se deben migrar a una nueva arquitectura de microservicios. A sí mismo conocer el flujo del funcionamiento del aplicativo. Para la elección de los componentes que se utilizaron en la arquitectura de microservicios, se realizó una investigación bibliográfica que permitió identificar los componentes básicos que se necesitan tal como: API Gateway, contenedores, comunicación, almacenamiento de información, microservicios, clientes y socket, y para el diseño final de la arquitectura se los incorporó en un diagrama de componentes en donde se detalla la integración de cada uno de ellos junto a un prototipo desarrollado con Node.js.

De esta manera se pudo determinar que, la arquitectura diseñada y propuesta en el presente TT mejorará la escalabilidad y agilidad del aplicativo Ktaxi dado que en pruebas iniciales del prototipo y socialización con la empresa Kradac Cia Ltda. se obtuvo muy buenos resultado, además porque cada servicio se enfoca en una sola tarea y puede ser escalado de forma específica a diferencia de la arquitectura monolítica donde escalar una funcionalidad implica escalar toda la aplicación, también por el uso de tecnologías de contenedores que permitió una rápida implementación de los servicios así como la independencia en el desarrollo, despliegue y mantenimiento puesto que los equipos de desarrollo pueden trabajar en paralelo en diferentes servicios ,realizar actualizaciones y cambios de forma más rápida y frecuente.

Palabras clave: *arquitectura, microservicios, API Gateway, Docker, RabbitMQ, SCRUM*

2.1 Abstract

In the last decade, there has been an increase in the use of microservice architectures, motivated by several factors, including the need for greater scalability and flexibility, as well as the possibility of implementing more advanced technologies. In addition, microservice architectures allow developers to build and deploy services independently, which favors collaboration and continuous improvement in the development process. On the other hand, the implementation of microservices also provides greater ease of maintenance and updating of applications, as each service focuses on a specific task.

This degree project was developed based on the information of the company Kradac Cia. Ltda. and as a case study a functional prototype was made, for this, information was obtained from the current architecture of the Ktaxi application resulting in the design of a monolithic architecture that allowed to know the components that should be migrated to a new microservices architecture. At the same time, the flow of the application's operation was known. In order to choose the components that were used in the microservices architecture, a bibliographic research was carried out to identify the basic components needed such as: API Gateway, containers, communication, information storage, microservices, clients and socket, and for the final design of the architecture they were incorporated in a diagram of components where the integration of each one of them is detailed together with a prototype developed with Node.js.

In this way it was determined that the architecture designed and proposed in this degree project will improve the scalability and agility of the Ktaxi application since in initial tests of the prototype and socialization with the company Kradac Cia. Ltda. was obtained very good results, also because each service is focused on a single task and can be scaled specifically unlike the monolithic architecture where scaling a functionality involves scaling the entire application, also by the use of container technologies that allowed a rapid implementation of services and independence in the development, deployment and maintenance since the development teams can work in parallel on different services, make updates and changes more quickly and frequently.

Keywords: *architecture, microservices, API Gateway, Docker, RabbitMQ, SCRUM*

3. Introducción

Según Blinowski, Ojdowska y Przybylek [1], la evolución de las arquitecturas de software ha sido impulsada por la necesidad de lograr una mejor capacidad de descomponer y organizar sistemas en módulos lógicamente cohesivos y vagamente acoplados que ocultan su implementación entre sí y presentan servicios a través de interfaces bien definidas. Desde su proclamación en 2012, las arquitecturas basadas en microservicios se han vuelto cada vez más populares debido a las ventajas que ofrecen, tales como la mejora de la disponibilidad, la tolerancia a fallos y la escalabilidad horizontal. Además, también han mejorado la agilidad de desarrollo de software, permitiendo una mayor independencia en el desarrollo y despliegue de servicios, cambios, automatización de tareas de mantenimiento, actualizaciones más rápidas y frecuentes y el uso de tecnologías de contenedores para una implementación más rápida y consistente.

Teniendo presente esta evolución en las arquitecturas de microservicios el presenta TT está encaminado a ofrecer un diseño basada en microservicios en la nube, mediante la integración de un conjunto de componentes bien definidos plasmados en un prototipo funcional, que permita mejorar algunos aspectos importantes como la escalabilidad, mantenimiento, agilidad, tolerancia a fallos entre otros.

Para ello se planteó como objetivo principal “Diseñar una arquitectura basada en microservicios en la nube para el aplicativo Ktaxi de la empresa Kradac Cia. Ltda. de la ciudad de Loja” y para cumplir con este, se definió 3 objetivos específicos los cuales son: “Realizar un análisis del estado actual del aplicativo Ktaxi mediante un diagnóstico interno en base a la información de la empresa Kradac Cia. Ltda., para poder determinar su arquitectura actual”, “Identificar los principales componentes que intervienen en el diseño de una arquitectura basada en microservicios en la nube” y “Diseñar una arquitectura basada en microservicios en la nube para el aplicativo Ktaxi”.

Para abordar lo antes mencionado, se realizó un análisis de información de la empresa Kradac Cia. Ltda. lo cual permitió el diseño de su arquitectura monolítica actual, en base a los componentes previamente identificados, permitiéndonos cumplir con el primer objetivo. Para dar cumplimiento al segundo objetivo se realizó una revisión bibliográfica de información obteniendo como resultado los componentes principales a utilizar, su función e implementación dentro de la arquitectura de microservicios. Para el cumplimiento del tercer objetivo se diseñó

una arquitectura que integra todos los componentes identificados en el objetivo anterior, de igual manera se desarrolló un prototipo que permitió validar el funcionamiento de la arquitectura propuesta, junto al departamento de Tecnologías de la información (TI) de la empresa Kradac Cia. Ltda.

Por otro parte, el presente documento se encuentra estructurado de la siguiente manera:

En la sección Marco teórico, se elaboraron nueve capítulos del área de estudio que ayudaron a sustentar los conocimientos aplicados en la ejecución del presente TT. La sección de Metodología, permite detallar el contexto y procedimiento para el desarrollo del presente TT. La sección Resultados, sirve para presentar todos los datos relevantes obtenidos en la ejecución del TT, para el primer objetivo se utiliza la información proporcionada por la empresa Kradac Cia. Ltda., para el segundo objetivo se identifican los componentes principales para el desarrollo de la arquitectura, así como su función e integración, para el tercer objetivo se realiza el diseño de todos los componentes seleccionados en el punto anterior y para su validación se desarrolla e implementa un prototipo funcional. En la sección Discusión, se realiza un análisis de los resultados obtenidos y como se cumple con los objetivos. La sección Conclusiones, permite expresar las ideas más relevantes rescatadas luego de haber culminado el presente TT. Finalmente, en la sección Recomendaciones, se plantean aspectos a considerar para el desarrollo de trabajos futuros.

4. Marco teórico

4.1 Arquitectura Monolítica

En una arquitectura monolítica, toda la funcionalidad está encapsulada en una sola aplicación, por lo que sus módulos no se pueden ejecutar independientemente. Este tipo de arquitectura está estrechamente acoplada, y toda su lógica para manejar una solicitud se ejecuta en un solo proceso [2]. Su estructura interna consiste en una torre o monolito donde las partes de la aplicación se apilan una encima de otra. Desarrollar todo un sistema siguiendo una arquitectura por capas se puede considerar una arquitectura monolítica porque los componentes que lo constituyen no se pueden desplegar de forma independiente. Por su sencillez, se pueden emplear en sistemas simples [3].

4.1.1 Desventajas de la arquitectura monolítica

Entre sus principales desventajas podemos citar las siguientes en base a la información de [3]:

- En las aplicaciones monolíticas aumenta la complejidad conforme aumenta su tamaño. Como consecuencia, el sistema se hace más difícil de evolucionar y mantener.
- Un sistema monolítico tiene limitada su escalabilidad, que principalmente se consigue replicando toda la aplicación y repartiendo las peticiones que recibe entre las diferentes réplicas. De esta forma, se están escalando componentes del sistema que no requieren hacerlo.
- Los sistemas monolíticos limitan el uso de lenguajes y herramientas que se pueden emplear. Esto puede suponer que solo se pueda hacer uso de los lenguajes y herramientas que se utilizan desde el inicio de su construcción.
- Actualizar una dependencia en un monolito puede conllevar efectos secundarios indeseados. Esto se debe a que todos los componentes que lo forman han de hacer uso de la misma versión de una librería.

4.2 Arquitectura de Microservicios

Una arquitectura de microservicios, o simplemente "microservicios", es un método de arquitectura que se basa en una serie de servicios con capacidades de negocio y que se pueden implementar de forma independiente. Estos servicios tienen su propia lógica empresarial y base de datos con un objetivo específico. La actualización, las pruebas, la implementación y el

escalado se llevan a cabo dentro de cada servicio. Los microservicios desacoplan los principales intereses específicos de dominios empresariales en bases de código independientes. Los microservicios no reducen la complejidad, pero hacen que cualquier complejidad sea visible y más gestionable, ya que separan las tareas en procesos más pequeños que funcionan de manera independiente entre sí y contribuyen al conjunto global [4] [5].

En la Fig. 1, se muestra el ejemplo del diseño de una arquitectura basada en microservicios, donde se puede observar que un cliente puede acceder a tres microservicios definidos como payment, shopping card e inventory.

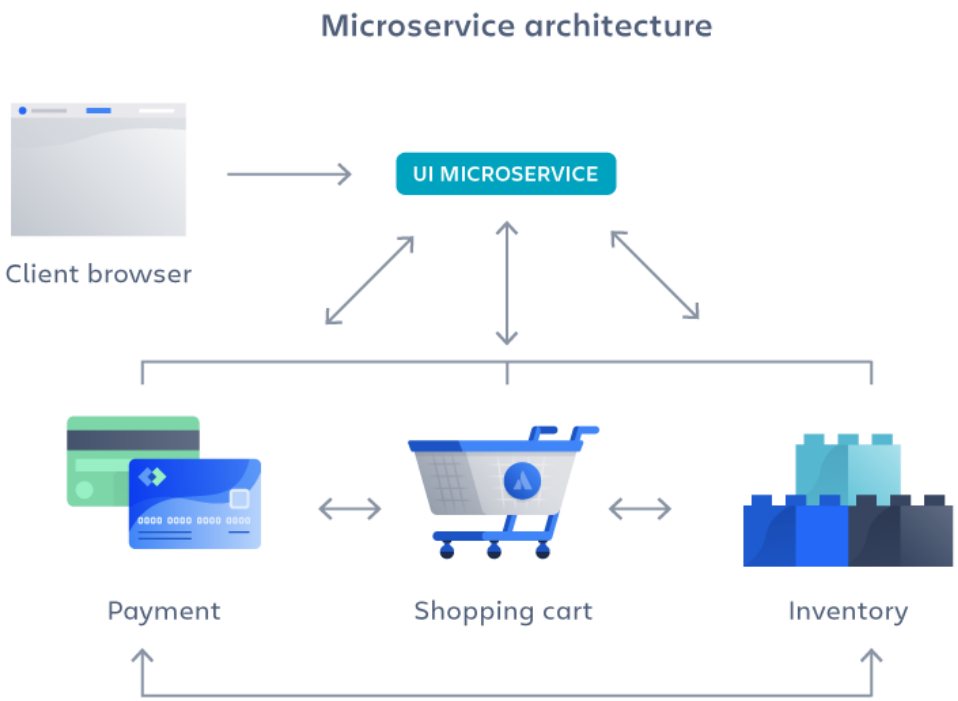

Fig. 1. Ejemplo de una arquitectura basada en microservicios.

4.2.1 Ventajas de los microservicios

Las ventajas de los microservicios son significativas y pueden ayudar a los equipos de desarrollo a crear aplicaciones más escalables, flexibles y fáciles de mantener. Algunas de las principales ventajas son las siguientes [4]:

- **Agilidad:** Promueve formas ágiles de trabajar con equipos pequeños que implementen con frecuencia.

- **Escalado flexible:** Si un microservicio está alcanzando su capacidad de carga, se pueden implementar rápidamente nuevas instancias de ese servicio al clúster que lo acompaña para aliviar la presión. Ahora se tiene varios inquilinos y no se tiene control de estado, con clientes repartidos en varias instancias. Así, se puede respaldar tamaños de instancias mucho mayores.
- **Implementación continua:** Se tiene ciclos de lanzamiento frecuentes y más rápidos. Antes se enviaban actualizaciones una vez a la semana y ahora se puede hacerlo dos o tres veces al día.
- **Muy fácil de mantener y probar:** Los equipos pueden hacer pruebas con el código y dar marcha atrás si algo no funciona como esperan. Esto facilita la actualización del código y acelera el tiempo de salida al mercado de nuevas funciones. Además, es fácil aislar y corregir fallos y errores en los servicios individuales.
- **Implementación independiente:** Los microservicios son unidades individuales, por lo que permiten una implementación independiente rápida y sencilla de funciones individuales.
- **Flexibilidad tecnológica:** Con las arquitecturas de microservicios, los equipos pueden elegir con libertad las herramientas que desean.
- **Alta fiabilidad:** puedes implementar cambios para un servicio en concreto sin el riesgo de que se caiga toda la aplicación.
- **Equipos más satisfechos:** Los equipos de Atlassian que trabajan con microservicios están mucho más satisfechos, ya que son más autónomos y pueden compilar e implementar de forma independiente, sin esperar semanas a que se apruebe una solicitud de incorporación de cambios.

4.3 **Arquitectura monolítica vs arquitectura de microservicios**

En la Fig. 2, se presentan algunas diferencias entre una arquitectura monolítica y una arquitectura de microservicios en base a la información de Daniel López en su paper descrito en [6].

Categoría	Arquitectura Monolítica	Arquitectura de microservicios
Código	Una base de código única para toda la aplicación.	Múltiples bases de código. Cada microservicio tiene su propia base de código.
Comprensibilidad	A menudo confuso y difícil de mantener.	Mayor facilidad de lectura y mucho más fácil de mantener.
Despliegue	Implementaciones complejas con ventanas de mantenimiento y paradas programadas.	Despliegue sencillo ya que cada microservicio se puede implementar de forma individual, con un tiempo de inactividad mínimo, si no es cero.
Lenguaje	Típicamente totalmente desarrollado en un lenguaje de programación.	Cada microservicio puede desarrollarse en un lenguaje de programación diferente.
Escalamiento	Requiere escalar la aplicación entera, aunque los cuellos de botella estén localizados.	Cada microservicio puede desarrollarse en un lenguaje de programación diferente.

Fig. 2. Comparativa de una arquitectura monolítica y de microservicios

4.4 Comunicación entre microservicios

4.4.1 RabbitMQ

RabbitMQ es una solución gratuita de código abierto que sirve como intermediario de mensajes a través del protocolo AMQP (Advanced Message Queuing Protocol). Es altamente accesible, tolerante a fallas y escalable. En las aplicaciones de microservicios actuales, RabbitMQ se utiliza como intermediario entre microservicios individuales, lo que les permite comunicarse entre sí sin preocuparse por la pérdida de mensajes. Al igual que las abejas, cada equipo de desarrollo crea un componente separado utilizando una de las tecnologías. Cada uno de los componentes individuales forma una estructura sólida de la llamada colmena de microservicios y desempeña un papel separado en el sistema [7].

4.4.2 Agente de mensajes RabbitMQ

RabbitMQ es un intermediario de mensajes ligero y escalable basado en AMQP [12]. RabbitMQ actúa como intermediario entre los distintos servicios. Se utiliza para reducir la carga y el tiempo de entrega en las aplicaciones web del servidor al delegar tareas que normalmente requerirían mucho tiempo y recursos. En la Fig. 3, se puede observar que el trabajo de los comparadores de mensajes, como RabbitMQ, es garantizar que los mensajes de

los editores lleguen a los consumidores correctos. Para hacer esto, el corredor utiliza dos componentes clave, a saber, Exchange y Queue [7].

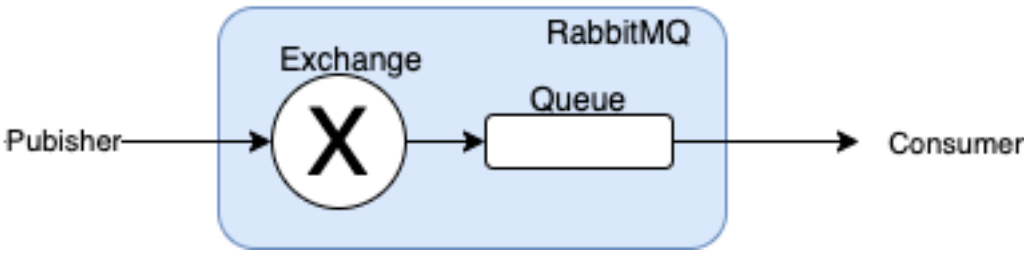

Fig. 3. Principio de funcionamiento del intermediario RabbitMQ

Se ejemplifica tomando como referencia la Fig. 1 del artículo "Microservice development using RabbitMQ message broker" [7].

4.4.2.1 Tipos de intercambio en RabbitMQ

El protocolo AMQP admite múltiples tipos de intercambios. En lugar de reenviar mensajes directamente a las colas, los editores reenvían mensajes a los intercambios. El intercambio supervisa la redirección de mensajes a diferentes colas a través de enlaces y claves de enrutamiento. La vinculación es el vínculo entre el pedido y el intercambio. El protocolo AMQP define los siguientes tipos de intercambios [7]:

- **Direct**: Un intercambio directo que reenvía el mensaje a una cola cuya clave de enrutamiento coincide con la clave de vinculación.
- **Topic**: Es similar al intercambio directo, pero incluye una plantilla. Las colas que cumplen la condición de la plantilla reciben un mensaje.
- **Fanout**: Los mensajes se reenvían a todas las colas.
- **Headers**: Intercambio de encabezados. Los mensajes se pasan en las colas por encabezado y la clave de enrutamiento se ignora

4.5 Exponer públicamente microservicios

La capa API Gateway es la interfaz para exponer públicamente microservicios al exterior, distribuyendo información que provee un microservicio a clientes que solicitan consumirla mediante una petición (aplicaciones y/o microservicios de terceros o implementados por la misma organización) como se puede observar en la Fig. 4. Además, éstos tienen que presentar su identidad ante la API Gateway mediante el inicio único de sesión (Single Sign-On, SSO) u OAuth, y ésta se encargará de conceder o denegar el acceso para poder consumir los microservicios expuestos. La implementación de la API Gateway debe asignar la identidad

de los clientes a un protocolo real, como una URL en una API REST o CoAP, una cola o intercambio en AMQP, o un topic en MQTT [8].

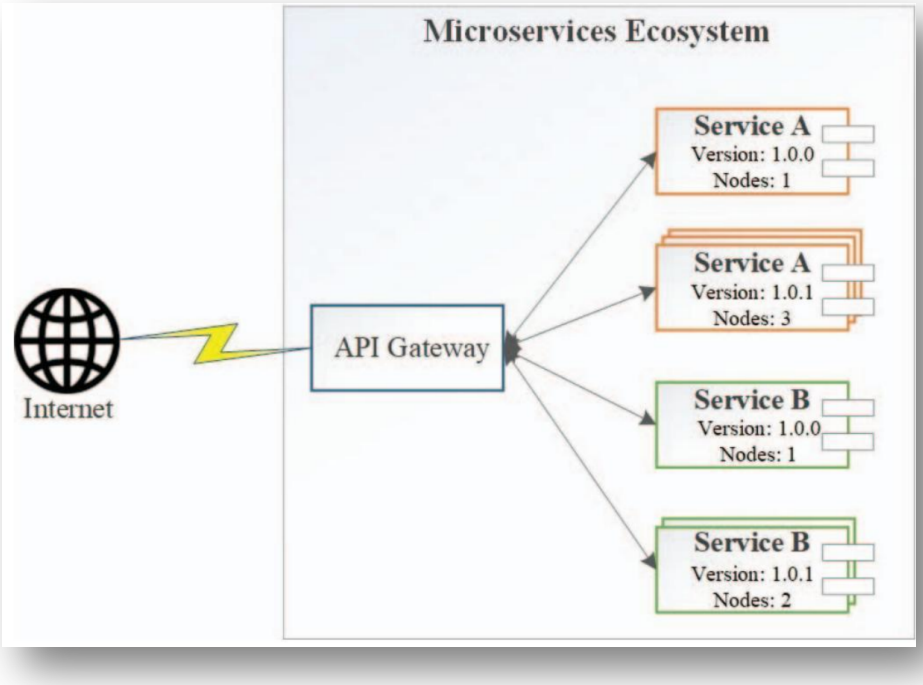

Fig. 4. Api Gateway como puerta de enlace a los microservicios.

Se ejemplifica tomando como referencia la Fig. 1 del artículo "Software Versioning with Microservices through the API Gateway Design Pattern" [9].

4.6 Contenedores

4.6.1 Docker

Unidad de software que permite empaquetar el código y todas las dependencias de manera que la aplicación pueda ejecutar de manera rápida y confiable en cualquier ambiente. A este "paquete" generado se le denomina imagen y tiene la particularidad de ser ligero, independiente e incluye todo lo necesario para poder ejecutar una aplicación: código, herramientas del sistema, librerías, configuración y ambiente de ejecución [10].

A su vez, se almacenan aplicaciones con el fin de que queden desacopladas del sistema operativo en el cual ejecuta, esto permite que una aplicación alojada en un contenedor pueda ser transportada y ejecutada independientemente del Sistema Operativo. Para lograrlo, los contenedores suelen contener aplicaciones conjuntamente con todas sus dependencias (como librerías y frameworks), por lo cual en definitiva un contenedor está compuesto por [10]:

- La aplicación.

- Todas sus dependencias, librerías y demás archivos binarios.
- Archivos de configuración que son necesarios para ejecutarlo

4.6.2 Kubernetes

Plataforma de orquestación de contenedores de nivel de producción que automatiza la gestión de aplicaciones nativas de la nube en infraestructuras informáticas a gran escala. Kubernetes orquesta la colocación y ejecución de contenedores de aplicaciones dentro y entre clústeres de nodos. Un clúster de Kubernetes consta de un plano de control y un conjunto de nodos trabajadores. El plano de control es ejecutado por diferentes componentes que se ejecutan dentro de un nodo maestro. Los nodos trabajadores son responsables de la ejecución de cargas de trabajo de aplicaciones en contenedores. En particular, las cargas de trabajo de las aplicaciones consisten en Pods, que a su vez contienen uno o más contenedores. Un pod representa la unidad mínima de implementación en Kubernetes [11].

4.7 Metodologías de desarrollo ágil

Los usuarios potenciales se involucran en las pruebas al producto desde etapas tempranas del proyecto, buscando resultados en menor tiempo sin disminuir su calidad. El cliente indica lo que espera a medida que recibe entregas tempranas y constantes, lo cual requiere su compromiso con los desarrolladores. Los principios de estas metodologías se agrupan en el "Manifesto for Agile Software Development", siendo ellos: satisfacción del cliente, adaptación al cambio, entregables de software, trabajo en equipo, motivación, diálogo, software funcional, desarrollo sostenible, atención continua, simplicidad, organización y efectividad. A continuación, se describen las características de las metodologías ágiles más referenciadas en la literatura [12].

4.7.1 Metodología Scrum

Es la metodología ágil más utilizada, diseñada para lograr la colaboración eficaz de los diferentes equipos relacionados con el proyecto. Se realizan sprints o entregas iterativas del producto al cliente para que lo pruebe y realice observaciones. Deben efectuarse reuniones diarias de máximo 15 minutos con el equipo de trabajo para coordinar el proyecto debidamente, y asignar roles claros a los miembros del equipo desarrollador. En la revisión bibliográfica realizada por Vallon et al., ocho de las diez metodologías ágiles más usadas en el mundo en el período 2010–2016 se basan en el modelo scrum [12].

4.7.2 Programación extrema (XP)

La metodología se basa en cinco valores: simplicidad, comunicación, respeto y coraje para alcanzar retroalimentación rápida, simplicidad, cambio [12].

4.7.3 Proceso unificado ágil (AUP)

El AUP aplica técnicas ágiles incluyendo desarrollo dirigido por pruebas (TDD por sus siglas en inglés), modelado ágil, gestión ágil de cambios, y refactorización de base de datos para mejorar la productividad. El ciclo de vida del AUP tiene cuatro fases: concepción, donde se define el alcance y las arquitecturas para el software, elaboración en donde el equipo de desarrollo comprende los requisitos y valida la arquitectura del sistema, construcción en la cual el sistema es desarrollado y finalmente se despliega [12].

4.7.4 Proceso unificado abierto (OpenUP)

Esta metodología se caracteriza por el desarrollo incremental, uso de casos de uso y escenarios, manejo de riesgos, y diseño basado en la arquitectura [12].

4.8 Tecnologías para el desarrollo backend

El desarrollo de servicios web se ha convertido en un estándar en la programación al momento de compartir información entre diferentes aplicaciones de software, ya que para distribuir sus funcionalidades estos son independientes del hardware, el sistema operativo, el lenguaje de programación y al igual que son independientes para proveer las funcionalidades, son independientes al momento de ejecutarse y desarrollarse para su uso [13].

4.8.1 REST

Los servicios web basados en la arquitectura REST (Representational State Transfer), son una alternativa más simple a SOAP y a WSDL, la cual transmite los datos sobre el protocolo estandarizado HTTP. Varias empresas como Google, Facebook y Yahoo! son casos de éxito al migrar sus servicios a esta tecnología. REST fue publicado en el año 2000 por Roy Fielding en la Universidad de California en una conferencia en la cual presentaba acerca de principios arquitectónicos de software para usar a la web como una plataforma de procesamiento distribuido. El cliente tiene acceso a los recursos usando una única URI y una representación del recurso es devuelto con un estado de la transferencia de la información. REST no está

estrictamente relacionado con el protocolo HTTP, pero es con el que más comúnmente se asocia. Los servicios web basados en REST cumplen los siguientes principios [13].:

- Utiliza métodos de HTTP de manera explícita: Cada método tiene una función específica: GET su uso es para obtener un recurso desde el servidor, POST se usa para crear un nuevo recurso en él servidor, PUT actualiza o cambia el estado de un recurso, DELETE elimina un recurso.
- REST no mantiene estado: Es necesario tener un escalamiento para cubrir la demanda constante y en crecimiento mediante balanceador, clúster y servidores con alta disponibilidad con el objetivo de distribuir las peticiones entre los equipos para disminuir el tiempo de respuesta.
- URI en formato de directorios: La estructura de los URI se la crea de una manera similar a directorios que es fácil de entender lo cual permite poder utilizarlos de manera intuitiva.
- Representaciones: Los servicios web REST emiten una respuesta en formato JSON o XML los cuales pueden ser usados por cualquier aplicación desarrollada en cualquier lenguaje.

Existen diferentes herramientas que se pueden utilizar al momento de desarrollar un servicio REST, lo importante es tener en claro en que plataforma se va a desarrollar, si será un sistema distribuido, lo más importante es emplear tecnologías de similares características que tengan compatibilidad en el sistema distribuido. Este artículo se centra en dos importantes tecnologías JAVA y JAVASCRIPT, en el caso de JAVA nos vamos a centrar en Spring Boot y para JAVASCRPIT usaremos Node.js.

4.8.2 Spring Boot

Es un framework Java basado en el Modelo Vista Controlador, mediante el cual gracias a los componentes y librerías que brinda hace fácil el desarrollo y despliegue de los servicios REST. Ha eliminado la necesidad de configurar la aplicación con el uso de archivos XML haciendo énfasis en el desarrollo de la misma [13].

Como las principales características de Spring Boot tenemos que nos permite crear aplicaciones de Spring independientes, al poseer servidores de aplicación embebidos como Tomcat, Jetty o Undertow, no es necesario generar un WAR para su ejecución, también permite configurar automáticamente bibliotecas de Spring y de terceros, no es necesario realizar una configuración

en archivos XML, por lo que es más fácil y rápido la integración con otros proyectos de Spring [13].

4.8.3 Node.js

Node es un entorno de ejecución orientado a eventos asíncronos de JavaScript diseñado para construir aplicaciones de red escalables. Como las principales características de Node.js tenemos que permite desarrollar aplicaciones de red escalables, impulsada por eventos asíncronos, si no hay trabajo por hacer Node.js permanece dormido hasta que haya una conexión para que se active una llamada al servicio, todo esto es posible ya que se usa JavaScript del lado de servidor [13].

4.9 DevOps

DevOps esta destinada a salvar la brecha entre Desarrollo y Operaciones enfatizando comunicación y colaboración, integración continua, calidad aseguramiento y entrega con implementación automatizada utilizando un conjunto de prácticas de desarrollo. DevOps es un concepto reciente y como tal, no tiene consenso su definición. Se han proporcionado varias definiciones de DevOps en la literatura. Un buen número de las definiciones sugieren que el término enfatiza la colaboración entre el desarrollo y operaciones. Perera et al. Describe a DevOps como diferentes formas a través de las cuales los desarrolladores y operaciones discuten y participan en debates colectivos a fin de producir software y prestar servicios de forma más rápida, fiable y con mayor calidad. Todas las definiciones anteriores describen DevOps, sin embargo, estos se limitan al desarrollo y operación de software [14].

5. Metodología

Para llevar a cabo correctamente el desarrollo del TT, en la fase de pre análisis se utilizó el entorno de la empresa Kradac Cia. Ltda. de la ciudad de Loja, donde se solicitó acceso a la información técnica y operativa como se observa en el ***Anexo 1.*** *Solicitud de acceso a la información del aplicativo Ktaxi,* así como un asesor como se observa en el ***Anexo 2.*** *Solicitud para la asignación de un asesor técnico para determinar la arquitectura de Ktaxi*, una vez obtenido el acceso a la información proporcionada por ellos sobre el aplicativo Ktaxi y la asignación de un asesor para determinar su arquitectura como se observa en el ***Anexo 3.*** *Aprobación de acceso a la información de Ktaxi y asignación de un asesor para determinar su arquitectura*, se logró definir la arquitectura actual y los componentes que la conforman, en la fase de análisis se analizó los componentes que intervienen en una arquitectura en base a información bibliográfica logrando determinar el uso de 7 componentes fundamentales para la arquitectura propuesta, en la fase de diseño se generó todos los diagramas de los componentes antes identificados, logrando detallar su funcionamiento e implementación dentro de la arquitectura y en la fase de prototipado y pruebas se utilizó un ambiente simulado local, en donde se implementó el prototipo propuesto en base al diseño generado para su correcta validación y funcionamiento, posteriormente se solicitó permiso para socializar el diseño y el prototipo como se puede observar en el ***Anexo 4.*** *Solicitud de socialización del diseño de la arquitectura de microservicios*, en el entorno del departamento de TI de la empresa Kradac Cia. Ltda., los cuales son los encargados de la parte de desarrollo de software de las soluciones de la empresa Kradac Cia. Ltda.

Para el Diseño de una arquitectura basada en microservicios en la nube para mejorar la escalabilidad y agilidad del aplicativo Ktaxi de la empresa Kradac Cia. Ltda., se estableció una metodología de desarrollo propia adaptable a la naturaleza del presente TT, en la cual se incorporan varias áreas de estudio: arquitectura de microservicios, contenedores, API Gateway, base de datos, comunicación en tiempo real, programación, análisis y diseño de sistemas. Misma que se define en 5 fases, correspondientes a cada uno de los objetivos.

5.1 Fase 1: Pre análisis

En esta fase, se realizó un pre análisis de la información proporcionada por la empresa Kradac Cia. Ltda., y junto al asesor asignado Ing. Ricardo Jumbo se definieron los aspectos técnicos de su implementación, mediante la cual se pudo identificar, evaluar y verificar como se puede observar en el ***Anexo 5.*** *Aprobación de la arquitectura actual del aplicativo Ktaxi*, la

arquitectura monolítica actual que mantiene el aplicativo, que determinaron los componentes, clientes, tecnologías e integraciones que utilizan, los cuales fueron validados y los cuales se deben tener en cuenta para el desarrollo del TT, mismos que se pueden visualizar en la FASE 1: Realizar un análisis del estado actual del aplicativo Ktaxi mediante un diagnóstico interno en base a la información de la empresa Kradac Cia. Ltda., para poder determinar su arquitectura actual.

5.2 **Fase 2: Análisis**

En esta fase, se identificó mediante una revisión bibliográfica de información los componentes fundamentales para la arquitectura de microservicios, se estableció el uso de 7 componentes bien definidos dentro de la arquitectura, así mismo permitió conocer su función y uso principal para la generación de los diagramas de implementación necesarios para la siguiente fase.

Finalizando la fase de análisis se expuso algunos diagramas generales indicando el proceso de integración de cada componente seleccionado así como las tecnologías, protocolos y funciones necesarias para su correcta implementación, la cual se desarrolla en la FASE 2: Analizar los principales componentes que intervienen en el diseño de una arquitectura basada en microservicios en la nube mediante una revisión bibliográfica de información para identificar los componentes a utilizar.

5.3 **Fase 3: Diseño, prototipo y pruebas**

En esta fase, se diseñó los diagramas de componentes generales, integración, comunicación y seguridad para la construcción de la arquitectura de microservicios, en base a estos diseños para el desarrollo del prototipo se utilizó la metodología de desarrollo ágil SCRUM la cual permitió dividir el desarrollo del prototipo en 7 sprints pequeños cada uno de ellos enfocado a un componente previamente seleccionado detallados a continuación:

- **Sprint 1:** Desarrollo del componente API Gateway
- **Sprint 2:** Desarrollo del componente para la comunicación entre microservicios con RabbitMQ
- **Sprint 3:** Desarrollo del componente para contenedor el desarrollo de los microservicios con Docker
- **Sprint 4:** Desarrollo del componente para el almacenamiento y gestión de la información

- **Sprint 5:** Desarrollo del componente para los microservicios propiamente dichos
- **Sprint 6:** Desarrollo del componente para la comunicación en tiempo real con Socket.IO
- **Sprint 7:** Desarrollo del componente cliente para el consumo de los microservicios

Cada uno de ellos integrados en el prototipo funcional que además incorporó mecanismos para garantizar la disponibilidad y monitoreo de su ejecución. Las pruebas se las realizó en una socialización autorizada como se puede ver en el **Anexo *6*.** Aprobación para la socialización de la arquitectura de microservicios, en el departamento de TI de la empresa Kradac Cia. Ltda., junto al equipo de desarrollo de software encargado del desarrollo backend, obteniendo como resultado una aceptación y validación satisfactorios, mismos que se pueden visualizar en la FASE 3: Diseñar una arquitectura basada en microservicios en la nube para el aplicativo Ktaxi.

6. Resultados

6.1 FASE 1: Realizar un análisis del estado actual del aplicativo Ktaxi mediante un diagnóstico interno en base a la información de la empresa Kradac Cia. Ltda., para poder determinar su arquitectura actual.

6.1.1 Análisis general del aplicativo Ktaxi

Para el análisis del aplicativo Ktaxi se utilizó el **Anexo *7.*** Descripción general de Clipp Taxi, proporcionado por la empresa Kradac Cia. Ltda., donde se pudo determinar que el aplicativo Ktaxi o también denominado Clipp Taxi es un sistema completo de administración y gestión de despacho de unidades de taxi, que permite acoplarse a las diferentes formas de trabajo de las operadoras de taxi en cada ciudad. Cuenta con un aplicativo móvil para usuarios conductores denominado Ktaxi conductor, clientes denominado Ktaxi cliente y diferentes periféricos o canales de solicitud como: sistemas web, quioscos, bots, redes sociales, CallCenter - radio taxi, entre otros.

En la Fig. 5, se pueden observar los componentes principales del aplicativo Ktaxi cliente y conductor y cuales son los componentes principales de cada uno de ellos.

Fig. 5. Componentes del aplicativo Ktaxi cliente y conductor

El aplicativo Ktaxi consta de 5 módulos principales los cuales se pueden observar en la Fig. 6 y que se describen a continuación:

Fig. 6. Módulos principales del sistema Clipp taxi

- **Tecnología Cliente**: El aplicativo denominado Ktaxi Cliente es la solución brindada por la empresa Kradac Cia. Ltda., para solicitar el servicio de taxi dentro y fuera del país, mismo que ofrece alternativas para varios sistemas operativos, dispositivos u otros sistemas que el usuario posea o desee utilizar como: Aplicativo móvil para Android, iOS, plataforma web, sistema corporativo o Bot de Messenger.

- **Tecnología Conductor**: El aplicativo denominado Ktaxi Conductor es la solución brindada por la empresa Kradac Cia. Ltda., para que el conductor pueda brindar su servicio requerido por los clientes fuera y dentro del país, este se encuentra disponible para varios sistemas operativos como: Aplicativo móvil para Android, HarmonyOS e iOS.

- **Tecnología CallCenter:** Solución para brindar el servicio de taxi desde la central dentro y fuera del país esta solución es multiplataforma consta de algunos servicios como: Software de despacho de carreras, robot charlie ENV atención automatizada de llamadas, central telefónica virtual CallCenter y sistema web de administración, gestión y monitoreo.

- **Sistema de monitoreo:** Permite monitorear toda la operación de los diferentes componentes del sistema, dicho monitoreo se realiza mediante la creación de usuarios con sus respectivas contraseñas de seguridad.

- **Sistema de Gestión de Administración y Gestión de Taxis:** Todo el sistema para que el administrador tenga el control total del mismo, en este se puede visualizar las solicitudes realizadas de taxis, información de usuario y conductores registrado en los aplicativos antes mencionados y todo lo relacionado a la gestión de la información del sistema.

6.1.2 Componentes del aplicativo Ktaxi

Según lo establecido en el **Anexo *7.*** Descripción general de Clipp Taxi, Ktaxi se consolida en una plataforma integral para el trabajo de operadoras de taxi, que consta de múltiples componentes que se integran en su aplicación los cuales se pueden observar en la Fig. 7, dichos componentes se conforman en cada uno de los módulos mencionados en la Fig. 6.

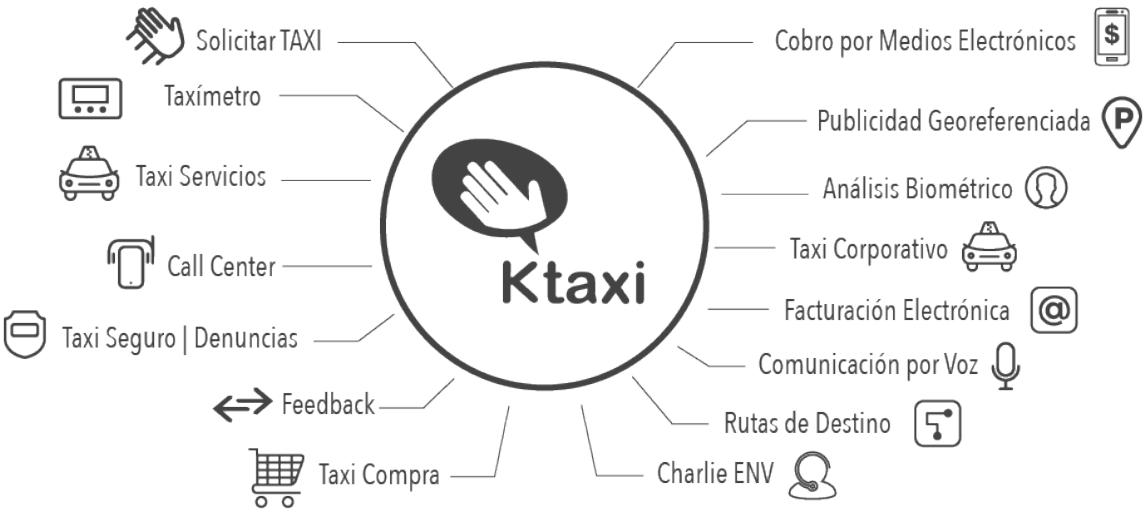

Fig. 7. Componentes del sistema aplicativo Ktaxi

A continuación, en la TABLA I., se describe cada componente de la Fig. 7, haciendo énfasis en la funcionalidad principal que cada uno desarrolla dentro del aplicativo Ktaxi.

TABLA I.

DESCRIPCIÓN DE LOS COMPONENTES DE CLIPP TAXI

Componente	**Descripción**
Solicitar Taxi	Es la tecnología para que los clientes puedan solicitar una unidad de taxi, entre estos están: - Aplicativo nativo para Android e iOS - Portal web para solicitudes - Aplicativo para Kiosco - Dashbuttom - Bot de Messenger.
Taxímetro	Se cuenta con el desarrollo electrónico e informático para el taxímetro, mismo que respeta y se acopla a los requerimientos de telemetría que requieren las diferentes entidades de control en donde funciona el aplicativo.
Taxi servicios	Ofrece múltiples servicios como taxi eléctrico, ayuda mecánica, encomiendas, transporte de mascotas, transporte de bicicletas, aeropuerto, entre otras, que garantizan la disponibilidad del servicio para todo tipo de clientes.
CallCenter	Es un sistema que nos permite modernizar las centrales o CallCenter de las operadoras de radio taxi, el sistema de CallCenter se integra a todo el sistema de aplicativos móviles lo que permite ser una solución incluyente y no excluyente. Los componentes principales son: - Software de Despacho de carreras. - Robot Charlie, atención automatizada de llamadas. - Aplicativo móvil CallCenter (callerID). - Sincronizador de datos a la nube.

Taxi Seguro	Permite consultar los datos como números de operadora de taxi, fotos de vehículo y propietario de los mismos registrados en el sistema, ofreciendo seguridad y confianza a los clientes.
Taxi Compra	Servicio de compras a domicilio puerta a puerta.
Cobro por medios electrónicos	Se cuenta con diferentes formas de pago disponibles para que el cliente pueda elegir la que mayor le convenga. • **Código Promocional:** Se puede hacer sorteos de carreras o premiar a los clientes mediante un código promocional. • **Voucher:** Es un saldo tipo crédito que lo gestionan las empresas con sus trabajadores. • **Tarjetas de Crédito o débito:** Se tiene implementando pasarela de pagos para facilidad del usuario. • **Saldo Clipp:** Permite comprar crédito a través de entidades financieras, tarjeta de crédito o débito y usarse para el pago de los diferentes servicios disponibles. • **PayPhone:** Se tiene un convenio para realizar el cobro por medio de PayPhone. • **Efectivo**: Para satisfacción del cliente también se aceptan pagos en efectivo.
Publicidad Georreferenciada	El componente de publicidad ofrece diferentes módulos publicitarios tanto en la app cliente como conductor, con analística de datos para que los clientes que deseen pautar publicidad puedan medir resultados. Los módulos

	publicitarios son configurables es decir se pueden realizar cambios a satisfacción del cliente.
Taxi Corporativo	Taxi corporativo para su empresa o flota de vehículos.
Facturación Electrónica	Sistema de facturación electrónica integrado a nuestro sistema.
Rutas de destino	Cálculo del valor de una solicitud mediante puntos de origen y destino con la generación de rutas en mapa y valores aproximados de cobro.
Charlie	Se cuenta con un módulo que permite interactuar con los usuarios de forma automatizada sin interacción humana, los bots (programas informáticos) son capaces de interactuar con los usuarios que necesiten un taxi y lo pueden realizar desde Messenger de la aplicación Facebook.
Comunicación por voz	Los aplicativos y sistemas incorporan la comunicación mediante mensajes de voz, entre los clientes y los usuarios.

6.1.3 Arquitectura actual del aplicativo Ktaxi

La arquitectura actual del aplicativo Ktaxi en base a la información proporcionada por la empresa Kradac Cia. Ltda., y en base a la experiencia del asesor asignado **Ing. Ricardo Jumbo** se pudo concluir que es una arquitectura monolítica que está desarrollada en una misma unidad lógica que encapsula toda la funcionalidad de negocio como: funciones, APIs, recursos, métodos y demás artefactos, en un mismo servidor que comparten los mismo recursos y memoria como se puede visualizar en el diagrama que se realizó en la Fig. 8, haciéndolo un servicio autosuficiente que contiene todas las funcionalidades necesaria para realizar las tareas para la cual fue diseñado.

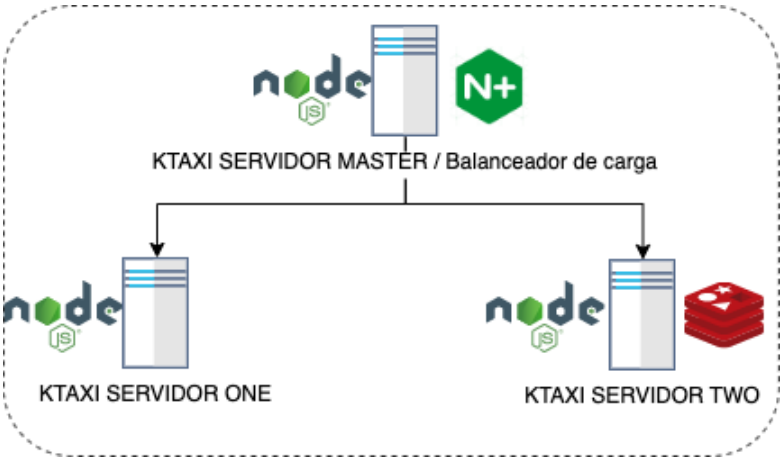

Fig. 8. Servidores monolíticos del aplicativo Ktaxi cliente y conductor

Los aplicativos Ktaxi cliente, conductor y CallCenter como se puede visualizar en la Fig. 9, se conectan a un único servidor principal que es el encargado de redirigir la solicitud para sí mismo o a otros 2 servidores que contienen la misma estructura lógica y que tienen la capacidad de atender la solicitud lo más rápido posible, y es aquí en donde radica toda la funcionalidad necesaria para que los aplicativos pueden funcionar y realizar las acciones requeridas por los usuarios o clientes.

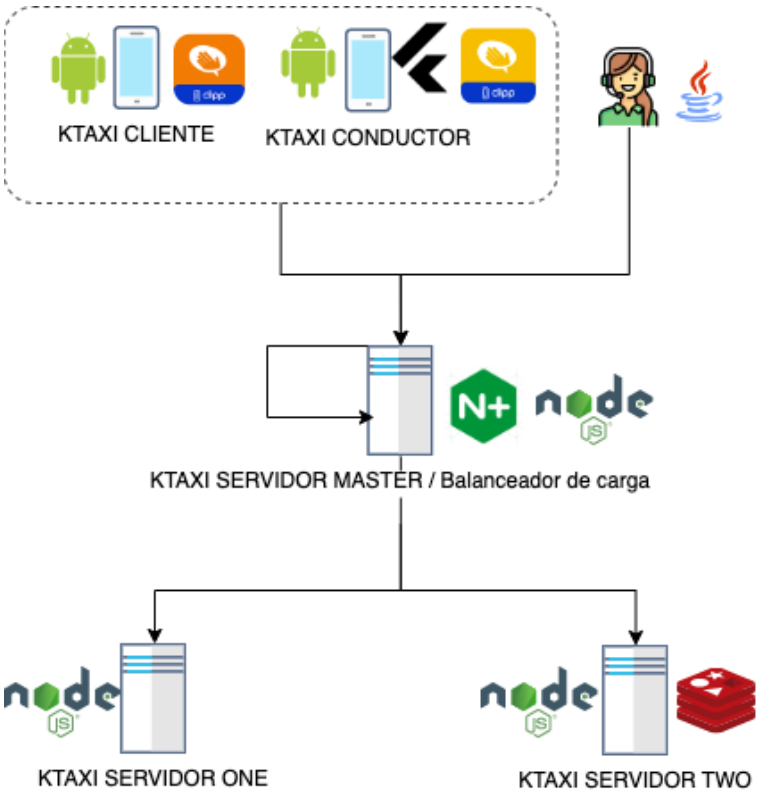

Fig. 9. Arquitectura monolítica del aplicativo Ktaxi cliente, conductor y CallCenter

En la Fig. 10, se detalla el diagrama que se realizó como los sistemas de administración y gestión de taxis así como el monitoreado de flotas se conectan a un único servidor web, que proporciona el servicio para que los administradores puedan gestionar la información relevante inherente a sus necesidades, así también conocer o visualizar en tiempo real la ubicación e información de los usuarios, dicho servidor se conecta al servidor principal del aplicativo Ktaxi para poder realizar operaciones cruzadas como un aplicativo más.

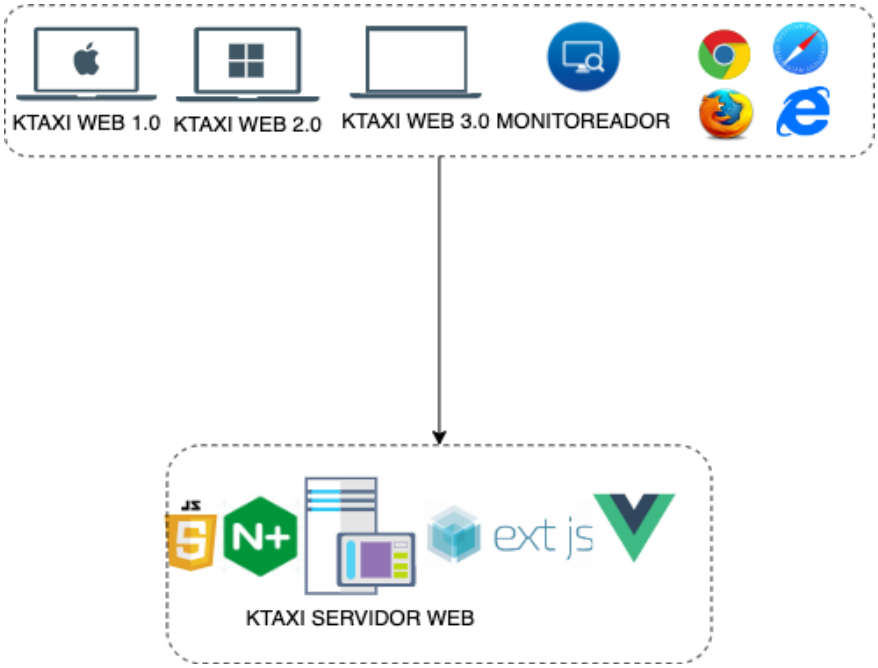

Fig. 10. Arquitectura del sistema de administración, gestión y monitoreo.

Adicionalmente cuenta con servicios especiales que utilizan servidores para cada uno de estos y que dependiendo de la institución o función que desempeñen se asigna un servidor único para realizar las acciones requeridas, estos manejan una arquitectura monolítica en donde encapsulan toda su funcionalidad en una única unidad, como se puede visualizar en el diagrama que se realizó en la Fig. 11.

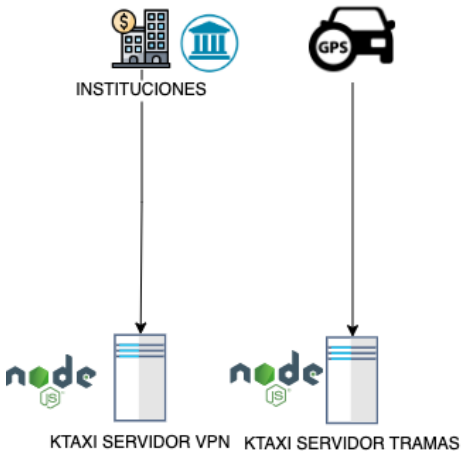

Fig. 11. Arquitectura monolítica de servicio especiales

En la Fig. 12, se establece el diagrama de la arquitectura general que se realizó resultante del análisis de la información y asesoría del **Ing. Ricardo Jumbo,** en donde se detallan los componentes que conforman la arquitectura monolítica del aplicativo Ktaxi, haciendo énfasis en el servidor del aplicativo Ktaxi cliente y conductor que es la parte central encargada de solventar todas las peticiones entrantes de los aplicativos, CallCenter, sistemas web y monitoreadores.

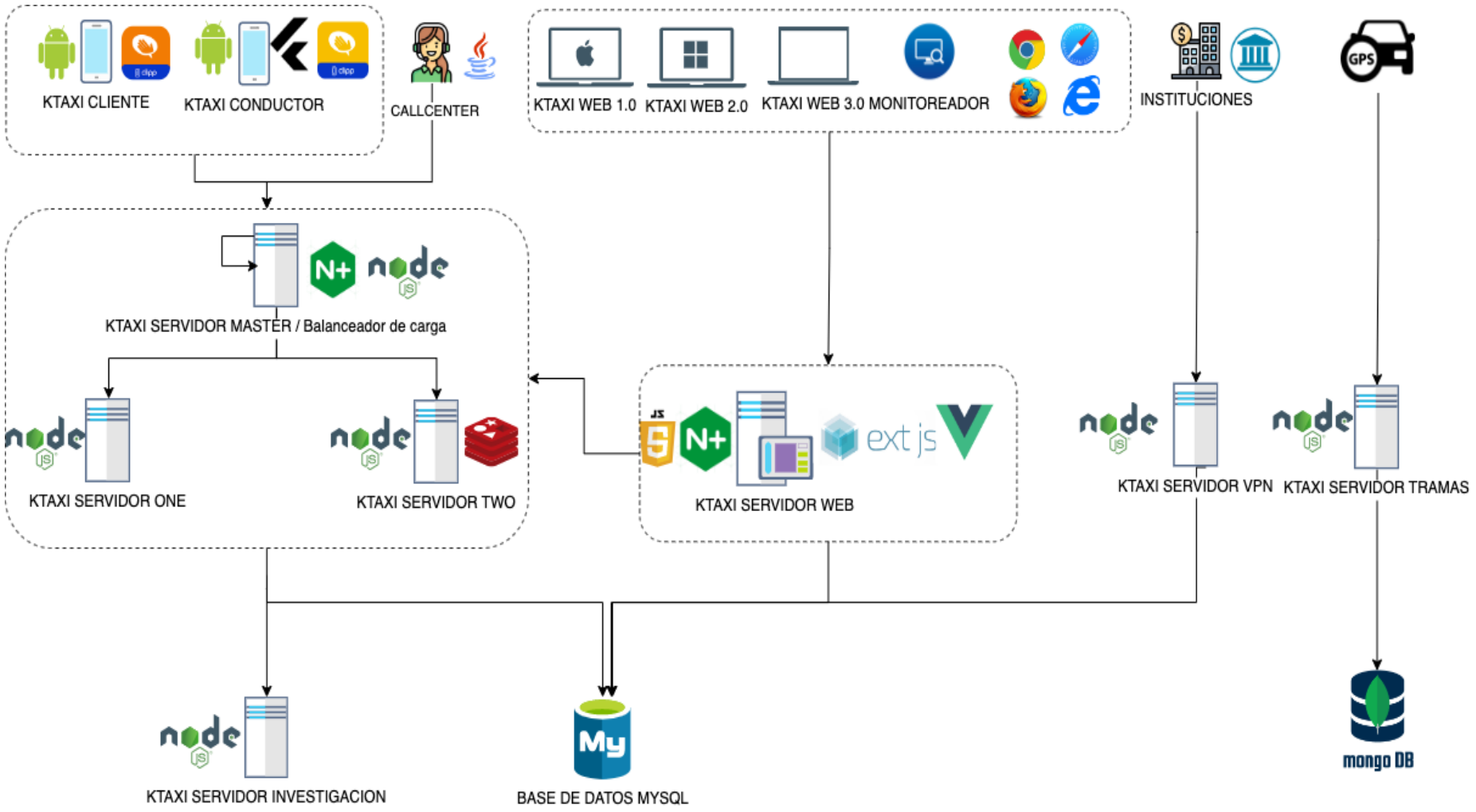

Fig. 12. Diagrama general de la arquitectura actual del aplicativo Ktaxi.

6.1.3.1 Descripción de la arquitectura general

La arquitectura del sistema Ktaxi consta de 3 servidores centOS en su versión 7 denominados como Master, One y Two, funcionando como backend para el aplicativo Ktaxi cliente, conductor y CallCenter cada uno de ellos integrado para realizar las mismas tareas con el objetivo principal de optimizar la carga transaccional de las solicitudes.

Para los sistemas de administración y monitoreo cuenta con un servidor web centOS en su versión 7 denominado web, funcionando como backend y frontend, donde radican dichos sistemas y que están disponibles para los administradores o gerentes de negocio, este cuenta con conexión al servidor principal de Ktaxi para realizar operaciones y gestionar información relevante para su funcionamiento.

El almacenamiento y gestión de la información lo realiza en un servidor donde tiene una base de datos relacional MYSQL, aquí se almacena toda la información proveniente de los aplicativos Ktaxi cliente, conductor, CallCenter, sistemas web y monitoreadores, debido a la inmensa cantidad de información transaccional mantienen bases de datos históricas para migrar dicha información y garantizar la disponibilidad de la misma.

Los archivos estáticos como imágenes, archivos, documentos entre otros, cuenta con un servidor centOS en su versión 7 denominado investigación que funciona con un repositorio para almacenar todo este tipo de información que generalmente suele ser usada por el servidor de Ktaxi y el servidor web. Por último, mantiene 2 servidores centOS en su versión 7 funcionando como backend para el servicio de instituciones y para la gestión de tramas de gps.

En la TABLA II., se puede observar la descripción de cada uno de los componentes que conforman la arquitectura monolítica del aplicativo Ktaxi.

TABLA II.

COMPONENTES DE LA ARQUITECTURA MONOLÍTICA DEL APLICATIVO KTAXI

Componente	Descripción
SERVIDOR MASTER	Servidor principal del aplicativo Ktaxi cliente, conductor y CallCenter, maneja la lógica de negocio (funciones, métodos, recursos, APIs) y funciona como un balanceador

	de carga para distribuir el trabajo entre sí mismo, el servidor one o el servidor two.
SERVIDOR ONE	Maneja la lógica de negocio (funciones, métodos, recursos, APIs).
SERVIDOR TWO	Maneja la lógica de negocio (funciones, métodos, recursos, APIs).
BASE DE DATOS MYSQL	Almacenamiento de toda la información correspondiente a solicitudes, usuarios, configuraciones, formas de pago, valores de taxímetros, etc. Cada uno de los servidores antes mencionados se conectan para gestionar la información.
SERVIDOR VPN	Contiene las configuraciones necesarias para garantizar el acceso permitido a ciertos recursos por parte de ciertas instituciones.
SERVIDOR DE TRAMAS	Permite el desentramado de información proveniente de equipos físicos instalados en unidades vehiculares como taxis, cuya información permite conocer su geoposición en tiempo real.
SERVIDOR DE INVESTIGACIÓN	Contiene información estática como archivos, imágenes e implementa lógica de negocio enfocada a las rutas, los servidores Master, One y Two pueden acceder para obtener información relevante para su servicio.
SERVIDOR WEB	Contiene sistemas de administración y gestión de la información, así como el monitoreador.

6.1.3.2 Diagrama de componentes del servidor del aplicativo Ktaxi

El aplicativo Ktaxi cliente y conductor se conectan al servidor backend denominado Master que tiene la función de garantizar la optimización de los recursos asignados mediante la configurado un balanceador de carga desarrollado en NGINX que se encarga de administrar las peticiones que ingresan y designa cuál de los tres servidores disponibles (Master, One y

Two) incluyéndose lo va a gestionar como se puede visualizar en el diagrama que se realizó en la Fig. 13, y cuya descripción de las tecnologías y herramientas en la cual está desarrollado se pueden observar en la TABLA III.

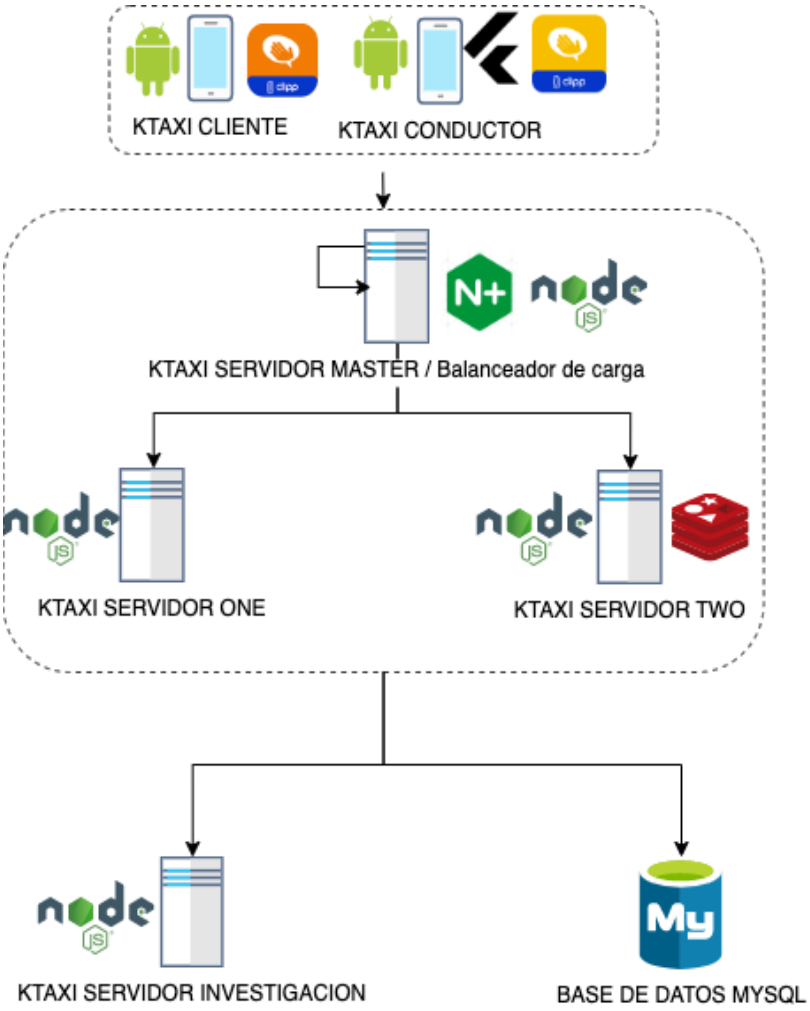

Fig. 13. Diagrama de componentes del aplicativo Ktaxi

TABLA III.
TECNOLOGÍAS Y HERRAMIENTAS DE LA ARQUITECTURA ACTUAL DEL APLICATIVO KTAXI.

SERVIDOR MASTER	
Tecnologías / Herramientas	**Lenguaje de programación:** JavaScript como tecnología del lado del servidor, el servicio está desarrollado en esta tecnología.

	Entorno de ejecución: Node.js como entorno en tiempo de ejecución de JavaScript
	Balanceador de carga: NGINX configurado como balanceador de carga entre los servidores Master, One y Two
	Monitoreador: HTOP que nos permite visualizar los procesos y gestionarlo
	Gestor de procesos: PM2 gestor de procesos de producción para Node.js permite levantar un servicio interno en el servidor en tiempo de ejecución.
	Seguridad: Certbot para la administración de certificados SSL a través de complementos.
S.O	CentOS 7
Respaldo	Antes de cada nueva actualización se genera un respaldo junto a la versión actual en un directorio separado.
Réplica	One y Two
SERVIDOR ONE	
Tecnologías / Herramientas	**Lenguaje de programación:** JavaScript como tecnología del lado del servidor, el servicio está desarrollado en esta tecnología.
	Entorno de ejecución: Node.js como entorno en tiempo de ejecución de JavaScript
	Monitoreador: HTOP que nos permite visualizar los procesos y gestionarlo
	Gestor de procesos: PM2 gestor de procesos de producción para Node.js permite levantar un servicio interno en el servidor en tiempo de ejecución.
	Seguridad: Certbot para la administración de certificados SSL a través de complementos.

S.O	CentOS 7
Respaldo	Antes de cada nueva actualización se genera un respaldo junto a la versión actual en un directorio separado.
Réplica	Master y Two
SERVIDOR TWO	
Tecnologías / Herramientas	**Lenguaje de programación:** JavaScript como tecnología del lado del servidor, el servicio está desarrollado en esta tecnología.
	Entorno de ejecución: Node.js como entorno en tiempo de ejecución de JavaScript
	Monitoreador: HTOP que nos permite visualizar los procesos y gestionarlo
	Almacenamiento de información en memoria: Redis para el acceso rápido a la información
	Gestor de procesos: PM2 gestor de procesos de producción para Node.js permite levantar un servicio interno en el servidor en tiempo de ejecución.
	Seguridad: Certbot para la administración de certificados SSL a través de complementos.
S.O	CentOS 7
Respaldo	Antes de cada nueva actualización se genera un respaldo junto a la versión actual en un directorio separado.
Réplica	One y Master

Cabe mencionar que debido a que los 3 servidores antes mencionados tienen lo mismo, es decir tienen la misma estructura de código y la misma lógica de negocio, para garantizar su correcto

funcionamiento deben tener las mismas tecnologías instaladas y configuraciones referentes al servicio.

6.1.3.3 Diagrama de comunicación de la arquitectura del aplicativo Ktaxi

La comunicación la establecen bajo el protocolo HTTPS para garantizar que la información que viaja desde los aplicativos, CallCenter, sistemas web y monitoreador como se puede visualizar en el diagrama que se realizó de la Fig. 14, lo haga de manera cifrada, una vez la petición sea realizada se devuelve como respuesta por parte del servicio un objeto JSON que contiene la información requerida al cliente que la solicitó.

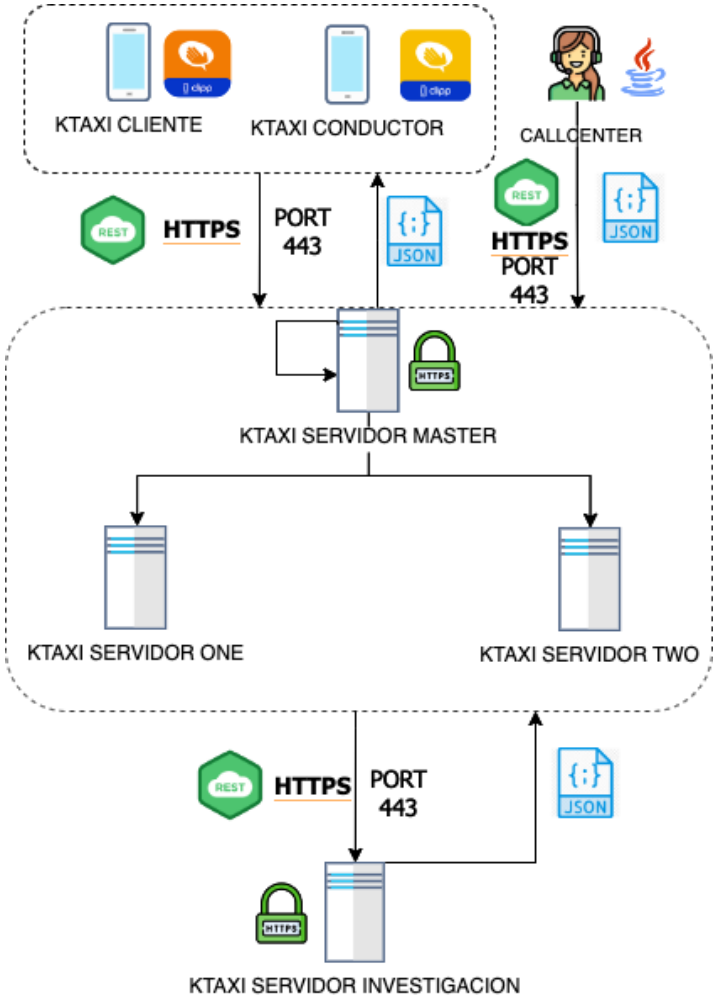

Fig. 14. Comunicación del aplicativo Ktaxi cliente, conductor y CallCenter con el servidor del aplicativo Ktaxi.

En la Fig. 15, se puede visualizar el diagrama que se realizó sobre comunicación de cada servidor con sus respectivos clientes, cada uno de ellos implementa un certificado de seguridad SSL y se ejecutan mediante llamadas REST al servicio que una vez resuelto devuelve un objeto JSON con la respuesta.

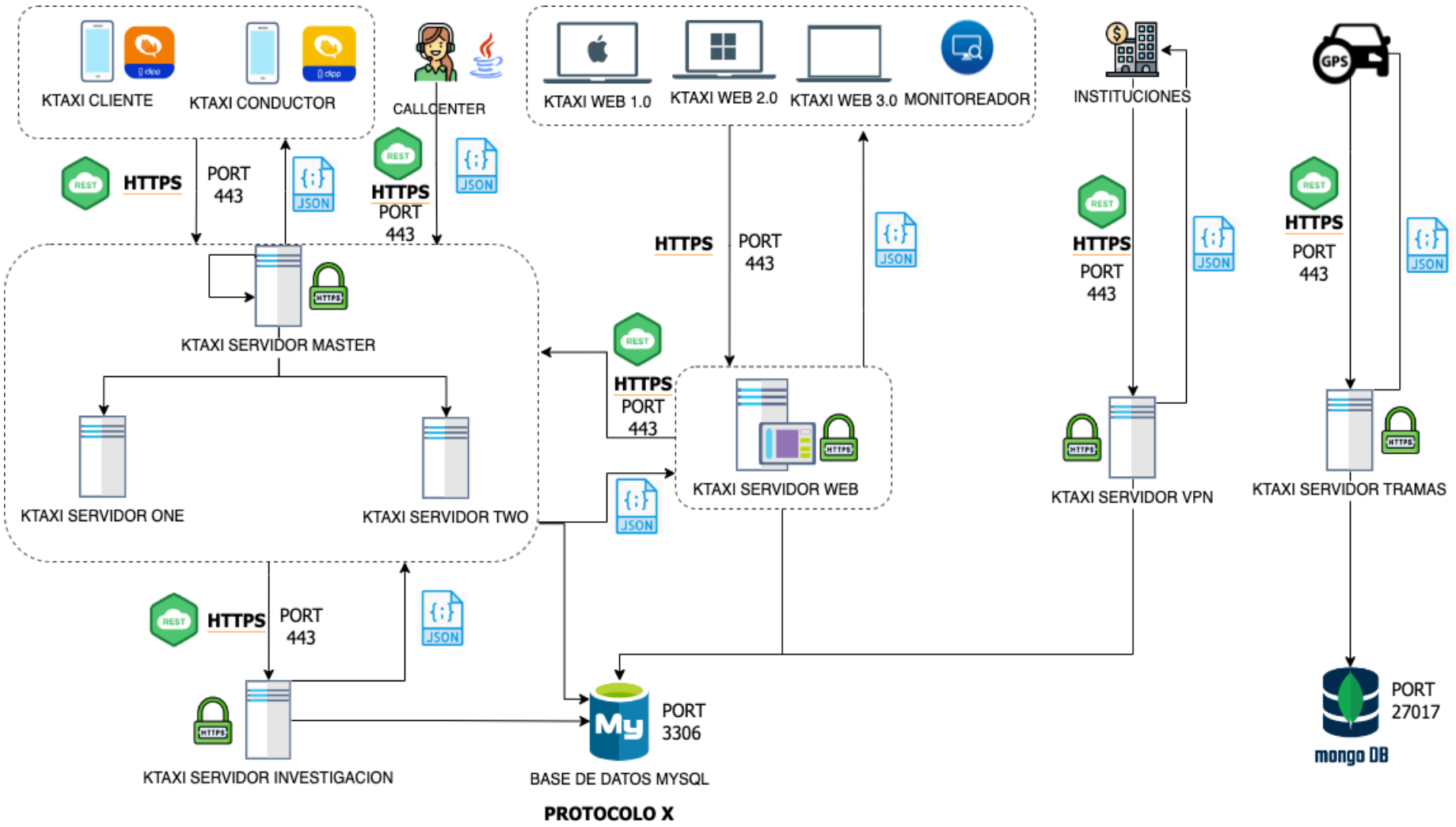

Fig. 15. Diagrama de comunicación actual de la arquitectura del aplicativo Ktaxi

En la TABLA IV., se realizó un resumen de la comunicación de los componentes descritos en la Fig. 15, así como el protocolo que utilizan, el puerto, el formato de la respuesta y el formato de consumo de las peticiones.

TABLA IV.

COMUNICACIÓN A LOS DIFERENTES SERVIDORES DEL APLICATIVO KTAXI

Sistema	**Servidor**	**Protocolo**	**Puerto**	**Respuesta**	**Consumo**
Ktaxi Cliente	Ktaxi servidor Master	HTTPS	443	JSON	REST
Ktaxi Conductor	Ktaxi servidor Master	HTTPS	443	JSON	REST
CallCenter	Ktaxi servidor Master	HTTPS	443	JSON	REST
Ktaxi web 1.0	Ktaxi servidor Web	HTTPS	443	JSON	HTTPS
Ktaxi web 2.0	Ktaxi servidor Web	HTTPS	443	JSON	HTTPS
Ktaxi web 3.0	Ktaxi servidor Web	HTTPS	443	JSON	HTTPS
Ktaxi servidor Web	Ktaxi servidor Master	HTTPS	443	JSON	REST
Ktaxi servidor Master	Ktaxi servidor Investigación	HTTPS	443	JSON	REST
Ktaxi servidor One	Ktaxi servidor Investigación	HTTPS	443	JSON	REST
Ktaxi servidor Two	Ktaxi servidor Investigación	HTTPS	443	JSON	REST

6.1.3.4 Diagrama de seguridad de la arquitectura del aplicativo Ktaxi

La capa de seguridad definida en la arquitectura monolítica corresponde a un firewall en cada uno de sus servidores que tiene configurado un grupo de reglas que dictan qué tráfico debe permitir. Esto permite prevenir la actividad maliciosa y evitar que cualquier persona dentro o fuera de la red pueda realizar acciones no autorizadas, como ejemplo se puede visualizar el firewall del servidor Master en la Fig. 16, cuyo estado se encuentra en activo.

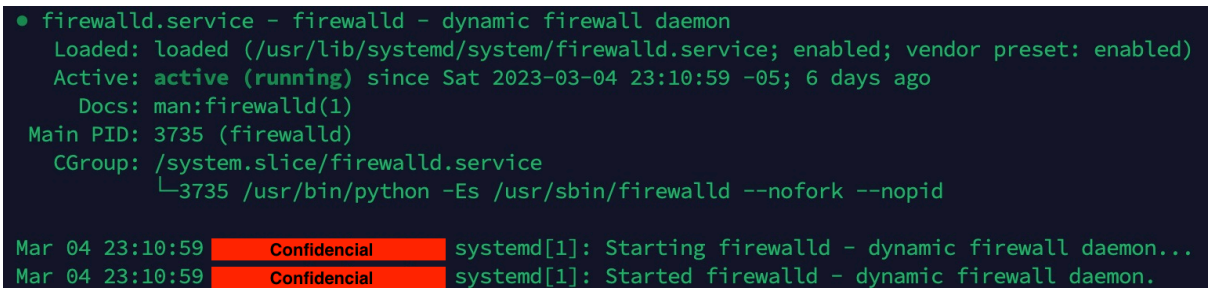

Fig. 16. Firewall activo servidor Master

En la TABLA V., se detalló la configuración del firewall de cada uno de los servidores del aplicativo Ktaxi descritos en la Fig. 17, así como el estado actual de los mismos.

TABLA V.
LISTA DE FIREWALL EN LA ARQUITECTURA DEL APLICATIVO KTAXI

Servidor	**Servicio**	**Estado**
Ktaxi servidor Master	*firewalld.service*	*active*
Ktaxi servidor One	*firewalld.service*	*active*
Ktaxi servidor Two	*firewalld.service*	*active*
Ktaxi servidor Investigación	*firewalld.service*	*active*
Ktaxi servidor VPN	*firewalld.service*	*active*
Ktaxi servidor Web	*firewalld.service*	*active*
Ktaxi servidor Tramas	*firewalld.service*	*active*

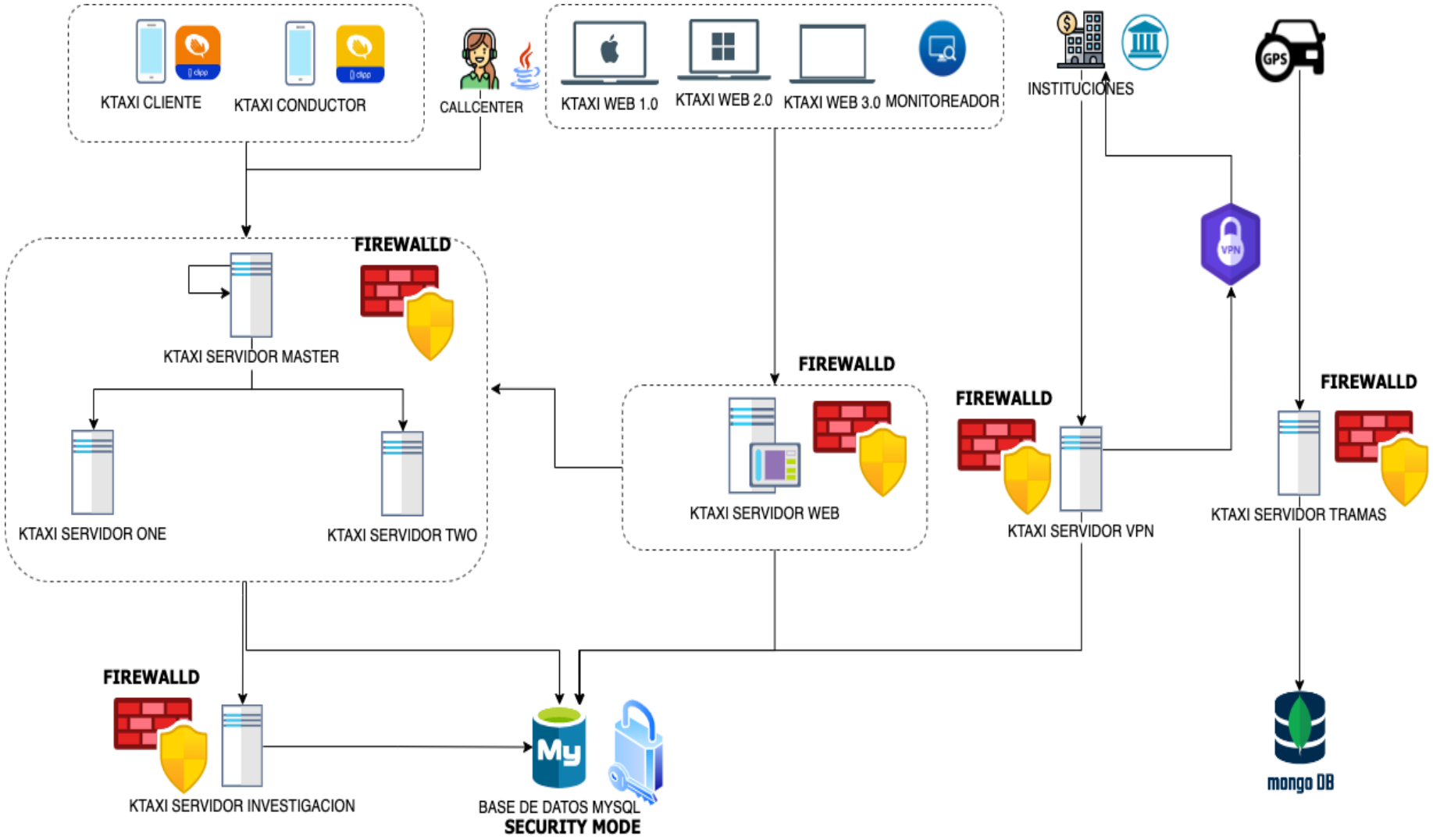

Fig. 17. Diagrama de seguridad actual de la arquitectura del aplicativo Ktaxi

6.2 FASE 2: Analizar los principales componentes que intervienen en el diseño de una arquitectura basada en microservicios en la nube mediante una revisión bibliográfica de información para identificar los componentes a utilizar.

La propuesta de arquitectura basada en microservicios en la nube que se pretendió establecer necesitó un conjunto de componentes bien definitivos cada uno de ellos cumpliendo un objetivo específico para garantizar la correcta integración y comunicación entre ellos, así como su integración y despliegue continuo.

En base a la información obtenida del análisis bibliográfico de información de empresas tecnológicas líderes en el mercado, revisiones de literatura, reportes y consultoras tecnológicas que se puede ver en el **Anexo *8.*** Análisis de los principales componentes de una arquitectura basada en microservicios, se analizó y se identificó 6 componentes principales para el desarrollo de la arquitectura propuesta basada en microservicios, los cuales se pueden observar en la Fig. 18, y se encuentran fundamentados en la TABLA VI, adicionalmente se agregó un componente más (***Socket.IO***) para la comunicación en tiempo real entre cliente - servidor necesario para la lógica de negocio del aplicativo Ktaxi, de igual manera en base a la experiencia adquirida, componentes seleccionados y a la estructura tecnológico que incorpora Kradac Cia. Ltda., se propone el uso de algunas tecnologías para el desarrollo de cada uno de ellos descritos en la TABLA VI.

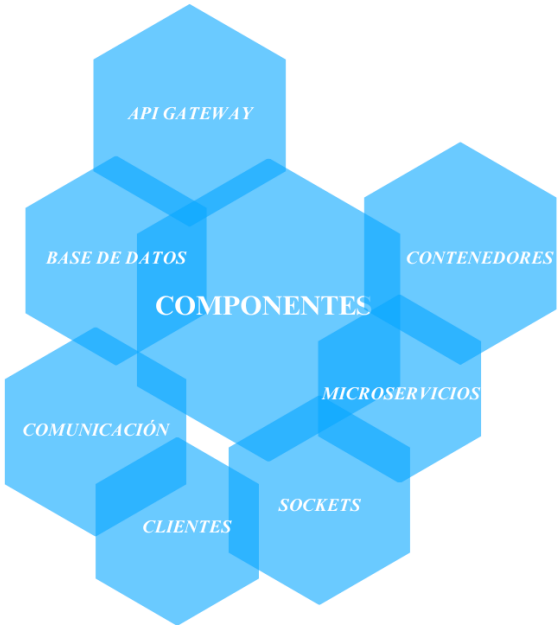

Fig. 18. Componentes seleccionados para la arquitectura de microservicios

TABLA VI.

COMPONENTES PARA LA ARQUITECTURA PROPUESTA BASADA EN MICROSERVICIOS EN LA NUBE

Componentes	**Características principales para su utilización**	**Tecnologías y protocolos propuestos**
API Gateway	API Gateway resuelve el problema de cómo las personas llaman a microservicios independientes [15], cuyas ventajas de su selección se detallan a continuación: • Encapsula la implementación interna y la interfaz específicas del sistema. • Permite que los clientes no se vean afectados por la ubicación de la instancia del servicio y que sea indetectable como la aplicación se divide en múltiples servicios. • Proporciona la API óptima para que cada cliente reduzca la solicitud. • Tiene funciones como verificación de permisos, equilibrio de carga, almacenamiento en caché y monitoreo.	• ***Middleware access security*** Se lo seleccionó como control de acceso para el consumo de los microservicios. • ***Middleware proxy*** Se lo seleccionó para recuperar microservicios en nombre de un cliente externo.

Comunicación entre microservicios	Lo ideal es intentar minimizar la comunicación entre los microservicios internos. Cuantas menos comunicaciones haya entre microservicios, mejor [16]. Sin embargo, para la presente propuesta se seleccionó una comunicación asíncrona que proporciona una mejor eficiencia de rendimiento y mayor disponibilidad [17]. Esto para evitar el mal rendimiento de cada microservicio al crear una cadena de solicitudes y que alguno de ellos falle mientras se atiende la solicitud del cliente.	• ***RabbitMQ*** Implementa un protocolo estándar llamado Advanced Message Queuing Protocol (AMQP), que se centra tanto en el comportamiento del servidor que ofrece los mensajes, como en el del cliente de la mensajería. Es de código abierto, con licencia pública de Mozilla, y gratuito para cualquier uso que se pretenda realizar en aplicaciones comerciales. Se seleccionó RabbitMQ ya que emplea el modelo de corredor inteligente / consumidor simple. El corredor entrega constantemente mensajes a los consumidores y realiza un seguimiento de su estado mientras que en otros modelos como Kafka los consumidores deben controlar su posición en cada registro.
Contenedores	Elimina la mayoría de los problemas que surgen de configuraciones de entorno incoherentes y los problemas que se producen con ellos. Además, los contenedores permiten una funcionalidad rápida de escalado vertical de aplicaciones mediante la creación de nuevos contenedores según sea necesario [18].	• ***Docker*** Se lo seleccionó como entorno ligero y portátil que permiten a los desarrolladores empaquetar y ejecutar sus aplicaciones con todas las dependencias necesarias. Cada contenedor ejecuta un solo proceso,

	Es un entorno operativo aislado, controlado por recursos y portátil en el que una aplicación se puede ejecutar sin tocar los recursos de otros contenedores o el host. Las empresas adoptan cada vez más contenedores al implementar aplicaciones basadas en microservicios y Docker se ha convertido en la implementación de contenedores estándar que han adoptado la mayoría de las plataformas de software y los proveedores de nube [18].	lo que proporciona una forma de aislar y administrar múltiples aplicaciones en una sola máquina host.
Base de datos	MySQL es la base de datos de software libre más popular. Se integra perfectamente con los lenguajes de desarrollo más utilizados en la industria (Java, PHP, .Net, Python, Perl, Ruby, etc.), y con las arquitecturas más modernas como de microservicios o REST gracias a su soporte nativo de JSON [19] [20]. Muchas de las organizaciones más grandes y de más rápido crecimiento del mundo, como Facebook, Twitter, Booking.com y Verizon, confían en MySQL para ahorrar tiempo y dinero en la potenciación de sus sitios web de alto volumen, sistemas críticos para el negocio y paquetes de software [21].	• ***MySQL*** Se seleccionó MySQL debido al modelo de negocios de la empresa Kradac Cia. Ltda., además porque es gratis bajo la licencia GPL de código abierto, y el costo por licencia comercial es muy razonable, también porque contiene una gran cantidad de librerías desarrolladas para conectarse a esta base de datos desde lenguajes de programación como Node.js, Java o Python. Además, dispone de una imagen en Docker Hub.

Socket	Proporciona garantías adicionales, como el respaldo al sondeo largo HTTP o la reconexión automática, es un protocolo de comunicación que proporciona un canal full-duplex y de baja latencia entre el servidor y el navegador [22].	• ***Socket.IO*** Se lo seleccionó para que las aplicaciones tengan una comunicación en tiempo real mediante un canal bidireccional, dentro de la arquitectura cada microservicio podrá enviar mensajes asíncronos con RabbitMQ al microservicio de Socket.IO.
Microservicios	Estos servicios están construidos alrededor de las capacidades de negocio, y son desplegados por herramientas automáticas de forma independiente. Además, hay una mínima gestión centralizada de servicios, y pueden ser escritos en diferentes lenguajes de programación y usar diferentes tecnologías de almacenamiento.	• ***JWT*** Se lo seleccionó para la autenticación API y la autorización de servidor a servidor, es una cadena codificada y segura para URL que puede contener una cantidad ilimitada de datos y que está firmada criptográficamente. Se la seleccionó para la generación de tokens seguros basados en la autenticación de los clientes. • ***API REST*** Se lo seleccionó para la comunicación entre los clientes y los microservicios mediante la exposición de una API REST con métodos bien definidos y cuya respuesta será en formato JSON. • ***Control de versionado en el encabezado HTTP*** • ***Monitoreo PM2***

		Se utilizará para la gestión de procesos y recursos de cada microservicio. • ***Node js***
Clientes	Necesarios para consumir o utilizar los diferentes microservicios pueden ser diferentes dispositivos como teléfonos, computadoras, tabletas, smart watch, entre otros.	• **Ktaxi Cliente** Cliente consumidor de microservicios. • **Ktaxi Conductor** Cliente consumidor de microservicios.

6.2.1 Componente API Gateway

En la Fig. 19, se encuentra el diagrama que se realizó del API Gateway para implementarse en la arquitectura propuesta, como se puede observar en el diagrama la función principal dentro de la arquitectura es enrutar una solicitud al microservicio adecuado y enviar una respuesta al cliente que lo solicitó. Además, incorporará políticas de autenticación y control de acceso para la protección de la información.

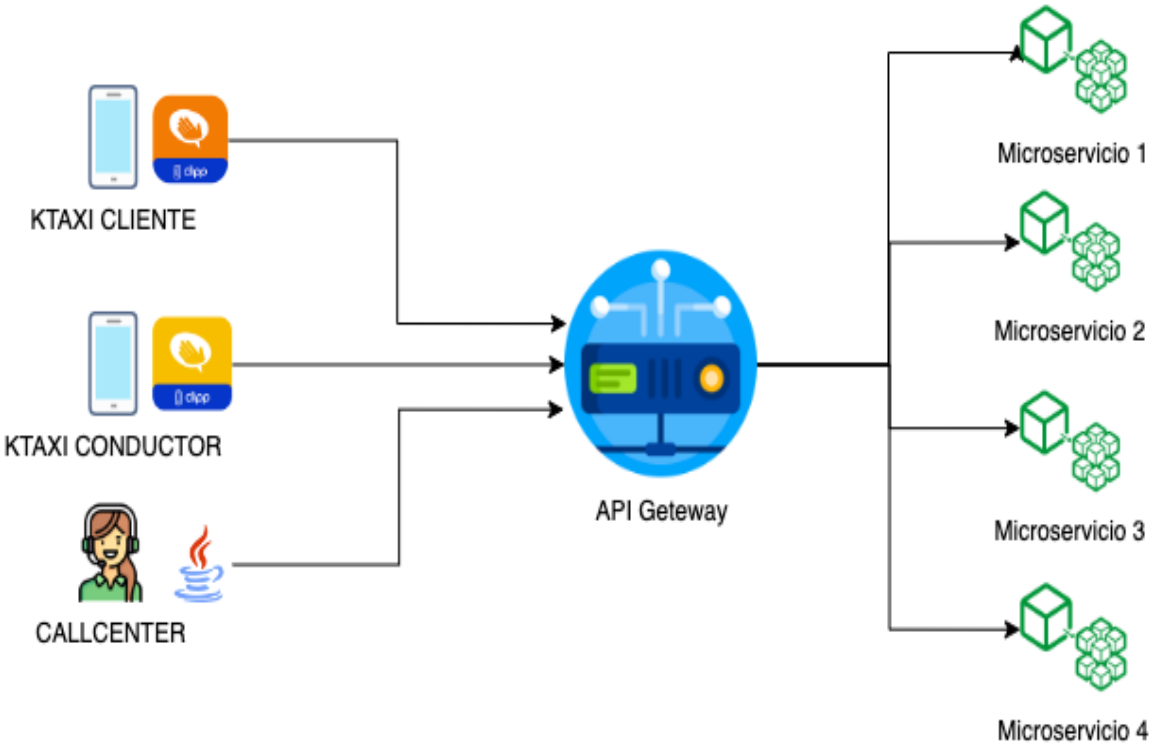

Fig. 19. Diagrama API Gateway para enrutar las solicitudes a los microservicios

Todos los microservicios definidos serán registrados mediante un punto final en el API Gateway que contendrá la dirección del microservicio y mediante un proxy redirigirá cada petición a cada microservicio correspondiente, cuando un cliente solicite un microservicio anteponiendo el punto final el API Gateway buscará el registro correspondiente en su configuración y si lo encuentra lo redirigirá a la dirección registrada caso contrario se presentará un error ***404.***

En la Fig. 20, se puede observar un diagrama que se realizó en donde se puede evidenciar el registro de 3 microservicios cada uno de ellos con una dirección propia registrados con un punto final diferente.

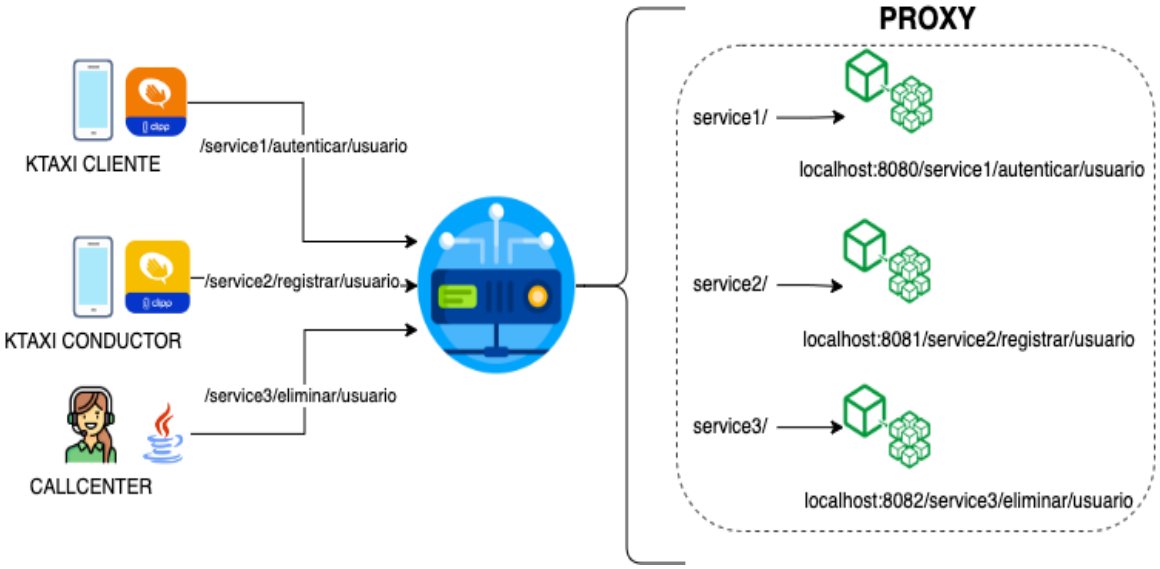

Fig. 20. Ejemplo del registro de microservicios en el API Gateway

Cuando un cliente necesite consumir un microservicio deberá autenticarse mediante un punto final ***/login*** en el API Gateway con la finalidad de garantizar que la solicitud sea de origen confiable, si un cliente accede a un microservicio el API Gateway verificará si el acceso a dicho microservicio tiene restricción de autenticación y verificará mediante un middleware si el acceso ha sido previamente concedido luego mediante un proxy el API Gateway redirigirá la solicitud al microservicio de destino, caso contrario devuelve una respuesta al cliente indicando que no a sido posible verificar su sesión, como se indica en el diagrama que se realizó en la Fig. 21, solo el microservicio 4 podrá acceder directamente al microservicio sin la necesidad de especificar una autenticación.

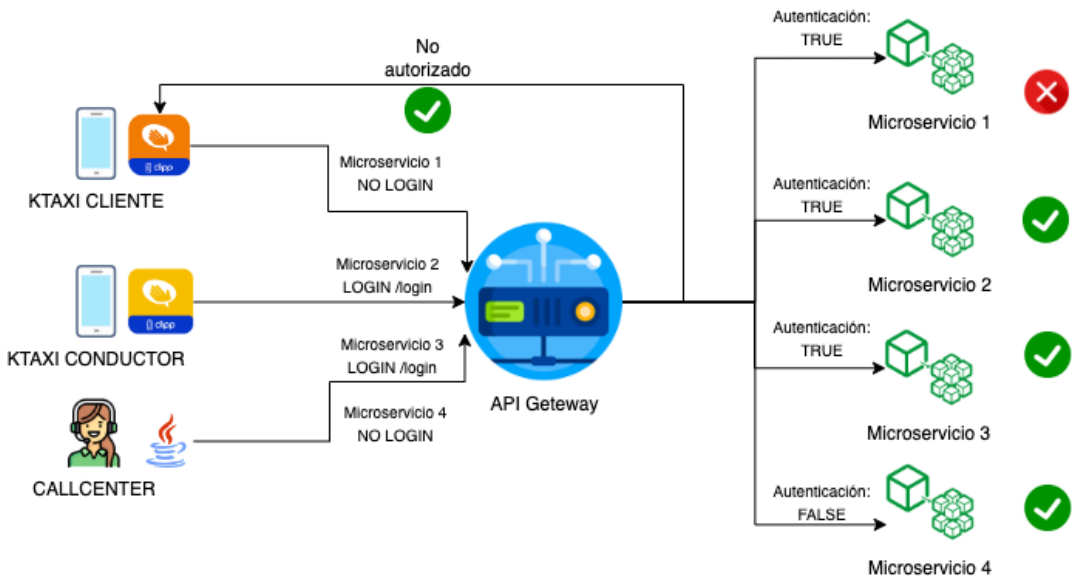

Fig. 21. Autenticación para el acceso a los microservicios del API Gateway

6.2.2 Componente para la comunicación entre microservicios

Cada microservicio es independiente y realiza una sola cosa durante su tiempo de ejecución, es decir cada microservicio no depende de otro para realizar una acción u operación esto con el propósito de evitar bloqueos en los subprocesos, sin embargo, existen casos particulares donde un microservicio necesita notificar a otro microservicio que a realizado o ejecutado una acción y necesitará que esta acción se notifique a otro microservicio para que este pueda ejecutar alguna acción o tarea.

La Fig. 22, muestra el diagrama que se realizó para indicar comunicación entre microservicios mediante el envió y recepción de mensajes, para ello se propone realizar una comunicación asíncrona para garantizar que no dependa de ninguna respuesta, simplemente enviará el mensaje a una cola de mensajes y el receptor o consumidor podrá realizar ciertas operaciones según sea el caso. En la comunicación asíncrona el microservicio remitente o productor no espera la confirmación de ejecución del microservicio receptor o consumidor únicamente envía el mensaje y su interacción termina, así mismo el microservicio receptor o consumidor recibe el mensaje ejecuta alguna operación o tarea y su interacción termina.

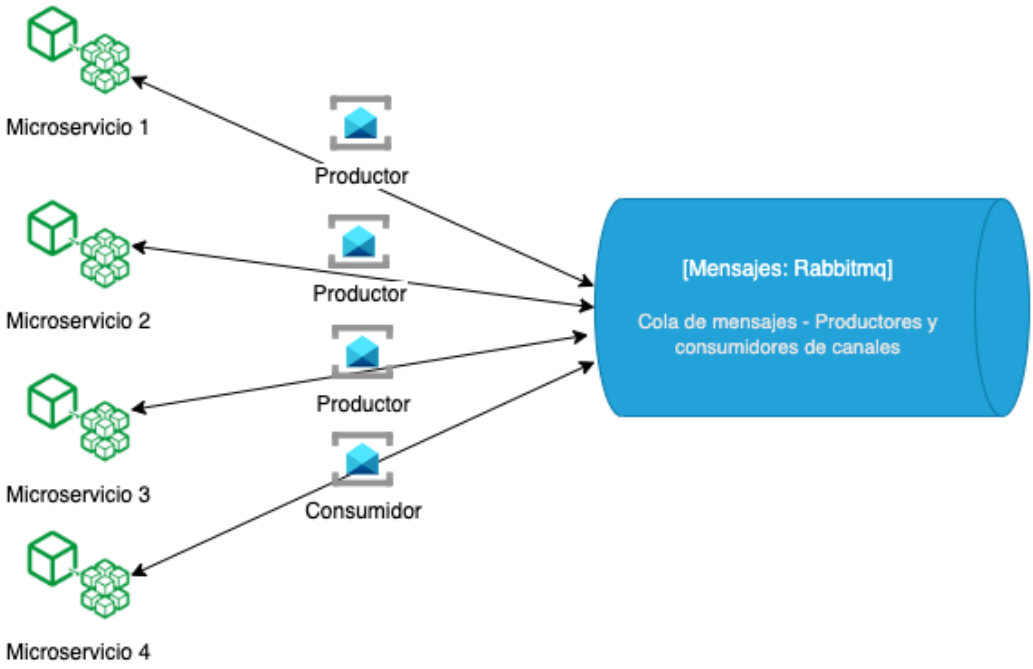

Fig. 22. Publicación y suscripción de canales para el envío de mensajes

6.2.2.1 RabbitMQ

Se definió el uso de RabbitMQ para la comunicación eficiente entre productores aquellos microservicios que envían mensajes y consumidores aquellos microservicios que reciben

mensajes, que implementa el protocolo de mensajería de capa de aplicación AMQP enfocado a los mensajes asíncronos con garantía de entrega, lo que realiza rabbitMQ, es definir una cola de mensajes que va a guardar dichos mensajes enviados por los productores hasta que los consumidores obtengan el mensaje y lo procesen, como se puede observar en la Fig. 23, 1 productor produce un mensaje que envía a 2 colas en las cuales se encuentran suscritos 2 consumidores que los van a recibir.

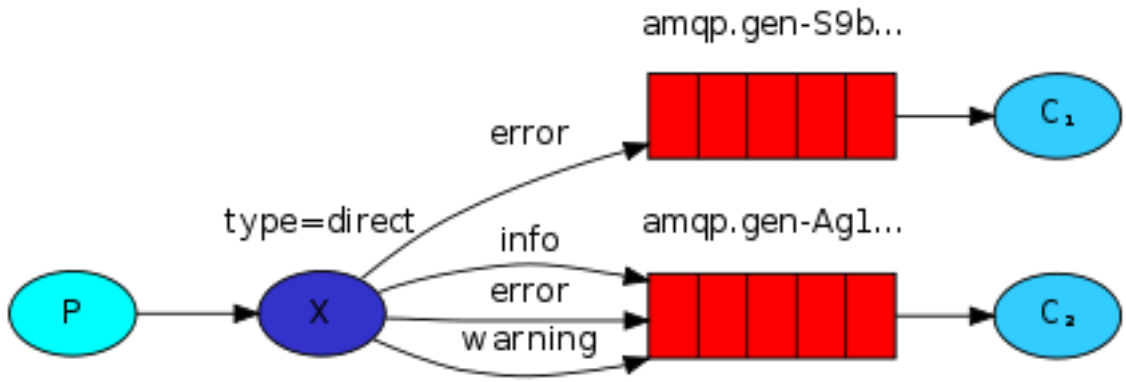

Fig. 23. Funcionamiento de rabbitMQ para la producción y consumo de mensajes

La imagen ha sido obtenida de la referencia descrita en [23].

RabbitMQ incorpora varias características que le permiten garantizar la entrega de los mensajes. Entre ellas, proporciona almacenamiento cuando no hay consumidores disponibles para recibir el mensaje, brinda la posibilidad de que el consumidor acepte la entrega del mensaje para asegurarse de que lo procesó correctamente y, en caso de que haya fallado su procesamiento, permite que el mensaje se pueda reencolar para ser consumido por una instancia diferente del consumidor o que sea procesado de nuevo por el mismo consumidor que inicialmente falló, cuando este se recupere.

En la arquitectura propuesta RabbitMQ gestionará la comunicación de aquellos microservicios que lo necesiten, haciendo énfasis en que la comunicación se la realizará de manera asíncrona para que no dependen el uno del otro sino cada uno de ellos trabajan de forma a independiente, publicando y consumiendo mensajes de las colas definidas por los microservicios.

En la Fig. 24, se propone el diseño de un diagrama que se realizó para como se implementará rabbitMQ, se puede observar como el microservicio 1 produce o publica un mensaje al canal 1 y al canal 2 creados en rabbitMQ, el microservicio 2 al estar suscrito al canal 1 consume el

mensaje que se envía mientras que los microservicios 3 y 4 están suscritos al canal 2 por lo que consumen el mensaje enviado a este canal.

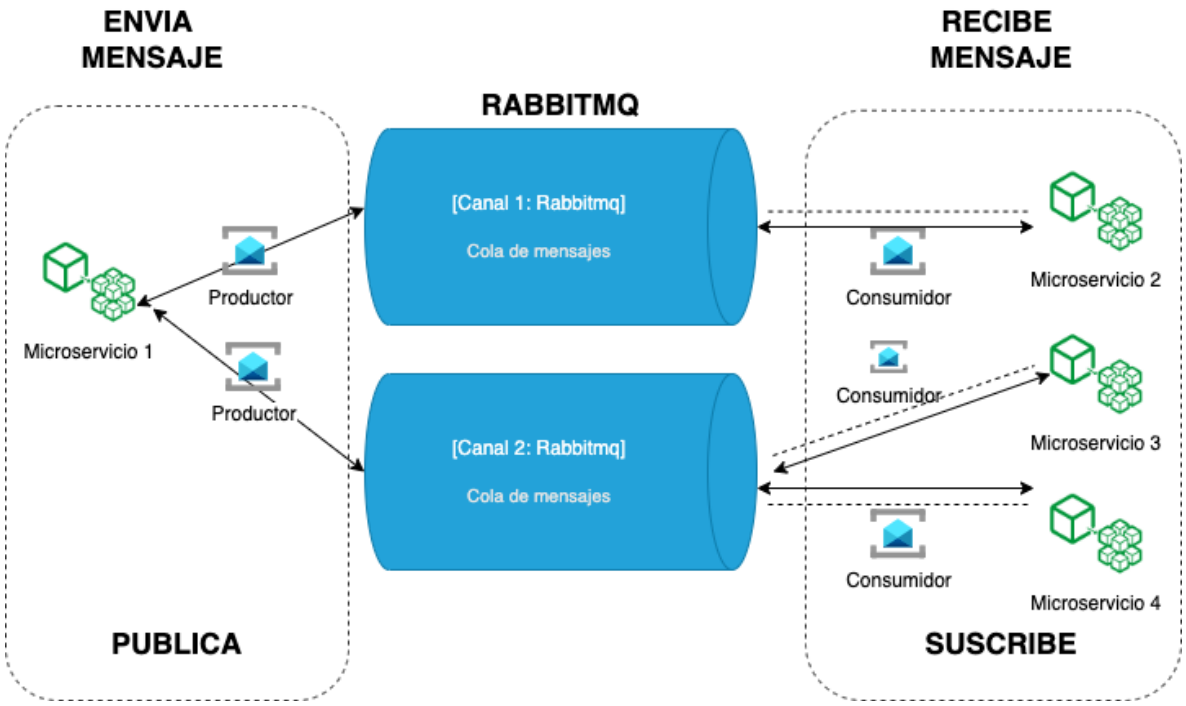

Fig. 24. Protocolo AMQP para la comunicación asíncrono entre microservicios

6.2.3 Componente para la contención de los microservicios

Se propone el uso de contenedores que contendrán todos los elementos necesarios para realizar la ejecución de los microservicios, dentro de las que se incluirán las bibliotecas y herramientas del sistema y su configuración, así como el código y el tiempo de ejecución respectivo, esto con la finalidad de ponerlos en producción fácilmente en diferentes sistemas operativos y plataformas de igual manera poder escalar las aplicaciones con mayor rapidez.

6.2.3.1 Docker

Se seleccionó para poder ejecutar los microservicios de manera independiente permitiendo su ejecución rápida y fiable en entornos informáticos, cada contenedor docker es un contendor ejecutable, independiente y ligero, además al definir qué herramientas y configuraciones se van a realizar para levantar cada microservicio su implementación será mucha más sencilla y rápida, como se puede observar en la Fig. 25, cada microservicio será dockerizado y puesto a producción para su consumo.

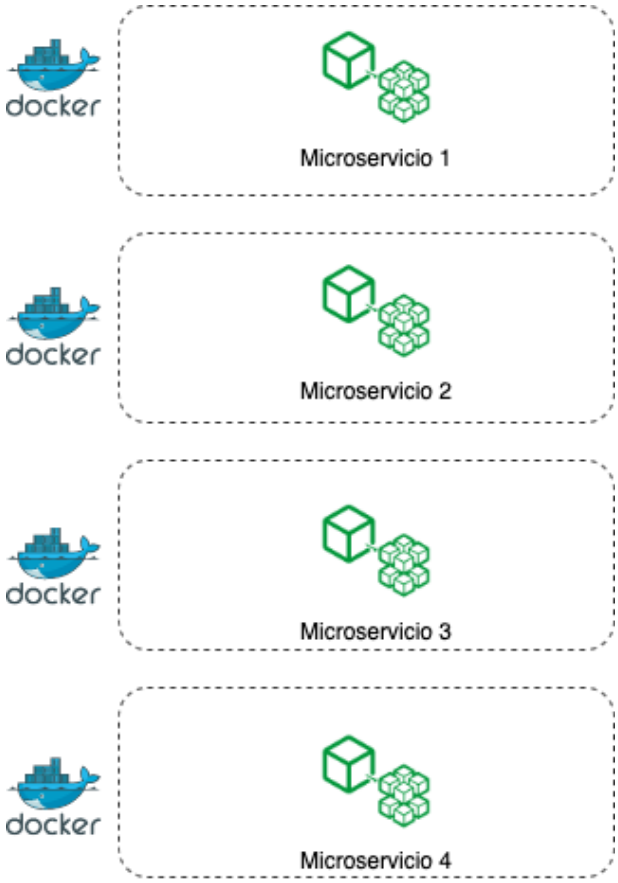

Fig. 25. Microservicios con Docker

6.2.4 Componente para la gestión de la base de datos

En base a la información actual de la empresa Kradac Cia. Ltda., y en base a la tecnología utilizada para la gestión de la información, la propuesta planteada propone el uso de la base de datos MySQL, ya que es de código abierto, multiplataforma, acepta diferentes tipos de datos, existe una gran comunidad oficial y es muy amplia por tal motivo la documentación siempre estará actualizada y disponible.

Cada microservicio tendrá su propia base de datos es decir los datos son privados para cada microservicio y que solo se deberá acceder a ellos de forma síncrona a través de los puntos de conexión de su API REST como se puede observar en el diagrama que se realizó en la Fig. 26, o bien de manera asíncrona a través del Protocolo de Cola de Mensajes Avanzado AMQP como rabbitMQ o similares.

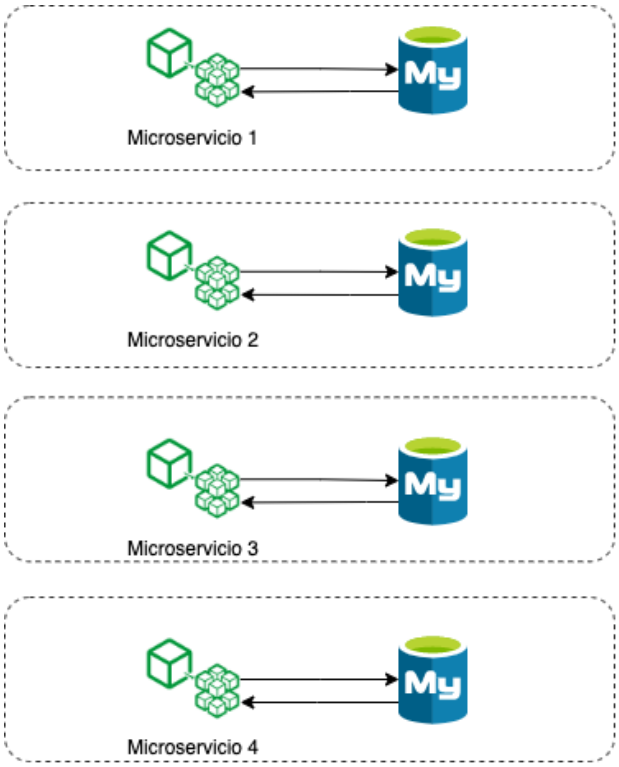

Fig. 26. Gestión de la información de microservicios con MYSQL

6.2.5 Componente para la comunicación en tiempo real

Para lograr una comunicación más fluida entre cliente servidor se propone el uso de Socket.IO, librería que se basa en eventos que permitirá crear aplicaciones en tiempo real proporcionando un canal de comunicación bidireccional entre cliente servidor.

En la Fig. 27, se muestra el diagrama que se realizó para implementar la comunicación mediante Socket.IO para el envió de información mediante emits a los clientes conectados en tiempo real, cada microservicio puede enviar uno a varios emits cada vez que ejecute alguna acción u operación y necesite que el clientes sepa que se ejecutó algo y este pueda realizar o mostrar alguna información necesaria o pertinente, de la misma forma un cliente puede enviar uno o varios emits solicitando información o la ejecución de algún proceso por parte de los microservicios que se encuentren conectados al socket y tengan establecido un oyente para el emit enviado.

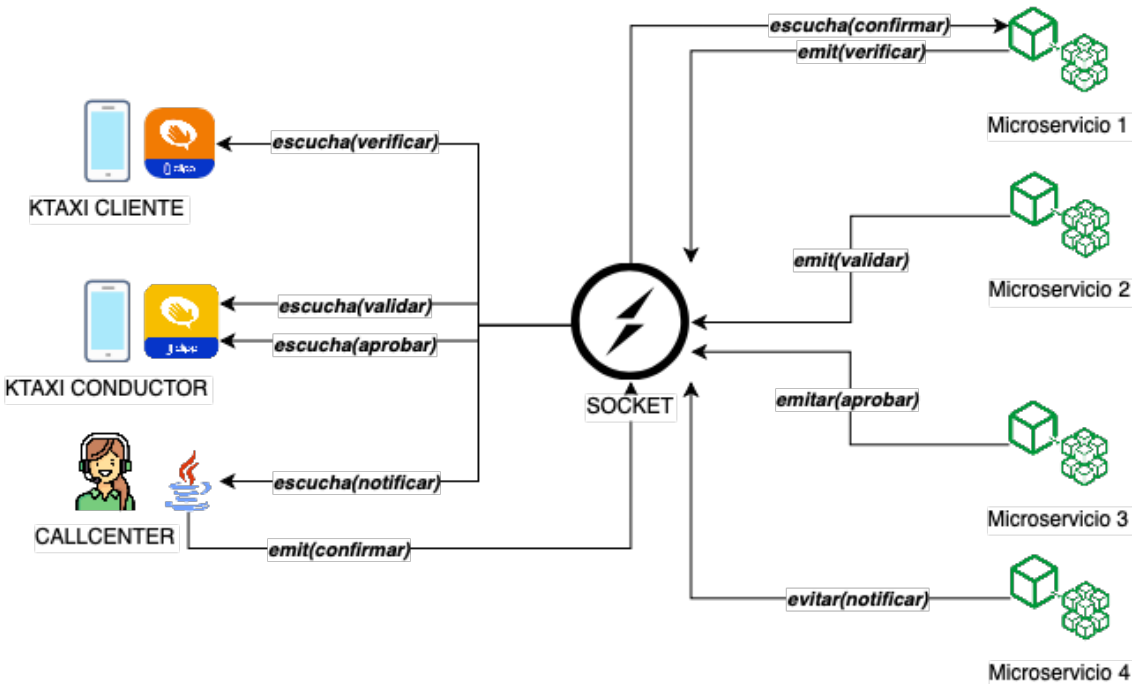

Fig. 27. Comunicación cliente - servidor mediante Socket.IO

6.2.6 Componente para la lógica del microservicio

Cada microservicio que se pretende implementar realizará una sola cosa de manera puntual sin interrumpir o afectar la ejecución de los demás microservicios, en la Fig. 28, se puede observar el diagrama que se realizó sobre 4 microservicios cada uno de ellos tiene implementado la ejecución de una sola tarea administrando su propia base de datos y sin tener interacción con los demás microservicios.

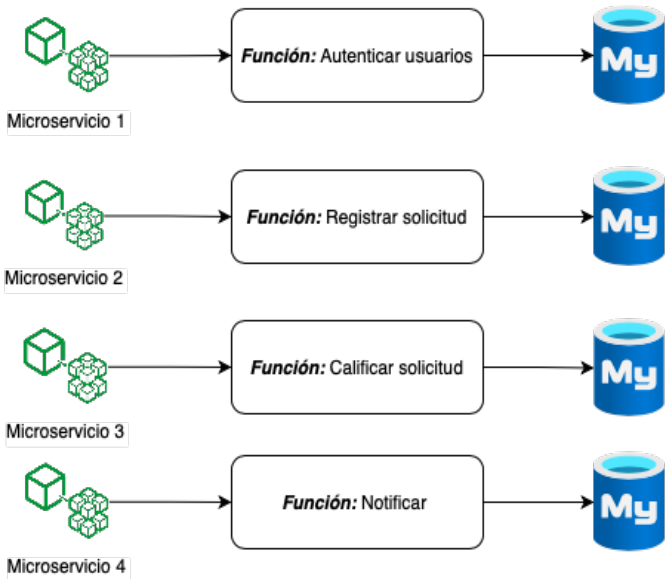

Fig. 28. Lógica de funcionamiento de Microservicios

Para la lógica de negocios de Kradac Cia. Ltda., se plantea el diseño de la arquitectura en Node.js entorno en tiempo de ejecución multiplataforma, de código abierto junto a JSON Web Tokens (JWT) para la verificación de identidad y generación de tokens seguros para el acceso a los microservicios, la comunicación se la realizará mediante API REST y para el intercambio de datos el formato JSON. Adicionalmente el registro la supervisión y administración de procesos de cada microservicio se lo realizará con el gestor de procesos de producción PM2 de código abierto y multiplataforma que permite reiniciar automáticamente el servidor de Node.js, de igual manera se utilizará Express Status Monitor para control el estado de nuestros microservicios mediante un punto final ***/status***.

6.2.6.1 JWT

Se propone el uso de JWT como mecanismo para poder propagar entre dos partes, y de forma segura, la identidad de un determinado usuario codificados en objetos de tipo JSON, que se incrustan dentro de las cabeceras de la petición que va firmado digitalmente. La firma se construirá de tal forma que se podrá verificar que el remitente es quien dice ser.

De esta forma, si alguien modifica el token por el camino, por ejemplo, inyectando alguna credencial o algún dato malicioso, entonces se podrá verificar que la comprobación de la firma no es correcta. En consecuencia, la persona encargada de la validación del token no podrá confiar en el token recibido y deberá denegar la solicitud de recursos que se haya realizado.

En la Fig. 29, se indica el diagrama que se realizó y que se pretende implementar, una vez que el cliente pueda verificar con éxito sus credenciales de acceso al microservicios de autenticación se generará un token JWT que deberá enviarse como parámetro dentro de las cabeceras de cada petición a cada microservicio, si el token JWT es válido y su firma es correcta el microservicio ejecutará su función y devolverá una respuesta al cliente, sin embargo existirán microservicios públicos es decir que no necesitarán un token en este caso no se tomará en cuenta el token JWT y su ejecución será directa.

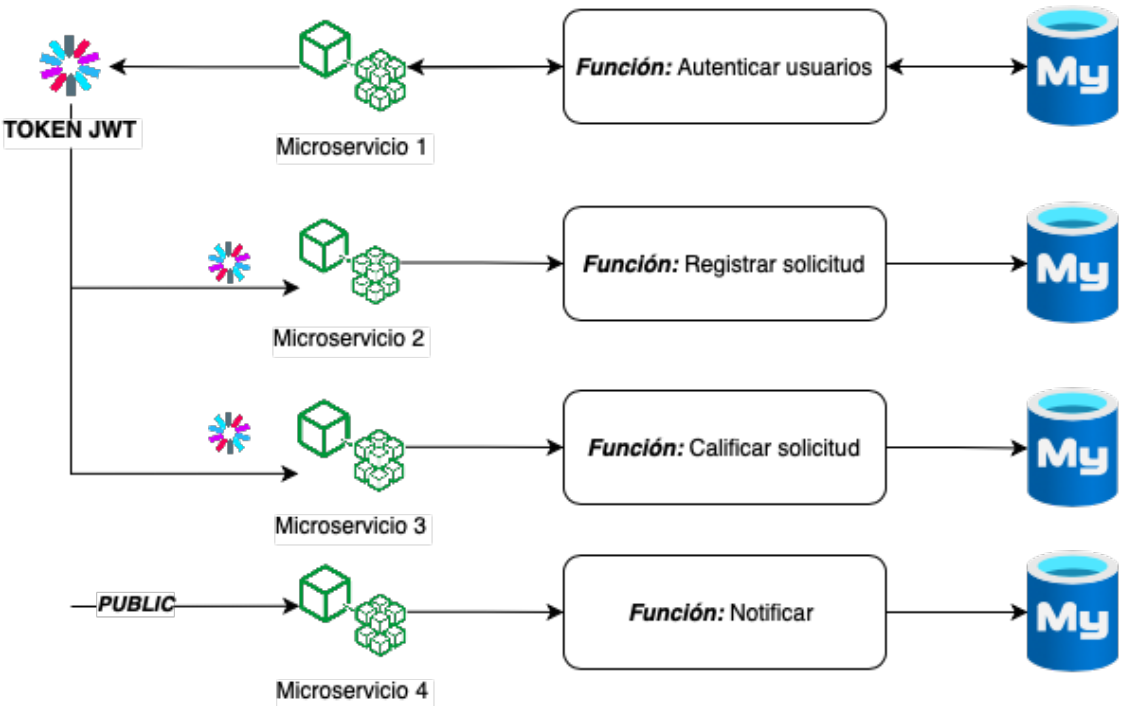

Fig. 29. Uso de tokens JWT para la autenticación y consumo de microservicios

6.2.6.2 API REST

Para la comunicación e intercambio de información de manera segura se definió el uso de API REST, cada cliente que desee acceder a la información utilizará el API expuesta siguiendo el formato de la solicitud de manera que el microservicio pueda entender, los microservicios REST recibirán la solicitud e identificarán los recursos mediante el uso de un localizador uniforme de recursos URL y devolverá una respuesta con el formato JSON.

En la Fig. 30, se puede visualizar el diagrama que se realizó indicando cómo los clientes utilizarán los diferentes métodos expuestos por el API REST para comunicarse con los microservicios, estos a su vez evalúan la solicitud y devuelven un objeto en donde se definirá un código de estado indicando si la solicitud se procesó correctamente o dio algún error y el cuerpo de la respuesta que contiene la representación del recurso.

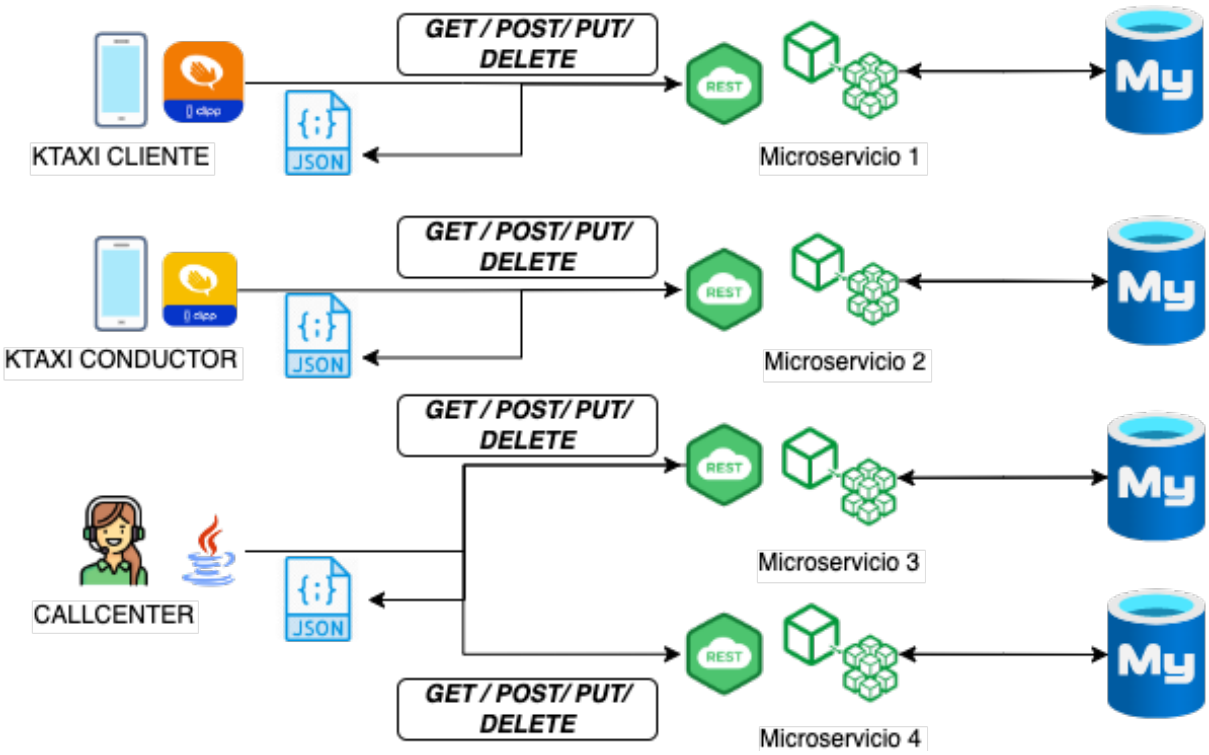

Fig. 30. API REST para la comunicación entre clientes y microservicios

6.2.6.3 Control de versionado

Para el control de versionado de los recursos expuestos por los microservicios se propone realizarlo en la cabecera en base al parámetro ***version*** definido en la solicitud del cliente, cuando un microservicio incorpore, mejore o escale nuevas funcionalidades en su lógica de negocio se creará una nueva función que se identificará mediante un número de 3 dígitos separado cada uno de ellos pon un punto como por ejemplo ***1.0.0,*** cada microservicio deberá incorporar una nueva versión para no modificar o dañar el comportamiento ya establecido del recurso actual, generalmente se definen nuevas versiones cuando un microservicio escala soluciones con nuevos parámetros que no se tomaron en cuanto o que no se necesitaban al momento de su implementación y que por el cambio constante de las empresas o soluciones tecnológicas se lo necesita realizar, un ejemplo claro de esto es cuando se necesita un nuevo parámetro de un dispositivo móvil incorporado en los nuevos terminales que antes no se tenía.

En la Fig. 31, se puede observar el diagrama que se realizó para indicar como que el microservicio 1 contiene tres funciones diferentes para cada versión y los clientes pueden acceder a ellas mediante la definición de la versión que deseen en la cabecera, cabe recalcar que si no se envía ninguna versión el microservicio definirá por defecto la versión **1.0.0.**

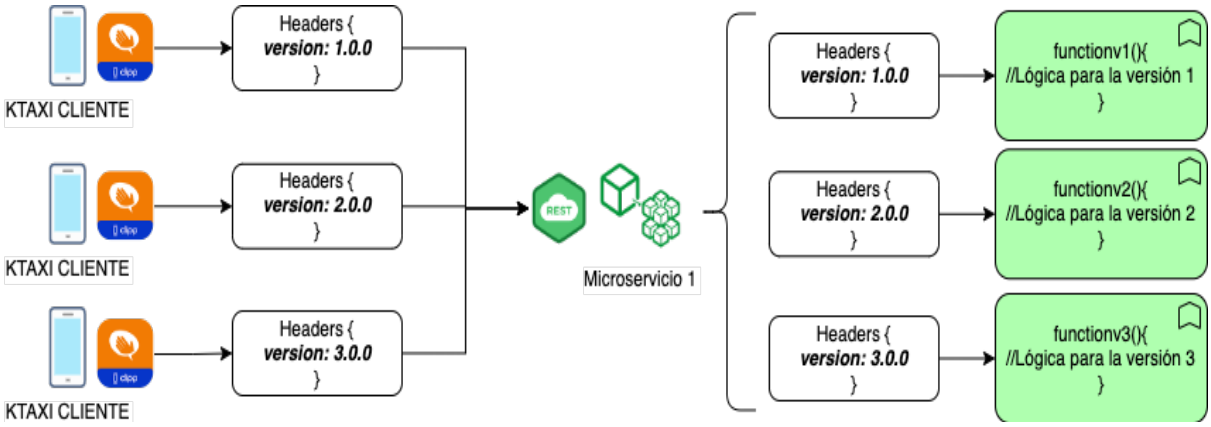

Fig. 31. Control de versionado en los microservicios

6.2.6.4 Monitoreo PM2 y Express Status Monitor

Para el monitoreo y despliegue de cada microservicio propone el uso de PM2 que va a permitir controlar el reinicio si este fallara o se bloqueara, obtener información útil sobre el rendimiento en tiempo de ejecución, el consumo y gestión de los diferentes recursos asignados. PM2 nos permitirá ejecutar nuestros servicios en segundo plano utilizando un ID propio por cada servicio lo que facilita la gestión de cada instancia.

En la Fig. 32, se puede observar el diagrama que se realizó de la implementación de PM2 en cada microservicio para mantener las aplicaciones vivas en segundo plano, recargarlas sin tiempo de inactividad y facilitar las tareas comunes de administración del sistema, adicionalmente se utilizará Express Status Monitor expuesto como punto final /**status** para obtener información sobre las métricas del servidor en tiempo real para los servidores de Node.js basados en Express.

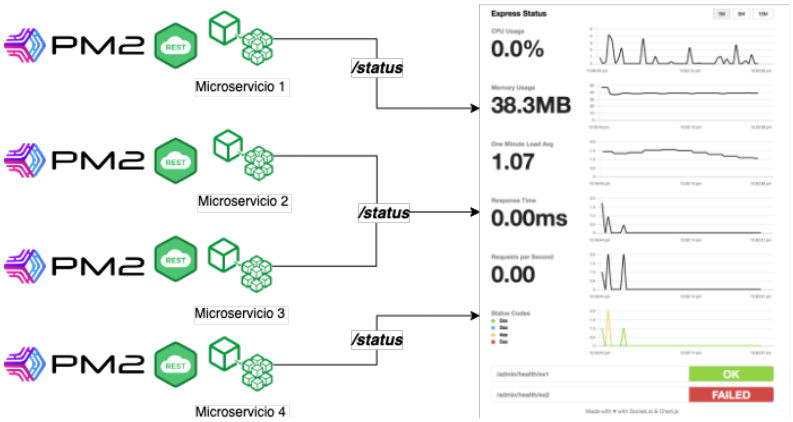

Fig. 32. PM2 para la gestión de cada microservicio y Express Status Monitor para las métricas en tiempo real.

6.2.6.5 Node.js

Node.js es un entorno Open Source que se propuso como núcleo para los microservicios debido a su naturaleza asíncrona y no bloqueante. Esto permitió desarrollar aplicaciones escalables y de gran rapidez, ya que es capaz de procesar múltiples conexiones de forma simultánea. Node.js utiliza el lenguaje de programación JavaScript y el motor de Google V8, lo que lo convierte en uno de los entornos de ejecución con mayor crecimiento para el desarrollo de aplicaciones web o de escritorio. Además, cuenta con una buena gestión de paquetes gracias a NPM, que contiene paquetes y librerías con soluciones prediseñadas listas para su uso.

En el ambiente tecnológico de la empresa Kradac Cia. Ltda., se utiliza Node.js para el desarrollo de sus soluciones tecnológicas actuales y futuras, obteniendo excelentes resultados. Esto hace que la utilización de Node.js sea una razón más para su inclusión en la presente propuesta, ya que se aprovecharía la experiencia previa y se maximizará los beneficios de la tecnología en la arquitectura propuesta, cada microservicio de desarrollará en Node.js como se visualiza en el diagrama que se realizó en la Fig. 1.

Fig. 33. Node.js como núcleo para el desarrollo de los microservicios

6.2.7 Componente Clientes

Los clientes para la presente propuesta fueron Ktaxi cliente y Ktaxi conductor, estos utilizaron la arquitectura basada en microservicios en la nube para mejorar algunos aspectos técnicos y tecnológicos que retrasan el crecimiento de los mismo debido a la arquitectura monolítica actual.

6.3 **FASE 3: Diseñar una arquitectura basada en microservicios en la nube para el aplicativo Ktaxi.**

6.3.1 Diagrama de componentes generales de la arquitectura propuesta

El diseño de la arquitectura propuesta incorporó todos y cada uno de los componentes descritos en el objetivo anterior. Como resultado, se creó una arquitectura escalable que puede aumentar la demanda de funcionalidades o recursos sin afectar a los ya existentes o a su rendimiento actual. Esto se debe a que cada microservicio contiene su propia funcionalidad y se encuentra aislado de los demás. Asimismo, se mejoró la agilidad para responder a los cambios del mercado gracias a la implementación de microservicios independientes que pueden ser desarrollados e implementados por cualquier desarrollador en cualquier lenguaje de programación. De esta manera, se facilita la adaptación de la organización a nuevas circunstancias, tecnologías o retos y asegura su competitividad en el mercado.

En la Fig. 34, se puede observar el diseño general de la arquitectura propuesta, de manera general se indicó como los aplicativo o cliente se conectan a un único punto de acceso denominado API Gateway este componente se lo estableció como un mecanismo capaz de recibir solicitudes e identificar a que microservicios desea acceder, los microservicios diseñados ejecutan una acción y devuelven una respuesta al cliente, si necesita ejecutar una acción de otro microservicio se implementó el envío de mensajes mediante RabbitMQ indicando la acción a ejecutarse. Para comprobar que el diseño de la arquitectura propuesta funcionó se desarrolló un prototipo cuyo diseño está descrito en la Fig. 35, en donde se implementaron los componentes del diseño general. Y el flujo que se siguió para el desarrollo se lo describe en la Fig. 36.

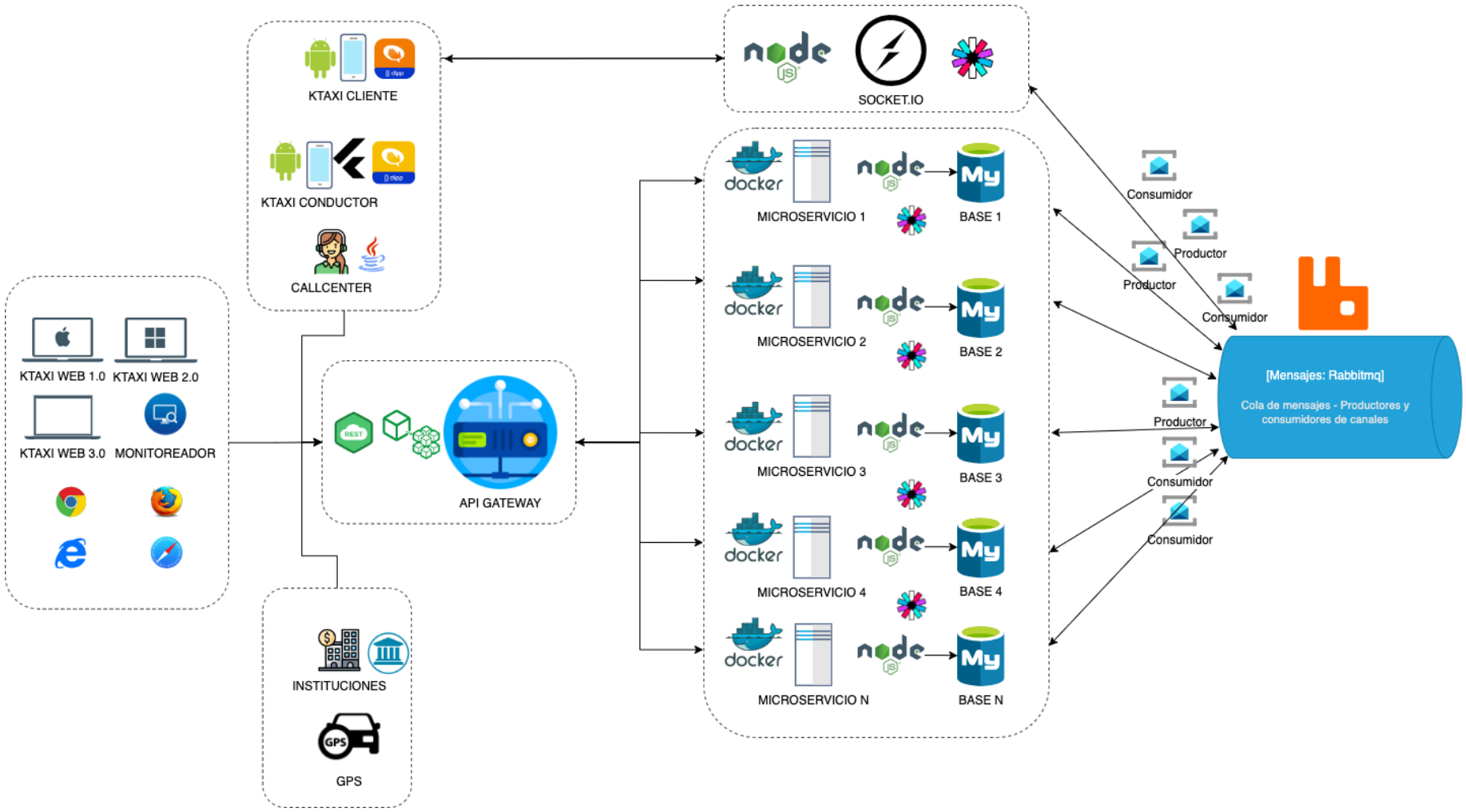

Fig. 34. Diagrama general de la arquitectura de microservicios propuesta

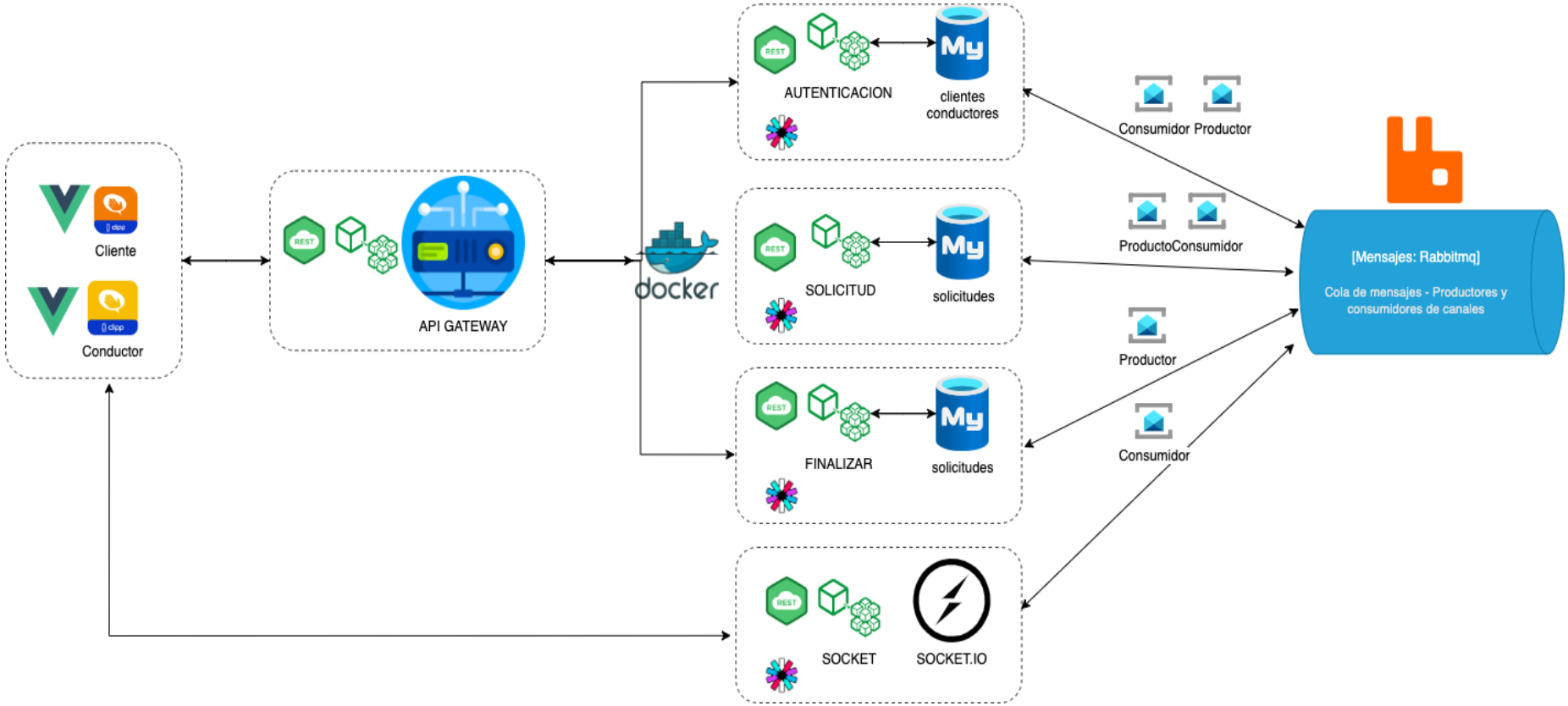

Fig. 35. Prototipo propuesto para validar la arquitectura propuesta

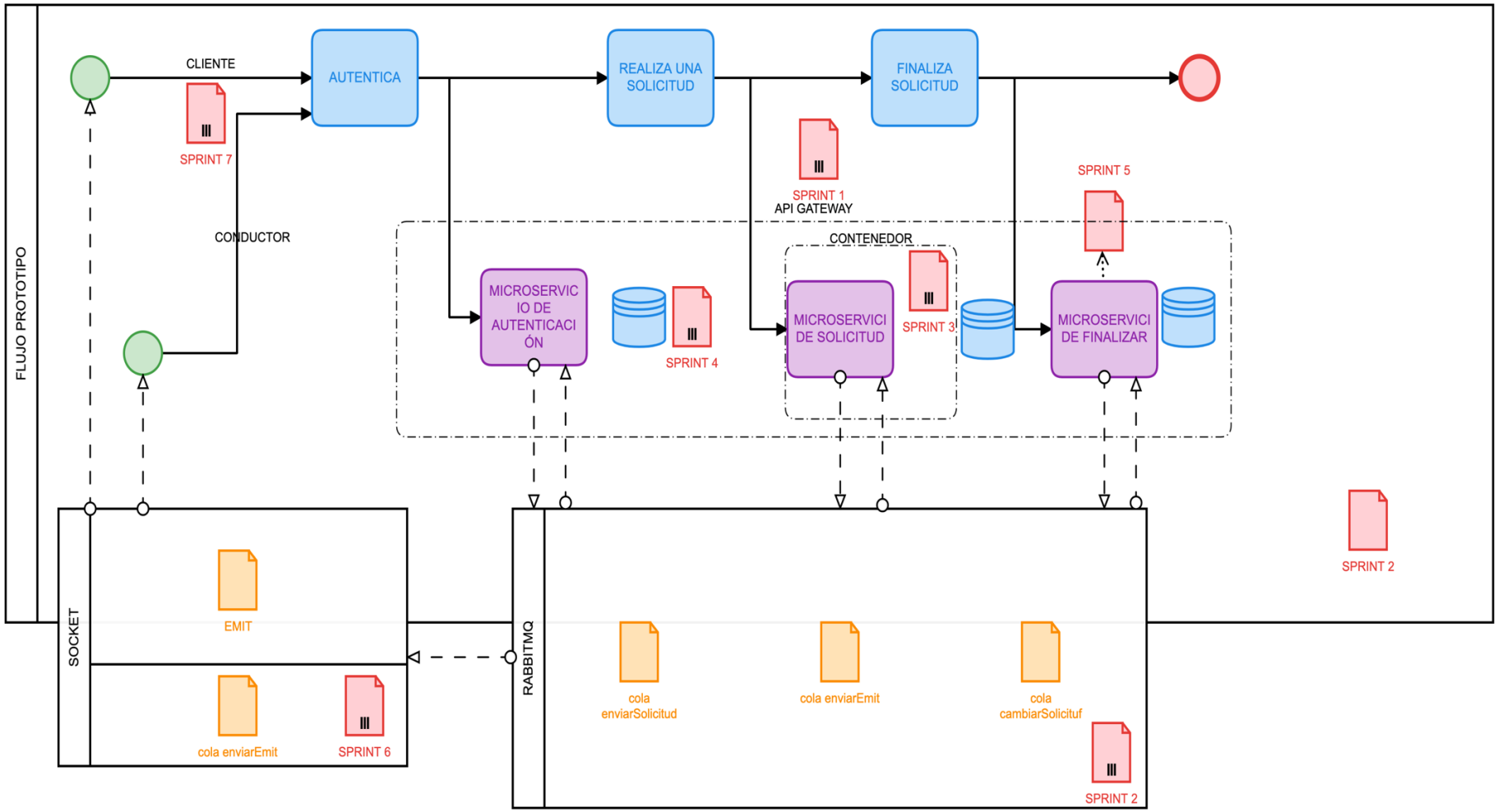

Fig. 36. Diagrama de procesos que muestra las etapas que siguió el prototipo propuesto

6.3.2 Diseño del componente API Gateway

La Fig. 37, detalla el diagrama que se realizó para el componente API Gateway, se puede observar como varios usuarios del aplicativo Ktaxi envían peticiones al API Gateway y este a su vez las envía a los microservicios, a continuación, se detalla las funciones principales que API Gateway deberá realizar dentro de la arquitectura de microservicios:

- Recibir peticiones de los clientes del aplicativo Ktaxi
- Identificar el punto de acceso contenido en la petición
- Verificar el proxy y puerto asignados al punto de acceso
- Redirigir la petición al microservicio de destino
- Recibir la petición al cliente del microservicio una vez se resuelva

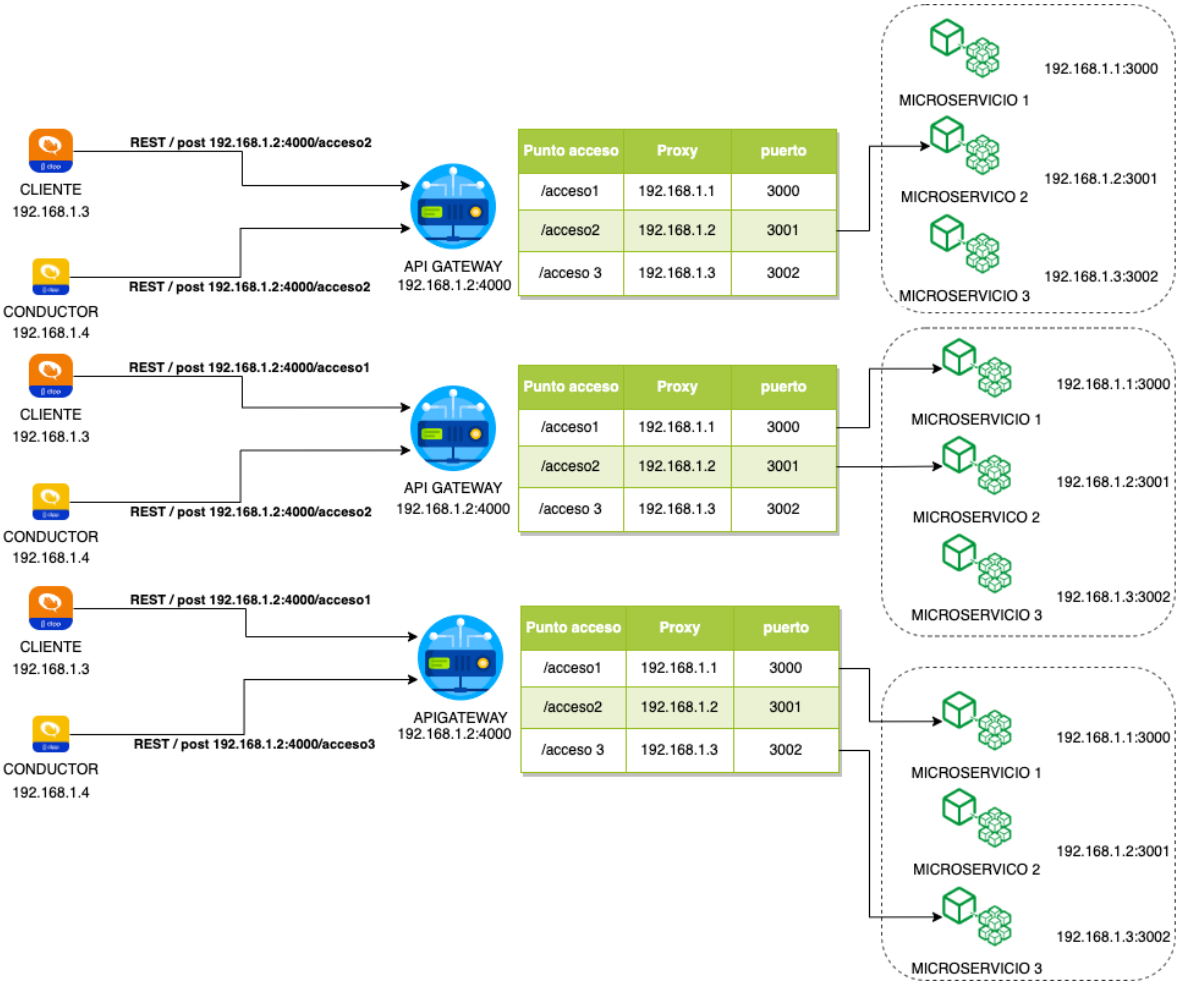

Fig. 37. Componente API Gateway para la arquitectura propuesta

Para el desarrollo del componente API Gateway dentro del prototipo, se utilizó Node.js y se configuró tres puntos de acceso para poder identificar las solicitudes entrantes por los clientes y poder redirigirlas en base a la configuración establecida, en la Fig. 38, se puede observar cómo se configuró API Gateway hacia el punto de acceso denominado ***/solicitud*** que hace referencia a la IP ***169.62.217.189*** en el puerto ***3003*** que es en donde se encuentra ejecutándose el microservicio de solicitudes, adicionalmente se incorporó algunos mecanismos de validación por si existe algún error o inconveniente con el microservicio y garantizar el servicio.

```
"/solicitud": {
        protected: false,
        target: "http://169.62.217.189:3003/",
        changeOrigin: false,
        pathRewrite: {
            [`^/solicitud`]: "/solicitud",
        },
        logLevel: 'debug',
        onError(err, req, res) {
            res.writeHead(500, {
                'Content-Type': 'text/plain'
            });
            res.end('Erro API de solicitud, por favor inténtelo más tarde.' + err);
        },
        onProxyReq(proxyReq, req, res) {
            console.log('Enviado cabeceras al microservicio de solicitudes')
            if (req.body) {
                let bodyData = JSON.stringify(req.body);
                proxyReq.setHeader('Content-Type', 'application/json');
                proxyReq.setHeader('Content-Length', Buffer.byteLength(bodyData));
                proxyReq.write(bodyData);
            }
        }
    },
```

Fig. 38. API Gateway de la arquitectura propuesta

Cada configuración del punto de acceso dependerá mucho de la lógica de negocios en este caso también se configuró una variable denominada ***protected,*** que se utilizó para una validación adicional al acceso del microservicio, este nos permitió definir si es de ambiente público o de ambiente privado en donde será necesaria la autenticación previa para validar su acceso. Para el prototipo se configuró 3 puntos de acceso hacia 3 microservicios ejecutándose como servicios independientes, dichos puntos de acceso se pueden observar el la Fig. 39.

Punto acceso	Proxy	puerto
/solicitud	169.62.217.189	3003
/autenticacion	169.62.217.189	3002
/finalizar	64.226.112.105	3003

Fig. 39. Puntas de acceso API Gateway prototipo

6.3.2.1 Seguridad API Gateway

Para controlar el acceso se implementó con Node.js un API REST con el punto de acceso **/login** como se puede visualizar en la Fig. 40, que creará una sesión de autenticación y la agregará en las cabeceras de cada petición entrante para cada cliente, este acceso será configurable mediante la configuración establecida para el API Gateway y se aplicó como medida de protección para aquellos microservicios que no son de entorno público.

```
app.get("/login", (req, res, next) => {
  if (!req.session.authenticated) {
    req.session.authenticated = 1;
    console.log(req.session)
    req.session.save(() => {
      res.send({
        en: 1,
        m: 'Autenticado con éxito'
      });
    });
  } else {
    res.send({
      en: 2,
      m: 'Usuario ya autenticado con éxito'
    });
  }
});
```

Fig. 40. API REST para el acceso al API Gateway

6.3.3 Diseño del componente para la comunicación entre microservicios

En la Fig. 41, se puede observar el diagrama que se realizó para la integración de RabbitMQ como mecanismo para comunicación entre los microservicios, se puede observar como un microservicio envía o publica un mensaje que generalmente se los conoce como

PRODUCTORES, hacía RabbitMQ que a su vez los clasifica mediante un intercambio en una cola de mensajes y los envía a otros microservicios también denominados **CONSUMIDORES** para su posterior tratamiento.

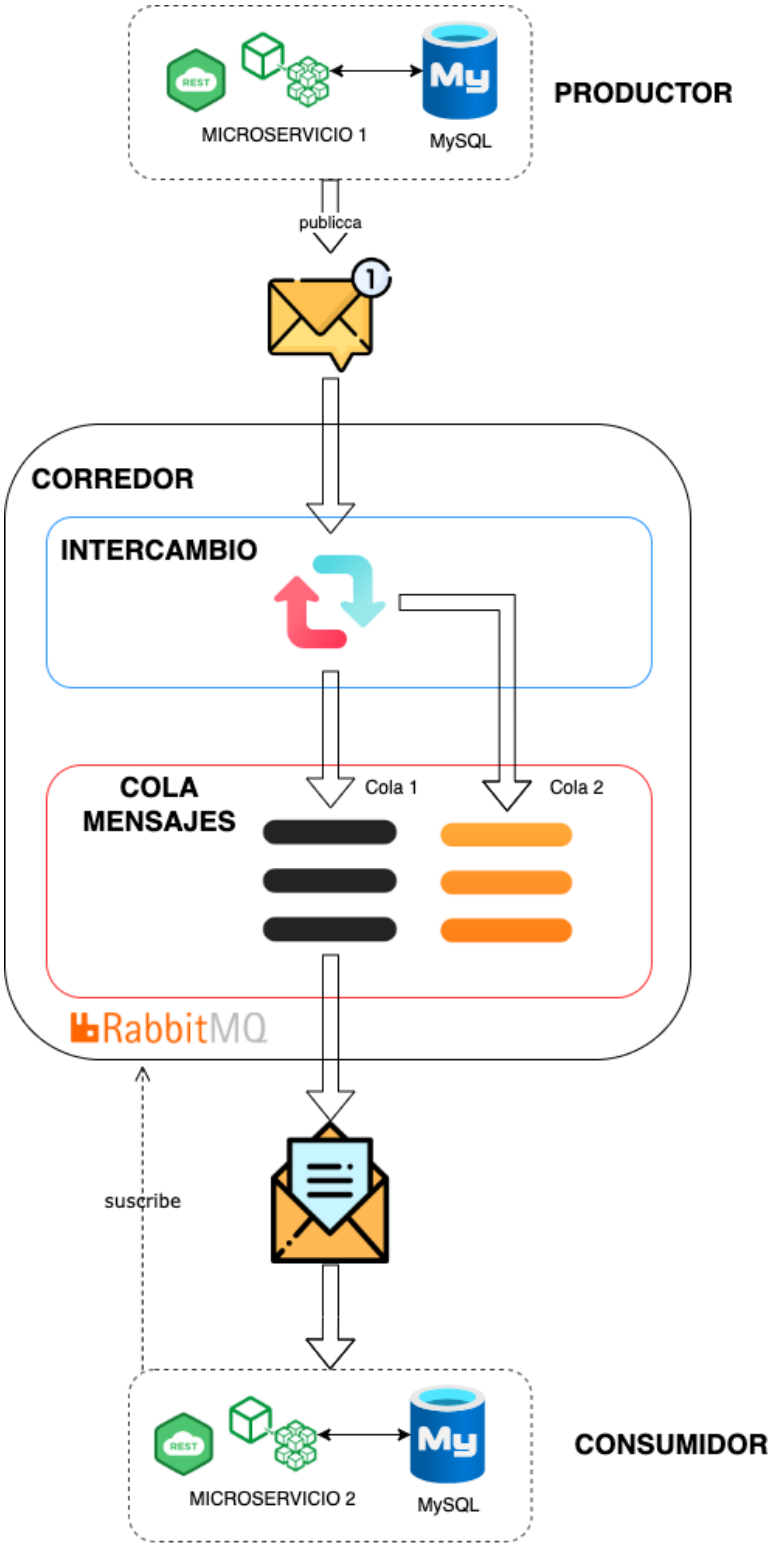

Fig. 41. RabbitMQ para la comunicación entre microservicios de la arquitectura propuesta

Para el prototipo propuesto se creó 3 colas de mensajes definidos en la Fig. 42, uno para solicitar al microservicio de autenticación que asigne un conductor a una solicitud, otro para notificarle al microservicio de socket que debe enviar un mensaje al cliente y conductor por la asignación de una solicitud y el último para cambiar el estado de una solicitud cuando haya finalizado.

Descripción general				Mensajes			Tarifas de mensajes		
Nombre	**Tipo**	Características	**Estado**	**Listo**	**Desnudo**	**Total**	**entrante**	**entregar / recibir**	**reconocer**
asignarConductor	clásico	D	inactivo	0	0	0	0.00/segundo	0.00/segundo	0.00/segundo
cambioSolicitud	clásico	D	inactivo	0	0	0	0.00/segundo	0.00/segundo	0.00/segundo
enviarEmit	clásico	D	inactivo	0	0	0	0.00/segundo	0.00/segundo	0.00/segundo

Fig. 42. Colas de mensajes para la comunicación de microservicios

En la Fig. 43, se puede observar la conexión que se implementó en del microservicio de solicitud al servicio de RabbitMQ, para lo cual se necesitó la clave, el usuario, la dirección en donde se implementó el servicio y el puerto en el cual está levantado, una vez se lo conectó con éxito se creó un canal de comunicación entre el microservicio y el servicio RabbitMQ, este canal fue el encargado de evitar y recibir mensajes a las colas registradas en su configuración, para el prototipo se configuró las colas de mensajes ***asignarConductor*** como PRODUCTOR y ***cambioSolicitud*** como CONSUMIDOR**.**

Adicionalmente se configuró algunos parámetros para garantizar que los mensajes no se pierdan si el microservicio o RabbitMQ no está funcionando, esto permitió guardar la información hasta que el servicio esté nuevamente en funcionamiento.

- **durable**: true -> Se definió en true para guardar los mensajes y las colas en memoria si el servicio de rabbitMQ se detiene por alguna
- **ack**(mensaje) -> Se definió ACK para permitir avisar a rabbitMQ que el mensaje ya fue procesado y se puede eliminar
- **noAck**: Se definió en false para recuperar mensajes perdidos si por A o B razones se desconecta automáticamente se vuelven a enviar.

```
amqp.connect('amqp://admin:admin@64.226.112.105:5672', function (error0, connection) {
    if (error0) {
        console.log('ERROR NO SE PUDO CONECTARSE CON RABBIT')
        console.log(error0)
        throw error0;
    }
    connection.createChannel(function (error1, channel) {
        console.log('rabit fue conectado correctamente')
        if (error1) {
            throw error1;
        }
        rabbit = channel;
        var queue = 'asignarConductor';
        var queue1 = 'cambioSolicitud';

        rabbit.assertQueue(queue, {
            durable: true
        });
        rabbit.assertQueue(queue1, {
            durable: true
        });

        rabbit.consume(queue1, function (msg) {
            var d = JSON.parse(msg.content.toString())
            console.log(d)
            solicitud.actualizarSolicitudConductor(d);
            setTimeout(function () {
                console.log(" [x] Done");
                rabbit.ack(msg);
            }, 10000);
        }, {
            noAck: false
        });
    });

});
```

Fig. 43. Conexión al servicio de RabbitMQ para la gestión de mensajes del microservicio solicitud

La Fig. 44, muestra el diagrama de secuencia que se realizó para la comunicación del prototipo desarrollado, los tres microservicios fueron conectados al servicio de RabbitMQ y se comunican para enviar mensajes a otros microservicios o escuchar los mensajes enviados por otros microservicios. Esto permitió tener una comunicación asíncrona cuando se necesitó ejecutar una funcionalidad definida por otro microservicio y garantizó que no existan bloqueos debido a que los mensajes se envían sin esperar una respuesta de su ejecución.

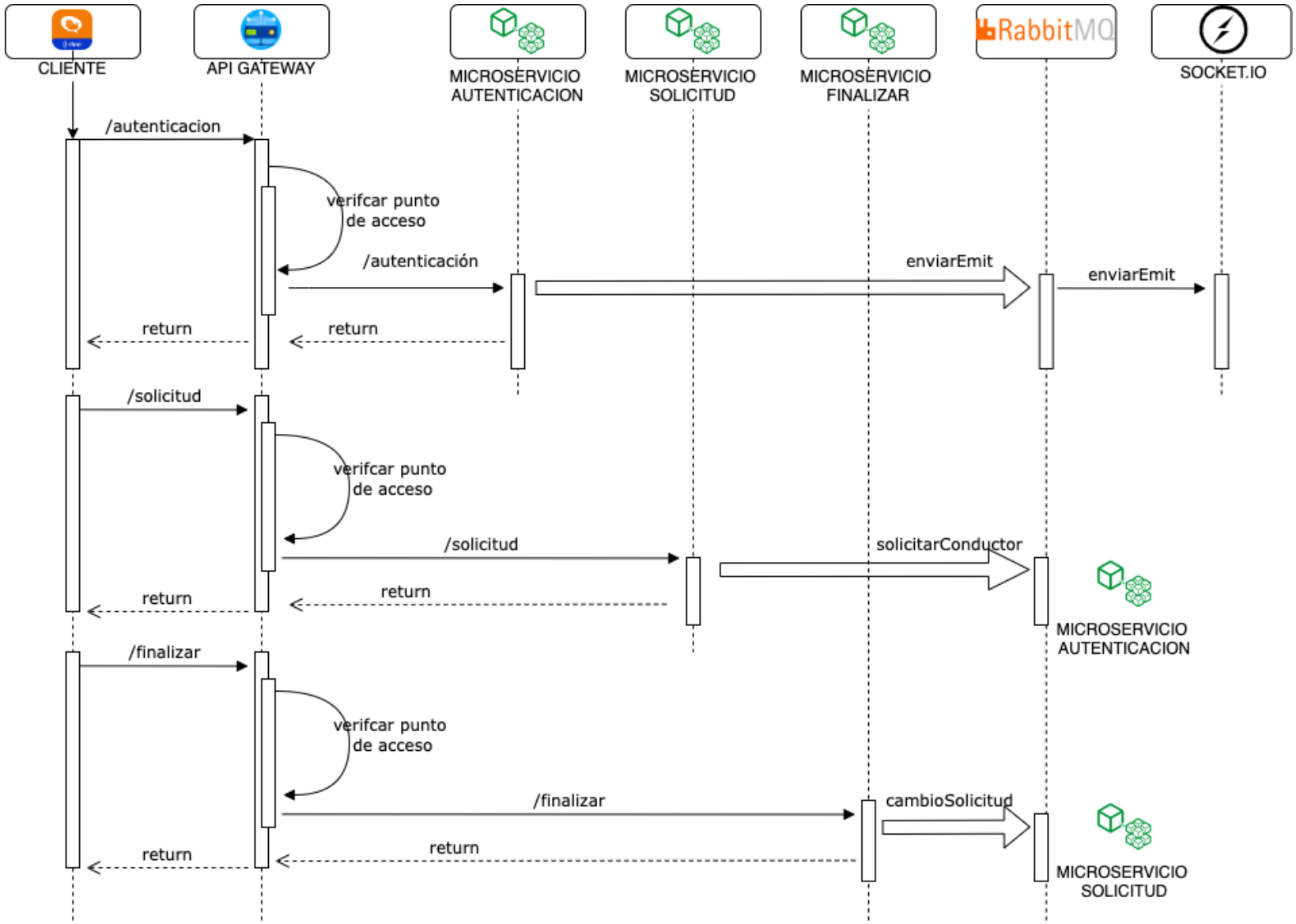

Fig. 44. Diagrama de secuencia para el envío de mensajes utilizando RabbitMQ dentro de la arquitectura propuesta

6.3.4 Diseño del componente para la base de datos

En la Fig. 45, se puede observar el diagrama que se realizó para definir el almacenamiento en donde se especificó que la gestión y almacenamiento de la información es gestionada por el microservicio, solo este podrá acceder y modificar dicha información garantizando la disponibilidad de los demás microservicios si alguna base de datos llegará a fallar. También se puede observar que cada microservicio independiente y aislado puede utilizar cualquier tipo de base de datos referente al modelo de negocios de cada institución o empresa.

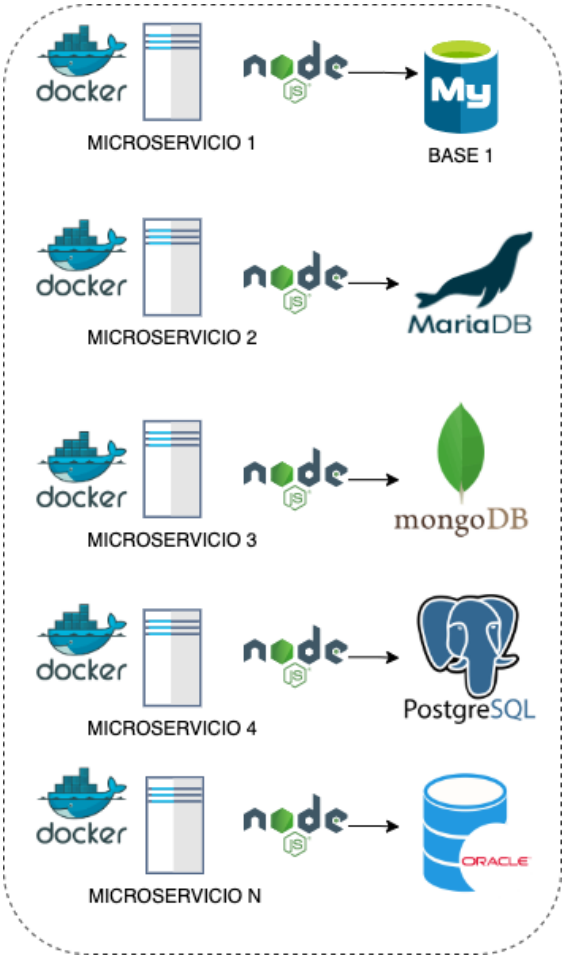

Fig. 45. Componente para la gestión del almacenamiento de información

Para el prototipo propuesto se implementó una base de datos relacional MySQL en cada uno de los microservicios desarrollados, se configuró una instancia para acceder a su base de datos para lo cual se necesitó el usuario, la contraseña, el nombre de la base de datos, la dirección IP en donde fue instalado y especificar el dialecto en este caso mysql, como se puede visualizar el la Fig. 46, además también se puede definir varias configuraciones dependiendo el ambiente de ejecución para el prototipo establecido se configuró para development.

Fig. 46. Conexión a la base de datos de los microservicios

Se marca en color rojo para evitar la divulgación de información confidencial del usuario de la imagen.

6.3.5 Diseño del componente para los contenedores

La Fig. 47, detalla el diagrama que se realizó para Dockerizar los microservicios se puede observar que se utilizó Docker Hub y que es necesario especificar el uso de las imágenes de Node.js y PM2 que se requieren para levantar los microservicio, una vez creada la imagen mediante un archivo denominado Dockerfile será necesario actualizarla al repositorio de Docker Hub para poder descargarla y ejecutarla.

También se puede observar que la base de datos no está contenida como imagen dentro del contenedor dado que dentro del diseño la gestión de la base de datos se la diseñó como un componente externo a los microservicios.

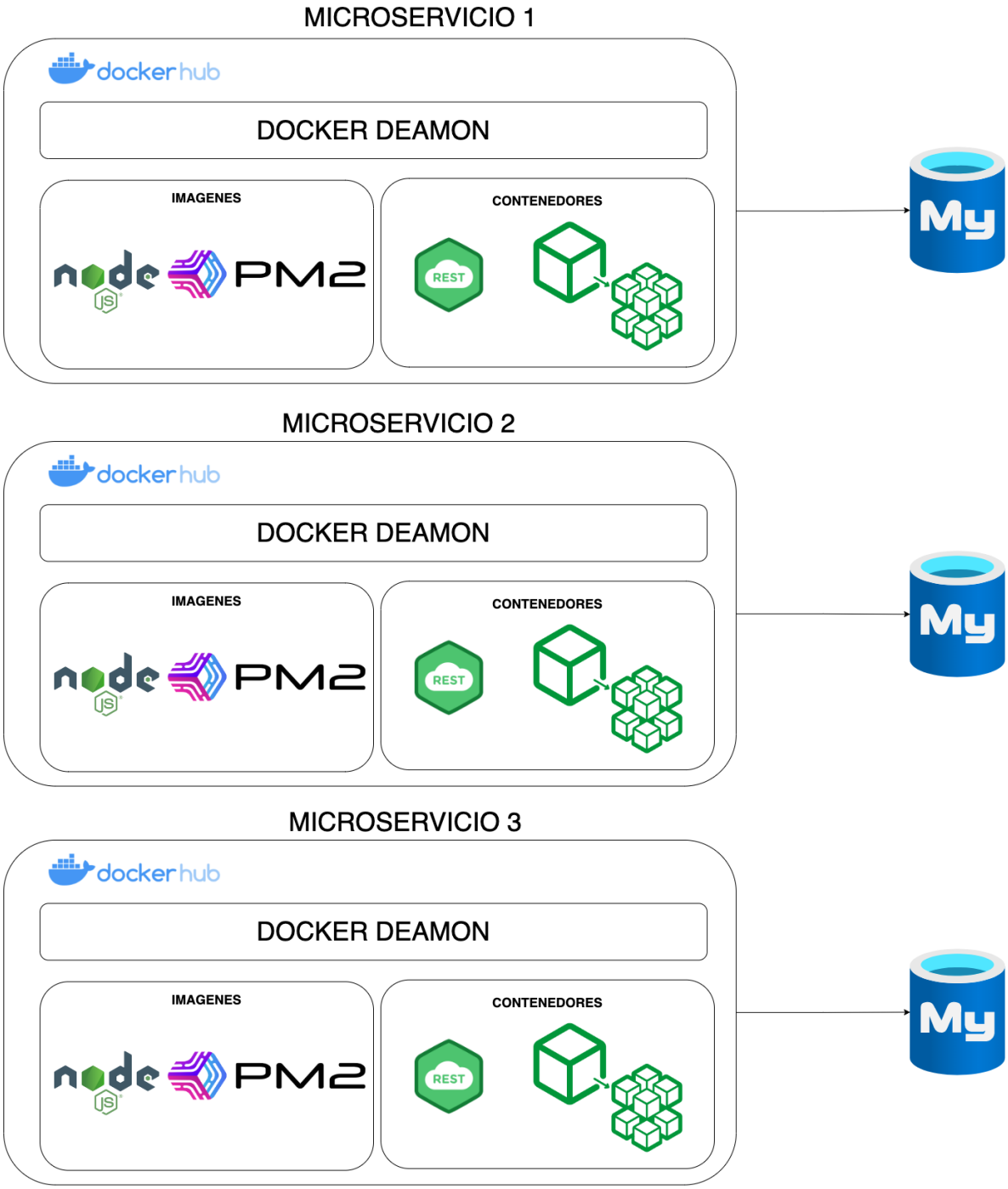

Fig. 47. Diagrama de contenedores Docker

Para el prototipo que se desarrolló se utilizó el archivo Dockerfile como se puede observar en la Fig. 48, para Dockerizar el microservicio de solicitud, en donde se definió el uso de Node.js en su versión 18 y PM2, también las instrucciones para el ensamblaje de la imagen las cuales se encuentran detalladas en el **Anexo *9.*** Opciones para la definición del Dockerfile, para la creación se utilizó el comando docker build que se encargó de que el sistema siga las instrucciones especificadas en el archivo Dockerfile y genere la imagen.

```
FROM node:18-alpine
FROM keymetrics/pm2:latest-alpine
ENV NODE_ENV=production

WORKDIR /app

COPY ["package.json", "package-lock.json*", "./"]
COPY pm2.json .

ENV NPM_CONFIG_LOGLEVEL warn
RUN npm install pm2 -g

RUN npm install --production

COPY . .

RUN ls -al -R

CMD [ "pm2-runtime", "start", "pm2.json" ]
```

Fig. 48. Archivo Dockerfile para la definición de las imágenes e instrucciones a ejecutarse para su despliegue

Una vez creada la imagen se procedió a subirla a Docker Hub como se puede visualizar en la Fig. 49, y descargó en el servidor del microservicio solicitud con el comando docker run. Cabe recalcar que la ventaja principal del uso de Docker es que permitió empaquetar el microservicio junto con todas sus dependencias en un paquete portátil y autónomo, para no depender de las tecnologías del servidor.

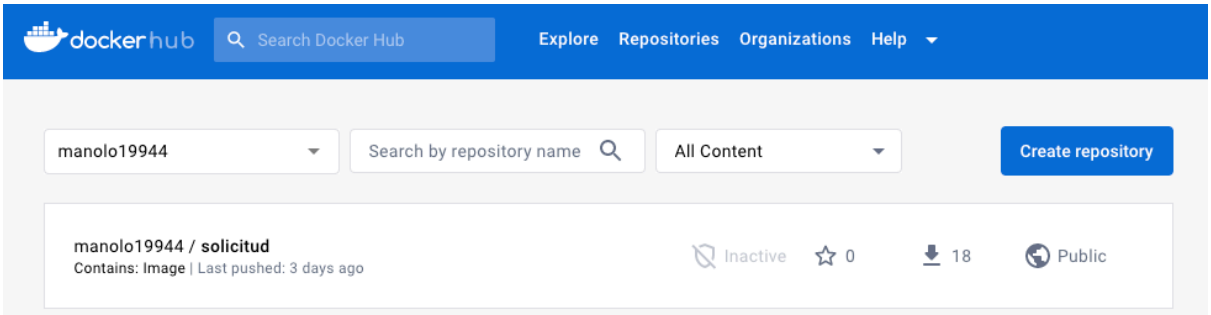

Fig. 49. Imagen Docker almacenada en Docker Hub

6.3.6 Diseño del componente microservicios

En la Fig. 50, se detalla el diagrama que se realizó para los microservicios, se puede visualizar que se definió el uso de Node.js , JWT, PM2 y Express Monitor y que se integran con RabbitMQ y MySQL, también se puede observar que deberán estar Dockerizados para su ejecución.

MICROSERVICIO	TECNOLOGÍAS	INTEGRACIÓN
MICROSERVICIO 1	Node.js, JWT, PM2, Express Monitor	RabbitMQ, MySQL
MICROSERVICIO 2	Node.js, JWT, PM2, Express Monitor	RabbitMQ, MySQL
MICROSERVICIO 3	Node.js, JWT, PM2, Express Monitor	RabbitMQ, MongoDB
MICROSERVICIO 4	Node.js, JWT, PM2, Express Monitor	RabbitMQ, MongoDB

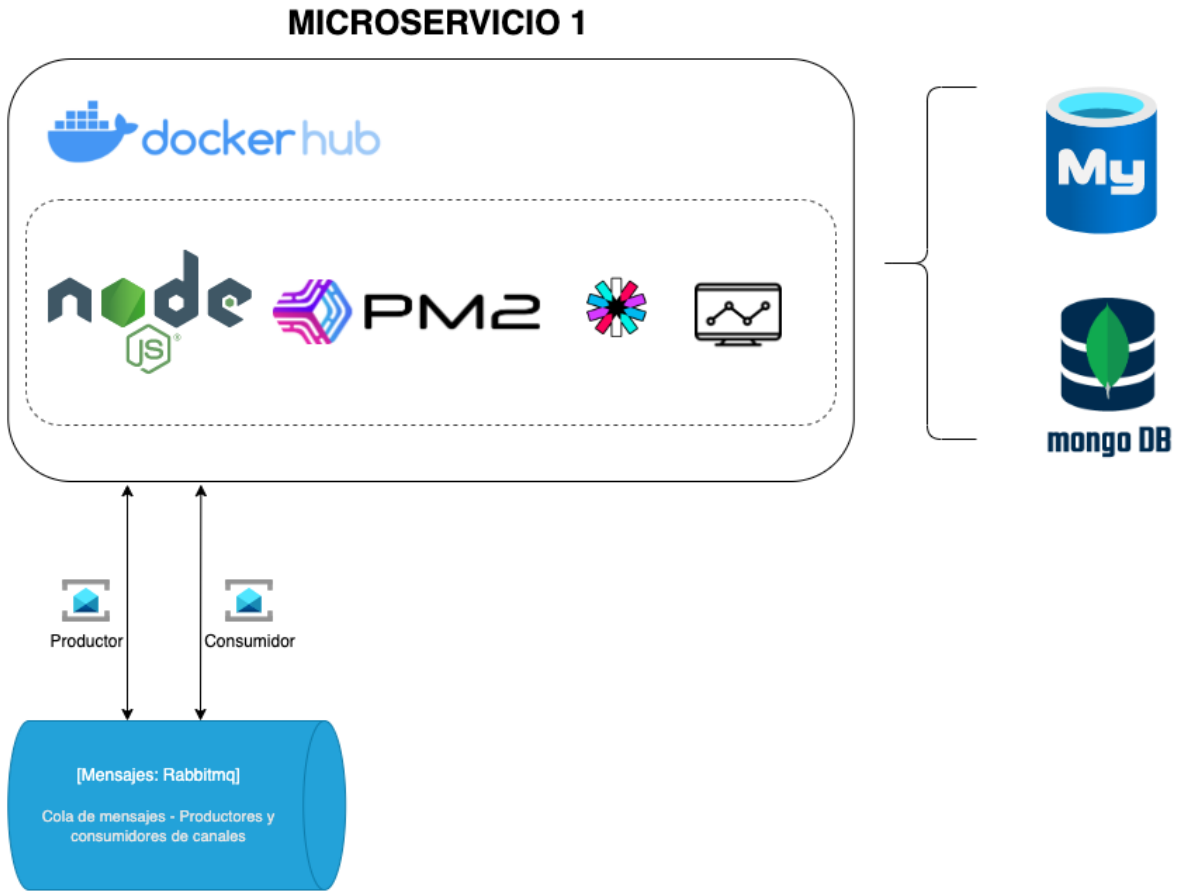

Fig. 50. Tecnologías e integraciones de cada microservicio propuestos para la arquitectura

Los microservicios del prototipo planteado fueron desarrollados con Node.js, JWT como estándar para la creación de Tokens seguros para la propagación de identidad que se enviaron en las cabeceras a los microservicios, su generación y validación por el microservicio de autenticación se pueden observar en la Fig. 51, también se implementó PM2 para el monitoreo de los recursos y el estado del microservicio y finalmente se configuró Monitor Express para un revisión interna del funcionamiento de las APIs definidas en su lógica.

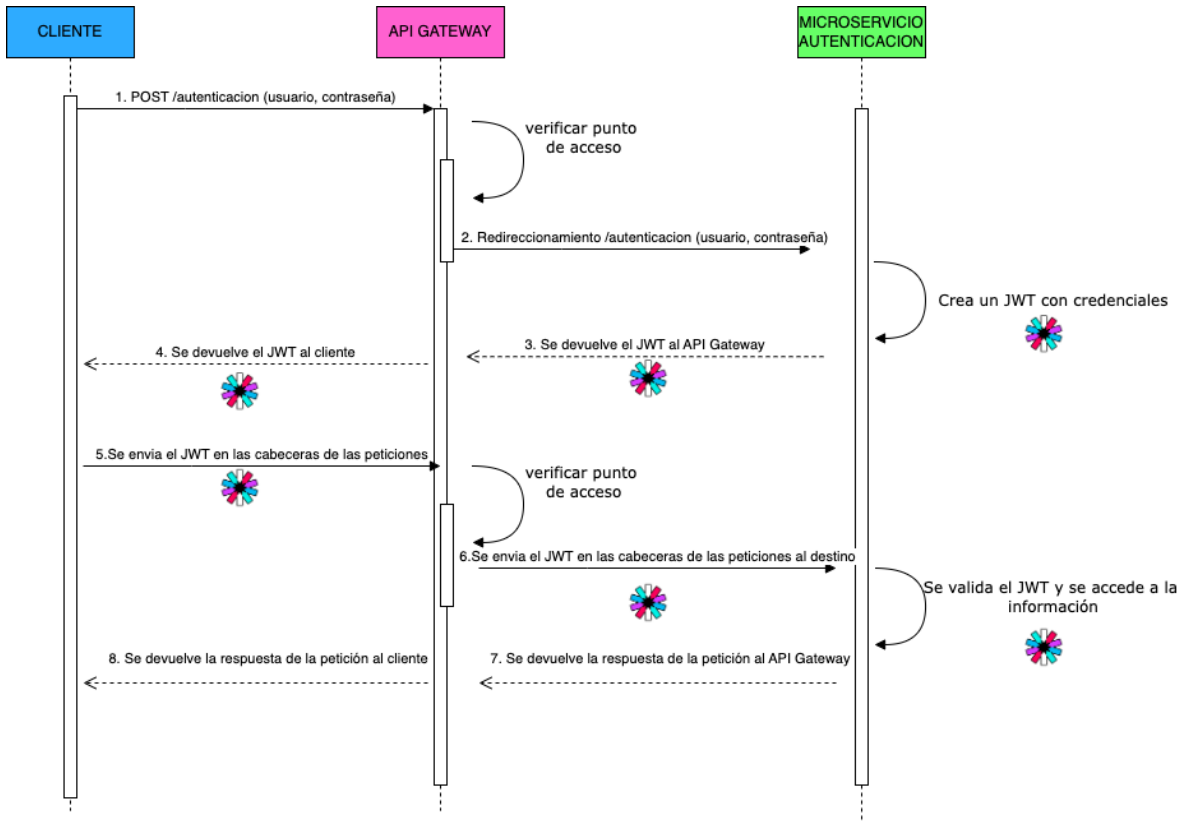

Fig. 51. Creación y validación de tokens seguros con JWT para el prototipo propuesto

En la Fig. 52, se puede observar los puntos finales API REST que se implementó en el microservicio finalizar.

```
app.post('/finalizar/solicitud/finalizarCliente/', solicitudController.finalizarCliente);
    app.post('/finalizar/solicitud/finalizarConductor/', solicitudController.finalizarConductor);
```

Fig. 52. API REST del microservicio de finalizar

Para la documentación de las APIs se utilizó Swagger que permitió crear una plataforma que ordena cada uno de nuestros métodos (get, put, post y delete) y categoriza nuestras operaciones. Cada uno de los métodos es expandible, y en ellos podemos encontrar un listado completo de los parámetros con sus respectivos ejemplos. En la Fig. 53, se puede visualizar la documentación del microservicio solicitud.

Fig. 53. Documentación del API del microservicio solicitud

6.3.7 Componente para la comunicación en tiempo real

La Fig. 54, detalla el diagrama que se realizó para poder manejar las actualizaciones en tiempo real de las aplicaciones con el uso de Socket.IO, en el diseño se puede observar que las aplicaciones están conectadas al microservicio de socket y tienen implementados escuchas que permiten recibir información del socket y actualizarse de forma automática, cabe recalcar que las aplicaciones deberán crear una instancia hacia el microservicio de socket para generar una conexión y poder enviar la información. También se puede visualizar que Socket.io tiene una instancia de conexión hacia RabbitMQ esto debido a que será configurado como CONSUMIDOR de mensajes para poder ejecutar el envió de emits.

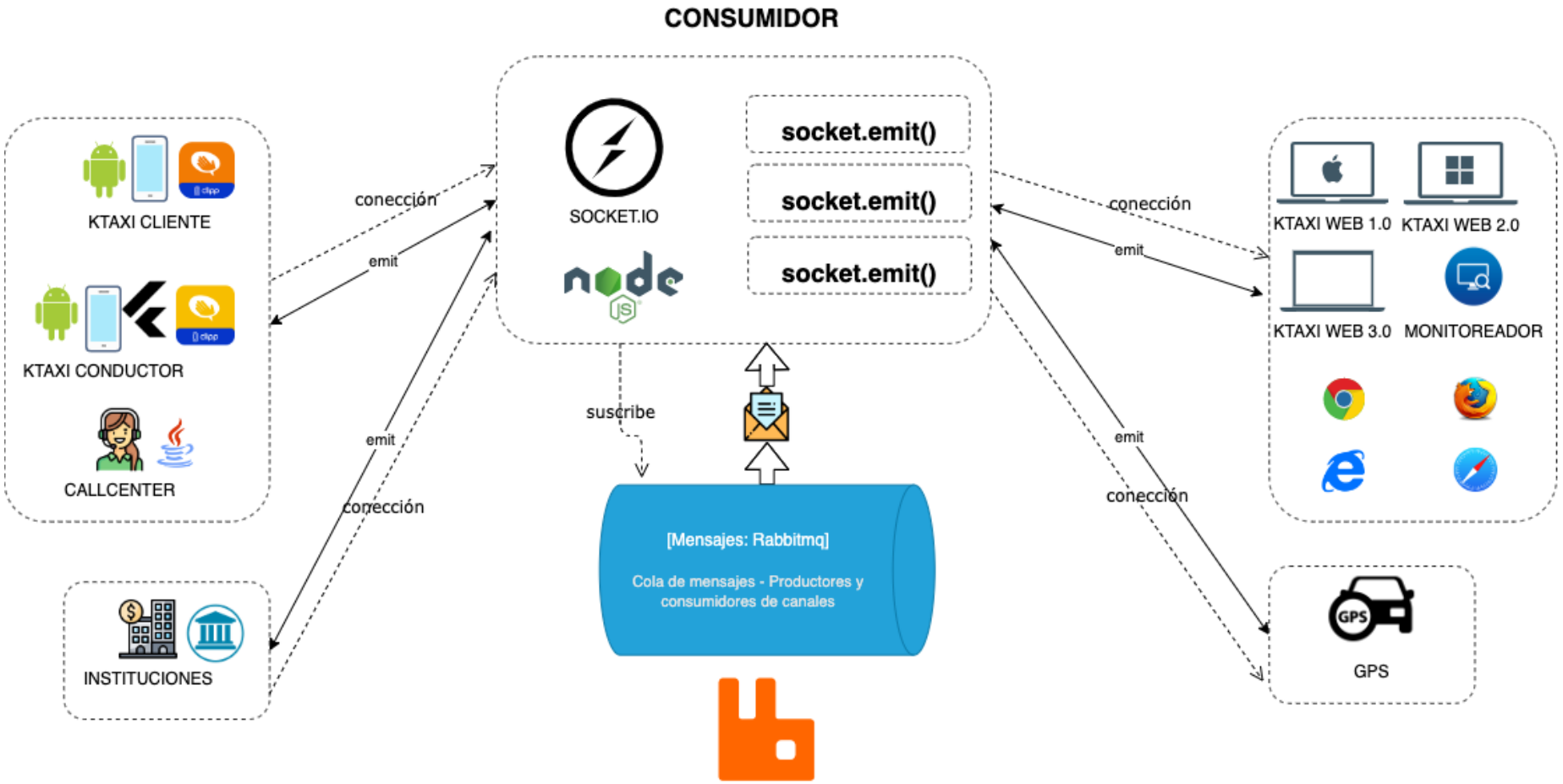

Fig. 54. Socket.IO para la actualización de las aplicaciones en tiempo real de la arquitectura propuesta

Para el prototipo propuesto las aplicaciones clientes se conectaron al microservicio de socket para poder recibir información para su representación lo que mejoró la experiencia del cliente al poder visualizar la interacción del sistema con la información en tiempo real. Cada microservicio envía un mensaje por medio del canal ***enviarEmit***, el microservicio de socket se configuró como un **CONSUMIDOR** de esta cola de mensajes y envía la información a los clientes conectados para la actualización de la información, mediante el emit ***dataConductor,*** como se puede observar en la Fig. 55.

```
function enviarEmit(data) {
    let d = JSON.parse(data);
    config.ejecutarsqlSelect(
'SELECT socketId FROM  ktaxiSocket.socket where idUsuario = ? ;', [d.
idCliente], function (res) {
        if (res.en == 1) {
            if (res.data.length > 0) {
                socketio.to(res.data[0].socketId).emit("dataConductor", d);
            }
        }
        else
            console.log('SQL no EJECUTADO - REGISTRO no GUARDADO')
    });
    config.ejecutarsqlSelect(
'SELECT socketId FROM  ktaxiSocket.socketConductor where idConductor = ? ;'
, [d.id], function (res) {
        if (res.en == 1) {
            console.log('Socket del conductor encontrado')
            if (res.data.length > 0) {
                socketio.to(res.data[0].socketId).emit("dataCliente", d);
            }
        }
        else
            console.log('SQL no EJECUTADO - REGISTRO no GUARDADO')
    })
}
```

Fig. 55. Envío de información por parte del microservicio socket

6.3.8 Componente cliente

El diseño de la arquitectura que se realizó definió a los aplicativos clientes de Ktaxi descritos en la Fig. 56, estos son los responsables de consumir todos y cada uno de los microservicios antes mencionados, cabe recalcar que los clientes pueden ser cual aplicativo, sistema, API e incluso otro microservicio que se conecta al API Gateway para obtener cierta información o realizar alguna acción.

Fig. 56. Clientes que se consumen microservicios

Para el prototipo propuesto se desarrolló 2 aplicaciones web una que simuló ser un cliente y otra ser un conductor en el lenguaje de programación JavaScript bajo el framework Vue.js, estos clientes se conectaron al API Gateway y consumieron los microservicios detallados anteriormente mismos que se pueden visualizar en el **Anexo *13*.** Prototipo desarrollado para la arquitectura de microservicios.

6.3.9 Diseño del diagrama de comunicación de la arquitectura de microservicios

La Fig. 57, describe el diagrama que se realizó para la comunicación de los diferentes componentes de la arquitectura propuesta, en la cual se propuso el uso de certificados de seguridad SSL para que el envío de la información sea bajo el protocolo HTTPS y la devolución de la información se realizará bajo el formato JSON.

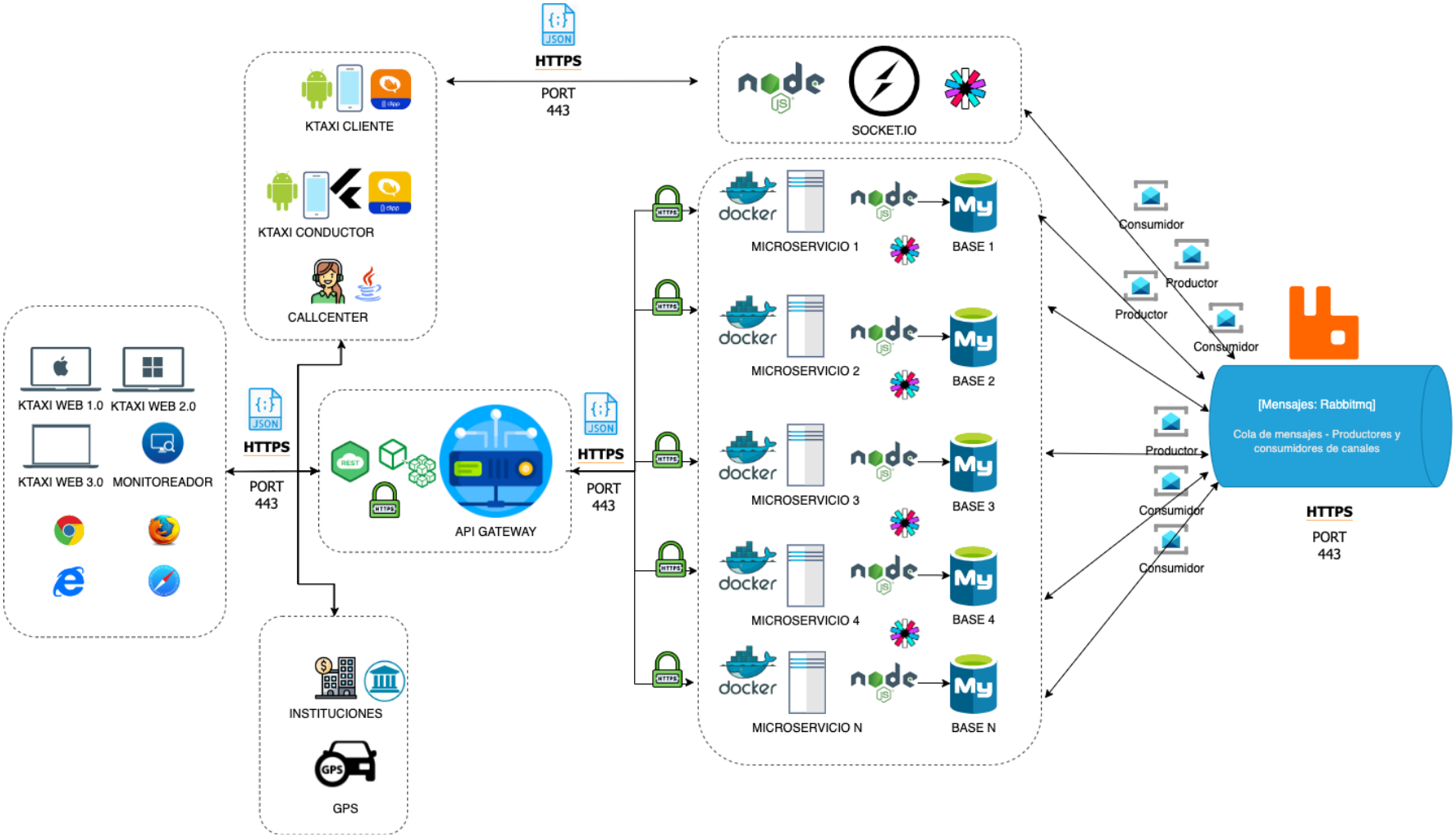

Fig. 57. Diagrama de la comunicación de la arquitectura de microservicio propuesta

Para el prototipo desarrollado no se utilizó una comunicación HTTPS por cuestiones económicas de adquirir certificados de seguridad, sin embargo, al ser un prototipo nos enfocamos más en la parte funcional del mismo, para comprobar que todos los microservicios funcionen de manera correcta, para lo cual se utilizó una comunicación HTPP y JSON que permitió demostrar su funcionamiento.

6.3.10 Diagrama de integración de componentes

El la Fig. 58, se detalla el diagrama que se realizó para indicar cómo se integraron todos los componentes seleccionados el cual se detalla a continuación:

1. Los aplicativos **clientes** de Ktaxi como: Ktaxi cliente, Ktaxi conductor y CallCenter, tienen establecida la conexión hacia el API Gateway

2. El **API Gateway** tiene configurado los puntos de acceso hacia los microservicios, así como el punto de acceso a cada uno de ellos.

3. Los **microservicios** se ejecutan en **contenedores** Docker y tienen una instancia del componente de comunicación RabbitMQ para el envío y consumo de mensajes, y gestiona la información de su propia **base de datos,** los microservicios al ser independiente pueden estar desarrollado en cualquier lenguaje de programación para el diseño propuesto se Node.js.

4. **RabbitMQ** se comunica mediante el envió de mensajes a los demás microservicios y al Socket que se configuró como CONSUMIDOR, mediante la configuración de leído de mensajes sabrá cuando los debe eliminar y cuando los debe almacenar hasta que exista un consumidor conectado.

5. **Socket.IO** envía emits hacia las aplicaciones clientes como: Ktaxi cliente, Ktaxi conductor y CallCenter, para la actualización de la información, también funciona como consumidor de mensajes de RabbitMQ para saber cuándo comunicarse.

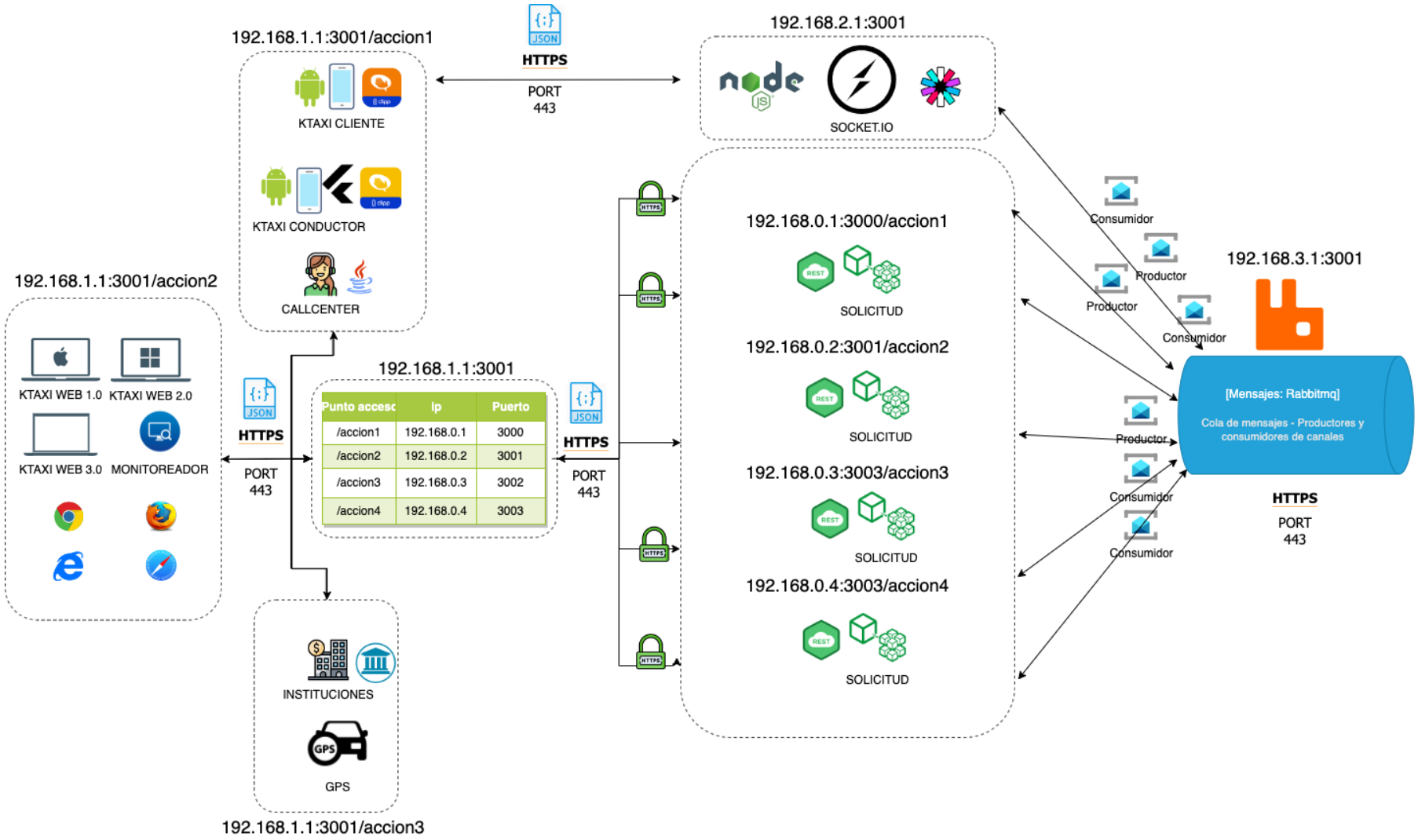

Fig. 58. Diagrama de integración de componentes de la arquitectura propuesta

6.3.11 Diagrama para la seguridad de la arquitectura de microservicios

La Fig. 59, detalla el diagrama que se realizó para la seguridad en el cual se propone que se deberá ser mediante la configuración de un firewall para el control de acceso a cada servidor donde se ejecutan los microservicios, adicionalmente se recomienda la restricción por IP entre el API Gateway y los puntos de acceso configurados, esto garantiza que solo el API Gateway pueda acceder a los microservicios, adicionalmente como se definió en los puntos anteriores el uso de JWT para la validación de credenciales mediante tokens de seguridad

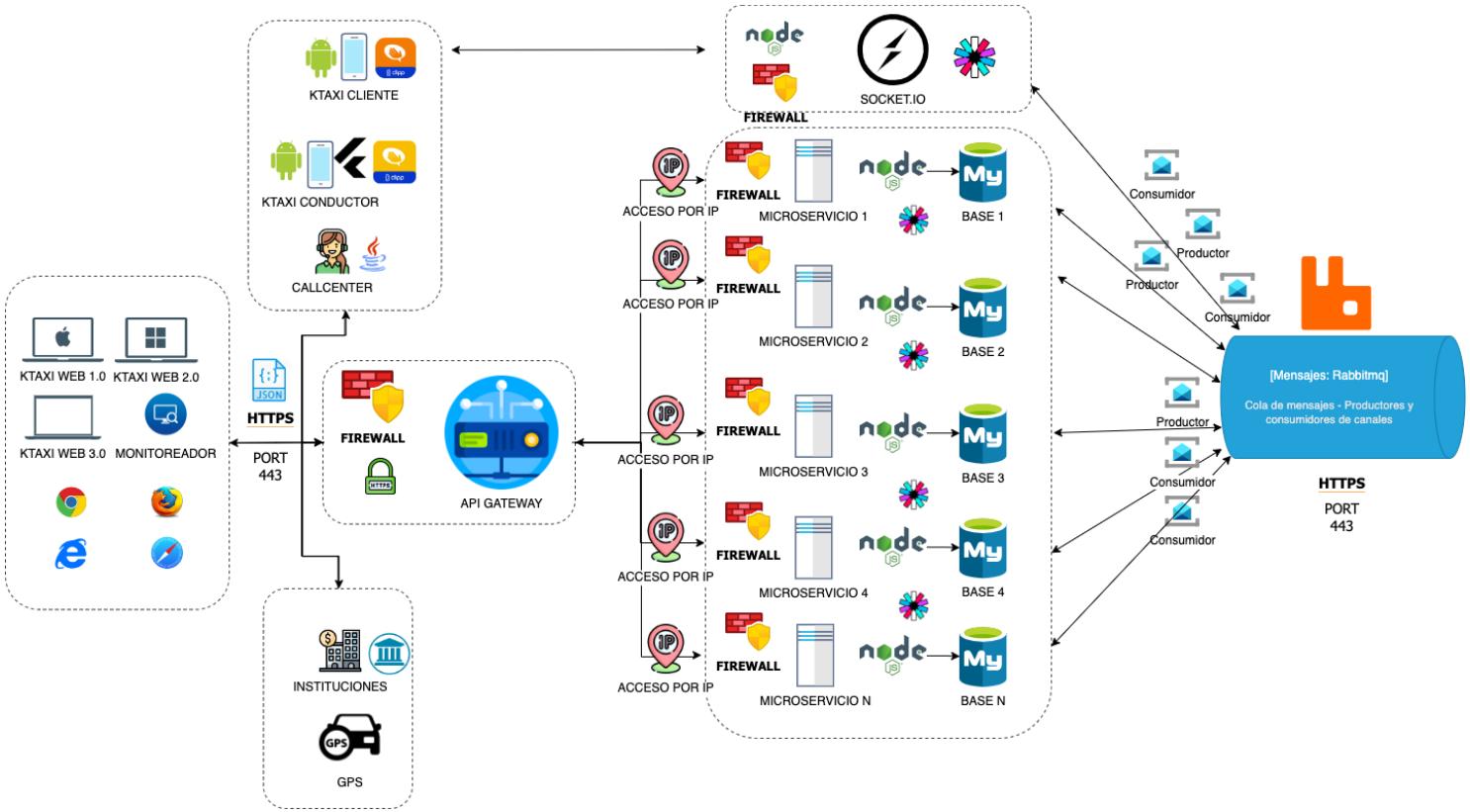

Fig. 59. Diagrama de seguridad de la arquitectura propuesta

6.3.12 Pruebas del diseño y prototipos definidos

El desarrollo de las pruebas de los diagramas y del prototipo propuesto se realizó en el departamento de TI de la empresa Kradac Cia. Ltda., previa autorización por parte de la empresa como se puede ver en el **Anexo *6*.** Aprobación para la socialización de la arquitectura de microservicios, en donde se indicó como están definidos los diagramas, cuales fueron los componentes que se escogieron para la arquitectura, el funcionamiento de cada uno de ellos y la tecnología en la cual se lo realizó.

Para validar el prototipo se desarrolló 2 sistema web en Vue.js para que sean los clientes que van a utilizar los microservicios como se ve en el **Anexo *13*.** Prototipo desarrollado para la arquitectura de microservicios, cada uno de ellos envió peticiones al API Gateway y este los redirigió al microservicio de destino, los microservicios fueron desarrollados con Node.js y se encuentran en 2 servidores diferentes al igual que la base de datos que se encuentra como componente externo. La instancia de Socket y RabbitMQ también fueron levantados como servicios aparate para poder indicar el principio de independencia de los microservicios.

Luego de las pruebas y en base a una encuesta de satisfacción que se puede ver en el ***Anexo 10.*** *Encuesta de evaluación de la arquitectura basada en microservicios en la nube*. se obtuvo de manera general una aceptación de la arquitectura del 93.75%, como se puede observar en el ***Anexo 11.*** *Análisis de los resultados obtenidos de la encuesta*.

En la que se hizo hincapié en la pregunta 1 que se puede visualizar en la Fig. 60 y en la pregunta 2 que se puede visualizar en la Fig. 61 que fueron el objeto del presente TT.

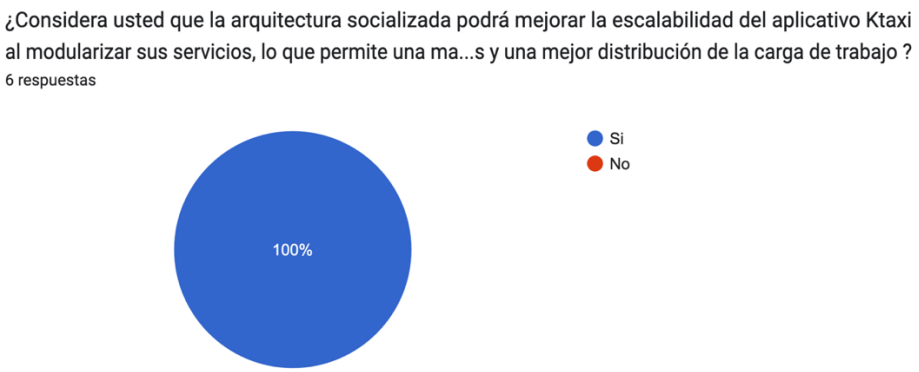

Fig. 60. Resultado de la escalabilidad de la arquitectura para el aplicativo Ktaxi

Fig. 61. Resultado de la agilidad de la arquitectura para el sistema Ktaxi

De la población definida paras las pruebas y que evaluaron la misma el 100% confirmo que la arquitectura propuesta mejoraría la escalabilidad del sistema Ktaxi, y el 83.33% confirmó que la arquitectura propuesta mejoraría la agilidad del sistema Ktaxi, considerando que el 16.7% de la segunda pregunta se enfocó en un aspecto de tiempo más no de viabilidad.

7. Discusión

7.1 Evaluación de la propuesta

Para la evaluación de la arquitectura basada en microservicios propuesta para el aplicativo Ktaxi, se examinará el cumplimiento de cada uno de los objetivos específicos que conforman la base para alcanzar el objetivo general del presente TT.

- **Objetivo Específico 1: Realizar un análisis del estado actual del aplicativo Ktaxi mediante un diagnóstico interno en base a la información de la empresa Kradac Cia. Ltda., para poder determinar su arquitectura actual.**

Para lograr este objetivo, se inició con las solicitudes de autorización pertinentes para el acceso a la información, así como la asignación de un asesor técnico por parte de la empresa Kradac Cia. Ltda., una vez obtenida la autorización se realizó un análisis general de la información proporcionada para conocer el funcionamiento y estado actual del aplicativo Ktaxi. Luego del posterior análisis se empezó a identificar cuales son los componentes que conforman el aplicativo Ktaxi llegando a la conclusión de que la arquitectura actual es una arquitectura monolítica, puesto que toda su lógica de negocios como: recursos, métodos, funciones, APIS, entre otros, se encuentra en el mismo servidor. Después se diseñó el diagrama de la arquitectura actual, diagrama de componentes, diagrama de comunicación y seguridad, también se detalló aspectos técnicos como: versiones, sistemas operativos, protocolos de seguridad, protocolos de comunicación, entre otros, de la arquitectura de la organización junto al asesoramiento del **Ing. Ricardo Jumbo** responsable técnico encargado, quién fue la persona que constantemente verificaba que toda la información sea correcta. Finalmente, se evaluó la arquitectura actual del sistema Ktaxi por parte del asesor técnico quien dio fe de que la arquitectura diseñada representa el estado real del aplicativo Ktaxi como se puede ver en el **Anexo *5*.** Aprobación de la arquitectura actual del aplicativo Ktaxi.

- **Objetivo Específico 2: Analizar los principales componentes que intervienen en el diseño de una arquitectura basada en microservicios en la nube mediante una revisión bibliográfica de información para identificar los componentes a utilizar.**

El desarrollo de este objetivo comenzó con el análisis de información utilizando la técnica de revisión bibliografía en fuentes como empresas de tecnológicas líderes en el mercado, revisiones de literatura, reportes y consultoras tecnológicas la cual permitió establecer los

componentes que se utilizan para el desarrollo de una arquitectura basada en microservicios, posteriormente con este referente se evaluó las más utilizadas y se concluyó el uso de las siguientes: API Gateway, contenedores, base de datos, clientes, comunicación, microservicios y Socket.IO esta última seleccionada como referencia personal debido a la lógica de negocios de la empresa Kradac Cia. Ltda.

Posteriormente se prosiguió con el justificativo bibliográfico de los componentes seleccionados y de las tecnologías para el desarrollo de cada una de ellas derivando en el uso de Node.js, API REST, control de versión, JWT, PM2, Monitor Express y Vue.js, luego se detalló cada componente indicando como funciona y como se implementará dentro de la arquitectura, se realizaron algunos diseños para tener una mejor comprensión al momento su diseño en la arquitectura final.

- **Objetivo Específico 3: Diseñar una arquitectura basada en microservicios en la nube para el aplicativo Ktaxi.**

Para alcanzar este objetivo se comenzó con la especificación del diagrama general de la arquitectura basada en microservicio, la cual incorporó a todos los componentes definidos en el punto anterior, indicando como interactúan cada uno de ellos como un todo, también se diseñó un diagrama general del prototipo diseñado para poder validar los diseños especificados.

Posteriormente se detalló la implementación de cada componente mediante el desarrollo de un diagrama detallado de como funcionan y como se deberán integrar, de igual manera se desarrolló un prototipo funcional de cada componente utilizado, indicando el uso de Node.js, JWT, PM2, Docker e integrando RabbitMQ y MySQL. Para validar que el prototipo propuesto funciona se solicitó autorización para poder tener una reunión con el departamento de TI de la empresa Kradac Cia. Ltda., una vez aprobada se reunió con el equipo de backend y se expuso la arquitectura propuesta, luego una vez finalizada la reunión se implementó una encuesta para validar si consideran que el diseño y prototipo presentados satisface las necesidades del aplicativo Ktaxi, obteniendo como resultado una aceptación del diseño y prototipo del 93.75%, considerando que el porcentaje disminuye por consideraciones técnicas como el tiempo de implementación por parte de la empresa y la gestión de una sola base de datos cuestiones fuera del alcance del diseño, como se puede ver en el **Anexo *11*.** Análisis de los resultados obtenidos de la encuesta.

8. Conclusiones

- En base a los resultados obtenidos del análisis de la información solicitada en la empresa Kradac Cia. Ltda., y en paralelo con la asesoría del Ing. Ricardo Jumbo se determinó que el aplicativo Ktaxi objeto de estudio tiene implementada una arquitectura monolítica con tres servidores para maximizar la gestión de la carga, aprobada en el **Anexo *5***.

- El desarrollo del prototipo propuesto en el presente TT integrado con los diseños de cada componente seleccionados, permitió llevar a cabo una socialización en el departamento de TI de la empresa Kradac Cia. Ltda. con lo que se logró obtener una tasa del 93.7% de aceptación de la arquitectura, y que servirá como base para los nuevos desarrollos dentro de la empresa.

- El análisis de la información bibliográfica sobre los componentes que intervienen en el desarrollo de una arquitectura basada en microservicios del **Anexo *8***, permitió establecer los componentes fundamentales (API Gateway, contenedores, base de datos, comunicación, microservicios y clientes), que debe llevar la arquitectura para su correcto funcionamiento.

- El diseño detallado de los diagramas de cada componente permitió desarrollar el prototipo de una manera rápida y ágil ya que se especifica todos los aspectos técnicos como la comunicación, integración y seguridad, así como los aspectos tecnológicos como el lenguaje de programación (Node.js) y herramientas para la gestión dinámica de la implementación (PM2, JWT, Monitor Express).

- Para medir el rendimiento de los microservicios del prototipo propuesto se utilizó pruebas de carga, donde se comprobó que los tiempos de respuesta son óptimos al tener un promedio de 110 milisegundos con un rango de 30.000 solicitudes por minuto o 500 solicitudes por segundo. Por lo tanto, se ratifica que los microservicios soportan grandes cantidades de solicitudes.

- El presente TT comenzó como un trabajo técnico – académico, sin embargo, el desarrollo de un prototipo para validar los diseños establecidos generó una gran expectativa en el departamento de TI de la empresa Kradac Cia. Ltda., y que servirá como base para futuras arquitecturas basadas en microservicios.

9. Recomendaciones

- Los desgarradores de software conozcan y apliquen tecnologías de contenedores como Docker para la implementación de microservicios, ya que este ofrece características de creación, ejecución y administración de contenedores aislados con sus propias dependencias portátiles y fáciles de mantener.

- Al integrar un componente para los microservicios se debe leer la documentación oficial para poder elegir la mejor opción o configuración en base a las necesidades de su entorno personal o empresarial.

- Para el desarrollo de los diagramas se puede utilizar la herramienta en línea diagram.net que contiene diagramas básicos, de negocio, ingeniería, software, UML, entre otros.

- Utilizar versiones estables y actuales de los lenguajes de programación como las herramientas para que no existan problemas de implementación o incompatibilidad durante la creación de los microservicios

- Las aplicaciones que quieran migrar su arquitectura a un de microservicios deben realizar un análisis de viabilidad, costos y personal puesto que su implementación de forma correcta garantizara la escalabilidad del sistema, pero una implementación mala la empeorará.

- Para la integración y despliegue continuo se use el enfoque DevOps ya que este permite tener un control de todos los microservicios, puesto que si no es así la arquitectura será muy compleja de mantener

10. Bibliografía

[1] G. Blinowski, A. Ojdowska, and A. Przybylek, "Monolithic vs. Microservice Architecture: A Performance and Scalability Evaluation," *IEEE Access*, vol. 10, pp. 20357–20374, 2022.

[2] F. Ponce, G. Marquez, and H. Astudillo, "Migrating from monolithic architecture to microservices: A Rapid Review," *Proc. - Int. Conf. Chil. Comput. Sci. Soc. SCCC*, vol. 2019-Novem, 2019.

[3] V. A. Iranzo Jiménez, "Desarrollo de software basado en microservicios: un caso de estudio para evaluar sus ventajas e inconvenientes," 2018.

[4] "Comparación entre la arquitectura monolítica y la arquitectura de microservicios." [Online]. Available: https://www.atlassian.com/es/microservices/microservices-architecture/microservices-vs-monolith. [Accessed: 28-Mar-2023].

[5] G. Márquez, F. Osses, and H. Astudillo, "Review of Architectural Patterns and Tactics for Microservices in Academic and Industrial Literature," vol. 16, no. 9, 2018.

[6] D. Lopez and M. Edgar, "Arquitectura de Software basada en Microservicios para Desarrollo de Aplicaciones Web," in *Séptima Conferencia de Directores de Tecnología de Información, TICAL 2017 Gestión de las TICs para la Investigación y la Colaboración, San José, del XX al XX de julio de 2017*, 2017, pp. 5–7.

[7] A. Ćatović, N. Buzađija, and S. Lemes, "Microservice development using RabbitMQ message broker," *Sci. Eng. Technol.*, vol. 2, no. 1, pp. 30–37, 2022.

[8] F. M. García-Moreno, M. Bermúdez-Edo, M. J. Rodriguez-Fortiz, and J. L. Garrido, "Una Arquitectura Orientada a Microservicios y Dirigida por Eventos para el Desarrollo de Sistemas de eSalud Avanzados : Caso de Evaluación de Fragilidad en Mayores," *Sistedes 2019*, no. September, pp. 1–9, 2019.

[9] A. Akbulut and H. G. Perros, "Software Versioning with Microservices through the API Gateway Design Pattern," *2019 9th Int. Conf. Adv. Comput. Inf. Technol. ACIT 2019 - Proc.*, pp. 289–292, Jun. 2019.

[10] G. Verdier, Diego Rodriguez, "Implementación de Patrones de Microservicios," pp. 5–129, 2020.

[11] A. Marchese and O. Tomarchio, "Communication Aware Scheduling of Microservices-based Applications on Kubernetes Clusters," *Int. Conf. Cloud Comput. Serv. Sci. CLOSER - Proc.*, no. January, pp. 190–198, 2022.

[12] S. M. Velásquez Restrepo, J. D. Vahos Montoya, M. E. Gómez Adasme, A. A. Pino, E.

Juliett Restrepo-Zapata, and S. Londoño Marín, “A comparative review about traditional and modern software development methodologies,” *Rev. CINTEX*, vol. 24, no. 2, pp. 13–23, 2019.

[13] E. Haro, T. Guarda, A. Zambrano, and G. Ninahualpa, “Desarrollo backend para aplicaciones web, Servicios Web Restful: Node.js vs Spring Boot,” *Rev. Ibérica Sist. e Tecnol. Informação*, no. E17, pp. 309–321, 2018.

[14] A. Mishra and Z. Otaiwi, “DevOps and software quality: A systematic mapping,” *Comput. Sci. Rev.*, vol. 38, Nov. 2020.

[15] J. T. Zhao, S. Y. Jing, and L. Z. Jiang, “Management of API Gateway Based on Micro-service Architecture,” *J. Phys. Conf. Ser.*, vol. 1087, no. 3, 2018.

[16] Microsoft, “Comunicación en una arquitectura de microservicio.” [Online]. Available: https://learn.microsoft.com/es-es/dotnet/architecture/microservices/architect-microservice-container-applications/communication-in-microservice-architecture. [Accessed: 28-Mar-2023].

[17] B. Shafabakhsh, R. Lagerström, and S. Hacks, “Evaluating the impact of inter process communication in microservice architectures,” *CEUR Workshop Proc.*, vol. 2767, no. QuASoQ, pp. 55–63, 2020.

[18] Microsoft, “Microservicios en contenedores.” [Online]. Available: https://learn.microsoft.com/es-es/dotnet/architecture/maui/micro-services. [Accessed: 28-Mar-2023].

[19] MySQL, “MySQL para desarrolladores de software.” [Online]. Available: https://www.mysql.com/news-and-events/web-seminars/mysql-para-desarrolladores-de-software/. [Accessed: 28-Mar-2023].

[20] R. Gesteira, “Implementación de una arquitectura de microservicios para una red de sensores IoT sobre Arduino,” 2020.

[21] MySQL, “¿Por qué MySQL?” [Online]. Available: https://www.mysql.com/why-mysql/. [Accessed: 28-Mar-2023].

[22] Socket.IO, “Introduction | Socket.IO.” [Online]. Available: https://socket.io/docs/v4/. [Accessed: 28-Mar-2023].

[23] RabbitMQ, “Tutorial de RabbitMQ - Publicar/Suscribir.” [Online]. Available: https://www.rabbitmq.com/tutorials/tutorial-three-python.html. [Accessed: 27-Mar-2023].

[24] GitLab, “What are microservices?” [Online]. Available: https://about.gitlab.com/topics/microservices/#microservice-core-components.

[Accessed: 26-Mar-2023].

[25] Adservio, “What are Microservice Architecture – Features and Components.” [Online]. Available: https://www.adservio.fr/post/what-are-microservice-architecture-features-and-components. [Accessed: 26-Mar-2023].

[26] E. T. S. Ingeniería, S. Informáticos, E. Albertos Gómez, J. E. Pérez Martínez, and J. Díaz Fernandez, “UNIVERSIDAD POLITÉCNICA DE MADRID Arquitecturas Software para Microservicios: Una Revisión Sistemática de la Literatura Tutor: Co-Tutor,” 2018.

[27] Microsoft, “Estilo de arquitectura de microservicios - Azure Architecture Center.” [Online].Available:https://learn.microsoft.com/eses/azure/architecture/guide/architecture-styles/microservices. [Accessed: 26-Mar-2023].

11. Anexos

Anexo 1. Solicitud de acceso a la información del aplicativo Ktaxi

Loja, 15 de febrero del 2023

Sr.
Ing. Bruno Valarezo
CEO de la empresa Kradac Cia. Ltda.

Reciba un muy cordial saludo, a través de la presente, yo, **MANUEL STALIN ARMIJOS ORDÓÑEZ** con cédula de identidad **1105593238,** estudiante de la **Maestría en Ingeniería en Software** de la Universidad Nacional de Loja, previo a la obtención del título **Magíster en Ingeniería en Software** con resolución **RPC-SO-30-No.499-2019** solicito de la manera más atenta me ayuden con información técnica y operativa del aplicativo denominado **KTAXI** para lograr determinar cual es la arquitectura que maneja y cuales son los componentes que la integran, para con ello poder definir un diagrama general de la arquitectura actual.

Sin más que agregar, le agradezco por su atención y tiempo prestado y quedo atento a su respuesta.

Atentamente,

MANUEL STALIN ARMIJOS ORDÓÑEZ

1105593238

Estudiante de la **Maestría en Ingeniería en Software**

Resolución **RPC-SO-30-No.499-2019**

RECIBIDO
Por:
Fecha: 15-02-2023 Hora:

Anexo 2. Solicitud para la asignación de un asesor técnico para determinar la arquitectura de Ktaxi

Loja, 15 de febrero del 2023

Sr.
Ing. Bruno Valarezo
CEO de la empresa Kradac Cia. Ltda.

Reciba un muy cordial saludo, a través de la presente, yo, **MANUEL STALIN ARMIJOS ORDÓÑEZ** con cédula de identidad **1105593238,** estudiante de la **Maestría en Ingeniería en Software** de la Universidad Nacional de Loja, previo a la obtención del título **Magíster en Ingeniería en Software** con resolución **RPC-SO-30-No.499-2019** solicito de la manera más atenta la asignación de un asesor técnico referente al aplicativo denominado **KTAXI** para lograr determinar cual es la arquitectura que maneja y cuales son los componentes que la integran, para con ello poder definir un diagrama general de la arquitectura actual.

Sin más que agregar, le agradezco por su atención y tiempo prestado y quedo atento a su respuesta.

Atentamente,

MANUEL STALIN ARMIJOS ORDÓÑEZ

1105593238

Estudiante de la **Maestría en Ingeniería en Software**

Resolución **RPC-SO-30-No.499-2019**

RECIBIDO
Por:
Fecha: Hora:

Anexo 3. Aprobación de acceso a la información de Ktaxi y asignación de un asesor para determinar su arquitectura

clipp

Loja, 16 de febrero de 2023

Ing. .
Manuel Armijos.
Desarrollador de Software de la empresa Kradac Cia. Ltda.

Ciudad. -

De mis consideraciones. -

A través de la presente, me complace informarle que su petición para acceso a la información técnica y operativa, así como un asesor técnico a sido aprobada, se le asignó al **Ing. Ricardo Jumbo** para que le pueda ayudar con la información necesaria.

Espero que esta información sea de su interés y estoy a su disposición para cualquier duda o consulta adicional.

Atentamente,

KRADAC Cía. Ltda.
RUC 1191734609001

Ing. Soledad Armijos
Encargada del departamento de Talento Humano de la Empresa Kradac Cia. Ltda

KRADAC

Anexo 4. Solicitud de socialización del diseño de la arquitectura de microservicios

Loja, 19 de abril del 2023

Sr.
Ing. Bruno Valarezo
CEO de la empresa Kradac Cia. Ltda.

Reciba un muy cordial saludo, a través de la presente, yo, **MANUEL STALIN ARMIJOS ORDÓÑEZ** con cédula de identidad **1105593238,** estudiante de la **Maestría en Ingeniería en Software** de la Universidad Nacional de Loja, previo a la obtención del título **Magíster en Ingeniería en Software** con resolución **RPC-SO-30-No.499-2019** solicito de la manera más atenta se me permita tener una reunión con el departamento de T.I de su empresa para realizar la socialización de la arquitectura basada en microservicios en la nube propuesta, de igual manera la presentación de un prototipo funcional que permita validar su diseño y su implementación en un ambiente real.

Sin más que agregar, le agradezco por su atención y tiempo prestado y quedo atento a su respuesta.

Atentamente,

MANUEL STALIN ARMIJOS ORDÓÑEZ

1105593238

Estudiante de la **Maestría en Ingeniería en Software**

Resolución **RPC-SO-30-No.499-2019**

RECIBIDO

Anexo 5. Aprobación de la arquitectura actual del aplicativo Ktaxi

clipp

Loja, 19 de abril de 2023

Ing. .
Manuel Armijos.
Desarrollador de Software de la empresa Kradac Cia. Ltda.

Ciudad. -

De mis consideraciones. -

A través de la presente, yo, **Ricardo Fabián Jumbo Herrera,** con cédula de identidad **1104671738,** asignado como asesor técnico del aplicativo **Ktaxi** para el levantamiento de información, me complace informarle que he revisado el diseño de la arquitectura del aplicativo presentado por usted y puedo confirmar que cumple con los requisitos actuales del aplicativo, además de reflejar su realidad. Por tanto, se aceptado como una arquitectura monolítica válida.

Espero que esta información sea de su interés y estoy a su disposición para cualquier duda o consulta adicional.

Atentamente,

KRADAC Cía. Ltda.
RUC 1191734609001

Ing. Ricardo Jumbo
VP of Engineer del departamento de T.I de la empresa Kradac Cia. Ltda.

KRADAC

Anexo 6. Aprobación para la socialización de la arquitectura de microservicios

clipp

Loja, 19 de abril de 2023

Ing. .
Manuel Armijos.
Desarrollador de Software de la empresa Kradac Cia. Ltda.

Ciudad. -

De mis consideraciones. -

A través de la presente, me complace informarle que su petición para la socialización de su arquitectura basada en microservicios a sido aprobada misma que podrá realizarla en coordinación con el encargado de departamento de T.I.

Espero que esta información sea de su interés y estoy a su disposición para cualquier duda o consulta adicional.

Atentamente,

KRADAC Cía. Ltda.
RUC 1191734609001

Ing. Soledad Armijos
Encargada del departamento de Talento Humano de la Empresa Kradac Cia. Ltda.

KRADAC

Anexo 7. Descripción general de Clipp Taxi

“CLIPP TAXI”

La información contenida en este documento es confidencial y de propiedad de la KRADAC - CLIPP Ltda.

Elaborado por:	Equipo
Pineda Criollo Jessica Mariuxi	Clipp Taxi
Ramón Campoverde Elizabeth del Rocío	Clipp

1. Términos y abreviaturas

- **Usuario/pasajero:** Es aquella persona que realiza un viaje en taxi sin conducirlo de un punto o ubicación hacia otra.
- **Conductor**
- **Viaje:** Es el desplazamiento en taxi de un usuario/pasajero desde un origen a un destino predefinido.
- **Tarifa:** Se refiere a un monto monetario calculado, indicado y presentado como costo dado por un taxímetro, como consecuencia del viaje realizado en el taxi, basado en un costo fijo inicial (excluyendo costos suplementarios) y/o la distancia y/o el tiempo de duración del viaje.
- **Taxímetro:** Dispositivo de medida mecánico o electrónico instalado en los taxis, similar a un odómetro. Calcula el importe a cobrar en relación tanto a la distancia recorrida como en el tiempo transcurrido.
- **Peaje:** Es el pago correspondiente a los derechos de tránsito o circulación por determinados lugares, como algunas autopistas, puentes, túneles, aduanas, etc.
- **Estándar:** Son acuerdos documentados que contienen especificaciones técnicas o criterios precisos que son utilizados consistentemente, como reglas, guías o definiciones de características para asegurar que los productos, procesos y servicios cumplen con su propósito.
- **Protocolo:** Puede ser un documento o una normativa que establece cómo se debe actuar en ciertos procedimientos.
- **APP:** El término app es una abreviatura para referirse a una aplicación informática para dispositivos móviles y tabletas.

- **TAG:** El Tag o Televía es un dispositivo electrónico que permite detectar el paso de un vehículo por los pórticos de las autopistas urbanas de Santiago, con el propósito de realizar el cobro de los tránsitos efectuados en cada una de estas vías concesionadas.

2. Datos Generales

Nombre del Proyecto:

CLIPP TAXI

Fig. A7. 1. Presencia de Clipp Taxi

CLIPP TAXI

Clipp Taxi es un sistema completo de administración y gestión de despacho de unidades de taxi, permite acoplarse a las diferentes formas de trabajo de las operadoras de taxi en cada ciudad, pensando siempre en la experiencia de usuario, se trabaja con aplicativo móvil para

usuarios como conductores y diferentes periféricos o canales de solicitud (web, kioskos, bots redes sociales, etc) así también se trabaja con sistemas de CallCenter - radio taxi.

Fig. A7. 2. Componente Cliente y Conductor

Nuestro sistema consta de 5 módulos principales los cuales son:

- **Tecnología Cliente:** Multiplataforma tecnológica para solicitar el servicio de taxi
    - Aplicativo móvil para Android.
    - Aplicativo móvil para iOS.
    - Plataforma web.
    - Sistema corporativo.
    - Bot de Messenger.
- **Tecnología Conductor:** Toda la tecnología en app móviles para que el conductor pueda brindar su servicio.
    - Aplicativo móvil para Android.
    - Aplicativo móvil para HarmonyOS
    - Aplicativo móvil para iOS.
- **Tecnología CallCenter:** Toda la tecnología para brindar el servicio desde la central.
    - Solución multiplataforma (Windows)

- Software de despacho de carreras.
- Robot Charlie ENV,, atención automatizada de llamadas
- Central telefónica virtual Call Center.
- Sistema web de administración, gestión y monitoreo.

- **Sistema de monitoreo**: Permite monitorear toda la operación de los diferentes componentes del sistema, dicho monitoreo se realiza mediante la creación de usuarios con sus respectivos password de seguridad.
- **Sistema de Gestión de Administración y Gestión de Taxis**: todo el sistema para que el administrador tenga el control total del mismo.

Fig. A7. 3. Gestión de administración de taxis

COMPONENTES DE CLIPP TAXI

Clipp Taxi se consolida en una plataforma integral para el trabajo de operadoras de taxi.

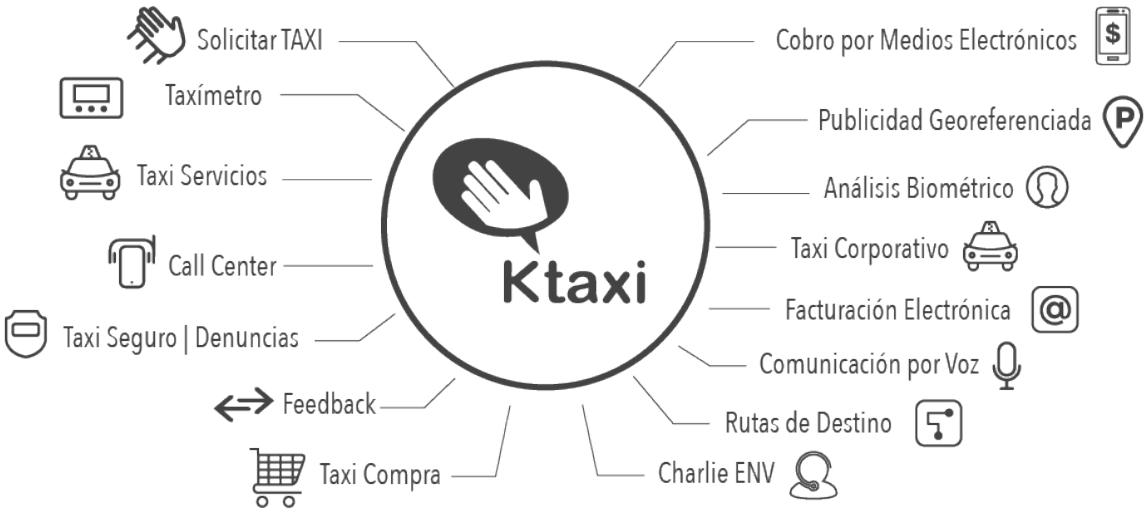

Fig. A7. 4. Componentes de Clipp Taxi

Solicitar Taxi: es la diferente tecnología relacionada para que los clientes puedan solicitar una unidad de taxi, entre esto está:

- Aplicativo nativo para Android e iOS
- Portal web para solicitud
- Aplicativo para Kiosco
- Dashbuttom
- Bot de Messenger.

Taxímetro: Se cuenta con el desarrollo electrónico e informático para el taxímetro. El mismo que respeta y acopla a los requerimientos de telemetría que requiere las diferentes entidades de control en donde funciona el aplicativo.

Uno de los parámetros de un modelo de Taxímetro que manejamos es de que el dispositivo móvil se conecta al vehículo, es por eso que se conecta de forma directa al odómetro de la unidad.

- **Exactitud en tarifación. –**

Debido a la conexión directa con el odómetro que brinda los datos desde la computadora del vehículo este taxímetro brinda una mejor exactitud en la tarifación de los servicios realizados, ya que la lectura de datos se la realiza mediante la comunicación vía bluetooth o wifi desde el módulo electrónico a la Tablet.

- **Reprogramación rápida y libre de mantenimiento. -**

Ya que se trabaja bajo una modalidad de aplicativo móvil integrando tecnologías y gestión de flotas mediante el uso de telecomunicaciones e internet se puede realizar actualizaciones de tarifa de forma remota y en tiempo inmediato con garantías de no tener alteraciones y corrupción de tarifas.

- **Interacción de fácil y rápida**

Debido a que se encuentra diseñado de acuerdo a las normativas de regulación de nuestro país la interacción y gestión del taxímetro Clipp Taxi es sencilla y rápida

- **Equipado con impresora tarifador**

El presente taxímetro se encuentra equipado con impresora térmica para la emisión de los respectivos recibos por cada uno de los servicios que se realizan.

CallCenter: Es un sistema que nos permite modernizar las centrales o CallCenter de las operadoras de radio taxi, el sistema de CallCenter se integra a todo el sistema de aplicativos móviles lo que permite ser una solución incluyente y no excluyente. Los componentes principales son:

- Software de Despacho de carreras.
- Robot Charlie, atención automatizada de llamadas. (En desarrollo).
- Aplicativo móvil CallCenter (callerID).
- Sincronizador de datos a la nube.

Los casos principales de uso son:

- Despacho convencional bajo la modalidad de un trabajo de sistema de radio.

- Recibir solicitudes por medio del aplicativo y despacharlos por medio de radio.
- El despacho de carreras a las unidades que tienen aplicativo.
- Comunicación con el conductor (audio y texto)
- Registro de clientes nuevos
- Registro de compras
- Control de multas y sanciones
- Control y registro de solicitudes de carreras
- Interacción con diferentes centrales telefónicas o identificadores de CallerID o líneas digitales o IP

Robot Charlie: Es un software que permite contestar llamadas de forma automática, aumentando la producción del CallCenter debido a que en horas pico el software interactúa con los como una centralista aportando a la eficiencia del servicio, lo que en un futuro se espera tener una atención automatizada.

Se cuenta con un módulo que nos permite interactuar con los usuarios de forma automatizada sin interacción humana, los bots (programas informáticos) son capaces de interactuar con los usuarios que necesiten un taxi y lo pueden realizar desde Messenger de la aplicación Facebook.

Taxi Seguro | Denuncias | Contáctenos | Sugerencia: Enfocado en un buen servicio, seguridad y atención al cliente se tiene estos 4 módulos los cuales brindan lo siguiente:

Taxi Seguro: permite consultar los datos como números de operadora de taxi, fotos de vehículo y propietario de los vehículos registrados en el sistema, ofreciendo seguridad y confianza a los usuarios.

Denuncias: Un usuario puede realizar una denuncia en caso de que algún conductor incurra en alguna falta.

Contáctenos: en caso de que los usuarios tengan dudas o preguntas sobre el servicio o aplicativo pueden realizarlo por este medio.

Sugerencias: con la filosofía de escuchar a nuestros usuarios se tiene la sección para que nos puedan enviar sus sugerencias tanto del aplicativo como del servicio.

Centro Virtual Android: es un aplicativo móvil que trabaja bajo Android el mismo se lo emplea para conectar las líneas telefónicas al sistema despacho, así en la línea celular permite gestionar carreras por medio de WhatsApp ya que no se pierde la conectividad.

FORMAS DE PAGO CLIPP TAXI

Dentro del aplicativo se brinda diferentes modalidades de cobros, lo cual facilita a los usuarios el acceso al servicio y gestión del mismo, tales como:

Formas de pago: Se cuenta con diferentes formas de pago disponibles para que el usuario pueda elegir la que mayor le convenga.

- **Código Promocional:** Para poder hacer sorteos de carreras o premiar a usuarios vamos a tener en el sistema una opción de código promocional.
- **Voucher:** Es un saldo tipo crédito que lo gestionan las empresas con sus trabajadores.
- **Tarjetas de Crédito o débito:** Se tiene implementando pasarela de pagos para facilidad del usuario.
- **Saldo Clipp:** Permite comprar crédito a través de entidades financieras, tarjeta de crédito o débito,
- **PayPhone:** se tiene un convenio para realizar el cobro por medio de PayPhone.
- **Efectivo:** Para satisfacción del cliente también se aceptan pagos en efectivo.

PROPINA o RECARGO:

El usuario tiene la opción de ofrecer una propina previo al realizar una solicitud. Esto se puede activar por ciudad.

PUBLICIDAD:

Se ofrece diferentes módulos publicitarios tanto en la app cliente como conductor, con analística de datos para que los clientes que deseen pautar publicidad puedan medir resultados. Los módulos publicitarios son configurables es decir se pueden realizar cambios a satisfacción del cliente. Entre los módulos publicitarios se tiene:

- Pin GPS

- Banner
- Mensaje directo
- Saludos automáticos en app de conductor
- Chat automático
- Agenda de eventos
- Personalización de vehículos
- Personalización de splash principal
- Blogs
- Redes Sociales
- Mailing

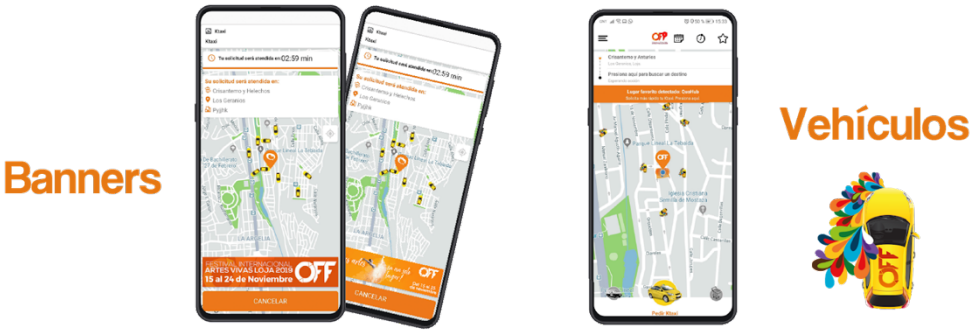

Fig. A7. 5. Tipo de servicio y clientes Clipp Taxi

Fig. A7. 6. Servicios que se atienden en nuestro sistema

La imagen anterior engloba los clientes que podemos atender desde nuestro sistema.

- **Convencional:** Quienes realizan llamadas telefónicas de forma tradicional
- **Móviles:** clientes que realicen desde aplicativo móvil.
- **Computador:** quien desde un portal web lo pueda realizar.
- **Bot de Messenger:** una forma más rápida de solicitar taxi

MODALIDAD DE SERVICIO

Se puede personalizar el sistema de acuerdo al tipo de servicio.

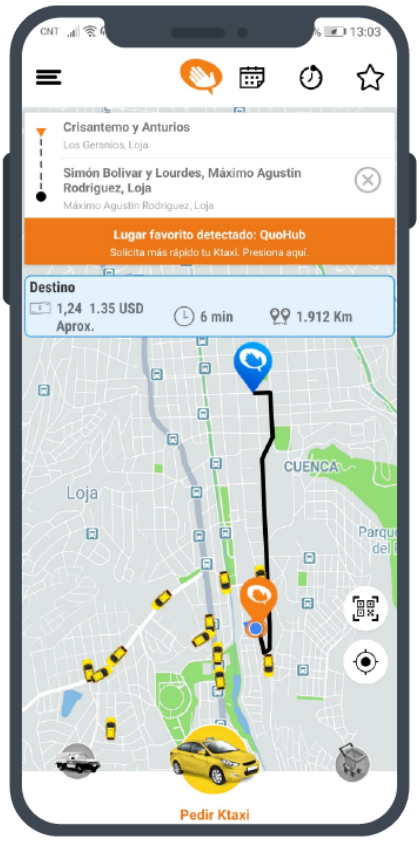

Fig. A7. 7. Modalidad de servicios

Otras funcionalidades

Clipp Taxi cuenta con otras funcionalidades que complementan al sistema y a su vez le dan la robustez necesaria a la solución y se convierte en una herramienta para las operadoras y el socio conductor.

Sensor Asiento: permite detectar de forma automática por medio de un sensor de presión si el taxi está siendo ocupado o está libre, lo que permite calcular la rentabilidad del taxi y a su vez obtener datos para cálculos de Origen / Destino.

Taxi pirata: Es una funcionalidad que permite informar cuando el socio conductor visualice una unidad no autorizada realizando la actividad del Taxismo puede notificar reportando el sitio y la placa de la unidad teniendo una base de datos.

Pánico: Permite informar de forma colaborativa a todos los conductores cercanos al punto de una alerta de pánico pudiendo integrarse a cualquier otro sistema de alertas.

Encomiendas: Similar al sistema de compras, pero es un módulo puntual que permite gestionar él envió de un lugar a otro, en donde se ubica los datos del destinatario y datos del destino de tal manera que el conductor ya conoce el origen y el destino.

Otros ingresos: Dentro del sistema se permitirá poder generar diferentes ingresos adicionales dentro los cuales estarán:

- **Transacciones con dinero electrónico:**

Publicidad: El conductor principalmente en el taxi podrá generar publicidad de vídeo, provocando esto como un ingreso extra en la unidad.

Fig. A7. 8. Videos publicitarios

Favoritos e Historial: Para facilidad del usuario y que tenga un acceso rápido podrá realizar de una manera ágil y rápida utilizando un historial que son las últimas direcciones y favoritos que serían registradas como ubicaciones de uso frecuente. Así mismo en el mapa se puede encontrar las direcciones favoritas para mayor facilidad de uso.

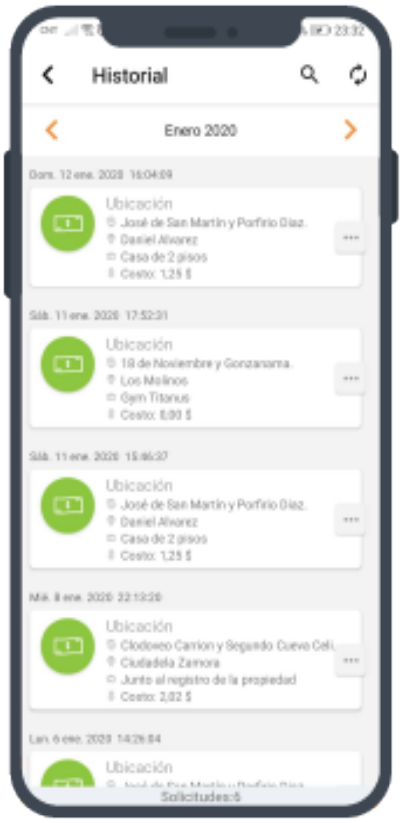

Fig. A7. 9. Imagen de historial y favorito de Clipp Taxi

Calificación: una vez finalizado el servicio se permite calificar y comentar sobre el servicio al usuario de parte del conductor y al conductor de parte del usuario con el objetivo de poder tener retroalimentación de la calidad de servicio brindado y así tomar acciones correctivas.

Compartir Carrera: Permite al usuario compartir todos los datos de la carrera como el recorrido del mismo a sus familiares o en sus redes sociales.

Reserva: Permite generar una alarma o realizar una petición de carrera que se planifique con antelación.

Login / Registro Facebook: Facebook al momento del registro permite conectarse a Facebook para extraer los datos o Login de forma directa si está previamente registrado

Tarifario/Cotizador: permite tener valores preestablecidos como en carreras entre ciudades, y cotizar consiste en un módulo en que el conductor previamente manifiesta el valor que va a cobrar por dicho servicio. o ubicar un destino y establecer el valor aproximado a cobrará basado en las tarifas de la ciudad y considerando la distancia el el tiempo.

Interacción con otras plataformas: interactuar con otras plataformas nos permite ser una mejor herramienta para el socio conductor, Waze o Google Transit para el guiado de la ruta o tráfico.

Registro de IMEI: A los conductores se les registra el IMEI del dispositivo móvil con la finalidad de que solamente puedan ingresar desde un móvil y así evitar que se presten los móviles o se den credenciales a otros conductores no autorizados.

Login del conductor: Se cuenta en el sistema con una sección de preguntas frecuentes, una verificación por clave y con el registro del Imei del dispositivo permitiendo podrá acceder solo en uno registrado y activado por un administrador del sistema.

Fig. A7. 10. Ingreso al app conductor

Con un código de seguridad que es utilizado en voucher y como medio de seguridad entre usuario y conductor.

Lectura de dispositivos beacon: Nuestras apps cuentan con la detección de dispositivos beacon o iBeacon los mismos permiten al usuario en caso de no tener el app instalado hacerlo en ese momento, caso contrario mostrar información del taxi y conductores de la unidad.

Notificación por voz de la cercanía de la unidad: En nuestro aplicativo se muestra alerta por voz indicando se está próximo a llegar al destino especificado por el usuario cliente.

Fig. A7. 11. Notificaciones y alertas

Plataforma multilingüe: Nuestra plataforma cuenta con selección de idiomas permitiendo establecer el idioma predeterminado favorito que se desee.

Estadística: El sistema web de administración cuenta con un módulo donde se lleva gráficas estadísticas como lo es de carreras, clientes, compras, entre otros.

Servicio adicional: Se indica al usuario qué tipo de servicios adicionales tiene la unidad que le recogerá como lo puede ser:

- Wifi
- Aire acondicionado
- Porta bicicletas
- Llevar mascotas, entre otros.

Tipo de servicio y Categoría: se podrá parametrizar los servicios como es Smart taxi, Radio taxi, además de la empresa que provee el servicio, así como el usuario poder seleccionar si quiere el servicio de la unidad que está más cercana o elegir de las unidades que postulen a esa carrera.

Fig. A7. 12. Servicios que cuenta Clip Taxi

Adicional a esto se podrá categorizar o priorizar en la asignación de carreras a ejemplo conductores que cuenten con promedio superior a 4,5 y el usuario también podría seleccionar este tipo de servicio, en las unidades se podrá contar con tarjetas RFID o códigos QR que puedan identificar la unidad y sirve para que el usuario verifique la unidad que aborda.

Flota del plan piloto

Para el plan piloto se arrancará con 30 Vehículos Taxis inscritos en el MTT se adjuntará archivos de conductores los cuales ya están inscritos en la plataforma.

Duración de piloto

La duración del plan piloto en Clipp taxi debería ser de 60 días a contar de la aprobación MTT.

Propuesta metodológica de evaluación del plan piloto

La metodología a emplear para la evaluación se realizará 3 meses después de la implementación del plan piloto “Tecnologia App para Taxis en Chile”. La evaluación incluirá:

- Análisis de la información contenida en el web 1.0 de Clipp Taxi como el reporte de carreras.
- Entrevista a usuarios (conductores y clientes).

Modalidad de taxi

- **Taxi Basico:** Es un servicio que atiende viajes cuyo origen y destino es determinado por los pasajeros que lo utilizan, pudiendo contar con paraderos, y se identifica claramente por sus colores Amarillo y Negro.

En la Región Metropolitana las tarifas son fijadas en enero y junio de cada año por la Seremi de Transportes y desde el 08 de agosto de 2022 rigen estas:

- ***Bajada de Bandera:*** $ 400
- ***Cobro variable (c/60 seg o c/200 mts):*** Mínima: $140 y Máxima: $180
- **Taxi Ejecutivos:** Es una submodalidad del taxi básico, que sólo atiende viajes solicitados a distancia por cualquier persona, a través de los distintos medios de comunicación, no pudiendo atender viajes solicitados en la vía pública, y no se le aplica exigencia de color. Se identifica principalmente por su placa patente naranja con letras negras. El cobro de este servicio es exclusivo por medio del uso de taxímetro.

En el caso de la Región Metropolitana las tarifas son fijadas en enero y junio de cada año por la Seremi de Transportes y desde el 08 de agosto de 2022 rigen estas:

- ***Bajada de Bandera:*** $1800
- ***Cobro variable (c/60 seg o c/200 mts):*** Mínima: $140 y Máxima: $180

Anexo 8. Análisis de los principales componentes de una arquitectura basada en microservicios

Los microservicios maximizan la confiabilidad de las aplicaciones y la velocidad de implementación. Esto es particularmente importante en un mercado que se mueve y evoluciona más rápido que nunca. La arquitectura en contenedores de los microservicios mueve las aplicaciones a cualquier lugar sin alterar ni interrumpir el entorno, lo que facilita la velocidad y reduce el tiempo de inactividad [24].

Según lo expuesto por Gitlab [24], los componentes centrales de una arquitectura de microservicios son:

- **Clientes:** las aplicaciones de los clientes generalmente deben consumir la funcionalidad de múltiples microservicios y requieren actualizaciones frecuentes.
- **Bases de datos:** una API de servicio actualiza las bases de datos de microservicios al transportar todos los servicios remotos que admiten comunicaciones entre procesos para diferentes pilas.
- **Puerta de enlace de API:** una puerta de enlace de API es un patrón de diseño de microservicios que es un punto de entrada de aplicación único fundamental para enrutar solicitudes y traducir protocolos.
- **Proveedores de identidad:** un microservicio de identidad debe permitir el acceso de servidor a servidor y controlado por el usuario a los datos de identidad.
- **Formatos de mensajería:** los microservicios se comunican con patrones de arquitectura de microservicios síncronos o asíncronos.
- **Contenido estático:** los clientes reciben contenido estático a través de servicios de almacenamiento basados en la nube y redes de entrega de contenido.
- **Gestión:** este componente permite a los usuarios empresariales configurar servicios durante el tiempo de ejecución.
- **Detección de servicios:** se requiere un mecanismo de detección de servicios para solicitar el servicio.

La empresa de consultoría tecnológica de París Adservio [25], expone que los componentes claves de un arquitectura basada en microservicios está confirmada por 5 componentes que son:

- **Microservicios:** El componente principal de la arquitectura de microservicios son los servicios autónomos. Pueden ser cualquier lenguaje y función separados entre sí, lo que los hace ideales para la implementación a través de varios equipos de software. También está la malla de servicio. Es responsable de la comunicación entre microservicios a través de una capa de mensajería.

- **Contenedores:** Los contenedores son un paquete de software que funciona de forma independiente. Funcionan para aislar cada servicio en el mismo entorno. Para crear microservicios, una máquina virtual puede servir como contenedor y simular las funciones de una computadora física estándar. Los contenedores brindan eficiencia porque solo se basan en dependencias específicas y el código subyacente dentro del servicio.

- **Malla de servicios:** La red de servicios ofrece una capa de infraestructura que brinda seguridad adicional y genera comportamientos más predecibles en términos de comunicación.

- **Descubrimiento de servicios:** El descubrimiento de servicios ayuda a administrar la implementación y distribuir uniformemente la carga. Este componente presenta un consumo de servicios, un proveedor de servicios y un registro de servicios.

- **Puerta de enlace API:** La puerta de enlace API es una parte esencial de la comunicación en la compleja arquitectura de microservicios distribuidos. Puede manejar tareas administrativas, garantizar que la carga de microservicios sea liviana y realizar acciones de equilibrio de carga cuando sea necesario.

Así mismo Albertos Gómez en su revisión sistemática de literatura en [26], indica en los resultados obtenidos que los componentes que intervienen en una arquitectura de microservicios son:

- **Service Discovery:** Para el descubrimiento de servicios en la red

- **API Gateway:** Define como los clientes acceden a los servicios, siendo capaz de ofrecer diferentes APIs por cada cliente. En realidad, agrupa interfaces y realiza el enrutamiento de la petición entre servicios, en los dos sentidos de la comunicación. Es un elemento arquitectónico relevante en la mayoría de los estudios primarios.
- **Circuit Breaker y Health Status:** Evitan la sobrecarga de un nodo en fallo
- **Contendores (Docker):** Para el despliegue
- **Balanceador de carga**
- **Middleware:** Comunicación centralizada como medio de comunicación entre microservicios.

Microsoft en su publicación 'Estilo de arquitectura de microservicios' en [27], indica que una arquitectura de microservicios consta de una colección de servicios autónomos y pequeños como se puede observar en la Fig. A8. 1. Cada uno de estos servicios es independiente y debe implementar una funcionalidad de negocio individual dentro de un contexto delimitado. Un contexto delimitado es una división natural de una empresa y proporciona un límite explícito dentro del cual existe un modelo de dominio.

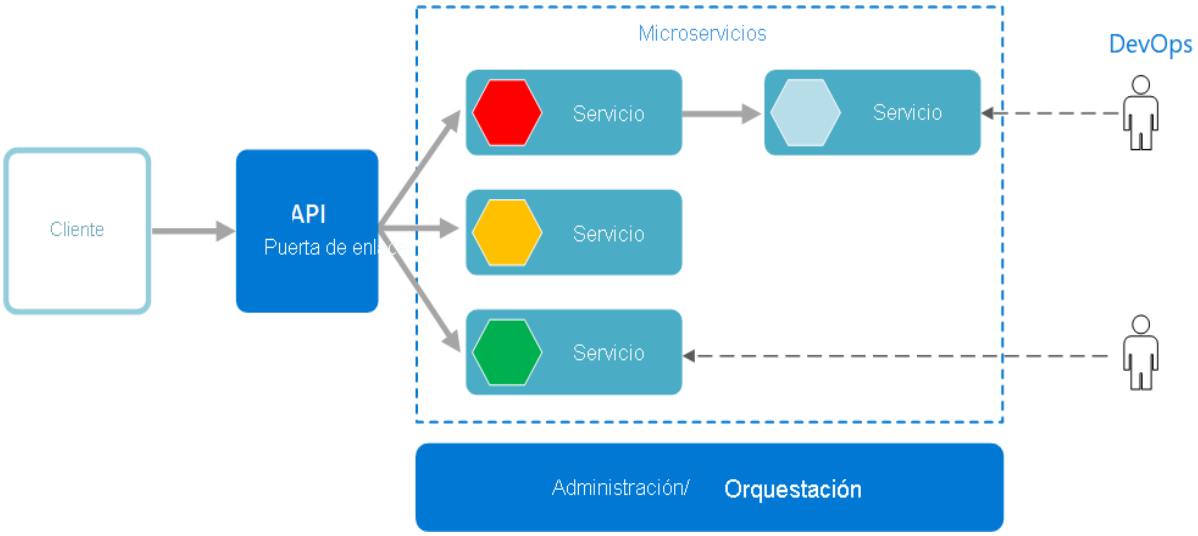

Fig. A8. 1. Estilo de arquitectura de microservicios propuesto por Microsoft

Y que entre los componentes que aparecen en una arquitectura típica de microservicios se tienen los siguientes:

- **Administración e implementación:** Este componente es el responsable de la colocación de servicios en los nodos, la identificación de errores, el reequilibrio de servicios entre nodos, etc. Normalmente, este componente es una tecnología estándar, como Kubernetes, en lugar de algo creado de forma personalizada.

- **Puerta de enlace de API:** La puerta de enlace de API es el punto de entrada para los clientes. En lugar de llamar a los servicios directamente, los clientes llaman a la puerta de enlace de API, que reenvía la llamada a los servicios apropiados en el back-end. Entre las ventajas de usar una puerta de enlace de API se encuentran las siguientes:

- Desacoplan los clientes de los servicios. Los servicios pueden cambiar de versión o refactorizarse sin necesidad de actualizar todos los clientes.

- Los servicios pueden utilizar los protocolos de mensajería que no son fáciles de usar para un servicio web, como AMQP.

- La puerta de enlace de API puede realizar otras funciones transversales como la autenticación, el registro, la terminación SSL y el equilibrio de carga.

- Directivas estándar, como, por ejemplo, para la limitación, el almacenamiento en caché, la transformación o la validación.

En un reporte presentado por Petter Johansson sobre la comunicación entre una aplicación de cliente y los microservicios y también una revisión de cómo el rendimiento y la escalabilidad de los microservicios se ven afectados por diferentes opciones de hospedaje indica que los componentes básicos para una arquitectura de microservicios son los siguientes:

- **Registro de servicios:** Dado que los microservicios están diseñados para implementarse en un entorno de nube donde las direcciones de los servidores son dinámicas y pueden cambiar en cualquier momento, cada servicio en ejecución debe registrarse en un almacenamiento con una dirección conocida. Los clientes que se van a comunicar con los servicios obtienen sus direcciones del registro de servicios con una dirección estática conocida.

- **Equilibrador de carga:** Los microservicios se amplían aumentando el número de instancias de un determinado servicio. El equilibrador de carga distribuye el tráfico entrante a los servicios que tienen poca carga para evitar un cuello de botella en el

sistema que resulte en un mayor tiempo de respuesta. Se le asigna la responsabilidad de realizar constantemente controles de salud en los servicios para que sea posible escalar el sistema después de las solicitudes entrantes actualmente.

- **Comunicación interna:** En la arquitectura de microservicios, los servicios se comunican entre sí mediante el paso de mensajes a través de una red. Dado que los servicios están diseñados para construirse en paralelo entre sí, se requieren métodos de comunicación predefinidos. Por lo tanto, los métodos de comunicación elegidos deben ser compatibles con todas las plataformas e idiomas que se utilizan en el sistema o que se pueden utilizar en el futuro. Alternativas comunes como el paso de mensajes asincrónicos con REST o patrones de publicación/suscripción donde los servicios pueden suscribirse a canales y recibir un aviso cuando se agrega un nuevo mensaje a ese canal.

- **Comunicación externa:** La comunicación externa debe usar un protocolo o arquitectura que pueda admitir clientes creados para computadoras de escritorio, teléfonos móviles o aplicaciones web. La arquitectura REST se ha convertido en una de las alternativas más comunes, ya que es completamente independiente de la plataforma cuando se usa con JSON como serialización de mensajes.

Resultados:

La TABLA A8. 1, detalla los resultados obtenidos del análisis de los componentes para una arquitectura basada en microservicios, de los cuales se seleccionaron ***clientes, API Gateway, comunicación, microservicio y contenedores*** debido a su mayor cantidad de presencia en la información analizada, además se determinó el componente de ***base de datos*** debido a que el modelo de negocios de la empresa Kradac Cia. Ltda., se basa en la transacción de la información en bases de datos relacionales.

TABLA A8. 1

RESULTADO OBTENIDOS DEL ANÁLISIS DE INFORMACIÓN DE COMPONENTES PARA MICROSERVICIOS

		Bibliografías evaluadas					
		Gitlab	**Microsoft**	**Adservio**	**Albertos Gómez**	**Petter Johansson**	**Resultados**
	Clientes	1	1			1	**3**
	Base de datos	1					**1**

Componente	**Api Gateway**	1	1	1	1		**4**
	Proveedor de identidad	1					**1**
	Comunicación (Mensajería)	1	1		1	1	**4**
	Contenido estático	1					**1**
	Detección de servicios	1		1	1		**3**
	Microservicios	1	1	1			**3**
	Contenedores		1	1	1		**3**
	Malla de servicios			1			**1**
	Circuit Breaker y Health Status				1		**1**
	Balanceador de carga		1		1		**2**

En la Fig. A8. 2, se puede visualizar el resultado obtenido del análisis de la información considerando los componentes con mayor presencia en las fuentes citas anteriormente.

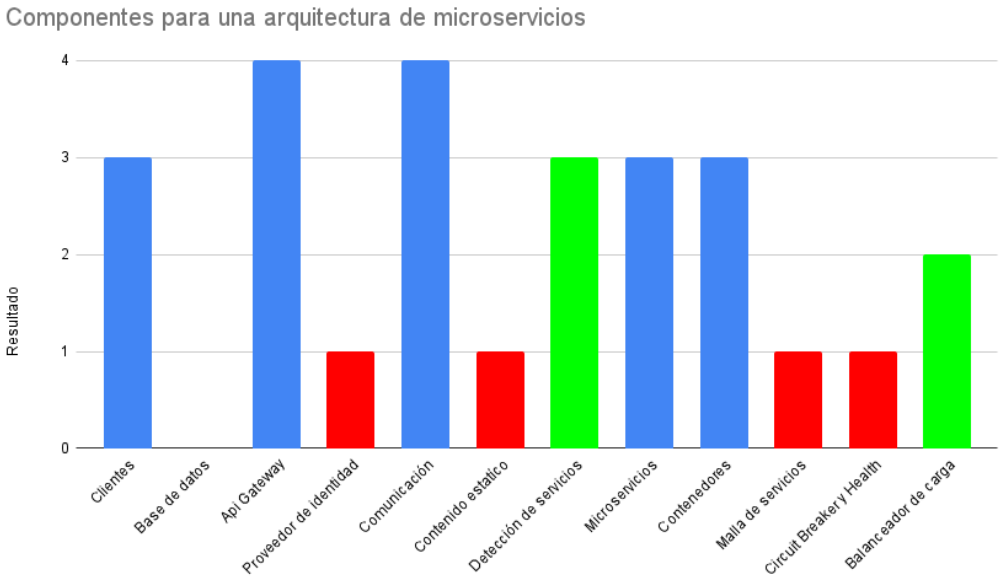

Fig. A8. 2. Resultados del análisis de información referente a los componentes de microservicios

Anexo 9. Opciones para la definición del Dockerfile

Dentro de las opciones más relevantes para un Dockerfile se encuentran herramientas encargadas de labores, como el establecimiento de la imagen base, el cambio de usuario o los elementos preestablecidos para el arranque de un contenedor en Docker, entre otros.

Algunas de estas opciones son:

- **FROM**: es una opción de Dockerfile que debe presentarse como la primera instrucción. Cumple con la función de establecer la imagen sobre la que los pasos e imágenes siguientes se desarrollan en el sistema. La imagen mínima que da origen al resto de imágenes en Docker es llamada scratch.

- **ENV**: hace referencia a la opción que indica las variables de entorno que se necesitan en el proceso de construcción de una imagen en Docker y permite la ejecución de los contenedores y sus labores en el sistema.

- **USER**: esta herramienta se utiliza en los archivos de instrucciones de Dockerfile con el objetivo de cambiar el usuario y su pertenencia a un grupo determinado. Una vez se ejecute esta opción, se aplicará a la totalidad de instrucciones siguientes.

- **RUN**: es una de las opciones de mayor importancia y popularidad en Dockerfile. Cumple la labor de ejecutar una instrucción incluida en la línea de comandos de la imagen durante su proceso de construcción. Dockerfile RUN puede escribirse en formato SHELL o bajo la opción de escritura EXEC.

- **ADD**: este elemento se encarga de las tareas relacionadas con la copia de ficheros, directorios y archivos de una imagen en Dockerfile. Se debe tener en cuenta que el uso de la instrucción ADD implica la creación de una nueva capa de imagen, por lo que debes ser cuidadoso al implementar esta opción.

- **EXPOSE**: es la opción que tiene como labor la definición de las asignaciones referentes a los puertos para los contenedores de la plataforma que se encuentren en su etapa de ejecución.

Anexo 10. Encuesta de evaluación de la arquitectura basada en microservicios en la nube.

Socialización de la arquitectura basada en microservicios para el aplicativo Ktaxi de la empresa Kradac Cia. Ltda.

El objetivo es evaluar la arquitectura diseña en base a los diseños y prototipos presentados, con el fin de determinar si su futura implementación será viable en base a los objetivos de la empresa.

Consideraciones importantes:

Por favor, contestar a la siguiente encuesta con la mayor sinceridad posible ya que la misma ayudará a mejorar la arquitectura propuesta.

1. **¿Considera usted que la arquitectura socializada podrá mejorar la escalabilidad del aplicativo Ktaxi al modularizar sus servicios, lo que permite una mayor flexibilidad en la asignación de recursos y una mejor distribución de la carga de trabajo?**

Si ( )

No ( )

Otra… ----------

2. **¿Considera usted que la arquitectura socializada podrá mejorar la agilidad de desarrollo al descomponer la aplicación en servicios pequeños e independientes que pueden ser desarrollados, desplegados y escalados de forma independiente ?**

Si ( )

No ( )

Otra… ----------

3. **¿Considera usted que la arquitectura socializada mejora la flexibilidad de desarrollo permitiendo a los desarrolladores elegir diferentes lenguajes de programación y tecnologías para cada servicio y no estar limitados por una única tecnología?**

Si ( )

No ( )

Otra ----------

4. **¿Considera usted que los componentes utilizados en el diseño de la arquitectura satisfacen las necesidades actuales del aplicativo Ktaxi ?**

Si ( )

No ( )

Otra ----------

5. **¿Considera usted que el diseño de la arquitectura está bien detallado y que contiene la información necesaria para su posterior implementación?**

Si ( )

No ( )

Otra ----------

6. **¿Considera usted que cada componente seleccionado para la arquitectura cumple su función de manera correcta y se integra de tal forma que su funcionamiento garantiza el flujo ideal ?**

Si)

No ( )

Otra ----------

7. **¿Considera usted que el prototipo presentado refleja el diseño de la arquitectura propuesta, así como su integración con los componentes seleccionados ?**

Si ( )

No ( )

Otra ----------

8. **¿Considera usted que el prototipo presentado funciona de manera correcta, y se encuentra desarrollado respetando los principios fundamentales de microservicios?**

Si ( )

No ( )

Otra ----------

9. Por favor, sugiera mejoras u observaciones para mejorar la arquitectura y prototipo propuesto

Texto de respuesta larga

Anexo 11. Análisis de los resultados obtenidos de la encuesta

Después de aplicar las encuestas al personal de desarrollo backend del departamento de TI de la empresa Kradac Cia. Ltda., de la ciudad de Loja a través de los formularios de Google (Disponible en: https://docs.google.com/forms/d/1fBkGTgJpGSnhTqrMMe47CD6RwdFAy1RdS70idxZ1hZU/edit#responses) cuyo tamaño de muestra fue de 6 personas, se obtuvo los siguientes resultados en base a las respuestas obtenidas de los encuestados que se describen en las imágenes desde la Fig. A11. 1 – Fig. A11. 9.

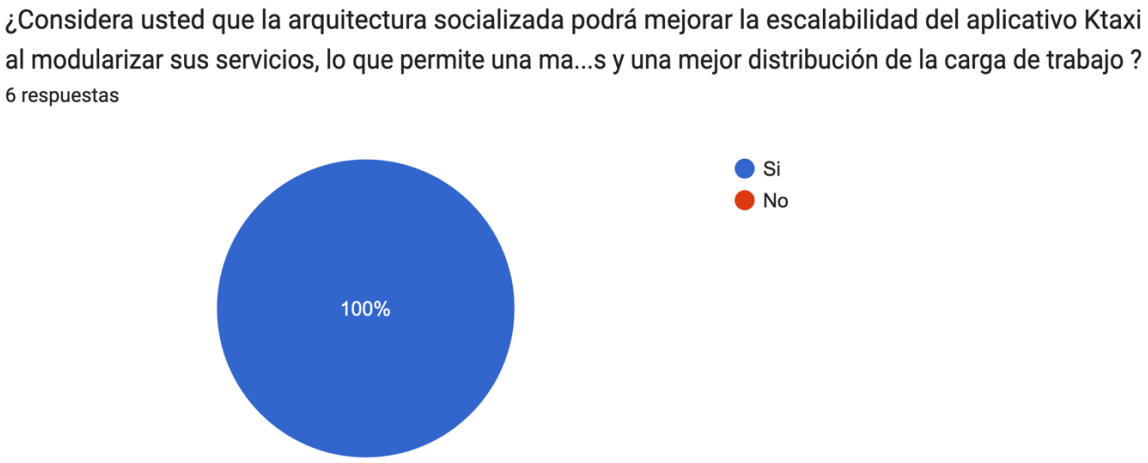

Fig. A11. 1. Resultado de la pregunta uno

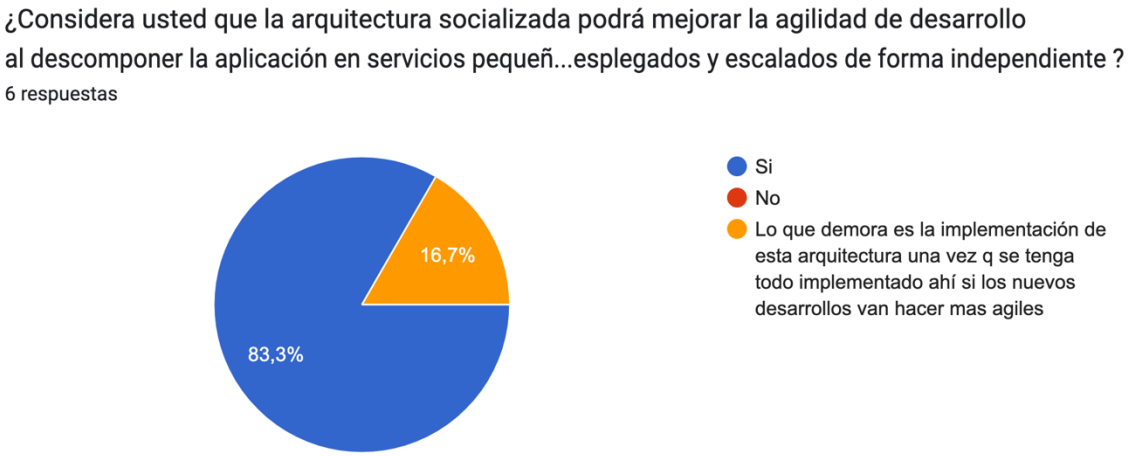

Fig. A11. 2. Resultado de la pregunta dos

¿Considera usted que la arquitectura socializada mejora la flexibilidad de desarrollo permitiendo a los desarrolladores elegir diferentes lenguajes de p...icio y no estar limitados por una única tecnología?

6 respuestas

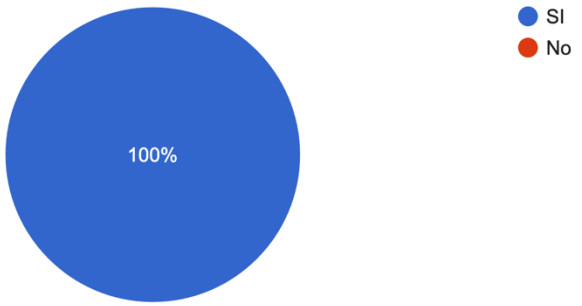

Fig. A11. 3. Resultado de la pregunta tres

¿Considera usted que los componentes utilizados en el diseño de la arquitectura satisface las necesidades actuales del aplicativo Ktaxi ?

6 respuestas

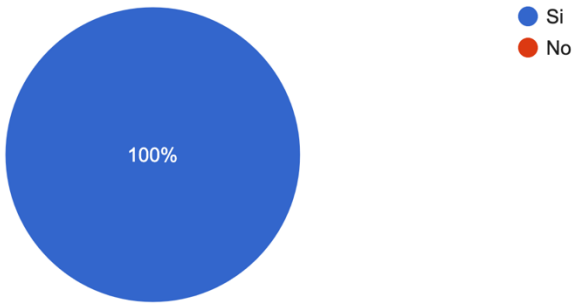

Fig. A11. 4. Resultado de la pregunta cuatro

¿Considera usted que el diseño de la arquitectura está bien detallado y que contiene la información necesaria para su posterior implementación?

6 respuestas

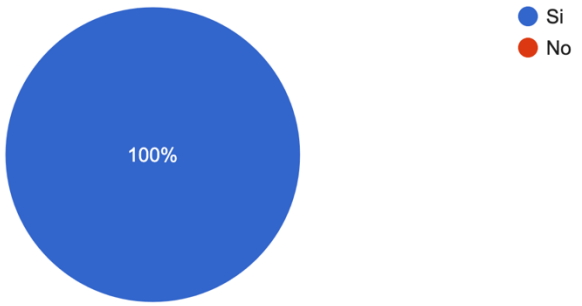

Fig. A11. 5. Resultado de la pregunta dos

¿Considera usted que cada componente seleccionado para la arquitectura cumple su función de manera correcta y se integra de tal forma que su funcionamiento garantiza el flujo ideal ?

6 respuestas

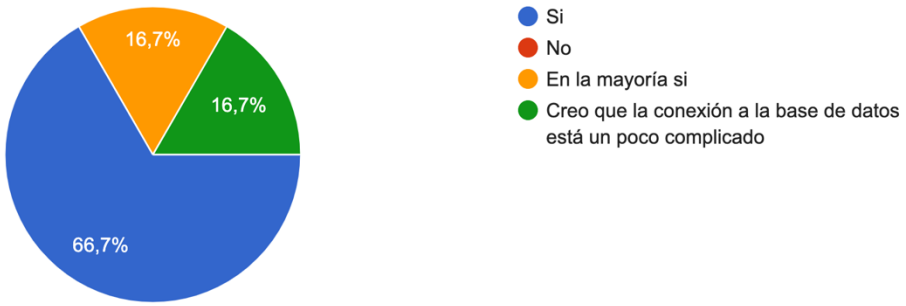

Fig. A11. 6. Resultado de la pregunta seis

¿Considera usted que el prototipo presentado refleja el diseño de la arquitectura propuesta, así como su integración con los componentes seleccionados ?

6 respuestas

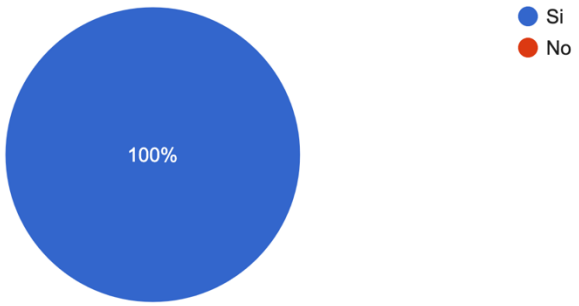

Fig. A11. 7. Resultado de la pregunta dos

¿Considera usted que el prototipo presentado funciona de manera correcta, y se encuentra desarrollado respetando los principios fundamentales de microservicios?

6 respuestas

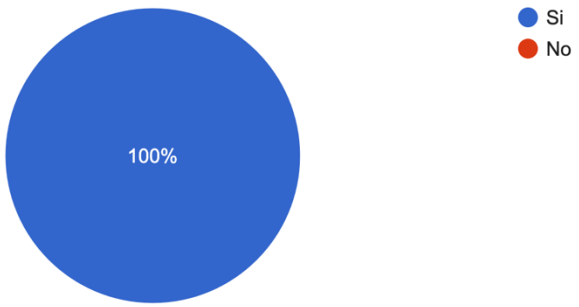

Fig. A11. 8. Resultado de la pregunta ocho

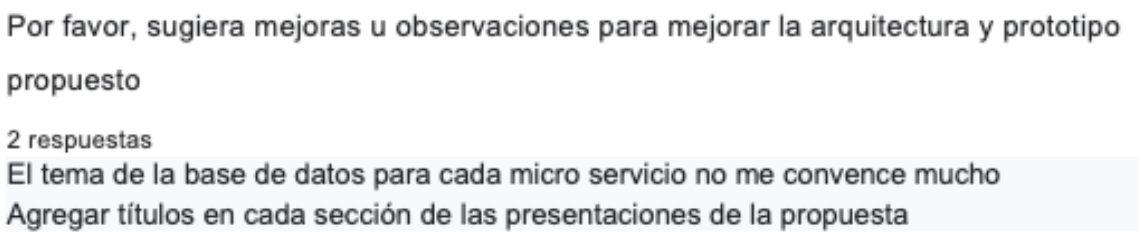

Fig. A11. 9. Resultado de la Fig. 9

En base a las respuestas obtenidas de los 6 participantes se puede establecer que la arquitectura propuesta tiene una aceptación del 93.75%, y que en base a la implementación y socialización del prototipo se obtuvieron excelentes resultados.

Anexo 12. Pruebas de carga a los microservicios desarrollados en el prototipo propuesto.

Microservicio Finalizar

Pruebas de carga al microservicio de finalizar con un rango de 500 solicitudes ejecutándose cada segundo durante un minuto, cuyos resultados se pueden observar en la Fig. A12. 1.

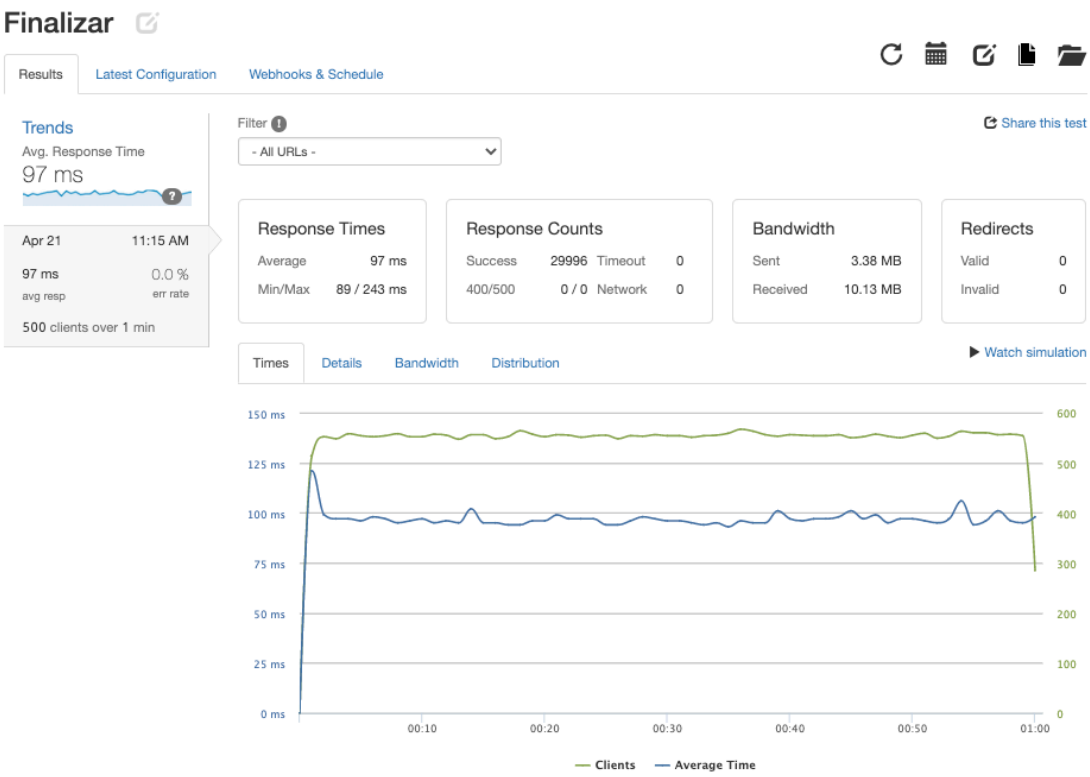

Fig. A12. 1. Pruebas de carga microservicio finalizar

Las pruebas de carga realizados el microservicio de finalizar con un total de 30.000 solicitudes se tiene un tiempo de respuesta promedio de 97 milisegundos, con un 0% de error, el servicio se encuentra ejecutándose en un servidor CentOS 7 con una memoria RAM de 500 MB y 1 CPU DO-Regular.

Microservicio autenticar

Pruebas de carga al microservicio de autenticar con un rango de 500 solicitudes ejecutándose cada segundo durante un minuto, cuyos resultados se pueden observar en la Fig. A12. 2.

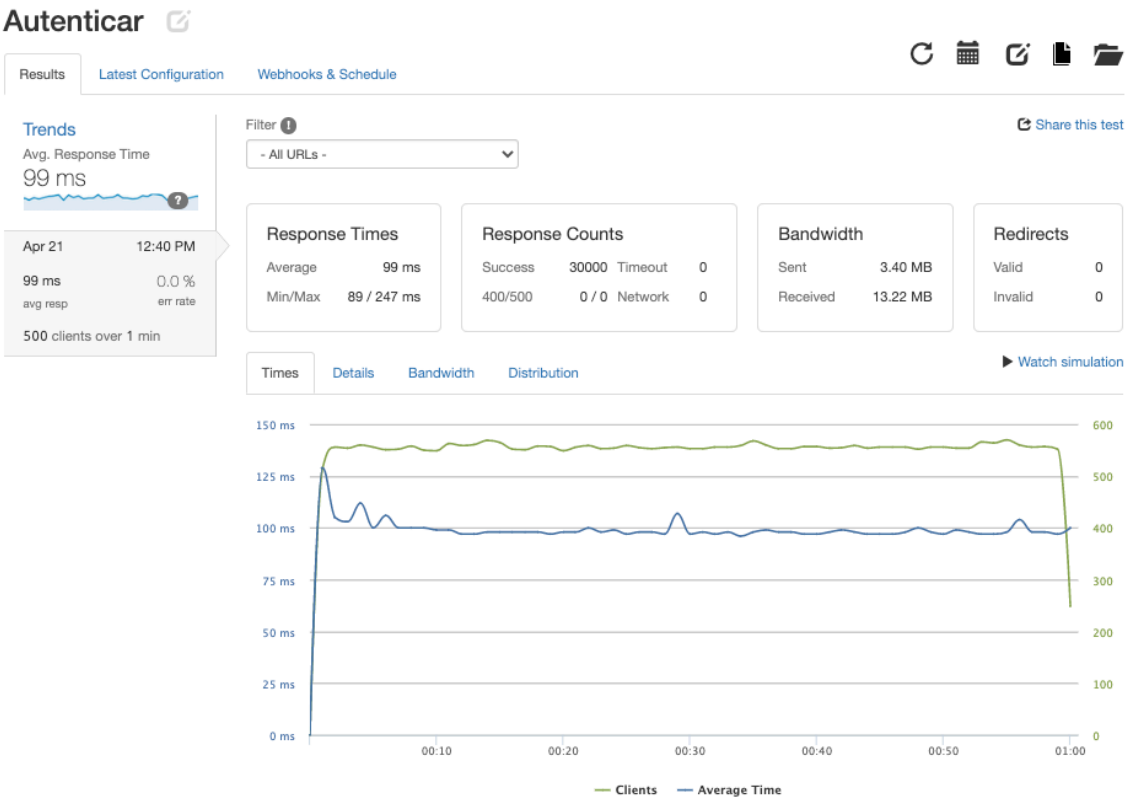

Fig. A12. 2. Pruebas de carga microservicio autenticar

Las pruebas de carga realizados el microservicio de autenticar con un total de 30.000 solicitudes se tiene un tiempo de respuesta promedio de 99 milisegundos, con un 0% de error, el servicio se encuentra ejecutándose en un servidor CentOS 7 con una memoria RAM de 500 MB y 1 CPU DO-Regular.

Microservicio solicitud

Pruebas de carga al microservicio de solicitud con un rango de 500 solicitudes ejecutándose cada segundo durante un minuto, cuyos resultados se pueden observar en la Fig. A12. 3.

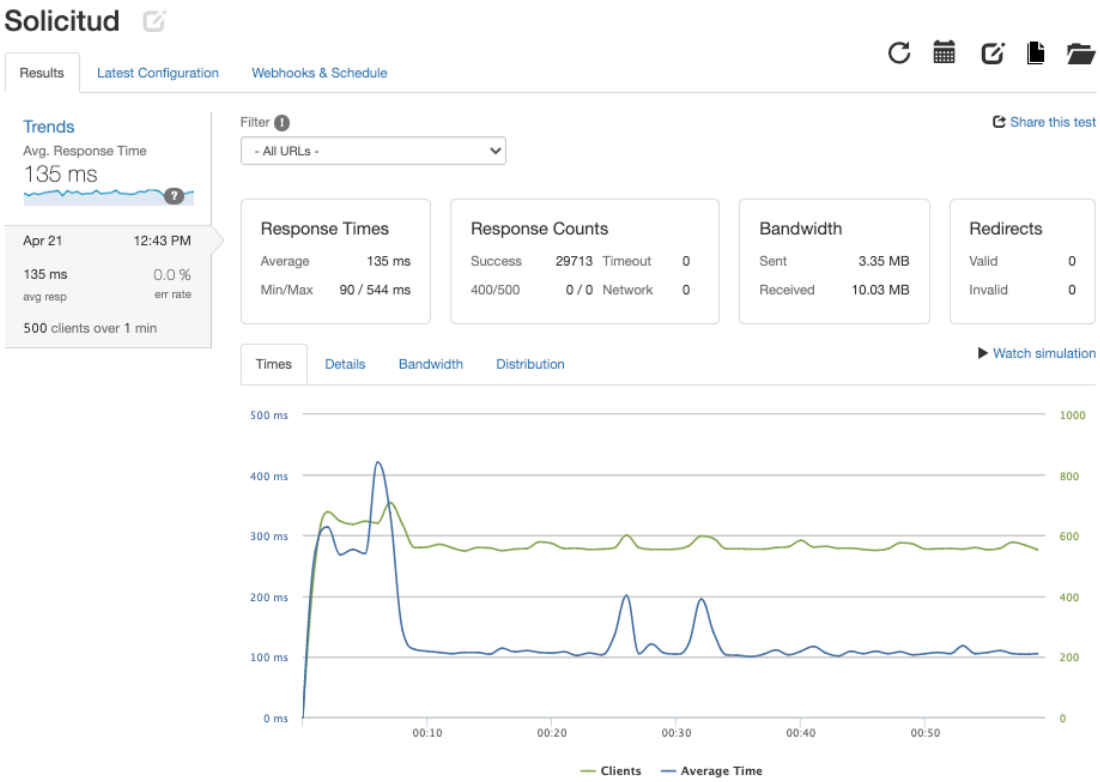

Fig. A12. 3. Pruebas de carga microservicio solicitud

Las pruebas de carga realizados el microservicio de solicitud con un total de 30.000 solicitudes se tiene un tiempo de respuesta promedio de 135 milisegundos, con un 0% de error, el servicio se encuentra ejecutándose en un servidor CentOS 7 con una memoria RAM de 500 MB y 1 CPU DO-Regular.

Anexo 13. Prototipo desarrollado para la arquitectura de microservicios

Se creó 2 sistemas web para simular aplicativos de Ktaxi, el primero para clientes y el segundo para conductores. Cada sistema web está conectado al API Gateway para la comunicación con los microservicios.

La Fig. A13. 1, Muestra la interfaz desarrollada para el login del cliente esta interfaz llama al microservicio de autenticación para poder ingresar en el sistema.

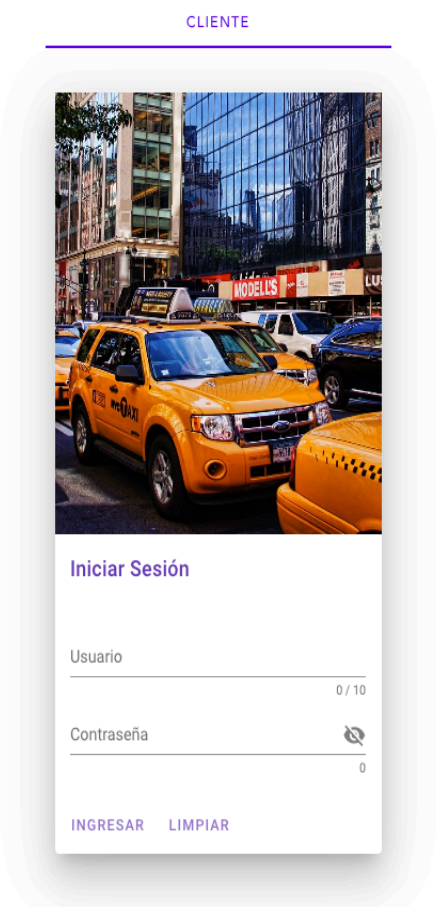

Fig. A13. 1. Pagina de inicio de sesión de los clientes

La Fig. A13. 2, Muestra la página principal para poder solicitar un taxi una vez se ha ingresado al sistema.

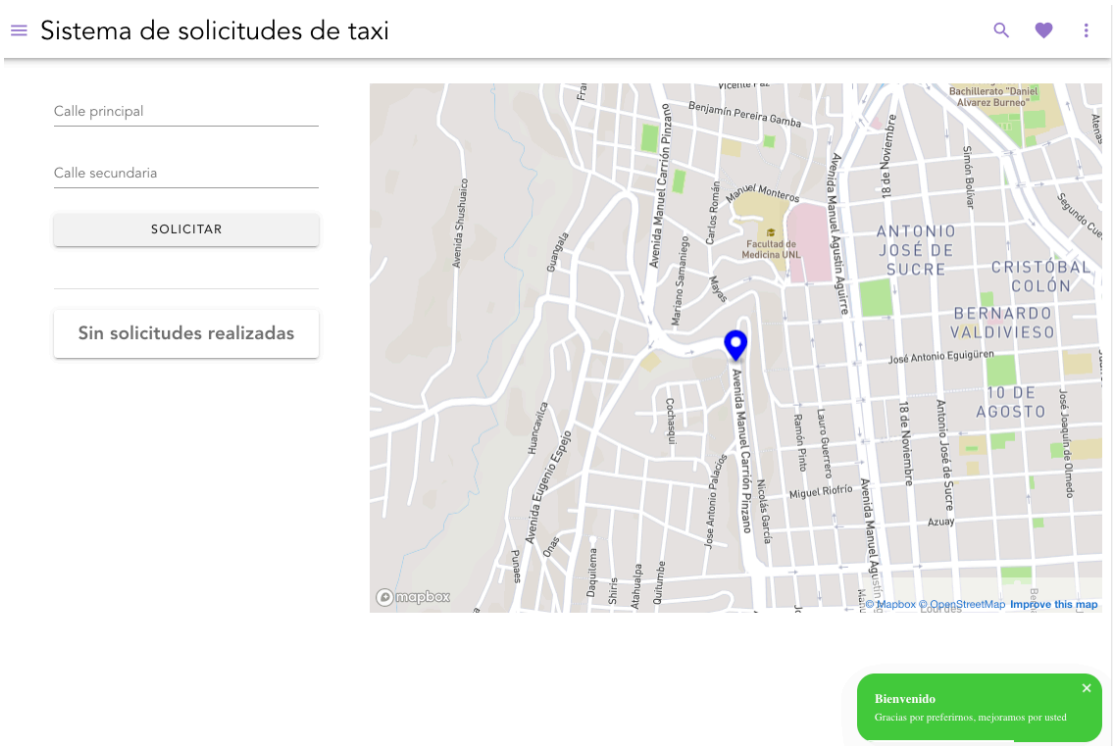

Fig. A13. 2. Acceso a la página principal del cliente

La Fig. A13. 3, Muestra la generación de una solicitud de taxi, para lo cual se utiliza el microservicio de solicitud, y este a su vez mediante rabbitMQ notifica al microservicio de autenticación que busque un conductor para la solicitud notifique al conductor la solicitud.

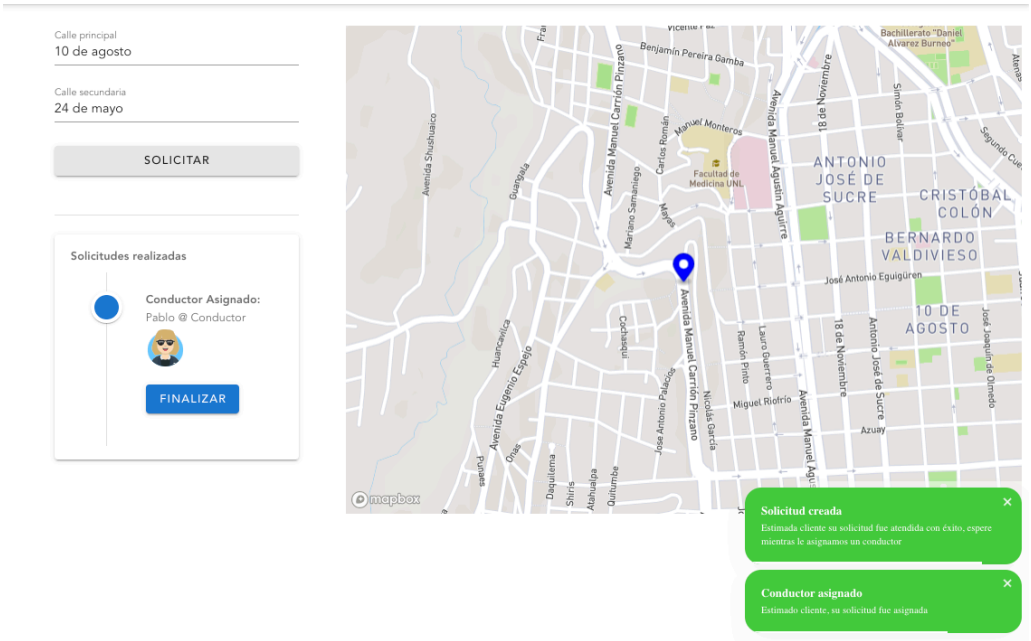

Fig. A13. 3. Solicitud de un taxi por parte del cliente

La Fig. A13. 4, Muestra la finalización de una solicitud para lo cual se utiliza el microservicio de finalizar, hasta este punto se ha utilizado todos los componentes de la arquitectura.

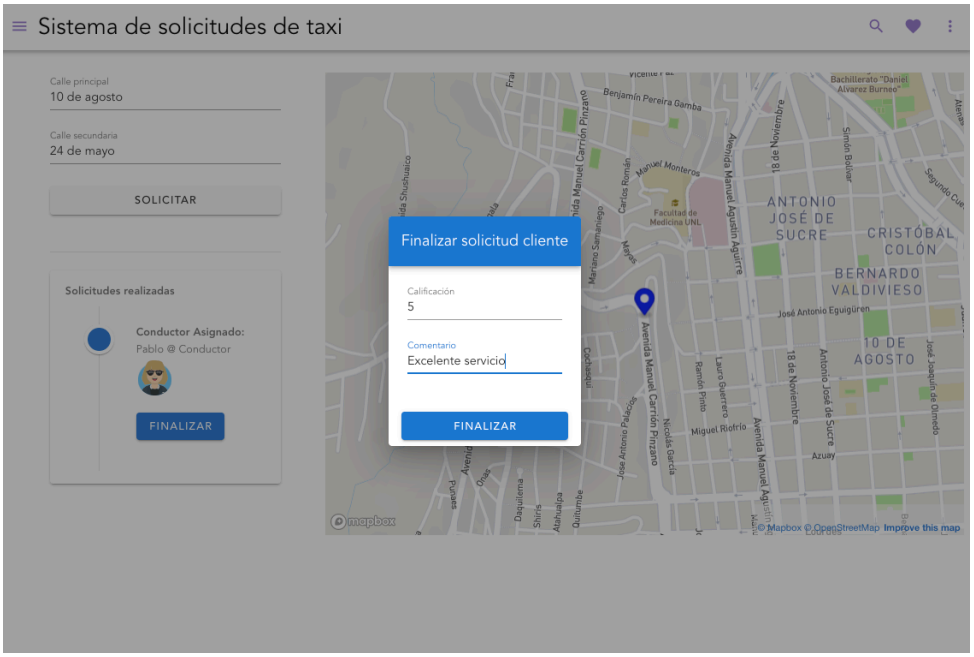

Fig. A13. 4. Finalización de una solicitud

La Fig. A13. 5, Muestra la página para el acceso del conductor al sistema para lo cual utilizará el microservicio de autenticación que es el mismo microservicio que utiliza el cliente.

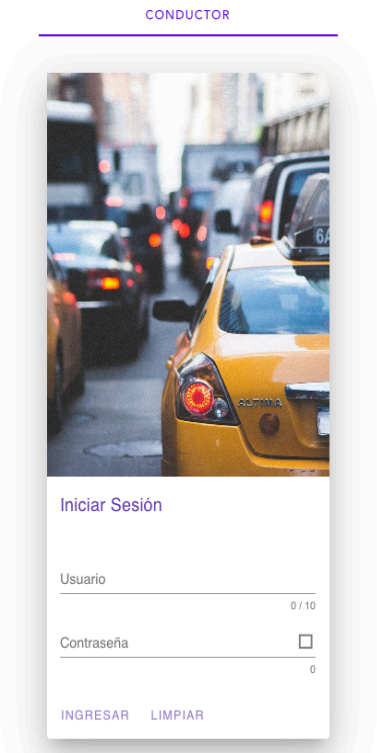

Fig. A13. 5. Inició de sesión del conductor

La Fig. A13. 6, Muestra la página principal del conductor en donde se le notificarán de solicitudes nuevas que han sido asignadas por el microservicio de autenticación que es que maneja la base de datos de clientes y usuarios.

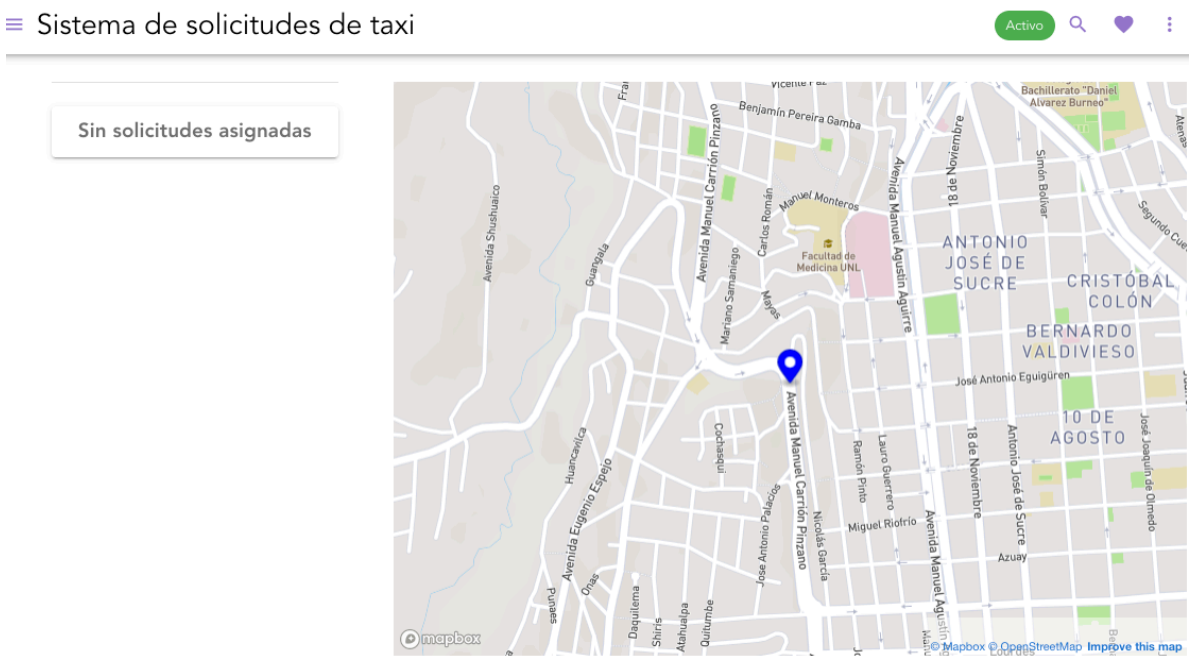

Fig. A13. 6. Sistema principal del conductor una vez tengo acceso al sistema

La Fig. A13. 7, Se muestra la asignación de una solicitud al conductor desde el microservicio de autenticación para lo cual utiliza el sistema está conectado a socket para la comunicación en tiempo real.

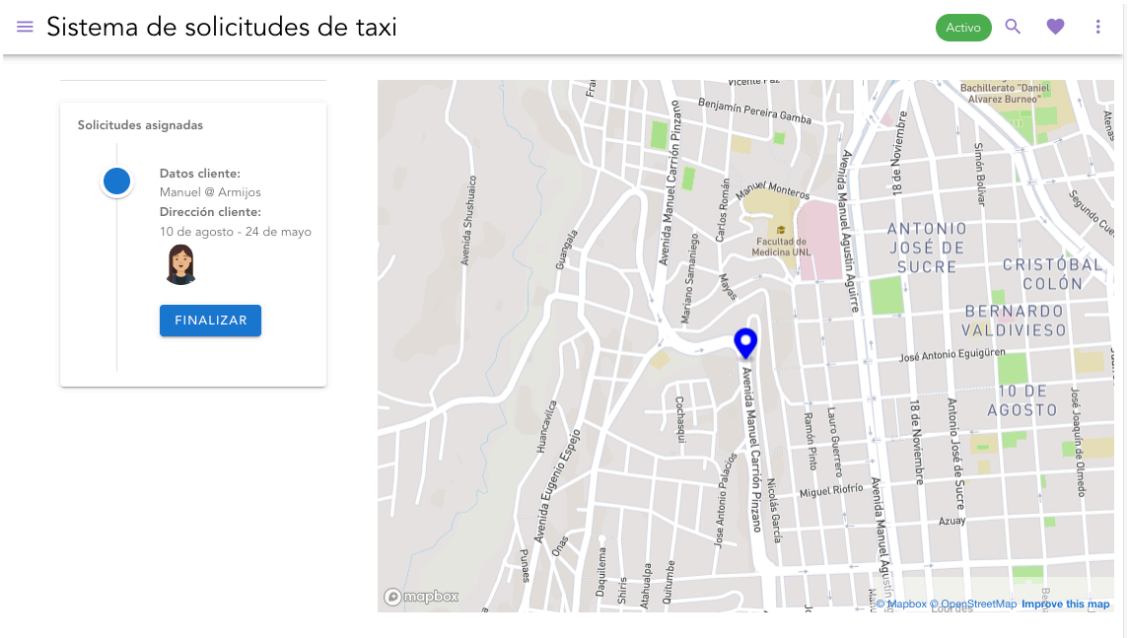

Fig. A13. 7. Asignación de una solicitud proveniente del cliente, socket es en encargo de avisarle de la asignación

Anexo 14. Socialización de los diseños y prototipo propuesto de la arquitectura basada en microservicios

Personal Involucrado

El personal involucrado para la socialización del presente tema de titulación fue asignado por el Ing. Jonathan Arrobo, Director de Tecnologías, los mismo que se pueden visualizar en la Fig. A14. 1.

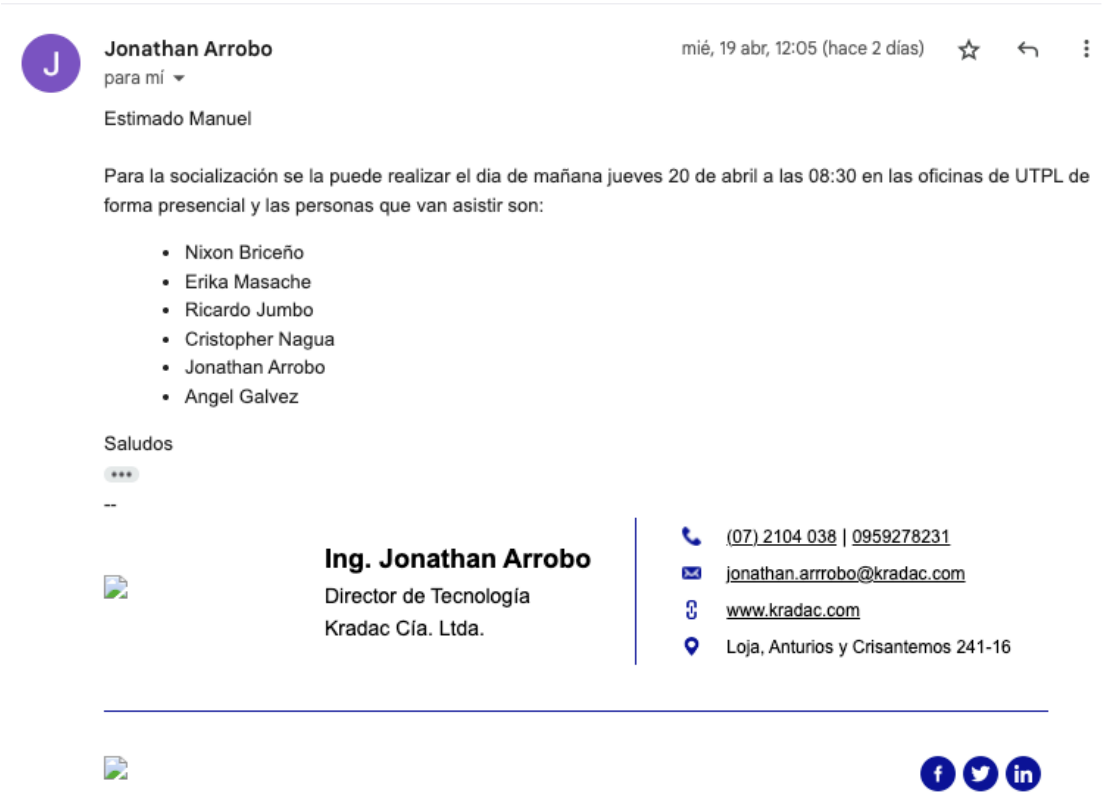

Fig. A14. 1. Personal involucrado en las pruebas de la arquitectura

La socialización de los diseños y el prototipo propuesto se realizó con profesionales de la empresa Kradac Cia. Ltda., los cuales consideran que la arquitectura propuesta funciona y cuyo objetivo es implementarla en los proyectos nuevos de la empresa.

En las imágenes posteriores se puede visualizar la socialización de los diagramas establecidos, el prototipo propuesto y las tecnologías utilizadas para su desarrollo.

Fig. A14. 2. socialización del diagrama general de la arquitectura propuesta

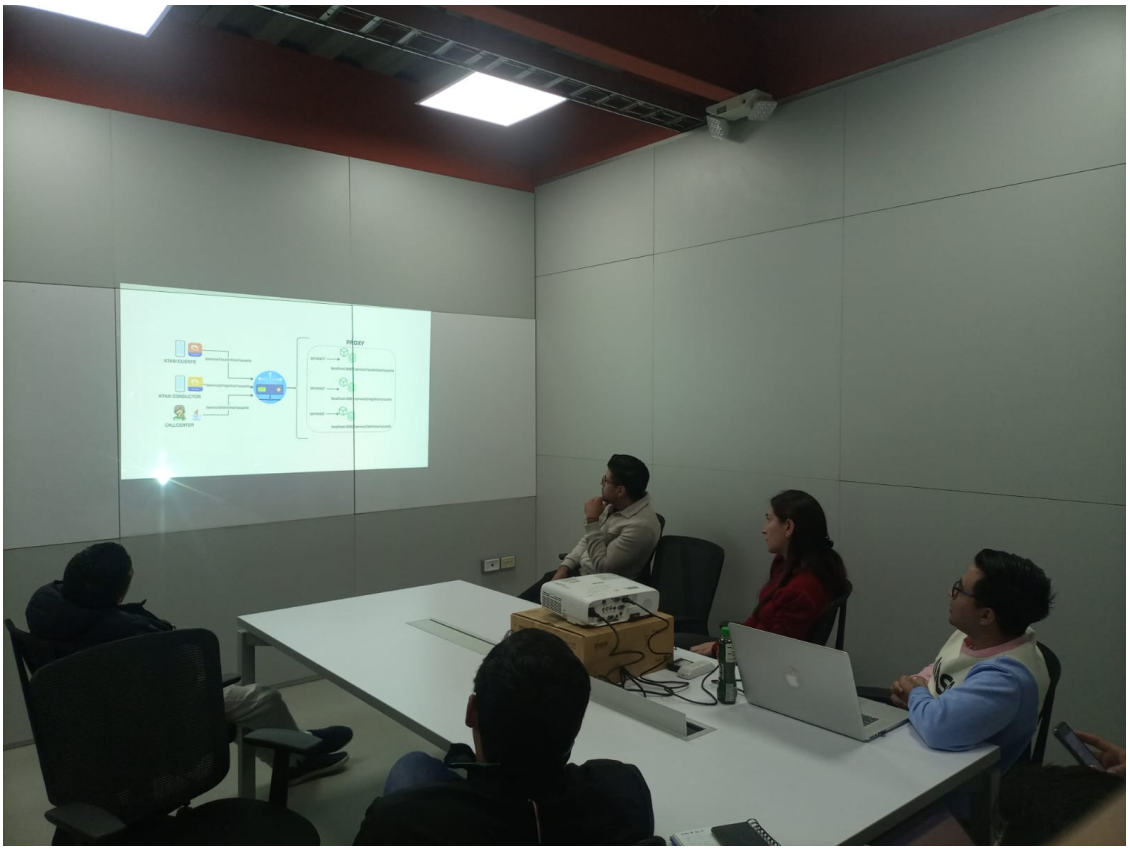

Fig. A14. 3. Socialización del diagrama del componente API Gateway de la arquitectura propuesta

Fig. A14. 4. Socialización de la seguridad del componente API Gateway

Fig. A14. 5. Personal involucrado en la Socialización del proyecto de titulación

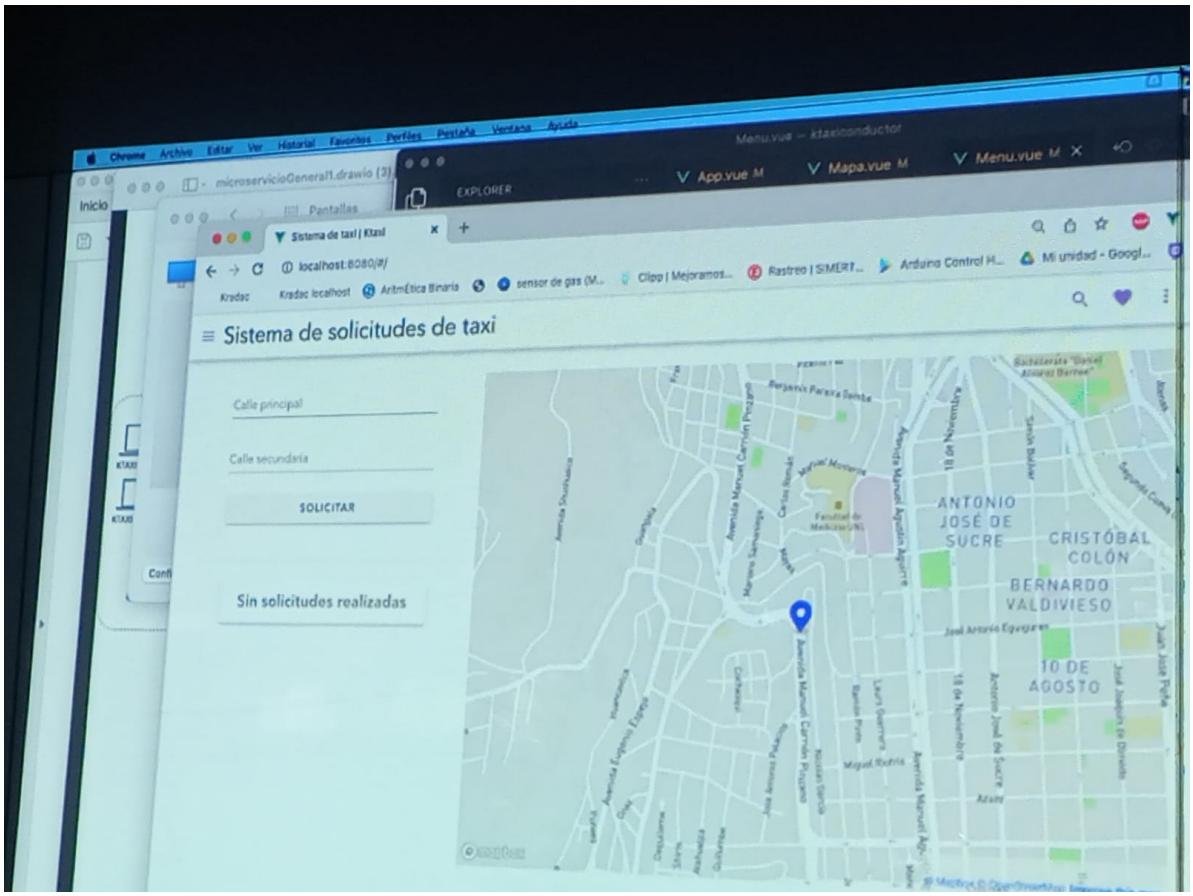

Fig. A14. 6. Socialización del prototipo funcional

Anexo 15. Certificación de la traducción del resumen

Certificación de traducción al idioma inglés.

Lic. Hernán Ezequiel Jiménez Armijos Mg. Sc.
LICENCIADO EN IDIOMA INGLÉS

Yo, Mgtr. Hernán Ezequiel Jiménez Armijos, con cédula de identidad 1106061896, licenciado en Ciencias de la Educación mención Idioma Inglés, registrado en la Senescyt con número **1008-2018-1998230**

CERTIFICO:

Que he realizado la traducción de español a inglés del resumen derivado de la tesis denominada: **Diseño de una arquitectura basada en microservicios en la nube para mejorar la escalabilidad y agilidad del aplicativo Ktaxi de la empresa Kradac Cia. Ltda.**, de autoría de **Manuel Stalin Armijos Ordóñez,** portador de número de cédula **1105593238,** estudiante del programa **"Maestría en Ingeniería en Software"** de la Facultad de la Energía, las Industrias y los Recursos Naturales no Renovables de la **Universidad Nacional de Loja,** la misma que se encuentra bajo la dirección del **Ing. Roberth Gustavo Figueroa Díaz, Mg.Sc.**, previo a la obtención del título de magíster en ingeniería en software.

Es todo cuanto puedo certificar en honor a la verdad, facultando al interesado hacer uso del presente en lo que creyere conveniente.

Loja, 26 de abril de 2023.

Lic. Hernán Ezequiel Jiménez Armijos Mg. Sc.
Docente / Coordinador del Área de inglés (sección Básica) de la UEPEE