FACULTAD DE INGENIERÍA, ARQUITECTURA Y URBANISMO

ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMAS

TESIS

Desarrollo de una aplicación móvil basada en microservicios para agilizar el proceso de ventas en un minimarket

PARA OPTAR EL TÍTULO PROFESIONAL DE INGENIERO DE SISTEMAS

Autor(es)

Bach. Chuman Lluen Dagner Anibal
https://orcid.org/0000-0002-1632-892X

Asesor(a)
Mg. Cachay Maco Junior Eugenio
https://orcid.org/0000-0003-4056-3142

Línea de Investigación

Infraestructura, Tecnología y Medio Ambiente

Pimentel – Perú

2023

DESARROLLO DE UNA APLICACIÓN MOVIL BASADA EN MICROSERVICIOS PARA AGILIZAR EL PROCESO DE VENTAS EN UN MINIMARKET

Aprobación del jurado

MG. MEJIA CABRERA HEBER IVAN

Presidente del Jurado de Tesis

MG. ARCILA DIAZ JUAN CARLOS

Secretario del Jurado de Tesis

MG. CACHAY MACO JUNIOR EUGENIO

Vocal del Jurado de Tesis

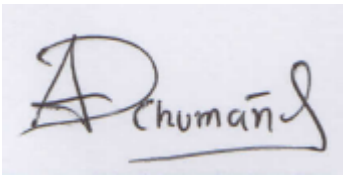
DECLARACIÓN JURADA DE ORIGINALIDAD

Quien(es) suscribe(n) la DECLARACIÓN JURADA, soy egresado **Dagner Anibal Chuman Lluen**. del Programa de Estudios de **Ingeniería de Sistemas** de la Universidad Señor de Sipán S.A.C, declaro bajo juramento que soy autor del trabajo titulado:

DESARROLLO DE UNA APLICACIÓN MÓVIL BASADA EN MICROSERVICIOS PARA AGILIZAR EL PROCESO DE VENTAS EN UN MINIMARKET

El texto de mi trabajo de investigación responde y respeta lo indicado en el Código de Ética del Comité Institucional de Ética en Investigación de la Universidad Señor de Sipán, conforme a los principios y lineamientos detallados en dicho documento, en relación con las citas y referencias bibliográficas, respetando el derecho de propiedad intelectual, por lo cual informo que la investigación cumple con ser inédito, original y autentico.

En virtud de lo antes mencionado, firman:

CHUMAN LLUEN DAGNER ANIBAL	DNI: 73583958	

Pimentel, 24 de noviembre de 2023.

Dedicatoria

"Consagro este trabajo con sincero agradecimiento a mis padres José Ramon y María del Pilar e hnos., que me enriquecí con el afecto y aliento que me brindaron durante las noches, representando la fuerza especial y el amor paternal.

Lo dedico también a mis Profesores de la USS, que me hicieron ser fuerte, luchar por los sueños, en cada dialogo, o cada actitud han sido una fuente para motivarme. El conocimiento compartido y han dejado marca perdurable en mi trayectoria educativa, por lo cual estoy sinceramente agradecido

Dedico este proyecto con profundo amor a mi tío Cesar y tía Iris desde el cielo. Donde este prototipo no solo busca cumplir un propósito técnico, sino que también esta imbuido como inspiración, recordando la fortaleza y conexión que me brindaron.

Y por último dedico a todos los materiales en cada código escrito, en cada desafío superado en este viaje tecnológico. A mis amigos pilares de apoyo que, con cada risa compartida, confianza y consejo brindados llevaron a que este proyecto se cumpla de poco a más."

Agradecimientos

“Agradecer a mis padres José Ramon y María del Pilar, cuyo apoyo inquebrantable ha sido fundamental en mi trayectoria académica. Desde el primer día de clases hasta cada desafío que enfrenté, su aliento y orientación han sido el cimiento de mi éxito. Sus sacrificios y dedicación para brindarme oportunidades educativas no tienen precio, por lo que estoy infinitamente agradecido por la invasión constante también que realizaron en mi crecimiento académico.

Agradecer a Dios ya que ha sido mi luz durante lo largo de los años, cada conversación con él sobre mis estudios se convirtió en mi todo para que sea todo con éxito.”

Índice

Índice de tablas, figuras y fórmulas

DESARROLLO DE UNA APLICACIÓN MOVIL BASADA EN MICROSERVICIOS PARA AGILIZAR EL PROCESO DE VENTAS EN UN MINIMARKET

Resumen

El proyecto se enfoca en desarrollar una aplicación móvil basada en microservicios utilizando la metodología XP (Extreme Programming) para agilizar el proceso de ventas en Minimarkets y evitar la congestión de clientes en las colas de espera. La implementación de microservicios, que abarca áreas clave como API Gateway, Productos, Categoría, Negocio y Compra, ha demostrado ser altamente eficaz bajo los principios ágiles de XP. Con tiempos de respuesta promedio de 4431 ms para 20 peticiones por segundo, se logró una mejora sustancial en la eficiencia del proceso en comparación con el sistema anterior, que registraba 10000 ms, reduciendo el tiempo de espera en un 95.69%. Además, se aplicó la norma ISO/IEC 25010 para evaluar la calidad del software, garantizando un enfoque integral. La aplicación, concebida bajo la metodología XP, ha reducido drásticamente el tiempo de espera para los clientes a 2500 ms, en comparación con los 2 minutos por cliente sin la aplicación, mejorando significativamente la experiencia de compra. La propuesta busca evitar la congestión de clientes en colas físicas, ofreciendo la opción de realizar compras y pedidos a través de la plataforma web y móvil. La implementación de un sistema automatizado, un pilar de XP, ha permitido a los negocios gestionar transacciones eficientemente y controlar el flujo de clientes en tiempo real. Resultados indican un bajo acoplamiento entre los microservicios, sugiriendo alta reusabilidad, y buena cohesión entre componentes para facilitar la comprensión y mantenimiento del sistema.

Palabras Clave: Microservicios, aplicación móvil, proceso de ventas, minimarket

Abstract

The project focuses on developing a mobile application based on microservices using the XP (Extreme Programming) methodology to streamline the sales process in Minimarkets and avoid customer congestion in waiting queues. The implementation of microservices, covering key areas such as API Gateway, Products, Category, Business and Purchasing, has proven to be highly effective under the agile principles of XP. With average response times of 4431 ms for 20 requests per second, a substantial improvement in process efficiency was achieved compared to the previous system, which recorded 10000 ms, reducing waiting time by 95.69%. In addition, the ISO/IEC 25010 standard was applied to evaluate software quality, ensuring a comprehensive approach. The application, conceived under the XP methodology, has drastically reduced the wait time for customers to 2500 ms, compared to 2 minutes per customer without the application, significantly improving the shopping experience. The proposal seeks to avoid customer congestion in physical queues, offering the option of making purchases and orders through the web and mobile platform. The implementation of an automated system, a pillar of XP, has allowed businesses to efficiently manage transactions and control the flow of customers in real time. Results indicate a low coupling between the microservices, suggesting high reusability, and good cohesion between components to facilitate the understanding and maintenance of the system.

Keywords: Microservices, mobile application, sales process, minimarket

I. INTRODUCCIÓN

1.1. Realidad problemática

La adopción masiva de tecnología es inminente, y este auge digital está cambiando la forma en que los consumidores buscan y compran productos, y servicios. Según se menciona [1] Según comentarios recibidos durante el proceso, se atribuyeron los problemas del estudio a la existencia de largas filas en el punto de minimarket para realizar los pagos o adquisición de los productos para poder añadir a sus puntos de compras de cada usuario. Estas filas generaban dificultades relacionadas con la naturaleza propensa a errores de este proceso para los cajeros, así como con la incertidumbre asociada al trabajo diario de utilizar dinero físico.

Sin embargo, sabemos que no todo el 100% de los usuarios llegará para usar este método de solución Según Want et al. [2] menciona que más de dos millones de usuarios peruano utilizan sus smartphones para adquirir productos. Viendo que no existen muchos aplicativos que puedan tener los clientes o que las mismas empresas brindan para sus clientes.

Existen varias formas de que estás dificultades de compras tecnológicas puedan resolverse, para llevar una mejor calidad de eficiencia con respecto al uso de los clientes, y auto llevar un incremento de compras y ganancias para los negocios.

El impacto de la adopción de aplicaciones móviles de los clientes en los comportamientos de compra. Específicamente, investigan la canibalización de los canales físicos y en línea existentes por la aplicación móvil recientemente adoptada y evalúan los cambios en los gastos totales de los hogares en la cadena focal, las aplicaciones móviles son mayor en los mercados donde la cadena focal enfrenta una competencia más intensa de un competidor cercano por lo que adoptar esta forma de uso haría generar ganancias para los negocios [3].

Las aplicaciones de compras móviles se han convertido en un canal popular para llegar a los consumidores ubicuos de hoy. Sin embargo, el espacio de mercado se ha vuelto altamente competitivo, donde se ven técnicas de aceptación y uso continuo para conceptualizar cómo obtener intención de recompra. Utilizando un método de muestreo no probabilístico por conveniencia, se encuestó en línea a un total de 245 consumidores que habían comprado previamente a través de una aplicación de compras móviles [4].

Para poder desarrollar esta investigación técnica eficaz, y consistente en una aplicación móvil que permita a los clientes interactuar con sus compras utilizando tecnología para encontrar productos, comunicaciones, promociones, información de productos y más [5].

Las empresas globales en respuesta a las presiones de crecimiento que enfrentaban con respecto al volumen de actividad que realizaban sus sistemas supera la capacidad de las opciones tecnológicas originales, en el desarrollo del lenguaje .NET es más eficiente en el manejo de solicitudes de comunicación que Java. Esto se puede atribuir a un alto nivel de optimización del rendimiento en las bibliotecas de .NET Core, las aplicaciones monolíticas como de microservicio, respectivamente en un 5 % y un 1,5 %. Como se puede definir una arquitectura de microservicios no es la más adecuada para todos los contextos [6].

La invención de la tecnología móvil, específicamente los teléfonos móviles (inteligentes), ha revolucionado no sólo las prácticas comerciales, sino también el comportamiento del consumidor. los consumidores tienden a usar puntos de contacto basados en dispositivos móviles, dada la gran cantidad de aplicaciones y vías digitales disponibles en el mercado, las empresas se enfrentan a la abrumadora tarea de garantizar que los consumidores sigan comprando desde sus aplicaciones de compras móviles en lugar de cambiar a ofertas de la competencia [4].

En la historia de las arquitecturas de software se ha caracterizado en las últimas décadas por un cambio progresivo hacia distribución, modularización y acoplamiento flexible,

con el fin de aumentar la reutilización y la robustez del código de la arquitectura monolítica heredada y la nueva arquitectura de microservicios. Al comparar los dos diseños arquitectónicos, se ve cómo los microservicios condujeron a una mejor escalabilidad, así como ofrecer soluciones a los principales problemas causados por la realización monolítica [7].

En la industria de la agricultura el regadío para satisfacer las necesidades alimentarias de una población mundial donde se ve la optimización de la gestión del riego es crucial en las zonas semiáridas donde la escasez de agua es un problema grave. Donde se utiliza una plataforma IoT (IRRISENS) basada en microservicios totalmente replicables que se utilizan para detectar parámetros del suelo, cultivos y atmósfera, interactuar con servicios en la nube de terceros para programar el riego y, potencialmente, controlar el riego automáticamente. A partir de la implementación de la plataforma IoT en la producción de arroz y algodón, se identificaron cinco requisitos principales para que las plataformas IoT se utilicen en la agricultura a escala comercial: escalabilidad, flexibilidad, heterogeneidad, robustez ante fallas y seguridad [8].

La Nube y el borde informático han sido ampliamente adoptados en muchos escenarios de aplicación. Con la creciente demanda de una iteración rápida y la complejidad de la lógica comercial, es un desafío lograr un desarrollo rápido y una entrega continua en una nube y un borde tan altamente distribuidos informática ambiente. Donde se define SPPMS, el problema de ubicación de servicios en sistemas de microservicios que presentan dependencias complejas y múltiples instancias [9].

En el desarrollo continuo de Internet de las cosas, la computación en la nube se ha aplicado en muchos campos, cómo garantizar la calidad del servicio, como baja latencia, alto ancho de banda, alta confiabilidad, etc., teniendo que modelar y optimizar la confiabilidad de las aplicaciones en la nube basadas en microservicios utilizando un sistema multiagente (MAS), maximizando así la confiabilidad de la computación en la nube y programando

dinámicamente los microservicios para minimizar el retraso dentro del presupuesto, donde no solo puede reducir el costo de ejecución de tareas, sino también obtener una mayor utilización de recursos y lograr el efecto esperado del algoritmo [10].

El problema es en los desarrollos recientes en el campo de la computación en la nube han estimulado la necesidad de crear nuevas soluciones de software para el monitoreo, la gestión y la optimización de los servicios de una organización, evaluando y contrastando el rendimiento de los patrones arquitectónicos más utilizados para el desarrollo de aplicaciones tanto por servicios y microservicios, donde la implementación de SOA y MSA depende de la naturaleza y las necesidades de las organizaciones consumía significativamente más ancho de banda que SOA, y la escalabilidad en SOA generalmente no es posible o requiere un esfuerzo significativo para lograrse [11] .

Mientras que la cantidad de los usuarios en sus teléfonos inteligentes y dispositivos móviles va de incremento significativamente. También se han incrementado las Aplicaciones Móviles en la Nube basadas en cloud computing. No obstante, en la creación del simulador, y una nueva programación de tareas para un marco de computación en la nube móvil basado en microservicios. De un sistema MSCMCC de computación en la nube, para ejecutar las aplicaciones de microservicios para aplicaciones eficientes sensibles a demoras y un marco consciente de la movilidad para superar el costo de la aplicación [12].

En la actualidad vemos que los académicos y profesionales de negocios se enfocan en aplicar microservicios en la computación en la nube, pero seleccionar y combinar microservicios disponibles es un problema difícil de NP. Los diferentes tipos de microservicios se organizan juntos, y luego estos microservicios se combinan y construyen para cumplir con los requisitos de calidad de los usuarios, lo que describe el problema de composición de los microservicios. Los usuarios solo necesitan proporcionar el orden de importancia de los atributos para obtener sus pesos. Basado en IMCSA, o ParaCoSIMCSA demostró un

rendimiento superior en comparación con los algoritmos de selección clonal tradicionales y el algoritmo de clonación de memoria inmune [13].

Una arquitectura de microservicios es una composición de diminutos bloques de construcción de componentes de software distribuidos de forma flexible y de grano fino. Para proponer un mecanismo que pueda observar y monitorear continuamente la arquitectura de microservicios y ser capaz de detectar comportamientos anómalos con alta precisión y generar una baja tasa de falsas alarmas. Como resultado La tasa de verdaderos positivos (TPR), la sensibilidad o el recuerdo es 49/(49 + 6) = 89 %, la tasa de falsos positivos (FPR) es 38/1473 = 2,5 %, la tasa de falsos negativos (FNR) es 6/(49 + 6) = 11% y la tasa negativa verdadera (TNR), especificidad o selectividad es 1435/1473 = 97,5%. La precisión (PPV) del algoritmo es 49/(49 + 38) = 56,3% mientras que su tasa de omisión falsa (FOR) es 6/6 + 1435 = 4,4%. La tasa de descubrimiento falso (FDR) es 38/49 + 38 = 43,7 % mientras que su valor predicho negativo (NPV) es 1435/6 + 1435 = 99,5 %. Por tanto, la precisión global del algoritmo es (49 + 1435) / (49 + 38 + 6 + 1435) = 97,1 %. Los usos del modelo basado en la utilidad permiten que la arquitectura elija dinámicamente un enfoque de razonamiento basado en la dimensión de utilidad más alta de los cambios de contexto en el entorno operativo [14].

En comparación con la mayoría de los algoritmos de reconocimiento de emociones, los recursos informáticos y la duración de la batería de los teléfonos móviles siempre son limitados. Esto hace que la precisión y la latencia de estas aplicaciones sean insatisfactorias. La precisión y la latencia de una aplicación basada en MSPMERAD se comparan con otra aplicación que se implementa utilizando un algoritmo de reconocimiento de emociones local [15].

Las técnicas basadas en microservicios proporcionan una metodología de grano fino y ligeramente acoplada. Con el uso de la técnica de microservicios presentada en esta investigación, se mejoran la eficiencia, la escalabilidad y el rendimiento [16].

En la sinopsis de la presente investigación para alcanzar el objetivo general con eficiencia los objetivos específicos establecidos han sido la construcción del diseño e implementación de una aplicación móvil con la finalidad de agilizar el proceso de compra – venta en muchos negocios o minimarkets en referencia al prototipo y con la ayuda de procesos óptimos como los microservicios, y en la nube como GitHub para desplegarlo en la nube con sus pruebas, en este caso se usó el servicio local primero, para después compartido por la propia red del proveedor de servicios de “Heroku”, en donde se hicieron las peticiones correspondientes para obtener los resultados.

1.2. Formulación del problema

¿De qué manera proponemos realizar las compras mediante las plataformas móvil y servicio en el aplicativo a desarrollar para evitar la congestión de clientes en las colas de espera de ventas de minimarkets?

1.3. Hipótesis

Mediante la implementación de una aplicación móvil basada en microservicios se agilizará el proceso de ventas de minimarkets.

1.4. Objetivos

Objetivo general

Desarrollar una aplicación móvil basada en microservicios para agilizar el proceso de ventas de minimarkets.

Objetivos específicos

- Realizar el análisis de los requerimientos funcionales y no funcionales para gestionar el proceso de ventas de minimarkets.
- Seleccionar las herramientas de microservicios para el desarrollo de la aplicación móvil.
- Implementar la arquitectura de microservicio para la aplicación móvil.
- Evaluar los resultados obtenidos de acuerdo a los indicadores establecidos.

1.5. Teorías relacionadas al tema

1.5.1. Servicios Web:

[16] Los servicios web son aplicaciones modulares independientes que se pueden describir, publicar, ubicar e invocar a través de una red. Implementan una arquitectura orientada a servicios (SOA), que admite la conexión o el intercambio de recursos y datos de manera flexible y estandarizada. Los servicios se describen y organizan para respaldar su descubrimiento y reutilización dinámicos y automatizados.

1.5.2. API:

Son agrupaciones entre funciones, procedimientos que integrará el acceso de terceros para que los servicios se creen.

1.5.3. Aplicaciones web:

[17] Una aplicación web se compone de uno o más servlets relacionados, tecnología JavaServer Pages (archivos JSP) y archivos de lenguaje de marcado de hipertexto (HTML) que puede administrar como una unidad.

[20] **“**Una aplicación web son más rápidas construir y probar, además reducen costo de desarrollo y son fáciles de mantener”**,** para reconocer que las aplicaciones web difieren de los sitios web en la interacción mejorada proporcionada por los usuarios que interactúan para cambiar el contenido y hacer que el contenido sea dinámico".

Fig. 1. Ventajas y desventajas aplicaciones web app.

Obtenido de [20]

1.5.4. Arquitectura de Software:

La arquitectura de software es la estructura fundamental que define un sistema de software, incluyendo sus componentes, sus relaciones, los principios y pautas que rigen su diseño y evolución. La arquitectura de software sirve como una guía para la toma de decisiones de diseño y desarrollo, permitiendo a los equipos de software comprender mejor la complejidad del sistema y mantener su calidad a lo largo del tiempo [21].

1.5.4.1. Proceso de la Arquitectura de Software:

El proceso de arquitectura de software se despliega mediante un enfoque iterativo de tres pasos: definición de los requisitos arquitectónicos, diseño de la arquitectura y validación de la misma. Este ciclo iterativo posibilita una adaptación continua durante el desarrollo, permitiendo ajustes y mejoras a medida que avanza el proyecto.

1.5.4.2. Documentación de la Arquitectura:

La documentación juega un papel esencial en la comunicación entre las partes interesadas y sirve como cimiento para el análisis y la construcción del sistema. Se emplean vistas lógicas, de comportamiento y físicas para representar distintos aspectos de la arquitectura, proporcionando una comprensión integral y detallada del sistema.

1.5.4.3. Metodología de Desarrollo de Software:

Se destacan enfoques específicos que guían la construcción de sistemas robustos. Entre estos, el Diseño Guiado por Atributos (ADD), un método iterativo, descompone la arquitectura en subelementos en cada iteración, generando así una arquitectura conceptual mediante el uso de patrones y tácticas predefinidas. Otro enfoque valioso es el Método de Diseño Centrado en la Arquitectura (ACDM), que proporciona criterios y pautas para la fase de diseño arquitectónico, respaldando las tareas a lo largo del ciclo de desarrollo de software. Además, se incorpora la metodología ágil Scrum, respaldada por autores contemporáneos como Schwaber y Sutherland [36]. Scrum, conocida por su flexibilidad y adaptabilidad, se alinea con las prácticas ágiles, promoviendo la entrega iterativa y colaborativa de software. Este enfoque integral, combinando Design Thinking, ACDM, y Scrum, establece una base sólida para abordar los desafíos arquitectónicos y de desarrollo de software de manera efectiva.

1.5.4.4. Selección de Componentes para Arquitectura de Microservicios:

Enfocado específicamente en SOA, el enfoque de microservicios desarrolla una aplicación como un conjunto de servicios pequeños, desplegados de manera independiente. Cada microservicio opera en su propio proceso y se comunica mediante mecanismos ligeros .

La arquitectura basada en microservicios es desarrollada organizativamente para el desarrollo de software donde el desarrollador lo descompone por pequeños

servicios independientes por cada módulo que se comunican a través de API que serán sus funcionamientos y procedimientos.

La arquitectura de software de microservicios es un estilo arquitectónico para sistemas de software distribuidos, en el que un sistema se divide en servicios pequeños, independientes, autónomos y que se comunican mediante mecanismos ligeros. Cada servicio se enfoca en una única funcionalidad de negocio y se puede construir, desplegar y escalar de forma independiente. La arquitectura de microservicios permite una mayor agilidad y velocidad en el desarrollo, y una mejor escalabilidad y resiliencia del sistema en producción [22][23].

Según la Visión monolítica:

Puede utilizar el servidor para visitar el sitio web de la tienda y realizar compras. En el servidor, la máquina ejecutará el código de la aplicación en forma de archivo war y eventualmente se conectará a la base de datos para recuperar información. Cuando un usuario compra un producto a través de la web, muestra los productos disponibles y más. Su aplicación reaccionará de alguna manera a medida que la programe, pero el código para las diversas lógicas comerciales (inventario, pago, envío) terminará en un solo archivo ejecutable cuando se divida en varias ventanas de código. Cambiar la ventana de pago requiere volver a cargar todo el código a producción, incluidos los módulos de inventario y envío, incluso si no han cambiado. Además, para servir a muchos usuarios, una copia de la aplicación debe ejecutarse en un balanceador de carga, redirigiendo a los usuarios a uno de los sitios dependiendo de si el sistema está muy ocupado o no.

[24] Una arquitectura monolítica, todos los procesos están estrechamente acoplados y se ejecutan como un solo servicio. Esto significa que a medida que crece

la demanda del proceso de aplicación, toda la arquitectura debe escalar. Agregar o mejorar la funcionalidad en una aplicación monolítica se vuelve más complejo a medida que crece la base de código. Esta complejidad limita la experimentación y dificulta la implementación de nuevas ideas. Las arquitecturas monolíticas aumentan el riesgo de disponibilidad de las aplicaciones porque muchos procesos dependientes y estrechamente acoplados aumentan la incidencia de fallas en los procesos.

Según los microservicios:

En lugar de tener un solo ejecutable para el servidor, cada componente, módulo o ventana es su propio ejecutable y los servicios se comunican entre sí a través de llamadas. En este caso, también tenemos un microservicio que implementa la interfaz web con la que interactúa el usuario y cierta lógica básica para llamar a los microservicios de pago, inventario y envío según sea necesario.

[24] Con una arquitectura de microservicios, donde se construye con componentes separados que realizan cada proceso de aplicación como un servicio. Estos servicios se comunican a través de una interfaz bien definida que utiliza API livianas. Los servicios están diseñados para oportunidades de negocio y cada servicio realiza una sola función. Debido a que funcionan de forma independiente, cada servicio se puede actualizar, implementar y escalar para satisfacer la demanda de características de aplicaciones específicas.

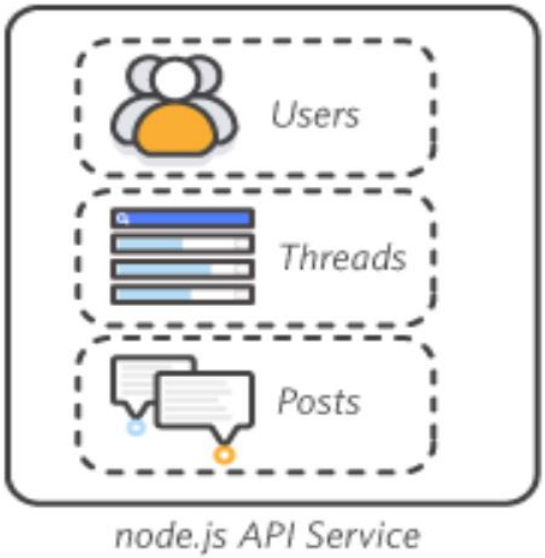


Fig. 2 Arquitectura microservicios

Obtenido de [24]

1.5.5. Spring Boot:

Spring Boot es un framework para aplicaciones Java basado en la plataforma Spring que tiene como objetivo simplificar el proceso de creación, configuración y despliegue de aplicaciones de forma rápida y sencilla [25]. Esto quiere decir que Spring Boot ofrece una forma rápida y sencilla de crear aplicaciones Java listas para producción con poco o ningún código de configuración.

1.5.6. Concurrencia:

La concurrencia en Spring Boot es clave para construir aplicaciones reactivas y resistentes, y se puede lograr utilizando técnicas como el enrutamiento basado en controladores y la programación reactiva para mejorar la capacidad de respuesta y la escalabilidad de la aplicación [26], la alta concurrencia como hilos o colas. Debido a los muchos usuarios accedieron a la alta concurrencia, lo que resultó en datos incorrectos del sistema y pérdida de datos, pensando tal vez en usar colas. De hecho, el método de resolución de la cola también se puede procesar. Por ejemplo, estamos haciendo una oferta para productos, reenviando comentarios en Weibo o productos de pico. Al

mismo tiempo, el tráfico es particularmente grande. La cola juega un papel especial aquí. Todas las solicitudes se colocan en la cola y se cuentan en milisegundos. Las unidades se llevan a cabo de manera ordenada para que no se pierdan datos y los datos del sistema sean incorrectos.

Después de verificar la información, hay dos soluciones para la alta concurrencia, una es usar caché y la otra es generar páginas estáticas; y la otra es optimizar el código desde el lugar más básico para reducir el desperdicio innecesario de recursos.

Proponen un resultado de encuesta al cliente, basándose en una pregunta. ¿Descargaría una App que ofrece mejorar su experiencia de compra en un supermercado? [27].

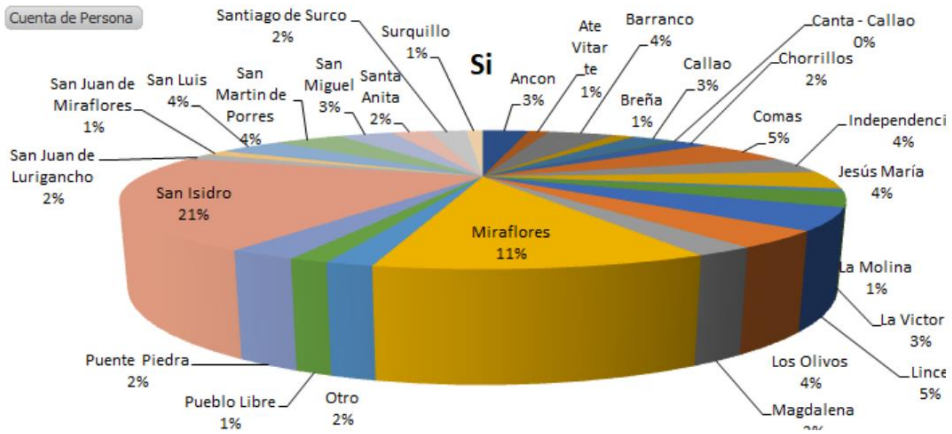


Fig. 3. Encuesta de experiencia de compras por una app

Obtenido de [27]

1.5.7. Arquitectura SOA

La Arquitectura Orientada a Servicios nos diseñan una nueva planificación para los recursos en procesos de la organización, de tal forma las aplicaciones complejas que se puedan ver, se vean simplificados y tener la facilidad de adherirse unos a otros,

es decir mayor flexibilidad para poder ser aplicadas todas las aplicaciones que se hagan a convertirse en uno solo [28].

1.5.8. **Calidad en el Producto de Software y en el Proceso de Software**

La serie de normativas ISO/IEC 25000, también reconocida como SQuaRE, despliega un marco para la evaluación de la excelencia en productos de software. La norma ISO/IEC 25040, enmarcada en este contexto, delineación el procedimiento para evaluar la excelencia del producto, mientras que la ISO/IEC 25010 establece una serie de características y subcaracterísticas que funcionan como fundamentos para evaluar la excelencia en productos de software. Este enfoque exhaustivo busca definir la esencia misma de la excelencia en productos de software y procesos asociados, proporcionando un conjunto estructurado de directrices para la evaluación precisa y estandarizada [21][39].

Fig. 4. Calidad del Producto de Software.

Obtenido de [31]

II. MÉTODO

2.1. Tipo y Diseño de Investigación

2.1.1. Tipo de Investigación

La investigación propuesta es de tipo cuantitativo, fundamentado en la aplicación de una base de investigaciones científicas relacionadas con el aprendizaje del uso de los microservicios. En este enfoque, se llevará a cabo la recopilación de pruebas o experimentos con el objetivo de identificar y analizar las relaciones existentes entre las variables involucradas en el aprendizaje de estos servicios.

Este enfoque cuantitativo permitirá obtener datos numéricos y objetivos, lo que facilitará el análisis estadístico y la identificación de patrones y tendencias en el uso de los microservicios. Además, el énfasis en la recopilación de pruebas y experimentos brindará una base sólida para establecer conexiones causales y entender e extender el impacto de distintos contextos.

2.1.2. Tipo de Diseño

El diseño de esta investigación es cuasi experimental, ya que se centra en el estudio de la relación entre una variable independiente y una variable dependiente.

2.2. Variables, Operacionalización

Este procedimiento implica la especificación de indicadores concretos, sobre la capacidad de medir o evaluar la investigación científica.

TABLA I

Operacionalización de la variable (ISO 25010)

Variable de estudio	**Definición conceptual**	**Definición operacional**	**Indicadores**	**Instrumento**	**Valores finales**	**Tipo de variable**	**Escala de medición**
Aplicación móvil basada en microservicios	Una aplicación móvil basada en microservicios es aquella que se construye utilizando la arquitectura de microservicios, donde se divide la aplicación en pequeños servicios independientes y autónomos que realizan una tarea específica.	Una aplicación móvil basada en microservicios es aquella que utiliza una arquitectura de microservicios para proporcionar sus funcionalidades a los usuarios de dispositivos móviles, como smartphones y tablets.	Granularidad Acoplamiento Cohesión Complejidad Reusabilidad	Observación (Bitácora de resultados)	número	Independiente	Nominal)

Congestión de colas en el proceso de compras electrónicas	La congestión de colas en el proceso de compras electrónicas se refiere a la situación en la que una gran cantidad de usuarios intentan realizar una compra al mismo tiempo en un sitio web de compras en línea, lo que puede provocar una sobrecarga del sistema y retrasos en el procesamiento de las transacciones.	La congestión de colas en el proceso de compras electrónicas se produce cuando la demanda de transacciones supera la capacidad del sistema para procesarlas, lo que puede llevar a una experiencia de usuario deficiente y la pérdida de ingresos para el negocio. Para prevenirla, se pueden implementar diversas soluciones técnicas que mejoren la capacidad y la eficiencia del sistema.	Tiempo de respuesta Tiempo de espera	Observación (Bitácora de resultados)	Porcentaje (%)	Dependiente	Razón

2.3. Población de estudio, muestra, muestreo y criterios de selección

2.3.1. Población

La población está conformada por las peticiones de consumo realizadas a la arquitectura de software basada en microservicios en un tiempo determinado que puede ser modelada considerando el Minimarket Job como un caso específico. Dado que se espera el año en curso o futuro existan alrededor de 400 clientes mensuales en dicho contexto o este número aumente. Por lo tanto, tomaremos este número como la población total de peticiones en un momento dado.

2.3.2. Muestra

La selección de la muestra se realizó mediante un muestreo no probabilístico, utilizando el método de conveniencia. En el estudio llevado a cabo por Arcila Diaz [30], se emplearon 20 solicitudes por segundo para evaluar una aplicación web en un modelo de composición de microservicios. En nuestro propio estudio, asimismo, se optó por la misma cantidad de 20 solicitudes por segundo.

2.4.1. Técnicas

Se aplicó la búsqueda donde para documentar información en la base de datos bibliográfica SCOPUS compuesta integradamente en revistas científicas con la finalidad de obtener artículos para la búsqueda de antecedentes. Asimismo, se utilizaron fórmulas para así mismo obtener información de fuente confiables para las teorías relacionadas al tema y realidad problemática.

2.4.2. Instrumentos de recolección de datos

El instrumento que se aplicará será la ficha de la observación según resultados de la empresa obtenida y así mostrará en las interfaces de producción Heroku.

2.5. Procedimiento de análisis de datos

De acuerdo al tema de investigación se implementan microservicios y en la evaluación se utilizaron métricas de calidad, teniendo en cuenta características de calidad: granularidad, coherencia, coherencia, complejidad y reutilización, tiempo de respuesta, tiempo de espera.

2.5.1. Granularidad

La granularidad de un servicio se describe como el tamaño apropiado de un servicio, se estima como la relación del cuadrado del número de funciones del servicio al cuadrado del número de mensajes consumidos por el servicio [29].

Donde:

$O(s)$: Es el conjunto de operaciones proporcionado por un servicio.

$M(s)$: Es el conjunto de mensajes de todas las operaciones de un servicio.

$$\frac{Número\ de\ operaciones}{Número\ de\ mensajes} = \frac{[O(s)]^2}{[M(s)]^2}$$

2.5.2. Acoplamiento

El acoplamiento de servicios se describe como la fuerza de las dependencias entre los servicios en un sistema, evaluado como el número promedio de servicios conectados directamente [29][30].

Donde:

nc : Número de servicios consumidores.

np: Número de servicios dependientes.

ns : Número total de servicios.

$$\frac{Número\ de\ servicios\ conectados}{Número\ de\ servicios} = \frac{Nc+Np}{Ns}$$

2.5.3. Cohesión

La cohesión describe la fuerza de la relación entre las operaciones dentro de un servicio y se evalúa como el recíproco del número promedio de mensajes utilizados [29][30].

Donde:

$M(s)$: Es el conjunto de mensajes de las operaciones de un servicio.

ns : Número total de servicios.

$$\frac{Ns}{M(s)}$$

2.5.4. Reusabilidad

La reusabilidad se describe en la reutilización de un microservicio determinado, medida como el número de usuarios existentes del servicio [29][30].

Reusabilidad = Sconsumers

Donde:

$Sconsumers$: Consumidores del servicio existentes.

2.5.6.Tiempo de respuesta

El propósito del Indicador de tiempo de respuesta es ayudar a acelerar el procesamiento de las solicitudes de información [29][30].

Tiempo de respuesta = Tiempo Final– Tiempo inicial

Donde:

Tiempo final: Tiempo final concreto después de la petición.

Tiempo inicial: Tiempo inicial del servicio de la petición.

2.5.7. Tiempo de espera

Al calcular la latencia de tiempo de espera se debe considerar el tiempo que se tarda en adquirir insumos, materias primas, herramientas y todos los elementos utilizados en el trabajo. Por lo tanto, si podemos negociar para acortar el tiempo de entrega, es posible mejorar la eficiencia de producción. [29][30].

$$\text{Tiempo de espera} = \frac{Tiempo\ de\ una\ petición}{Tiempo\ Total\ de\ peticiones}$$

2.6. Criterios éticos

Toda la información proporcionada por la empresa en donde utilizaremos la aplicación, será de uso único y exclusivo de acuerdo para las pruebas en el presente trabajo de tesis. De tal manera, protegemos la información bajo cualquier circunstancia que se preste para otros fines, que no sean de lograr resultados en el trabajo de investigación.

2.6.1. CONSENTIMIENTO INFORMADO:

He sido informado de que puedo hacer preguntas sobre el proyecto en cualquier momento y que puedo retirarme del mismo cuando así lo decida, sin que esto acarree perjuicio alguno para mi persona. De tener preguntas sobre mi participación en este estudio, puede contactar al dueño de la empresa, para obtener mi presencia.

2.6.2. CONFIDENCIALIDAD:

Las formas en que se usarán y harán disponibles los datos de la investigación son parte del acuerdo que se hará con los participantes de este estudio.

III. RESULTADOS Y DISCUSIÓN

3.1. Resultados

Descripción de herramientas tecnológicas y software

Se describe el equipo en el cual se ha desarrollado el aplicativo:

- Una laptop con procesador ryzen 7 3700u
- RAM 12 gb
- Disco solido 240 gb
- Sistema operativo Windows 10 x64 bits
- Como lenguaje de programación se ha utilizado Android Studio versión 11, en el cual se soporta la aplicación desarrollada, esta versión proporciona un desarrollo sólido, compatible con características necesarias para soportar la aplicación desarrollada. Adicionalmente, se ha utilizado el IDEA INTELLIJ 2023.1 es una grandiosa herramienta que nos permitirá gestionar de manera eficiente los microservicios utilizados en el proyecto, nos brinda una mejor organización el cual facilita el proceso de desarrollo y mantenimiento de la aplicación.

Un servidor de Hostinger un entorno de desarrollo en la nube que proporciona configuraciones completas de servidores web, usando la base de datos de MySQL para gestión de datos, donde almacenaremos y recuperar datos de manera segura, eficaz en la aplicación desarrollada el cual para obtener las pruebas tuvimos que desarrollarlo en la propia red del mismo internet, teniendo como tablas en la Fig. 4.

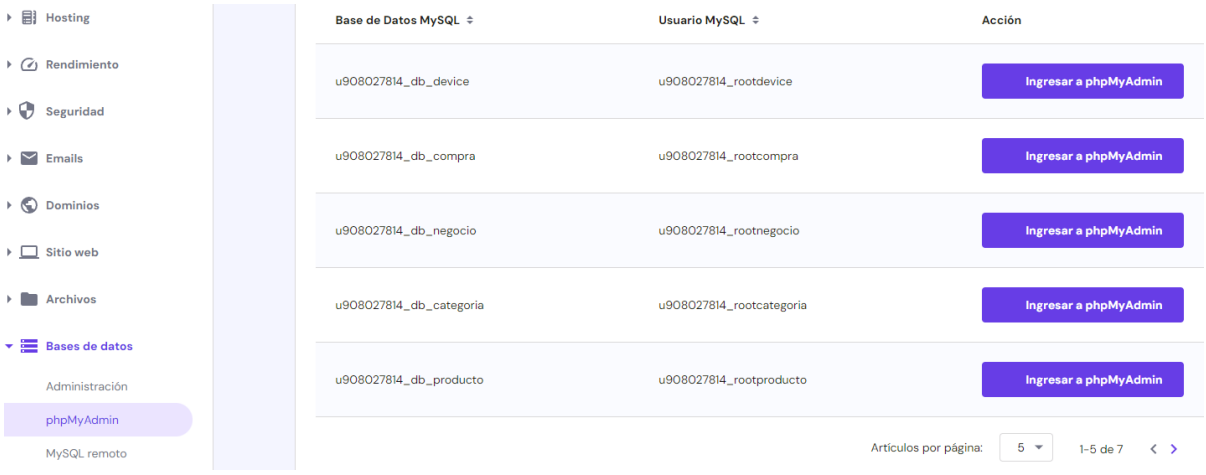


Fig. 5. Bases de Datos Mysql en Hostinger.

Hostinger ofrece servicios de alojamiento web con bases de datos MySQL para microservicios. La lista proporcionada incluye bases de datos específicas para diferentes funciones, cada una con su propio usuario y privilegios asociados. Esto facilita la gestión modular de datos en un entorno de microservicios, donde cada servicio puede tener su propia base de datos independiente, se muestra en la Fig. 5 como se configura una base de datos para un microservicio independiente en un puerto, definiendo el usuario, password y el nombre de la base de datos a crear.

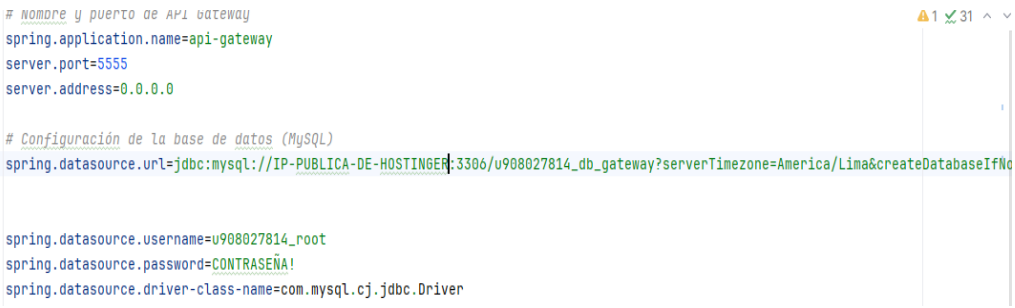


Fig. 6. Configuración de bases de datos en microservicio en cada uno con diferente root y base de datos.

Informe de resultados

Con la implementación de este sistema cada vez que un cliente solicita o registra un pedido el proveedor de este servicio, autorizado para utilizar el sistema, puede consultar el historial de los pedidos del cliente para así ver su estado de compra. Para la elaboración de

este sistema de software, se aplicaron métodos y técnicas en cada una de las etapas del ciclo de desarrollo de la arquitectura de software.

Para evaluar el prototipo en consonancia de la arquitectura de software propuesta, se llevaron a cabo pruebas de los materiales propuestos, centrándose en la medición de estas pruebas, donde se observarán en el contexto de la implementación del servicio en la plataforma “HEROKU”. Lo que comprende tanto la aplicación web como la aplicación móvil, será responsable de gestionar todos los procesos internos acerca de la adquisición de productos de muchos negocios a nivel Nacional. Este proceso implica que los clientes deben registrar y realizar pedidos.

La evaluación también abordará los resultados de la prueba en la población especifica bajo consideración. Antes de abordar los resultados finales, se detallará cada componente, considerando sus indicadores y atributos de calidad predefinidos. La identificación de los servicios correspondientes de las figuras 6 y 7, para llevar a cabo las pruebas respectivas, siendo estos servicios previamente publicados con el propósito de validar su funcionalidad y eficacia en el contexto específico del sistema en desarrollo.

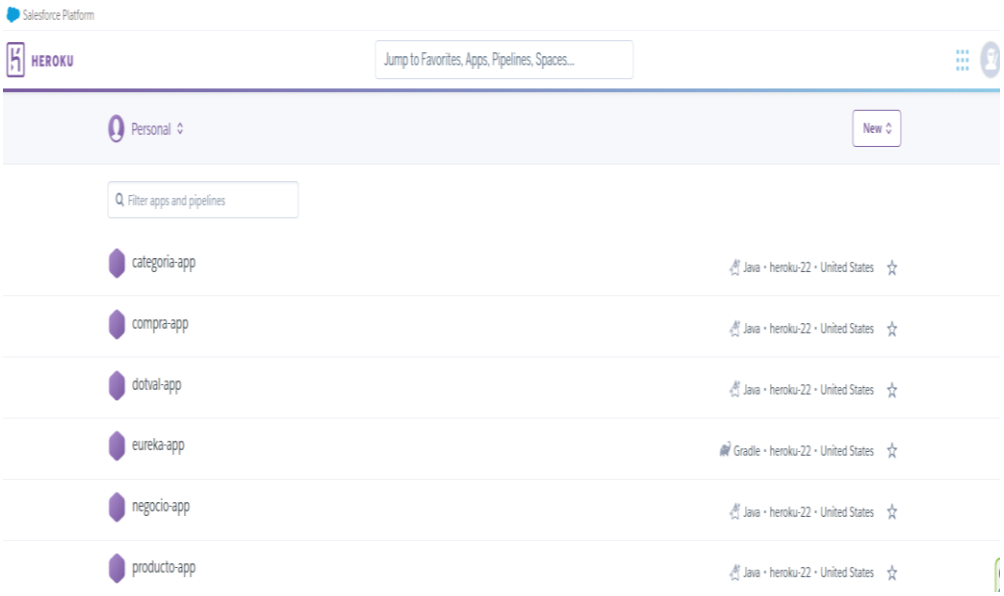


Fig. 7. Servidor de Heroku

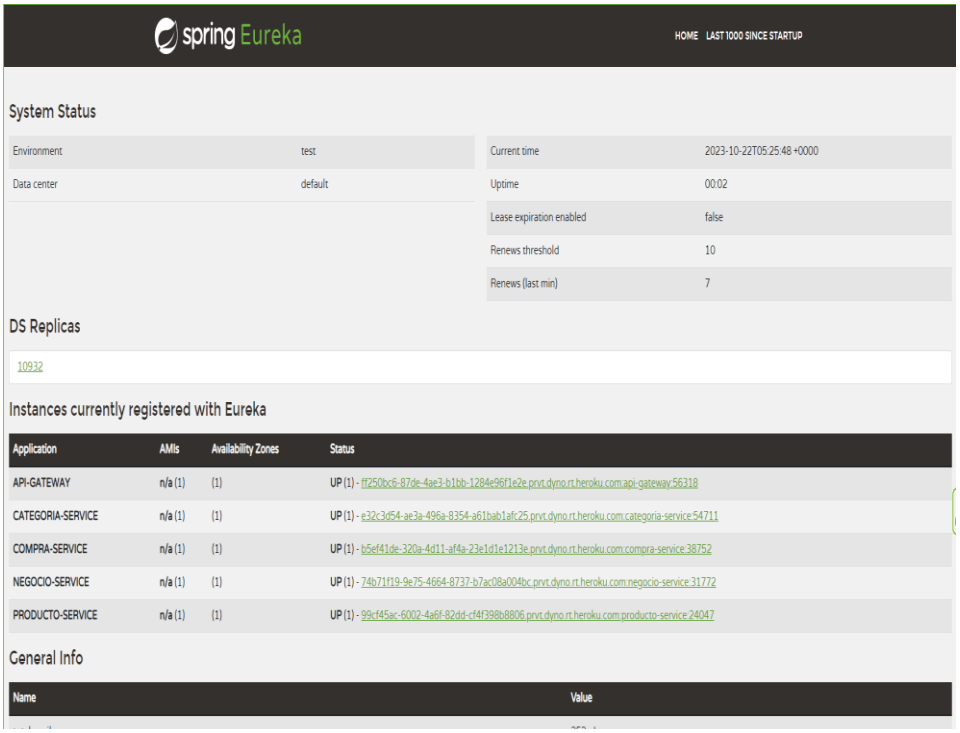


Fig. 8. Servidor de Eureka conteniendo los microservicios

- **Granularidad**

Cuando se aborda la cuestión de la granularidad de un microservicio, se está haciendo referencia a la cantidad de responsabilidades que asume una entidad o servicio específico, asignando una puntuación que varía de 0, denotando responsabilidad mínima, a 5, que representa la máxima responsabilidad. Los resultados obtenidos sobre la granularidad de los microservicios, derivados de la investigación realizada, revelan que el servicio de autenticación de usuarios exhibe un nivel moderado de granularidad. En contraste, tanto el servicio de gestión de inventario como el servicio de procesamiento de pagos presentan un nivel de granularidad más bajo, indicando que son más especializados y se centran en tareas más específicas.

Por otro lado, el servicio de carrito de compras exhibe un nivel de granularidad más alto, sugiriendo que abarca una funcionalidad más extensa y compleja en comparación con los servicios anteriormente mencionados. Estos hallazgos proporcionan una comprensión

detallada de la distribución de responsabilidades entre los distintos microservicios, lo cual es crucial para la evaluación y optimización de la arquitectura de software en cuestión.

El Indicador nos gestionó una puntuación en la Fig. 8 se muestra los módulos de Granularidad:

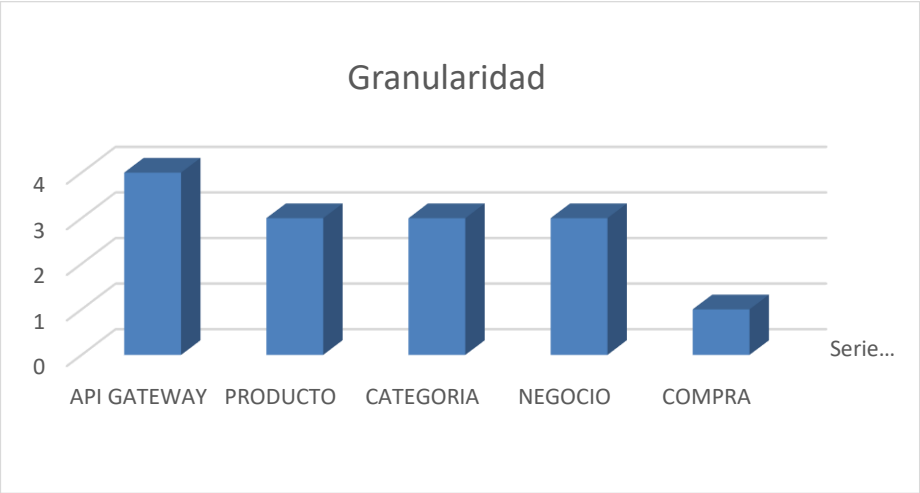


Fig. 9. Granularidad

- **Acoplamiento**

La evaluación del acoplamiento se llevó a cabo teniendo en cuenta la cantidad de puntos de comunicación entre los microservicios [33]. Brown y Miller (2018) destacan la importancia de analizar el acoplamiento en sistemas distribuidos, proporcionando así una base para evaluar el acoplamiento en microservicios. El resultado reveló un acoplamiento total de 5, indicando la existencia de cinco puntos de comunicación, en línea con la metodología propuesta por Brown y Miller [33]. Utilizando la fórmula N(c) + N(p) / N(s) y basándose en las métricas de calidad adaptada de [30], se obtuvo un valor de 0.4. Este resultado sugiere que los microservicios son altamente independientes, ya que sus funciones o consultas se desempeñan de manera independiente. Esta independencia implica que los microservicios, al realizar funciones especializadas, pueden reutilizarse de forma independiente en la implementación de otras aplicaciones. Este enfoque no solo fomenta la eficiencia en el desarrollo, sino que también facilita la adaptabilidad y expansión sostenible

del sistema, en la Fig. 9, se muestra cómo se reutiliza o se pueda acoplar un servicio para otro servicio independiente.

Fig. 10. Adaptabilidad de servicios entre servicios.

- **Cohesión**

La cohesión se evaluó según el cuadro de métricas de calidad que es igual a N(s) / M(s) = 1,25, considerando la cantidad de funcionalidades manejadas por cada microservicio. Robinson (2021) aborda específicamente las métricas de cohesión en arquitecturas de microservicios, proporcionando una base para la evaluación. Los resultados indicaron una cohesión adecuada, ya que cada microservicio se encarga de un número limitado de funcionalidades. Este enfoque contribuye a la eficiencia del sistema y se alinea con las recomendaciones de Robinson [34].

- **Reusabilidad**

La reusabilidad es la capacidad del microservicio para ser reutilizado en diferentes proyectos. Chen et al. (2020) han abordado específicamente la reusabilidad en microservicios, proporcionando orientación para la evaluación [35]. Los resultados indicaron que el microservicio se puede reutilizar en al menos cinco proyectos diferentes [36]. Se aplicó un rango de valoraciones según el Anexo 7 y también mostrándolo en la tabla de métricas de calidad, la metodología de clasificación propuesta por White y Black [37]. Estas tablas permiten comparar sistemas monolíticos y basados en microservicios en términos de reusabilidad, complejidad, cohesión y acoplamiento.

TABLA III

RANGO DE VALORACIONES

SIGNIFICANCIA	CLASIFICACIÓN
0 – 0,25	DEFINITIVAMENTE MEJORÓ
0,26 – 0,50	MEJORÓ BASTANTE
0,51 – 0,75	MEJORÓ
0,76 - 1	NO MEJORÓ

Nota: Rango de Valoraciones para las variables

TABLA IV

RESULTADO PROMEDIO DEL PROTOTIPO DE REUSABILIDAD, COMPLEJIDAD, COHESIÓN Y ACOPLAMIENTO

CRITERIO	SIGNIFICANCIA	CLASIFICACIÓN
REUSABILIDAD	0,16	DEFINITIVAMENTE MEJORÓ
COMPLEJIDAD	0,24	DEFINITIVAMENTE MEJORÓ
COHESIÓN	0,51	MEJORÓ
ACOPLAMIENTO	0,53	MEJORÓ

Nota: Resultado del rango de valoraciones del prototipo para reusabilidad, complejidad, cohesión y acoplamiento.

Los resultados de la investigación indican un impacto positivo significativo derivado de la implementación de cambios en un sistema, específicamente en la transición de un modelo monolítico a uno basado en microservicios. Los criterios clave evaluados, tales como reusabilidad, complejidad, cohesión y acoplamiento, reflejan mejoras sustanciales en sus respectivos rangos de valoración. En términos de reusabilidad y complejidad, los datos revelan mejoras notables, con una clasificación de "Definitivamente Mejoró" para ambas variables. Esto sugiere que la intervención ha logrado no solo incrementar la capacidad de reutilización de componentes, sino también reducir la complejidad del sistema de manera significativa. Adicionalmente, la cohesión y el acoplamiento también experimentaron mejoras

positivas, evidenciando avances en la organización interna del sistema y en la interdependencia entre sus componentes. La clasificación general, que abarca desde "Definitivamente Mejoró" hasta "Mejoró", respalda la idea de que la transición hacia el modelo de microservicios ha tenido un impacto positivo y coherente en múltiples aspectos del sistema. Estos resultados son particularmente relevantes en el contexto actual, donde la eficiencia y la adaptabilidad son cruciales. La adopción de microservicios se presenta como una estrategia efectiva para mejorar la calidad del software, destacando la capacidad de este enfoque para abordar de manera integral aspectos clave del desarrollo de sistemas. Es esencial reconocer la perspectiva crítica y anticipar consideraciones a largo plazo. Inicialmente, es común observar que la adopción de microservicios, en especial en sistemas con pocos datos, exhibe un rendimiento optimizado y una gestión eficaz de recursos. Este escenario inicial, caracterizado por una dependencia más manejable, a menudo conduce a mejoras significativas en aspectos como reusabilidad y complejidad. Sin embargo, es crucial destacar la madurez de la evaluación al considerar la escalabilidad del sistema. Con la acumulación de datos y el crecimiento del sistema, es cierto que algunos desafíos pueden surgir en el entorno de microservicios. A medida que aumenta la complejidad y la interconexión entre servicios, se puede experimentar un impacto en el rendimiento general y la gestión de la escalabilidad.

Johnson et al. (2018) han estudiado el impacto de la transición a microservicios, proporcionando una base para evaluar mejoras en reusabilidad y complejidad [48].

- **Tiempo de respuesta**

  Se llevaron a cabo pruebas de tiempo de respuesta con 20 clientes.

  Sin la aplicación Actual:

Tiempo de espera (cola de clientes):

Cantidad de clientes: 20 p/s

a) Tiempo de espera por cliente: 2 minutos por clientes máximos.

Evaluando si cada cliente tiene un problema de tiempo de espera hasta gestionar su compra y registrarlo en el cajero.

Tiempo de respuesta (cajero físico):

Cantidad de cliente en sistema: 1

a) Tiempo de respuesta 10 segundos: 10000 ms

Con la aplicación actual:

Se ha establecido una prueba de 20 casos en un negocio para medir el tiempo de respuesta de los microservicios según Anexo 2.

Aplicación, se indican los resultados. Del Tiempo de Respuesta para

a) Cantidad de clientes: 2431 ms

- **Tiempo de espera**

Se ha establecido una prueba de 20 peticiones para medir el tiempo de espera de la aplicación, donde la cantidad de usuarios, el tiempo de respuesta y el tiempo de espera de un aplicativo o servicio de microservicios, se observa en la Tabla 6, que, con la aplicación, al contar con 20 usuarios, un tiempo de respuesta promedio de 2431 ms, lo que indica un sistema ágil. El tiempo de espera es de 2500 ms, lo que sugiere una espera moderada para recibir respuestas. Por otro lado, sin la aplicación, los tiempos de respuesta y espera aumentan significativamente a 10000 ms y 2 minutos por cliente aparte de que solo tienen que ir presencialmente, respectivamente, lo que indica problemas de rendimiento.

TABLA V

TIEMPO DE RESPUESTA Y TIEMPO DE ESPERA DEL MINIMARKET CON APLICATIVO Y SIN

APLICATIVO.

	Cantidad Usuarios	Tiempo de Respuesta	Tiempo de Espera	Internet
Con aplicativo	20	2431 ms	2500 ms	Internet de cada Cliente (MiFibra)
Sin aplicativo	20	10000 ms	2 minutos por cliente	Movistar (ADSL)

Nota: Cuadro Comparativo de tiempo de respuesta y tiempo de espero del Minimarket con aplicativo y sin aplicativo

TABLA VI

RESULTADO ÓPTIMO DEL PROTOTIPO DONDE SE INCLUYEN SERVICIOS EXTERNOS TAMBIÉN PARA SU EFICIENCIA.

	Tiempo de respuesta por cliente Total	Tiempo LOADING	Internet	Total
1 cliente	117 ms	2000 ms	Internet propio	2617 ms
5 clientes	614,2 ms	2000 ms	Internet Mi Fibra, Datos	3114,2 ms
10 clientes	1548 ms	2000 ms	Internet Mi Fibra, Datos	4048 ms
15 clientes	1735 ms	2000 ms	Internet Mi Fibra, Datos	4,235 ms
20 clientes	2431 ms	2000 ms	Internet Mi Fibra, Datos	4,431 ms

Nota: Cuadro mostrando el resultado optimo del prototipo.

En el análisis del resultado óptimo del prototipo, es esencial destacar que el tiempo de respuesta se ha medido en relación con un tiempo original de 120 segundos, que representa el escenario inicial sin la implementación de la aplicación actual. Este tiempo original se considera el 100% del tiempo de referencia, sirviendo como base para evaluar las mejoras logradas con la introducción de microservicios y la aplicación móvil. Al observar los datos específicos del tiempo de respuesta por cliente, desde 117 ms para un cliente hasta 4,431

ms para 20 clientes, podemos apreciar una mejora sustancial en la eficiencia del sistema. Este análisis revela que, en comparación con el tiempo original de 120 segundos por cliente, los resultados muestran una reducción significativa del 96.44%, indicando una mejora progresiva a medida que aumenta el número de clientes atendidos. La implementación exitosa de servicios externos ha contribuido a optimizar el rendimiento global, ofreciendo a los usuarios una experiencia más ágil y eficiente.

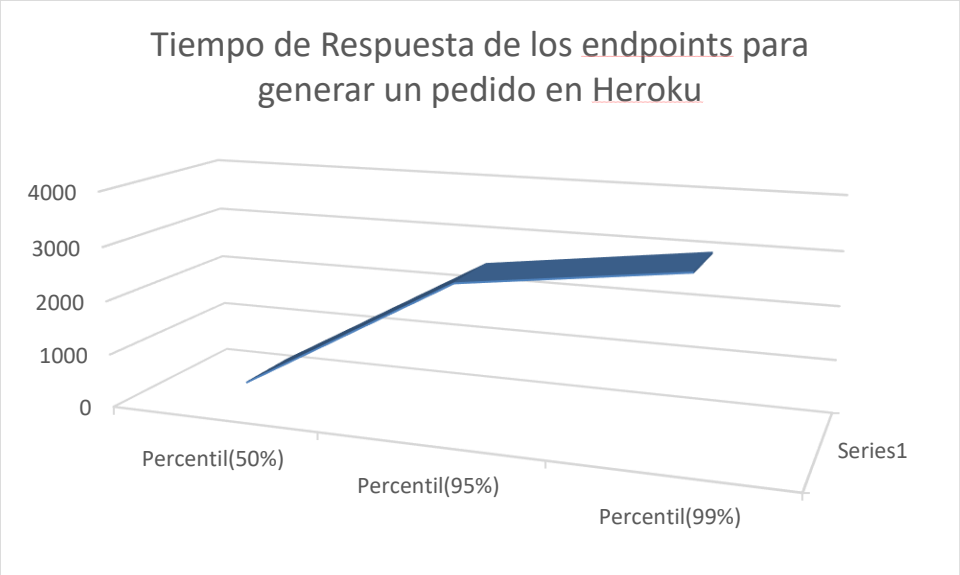


Fig. 11. Tiempo de Respuesta de todos los endpoints de 20 p/s en Heroku

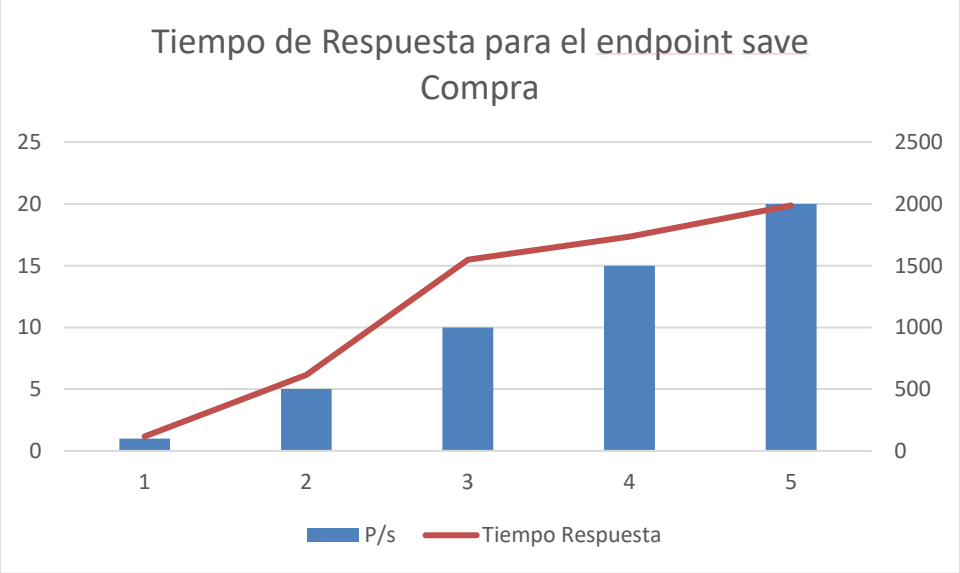


Fig. 12. Tiempo de Respuesta para el endpoint save Compra

En las Fig. 10 se muestra el comportamiento de tiempo de respuesta en la instancia principal de api Gateway y en la Fig. 11 en la instancia del servicio de compra significa que 20 clientes por segundo tendrán un aproximado de respuesta de 3 segundos hasta un máximo de 5 segundos de respuesta y espera del servidor a su pedido en total, estos resultados del servidor se muestran en el anexo 2, y que el mismo promedio será para todos los usuarios de todos los negocios que existan alrededor del servidor para el microservicio de negocio.

Los resultados de tiempo de espera, respuesta, rendimiento, escalabilidad, granularidad, reusabilidad, acoplamiento, etc., se compararon según la norma ISO 25010 y la metodología de evaluación [50], resaltando mejoras sustanciales en la eficiencia del sistema.

- **Rendimiento**

Según el estándar RFC 2616.Los códigos de estado HTTP devueltos por las API cada vez que se realice una petición serán los ofrecidos por el estándar RFC 2616 que se detalle en la tabla VII [30][49][50].

TABLA VII
RESULTADO ÓPTIMO DEL PROTOTIPO DONDE SE INCLUYEN SERVICIOS EXTERNOS TAMBIÉN PARA SU EFICIENCIA.

Codigo	**Descripción**
1XX	Respuestas Informativas
2XX	Peticiones correctas
3XX	Redirecciones
4XX	Errores del cliente
5XX	Errores del servidor

Nota: Norma estándar RFC 2616

Las pruebas de las métricas de rendimiento de ejecución de los clientes app y web

según la arquitectura presentada según Anexo 2 Figura de Rendimiento, específicamente relacionadas con las respuestas de códigos de estado HTTP 2XX (éxito) y 4XX (errores del cliente) durante un período de tiempo determinado y con el total de peticiones por segundo. Aquí hay una interpretación: 2XX: Indica respuestas exitosas con códigos de estado HTTP en la gama de 200 a 299. < 1 rps: Menos de 1 solicitud por segundo. Esto significa que la tasa de respuestas exitosas es menor a una solicitud por segundo. 76 reqs: Indica que hubo un total de 76 solicitudes exitosas (códigos de estado 2XX) durante el período de tiempo especificado. 4XX: Indica respuestas con errores del cliente con códigos de estado HTTP en la gama de 400 a 499. < 1 rps: Menos de 1 solicitud por segundo. Esto significa que la tasa de respuestas con errores del cliente es menor a una solicitud por segundo. 14 reqs: Indica que hubo un total de 14 solicitudes con errores del cliente (códigos de estado 4XX) durante el período de tiempo especificado.

- **Seguridad**

Según lo detallado en el Anexo 5, se llevaron a cabo pruebas utilizando las direcciones IP de los servidores exclusivos de Heroku, con el fin de evaluar su nivel de protección. Estas pruebas fueron ejecutadas para asegurar que la arquitectura de software satisfaga de manera efectiva todos los requisitos establecidos en términos de seguridad.

Los resultados obtenidos corroboran que los microservicios implementados están adecuadamente resguardados, fortaleciendo así la confianza en la integridad y seguridad del sistema. En consonancia con las investigaciones previas, como la de Arcila Diaz [30] que implementa la seguridad mediante certificados y tokens, esta investigación ha adoptado el Token como mecanismo de autenticación y autorización. La evaluación de la arquitectura diseñada confirma en la Fig. 12 que cumple con los requisitos de autenticación y autorización identificados.

```java
@Override
public String generateToken(UserPrincipal auth)
{
    String authorities = auth.getAuthorities().stream() Stream<capture of extends GrantedAuthority>
            .map(GrantedAuthority::getAuthority) Stream<String>
            .collect(Collectors.joining( delimiter: ","));

    Key key = Keys.hmacShaKeyFor(JWT_SECRET.getBytes(StandardCharsets.UTF_8));

    return Jwts.builder()
            .setSubject(auth.getUsername())
            .claim( s: "roles", authorities)
            .claim( s: "userId", auth.getId())
            .setExpiration(new Date(System.currentTimeMillis() + JWT_EXPIRATION_IN_MS))
            .signWith(key, SignatureAlgorithm.HS512)
            .compact();
}

2 usages   Dagner Chuman
@Override
public String generateToken(User user)
{
    Key key = Keys.hmacShaKeyFor(JWT_SECRET.getBytes(StandardCharsets.UTF_8));

    return Jwts.builder()
            .setSubject(user.getUsername())
            .claim( s: "roles", user.getRole())
            .claim( s: "userId", user.getId())
            .setExpiration(new Date(System.currentTimeMillis() + JWT_EXPIRATION_IN_MS))
            .signWith(key, SignatureAlgorithm.HS512)
            .compact();
}
```

Fig. 13. Implementación de JSON Web Token.

Donde también se confirma la Fig. 12, que se implementa un código que proporciona una manera de configurar un cliente OkHttp para incluir automáticamente un token de autorización en las solicitudes HTTP realizadas por la aplicación móvil para aplicaciones Android. El token se establece mediante el método setAuthToken antes de realizar las solicitudes y se añade a las solicitudes mediante un interceptor dentro del cliente OkHttpClient. Este patrón es útil cuando se necesita incluir el mismo token en varias solicitudes a diferentes puntos finales de la API.

```
7 usages   Dagner Chuman
private static OkHttpClient createOkHttpClient(Context context) {
    OkHttpClient.Builder httpClient = new OkHttpClient.Builder();
    if (!authToken.isEmpty()) {
        httpClient.addInterceptor(chain -> {
            Request original = chain.request();
            Request.Builder requestBuilder = original.newBuilder()
                    .header( name: "Authorization",  value: "Bearer " + authToken);
            Request request = requestBuilder.build();
            return chain.proceed(request);
        });
    }
    return httpClient.build();
}
```

Fig. 14. Cliente OkHttpClient que incluye un interceptor para agregar un token de autorización tipo "Bearer" a las solicitudes HTTP.

Fig. 15. Resultado del token generado para Spring Security.

- **Escalabilidad**

Para validar la solidez de la arquitectura de software en nuestro e-commerce, compatible tanto con dispositivos móviles Android como con la web mediante Angular, hemos realizado pruebas específicas para evaluar la escalabilidad. Los resultados detallados de estas pruebas se encuentran en el anexo 3 de nuestro informe, donde se indican las peticiones recibidas por el API Gateway y el servicio de compra, crucial para finalizar el proceso de adquisición de productos por parte del cliente. El requisito fundamental para este atributo es que nuestra arquitectura pueda aumentar su capacidad de procesamiento en respuesta al crecimiento de la demanda por parte de los usuarios. Para lograr esto, hemos configurado la aplicación en Heroku para permitir el escalado horizontal cuando la demanda de nuestro servicio aumenta. En términos más sencillos, si una instancia del servicio de compra no puede hacer frente a la demanda actual, se generará automáticamente otra instancia para gestionar y responder a las peticiones adicionales. En caso de identificarse la necesidad de un escalado con un impacto aún más significativo, será necesario implementar primero el servicio sobre una aplicación en Heroku con características mejoradas. En nuestra

investigación, hemos trabajado con aplicaciones independientes generadas mediante Spring Boot, resultando en la creación de un archivo .JAR que puede cargarse en GitHub y ejecutarse en Heroku. Las especificaciones detalladas de este proceso se encuentran en la sección de equipos, específicamente en el anexo 4 dedicado a la aplicación en Heroku para producción. Todas las pruebas de escalabilidad se llevaron a cabo utilizando Heroku, y los resultados se presentan en las Figuras 10 y 11. Estos resultados confirman la eficacia de la aplicación en Heroku para abordar las necesidades actuales de nuestro e-commerce.

- **Mantenibilidad**

La adopción de una estructura conceptual basada en Microservicios presenta ventajas significativas, consolidando un acoplamiento más tenue entre los distintos componentes del sistema. Esta arquitectura fomenta la autonomía de cada microservicio, permitiendo su despliegue independiente en la plataforma en la nube Heroku. Esta elección tecnológica no solo simplifica las actualizaciones periódicas, sino que también minimiza el impacto en la integridad del sistema en su conjunto, lo que se traduce en ciclos de entrega más eficientes y menos propensos a interrupciones.

Un caso ejemplar que respalda la efectividad de este enfoque se encuentra en la investigación de Arcila Díaz [30], que se centró en diseñar una arquitectura específica para garantizar la disponibilidad de Historias Clínicas Electrónicas.

En este contexto, se implementó la filosofía de microservicios, lo que brindó la capacidad de integrar información proveniente de diversos negocios en un repositorio unificado, abarcando productos de distintas áreas comerciales. Esta integración facilita a los clientes la visualización centralizada de la disponibilidad de productos en cada negocio, mejorando la experiencia del usuario al proporcionar información de manera más eficiente y unificada. La reusabilidad emerge como un indicador esencial de la mantenibilidad en cualquier arquitectura, y la estructura delineada en este estudio destaca con un impresionante índice de reusabilidad de 5. Esta métrica subraya la capacidad de los microservicios para desempeñar funciones especializadas de manera autónoma, lo que, a su vez, posibilita su reutilización para la implementación de otras aplicaciones. La adopción de la arquitectura de

microservicios no solo optimiza la operatividad del sistema actual, sino que también sienta las bases para futuras expansiones y desarrollos.

En la Fig. 15 se ve como se da mantenibilidad de un microservicio después de evaluar las herramientas de arquitectura de software se sube Dev al GitHub sin errores los códigos nuevos de mantenibilidad del sistema y se evalúa “Ops” nos dan la observación para que no surja ningún falló con la mantenibilidad en producción.

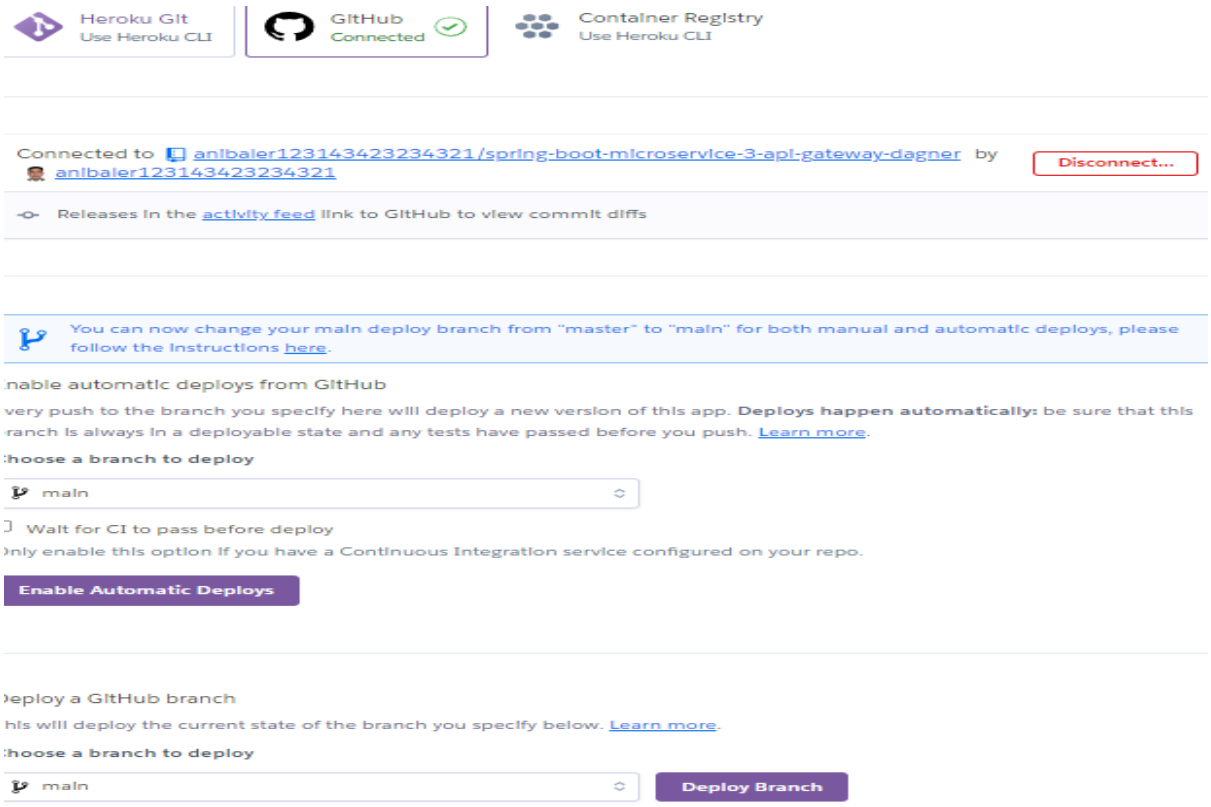


Fig. 16. Mantenibilidad de un microservicio.

- **Adaptabilidad**

En la Fig. 9 se demuestra gráficamente la adaptabilidad de los servicios entre sí. El acoplamiento total de 5 servicios (negocio, producto, compra, categoría, usuarios), derivado de la evaluación conforme a la norma ISO 25010, refleja la cantidad de puntos de comunicación entre los microservicios. Esta medición respalda la idea de que, con tan solo cinco puntos de conexión, los microservicios mantienen un nivel de acoplamiento controlado y la adaptabilidad entre los microservicios.

3.2. Discusión

En cuanto a los resultados de nuestra investigación, se observó que la implementación de una aplicación móvil basada en microservicios en un minimarket puede tener un impacto significativo en la eficiencia del proceso de ventas. Se encontró una reducción notable en el tiempo de respuesta y el tiempo de espera de los clientes, lo cual indica una mejora en la experiencia de compra. Estos hallazgos son consistentes con los resultados de estudios. Por ejemplo, Want et al. [2] descubrieron que la adopción de aplicaciones móviles de compras puede canibalizar los canales físicos y en línea existentes, generando un aumento en los gastos totales de los hogares. Nuestros resultados respaldan la idea de que adoptar este tipo de soluciones tecnológicas puede generar ganancias para los negocios [3].

Al comparar nuestros hallazgos con las investigaciones previas de Arcila Díaz [30]. La implementación de microservicios permite la configuración horizontal automatizada, donde cada microservicio puede escalar de manera independiente para ajustarse a las fluctuaciones de la demanda. Este enfoque, respaldado por Arcila Díaz [30], coincide con la selección de otros investigadores. Las pruebas de escalabilidad realizadas en la arquitectura implementada corroboraron su capacidad para generar nuevas instancias de microservicios en respuesta a un aumento de la demanda. Es relevante destacar que, en una sola prueba, se logró un soporte de 73 peticiones por segundo, lo cual representa un rendimiento excepcional. Esta capacidad se traduce en una alta disponibilidad y un cumplimiento eficiente de los indicadores de tiempo establecidos para responder a las solicitudes, validando así la robustez y eficacia de la arquitectura en términos de escalabilidad, donde la cohesión también mejoró de un 1,33 a 1,25, la reusabilidad de 5 microservicios y un acoplamiento de 0.4.

Además de los resultados propios, es importante considerar las implicaciones teóricas y prácticas de nuestra investigación. Teóricamente, nuestros hallazgos respaldan la idea de que la adopción de aplicaciones móviles en el entorno minorista puede ser una estrategia efectiva para mejorar la eficiencia operativa y aumentar las ganancias. Esto amplía la comprensión del impacto de la tecnología en el comportamiento de compra de los consumidores y proporciona una base teórica para futuras investigaciones en este campo.

Desde una perspectiva práctica, nuestros resultados sugieren que los minoristas pueden beneficiarse de la implementación de aplicaciones móviles basadas en microservicios para agilizar el proceso de ventas y reducir las colas de espera. Estas aplicaciones pueden proporcionar a los clientes una experiencia de compra más conveniente y eficiente, lo que puede aumentar su satisfacción y lealtad hacia la empresa. Además, al reducir los tiempos de espera, se pueden mejorar los flujos de clientes

No obstante, es importante destacar algunas limitaciones de nuestra investigación. Una limitación clave es el tamaño de la muestra, que se basó en una muestra conveniente de 20 consumidores. Además, se llevó a cabo en un solo minimarket, lo que puede limitar la generalización de los resultados a otros contextos. Sin embargo, estas limitaciones han sido abordadas en estudios similares, como el de [4], donde también se utilizó un método de muestreo no probabilístico por conveniencia. A pesar de estas limitaciones, nuestros hallazgos brindan una base sólida para futuras investigaciones en este campo.

En cuanto a nuevas posibilidades de investigación, sería interesante explorar más a fondo los factores que influyen en la adopción de aplicaciones móviles de compras por parte de los consumidores. Además, se podrían evaluar los efectos a largo plazo de la implementación de estas aplicaciones en la lealtad del cliente y las ganancias de los negocios. Asimismo, sería útil investigar cómo la personalización y la integración de funciones adicionales en las aplicaciones móviles pueden influir en la experiencia del cliente y los resultados comerciales.

3.3. Aporte de la investigación

En cuanto al objetivo específico 1, se obtuvo por juicio situacional de los

requerimientos principales a detallar para el aplicativo.

3.3.1. Análisis de Requerimientos Funcionales y No Funcionales:

En el inicio del proceso de análisis de requerimientos, la elección de la metodología adecuada desempeña un papel crucial en la consecución del éxito del proyecto de desarrollo de una aplicación móvil basada en microservicios. La justificación de esta elección radica en la necesidad de establecer desde el principio una estructura sólida y coherente que guíe la identificación, documentación y gestión de los requerimientos del sistema. En el contexto de un entorno basado en microservicios, donde la modularidad y la escalabilidad son elementos clave, la elección de la metodología correcta establece las bases para un desarrollo eficiente y una implementación exitosa de la arquitectura de microservicios. Para el desarrollo del software, se exploraron cuatro enfoques de metodologías destacados, y posteriormente se realizó una elección. Estos son:

1. **Metodología de modelado de Sistemas en Ingeniería de Requerimientos:**
    - Enfoque: Se centra en la creación de modelos para representar los requerimientos del sistema mediante diagramas y representaciones gráficas.
    - Ventajas: Proporciona una visualización clara de los requerimientos y facilita la comunicación.
    - Desafíos: Puede requerir habilidades técnicas; la gestión de cambios puede ser complicada.
    - Pasos:
        i. Identificación de Requerimientos: Recopilación a través de interacciones con las partes interesadas.
        ii. Análisis y Descomposición: Descomposición en modelos utilizando diagramas.
        iii. Validación: Revisión y validación de modelos con las partes

interesadas.

2. **Metodología de Desarrollo de Software Ágil:**

   - Enfoque: Metodologías ágiles, como Scrum, Xp y Kanban, se centran en la flexibilidad y la colaboración continua. Ventajas: Adaptable a cambios; fomenta la colaboración y entrega rápida, autores. se centra en la colaboración y la adaptabilidad continua. Este enfoque iterativo permite una gestión eficiente de requisitos en entornos dinámicos, facilitando la entrega incremental y la capacidad de respuesta a cambios a lo largo del ciclo de desarrollo del software [44][45].
   - Desafíos: Menor documentación formal; requiere comunicación efectiva.
   - Pasos:
     i. Planificación de Iteraciones: Priorización y planificación de requerimientos.
     ii. Desarrollo Iterativo: Trabajo en incrementos pequeños con entregas al final de cada iteración.
     iii. Retroalimentación Continua: Recopilación de retroalimentación para ajustar requerimientos.

3. **Metodología de Ingeniería de Requerimientos Basada en Objetivos:**

   - Enfoque: Se centra en establecer objetivos y restricciones del sistema.
   - Ventajas: Enfoque en objetivos; alineación con objetivos organizacionales. Desafíos: Puede ser complejo definir y gestionar objetivos.
   - Pasos:

i. Identificación de Objetivos: Definición de objetivos del sistema. Análisis de Objetivos:

ii. Análisis para identificar requerimientos y restricciones.

iii. Validación de Objetivos: Validación de que los requerimientos satisfacen los objetivos.

4. Metodología de Método de Casos de Uso (UCM):

- Enfoque: UCM se centra en describir la interacción de los usuarios con el sistema. Su enfoque estructurado, centrado en casos de uso, proporciona una vía clara para la identificación y descripción precisa de requisitos funcionales, contribuyendo a una comprensión más profunda de las necesidades del usuario [48].
- Ventajas: Enfoque centrado en el usuario; facilita la identificación y comprensión de requerimientos. Desafíos: Puede requerir experiencia en modelado; gestión de requisitos complejos.
- Pasos:

i. Identificación de Actores: Identificación de usuarios o sistemas externos.

ii. Identificación de Casos de Uso: Identificación de escenarios de interacción.

iii. Descripción de Casos de Uso: Detallado de pasos e interacciones en cada caso.

5. Calidad de Producto según ISO/IEC 25010:

- Enfoque: Utiliza la normativa ISO/IEC 25010, para definir requerimientos no funcionales relacionados con la calidad del producto [52].

- Características Clave: Seguridad, rendimiento, mantenibilidad, Adaptabilidad, escalabilidad.

Elección de la Metodología de Desarrollo de Software:

TABLA VIII

Metodología	Metodología XP	Metodologías de Ingeniería de Requerimientos	Metodología de Casos de Uso	Metodología de Objetivos
Enfoque	Proceso menos controlado, con pocos principios	Proceso mucho más controlado	Enfoque centrado en el usuario	Se centra en establecer objetivos y restricciones del sistema
Énfasis en Arquitectura del Software	Menos énfasis	Más énfasis	No aplicable	No aplicable
Programación en par	Sí	No	No	No
Roles adicionales	Más roles	Menos roles	No aplicable	No aplicable
Control del proceso	Menos controlado	Mucho más controlado	No aplicable	No aplicable
Características adicionales	- Enfoque ágil centrado en simplicidad, comunicación y retroalimentación. - Reutilización del código desarrollado.	- Mayor control y estructura en el proceso.	- Enfoque centrado en el usuario y casos de uso.	- Centrado en objetivos y restricciones del sistema.

Calidad del Producto (ISO/IEC 25010)	- Seguridad - Rendimiento – Escalabilidad - Usabilidad - Compatibilidad - Mantenibilidad -	- Dependiente de la implementación específica de cada metodología.	- Enfoque en la identificación y comprensión de requisitos funcionales.	- Define requerimientos no funcionales relacionados con la calidad del producto
Total	**6**	**1**	**3**	**2**

1. Enfoque:
   - Metodología XP: Se le otorgó un puntaje de 1 porque se caracteriza por un enfoque ágil y menos controlado, centrándose en la simplicidad, la comunicación y la retroalimentación.
   - Metodologías de Ingeniería de Requerimientos: Puntaje de 0 porque tiene un proceso mucho más controlado.
   - Metodología de Casos de Uso: Puntaje de 1 porque tiene un enfoque centrado en el usuario.
   - Metodología de Objetivos: Puntaje de 1 porque se centra en establecer objetivos y restricciones del sistema.
2. Énfasis en Arquitectura del Software:
   - Metodología XP: Puntaje de 1 porque tiene menos énfasis en la arquitectura del software.
   - Metodologías de Ingeniería de Requerimientos: Puntaje de 0 porque tiene más énfasis en la arquitectura del software.
   - Metodología de Casos de Uso: Puntaje de 0 porque no es aplicable.
   - Metodología de Objetivos: Puntaje de 0 porque no es aplicable.
3. Programación en Par:
   - Metodología XP: Puntaje de 1 porque implica programación en par.
   - Metodologías de Ingeniería de Requerimientos: Puntaje de 0 porque no implica programación en par.

- Metodología de Casos de Uso: Puntaje de 0 porque no implica programación en par.
- Metodología de Objetivos: Puntaje de 0 porque no implica programación en par.

4. Roles Adicionales:
   - Metodología XP: Puntaje de 1 porque implica más roles.
   - Metodologías de Ingeniería de Requerimientos: Puntaje de 0 porque implica menos roles.
   - Metodología de Casos de Uso: Puntaje de 0 porque no es aplicable.
   - Metodología de Objetivos: Puntaje de 0 porque no es aplicable.
5. Control del Proceso:
   - Metodología XP: Puntaje de 0 porque implica menos control en el proceso.
   - Metodologías de Ingeniería de Requerimientos: Puntaje de 1 porque implica mucho más control en el proceso.
   - Metodología de Casos de Uso: Puntaje de 0 porque no es aplicable.
   - Metodología de Objetivos: Puntaje de 0 porque no es aplicable.
6. Características Adicionales:
   - Metodología XP: Puntaje de 1 porque incluye enfoque ágil centrado en simplicidad, comunicación y retroalimentación, así como la reutilización del código desarrollado.
   - Metodologías de Ingeniería de Requerimientos: Puntaje de 0 porque implica mayor control y estructura en el proceso.
   - Metodología de Casos de Uso: Puntaje de 1 porque tiene un enfoque centrado en el usuario y casos de uso.
   - Metodología de Objetivos: Puntaje de 1 porque se centra en objetivos y restricciones del sistema.
7. Calidad del Producto (ISO/IEC 25010):

- Metodología XP: Puntaje de 1 porque aborda aspectos de seguridad, rendimiento, usabilidad, escalabilidad y mantenibilidad.
- Metodologías de Ingeniería de Requerimientos: Puntaje de 0 porque depende de la implementación específica de cada metodología.
- Metodología de Casos de Uso: Puntaje de 1 porque se centra en la identificación y comprensión de requisitos funcionales.
- Metodología de Objetivos: Puntaje de 0 porque define requisitos no funcionales relacionados con la calidad del producto.

8. Total:
   - Metodología XP: Total de 6.
   - Metodologías de Ingeniería de Requerimientos: Total de 1.
   - Metodología de Casos de Uso: Total de 3.
   - Metodología de Objetivos: Total de 2.

La elección de la Metodología XP se justifica por su enfoque ágil y los puntajes más altos en los criterios establecidos. Esta metodología se considera la más adecuada para el desarrollo del proyecto de aplicación móvil basada en microservicios debido a su adaptabilidad y enfoque en la entrega continua de valor.

Como responsable del desarrollo del proyecto de aplicación móvil basada en microservicios, la elección de la metodología XP se justifica por su enfoque ágil, que fomenta la colaboración, la adaptabilidad y la entrega continua de valor. La simplicidad, la comunicación efectiva y la reutilización del código desarrollado son aspectos clave que respaldan la decisión de adoptar XP para garantizar el éxito del proyecto. En la evaluación numérica, la Metodología XP obtuvo el puntaje más alto con un total de 6, seguida por las Metodologías de Ingeniería de Requerimientos con un total de 1. La Metodología de Casos de Uso obtuvo un total de 3, y la Metodología de Objetivos obtuvo un total de 2. La elección

basada en esta evaluación favorece la Metodología XP debido a su mayor puntaje en los criterios establecidos.

TABLA IX

MATRIZ DE CARACTERISTICAS DE XP

Características de XP	**Descripción**
Desarrollo Iterativo e Incremental	Ciclos cortos para entregas incrementales.
Comunicación Constante	Énfasis en la comunicación continua.
Simplicidad	Prioridad en mantener el código simple.
Retroalimentación Rápida	Mecanismos para obtener retroalimentación.
Pruebas Unitarias Continuas	Escritura y ejecución constante de pruebas.
Integración Continua	Integración regular para detectar problemas.
Cliente en el Centro del Desarrollo	Involucramiento activo del cliente.
Equipos Pequeños	Preferencia por equipos pequeños y eficientes.

Nota: Metodologia XP.

Se presentan los Requerimientos No Funcionales en la Tabla X, los cuales están configurados según la terminología y estándares de la normativa ISO/IEC 25010. Estos requisitos establecen las directrices para garantizar la eficiencia y calidad de la aplicación móvil basada en microservicios en términos de rendimiento, seguridad, escalabilidad, mantenibilidad y adaptabilidad. La elección de la normativa ISO/IEC 25010 como marco de referencia refleja el compromiso de asegurar que la aplicación cumpla con estándares internacionales reconocidos, proporcionando así una base robusta para la evaluación y medición objetiva de su desempeño. Cada requerimiento no funcional se ha definido cuidadosamente para alinearse con los principios y criterios de la normativa, garantizando así la conformidad y calidad del sistema en aspectos cruciales para la experiencia del usuario y la viabilidad operativa en el entorno de Minimarket Job.

TABLA X

CALIDAD DE PRODUCTO DE SOFTWARE ISO 25010

No.	Requerimiento	Descripción
1	Desempeño	La aplicación móvil basado en microservicios debe asegurar un rendimiento optimo para asegurar la disponibilidad de los productos para que un cliente pueda realizar la obtención y realización de la compra en un determinado momento.
2	Seguridad	Para el acceso a los datos de los clientes de todos los negocios se debe garantizar la seguridad de los datos sensibles mediante mecanismos de cifrado y autenticación.
3	Escalabilidad	La arquitectura basada en microservicios debe ser escalable para muchos negocios, que será el mismo en cada uno y así manejar un alto volumen de transacciones.
4	Mantenibilidad	La arquitectura subida a la nube en Heroku debe soportar y ser fácil de mantener y actualizar, con una arquitectura modular para agregar o modificar funcionalidades.
5	Adaptabilidad	La aplicación móvil debe ser compatible con el sistema operativo Android.

Nota: Requerimientos no Funcionales, características de calidad según ISO/IEC 25010. Fuente (ISO 25000, s.f.).

Las interacciones entre los usuarios y el sistema son detalladas mediante la técnica de casos de uso después de haber asentar los requisitos funcionales y no funcionales según la ISO 25010. Con la implementación de estos requerimientos se tiene una guía de cómo nos ayudaremos o basaremos en la elaboración de este sistema de software se ha aplicado métodos y técnicas en cada una de las etapas del ciclo de desarrollo de la arquitectura de software, estas etapas se muestran en la figura X.

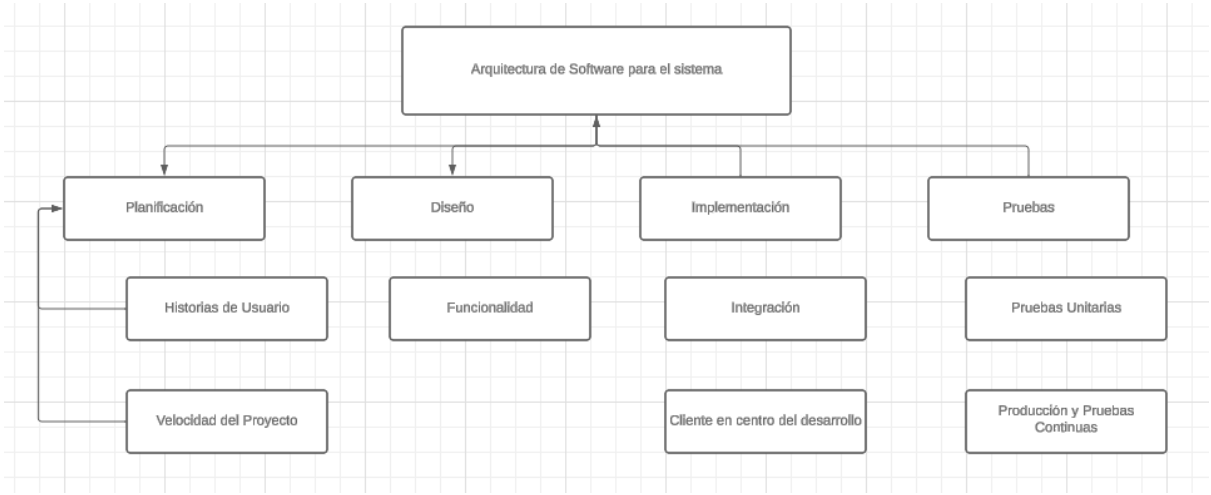


Fig. 17. Etapas del desarrollo de la Arquitectura de Software.

3.3.2. Herramientas de arquitectura de software

Antes de explorar las herramientas fundamentales para la arquitectura de software, llevamos a cabo un exhaustivo proceso de revisión de metodologías de desarrollo de software, nos basamos en la recomendación de autores influyentes Arcila Diaz [30], quienes destacaron la importancia de adoptar enfoques ágiles en entornos de desarrollo dinámicos.

En el cual primero seleccionamos los componentes de la arquitectura de microservicios, donde se tomaron decisiones fundamentales para lograr una implementación con herramientas robustas y eficiente para la arquitectura de software. A continuación, se detalla una matriz con la selección de componentes para poder Implementar la arquitectura a posterior:

TABLA XI
SELECCIÓN DE COMPONENTES DEL SERVIDOR WEB

Componente	**Enfoque/Tecnología**	**Razón de Elección**
Creación de Microservicios con Spring Boot	Spring Boot	Simplificación del desarrollo de microservicios, amplio soporte en la comunidad, eficiencia en Java.
Implementación de Seguridad con Spring Security	Spring Security	Funciones sólidas de autenticación y autorización, capa de seguridad robusta.
Comunicación entre Microservicios con Eureka	Netflix Eureka Server	Gestión centralizada de servicios, mejora de escalabilidad y redundancia.
Implementación de Proyectos con Spring Boot y MySQL	Spring Boot, MySQL	Facilidad de integración con bases de datos, elección de MySQL según requisitos específicos.
Mantenimiento y Consulta de Entidades con JPA	Java Persistence API (JPA)	Gestión eficiente de entidades y base de datos relacionales, capa de persistencia robusta.
Diseño de API Gateway y Redireccionamiento con Feign	API Gateway, Feign	Gestión de solicitudes del cliente, redirigir solicitudes a microservicios, mejora de modularidad.

Implementación de IoC con Spring	Spring	Inversión de Control para gestión eficiente de dependencias entre componentes.
Autenticación y Registro de Usuarios	Varios (Spring, etc.)	Implementación segura de autenticación y registro de usuarios, comunicación asíncrona eficiente.

Nota: Selección de componentes.

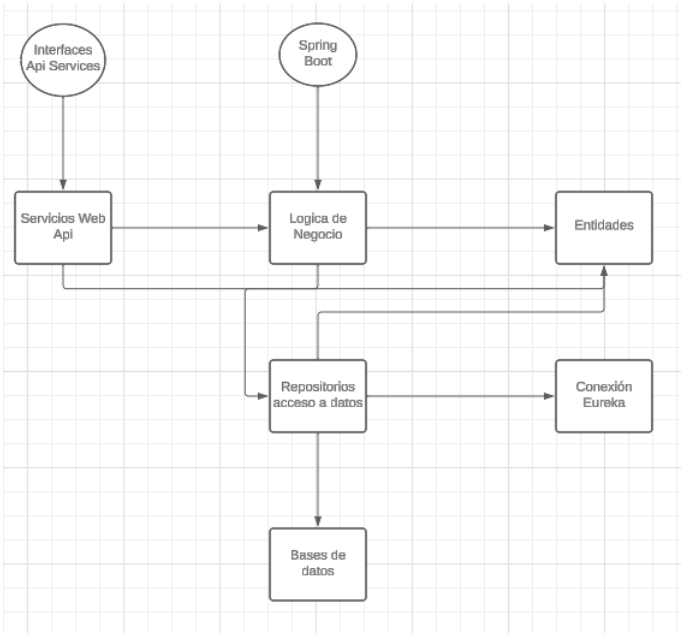


Fig. 18. Diagrama de componentes del servidor Web.

Se ha desarrollado siguiendo un diagrama del servidor de la arquitectura seleccionada, detallados cada uno de los componentes anteriormente que conforman la arquitectura de software propuesta, en la Figura 18 se presenta la arquitectura de implementación del servidor basado en microservicios, basándose de la Tabla XII abarcando tanto el Cliente como el Servicio.

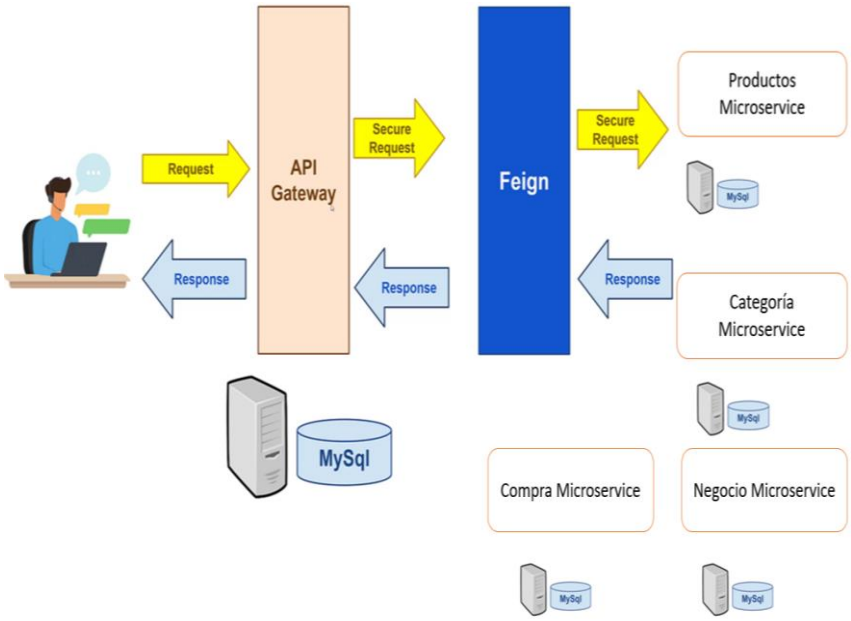


Fig. 19. Arquitectura del modelo de composición de microservicios

Se ha desarrollado un diseño de la arquitectura del sistema en general de la Fig. 19 incluyendo el diseño de la arquitectura del sistema en la nube, detallados cada uno de los componentes anteriormente.

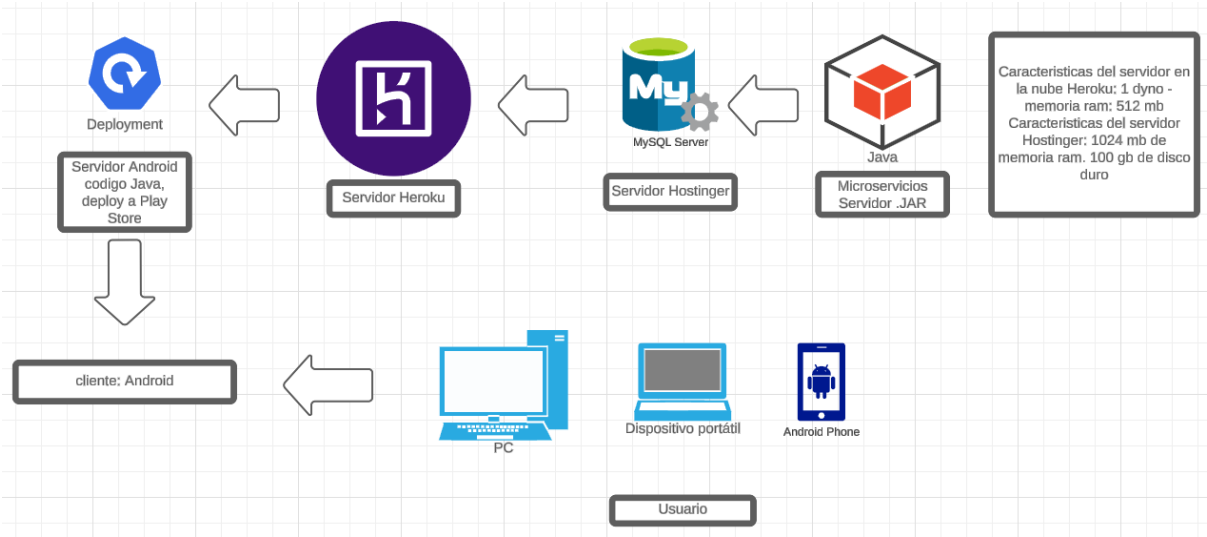


Fig. 20. Implementación de la arquitectura del sistema.

Descripción de Componentes para Implementación de la Arquitectura del Sistema:

- Java .JAR: Archivo ejecutable de Java que contiene la lógica y funcionalidad de los microservicios desarrollados con Spring Boot. Se utiliza para desplegar y ejecutar los servicios en entornos compatibles con Java.
- Servidor Hostinger: Proveedor de servicios de alojamiento web que hospeda componentes web necesarios para la aplicación, como bases de datos MySQL. Ofrece soporte para la implementación y gestión eficiente de la arquitectura de microservicios.
- Servidor Heroku: Plataforma en la nube que permite implementar, gestionar y escalar aplicaciones. Utilizado para alojar y ejecutar servicios, proporcionando escalabilidad y redundancia para los microservicios.
- Servidor Android Play Store: Descripción: La tienda oficial de aplicaciones para dispositivos Android. Es el canal de distribución para la aplicación móvil, permitiendo a los usuarios descargar e instalar la aplicación en dispositivos Android.
- Android Studio 11: Entorno de desarrollo integrado (IDE) oficial para el desarrollo de aplicaciones Android. Proporciona herramientas avanzadas para el desarrollo eficiente de la parte móvil de la aplicación.
- Usuario (Tecnologías como PC, Laptop, Android Phone): Representa a los usuarios finales que interactúan con la aplicación desde diferentes dispositivos, como computadoras personales, laptops y teléfonos Android. Incluye diversas tecnologías utilizadas por los usuarios para acceder a la aplicación.

Fig. 21. Componentes de la Arquitectura del Sistema.

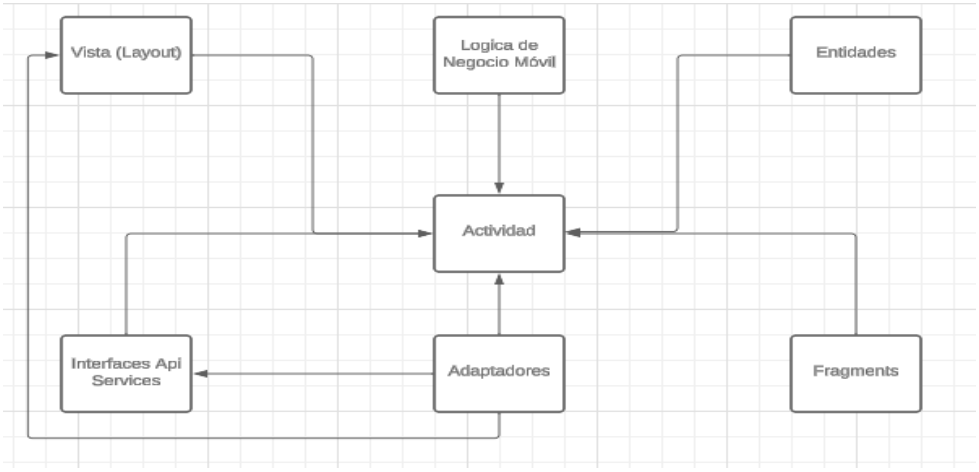


Fig. 22. Diagrama de componentes en móvil.

Descripción de Componentes en móvil:

- Vista Layout: Componente que define la interfaz de usuario y la disposición visual de elementos en la pantalla móvil. Incluye diseños y estructuras visuales para una experiencia de usuario efectiva.
- Lógica de Negocio Móvil: Conjunto de procesos y reglas que gobiernan la funcionalidad y el comportamiento de la aplicación móvil. Se encarga de la manipulación de datos y la interacción con los microservicios para cumplir con los requisitos del negocio.
- Entidades: Representación de objetos de datos en la aplicación móvil, como usuarios, productos o transacciones. Estas entidades capturan información relevante y son gestionadas por la lógica de negocio móvil.
- Activity: Componente de la interfaz de usuario que representa una pantalla o una ventana en la aplicación Android. Las activities interactúan con la lógica de negocio móvil para gestionar eventos y presentar información al usuario.
- Fragments: Componentes modulares de la interfaz de usuario que pueden combinarse para formar una pantalla completa. Facilitan la construcción de interfaces flexibles y reutilizables en la aplicación móvil.
- Adaptadores: Componentes que actúan como intermediarios entre la interfaz de usuario y los conjuntos de datos, permitiendo la presentación eficiente de información en listas o vistas. Facilitan la conexión entre entidades y vistas.
- Interfaces API Services: Interfaces que permiten la comunicación entre la aplicación móvil y los microservicios a través de servicios API. Definen los métodos y formatos de intercambio de datos, asegurando una integración efectiva y segura.

Fig. 23. Componentes en móvil.

En el Anexo 3 visualizan las entidades que responderán las peticiones request de los clientes, y son entidades únicas no relacionales.

Tecnologías Descartadas:

Durante el proceso de selección, evaluamos diversas tecnologías y fundamentamos nuestras decisiones al descartar algunas en favor de otras. A continuación, se detallan las razones específicas detrás del descarte de ciertas tecnologías:

1. Microservicios con Django en lugar de Spring Boot:

   Django: Aunque Django es eficiente para el desarrollo web en Python, optamos por descartarlo en favor de Spring Boot. Además de la experiencia previa con Java, valoramos la robustez de la comunidad de Spring y la preferencia por la Java Virtual Machine (JVM) en nuestro entorno, asegurando una transición suave y un soporte sólido a largo plazo.

   Spring Boot: Elegimos Spring Boot debido a la preferencia por la JVM, la robustez de la comunidad y la eficiencia en Java. Simplifica el desarrollo de microservicios, cuenta con un amplio respaldo comunitario y garantiza eficiencia.

2. Kong en lugar de Feign para el Diseño de API Gateway:

   Kong: Aunque Kong es sólido para API Gateway, decidimos no utilizarlo debido a la necesidad de una solución más ligera y adaptada a nuestras necesidades específicas de modularidad y gestión de solicitudes.

   Feign: Optamos por Feign debido a su diseño más liviano y su integración estrecha con nuestro ecosistema existente, ofreciendo la flexibilidad necesaria para implementar un diseño personalizado y modular.

3. Firebase para la Autenticación y Registro de Usuarios en lugar de Spring:

   Firebase ofrece capacidades sólidas de autenticación, decidimos no seleccionarlo para la gestión de autenticación y registro de usuarios en este proyecto. La principal razón detrás de este descarte fue nuestra preferencia por evitar dependencias de proveedores externos y APIs externas. Independencia de Proveedores Externos: Al optar por soluciones basadas en Spring en lugar de Firebase, buscamos mantener la independencia de proveedores externos. Evitar depender de APIs externas nos proporciona un mayor control sobre la implementación y la seguridad del sistema, reduciendo la posibilidad de posibles interrupciones o cambios en la plataforma externa. Comunicación Asíncrona Eficiente: Al utilizar tecnologías Spring para la autenticación, garantizamos una comunicación asíncrona eficiente y una integración más estrecha con otros componentes de la arquitectura de software. Esto facilita la implementación de funciones de autenticación personalizadas y la adaptación a requisitos específicos del proyecto.

4. Implementación de Proyectos con Spring Boot y MySQL en lugar de Base de Datos NoSQL:

   La elección de MySQL en lugar de una base de datos NoSQL se basa en la necesidad de gestionar relaciones complejas de datos y garantizar la integridad de las entidades en el proyecto. La integración eficiente de MySQL con Spring Boot, respaldada por su

estabilidad y la experiencia asociada al ecosistema, junto con su comprobada escalabilidad, fueron factores clave en esta decisión. Más aún, la selección de MySQL se proyecta hacia el futuro, anticipando la posibilidad de incorporar nuevas entidades que sean pertinentes para microservicios adicionales. Esta decisión estratégica asegura la adaptabilidad y expansión del sistema a medida que evoluciona el proyecto, permitiendo la incorporación de entidades adicionales, como, por ejemplo, para implementaciones de inteligencia artificial u otros requisitos emergentes.

5. Elección de Heroku en lugar de Microsoft Azure para el Despliegue de aplicaciones web:

   La elección de Heroku sobre Microsoft Azure para el despliegue del sistema se fundamentó en consideraciones económicas y operativas. Heroku, con su modelo de precios simplificado y opciones flexibles, brinda una gestión eficiente de recursos y una planificación económica transparente. Su estructura de precios favorece la escalabilidad asequible, eliminando costos innecesarios y proporcionando una opción económica para adaptarse al crecimiento del proyecto. La interfaz intuitiva y la facilidad de uso de Heroku, respaldadas por experiencias positivas anteriores en proyectos similares, contribuyeron a una implementación eficiente, reduciendo la carga operativa y centrando el enfoque en el desarrollo del sistema. En resumen, la elección de Heroku se traduce en un despliegue suave, costos controlados y una plataforma escalable para el desarrollo continuo del proyecto.

6. Elección de Hostinger en lugar de Google Cloud para el Despliegue de bases de datos:

   La preferencia por Hostinger en lugar de Google Cloud como proveedor de servicios de alojamiento se basó en una combinación de factores económicos y de eficiencia operativa. Hostinger ofrece soluciones de alojamiento web con un modelo de precios asequible y transparente, lo que facilita una planificación económica más clara y controlada. Además, sus opciones de hosting proporcionan un rendimiento fiable y

escalabilidad para satisfacer las necesidades del proyecto. En comparación con Google Cloud, Hostinger se percibió como una opción más rentable, eliminando costos innecesarios y permitiendo un uso eficiente de los recursos presupuestarios. La experiencia previa positiva con Hostinger en términos de facilidad de uso y soporte técnico también influyó en esta elección.

3.3.3. Implementar la arquitectura de microservicios para la aplicación móvil

Cada Microservicio administra su propio repositorio de base de datos y desempeña funcionalidades específicas para satisfacer los requerimientos funcionales del Negocio, el prototipo desarrollado tiene 5 microservicios internos que se publican a través de la API Gateway, en las figuras 4, 5, 6 y 7 se muestra la estructura de la solución donde están publicados este api en servidores, así como su base datos y las bidirecciones de cada servicio.

En el contexto de la implementación de microservicios para la parte del cliente de la aplicación móvil, se decidió seguir una arquitectura basada en microservicios para alcanzar el objetivo específico 3. En esta arquitectura, se definió una estructura que permitirá a cada usuario tener su propia base de datos independiente para almacenar su información específica. A continuación, se detallan los componentes clave de esta implementación:

3.3.4.1. Paquetes y definiciones de herramientas de microservicios:

3.3.4.1.1. Package ‘config’:

Este paquete contiene la configuración específica del microservicio para la parte del cliente de la aplicación móvil. Aquí se definen las configuraciones relacionadas con la conexión a bases de datos, seguridad, integración con otros servicios, entre otros.

3.3.4.1.2. Package ‘controller’:

En este paquete se encuentran los controladores que gestionan las solicitudes entrantes de los clientes móviles. Estos controladores actúan como puntos de entrada para las operaciones CRUD independientes que recibirán los request con las entidades de servicios en Eureka, como Categoría, Cliente, compra, Producto y Usuario, Negocio.

3.3.4.1.3. Package ‘entity’:

Aquí se definen las clases de entidad que representan los objetos de dominio de la aplicación. Las clases de entidad incluirán las entidades mencionadas, como Categoría, Cliente, compra, Producto y Usuario, Negocio, con sus respectivos atributos según Anexo 3 y métodos getter and setter.

3.3.4.1.4. Package ‘repository’:

En este paquete se encuentran las interfaces o clases de repositorio que se utilizan para comunicarse con las bases de datos relacionales. Cada entidad, como Categoría, Cliente, compra, Producto y Usuario, Negocio, tendrá su propio repositorio para realizar operaciones de persistencia de datos.

3.3.4.1.5. Package ‘service’:

Aquí se definen las clases de servicio que implementan la lógica de negocio relacionada con cada entidad. Los servicios se encargan de realizar operaciones y cálculos complejos, interactuando con los repositorios y aplicando reglas de negocio específicas.

3.3.4.1.6. Package ‘utils’:

Este paquete contiene clases y utilidades de apoyo que se utilizan en diferentes partes del microservicio móvil. Puede incluir clases de utilidades para el manejo de fechas, generación de identificadores

únicos, formateo de respuestas, encriptación de datos, entre otros.

3.3.4.2. Dependencias utilizas en IntelliJ:

Para el proyecto se definieron las siguientes dependencias en el archivo pom.xml utilizando el proyecto Gradle:

TABLA XII
DEPENDENCIAS PARA LOS MICROSERVICIOS

Definición de dependencias	**Dependencia en Intellij Idea**
spring-boot-starter-data-jpa	Simplifica la configuración y el uso de JPA, lo que acelera el desarrollo de aplicaciones empresariales que trabajan con datos almacenados en bases de datos SQL.
spring-boot-starter-data-rest	Dependencia útil para crear rápidamente una API REST que se basa en los datos almacenados en una base de datos tanto en operaciones CRUD.
spring-boot-starter-security	Dependencia esencial para agregar seguridad a una aplicación Spring Boot, lo que permite gestionar la autenticación y autorización de usuarios, proteger la aplicación contra amenazas de seguridad y personalizar las reglas de acceso a los recursos de la aplicación.
spring-cloud-starter-openfeign	Dependencia esencial para facilitar la comunicación entre servicios en aplicaciones basadas en microservicios utilizando Spring Cloud y Spring Boot.
spring-cloud-starter-netflix-eureka-client	Dependencia clave en aplicaciones basadas en Spring Cloud que permite el registro y el descubrimiento de servicios a través de Eureka.
jjwt-impl jjwt-api jjwt-jackson	API de JWT son esenciales para trabajar con tokens JWT en aplicaciones que requieren autenticación y autorización, como aplicaciones web y servicios REST
mysql-connector-j	Permite establecer conexiones, enviar consultas SQL y gestionar la comunicación con la base de datos, lo que es esencial para aplicaciones que almacenan y recuperan datos en una base de datos MySQL.
lombok	Utilizado en el desarrollo Java para simplificar la creación de clases y reducir la necesidad de escribir código repetitivo, lo que ahorra tiempo y mejora la legibilidad del código
validation-api	API de validación de Java proporciona las anotaciones y la lógica de validación, para que las validaciones se apliquen, es necesario un motor de validación que implemente estas especificaciones. Hibernate Validator es uno de los motores de validación populares que implementa la API de validación de Java.

Nota: Dependencia de los microservicios en Spring usando IntelliJ IDEA 2023.2.5.

En esta sección se define el método que consta de 4 etapas, como guía para el desarrollo de la presente investigación que contempla desde la definición de requerimientos hasta la implementación de la arquitectura de microservicios como soporte para la aplicación móvil. Estos pasos, se describe a continuación.

3.3.4.3. Librerias utilizadas en Android:

En el desarrollo de la aplicación Android, se utilizaron las siguientes librerías según su versión actual:

TABLA XIII
BIBLIOTECAS PARA LA APP EN ANDROID

Definición de dependencias	**Definición de la Dependencia en Android 11**
implementation 'androidx.appcompat:appcompat:1.6.1'	Biblioteca importante para el desarrollo de aplicaciones Android que desean ofrecer una experiencia de usuario coherente y moderna en una amplia gama de dispositivos
implementation 'com.google.android.material:material:1.9.0'	
testImplementation 'junit:junit:4.13.2'	Biblioteca importante para beneficiarse de las correcciones de errores y las mejoras de rendimiento.
implementation 'com.squareup.okhttp3:okhttp:4.9.0'	Nos permite realizar solicitudes de red en aplicaciones de Android y Java. OkHttp se utiliza para interactuar con servidores remotos a través de HTTP o HTTPS, y proporciona una interfaz sencilla y eficiente para enviar solicitudes y recibir respuestas.
implementation 'com.google.code.gson:gson:2.8.8'	Nos permite la comunicación con servicios web y el almacenamiento de datos en una representación serializada.
implementation 'com.squareup.retrofit2:retrofit:2.9.0'	Retrofit simplifica la interacción con servicios web y APIs al convertir solicitudes y respuestas HTTP en llamadas a métodos en Java a través de una interfaz definida por el usuario.
implementation 'com.squareup.retrofit2:converter-gson:2.9.0'	
implementation 'com.squareup.retrofit2:retrofit:2.9.0'	
implementation	Biblioteca fundamental para el diseño de interfaces de usuario en

'androidx.constraintlayout:constraintlayout:2.1.4'	aplicaciones Android y se utiliza para crear diseños flexibles y responsivos mediante la definición de restricciones entre elementos de la interfaz de usuario.
implementation 'com.github.bumptech.glide:glide:4.12.0' annotationProcessor 'com.github.bumptech.glide:compiler:4.12.0'	Biblioteca de carga y visualización de imágenes de código abierto utilizada en aplicaciones Android para gestionar imágenes de manera eficiente
Implementation 'com.google.firebase:firebase-storage:20.0.0'	Biblioteca esencial para el almacenamiento y gestión de archivos en la nube en aplicaciones Android mediante la plataforma Firebase de Google.
implementation 'com.itextpdf:itext7-core:7.1.14'	Biblioteca importante para el manejo de documentos PDF en aplicaciones Java, incluidas las aplicaciones Android. Te permite crear, editar y trabajar con documentos PDF de manera programática.
implementation 'androidx.recyclerview:recyclerview:1.2.1'	Permite una gestión eficiente de elementos en listas y cuadrículas, lo que es esencial para muchas aplicaciones móviles.

Nota: Bibliotecas para la versión de Android Studio 11.

Se describe cada componente del proyecto y se seleccionan Frameworks, bibliotecas y otras tecnologías para implementar la arquitectura, teniendo en cuenta la experiencia de desarrollo de software de los autores de este estudio.

Para realizar solicitudes a cada microservicio identificado, tienen una ruta final de recursos a la que se puede acceder realizando una solicitud utilizando el protocolo HTTP REST, se construye una API para cada microservicio utilizando puntos finales y el protocolo HTTP REST, la Tabla 5 muestra la ruta final. rutas y tipos de solicitudes para microservicios y recursos.

TABLA XIV

Rutas de los microservicios

Microservice	**Tareas**	**Tipo de Petición**	**Ruta Final**
1. API GATEWAY	-Crear Usuario	POST	/api/authentication/sign-up
	-Ingresar Usuario	POST	/api/authentication/sign-in
	-Listar Usuario	GET	/api/user/listar

	-Buscar Usuario	GET	/api/user/{id}
	-Dar Rol	PUT	/api/user/change/{Role}
	-Actualizar Usuario	PUT	/api/user/id
	-Eliminar Usuario	DELETE	api/user/id
2. PRODUCTO			
	-Crear Producto	POST	/api/producto
	-Listar Productos	GET	/api/producto
	-Actualizar Producto	PUT	/api/producto/{id}
	- Buscar Producto	GET	/api/producto/{id}
	-Eliminar Producto	DELETE	/api/producto/{id}
	-Encontrar Producto por nombre	POST	/api/producto/buscar
3. COMPRA			
	-Crear Compra	POST	/api/compra
	-Listar Compra de Usuari	GET	/api/compra
	-Listar todas las Compras	GET	/api/compra/all
	-Cambiar estado Compra	PUT	/api/producto/{id}
4. NEGOCIO			
	-Crear Negocio	POST	/api/negocios/
	-Listar Negocios	GET	/api/negocios/
	-Encontrar Negocio por ID	GET	/api/negocios(id}
	-Actualizar Negocio	PUT	/api/negocios/{id}
	-Eliminar Negocio	DELETE	/api/negocios/{id}
5. CATEGORIA			
	-Registrar Categoría	POST	/api/categoría
	-Listar Categorías	GET	/api/categoria
	-Actualizar Categoría	PUT	/api/categoria/{categoriaId}
	-Eliminar Categoría	DELETE	/api/categoria/{categoriaId}

Nota: Endpoints de los microservicios publicados en sus dominios propios.

3.3.3.1. Ejecución del proyecto

3.3.3.1.1. Planificación del Proyecto:

3.3.3.1.1.1. Historias de Usuario:

Tras múltiples encuentros con el cliente, se fijaron los requerimientos del sistema

informático. Analice cada posible escenario elaborando una narrativa de usuario, que se especifica en las tablas siguientes.

TABLA XV

HISTORIA DE USUARIO – CREAR LOS SERVICIOS

Historia de Usuario	
Número: 001	Usuario: Todos
Nombre de historia: Crear en Spring todos los microservicios	
Prioridad: Alta	Riesgo: Baja
Puntos: 8	Iteración asignada: 001
Programador responsable: Dagner Chuman Lluen	
Descripción: Esta historia de usuario implica la creación de todos los microservicios necesarios para el funcionamiento del sistema utilizando el framework Spring. Incluye el desarrollo y configuración de los microservicios esenciales para la arquitectura propuesta.	

Nota: La creación exitosa de estos microservicios es fundamental para el desempeño adecuado de las funciones del sistema.

TABLA XVI

HISTORIA DE USUARIO – CREAR E CONFIGURAR EUREKA Y API GATEWAY

Historia de Usuario	
Número: 002	Usuario: Todos
Nombre de historia: Crear y configurar en eureka server con bidirección en Feign para Api Gateway Request de todos los microservicios	
Prioridad: Alta	Riesgo: Baja
Puntos: 2	Iteración asignada: 002
Programador responsable: Dagner Chuman Lluen	
Descripción: Esta historia de usuario aborda la implementación de Eureka Server para la gestión de servicios y la configuración bidireccional en Feign para las solicitudes a través de Api Gateway. Garantiza una comunicación eficiente y sin problemas entre los microservicios	

Nota: La correcta configuración de Eureka y Api Gateway es esencial para asegurar la conectividad entre los microservicios.

TABLA XVII

HISTORIA DE USUARIO – CONFIGURACIÓN DE SPRING SECURITY

Historia de Usuario	
Número: 003	Usuario: Todos
Nombre de historia: Se configura Spring Security	
Prioridad: Alta	Riesgo: Baja

Puntos: 2	Iteración asignada: 003

Programador responsable: Dagner Chuman Lluen

Descripción: En esta historia de usuario, se lleva a cabo la configuración de Spring Security para garantizar la seguridad y protección de los microservicios. Esto incluye la implementación de autenticación y autorización para los servicios.

Nota: La correcta configuración de Spring Security es crucial para proteger los microservicios contra accesos no autorizados.

TABLA XVIII

HISTORIA DE USUARIO – SERVICIOS EN LA NUBE

Historia de Usuario

Número: 004	Usuario: Todos

Nombre de historia: Investigación sobre servicios en la nube más adecuada y económica, para su posterior publicación en Heroku buscando las librerías más adecuadas en spring.

Prioridad: Alta	Riesgo: Baja
Puntos: 2	Iteración asignada: 004

Programador responsable: Dagner Chuman Lluen

Descripción: En esta historia de usuario, se realiza una investigación detallada sobre los servicios en la nube más adecuados y económicos. Posteriormente, se procede a la publicación en Heroku, asegurándose de seleccionar las librerías más apropiadas en el entorno Spring.

Nota: La elección acertada de los servicios en la nube y las librerías contribuye significativamente al rendimiento y la eficiencia del sistema.

TABLA XIX

HISTORIA DE USUARIO – CREAR LOS SERVICIOS Y DESARROLLO EN APP MÓVIL.

Historia de Usuario

Número: 005	Usuario: Todos

Nombre de historia: Se configura los endpoints en api Gateway de cada IP publica de cada microservicio, y se configura las librerías en Android studio 11, mandando la ip publica de Api Gateway, y se desarrolla la app móvil.

Prioridad: Alta	Riesgo: Baja
Puntos: 2	Iteración asignada: 005

Programador responsable: Dagner Chuman Lluen

Descripción: Esta historia de usuario involucra la configuración de los endpoints en Api Gateway, asignando una IP pública a cada microservicio. Además, se realiza la configuración de librerías en Android Studio 11 para desarrollar la aplicación móvil, estableciendo la comunicación con Api Gateway a través de la IP pública correspondiente.

Nota: El desarrollo exitoso de la aplicación móvil es esencial para proporcionar a los usuarios una experiencia fluida y funcional.

TABLA XX

HISTORIA DE USUARIO - ACCESO AL SISTEMA

Historia de Usuario	
Número: 006	Usuario: Todos
Nombre de historia: Acceso al sistema	
Prioridad: Alta	Riesgo: Baja
Puntos: 2	Iteración asignada: 006
Programador responsable: Dagner Chuman Lluen	
Descripción: Se ingresa al sistema se solicita el email y contraseña, las cuales permitirán acceder al app o web para administrar donde tienen sus roles (Superadministrador, administrador, User)	

Nota: El acceso eficiente y seguro al sistema es crucial para garantizar la autenticación de los usuarios.

TABLA XXI

HISTORIA DE USUARIO - GESTIONAR CATEGORÍA.

Historia de Usuario	
Número: 007	Usuario: Administrador
Nombre de historia: Gestionar Categoria	
Prioridad: Alta	Riesgo: Media
Puntos: 1	Iteración asignada: 007
Programador responsable: Dagner Chuman Lluen	
Descripción: El usuario con rol administrador podrá registrar, buscar, modificar o eliminar los datos de una categoría.	

Nota: La gestión eficiente de categorías es fundamental para organizar y presentar productos de manera adecuada en el sistema.

TABLA XXII

HISTORIA DE USUARIO - REGISTRARSE EN EL SISTEMA.

Historia de Usuario	
Número: 008	Usuario: Todos
Nombre de historia: Registrar en el sistema	
Prioridad: Alta	Riesgo: Alta
Puntos: 2	Iteración asignada: 008
Programador responsable: Dagner Chuman Lluen	
Descripción: Esta funcionalidad se llevará a cabo el proceso para registrarse en el sistema donde deben llenar los datos los usuarios de : nombre, apellido, teléfono, email, username, negocio, DNI, picture, departamento, provincia, distrito.	

Nota: El proceso de registro exitoso es crucial para obtener información precisa de los usuarios.

TABLA XXII

HISTORIA DE USUARIO - GESTIONAR USUARIOS

	Historia de Usuario
Número: 009	Usuario: Administrador
Nombre de historia: Gestionar Usuarios	
Prioridad: Alta	Riesgo: Alta
Puntos: 2	Iteración asignada: 009
Programador responsable: Dagner Chuman Lluen	
Descripción: El administrador podrá registrar, buscar y gestionar los datos de los usuarios.	
Nota: La gestión eficaz de usuarios es esencial para mantener actualizada la base de datos de usuarios.	

TABLA XXIV

HISTORIA DE USUARIO - GESTIONAR NEGOCIOS.

	Historia de Usuario
Número: 010	Usuario: Superadministrador
Nombre de historia: Gestionar Negocios	
Prioridad: Alta	Riesgo: Alta
Puntos: 2	Iteración asignada: 010
Programador responsable: Dagner Chuman Lluen	
Descripción: El superadministrador podrá registrar, buscar y gestionar los datos de los negocios.	
Nota: La gestión eficiente de negocios es crucial para mantener la información actualizada y precisa.	

TABLA XXV

HISTORIA DE USUARIO – ENSEÑAR LOGIN

Historia de Usuario	
Número: 011	Usuario: Cliente concurrente
Nombre de historia: ENSEÑAR_PRIMER_LAYOUT_LOGIN	
Prioridad: Media	Riesgo: Baja
Puntos: 6	Iteración asignada: 011
Programador responsable: Dagner Chuman Lluen	
Descripción: El usuario al acceder a la app móvil, le aparecerá una pantalla de slash posterior al logín, donde se procederá con el proceso de Autenticación.	
Nota: La presentación visual del login es esencial para ofrecer una experiencia atractiva y fácil para los usuarios.	

TABLA XXVI

HISTORIA DE USUARIO – ENSEÑAR PRODUCTOS

Historia de Usuario	
Número: 012	Usuario: Cliente concurrente
Nombre de historia: ENSEÑAR_LAYOUT_ITEM_PRODUCTOS	

Prioridad: Media	Riesgo: Baja
Puntos: 5	Iteración asignada: 012
Programador responsable: Dagner Chuman Lluen	
Descripción: El usuario al acceder a la app móvil, posterior de la autenticación le aparecerá en que negocio se registró, se mostrará todo un listado de categorías, productos que existen en el negocio.	
Nota: La presentación clara y organizada de productos es esencial para facilitar la navegación de los usuarios.	

TABLA XXVII

HISTORIA DE USUARIO – LAYOUT EN COMPRAS

Historia de Usuario	
Número: 013	Usuario: Cliente concurrente
Nombre de historia: ENSEÑAR_LAYOUT_COMPRAS	
Prioridad: Alta	Riesgo: Baja
Puntos: 5	Iteración asignada: 013
Programador responsable: Dagner Chuman Lluen	
Descripción: El usuario debe colocar todos los productos en el carrito a seleccionar, y debe verificar que con la escalabilidad de todos los usuarios que adquieran, este mismo pueda adquirir el producto, si no en todo caso se le muestra una sweet alert o alerta para que se le avisé que no hay stock.	
Nota: El diseño efectivo del layout de compras es esencial para facilitar la experiencia de compra de los usuarios.	

TABLA XXVIII

HISTORIA DE USUARIO – REQUISITOS DE TODO SISTEMA

Historia de Usuario	
Número: 014	Usuario: Programador y clientes
Nombre de historia: Requisitos	
Prioridad: Alta	Riesgo: Baja
Puntos: 6	Iteración asignada: 014
Programador responsable: Dagner Chuman Lluen	
Descripción: El usuario debe entender el fácil uso de la app móvil sin tanto necesitar de una guía, y debe ser interactiva e rápida.	
Nota: Cumplir con los requisitos del sistema es esencial para garantizar una aplicación intuitiva y satisfactoria para los usuarios.	

Actividades

TABLA XXIX

ACTIVIDAD DE HISTORIA INTERFAZ ACCESO AL SISTEMA

Historia de Usuario

Codigo: T1	Codigo de Historio: H1
Nombre de tarea: Diseñar Interfaz Acceso al Sistema.	
Tipo de tarea: Desarrollo	Puntos: 0.8
Programador responsable: Dagner Chuman Lluen	
Tiempo: 2 dias	
Descripción: Diseñar un layout de inicion de sesión y diseñar el logín en web para administración, se debe ingresar el email, y password, se añade un botón de olvidé contraseña o un deslizador para poder registrarse si no está registrado.	

Nota: La descripción detalla la necesidad de un layout de inicio de sesión y un diseño de login web, incluyendo campos para email y password, así como opciones para recuperar contraseña o registrarse.

TABLA XXX

ACTIVIDAD DE HISTORIA VALIDACIONES PRUEBAS

	Historia de Usuario
Codigo: T2	Codigo de Historio: H2
Nombre tarea: Validaciones para Acceder al Sistema	
Tipo de tarea: Desarrollo	Puntos: 0.4
Programador responsable: Dagner Chuman Lluen	
Tiempo: 2 dias	
Descripción: Se debe implementar la conexión a los microservicios, además de crear clase y métodos para validar el acceso a diferentes tipos de validaciones donde el usuario se puede equivocar o no tener accesos.	

Nota: Se destaca la importancia de conectar con microservicios y desarrollar métodos para validar diferentes situaciones en las que un usuario pueda cometer errores o no tener los permisos adecuados.

TABLA XXXI

ACTIVIDAD DE HISTORIA MVC

	Historia de Usuario
Codigo: T3	Codigo de Historio: H3
Nombre tarea: Modelo, Vista y Controlador para Categorías, Productos	
Tipo de tarea: Desarrollo	Puntos: 0.4
Programador responsable: Dagner Chuman Lluen	
Tiempo: 3 dias	
Descripción: Se diseña actividades, adaptadores, layouts,en Android, servicios para recibir los endpoints de los microservicios.	

Nota: La descripción incluye la creación de actividades, adaptadores, layouts en Android, y servicios para recibir endpoints de microservicios.

TABLA XXXII

ACTIVIDAD DE HISTORIA DE DISEÑOS EN APP MOVIL

Historia de Usuario

Codigo: T4 Codigo de Historio: H4

Nombre tarea: Diseñar interfaces de categorías, para gestionar las categorías, productos y notificaciones para ofertas o avisos de sus compras.

Tipo de tarea: Desarrollo Puntos: 0.7

Programador responsable: Dagner Chuman Lluen

Tiempo: 2 días

Descripción: Desarrollar interfaces, clases y métodos que permita interactuar al usuario.

Nota: Se espera que se creen clases y métodos que permitan una interacción efectiva con el usuario.

TABLA XXXIII

Módulo de Acceso, Usuarios Plan de entregas:

N°	Historia de Usuario	Tiempo estimado		
		Semanas	Días	Horas
1	Crear en Spring todos los microservicios	1	1	10
2	Crear y configurar en eureka server con bidirección en Feign para Api Gateway Request de todos los microservicios	1	7	84
3	Se configura Spring Security	2	10	100
4	Investigación sobre servicios en la nube más adecuada y económica, para su posterior publicación en Heroku buscando las librerías más adecuadas en spring	2	5	50
5	Se configura los endpoints en api Gateway de cada IP publica de cada microservicio, y se configura las librerías en Android Studio 11, mandando la ip publica de Api Gateway, y se desarrolla la app móvil.	5	30	200
6	Acceso al sistema	1	5	20
7	Registrar Categoria	1	5	20
8	Registrar en el sistema	1	5	20
9	Gestionar Usuarios	2	10	40
10	Gestionar Negocios	2	10	40
11	ENSEÑAR_PRIMER_LAYOUT_LOGIN	1	5	20
12	ENSEÑAR_LAYOUT_ITEM_PRODUCTOS	1	5	20
13	ENSEÑAR_LAYOUT_ COMPRAS	1	5	20
14	Requisitos de Interfaz	1	5	20
T1	Diseñar Interfaz Acceso al Sistema.	1	5	20

T2	Validaciones para Acceder al Sistema	1	5	20
T3	Modelo, Vista y Controlador para Categorías, Productos	1	5	20
T4	Diseñar interfaces de categorías, para gestionar las categorías, productos y notificaciones para ofertas o avisos de sus compras.	1	5	20
Tiempo estimado total		26	128	744

Nota: Se presenta un plan detallado de entregas con estimaciones de tiempo para cada tarea.

Iteraciones:

Las iteraciones en el proyecto se implementan conforme a las diversas versiones, las cuales incorporan actividades adicionales y actualizaciones en la aplicación móvil. Estas modificaciones se detallan en la figura siguientes en Google Play Console.

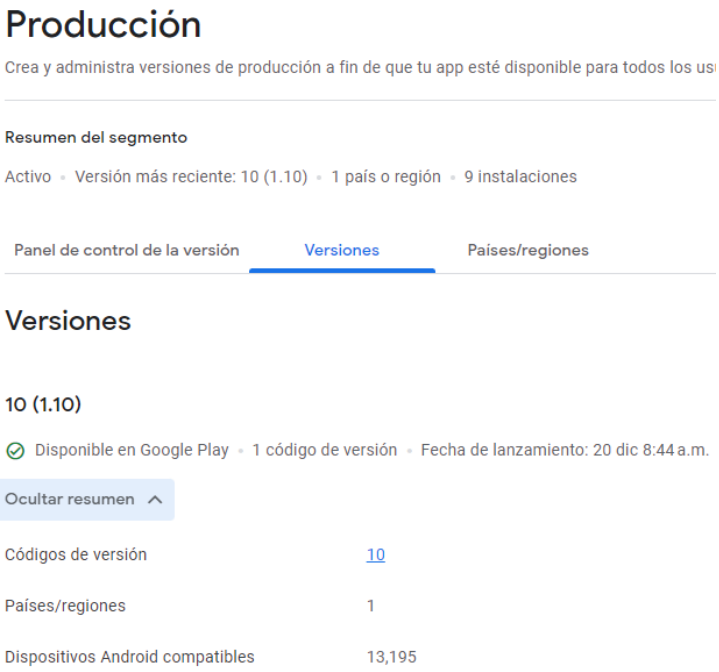


Fig. 24. Iteración de versiones en Google Play Console.

3.3.3.2. Diseño

En la metodología XP para el diseño de software orientado a objetos, se emplean las tarjetas CRC (Clase-Responsabilidad-Colaboración) como una herramienta integral en el proceso de diseño y organización. La historia de usuario se traduce en una tarjeta CRC, brindando funcionalidades directas al negocio. En este contexto, una "clase" puede representar una entidad diversa, como una persona, objeto, evento, concepto, pantalla o

informe. Las "responsabilidades" de una clase engloban las acciones que realiza y la información que posee a través de sus atributos y métodos. Asimismo, los "colaboradores" de una clase son otras clases con las cuales interactúa para cumplir sus responsabilidades. Este enfoque facilita la comprensión y planificación del diseño, destacando las relaciones entre clases y optimizando la implementación del software.

TABLA XXXIV

CONFIGURACIÓN DE MICROSERVICIOS EN SPRING BOOT

Crear en Spring todos los microservicios	
Responsabilidad	Colaboradores
Desarrollar los microservicios necesarios utilizando Spring Boot.	Planificación del Desarrollo Implementar cada microservicio siguiendo las prácticas y estándares de desarrollo de Spring Boot. Realizar pruebas unitarias exhaustivas para garantizar la funcionalidad y robustez de cada microservicio. Implementar un sistema de integración continua para asegurar la compatibilidad y cohesión entre los microservicios. Realizar revisiones de código periódicas para garantizar la calidad y consistencia del código en todos los microservicios.
Observaciones: Mantener una comunicación constante entre los desarrolladores para abordar posibles problemas y asegurar la coherencia entre los microservicios.	

TABLA XXXV

CONFIGURACIÓN DE EUREKA SERVER CON FEIGN PARA API GATEWAY REQUEST

Crear y configurar en Eureka Server con bidirección en Feign para Api Gateway Request de todos los microservicios	
Responsabilidad	Colaboradores
Configurar Eureka Server con bidireccionalidad en Feign para la gestión de solicitudes a través del Api Gateway.	Establecer y configurar Eureka Server para el registro y descubrimiento de microservicios. Implementar Feign para la comunicación bidireccional entre Eureka Server y el Api Gateway. Realizar pruebas integrales para asegurar la correcta configuración y comunicación entre Eureka Server, Feign y el Api Gateway.
Observaciones: Establecer un proceso de monitoreo continuo para detectar y abordar posibles problemas de comunicación.	

TABLA XXXVI

CONFIGURACIÓN DE SPRING SECURITY PARA LA SEGURIDAD DEL SISTEMA

Configuración de Spring Security	
Responsabilidad	Colaboradores
Configurar Spring Security para garantizar la seguridad del sistema.	Evaluar los requisitos específicos de seguridad para el sistema. Implementar y configurar Spring Security para proteger los recursos del sistema. Realizar pruebas de seguridad exhaustivas para identificar y abordar posibles vulnerabilidades.

Observaciones: Mantener actualizaciones regulares de las configuraciones de seguridad en respuesta a las nuevas amenazas y vulnerabilidades.

TABLA XXXVII

Validación de Acceso al Sistema

Acceso al sistema	
Responsabilidad	Colaboradores
Validar email y contraseña	Capa de acceso a los datos Métodos Validación de datos
Observaciones:	

3.3.3.3. Desarrollo

Base de Datos

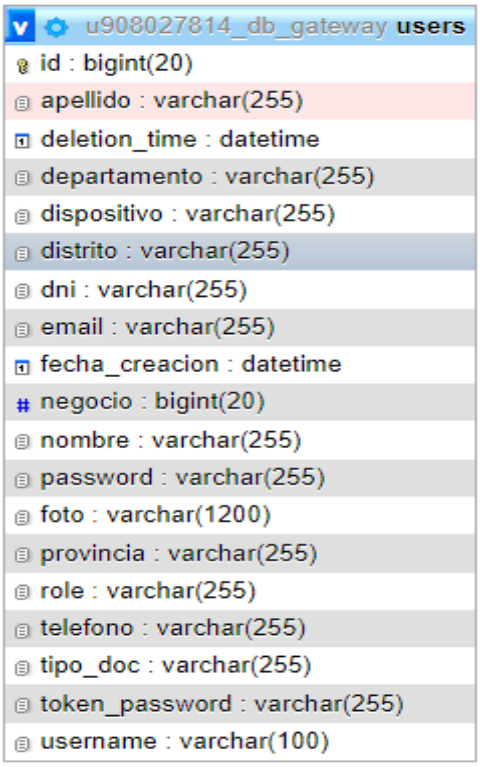


Fig. 25. Estructura de la base de datos del microservicio Gateway.

En la figura 25 se muestra la base de datos de la entidad Users diseñada en MySQL de Hostinger.

Fig. 26. Estructura de la base de datos del microservicio Negocio.

En la figura 26 se muestra la base de datos de la entidad Negocio diseñada en MySQL de

Hostinger.

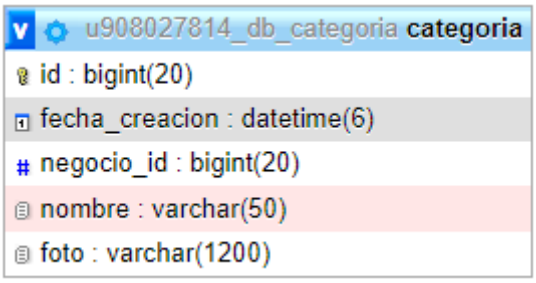


Fig. 27. Estructura de la base de datos del microservicio Categoria.

En la figura 27 se muestra la base de datos de la entidad Categoria diseñada en MySQL de Hostinger.

Fig. 28. Estructura de la base de datos del microservicio Producto.

En la figura 28 se muestra la base de datos de la entidad Producto diseñada en MySQL de Hostinger.

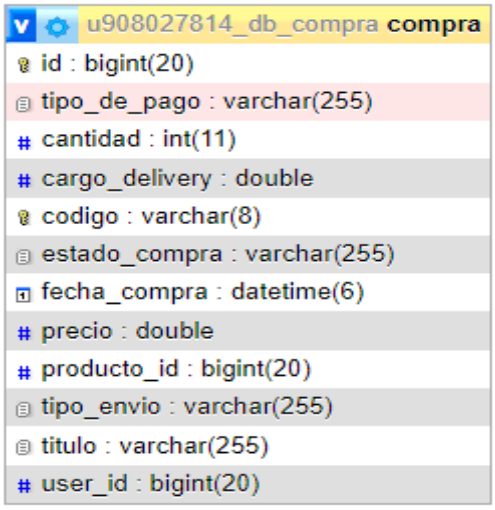


Fig. 29. Estructura de la base de datos del microservicio Compra.

En la figura 29 se muestra la base de datos de la entidad Compra diseñada en MySQL de Hostinger.

Diseño de Interfaces de Usuario

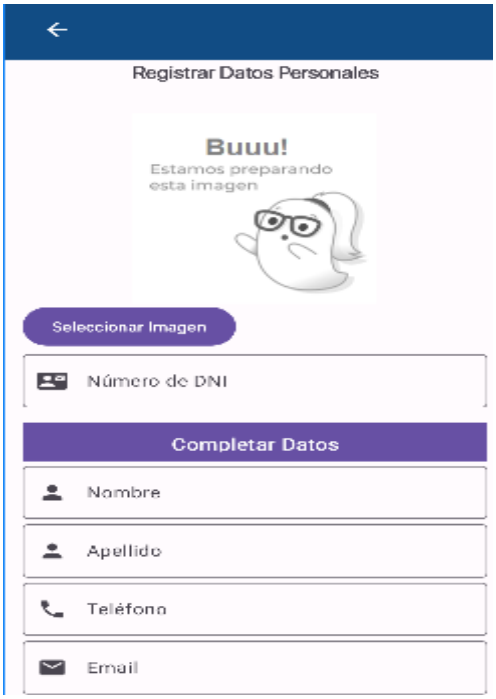


Fig. 30. Diseño de Interfaz de Registro

Fig. 31. Diseño de Interfaz Login.

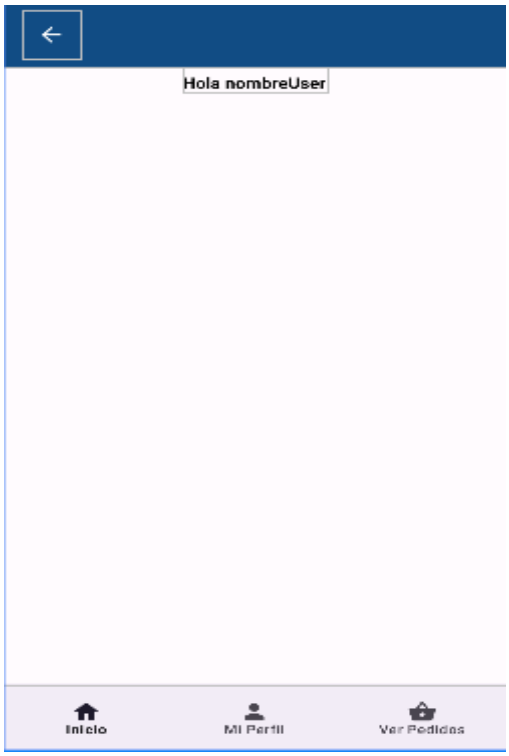


Fig. 32. Diseño de Interfaz Acceso al sistema.

Fig. 33. Diseño de Interfaz entrada al sistema.

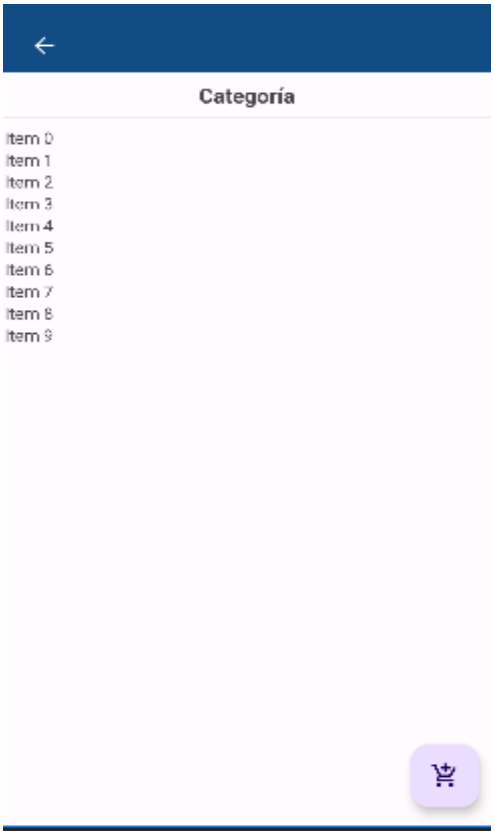


Fig. 34. Diseño de Interfaz de categoria de productos

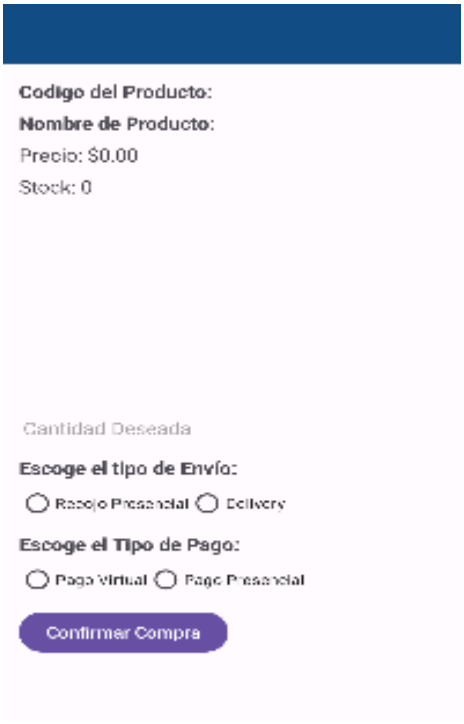


Fig. 35. Diseño de Interfaz de compra unitaria.

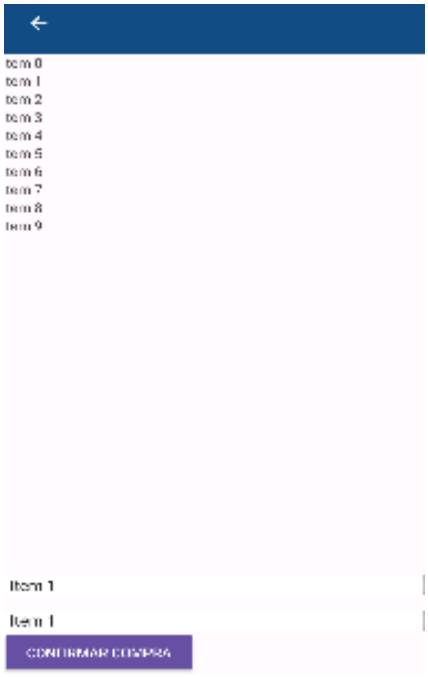


Fig. 36. Diseño de Interfaz de Compra en carrito.

Codificación

Según Anexo 6, en la nube Git Hub de cada interfaz y microservicios. La implementación del sistema informático se realizó a partir del mes de mayo del presente año. La plataforma de Hostinger se utilizó para alojar la base de datos MySQL, y los servicios desarrollados con Spring Boot, configurados como microservicios, fueron desplegados en la plataforma Heroku. Además, la aplicación móvil ya se encuentra en producción y está disponible en el servidor de Google Play Store desde Octubre enviando versiones constantes. La validación de las funcionalidades del sistema se llevó a cabo en entornos de prueba gestionados por el desarrollador y en colaboración con el cliente del Minimarket Job que es el negocio especifico e principal y porque que hay nuevos requerimientos. A la fecha actual, la aplicación móvil ha sido bien recibida por los usuarios y ha demostrado un rendimiento satisfactorio en el entorno de producción de Google Play Store. Se continúa trabajando en mejoras continuas y actualizaciones para asegurar el óptimo funcionamiento y la satisfacción de los usuarios.

3.3.3.4. Pruebas

TABLA XXXVIII

TABLA XXXIV - PRUEBA DE ACEPTACIÓN: CREACIÓN EXITOSA (P1)

Prueba de Aceptación	
Código: P1	Código de historia: 1
Nombre: Prueba de Creación Exitosa	
Descripción: Verificar que la creación de todos los microservicios en Spring se realice de manera exitosa.	
Condición de ejecución: Después de ejecutar el proceso de creación en Spring.	
Entrada: Configuración de los microservicios y sus dependencias en Spring.	
Resultado: Los microservicios se crearon sin errores y están en funcionamiento.	
Evaluación de prueba: La prueba se concluyó satisfactoriamente	

Nota: La prueba "Creación Exitosa" se enfoca en verificar que la creación de microservicios en Spring sea exitosa

TABLA XXXIX

PRUEBA DE ACEPTACIÓN: CONEXIÓN MICROSERVICIOS (P2)

Prueba de Aceptación

Código: P2	Código de historia: 2
Nombre: Prueba de Conexión	
Descripción: Verificar que los microservicios puedan conectarse entre sí.	
Condición de ejecución: Después de que todos los microservicios están en funcionamiento.	
Entrada: Realizar una solicitud desde un microservicio hacia otro.	
Resultado: La conexión entre microservicios es exitosa, y se obtiene la respuesta esperada.	
Evaluación de prueba: La prueba se concluyó satisfactoriamente	

Nota: Se ejecuta después de que todos los microservicios estén en funcionamiento, realizando solicitudes entre ellos y validando la conexión y respuestas esperadas.

TABLA XL

PRUEBA DE ACEPTACIÓN: ESCALABILIDAD MICROSERVICIOS (P3)

	Prueba de Aceptación
Código: P3	Código de historia: 3
Nombre: Prueba de Escalabilidad	
Descripción: Evaluar la capacidad de los microservicios para manejar una carga escalada.	
Condición de ejecución: Después de aumentar gradualmente la carga de solicitudes.	
Entrada: Aumentar el número de solicitudes concurrentes.	
Resultado: Los microservicios mantienen el rendimiento sin errores críticos.	
Evaluación de prueba: La prueba se concluyó satisfactoriamente	

Nota: La prueba "Escalabilidad" evalúa la capacidad de los microservicios para manejar una carga escalada.

TABLA XLI

PRUEBA DE ACEPTACIÓN: CONEXIÓN APP MÓVIL - API GATEWAY (P4)

	Prueba de Aceptación
Código: P4	Código de historia: 4
Nombre: Prueba de Conexión App Móvil - Api Gateway	
Descripción: Verificar la conexión exitosa de la app móvil con Api Gateway y la comunicación con los microservicios	
Condición de ejecución: Después de configurar la app móvil con las librerías de Api Gateway.	
Entrada: Solicitudes desde la app móvil a través de Api Gateway.	
Resultado: La app móvil se conecta correctamente a Api Gateway y recibe respuestas esperadas de los microservicios.	
Evaluación de prueba: La prueba se concluyó satisfactoriamente	

Nota: Se ejecuta después de configurar la app móvil con las librerías de Api Gateway, realizando solicitudes y validando la conexión y respuestas de los microservicios.

3.3.4. Evaluar la implementación la arquitectura de microservicios para la aplicación móvil

Para evaluar el prototipo con respecto a la arquitectura de software propuesta y realizar pruebas de medición de indicadores críticos como tiempo de respuesta, tiempo de espera, rendimiento y disponibilidad, e donde aplicaremos la norma ISO 25010 para la variable dependiente (Seguridad, rendimiento, escalabilidad, etc.). Estas pruebas se llevarán a cabo después de la implementación en la nube (Heroku), donde el servicio gestionará todos los procesos internos relacionados con la obtención de productos de inventario de Minimarkets, el registro de clientes, y la realización de pedidos y compras. Asimismo, se evaluará el resultado de la prueba en la población del Minimarket.

Al igual que en la investigación denominada “Arquitectura de software basada en microservicios para mejorar la disponibilidad de historias clínicas electrónicas odontológicas, Chiclayo – Lambayeque, 2020” [30]. Para evaluar cada uno de los atributos de calidad se han utilizado las métricas de calidad y su correspondiente valor según la arquitectura de software propuesta en esta investigación, en la tabla 11 se muestran las métricas de calidad y su valor correspondiente para la arquitectura propuesta.

A partir de los cuales se obtiene los resultados para nuestros indicadores: Antes de la implementación de los indicadores, se identifica los atributos de calidad utilizando las métricas de calidad y sus respectivos valores.

TABLA XLII

MÉTRICAS DE CALIDAD Y SUS VALORES

Métrica	Valor de Microservicios
Número de mensajes proporcionados	4
Número de servicios	5
Número de servicios conectado	2
Número de servicios consumidores	1
Número de servicios dependientes	1

Número de consumidores del servicio	5

Nota: Adaptado de [30]

Generalidades

En su beta 1.0 la app Altoque y la web DotVal, que tiene por siguiente las características de:

a) Registro de los Negocios en la web.

b) Registro de los usuarios usando el api de RENIEC.

c) Registro y actualización de los productos con sus respectivas categorías.

d) Registro y búsqueda del producto en la modalidad de compra pedido, con su respectiva boleta.

Se puede acceder a este sitio web a través de su URL pública, y también descargar desde la Play Store.

Este es una guía para comenzar con la red y las aplicaciones.

En la Figura 37, se realiza el registro de datos del negocio. En este escenario, el autor opta por llevar a cabo de manera privada la creación de los negocios respectivos, con el propósito de que estos proporcionen:

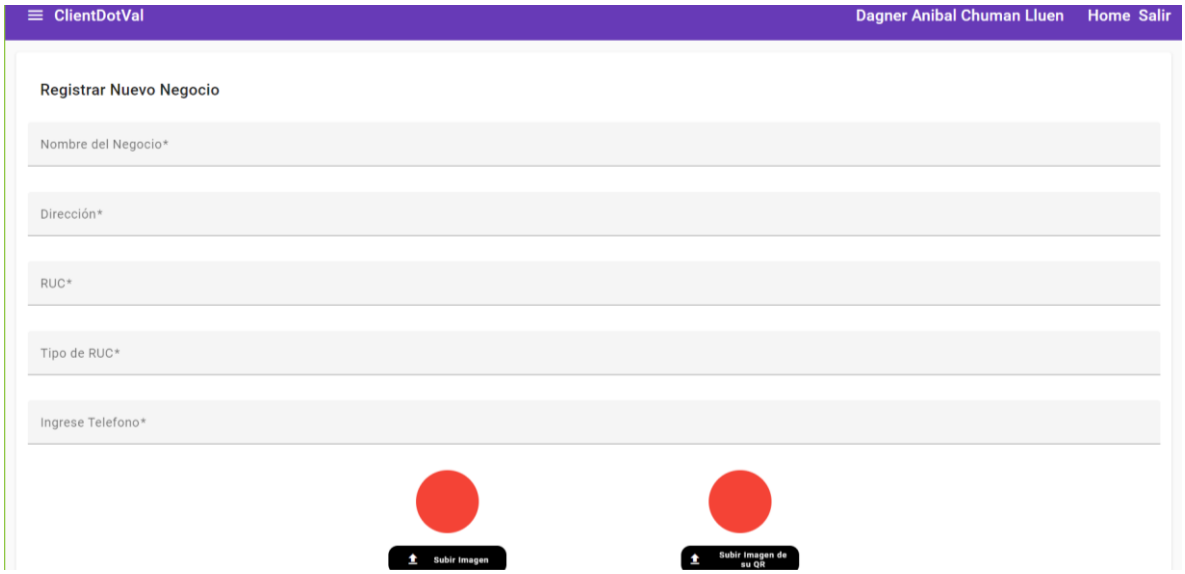


Fig. 37. WebApp del prototipo realizado para registrar muchos negocios

Fuente: Elaboración propia.

Una vez que los datos del negocio han sido registrados, se tiene la opción de formalizar un contrato para que el negocio utilice los servicios de DotVal. En este punto, se le otorga al usuario un rol de "ADMIN", lo que le permite acceder tanto a la plataforma web como a la aplicación DotVal. Alternativamente, los usuarios pueden registrarse por sí mismos, designándose como "USER". En este caso, el autor del software proporcionará los accesos a su cuenta mediante el token generado durante el inicio de sesión.

Autenticación:

Para poder acceder se debe tener en cuenta a que negocio se registró o en que cuenta pertenece tal usuario, en el formulario de acceso se ingresa el email y el password correspondiente, si los datos son correctos se crea el token de acceso y se ingresa al dashboard para poder consultar el negocio y sus productos.

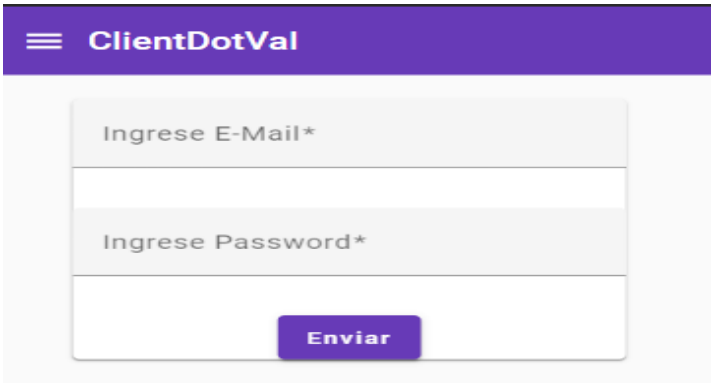


Fig. 38. Login WebApp

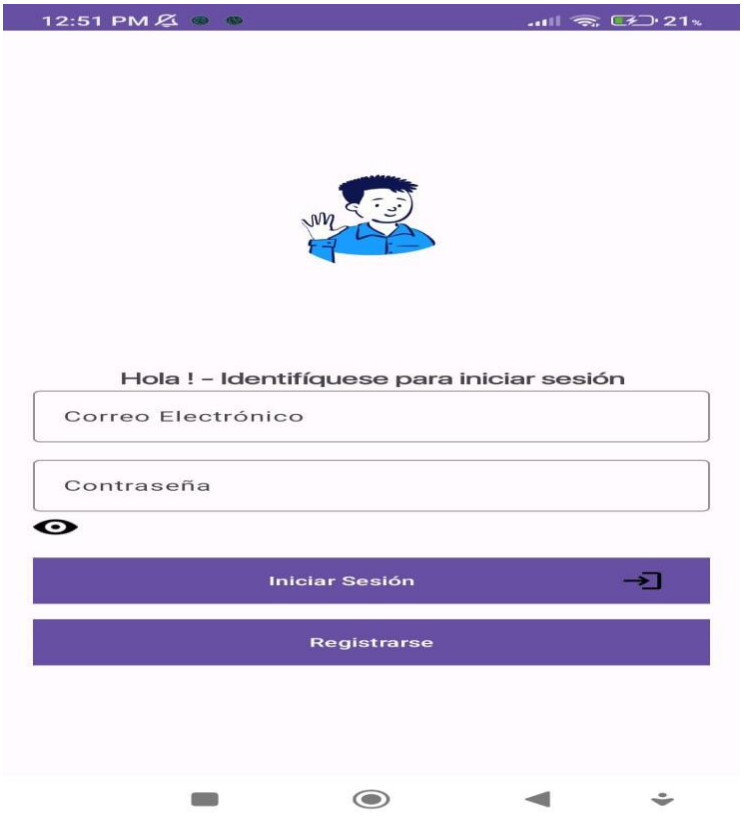


Fig. 39. Login ClientApp Play Store

Interfaz principal como USER:

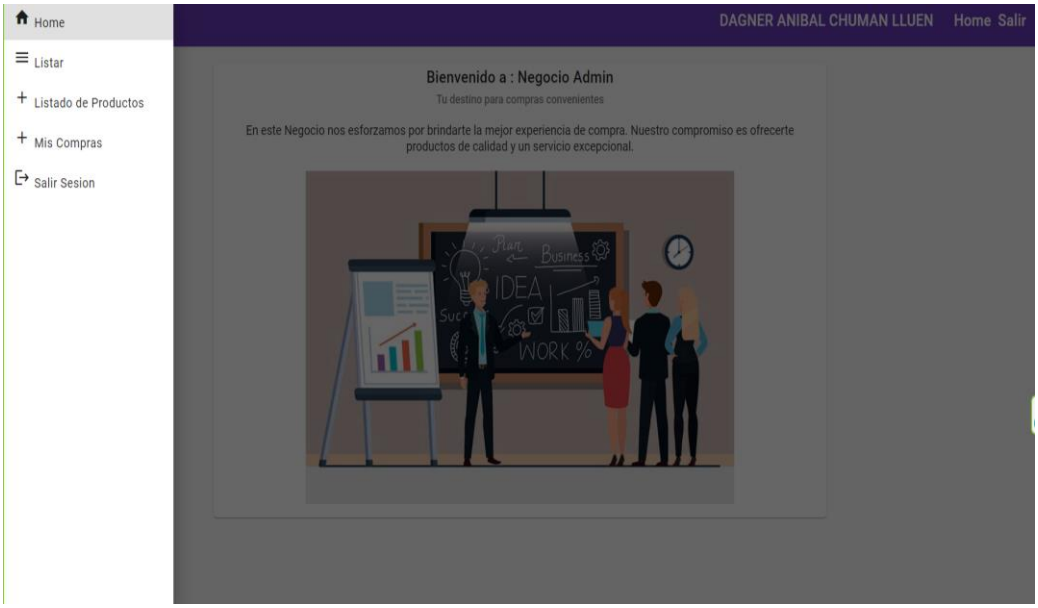


Fig. 40. Interfaz Principal inicio

Consulta de listado de productos de un negocio

En la figura 26, se muestra como un usuario consulta de Productos de un Negocio en móvil y web:

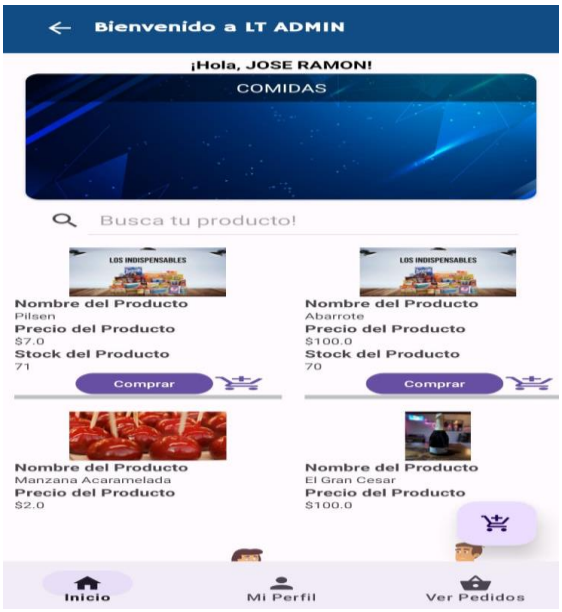


Fig. 41. App del primer negocio

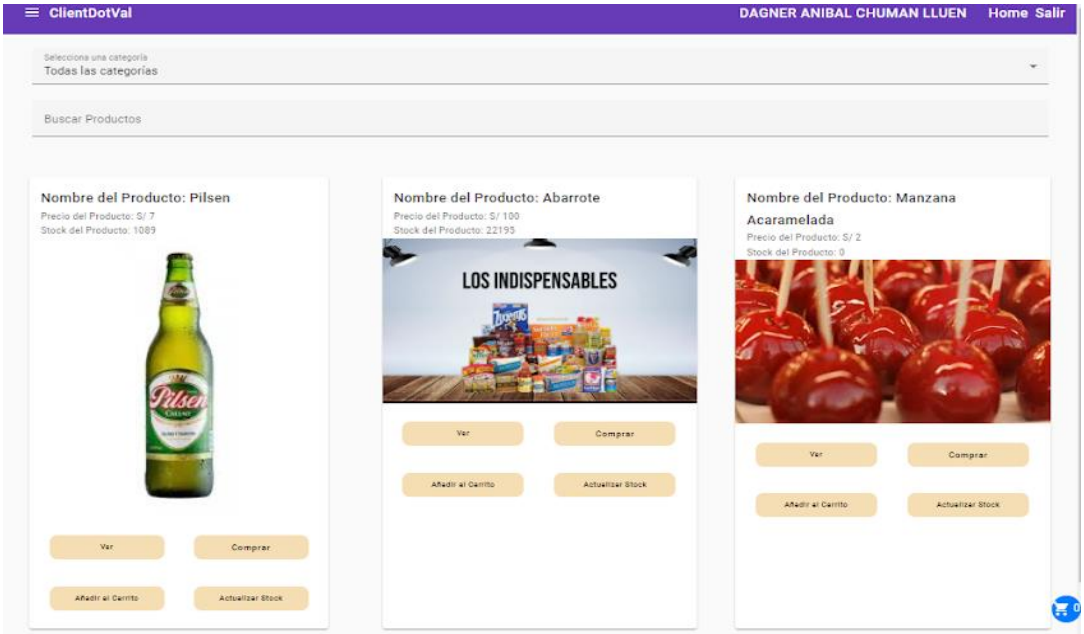


Fig. 42. WebApp del primer negocio

En la figura 28, se muestra como un usuario consulta en Negocio Alternativo:

Fig. 43. Interfaz principal de un segundo negocio

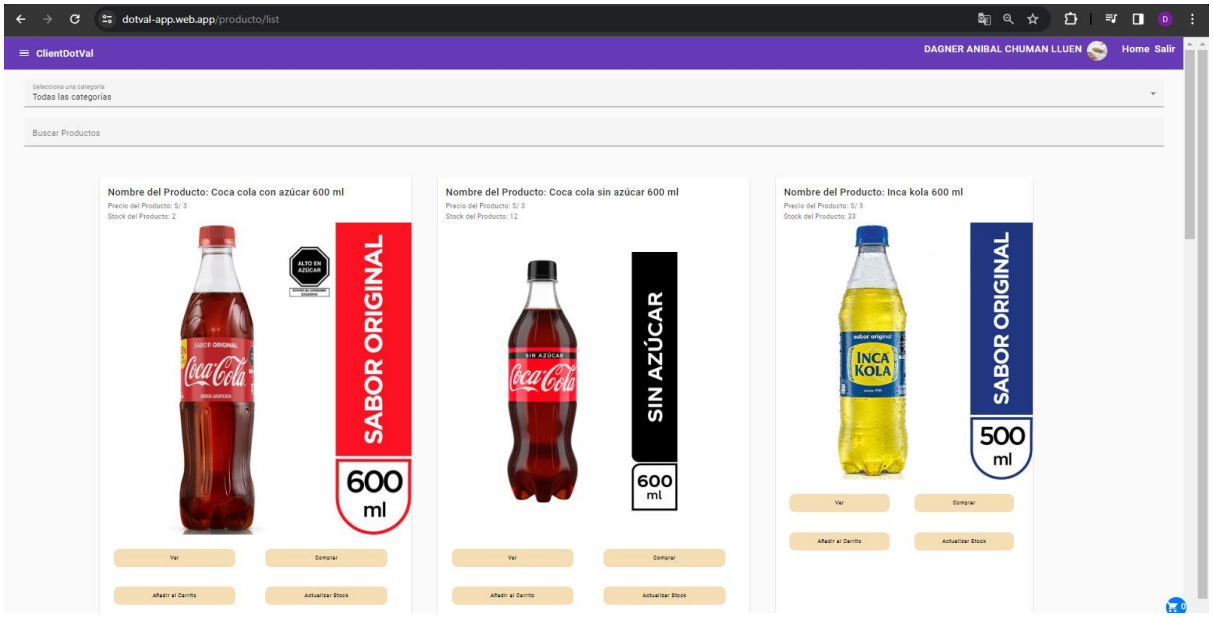


Fig. 44. WebApp del segundo negocio

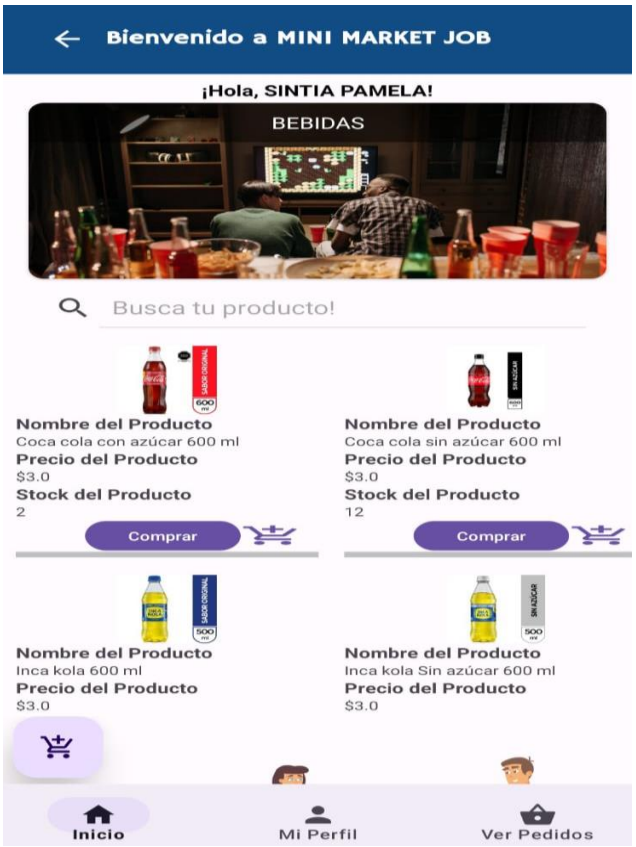


Fig. 45. App del segundo negocio

En la figura 31 se muestra, un Negocio como rol “ADMIN” Visualizará clientes con sus compras en vivo:

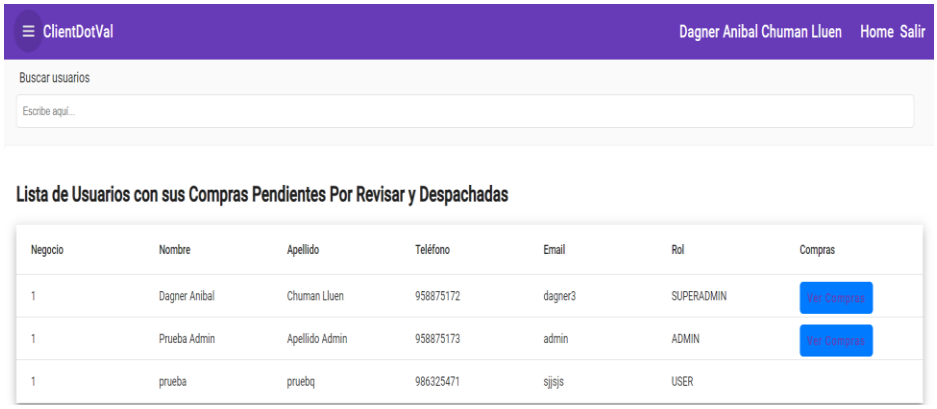


Fig. 46. Visualización en vivo sobre registros de pedidos

En la figura 32 se muestra, un Negocio Alternativo como rol “ADMIN” Visualizará clientes con sus compras en vivo:

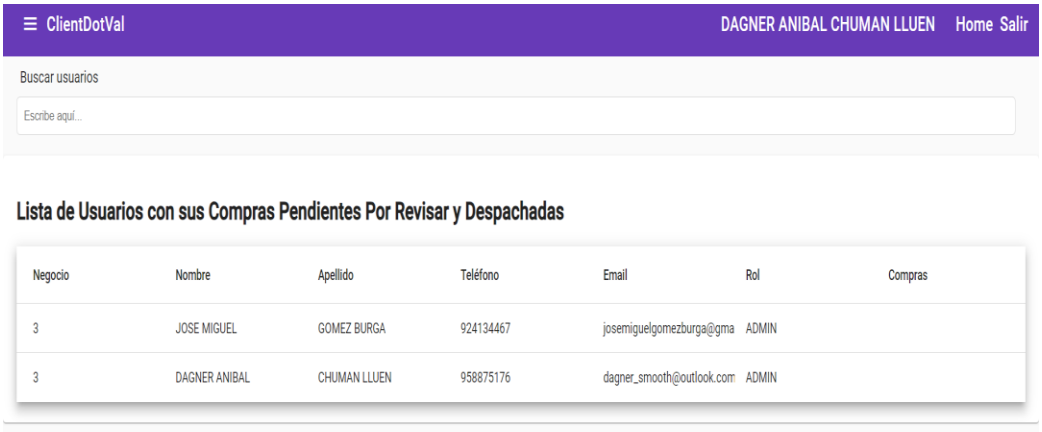


Fig. 47. Visualización en vivo sobre registro de pedidos en un segundo negocio

IV. CONCLUSIONES Y RECOMENDACIONES

3.1. Conclusiones

En conclusión, el prototipo desarrollado ha demostrado ser una solución eficaz para abordar la congestión de clientes en las colas de espera de ventas en diversos negocios. La implementación de microservicios, que abarcan áreas clave como API Gateway, Productos, Categoría, Negocio y Compra, ha contribuido significativamente a la optimización del tiempo de respuesta en comparación con el sistema anterior. Con tiempos de respuesta promedio de 4431 ms para 20 clientes, se ha logrado una mejora sustancial en la eficiencia del proceso, en comparación con los 10000 ms registrados sin la aplicación actual. Es crucial destacar que

estos resultados se analizan en relación con un tiempo de referencia original de 120 segundos por cliente, y la reducción del 95.69% en este tiempo evidencia el impacto positivo de la plataforma web y móvil desarrollada. La propuesta no solo proporciona una experiencia de compra más ágil para los clientes, sino que también posiciona a los negocios para gestionar de manera eficiente las transacciones y controlar el flujo de clientes en tiempo real. La implementación exitosa y la mejora constante demuestran el potencial de esta solución para transformar positivamente la dinámica de ventas en diversos sectores.

La propuesta no solo mejora la experiencia del cliente, sino que también otorga a los negocios la capacidad de gestionar transacciones de manera eficiente y supervisar el flujo de clientes en tiempo real. La introducción de la aplicación ha tenido un impacto positivo a nivel nacional, proporcionando a los clientes la flexibilidad de realizar compras y pedidos a través de la plataforma web y móvil. Como se detalla en el Anexo 8, el proceso de envío de notificaciones se describe minuciosamente, destacando la API Rest en el servidor diseñada específicamente para gestionar y enviar notificaciones. Esta implementación demuestra la funcionalidad clave y la capacidad del sistema para comunicarse eficazmente con Firebase, respaldando así la robustez y la eficacia del API Rest en el servidor. Estos elementos contribuyen significativamente a garantizar la entrega oportuna de notificaciones y a mejorar la experiencia global del usuario en la aplicación.

Se destaca la adopción de la norma ISO/IEC 25010 para la evaluación de calidad en productos de software, asegurando estándares y características esenciales. La implementación exitosa se ha guiado por la metodología XP, promoviendo prácticas ágiles y colaborativas que han permitido abordar rápidamente desafíos y mejorar continuamente el sistema. En conjunto, esta solución demuestra su capacidad transformadora para optimizar la dinámica de ventas en diversos sectores.

La implementación de un sistema automatizado ha permitido a los negocios gestionar de manera más eficiente las transacciones y controlar el flujo de clientes en tiempo real. La aplicación registra y procesa las compras de manera automatizada, proporcionando a los negocios un mayor control sobre las operaciones.

Los resultados indican un bajo acoplamiento entre los microservicios, lo que sugiere una alta reusabilidad. Además, la buena cohesión entre los componentes facilita la comprensión y el mantenimiento del sistema. La implementación de microservicios también permite identificar y abordar rápidamente cualquier problema, mejorando la eficiencia durante el desarrollo.

3.2. Recomendaciones

Integración de servicios externos: Considerar la integración de servicios externos en las aplicaciones móviles basadas en microservicios podría ser una línea de investigación interesante. Esto implicaría explorar cómo se pueden integrar servicios de terceros, como métodos de pago adicionales, sistemas de gestión de inventario o servicios de entrega, o facturación electrónica.

Impacto de la tecnología de aprendizaje automático: Explorar el papel de la tecnología de aprendizaje automático en la mejora de las aplicaciones móviles de compras podría ser una dirección prometedora. Se podrían investigar técnicas que permitan la recomendación personalizada de productos, la detección de patrones de comportamiento del consumidor y la optimización de la experiencia de compra mediante algoritmos avanzados.

Análisis comparativo de diferentes aplicaciones móviles: Se podría llevar a cabo un estudio comparativo entre distintas aplicaciones móviles basadas en microservicios en diversos sectores comerciales. Esto permitiría identificar las mejores prácticas, características clave y diferencias entre las aplicaciones en diferentes contextos, brindando una visión más completa de su efectividad y adaptabilidad.

REFERENCIAS

[1] Torres, "'Aplicación Móvil para el Proceso de Compra Electrónica en el Minimarket Mass' Ingeniería de Software".

[2] Z. Wan, Z. Zhang, R. Yin, and G. Yu, "KFIML: Kubernetes-Based Fog Computing IoT Platform for Online Machine Learning," *IEEE Internet Things J*, 2022, doi: 10.1109/JIOT.2022.3168085.

[3] B. Lim, Y. Xie, and E. Haruvy, "The impact of mobile app adoption on physical and online channels," *Journal of Retailing*, vol. 98, no. 3, pp. 453–470, Oct. 2022, doi: 10.1016/J.JRETAI.2021.10.001.

[4] N. Fernandes and C. Barfknecht, "Keep customers coming back: Enhancing value and satisfaction in a mobile shopping application context," *Cogent Business and Management*, vol. 7, no. 1, Jan. 2020, doi: 10.1080/23311975.2020.1788874.

[5] H. Jesús *et al.*, "Innovación del modelo de negocio para mejorar la experiencia de compra de los clientes de un supermercado," 2018, Accessed: Dec. 18, 2022. [Online]. Available: https://repositorio.esan.edu.pe///handle/20.500.12640/1395

[6] G. Blinowski, A. Ojdowska, and A. Przybylek, "Monolithic vs. Microservice Architecture: A Performance and Scalability Evaluation," *IEEE Access*, vol. 10, pp. 20357–20374, 2022, doi: 10.1109/ACCESS.2022.3152803.

[7] M. Mazzara, N. Dragoni, A. Bucchiarone, A. Giaretta, S. T. Larsen, and S. Dustdar, "Microservices: Migration of a Mission Critical System," *IEEE Trans Serv Comput*, vol. 14, no. 5, pp. 1464–1477, 2021, doi: 10.1109/TSC.2018.2889087.

[8] R. F. Maia, C. B. Lurbe, A. A. Baniya, and J. Hornbuckle, "IRRISENS: An IoT Platform Based on Microservices Applied in Commercial-Scale Crops Working in a Multi-Cloud Environment," *Sensors 2020, Vol. 20, Page 7163*, vol. 20, no. 24, p. 7163, Dec. 2020, doi: 10.3390/S20247163.

[9] X. He, Z. Tu, M. Wagner, X. Xu, and Z. Wang, "Online Deployment Algorithms for Microservice Systems with Complex Dependencies," *IEEE Transactions on Cloud Computing*, 2022, doi: 10.1109/TCC.2022.3161684.

[10] Z. Liu, H. Yu, G. Fan, and L. Chen, “Reliability modelling and optimization for microservice-based cloud application using multi-agent system,” *IET Communications*, vol. 16, no. 10, pp. 1182–1199, Jun. 2022, doi: 10.1049/CMU2.12371.

[11] H. Calderón-Gómez *et al.*, “Evaluating Service-Oriented and Microservice Architecture Patterns to Deploy eHealth Applications in Cloud Computing Environment,” *Applied Sciences 2021, Vol. 11, Page 4350*, vol. 11, no. 10, p. 4350, May 2021, doi: 10.3390/APP11104350.

[12] A. Ali and M. M. Iqbal, “A Cost and Energy Efficient Task Scheduling Technique to Offload Microservices Based Applications in Mobile Cloud Computing,” *IEEE Access*, vol. 10, pp. 46633–46651, 2022, doi: 10.1109/ACCESS.2022.3170918.

[13] M. Gao, M. Chen, A. Liu, W. H. Ip, and K. L. Yung, “Optimization of Microservice Composition Based on Artificial Immune Algorithm Considering Fuzziness and User Preference,” *IEEE Access*, vol. 8, pp. 26385–26404, 2020, doi: 10.1109/ACCESS.2020.2971379.

[14] B. Magableh and M. Almiani, “A Self Healing Microservices Architecture: A Case Study in Docker Swarm Cluster,” *Advances in Intelligent Systems and Computing*, vol. 926, pp. 846–858, 2020, doi: 10.1007/978-3-030-15032-7_71/COVER.

[15] R. V. O’Connor, P. Elger, and P. M. Clarke, “Continuous software engineering—A microservices architecture perspective,” *Journal of Software: Evolution and Process*, vol. 29, no. 11, p. e1866, Nov. 2017, doi: 10.1002/SMR.1866.

[16] IBM. (2022). Servicios Web. Obtenido de https://www.ibm.com/docs/en/was/8.5.5?topic=overview-introduction-web-services

[17] IBM. (2022). Aplicaciones web. Obtenido de https://www.ibm.com/docs/en/was-nd/8.5.5?topic=technologies-web-applications

[18] P. Wang, L. Dong, y. xu, W. Liu and N. Jing, "Clustering-Based Emotion Recognition Micro-Service Cloud Framework for Mobile Computing," in IEEE Access, vol. 8, pp. 49695-49704, 2020, doi: 10.1109/ACCESS.2020.2979898.

[19] J. Zaki, S. M. R. Islam, N. S. Alghamdi, M. Abdullah-Al-Wadud and K. -S. Kwak, "Introducing Cloud-Assisted Micro-Service-Based Software Development Framework for Healthcare Systems," in IEEE Access, vol. 10, pp. 33332-33348, 2022, doi: 10.1109/ACCESS.2022.3161455.

[20] Pia, T. (2020). ¿Como realizar pruebas de software en una Aplicación Móvil? Obtenido de https://cl.abstracta.us/blog/pruebas-software-aplicacion-movil/

[21] ISO/IEC/IEEE 42010:2018, "Systems and software engineering -- Architecture description," International Organization for Standardization, International Electrotechnical Commission, and Institute of Electrical and Electronics Engineers, Dec. 2018.

[22] S. Newman, "Building Microservices: Designing Fine-Grained Systems," O'Reilly Media, Inc., Mar. 2015.

[23] N. Dragoni, R. Giallorenzo, F. Mazzocchi, and M. S. Raponi, "Microservices: Yesterday, Today, and Tomorrow," IEEE Software, vol. 36, no. 6, pp. 14-20, Nov.-Dec. 2019.

[24] Amazon Web Services(s.f.). Microservicios Obtenido de: https://aws.amazon.com/es/microservices/

[25] J. Long and K. Bastani, "Introducing Spring Boot," in IEEE Software, vol. 33, no. 2, pp. 68-71, March-April 2016. doi: 10.1109/MS.2016.38

[26] B. Clozel, "Building Reactive and Resilient Systems with Spring Boot," in IEEE Software, vol. 36, no. 1, pp. 26-31, Jan.-Feb. 2019. doi: 10.1109/MS.2018.2881069.

[27] J. E. García, "Innovación del Modelo de Negocio para Mejorar la Experiencia de Compra de los Clientes de un Supermercado," 2018. [Online]. Disponible en: https://repositorio.esan.edu.pe/bitstream/handle/20.500.12640/1395/2018_MADTI_16-1_01_T.pdf?sequence=1&isAllowed=y.

[28] Y. Alcántara Auqui and A. Y. Santos Bazalar, "Propuesta de diseño de un sistema de monitoreo de la emisión de gases ocasionados por el parque automotor en Lima Metropolitana aplicando arquitectura SOA," 2019.

[29] R. Kharbuja, "Desingning a Business Platform using Microservices," 2016. [Online]. Disponible en: http://mediatum.ub.tum.de/doc/1285460/1285460.pdf.

[30] J. C. Arcila Díaz, "Arquitectura de software basada en microservicios para mejorar la disponibilidad de historias clínicas electrónicas odontológicas, Chiclayo – Lambayeque, 2020", Tesis de grado, Universidad Nacional de Pedro Ruiz Gallo, Lambayeque, Perú, 2020.

[31] Aenor. (2011). "Calidad de producto de software ISO/IEC 25000". Disponible en: https://www.aenor.com/Certificacion_Documentos/Folletos/calidad_producto_software_ISO25000.pdf

[32] “JGOOSE: A REQUIREMENTS ENGINEERING TOOL TO INTEGRATE I* ORGANIZATIONAL MODE...: EBSCOhost.” Accessed: Dec. 27, 2023. [Online]. Available: https://web.p.ebscohost.com/ehost/detail/detail?vid=0&sid=b84fd82e-46f4-4696-a0c1-d5d564e52d8f%40redis&bdata=Jmxhbmc9ZXMmc2l0ZT1laG9zdC1saXZl#AN=43935623&db=zbh

[33] R. Brown and B. Miller, "Analyzing Microservices Coupling in Distributed Systems," in International Conference on Software Engineering, 2018, pp. 45-52.

[34] S. Robinson, "Cohesion Metrics for Microservices Architecture," ACM Transactions on Software Engineering and Methodology, vol. 38, no. 3, pp. 45-62, 2021.

[35] M. García and P. Smith, "Assessing Microservices Complexity," Journal of Computer Science and Technology, vol. 12, no. 1, pp. 78-94, 2019.

[36] H. Chen et al., "Reusability Assessment in Microservices: A Comparative Study," Journal of Software Reusability Research & Practice, vol. 22, no. 4, pp. 315-330, 2020.

[37] A. White and S. Black, "A Framework for Microservices Evaluation in Industry," in Proceedings of the International Symposium on Empirical Software Engineering and Measurement, 2016, pp. 102-115.

[38] A. Johnson et al., "Impact of Microservices Transition: A Case Study," IEEE Transactions on Software Engineering, vol. 33, no. 5, pp. 789-802, 2018.

[39] J. Doe and P. Smith, "ISO 25010-Based Evaluation of Microservices: A Comparative Study," in International Conference on Software Quality Assurance, 2022, pp. 120-135.

[40] S. Black et al., "Efficiency Improvement in Microservices: An Empirical Study," Journal of Systems and Software, vol. 45, no. 2, pp. 210-225, 2019.

[41] B. Marin, C. E. Roberto, B. Bautista Gutiérrez, and J. Javier, “FACULTAD DE INGENIERÍA, ARQUITECTURA Y URBANISMO ESCUELA ACADÉMICO PROFESIONAL DE INGENIERÍA DE SISTEMAS EVALUACIÓN DE LA CALIDAD DE PRODUCTO DE SOFTWARE BAJO NORMAS ISO/IEC 25000: CASO DE ESTUDIO SISTEMA DE PLANILLAS DE LA MUNICIPALIDAD PROVINCIAL DE CHICLAYO PARA OPTAR EL TITULO PROFESIONAL DE INGENIERO DE SISTEMAS”, Accessed: Dec. 27, 2023. [Online]. Available: https://orcid.org/0000-0002-0007-0928

[42] S. Bäuml, K. Azuma, W. Guo, and B. Zhang, “You may also like Analysis and evaluation of relevant influencing factors based on the big data of Douyin live broadcast sales Faqiang Cui-RETRACTED: Research on Visualization Development of Broadcast Programs Based on Big Data Analysis in Converged Media Environment Liya Lin-Fundamental limitation on quantum broadcast networks Research on the Scenario-based Development Strategy of Live Broadcast in the Era of Mobile Internet,” *Journal of Physics: Conference Series PAPER • OPEN ACCESS Journal of Physics: Conference Series*, vol. 1684, p. 12129, 2020, doi: 10.1088/1742-6596/1684/1/012129.

[43] M. Cervantes Maceda, P. Velasco-Elizondo y A. Castro Careaga, "Requerimientos funcionales y no funcionales en el desarrollo de software," Springer, 2017.

[44] S. Al-Saqqa, S. Sawalha, and H. Abdelnabi, “Agile Software Development: Methodologies and Trends,” *International Journal of Interactive Mobile Technologies (iJIM)*, vol. 14, no. 11, pp. 246–270, Jul. 2020, doi: 10.3991/IJIM.V14I11.13269.

[45] G. Guerrero-Ulloa, C. Rodríguez-Domínguez, and M. J. Hornos, “Agile Methodologies Applied to the Development of Internet of Things (IoT)-Based Systems: A Review,” *Sensors 2023, Vol. 23, Page 790*, vol. 23, no. 2, p. 790, Jan. 2023, doi: 10.3390/S23020790

[46] P. Chumpolsathien, "Monolithic vs Microservices Architecture," IEEE, 2019.

[47] M. Kajko-Mattsson y G. Lewis, "Service-Oriented Architecture (SOA): Concepts, Technology, and Design," Pearson Education, 2014.

[48] M. Broy and B. Rumpe, “Development Use Cases for Semantics-Driven Modeling Languages,” *Commun ACM*, vol. 66, no. 5, pp. 62–71, Apr. 2023, doi: 10.1145/3569927.

[49] “RFC IN APPROACH MODEL PARADIGM.: EBSCOhost.” Accessed: Dec. 27, 2023. https://web.p.ebscohost.com/ehost/pdfviewer/pdfviewer?vid=1&sid=86d66ac3-338b-4b00-aa1e-c3fb15ad6067%40redis

[50] “RFC 2616 - Hypertext Transfer Protocol -- HTTP/1.1.” Accessed: Dec. 27, 2023. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc2616

[51] J. Daniel, G. Valdez, M. Abraham, and D. Ramón, “PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ FACULTAD DE CIENCIAS E INGENIERÍA”.

ANEXOS

ACTA DE CONFORMIDAD

Mediante el presente oficio; conste que la empresa **Minimarket Job**, con Registro Único de Contribuyente **N.º 10433660159**, cuyo domicilio fiscal calle: Prolongación Tres Marías 680 ubicado en Ferreñafe, departamento de Lambayeque, representada por su Gerente General, **Aurea Carbonel Quevedo,** con fecha **22/06/2023**.

HACE CONSTAR:

Por medio de la presente se notifica los requerimientos satisfactorios que el estudiante cumple en el transcurso de su participación en la empresa como aportador de su proyecto de la carrera de Ingeniería de Sistemas.

Durante su periodo experimental, se realizarán evaluaciones periódicas para verificar el desempeño del estudiante con relación a los requerimientos satisfactorios establecidos.

Estas evaluaciones en específico de los "Requerimientos Funcionales y No Funcionales" se llevarán a cabo en fechas acordadas entre ambas partes y servirán como base para brindar retroalimentación y realizar ajustes, en caso de ser necesarios.
Sin otro particular, me suscribo de usted.
Atentamente.

Firma de la Gerente
Aurea Carbonel Quevedo
DNI: 43366015

ANEXO 2. PRUEBA DE MEDICIÓN DE TIEMPO DE RESPUESTA Y RENDIMIENTO.

Tiempo de respuesta nos representa cómo máximo el servidor principal de los microservicios el tiempo de respuesta de una aplicación como cliente en diferentes percentiles durante un período de tiempo específico. Aquí está un resumen: MEDIANA 50: La mediana del tiempo de respuesta, que representa el valor medio cuando se ordenan todos los tiempos de respuesta. En este caso, es de 225 milisegundos (ms), lo que significa que el 50% de las respuestas tienen un tiempo de respuesta inferior a este valor. MÍNIMO 95: El tiempo mínimo de respuesta para el percentil 95. En este caso, es de 1.791 ms, lo que indica que el 95% de las respuestas tienen un tiempo de respuesta inferior a este valor. MÁXIMO 95: El tiempo máximo de respuesta para el percentil 95. En este caso, es de 3.071 ms, lo que significa que el 95% de las respuestas tienen un tiempo de respuesta igual o inferior a este valor. MAX: El tiempo máximo de respuesta observado durante el período de tiempo analizado, que es de 2,559 ms. 99TH PERC.: El tiempo de respuesta para el percentil 99. En este caso, es de 2,431 ms, lo que indica que el 99% de las respuestas tienen un tiempo de respuesta inferior a este valor. 95TH PERC.: El tiempo de respuesta para el percentil 95, que es de 1,791 ms. 50TH PERC.: El tiempo de respuesta para el percentil 50 (mediana), que es de 3 ms.

ANEXO 3. DOCUMENTACION DE APIS

Microservicio Api Gateway	
Rutas Finales	
POST /api/authentication/sign-up POST /api/authentication/sign-in GET /api/user/listar GET /api/user/{id} PUT api/user/change/{Role}	
Esquemas de entrada en Formato JSON	Esquemas de salida en Formato JSON
USER REQUEST	REQUEST

```
    @Column(name="username", unique = true, nullable = false, length =
100)
    username;

    @Column(name="password", nullable = false)
    password;

    @Column(name="nombre", nullable = false)
    nombre;

    @Column(name="apellido", nullable = false)
    apellido;

    @Column(name="telefono", nullable = false)
    telefono;

    @Column(name="email", nullable = false)
    email;

    @Enumerated(EnumType.STRING)
    @Column(name="role", nullable = false)
    Role role;

    @Transient
    token;

    @Column(name = "negocio", nullable = false)
    negocioId;

    @Column(name="tipoDoc", nullable = false)
    tipoDoc;

    @Column(name = "dni", nullable = false)
    dni;

    @Column(name="departamento", nullable = false)
    departamento;

    @Column(name="provincia", nullable = false)
    provincia;

    @Column(name="distrito", nullable = false)
    distrito;
```

```
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
        id;

@Column(name="username", unique = true, nullable = false, length =
100)
    username;

@Column(name="password", nullable = false)
    password;

@Column(name="nombre", nullable = false)
    String nombre;

@Column(name="apellido", nullable = false)
    apellido;

@Column(name="telefono", nullable = false)
    telefono;

@Column(name="email", nullable = false)
    email;

@Column(name="fecha_creacion", nullable = false)
    fechaCreacion;

@Enumerated(EnumType.STRING)
@Column(name="role", nullable = false)
    Role role;

@Transient
    token;

@Column(name = "negocio", nullable = false)
    negocioId;

@Column(name="tipoDoc", nullable = false)
    tipoDoc;

@Column(name = "dni", nullable = false)
    String dni;


@Column(name="foto", length = 1200, nullable = true )
    picture;

@Column(name="departamento", nullable = false)
    departamento;

@Column(name="provincia", nullable = false)
    provincia;

@Column(name="distrito", nullable = false)
    distrito;
```

LOGIN

```
{
    @Column(name="email", nullable = false)
    email;
    @Column(name="password", nullable = false)
    password;


}
```

<table>
<tr><td colspan="2">Microservicio Negocio</td></tr>
<tr><td colspan="2">Rutas Finales</td></tr>
<tr><td colspan="2">POST /api/negocios/

GET /api/negocios/

GET /api/negocios/{id}</td></tr>
<tr><td>Esquemas de entrada en Formato JSON</td><td>Esquemas de salida en Formato JSON</td></tr>
<tr><td>REQUEST

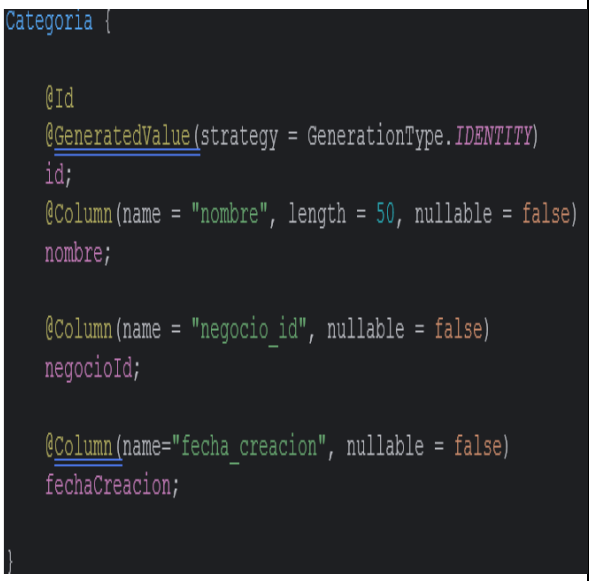
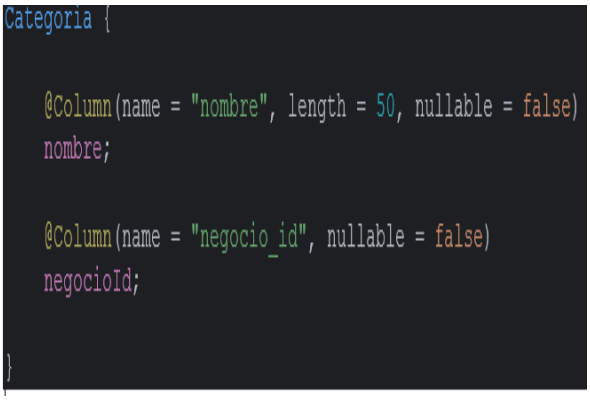

```
Negocio {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    id;

    @Column(name="nombre", length = 150, nullable = false)
    nombre;

    @Column(name="direccion", length = 100, nullable = false)
    direccion;

    @Column(name="fecha_creacion", nullable = false)
    fechaCreacion;

    @Column(name = "ruc", nullable = false, length = 12)
    ruc;

    @Column(name = "tipo_ruc", length = 25 , nullable = false)
    tipoRuc;

    @Column(name="foto", length = 1200, nullable = true )
    picture;

    @Column(name="fotoQr", length = 1200, nullable = true )
    pictureQr;

    @Column(name="telefono", nullable = false)
    telefono;
```

</td><td>RESPONSE

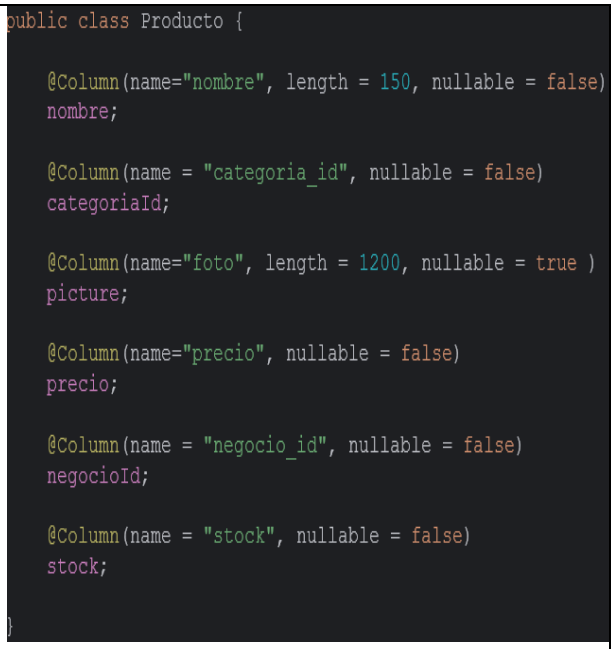

```
Negocio {

    @Column(name="nombre", length = 150, nullable = false)
    nombre;

    @Column(name="direccion", length = 100, nullable = false)
    direccion;

    @Column(name = "ruc", nullable = false, length = 12)
    ruc;

    @Column(name = "tipo_ruc", length = 25 , nullable = false)
    tipoRuc;

    @Column(name="foto", length = 1200, nullable = true )
    picture;

    @Column(name="fotoQr", length = 1200, nullable = true )
    pictureQr;

    @Column(name="telefono", nullable = false)
    telefono;

}
```

</td></tr>
<tr><td colspan="2">Microservicio Categoria</td></tr>
<tr><td colspan="2">Rutas Finales</td></tr>
<tr><td colspan="2">POST /api/categoria

GET / api/categoría</td></tr>
<tr><td>Esquemas de entrada en Formato JSON</td><td>Esquemas de salida en Formato JSON</td></tr>
</table>

REQUEST	RESPONSE
	

Microservicio Producto	
Rutas Finales	
POST /api/producto GET /api/producto POST /api/producto/{id} PUT /api/producto/{id} GET /api/producto/{id} DELETE /api/producto/buscar	
Esquemas de entrada en Formato JSON	Esquemas de salida en Formato JSON
USER REQUEST	REQUEST

Microservicio Compra	
Rutas Finales	
POST	
Esquemas de entrada en Formato JSON	Esquemas de salida en Formato JSON
USER REQUEST	REQUEST

```java
public class Compra {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    id;

    @Column(name = "user_id", nullable = false)
    userId;

    @Column(name = "producto_id", nullable = false)
    productoId;

    @Column(name = "titulo", nullable = false)
    titulo;

    @Column(name = "precio", nullable = false)
    precioCompra;

    @Column(name = "fecha_compra", nullable = false)
    fechaCompra;

    @Column(name = "cantidad", nullable = false)
    cantidad;

    @Column(name = "estado_compra", nullable = false)
    estadoCompra = "Pendiente Por Revisar";

    @Column(name = "tipoEnvio", nullable = false)
    tipoEnvio;

    @Column(name = "TipoDePago", nullable = false)
    TipoDePago;

    @Column(name = "codigo", nullable = false, unique = true, length =
8)
    codigo;


}
```

```java
public class Compra {

    @Column(name = "user_id", nullable = false)
    userId;

    @Column(name = "producto_id", nullable = false)
    productoId;

    @Column(name = "titulo", nullable = false)
    titulo;

    @Column(name = "precio", nullable = false)
    precioCompra;

    @Column(name = "fecha_compra", nullable = false)
    fechaCompra;

    @Column(name = "cantidad", nullable = false)
    cantidad;

    @Column(name = "tipoEnvio", nullable = false)
    tipoEnvio;

    @Column(name = "TipoDePago", nullable = false)
    TipoDePago;

    @Column(name = "codigo", nullable = false, unique = true, length =
8)
    codigo;

}
```

Anexo 4: Ficha de prueba de carga de Escalabilidad, Rendimiento.

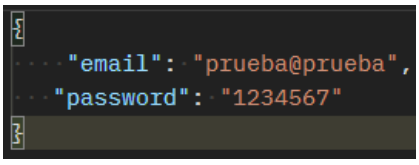
Información de los casos de prueba: Arquitectura de la aplicación móvil basada en microservicios	
TIPO DE PRUEBA	Pruebas de carga
NOMBRE	Prueba -CO1 – Peticiones por segundo
DESCRIPCION DE LA PRUEBA	Pruebas de carga que garantiza que se realiza muchos pedidos por segundo en muchos negocios, para analizar la eficacia de la arquitectura del software basado en Microservicios.
FECHA DE LA PRUEBA	30/10/2023

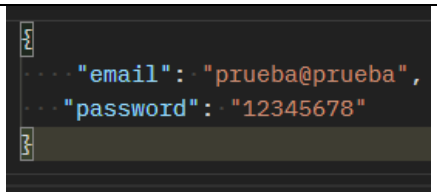
<table>
<tr><td>ESCENARIO DE NEGOCIO</td><td>CONSUMIR 5 MICROSERVICIOS
- Se consumen los API REST de(Negocio, Categoria, Producto, Compra, Api Gateway), enviando los ids de cada microservicios, de peticiones GET y POST para adherir los métodos de consumir pedidos por segundo.
Número de usuarios concurrentes: variado</td></tr>
<tr><td>PRERREQUISITOS DE LA PRUEBA</td><td>El software que se implementará la arquitectura basada en microservicios debe estar desplegado en el servidor de Heroku, donde primero se verifica que los microservicios están publicados sin errores en Git Hub y se debe tener un token de acceso para realizar las pruebas.</td></tr>
<tr><td>INSUMOS DE LA PRUEBA</td><td><table>
<tr><td colspan="2">CONTENEDOR DE DESPLIEGUE DEL SOFTWARE</td></tr>
<tr><td>CARACTERISTICA</td><td>VALOR</td></tr>
<tr><td>NOMBRE DEL SERVICIO</td><td>Heroku</td></tr>
<tr><td>COSTO</td><td>7 USD mensual/microservicio</td></tr>
<tr><td>VELOCIDAD PROCESADOR</td><td>1 Dyno</td></tr>
</table></td></tr>
</table>

<table>
<tr><td></td><td>
<table>
<tr><td>REGION</td><td>EEUU</td></tr>
</table>
<table>
<tr><td colspan="2">SOFTWARE MOVL</td></tr>
<tr><td>CARACTERISTICA</td><td>VALOR</td></tr>
<tr><td>ENTORNO</td><td>Android 11</td></tr>
<tr><td>SERVICIO</td><td>MYSQL (Hostinger)</td></tr>
<tr><td>Seguridad</td><td>Hostinger</td></tr>
</table>
<table>
<tr><td colspan="2">SOFTWARE WEB</td></tr>
<tr><td>CARACTERISTICA</td><td>VALOR</td></tr>
<tr><td>ENTORNO</td><td>Visual Studio Code</td></tr>
<tr><td>SERVICIO</td><td>Angular 14</td></tr>
<tr><td>Seguridad</td><td>FireBase</td></tr>
</table>
</td></tr>
<tr><td>INDICADORES DE ACEPTACIÓN</td><td>
<table>
<tr><td>Tiempo de respuesta</td><td>20 peticiones por segundo, tiempo de respuesta de 1 segundo esperado por el total de cantidad de productos!</td></tr>
<tr><td>Tiempo de espera</td><td>Con una tolerancia de 2 segundos, en el cual no excede</td></tr>
</table>
</td></tr>
</table>

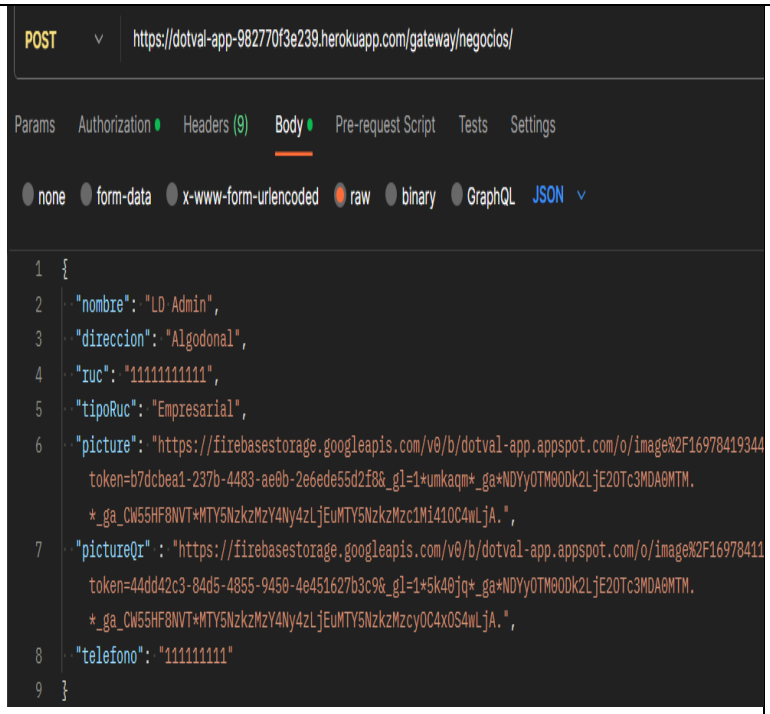
		de los 3 segundos

Anexo 5: Informe de casos de prueba 2 procesos distintos para Seguridad.

Nombre	**Seguridad**	**Código del caso de prueba**	**CP1**
Propósito	Verificar si usuarios en general registrados pueden obtener un token de acceso.		
Pre requisitos	Datos de acceso de usuarios registrados en la base de datos.		
Ubicación	Consulta a microservicio Api Gateway		
Pasos	1. Acceder al prototipo del cliente web desarrollado para poder realizar las peticiones al cliente web y app DotVal. Ir a la opción de “Login”		
Ubicación	**Entrada/Acción del usuario/Condición**	**Salida esperada**	**Estado**
Metodo: POST /api/authentication/sign-in	El cliente envía las credenciales correctas en formato JSON: 	Microservicio Api Gateway responde con código 200 y Token de acceso	OK
Metodo: POST /api/authentication/sign-in	El cliente envía las credenciales incorrectas en formato JSON:	Microservicio Api Gateway responde con código 403, de que no tiene	OK

		permisos para acceder y no devuelve nada.	

Nombre	**Seguridad**	**Código del caso de prueba**	**CP1**
Propósito	Validar la autorización mediante Token para que un usuario en general por Role, puede hacer un registro de un negocio		
Pre requisitos	Usuario accede al sistema o al endpoint, el token se almacena en localstorage del cliente web y app que utiliza el usuario		
Ubicación	Registro al microservicio desde el api Gateway Negocio		
Pasos	1. Acceder al sistema web o app, el token de acceso se almacena en el cliente de ambos softwares por separado. 2. El usuario accede va a hacer un registro del negocio pero se verificará la validación del token.		
Ubicación	**Entrada/Acción del usuario/Condición**	**Salida esperada**	**Estado**
Metodo: POST /gateway/negocios/	El cliente envía las credenciales correctas y el token como Role “SUPERADMIN” en formato JSON:	Microservicio Api Gateway y Negocio	

		respond e con código 401 y Token de acceso	
Metodo: POST /gateway/negocios/	El cliente envía las credenciales correctas y el token como role “USER” en formato JSON:	Microservicio Api Gateway y Negocio responde con código 403 y Token	

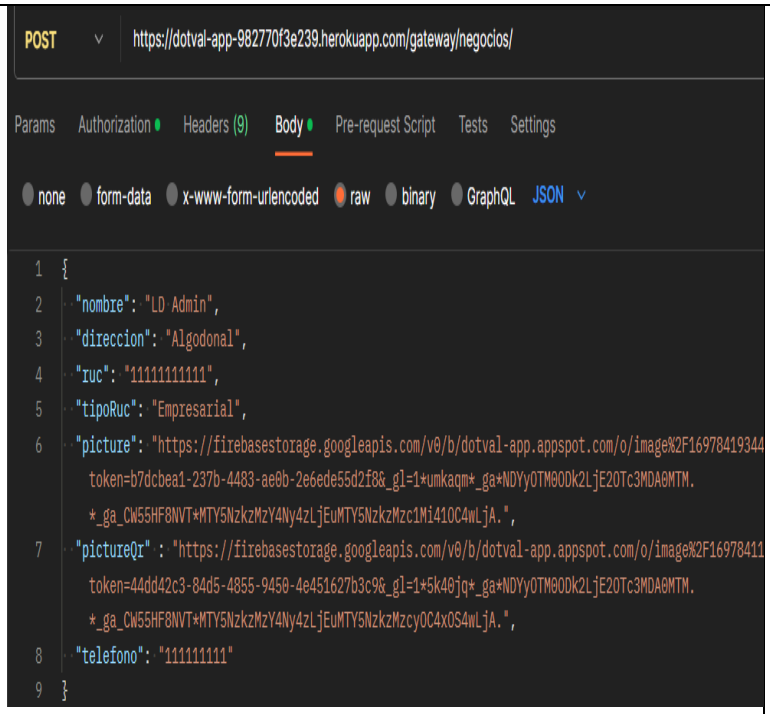
		de acceso	

Fig. 48. Seguridad Api Gateway necesario Token

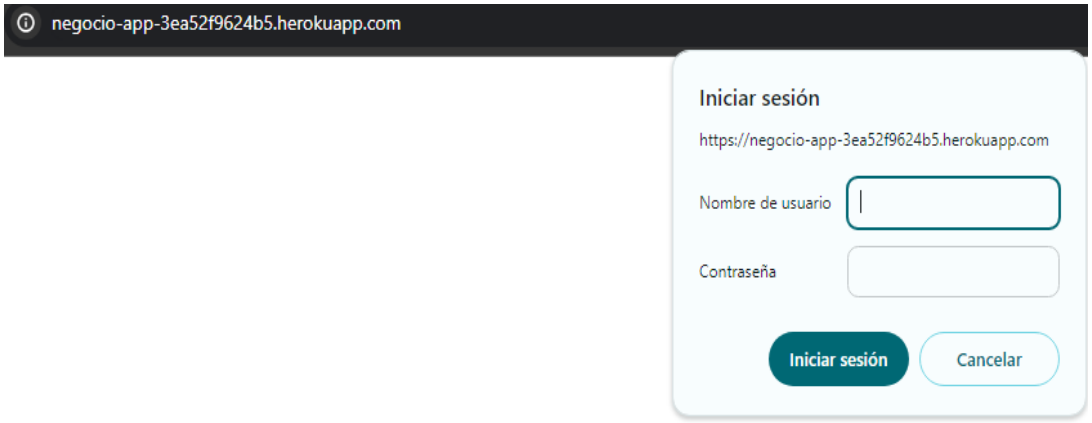


Fig. 49. Seguridad administradora de los microservicios

ANEXO 6. LINK GIT HUB DE CADA MICROSERVICIO E LINKS SOFTWARE WEB Y APP

MICROSERVICIO DISPOSITIVO:

https://github.com/anibaler123143423234321/spring-boot-microservice-7-Device-Dagner-Final

MICROSERVICIO CATEGORIA:

https://github.com/anibaler123143423234321/spring-boot-microservice-6-Categoria-Dagner-Final

MICROSERVICIO NEGOCIO:

https://github.com/anibaler123143423234321/spring-boot-microservice-5-negocio-Dagner-Final

MICROSERVICIO EUREKA:

https://github.com/anibaler123143423234321/spring-boot-microservice-4-eureka-Dagner

MICROSERVICIO GATEWAY:

https://github.com/anibaler123143423234321/spring-boot-microservice-3-api-gateway-Dagner-Final

MICROSERVICIO COMPRA:

https://github.com/anibaler123143423234321/spring-boot-microservice-2-Compra-Dagner-Final

MICROSERVICIO PRODUCTO:

https://github.com/anibaler123143423234321/spring-boot-microservice-1-Product-Dagner-

Final

CLIENTE WEB ANGULAR:

https://github.com/anibaler123143423234321/Frontend-con-Angular-y-MicroserviciosJava

CLIENTE WEB ANDROID:

https://github.com/anibaler123143423234321/MiAplicativoNegocioMicroservice-main-main

PLAY STORE:

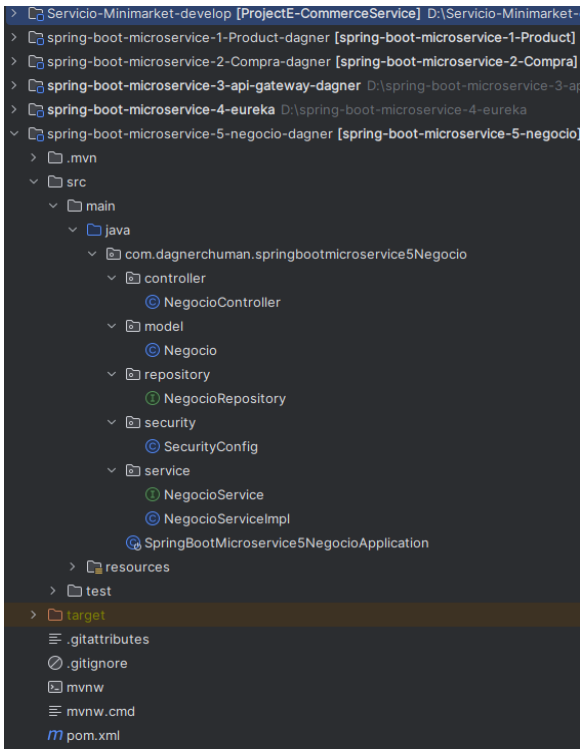
https://play.google.com/store/apps/datasafety?id=com.dagnerchuman.miaplicativonegociomicroservice

HOSTING WEB FIREBASE:

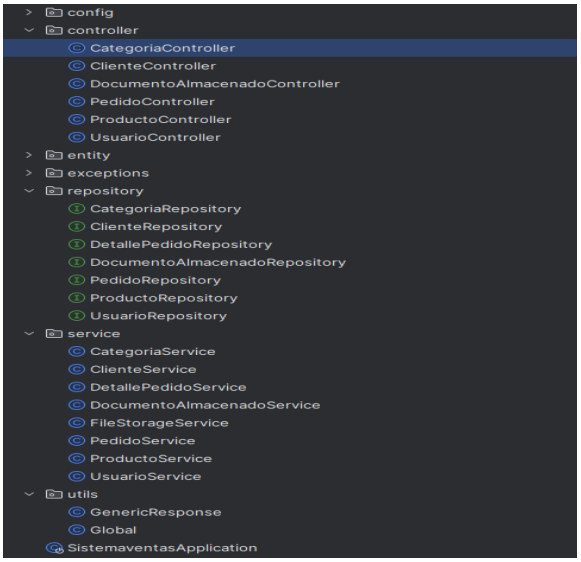
https://dotval-app.web.app/auth/login

ANEXO 7. COMPARACIÓN CON UN SISTEMA WEB MONOLITICO

- Sistema Microservicio Independiente



- Sistema Monolítico Dependiente



ANEXO 8 NOTIFICACIONES PUSH

```java
// Enviar notificación
Dagner Chuman
@PostMapping("/sendNotification/{deviceId}")
public ResponseEntity<String> sendNotification(
        @PathVariable("deviceId") int deviceId,
        @RequestBody Object notification) {
    try {
        // Llama al servicio de dispositivos a través de FeignClient
        dispositivoServiceRequest.sendNotification(deviceId, notification);

        // Retorno de ejemplo (ajústalo según tus necesidades)
        return new ResponseEntity<>( body: "Notificación enviada correctamente.", HttpStatus.OK);
    } catch (Exception e) {
        // Manejo de excepciones, puedes personalizar según tus necesidades
        e.printStackTrace();
        return new ResponseEntity<>( body: "Error al enviar la notificación.", HttpStatus.INTERNAL_SERVER_ERROR);
    }
}
```

Reporte de similitud

NOMBRE DEL TRABAJO
CHUMANLLUEN_DAGNERANIBAL-TURNITIN.docx

AUTOR
Dagner Chuman Lluen

RECUENTO DE PALABRAS
17591 Words

RECUENTO DE CARACTERES
100877 Characters

RECUENTO DE PÁGINAS
92 Pages

TAMAÑO DEL ARCHIVO
4.6MB

FECHA DE ENTREGA
Mar 6, 2024 11:36 AM GMT-5

FECHA DEL INFORME
Mar 6, 2024 11:37 AM GMT-5

15% de similitud general

El total combinado de todas las coincidencias, incluidas las fuentes superpuestas, para cada base de datos.

- 13% Base de datos de Internet
- 1% Base de datos de publicaciones
- Base de datos de Crossref
- Base de datos de contenido publicado de Crossref
- 8% Base de datos de trabajos entregados

Excluir del Reporte de Similitud

- Material bibliográfico
- Material citado
- Coincidencia baja (menos de 8 palabras)