UNIVERSIDAD POLITÉCNICA SALESIANA ECUADOR

POSGRADOS

Maestría en
Software con mención en Diseño de Arquitectura de Sistemas

RPC-SO-34-NO.778-2021

Opción de Titulación:

Proyecto de titulación con componentes de investigación aplicada y/o de desarrollo

Tema:

Construcción de una plataforma de comercio electrónico basado en microservicios para la gestión, entrega y rastreo de productos a domicilio

Autor:

Jonathan Paúl Velasco Navas

Director:

Daniel Giovanny Díaz Ortiz

Quito – Ecuador

2024

AUTOR:

JONATHAN PAÚL VELASCO NAVAS
Ingeniero de Sistemas
Candidato a Magister en Software, mención en Diseño de Arquitectura de Sistemas. Por la Universidad Politécnica Salesiana – Sede Quito
jvelascon@est.ups.edu.ec

DIRIGIDO POR:

DANIEL GIOVANNY DÍAZ ORTIZ
Ingeniero de Sistemas
Master Universitario en Software Libre
Magister en Redes de Comunicaciones
ddiaz@ups.edu.ec

JONATHAN PAÚL VELASCO NAVAS

CONSTRUCCIÓN DE UNA PLATAFORMA DE COMERCIO ELECTRÓNICO BASADO EN MICROSERVICIOS PARA LA GESTIÓN, ENTREGA Y RASTREO DE PRODUCTOS A DOMICILIO

AGRADECIMIENTOS

Hoy vuelvo a caminar queriendo encontrar
alguna señal en el tiempo.

Hoy vuelvo a soñar o a imaginar
alguna señal todo el tiempo.

DEDICATORIA

WOS - Final Internacional 2018
Minuto 1:52 – 3:10

Tabla de Contenido

Índice de Figuras

Índice de Tablas

Construcción de una plataforma de comercio electrónico basado en microservicios para la gestión, entrega y rastreo de productos a domicilio.

Autor:

JONATHAN PAÚL VELASCO NAVAS

Resumen

Tradicionalmente, los sistemas informáticos se solían compilar en un único archivo que albergaba todos los elementos de una aplicación. No obstante, a medida que estos sistemas evolucionan y crecen, estas arquitecturas monolíticas enfrentan desafíos relacionados con el acoplamiento por la alta dependencia entre sus componentes. Para abordar estos problemas inherentes a la arquitectura monolítica, surge la idea de la utilizar la arquitectura basada en microservicios, misma que está constituida por una serie de pequeños programas independientes que se comunican unos con otros a través de canales de mensajería o API's.

El propósito de este proyecto es detallar el proceso de desarrollo de una plataforma de comercio electrónico basado en microservicios para la gestión entrega y rastreo de productos a domicilio, cuyo enfoque principal se centra en el diseño arquitectónico de microservicios con la propuesta de automatizar la venta, la gestión y el despacho de pedidos a los clientes de la empresa PYME «Atlas».

Se llevó a cabo una investigación exhaustiva sobre los diversos estilos arquitectónicos ampliamente empleados, como el Monolítico, REST y Microservicios, analizando su evolución a lo largo del tiempo y subrayando las virtudes y desafíos asociados a cada uno de estos estilos arquitectónicos. Gracias a este análisis, se optó por la arquitectura de microservicios, que se distingue por su alta cohesión al no depender de otros microservicios y por la independencia de sus componentes.

Para el desarrollo de los microservicios se utilizaron tecnologías vinculadas al lenguaje de Python, como Django y Django REST Framework y la comunicación entre los microservicios se ha realizado utilizando Apache Kafka. El Frontend del sistema web se construyó con las templates propias de Django, mientras que para el sistema móvil se trabajó con el IDE de Android Studio utilizando el lenguaje de Java.

Finalmente, se logró construir una plataforma de comercio electrónico de manera operativa, donde se llevaron a cabo pruebas de funcionalidad para asegurar que se cumplieran los requisitos tanto en el sistema web como en el sistema móvil, y los resultados obtenidos fueron conformes a lo esperado.

Palabras clave: Arquitectura monolítica, REST, microservicios, API, PYME, acoplamiento, escalabilidad, apache Kafka, sistema web, sistema móvil.

ABSTRACT

Traditionally, computer systems used to be compiled into a single file that housed all the elements of an application. However, as these systems evolve and grow, these monolithic architectures face challenges related to coupling due to the high dependency among their components. To address these inherent issues with monolithic architecture, the idea of using microservices-based architecture emerges, which is comprised of a series of small independent programs that communicate with each other through messaging channels or APIs.

The purpose of this project is to detail the development process of an e-commerce platform based on microservices for the management, delivery, and tracking of home delivery products, with a primary focus on the architectural design of microservices with the proposal to automate the sale, management, and dispatch of orders to the customers of the SME "Atlas" company.

A comprehensive investigation was conducted on various widely used architectural styles, such as Monolithic, REST, and Microservices, analyzing their evolution over time and highlighting the advantages and disadvantages associated with each of them. Thanks to this analysis, the decision was made to use the microservices architecture, which is characterized by its high cohesion as it does not depend on other microservices and the independence of its components.

For the development of the microservices, technologies related to the Python language were used, such as Django and Django REST Framework, and communication between the microservices was carried out using Apache Kafka. The frontend of the web system was built using Django's own templates, while the mobile system was developed using Android Studio IDE with the Java language.

Finally, an operational e-commerce platform was successfully built, where functionality tests were conducted to ensure that the requirements were met both in the web system and the mobile system, and the results obtained were as expected.

Keywords: Monolithic architecture, REST, microservices, API, SME, coupling, scalability, Apache Kafka, web system, mobile system.

1. Introducción

1.1. Antecedentes

El comercio es esencial en la sociedad debido a que permite el intercambio de bienes o servicios por dinero u otros bienes de igual valor, además de establecer relaciones y llegar a acuerdos por un bien o servicio adquirido. Juan González en su artículo denominado «El e-commerce: sus orígenes, presente y tendencias futuras», expresa que el surgimiento del internet específicamente de la World Wide Web en el año de 1989, es cuando se produce una evolución en el comercio, pasando de un catálogo de productos impreso a la creación de portales exclusivos para la venta de productos de distintas categorías, como es el caso de eBay y Amazon que hoy en día siguen en funcionamiento. De igual manera el comercio electrónico o e-commerce, puede ser descrito como toda actividad empresarial que implica el uso de equipos tecnológicos fijos o móviles para transacciones de compra y venta de productos o adquisición de servicios utilizando internet. En la actualidad el comercio electrónico es de gran importancia, porque los hábitos de compra han cambiado, y cada vez más personas prefieren comprar en línea en lugar de ir a una tienda física. Bajo este contexto los negocios y empresas deben posicionarse de forma efectiva ante sus clientes, razón por la cual las empresas con diferentes actividades económicas están integrando el comercio electrónico a sus operaciones de negocio, como una propuesta innovadora para la atracción de más clientes a sus negocios (González, 2016).

Nuria Gamella, en su artículo descrito como: «Qué es un e-commerce: tipos de negocios y pasos para crearlo», expone que las ventas globales en línea han experimentado un rápido crecimiento durante la última década. Esta tendencia se ha intensificado desde el año 2020, impulsada en gran parte por la pandemia de coronavirus COVID-19. Se espera que las ventas globales en el espacio de comercio electrónico alcancen los 7390 millones de dólares para 2025. En Ecuador, el comercio electrónico no estaba bien posicionado hasta antes del año 2020, pues existían factores que no convencían a los ecuatorianos a comprar en línea, como: la costumbre de comprar en forma presencial, no contar con tarjetas de crédito y/o débito para efectuar pagos, desconocimiento de los beneficios del comercio electrónico, etc. A inicios del año 2021, la respuesta del comercio electrónico en Ecuador no fue diferente al resto del mundo (Gamella, 2022).

Como Zambrano Bernardo, Castellanos Esther y Miranda Mario, sostienen en su artículo para la Cámara Ecuatoriana de Comercio Electrónico, las compras realizadas a través de canales digitales o sitios web se han multiplicado al menos 15 veces desde que comenzó el distanciamiento social a causa de la pandemia por COVID-19. Este crecimiento demuestra que los ecuatorianos están dispuestos a utilizar este tipo de plataformas para compras digitales. Las personas compran electrónicamente con múltiples intereses y motivaciones, es ahí donde las empresas deben enfocarse para atraer el público hacia sus tiendas digitales, donde las personas buscan productos de calidad a precios accesibles, pagar de forma segura y recibir sus pedidos inmediatamente (Zambrano, Castellanos & Miranda, 2021).

En cuanto a las tecnologías utilizadas para el desarrollo de sistemas informáticos existen varios estilos arquitectónicos, los cuales con el tiempo y las necesidades del negocio están en constante evolución debido a la naturaleza propia del software, que es adaptarse a los cambios de forma oportuna. Oscar Blancarte, en su libro «Introducción a la Arquitectura de Software», señala que los estilos arquitectónicos no determinan la tecnología en la que se construye el software, por el contrario un estilo arquitectónico especifica ciertos lineamientos y características que el software debe cumplir para calificar como un estilo de arquitectura específico (Blancarte, 2020).

Sam Newman reconocido por su contribución al diseño de arquitecturas de software, establece que un estilo arquitectónico muy utilizado en el desarrollo de software es el estilo «Monolítico», donde todos los módulos esenciales se encuentran empaquetados en una única pieza; en un aplicativo de estilo monolítico, la capa de vista, lógica de negocio y acceso a datos son componentes que residen juntos en un solo sistema. Es completamente erróneo pensar que una aplicación monolítica está desordenada por dentro, con todo el código amontonado, creer que hay miles de clases u objetos desorganizados (que puede ser el caso) pero esos factores dependen del desarrollador, no del estilo arquitectónico (Newman, 2015).

Para Martin Fowler, uno de los arquitectos de software más reconocidos sostiene que “empezar con un monolito no es desacertado, sino que es lo correcto”. Al iniciar con un sistema de software desde cero, los monolitos son una excelente opción, porque evitan el rediseño, reducen el mantenimiento y agilizan actividades de testeo. Sin embargo, mientras estos sistemas empiezan a crecer, se vuelven más grandes y complejos en donde

la base de código puede ser extensa, es ahí donde la resolución de problemas se torna en un trabajo muy difícil. Durante este crecimiento, se debe considerar migrar el aplicativo a una arquitectura modular como es la arquitectura de microservicios (Fowler, 2015). Según Lewis y Fowler en su artículo titulado: «Microservicios», el enfoque arquitectónico de los microservicios se basa en la creación de pequeños módulos de software que se dedican a tareas específicas. Cada microservicio es autónomo, lo que permite la escalabilidad y la evolución independiente de los demás microservicios. Una ventaja de este enfoque es la capacidad de combinarlos para realizar operaciones conjuntas, sin importar la tecnología utilizada, el lenguaje de programación o incluso las bases de datos empleadas. (Lewis & Fowler, 2014). El estudio de Anabella Guimarey denominado «Beneficios y riesgos de migrar una arquitectura monolítica a microservicios», argumenta que para una exitosa migración de un monolito a microservicios ya no se debe seguir programando sobre la base de código, más bien se debe iniciar por el diseño de la arquitectura de software teniendo un esquema gráfico de cómo va a ser la interacción entre las estructuras del sistema, los componentes que la conformarían y sus relaciones (Guimarey, 2020).

1.2. Justificación

El presente trabajo, busca contribuir en la mejora de una plataforma de comercio electrónico utilizando el estilo arquitectónico de «Microservicios», como una solución tecnológica innovadora, permitiendo a una empresa PYME como Atlas, automatizar sus operaciones de venta, rastreo y entrega de productos en línea; el giro de negocio de la empresa radica principalmente en la promoción y venta de productos basados en electrónica de consumo, englobando artículos como: auriculares, parlantes, relojes inteligentes, etc. Con el desarrollo de esta plataforma tecnológica se desea llevar un control más organizado sobre las ventas realizadas y generar confianza en los clientes de la empresa, al poder comprar de forma segura con pagos en efectivo, tarjeta de crédito/débito y poder geolocalizar el estado de sus pedidos en tiempo real. La importancia de utilizar el estilo arquitectónico de microservicios es que se puede pensar en agregar nuevas funcionalidades al sistema dentro de un microservicio específico de forma inmediata, porque cada microservicio funciona como un proyecto independiente, donde se pueden hacer despliegues rápidos a producción sin afectar al resto de componentes. Otro factor importante de los microservicios es la interoperabilidad, dado que se comunican a través de las API, no solo entre microservicios si no con otros

ecosistemas como son las plataformas móviles. Las plataformas móviles, conforman un ecosistema de dispositivos móviles como los teléfonos inteligentes, tabletas, televisores inteligentes, etc. Los teléfonos inteligentes son los más utilizados, hoy en día están al alcance de millones de personas, razón por la cual muchas empresas están optando por ofrecer sus productos o servicios a través de las «Apps» para las diferentes plataformas móviles. Kramer Dean, Clark Tony y Oussena Samia contrastan en su trabajo: «Un lenguaje específico de dominio para implementación de múltiples plataformas móviles», dos estrategias de desarrollo de apps móviles: híbrido, usando lenguajes como Ionic, React Native y Flutter para los sistemas operativos de iOS y Android con un solo código; y nativo, con Swift para iOS y Java o Kotlin para Android, ofreciendo mejor rendimiento pero con código específico por plataforma, restringiendo su reutilización.

Finalmente, este trabajo propone la creación de una plataforma de comercio electrónico, para la venta y entrega a domicilio de productos tecnológicos, donde el cliente mediante su teléfono inteligente podrá ver en tiempo real la ubicación del repartidor para la recepción de su pedido.

1.3. Objetivos

1.4.1 Objetivo General

Construir una plataforma de comercio electrónico empleando una arquitectura de microservicios que contribuya a la mejora de procesos como la gestión, rastreo y entrega de productos a domicilio de la empresa Atlas.

1.4.2 Objetivos Específicos

- Analizar el proceso actual en la entrega de productos a domicilio de un comercio electrónico obteniendo un enfoque sistemático de los requerimientos de software.
- Diseñar una arquitectura basada en microservicios que permita la interconexión entre una aplicación web y una aplicación móvil.
- Desarrollar una aplicación web orientada a la recepción de pedidos y gestión de productos de forma organizada.
- Desarrollar una aplicación móvil que brinde al cliente la posibilidad de comprar en línea y ver el estado de su compra en tiempo real.

1.4. Alcance

La plataforma de comercio electrónico a ser desarrollada tiene como propósito contribuir a la mejora y automatización de procesos como la venta, gestión, rastreo y entrega de productos tecnológicos a domicilio para la empresa Atlas; con el fin de cumplir este propósito, se analizará el proceso actual de venta y entrega de productos a domicilio por parte del administrador y repartidor respectivamente. De igual forma se analizará el proceso de compra por parte de los clientes, obteniendo un enfoque sistemático para el levantamiento de requerimientos. Con este enfoque previamente definido se procederá a diseñar una arquitectura de software basada en microservicios que permitirá la interconexión entre la plataforma web principal y la aplicación móvil.

La plataforma web permitirá la recepción de pedidos en tiempo real por parte de los clientes, asimismo el administrador de la empresa podrá asignar los pedidos a su repartidor, proporcionando datos como: nombre del cliente, producto a entregar, forma de pago, ubicación y se manejaran 4 estados del pedido: solicitado, asignado, en curso, entregado. A nivel de arquitectura de software la plataforma web de comercio electrónico utilizará el estilo de microservicios, donde cada microservicio podrá resolver una tarea especifica del negocio como manejo de productos, usuarios, ventas, etc. Esta arquitectura brindará beneficios de agilidad, escalabilidad y se integrará fácilmente con diferentes plataformas.

La aplicación móvil la cual se comunica con la plataforma web mediante un API Gateway que permitirá al cliente comprar varios productos tecnológicos, al seleccionar los productos, ingresar su ubicación y elegir un método de pago, ya sea en efectivo, tarjeta de crédito y/o débito; una vez realizada la compra el cliente podrá ver dentro de la aplicación móvil un mapa con el recorrido del repartidor hacia su domicilio. Por otro lado, la aplicación móvil tendrá un menú extra únicamente para el usuario con el rol de "Repartidor", en este menú el repartidor podrá visualizar la lista de pedidos que debe entregar, junto con los datos del cliente, productos a entregar y la ubicación correspondiente a cada cliente. Finalmente se presentará la plataforma de comercio electrónico como un proyecto piloto innovador para la empresa Atlas, con el fin de extender las operaciones de venta.

1.5. METODOLOGÍA

La metodología por implementar es la denominada “Kanban”, debido a que emplea un sistema visual de seguimiento para supervisar el avance del trabajo a medida que se desplaza a lo largo del proceso de creación de valor. Por lo general, se usa un panel físico con hojas pequeñas adhesivas o un tablero virtual de tarjetas para gestionar las labores y las asignaciones. La principal contribución de Kanban a las metodologías ágiles es su capacidad para proporcionar transparencia al proceso mediante la implementación de tarjetas visuales. Esto implica que el proceso de trabajo se muestra abiertamente, lo que posibilita la identificación de obstáculos, acumulaciones, variaciones y pérdidas en el transcurso del tiempo, además de cualquier elemento que influya en el desempeño de la entidad en relación con la cantidad de trabajo completado y el tiempo necesario para hacerlo. En síntesis, Kanban brinda visibilidad a todos los miembros del equipo de trabajo y a los interesados acerca de los resultados de sus acciones o su omisión, esto a su vez, estimula una colaboración más intensa en el ámbito laboral. Asimismo, estimula el debate en torno a cambios posibles y alienta a los grupos de desarrollo de software a implementar optimizaciones a lo largo del desarrollo de las aplicaciones informáticas. (Maida & Pacienzia, 2015).

Kanban se desarrolló tomando como base los principios de producción «Justo a Tiempo» (JIT) creados por Toyota, que hacían uso de tarjetas visuales para detectar las demandas de material en la línea de producción. Esta metodología tiene como objetivo principal la gestión general de las tareas. Sus principales ventajas incluyen su facilidad de uso, actualización y adopción. También es notable por ser un enfoque de administración de actividades altamente gráfico y eficaz, que facilita la supervisión del avance de los proyectos y la organización del trabajo de forma eficaz (Radigan, 2021).

De acuerdo con Pilar Rodríguez y otros colaboradores en su artículo «Adopción de Metodologías Ágiles: un estudio comparativo entre España y Europa», en el contexto del desarrollo de software, existen diversas metodologías ágiles que se basan en un enfoque iterativo. En este proceso, los requisitos y las soluciones de software evolucionan mediante la colaboración entre equipos funcionales y autoorganizados. El trabajo se divide en segmentos, y cada tarea se asigna a uno de estos segmentos, que se exhibe en un tablero principal denominado tablero Kanban, como se ilustra en la Figura No 1. En cada segmento, se establecen restricciones para el trabajo en curso durante las diferentes

fases, generalmente denominadas como trabajo por hacer, en ejecución y finalizado. La cantidad y los nombres de las columnas varían en función de las exigencias del equipo. Las tareas se incorporan de forma continua al flujo de trabajo sin restricciones, a menos que se exceda el límite, en cuyo caso se mide el tiempo necesario para completar una tarea. Este enfoque ágil para el desarrollo de software posibilita la ágil producción de software funcional para atender las necesidades y las expectativas de las partes interesadas en la aplicación. La metodología Kanban para el desarrollo de software facilita la gestión de actividades y tareas, tratándolas como características, historias de usuario o requisitos del sistema que están pendientes, en proceso y completadas en las diversas etapas del desarrollo del software (Rodríguez et al., 2010).

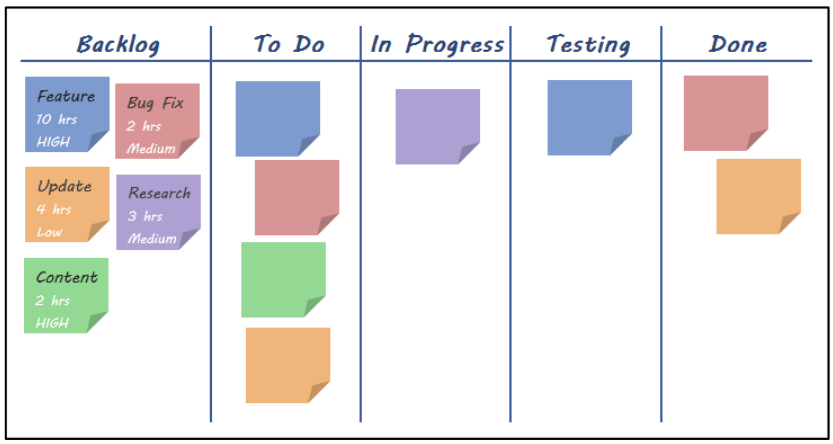


Figura 1: *Panel de trabajo Kanban*

Laura Castellano en su artículo titulado: «Kanban. Metodología para aumentar la eficiencia de los procesos», resalta que Kanban es sin duda una parte esencial en el desarrollo de sistemas de software, donde su objetivo principal se enfoca en supervisar la producción; para la plataforma de comercio electrónico el flujo de producción sería las necesidades del cliente hacia la empresa, es decir producir basándose en lo que el cliente demanda, dado que todo lo que aparezca fuera de planificación se considerará sobreproducción; otro punto a destacar son los roles de trabajo, debido que al ser un proyecto técnico desarrollado por una sola persona los roles son irrelevantes (Castellano, 2019).

La metodología Kanban trasciende su función como simple herramienta de visualización para impulsar la eficiencia y el mejoramiento constante dentro de los procesos de desarrollo de software. Centrándose en restringir la cantidad de trabajo pendiente, esta estrategia evita la sobrecarga de los equipos y mantiene un flujo de trabajo uniforme. La representación visual del trabajo permite a los equipos detectar rápidamente problemas y

áreas susceptibles de mejora, facilitando así la rápida adaptación y refinamiento de los procesos.

Kanban también cultiva un entorno de trabajo colaborativo y de comunicación transparente, permitiendo que todos los participantes del proyecto compartan la misma visión actualizada, lo cual simplifica la toma de decisiones. Su aplicabilidad va más allá del ámbito del desarrollo de software, pudiendo ser implementada en diversos campos y disciplinas, lo que resalta su flexibilidad y efectividad en la administración de proyectos. Implementar Kanban puede incrementar la satisfacción del equipo gracias a un proceso más claro y estructurado, reduciendo el estrés y fomentando un entorno de trabajo ordenado y previsible.

La visualización del tablero Kanban es de gran utilidad porque en él se reflejan los estados de cada tarea, para este proyecto se utilizarán los siguientes estados:

Estado	Descripción en Desarrollo de Software
Lista de pendientes	Tareas y requisitos aún no asignados o iniciados, esperando ser abordados.
Diseño	Fase donde se planifican las características técnicas y se diseñan las soluciones.
En proceso	Tareas que están siendo activamente desarrolladas o programadas.
Pruebas	Fase de verificación y validación para asegurar que el software cumple con los requisitos.
Completado	Tareas que han sido finalizadas, pasaron todas las pruebas y están listas para su entrega o despliegue.

Tabla 1: *Descripción de estados en el desarrollo de software*

Por lo tanto, Kanban fomenta un avance gradual y continuo de los procedimientos actuales, alineándose con los fundamentos de las metodologías ágiles. En lugar de buscar una alteración radical en la manera en que los individuos llevan a cabo sus tareas, Kanban propone una transformación paulatina respaldada por la comprensión y el acuerdo de todos los participantes del proyecto.

2. Marco Teórico Referencial

2.1. Introducción

En este apartado se muestra un resumen de las investigaciones previas llevadas a cabo sobre la arquitectura de software, los estilos arquitectónicos más utilizados para el desarrollo de sistemas y como estos estilos han ido evolucionado a través del tiempo; haciendo énfasis en las ventajas del estilo arquitectónico de «Microservicios» frente a los estilos arquitectónicos «REST» y «Monolítico».

En esta sección también se describirá el framework de Django para la construcción del sistema web y Android Studio bajo el lenguaje de programación Java para el desarrollo del sistema móvil, que juntos forman la plataforma de comercio electrónico para la gestión entrega y seguimiento de productos a domicilio, asimismo se definirán las tecnologías implicadas como Apache Kafka, Retrofit 2 y Google API Directions.

2.2. Arquitectura de Software

La arquitectura de software tiene varias definiciones con relación a lo descrito por Garlan y Shaw, se refiere a las estructuras de un sistema de software, los componentes que la conforman y el nexo que existe entre estos elementos, visualizando aspectos que van más allá de las estructuras de datos y los algoritmos de programación (Garlan & Shaw, s. f.). Por otro lado Blancarte Oscar, define que la arquitectura de software es el nivel de diseño más alto sobre la estructura de un sistema y consiste en un conjunto de abstracciones y patrones que sirven como base para la implementación de un sistema (Blancarte, 2020). Si bien existen numerosas definiciones similares acerca de la arquitectura de software, el núcleo de todas ellas es la noción de que la arquitectura de un sistema describe su estructura general utilizando una vista global de todo el sistema y para cada componente individual utiliza una vista a profundidad. Para la creación de diagramas de arquitectura de software se puede utilizar el «Modelo C4» que consiste en un conjunto jerárquico con diferentes niveles de abstracción para contexto, contenedores, componentes y código, donde cada nivel es relevante para un público diferente. Cabe resaltar que una buena arquitectura ayuda a garantizar que el aplicativo cumpla con los requisitos clave en aspectos de rendimiento, confiabilidad, portabilidad, escalabilidad e interoperabilidad.

2.3. Estilos Arquitectónicos

2.3.1. Estilo arquitectónico Monolítico

El estilo arquitectónico monolítico es donde todos los módulos esenciales se encuentran empaquetados en una única pieza. Newman Sam, establece que en un aplicativo de estilo monolítico la capa de presentación, lógica de negocio y acceso a datos son componentes que residen juntos en un solo sistema. Bajo este contexto una aplicación monolítica es un sistema que está fuertemente acoplado, debido a que sus componentes comparten los mismos recursos y memoria (Newman, 2015).

Sin embargo, cuando la aplicación monolítica crece en tamaño, se convierte en una pieza de software difícil de administrar y escalar, porque es más complicado dividir el trabajo entre los desarrolladores y detectar imprevistos en el sistema puede ser una tarea engorrosa.

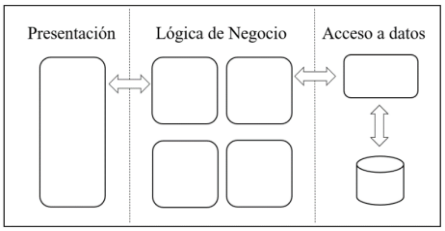


Figura 1: *Arquitectura monolítica (capas)*

Las ventajas más relevantes del estilo arquitectónico monolítico son las siguientes:

- Al iniciar un nuevo proyecto son fáciles de desarrollar. La implementación, es sencilla realizada desde un único archivo o directorio ejecutable.
- El performance de las aplicaciones monolíticas es más rápido, debido a que el procesamiento de datos es realizado localmente y no por procesos distribuidos.

No obstante, las desventajas son notorias cuando el sistema se encuentra en una etapa de crecimiento, y son las siguientes:

- Control de versiones, cualquier cambio por más pequeño que sea implicara volver a compilar todo el aplicativo.
- Escalabilidad, no se pueden escalar componentes de forma individual.
- Más costoso de mantener debido al gran volumen que puede tener una aplicación.
- Anclado a una pila tecnológica, lo que le impide aprovechar las nuevas tecnologías disponibles.

2.3.2. Estilo arquitectónico Rest

REST fue acuñado por Fielding Roy en su tesis doctoral y son las siglas de «Representational State Transfer», se define como un tipo de arquitectura de desarrollo web diseñado para sistemas distribuidos proporcionando una interfaz de interoperabilidad bajo el protocolo HTTP. Esta arquitectura se ha convertido en uno de los estilos más usados en la industria de software. Fielding Roy, también detalla a REST como un conjunto de características arquitectónicas que cuando se aplican como un todo, enfatizan el uso de interfaces comunes, ampliación de comunicación entre componentes e implementación independiente de los mismos; los elementos intermedios son utilizados para reducir la latencia en envió de información, mejorar la protección del sistema y encapsular sistemas heredados (Fielding, s. f.).

Las características de la arquitectura REST son las siguientes:

- El estilo arquitectónico debe estar separado en un modelo Cliente-Servidor, con el fin de mejorar la portabilidad de la interfaz de usuario a través de múltiples plataformas, optimizando en gran parte el factor de la escalabilidad porque se simplifican los componentes del servidor por un lado y la interfaz de usuario por otro lado.
- La URI «Identificador Uniforme de Recursos» y ningún otro elemento son identificadores únicos para cada recurso de un sistema REST. Las URI facilitan el acceso a la información para modificarla o eliminarla, incluso para compartir su ubicación exacta con terceros.
- Uso de una interfaz uniforme entre componentes donde se aplican acciones concretas como: GET (leer información), POST (crear registros), PUT (actualizar un registro existente), DELETE (eliminar un determinado registro), sobre los recursos siempre que estén identificados con una URI.
- Mejora el comportamiento de la escalabilidad en internet mediante un «Sistema de capas», limitando el comportamiento de los componentes para que no puedan ver más allá de la capa con la que esta interactuando.
- Soporte para JSON y XML, donde es indispensable que las respuestas se realicen siempre en uno de estos dos lenguajes de intercambio de información.
- Las respuestas a las solicitudes deben poder marcarse como en «caché» o «no caché», si una respuesta se puede almacenar en caché, los clientes de almacenamiento en

caché pueden reutilizar esa respuesta cuando realicen solicitudes similares más adelante.

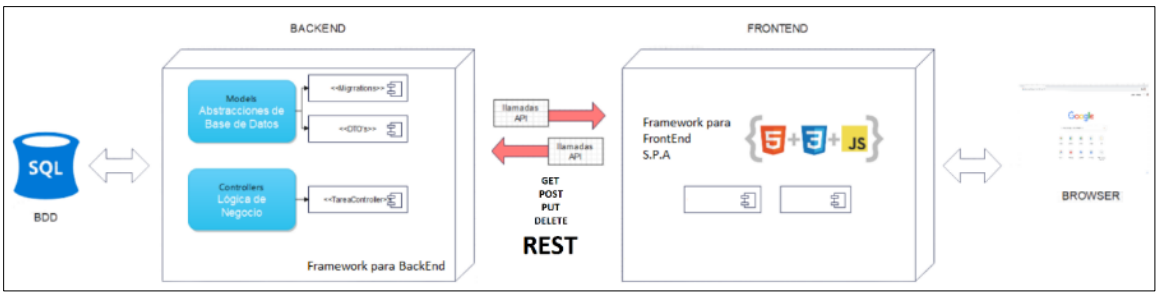


***Figura 2**: Arquitectura REST (backend - frontend)*

Las ventajas del estilo arquitectónico REST son:

- Escalabilidad: Debido a la división entre cliente y servidor, el producto final puede escalar gracias a la reducción del acoplamiento entre componentes.
- Interoperabilidad: REST es independiente del stack tecnológico, por lo que se puede implementar con cualquier lenguaje de programación.
- Visibilidad: Diseñado para ser visible y simple, lo que significa que cada aspecto del servicio debe seguir el estándar HTTP para ser autodescriptivo.
- Implementación: REST solicita menos recursos del servidor, debido a que las peticiones al no tener estado, no requiere memoria y puede atender más peticiones, de igual manera no requiere escribir o renderizar el código HTML, dando lugar a un menor procesamiento del lado del servidor.

Entre algunas desventajas se pueden destacar las siguientes:

- Performance: Al ser un sistema distribuido, la latencia es una constante en este tipo de sistemas.
- Documentación: Se debe documentar cada uno de los endpoints que se realicen, debido a que puede existir la creación de servicios similares, provocando funcionalidades repetidas en el sistema.
- Puntos de falla: Debido a su arquitectura distribuida, que posee múltiples componentes se multiplican los puntos de falla por cada componente.
- Sincronización en el desarrollo: Al ser una arquitectura separada en Frontend y Backend, cuando surgen nuevos requisitos las dos partes deben estar alineadas para cumplir con los objetivos del desarrollo y evitar pérdidas de tiempo (Pautasso et al., 2013).

2.3.3. Estilo Arquitectónico de Microservicios

En la actualidad el estilo arquitectónico de Microservicios se ha convertido en el más popular en los últimos años, donde este estilo arquitectónico surge a partir de que los sistemas de software empiezan a crecer y se vuelven mucho más grandes y complejos, como una alternativa a las arquitecturas tradicionales vistas anteriormente.

Para Lewis y Fowler, el estilo arquitectónico de «Microservicios» se fundamenta en crear y diseñar pequeños componentes de software que realizan una tarea en específico, donde cada microservicio es autosuficiente, concediendo la posibilidad de escalar y evolucionar de forma independiente al resto de microservicios, los microservicios se comunican entre sí por medio de una interfaz de programación «API REST» sobre el protocolo de transferencia de hipertexto «HTTP» (Lewis & Fowler, 2014).

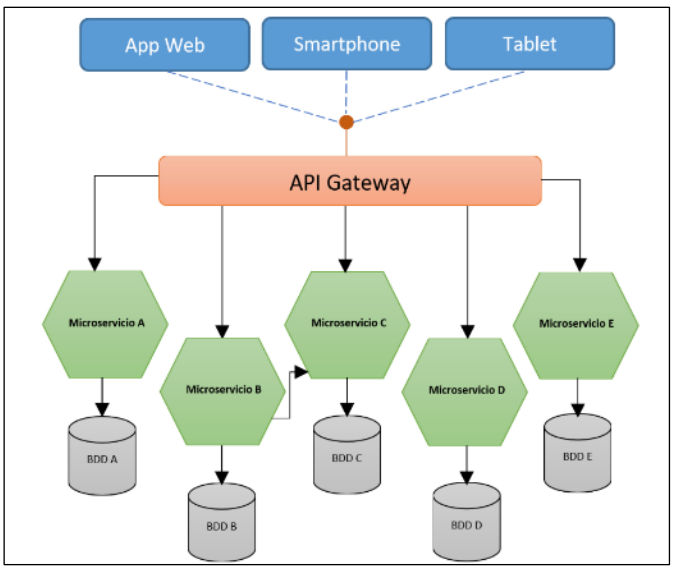


Figura 3: *Arquitectura de Microservicios*

Como se puede observar en la Figura 3, cada microservicio está conformado por su correspondiente y única lógica de negocio, manejando su propia base de datos y la comunicación se efectúa con otros microservicios. Esta arquitectura al poseer servicios autónomos facilita la posibilidad de escalar y cambiar gradualmente nuevas características por cada microservicio sin afectar al resto del sistema. Zhao y otros autores, sostienen que un componente importante es el «API Gateway», que actúa como puerta principal para controlar los accesos a los recursos o servicios internos que estarán disponibles en la red pública para finalmente mostrarlos en la interfaz de usuario, de esta forma se exponen solo los recursos necesarios y se ocultan los demás personalizando la experiencia para cada tipo de cliente (Zhao, Jing & Jiang, 2018).

Las ventajas más relevantes del estilo arquitectónico de microservicios son:

- Escalabilidad: Permite el crecimiento horizontal y vertical de cada componente de forma independiente sin que afecte al sistema global en el que se encuentra cada microservicio.
- Facilidad en el despliegue: Al ser un estilo diseñado por partes, no se debe desplegar una aplicación de gran tamaño, por el contrario, se despliega el microservicio que contenga los últimos cambios optimizando la velocidad en la que se liberan las nuevas funcionalidades del sistema.
- Diversidad de tecnologías: Se puede escoger una tecnología diferente para cada microservicio, ya sean el lenguaje de programación o la base de datos para utilizar dentro del microservicio.
- Tolerancia a fallos: Una falla en un microservicio puede aislarse y no propagarse a otros microservicios. Además, el tiempo de recuperación de un microservicio es menor que el tiempo que se tarda en restaurar todo un sistema completo, aumentando la disponibilidad del sistema.

Pese a las múltiples ventajas que ofrecen los microservicios, también se incluyen ciertas desventajas como:

- Performance: Los microservicios congenian con la naturaleza de los sistemas distribuidos, donde se agrega una latencia significativa en todo el sistema.
- Integridad de los datos: Al existir más una base de datos, se deben implementar mecanismos para asegurar la integración y consistencia de todos los datos situados en los demás microservicios.
- Trazabilidad: Al ser un sistema distribuido, la tarea de recuperar y realizar un seguimiento completo de la ejecución del proceso es complicada porque cada microservicio arroja su seguimiento o registro por separado, que luego se debe recopilar y consolidar para obtener un seguimiento completo.

En síntesis, el estilo arquitectónico de Microservicios ofrece muchas ventajas frente arquitecturas tradicionales, como la posibilidad de elegir diferentes tecnologías para cada servicio y en escenarios donde se piensa tener un gran escalamiento y agilidad en el desarrollo de cada componente de software.

2.3.4. Comparativa de Estilos Arquitectónicos

Una vez comprendidos los diferentes estilos arquitectónicos de software, cabe resaltar que ningún estilo arquitectónico es malo, sino que la decisión del estilo arquitectónico a utilizar depende mucho de las necesidades del proyecto que se va a desarrollar. A continuación, se exhibe una tabla comparativa que muestra las diferencias más destacadas de cada estilo arquitectónico de software.

Atributos	Monolítico	API REST	Microservicios
Cohesión	Baja cohesión, por la correlación de capas de software	Media cohesión, al separar la capa de presentación del Backend	Alta cohesión, al no estar sujeto de otros microservicios
Acoplamiento	Alto acoplamiento, por la dependencia entre clases	Bajo acoplamiento, porque los conectores definen interfaces abstractas	Bajo acoplamiento, por la independencia de los microservicios
Diseño	Centralizado	Descentralizado	Descentralizado
Rendimiento	Muy Rápido	Posee latencia ligera	Posee latencia ligera
Tecnología	Atado a un solo stack tecnológico	Puede usar diferentes tecnologías	Cada microservicio puede usar diferentes tecnologías
Comunicación	Requiere tuberías complejas entre módulos	Sencilla gracias a la comunicación por medio de API'S	Sencilla gracias a la comunicación por medio de API'S
Despliegue	Sencillo al desplegar un solo aplicativo	Dificultad media, al separar despliegue en Backend y frontend	Complejo, para mantener armonía en los microservicios
Tolerancia a fallos	Baja, si falla un componente falla todo el sistema	Media, puede ser falla en Frontend, backend o base de datos	Alta, porque se puede aislar las fallas sin propagar a los demás microservicios
Escalabilidad	Baja, el sistema al crecer va reduciendo su rendimiento	Media, se pueden agregar nuevos nodos para distribuir la carga entre los servidores	Alta, cada microservicio se adapta a las nuevas funcionalidades y el rendimiento no se ve afectado

Tabla 2: *Comparativa entre Arquitecturas de Software*

Se puede concluir que el estilo arquitectónico de Microservicios ofrece múltiples ventajas frente a las arquitecturas de estilo «Monolítico» y «REST», la más relevante es la fiabilidad, porque si un microservicio falla, solo afecta a una parte especifica de la aplicación, en lugar de afectar a toda la aplicación como en una arquitectura monolítica. Los microservicios se pueden escalar de forma independiente unos de otros, lo que permite una mayor escalabilidad en comparación de arquitecturas REST.

2.4. Django Framework

Un framework se define como un esquema de trabajo o plantilla, que proporciona a los programadores herramientas configurables que agilizan el proceso de desarrollo de software. Django es un framework o marco de trabajo web de alto nivel del lenguaje Python, gratuito y de código abierto, también fomenta el desarrollo rápido, diseño limpio y funcional.

Django utiliza el principio DRY «Don't Repat Yourself», el cual determina que se debe evitar duplicar código o funcionalidades en el sistema a desarrollar, dando como resultado menos líneas de código; en un nivel más bajo también agrupa el código relacionado en módulos bajo el patrón *Model View Controller* «*MVC*», aunque la nomenclatura correcta de Django es *Model View Template* «*MVT*», cumpliendo con el mismo propósito de una arquitectura MVC («Curso Django», 2012).

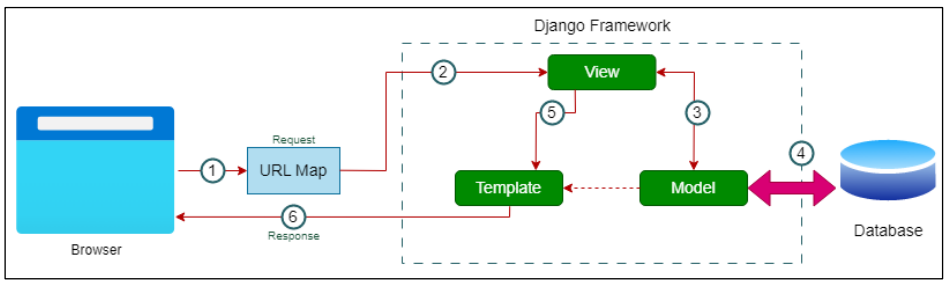


Figura 4: *Arquitectura Django (MVT)*

A continuación, se muestran las características más relevantes del framework de Django:

- Posee un ORM (Object-Relational mapping) sencillo, abstrayendo la necesidad de crear tablas y de realizar consultas SQL.
- Cuenta con un Sistema de autenticación de usuarios con JWT (JSON Web Token).
- Incluye protección para las aplicaciones, por ejemplo, contra ataques de inyección SQL y ataques XSS (Cross Site Scripting).
- Dispone de un panel de Administrador, el cual implementa acciones CRUD (Crear, Leer, Actualizar y Borrar) a la base de datos de forma sencilla.

2.4.1. Django REST Framework

Si bien Django framework funciona bajo el estilo arquitectónico monolítico (MVC), Django REST framework (DRF) permitirá romper este enfoque monolítico para poder llevar la arquitectura del proyecto hacia un enfoque de microservicios. Django REST framework ayuda a crear API's con funcionalidades RESTful (Get, Post, Put y Delete) al

agregar serializadores de objetos, opciones de autenticación adicionales y conjuntos de vistas combinadas (vistas avanzadas). Bajo este contexto Rest Framework será utilizado, para la creación de API's que permitan mantener la conexión y comunicación entre los distintos microservicios del aplicativo (Cleveland et al., 2020).

2.5. Android Studio

Android Studio se trata de un entorno de desarrollo integrado «IDE», basado en la suite de entornos de programación IntelliJ IDEA y desarrollado por Google, enfocado en la construcción de aplicaciones móviles nativas para el sistema operativo de Android, su codificación se realiza en el lenguaje de Kotlin (actual lenguaje oficial) o en el lenguaje de Java; asimismo Android Studio está basado en Gradle que favorece la reutilización de código y la compilación de aplicaciones. Otra característica es que permite probar las aplicaciones de forma virtual por medio de su emulador de dispositivos Android.

2.5.1. Lenguaje de Programación Java

Java se trata de un lenguaje de programación, el cual se puede utilizar para muchos propósitos, también es dinámico, orientado a objetos y se encuentra en constante evolución. Java en fue lanzado al mercado por primera vez en 1995 por Sun Microsystems y permite a los desarrolladores de software escribir el código del sistema una sola vez y su ejecución se realiza en varios dispositivos. Java se utiliza para desarrollar todo tipo de software, desde aplicaciones móviles, aplicaciones web, software empresarial hasta tecnologías de servidor y aplicaciones de Big Data (AWS, s. f.).

A continuación, se listan las características más relevantes de Java:

- Java es rápido, debido a que la JVM (Java Virtual Machine – interprete de Java) genera código nativo en tiempo de ejecución que es transparente para programadores y usuarios.
- JVM es políglota porque puede ejecutar código proveniente de diferentes lenguajes como Java junto a Kotlin, de forma simultánea.
- Java tiene el respaldo de una gran numero de programadores en todo el planeta, según el ranking TIOBE, java es uno de los lenguajes que ocupó el segundo y primer lugar como lenguaje de programación más utilizado en los últimos 20 años (Arnau, s. f.).

2.6. Otras Tecnologías

2.6.1. Apache Kafka

Apache Kafka es una plataforma distribuida de transmisión de datos en tiempo real y permite a los microservicios comunicarse entre sí de forma rápida y segura, se basa en el patrón «Publish/Subscribe». Contrastado por M.J. Clarke y otros autores, Apache Kafka permite que las aplicaciones anuncien eventos de forma asincrónica a múltiples consumidores sin emparejar remitentes con receptores, destacando que es una excelente opción cuando se utiliza, porque facilita la integración asíncrona con eventos aumentando el uso de las API's hacia la integración síncrona, respaldando a los microservicios con comunicación e incorporación ágil (Clarke et al., 2018)

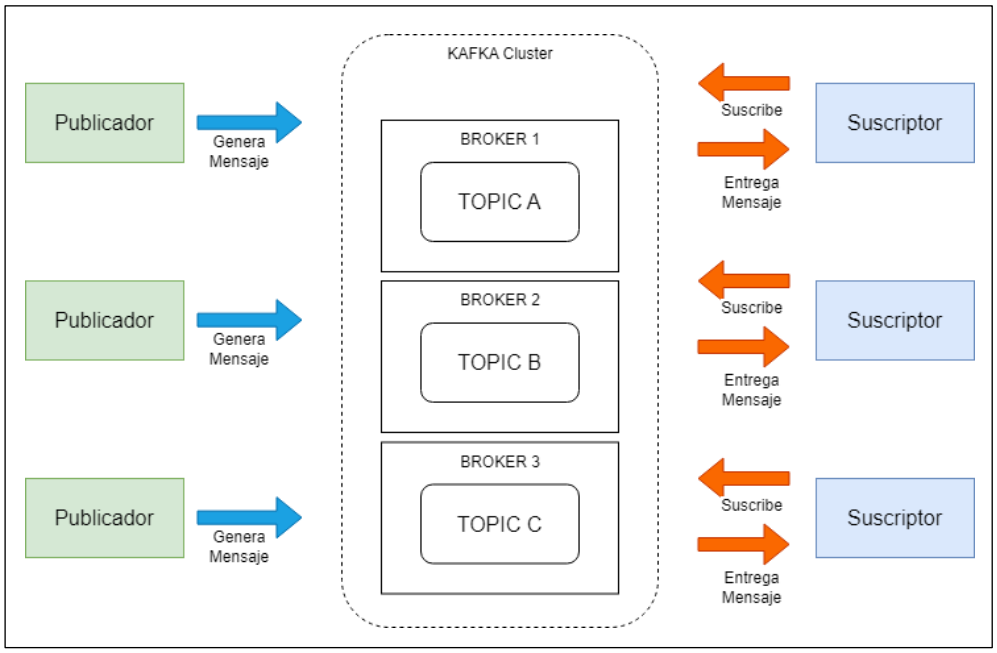


Figura 5: *Patrón Publish/Subscribe*

El funcionamiento del patrón Publish/Subscribe, es que un «Publicador» genera un mensaje, pero no lo envía directamente a un «Suscriptor», sino que envía a una lista de temas (Topics), y solo si el receptor esta suscrito a la lista de temas recibirá el mensaje.

Con este enfoque ya definido Vergara Sergio, establece que al implementar Apache Kafka en la arquitectura de microservicios, se evitan los cuellos de botella en la transferencia de información porque proporciona almacenamiento de información persistente y permite distribuir datos a través de múltiples nodos, garantizando una alta disponibilidad de información de diferentes bases de datos (Vergara, 2019).

2.6.2. Retrofit 2

Retrofit 2 es una biblioteca limpia, simple y liviana para el desarrollo de aplicaciones en Android, con relación a lo descrito por Lachgar Mohamed, Benouda Hanane y Elfirdoussi Selwa, Retrofit 2 es un cliente REST para Android, por la cual se pueden crear interfaces fáciles de utilizar convirtiendo cualquier aplicación Android en una aplicación robusta que puede enviar y recibir información con otras aplicaciones y bases de datos remotas (Lachgar, Benouda & Elfirdoussi, 2018).

Una de las características esenciales de Retrofit es que puede realizar consultas asíncronas y síncronas escaneando una petición en formato JSON de forma rápida y sin ningún esfuerzo. A continuación, se muestra la creación de una interfaz con Retrofit 2:

```java
package com.mycompany.example;

import retrofit2.Call;
import retrofit2.http.Body;
import retrofit2.http.Field;
import retrofit2.http.FormUrlEncoded;
import retrofit2.http.GET;
import retrofit2.http.POST;
import retrofit2.http.Query;
import com.mycompany.example.MultipleResource;
import com.mycompany.example.pojo.User;
import com.mycompany.example.pojo.UserList;

interface APIInterface {

    @GET("/api/unknown")
    Call<MultipleResource> doGetListResources();

    @POST("/api/users")
    Call<User> createUser(@Body User user);

    @GET("/api/users?")
    Call<UserList> doGetUserList(@Query("page") String page);

    @FormUrlEncoded
    @POST("/api/users?")
    Call<UserList> doCreateUserWithField(@Field("name") String name, @Field("job") String job);
```

Figura 6: *Ejemplo de interfaz Retrofit 2, con anotaciones HTTP*

2.6.3. Google API Directions

Se trata de un servicio en particular proveniente de Google Cloud Platform similar a Google Maps, el cual permite acceder a la trazabilidad de rutas desde un punto de inicio y un punto final; se puede acceder a la API de Directions por medio de una interfaz HTTP y las solicitudes se construyen como cadenas de URL. Esta cadena contiene el punto origen y destino de la ruta que se desea dibujar, también se debe proporcionar la clave de API con cada solicitud que realice al API de Directions.

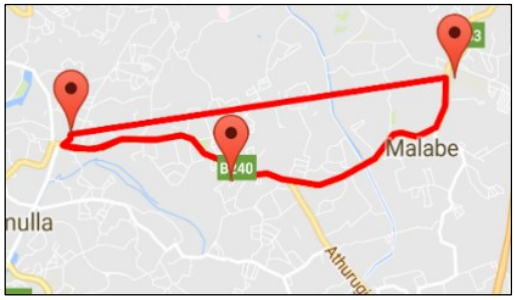


Figura 7: *Trazabilidad con API Directions*

3. Desarrollo

3.1. Análisis de Requisitos de Software

En este apartado, se presenta la Especificación de los Requisitos de Software, basada en las directrices establecidas por los estándares IEEE 830. Aquí se especifican las funciones principales del sistema, se establecen los requisitos funcionales, requisitos no funcionales y finalmente se consideran las limitaciones y características específicas del proyecto. Adicionalmente, se detalla cómo estos requisitos apoyan el logro de los objetivos del proyecto, asegurando que todas las partes interesadas tengan un entendimiento claro y completo de lo que la plataforma debe realizar. Esto incluye una descripción de la interacción esperada entre el usuario y el sistema, así como una explicación de cómo cada requisito contribuye a la funcionalidad general y la usabilidad del sistema.

3.1.1. Alcance

El objetivo del proyecto es desarrollar una herramienta informática que permita la gestión, entrega y monitoreo de la compra de productos en línea con entrega a domicilio del producto comprado por parte de los clientes de una compañía en particular. Este proyecto se llevará a cabo como un prototipo funcional para la empresa ATLAS. Para ello se desarrollará una plataforma de comercio electrónico, que permitirá a los clientes de la empresa realizar compras en línea por medio de un aplicativo móvil, de igual manera los clientes podrán compartir su ubicación y ver en tiempo real la localización de sus pedidos hasta su domicilio. Por medio del sistema web, el administrador de la empresa captará los pedidos de sus clientes y los asignará a un determinado repartidor, en base a criterios que el administrador considere primordiales para cada uno de sus clientes. Dentro del sistema móvil, el repartidor recibirá una lista de pedidos para entregarlos de forma inmediata al cliente final. Cabe resaltar que la plataforma de comercio electrónico requiere de la interconexión entre el sistema web basado en microservicios que será utilizado por el administrador de la empresa y del sistema móvil, que utilizaran los clientes y repartidores.

3.1.2. Definiciones, acrónimos y abreviaturas

- **ATLAS:** Empresa ecuatoriana especializada en la comercialización de tecnología de uso diario como, por ejemplo: relojes inteligentes, audífonos inalámbricos, parlantes, etc.

- **Administrador:** Persona encargada de dirigir y supervisar las tareas de un negocio o empresa en particular.
- **Repartidor:** Empleado de una institución responsable de entregar o distribuir bienes o productos hacia un lugar determinado.
- **Cliente:** Individuo u organización que compra productos o contrata servicios con el objetivo de satisfacer sus necesidades.

3.1.3. Descripción General

El propósito del proyecto es desarrollar una plataforma de comercio electrónico compuesta por un sistema web y sistema móvil. La plataforma integra distintos microservicios, donde cada microservicio cumple una función específica dentro del sistema. En consecuencia, el sistema web debe ser capaz de manejar los datos relacionados con los usuarios, productos, compras, y pedidos, así como con el inicio de sesión y registro de cuentas. Por otro lado, el sistema móvil, permitirá a los clientes iniciar sesión, comprar en línea y rastrear pedidos en tiempo real; asimismo permitirá a los repartidores entregar el pedido al cliente en la dirección solicitada.

3.1.4. Características del Usuario

Se ha clasificado a los usuarios en tres categorías basadas en las acciones que pueden ejecutar dentro de la plataforma de comercio electrónico, estas categorías son:

- **Administrador:** Usuario avanzado en tecnología, con conocimiento integral de la plataforma de comercio electrónico, que utiliza el sistema web.
- **Repartidor:** Experto en motocicletas, con habilidades básicas en tecnología para operar el sistema móvil, su función es entregar pedidos según lo indique el administrador.
- **Cliente:** Usuario con conocimientos básicos en tecnología, usa el sistema móvil para realizar compras en línea y recibir productos en su domicilio.

3.1.5. Funciones del Sistema

El sistema web tendrá que satisfacer varias características, las cuales estarán relacionadas con requerimientos específicos del negocio. Por consiguiente, se creará un microservicio individual para cada una de ellas. De esta forma, la aplicación en su totalidad deberá:

Microservicio	Rol	Funcionalidades
Administrar cuentas de usuario	Cliente	Creación de una cuenta
		Inicio de sesión/acceso para el usuario
		Visualización de una cuenta
		Actualización de la información de una cuenta
	Administrador	Eliminación de una cuenta
		Visualización de todas las cuentas
		Asignar rol de repartidor
Administrar catálogo de productos	Administrador	Gestión de categorías ofrecidas
		Gestión de productos en el catálogo
	Cliente	Catálogo de categorías ofrecidas
		Visualización de productos disponibles
		Descripción de un producto en particular
		Listado completo de los productos dentro de una categoría específica
Administrar carrito de compras	Cliente	Agregar productos al carrito de compras
		Listar productos del carrito
		Eliminar productos del carrito
		Validar el contenido del carrito de compras
Administrar pedidos	Cliente	Creación de pedidos
	Administrador	Listado de pedidos
		Asignar pedido a un repartidor

Tabla 3: *Funciones esenciales del sistema web.*

Además, se desarrollará el sistema móvil, donde los clientes y el usuario repartidor podrán acceder mediante un proceso de autenticación que incluirá los siguientes componentes:

Rol	Módulo	Funcionalidades
Cliente	Gestionar perfil	Registro de un usuario
		Inicio de sesión/acceso para el usuario
		Actualizar información del usuario
		Cierre de sesión para el usuario
	Gestionar catálogo	Consultar categorías
		Consultar productos
		Consultar productos por categoría
		Consultar un producto
	Gestionar pedidos	Crear pedido
		Visualizar pedido
Repartidor	Gestionar perfil	Registro de un usuario
		Inicio de sesión/acceso para el usuario
		Actualizar información del usuario
		Cierre de sesión para el usuario
	Gestionar entregas	Lista de entregas
		Finalizar entrega

Tabla 4: *Funciones esenciales del sistema móvil*

3.1.6. Restricciones del sistema

Al ser una plataforma con arquitectura de microservicios, el sistema web se alojará localmente en contenedores de Docker, ya que ofrece escalabilidad, flexibilidad y accesibilidad desde cualquier lugar. Por otro lado, el sistema móvil se alojará únicamente en smartphones con el sistema operativo de Android.

3.1.7. Particularidades y Dependencias

Los microservicios que implementan la lógica de negocio planteada serán desplegados localmente utilizando Docker esta solución permite un entorno de desarrollo y prueba consistente y fácilmente replicable, aprovechando la eficiencia y portabilidad que ofrecen los contenedores Docker. Para la comunicación entre microservicios se utilizará Apache Kafka y cada microservicio contará con su propia base de datos. Finalmente, para gestionar las solicitudes entre los sistemas web y móvil con los microservicios se utilizará un API Gateway montado sobre Nginx, para enrutar las solicitudes a los microservicios correspondientes a las diferentes solicitudes.

3.1.8. Requisitos Específicos

3.1.8.1. Requisitos de la plataforma

- Interfaces de Usuario

Se utilizarán interfaces de usuario para los sistemas web y móvil que resulten intuitivas para lograr una experiencia agradable en la interacción con el sistema. Se buscará emplear interfaces limpias y minimalistas para tal fin.

- Interfaces de Hardware

Para el sistema web es necesario que el sistema sea compatible con arquitecturas AMD64, memoria RAM de 1 GB como mínimo. Para el sistema móvil es necesario que el hardware específico del smartphone o tablet que debe contar con cámara, GPS y sensor de proximidad.

- Interfaces de Software

La plataforma de comercio electrónico está compuesta por el sistema web y sistema móvil, la comunicación entre ambos sistemas se efectúa mediante la interfaz de comunicación API REST, misma que envía y recibe peticiones en formato JSON. Se

aconseja que el sistema móvil sea utilizado en dispositivos Android como smartphones o tablets, cuyo sistema operativo tenga una versión igual o superior a la 5.0 (Lollipop).

3.1.9. Requisitos Funcionales

En la elaboración de los requisitos funcionales, se tendrá en cuenta que cada microservicio está integrado dentro del sistema web y cada módulo que se encontrará alojado en el sistema móvil.

3.1.9.1. Sistema Web

- Microservicio de Cuentas

Requisito Funcional 1.1: Crear cuenta de usuario

Código	RF 1.1	**Prioridad**	Alta
Descripción	Registrar un usuario a partir de su información personal		
Precondición	Ninguna		
Entrada	Datos personales del usuario: (nombre, apellido, correo electrónico, celular, dirección y contraseña)		
Proceso	Datos ingresados en el sistema web		
Salida	Mensaje de registro exitoso / Mensaje de error		

Requisito Funcional 1.2: Acceder a cuenta de usuario

Código	RF 1.2	**Prioridad**	Alta
Descripción	Permitir a los clientes acceder a la plataforma por medio de sus credenciales		
Precondición	RF 1.1		
Entrada	Credenciales de usuario: (correo electrónico y contraseña)		
Proceso	Credenciales validadas para el acceso al sistema web		
Salida	Mensaje de ingreso exitoso / Mensaje de error		

Requisito Funcional 1.3: Listar cuenta de usuario

Código	RF 1.3	**Prioridad**	Alta
Descripción	A cada usuario se le permitirá visualizar los datos personales vinculados a su cuenta		
Precondición	RF 1.2		
Entrada	Ninguna		
Proceso	Datos personales consultados en el sistema web		
Salida	Datos personales del usuario: (nombre, apellido, correo electrónico, celular, dirección)		

Requisito Funcional 1.4: Modificar cuenta de usuario

Código	RF 1.4	**Prioridad**	Alta
Descripción	Los datos personales pueden ser modificados por el usuario correspondiente.		
Precondición	RF 1.2		
Entrada	Datos personales del usuario: (nombre, apellido, correo electrónico, celular, dirección)		
Proceso	Datos personales modificados por cada usuario		
Salida	Mensaje de datos actualizados / Mensaje de error		

Requisito Funcional 1.5: Eliminar cuenta de usuario

Código	RF 1.5	**Prioridad**	Baja
Descripción	Únicamente el administrador del sistema web podrá eliminar una cuenta de usuario		
Precondición	RF 1.2		
Entrada	Selección de usuario a ser eliminado		
Proceso	Eliminar información correspondiente al usuario seleccionado		
Salida	Mensaje de usuario eliminado / Mensaje de Error		

Requisito Funcional 1.6: Listar cuentas de usuario

Código	RF 1.6	**Prioridad**	Alta
Descripción	Solo el administrador del sistema web podrá listar todas las cuentas de usuario registradas en el sistema		
Precondición	RF 1.2		
Entrada	Ninguna		
Proceso	Consultar información relevante de todos los usuarios registrados		
Salida	Detalle de usuarios registrados / Mensaje de Error		

Requisito Funcional 1.7: Asignar rol de repartidor

Código	RF 1.7	**Prioridad**	Alta
Descripción	El administrador del sistema web tendrá la capacidad de designar a un usuario específico el rol de repartidor		
Precondición	RF 1.2		
Entrada	Selección de un usuario determinado		
Proceso	Asignar rol de repartidor a usuario seleccionado		
Salida	Mensaje de asignación de rol / Mensaje de Error		

- MICROSERVICIO DE TIENDA

Requisito Funcional 2.1: Gestión de categorías

Código	RF 2.1	**Prioridad**	Alta
Descripción	El administrador del sistema web podrá agregar, modificar y eliminar categorías para los productos que ofrece la empresa		
Precondición	Ninguna		
Entrada	Datos de la categoría: (código, nombre, slug y descripción)		
Proceso	Datos modificados en el sistema web		
Salida	Mensaje de aceptación / Mensaje de error		

Requisito Funcional 2.2: Consultar categorías

Código	RF 2.2	**Prioridad**	Alta
Descripción	Mostrar a los usuarios las categorías disponibles		
Precondición	RF 2.1		
Entrada	Ninguna		
Proceso	Datos de categorías consultadas en el sistema web		
Salida	Detalle de categorías registradas / Mensaje de error		

Requisito Funcional 2.3: Gestión de productos

Código	RF 2.3	**Prioridad**	Alta
Descripción	El administrador del sistema web podrá agregar, modificar y eliminar los productos que ofrece la empresa		
Precondición	RF 2.1		
Entrada	Datos del producto: (código, nombre, slug, precio, descripción, imagen, stock, categoría)		
Proceso	Datos modificados en el sistema web		
Salida	Mensaje de aceptación / Mensaje de error		

Requisito Funcional 2.4: Consultar productos

Código	RF 2.4	**Prioridad**	Alta
Descripción	Mostrar a los usuarios una lista de con todos los productos disponibles		
Precondición	RF 2.3		
Entrada	Ninguna		
Proceso	Datos de los productos consultados en el sistema web		
Salida	Información relevante de los productos registrados / Mensaje de error		

Requisito Funcional 2.5: Consultar un producto

Código	RF 2.5	**Prioridad**	Alta
Descripción	Mostrar a los usuarios un producto en concreto de forma detallada		
Precondición	RF 2.3		
Entrada	Selección de un producto		
Proceso	Datos del producto seleccionado en el sistema web		
Salida	Detalle de producto elegido / Mensaje de error		

Requisito Funcional 2.6: Consultar productos por categoría

Código	RF 2.6	**Prioridad**	Alta
Descripción	Mostrar a los usuarios una lista todos los productos pertenecientes a una determinada categoría		
Precondición	RF 2.2 - RF 2.3		
Entrada	Selección de categoría		
Proceso	Datos de los productos pertenecientes a una categoría en el sistema web		
Salida	Información relevante de los productos por categoría / Mensaje de error		

- MICROSERVICIO DE PEDIDOS

Requisito Funcional 3.1: Agregar productos al carrito

Código	RF 3.1	**Prioridad**	Alta
Descripción	Permitir al usuario solicitante añadir productos al carrito de compras		
Precondición	RF 1.2 - RF 2.5		
Entrada	Selección de productos		
Proceso	Datos de productos agregados al carrito		
Salida	Mensaje de registro exitoso / Mensaje de error		

Requisito Funcional 3.2: Listar productos del carrito

Código	RF 3.2	**Prioridad**	Alta
Descripción	Cada usuario solicitante, tendrá la posibilidad de visualizar el contenido de su carrito de compras		
Precondición	RF 3.1		
Entrada	Inserción de productos dentro del carrito		
Proceso	Mostrar datos relevantes de los productos agregados al carrito		
Salida	Detalle de los productos que están dentro del carrito / Mensaje de error		

Requisito Funcional 3.3: Eliminar productos del carrito

Código	RF 3.3	**Prioridad**	Alta
Descripción	El usuario solicitante podrá vaciar el contenido de su carrito		
Precondición	RF 3.1 – RF 3.2		
Entrada	Productos dentro del carrito a ser eliminados		
Proceso	Datos de productos eliminados del carrito		
Salida	Detalle de los productos que están dentro del carrito / Mensaje de error		

Requisito Funcional 3.4: Validar carrito

Código	RF 3.4	**Prioridad**	Alta
Descripción	La validación del carrito resultará en la creación de un pedido de forma automática		
Precondición	RF 3.1 – RF 3.2		
Entrada	Datos del carrito de compras: (productos, cantidad, precio)		
Proceso	Generación de un nuevo pedido con los productos seleccionados		
Salida	Mensaje de aceptación/ Mensaje de error		

Requisito Funcional 4.1: Crear pedido

Código	RF 4.1	Prioridad	Alta
Descripción	Generar un nuevo pedido a partir de los datos del carrito		
Precondición	RF 3.4		
Entrada	Datos del carrito de compras: (productos, cantidad, precio, impuestos, dirección de entrega)		
Proceso	El usuario comprueba y da su conformidad al pedido		
Salida	Mensaje de aprobación / Mensaje de error		

Requisito Funcional 4.2: Listar pedidos

Código	RF 4.2	Prioridad	Alta
Descripción	El administrador del sistema web podrá ver una tabla que contiene un listado de pedidos a ser entregados.		
Precondición	RF 4.1		
Entrada	Ninguna		
Proceso	Visualización de una tabla con el detalle de los pedidos		
Salida	Resumen de pedidos / Mensaje de error		

Requisito Funcional 4.3: Asignar pedido

Código	RF 4.3	Prioridad	Alta
Descripción	El administrador del sistema web podrá asignar un determinado pedido al usuario con el rol de "Repartidor", en base a los criterios que el crea conveniente		
Precondición	RF 4.2		
Entrada	Datos del pedido: Información del cliente, pedido y estado del pedido		
Proceso	Cambiar el estado del pedido de "Creado" a "Asignado"		
Salida	Mensaje de asignación / Mensaje de error		

3.1.9.2. Sistema Móvil

- Módulo Gestionar Perfil

Requisito Funcional 5.1: Activar cuenta de usuario

Código	RF 5.1	Prioridad	Alta
Descripción	Registrar un usuario a partir de su información personal		
Precondición	Ninguna		
Entrada	Datos personales del usuario: (nombre, apellido, correo electrónico, celular, dirección y contraseña)		
Proceso	Datos ingresados en el sistema móvil		
Salida	Mensaje de registro exitoso / Mensaje de error		

Requisito Funcional 5.2: Acceder a cuenta de usuario

Código	RF 5.2	Prioridad	Alta
Descripción	Permitir a los diferentes tipos de usuarios (cliente o repartidor) acceder al sistema móvil por medio de sus credenciales.		
Precondición	RF 5.1		
Entrada	Credenciales de usuario: (correo electrónico y contraseña)		
Proceso	Credenciales validadas para acceder al sistema móvil		
Salida	Mensaje de ingreso exitoso / Mensaje de error		

Requisito Funcional 5.3: Actualizar cuenta de usuario

Código	RF 5.3	**Prioridad**	Alta
Descripción	El usuario correspondiente (cliente o repartidor), tiene la posibilidad de realizar modificaciones en sus datos personales		
Precondición	RF 5.2		
Entrada	Datos personales del usuario: (nombre, apellido, correo electrónico, celular, dirección)		
Proceso	Datos de usuario actualizados		
Salida	Mensaje de aceptación / Mensaje de error		

Requisito Funcional 5.4: Cerrar sesión de usuario

Código	RF 5.4	**Prioridad**	Alta
Descripción	Permitir a los diferentes tipos de usuarios (cliente o repartidor) terminar la sesión actual dentro del sistema móvil		
Precondición	RF 5.2		
Entrada	Ninguna		
Proceso	Finalizar sesión de usuario		
Salida	Mensaje de aceptación / Mensaje de error		

- Módulo Gestionar Tienda

Requisito Funcional 6.1: Consultar categorías

Código	RF 6.1	**Prioridad**	Alta
Descripción	Mostrar al usuario cliente las categorías de productos disponibles		
Precondición	RF 5.2		
Entrada	Ninguna		
Proceso	Datos de categorías registradas en la plataforma de comercio electrónico		
Salida	Detalle de categorías registradas / Mensaje de error		

Requisito Funcional 6.2: Consultar productos

Código	RF 6.2	**Prioridad**	Alta
Descripción	Mostrar al usuario cliente una lista de todos los productos disponibles para comprar		
Precondición	RF 5.2		
Entrada	Ninguna		
Proceso	Datos de los productos pertenecientes a una categoría		
Salida	Información relevante de los productos por categoría / Mensaje de error		

Requisito Funcional 6.3: Consultar productos por categoría

Código	RF 6.3	**Prioridad**	Alta
Descripción	Mostrar al usuario cliente una lista todos los productos pertenecientes a una determinada categoría		
Precondición	RF 6.1 - RF 6.2		
Entrada	Selección de categoría		
Proceso	Datos de los productos pertenecientes a una categoría en el sistema móvil		
Salida	Información relevante de los productos por categoría / Mensaje de error		

Requisito Funcional 6.4: Consultar un producto

Código	RF 6.4	**Prioridad**	Alta
Descripción	Mostrar al usuario cliente un producto en concreto de forma detallada		
Precondición	RF 6.2		
Entrada	Selección de un producto		
Proceso	Datos del producto seleccionado en el sistema móvil		
Salida	Detalle de producto elegido / Mensaje de error		

- Módulo Gestionar Pedidos

Requisito Funcional 7.1: Crear pedido

Código	RF 7.1	**Prioridad**	Alta
Descripción	El usuario cliente puede generar un nuevo pedido a partir de los productos seleccionados en el sistema móvil		
Precondición	RF 6.3		
Entrada	Datos del pedido: (productos, cantidad, precio, impuestos, dirección de entrega, coordenadas de geolocalización)		
Proceso	El usuario comprueba y da su conformidad al pedido, el estado del pedido se denomina Creado.		
Salida	Mensaje de aprobación / Mensaje de error		

Requisito Funcional 7.2: Visualizar pedido

Código	RF 7.2	**Prioridad**	Alta
Descripción	El usuario cliente puede visualizar el estado del pedido: (alistando – en camino – entregado)		
Precondición	RF 7.1		
Entrada	Ninguna		
Proceso	El usuario comprueba en tiempo real el estado de su pedido		
Salida	Mensaje de aprobación / Mensaje de error		

- Módulo Gestionar Entregas

Requisito Funcional 8.1: Lista de entregas

Código	RF 8.1	**Prioridad**	Alta
Descripción	El usuario repartidor podrá visualizar un listado de las entregas que debe realizar		
Precondición	RF 5.2		
Entrada	Datos del pedido: (datos del cliente, productos, cantidad, precio, impuestos, dirección de entrega, coordenadas de geolocalización)		
Proceso	El usuario repartidor comprueba los datos del pedido y se dirigirá a la dirección del cliente para entregar el pedido solicitado, el estado del pedido cambiará a “en curso”		
Salida	Mapa con la geolocalización del cliente / Mensaje de error		

Requisito Funcional 8.2: Finalizar entrega

Código	RF 8.2	**Prioridad**	Alta
Descripción	Una vez que el usuario repartidor entregue el pedido al cliente, cambiara el estado de la tarea de “en curso” a “entregado”		
Precondición	RF 8.1		
Entrada	Ninguna		
Proceso	El usuario repartidor podrá cambiar el estado de la tarea a “entregado”		
Salida	Mensaje de aprobación / Mensaje de error		

3.1.10.Requisitos No Funcionales

Tanto el sistema web como el sistema móvil de la plataforma de comercio electrónico, tendrán una variedad de cualidades y características particulares que se han detallado a continuación:

3.1.10.1. Disponibilidad

Requisito No Funcional 1.1: Disponibilidad de la plataforma

Código	RNF 1.1	**Prioridad**	Alta
Descripción	La plataforma debe estar disponible en todo momento		
Entrada	Arquitectura de la plataforma		
Proceso	La plataforma debe garantizar la disponibilidad ininterrumpida de sus aplicaciones, a menos que se vean afectadas por factores externos que estén fuera de control		
Salida	No aplica		

3.1.10.2. Usabilidad

Requisito No Funcional 2.1: Interfaz de usuario

Código	RNF 2.1	**Prioridad**	Alta
Descripción	El diseño de la interfaz de la plataforma debe ser simple y fácil de usar		
Entrada	Formularios, botones, menús, iconos		
Proceso	La plataforma tendrá interfaces limpias y dinámicas, diseñadas para una experiencia de usuario atractiva y fácil de utilizar		
Salida	Plantillas HTML para el sistema web y diseños XML para el sistema móvil, juntos componen la interfaz gráfica de la plataforma.		

Requisito No Funcional 2.2: Diseño adaptable

Código	RNF 2.2	**Prioridad**	Alta
Descripción	La plataforma asegurará una visualización óptima de las interfaces de usuario		
Entrada	Interfaces de usuario		
Proceso	La plataforma debe contar con un diseño adaptable y sensible a la pantalla, que permita el uso de las aplicaciones en diferentes dispositivos, como computadoras, smartphones y tabletas		
Salida	Interfaz con diseño “responsive” para cada dispositivo		

3.1.10.3. Seguridad

Requisito No Funcional 3.1: Seguridad de credenciales

Código	RNF 3.1	**Prioridad**	Alta
Descripción	La plataforma debe proteger las contraseñas de los usuarios mediante cifrado		
Entrada	Contraseñas en la plataforma por cada usuario		
Proceso	La plataforma deberá cifrar la contraseña al momento de crear un usuario y antes de almacenarla en la base de datos, con el fin de garantizar la seguridad		
Salida	Encriptación de contraseñas		

3.1.10.4. Portabilidad

Requisito No Funcional 4.1: Interoperabilidad

Código	RNF 4.1	**Prioridad**	Alta
Descripción	La plataforma será compatible con diferentes sistemas operativos		
Entrada	Arquitectura de la plataforma		
Proceso	El sistema web, podrá ser ejecutado en cualquier sistema operativo a través del navegador Web, sin restricciones. El sistema móvil se ejecutará en smartphones y tabletas con la restricción de ejecutarse únicamente bajo el sistema operativo Android		
Salida	Ejecución exitosa de la plataforma		

3.2. Diseño

La arquitectura de software debe basarse en el estilo arquitectónico de «microservicios» y seguir la especificación de requisitos de software para cubrir las tres áreas de negocio básicas de la plataforma de comercio electrónico como: administrar cuentas de usuario, administrar catálogo de productos, administrar pedidos junto con el carrito de compras. Cada uno de estos tres microservicios tendrá su propia base de datos independiente para administrar su información.

El diseño debe incluir dos componentes de interconexión esenciales, el primero debe permitir la comunicación entre los microservicios de toda la plataforma de comercio electrónico, como la creación de un nuevo pedido a partir de la información del usuario, lo que requerirá una llamada al microservicio de tienda y pedidos desde el microservicio de cuentas de usuario. El segundo componente es crucial para el intercambio de información, debido a que funciona como un intermediario entre los clientes del sistema web o del sistema móvil y los microservicios de la plataforma. En este caso el «API Gateway» se encarga de enrutar las solicitudes de los clientes al microservicio adecuado, actuando como un único punto de entrada.

Además, se debe tomar en cuenta que los sistemas web y móvil deben: gestionar perfil de usuario, gestionar catálogo, gestionar pedidos y gestionar entregas. Con el fin de que el cliente reciba su pedido a domicilio y que el repartidor haga la entrega de los pedidos hacia el cliente.

En resumen, se requiere lo siguiente:

- Identificar las tres áreas de negocio y crear un microservicio para cada una de ellas.
- Cada microservicio debe disponer de su propia base de datos para la gestión de la información que maneja.
- Se necesitará una herramienta que permita la comunicación entre los distintos microservicios, como por ejemplo Apache Kafka.
- Crear una conexión entre los clientes del sistema web y el sistema móvil con los microservicios de la plataforma de comercio electrónico, empleando el API Gateway con Nginx.

Por lo tanto, la arquitectura de software se puede visualizar mediante el gráfico que se muestra a continuación:

3.2.1 Arquitectura de Software

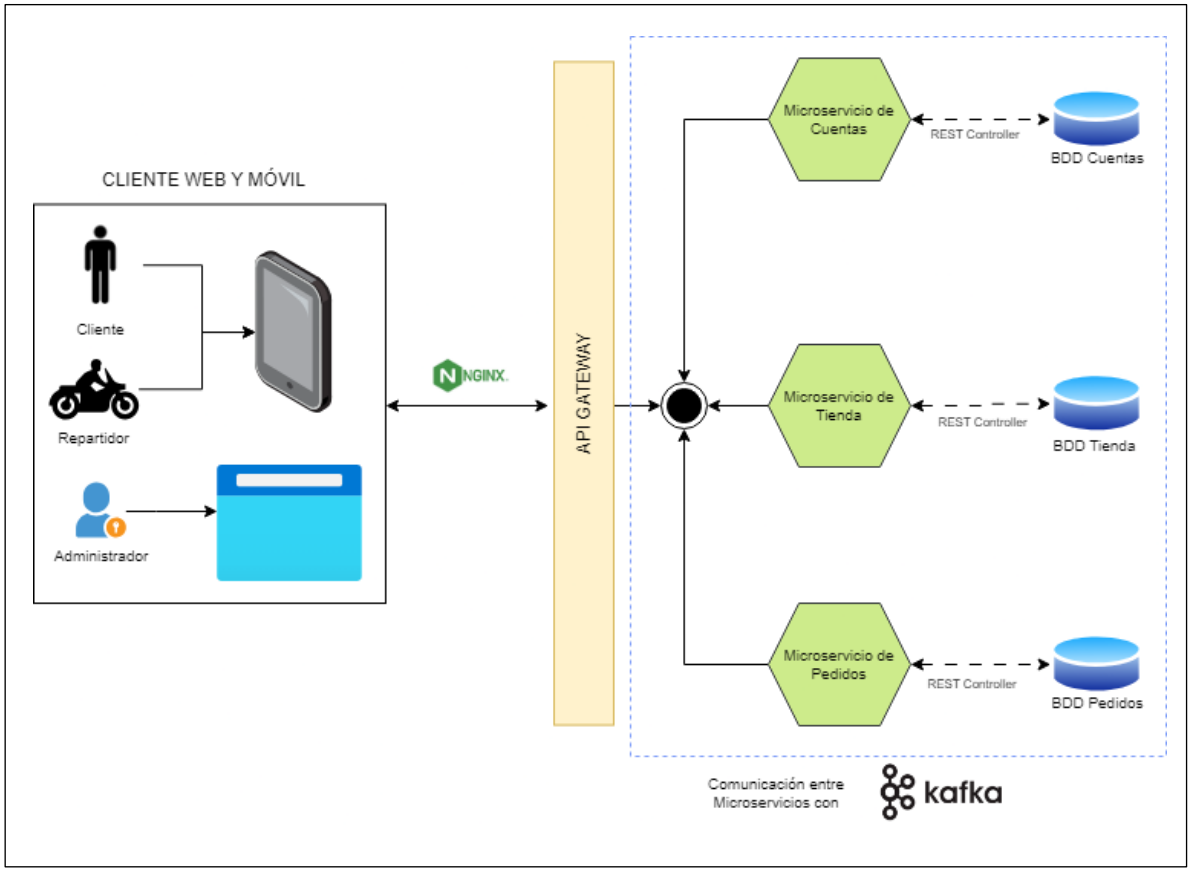


Figura 2: *Descripción general de la arquitectura del sistema*

En la imagen de arriba se puede observar que cada microservicio, incluye un controlador que recibe solicitudes relacionadas con los recursos, y luego llama al servicio correspondiente según la acción requerida. Para obtener o aplicar los datos de la solicitud, el servicio utiliza una base de datos exclusiva para cada microservicio.

Apache Kafka, supervisa todo este proceso y se encarga de gestionar los microservicios y coordinar la comunicación entre ellos. Cada servicio se registra en esta herramienta para definir su ubicación, Apache Kafka también recibe solicitudes del cliente y las dirige a cada uno de los microservicios. La plataforma de comercio electrónico tendrá dos sistemas esenciales, el sistema web que será utilizado por el administrador y el sistema móvil que será utilizado tanto por los clientes como por el usuario repartidor.

3.2.2 Diagramas de Casos de Uso

A continuación, se presentan los diagramas de casos de uso que muestran las principales características de la plataforma agrupadas en los principales microservicios del sistema de manera general. Estos esquemas son utilizados para identificar, describir y analizar las interacciones entre los usuarios o actores y el sistema.

3.2.2.1 Caso de uso 1: Microservicio de Cuentas

Este microservicio permitirá al cliente registrarse, iniciar sesión, visualizar su información y actualizarla. Por otro lado, el Administrador podrá visualizar todas las cuentas de usuario creadas en el sistema, eliminar una o varias cuentas a su elección y asignar el rol de Repartidor a un determinado usuario.

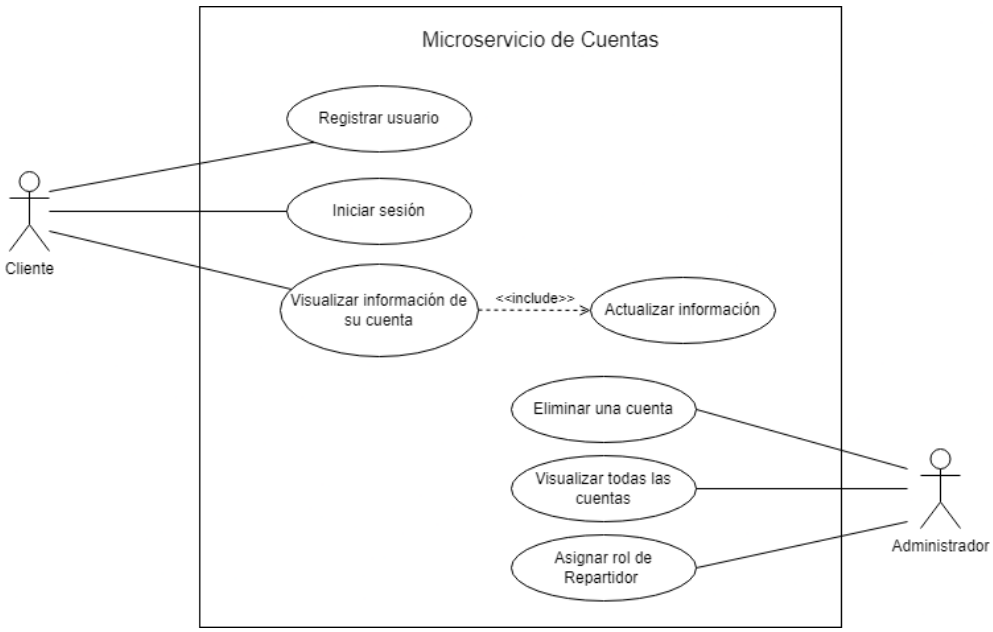


Figura 3: *Caso de uso 1 – Microservicio de Cuentas para administrar cuentas de usuario*

3.2.2.2 Caso de uso 2: Microservicio de Tienda

La funcionalidad de este microservicio permitirá al Administrador, crear categorías para cada producto que se visualizará en el catálogo de productos. Asimismo, el cliente podrá observar el catálogo de productos existentes agrupados por una determinada categoría.

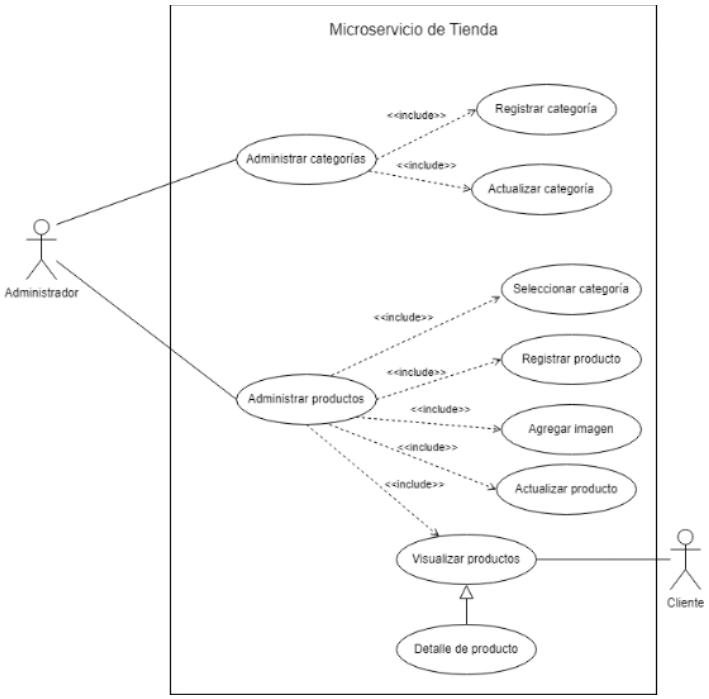


Figura 4: *Caso de uso 2 - Microservicio de Tienda para administrar catálogo de productos*

3.2.2.3 Caso de uso 3: Microservicio Pedidos

Este microservicio permitirá al Cliente agregar productos al carrito de compras, listar los productos y eliminar los productos del carrito de compras. Además, permite a los clientes generar sus pedidos y consultar su historial de pedidos. Después de esto, el administrador se encarga de asignar el pedido del cliente a un repartidor para que realice la entrega a domicilio correspondiente.

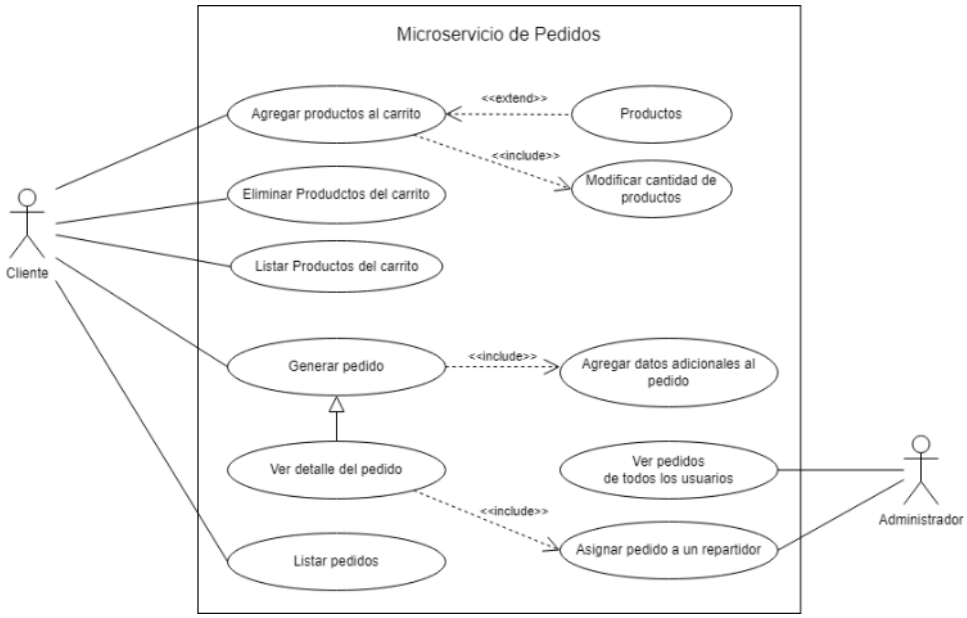


Figura 5: *Caso de uso 3 – Microservicio de Pedidos*

3.2.2.4 Caso de uso 4: Proceso de compra y entrega de productos a domicilio

A continuación, se muestra el proceso de compra de un producto por parte del cliente, como el Administrador gestiona el pedido y lo asigna al repartidor, para que finalmente el repartidor entregue el pedido en el domicilio del cliente.

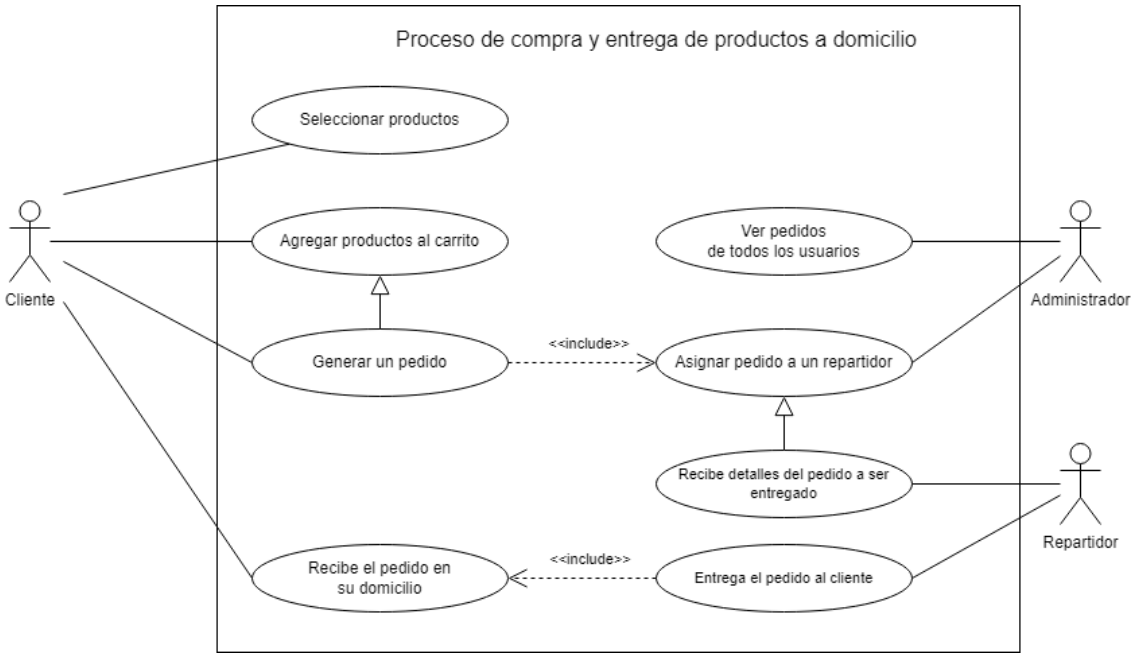


Figura 6: *Caso de uso 4 - Proceso de compra y entrega de productos a domicilio*

3.2.3 Diagramas de base de datos

Los requisitos propuestos han llevado a la determinación de que la plataforma debe incluir tres bases de datos, una para cada microservicio. Se han creado los siguientes diagramas conceptuales para cada microservicio, los cuales incluyen las tablas y relaciones necesarias para almacenar todos los datos relacionados con la gestión, entrega y seguimiento de productos a domicilio para la empresa ATLAS.

3.2.3.1 Diagrama de base de datos Microservicio Cuentas

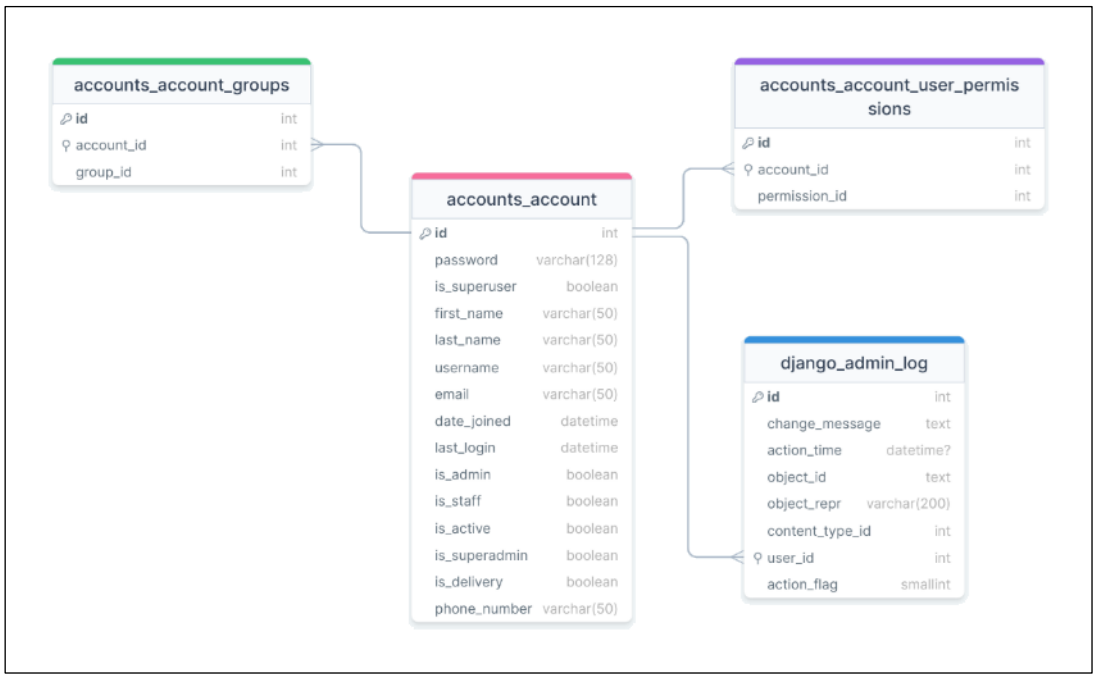


Figura 7: *Esquema conceptual de la base de datos del microservicio de cuentas*

3.2.3.2 Diagrama de base de datos Microservicio Tienda

Figura 8: *Esquema conceptual de la base de datos del microservicio de catálogo*

3.2.3.3 Diagrama de base de datos Microservicio Pedidos

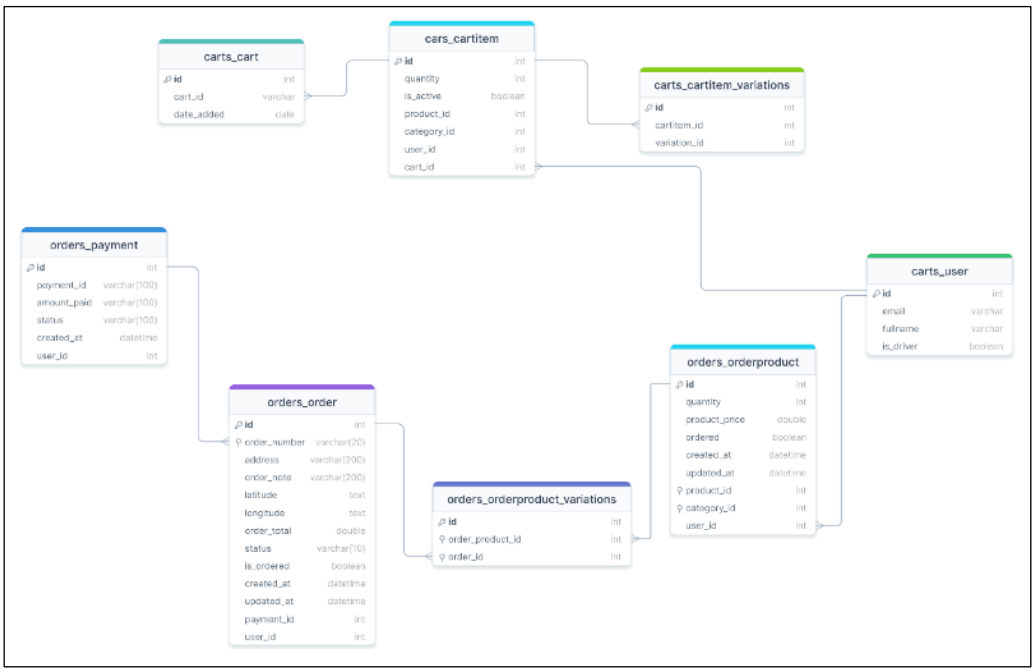


Figura 9: *Esquema conceptual de la base de datos del microservicio de pedidos*

Debido a que se adoptó una arquitectura de microservicios, se estableció la creación de bases de datos individuales para cada uno de ellos. Además, se tomó la decisión de utilizar nombres en inglés para las tablas y campos, siguiendo así las mejores prácticas de programación.

3.2.4 Interfaces Gráficas

Enseguida se presentan los modelos iniciales de las interfaces de usuario tanto para el sistema web como para el sistema móvil. Estos modelos fueron creados utilizando Mockplus y JustinMind como herramientas de diseño.

3.2.4.1 Sistema Web

Se presentan las distintas interfaces pertenecientes a cada microservicio en específico dentro del sistema Web, cabe mencionar que únicamente se presentan las interfaces más relevantes, de la plataforma de comercio electrónico.

Microservicio de Cuentas

- **Inicio de sesión**

  La pantalla inicial de la aplicación tendrá como propósito que los usuarios puedan ingresar sus credenciales con el fin de acceder a la plataforma.

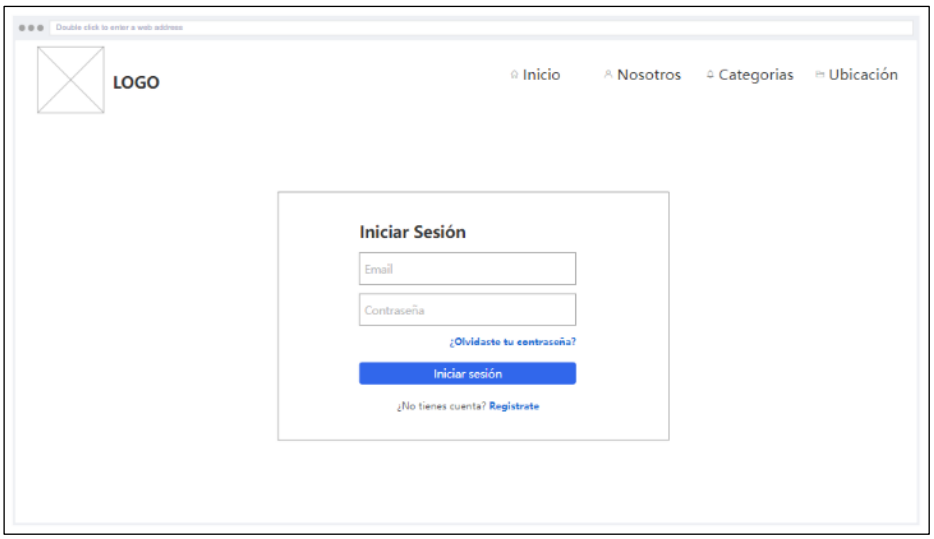


***Figura 10**: Prototipo de Interfaz Web - inicio de sesión*

- **Crear nueva cuenta**

  Esta interfaz permitirá a los usuarios ingresar su información personal como nombres, apellidos, correo electrónico, celular y crear una contraseña; de tal modo que al completar esta información se creará una cuenta totalmente nueva en la plataforma de comercio electrónico.

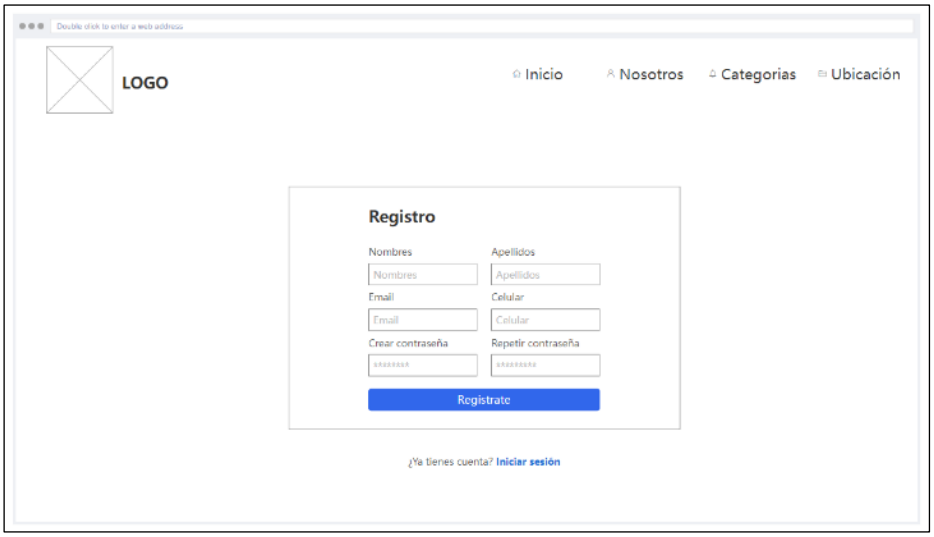


***Figura 11:** Prototipo de Interfaz Web - crear nueva cuenta*

- **Lista de Productos**

  En esta interfaz se exhiben de manera categorizada todos los productos que la empresa ATLAS ofrece, mostrando una imagen y el precio correspondiente a cada producto dentro de la plataforma de comercio electrónico.

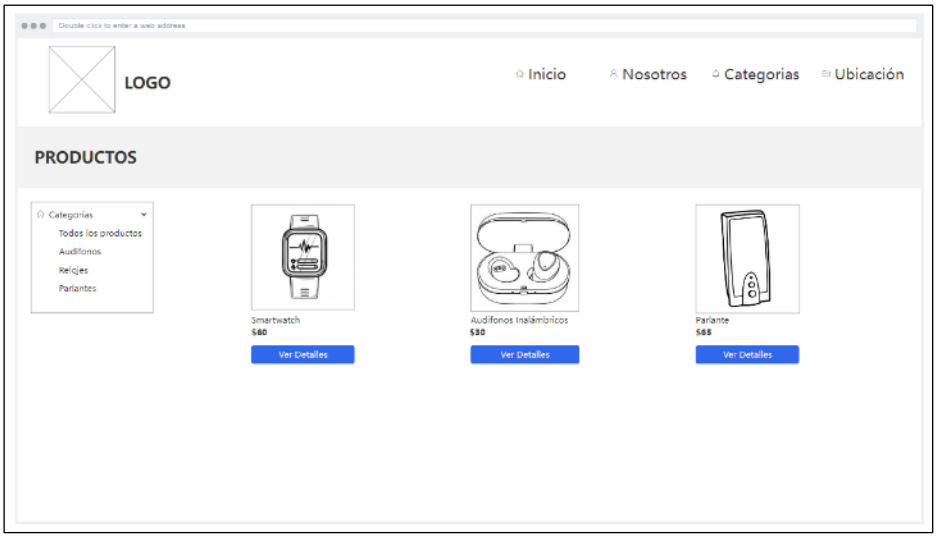


Figura 12: *Prototipo de Interfaz Web - lista de productos*

- **Detalle de un producto**

  Se trata de una interfaz que muestra la descripción detallada de un producto seleccionado junto con un carrusel de imágenes y el precio del producto seleccionado.

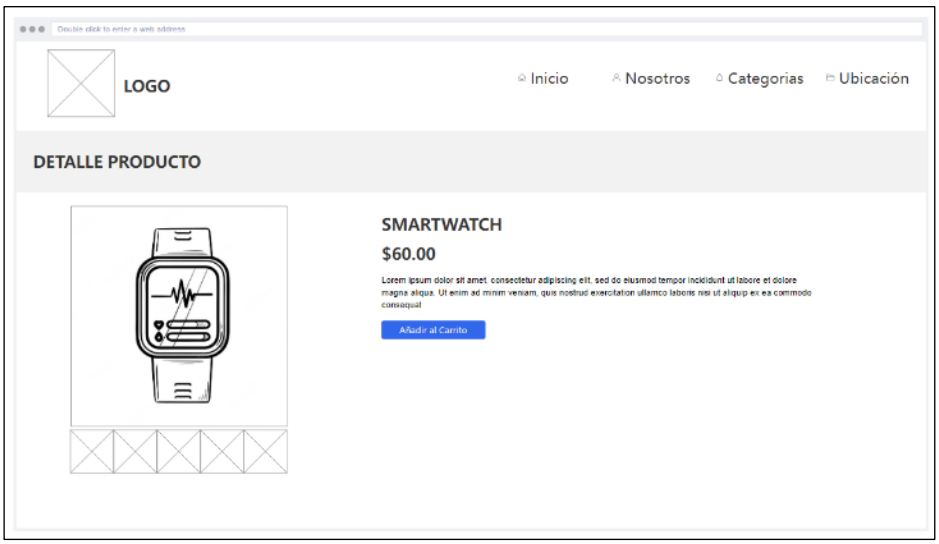


Figura 13: *Prototipo de Interfaz Web - detalle de un producto*

- **Validar Carrito**

  Dentro de esta interfaz, se puede visualizar los productos que están dentro del carrito de compras, también se puede modificar la cantidad de uno o varios productos dentro del carrito de compras. Los cálculos de precio, impuestos y envió se calculan automáticamente dentro del sistema.

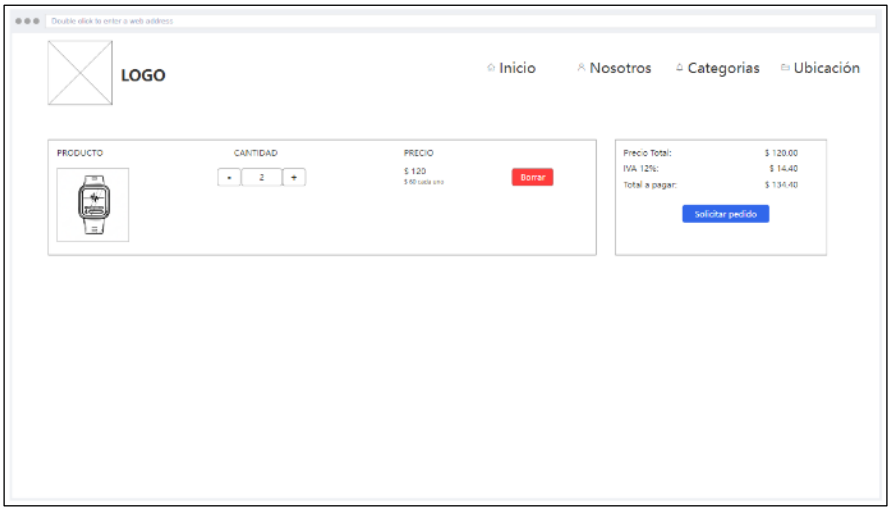


Figura 14: *Prototipo de Interfaz Web - validar carrito de compras*

- **Listar Pedidos**

  Esta interfaz estará disponible únicamente para el usuario con rol de «Administrador», quien tendrá la capacidad de visualizar los pedidos pendientes, mismos que debe entregar a sus clientes. En base a un criterio de su elección, el administrador asignará los pedidos al repartidor correspondiente para llevar a cabo la entrega del pedido en el domicilio del cliente.

Figura 15: *Prototipo de Interfaz Web - listar pedidos*

3.2.4.2 Sistema Móvil

Los prototipos del sistema móvil serán utilizados en smartphones o tablets únicamente con el sistema operativo de Android. Estos prototipos se muestran como ideas y opciones de diseño atractivas para los clientes y repartidores, donde el sistema móvil ha sido organizado en diferentes módulos que corresponden a las diversas interfaces que se han creado.

Módulo Gestionar Perfil

- **Acceder a cuenta de usuario**

  La primera interfaz de opciones de la aplicación permite al usuario elegir su rol, ya sea como cliente o como repartidor. Luego, redirige al usuario a la segunda interfaz de inicio de sesión, donde deberá ingresar sus credenciales para acceder al sistema móvil.

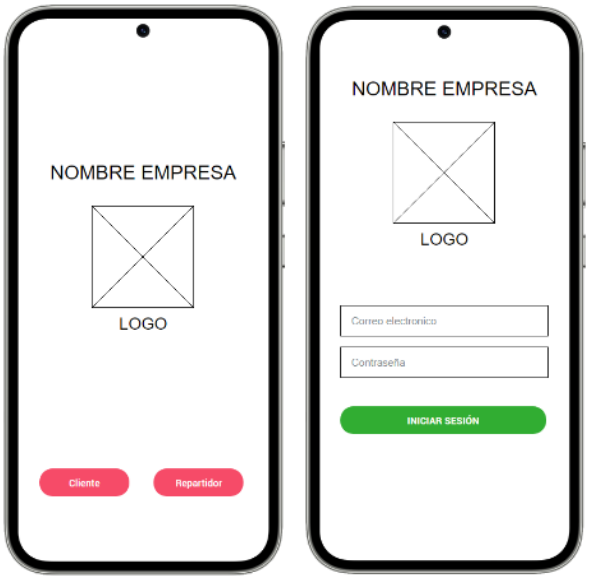


Figura 16: *Prototipo de Interfaz móvil - inicio de sesión*

- **Registrar usuario**

  En esta interfaz de usuario, el cliente tiene la opción de inscribirse proporcionando su información personal, como su nombre, apellido, correo electrónico, número de teléfono móvil y establecer una contraseña. Al completar el formulario, se generará automáticamente una cuenta nueva en la plataforma de comercio electrónico.

Figura 17: *Prototipo de interfaz móvil - registrar usuario*

Módulo Gestionar Catálogo

- **Consultar productos**

  La interfaz que se presenta a continuación exhibe una lista de los productos tecnológicos existentes en la tienda, con su respectiva imagen y precio, con el objetivo de permitir al cliente elegir los productos que se ajusten a sus requerimientos de compra.

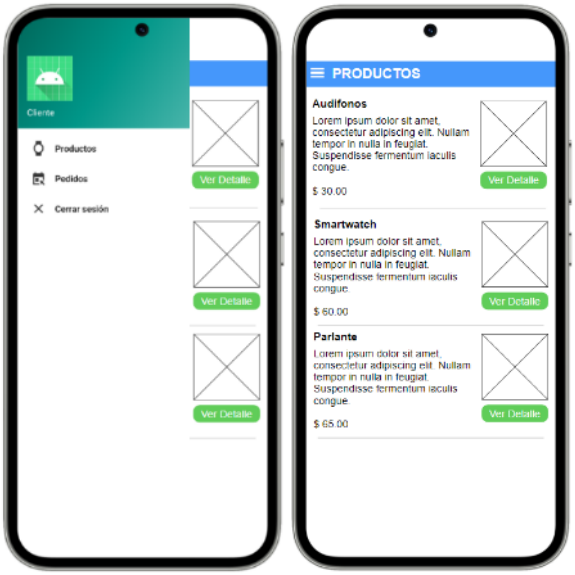


Figura 18: *Prototipo de interfaz móvil - consultar productos*

- **Detalle de producto**

  La siguiente interfaz muestra el detalle del producto seleccionado, al dar clic en el botón de “ver detalle” por cada producto existente.

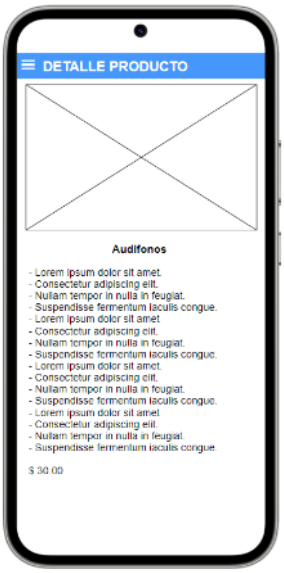


Figura 19: *Prototipo de interfaz móvil - detalle de producto*

Módulo Gestionar Pedidos

- **Crear pedido**

  Las siguientes interfaces permiten al cliente seleccionar la cantidad de productos que desea comprar, ingresar datos adicionales al pedido e ingresar la ubicación de donde desea recibir el pedido a domicilio.

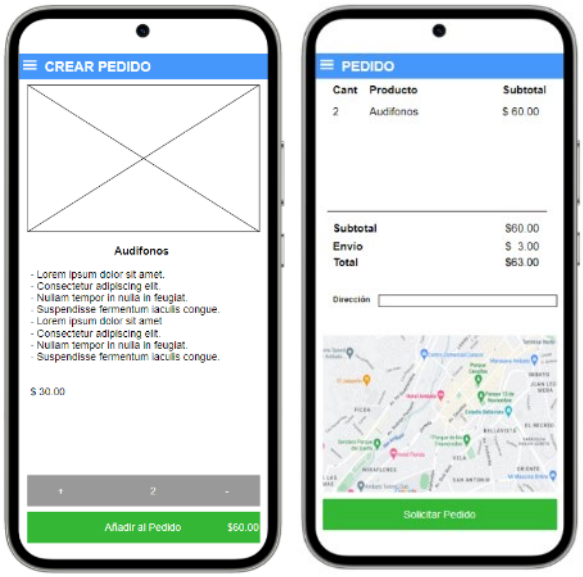


Figura 20:*Prototipo de interfaz móvil - crear pedido*

- **Visualizar estado del pedido**

  Esta interfaz muestra al cliente el trayecto que toma el repartidor hasta el domicilio del cliente, en la parte inferior se visualiza un mensaje con el estado del pedido (solicitado, en camino y entregado).

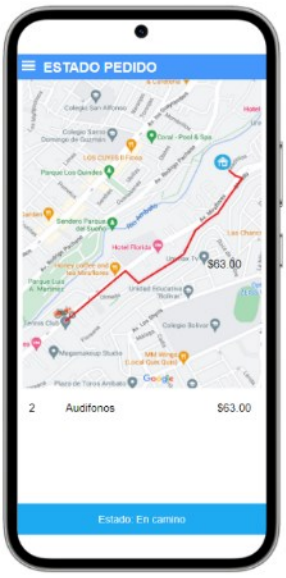


***Figura 21**:Prototipo de interfaz móvil - visualizar estado del pedido*

Módulo Gestionar Entregas

- **Lista de entregas**

  Esta interfaz está disponible solamente para usuarios que tengan el rol de "Repartidor". En dicha interfaz se muestran los pedidos que deben ser entregados, junto con la información del cliente, la dirección del cliente y los productos que se deben entregar.

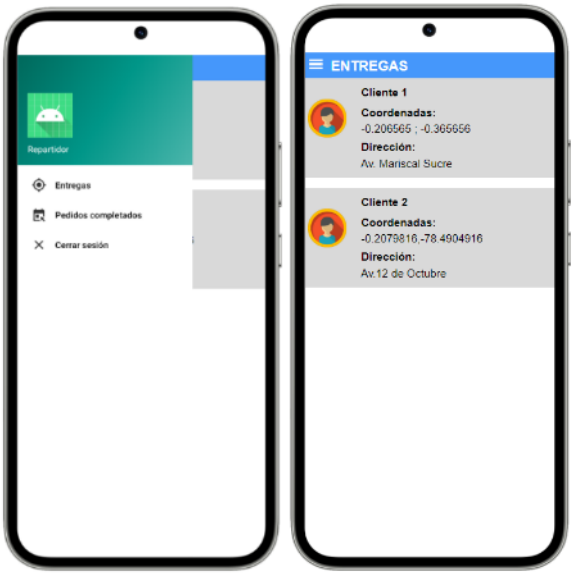


***Figura 22:**Prototipo de interfaz móvil - lista de entregas*

3.2.5 Diagrama de despliegue

El diagrama de despliegue presenta la interconexión existente entre los distintos elementos de la plataforma de comercio electrónico, como se puede observar en la siguiente imagen:

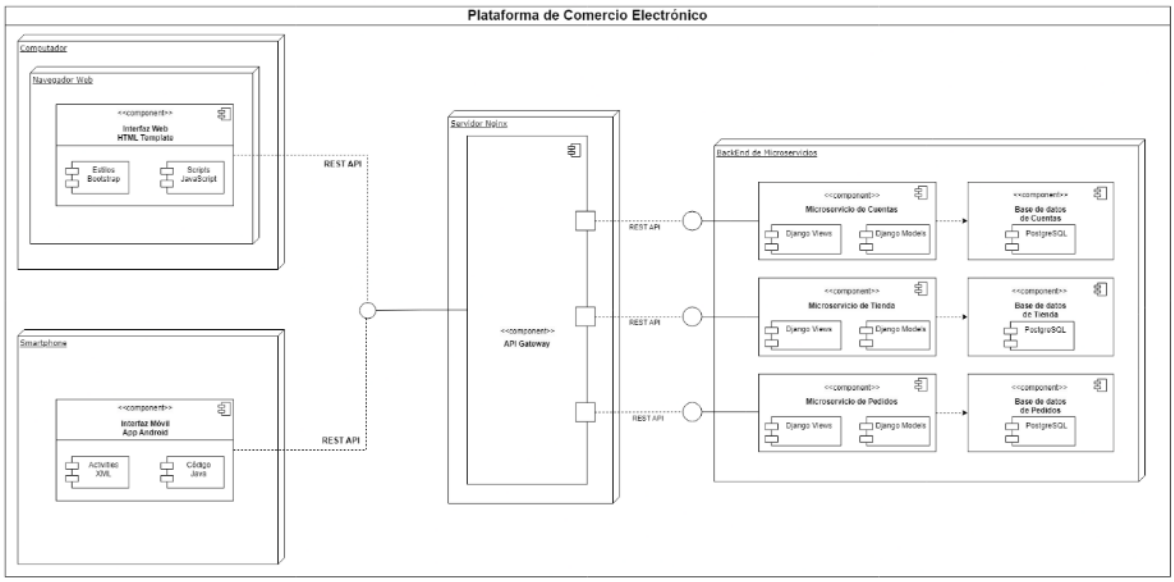


Figura 23: *Diagrama de despliegue de la plataforma de comercio electrónico*

3.3 Construcción

En esta sección se describe el desarrollo de la plataforma de comercio electrónico, siguiendo los criterios establecidos en las etapas de análisis y diseño, utilizando las herramientas y enfoques adecuados para crear un producto de calidad. Además, se presenta el modelo C4 de arquitectura de software en sus diferentes niveles de abstracción. La metodología empleada será Kanban, que una herramienta de gestión visual utilizada en el desarrollo de proyectos de software para visualizar y gestionar el flujo de trabajo en un tablero. Es especialmente útil en proyectos complejos, debido a que permite a los equipos de desarrollo monitorear y controlar el progreso de las tareas, identificar cuellos de botella y gestionar la carga de trabajo.

3.3.1 Vistas Modelo C4

Se empleará el modelo C4 de arquitectura de software, que es una metodología gráfica normalizada utilizada para representar la estructura de sistemas de software de manera detallada. Este modelo consta de una serie de diagramas jerárquicos de arquitectura de software que abarcan diferentes niveles de abstracción, incluyendo el contexto, los contenedores, los componentes y el código. Cada nivel de abstracción es relevante para una audiencia específica, y en conjunto, los diagramas C4 ofrecen una visión completa y detallada de la arquitectura de software (Vivanco, 2019).

3.3.1.1 Vista Diagrama de Contexto

El diagrama de contexto es una herramienta importante en el diseño de sistemas de software, ya que permite una visión general del sistema y su relación con el mundo real. En el desarrollo de la plataforma de comercio electrónico, el diagrama de contexto muestra cómo el sistema interactúa con los diferentes tipos de usuarios que utilizarán todo el sistema. Además, el diagrama de contexto también puede ser utilizado para identificar posibles problemas de diseño o implementación, como la necesidad de integrar sistemas de terceros o mejorar la navegación del sitio web.

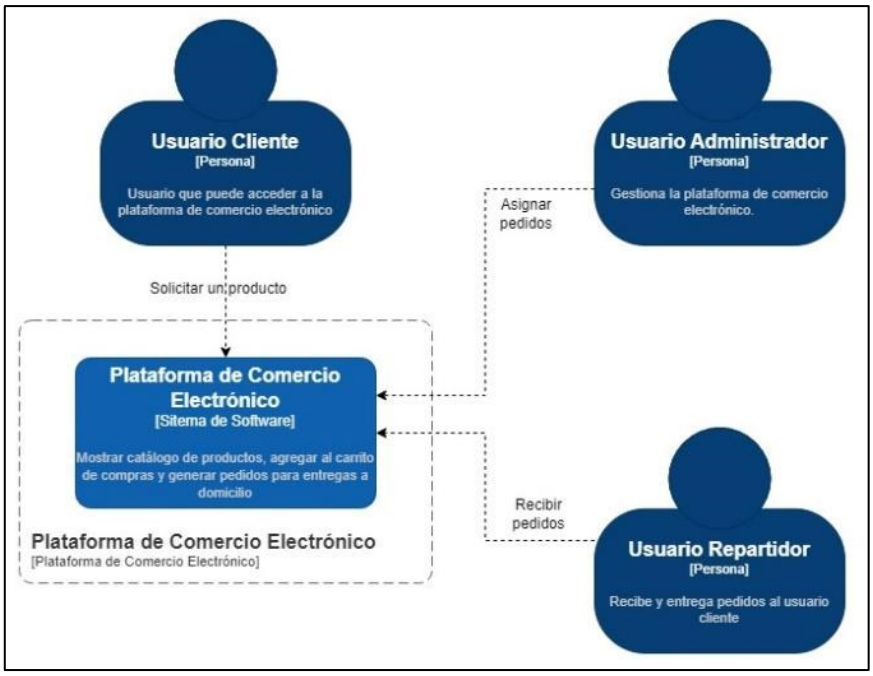


Figura 24: *Diagrama de Contexto General*

- **Usuario Cliente:** Persona que puede acceder a la plataforma de comercio electrónico, seleccionar productos, generar un pedido y recibir el pedido en su domicilio.

Figura 25: *Diagrama de Contexto - Usuario Cliente*

- **Usuario Administrador:** Persona que gestiona la plataforma de comercio electrónico, también (ingresa, edita y elimina) el catálogo de productos recibe pedidos y asigna pedidos al repartidor para la entrega de productos a domicilio.

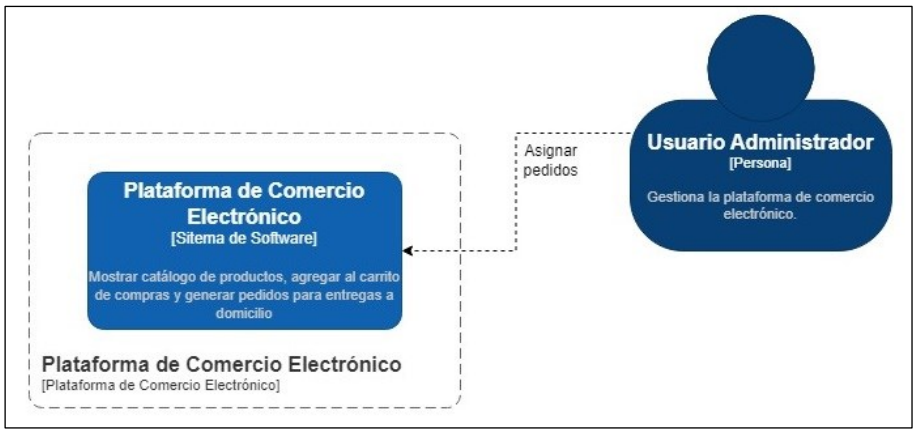


Figura 26: *Diagrama de Contexto - Usuario Administrador*

- **Usuario Repartidor:** Persona encargada de recibir los pedidos del usuario administrador en la plataforma de comercio electrónico, los cuales deben ser entregados en la dirección del usuario cliente.

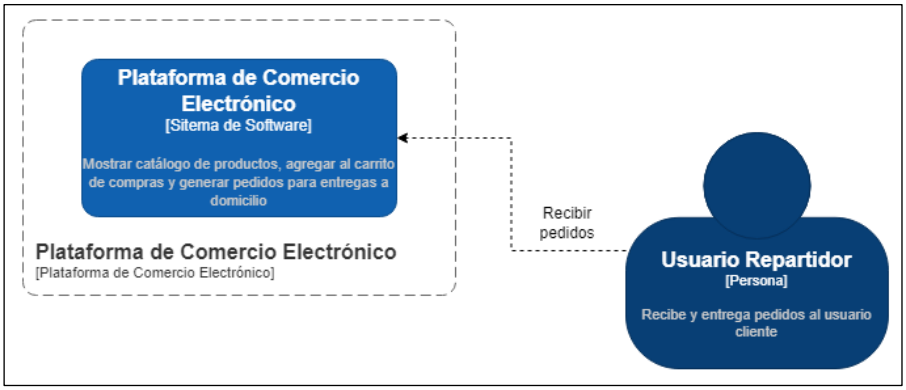


Figura 27: *Diagrama de Contexto - Usuario Repartidor*

3.3.1.2 Vista Diagrama de Contenedor

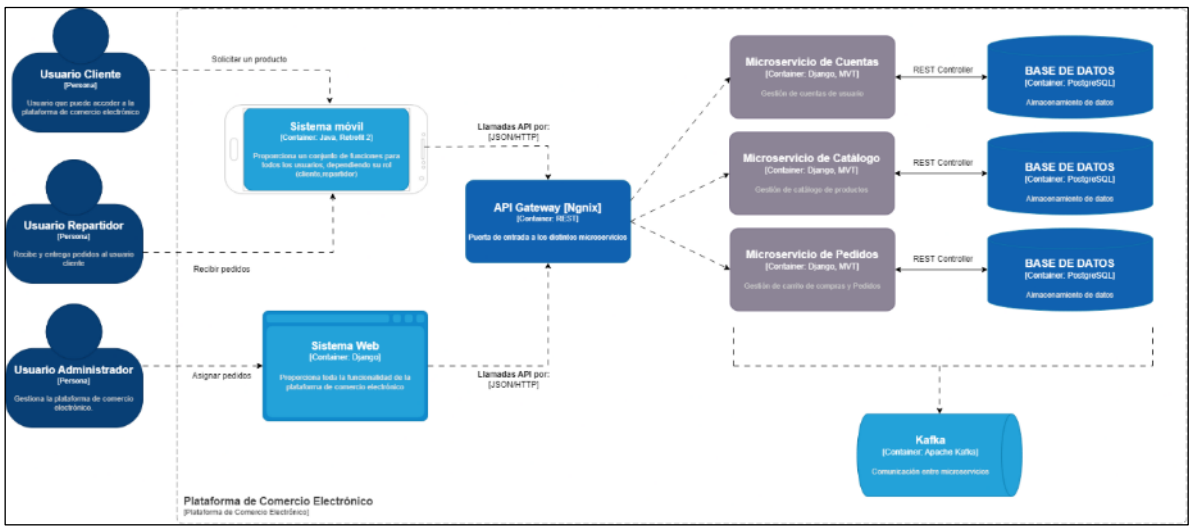


Figura 28:*Diagrama de Contenedor C4*

3.3.1.3 Vista Diagrama de Componentes

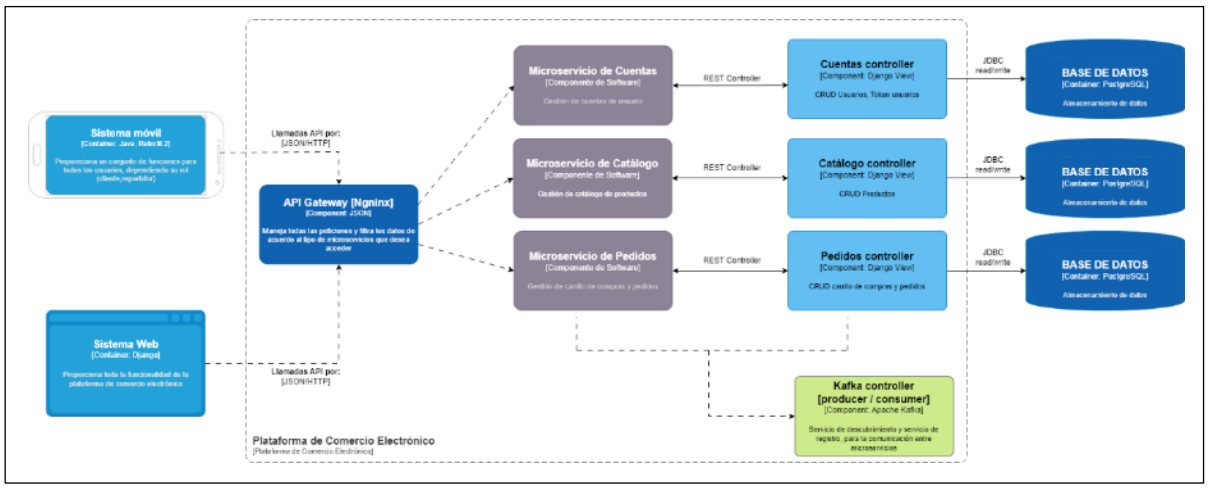


Figura 29: *Diagrama de Componentes*

3.3.2 Código del Sistema

A continuación, se detallará el código que fue elaborado para el cumplimiento de los requisitos funcionales y no funcionales de toda la plataforma, donde se muestran los componentes más importantes de la programación de la plataforma de comercio electrónico.

Es importante destacar que la creación de los microservicios se llevó a cabo utilizando Python 3.8 y Django Framework 4.2.6. Para la base de datos de cada microservicio se eligió PostgreSQL 12. El API Gateway se implementó mediante Nginx 1.24.0. En cuanto al Frontend del sistema web, se optó por las Templates de Django Framework. Por otro lado, el Frontend del sistema móvil se desarrolló con Java versión 17, empleando Android Studio como IDE. Respecto a la metodología de desarrollo, se aplicó Kanban, lo que implicó abordar los requerimientos de manera individual y trasladar las tareas finalizadas en el tablero Kanban. Por ello, se decidió documentar únicamente las secciones más cruciales de toda la plataforma, divididas en los siguientes componentes:

- Sistema Web
    - Backend Microservicio de Cuentas
    - Backend Microservicio de Tienda
    - Backend Microservicio de Órdenes
- API Gateway
- Endpoints de comunicación
- Sistema Móvil

3.3.2.1 Sistema Web

Esta sección está organizada alrededor de los microservicios más cruciales que necesitan ser configurados y codificados para garantizar la operatividad efectiva de la arquitectura de microservicios.

3.3.2.2 Backend Microservicio de Cuentas

Este apartado se presenta dividido en los elementos clave que se programaron para desarrollar el Microservicio de Cuentas.

- **Dockerfile**

Este Dockerfile configura un entorno Python 3.8 para una aplicación Django, instalando dependencias y preparando el servidor de desarrollo para ejecutarse en el puerto 8000. En la Figura No. 30 se exhibe los parámetros de configuración del Dockerfile.

```
1  FROM python:3.8
2  ENV PYTHONUBBUFFERED=1
3  WORKDIR /app
4  COPY requirements.txt /app/requirements.txt
5  RUN pip install -r requirements.txt
6  COPY . /app
7  CMD python manage.py runserver 0.0.0.0:8000
```

Figura 30: *Dockerfile del Microservicios de Cuentas*

- **Docker-compose.yml**

Este archivo se encarga de configurar múltiples servicios, donde se detalla el servicio de backend de Django llamado «backend_account», que usa una imagen personalizada construida desde el Dockerfile y colocada en el directorio actual, asignándole un nombre de contenedor junto con la configuración del volumen para medios y mapeando el puerto 8000. Se añade una base de datos PostgreSQL llamada «db», un servicio de colas de eventos con Apache Kafka «queue_auth» y un almacén de datos en memoria «redis». Cada servicio está configurado con sus propios parámetros, como nombres de contenedores, puertos y volúmenes persistentes, y todos se interconectan a través de una red de microservicios definida como externa, lo que facilita la comunicación y el aislamiento entre ellos. Estas configuraciones se muestran en la Figura No. 31.

```yaml
version: '3.8'
services:
  # Django App
  backend_account:
    container_name: paul_ms_account
    build: .
    volumes:
      - .:/app
      - media_volume:/app/media
    ports:
      - "8000:8000"
    depends_on:
      - db
      - queue_auth
      - redis
    networks:
      - microservices-network

  # Event Queue Kafka
  queue_auth:
    build: .
    command: >
      sh -c "python consumer.py"
    depends_on:
      - db
    networks:
      - microservices-network

  # Database PostgreSQL
  db:
    image: postgres
    container_name: account_api_postresql
    restart: always
    environment:
      POSTGRES_DB: account_dbms
      POSTGRES_USER: paul
      POSTGRES_PASSWORD: 1234
    ports:
      - "5432:5432"
    volumes:
      - .dbdata:/var/lib/postgresql
    networks:
      - microservices-network

  redis:
    image: redis
    container_name: django_auth_api_redis
    ports:
      - "6379:6379"
    networks:
      - microservices-network

volumes:
  media_volume:

networks:
  microservices-network:
    external: true
```

Figura 31: *Docker-compose para el Microservicio de Cuentas*

- **Consumer.py**

La Figura No. 32 muestra un archivo que es esencial debido a que se conecta al servidor Kafka como consumidor, escucha un tópico específico y procesa los mensajes entrantes. Está diseñado para manejar la creación de usuarios y reportar errores si ocurren durante el procesamiento de los mensajes.

```python
import json
import os
import django
from confluent_kafka import Consumer
from rest_framework.exceptions import ValidationError
from decouple import config
import sys

# Configuración de Django
os.environ.setdefault("DJANGO_SETTINGS_MODULE", "core.settings")
django.setup()

print("consumidor auth")
print(config('KAFKA_BOOTSTRAP_SERVER'))

# Configuración del consumidor de Kafka
consumer = Consumer({
    'bootstrap.servers': config('KAFKA_BOOTSTRAP_SERVER'),
    'security.protocol': config('KAFKA_SECURITY_PROTOCOL'),
    'sasl.username': config('KAFKA_USERNAME'),
    'sasl.password': config('KAFKA_PASSWORD'),
    'sasl.mechanism': 'PLAIN',
    'group.id': config('KAFKA_GROUP'),
    'auto.offset.reset': 'earliest'
})

# Suscribirse al tema de Kafka
consumer.subscribe([config('KAFKA_TOPIC')])

# Bucle de consumo de mensajes
while True:
    msg = consumer.poll(1.0)

    if msg is None:
        continue

    if msg.error():
        print("Consumer error: {}".format(msg.error()))
        continue

    # Procesar mensajes de Kafka
    topic = msg.topic()
    value = msg.value()
    data = json.loads(value)

    # Logica de procesamiento de mensajes uuuu

    print(f'Got this message with Topic: {topic} and value: {value}, with Data: {data}')
    sys.stdout.flush()

    # O realizar alguna acción específica basada en el contenido del mensaje
    if topic == config('KAFKA_TOPIC'):
        if msg.key() == b'create_user':
            try:
                print(f"usuario creado desde account {data['id']}")
            except ValidationError as e:
                print(f"Fallo al crear usuario desde account con ID {data['id']}: {str(e)}")

#Cierra el consumidor al salir del bucle
consumer.close()
```

Figura 32: *Consumidor de mensajes Kafka dentro del Microservicio de Cuentas*

- **Producer.py**

Este archivo se encarga de producir los mensajes que serán dirigidos a Kafka desde el microservicio de Cuentas hacia los demás microservicios, los detalles de esta configuración se muestran en la Figura No. 33.

```python
from confluent_kafka import Producer
import os


def delivery_report(err, msg):
    if err is not None:
        print('Error enviado el mensaje: {}'.format(err))
    else:
        print('Mensaje enviado a {} [{}]'.format(msg.topic(), msg.partition()))


producer = Producer({
    'bootstrap.servers': os.environ.get('KAFKA_BOOTSTRAP_SERVER'),
    'security.protocol': os.environ.get('KAFKA_SECURITY_PROTOCOL'),
    'sasl.username': os.environ.get('KAFKA_USERNAME'),
    'sasl.password': os.environ.get('KAFKA_PASSWORD'),
    'sasl.mechanism': 'PLAIN',
})
```

Figura 33: *Productor del Microservicio de Cuentas*

- **Middleware.py**

La Figura No. 34 muestra, como se creó un middleware personalizado en Django, diseñado para manejar las solicitudes CORS (Cross-Origin Resource Sharing) al añadir cabeceras que permiten el acceso desde cualquier origen y aceptan cualquier cabecera. Esta configuración permite la comunicación entre los distintos microservicios.

```python
# core/middleware.py

class CustomCorsMiddleware:
    def __init__(self, get_response):
        self.get_response = get_response

    def __call__(self, request):
        response = self.get_response(request)
        response["Access-Control-Allow-Origin"] = "*"
        response["Access-Control-Allow-Headers"] = "*"
        return response
```

Figura 34: *Middleware del Microservicio de Cuentas*

- **Settings.py**

Como se aprecia en la Figura No. 35, el archivo settings.py actúa como la configuración central de Django, e incluye una modificación crucial que facilita la interacción entre los diversos microservicios. Esta alteración clave es el uso de una SECRET_KEY uniforme, que se comparte entre todos los microservicios para asegurar la cohesión los mismos. Adicionalmente, se ha configurado adecuadamente el manejo de CORS para permitir una integración segura y eficiente entre los diferentes microservicios. Finalmente se aprecia la instalación de Django REST Framework.

```python
SECRET_KEY = env('SECRET_KEY')

MIDDLEWARE = [
    'corsheaders.middleware.CorsMiddleware',
    'whitenoise.middleware.WhiteNoiseMiddleware',
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
    'core.middleware.CustomCorsMiddleware',
]
SESSION_EXPIRE_SECONDS = 3600  # 1 hour
SESSION_EXPIRE_AFTER_LAST_ACTIVITY = True
SESSION_TIMEOUT_REDIRECT = 'home'

# REST_FRAMEWORK
REST_FRAMEWORK = {
    "NON_FIELD_ERRORS_KEY": "errors",
    "DEFAULT_AUTHENTICATION_CLASSES": (
        "rest_framework.authentication.SessionAuthentication",
        "rest_framework.authentication.TokenAuthentication",
    ),
    "DEFAULT_PERMISSION_CLASSES": (
        "rest_framework.permissions.IsAuthenticated",
    ),
    "DEFAULT_RENDERER_CLASSES": [
        "rest_framework.renderers.JSONRenderer",
    ],
}
```

Figura 35: *Parámetros de configuración para la comunicación entre Microservicios*

- **Account/Models.py**

En Django, los models son una parte esencial del framework que proporciona una forma de definir la estructura de la base de datos de manera abstracta y orientada a objetos. Django posee su propio ORM (Object-Relational Mapping), que permite a los desarrolladores crear, recuperar, actualizar y borrar registros sin necesidad de escribir consultas SQL rigurosas. Como se visualiza en la Figura No. 36, la clase «Account» es una versión personalizada del modelo de usuario en Django que hereda de «AbstractUser». Incluye campos estándar y personalizados como nombre, apellido, correo electrónico y teléfono, así como indicadores booleanos para roles y permisos. Establece el email como identificador principal y ofrece métodos para obtener el nombre completo y representar al usuario como una cadena de su email.

```python
class Account(AbstractUser):
    first_name = models.CharField(max_length=50, verbose_name="Nombre")
    last_name = models.CharField(max_length=50, verbose_name="Apellido")
    username = models.CharField(max_length=50, unique=True, verbose_name="Usuario", null=True, blank=True)
    email = models.EmailField(max_length=50, unique=True, verbose_name="Email")
    phone_number = models.CharField(max_length=50, verbose_name="Telefono")
    date_joined = models.DateTimeField(auto_now_add=True, verbose_name="Fecha de registro")
    last_login = models.DateTimeField(auto_now_add=True, verbose_name="Último acceso")
    is_admin = models.BooleanField(default=False, verbose_name="Administrador")
    is_staff = models.BooleanField(default=False, verbose_name="Personal")
    is_active = models.BooleanField(default=False, verbose_name="Activado")
    is_superadmin = models.BooleanField(default=False, verbose_name="SuperAdministador")
    # Agrega el campo is_driver
    is_driver = models.BooleanField(default=False, verbose_name="Conductor")  # Se define false predeterminado
    location = models.CharField(max_length=200, blank=True, null=True, verbose_name="Ubicación del conductor")

    USERNAME_FIELD = 'email'
    REQUIRED_FIELDS = ['username', 'first_name', 'last_name']

    objects = MyAccountManager()

    def get_full_name(self):
        return f'{self.first_name} {self.last_name}'

    def __str__(self):
        return self.email

    # metodos mandatorios
    def has_perm(self, perm, obj=None):
        return self.is_admin

    def has_module_perms(self, app_label):
        return True
```

Figura 36: *Modelo Account - Microservicio Cuentas*

- **Account/Serializers.py**

Un serializador en Django es una clase que convierte datos complejos, como instancias de modelos y consultas de bases de datos, en tipos de datos nativos de Python que luego pueden ser fácilmente renderizados en JSON, XML u otros formatos de contenido. Para ejemplificar este concepto se presenta la clase «AccountSerializer» que es un serializador de modelo en Django que se utiliza para convertir instancias del modelo «Account» a formato JSON.

```python
class AccountSerializer(serializers.ModelSerializer):
    userprofile = UserProfileSerializer()

    class Meta:
        model = Account
        fields = ['id', 'first_name', 'last_name', 'username', 'email', 'phone_number',
                  'date_joined', 'last_login', 'is_admin', 'is_staff', 'is_active',
                  'is_superadmin', 'is_driver', 'userprofile']
```

Figura 37: *Serializador AccountSerializer - Microservicio Cuentas*

- **Account/Views.py**

Las vistas en Django acceden a los datos necesarios para satisfacer la solicitud, a través de modelos, y delegan la respuesta a plantillas o serializadores para su presentación al usuario. En este fragmento de código vemos que la Figura No. 38 contiene la clase «CustomAuthToken» que envía una petición POST, la misma que autentica al usuario utilizando los datos proporcionados, genera un token de acceso y luego devuelve una respuesta con el token y un campo adicional is_driver que indica si el usuario es conductor. La respuesta es enviada con un estado HTTP 200 OK.

```python
class CustomAuthToken(ObtainAuthToken):
    def post(self, request, *args, **kwargs):
        # Realiza la autenticación del usuario
        serializer = self.serializer_class(data=request.data, context={'request': request})
        serializer.is_valid(raise_exception=True)
        user = serializer.validated_data['user']

        # Genera el token
        token, created = Token.objects.get_or_create(user=user)

        # Agrega el campo 'is_driver' al diccionario de datos de respuesta
        data = {
            'token': token.key,
            'is_driver': user.is_driver,
        }

        return Response(data, status=status.HTTP_200_OK)
```

Figura 38: *Obtención del Token de un Usuario - Microservicio Cuentas*

Después de llevar a cabo la implementación de los requisitos del Microservicio de Cuentas, se procedió con la fase de pruebas, donde dichos requisitos se trasladan de la columna « En proceso » a la columna de « Pruebas ». Los detalles de la funcionalidad de este microservicio se encuentran en el capítulo 4. La Figura No. 39 exhibe el panel Kanban definitivo correspondiente a este microservicio.

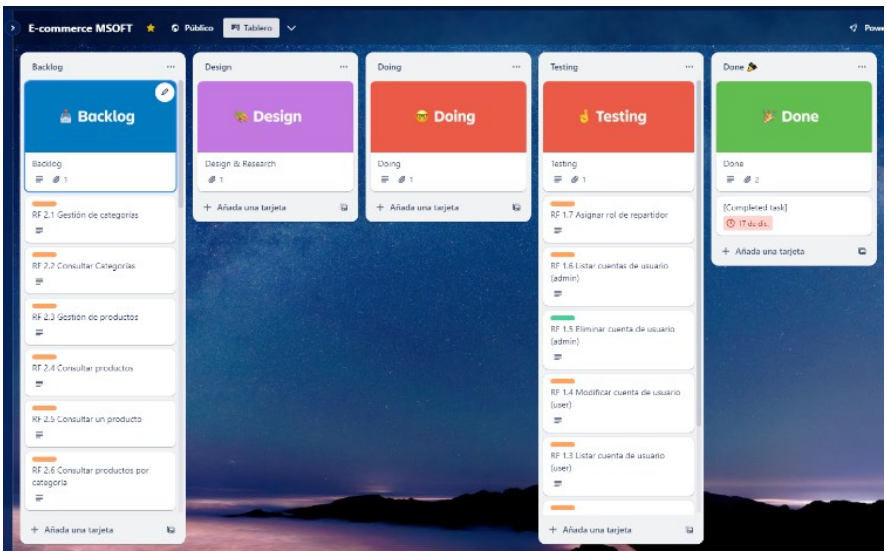


Figura 39: *Imagen del panel Kanban para el Microservicio de Cuentas*

3.3.2.3 Backend Microservicio de Tienda

Esta sección está organizada en torno a los componentes clave que se programaron para el desarrollo del Microservicio de Tienda.

- **Dockerfile**

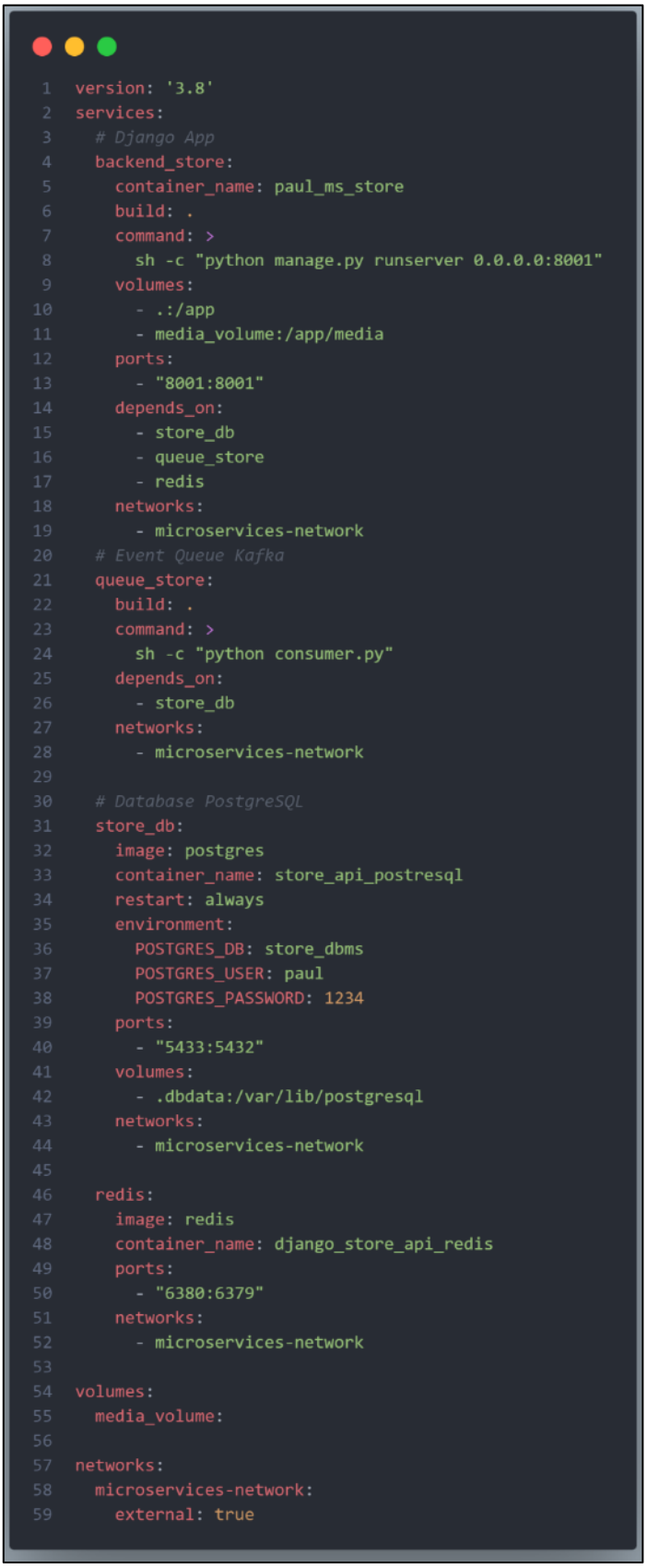
El Dockerfile presentado establece un ambiente con Python 3.8 para la aplicación Django correspondiente al Microservicio de Tienda, donde se gestionan las dependencias necesarias y se configura el servidor de desarrollo para operar en el puerto 8001, a diferencia del Microservicio de Cuentas que corre en el puerto 8000. La configuración detallada de este Dockerfile se muestra en la Figura No. 40.

```
1  FROM python:3.8
2  ENV PYTHONUBBUFFERED=1
3  WORKDIR /app
4  COPY requirements.txt /app/requirements.txt
5  RUN pip install -r requirements.txt
6  COPY . /app
7  CMD python manage.py runserver 0.0.0.0:8001
```

Figura 40: *Dockerfile del Microservicios de Tienda*

- **Docker-compose.yml**

El fichero docker-compose.yml configura un entorno de microservicio, donde se define un servicio llamado «backend_store» que construye una imagen Docker y corre un servidor de desarrollo Django en el puerto 8001. También configura servicios dependientes como «store_db» para la base de datos PostgreSQL, «queue_store» para la cola de eventos Kafka y redis para el almacenamiento en caché, asegurando su interconexión a través de una red común llamada microservices-network. Además, especifica volúmenes para el código del aplicativo y para los archivos multimedia, con el objetivo de manejar la persistencia de datos. Este entorno facilita el desarrollo, despliegue y escalabilidad del Microservicio de Tienda. Dichas configuraciones se muestran en la Figura No. 41.



```yaml
version: '3.8'
services:
  # Django App
  backend_store:
    container_name: paul_ms_store
    build: .
    command: >
      sh -c "python manage.py runserver 0.0.0.0:8001"
    volumes:
      - .:/app
      - media_volume:/app/media
    ports:
      - "8001:8001"
    depends_on:
      - store_db
      - queue_store
      - redis
    networks:
      - microservices-network
  # Event Queue Kafka
  queue_store:
    build: .
    command: >
      sh -c "python consumer.py"
    depends_on:
      - store_db
    networks:
      - microservices-network

  # Database PostgreSQL
  store_db:
    image: postgres
    container_name: store_api_postresql
    restart: always
    environment:
      POSTGRES_DB: store_dbms
      POSTGRES_USER: paul
      POSTGRES_PASSWORD: 1234
    ports:
      - "5433:5432"
    volumes:
      - .dbdata:/var/lib/postgresql
    networks:
      - microservices-network

  redis:
    image: redis
    container_name: django_store_api_redis
    ports:
      - "6380:6379"
    networks:
      - microservices-network

volumes:
  media_volume:

networks:
  microservices-network:
    external: true
```

Figura 41: *Docker-compose para el Microservicio de Tienda*

- **Consumer.py**

En la Figura No. 42, se muestra un archivo esencial porque está diseñado para funcionar como consumidor de Kafka en el contexto de un microservicio de tienda. Configura la conexión a Kafka, se suscribe a un tópico específico y define funciones para recibir la creación de usuarios y gestionar tokens de autenticación. Las funciones incluyen la creación de un nuevo usuario en la base de datos, la asignación y actualización de tokens de usuario, y la eliminación de tokens y datos de caché asociados con un usuario provenientes del Microservicio de Cuentas.

```python
from decouple import config
import sys
from contextlib import closing
from django.core.exceptions import ObjectDoesNotExist
from django.core.cache import cache

# Configuración de Django
os.environ.setdefault("DJANGO_SETTINGS_MODULE", "core.settings")
django.setup()

print("consumidor store")
print(config('KAFKA_BOOTSTRAP_SERVER'))

# Configuración del consumidor de Kafka
consumer = Consumer({
    'bootstrap.servers': config('KAFKA_BOOTSTRAP_SERVER'),
    'security.protocol': config('KAFKA_SECURITY_PROTOCOL'),
    'sasl.username': config('KAFKA_USERNAME'),
    'sasl.password': config('KAFKA_PASSWORD'),
    'sasl.mechanism': 'PLAIN',
    'group.id': config('KAFKA_GROUP'),
    'auto.offset.reset': 'earliest'
})

# Suscribirse al tema de Kafka
consumer.subscribe([config('KAFKA_TOPIC')])

# Create user in microservice of Cart
from django.apps import apps

User = apps.get_model('carts', 'User')
UsuarioToken = apps.get_model('carts', 'UsuarioToken')


def create_user(data):
    try:
        User.objects.create(
            id=data['id'],
            fullname=data['fullname'],
            email=data['email'],
            is_driver=data['is_driver']
        )
        print(f"User created successfully in function")
        sys.stdout.flush()
    except Exception as e:
        print(f"Failed to create user: {str(e)}")
        sys.stdout.flush()


def asignarToken(id, token_key):
    try:
        usuario_ms = User.objects.get(id=id)
        token, created = UsuarioToken.objects.update_or_create(
            user_id=usuario_ms,
            defaults={'key': token_key}
        )
        action = "creado" if created else "actualizado"
        print(f"Token {action} para el usuario {id}")
        sys.stdout.flush()
    except User.DoesNotExist:
        print(f"Usuario no encontrado: {id}")
        sys.stdout.flush()
        # Aquí podrías considerar crear el usuario si es necesario
    except Exception as e:
        print(f"Error al asignar token: {str(e)}")
        sys.stdout.flush()


def handle_logout(user_id):
    try:
        # Eliminar el token asociado al usuario
        UsuarioToken.objects.filter(user_id=user_id).delete()
        cache.delete('user_id_' + str(user_id))  # Asegurándose de eliminar también de la caché
        print(f"Token y caché eliminados para el usuario {user_id}")
        sys.stdout.flush()
    except Exception as e:
        print(f"Error al eliminar el token para el usuario {user_id}: {str(e)}")
        sys.stdout.flush()
```

Figura 42: *Consumidor de mensajes Kafka dentro del Microservicio de Tienda*

- **Middleware.py**

La Figura No. 43 señala la creación de un middleware a medida en Django que se ocupa de las solicitudes CORS, añadiendo cabeceras que habilitan el acceso desde todos los orígenes y la aceptación de todas las cabeceras. Este ajuste facilita la interacción entre varios microservicios.

```python
# core/middleware.py

class CustomCorsMiddleware:
    def __init__(self, get_response):
        self.get_response = get_response

    def __call__(self, request):
        response = self.get_response(request)
        response["Access-Control-Allow-Origin"] = "*"
        response["Access-Control-Allow-Headers"] = "*"
        return response
```

Figura 43: *Middleware del Microservicio de Tienda*

- **Settings.py**

Como se aprecia en la Figura No. 44, el archivo settings.py muestra la configuración del Microservicio de Tienda en Django, incluyendo claves secretas y parámetros de CORS para permitir todas las solicitudes de origen cruzado. Configura las credenciales de la base de datos PostgreSQL y establece la autenticación y permisos predeterminados del framework REST de Django. También se especifica una autenticación personalizada para manejar los tokens de usuario.

```python
SECRET_KEY = env('SECRET_KEY')
CORS_ORIGIN_ALLOW_ALL = True
CORS_ALLOW_CREDENTIALS = True
CORS_ALLOW_ALL_ORIGINS = True
CORS_ORIGIN_WHITELIST = os.environ.get('CORS_ORIGIN_WHITELIST_DEV').split(',')
CSRF_TRUSTED_ORIGINS = os.environ.get('CSRF_TRUSTED_ORIGINS_DEV').split(',')
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': 'store_dbms',
        'USER': 'paul',
        'PASSWORD': '1234',
        'HOST': 'store_db',
        'PORT': '5432',
    }
}
# REST_FRAMEWORK
REST_FRAMEWORK = {
    "NON_FIELD_ERRORS_KEY": "errors",
    "DEFAULT_AUTHENTICATION_CLASSES": (
        "rest_framework.authentication.TokenAuthentication",
        "orders.authentication.CustomTokenAuthentication",
    ),
    "AUTH_TOKEN_MODEL": "carts.UsuarioToken",
    "DEFAULT_PERMISSION_CLASSES": (
        "rest_framework.permissions.AllowAny",
    ),
    "DEFAULT_RENDERER_CLASSES": [
        "rest_framework.renderers.JSONRenderer",
    ],
}
```

Figura 44: *Archivo de configuración del Microservicio de Tienda*

- **Store/Models.py**

Dentro de la Figura No. 45, la clase «Product» es la representación de los productos que se ofrecen en la tienda con campos para el nombre, slug único, descripción enriquecida, precio, imagen, stock, disponibilidad y una relación de clave foránea con una categoría. También rastrea las fechas de creación y modificación del producto. El modelo incluye métodos para devolver el nombre del producto como una cadena y para obtener la URL del detalle del producto.

```python
class Product(models.Model):
    product_name = models.CharField(max_length=200, unique=True, verbose_name="Nombre Producto")
    slug = models.SlugField(max_length=200, unique=True, verbose_name="Slug")
    description = RichTextField(max_length=2000, blank=True, verbose_name="Descripción")
    price = models.IntegerField(verbose_name="Precio")
    images = models.ImageField(upload_to='photos/products', blank=True, verbose_name="Imágenes")
    stock = models.IntegerField(verbose_name="Stock")
    is_available = models.BooleanField(default=True, verbose_name="Disponible")
    category = models.ForeignKey(Category, on_delete=models.CASCADE,
                                 verbose_name="Categoria")  # Relation 1 a N con tabla category
    created_date = models.DateTimeField(auto_now_add=True, verbose_name="Fecha de creación")
    modified_date = models.DateTimeField(auto_now=True, verbose_name="Fecha de modificación")

    class Meta:
        verbose_name = ("producto")
        verbose_name_plural = ("productos")

    def __str__(self):
        return self.product_name

    def get_url(self):
        # Llamamos al nombre de la URL que se encuentra en store/urls.py
        return reverse("products_detail", args=[self.category.slug, self.slug])
```

Figura 45: *Modelo Product - Microservicio Tienda*

- **Store/Serializers.py**

La clase «ProductSerializer» es un serializador que transforma la información del modelo «Product» en un formato JSON, incluyendo detalles como el nombre del producto, descripción, precio y otros campos relevantes. Como se aprecia en la Figura No. 46.

```python
class ProductSerializer(serializers.ModelSerializer):
    category = CategorySerializer()
    product_url = serializers.SerializerMethodField(read_only=True)

    class Meta:
        model = Product
        fields = ['id', 'product_name', 'slug', 'description', 'price', 'images', 'stock', 'is_available',
                  'created_date', 'modified_date', 'category', 'product_url']

    def get_product_url(self, obj):
        request = self.context.get('request')
        if request is None:
            return obj.get_url()  # No está en una solicitud HTTP
        return request.build_absolute_uri(obj.get_url())
```

Figura 46: *Serializador AccountSerializer - Microservicio Cuentas*

- **Store/Views.py**

El segmento de código ilustrado en la Figura No. 47 utiliza un slug de categoría para filtrar los productos por su categoría y si están disponibles. Si no se proporciona un slug de categoría, la vista presenta todos los productos en existencia. El acceso a esta vista está protegido con un token y los productos paginados junto con su cantidad total se envían a la plantilla «store/store.html» para la presentación.

```python
@token_required
def store(request, category_slug=None):
    # mostrar productos filtrados por el campo is_available
    categories = None
    products = None

    if category_slug is not None:
        # slug campo de Category
        categories = get_object_or_404(Category, slug=category_slug)
        # category campo de Product
        products = Product.objects.filter(category=categories, is_available=True)
        # paginacion -- productos a mostrar por pagina
        paginator = Paginator(products, 3)
        page = request.GET.get('page')  # ?page=1 en URL
        paged_products = paginator.get_page(page)
        product_count = products.count()
    else:
        products = Product.objects.all().filter(is_available=True).order_by('id')
        # paginacion -- productos a mostrar por pagina
        paginator = Paginator(products, 8)
        page = request.GET.get('page')  # ?page=1 en URL
        paged_products = paginator.get_page(page)
        product_count = products.count()

    context = {
        'products': paged_products,
        'product_count': product_count,
    }
    return render(request, 'store/store.html', context)
```

Figura 47: *Catálogo de Productos - Microservicio Tienda*

Luego de llevar a cabo la incorporación de las características del Microservicio de Tienda, se ejecutaron pruebas, cambiando los requisitos de la etapa de desarrollo a la fase de pruebas, mismas que se detallan en el capítulo 4. La Figura No. 48 muestra el estado final del Kanban correspondiente a este microservicio.

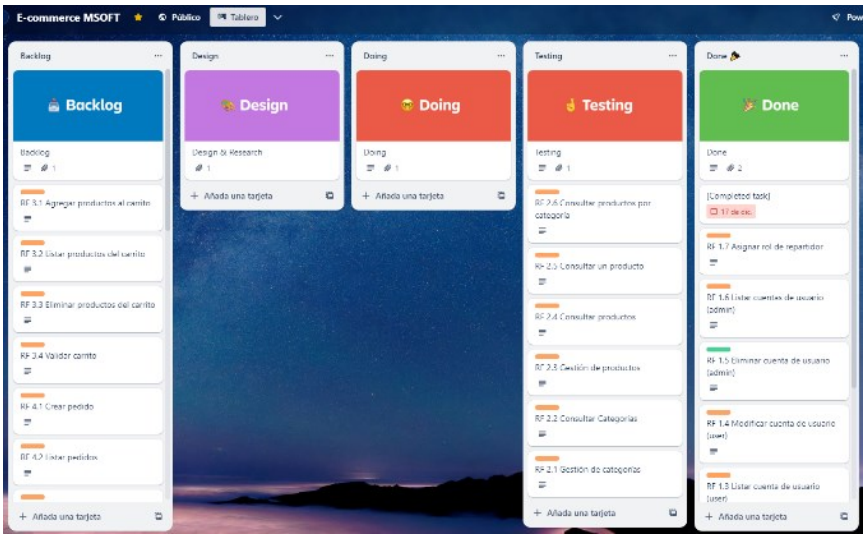


Figura 48: *Imagen del panel Kanban para el Microservicio de Tienda*

3.3.2.4 Backend Microservicio de Órdenes

Esta sección está organizada entorno a los componentes clave que se programaron para el desarrollo del Microservicio de Órdenes.

- **Dockerfile**

El Dockerfile mostrado configura un entorno Python 3.8 para la aplicación Django del Microservicio de Tienda, manejando dependencias y ajustando el servidor de desarrollo para ejecutarse en el puerto 8002, en contraste con el puerto 8001 usado por otro servicio de la tienda. Los detalles específicos de esta configuración se presentan en la Figura No. 49.

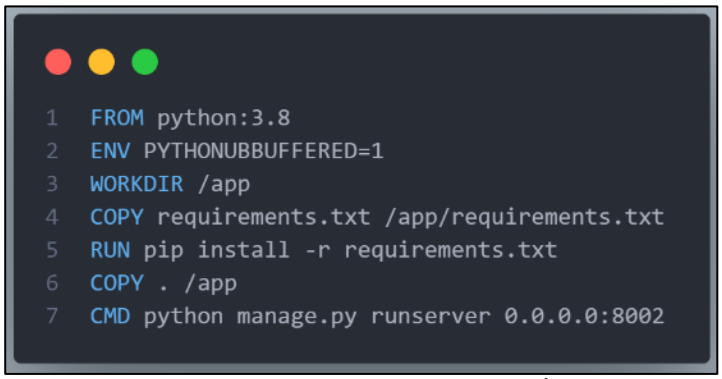


Figura 49: *Dockerfile del Microservicio de Órdenes*

- **Docker-compose.yml**

El fichero de configuración docker-compose.yml en la Figura No. 50 configura el entorno para el Microservicio de Órdenes en Django. Define un servicio «backend_order» que se ejecuta en el puerto 8002 y depende de una base de datos PostgreSQL «order_db», un servicio de cola Kafka «queue_order» y Redis. Se incluyen volúmenes para el código de este aplicativo y para los archivos multimedia, y todos los servicios están conectados a una red común de microservicios que se especifica como externa.

```yaml
version: '3.8'
services:
  # Django App
  backend_order:
    container_name: paul_ms_order
    build: .
    command: >
      sh -c "python manage.py runserver 0.0.0.0:8002"
    volumes:
      - .:/app
      - media_volume:/app/media
    ports:
      - "8002:8002"
    depends_on:
      - order_db
      - queue_order
      - redis
    networks:
      - microservices-network
  # Event Queue Kafka
  queue_order:
    build: .
    command: >
      sh -c "python consumer.py"
    depends_on:
      - order_db
    networks:
      - microservices-network

  # Database PostgreSQL
  order_db:
    image: postgres
    container_name: order_api_postresql
    restart: always
    environment:
      POSTGRES_DB: order_dbms
      POSTGRES_USER: paul
      POSTGRES_PASSWORD: 1234
    ports:
      - "5433:5432"
    volumes:
      - .dbdata:/var/lib/postgresql
    networks:
      - microservices-network

  redis:
    image: redis
    container_name: django_order_api_redis
    ports:
      - "6380:6379"
    networks:
      - microservices-network

volumes:
  media_volume:

networks:
  microservices-network:
    external: true
```

Figura 50: *Docker-compose para el Microservicio de Órdenes*

- **Consumer.py**

La Figura No. 51 contiene un fragmento de código del consumidor de Kafka en Django para el Microservicio de Órdenes. Este código ejecuta un bucle se mantiene en escucha de mensajes de un tópico de Kafka y los procesa. Si el mensaje recibido tiene como clave 'update_driver_status', actualiza el estado de conductor de un usuario en el Microservicio de Cuentas; si la clave es 'update_fullname', actualiza el nombre completo del usuario en los Microservicios de Cuentas y Store. El código maneja errores para ambos casos y asegura el cierre adecuado del consumidor al finalizar la ejecución.

```python
# Bucle de consumo de mensajes
with closing(consumer):
    while True:
        msg = consumer.poll(1.0)

        if msg is None:
            continue

        if msg.error():
            print("Consumer error: {}".format(msg.error()))
            continue

        # Procesar mensajes de Kafka
        topic = msg.topic()
        value = msg.value()
        data = json.loads(value)


        print(f'SMS recibido con Topic: {topic} y valor: {value}, con Data: {data}')
        sys.stdout.flush()

        # O realizar alguna acción especifica basada en el contenido del mensaje
        if topic == config('KAFKA_TOPIC'):
            if msg.key().decode('utf-8') == 'update_driver_status':
                try:
                    user_id = data['id']
                    is_driver = data['is_driver']
                    # Actualiza el estado is_driver del usuario en el microservicio de store
                    update_driver_status(user_id, is_driver)
                except Exception as e:
                    print(f"Error al actualizar el estado de conductor: {str(e)}")
                    sys.stdout.flush()
            if msg.key().decode('utf-8') == 'update_fullname':
                try:
                    user_id = data['id']
                    fullname = data['fullname']
                    # Actualiza el nombre completo del usuario en el microservicio de store
                    update_fullname(user_id, fullname)
                except Exception as e:
                    print(f"Error al actualizar el nombre completo: {str(e)}")
                    sys.stdout.flush()

# Cierra el consumidor al salir del bucle
consumer.close()
```

Figura 51: *Consumidor de mensajes Kafka dentro del Microservicio de Órdenes*

- **Middleware.py**

La Figura No. 52 muestra cómo se desarrolló el middleware personalizado en Django dentro del Microservicio de Órdenes, para gestionar las solicitudes CORS, incorporando cabeceras que permiten accesos de cualquier origen y aceptan todas las cabeceras entrantes, mejorando así la comunicación entre distintos microservicios.

```python
# core/middleware.py

class CustomCorsMiddleware:
    def __init__(self, get_response):
        self.get_response = get_response

    def __call__(self, request):
        response = self.get_response(request)
        response["Access-Control-Allow-Origin"] = "*"
        response["Access-Control-Allow-Headers"] = "*"
        return response
```

Figura 52: *Middleware del Microservicio de Órdenes*

- **Settings.py**

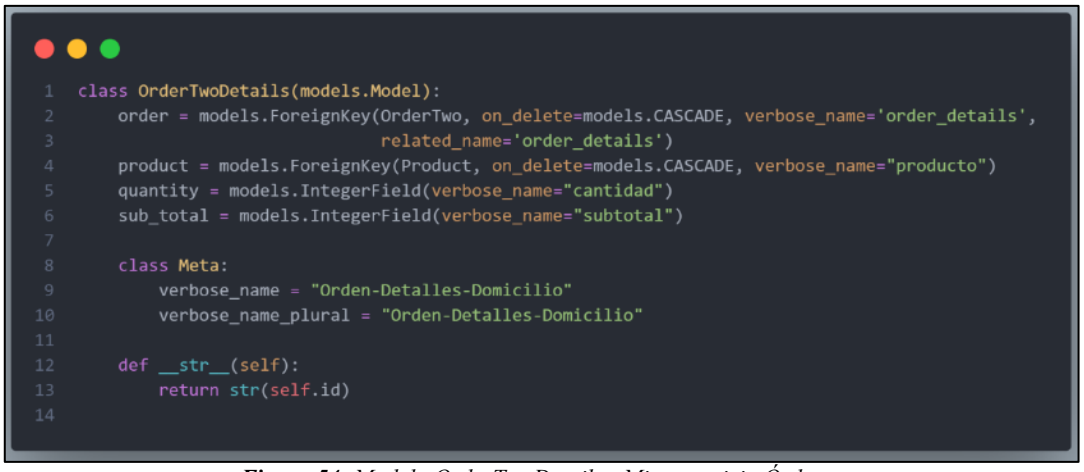
Como se muestra en la Figura No. 53, el archivo settings.py muestra la configuración del Microservicio de Órdenes en Django, donde se define la clave secreta del proyecto y se configura el middleware, este middleware es personalizado para manejar las solicitudes CORS, permitiendo así el acceso desde cualquier origen. También configura las sesiones para expirar después de una hora de inactividad y redirige a la página de inicio después del tiempo de inactividad. Además, se establecen las configuraciones del REST Framework de Django.

```python
SECRET_KEY = env('SECRET_KEY')

MIDDLEWARE = [
    'corsheaders.middleware.CorsMiddleware',
    'whitenoise.middleware.WhiteNoiseMiddleware',
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
    'core.middleware.CustomCorsMiddleware',
]
SESSION_EXPIRE_SECONDS = 3600  # 1 hour
SESSION_EXPIRE_AFTER_LAST_ACTIVITY = True
SESSION_TIMEOUT_REDIRECT = 'home'

# REST_FRAMEWORK
REST_FRAMEWORK = {
    "NON_FIELD_ERRORS_KEY": "errors",
    "DEFAULT_AUTHENTICATION_CLASSES": (
        "rest_framework.authentication.SessionAuthentication",
        "rest_framework.authentication.TokenAuthentication",
    ),
    "DEFAULT_PERMISSION_CLASSES": (
        "rest_framework.permissions.IsAuthenticated",
    ),
    "DEFAULT_RENDERER_CLASSES": [
        "rest_framework.renderers.JSONRenderer",
    ],
}
```

Figura 53: *Archivo de configuración del Microservicio de Órdenes*

- **Order/Models.py**

La Figura No. 54 exhibe el código del modelo en el Microservicio de Órdenes denominado OrderTwoDetails. Este modelo está vinculado a un modelo de OrderTwo, el cual se utilizó para la gestión de pedidos a domicilio desde el aplicativo móvil. Este modelo a través de claves foráneas sigue la política de eliminación en cascada. Además, incluye campos para la cantidad y el subtotal del pedido, elementos clave para el cálculo de costos dentro de las órdenes a domicilio. También se codifico un método __str__ que devuelve el identificador de la instancia del modelo como cadena de texto

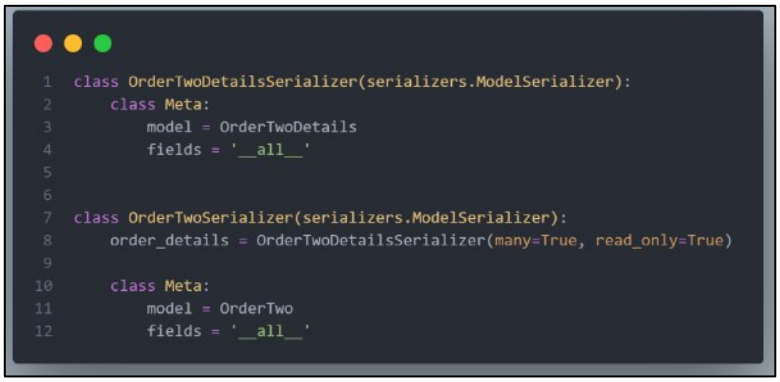


```python
class OrderTwoDetails(models.Model):
    order = models.ForeignKey(OrderTwo, on_delete=models.CASCADE, verbose_name='order_details',
                              related_name='order_details')
    product = models.ForeignKey(Product, on_delete=models.CASCADE, verbose_name="producto")
    quantity = models.IntegerField(verbose_name="cantidad")
    sub_total = models.IntegerField(verbose_name="subtotal")

    class Meta:
        verbose_name = "Orden-Detalles-Domicilio"
        verbose_name_plural = "Orden-Detalles-Domicilio"

    def __str__(self):
        return str(self.id)

```

Figura 54: *Modelo OrderTwoDetails - Microservicio Órdenes*

- **Order/Serializers.py**

Como se aprecia en la Figura No 55 se muestran dos serializadores. El primero es «OrderTwoDetailsSerializer», que serializa todos los campos del modelo 'OrderTwoDetails'. El segundo, «OrderTwoSerializer», incluye un campo anidado 'order_details' que utiliza «OrderTwoDetailsSerializer» para representar los detalles de la orden relacionada. Este campo está marcado para serializar múltiples instancias relacionadas.



```python
class OrderTwoDetailsSerializer(serializers.ModelSerializer):
    class Meta:
        model = OrderTwoDetails
        fields = '__all__'


class OrderTwoSerializer(serializers.ModelSerializer):
    order_details = OrderTwoDetailsSerializer(many=True, read_only=True)

    class Meta:
        model = OrderTwo
        fields = '__all__'
```

Figura 55: *Serializador AccountSerializer - Microservicio Cuentas*

- **Order/Views.py**

El segmento de código ilustrado en la Figura No. 56 muestra una vista de la API en Django Rest Framework llamada «ClientGetAllOrders». Esta vista permite a un determinado usuario obtener todas las órdenes asociadas a él. Primero, verifica el token de acceso para identificar al usuario. Luego, recupera todas las órdenes asociadas al usuario ordenadas por fecha de creación y las serializa en formato JSON.

```python
class ClientGetAllOrders(APIView):
    permission_classes = [permissions.AllowAny]

    def get(self, request):
        # Obtén el token de acceso a partir de la solicitud
        access_token_key = request.GET.get("access_token")
        try:
            token = UsuarioToken.objects.get(key=access_token_key)
            user = token.user_id
        except UsuarioToken.DoesNotExist:
            return Response({"status": "fallido", "error": "Token de acceso inválido"})

        # Obtén todas las órdenes asociadas al usuario
        orders = OrderTwo.objects.filter(client=user).order_by('-created_at')

        # Serializa las órdenes
        serialized_orders = OrderTwoSerializer(orders, many=True).data

        for order in serialized_orders:
            order_details = OrderTwoDetails.objects.filter(order=order['id'])
            order['order_details'] = OrderTwoDetailsSerializer(order_details, many=True).data

            # Agrega el nombre del producto a cada detalle de la orden
            for detail in order['order_details']:
                product = Product.objects.get(id=detail['product'])
                detail['product_name'] = product.product_name

        return JsonResponse({"orders": serialized_orders})
```

Figura 56: *Órdenes por Usuario - Microservicio Órdenes*

Una vez que se completaron las funcionalidades del Microservicio de Órdenes, se realizaron evaluaciones de funcionamiento, desplazando las tarjetas de la fase «En proceso» a «Pruebas»; los detalles de la funcionalidad de este microservicio se encuentran en capítulo cuatro. La Figura 57 representa el panel Kanban finalizado para este microservicio.

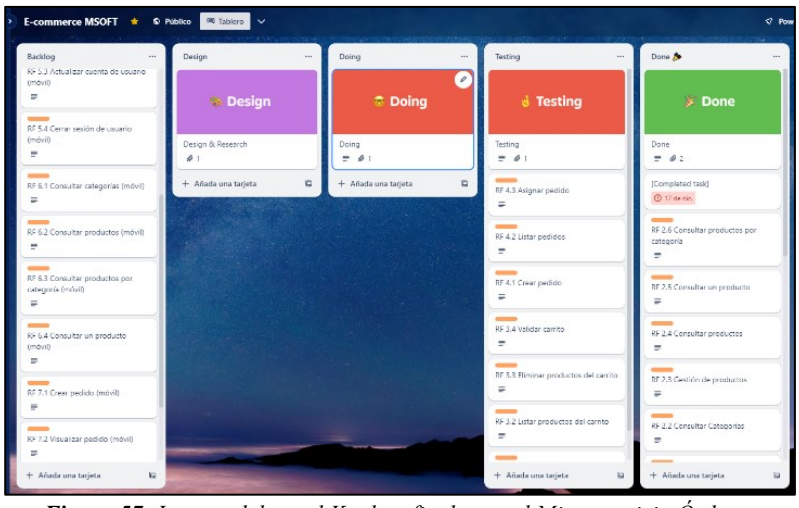


Figura 57: *Imagen del panel Kanban final para el Microservicio Órdenes*

3.3.2.5 API Gateway

Un API Gateway para microservicios es un componente de gestión de tráfico que actúa como punto de entrada unificado para las solicitudes externas dirigidas a los servicios de una arquitectura de microservicios. Funciona como intermediario que recibe las llamadas API, las enruta a los microservicios apropiados y luego agrega las respuestas para enviarlas de vuelta al solicitante. En esta sección se mostrará la configuración más relevante para el API Gateway.

- **Docker-compose.yml**

A continuación, se muestra la Figura No. 58, la cual configura un servicio de Nginx, que se construye desde el directorio actual y se establece para compartir un volumen llamado media_volume, mapeando una carpeta interna /nginx/media. El servicio Nginx se expone en el puerto 80.

Figura 58: *Docker-compose.yml para la creación del API Gateway*

- **Nginx.conf**

El archivo de configuración de Nginx actúa como un API Gateway y está configurado para enrutamiento de microservicios. Define bloques de 'upstream' para los Microservicios de Cuentas, Tienda y Órdenes, encaminando el tráfico al puerto 8000, 8001 y 8002, respectivamente. Escucha en el puerto 80 y configura rutas específicas: la raíz '/' para el servicio de cuentas y '/store' para el servicio de tienda. También maneja el contenido estático bajo la ruta '/media', que sirve archivos desde el volumen montado. Para cada ruta, establece encabezados necesarios para manejar conexiones proxy, incluyendo el manejo de direcciones IP reales y esquemas de redireccionamiento. Estas configuraciones se muestran en la Figura No. 59.

```
events {
    worker_connections  1024;
}

http {
    upstream account_service {
        server paul_ms_account:8000;
    }

    upstream store_service {
        server paul_ms_store:8001;
    }

    upstream store_service {
        server paul_ms_order:8002;
    }

    server {
        listen 80;
        server_name 0.0.0.0; # se reemplaza por el nombre de un dominio

        location / {
            proxy_pass http://account_service;
            proxy_http_version 1.1;
            proxy_set_header Upgrade $http_upgrade;
            proxy_set_header Connection "upgrade";
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header X-Forwarded-Proto $scheme;
            proxy_set_header Cookie $http_cookie;
        }
        location /store/ {
            proxy_pass http://store_service;
            proxy_http_version 1.1;
            proxy_set_header Upgrade $http_upgrade;
            proxy_set_header Connection "upgrade";
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header X-Forwarded-Proto $scheme;
            proxy_set_header Cookie $http_cookie;
        }
        location /order/ {
            proxy_pass http://order_service;
            proxy_http_version 1.1;
            proxy_set_header Upgrade $http_upgrade;
            proxy_set_header Connection "upgrade";
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header X-Forwarded-Proto $scheme;
            proxy_set_header Cookie $http_cookie;
        }
        location /media/ {
            alias /nginx/media/;  # Esta ruta apunta al volumen montado
            proxy_pass http://store_service;
            proxy_http_version 1.1;
            proxy_set_header Upgrade $http_upgrade;
            proxy_set_header Connection "upgrade";
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header X-Forwarded-Proto $scheme;
            proxy_set_header Cookie $http_cookie;
        }
        # Agregar otros `location` según sea necesario
    }
}

```

Figura 59: *Archivo de configuración para el API Gateway*

3.3.2.6 Endpoints de Comunicación

Un factor muy importante para la comunicación entre el Sistema Móvil y el Backend de Microservicios son los Endpoints de comunicación, que actúan como puntos de acceso definidos y expuestos por el backend, permitiendo al sistema móvil realizar solicitudes y recibir respuestas en un formato estructurado, usualmente a través de APIs RESTful. Estos endpoints facilitan la integración, el intercambio y la transferencia de datos en tiempo real, garantizando que el sistema móvil pueda interactuar de manera eficiente con los microservicios del backend, lo cual es esencial para mantener la funcionalidad y la experiencia del usuario. A continuación, se muestran 3 colecciones de Endpoints agrupadas por cada Microservicio, junto con el ejemplo de una petición GET a un Endpoint perteneciente al microservicio de tienda

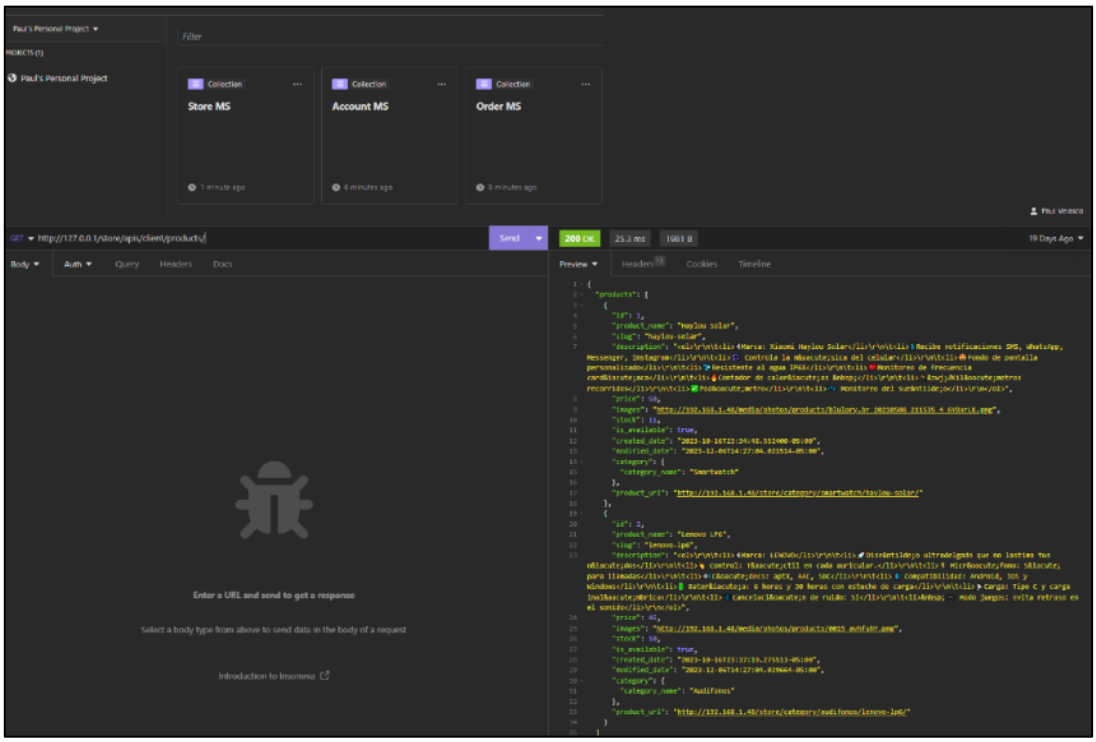


Figura 60: *Endpoints de comunicación*

3.3.2.7 Sistema Móvil

Esta sección describe el código fuente empleado en el aplicativo móvil, destacando las configuraciones clave y los fragmentos de código críticos para la operatividad del aplicativo. A continuación, se muestran los módulos más relevantes dentro del sistema móvil.

- Módulo Gestionar Perfil

Este módulo tiene una relevancia significativa, debido a que se encarga de la gestión de usuarios, como las funciones de inicio y cierre de sesión, edición de perfiles, registro de usuarios, entre otros. La Figura No. 61 ilustra el DTO (Data Transfer Object) de Account, una clase Java que agrupa datos de usuario, incluyendo identificadores, información personal, credenciales y el estado de la cuenta, así como el perfil de usuario vinculado, con anotaciones que aseguran su alineación con JSON para la serialización.

Account.java

```java
public class Account
{

    @SerializedName("id")
    private Integer id;
    @SerializedName("first_name")
    private String firstName;
    @SerializedName("last_name")
    private String lastName;
    @SerializedName("username")
    private String username;
    @SerializedName("email")
    private String email;
    @SerializedName("phone_number")
    private String phoneNumber;
    @SerializedName("date_joined")
    private String dateJoined;
    @SerializedName("last_login")
    private String lastLogin;
    @SerializedName("is_admin")
    private Boolean isAdmin;
    @SerializedName("is_staff")
    private Boolean isStaff;
    @SerializedName("is_active")
    private Boolean isActive;
    @SerializedName("is_superadmin")
    private Boolean isSuperadmin;
    @SerializedName("is_driver")
    private Boolean isDriver;
    @SerializedName("userprofile")

    private Userprofile userprofile;

    public Account()
    {

    }
```

Figura 61: *DTO Account - Sistema Móvil*

LoginActivity.java

La clase «LoginActivity.java» muestra un fragmento de código dentro de la Figura 62. Este código define un método 'doLogin' que realiza una solicitud de autenticación al Microservicio de Cuentas. Si la respuesta es exitosa, el método guarda el token de sesión y el estado de si el usuario es conductor en las preferencias compartidas, y luego redirige al usuario a la actividad principal correspondiente a su rol, ya sea cliente o conductor. Si la autenticación falla, muestra un mensaje de error indicando credenciales incorrectas o falta de conexión.

```
private void doLogin(String username, String password, String userType)
    {
        String contentType = "application/x-www-form-urlencoded";
        Call<Token> call = apiGetToken.getToken(contentType,username, password);
        call.enqueue(new Callback<Token>() {
            @Override
            public void onResponse(Call<Token> call, Response<Token> response)
            {
                if (response.isSuccessful())
                {
                    Token myResponse = response.body();
                    Log.d("LOGIN access_token", String.valueOf(myResponse.getToken()));
                    Log.d("is driver", String.valueOf(myResponse.isDriver()));
                    SharedPreferences.Editor editor = mPref.edit();
                    String tipodeUsuario = mPref.getString("usuario", "");

                    editor.putString("token", response.body().getToken());
                    editor.putString("tipousuario", tipodeUsuario);
                    editor.putBoolean("is_driver", response.body().isDriver());
                    editor.apply();

                    // Se inicia la actividad principal
                    if (userType.equals("cliente") && !myResponse.isDriver())
                    {
                        Intent intent = new Intent(getApplicationContext(), ClientMainActivity.class);
                        intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK | Intent.FLAG_ACTIVITY_CLEAR_TASK);
                        startActivity(intent);

                    } else if (userType.equals("conductor") && myResponse.isDriver())
                    {

                        Intent intent = new Intent(getApplicationContext(), DriverMainActivity.class);
                        intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK | Intent.FLAG_ACTIVITY_CLEAR_TASK);
                        startActivity(intent);
                    }
                    else
                    {
                        Toast temp = Toast.makeText(LoginActivity.this, "Verifique si su usuario es cliente o conductor", Toast.LENGTH_LONG);
                        temp.show();
                    }
                }
                else
                {
                    Log.e("TAG", "Error en la respuesta: " + response.code());
                    Toast temp = Toast.makeText(LoginActivity.this, "Usuario y/o contraseña incorrectos ", Toast.LENGTH_SHORT);
                    temp.show();
                }
            }

            @Override
            public void onFailure(Call<Token> call, Throwable t)
            {
                Log.e("TAG", "Error en la solicitud: " + t.getMessage());
                Toast temp = Toast.makeText(LoginActivity.this, "Sin conexión", Toast.LENGTH_LONG);
                temp.show();
            }
        });
    }
```

Figura 62: *Método DoLogin para inicio de sesión en el Sistema Móvil*

Después de desarrollar las características del Módulo de Gestión de Perfil en la aplicación móvil, se efectuaron pruebas, trasladando las tareas de «En proceso» a la columna de «Pruebas», cuyos pormenores se describen en el capítulo 4. La Figura No. 63 muestra el panel de Kanban finiquitado para ese módulo.

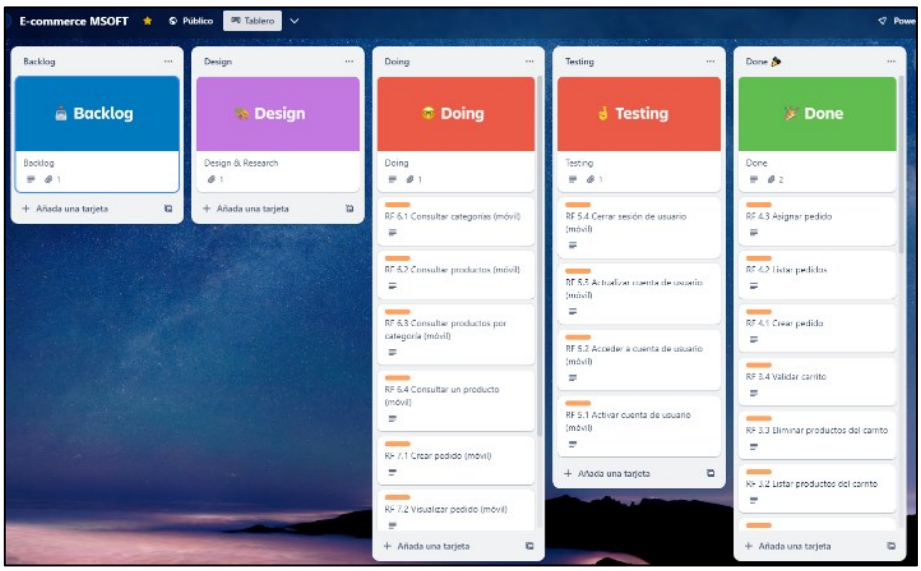


Figura 63: *Imagen del panel Kanban final para el Módulo Gestionar Perfil*

- Módulo Gestionar Catálogo

El siguiente módulo es de gran importancia, debido a que muestra el catálogo de productos de la tienda, incluyendo imágenes y precios. Cuando se elige un producto, se añade a un carrito de compras que se mantiene en una base de datos local dentro de la aplicación mediante Room, para su futura compra.

Dentro de la Figura 64 se puede apreciar un fragmento de código el cual muestra un adaptador personalizado en Android llamado «ProductAdapter», que vincula datos de productos con una interfaz de usuario, mostrando detalles como nombre, categoría, precio e imagen. Además, establece un método 'OnClickListener' para cada elemento, que, al dar un clic, pasa los detalles del producto a una nueva actividad para mostrar más información.

ProductAdapter.java

```java
public class ProductAdapter extends BaseAdapter {
    private Activity activity;
    private ArrayList<Product> productList;

    public ProductAdapter(Activity activity, ArrayList<Product> productList) {
        this.activity = activity;
        this.productList = productList;
    }

    @Override
    public int getCount() {
        return productList.size();
    }

    @Override
    public Object getItem(int position) {
        return productList.get(position);
    }

    @Override
    public long getItemId(int position) {
        return position;
    }

    @SuppressLint("SetTextI18n")
    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        if (convertView == null)
        {
            convertView = LayoutInflater.from(activity).inflate(R.layout.list_item_product,null);
        }
        final Product product = productList.get(position);

        TextView productName = convertView.findViewById(R.id.product_name);
        TextView productCategory = convertView.findViewById(R.id.product_category);
        TextView productPrice = convertView.findViewById(R.id.product_price);
        ImageView productImage = convertView.findViewById(R.id.product_image);

        productName.setText(product.getProductName());
        productCategory.setText(product.getCategory().getCategoryName());
        productPrice.setText("$ "+ product.getPrice());

        Picasso.get().load(product.getImages()).fit().into(productImage);
        convertView.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v)
            {
                Intent intent = new Intent(activity, ProductDetailActivity.class);
                intent.putExtra("productId", product.getId());
                intent.putExtra("productName", product.getProductName());
                intent.putExtra("productCategory", product.getCategory().getCategoryName());
                intent.putExtra("productDescription", product.getDescription());
                intent.putExtra("productImage", product.getImages());
                intent.putExtra("productPrice", product.getPrice());

                activity.startActivity(intent);

            }
        });

        return convertView;
    }
}
```

Figura 64: *ProductAdapter - Sistema Móvil*

Car.java

La clase «Car.java» en la Figura No. 65, muestra la utilización de la entidad 'Car' en la base de datos de Room también se visualizan campos anotados para mapear la clase con una tabla de base de datos: id como clave primaria autoincrementada (@PrimaryKey(autoGenerate = true)), y campos productId, productName, productPrice, y productQuantity, cada uno con anotaciones @ColumnInfo que definen los nombres de las columnas en la base de datos. La clase incluye métodos básicos para obtener y asignar el id.

```java
@Entity
public class Car
{
    @PrimaryKey(autoGenerate = true)
    private int id;
    @ColumnInfo(name="product_id")
    private int productId;
    @ColumnInfo(name="product_name")
    private String productName;
    @ColumnInfo(name="product_price")
    private Float productPrice;
    @ColumnInfo(name="product_quantity")
    private int productQuantity;

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }
```

Figura 65: *Entidad Car para la BDD de Room - Sistema Móvil*

CarDAO.java

La Figura 66 muestra una interfaz en Java para un DAO (Data Access Object) llamado 'CarDAO', anotada con @Dao para indicar su uso con Room, una librería de persistencia en Android. Incluye métodos para operaciones de base de datos: recuperar todos los registros 'getAll', insertar múltiples registros 'insertAll', eliminar todos los registros 'deleteAll', buscar un carro por productId 'getCarByProductId' y actualizar la cantidad de un producto en un carro 'updateCar'. Cada uno de estos métodos está anotado con @Query o @Insert para especificar la operación SQL correspondiente.

```java
@Dao
public interface CarDAO
{
    @Query("SELECT * FROM car")
    List<Car> getall();

    @Insert
    void insertAll(Car... cars);

    @Query("DELETE FROM car")
    void deleteAll();

    @Query("SELECT * FROM car WHERE product_id = :productId")
    Car getCarByProductId(int productId);

    @Query("UPDATE car SET product_quantity = product_quantity + :productQty WHERE product_id = :productId")
    void updateCar(int productQty, int productId);
}
```

Figura 66: *Acceso a datos CarDAO - Sistema Móvil*

Una vez completado el desarrollo de las funcionalidades del Módulo de Catálogo en el sistema móvil, se procedió con su fase de pruebas, moviendo las tareas desde «En proceso» hasta «Pruebas», con una explicación detallada de estas pruebas de funcionalidad en el capítulo 4. La Figura No. 67 expone el panel de Kanban terminado correspondiente a dicho módulo.

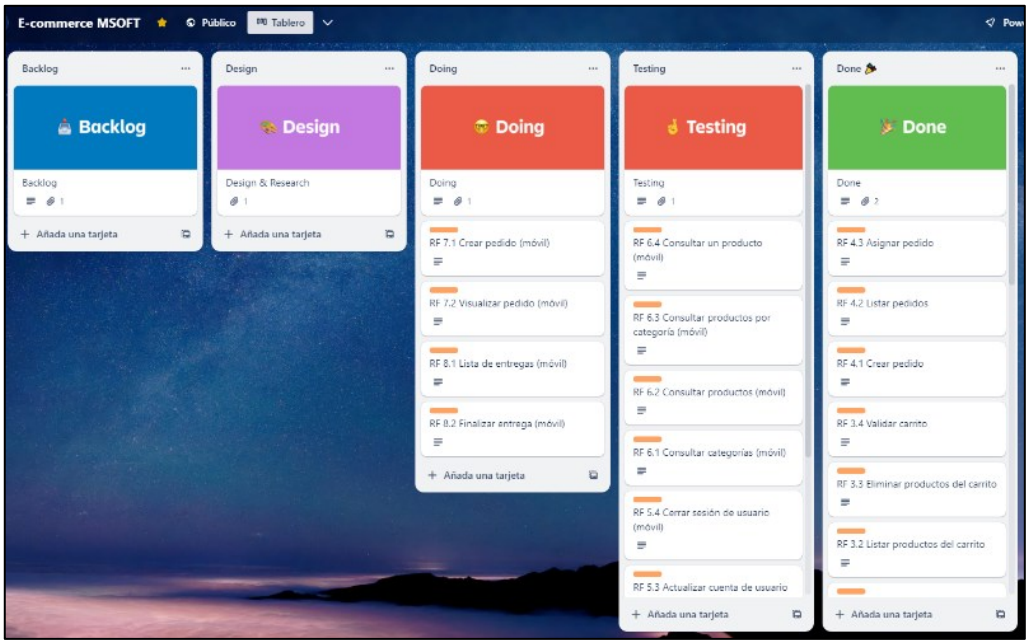


Figura 67: *Imagen del panel Kanban final para el Módulo Catálogo*

- MÓDULO GESTIONAR PEDIDOS

Este módulo facilita que el cliente seleccione productos del carrito, ingrese su dirección de entrega, realice el pago del pedido y espere la asignación de un repartidor por parte del Administrador de la tienda para la entrega a domicilio. Adicionalmente, ofrece la

posibilidad de monitorear la ubicación y ruta del repartidor en tiempo real a través del mapa de seguimiento de pedidos. La Figura 68 pertenece a la clase «CarActivity.java», dentro de ella se creó un método llamado 'handleMapAddress' el cual establece un 'OnEditorActionListener' en un campo de texto para manejar la acción 'done' del teclado. Cuando se activa, utiliza un Geocoder para convertir el texto del campo de dirección en coordenadas geográficas.

CarActivity.java

```
private void handleMapAdress()
    {
        address.setOnEditorActionListener(new TextView.OnEditorActionListener() {
            @Override
            public boolean onEditorAction(TextView text, int i, KeyEvent event)
            {
                if (i == EditorInfo.IME_ACTION_DONE)
                {
                    Geocoder coder = new Geocoder(CarActivity.this);
                    try
                    {
                        ArrayList<Address> addresses = (ArrayList<Address>) coder.getFromLocationName(text.getText().toString(),1);
                        if(!addresses.isEmpty())
                        {
                            double lat = addresses.get(0).getLatitude();
                            double lng = addresses.get(0).getLongitude();

                            LatLng pos = new LatLng(lat,lng);
                            mMap.moveCamera(CameraUpdateFactory.newLatLngZoom(pos,DEFAULT_ZOOM));
                            mMap.clear();
                            mMap.addMarker(new MarkerOptions().position(pos));
                        }
                    }
                    catch (IOException e)
                    {
                        e.printStackTrace();
                    }
                }


                return false;
            }
        });
    }
```

Figura 68: *Geocoder convertir dirección a coordenadas geográficas - Sistema Móvil*

PaymentActivity.java

La Figura 69 muestra un fragmento de código de la clase «PaymentActivity.java», el método addOrder toma un token de Stripe y utiliza Retrofit para enviar una solicitud POST a un servicio web con detalles de la orden como el token de acceso, dirección, latitud, longitud, detalles del producto y el token de Stripe. Si la respuesta es exitosa y la orden no ha fallado, redirige al usuario a la actividad principal y muestra un mensaje de éxito. Si la orden ha fallado, se eliminan los artículos del carrito y aparece una notificación de error. El método también maneja fallos en la llamada de red y muestra un mensaje de error correspondiente.

```java
private void addOrder(String stripeToken) {
    String accessToken = sharedPref.getString("token", "");
    AddOrder addOrder = new AddOrder(
            accessToken,
            address,
            latlng,
            orderDetails,
            stripeToken);
    // Se crea una solicitud HTTP POST utilizando Retrofit y se envía el cuerpo JSON
    Call<StatusResponse> call = apiAddOrder.addOrder(addOrder.getAccessToken(), addOrder.getAddress(),
            addOrder.getLocation(), addOrder.getOrderDetails(), addOrder.getStripeToken());
    call.enqueue(new Callback<StatusResponse>() {

        @Override
        public void onResponse(Call<StatusResponse> call, Response<StatusResponse> response) {
            Log.e("respuuesta", response.toString());
            if (response.isSuccessful()) {
                Log.e("success", "" + response.body().getStatus());

                if (!response.body().getStatus().equals("fallido")) {
                    deleteCar();
                    // Ir al otro Activity
                    Intent intent = new Intent(getApplicationContext(), ClientMainActivity.class);
                    intent.putExtra("screen", "order");
                    startActivity(intent);
                    Toast.makeText(getApplicationContext(), "Compra exitosa revise el menú Pedidos",
                            Toast.LENGTH_SHORT).show();
                } else {
                    showErrorAndReturnToCart(response.body().getError());
                }

            } else {
                showErrorAndReturnToCart(response.body().getError());
            }
        }

        @Override
        public void onFailure(Call<StatusResponse> call, Throwable t) {
            Log.e("respuuesta fail: ", t.getMessage());
        }
    });
}
```

Figura 69: *Pagar Orden - Sistema Móvil*

Después de incorporar las funciones en el Módulo de Pedidos de la aplicación móvil, se realizaron pruebas, trasladando las tareas de la sección «En Proceso» a «Pruebas». La Figura 70 exhibe el panel Kanban completo para este módulo.

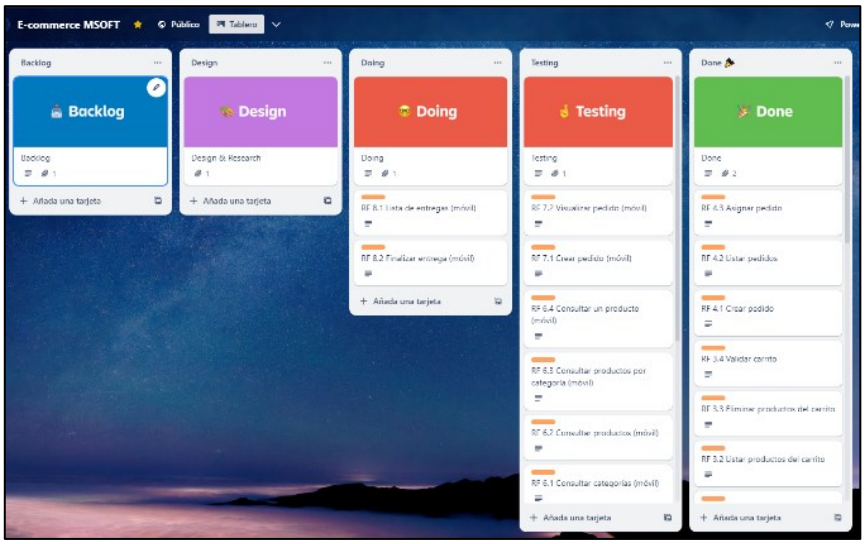


Figura 70: *Imagen del panel Kanban final para el Módulo de Pedidos*

Este módulo se habilita a los usuarios con el rol de 'Repartidor' para elegir la lista de pedidos asignados para entrega. El repartidor debe dirigirse al destino del cliente para realizar la entrega de los productos. Dentro de este módulo, el repartidor tiene la capacidad de ver tanto el número de productos como la dirección exacta del cliente. Una vez entregado el pedido el repartidor también puede visualizar los pedidos que ha entregado exitosamente y una gráfica de pedidos que ha entregado semanalmente.

OrderListDriverActivity.java

La Figura 71 muestra un método en de la clase «OrderListDriverActivity.java» llamado 'getDriverOrders', que recupera los pedidos de un repartidor. El método obtiene un token de acceso almacenado en las preferencias compartidas y lo utiliza para hacer una llamada a una API con Retrofit. Si la respuesta es exitosa, limpia la lista actual de pedidos y añade los nuevos pedidos recuperados, notificando al adaptador del cambio de datos para actualizar la vista. En caso de error, registra un mensaje de error.

```
private void getDriverOrders()
    {
        String accessToken = sharedPref.getString("token", "");
        String palabra = "Token "+accessToken;
        Call<ListOrdersDriver> call = apiListDriverOrders.getDriverOrdersByToken(palabra);
        call.enqueue(new Callback<ListOrdersDriver>() {
            @Override
            public void onResponse(Call<ListOrdersDriver> call, Response<ListOrdersDriver> response)
            {
                if (response.isSuccessful())
                {
                    VOrders = response.body().getOrders();
                    orderArrayList.clear();
                    orderArrayList.addAll(VOrders);
                    adapter.notifyDataSetChanged();
                }
            }

            @Override
            public void onFailure(Call<ListOrdersDriver> call, Throwable t)
            {
                Log.e("ERROR", t.getMessage());
            }
        });

    }
```

Figura 71: *Lista de pedidos a entregar - Sistema Móvil*

Después de incorporar las funciones en el Módulo de Entregas de la aplicación móvil, se realizaron pruebas, moviendo los elementos de trabajo de las secciones En Proceso (Doing) y Pruebas (Testing) al estado Completado (Done), y los detalles de estas pruebas se describen en el capítulo 4. La Figura 72 presenta la pizarra Kanban con todas las tareas finalizadas dentro de la plataforma de comercio electrónico.

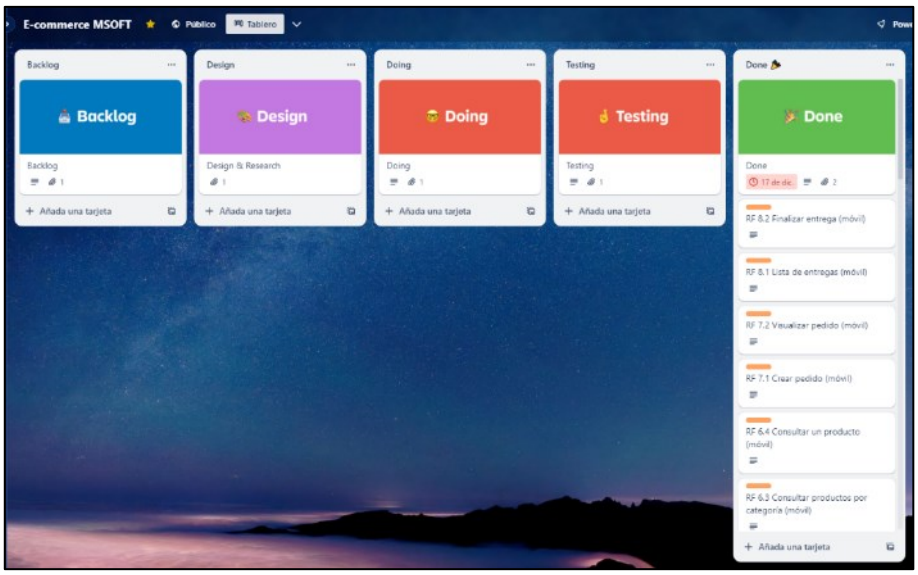


Figura 72: *Imagen del panel de Kanban final de todas las tareas*

4. Resultados y Discusión

Finalizada la fase de desarrollo se ha obtenido una plataforma de comercio electrónico construida bajo el patrón arquitectónico de microservicios, para la gestión entrega y rastreo de productos a domicilio. Dentro de este apartado se exponen los datos y conclusiones de las pruebas de funcionalidad realizadas. Se emplearon casos de prueba específicos para asegurar que la plataforma de comercio electrónico cumple con los resultados previstos. Estos casos de prueba están diseñados en función de los requisitos de software de la aplicación y para cada uno se definen los roles del sistema que llevan a cabo la prueba de funcionalidad, los datos que se deben introducir, una secuencia de pasos detallada para la prueba, así como los resultados esperados y los efectivamente logrados tras realizar la prueba.

4.1 Pruebas de funcionalidad Sistema Web

4.1.1 Microservicio Cuentas

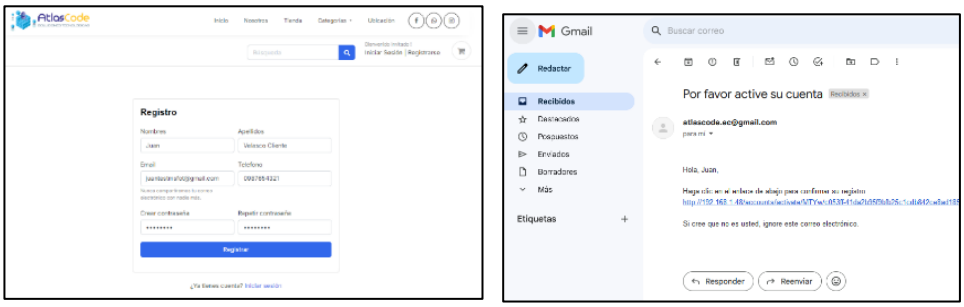
Prueba de funcionalidad 1	
Requerimiento	RF 1.1 Crear cuenta de usuario
Rol	Cliente
Datos de entrada	• Rol: Cliente • Nombres: Juan • Apellidos: Velasco Cliente • Email: juantestmsfot@gmail.com • Teléfono: 0987654321 • Contraseña: paul1234
Procedimiento	1. Iniciar el navegador y acceder al enlace http://127.0.0.1/ 2. Seleccionar el enlace de «registrarse» 3. Insertar los datos o información de entrada 4. Dar clic en el botón registrar
Logro previsto	Se crea la cuenta de usuario
Logro alcanzado	El logro previsto es alcanzado. En este caso, el usuario recibe un email para activar su cuenta.
Capturas de Pantalla	

Tabla 5: *Prueba de funcionalidad para crear cuenta de usuario*

Prueba de funcionalidad 2	
Requerimiento	RF 1.2 Acceder a una cuenta de usuario
Rol	Cliente, administrador o repartidor
Datos de entrada	• Rol: Cliente • Email: juantestmsfot@gmail.com ; Contraseña: paul1234
Procedimiento	1. Iniciar el navegador y acceder al enlace http://127.0.0.1/ 2. Presionar el enlace de iniciar sesión 3. Introducir datos de entrada 4. Dar clic en el botón ingresar
Logro previsto	El usuario entra a la plataforma utilizando el rol adecuado
Logro alcanzado	El logro previsto es alcanzado. En este caso, el usuario con el rol de cliente accede al sistema y muestra su respectivo dashboard.
Captura de Pantalla	

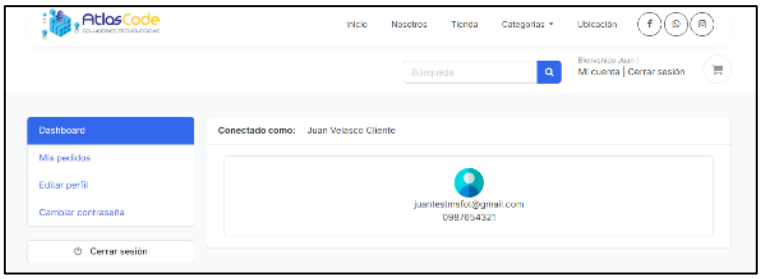


Tabla 6: *Prueba de funcionalidad para inicio de sesión*

Prueba de funcionalidad 3	
Requerimiento	RF 1.3 Modificar cuenta de un usuario
Rol	Cliente, administrador o repartidor
Datos de entrada	• Rol: Cliente • Nombre: Juan, Apellido: Cliente, Teléfono: 0987654321, Dirección: av. Occidental, 6 de diciembre, Quito, Pichincha, Ecuador
Procedimiento	1. Iniciar el navegador y acceder al enlace http://127.0.0.1/ 2. Iniciar sesión con email y contraseña 3. Dar clic en la pestaña de la izquierda 'editar perfil' 4. Ingresar datos de entrada y guardar
Logro previsto	La información del usuario se actualiza en el sistema, y los cambios realizados se reflejan en la pestaña 'editar perfil'
Logro alcanzado	El logro previsto es alcanzado. En este caso, el usuario con el rol de cliente modifica su información personal
Captura de Pantalla	

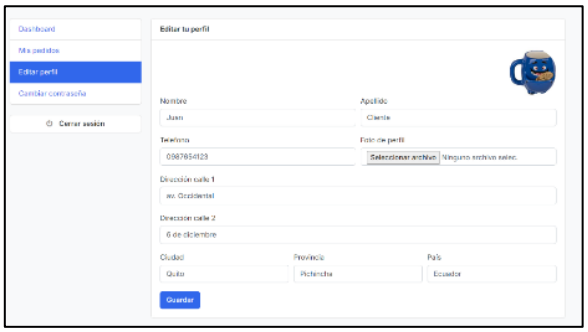


Tabla 7: *Prueba de funcionalidad para modificar datos de un usuario*

4.1.2 Microservicio Tienda

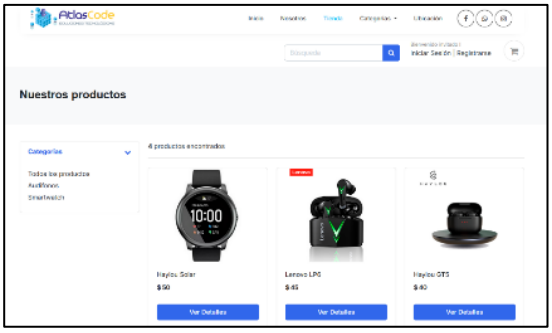
Prueba de funcionalidad 11	
Requerimiento	RF 2.4 Consultar productos
Rol	Cliente, administrador o repartidor
Datos de entrada	• Rol: Todos los roles • Opcionalmente se puede iniciar sesión o no
Procedimiento	1. Iniciar el navegador y acceder al enlace http://127.0.0.1/ 2. Cargar los datos de entrada 3. Dar clic en el menú 'Tienda'
Logro previsto	Los productos disponibles son mostrados a todos los usuarios
Logro alcanzado	El logro previsto es alcanzado de forma exitosa
Captura de Pantalla	

Tabla 8: *Prueba de funcionalidad consultar productos*

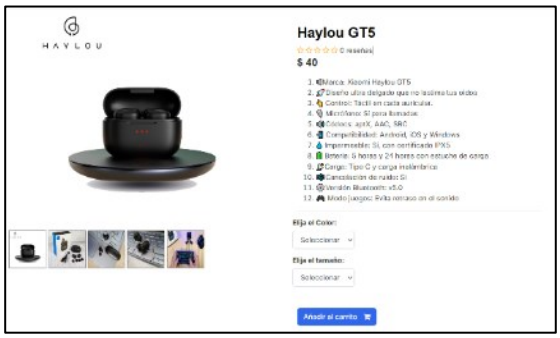
Prueba de funcionalidad 12	
Requerimiento	RF 2.5 Consultar un producto
Rol	Cliente, administrador o repartidor
Datos de entrada	• Rol: Todos los roles • Opcionalmente se puede iniciar sesión o no
Procedimiento	1. Iniciar el navegador y acceder al enlace http://127.0.0.1/ 2. Cargar los datos de entrada 3. Dar clic en el menú 'Tienda' 4. Seleccionar producto y dar clic en el botón 'Ver Detalles'
Logro previsto	Los productos disponibles son mostrados a todos los usuarios de forma detallada
Logro alcanzado	El logro previsto es alcanzado. En este caso se observa datos complementarios del producto seleccionado.
Captura de Pantalla	

Tabla 9*:Prueba de funcionalidad consultar un producto a detalle*

4.1.3 Microservicio Órdenes

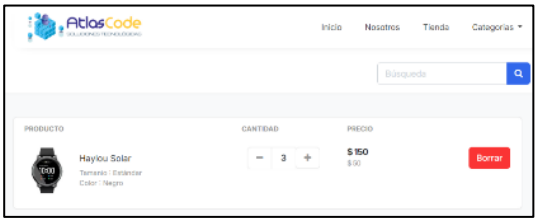
Prueba de funcionalidad 14	
Requerimiento	RF 3.1 Agregar productos al carrito
Rol	Cliente
Datos de entrada	• Rol: Cliente • Ingresar e-mail y contraseña para inicio de sesión
Procedimiento	1. Iniciar el navegador y acceder al enlace http://127.0.0.1/ 2. Cargar los datos de entrada 3. Dar clic en el menú 'Tienda' 4. Seleccionar producto y dar clic en el botón 'Ver Detalles' 5. Seleccionar color y tamaño 6. Dar clic en añadir al carrito
Logro previsto	El producto seleccionado se añade al carrito de compras
Logro alcanzado	El logro previsto es alcanzado. En este caso se observa una imagen pequeña del producto seleccionado y también se puede modificar la cantidad de productos seleccionados con los botones de '-' y '+'
Captura de Pantalla	

Tabla 10: *Prueba de funcionalidad agregar productos al carrito*

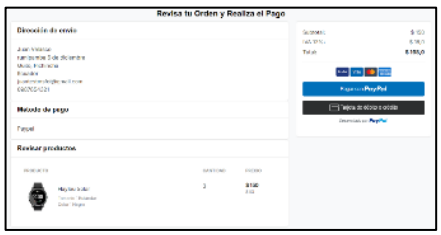
Prueba de funcionalidad 18	
Requerimiento	RF 4.1 Generar pedido a partir de los productos dentro del carrito
Rol	Cliente
Datos de entrada	• Rol: Cliente • Datos de entrada (nombre, apellido, email, dirección, etc)
Procedimiento	1. Dar clic en el icono de cesta de compras 2. Dar clic en checkout 3. Cargar datos de entrada 4. Dar clic en Realizar pedido 5. Pagar
Logro previsto	El producto seleccionado pasa del carrito de compras a pedidos para su posterior pago.
Logro alcanzado	El logro previsto es alcanzado. En la primera imagen se muestran los datos del cliente y el producto seleccionado para su posterior pago. La segunda imagen muestra el detalle del pago del pedido
Capturas de Pantalla	 

Tabla 11: *Prueba de funcionalidad generar pedido*

4.2 Pruebas de Funcionalidad Sistema Móvil

4.2.1 Módulo Gestionar Perfil

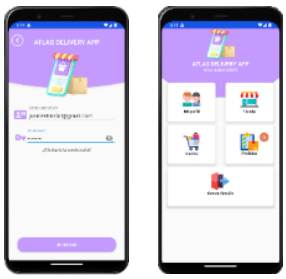
Prueba de funcionalidad 22	
Requerimiento	RF 5.2 Acceder a una cuenta de usuario
Rol	Cliente, Repartidor
Datos de entrada	• Rol: Cliente o Repartidor • Email: juantestmsfot@gmail.com • Contraseña: paul1234
Procedimiento	1. Abrir el aplicativo móvil 2. Dar clic en soy Cliente o soy Repartidor 3. Cargar la información o datos de entrada 4. Dar clic en 'iniciar sesión'
Logro previsto	El usuario ingresa al sistema utilizando el rol asignado
Logro alcanzado	El logro previsto es alcanzado.
Capturas de Pantalla	

Tabla 12: *Prueba de funcionalidad para inicio de sesión*

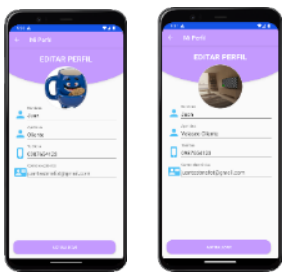
Prueba de funcionalidad 23	
Requerimiento	RF 5.3 Modificar datos del usuario
Rol	Cliente, Repartidor
Datos de entrada	• Rol: Cliente o Repartidor • Datos de entrada (nombres, apellidos, teléfono, imagen)
Procedimiento	1. Abrir el aplicativo móvil 2. Iniciar sesión previamente 3. Dar clic en la tarjeta de 'Mi Perfil' 4. Ingresar datos de entrada 5. Dar clic en el botón Actualizar
Logro previsto	La información del usuario es actualizada en todo el sistema
Logro alcanzado	El logro previsto es alcanzado.
Capturas de Pantalla	

Tabla 13: *Prueba de funcionalidad para modificar datos del usuario*

4.2.2 Módulo Gestionar Catálogo

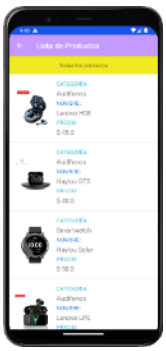
Prueba de funcionalidad 26	
Requerimiento	RF 6.2 Consultar Productos
Rol	Cliente
Datos de entrada	• Rol: Cliente • Datos de entrada inicio de sesión (email y contraseña)
Procedimiento	1. Abrir el aplicativo móvil 2. Iniciar sesión previamente 3. Dar clic en la tarjeta de 'Tienda'
Logro previsto	Los productos disponibles en la tienda se muestran al cliente
Logro alcanzado	El logro previsto es alcanzado, junto con el precio y nombre de los productos
Capturas de Pantalla	

Tabla 14: *Prueba de funcionalidad consultar productos*

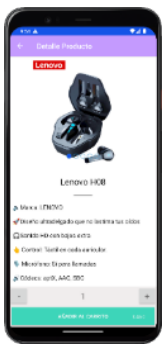
Prueba de funcionalidad 28	
Requerimiento	RF 6.4 Consultar un producto
Rol	Cliente
Datos de entrada	• Rol: Cliente • Datos de entrada inicio de sesión (email y contraseña)
Procedimiento	1. Abrir el aplicativo móvil 2. Iniciar sesión previamente 3. Clic en la tarjeta de 'Tienda' 4. Seleccionar un producto en especifico
Logro previsto	El detalle del producto se muestra al cliente
Logro alcanzado	El logro previsto es alcanzado, junto con las características del producto y un carrusel de imágenes del producto
Capturas de Pantalla	

Tabla 15: *Prueba de funcionalidad consultar un producto*

4.2.3 Módulo Gestionar Pedidos

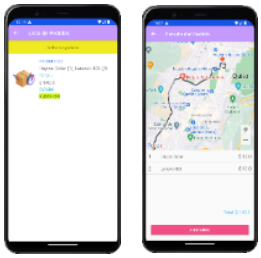
Prueba de funcionalidad 30	
Requerimiento	RF 7.2 Visualizar Pedido
Rol	Cliente
Datos de entrada	• Rol: Cliente
Procedimiento	1. Abrir el aplicativo móvil 2. Clic en la tarjeta de 'Pedidos' 3. Clic en el pedido que desea revisar el estado 4. Visualizar el tracking del repartidor en la pantalla
Logro previsto	El cliente puede visualizar el estado de su pedido (alistando – en camino – entregado)
Logro alcanzado	El logro previsto es alcanzado, junto con el tracking en tiempo real del repartidor que va a entregar su pedido.
Capturas de Pantalla	

Tabla 16: *Prueba de funcionalidad visualizar pedido*

4.2.4 Módulo Gestionar Entregas

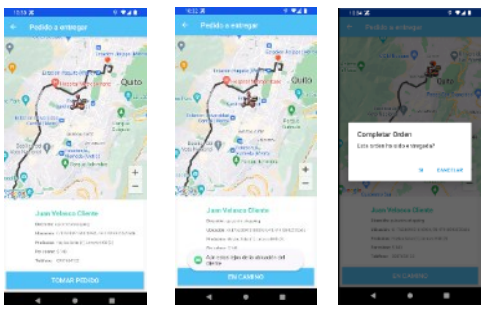
Prueba de funcionalidad 32	
Requerimiento	RF 8.2 Finalizar Entrega
Rol	Repartidor
Datos de entrada	• Rol: Repartidor
Procedimiento	1. Abrir el aplicativo móvil 2. Clic en la tarjeta de 'Órdenes', luego clic en el pedido que el repartidor debe entregar 3. Clic en tomar Pedido, llegada a la dirección del cliente clic en 'En camino' para completar la orden
Logro previsto	El repartidor puede visualizar la lista de pedidos que debe entregar y dirigirse a la ubicación del cliente, para finalizar el pedido
Logro alcanzado	El logro previsto es alcanzado, también el repartidor debe estar a 10 metros a la redonda de la ubicación del cliente para finalizar el pedido
Capturas de Pantalla	

Tabla 17: *Prueba de funcionalidad finalizar entrega*

5. CONCLUSIONES

1. La creación de una plataforma de comercio electrónico basada en microservicios ha demostrado ser una estrategia eficaz para lograr independencia de componentes y escalabilidad individual de los mismos, donde cada microservicio funciona de manera independiente lo que permite que el sistema crezca y se actualice fácilmente de acuerdo con las necesidades del negocio.

2. La utilización de Kafka como un sistema de mensajería para la comunicación entre distintos microservicios ilustra cómo se pueden manejar de forma eficiente flujos de datos y eventos complejos en tiempo real. Sin embargo, también destaca el desafío inherente de gestionar un sistema distribuido donde los servicios deben estar perfectamente coordinados y sincronizados.

3. La implementación de un API Gateway se ha revelado como una estrategia efectiva para administrar las peticiones entre el sistema móvil y los microservicios del sistema web. Al ofrecer un punto único de acceso, facilita considerablemente la interacción entre los usuarios y los diferentes servicios del sistema.

4. La aplicación de la metodología Kanban resultó eficaz en el manejo del proceso de desarrollo de toda la plataforma. Se clasificaron los requerimientos funcionales en distintas etapas representadas por las columnas Backlog, Design, Doing, Testing y Done en el tablero Kanban. Esta estructura proporcionó una visión integral del progreso del proyecto. Una vez completados los sistemas web y móvil, se realizaron las pruebas necesarias y se logró mover todos los requerimientos funcionales a la columna Done, indicando su finalización exitosa.

5. Se llevaron a cabo diversas pruebas para verificar la funcionalidad de la plataforma, las cuales proporcionaron resultados que satisfacen las expectativas y cumplen con las necesidades establecidas en la toma de requisitos.

6. GLOSARIO

Apache Kafka: Plataforma para transmitir datos en tiempo real con sistema de mensajes distribuidos y modelo de publicación/suscripción. Usado en análisis en streaming y procesamiento de eventos en tiempo real.

API Gateway: Se trata de un componente que gestiona las solicitudes de API y las dirige a los servicios correspondientes.

Arquitectura de Software: Es la estructura y diseño fundamental de un sistema de software.

Backend: Considerada la parte de una aplicación que maneja la lógica y el procesamiento detrás de escena.

Docker: Es una plataforma de contenedores que facilita la creación, implementación y gestión de aplicaciones.

Comercio electrónico: Actividad orientada a la adquisición y venta de productos con el uso del internet.

Framework: Un conjunto de herramientas y reglas que facilitan el desarrollo de software.

Frontend: Es la parte visible de una aplicación que los usuarios interactúan directamente.

Kanban: Se trata de una metodología de gestión visual de proyectos que utiliza tarjetas para rastrear el flujo de trabajo.

Microservicios: Una arquitectura de software en la que una aplicación se divide en pequeños servicios independientes que se comunican entre sí.

Monolito: Es un tipo de arquitectura de software en la que todas las funciones y componentes están integrados en una sola aplicación.

REST: Un estilo arquitectónico de software para diseñar servicios web que utiliza métodos HTTP para acceder y manipular recursos.

Sistema Móvil: Aplicaciones diseñadas para dispositivos móviles como smartphones y tablets.

Sistema Web: Un software que opera a través de la World Wide Web y se accede mediante un navegador.

Referencias

Arnau, R. (s. f.). *Por qué DEBES usar JAVA para programar tus proyectos—//Arteco*. Recuperado 9 de enero de 2023, de https://www.arteco-consulting.com/post/porque-debes-usar-java

AWS. (s. f.). *¿Qué es Java? - Guía de Java empresarial para principiantes - AWS*. Amazon Web Services, Inc. Recuperado 9 de enero de 2023, de https://aws.amazon.com/es/what-is/java/

Blancarte, O. (2020). *Introducción a la arquitectura de software* (primera). https://www.oscarblancarteblog.com

Castellano, L. (2019). Kanban. Metodología para aumentar la eficiencia de los procesos. *3c Tecnología: glosas de innovación aplicadas a la pyme*, *8*(1), 30-41.

Clarke, M. J. & others. (2018). Live Visualisation of Experiment Data at ISIS and the ESS. *Proc. of International Conference on Accelerator and Large Experimental Control Systems (ICALEPCS'17), Barcelona, Spain, 8-13 October 2017*, *16*, 431-434. https://doi.org/10.18429/JACoW-ICALEPCS2017-TUPHA029

Cleveland, S. B., Jamthe, A., Padhy, S., Stubbs, J., Packard, M., Looney, J., Terry, S., Cardone, R., Dahan, M., & Jacobs, G. A. (2020). Tapis API Development with Python: Best Practices In Scientific REST API Implementation: Experience implementing a distributed Stream API. *Practice and Experience in Advanced Research Computing*, 181-187. https://doi.org/10.1145/3311790.3396647

Curso Django: Entendiendo como trabaja Django. (2012, abril 30). *Maestros del Web*. http://www.maestrosdelweb.com/curso-django-entendiendo-como-trabaja-django/

Fielding, R. T. (s. f.). *Estilos Arquitectónicos y Diseño de Arquitecturas de Software Basadas en Red*. Recuperado 30 de diciembre de 2022, de

https://www.proquest.com/openview/fc2d064044b971dda476dfb429a2b344/1?pq-origsite=gscholar&cbl=18750&diss=y

Fowler, M. (2015, julio 1). *Microservice Trade-Offs*. martinfowler.com. https://martinfowler.com/articles/microservice-trade-offs.html

Gamella, N. (2022). *Qué es un e-commerce: Tipos de negocios y pasos para crearlo*. 1-5.

Garlan, D., & Shaw, M. (s. f.). *An Introduction to Software Architecture*. Recuperado 30 de diciembre de 2022, de https://userweb.cs.txstate.edu/~rp31/papers/intro_softarch.pdf

González, J. L. (2016, diciembre 21). *EL E-COMMERCE: SUS ORÍGENES, PRESENTE Y TENDENCIAS FUTURAS*. https://www.esic.edu/saladeprensa/prensa/noticia/el-e-commerce-sus-origenes-presente-y-tendencias-futuras

Guimarey, A. (2020). *Beneficios y riesgos de migrar una arquitectura monolítica a microservicios*. https://www.researchgate.net/publication/348309479

Lachgar, M., Benouda, H., & Elfirdoussi, S. (2018). Android REST APIs: Volley vs Retrofit. *2018 International Symposium on Advanced Electrical and Communication Technologies (ISAECT)*, 1-6. https://doi.org/10.1109/ISAECT.2018.8618824

Lewis, J., & Fowler, M. (2014). Microservices. *Martinfowler.Com*. https://martinfowler.com/articles/microservices.html

Maida, E., & Pacienzia, J. (2015). *Metodologías de desarrollo de software* [Universidad Católica de Argentina]. https://repositorio.uca.edu.ar/bitstream/123456789/522/1/metodologias-desarrollo-software.pdf

Newman, S. (2015). *Building Microservices* (O'Reilly Medi). https://www.thoughtworks.com/insights/books/building-microservices

Pautasso, C., Wilde, E., & Alarcon, R. (2013). *REST: advanced research topics and practical applications*. Springer.

Radigan, D. (2021, agosto 21). *Kanban How the kanban methodology applies to software development*. Atlassian. https://www.atlassian.com/agile/kanban

Rodríguez, P., Musat Salvador, D., Yagüe Panadero, A., Turhan, B., Rohunnen, A., Kuvaja, P., & Oivo, M. (2010). Adopción de Metodologías Ágiles: Un estudio comparativo entre España y Europa. *Revista Española en Innovación, Calidad e Ingeniería del Software(REICIS)*, *6*(4), Article 4.

Vergara, S. (2019, julio 4). ¿Kafka para gestionar los datos en tiempo real de mis servicios? *Blog ITDO - Agencia de desarrollo Web, APPs y Marketing en Barcelona*. https://www.itdo.com/blog/kafka-para-gestionar-los-datos-en-tiempo-real-de-mis-servicios/

Vivanco, J. (2019, junio 5). *El modelo C4 de documentación para la Arquitectura de Software | by Javier Vivanco | Medium*. https://medium.com/@javiervivanco/el-modelo-c4-de-documentaci%C3%B3n-para-la-arquitectura-de-software-424704528390

Zambrano, B., Castellanos, E., & Miranda, M. (2021). El E-Commerce en las empresas ecuatorianas: Un análisis de los informes de la Cámara Ecuatoriana de Comercio Electrónico (CECE) en el marco de la pandemia covid-19. *Revista Publicando*, *8*(29), 13-20. https://doi.org/10.51528/rp.vol8.id2176

Zhao, J. T., Jing, S. Y., & Jiang, L. Z. (2018). Management of API Gateway Based on Micro-service Architecture. *Journal of Physics: Conference Series*, *1087*(3), 032032. https://doi.org/10.1088/1742-6596/1087/3/032032