UTMACH

FACULTAD DE INGENIERÍA CIVIL

CARRERA DE INGENIERÍA DE SISTEMAS

DESARROLLO DE LA APLICACIÓN MÓVIL EN LA PLATAFORMA DE REALIDAD AUMENTADA

RAMON RAMON RICARDO ALEXANDER
INGENIERO DE SISTEMAS

MACHALA
2021

UTMACH

FACULTAD DE INGENIERÍA CIVIL

CARRERA DE INGENIERÍA DE SISTEMAS

Desarrollo de la aplicación móvil en la plataforma de realidad aumentada

RAMON RAMON RICARDO ALEXANDER
INGENIERO DE SISTEMAS

MACHALA
2021

UTMACH

FACULTAD DE INGENIERÍA CIVIL

CARRERA DE INGENIERÍA DE SISTEMAS

TRABAJO TITULACIÓN
PROPUESTAS TECNOLÓGICAS

Desarrollo de la aplicación móvil en la plataforma de realidad aumentada

RAMON RAMON RICARDO ALEXANDER
INGENIERO DE SISTEMAS

RIVAS ASANZA WILMER BRAULIO

MACHALA, 24 DE SEPTIEMBRE DE 2021

MACHALA
2021

ricardo ramon

INFORME DE ORIGINALIDAD

5%	4%	1%	2%
INDICE DE SIMILITUD	FUENTES DE INTERNET	PUBLICACIONES	TRABAJOS DEL ESTUDIANTE

FUENTES PRIMARIAS

1	Submitted to Universidad de San Buenaventura Trabajo del estudiante	<1%
2	Submitted to Universidad de Cundinamarca Trabajo del estudiante	<1%
3	Submitted to Universidad Carlos III de Madrid Trabajo del estudiante	<1%
4	documents.mx Fuente de Internet	<1%
5	dspace.espoch.edu.ec Fuente de Internet	<1%
6	Submitted to Pontificia Universidad Catolica del Ecuador - PUCE Trabajo del estudiante	<1%
7	"Proceedings of International Conference on Sustainable Expert Systems", Springer Science and Business Media LLC, 2021 Publicación	<1%

CLÁUSULA DE CESIÓN DE DERECHO DE PUBLICACIÓN EN EL REPOSITORIO DIGITAL INSTITUCIONAL

El que suscribe, RAMON RAMON RICARDO ALEXANDER, en calidad de autor del siguiente trabajo escrito titulado Desarrollo de la aplicación móvil en la plataforma de realidad aumentada, otorga a la Universidad Técnica de Machala, de forma gratuita y no exclusiva, los derechos de reproducción, distribución y comunicación pública de la obra, que constituye un trabajo de autoría propia, sobre la cual tiene potestad para otorgar los derechos contenidos en esta licencia.

El autor declara que el contenido que se publicará es de carácter académico y se enmarca en las dispociones definidas por la Universidad Técnica de Machala.

Se autoriza a transformar la obra, únicamente cuando sea necesario, y a realizar las adaptaciones pertinentes para permitir su preservación, distribución y publicación en el Repositorio Digital Institucional de la Universidad Técnica de Machala.

El autor como garante de la autoría de la obra y en relación a la misma, declara que la universidad se encuentra libre de todo tipo de responsabilidad sobre el contenido de la obra y que asume la responsabilidad frente a cualquier reclamo o demanda por parte de terceros de manera exclusiva.

Aceptando esta licencia, se cede a la Universidad Técnica de Machala el derecho exclusivo de archivar, reproducir, convertir, comunicar y/o distribuir la obra mundialmente en formato electrónico y digital a través de su Repositorio Digital Institucional, siempre y cuando no se lo haga para obtener beneficio económico.

Machala, 24 de septiembre de 2021

RAMON RAMON RICARDO ALEXANDER
0706321551

Dedicatoria

Este proyecto va dedicado a mi familia, la cual me apoyó toda la vida en mis estudios y me motivó a seguir avanzando académicamente con su apoyo, motivación y consejo; especialmente a mi madre, que me supo entender en los momentos más difíciles de mi carrera.

Ricardo Alexander Ramón Ramón

Agradecimiento

Agradezco a Dios por permitirme llegar a la finalización de mi periodo universitario con salud y con la compañía de mis padres y amigos, los cuales, sin su apoyo incondicional no habría llegado hasta donde estoy.

Agradezco a mis compañeros de la 19ava promoción y por la amistad brindada y su apoyo y ayuda durante estos 5 años.

Por último, a mi tutor, Ing. Sist. Rivas Asanza Wilmer Braulio, Phd, por sus enseñanzas durante el proceso académico y su guía, ayuda y compromiso con el desarrollo de este proyecto.

Resumen

La venta de artículos de varias marcas a través de aplicaciones y medios digitales ha generado un sistema de comercio móvil explotable y aplicable a cualquier sector comercial, principalmente hoy en día que se puede aprovechar esta tecnología y aplicarla en un medio de amplio uso como lo son las aplicaciones móviles.

Dado que el uso de las aplicaciones móviles está muy presente en la vida cotidiana de la población general hace que un enfoque de desarrollo hacia las aplicaciones móviles llegue a tener más impacto y reconocimiento que una aplicación orientada a la web o como aplicación de escritorio. El desarrollo de un prototipo orientado a un entorno móvil propone el diseño de una interfaz más amigable y con una fácil y rápida operabilidad.

Entre las tendencias actuales aplicadas al desarrollo móvil se encuentra el uso de modelos 3D para realidad aumentada. Si bien esta tecnología está más aplicada a aplicaciones interactivas o videojuegos, se consideró para el desarrollo de esta aplicación viendo el potencial que tiene como herramienta de marketing.

Teniendo en cuenta esta oportunidad, se desarrolló el prototipo de una aplicación móvil orientada al comercio electrónico que pueda hacer uso de la tecnología de realidad aumentada cómo una innovación tecnológica, misma que serviría para proyectar cómo se vería el objeto a la venta en el entorno de un cliente, ayudándolo a su toma de decisiones.

La aplicación se desarrolló bajo el framework multiplataforma React Native, utilizando como lenguaje de programación a JavaScript; se utilizó la plataforma Expo como herramienta de testeo y lanzamiento de la aplicación de manera nativa, tanto para Android como iOS. Se aplicó la metodología XP para el desarrollo del prototipo dado su enfoque ágil y que está más centrado en el desarrollo que en los involucrados, y la plataforma Stripe como pasarela para la ejecución de pagos.

Para el acceso al contenido de la aplicación, el usuario deberá crear una cuenta con un nombre de usuario, correo y contraseña para de loguearse, dentro podrá agregar más información como su nombre, además de agregar información

sobre sus dirección, necesarias para poder ejecutar compras; los productos a la venta indicarán información como el nombre, el precio, el descuento en caso de tener, imágenes y botones funcionales que permitirán agregar el producto a una lista de favoritos, al carrito de compras y hacer una proyección de realidad aumentada en caso de que dicho producto esté asociado a un modelo 3D.

Al momento de ejecutar una compra se registra en la base de datos únicamente información para el historial de pedidos como la dirección, el cliente, el producto y el precio; la información de la tarjeta no se registra ni se guardará de ninguna manera por motivos de seguridad de la información del usuario.

La evaluación aplicada al prototipo desarrollado se diseñó basándose en el estándar de calidad ISO/IEC 25010, dado a que este se centra en la calidad del producto de software, así como pruebas unitarias que verifican el cumplimiento de las historias de usuarios planteadas.

Palabras clave: Aplicación móvil, comercio móvil, Metodología XP, desarrollo multiplataforma.

Abstract

The sale of articles from various brands through applications and digital media has generated an exploitable mobile commerce system applicable to any commercial sector, mainly today that this technology can be used and applied in a widely used environment such as mobile applications.

Since the use of mobile applications is very present in the daily life of the general population, a development approach towards mobile applications has more impact and recognition than an application oriented to the web or as a desktop application. The development of a prototype oriented to a mobile environment proposes the design of a more friendly interface and with an easy and fast operability.

Among the current trends applied to mobile development is the use of 3D models for augmented reality. Although this technology is more applied to interactive applications or video games, it is considered for the development of this application seeing the potential that it has as a marketing tool.

Taking this opportunity into account, the prototype of a mobile application oriented to electronic commerce was developed that can make use of augmented reality technology as a technological innovation, which would serve to project how the object for sale would look in the environment of a client, helping him with his decision making.

The application was developed under the React Native multiplatform framework, using JavaScript as the programming language; The Expo platform was used as a tool for testing and launching the application natively, both for Android and iOS. The XP methodology was applied for the development of the prototype given its agile approach and that it is more focused on the development than on those involved, and the Stripe platform as a gateway for the execution of payments.

To access the content of the application, the user must create an account with a username, email and password to log in, inside you can add more information such as your name, in addition to adding information about your address, necessary to make purchases; The products for sale indicate information such as the name, the price, the discount if they have it, images and functional buttons that will allow adding that product to a list of favorites, to the shopping cart and to

make an augmented reality projection in case of that said product is associated with a 3D model.

At the time of making a purchase, only information for order history such as address, customer, product and price is recorded in the database; Card information is not recorded or saved in any way for user information security reasons.

The evaluation applied to the developed prototype was designed based on the ISO / IEC 25010 quality standard, since it focuses on the quality of the software product, as well as unit tests that verify compliance with the user stories raised.

Keywords: Mobile application, mobile commerce, XP Methodology, multiplatform development.

Introducción

El manejo de dispositivos móviles y el uso de aplicaciones se han convertido en un hábito arraigado a la actual cultura tecnológica de la población general, dado su accesibilidad, facilidad de uso y la resolución de necesidades específicas. La pandemia de Covid-19 generó un aislamiento que debilitó ampliamente la economía de empresas que subsisten de la venta tradicional.

El uso de tecnología 3D para realidad virtual y aumentada es una tendencia que aumenta cada vez más y abarca una amplia gama de usos y dispositivos, desde consolas hasta teléfonos inteligentes, permitiendo que pueda estar al alcance del usuario la necesidad de adquirir un equipo específico o altamente costoso para su ejecución.

Considerando lo anterior, este trabajo presenta el desarrollo de un prototipo de aplicación móvil apta para el comercio móvil, convirtiéndose en una alternativa para la venta física de artículos, orientado a la venta electrodomésticos y mueblería; diseñada para la implementación de tecnología de realidad aumentada que permita la proyección de los artículos a la venta como modelos 3D en el mundo real.

El prototipo cuenta con una interfaz interactiva e intuitiva, que hace uso de marcos de trabajo para el desarrollo nativo de aplicación que facilitan su producción; además de acceder a diferentes funciones a través del consumo API’s de un back-end robusto y estructurado.

El informe divide sus contenidos en diferentes capítulos, siendo así el primero que detalla el ámbito de la aplicación, los requerimientos a los que se adhiere su desarrollo y la justificación del porqué se debe satisfacer dichos requerimientos; el segundo capítulo consta de una definición del prototipo, fundamentación teórica, definición de la metodología a implementarse, objetivos de desarrollo y algunos elementos de la metodología, en este caso su fase de planificación, diseño y ejecución; y el tercer capítulo que consta del plan de evaluación y el análisis de sus resultados.

índice de contenido

Índice de Tablas

Índice de Figuras

1. DIAGNÓSTICO DE NECESIDADES Y REQUERIMIENTOS

1.1 Ámbito de Aplicación: descripción del contexto y hechos de interés

Dados los antecedentes de la pandemia actual y el aumento de los hábitos de compra online, sugieren cambiar el modelo de negocio actual y animar a las empresas y almacenes a involucrarse en el comercio electrónico.

Dado los avances tecnológicos de la última década y el globalizado uso de internet, la gran mayoría de la población económicamente activa tiene acceso al uso diario de un teléfono inteligente.

La realidad aumentada es una innovación tecnológica que apoya el desarrollo tecnológico y otros campos de formas cada vez más diversas e interesantes. Estando presente en aplicaciones para juegos, filtros de redes sociales, arte o en el ámbito educativo.

El acoplamiento de la tecnología de realidad aumentada proporciona un punto de vista más claro sobre cómo percibir un objeto que no tenemos en un ambiente común como usuarios, dando la sensación de posesión del objeto o de interacción con él.

La percepción humana de cómo luce un objeto en un ambiente donde no se encuentra únicamente a través de una imagen mental da paso a varios errores de imprecisión, los cuales afectarían a una toma de decisiones acertada, principalmente en un ambiente comercial donde las decisiones tomadas afectan económicamente a los usuarios o clientes finales.

El alcance de este proyecto es desarrollar un prototipo funcional bajo el framework de React Native que permita la compra de artículos de manera online bajo una interfaz de usuario agradable e intuitiva con la capacidad de mostrarle al usuario mediante realidad aumentada cómo se vería el objeto en su entorno antes de decidir comprarlo.

1.2 Establecimiento de requerimientos

Con el fin de desarrollar una aplicación que sea de uso simple para los usuarios y no consuma mucho de su tiempo se definieron los requerimientos generales para el funcionamiento del prototipo:

- La aplicación permitirá el acceso y registro de cualquier usuario de manera rápida con un correo/usuario y contraseña.
- La interfaz permitirá navegar con facilidad entre las diferentes pantallas de la aplicación.
- La aplicación permitirá el ingreso de varias direcciones para el envío de los productos comprados.
- El sistema de búsqueda almacenará un historial local de las búsquedas realizadas para un acceso más rápido a productos relacionados.
- La interfaz de los productos proporcionará el nombre, precio, galería de imágenes relacionados, la posibilidad de guardar dichos productos en su lista de favoritos, agregar el producto al carrito y una opción para la visualización del producto como un modelo 3D y mediante realidad aumentada en el entorno.
- El apartado de cuenta permitirá ingresar la información del usuario como nombres y apellidos, además de modificar la información de acceso como son el correo/usuario y contraseña.
- Se podrá realizar los pagos mediante el ingreso de datos de tarjetas de crédito/débito.
- La plataforma no registrará la información de la tarjeta que ingrese el usuario.
- La aplicación registrará un historial de los pedidos realizados.
- El prototipo será preparado para su despliegue en dispositivos de sistema operativo iOS.

1.3 Justificación de requerimiento a satisfacer

Este prototipo se desarrolla bajo la línea de investigación de Innovación Tecnológica Y Creación De Nuevos Productos.

Desde el surgimiento del comercio electrónico (e-commerce) y el comercio móvil (m-commerce) es mucho más frecuente cada año que se realicen compras vía internet a través de páginas web o dispositivos móviles; factor que se ha incrementado debido a las medidas de distanciamiento propuestas para evitar la propagación de Covid-19 durante la pandemia.

A través de imágenes promocionales, los clientes pueden entender el concepto del producto que quieren comprar, o intentar imaginar cómo les quedará una vez que tengan el artículo a comprar; la integración de la realidad aumentada y la plataforma de comercio electrónico eliminará esta duda, proporcionando un panorama más claro de cómo interactuaría dicho artículo en su entorno.

El manejo de una interfaz sencilla que solicita únicamente la información necesaria del usuario le ayuda a familiarizarse mejor con su funcionamiento y evitar que se abrumen con funciones, rutas o interfaces complejas que le hagan perder el interés, así como la decisión de no guardar información importante como los datos de tarjetas de crédito/débito para mayor seguridad.

El prototipo, a pesar de utilizar un framework multiplataforma, se pensó para su desarrollo y despliegue principalmente en dispositivos iOS, dada la afinidad existente entre el sistema operativo y React Native.

2. DESARROLLO DEL PROTOTIPO

2.1 Definición del prototipo tecnológico

2.1.1. Definición del sistema

Las aplicaciones móviles presentan un nuevo frente en el comercio electrónico a través dado el uso extendido de estos dispositivos en la población general. Una aplicación móvil que funcione como una tienda presenta una oportunidad de innovación tecnológica que puede presentar una oportunidad comercial debido al contexto de la pandemia actual que dificulta realizar compras de manera presencial.

La aplicación consiste en una pieza de software desarrollada bajo un framework multiplataforma (React-Native) para ampliar la posibilidad de publicarse en dispositivos Android y iOS sin mayor dificultad. Su desarrollo se centró principalmente en presentar una interfaz fácil de manejar y entender por parte del usuario para presentar productos relacionados a la línea blanca y mueblería.

La selección de este tipo de productos se dio debido a que el atractivo principal de la aplicación es el uso de realidad aumentada para darle al cliente la sensación de pertenencia del objeto antes de decidir comprarlo, y estos artículos son más compatibles a este concepto por su tamaño y relación armónica con un ambiente de hogar.

2.1.2. Definición del alcance

Se propone el desarrollo de una aplicación orientada al comercio móvil para la venta de productos de línea blanca y/o mueblería aplicando técnicas de realidad aumentada. La aplicación se desarrolla para el sistema operativo iOS bajo el framework React Native aplicando la metodología de desarrollo ágil XP.

2.1.3. Identificación de riesgos

Tabla 1: Identificación de riesgos

No.	Riesgo	Efecto	Impacto	Prob.	Evaluación de riesgo	Estrategia de Mitigación
1	Comunicación ineficiente en	Una mala comunicación	2	2	Medio	Diálogo constante y

	el equipo de trabajo en el diseño de bocetos	provocaría un desacuerdo en cómo se vería el diseño final.				exposición de ideas entre el equipo de diseño y desarrollo.
2	Comunicación ineficiente en el equipo de trabajo en el desarrollo del prototipo	El prototipo desarrollado no satisface las expectativas del equipo.	3	2	Alto	Diálogo constante y exposición de ideas entre el equipo de diseño y desarrollo.
3	Problemas de compatibilidad entre servicios de Backend y Frontend	La aplicación ejecutaría algunas o todas sus funciones de forma ineficiente	3	1	Medio	Realizar pruebas para determinar la compatibilidad entre servicios.
4	Problemas de compatibilidad entre hardware y software	Problemas de compatibilidad entre la interfaz e-commerce y la plataforma móvil.	2	1	Bajo	Desarrollo de pruebas constantes durante el desarrollo.
5	Fallo en la ejecución de pagos	Problemas de compatibilidad entre la pasarela de pago y la plataforma móvil.	3	1	Medio	Optar por una herramienta para la gestión de pagos alternativa.

Fuente: Elaboración propia

2.1.4. Requerimientos funcionales y no funcionales

Tabla 2: Definición de requerimientos funcionales

Especificación de requerimientos funcionales	
RF-01	Permite que el usuario se registre con un correo, nombre de usuario y contraseña
RF-02	Permite que el cliente acceda al contenido de la aplicación con su correo/usuario y contraseña
RF-03	Permite que el cliente modifique su información en el apartado de cuenta
RF-04	Permite que el cliente agregue, edite y elimine sus direcciones de envío.

RF-05	El cliente podrá realizar búsqueda de los productos mediante palabras clave
RF-06	La aplicación guardará un historial de las búsquedas realizadas
RF-07	El cliente podrá agregar productos a su lista de favoritos y carrito de compras
RF-08	El cliente podrá realizar sus pagos desde la aplicación cuando termine su compra
RF-09	La aplicación registrará un historial de compras que podrá ver el usuario.

Fuente: Elaboración propia

Tabla 3: Definición de requerimientos no funcionales

Especificación de requerimientos no funcionales	
RNF-01	La aplicación se desarrollará principalmente para dispositivos iOS.
RNF-02	Las interfaces deben ser fáciles de navegar y manejar.
RNF-03	Las peticiones al servidor se realizarán de forma segura para proteger la información sensible.
RNF-04	La aplicación deberá ser responsiva para adaptarse a cualquier pantalla.
RNF-05	Los colores, iconos y elementos gráficos deberán ser intuitivos para una navegación ágil.
RNF-06	La aplicación deberá responder de forma rápida a los tiempos de carga para ejercer operaciones u obtener información solicitada.

Fuente: Elaboración propia

2.2 Fundamentación teórica del prototipo

2.2.1. Entorno de Trabajo

2.2.1.1. Herramientas de software

2.2.1.1.1. Visual Studio Code

Según lo indica [1] Visual Studio Code es un editor de código fuente gratuito que facilita el desarrollo y ejecución de aplicaciones de manera local y está disponible para sistemas operativos Windows, macOS y Linux. Este editor tiene un soporte

para el manejo de JavaScript, TypeScript y Node.js; además de un amplio sistema de extensiones para otros lenguajes y tiempos de ejecución.

2.2.1.1.2. Android Studio

Tal como lo indica [2] Android Studio es el entorno de desarrollo integrado o IDE oficial utilizado para el desarrollo de aplicaciones móviles para el sistema operativo Android.

Android Studio ofrece funciones para un aumento productivo en el desarrollo de Apps para Android, algunas de dichas funciones son:

- Un emulador rápido y cargado de funciones.
- Un entorno unificado para el desarrollo de aplicaciones para todos los dispositivos Android.
- Aplicación de cambios en el código y recursos de la aplicación mientras está en tiempo de ejecución sin necesidad de detenerse.
- Fácil integración con GitHub para el respaldo del código de la aplicación.
- Amplia variedad de frameworks y herramientas de desarrollo de prueba.
- Herramientas para la identificación de problemas en el rendimiento, la usabilidad y compatibilidad de versiones.
- Compatibilidad con C++ y NDK.

2.2.1.1.3. Xcode

Xcode es un paquete de software utilizado por los programadores para escribir software para Mac OS X, dispositivos iOS, Apple Watch y ahora Apple TV.

Xcode es un paquete llamado IDE (Integrated Development Environment), que contiene editores, compiladores y otras herramientas de software que trabajan juntas para ayudarlo a escribir software, compilarlo, cargarlo en el dispositivo, depurarlo y finalmente enviarlo a la App Store. [3]

2.2.1.1.4. JavaScript

JavaScript es un lenguaje de programación ligero, interpretado o compilado que trabaja con funciones de primera clase. Es un lenguaje dinámico de un solo

subproceso, de múltiples paradigmas y creación de prototipos con soporte para programación orientada a objetos, imperativa y declarativa. [4]

Creado por Brendan Eich en 1995, es uno de los lenguajes de programación más utilizados en el desarrollo web debido a su método simple de incrustar HTML. También es uno de los lenguajes de programación más potentes, extensos y flexibles que existen. [5]

Inicialmente comenzó como un medio para la progresión de sitios web de lado del cliente, actualmente ha evolucionado para la programación del lado del cliente como del servidor, así como el desarrollo de aplicaciones nativas para dispositivos móviles. Dado que JavaScript es un lenguaje interpretado, los desarrolladores no están equipados con un compilador que pueda ayudarlos a detectar código erróneo y no optimizado. [6]

2.2.1.1.5. MongoDB

Tal como lo indica [7] MongoDB es un almacén de datos NoSQL de código abierto basado en documentos que es comercialmente compatible con 10gen. Aunque MongoDB no es relacional, implementa muchas características de las bases de datos relacionales, como clasificación, índices secundarios, consultas de clasificación y consultas de documentos anidados.

Las bases de datos NoSQL surgieron para hacer frente a las limitaciones de los métodos relacionales tradicionales La mayoría de los aspectos de estas tecnologías NoSQL son muy diferentes y tienen poco en común, excepto que no utilizan un modelo de datos relacionales. [8]

NoSQL proporciona un modelo de datos más adecuado para estas necesidades, ya que, a diferencia de los modelos tradicionales, no requiere ningún tipo de esquema de tabla fijo. [9]

El enfoque de MongoDB está orientado a cuatro aspectos: flexibilidad, funcionalidad, velocidad y facilidad de uso. Admite el índice y el servidor de replicación, y proporciona controladores para múltiples lenguajes de programación.

2.2.1.1.6. Strapi

Strapi es un sistema de administración de contenido de código abierto que permite una fácil administración de contenido a largo plazo a través del panel de administración. Su sistema de plug-in extensible incluye panel de administración, autenticación y administración de derechos, administración de contenido, generador de API, etc. [10]

Permite a los editores administrar y distribuir fácilmente su contenido, crear API's de forma rápida y sencilla y proporcionar una interfaz fácil de usar que permite a los usuarios crear tipos de contenido, parecido a las tablas de una base de datos.

2.2.1.1.7. Stripe

Stripe Connect es un producto que proporciona 4 tipos de procesamiento de pagos compatible, flujo de caja, incorporación de compradores y vendedores, precios, paneles y otros servicios para una gran cantidad de empresas, incluidos muchos mercados en línea. [11]

2.2.1.2. Comparativa de Frameworks móviles multiplataforma

Aunque las soluciones nativas admiten el acceso a todas las funciones y sensores de dispositivos nativos, este no es el caso de los marcos multiplataforma. El soporte proporcionado por cada marco para una función de dispositivo específica depende del marco en sí y, a veces, de la plataforma elegida para la implementación. [12]

Actualmente Flutter, Xamarin y React son los marcos de trabajo para desarrollo multiplataforma más utilizados.

React trabaja con el lenguaje JavaScript, de forma que utiliza un solo código para desplegar tanto a Android como iOS, Flutter tiene una compatibilidad total con Google, a pesar de que no es tan definido en el desarrollo de aplicaciones nativas y Xamarin es un marco que trabajo con C# y funciona para desarrollar aplicaciones para plataformas como Android, iOS y Windows.

Según lo define [13] Xamarin es una plataforma tecnológica .NET, que permite el desarrollo Aplicaciones IOS y Android que utilizan código C#, lo que permite

compartir código entre plataformas hasta en un 75% usando Xamarin clásico o hasta en un 100% usando Xamarin Forms.

Según un estudio de [14], en términos de uso y popularidad del framework multiplataforma, los desarrolladores, están más interesados en explorar React Native que marcos como PhoneGap o Ionic Framework.

La elección de un framework para el desarrollo de aplicaciones móviles depende de cuál sea más adecuado al fin de la aplicación a desarrollarse, ya que cada una posee un fuerte sobre la velocidad de la aplicación, productividad del desarrollador, arquitectura y facilidad de aprendizaje.

2.2.1.2.1. React y React Native

Según [15], React, a veces denominado React.js, es un marco de JavaScript desarrollado por Facebook y lanzado como código abierto en 2013 para ayudar a la comunidad de desarrollo a construir interfaces.

React Native es un framework presentado en la conferencia de React.js de Facebook en 2015 el cual, en un principio, funcionaba únicamente para iOS, aunque actualmente ya tiene soporte para aplicaciones Android. [16]

El propósito del desarrollo de aplicaciones móviles a través de React Native es la programación única de la aplicación que pueda ser exportada con facilidad tanto para sistemas Android como iOS. [17]

Para [18], React Native tiene como objetivo combinar las ventajas de los elementos nativos de la interfaz de usuario con la familiaridad de JavaScript por parte de los desarrolladores web.

2.2.1.3. Librerías

2.2.1.3.1. Expo

Expo es el marco y la plataforma para aplicaciones generales desarrolladas bajo el framework de React. Es un conjunto de herramientas y servicios construidos alrededor de React Native y plataformas nativas que pueden ayudarlo a desarrollar, construir, implementar e iterar entre iOS, Android y aplicaciones web utilizando JavaScript de manera rápida. [19]

2.2.1.3.2. Lodash

Como lo indica [20] lodash es una biblioteca de utilidades de JavaScript moderna que ofrece modularidad, rendimiento y extras; utilizada para el manejo óptimo de arreglos. Esta librería facilita el trabajo de JavaScript en lo que respecta al manejo de matrices, números, objetos, cadenas, operaciones básicas, etc.

2.2.1.3.3. Formik

Formik está diseñado para administrar formularios fácilmente con validación compleja, admite la validación a nivel de formulario y de campo sincrónica y asincrónica; además, también admite la validación de nivel de formulario basada en patrones a través del soporte integrado de Yup. [21]

2.2.1.3.4. Yup

Yup es un generador de esquemas de JavaScript para el análisis y la verificación de valores con el cual se puede definir patrones, convertir valores en forma de coincidencia, verifique los valores existentes o ambos. Los esquemas Yup son muy expresivos, lo que le permite modelar verificaciones o transformaciones de valores complejas e interdependientes. La API de Yup está fuertemente inspirada en Joi, pero es más ágil y está construida con la verificación del lado del cliente como el caso de uso principal. [22]

2.2.2. Aplicaciones móviles

Las aplicaciones móviles o apps son piezas de software destinadas al despliegue y funcionamiento en sistemas operativos iOS o Android.

Para [23] el desarrollo de aplicaciones móviles es un caso especial de desarrollo de software porque los desarrolladores deben considerar diferentes aspectos, como el ciclo de vida de desarrollo corto, las características del dispositivo móvil, la movilidad, las especificaciones del dispositivo móvil, el diseño y la navegación de la interfaz de usuario de la aplicación y la seguridad.

Las aplicaciones móviles según [24] se pueden clasificar en 3 tipos: nativas, híbridas y web. A diferencia de las aplicaciones web, las aplicaciones nativas e híbridas se instalan físicamente en el dispositivo, por lo que siempre están

disponibles para los usuarios. Las aplicaciones nativas se desarrollan para un sistema operativo en específico, una aplicación web es un sitio web que parece una aplicación nativa, mientras que una aplicación híbrida combina los aspectos anteriores y funciona en un entorno web y un sistema operativo móvil. [25]

El desarrollo de aplicaciones móviles está creciendo rápidamente y es de gran importancia económica y científica. Una de las principales razones del fracaso del desarrollo de aplicaciones móviles es el aumento del número de plataformas móviles. [26]

El desarrollo de aplicaciones móviles conlleva muchos desafíos adicionales, algunos relacionados con los requisitos comerciales que cambian rápidamente, algunos relacionados con la fragmentación del hardware y la plataforma, la experiencia del usuario, la experiencia en el desarrollo de software, seguridad, planificación y presupuestos ajustados.

2.2.3. Desarrollo ágil de software

Con el fin de mejorar la productividad, las organizaciones han ido adoptando métodos basados en los valores y principios propuestos en el manifiesto de desarrollo ágil de software, el cual fue propuesto en 2001 y generó una serie de prácticas que creen que puede brindar mayor valor a los clientes. [27]

Para [28] el desarrollo ágil de software en la gestión de proyectos enfatiza un enfoque incremental para el desarrollo de software del proyecto, en lugar de seguir una secuencia predefinida de fases. Este movimiento pone más énfasis en los factores humanos involucrados en el desarrollo de software, en lugar de los procesos y herramientas, las pruebas de software en curso y la cooperación entre desarrolladores y clientes. Por tanto, la implementación del desarrollo de software ágil permite a las organizaciones responder rápidamente a los cambios en el entorno empresarial, con el foco en la adaptabilidad a la incertidumbre y los cambios frecuentes.

Según lo indica [29] el Manifiesto Ágil refleja una filosofía de desarrollo de software que enfatiza un entorno con recursos escasos y una demanda fluctuante. El desarrollo ágil se logra a través de varios métodos, entre los diversos estilos de Agile, Extreme Programming (XP) y Scrum son los dos más

estudiados. De manera similar, estos dos métodos también se consideran los métodos más utilizados en la industria del desarrollo de software.

Adoptar prácticas ágiles es un proceso muy complejo que involucra cambios en el proceso de programación, así como cambios en la cultura organizacional, patrones sociales y el comportamiento de los actores involucrados; sin embargo, sus beneficios potenciales los hacen notables, especialmente para grandes empresas emergentes y grandes organizaciones. [30]

2.2.4. Desarrollo multi plataforma

Según indica [23] el desarrollar la misma aplicación para diferentes plataformas significa repetir el mismo trabajo muchas veces, porque cada proveedor de plataforma proporciona a los desarrolladores diferentes lenguajes de programación y herramientas de desarrollo.

Generalmente, las aplicaciones móviles se pueden desarrollar como aplicaciones web, nativas o híbridas. La aplicación web se ejecuta en un servidor basado en web, se puede acceder a ella a través de un navegador web de escritorio / móvil y es altamente portátil en múltiples plataformas. Debido a que estas aplicaciones se pueden usar en múltiples plataformas, el tiempo requerido para el ciclo de desarrollo e implementación y los costos asociados con el desarrollo de aplicaciones se reducen. [31]

El desarrollo de aplicaciones móviles utilizando marcos multiplataforma no significa intrínsecamente que la disponibilidad de la aplicación en múltiples plataformas sea el objetivo final; también podrían ser otros factores, como la competencia técnica interna. [32]

2.2.5. Mobile Commerce

El comercio móvil se refiere a todas las actividades relacionadas con transacciones comerciales a través de redes de comunicación que interactúan con dispositivos inalámbricos, así como cualquier transacción que implique la transferencia de propiedad o el derecho a usar bienes y servicios, y estas transacciones se inician y/o completan a través de dispositivos móviles. El comercio móvil aún se encuentra en la etapa de desarrollo y hasta ahora está recibiendo una atención asombrosa. [33]

Según la investigación de [34] la sola percepción de utilizar el comercio móvil como entretenimiento y experiencias divertidas tiene un impacto directo y positivo en la intención de los consumidores ecuatorianos de bajos ingresos de utilizar el comercio móvil.

Como lo indica [35] proporciona muchos beneficios a las organizaciones, incluida una mayor productividad, una mayor satisfacción del cliente y una reducción de los costos operativos; se considera una forma rentable para que las empresas promocionen sus productos y servicios en línea, y cada vez tiene más popularidad en todo el mundo, el aumento significativo en los ingresos del mercado global es un ejemplo.

2.2.6. Metodología XP

La metodología XP está diseñada para enfocarse en los usuarios, pero no define los principios de integración con usabilidad e interacción humano-computador (HCI), que trae varios desafíos, como la integración interna de conceptos de evaluación de HCI. Iteraciones, entrada ad hoc, falta de una vista holística en XP y la necesidad de ejecutar pruebas de usabilidad. [36]

En XP (Programación extrema), el trabajo en parejas es una de las doce mejores prácticas para el desarrollo de software. La cantidad de personas en un equipo de proyecto debe estar entre tres y cuatro, y las reglas de emparejamiento de personal son similares a la programación por pares. [37]

Se considera que el método XP implementa las recomendaciones del manifiesto ágil principalmente a través de 12 prácticas, que se aplican a lo largo del desarrollo de software. [29]

2.3 Metodología

Teniendo en cuenta el enfoque ágil adoptado en la creación de este prototipo, se decidió utilizar el método XP porque el trabajo se realiza bajo un estricto control de las actividades en secuencia, iteraciones cortas y el uso de buenas prácticas en el desarrollo de software, características que son más compatibles con esta metodología que con otras metodologías ágiles.

2.4 Objetivos del prototipo

2.4.1. Objetivo general

Desarrollar una aplicación móvil tipo tienda online para su integración con un sistema visor de realidad aumentada mediante programación multiplataforma.

2.4.2. Objetivos específicos

- Programar la aplicación móvil para sistemas iOS mediante React Native.
- Aplicar las fases de la metodología XP durante el proceso de desarrollo.
- Definir los requerimientos funcionales y no funcionales respecto al funcionamiento de la aplicación.
- Diseñar las interfaces, vistas de la aplicación y descripción del proceso lógico mediante herramientas CASE.
- Desarrollar las pruebas unitarias y evaluación de calidad de la aplicación.

2.5 Fase de planificación

La fase de planificación es una de las primeras y más importantes fases de la metodología XP, en esta etapa se describen los procesos que describen el flujo de trabajo del proyecto. En esta sección se describe la definición de roles, el cronograma de actividades, las historias de usuario y el análisis del sistema.

2.5.1. Definición de roles

Tabla 4: Definición de roles

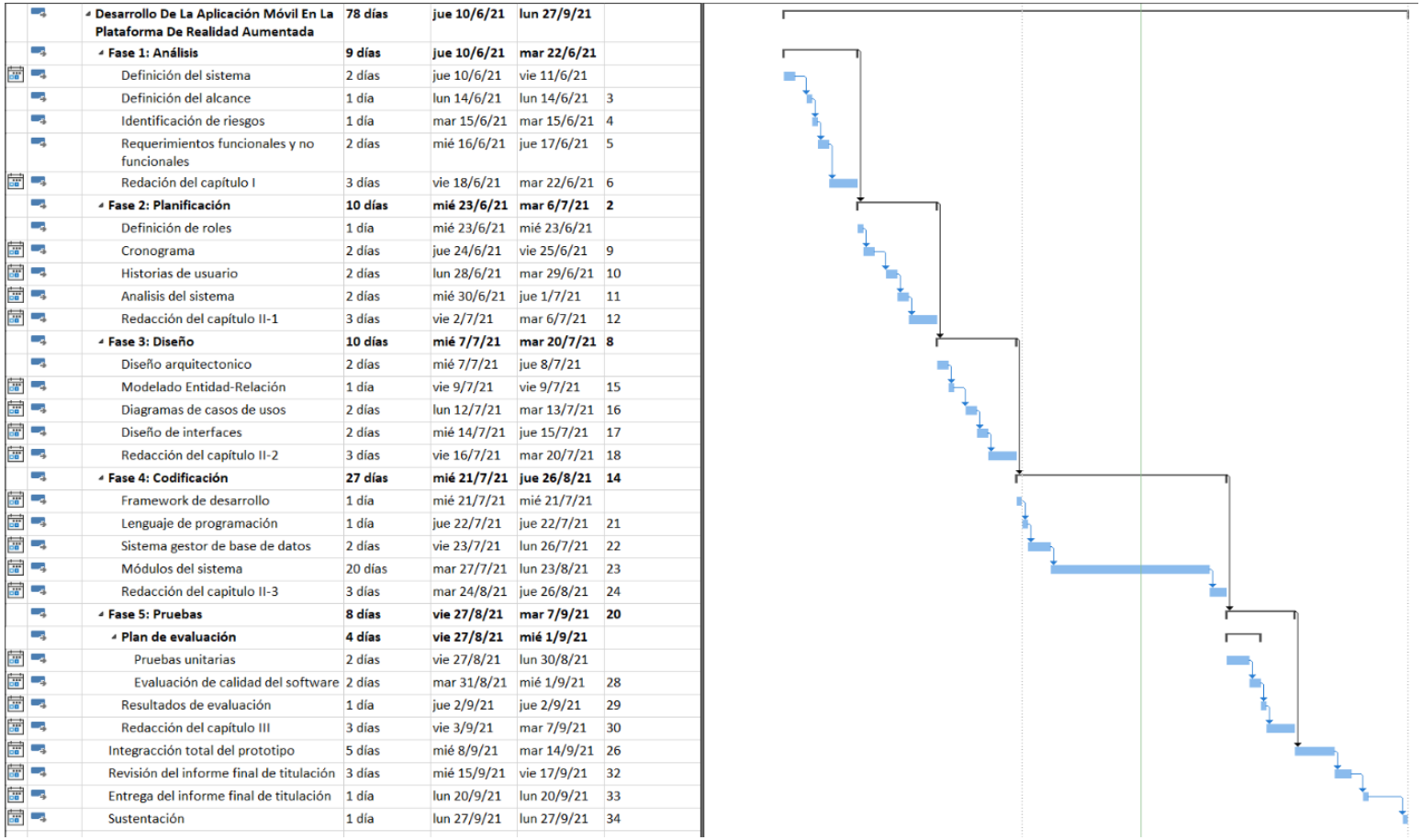
Perfil de Cargo			
Nombre del cargo:	Desarrollador		
Departamento:	Desarrollo	**Sección**	
Elaborado por:	Ricardo Ramón	**Ejerce supervisión sobre:**	----
Descripción general	Persona encargada del desarrollo de la aplicación móvil.		
Descripción del cargo			
Funciones/Responsabilidades			
1.	Bocetos de diseño de aplicación		
2.	Diseño tentativo de interfaces gráficas		
3.	Prototipo de aplicación		
4.	Desarrollo de interfaz interactivas		
5.	Diseño de interfaces específicas para objetos e-commerce		
6.	Interfaz de e-commerce		

7.	Implementación de métodos y pasarelas de pagos					
8.	Diseño del catálogo de e-commerce					
Análisis de competencias necesarias para el puesto						
Grado académico y área de estudio						
Educ. Básica completa (primaria)			**X**	General unificado		
Educ. Media completa (secundaria)			**X**	Técnico (Electrónica)		
Educ. Técnico-profesional (técnico medio o superior)			**-**	----------		
Educ. Universitaria			**X**	Ingeniería en sistema		
Experiencia						
	Exp. Laboral	**Exp. En el campo**		**Exp. En la especialidad**	**Exp. Rubro / negocio**	
Meses						
Años						
Observaciones:						
Idiomas						
	Frecuencia con que lo usa			**Nivel de Manejo**		
Idioma	**Diaria**	**Semanal**	**Mensual**	**Básico**	**Medio**	**Alto**
Inglés		X			X	
Español	X					X
Programas computacionales						
Balsamiq Wireframes			Xcode			
Stripe			Strapi			
Visual Studio Code						

Fuente: Elaboración propia

2.5.2. Cronograma

Figura 1: Cronograma de actividades



	Desarrollo De La Aplicación Móvil En La Plataforma De Realidad Aumentada	78 días	jue 10/6/21	lun 27/9/21	
	Fase 1: Análisis	**9 días**	**jue 10/6/21**	**mar 22/6/21**	
	Definición del sistema	2 días	jue 10/6/21	vie 11/6/21	
	Definición del alcance	1 día	lun 14/6/21	lun 14/6/21	3
	Identificación de riesgos	1 día	mar 15/6/21	mar 15/6/21	4
	Requerimientos funcionales y no funcionales	2 días	mié 16/6/21	jue 17/6/21	5
	Redación del capítulo I	3 días	vie 18/6/21	mar 22/6/21	6
	Fase 2: Planificación	**10 días**	**mié 23/6/21**	**mar 6/7/21**	**2**
	Definición de roles	1 día	mié 23/6/21	mié 23/6/21	
	Cronograma	2 días	jue 24/6/21	vie 25/6/21	9
	Historias de usuario	2 días	lun 28/6/21	mar 29/6/21	10
	Analisis del sistema	2 días	mié 30/6/21	jue 1/7/21	11
	Redacción del capítulo II-1	3 días	vie 2/7/21	mar 6/7/21	12
	Fase 3: Diseño	**10 días**	**mié 7/7/21**	**mar 20/7/21**	**8**
	Diseño arquitectonico	2 días	mié 7/7/21	jue 8/7/21	
	Modelado Entidad-Relación	1 día	vie 9/7/21	vie 9/7/21	15
	Diagramas de casos de usos	2 días	lun 12/7/21	mar 13/7/21	16
	Diseño de interfaces	2 días	mié 14/7/21	jue 15/7/21	17
	Redacción del capítulo II-2	3 días	vie 16/7/21	mar 20/7/21	18
	Fase 4: Codificación	**27 días**	**mié 21/7/21**	**jue 26/8/21**	**14**
	Framework de desarrollo	1 día	mié 21/7/21	mié 21/7/21	
	Lenguaje de programación	1 día	jue 22/7/21	jue 22/7/21	21
	Sistema gestor de base de datos	2 días	vie 23/7/21	lun 26/7/21	22
	Módulos del sistema	20 días	mar 27/7/21	lun 23/8/21	23
	Redacción del capitulo II-3	3 días	mar 24/8/21	jue 26/8/21	24
	Fase 5: Pruebas	**8 días**	**vie 27/8/21**	**mar 7/9/21**	**20**
	Plan de evaluación	**4 días**	**vie 27/8/21**	**mié 1/9/21**	
	Pruebas unitarias	2 días	vie 27/8/21	lun 30/8/21	
	Evaluación de calidad del software	2 días	mar 31/8/21	mié 1/9/21	28
	Resultados de evaluación	1 día	jue 2/9/21	jue 2/9/21	29
	Redacción del capítulo III	3 días	vie 3/9/21	mar 7/9/21	30
	Integracción total del prototipo	5 días	mié 8/9/21	mar 14/9/21	26
	Revisión del informe final de titulación	3 días	mié 15/9/21	vie 17/9/21	32
	Entrega del informe final de titulación	1 día	lun 20/9/21	lun 20/9/21	33
	Sustentación	1 día	lun 27/9/21	lun 27/9/21	34

Fuente: Elaboración propia

2.5.3. Historias de usuario

Una historia de usuario es una descripción breve y simple de la función desde la perspectiva de la persona que necesita la nueva función (generalmente el usuario o cliente del comportamiento del sistema). Las historias de usuario son especialmente adecuadas para describir sistemas de software en los que las personas y las organizaciones desempeñan un papel clave, como las aplicaciones móviles actuales. [38]

Tabla 5: Historia de usuario de registro de clientes

Historia de Usuario	
Número: 1	**Usuario:** Cliente
Nombre historia: Registro de información usuario	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Alta
Días estimados: 1	**Iteración asignada:** 1
Programador responsable: Ricardo Ramón	
Descripción: El usuario ingresa sus datos de registro en un formulario validado que enviará esa información a la base de datos para usarse en un posterior logueo.	
Observaciones: Marca en rojo los campos que no se han ingresado correctamente.	

Fuente: Elaboración propia

Tabla 6: Historia de usuario inicio de sesión del cliente

Historia de Usuario	
Número: 2	**Usuario:** Cliente
Nombre historia: Loguin del usuario	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Alta
Días estimados: 1	**Iteración asignada:** 1
Programador responsable: Ricardo Ramón	
Descripción: El usuario ingresa su correo electrónico/usuario y su contraseña para acceder en la aplicación.	

Observaciones: Marca en rojo los campos que no se han ingresado correctamente.

Fuente: Elaboración propia

Tabla 7: Historia de usuario de sistema de navegación

Historia de Usuario	
Número: 3	**Usuario:** Administrador
Nombre historia: Navegación entre pestañas	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Media
Días estimados: 1	**Iteración asignada:** 1
Programador responsable: Ricardo Ramón	
Descripción: El usuario podrá cambiar las pantallas dentro de la aplicación (Productos/Favoritos/Carrito/Cuenta).	
Observaciones: Ninguna.	

Fuente: Elaboración propia

Tabla 8: Historia de usuario agregar/modificar nombre

Historia de Usuario	
Número: 4	**Usuario:** Cliente
Nombre historia: Agregar y/o actualizar nombres	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Media
Días estimados: 1	**Iteración asignada:** 2
Programador responsable: Ricardo Ramón	
Descripción: El usuario agregará nombre y apellidos a su cuenta y podrá actualizarlos.	
Observaciones: Marca en rojo los campos que no se han ingresado correctamente.	

Fuente: Elaboración propia

Tabla 9: Historia de usuario modificar correo

Historia de Usuario	
Número: 5	**Usuario:** Cliente
Nombre historia: Actualizar e-mail	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Alta

Días estimados: 1	**Iteración asignada:** 2
Programador responsable: Ricardo Ramón	
Descripción: El usuario podrá actualizar el correo electrónico para el inicio de sesión.	
Observaciones: Marca en rojo los campos que no se han ingresado correctamente.	

Fuente: Elaboración propia

Tabla 10: Historia de usuario modificar username

Historia de Usuario	
Número: 6	**Usuario:** Cliente
Nombre historia: Actualizar nombre de usuario	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Alta
Días estimados: 1	**Iteración asignada:** 2
Programador responsable: Ricardo Ramón	
Descripción: El usuario podrá actualizar el nombre de usuario para el inicio de sesión.	
Observaciones: Marca en rojo los campos que no se han ingresado correctamente.	

Fuente: Elaboración propia

Tabla 11: Historia de usuario modificar contraseña

Historia de Usuario	
Número: 7	**Usuario:** Cliente
Nombre historia: Actualizar contraseña	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Alta
Días estimados: 1	**Iteración asignada:** 2
Programador responsable: Ricardo Ramón	
Descripción: El usuario podrá actualizar la contraseña para el inicio de sesión.	
Observaciones: Ninguna.	

Fuente: Elaboración propia

Tabla 12: Historia de usuario crear direcciones de envío

Historia de Usuario

Número: 8	**Usuario:** Cliente
Nombre historia: Crear direcciones de envío	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Alta
Días estimados: 1	**Iteración asignada:** 2
Programador responsable: Ricardo Ramón	
Descripción: El usuario podrá crear direcciones para el envío de sus pedidos.	
Observaciones: Marca en rojo los campos que no se han ingresado correctamente.	

Fuente: Elaboración propia

Tabla 13: Historias de usuario editar direcciones de envío

Historia de Usuario	
Número: 9	**Usuario:** Cliente
Nombre historia: Modificar direcciones de envío	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Baja
Días estimados: 0.5	**Iteración asignada:** 2
Programador responsable: Ricardo Ramón	
Descripción: El usuario podrá modificar sus direcciones de envío.	
Observaciones: Marca en rojo los campos que no se han ingresado correctamente.	

Fuente: Elaboración propia

Tabla 14: Historia de usuario eliminar direcciones de envío

Historia de Usuario	
Número: 10	**Usuario:** Cliente
Nombre historia: Eliminar direcciones de envío	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Baja
Días estimados: 0.5	**Iteración asignada:** 2
Programador responsable: Ricardo Ramón	

Descripción: El usuario podrá eliminar las direcciones que ya no desee.
Observaciones: Ninguna.

Fuente: Elaboración propia

Tabla 15: Historia de usuario agregar productos

Historia de Usuario	
Número: 11	**Usuario:** Administrador
Nombre historia: Agregar productos	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Alta
Días estimados: 1.5	**Iteración asignada:** 3
Programador responsable: Ricardo Ramón	
Descripción: El usuario accede a una vista de los productos almacenados en la base de datos.	
Observaciones: Ninguna.	

Fuente: Elaboración propia

Tabla 16: Historia de usuario administrar favoritos/carrito

Historia de Usuario	
Número: 12	**Usuario:** Cliente
Nombre historia: Agregar producto a favoritos/carrito	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Media
Días estimados: 0.5	**Iteración asignada:** 3
Programador responsable: Ricardo Ramón	
Descripción: El cliente puede agregar el producto que desea a su lista de favoritos y carrito.	
Observaciones: Ninguna.	

Fuente: Elaboración propia

Tabla 17: Historia de usuario agregar banners promocionales

Historia de Usuario	
Número: 13	**Usuario:** Administrador
Nombre historia: Agregar banners promocionales	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Baja

Días estimados: 1	**Iteración asignada:** 3
Programador responsable: Ricardo Ramón	
Descripción: El usuario accede a un banner de productos promocionales que lo redirigen a los productos de la tienda.	
Observaciones: Ninguna.	

Fuente: Elaboración propia

Tabla 18: Historia de usuario búsqueda de productos

Historia de Usuario	
Número: 14	**Usuario:** Cliente
Nombre historia: Búsqueda de productos desde la barra	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Media
Días estimados: 1	**Iteración asignada:** 3
Programador responsable: Ricardo Ramón	
Descripción: El usuario puede buscar un producto con palabras clave que se obtienen desde el nombre y etiquetas del producto.	
Observaciones: Ninguna.	

Fuente: Elaboración propia

Tabla 19: Historia de usuario búsqueda desde historial

Historia de Usuario	
Número: 15	**Usuario:** Cliente
Nombre historia: Búsqueda de productos desde el historial	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Media
Días estimados: 1	**Iteración asignada:** 3
Programador responsable: Ricardo Ramón	
Descripción: El usuario puede buscar un producto desde el historial de búsqueda de productos previos.	
Observaciones: Ninguna.	

Fuente: Elaboración propia

Tabla 20: Historia de usuario agregar a favoritos

Historia de Usuario	
Número: 16	**Usuario:** Cliente
Nombre historia: Agregar productos a favoritos	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Alta
Días estimados: 1	**Iteración asignada:** 4
Programador responsable: Ricardo Ramón	
Descripción: El usuario puede seleccionar los productos que quiere añadir a favoritos.	
Observaciones: Ninguna.	

Fuente: Elaboración propia

Tabla 21: Historia de usuario eliminar de favoritos

Historia de Usuario	
Número: 17	**Usuario:** Cliente
Nombre historia: Eliminar productos de favoritos	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Media
Días estimados: 1	**Iteración asignada:** 4
Programador responsable: Ricardo Ramón	
Descripción: El usuario puede eliminar los productos que desee de su lista de favoritos.	
Observaciones: Ninguna.	

Fuente: Elaboración propia

Tabla 22: Historia de usuario agregar al carrito

Historia de Usuario	
Número: 18	**Usuario:** Cliente
Nombre historia: Agregar productos al carrito	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Alta
Días estimados: 1	**Iteración asignada:** 4
Programador responsable: Ricardo Ramón	
Descripción: El usuario puede seleccionar los productos que quiere añadir al carrito de compras.	

Observaciones: Ninguna.

Fuente: Elaboración propia

Tabla 23: Historia de usuario eliminar del carrito

Historia de Usuario	
Número: 19	**Usuario:** Cliente
Nombre historia: Eliminar productos del carrito	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Media
Días estimados: 1	**Iteración asignada:** 4
Programador responsable: Ricardo Ramón	
Descripción: El usuario puede eliminar los productos que desee de su carrito de compras.	
Observaciones: Ninguna.	

Fuente: Elaboración propia

Tabla 24: Historia de usuario generar pagos

Historia de Usuario	
Número: 20	**Usuario:** Cliente
Nombre historia: Generación de pagos	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Alta
Días estimados: 1	**Iteración asignada:** 4
Programador responsable: Ricardo Ramón	
Descripción: El usuario puede ejecutar la compra de los productos de su carrito.	
Observaciones: Ni la aplicación ni la base de datos guardan la información de la tarjeta.	

Fuente: Elaboración propia

Tabla 25: Historia de usuario lista de pedidos

Historia de Usuario	
Número: 21	**Usuario:** Cliente
Nombre historia: Listado de pedidos	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Media
Días estimados: 1	**Iteración asignada:** 4
Programador responsable: Ricardo Ramón	

Descripción:
El usuario obtiene la información de cada pedido generado.
Observaciones: Ninguna.

Fuente: Elaboración propia

2.5.4. Análisis del sistema

El análisis del sistema consta del plan de publicación, el plan de duración de iteraciones, el plan de entrega y el cálculo de la velocidad del proyecto.

2.5.4.1. Plan de publicaciones

El plan de publicación desarrollado ordena las actividades a desarrollar según la prioridad de las historias de usuario, debido a que el desarrollo de algunas actividades depende del cumplimiento de otras.

En el plan de publicación se indica la tarea general, las historias de usuario que se desarrollan en ella, la prioridad de desarrollo, el riesgo de desarrollo y el tiempo de duración de la actividad en días.

Tabla 26: Plan de publicaciones

N°	Tarea	Historia de usuario	Prioridad	Riesgo	Días estimados
1	Autenticación del usuario	Registro de información usuario	Alta	Alta	2 días
		Loguin del usuario	Alta	Alta	
2	Sistema de navegación	Sistema de navegación	Alta	Media	1 días
3	Información de la cuenta	Agregar y/o actualizar nombres	Alta	Media	4 días
		Actualizar e-mail	Alta	Alta	
		Actualizar nombre de usuario	Alta	Alta	
		Actualizar contraseñas	Alta	Alta	

4	Sistema de direcciones	Crear direcciones de envío	Alta	Alta	2 día
		Modificar direcciones de envío	Alta	Baja	
		Eliminar direcciones de envío	Alta	Baja	
5	Sistema de productos	Agregar productos	Alta	Alta	2 días
		Agregar producto a favoritos/carrito	Alta	Media	
6	Home Slider	Agregar banners promocionales	Alta	Baja	1 día
7	Sistema de búsqueda	Búsqueda de productos desde la barra	Alta	Media	2 días
		Búsqueda de productos desde el historial	Alta	Media	
8	Sistema de favoritos	Agregar productos a favoritos	Alta	Alta	2 días
		Eliminar productos a favoritos	Alta	Media	
9	Sistema de carrito	Agregar productos al carrito	Alta	Alta	2 días
		Eliminar productos del carrito	Alta	Media	
10	Sistema de pago	Generación de pagos	Alta	Alta	2 días
		Listado de pedidos	Alta	Media	

Fuente: Elaboración propia

2.5.4.2. Plan de duración de iteraciones

Después de definir las historias de usuario agrupadas en diferentes tareas se deben agrupar estas tareas en iteración distintas, cada iteración contiene las historias de usuario agrupadas según la actividad que desarrollan; en la primera

iteración se desarrolla el módulo de inicio de sesión (login y registro) y el stack de navegación general de la aplicación; en la segunda iteración se desarrollan los formularios para el manejo de información del usuario y las direcciones asociadas a su cuenta; en la tercera iteración se desarrollan los contenidos relacionados a los productos (ingreso en base de datos, pantallas de información individual, banners promocionales, búsqueda) y en la cuarta iteración se maneja la asignación de productos a los favoritos y pasarelas de pago.

Tabla 27: Plan de duración de iteraciones

Iteración	Orden de las historias de usuario	Duración de la iteración
Primera Iteración	Autenticación del usuario	3 días
	Sistema de navegación	
Segunda Iteración	Información de la cuenta	6 días
	Sistema de direcciones	
Tercera Iteración	Sistema de productos	5 días
	Home Slider	
	Sistema de búsqueda	
Cuarta Iteración	Sistema de favoritos	6 días
	Sistema de carrito	
	Sistema de pago	

Fuente: Elaboración propia

2.5.4.3. Plan de entrega

Las historias de usuario se agrupan según las iteraciones a las que pertenecen, ordenados según su proceso de desarrollo y con las fechas que conciernen al desarrollo del prototipo según el cronograma. Para la duración y asignación de las fechas se consideraron únicamente los días hábiles entre semana; la siguiente tabla indica las fechas del inicio y fin de cada iteración:

Tabla 28: Plan de entrega

Iteración	Fecha de entrega
Primera Iteración	27/07/2021 - 29/07/2021
Segunda Iteración	30/07/2021 - 06/08/2021

Tercera Iteración	09/08/2021 - 13/08/2021
Cuarta Iteración	16/08/2021 - 23/08/2021

Fuente: Elaboración propia

2.5.4.4. Velocidad del proyecto

La velocidad del proyecto es un valor estimado respecto a la cantidad promedio de historias de usuario que se debe cumplir por cada iteración del proyecto.

Dado que la dificultad de algunas actividades es mayor que otras el tiempo que duran las iteraciones varía entre ellas.

Tabla 29: Velocidad del proyecto

Descripción	Primera iteración	Segunda iteración	Tercera iteración	Cuarta iteración
Historia de usuario	3	7	6	6
Tiempo (días)	3	6	5	6

Fuente: Elaboración propia

Para el cálculo de la velocidad del proyecto se aplica la siguiente fórmula:

$$Velocidad = \frac{Número\ de\ historias\ de\ usuario}{Número\ de\ iteraciones}$$

$$Velocidad = \frac{3+7+6+6}{4} = \frac{22}{4}$$

$$Velocidad = 5.5 \approx \boldsymbol{6\ historias\ de\ usuario/iteración}$$

2.6 Diseño del prototipo

Para el diseño del prototipo se consideró un diseño arquitectónico que muestre una representación del funcionamiento de la aplicación móvil; también diagramas de entidad-relación y el modelo relacional, los cuales, a pesar de que la aplicación usa una base de datos no relacional, servirán para hacer una representación más real de cómo funciona la base de datos; seguido de los diagramas de casos de uso que muestra cómo funciona la aplicación del lado del cliente y cómo el usuario interactúa con ella, además de un diseño preliminar de las interfaces a utilizar.

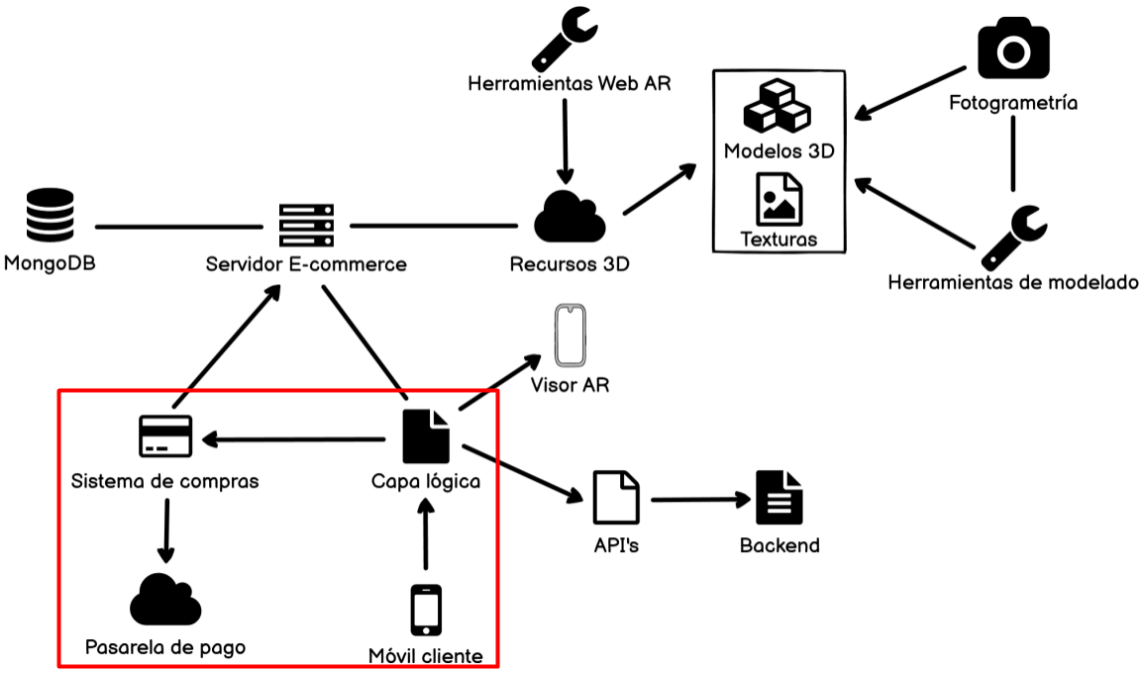
2.6.1. Diseño arquitectónico

La aplicación desarrollada trabaja bajo el modelo de 3 capas, dado que el usuario o cliente interactúa con la aplicación bajo la capa de negocios y ésta realiza operaciones según la información almacenada en la capa de datos.

Según lo indica [39] La arquitectura de software se refiere a la división de una pieza de software en un conjunto de sus componentes, patrones y relaciones visibles externamente. Estas relaciones y patrones definen la colaboración entre ellos para realizar sus funciones, convirtiéndose así en una herramienta estándar para describir y diseñar software, porque Proporcionar un vocabulario general para simular el diseño global del software a través de la estructura del software y el comportamiento en línea con los objetivos comerciales.

El diseño arquitectónico de la aplicación engloba todos los procesos aplicados para su desarrollo y funcionamiento.

Figura 2: Diseño arquitectónico del prototipo



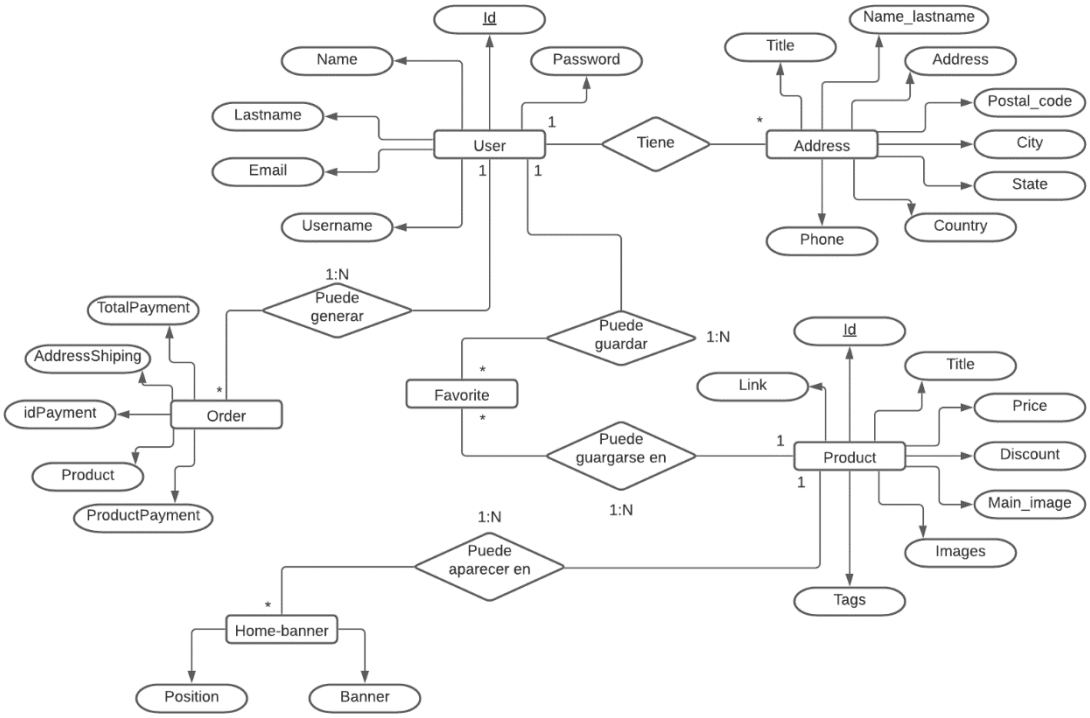
Fuente: Elaboración propia

2.6.2. Modelo Entidad-Relación y Relacional

A pesar que el lenguaje de modelado unificado se centra en software orientado a objetos, también se puede utilizar para expresar modelo relacional de bases de datos y modelos Entidad-Relación. [40]

Dado que este prototipo tiene una base de datos no relacional (MongoDB), los modelos entidad-relacional y entidad-relacional están diseñados para ayudarlos a comprender mejor cómo funciona la base de datos.

Figura 3: Modelo entidad-relación



Fuente: Elaboración propia

User (<u>Id</u>, Name, Lastname, Email, Username, Password)

Address (Title, Name_lastname, Address, Postal_code, City, State, Country, Phone, **<u>User</u>**)

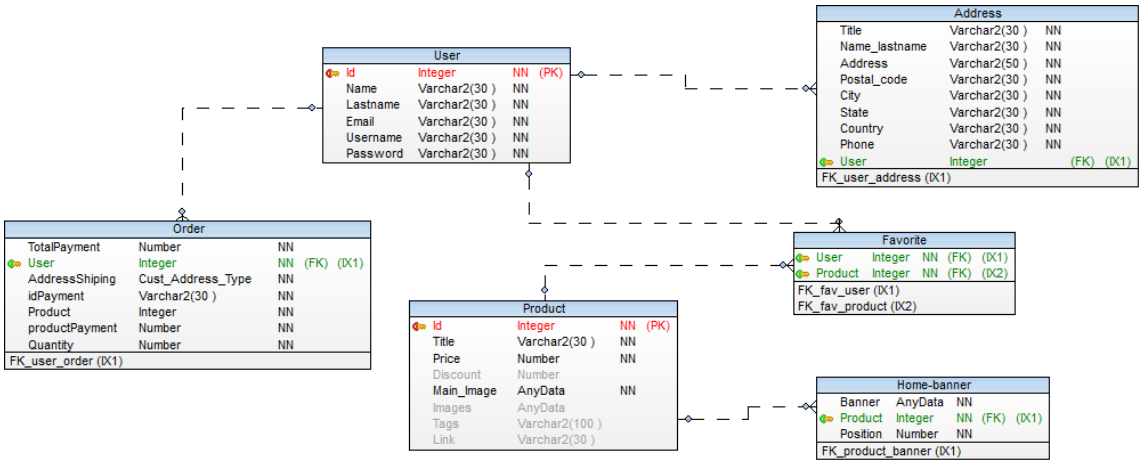
Order (**<u>User</u>**, TotalPayment, AddressShiping, idPayment, Product, ProductPayment)

Product (<u>Id</u>, Title, Price, Discount, Main_image, Tags, Link)

Favorite (**<u>User</u>**, **<u>Product</u>**)

Home-banner (**Product**, Banner, Position)

Figura 4: Modelo relacional



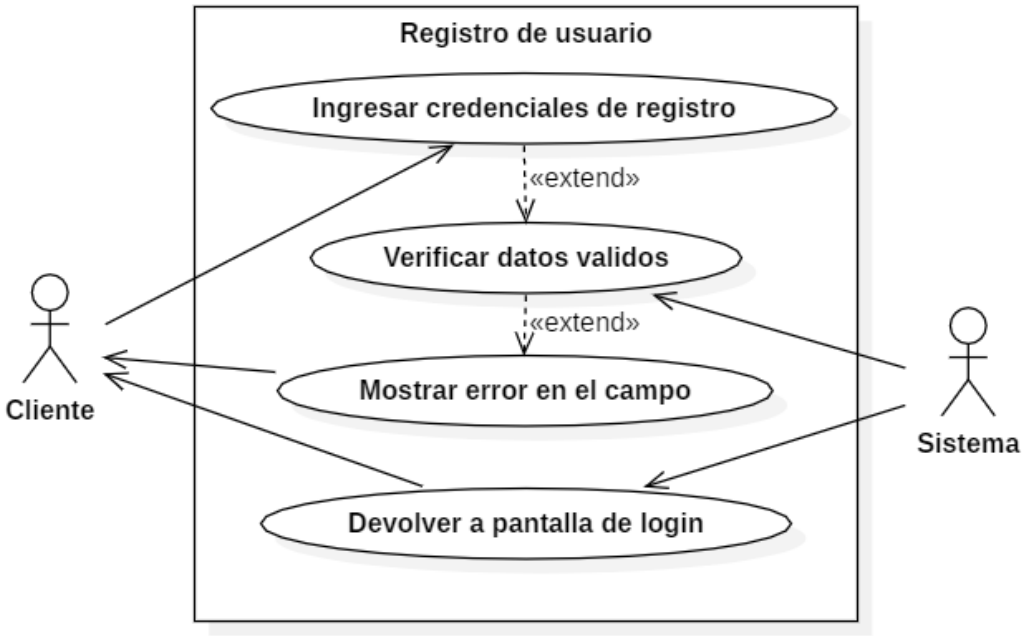
Fuente: Elaboración propia

2.6.3. Diagrama de casos de uso

El diagrama de casos de uso representa en forma del diagrama el comportamiento de una aplicación mediante un lenguaje modelado unificado.

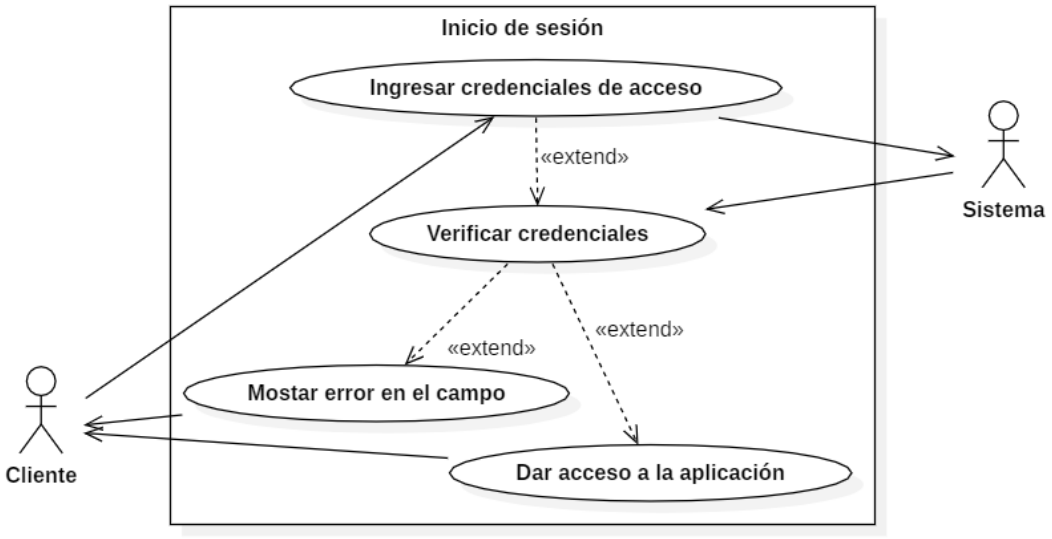
Este diagrama se realiza a partir de editor de diagramas de casos de uso, el cual proporciona la posibilidad de dibujar los componentes del diagrama de casos de uso como actor, caso de uso y asociación (asociación simple, incluir, extender). [41]

Figura 5: Caso de uso registro de usuario



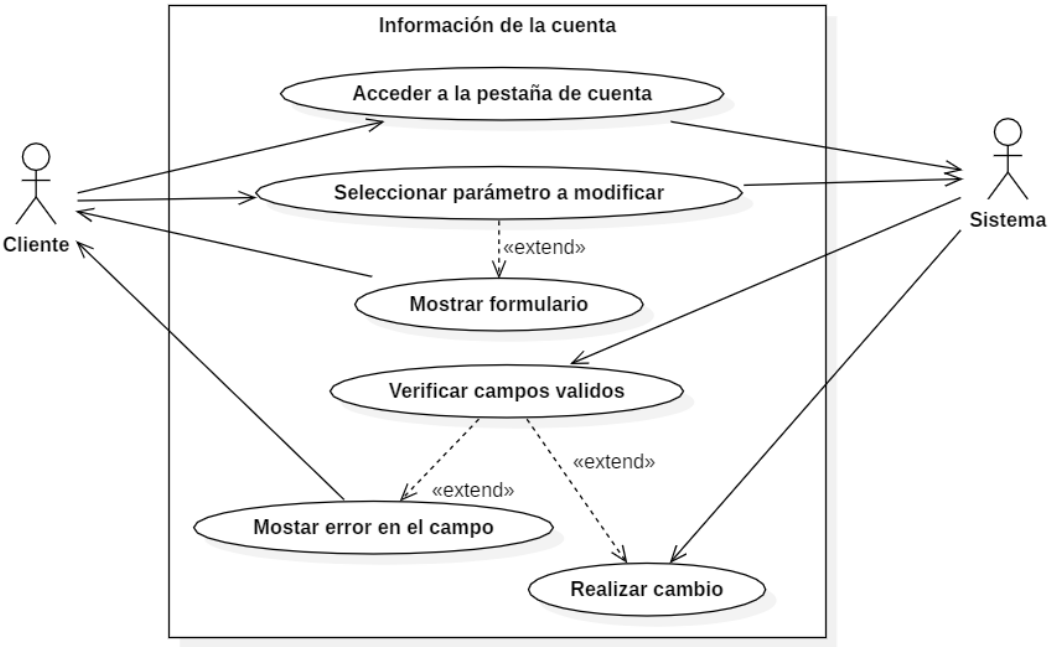
Fuente: Elaboración propia

Figura 6: Caso de uso inició de sesión



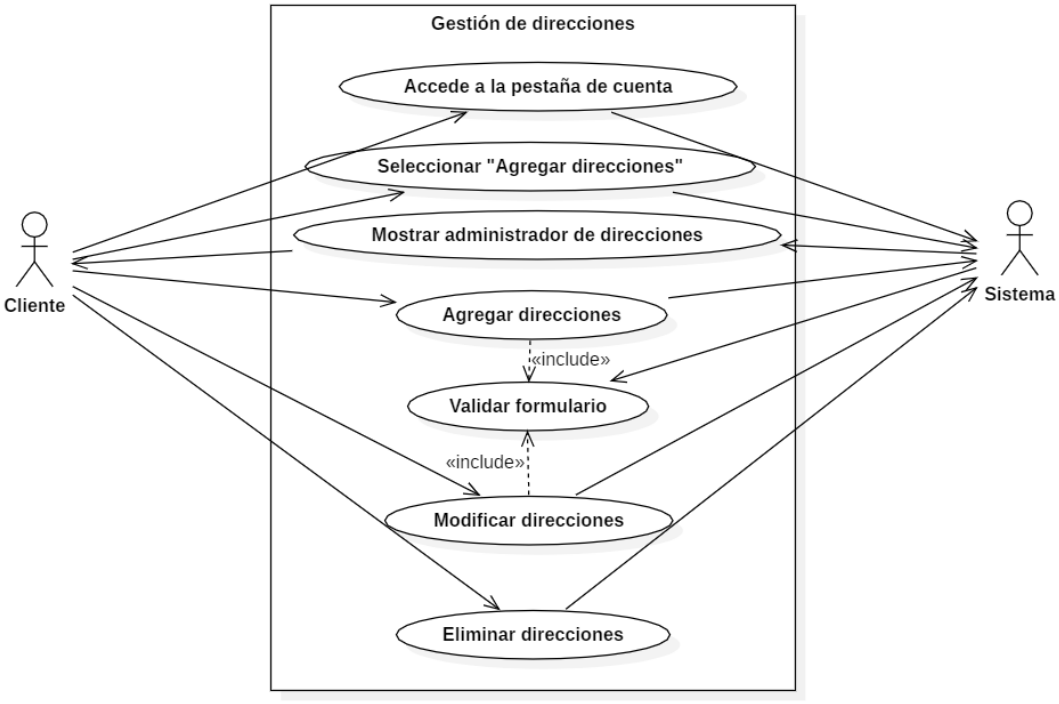
Fuente: Elaboración propia

Figura 7: Caso de uso información de la cuenta



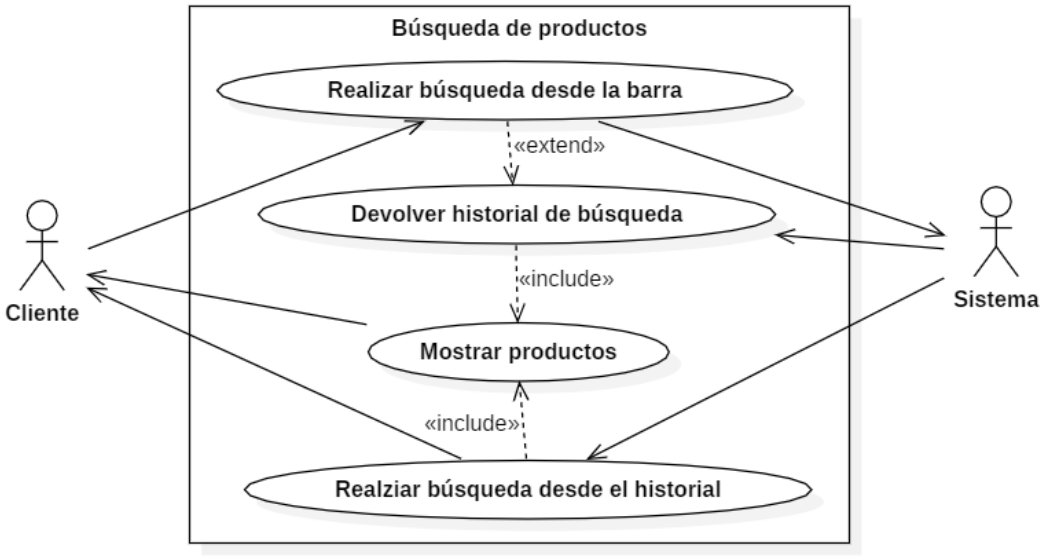
Fuente: Elaboración propia

Figura 8: Caso de uso gestión de direcciones



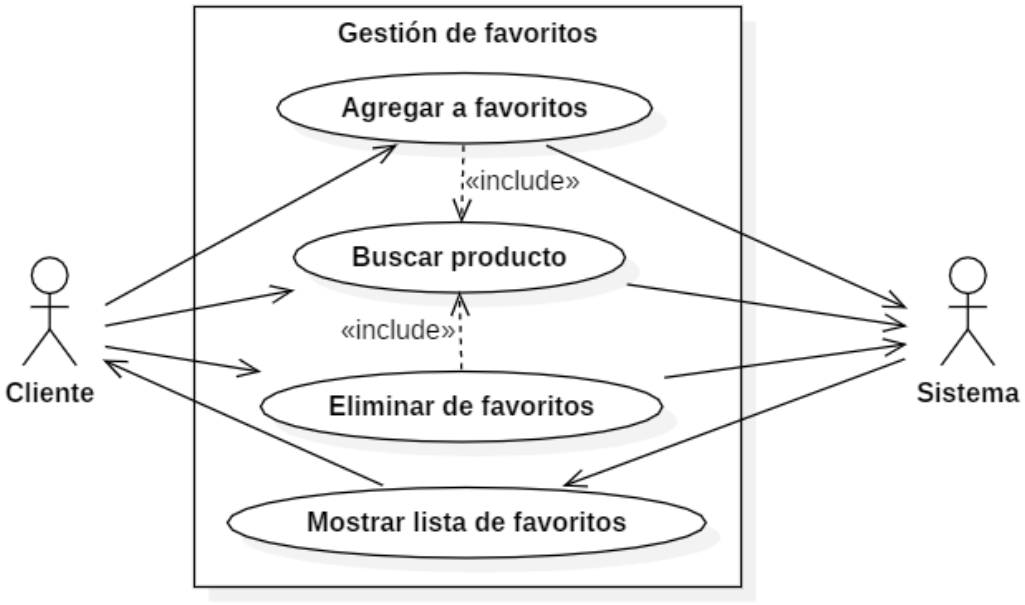
Fuente: Elaboración propia

Figura 9: Caso de uso búsqueda de productos



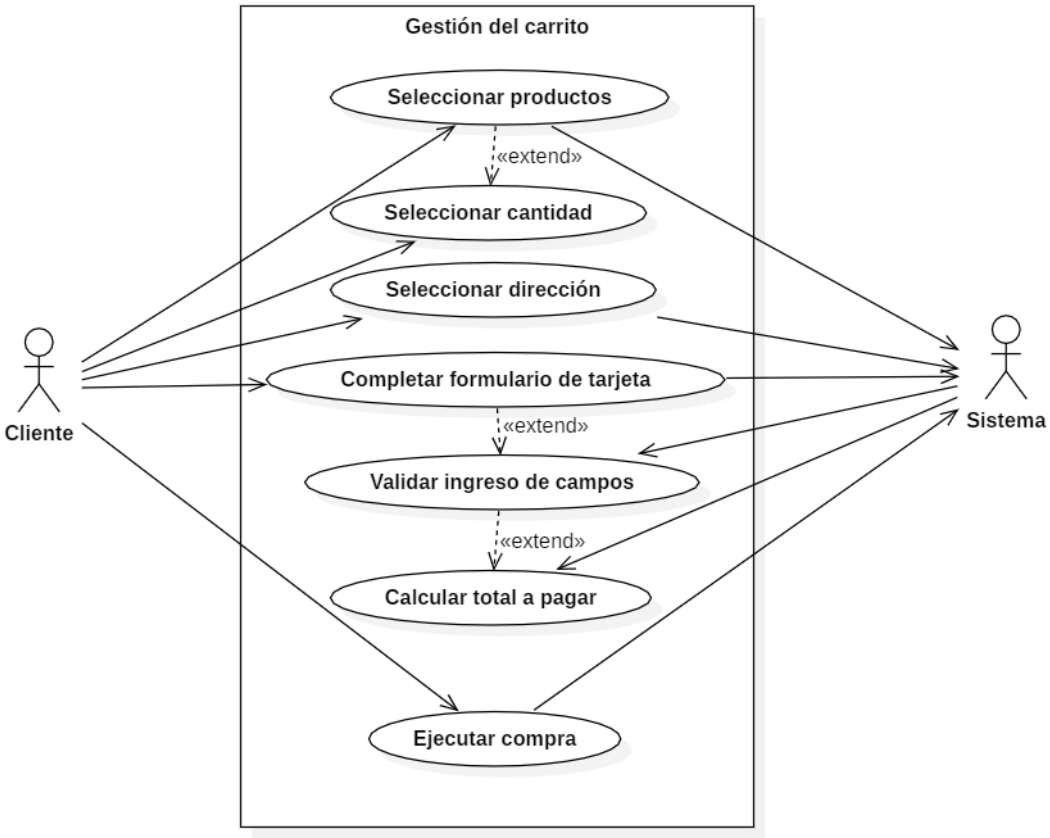
Fuente: Elaboración propia

Figura 10: Caso de uso gestión de favoritos



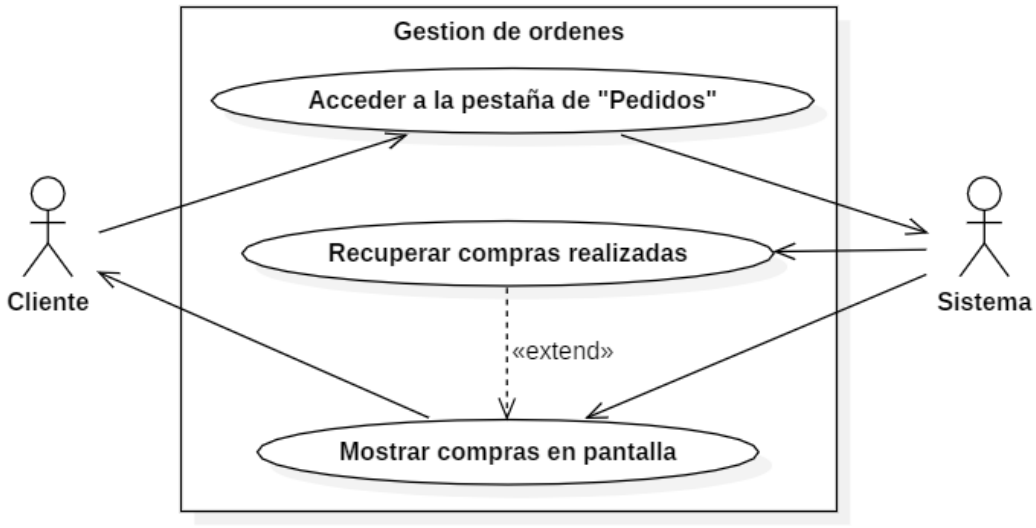
Fuente: Elaboración propia

Figura 11: Caso de uso gestión de carrito



Fuente: Elaboración propia

Figura 12: Caso de uso gestión de órdenes



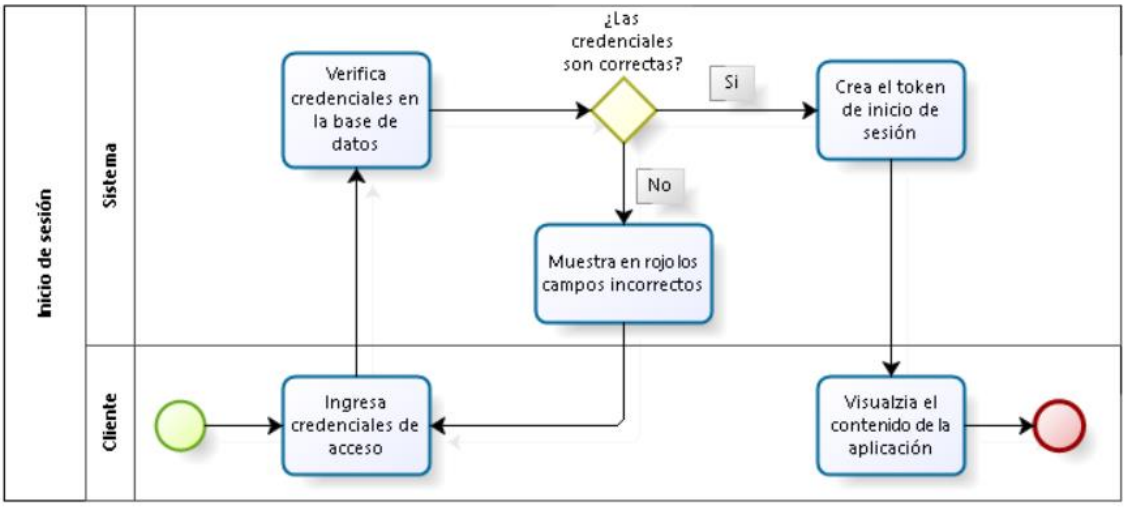
Fuente: Elaboración propia

En el lado del frontend la interacción principal se realiza entre las peticiones del cliente y como el sistema reacciona ante dichas peticiones.

2.6.4. Diagramas de actividad

Los diagramas de actividad son una representación del funcionamiento lógico de las operaciones de la aplicación.

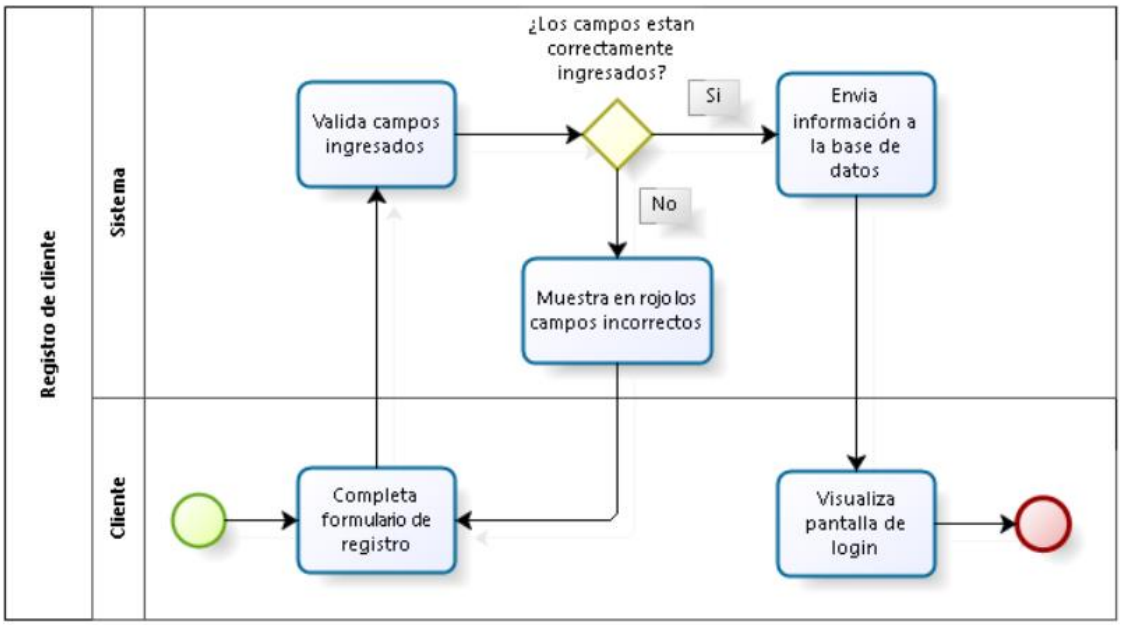
Figura 13: Diagrama de actividades inicio de sesión



Fuente: Elaboración propia

Para el inicio de sesión el cliente ingresa sus credenciales, las cuales se verifican por parte de la aplicación; si son correctas generará un token local para mantener la sesión abierta en el dispositivo y dará pase a la aplicación; caso contrario pedirá nuevamente las credenciales.

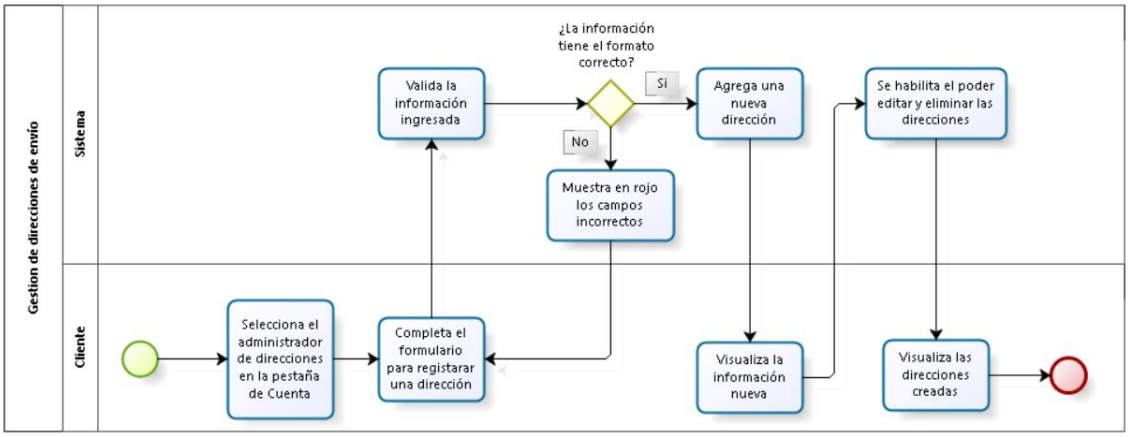
Figura 14: Diagrama de actividades registro de clientes



Fuente: Elaboración propia

Para el registro del cliente se presenta un formulario que el cliente debe rellenar, una vez completo el sistema comprueba si el texto ingresado cumple el formato adecuado; de ser así enviará el registro a la base de datos y mostrar la página de login para que el usuario acceda a la aplicación; caso contrario mostrará el error al cliente, por lo que deberá corregirlo antes de hacer la petición.

Figura 15: Diagrama de actividades gestión de direcciones

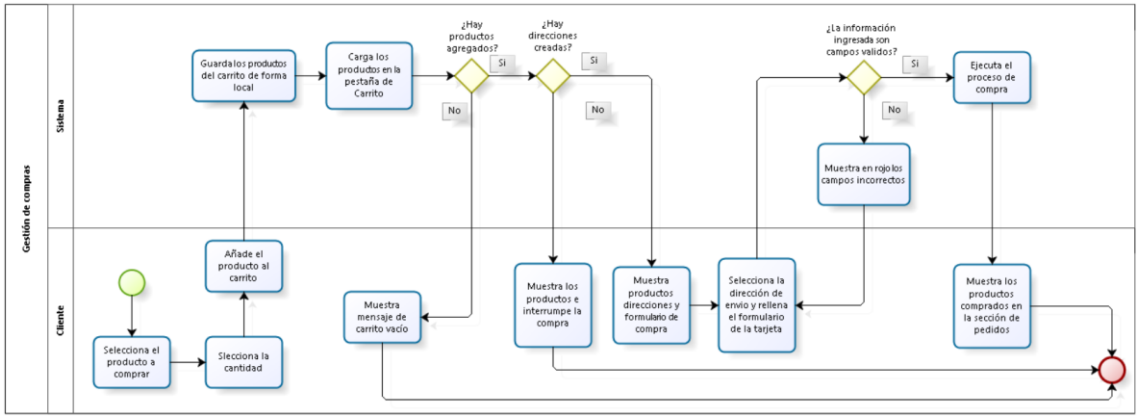


Fuente: Elaboración propia

Para poder realizar el envío se necesita al menos una dirección; para esto el cliente debe seleccionar la opción de Mis direcciones para agregar una nueva; al completar el formulario, si tiene errores, el sistema lo indicará, haciendo que el cliente deba escribirlos de forma adecuada; si el formato se presenta de

manera correcta la dirección será agregada y el cliente podrá verla, editarla y/o borrarla.

Figura 16: Diagrama de actividad gestión de compras



Fuente: Elaboración propia

Para la realización de una compra primeramente se añade al carrito los productos que se desea, seleccionado la cantidad; cuando se añade, el sistema guarda el id de los productos de manera local y los carga en la pestaña del carrito; en dicha pestaña, si no hay productos muestra una pantalla con un mensaje indicando lo dicho anteriormente; si se encuentran productos se carga la lista de direcciones asociadas a la cuenta; s hay productos pero no direcciones entonces se mostraran los productos pero no se podrá acceder al formulario de compra; si hay productos y al menos una dirección entonces el formulario de la tarjeta aparecerá, dicho formulario necesita información válida ingresada, si la información proporcionada no cumple el formato adecuado se deberá corregir para poder hacer el pedido, si todo está bien escrito entonces el sistema genera el pago y carga los productos comprados en la pestaña de pedidos.

2.6.5. Diseño de interfaces

El diseño de interfaces se realizó con la herramienta Balsamiq Mockups, buscando una representación lo más fiel posible al diseño final deseado; para un reconocimiento más sencillo se dividió el diseño de las pantallas por componentes.

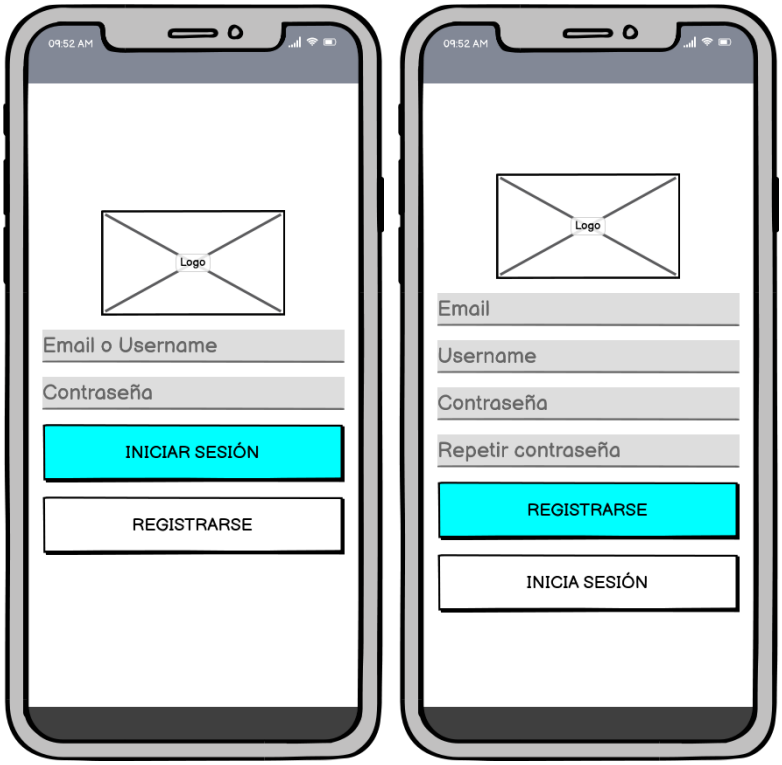
En el componente de login se pueden apreciar dos pantallas:

- **El inicio de sesión**: En esta pantalla se aprecia el logo que representa la aplicación, así como dos inputs para ingresar el correo o usuario y su

contraseña y poder iniciar sesión; si no se tiene aún una cuenta también posee un botón para abrir el formulario de registro.

- **Registro**: Contiene la imagen de logo de la aplicación y un formulario que solicita un correo, un nombre de usuario, contraseña y la confirmación de la misma; Con el botón registrará se guardará esa información en la base de datos de usuarios.

Figura 17: Diseño de interfaz login y registro



Fuente: Elaboración propia

Para la pestaña de cuenta se pueden apreciar las siguientes pantallas:

- **Información de la cuenta**: Información como el nombre y sus direcciones aparecerán en blanco dado que el registro únicamente solicitó un correo, usuario y contraseña; por lo que en esta opción se podrá agregar un nombre, así como modificar los otros parámetros y agregar direcciones de envió; también se encuentra la opción para ver los pedidos realizados, favoritos y cerrar sesión
- **Cerrar sesión**: al cerrar sesión aparecerá una alerta de confirmación en caso de que haya sucedido sin querer.

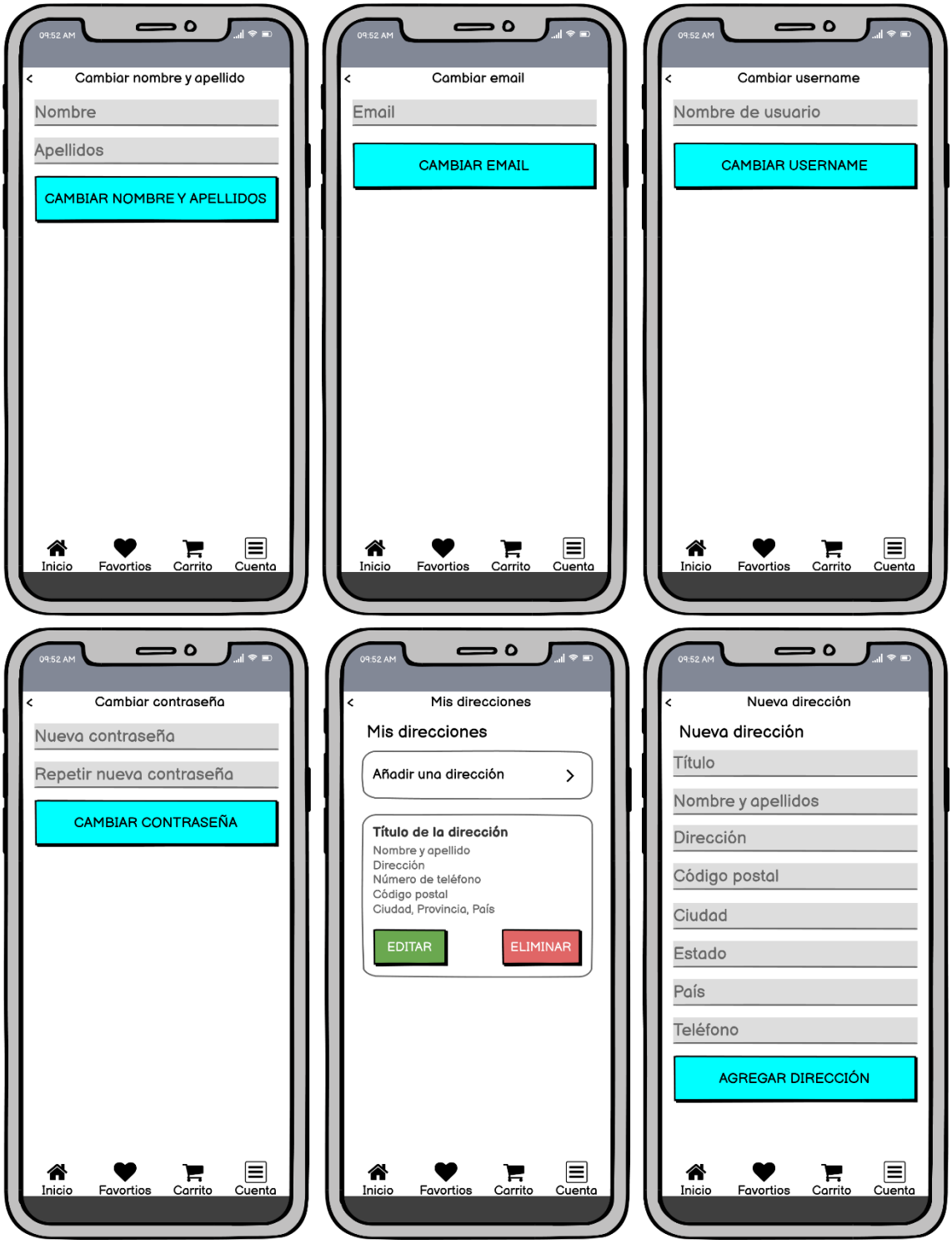
Figura 18: Diseño de interfaz “Cuenta”

Fuente: Elaboración propia

Las pantallas de la configuración de la cuenta son inputs para manipular la información del nombre del cliente, el username, correo, contraseña y las direcciones que se utilizarán para el envío; en el apartado de direcciones se pueden agregar nuevas a través de un formulario simple y para cada dirección creada se podrá modificar o eliminar.

Si ya existe información que cubra esos campos al abrir el formulario se rellenarán con esa información; cada campo posee una expresión regular para el control del ingreso de información.

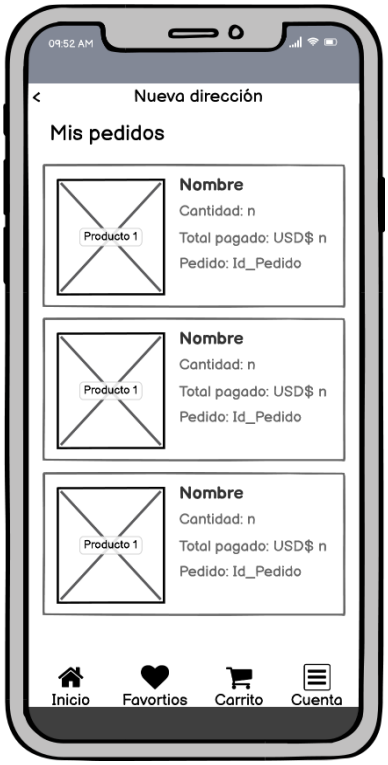
Figura 19: Diseño de interfaz información del usuario



Fuente: Elaboración propia

Dentro de la información de la cuenta se encuentra un apartado para visualizar los pedidos realizados en los cuales se puede apreciar el producto comprado, la cantidad y el id de la compra realizada, dado que en los pedidos se despliegan producto por producto.

Figura 20: Diseño de interfaz “Pedidos”

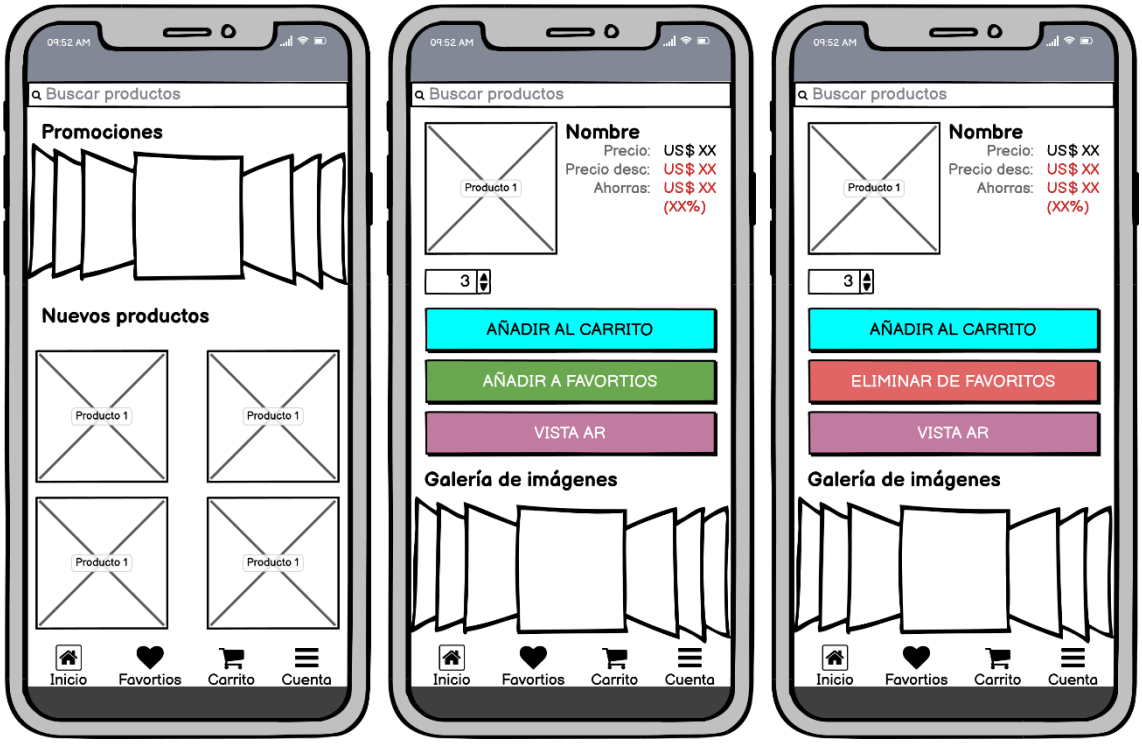


Fuente: Elaboración propia

En la pestaña de inicio del menú principal se encuentran los productos publicados, en la pantalla principal se observa un banner de productos nuevos y/o promocionales, seguido de la lista de productos a ofertar.

Si se selecciona un producto en específico o un imagen del banner se cargara una pantalla con la información específica de cada producto asociado; en esa pantalla se observa el nombre del producto, el precio real, el precio descuento, un selector de cantidad de productos, una galería de imágenes del producto y tres botones; el primero guardará el producto en el carrito de compras, el segundo guardará el producto en la lista de favoritos y el tercero abre una vista de realidad aumentada del producto como un modelo 3D.

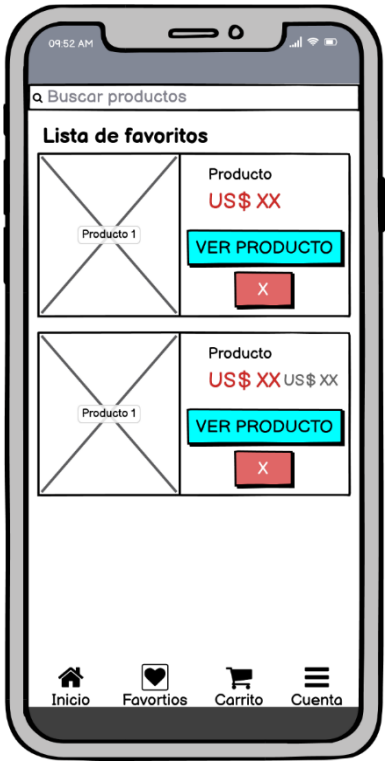
Figura 21: Diseño de interfaz “Productos”



Fuente: Elaboración propia

La pestaña de favoritos muestra una lista de los productos guardados como favoritos desde la pestaña de productos; en esta pantalla se observa una imagen, nombre, precio del producto y un enlace para ver la pantalla del producto.

Figura 22: Diseño de interfaz “Favoritos”



Fuente: Elaboración propia

El carrito de compras es una pantalla que muestra los productos agregados al carrito, en el cual se muestra la imagen, nombre, precio, cantidad y una opción para aumentar o disminuir dicha cantidad.

Seguido de los productos se muestra una lista de direcciones agregadas, en la cual se debe seleccionar cual es la dirección donde se desea realizar el envío, por defecto aparece seleccionada la primera en caso de haber más de una.

Después está el formulario de compra que envía la información de la tarjeta a la pasarela de pago; por razones de seguridad no se guarda la información de la tarjeta en la aplicación ni en la base de datos.

Por último se encuentra un botón para confirmar el pago con el valor total a pagar por los productos agregados.

Figura 23: Diseño de interfaz “Carrito”

Fuente: Elaboración propia

En las pantallas de cuenta, productos, favoritos y en el carrito cuando está vacío aparece una barra de búsqueda en la parte superior de la pantalla, en ella se pueden realizar búsquedas de productos con palabras clave, las cuales se guardarán como un historial de búsqueda para facilitar una búsqueda posterior.

2.7 Ejecución y/o ensamblaje del prototipo

2.7.1. Framework de desarrollo

Los marcos de trabajo aplicados en el desarrollo del prototipo son Expo, dado que permite una ejecución y pruebas del código de manera más sencilla al trabajar con JavaScript, y React Native, dado que se trata de una aplicación nativa orientada principalmente a su despliegue y funcionamiento en sistemas operativos iOS; además de su compatibilidad con Expo.

2.7.2. Lenguaje de programación

Dado que se utiliza React Native como marco de trabajo, el lenguaje de programación aplicado es JavaScript, de esta manera se puede realizar la

programación en un único lenguaje que puede renderizarse en aplicaciones nativas para sistemas distintos sin necesidad de reescribir la aplicación.

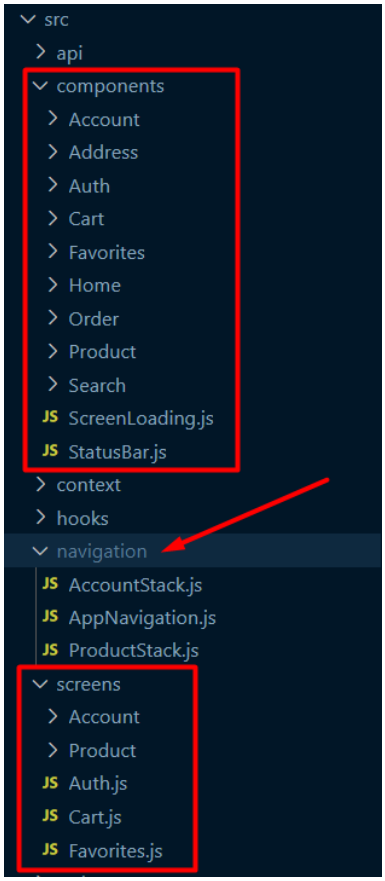
2.7.3. Sistema gestor de base de datos

Para la gestión de los datos se utiliza una base de datos no relacional, MongoDB, dado que su estructura de datos no es tan restrictiva y se puede manipular para simular un entorno relacional sin problema, además de ser compatible con Strapi, API que permite la manipulación de los datos de manera gráfica como si se tratase de un dashboard administrativo.

2.7.4. Módulos del sistema

Para una programación más ordenada de los archivos que conforman el lado del cliente se realizó la distribución entre el directorio de screens, que representan las pantallas principales, el directorio de componentes, el que contiene las funciones y formularios de cada pantalla; y el directorio de navegación, que contiene las rutas para realizar el cambio de pantalla.

Figura 24: Pantallas y componentes del prototipo



Fuente: Elaboración propia

2.7.4.1. Login y registro

Para el login y registro se utiliza la screen Auth.js y dos componentes que interactúan como el inicio de sesión (LoginForm.js) y el registro de usuario (RegisterForm.js).

Figura 25: Pantalla de login y registro

src > screens > Auth.js > Auth

```
import React, { useState } from 'react';
import { StyleSheet, View, Image, KeyboardAvoidingView, Platform } from 'react-native';
import RegisterForm from '../components/Auth/RegisterForm';
import LoginForm from '../components/Auth/LoginForm';
import logo from '../../assets/logo.png';
import { layoutStyle } from "../styles";

export default function Auth(){
    const [ showLogin, setShowLogin ] = useState(true);
    const changeForm = () => setShowLogin(!showLogin);

    return (
        <View style = {layoutStyle.container}>
            <Image style = {styles.logo} source={logo} />
            <KeyboardAvoidingView behavior={Platform.OS === "ios" ? "padding" : "height"}>
                {showLogin ? (<LoginForm changeForm={changeForm} />) : (<RegisterForm changeForm={changeForm} />)}
            </KeyboardAvoidingView>
        </View>
    );
}

const styles = StyleSheet.create({
    logo: {
        width: "100%",
        height: 50,
        resizeMode: "contain",
        marginBottom: 20,
    },
});
```

Fuente: Elaboración propia

Al momento de iniciar la aplicación se mostrará la pantalla de inicio que contiene el logo de la aplicación y carga el formulario con dos inputs, el primero admite un username o un correo y el segundo admite la contraseña. Al ingresar la información de inicio de sesión se genera un token local que mantiene la sesión iniciada, de forma que cuando el usuario accede a la aplicación aparezca directamente en el contenido de la aplicación y no tenga que solicitar credenciales nuevamente.

Figura 26: Componente formulario de login-1

src > components > Auth > JS LoginForm.js > LoginForm

```
import React, { useState } from 'react';
import { View } from 'react-native';
import { TextInput, Button } from 'react-native-paper';
import { useFormik } from 'formik';
import * as Yup from 'yup';
import Toast from 'react-native-root-toast';
import useAuth from '../../hooks/useAuth';
import { loginApi } from '../../api/user'
import { formStyles } from "../../styles";

export default function LoginForm(props) {
    const {changeForm} = props;
    const [loading, setLoading] = useState(false);
    const { login } = useAuth();

    const formik = useFormik({
        initialValues: initialValues(),
        validationSchema: Yup.object(validationSchema()),
        onSubmit: async (formData) => {
            setLoading(true);
            try {
                const response = await loginApi(formData);
                if (response.statusCode) throw "Error en el usuario o contraseña";
                login(response)
            } catch (error) {
                Toast.show(error, {
                    position: Toast.positions.CENTER,
                });
                setLoading(false);
            }
        },
    });

```

Fuente: Elaboración propia

Figura 27: Componente formulario de login-2

```
    return (
        <View>
            <TextInput
                label = "Email o Username"
                style = {formStyles.input}
                onChangeText = {(text) => formik.setFieldValue("identifier", text)}
                value = {formik.values.identifier}
                error = {formik.errors.identifier} />
            <TextInput
                label = "Contraseña"
                style = {formStyles.input} secureTextEntry
                onChangeText = {(text) => formik.setFieldValue("password", text)}
                value = {formik.values.password}
                error = {formik.errors.password} />
            <Button
                mode = "contained"
                style = {formStyles.btnSuccess}
                onPress = {formik.handleSubmit}
                loading = {loading}
                >Iniciar sesión</Button>
            <Button
                mode = "text"
                style = {formStyles.btnText}
                labelStyle = {formStyles.btnTextLabel}
                onPress = {changeForm}>Registrarse</Button>
        </View>
    )
}

function initialValues() {
    return {
        identifier: "",
        password: ""
    };
}

function validationSchema() {
    return {
        identifier: Yup.string().required(true),
        password: Yup.string().required(true)
    };
}
```

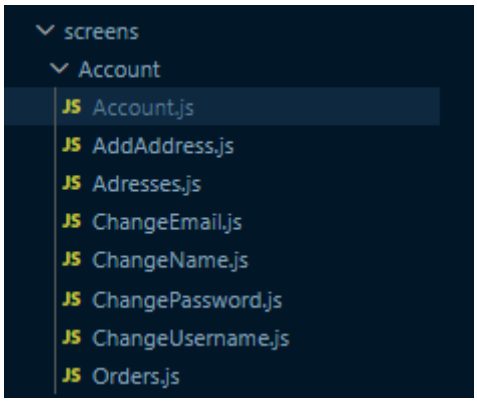
Fuente: Elaboración propia

Para el registro del usuario se carga la misma pantalla, pero con un formulario diferente, el cual contiene los campos para realizar un registro rápido, solicitando únicamente un correo, nombre de usuario y contraseña; una vez se registra un usuario se cambia el formulario al de inicio de sesión para que pueda acceder a la aplicación. La aplicación interna se divide en 4 pantallas principales (Productos, Favoritos, Carrito e información de la cuenta).

2.7.4.2. Información de la cuenta

La información de la cuenta se carga en la screen principal Account.js la cual a su vez carga las pantallas que interactúan con la información del usuario.

Figura 28: Screens utilizadas para mostrar la información de la cuenta



Fuente: Elaboración propia

Figura 29: Pantalla principal de la información de la cuenta

src > screens > Account > Account.js > Account

```
import React, { useState, useCallback } from 'react';
import { ScrollView } from 'react-native';
import { useFocusEffect } from '@react-navigation/native';
import StatusBar from '../../components/StatusBar';
import Search from '../../components/Search';
import ScreenLoading from '../../components/ScreenLoading';
import UserInfo from '../../components/Account/UserInfo';
import Menu from '../../components/Account/Menu';
import { getMeApi } from '../../api/user';
import useAuth from '../../hooks/useAuth';
import colors from '../../styles/colors';

export default function Account() {
    const [ user, setUser ] = useState(null);
    const { auth } = useAuth();

    useFocusEffect(
        useCallback(() => {
            ( async () => {
                const response = await getMeApi(auth.token);
                setUser(response);
            })()
            }, [])
    );

    return (
        <>
            <StatusBar backgroundColor = {colors.bgDark} barStyle = "light-content" />
            {!user ? (
                <ScreenLoading size = "large" />
            ) : (
                <>
                    <Search />
                    <ScrollView>
                        <UserInfo user = {user} />
                        <Menu />
                    </ScrollView>
                </>
            )}
        </>
    );
}
```

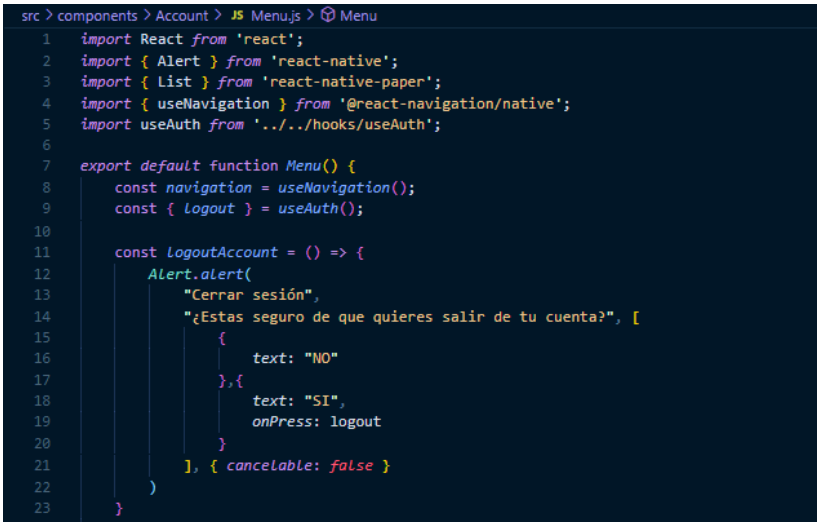
Fuente: Elaboración propia

Al iniciar sesión la información obtiene los datos del usuario a través del componente UserInfo.js y le permite modificar dichos parámetros; cuando el usuario es nuevo aparecerá su correo en el apartado de su nombre, lo cual puede corregirse al agregar un nombre en las opciones de la cuenta. El resto de opciones permite modificar el correo, nombre de usuario y contraseña; al momento de entrar en esas opciones se carga la información previamente proporcionada gracias al componente Menu.js.

Figura 30: Componente de la información del usuario

Fuente: Elaboración propia

Figura 31: Componente para el menú de opciones de la cuenta-1



```
src > components > Account > JS Menu.js > Menu
import React from 'react';
import { Alert } from 'react-native';
import { List } from 'react-native-paper';
import { useNavigation } from '@react-navigation/native';
import useAuth from '../../hooks/useAuth';

export default function Menu() {
    const navigation = useNavigation();
    const { logout } = useAuth();

    const logoutAccount = () => {
        Alert.alert(
            "Cerrar sesión",
            "¿Estas seguro de que quieres salir de tu cuenta?", [
                {
                    text: "NO"
                },{
                    text: "SI",
                    onPress: logout
                }
            ], { cancelable: false }
        )
    }
```

Fuente: Elaboración propia

Figura 32: Componente para el menú de opciones de la cuenta-2

```
src > components > Account > JS Menu.js > Menu

    return (
        <>
            <List.Section>
                <List.Subheader>Mi cuenta</List.Subheader>
                <List.Item
                    title = "Cambiar nombre"
                    description = "Cambia el nombre de tu cuenta."
                    left = {(props) => <List.Icon {...props} icon = "face" />}
                    onPress = {() => navigation.navigate("change-name")} />
                <List.Item
                    title = "Cambiar email"
                    description = "Cambia el email asociado a tu cuenta."
                    left = {(props) => <List.Icon {...props} icon = "at" />}
                    onPress = {() => navigation.navigate("change-email")} />
                <List.Item
                    title = "Cambiar username"
                    description = "Cambia el nombre de usuario de tu cuenta."
                    left = {(props) => <List.Icon {...props} icon = "sim" />}
                    onPress = {() => navigation.navigate("change-username")} />
                <List.Item
                    title = "Cambiar contraseña"
                    description = "Cambia la contraseña asociada a tu cuenta."
                    left = {(props) => <List.Icon {...props} icon = "key" />}
                    onPress = {() => navigation.navigate("change-password")} />
                <List.Item
                    title = "Mis direcciones"
                    description = "Administra tus direcciones de envío."
                    left = {(props) => <List.Icon {...props} icon = "map" />}
                    onPress = {() => navigation.navigate("adresses")} />
            </List.Section>
            <List.Section>
                <List.Subheader>App</List.Subheader>
                <List.Item
                    title = "Pedidos"
                    description = "Estado de todos los pedidos."
                    left = {(props) => <List.Icon {...props} icon = "clipboard-list" />}
                    onPress = {() => navigation.navigate("orders")} />
                <List.Item
                    title = "Lista de deseos"
                    description = "Listado de productos que quieres comprar."
                    left = {(props) => <List.Icon {...props} icon = "heart" />}
                    onPress = {() => navigation.navigate("favorites")} />
                <List.Item
                    title = "Cerrar sesión"
                    description = "Cierra esta sesión e incia con otra."
                    left = {(props) => <List.Icon {...props} icon = "logout" />}
                    onPress = {logoutAccount} />
            </List.Section>
        </>
    );
}
```

Fuente: Elaboración propia

Otra opción sobre la información del usuario es el permitirle agregar, modificar o eliminar direcciones, el cual es un formulario que solicita información como código postal, números de teléfono, dirección, etc. Las cuales servirán para el momento de realizar una compra. El componente menú permite llamar a las diferentes screens que Realizan dichas operaciones.

La screen ChangeName.js carga la información del nombre del cliente (en caso de haber creado la cuenta por primera vez aparecerá vacío). En esta screen se podrá cambiar la información del nombre y apellidos con campos validados por una expresión regular que admite únicamente caracteres de letra y espacios.

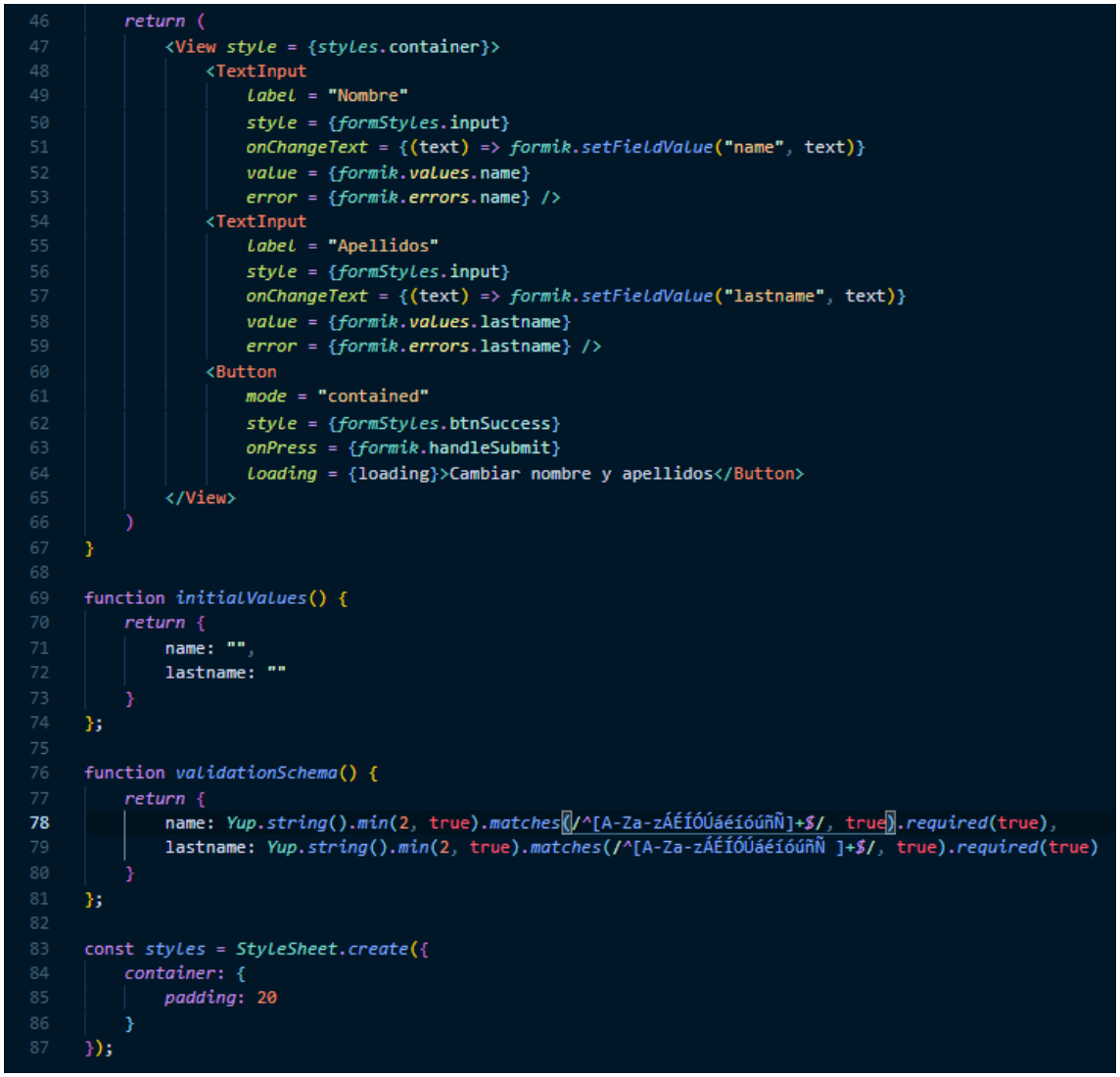
Figura 33: Screen para modificar el nombre del usuario

src > screens > Account > JS ChangeName.js > validationSchema > name

```
1   import React, { useState, useCallback } from 'react';
2   import { StyleSheet, View } from 'react-native';
3   import { TextInput, Button } from 'react-native-paper';
4   import { useFocusEffect, useNavigation } from '@react-navigation/native';
5   import { useFormik } from 'formik';
6   import * as Yup from 'yup';
7   import Toast from 'react-native-root-toast';
8   import { getMeApi, updateUserApi } from '../../api/user';
9   import useAuth from '../../hooks/useAuth';
10  import { formStyles } from '../../styles';
11
12  export default function ChangeName() {
13      const [ loading, setLoading] = useState(false);
14      const { auth } = useAuth();
15      const navigation = useNavigation();
16
17      useFocusEffect(
18          useCallback(() => {
19              (async () => {
20                  const response = await getMeApi(auth.token);
21                  if (response.name && response.lastname) {
22                      await formik.setFieldValue("name", response.name);
23                      await formik.setFieldValue("lastname", response.lastname);
24                  }
25              })();
26          },[])
27      );
28
29      const formik = useFormik({
30          initialValues: initialValues(),
31          validationSchema: Yup.object(validationSchema()),
32          onSubmit: async(formData) => {
33              setLoading(true);
34              try {
35                  await updateUserApi(auth, formData);
36                  navigation.goBack();
37              } catch (error) {
38                  Toast.show("Error al actualizar los datos.",{
39                      position: Toast.positions.CENTER
40                  });
41                  setLoading(false);
42              }
43          }
44      });
45
```

Fuente: Elaboración propia

Figura 34: Screen para modificar el nombre del usuario



```
    return (
        <View style = {styles.container}>
            <TextInput
                label = "Nombre"
                style = {formStyles.input}
                onChangeText = {(text) => formik.setFieldValue("name", text)}
                value = {formik.values.name}
                error = {formik.errors.name} />
            <TextInput
                label = "Apellidos"
                style = {formStyles.input}
                onChangeText = {(text) => formik.setFieldValue("lastname", text)}
                value = {formik.values.lastname}
                error = {formik.errors.lastname} />
            <Button
                mode = "contained"
                style = {formStyles.btnSuccess}
                onPress = {formik.handleSubmit}
                loading = {loading}>Cambiar nombre y apellidos</Button>
        </View>
    )
}

function initialValues() {
    return {
        name: "",
        lastname: ""
    }
};

function validationSchema() {
    return {
        name: Yup.string().min(2, true).matches(/^[A-Za-zÁÉÍÓÚáéíóúñÑ]+$/, true).required(true),
        lastname: Yup.string().min(2, true).matches(/^[A-Za-zÁÉÍÓÚáéíóúñÑ ]+$/, true).required(true)
    }
};

const styles = StyleSheet.create({
    container: {
        padding: 20
    }
});
```

Fuente: Elaboración propia

La screen ChangeEmail.js carga un formulario para cambiar el correo asociado a la cuenta, cargando el correo actual del usuario. Para un nuevo correo se requiere que este tenga un formato válido para que se permita el ingreso, esta validación se da gracias a la librería Yup de manera automática.

Figura 35: Screen para modificar el correo del usuario-1

src > screens > Account > JS ChangeEmail.js > ChangeEmail

```
import React, { useState, useCallback } from 'react';
import { View, StyleSheet } from 'react-native';
import { TextInput, Button } from 'react-native-paper';
import { useFocusEffect, useNavigation } from '@react-navigation/native';
import { useFormik } from 'formik';
import * as Yup from 'yup';
import Toast from 'react-native-root-toast';
import { getMeApi, updateUserApi } from '../../api/user';
import useAuth from '../../hooks/useAuth';
import { formStyles } from '../../styles'

export default function ChangeEmail() {
    const { auth } = useAuth();
    const [loading, setLoading] = useState(false);
    const navigation = useNavigation();

    useFocusEffect(
        useCallback(() => {
            (async () => {
                const response = await getMeApi(auth.token);
                await formik.setFieldValue("email", response.email);
            })();
        }, [])
    );

    const formik = useFormik({
        initialValues: initialValues(),
        validationSchema: Yup.object(validationSchema()),
        onSubmit: async (formData) => {
            setLoading(true);
            try {
                const response = await updateUserApi(auth, formData);
                if (response.statusCode) throw "El email ya existe";
                navigation.goBack();
            } catch (error) {
                Toast.show(error, {
                    position: Toast.positions.CENTER
                });
                formik.setFieldError("email", true);
                setLoading(false);
            }
        }
    });

```

Fuente: Elaboración propia

Figura 36: Screen para modificar el correo del usuario-2

```
45      return (
46          <View style = {styles.container}>
47              <TextInput
48                  label = "Email"
49                  style = {formStyles.input}
50                  onChangeText = {(text) => formik.setFieldValue("email", text)}
51                  value = {formik.values.email}
52                  error = {formik.errors.email} />
53              <Button
54                  mode = "contained"
55                  style = {formStyles.btnSuccess}
56                  onPress = {formik.handleSubmit}
57                  loading = {loading} >Cambiar email</Button>
58          </View>
59      )
60  }
61
62  function initialValues() {
63      return {
64          email: ""
65      };
66  }
67
68  function validationSchema() {
69      return {
70          email: Yup.string().email(true).required(true)
71      };
72  }
73
74  const styles = StyleSheet.create({
75      container: {
76          padding: 20
77      }
78  });
```

Fuente: Elaboración propia

La screen ChangePassword.js carga un formulario para cambiar la contraseña asociada. Para que la nueva contraseña sea válida sebe tener al menos 4 caracteres.

Figura 37: Screen para modificar la contraseña del usuario-1

src > screens > Account > JS ChangePassword.js > validationSchema

```
import React, { useState } from 'react';
import { View, StyleSheet } from 'react-native';
import { TextInput, Button } from 'react-native-paper';
import { useNavigation } from '@react-navigation/native';
import { useFormik } from 'formik';
import * as Yup from 'yup';
import Toast from 'react-native-root-toast';
import { updateUserApi } from '../../api/user';
import useAuth from '../../hooks/useAuth';
import { formStyles } from '../../styles'

export default function ChangePassword() {
    const { auth } = useAuth();
    const [loading, setLoading] = useState(false);
    const navigation = useNavigation();

    const formik = useFormik({
        initialValues: initialValues(),
        validationSchema: Yup.object(validationSchema()),
        onSubmit: async (formData) => {
            setLoading(true);
            try {
                const response = await updateUserApi(auth, formData);
                if (response.statusCode) throw "Error al cambiar la contraseña";
                navigation.goBack();
            } catch (error) {
                Toast.show(error, {
                    position: Toast.positions.CENTER
                });
                setLoading(false);
            }
        }
    });

```

Fuente: Elaboración propia

Figura 38: Screen para modificar el correo del usuario-2

Fuente: Elaboración propia

La screen ChangeUsername.js carga un formulario para cambiar el nombre de usuario asociado a la cuenta y que sirve para el inicio de sesión. Para que se permita el cambio, el nuevo username debe respetar la expresión regular de su validación, sin espacios ni caracteres especiales.

Figura 39: Screen para modificar el username del cliente-1

src > screens > Account > JS ChangeUsername.js > styles > container

```
import React, { useState, useCallback } from 'react';
import { View, StyleSheet } from 'react-native';
import { TextInput, Button } from 'react-native-paper';
import { useFocusEffect, useNavigation } from '@react-navigation/native';
import { useFormik } from 'formik';
import * as Yup from 'yup';
import Toast from 'react-native-root-toast';
import { getMeApi, updateUserApi } from '../../api/user';
import useAuth from '../../hooks/useAuth';
import { formStyles } from '../../styles'

export default function ChangeUsername() {
    const { auth } = useAuth();
    const [loading, setLoading] = useState(false);
    const navigation = useNavigation();

    useFocusEffect(
        useCallback(() => {
            (async () => {
                const response = await getMeApi(auth.token);
                await formik.setFieldValue("username", response.username);
            })();
        }, [])
    );

    const formik = useFormik({
        initialValues: initialValues(),
        validationSchema: Yup.object(validationSchema()),
        onSubmit: async (formData) => {
            setLoading(true);
            try {
                const response = await updateUserApi(auth, formData);
                if (response.statusCode) throw "El nombre de usuario ya existe";
                navigation.goBack();
            } catch (error) {
                Toast.show(error, {
                    position: Toast.positions.CENTER
                });
                formik.setFieldError("username", true);
                setLoading(false);
            }
        }
    });

```

Fuente: Elaboración propia

Figura 40: Screen para modificar el username del cliente-2

Fuente: Elaboración propia

Entre las opciones de la aplicación que se pueden ver dentro de esta pantalla está una opción para ver los pedidos ejecutados, una lista de productos agregados en favoritos y un cierre de sesión, al seleccionarlo mostrará un mensaje de alerta en caso de que se seleccionase por error, una vez se confirme que se desea cerrar sesión se eliminará el token del inicio de sesión y se mostrará la pantalla de inicio de sesión.

La screen Orders.js es la que permite ver los pedidos realizados desde la aplicación.

Figura 41: Screen para ver las órdenes emitidas

src > screens > Account > JS Orders.js > styles > container > padding

```
import React, { useState, useEffect, useCallback } from 'react';
import { StyleSheet, ScrollView, Text, ActivityIndicator } from 'react-native';
import { useFocusEffect } from '@react-navigation/native';
import { size } from 'lodash';
import StatusBar from '../../components/StatusBar';
import ListOrder from '../../components/Order/ListOrder';
import useAuth from '../../hooks/useAuth';
import { getOrderApi } from '../../api/orders';
import colors from '../../styles/colors'

export default function Orders() {
    const [orders, setOrders] = useState(null);
    const { auth } = useAuth();

    useFocusEffect(
        useCallback(() => {
            (async() => {
                const response = await getOrderApi(auth);
                setOrders(response);
            })()
        }, [])
    );

    return (
        <>
            <StatusBar />
            <ScrollView style = {styles.container}>
                <Text style = {styles.title}>Mis pedidos</Text>
                {!orders ? (
                    <ActivityIndicator size = "large" style = {styles.loading} />
                ) : size(orders) === 0 ? (
                    <Text style = {styles.noOrderText}>Mis pedidos</Text>
                ) : (
                    <ListOrder orders = {orders} />
                )}
            </ScrollView>
        </>
    );
}

```

Fuente: Elaboración propia

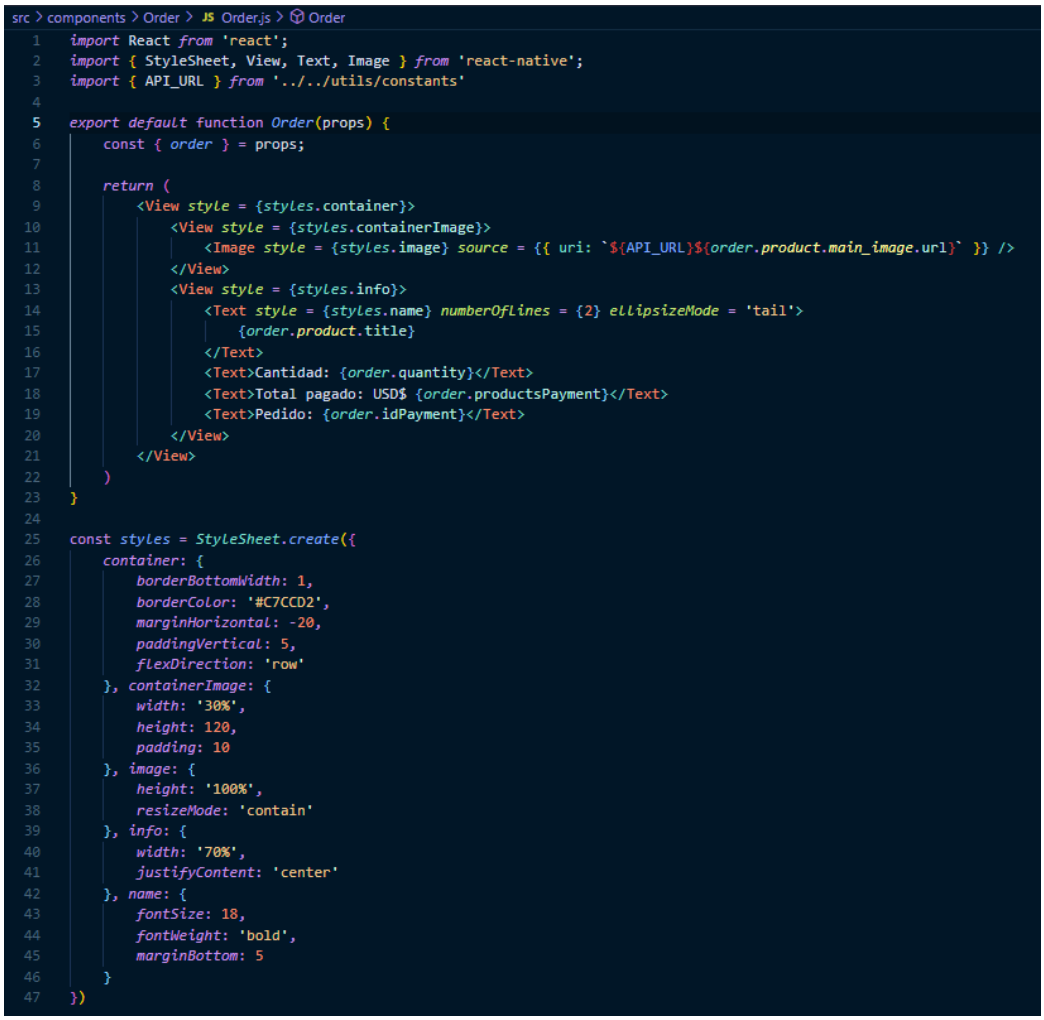
Dicha screen utiliza dos componentes, ListOrder.js que carga una lista de los productos comprados en el carrito; y el componente Order.js que carga la información de cada producto de los pedidos (imagen, nombre, cantidad, precio e id del pedido).

Figura 42: Componente para cargar la lista de órdenes emitidas

```
src > components > Order > JS ListOrder.js > ListOrder > map() callback
1   import React from 'react';
2   import { StyleSheet, View, Text } from 'react-native';
3   import Order from './Order';
4   import { map } from 'lodash';
5
6   export default function ListOrder(props) {
7       const { orders } = props;
8
9       return (
10          <View style = {styles.container}>
11              {map(orders, (order) => (
12                  <Order key = {order._id} order = {order} />
13              ))}
14          </View>
15      )
16  }
17
18  const styles = StyleSheet.create({
19      container: {
20          marginTop: 20,
21          marginBottom: 40
22      }
23  })
```

Fuente: Elaboración propia

Figura 43: Componente para ver los productos de las órdenes



```
src > components > Order > JS Order.js > Order
1   import React from 'react';
2   import { StyleSheet, View, Text, Image } from 'react-native';
3   import { API_URL } from '../../utils/constants'
4
5   export default function Order(props) {
6       const { order } = props;
7
8       return (
9           <View style = {styles.container}>
10              <View style = {styles.containerImage}>
11                  <Image style = {styles.image} source = {{ uri: `${API_URL}${order.product.main_image.url}` }} />
12              </View>
13              <View style = {styles.info}>
14                  <Text style = {styles.name} numberOflines = {2} ellipsizeMode = 'tail'>
15                      {order.product.title}
16                  </Text>
17                  <Text>Cantidad: {order.quantity}</Text>
18                  <Text>Total pagado: USD$ {order.productsPayment}</Text>
19                  <Text>Pedido: {order.idPayment}</Text>
20              </View>
21          </View>
22      )
23  }
24
25  const styles = StyleSheet.create({
26      container: {
27          borderBottomWidth: 1,
28          borderColor: '#C7CCD2',
29          marginHorizontal: -20,
30          paddingVertical: 5,
31          flexDirection: 'row'
32      }, containerImage: {
33          width: '30%',
34          height: 120,
35          padding: 10
36      }, image: {
37          height: '100%',
38          resizeMode: 'contain'
39      }, info: {
40          width: '70%',
41          justifyContent: 'center'
42      }, name: {
43          fontSize: 18,
44          fontWeight: 'bold',
45          marginBottom: 5
46      }
47  })
```

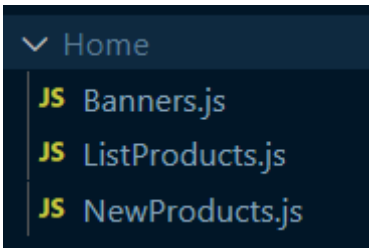
Fuente: Elaboración propia

2.7.4.3. Productos

La sección cuenta con 3 pantallas: la pantalla de home que contiene todos los productos en venta, una pantalla que carga la información individual y una pantalla que carga los productos que coincidan con una búsqueda específica.

La pantalla de “Home” utiliza 3 componentes: un Banner que carga imágenes de productos promocionales o nuevos y un componente que carga los productos agregados según se hayan agregado a la base de datos; este último utiliza el tercer componente, el cual genera botones con la imagen y el nombre de cada producto.

Figura 44: Screens para el manejo de productos



Fuente: Elaboración propia

Figura 45: Screen principal de la pestaña Home

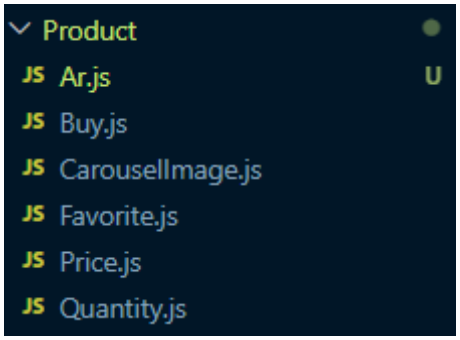
src > screens > Product > Home.js > Home

```
import React from 'react';
import { ScrollView, Text, StyleSheet } from 'react-native';
import StatusBar from '../../components/StatusBar';
import Search from '../../components/Search'
import NewProducts from '../../components/Home/NewProducts';
import Banners from '../../components/Home/Banners';
import colors from '../../styles/colors';

export default function Home() {
    return (
        <>
            <StatusBar backgroundColor = {colors.bgDark} barStyle = "light-content" />
            <Search />
            <ScrollView>
                <Text style = {styles.title}>Promociones</Text>
                <Banners />
                <NewProducts />
            </ScrollView>
        </>
    );
}

const styles = StyleSheet.create({
    title: {
        padding: 10,
        fontWeight: "bold",
        fontSize: 20,
        marginBottom: 5
    }
});
```

Fuente: Elaboración propia

La pantalla “Product” utiliza 5 componentes. Dentro de esta screen se aprecia la información del producto como la imagen principal y el nombre; uno de los componentes permite calcular el precio de la aplicación, dado que algunos productos pueden tener descuento; también presenta un componente que carga una selección para la cantidad de productos a comprar.

Figura 46: Componentes cargados en la pantalla de productos



Fuente: Elaboración propia

El componente Quantity carga una lista estática de valor para seleccionar la cantidad que se desea adquirir de un producto.

Figura 47: Componente para cargar la cantidad de productos

src > components > Product > JS Quantity.js > Quantity

```
import React from 'react';
import { StyleSheet } from 'react-native';
import DropDownPicker from 'react-native-dropdown-picker';

export default function Quantity(props) {
    const {quantity, setQuantity} = props;

    return (
        <DropDownPicker
            items = {[
                {
                    label: "1",
                    value: 1
                }, {
                    label: "2",
                    value: 2
                }, {
                    label: "3",
                    value: 3
                }
            ]}
            defaultValue = {quantity}
            containerStyle = {styles.containerStyle}
            itemStyle = {styles.itemStyle}
            DropDownStyle = {styles.DropDownPicker}
            style  = {styles.DropDownPicker}
            labelStyle = {styles.labelStyle}
            onChangeItem = {(item) => setQuantity(item.value)} />
    );
}
```

Fuente: Elaboración propia

Debajo de esta información se podrá interactuar con 3 botones, uno de ellos (Favorite.js) permite agregar el producto a la lista de favoritos, una vez hecho el texto y función del botón cambiarán para poder eliminar al producto de la lista si se desea.

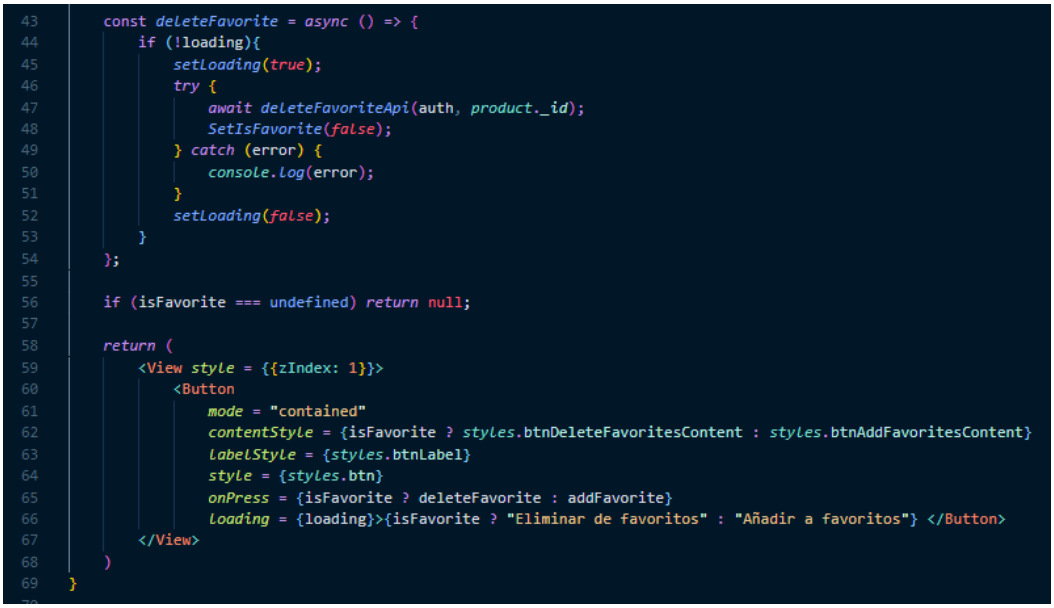
Figura 48: Componente para añadir productos a lista de favoritos

src > components > Product > JS Favorite.js > Favorite

```
import React, { useState, useEffect } from 'react';
import { StyleSheet, View } from 'react-native';
import { Button } from 'react-native-paper';
import Toast from 'react-native-root-toast';
import { size } from 'lodash';
import useAuth from '../../hooks/useAuth';
import { isFavoriteApi, addFavoriteApi, deleteFavoriteApi } from '../../api/favorite';
import colors from '../../styles/colors';

export default function Favorite(props) {
    const { product } = props;
    const [isFavorite, SetIsFavorite] = useState(undefined);
    const [loading, setLoading] = useState(false);
    const { auth } = useAuth();

    useEffect(() => {
        (async () => {
            const response = await isFavoriteApi(auth, product._id)
            if (size(response) === 0) SetIsFavorite(false);
            else SetIsFavorite(true);
        })()
    }, [product])

    const addFavorite = async () => {
        if (!loading){
            setLoading(true);
            try {
                await addFavoriteApi(auth, product._id);
                SetIsFavorite(true);
                Toast.show("Producto añadido a favoritos", {
                    position: Toast.positions.CENTER
                })
            } catch (error) {
                console.log(error);
                Toast.show("ERROR al añadir producto a favoritos", {
                    position: Toast.positions.CENTER
                })
            }
            setLoading(false);
        }
    };
```

Fuente: Elaboración propia

Figura 49: Componente para eliminar productos lista de favoritos



```
    const deleteFavorite = async () => {
        if (!loading){
            setLoading(true);
            try {
                await deleteFavoriteApi(auth, product._id);
                SetIsFavorite(false);
            } catch (error) {
                console.log(error);
            }
            setLoading(false);
        }
    };

    if (isFavorite === undefined) return null;

    return (
        <View style = {{zIndex: 1}}>
            <Button
                mode = "contained"
                contentStyle = {isFavorite ? styles.btnDeleteFavoritesContent : styles.btnAddFavoritesContent}
                labelStyle = {styles.btnLabel}
                style = {styles.btn}
                onPress = {isFavorite ? deleteFavorite : addFavorite}
                loading = {loading}>{isFavorite ? "Eliminar de favoritos" : "Añadir a favoritos"} </Button>
        </View>
    )
}
```

Fuente: Elaboración propia

El segundo botón permitirá agregar el producto al carrito de comprar (Buy.js).

Figura 50: Componente para añadir productos al carrito

```
src > components > Product > JS Buy.js > styles > btn > marginTop
import React from 'react';
import { StyleSheet, View } from 'react-native';
import { Button } from 'react-native-paper';
import Toast from 'react-native-root-toast';
import { addProductCartApi } from '../../api/cart';
import colors from '../../styles/colors';

export default function Buy(props) {
    const { product, quantity } = props;

    const addProductCart = async () => {
        const response = await addProductCartApi(product._id, quantity);

        if(response) {
            Toast.show("Producto añadido al carrito", {
                position: Toast.positions.CENTER
            })
        } else {
            Toast.show("ERROR al añadir el producto al carrito", {
                position: Toast.positions.CENTER
            })
        }
    };

    return (
        <View style = {{zIndex: 1}}>
            <Button
                mode = "contained"
                contentStyle = {styles.btnBuyContent}
                labelStyle = {styles.btnLabel}
                style = {styles.btn}
                onPress = {addProductCart}>Añadir al carrito</Button>
        </View>

    )
}
```

Fuente: Elaboración propia

Un tercer botón abrirá un visualizador para ver el producto como un modelo 3D de realidad aumentada (AR.js); este componente carga un enlace desde la base de datos para visualizarlo en un visor externo.

Figura 51: Componente para cargar el visor AR

src > components > Product > JS Ar.js > Buy

```
1   import React from 'react';
2   import { StyleSheet, View } from 'react-native';
3   import { Button } from 'react-native-paper';
4   import * as WebBrowser from "expo-web-browser";
5
6   export default function Buy(props) {
7       const { product } = props;
8
9       _handleOpenWithWebBrowser = () => {
10          console.log(product.Link)
11          WebBrowser.openBrowserAsync(`${product.Link}`);
12        };
13
14      return (
15          <View style = {{zIndex: 1}}>
16              <Button
17                  mode = "contained"
18                  contentStyle = {styles.btnBuyContent}
19                  labelStyle = {styles.btnLabel}
20                  style = {styles.btn}
21                  onPress = {this._handleOpenWithWebBrowser}>Vista AR</Button>
22          </View>
23
24      )
25  }
```

Fuente: Elaboración propia

Por último, se carga un carrusel de imágenes con imágenes promocionales o fichas técnicas del producto a través del componente CarouselImage.js.

Figura 52: Componente para el carrusel de imágenes

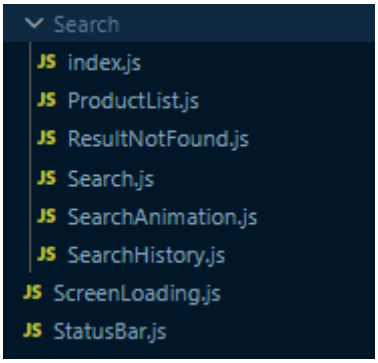
src > components > Product > JS CarouselImage.js > CarouselImage

```
1   import React, { useState } from 'react';
2   import { StyleSheet, Image, Dimensions } from 'react-native';
3   import { API_URL } from '../../utils/constants';
4   import Carousel, { Pagination } from 'react-native-snap-carousel';
5   import { size } from 'lodash';
6
7   const width = Dimensions.get("window").width;
8   const height = 400;
9
10  export default function CarouselImage(props) {
11      const { images } = props;
12      const [imageActive, setImageActive] = useState(0)
13
14      const renderItem = ({ item }) => {
15          return <Image style = {styles.carousel} source = {{uri: `${API_URL}${item.url}`}} />
16      }
17
18      return (
19          <>
20              <Carousel
21                  layout = {"default"}
22                  data = {images}
23                  sliderWidth = {width}
24                  itemWidth = {width}
25                  renderItem = {renderItem}
26                  onSnapToItem = {(index) => setImageActive(index)} />
27              <Pagination
28                  dotsLength = {size(images)}
29                  activeDotIndex = {imageActive}
30                  inactiveDotOpacity = {0.4}
31                  inactiveDotScale = {0.6} />
32          </>
33      )
34  }
35
36  const styles = StyleSheet.create({
37      carousel: {
38          width,
39          height,
40          resizeMode: "contain"
41      }
42  })
```

Fuente: Elaboración propia

En la parte superior de la pantalla se encuentra una barra de búsqueda en la cual se podrá ingresar un término de búsqueda; al realizar la búsqueda cargar una pantalla que muestra todos los productos que coincidan con la búsqueda en caso de haber coincidencia, caso contrario cargará una pantalla indicando la falta de resultados; después de realizar una o varias búsquedas se mostrará una pantalla que contiene los términos previamente buscados a manera de historial.

Figura 53: Componentes para la pantalla Search



Fuente: Elaboración propia

Figura 54: Pantalla de la sección de búsquedas

src > screens > Product > Search.js > SearchScreen

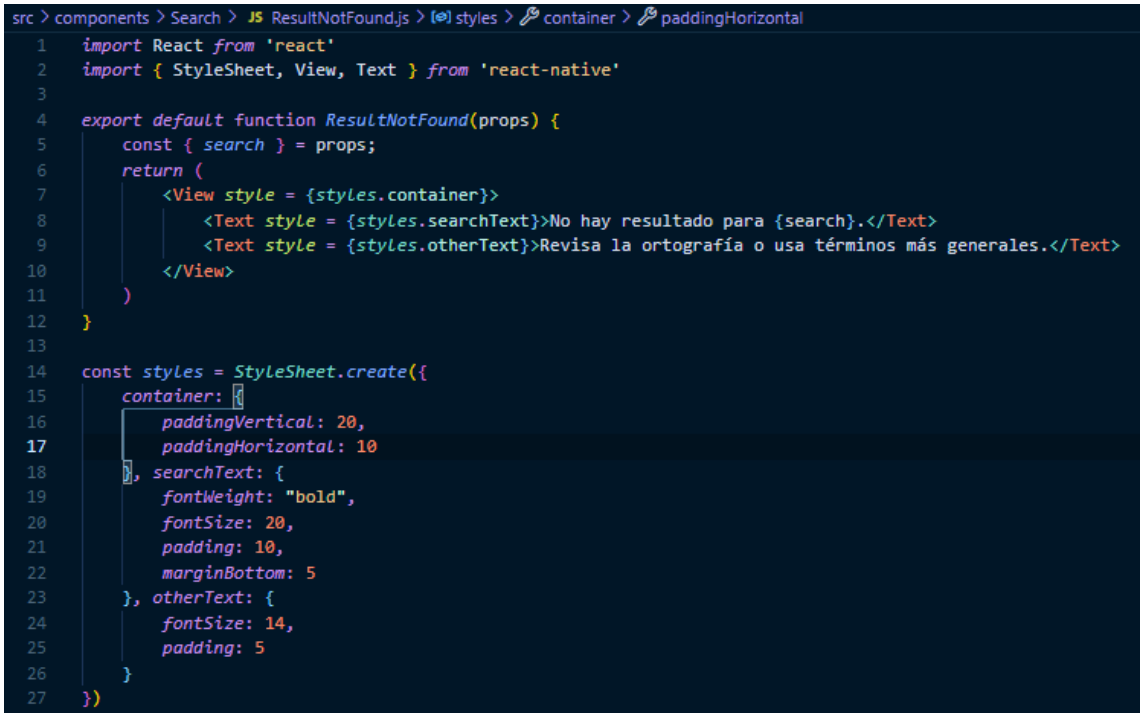
```
1   import React, { useState, useEffect } from 'react';
2   import { size } from 'lodash';
3   import StatusBar from '../../components/StatusBar';
4   import Search from '../../components/Search/Search';
5   import ScreenLoading from '../../components/ScreenLoading';
6   import ResultNotFound from '../../components/Search/ResultNotFound';
7   import ProductList from '../../components/Search/ProductList';
8   import { searchProductsApi } from '../../api/search';
9   import colors from '../../styles/colors';
10
11  export default function SearchScreen(props) {
12      const { route } = props;
13      const { params } = route;
14      const [products, setProducts] = useState(null)
15
16      useEffect(() => {
17          (async () => {
18              setProducts(null);
19              const response = await searchProductsApi(params.search);
20              setProducts(response);
21          })()
22      }, [params.search]);
23
24      return (
25          <>
26              <StatusBar backgroundColor = {colors.bgDark} barStyle = 'light-content' />
27              <Search currentSearch = {params.search} />
28              {!products ? (
29                  <ScreenLoading text = "Buscando productos"/>
30              ) : size(products) === 0 ? (
31                  <ResultNotFound search = {params.search} />
32              ) : (
33                  <ProductList products = {products} />
34              )}
35          </>
36      )
37  }
```

Fuente: Elaboración propia

Los componentes principales de la pantalla de búsqueda son el ResultNotFound.js y el ProductList.js.

En el componente ResultNotFound.js se carga una pantalla que indica que no se encontraron coincidencias para la búsqueda realizada.

Figura 55: Componente para búsquedas sin resultados



Fuente: Elaboración propia

El Componente ProductList.js carga todos los productos que coincidan con los términos de búsqueda implementados; mostrando la imagen principal, nombre y el precio y descuento en caso de tener alguno.

Figura 56: Componente para cargar lista de productos encontrados

src > components > Search > JS ProductList.js > styles > container > paddingHorizontal

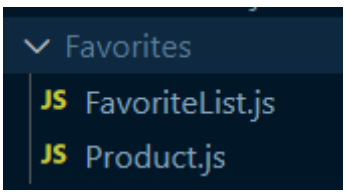
```
import React from 'react';
import { StyleSheet, View, ScrollView, Text, TouchableWithoutFeedback, Image } from 'react-native';
import { Button } from 'react-native-paper';
import { map } from 'lodash';
import { API_URL } from '../../utils/constants';
import { useNavigation } from '@react-navigation/native';
import colors from '../../styles/colors';

export default function ProductList(props) {
    const { products } = props;
    const navigation = useNavigation();

    const calcPrice = (price, discount) => {
        if (!discount) return price;

        const discountAmount = (price * discount)/100;
        return (price - discountAmount).toFixed(2);
    }

    const goToProduct = (id) => {
        navigation.push('product', {idProduct: id})
    }

    return (
        <ScrollView contentContainerStyle = {styles.container}>
            <Text style = {styles.title}>Resultados</Text>
            {map(products, (product) => (
                <TouchableWithoutFeedback key = {product._id} onPress = {() => goToProduct(product._id)}>
                    <View style = {styles.product}>
                        <View style = {styles.containerImage}>
                            <Image style = {styles.image} source ={{uri: `${API_URL}${product.main_image.url}`}} />
                        </View>
                        <View style = {styles.info}>
                            <Text style = {styles.name} numberOfLines = {3} ellipsizeMode = 'tail'>{product.title}</Text>
                            <View style = {styles.price}>
                                <Text style = {styles.currentPrice}>US$ {calcPrice(product.price, product.discount)}</Text>
                                {!!product.discount && (
                                    <Text style = {styles.oldPrice}>US$ {product.price}</Text>
                                )}
                            </View>
                            <Button style = {styles.btn} color = {colors.primary}>Ir al producto</Button>
                        </View>
                    </View>
                </TouchableWithoutFeedback>
            ))}
        </ScrollView>
    )
}
```

Fuente: Elaboración propia

2.7.4.4. Favoritos

En la pantalla de favoritos se carga una lista de los productos que hayan sido seleccionados como favoritos desde las pantallas individuales de cada producto; en cada pantalla se podrá apreciar el nombre e imagen de cada producto y dos botones, uno para visualizar el producto y uno para eliminar al producto de la lista de favoritos.

Figura 57: Componentes para la sección de favoritos



Fuente: Elaboración propia

La screen Favorites.js carga un spinner cuando está activa y lanza una pantalla que indica la ausencia de favoritos en caso de no tener ninguno en la base de datos, o cargar el componente FavoriteList.js.

Figura 58: Screen principal de vista de favoritos

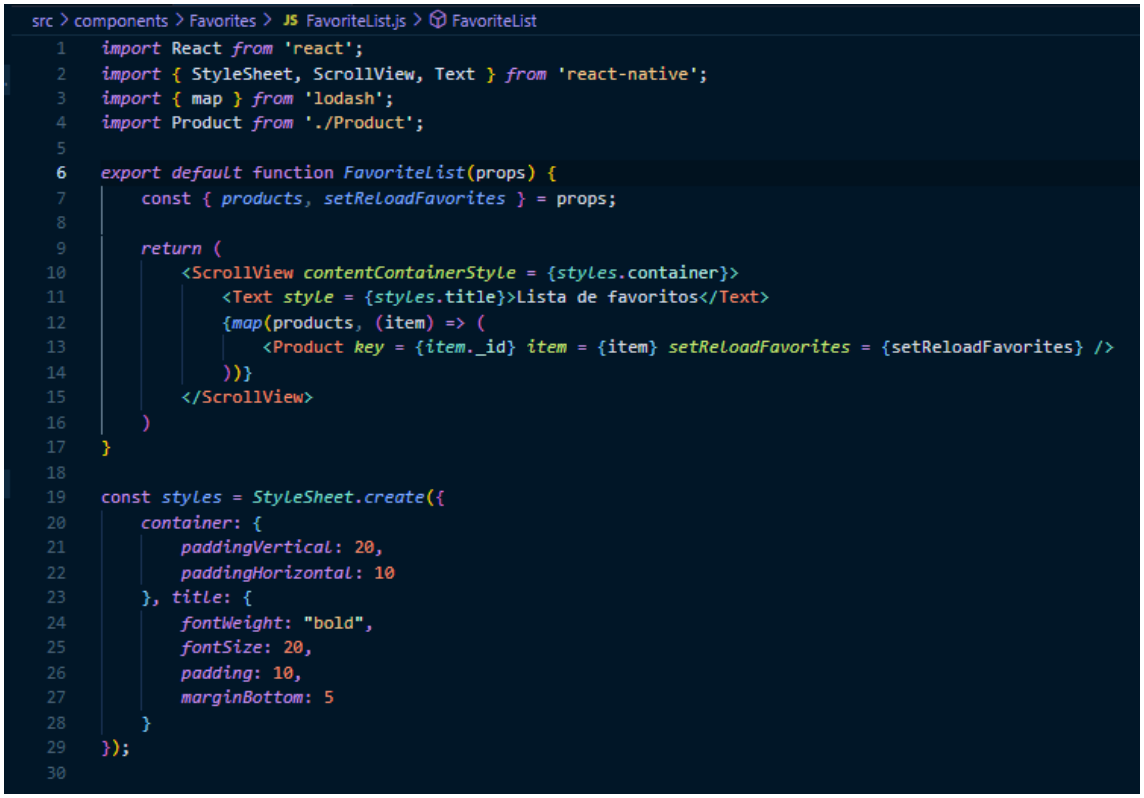
src > screens > JS Favorites.js > Favorites

```
import React, { useState, useCallback } from 'react'
import { StyleSheet, View, Text } from 'react-native';
import { useFocusEffect } from '@react-navigation/native';
import { size } from 'lodash';
import StatusBar from '../components/StatusBar';
import Search from '../components/Search';
import ScreenLoading from '../components/ScreenLoading';
import FavoriteList from '../components/Favorites/FavoriteList';
import { getFavorites } from '../api/favorite';
import useAuth from '../hooks/useAuth'
import colors from '../styles/colors';

export default function Favorites() {
    const [products, setProducts] = useState(null);
    const [reloadFavorites, setReloadFavorites] = useState(false)
    const { auth } = useAuth();

    useFocusEffect(
        useCallback(() => {
            setProducts(null);
            (async () => {
                const response = await getFavorites(auth);
                setProducts(response);
            })();
            setReloadFavorites(false);
        }, [reloadFavorites])
    );

    return (
        <>
            <StatusBar backgroundColor = {colors.bgDark} barStyle = 'light-content' />
            <Search />
            {!products ? (
                <ScreenLoading text = "Cargando mis favoritos" size = "large" />
            ) : size(products) === 0 ? (
                <View style = {styles.container}>
                    <Text style = {styles.title}>Lista de favoritos</Text>
                    <Text style = {styles.otherText}>No tienes productos en tu lista</Text>
                </View>
            ) : (
                <FavoriteList products = {products} setReloadFavorites = {setReloadFavorites} />
            )}
        </>
    );
}
```

Fuente: Elaboración propia

El componente FavoriteList.js es un contenedor que enlista los distintos productos agregados a la pantalla de favoritos, productos que se cargan individualmente gracias al componente Product.js.

Figura 59: Componentes para listar productos en favoritos



Fuente: Elaboración propia

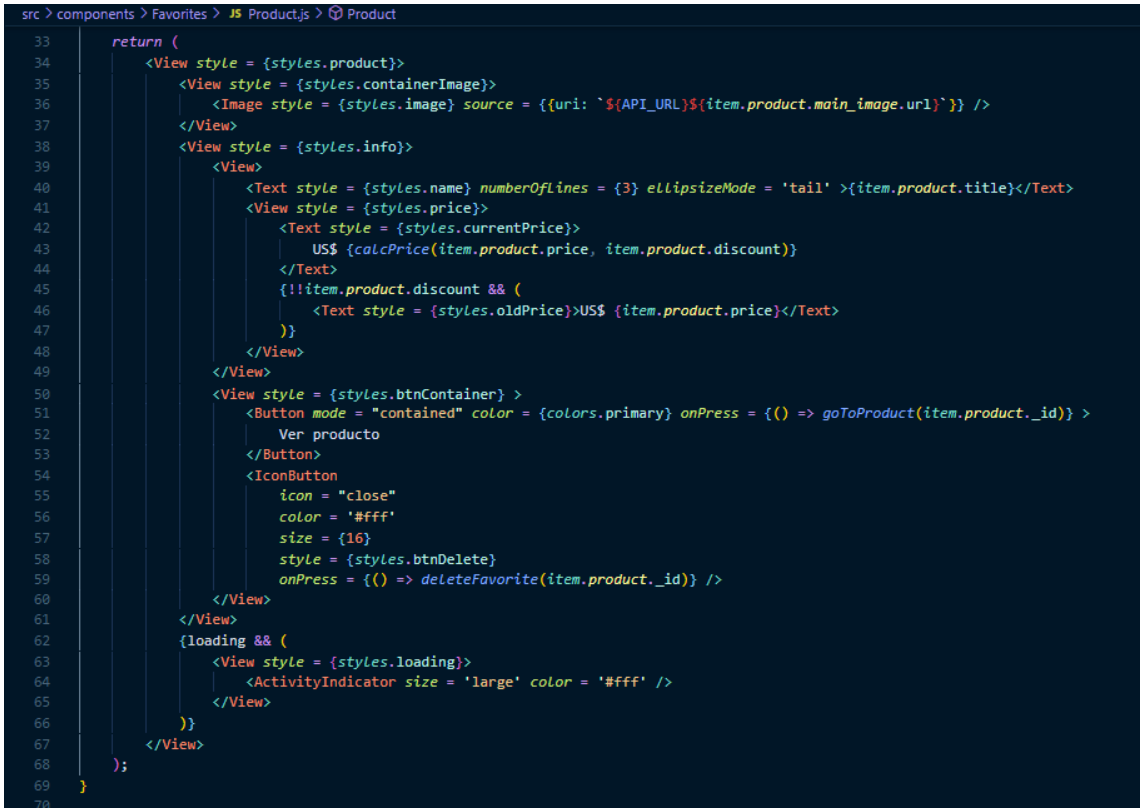
El componente Product.js carga los productos uno por uno, en esas fichas individuales se indica la imagen principal del producto, su nombre, precio y descuento, en caso de tener uno; además de dos botones, uno para ir a la ventana del producto y otro para eliminarlo de la lista de favoritos.

Figura 60: Componente para información de productos en favoritos-1

```
src > components > Favorites > JS Product.js > Product
import React, { useState } from 'react';
import { StyleSheet, View, Text, Image, ActivityIndicator } from 'react-native';
import { Button, IconButton } from 'react-native-paper';
import { useNavigation } from '@react-navigation/native';
import { deleteFavoriteApi } from '../../api/favorite';
import useAuth from '../../hooks/useAuth';
import { API_URL } from '../../utils/constants';
import colors from '../../styles/colors';

export default function Product(props) {
    const { item, setReloadFavorites } = props;
    const [loading, setLoading] = useState(false)
    const navigation = useNavigation();
    const { auth } = useAuth();

    const calcPrice = (price, discount) => {
        if(!discount) return (price).toFixed(2);
        const discountAmount = (price * discount)/100;
        return(price - discountAmount).toFixed(2);
    };

    const goToProduct = (id) => {
        navigation.navigate("product", {idProduct: id})
    };

    const deleteFavorite = async (id) => {
        setLoading(true);
        await deleteFavoriteApi(auth, id);
        setReloadFavorites(true);
        setLoading(false);
    }
```

Fuente: Elaboración propia

Figura 61: Componente para información de productos en favoritos-2



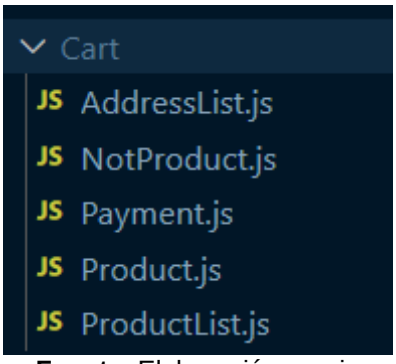
```
src > components > Favorites > JS Product.js > Product
    return (
        <View style = {styles.product}>
            <View style = {styles.containerImage}>
                <Image style = {styles.image} source = {{uri: `${API_URL}${item.product.main_image.url}`}} />
            </View>
            <View style = {styles.info}>
                <View>
                    <Text style = {styles.name} numberOfLines = {3} ellipsizeMode = 'tail' >{item.product.title}</Text>
                    <View style = {styles.price}>
                        <Text style = {styles.currentPrice}>
                            US$ {calcPrice(item.product.price, item.product.discount)}
                        </Text>
                        {!!item.product.discount && (
                            <Text style = {styles.oldPrice}>US$ {item.product.price}</Text>
                        )}
                    </View>
                </View>
                <View style = {styles.btnContainer} >
                    <Button mode = "contained" color = {colors.primary} onPress = {() => goToProduct(item.product._id)} >
                        Ver producto
                    </Button>
                    <IconButton
                        icon = "close"
                        color = '#fff'
                        size = {16}
                        style = {styles.btnDelete}
                        onPress = {() => deleteFavorite(item.product._id)} />
                </View>
            </View>
            {loading && (
                <View style = {styles.loading}>
                    <ActivityIndicator size = 'large' color = '#fff' />
                </View>
            )}
        </View>
    );
}
```

Fuente: Elaboración propia

2.7.4.5. Carrito

La pantalla de carrito carga componentes que permiten observar los productos que se desean comprar; cuando no exista una lista de productos en el carrito la pantalla cargará un componente que muestre la inexistencia de los productos.

Figura 62: Componentes para la sección del carrito



Fuente: Elaboración propia

Figura 63: Screen principal para visualizar el carrito de compras-1

src > screens > JS Cart.js > Cart

```
import React, { useState, useCallback, useEffect } from 'react';
import { StyleSheet, ScrollView } from 'react-native';
import { useFocusEffect } from '@react-navigation/native';
import { size } from 'lodash';
import { KeyboardAwareScrollView } from 'react-native-keyboard-aware-scroll-view';
import StatusBar from '../components/StatusBar';
import Search from '../components/Search';
import NotProducts from '../components/Cart/NotProduct';
import ProductList from '../components/Cart/ProductList';
import AddressList from '../components/Cart/AddressList';
import Payment from '../components/Cart/Payment';
import useAuth from '../hooks/useAuth';
import { getProductCartApi } from '../api/cart';
import { getAddressesApi } from '../api/address';
import colors from '../styles/colors'

export default function Cart() {
    const [cart, setCart] = useState(null);
    const [products, setProducts] = useState(null);
    const [reloadCart, setReloadCart] = useState(false);
    const [addresses, setAddresses] = useState(null);
    const [selectedAddress, setSelectedAddress] = useState(null);
    const [totalPayment, setTotalPayment] = useState(null)
    const { auth } = useAuth();



    useFocusEffect(
        useCallback(() => {
            setCart(null);
            setAddresses(null);
            setSelectedAddress(null)
            loadCart();
            loadAddresses();
            setReloadCart(false);
        }, [setReloadCart])
    );

    useEffect(() => {
        if (reloadCart) {
            loadCart();
            setReloadCart(false);
        }
    }, [reloadCart]);
```

Fuente: Elaboración propia

Figura 64: Screen principal para visualizar el carrito de compras-2

```
src > screens > JS Cart.js > Cart
46      const loadCart = async () => {
47          const response = await getProductCartApi();
48          setCart(response);
49      }
50
51      const loadAddresses = async () => {
52          const response = await getAddressesApi(auth);
53          setAddresses(response);
54      }
55
56      return (
57          <>
58              <StatusBar backgroundColor = {colors.bgDark} barStyle = "light-content" />
59              {!cart || size(cart) === 0 ? (
60                  <>
61                      <Search />
62                      <NotProducts />
63                  </>
64              ) : (
65                  <KeyboardAwareScrollView extraScrollHeight = {25}>
66                      <ScrollView style = {styles.cartContainer}>
67                          <ProductList
68                              cart = {cart}
69                              products = {products}
70                              setProducts = {setProducts}
71                              setReloadCart = {setReloadCart}
72                              setTotalPayment = {setTotalPayment} />
73                          <AddressList
74                              addresses = {addresses}
75                              selectedAddress = {selectedAddress}
76                              setSelectedAddress = {setSelectedAddress} />
77                          <Payment
78                              products = {products}
79                              selectedAddress = {selectedAddress}
80                              totalPayment = {totalPayment} />
81                      </ScrollView>
82                  </KeyboardAwareScrollView>
83              )}
84          </>
85      );
86  }
87
```

Fuente: Elaboración propia

La screen Cart.js carga directamente 4 componentes; en caso de no haber productos el carrito carga el componente NotProdcut.js, este componente muestra únicamente una pantalla indicando que no existen productos en el carrito.

Figura 65: Componente para carrito vacío

src > components > Cart > JS NotProduct.js > ResultNotFound

```
import React from 'react'
import { StyleSheet, View, Text } from 'react-native'

export default function ResultNotFound() {
    return (
        <View style = {styles.container}>
            <Text style = {styles.title}>Carrito de compras</Text>
            <Text style = {styles.otherText}>No tienes productos en el carrito.</Text>
        </View>
    )
}

const styles = StyleSheet.create({
    container: {
        paddingVertical: 20,
        paddingHorizontal: 10
    }, title: {
        fontWeight: "bold",
        fontSize: 20,
        padding: 10,
        marginBottom: 5
    }, otherText: {
        fontSize: 18,
        padding: 5
    }
})
```

Fuente: Elaboración propia

Si se encuentran productos se cargará una lista a través del componente ProductList.js; este componente carga de manera individual cada producto agregado a la lista a través del componente Product.js.

Figura 66: Componente para lista de productos del carrito

src > components > Cart > JS ProductList.js > ProductList

```
import React, { useEffect } from 'react';
import { StyleSheet, View, Text } from 'react-native';
import { map } from 'lodash';
import ScreenLoading from '../../components/ScreenLoading';
import Product from '../../components/Cart/Product';
import { getProductApi } from '../../api/product';

export default function ProductList(props) {
    const { cart, products, setProducts, setReloadCart, setTotalPayment } = props;

    useEffect(() => {
        setProducts(null);
        (async () => {
            const productTemp = [];
            let totalPaymentTemp = 0;
            let totalValue = 0;
            for await (const product of cart) {
                const response = await getProductApi(product.idProduct);
                response.quantity = product.quantity
                productTemp.push(response);
                totalPaymentTemp += (response.price - (response.price * response.discount)/100) * response.quantity;
                totalValue = (totalPaymentTemp).toFixed(2)
            }
            setProducts(productTemp);
            setTotalPayment(totalValue);
        })()
    }, [cart])

    return (
        <View>
            <Text style = {styles.title}>Carrito de compras</Text>
            {!products ? (
                <ScreenLoading text = "Cargando carrito..." size = "large" />
            ) : (
                map(products, (product) => (
                <Product
                    key = {product._id}
                    product = {product}
                    setReloadCart = {setReloadCart} />
                ))
            )}
        </View>
    )
}
```

Fuente: Elaboración propia

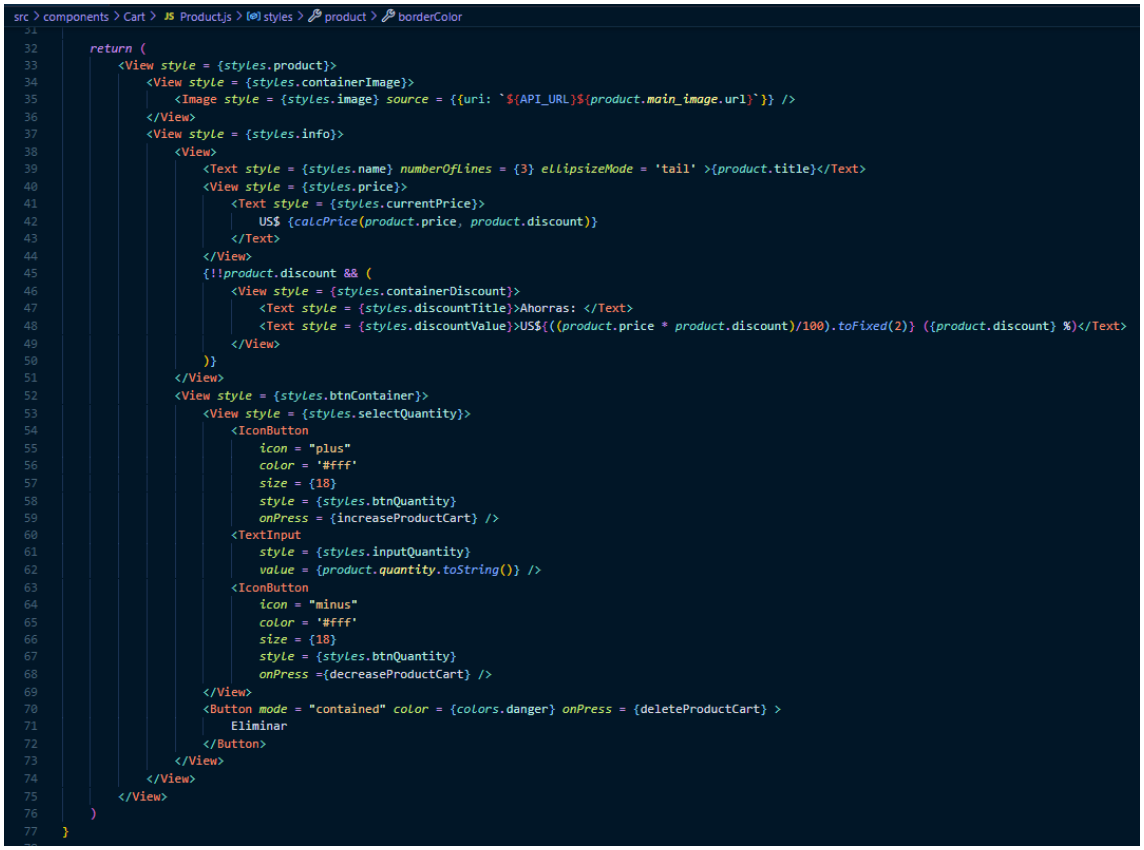
El componente Product.js carga la información de cada producto agregado, incluyendo la imagen, el nombre, el precio y el descuento en caso de tener; además de una sección de botones para aumentar o disminuir la cantidad de cada producto y uno para eliminar el producto directamente del carrito.

Figura 67: Componente para la información de productos del carrito-1

```
src > components > Cart > JS Product.js > styles > product > borderColor
import React from 'react';
import { StyleSheet, View, Text, Image, TextInput } from 'react-native';
import { Button, IconButton } from 'react-native-paper';
import { deleteProductCartApi, increaseProductCartApi, decreaseProductCartApi } from "../../api/cart"
import { API_URL } from '../../utils/constants';
import colors from '../../styles/colors';

export default function Product(props) {
    const { product, setReloadCart } = props;

    const calcPrice = (price, discount) => {
        if(!discount) return (price).toFixed(2);
        const discountAmount = (price * discount)/100;
        return (price - discountAmount).toFixed(2);
    };

    const deleteProductCart = async () => {
        const response = await deleteProductCartApi(product._id);
        if(response) setReloadCart(true);
    };

    const increaseProductCart = async () => {
        const response = await increaseProductCartApi(product._id);
        if(response) setReloadCart(true);
    }

    const decreaseProductCart = async () => {
        const response = await decreaseProductCartApi(product._id)
        if(response) setReloadCart(true);
    }
```

Fuente: Elaboración propia

Figura 68: Componente para la información de productos del carrito-2



```
src > components > Cart > JS Product.js > styles > product > borderColor
    return (
        <View style = {styles.product}>
            <View style = {styles.containerImage}>
                <Image style = {styles.image} source = {{uri: `${API_URL}${product.main_image.url}`}} />
            </View>
            <View style = {styles.info}>
                <View>
                    <Text style = {styles.name} numberOfLines = {3} ellipsizeMode = 'tail' >{product.title}</Text>
                    <View style = {styles.price}>
                        <Text style = {styles.currentPrice}>
                            US$ {calcPrice(product.price, product.discount)}
                        </Text>
                    </View>
                    {!!product.discount && (
                        <View style = {styles.containerDiscount}>
                            <Text style = {styles.discountTitle}>Ahorras: </Text>
                            <Text style = {styles.discountValue}>US${((product.price * product.discount)/100).toFixed(2)} ({product.discount} %)</Text>
                        </View>
                    )}
                </View>
                <View style = {styles.btnContainer}>
                    <View style = {styles.selectQuantity}>
                        <IconButton
                            icon = "plus"
                            color = '#fff'
                            size = {18}
                            style = {styles.btnQuantity}
                            onPress = {increaseProductCart} />
                        <TextInput
                            style = {styles.inputQuantity}
                            value = {product.quantity.toString()} />
                        <IconButton
                            icon = "minus"
                            color = '#fff'
                            size = {18}
                            style = {styles.btnQuantity}
                            onPress ={decreaseProductCart} />
                    </View>
                    <Button mode = "contained" color = {colors.danger} onPress = {deleteProductCart} >
                        Eliminar
                    </Button>
                </View>
            </View>
        </View>
    )
}
```

Fuente: Elaboración propia

Seguido de la lista de productos se encuentra una lista de direcciones, las cuales fueron ingresadas por el cliente desde la pestaña de cuenta; se necesita al menos una para realizar una compra. Esta información se presenta gracias al componente AddressList.js.

Figura 69: Componente para cargar direcciones creadas

Fuente: Elaboración propia

Después se aprecia un formulario de compra, dicho formulario admite la información de las tarjetas de crédito para la realización de los pagos, dicha información no es capturada ni por la aplicación ni por la base de datos, se maneja directamente por la pasarela de pago por temas de seguridad.

Figura 70: Componente para formulario de pago-1

Fuente: Elaboración propia

Figura 71: Componente para formulario de pago-2

Fuente: Elaboración propia

Los campos del formulario tienen un ingreso validado para admitir únicamente un formato correcto para el campo solicitado.

Figura 72: Componente para formulario de pago-3

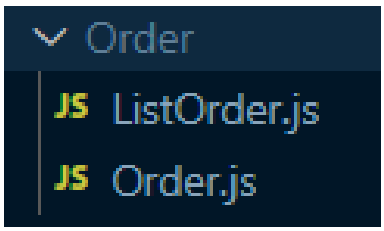
Fuente: Elaboración propia

Una vez ejecutado el pago se eliminará el carrito y se registrará el pedido realizado en la opción de Pedidos que está en la opción de Cuenta.

2.7.4.6. Pedidos

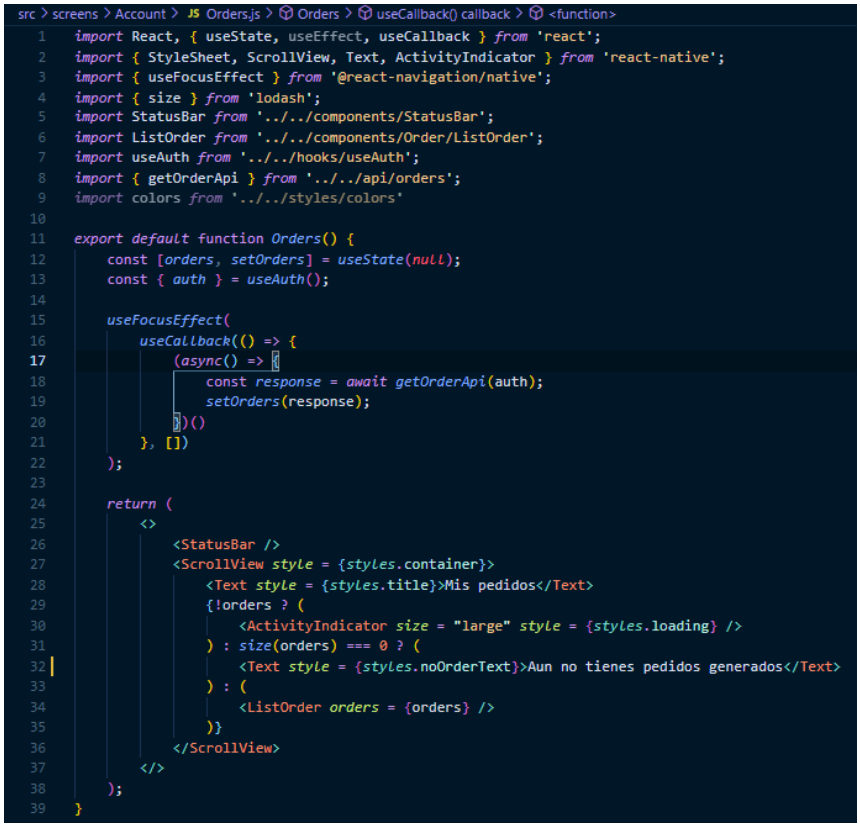
Los pedidos realizados se pueden visualizar a través de la screen Orders.js, la cual utiliza 2 componentes para cargar la información de la compra realizada desde la base de datos.

Figura 73: Componentes para pantalla de ordenes



Fuente: Elaboración propia

Figura 74: Pantalla principal para ver pedidos realizados



Fuente: Elaboración propia

La screen Orders.js primero comprueba si existe al menos un pedido ejecutado, en caso de no ser así mostrará un mensaje que indica que no tiene pedidos generados; si encuentra un pedido generado carga el componente ListOrder.js; el cual es una lista que carga los productos comprados de uno en uno.

Figura 75: Componente para cargar lista de ordenes

Fuente: Elaboración propia

Este componente carga el componente Order.js; el cual muestra la información de los productos comprados; indicando la imagen, el nombre, la cantidad comprada, el valor pagado y el id del pedido generado.

Figura 76: Componente para ver información de pedidos realizados

src > components > Order > JS Order.js > Order

```
import React from 'react';
import { StyleSheet, View, Text, Image } from 'react-native';
import { API_URL } from '../../utils/constants'

export default function Order(props) {
    const { order } = props;

    return (
        <View style = {styles.container}>
            <View style = {styles.containerImage}>
                <Image style = {styles.image} source = {{ uri: `${API_URL}${order.product.main_image.url}` }} />
            </View>
            <View style = {styles.info}>
                <Text style = {styles.name} numberOflines = {2} ellipsizeMode = 'tail'>
                    {order.product.title}
                </Text>
                <Text>Cantidad: {order.quantity}</Text>
                <Text>Total pagado: USD$ {order.productsPayment}</Text>
                <Text>Pedido: {order.idPayment}</Text>
            </View>
        </View>
    )
}
```

Fuente: Elaboración propia

2.7.4.7. Navegación

La aplicación utiliza 3 sistemas de navegación dada la cantidad de pantallas que maneja. AccountStack.js maneja las direcciones para abrir los formularios en la pestaña de cuenta que permiten modificar la información del cliente.

Figura 77: Stack de navegación de la información de la cuenta


```
13  const Stack = createStackNavigator();
14
15  export default function AccountStack() {
16      return (
17          <Stack.Navigator screenOptions = {{
18              headerTintColor: colors.fontLight,
19              headerStyle: { backgroundColor: colors.bgDark },
20              cardStyle: { backgroundColor: colors.bgLight },
21              headerTitleAlign: "center"
22          }} >
23              <Stack.Screen
24                  name = "account"
25                  component = {Account}
26                  options = {{ title: "Cuenta", headerShown: false }} />
27              <Stack.Screen
28                  name = "change-name"
29                  component = {ChangeName}
30                  options = {{ title: "Cambiar nombre y apellido" }} />
31              <Stack.Screen
32                  name = "change-email"
33                  component = {ChangeEmail}
34                  options = {{ title: "Cambiar email" }} />
35              <Stack.Screen
36                  name = "change-username"
37                  component = {ChangeUsername}
38                  options = {{ title: "Cambiar username" }} />
39              <Stack.Screen
40                  name = "change-password"
41                  component = {ChangePassword}
42                  options = {{ title: "Cambiar contraseña" }} />
43              <Stack.Screen
44                  name = "adresses"
45                  component = {Adresses}
46                  options = {{ title: "Mis direcciones" }} />
47              <Stack.Screen
48                  name = "add-address"
49                  component = {AddAddress}
50                  options = {{ title: "Nueva dirección" }} />
51              <Stack.Screen
52                  name = "orders"
53                  component = {Orders}
54                  options = {{ title: "Mis pedidos" }} />
55          </Stack.Navigator>
56      )
57  }
58
```

Fuente: Elaboración propia

AppNavigation.js maneja la navegación de las 4 pantallas principales de la aplicación (Home, Favorites, Cart y Account).

Figura 78: Stack de navegación de pantallas principales

src > navigation > JS AppNavigation.js > setIcon

```
import React from 'react';
import { StyleSheet } from 'react-native'
import { NavigationContainer } from '@react-navigation/native';
import { createMaterialBottomTabNavigator } from '@react-navigation/material-bottom-tabs';
import AwesomeIcon from 'react-native-vector-icons/FontAwesome';
import colors from '../styles/colors';
import Favorites from '../screens/Favorites';
import Cart from '../screens/Cart';
import AccountStack from './AccountStack';
import ProductStack from './ProductStack';

const Tab = createMaterialBottomTabNavigator();

export default function AppNavigation() {
    return (
        <NavigationContainer>
            <Tab.Navigator barStyle = { styles.navigation } screenOptions = {({ route }) => ({
                tabBarIcon: (routeStatus) => {
                    return setIcon(route, routeStatus);
                }
            })} >
                <Tab.Screen
                    name = "home"
                    component = {ProductStack}
                    options = {{title: "Inicio",}} />
                <Tab.Screen
                    name = "favorites"
                    component = {Favorites}
                    options = {{title:"Favoritos",}} />
                <Tab.Screen
                    name = "cart"
                    component = {Cart}
                    options = {{title:"Carrito",}} />
                <Tab.Screen
                    name = "account"
                    component = {AccountStack}
                    options = {{title:"Mi cuenta",}} />
            </Tab.Navigator>
        </NavigationContainer>
    );
}
```

Fuente: Elaboración propia

ProductStack.js maneja la navegación dentro de la pestaña “Home” para poder visualizar todos los productos, los productos de manera individual y los productos que devuelven las búsquedas realizadas.

Figura 79: Stack de navegación entre pantallas de productos

src > navigation > JS ProductStack.js > ProductStack

```
1   import React from 'react';
2   import { createStackNavigator } from '@react-navigation/stack';
3   import Home from '../screens/Product/Home';
4   import Product from '../screens/Product/Product';
5   import Search from '../screens/Product/Search';
6   import colors from '../styles/colors';
7
8   const Stack = createStackNavigator();
9
10  export default function ProductStack() {
11      return (
12          <Stack.Navigator screenOptions = {{
13              headerTintColor: colors.fontLight,
14              headerStyle: { backgroundColor: colors.bgDark },
15              cardStyle: { backgroundColor: colors.bgLight },
16              headerTitleAlign: "center"
17          }} >
18              <Stack.Screen
19                  name = "home"
20                  component = {Home}
21                  options = {{headerShown: false}} />
22              <Stack.Screen
23                  name = "product"
24                  component = {Product}
25                  options = {{headerShown: false}} />
26              <Stack.Screen
27                  name = "search"
28                  component = {Search}
29                  options = {{headerShown: false}} />
30          </Stack.Navigator>
31      )
32  }
33
```

Fuente: Elaboración propia

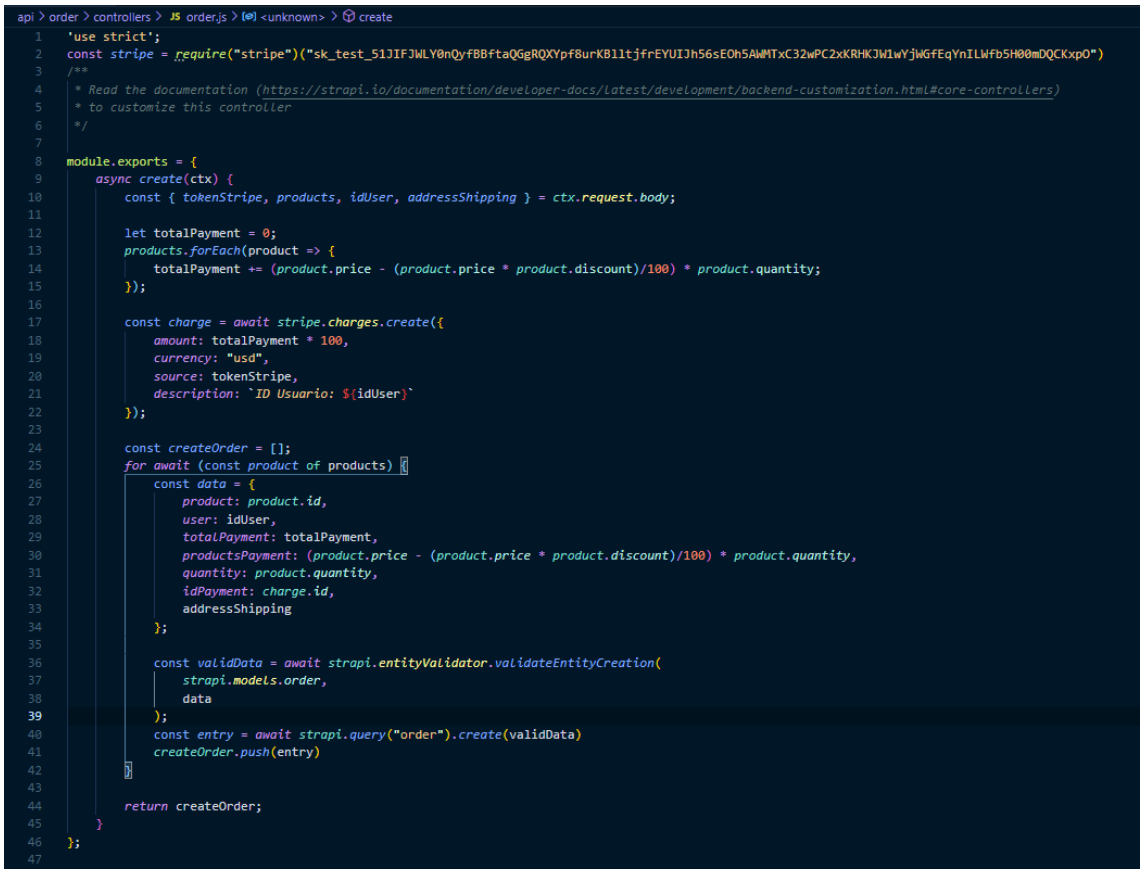
2.7.4.8. Pasarela de pago

Si bien el servidor de Strapi realiza una configuración de manera automática al momento de agregar más colecciones de datos, el pago requiere una configuración especifica; esto debido a que no se puede enviar la información del coste total desde el frontend por motivos de evitar algún tipo de violación de seguridad por parte de algún usuario.

Una vez se instala Stripe en el servidor se utiliza una clave que permite realizar envíos hacia su plataforma de pago y se ejecuta el mismo cálculo del total que se presenta en el frontend.

Ese cargo se envía hacia la pasarela de pago para su ejecución y hacia el servidor de stripe para que los pedidos se puedan obtener desde la base de datos.

Figura 80: Controlador para el cálculo del coste total



Fuente: Elaboración propia

Los stacks de navegación utilizan un nombre que identifica la pantalla a cargar; la opción “component” indicando cual es la screen y opciones configurables para visualizar la cabecera de la aplicación en caso de no requerir la barra de búsqueda.

3. EVALUACIÓN DEL PROTOTIPO

La fase de evaluación se desarrolla según el cumplimiento de las fases de la metodología XP; dentro de esta fase se desarrollan actividades y formularios que se utilizan para valorar el cumplimiento de las funciones del prototipo desarrollado.

3.1 Plan de evaluación

Para la evaluación del funcionamiento del prototipo se realizarán dos tipos de pruebas: las pruebas unitarias realizadas en base a las historias de usuario por parte del desarrollador, y las pruebas de control de calidad en base a una métrica específica, en este caso, la métrica de calidad ISO 25010 para productos de software.

3.1.1. Pruebas unitarias

Las pruebas unitarias son un tipo de evaluación muy popular que consiste en aplicar un caso de prueba sobre una unidad de software o un grupo relacionado; definición que se ha mantenido con el pasar del tiempo a pesar de los cambios sobre el contexto de desarrollo de software. [42]

Estas pruebas se aplican por parte del desarrollador o desarrolladores a conjuntos del software, como los módulos desarrollados o sus componentes descritos en las historias de usuario.

A continuación, se describen las pruebas de aceptación sobre los objetivos a cumplir de cada historia de usuario.

Tabla 30: Prueba de aceptación N°1

Pruebas de Aceptación	
Plataforma para e-commerce con realidad aumentada	
Código de Prueba	PA001
Nº Historia de Usuario que prueba:	1
Nombre de historia de usuario que aprueba:	Registro de información usuario
Descripción:	El usuario ingresa sus datos de registro en un formulario validado que enviará esa información

	a la base de datos para usarse en un posterior logueo
Pasos de Ejecución	1. El cliente debe ingresar a la aplicación. 2. Agregar nombre de usuario 3. Agregar correo 4. Agregar y confirmar contraseña 5. Dar clic en Registrarse
Resultado obtenido:	El usuario logró registrarse en el sistema.
Evaluación:	**APROBADO**

Fuente: Elaboración propia

Tabla 31: Prueba de aceptación N°2

Pruebas de Aceptación	
Plataforma para e-commerce con realidad aumentada	
Código de Prueba	PA002
Nº Historia de Usuario que prueba:	2
Nombre de historia de usuario que aprueba:	Loguin del usuario
Descripción:	El usuario ingresa su correo electrónico/usuario y su contraseña para acceder en la aplicación.
Pasos de Ejecución	1. El cliente debe ingresar a la aplicación. 2. Introducir nombre de usuario 3. Introducir y confirmar contraseña 4. Dar clic en Iniciar sesión
Resultado obtenido:	El usuario logró iniciar sesión en la aplicación.
Evaluación:	**APROBADO**

Fuente: Elaboración propia

Tabla 32: Prueba de aceptación N°3

Pruebas de Aceptación	
Plataforma para e-commerce con realidad aumentada	
Código de Prueba	PA003
Nº Historia de Usuario que prueba:	3
Nombre de historia de usuario que aprueba:	Navegación entre pestañas
Descripción:	El usuario podrá cambiar las pantallas dentro de la aplicación (Productos/Favoritos/Carrito/Cuenta).
Pasos de Ejecución	1. El cliente debe ingresar a la aplicación. 2. El usuario debe seleccionar una opción del menú inferior

Resultado obtenido:	El usuario logró navegar entre las pestañas principales.
Evaluación:	**APROBADO**

Fuente: Elaboración propia

Tabla 33: Prueba de aceptación N°4

Pruebas de Aceptación	
Plataforma para e-commerce con realidad aumentada	
Código de Prueba	PA004
Nº Historia de Usuario que prueba:	4
Nombre de historia de usuario que aprueba:	Agregar y/o actualizar nombres
Descripción:	El usuario agregará nombre y apellidos a su cuenta y podrá actualizarlos
Pasos de Ejecución	1. El cliente debe ingresar a la aplicación. 2. El usuario debe seleccionar una opción de cuenta del menú inferior. 3. El usuario debe seleccionar la opción de “Cambiar nombre”. 4. El usuario debe escribir un nuevo nombre o editar el existente. 5. El usuario debe confirmar el cambio realizado.
Resultado obtenido:	El usuario logró agregar/editar su nombre en la información de la cuenta.
Evaluación:	**APROBADO**

Fuente: Elaboración propia

Tabla 34: Prueba de aceptación N°5

Pruebas de Aceptación	
Plataforma para e-commerce con realidad aumentada	
Código de Prueba	PA005
Nº Historia de Usuario que prueba:	5
Nombre de historia de usuario que aprueba:	Actualizar e-mail
Descripción:	El usuario podrá actualizar el correo electrónico para el inicio de sesión.
Pasos de Ejecución	1. El cliente debe ingresar a la aplicación. 2. El usuario debe seleccionar una opción de cuenta del menú inferior.

	3. El usuario debe seleccionar la opción de “Cambiar email”. 4. El usuario debe escribir un nuevo correo. 5. El usuario debe confirmar el cambio realizado.
Resultado obtenido:	El usuario logró editar su correo en la información de la cuenta.
Evaluación:	**APROBADO**

Fuente: Elaboración propia

Tabla 35: Prueba de aceptación N°6

Pruebas de Aceptación	
Plataforma para e-commerce con realidad aumentada	
Código de Prueba	PA006
Nº Historia de Usuario que prueba:	6
Nombre de historia de usuario que aprueba:	Actualizar nombre de usuario
Descripción:	El usuario podrá actualizar el nombre de usuario para el inicio de sesión.
Pasos de Ejecución	1. El cliente debe ingresar a la aplicación. 2. El usuario debe seleccionar una opción de cuenta del menú inferior. 3. El usuario debe seleccionar la opción de “Cambiar username”. 4. El usuario debe escribir un nuevo username. 5. El usuario debe confirmar el cambio realizado.
Resultado obtenido:	El usuario logró editar su username en la información de la cuenta.
Evaluación:	**APROBADO**

Fuente: Elaboración propia

Tabla 36: Prueba de aceptación N°7

Pruebas de Aceptación	
Plataforma para e-commerce con realidad aumentada	
Código de Prueba	PA007
Nº Historia de Usuario que prueba:	7
Nombre de historia de usuario que aprueba:	Actualizar contraseña
Descripción:	El usuario podrá actualizar la contraseña para el inicio de sesión.

Pasos de Ejecución	1. El cliente debe ingresar a la aplicación. 2. El usuario debe seleccionar una opción de cuenta del menú inferior. 3. El usuario debe seleccionar la opción de “Cambiar contraseña”. 4. El usuario debe escribir una nueva contraseña. 5. El usuario debe confirmar el cambio realizado.
Resultado obtenido:	El usuario logró editar su contraseña en la información de la cuenta.
Evaluación:	**APROBADO**

Fuente: Elaboración propia

Tabla 37: Prueba de aceptación N°8

Pruebas de Aceptación	
Plataforma para e-commerce con realidad aumentada	
Código de Prueba	PA008
Nº Historia de Usuario que prueba:	8
Nombre de historia de usuario que aprueba:	Crear direcciones de envío
Descripción:	El usuario podrá crear direcciones para el envío de sus pedidos.
Pasos de Ejecución	1. El cliente debe ingresar a la aplicación. 2. El usuario debe seleccionar una opción de cuenta del menú inferior. 3. El usuario debe seleccionar la opción de “Mis direcciones”. 4. El usuario debe seleccionar la opción de “Añadir nueva dirección”. 5. El usuario debe rellenar el formulario de registro. 6. El usuario debe confirmar la operación realizada.
Resultado obtenido:	El usuario logró agregar una dirección de envío.
Evaluación:	**APROBADO**

Fuente: Elaboración propia

Tabla 38: Prueba de aceptación N°9

Pruebas de Aceptación	
Plataforma para e-commerce con realidad aumentada	
Código de Prueba	PA009

Nº Historia de Usuario que prueba:	9
Nombre de historia de usuario que aprueba:	Modificar direcciones de envío
Descripción:	El usuario podrá modificar sus direcciones de envío.
Pasos de Ejecución	1. El cliente debe ingresar a la aplicación. 2. El usuario debe seleccionar una opción de cuenta del menú inferior. 3. El usuario debe seleccionar la opción de “Mis direcciones”. 4. El usuario debe seleccionar la opción de “Editar” en una de las direcciones creadas. 5. El usuario debe modificar el formulario de direcciones. 6. El usuario debe confirmar la operación realizada.
Resultado obtenido:	El usuario logró editar una de sus direcciones creadas.
Evaluación:	**APROBADO**

Fuente: Elaboración propia

Tabla 39: Prueba de aceptación N°10

Pruebas de Aceptación	
Plataforma para e-commerce con realidad aumentada	
Código de Prueba	PA010
Nº Historia de Usuario que prueba:	10
Nombre de historia de usuario que aprueba:	Eliminar direcciones de envío
Descripción:	El usuario podrá eliminar sus direcciones de envío.
Pasos de Ejecución	1. El cliente debe ingresar a la aplicación. 2. El usuario debe seleccionar una opción de cuenta del menú inferior. 3. El usuario debe seleccionar la opción de “Mis direcciones”. 4. El usuario debe seleccionar la opción de “Eliminar” en una de las direcciones creadas.
Resultado obtenido:	El usuario logró eliminar una de sus direcciones creadas.
Evaluación:	**APROBADO**

Fuente: Elaboración propia

Tabla 40: Prueba de aceptación N°11

Pruebas de Aceptación	
Plataforma para e-commerce con realidad aumentada	
Código de Prueba	PA011
Nº Historia de Usuario que prueba:	11
Nombre de historia de usuario que aprueba:	Agregar productos
Descripción:	El usuario accede a una vista de los productos almacenados en la base de datos.
Pasos de Ejecución	1. El administrador agregará los productos directamente en la base de datos 2. El usuario apreciará los productos de la base de datos en la opción de “Home” de la aplicación.
Resultado obtenido:	El usuario logró visualizar los productos agregados.
Evaluación:	**APROBADO**

Fuente: Elaboración propia

Tabla 41: Prueba de aceptación N°12

Pruebas de Aceptación	
Plataforma para e-commerce con realidad aumentada	
Código de Prueba	PA012
Nº Historia de Usuario que prueba:	12
Nombre de historia de usuario que aprueba:	Agregar producto a favoritos/carrito
Descripción:	El cliente puede agregar el producto que desea a su lista de favoritos y carrito
Pasos de Ejecución	1. El cliente debe ingresar a la aplicación. 2. El usuario debe seleccionar la opción de “Home” del menú inferior. 3. El usuario debe seleccionar un producto en específico. 4. El usuario debe seleccionar las opciones de “Añadir al carrito” o “Añadir a favoritos”.
Resultado obtenido:	El usuario logró visualizar los productos agregados.
Evaluación:	**APROBADO**

Fuente: Elaboración propia

Tabla 42: Prueba de aceptación N°13

Pruebas de Aceptación	
Plataforma para e-commerce con realidad aumentada	
Código de Prueba	PA013
Nº Historia de Usuario que prueba:	13
Nombre de historia de usuario que aprueba:	Agregar banners promocionales
Descripción:	El usuario accede a un banner de productos promocionales que lo redirigen a los productos de la tienda.
Pasos de Ejecución	1. El administrador agregará banners de productos en promoción o nuevos directamente en la base de datos. 2. El usuario podrá acceder a ellos desde la pantalla de “Home”.
Resultado obtenido:	El usuario logró visualizar los banners promocionales de los productos agregados.
Evaluación:	**APROBADO**

Fuente: Elaboración propia

Tabla 43: Prueba de aceptación N°14

Pruebas de Aceptación	
Plataforma para e-commerce con realidad aumentada	
Código de Prueba	PA014
Nº Historia de Usuario que prueba:	14
Nombre de historia de usuario que aprueba:	Búsqueda de productos desde la barra
Descripción:	El usuario puede buscar un producto con palabras clave que se obtienen desde el nombre y etiquetas del producto.
Pasos de Ejecución	1. El cliente debe ingresar a la aplicación. 2. El usuario debe seleccionar la opción de “Home” del menú inferior. 3. El cliente debe seleccionar la barra de búsqueda de la parte superior de la pantalla. 4. El cliente debe ingresar un término de búsqueda. 5. Se mostrará en la pantalla los resultados que coincidan con el término ingresado 6. El cliente seleccionará el producto que se relacione más a la búsqueda realizada.

Resultado obtenido:	El usuario logró realizar una búsqueda de productos.
Evaluación:	**APROBADO**

Fuente: Elaboración propia

Tabla 44: Prueba de aceptación N°15

Pruebas de Aceptación	
Plataforma para e-commerce con realidad aumentada	
Código de Prueba	PA015
Nº Historia de Usuario que prueba:	15
Nombre de historia de usuario que aprueba:	Búsqueda de productos desde el historial
Descripción:	El usuario puede buscar un producto desde el historial de búsqueda de productos previos.
Pasos de Ejecución	1. El cliente debe ingresar a la aplicación. 2. El usuario debe seleccionar la opción de "Home" del menú inferior. 3. El cliente debe seleccionar la barra de búsqueda de la parte superior de la pantalla. 4. Al seleccionar la barra de búsqueda se apreciará una pantalla que registra las búsquedas realizadas. 5. El cliente seleccionará uno de los términos registrados. 6. La aplicación mostrará el resultado de la búsqueda en base al término seleccionado.
Resultado obtenido:	El usuario logró realizar una búsqueda desde el historial de productos.
Evaluación:	**APROBADO**

Fuente: Elaboración propia

Tabla 45: Prueba de aceptación N°16

Pruebas de Aceptación	
Plataforma para e-commerce con realidad aumentada	
Código de Prueba	PA016
Nº Historia de Usuario que prueba:	16
Nombre de historia de usuario que aprueba:	Agregar productos a favoritos
Descripción:	El usuario puede seleccionar los productos que quiere añadir a favoritos
Pasos de Ejecución	1. El cliente debe ingresar a la aplicación.

	2. El usuario debe seleccionar la opción de “Home” del menú inferior. 3. El cliente debe seleccionar un producto o buscarlo. 4. El cliente debe seleccionar la opción de “Añadir a favoritos” en el producto. 5. El cliente debe seleccionar la opción de “Favoritos” en el menú inferior. 6. El cliente observará la lista de productos añadidos a su lista de favoritos.
Resultado obtenido:	El usuario logró agregar productos a su lista de favoritos.
Evaluación:	**APROBADO**

Fuente: Elaboración propia

Tabla 46: Prueba de aceptación N°17

Pruebas de Aceptación	
Plataforma para e-commerce con realidad aumentada	
Código de Prueba	PA017
Nº Historia de Usuario que prueba:	17
Nombre de historia de usuario que aprueba:	Eliminar productos a favoritos
Descripción:	El usuario puede eliminar los productos que desee de su lista de favoritos.
Pasos de Ejecución	1. El cliente debe ingresar a la aplicación. 2. El usuario debe seleccionar la opción de “Home” del menú inferior. 3. El cliente debe un producto o buscarlo. 4. El cliente debe seleccionar la opción de “Eliminar de favoritos” en un producto previamente añadido. 5. El cliente debe seleccionar la opción de “Favoritos” en el menú inferior. 6. El cliente podrá eliminar un producto de la pantalla de favoritos con el botón “X”
Resultado obtenido:	El usuario logró eliminar productos de su lista de favoritos.
Evaluación:	**APROBADO**

Fuente: Elaboración propia

Tabla 47: Prueba de aceptación N°18

Pruebas de Aceptación

Plataforma para e-commerce con realidad aumentada	
Código de Prueba	PA018
Nº Historia de Usuario que prueba:	18
Nombre de historia de usuario que aprueba:	Agregar productos al carrito
Descripción:	El usuario puede seleccionar los productos que quiere añadir al carrito de compras.
Pasos de Ejecución	1. El cliente debe ingresar a la aplicación. 2. El usuario debe seleccionar la opción de "Home" del menú inferior. 3. El cliente debe seleccionar un producto o buscarlo. 4. El cliente debe seleccionar la opción de "Añadir al carrito" en el producto. 5. El cliente debe seleccionar la opción de "Carrito" en el menú inferior. 6. El cliente observará la lista de productos añadidos a su carrito de compras.
Resultado obtenido:	El usuario logró agregar productos a su carrito de compras.
Evaluación:	**APROBADO**

Fuente: Elaboración propia

Tabla 48: Prueba de aceptación N°19

Pruebas de Aceptación	
Plataforma para e-commerce con realidad aumentada	
Código de Prueba	PA019
Nº Historia de Usuario que prueba:	19
Nombre de historia de usuario que aprueba:	Eliminar productos del carrito
Descripción:	El usuario puede eliminar los productos que desee de su carrito de compras.
Pasos de Ejecución	1. El cliente debe ingresar a la aplicación. 2. El usuario debe seleccionar la opción de "Home" del menú inferior. 3. El cliente debe seleccionar un producto o buscarlo. 4. El cliente debe seleccionar la opción de "Eliminar del carrito" en un producto previamente añadido.

	5. El cliente debe seleccionar la opción de “Carrito” en el menú inferior. 6. El cliente podrá eliminar un producto de la pantalla de favoritos con el botón “Eliminar”
Resultado obtenido:	El usuario logró eliminar productos de su carrito de compras.
Evaluación:	**APROBADO**

Fuente: Elaboración propia

Tabla 49: Prueba de aceptación N°20

Pruebas de Aceptación	
Plataforma para e-commerce con realidad aumentada	
Código de Prueba	PA020
Nº Historia de Usuario que prueba:	20
Nombre de historia de usuario que aprueba:	Generación de pagos
Descripción:	El usuario puede ejecutar la compra de los productos de su carrito
Pasos de Ejecución	1. El cliente debe ingresar a la aplicación. 2. El usuario debe seleccionar la opción de “Home” del menú inferior. 3. El cliente debe seleccionar un producto o buscarlo. 4. El cliente debe seleccionar la opción de “Añadir al carrito” en un producto previamente añadido. 5. El cliente debe seleccionar la opción de “Carrito” en el menú inferior. 6. El cliente debe seleccionar una de las direcciones que haya ingresado 7. El cliente debe rellenar el formulario de la tarjeta 8. El cliente debe ejecutar el pago.
Resultado obtenido:	El usuario logró realizar el pago de un pedido desde la aplicación.
Evaluación:	**APROBADO**

Fuente: Elaboración propia

Tabla 50: Prueba de aceptación N°21

Pruebas de Aceptación	
Plataforma para e-commerce con realidad aumentada	
Código de Prueba	PA021

Nº Historia de Usuario que prueba:	21
Nombre de historia de usuario que aprueba:	Listado de pedidos
Descripción:	El usuario obtiene la información de cada pedido generado.
Pasos de Ejecución	1. El cliente debe ingresar a la aplicación. 2. El usuario debe seleccionar la opción de "Cuenta" del menú inferior. 3. El cliente debe seleccionar la opción de "Mis pedidos". 4. El cliente observará los productos que ha comprado desde la interfaz del carrito.
Resultado obtenido:	El usuario logró verificar los productos comprados desde la aplicación.
Evaluación:	**APROBADO**

Fuente: Elaboración propia

3.1.2. Evaluación de calidad del software

La norma ISO 25010 es un estándar internacional para evaluar la calidad del software, norma que define un modelo de calidad aplicable a todo tipo de software; el modelo está compuesto de características y luego se subdivide en sub-características [43]. Es un estándar ampliamente conocido y utilizado en la aplicación de evaluaciones a productos de software.

Para esta evaluación se considera una calificación basada en la escala de Likert para interpretar el nivel de satisfacción sobre el cumplimiento de los objetivos que se estipulan en las sub-características de la métrica ISO 25010.

Tabla 51: Indicador de evaluación de calidad

Indicadores de evaluación	
Valoración	Interpretación de resultados
5	Excelente
4	Muy bueno
3	Bueno
2	Malo
1	Muy malo

Fuente: Elaboración propia

Tabla 52: Evaluación de calidad según ISO 25010

Característica	Sub-característica	Objetivo	1	2	3	4	5	Total
Funcionalidad	Completitud.	El sistema cubre todas las funciones para las que fue diseñado				X		4
	Corrección.	Las funciones del programa proporcionan el resultado correcto				X		4
	Pertinencia.	Las funciones del programa van de acuerdo a sus objetivos.				X		4
Eficiencia	Comportamiento.	La aplicación otorga buen tiempo de respuesta al ejecutar funciones					X	5
	Utilización de recursos.	La aplicación utiliza los recursos adecuados en el cumplimiento determinado de sus funciones				X		4
	Capacidad.	La aplicación cumple con los límites máximos para cada requisito propuesto.				X		4
Compatibilidad	Coexistencia.	La aplicación puede desplegarse en un sistema distinto al destinado originalmente.				X		4
	Interoperabilidad.	La aplicación opera con interactúa con módulos y API's externos con facilidad.				X		4
Usabilidad	Inteligibilidad.	La aplicación es de fácil entendimiento para su usuario.					X	5
	Aprendizaje.	El usuario puede aprender con facilidad el funcionamiento de la aplicación.					X	5
	Operabilidad.	La aplicación puede ser manejada por el usuario con facilidad.					X	5

	Protección frente a errores de usuario.	La aplicación captura y evita saltos de errores.				X		4
	Estética.	La interfaz de usuario es agradable a la vista.					X	5
	Accesibilidad.	La aplicación es altamente accesible para los usuarios en general.				X		4
Fiabilidad	Madurez.	La aplicación satisface las necesidades del usuario en condiciones normales.					X	5
	Disponibilidad.	La aplicación permanece operativa en cualquier momento.				X		4
	Tolerancia a fallos.	La aplicación opera con normalidad ante una falla de hardware o software.				X		4
	Capacidad de recuperación.	La aplicación recupera y restaura datos en caso de ocurrir un error o fallo.			X			3
Seguridad	Confidencialidad.	La aplicación protege contra el acceso de datos no autorizados.					X	5
	Integridad.	La aplicación previene el acceso o modificaciones no autorizados.				X		4
	No repudio.	La aplicación guarda acciones que puedan ser deshechas en un futuro				X		4
	Autenticidad.	La aplicación permite el acceso a usuarios autenticados				X		4
	Responsabilidad.	La aplicación rastrea y guarda las acciones realizadas.			X			3
Mantenibilidad	Modularidad.	La aplicación trabaja con sus módulos independientes uno de otros					X	5

	Reusabilidad.	Los componentes de la aplicación pueden ser usados en más de una vez				X		4
	Analizabilidad.	Es apreciable el cambio realizado en un módulo específico				X		4
	Capacidad de ser modificado.	La aplicación puede ser modificada de manera efectiva y eficiente.					X	5
	Capacidad de ser aprobado.	La aplicación puede ser sometida a pruebas.					X	5
Portabilidad	Adaptabilidad.	La aplicación se adapta eficientemente a diferentes dispositivos.					X	5
	Facilidad de instalación.	La aplicación se instala sin presentar problemas.					X	5
	Capacidad de ser reemplazado.	La aplicación puede ser reemplazada y/o actualizada.					X	5

Fuente: Elaboración propia

3.2 Resultados de la evaluación

La siguiente tabla representa un resumen de la matriz de evaluación de la calidad de software según la norma ISO/IEC 25010:

Tabla 53: Resumen de resultados de evaluación

Características	Parámetros	Puntaje	Promedio	Total
Funcionalidad	3	12	4	Muy bueno
Eficiencia	3	13	4.3	Muy bueno
Compatibilidad	2	8	4	Muy bueno
Usabilidad	6	28	4,67	Excelente
Fiabilidad	4	16	4	Muy bueno
Seguridad	5	20	4	Muy bueno
Mantenibilidad	5	23	4,6	Excelente
Portabilidad	3	15	5	Excelente
			34,57	Muy bueno

Fuente: Elaboración propia

El prototipo evaluado presenta mayor fortaleza con respecto a la portabilidad, mantenibilidad y usabilidad dado que es una aplicación nativa que puede desplegarse como multiplataforma con facilidad. Un promedio a la puntuación obtenida indica que la aplicación desarrollada respeta la normativa de calidad ISO/IEC 25010 en un 4,32/5.

4. Conclusiones y recomendaciones

Una vez finalizado el proceso de desarrollo de la aplicación móvil para la plataforma de realidad aumentada se concluye que:

- La elección de React Native como framework de desarrollo permitió programar la aplicación de manera nativa para dispositivos iOS, abriendo la posibilidad en un futuro de desplegarse para dispositivos Android.
- Se consiguió un desarrollo ágil a través de la aplicación de la metodología XP, debido a que sus lineamientos se enfocan en el desarrollo del prototipo y se acopla a las necesidades surgidas para este proyecto.
- La identificación de requerimientos funcionales y no funcionales ayudo al desarrollo de las funciones específicas y a establecer los límites de las operaciones que puede ejecutar.
- Se logró diseñar interfaces de usuario que permiten el manejo de la aplicación de manera sencilla, interactiva y que no abrumen al cliente; además de la definición del comportamiento de la aplicación gracias a la elaboración de diagramas de actividad y casos de uso por medio de herramientas CASE.
- Se evaluó satisfactoriamente el cumplimiento de las funciones de la aplicación mediante las pruebas unitarias sobre las historias de usuario definidas, además de que el uso de la ISO/IEC 25010 ayudó a elaborar una evaluación personalizada para garantizar el cumplimiento del estándar de calidad de la aplicación.

Considerando las dificultades, imprevistos y cantidad de procesos manejados a lo largo del desarrollo de este proyecto, se hacen las siguientes recomendaciones.

- Organizar la programación por pantallas, componentes y funciones ayuda a la reutilización de código de forma efectiva y a la corrección de errores que no afecte el funcionamiento de un componente separado.
- Considerar el uso de frameworks multiplataforma para el desarrollo de aplicaciones móviles a fin de evitar retrasos financieros y de tiempo al crear sus propias versiones para otros sistemas operativos.
- Utilizar herramientas de control de versiones para el código de la aplicación ayuda mucho en el manejo del flujo de trabajo, teniendo respaldos del código de la aplicación tras el desarrollo de cada función, módulo o corrección.
- Regirse a las métricas de un estándar de desarrollo o de calidad ayuda a elaborar y ejecutar pruebas de desarrollo y calidad que garanticen el funcionamiento correcto de la aplicación y satisfagan las expectativas de los usuarios.

Bibliografía

[1] Visual Studio Code, «Visual Studio Code,» Microsoft, [En línea]. Available: https://code.visualstudio.com/docs. [Último acceso: 11 Julio 2021].

[2] Android Studio, «Android Studio,» Google, [En línea]. Available: https://developer.android.com/studio/intro. [Último acceso: 11 Julio 2021].

[3] K. S. Duisebekova, D. K. Kozhamzharova, S. B. Rakhmetulayeva, F. A. Umarov y M. Z. Aitimov, «Development of an information-analytical system for the analysis and monitoring of climatic and ecological changes in the environment: Part 1,» *Procedia Computer Science,* vol. MCLXX, pp. 578-583, 2020.

[4] MDN contributors, «MDN Web Docs,» Mozilla, [En línea]. Available: https://developer.mozilla.org/es/docs/Web/JavaScript. [Último acceso: 11 Julio 2021].

[5] H. S. Krohn, «Programación de buscadores en JavaScript para diccionarios digitales,» *Cuadernos de Lingüística Hispánica,* nº 34, pp. 109-130, 2019.

[6] D. Johannes, F. Khomh y G. Antoniol, «A large-scale empirical study of code smells in JavaScript projects,» *Software Quality Journal,* vol. XXVII, p. 1271–1314, 2019.

[7] A. Makris, K. Tserpes, G. Spiliopoulos, D. Zissis y D. Anagnostopoulos, «MongoDB Vs PostgreSQL: A comparative study on performance aspects,» *GeoInformatica,* vol. XXV, p. 243–268, 2021.

[8] R. Čerešňák y K. Michal, «Comparison of query performance in relational a non-relation databases,» *Transportation Research Procedia,* vol. XL, pp. 170-177, 2019.

[9] J. W. Durán-Cazar, E. J. Tandazo-Gaona, M. R. Morales-Morales y S. Morales Cardoso, «Rendimiento de bases de datos columnares,» *Ingenius. Revista de Ciencia y Tecnología,* nº 22, pp. 47-58, 2019.

[10] K. A. Gomez Espinosa y A. Magbag, «A mobile application for the Liturgical planner-calendar of a Roman Catholic Diocese in the Philippines A mobile application for the Liturgical planner-calendar of a Roman Catholic Diocese in the Philippines,» *International Journal of Research Studies in Education,* vol. IX, nº 5, pp. 25-33, 2020.

[11] N. Hilger, «How Do Marketplaces Compete? The View From Stripe Connect,» *SSRN Electronic Journal,* pp. 1-31, 2018.

[12] M. Ciman y O. Gaggi, «An empirical analysis of energy consumption of cross-platform frameworks for mobile development,» *Pervasive and Mobile Computing,* vol. XXXIX, pp. 214-230, 2017.

[13] J. Londoño Gallego, S. Londoño Marín, C. López Romero, J. D. Vahos Montoya, L. Á. Escobar Castrillón y S. Rendón Pareja, «Desarrollo de un aplicativo móvil y web que calcule la huella de carbono en el sector educativo y transporte,» *Lámpsakos,* nº 23, pp. 45-55, 2020.

[14] A. Biørn-Hansen, T.-M. Grønli, G. Ghinea y S. Alouneh, «An Empirical Study of Cross-Platform Mobile Development in Industry,» *Wireless Communications and Mobile Computing,* vol. MMXIX, pp. 1-12, 2019.

[15] W. Danielsson, «React,» de *React Native application development*, Linköping, Linköpings universitet, 2016, p. 8.

[16] W. Danielsson, «React Native,» de *React Native application development*, Linköping, Linköpings universitet, 2019, p. 10.

[17] A. B. Tresan Andanaputra y H. Ham, «Mobile Based Application of Mosquito Larvae Checking Reports : Malaka Sari Village Case,» *Procedia Computer Science,* vol. CLXXIX, pp. 615-623, 2021.

[18] C. Rieger y T. A. Majchrzak, «Towards the definitive evaluation framework for cross-platform app development approaches,» *Journal of Systems and Software,* vol. CLIII, pp. 175-199, 2019.

[19] Expo, «Expo,» [En línea]. Available: https://docs.expo.dev/. [Último acceso: 4 Agosto 2021].

[20] J.-P. Sirois y Z. Hall, «Lodash,» [En línea]. Available: https://lodash.com/. [Último acceso: 22 Julio 2021].

[21] J. Palmer y I. White, «Formik,» [En línea]. Available: https://formik.org/docs/guides/validation. [Último acceso: 22 Julio 2021].

[22] Yup, «Yup,» GitHub, [En línea]. Available: https://github.com/jquense/yup. [Último acceso: 22 Julio 2021].

[23] W. S. El-Kassas, B. A. Abdullah, A. H. Yousef y A. M. Wahba, «Taxonomy of Cross-Platform Mobile Applications Development Approaches,» *Ain Shams Engineering Journal,* vol. XIII, pp. 163-190, 2017.

[24] S. M. Velasquez, D. E. Monsalve Sossa, M. E. Zapata, M. E. Gómez Adasme y J. P. Ríos, «Pruebas a aplicaciones móviles: avances y retos,» *Lámpsakos,* nº 21, pp. 39-50, Lámpsakos.

[25] J.-J. H. Hew, V.-H. Lee, K.-B. Ooi y J. Wei, «What catalyses mobile apps usage intention: an empirical analysis,» *Industrial Management & Data Systems,* vol. VII, pp. 1269-1291, 2015.

[26] A. Ahmad, K. Li, C. Feng, S. M. Asim, A. Yousif y S. Ge, «An Empirical Study of Investigating Mobile Applications Development Challenges,» *IEEE Access,* vol. VI, pp. 17711-17728, 2018.

[27] G. Hernández, Á. Martínez, R. Jiménez y F. Jiménez, «Métricas de productividad para equipo de trabajo de desarrollo ágil de software: una revisión sistemática,» *TecnoLógicas,* vol. XXII, pp. 63-81, 2019.

[28] J. C. Pereira y R. d. F. S. M. Russo, «Design Thinking Integrated in Agile Software Development: A Systematic Literature Review,» *Procedia Computer Science,* vol. CXXXVIII, pp. 775-782, 2018.

[29] J. Holvitie, S. A. Licorish, R. O. Spínola, S. Hyrynsalmi, S. G. MacDonell, T. S. Mendes, J. Buchan y V. Leppänen, «Technical debt and agile software development practices and processes: An industry practitioner survey,» *Information and Software Technology,* vol. XCVI, pp. 141-160, 2018.

[30] S. Abdalhamid y A. Mishra, «Adopting of Agile methods in Software Development Organizations: Systematic Mapping,» *TEM Journal,* vol. VI, nº 4, p. 817 – 825, 2017.

[31] D. Inupakutika, S. Kaghyan, D. Akopian, P. Chabela y A. G. Ramirez, «Facilitating the development of cross-platform mHealth applications for chronic supportive care and a case study,» *Journal of Biomedical Informatics,* vol. CV, pp. 1-13, 2020.

[32] A. Biørn-Hansen, C. Rieger, T.-M. Grønli, T. Majchrzak y G. Ghinea, «An empirical investigation of performance overhead in cross-platform mobile development frameworks,» *Empirical Software Engineering,* vol. XXV, p. 2997–3040, 2020.

[33] M. S. Chhonker, D. Verma y A. K. Kar, «Review of Technology Adoption frameworks in Mobile Commerce,» *Procedia Computer Science,* vol. CXXII, pp. 888-895, 2017.

[34] S. Dakduk, Z. Santalla-Banderali y J. R. Siqueira, «Acceptance of mobile commerce in low-income consumers: evidence from an emerging economy,» *Heliyon,* vol. VI, pp. 1-15, 2020.

[35] N. T. Chau y H. Deng, «Critical Determinants for Mobile Commerce Adoption in Vietnamese SMEs: A Conceptual Framework,» *Procedia Computer Science,* vol. CXXXVIII, pp. 433-440, 2018.

[36] D. Sánchez-Hernández, F. Lizano-Madriz y M. M. Sandoval-Carvajal, «Integración de pruebas remotas de usabilidad en Programación Extrema: revisión de literatura,» *Uniciencia,* vol. XXXIV, nº 1, pp. 20-31, 2020.

[37] J. Yang, X. L. Zhang y P. Su, «Deep-Learning-Based Agile Teaching Framework of Software Development Courses in Computer Science Education,» *Procedia Computer Science,* vol. CLIV, pp. 137-145, 2019.

[38] Y. Wautelet, S. Heng, S. Kiv y M. Kolp, «User-story driven development of multi-agent systems: A process fragment for agile methods,» *Computer Languages, Systems & Structures,* vol. L, pp. 159-176, 2017.

[39] M. Campo, A. Amandi y J. C. Biset, «A Software Architecture Perspective about Moodle Flexibility for Supporting Empirical Research of Teaching Theories,» *Education and Information Technologies,* vol. XXVI, nº 1, pp. 817-842, 2021.

[40] H. Aljumaily, D. Cuadra y D. F. Laefer, «An Empirical Study to Evaluate Students' Conceptual Modeling Skills Using UML,» *Computer Science Education,* vol. XXIX, nº 4, pp. 407-427, 2019.

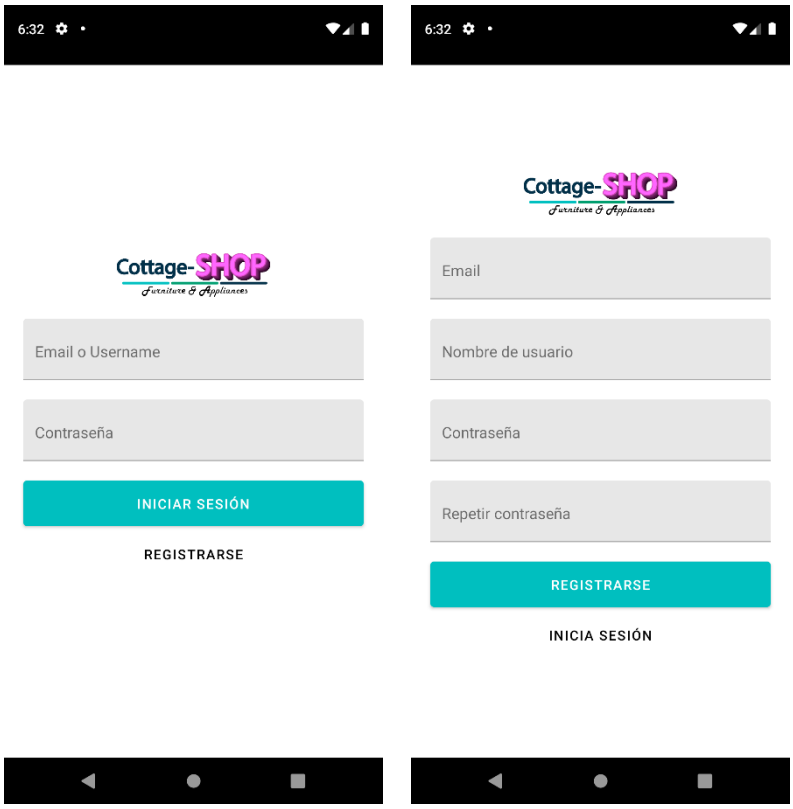
[41] V. Vachharajani y J. Pareek, «Framework to Approximate Label Matching for Automatic Assessment of Use-Case Diagram,» *International Journal of Distance Education Technologies,* vol. XVII, nº 3, pp. 75-95, 2019.

[42] F. Trautsch, S. Herbold y J. Grabowski, «Are unit and integration test definitions still valid for modern Java projects? An empirical study on open-source projects,» *The Journal of Systems and Software,* vol. CLIX, pp. 1-15, 2020.

[43] A. Krouska, C. Troussas y M. Virvou, «A literature review of Social Networking-based Learning Systems using a novel ISO-based framework,» *Intelligent Decision Technologies,* vol. XIII, nº 2, pp. 23-39, 2019.

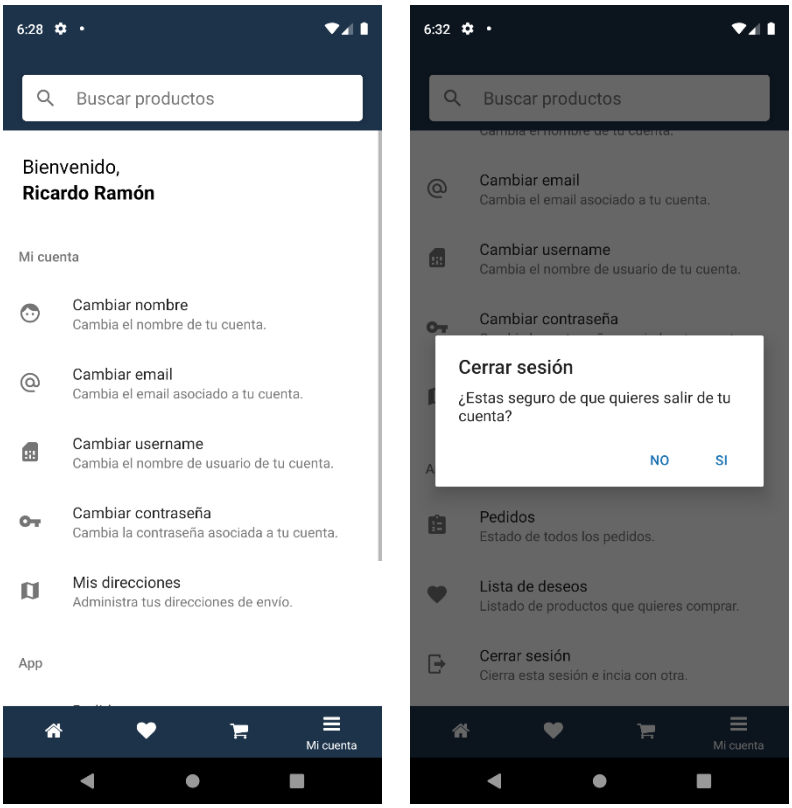
Anexos

Figura 81: Ejecución de loguin y registro



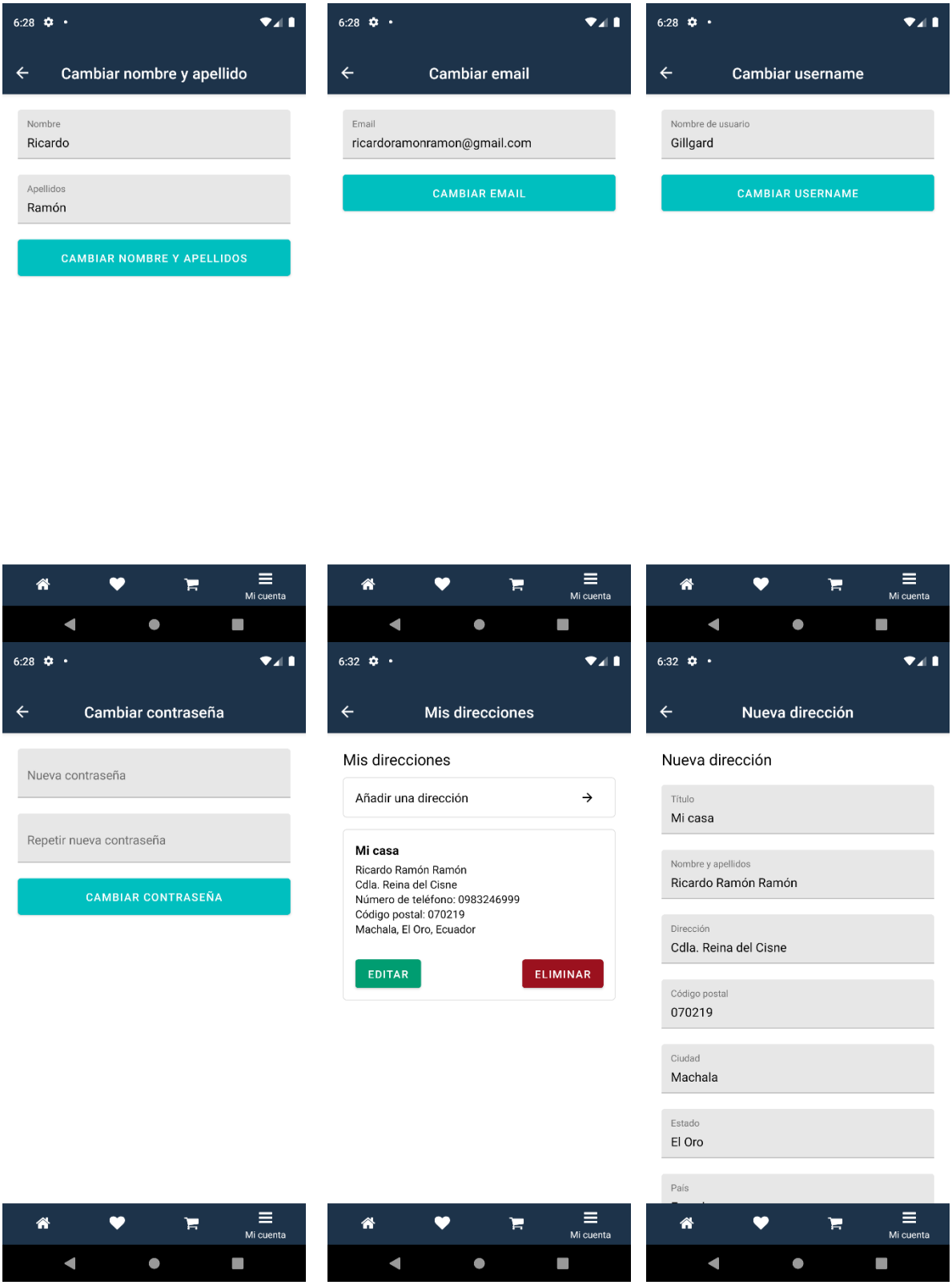
Fuente: Elaboración propia

Figura 82: Ejecución de cuenta y log-out



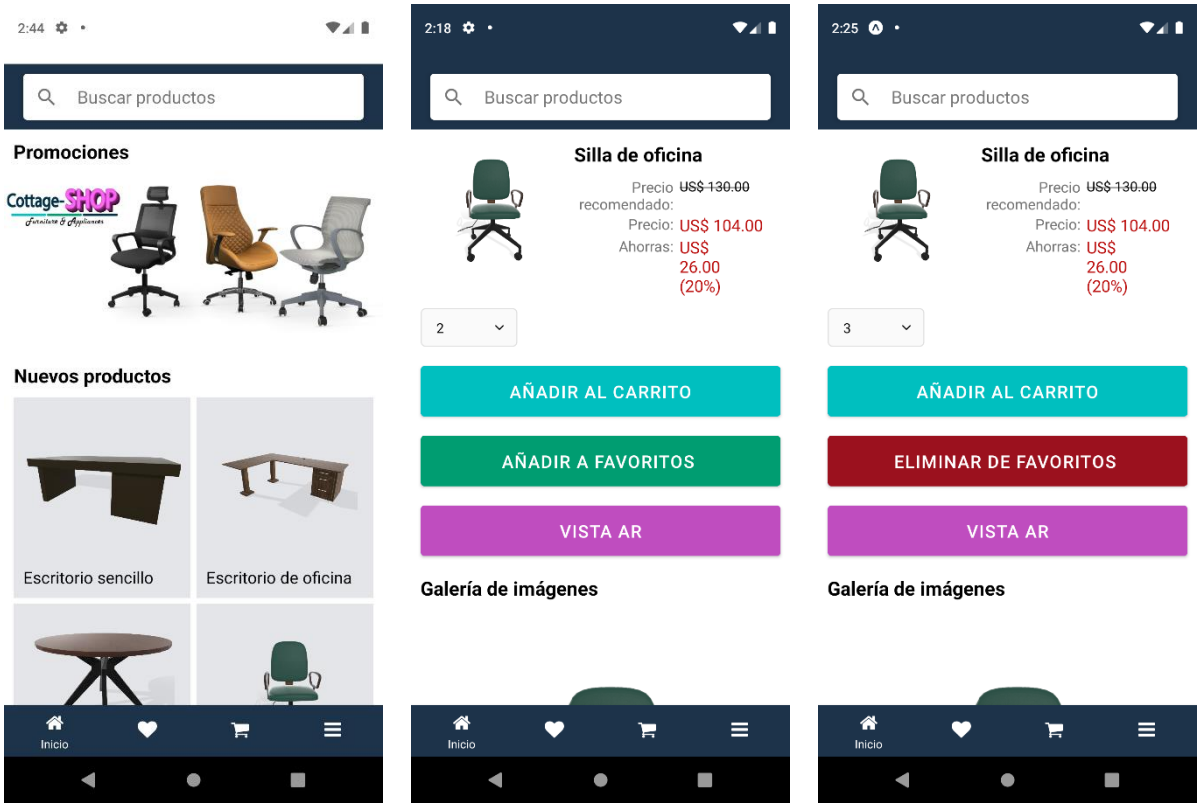
Fuente: Elaboración propia

Figura 83: Ejecución de ajustes de la cuenta



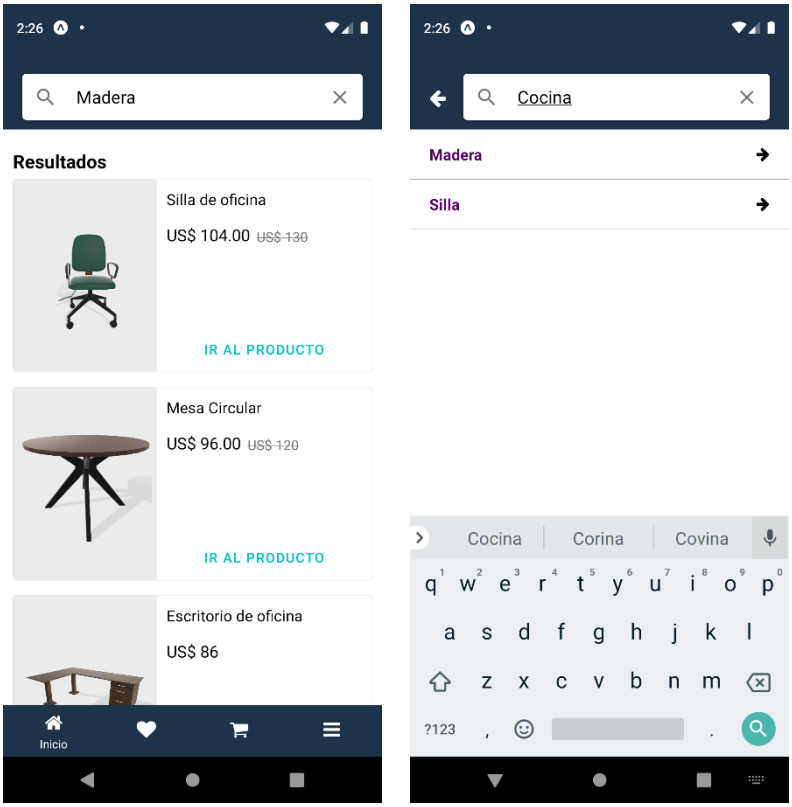
Fuente: Elaboración propia

Figura 84: Ejecución de ventana de productos



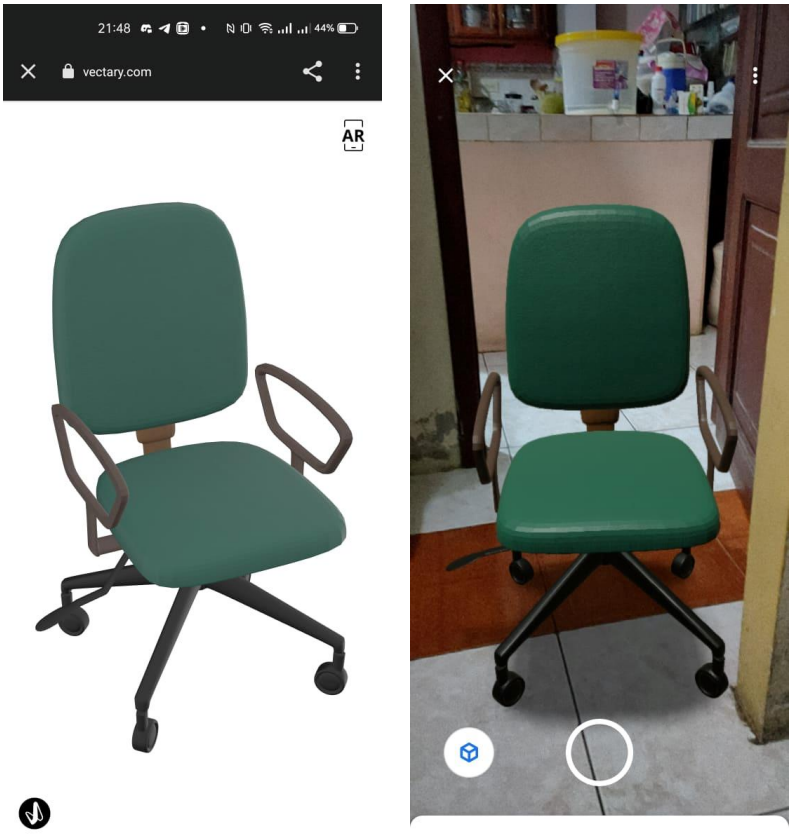
Fuente: Elaboración propia

Figura 85: Ejecución de búsqueda de productos



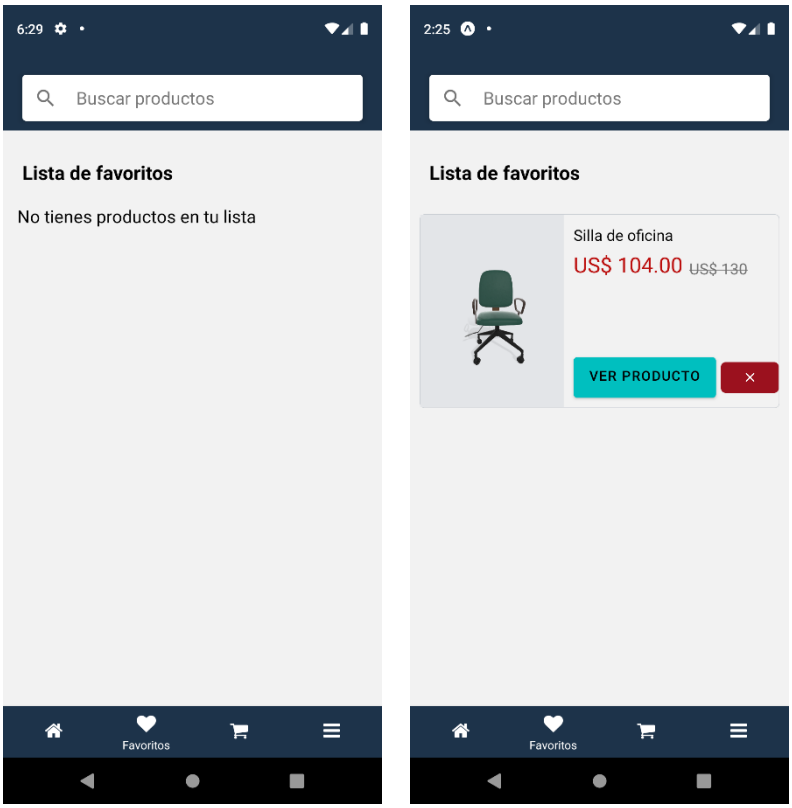
Fuente: Elaboración propia

Figura 86: Funcionamiento de vista AR



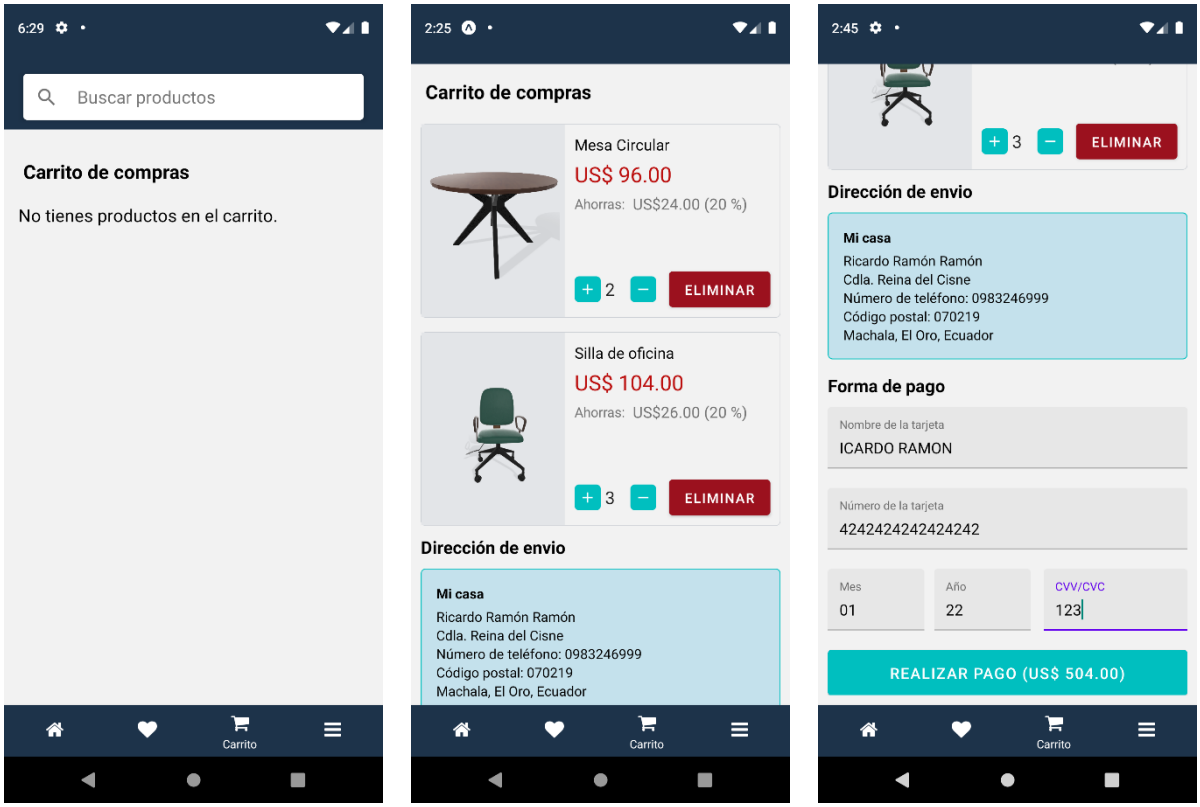
Fuente: Elaboración propia

Figura 87: Funcionamiento de lista de favoritos



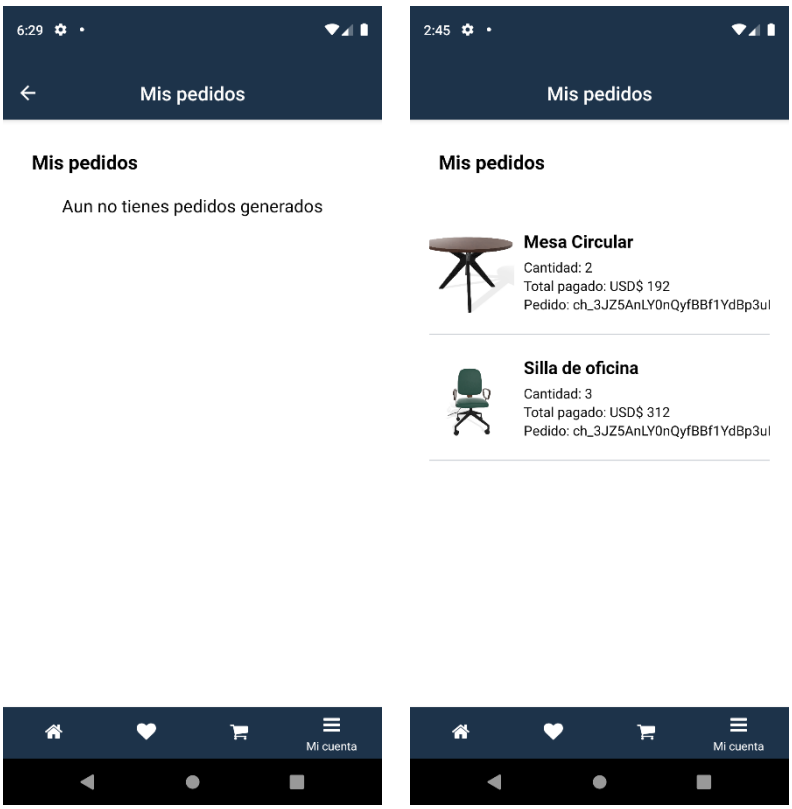
Fuente: Elaboración propia

Figura 88: Funcionamiento de carrito de compras



Fuente: Elaboración propia

Figura 89: Funcionamiento de lista de pedidos



Fuente: Elaboración propia