Prototipo de plataforma marketplace para la venta de productos agrícolas.

Alejandro Romero Serrano y Jorge Luis Sandoval Anaya

Trabajo de Grado para Optar al Título de Ingeniero de Sistemas

Emilio Justiniano Carcamo Troconis

Ingeniero de Sistemas

Universidad Industrial de Santander

Facultad de Ingenierías Fisicomecánicas

Escuela de Ingeniería de Sistemas

Ingeniería de Sistemas

Bucaramanga

2024

Dedicatoria de Jorge

A mi querida mamá, a mi amado hermano y a los increíbles amigos que he conocido a lo largo de este proceso,

Este proyecto no solo es un testimonio de mi dedicación, sino también del apoyo inquebrantable que he recibido de las personas más importantes en mi vida. Mamá, tu amor y sacrificio son la base de mis logros; hermano, tu aliento siempre ha sido mi inspiración.

A mis amigos, quienes han compartido risas, desafíos y triunfos a lo largo de este viaje, su amistad ha sido una luz brillante en los días más difíciles. Cada uno de ustedes ha dejado una huella imborrable en mi corazón, y este logro es también suyo.

Con gratitud infinita y amor,

Jorge

Dedicatoria de Alejandro

A mi amorosa madre Nubia, a mi disciplinado padre William Duván y a mi talentoso hermano William David, quienes han sido los pilares de mi vida y quienes con su esfuerzo, dedicación y apoyo me han ayudado a llegar hasta este punto. Por enriquecer mi vida, ser mi refugio y ser las personas que me han hecho sentir tener la mayor de las fortunas.

A cada uno de mis familiares: abuelos, tíos y primos que me han apoyado de múltiples formas, y que han sido fuente de fuerza para seguir adelante.

A cada uno de mis amigos y compañeros, con quienes compartí gran parte de la vida universitaria y con quienes conviví momentos especiales y difíciles.

Al comedor y bienestar estudiantil, donde estuve mis últimos dos años como auxiliar y donde fuí apoyado en situaciones de mi vida por muchas de las personas que las conforman.

Es a todos y cada uno de ustedes que dedico el esfuerzo empleado en este proyecto para optar al título de pregrado y el de todos los años de mi carrera en general.

Con amor y agradecimiento,

Alejandro

Agradecimientos

Agradecemos especialmente a nuestro profesor Emilio Justiniano Carcamo Troconis por haber sido la persona que decidió apoyarnos inicialmente a desarrollar este proyecto y quien por voluntad propia asumió el papel de director de trabajo de grado.

Al profesor José Geralbert Rubiano quien fue el calificador de nuestro plan de trabajo de grado y nos orientó sobre cómo documentar adecuadamente los elementos de nuestros escritos y nos ayudó a enfocarnos en aspectos fundamentales.

A la Escuela de Ingeniería de Sistemas y a cada uno de sus integrantes, por haber permitido que nuestro tema fuese aprobado y por aportarnos los conocimientos necesarios para lograr completarlo.

A la Universidad Industrial de Santander, por cada una de las experiencias vividas porque es por ella que hoy tenemos muchas de las personas y además muchas de las herramientas con las que llegamos hasta acá.

Tabla de Contenido

Pág.

Lista de Tablas

Pág.

Lista de Figuras

Pág.

Lista de Apéndices

“Los apéndices están adjuntos y puede visualizarlos en la base de datos de la biblioteca UIS”

Resumen

Título: Prototipo de plataforma marketplace para la venta de productos agrícolas.[1*]

Autor: Alejandro Romero Serrano y Jorge Luis Sandoval Anaya.[2*]

Palabras Clave: Agricultura, comercio electrónico, gestión agrícola.

Descripción: Este proyecto tiene como enfoque principal la creación de un prototipo de una plataforma de mercado que busca facilitar la venta de productos agrícolas. La iniciativa comienza como una respuesta a la emergente necesidad de conectar a los agricultores y compradores en un espacio digital. La plataforma desarrollada se concibe como un entorno de uso simple, que le permite a los agricultores presentar sus productos de forma efectiva y a los compradores acceder a estos fácilmente.

La plataforma no solo se basa en facilitar la comercialización de los productos, sino además, busca proporcionar a los agricultores algunas herramientas para mejorar la administración y gestión de sus inventarios. Además, para los compradores existirá la posibilidad del acceso a una variedad significativa de productos, lo que permite las comparaciones y fomenta la elección de productos que respalden las prácticas agrícolas responsables.

Por ende, este proyecto busca representar un paso más a la modernización del intercambio agrícola, al otorgar un aplicativo que no solo simplifica las transacciones de comercialización, sino que también promueve la responsabilidad en la elección de productos.

[1*] Trabajo de Grado

[2**] Facultad de Ingenierías Fisicomecánicas. Escuela de Ingeniería de Sistemas. Ingeniería de Sistemas. Director: Emilio Justiniano Carcamo Troconis. Ingeniero de Sistemas.

Abstract

Title: Prototype of a marketplace platform for the sale of agricultural products[3*]

Author(s): Alejandro Romero Serrano y Jorge Luis Sandoval Anaya.[4**]

Key Words: Agriculture, e-commerce, agricultural management.

Description: This project focuses on creating a prototype for a marketplace platform aimed at streamlining the sale of agricultural products. The initiative arises in response to the growing need to connect farmers and buyers in the digital space. The developed platform is envisioned as a user-friendly environment, allowing farmers to effectively showcase their products and enabling buyers to easily access them.

The platform not only aims to facilitate the marketing of products but also seeks to provide farmers with tools to enhance the management and administration of their inventories. Furthermore, for buyers, there will be the possibility of accessing a significant variety of products, encouraging comparisons and fostering the selection of items that support responsible agricultural practices.

Therefore, this project aims to represent a further step in the modernization of agricultural exchange by providing an application that not only simplifies marketing transactions but also promotes responsibility in product selection.

[3*] Degree Work

[4**] Faculty of Physicomechanical Engineering. School of Systems Engineering. Systems Engineering. Director: Emilio Justiniano Carcamo Troconis. Systems Engineer.

Introducción

Históricamente la actividad de comercialización agrícola se ha desarrollado principalmente de forma presencial, lo que ha conllevado algunos inconvenientes para los campesinos. La dependencia de los canales tradicionales generalmente provoca la necesidad de participación de intermediarios que adquieren los productos a bajo precio. Por otra parte, no hay visibilidad suficiente para los productos provenientes del campo en los mercados convencionales (como supermercados y plazas de mercado), dificultando que los agricultores destaquen las características de sus productos.

Considerando esto, las tecnologías digitales se presentan como una oportunidad para transformar la forma en que los campesinos llevan sus productos al mercado. La creación de un marketplace enfocado a la venta de productos agrícolas como hortalizas, frutas, entre otros, no solo está buscando reducir los riesgos arraigados a la comercialización tradicional, sino que busca darle a los agricultores un canal directo para la presentación y venta de sus productos. En este contexto, se inicia el proyecto ‘AgroSelling-Co’, un sistema de información que facilitará la venta de productos agrícolas.

1. Planteamiento y justificación del problema

En Colombia se cuenta con 22 millones de hectáreas disponibles para la agricultura, de las cuales solo se están aprovechando 5.3 millones para la producción relacionada a esta profesión. Esta generación de productos alimenticios es una pieza fundamental para el desarrollo y el sostenimiento del país, debido a que el 40% de la canasta familiar proviene de ella. Para el segundo semestre del año 2023, la importancia relativa de la agricultura en el PIB del país aumentó un 0.2% con respecto al mismo semestre del año pasado (2022) al pasar del 9.08% al 9.28%, indicando que el sector agrícola está al alza. (De Comercio De Bogotá, s. f.)

No obstante, los campesinos se enfrentan a las dificultades para realizar la comercialización de los productos cosechados, pues tienen la necesidad urgente de encontrar gente con la cual puedan comercializar sus productos para evitar que se estropeen y pierdan su valor, afectándolos económicamente.

Los productos generados de la agricultura pueden comercializarse entre la población mucho más fácil si usamos herramientas tecnológicas y tenemos en cuenta el uso de una plataforma Marketplace. La tecnología toma un papel importante ya que permite mejorar la comercialización agrícola en Colombia, siendo beneficiosa tanto para los campesinos como para los consumidores, haciendo que adquiera un gran valor el desarrollo de una plataforma que ofrezca, a los vendedores especialmente, una buena cantidad de soluciones prácticas a sus necesidades de comercialización de artículos agrícolas.

El desarrollo de esta plataforma involucra el uso de múltiples conocimientos adquiridos a lo largo de la carrera de Ingeniería de Sistemas, como lo son elegir las adecuadas metodologías de

desarrollo de software, la determinación de las etapas del ciclo del software y los lenguajes de programación que permitan generar el producto deseado exitosamente.

2. Objetivos

2.1 Objetivo General–

Desarrollar un prototipo de aplicación web y móvil que permita la comercialización de productos agrícolas.

2.2 Objetivos Específicos

Definir los requerimientos funcionales y no funcionales asociados a la venta de productos agrícolas a través de un marketplace.

Seleccionar las herramientas adecuadas para cumplir los requerimientos funcionales y no funcionales.

Realizar el diseño de la arquitectura del marketplace.

Evaluar el funcionamiento del prototipo a través de un plan de pruebas de calidad.

Implementar el prototipo de acuerdo a los requerimientos establecidos.

3. Marco de referencia

3.1 Estado del arte

3.1.1 Tierracol

Es un e-commerce creado en 2017, que se encarga de facilitar el comercio de productos agrícolas para las regiones alejadas de Colombia, lo que hace que los campesinos puedan mejorar su emprendimiento. Es una plataforma que permite facilitar la comunicación entre productores y consumidores para lograr el objetivo de la comercialización de manera sencilla. Fue premiada (por ocupar un tercer lugar entre 600) dentro del programa iNNpulsa Empodera, que nació como iniciativa de iNNpulsa Colombia junto al Ministerio de Comercio Industria y Turismo, que tenía como objetivo fortalecer a los emprendimientos más innovadores del país (Semana, 2022).

3.1.2 ComproAgro

ComproAgro es una iniciativa que surgió en 2014 como respuesta a una crisis económica causada por los bajos precios de los productos agrícolas. Inspirados por la visión de aliviar la carga financiera de sus familiares y vecinos, quienes se encontraban endeudados con entidades bancarias y otros prestamistas, decidieron fundar ComproAgro.com. El objetivo principal de esta plataforma es reducir la cadena de intermediación, permitiendo a los agricultores colombianos establecer contacto directo con compradores. La misión de ComproAgro es mejorar los ingresos y la calidad de vida de los agricultores, empoderandolos económicamente. (*ComproAgro - Quienes somos*, s. f.)

3.2 Tecnologías utilizadas

3.2.1 Angular

Es un framework de Javascript útil a la hora de desarrollar aplicaciones frontend modernas, que tienen gran complejidad. El tipo de ‘Apps’ construidas mediante Angular siguen el patrón SPA (Single Page Application) o PWA (Progressive Web App), los cuales brindan una experiencia de usuario más fluida y rápida. (*Angular*, s. f.)

3.2.2 Flutter

Flutter es un conjunto de herramientas de desarrollo e interfaz de usuario (SDK) desarrollado por Google que permite la creación de aplicaciones multiplataforma de alto rendimiento con un código único. En este momento, Flutter en su versión 3 nos permite desarrollar aplicaciones nativas para Android e iOS, aplicaciones web y aplicaciones de escritorio para Windows y macOS. (Cagigas, 2022)

3.2.3 Spring Boot

Una extensión del marco de Spring Framework que sigue el enfoque de "convención sobre configuración", Spring Boot ayuda a construir aplicaciones basadas en Spring de manera rápida y fácil. El objetivo principal de Spring Boot es que los desarrolladores puedan crear rápidamente aplicaciones basadas en Spring sin tener que escribir la misma configuración una y otra vez. (Reddy, 2017)

3.2.4 Git

Git es un sistema de control de versiones distribuido, lo que significa que un clon local del proyecto es un repositorio de control de versiones completo. Estos repositorios locales plenamente funcionales permiten trabajar sin conexión o de forma remota con facilidad. (Mijacobs, 2023)

3.2.5 Jira

Jira es una gran plataforma para la gestión de proyectos que le permite supervisar, administrar, rastrear y manejar cualquier tipo de proyecto en su propia empresa. (Rootstack, s. f.)

3.2.6 Amazon S3

Amazon Simple Storage Service (Amazon S3) es un proveedor líder de servicios de almacenamiento de objetos con escalabilidad, rendimiento, disponibilidad de datos y seguridad excepcionales. Permite a clientes de todos los tamaños e industrias almacenar y proteger cualquier cantidad o tipo de datos para casi cualquier uso de aplicación, incluidos lagos de datos, aplicaciones nativas de la nube y aplicaciones móviles. (*AWS | Almacenamiento de datos seguro en la nube (S3)*, s. f.)

3.2.7 PrimeNG

La biblioteca PrimeNG compatible con Angular es un marco de componentes de interfaz de usuario de código abierto. Los desarrolladores pueden utilizar una amplia gama de funciones, como tablas, menús y botones para crear aplicaciones web fáciles de usar, que incluyen elementos gráficos como calendarios. (*PrimeNG La Opción Para Principiantes En Angular*, 2023)

3.2.8 MySQL

MySQL es un sistema de gestión de bases de datos global que se origina en Oracle. Su función principal es almacenar datos para varios servicios web y se basa en álgebra relacional. (Equipo editorial de IONOS, 2023)

3.2.9 Spring cloud

Spring Cloud es un módulo de Spring que proporciona la función RAD (Rapid Application Development) al marco de Spring. Podemos desarrollar rápidamente la asignación basada en la nube con la ayuda de Spring Cloud Framework. (*Spring Cloud Tutorial - Javatpoint*, s. f.)

3.2.10 Java Persistence Api (JPA)

JPA es la solución estándar que Java ofrece para implementar un Framework de Mapeo Objeto-Relacional (ORM), el cual facilita la interacción con la base de datos a través de objetos. De esta manera, JPA se encarga de convertir los objetos Java en instrucciones para el Manejador de Base de Datos (MDB). (*Java Persistence API (JPA) - Oscar Blancarte - Software Architecture*, 2020)

3.2.11 Project Lombok

Project Lombok es una biblioteca de Java que proporciona numerosas funcionalidades. Para lograr esto, es necesario conectarlo a nuestro compilador, ya que su objetivo es simplificar el desarrollo de nuestro código evitando la necesidad de escribir ciertos métodos que son repetitivos y que realmente no aportan lógica al negocio. (Manjón, 2023)

3.2.12 ORM

Un ORM es una herramienta de programación que facilita la interacción con bases de datos relacionales, como SQL Server, Oracle, MySQL, entre otros. Su objetivo principal es simplificar y agilizar el desarrollo de aplicaciones al mapear las estructuras de la base de datos en una estructura lógica de entidades. (*¿Qué es un ORM?*, s. f.)

3.2.13 Mockito

Mockito es una biblioteca de Java utilizada para crear objetos simulados, muy utilizados en el desarrollo de pruebas unitarias en Test Driven Development, basada en EasyMock. El objetivo de Mockito es simplificar y resolver algunos de los problemas mencionados anteriormente. Aunque EasyMock y Mockito pueden realizar las mismas tareas, Mockito tiene una API más natural y práctica de utilizar. (*Mockito - Dos ideas.*, s. f.)

3.2.14 JUnit

JUnit es un marco de desarrollo de código abierto que se utiliza para automatizar las pruebas en proyectos de software, tanto pruebas unitarias como de integración. Este marco proporciona al usuario herramientas, clases y métodos que facilitan la tarea de realizar pruebas en el sistema, garantizando así su consistencia y funcionalidad. (JUNIT | *Marco de Desarrollo de la Junta de Andalucía*, s. f.).

3.2.15 Apache JMeter

JMeter es una herramienta de pruebas de rendimiento muy conocida y ampliamente utilizada en el ámbito de la programación. Esta herramienta, basada en Java y de código abierto,

permite realizar pruebas de carga en diferentes servicios web y de software, así como en aplicaciones web y otros servicios basados en protocolos como SOAP y REST. Su popularidad se debe a su eficacia y versatilidad en la ejecución de pruebas de rendimiento. (LoadView by Dotcom-Monitor, 2023)

3.2.16 Karma

Karma es una herramienta que se encarga de ejecutar tareas para nuestros tests. Utiliza un archivo de configuración para preparar el entorno de ejecución, los reportadores, el framework para realizar pruebas y el navegador, entre otras funcionalidades. (Pulido, 2019)

3.3 Marco teórico

3.3.1 Arquitectura de microservicios

La arquitectura de microservicios, también conocida como microservicios, se refiere a un enfoque arquitectónico para el desarrollo de aplicaciones. Este estilo permite la división de una aplicación extensa en componentes más pequeños e independientes, cada uno con su propia área de responsabilidad. En una aplicación basada en microservicios, se pueden invocar diversos microservicios internos para compilar una respuesta, con el objetivo de satisfacer una única solicitud del usuario. (*¿Qué es la arquitectura de microservicios? | Google Cloud | Google Cloud*, s. f.)

3.3.2 Marketplace

Un marketplace es una extensa plataforma donde diversas marcas, empresas o tiendas tienen la posibilidad de comercializar sus productos o servicios. En otras palabras, funciona como un centro comercial en línea. Este modelo de negocio no es novedoso, y el concepto en sí mismo implica que el marketplace actúa como un intermediario entre los vendedores y los clientes. (Palau, s. f.)

3.3.3 Agronomía

La agronomía es la disciplina científica dedicada a la utilización de los recursos naturales, como el suelo, el agua y las plantas, con el objetivo de generar alimentos para la población. Su principal enfoque radica en el cuidado y la preservación de los recursos de la tierra, asegurando beneficios para los seres vivos. En resumen, la agronomía se esfuerza por aumentar la productividad del suelo con el fin de producir alimentos de calidad para el consumo. Este proceso implica la adquisición de un amplio conocimiento en el campo, que se obtiene mediante la disciplina y el trabajo riguroso en la formación académica, como la licenciatura. (Euroinnova Business School, 2023)

3.3.4 Arquitectura limpia

La arquitectura limpia es una filosofía de diseño de software introducida por Robert C. Martin en 2017 en su libro homónimo. En esencia, se centra en la organización del código en componentes o módulos separados y en la forma en que estos elementos se interrelacionan. El principal objetivo de la Arquitectura Limpia es proporcionar una metodología para desarrollar código que funcione de manera óptima, tenga pocas dependencias y sea fácil de comprender y

mantener. Este enfoque busca reducir costos y maximizar la productividad del programador. (Calderón, 2022)

3.4 Fundamentos teóricos

3.4.1 JSON Web Token

JSON Web Token (JWT) es un estándar abierto, descrito en el RFC 7519, que establece un formato compacto e independiente para transmitir información de manera segura entre distintas partes mediante un objeto JSON. La información contenida en un JWT puede ser verificada y considerada confiable, ya que está firmada digitalmente. Estos tokens pueden ser firmados utilizando un secreto con el algoritmo HMAC (Hash-based Message Authentication Code) o un par de claves pública/privada mediante los algoritmos RSA o ECDSA (Elliptic Curve Digital Signature Algorithm).

3.4.2 Typescript

Typescript es un lenguaje de programación utilizado generalmente usado tanto para frontend, como para backend. Siendo desarrollado y mantenido por Microsoft, y siendo Opensource, extiende la sintaxis de Javascript, lo que implica que cualquier código de JavaScript funciona bien en Typescript. En cuanto a esta relación, se puede considerar a Typescript como un lenguaje más explícito, más simple, fiable e intercambiable. (Simões, 2021)

3.4.3 API REST

Una API REST, o API de RESTful, es una interfaz de programación de aplicaciones (API o API web) que sigue los principios de la arquitectura REST y permite la interacción con servicios web RESTful. Esta tecnología se basa en la transferencia de estado representacional (REST), creada por el informático Roy Fielding. Las APIs son conjuntos de definiciones y protocolos utilizados para diseñar e integrar el software de aplicaciones. Se consideran como contratos entre el proveedor de información y el usuario, estableciendo el contenido necesario para la solicitud (llamada) y la respuesta requerida por el productor. (*¿Qué es una API REST?*, s. f.)

3.4.4 Control de versiones

El control de versiones, también denominado "control de código fuente", es la práctica de supervisar y gestionar los cambios en el código de software. Los sistemas de control de versiones son herramientas de software que asisten a los equipos de desarrollo en la gestión de las modificaciones en el código fuente a lo largo del tiempo. (Atlassian, s. f.)

3.4.5 Pruebas unitarias

Las pruebas unitarias constituyen el proceso de evaluación de la unidad funcional de código más pequeña. Estas pruebas de software desempeñan un papel fundamental en asegurar la calidad del código y forman parte integral del desarrollo de software. Una práctica aconsejada en el desarrollo de software consiste en escribir el código en forma de unidades pequeñas y funcionales, seguido de la creación de una prueba unitaria para cada una de estas unidades de código. (*¿Qué son las pruebas unitarias?: Explicación de las pruebas unitarias en AWS*, s. f.)

4. Metodología

En este proyecto, la metodología a emplear será la Scrum, que es un marco de gestión de proyectos de metodología ágil que ayuda a los equipos a estructurar y gestionar el trabajo mediante un conjunto de valores, principios y prácticas (Atlassian, s. f.-b)**.** Detallaremos el alcance (funciones y entregables) que componen al proyecto y las etapas sobre las que se construirá el mismo.

4.1 Análisis del proyecto

Etapa útil para definir qué hará el software, qué se necesita y qué requerimientos se deben complacer. En este punto, se especifican:

- Requerimientos funcionales y no funcionales.
- Diagramas de uso.
- Historias de usuario.
- Modelo de datos.
- Tareas / actividades en el marco.

4.2 Alcance

El proyecto busca satisfacer algunos requerimientos relacionados a las actividades de comercialización de productos agrícolas, lo que hace importante definir el alcance para identificarlos. Los módulos a trabajar en el alcance son:

4.2.1 MarketPlace

Módulo encargado de la visualización y monitoreo de los productos agrícolas a comercializar. Se enfoca en el desarrollo de:

- Clasificación categórica de los productos: Permite a los compradores filtrar y ver productos por categoría, facilitando la búsqueda y selección de artículos específicos.
- Carrito de compras: Facilita a los usuarios la selección y compra de productos, permitiendo agregar artículos al carrito y realizar transacciones de manera eficiente.
- Registro de producto: Permite a los vendedores registrar nuevos productos en el sistema, incluyendo detalles como descripciones, imágenes y precios.
- Registro de usuario: Facilita el proceso de registro para nuevos usuarios, brindando acceso personalizado y la posibilidad de realizar compras.
- Registro de descuento: Permite a los vendedores registrar y gestionar descuentos en los productos que ofrece.

4.2.2 Módulo gestión financiera

Módulo específico destinado a asegurar el correcto funcionamiento del CRUD en las finanzas de la comercialización. El objetivo de este módulo es desarrollar:

- Visualización de las ventas.

4.3 Diseño del software

Se encarga de determinar cómo se va a construir el sistema y cómo se van a comunicar entre sí los componentes, especialmente para la visualización del software. Comprende varias tareas como:

- Diseño del prototipo en su componente frontend.

    - Selección de imágenes.
    - Creación de interfaz de usuario.
    - Interacción con la interfaz de usuario.
- Diseño de arquitectura a implementar.

4.4 Implementación del software

Fase de desarrollo con código. Antes de iniciarla, es importante identificar un entorno de desarrollo y un lenguaje de programación conocidos y aplicables para el trabajo que se desea construir. En este punto, se completa el backend y frontend, pero teniendo en cuenta algunas pautas como:

- Adecuar un repositorio para trabajo en equipo.
- Desarrollo de módulos en sus componentes backend y frontend
- Conexión de los componentes backend y frontend

Además de realizar estas tareas, se debe tener cuidado al subir y bajar los desarrollos correspondientes en el repositorio y manejar la lógica de manera que no existan riesgos en ejecución a la hora del manejo para el usuario. Por ejemplo: que un usuario guarde un producto al mismo tiempo que el otro, pero no se tome en cuenta el stock, no se agote (en realidad sí) y ambos guarden en el carrito para comprar, lo que generaría conflicto al momento que un usuario espere su producto, existiendo ausencia de estos en el inventario.

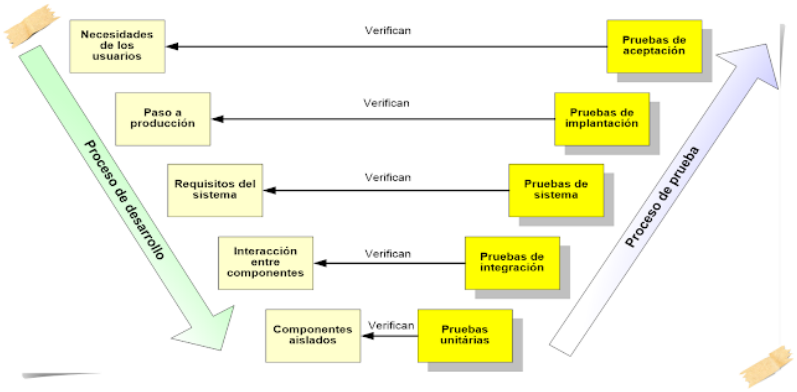
4.5 Pruebas

En esta fase se realiza un plan de pruebas para gestionar y revisar el correcto funcionamiento del software y el cumplimiento de lo planteado en las etapas de diseño y análisis. las pruebas que se aplicarán son:

- Pruebas unitarias.
- Prueba de integración.
- Pruebas de sistema.

Figura 1.

Proceso de pruebas.



Nota: Imagen representativa del proceso de pruebas. Recuperado de https://adsitestingsfw.blogspot.com/2016/02/proceso-de-pruebas.html

Nota: De todo de este proceso se llegará a las pruebas del sistema. Se tomarán una cantidad de personas al azar y mediante un cuestionario se realizará una prueba para evaluar si el prototipo es entendible.

5. Desarrollo del proyecto

Para el desarrollo de este proyecto, se optó por usar la metodología ágil Scrum, la cual se dividió en cuatro sprints (iteraciones) que serán detallados a lo largo del documento.

5.1 Fase de análisis

En la fase de análisis, se establecieron las bases fundamentales que sustentaron el proyecto, con el fin de alcanzar de manera efectiva los objetivos planteados inicialmente. El desarrollo del proyecto se inició mediante la definición de los requerimientos de software necesarios, entre ellos tenemos:

5.1.1 Requerimientos

En cuanto a los requerimientos es importante destacar que están divididos en dos categorías: funcionales, que se encargan de especificar las acciones del sistema y los no funcionales, que hacen lo mismo con el comportamiento.

5.1.1.1 Requerimientos funcionales

Tabla 1.

Requerimientos funcionales para iniciar sesión.

NOMBRE	Iniciar sesión
DESCRIPCIÓN	El sistema debe permitir que los usuarios se autentiquen mediante un proceso de inicio de sesión.
CRITERIOS DE ACEPTACIÓN	● Los usuarios deben tener un correo, nombre y contraseña únicos para ingresar al sistema. ● Los usuarios deben ingresar llenando todos los espacios: correo electrónico y contraseña. ● Una vez el usuario ingrese los datos correctos puede ingresar al sistema, en caso contrario se le notificará error. ● En caso de credenciales válidas, el sistema generará y retornará un token de sesión válido.
PRIORIDAD	Alta.

Tabla 2.

Requerimientos funcionales para registrar la cuenta de usuario.

NOMBRE	Registrar cuenta.
DESCRIPCIÓN	El sistema, en su aplicación web y móvil, debe permitir que los usuarios creen su cuenta personal mediante un proceso de registro.
CRITERIOS DE ACEPTACIÓN	● Los usuarios deben ingresar un correo, nombres y apellidos, contraseña y número celular para registrar su cuenta en el sistema. ● El sistema debe validar que el correo ingresado por el usuario no se encuentre registrado con anterioridad. ● En caso de que el correo ya haya sido registrado, se debe notificar con un mensaje de error. ● La contraseña del usuario se almacenará de manera segura utilizando un algoritmo de encriptación robusto.
PRIORIDAD	Alta.

Tabla 3.

Requerimientos funcionales para recuperar la cuenta de usuario.

NOMBRE	Recuperar cuenta.
DESCRIPCIÓN	El sistema debe permitirles a los usuarios recuperar su cuenta si olvida su contraseña.
CRITERIOS DE ACEPTACIÓN	● El usuario tiene que ingresar el correo de la cuenta en la que ha olvidado su contraseña. ● El sistema debe validar que el correo ingresado por el usuario se encuentre registrado con anterioridad. ● En caso de que el correo ya haya sido registrado, se enviará un correo electrónico con un código de seguridad que tiene 5 minutos de validez. ● El sistema debe redirigir a una vista en la que el usuario debe ingresar el código de seguridad y la nueva contraseña. ● El sistema verificará que el código de seguridad, que debe ingresar el usuario, aún válido y haya sido generado por el sistema para aprobar el ingreso de una nueva contraseña. En caso de ser válido el usuario podrá actualizar su contraseña, y en caso de no serlo, se le notificará con un mensaje de error.
PRIORIDAD	Alta.

Tabla 4.

Requerimientos funcionales para crear la empresa del usuario.

NOMBRE	Crear empresa.
DESCRIPCIÓN	El sistema, únicamente en su versión web, debe hacer que los usuarios nuevos creen la empresa a la que pertenecen para continuar con el funcionamiento de la aplicación.
CRITERIOS DE ACEPTACIÓN	● El sistema asegura que el usuario solo cree una empresa a la que pertenece.
PRIORIDAD	Alta.

Tabla 5.

Requerimientos funcionales para la gestión de cultivos agrícolas.

NOMBRE	Gestionar cultivos.
DESCRIPCIÓN	El sistema, en su aplicación web y móvil, debe permitirles a los administradores de la empresa crear, actualizar, eliminar y visualizar cultivos agrícolas.
CRITERIOS DE ACEPTACIÓN	● El sistema debe permitir que el administrador de la empresa ingrese un nombre y descripción del cultivo que desea crear. ● Los cultivos creados o actualizados por el administrador de la empresa deben tener un nombre único. En caso de no ser así, se mostrará un mensaje de error. ● Los campos del formulario que son obligatorios deben ser rellenados para completar el registro. Si no, no se permitirá y se notificará inmediatamente. ● El sistema debe permitir visualizar los cultivos asociados a su empresa. ● El sistema permite actualizar el nombre y descripción de sus cultivos. ● Un cultivo no puede ser eliminado si tiene otros registros relacionados como plagas o variables, además de mostrar su mensaje de error respectivo. En caso de no tener más registros asociados si se permite su eliminación.
PRIORIDAD	Alta.

Tabla 6.

Requerimientos funcionales para la gestión de productos con rol de administrador.

NOMBRE	Gestionar productos con rol de administrador.
DESCRIPCIÓN	El sistema, en su aplicación web y móvil, debe permitirles a los administradores de la empresa crear, actualizar, eliminar y visualizar productos agrícolas.
CRITERIOS DE ACEPTACIÓN	● El sistema debe permitir que el administrador de la empresa ingrese un nombre, resumen, descripción, categoría, precio, estado, cantidad y cultivo asociado del producto que desea crear. ● Los productos creados o actualizados por el administrador de la empresa deben tener un nombre único dentro de los demás de la misma empresa. En caso de no ser así, se mostrará un mensaje de error. ● Los campos del formulario que son obligatorios deben ser rellenados para completar el registro. Si no, no se permitirá y se notificará inmediatamente. ● El sistema debe permitir visualizar los productos asociados a su empresa. ● El sistema permite actualizar el nombre, descripción, resumen, categoría, precio, estado, cantidad y descuentos que hayan sido creados previamente.
PRIORIDAD	Alta.

Tabla 7.

Requerimientos funcionales para la gestión de descuentos.

NOMBRE	Gestionar descuentos con rol de administrador.
DESCRIPCIÓN	El sistema, en su aplicación web y móvil, debe permitirles a los administradores de la empresa crear, actualizar, eliminar y visualizar descuentos.
CRITERIOS DE ACEPTACIÓN	● El sistema debe permitir que el administrador de la empresa ingrese un valor, tipo, fecha de inicio y fecha de final del descuento. ● Los descuentos creados o actualizados por el administrador de la empresa deben tener un valor único dentro de los demás de la misma empresa. En caso de no ser así, se mostrará un mensaje de error. ● Los campos del formulario que son obligatorios deben ser rellenados para completar el registro. Si no, no se permitirá y se notificará inmediatamente. ● El sistema debe permitir visualizar los descuentos asociados a su empresa. ● El sistema permite actualizar el valor, tipo y fechas de los descuentos que hayan sido creados previamente.
PRIORIDAD	Alta.

Tabla 8.

Requerimientos funcionales para la gestión de envíos.

NOMBRE	Gestionar envíos con rol de administrador.
DESCRIPCIÓN	El sistema, en su aplicación web y móvil, debe permitirles a los administradores de la empresa crear, actualizar, eliminar y visualizar envíos.
CRITERIOS DE ACEPTACIÓN	● El sistema debe permitir que el administrador de la empresa ingrese un departamento, una ciudad, un nombre, días de demora para entregar y un precio. ● Los envíos creados o actualizados por el administrador de la empresa deben tener un nombre único dentro de los demás de la misma empresa. En caso de no ser así, se mostrará un mensaje de error. ● Los campos del formulario que son obligatorios deben ser rellenados para completar el registro. Si no, no se permitirá y se notificará inmediatamente. ● El sistema debe permitir visualizar los envíos asociados a su empresa. ● El sistema permite actualizar el nombre, ciudad, departamento, días de demora, y el precio de los envíos que hayan sido creados previamente.
PRIORIDAD	Alta.

Tabla 9.

Requerimientos funcionales para la visualización de pedidos.

NOMBRE	Visualizar los pedidos con rol de administrador.
DESCRIPCIÓN	El sistema, en su aplicación web y móvil, debe permitirles a los administradores de la empresa visualizar pedidos.
CRITERIOS DE ACEPTACIÓN	● El sistema debe permitirle al administrador visualizar todos los pedidos que han sido realizados. ● El sistema le debe permitir al administrador cambiar el estado del pedido a despacho. ● El sistema debe permitir que el administrador ingrese nombre y teléfono del domiciliario.
PRIORIDAD	Alta.

Tabla 10.

Requerimientos funcionales para la visualización de despachos.

NOMBRE	Visualizar los despachos con rol de administrador.
DESCRIPCIÓN	El sistema, en su aplicación web y móvil, debe permitir a los administradores de la empresa visualizar despachos.
CRITERIOS DE ACEPTACIÓN	● El sistema debe permitirle al administrador visualizar todos los despachos que han sido realizados. ● El sistema debe permitirle al administrador la opción de verificación de entrega del despacho.
PRIORIDAD	Alta.

Tabla 11.

Requerimientos funcionales para la visualización del historial de pedidos.

NOMBRE	Visualizar el historial de pedidos con rol de administrador.
DESCRIPCIÓN	El sistema debe permitirles a los administradores de la empresa visualizar pedidos.
CRITERIOS DE ACEPTACIÓN	● El sistema permitirá ver todo el historial de pedidos que se hayan hecho a la tienda desde que ha sido creada.
PRIORIDAD	Media.

Tabla 12.

Requerimientos funcionales para la visualización de los productos del marketplace.

NOMBRE	Visualizar los productos en el marketplace.
DESCRIPCIÓN	El sistema, únicamente en el aplicativo web, debe permitirles a todos los usuarios visualizar los productos en el marketplace.
CRITERIOS DE ACEPTACIÓN	● El sistema mostrará los productos agregados la última semana de manera aleatoria. ● El sistema mostrará los productos con descuentos de forma aleatoria. ● El sistema mostrará 10 productos de forma aleatoria.
PRIORIDAD	Media.

Tabla 13.

Requerimientos funcionales para el filtro por categoría de los productos del marketplace.

NOMBRE	Filtrar los productos en el marketplace por categoría.
DESCRIPCIÓN	El sistema, únicamente en el aplicativo web, debe permitirles a todos los usuarios filtrar los productos en el marketplace a dependencia de la categoría.
CRITERIOS DE ACEPTACIÓN	● El sistema mostrará los productos asociados a cierta categoría. ● Dentro del filtro por categoría, también se permite filtrar el orden por precio y nombre del producto. ● El sistema otorga visualización de las categorías existentes en el momento.
PRIORIDAD	Media.

Tabla 14.

Requerimientos funcionales para el filtro por empresa de los productos del marketplace.

NOMBRE	Filtrar los productos en el marketplace por empresa.
DESCRIPCIÓN	El sistema, únicamente en el aplicativo web, debe permitirles a todos los usuarios filtrar los productos en el marketplace a dependencia de la empresa a la que pertenece.
CRITERIOS DE ACEPTACIÓN	● El sistema mostrará los productos asociados a cierta empresa. ● Dentro del filtro por categoría, también se permite filtrar el orden por precio y nombre del producto. ● El sistema otorga visualización de las empresas existentes en el momento.
PRIORIDAD	Media.

Tabla 15.

Requerimientos funcionales para añadir al carrito.

NOMBRE	Añadir al carrito.
DESCRIPCIÓN	El sistema, únicamente en el aplicativo web, debe permitirles a todos los usuarios agregar sus productos seleccionados en el marketplace a un carrito de pago.
CRITERIOS DE ACEPTACIÓN	● El sistema debe avisar si el producto se pudo agregar al carrito o hubo error. ● En caso de agregarse, este debe aparecer en el carrito personalizado del usuario con su stock y su precio particular. ● El sistema debe mostrar el valor del subtotal de todos los productos agregados. ● El carrito debe permitir al usuario dirigirse a una pantalla de compra en la que el usuario deberá completar un formulario para empezar el despacho.
PRIORIDAD	Media.

Tabla 16.

Requerimientos funcionales para proceder al pago.

NOMBRE	Proceder al pago.
DESCRIPCIÓN	El sistema, únicamente en el aplicativo web, debe permitirles a todos los usuarios (después de añadir productos a su carrito) realizar el pago de su pedido.
CRITERIOS DE ACEPTACIÓN	● El sistema debe pedir agregar una dirección de envío y calcular el costo del pedido antes de proceder con el pago. ● El sistema debe verificar que no haya errores con los productos a pagar al calcular al pedido de pago, como lo pueden ser una dirección de entrega del pedido, disponibilidad del producto o algún valor relacionado. ● En caso de cumplir los últimos requisitos, el sistema debe permitir acceder a los métodos de pago para continuar la transacción.
PRIORIDAD	Alta.

Tabla 17.

Requerimientos funcionales para administrar el perfil del cliente.

NOMBRE	Administrar perfil de cliente.
DESCRIPCIÓN	El sistema, únicamente en el aplicativo web, debe permitirles a todos los clientes administrar los detalles de su cuenta.
CRITERIOS DE ACEPTACIÓN	● El sistema debe permitir visualizar los pedidos hechos por el cliente, mostrando su fecha, método de pago, valor y los productos encargados. ● El sistema debe permitir a los clientes actualizar sus datos: nombres, apellidos, correo, teléfono y contraseña. ● El sistema debe permitir cerrar la sesión del usuario y salir nuevamente a la página de ingreso.
PRIORIDAD	Alta.

5.1.1.2 Requerimientos NO funcionales

Tabla 18.

Requerimientos no funcionales base aplicados en el proyecto y su descripción.

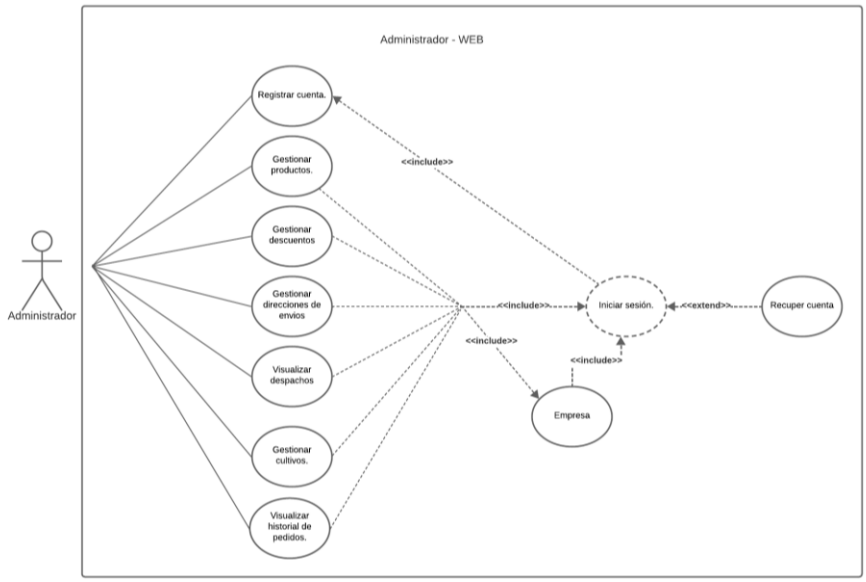
Requerimiento NO funcional	Descripción
1. Usabilidad	La aplicación debe tener una interfaz intuitiva y fácil de usar.
2. Compatibilidad.	La aplicación debe ser accesible de forma web y móvil.
3. Mantenibilidad	Debe permitirse realizar correcciones de errores y mejora de módulos sin mayor dificultad.
4. Seguridad	Los usuarios solo pueden acceder a ciertas funciones a dependencia de los roles.

5.1.2 Diagramas de uso

5.1.2.1 Diagrama de uso de administrador en web

Figura 2.

Diagrama de uso de administrador en web.

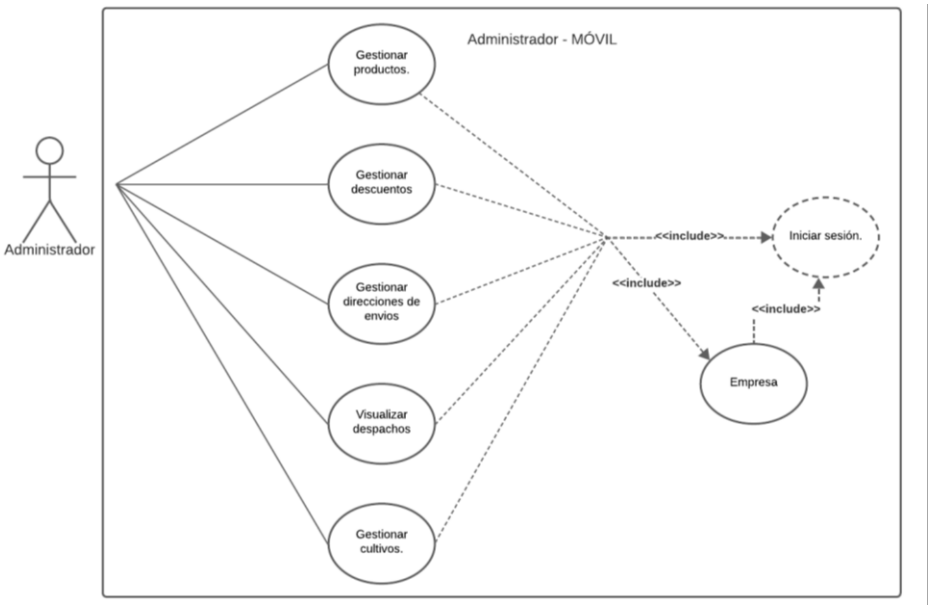


Nota: *Romero, A., & Sandoval, J.* (2024). Diseño propio.

5.1.2.2 Diagrama de uso de administrador en móvil

Figura 3.

Diagrama de uso de administrador en móvil.

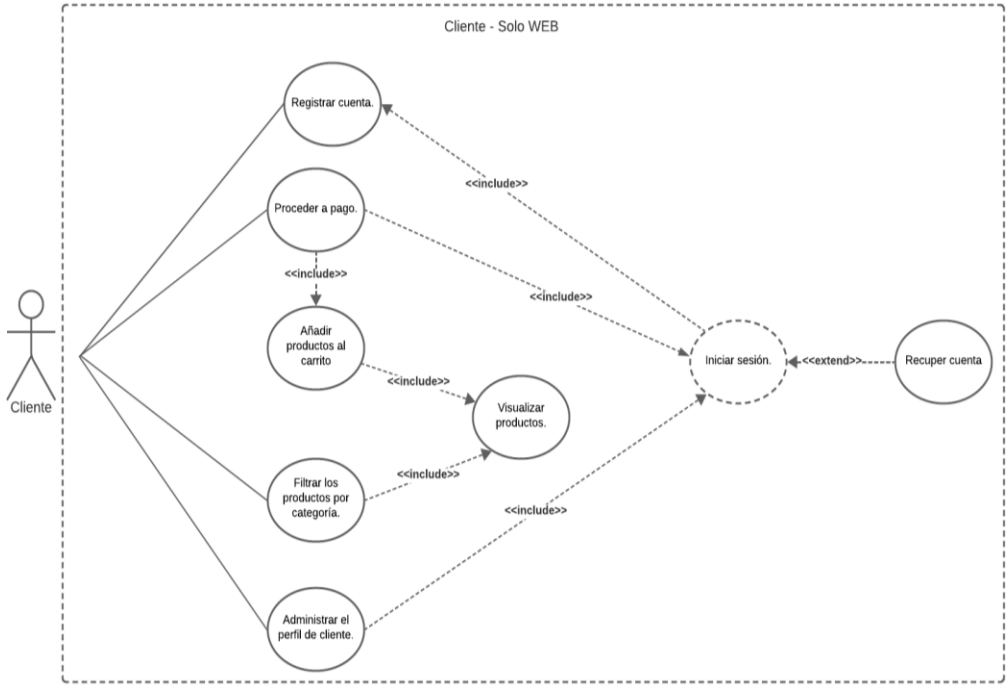


Nota: *Romero, A., & Sandoval, J.* (2024). Diseño propio.

5.1.2.3 Diagrama de uso de cliente (solo en web)

Figura 4.

Diagrama de uso de cliente.



Nota: *Romero, A., & Sandoval, J.* (2024). Diseño propio.

Considerando que la misión de los diagramas de casos de uso es representar el alcance y los objetivos de las interacciones entre el actor correspondiente y el sistema en general, es evidente que el administrador, especialmente en la sección del aplicativo web, lleva a cabo la mayoría de sus tareas dependiendo de la empresa a la que se vincula al iniciar sesión por primera vez en su cuenta web. Estas acciones están meticulosamente diseñadas para cumplir con el objetivo de lograr una implementación exitosa de un marketplace.

5.1.3 Historias de usuario

Las historias de usuario son narrativas concisas que describen las interacciones entre el usuario y el sistema. Estas historias están directamente vinculadas a las funcionalidades del sistema y desempeñan un papel crucial al proporcionar una visión clara de lo que se pretende implementar. A continuación, se presentan las historias de usuario más significativas.

Tabla 19.

Historias de usuario existentes en el desarrollo del software.

HISTORIA DE USUARIO 1	
Descripción: Autentificar cuenta.	
COMO	Administrador
QUIERO	Registrar cuenta e iniciar sesión en el sistema.
PARA	Acceder a las funcionalidades de creación de cultivos y productos.
HISTORIA DE USUARIO 2	
Descripción: Autentificar cuenta.	
COMO	Cliente
QUIERO	Registrar cuenta e iniciar sesión en el sistema.
PARA	Ingresar a las funcionalidades de visualización y compra de los productos presentes en el sistema.
HISTORIA DE USUARIO 3	
Descripción: Gestión de cultivos.	
COMO	Administrador
QUIERO	Gestionar los cultivos de mi empresa.
PARA	Visualizar el histórico de los cultivos que maneja mi empresa y gestionar sus registros.
HISTORIA DE USUARIO 4	

Descripción: Gestión de productos.	
COMO	Administrador
QUIERO	Gestionar los productos de mi empresa.
PARA	Visualizar el histórico de los productos que maneja mi empresa y gestionar sus registros.
HISTORIA DE USUARIO 5	
Descripción: Gestión de envíos.	
COMO	Administrador
QUIERO	Gestionar los envíos de mi empresa.
PARA	Visualizar el histórico de los envíos que ha realizado mi empresa y gestionar sus registros.
HISTORIA DE USUARIO 6	
Descripción: Gestión de descuentos.	
COMO	Administrador
QUIERO	Gestionar los descuentos de los productos de mi empresa.
PARA	Visualizar el histórico de los descuentos que tienen los productos de mi empresa y gestionar sus registros.
HISTORIA DE USUARIO 7	
Descripción: Visualización de pedidos	
COMO	Administrador
QUIERO	Visualizar los pedidos de mi empresa.
PARA	Visualizar el histórico de los pedidos que han realizado a mi empresa.
HISTORIA DE USUARIO 8	
Descripción: Visualización de productos.	
COMO	Cliente.
QUIERO	Visualizar todos los productos a disposición.
PARA	Poder filtrarlos y/o agregarlos a mi carrito de compras.

HISTORIA DE USUARIO 9	
Descripción: Agregar al carrito.	
COMO	Cliente.
QUIERO	Agregar todos los productos que pueda visualizar en el sistema al carrito de compras.
PARA	Poder continuar con el proceso de pago y recibir el producto.
HISTORIA DE USUARIO 10	
Descripción: Proceder al pago.	
COMO	Cliente.
QUIERO	Proceder al pago de los productos agregados a mi carrito de compras.
PARA	Conseguir los productos deseados.
HISTORIA DE USUARIO 11	
Descripción: Filtrar los productos.	
COMO	Cliente.
QUIERO	Filtrar los productos por categorías, precio y empresas.
PARA	Visualizar únicamente los productos a los que desee comprar, evitando a los que no.
HISTORIA DE USUARIO 12	
Descripción: Administrar el perfil.	
COMO	Cliente.
QUIERO	Administrar los datos del perfil de mi cuenta.
PARA	Controlar los datos de autentificación y de identificación.

Nota: *Romero, A., & Sandoval, J.* (2024). Diseño propio.

5.1.4 Tareas en el marco

En esta sección mostraremos la planificación de tareas dentro del marco de trabajo ágil SCRUM. Se llevaron cuatro sprints, durante los cuales definimos las tareas y estimamos sus esfuerzos respectivos. En seguida detallaremos la planificación SCRUM:

Tabla 20.

Roles de participantes Scrum.

NOMBRE	**ROL**
Emilio Justiniano Carcamo Troconis	Product owner, Scrum máster.
Jorge Luis Sandoval Anaya	Miembro del equipo de desarrollo.
Alejandro Romero Serrano	Miembro del equipo de desarrollo.

Nota: *Romero, A., & Sandoval, J.* (2024). Diseño propio.

5.1.4.1 Backlog. Se construyó un backlog que englobó todas las tareas a desarrollar, abarcando las funcionalidades planificadas para la implementación en el sistema.

5.1.4.1.1 Sprints. Durante el desarrollo del proyecto, se llevaron a cabo cuatro sprints, cada uno enfocado en una fase específica del ciclo de vida del software. En el Sprint 1, que abarcó del 10 de septiembre al 10 de octubre, se realizaron 9 tareas relacionadas con la fase de planeación, con un esfuerzo total de 32 puntos.

El Sprint 2, que se extendió desde el 11 de octubre hasta el 10 de noviembre, estuvo centrado en la fase de diseño. En este periodo, se completaron 4 tareas, con un esfuerzo acumulado de 42 puntos.

La fase de implementación se abordó en el Sprint 3, que tuvo lugar del 11 de noviembre al 27 de diciembre. En este sprint se llevaron a cabo 17 tareas, siendo la fase más extensa en términos de esfuerzo, con un total de 57 puntos.

El cuarto sprint, del 28 de diciembre al 10 de enero, se enfocó en la fase de pruebas. Durante este periodo, se realizaron 6 tareas con un esfuerzo total de 14 puntos.

Cada fase concluyó con una revisión coordinada con el Scrum Máster para evaluar el progreso del producto. Estas reuniones no solo se centraron en el estado actual del producto, sino que también brindaron un espacio colaborativo para analizar y proponer posibles ajustes en el desarrollo del sistema.

5.2 Fase de diseño

En esta fase, se presenta una visión del diseño del sistema, tomando en cuenta aspectos clave que incluyen la elección de lenguajes y frameworks, la arquitectura de microservicios para el backend, la estructura de directorios en el backend, frontend y aplicación móvil, así como el diseño detallado de la base de datos.

Aquí es donde diseñamos el cómo nuestro sistema tomará forma, asegurando no solo su funcionalidad, sino también su estética y eficiencia.

5.2.1 Elección de imágenes

Antes de analizar lo anteriormente mencionado, vamos a visualizar cada una de las imágenes más importantes que hacen parte del desarrollo del prototipo. Empezamos con el desarrollo del logo, que tiene una figura de una casa de compra y venta de productos agrícolas con el título de nuestro proyecto: ‘AgroSellingCO’, que fue diseñado por nosotros mismos para identificar a nuestra aplicación web.

Figura 5.

Logo del prototipo.

Nota: *Romero, A., & Sandoval, J.* (2024). Diseño propio.

En cuanto al inicio de sesión encontramos otro logo que sirve para identificar nuestro aplicativo en su modelo de autenticación. Esta figura representa la relación con la agricultura que tiene nuestro aplicativo.

Figura 6.

Icono de inicio de sesión.

Nota: Figura de mano con planta que representa agricultura. Recuperado de https://es.dreamstime.com/

Otra imagen importante seleccionada es la que encontramos al ingresar a la página de inicio de nuestro marketplace en el aplicativo web. Nuevamente, es una imagen relacionada con la agricultura, pero en este caso, más relacionado a la clase de productos que se pueden comercializar.

Figura 7.

Logo de página de inicio al marketplace.

Nota: Imagen de persona sosteniendo un ejemplo de producto comercializable. Recuperado de https://www.caaarem.mx/

La última imagen, pero no menos importante, es el fondo que tenemos en la autenticación, pero esta vez de nuestro aplicativo móvil, donde nuevamente se apela a la relación de la agricultura con el proyecto.

Figura 8.

Fondo de inicio de sesión en el aplicativo móvil.

Nota: Imagen de persona sosteniendo una bolsa con ejemplo de productos comercializables. Recuperado de https://www.amazon.com/

5.2.2 Diseño de la base de datos.

5.2.2.1 Elección de gestor de base de datos. En el proceso de selección del gestor de la base de datos para este proyecto de marketplace evaluamos diversas opciones teniendo en cuenta criterios fundamentales como el rendimiento, modelo de datos, escalabilidad, consistencia y el costo. A continuación, se presenta una tabla que resume las opciones disponibles, destacando sus respectivas características en relación con los criterios mencionados.

Tabla 21.

Bases de datos con parámetros de elección.

Base de datos	Rendimiento	Modelo de datos	Escalabilidad	Consistencia	Costo
MySQL	Sólido	Modelo relacional	Vertical y horizontal	Alta	Bajo
MariaDB	Sólido	Modelo relacional	Vertical y horizontal	Estándar	Bajo
PostgreSQL	Robusto	Modelo relacional avanzado	Vertical y horizontal	Alta	Bajo
Oracle	Robusto	Modelo relacional avanzado	Vertical y horizontal	Alta	Alto

Nota: *Romero, A., & Sandoval, J.* (2024). Diseño propio.

La elección de MySQL como la base de datos para este proyecto no solo se basa en las características técnicas identificadas anteriormente, sino también en el conocimiento previo que el equipo de trabajo posee sobre esta plataforma.

Rendimiento: En un contexto como el de este proyecto, en el que se trabajan consultas a tablas de forma básica, MySQL otorga un rendimiento muy eficiente.

Modelo de datos: La estructura y las relaciones entre las tablas de una base de datos relacional es fácil de entender y eso lo hace conveniente para el desarrollo del proyecto.

Escalabilidad: Para la escalabilidad, no hay mucha importancia pues la escalabilidad no suele ser una preocupación en un prototipo, pero aun siéndolo, con MySql se puede escalar horizontal y verticalmente.

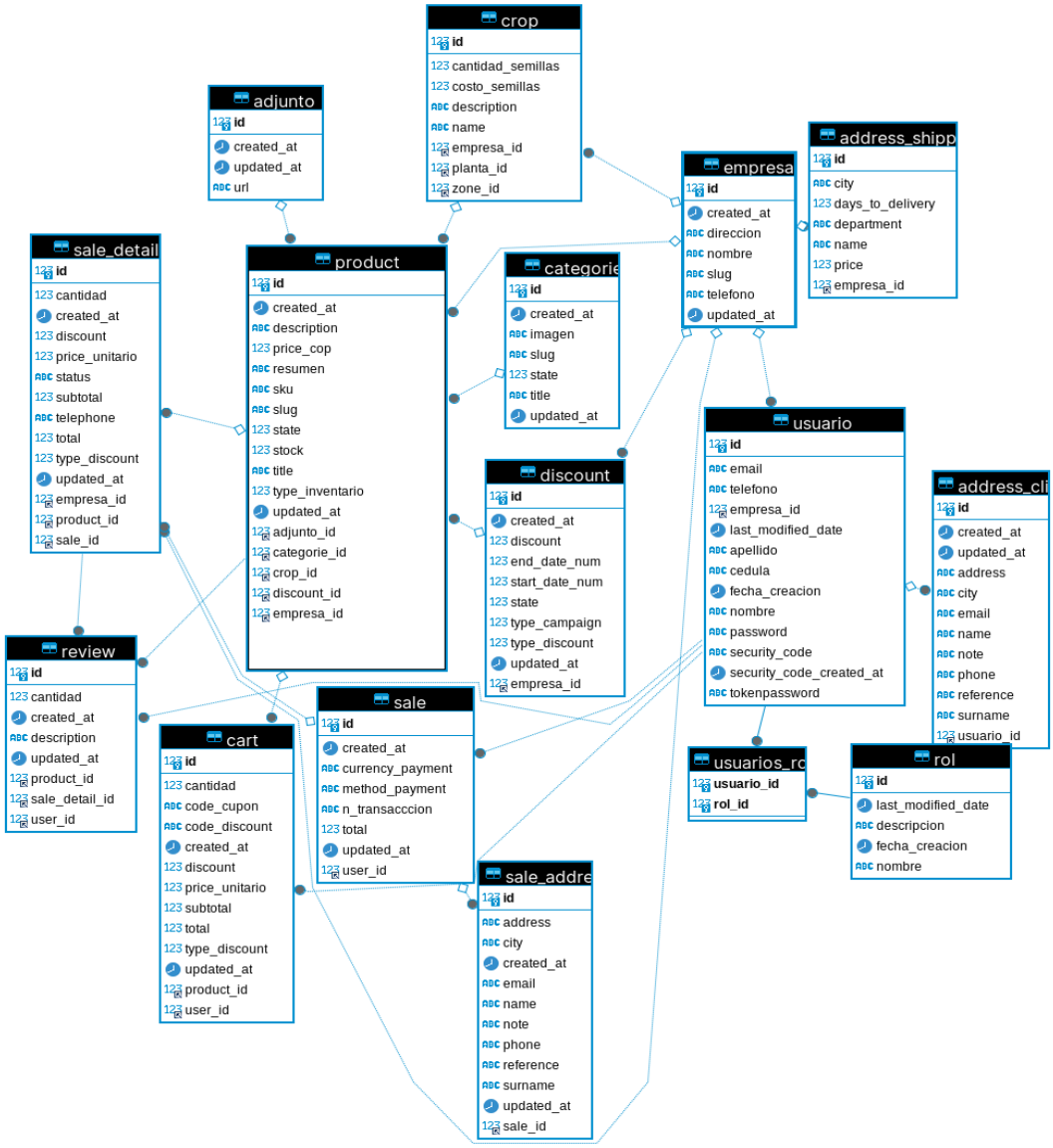
Consistencia: Mantiene muy bien la consistencia en operaciones estándar y la integridad de los datos, así sea un prototipo.

Además, MySQL suele ser muy usado en los entornos web y tiene mucha compatibilidad con algunos lenguajes de programación lo que sería muy útil para realizar un prototipo de aplicación web. (Ollarves, 2023)

5.2.2.2 Diagrama del modelo de datos. En la siguiente figura se muestra el diagrama de modelo de datos sobre el que se trabajó en la fase de implementación del prototipo.

Figura 9.

Diagrama del modelo de datos.



Nota: *Romero, A., & Sandoval, J.* (2024). Diseño propio.

5.2.3 Diseño del backend.

5.2.3.1 Elección del lenguaje. Dentro de los siguientes lenguajes de programación, cuyos parámetros de elección son la facilidad de desarrollo, el rendimiento, bibliotecas y escalabilidad, se tenía que escoger uno que fuese más acorde a nuestro proyecto de marketplace.

Tabla 22.

Lenguajes de programación con parámetros de elección.

Lenguaje	Facilidad	Rendimiento	Bibliotecas	Escalabilidad
Python	Muy alta	Bueno	Muy buenas	Normal
Node js	Alta	Muy bueno	Muy buenas	Buena
Ruby	Muy alta	Bueno	Buenas	Normal
Java	Alta	Muy bueno	Buenas	Buena

Nota: *Romero, A., & Sandoval, J.* (2024). Diseño propio.

Rendimiento y escalabilidad: Java es destacado por alto rendimiento y escalabilidad significativa. La capacidad de escalar servicios de manera independiente facilita el manejo eficiente de la carga de trabajo, una característica crucial para una plataforma en desarrollo.

Bibliotecas: Aunque Java puede no tener una biblioteca tan extensa como Python o Node js, su conjunto de bibliotecas es sólido y ampliamente utilizado en el desarrollo empresarial. Además, la comunidad y la documentación existente brinda un respaldo más que valioso para el equipo de desarrollo.

Facilidad: Aunque Python y Ruby son conocidos por su facilidad, optamos por Java debido al conocimiento previo disponible en el equipo. La familiaridad con Java reduce la curva de aprendizaje y permite una implementación más rápida y eficiente.

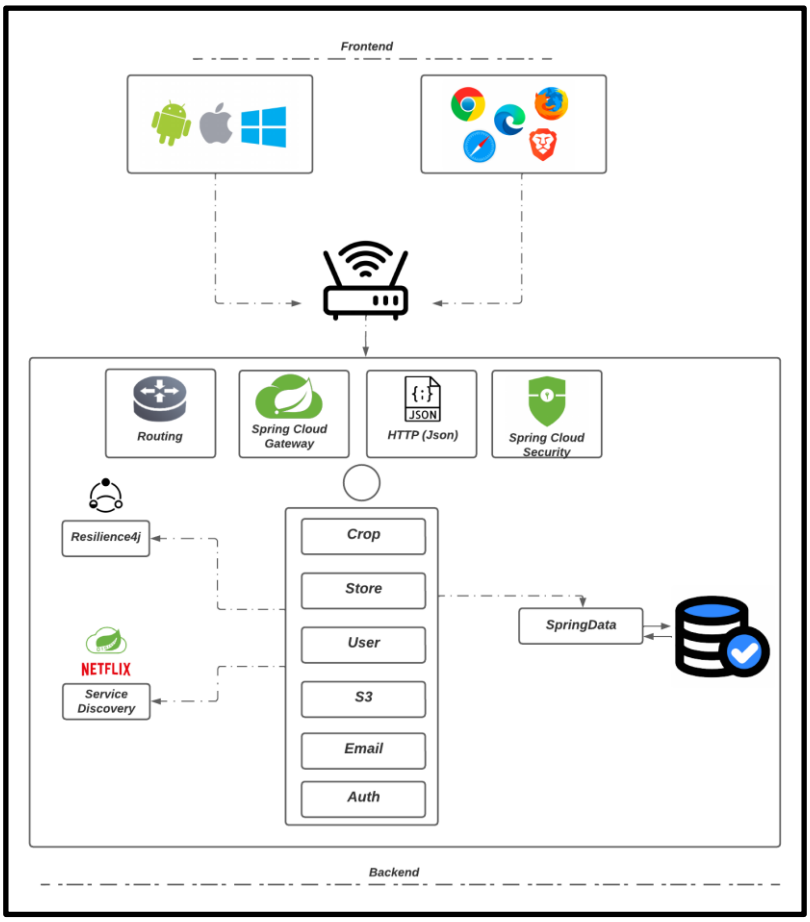
Microservicios: La elección de Java para implementar la arquitectura de microservicios se basó en un criterio crucial. Spring Boot, como framework destacado, agiliza el desarrollo de microservicios gracias a su configuración sencilla y convenciones predefinidas. Su amplio ecosistema, especialmente con herramientas como Spring Cloud, simplifica la implementación de patrones de microservicios. Además, Spring Boot ofrece características integradas que mejoran la

seguridad, eliminando la necesidad de configuraciones extensas para implementar prácticas seguras.

5.2.3.2 Arquitectura backend. En cuanto a la arquitectura del backend, se ha optado por una arquitectura de microservicios, permitiendo una mayor modularidad y escalabilidad. Cada servicio se encargará de funciones específicas, facilitando el mantenimiento y desarrollo ágil.

Figura 10.

Arquitectura backend.

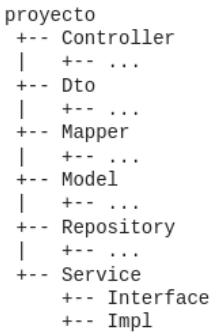


Nota: *Romero, A., & Sandoval, J.* (2024). Diseño propio.

5.2.3.3 Estructura de directorios. En la siguiente figura se representa la estructura de directorios con la que cuenta cada proyecto de microservicio.

Figura 11.

Estructura de directorios backend.



Nota: *Romero, A., & Sandoval, J.* (2024). Diseño propio.

Controller: Contendrá las clases que manejan la lógica de los controladores rest. Estos archivos manejan las solicitudes HTTP, interactúan con los servicios y gestionan las respuestas.

Dto: Este directorio contendrá las clases que encapsulan datos para transferir información entre servicios.

Mapper: Contendrá las interfaces que son responsables de convertir entre diferentes estructuras de datos, como convertir de Dto a entidades.

Model: En este directorio contendrá las clases que representan las entidades del negocio, como representaciones de tablas en la base de datos.

Repository: En esta carpeta se encuentran las interfaces que extienden JpaRepository, las cuales son responsables de definir métodos para gestionar operaciones de acceso y manipulación de registros en la base de datos.

Service: Esta carpeta contendrá los directorios:

Interface: En el cual estarán las interfaces que describen los métodos proporcionados por los servicios.

Impl: En el cual estarán las clases que implementan las interfaces y donde se implementaría la lógica del negocio.

5.2.4 Diseño del prototipo en su componente frontend.

5.2.4.1 Elección de framework. Con respecto a la elección del más adecuado de los siguientes frameworks de JavaScript, para un proyecto de marketplace, esta dependió de varios factores como fueron la curva de aprendizaje, documentación y el manejo bidireccional de los datos.

Tabla 23.

Frameworks de Javascript con parámetros de elección.

Framework	Curva de aprendizaje	Documentación	Manejo bidireccional de los datos
React	Normal	Amplia	Normal (gestionando el estado)
Angular	Alta	Muy amplia	Fácil
Vuejs	Baja	Suficiente	Fácil

Nota: *Romero, A., & Sandoval, J.* (2024). Diseño propio.

La elección de Angular como el framework de Javascript para este proyecto no solo se basa en las características técnicas identificadas anteriormente, sino también en el conocimiento previo que el equipo de trabajo posee sobre esta herramienta.

Curva de aprendizaje: A pesar de ser Angular aquel lenguaje con la curva de aprendizaje más alta de las tres opciones planteadas, como se mencionó, el equipo ya tenía conocimiento sobre este framework lo que hace que la implementación con esta herramienta sea más eficiente.

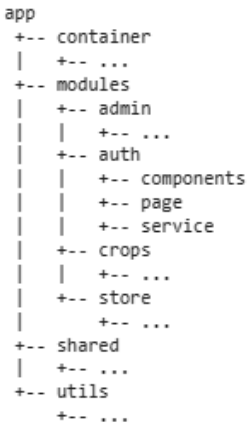
Documentación: En este apartado se aprovecha tal conocimiento, pues ya se tiene una documentación mucho más grande que en las otras dos opciones lo que posibilita realizar el prototipo con muchas más alternativas.

Manejo bidireccional de los datos: Únicamente se equipara con Vuejs, sin embargo, ambas son sencillas y Angular se sobrepone a este por el conocimiento previo, pues facilita aún más este manejo.

5.2.4.2 Estructura de directorios. En la siguiente figura se representa la estructura de directorios que cuenta el proyecto en su apartado frontend.

Figura 12.

Estructura de directorios frontend.



Nota: *Romero, A., & Sandoval, J.* (2024). Diseño propio.

app: Este directorio contiene el código fuente principal de toda la aplicación. Acá se organizan los módulos y otros archivos relacionados con la aplicación.

container: En este directorio se guardan aquellos componentes que actúan de manera estática en la aplicación, como lo son el header, el footer, menús y barras de navegación.

modules: Los módulos son utilizados para organizar y estructurar la aplicación en funcionalidades coherentes. Los módulos permiten encapsular componentes, servicios y otros artefactos relacionados, facilitando la modularidad y el mantenimiento de la aplicación.

admin: Acá se encuentran los componentes relacionados a las vistas de ingreso como administrador y la creación de la empresa.

auth: Directorio en el que se encuentran los componentes y servicios relacionados a la autentificación, ingreso, registro y recuperación de cuenta.

crops: Directorio en el que se encuentran los componentes y servicios relacionados a la gestión y visualización de los cultivos de la empresa.

store: Directorio en el que se encuentran los componentes y servicios relacionados a la creación de productos y funcionalidades del marketplace.

Para el desarrollo de cada uno de los componentes presentes en cada directorio nos apoyamos en la biblioteca de componentes Primeng.

5.2.5 Diseño del prototipo en su aplicativo móvil.

5.2.5.1 Elección de framework. Para tomar uno de los siguientes frameworks (presentes entre los más famosos) de desarrollo móvil, fue importante tener en cuenta algunos criterios de elección como lo fueron la curva de aprendizaje, el desarrollo multiplataforma y el rendimiento.

Tabla 24.

Frameworks de aplicaciones móviles con parámetros de elección.

Framework	Curva de aprendizaje	Desarrollo multiplataforma	Rendimiento
Flutter	Moderada	Excelente	Excelente
Xamarin	Moderada	Excelente	Bueno
React Native	Moderada	Excelente	Bueno
Ionic	Baja	Excelente	Moderado

Nota: *Romero, A., & Sandoval, J.* (2024). Diseño propio.

Así que teniendo en cuenta la información de la tabla rellenada tras la investigación de las características de cada uno de los frameworks, se eligió el framework de Flutter (por tener mejor rendimiento que el resto), que conlleva usar el lenguaje de programación Dart. Sin embargo, fue elegido siendo un lenguaje sobre el que el equipo no poseía ningún conocimiento previo, por lo que se tuvo que tomar a pesar de poseer una curva de aprendizaje mayor a la de Ionic. (*5 frameworks populares para el desarrollo de aplicaciones móviles - blog*, 2023)

5.2.5.2 Arquitectura móvil. En la app móvil se implementa la arquitectura limpia, pues es un patrón de diseño que busca mantener una separación entre las diferentes capas y responsabilidades de una aplicación. Se compone de capas como:

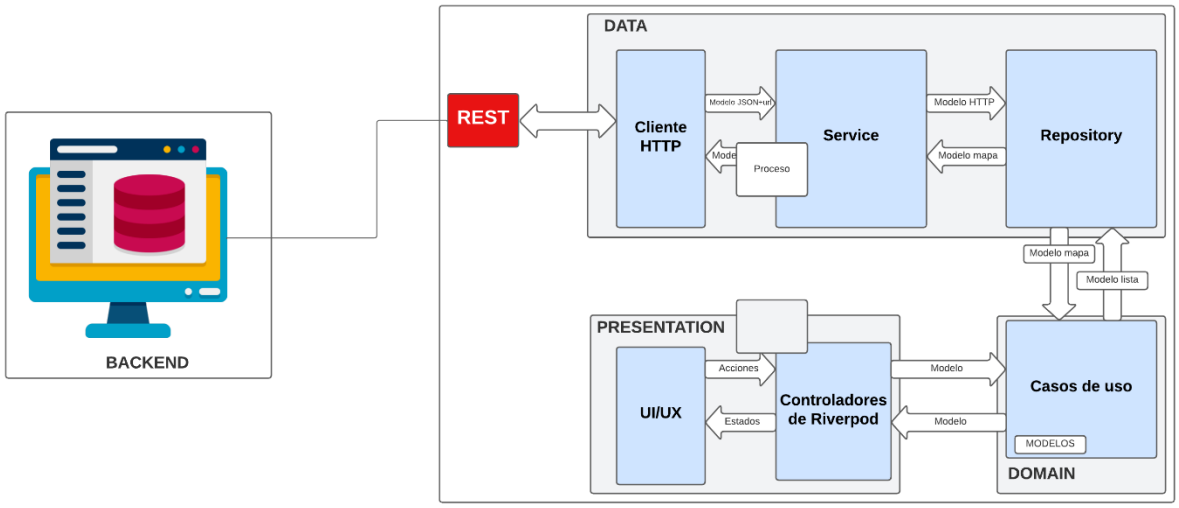
Capa de presentación (UI): Puedes utilizar Riverpod para administrar el estado local de los widgets y componentes de la interfaz de usuario. Los proveedores de Riverpod pueden proporcionar datos específicos de la pantalla y eventos que afectan al estado.

Capa de dominio: En esta capa, puedes utilizar Riverpod para manejar el estado global de la aplicación que afecta a múltiples casos de uso. Esto puede incluir estados de autenticación, configuración y otros aspectos que no son específicos de una pantalla.

Capa de datos: Riverpod se puede utilizar para proporcionar instancias de repositorios y fuentes de datos a la capa de Dominio, manteniendo la independencia de los detalles de implementación.

Figura 13.

Diagrama de implementación de riverpod con arquitectura limpia.



0Nota: *Romero, A., & Sandoval, J.* (2024). Diseño propio.

En la organización de directorios, se ha establecido una estructura coherente en todas las plataformas. Tanto en el backend como en el frontend y la aplicación móvil, la disposición de directorios sigue un patrón lógico que facilita la navegación y el desarrollo colaborativo.

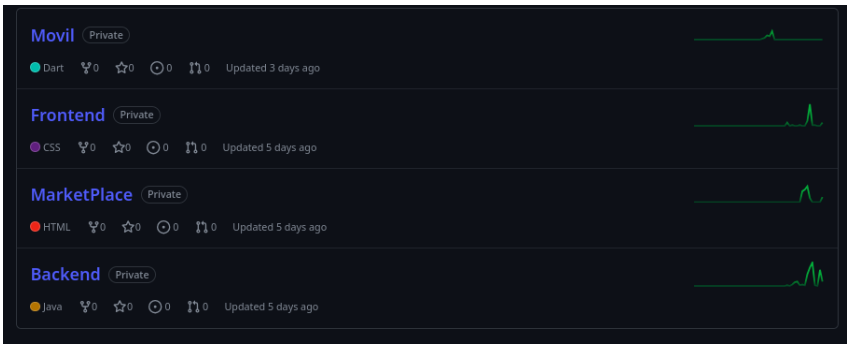
5.3 Fase de implementación

5.3.1 Adecuación del repositorio

Antes de iniciar la implementación del código, se configuraron cuatro repositorios distintos para abordar las áreas específicas del proyecto: backend, frontend, marketplace y móvil. Es importante destacar que en el backend se adoptó un enfoque de monorepositorio, centralizando así todos los microservicios que serán desarrollados y que el frontend se dividió en un repositorio con el mismo nombre ‘frontend’ y en otro con el nombre ‘marketplace’, para llevar una mejor diferenciación de tareas, donde en el primero se desarrollaron los módulos de administración y en el segundo los módulos de comercialización. El enlace que nos dirige a esta organización es: https://github.com/AgroTech-Uis.

Figura 14.

Organización del repositorio del proyecto.



Nota: Organización de los repositorios empleados. Recuperado de https://github.com/AgroTech-Uis

5.3.2 Implementación del backend

Como previamente se ha mencionado, este proyecto se basa en la arquitectura de microservicios (Ver figura 10). A continuación, se presenta un desglose detallado del

funcionamiento de cada microservicio. Antes de entrar en esos detalles, es importante mencionar los pasos iniciales llevados a cabo para la creación de los proyectos de Spring Boot, la configuración de la base de datos y otras configuraciones globales compartidas por todos los microservicios. Es importante resaltar que la estructura del proyecto se ha diseñado de manera modular, con distintos subproyectos, siendo cada uno de ellos la representación de un microservicio específico.

5.3.2.1 Configuracion del proyecto. El proyecto se configuró como un conjunto de subproyectos, donde cada microservicio se trata como un módulo individual del proyecto general. Cada vez que se integran nuevos microservicios en el proyecto, se adopta la práctica de agregar un nuevo módulo, lo que automáticamente generaba la inclusión de dicho módulo en el archivo pom.xml global.

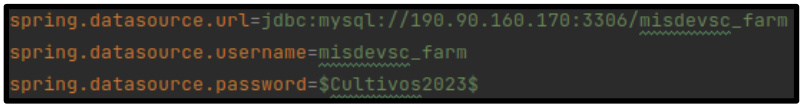
El propósito detrás de esta estrategia era consolidar todos los microservicios dentro de un solo repositorio. Además, esto facilitó la instalación de dependencias en el proyecto ya que solo era necesario ejecutar el comando de instalación una única vez.

Al crear un nuevo módulo, se elegía la opción de utilizar el generador de Spring Initializr. Se proporcionaba el nombre del proyecto, se seleccionaba Maven como sistema de gestión de proyectos y se asignaba un nombre al grupo (group) y al artefacto (artifact). Luego se agregan dependencias necesarias para cada microservicio en el pom.xml.

5.3.2.2 Conexión con la base de datos. Para interactuar con la base de datos se realizó utilizando el ORM JPA lo que nos facilita gestionar todas las operaciones con la base de datos. La configuración de la conexión con la base de datos se llevó a cabo en el archivo application.properties de cada microservicio. En dicho archivo se especificaba la URL del host de la base de datos, el nombre de usuario y la contraseña para acceder.

Figura 15.

Configuración de la conexión con la base de datos.



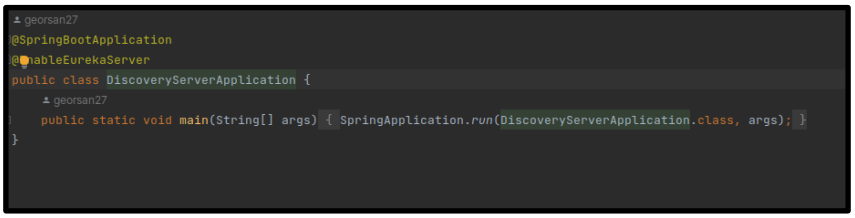
Nota: *Romero, A., & Sandoval, J.* (2024). Diseño propio.

5.3.2.3 Configuración de Spring Cloud Netflix Eureka. Con el propósito de incorporar Eureka Server al proyecto, se creó un microservicio dedicado a implementar este servicio. Eureka Server se encarga de que todos los microservicios se registren en un único punto, permitiéndoles encontrarse y comunicarse entre sí sin la necesidad de configurar manualmente el host y el puerto, sino referenciándose simplemente por el nombre del microservicio. En el proyecto, este servicio se denomina "service register".

La configuración de este servicio implica agregar la dependencia spring-cloud-starter-netflix-eureka-server y habilitar el servidor Eureka mediante la anotación @EnableEurekaServer en la clase principal (main). Posteriormente, se realiza la configuración en el archivo application.properties para que el servidor se ejecute en el puerto 8761 y para que no se registre, ya que su función es actuar como servidor central.

Figura 16.

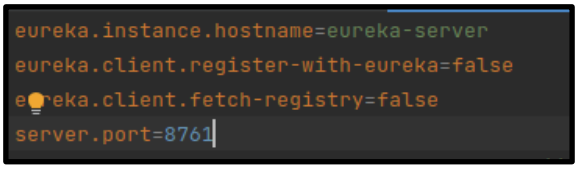
Habilitación de eureka server.



Nota: *Romero, A., & Sandoval, J.* (2024). Diseño propio.

Figura 17.

Configuración de eureka server.



Nota: *Romero, A., & Sandoval, J.* (2024). Diseño propio.

5.3.2.4 Configuración de Spring Cloud Netflix Eureka cliente. En cada microservicio, fue necesario realizar una configuración específica para activar el cliente Eureka. Esto tiene como objetivo permitir que cada microservicio se registre en el servidor Eureka dentro del proyecto. Para facilitar este proceso, se utiliza la dependencia Maven spring-cloud-starter-netflix-eureka-client. Al incluir esta dependencia en el archivo pom.xml, no se requiere configuración adicional compleja, ya que las clases y configuraciones necesarias son proporcionadas automáticamente.

Posteriormente, para que el microservicio se registre en el servidor Eureka, fue necesario agregar la dirección del servidor en el archivo de configuración application.properties y el nombre de la aplicación.

Figura 18.

Configuración de eureka cliente.

Nota: *Romero, A., & Sandoval, J.* (2024). Diseño propio.

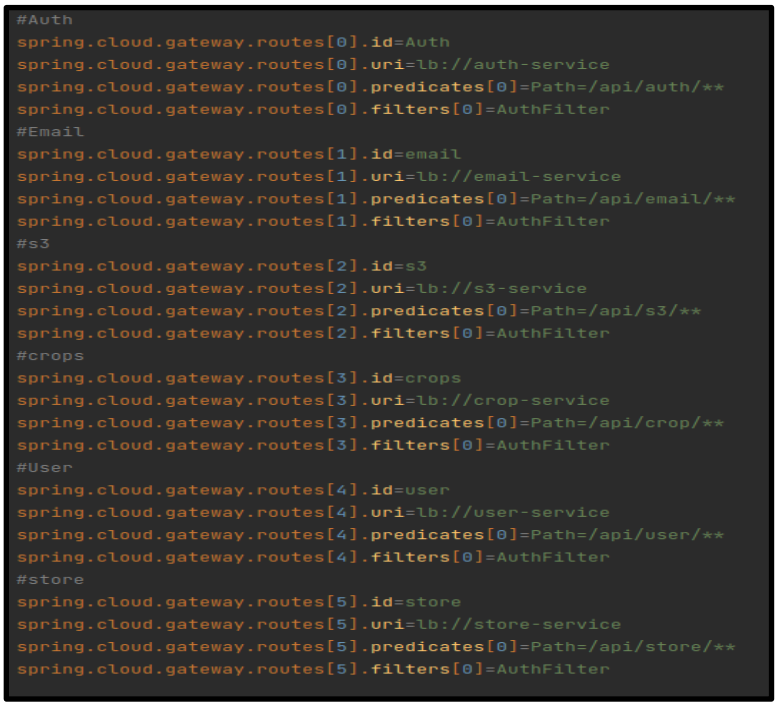
5.3.2.5 Configuración del ApiGateway. En el marco de este proyecto, se ha desarrollado un servicio Spring Cloud Gateway con el propósito de unificar todas las solicitudes REST provenientes de un cliente. Este servicio está diseñado para gestionar y redirigir de manera eficiente las peticiones hacia los microservicios específicos. La configuración necesaria para esta funcionalidad se ha simplificado gracias a la inclusión de la dependencia spring-cloud-starter-gateway, encargada de establecer la configuración esencial para el gateway.

Adicionalmente, se ha implementado un filtro de seguridad en el gateway para verificar la presencia de un token JWT en el encabezado de las solicitudes. Si una petición carece de este token, se responde con un error al cliente. En caso contrario, el servicio realiza una verificación adicional mediante una solicitud al microservicio de autenticación. Este último cuenta con un endpoint específico para validar si el token posee los permisos necesarios para la ejecución de la solicitud. En caso contrario, se notifica al cliente acerca de la falta de autorización.

La configuración de las rutas que el API Gateway gestionará se realiza en el archivo application.properties. Aquí se establecen las correspondencias entre las rutas y los microservicios de destino.

Figura 19.

Configuración de las rutas en ApiGateway.



```
#Auth
spring.cloud.gateway.routes[0].id=Auth
spring.cloud.gateway.routes[0].uri=lb://auth-service
spring.cloud.gateway.routes[0].predicates[0]=Path=/api/auth/**
spring.cloud.gateway.routes[0].filters[0]=AuthFilter
#Email
spring.cloud.gateway.routes[1].id=email
spring.cloud.gateway.routes[1].uri=lb://email-service
spring.cloud.gateway.routes[1].predicates[0]=Path=/api/email/**
spring.cloud.gateway.routes[1].filters[0]=AuthFilter
#s3
spring.cloud.gateway.routes[2].id=s3
spring.cloud.gateway.routes[2].uri=lb://s3-service
spring.cloud.gateway.routes[2].predicates[0]=Path=/api/s3/**
spring.cloud.gateway.routes[2].filters[0]=AuthFilter
#crops
spring.cloud.gateway.routes[3].id=crops
spring.cloud.gateway.routes[3].uri=lb://crop-service
spring.cloud.gateway.routes[3].predicates[0]=Path=/api/crop/**
spring.cloud.gateway.routes[3].filters[0]=AuthFilter
#User
spring.cloud.gateway.routes[4].id=user
spring.cloud.gateway.routes[4].uri=lb://user-service
spring.cloud.gateway.routes[4].predicates[0]=Path=/api/user/**
spring.cloud.gateway.routes[4].filters[0]=AuthFilter
#store
spring.cloud.gateway.routes[5].id=store
spring.cloud.gateway.routes[5].uri=lb://store-service
spring.cloud.gateway.routes[5].predicates[0]=Path=/api/store/**
spring.cloud.gateway.routes[5].filters[0]=AuthFilter
```

Nota: *Romero, A., & Sandoval, J.* (2024). Diseño propio.

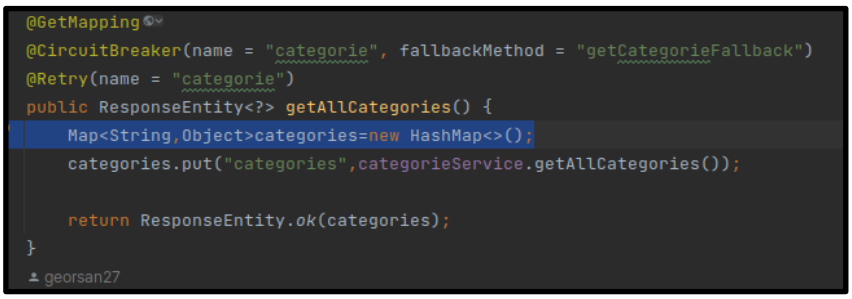
5.3.2.6 Configuración de los patrones CircuitBreaker, Retry y Fallback. En cada microservicio se ha implementado los patrones circuitBreaker,retry y fallback con el objetivo de fortalecer la resiliencia y tolerancia de cada servicio. estos patrones forman parte de la biblioteca resilience4j. El patrón circuitBreaker se encargará de prevenir y detectar fallos temporales al evitar operaciones innecesarias o aquellas con una elevada probabilidad de error. El patrón retry se utiliza para reintentar una petición un número específico de veces, mientras que fallback se encarga de manejar de manera controlada las situaciones en las que una petición falla.

La implementación de estos patrones en cada microservicio se llevó a cabo mediante el uso de anotaciones específicas, como el @CircuitBreaker y @Retry aplicadas en los controladores de cada solicitud.

En cuanto a la implementación del fallback por cada controlador se creaba una función donde se manejaban los posibles errores que iba a generar ese servicio.

Figura 20.

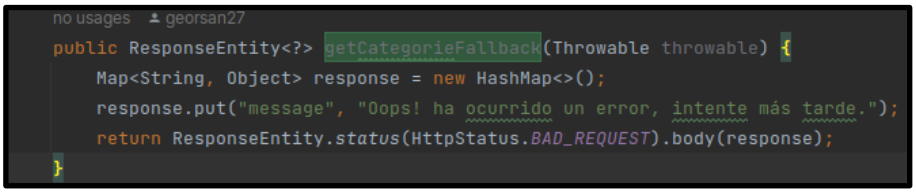
Configuración de CircuitBreaker y retry.



Nota: *Romero, A., & Sandoval, J.* (2024). Diseño propio.

Figura 21.

Implementación del fallback.



Nota: *Romero, A., & Sandoval, J.* (2024). Diseño propio.

.

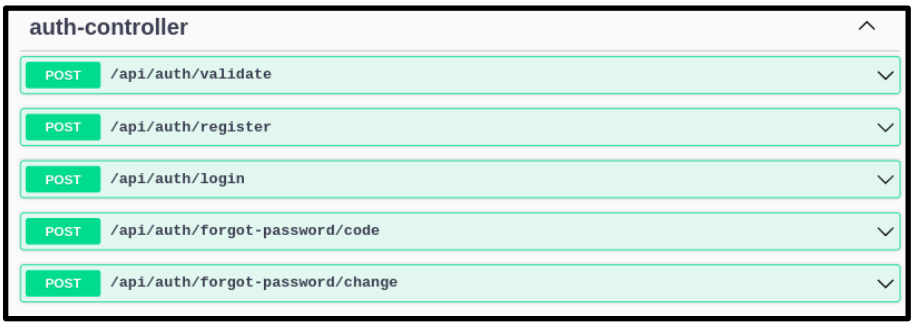
5.3.2.7 Microservicio Auth. Este microservicio se encarga de gestionar la autenticación, registro, validación de tokens y recuperación de cuentas. Durante el registro de un usuario, la

contraseña se cifra mediante el algoritmo bcrypt. Cuando un usuario inicia sesión exitosamente, se emite un token con una duración de 4 días y los permisos correspondientes según su rol.

En el escenario en que un usuario desee recuperar su contraseña, se realiza una validación de su existencia en el sistema. Posteriormente, se genera un código de seguridad con una vigencia de 1 minuto. Este código se envía al correo del usuario mediante una solicitud al microservicio de email. La figura 22 expone los endpoints disponibles en este microservicio.

Figura 22.

Endpoints implementados.



5.3.2.8 Microservicio de cultivos. Este microservicio se encarga de gestionar los cultivos de una empresa, con una validación que impide la existencia de cultivos con nombres duplicados en una empresa. Los endpoints disponibles incluyen la obtención de todos los cultivos de la empresa, la lista de nombres de cultivos en la base de datos, la creación, actualización y eliminación de cultivos. La figura 23 muestra de manera visual los endpoints disponibles en este microservicio.

Figura 23.

Endpoints implementados para la gestión de cultivos.

5.3.2.9 Microservicio Store. Este microservicio administra diversos aspectos relacionados con la tienda, proporcionando servicios para la gestión de la tienda en general, productos, envíos y cálculos de precios para envíos, pedidos, descuentos y despachos. Las siguientes figuras muestran visualmente los endpoints disponibles en este microservicio.

Figura 24.

Endpoints implementados para la gestión de productos

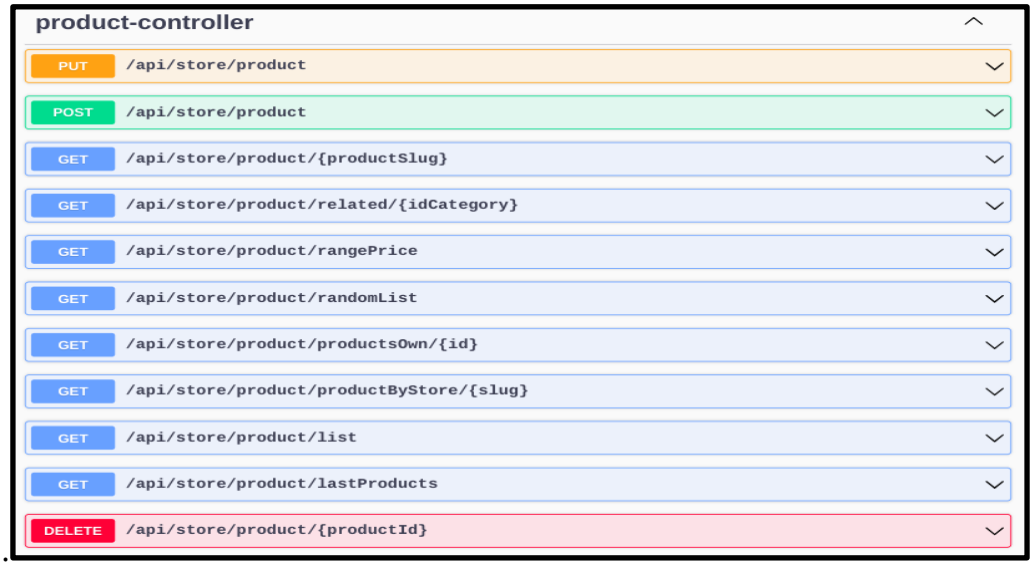


Figura 25.

Endpoints implementados para la gestión de descuentos.

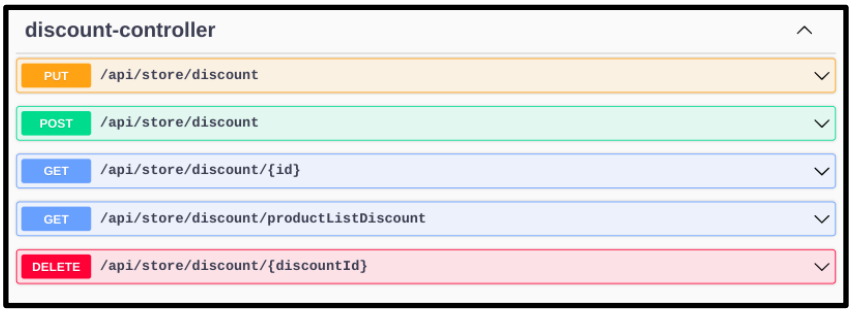


Figura 26.

Endpoints implementados para la gestión de empresas.

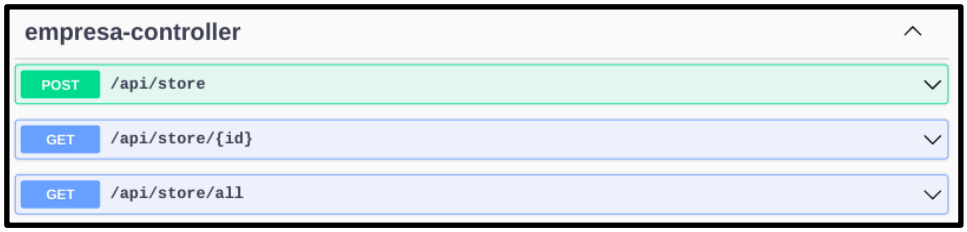


Figura 27.

Endpoints implementados para la gestión de compras en el marketplace.

Figura 28.

Endpoint implementado para obtener las categorías del marketplace.

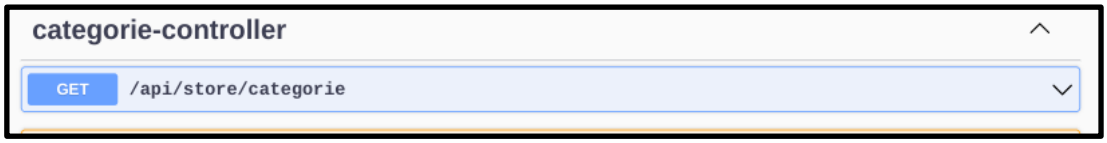


Figura 29.

Endpoints implementados para gestionar las direcciones de envío de la tienda.

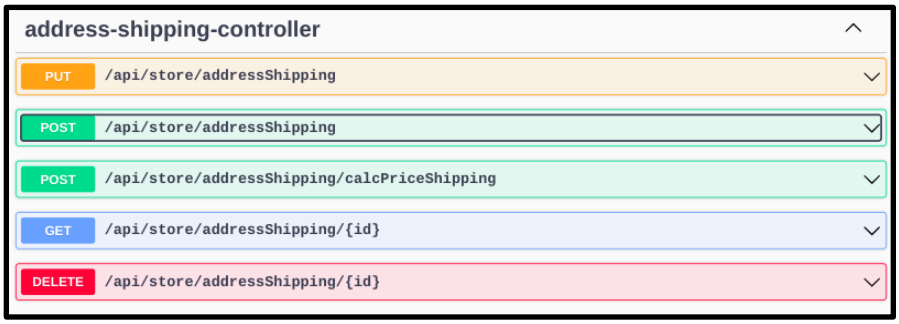


Figura 30.

Endpoint implementado para los detalles de la compra.

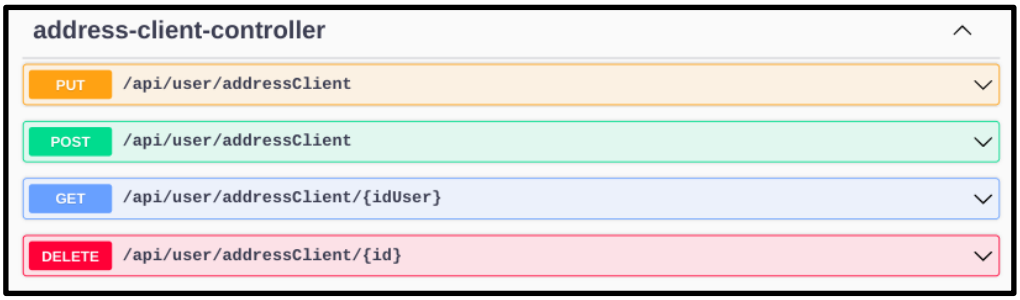
5.3.2.10 Microservicio User. El Microservicio de Usuario (User) se encarga de manejar todas las operaciones relacionadas con un usuario. Esto incluye la capacidad de actualizar la información del usuario y de registrar detalles sobre las direcciones de envío a las cuales desea recibir sus productos. En las siguientes figuras se puede ver los endpoint disponibles para este microservicio

Figura 31.

Endpoint implementado para actualizar la información del usuario.

Figura 32.

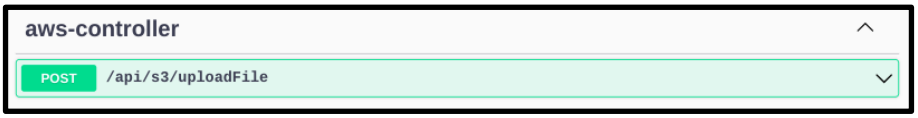
Endpoints implementados para gestionar las direcciones de un usuario.



5.3.2.11 Microservicio S3. Este microservicio tiene la responsabilidad de cargar una imagen a un bucket en Amazon S3. Para lograr esto, se utiliza el servicio de AWS S3, donde se ha creado un bucket específico para almacenar las imágenes que los usuarios suban desde un cliente. Es importante destacar que el bucket fue creado a través de la consola de AWS, y el microservicio hace uso del SDK disponible para Java, permitiendo así la integración efectiva con el servicio de almacenamiento en la nube de Amazon.

Figura 33.

Endpoints implementados para subir una imagen a AWS S3.



5.3.2.12 Microservicio Email. Este microservicio se encarga de enviar correos electrónicos a los usuarios utilizando la dependencia spring-boot-starter-mail, la cual nos proporcionó acceso a clases encargadas del envío de emails. Posteriormente, era necesario configurar en el archivo application.properties la información detallada en la figura 34.

Figura 34.

Configuración de spring mail.

```
spring.mail.host=smtp.gmail.com
spring.mail.port=587
spring.mail.username=marketplace234e@gmail.com
spring.mail.password=wybmprhawfncorsw
spring.mail.properties.mail.smtp.auth=true
spring.mail.properties.mail.smtp.starttls.enable=true
spring.mail.properties.mail.smtp.starttls.trust=smtp.gmail.com
```

Nota: *Romero, A., & Sandoval, J.* (2024). Diseño propio.

En ese momento, elegimos el host de Gmail, donde creamos una cuenta de correo dedicada para su uso. En el mismo archivo de configuración, se ingresa la dirección de correo electrónico y la contraseña, que requería configuración previa para su obtención. Este proceso implicaba acceder a la configuración del correo, buscar la sección de "Contraseña de aplicaciones" y generar una contraseña específica. Luego, generamos un archivo HTML que contenía un mensaje con el código de seguridad.

Figura 35.

Endpoints implementados para enviar un email

5.3.3 Implementación del frontend

En este proyecto, desarrollamos el frontend con dos proyectos distintos en Angular: uno para la administración y otro para el marketplace. Esta separación de responsabilidades mejora la organización y el mantenimiento, favorece la escalabilidad y facilita actualizaciones.

5.3.3.1 Creación de proyectos angular. Para dar inicio al desarrollo del frontend, se crearon dos proyectos separados utilizando el comando ng init en la terminal, lo cual generó la estructura básica necesaria. Luego, se procedió a la instalación de los paquetes adicionales mediante el comando npm, asegurando la disponibilidad de las dependencias necesarias.

5.3.3.2 Implementacion frontend administrador. Este proyecto de frontend se enfoca en gestionar todas las operaciones relacionadas con la administración del marketplace, así como en proporcionar funcionalidades útiles para los campesinos.

5.3.3.2.1 Módulo de autentificación. El primer módulo desarrollado fue el módulo de autenticación. Este incluye interfaces para iniciar sesión, registrar nuevos usuarios y recuperar cuentas.

5.3.3.2.1.1 Interfaz de inicio de sesión. Esta interfaz presenta un formulario con campos exclusivos para correo electrónico y contraseña, incorporando validaciones para ambos. Se optimiza el espacio al incluir enlaces que redirigen a las vistas de creación de cuenta y recuperación de contraseña.

Figura 36.

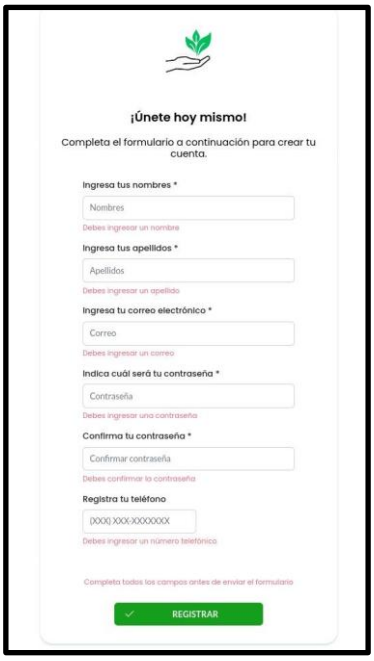
Vista de inicio de sesión.

Nota: *Romero, A., & Sandoval, J.* (2024). Diseño propio.

5.3.3.2.1.2 Interfaz de registro de usuario. La interfaz de registro de usuario incorpora un formulario que incluye diversos campos destinados a la recopilación de datos personales de los usuarios. Entre estos campos los usuarios pueden ingresar su nombres, apellidos, teléfono y contraseña, cada uno con su correspondiente proceso de validación.

Figura 37.

Vista de registro de cuenta.



Nota: *Romero, A., & Sandoval, J.* (2024). Diseño propio.

5.3.3.2.1.3 Interfaz de recuperación de cuenta. Con la vista de recuperación de contraseña de la cuenta, el usuario debe ingresar el correo electrónico con el cual está registrado. A este correo se le enviará un código de seguridad.

Figura 38.

Vista de recuperación de contraseña.

Nota: *Romero, A., & Sandoval, J.* (2024). Diseño propio.

5.3.3.2.1.4 Interfaz de cambio de contraseña. En esta interfaz, el usuario deberá ingresar el código de seguridad que ha sido enviado a su correo electrónico. Además, se le solicitará que establezca una nueva contraseña para completar el proceso de recuperación de cuenta. Una vez completado este procedimiento, el usuario tendrá acceso a todas las funcionalidades disponibles cuando esté autenticado en su cuenta.

Figura 39.

Vista de cambio de contraseña.

Nota: *Romero, A., & Sandoval, J.* (2024). Diseño propio.

5.3.3.2.1.4 Correo electrónico de recuperación. El código que se debe ingresar en la vista anterior corresponde al mismo que se envía al correo, como se aprecia a continuación.

Figura 40.

Correo electrónico para recuperar contraseña.

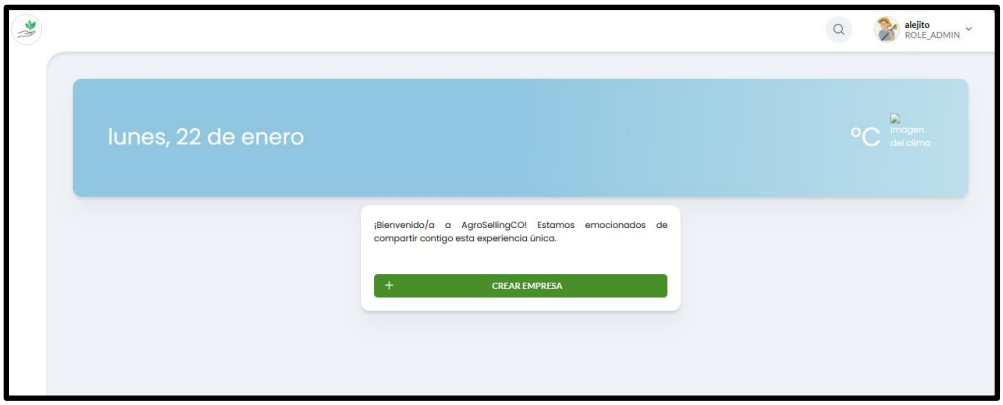
Nota: Correo electrónico enviado a nuestra dirección con código para recuperar contraseña.

Recuperado de https://outlook.live.com/mail/0/

5.3.3.2.2 Módulo administrador. En este módulo, cuando un nuevo usuario ingresa, se le solicitará que cree una empresa. Este paso es fundamental para acceder a las funcionalidades completas de la aplicación.

Figura 41.

Creación de la empresa al iniciar sesión en nueva cuenta.



Nota: *Romero, A., & Sandoval, J.* (2024). Diseño propio.

Para crear la empresa, es necesario que el usuario rellene un formulario así:

Figura 42.

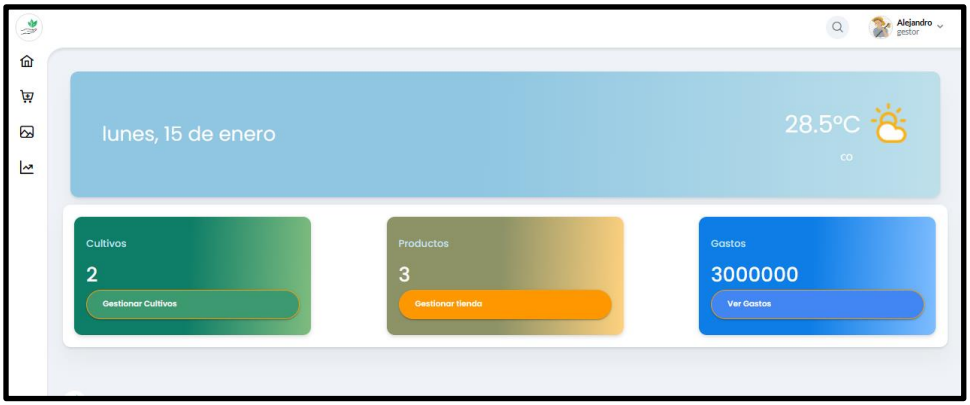
Formulario de creación de la empresa.

Nota: *Romero, A., & Sandoval, J.* (2024). Diseño propio.

5.3.3.2.2.1 Vista de inicio de administrador. Una vez que el usuario ha creado su empresa, podrá explorar la vista inicial de la aplicación. En esta sección, se le ofrecerá la posibilidad de seleccionar diversas funcionalidades que lo dirigirán a otras vistas específicas. Esto incluye la opción de gestionar los cultivos de la empresa, así como los productos asociados.

Figura 43.

Vista de panel de administración.



Nota: *Romero, A., & Sandoval, J.* (2024). Diseño propio.

5.3.3.2.2.2 Interfaz de gestión agrícola. Este módulo contiene todos los componentes dedicados a la gestión de cultivos. Además, proporciona acceso a otras funcionalidades desarrolladas que guardan relación con la gestión agrícola. Los detalles específicos sobre estas funciones adicionales se encuentran en los anexos.

Figura 44.

Vista de gestión de los cultivos agrícolas.

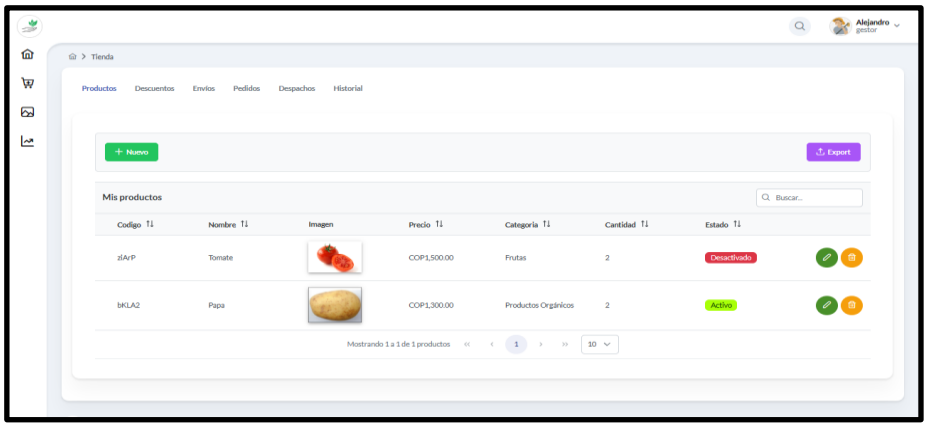
Nota: *Romero, A., & Sandoval, J.* (2024). Diseño propio.

5.3.3.2.3 Módulo gestión de marketplace

5.3.3.2.3.1 Interfaz de gestión de productos. Este módulo concentra todos los componentes destinados a la gestión del Marketplace. Incluye componentes para la gestión de productos, descuentos, envíos y zonas de envío, y la visualización de las listas de pedidos, listas de despachos e historial de todos los productos vendidos durante la permanencia del usuario en la plataforma.

Figura 44.

Vista de gestión de los productos a comercializar.

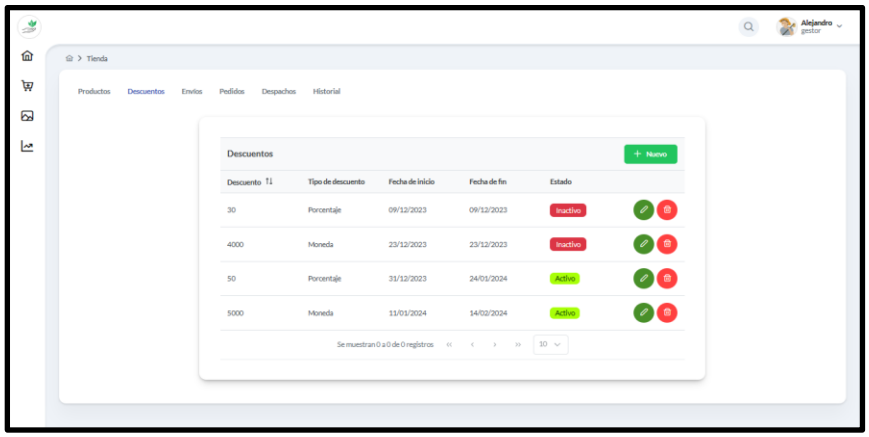


Nota: *Romero, A., & Sandoval, J.* (2024). Diseño propio.

5.3.3.2.3.2 Interfaz de gestión de descuentos. Es importante tener en cuenta que cada uno de los productos generados tiene la posibilidad de asignación de descuento. Estos descuentos deben estar preestablecidos y como administrador, se debía tener la posibilidad de gestionarlos.

Figura 45.

Vista de gestión de los descuentos.

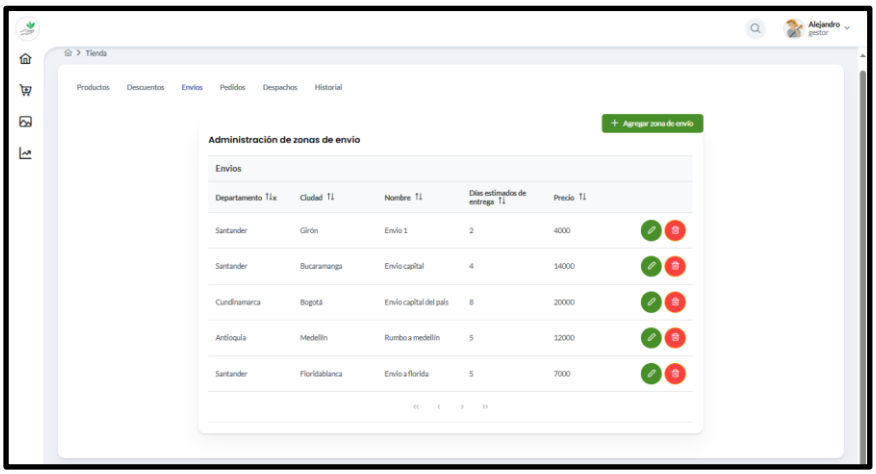


Nota: *Romero, A., & Sandoval, J.* (2024). Diseño propio.

5.3.3.2.3.3 Interfaz de gestión de envíos. La misma situación se presenta con los envíos, pues debemos tener una lista preestablecida y gestionable de envíos para asignarle el valor al que corresponde cada uno a los pedidos realizados por los usuarios.

Figura 46.

Vista de gestión de las zonas de envío.

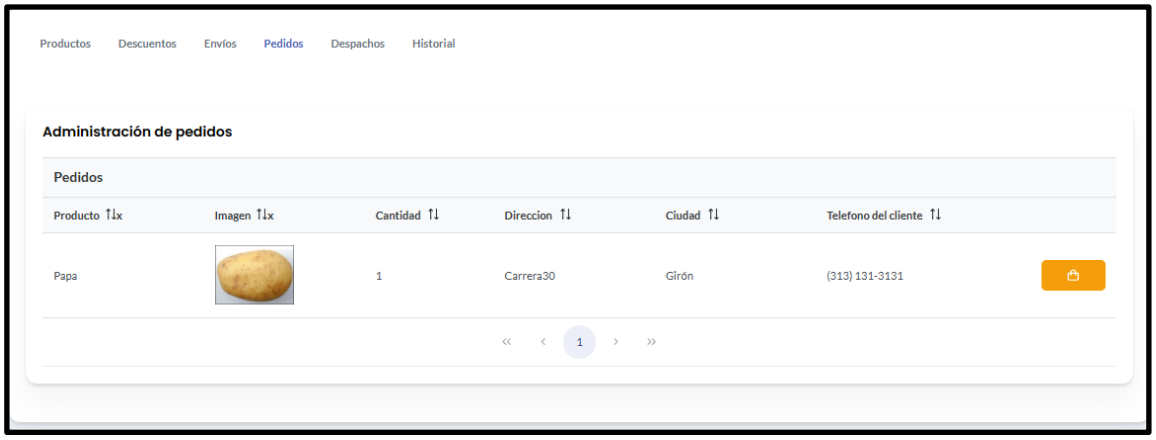


Nota: *Romero, A., & Sandoval, J.* (2024). Diseño propio.

Por otra parte, el administrador debe tener acceso a la gestión de los pedidos realizados por los clientes, y para ello se crea esta interfaz.

Figura 47.

Vista de los pedidos.



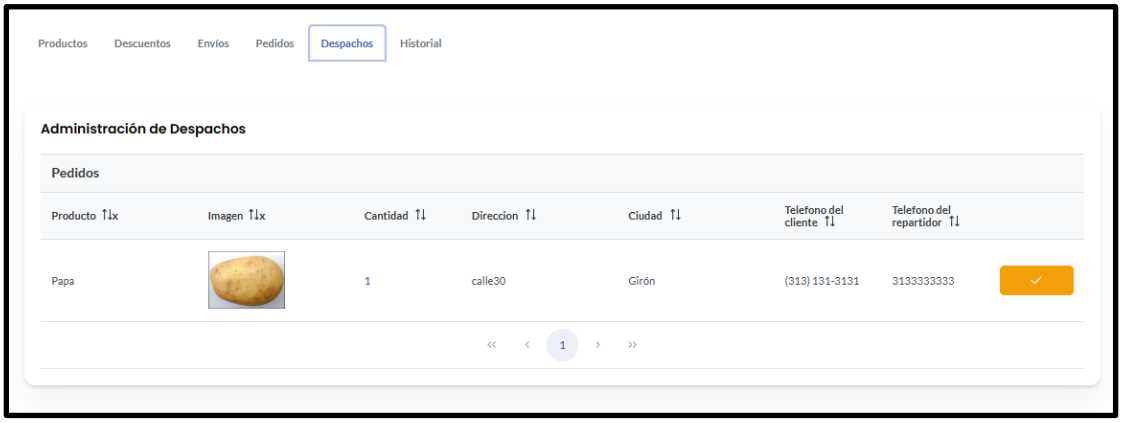
Nota: *Romero, A., & Sandoval, J.* (2024). Diseño propio.

El administrador también debía tener la posibilidad de decidir cuáles productos han sido despachados y cuáles no, para ello, se creó la vista de gestión de despachos. Para completar este

proceso, el administrador también debe tener acceso a la posibilidad de dar por entregados estos despachos o no, todo esto dentro de la misma gestión de despachos.

Figura 48.

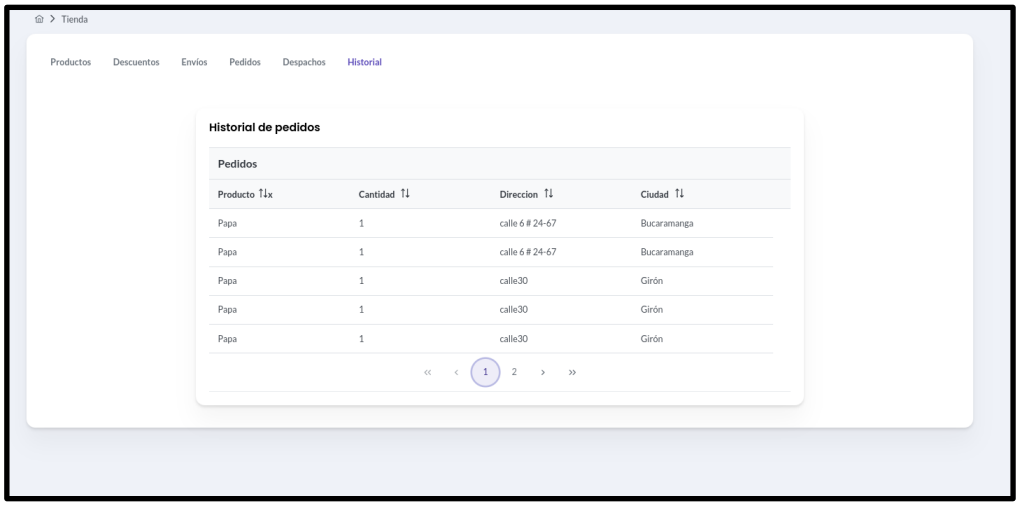
Vista de los despachos.



Nota: *Romero, A., & Sandoval, J.* (2024). Diseño propio.

Figura 49.

Vista del historial.



Nota: *Romero, A., & Sandoval, J.* (2024). Diseño propio.

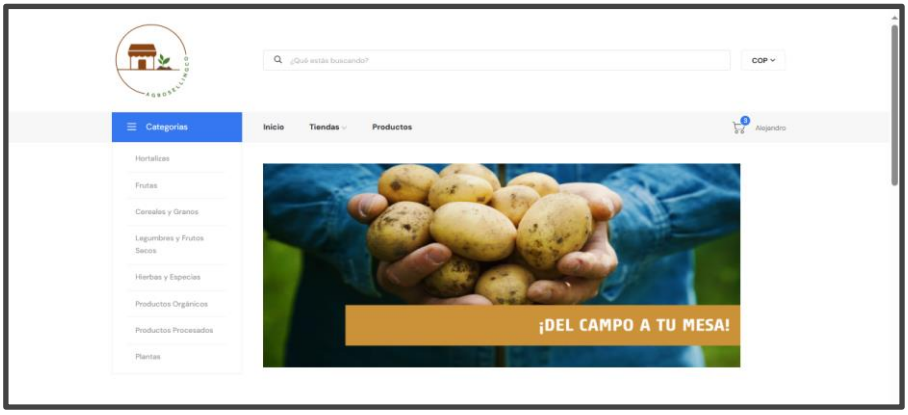
5.3.3.3 Implementación frontend MarketPlace. Este proyecto de frontend se enfoca en módulos para un marketplace, como la visualización de productos, carrito, proceso de compra y perfil de usuario. Utilizamos la plantilla de Etrade en Angular, adaptando y editando su código HTML, CSS y JS para nuestras necesidades.

5.3.3.3.1 Módulo Marketplace acceso libre. Este módulo ofrece componentes públicos que cualquier persona puede explorar sin necesidad de hacer un proceso de autentificación.

5.3.3.3.1.1 Componente inicio del Marketplace. Este componente constituye la página principal de la aplicación, donde los usuarios pueden explorar diversas categorías, acceder a tiendas y productos, así como gestionar su carrito de compras. Cabe destacar que esta funcionalidad forma parte del encabezado de la vista. Además, la vista principal incluye secciones específicas, como 'Productos recién agregados', donde se muestran los productos añadidos en la última semana. También, hay una sección dedicada a productos con descuentos y otra que presenta aleatoriamente diez productos de la tienda.

Figura 50.

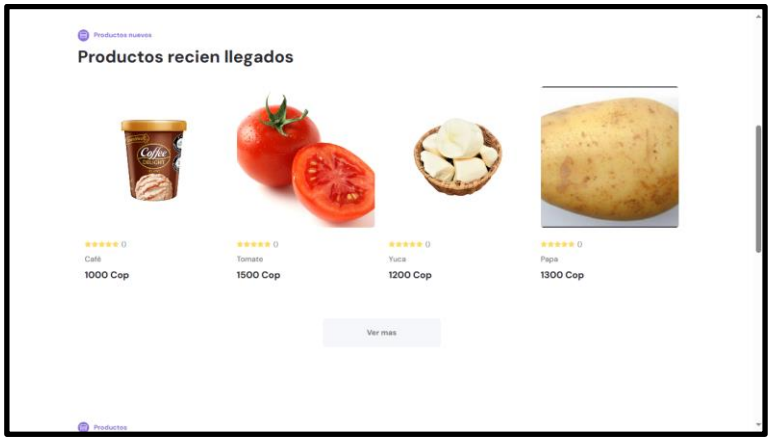
Vista de header del Marketplace.



Nota: *Romero, A., & Sandoval, J.* (2024). Diseño propio.

Figura 51.

Sección de productos recién llegados.



Nota: *Romero, A., & Sandoval, J.* (2024). Diseño propio.

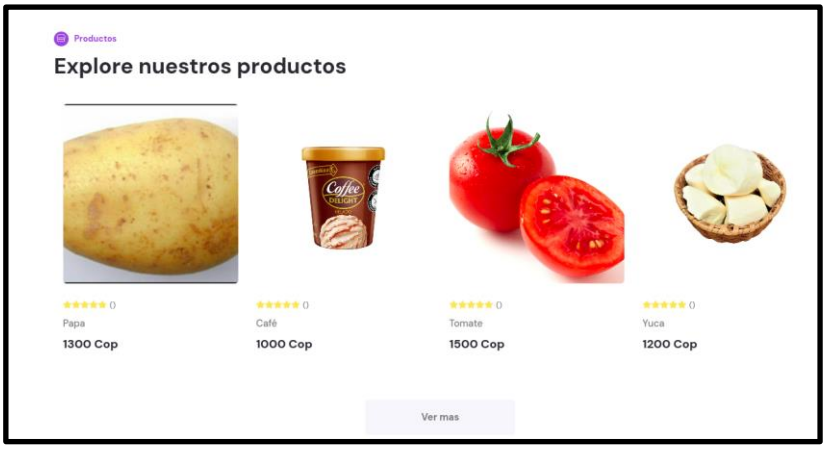
Figura 52.

Sección de descuentos.

Nota: *Romero, A., & Sandoval, J.* (2024). Diseño propio.

Figura 53.

Sección de productos aleatorios.

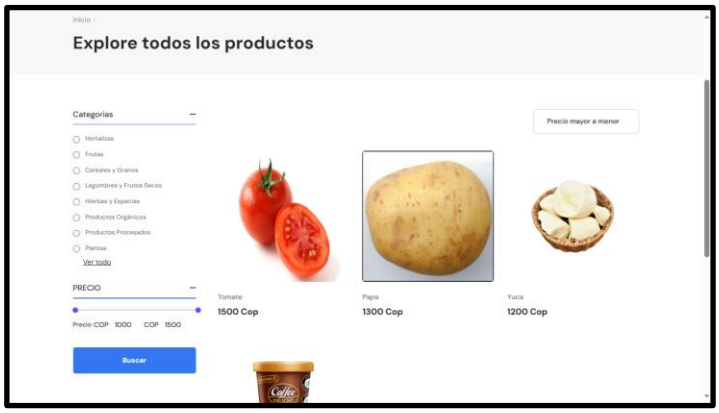


Nota: *Romero, A., & Sandoval, J.* (2024). Diseño propio.

5.3.3.3.1.2 Componente filtrar productos. Este componente ofrece visualizar todos los productos disponibles en el marketplace. Los usuarios tienen la flexibilidad de aplicar filtros para buscar productos por categoría y precio. Además, se proporciona la opción de ordenar los resultados según diferentes criterios, como precio (mayor a menor y viceversa) y nombre (de la A a la Z y de la Z a la A).

Figura 54.

Vista de panel para filtrar productos.

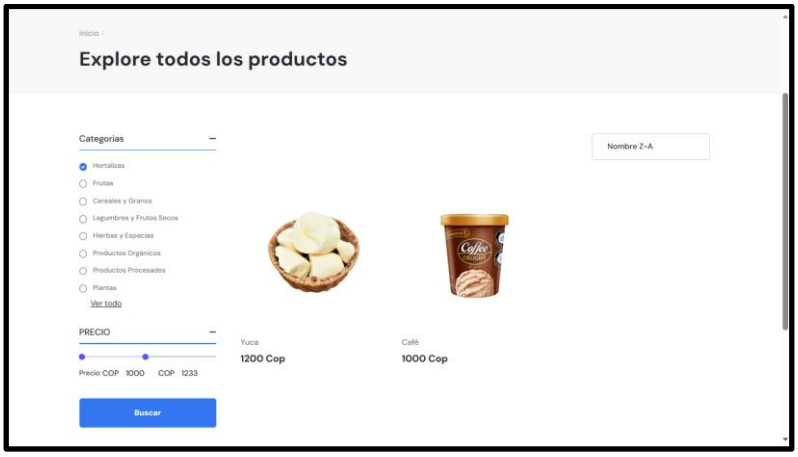


Nota: *Romero, A., & Sandoval, J.* (2024). Diseño propio.

Un ejemplo del filtro es la siguiente imagen, en la que vale la pena identificar cómo se muestran los productos pertenecientes a las hortalizas, que tienen un precio entre 1000 y 1233 COP, y cuyo orden alfabético va de la Z a la A. El resultado se ve así.

Figura 55.

Ejemplo de producto filtrado del panel de los productos filtrados.

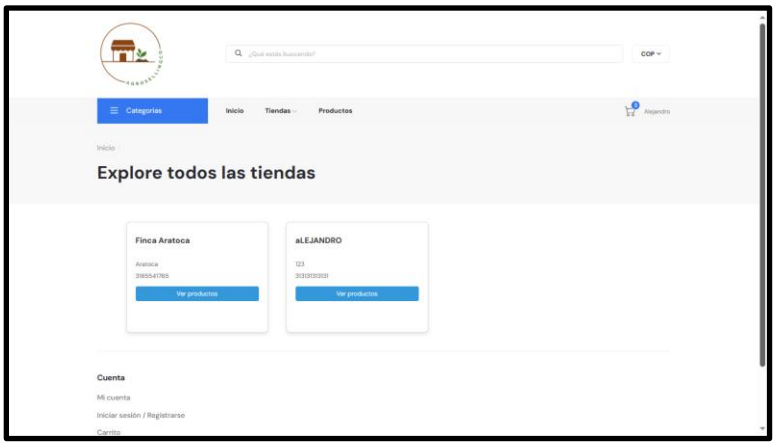


Nota: *Romero, A., & Sandoval, J.* (2024). Diseño propio.

5.3.3.3.1.3 Componente visualizar tiendas. Este componente permite ver todas las tiendas que se encuentran registradas en el marketplace.

Figura 56.

Vista de tiendas existentes.

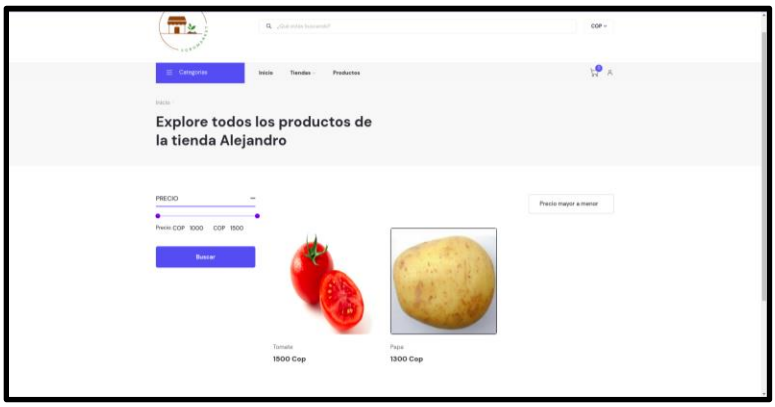


Nota: *Romero, A., & Sandoval, J.* (2024). Diseño propio.

5.3.3.3.1.4 Componente catálogo de productos de la empresa. Este componente ofrece una visualización de todos los productos pertenecientes a una empresa específica.

Figura 57.

Catálogo de productos de la empresa.

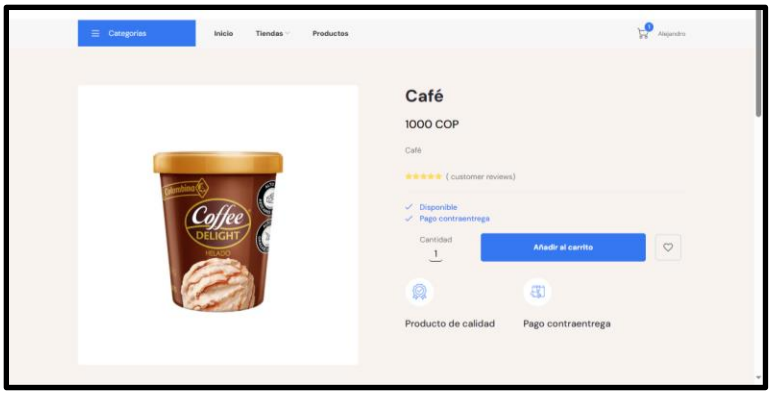


Nota: *Romero, A., & Sandoval, J.* (2024). Diseño propio.

5.3.3.3.1.5 Componente de descripción de un producto. Este componente permite ver la descripción de un producto en específico.

Figura 58.

Interfaz de descripción de producto.

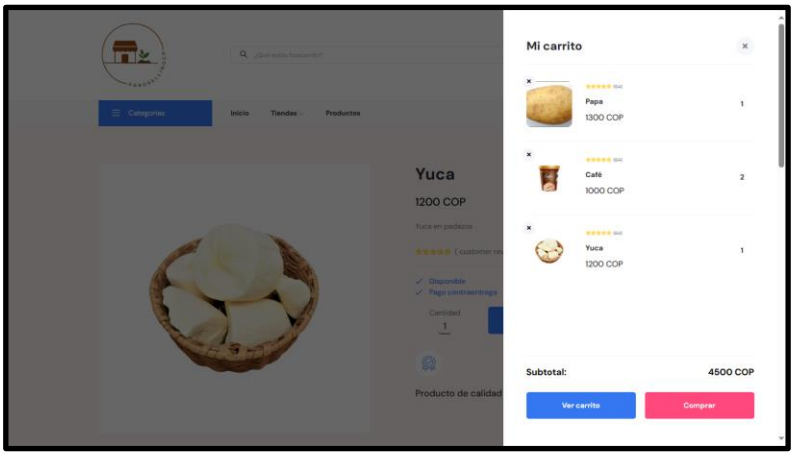


Nota: *Romero, A., & Sandoval, J.* (2024). Diseño propio.

5.3.3.3.1.6 Componente carrito de compras. Este componente permite ver todos los productos añadidos al carrito de compras

Figura 59.

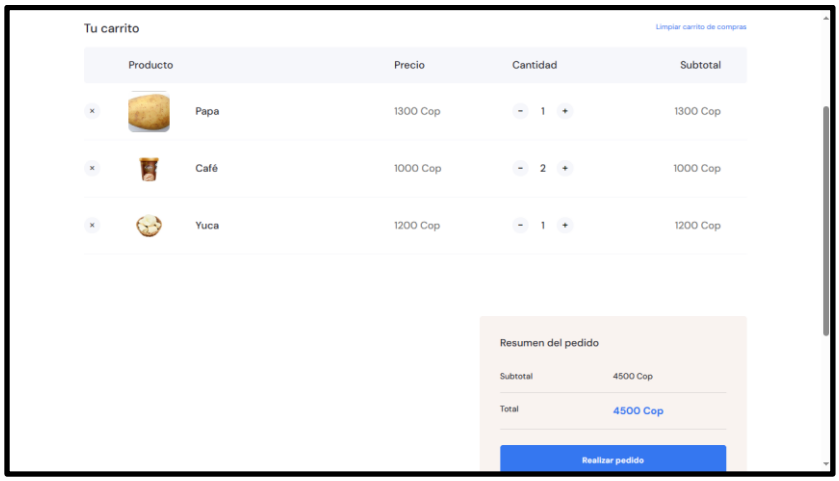
Vista del menú de carrito con los productos seleccionados.



Nota: *Romero, A., & Sandoval, J.* (2024). Diseño propio.

Figura 60.

Vista detalles del carrito con sus productos.



Nota: *Romero, A., & Sandoval, J.* (2024). Diseño propio.

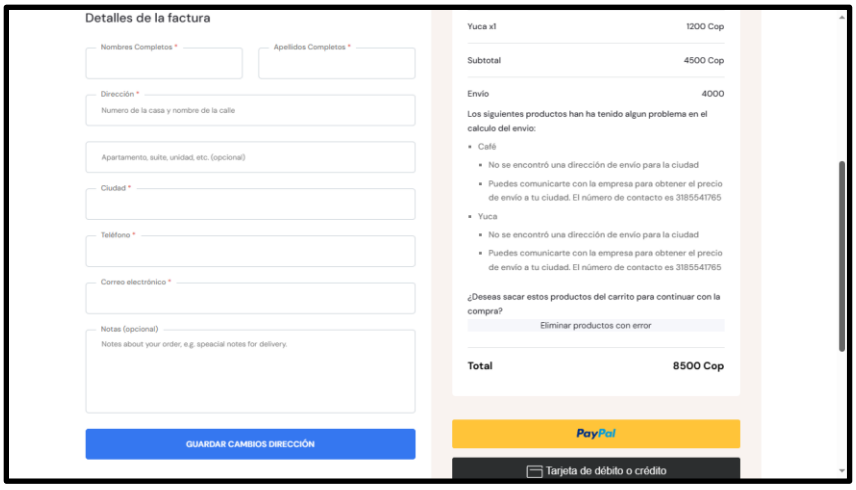
5.3.3.3.2 Módulo Marketplace acceso protegido. Para acceder a las vistas de este módulo es necesario realizar los procesos del módulo de autentificación (registro e inicio de sesión).

5.3.3.3.2.1 Componente detalle factura. Este componente permite observar de manera detallada todos los elementos que componen la factura. Además, brinda al usuario la opción de seleccionar una dirección de envío para sus productos. Previo al proceso de pago (checkout), el usuario deberá calcular el costo del envío, lo cual implica una consulta a la base de datos para verificar la disponibilidad del producto y la existencia de la tienda en la ciudad de destino.

Durante este proceso, se verificará la existencia del producto en la base de datos y se comprobará si la tienda cuenta con servicios de envío hacia la ciudad solicitada por el cliente. En caso de que el producto esté agotado, no esté disponible o la tienda no ofrezca envíos a la ciudad especificada, se notificará al usuario sobre la naturaleza del problema. Asimismo, se proporcionará el número de contacto de la tienda correspondiente para que el usuario pueda comunicarse y resolver la situación.

Figura 61.

Vista detalle factura.



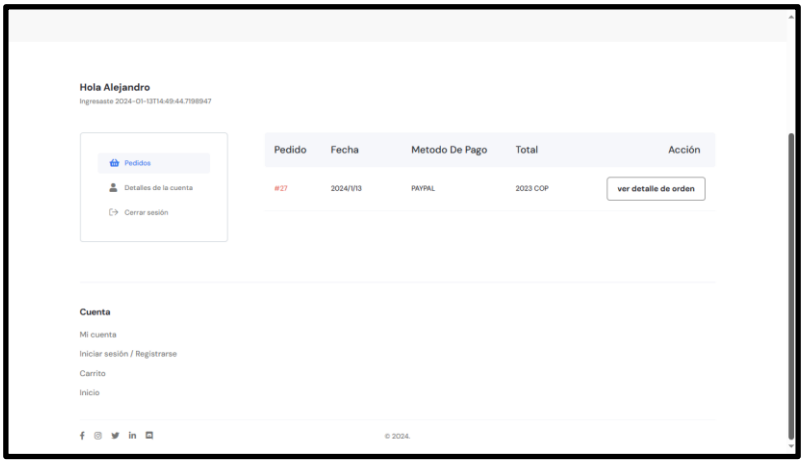
Nota: *Romero, A., & Sandoval, J.* (2024). Diseño propio.

Es relevante señalar que hemos integrado la pasarela de pago PayPal en la sección de pagos con el propósito específico de realizar pruebas en los módulos correspondientes y validar el correcto funcionamiento de estos. Es importante destacar que esta integración se utilizará exclusivamente con fines de prueba

5.3.3.3.2.2 Componente historial de pedidos del usuario. Esta vista permite visualizar el historial de pedidos realizados por un usuario a través del marketplace.

Figura 62.

Vista historial de pedidos del usuario.

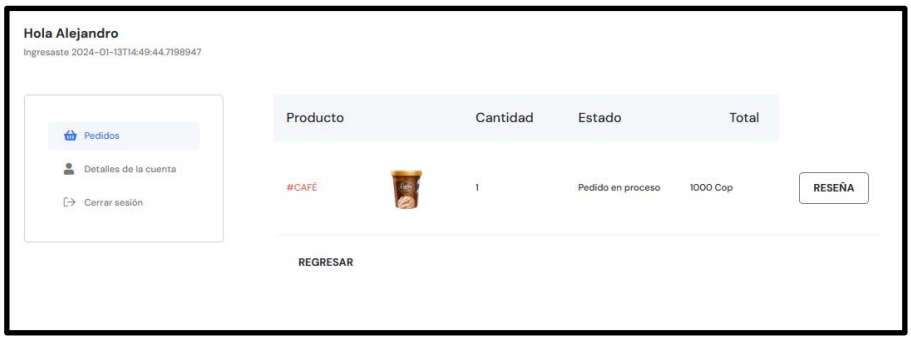


Nota: *Romero, A., & Sandoval, J.* (2024). Diseño propio.

5.3.3.3.2.3 Componente detalle de un pedido. Esta interfaz ofrece una visión detallada de un pedido, permitiendo al usuario observar los productos adquiridos, sus respectivos precios y el estado actual que se encuentra.

Figura 63.

Vista de administración de cuenta en el módulo tienda, apartado de los productos pedidos.

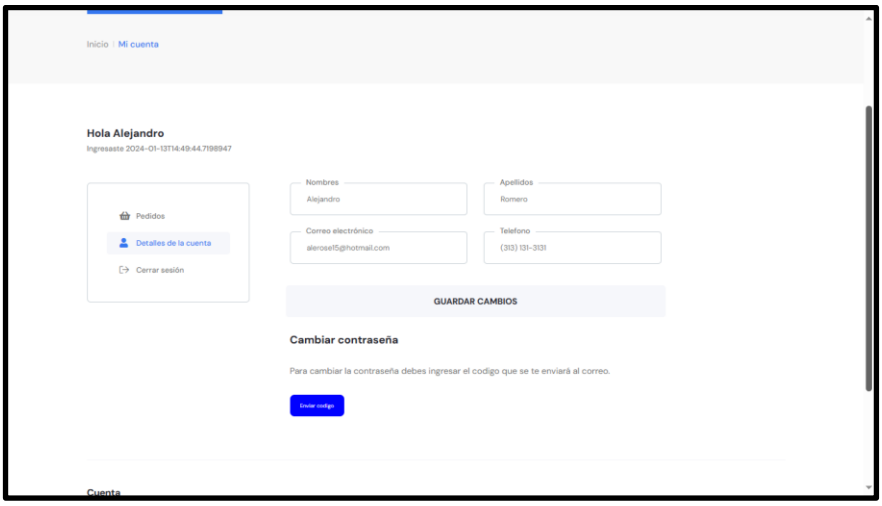


Nota: *Romero, A., & Sandoval, J.* (2024). Diseño propio.

5.3.3.3.2.3 Componente detalle de perfil del usuario. En esta interfaz, el usuario puede editar su información de cuenta.

Figura 64.

Vista detalle de perfil de usuario



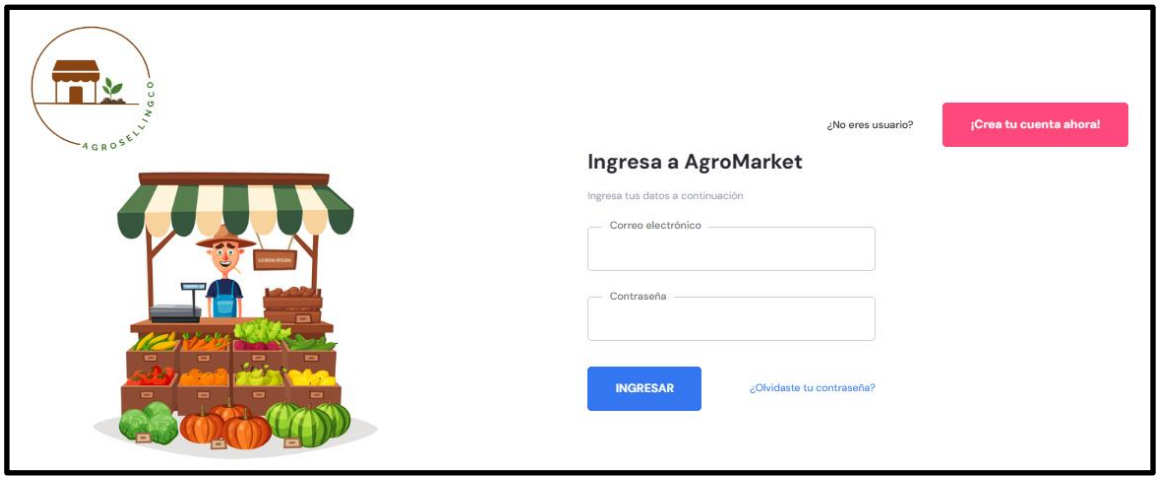
Nota: *Romero, A., & Sandoval, J.* (2024). Diseño propio.

5.3.3.3.3 Módulo de autenticación Marketplace. Este módulo se encarga de los componentes relacionados con la autenticación en el marketplace. Aquí, los usuarios pueden registrarse, recuperar sus cuentas y realizar el inicio de sesión correspondiente

5.3.3.3.2.1 Componente inicio de sesión

Figura 65.

Vista inicio de sesión.

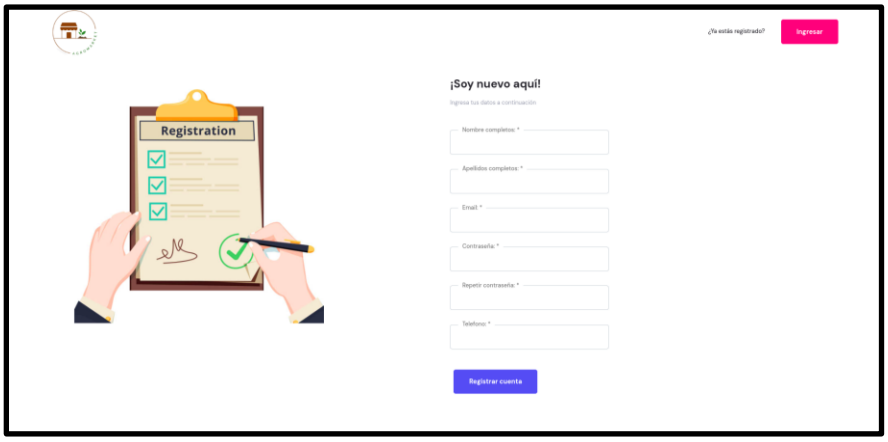


Nota: *Romero, A., & Sandoval, J.* (2024). Diseño propio.

5.3.3.3.3.2 Componente crear cuenta

Figura 66.

Vista crear cuenta.



Nota: *Romero, A., & Sandoval, J.* (2024). Diseño propio.

5.3.3.3.3.3 Componente recuperar cuenta

Para recuperar la cuenta, somos redirigidos a la vista de a continuación.

Figura 67.

Vista recuperar cuenta.

Nota: *Romero, A., & Sandoval, J.* (2024). Diseño propio.

5.3.3.4 Implementación de aplicativo móvil. La aplicación móvil de nuestro proyecto de frontend adoptó el mismo enfoque que el rol de administrador en la aplicación web, abordando módulos clave como la gestión de cultivos, productos, envíos, y descuentos, y la visualización de pedidos y despachos. Este proyecto se inició desde cero, utilizando el lenguaje de programación Dart y su framework Flutter para el desarrollo

5.3.3.4.1 Módulo de autenticación en el aplicativo móvil. Este módulo se centra en la funcionalidad de inicio de sesión dentro de la aplicación móvil.

5.3.3.4.1.1 Inicio de sesión. La implementación del inicio de sesión se realiza a través de un ConsumerWidget de Riverpod. Este widget permite la escucha de proveedores, que son clases encargadas de suministrar datos o servicios a otros componentes. Gracias a esta funcionalidad, la interfaz de usuario se actualiza de manera automática en respuesta a los cambios que puedan ocurrir en el proveedor asociado.

Figura 68.

Vista de inicio de sesión en aplicativo móvil.

Nota: *Romero, A., & Sandoval, J.* (2024). Diseño propio.

5.3.3.4.2 Módulo de administrador en el aplicativo móvil. Este módulo se renderiza después de que el usuario ha ingresado con éxito a la aplicación. Presenta la vista inicial de la aplicación, donde se encuentra un menú de sesión que permite la navegación hacia otras vistas.

5.3.3.4.2.1 Vista de inicio en el aplicativo móvil

Figura 69.

Vista de inicio en el aplicativo móvil.

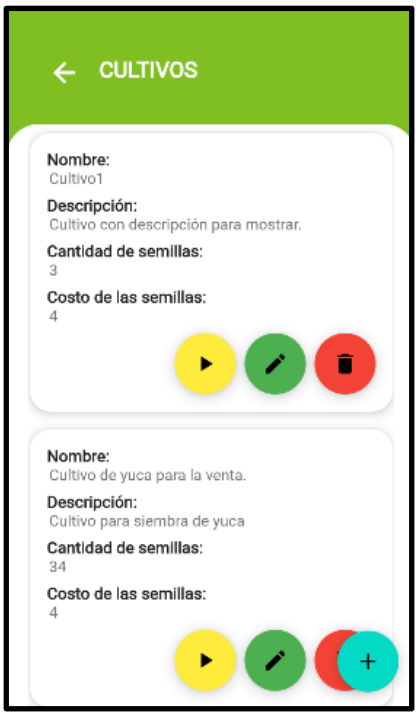
Nota: *Romero, A., & Sandoval, J.* (2024). Diseño propio.

5.3.3.4.3 Módulo de gestión agrícola en el aplicativo móvil. Este módulo abarca todas las actividades relacionadas con la gestión agrícola de una empresa. En esta instancia, se mostrará específicamente la vista de cultivos, mientras que las demás vistas se detallarán en los anexos de los módulos adicionales que hemos desarrollado.

5.3.3.4.3.1 Vista de gestión de cultivos en el aplicativo móvil

Figura 70.

Vista de gestión de los cultivos agrícolas en el aplicativo móvil.



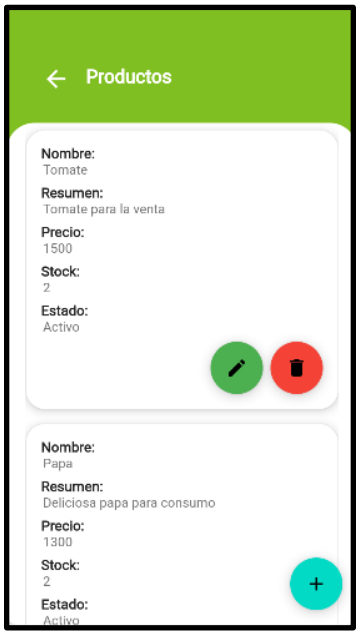
Nota: *Romero, A., & Sandoval, J.* (2024). Diseño propio.

5.3.3.4.4 Módulo de gestión marketplace en el aplicativo móvil. Este módulo se encarga de la administración del marketplace, incluye las vistas de gestión de productos, descuentos, y zonas de envío. También las de visualización de despachos, pedidos.

5.3.3.4.4.1 Vista de gestión de productos en el aplicativo móvil. Los productos que se van a comercializar también pueden ser gestionados por el usuario desde el aplicativo móvil mediante esta vista.

Figura 71.

Vista de gestión de los productos agrícolas en el aplicativo móvil.

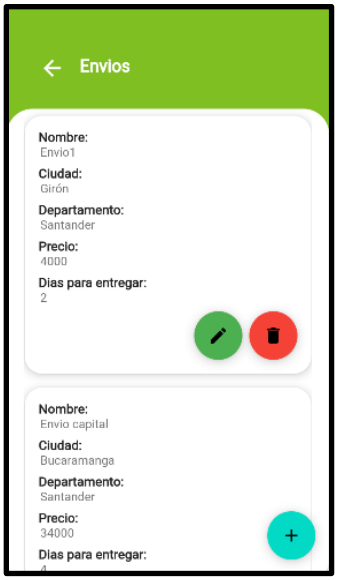


Nota: *Romero, A., & Sandoval, J.* (2024). Diseño propio.

5.3.3.4.4.2 Vista de gestión de envíos en el aplicativo móvil. La misma situación se presenta para los envíos, desde la vista de gestión de envíos en el panel principal donde podremos administrarlos.

Figura 72.

Vista de gestión de los envíos en el aplicativo móvil.

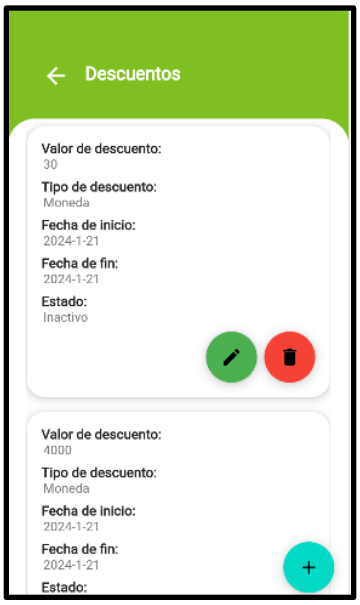


Nota: *Romero, A., & Sandoval, J.* (2024). Diseño propio.

5.3.3.4.4.2 Vista de gestión de descuentos en el aplicativo móvil. Para los descuentos trabajamos de forma similar.

Figura 73.

Vista de gestión de los descuentos en el aplicativo móvil.

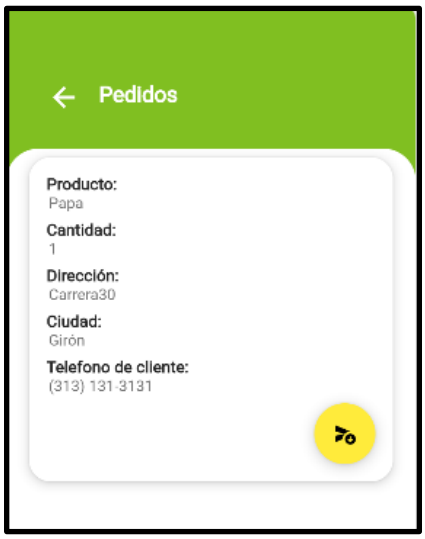


Nota: *Romero, A., & Sandoval, J.* (2024). Diseño propio.

Los pedidos generados después de realizada la compra por parte de los clientes también tienen su apartado, pero este es única y exclusivamente de visualización. La única opción disponible para el administrador es despachar este pedido.

Figura 74.

Vista de los pedidos en el aplicativo móvil.



Nota: *Romero, A., & Sandoval, J.* (2024). Diseño propio.

Los pedidos despachados desde la vista anterior tienen que aparecer acordemente en la lista de visualización de los despachos.

Figura 75.

Vista de los despachos en el aplicativo móvil.

Nota: *Romero, A., & Sandoval, J.* (2024). Diseño propio.

Los despachos resultantes quedaron en esta lista de visualización, con la opción de verificar si estos ya fueron entregados a los clientes. En caso de ser así, saldrán de la lista.

5.4 Fase de pruebas

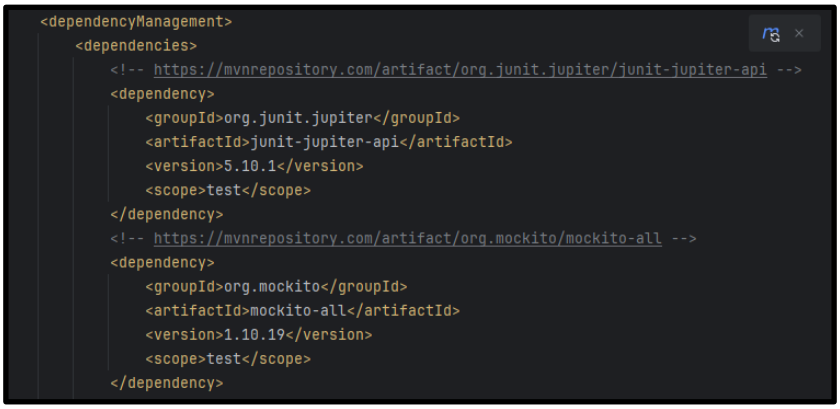
Esta fase consta del desarrollo de las pruebas en el contexto backend y frontend, teniendo en cuenta que estas son clasificadas como unitarias, de integración y de sistema.

5.4.1 Pruebas unitarias y de integración

5.4.1.1 Pruebas unitarias y de integración en Backend. Después de todas las fases de desarrollo llegamos a la de pruebas, donde inicialmente nos centramos en hacer las pruebas unitarias y de integración del Backend apoyado en Spring Boot. Pero antes de eso, fue necesario importar las dependencias de JUnit y Mockito en el archivo pom.xml.

Figura 76.

Importación de dependencias para testing.



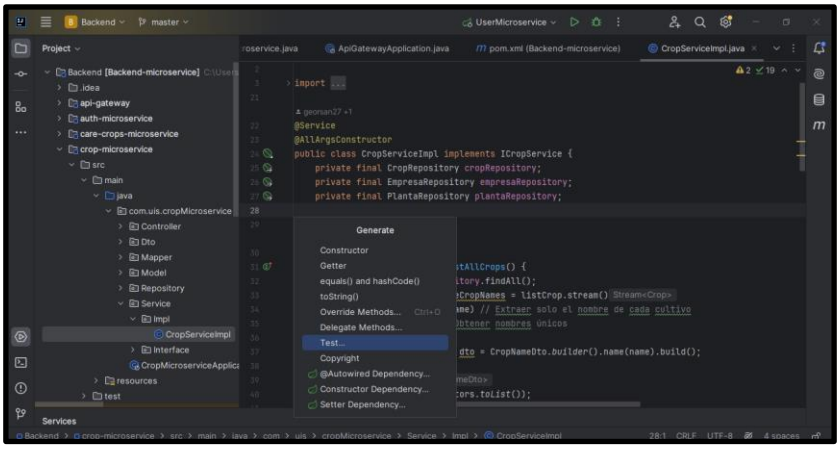
```
<dependencyManagement>
    <dependencies>
        <!-- https://mvnrepository.com/artifact/org.junit.jupiter/junit-jupiter-api -->
        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter-api</artifactId>
            <version>5.10.1</version>
            <scope>test</scope>
        </dependency>
        <!-- https://mvnrepository.com/artifact/org.mockito/mockito-all -->
        <dependency>
            <groupId>org.mockito</groupId>
            <artifactId>mockito-all</artifactId>
            <version>1.10.19</version>
            <scope>test</scope>
        </dependency>
```

Nota: *Romero, A., & Sandoval, J.* (2024). Diseño propio.

Una vez realizada la importación de dependencias, podemos continuar con la generación de pruebas en los archivos de service y de controladores. Para ello entramos a cada microservicio y dentro de cada archivo del directorio service generamos una prueba con clic derecho sobre el código, donde tomamos todos los métodos que deseamos testear.

Figura 77.

Generación de pruebas en archivos de service.

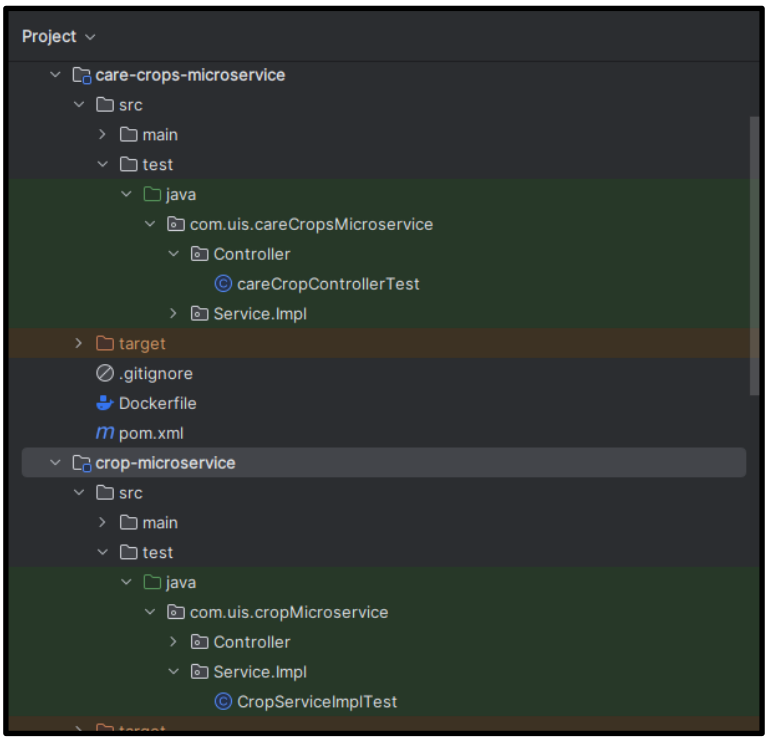


Nota: *Romero, A., & Sandoval, J.* (2024). Diseño propio.

El resultado de esta creación debía terminar con un nuevo directorio de test junto al original main, que era donde se encontraba el archivo correspondiente a las pruebas.

Figura 78.

Directorio post creación de pruebas.



Nota: *Romero, A., & Sandoval, J.* (2024). Diseño propio.

Una vez completada la creación de estos archivos se procedió a realizar la prueba, importando las clases necesarias junto a la librería Mockito y las bibliotecas JUnit. En el siguiente ejemplo podremos apreciar mejor la importación de estas clases. El @Mock se utiliza para crear objetos simulados que imitan comportamientos de objetos reales dentro del sistema.

Figura 79.

Ejemplo de prueba en Spring Boot.

```
Alejandro RS
@SpringBootTest
@AutoConfigureMockMvc
class CropControllerTest {

    @Autowired
    private MockMvc mockMvc;

    no usages
    @Mock
    private ICropService cropService;

    5 usages
    @InjectMocks
    private CropController cropController;

    Alejandro RS
    @Test
    void getAllCropsByEmpresaId() throws Exception {
        Long empresaId = 1L;
```

Nota: *Romero, A., & Sandoval, J.* (2024). Diseño propio.

Un ejemplo de la realización de las pruebas es el que tenemos en el microservicio Crop, en el método para obtener el cultivo por el id de empresa, donde apoyados con una clase de Spring conocida como MockMvc, simulamos una solicitud HTTP y verificamos que esta se haya cumplido satisfactoriamente.

Figura 80.

Ejemplo de prueba de integración en desarrollo Backend.

```
@Test
void getAllCropsByEmpresaId() throws Exception {
    Long empresaId = 1L;

    mockMvc.perform(MockMvcRequestBuilders.get( urlTemplate: "/api/crop/" + empresaId)
                    .contentType(MediaType.APPLICATION_JSON))
            .andExpect(status().isOk())
            .andExpect(content().contentType(MediaType.APPLICATION_JSON))
            .andExpect(jsonPath( expression: "$.crops").isArray())
            .andExpect(jsonPath( expression: "$.crops[0].id").value( expectedValue: 1))
            .andExpect(jsonPath( expression: "$.crops[0].name").value( expectedValue: "Prueb"))
            .andExpect(jsonPath( expression: "$.crops[0].description").value( expectedValue: "Prueba"))
            .andExpect(jsonPath( expression: "$.crops[0].cantidad_semillas").doesNotExist())
            .andExpect(jsonPath( expression: "$.crops[0].costo_semillas").doesNotExist());
}
```

Nota: *Romero, A., & Sandoval, J.* (2024). Diseño propio.

Con todas las pruebas en cada uno de los controladores y servicios de los microservicios podemos dar por completadas las pruebas unitarias y de integración en nuestro backend.

5.4.1.2 Pruebas unitarias en Frontend. Para la realización de las pruebas unitarias en el frontend, tenemos que ir a cada uno de los componentes en su archivo 'spec', donde realizaremos las verificaciones de los métodos existentes. Un ejemplo válido es la siguiente figura, en la que se verifica que, al abrir el método para editar un envío, el formulario de edición este tenga sus valores iniciales.

Figura 81.

Ejemplo de prueba unitaria en el desarrollo frontend.

```
it('should set shipping and populate form on editShipping()', () => {
  const mockShipping = {
    id: '1',
    department: 'Department1',
    city: 'City1',
    name: 'Shipping1',
    days_to_delivery: 3,
    price: 10.0,
  };
  component.shippingUpdateDialog = false;
  component.shippingUpdateForm.setValue({
    department: '',
    city: '',
    name: '',
    days: '',
    price: '',
  });

  component.editShipping(mockShipping);

  expect(component.shippingUpdateDialog).toBeTrue();
  expect(component.shippingUpdateForm.value).toEqual({
    department: 'Department1',
    city: 'City1',
    name: 'Shipping1',
    days: 3,
    price: 10.0,
  });
  expect(component.idShipping).toBe('1');
});
```

Nota: *Romero, A., & Sandoval, J.* (2024). Diseño propio.

Figura 82.

Ejecución de las pruebas realizadas en Karma.

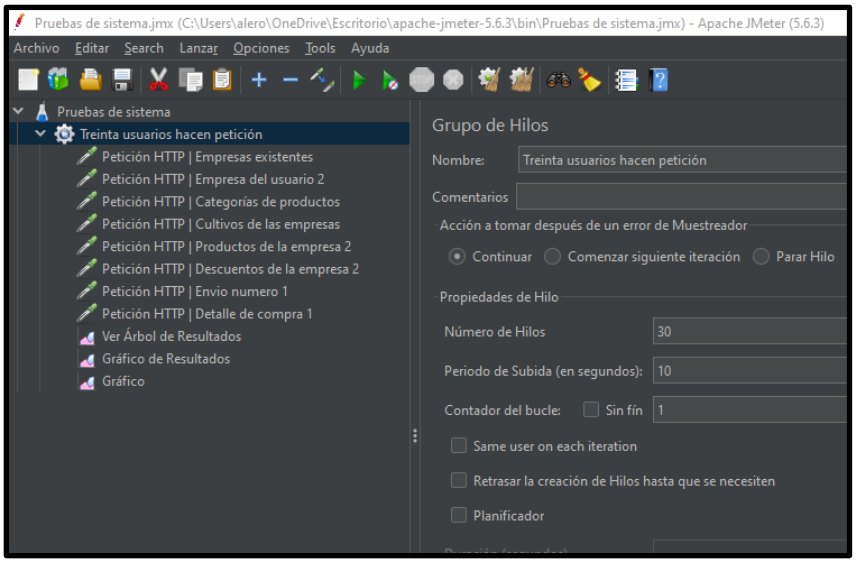
Nota: *Romero, A., & Sandoval, J.* (2024). Diseño propio.

5.4.2 Pruebas de sistema

5.4.2.1 Pruebas de carga. Las pruebas de carga se realizaron elaborando las solicitudes HTTP más importantes del sistema dentro del programa JMeter en un plan de pruebas, más específicamente en un grupo de hilos, donde configuramos la cantidad de usuarios (hilos) y el tiempo en la que se trabajara la simulación de estas solicitudes. En nuestro caso de ejemplo, definimos 30 usuarios por 10 segundos.

Figura 83.

Configuración de las pruebas de carga.

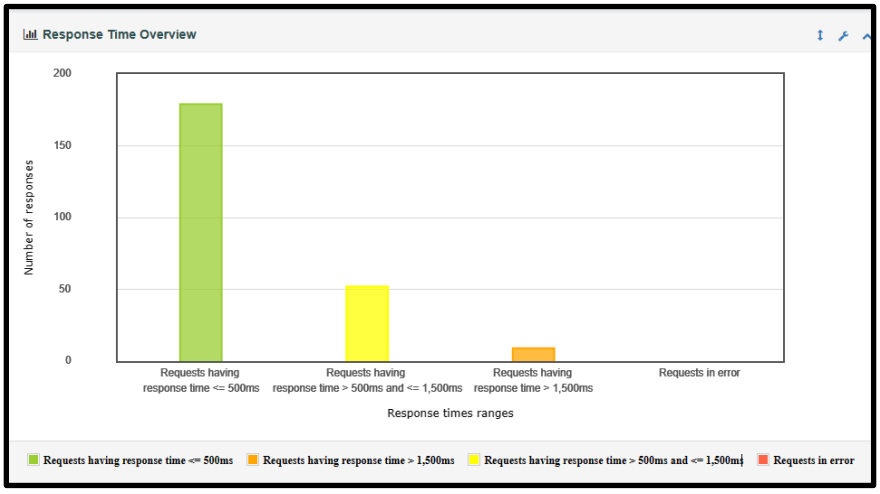


Nota: *Romero, A., & Sandoval, J.* (2024). Diseño propio.

Una vez configurado el grupo de hilos para su ejecución procedemos a completar la prueba de carga y a obtener los resultados. Estos resultados se representan en el tiempo promedio de respuesta (de menor a mayor) y las cantidades de estas respuestas. Podemos apreciar cómo la mayoría de las respuestas suceden de forma rápida y no existen errores en las peticiones.

Figura 84.

Gráfico de resultados de la prueba de carga.

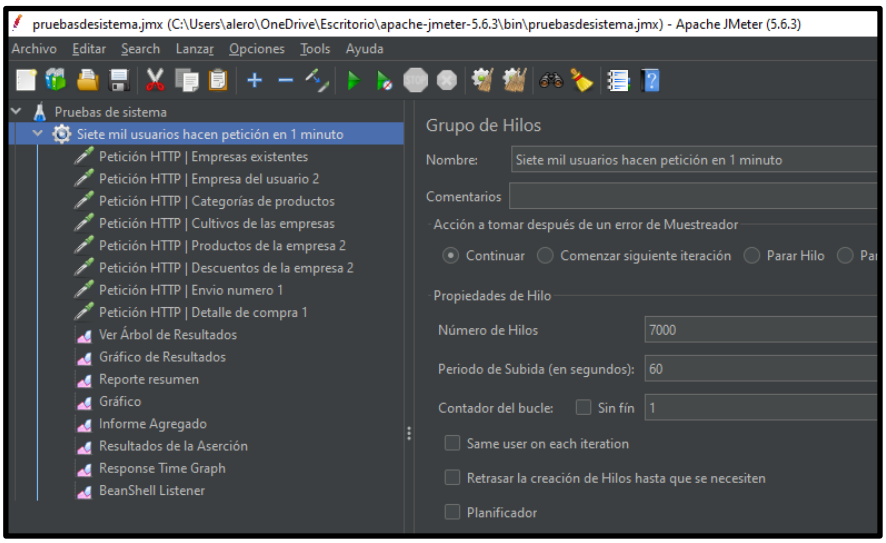


Nota: *Romero, A., & Sandoval, J.* (2024). Diseño propio.

5.4.2.2 Pruebas de estrés. Para el caso de las pruebas de estrés del sistema, mantuvimos las solicitudes HTTP, pero se ejecutaron con una configuración de 7000 hilos con un tiempo de 60 segundos.

Figura 85.

Configuración de las pruebas de estrés.

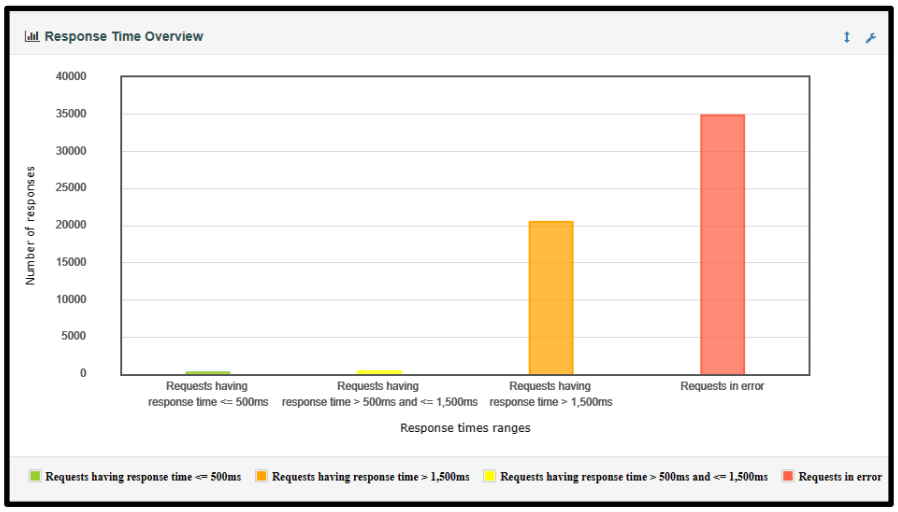


Nota: *Romero, A., & Sandoval, J.* (2024). Diseño propio.

Los resultados demostraron cómo la duración de respuesta de las solicitudes incrementó y hubo mayoría de errores de ejecución. Sin embargo, las respuestas con tardanza se ejecutaron exitosamente.

Figura 86.

Gráficos de resultado de la prueba de estrés.



Nota: *Romero, A., & Sandoval, J.* (2024). Diseño propio.

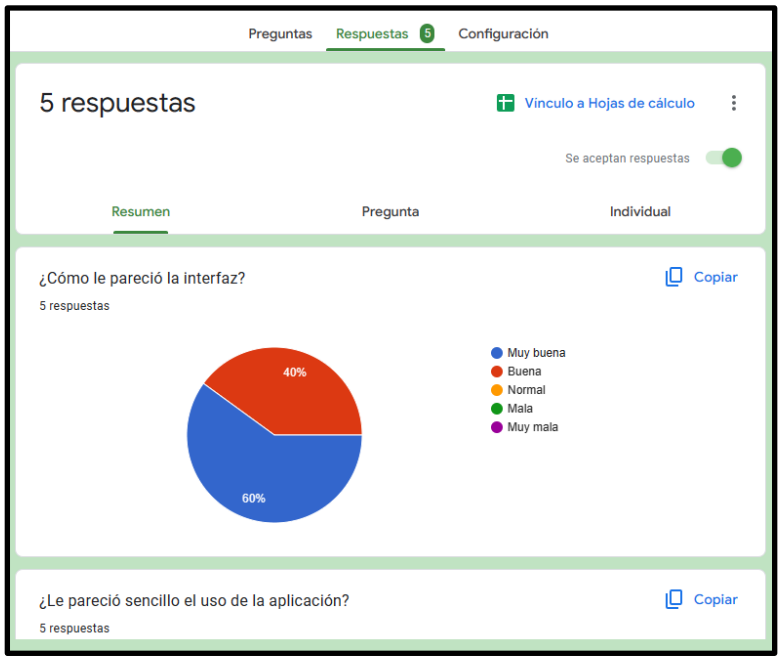
5.5 Encuesta

Por otra parte, se hace una encuesta para asegurarnos de la experiencia de algunos usuarios (5) que utilizaron nuestro aplicativo. En tres preguntas diferentes, encontramos las siguientes respuestas:

Donde los usuarios respondieron asertivamente al sobre una opinión de la interfaz con tres respuestas como ‘Muy buena’ y dos respuestas como ‘Buena’.

Figura 87.

Respuestas de la encuesta parte 1.



Nota: *Romero, A., & Sandoval, J.* (2024). Diseño propio.

Por otra parte, todos los usuarios encuestados respondieron con ‘Sí’ a la pregunta de si les pareció sencillo el uso de la aplicación. Solo una persona escribió en el recuadro de recomendaciones, pero solo nos añadió un comentario positivo.

Figura 88.

Respuestas de la encuesta parte 2.

Nota: *Romero, A., & Sandoval, J.* (2024). Diseño propio.

6. Módulos adicionales

Estos módulos fueron desarrollados fuera de los objetivos iniciales de este proyecto; sin embargo, ofrecen funcionalidades adicionales que resultan beneficiosas para que los agricultores puedan llevar un registro detallado de diversos aspectos, como plagas, tratamientos, variables ambientales y cuidados especiales. Esta información se considera crucial y es fundamental tenerla fácilmente accesible. Por esta razón, se crearon estos módulos con el propósito de ampliar la plataforma, de manera que no solo asista en un marketplace si no que atienda aspectos agrícolas.

6.1 Fase de análisis

6.1.1 Requerimientos

Como se había explicado en el proyecto principal, estos requerimientos están divididos en dos categorías: funcionales, que se encargan de especificar las acciones del sistema y los no funcionales, que hacen lo mismo con el comportamiento.

6.1.1.1 Requerimientos funcionales

Tabla 25.

Requerimientos funcionales para gestionar las plagas.

NOMBRE	Gestionar plagas.
DESCRIPCIÓN	El sistema, en su aplicación web y móvil, debe permitir a los administradores de la empresa crear, actualizar, eliminar y visualizar las plagas que pertenecen a un cultivo.
CRITERIOS DE ACEPTACIÓN	● El sistema debe permitir que el administrador de la empresa ingrese un nombre, descripción, estado, observaciones, fecha y familia de la plaga que desea crear. ● Las plagas creadas o actualizadas por el administrador de la empresa deben tener un nombre único. En caso de no ser así, se mostrará un mensaje de error. ● Los campos del formulario que son obligatorios deben ser rellenados para completar el registro. Si no, no se permitirá y se notificará inmediatamente. ● El sistema debe permitir visualizar las plagas asociadas a su cultivo específico. ● El sistema permite actualizar el nombre, descripción, fecha, familia, estado y observaciones de sus plagas. ● Una plaga no puede ser eliminada si tiene otros registros relacionados como tratamientos, además de mostrar su mensaje de error respectivo. En caso de no tener más registros asociados si se permite su eliminación.
PRIORIDAD	Baja.

Tabla 26.

Requerimientos funcionales para gestionar los tratamientos de las plagas.

NOMBRE	Gestionar tratamientos.
DESCRIPCIÓN	El sistema, en su aplicación web y móvil, debe permitir a los administradores de la empresa crear, actualizar, eliminar y visualizar los tratamientos que se pueden estar manejando sobre una plaga.
CRITERIOS DE ACEPTACIÓN	● El sistema debe permitir que el administrador de la empresa ingrese un nombre, descripción, forma, estado, fechas de inicio y fin del tratamiento que desea crear. ● Los tratamientos creados o actualizados por el administrador de la empresa deben tener un nombre único. En caso de no ser así, se mostrará un mensaje de error. ● Los campos del formulario que son obligatorios deben ser rellenados para completar el registro. Si no, no se permitirá y se notificará inmediatamente. ● El sistema debe permitir visualizar los tratamientos asociadas a su plaga específica. ● El sistema permite actualizar el nombre, descripción, forma, estado y fechas de sus tratamientos.
PRIORIDAD	Baja.

Tabla 27.

Requerimientos funcionales para gestionar las variables ambientales.

NOMBRE	Gestionar variables ambientales.
DESCRIPCIÓN	El sistema, en su aplicación web y móvil, debe permitir a los administradores de la empresa crear, actualizar, eliminar y visualizar las variables ambientales que están presentes en un cultivo.
CRITERIOS DE ACEPTACIÓN	● El sistema debe permitir que el administrador de la empresa ingrese un nombre, descripción, método de toma, fecha de obtención y el instrumento con el que se toma la variable que desea crear. ● Las variables creadas o actualizadas por el administrador de la empresa deben tener un nombre único. En caso de no ser así, se mostrará un mensaje de error. ● Los campos del formulario que son obligatorios deben ser rellenados para completar el registro. Si no, no se permitirá y se notificará inmediatamente. ● El sistema debe permitir visualizar las variables asociadas a su cultivo específico. ● El sistema permite actualizar el nombre, descripción, método de toma, fecha de obtención y el instrumento de sus variables. ● Una variable no puede ser eliminada si tiene otros registros relacionados como medidas tomadas, además de mostrar su mensaje de error respectivo. En caso de no tener más registros asociados si se permite su eliminación.
PRIORIDAD	Baja.

Tabla 28.

Requerimientos funcionales para gestionar las medidas de las variables.

NOMBRE	Gestionar medidas ambientales.
DESCRIPCIÓN	El sistema, en su aplicación web y móvil, debe permitir a los administradores de la empresa crear, actualizar, eliminar y visualizar las medidas ambientales que están presentes en una variable tomada.
CRITERIOS DE ACEPTACIÓN	● El sistema debe permitir que el administrador de la empresa ingrese un valor, descripción, fecha de toma y unidad con el que se trabaja la medida que desea crear. ● Los campos del formulario que son obligatorios deben ser rellenados para completar el registro. Si no, no se permitirá y se notificará inmediatamente. ● El sistema debe permitir visualizar las medidas asociadas a su variable ambiental específica. ● El sistema permite actualizar el valor, descripción, fecha de toma y unidad de sus medidas.
PRIORIDAD	Baja.

6.1.1.2 Requerimientos NO funcionales

Tabla 29.

Requerimientos no funcionales base aplicados en los módulos extra y su descripción.

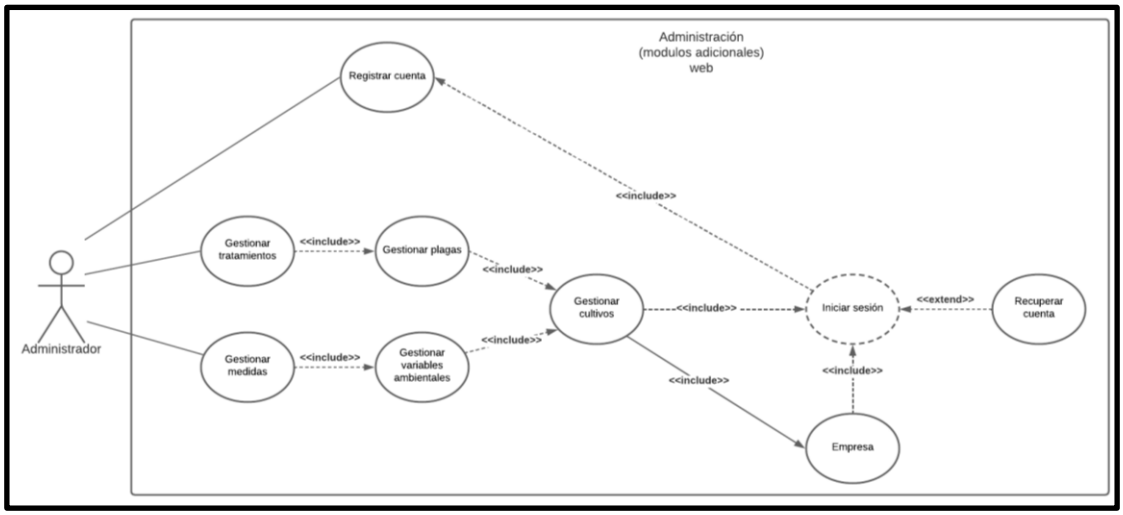
Requerimiento NO funcional	Descripción
1. Usabilidad	La aplicación debe tener una interfaz intuitiva y fácil de usar.
2. Compatibilidad.	La aplicación debe ser accesible de forma web y móvil.
3. Mantenibilidad	Debe permitirse realizar correcciones de errores y mejora de módulos sin mayor dificultad.
4. Seguridad	Los usuarios solo pueden acceder a ciertas funciones a dependencia de los roles.

6.1.2 Diagramas de uso

6.1.2.1 Diagrama de uso de administrador en web

Figura 89.

Diagrama de uso de administrador en web.

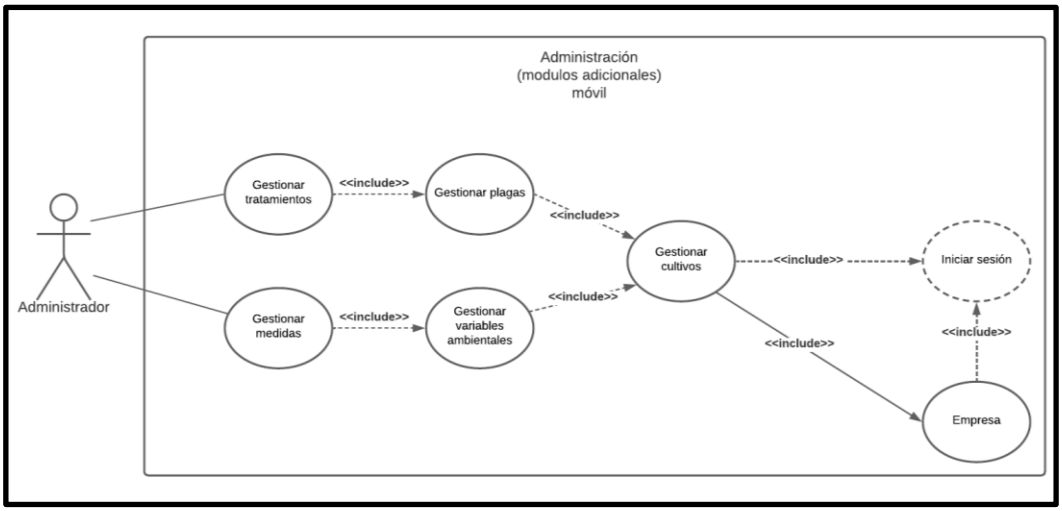


Nota: *Romero, A., & Sandoval, J.* (2024). Diseño propio.

6.1.2.2 Diagrama de uso de administrador en móvil

Figura 90.

Diagrama de uso de administrador en móvil.



Nota: *Romero, A., & Sandoval, J.* (2024). Diseño propio.

Igual que en el proyecto, con el diagrama de uso se busca representar el alcance y objetivo de las interacciones entre el actor y el sistema en estas funcionalidades.

6.2 Fase de diseño

En esta fase, se presenta una visión del diseño de los módulos extra del sistema, tomando en cuenta aspectos clave que incluyen la arquitectura de microservicios para el backend de estos módulos, así como el diagrama del modelo de datos empleado. Cabe aclarar que se maneja la misma estructura de directorios, el mismo gestor de base de datos, los mismos lenguajes y frameworks. La arquitectura móvil se mantiene.

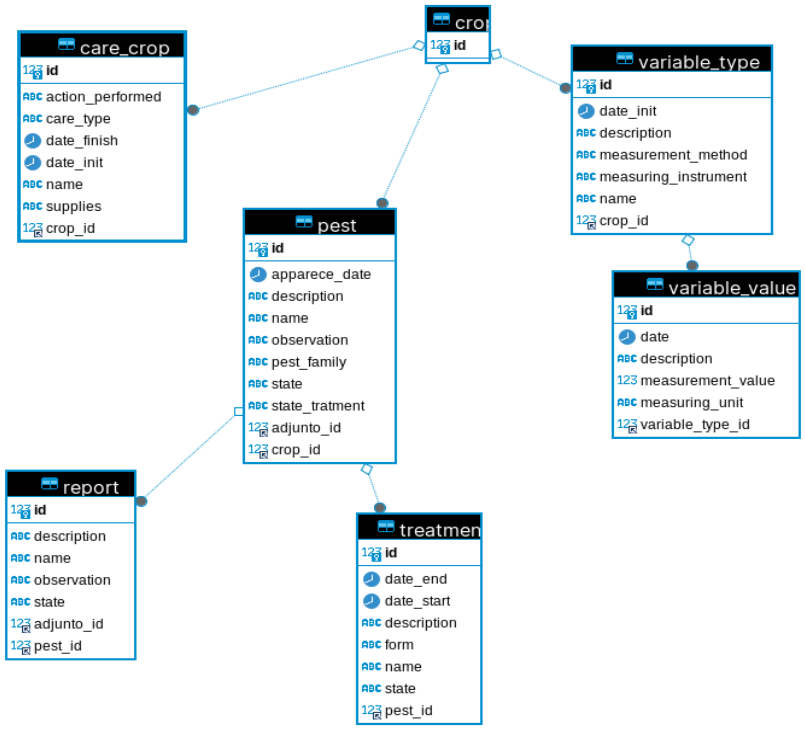
6.2.1 Diseño de la base de datos

La base de datos original del proyecto se mantuvo, pero se amplió con nuevas tablas para respaldar las funcionalidades adicionales. Estas tablas específicas para el registro de variables, plagas, tratamientos y cuidados especiales se mostrarán en el diagrama.

Figura 91.

Diagrama del modelo de datos.

6.2.1.1 Diagrama del modelo de datos



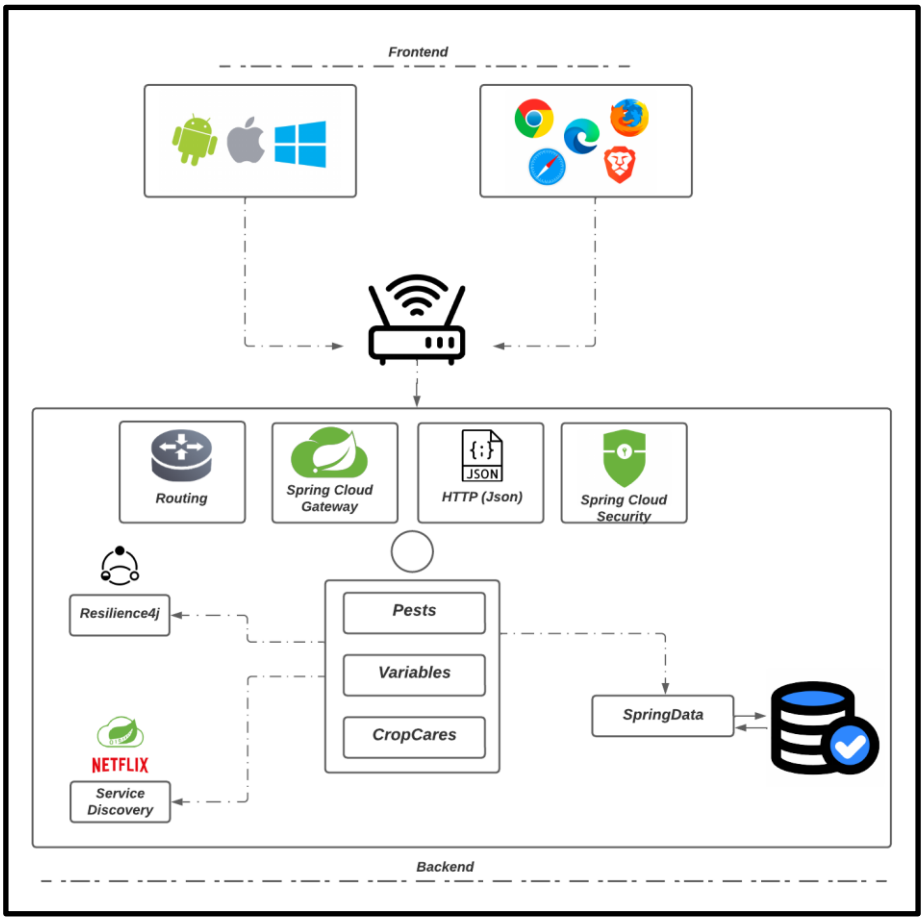
Nota: *Romero, A., & Sandoval, J.* (2024). Diseño propio.

6.2.2 Diseño del backend

6.2.2.1 Arquitectura backend. Los microservicios recién añadidos han sido integrados en la arquitectura mencionada anteriormente. En la siguiente figura, se presentan los microservicios recién creados.

Figura 92.

Arquitectura de Microservicios.



Nota: *Romero, A., & Sandoval, J.* (2024). Diseño propio.

6.3 Fase de implementación

6.3.1 Implementación del backend

Los nuevos microservicios trabajados en esta implementación fueron los siguientes:

6.3.1.1 Microservicio pest. Este microservicio se encarga de la gestión integral de plagas y de los tratamientos asociados aplicados para controlar dichas plagas.

6.3.1.2 Microservicio variables. Este microservicio se encarga de gestionar las variables ambientales ingresadas manualmente, así como los valores de las medidas asociadas a dichas variables.

6.3.1.3 Microservicio CropCares. Este microservicio se encarga de gestionar todos los aspectos relacionados con los cuidados especiales aplicados a un cultivo.

6.3.2 Implementación del frontend

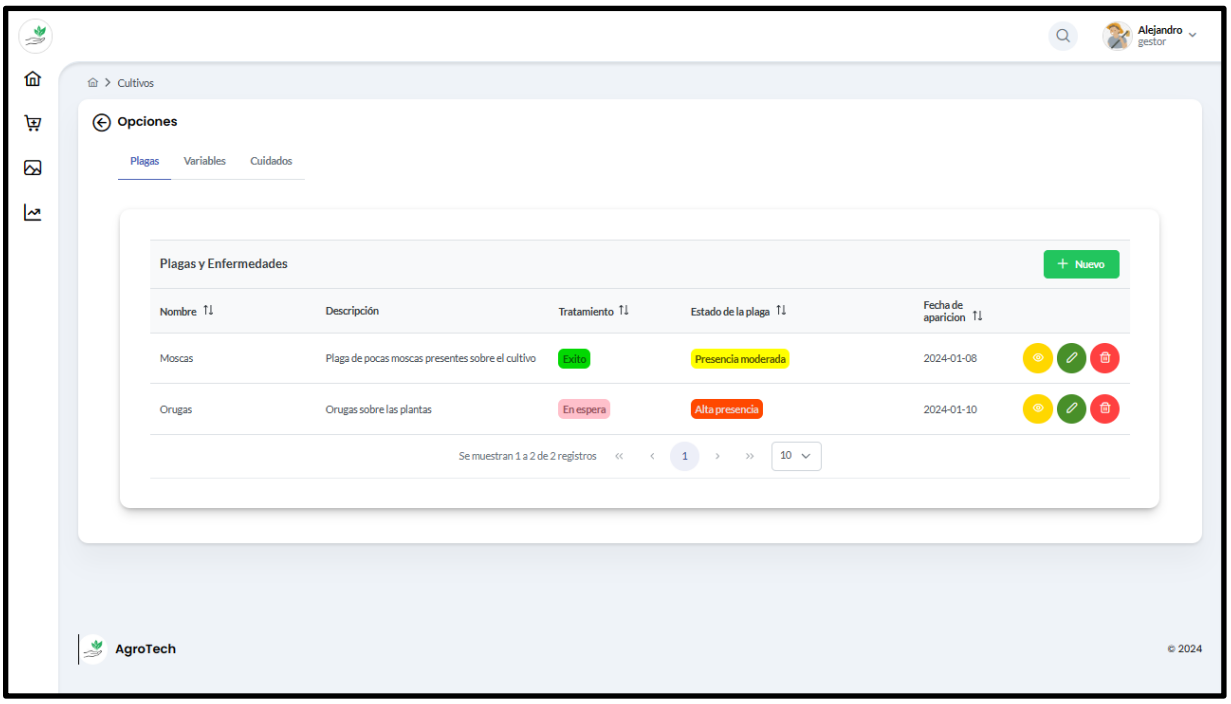
Estos nuevos módulos se integraron al proyecto de Angular destinado al administrador. Las incorporaciones incluyen:

6.3.2.1 Módulo gestión de plagas. En este módulo, posterior al ingreso por los cultivos que tiene nuestra empresa, se mostrará una interfaz de gestión de plagas con tres opciones posibles en la parte superior: las mismas plagas, variables ambientales y cuidados del cultivo.

6.3.2.1.1 Componente de gestión de plagas. Este componente se encarga de la gestión de plagas presente en un cultivo.

Figura 93.

Vista de gestión de las plagas presentes en un cultivo.

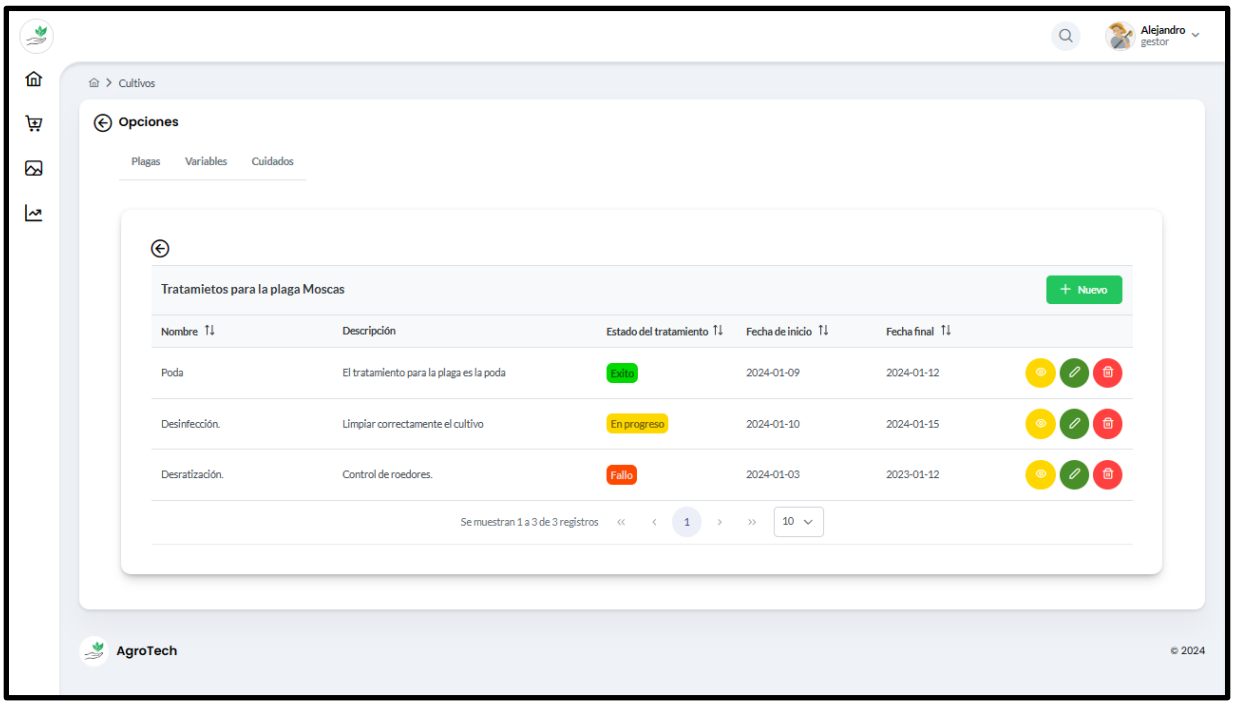


Nota: *Romero, A., & Sandoval, J.* (2024). Diseño propio.

6.3.2.1.2 Componente de gestión de tratamientos. Esta componente se encarga de gestionar los tratamientos aplicados a una plaga específica

Figura 94.

Vista de gestión de los tratamientos presentes en una plaga.



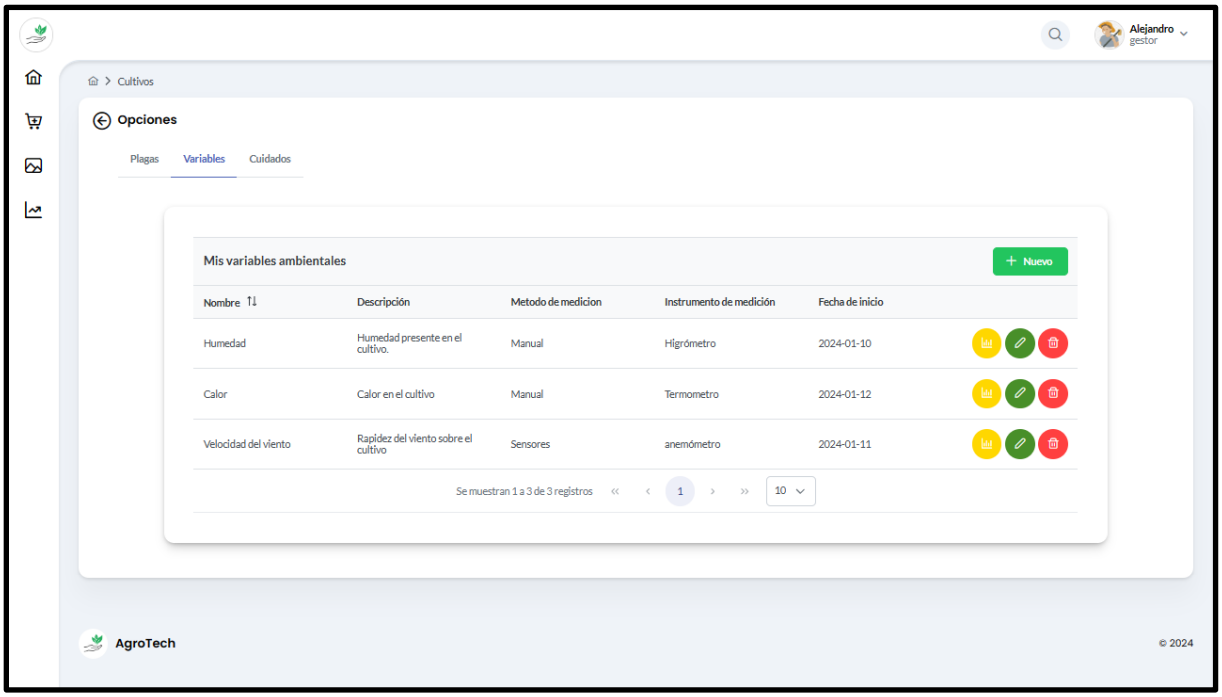
Nota: *Romero, A., & Sandoval, J.* (2024). Diseño propio.

6.3.2.2 Módulo gestión de variables ambientales. Este módulo incluye los componentes encargados de gestionar variables ambientales.

6.3.2.2.1 Componente de gestión de variables. Este componente contendrá todas las variables ambientales que se estén midiendo en un cultivo.

Figura 95.

Vista de gestión de las variables ambientales presentes en un cultivo.

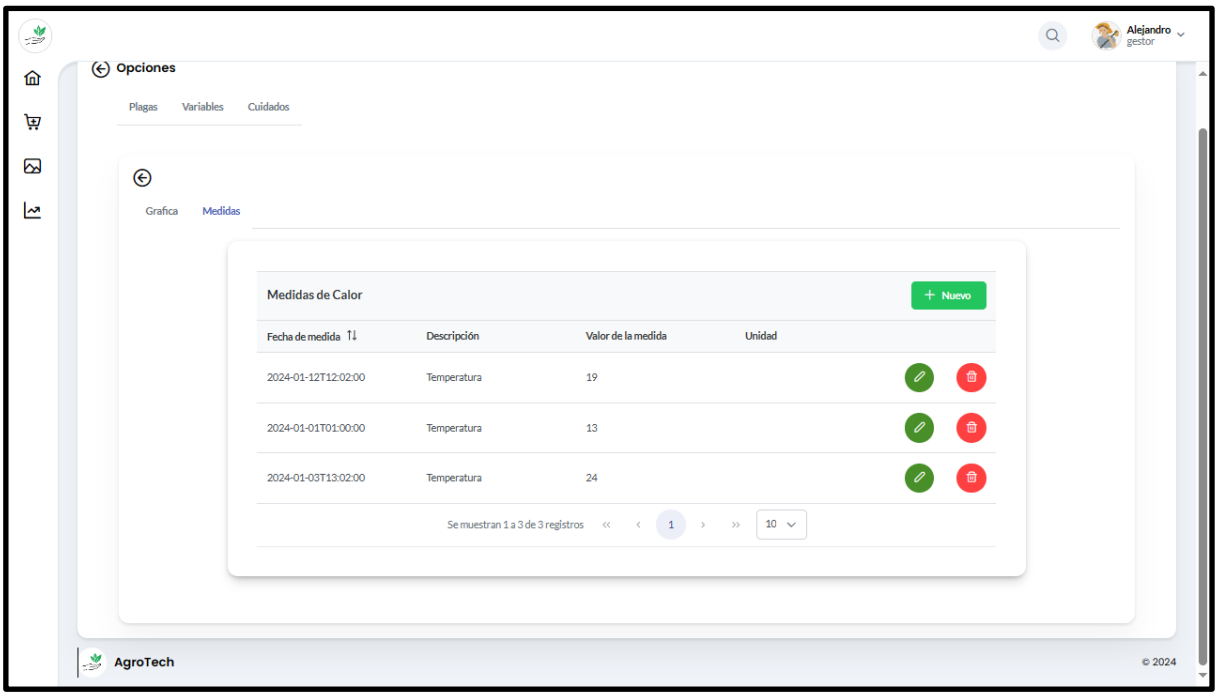


Nota: *Romero, A., & Sandoval, J.* (2024). Diseño propio.

6.3.2.2.2 Componente de gestión de medidas. Este componente se encarga de gestionar las medidas tomadas de una variable específica, las cuales son ingresadas manualmente al sistema.

Figura 96.

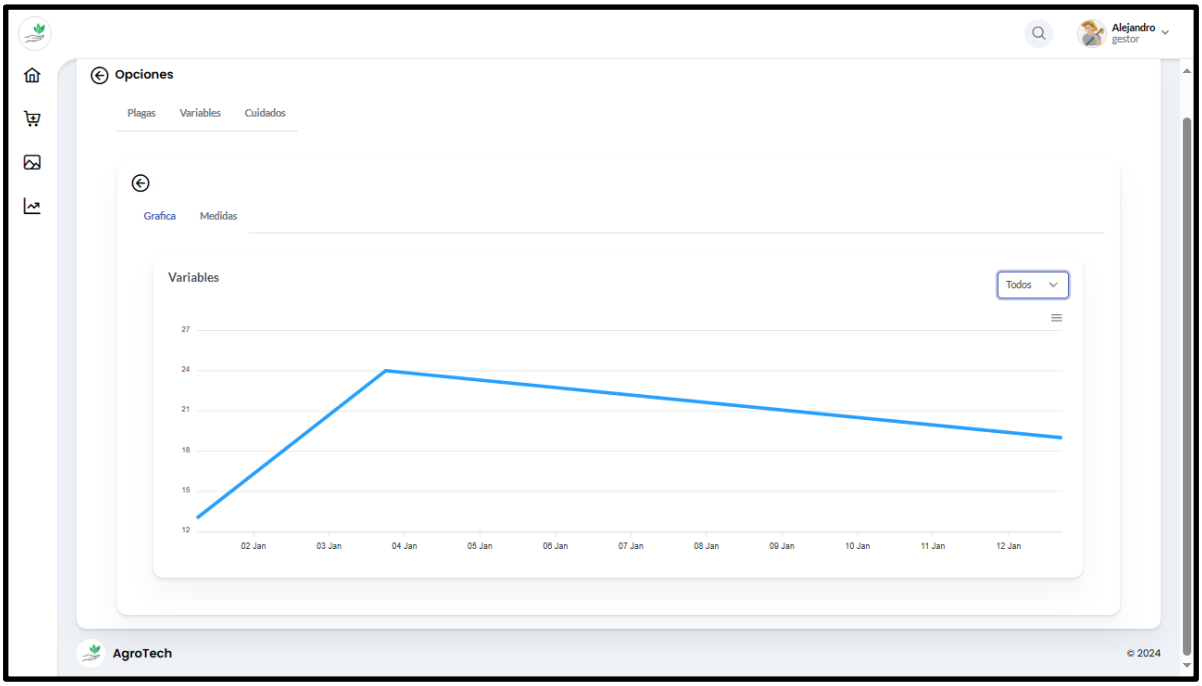
Vista de gestión de las medidas de una variable ambiental en tabla.



Nota: *Romero, A., & Sandoval, J.* (2024). Diseño propio.

Figura 97.

Vista de gestión de las medidas de una variable ambiental en gráfico.



Nota: *Romero, A., & Sandoval, J. (2024).* Diseño propio.

6.3.3 Implementación móvil administrador agrícola

La implementación móvil mantiene la arquitectura limpia y consta de las interfaces necesarias para trabajar sobre las plagas, variables ambientales y los cuidados que también se consideraron en la implementación web.

Figura 98.

Vista de panel de gestión agrícola móvil.

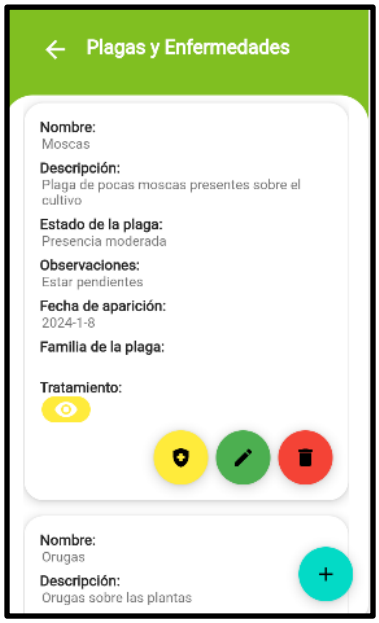
Nota: *Romero, A., & Sandoval, J.* (2024). Diseño propio.

6.3.3.1 Módulo gestión de plagas. Este módulo, al igual que su contraparte web, se encarga de la gestión de las plagas y sus respectivos tratamientos.

6.3.3.1.1 Interfaz de gestión de plagas

Figura 99.

Vista de gestión de las plagas presentes en un cultivo móvil.



Nota: *Romero, A., & Sandoval, J.* (2024). Diseño propio.

6.3.3.1.2 Interfaz de gestión de tratamientos

Figura 100.

Vista de gestión de los tratamientos específicos de una plaga móvil.

Nota: *Romero, A., & Sandoval, J.* (2024). Diseño propio.

6.3.3.2 Módulo gestión de variables ambientales. Este módulo, al igual que su contraparte web, contiene los componentes encargados de gestionar las variables ambientales tomadas en un cultivo.

6.3.3.2.1 Interfaz de gestión de variables

Figura 101.

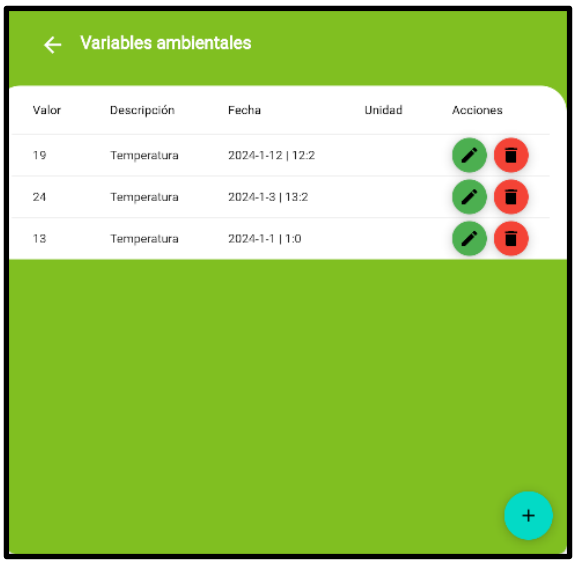
Vista de gestión de las variables ambientales presentes en un cultivo móvil.

Nota: *Romero, A., & Sandoval, J.* (2024). Diseño propio.

6.3.3.2.2 Interfaz de gestión de medidas de una variable específica

Figura 102.

Vista de gestión de las medidas de una variable ambiental.



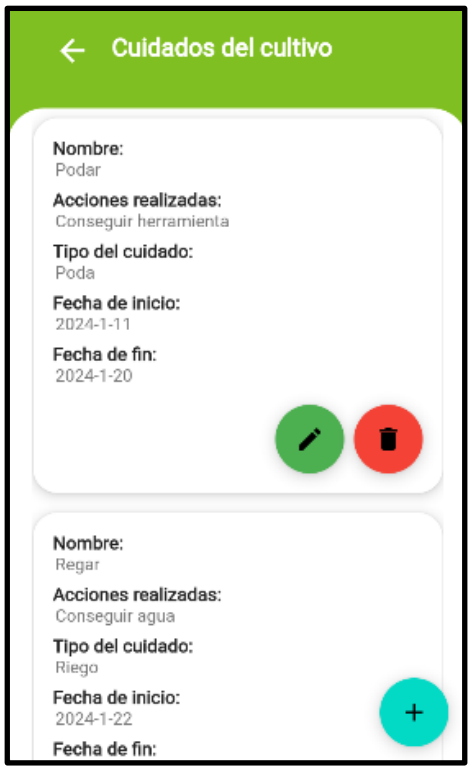
Nota: *Romero, A., & Sandoval, J.* (2024). Diseño propio.

6.3.3.3 Módulo gestión de cuidados. Este módulo se encarga de albergar las interfaces encargadas de gestionar la información sobre los cuidados que se aplican a un cultivo.

6.3.3.3.1 Interfaz de gestión de cuidados

Figura 103.

Vista de gestión de cuidados sobre el cultivo móvil.



Nota: *Romero, A., & Sandoval, J.* (2024). Diseño propio.

7. Conclusiones

Este proyecto no solo se enfocó en la creación de un prototipo para la comercialización de productos agrícolas, sino que también integró funcionalidades adicionales en AgroSellingCO. A lo largo del desarrollo, esta herramienta fue diseñada, implementada y sometida a un riguroso plan de pruebas de calidad, demostrando contar con funcionalidades robustas y destacando un potencial significativo para su aplicación en el ámbito agrícola.

Además de cumplir con el objetivo inicial de proporcionar a los campesinos una plataforma para exhibir sus productos, se han agregado características adicionales con el propósito de centralizar la información sobre plagas, tratamientos, variables y cuidados especiales que un agricultor registra. Estas adiciones no solo enriquecen la utilidad del sistema, sino que también responden de manera proactiva a las necesidades específicas de los usuarios agrícolas, permitiéndoles consultar fácilmente la información sin tener que recurrir a documentos en papel. De esta manera, se fortalece la funcionalidad y aplicabilidad de AgroSellingCO en el entorno agrícola.

Al inicio del desarrollo, se llevaron a cabo las acciones esenciales para definir de manera adecuada los requerimientos que se consideraron pertinentes implementar posteriormente, asegurando así la coherencia y viabilidad de la aplicación. Todas estas actividades implicaron la aplicación de conocimientos en la toma de requerimientos y el diseño de software. Al culminar esta fase, se logró obtener los requerimientos y las historias de usuario apropiadas, ajustadas al alcance de nuestro conocimiento, y que fueron totalmente aplicados en la construcción del prototipo.

En el proceso de seleccionar las herramientas necesarias, realizamos una exhaustiva investigación sobre los lenguajes y frameworks que podrían ser útiles para implementar prototipos como el nuestro. Cada opción fue clasificada según los datos obtenidos durante esta indagación. A medida que avanzaba el proyecto, nos enfrentamos a herramientas que eran inicialmente desconocidas, pero a lo largo del tiempo, logramos comprenderlas y manejarlas con mayor claridad.

El diseño de nuestro prototipo incluyó la conceptualización de arquitecturas clave, como la arquitectura de microservicios, identificando qué microservicios serían necesarios, y la arquitectura limpia para el desarrollo móvil. Además, se estableció como requisito fundamental la escalabilidad del diseño para futuras implementaciones y adaptaciones del sistema.

Una vez que contábamos con todos los elementos fundamentales para concluir la implementación, llevamos a cabo un plan de pruebas integral. Este plan inició con las pruebas unitarias en los componentes backend y frontend, prosiguió con las pruebas de integración en el backend, y culminó con las pruebas de carga y estrés del sistema. Es importante destacar que todas estas pruebas se ejecutaron de manera satisfactoria, indicando así que cada componente de nuestro prototipo cumplió con su tarea específica de acuerdo con lo planificado.

Finalmente, hemos logrado desarrollar un prototipo completamente funcional que permite llevar a cabo el proceso de venta de productos de origen agrícola. Este prototipo incluye otros módulos que proporcionan funcionalidades adicionales, brindando así mayores beneficios a los agricultores.

8. Trabajo futuro

Como trabajo futuro, se continuará desarrollando esta herramienta con el objetivo de transformarla no solo en un marketplace, sino en una herramienta que permita a los campesinos llevar un registro detallado de las actividades y gastos de sus cultivos además de vender productos. Se tiene la intención de agregar la capacidad de recopilar datos por sensores para abordar las variables relacionadas con el entorno agrícola.

En la sección del marketplace, se tiene previsto implementar una pasarela de pago colombiana que esté adaptada a las necesidades de un mercado. Para la aplicación móvil, se buscará implementar la posibilidad de sincronizar los datos tomados cuando no haya conexión a internet con la nube, mejorando así la accesibilidad y utilidad de la herramienta en entornos con conectividad limitada.

El plan futuro contempla poner en marcha el prototipo inicialmente en una localidad pequeña, tal como un pueblo, con la colaboración de agrónomos u otras personas con experiencia en el ámbito agrícola. A través de este método, será posible evaluar y ajustar la herramienta de acuerdo con las necesidades y comentarios específicos de los usuarios antes de que se realice una implementación a mayor escala.

Referencias Bibliográficas

5 frameworks populares para el desarrollo de aplicaciones móviles - blog. (2023, 31 octubre). s.p. Digital. https://spdigitalagency.com/es/blog/popular-frameworks-in-2023-for-mobile-application-development

AgroTech-Uis. (s. f.). GitHub. https://github.com/AgroTech-Uis

Amazon.com. (s. f.). https://www.amazon.com/

Angular. (s. f.). Desarrollo Web. https://desarrolloweb.com/home/angular

Atlassian. (s. f.). *Qué es el control de versiones | Atlassian Git tutorial*. https://www.atlassian.com/es/git/tutorials/what-is-version-control

Atlassian. (s. f.-b). *¿Qué es scrum? [+ Cómo empezar] | Atlassian*. https://www.atlassian.com/es/agile/scrum

auth0.com. (s. f.). *JWT.IO - JSON Web Tokens Introduction*. JSON Web Tokens - jwt.io. https://jwt.io/introduction

AWS | Almacenamiento de datos seguro en la nube (S3). (s. f.). Amazon Web Services, Inc. https://aws.amazon.com/es/s3/

Caaarem. (s. f.). *CAAAREM*. https://www.caaarem.mx/

Cagigas, R. (2022, diciembre). Flutter: Qué es, ventajas (y desventajas) y comparativa con otros frameworks. *NTS Seidor*. https://www.nts-solutions.com/blog/flutter-que-es.html#-significado-. (Consultado el 9 de marzo de 2023).

Calderón, N. (2022, 7 enero). Entendiendo a la arquitectura limpia - Nestor Calderón - Medium. *Medium*. https://nescalro.medium.com/entendiendo-a-la-arquitectura-limpia-7877ad3a0a47

ComproAgro - Quienes somos. (s. f.). https://comproagro.com/quienes_somos

De Comercio De Bogotá, C. (s. f.). Cámara de Comercio de Bogotá. https://www.ccb.org.co/informacion-especializada/observatorio/analisis-economico/crecimiento-economico/pib-bogota-y-colombia-2015-2022

Dreamstime. (s. f.). *Fotos de stock e imágenes libres de derechos por Dreamstime*. https://es.dreamstime.com/

Equipo editorial de IONOS. (2023, 18 enero). *¿Qué es MySQL?* IONOS Digital Guide. https://www.ionos.es/digitalguide/servidores/know-how/que-es-mysql/

Euroinnova Business School. (2023, 27 diciembre). *Cursos de ONG*. https://www.euroinnova.co/blog/que-es-la-agronomia-y-para-que-sirve#que-es-la-agronomia-y-para-que-sirve

Java Persistence API (JPA) - Oscar Blancarte - Software Architecture. (2020, 25 junio). Oscar Blancarte - Software Architecture. https://www.oscarblancarteblog.com/tutoriales/java-persistence-api-jpa/

JUNIT | Marco de Desarrollo de la Junta de Andalucía. (s. f.). https://www.juntadeandalucia.es/servicios/madeja/contenido/recurso/248

LoadView by Dotcom-Monitor. (2023, 17 mayo). *La guía 2024 de JMeter: tutorial de pruebas de carga y rendimiento*. LoadView. https://www.loadview-testing.com/es/la-guia-definitiva-de-jmeter-tutorial-de-pruebas-de-carga-y-rendimiento/

Manjón, D. S. (2023, 18 abril). Qué es Lombok. *OpenWebinars.net*. https://openwebinars.net/blog/que-es-lombok/

Mijacobs. (2023, 5 octubre). *¿Qué es Git? - Azure DevOps*. Microsoft Learn. https://learn.microsoft.com/es-es/devops/develop/git/what-is-git

Mockito - Dos ideas. (s. f.). https://dosideas.com/wiki/Mockito

Ollarves, R. (2023, 21 febrero). 7 razones para elegir MySQL como gestor de base de datos. *interfell*. https://blog.interfell.com/7-razones-para-elegir-mysql#:~:text=MySQL%20es%20muy%20popular%20por,capacidad%20de%20mejora%20o%20crecimiento).

Outlook. (s. f.). https://outlook.live.com/mail/0/

Palau, D. (s. f.). *¿Qué es un marketplace? cómo funcionan, tipos y ejemplos*. https://www.cyberclick.es/que-es/marketplace

Pretonik. (2015, 12 marzo). *Probando Spring Netflix – Parte 5 (Feign + Ribbon)*. Julio Muñoz. https://juliomunoz.wordpress.com/2015/03/02/probando-spring-netflix-parte-5-feign-ribbon/

PrimeNG la opción para principiantes en angular. (2023, 11 agosto). Soluciones Modernas 10X. https://www.10x.gt/blog/desarrollo-de-interfaces-de-usuario-146/primeng-la-opcion-para-principiantes-en-angular-7

Pulido, M. (2019, 27 agosto). *Tutorial: Unit testing con Karma y Jasmine - SlashMobility | Soluciones Mobile*. SlashMobility | Soluciones mobile. https://slashmobility.com/blog/2019/08/tutorial-unit-testing-con-karma-y-jasmine/

¿Qué es la arquitectura de microservicios? | Google Cloud | Google Cloud. (s. f.). Google Cloud. https://cloud.google.com/learn/what-is-microservices-architecture?hl=es-419

¿Qué es un ORM? (s. f.). Deloitte Spain. https://www2.deloitte.com/es/es/pages/technology/articles/que-es-orm.html

¿Qué es una API REST? (s. f.). https://www.redhat.com/es/topics/api/what-is-a-rest-api

¿Qué son las pruebas unitarias?: Explicación de las pruebas unitarias en AWS. (s. f.). Amazon Web Services, Inc. https://aws.amazon.com/es/what-is/unit-testing/#:~:text=Una%20prueba%20unitaria%20es%20un,la%20l%C3%B3gica%20te%C3%B3rica%20del%20desarrollador.

Reddy, K. S. P. Beginning Spring Boot 2.

Rootstack. (s. f.). *Qué es JIRA: lo que todo gerente de proyecto debería saber*. Rootstack. https://rootstack.com/es/blog/que-es-jira-lo-que-todo-gerente-de-proyecto-deberia-saber

Semana. (2022, 13 enero). TierraCol, la plataforma digital que permite acceder a productos típicos colombianos sin intermediarios. Semana.com Últimas Noticias de Colombia y el Mundo. https://www.semana.com/mejor-colombia/articulo/tierracol-la-plataforma-de-los-pequenos-productores-para-vender-sin-intermediarios/202100/

Simões, C. (2021, 6 julio). *¿Qué es TypeScript, y por qué utilizarlo?* Blog ITDO - Agencia de desarrollo Web, APPs y Marketing en Barcelona. https://www.itdo.com/blog/que-es-typescript-y-por-que-utilizarlo/

Spring Cloud Tutorial - Javatpoint. (s. f.). www.javatpoint.com. https://www.javatpoint.com/spring-cloud

Unknown. (s. f.-c). *Proceso de pruebas*. https://adsitestingsfw.blogspot.com/2016/02/proceso-de-pruebas.html

Apéndices

Apéndice A. Manual de instalación.

Descripción: Este apéndice contiene el manual detallado de instalación del prototipo desarrollado como parte de este trabajo.

Apéndice B. Organización de GitHub.

Descripción: Este apéndice contiene el enlace para redirigirse a la organización en la que se encuentran los repositorios del proyecto. Esta organización se puede encontrar en https://github.com/AgroTech-Uis.