SOFTWARE PARA GESTIÓN DE CULTIVOS AGRÍCOLAS

Herramienta software para el soporte a la gestión administrativa, productiva y financiera de cultivos agrícolas a pequeña y mediana escala.

Juan José Prenss Castro, Ruben Dario Rodriguez Moreno y Sergio Andres Sánchez Niño

Trabajo de Grado para Optar al Título de Ingeniero de Sistemas e Informática

Director

Emilio Justiniano Carcamo Troconis

MSc Nuevas tecnologías y dirección de proyectos

Universidad Industrial de Santander

Facultad de Facultad de Ingeniería Físico-mecánicas

Escuela de Ingeniería de Sistemas e Informática

Ingeniería de Sistemas

Bucaramanga

2024

Dedicatoria

Dedico este trabajo a mis padres, quienes desde el principio me brindaron su apoyo inquebrantable. También quiero dedicar este trabajo a mis amigos y hermanos, el ingeniero Juan Camilo Marín y el ingeniero Juan Pablo Jiménez, quienes compartieron risas, ánimo y momentos inolvidables a lo largo de esta travesía. Agradezco a Valentian Escobar, a Braian Higuera y Julián Serrano por su apoyo. Por último, quiero extender mi gratitud a mis padrinos por estar siempre pendientes.

Sergio Andres Sanchez Niño

Estoy en especial gratitud con mis seres queridos quienes creyeron en mí y me acompañaron en algún momento de esta travesía, a mis compañeros de los cuales aprendí y compartí preciados momentos. A las personas y a ti fortachona que me enseñaron de la vida y llegaron para hacerla mejor.

Ruben Dario Rodriguez Moreno

Este proyecto habría sido imposible sin el invaluable apoyo que he recibido a lo largo de mi vida. Quiero expresar mi profundo agradecimiento a todas las personas que han contribuido a este logro, dedico este proyecto especialmente a mis padres y abuela que con su dedicación y esfuerzo incansables lograron impulsarme hasta este punto. Quiero agradecer a todos aquellos compañeros que me ayudaron a lo largo de la carrera, me brindaron su compañerismo y momentos preciados. También, dar las gracias a todas aquellas personas las cuales son partes fundamentales en mi vida y que, sin ellos, esto no tendría sentido.

Juan José Prenss Castro

Agradecimientos

Queremos expresar nuestra más sincera gratitud al Profesor Emilio Justiniano Carcamo Troconis por su inquebrantable compromiso con nuestro proyecto de grado. Su profundo conocimiento en el campo del desarrollo y su dedicación para brindarnos orientación y apoyo fueron esenciales en cada etapa de esta investigación. Su paciencia, sabiduría y entusiasmo contagioso nos inspiraron a dar lo mejor de nosotros mismos en este trabajo.

Tabla de Contenido

Lista de Tablas

Pág.

Lista de Figuras

Pág.

Lista de Apéndices

Resumen

Título: Herramienta software para el soporte a la gestión administrativa, productiva y financiera de cultivos agrícolas a pequeña y mediana escala.[1]

Autor: Juan José Prenss Castro, Ruben Dario Rodriguez Moreno, Sergio Andres Sánchez Niño[2*3*]

Palabras Clave: Sistema web, App móvil, Microservicios, Agricultura, Gestión de cultivos.

Descripción: El proyecto consiste en una solución completa para la gestión eficiente de tareas agrícolas, que incluye una aplicación móvil para agricultores y un panel web centralizado. La aplicación móvil permite a los agricultores planificar, realizar un seguimiento y recopilar datos sobre actividades como siembra, riego y cosecha, además de recibir alertas. El panel web central proporciona a los administradores una visión completa de las operaciones agrícolas, lo que les permite gestionar usuarios, asignar tareas, analizar el rendimiento y sincronizar datos con dispositivos móviles. El objetivo del proyecto es optimizar las actividades agrícolas, mejorando la planificación, ejecución y supervisión de tareas con herramientas como Jira para el seguimiento del desarrollo.

Resultados esperados o conclusiones: El objetivo principal de este proyecto fue desarrollar una herramienta de software eficaz para mejorar la gestión agrícola en explotaciones de pequeño y mediano tamaño. Para lograr este propósito, la herramienta proporcionará funciones específicas para tomar decisiones informadas, aumentando así la eficiencia y rentabilidad. Además, se pretende prevenir pérdidas por plagas, enfermedades y condiciones climáticas adversas mediante el registro de información. En última instancia, el proyecto busca fomentar la adopción de tecnología agrícola avanzada en explotaciones de menor escala, mejorando así su gestión y operación.

---

[1] Trabajo de Grado

[2] Facultad de Ingenierías Físico-Mecánicas. Escuela de Ingeniería de Sistemas e Informática.

[3] Director: Emilio Justiniano Cárcamo Troconis. MSc Nuevas tecnologías y dirección de proyectos.

Abstract

Title: Software tool for supporting administrative, productive, and financial management of small and medium-scale agricultural crops.[4]

Author(s): Juan José Prenss Castro, Ruben Dario Rodriguez Moreno, Sergio Andres Sánchez Niño[5]*[6]*

Key Words: Web system, Mobile app, Microservices, Agriculture, Crop management.

Description: The project involves a comprehensive solution for the efficient management of agricultural tasks, including a mobile application for farmers and a centralized web dashboard. The mobile application allows farmers to plan, track, and collect data on activities such as planting, irrigation, and harvesting, in addition to receiving alerts. The central web dashboard provides administrators with a complete view of agricultural operations in real-time, enabling them to manage users, assign tasks, analyze performance, and synchronize data with mobile devices. The project's goal is to optimize agricultural activities by improving planning, execution, and task supervision with tools like Jira for development tracking.

Expected or conclusions: The main objective of this project was to develop an effective software tool to enhance agricultural management on small and medium-sized farms. To achieve this purpose, the tool will provide specific functions for making informed decisions, thereby increasing efficiency and profitability. Additionally, it aims to prevent losses from pests, diseases, and adverse weather conditions through the recording of information. Ultimately, the project seeks to promote the adoption of advanced agricultural technology on smaller-scale farms, thereby improving their management and operation.

---

[4] Degree Work

[5] Faculty of Physical-Mechanical Engineering. School of Systems Engineering and Informatics.

[6] Director: Emilio Justiniano Cárcamo Troconis.MSc New technologies and project management.

Glosario

API (Interfaz de Programación de Aplicaciones): Una API es un conjunto de reglas y protocolos que permite que diferentes programas y aplicaciones se comuniquen entre sí de manera estandarizada. En esencia, actúa como un intermediario que permite que dos sistemas informáticos diferentes se comprendan y cooperen. Las APIs definen cómo las solicitudes y respuestas de datos deben estructurarse y qué funciones o servicios están disponibles para su uso.

Backlog: El Backlog es una lista priorizada de tareas o elementos pendientes que deben completarse en un proyecto de desarrollo de software. Puede incluir características, errores, mejoras y otras actividades relacionadas con el proyecto.

Sprint: Un Sprint es un período de tiempo fijo y corto en el desarrollo ágil de software durante el cual se trabaja en un conjunto específico de tareas o funcionalidades previamente definidas.

Base de datos: Una Base de Datos es un conjunto organizado de datos que se almacenan y gestionan de manera electrónica. Las bases de datos se utilizan para almacenar información de manera estructurada y eficiente, lo que facilita su acceso, recuperación y gestión. Las bases de datos pueden ser de diversos tipos, como SQL, NoSQL y más.

Arquitectura de Software: La Arquitectura de Software es la estructura fundamental y el diseño de un sistema de software. Describe cómo los componentes de software interactúan entre sí, cómo se organizan y cómo cumplen con los requisitos funcionales y no funcionales del sistema.

Asincronismo: En el contexto de las aplicaciones, el asincronismo se refiere a la capacidad de una aplicación para realizar tareas de manera independiente y sin requerir una conexión de internet constante. Esto permite que la aplicación continúe funcionando y procesando datos incluso cuando la conexión a internet es intermitente o está ausente.

Introducción

Dentro del contexto de la creciente globalización en el que nos encontramos, es cada vez más común que la denominada Revolución 4.0 se integre plenamente en nuestra vida cotidiana; estas tecnologías avanzadas se aplican en una amplia gama de áreas o ámbitos en la sociedad, con el propósito de mejorar la calidad y la eficiencia de los procesos de producción, para contribuir al crecimiento y desarrollo de una zona determinada. No se puede negar que existe una brecha marcada entre el sector rural y el urbano, siendo este último frecuentemente olvidado y descuidado tanto por las autoridades locales, el gobierno nacional y por los propios habitantes del campo.

El actual panorama del sector agrícola se ve afectado por una serie de desafíos notables, entre ellos los elevados niveles de ineficiencia, los costos de transporte onerosos, la baja calidad y los considerables retrasos en el ciclo de cosecha de los productos. Estas problemáticas impactan de manera negativa en las áreas rurales donde se lleva a cabo la actividad agrícola, resultando en un crecimiento económico limitado o incluso nulo. Uno de los efectos más evidentes de estas dificultades radica en las condiciones de vida precarias a las que se enfrentan los agricultores, lo que ha generado altas tasas de migración de la mano de obra campesina hacia áreas urbanas.

Es importante señalar que la brecha tecnológica también juega un papel crucial en este escenario. El acceso limitado a la tecnología agrícola adecuada se suma a los desafíos existentes, obstaculizando el desarrollo de actividades de administración agrícola eficientes. La adopción de tecnologías modernas en el sector podría ser clave para superar los obstáculos mencionados, mejorando la eficiencia, reduciendo los costos y elevando la calidad de los productos agrícolas. En este sentido, la implementación de tecnologías avanzadas podría facilitar la gestión de las

operaciones agrícolas, desde la planificación de cultivos hasta el monitoreo de la cadena de suministro. Asimismo, la adopción de tecnologías puede contribuir a cerrar la brecha entre las áreas rurales y urbanas, generando oportunidades de empleo en el sector agrícola y mitigando el éxodo hacia las ciudades. En resumen, el uso estratégico de la tecnología podría ser un elemento clave para impulsar el desarrollo sostenible del sector agrícola y, en consecuencia, contribuir positivamente al crecimiento económico y al bienestar general en Colombia.

Por lo tanto, a raíz de esta observación surge una pregunta fundamental: ¿Cómo podemos implementar una herramienta de software para respaldar la gestión administrativa y financiera de los cultivos agrícolas a pequeña y mediana escala?, una posible solución que puede ayudar a mitigar los efectos de la problemática señalada implica la integración de instrumentos de gestión tecnológica; lo que conlleva a mejoras significativas en la eficiencia, productividad y toma de decisiones. Dentro de las ventajas que trae este enfoque se encuentran: En primer lugar, la automatización de procesos reduce la ineficiencia y disminuye la probabilidad de errores humanos, lo que, además, se le adiciona el beneficio de la reducción de costos. Segundo, proporciona el acceso a información temprana que proporciona una visión completa del panorama. Tercero, mejoras en la coordinación de procesos, ya que, facilita la colaboración de múltiples actores. Cuarto, permiten la recopilación, gestión y análisis de datos con los cuales se pueden tomar decisiones acertadas basadas en información previamente recolectada y analizada.

Por último, presentan una gran capacidad de adaptación debido a su naturaleza flexible la cual les permite ajustarse a las necesidades de cada productor. James Scobie en "Revolution on the Pampas. A Social History of the Argentine Wheat, 1860-1910" (1964) explica el surgimiento del capitalismo agrario en una región periférica de gran importancia como proveedora de alimentos para el mercado global. Esto se debió a una combinación particular de factores como

la tierra, el trabajo y el capital en el hemisferio sur. Scobie presenta de manera clara y accesible los debates sobre la formación del capitalismo agrario en "nuevas tierras" y demuestra las diversas dimensiones que adopta la innovación tecnológica en la agricultura, abarcando prácticas agrícolas y ganaderas, así como innovaciones biológicas, mecánicas y químicas. Todo esto se desarrolla en un contexto de condiciones ambientales y disponibilidad de recursos propios de las regiones templadas, que son conocidas por su producción de cereales.

Conceptualmente las herramientas de gestión se pueden definir como software o tecnologías, diseñadas con el propósito de ayudar a las empresas a llevar a cabo sus operaciones de la forma más eficiente y efectiva. Estas herramientas abarcan una gran variedad de aplicaciones, las cuales pueden ir desde sistemas de gestión de proyectos hasta software de gestión de las relaciones con clientes (CRM) y de gestión de la cadena de suministros. Dichas aplicaciones brindan información valiosa y precisa que puede ser aprovechada para tomar decisiones acertadas sobre lo acontecido en el proyecto. Ahora bien, la importancia en el uso de estas tecnologías en el sector agrícola es crucial para mejorar la competitividad en un mercado que cada día crece y, con el pasar del tiempo se vuelve más exigente (competitivo), demandando una mayor eficiencia en sus procesos. Las aplicaciones de gestión permiten a los administradores eliminar aquellas tareas repetitivas y tediosas que se pueden volver monótonas, dejando así vía libre para que los usuarios logren poner su esfuerzo en tareas estratégicas y valiosas. Como ejemplo de la eficiencia de las herramientas tecnológicas en este campo tenemos a la agricultura de precisión, en esta ha comprobado que se pueden mejorar la eficiencia y productividad del campo. En su concepto la podemos definir como la aplicación de la tecnología de la información la cual permite manejar los suelos y cultivos basado en la variabilidad presentada en un lote (García, E., y Flego, F, 2008). Básicamente vemos la utilidad del uso de esta tecnología, pues,

suple la necesidad de la óptima administración de grandes, medianos y pequeños terrenos; esto debido al análisis de resultados que se pueden realizar en múltiples sectores ubicados en un mismo lote y los cuales no necesariamente tienen la misma siembra, así permitiendo una administración diferencial y la cual se adapta a cada cultivo.

Después de la pandemia del Covid-19, quedó claro que el modelo agrícola tradicional se había vuelto insuficiente para satisfacer las necesidades actuales. Por tanto, resulta crucial iniciar la implementación de tecnologías en este ámbito. Las nuevas tecnologías, junto con mejoras en la infraestructura y las telecomunicaciones, desempeñarán un papel fundamental en la adopción de estos modernos enfoques agrícolas inteligentes. Para lograr esto, es imperativo que tanto los gobiernos como las empresas comiencen a invertir en recursos destinados a mejorar la conectividad en las zonas rurales. La agricultura es el pilar fundamental del desarrollo regional, ya que garantiza la seguridad alimentaria y nutricional de todo el país. Lamentablemente, el sector agrícola ha sido relegado al olvido, y no se ha fomentado su desarrollo de manera adecuada.

Debido a la vital importancia de estas herramientas, en el marco de este proyecto se dispone el estudio y desarrollo de un prototipo de aplicación web. Tiene como objetivo el facilitar y simplificar la gestión administrativa, productiva y financiera de cultivos agrícolas a pequeña y mediana escala; se diseñará de acuerdo con unos requerimientos previamente definidos. Además, su desarrollo se basará en una arquitectura fundamentada en microservicios.

1. Planteamiento y justificación del problema

Actualmente se ha reconocido la importancia de la producción agrícola a nivel mundial, este juega un papel crucial en la economía y la sociedad contemporánea. Hoy en día, su importancia se destaca por diversas razones fundamentales que abarcan desde la seguridad

alimentaria hasta la sostenibilidad medioambiental. En primer lugar, se considera que es esencial para garantizar la seguridad alimentaria de la población global en constante crecimiento; a medida que la población mundial aumenta, la demanda de alimentos se dispara y crece. La agricultura moderna es la responsable de proporcionar una amplia variedad de alimentos esenciales, desde cereales y verduras hasta proteínas animales. Entonces básicamente sin un sector agrícola sólido y eficiente, la posibilidad de alimentar a la población colombiana sería una tarea verdaderamente difícil. Además, la agricultura si se observa en otros países, desempeña un papel crucial en la economía y desarrollo de estos. Dentro de sus puntos positivos está la generación de empleo en zonas rurales (campo) y contribuye significativamente al producto interno bruto (PIB) de la nación.

El sector agrícola también se encuentra en la primera línea de la lucha contra el cambio climático y la conservación del medio ambiente, pues han empezado a optar por la adopción de prácticas agrícolas sostenibles, las cuales son esenciales para preservar los recursos naturales como el suelo y el agua, y reducir las emisiones de gases de efecto invernadero. La agricultura moderna busca cada vez más métodos respetuosos con el medio ambiente, como la agricultura de conservación y la agricultura orgánica. Por último, también está vinculado a la innovación y la tecnología. La investigación agrícola y el desarrollo de nuevas técnicas y tecnologías, como la biotecnología y la agricultura de precisión, son esenciales para aumentar la productividad y la eficiencia en la producción de alimentos. Lamentablemente en Colombia el campo ha sido dejado en el abandono, toda la inversión en nuevas técnicas y tecnologías se quedan en la zona urbano, aumentando el atraso y la brecha que existe en los territorios rurales.

Mientras que las zonas amplias (ciudades) experimentan avances constantes en diversos sectores, desde la tecnología hasta la infraestructura, las zonas rurales han quedado rezagadas, lo

que perpetúa un ciclo de atraso y desigualdad. Esta falta de inversión agrava aún más los desafíos existentes en el sector agrícola, que enfrenta presiones cada vez mayores para satisfacer la creciente demanda de alimentos de una población en constante crecimiento como es la colombiana. La brecha tecnológica no es la única consecuencia negativas, pues, también hay un notable déficit en la parte económica y social; la falta de inversión en infraestructura rural, educación agrícola y acceso a crédito para los agricultores rurales ha dejado a muchas comunidades en una situación precaria, esto se traduce en una menor calidad de vida para quienes dependen de la agricultura como su principal fuente de ingresos.

Pero el problema no se queda ahí, la migración de poblaciones rurales hacia las ciudades en busca de mejores oportunidades se ha convertido en una presión latente en los servicios urbanos contribuyendo a la urbanización descontrolada. Esta migración también resulta en la pérdida de tradiciones culturales y la disminución de la diversidad agrícola, lo que a largo plazo puede ser perjudicial para la seguridad alimentaria y la preservación de la biodiversidad del territorio. Por eso, para revertir esta situación, es imperativo que el gobierno y otros actores relevantes prioricen la inversión en el sector agrícola y rural. Esto incluye la implementación de programas de capacitación para agricultores, el acceso equitativo a recursos y tecnologías agrícolas, y la mejora de la infraestructura rural. Además, se debe fomentar la valorización de la agricultura como una parte integral de la economía y la identidad cultural de Colombia.

Y es así que vuelve a entrar en juego una pregunta ya antes mencionada: ¿Cómo se puede implementar una herramienta de software para respaldar la gestión administrativa y financiera de los cultivos agrícolas a pequeña y mediana escala?, pues bien, según los estudiosos en economía, la correcta administración de la tierra ha sido clave en la evolución y desarrollo de la sociedad, de ahí que su buena gestión y aplicación de técnicas de cultivo sean piezas

fundamentales en las economías de las naciones, empresas e individuos. Por tanto, se puede considerar a la agricultura como una de las actividades económicas, sociales y ambientales más esenciales para el ser humano. El Banco Mundial mencionó que la agricultura en 2018 representó el 4% del producto bruto (PIB) y en algunos países en desarrollo puede representar más del 25% del PIB (Banco Mundial,2022) y según este mismo estudio el sector agrícola es entre dos y cuatro veces más eficaz para aumentar los ingresos de los más pobres en comparación con otros sectores. Si este análisis es trasladado a Colombia, se puede notar el potencial que representa el agro en la economía de este, pero, lamentablemente, este es un sector poco explotado. Muestra de esto lo encontramos en los aportes presentados al PIB de Colombia en el segundo trimestre del 2022, en el cual según el reporte del DANE el sector agropecuario solo tuvo un crecimiento del 1% (Meneses, 2022).

Ahora se debe mencionar algunos de los factores que influyen en la poca tecnificación que existe en el campo colombiano en la actualidad, dentro de estos se encuentra la falta de inversión en nuevas tecnologías por parte del gobierno y de las empresas privadas. Es por esta razón, que los agricultores e incluso los ganaderos presentan desventajas en el mercado internacional, ya que no cuentan con las herramientas o instrumentos necesarios para mejorar los procesos productivos de sus tierras y estar al nivel de la competencia. El campo colombiano presenta una deficiente implementación tecnológica la cual repercute negativamente en la economía del país y en la vida de los agricultores, ganaderos y en general de cualquier persona que se dedique a esta labor y de nosotros mismos como consumidores de sus productos. Para solventar este problema es crucial invertir en tecnología y formación técnica para optimizar la productividad y rentabilidad del sector agrícola. Así, se podrían mejorar las condiciones de vida de aquellos que dependen de esta actividad económica principalmente en Colombia.

La apropiación de la tecnología en el sector agropecuario viene acompañada de un aumento de la productividad y eficiencia, pues se reduce el gasto de tiempo y energía en actividades las cuales se pueden realizar con mayor agilidad y rapidez si son automatizadas o tecnificadas. En Colombia se presenta un problema, este se trata de la brecha en la aplicación de las tecnologías en el campo (lo cual se ve claramente plasmado en la falta de conectividad), lo cual, disminuye la eficiencia en la producción local. Teniendo esto en cuenta y dada la fácil accesibilidad que se tiene a los aplicativos webs en la actualidad, se plantea para este proyecto el desarrollo de un prototipo de servicio software para ayudar en la gestión administrativa, productiva y financiera de cultivos agrícolas. Esto con el fin de proponer estrategias donde los campesinos, pequeñas y medianas empresas puedan aprovechar las TICS para manejar distintos aspectos característicos del sector agropecuarios tales como la producción y gestión de los cultivos y fomentar la apropiación tecnológica del sector agro.

2. Objetivos

2.1 Objetivo General

Desarrollar un prototipo de aplicación web y móvil para facilitar la gestión administrativa y financiera de cultivos agrícolas a pequeña y mediana escala.

2.2 Objetivos Específicos

- Desarrollar un prototipo de aplicación web para la gestión administrativa y financiera de cultivos agrícolas que incluyan:
    - Gestión de cultivos y labores de mantenimiento.
    - Gestión de ingresos y egresos asociados a las actividades agrícolas.
- Desarrollar un prototipo de aplicación móvil para la gestión asíncrona de servicios agrícolas incluya:
    - Informes de operación.
    - Seguimiento de ingresos y egresos asociados a las actividades agrícolas.
    - Seguimiento a labores de mantenimiento y programación de alertas.
- Diseñar un plan de pruebas para validar el funcionamiento del prototipo de software.

3. Marco teórico

A continuación, se exponen una serie de conceptos y definiciones que fueron considerados en la elaboración de este proyecto de tesis.

3.1 Arquitectura basada en microservicios

Según Martín Fowler y James Lewis, quienes son dos “pioneros” en el campo de la arquitectura basada en microservicios, en su artículo “Mircoservices” (hh) se definen como “estilo arquitectural en el que múltiples servicios, cada uno corriendo de manera individual y desplegados de la forma más automatizada posible, se comunican entre sí mediante mecanismos ligeros, generalmente un recurso API basado en HTTP”.

Una arquitectura de microservicios consta de una colección de servicios autónomos y pequeños. Cada uno de los servicios es independiente y debe implementar una funcionalidad de negocio individual dentro de un contexto delimitado. Un contexto delimitado es una división natural de una empresa y proporciona un límite explícito dentro del cual existe un modelo de dominio (Microsoft,s.f.).

- Los servicios son los responsables de conservar sus propios datos o estado externo. Esto difiere del modelo tradicional, donde una capa de datos independiente controla la persistencia de los datos.
- Los servicios se comunican entre sí mediante API bien definidas. Los detalles de la implementación interna de cada servicio se ocultan frente a otros servicios.
- Admite la programación políglota. Por ejemplo, no es necesario que los servicios compartan la misma pila de tecnología, las bibliotecas o los marcos.

3.2 Control de versiones

Los sistemas de control de versiones son software que ayudan a realizar un seguimiento de los cambios realizados en el código a lo largo del tiempo. A medida que un desarrollador edita el código, el sistema de control de versiones, como Git, toma una instantánea de los archivos. Esta funcionalidad de control de versiones es esencial para mantener un historial completo de las modificaciones realizadas en el código, lo que facilita la colaboración entre desarrolladores y permite la gestión efectiva de proyectos de software. Git, en particular, es ampliamente utilizado debido a su capacidad de almacenar y rastrear cambios de manera eficiente, así como su compatibilidad con flujos de trabajo ramificados y fusiones que permiten una gestión ágil y efectiva del código fuente. (Microsoft, s.f.)

El control de versiones en ingeniería de sistemas se refiere a la gestión sistemática de cambios realizados en un proyecto de software o sistema a lo largo de su ciclo de vida. Por ejemplo, Linus Torvalds desarrolló Git como una solución que se basa en los principios teóricos de control de versiones, que son fundamentales en la ingeniería de sistemas. La teoría detrás de este concepto implica el registrar y rastrear cambios en el código fuente, documentación y otros recursos del proyecto a medida que evoluciona.

3.3 Acoplamiento

El acoplamiento se refiere a qué tan relacionadas o dependientes están dos clases o módulos entre sí. Para las clases acopladas bajas, un cambio importante en una clase tiene un impacto bajo en la otra. El alto acoplamiento en un sistema hace que sea difícil de mantener, ya que un cambio en una clase también tendrá un impacto en otras clases. Esto podría provocar que un cambio fluya a través de un sistema como una mancha de aceite, a veces incluso requiriendo una revisión completa para implementarlo por completo.(Chavez Ramos, 2021)

3.4 Arquitectura limpia

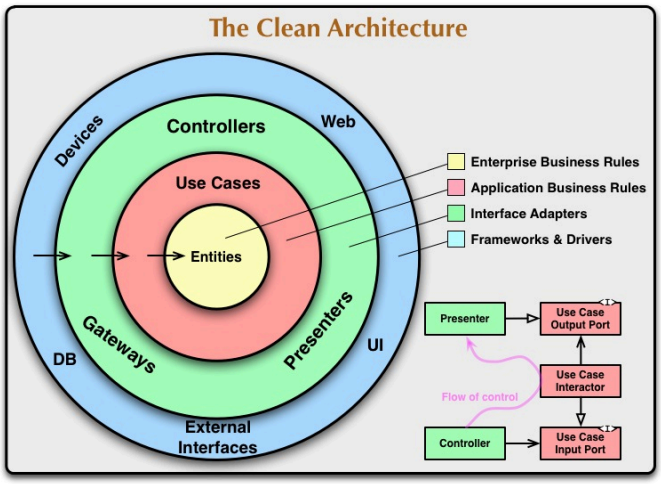
Clean Architecture es un enfoque de diseño de software propuesto por Robert C. Martin (Uncle Bob) en su blog. Su objetivo es crear sistemas de software que sean independientes de detalles tecnológicos y externos, permitiendo que las decisiones clave sobre el diseño estén orientadas hacia el negocio. La arquitectura limpia promueve la separación de preocupaciones en capas concéntricas, donde las capas internas son independientes de las externas. Esto fomenta la facilidad de mantenimiento, escalabilidad y adaptabilidad del software.

La arquitectura limpia se basa en la idea de que el software debe estar organizado en torno a círculos concéntricos, con las capas más internas representando las políticas y reglas de negocio más abstractas, y las capas externas manejando los detalles técnicos y las interacciones con el mundo exterior. Este enfoque promueve la inversión de dependencias y la cohesión, lo que facilita la evolución y prueba del sistema.

Clean Architecture no solo beneficia la calidad técnica del software, sino que también tiene implicaciones positivas en la colaboración entre equipos de desarrollo. Al fomentar la separación clara de responsabilidades y la modularidad, este enfoque facilita la asignación de tareas a equipos especializados, mejorando la eficiencia y la colaboración en el desarrollo de software.

Figura 1

Arquitectura limpia



Fuente: Uncle Bob. (2012). The Clean Architecture. Recuperado de https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html

3.5 Pruebas unitarias

Las pruebas unitarias son una práctica fundamental en el desarrollo de software. Consisten en evaluar las partes más pequeñas y aisladas de un programa para asegurarse de que funcionen como se espera. Cada unidad individual, como funciones, métodos o clases, se somete a pruebas exhaustivas para verificar su correcto comportamiento. Estas pruebas se realizan de manera automatizada, lo que permite detectar y corregir errores de manera temprana en el proceso de desarrollo.

3.6 API RESTful

La API RESTful es una interfaz que dos sistemas de computación utilizan para intercambiar información de manera segura a través de Internet. La mayoría de las aplicaciones para empresas deben comunicarse con otras aplicaciones internas o de terceros para llevar a cabo varias tareas. Por ejemplo, para generar nóminas mensuales, su sistema interno de cuentas debe

compartir datos con el sistema bancario de su cliente para automatizar la facturación y comunicarse con una aplicación interna de planillas de horarios. Las API RESTful admiten este intercambio de información porque siguen estándares de comunicación de software seguros, confiables y eficientes (Amazon, s.f.)

Es considerado como un estilo de arquitectura de software para recursos distribuidos que usan HTTP como método de comunicación. Tiene los siguientes principios:

Utiliza los verbos HTTP , dentro de los verbos que se utiliza REST está get, post, puy y delete; se tiene la siguiente convención para su uso: Get, comúnmente usado cuando se desea obtener un recurso. Post, se usa para la creación de algún recurso. Put para actualizar un recurso. Delete se usa para eliminar un recurso.

No mantiene estados a veces, es necesario contar con un sistema que pueda escalar y también, hay sistemas complejos que tienen la necesidad de usar balanceadores de carga entre dos o más servidores.

3.7 Marcos ágiles

En concreto, los marcos ágiles de desarrollo de software buscan proporcionar en poco tiempo pequeñas piezas de software en funcionamiento para aumentar la satisfacción del cliente. Estas metodologías utilizan enfoques flexibles y el trabajo en equipo para ofrecer mejoras constantes. Por lo general, el desarrollo ágil de software implica que pequeños equipos autoorganizados de desarrolladores y representantes empresariales se reúnan regularmente en persona durante el ciclo de vida del desarrollo de software. La metodología ágil favorece un enfoque sencillo de la documentación de software y acepta los cambios que puedan surgir en las diferentes etapas del ciclo de vida, en lugar de resistirse a ellos(Redhat, s.f.).

Para la realización de este proyecto se optó por la metodología ágil SCRUM.

Los marcos ágiles se basan en varios principios y prácticas que promueven la entrega de software de alta calidad de manera rápida y flexible. Uno de los teóricos más influyentes en este campo es Jeff Sutherland, quien es uno de los co-creadores del marco Scrum. Estos marcos se basan en el principio fundamental de que los proyectos son intrínsecamente complejos y cambiantes; normalmente aquellos que desarrollan estos tipos de marcos ágiles se enfocan en la adaptabilidad y la colaboración. Algunos complejos claves desde la perspectiva de Jeff Sutherlan son:

Iteración y Entrega Incremental: los equipos ágiles dividen el trabajo en iteraciones más pequeñas llamadas "sprints" en el caso de Scrum. tienen una duración fija y producen un incremento funcional del producto que se puede probar y entregar al final del sprint.

Priorización y Valor para el Cliente: se centran en entregar valor real al cliente de manera temprana y continua.

Inspección y Adaptación: realizan revisiones regulares para inspeccionar y evaluar su trabajo. En Scrum, esto se hace en las reuniones de retrospectiva y revisión.

Colaboración y Autonomía del Equipo: Jeff Sutherland y otros teóricos promueven la colaboración cercana entre los miembros del equipo.

Transparencia: en la comunicación y la visibilidad del trabajo realizado son fundamentales en los marcos ágiles. Los tableros de seguimiento, las reuniones regulares y la documentación clara son prácticas comunes para mantener a todos informados.

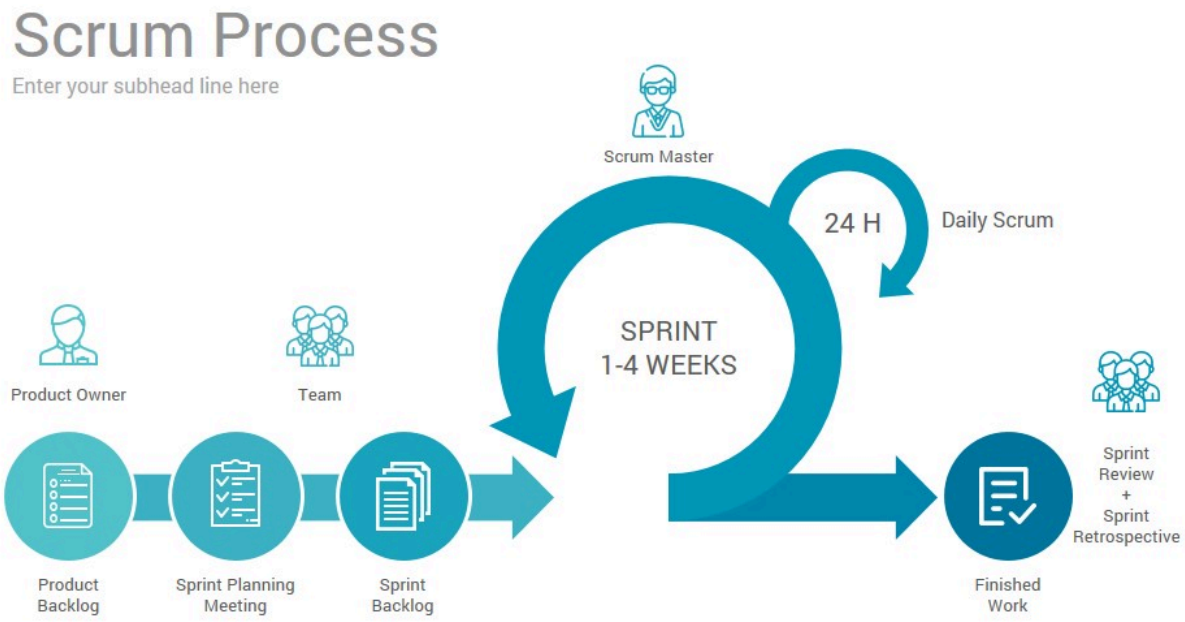
3.8 SCRUM

Scrum 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. Al igual que un equipo de rugby (de donde proviene su nombre) cuando entrena para un gran

partido, scrum anima a los equipos a aprender a través de las experiencias, a autoorganizarse mientras aborda un problema y a reflexionar sobre sus victorias y derrotas para mejorar continuamente (Atlassian, s.f.).

Figura 2

Marco de trabajo ágil Scrum



***Fuente**: Web Design Cusco. (s.f.). Qué es Scrum y para qué sirve. Recuperado de https://webdesigncusco.com/que-es-scrum-y-para-que-sirve/*

3.9 MVC

El MVC separa una aplicación en tres componentes principales, cada uno con una función específica y bien definida: el Modelo, la Vista y el Controlador.

Modelo: el Modelo se representa a través de objetos llamados "servicios". Los servicios en Angular son clases que contienen lógica de negocio, manipulan datos y gestionan la comunicación con servidores o servicios externos. Estos son compartidos entre diferentes componentes y proporcionan una forma de acceder y modificar datos de manera centralizada

Vista: La Vista en Angular está representada por componentes. Los componentes son la parte de la interfaz de usuario de la aplicación y controlan la presentación de datos. Los componentes pueden contener plantillas HTML, donde se define cómo se muestra la información al usuario. Además, Angular permite la creación de directivas personalizadas para manipular la Vista de manera más avanzada.

Controlador: En Angular, el Controlador no existe como tal. En su lugar, se utiliza el concepto de "Controlador de Componente". Cada componente de Angular actúa como su propio controlador. Los controladores de componentes son responsables de gestionar la lógica relacionada con la vista y la interacción del usuario. Además, los componentes pueden comunicarse entre sí a través de servicios para compartir datos y funcionalidades.

4. Metodología

Al embarcarse en la creación de un proyecto de considerable complejidad, con la flexibilidad para evolucionar a medida que avanza el tiempo, se torna esencial no solo contar con una planificación sólida, un seguimiento meticuloso y un progreso constante, sino también con un enfoque que fomente estas cualidades. En este sentido, surge como una alternativa sumamente viable la adopción de una metodología fundamentada en Scrum. Al basar el proceso en los principios ágiles de Scrum, se establece un marco de trabajo que no solo permite una adaptación fluida a los cambios, sino que también fomenta la colaboración continua, la entrega incremental de valor y la introspección regular, aspectos cruciales para lograr un avance óptimo en proyectos en constante evolución.

Figura 3
Fases de los sprints en scrum

Fuente: *Diseño propio*

4.1 Análisis del proyecto

En Scrum, la definición del proyecto se realiza en la fase de "planificación del sprint". En esta fase, el equipo Scrum trabaja junto para definir los objetivos del sprint y las funcionalidades que se implementarán. El equipo Scrum identifica los elementos del backlog que se pueden completar en el sprint, para esto los integrantes del desarrollo cuenta con un aproximado de 14 semanas en las cuales se hace la planificación.

4.2 Planificación y diseño

En Scrum, la planificación y el diseño se realizan en la misma fase que la definición del proyecto, es decir, en la "planificación del sprint". Durante esta fase, el equipo Scrum discute cómo se implementarán las funcionalidades identificadas en el Product Backlog y crea un plan detallado para el sprint, la planificación de cada ciclo se realiza en un tiempo estimado de una semana para cada uno, con un total de 3 procesos de planificación.

4.3 Implementación

La implementación en Scrum se lleva a cabo durante el sprint. El equipo Scrum trabaja en las tareas identificadas en la "planificación del sprint" y se reúne periódicamente en la reunión de Scrum para asegurarse de que el trabajo avanza según lo planificado. En Scrum, se enfatiza la

entrega continua de software funcionando y se fomenta el trabajo en equipo y la colaboración. La implementación se divide en 3 partes coincidiendo con el número de módulos a trabajar, las semanas asignadas a cada fracción están dadas según la complejidad y cantidad de trabajo que se estima para cada uno de ellos, la división realizada se puede observar a detalle en el cronograma de actividades.

4.4 Pruebas

Las pruebas son una parte importante de Scrum y se realizan continuamente durante todo el proceso de desarrollo. En Scrum, las pruebas pueden ser realizadas en cualquier momento durante el sprint. A esta fase también se le adicionaran los eventos de revisión de Sprint y retrospectiva de Sprint para llevar mejor control de las pruebas. Para la fase de pruebas se asigna un periodo de dos semanas después de concluida la implementación con el motivo de evaluar la parte funcional de la entrega y así validar el producto potencialmente entregable.

5. Desarrollo del proyecto

En esta sección, revisaremos el proceso de desarrollo que se ha llevado a cabo a lo largo de los tres sprints previamente concluidos en el proyecto. Durante este proceso, se dividieron las actividades en cuatro fases esenciales: Definición de proyecto, Planificación y Diseño, Implementación y Pruebas.

5.1 Definición del proyecto

En esta fase, se establecieron las bases del proyecto. Se definieron los requerimientos funcionales y no funcionales, se plantearon las actividades necesarias y se crearon los diagramas de casos de uso y se identifican los actores del sistema. Estos pasos son esenciales para comprender el alcance y las expectativas del proyecto. Ahora, profundicemos en cada uno de estos aspectos.

5.1.1 Definición de requerimientos funcionales y no funcionales.

Requerimientos funcionales Front-End:

- Registro de proveedores y clientes: El usuario cuenta con la vista para realizar la solicitud de registro de proveedores y clientes.
- Gestión de usuarios: La aplicación cuenta con una interfaz que permite gestionar usuarios.
- Gestión de Cultivos: Se permite visualizar y administrar la información relacionada con los cultivos.
- Gestión de datos de plantas: Se permite manejar los datos registrados sobre las plantas de cierto cultivo.
- Gestión de plagas y tratamientos: La aplicación facilita la visualización del registro histórico de las plagas y sus tratamientos.

Requerimientos no funcionales Front-End:

- Experiencia del Usuario: Las interfaces gráficas resultan intuitivas y claras.
- Responsividad: La aplicación deberá adaptarse de manera sensible a los cambios en la pantalla del dispositivo en el cual se visualiza.

Requerimientos funcionales Back-End:

- Administración de especies de flora: debe almacenar la información de las diferentes plantas así como asociar si pertenecen a una planta padre. Adicionalmente se almacena los mantenimientos/tratamientos que éstas reciben.
- Control de áreas de producción: se debe llevar control de la facturación de la empresa para cada área ya sea en la actividades en los cultivos como en las ventas y consumos.

- Administración de usuarios: un administrador debe registrar al personal y adicionalmente controlar la data acerca de los proveedores los cuales están asociados a la empresa.
- Visualización de los diferentes endpoints: haciendo uso de swagger vemos los diferentes endpoints en cada uno de los microservicios, proporcionando información para su uso.

Requerimientos no funcionales Back-End:

- Integridad de datos: se garantiza la protección de los datos registrados.
- Estabilidad de los servicios: los servicios se deben mantener en pie aun cuando se presenta un gran flujo de peticiones.
- Escalabilidad: dada la arquitectura de microservicios podemos tanto acoplar o remover módulos manteniendo así el funcionamiento total.

Requerimientos funcionales App-Móvil:

- Visualización de cultivos: un usuario puede visualizar la información de sus cultivos.
- Notificaciones y Recordatorios: La aplicación puede enviar notificaciones y recordatorios a los usuarios sobre tareas.
- Actualización de tareas: La app debe contar con una función para actualizar el estado de las actividades asignadas.
- Informes: La aplicación debe mostrar los datos de producción con gráficas y resultados generales.
- Historial de movimientos: La aplicación debe contar con un historial de movimientos.

Requerimientos no funcionales App-Móvil:

- Mantenibilidad: La aplicación debe seguir buenas prácticas de programación para facilitar futuras actualizaciones y mejoras.

- Usabilidad y Experiencia del Usuario: La interfaz de usuario debe ser intuitiva y fácil de usar.
- Modo asíncrono: La aplicación debe ser capaz de funcionar de manera completa o parcial sin acceso a una conexión a Internet, permitiendo a los usuarios realizar ciertas funciones incluso cuando no están conectados.
- Notificación de conexión: La app debe contar con mensajes que notifique al usuario sobre su conexión a internet.

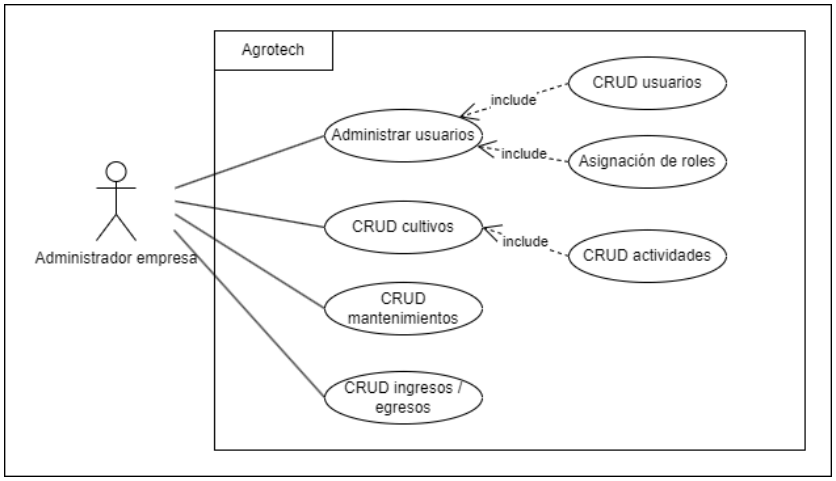
5.1.2 Definición de diagramas de casos de uso y actores de sistema.

Los actores son las personas o roles clave que interactúan con el sistema y desempeñan funciones específicas en su operación y administración. A continuación, se describen los actores seleccionados y sus roles en el sistema:

- **Administrador de la Empresa:** El administrador de la empresa es responsable de gestionar los aspectos específicos de su propia empresa dentro del sistema. Esto incluye la gestión de cultivos, proveedores, clientes y empleados relacionados con su empresa en particular.

Figura 4

Diagrama de casos de uso del rol administrador de la empresa.

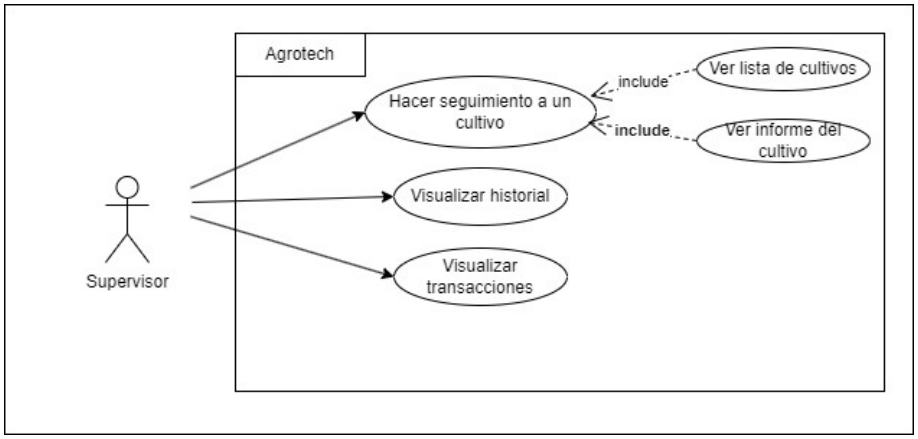


Fuente. *Diseño propio*

- **Supervisor:** El supervisor tiene un rol de supervisión en el campo. Utiliza la aplicación móvil para acceder a información en tiempo real sobre los cultivos y las tareas. Puede asignar tareas a los obreros y tomar decisiones basadas en la información proporcionada por el sistema.

Figura 5

Diagrama de casos de uso del rol supervisor.

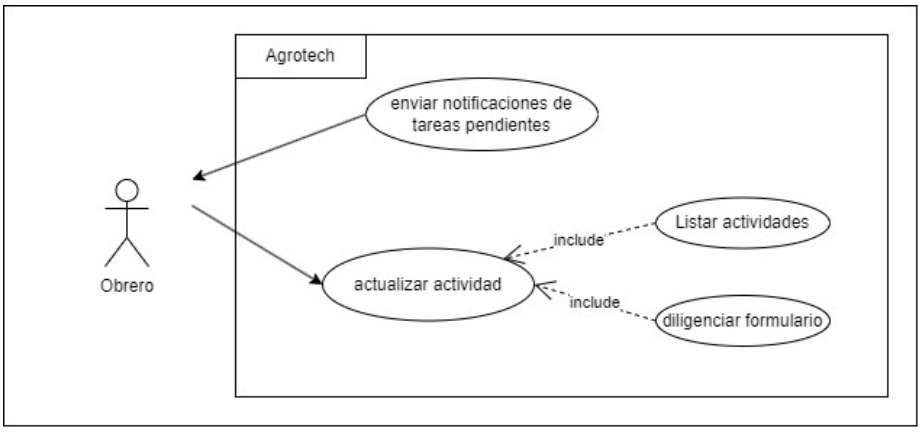


Fuente. *Diseño propio*

- **Obrero:** Los obreros son los trabajadores en el campo. Utilizan la aplicación móvil para acceder a las tareas asignadas, actualizar el estado de las actividades y registrar información relevante sobre los cultivos.

Figura 6

Diagrama de casos de uso del rol obrero.



Fuente. *Diseño propio*

La elección de los roles en un proyecto, como los actores mencionados (administrador de la empresa, supervisor y obrero), se basa en la necesidad de distribuir responsabilidades y funciones para lograr los objetivos del proyecto. La justificación detrás de la selección de estos roles se basa en las siguientes consideraciones:

Complejidad del Sistema: Un sistema de gestión de cultivos es inherentemente complejo y requiere una gestión eficiente y coordinada. Los roles de administrador general y administrador de la empresa se justifican por la necesidad de supervisar y administrar aspectos globales y específicos del sistema, respectivamente.

Supervisión en el Campo: La gestión de cultivos implica actividades en el campo, donde un supervisor desempeña un papel crucial para supervisar el progreso y tomar decisiones basadas en la información en tiempo real.

Ejecución de Tareas: Los obreros son los ejecutores de las tareas en el campo y necesitan un rol específico para registrar y actualizar el estado de las actividades. Esto asegura una ejecución eficiente de las operaciones.

Distribución de Responsabilidades: La elección de roles se basa en la necesidad de distribuir las responsabilidades de manera equitativa y eficiente, garantizando que cada componente del sistema tenga un enfoque específico.

5.1.3 Planteamiento de actividades.

Durante esta fase, hemos convertido los requerimientos y objetivos generales en actividades concretas. Estas actividades servirán como la base sólida sobre la cual construiremos nuestros sprints y backlog en las siguientes etapas del proyecto.

Tabla 1

Actividades de desarrollo web

N°	Actividades
WEB1	Diseñar una interfaz que permita la creación de cultivos
WEB2	Diseñar una interfaz que permita dar seguimiento a los cultivos
WEB3	Desarrollo de una interfaz que permita la asignación de múltiples actividades a los cultivos
WEB4	Desarrollo de una interfaz que permita la administración de las actividades por cultivos.
WEB5	Desarrollo de una interfaz que permita visualizar y editar las actividades en cada cultivo
WEB6	Desarrollo de una interfaz que permita la visualización del manejo económico generado por los cultivos.
WEB7	Desarrollo de una interfaz para crear, visualizar y editar los mantenimientos llevados en cada cultivo.

Tabla 2

Actividades de desarrollo back-end

N°	Actividades
BK1	Servicios de cultivo
BK2	Servicio de actividades de cultivo
BK3	Servicios de ingresos y egresos
BK6	CRUD cultivos
BK7	CRUD mantenimientos
BK8	CRUD actividades
BK9	CRUD ingresos y egresos
BK10	CRUD empresa
BK11	CRUD Plantas

Tabla 3

Actividades de desarrollo app móvil

N°	Actividades
APP1	Crear api Mock para pruebas
APP2	Integración de servicio de login.
APP3	Interfaz de usuario inicio de sesión.
APP4	Implementar funcionalidad para manejo de roles
APP5	Implementar detección de conexión a internet.
APP6	Integración de servicios de mantenimiento.
APP7	Interfaz de usuario para mantenimientos.
APP8	Formulario para actualizar mantenimientos de cultivo.
APP9	Implementar envío de notificaciones
APP10	Implementar asincronía en los servicios.
APP11	Integración de servicios de movimientos monetarios.
APP12	Interfaz de usuario para visualizar ingresos monetarios.
APP13	Interfaz de usuario para visualizar egresos monetarios.
APP14	Interfaz de usuario para visualizar informes diarios, informes financieros e informes de cultivo.
APP15	Interfaz de usuario para visualizar historial.

5.2 Diseño.

En esta sección, abordamos la fase de planificación y diseño. Esto implicó el desarrollo de un modelo de base de datos, la planificación de sprints, la construcción de un backlog y la definición de la arquitectura.

5.2.1 Diseño de modelo de base de datos.

En esta parte, nos enfocamos en la estructura de datos que respalda nuestro sistema de gestión de cultivos. El diseño de la base de datos implicó la creación de tablas, relaciones y la definición de cómo se almacenará y gestionará la información. El resultado final se dio después de varias iteraciones y cambios, hasta crear un modelo que permitiera almacenar un historial de los datos asociados a un cultivo e identificar los actores implicados en cada proceso.

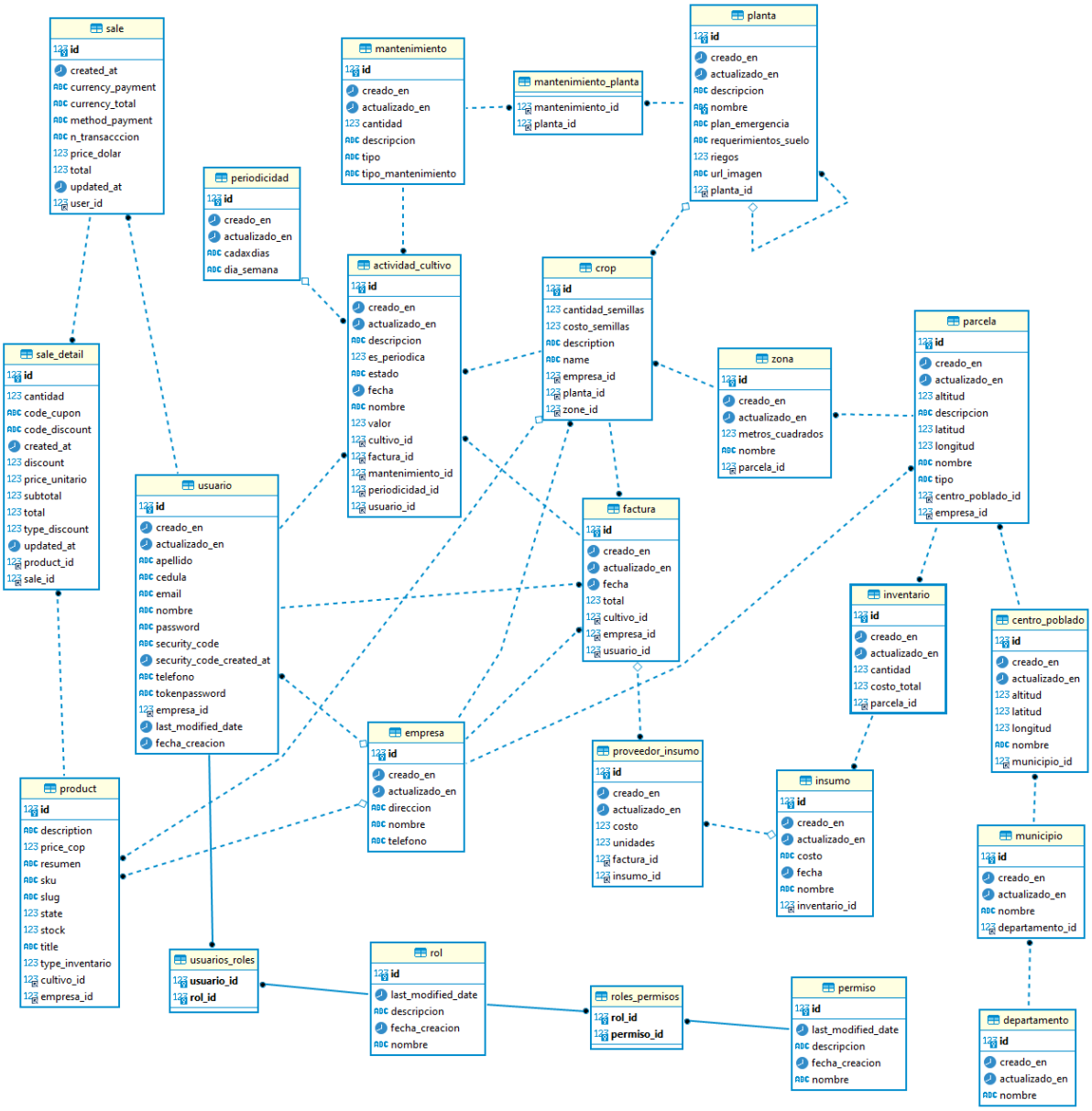
Al momento de diseñar, se llevó a cabo un análisis para determinar qué datos necesarios para un control de cultivos. Aunque es común registrar información como el tipo de planta y la zona cultivada, se identificó la necesidad de incorporar tablas para los mantenimientos que necesita una planta, plagas encontradas y el historial de actividades realizadas.

Dentro de la Agricultura, la rotación de cultivos es una práctica más que necesaria. Se trata de cambiar el tipo de cultivo año tras año en un mismo lote para evitar plagas, enfermedades o malezas. (casafe) Para llevar un registro de este comportamiento se guarda el lugar cultivado de cada planta.

Al momento de llevar cuentas en un cultivo se encontró que los ingresos se obtienen por las ventas ya sea de sus frutos, como también de otros productos adicionales y el manejo de sus egresos se debe a la paga de sus obreros y a la compra de insumos agrícolas, de esta forma puedo encontrar cuales son las ganancias que obtuve del cultivo y los gastos que representó almacenando estos datos.

Figura 7

Modelo de base de datos



Fuente. *Diseño propio (DBearer)*

5.2.2 Planeación de sprints y Backlog.

En esta sección, abordaremos la planificación de sprints y la gestión del backlog. A lo largo de esta sección, explicaremos cómo se organizaron las actividades planificadas dentro del backlog de cada sprint. En este proceso se utilizó la plataforma Jira como herramienta para gestionar el sprint.

Jira nos proporcionó una solución robusta y altamente personalizable para organizar y priorizar las historias de usuario. A través de esta plataforma, creamos una estructura que permitió a nuestro equipo asignar historias a sprints según las necesidades del proyecto.

Tabla 4

Backlog sprint 1

Backlog Sprint 1	
N°	**Actividades app web**
WEB1	Diseñar una interfaz que permita la creación de cultivos
WEB2	Diseñar una interfaz que permita dar seguimiento a los cultivos
N°	**Actividades back-end**
BK1	Desarrollar servicios de cultivos
BK2	Servicio de actividades de cultivo
BK3	Servicios de ingresos y egresos
N°	**Actividades app móvil**
APP1	Crear api Mock para pruebas
APP2	Integración de servicio de login.
APP3	Interfaz de usuario inicio de sesión.
APP4	Implementar funcionalidad para manejo de roles
APP5	Implementar detección de conexión a internet.

Tabla 5

Backlog sprint 2

Backlog Sprint 2	
N°	**Actividades app web**
WEB3	Desarrollo de una interfaz que permita la asignación de múltiples actividades a los cultivos
WEB4	Desarrollo de una interfaz que permita la administración de las actividades por cultivos.
N°	**Actividades back-end**
BK6	CRUD cultivos
BK7	CRUD mantenimientos
N°	**Actividades app móvil**
APP6	Integración de servicios de mantenimiento.
APP7	Interfaz de usuario para mantenimientos.
APP8	Formulario para actualizar mantenimientos de cultivo.
APP9	Implementar envío de notificaciones
APP10	Implementar asincronía en los servicios.

Tabla 6

Backlog sprint 3

Backlog Sprint 3	
N°	**Actividades app web**
WEB5	Desarrollo de una interfaz que permita visualizar y editar las actividades en cada cultivo
WEB6	Desarrollo de una interfaz que permita la visualización del manejo económico generado por los cultivos.
WEB7	Desarrollo de una interfaz para crear, visualizar y editar los mantenimientos llevados en cada cultivo.
N°	**Actividades back-end**
BK8	CRUD actividades
BK9	CRUD ingresos y egresos
N°	**Actividades app móvil**
APP11	Integración de servicios de movimientos monetarios.
APP12	Interfaz de usuario para visualizar ingresos monetarios.
APP13	Interfaz de usuario para visualizar egresos monetarios.
APP14	Interfaz de usuario para visualizar informes.
APP15	Interfaz de usuario para visualizar historial.

5.2.2 Definición de arquitectura.

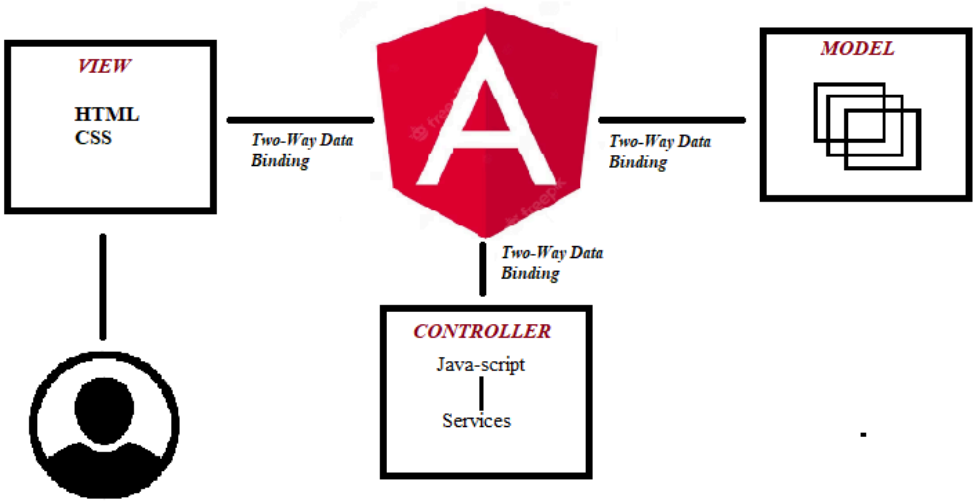
La arquitectura es un pilar fundamental en el desarrollo de software, proporcionando la estructura y el marco para las aplicaciones. En esta sección, explicaremos las arquitecturas clave de nuestro proyecto: web, backend y aplicación móvil.

a. Arquitectura Web

En el desarrollo del componente web se hizo uso del framework Angular el cual implementa el Modelo-Vista-Controlador (MVC) este nos permite organizar y estructurar aplicaciones de manera modular y eficiente, lo que facilita su mantenimiento, escalabilidad y comprensión.

Figura 8

Diagrama del MVC en angular.



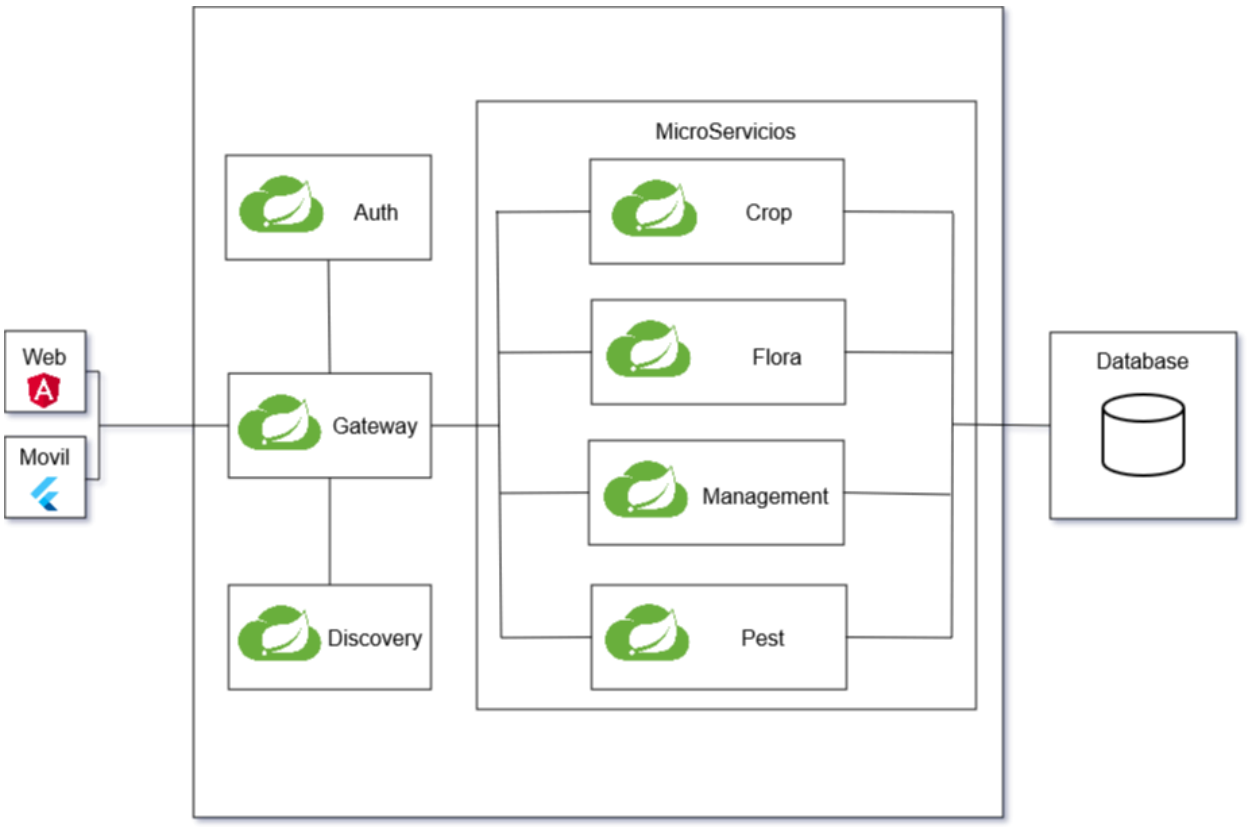
Fuente. *Diseño propio.*

b. Arquitectura Back-End

Se opta por una arquitectura de microservicios la cual nos ofrece modularidad, con esto podemos agregar o eliminar nuevos servicios, dándonos flexibilidad, escalabilidad y fácil mantenimiento de la aplicación.

Figura 9

Diagrama de arquitectura por microservicios.



Fuente: *Diseño propio.*

c. Arquitectura de la App-Móvil

En la app se implementa Clean Architecture ya que es un patrón de diseño que busca mantener una separación entre las diferentes capas y responsabilidades de una aplicación. Se compone de capas como:

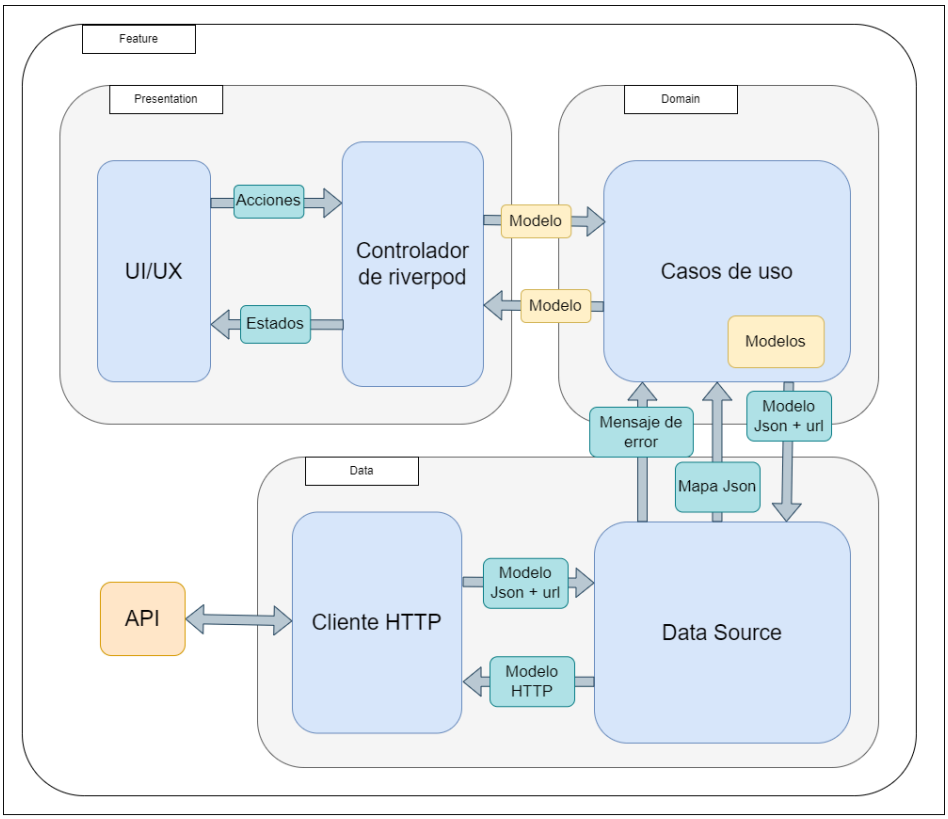
Capa de Presentación (UI): Puedes utilizar Riverpod para administrar el estado local de los widgets y componentes de la interfaz de usuario. Los proveedores de Riverpod pueden proporcionar datos específicos de la pantalla y eventos que afectan al estado.

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 10

Diagrama de implementación de riverpod con arquitectura limpia.



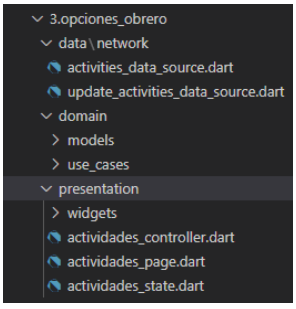
Fuente. *Diseño propio.*

Las capas se comunican a través del intercambio de información ya sea escuchando un estado o intercambiando modelos de datos. En el caso de los componentes que se encuentran en la capa de presentación cuando hay una comunicación entre sí, se hace llamado de una función del controlador desde la UI/UX y dependiendo del caso se hace un cambio de estado que puede que cambie la visualización de la app o hace un llamado a la capa de dominio, pasando a la capa de dominio aquí se ejecutan los casos de uso que interactúan con los modelos de datos que van tanto a la capa de presentación como a la capa de datos. Por último tenemos la capa de Datos donde sus componentes convierten y mandan datos de formato JSON a un mapa y viceversa, dependiendo si se va a llevar a la capa de domain o se usa como parte de la consulta en un API; en esta capa también se suelen mandar mensajes de error para casos de fallo que se controlan den las demás capas.

Para facilitar mejor esta implementación de las capas, se decide identificarlas a través de directorios con su nombre respectivo, cada carpeta contiene sus partes respectivas y en dado caso que escale demasiado se agregan carpetas con los nombres de los componentes para reconocerlos más rápido.

Figura 11

Manejo de directorios con clean architecture



Fuente. *Diseño propio*

5.3 Fase de Implementación.

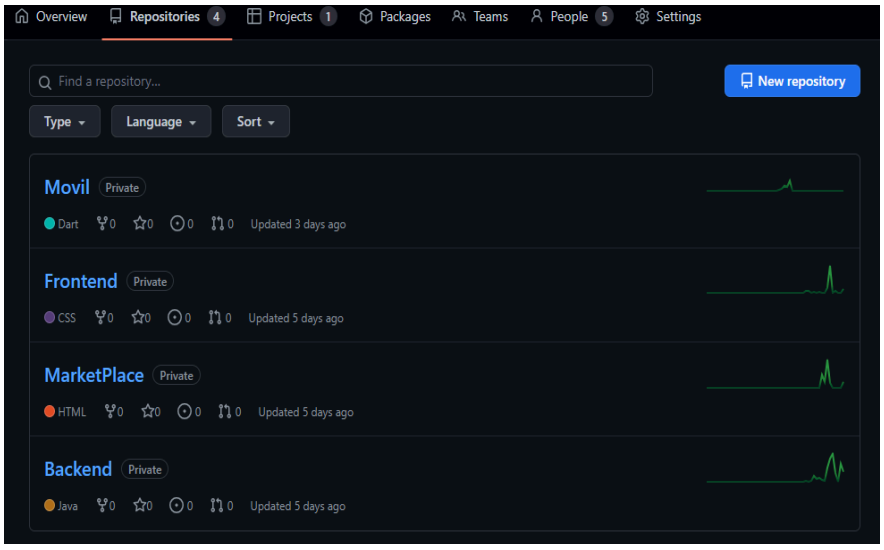
La fase de implementación es esencial en el proceso de desarrollo del proyecto. Durante esta etapa, se llevaron a cabo dos actividades esenciales: la creación de vistas y componentes para plataformas web y móviles, así como el desarrollo de microservicios. Durante esta etapa, se tradujeron los diseños y requerimientos en soluciones funcionales y prácticas. Ahora, examinaremos en detalle los resultados.

5.3.1 Repositorio

Se creó una organización en git hub la cual agrupará las distintas implementaciones del proyecto, siendo estas backend, frontend y móvil.

Figura 12

Repositorio de github.

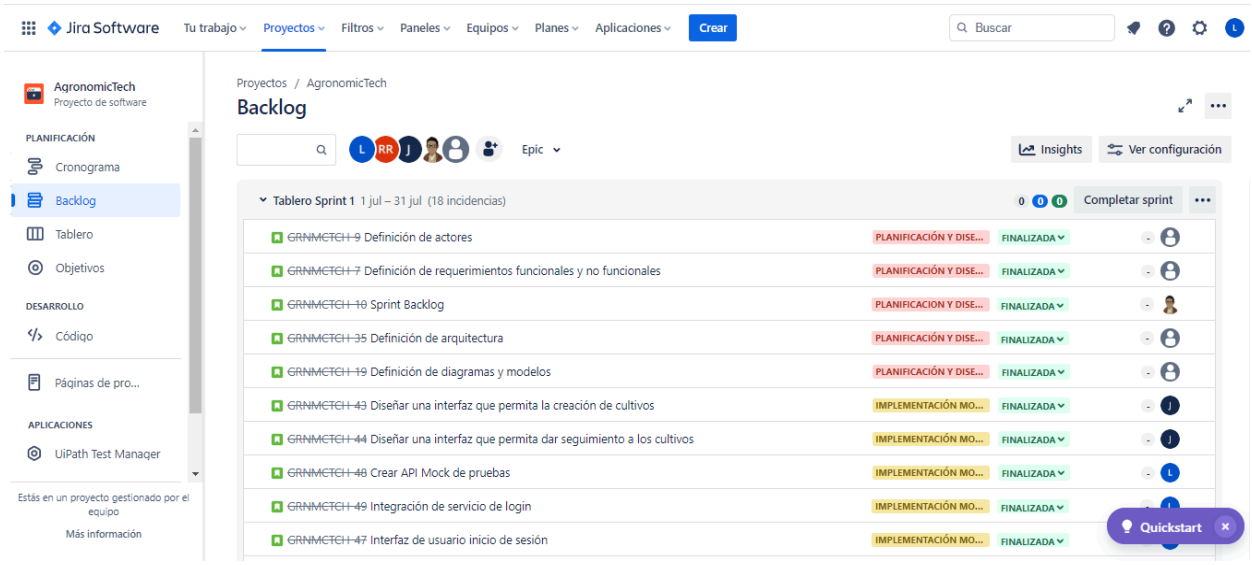


Fuente: *Diseño propio*

Se empleó la plataforma de gestión de proyectos y seguimiento de problemas Jira, la cual fue de gran utilidad.Contribuyó a la realización de las actividades planificadas, al ofrecer una estructura clara para la organización y estimación de tiempos de las tareas pendientes.

Figura 13

Uso de Jira para el control de los sprints.



Fuente: *Diseño propio*

5.3.2 Implementación de web.

En este proyecto, se utilizó Angular como framework base para el desarrollo web, aprovechando las ventajas que ofrece. Para complementar el uso de esta herramienta, se incorporó la biblioteca PrimeNG, la cual es de código abierto y diseñada para aplicaciones web desarrolladas con Angular. PrimeNG proporciona una amplia variedad de componentes de interfaz de usuario, facilitando el acceso a elementos esenciales para la construcción de aplicaciones web, como tablas, gráficos, menús, paneles y formularios.

a. Módulo de cultivos

En esta vista se puede realizar la creación de cultivos, estos cultivos tienen como propiedades obligatorias los campos nombre del cultivo, parcela, dirección, zona, identificador, fecha aproximada de inicio y fecha de conclusión.

Figura 14

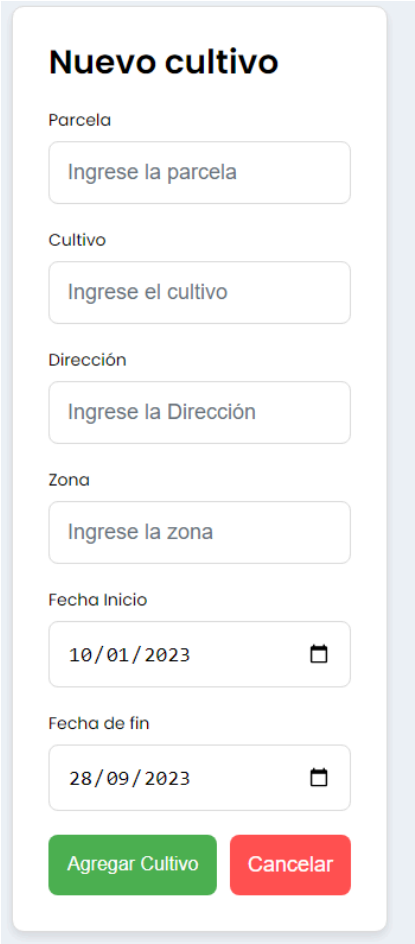
Vista del Módulo de Cultivos

Fuente: *Diseño propio*

La 'parcela' se refiere al nombre de la finca, mientras que la 'dirección' proporciona detalles específicos sobre su ubicación. La 'zona' es el término utilizado para designar la localización general del cultivo. El 'identificador' es un número asignado automáticamente, el cual nos ayuda a registrar de manera única a cada entidad . Las fechas de inicio y conclusión indican el periodo durante el cual se estima que estará presente el cultivo, desde su plantación hasta su cosecha. Esta interfaz permite realizar un seguimiento de los cultivos creados, así como editar sus propiedades básicas y eliminarlos. Además, cuenta con un filtro que facilita la discriminación por cultivos.

Figura 15

Vista Agregación de nuevos cultivos



Fuente: *Diseño propio*

Este modal permite hacer la incorporación de nuevos cultivos con sus campos requeridos.

Figura 16

Vista que permite la edición cultivos

Fuente: *Diseño propio*

Este modal permite editar las propiedades de los cultivos creados.

b. Asignación de actividades

Este apartado tiene como objetivo crear actividades específicas para cada cultivo. Aquí, se presenta un breve resumen de cada actividad y se proporciona la opción de asignar la tarea a la persona correspondiente.

Figura 17

Vista del Módulo de Actividades

Fuente: *Diseño propio*

La actividad incluye las siguientes propiedades: ‘nombre de la actividad’ Este campo nos indica la acción a realizar en el cultivo, se recomienda ser lo más acertado y descriptivo con él. ‘Persona Asignada’ esta columna nos da el responsable de ejecutar la tarea, se puede incluir uno o varios participantes. Por último, se establece un plazo para culminar los trabajos realizados, denominado 'Fecha de realización'. Este tiempo debe calcularse según la estimación de cuánto se tomará realizar las labores establecidas.

Figura 18

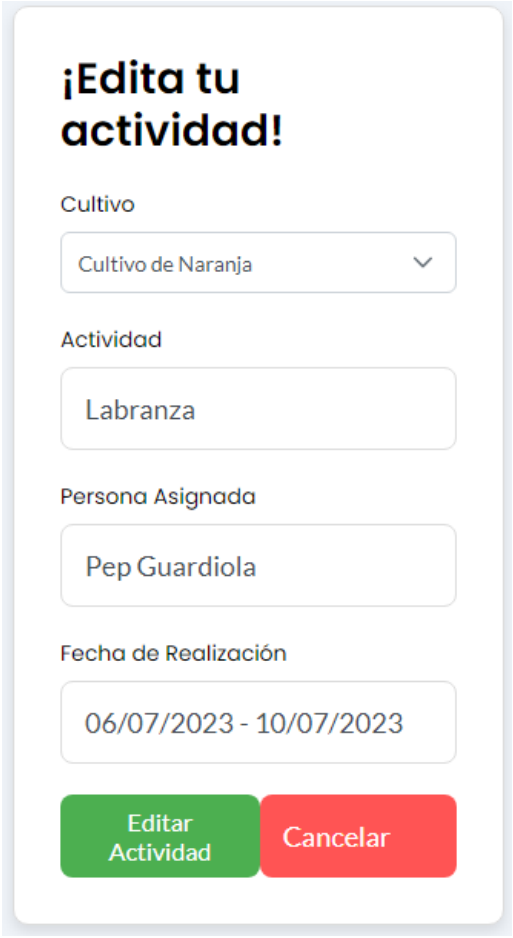
Vista Agregación de nueva actividad

Fuente: *Diseño propio*

Este modal posibilita la asignación de un cultivo a la actividad correspondiente, así como la descripción, la fecha de realización y el responsable de completar el trabajo.

Figura 19

Vista que permite la edición de actividades



Fuente: *Diseño propio*

Esta interfaz permite editar las propiedades ya creadas para cada actividad.

c. Seguimiento de mantenimientos

Esta sección abordará una lista de actividades sugeridas para los cultivos, ofreciendo al usuario la capacidad de visualizar, registrar, editar y eliminar dichas actividades según sus necesidades. Su propósito radica en la gestión y cuidado de los cultivos mediante la implementación de actividades predefinidas que buscan garantizar un manejo eficiente y efectivo de sus recursos.

Figura 20

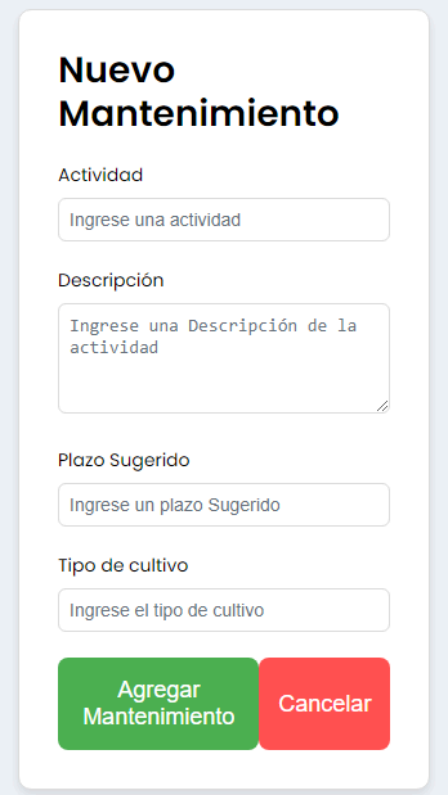
Vista del Módulo de mantenimientos

Fuente: *Diseño propio*

El módulo de actividad incluye las siguientes propiedades: ‘Actividad’ este es el nombre asignado a la tarea a realizar. Se recomienda que sea claro y descriptivo para facilitar la identificación de la acción correspondiente. La ‘descripción’ aquí se puede abordar de mejor manera qué se debe hacer, al dar información detallada de esta. Por último, tenemos el ‘Plazo sugerido’, esta propiedad representa la estimación de tiempo para completar el mantenimiento. Sirve como una guía temporal para la realización de la tarea.

Figura 21

Vista Agregación de nuevo Mantenimiento



Fuente: *Diseño propio*

Modal el cual permite el registro de un nuevo mantenimiento.

Figura 22

Vista que permite la edición Mantenimientos

Fuente: *Diseño propio*

Esta parte permite editar los mantenimientos creados en la aplicación.

d. Seguimiento Financiero.

En este apartado, se ofrece un registro histórico de los gastos en comparación con los ingresos generados por cada cultivo. Aquí, es posible ingresar tanto los valores de inversión como los de ganancia, brindando así una visión clara de cada gasto asociado.

Figura 23

Vista Módulo Financiero

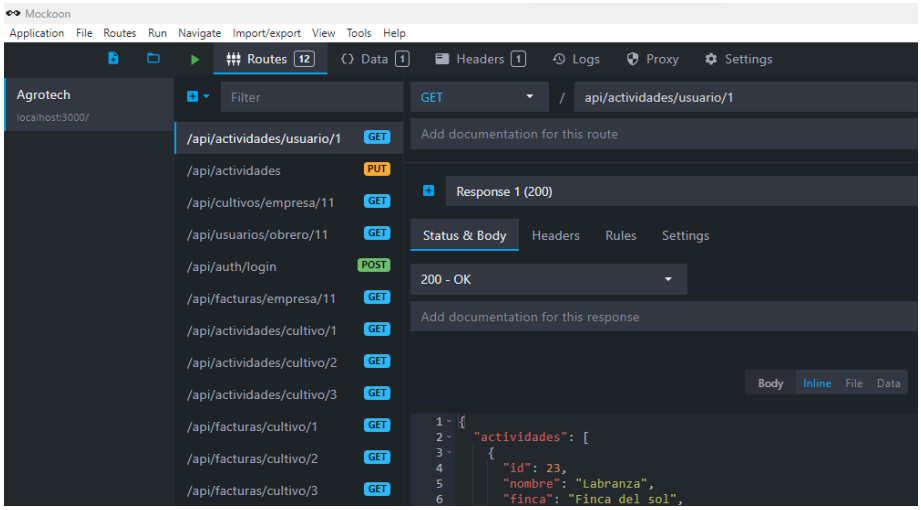
Fuente: *Diseño propio*

6.3.3 Implementación de app móvil.

Durante el primer sprint se creó una api Mock que pudiera simular el comportamiento de los servicios necesarios en la app para el primer sprint, este mock contenía el servicio de login, este respondía los datos del usuario incluyendo el rol. Este API se realizó con el programa Mockoon, se ejecutó en el localhost y se conectó a un emulador de android en red local.

Figura 24

Servicios de Mockoon para el API Mock.



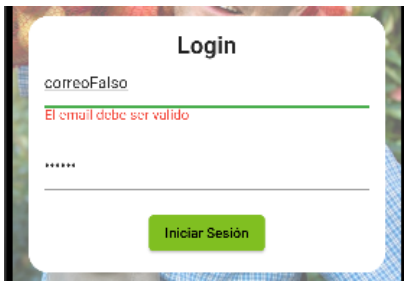
Fuente. *Diseño propio*

En la integración del servicio se usó la arquitectura limpia, agregando las capas de datos y capturando posibles errores, también se implementó un archivo de configuración para cambiar del API de Mockoon al API de back-end según fuera necesario. Adicionalmente se desarrolló la interfaz de login, esta incluye la pantalla de login, las pantallas de inicio de sesión, el ingreso por desconexión y la pantalla de home.

En la pantalla de login, se incorporaron validaciones de correo y clave de acuerdo con la respuesta del login. En caso de desconexión, se añadió un ingreso alternativo a la aplicación mediante un control global de la misma, permitiendo el lanzamiento de servicios de forma asíncrona.

Figura 25

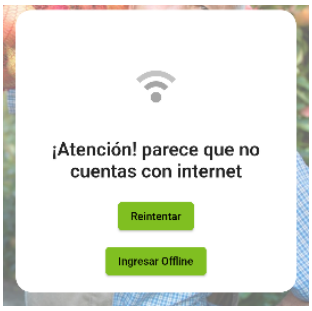
Validaciones de inicio de sesión en login.



Fuente. *Diseño propio*

Figura 26

Camino alternativo para ingresar a la app sin internet.

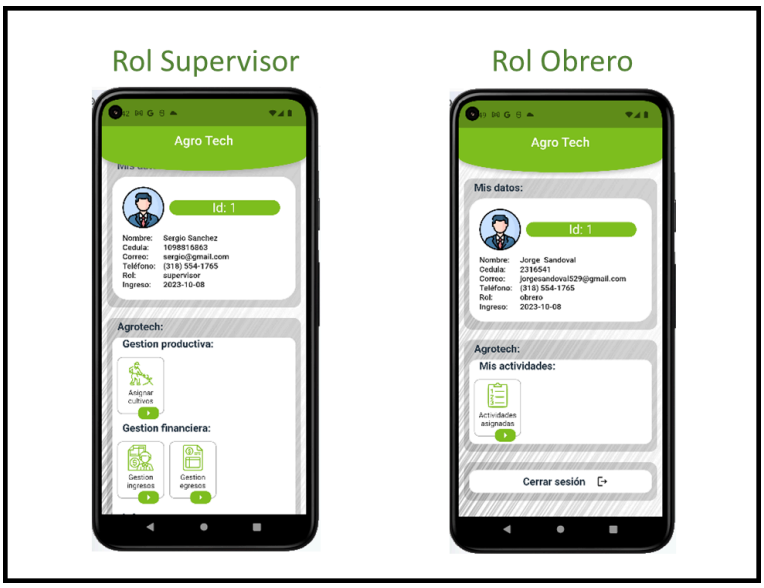


Fuente. *Diseño propio*

Con el control de roles, se establecieron condiciones en el módulo de home para que un usuario solo pueda llevar a cabo acciones correspondientes a su rol. Además, en la parte superior de esta pantalla, se muestran los datos del usuario devueltos por el servicio de login.

Figura 27

Pantalla Home con rol Supervisor y obrero.

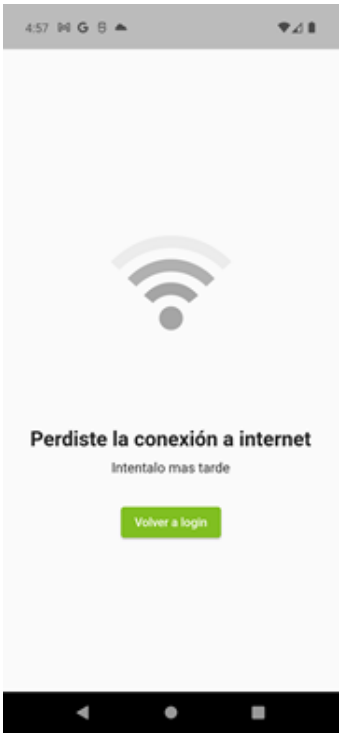


Fuente. *Diseño propio*

Para el manejo asíncrono se desarrolló una pantalla de desconexión que permite a los usuarios cerrar sesión. Se creó un estándar para que la app informe al usuario sobre la desconexión en cada llamado a un servicio.

Figura 28

Pantalla sin conexión a internet.



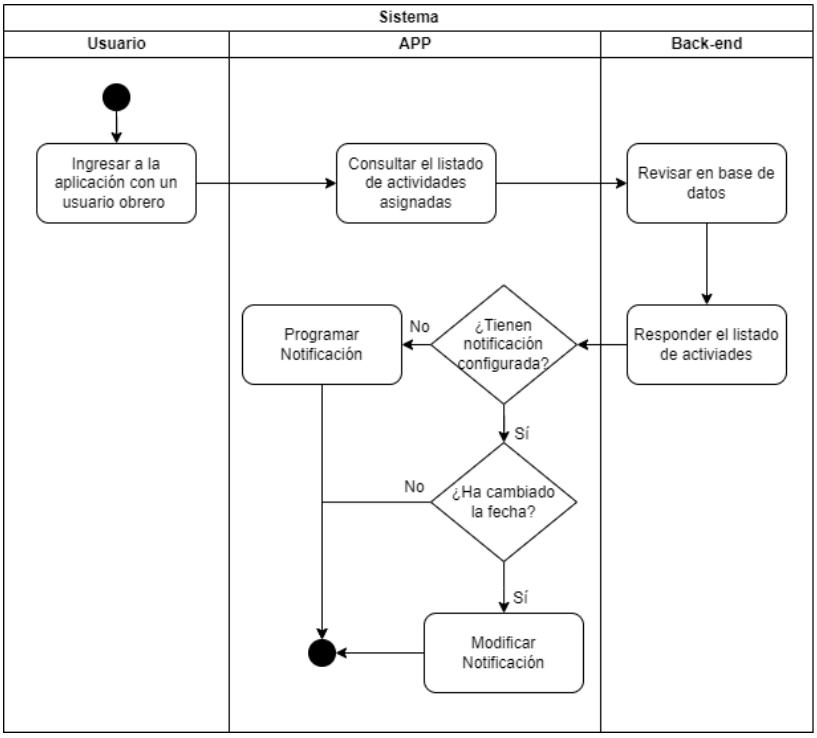
Fuente. *Diseño propio*

En el segundo sprint, se estableció la conexión con los servicios relacionados con mantenimientos. Se implementó una funcionalidad para los supervisores, permitiéndoles revisar un historial de tareas de mantenimiento en cultivos. Además, se les facilita la creación y asignación de tareas. Por otro lado, se complementa esta funcionalidad con la creación de una interfaz para los usuarios obreros que incluye la lista de sus tareas asignadas, un formulario para actualizar el estado de las mismas y notificaciones programadas.

El envío de notificaciones se desarrolló con la finalidad de informar al usuario una única vez sobre una tarea, en la fecha y hora configurada. Cada notificación redirige a la aplicación y, en caso de que se edite la tarea, se cambiará la fecha de notificación o se mostrará nuevamente.

Figura 29

Diagrama de actividades para el manejo de notificaciones.

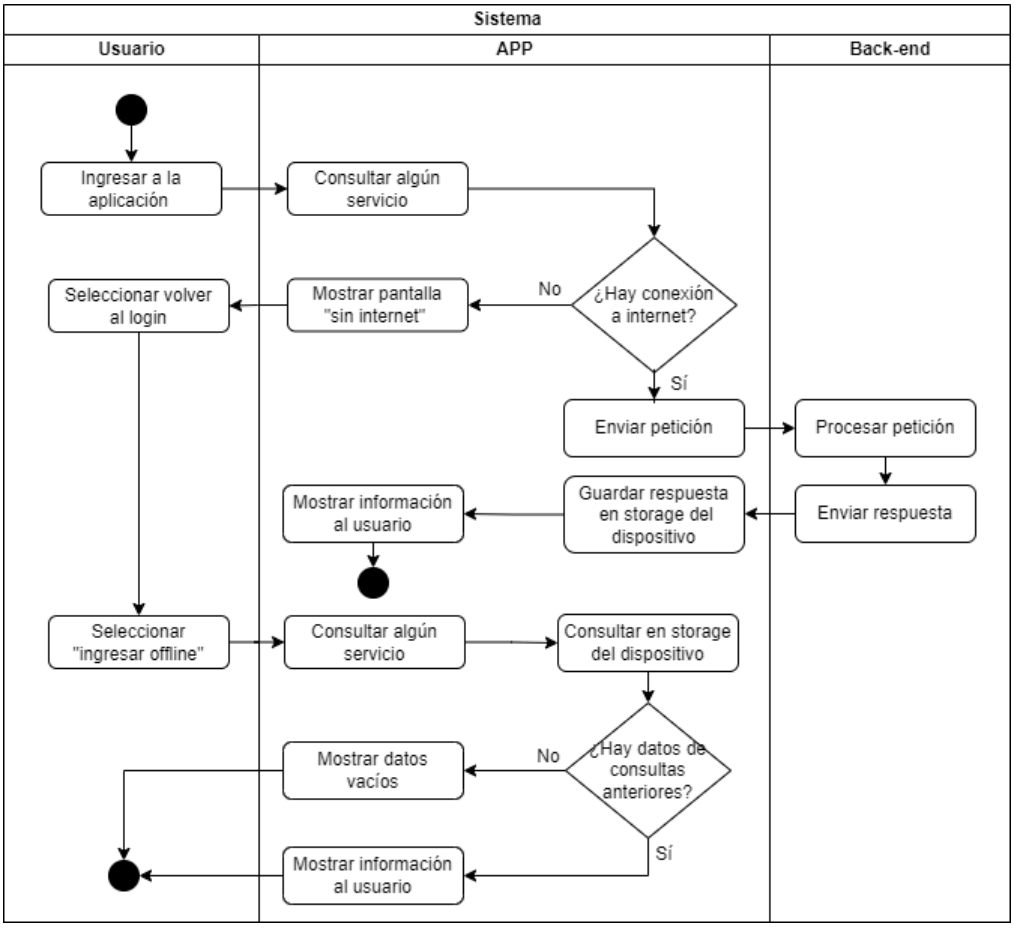


Fuente. *Diseño propio*

En el desarrollo de la asincronía de servicios, se manejó informando al usuario sobre la conectividad y enviándolo al login por seguridad. Si en el login continúa la desconexión, se sugiere ingresar de forma offline, este modo permite revisar la información de las consultas que han sido guardadas en el dispositivo para que el usuario pueda seguir interactuando en la app. el proceso se explica de forma más detallada en la siguiente figura.

Figura 30

Diagrama de actividades para el manejo asíncrono de servicios.



Fuente. *Diseño propio*

Durante el tercer sprint, se integraron los servicios vinculados a los movimientos monetarios. Estos movimientos recibieron un tratamiento para generar diversos informes, y además, se implementó una interfaz que permite filtrar los datos por fecha.

Figura 31

Pantallas de informes.

Fuente. *Diseño propio*

5.3.4 Implementación de microservicios.

Dada la arquitectura de microservicios se realizan las siguientes configuraciones e implementaciones:

a. Configuración

Inicialización del proyecto base que contendrá cada módulo/microservicio teniendo así una configuración global de dependencias y servicios.

b. Conexión a la base de datos

Haciendo uso de JPA(Java Persistence Api) el cual nos permite gestionar de manera sencilla el manejo de la base de datos, siendo así configurada la conexión en cada microservicio desde el application.properties.

Figura 32

Datos para la conexión a la base de datos.

```
spring.datasource.url=jdbc:mysql://190.90.160.170:3306/misdevsc_farm
spring.datasource.username=misdevsc_farm
spring.datasource.password=$Cultivos2023$
```

Fuente. *Diseño propio*

c. Configuración Spring Cloud Netflix Eureka

Con la idea de centralizar los microservicios dado que estos mismos pertenecen y/o se comparten con otros equipos de desarrollo se hace uso de Eureka Server creando un microservicio que lleve a cabo esta operación.

De tal forma spring-cloud-starter-netflix-eureka-server hace parte de las dependencias y lo habilitaremos anotando la clase principal con la notación @EnableEurekaServer exponiendo así en las propiedades de la aplicación el endpoint 8561.

Figura 33

Habilitación de Eureka Server

```
georsan27
@SpringBootApplication
@EnableEurekaServer
public class DiscoveryServerApplication {
    georsan27
    public static void main(String[] args) { SpringApplication.run(DiscoveryServerApplication.class, args); }
}
```

Fuente. *Diseño propio*

Figura 34

Configuración de Eureka Server

```
eureka.instance.hostname=eureka-server
eureka.client.register-with-eureka=false
eureka.client.fetch-registry=false
eureka.client.serviceUrl.defaultZone=http://localhost:8761/eureka/
server.port=8761
```

Fuente. *Diseño propio*

d. Configuración Spring Cloud Netflix Eureka cliente

Ahora para registrar los microservicios añadimos la independencia spring-cloud-starter-netflix-eureka-client y adicionalmente a esto agregamos las variables a la configuración de cada microservicio.

Figura 35

Configuración de Eureka Server Client

```
eureka.client.serviceUrl.defaultZone=http://localhost:8761/eureka
spring.application.name=management-service
server.port=8092
```

Fuente. *Diseño propio*

e. Configuración del ApiGateway

Haciendo uso de la dependencia spring-cloud-starter-gateway se logró centralizar el sistema de rutas de los diferentes microservicios, y así mismo esta dependencia nos da una configuración inicial bastante completa permitiéndonos el balanceo e implementación de patrones como lo es el circuit breaking.

Figura 36

Configuración de las rutas en el Api Gateway

```
#crops
spring.cloud.gateway.routes[10].id=crops
spring.cloud.gateway.routes[10].uri=lb://crop-service
spring.cloud.gateway.routes[10].predicates[0]=Path=/api/crop/**


#facturas
spring.cloud.gateway.routes[14].id=facturas
spring.cloud.gateway.routes[14].uri=lb://management-service
spring.cloud.gateway.routes[14].predicates[0]=Path=/api/facturas/**


#flora
spring.cloud.gateway.routes[15].id=flora
spring.cloud.gateway.routes[15].uri=lb://flora-service
spring.cloud.gateway.routes[15].predicates[0]=Path=/api/plantas/**
```

Fuente. *Diseño propio*

f. Implementación de clase y métodos para el manejo de excepciones

Las diferentes excepciones se podían manejar lanzando una de un solo tipo, sin embargo para la simplicidad del soporte y reconocimiento de errores optamos por implementar diferentes manejadores de excepciones como asimismo implementar excepciones que nos permitieran ser claros a la hora de identificar el motivo por el cual un servicio ha fallado.

Figura 37

Implementación de la clase y métodos que manejan las excepciones

```java
@ControllerAdvice
public class ErrorHandlerConfig {
    Ruben Rodriguez
    @ExceptionHandler(Exception.class)
    public ResponseEntity<Map<String, Object>> handleException(Exception ex) {
        Map<String, Object> response = new HashMap<>();
        response.put("message", "An unexpected error occurred");
        response.put("error", ex.getMessage());
        return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body(response);
    }

    Ruben Rodriguez
    @ExceptionHandler(MethodArgumentNotValidException.class)
    @ResponseStatus(HttpStatus.BAD_REQUEST)
    public ResponseEntity<Map<String, Object>> handleValidationException(MethodArgumentNotValidException ex) {
        Map<String, Object> errors = new HashMap<>();
        errors.put("message", "Validation error");
        errors.put("errors", getValidationErrors(ex));
        return ResponseEntity.badRequest().body(errors);
    }

    Ruben Rodriguez
    @ExceptionHandler(DataNotFoundException.class)
    @ResponseStatus(HttpStatus.NOT_FOUND)
    public ResponseEntity<Map<String, Object>> handleDataNotFoundException(DataNotFoundException ex) {
        Map<String, Object> response = new HashMap<>();
        response.put("message", "Resource not found");
        response.put("error", ex.getMessage());
        return ResponseEntity.status(HttpStatus.NOT_FOUND).body(response);
    }

    1 usage  Ruben Rodriguez
    private Map<String, String> getValidationErrors(MethodArgumentNotValidException ex) {
        Map<String, String> validationErrors = new HashMap<>();
        ex.getBindingResult().getFieldErrors().forEach(error -> {
            validationErrors.put(error.getField(), error.getDefaultMessage());
        });
        return validationErrors;
    }
}
```

Fuente. *Diseño propio*

Figura 38

Implementación de la clase DataNotFoundException

```
public class DataNotFoundException extends RuntimeException {
    11 usages  Ruben Rodriguez
    public DataNotFoundException(String message) { super(message); }

    no usages  Ruben Rodriguez
    public DataNotFoundException(String message, Throwable cause) { super(message, cause); }
```

Fuente. *Diseño propio*

g. Implementación de patrones CircuitBreaker,Retry y Fallback

Se hizo uso de estos patrones para gestionar fallos a la hora del consumo de los servicios, así como reintentar y asignar una alternativa a la hora de encontrarse con un fallo, usando así los métodos para manejar las excepciones. Para la implementación de estos patrones se hace uso de resilience4j el cual nos proporcionará las notaciones @CircuitBreaker y @Retry

Figura 39

Uso de la notación @CircuitBreaker y @Retry, adicionalmente uso de los handlers implementados en la configuración de excepciones.

```
@PutMapping("/edit")
@CircuitBreaker(name = "centro_poblado", fallbackMethod = "handleValidationException")
@Retry(name = "centro_poblado")
public ResponseEntity<Map<String, Object>> editCentroPoblado(@Valid @RequestBody CentroPobladoDTO request){
    Map<String, Object> responseMap = new HashMap<>();
    responseMap.put("centroPoblado", centroPobladoService.edit(request));
    return ResponseEntity.ok(responseMap);
}

Ruben Rodriguez *
@DeleteMapping("/delete/{centroPobladoId}")
@CircuitBreaker(name = "centro_poblado", fallbackMethod = "handleException")
@Retry(name = "centro_poblado")
public ResponseEntity<?> deleteCentroPoblado(@PathVariable Long centroPobladoId){
    centroPobladoService.deleteById(centroPobladoId);
    return ResponseEntity.noContent().build();
}
```

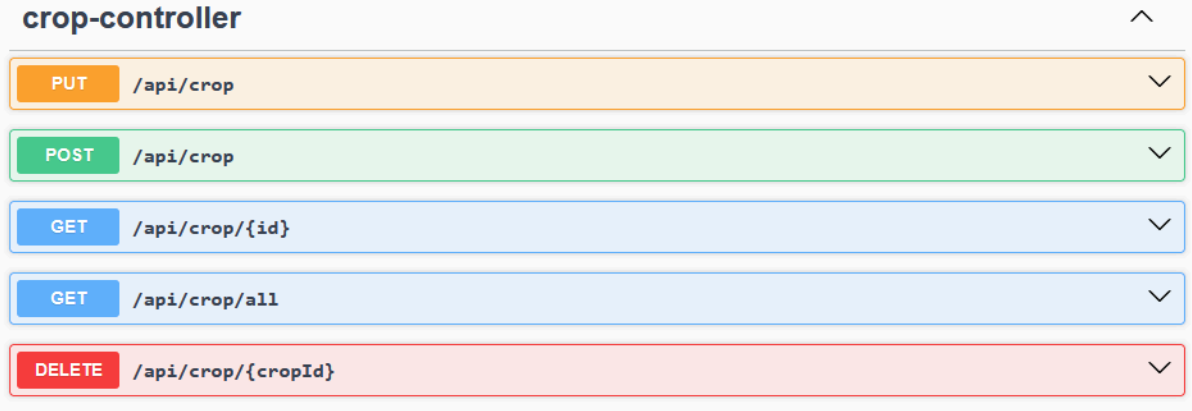
Fuente. *Diseño propio*

h. Microservicio de cultivos

Con este componente nos encargamos de realizar la gestión de los diferentes cultivos asociados a la que sería su empresa correspondiente,

Figura 40

Endpoint de los cultivos



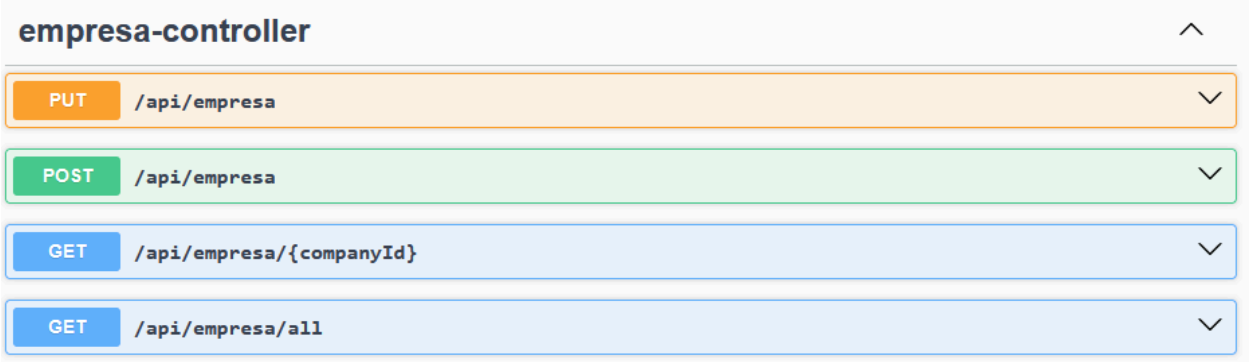
Fuente. *Diseño propio*

i. Microservicio administración nombrado management

Con este componente nos encargamos de realizar la gestión de los diferentes modelos así como actividades de los cultivos, empresas, facturación, etc.

Figura 41

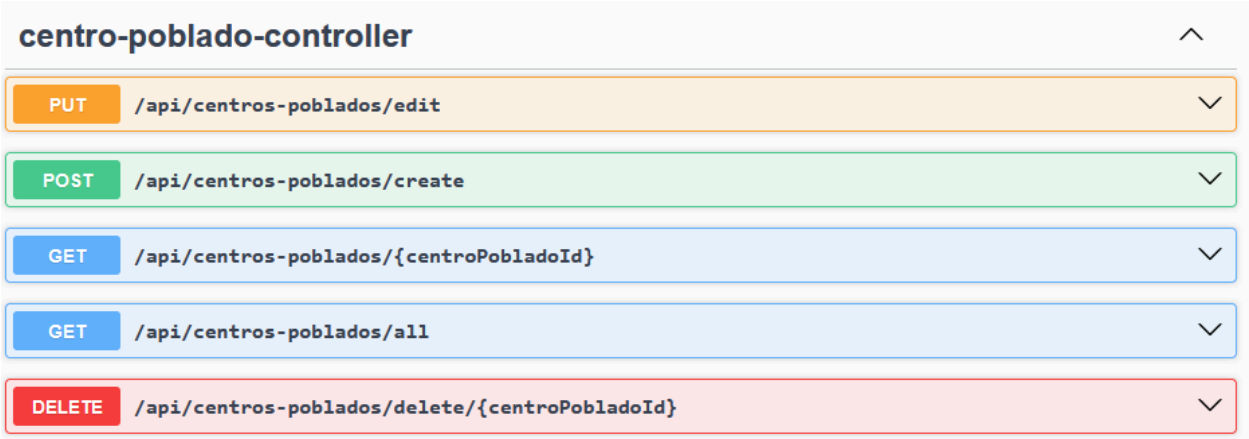
Endpoint de las empresas.



Fuente. *Diseño propio*

Figura 42

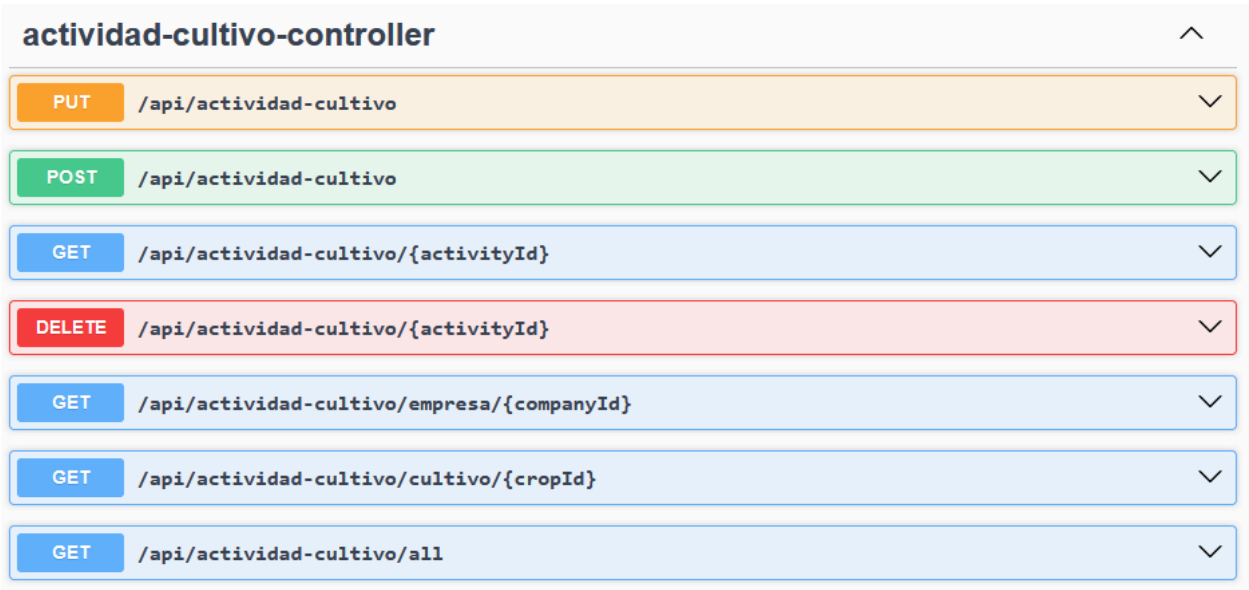
Endpoint de los centros poblados



Fuente. *Diseño propio*

Figura 43

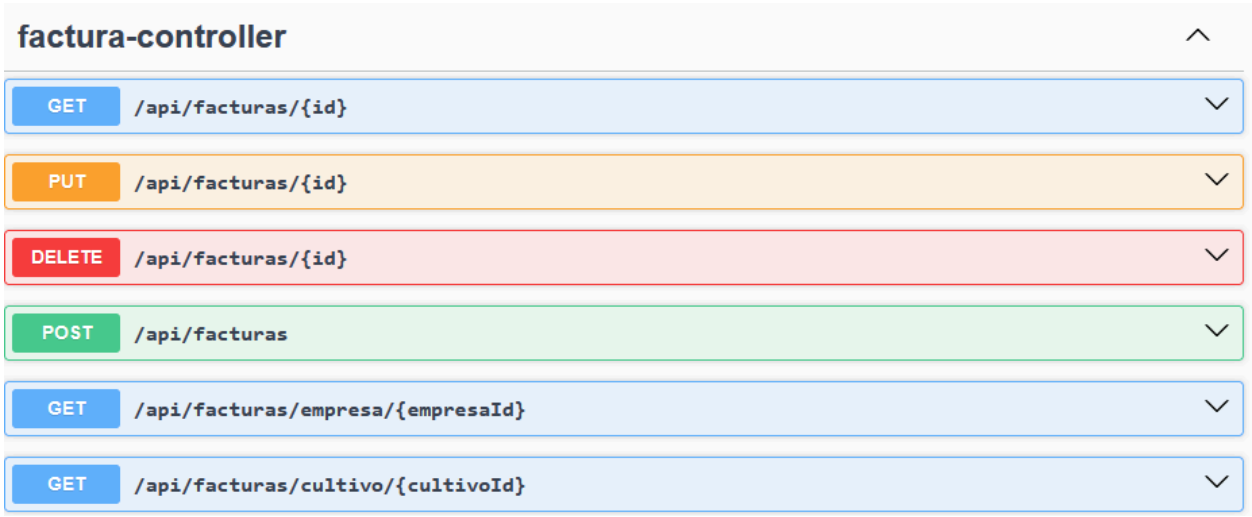
Endpoint de las actividades



Fuente. *Diseño propio*

Figura 44

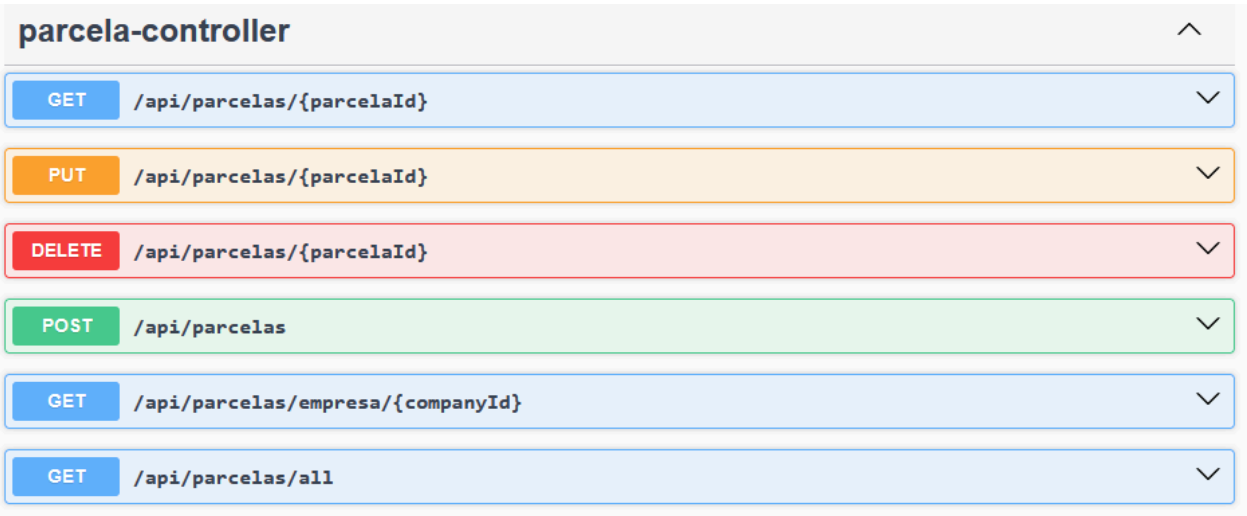
Endpoint de las facturas



Fuente. *Diseño propio*

Figura 45

Endpoint de las parcelas



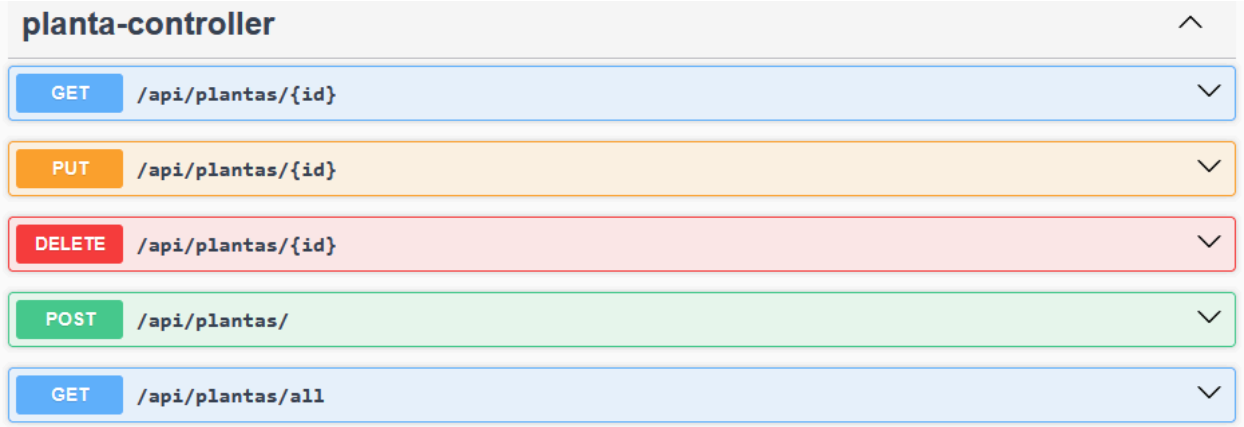
Fuente. *Diseño propio*

j. Microservicio Flora

Con este componente nos encargamos de realizar la gestión de los diferentes modelos así como plantas y mantenimiento

Figura 46

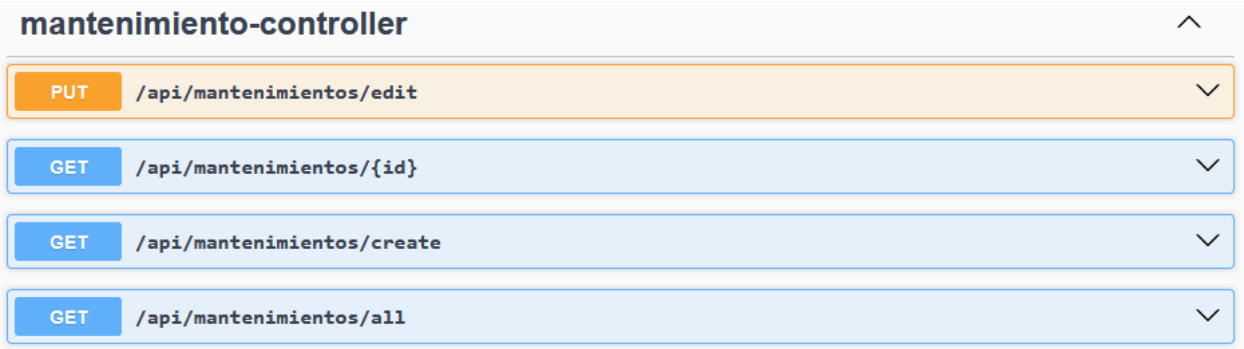
Endpoint de las plantas



Fuente. *Diseño propio*

Figura 47

Endpoint mantenimientos



Fuente. *Diseño propio*

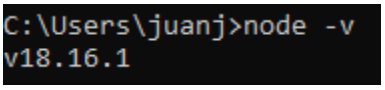
5.4 Fase de Pruebas.

5.4.1 Pruebas web

Para este proyecto, como se mencionó anteriormente, se empleó el framework de Angular. Para llevar a cabo el despliegue de la aplicación a nivel local, resulta imprescindible contar con el entorno de ejecución JavaScript Node.js. Además, es esencial disponer de npm (Node Package Manager), un sistema de gestión de paquetes para Node.js que facilita la instalación, actualización y administración de las dependencias del proyecto.

Figura 48

Verificación versión de Node.js.

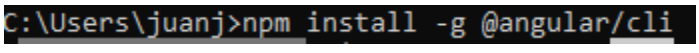


Fuente. *Diseño propio*

Adicionalmente, será necesario utilizar Angular CLI (Command Line Interface), una herramienta de línea de comandos que simplifica la creación, construcción y gestión de proyectos Angular. La integración de Angular CLI en el flujo de trabajo posibilitará una administración eficiente y consistente del proyecto, lo que contribuirá a mejorar significativamente la productividad durante el desarrollo.

Figura 49

Instalación de angular CLI.

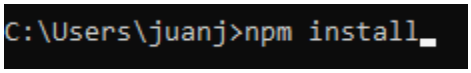


Fuente. *Diseño propio*

Una vez que se cuente con estas herramientas, se debe utilizar la terminal o la línea de comando para ubicarse en el directorio del proyecto Angular e instalar las dependencias definidas en el archivo package.json mediante la herramienta npm.

Figura 50

Instalación de dependencias.

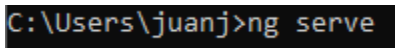


Fuente. *Diseño propio*

Una vez finalizada la instalación de las dependencias, es posible iniciar un servidor local dedicado a nuestra aplicación Angular. Este servidor realiza la compilación y construcción de la aplicación, transformando el código TypeScript a JavaScript y llevando a cabo otras tareas esenciales de construcción. Una vez que el servidor está activo, monitoriza los cambios efectuados en nuestros archivos fuente, permitiendo la reconstrucción automática y la recarga del navegador de la aplicación. Esto proporciona un entorno de desarrollo en tiempo real que simplifica tanto el proceso de desarrollo como la depuración.

Figura 51

Ejecución de la aplicación

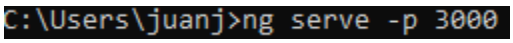


Fuente. *Diseño propio*

Por lo general, Angular CLI inicia la aplicación en el puerto 4200 de manera predeterminada, aunque, este valor puede modificarse mediante la opción '--port' o su equivalente abreviado '-p'.

Figura 52

Cambio del puerto por defecto

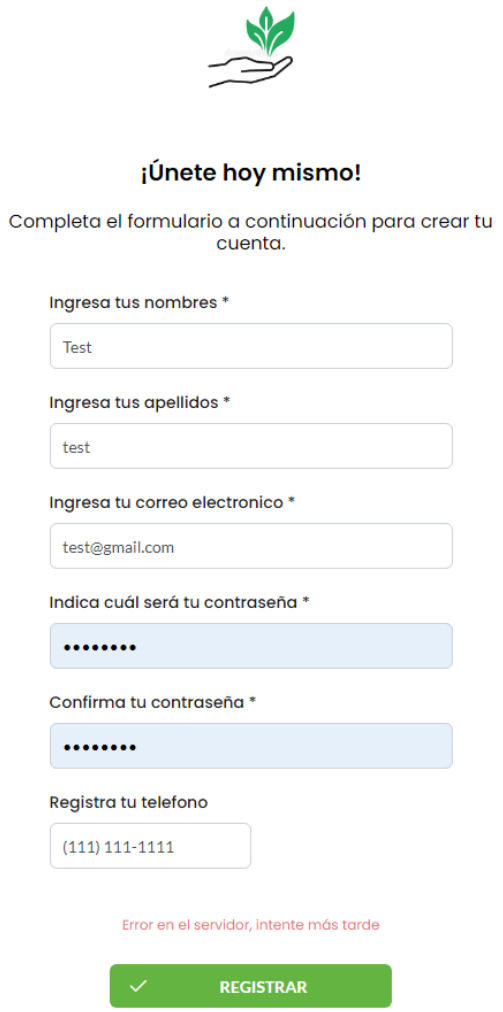


Fuente. *Diseño propio*

Es importante destacar que se debe iniciar primero la parte backend. En caso de comenzar los servicios del frontend antes, se permitirá la visualización del formulario de inicio de sesión, pero no se permitirá el registro ni el acceso a través de cuentas previamente creadas, se mostrará que las credenciales son incorrectas.

Figura 53

Error al iniciar el frontend sin el backend activo.



Fuente. *Diseño propio*

5.4.2 Pruebas app móvil.

En la app móvil se hicieron pruebas unitarias, estas consisten en testear las funciones que tenga la aplicación para así poder encontrar posibles errores e ir mejorando el código, se hicieron un total de 71 pruebas, las cuales pasaron de forma satisfactoria.

Figura 54

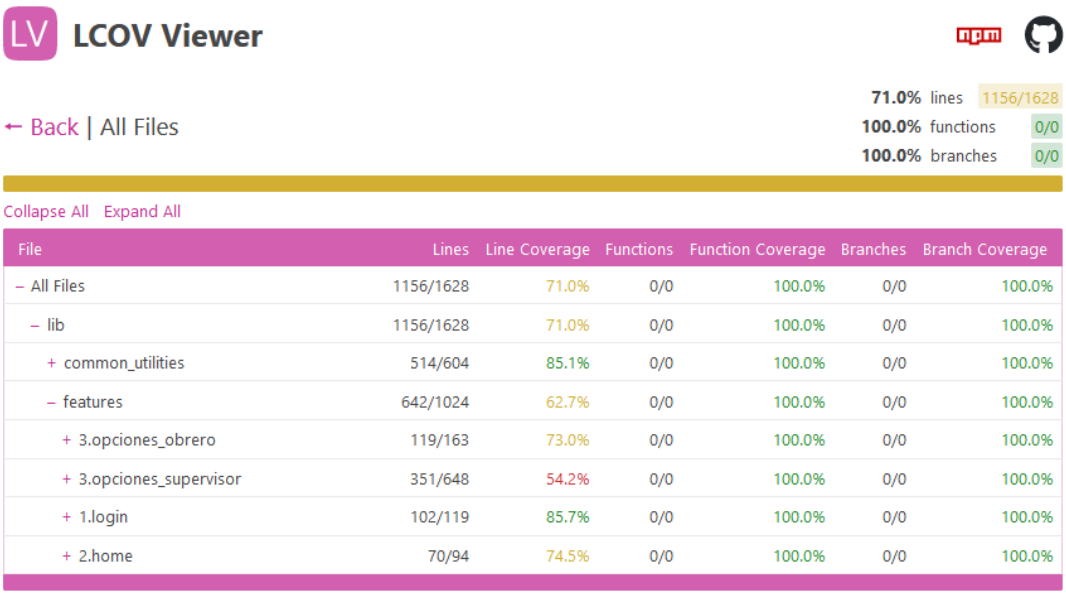
Resultados de las pruebas unitarias.

Fuente. *Diseño propio*

En cuanto al porcentaje de líneas probadas en la aplicación, se logró un 71%. Se alcanzó un 85% de cobertura para la funcionalidad de inicio de sesión, un 74% para la pantalla principal (home), un 74% para las funcionalidades del rol de obrero y un 54% para las funcionalidades del rol de supervisor.

Figura 55

Cobertura de las pruebas unitarias.



Fuente. *Diseño propio*

5.4.3 Pruebas de microservicios.

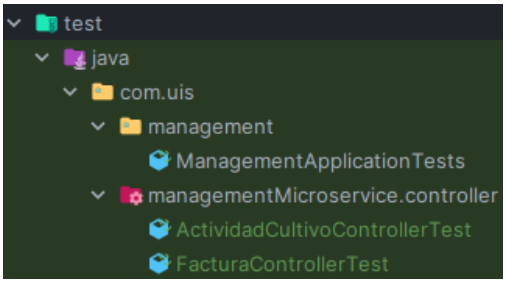
Una vez implementados los diferentes métodos necesarios para el funcionamiento del proyecto, realizaremos tanto pruebas unitarias, estas pruebas por supuesto se iban realizando durante el desarrollo de cada módulo y así mismo se realizaron pruebas de estrés.

a. Pruebas unitarias

Ya habiendo desarrollado los diferentes módulos se realizaron pruebas unitarias con el objetivo de observar el comportamiento de los diferentes módulos. Haciendo uso del IDE Intellij creamos las respectivas clases para realizar las pruebas de los diferentes métodos implementados.

Figura 56

Archivos y carpetas generadas



Fuente. *Diseño propio*

Se hizo uso de JUnit y Spring Boot Starter Test para realizar las diferentes pruebas en los controladores.

Figura 57

Dependencias para testing.

```xml
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
</dependency>
```

Fuente. *Diseño propio.*

Ya creadas las clases y teniendo las herramientas proporcionadas por las dependencias procedemos a implementar las pruebas.

Figura 58

Implementación de una prueba.

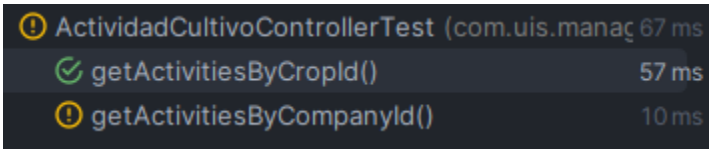
```java
@SpringBootTest(
        webEnvironment = SpringBootTest.WebEnvironment.MOCK,
        classes = Application.class)
@AutoConfigureMockMvc
class ActividadCultivoControllerTest {

    @Autowired
    private MockMvc mvc;

    @Autowired
    private ActividadCultivoController actividadCultivoController;
    new *
    @Test
    void getActivitiesByCompanyId() throws Exception {
        mvc.perform(MockMvcRequestBuilders.get( urlTemplate: "/actividadCultivo/1")
                .contentType(MediaType.APPLICATION_JSON))
                .andExpect((ResultMatcher) jsonPath( expression: "$.actividades[0].id", is( value: 1)))
                .andExpect((ResultMatcher) jsonPath( expression: "$.actividades[0].nombre", is(Integer.parseInt( s: "test"))))
                .andExpect((ResultMatcher) jsonPath( expression: "$.actividades[0].descripcion", is(Integer.parseInt( s: "test"))))
                .andExpect((ResultMatcher) jsonPath( expression: "$.actividades[0].fecha", is(Integer.parseInt( s: "2024-01-08T00:25:57.000+00:00"))))
                .andExpect((ResultMatcher) jsonPath( expression: "$.actividades[0].estado", is(Integer.parseInt( s: "test"))))
                .andExpect((ResultMatcher) jsonPath( expression: "$.actividades[0].valor", is((int) 111.0)))
                .andExpect((ResultMatcher) jsonPath( expression: "$.actividades[0].esPeriodica", is( value: true)))
                .andExpect((ResultMatcher) jsonPath( expression: "$.actividades[0].periodicidadId", is( value: 1)))
                .andExpect((ResultMatcher) jsonPath( expression: "$.actividades[0].usuarioId", is( value: 6)))
                .andExpect((ResultMatcher) jsonPath( expression: "$.actividades[0].cultivatorId", is( value: 1)))
                .andExpect((ResultMatcher) jsonPath( expression: "$.actividades[0].mantenimientoId", is( value: 1)))
                .andExpect((ResultMatcher) jsonPath( expression: "$.actividades[0].facturaId", is( value: 1)));;
```

Fuente. *Diseño propio.*

De esta manera realizamos las pruebas para los diferentes controladores.

Figura 59

Ejemplo resultado de test



Fuente. *Diseño propio.*

b. Pruebas de rendimiento

El objetivo de estas pruebas es determinar la estabilidad de la aplicación, de tal forma se hará uso de la herramienta Jmeter la cual nos permitirá realizar las pruebas de estrés y estabilidad como así mismo proporcionar resultados.

1. Configuración de Jmeter

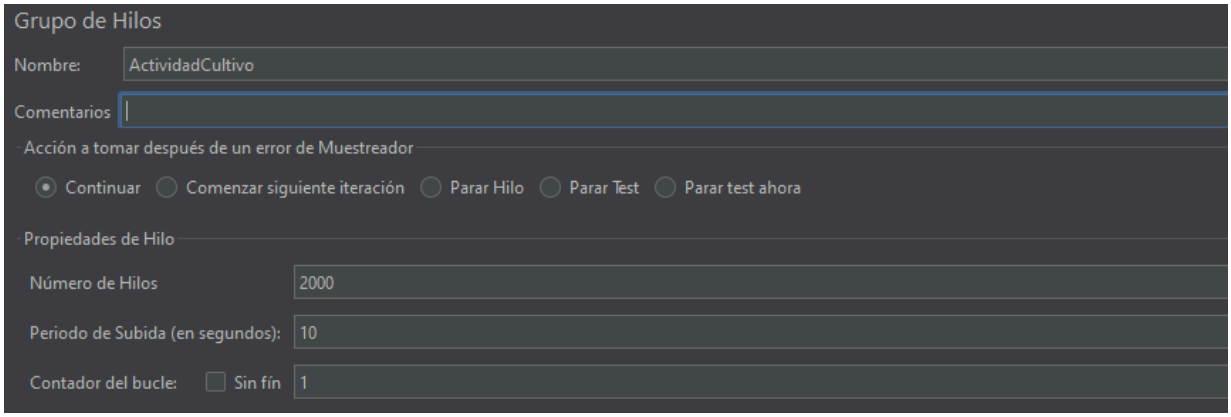
De tal forma se inicia configurando el entorno donde crearemos los grupos de hilos que actuaría como múltiples clientes usando la aplicación de tal forma que podemos indicar la cantidad de estos mismos, el intervalo de incremento y los ciclos, así como las diferentes peticiones http, donde se indicará lo necesario para las peticiones como la url, puerto o data. De igual forma los receptores nos proporcionarán resultados y métricas.

2. Pruebas de estrés

El objetivo de estas pruebas es determinar los límites de la aplicación, de tal forma se hará uso de la herramienta Jmeter la cual nos permitirá realizar las pruebas de estrés y estabilidad como así mismo proporcionar resultados. La configuración de este grupo de hilos se evaluó la cantidad de 2000 hilos durante 10 segundos, de tal manera que cada segundo se realizaron 200 solicitudes, sin embargo esto puede variar debido a factores explicados más adelante.

Figura 60

Configuración del grupo de hilos e intervalo en la prueba de estrés.

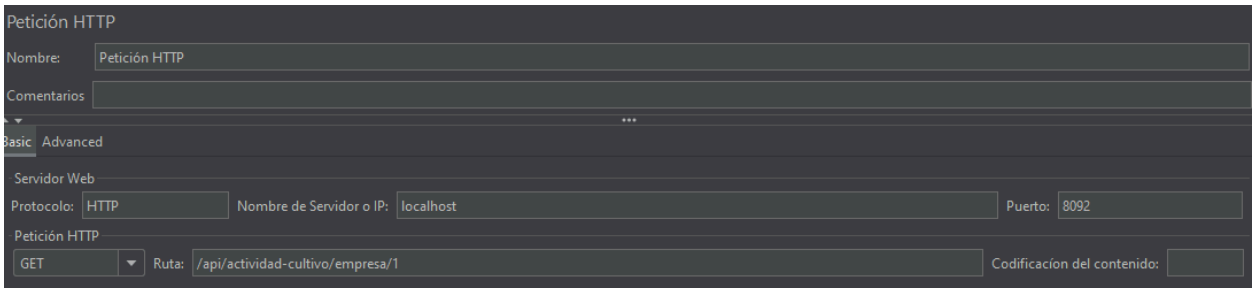


Fuente. *Diseño propio con Jmeter.*

Ya configurados el grupo de hilos se realizó la configuración de las diferentes peticiones. Para este caso en particular se realizará una petición al servicio de actividades por empresa.

Figura 61

Configuración de la petición

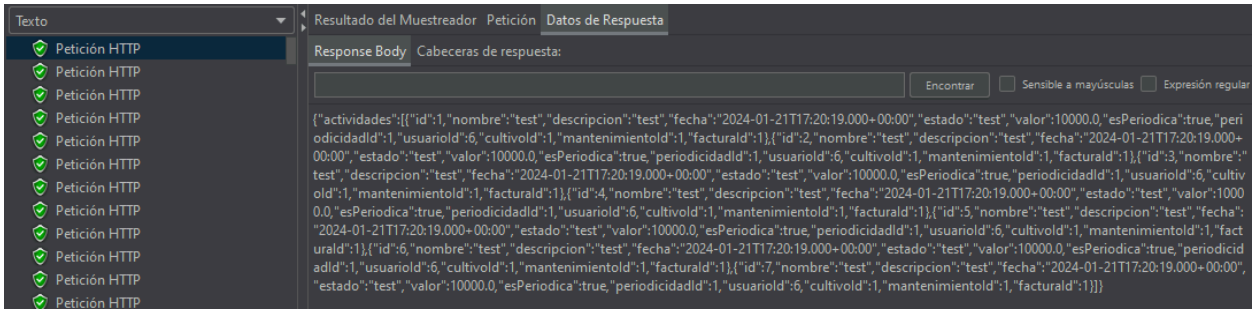


Fuente. *Diseño propio con Jmeter.*

Ya habiendo realizado la prueba y haciendo uso del apartado árbol de resultados proporcionado por Jmeter podemos ver que las 2000 peticiones se realizaron correctamente.

Figura 62

Árbol de resultados del grupo de peticiones en la prueba de estrés.

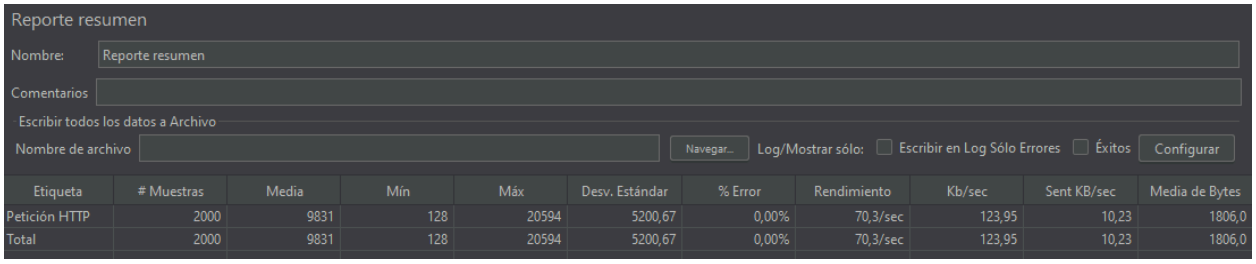


Fuente. *Diseño propio con Jmeter.*

Haciendo uso del apartado reporte resumen se muestran diferentes métricas como lo son la media, el porcentaje de error, el rendimiento, etc.

Figura 63

Reporte resumen en la prueba de estrés.



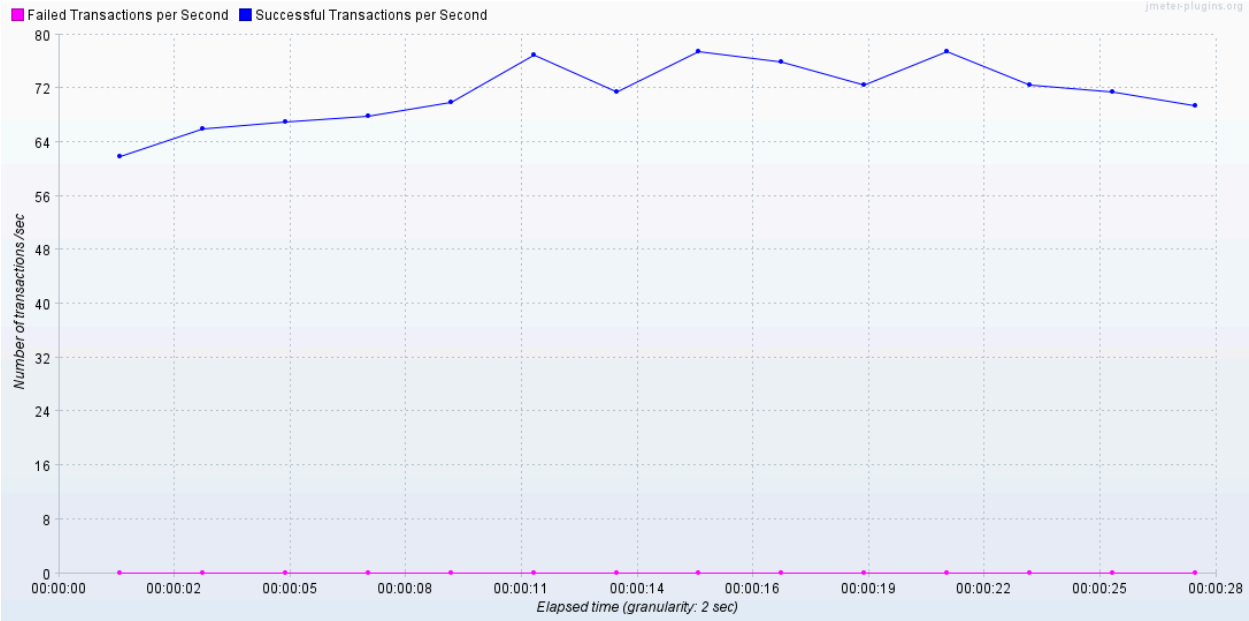
Etiqueta	# Muestras	Media	Mín	Máx	Desv. Estándar	% Error	Rendimiento	Kb/sec	Sent KB/sec	Media de Bytes
Petición HTTP	2000	9831	128	20594	5200,67	0,00%	70,3/sec	123,95	10,23	1806,0
Total	2000	9831	128	20594	5200,67	0,00%	70,3/sec	123,95	10,23	1806,0

Fuente. *Diseño propio con Jmeter.*

De igual se visualizó el número de transacciones durante el intervalo de la prueba, sin embargo no se visualiza el comportamiento ideal, lo que observamos fue que las transacciones por segundo se vieron reducidas, esto pudo ser debido al rendimiento de la máquina usada así como la cantidad de la data de vuelta en la respuesta.

Figura 64

Gráfico de transacciones versus tiempo en la prueba de estrés.



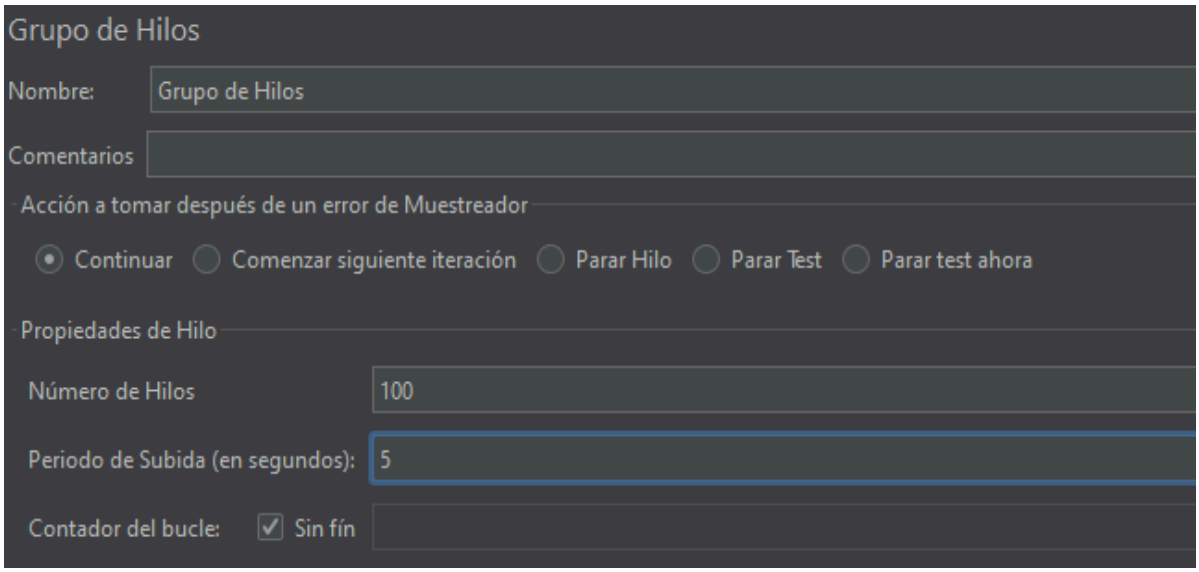
Fuente. *Diseño propio con Jmeter.*

c. Pruebas de estabilidad

El objetivo de estas pruebas es determinar que la aplicación soporta carga continuamente y así mismo mantenerse en pie. De tal forma la configuramos de la siguiente manera el grupo de hilos, 100 hilos en un intervalo de 5 segundos es decir 20 transacciones por segundo, de igual forma el contador sin determina que constantemente se repetirá el proceso sin fin hasta terminar manualmente la prueba.

Figura 65

Configuración del grupo de hilos e intervalo en la prueba de estabilidad.

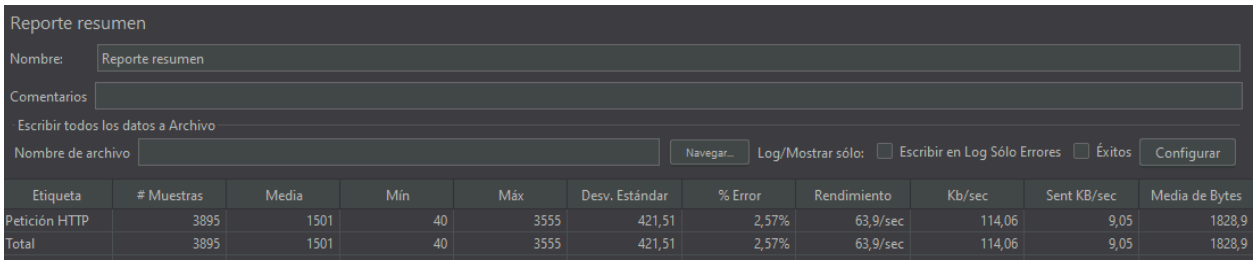


Fuente. *Diseño propio con Jmeter.*

En un periodo de un minuto se enviaron 3895 muestras obteniendo un error del 2,57% sin embargo esto fue debido a detener de manera forzosa la prueba.

Figura 66

Reporte resumen en la prueba de estabilidad.

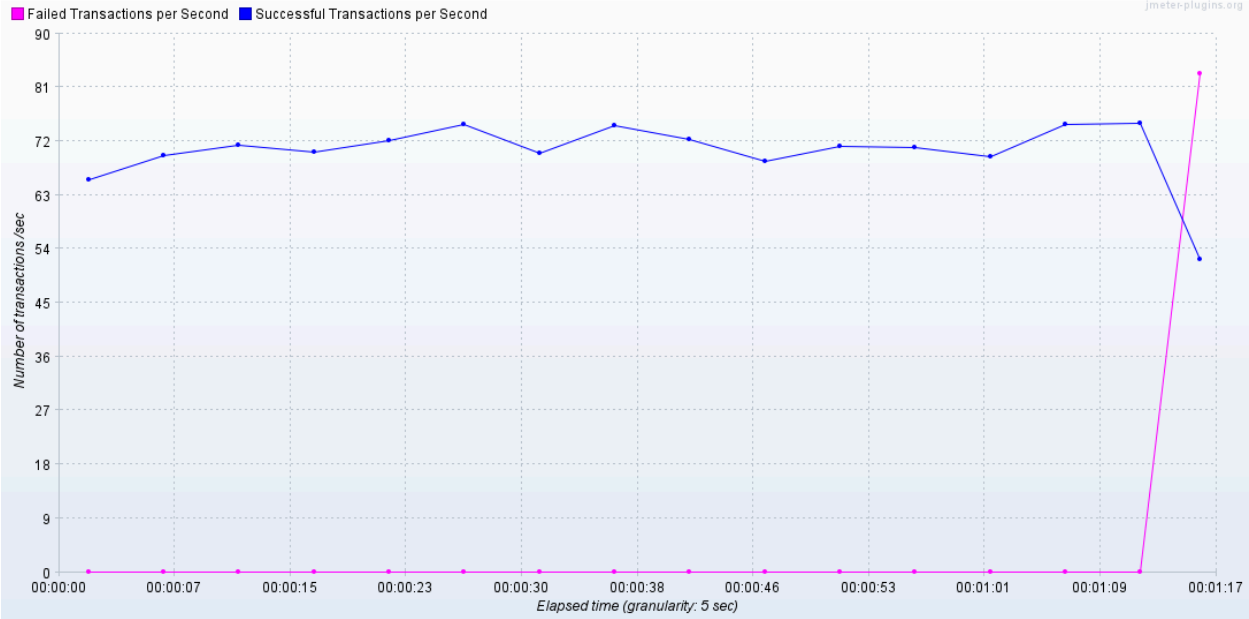


Fuente. *Diseño propio con Jmeter.*

Se pudo observar que la aplicación es estable aun en ocasiones donde la carga es constante durante el intervalo de tiempo, además a pesar de que se encuentra el error esto no afecta la prueba dado al entorno en el que nos encontramos en donde la ejecución de esta se detuvo.

Figura 67

Gráfico de transacciones versus tiempo en la prueba de estabilidad.



Fuente. *Diseño propio con Jmeter.*

6. Conclusiones

El objetivo que orientó la ejecución de este proyecto fue desarrollar un prototipo de aplicación web y móvil el cual facilitara la gestión administrativa y financiera de los cultivos agrícolas a pequeña y mediana escala. Esta tarea multifuncional implicó la colaboración de varios autores, quienes trabajaron conjuntamente para lograr lo planteado en un inicio. Esto requirió de coordinación y seguimiento de las actividades realizadas, por lo cual, se implementó la metodología SCRUM con el respaldo de la aplicación JIRA para la creación y gestión de tareas.

Este enfoque permitió establecer una visión clara del proyecto, trazando una ruta definida para alcanzar los objetivos propuestos.

Este proyecto, ha logrado alcanzar el objetivo el cual lo guiaba, al ofrecer una aplicación web y móvil la cual logra facilitar al usuario la gestión administrativa y financiera de cultivos agrícolas. Con el fin de lograr este objetivo, se centró principalmente en desarrollar una aplicación capaz de gestionar eficientemente los cultivos, actividades agrícolas, labores de mantenimiento, y llevar un registro detallado de los ingresos y egresos de las empresas, lo cual fue realizado satisfactoriamente.

La creación de una herramienta tecnológica que simplifica la administración eficiente de diversas tareas y procesos relacionados con la agricultura permite optimizar el uso de recursos. Facilita la recopilación y gestión de datos agronómicos como el registro de siembras, los tratamientos a seguir y las ganancias generadas. Esta información, acumulada a lo largo del tiempo, proporciona una base sólida para la toma de decisiones informadas, contribuyendo así a la planificación futura de cultivos y a la implementación de prácticas agrícolas sostenibles. Esta ventaja se traduce en la maximización de la eficiencia en la producción.

Se busca resaltar lo aprendido acerca de procesos que, en un principio, podrían parecer ajenos, como los relacionados con la agricultura, pero que brindan un valioso aprendizaje. Es significativo entender cómo llevar a cabo un seguimiento integral de todo el proceso que atraviesa un cultivo y tomar decisiones informadas con los datos obtenidos. Además, aprender acerca de las condiciones a las que se enfrentan las personas involucradas en este ámbito, identificar sus necesidades y responder de manera adecuada, contribuye a la formación del ingeniero para ofrecer soluciones a los problemas que requiere la sociedad y que enfrentará en el futuro.

Gracias a la realización de este proyecto, se logró afianzar los conocimientos adquiridos a lo largo de la vida universitaria en un marco de trabajo real. Este se caracterizó por un proceso integral que abarcó desde la investigación inicial hasta el análisis detallado y, finalmente, un desarrollo práctico y funcional.

7. Recomendaciones y trabajo futuro

El objetivo de este proyecto es presentar un prototipo de la aplicación agrotech; por lo tanto, algunas mejoras o implementaciones adicionales quedaron excluidas en esta versión. Una de las mejoras que podría abordarse es la creación de un administrador general encargado de gestionar las distintas plantas presentes en un cultivo. Esto proporciona un mayor control sobre las siembras, ya que, al conocer las necesidades específicas de cada cultivo, se puede ajustar las prácticas agrícolas como el riego, la fertilización y el control de plagas. Esto resultaría en una optimización de los recursos y la prevención de pérdida de cultivos debido a errores humanos.

Una idea considerada en este proyecto es la implementación de un módulo que facilite la gestión de la ubicación de cada cultivo. Esto sería beneficioso para optimizar tareas como el desplazamiento de los trabajadores y la distribución de los cultivos. Al conocer la ubicación específica de cada uno, se pueden definir actividades condicionadas a factores ambientales particulares, mejorando así la eficiencia operativa en el entorno específico de cada cultivo.

Con el propósito de proporcionar una ayuda adicional al usuario y cumplir con el objetivo establecido para el módulo de mantenimiento, este se podría asociar a un cultivo específico, el cual actuaría como una actividad ya predefinida y cargada en base de datos por parte de los administradores de Agrotech o por el mismo usuario. Cuando un agricultor crea un cultivo,

basándose en parámetros como las plantas presentes, se podría vincular automáticamente un mantenimiento recomendado el cual sería mostrado en la interfaz. El usuario tendría la opción de decidir si desea asignar dicho mantenimiento. Esta funcionalidad está diseñada para ofrecer un marco de trabajo que facilite a usuarios menos experimentados la identificación de actividades adecuadas. Para usuarios más expertos su función sería la de no crear actividades desde cero, esto promueve la reutilización de actividades comunes. Este enfoque favorece la eficiencia y ahorra tiempo de uso en la aplicación “evita reinventar la rueda”.

Al módulo financiero se le pueden incorporar funcionalidades adicionales, como la implementación de filtros que lleven a una gestión más efectiva de las finanzas de la parcela. Entre las opciones de filtrado se incluyen la capacidad de analizar las ganancias netas por cultivo, zonas, plantas y periodos de tiempo. Una vez que se seleccionan uno o varios filtros, los cuales no son mutuamente excluyentes, la vista se ajustaría para mostrar gráficas adaptadas a estos parámetros, así como los datos de las ventas totales, ganancias totales y la inversión requerida.

Esta mejora tiene como finalidad permitir a los usuarios facilitar el análisis de los datos presentados en la aplicación al proporcionar una visión más detallada y personalizada de la información, dando como resultado una mejora en la toma de decisiones.

Referencias Bibliográficas

Casa Fertilizantes y Agroquímicos. (n.d.). Diversificación de cultivos: una práctica más que necesaria [Página web]. https://www.casafe.org/diversificacion-de-cultivos-una-practica-mas-que-necesaria/

Chavez Ramos, L. G. (2021, 17 de junio). Cohesión y Acoplamiento relacionado con la rigidez, fragilidad e inmovilidad. LinkedIn. https://es.linkedin.com/pulse/cohesi%C3%B3n-y-acoplamiento-relacionado-con-la-rigidez-e-chavez-ramos

FAO. (n.d.). TECA - Technologies and Practices for Small Agricultural Producers [Página web]. https://teca.apps.fao.org/teca/

Micex. (n.d.). Plagas de los cultivos [Página web]. https://www.micex.es/leccion/1-plagas-de-los-cultivos/

Microsoft. (2022). Unit Test Basics [Página web]. Microsoft Learn. https://learn.microsoft.com/es-es/visualstudio/test/unit-test-basics?view=vs-2022

Robert C. Martin. (2012, 13 de agosto). The Clean Architecture. Uncle Bob's Blog. https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html

Apéndices

Apéndice A. Repositorio del proyecto

En el siguiente apéndice, se dejará la ruta de los repositorios del proyecto **Repositorio en Github del proyecto.**

Apéndice B. Manual de instalación

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