



TRABALLO FIN DE GRAO
GRAO EN ENXEÑARÍA INFORMÁTICA
MENCIÓN EN ENXEÑARÍA DO SOFTWARE



Desarrollo de un sistema para la gestión de explotaciones ganaderas

Estudiante: Santiago Gutiérrez Gómez

Dirección: Alfonso Castro Martínez

A Coruña, junio de 2024.

A mi familia y amigos

Agradecimientos

En primer lugar, quiero agradecer a mi director del Trabajo de Fin de Grado, Dr. Alfonso Castro Martínez, por siempre estar dispuesto a guiarme y ayudarme en todo lo que necesitase durante todo este largo proceso.

También quiero dar especial gracias a mi familia, mis padres y mis hermanos, ya que sin ellos no hubiese podido sacar adelante esta carrera. Sois los que siempre me apoyaron en los momentos más complicados.

Por último, me gustaría agradecer a mis amigos Pablo y Anna, los cuales me han acompañado de grata manera durante todos estos años.

Resumen

Este Trabajo de Fin de Grado presenta una aplicación web desarrollada para la gestión de pequeñas explotaciones ganaderas, tanto lecheras como cárnicas. Este sistema está formado por diferentes módulos pensados para facilitar las tareas más críticas de las explotaciones, como la gestión del stock de alimentos del ganado, gestión y asignación de tareas a empleados, histórico de producción de leche o carne, gestión del ganado, etc. Además, este sistema está pensado para usar la información de la explotación para generar ciertas métricas que ayuden al productor a ver el estado actual de su negocio y así poder tomar decisiones de negocio con mayor información.

La idea de realizar este sistema surgió por las necesidades identificadas en la explotación ganadera "Santa Ana". En dicha explotación no se usa una herramienta de gestión debido a los altos precios de las licencias de los productos que se encuentran en el mercado actualmente, por lo que para esta pequeña explotación le conviene un sistema más sencillo y simplificado que le permita acceder en todo momento y pueda ver el estado general del negocio. Es importante resaltar que el diseño del sistema se adapta para que pueda ser usado por diferentes explotaciones que padezcan del mismo problema y tengan características similares.

El desarrollo de la aplicación web se llevó a cabo con diferentes tecnologías. Entre estas tecnologías se pueden resaltar Java, Hibernate y Spring Boot para la realización del backend, donde Hibernate se usó para facilitar la persistencia de los datos y Spring Boot para la configuración general. En cuanto al frontend y la interfaz de usuario se uso Javascript y React para poder implementar de manera eficiente una SPA (Single Page Application) que pueda ser usada en diferentes dispositivos.

Abstract

This degree thesis presents a web application developed for managing small livestock farms, both dairy and meat producers. This system consists of different modules designed to facilitate the most critical tasks of the farms, such as managing livestock feed stock, task management and assignment to employees, production history of milk or meat, livestock management, etc. Additionally, this system is designed to use farm information to generate certain metrics that help the producer see the current state of their business and thus make more informed business decisions.

The idea of creating this system arose from the needs identified in the "Santa Ana" live-stock farm. In this farm, a management tool is not used due to the high prices of licenses for products currently available in the market, so for this small farm, a simpler and streamlined

system is more suitable, allowing access at all times and enabling the visualization of the overall state of the business. It is important to highlight that the system's design is adaptable for use by different farms facing the same problem and having similar characteristics.

The development of the web application was carried out using different technologies. Among these technologies, Java, Hibernate, and Spring Boot are noteworthy for the backend development, where Hibernate was used to facilitate data persistence and Spring Boot for general configuration. As for the frontend and user interface, JavaScript and React were used to efficiently implement a Single Page Application (SPA) that can be used on different devices.

Palabras clave:

- SPA
- Explotaciones Ganaderas
- Java
- Spring Boot
- Hibernate
- React
- Redux
- REST
- SCRUM
- Docker

Keywords:

- SPA
- livestock farms
- Java
- Spring Boot
- Hibernate
- React
- Redux
- REST
- SCRUM
- Docker

Índice general

1	Introducción	1
1.1	Motivación y Contexto General	1
1.2	Objetivos	2
1.3	Estructura de la memoria	2
2	Tecnologías y herramientas utilizadas	4
2.1	Software	4
2.1.1	Base de datos	4
2.1.2	Backend	4
2.1.3	Frontend	5
2.1.4	Despliegue	5
2.1.5	Control de versiones	5
2.1.6	Planificación del trabajo	6
2.1.7	Diagramas	6
2.2	Hardware	6
3	Metodología	7
3.1	Scrum	7
3.1.1	Características	8
3.2	Kanban	9
3.3	Adaptaciones al proyecto	10
4	Análisis	13
4.1	Análisis de requisitos	13
4.1.1	Requisitos funcionales	13
4.1.2	Requisitos no funcionales	14
4.2	Historias de usuario	15
4.3	Estimación Inicial	19

4.3.1	Estimación en tiempo	19
4.3.2	Estimación del coste	21
5	Diseño	22
5.1	Arquitectura	22
5.2	Estructura	23
5.3	Base de datos	24
5.3.1	UML	24
5.3.2	Esquema relacional	26
6	Desarrollo e implementación	29
6.1	Desarrollo	29
6.1.1	Primera iteración	29
6.1.2	Segunda iteración	35
6.1.3	Tercera iteración	43
6.1.4	Cuarta iteración	50
6.1.5	Quinta iteración	59
6.2	Testing	66
6.2.1	Pruebas de integración	66
6.2.2	Pruebas de sistema	67
6.2.3	Pruebas de aceptación	68
6.3	Despliegue	68
6.3.1	Docker	68
6.3.2	Kubernetes	68
7	Conclusiones y trabajos futuros	69
7.1	Conclusiones	69
7.2	Trabajos para el futuro	70
Lista de acrónimos		73
Glosario		74
Bibliografía		75

Índice de figuras

3.1	Scrum aplicado en el proyecto	12
4.1	Fórmula del coste estimado	21
5.1	Arquitectura del sistema	23
5.2	Estructura del sistema	25
5.3	Diagrama UML de la base de datos	27
5.4	Esquema relacional de la base de datos	28
6.1	Tablero Kanban al comenzar la primera iteración.	31
6.2	Login en la aplicación.	32
6.3	Lista de empleado.	33
6.4	Actualizar perfil.	33
6.5	Tablero Kanban en la segunda iteración.	38
6.6	Crear tarea.	39
6.7	Listar y filtrar tareas.	40
6.8	Listado de tareas en Dashboard para empleados.	40
6.9	Actualizar y eliminar datos de compra de alimentos.	40
6.10	Tablero Kanban en la tercera iteración.	46
6.11	Listado compras de alimentos.	47
6.12	Visualización y edición de datos de ganado.	47
6.13	Registrar consumo de alimento.	48
6.14	Gráfica consumo de un lote de alimento.	48
6.15	Tablero Kanban en la cuarta iteración.	53
6.16	Listado y filtrado de ganado.	55
6.17	Listado y filtrado de consumo de comida.	55
6.18	Visualización de datos de pesaje.	56
6.19	Visualización de datos de ordeño.	56

6.20	Visualización de datos de ordeño.	56
6.21	Tablero Kanban de la quinta iteración.	60
6.22	Gráfico de consumo de comida de un animal.	61
6.23	Gráfico de evolución de peso de un animal.	62
6.24	Gráfico de producción lechera de un animal.	62
6.25	Gráfico de stock de alimentos.	63
6.26	Gráfico de consumo de comida de la explotación.	63
6.27	Gráfico de producción lechera de la explotación.	64
6.28	Gráfico de pesajes destinados a la producción cárnica.	64
6.29	Coveraga de las pruebas de integración.	67
6.30	Pruebas de sistema con Postman.	67

Índice de cuadros

4.1	Estimación inicial de las historias de usuario	19
6.1	US-01 Iniciar sesión	30
6.2	US-02 Cerrar sesión	30
6.3	US-03 Actualizar datos del perfil	30
6.4	US-04 Crear empleado	30
6.5	US-05 Listar empleado	31
6.6	US-06 Suspender empleado	31
6.7	US-07 Eliminar empleado	31
6.8	Comparación del tiempo estimado y el real de la primera iteración	32
6.9	US-08 Crear tarea	35
6.10	US-09 Actualizar datos de tarea	35
6.11	US-10 Eliminar una tarea	36
6.12	US-11 Cambiar estado de tarea	36
6.13	US-12 Listar tareas	36
6.14	US-13 Registrar compras de alimentos	36
6.15	US-14 Editar datos de compra	37
6.16	US-15 Eliminar compra de alimentos	37
6.17	US-16 Visualizar datos de compra	37
6.18	Comparación del tiempo estimado y el real de la segunda iteración	38
6.19	US-17 Listar compra de alimentos	43
6.20	US-19 Registrar nuevo ganado	44
6.21	US-20 Editar datos de ganado	44
6.22	US-21 Cambiar estado de ganado	44
6.23	US-22 Visualizar datos de ganado	44
6.24	US-25 Registrar Consumo de alimento	45
6.25	US-18 Ver gráfica de consumo de una compra de alimento	45

6.26 US-26 Eliminar consumo de alimento	45
6.27 Comparación del tiempo estimado y el real de la tercera iteración	46
6.28 US-23 Listar ganado de explotación	50
6.29 US-27 Listar consumos de alimento	51
6.30 US-28 Regsitrar pesaje	51
6.31 US-29 Editar datos de un pesaje	51
6.32 US-30 Eliminar Registro de pesaje	51
6.33 US-31 Visualizar datos de pesaje	52
6.34 US-32 Listar registros de pesajes	52
6.35 US-33 Regsitrar ordeño	52
6.36 US-34 Editar datos de ordeño	52
6.37 US-35 eliminar Registr deo ordeño	53
6.38 US-36 Visualizar datos de ordeño	53
6.39 Comparación del tiempo estimado y el real de la cuarta iteración	54
6.40 US-37 Listar registros de ordeño	59
6.41 US-24 Ver gráfica sobre un animal	59
6.42 US-38 Dashboard de gráficas	60
6.43 Comparación del tiempo estimado y el real de la quinta iteración	61

Capítulo 1

Introducción

ESTE capítulo, se presentan el estado de la situación actual y con ella las motivaciones para realizar este proyecto, también se mencionarán los diferentes objetivos a cumplir con el proyecto y como sería su modelo de negocio. Finalmente, se describe la organización de esta memoria.

1.1 Motivación y Contexto General

El origen de este sistema surge gracias al estado actual de la explotación ganadera "Santa Ana", la cual es considerada como una explotación pequeña (menos de 5 empleados). En esta explotación no hay una herramienta para poder gestionar las diferentes tareas relacionadas con la producción de leche y carne, la distribución y planificación de las tareas, y tampoco se lleva un registro histórico de la producción que pueda ser accesible por el jefe/dueño de forma remota. Gracias a todos estos inconvenientes, se decidió desarrollar una herramienta que satisfaga las necesidades de esta explotación y las de otras que estén en una situación similar, ya que las herramientas que están actualmente en el mercado están mayormente dirigidas a explotaciones de mayor envergadura.

La herramienta va a consistir en un modelo [Software as a Service \(SaaS\)](#), ya que esto permitiría a los usuarios poder acceder al sistema siempre que tengan una conexión a Internet. Este modelo trae muchas ventajas para este contexto, debido a que con un SaaS el mantenimiento del sistema y su operabilidad recae sobre la empresa desarrolladora, facilitando así las futuras actualizaciones y mantenimiento del mismo.

Al desear que el sistema pueda ser usado por varias explotaciones, se sacrificará la capacidad de personalización de ciertos aspectos de la herramienta por parte de los usuarios. De esta manera, las funcionalidades e interfaz serán lo suficientemente sencillas y fáciles para que la curva de aprendizaje para usar el sistema sea la menor posible.

1.2 Objetivos

El principal objetivo de sistema es proporcionar a pequeños productores ganaderos una herramienta sencilla de usar, completa y eficiente que facilite las tareas diarias de la explotación. Además se busca que todos los datos introducidos en la aplicación generen métricas relevantes al productor para saber como esta llevando su negocio y de esta forma poder tomar decisiones basadas en estas:

1. **Gestión de producción lechera y cárnica:** Módulo encargado de la creación, edición y visualización de datos de producción.
2. **Gestión del ganado:** Módulo encargado de la creación, edición y visualización de los datos del ganado.
3. **Gestión del alimento:** Módulo encargado de la creación, edición y visualización de los datos de las compras de alimento y su respectivo consumo.
4. **Gráficos y métricas:** Módulo encargado de la creación de métricas y gráficos sobre los datos de producción (leche y carne) y el consumo de alimentos.
5. **Tareas:** Módulo encargado de la creación, edición y visualización de las tareas ha realizar en la explotación.
6. **Empleados:** Módulo encargado de la creación, visualización y edición de los datos de los empleados de la explotación.

1.3 Estructura de la memoria

la memoria que documenta este proyecto sigue la siguiente estructura:

1. **Introducción:** En este capítulo contextualizamos el proyecto junto con los motivos de su creación y se dan a conocer los objetivos que se buscan alcanzar con el mismo.
2. **Tecnologías y herramientas utilizadas:** Este capítulo define y explica la selección de las diferentes tecnologías y herramientas para el desarrollo del sistema.
3. **Metodología de desarrollo:** Se define la metodología usada, como se aplicó al proyecto y sus principales características.
4. **Análisis:** Este capítulo se centra en el análisis de requisitos (funcionales y no funcionales) del sistema. Además se habla de las estimaciones de tiempo y costes.

5. **Diseño de la aplicación:** En este capítulo nos centramos en el diseño de los diferentes componentes que componen el sistema.
6. **Desarrollo:** Este capítulo se centra en la implementación de sistema, las pruebas realizadas y el despliegue del sistema.
7. **Conclusiones y mejoras futuras:** En este capítulo se extraen conclusiones obtenidas durante el desarrollo y se plantean posibles mejoras y trabajos futuros sobre el sistema desarrollado.

Capítulo 2

Tecnologías y herramientas utilizadas

EN este catulo se habla sobre las diferentes tecnologías y herramientas utilizadas durante el desarrollo de la aplicación.

2.1 Software

2.1.1 Base de datos

Se ha optado por MySQL [1] como Sistema de Gestión de Base de Datos (SGBD) de la aplicación, ya que es una tecnología en la cual tengo bastante experiencia a lo largo de estos ultimo años. Además, cuanta con un gran rendimiento cuando trabaja con gran cantidad de datos.

Durante el desarrollo se utilizó el entorno gráfico de visualización y edición de base de datos que trae el Integrated Development Environment (IDE) IntelliJ IDEA [2], logrando así tener un entorno cómodo y fácil de usar que estuviese junto con el código desarrollado.

2.1.2 Backend

Para el backend **backend** se usaron varias tecnologías en conjunto, las cuales se complementaban entre sí para facilitar el desarrollo lo más posible. Como principales tecnologías destacan Java [3] como lenguaje de programación y el uso de Spring Boot [4] junto con Hibernate [5].

Se eligió Java debido a la gran experiencia previa del equipo de desarrollo en este lenguaje y por se un lenguaje multiplataforma. Al ser usado con Spring boot, este nos facilita mucho la configuración general del proyecto para centrarnos más en el desarrollo. Por otro lado, Hibernate nos facilita de gran manera la interacción con la base de datos debido al mapeo

objeto-realcial. Cabe destacar que estas tres tecnologías cuentan con una gran comunidad activa detrás de ellas, por lo que se hace muy fácil encontrar información y soporte a diferentes problemas que puediesen surgir a lo largo del desarrollo.

Para la implementación de pruebas automatizadas se usaron las librerías de JUnit 5 [6]. Con esto aseguramos el correcto funcionamiento y la detección de errores.

En cuanto a la gestión de dependencias del proyecto se usó Maven [7]. La decisión de usar este gestor viene dada principalmente por la familiaridad y experiencia del equipo de trabajo con esta tecnología.

Como IDE se ha usado IntelliJ IDEA Ultimate 2024.1 debido a su gran interfaz y por tener incorporada herramientas de gran utilidad, como por ejemplo el visor de datos de la base de datos y la posibilidad de ejecutar los test con análisis de cobertura.

Por último, para testear los endpoints de la aplicación se usó la herramienta Postman [8].

2.1.3 Frontend

En cuanto a las tecnologías utilizadas para el desarrollo del frontend **frontend**, se ha optado por el uso de JavaScript [9] junto con las librerías de React [10] y Redux [11]. Las librerías de React nos permiten crear los componentes de la **Single Page Application (SPA)** mientras que Redux se encarga de la gestión del estado de la aplicación.

Además, cabe destacar el uso de la librería Apache Echarts [12], la cual nos proporciona una gran cantidad de opciones para la creación de gráficos de manera sencilla con los diferentes datos de la aplicación.

Por último, en la capa frontend se utiliza Node.js [13] para la ejecución de código JS desarrollado. Esta entorno nos permite ejecutar código javascript en diferentes plataformas, facilitando así la creación y ejecución de aplicaciones web.

2.1.4 Despliegue

En cuanto al despliegue de la aplicación, se piensa utilizar la combinación de Docker [14] y Kubernetes [15], donde Docker permite empaquetar aplicaciones en contenedores con sus dependencias y Kubernetes nos da la posibilidad de gestionar esos contenedores de manera sencilla.

2.1.5 Control de versiones

En cuanto al control de versiones del proyecto, se usó Git [16], ya que permite el seguimiento de los cambios realizados en el código de manera detallada y permite el desarrollo en paralelo mediante ramas aisladas para evitar posibles conflictos entre desarrollos realizados por diferentes miembros del equipo.

Para facilitar el uso de Git se optó por utilizar GitHub [17], ya que este proporciona una interfaz web que simplifica el uso de Git por parte de todo el equipo de desarrollo.

2.1.6 Planificación del trabajo

Para la planificación del trabajo se uso la herramienta web Trello [18], ya que esta nos proporciona la opción de crear un tablero Kanban totalmente personalizable para organizar el trabajo de cada Sprint que se realize a lo largo del proyecto.

2.1.7 Diagramas

Esta memoria cuenta con varios diagramas creadas con la herramienta web Draw.io [19]. Esta herramienta proporciona una interfaz web sencilla que permite crear todo tipo de diagramas y la posibilidad de guardarlos en diferentes formatos (pdf, png, etc).

2.2 Hardware

El desarrollo del proyecto se hizo en un ordenador ASUS ROG Strix G531GT con estas características:

- **Procesador:** Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz
- **Memoria:** 16 GB
- **Disco:** 1 TB SSD
- **Sistema operativo:** Ubuntu 20.04.6 LTS

Capítulo 3

Metodología

En este catulo se habla sobre la metodoloía de desarrollo escogida para la realización del trabajo del proyecto.

En concreto, la metodología escogida para el desarrollo del proyecto ha sido una adaptación de Scrum, debido a que la versión original de Scrum esta pensada para un equipo de trabajo de 3 a 9 integrantes y este proyecto se trata de un trabajo individual. Uno de los principales motivos de la elección de esta metodología es que es la metodología en la cual tengo mas experiencia y me siento más comodo al desarrollar. Otro motivo de su elección es por las diferentes revisiones periódicas en los *sprints*, dandono así la ventaja de poder detectar los problemas y errores de manera mas eficiente.

3.1 Scrum

Scrum [20] es una de las metodologías ágiles mas utilizadas hoy en día en el desarrollo de software, aunque también se puede aplicar en otros tipos de proyectos. Una de las ventajas de esta metodología es que, gracias a su enfoque iterativo, podemos revisar de manera frecuente el estado del producto en funcionamiento al terminar carda iteración.

Scrum es una metodología iterativa, es decir, se compone de ciclos de desarrollo llamados **Sprints**. Lo importante de los Sprints es que al final de cada uno de estos se debe poder observar software real en funcionamiento, con el objetivo de sea validado y tomar una decisión de si liberarlo su entorno de producción o realizar otro sprint para su refinamiento. La gran ventaja de este enfoque iterativo es que permite la entrega continua y la capacidad de adaptarse a diferentes situaciones que puedan surgir a lo largo del desarrollo de producto.

Scrum esta fuertemente dirigido al negocio y la rentabilidad; es el negocio quien establece las funcionalidades críticas y prioritarias del proyecto, dejando así la responsabilidad al equipo de desarrollo de autoorganizarse para entregar las funcionalidades prioritarias. Este enfoque proporciona gran valor para el cliente, ya que en cada iteración siempre se tiene como objetivo

cumplir con las prioridades de este.

3.1.1 Características

Las características mas relevantes de la metodología Scrum son:

- **Principios y filosofía:**

1. **Enfoque en el valor del cliente:** Scrum se centra en proporcionar el mayor valor posible al cliente mediante entregas continuas de incrementos con funcionalidades de alto valor.
2. **Transparencia:** Todos los aspectos del proceso de desarrollo y resultados obtenidos deben ser transparentes y visibles para todas las partes interesadas.
3. **Inspección:** Se realizan revisiones periódicas del trabajo (al finalizar cada sprint) para detectar cualquier desviación o problema que haya podido surgir.

- **Roles:**

1. **Scrum Master:** Es el líder de equipo Scrum. Se encarga de eliminar los posibles obstáculos que puedan aparecer para el equipo y garantiza que el equipo de desarrollo pueda operar de manera ágil y eficiente, ayudándolos a adoptar y seguir los principios y prácticas de Scrum.
2. **Product Owner:** es responsable de definir la visión del producto, establecer las prioridades y tomar decisiones para que el producto obtenga el mayor valor posible. Además, es el responsable de administrar el *backlog* de producto.
3. **Equipo de desarrollo:** Es el grupo de profesionales encargados de desarrollar el incremento en cada sprint. El equipo es autoorganizado y sus integrantes colaboran activamente entre sí.

- **Artefactos:**

1. **Product Backlog:** Es la lista de historias de usuario mantanida por el product owner. Esta lista cambia y se desarrolla a medida que van pasando los sprints.
Una **historia de usuario** Es una técnica utilizada para capturar requisitos del cliente desde su perspectiva. Esta compuesta por título descriptivo y una breve descripción de como debería ser la funcionalidad deseada.
2. **Incremento:** Parte del producto que se obtiene al terminar un Sprint. Este se caracteriza por funcional y potencialmente entregable.
3. **Sprint Backlog:** Es una lista de historias de usuarios provenientes del product backlog, las cuales serán las que comprenderán el incremento del sprint.

4. **Sprint:** Es el período de tiempo fijo y corto durante el cual se produce el incremento de producto. La duración típica varía entre 2 y 4 semanas.

- **Reuniones y Eventos:**

1. **Backlog Grooming:** El equipo de desarrollo junto con el Product Owner revisan y ajustan las historias de usuario del backlog para garantizar que estén claras, detalladas y priorizadas para que estén listas para ser implementadas.
2. **Sprint Planning:** Es la reunión a comienzo de cada sprint donde se seleccionan las diferentes historias de usuario que serán desarrolladas durante el sprint e incorporadas en el incremento.
3. **Sprint Review:** Reunión realizada a final de cada sprint donde se presenta el incremento obtenido y se recopila feedback. Generalmente el Product Owner prueba el incremento y comprueba que corresponda con las historias de usuario seleccionadas en el sprint backlog.
4. **Sprint Retrospective:** Reunión realizada al final de cada sprint dirigida por el Scrum Master donde el equipo de desarrollo reflexiona sobre cómo se realizó el sprint anterior y se proporcionan alternativas y posibles mejoras para los sprints siguientes.

3.2 Kanban

Kanban [21] es un marco de trabajo utilizado para la gestión del flujo de trabajo en diferentes tipos de proyectos. Se basa principalmente en la representación visual del flujo de trabajo realizado por el equipo. Kanban ayuda para que la gestión del esfuerzo de equipo durante el sprint sea lo más eficiente posible y evitar bloqueos de los diferentes miembros del equipo.

Una de las grandes ventajas de Kanban es que da una visión completa y clara del estado del proyecto en cada momento, por lo que es más fácil detectar problemas y cualquier tipo de retraso en el desarrollo de las tareas.

En Kanban se destacan tres elementos clave:

- **Tablero:** Es quien muestra el flujo del trabajo del equipo y el estado de las tareas en cierto momento. Está dividido en columnas, donde cada una representa una etapa del desarrollo.
- **Tarjeta:** Representa una tarea, la cual va avanzando por las columnas del tablero a medida que se va desarrollando.

- **Work in Progress (WIP):** Límite de número de tarjetas que puede haber en una columna del tablero, de esta manera se evita sobrecargar a los miembros del equipo de desarrollo.

3.3 Adaptaciones al proyecto

Como se mencionó previamente en este capítulo, se ha utilizado una adaptación de la metodología Scrum dada por el contexto de este proyecto. La duración de los sprints serán de dos semanas cada uno (como recomienda originalmente la metodología), de esta manera se da tiempo a desarrollar suficientes historias de usuario sin tener que esperar mucho tiempo para obtener feedback por parte del product owner.

En cuanto a los roles de Scrum, se asignaron de la siguiente forma:

- **Product Owner:** Será desempeñado por los dueños de la explotación ganadera "Santa Ana".
- **Scrum Master:** Este rol es asumido por el estudiante.
- **Equipo de desarrollo:** Será asumido únicamente por el estudiante, debido a que es un proyecto individual.

El proceso de Scrum comenzó con una reunión inicial con los dueños de la explotación para realizar un análisis profundo del dominio y así obtener la mayor cantidad de historias de usuario posibles. Estas historias de usuarios alimentan el Product Backlog, el cual pasó por el proceso de Backlog Grooming para que las historias de usuario estuviesen priorizadas, correctamente descritas y tuviesen una estimación inicial, la cual seguramente sufrirá modificaciones más adelante en el proceso cuando se realicen los sprint planning.

Una vez creado el Product Backlog con las historias de usuarios obtenidas, se procede a celebrar una reunión de sprint planning donde se obtendrá el Sprint Backlog del sprint. Para obtener el Sprint Backlog se procede a coger las historias de usuarios más prioritarias del product backlog y subdividirlas en tareas más pequeñas, las cuales serán estimadas en horas. Una vez obtenido el Sprint Backlog, se procede a dar inicio al sprint de 2 semanas. Cabe destacar que debido a la naturaleza del equipo de trabajo, no se realizarán Daily meetings.

Una vez obtenido el sprint backlog, se procede a crear el tablero Kanban del sprint. Este tablero consta de 5 columnas:

1. **To do:** Es el sprint backlog, las tareas que están pendientes de hacer.
2. **Design:** Se analiza y se diseña la solución que se desarrollará para dicha tarea.
3. **Doing:** Se está desarrollando la tarea.

4. **Testing:** Se procede a realizar pruebas a lo desarrollado. Pueden ser pruebas automatizadas o no.

5. **Done:** La tarea ya está hecha.

Para este tablero se definió un límite de trabajo en progreso (WIP) de 1 en la columna **Doing**. Esta decisión vino dada por la naturaleza del equipo de desarrollo, ya que al ser de una sola persona es muy propenso a ocurrir sobrecargas de trabajo. Con evitar estas sobrecargas podemos mantener la calidad del trabajo y así logramos evitar ciertos errores que más adelante pueden causar mayores perdidas de tiempo.

Cabe destacar que para todos los tableros Kanban se debe definir lo que se entiende por **Hecho**, es decir, que es lo que se considera que está terminado y puede pasar a la última columna, que en nuestro caso sería Done. Para este tablero, una tarea puede pasar al estado Done si todas las tareas relacionadas a una historia de usuario han llegado a la columna testing, es decir, se han realizado las pruebas necesarias a los diferentes tareas que componen la historia de usuario.

Una vez terminado todas las tareas y obtenido el incremento se procede a realizar el sprint review, donde el product owner revisan y prueban el incremento para validarla y darnos el feedback necesario para poder seguir en el siguiente sprint. En esta reunión también se lleva a cabo una revisión del Product Backlog, con el motivo de validar las historias de usuario con el product owner y ver si este quiere hacer ciertas modificaciones a estas.

Debido a que el equipo de desarrollo y el scrum master son la misma persona, no se realizan las reuniones de Sprint Retrospective.

Una vez acabado el sprint, se procede a actualizar el Product Backlog con los datos obtenidos en el sprint review y se repite el ciclo. Este ciclo se repetirá hasta que se vacie el Product Backlog y el cliente de por válido el producto final.

En la figura 3.1 se representa gráficamente como se aplica la metodología de Scrum en el proyecto.

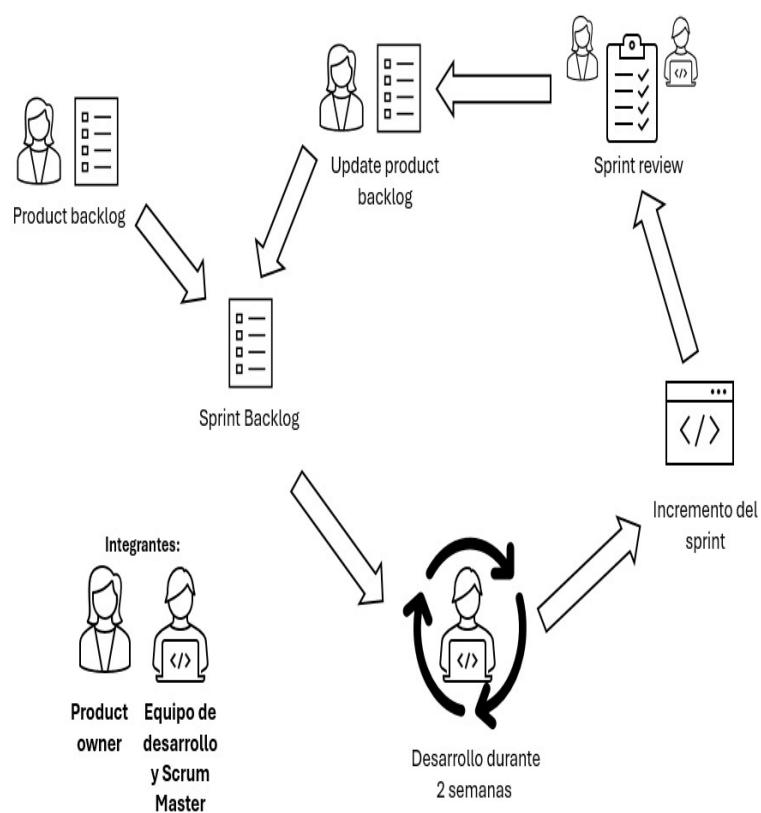


Figura 3.1: Scrum aplicado en el proyecto

Capítulo 4

Análisis

En este capítulo nos centraremos en el análisis del proyecto, en el cual se determina tanto los requisitos funcionales como los no funcionales y las Historias de usuarios. Una vez obtenidas los requisitos se procederá a realizar estimaciones de tiempo y coste sobre el proyecto, de esta manera se le comunica al cliente el alcance de la aplicación y que impacto económico conllevaría.

4.1 Análisis de requisitos

Para la obtención de los requisitos funcionales y no funcionales se realizaron reuniones con los dueños de la explotación "Santa Ana". Estas reuniones consistieron principalmente en preguntarles cuales eran los principales características que deseaban tener en la aplicación y que tareas y actividades eran las más criticadas en su día a día en la explotación.

4.1.1 Requisitos funcionales

Los requisitos funcionales describen el comportamiento y características que el sistema debe de tener para cumplir con las expectativas del cliente y generarle valor. Se centran en los que el sistema debe hacer desde el punto de vista del usuario.

- **FR-01 Autentificación de usuarios:** La aplicación debe de permitir que los usuarios puedan iniciar sesión en el sistema de forma segura.
- **FR-02 Cerrar sesión:** El sistema debe de permitir cerrar la sesión a todos los usuarios que se encuentren logueados en la aplicación.
- **FR-03 Gestión de tareas:** El sistema debe de permitir crear, modificar y eliminar tareas. Estas tareas deben de poder asignables a los empleados de una explotación.

- **FR-04 Gestión de empleados:** La aplicación debe de permitir crear empleados relacionados a la explotación. Los empleados pueden ser modificados, eliminados y bloqueados por cierto tiempo.
- **FR-05 Gestión del ganado:** El sistema debe permitir el registro de nuevas cabezas de ganado, modificar sus datos, eliminarlos y realizar un listado con filtro. Además, para cada animal se podrá visualizar sus datos específicos y gráficos relevantes.
- **FR-06 Producción lechera:** El sistema debe de permitir crear registros de ordeño de los diferentes animales de producción lechera. Estos registros de ordeño pueden ser listados, filtrados, editables y eliminables.
- **FR-07 Producción cárnea:** La aplicación debe permitir crear registros de peso de todos los animales de la explotación, diferenciando si el pesaje realizado esta destinado a la producción de carne o si es por otros motivos (enfermedad, rutina, etc). Al igual que los registros de ordeño, estos pesajes pueden ser listados y filtrados, editables y eliminables.
- **FR-08 Compra de alimentos:** El sistema debe de permitir registrar compras de alimentos que serán guardados en el stock. Estas compras pueden ser listadas, editables y eliminadas. Además, se podrán visualizar gráficamente ha sido el consumo de ese alimento a medida que se va consumiendo.
- **FR-09 Consumo de alimentos:** En el sistema se podrán registrar los diferentes consumos de alimentos del stock hecho por los diferentes animales. Estos registros de consumo serán listables, editables y eliminables.
- **FR-10 Dashboard:** En la aplicación se deben de poder visualizar gráficos con información relevante del estado general de la explotación.

4.1.2 Requisitos no funcionales

Los requisitos no funcionales son las características y cualidades que debe de tener el sistema para cumplir con los estándares de calidad, seguridad, rendimiento y usabilidad del cliente, es decir, estos se centran en como debe ser el sistema en vez de lo que debe de hacer.

- **NFR-01 Seguridad:** El sistema debe de establecer controles y métodos de seguridad para proteger la información de los usuarios de la aplicación y para evitar cualquier tipo de acceso no autorizado.
- **NFR-02 Mantenibilidad:** El sistema debe de estar hecho con una estructura que facilite el mantenimiento y lanzamiento de actualizaciones.

- **NFR-03 Rendimiento:** La aplicación debe garantizar un tiempo de respuesta adecuado y ser lo más eficiente posible con el consumo de memoria y procesador.
- **NFR-04 Usabilidad:** La aplicación debe de contar con una interfaz de usuario intuitiva y fácil de usar para que pueda ser usada por todo tipo de usuarios, independientemente de su nivel de educación.
- **NFR-05 Compatibilidad:** El sistema debe de poder usarse en los navegadores más populares y en diferentes tipos de dispositivos (principalmente en ordenador, tablets y móviles).
- **NFR-06 Escalabilidad:** El sistema debe de soportar un aumento en la cantidad de usuarios sin perder cualidades en el rendimiento.

4.2 Historias de usuario

Antes de describir las diferentes historias de usuarios que se encuentran en la aplicación, es preciso mencionar los principales actores que harán de estas funcionalidades. Los actores representan las diversas interacciones que ocurren en la aplicación.

- **Usuario:** Representa a cualquier usuario que utilice la aplicación
- **Administrador:** Es el Jefe/Dueño de la explotación, por lo que podrá realizar la mayor cantidad de funcionalidades
- **Empleado:** Representa un empleado de la explotación ganadera específica.
- **Master:** Actor con la capacidad de realizar todas las funciones disponibles.

Para la descripción de las historias de usuario se seguirá el siguiente formato: "Como {1}, quiero {2} para {3}", donde **1** será el actor, **2** es la acción a realizar y **3** el objetivo que se quiere lograr.

1. **US-01 Iniciar sesión:** "Como usuario, quiero poder iniciar sesión en la aplicación de forma segura para hacer uso de esta".
2. **US-02 Cerrar sesión:** "Como usuario, quiero poder cerrar sesión en la aplicación para ayudar a la privacidad de mis datos".
3. **US-03 Actualizar datos de perfil:** "Como usuario, quiero poder actualizar los datos personales de mi perfil para cuando haya un cambio en mis datos personales".

4. **US-04 Crear empleado:** "Como administrador, quiero crear perfiles de empleados para que el staff de la explotación puedan hacer uso de la aplicación y así mantener un registro de su actividad".
5. **US-05 Listar empleados:** "Como administrador, quiero poder listar los empleados de la explotación con sus datos principales para poder gestionarlos".
6. **US-06 Suspender empleado:** "Como administrador, quiero tener la opción de suspender/bloquer un empleado para que cuando no esté disponible temporalmente (vacaciones, baja, etc) no se le pueda asignar ninguna tarea ni pueda acceder a la aplicación".
7. **US-07 Eliminar empleado:** "Como administrador, quiero eliminar perfiles de empleados para que no vuelvan a acceder a la aplicación cuando estos dején de ser parte de la explotación."
8. **US-08 Crear tarea:** "Como administrador, quiero poder crear tareas para poder asignárselas a los empleados".
9. **US-09 Actualizar datos de tarea:** "Como administrador, quiero poder editar y actualizar los datos de una tarea para reflejar cambios que puedan ocurrir en esta".
10. **US-10 Eliminar una tarea:** "Como administrador, quiero eliminar una tarea para cuando esta ya no sea relevante o no se vaya a realizar".
11. **US-11 Cambiar estado de tarea:** "Como empleado, quiero poder cambiar el estado de una tarea a -REALIZADO- cuando esta ya esté realizada"
12. **US-12 Listar tareas:** "Como administrador o empleado, quiero poder listar las tareas para ver todas en la que esté asociado y ver los datos de las tareas". (el administrador podrá ver todas las tareas de la explotación)
13. **US-13 Registrar compra de alimentos:** "Como administrador, quiero registrar compras de alimento para mantener un stock y tener un histórico de compras".
14. **US-14 Editar datos de compra:** "Como administrador, quiero poder editar los datos de una compra de alimentos para corregir errores que hayan ocurrido al llenar el formulario de creación".
15. **US-15 Eliminar compra de alimentos:** "Como administrador, quiero poder eliminar una compra de alimento para reflejar en la aplicación si dicha compra no se realizó".
16. **US-16 Visualizar datos de compra:** "Como administrador, quiero poder visualizar los datos de una compra de alimento para acceder a la información detallada de este".

17. **US-17 Listar compras de alimento:** "Como administrador, quiero listar las compras de alimentos que se hayan realizado en la explotación para poder filtrarlas y realizar otras acciones como sobre estas".
18. **US-18 Ver gráfica de consumo de una compra de alimento:** "Como administrador, quiero poder ver gráficamente el ritmo de consumo que ha sufrido un lote de comida para tomar una decisión sobre futuras compras de alimentos."
19. **US-19 Registrar nuevo ganado:** "Como administrador, quiero registrar nuevo ganado en la aplicación para digitalizar sus datos de producción".
20. **US-20 Editar datos de ganado:** "Como administrador, quiero poder editar los datos de un animal para reflejar cambiar cambio que pueda haber ocurrido en su estado."
21. **US-21 Cambiar estado de ganado:** "Como administrador, quiero cambiar el estado de un animal para reflejar en la aplicación cuando este ya no se encuentra disponible (venta, fallecimiento, etc)".
22. **US-22 Visualizar datos del ganado:** "Como administrador o empleado, quiero poder visualizar los datos específicos de una animal para obtener información detallada sobre este".
23. **US-23 Listar ganado de la explotación:** "Como administrado o empleado, quiero listar y filtrar el ganado de la explotación para ver el histórico de ganado y poder realizar otras acciones sobre estos".
24. **US-24 Ver gráficos sobre un animal:** "Como administrador o empleado, quiero poder ver gráficamente gráficos sobre datos de producción y consumo de alimento de un animal para ver su evolución de peso y producción en relación al alimento consumido".
25. **US-25 Registrar consumo de alimento:** "Como administrador o empleado, quiero registrar consumo de alimento de uno o varios animales para llevar un histórico de su ritmo de alimentación y llevar una cuenta de cuanto alimento queda en stock".
26. **US-26 Eliminar consumo de alimento:** "Como administrador o empleado, quiero poder eliminar un registro de consumo de alimento para corregir errores a la hora de llenar el formulario de registro de consumo o si dicho consumo no se llevó a cabo realmente".
27. **US-27 Listar registros de consumo de alimento:** "Como administrador o empleado, quiero listar y filtrar los registros de consumo de alimento para ver el histórico de consumos y ver los datos de estos".

28. **US-28 Registrar pesaje:** "Como administrador o empleado, quiero registrar pesajes de animales para llevar un histórico del peso de cada uno y ver su evolución a lo largo del tiempo".
29. **US-29 Editar datos de un pesaje:** "Como administrador o empleado, quiero poder editar los datos de un pesaje para corregir cualquier error que haya ocurrido durante el pesaje o la cumplimentación del formulario de pesaje".
30. **US-30 Eliminar registro de pesaje:** "Como administrador o empleado, quiero eliminar un registro de pesaje para reflejar que ese pesaje ya no es relevante o no se llevó a cabo".
31. **US-31 Visualizar datos de pesaje:** "Como administrador o empleado, quiero poder visualizar los datos específicos de un pesaje para obtener información detallada sobre este".
32. **US-32 Listar registros de pesajes:** "Como administrador o empleado, quiero poder listar y filtrar los registros de pesajes para ver el histórico de pesajes y poder realizar acciones sobre estos".
33. **US-33 Registrar ordeño:** "Como administrador o empleado, quiero registrar ordeños de animales para llevar un histórico del nivel de producción de cada animal y ver su evolución a lo largo del tiempo".
34. **US-34 Editar datos de ordeño:** "Como administrador o empleado, quiero poder editar los datos de un ordeño para corregir cualquier error que haya ocurrido durante el ordeño o la cumplimentación del formulario de registro de ordeño".
35. **US-35 Eliminar registro de ordeño:** "Como administrador o empleado, quiero poder eliminar un registro de ordeño para reflejar que ese ordeño ya no es relevante o no se llevó a cabo".
36. **US-36 Visulizar datos de ordeño:** "Como administrador o empleado, quiero poder visualizar los datos específicos de un ordeño para obtener información detallada sobre este".
37. **US-37 Listar resgistros de ordeño:** "Como administrador o empleado, quiero listar y filtrar los registros de ordeños para ver el histórico de ordeños y poder realizar acciones sobre estos".
38. **US-38 Dashboard de gráficos:** "Como administrador o empleado, quiero poder ver una página (dashboard) de gráficos de datos generales de la explotación para ver el estado actual de la explotación y tomar decisiones en base a esos datos mostrados".

4.3 Estimación Inicial

Se llevó a cabo una estimación inicial del proyecto para comunicarle al cliente el tiempo y coste aproximado. Además, al tener una estimación de tiempo inicial esto facilita la organización de las tareas de desarrollo. Al utilizar un metodología como Scrum, estas estimaciones pueden ir variando o sufrir cambios a medida que avanza el proyecto.

4.3.1 Estimación en tiempo

Para la estimación de tiempo, se realizó con el método Planning Poker [22]. Este método usa como valor de tiempo los [Punto historia \(PH\)](#), pero para reducir la complejidad de la estimación se establece que un PH corresponde con una hora de trabajo.

En el Planning Poker se usó la sucesión de Fibonacci para las tarjetas, ya que esta sucesión representa bien la incertidumbre que uno puede tener al estimar una tarea. Como el equipo de desarrollo solo está compuesto por el estudiante, este fue el único que participó en el planning poker, aún sabiendo que el planning poker es más efectivo cuando se hace con equipos de dos o más personas.

Cuadro 4.1: Estimación inicial de las historias de usuario

Historias de usuario	Horas
US-01 Iniciar sesión	21
US-02 Cerrar sesión	3
US-03 Actualizar datos de perfil	13
US-04 Crear empleado	21
US-05 Listar empleados	8
US-06 Suspender empleado	2
US-07 Eliminar empleado	3
US-08 Crear tarea	21
US-09 Actualizar datos de tarea	8
US-10 Eliminar una tarea	5

..... (continúa na páxina seguinte)

Cuadro 4.1 – (*vén da páxina anterior*)

Historias de usuario	Horas
US-11 Cambiar estado de tarea	2
US-12 Listar tareas	13
US-13 Registrar compra de alimentos	21
US-14 Editar datos de compra	8
US-15 Eliminar compra de alimentos	2
US-16 Visualizar datos de compra	5
US-17 Listar compras de alimento	13
US-18 Ver gráfica de consumo de una compra de alimento	5
US-19 Registrar nuevo ganado	21
US-20 Editar datos de ganado	8
US-21 Cambiar estado de ganado	2
US-22 Visualizar datos del ganado	5
US-23 Listar ganado de la explotación	13
US-24 Ver gráficos sobre un animal	34
US-25 Registrar consumo de alimento	21
US-26 Eliminar consumo de alimento	2
US-27 Listar registros de consumo de alimento	13
US-28 Registrar pesaje	8
US-29 Editar datos de un pesaje	5
US-30 Eliminar registro de pesaje	2
US-31 Visualizar datos de pesaje	3
US-32 Listar registros de pesajes	13

..... (*continúa na páxina seguinte*)

Cuadro 4.1 – (vén da páxina anterior)

Historias de usuario	Horas
US-33 Registrar ordeño	8
US-34 Editar datos de ordeño	5
US-35 Eliminar registro de ordeño	2
US-36 Visualizar datos de ordeño	3
US-37 Listar registros de ordeño	13
US-38 Dashboard de gráficos	34
TOTAL	389

4.3.2 Estimación del coste

Ya con una estimación inicial del tiempo que llevaría desarrollar la aplicación, se puede proceder a realizar una estimación del coste que supondría desarrollar la aplicación. Con un tiempo total estimado de 355 horas y las 30 horas de análisis previo, nos quedaría un suma de 375 horas de dedicación.

Debido a la naturaleza del equipo de trabajo, solo hay una persona que se encarga del desarrollo y análisis del sistema, lo que conlleva a que esta persona recaiga esa carfa de 375 horas de esfuerzo. Teniendo en cuenta que el salario promedio de un ingeniero informático en España en 2024 son 27.000€ al año [23], nos queda un precio por hora a 13,85€. Además del precio del desarrollador, hay que tener en cuenta el coste de los recursos utilizados, por lo que a cada hora de desarrollo se le añadirán 5€ que representarán estos recursos (ordenador, internet, electricidad, etc). Teniendo en cuenta todo esto, el coste estimado de la aplicación es de **6968,75€**.

$$\text{Coste total} = \left(355h \text{ ordenador} \cdot 5 \frac{\text{€}}{h} \right) + (375h \cdot 13.85\text{€}) = 6968.75\text{€}$$

Figura 4.1: Fórmula del coste estimado

Capítulo 5

Diseño

En este capítulo se tratarán varios aspectos del diseño que se ha realizado sobre el sistema. Nos centraremos principalmente en los diseños de la base de datos, la arquitectura general del sistema y su estructura (directorios y ficheros relevantes).

5.1 Arquitectura

Para implementar el sistema se decidió seguir la arquitectura **Cliente-Servidor** en capas; donde el servidor es quién realiza las tareas y se comunica con la base de datos, mientras que los clientes simplemente se encarga de mostrar la interfaz de usuario para que este pueda acceder a las diferentes funcionalidades.

En la parte de cliente se usa la arquitectura [Single Page Application \(SPA\)](#). Las SPA nos permiten una navegación más fluida y rápida ya que esta trabaja sobre un solo documento HTML y va actualizando de forma selectiva partes de este documento en vez de traer páginas completas del servidor. Las SPA nos proporcionan una experiencia de usuario más fluida y rápida pero puede ser más complicado la gestión del estado de la aplicación en el lado del cliente.

La parte de cliente se divide en dos capas:

- **Interfaz de usuario:** Es la capa que se encarga del control del DOM y la interacción con el usuario.
- **Acceso a servicios:** Capa que se encarga de la comunicación (envío y recepción de datos) con el servidor usando la API de este.

En cuanto a la parte del servidor, también se siguió una arquitectura en capas ya que esta facilita el desarrollo y la mantenibilidad al mantener una separación de responsabilidades. Las capas que componen el servidor son:

- **Servicios:** Capa encargada de recibir y procesar las solicitudes que son enviadas desde los clientes usando el API REST.
- **Modelo:** esta compuesta de dos subcapas:
 - **Lógica de negocio:** Capa que contiene la lógica de las funcionalidades y la encargada de mantener la coordinación de los flujos de trabajo de la aplicación.
 - **Acceso a datos:** Es la encargada de comunicarse con la base de datos para escribir y leer los datos de la aplicación.

En la figura 5.1 se ve representada gráficamente la arquitectura del sistema.

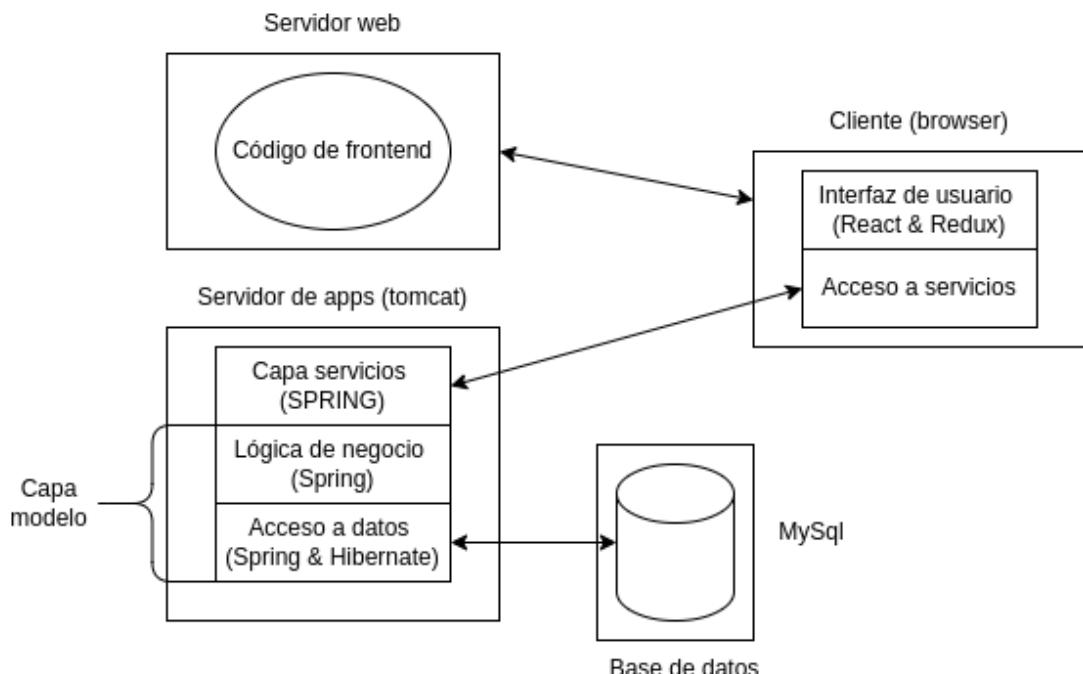


Figura 5.1: Arquitectura del sistema

5.2 Estructura

La estructura del sistema se compone de los siguientes directorios:

- **Principales:**
 - **backend:** contiene los archivos y assets relacionados con la parte del servidor.
 - **frontend:** contiene los archivos y assets relacionados con la parte del cliente.

- **subdirectorios más importantes de backend/:**
 - **main/java/es/easyfarm/backend/model:** es el directorio que contiene el código de la capa modelo (entidades y lógica de negocio).
 - **main/java/es/easyfarm/backend/rest:** directorio que contiene el código de la capa de servicios (controladores y dtos).
 - **sql/:** Contiene los scripts de creación de la base de datos de producción y de test.
 - **test/:** Contiene los archivos con el código de las pruebas automatizadas.
- **subdirectorios más importantes de frontend/:**
 - **modules/:** contiene los diferentes componentes que componen la SPA.
 - **store/:** contiene los archivos relacionados con la configuración de la gestión de estado de la SPA.
 - **backend/:** contiene los archivos relacionados con las peticiones al servidor (capa de acceso a servicios).

5.3 Base de datos

Para dar una vista más completa sobre la base de datos se realizarán dos diagramas que se complementan entre ellos: [Unified Modeling Language \(UML\)](#) y el esquema relacional. Estos diagramas se complementan bien entre ellos ya que el UML se centra más en las entidades y como se relacionan entre ellas, mientras que el esquema relacional está dirigido mayormente para representar la estructura interna de las tablas y sus atributos.

5.3.1 UML

El diagrama UML (figura 5.3) ofrece una representación clara y sencilla de las diferentes entidades del sistema y las relaciones entre ellas. Cada entidad tiene un nombre y es definida por sus atributos y las relaciones entre ellas se ilustran mediante líneas, donde cada línea represente un tipo de relación.

1. **Entidades:** las entidades presentes en el diagrama UML son:

- **Farm:** representa la explotación donde se desarrolla la actividad.
- **User:** representa a los usuarios que hacen uso de la aplicación.
- **Issue:** define una tarea creada.
- **Animal:** representa a los animales asociados a la explotación.

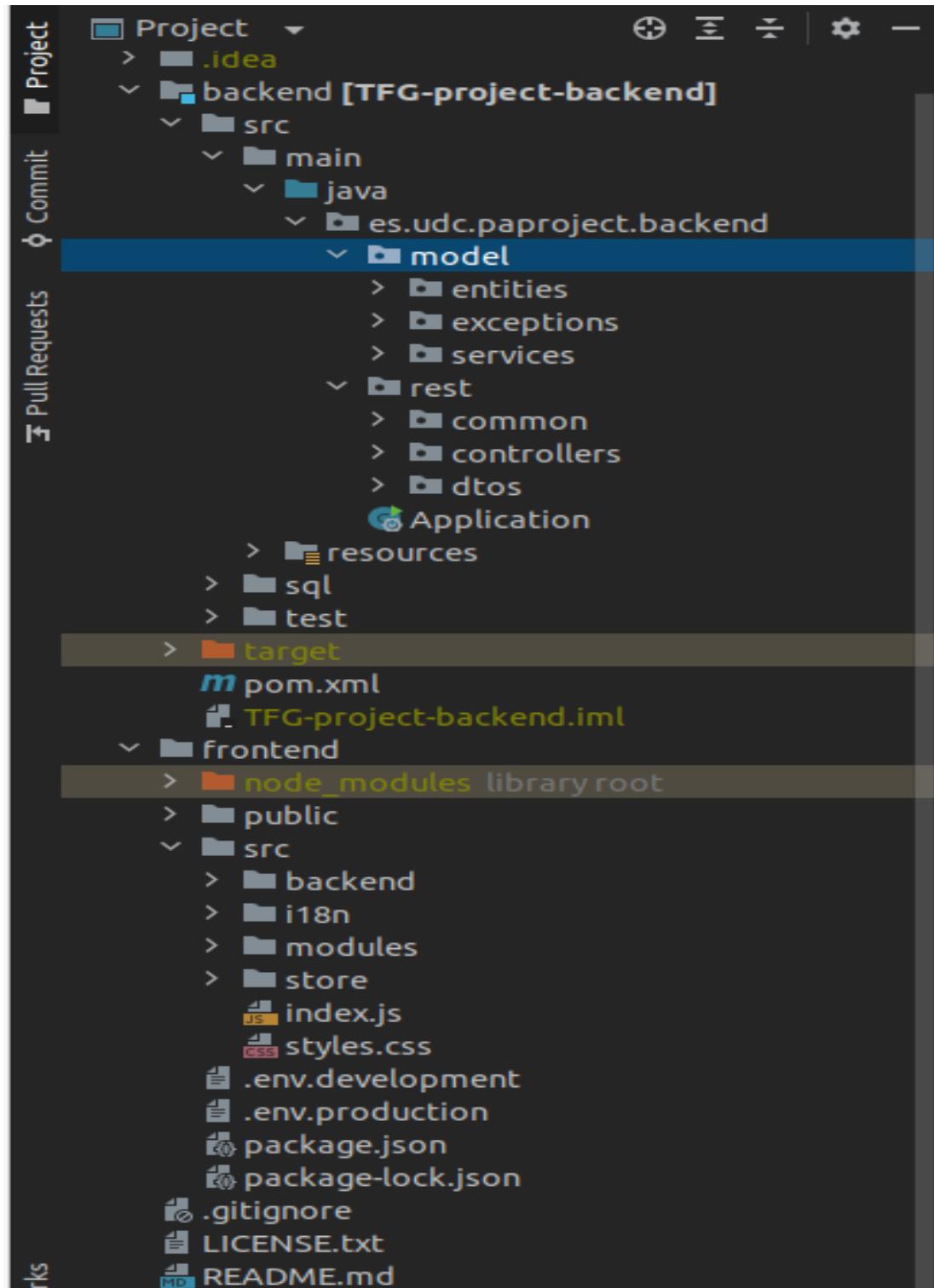


Figura 5.2: Estructura del sistema

- **FoodPurchase:** define un registro de compra de alimentos.
 - **FoodConsumption:** representa una consumición de alimento.
 - **Weighing:** representa un registro de pesaje.
 - **Milking:** representa un registro de ordeño.
2. **Entidades:** Las relaciones usadas son de tipo **Asociación**, las cuales indican la relación entre dos entidades.
- **works at:** representa la relación de una explotación y los usuarios de esta (empleados y administrador).
 - **created by:** relación entre una tarea y cual usuario la creó.
 - **assigned to:** relación entre una tarea y el usuario a la cual fue asignada.
 - **belongs to:** representa la relación de una explotación y los animales que pertenecen a ella.
 - **made by:** realción entre la compra de alimento y quién la realizó.
 - **eats:** representa la relación de un consumo de alimento y cual animal fue quien la realizó.
 - **eats from:** representa la relación entre un consumo de alimento y de que lote de alimento fue consumido.
 - **weighed by:** relación entre un pesaje y quién lo realizó.
 - **weighed:** relación entre pesaje y que animal fue pesado.
 - **milked by:** relación entre ordeño y quien lo realizó.
 - **milked:** relación entre ordeño y que animal fue ordeñado.

5.3.2 Esquema relacional

Este esquema (figura 5.4) representa de forma clara la estructura de la base de datos, mostrando cada tabla con todos sus atributos. En la figura, cada tabla representa una entidad de la base de datos, las cuales todas tendrán una clave primaria artificial (ID) para facilitar el desarrollo y dar una mayor claridad al esquema. También se muestran todas las claves foraneas, estas son representadas con flechas que apuntan al atributo que con que se relacionan.

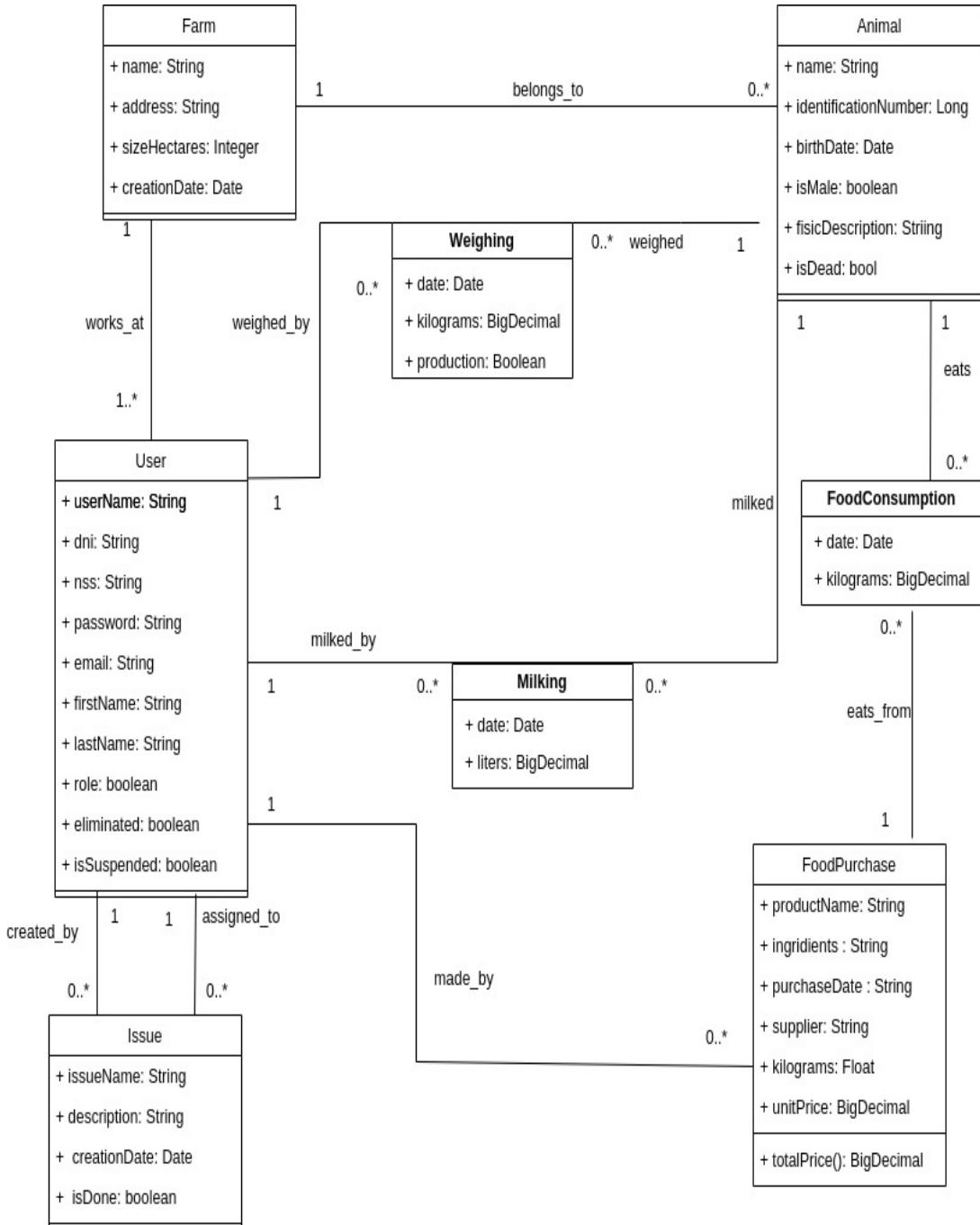


Figura 5.3: Diagrama UML de la base de datos

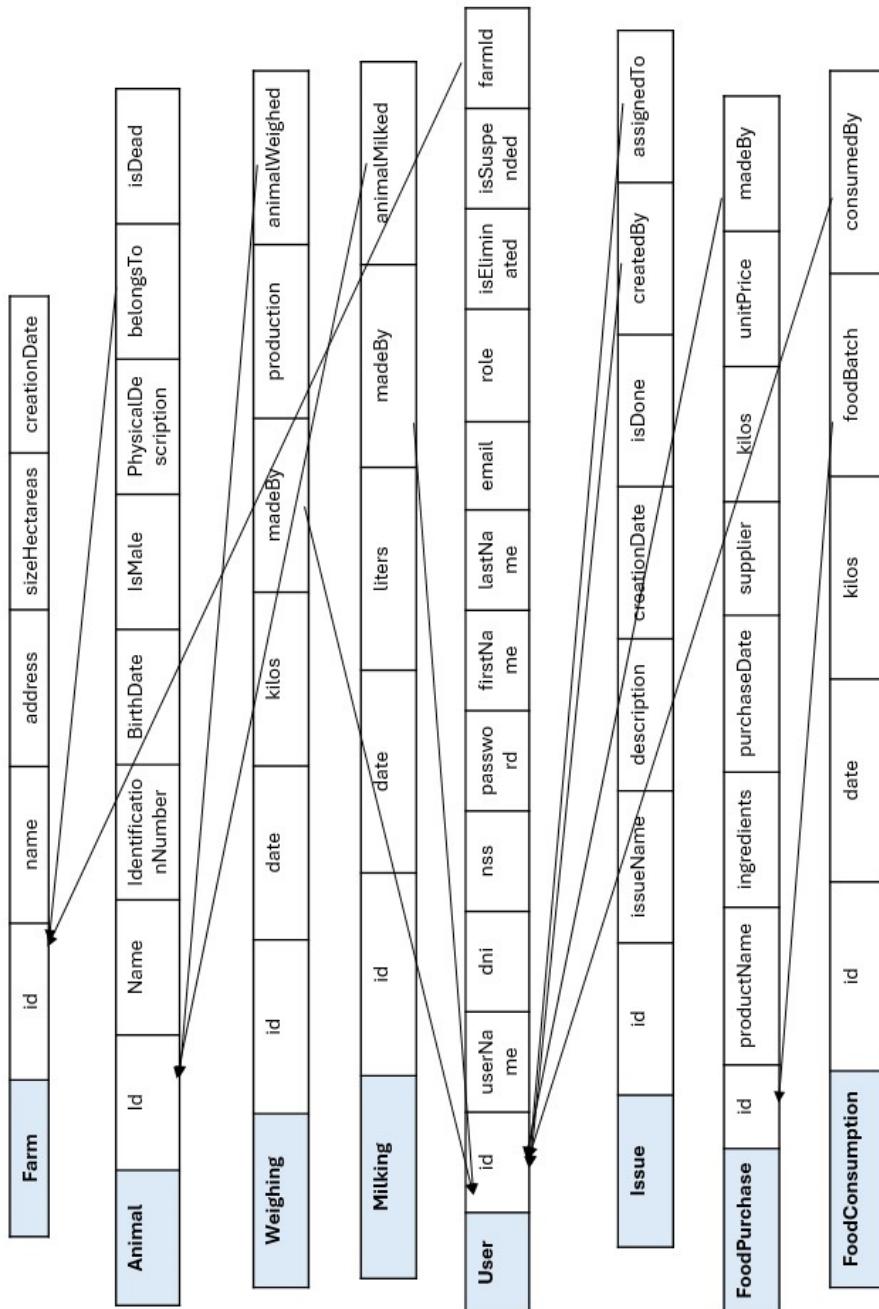


Figura 5.4: Esquema relacional de la base de datos

Capítulo 6

Desarrollo e implementación

En este capítulo mostraremos como fue el proceso de la codificación de sistema dentro del marco de desarrollo establecido. Se describirá como se usó las metodologías Scrum y Kanban para desarrollar la aplicación y lograr un trabajo eficiente y ordenado.

6.1 Desarrollo

En esta sección se detalla cada una de las iteraciones de Scrum realizadas hasta la culminación de proyecto.

6.1.1 Primera iteración

En la primera iteración iniciamos con el Product Backlog inicial, el cual fue obtenido con la primera reunión con el cliente.

Sprint Planning

En la reunión de sprint planning se obtiene el Sprint backlog a partir del product backlog. El sprint backlog contiene las historias de usuario a desarrollar ordenadas en función a su prioridad. Las historias de usuario serán divididas en subtareas mas pequeñas para facilitar la organización. Desde la figura 6.1 a la figura 6.7 se muestran las divisiones en subtareas las historias de usuarios del sprint backlog.

Una vez obtenido el sprint backlog, se procedió a la puesta en marcha del sprint de 2 semanas. Durante estas dos semanas se usó la herramienta **TRELLO** para implementar el tablero KanBan con las tareas del sprint backlog. Esto nos permite evaluar el estado del sprint en con tan solo una mirada al tablero y así sabes si hay problemas de rendimiento o bloqueos con las tareas. En la siguiente figura (6.1) se puede apreciar el estado del sprint al comenzarlo.

Historia de usuario	Tareas	Horas
US-01 Iniciar sesión	Configuración	10
	Crear BD	1
	Desarrollo backend	3
	Desarrollo frontend	6
	Tests	1

Cuadro 6.1: US-01 Iniciar sesión

Historia de usuario	Tareas	Horas
US-02 Cerrar sesión	Desarrollo backend	1
	Desarrollo frontend	1
	Tests	1

Cuadro 6.2: US-02 Cerrar sesión

Historia de usuario	Tareas	Horas
US-03 Actualizar datos del perfil	Desarrollo backend	5
	Desarrollo frontend	6
	Tests	2

Cuadro 6.3: US-03 Actualizar datos del perfil

Historia de usuario	Tareas	Horas
US-04 Crear empleado	Modifcar base de datos	1
	Desarrollo backend	9
	Desarrollo frontend	10
	Tests	1

Cuadro 6.4: US-04 Crear empleado

Historia de usuario	Tareas	Horas
US-05 Listar empleado	desarrollo backend	3
	Desarrollo frontend	4
	Tests	1

Cuadro 6.5: US-05 Listar empleado

Historia de usuario	Tareas	Horas
US-06 Suspender empleado	desarrollo backend	0.75
	Desarrollo frontend	0.75
	Tests	0.5

Cuadro 6.6: US-06 Suspender empleado

Historia de usuario	Tareas	Horas
US-07 Eliminar empleado	desarrollo backend	1.75
	Desarrollo frontend	0.75
	Tests	0.5

Cuadro 6.7: US-07 Eliminar empleado

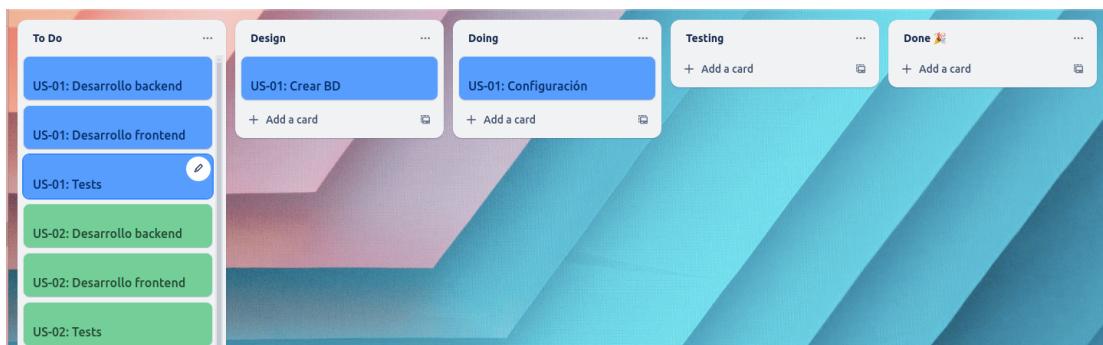


Figura 6.1: Tablero Kanban al comenzar la primera iteración.

Sprint Review

En la reunión de sprint review, se le presentó al Product Owner el incremento realizado y este prueba todas las funcionalidades y las da como aprobadas, por lo que en este sprint se completaron las 7 historias de usuario correctamente.

Además de la validación de las funcionalidades entregadas, se procedió a medir la desviación del tiempo de desarrollo del incremento con el estimado antes del comienzo de sprint. En la figura 6.8 se puede apreciar la diferencia entre las horas estimadas y el tiempo real de desarrollo.

Cuadro 6.8: Comparación del tiempo estimado y el real de la primera iteración

Historias de usuario	Horas estimadas	Horas reales
US-01 Iniciar sesión	21	23
US-02 Cerrar sesión	3	3
US-03 Actualizar datos del perfil	13	12
US-04 Crear empleado	21	18
US-05 Listar empleado	8	8
US-06 Suspender empleado	2	2
US-07 Eliminar empleado	3	2
TOTAL	71	68

En las figuras 6.2 a la 6.4 se pueden apreciar algunas de las historias de usuarios más destacadas desarrolladas durante el sprint.

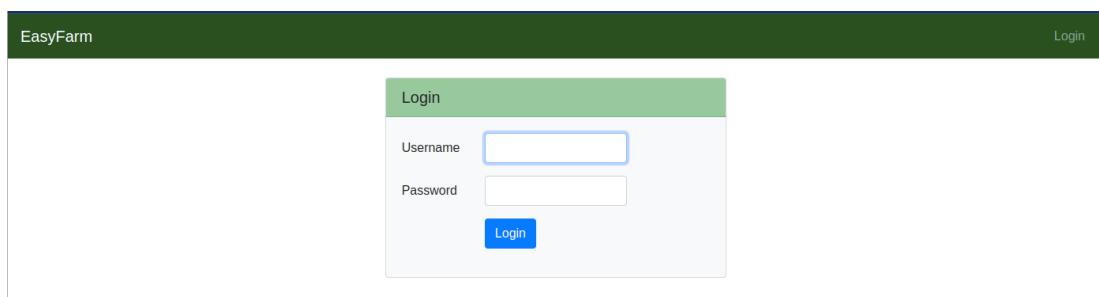


Figura 6.2: Login en la aplicación.

Username	First name	Last name	Email address	Suspend	Delete
employee1	Paulo	Gutiérrez	paulo.gutierrez@gmail.com		
employee2	Alexandra	Gutiérrez	ale.gutierrez@gmail.com		
test	test	test	test@gmail.com		

Figura 6.3: Lista de empleado.

Update profile

First name

Last name

Email address

Figura 6.4: Actualizar perfil.

En este sprint se trató uno de los requisitos no funcionales más importantes, la Seguridad y privacidad de la información de los usuarios. Uno de los requisitos para cumplir este requisito fué el tema de como guardar la contraseña del perfil de cada usuario. Las buenas prácticas indican que las contraseñas deben ser guardadas en la base de datos mediante un hash generado mediante un proceso de encriptación.

Para lograr este proceso de encriptación y desencriptación se usó la clase **BCryptPasswordEncoder** proporcionada por la librería de Springframework. Esta clase nos proporciona varios métodos que nos permiten encriptar las contraseñas de forma sencilla para ser guardadas en la base de datos. A continuación, se puede apreciar como el método **encode** (línea 8 6.1) encripta la contraseña de manera transparente para el desarrollador y nos ahorra horas de desarrollo.

```

1 @Override
2 public void signUp(User user) throws DuplicateInstanceException {
3
4     if (userDao.existsByUserName(user.getUserName())) {
5         throw new
6             DuplicateInstanceException("project.entities.user",
7                 user.getUserName());
8     }
9
10    user.setPassword(passwordEncoder.encode(user.getPassword()));
11    user.setRole(User.RoleType.EMPLOYEE);
12
13    userDao.save(user);
14 }
```

Listing 6.1: Función signUp del sistema.

Siguiendo con el tema de la seguridad, también se implementó la clase **SecurityConfig**. Esta clase es de suma importancia en la seguridad del sistema ya que establece las normas y políticas de acceso que los usuarios pueden tener a los diferentes endpoints del API REST que ofrece el backend. En este código 6.2 se puede apreciar como la función **configure** establece que cualquier usuario puede realizar la acción de login, pero solo los usuarios con rol "ADMIN" logueados pueden acceder a la funcionalidad de crear un empleado.

```

1 @Override
2 protected void configure(HttpSecurity http) throws Exception {
3
4     http.cors().and().csrf().disable()
5         .sessionManagement()
6
7         .sessionCreationPolicy(SessionCreationPolicy.STATELESS).and()
```

```

7      .addFilter(new JwtFilter(authenticationManager(),
8          jwtGenerator))
9      .authorizeRequests()
10     .antMatchers(HttpMethod.POST, "/users/signUp").permitAll()
11     .antMatchers(HttpMethod.POST, "/users/login").permitAll()
12     .antMatchers(HttpMethod.POST,
13         "/users/loginFromServiceToken").permitAll()
14     .antMatchers(HttpMethod.POST,
15         "/users/createEmployee").hasRole("ADMIN")
16     .anyRequest().denyAll();
}

```

Listing 6.2: Configuración de accesos a las funcionalidades.

6.1.2 Segunda iteración

Sprint Planning

En esta reunión entre los miembros de equipo de Scrum se procede a revisar el product backlog y elegir cuales son las siguientes funcionalidades que se desarrollarán en el incremento de este Sprint. Las historias de usuario elegidas y su división en sub tareas se pueden apreciar en las figuras 6.9 a la 6.17.

Historia de usuario	Tareas	Horas
US-08 Crear tarea	Modificar base de datos	1
	Desarrollo backend	9
	Desarrollo frontend	9
	Tests	2

Cuadro 6.9: US-08 Crear tarea

Historia de usuario	Tareas	Horas
US-09 Actualizar datos de tarea	Desarrollo backend	2
	Desarrollo frontend	5
	Tests	1

Cuadro 6.10: US-09 Actualizar datos de tarea

Historia de usuario	Tareas	Horas
US-10 Eliminar una tarea	Desarrollo backend	2
	Desarrollo frontend	2
	Tests	5

Cuadro 6.11: US-10 Eliminar una tarea

Historia de usuario	Tareas	Horas
US-11 Cambiar estado de tarea	Desarrollo backend	0.5
	Desarrollo frontend	1
	Tests	0.5

Cuadro 6.12: US-11 Cambiar estado de tarea

Historia de usuario	Tareas	Horas
US-12 Listar tareas	Desarrollo backend	5
	Desarrollo frontend	6
	Tests	2

Cuadro 6.13: US-12 Listar tareas

Historia de usuario	Tareas	Horas
US-13 Registrar compras de alimentos	Modificar base de datos	1
	Desarrollo backend	9
	Desarrollo frontend	9
	Tests	2

Cuadro 6.14: US-13 Registrar compras de alimentos

Historia de usuario	Tareas	Horas
US-14 Editar datos de compra	Desarrollo backend	2
	Desarrollo frontend	5
	Tests	1

Cuadro 6.15: US-14 Editar datos de compra

Historia de usuario	Tareas	Horas
US-15 Eliminar compra de alimentos	Desarrollo backend	0.5
	Desarrollo frontend	1
	Tests	0.5

Cuadro 6.16: US-15 Eliminar compra de alimentos

Historia de usuario	Tareas	Horas
US-16 Visualizar datos de compra	Desarrollo backend	1
	Desarrollo frontend	3
	Tests	1

Cuadro 6.17: US-16 Visualizar datos de compra

Una vez obtenido el sprint backlog, se procedió a actualizar el tablero Kanban en Trello con las tareas de este sprint [6.5](#).

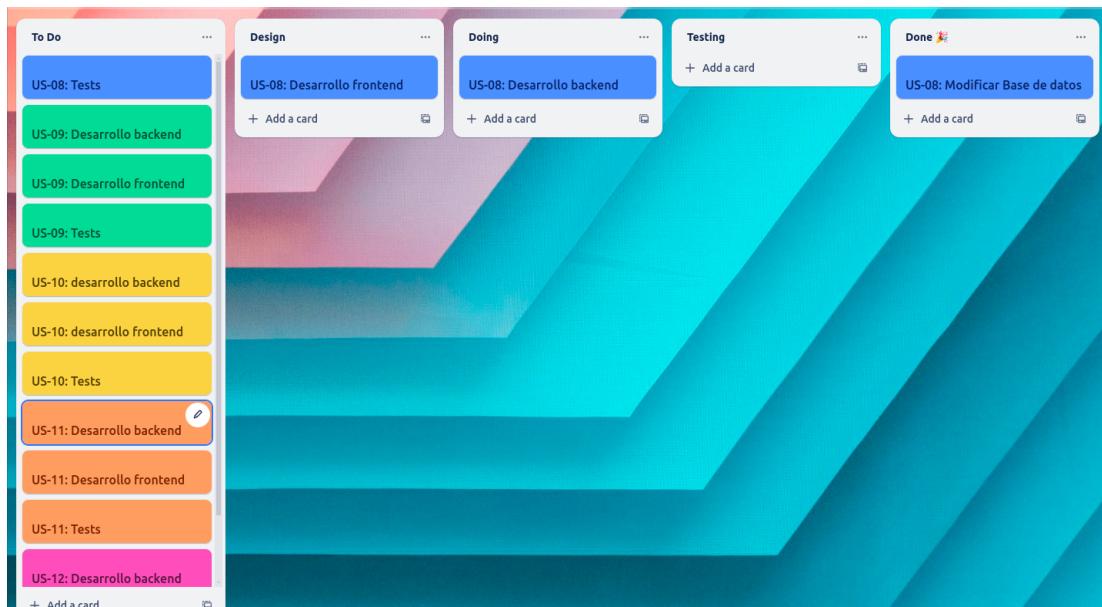


Figura 6.5: Tablero Kanban en la segunda iteración.

Sprint Review

Una vez acabado el sprint, se procedió a celebrar la reunión de sprint review con el product owner. En dicha reunión, el product owner probó todas las funcionalidades contenidas en el incremento y dando un feedback positivo sobre estas, por lo que en este sprint se lograron entregar 9 historias de usuario.

Al igual que la iteración anterior, una vez validado el incremento se procedió a medir la desviación del tiempo real de desarrollo del sprint con la estimación inicial. En la figura 6.18 se puede apreciar la diferencia entre el tiempo real y las estimaciones de las historias de usuarios desarrolladas.

Cuadro 6.18: Comparación del tiempo estimado y el real de la segunda iteración

Historias de usuario	Horas estimadas	Horas reales
US-08 Crear tarea	21	17
US-09 Actualizar datos de tarea	8	6
US-10 Eliminar una tarea	5	3
US-11 Cambiar estado de tarea	2	2

..... (continúa na páxina seguinte)

Cuadro 6.18 – (*vén da páxina anterior*)

Historias de usuario	Horas estimadas	Horas reales
US-12 Listar tareas	13	14
US-13 Registrar compras de alimentos	21	18
US-14 Editar datos de compra	8	5
US-15 Eliminar compra de alimentos	2	2
US-16 Visualizar datos de compra	5	6
TOTAL	85	73

En las figuras 6.6 a la 6.9 se pueden apreciar algunas de las historias de usuarios más destacadas del incremento desarrollado.

The screenshot shows a user interface for creating a new issue. At the top, there is a green header bar with the text "Create new issue". Below this, there are three input fields: "Issue name" (with a placeholder "Issue name"), "Description" (with a large text area for entering details), and "Assigned to" (with a dropdown menu showing "employee1"). At the bottom of the form is a blue "Create" button.

Figura 6.6: Crear tarea.

The screenshot shows a search interface at the top with fields for 'Issue name', 'State' (set to 'All'), and 'Creation date'. A 'Search' button is on the right. Below is a list of tasks. One task, 'test', is highlighted in green and shown in detail: 'issue test', 'Creation date: 5/21/2024', 'Assigned to: employee1', and a 'See details' button. At the bottom are 'Back' and 'Next' buttons.

Issue name:	State:	Creation date:
Issue name	All	

test Active

issue test
Creation date: 5/21/2024
Assigned to: employee1
[See details](#)

[Back](#) [Next](#)

Figura 6.7: Listar y filtrar tareas.

The screenshot shows a table titled 'My active issues' with columns for 'Issue name', 'Creation date', and 'Description'. One row is visible: 'test', '5/21/2024', 'issue test'. A 'Set as done' button is on the right.

Issue name	Creation date	Description
test	5/21/2024	issue test

Figura 6.8: Listado de tareas en Dashboard para empleados.

The screenshot shows a form titled 'Update food purchase' with fields for 'Product name' (Test compra), 'Ingredients' (millo), 'Supplier' (Purina S.A.), 'Kilograms' (100), and 'Price per kilo (€)' (2). Buttons for 'Edit' and 'Delete' are at the bottom.

Data Stats

Update food purchase

Product name	Test compra
Ingredients	millo
Supplier	Purina S.A.
Kilograms	100
Price per kilo (€)	2

[Edit](#) [Delete](#)

Figura 6.9: Actualizar y eliminar datos de compra de alimentos.

Una de las funcionalidades más complicadas de esta iteración fué el listado y filtrado de las tareas. La complejidad de esta funcionalida recide en los diferentes filtros que se pueden aplicar al listado y como estos deben ser representados en el código SQL para que la consulta a la base de datos traiga los datos con el filtrado correcto.

Para poder realizar el filtrado de manera sencilla, se hizo la consulta con JPQL [24] en el DAO de tarea. JPQL nos permite crear querys a base de datos utilizando las entidades de java correspondientes a las tablas de la base de datos, de esta manera la query es más sencilla de construir ya que lo único que debe de conocer el desarrollador es la entidad de java (en este caso Tarea) y sus atributos. A continuación se presenta el código de la consulta en JPQL para el listado y filtrado de tareas 6.3.

```

1  @Override
2   public Slice<Issue> find(Long farmId, String issueName, String
3     startDate, String endDate, Boolean isDone, int page, int size) {
4
5     String[] tokens = getTokens(issueName);
6     String queryString = "SELECT i FROM Issue i "
7       "LEFT JOIN User u ON i.createdBy.id = u.id "
8       "WHERE u.farm.id = " + farmId.toString();
9
10    if (tokens.length > 0 || startDate != null || endDate !=
11      null || isDone != null) {
12        queryString += " AND ";
13    }
14
15    // issueName
16    if (tokens.length > 0) {
17
18      for (int i = 0; i < tokens.length - 1; i++) {
19        queryString += "LOWER(i.issueName) LIKE
20          LOWER(:token" + i + ") AND ";
21      }
22
23      queryString += "LOWER(i.issueName) LIKE LOWER(:token"
24      + (tokens.length - 1) + ")";
25    }
26
27    // startTime
28    if (startDate != null) {
29      if (tokens.length > 0) {
30        queryString += "AND ";
31      }
32      queryString += "i.creationDate >= :startDate ";
33    }
34
35    // endTime
36    if (endDate != null) {
37      if (tokens.length > 0 || startDate != null) {
38        queryString += " AND ";
39      }
40    }
41
42    return Page.of(queryString, page, size);
43  }

```

```

35         }
36         queryString += " i.creationDate <= :endDate ";
37     }
38
39     // isDone
40     if (isDone != null) {
41         if (tokens.length > 0 || startDate != null || endDate
42 != null) {
43             queryString += " AND ";
44         }
45         queryString += " i.isDone = :isDone ";
46     }
47
48     queryString += " ORDER BY i.creationDate DESC";
49
50     Query query = entityManager.createQuery(queryString)
51     .setFirstResult(page*size).setMaxResults(size+1);
52
53     // issueName
54     if (tokens.length > 0) {
55         for (int i = 0; i < tokens.length; i++) {
56             query.setParameter("token" + i, '%' + tokens[i] + '%');
57         }
58
59         // startDate
60         if (startDate != null) {
61             //query.setParameter("startTime",
62             startTime.toString().substring(0,10));
63             DateTimeFormatter formatter =
64             DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");
65             LocalDateTime sd = LocalDateTime.parse(startDate + "
66             23:59:59", formatter);
67             sd = sd.plusDays(1);
68             query.setParameter("startDate", sd);
69         }
70
71         // endTime
72         if (endDate != null) {
73             DateTimeFormatter formatter =
74             DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");
75             LocalDateTime ed = LocalDateTime.parse(endDate + "
76             23:59:59", formatter);
77             ed = ed.plusDays(1);
78             query.setParameter("endDate", ed);
79         }
80
81     }
82
83     return query.getResultList();
84 }

```

```

75
76     // isDone
77     if (isDone != null) {
78         query.setParameter("isDone", isDone);
79     }
80
81     List<Issue> issues = query.getResultList();
82     boolean hasNext = issues.size() == (size+1);
83
84     if (hasNext) {
85         issues.remove(issues.size()-1);
86     }
87
88     return new SliceImpl<>(issues, PageRequest.of(page, size),
89     hasNext);
}

```

Listing 6.3: Consulta JPQL para listado de tareas.

6.1.3 Tercera iteración

Sprint Planning

En la reunión de sprint planning el product owner establece las próximas historias de usuario a desarrollar según las prioridades que estan tengan. Las historias de usuario elegidas y su división en tareas son las que se muestran en las siguientes figuras ([6.19](#) a la [6.26](#)).

Historia de usuario	Tareas	Horas
US-17 Listar compra de alimentos	Desarrollo backend	5
	Desarrollo frontend	6
	Tests	2

Cuadro 6.19: US-17 Listar compra de alimentos

Historia de usuario	Tareas	Horas
US-19 Registrar nuevo ganado	Modificar base de datos	1
	Desarrollo backend	9
	Desarrollo frontend	9
	Tests	2

Cuadro 6.20: US-19 Registrar nuevo ganado

Historia de usuario	Tareas	Horas
US-20 Editar datos de ganado	Desarrollo backend	2
	Desarrollo frontend	5
	Tests	1

Cuadro 6.21: US-20 Editar datos de ganado

Historia de usuario	Tareas	Horas
US-21 Cambiar estado de ganado	Desarrollo backend	0.5
	Desarrollo frontend	1
	Tests	0.5

Cuadro 6.22: US-21 Cambiar estado de ganado

Historia de usuario	Tareas	Horas
US-22 Visualizar datos de ganado	Desarrollo backend	1
	Desarrollo frontend	3
	Tests	1

Cuadro 6.23: US-22 Visualizar datos de ganado

Historia de usuario	Tareas	Horas
US-25 Registrar Consumo de alimento	Modificar base de datos	1
	Desarrollo backend	9
	Desarrollo frontend	9
	Tests	2

Cuadro 6.24: US-25 Registrar Consumo de alimento

Historia de usuario	Tareas	Horas
US-18 Ver gráfica de consumo de una compra de alimento	Desarrollo backend	2.5
	Desarrollo frontend	2
	Tests	0.5

Cuadro 6.25: US-18 Ver gráfica de consumo de una compra de alimento

Historia de usuario	Tareas	Horas
US-26 Eliminar consumo de alimento	Desarrollo backend	0.5
	Desarrollo frontend	1
	Tests	0.5

Cuadro 6.26: US-26 Eliminar consumo de alimento

Ya obtenido el sprint backlog, se actualizó el tablero Kanban de Trello con las tareas correspondientes al sprint actual (figura 6.10).

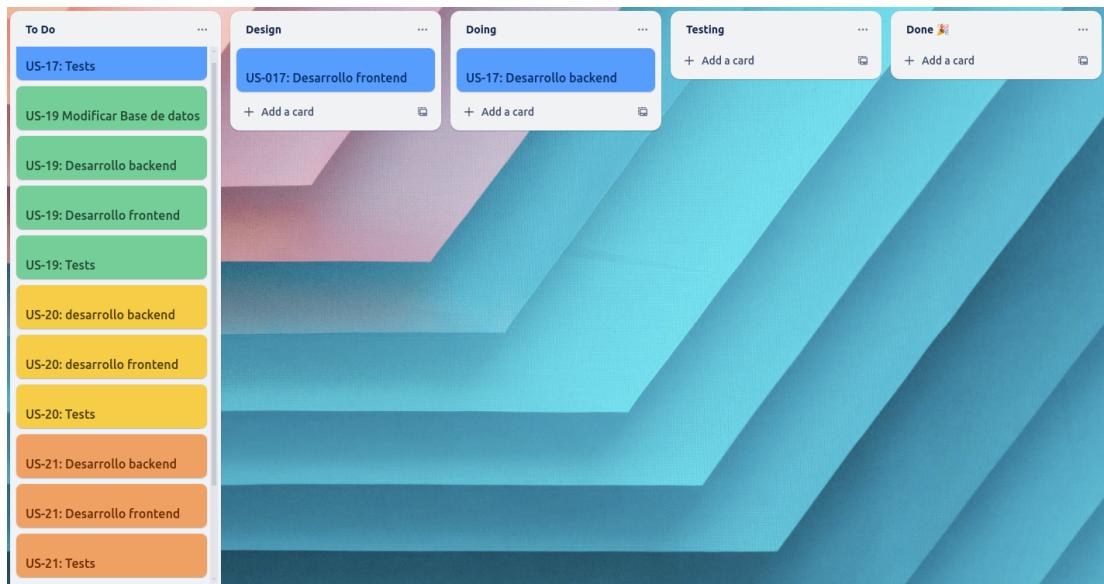


Figura 6.10: Tablero Kanban en la tercera iteración.

Sprint Review

En la reunión de sprint review, el product owner probó el incremento y pudo validar las 8 historias de usuarios desarrolladas. Siguiendo con la metodología, se procedió a medir el tiempo real de desarrollo de incremento y compararlo con las estimaciones iniciales (figura 6.27).

Cuadro 6.27: Comparación del tiempo estimado y el real de la tercera iteración

Historias de usuario	Horas estimadas	Horas reales
US-17 Listar compra de alimentos	13	12
US-19 Registrar nuevo ganado	21	17
US-20 Editar datos de ganado	8	7
US-21 Cambiar estado de ganado	2	2
US-22 Visualizar datos de ganado	5	5
US-25 Registrar Consumo de alimento	21	16
US-18 Ver gráfica de consumo...	5	5

..... (continúa na páxina seguinte)

Cuadro 6.27 – (vén da páxina anterior)

Historias de usuario	Horas estimadas	Horas reales
US-26 Eliminar consumo de alimento	2	2
TOTAL	77	66

En las figuras 6.11 a la 6.14 se pueden apreciar algunas de las historias de usuarios más destacadas del incremento desarrollado.

Product name:

Supplier:

Creation date:

Search

Total: 200€

millo

Supplier: Purina S.A.

Purchase date: 5/21/2024

Kilograms: 100

Price per kilo: 2€

View details

Back **Next**

Figura 6.11: Listado compras de alimentos.

Data Stats

Update animal info

Animal name: animal test

Identification Code: 123456

Physical description: rubia gallega

Date of birth: 05/21/2024

Gender: female male

Edit **Set unavailable**

Figura 6.12: Visualización y edición de datos de ganado.

Register food consumption

Food batch: Test compra

Consumed by: Select...

Kilos consumed: 1 Max Kilos: 62

Create

Figura 6.13: Registrar consumo de alimento.

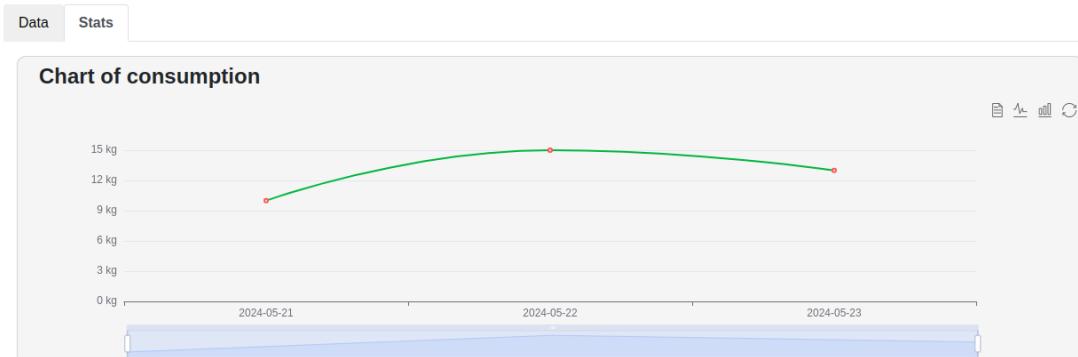


Figura 6.14: Gráfica consumo de un lote de alimento.

En esta iteración se llevó a cabo el desarrollo de gráficos de consumo de alimentos, el cual es una de las funcionalidades más importantes y críticas de la aplicación, ya que con este gráfico el usuario puede ver el patrón de consumo y así poder hacer compras de alimentos más eficientes en el futuro.

La librería utilizada para la creación de gráficos fue Apache Echarts debido a su fácil uso y la gran cantidad de estilos de gráficos que ofrece. Para el gráfico de consumo de un lote de alimentos, se agruparon por día los consumos de los diferentes animales que fueron alimentados con ese lote, por lo que cada punto del gráfico representa la suma de los kilos de alimento consumido por los animales en un día en concreto. A continuación, se muestra el código de configuración del gráfico en JavaScript con los datos obtenidos de servidor.

```

1 let optionDefault = {
2     tooltip: {
3         trigger: 'axis',
4         axisPointer: {
5             type: 'shadow'
6         },

```

```

7         valueFormatter: (value) => value + ' kg'
8     },
9     toolbox: {
10        feature: {
11            dataView: { show: true, readOnly: true },
12            magicType: { show: true, type: ['line', 'bar'] },
13            restore: { show: true },
14        }
15    },
16    legend: {},
17    dataZoom: [
18        {
19            show: true,
20        },
21        {
22            type: 'inside',
23            show: true
24        }
25    ],
26    xAxis: {
27        type: 'category',
28        data: []
29    },
30    yAxis: {
31        type: 'value',
32        axisLabel: {
33            formatter: '{value} kg'
34        }
35    },
36    series: [
37    ]
38};
39
40const [option, setoption] = useState(optionDefault);
41
42useEffect(() => {
43    const id = Number(foodPurchaseId);
44    if (isLoading) {
45        // get chartData
46        getFoodConsumptionChartData(id, (chartData) => {
47            console.log(chartData);
48            setData(chartData);
49            let data = {
50                data: [],
51                type: 'line',
52                smooth: true,

```

```

53         itemStyle: {
54             color: '#fc5858'
55         },
56         lineStyle: {
57             color: '#00B63E'
58         }
59     };
60     chartData.forEach(dto => {
61         data.data.push(dto.kilos);
62
63         optionDefault.xAxis.data.push(dto.consumptionDate);
64     });
65     optionDefault.series.push(data);
66     setOption(optionDefault);
67     setIsLoading(false);
68 }, errors => {
69     setBackendErrors(errors);
70     setIsLoading(false);
71 });
72 }, []);

```

Listing 6.4: Código de creación de gráfico de consumo de alimento.

En el código 6.4 se puede apreciar que el objeto **optionDefault** contiene la configuración inicial del gráfico pero sin los datos, los datos son luego obtenidos desde el servidor con el hook **useEffect** que nos proporciona React.

6.1.4 Cuarta iteración

Sprint Planning

En el sprint Planning, el product owner indicó las siguientes historias de usuario a desarrollar según las prioridades de negocio. Las historias de usuario escogidas y su respectiva subdivisión en tareas se pueden apreciar en las figuras de la 6.28 a la 6.38.

Historia de usuario	Tareas	Horas
US-23 Listar ganado de explotación	Desarrollo backend	5
	Desarrollo frontend	6
	Tests	2

Cuadro 6.28: US-23 Listar ganado de explotación

Historia de usuario	Tareas	Horas
US-27 Listar consumos de alimento	Desarrollo backend	5
	Desarrollo frontend	6
	Tests	2

Cuadro 6.29: US-27 Listar consumos de alimento

Historia de usuario	Tareas	Horas
US-28 Registrar pesaje	Modificar Base de datos	1
	Desarrollo backend	3
	Desarrollo frontend	3
	Tests	1

Cuadro 6.30: US-28 Registrar pesaje

Historia de usuario	Tareas	Horas
US-29 Editar datos de un pesaje	Desarrollo backend	2
	Desarrollo frontend	2
	Tests	1

Cuadro 6.31: US-29 Editar datos de un pesaje

Historia de usuario	Tareas	Horas
US-30 Eliminar Registro de pesaje	Desarrollo backend	0.5
	Desarrollo frontend	1
	Tests	0.5

Cuadro 6.32: US-30 Eliminar Registro de pesaje

Historia de usuario	Tareas	Horas
US-31 Visualizar datos de pesaje	Desarrollo backend	1
	Desarrollo frontend	2
	Tests	0.5

Cuadro 6.33: US-31 Visualizar datos de pesaje

Historia de usuario	Tareas	Horas
US-32 Listar registros de pesajes	Desarrollo backend	5
	Desarrollo frontend	6
	Tests	2

Cuadro 6.34: US-32 Listar registros de pesajes

Historia de usuario	Tareas	Horas
US-33 Regsitrar ordeño	Modificar Base de datos	1
	Desarrollo backend	3
	Desarrollo frontend	3
	Tests	1

Cuadro 6.35: US-33 Regsitrar ordeño

Historia de usuario	Tareas	Horas
US-34 Editar datos de ordeño	Desarrollo backend	2
	Desarrollo frontend	2
	Tests	1

Cuadro 6.36: US-34 Editar datos de ordeño

Historia de usuario	Tareas	Horas
US-35 eliminar Registr deo ordeño	Desarrollo backend	0.5
	Desarrollo frontend	1
	Tests	0.5

Cuadro 6.37: US-35 eliminar Registr deo ordeño

Historia de usuario	Tareas	Horas
US-36 Visualizar datos de ordeño	Desarrollo backend	1
	Desarrollo frontend	1
	Tests	1

Cuadro 6.38: US-36 Visualizar datos de ordeño

Una vez obtenido el sprint backlog, se procede a actualizar el tablero kanban con las tareas correspondientes (figura 6.15).

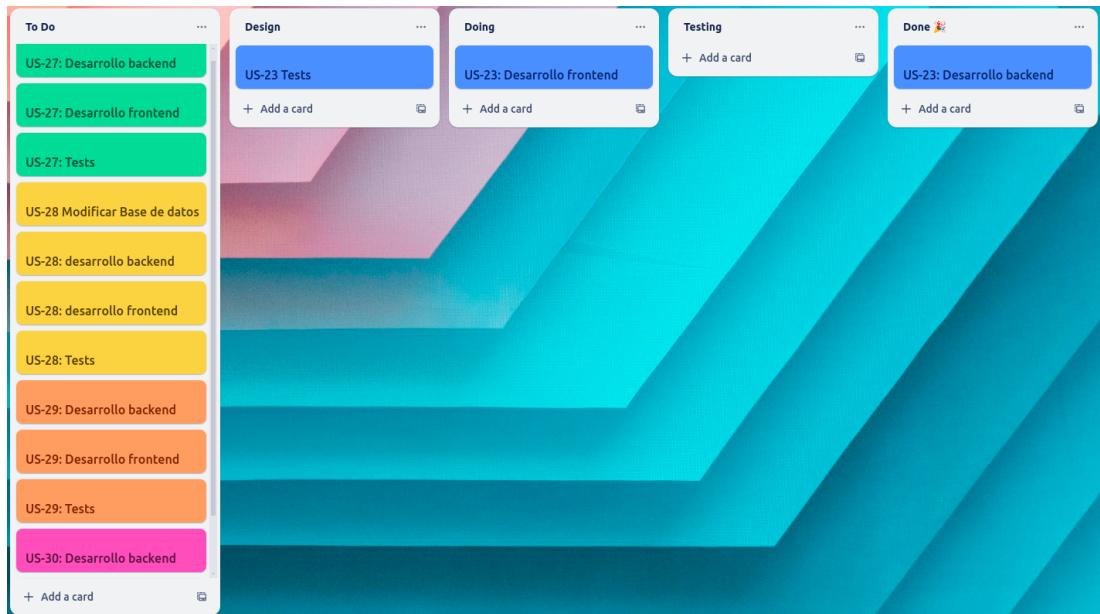


Figura 6.15: Tablero Kanban en la cuarta iteración.

Sprint Review

Siguiendo con las costumbre de las iteraciones pasadas, en esta reunión el product owner probó las funcionalidades del incremento y dió el visto bueno a las 11 historias de usuario desarrolladas. Una vez obtenido el feedback del product owner, se procedió a medir el tiempo real de desarrollo del incremento y compararlo con las estimaciones iniciales [6.39](#).

Cuadro 6.39: Comparación del tiempo estimado y el real de la cuarta iteración

Historias de usuario	Horas estimadas	Horas reales
US-23 Listar ganado de explotacion	13	11
US-27 Listar consumos de alimento	13	11
US-28 Registrar pesaje	8	9
US-29 Editar datos de un pesaje	5	5
US-30 Eliminar registros de pesaje	2	2
US-31 Visualizar datos de pesaje	3	4
US-32 Listar registros de pesaje	13	10
US-33 Registrar ordeño	8	8
US-34 Editar datos de ordeño	5	5
US-35 Eliminar registros de ordeño	2	2
US-36 Visualizar datos de ordeño	5	5
TOTAL	75	72

En las figuras [6.16](#) a la [6.20](#) se pueden apreciar algunas de las historias de usuarios desarrolladas en este sprint.

Animal name: Identification Code: State: Creation date:

animal name		Available	

Search

123456

rubia gallega

Animal name: animal test
Date of birth: 5/21/2024
Gender: female

See details

Back **Next**

Figura 6.16: Listado y filtrado de ganado.

Consumed by Food batch Date of consumption:

Select...	-- All options --	

Search

Food consumption

Consumed by: animal test / 123456
Food batch: Test compra
Kilos consumed: 13
Date of consumption: 5/23/2024

Delete

Food consumption

Consumed by: animal test / 123456
Food batch: Test compra
Kilos consumed: 15
Date of consumption: 5/22/2024

Delete

Figura 6.17: Listado y filtrado de consumo de comida.

Update weighing record

Animal weighed: animal test / 123456

Weight (kg): 199

Do you want this weighing to be used in meat production stats?

[Edit](#) [Delete](#)

Figura 6.18: Visualización de datos de pesaje.

Update milking record

Animal milked: animal test / 123456

Liters: 15

[Edit](#) [Delete](#)

Figura 6.19: Visualización de datos de ordeño.

Animal weighed: [-- All options --](#)

Type of weighing: [All](#)

Date of weighing:

Weight range (kg): [min](#) [max](#)

[Search](#)

animal test / 123456

Weight (kg): 199

Date of weighing: 5/21/2024

[See details](#)

[Back](#) [Next](#)

Figura 6.20: Visualización de datos de ordeño.

Uno de los aspectos más importantes del frontend es su capa accesos a servicio, ya que con esta puede hacer peticiones al servidor mediante la API que este expone a los clientes. En este proyecto se hace uso del método `fetch()` [25], la cual nos permite acceder a recursos mediante la red haciendo peticiones HTTP. A continuación se puede observar el archivo `appFetch.js`

6.5 del frontend, el cual define la función appFetch, la cual es usada por la capa acceso a servicios para hacer uso de las API desarrollada en la capa servicios del servidor.

```

1 import NetworkError from './NetworkError';
2 const SERVICE_TOKEN_NAME = 'serviceToken';
3
4 let networkErrorCallback;
5 let reauthenticationCallback;
6
7 const isJson = response => {
8     const contentType = response.headers.get("content-type");
9     return contentType && contentType.indexOf("application/json")
10    !== -1;
11 }
12
13 const handleOkResponse = (response, onSuccess) => {
14     if (!response.ok) {
15         return false;
16     }
17     if (!onSuccess) {
18         return true;
19     }
20     if (response.status === 204) {
21         onSuccess();
22         return true;
23     }
24     if (isJson(response)) {
25         response.json().then(payload => onSuccess(payload));
26     }
27     return true;
28 }
29
30 const handle4xxResponse = (response, onErrors) => {
31     if (response.status < 400 || response.status >= 500) {
32         return false;
33     }
34     if (response.status === 401 && reauthenticationCallback){
35         reauthenticationCallback();
36         return true;
37     }
38     if (!isJson(response)) {
39         throw new NetworkError();
40     }
41     if (onErrors) {
42         response.json().then(payload => {
43             if (payload.globalError || payload.fieldErrors) {
44                 onErrors(payload);
45             }
46         });
47     }
48 }
49
50 const handle5xxResponse = (response, onErrors) => {
51     if (response.status >= 500) {
52         return false;
53     }
54     if (onErrors) {
55         response.json().then(payload => {
56             if (payload.globalError || payload.fieldErrors) {
57                 onErrors(payload);
58             }
59         });
60     }
61 }
62
63 const appFetch = (url, options) => {
64     return fetch(url, options)
65        .then(handleOkResponse)
66        .catch(handle4xxResponse)
67        .catch(handle5xxResponse);
68 }
69
70 export default appFetch;
71 
```

```

43         onErrors(payload);
44     }
45   });
46 }
47 return true;
48 }

50 const handleResponse = (response, onSuccess, onErrors) => {
51   if (handleOkResponse(response, onSuccess)) {
52     return;
53   }
54   if (handle4xxResponse(response, onErrors)) {
55     return;
56   }
57   throw new NetworkError();
58 }

59 export const init = callback => networkErrorCallback = callback;
60
62 export const setReauthenticationCallback = callback =>
63   reauthenticationCallback = callback;
64
65 export const setServiceToken = serviceToken =>
66   sessionStorage.setItem(SERVICE_TOKEN_NAME, serviceToken);

67 export const getServiceToken = () =>
68   sessionStorage.getItem(SERVICE_TOKEN_NAME);

69 export const removeServiceToken = () =>
70   sessionStorage.removeItem(SERVICE_TOKEN_NAME);

71 export const config = (method, body) => {
72   const config = {};
73   config.method = method;
74   if (body) {
75     if (body instanceof FormData) {
76       config.body = body;
77     } else {
78       config.headers = {'Content-Type': 'application/json'};
79       config.body = JSON.stringify(body);
80     }
81   }
82   let serviceToken = getServiceToken();
83   if (serviceToken) {
84     if (config.headers) {
85       config.headers['Authorization'] = `Bearer

```

```

87     `${serviceToken}` ;
88   } else {
89     config.headers = { 'Authorization': `Bearer
90       ${serviceToken}` };
91   }
92   return config;
93 }
94 export const appFetch = (path, options, onSuccess, onErrors) =>
95   fetch(`process.env.REACT_APP_BACKEND_URL${path}`, options)
96     .then(response => handleResponse(response, onSuccess,
97       onErrors))
98       .catch(networkErrorCallback);

```

Listing 6.5: Código del archivo appFetch.

6.1.5 Quinta iteración

Sprint Planning

Siendo este el último sprint, en la reunión de sprint planning el product owner nos indicó las historias de usuario que quedan pendientes. Las historias de usuario son las siguientes tres (cuadros 6.40, 6.41 y 6.42).

Historia de usuario	Tareas	Horas
US-37 Listar registros de ordeño	Desarrollo backend	5
	Desarrollo frontend	6
	Tests	2

Cuadro 6.40: US-37 Listar registros de ordeño

Historia de usuario	Tareas	Horas
US-24 Ver gráfica sobre un animal	Desarrollo backend	13
	Desarrollo frontend	16
	Tests	5

Cuadro 6.41: US-24 Ver gráfica sobre un animal

Historia de usuario	Tareas	Horas
US-38 Dashboard de gráficas	Desarrollo backend	13
	Desarrollo frontend	16
	Tests	5

Cuadro 6.42: US-38 Dashboard de gráficas

Ya obtenido el sprint backlog con todas las tareas a desarrollar, se procede a actualizar el tablero Kanban en Trello [6.21](#).

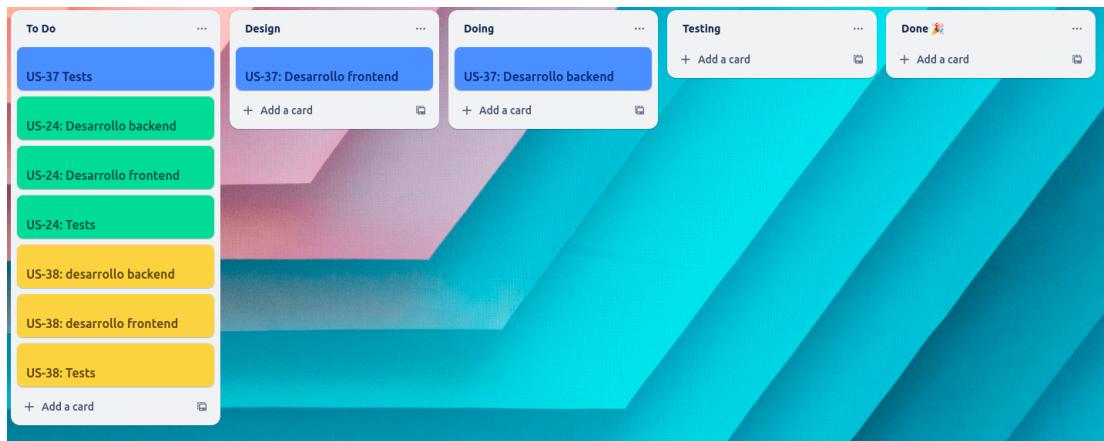


Figura 6.21: Tablero Kanban de la quinta iteración.

Sprint Review

En la última reunión de Sprint review, el product owner probó todas las historias de usuarios desarrolladas hasta la fecha. De esta manera, el product owner al probar todas las funcionalidades de la aplicación, pudo dar el visto bueno para finalizar la fase de desarrollo de proyecto.

Una vez validado todas las funcionalidades, se procedió a medir el tiempo real de desarrollo y compararlo con el tiempo estimado inicial [6.43](#).

Cuadro 6.43: Comparación del tiempo estimado y el real de la quinta iteración

Historias de usuario	Horas estimadas	Horas reales
US-37 Listar registros sobre un animal	13	10
US-24 Ver gráficas sobre un animal	34	26
US-38 Dashboard de gráficas	34	27
TOTAL	81	63

En las figuras 6.22 a la 6.28 se muestran algunas de las funcionalidades más destacadas del incremento entregado.

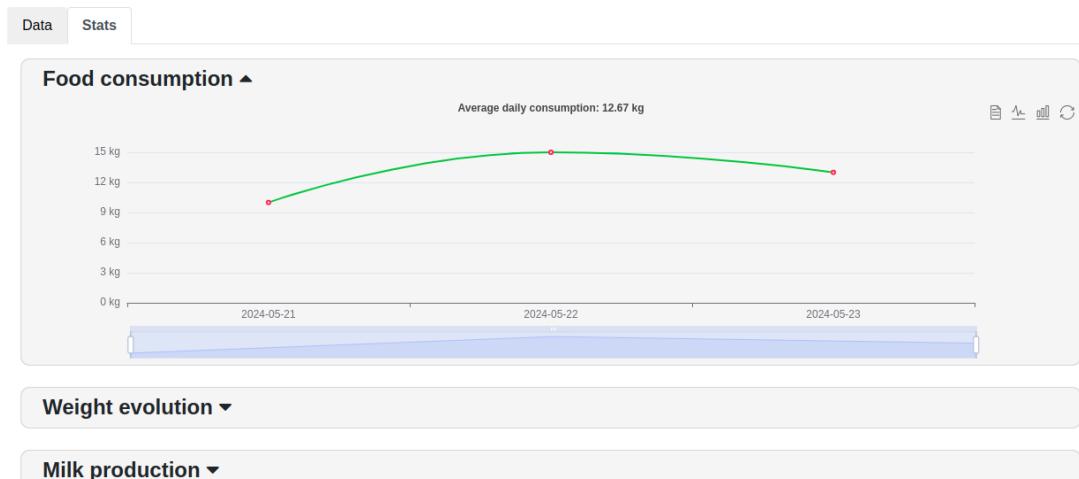


Figura 6.22: Gráfico de consumo de comida de un animal.

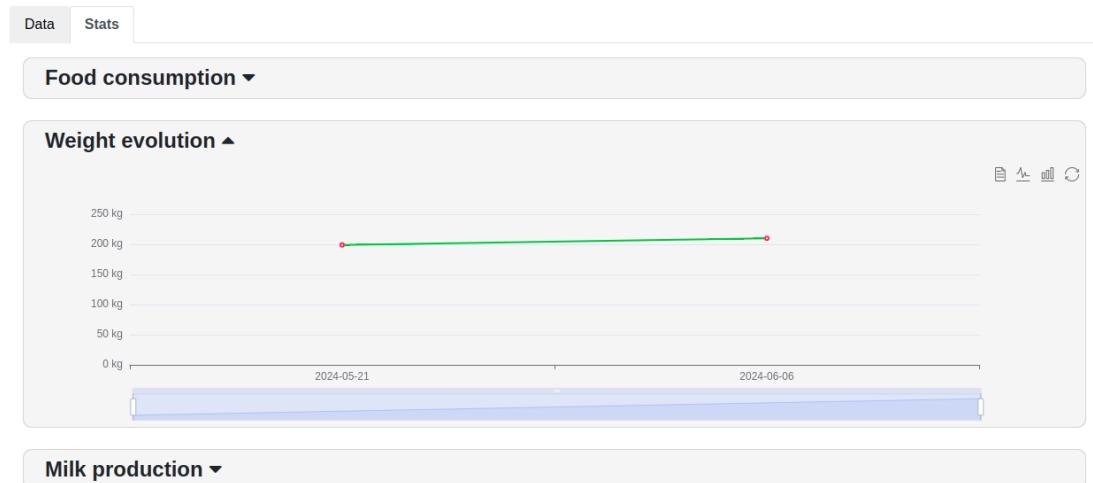


Figura 6.23: Gráfico de evolución de peso de un animal.



Figura 6.24: Gráfico de producción lechera de un animal.

Dashboard

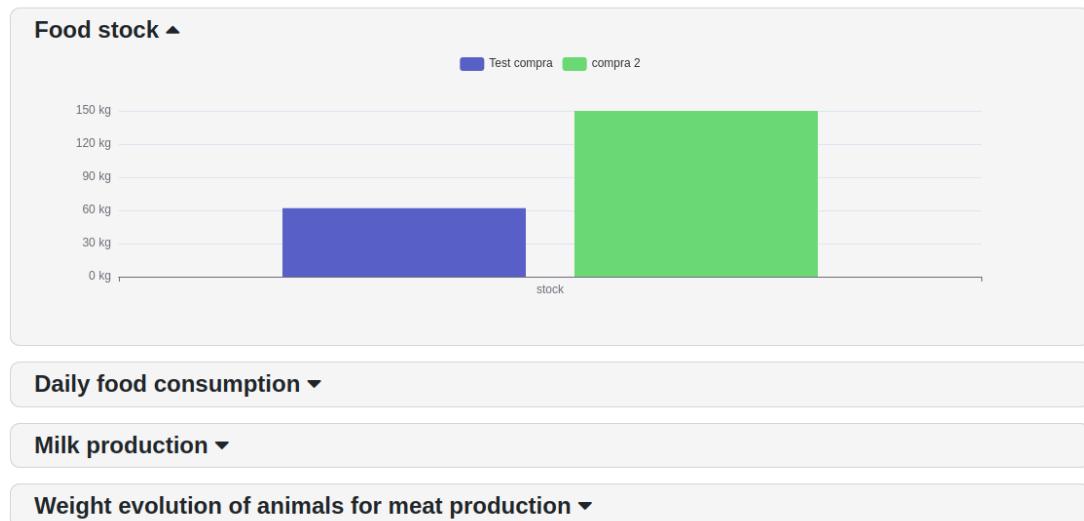


Figura 6.25: Gráfico de stock de alimentos.

Dashboard

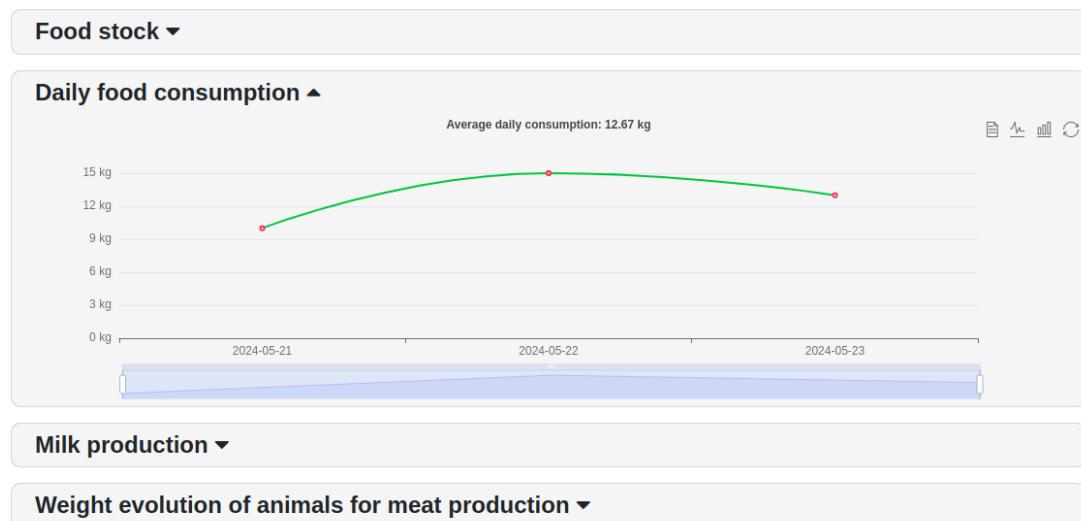


Figura 6.26: Gráfico de consumo de comida de la explotación.

Dashboard



Figura 6.27: Gráfico de producción lechera de la explotación.

Dashboard



Figura 6.28: Gráfico de pesajes destinados a la producción cárnica.

Este Sprint fue uno de los más importantes en cuanto a la importancia de las funcionalidades para los clientes, ya que en este sprint se desarrolló la mayoría de gráficos para el análisis del estado de la explotación y la toma de decisiones de negocio.

Uno gráfico muy importante es el del stock de alimento, ya que este indica cuantos kilogramos de alimentos quedan de cada lote en la explotación. Con esta información, el usuario

puede saber cuando hay que comprar alimentos y cuanta cantidad ha de comprar.

En los siguientes fragmentos de código se puede apreciar como es el proceso de obtención de la información de este gráfico. En el código 6.6 se puede apreciar como es la obtención de los datos en la capa **lógica de negocio** del servidor, en esta capa primero se hacen comprobaciones de que la petición la haga un usuario válido y luego se procede a hacer la llamada al DAO para obtener todos las compras de alimentos con stock disponible, el código de este método del DAO se puede apreciar en este fragmento de código 6.7. Una vez obtenida todos los lotes de alimentos disponibles, se procede a realizar un bucle sobre esta lista para crear los **Data transfer object (DTO)** y pasarselos a la capa de servicios y esta eventualmente enviarselos al frontend.

```

1 @Override
2 public List<StockChartDto> getStockChart(Long userId) throws
3     InstanceNotFoundException {
4     Optional<User> optionalUser = userDao.findById(userId);
5
6     if (optionalUser.isEmpty()) {
7         throw new
8             InstanceNotFoundException("project.entities.user", userId);
9     }
10
11    User user = optionalUser.get();
12    Farm farm = user.getFarm();
13
14    // foodBatches with stock left.
15    List<FoodPurchase> foodPurchasesWithStockLeft =
16        foodPurchaseDao.getAllFoodPurchasesWithStockLeft(farm.getId());
17
18    List<StockChartDto> stockChartDtos = new ArrayList<>();
19
20    foodPurchasesWithStockLeft.forEach(fp -> {
21        Long sumOfKilosConsumed =
22            foodPurchaseDao.getAllKilosConsumedByFoodPurchaseId(fp.getId());
23        StockChartDto dto = new
24            StockChartDto(FoodPurchaseConversor.toFoodPurchaseDto(fp),
25            fp.getKilos() - (sumOfKilosConsumed != null ? sumOfKilosConsumed
26                : 0L));
27        stockChartDtos.add(dto);
28    });
29    return stockChartDtos;
30}
```

Listing 6.6: Código de obtención de datos para el gráfico de stock en la capa lógica de negocio.

```

1 @Query("Select fp from FoodPurchase fp " +
```

```

2      "join User u ON fp.madeBy.id = u.id " +
3      "left join FoodConsumption fc ON fp.id =
4          fc.foodBatch.id " +
5      "where u.id = :farmId " +
6      "group by fp.id " +
7      "Having COALESCE(fp.kilos - COALESCE(SUM(fc.kilos), 0),
8          fp.kilos) > 0")
9  List<FoodPurchase> getAllFoodPurchasesWithStockLeft(Long
10     farmId);
11 }
```

Listing 6.7: Código de obtención de los lotes de alimentos disponibles.

6.2 Testing

el testing de una aplicación es una de las partes más fundamentales e importantes de su desarrollo, ya que con las pruebas podemos generar más confianza sobre el trabajo desarrollado y se detectan posibles errores y bugs antes de que estos lleguen al entorno de producción.

En este proyecto se llevaron a cabo diferentes tipos de pruebas sobre las funcionalidades desarrolladas, tanto automatizadas como manuales. Los tipos de pruebas realizadas en el proyecto fueron de integración, de sistema y de aceptación.

6.2.1 Pruebas de integración

Para las pruebas de integración se utilizó el framework [framework](#) de testing Junit [6] para su implementación. Uno de las métricas más importantes para medir la calidad de los test de integración es su coverage sobre el código, es decir, cuantas líneas de código fuente es ejecutado mediante los tests. Para visualizar el coverage de las pruebas desarrolladas, se usó el módulo de test coverage que nos proporciona el IDE IntelliJ IDEA, la cual hace uso de la herramienta JaCoCo.

Para las pruebas de integración, el objetivo es probar el correcto funcionamiento de la capa de lógica de negocio, ya que esta hace uso de las capa modelo (DAOs y entidades) y es la que se encarga de que cada funcionalidad se comporte como los dictan los requisitos. En la figura [6.29](#) se puede apreciar el coverage obtenido en el directorio **services**, el cual es el que contiene el código correspondiente a la capa de lógica de negocio.

Coverage: services in TFG-project-backend				
Element	Class, %	Method, %	Line, %	Branch, %
es.udc.paproject.backend	52% (40/76)	38% (213/549)	55% (832/1504)	44% (281/638)
model	87% (28/32)	94% (195/207)	82% (709/856)	46% (279/594)
entities	100% (15/15)	99% (131/132)	76% (352/459)	29% (105/352)
services	100% (9/9)	93% (60/64)	91% (349/381)	71% (174/242)
UserServiceImpl	100% (1/1)	100% (10/10)	97% (44/45)	84% (22/26)
FoodConsumptionServiceImpl	100% (1/1)	100% (10/10)	100% (50/50)	80% (21/26)
FoodPurchaseServiceImpl	100% (1/1)	100% (9/9)	91% (57/62)	67% (23/34)
MilkingServiceImpl	100% (1/1)	100% (7/7)	100% (42/42)	83% (25/30)
WeighingServiceImpl	100% (1/1)	100% (7/7)	97% (43/44)	78% (25/32)
AnimalServiceImpl	100% (1/1)	100% (7/7)	98% (55/56)	77% (28/36)
IssueServiceImpl	100% (1/1)	100% (7/7)	98% (50/51)	77% (28/36)
FoodPurchaseService	100% (0/0)	100% (0/0)	100% (0/0)	100% (0/0)
FoodConsumptionService	100% (0/0)	100% (0/0)	100% (0/0)	100% (0/0)
MilkingService	100% (0/0)	100% (0/0)	100% (0/0)	100% (0/0)
AnimalService	100% (0/0)	100% (0/0)	100% (0/0)	100% (0/0)
WeighingService	100% (0/0)	100% (0/0)	100% (0/0)	100% (0/0)
IssueService	100% (0/0)	100% (0/0)	100% (0/0)	100% (0/0)
PermissionChecker	100% (0/0)	100% (0/0)	100% (0/0)	100% (0/0)
UserService	100% (0/0)	100% (0/0)	100% (0/0)	100% (0/0)
PermissionCheckerImpl	100% (1/1)	50% (1/2)	66% (4/6)	50% (2/4)
Block	100% (1/1)	40% (2/5)	16% (4/25)	0% (0/18)
exceptions	50% (4/8)	36% (4/11)	50% (8/16)	100% (0/0)
Application	100% (1/1)	75% (3/4)	88% (8/9)	100% (0/0)
rest	25% (11/43)	4% (15/338)	17% (115/639)	4% (2/44)

Figura 6.29: Coveraga de las pruebas de integración.

6.2.2 Pruebas de sistema

Para las pruebas de sistema, se usó la herramienta **Postman** [8], la cual nos permite probar el API del servidor. Para probar el API manualmente, se creó una petición por cada endpoint expuesto por el API. En la figura 6.30 se puede apreciar las peticiones creadas en Postman para probar algunos endpoints del API de usuario.

The screenshot shows the Postman application interface. On the left, there's a sidebar with a tree view of collections: 'TFG' is expanded, showing its sub-endpoints. The main area is titled 'HTTP TFG / Login'. A 'POST' request is selected, with the URL 'http://localhost:8080/users/login' in the address bar. Below the URL, there are tabs for 'Params', 'Authorization', 'Headers (8)', 'Body', 'Pre-request Script', 'Tests', and 'Settings'. The 'Body' tab is active, showing a JSON payload with two fields: 'userName' and 'password', both set to 'admin' and 'pa2122' respectively. There are also tabs for 'none', 'form-data', 'x-www-form-urlencoded', 'raw', 'binary', and 'GraphQL'.

Figura 6.30: Pruebas de sistema con Postman.

6.2.3 Pruebas de aceptación

Como recomienda la metodología Scrum, las pruebas de aceptación fueron realizadas por el product owner del equipo mediante el uso de la aplicación en un entorno controlado. Al realizarse las pruebas de aceptación y no detectar ningún problema, se dan por validadas las funcionalidades implementadas, es decir, las funcionalidades se desarrolladas correspondían y satisfacían los requisitos del usuario final.

6.3 Despliegue

En cuanto al despliegue de la aplicación, se puede llevar con el uso de dos herramientas: Docker [14] para despliegues en base a contenedores, y Kubernetes [15] para hacer despliegues más complejos enfocados a una estructura de microservicios.

6.3.1 Docker

Docker es una herramienta que ayuda a automatizar despliegues de aplicaciones dentro de contenedores aislados, garantizando así una consistencia de la aplicación en distintos entornos gracias a la encapsulación de sus dependencias. Para este proyecto, el frontend, el backend y la base de datos se encapsularían en diferentes contenedores para reducir la complejidad del despliegue y tener la opción de desplegar cada uno en entornos diferentes por si fuese necesario en el futuro.

Una de las ventajas de docker es la fácil gestión de los contenedores gracias al uso de **Docker Desktop**, la cual nos proporciona una interfaz gráfica sencilla para crear, modificar y ejecutar los diferentes contenedores que tenemos en nuestro sistema.

6.3.2 Kubernetes

Kubernetes también es una herramienta de despliegues basado en contenedores pero con la diferencia de que esta enfocada más hacia una arquitectura de cliente-servidor con microservicios, lo cual hace que sea más compleja de gestionar pero de ciertas ventajas que otras arquitecturas no ofrecen como: la tolerancia a fallos, actualizaciones en caliente, reinicio automático, entre otros.

Debido a que el sistema desarrollado no es tan complejo como para subdividirlo en microservicios, se puede crear un pequeño cluster de Kubernetes que contengan los tres contenedores de docker del sistema. Con los contenedores de docker metidos en un contenedor de Kubernetes, se pueden crear réplicas de este para distribuir la carga de trabajo y tener la garantía de que si un contenedor falla esté el otro como backup.

Capítulo 7

Conclusiones y trabajos futuros

Como último capítulo de la memoria, se van a tratar las conclusiones que se obtuvieron de la realización del proyecto y se plantearán algunas posibles mejoras futuras que pueden ayudar a mejorar la aplicación.

7.1 Conclusiones

Tras finalizar el proyecto, se pueden sacar las siguientes conclusiones:

- **Objetivos cumplidos:** Una vez terminado el desarrollo del sistema, se considera que se han cumplido los objetivos planteados inicialmente, ya que el feedback recibido por parte del cliente es muy positivo. Además, se llegaron a desarrollar todas las historias de usuarios propuestas por el cliente y este las pudo validar mediante la realización de las pruebas de aceptación, las cuales fueron pasadas con total éxito. Por otra parte, también se cumplieron con todos los requisitos no funcionales establecidos al inicio del proyecto gracias a la arquitectura del proyecto, las tecnologías usadas y los patrones de diseño utilizados.
- **Desviación de la planificación:** Al inicio del proyecto después de la estimación inicial se obtuvo que el desarrollo llevaría unas 389 horas de esfuerzo. Sin embargo, el desarrollo de proyecto llevó unas 342 horas, por lo tanto hubo de una desviación de 47 horas. El principal motivo de esta sobreestimación es la falta de históricos por parte del equipo de Scrum, ya que al no tener ningún tipo de referencia de proyectos previamente realizados por el equipo, las estimaciones que se hagan tendrán una gran incertidumbre.
- **Tecnologías utilizadas:** En cuanto a las tecnologías escogidas, ya tenía cierta experiencia con estas, sobre todo con Spring boot, React y Redux. Sin embargo, si tuve que pasar por una cierta curva de aprendizaje con la librería gráfica Apache Echarts ya que

no la había utilizado nunca y me resultaba complicado utilizarla al principio. Una lección que me llevo de esto es que es de gran importancia hacer un buen estudio previo del alcance de las tecnologías que se utilizarán, ya que estas son las herramientas darán forma a nuestro proyecto y delimitan de cierta manera lo que podemos o no hacer.

- **Organización y metodología:** Uno de los aspectos más importantes para lograr un buen proyecto y no tener problemas durante su desarrollo fue elegir e implementar de manera correcta las metodologías de Scrum y Kanban, ya que con estas metodologías podíamos saber el estado del proyecto de manera sencilla y teníamos siempre presente cuales eran los aspectos y funcionalidades mas importantes para los clientes. Cabe destacar la importancia de los sprint planning y sprint reviews, ya que al tener a los clientes presentes en estas reuniones (product owners) había una comunicación más directa y así se podían obtener con mas detalle los requisitos que estos planteaban.

7.2 Trabajos para el futuro

Aún habiendo cumplido con todos los objetivos y requisitos del proyecto, se proponen ciertas funcionalidades y módulos que pueden ayudar a mejorar la utilidad del sistema y su alcance.

- **Importación de datos:** En muchas explotaciones ganaderas, los datos de producción están guardadas en archivos de tipo excel o csv, por lo que sería de gran utilidad estudiar la posibilidad de desarrollar un módulo en la aplicación a la cual se le pase un archivo con estos datos y el sistema los importe a la base de datos.
- **Registro de explotaciones:** Por el momento cualquier explotación ganadera que quiera utilizar la aplicación debe de ponerse en contacto con alguno de los administradores del sistema. Esto se hizo por motivos de seguridad y para evitar que cualquiera se pudiese crear una cuenta y gastar recursos de forma irresponsable. Esto lleva a que se deben de tener personal pendiente de las solicitudes de creación de cuentas y realizar la comprobación, por lo que se propone un proceso automatizado en la aplicación para la creación de las cuentas.
- **Módulo veterinario:** Uno de los aspectos más delicados sobre las explotaciones ganaderas es mantener el ganado correctamente vacunado y estar al día de todas las tareas sanitarias sobre estos. Debido a esto, se puede estudiar la creación de un módulo en la aplicación que se encargue de llevar un registro de las vacunas que tiene cada animal y recordar al usuario de jornadas de vacunación futuras.

- **Módulo avícola y/o porcicultura:** Ahora mismo el sistema solo gestiona explotaciones ganaderas, pero es cierto que hoy en día la avicultura **avicultura** y la porcicultura **porcicultura** están tan extendidas como la ganadera, por lo que se puede desarrollar en un futuro dos módulos con varias funcionalidades para la gestión de explotaciones que se encarguen a estas otras variantes agrícolas.

Apéndices

Lista de acrónimos

DTO Data transfer object. [65](#)

IDE Integrated Development Environment. [4](#)

PH Punto historia. [19](#)

SaaS Software as a Service. [1](#)

SGBD Sistema de Gestión de Base de Datos. [4](#)

SPA Single Page Application. [5, 22](#)

UML Unified Modeling Language. [24](#)

WIP Work in Progress. [10](#)

Glosario

avicultura Se refiere a todo lo relacionado con la cría y explotación de aves de corral, especialmente para la producción de carne, huevos y, en algunos casos, plumas.. [71](#)

backend Se refiere a la parte de la aplicación que se encuentre en el servidor y al que a menudo se le denomina “el lado del servidor”.. [4](#)

framework Un framework (o marco de trabajo) es una estructura de soporte definida en la cual otro software puede desarrollarse. Proporciona una base reutilizable sobre la cual los desarrolladores pueden construir aplicaciones más específicas o complejas.. [66](#)

frontend Es la parte de una aplicación que interactúa con los usuarios, es conocida como el lado del cliente.. [5](#)

porcicultura La porcicultura es la actividad dedicada a la cría y explotación de cerdos, conocida también como ganadería porcina.. [71](#)

Bibliografía

- [1] MySQL, “Sistema de gestión de bases de datos relacional,” 2024. [En línea]. Disponible en: <https://www.mysql.com/>
- [2] I. IDEA, “Integrated development environment,” 2024. [En línea]. Disponible en: <https://www.jetbrains.com/es-es/idea/>
- [3] Java, “Lenguaje de programación,” 2024. [En línea]. Disponible en: <https://www.java.com/es/>
- [4] S. Boot, “Framework para el desarrollo de aplicaciones en java,” 2024. [En línea]. Disponible en: <https://spring.io/>
- [5] Hibernate, “Mapeo objeto-realclional para aplicaciones java,” 2024. [En línea]. Disponible en: <https://hibernate.org/>
- [6] J. 5, “Librerías para crear pruebas automatizadas en java.” [En línea]. Disponible en: <https://junit.org/junit5/>
- [7] Maven, “Gestión de dependencias.” [En línea]. Disponible en: <https://maven.apache.org/>
- [8] Postman, “Test de api rest.” [En línea]. Disponible en: <https://www.postman.com/>
- [9] M. Developer, “Lenguaje de programación javascript.” [En línea]. Disponible en: <https://developer.mozilla.org/es/docs/Web/JavaScript>
- [10] React, “Biblioteca para interfaces de usuario web y nativas.” [En línea]. Disponible en: <https://es.react.dev/>
- [11] Redux, “Gestor de estado.” [En línea]. Disponible en: <https://redux.js.org/>
- [12] Echarts, “An open source javascript visualization library.” [En línea]. Disponible en: <https://echarts.apache.org/en/index.html>

- [13] Node.js, “Javascript runtime environment.” [En línea]. Disponible en: <https://nodejs.org/en>
- [14] Docker, “Uso de contenedores para el despliegue de aplicaciones.” [En línea]. Disponible en: <https://www.docker.com/>
- [15] Kubernetes, “Gestión de contenedores y aplicaciones.” [En línea]. Disponible en: <https://www.docker.com/>
- [16] Git, “Control de versiones.” [En línea]. Disponible en: <https://git-scm.com/>
- [17] GitHub, “Servicio web de git.” [En línea]. Disponible en: <https://github.com/>
- [18] Trello, “Herramienta de gestión de proyectos.” [En línea]. Disponible en: <https://trello.com/es>
- [19] Draw.io, “Herramienta para la creación de diagramas.” [En línea]. Disponible en: <https://app.diagrams.net/>
- [20] Scrum, “Guía de scrum: qué es, cómo funciona y cómo empezar.” [En línea]. Disponible en: <https://www.atlassian.com/es/agile/scrum>
- [21] Kanban, “Aplicación de la metodología kanban en el desarrollo de software.” [En línea]. Disponible en: <https://www.atlassian.com/es/agile/kanban>
- [22] P. Poker, “Easy-to-use and fun story point estimations.” [En línea]. Disponible en: <https://planningpokeronline.com/>
- [23] Talent, “Salario medio para ingeniero informático en España, 2024.” [En línea]. Disponible en: <https://es.talent.com>
- [24] JPQL, “Java persistence query language.” [En línea]. Disponible en: https://www.tutorialspoint.com/jpa/jpa_jpql.htm
- [25] fetch, “fetch() global function.” [En línea]. Disponible en: <https://developer.mozilla.org/en-US/docs/Web/API/fetch>