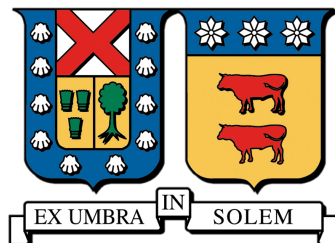


UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE ELECTRÓNICA

VALPARAÍSO - CHILE



“DESARROLLO DE APLICACIÓN MÓVIL
ESCALABLE PARA EL REGISTRO Y
SEGUIMIENTO DE LA INFORMACIÓN MÉDICA
DE MASCOTAS”

PILAR GABRIELA ARANCIBIA ECHEVERRÍA

MEMORIA DE TITULACIÓN PARA OPTAR AL TÍTULO DE INGENIERO EN
EJECUCIÓN ELECTRÓNICA MENCIÓN COMPUTADORES

PROFESOR GUÍA: AGUSTÍN GONZALEZ

PROFESOR CORREFERENTE: JOSÉ MARTINEZ

JULIO - 2024



Agradecimientos

No puedo empezar a agradecer a nadie antes que a las personas que lo dieron todo para que yo esté donde estoy ahora: mis padres, **Sylvia Echeverría** y **Mauricio Arancibia**. A ellos les debo mucho, incluyendo la oportunidad de estudiar y todo el constante apoyo que me han brindado desde que nací, siempre tienen algo lindo que decirme, jamás me negaron el apoyo y me tuvieron mucha paciencia. Son unos padres ejemplares y los adoro.

A mis hermanos, **Lorena Arancibia**, **Gabriela Arancibia** y **Xavier Arancibia**, que estuvimos juntos, apoyándonos en la experiencia de vivir lejos de casa. Agradezco que hayan sido un punto de apoyo y sostén importante para mí y que hayamos evolucionado en nuestras relaciones de manera tan bonita.

A mi nueva familia, **Sergio Olmos**, mi pareja. No solo me ha apoyado emocionalmente y en el hogar para que yo pudiera concentrarme en mis obligaciones, sino que también me ha apoyado activamente en todo el proceso de este proyecto, programando, diseñando, opinando y gestionando. Sergio, has sido el mayor y más cercano punto de apoyo en este proceso tan difícil, y por ello te estoy profundamente agradecida.

A las personas que formaron parte del equipo para el desarrollo de este proyecto:

- **Esteban Zepeda**, quien fue el primero en apoyarme y alentarme a ir más allá con esta idea. Fuiste realmente la primera persona que creyó en este proyecto y estuviste allí conmigo en todo el proceso de creación de la idea.
- **Manuel Santelices**, siempre apoyándome como amigo y como un profesional experimentado en muchos puntos críticos para el desarrollo del proyecto. Muchas gracias, Manuel, por estar siempre allí para brindarme tu ayuda, tus conocimientos y tu apoyo emocional.
- **Pablo Veliz**, que no solo me ha ayudado en todo el diseño e imagen de la aplicación, sino que también siempre está atento y trayendo nuevas ideas y diferentes tipos de apoyo que han sido muy importantes para mí. Me hace feliz que hayas estado trabajando con nosotros.



- **Gabriela Arancibia**, mi hermana, que me ayudó no solo a realizar tareas de backend, sino que también me envía información y me educa sobre cómo llevar este proyecto y qué cosas nuevas puedo agregar. Muchas veces me ha retado cuando me desvíó del camino, pero siempre con el entusiasmo y la voluntad por ayudarme de todas las formas posibles, muchas gracias.
- **Sebastián Spataris**, que disfrutó mucho de trabajar con nosotros, siempre fue muy activo y agradecido. Soy yo quien debe agradecerle a él por su trabajo y por toda la ayuda que nos brindó en este proyecto.
- **Cesar Gonzalez**, al cuál le pedí consejos sobre el manejo de personas y no solo me estuvo ayudando activamente en eso, sino que también comenzó a interesarse más por el proyecto, dándonos mucho feedback, ayudandome en cosas varias y aportando con su visión, sus consejos sobre negocios y manejo de proyectos, pensando en el futuro, estoy muy agradecida contigo.

Quiero también agradecer a las personas que me guiaron en el desarrollo de esta memoria, aportándome con sus consejos y correcciones. Sin ellos, el hecho de haber estado tanto tiempo lejos de la universidad me hubiera hecho cometer muchos errores. Muchas gracias a **Camila Retamal**, que estuvo muy atenta y abierta a corregirme, a aconsejarme, a probar la aplicación y contestar todas mis preguntas, por más tontas que fueran, te lo agradezco demasiado, y a **Gonzalo Rojas**, por brindarme también mucho apoyo y aconsejarme en este proceso académico, estoy muy agradecida.

Agradezco a mis compañeros de trabajo que me ayudaron a complementar la información y se tomaron el tiempo de explicarme cosas que en ese momento no estaban tan claras para mí. Muchas gracias a **Pablo Carreño**, quien me ayudó a entender cómo era el proceso de subir imágenes a la nube, a **Rodrigo Navarro**, por su feedback sobre el manejo de microservicios, el uso de Docker y la subida a AWS, y a **Matías Lucero**, por leer mis escritos, probar la aplicación, darme buen feedback y mucho apoyo en el trabajo cuando estaba cansada o estresada por el proyecto. Gracias a **Juan Camilo Sandoval**, mi líder, por leer mis escritos, darme buen feedback y ofrecerme soporte emocional y un apoyo importante en el trabajo, dándome espacios y tiempos



cruciales para cumplir con los plazos de entrega y tener momentos de descanso.

En general, agradezco a todas las personas que probaron la aplicación y mostraron interés en todo lo que estábamos haciendo. Les agradezco un montón a todos.

Este proyecto es bastante ambicioso, por lo que no lo hubiera podido lograr sola, y me alegra haber contado con cada uno de ustedes. No puedo estar más agradecida con la gente a mi alrededor.



DESARROLLO DE APLICACIÓN MÓVIL PARA EL REGISTRO MÉDICO DE MASCOTAS

Pilar Gabriela Arancibia Echeverría

Memoria para optar al título de Ingeniero en Ejecución Eléctronica, Mención Computadores

Universidad Técnica Federico Santa María

Profesor Guía: Agustín Gonzalez

Julio 2024

Resumen

Actualmente, los registros médicos de las mascotas se llevan en papel y se almacenan en las clínicas veterinarias, lo que genera inconvenientes para los tutores, tales como la necesidad de transportar físicamente las fichas y la posibilidad de pérdida o daño de la información. Sin acceso a estos datos, los dueños pueden desconocer vacunas necesarias y el historial de enfermedades, llevando a diagnósticos incorrectos o tratamientos inadecuados.

Para resolver estos problemas, el proyecto propone desarrollar una aplicación móvil denominada Digipaw, que permita a los usuarios mantener un registro digital y centralizado de la información médica de sus mascotas. Esta digitalización y el almacenamiento seguro de la información médica permiten a los tutores acceder a los datos de sus mascotas en cualquier momento y lugar, mejorando la gestión de la salud de las mascotas, reduciendo la pérdida de información y facilitando recordatorios, estadísticas y recomendaciones especializadas. La aplicación está diseñada para garantizar la persistencia de los registros en el tiempo y asegurar su mantenibilidad, estabilidad y escalabilidad, permitiendo futuras funcionalidades como adopción, localización de servicios, tiendas y redes sociales.

Las pruebas realizadas con usuarios mostraron una recepción positiva hacia la aplicación en cuanto a su funcionalidad básica, aunque también se identificaron aspectos importantes para mejorar la usabilidad y ampliar las funcionalidades. A nivel técnico, la aplicación se desarrolló



con una arquitectura escalable, lo que permite integrar nuevas características de manera sencilla y flexible. El proyecto fue llevado a cabo por un equipo multidisciplinario, donde la colaboración y la distribución eficiente de tareas fueron clave para avanzar en los plazos establecidos. La metodología ágil aplicada resultó fundamental para gestionar el trabajo en equipo, permitiendo un seguimiento constante del progreso y facilitando la adaptación rápida a cambios y mejoras durante el desarrollo. Estos resultados indican que, aunque la aplicación cumple con su propósito inicial, existe un camino claro para optimizarla y expandirla en futuras iteraciones.

Palabras Claves: Aplicación Móvil, Digitalización de datos médicos, Arquitectura escalable, Gestión de datos en la nube.



DEVELOPMENT OF A MOBILE APPLICATION FOR TRACKING PET MEDICAL INFORMATION

Pilar Gabriela Arancibia Echeverría

Final project report towards the fulfillment of the Electronic Engineer,
minor in computer degree

Universidad Técnica Federico Santa María

Advisor: Agustín Gonzalez

July 2024

Abstract

Currently, pet medical records are kept on paper and stored in veterinary clinics, which causes inconveniences for pet owners, such as the need to physically transport the files and the risk of information loss or damage. Without access to this data, owners may be unaware of necessary vaccinations and medical history, leading to incorrect diagnoses or inadequate treatments.

To address these issues, the project proposes developing a mobile application called Digipaw, which allows users to maintain a digital and centralized record of their pets' medical information. This digitalization and secure storage of medical data enable pet owners to access their pets' information anytime and anywhere, improving pet health management, reducing information loss, and facilitating reminders, statistics, and specialized recommendations. The application is designed to ensure the long-term persistence of records and to guarantee its maintainability, stability, and scalability, allowing for future functionalities such as adoption, service localization, stores, and social networks.

User testing showed a positive reception towards the application regarding its basic functionality, although important aspects for improving usability and expanding features were also identified. Technically, the application was developed with a scalable architecture, enabling the



seamless and flexible integration of new features. The project was carried out by a multidisciplinary team, where collaboration and efficient task distribution were key to meeting established deadlines. The agile methodology applied was fundamental in managing teamwork, allowing constant monitoring of progress and facilitating quick adaptation to changes and improvements during development. These results indicate that, although the application meets its initial purpose, there is a clear path to optimize and expand it in future iterations.

Keywords: Mobile Application, Medical Data Digitalization, Scalable Architecture, Cloud Data Management.



Índice general

1. Introducción	1
1.1. Problema	1
1.2. Objetivos	2
1.3. Estructura General del Documento	3
2. Organización del equipo y planificación	5
2.1. Equipo	5
2.2. Roadmap	7
2.3. Documentación	8
2.4. Asignación y Visibilidad de Tareas	11
2.5. Comunicación del Equipo	15
3. Aplicaciones conectadas a la nube	19
3.1. Estado del arte	19
3.1.1. Base de datos	19
3.1.2. Backend	20
3.1.3. Frontend	21
3.1.4. Estrategia de comunicación frontend-backend	22
3.1.5. Arquitectura de la aplicación	23
3.1.6. Pruebas de software	24
3.2. Selección de Alternativa	25
3.2.1. Criterios de Selección	25
3.2.2. Elección de la base de datos	25
3.2.3. Elección del framework para backend	25
3.2.4. Elección de usar nativo sobre híbrido	26
3.2.5. Estrategia de comunicación frontend-backend	26



3.2.6.	Arquitectura de la aplicación Android	26
3.2.7.	Otras consideraciones	27
4.	Diseño de la interfaz gráfica y experiencia de usuario de Digipaw	28
4.1.	Levantamiento de requerimientos	28
4.2.	Materialización de la Idea	29
4.3.	Creación de Mockups	30
4.4.	Flujos de la aplicación	31
4.4.1.	Autenticación de Usuario	31
4.4.2.	Registro de usuario	32
4.4.3.	Listado de Mascotas (Home)	35
4.4.4.	Agregar Mascota	37
4.4.5.	Perfil de Mascota	42
4.4.6.	Historial de Vacunas	43
4.4.7.	Agregar Vacunas	47
5.	Desarrollo del Backend de Digipaw	50
5.1.	Base de datos	50
5.1.1.	MongoDB	52
5.2.	Backend	57
5.2.1.	Estructura de una llamada de red	57
5.2.2.	Sincronización entre Frontend y Backend	59
5.2.3.	Spring Boot	61
5.2.4.	Microservicios	62
5.3.	Arquitectura del Backend	63
5.3.1.	Servicio Auth	63
5.3.2.	Servicio Pet	66
5.3.3.	Servicio Health	67
5.3.4.	Servicio Middleend	68
5.3.5.	Gateway	70



6. Desarrollo del Frontend de Digipaw	73
6.1. Antes de Desarrollar	73
6.2. Preparación base de la aplicación móvil	77
6.3. Construcción de un flujo	89
7. Publicación a Producción	97
7.1. Creación de un Bucket S3 en Amazon Web Services	97
7.2. Implementación de Servidores en Amazon EC2	101
7.3. Implementación de Docker Compose	106
7.4. Publicación en Google Play Store	109
8. Pruebas de Digipaw	115
8.1. Estrategia de Pruebas de Código	115
8.1.1. Pruebas Unitarias	115
8.1.2. Pruebas End-to-End (e2e)	118
8.2. Pruebas con usuarios y resultados	121
8.2.1. Distribución para Pruebas de Desarrollo Internas	121
8.2.2. Preguntas	123
8.2.3. Resultados	125
9. Conclusiones	134
9.1. Trabajo en equipo y liderazgo	134
9.2. Seguridad y vulnerabilidades	135
9.3. Escalabilidad y arquitectura	135
9.4. Uso de microservicios	136
9.5. Elección de herramientas y lenguajes en proyectos pequeños	137
9.6. Pruebas y calidad del producto	138
9.7. Pruebas con usuarios	138
9.8. Comentarios finales	138
Referencias Bibliográficas	139



Anexos	142
.1. Excel con las respuestas de pruebas de usuarios.	143
.2. Página de Figma con MockUps de ejemplo.	143
.3. Repositorio de aplicación Android de ejemplo.	143
.4. Repositorio de servicio de Autenticación de ejemplo.	143
.5. Repositorio de servicio de Mascotas de ejemplo.	143
.6. Repositorio de servicio de Middleend de ejemplo.	144



Índice de figuras

2.1. Distribución de Cargos en el Proyecto	5
2.2. Organización de las páginas en Confluence para el proyecto Digipaw	9
2.3. Ejemplo de un contrato documentado en Confluence	10
2.4. Ejemplo de backlog en Jira	11
2.5. Tablero de sprint en Jira	12
2.6. Monitoreo de épicas en Jira	13
2.7. Ejemplo de una tarea asignada y descrita en Jira	14
2.8. Canales en Slack para la organización del equipo	16
2.9. Canal de revisión de PRs en Slack	17
3.1. Diagrama general de la estructura de un sistema de software en la nube.	20
4.1. Dibujos iniciales de los flujos de la aplicación	29
4.2. Ejemplo de componentes en figma	31
4.3. Pantalla de autenticación de usuario en Figma	32
4.4. Flujo de registro de usuario en Figma	33
4.5. Error de correo electrónico ya registrado.	34
4.6. Error de contraseñas no coincidentes.	34
4.7. Diseño ideal de la pantalla del home con la lista de mascotas.	35
4.8. Estado actual del home con la lista de mascotas.	36
4.9. Idea de pantalla para cuando el usuario no tiene mascotas.	36
4.10. Pantalla de selección de foto para la mascota	37
4.11. Pantalla para ingresar el nombre y seleccionar la especie	38
4.12. Uso de componente con iconografía, para elegir el sexo de la mascota	38
4.13. Ingresar la fecha de nacimiento de la mascota.	39
4.14. Despliegue de opciones para seleccionar la raza de la mascota	39
4.15. Agregar descripción de la mascota.	40



4.16. Pantalla de éxito al agregar una mascota	41
4.17. Mensajes de error si los campos obligatorios no son completados	41
4.18. Pantalla de perfil de mascota	42
4.19. Acceso al historial de vacunas desde el perfil de la mascota.	44
4.20. Información del esquema de vacunación.	45
4.21. Ampliación de la imagen del sticker de vacuna.	46
4.22. Historial completo de vacunas para un esquema específico.	47
4.23. Flujo de agregar vacuna. parte 1	48
4.24. Flujo de agregar vacuna. parte 2	49
5.1. Diagrama de la Base de Datos	51
5.2. Colecciones de la base de datos creadas en MongoDB Compass	54
5.3. MongoDB Compass, para visualizar y administrar la base de datos	56
5.4. Diagrama del uso de contratos entre el Backend y el Frontend	60
5.5. Diagrama de secuencia del servicio /validate-token	65
5.6. Diagrama de la comunicación entre el MiddleEnd y los diferentes servicios de .	70
5.7. Diagrama de la arquitectura de servicios con Gateway	71
6.1. Ejemplo de algunos componentes reutilizables de la aplicación.	74
6.2. Ejemplo de casos de error de falta de datos en validación.	76
6.3. Estructura de módulos de Gradle en la aplicación.	78
6.4. Patrón MVI.	85
6.5. Visualización del gráfico de navegación hecho en xml en Android Studio. . . .	89
6.6. Estructura de paquetes de la capa de data.	90
6.7. Estructura de paquetes de la capa de data, mostrando todas las clases.	91
6.8. Estructura de paquetes de la capa de presentación, mostrando todas las clases. .	94
7.1. Visualización del bucket digipaw-images con las carpetas user-profile, pet-profile y health.	98
7.2. Archivos de imágenes dentro de la carpeta pet-profile.	99
7.3. Instancia Digipaw-Dev en ejecución en Amazon EC2.	103
7.4. Instrucciones para conectarse a la instancia EC2 en AWS.	104



7.5. Consola de la instancia EC2 mostrando el uso de recursos y la estructura de directorios.	105
7.6. Subida de un App Bundle en Google Play Console.	112
7.7. Opciones de pruebas en Google Play Console.	113
7.8. Formularios requeridos en Google Play Console.	113
8.1. Interfaz de Insomnia mostrando la configuración y respuesta de un endpoint. . .	118
8.2. Visualización de datos en MongoDB Compass.	119
8.3. Interfaz de Flipper mostrando las llamadas de red y sus respuestas.	120
8.4. Grupo de WhatsApp utilizado para la distribución del APK y comunicación con los testers.	122
8.5. Distribución de edades de los participantes.	125
8.6. Número de mascotas por participante.	126
8.7. Facilidad para crear una cuenta de usuario.	126
8.8. Facilidad para agregar una nueva mascota.	127
8.9. Facilidad para agregar una nueva vacuna.	127
8.10. Utilidad del historial de vacunas.	128
8.11. Utilidad de la información mostrada sobre las mascotas.	128
8.12. Utilidad de la aplicación para la gestión de vacunas.	129
8.13. Recurrencia de uso de la aplicación para gestionar vacunas.	130
8.14. Probabilidad de seguir usando la aplicación en el futuro.	130



Índice de cuadros



Capítulo 1

Introducción

El proyecto de titulación *Desarrollo de aplicación móvil escalable para el registro y seguimiento de la información médica de mascotas en sistemas operativos Android*, se enfoca en solucionar la problemática de la gestión de registros médicos de mascotas. Actualmente, estos registros se mantienen en papel, lo que genera múltiples inconvenientes y riesgos para la salud de los animales. La propuesta es desarrollar una aplicación móvil que permita digitalizar y centralizar esta información, mejorando la accesibilidad y fiabilidad de los datos médicos. Este proyecto no solo busca optimizar la gestión de salud de las mascotas, otorgándole a los usuarios facilidad en el control de sus datos y agendas, sino también proporcionar una plataforma escalable y mantenible que pueda incorporar futuras funcionalidades.

1.1. Problema

La gestión de los registros médicos de las mascotas en formato papel presenta varios problemas significativos. Entre ellos, la necesidad de transportar físicamente las fichas médicas de una clínica a otra, la probabilidad de extravío o daño de los documentos, y la dificultad de acceso a la información cuando es necesaria. Estas situaciones pueden llevar a diagnósticos incorrectos o tratamientos inadecuados, ya que los dueños pueden desconocer las vacunas necesarias o el historial de enfermedades de sus mascotas. La pérdida de registros médicos impide un seguimiento adecuado de la salud del animal, ocasionando retrasos en la atención médica y potencialmente agravando condiciones de salud, lo que a su vez incrementa los costos y reduce la efectividad del tratamiento.

Según un estudio realizado por GfK, el 73% de los hogares en Chile tiene al menos una mascota, y esta cifra ha ido en aumento en los últimos años [12]. A nivel mundial, la tendencia es similar, con un creciente reconocimiento de las mascotas como *nuevos hijos*. Un informe de Grand View Research estima que el mercado global de cuidado de mascotas alcanzará los 202.6



mil millones de dólares para 2025, reflejando un crecimiento anual compuesto del 5.4% [13]. Estos datos respaldan la relevancia y oportunidad de desarrollar una aplicación que facilite la gestión de la salud de las mascotas.

1.2. Objetivos

Objetivo General

Desarrollar una aplicación móvil para sistemas operativos Android, denominada **Digipaw**, que permita a los usuarios mantener un registro digital y centralizado de la información médica de sus mascotas. Para ello, también es necesario construir una red de servidores que funcionen en la nube para que la aplicación pueda persistir datos y ser utilizada desde cualquier lugar.

Objetivos Específicos

- Diseñar la experiencia del usuario (UX) y crear MockUps detallados para asegurar una interfaz intuitiva y atractiva¹.
- Garantizar la persistencia en el tiempo de los registros médicos digitales, evitando su pérdida como ocurre con los registros en papel.
- Diseñar y desarrollar la aplicación de manera que sea mantenible, estable y escalable, asegurando la capacidad de soportar nuevas funcionalidades en el futuro.
- Implementar funcionalidades que permitan a los usuarios acceder fácilmente al historial médico y al control de cada una de sus mascotas.
- Desarrollar servidores utilizando una arquitectura de microservicios para mejorar la escalabilidad y mantenibilidad del sistema.
- Implementar la infraestructura necesaria para alojar la aplicación en Amazon Web Services (AWS), asegurando su disponibilidad y rendimiento.

¹Un MockUp es una representación visual detallada de la interfaz de usuario de una aplicación. Incluye colores, tipografías, íconos y otros elementos visuales, proporcionando una visualización clara de cómo se verá la aplicación final.



1.3. Estructura General del Documento

Capítulo 1: Introducción

Se exponen los problemas y objetivos del proyecto.

Capítulo 2: Organización del equipo y planificación

Se explica la metodología utilizada, incluyendo la composición del equipo, el roadmap del proyecto, la documentación, la comunicación y las decisiones de deadline y trade-offs.

Capítulo 3: Aplicaciones conectadas a la nube

Se explora la tecnología y arquitectura utilizada en la actualidad para el desarrollo de la aplicación y su conexión en la nube. Revisando puntos como las bases de datos, backend, frontend y estrategias de comunicación.

Capítulo 4: Diseño de la interfaz gráfica de Digipaw

Se detalla el proceso de la creación y diseño de la idea hasta su concepción gráfica, que se usará para crear la aplicación. La importancia de que el diseño de la aplicación sea intuitiva y fácil de usar. Incluye el uso de herramientas de diseño y detalles de cada uno de los flujos de la aplicación.

Capítulo 5: Desarrollo del Backend de Digipaw

Se describe la implementación técnica de la aplicación, desde el modelado de la base de datos hasta la creación del backend con microservicios.

Capítulo 6: Desarrollo del Frontend de Digipaw

Se describe la implementación técnica de la aplicación móvil.



Capítulo 7: Publicación a producción

Se detalla el proceso de despliegue de la aplicación en producción, utilizando herramientas como Docker Compose, AWS EC2 y Amazon S3. También se describe la publicación en Google Play Store.

Capítulo 8: Pruebas de Digipaw

Se presentan las estrategias de pruebas de código, incluyendo pruebas unitarias y end-to-end, para asegurar la calidad y el correcto funcionamiento del software, además de incluir el periodo de pruebas con usuarios y sus resultados.

Capítulo 9: Conclusiones

Se resumen los logros y dificultades del proyecto, ofreciendo reflexiones sobre posibles mejoras y futuras funcionalidades.

Capítulo 2

Organización del equipo y planificación

Antes de iniciar el desarrollo del software, es fundamental planificar las tareas a realizar, ya que esto permite definir el alcance del proyecto, los recursos necesarios y evaluar si será necesario incorporar más personas para alcanzar los objetivos propuestos.

2.1. Equipo

El proyecto estaba originalmente planificado para completarse en un plazo de seis meses. Sin embargo, no es posible obtener ese tiempo para el desarrollo. Por lo tanto, el trabajo debe finalizarse en tres meses. Esta reducción en el tiempo disponible ha requerido la formación de un equipo de apoyo para paralelizar tareas y cumplir con las fechas establecidas, esta formación está representada en el diagrama de la figura 2.1.

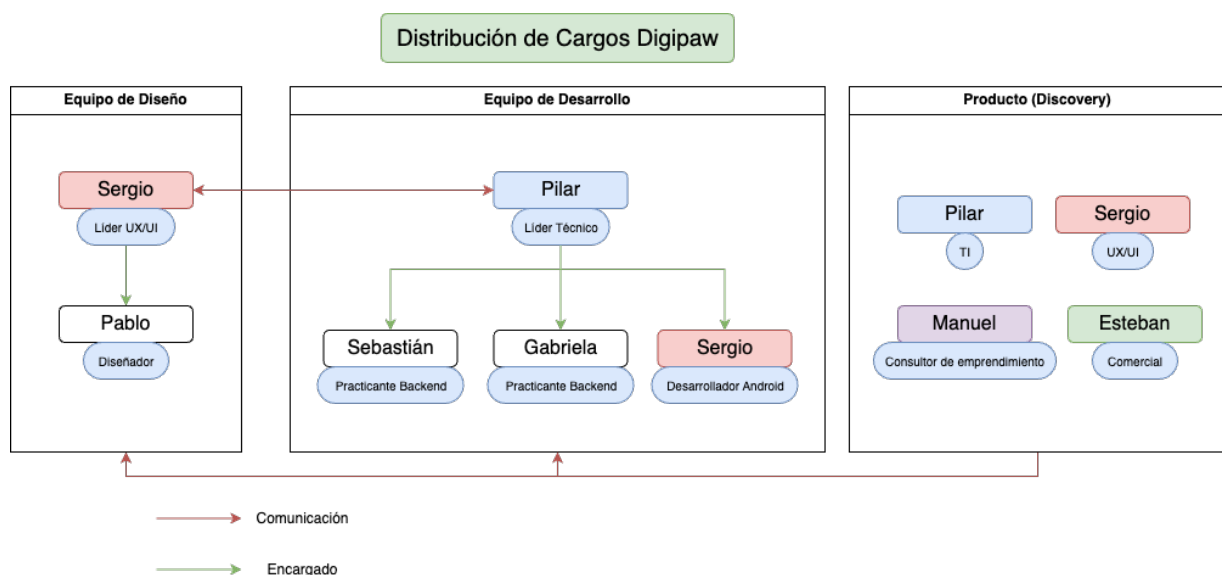


Figura 2.1: Distribución de Cargos en el Proyecto



2.1.0.1. Equipo de Desarrollo

El equipo de desarrollo es responsable de la creación del software, y es donde se concentra la mayor parte del trabajo. Como se ha ido viendo en este escrito, el trabajo de desarrollo se divide, principalmente, entre backend y frontend.

Frontend Sergio Olmos, un desarrollador semi-senior en Android, forma parte del equipo de frontend. **Sergio** ya cuenta con una experiencia de 2 años y medio en el rubro, por lo que su experiencia y conocimiento han sido invaluable para avanzar a buen paso con la aplicación.

Backend El equipo de backend incluye a **Gabriela Arancibia** y **Sebastián Spataris**. **Gabriela**, en su segundo año de técnico informático, ha estado siendo motivada y asistida en su desarrollo profesional. **Sebastián**, un profesional recién graduado de técnico informático, busca especializarse en el backend. Ambos han contribuido significativamente al proyecto, bajo supervisión y guía.

Para organizar el trabajo, se utilizó GIT junto con el sistema de branching de gitflow. Este sistema implica trabajar en ramas separadas para cada nueva característica o corrección de errores, las cuales se integran en la rama principal solo después de la aprobación de un *pull request*. Un *pull request* es una solicitud para revisar y fusionar cambios de una rama a otra, asegurando que el código sea revisado y validado antes de integrarse en la rama principal.

2.1.0.2. Equipo de Producto

El equipo de producto ha sido útil para definir la idea y explorar su potencial de mercado. Este equipo incluye a personas con diversas habilidades y experiencias relevantes que han brindado su ayuda para guiar el proyecto.

Manuel Santelices, con experiencia en emprendimientos y lanzamiento de productos, ha proporcionado asesoramiento sobre cómo llevar la idea a producción, cómo postular a fondos y en temas de seguridad informática.

Esteban Zepeda, ingeniero comercial, ha ayudado en la definición de estrategias de monetización.



Sergio Olmos ha contribuido tanto en el desarrollo de software como en el diseño, aportando una perspectiva centrada en la experiencia de usuario.

Finalmente, **Pilar Arancibia**, creadora y líder de la idea, cuenta con experiencia en desarrollo de software empresarial y sabe cómo construir un producto de principio a fin. Como autora de esta memoria, es la persona más motivada con la idea y lleva la batuta del proyecto.

Este equipo ha proporcionado soporte adicional, especialmente en la fase inicial de conceptualización y durante la refinación de la idea, lo cual podría ser de gran ayuda en el caso de que se decida continuar con esta idea en el futuro.

En resumen, el equipo se ha estructurado de manera que permite una paralelización efectiva de tareas, con roles y responsabilidades claras. Esto ha sido vital para avanzar rápidamente y cumplir con los plazos ajustados del proyecto.

2.2. Roadmap

Un roadmap es una estrategia planificada que describe las etapas y plazos para completar un proyecto. Es esencial para asegurar que el proyecto avance de manera organizada y que se cumplan los plazos establecidos.

En este proyecto, primero se definen cada uno de los puntos enviados en la carta Gantt y se determinan las entregas mensuales. Por ejemplo, si una entrega mensual estaba pensada para entregar el perfil de mascota, ahora, debido a los ajustes en los plazos, tendría que incluir tanto el perfil de mascota como la agregación de mascota. Esto significa más trabajo en el mismo tiempo.

En este contexto, se hace evidente la necesidad de paralelizar tareas y hacer compromisos estratégicos, conocidos como *trade-offs*. Un trade-off es una decisión que implica renunciar a ciertos aspectos del proyecto en favor de otros más críticos, con el fin de cumplir con los plazos y recursos disponibles.

Se desglosan cada una de las tareas tanto del frontend como del backend. Se escribe todo en



una pizarra y se dividen cada una de estas tareas en conceptos específicos. Por ejemplo, todas las tareas relacionadas con el perfil de mascota se dividen en backend y Android, al igual que las tareas de agregación de mascota.

Una vez hecho esto, se pueden identificar qué tareas se pueden realizar en paralelo y asignar las tareas a las personas adecuadas. Este proceso permite tener una visión clara del tiempo necesario para completar las tareas, considerando la dificultad de cada tarea, el tiempo disponible y la capacidad de la persona asignada.

A partir de esta evaluación, se decide cuáles serán los trade-offs necesarios. Se simplifica la aplicación, dejando solo las funcionalidades críticas para probar su funcionalidad y eliminando partes del desarrollo que, aunque importantes, no son cruciales para una entrega. Por ejemplo, la capacidad de editar el perfil de mascota, eliminar una mascota, manejar casos de error de red y gestionar situaciones en las que un usuario no tenga mascotas son postergadas. Se concentra en asegurar que lo esencial esté completado y que la aplicación sea funcional y utilizable para el propósito esperado.

Este enfoque estratégico y organizado es esencial para garantizar que el proyecto avance según lo planeado y se cumplan los plazos ajustados.

2.3. Documentación

Para gestionar y centralizar toda la documentación necesaria durante el desarrollo del proyecto, se utiliza Confluence, una herramienta de Atlassian ampliamente usada en empresas de TI. Confluence permite almacenar, organizar y compartir documentos, facilitando la colaboración y la gestión del conocimiento entre los miembros del equipo.

En Confluence, se documentan tanto las ideas como los aspectos técnicos del desarrollo. Esta plataforma se convierte en el repositorio central de toda la información relevante, asegurando que



todos los integrantes del equipo tengan acceso a la documentación actualizada.

- > Producto
- ▼ Desarrollo
 - ▼ Backend
 - ▼ Base de datos
 - Propuesta de modelo de vacunas
 - Actualizar datos en MongoDB
 - ▼ Contratos
 - **Perfil Mascota**
 - Añadir Mascota
 - Añadir Vacuna
 - Historial de Esquemas de vacunas
 - Historial de vacuna
 - Android

Figura 2.2: Organización de las páginas en Confluence para el proyecto Digipaw

Una de las principales utilidades de Confluence en este proyecto es la documentación de los contratos entre el backend y el frontend.

Historial de Esquemas de vacunas



Propiedad de Sergio Olmos ...

Última actualización: jun 13, 2024 • Ver cuántas personas han visto esta página

@GET("/health/vaccine/record/{petId}")

```
1  [
2    {
3      "id": "TF",
4      "name": "Triple Felina",
5      "lastApplication": "Última aplicación: 20/03/2024"
6      "description": "Esta es una vacuna que se vacuna cuando vacunen. Es buena para
7      "diseases": "Rinotraqueitis Felina, Calicivirus Felino, Panleucopenia Felina, C
8      "statusLabel": "Próxima aplicación: 20/04/2025",
9      "actualDose": "2/3",
10     "sticker": "http://asjdaisjd.asdk.jpg",
11     "status": 0, //0 ok-verde, 1 cerca-amarillo, 2 pasado-rojo
12     "advice": "Incluida dentro del plan de vacunación recomendado", //este campo lle
13     "brand": "Felocell4 - Zoetis"
14   }
15 ]
```

Figura 2.3: Ejemplo de un contrato documentado en Confluence

En la figura 2.2, se puede observar la estructura de las páginas en Confluence para el proyecto Digipaw. En la sección de desarrollo, se encuentran documentadas las diferentes áreas del backend y del frontend, incluyendo propuestas de modelos, actualizaciones en MongoDB, y detalles específicos de cada funcionalidad.

La figura 2.3 muestra un ejemplo de un contrato documentado en Confluence. Este contrato especifica la estructura de un endpoint del backend, detallando los campos y sus valores esperados. Este tipo de documentación es fundamental para asegurar que ambos equipos, de backend y frontend, estén alineados y puedan trabajar de manera paralela y coordinada.

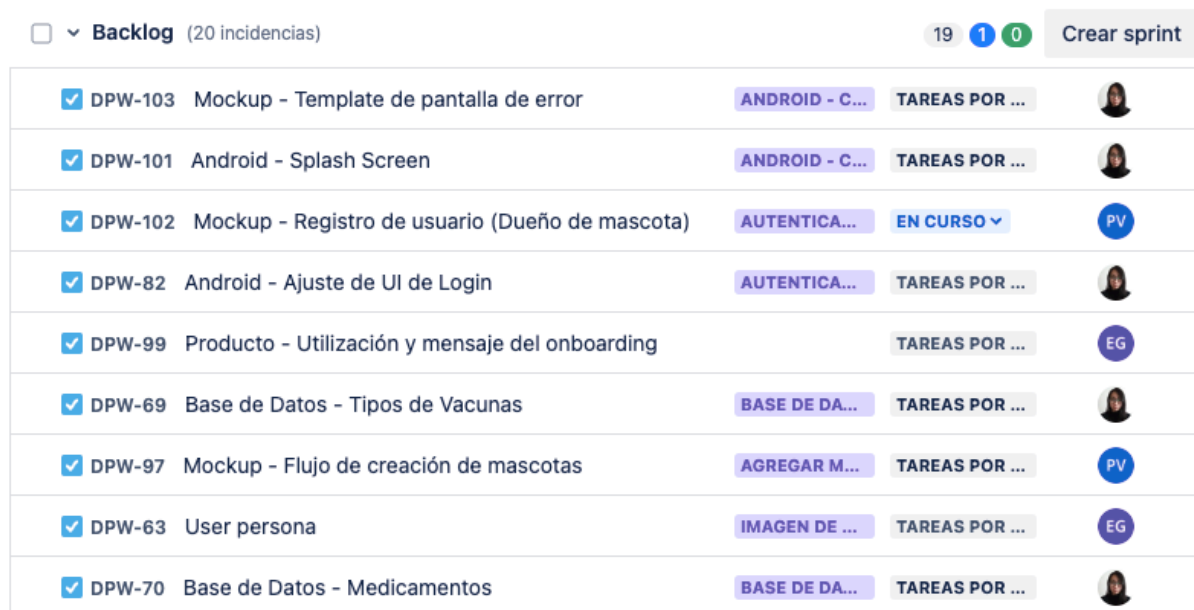
El uso de Confluence facilita la colaboración y mejora la eficiencia del equipo, permitiendo acceder a la documentación de manera centralizada y actualizada.

2.4. Asignación y Visibilidad de Tareas

Para gestionar y visibilizar las tareas del proyecto, se utiliza Jira [23] de Atlassian. Jira es una herramienta de gestión de proyectos ampliamente utilizada en la industria del software para planificar, rastrear y gestionar el desarrollo de proyectos. Esta herramienta facilita la organización de tareas, la asignación de responsabilidades y el seguimiento del progreso del proyecto.

En este proyecto, se adopta un enfoque de metodología ágil, específicamente Kanban. Kanban es un método de gestión de proyectos que utiliza un tablero visual para gestionar el flujo de trabajo, destacando las tareas en curso, las tareas pendientes y las tareas completadas. Este enfoque ayuda a identificar cuellos de botella y mejorar la eficiencia del equipo.

Todas las tareas definidas en el roadmap se agregan al backlog de Jira. El backlog es una lista priorizada de tareas que el equipo debe completar. Cada tarea se describe detalladamente para asegurar un entendimiento claro entre la persona que crea la tarea y la persona que la va a realizar. En la figura 2.4, se muestra un ejemplo del backlog en Jira:



La imagen muestra una captura de pantalla de la interfaz de Jira, específicamente la vista del backlog. En la parte superior, se indica 'Backlog (20 incidencias)' con un menú desplegable. A la derecha, hay un resumen de tickets: 19 en total, 1 en progreso (azul) y 0 completados (verde), junto a un botón 'Crear sprint'. La lista de tareas está organizada en una tabla con las siguientes columnas: ID de la tarea, descripción, etiquetas de categoría, estado de la tarea y el responsable asignado.

ID	Descripción	Categoría	Estado	Responsable
DPW-103	Mockup - Template de pantalla de error	ANDROID - C...	TAREAS POR ...	[Avatar]
DPW-101	Android - Splash Screen	ANDROID - C...	TAREAS POR ...	[Avatar]
DPW-102	Mockup - Registro de usuario (Dueño de mascota)	AUTENTICA...	EN CURSO	PV
DPW-82	Android - Ajuste de UI de Login	AUTENTICA...	TAREAS POR ...	[Avatar]
DPW-99	Producto - Utilización y mensaje del onboarding		TAREAS POR ...	EG
DPW-69	Base de Datos - Tipos de Vacunas	BASE DE DA...	TAREAS POR ...	[Avatar]
DPW-97	Mockup - Flujo de creación de mascotas	AGREGAR M...	TAREAS POR ...	PV
DPW-63	User persona	IMAGEN DE ...	TAREAS POR ...	EG
DPW-70	Base de Datos - Medicamentos	BASE DE DA...	TAREAS POR ...	[Avatar]

Figura 2.4: Ejemplo de backlog en Jira

El tablero de sprint en Jira se utiliza para monitorizar el progreso de las tareas en curso. Este

tablero muestra las tareas distribuidas en diferentes columnas que representan los estados de las tareas, como *Por Hacer*, *En Curso*, *En Revisión* y *Listo*. En la figura 2.5, se muestra un ejemplo del tablero de sprint:

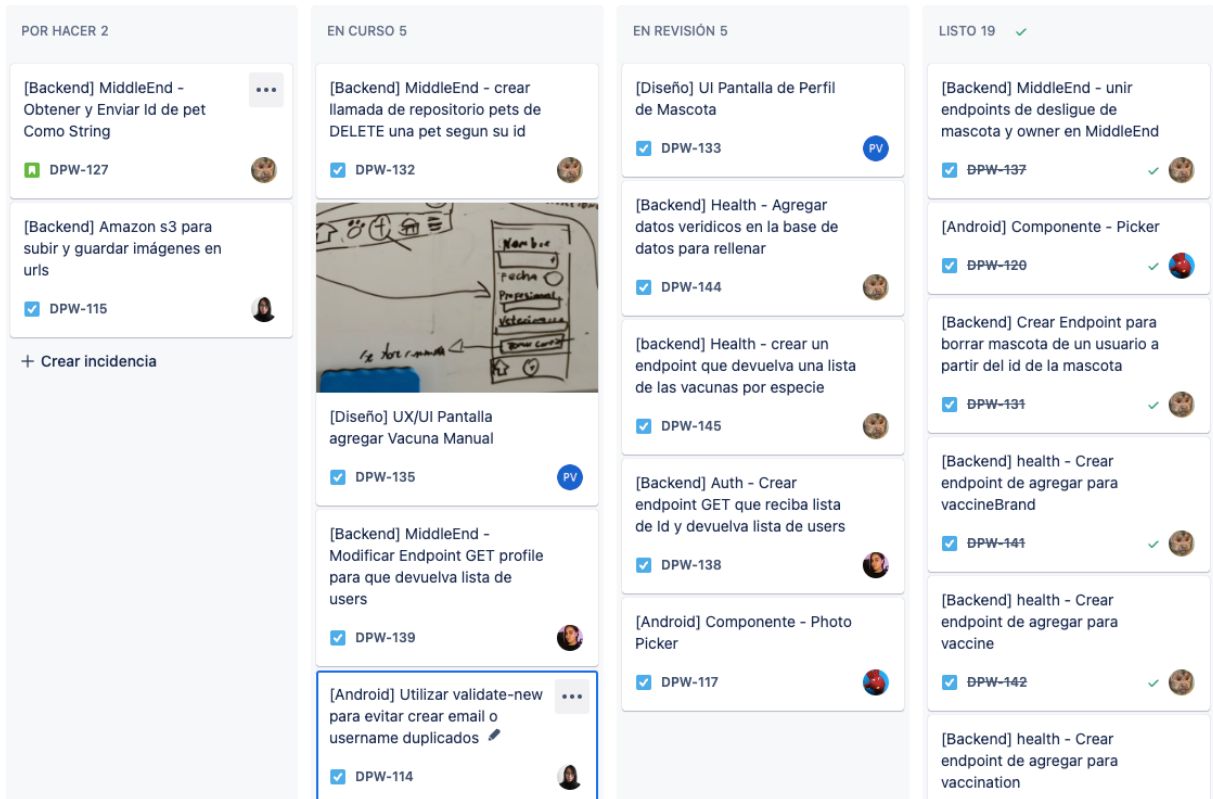


Figura 2.5: Tablero de sprint en Jira

En Jira, también se pueden gestionar y monitorizar épicas. Una épica es una gran unidad de trabajo que se puede descomponer en tareas más pequeñas. Las épicas ayudan a organizar el trabajo y a visualizar el progreso general del proyecto. En la figura 2.6, se muestra un ejemplo del monitoreo de épicas en Jira:



Figura 2.6: Monitoreo de épicas en Jira

Además, cada tarea en Jira se asigna a un miembro del equipo y se describe detalladamente. Esto incluye una descripción de lo que se necesita hacer, los criterios de aceptación y cualquier otra información relevante. En la figura 2.7, se muestra un ejemplo de una tarea asignada y descrita en Jira:



Proyectos / Digipaw / DPW-81 / DPW-112



[Backend] Crear endpoint de validación de username e email



Descripción

Se necesita un **endpoint nuevo** en el servicio de **auth**, al cual se le manden por **query params**, el *email* o *username* a validar.

Esta validación lo que hace es buscar si el parametro ya existe o no en la base de datos.

- Si el parametro tiene un valor existente en la base de datos, la respuesta sera un error 403.
- En el caso de que NO exista en la base de datos, se envia un 200 OK

Ejemplo

<http://localhost/api/auth/validate?email=pili@gmail.com>

<http://localhost/api/auth/validate?username=gabi>

<http://localhost/api/auth/validate?username=gabi&email=pili@gmail.com>

(este caso no debería suceder, pero debemos ser resilientes): En este caso, solo se manda 200 OK si AMBOS NO existen en la base de datos.

Aceptación (Casos de prueba):

- Mandando solo email, que no existe en la bd -> 200
- Mandando solo email, que si existe en la bd -> 403
- Mandando solo username, que no existe en la bd -> 200

Finalizada ▾

✓ Listo

⚡ Acciones ▾

Detalles

Persona asignada



Sebastian Spataris

[Asignarme a mí](#)

Etiquetas

Ninguno

Sprint

[Tablero Sprint 8](#) +1

Versiones corregidas

Ninguno

Informador



Pili Arancibia

Creado 2 de abril de 2024, 14:51

Actualizado 17 de abril de 2024, 13:37

Resuelto 8 de abril de 2024, 15:42

⚙ Configurar

Figura 2.7: Ejemplo de una tarea asignada y descrita en Jira

El uso de Jira y la metodología Kanban permiten una gestión eficiente de las tareas y una clara visibilidad del progreso del proyecto. Esto asegura que todos los miembros del equipo estén alineados y que el proyecto avance de manera organizada.



2.5. Comunicación del Equipo

Durante el proceso de desarrollo del proyecto, la comunicación efectiva fue clave, especialmente considerando que los miembros del equipo se encuentran en diferentes regiones de Chile. Para evitar el uso de WhatsApp y mantener una comunicación más estructurada y profesional, se utiliza Slack [24]. Slack es una herramienta de mensajería y colaboración diseñada específicamente para equipos de trabajo, ampliamente utilizada en empresas de TI. A diferencia de WhatsApp, Slack permite organizar la comunicación en canales temáticos, lo que facilita el seguimiento de conversaciones y la compartición de archivos.

En Slack, se crearon diferentes grupos para organizar la comunicación del equipo, como se muestra en la figura 2.8. Además del canal general, se dividieron los miembros del equipo en grupos específicos como desarrollo, revisión de Pull Requests, diseño y discovery. Esto permitió una comunicación más enfocada y eficiente, donde solo los interesados en un tema particular recibían las notificaciones correspondientes.

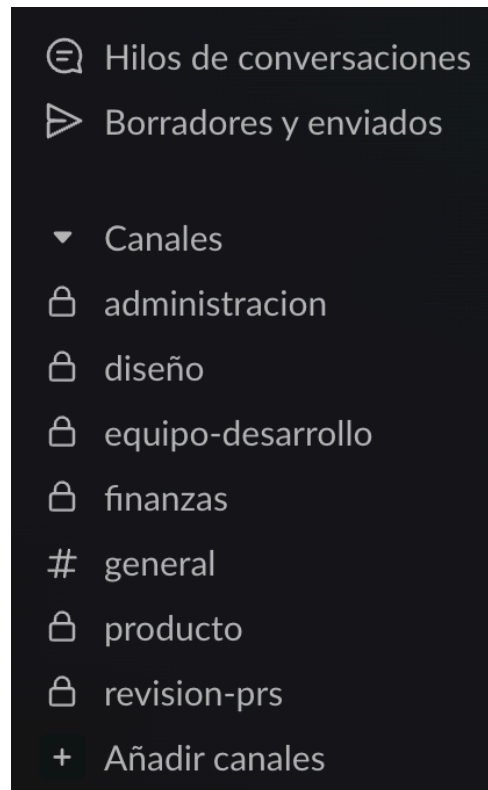


Figura 2.8: Canales en Slack para la organización del equipo

En el canal de desarrollo, se discutían temas generales del desarrollo del proyecto. El canal de revisión de PRs (Pull Requests) se utilizaba para subir PRs y solicitar revisiones, asegurando que las actualizaciones de código fueran revisadas y aprobadas antes de integrarse al código base, como se muestra en la figura 2.9.



Figura 2.9: Canal de revisión de PRs en Slack

El canal de diseño permitía al diseñador mostrar avances y propuestas, facilitando la retroalimentación y las discusiones sobre el diseño de la aplicación. De la misma forma los demás canales abordaban un tema específico y solo se encontraban los miembros del equipo que trabajaban en esos temas.

El uso de Slack mejoró significativamente la comunicación y la fluidez del trabajo, ya que permitía mantener las conversaciones ordenadas y separadas por temas. Esto evitaba que las comunicaciones de trabajo se mezclaran con la vida diaria en WhatsApp, y permitía que los miembros del equipo no interesados en ciertos temas no recibieran notificaciones irrelevantes.

Además, Slack permite compartir código de manera efectiva, utilizando la función de snippets, y facilita el seguimiento de conversaciones a través de hilos. Esta estructura de comunicación es especialmente útil en equipos de desarrollo, donde la claridad y la organización de la



información son cruciales.

La utilización de Slack también fue conveniente debido a que, al trabajar en una empresa que también utiliza Slack, era más fácil cambiar de espacio de trabajo para revisar las actualizaciones del proyecto sin necesidad de revisar WhatsApp.

En resumen, Slack proporcionó una plataforma de comunicación eficiente y organizada, adecuada para las necesidades del equipo de desarrollo del proyecto, lo que ayudó a mejorar la fluidez del trabajo y aumentar la velocidad del equipo en general.



Capítulo 3

Aplicaciones conectadas a la nube

Es necesario evaluar las soluciones actuales utilizadas para resolver problemas similares. Dado que las aplicaciones en la nube no son un concepto nuevo, resulta fundamental analizar las metodologías contemporáneas que se aplican en este ámbito.

3.1. Estado del arte

Es fundamental destacar que, con el fin de centralizar y persistir la información, no se puede utilizar la memoria del teléfono móvil, ya que esta puede eliminarse fácilmente por el usuario o perderse en procesos de limpieza, robo o cambio de dispositivo. Por consiguiente, se requiere una tecnología capaz de compartir y persistir información de manera remota y segura, lo que permitirá el acceso a la información desde cualquier teléfono.

Existen soluciones estandarizadas que se basan en el modelo Cliente-Servidor para abordar estos casos, que consisten en la separación de tres capas de software, como se observa en la figura 3.1, cada una con un rol específico y ejecutándose en dispositivos distintos con conexión a Internet. En este documento se analizará cada una, centrándose especialmente en la capa correspondiente en la que se encuentra la aplicación móvil.

3.1.1. Base de datos

Una base de datos es fundamental para este sistema de software, ya que permiten gestionar grandes cantidades de información y garantizar su integridad y seguridad.

Existen dos tipos principales de bases de datos: **relacionales** y **no relacionales** [1]. Las bases

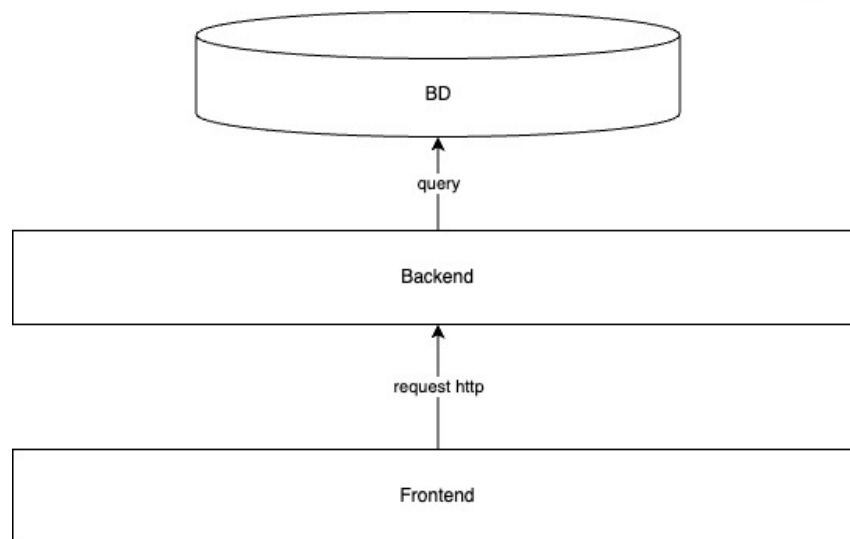


Figura 3.1: Diagrama general de la estructura de un sistema de software en la nube.

de datos relacionales se basan en un modelo de datos que utiliza tablas para almacenar información y establece relaciones entre ellas. Estas bases de datos son muy estructuradas y permiten realizar consultas complejas con facilidad. Por otro lado, las bases de datos no relacionales se basan en modelos de datos que no utilizan tablas ni relaciones entre ellas, sino que se enfocan en almacenar la información de manera más flexible y escalable.

Es importante elegir cuidadosamente el tipo de base de datos que se utilizará en un sistema de software, ya que cada tipo tiene sus ventajas y desventajas en términos de estructura, escalabilidad, rendimiento y capacidad de consulta.

3.1.2. Backend

El backend de un sistema de software es la parte que se encarga de procesar y almacenar los datos del usuario. En otras palabras, es la parte del sistema que se ejecuta en el servidor y que está diseñada para gestionar las operaciones en segundo plano, como la autenticación, el acceso a la base de datos y el procesamiento de la lógica de negocio.

Existen diversas herramientas y lenguajes de programación que se pueden utilizar para construir el backend de un sistema de software. Algunos de los más populares son **Spring Boot**



(Java/Kotlin), **Node.js** (JavaScript) y **.NET** (C#). En este proyecto se debe elegir la herramienta más adecuada para el contexto.

Además de elegir la herramienta de backend adecuada, también es importante considerar la arquitectura del sistema de software. Existen dos enfoques principales en la arquitectura de backend: el monolítico y el de microservicios.

3.1.3. Frontend

El frontend de un sistema de software es la capa que se ejecuta en el lado del cliente y que está diseñada para interactuar con el usuario final. En otras palabras, es la parte del sistema que el usuario ve y utiliza para interactuar con el software.

Existen tres formas para desarrollar una aplicación móvil, estas pueden ser Nativas, Cross-Platform o WebApps [2].

Una forma común de desarrollar una aplicación móvil es mediante el desarrollo **nativo**. Esto significa que la aplicación se construye utilizando herramientas y lenguajes de programación específicos para cada plataforma móvil, como Java/Kotlin para Android, o Swift/Objective-C para iOS.

Otra forma de desarrollar una aplicación móvil es mediante el desarrollo cruzado o *cross-platform*. Esto significa que se construye utilizando herramientas y tecnologías que permiten crear aplicaciones que se ejecuten en múltiples plataformas móviles, como Android y iOS, con un solo código fuente.

También existe la opción de construir una aplicación móvil como una **Progressive Web App (PWA)**, que es una aplicación web que se puede instalar y ejecutar en un dispositivo móvil. Este enfoque permite utilizar tecnologías web estándar para desarrollar la aplicación móvil, lo que puede simplificar el desarrollo y la implementación.



La elección de la forma de desarrollo del frontend adecuada dependerá de las necesidades y requisitos específicos del sistema. Cada enfoque difiere en términos de rendimiento, funcionalidad y experiencia de usuario.

3.1.4. Estrategia de comunicación frontend-backend

Crear una estrategia de comunicación entre el backend y el frontend es importante para el desarrollo de cualquier aplicación o sistema de software. Esto se debe a que una estrategia bien diseñada permite que el sistema funcione de manera más eficiente y escalable, lo que significa que puede manejar más usuarios o datos, o incluso mejorar el rendimiento. Además, una buena estrategia de comunicación también hace que sea más fácil para el equipo de desarrollo mantener el sistema y realizar actualizaciones en el futuro.

Algunas de las estrategias de comunicación, entre backend y frontend, más comunes son:

Endpoints de consulta: el backend proporciona puntos de acceso (*endpoints*) que permiten al frontend consultar la información que necesita. De esta forma, el frontend puede solicitar sólo los datos necesarios y mezclarlos según lo requiera.

Backend for Frontend (BFF)[3]: esta estrategia implica que el backend crea un contrato con el frontend, de manera que éste sólo haga una llamada por vista. Esto permite que muchas cosas puedan cambiarse en la aplicación móvil sin la necesidad de que los usuarios tengan que bajar nuevas versiones, lo que puede ser costoso y repercutir negativamente en la experiencia del usuario.

Server-driven UI (SDUI)[4]: esta estrategia implica que el backend controle mucho más la aplicación móvil y la lógica de visualización. De esta forma, el frontend sólo recibe información y elementos visuales del backend, ahorrándose mucha lógica en el Frontend.

MiddleEndSDK: patrón que se utiliza a manera de adaptar una estrategia más escalable que BFF, pero menos compleja que SDUI. Implica crear un contrato en base a componentes, pero



sin necesidad de que el backend tenga tanto control del frontend.

3.1.5. Arquitectura de la aplicación

La arquitectura de una aplicación se refiere a su estructura general, incluyendo cómo los diferentes componentes se comunican entre sí y cómo se organizan los datos. Una buena arquitectura puede mejorar su eficiencia, escalabilidad, mantenibilidad y flexibilidad [5].

Existen diferentes opciones de arquitectura para el desarrollo de una aplicación, como **Clean Architecture** [7], **Modelo-Vista-Controlador (MVC)**, **Modelo-Vista-Presentador (MVP)**, **Modelo-Vista-ViewModel (MVVM)** o **Modelo-Vista-Intención (MVI)** [9]. Cada una de estas opciones de arquitectura tiene su propio conjunto de fortalezas y debilidades, y es importante elegir la que mejor se adapte a las necesidades del proyecto.

Algunos aspectos importantes a considerar en la elección de una arquitectura incluyen la **estructura de los paquetes**, las **herramientas** y **patrones** como los inyectores de dependencias. Por ejemplo, una buena estructura de paquetes puede facilitar el mantenimiento y la escalabilidad de la aplicación. Los inyectores de dependencias pueden ayudar a simplificar la comunicación entre los diferentes componentes de la aplicación y hacer que el código sea más fácil de mantener y actualizar.

La **modularización** también es parte a considerar y se refiere a la separación de la aplicación en módulos más pequeños y reutilizables. Esto permite una mayor flexibilidad y escalabilidad en el desarrollo de la aplicación, pero requiere una estrategia cuidadosa para definir los módulos y su comunicación, así como una buena organización de los archivos y directorios.

La **estrategia de componentización** es otra opción a considerar en la elección de la arquitectura de una aplicación. La componentización se refiere a la división de la aplicación en componentes independientes y reutilizables, es parte de la modularización, pero orientada a los componentes visuales, lo que permite una mayor flexibilidad en la construcción visual de la apli-



cación.



La componenteización de una aplicación puede ayudar a mejorar la escalabilidad y mantenibilidad, ya que los diferentes componentes pueden ser actualizados o reutilizados sin afectar el funcionamiento de otros componentes. Además, esto puede simplificar el proceso de desarrollo, ya que pueden ser creados y probados por separado antes de ser integrados en la aplicación en su conjunto.

La elección de una buena arquitectura es fundamental para el éxito del proyecto de desarrollo de la aplicación. Una buena arquitectura puede mejorar la eficiencia, escalabilidad, mantenibilidad y flexibilidad de la aplicación, lo que puede ayudar a asegurar su éxito a largo plazo.

3.1.6. Pruebas de software

Por lo general, el proceso de desarrollo de aplicaciones de alta calidad incluye la realización de pruebas de software.

Entre las diferentes opciones de pruebas de software se encuentran las pruebas unitarias, de aceptación, de integración e instrumentación [8]. Es importante elegir qué pruebas serán automatizadas y cuáles necesitan ser realizadas de manera manual. También se debe evaluar la cobertura de las pruebas unitarias, la que indica la cantidad de código que es cubierto por las pruebas y puede indicar la efectividad de las mismas en la detección de problemas.

Además, es importante tener en cuenta la modularización de los diferentes actores de la aplicación, como por ejemplo el networking, y cómo se pueden realizar pruebas específicas en cada uno de ellos. Esto permite una mayor precisión en la detección de problemas y un mayor control en el proceso de pruebas.

Al tener una estrategia de pruebas efectiva y una buena modularización de los diferentes actores de la aplicación, se puede garantizar una mayor calidad y estabilidad en el software final.



3.2. Selección de Alternativa

3.2.1. Criterios de Selección

Los criterios de selección utilizados para evaluar las alternativas son los siguientes:

- **Escalabilidad:** Capacidad de adaptar y extender el código de manera eficiente a medida que crece el proyecto o se enfrenta a mayores demandas.
- **Simplicidad:** Importancia de la simplicidad en un proyecto llevado a cabo por una sola persona con un tiempo de entrega ajustado.
- **Rendimiento:** Eficiencia y rapidez con la que un sistema ejecuta tareas, influenciando la experiencia del usuario y la productividad del desarrollo.
- **Flexibilidad:** Capacidad de la alternativa de adaptarse a cambios en el producto o en los aspectos técnicos.
- **Madurez:** Estabilidad, comunidad sólida y suficiente documentación de la tecnología o herramienta utilizada.

3.2.2. Elección de la base de datos

Para el almacenamiento de datos, se evaluaron bases de datos relacionales y no relacionales. Las bases de datos relacionales, como MySQL y PostgreSQL, son conocidas por su estructura bien definida, integridad referencial y uso de SQL. Sin embargo, presentan limitaciones en escalabilidad horizontal y flexibilidad del esquema. Por otro lado, las bases de datos no relacionales, como MongoDB, ofrecen alta escalabilidad horizontal, flexibilidad del esquema y rendimiento optimizado para grandes volúmenes de datos, además de una destacada facilidad de uso, lo que las hace más adecuadas para este proyecto.

3.2.3. Elección del framework para backend

Para el desarrollo del backend, se consideraron varias opciones, incluyendo Node.js, Spring Boot, Ktor y .NET. Node.js es popular por su modelo de entrada y salida (E/S) sin bloqueo y su



ecosistema de módulos, aunque presenta desventajas con su tipado débil. Spring Boot, por otro lado, destaca por su escalabilidad, rendimiento y robustez en aplicaciones empresariales, además de permitir el uso de Java o Kotlin, lenguajes también utilizados en el frontend. Finalmente, se opta por Spring Boot debido a su alta escalabilidad y la ventaja de usar un solo lenguaje en todo el proyecto.

3.2.4. Elección de usar nativo sobre híbrido

En el desarrollo del frontend móvil, se evaluó entre desarrollar de manera nativa o híbrida. Las aplicaciones nativas, desarrolladas en Kotlin para Android, ofrecen mejor rendimiento, acceso a todas las funcionalidades del dispositivo y una experiencia de usuario superior. Las aplicaciones híbridas, aunque permiten reutilizar código entre plataformas, suelen sacrificar rendimiento y experiencia de usuario. Por estas razones, se decidió desarrollar la aplicación de manera nativa en Kotlin para aprovechar todas las capacidades del dispositivo y proporcionar una mejor experiencia al usuario.

3.2.5. Estrategia de comunicación frontend-backend

La comunicación entre el frontend y el backend se abordará utilizando el patrón BFF, utilizado en MiddleEnd. Este enfoque permite una optimización específica para las necesidades del frontend, mejorando el rendimiento y la flexibilidad. Facilita la integración y manejo de las llamadas y los datos necesarios para la interfaz de usuario, adaptándose a los cambios de requisitos con mayor facilidad.

3.2.6. Arquitectura de la aplicación Android

Para la arquitectura de la aplicación móvil se evaluaron varias opciones: MVC, MVP, MVVM, MVI y Clean Architecture. MVI se destaca por su unidireccionalidad del flujo de datos y la gestión del estado inmutable, lo que facilita la escalabilidad y la comprensión del código. Además, Clean Architecture promueve una estructura modular y desacoplada, mejorando la escalabilidad y mantenibilidad del proyecto. Se decidió combinar MVI con Clean Architecture para aprovechar sus beneficios y asegurar un diseño robusto y escalable.



3.2.7. Otras consideraciones

Además de las elecciones mencionadas, se trabajó en la evaluación de otras alternativas para diferentes partes del proyecto, como la modularización, inyección de dependencias y vinculación de vistas, asegurando una solución integral y bien fundamentada.

Estas elecciones aseguran que el proyecto sea robusto, escalable y eficiente, cumpliendo con los requisitos técnicos y ofreciendo herramientas de desarrollo que facilitan el trabajo para un equipo de pocas personas.



Capítulo 4

Diseño de la interfaz gráfica y experiencia de usuario de Digipaw

El diseño de la interfaz gráfica y la experiencia de usuario de Digipaw son aspectos fundamentales para asegurar que la aplicación sea intuitiva, atractiva y fácil de usar. En esta sección, se describe el proceso llevado a cabo para materializar la idea inicial en una interfaz coherente, incluyendo la planificación y creación de los flujos de usuario, la elección de herramientas de diseño y los principios de usabilidad que guiaron cada decisión. La meta principal fue lograr una experiencia que no solo sea visualmente agradable, sino también funcional, permitiendo a los usuarios gestionar la información médica de sus mascotas con una curva de aprendizaje mínima.

4.1. Levantamiento de requerimientos

Para asegurar que la aplicación cumpliera con las necesidades reales de los usuarios finales, se realizó un proceso de levantamiento de requerimientos que se estructuró en varias etapas. En primer lugar, se identificó un grupo de usuarios potenciales, compuesto por personas con mascotas, a quienes se les presentó la idea de la aplicación. A estos usuarios se les solicitó información sobre los datos que consideraban importantes para gestionar la salud de sus mascotas, como nombres, especies, fecha de nacimiento, historial médico, y en particular, el control de las vacunas.

Posteriormente, el análisis se enfocó en un grupo más específico: personas que han vacunado a sus mascotas. Estas entrevistas permitieron profundizar en los tipos de datos que se manejan al momento de vacunar, cómo se realiza el seguimiento de las dosis, la periodicidad, y las necesidades generales que puedan tener las mascotas y tutores.

Finalmente, se contactó a un grupo de al menos tres veterinarios para obtener una visión experta sobre la administración de vacunas, la periodicidad recomendada, las marcas de productos y otros detalles relevantes. La información obtenida en esta etapa fue crucial para organizar y estructurar los datos de manera que fueran tanto útiles como intuitivos para los usuarios finales, garantizando que la aplicación refleje las prácticas y necesidades reales de los dueños de mascotas y profesionales veterinarios.

4.2. Materialización de la Idea

Para materializar la idea, se comienza haciendo dibujos de los flujos de la aplicación de la manera más cercana a la que se imaginaba. Estos dibujos iniciales que se muestran en la figura 4.1 sirvieron como base para conceptualizar cómo los usuarios interactuarían con la aplicación y cómo se organizarían las diferentes funciones.

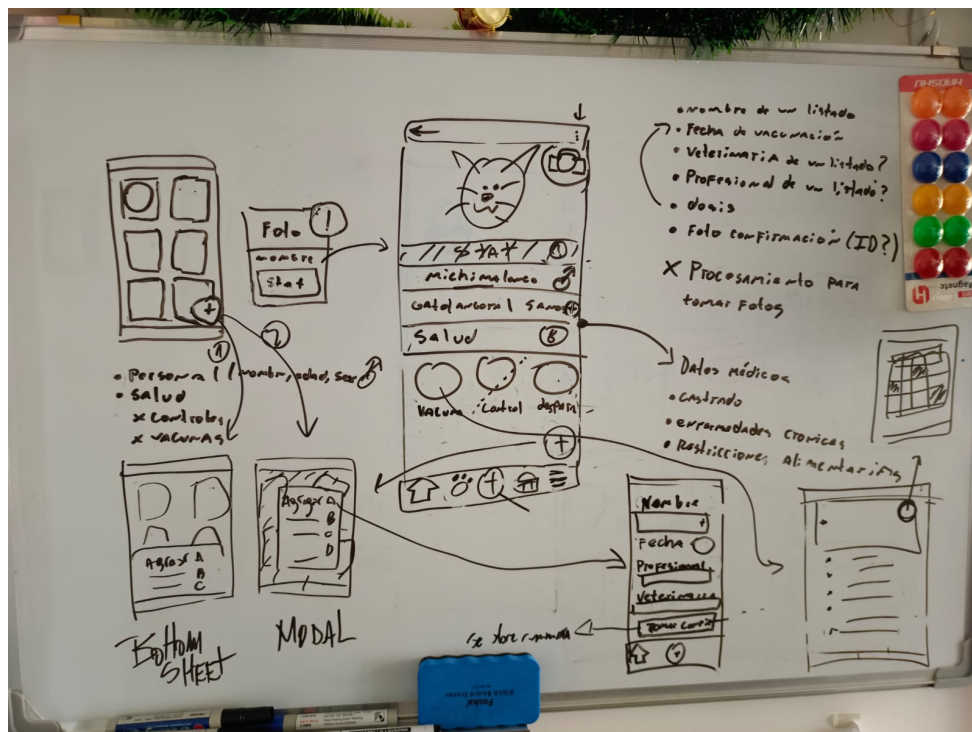


Figura 4.1: Dibujos iniciales de los flujos de la aplicación



4.3. Creación de Mockups

Con la idea bajada en un dibujo, se busca y trabaja con un diseñador profesional, quien transforma la idea en diseños digitales.

Para esta etapa en particular, se utiliza Figma, una herramienta de diseño basada en la web que permite a los diseñadores crear interfaces de usuario y colaborar en tiempo real. Es especialmente útil para crear mockups, que son representaciones visuales detalladas de las pantallas de una aplicación.

Los mockups son esenciales en el proceso de diseño, ya que permiten visualizar cómo se verá y funcionará la aplicación antes de comenzar el desarrollo. En Figma, los diseñadores pueden crear pantallas y componentes, definir interacciones y asegurarse de que el diseño sea coherente y fácil de usar.

- **Pantallas:** Cada pantalla de la aplicación se diseña en detalle, mostrando todos los elementos visuales, como botones, campos de texto, imágenes y menús.
- **Componentes:** Los componentes son elementos reutilizables que se pueden utilizar en múltiples pantallas, como botones, tarjetas y formularios. Figma permite crear una biblioteca de componentes que facilita la consistencia y la eficiencia en el diseño.
- **Interacciones:** Figma también permite definir interacciones entre las diferentes pantallas, simulando cómo se moverán los usuarios a través de la aplicación. Esto es crucial para evaluar la usabilidad y la experiencia del usuario.

La separación de componentes en cada pantalla es vital para la reutilización y la eficiencia en el desarrollo, un ejemplo de componentes de muestra en la figura 4.2. Los componentes deben programarse independientemente para ser reutilizados en diferentes partes de la aplicación. Esto no solo ahorra tiempo durante el desarrollo, sino que también asegura una mayor coherencia en la interfaz de usuario. Al reutilizar componentes, se reduce el esfuerzo de mantenimiento y se facilita la implementación de cambios o nuevas funcionalidades.

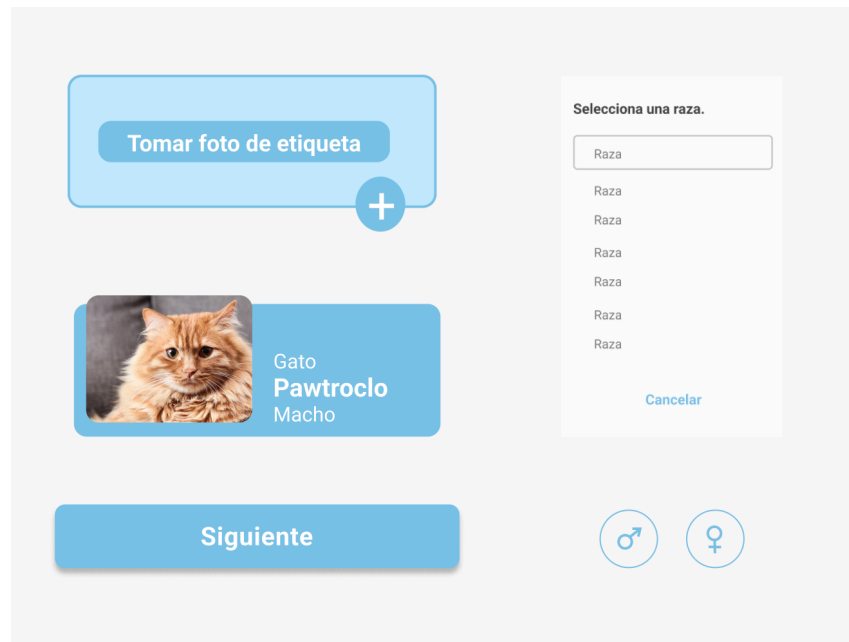


Figura 4.2: Ejemplo de componentes en figma

4.4. Flujos de la aplicación

4.4.1. Autenticación de Usuario

El flujo de autenticación de usuario permite a los usuarios existentes iniciar sesión en la aplicación. La pantalla de autenticación solicita al usuario que ingrese su correo electrónico y contraseña. Una vez validados estos datos, se concede acceso al sistema. Este flujo es fundamental para garantizar la seguridad y privacidad de la información médica de las mascotas y que los datos de las mascotas y sus usuarios solo sean accedidos a través de sus credenciales.

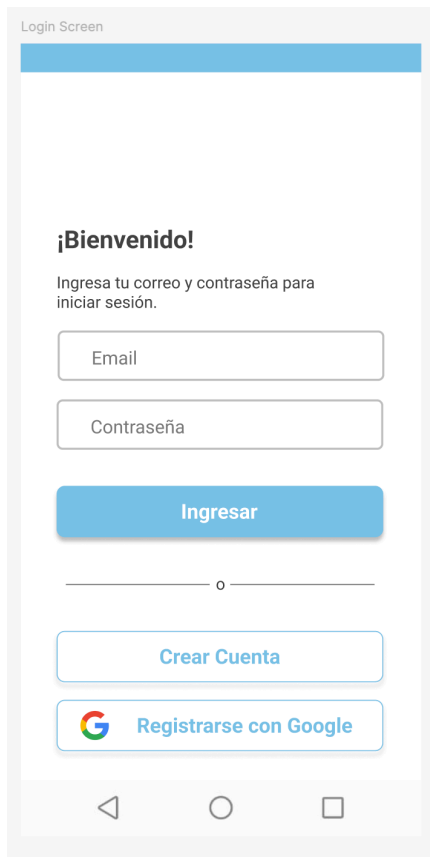


Figura 4.3: Pantalla de autenticación de usuario en Figma

La pantalla de autenticación en Figma, que se muestra en la figura 4.3 fue creada en pos de ser fácil de desarrollar, además se agregó un botón de registro con Google, el cuál no ha sido desarrollado para esta memoria.

4.4.2. Registro de usuario

El flujo de registro de usuario permite a los nuevos usuarios crear una cuenta en la aplicación. Este proceso implica la entrada de información básica como el nombre, correo electrónico y la creación de una contraseña, el ejemplo de este flujo se muestra en la figura 4.4. Además, se pueden incluir pasos adicionales, a futuro, para la verificación del correo electrónico para asegurar la autenticidad del usuario.



Figura 4.4: Flujo de registro de usuario en Figma

El flujo de registro comienza al hacer clic en el botón de **Crear Cuenta**, lo que llevará al usuario a la primera pantalla del proceso. En esta pantalla, se solicita al usuario que elija un nombre de usuario único. Este nombre de usuario está pensado solo para futuras funcionalidades planificadas. Al escribir el nombre de usuario y presionar **Siguiente**, se envía una llamada de red al backend para validar si el nombre de usuario ya existe en la base de datos.

En caso de que el nombre de usuario ya exista, se muestra un mensaje indicando que el nombre de usuario ya está en uso y no se permitirá avanzar a la siguiente pantalla. Si el nombre de usuario no existe en la base de datos, se permitirá avanzar a la siguiente pantalla, donde se realizará el mismo proceso de validación, pero esta vez con el correo electrónico del usuario, como se muestra en la figura 4.5.



Ingresar tu correo electrónico

Ingresar un correo electrónico válido para crear tu cuenta

Este email ya tiene una cuenta vinculada.

Figura 4.5: Error de correo electrónico ya registrado.

Después de verificar que el correo electrónico no está en uso y avanzar a la siguiente pantalla, el usuario deberá ingresar su nombre completo y crear una contraseña. Estos datos son obligatorios para poder crear la cuenta, y las casillas de contraseña deben coincidir para poder proceder, un ejemplo de error se muestra en la figura 4.6. Una vez realizado este paso y al presionar el botón **Siguiente**, se mostrará la pantalla de felicitaciones, proporcionando feedback positivo al usuario. Finalmente, el usuario será redirigido a la pantalla de inicio de sesión para que ingrese con sus nuevas credenciales.

Ingresar tu nombre y contraseña

Ingresar tu nombre, apellido y contraseña para poder crear tu cuenta.

Tu nombre y apellido te ayudarán a coordinar con la red de veterinarios.

Las contraseñas no coinciden

Las contraseñas no coinciden

Figura 4.6: Error de contraseñas no coincidentes.

4.4.3. Listado de Mascotas (Home)

Este flujo permite a los usuarios ver la lista de sus mascotas registradas en la aplicación. El usuario puede ver información relevante como el nombre, especie, raza y foto de cada mascota.

El diseño inicial del home se planteó con una interfaz moderna y visualmente atractiva. En la figura 4.7, se presenta la imagen de cómo se esperaba que fuera el diseño final.

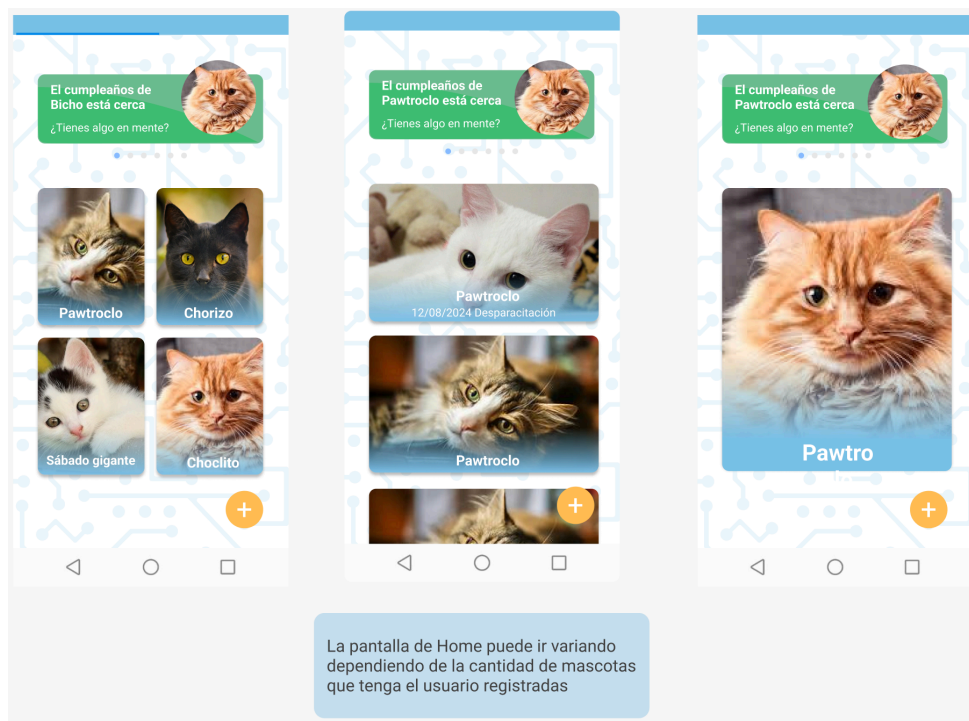


Figura 4.7: Diseño ideal de la pantalla del home con la lista de mascotas.

Sin embargo, por temas de tiempo y velocidad, se optó por implementar una versión más sencilla reutilizando un componente previamente creado para pruebas de concepto. La figura 4.8 muestra el estado actual del home.



Figura 4.8: Estado actual del home con la lista de mascotas.

No se alcanzó a plantear y desarrollar pantallas para los casos en los que el usuario no tiene mascotas o tiene problemas de red. En la figura 4.9, se muestra una idea recientemente creada para las pantallas sin mascotas.



Figura 4.9: Idea de pantalla para cuando el usuario no tiene mascotas.

4.4.4. Agregar Mascota

Este flujo permite a los usuarios agregar nuevas mascotas a su perfil. El usuario debe proporcionar información relevante como el nombre, especie, raza, y una foto de la mascota. Este flujo asegura que todos los datos básicos generales necesarios para el seguimiento médico estén completos y actualizados.

El flujo de agregar mascota comienza con una pantalla que solicita al usuario seleccionar o tomar una foto de su mascota. Esta funcionalidad se muestra en la Figura 4.10, donde al presionar el botón de agregar foto, el usuario tiene la opción de elegir una imagen desde su galería o tomar una foto nueva con la cámara de su dispositivo.

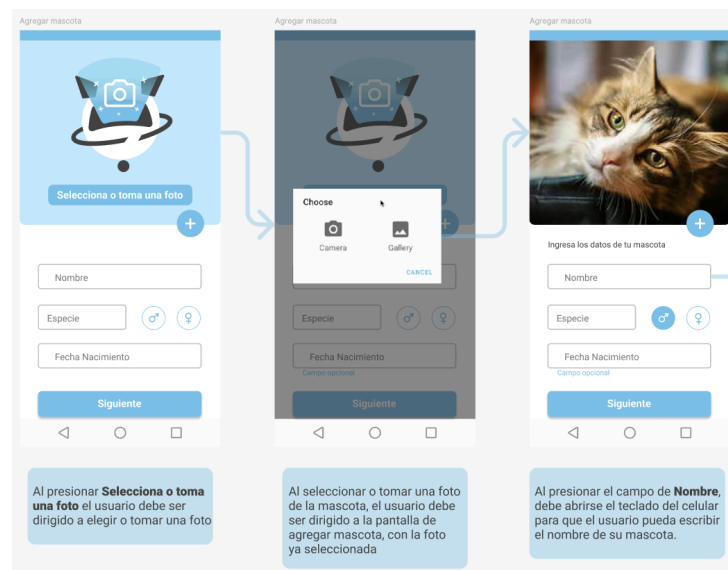


Figura 4.10: Pantalla de selección de foto para la mascota

Una vez seleccionada la foto, el usuario debe ingresar el nombre de la mascota. La Figura 4.11 muestra esta etapa, donde se proporciona un campo de texto para el nombre y un selector para la especie de la mascota.

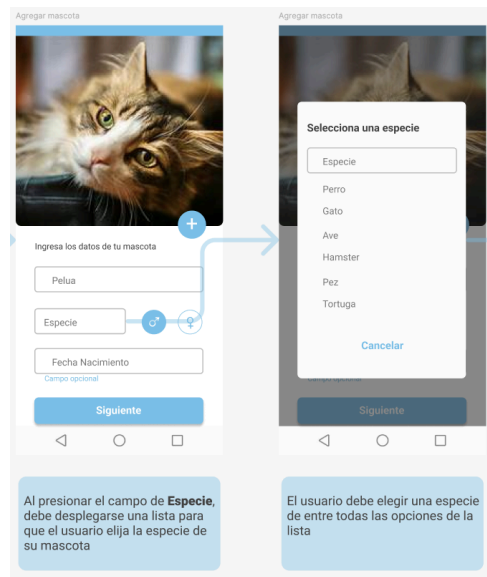


Figura 4.11: Pantalla para ingresar el nombre y seleccionar la especie

Al presionar el campo de especie, se despliega una lista de opciones que el usuario puede elegir. La figura 4.12 muestra también campos como el selector de macho o hembra y la fecha de nacimiento.

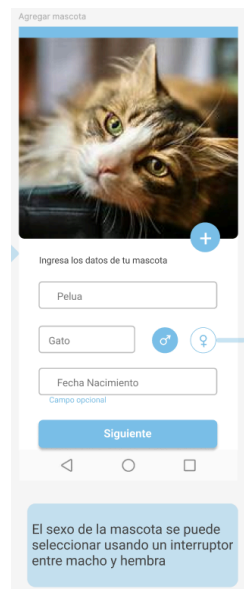


Figura 4.12: Uso de componente con iconografía, para elegir el sexo de la mascota

Además, el usuario puede proporcionar información sobre la fecha de nacimiento de la mascota, utilizando un componente desplegable para la selección de fecha. La Figura 4.13 muestra el uso de este componente.

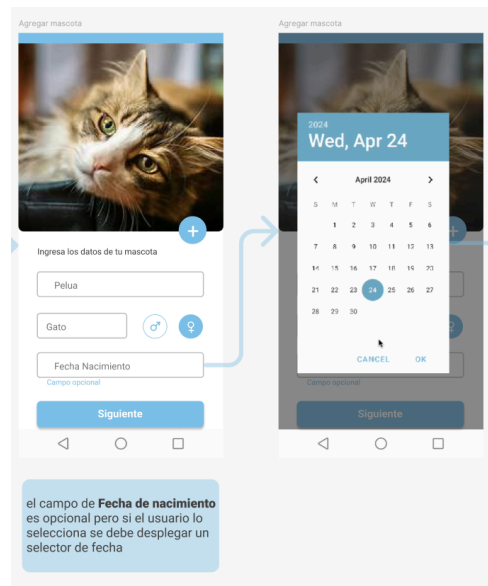


Figura 4.13: Ingresar la fecha de nacimiento de la mascota.

Además, el usuario puede proporcionar información adicional como la raza, la edad y una descripción de la mascota. La Figura 4.14 muestra el formulario donde se ingresan estos datos adicionales.

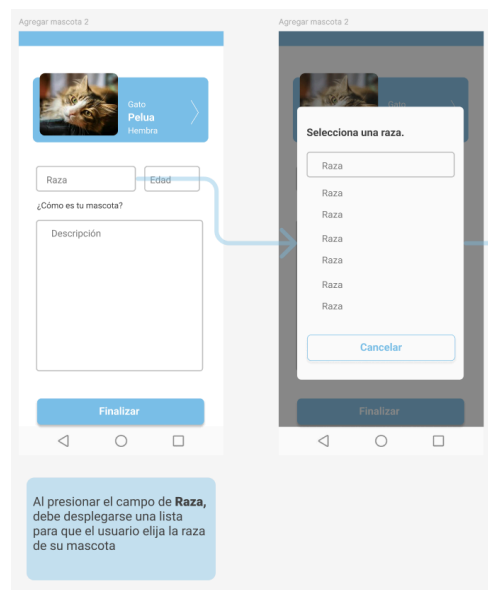


Figura 4.14: Despliegue de opciones para seleccionar la raza de la mascota

El campo de raza permite al usuario elegir entre una lista de opciones desplegables, como se muestra en la Figura 4.14. Esta lista desplegable de opciones dependerá de la especie que

haya elegido el usuario en la pantalla anterior, por ejemplo, si la mascota a agregar es un perro, entonces la lista de razas de esta pantalla solo mostrará razas de perros.

También el usuario puede libremente escribir sobre su mascota en el campo de descripción, como se muestra en la Figura 4.15

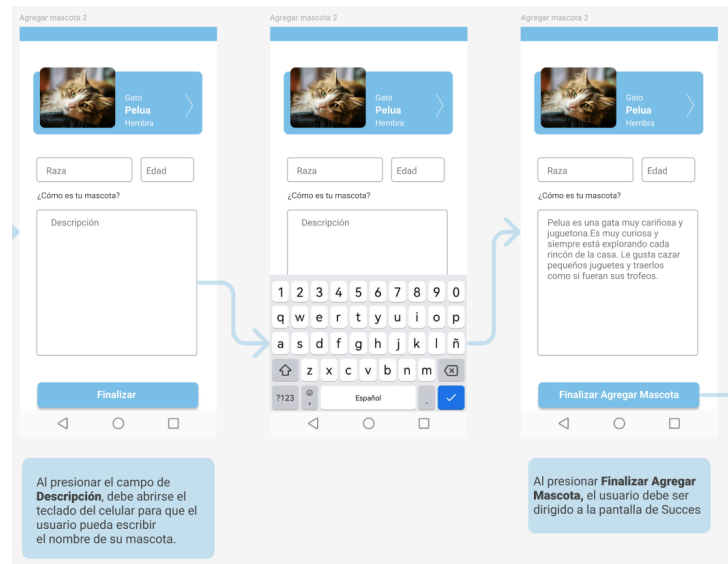


Figura 4.15: Agregar descripción de la mascota.

Finalmente, cuando el usuario completa todos los campos y presiona el botón de finalizar, se muestra una pantalla de éxito que confirma que la mascota ha sido añadida correctamente al perfil del usuario. La Figura 4.16 ilustra esta pantalla de confirmación.

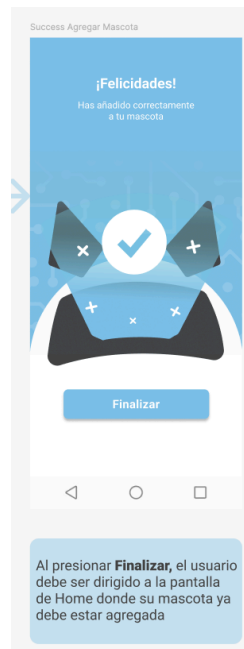


Figura 4.16: Pantalla de éxito al agregar una mascota

Además, se muestran mensajes de error si el usuario no completa los campos obligatorios como el nombre, la especie y la raza de la mascota. La Figura 4.17 muestra ejemplos de estos mensajes de error.

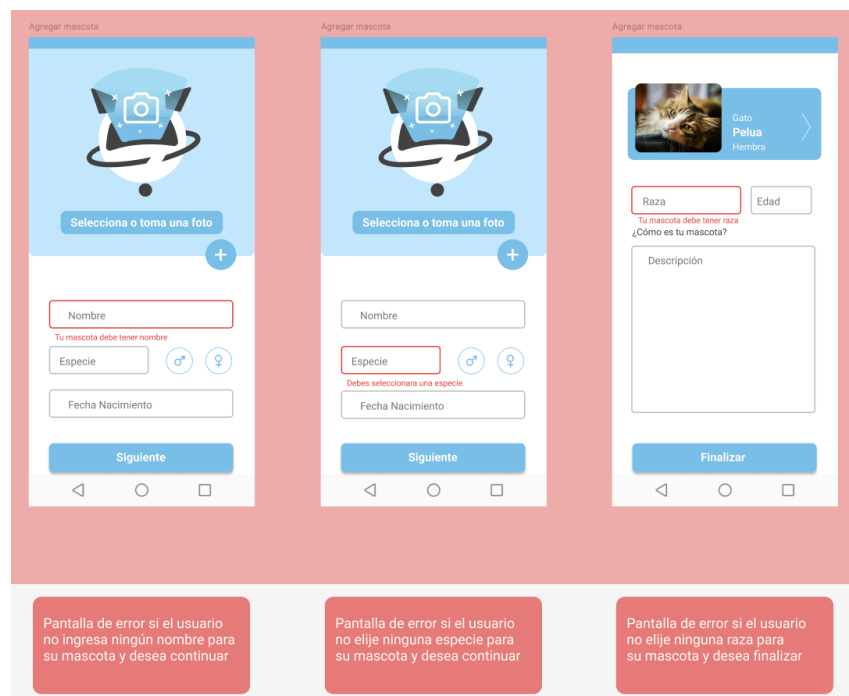


Figura 4.17: Mensajes de error si los campos obligatorios no son completados

Este flujo permite asegurar que toda la información necesaria sobre la mascota se recopile de manera adecuada.

4.4.5. Perfil de Mascota

El perfil de la mascota muestra información general sobre cada mascota registrada por el usuario, aquí además, se incluirán diferentes secciones, como puede ser datos médicos, historial de visitas al veterinario, vacunas, peso, última comida, etc. Los usuarios pueden actualizar la información y añadir nuevas entradas según sea necesario. En la figura 4.18 se muestra distintas variaciones del perfil de mascota.

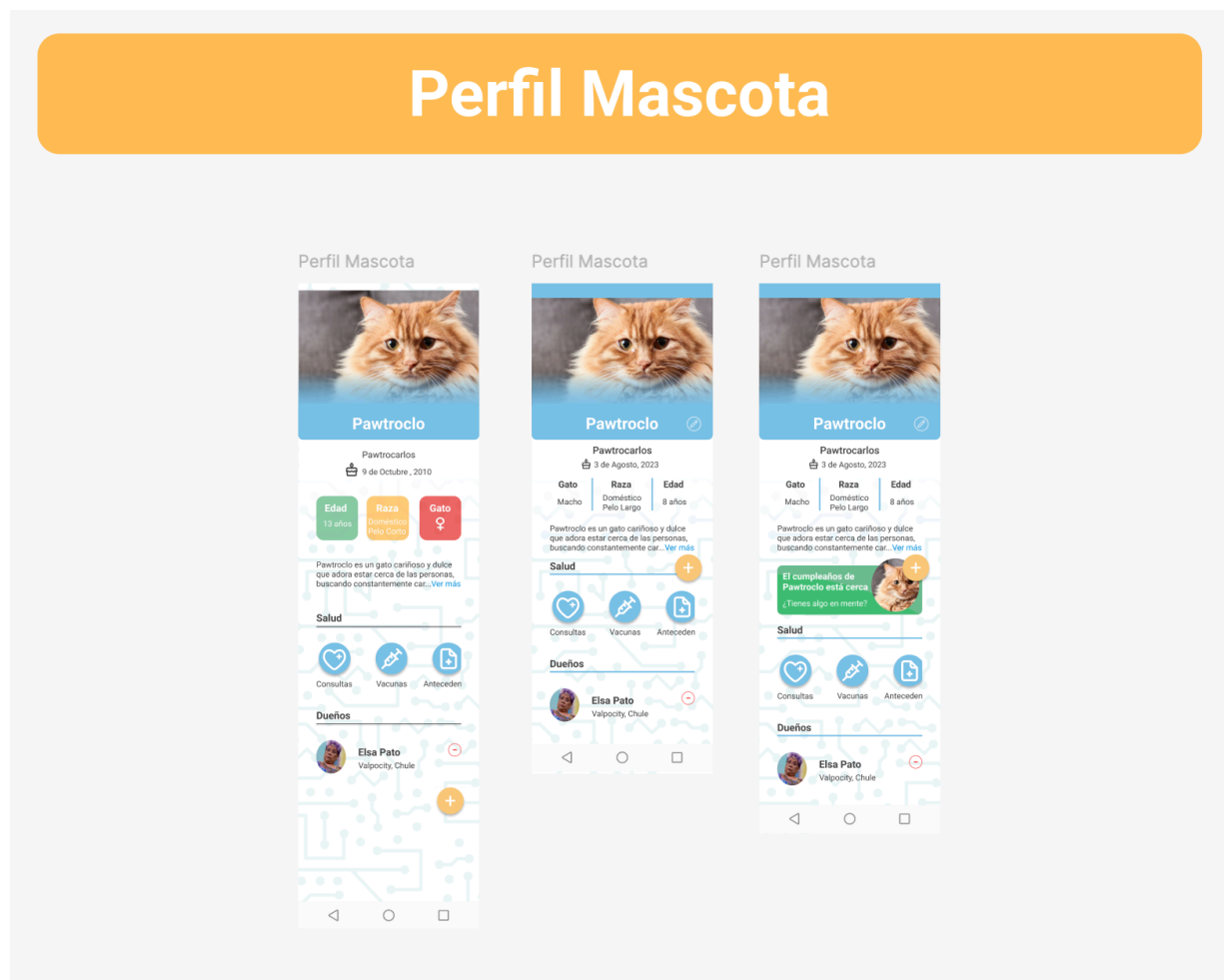


Figura 4.18: Pantalla de perfil de mascota



4.4.6. Historial de Vacunas

El flujo de historial de vacunas permite a los usuarios ver un registro detallado de las últimas vacunas administradas a la mascota, ordenadas por esquema de vacunación. La información presentada incluye la fecha de vacunación, una foto del sticker de la vacuna, la próxima fecha de vacunación, la dosis, las enfermedades que protege la vacuna, el tipo de vacuna y el nombre del veterinario que la administró.

Este flujo ordena las aplicaciones de vacunas desde la más próxima a administrar, utilizando una codificación de colores para alertar al usuario sobre el estado de las próximas dosis. Si la fecha de la siguiente dosis o vacuna está próxima, se muestra en amarillo; si la fecha ha pasado, se muestra en rojo; y si está dentro del rango adecuado, se muestra en verde. Esta funcionalidad es crucial para asegurar que las mascotas reciban sus vacunas a tiempo y mantener su salud óptima y al tutor informado.

El flujo comienza en el perfil de la mascota, en la sección de salud, donde se puede acceder al historial de vacunas presionando el ícono de vacunas, como se muestra en la figura 4.19.



Figura 4.19: Acceso al historial de vacunas desde el perfil de la mascota.

En esta pantalla, se puede ver un resumen de las vacunas organizadas por esquema. Al presionar el ícono de información que está en la esquina superior derecha de cada componente de vacuna, se muestra información detallada sobre el esquema de vacunación, como se muestra en la figura 4.20.



Figura 4.20: Información del esquema de vacunación.

Al presionar la imagen de la vacuna, se amplía la foto del sticker de la vacuna colocada, permitiendo al usuario verificar los detalles visuales del registro de vacunación, como se muestra en la figura 4.21.



Figura 4.21: Ampliación de la imagen del sticker de vacuna.

Finalmente, al presionar el componente de la vacuna, se accede a una pantalla que muestra todo el historial de vacunas administradas para ese esquema específico, ordenadas cronológicamente. Esta pantalla, retratada en la figura 4.22, permite al usuario desplazarse verticalmente para ver las vacunas administradas en años anteriores.



Figura 4.22: Historial completo de vacunas para un esquema específico.

Este flujo asegura que toda la información relevante sobre la vacunación de la mascota esté fácilmente accesible y organizada, permitiendo un seguimiento eficaz y completo del historial de vacunas. Además, facilita al tutor la obtención de información sobre las próximas vacunas de manera sencilla.

4.4.7. Agregar Vacunas

En este flujo, los usuarios pueden añadir nuevas vacunas al historial de la mascota. Se solicitan detalles como la fecha, una foto del sticker de la vacuna, el tipo de vacuna y datos opcionales como el nombre del veterinario y la clínica donde se administró. Este flujo es esencial para mantener un registro actualizado y preciso de las vacunas.

Para mejorar la facilidad y rapidez de uso, la pantalla prerrellena automáticamente la mascota seleccionada, aunque el usuario puede cambiarla fácilmente si es necesario. Además, dependiendo de la especie de la mascota elegida, solo se muestran en la lista las vacunas correspondientes



a esa especie.

Si la vacuna que se está eligiendo ya tiene una dosis aplicada con anterioridad, la aplicación prerrellena la selección de dosis con la siguiente dosis correspondiente.

Este diseño optimiza el tiempo que el usuario necesita para completar este flujo, asegurando eficiencia y precisión en el registro de las vacunas.

Mediante las figuras 4.23 y 4.24 se observan las diferentes etapas de todo el flujo.

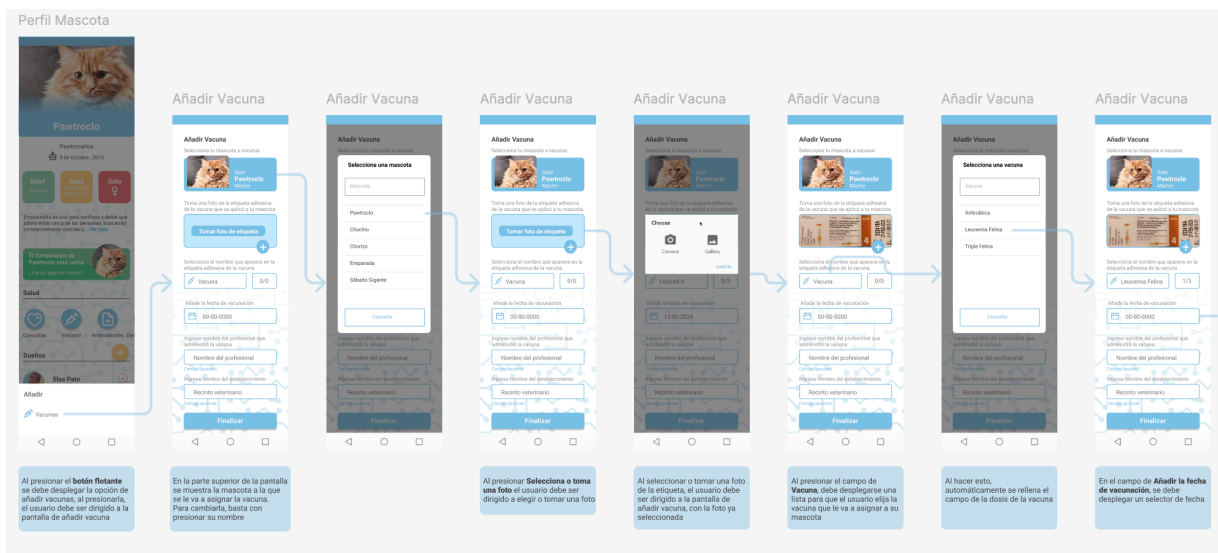


Figura 4.23: Flujo de agregar vacuna. parte 1

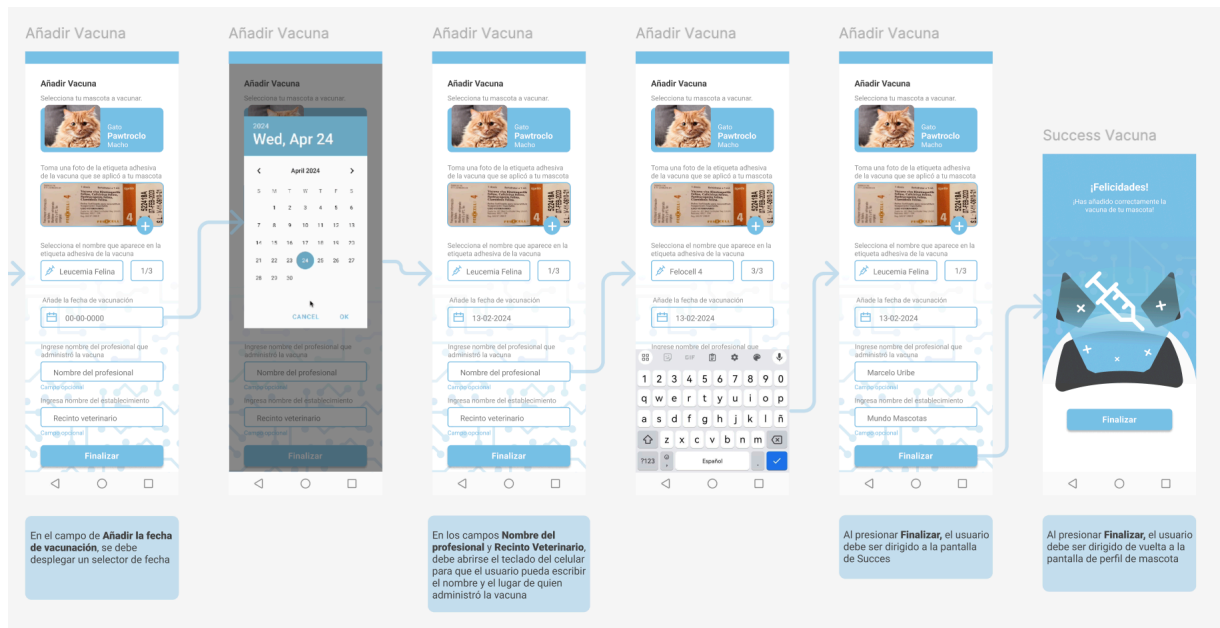


Figura 4.24: Flujo de agregar vacuna. parte 2



Capítulo 5

Desarrollo del Backend de Digipaw

El desarrollo del backend de Digipaw se enfocó en construir una arquitectura robusta y escalable que soportara las funcionalidades críticas de la aplicación. En esta sección, se describen los componentes clave, la organización de los servicios, y las decisiones técnicas que permitieron implementar una solución coherente para la gestión y procesamiento de los datos.

5.1. Base de datos

El modelado de la base de datos es crucial para garantizar que todos los datos necesarios sean almacenados de manera eficiente y que las relaciones entre los diferentes tipos de datos estén bien definidas.

El diseño de base de datos trabajado considera las siguientes entidades principales:

- **Usuario:** Un usuario puede tener varias mascotas.
- **Mascota:** Cada mascota puede tener múltiples tutores (usuarios), la mascota es de una especie y raza en específico.
- **Animal:** Cada especie de mascota que soportará la aplicación.
- **Raza:** Todas las razas de una especie.
- **Vacunación:** Registro de todas las vacunas administradas a una mascota.
- **Esquema de Vacuna:** Esquemas de vacunas recomendados por veterinarios, para cada especie.
- **Producto Vacuna:** Muchas vacunas pueden cumplir un esquema, cada vacuna tiene diferentes especificaciones dependiendo del laboratorio y la marca .



Diagrama de Modelado de Datos (UML) para un sistema de gestión de mascotas. El diagrama muestra entidades como Usuario, Mascota, Animal, Vacunación, Diagnóstico, Control, Tratamiento, Indicación, Afección, Producto Vacuna, Esquema Vacuna, Raza, Exámen, Visita, y Medicamento. Se incluyen relaciones con cardinalidades y restricciones. Hay dos historiales de detalles: 'Historial de Detalles' y 'Historial de Controles'. El diagrama está etiquetado como 'V0' en la esquina superior derecha.

Figura 5.1: Diagrama de la Base de Datos

Este diagrama fue diseñado considerando un contexto integral para la gestión de la salud de las mascotas, abarcando el registro de controles, diagnósticos, afecciones, tratamientos y más. Sin embargo, para priorizar el trabajo y ajustarse al tiempo disponible para desarrollar la aplicación, se ha definido una versión inicial (V0). Esta versión inicial, delineada con líneas punteadas en rojo en el diagrama, destaca las relaciones y entidades que se presentan específicamente en esta memoria. El objetivo es asegurar que podamos entregar un producto funcional dentro de las fechas previstas.

Para modelar estas relaciones en MongoDB, se sigue la siguiente estructura:

- **Usuario:** Contendrá un arreglo de referencias a mascotas.
- **Mascota:** Contendrá un arreglo de referencias a usuarios (tutores), un código de referencia a especie y uno de referencia a raza.



- **Vacunación:** Contendrá detalles de cada vacunación, incluyendo fecha, tipo de vacuna, sticker de vacuna, tendrá la referencia de la mascota vacunada y referencia del producto de vacuna.
- **Producto Vacuna:** Contendrá detalles del producto aplicado, los datos son obtenidos a partir de las páginas oficiales de los laboratorios fabricantes, tendrá una referencia al esquema de vacunación.
- **Esquema de Vacuna:** Contendrá detalles de los esquemas recomendados de vacunación, datos como; enfermedades que tratan, indicaciones, tiempo recomendado de vida de la mascota para aplicar una primera vacuna, especie a la que aplica.

La doble referencia entre usuarios y mascotas tiene ciertas ventajas y desventajas. La desventaja es que al tener dos fuentes de verdad, cabe la posibilidad de que hayan inconsistencias entre un array y otro, y esto genere resultados inconsistentes en la aplicación. Sin embargo, la ventaja de este enfoque radica en la velocidad de búsqueda y la reducción de las llamadas y relaciones necesarias en los servicios y/o bases de datos, mejorando así el rendimiento general del sistema.

5.1.1. MongoDB

MongoDB es una base de datos NoSQL que almacena datos en documentos JSON flexibles [15]. Esta característica permite una mayor agilidad en el desarrollo y escalabilidad en el manejo de grandes volúmenes de datos. A continuación, se detalla el proceso de configuración y creación de colecciones en MongoDB:

5.1.1.1. Crear una cuenta en MongoDB Atlas

Para comenzar a utilizar MongoDB, primero se debe crear una cuenta en MongoDB Atlas, que es la plataforma de base de datos como servicio (DBaaS) de MongoDB. Los pasos son los siguientes:

- Visitar el sitio web de MongoDB Atlas y registrarse para obtener una cuenta.
- Una vez registrado, crear un nuevo proyecto y luego un nuevo cluster dentro de ese proyecto. Un cluster es un conjunto de nodos de base de datos que almacenan sus datos.



5.1.1.2. Configurar el Cluster

Una vez creado el cluster, es necesario configurarlo. Los pasos incluyen:

- Seleccionar la configuración del cluster, como el tamaño, la ubicación geográfica de los nodos y la configuración de replicación.
- Desplegar el cluster una vez configurado. MongoDB Atlas se encargará de aprovisionar los recursos necesarios y poner en marcha el cluster.

5.1.1.3. Crear Colecciones

Con el cluster en funcionamiento, el siguiente paso es crear las colecciones necesarias para la aplicación. Las colecciones son equivalentes a las tablas en una base de datos relacional y se utilizan para organizar los documentos. Para este proyecto, creamos las siguientes colecciones:

- `users`: Almacena los datos de los usuarios.
- `pets`: Almacena los datos de las mascotas.
- `animals`: Almacena los datos de las especies, estas contienen un arreglo de razas.
- `vaccine-schemes`: Almacena los esquemas de vacunas.
- `vaccines`: Almacena la información de los productos de vacunas.
- `vaccinations`: Almacena la información de una aplicación de vacuna a una mascota.

Estas colecciones se crean directamente desde la interfaz de MongoDB Atlas o desde un visualizador de la base de datos, como lo es MongoDB Compass, el cual se muestra en la figura 5.2, lo que facilita la gestión y configuración de la base de datos.

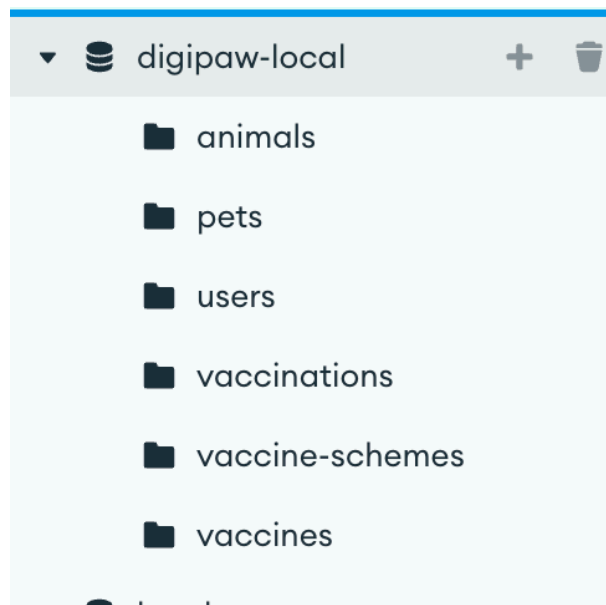


Figura 5.2: Colecciones de la base de datos creadas en MongoDB Compass

5.1.1.4. Definición de los Modelos de Datos

Para trabajar con MongoDB desde el backend, se utiliza en el proyecto Spring Boot junto con Spring Data MongoDB. Esto permite definir modelos de datos en forma de clases Kotlin, que se corresponden con las colecciones de la base de datos. A continuación se presentan ejemplos de los modelos de datos para las colecciones de usuarios y mascotas:

```
@Document(collection = "users")
data class Usuario(
    @MongoId
    var id: ObjectId = ObjectId.get(),
    var name: String = "",
    var email: String = "",
    var token: String = "",
    var username: String = "",
    var profilePhoto: String? = null,
    var city: String? = null,
    var region: String? = null,
    @JsonIgnore
    var password: String = ""
    set(value) {
```



```
        val encoder = BCryptPasswordEncoder()
        field = encoder.encode(value)
    },

    val pets: List<String> = mutableListOf() // Referencias a mascotas
)

@Document(collection = "pets")
data class Mascota(
    @MongoId
    var id: ObjectId = ObjectId.get(),
    var name: String = "",
    var genre: Genre = UNKNOWN,
    var description: String? = null,
    var birthDate: LocalDate? = null,
    var nickName: String? = null,
    var profilePhoto: String? = null,

    /** Scheme Relations */
    var animalType: AnimalType = AnimalType.UNSPECIFIED, //Referencia a
        ↪ especie
    var breedCode: String?, = null //Referencia a raza
    val ownerIds: List<String> = mutableListOf() // Referencias a usuarios
)
```

5.1.1.5. Conexión desde el Backend

Para conectar la aplicación móvil con MongoDB, se configura la URI de conexión en el archivo `application.properties` de Spring Boot. Esta URI es proporcionada por MongoDB Atlas y contiene la información necesaria para conectarse al cluster:

```
spring.data.mongodb.uri=mongodb+srv://<username>:<password>@cluster0.mongodb
    ↪ .net/test?retryWrites=true&w=majority
```

Para este proyecto, se implementó un sistema de autenticación mediante usuario y contraseña para conectarnos a la base de datos. Esta capa de seguridad es esencial para proteger la base de

datos contra accesos no autorizados. Es común que existan bots o individuos malintencionados que intenten acceder a diversas bases de datos. Si estas bases de datos no cuentan con una seguridad adecuada, pueden ser vulnerables a ataques que buscan secuestrar la información y exigir rescates en criptomonedas como bitcoins.

5.1.1.6. Visualización de Datos

Para visualizar y gestionar los datos almacenados en MongoDB, se utiliza MongoDB Compass, una herramienta gráfica que permite explorar y manipular los datos de manera intuitiva, esta herramienta se muestra en la figura 5.3. Esto facilita la verificación de que los datos se están almacenando correctamente y la realización de consultas ad hoc para analizar la estructura y el contenido de la base de datos.

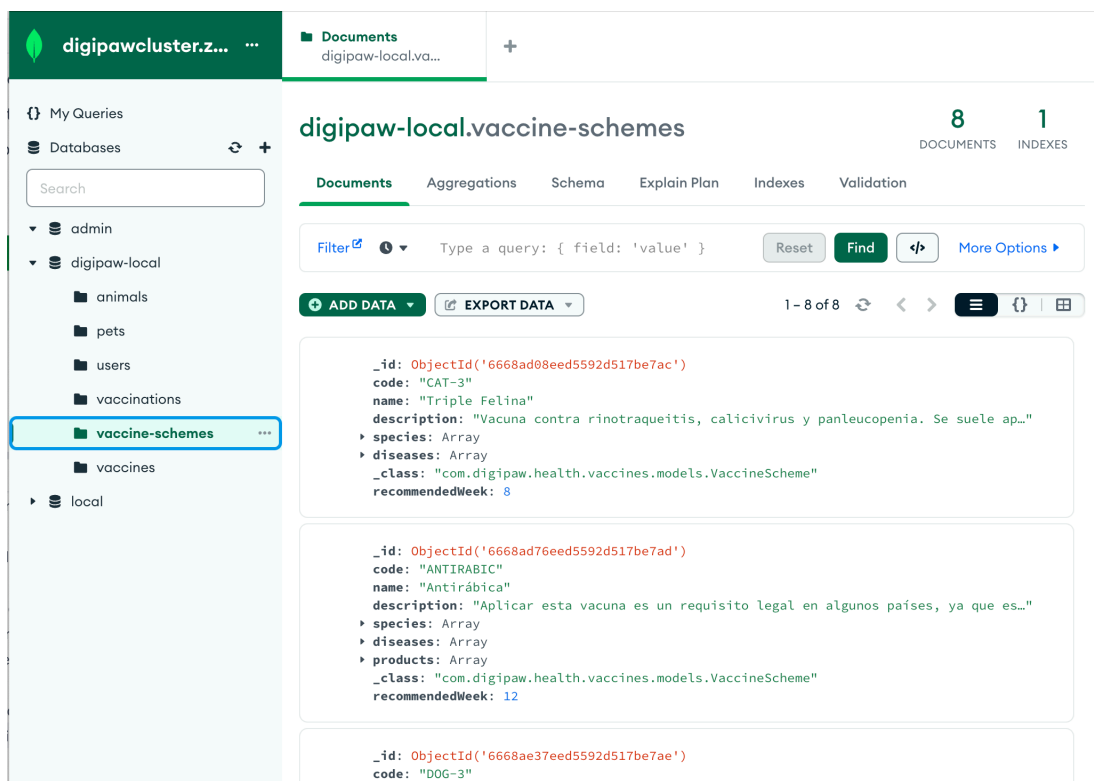


Figura 5.3: MongoDB Compass, para visualizar y administrar la base de datos

Con estas configuraciones, se garantiza que la base de datos sea robusta, escalable, segura y eficiente, soportando todas las funcionalidades requeridas por la aplicación.



5.2. Backend

La parte del sistema que se encarga de la lógica de negocio, el procesamiento de datos y la gestión de la comunicación con la base de dato

5.2.1. Estructura de una llamada de red

En la aplicación, se utilizan llamadas HTTP para la comunicación entre el frontend y el backend. Las llamadas HTTP son el método estándar para la transferencia de datos en la web y se componen de varios elementos clave:

- **URL (Uniform Resource Locator):** La URL especifica la dirección del recurso al que se está accediendo. Por ejemplo, una URL típica para acceder al perfil de un usuario podría ser `https://api.digipaw.com/users/id`.
- **Método de Petición (HTTP Method):** Indica la acción que se desea realizar. Los métodos más comunes son:
 - **GET:** Solicita datos del servidor.
 - **POST:** Envía datos al servidor para crear un nuevo recurso.
 - **PUT:** Envía datos al servidor para actualizar un recurso existente.
 - **DELETE:** Solicita la eliminación de un recurso del servidor.
- **Cabeceras (Headers):** Las cabeceras proporcionan información adicional sobre la solicitud, como el tipo de contenido (`Content-Type`), la autenticación (`Authorization`) y otras metainformaciones. Por ejemplo:

```
Authorization: Bearer <token>
Content-Type: application/json
```
- **Parámetros (Parameters):** Pueden ser parte de la URL (query parameters) o del cuerpo de la solicitud (body parameters). Los parámetros en la URL se añaden al final de esta, precedidos por un signo de interrogación (?). Por ejemplo:



`https://api.digipaw.com/users?status=active`

- **Cuerpo de la Solicitud (Request Body):** El cuerpo contiene los datos que se envían al servidor, generalmente en formato JSON. Esto es común en las solicitudes POST y PUT. Un ejemplo de cuerpo de solicitud en JSON sería:

```
{  
  "nombre": "Fido",  
  "edad": 3,  
  "raza": "Labrador"  
}
```

Ejemplo de una llamada HTTP:

```
POST /api/v1/pets HTTP/1.1  
Host: api.digipaw.com  
Authorization: Bearer <token>  
Content-Type: application/json  
Content-Length: 72
```

```
{  
  "nombre": "Fido",  
  "edad": 3,  
  "raza": "Labrador"  
}
```

En este ejemplo, se está realizando una solicitud POST al endpoint `/api/v1/pets` del servidor `api.digipaw.com` para añadir una nueva mascota. La cabecera `Authorization` incluye un token JWT (Json Web Token) para autenticación, y el cuerpo de la solicitud contiene los datos de la mascota en formato JSON.



Esta estructura permite una comunicación clara entre el frontend y el backend, facilitando la transferencia de datos y la ejecución de acciones específicas dentro de la aplicación.

5.2.2. Sincronización entre Frontend y Backend

Una de las prioridades en el desarrollo de la aplicación es mantener una sincronización efectiva entre el frontend y el backend. Esto implica que ambos componentes deben estar alineados en términos de funcionalidades y datos manejados. La sincronización es crucial para evitar problemas como discrepancias en los datos presentados al usuario y errores en las transacciones de red.

Para lograr esta sincronización, de deben tener las siguientes consideraciones:

- **Endpoints Consistentes:** Los endpoints del backend están diseñados para proporcionar datos de manera consistente y estructurada. Esto permite que el frontend sepa exactamente qué esperar en cada respuesta del backend.
- **Documentación Clara:** Toda la comunicación entre el frontend y el backend está documentada. Se utilizó Confluence¹[20] para subir y mantener actualizados los archivos JSON que definen los contratos de las API.

Creación de Contratos

Para permitir que los equipos de frontend y backend trabajen de manera paralela, se ha implementado la creación de contratos. Un contrato define las reglas y estructuras de los datos que se intercambian entre el frontend y el backend. Estos contratos son esenciales para el desarrollo colaborativo.

En la figura 5.4 se muestra cómo interactúan tanto el Backend como el Frontend con el contrato.

¹Herramienta de colaboración en equipo para documentar, organizar y compartir información.

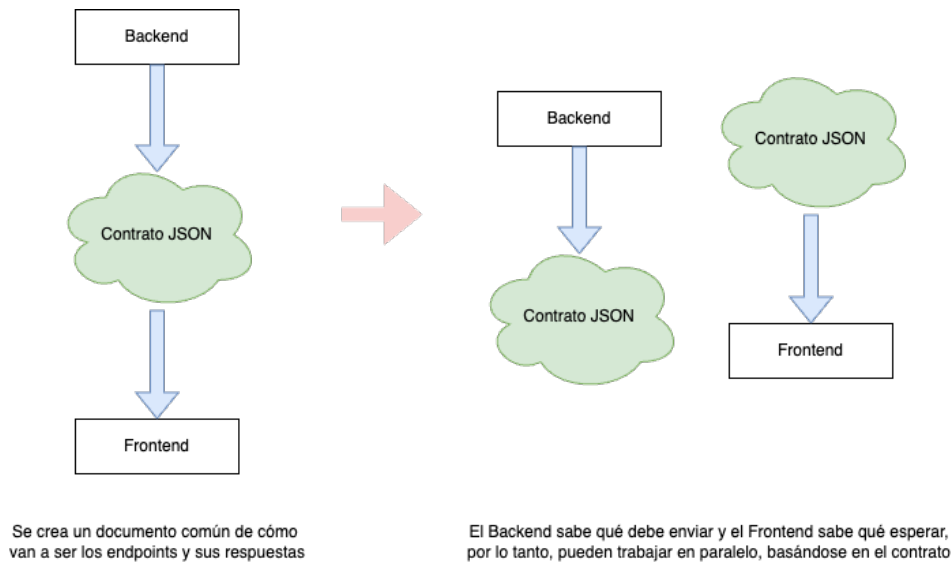


Figura 5.4: Diagrama del uso de contratos entre el Backend y el Frontend

- **Definición de Contratos:** Los contratos se definen en una etapa temprana del desarrollo y describen los formatos de solicitud y respuesta, tipos de datos, y cualquier validación necesaria. Estos contratos se crean manualmente y se documentan en Confluence.
- **Simulación de Respuestas:** Para permitir que el frontend avance en el desarrollo sin tener que esperar a que el backend esté completamente listo, se crean archivos JSON que simulan las respuestas del backend. Estos archivos se leen directamente en la aplicación para simular la respuesta del backend.
- **Validación Continua:** Durante el desarrollo, se utilizan herramientas como Insomnia [21], una herramienta de software que permite para validar automáticamente las solicitudes y respuestas contra los contratos definidos. Esto asegura que cualquier desviación se detecte y corrija de inmediato.

Importancia de los Contratos en el Desarrollo Paralelo

La creación de contratos no solo facilita la comunicación y la colaboración entre los equipos de frontend y backend, sino que también permite un desarrollo más ágil. Al definir claramente lo que se espera en cada interacción, ambos equipos pueden trabajar en sus respectivas áreas sin tener que esperar a que el otro esté completo.



Esto tiene varios beneficios:

- **Reducción de Dependencias:** Los equipos pueden trabajar de manera más independiente, reduciendo las dependencias y los bloqueos que a menudo retrasan los proyectos.
- **Aumento de la Productividad:** Con menos tiempo de espera y una mayor claridad sobre los requisitos, los equipos pueden ser más productivos y entregar funcionalidades más rápidamente.
- **Mejora en la Calidad del Código:** La validación continua contra los contratos ayuda a mantener una alta calidad del código, asegurando que las integraciones funcionen sin problemas.

En resumen, la sincronización y la creación de contratos son componentes clave de esta estrategia, permitiendo un desarrollo paralelo. A través de la definición clara de contratos y el uso de archivos JSON para simular respuestas, se puede asegurar que ambos equipos puedan trabajar de manera independiente y hacer entregas más rápidas.

5.2.3. Spring Boot

Spring Boot [14] es el marco principal que se utiliza para desarrollar servicios que se ejecutan en la nube. A continuación, se describe la configuración y estructura de los componentes clave de una aplicación Spring Boot:

Configuración

- El archivo `application.properties` contiene las configuraciones necesarias para conectar el servicio con MongoDB y otros componentes del sistema.

Repositorio

- Los repositorios son interfaces que extienden `MongoRepository` y proporcionan métodos CRUD (*Crear, Leer, Actualizar y Eliminar*) para interactuar con las colecciones de MongoDB, se anotan con `@Repository`.

Servicio



- Los servicios contienen la lógica de negocio de la aplicación y son anotados con `@Service`.

Controlador

- Los controladores gestionan las solicitudes HTTP entrantes y están anotados con `@RestController`. Utilizan los servicios para procesar las solicitudes y devolver las respuestas adecuadas.

5.2.4. Microservicios

La arquitectura de microservicios estructura una aplicación como una colección de servicios pequeños y autónomos que se comunican mediante APIs ligeras. Este enfoque permite una mayor escalabilidad, flexibilidad en el desarrollo y facilidad de mantenimiento.

Ventajas de los Microservicios

- **Escalabilidad Independiente:** Permite escalar cada microservicio de forma independiente.
- **Desarrollo y Despliegue Independiente:** Facilita el trabajo simultáneo en diferentes servicios y su despliegue independiente.
- **Aislamiento de Fallos:** Un fallo en un servicio no afecta a los demás, mejorando la resiliencia.
- **Tecnologías Diversas:** Permite el uso de diferentes tecnologías y lenguajes según las necesidades de cada servicio.

Desarrollo de Microservicios

Para desarrollar los microservicios, seguimos estos pasos:

- **Definición de Funcionalidad:** Cada microservicio aborda una funcionalidad específica, como la gestión de usuarios, mascotas o salud.
- **Configuración del Proyecto:** Se crea un proyecto Spring Boot para cada servicio.



- **Modelado de Datos:** Se definen las clases de datos que el servicio gestionará, una colección de la base de datos no puede ser gestionada por más de un servicio.
- **Repositorios:** Se implementan interfaces de repositorio para manejar operaciones CRUD.
- **Servicios:** Se desarrollan clases que contienen la lógica de negocio.
- **Controladores:** Se crean controladores para manejar las solicitudes HTTP entrantes.

Esta estructura asegura que cada microservicio sea independiente y escalable, facilitando el desarrollo y el mantenimiento de la aplicación.

5.3. Arquitectura del Backend

La arquitectura del Backend en Digipaw está basada en microservicios. A continuación se describen todos los servicios creados para la aplicación y las relaciones entre cada uno de ellos.

5.3.1. Servicio Auth

El servicio de autenticación (Auth Service) es responsable de gestionar la autenticación de los usuarios y la generación de tokens JWT (JSON Web Token) [16].

Es relevante señalar que en este proyecto las transacciones no están encriptadas. Aunque se gestionaron certificaciones para implementar HTTPS, esta tarea fue despriorizada debido a limitaciones de tiempo, priorizándose la finalización de funcionalidades críticas. Sin embargo, es crucial destacar la necesidad de implementar la encriptación de estas transacciones, tanto por motivos de seguridad como para cumplir con los requisitos de Google, ya que la publicación de aplicaciones en su tienda requiere que todas las comunicaciones estén encriptadas.

A continuación, se detalla cómo se crea y gestiona un token JWT en la aplicación.



5.3.1.1. Generación de Claves RSA

Para firmar los tokens JWT, se utilizaron claves RSA. El componente Keys se encarga de generar un par de claves, una pública y una privada. La clave privada se utiliza para firmar los tokens, asegurando su integridad y autenticidad.

5.3.1.2. Proceso de Inicio de Sesión y Generación de Token

Cuando un usuario intenta iniciar sesión, se realiza el siguiente proceso en el endpoint /login:

- **Validación del Usuario:** El sistema verifica si el usuario existe en la base de datos y si la contraseña proporcionada es correcta. Esta contraseña es encriptada antes de ser guardada en la base de datos.
- **Creación del Token JWT:** Si las credenciales son correctas, se crea un token JWT. Este token incluye el identificador del usuario (Issuer), una fecha de expiración (establecida a 15 días desde su creación) y se firma con la clave privada utilizando el algoritmo RS256.

El token generado se añade al perfil del usuario y se devuelve al cliente. Este token debe ser enviado en cada solicitud posterior desde la aplicación para autenticar al usuario. El token se envía en el encabezado de autorización de la siguiente manera:

```
Authorization: Gato <token>
```

5.3.1.3. Verificación del Token de Autenticación

El endpoint /validate-token permite verificar la validez de un token JWT. Este proceso es esencial para asegurar que solo los usuarios autenticados puedan acceder a los recursos protegidos.

En la figura 5.5 se muestran las verificaciones que realiza el servicio de Autenticación mediante el endpoint /validate-token.

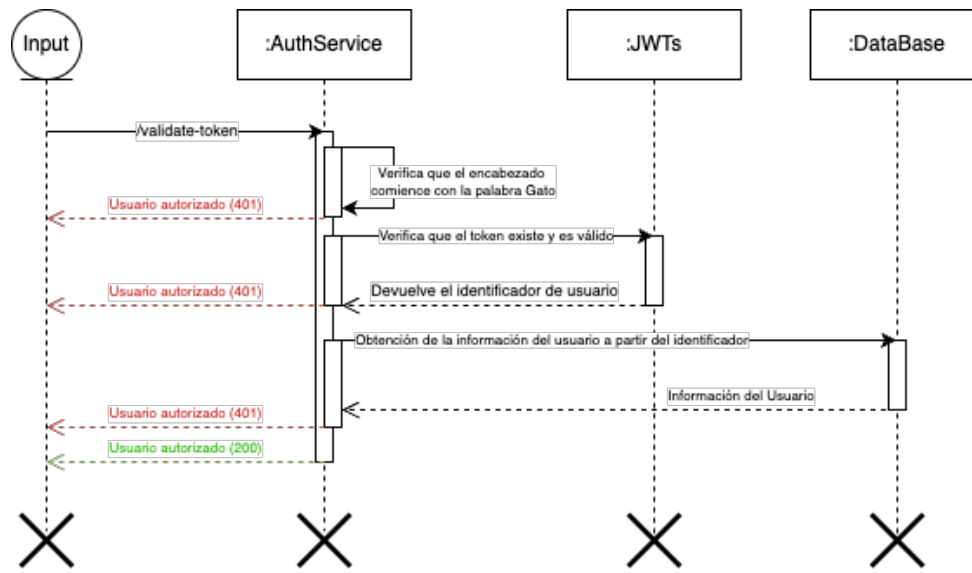


Figura 5.5: Diagrama de secuencia del servicio /validate-token

- Extrae el token del encabezado de autorización de la solicitud. El token debe comenzar con el prefijo *Gato* , que es la palabra secreta elegida para las llamadas de la aplicación.
- Verifica la estructura del token y que esté firmado correctamente utilizando la clave privada.
- Si el token es válido, extrae el identificador del usuario del token y lo utiliza para obtener la información del usuario.

Si el token es válido, el servidor devuelve una respuesta confirmando la autenticidad del usuario. Si no es válido, se devuelve un estado de no autorizado.

En resumen, el servicio de autenticación gestiona la creación y validación de tokens JWT para asegurar que solo los usuarios autenticados puedan acceder a los recursos protegidos. Utiliza claves RSA para firmar los tokens, garantizando su integridad y autenticidad. Además, proporciona endpoints para la verificación de tokens de autenticación, asegurando que los tokens utilizados en las solicitudes sean válidos y auténticos. Desde la aplicación, cada llamada a la API debe incluir el token de autenticación en el encabezado de la solicitud para validar al usuario.



5.3.2. Servicio Pet

El Servicio Pet gestiona toda la información relacionada con las mascotas dentro de la aplicación. Este servicio interactúa principalmente con las colecciones `pets` y `animals` en la base de datos, y proporciona una variedad de funcionalidades CRUD para las mascotas.

El servicio permite a los usuarios registrar nuevas mascotas, actualizar la información existente, recuperar datos de múltiples mascotas y gestionar la relación entre mascotas y propietarios. Esto incluye la capacidad de manejar múltiples propietarios para una sola mascota, facilitando la gestión compartida entre varios usuarios.

Integración con Otras Colecciones

El Servicio Pet no solo interactúa con la colección `pets`, sino que también se integra con la colección `animals` para obtener datos adicionales sobre el tipo de animal. Además, cuando se necesita actualizar los perfiles de usuario con las mascotas registradas, el Servicio Pet se comunica con el Servicio Auth, que gestiona la colección `users`. Esta integración asegura la coherencia y precisión de los datos a lo largo de toda la aplicación y evita que el servicio Pet manipule una colección que está siendo gestionada por otro servicio.

Operaciones y Consistencia de Datos

Una de las prioridades del Servicio Pet es mantener la consistencia de los datos. Cuando se realiza cualquier operación, como el registro de una nueva mascota, se asegura que todas las entidades relacionadas estén sincronizadas. Por ejemplo, al registrar una mascota, se actualizan tanto la colección `pets` como la colección `users` a través del Servicio Auth para reflejar la nueva relación.

En resumen, el Servicio Pet proporciona una gestión integral y eficiente de la información de las mascotas en la aplicación. Su diseño modular permite registrar, actualizar y gestionar datos de mascotas y propietarios, asegurando la precisión y coherencia de la información. Gracias a su construcción, el Servicio Pet puede ser fácilmente reutilizado por otros servicios dentro de la aplicación, facilitando la expansión y el mantenimiento del sistema.



5.3.3. Servicio Health

El Servicio Health se encarga de gestionar toda la información relacionada con la salud de las mascotas en la aplicación. Este servicio interactúa principalmente con las colecciones `vaccine`, `vaccination` y `vaccine-scheme` en la base de datos, proporcionando una variedad de funcionalidades para el manejo de vacunaciones y esquemas de vacunación.

5.3.3.1. Funcionalidades Principales

El Servicio Health permite a los usuarios registrar nuevas vacunaciones, actualizar esquemas de vacunación y obtener información detallada sobre las vacunaciones de sus mascotas. Este servicio es crucial para mantener un registro completo y preciso de la salud de las mascotas, asegurando que todas las vacunaciones necesarias se realicen a tiempo y de acuerdo con los esquemas de vacunación recomendados.

Integración con Otras Colecciones

El Servicio Health se integra con varias colecciones de la base de datos para asegurar una gestión completa y precisa de la información de salud. Interactúa con la colección `vaccine` para obtener datos sobre las vacunas específicas, con la colección `vaccination` para registrar y recuperar datos de vacunaciones, y con la colección `vaccine-scheme` para gestionar los esquemas de vacunación aplicables a cada tipo de mascota.

Uso de Agregaciones en MongoDB

El Servicio Health utiliza las agregaciones de MongoDB para manejar consultas complejas que requieren la combinación de datos de múltiples colecciones. Las agregaciones en MongoDB son un marco que permite procesar datos de manera eficiente dentro de la base de datos, aplicando operaciones como filtros, uniones, agrupaciones y ordenamientos en una sola consulta.

Debido a las múltiples operaciones CRUD complejas que se realizan en el Servicio Health, las agregaciones permiten relacionar varias entidades y obtener resultados detallados y organizados. Este enfoque es más rápido y eficiente que realizar múltiples consultas independientes, ya que reduce la sobrecarga y mejora el rendimiento general de la aplicación. Por ejemplo, para



obtener las últimas vacunaciones por esquema, se utiliza una agregación que une datos de las colecciones `vaccination`, `vaccine`, y `vaccine-scheme`, proporcionando una vista completa y actualizada de la salud de la mascota.

En resumen, el Servicio Health proporciona una gestión integral y eficiente de la información de salud de las mascotas en la aplicación. Su diseño modular permite registrar, actualizar y gestionar datos de vacunaciones y esquemas de vacunación, asegurando la precisión y coherencia de la información. Gracias al uso de agregaciones en MongoDB, el Servicio Health puede manejar consultas complejas de manera rápida, mejorando la experiencia del usuario. Además, su construcción permite que el Servicio Health sea fácilmente reutilizado por otros servicios dentro de la aplicación, facilitando la expansión y el mantenimiento del sistema.

5.3.4. Servicio Middleend

El servicio MiddleEnd actúa como el orquestador de las solicitudes provenientes del Frontend, gestionando la interacción con los servicios del Backend en la aplicación. No se conecta directamente a la base de datos, sino que realiza llamadas de red a otros servicios. Este enfoque se utiliza en la arquitectura Backend for Frontend (BFF) para armar respuestas que cumplan con los contratos definidos para cada pantalla de la aplicación.

Funcionalidades Principales

El Servicio MiddleEnd recibe las solicitudes del frontend y se encarga de procesarlas, realizando las llamadas necesarias a los servicios del backend (como el Servicio Pet y el Servicio Health) para obtener la información requerida. Luego, procesa esta información utilizando clases encargadas específicamente de transformar los datos para ajustarse a las estructuras esperadas por el frontend y, finalmente, se envían las respuestas adecuadas. A estas clases se las denomina Mapper.



Transformación de Datos

Para cumplir con los contratos definidos para cada pantalla de la aplicación, el Servicio Middleend utiliza clases **Mapper**. Esto incluye la reorganización de datos y la adición de información adicional necesaria para el frontend.

Las clases Mapper toman los datos crudos obtenidos de los servicios del backend y los forman en objetos que el frontend puede consumir directamente. Este proceso de mapeo es crucial para mantener la consistencia y la claridad en la comunicación entre el frontend y el backend.

Manejo de Imágenes

El Servicio MiddleEnd también es responsable de manejar las imágenes, como fotos de mascotas o stickers de vacunación. Para esto, utiliza el servicio S3 de Amazon para almacenar las imágenes y obtener una URL, que luego se envía a los servicios relevantes (Pet y Health) para ser guardada en la base de datos.

Cuando se necesita agregar una imagen, el MiddleEnd recibe la imagen, la sube a S3 y obtiene una URL de acceso. Esta URL se incluye en los datos que se envían a los servicios del backend, asegurando que las imágenes se gestionen de manera eficiente y segura.

En la figura 5.6 se muestra el diagrama de la interacción entre el MiddleEnd y los servicios Pet y Health.

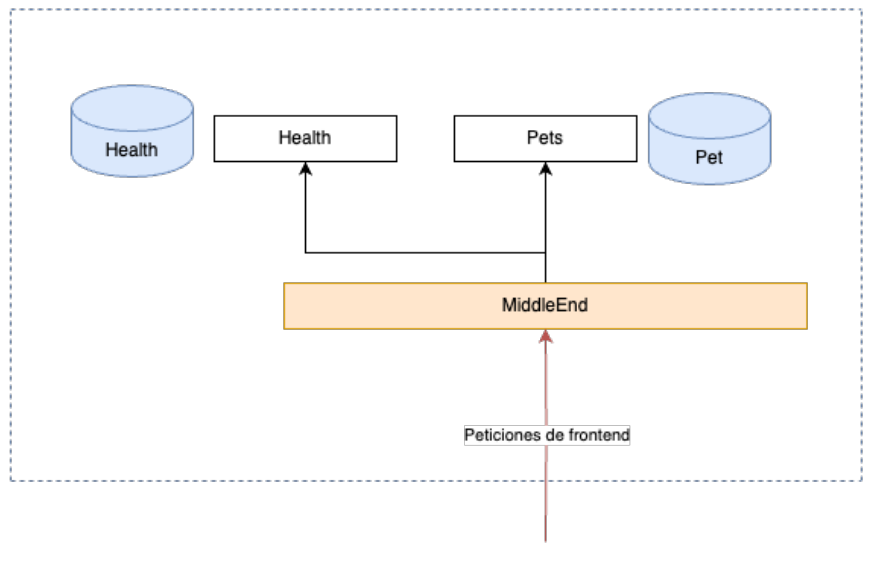


Figura 5.6: Diagrama de la comunicación entre el MiddleEnd y los diferentes servicios de

En resumen, el Servicio MiddleEnd actúa como un puente eficiente entre el frontend y el backend de la aplicación. Se encarga de gestionar las solicitudes entrantes, validar la autenticación, transformar los datos y manejar el almacenamiento de imágenes, asegurando que las respuestas cumplan con los contratos definidos para cada pantalla. Gracias a su construcción, el Servicio MiddleEnd puede ser el único servicio expuesto de todo el sistema, mejorando la seguridad de la infraestructura.

5.3.5. Gateway

Una vez completada la infraestructura de microservicios, se integra un Gateway al sistema utilizando NGINX [17]. Un Gateway actúa como un proxy², responsable de gestionar las solicitudes entrantes, ubicándose entre el Frontend y el Backend. Aunque este componente no contiene lógica de negocio y no es esencial para la operación básica de la aplicación, su función es clave para manejar las llamadas externas, permitiendo la configuración de aspectos críticos como seguridad y rendimiento. Además, el Gateway se convierte en la única interfaz visible de la infraestructura, protegiendo al MiddleEnd de ser expuesto como el punto de contacto inicial para todas las solicitudes.

²Un proxy es un intermediario que gestiona y redirige las solicitudes entre un cliente y un servidor, optimizando aspectos como la seguridad y el rendimiento de la red.

Funcionalidades Principales

El Gateway se encarga de enrutar las solicitudes del frontend al MiddleEnd, que a su vez se comunica con los servicios correspondientes del backend, como el Servicio Pet y el Servicio Health. Con NGINX, se han configurado reglas específicas para dirigir el tráfico de manera controlada.

Gestión de Autenticación

Una de las responsabilidades clave del Gateway es la verificación de la autenticación. Cada solicitud entrante incluye un token de autenticación, que es interceptado y validado mediante el servicio de autenticación (Auth Service). Esto garantiza que solo los usuarios autorizados puedan acceder a los recursos del sistema.

Para cada solicitud, el Gateway redirige la llamada al servicio de Auth, el cual verifica la validez del token de autenticación. Si el token es válido, el usuario es autenticado y la solicitud procede; de lo contrario, se devuelve un estado de no autorizado.

En la figura 5.7 se muestra el diagrama de la arquitectura final.

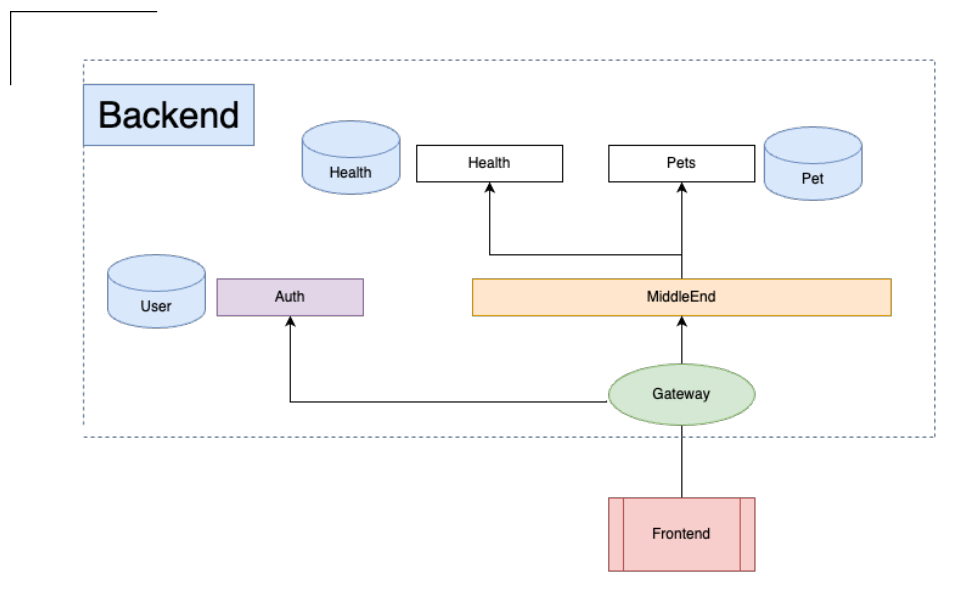


Figura 5.7: Diagrama de la arquitectura de servicios con Gateway

En resumen, la implementación del Gateway con NGINX ha mejorado la manera en que se manejan las solicitudes entrantes en la aplicación. Al actuar como la única cara visible de la



infraestructura y encargarse de la autenticación, el Gateway mejora la distribución de la carga, la seguridad y la escalabilidad del sistema. Esta implementación permite evitar el uso de MiddleEnd para gestionar la autenticación y poder enfocarse en sus funciones principales, optimizando así el rendimiento general de la aplicación. Aunque esta mejora aún no está completamente pulida, representa un avance significativo en la arquitectura del proyecto.



Capítulo 6

Desarrollo del Frontend de Digipaw

El desarrollo del frontend de Digipaw se enfocó en construir una aplicación nativa para Android que sea robusta, escalable y fácil de mantener. En esta sección, se describe cómo se organizó el código bajo una arquitectura modular que facilita su crecimiento y mantenimiento a medida que se agregan nuevas funcionalidades. También se discuten las decisiones técnicas adoptadas para asegurar que la aplicación mantenga su estabilidad en diferentes contextos de uso.

6.1. Antes de Desarrollar

Antes de comenzar el desarrollo de una aplicación móvil en Android, es fundamental realizar una serie de pasos para asegurar que el proceso sea lo más fluido y organizado posible. Estos pasos incluyen la creación de contratos con el backend, el reconocimiento de componentes reutilizables, la planificación de los flujos de la aplicación (happy path), el manejo de casos de error, y el diseño de los estados de pantalla y la navegación de flujo.

Contratos con Backend a partir de Mockups

Como se explica en la sección 5.2.2, se definen contratos claros con el backend utilizando archivos JSON para simular respuestas y probar funcionalidades del frontend antes de que el backend esté completamente implementado. Esto se facilita mediante la clase *LocalWebService*, que permite leer archivos JSON dentro de la aplicación para simular respuestas del backend.

Reconocer Componentes Reutilizables

Antes de comenzar el desarrollo, es importante revisar los mockups en Figma para identificar qué componentes de la interfaz de usuario pueden ser reutilizables. Un componente reutilizable es una parte de la interfaz de usuario que se puede usar en múltiples lugares dentro de la apli-

cación sin necesidad de duplicar el código. Reconocer estos componentes ayuda a mantener el código limpio y facilita el mantenimiento y la escalabilidad de la aplicación.

Al revisar los mockups, se deben identificar los elementos comunes como botones, tarjetas, listas, y otros componentes que se repiten en diferentes pantallas. Estos componentes se pueden abstraer en clases y layouts reutilizables. En la figura 6.1 se muestran algunos ejemplos.



Figura 6.1: Ejemplo de algunos componentes reutilizables de la aplicación.

A partir de estos reconocimientos, se puede comenzar a desarrollar estos componentes como códigos independientes, diseñados para recibir distintos parámetros que les proporcionen flexibilidad a la hora de usarlos en otros lugares de la aplicación. Primero se trabaja en todos los componentes de una pantalla antes de trabajar en la pantalla completa.

Happy Path

El *happy path* se refiere al flujo ideal de la aplicación, donde todo funciona correctamente y sin errores. Antes de desarrollar, es crucial definir este flujo revisando los mockups y planificando las interacciones del usuario con la aplicación. Esto incluye determinar las rutas de navegación, las interacciones principales y los resultados esperados en cada pantalla.

Planificación del Happy Path Al planificar el happy path, se debe tener en cuenta cómo los usuarios interactuarán con la aplicación desde el inicio hasta el final de un flujo específico. Esto



incluye:

- Determinar qué actividades y fragmentos se necesitan.
- Identificar en qué puntos se deben hacer llamadas de red.
- Establecer qué datos se deben enviar y recibir en cada llamada.

Casos de Error

Además del happy path, es esencial considerar los casos de error. Estos son los escenarios donde algo puede salir mal, como una llamada de red fallida, una entrada de datos incorrecta o falta de datos a ingresar en una validación. Identificar estos casos antes de desarrollar permite manejar los errores de manera efectiva y mejorar la experiencia del usuario.

Planificación de Casos de Error Para cada flujo principal, se deben identificar los posibles puntos de falla y definir cómo la aplicación debe responder. Esto puede incluir:

- Mensajes de error claros y útiles.
- Opciones para reintentar la acción fallida.
- Navegación alternativa en caso de error crítico.

En la figura 6.2 se pueden observar algunos ejemplos.

Diseño de Estados de Pantalla

Cada pantalla en la aplicación puede tener varios estados, como cargando, mostrando datos, vacío o error. Diseñar estos estados de pantalla antes de desarrollar asegura que se cubran todas las posibles situaciones y que la interfaz de usuario sea coherente y fácil de entender.

Planificación de Estados de Pantalla Para cada pantalla, se deben definir y diseñar los posibles estados. Esto incluye:

- Estado de carga: qué se muestra mientras se espera una respuesta de red.

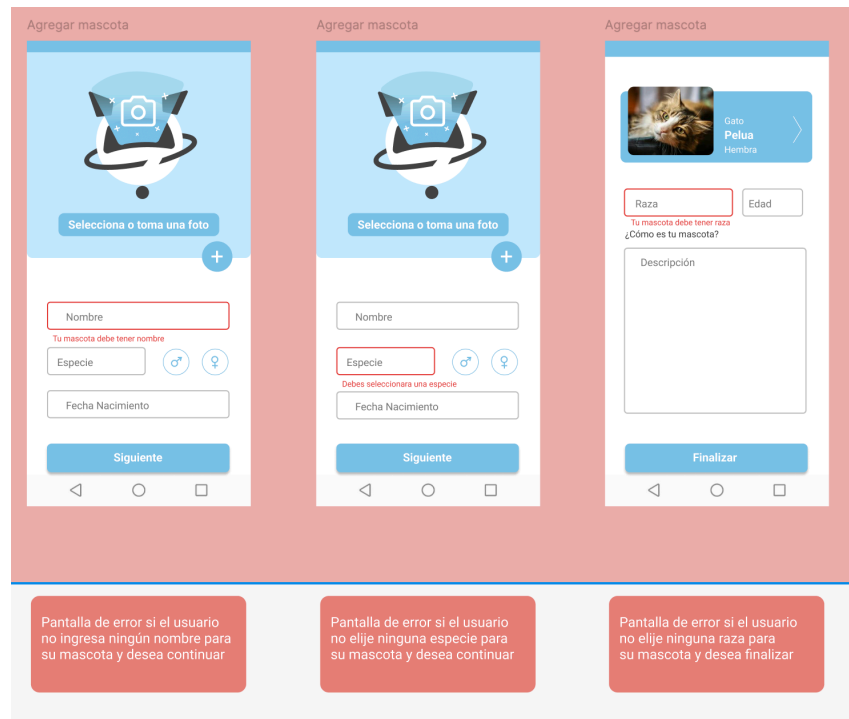


Figura 6.2: Ejemplo de casos de error de falta de datos en validación.

- Estado de éxito: cómo se muestran los datos cuando todo funciona correctamente.
- Estado vacío: qué se muestra cuando no hay datos disponibles.
- Estado de error: cómo se informan los errores al usuario.

Diseño de Navegación de Flujo

El diseño de la navegación de flujo implica planificar cómo los usuarios se moverán entre las diferentes pantallas y secciones de la aplicación. Usar el *Navigation Component* facilita esta tarea, permitiendo definir un gráfico de navegación en un archivo XML.

Planificación de la Navegación Antes de desarrollar, se debe revisar el flujo completo de la aplicación y definir:

- Las actividades y fragmentos necesarios.
- Las transiciones entre fragmentos.
- Las acciones que desencadenan la navegación (como botones o eventos de usuario).



Estos pasos aseguran que el desarrollo de la aplicación móvil sea organizado y claro desde el inicio, haciendo más fácil estimar el tiempo de desarrollo que puede tomar cada parte, reconocer cuándo se pueden paralelizar tareas y evitar tener que hacer refactors por una mala planificación inicial.

6.2. Preparación base de la aplicación móvil

El desarrollo de una aplicación nativa en Android utilizando Kotlin implica varios pasos fundamentales para asegurar que la aplicación esté bien estructurada y sea mantenible a largo plazo. A continuación, se describen los componentes principales involucrados en esta preparación.

Módulos de Gradle

La aplicación se ha dividido en varios módulos de Gradle, lo que permite una mejor organización y reutilización del código. Esta modularización es esencial para mantener la aplicación manejable y mejorar la velocidad de desarrollo. En entornos empresariales grandes, es común dividir una aplicación en múltiples librerías para evitar trabajar en un proyecto monolítico y masivo, que puede ser difícil de mantener y desarrollar. Cada componente de la aplicación puede ser desarrollado y mantenido de manera independiente, lo que facilita el trabajo en equipo y la implementación de nuevas funcionalidades.

- **app:** El módulo principal donde corre la aplicación base. Este módulo orquesta la interacción de los demás módulos y contiene la lógica principal de la aplicación.
- **network:** Este módulo se encarga de las conexiones de red. Implementa Retrofit[18], interceptores, configuraciones y factories para usar las APIs en el módulo de aplicación. La separación de la lógica de red en un módulo independiente permite un desarrollo y mantenimiento más sencillo y organizado.
- **uicomponents:** Los componentes de interfaz de usuario son elementos reutilizables que se utilizan en varias partes de la aplicación. Este módulo contiene todos estos componentes reutilizables. La creación de componentes reutilizables es importante porque reduce la duplicación de código y facilita la consistencia en la interfaz de usuario de la aplicación.

- **cache:** Este módulo maneja las implementaciones comunes del uso de *SharedPreferences* y *Room*. *SharedPreferences* se utiliza para almacenar datos clave-valor simples y *Room* es una librería de persistencia que proporciona una abstracción sobre SQL para manejar bases de datos locales.
- **utils:** Contiene clases o funciones de utilidad general que son importantes para ser reutilizadas en diferentes partes de la aplicación. Este módulo ayuda a mantener el código limpio y organizado, centralizando las funciones y clases utilitarias en un solo lugar.
- **featureflags:** Las implementaciones generales del uso de *Feature Flags* están contenidas en este módulo. Los *Feature Flags* permiten activar o desactivar funcionalidades de la aplicación de manera dinámica, lo que facilita las pruebas A/B y el lanzamiento controlado de nuevas características. En esta aplicación se usan las *Feature Flags* de Google Firebase.
- **mvi:** Contiene interfaces que facilitan la implementación de la arquitectura elegida, en este caso, Model-View-Intent (MVI). Este módulo ayuda a estructurar la lógica de la aplicación de manera clara y manejable.

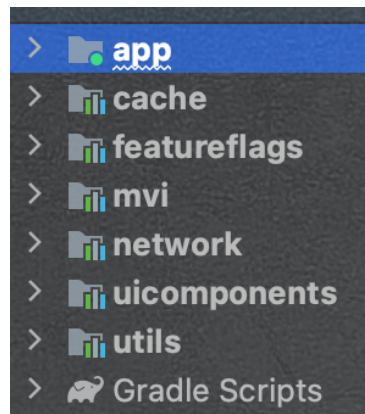


Figura 6.3: Estructura de módulos de Gradle en la aplicación.

Dividir la aplicación en estos módulos de Gradle, como se muestra en la figura 6.3, no solo mejora la organización del proyecto, sino que también permite trabajar de manera más efectiva en equipos grandes ya que se pueden reutilizar en diferentes proyectos que se han separado del principal. A medida que la aplicación crece, cada módulo puede ser desarrollado, probado



y mantenido de manera independiente, lo que reduce la complejidad y facilita la colaboración entre múltiples desarrolladores.

Conexión a Servidores - Módulo de Networking

La conexión a servidores en una aplicación Android se maneja comúnmente utilizando una librería llamada Retrofit. Retrofit facilita la creación de solicitudes HTTP y la interacción con APIs RESTful. El módulo de *network* en la aplicación se encarga de encapsular toda la lógica de comunicación con los servidores backend, asegurando que las llamadas de red se manejen de manera organizada y centralizada.

Implementación en Kotlin Para implementar las conexiones de red en Kotlin, se sigue una estructura que incluye la configuración de Retrofit y OkHttpClient, que es el cliente HTTP subyacente utilizado por Retrofit. A continuación, se describe el proceso de configuración y uso del módulo de networking.

Configuración de Retrofit Retrofit se configura mediante una clase que crea instancias del servicio web definido. Esta configuración incluye la creación de un cliente OkHttpClient con los interceptores necesarios para manejar la autenticación y el registro de solicitudes. Un interceptor es una pieza de código que puede interceptar y modificar solicitudes y respuestas HTTP.

La clase *RemoteWebService* es responsable de crear instancias de Retrofit configuradas con un cliente OkHttpClient que incluye un interceptor para el token de autenticación y un interceptor para el registro de solicitudes.

Intercepción de Tokens El *TokenInterceptor* se utiliza para agregar el token de autenticación a todas las solicitudes HTTP. Esto asegura que solo los usuarios autenticados puedan acceder a los recursos protegidos del backend. El interceptor añade un encabezado de autorización con el token a cada solicitud saliente.

```
class TokenInterceptor constructor(private val tokenManager: TokenManager) :  
    → Interceptor {  
    override fun intercept(chain: Interceptor.Chain): Response {  
        val headers = if (tokenManager.getToken().isEmpty()) {
```



```
        chain.request().headers
            .newBuilder()
            .add("Authorization", "Gato ${tokenManager.getToken()}")
            .build()
    } else {
        chain.request().headers
            .newBuilder()
            .build()
    }
    val request = chain.request().newBuilder().headers(headers).build()
    return chain.proceed(request)
}
}
```

Creación de Servicios Locales Para facilitar el desarrollo y las pruebas sin depender del backend, se utiliza una clase *LocalWebService*. Esta clase permite leer archivos JSON dentro de la misma aplicación y simular respuestas del backend. Esto es útil para probar funcionalidades del frontend antes de que el backend esté completamente implementado.

Factory de Servicios Web La clase *WebServiceFactory* facilita la creación de instancias de servicios web, seleccionando la configuración adecuada según el entorno (desarrollo, producción, etc.). Dependiendo del entorno, se puede configurar para usar servicios remotos o locales.

Definición de Servicios Web Los servicios web se definen mediante interfaces que especifican los endpoints y los métodos HTTP utilizados. Estas interfaces son luego implementadas por Retrofit para realizar las llamadas de red.

```
interface VaccinesWebService {

    @GET("api/frontend/vaccine-info/{petId}")
    suspend fun getHealthVaccines(@Path("petId") petId: String?):
        RemoteHealthVaccines

    @GET("api/frontend/vaccines")
```



```
suspend fun getVaccines(  
    @Query("specie") specieCode: String,  
    @Query("petId") petId: String  
) : List<RemoteVaccineProduct>  
  
@Multipart  
@POST("api/frontend/vaccine/add")  
suspend fun addVaccine(  
    @Part("body") body: RequestBody,  
    @Part vaccineSticker: MultipartBody.Part?  
)  
}
```

Utilización en la Aplicación La configuración de los servicios web se utiliza en la aplicación principal para realizar las llamadas a las APIs necesarias. Mediante el uso de factories, se crean instancias de los servicios web según el entorno y las necesidades de la aplicación.

Esta estructura modular y el uso de Retrofit y OkHttpClient permiten manejar de manera organizada las conexiones de red, asegurando que las solicitudes HTTP se realicen correctamente y que la autenticación se maneje de manera segura.

Flujo de Login

El flujo de login incluye la obtención, almacenamiento y envío del token de autenticación en cada llamada de red. Utilizando interceptores, se puede agregar el token a todas las solicitudes salientes automáticamente.

- **Obtención del Token:** Al iniciar sesión, el servidor devuelve un token que se almacena localmente.
- **Interceptor de Autenticación:** Un interceptor se configura para interceptar todas las solicitudes HTTP y agregar el token en el encabezado de autorización.
- **Almacenamiento Seguro:** El token se guarda de forma segura utilizando mecanismos como SharedPreferences con cifrado.



- **Cerrar sesión:** El token se elimina de la base de datos.

Inyección de Dependencias

La inyección de dependencias se realiza manualmente mediante una clase encargada de gestionar las instancias necesarias a lo largo de la aplicación. Esto permite desacoplar las clases y facilita la sustitución de dependencias durante las pruebas.

- **Clase Injector:** Una clase específica se encarga de crear y proporcionar las instancias de las dependencias.
- **Provisión de Dependencias:** Esta clase se utiliza en diferentes partes de la aplicación para obtener las instancias necesarias.

Arquitectura

La arquitectura de la aplicación sigue el patrón MVVM (Model-View-ViewModel), que ayuda a separar la lógica de negocio de la interfaz de usuario.

MVI (Model-View-Intent):

El patrón de diseño Modelo-Vista-Intención (MVI) es una arquitectura que se utiliza en el desarrollo de aplicaciones móviles para proporcionar una representación clara y unidireccional del flujo de datos y eventos. Se presenta como una mejora a MVVM, utilizando estados, separando las responsabilidades de diferentes actores y enfocándose en el flujo unidireccional de los datos.

- **Modelo (Model):** El modelo representa el estado actual de la aplicación. Puede ser cualquier estructura de datos que almacena la información relevante para la interfaz de usuario y la lógica de negocio. El modelo es inmutable y solo puede modificarse a través de la emisión de eventos.
- **Vista (View):** La vista es responsable de mostrar la interfaz de usuario y recibir las interacciones del usuario. La vista solo puede reflejar el estado actual del modelo y no tiene lógica



de negocio. Cuando se producen eventos de usuario, la vista los envía como intenciones (intents) al componente siguiente.

- **Intención (Intent):** La intención representa las acciones o eventos que ocurren en la interfaz de usuario. Las intenciones se envían desde la vista al siguiente componente en el flujo de datos. Por ejemplo, una intención puede ser un clic en un botón o una solicitud de actualización de datos.

Para utilizar este patrón, es necesario comprender los diferentes conceptos y capas que son fundamentales para su correcto funcionamiento. El patrón MVI se basa en la definición de cuatro tipos de eventos que se transforman a través de las diversas capas de procesos, las cuales se explican a continuación.

Eventos de MVI:

- **Intenciones de usuario (User Intent):** Representan las acciones que el usuario puede llevar a cabo en la aplicación. Estas intenciones se expresan en un lenguaje orientado a la acción del usuario y son emitidas por la capa de interfaz de usuario (UI).
- **Acciones (Actions):** Representan las acciones que el programa debe tomar en respuesta a las intenciones del usuario. Por ejemplo, si la intención del usuario es **ver** una lista de mascotas, la acción del programa sería **cargar** la lista de mascotas.
- **Resultados (Results):** Son los posibles resultados generados por las acciones del programa. Por ejemplo, al cargar la lista de mascotas, los resultados podrían ser: carga en progreso, error en la llamada de red o solicitud exitosa con los datos requeridos.
- **Estados (UI States):** Representan los estados en los que puede encontrarse la aplicación. Estos estados son una traducción visual coherente con el contexto del producto de los resultados de las acciones provocadas por las intenciones de usuario. Por ejemplo, la aplicación puede estar en un estado de carga mientras espera los datos, y la pantalla deberá mostrar una interfaz de usuario de carga, o en un estado de visualización de datos cuando se muestran adecuadamente los resultados obtenidos.



Capas de proceso de MVI:

- **Intent Interpretator:** Es responsable de interpretar las intenciones del usuario y convertirlas en acciones del programa. Es posible que varias intenciones de usuario se interpreten como una misma acción, lo que facilita la reutilización de flujos y código.
- **Processor:** Es la capa de procesamiento donde se encuentra la lógica de negocio. Es la única capa que interactúa con la clase de repositorio de datos del Frontend, esta clase es la que se comunica con el Backend. En esta capa, se realizan llamadas a la capa de datos para acceder a servicios de red o bases de datos locales, y se ejecutan las acciones del programa para generar resultados.
- **Reducer:** Es responsable de procesar los resultados obtenidos al ejecutar las acciones del programa y emitir los estados de la aplicación. El Reducer se puede programar como una máquina de estados, definiendo qué resultados pueden emitirse en el estado actual y hacia qué nuevo estado se direcciona.
- **ViewModel:** El ViewModel en MVI es similar al concepto de ViewModel en MVVM, pero además se encarga de observar las intenciones de usuario emitidas y generar el flujo descrito en la figura 6.4. La vista observa los estados emitidos desde el ViewModel para realizar el renderizado de la interfaz.

La capa de interfaz de usuario se suscribe a los eventos del ViewModel, emite intenciones y observa los estados resultantes del flujo para renderizar la vista de manera adecuada. Al utilizar este enfoque de MVI, se busca una mejor separación de responsabilidades y una arquitectura más clara y escalable en el desarrollo de aplicaciones Android.

Por lo tanto, el flujo de trabajo en MVI es el siguiente, y se diagrama en la figura 6.4:

1. El usuario interactúa con la vista, como hacer clic en un botón o introducir datos en un formulario.
2. La vista convierte estas interacciones en intenciones y las envía al ViewModel.

3. El ViewModel empieza el flujo descrito en la figura 6.4.
4. El Intent Interpretator toma las intenciones de usuario para generar acciones del programa.
5. El Processor toma estas intenciones para ejecutar lógica de negocio e interacción con la capa de data, esto genera resultados.
6. Los resultados pasan por el Reductor el cual actúa como máquina de estados, generando el siguiente estado a renderizar de la aplicación.
7. El nuevo estado se refleja en la vista, que actualiza la interfaz de usuario para mostrar los cambios correspondientes.
8. La vista vuelve a esperar nuevas interacciones del usuario.

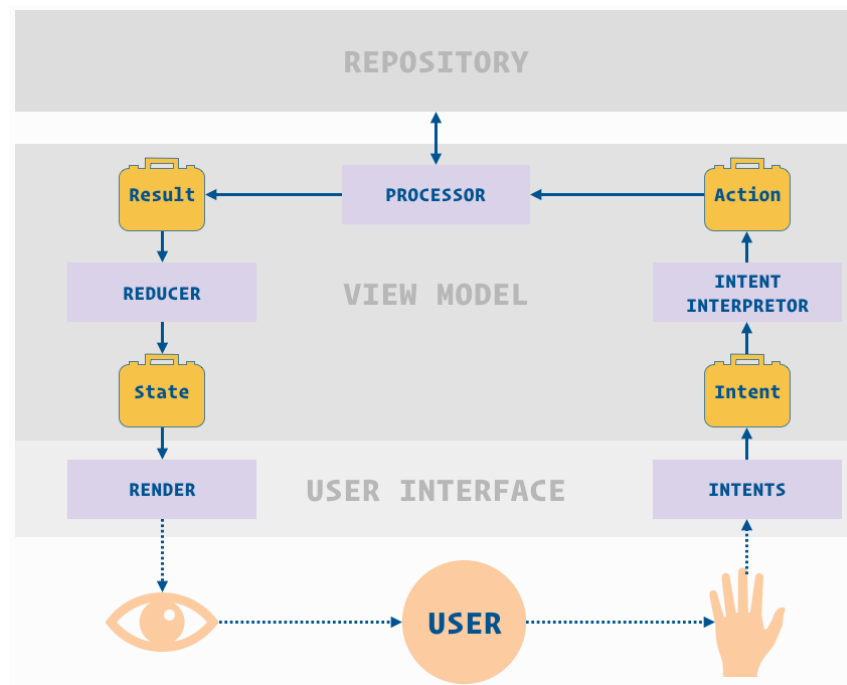


Figura 6.4: Patrón MVI.

La principal ventaja del patrón MVI es que proporciona un flujo de datos unidireccional y predecible. Al tener un único punto de entrada para las intenciones y un único punto de salida para el estado actualizado del modelo, se evitan los efectos secundarios y se simplifica la comprensión y depuración de la lógica de la aplicación. Además, el patrón MVI se presta bien para



la implementación de patrones reactivos y funcionales, como RxJava o Kotlin Coroutines.

Es importante mencionar que el patrón MVI puede requerir más esfuerzo inicial en comparación con otras arquitecturas, debido a la necesidad de definir las intenciones y el reductor. Sin embargo, a medida que la aplicación crece en tamaño y complejidad, el patrón MVI puede facilitar el mantenimiento, la prueba y la evolución de la aplicación.

Navegación

Usar *Navigation Component* facilita la gestión de fragmentos y la navegación entre ellos, proporcionando varias ventajas sobre las alternativas tradicionales [19].

Una Actividad por Concepto Se utiliza una actividad por concepto en lugar de una sola actividad para toda la aplicación o el uso de actividades para cada pantalla. Por ejemplo, la pantalla principal (*Home*) y el flujo de agregar mascotas tienen cada una su propia actividad. Dentro de estas actividades, se utilizan múltiples fragmentos para manejar diferentes partes del flujo.

- **Actividad Principal:** La actividad principal (*MainActivity*) actúa como el contenedor principal para los fragmentos relacionados con el home.
- **Flujo de Agregar Mascotas:** Este flujo tiene su propia actividad (*AddPetActivity*), que contiene varios fragmentos para cada paso del proceso de agregar una mascota. Cada fragmento maneja una parte específica del flujo, como la entrada de datos de la mascota, la carga de fotos y la confirmación.

Ventajas del Navigation Component El *Navigation Component* de Android ofrece varias ventajas en comparación con las alternativas tradicionales como *FragmentManager* o manejar la navegación manualmente:

- **Gráfico de Navegación:** Permite definir un gráfico de navegación en un archivo XML, lo que facilita la visualización y gestión de todas las rutas de navegación en la aplicación.



- **Transiciones Simplificadas:** Gestiona automáticamente las transiciones entre fragmentos, incluyendo animaciones y acciones de navegación.
- **Back Stack Gestionado:** Maneja automáticamente el back stack de fragmentos, simplificando la navegación hacia atrás y asegurando que las transiciones sean coherentes.
- **Argumentos de Navegación:** Facilita el paso de datos entre fragmentos utilizando un tipo seguro, lo que reduce el riesgo de errores y mejora la legibilidad del código.
- **Deep Linking:** Soporta enlaces profundos (deep linking), permitiendo a los usuarios navegar directamente a partes específicas de la aplicación desde notificaciones, correos electrónicos u otras aplicaciones.

Implementación en la Aplicación En la implementación, cada actividad principal tiene un *NavHostFragment* que actúa como un contenedor para los fragmentos gestionados por el *Navigation Component*. El gráfico de navegación se define en un archivo XML, especificando todas las rutas y acciones posibles dentro de esa actividad.

<!-- Ejemplo de un gráfico de navegación en XML -->

```
<navigation xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/nav_pets_graph"
    app:startDestination="@+id/addFragment">

    <fragment
        android:id="@+id/addFragment"
        android:name="com.digipaw.app.features.pet.add.ui.add.AddFragment"
        android:label="Añadir mascota"
        tools:layout="@layout/fragment_pet_add">

        <action
            android:id="@+id/from_add_to_description"
```



```
        app:destination="@+id/petDescriptionFragment" />
    </fragment>

    <fragment
        android:id="@+id/petDescriptionFragment"
        android:name="com.digipaw.app.features.pet.add.ui.add.PetDescriptionFragment"
        android:label="Añade descripción de mascota"
        tools:layout="@layout/fragment_pet_description">

        <argument
            android:name="petName"
            app:argType="string"
            app:nullable="false"/>

        <action
            android:id="@+id/from_add_to_success"
            app:destination="@+id/successFragment"
            app:popUpTo="@+id/nav_pets_graph"
            app:popUpToInclusive="true"/>
    </fragment>

    <fragment
        android:id="@+id/successFragment"
        android:name="com.digipaw.app.features.pet.add.ui.add.SuccessFragment"
        android:label=""
        tools:layout="@layout/fragment_pet_add_success"/>

</navigation>
```

En la figura 6.5 se muestra la visualización del grafo de navegación construido en XML y automáticamente generado por Android Studio.

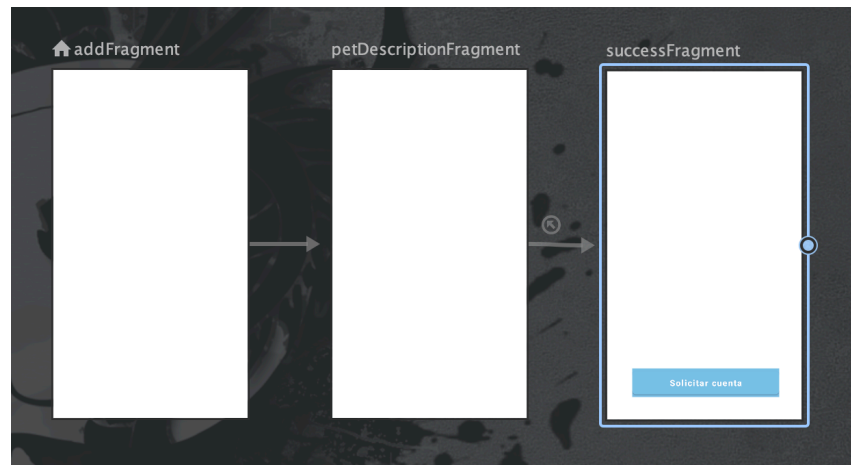


Figura 6.5: Visualización del gráfico de navegación hecho en xml en Android Studio.

Usar el *Navigation Component* en una aplicación Android permite gestionar la navegación de manera más organizada y visual. Al utilizar una actividad por concepto y fragmentos para las diferentes partes del flujo, se mejora la modularidad y mantenibilidad del código. Las ventajas del *Navigation Component*, como la simplificación de transiciones, gestión del back stack y soporte para deep linking, lo hacen una herramienta efectiva para gestionar la navegación en aplicaciones Android.

Resumen

La preparación base de la aplicación Android incluye la configuración de módulos de Gradle, la implementación de la conexión a servidores mediante un módulo de networking y el uso de Retrofit para las llamadas de red. El flujo de login se maneja mediante interceptores que añaden el token de autenticación a cada solicitud. La inyección de dependencias se realiza manualmente, y la arquitectura sigue el patrón MVI. Finalmente, Navigation Component se utiliza para gestionar la navegación dentro de la aplicación.

6.3. Construcción de un flujo

Para desarrollar un flujo en la aplicación, como por ejemplo agregar una mascota, se sigue una serie de pasos organizados según la arquitectura limpia (*Clean Architecture*). Esta arquitectura se basa en la separación de responsabilidades en capas, donde cada capa tiene un propósito



específico y conoce únicamente las capas adyacentes. La aplicación sigue este concepto, aunque se omite la capa de dominio ya que toda la lógica de negocio se maneja en el Backend For Frontend (BFF).

Capa de Data

La construcción de un flujo comienza por la capa de datos. Aquí, se crean los JSON con ejemplos de respuesta del backend según el contrato definido. Esto permite realizar pruebas sin esperar a que el backend esté completamente implementado. Luego, se crea la estructura de carpetas para el flujo, con una carpeta llamada *data*.

- **Repository:** Esta es la única clase visible y utilizable por otras capas del flujo. Actúa como el punto de entrada a la capa de datos.
- **Remote:** Una interfaz que describe todas las operaciones de red que se pueden realizar.
- **RemoteImpl:** Implementación de la interfaz *Remote*. Utiliza la clase *WebService*, que obtiene la configuración de Retrofit desde la capa de *network*.
- **WebService:** Define los endpoints y los modelos JSON que representan los datos enviados y recibidos desde el backend.
- **Modelos:** Las clases de datos (*data class*) representan los objetos enviados por el backend en los JSON. Estas clases utilizan etiquetas como `@SerializedName` para vincular un atributo del *data class* con uno del JSON enviado por el backend.

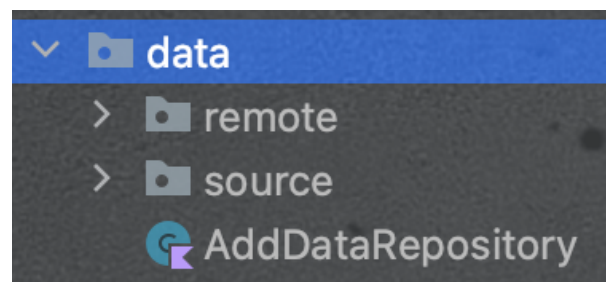


Figura 6.6: Estructura de paquetes de la capa de data.

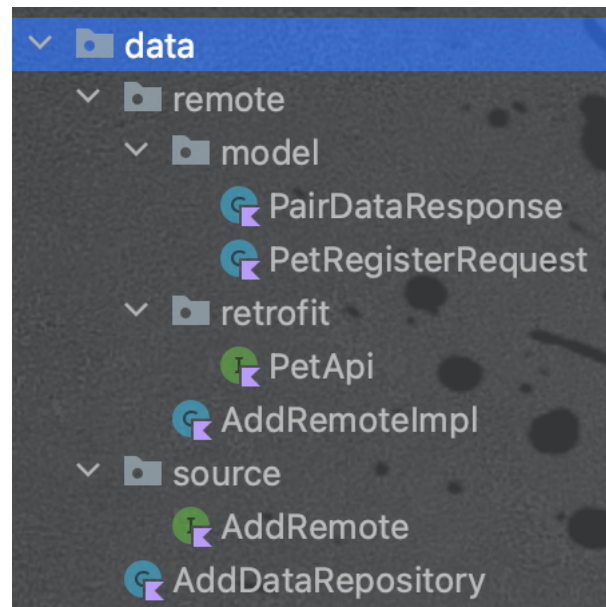


Figura 6.7: Estructura de paquetes de la capa de data, mostrando todas las clases.

Tests Unitarios de la Capa de Data

Después de implementar la capa de datos, es importante realizar pruebas unitarias a las clases correspondientes, como *Repository* y *RemoteImpl*. Estas pruebas aseguran que la lógica de obtención de datos y la comunicación con el backend funcionen correctamente, para estas pruebas generalmente en Android se utiliza la librería JUnit5 y mockk.

Capa de Presentación

En la capa de presentación, se crea el *ViewModel*, que sigue la arquitectura MVI (Model-View-Intent). Un *ViewModel* en Android actúa como un intermediario entre la interfaz de usuario y la capa de datos, gestionando los datos necesarios para la UI y manejando la lógica de presentación.

El *ViewModel* tiene un ciclo de vida que solo muere cuando el fragmento o la actividad que lo creó muere. Esto es beneficioso porque no depende de todos los cambios de ciclo de vida del fragmento, como rotaciones de pantalla o cambios de configuración, manteniendo los datos y la lógica de presentación persistentes.

- **ViewModel:** Llama al *Repository* de la capa de datos para realizar las llamadas necesarias. Utiliza *Flows* para que los datos puedan ser observados por la capa de UI.



La arquitectura MVI incluye varias clases clave que ayudan a gestionar los estados y eventos de la UI de manera estructurada.

UIIntent: *UIIntent* representa las intenciones del usuario. Son las acciones que el usuario desea realizar, como hacer clic en un botón o deslizar para actualizar. Cada intención se convierte en una acción que el *IntentInterpreter* procesa.

UIAction: *UIAction* representa la acción que debe tomar el programa debido a la intención del usuario. Mientras que un *UIIntent* podría ser *Ver mis mascotas*, una *UIAction* específica podría ser *Cargar datos de mascotas*. Las acciones son manejadas por el *Processor* para realizar operaciones específicas y obtener resultados.

UIResult: *UIResult* representa los resultados de las acciones realizadas por el *Processor*. Por ejemplo, después de una llamada de red para agregar una mascota, el *UIResult* podría ser éxito o error, junto con cualquier dato relevante.

UIState: *UIState* describe el estado actual de la pantalla. Esto incluye todos los datos que la UI necesita mostrar en un momento dado. El *ViewModel* emite nuevos estados a medida que los datos cambian o se actualizan en respuesta a las acciones del usuario.

UIEffect: *UIEffect* son eventos de una sola vez que la UI necesita manejar y que no generan un cambio en el estado de la pantalla, como mostrar un mensaje en un Toast o Snackbar¹ o navegar a una nueva pantalla. Los efectos son diferentes de los estados porque no son persistentes y deben manejarse inmediatamente.

Reducer: El *Reducer* toma el *UIResult* y el estado anterior (*UIState*) para crear un nuevo estado. Es una función pura que asegura que el estado de la UI se actualice de manera predecible en respuesta a los resultados de las acciones.

¹Mensajes emergentes breves que aparecen y desaparecen automáticamente.



Processor: El *Processor* maneja las *UIAction* y produce *UIResult*. Este componente realiza la lógica de negocio y las operaciones necesarias, como llamadas de red o procesamiento de datos, para cumplir con las acciones solicitadas.

Estas clases trabajan juntas para asegurar que la lógica de presentación esté claramente separada de la lógica de negocio y la manipulación de datos, permitiendo una UI reactiva y manejable.

En algunas ocasiones, también se pueden necesitar clases para manejar datos en la capa de presentación. En estos casos, es necesaria la creación de clases *mapper*, que transforman los datos recolectados en la capa presentación a clases específicas, adecuadas para hacer envíos de red según las necesidades del backend. Del mismo modo, puede ser necesario transformar datos de red en modelos de presentación.

Toda la estructura de paquetes y clases se ejemplifica en la figura 6.8.

Tests Unitarios de la Capa de Presentación Al igual que con la capa de datos, es esencial realizar pruebas unitarias en la capa de presentación. Se prueban los *ViewModels* para asegurarse de que gestionen correctamente los datos y la lógica de presentación.

Creación de Componentes

En el módulo de componentes reutilizables (*uicomponents*), se crean componentes que pueden ser reutilizados y adaptados según las necesidades de las distintas pantallas. La etiqueta `<merge/>` en XML se utiliza para evitar la creación de layouts adicionales innecesarios, optimizando la jerarquía de vistas.

- **Etiqueta `<merge/>`:** Permite que los elementos definidos dentro de `<merge/>` se inserten directamente en el lugar donde se infla el layout, evitando la creación de contenedores adicionales.

Actividad y Fragmentos

En la arquitectura de una aplicación Android, las actividades y fragmentos juegan un papel crucial en la construcción de la interfaz de usuario.

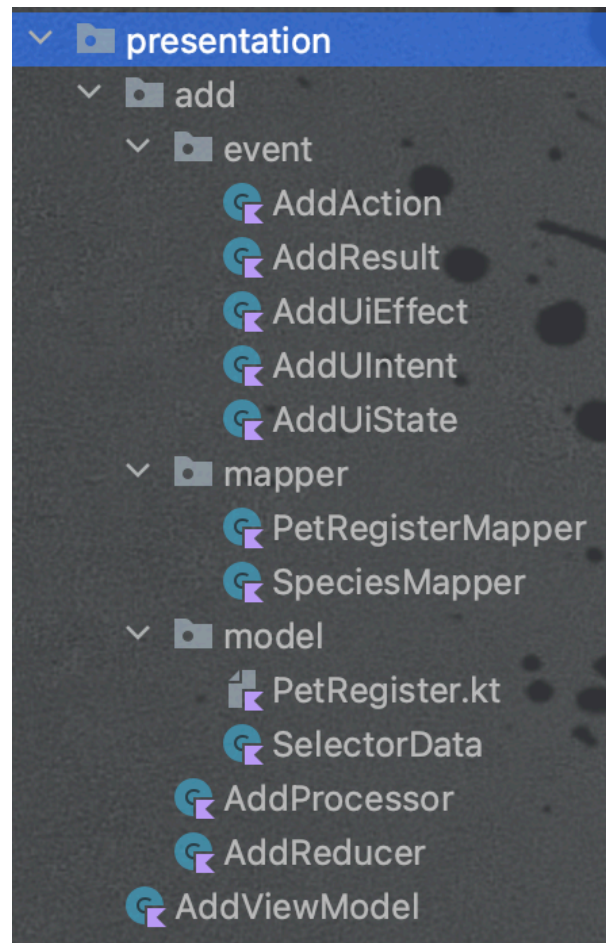


Figura 6.8: Estructura de paquetes de la capa de presentación, mostrando todas las clases.

Actividad

Una *Actividad* es un componente fundamental de una aplicación Android que proporciona una pantalla con la que los usuarios pueden interactuar para realizar una tarea específica. Es el punto de entrada para la interacción del usuario y actúa como un contenedor para varios fragmentos. Cada actividad en la aplicación tiene su propio ciclo de vida, que incluye estados como creación, inicio, reanudación, pausa, detención y destrucción.

En el contexto de un flujo específico, como agregar una mascota, la actividad actúa como el contenedor principal que orquesta y maneja la navegación entre los diferentes fragmentos que componen el flujo.



Fragmento

Un *Fragmento* es un componente modular y reutilizable de una interfaz de usuario dentro de una actividad. Los fragmentos representan una porción de la interfaz de usuario en una actividad y permiten una interfaz de usuario más flexible y dinámica. Cada fragmento tiene su propio ciclo de vida, que está estrechamente vinculado al ciclo de vida de la actividad que lo contiene.

La principal diferencia entre una actividad y un fragmento radica en que los fragmentos no pueden existir por sí solos; siempre deben estar contenidos dentro de una actividad. Esto permite que múltiples fragmentos sean combinados en una sola actividad para crear una interfaz de usuario multi-panel, común en dispositivos con pantallas grandes como tablets.

Implementación en el Flujo

Para el flujo de agregar una mascota, se puede estructurar de la siguiente manera:

- **Actividad Principal:** Actúa como un contenedor para los fragmentos y maneja la navegación entre ellos. Por ejemplo, *AddPetActivity* es la actividad que contiene todos los fragmentos necesarios para el flujo de agregar una mascota.
- **Fragmentos:** Cada paso del flujo se maneja mediante un fragmento. Por ejemplo:
 - *PetDetailsFragment*: Recoge los detalles básicos de la mascota.
 - *PetPhotoFragment*: Permite al usuario cargar una foto de la mascota.
 - *PetConfirmationFragment*: Muestra un resumen de los datos ingresados antes de confirmar la adición de la mascota.

Comunicación entre Fragmentos

La comunicación entre fragmentos se maneja a través de la actividad que los contiene. La actividad actúa como un intermediario, recibiendo datos de un fragmento y pasándolos a otro. Esto asegura una separación clara de responsabilidades y facilita el manejo del ciclo de vida de los fragmentos.



Navegación

Se utiliza el *Navigation Component* para gestionar la navegación dentro del flujo. Se define un gráfico de navegación (*navgraph*) y se utilizan *directions* para navegar entre fragmentos.

- **NavGraph:** Define las rutas y acciones de navegación en un archivo XML.
- **Directions:** Son clases generadas automáticamente por el *Navigation Component* que facilitan la navegación entre fragmentos de manera segura y tipada. Estos *directions* contienen la información necesaria para navegar a un destino específico, incluyendo cualquier parámetro que deba pasarse entre fragmentos.

Interacciones de UI

Finalmente, se refinan todas las interacciones de usuario necesarias para que la pantalla funcione correctamente. Esto incluye mostrar errores, manejar comportamientos específicos y realizar llamadas de red adicionales según sea necesario.

- **Errores:** Mostrar mensajes claros y útiles para el usuario.
- **Comportamientos Específicos:** Manejar interacciones y eventos particulares en la UI.
- **Llamadas de Red Adicionales:** Realizar solicitudes adicionales según las necesidades del flujo.

Para construir un flujo en una aplicación Android utilizando *Clean Architecture*, se sigue un proceso estructurado que incluye la creación de la capa de datos, la capa de presentación, sus respectivos tests unitarios, componentes reutilizables, actividades y fragmentos, y la navegación. Este enfoque asegura que cada parte de la aplicación esté bien definida y sea fácil de implementar, mantener y probar.



Capítulo 7

Publicación a Producción

En este capítulo se describe el proceso de publicación de la aplicación Digipaw, que incluye la configuración de instancias EC2 en Amazon Web Services (AWS) para el alojamiento del sistema y el uso de buckets S3 para el almacenamiento de imágenes y archivos. Además, se implementa la solución utilizando contenedores Docker para facilitar la gestión y despliegue del servicio. Aunque la publicación en Google Play Store se planifica, no se concreta en esta etapa del proyecto. El enfoque se centra en asegurar un entorno de producción estable y funcional que permita un crecimiento futuro de la aplicación.

7.1. Creación de un Bucket S3 en Amazon Web Services

Para el almacenamiento de imágenes en la aplicación, se decidió utilizar Amazon Web Services (AWS). AWS es una plataforma de servicios en la nube que ofrece una amplia gama de servicios a empresas y desarrolladores. Uno de estos servicios es Amazon S3 (Simple Storage Service), que proporciona almacenamiento escalable y de alta disponibilidad para datos y archivos.

La necesidad de utilizar AWS S3 surge debido a que la aplicación requiere guardar fotos de mascotas para mostrarlas en sus perfiles, así como fotos de los stickers de vacunas. A continuación, se detalla el proceso de creación y configuración de un bucket S3 para almacenar estas imágenes.

Proceso de Creación de un Bucket S3

1. **Acceder a la consola de AWS:** Se debe iniciar sesión en la consola de administración de AWS en <https://aws.amazon.com/console/>.



2. **Crear un nuevo bucket S3:** Se navega a la sección de S3 y se selecciona *Create bucket*. Es necesario proporcionar un nombre único para el bucket y seleccionar la región donde se desea crear el bucket. Para este proyecto, se creó un bucket llamado digipaw-images.
3. **Configurar el bucket:** Durante el proceso de creación, se pueden configurar opciones como el control de versiones, el cifrado y las políticas de acceso. Estas configuraciones aseguran que los datos sean manejados de manera segura.
4. **Crear carpetas dentro del bucket:** Una vez creado el bucket, se pueden crear carpetas dentro de él para organizar los diferentes tipos de imágenes. En este caso, se crearon las carpetas user-profile, pet-profile y health para almacenar las imágenes de perfil de usuario, perfil de mascota y stickers de vacunas respectivamente.

En la figura 7.1, se muestra la visualización del bucket digipaw-images con las carpetas creadas:

La imagen muestra la interfaz de usuario de Amazon S3 para el bucket 'digipaw-images'. En la parte superior, hay una barra de navegación con pestañas: 'Objetos' (seleccionada), 'Propiedades', 'Permisos', 'Métricas', 'Administración' y 'Puntos de acceso'. Debajo de las pestañas, se muestra el título 'Objetos (3)' y una descripción: 'Los objetos son las entidades fundamentales que se almacenan en Amazon S3. Puede utilizar el [inventario de Amazon S3](#) de forma explícita. [Más información](#)'. Hay un campo de búsqueda con el texto 'Buscar objetos por prefijo'. A continuación, se muestra una tabla con tres filas, cada una representando una carpeta creada.

☐	Nombre	Tipo
☐	health/	Carpeta
☐	pet-profile/	Carpeta
☐	user-profile/	Carpeta

Figura 7.1: Visualización del bucket digipaw-images con las carpetas user-profile, pet-profile y health.



Dentro de cada carpeta, se pueden almacenar los archivos de imágenes correspondientes. Por ejemplo, en la carpeta pet-profile, se almacenan las imágenes de perfil de las mascotas. En la figura 7.2, se muestra una visualización de los archivos dentro de esta carpeta:

Objetos (47) Información				Actualizar	Copiar URI de S3	Copiar URL	Descarga
Los objetos son las entidades fundamentales que se almacenan en Amazon S3. Puede utilizar el inventario de Amazon S3 para obtener una lista de todos los objetos de forma explícita. Más información							
Buscar objetos por prefijo							
☐	Nombre	Tipo	Última modificación				
☐	IMG_1713931439468_IMG_20240424_000348428.jpg	jpg	24 Apr 2024 12:04:01 AM -04				
☐	IMG_1715410210202_IMG_20240511_024731458.jpg	jpg	11 May 2024 2:50:12 AM -04				
☐	IMG_1715415538036_IMG_20240511_041813819.jpg	jpg	11 May 2024 4:19:00 AM -04				
☐	IMG_1715416976704_IMG_20240511_044243267.jpg	jpg	11 May 2024 4:42:59 AM -04				
☐	IMG_1715418200368_IMG_20240511_050257638.jpg	jpg	11 May 2024 5:03:22 AM -04				
☐	IMG_1715469316153_IMG_20240511_191415849.jpg	jpg	11 May 2024 7:15:18 PM -04				
☐	IMG_1715469343203_IMG_20240511_191415849.jpg	jpg	11 May 2024 7:15:44 PM -04				
☐	IMG_1715894622597_IMG_20240516_212203820.jpg	jpg	16 May 2024 5:23:44 PM -04				

Figura 7.2: Archivos de imágenes dentro de la carpeta pet-profile.

Configuración y Acceso a los Archivos

Para acceder y gestionar estos archivos desde la aplicación, se utiliza el SDK de AWS para integrar las funcionalidades de S3. Este SDK proporciona métodos para cargar, descargar y eliminar archivos en el bucket S3. Además, se configuran políticas de acceso para garantizar que solo los usuarios autorizados puedan realizar estas operaciones.

En el desarrollo con Spring Boot y Kotlin, se configura el acceso a AWS S3 mediante una clase de configuración que inicializa el cliente de Amazon S3 utilizando las credenciales proporcionadas:

```
@Configuration
class AwsConfiguration {

    @Value("${aws.accessKeyId}")
    private lateinit var accessKeyId: String

    @Value("${aws.secretKey}")
    private lateinit var secretKey: String

    @Bean
    fun amazonS3(): AmazonS3 {
        val credentials = BasicAWSCredentials(accessKeyId, secretKey)
        return AmazonS3ClientBuilder.standard()
            .withCredentials(AWSStaticCredentialsProvider(credentials))
            .withRegion(Regions.US_EAST_2)
            .build()
    }
}
```

Los archivos que se suben a S3 son usualmente de tipo *Multipart*. Un archivo *Multipart* es un archivo que se envía en varias partes, permitiendo que se manejen archivos grandes de manera eficiente. En el contexto de la aplicación, se utiliza un servicio para manejar la creación y subida de archivos *Multipart*:

```
@Service
class S3Service(private val amazonS3: AmazonS3, @Value("${s3.bucket}")
    ↪ private val bucketName: String) {

    fun createAndUploadFile(profilePhoto: MultipartFile, dir: S3Folder):
        ↪ String? {
        return try {
            val fileName = "IMG_${System.currentTimeMillis()}_${profilePhoto
                ↪ .originalFilename}"
            val key = "${dir.dir}/${fileName}"
            val putObjectRequest = PutObjectRequest(bucketName, key,
                ↪ profilePhoto.inputStream, ObjectMetadata())
```



```
        amazonS3.putObject(putObjectRequest)
        amazonS3.getUrl(bucketName, key).toString()
    } catch (e: Exception) {
        e.printStackTrace()
        null
    }
}

companion object {
    enum class S3Folder(val dir: String) {
        PET_PROFILE_FOLDER("pet-profile"),
        USER_PROFILE_FOLDER("user-profile"),
        VACCINE_STICKER_FOLDER("health/vaccine/sticker"),
    }
}
}
```

Esta configuración permite cargar archivos a las diferentes carpetas dentro del bucket S3 y obtener la URL de los archivos subidos para su uso en la aplicación. En resumen, el uso de AWS S3 facilita el almacenamiento y gestión de imágenes en la nube, proporcionando una solución robusta y escalable para las necesidades de la aplicación.

7.2. Implementación de Servidores en Amazon EC2

Para realizar pruebas con usuarios y permitir que todos puedan acceder a la aplicación, se decidió desplegar los servidores en Amazon Web Services (AWS) utilizando EC2. EC2 (Elastic Compute Cloud) es un servicio que permite crear y gestionar instancias de servidores virtuales en la nube, ofreciendo escalabilidad y control sobre los recursos utilizados.

Amazon EC2 proporciona una capacidad de procesamiento escalable en la nube, lo que permite ajustar los recursos según las necesidades del proyecto. A continuación, se detalla el proceso para crear y configurar una instancia EC2 para el proyecto Digipaw.



Proceso de Creación de una Instancia EC2

1. **Acceder a la consola de AWS:** Se inicia sesión en la consola de administración de AWS en <https://aws.amazon.com/console/> y se navega a la sección EC2.
2. **Lanzar una nueva instancia:** Se selecciona *Launch Instance* y se elige una Amazon Machine Image (AMI), que es una plantilla que contiene la configuración del servidor (sistema operativo, aplicaciones, etc.). Para este proyecto, se utiliza una AMI de Ubuntu.
3. **Seleccionar el tipo de instancia:** Se elige el tipo de instancia basado en los recursos necesarios. Para proyectos con poco uso de memoria y procesamiento, se pueden utilizar instancias *micro* que son gratuitas durante el primer año. En este caso, debido al uso intensivo de memoria por Spring Boot y varios microservicios, se optó por una instancia *small*.
4. **Configurar detalles de la instancia:** Se configuran los detalles de la instancia, como el número de instancias, configuración de red y almacenamiento.
5. **Añadir almacenamiento:** Se añade almacenamiento adicional si es necesario, además del volumen raíz que se crea por defecto.
6. **Agregar tags:** Se añaden etiquetas para organizar y gestionar mejor las instancias.
7. **Configurar el grupo de seguridad:** Se configura el grupo de seguridad para definir las reglas de firewall que controlan el tráfico de red hacia y desde la instancia.
8. **Revisar y lanzar:** Se revisa la configuración y se lanza la instancia. Durante este paso, se creará o seleccionará un par de claves PEM, que se utilizarán para acceder de forma segura a la instancia.

En la figura 7.3, se muestra una imagen de la instancia Digipaw-Dev en ejecución en la consola de EC2:



Instancias (1/1) Información

Buscar instancia por atributo o etiqueta (case-sensitive) Todos los e... Estado de la instancia

Estado de la instancia = running

☒	Nombre	ID de la instancia	Estado de la instancia	Tipo de instancia	Comprobación de	Estado de la instancia	Zona de disponibilidad	DNS de IPv4 pública	Dirección IP...	IP elástica
☒	Digipaw-Dev	i-0f7f6dc82ed0277ec	En ejecución	t2.small	2/2 comprobador	Ver alarmas	sa-east-1a	ec2-18-231-27-23.sa-e...	18.231.27.23	-

i-0f7f6dc82ed0277ec (Digipaw-Dev)

Detalles Estado y alarmas Monitoreo Seguridad Redes Almacenamiento Etiquetas

▼ Resumen de instancia Información

ID de la instancia
i-0f7f6dc82ed0277ec (Digipaw-Dev)

Dirección IPv6
-

Tipo de nombre de anfitrión
Nombre de IP: ip-172-31-4-25.sa-east-1.compute.internal

Responder al nombre DNS de recurso privado IPv4 (A)
Dirección IP asignada automáticamente
18.231.27.23 [IP pública]

Rol de IAM
-

IMDSv2
Required

Dirección IPv4 pública
18.231.27.23 | dirección abierta

Estado de la instancia
En ejecución

Nombre DNS de IP privada (solo IPv4)
ip-172-31-4-25.sa-east-1.compute.internal

Tipo de instancia
t2.small

ID de VPC
vpc-0107441853cfe43d7

ID de subred
subnet-03c4ba5672362a23b

Instance ARN
arn:aws:ec2:sa-east-1:590183951434:instance/i-0f7f6dc82ed0277ec

Direcciones IPv4 privadas
172.31.4.25

DNS de IPv4 pública
ec2-18-231-27-23.sa-east-1.compute.amazonaws.com | dirección abierta

Direcciones IP elásticas
-

Hallazgo de AWS Compute Optimizer
Suscribirse a AWS Compute Optimizer para recibir recomendaciones | Más información

Nombre del grupo de Auto Scaling
-

▼ Detalles de la instancia Información

Figura 7.3: Instancia Digipaw-Dev en ejecución en Amazon EC2.

Acceso a la Instancia

Para acceder a la instancia EC2, se utiliza una conexión SSH. Las claves PEM generadas al crear la instancia se utilizan para este propósito. A continuación, se muestran los pasos para conectarse a la instancia:

1. Localizar el archivo de clave privada (.pem) en tu sistema local.
2. Asegurarse de que el archivo de clave tenga los permisos correctos con el comando:

```
chmod 400 "nombre-clave.pem"
```

3. Conectarse a la instancia utilizando el comando SSH proporcionado por AWS. Por ejemplo:

```
ssh -i "nombre-clave.pem" ubuntu@ec2-XX-XXX-XX-XXX.compute.amazonaws.com
```



En la figura 7.4, se muestra una imagen de la sección de conexión de la consola de AWS:

Conectarse a la instancia Información

Conéctese a la instancia i-Of7f6dc82ed0277ec (Digipaw-Dev) mediante cualquiera de estas opciones

Conexión de la instancia EC2

Administrador de sesiones

Cliente SSH

Consola de serie de EC2

ID de la instancia
i-Of7f6dc82ed0277ec (Digipaw-Dev)

1. Abra un cliente SSH.
2. Localice el archivo de clave privada. La clave utilizada para lanzar esta instancia es digipaw-dev.pem
3. Ejecute este comando, si es necesario, para garantizar que la clave no se pueda ver públicamente.
chmod 400 "digipaw-dev.pem"
4. Conéctese a la instancia mediante su DNS público:
ec2-18-231-27-23.sa-east-1.compute.amazonaws.com

Ejemplo:
ssh -i "digipaw-dev.pem" ubuntu@ec2-18-231-27-23.sa-east-1.compute.amazonaws.com

Nota: En la mayoría de los casos, el nombre de usuario adivinado es correcto. Sin embargo, lea las instrucciones de uso de la AMI para comprobar si el propietario de la AMI ha cambiado el nombre de usuario predeterminado de la AMI.

Figura 7.4: Instrucciones para conectarse a la instancia EC2 en AWS.

Una vez conectado, se tiene acceso a la terminal de la instancia donde se pueden realizar configuraciones y desplegar aplicaciones. En la figura 7.5, se muestra una captura de pantalla de la consola de la instancia:

```
→ pem ssh -i "digipaw-dev.pem" ubuntu@ec2-18-231-27-23.sa-east-1.compute.amazonaws.com
Welcome to Ubuntu 24.04 LTS (GNU/Linux 6.8.0-1009-aws x86_64)

* Documentation:  https://help.ubuntu.com
* Management:    https://landscape.canonical.com
* Support:        https://ubuntu.com/pro

System information as of Mon Jun 24 02:20:01 UTC 2024

System load:  0.0          Processes:            131
Usage of /:   97.7% of 6.71GB Users logged in:        0
Memory usage: 61%         IPv4 address for enX0: 172.31.4.25
Swap usage:   0%

⇒ / is using 97.7% of 6.71GB

* Ubuntu Pro delivers the most comprehensive open source security and
  compliance features.

  https://ubuntu.com/aws/pro

Expanded Security Maintenance for Applications is not enabled.

35 updates can be applied immediately.
To see these additional updates run: apt list --upgradable

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status

Last login: Sun Jun 23 00:04:29 2024 from 190.121.68.190
ubuntu@ip-172-31-4-25:~$ ls
certs  digipaw-compose  snap
```

Figura 7.5: Consola de la instancia EC2 mostrando el uso de recursos y la estructura de directorios.

Consideraciones Adicionales

Para el proyecto, se eligió utilizar instancias EC2 debido a su flexibilidad y escalabilidad, además de ser la opción más utilizada en general dentro de las empresas de TI. A continuación, se describen algunas consideraciones importantes:

- **Costos:** Mientras que las instancias *micro* son gratuitas durante el primer año, instancias más grandes, como las *small*, tienen un costo asociado. Es importante monitorear el uso y los costos asociados.
- **Escalabilidad:** Para mejorar la escalabilidad y gestión de los microservicios, se puede considerar, a futuro, el uso de herramientas como Kubernetes para orquestar contenedores y balancear cargas.
- **Seguridad:** Las claves PEM proporcionan un método seguro de acceso, pero es fundamental configurar grupos de seguridad y políticas de acceso adecuadas para proteger los recursos.



La implementación en EC2 permite que la aplicación esté accesible en la nube, facilitando pruebas con usuarios y asegurando que todos los componentes del sistema funcionen correctamente en un entorno de producción. Además, se puede monitorear la consola, para actuar en caso de errores o caídas de los servicios.

7.3. Implementación de Docker Compose

Docker Compose es una herramienta utilizada para definir y ejecutar aplicaciones Docker multicontenedor. Permite describir la configuración de todos los servicios necesarios en un solo archivo, facilitando el despliegue y gestión de aplicaciones complejas. En el proyecto, se decidió recientemente utilizar Docker Compose para simplificar el proceso de pruebas y preparar el entorno para su despliegue en Amazon EC2.

¿Qué es Docker Compose?

Docker Compose permite definir un entorno de múltiples contenedores utilizando un archivo YAML. Este archivo describe los servicios, redes y volúmenes que la aplicación necesita. Al usar Docker Compose, se puede levantar todo el entorno de desarrollo con un solo comando, evitando la necesidad de iniciar cada servicio manualmente.

Beneficios de Docker Compose

- **Simplificación del Desarrollo:** Facilita la configuración y ejecución de múltiples servicios necesarios para la aplicación.
- **Aislamiento de Entornos:** Cada servicio se ejecuta en su propio contenedor, lo que asegura un entorno de ejecución consistente.
- **Facilidad de Despliegue:** Permite replicar el entorno de desarrollo en producción con pocos cambios.



Configuración de Docker Compose en el Proyecto

Para el proyecto, se configuraron varios servicios utilizando Docker Compose. A continuación, se explica el proceso de configuración y se proporciona una descripción de los archivos Docker necesarios.

Dockerfile

Cada servicio del proyecto tiene su propio archivo Dockerfile, que define cómo se construye la imagen Docker del servicio. A continuación, se muestra un ejemplo de Dockerfile para el servicio de pets:

```
# Utiliza la imagen base oficial de OpenJDK 17
FROM openjdk:17-jdk-slim

# Establece el directorio de trabajo en /app
WORKDIR /app

ARG JAR_FILE

# Copia el archivo JAR del proyecto al contenedor
COPY ${JAR_FILE} /app/pets.jar

# Exponer el puerto en el que se ejecuta la aplicación Spring Boot
EXPOSE 8080

# Comando para ejecutar la aplicación
ENTRYPOINT ["java", "-jar", "/app/pets.jar"]
```

Archivo docker-compose.yml

El archivo `docker-compose.yml` define todos los servicios que componen la aplicación. A continuación, se muestra un fragmento del archivo utilizado en el proyecto:

```
services
  middleend
```



```
container_name digipaw-middleend

build
  context .
  dockerfile ./digipaw-middleend/Dockerfile
  args
    - JAR_FILE=/snaps/middle-end-0.0.1-SNAPSHOT.jar
env_file ./digipaw-middleend/.env
restart on-failure

ports
  - "${MIDDLEEND_PORT}:${MIDDLEEND_PORT}"

volumes
  - ./digipaw-middleend/src/usr/app/src
  - ./digipaw-middleend/.env/app/.env

networks
  - digipaw-network

auth

container_name digipaw-auth

build
  context .
  dockerfile ./digipaw-auth/Dockerfile
  args
    - JAR_FILE=/snaps/auth-0.0.1-SNAPSHOT.jar
env_file
  - ./digipaw-auth/.env
  - ./digipaw-auth/dev.jvm.conf
restart on-failure

ports
  - "${AUTH_PORT}:${AUTH_PORT}"

volumes
  - ./digipaw-auth/src/usr/app/src
  - ./digipaw-auth/.env/app/.env

networks
  - digipaw-network
```



Componentes del archivo `docker-compose.yml`

- **services:** Define cada uno de los servicios que componen la aplicación, incluyendo su configuración de construcción, variables de entorno, puertos expuestos, volúmenes y redes.
- **networks:** Define las redes que se utilizan para la comunicación entre los servicios. En este caso, se define una red llamada `digipaw-network`.

Uso de Docker Compose

Para levantar todos los servicios definidos en el archivo `docker-compose.yml`, se utiliza el siguiente comando:

```
docker-compose up --build
```

Este comando construye las imágenes Docker y levanta los contenedores, permitiendo que todos los servicios estén listos para su uso.

La implementación de Docker Compose en el proyecto Digipaw facilita el proceso de desarrollo y despliegue de la aplicación. Al definir todos los servicios en un solo archivo, se asegura que el entorno de desarrollo sea consistente y fácil de replicar. Aunque esta configuración no está completamente pulida, proporciona una base sólida para la gestión y despliegue de los servicios de la aplicación.

7.4. Publicación en Google Play Store

Google Play Store es la plataforma oficial de distribución de aplicaciones para dispositivos Android. Publicar una aplicación en Google Play Store permite llegar a una amplia audiencia de usuarios y facilita la instalación y actualización de la aplicación en los dispositivos de los usuarios.



Requisitos para Publicar en Google Play Store

Para utilizar Google Play Console y publicar una aplicación en Google Play Store, es necesario contar con una cuenta de desarrollador de Google. Esta cuenta tiene un costo único de \$25 USD. Google Play Console es una herramienta que permite gestionar todas las etapas de la publicación de una aplicación, desde la creación y pruebas hasta el despliegue y monitoreo.

Hace algunos años, el proceso de publicación en Google Play Store era más sencillo y requería solo subir las imágenes de la aplicación y contestar algunos formularios respecto a la aplicación y su audiencia. Sin embargo, con el cambio de políticas en 2022, este proceso se ha vuelto más riguroso.

Proceso de Publicación en Google Play Store

1. **Creación de la Cuenta de Desarrollador:** Se debe crear una cuenta de desarrollador en Google Play Console, pagando la tarifa única de \$25 USD.
2. **Configuración Inicial:** Completar la configuración inicial de la cuenta, incluyendo la verificación de identidad, la creación de una política de privacidad y el establecimiento de opciones de seguridad para la aplicación.
3. **Preparación del App Bundle:** Un App Bundle es un formato de publicación que contiene todos los recursos y código de la aplicación, optimizados para su distribución en Google Play Store, los Bundles, a diferencia de los APK optimizan mejor los recursos y son más seguros. Hace unos años Google dejó de permitir la subida de APKs en la Playstore, siendo obligatorio el uso de Bundles.
4. **Cargar del App Bundle:** Subir el App Bundle a Google Play Console. En la sección de *Testing*, se puede optar por tres tipos de pruebas:
 - **Internal Testing:** Permite probar la aplicación internamente con un pequeño grupo de usuarios.
 - **Closed Testing:** Permite pruebas cerradas con un grupo más grande de usuarios.



- **Open Testing:** Permite pruebas abiertas a cualquier usuario interesado en probar la aplicación.
5. **Configuración de la Prueba:** Para este proyecto, se optó por *Internal Testing*, donde se especifican los correos electrónicos de los usuarios que van a probar la aplicación.
 6. **Completar Formularios:** Completar los formularios requeridos por Google, incluyendo la política de privacidad, la clasificación de contenido, la audiencia objetivo y la seguridad de datos. También se debe verificar que las llamadas de red estén encriptadas en tránsito y que haya una opción para borrar cuentas de usuario.

Create internal testing release



Drop app bundles here to upload

[Upload](#) [Add from library](#)

File type	Version	API levels	Target SDK	Screen layouts	ABIs	Required features	
App bundle	2 (0.0.1-dev)	23+	33	4	4	5	→

Release details

Release name *

v0.0.1

6 / 50

Release notes

[Copy from a previous release](#)

<es-419>
Primera versión de prueba de Digipaw que contiene:
- Crear cuenta
- Login
- Añadir Mascota

Figura 7.6: Subida de un App Bundle en Google Play Console.

La figura 7.6 muestra el proceso de carga de un App Bundle en Google Play Console. Este es el paso principal para preparar la aplicación para su distribución.

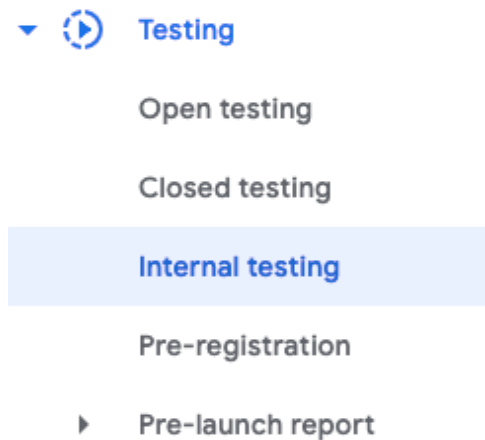


Figura 7.7: Opciones de pruebas en Google Play Console.

La figura 7.7 detalla las diferentes opciones de pruebas disponibles: *Internal Testing*, *Closed Testing* y *Open Testing*, que se detallaron anteriormente y que son parte de las opciones de la Playstore Console.

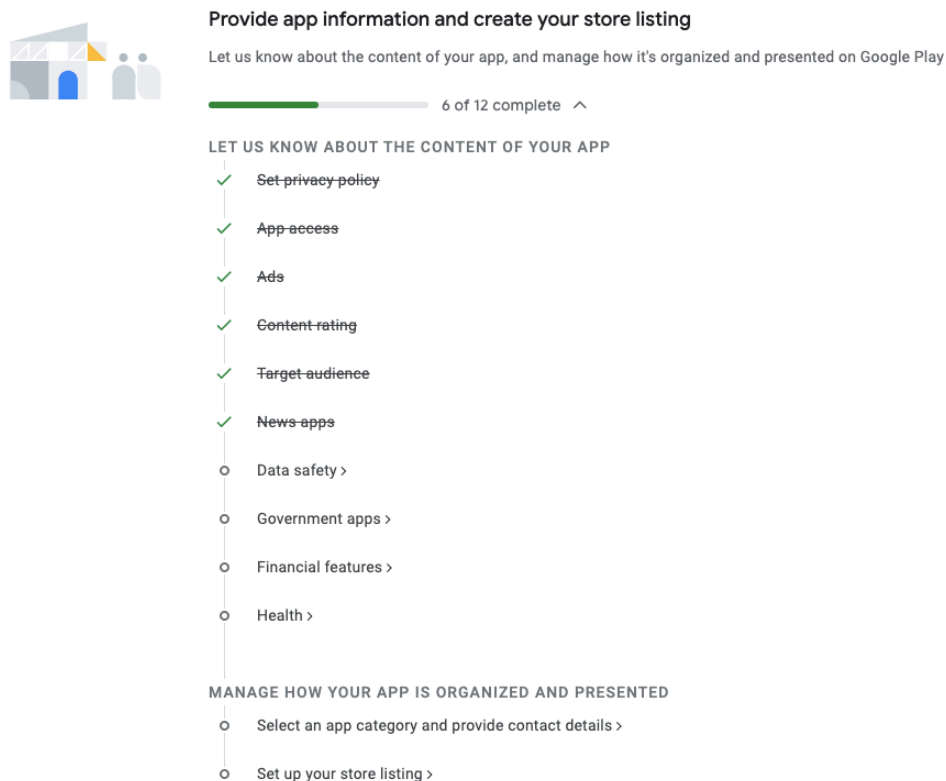


Figura 7.8: Formularios requeridos en Google Play Console.



La figura 7.8 muestra los formularios que deben completarse antes de que la aplicación pueda ser publicada, incluyendo la política de privacidad, la clasificación de contenido y la seguridad de datos.

En este proyecto, se encontraron algunos desafíos que impidieron la publicación en Google Play Store dentro del plazo establecido. Los requisitos adicionales, como la verificación de identidad y la implementación de políticas de privacidad, fueron parte de los trade-offs que se realizaron durante el desarrollo. Esto retrasó la publicación, ya que no se pudieron completar todos los pasos necesarios a tiempo.



Capítulo 8

Pruebas de Digipaw

Las pruebas de software son una parte importante del proceso de desarrollo, ya que aseguran que la aplicación cumpla con los criterios de funcionalidad y calidad definidos por el equipo de producto y UX. Existen diferentes tipos de pruebas que se implementan para validar distintos aspectos de la aplicación:

- **Pruebas Funcionales:** Estas pruebas se realizan manualmente y se centran en verificar que la aplicación cumple con los criterios de funcionalidad y UI definidos. Se comprueba que cada característica de la aplicación funcione como se espera.
- **Pruebas End-to-End (E2E):** En este caso, se realizan manualmente y consisten en probar el flujo completo de la aplicación, desde el frontend hasta el backend. Se verifica que todos los componentes del sistema funcionen correctamente juntos.
- **Pruebas Unitarias:** Estas pruebas son específicas del código y se enfocan en verificar la funcionalidad de unidades individuales del código, como funciones o métodos. Las pruebas unitarias son esenciales para detectar errores en etapas tempranas del desarrollo.
- **Pruebas con Usuarios Finales:** Se realizan en un entorno semi-productivo para validar que la solución implementada resuelva efectivamente el problema de los usuarios. Estas pruebas proporcionan feedback valioso sobre la usabilidad y efectividad de la aplicación.

8.1. Estrategia de Pruebas de Código

8.1.1. Pruebas Unitarias

Las pruebas unitarias son una técnica de prueba en la que se verifica el correcto funcionamiento de las unidades individuales de código, como funciones o métodos, de forma aislada del



resto del sistema. Estas pruebas son fundamentales para asegurar que cada parte del código funcione correctamente y ayudan a identificar errores en etapas tempranas del desarrollo.

En el desarrollo de aplicaciones Android, se utilizan librerías como JUnit 5 y MockK para realizar pruebas unitarias. JUnit 5 es un framework de pruebas unitarias para Java, mientras que MockK es una librería de mocking para Kotlin que permite simular el comportamiento de objetos y clases.

- **JUnit 5:** Permite definir y ejecutar pruebas unitarias de forma sencilla. Proporciona anotaciones como `@Test` para definir métodos de prueba y diversas utilidades para la aserción de resultados.
- **MockK:** Facilita la creación de mocks y stubs en Kotlin. Los mocks son objetos simulados que imitan el comportamiento de objetos reales, mientras que los stubs son implementaciones simplificadas que devuelven resultados predefinidos.

El proceso de una prueba unitaria implica crear una instancia del objeto a probar, definir los comportamientos esperados mediante mocks o stubs, ejecutar el método a probar y verificar que el comportamiento observado coincide con el comportamiento esperado.

A continuación, se muestra un ejemplo de código Kotlin y su prueba unitaria:

```
class RegisterRepository(private val remote: RegisterRemoteImpl) {  
    fun registerUser(remoteRegisterUserInfoParams:  
        RemoteRegisterUserInfoParams) =  
        flow {  
            val response = remote.registerUser(remoteRegisterUserInfoParams)  
            emit(response)  
        }  
}
```

```
internal class RegisterRepositoryTest {  
    private val remote = mockk<RegisterRemoteImpl>()  
  
    private val repository = RegisterRepository(remote)
```



```
@Test
fun `given remoteRegisterUserInfoParams, when registerUser, then save`()
    → = runBlocking {
        val remoteRegisterUserInfoParams = makeRemoteUserInfoParams()
        stubRegisterUser(remoteRegisterUserInfoParams)

        repository.registerUser(remoteRegisterUserInfoParams)

        coVerify { remote.registerUser(remoteRegisterUserInfoParams) }
    }

private fun stubRegisterUser(remoteInfo: RemoteRegisterUserInfoParams) {
    coEvery { (remote.registerUser(remoteInfo)) } returns Unit
}
}
```

En este ejemplo, se crea una prueba unitaria para el método `registerUser` de la clase `RegisterRepository`. Se utiliza `MockK` para crear un mock de `RegisterRemoteImpl`, ya que es una clase que se inyecta en `RegisterRepository`. Al hacer un mock de esta clase, se aísla la dependencia externa, permitiendo que la prueba se enfoque únicamente en la lógica de `RegisterRepository`. El comportamiento esperado se define utilizando `coEvery` y `coVerify`.

`coEvery` se utiliza para especificar que, cuando se llame al método `registerUser` con ciertos parámetros, el mock debe devolver un resultado específico o ejecutar una acción determinada. Esto es necesario para simular el comportamiento de la dependencia sin necesidad de ejecutar su lógica real, lo cual puede ser complejo o depender de factores externos.

Por otro lado, `coVerify` se usa para verificar que el método `registerUser` fue llamado con los parámetros esperados durante la ejecución de la prueba. Esto asegura que la interacción entre `RegisterRepository` y su dependencia se realice correctamente.

Las pruebas unitarias sirven para mantener la calidad del código y asegurar que cada componente funcione correctamente de manera aislada, lo que facilita la detección y corrección de errores en etapas tempranas del desarrollo.



8.1.2. Pruebas End-to-End (e2e)

Las pruebas end-to-end (e2e) son fundamentales para garantizar que todos los componentes de una aplicación funcionen correctamente de manera conjunta. Estas pruebas se realizan a lo largo de todo el proceso de desarrollo, no solo al final, para asegurar que las interacciones entre el frontend y el backend se comporten como se espera.

Para probar los endpoints del backend, se utiliza una herramienta llamada Insomnia. Insomnia es una plataforma de pruebas de API que permite configurar y ejecutar solicitudes HTTP a diversos endpoints, facilitando la verificación de las respuestas y el comportamiento del servidor. En la figura 8.1 se muestra cómo se ve la interfaz de esta herramienta.

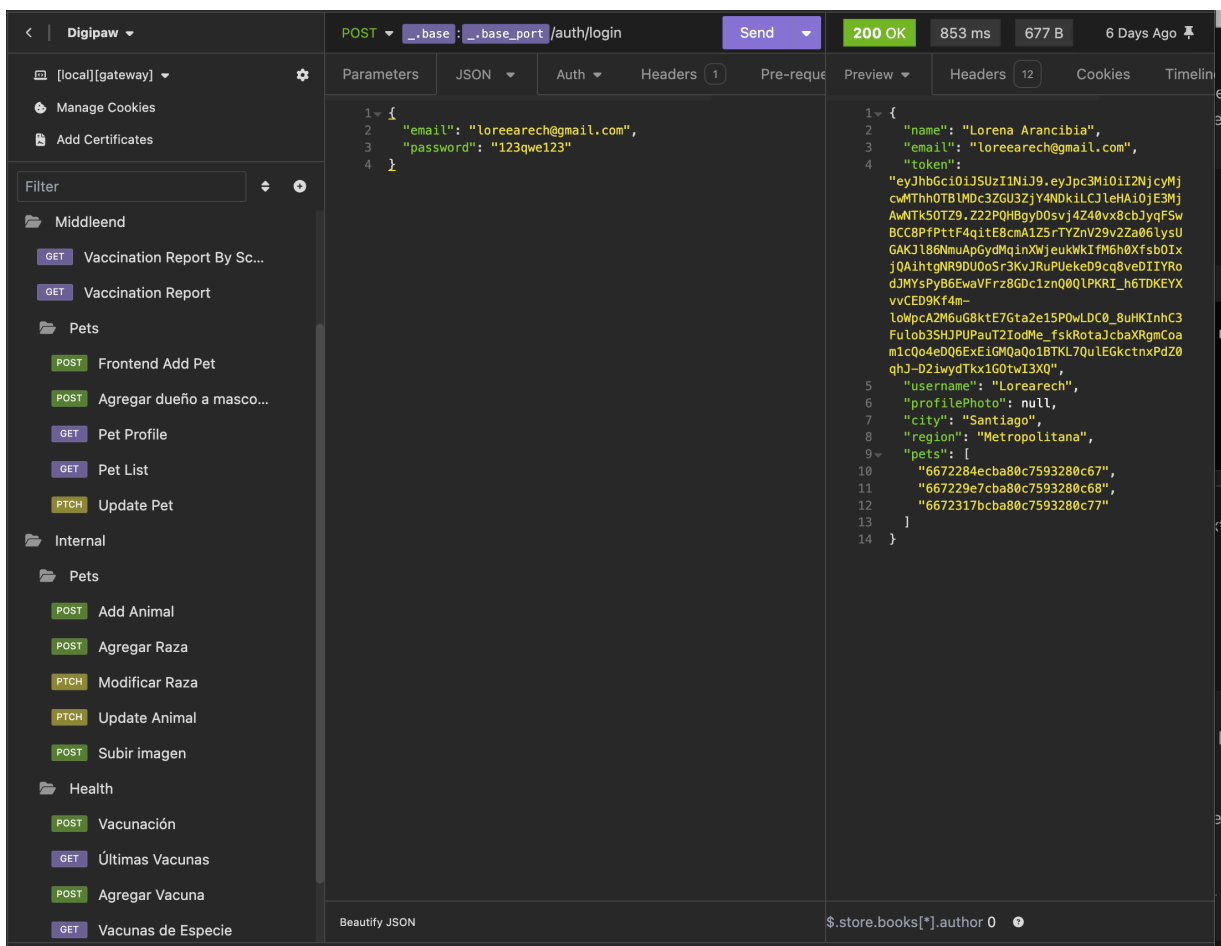


Figura 8.1: Interfaz de Insomnia mostrando la configuración y respuesta de un endpoint.

Insomnia se configura agregando nuevos endpoints a medida que se desarrollan. Se realizan

pruebas con diferentes datos para asegurarse de que las respuestas y las modificaciones en la base de datos sean correctas. Para verificar los cambios en la base de datos, se utiliza MongoDB Compass, una herramienta que proporciona una interfaz gráfica para visualizar y gestionar los datos en MongoDB, como se muestra en la figura 8.2.

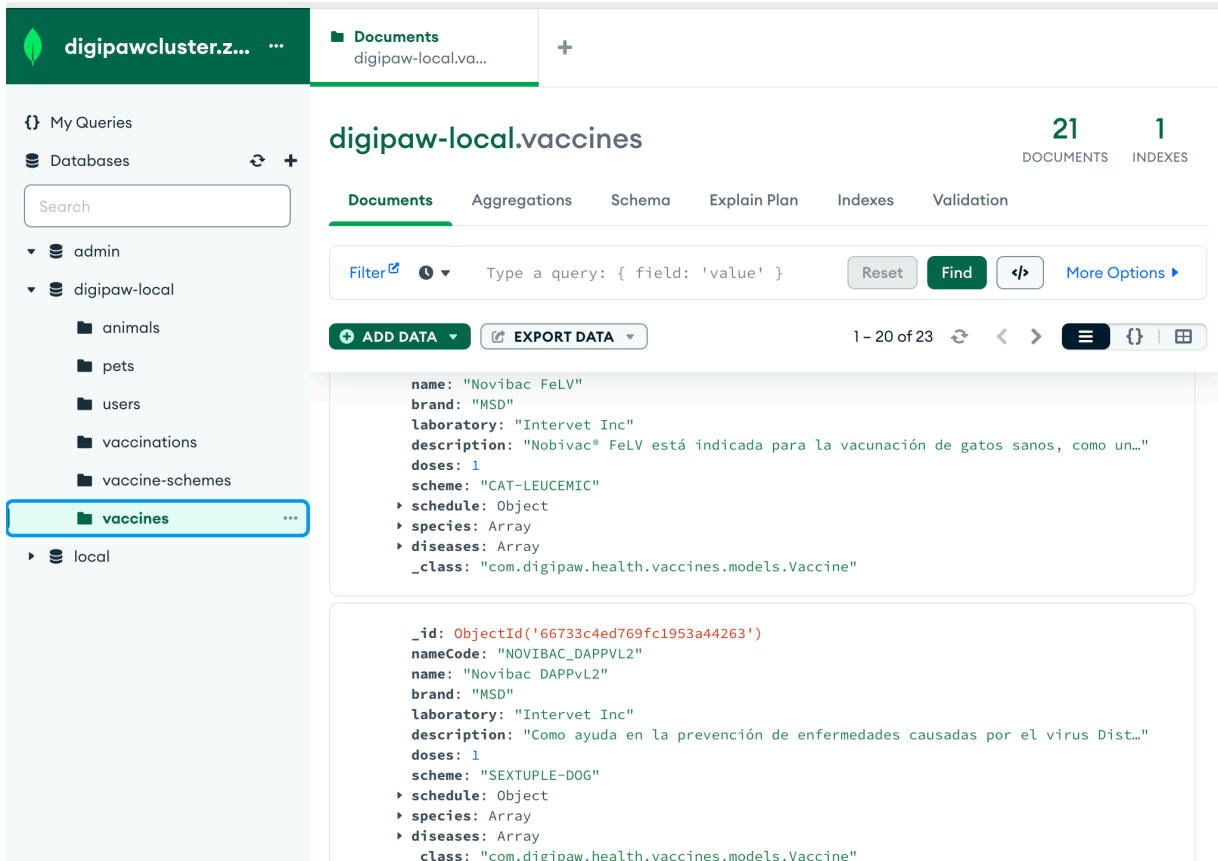


Figura 8.2: Visualización de datos en MongoDB Compass.

Una vez que se confirma que el backend funciona correctamente, se conecta a la aplicación móvil. Para monitorear las llamadas de red realizadas por la aplicación y las respuestas del backend, se utiliza Flipper [22]. Flipper es una herramienta de depuración para aplicaciones móviles desarrollada por Facebook. Permite inspeccionar las solicitudes de red, la base de datos local, el diseño de la interfaz de usuario y más, como se muestra en la figura 8.3.

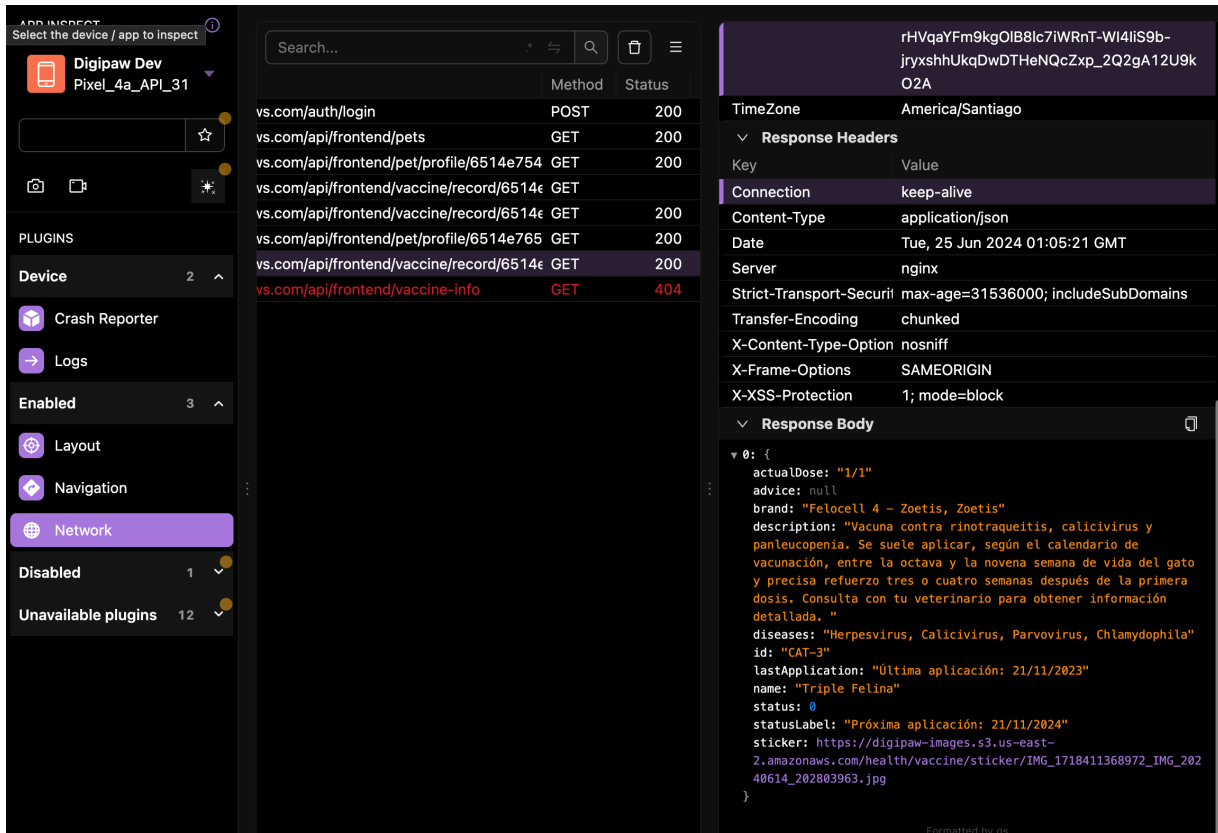


Figura 8.3: Interfaz de Flipper mostrando las llamadas de red y sus respuestas.

Para configurar Flipper en un proyecto de Android, se debe agregar la dependencia de Flipper en el archivo `build.gradle` y habilitar los plugins necesarios. Una vez configurado, Flipper proporciona una vista detallada de las actividades de la aplicación, permitiendo identificar y resolver problemas de manera eficiente.

Listing 8.1: Configuración de Flipper en `build.gradle`

```

dependencies {
    debugImplementation 'com.facebook.flipper:flipper:0.125.0'
    debugImplementation 'com.facebook.flipper:flipper-network-plugin:0.125.0'
    ↪
    releaseImplementation 'com.facebook.flipper:flipper-noop:0.125.0'
}

```

En resumen, las pruebas e2e implican probar los endpoints con Insomnia, verificar los datos en MongoDB Compass, y monitorear las llamadas de red en la aplicación móvil con Flipper. Este enfoque asegura que todas las partes del sistema funcionen correctamente y de manera in-



tegrada, proporcionando una mayor confianza en la calidad del software.

Si bien estas pruebas pueden no ser completamente automatizadas como en algunos entornos e2e, el uso combinado de estas herramientas permite una verificación exhaustiva y efectivos del funcionamiento del sistema.

8.2. Pruebas con usuarios y resultados

Las pruebas con un grupo reducido de usuarios son esenciales para obtener retroalimentación valiosa sobre cómo funciona, se siente y se ve la aplicación. Este proceso permite identificar problemas técnicos, recopilar opiniones sobre la experiencia del usuario y, lo más importante, validar si el producto pensado realmente sirve y resuelve un problema real para los usuarios finales. La retroalimentación obtenida durante estas pruebas sirve también para realizar ajustes y mejoras antes de lanzar la aplicación a un público más amplio.

8.2.1. Distribución para Pruebas de Desarrollo Internas

Inicialmente, se planeaba utilizar Google Play Store para distribuir la aplicación en un entorno de prueba interno. Sin embargo, debido a la priorización de funcionalidades críticas de la aplicación y el tiempo acotado, no fue posible publicar la aplicación de manera temprana en la plataforma. Por lo tanto, se optó por una alternativa: la creación de un grupo de WhatsApp para distribuir la aplicación.

Se creó un grupo de WhatsApp con personas interesadas en probar la aplicación, como se muestra en la figura 8.4. En este grupo, se proporcionaron las instrucciones necesarias y se compartió el archivo APK para que los usuarios pudieran descargarlo e instalarlo en sus dispositivos. Esta estrategia permitió un seguimiento en tiempo real, facilitando la resolución de problemas imprevistos y aclarando dudas directamente con los usuarios.

Para seleccionar a los participantes, se contactó inicialmente a personas cercanas a los integrantes del equipo. Luego, se amplió la convocatoria a personas del círculo extendido mediante



publicaciones en redes sociales. Con esta estrategia, se logró reunir a aproximadamente 20 familias con mascotas dispuestas a probar la aplicación.

A pesar de no utilizar Google Play Store, la distribución del APK a través de WhatsApp resultó efectiva y no presentó mayores inconvenientes. Esta metodología permitió obtener retroalimentación valiosa en un entorno controlado.

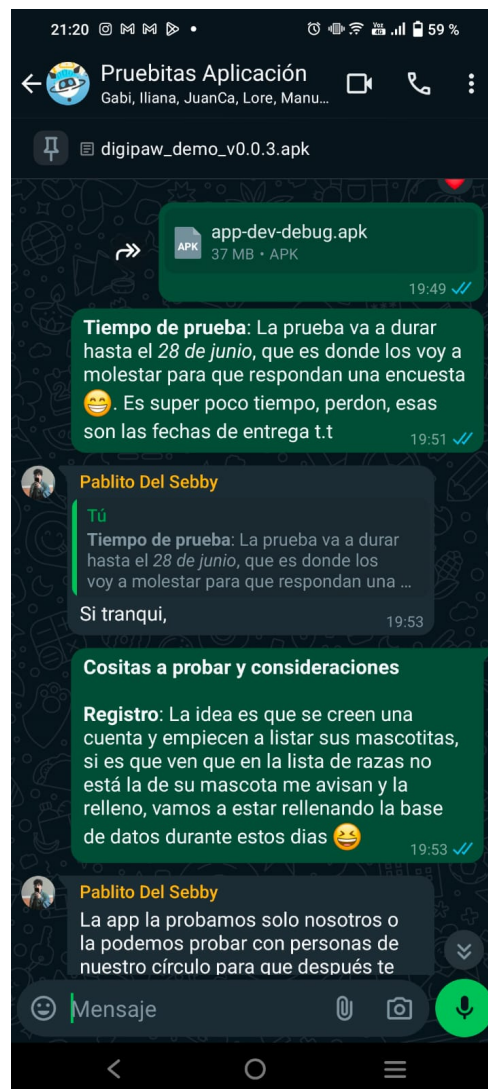


Figura 8.4: Grupo de WhatsApp utilizado para la distribución del APK y comunicación con los testers.



8.2.2. Preguntas

Para obtener retroalimentación estructurada de los usuarios, se utilizó Google Forms. Google Forms es una herramienta gratuita de Google que permite crear encuestas y formularios en línea de manera sencilla. Esta herramienta ofrece diversas bondades, como la posibilidad de recopilar respuestas en tiempo real, analizar datos automáticamente y exportar los resultados a una hoja de cálculo para un análisis más detallado.

Las preguntas del formulario se diseñaron para cubrir varios aspectos clave de la aplicación, desde la facilidad de uso hasta la utilidad percibida por los usuarios. A continuación, se explica el propósito de cada pregunta:

- **¿Qué edad tienes?** Permite identificar la demografía de los usuarios que probaron la aplicación.
- **¿Cuántas mascotas tienes?** Ayuda a comprender el contexto de uso de la aplicación y la experiencia previa de los usuarios con mascotas.
- **¿Qué tan fácil fue para ti crear una cuenta de usuario?** Evalúa la simplicidad y claridad del proceso de registro.
- **¿Qué tan fácil o intuitivo fue para ti agregar una nueva mascota?** Mide la usabilidad del flujo de agregar mascotas, un aspecto crucial para la funcionalidad de la aplicación.
- **¿Qué tan fácil o intuitivo fue para ti agregar una nueva vacuna?** Similar a la pregunta anterior, pero enfocada en la gestión de vacunas.
- **Si tuviste alguna dificultad al utilizar la aplicación, coméntanos cuál fue** Recopila información sobre posibles problemas o puntos de fricción en la experiencia del usuario.
- **¿Qué tan útil te resultó la información de historial de vacunas de tus mascotas que te proporciona la app?** Evalúa la relevancia y utilidad de una de las funcionalidades principales de la aplicación.
- **¿Qué tan útil te resulta la información que la app te muestra de tus mascotas?** Mide la percepción de valor de la información general proporcionada por la aplicación.



- **¿Crees que la aplicación pueda servirte en la gestión del control de vacunas de tus mascotas?** Verifica si los usuarios ven la aplicación como una herramienta valiosa para la gestión de vacunas.
- **Si colocaste que No en la pregunta anterior, ¿Qué le falta a la aplicación para que sientas que puede ser de utilidad en el control de vacunas de tus mascotas?** Recopila sugerencias específicas para mejorar la aplicación.
- **Si colocaste Si, ¿Con cuánta frecuencia crees que usarías la aplicación para gestionar las vacunas de tus mascotas?** Ayuda a entender la frecuencia potencial de uso de la aplicación.
- **¿Qué tan probable es que sigas usando la aplicación en el futuro?** Mide la intención de uso a largo plazo.
- **¿Qué sientes que se podría mejorar en la aplicación?** Recopila sugerencias generales para mejoras.
- **¿Qué aspectos de la aplicación te gustaron más?** Identifica las fortalezas percibidas de la aplicación.
- **¿Hay alguna función que te gustaría ver añadida en futuras versiones de la aplicación?** Recopila ideas para nuevas funcionalidades que podrían aumentar el valor de la aplicación.
- **¿Nos das un feedback general de tus impresiones con la aplicación?** Permite a los usuarios expresar cualquier comentario adicional que tengan.

En síntesis, las preguntas del formulario están diseñadas para recopilar una amplia gama de datos que permitan evaluar tanto la usabilidad como la utilidad de la aplicación. Esta información es crucial para identificar áreas de mejora, validar la propuesta de valor del producto y planificar futuras actualizaciones y mejoras de la aplicación.

8.2.3. Resultados

El análisis de las respuestas obtenidas a través del formulario de Google Forms proporciona una visión integral sobre la utilidad y aceptación de la aplicación por parte de los usuarios. A continuación, se presentan los resultados detallados de cada pregunta junto con gráficos que ilustran la distribución de las respuestas.

■ ¿Qué edad tienes?

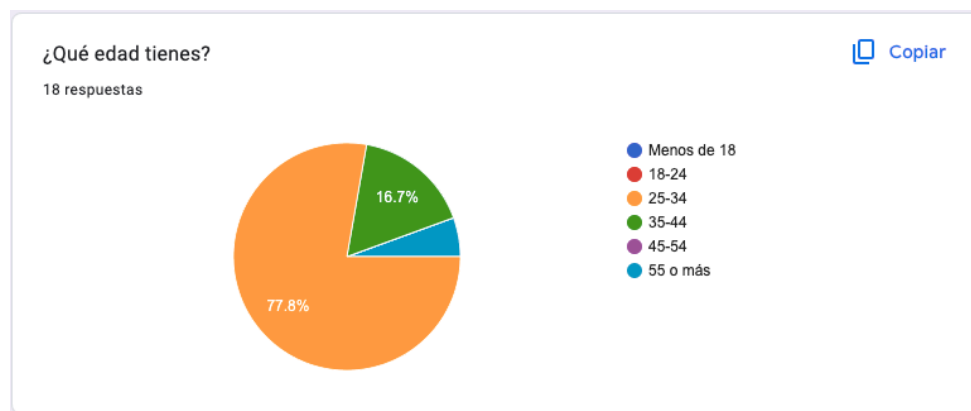


Figura 8.5: Distribución de edades de los participantes.

La distribución de edades de los usuarios que respondieron el formulario se muestra en la figura 8.5. La mayoría de los participantes (77.8%) se encuentra en el rango de 25-34 años, lo que indica que la aplicación fue probada mayoritariamente por adultos jóvenes. Este dato es relevante para comprender el perfil demográfico de los usuarios y adaptar futuras mejoras a este grupo etario.

■ ¿Cuántas mascotas tienes?

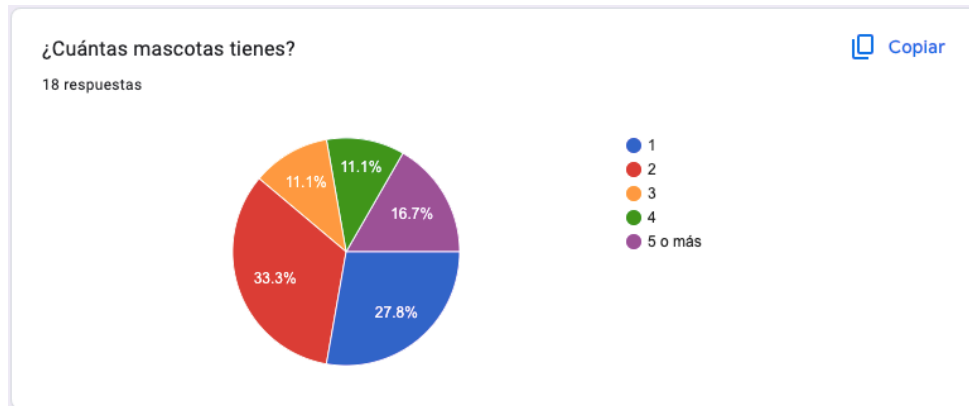


Figura 8.6: Número de mascotas por participante.

La figura 8.6 muestra que el 33.3 % de los usuarios tiene 3 mascotas, seguido por un 27.8 % que tiene 2 mascotas. Este dato es importante porque indica que la mayoría de los usuarios tienen más de una mascota, lo cual podría influir en la percepción de la utilidad de la aplicación para gestionar múltiples registros.

■ **¿Qué tan fácil fue para ti crear una cuenta de usuario?**

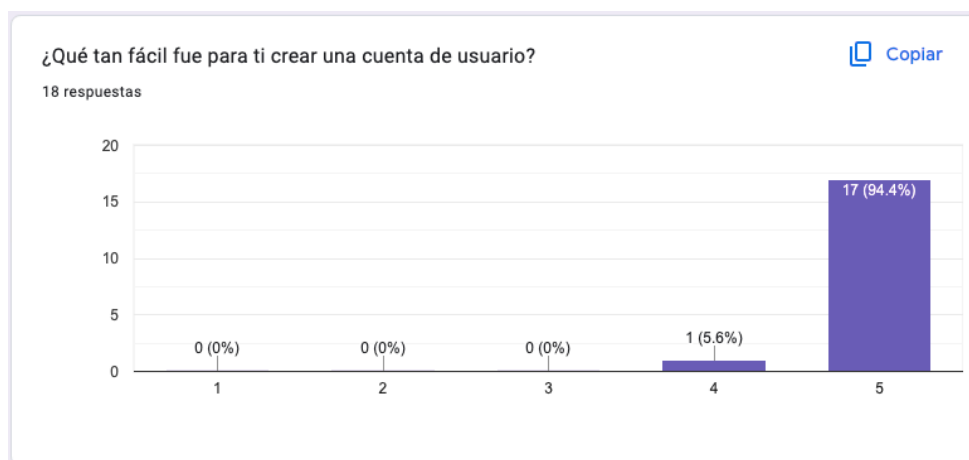


Figura 8.7: Facilidad para crear una cuenta de usuario.

En la figura 8.7, se observa que el 94.4% de los usuarios calificó la creación de una cuenta como muy fácil. Este resultado indica que el proceso de registro es intuitivo y accesible para la mayoría de los usuarios, lo cual es un aspecto positivo para la adopción de la aplicación.

■ **¿Qué tan fácil o intuitivo fue para ti agregar una nueva mascota?**

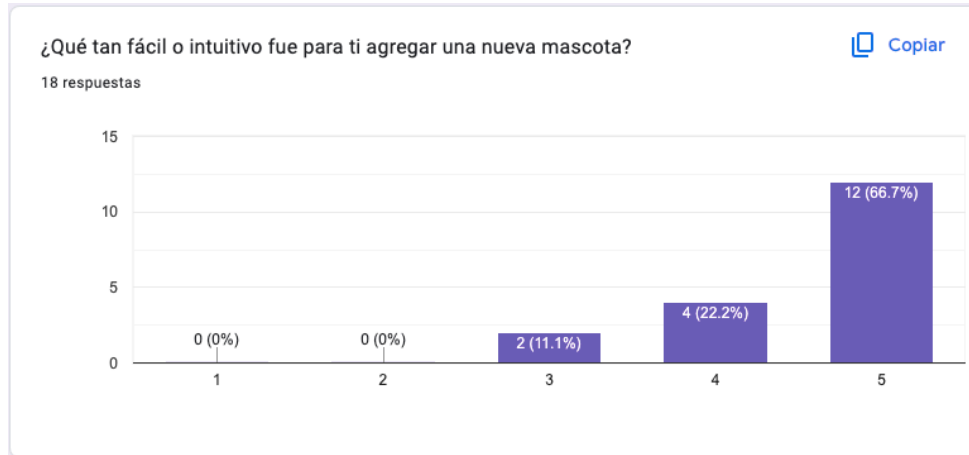


Figura 8.8: Facilidad para agregar una nueva mascota.

La figura 8.8 muestra que un 66.7% de los usuarios encontró muy fácil agregar una nueva mascota, mientras que un 22.2% lo calificó como fácil. Aunque la mayoría de los usuarios tuvo una experiencia positiva, un pequeño porcentaje encontró el proceso menos intuitivo, lo cual podría indicar áreas de mejora en la interfaz de usuario.

■ **¿Qué tan fácil o intuitivo fue para ti agregar una nueva vacuna?**

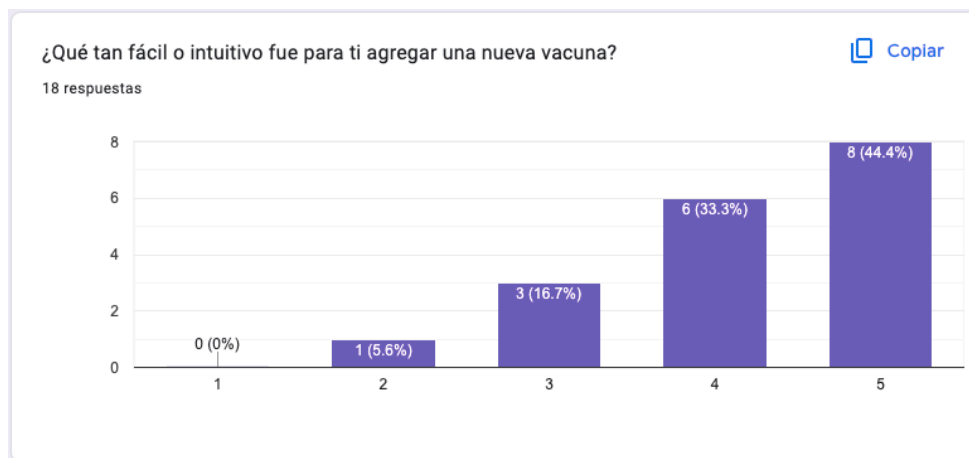


Figura 8.9: Facilidad para agregar una nueva vacuna.

Un 77.7% de los usuarios encontraron fácil o muy fácil agregar una nueva vacuna. Sin embargo, comparado con otras funcionalidades de la aplicación, el gráfico muestra una mayor dispersión en las calificaciones, indicando una percepción de menor facilidad de uso. Se

observan las primeras respuestas bajas (calificación de 2), lo cual sugiere que algunos usuarios encontraron dificultades adicionales al utilizar esta funcionalidad específica.

- **¿Qué tan útil te resultó la información de historial de vacunas de tus mascotas que te proporciona la app?**

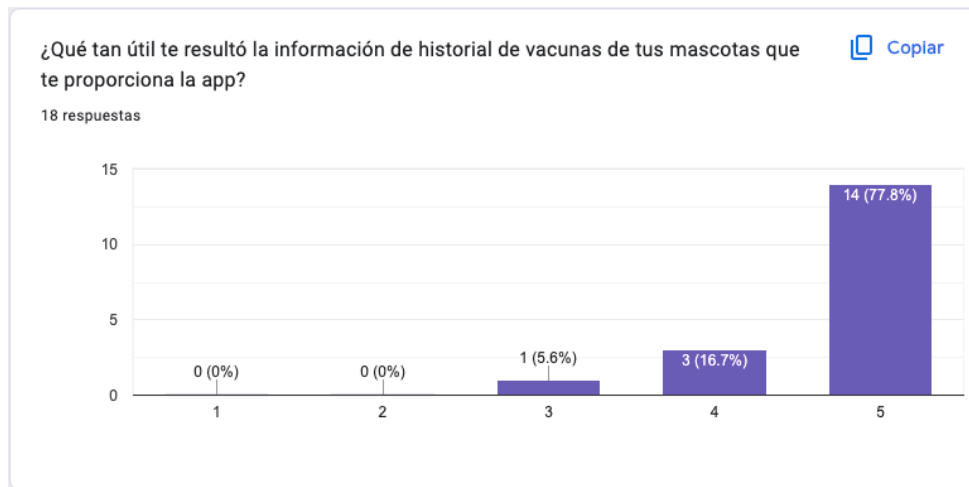


Figura 8.10: Utilidad del historial de vacunas.

La figura 8.10 revela que un 94.5 % de los usuarios encontraron muy útil la información del historial de vacunas proporcionada por la aplicación. Este resultado destaca la relevancia de esta funcionalidad para los usuarios, lo que valida uno de los objetivos principales del proyecto.

- **¿Qué tan útil te resulta la información que la app te muestra de tus mascotas?**

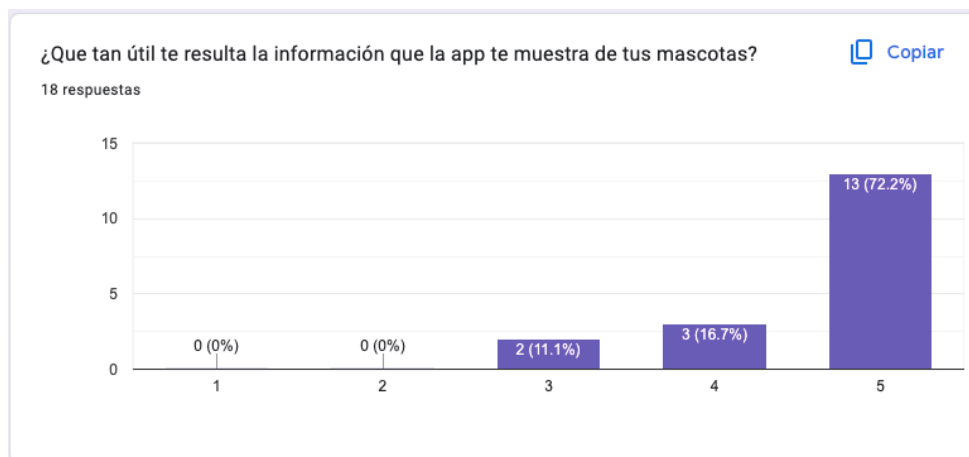


Figura 8.11: Utilidad de la información mostrada sobre las mascotas.

Como se muestra en la figura 8.11, un 72.2% de los usuarios encontraron muy útil la información general de las mascotas que proporciona la aplicación. Este dato confirma la importancia de ofrecer un registro detallado y accesible sobre las mascotas para los usuarios.

- **Si tuviste alguna dificultad al utilizar la aplicación, coméntanos cuál fue** Varios usuarios mencionaron dificultades específicas al utilizar la aplicación. Los problemas más recurrentes incluyeron la falta de una opción para editar la información de las mascotas una vez agregadas, y la necesidad de contar con notificaciones push para recordar las próximas vacunas. Estas sugerencias indican áreas claras de mejora para aumentar la usabilidad y funcionalidad de la aplicación.
- **¿Crees que la aplicación pueda servirte en la gestión del control de vacunas de tus mascotas?**

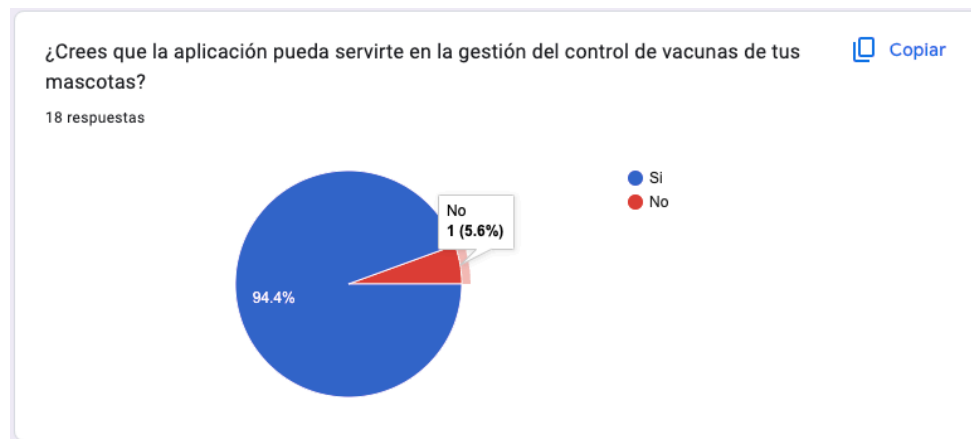


Figura 8.12: Utilidad de la aplicación para la gestión de vacunas.

La figura 8.12 muestra que un 94.4% de los usuarios considera que la aplicación puede servir en la gestión del control de vacunas de sus mascotas. Este alto porcentaje indica que la aplicación cumple con su propósito principal y es vista como una herramienta útil por la mayoría de los usuarios.

- **Si colocaste que No en la pregunta anterior, ¿Qué le falta a la aplicación para que sientas que puede ser de utilidad en el control de vacunas de tus mascotas?** Un usuario

mencionó la necesidad de agregar notificaciones alertar las fechas de próximas vacunas, lo cual puede ser una mejora a considerar en futuras versiones.

- Si colocaste que Sí, ¿Con cuánta frecuencia crees que usarías la aplicación para gestionar las vacunas de tus mascotas?

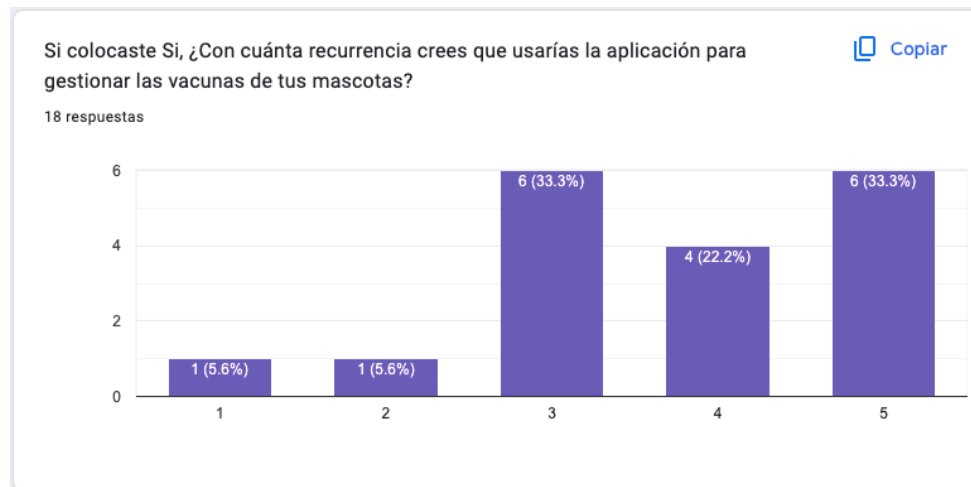


Figura 8.13: Recurrencia de uso de la aplicación para gestionar vacunas.

La mayoría de los usuarios indicaron que utilizarían la aplicación con frecuencia para gestionar las vacunas de sus mascotas.

- ¿Qué tan probable es que sigas usando la aplicación en el futuro?

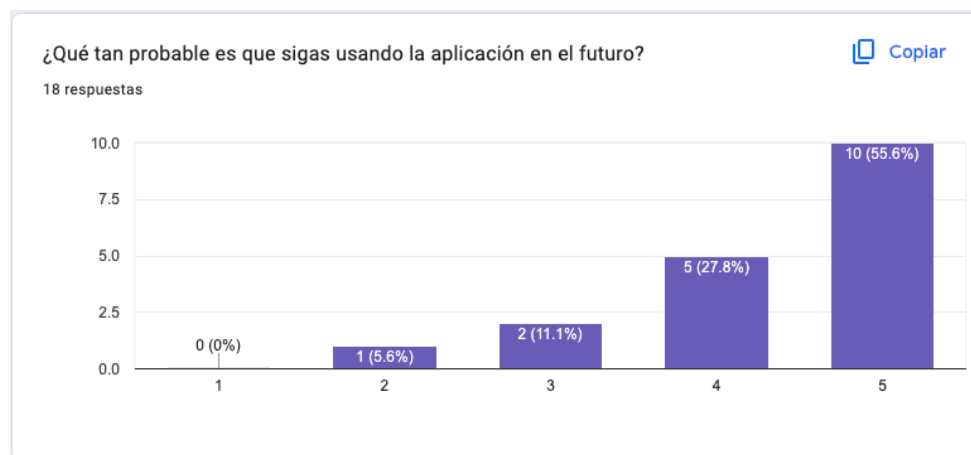


Figura 8.14: Probabilidad de seguir usando la aplicación en el futuro.

La figura 8.14 muestra que un 83.4% de los usuarios indicaron una alta probabilidad de



seguir utilizando la aplicación en el futuro. Este resultado es alentador, ya que sugiere una buena aceptación y satisfacción general con la aplicación.

- **¿Qué sientes que se podría mejorar en la aplicación?** Los usuarios sugieren varias mejoras significativas para la aplicación, destacando la necesidad de poder editar los datos de las mascotas, incluir un sistema de notificaciones para recordatorios de vacunas y desparasitaciones, y agregar más información y funcionalidades relacionadas con la salud de las mascotas, como el historial clínico y la identificación por microchip.
- **¿Qué aspectos de la aplicación te gustaron más?** Los usuarios valoran especialmente la facilidad de uso y manejo de la aplicación, destacando su diseño amigable y visualmente atractivo. Aprecian la practicidad de tener toda la información de sus mascotas centralizada y accesible desde el celular, así como la capacidad de agregar y visualizar datos importantes, como vacunas y etiquetas. También mencionan la fluidez de la aplicación y la posibilidad de compartir el control de una mascota con otros miembros del hogar. El concepto de digitalizar el registro de vacunas es bien recibido.
- **¿Hay alguna función que te gustaría ver añadida en futuras versiones de la aplicación?** Los usuarios sugieren varias funciones para futuras versiones de la aplicación, con un enfoque especial en la incorporación de recordatorios y notificaciones push para desparasitaciones, vacunas y citas veterinarias. También se menciona el interés en añadir funcionalidades para el control de tareas diarias, como la alimentación, y la capacidad de registrar el historial médico y uso de medicamentos. Otros usuarios proponen la integración de alertas y recomendaciones según la raza de la mascota, así como la posibilidad de cargar y compartir fotos divertidas. La personalización del usuario y la disponibilidad de la aplicación en iOS son también puntos interesantes a evaluar. Por último, se sugiere incluir un foro para usuarios y opciones para pedir comida para mascotas a través de la app.
- **¿Nos das un feedback general de tus impresiones con la aplicación?** En general, los usuarios han dado un feedback positivo sobre la aplicación, destacando su utilidad y la facilidad de uso. Muchos valoran la idea innovadora y práctica de tener un control digital



de las vacunas y otros aspectos de la salud de sus mascotas. Aprecian la simplicidad, la interfaz intuitiva y el concepto general. Sin embargo, también mencionan áreas de mejora, como la necesidad de pulir la propuesta gráfica, mejorar la combinación de colores y fuentes de letra, y añadir más funcionalidades, como el historial médico y recordatorios de citas y desparasitaciones. Además, sugieren la inclusión de información adicional en el perfil de la mascota, como alergias, dietas específicas y el peso de la mascota. En resumen, la aplicación tiene mucho potencial y, con algunas actualizaciones, puede ofrecer un servicio aún más completo y útil para los dueños de mascotas.

En el análisis de las respuestas de los usuarios, se identifican varios temas recurrentes que reflejan tanto los aspectos positivos como las áreas de mejora para la aplicación:

- **Facilidad de uso e interfaz intuitiva:** Muchos usuarios destacaron la simplicidad y facilidad de uso de la aplicación, mencionando que la interfaz es amigable y permite una navegación clara y sencilla.
- **Control digital de la salud de las mascotas:** Se aprecia la utilidad de la aplicación para mantener un registro digital de las vacunas, lo que facilita el acceso a la información en cualquier momento y lugar.
- **Necesidad de más funcionalidades:**
 - **Push notifications y recordatorios:** Hay una demanda significativa por notificaciones automáticas para vacunas, desparasitaciones y otras citas médicas.
 - **Gestión de desparasitaciones:** Los usuarios quieren poder registrar desparasitaciones y recibir recordatorios sobre ellas.
 - **Edición de datos:** Existe una necesidad de poder editar la información de las mascotas y las vacunas después de haberlas ingresado.
 - **Historial médico:** Muchos usuarios sugieren la incorporación de un historial médico completo de la mascota, que incluya información sobre alergias, dietas específicas y tratamientos.



- **Mejoras en el diseño gráfico:** Aunque la aplicación es valorada por su simplicidad, algunos usuarios creen que la propuesta gráfica puede ser pulida, sugiriendo mejoras en la combinación de colores y fuentes de letra para hacerla más atractiva.

En resumen, las palabras clave y conceptos más repetidos entre los usuarios son: interfaz intuitiva, push notifications, gestión de desparasitaciones, edición de datos, historial médico y mejoras en el diseño gráfico. Estas áreas representan tanto los puntos fuertes actuales de la aplicación como las oportunidades de mejora y expansión de funcionalidades.



Capítulo 9

Conclusiones

En la presente memoria de titulación se ha desarrollado una aplicación móvil escalable para el registro y seguimiento de la información médica de mascotas. A lo largo del proyecto, se ha abordado el desarrollo y la implementación de diversas funcionalidades, el diseño de una arquitectura escalable, la realización de pruebas exhaustivas y la colaboración en equipo. En base a ello y respecto de los objetivos planteados, se puede establecer que se ha cumplido:

1. Se ha diseñado e implementado una arquitectura escalable que permite el crecimiento futuro del sistema, asegurando su capacidad para manejar un aumento en el número de usuarios y funcionalidades.
2. Se ha desarrollado una aplicación móvil con funcionalidades clave para el registro y seguimiento de la información médica de mascotas, incluyendo autenticación de usuario, registro de mascotas y gestión de vacunas.
3. Se ha utilizado un enfoque colaborativo y de liderazgo, asumiendo roles de responsabilidad y gestión de equipo, lo que ha permitido una organización eficiente y una comunicación efectiva.
4. Se ha implementado un sistema de pruebas exhaustivas, tanto unitarias como de integración y con usuarios finales, para garantizar la calidad y el correcto funcionamiento de la aplicación.

9.1. Trabajo en equipo y liderazgo

Durante el desarrollo del proyecto, se asumió un rol de liderazgo y responsabilidad que no había sido experimentado antes. Se aprendieron valiosas lecciones sobre el manejo del equipo, la



comunicación asertiva y la organización. Esta experiencia permitió comprender mejor la perspectiva de los propios líderes en el ámbito laboral. La comunicación clara y abierta fue esencial para resolver problemas de manera eficiente y asegurar que todos los miembros del equipo estuvieran alineados con los objetivos del proyecto.

Además, la gestión de trade-offs fue un aspecto crucial del liderazgo y la organización del proyecto. No solo se trató de cumplir con las fechas de entrega, sino también de considerar los problemas y limitaciones de los integrantes del equipo. Si alguien se enfermaba o experimentaba burnout, era necesario replantear las metas y decidir qué aspectos del proyecto debían ser priorizados o descartados. Esta flexibilidad y empatía fueron fundamentales para mantener un ambiente de trabajo saludable y productivo.

9.2. Seguridad y vulnerabilidades

Durante el desarrollo del proyecto, se destacó la importancia de la seguridad en el software. Se aprendió sobre las diversas vulnerabilidades a las que puede estar expuesta una aplicación y la necesidad de implementar herramientas y prácticas adecuadas para proteger la información de los usuarios y garantizar la integridad del sistema.

9.3. Escalabilidad y arquitectura

Uno de los aspectos más destacados del desarrollo del proyecto fue la decisión de crear una arquitectura escalable. Este enfoque ofrece grandes beneficios a largo plazo, permitiendo que la aplicación crezca y se adapte a una mayor cantidad de usuarios y funcionalidades. Sin embargo, también presenta desafíos significativos.

Desarrollar una aplicación escalable implica una inversión considerable de tiempo y recursos en la planificación y la implementación de cada componente del sistema. Las ventajas de esta aproximación se cosechan a futuro, ya que una arquitectura bien diseñada puede manejar un au-



mento en la carga de usuarios y funcionalidades sin necesidad de reestructuraciones importantes. Esto es crucial si el proyecto crece y se convierte en una plataforma más amplia y compleja.

Sin embargo, si el proyecto no tiene una visión clara de crecimiento a futuro, esta inversión puede resultar en una pérdida de tiempo y recursos. En proyectos que no escalan, el tiempo y esfuerzo dedicados a diseñar una arquitectura escalable pueden ser vistos como innecesarios, ya que estos recursos podrían haberse utilizado para desarrollar otras funcionalidades o mejorar aspectos específicos de la aplicación en el corto plazo. Por lo tanto, es esencial balancear la visión de futuro con las necesidades actuales del proyecto y considerar cuidadosamente si la escalabilidad es una prioridad realista y necesaria desde el inicio.

9.4. Uso de microservicios

La adopción de una arquitectura de microservicios trajo consigo varias ventajas significativas para el proyecto. Este enfoque permitió separar responsabilidades en servicios individuales, facilitando el desarrollo, despliegue y mantenimiento de cada componente del sistema de manera independiente. Una de las principales bondades de los microservicios es la capacidad de aislar fallos. Si un servicio específico encuentra un error fatal o falla, esta situación no afecta a toda la aplicación, sino únicamente al servicio involucrado, manteniendo el resto de los servicios operativos. Esta característica es crucial para mantener la estabilidad y disponibilidad del sistema en producción, mejorando la resiliencia general del proyecto.

Además, los microservicios facilitan la escalabilidad horizontal. Cada servicio puede ser replicado y escalado de manera independiente según la demanda, optimizando el uso de recursos y permitiendo una respuesta eficiente a las variaciones en la carga de trabajo. Sin embargo, esta arquitectura también implica desafíos adicionales, como la gestión de la comunicación entre servicios, la implementación de mecanismos de seguridad y la orquestación de despliegues.



9.5. Elección de herramientas y lenguajes en proyectos pequeños

En proyectos con equipos reducidos y tiempo limitado, la elección de herramientas y lenguajes de programación es crucial para optimizar el desarrollo y minimizar problemas. La decisión de utilizar el mismo lenguaje, Kotlin, tanto en el backend como en el frontend, resultó ser un gran acierto. Este enfoque permitió una transición más fluida entre diferentes partes del sistema, facilitando el trabajo y mejorando la eficiencia.

La uniformidad en el lenguaje de programación simplificó la comunicación y la colaboración dentro del equipo, ya que todos los miembros podían trabajar en ambos lados del desarrollo sin necesidad de adaptarse a diferentes paradigmas o tecnologías. Esto no solo ahorró tiempo, sino que también redujo la posibilidad de errores y malentendidos.

Otro aspecto importante fue el uso de módulos de Gradle en lugar de dividir la aplicación en múltiples repositorios. Al tener todo el código en un solo proyecto, se evitó la necesidad de levantar y sincronizar diferentes partes de la aplicación por separado. Este enfoque modular en funcionalidades reutilizables, pero monolítico en la ubicación de los módulos, facilitó las pruebas y la integración continua, permitiendo a los desarrolladores realizar cambios rápidamente y verificar su impacto de manera inmediata.

En contraste, optar por una arquitectura distribuida con múltiples repositorios habría requerido una mayor coordinación y tiempo para gestionar dependencias y versiones. Si bien esta opción podría ofrecer beneficios en términos de reutilización de código y separación de responsabilidades, en un proyecto pequeño con recursos limitados, la simplicidad y la eficiencia de una estructura monolítica modularizada demostraron ser más adecuadas.



9.6. Pruebas y calidad del producto

Las pruebas, tanto unitarias como de integración y con usuarios finales, fueron esenciales para garantizar el correcto funcionamiento de la aplicación. Las pruebas unitarias verificaron el comportamiento de componentes individuales, mientras que las pruebas de integración aseguraron la correcta interacción entre los diferentes componentes del sistema. Las pruebas con usuarios finales proporcionaron valioso feedback sobre la usabilidad y funcionalidad del producto.

9.7. Pruebas con usuarios

El análisis de las respuestas del formulario revela que la aplicación ha sido bien recibida por los usuarios, quienes valoran su utilidad y facilidad de uso. Las sugerencias de mejoras proporcionan una guía valiosa para futuras versiones y destacan la importancia de seguir desarrollando características que respondan a las necesidades de los usuarios. En general, estos resultados demuestran que la aplicación tiene un potencial significativo para ayudar en la gestión del control de datos médicos de las mascotas, cumpliendo con los objetivos planteados en este proyecto.

9.8. Comentarios finales

En conclusión, el desarrollo del proyecto fue un viaje de aprendizaje y crecimiento. A pesar de los desafíos, el proyecto demostró la importancia de una buena planificación, colaboración efectiva y uso de tecnologías adecuadas. La decisión de crear una arquitectura escalable asegura que la aplicación esté preparada para el futuro. Este proyecto no solo resultó en una aplicación funcional y útil, sino que también proporcionó valiosas lecciones y experiencias que serán beneficiosas en futuros emprendimientos tecnológicos. Además, documentar todo el proceso de construcción de una aplicación desde cero hasta su despliegue es extremadamente valioso. Esta información puede servir de guía a futuros emprendedores, proporcionando un camino claro y detallado a seguir, facilitando la entrada a nuevas ideas e innovaciones tecnológicas.



Referencias Bibilográficas

- [1] MongoDB. (2021). *Relational vs Non-Relational Databases* Recuperado el 25 de junio, 2024, de <https://www.mongodb.com/compare/relational-vs-non-relational-databases>
- [2] Amazon Web Services. (s.f.). *The difference between web apps, native apps, and hybrid apps*. Recuperado el 25 de junio, 2024, de <https://aws.amazon.com/es/compare/the-difference-between-web-apps-native-apps-and-hybrid-apps/>
- [3] John Smith. (2019). *Backend for Frontend Pattern: Why You Need to Know It.*, Recuperado el 25 de junio, 2024 de <https://medium.com/mobilepeople/backend-for-frontend-pattern-why-you-need-to-know-it-46f94ce420b0>
- [4] Airbnb Engineering. (2018). *A Deep Dive Into Airbnb's Server-Driven UI System.*, Recuperado el 25 de junio, 2024 de <https://medium.com/airbnb-engineering/a-deep-dive-into-airbnbs-server-driven-ui-system-842244c5f5>
- [5] Android Developers. (s.f.). *Architecture Documentation*. Google, Recuperado el 25 de junio, 2024 de <https://developer.android.com/topic/architecture?hl=es-419>
- [6] C. Martin, Robert. (2008) *Clean Code: A Handbook of Agile Software Craftsmanship*. Prentice Hall.
- [7] C. Martin, Robert. (2017) *Clean Architecture. A Craftsman's Guide to Software Structure and Design*. Prentice Hall.
- [8] Sten Pittet, Atlassian Collaborator. (2021). *Tipos de pruebas de softwares.*. Atlassian, Recuperado 25 de junio, 2024, de <https://www.atlassian.com/es/continuous-delivery/software-testing/types-of-software-testing>



- [9] Mindorks. (2019). *The Evolution of Android Architectures: MVP, MVVM, MVI*. Medium, Recuperado el 25 de junio, 2024 de <https://medium.com/mindorks/the-evolution-of-android-architectures-mvp-mvvm-mvi-f72f67093b81>
- [10] Airbnb Engineering & Data Science. (2018, 28 de marzo). *Sunsetting React Native*. Medium. Recuperado el 25 de junio, 2024 de <https://medium.com/airbnb-engineering/sunsetting-react-native-1868ba28e30a>
- [11] Facebook Engineering. (2016, 13 de abril). *React Native - A Year in Review*. Recuperado el 25 de junio, 2024 de <https://engineering.fb.com/2016/04/13/android/react-native-a-year-in-review/>
- [12] GfK. (2022). *Estudio sobre tenencia de mascotas en Chile*. Recuperado el 25 de junio, 2024 de <https://www.gfk.com>
- [13] Grand View Research. (2021). *Pet Care Market Size, Share & Trends Analysis Report*. Recuperado el 25 de junio, 2024 de <https://www.grandviewresearch.com/industry-analysis/pet-care-market>
- [14] Spring Team. (s.f.). *Spring Boot Official Documentation*. Recuperado el 25 de junio, 2024 de <https://docs.spring.io/spring-boot/documentation.html>
- [15] MongoDB, Inc. (s.f.). *MongoDB Official Documentation*. Recuperado el 25 de junio, 2024 de <https://www.mongodb.com/docs/>
- [16] Auth0 by OktaJ. (s.f.). *WT Introduction*. Recuperado el 25 de junio, 2024 de <https://jwt.io/introduction>
- [17] F5 NGINX. (s.f.). *NGINX Documentation*. Recuperado el 25 de junio, 2024 de <https://docs.nginx.com/nginx/admin-guide/web-server/>
- [18] Square, Inc. (s.f.). *Retrofit Page*. Recuperado el 25 de junio, 2024 de <https://square.github.io/retrofit/>
- [19] Android Developers. (s.f.). *Navigation Documentation*. Google, Recuperado el 25 de junio, 2024 de <https://developer.android.com/guide/navigation?hl=es-419>



- [20] Atlassian Company. (s.f.). *Confluence Official Site*. Atlassian, Recuperado el 22 de Agosto, 2024 de <https://www.atlassian.com/software/confluence>
- [21] Kong Inc. (s.f.). *Insomnia Documentation*. Recuperado el 22 de Agosto, 2024 de <https://docs.insomnia.rest/insomnia/get-started>
- [22] Meta Platforms, Inc. (s.f.). *Flipper Official Site*. Recuperado el 22 de Agosto, 2024 de <https://fbflipper.com/>
- [23] Atlassian Company. (s.f.). *Jira Official Site*. Atlassian, Recuperado el 22 de Agosto, 2024 de <https://www.atlassian.com/software/jira>
- [24] Slack Technologies. (s.f.). *Slack Official Site*. Salesforce, Recuperado el 22 de Agosto, 2024 de <https://slack.com/intl/es-es>



Anexos



.1. Excel con las respuestas de pruebas de usuarios.

Se presenta el excel que contiene todas las preguntas y respuestas de los usuarios que utilizaron la aplicación.

<https://docs.google.com/spreadsheets/d/14NH16bf-D-2JPVt0643MPQx0ppufT-7tuJNJ4pS4tIk/>

.2. Página de Figma con MockUps de ejemplo.

Se presenta el ejemplo de una página de figma.

<https://www.figma.com/design/f2KkdaT2YQJgURDtPyu0BP/Untitled?node-id=0-1&t=wKHGWaE23EKecFTc-0>

.3. Repositorio de aplicación Android de ejemplo.

Se presenta una aplicación de ejemplo, para utilizarse como referencia.

<https://github.com/Piliwiwi/pet-app-example>

.4. Repositorio de servicio de Autenticación de ejemplo.

Se presenta un servicio de autenticación de ejemplo, para utilizarse como referencia.

<https://github.com/Piliwiwi/pet-svc-auth-example>

.5. Repositorio de servicio de Mascotas de ejemplo.

Se presenta un servicio de mascotas de ejemplo, para utilizarse como referencia.

<https://github.com/Piliwiwi/pet-svc-pets-example>



.6. Repositorio de servicio de Middleend de ejemplo.

Se presenta un servicio de middleend de ejemplo, para utilizarse como referencia.

<https://github.com/Piliwiwi/pet-svc-middleend-example>