UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
GRADUADO EN INGENIERÍA DEL SOFTWARE

# Sistema Integral de Registro y Seguimiento para la Gestión del Cuidado y Bienestar de Mascotas

# Comprehensive Registration and Tracking System for Pet Care and Wellbeing Management


Realizado por
**Juan Francisco Sánchez García**

Tutorizado por
**Luis Manuel Llopis Torres**

Departamento
**Lenguajes y Ciencias de la Computación**

UNIVERSIDAD DE MÁLAGA


MÁLAGA, SEPTIEMBRE DE 2025

# Abstract

Currently, pet care and welfare have become very important in society, motivating owners to seek more efficient ways to manage their animals' information and health. However, many people still use traditional methods such as physical records or isolated digital solutions, which makes proper monitoring difficult. This Final Degree Project proposes the development of a comprehensive software system, consisting of a web application and a mobile application, which centralises the management of all aspects related to pet care.

The system is designed for both pet owners and veterinary professionals. The mobile version allows users to manage animal information, such as vaccinations, veterinary appointments, feeding, walks or notes, from anywhere. The web version is specifically designed for professional use in veterinary clinics, facilitating customer management, appointment scheduling and the viewing of statistics and medical reports.

The proposal stands out for offering a unified, accessible and comprehensive solution that addresses both the specific needs of pet owners and the operational requirements of professionals in the veterinary sector.

**Keywords: Pets, Android, Web application, Veterinary management**

# Resumen

En la actualidad, el cuidado y bienestar de las mascotas ha cobrado una gran relevancia en la sociedad, motivando a los propietarios a buscar formas más eficientes de gestionar la información y salud de sus animales. No obstante, muchas personas siguen utilizando métodos tradicionales como registros físicos o soluciones digitales aisladas, lo que dificulta un seguimiento adecuado. Este Trabajo de Fin de Grado propone el desarrollo de un sistema software integral, compuesto por una aplicación web y una aplicación móvil, que centraliza la gestión de todos los aspectos relacionados con el cuidado de las mascotas.

El sistema está diseñado tanto para propietarios de mascotas como para profesionales veterinarios. La versión móvil permite a los usuarios gestionar desde cualquier lugar la información de los animales, como vacunas, citas veterinarias, alimentación, paseos o notas. Por su parte, la versión web está especialmente orientada al uso profesional en clínicas veterinarias, facilitando la gestión de clientes, programación de citas y visualización de estadísticas e informes médicos.

La propuesta se distingue por ofrecer una solución unificada, accesible y completa, que aborda tanto las necesidades particulares de los dueños de mascotas como los requisitos operativos de los profesionales del sector veterinario.

**Palabras clave: Mascotas, Android, Aplicación web, Gestión veterinaria**

# Índice

# 1 Introducción

## 1.1. Motivación

En la actualidad, el cuidado de las mascotas ha cobrado una gran importancia en la sociedad, generando una mayor conciencia entre los propietarios sobre el bienestar de sus animales. No obstante, muchas de estas personas no gestionan de manera eficiente la salud ni la información relacionada con sus mascotas, ya sea porque la conservan en formato físico o porque carecen de una solución tecnológica que centralice todos estos datos.

Esta situación pone de manifiesto la necesidad de desarrollar una aplicación integral que permita a los dueños organizar y gestionar, de forma centralizada, toda la información relevante relacionada con el cuidado de sus animales. Al mismo tiempo, el sistema debe ofrecer funcionalidades que también resulten útiles para los profesionales veterinarios, facilitando así la interacción entre ambas partes.

## 1.2. Objetivos

El objetivo principal de este TFG es el desarrollo de un sistema integral formado por dos aplicaciones: una **aplicación móvil** y una **aplicación web**. Este sistema estará orientado a facilitar la gestión del cuidado y bienestar de las mascotas, permitiendo tanto a los propietarios como a los profesionales veterinarios centralizar, consultar y gestionar toda la información relevante de forma eficiente y accesible.

La aplicación móvil está diseñada con un enfoque en la facilidad de uso, la portabilidad y el acceso inmediato a los datos, lo que la hace especialmente útil para los dueños de mascotas en su día a día. Por otro lado, la versión web, orientada al entorno clínico, proporciona a los veterinarios herramientas avanzadas para gestionar el seguimiento de sus pacientes, así como acceder a información profesional relevante como estadísticas sobre ingresos, citas atendidas,

vacunas administradas y otros indicadores clave relacionados con su actividad profesional.

Entre los objetivos específicos del sistema se encuentran:

- Facilitar el registro y la consulta de citas veterinarias, vacunas, tratamientos y hábitos de las mascotas.
- Permitir el seguimiento del estado de salud y la evolución de cada animal mediante historiales clínicos accesibles.
- Mejorar la comunicación entre propietarios y veterinarios a través de funcionalidades compartidas y sincronizadas.
- Proporcionar estadísticas e información visual para ayudar en la toma de decisiones y en la organización de las tareas veterinarias.
- Integrar herramientas como mapas, almacenamiento multimedia o gestión financiera, ampliando así la utilidad de la plataforma.
- Permitir el registro e inicio de sesión de usuarios mediante correo electrónico y contraseña o autenticación con Google.
- Facilitar la gestión de perfiles de mascotas, incluyendo su creación, edición, eliminación y visualización.
- Registrar y consultar el historial de visitas veterinarias y vacunas.
- Ofrecer rastreo en tiempo real de paseos mediante integración con Google Maps, incluyendo localización, recorrido, duración, distancia y toma de fotos durante el paseo.
- Proporcionar un sistema de notas clasificadas por categorías sobre cuidados y bienestar animal.
- Registrar los gastos asociados al cuidado de las mascotas y facilitar su planificación y gestión financiera, mediante el uso de estadísticas visuales y filtros por categoría y monto.
- Centralizar la información de los usuarios, sus mascotas y sus actividades en una base de datos unificada en Firestore, garantizando coherencia, disponibilidad y sincronización en tiempo real entre ambas plataformas.

## 1.3. Estructura de la memoria

Este documento se divide en un primer capítulo de **introducción**, en el cual se presenta el contexto del proyecto y su motivación, se mencionan los objetivos generales y específicos, además de una pequeña explicación de la estructura de esta memoria y un apartado dedicado a describir la metodología seguida para la realización de este trabajo.

El segundo capítulo, denominado **"Tecnologías Utilizadas"**, incluye las tecnologías y herramientas utilizadas para el desarrollo del sistema, separando las relacionadas con la aplicación móvil por una parte y con la aplicación web por otra.

La parte central está compuesta por los capítulos 3, 4, 5 y 6. El capítulo 3: **Análisis**, incluye el análisis de requisitos funcionales y no funcionales de las aplicaciones, así como sus casos de uso. El cuarto capítulo describe la **fase de diseño**, en la que se describe con detalle el diseño arquitectónico, la base de datos escogida y su estructura y el diseño particular de cada aplicación del sistema. El capítulo 5 recoge aspectos de la **implementación** del sistema. El último capítulo de esta parte principal está dedicado a describir las distintas pruebas realizadas para verificar y validar el sistema.

Finalmente, en el capítulo 7: **Conclusiones y Trabajos Futuros**, se destacan los resultados obtenidos, la evaluación del proyecto y las posibles líneas de mejora o ampliación futuras en el sistema.

Al final de este documento, además del capítulo correspondiente a las referencias, se incluyen como anexos un manual de instalación y un manual de usuario.

## 1.4. Metodología

Este Trabajo Fin de Grado se enmarca dentro de un enfoque eminentemente práctico, orientado al desarrollo de una solución funcional y aplicable a un contexto real. Se ha diseñado y construido un sistema compuesto por dos aplicaciones conectadas a una misma base de datos en la nube, con el objetivo de centralizar la gestión del cuidado y bienestar de las mascotas. Este enfoque permite no solo poner en práctica conocimientos adquiridos a lo largo del grado, sino también generar una herramienta con potencial de uso real tanto para propietarios como para veterinarios.

La metodología de desarrollo de software seguida para ello ha sido **Scrum**, aunque adapta-

da al contexto y condiciones de este proyecto. Las razones que han motivado esta decisión son las ventajas que ofrecen las metodologías ágiles, en concreto Scrum, como son la flexibilidad a los cambios en los requisitos y la entrega incremental de software funcional, que permite obtener una retroalimentación temprana y frecuente por parte del tutor (en este caso), tener un mayor control en el progreso del proyecto o adaptarse a los cambios en los requisitos, es decir, poder modificar ciertas partes del sistema sin tener que rehacer todo de nuevo.

El proceso se ha llevado a cabo de forma iterativa e incremental, como es común en este tipo de metodologías. Cada iteración se denomina *sprint*, y ha tenido una duración de entre 2 y 3 semanas. En cada *sprint* se implementan nuevas funcionalidades y se van ajustando las anteriores. Durante el desarrollo, se tienen en cuenta elementos como el *product backlog*, que son el conjunto de requisitos llamados historias, y el *sprint backlog*, que es el conjunto de tareas necesarias para llevar a cabo las historias de ese *sprint*. Además, se han mantenido reuniones periódicas con el tutor al final de cada iteración y al inicio de la siguiente, con el objetivo de presentar las funcionalidades implementadas (*retrospective*) y determinar las nuevas prioritarias de cara al siguiente (*sprint planning*) Figura 1.

Por otro lado, cabe señalar que en este proyecto individual, el propio desarrollador ha sido el que ha desempeñado el rol de *scrum master*, trabajando de manera autosuficiente y auto-organizada, como indica el principio 11 del Manifiesto Ágil: ”Las mejores arquitecturas, requisitos y diseños emergen de equipos auto-organizados.”(Alliance, 2025). Asimismo, el tutor ha actuado como *product owner*, aportando sugerencias respecto a los requisitos y funcionalidades del proyecto, además de haber ido revisando y validando las entregas parciales.

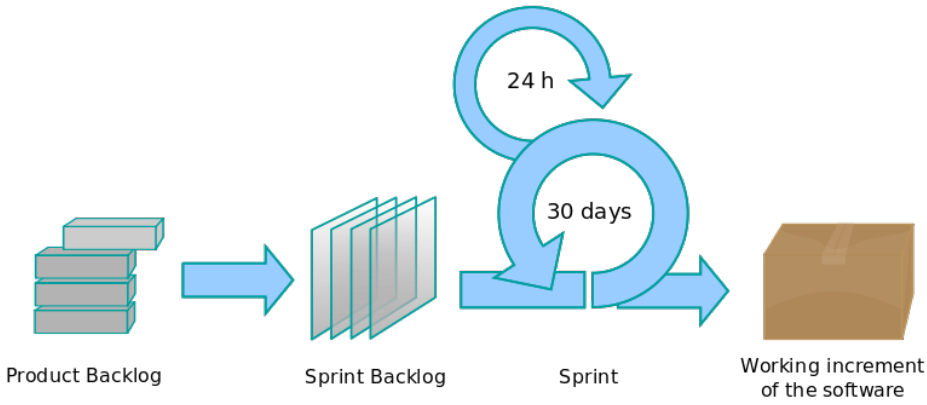



Figura 1: Flujo de trabajo en la metodología Scrum.

Fuente: https://www.troopsf.com/scrum

El desarrollo del proyecto se ha estructurado en varias fases (Figura 2), siguiendo una secuencia lógica adaptada a la metodología Scrum y a las necesidades propias de este TFG. Las principales fases han sido las siguientes:

**1. Análisis de requisitos.**

En primer lugar, se elaboró un análisis detallado de los requisitos funcionales y no funcionales del sistema, tomando como base los objetivos generales definidos al inicio del proyecto y aplicando criterios de utilidad práctica. Aunque algunos estuvieron claros desde un principio, a lo largo del proceso se han ido añadiendo unos y modificando otros de cara a obtener un producto final de mayor calidad.

Los requisitos se agruparon por áreas funcionales (gestión de usuarios, mascotas, citas, gastos, paseos, etc.), permitiendo tener una visión modular del sistema desde etapas tempranas.

**2. Diseño del sistema.**

Posteriormente, se llevó a cabo el diseño de la arquitectura general del sistema, estructurada en dos aplicaciones conectadas a una única base de datos en la nube. Se definieron los modelos de datos, la organización del servidor (*backend*) y del cliente (*frontend*), así como la lógica de comunicación entre las distintas capas. También se diseñaron las interfaces de usuario, buscando un enfoque limpio, moderno y adaptado a la temática del proyecto, priorizando

la usabilidad.

**3. Implementación.**

La implementación del sistema se realizó de forma incremental, alternando entre las dos aplicaciones según las prioridades funcionales definidas en el *backlog*. La aplicación móvil se desarrolló integrando distintas funcionalidades accesibles para cualquier usuario que se registre con éxito en la aplicación.

En la parte de la aplicación web se emplearon buenas prácticas de programación (uso de controladores, servicios, modularidad, validaciones, componentes reutilizables) y herramientas como Firebase Authentication o Firebase Storage.

**4. Pruebas y validación.**

Durante el desarrollo se realizaron pruebas manuales tanto en la aplicación móvil como en la plataforma web, evaluando la correcta funcionalidad de cada módulo tras su implementación. Además, se utilizaron herramientas como Postman para verificar el correcto funcionamiento de la API REST, asegurando que las operaciones respondieran correctamente en distintos escenarios.

Se llevaron a cabo pruebas de validación general sobre distintas partes de la aplicación, como el registro de mascotas o solicitud y confirmación de citas, detectando y corrigiendo posibles errores o comportamientos no deseados antes de avanzar a nuevas tareas.

**5. Documentación.**

Una vez completado el desarrollo, se elaboró la documentación correspondiente al proyecto, centrándose en la redacción de esta memoria como elemento principal del TFG. La documentación recoge tanto los aspectos técnicos como las decisiones de diseño, la estructura del sistema, capturas ilustrativas y consideraciones finales. Todo ello se realizó una vez finalizado el proceso de desarrollo, recogiendo de forma estructurada los resultados obtenidos.

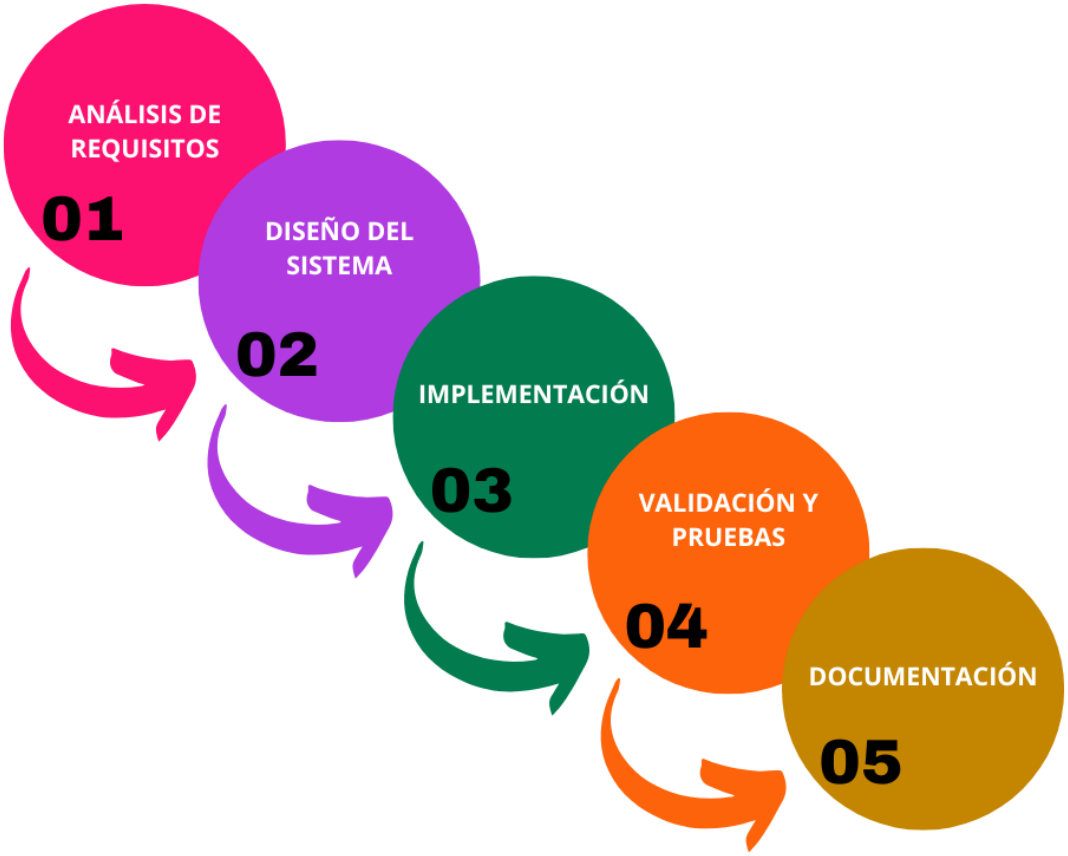



Figura 2: Fases principales del desarrollo del sistema.

Fuente: Elaboración propia.

# 2

# Tecnologías Utilizadas

Para el desarrollo de este TFG ha sido necesario seleccionar y aplicar un conjunto de tecnologías que han permitido materializar el sistema. La elección de cada una de ellas no ha sido arbitraria, sino que responde a criterios de adecuación a los requisitos funcionales y no funcionales del proyecto, así como a la necesidad de garantizar un equilibrio entre robustez, escalabilidad, mantenibilidad y facilidad de uso.

Se comienza presentando el ecosistema de Firebase como solución de servicios en la nube, junto con las herramientas empleadas para el control de versiones a lo largo de todo el proceso de implementación, dado que estas tecnologías son comunes al desarrollo de ambos tipos de aplicaciones.

A continuación, el capítulo se organiza en dos bloques principales. Primero, se describen las tecnologías utilizadas en el desarrollo de la aplicación móvil, es decir, las herramientas y lenguajes que han hecho posible la construcción de un entorno nativo para Android. Posteriormente, se detallan las tecnologías aplicadas en la aplicación web, tanto en el frontend como en el backend.

- **Firebase**

  Una pieza fundamental para el funcionamiento del sistema ha sido Firebase. Se trata de una plataforma de desarrollo de aplicaciones respaldada por Google que proporciona una amplia gama de servicios como base de datos, autenticación, almacenamiento, mensajería y funciones en la nube, entre otros. Estos servicios permiten acelerar el desarrollo de aplicaciones al evitar la necesidad de construir infraestructura desde cero, ofreciendo soluciones escalables y en tiempo real.

En este proyecto se ha hecho uso de varios servicios de Firebase para cubrir distintos aspectos funcionales tanto en la aplicación Android como en la aplicación web, permitiendo una integración sencilla y eficiente entre el frontend y el backend, así como una sincronización inmediata de los datos.

Los servicios de Firebase utilizados en este proyecto se explicarán con más detalle en el capítulo 4.


Figura 3: Logo de Firebase.

- **Control de versiones**

  Para el control de versiones del proyecto se ha utilizado **Git** junto con la plataforma **GitHub**, que ha permitido gestionar de forma ordenada el desarrollo tanto de ambas aplicaciones. Se ha mantenido un repositorio independiente para la aplicación móvil y otro único que agrupa el frontend y backend de la aplicación web.

  La gestión de versiones se ha realizado principalmente a través de las integraciones nativas de Git disponibles en los entornos de desarrollo utilizados (Android Studio y Visual Studio Code). Esto facilitó tareas básicas como la creación de commits, el envío de cambios al repositorio remoto y la sincronización de código.

  Al tratarse de un proyecto desarrollado de forma individual, la organización del control de versiones se centró en mantener un historial claro y estructurado de los cambios, permitiendo así realizar un seguimiento efectivo de la evolución del código, facilitar la identificación y corrección de errores, y asegurar la integridad del proyecto durante las diferentes fases de desarrollo.


Figura 4: Logo de Git y GitHub.

## 2.1. Aplicación Móvil

- **Android**

  Android es un sistema operativo móvil diseñado para dispositivos móviles con pantalla táctil, como teléfonos inteligentes, tabletas o relojes inteligentes, entre otros. Fue inicialmente desarrollado por Android Inc. y posteriormente adquirido por Google LLC en 2005. Android es el sistema operativo móvil más utilizado del mundo. En 2024, Android continúa liderando el mercado global de sistemas operativos móviles con una cuota del 70.1 %, mientras que iOS se encuentra en un 29.2 % (Wikipedia contributors, 2024).

  Esta ha sido una de las principales razones por las que se optó por desarrollar una aplicación móvil nativa para Android, aprovechando las capacidades y herramientas específicas de dicho sistema operativo. Otro factor determinante ha sido su facilidad de uso, pues permite crear aplicaciones utilizando interfaces visuales, plantillas y componentes prediseñados. Además, tiene un tiempo de desarrollo muy rápido gracias a sus procesos optimizados. Cabe destacar también que ofrece una gran escalabilidad e integración con sistemas existentes como bases de datos, herramientas, APIs, etc.


Figura 5: Logo de Android.

- **Kotlin**

  Para el desarrollo de la aplicación móvil compatible con el sistema operativo Android se ha empleado Kotlin como lenguaje de programación. Es un lenguaje moderno de tipado estático, orientado a objetos y funcional. Algunas de sus principales ventajas son:

  - Es simple, expresivo y conciso.
  - Permite tener un código más seguro al contar con muchas funciones de lenguaje que ayudan a evitar errores de programación comunes, como excepciones de puntero nulo.
  - Es completamente interoperable con Java, lo que permite llamar a código Java desde Kotlin y viceversa.
  - Se integra de manera sencilla en aplicaciones existentes.
  - Manejo eficiente de la concurrencia. Las corrutinas de Kotlin facilitan el trabajo tanto con operaciones asíncronas como con aquellas que bloquean el flujo de ejecución.

  Grandes empresas como McDonald's, Amazon Web Services (AWS), Philips, Adobe Experience Platform, Forbes y Atlassian utilizan Kotlin para el desarrollo de sus aplicaciones (JetBrains, 2025).


Figura 6: Logo de Kotlin.

- **Android Studio**

  Android Studio es el entorno de desarrollo integrado (IDE, por sus siglas en inglés) utilizado para la codificación de la aplicación móvil para Android. Es el IDE oficial para el desarrollo de este tipo de aplicaciones y proporciona una gran comodidad al programador debido a las funciones que ofrece, como un sistema de compilación flexible basado en Gradle, un emulador rápido, ediciones en vivo en dispositivos físicos a tiempo real, integración con GitHub y plantillas de código para ayudar a compilar funciones de apps

comunes o anotaciones visuales como herramientas de Lint para identificar problemas de rendimiento, usabilidad y compatibilidad de versiones (Google, 2025b).

Figura 7: Logo de Android Studio.

## 2.2. Aplicación Web

- **Node.js**

  Node.js es un entorno de ejecución para el lenguaje de programación JavaScript, gratuito, de código abierto y multiplataforma, que permite desarrollar aplicaciones del lado del servidor (backend), así como herramientas de línea de comandos y scripts. Su versatilidad y elevado rendimiento son factores clave que explican su amplia adopción (Node.js, 2025).

  Este entorno está basado en el motor V8 de Google Chrome, lo que garantiza una ejecución rápida y eficiente del código JavaScript fuera del navegador. Node.js utiliza un modelo de operación asíncrono y orientado a eventos, que permite gestionar múltiples conexiones concurrentes sin bloquear el hilo principal, facilitando así la escalabilidad y un alto rendimiento en aplicaciones web.

  En este proyecto, Node.js ha permitido implementar la lógica del servidor mediante una API REST, facilitando la comunicación entre la la aplicación web y la base de datos. Además, su ecosistema, gestionado a través del gestor de paquetes npm, proporciona una gran variedad de módulos reutilizables que aceleran el desarrollo y aportan funcionalidades adicionales.

  Finalmente, la comunidad activa y el soporte continuo hacen de Node.js una tecnología robusta y actualizada, ideal para proyectos modernos de desarrollo web.

Figura 8: Logo de Node.js.

- **Express**

  Express es un framework minimalista y flexible para el desarrollo del lado servidor con Node.js, orientado principalmente a la construcción de APIs RESTful (Express, 2025). En este proyecto, Express ha sido la base para implementar toda la lógica del servidor de la aplicación web. Gracias a su sistema de enrutamiento y soporte nativo para middleware, se ha estructurado una API REST que facilita la comunicación entre el cliente (aplicación Angular) y la base de datos. Su sintaxis simple y su integración con librerías de terceros han agilizado notablemente el proceso de desarrollo y mantenimiento del backend.

  Este popular framework proporciona una interfaz sencilla para manejar peticiones HTTP (GET, POST, PUT, DELETE, etc.), definir rutas y aplicar middleware personalizado, lo que permite construir una API robusta de manera ágil. Su diseño modular y su integración nativa con Node.js lo convierten en una herramienta eficaz para estructurar servidores escalables y mantenibles.

  Además, la amplia comunidad que lo respalda y la gran cantidad de paquetes disponibles a través del ecosistema de npm permiten extender sus funcionalidades según las necesidades del proyecto.

Figura 9: Logo de Express.

- **TypeScript**

  TypeScript es un lenguaje de programación de código abierto desarrollado por Microsoft que extiende las capacidades de JavaScript mediante la incorporación de tipado estático, clases, interfaces y otras características propias de los lenguajes orientados a objetos. Este lenguaje ha sido diseñado para facilitar el desarrollo de aplicaciones de gran escala, proporcionando mayor robustez, mantenibilidad y detección temprana de errores durante el desarrollo (Microsoft, 2025).

  En este proyecto, TypeScript ha sido utilizado como lenguaje principal para el desarrollo del frontend web con Angular, ya que dicho framework está construido de forma nativa sobre TypeScript. Su sistema de tipos permite definir con claridad las estructuras de datos utilizadas, reduciendo la probabilidad de errores en tiempo de ejecución y facilitando la comprensión del código por parte de otros desarrolladores.

  Entre sus principales ventajas destacan:

  - Mejora la productividad y calidad del código mediante la autocompletación, inferencia de tipos y validación estática en tiempo de compilación.
  - Facilita la detección temprana de errores y posibles incoherencias en el código, lo que contribuye a reducir costes de mantenimiento y depuración.
  - Permite una programación más estructurada y escalable, gracias a conceptos como clases, interfaces, enumeraciones y módulos.
  - Se compila a JavaScript estándar, lo que garantiza su compatibilidad con todos los navegadores modernos.

  Además, su integración directa con el entorno de desarrollo Visual Studio Code y su amplia adopción en el ámbito del desarrollo web lo convierten en una elección idónea para proyectos de este tipo que requieren escalabilidad, claridad y calidad de código.


Figura 10: Logo de TypeScript.

- **HTML5 / CSS**

  HTML5 es la quinta versión del lenguaje de marcado estándar utilizado para estructurar y presentar contenido en la web. Este estándar permite definir de forma clara la jerarquía del contenido, incorporar elementos multimedia sin necesidad de plugins externos, y garantizar una experiencia más coherente entre diferentes dispositivos y navegadores. Junto a él, se ha empleado CSS para aplicar estilos visuales, definir diseños responsivos y adaptar la interfaz de usuario a distintas resoluciones de pantalla.

  En este proyecto, se ha optado por utilizar SCSS (Sassy CSS), una variante de CSS compatible con Angular que añade funcionalidades propias de lenguajes de programación, como variables, funciones, herencia y anidamiento de reglas. Esto ha permitido una mayor modularidad, reutilización de estilos y mantenimiento más sencillo del código a medida que la aplicación crecía en complejidad.

  El uso conjunto de HTML5 y SCSS ha contribuido a crear una interfaz web estructurada y visualmente coherente, facilitando tanto el desarrollo como la experiencia del usuario final (WHATWG, 2025).


Figura 11: Logo de HTML y CSS.

- **Angular**

  Angular es un framework de desarrollo web de código abierto mantenido principalmente por Google, diseñado para la creación de aplicaciones web de una sola página (SPA, por sus siglas en inglés). Está escrito en TypeScript y proporciona una arquitectura robusta basada en componentes, que facilita la modularización, reutilización y mantenimiento del código.

  Este framework ha sido fundamental en la construcción de la aplicación en su versión web, ya que ofrece un ecosistema completo que permite desarrollar interfaces actuales, reactivas y con una excelente experiencia de usuario. Entre sus principales ventajas destaca el sistema de data binding bidireccional, que mantiene sincronizados de forma automática los datos entre el modelo y la vista, así como su sistema de inyección de dependencias, su potente CLI (interfaz de línea de comandos) para generar componentes y servicios y su integración nativa con herramientas como RxJS para la gestión de datos reactivos.

  Además, Angular incorpora funcionalidades como enrutamiento, internacionalización y lazy loading, que han sido especialmente útiles para estructurar una aplicación escalable y con tiempos de carga optimizados. La elección de Angular también ha permitido mantener una estructura clara entre los distintos módulos de la aplicación y facilitar la separación de responsabilidades entre vistas, lógica y datos.

  (Google, 2025a)

Figura 12: Logo de Angular.

- **Postman**

  Postman es una herramienta ampliamente utilizada para el desarrollo y prueba de APIs, especialmente en entornos que emplean arquitecturas RESTful. Permite enviar peticiones HTTP de manera sencilla, visualizar respuestas, configurar colecciones de pruebas y simular distintos escenarios de interacción entre cliente y servidor.

  Como veremos en el capítulo de pruebas, Postman se ha utilizado para verificar el correcto funcionamiento de los endpoints de la API REST desarrollada en Node.js y Express. Gracias a su interfaz gráfica intuitiva y su soporte para distintos métodos HTTP (GET, POST, PUT, DELETE, etc.), ha resultado muy útil para depurar errores, validar respuestas y comprobar el comportamiento esperado de cada ruta antes de integrarla con la aplicación Angular.

  (Postman, Inc., 2025)

Figura 13: Logo de Postman.

- **Visual Studio Code**

  Visual Studio Code (VS Code) es un editor de código fuente desarrollado por Microsoft, siendo actualmente una de las herramientas más populares entre desarrolladores de software de todo el mundo. Su éxito se debe en gran medida a su ligereza, rapidez, personalización y, sobre todo, a la gran cantidad de extensiones y lenguajes disponibles (Microsoft Corporation, 2025).

  VS Code ha tomado gran protagonismo durante el proceso de implementación, siendo el entorno de desarrollo principal tanto para el frontend de la aplicación web en Angular como para el backend en Node.js y Express. Gracias a su integración nativa con Git, su terminal integrada y su excelente soporte para TypeScript, ha facilitado enormemente

las tareas de edición, depuración, control de versiones y ejecución de pruebas. Además, se ha hecho uso de extensiones como Angular Files, Angular Language Service, Error Lens y Material Icon Theme, que han facilitado el mantenimiento de código limpio, coherente y mantenible, así como han facilitado la visualización de la estructura de carpetas y archivos de los proyectos y resaltado de forma más visible los errores, advertencias y sugerencias detectados por el sistema de diagnóstico de Visual Studio Code.

Figura 14: Logo de Visual Studio Code.

# 3
# Análisis

## 3.1. Análisis del Sistema

Esta sección tiene como objetivo realizar un análisis detallado del sistema propuesto, identificando y clasificando tanto los requisitos funcionales como los no funcionales que deben cumplirse para satisfacer las necesidades de los usuarios. Asimismo, se pretende establecer una comprensión clara del comportamiento esperado del sistema desde una perspectiva técnica y de uso. Este análisis sirve como base para las fases posteriores del diseño e implementación, asegurando que el desarrollo se base en especificaciones precisas, coherentes y alineadas con los objetivos del proyecto.

- **Descripción general del sistema**

  El sistema desarrollado en este Trabajo Fin de Grado tiene como finalidad ofrecer una solución integral para la gestión del cuidado y bienestar de las mascotas, tanto desde la perspectiva de los propietarios como desde la de los profesionales veterinarios. La plataforma permite registrar, consultar y dar seguimiento a diferentes aspectos fundamentales de la vida y salud de los animales: historial veterinario, alimentación, medicación, higiene, paseos, eventos, notas, imágenes y gastos económicos asociados.

  El objetivo funcional del sistema es centralizar en una única solución multiplataforma toda la información relevante para el seguimiento y la organización del cuidado animal, proporcionando una herramienta moderna, accesible y adaptable a distintos perfiles de usuario. Esta solución permite no sólo almacenar información, sino también facilitar la toma de decisiones, fomentar hábitos saludables y mejorar la comunicación entre dueños y veterinarios.

  Los **actores principales** del sistema son los siguientes:

- **Dueños de mascotas**: son los usuarios que pueden gestionar los perfiles de sus animales, registrar actividades y gastos, recibir recordatorios y hacer seguimiento de citas médicas y vacunas administradas.
- **Veterinarios**: estos usuarios tienen acceso a funcionalidades específicas relacionadas con la gestión clínica, como la visualización de informes veterinarios, gestión de su agenda profesional, control de vacunas y gestión de solicitudes de cita por parte de los clientes.
- **El propio sistema**: como actor automatizado encargado de tareas como generar y enviar notificaciones o ejecutar tareas en segundo plano.

El sistema se despliega sobre una infraestructura en la nube utilizando los servicios de *Firebase*, que proporciona tanto la base de datos NoSQL *Firestore* como el almacenamiento de archivos multimedia *Firebase Storage*, autenticación de usuarios y *Cloud Functions*.

## 3.2. Aplicación Móvil

En esta sección se expondrán los requisitos funcionales y no funcionales correspondientes a la aplicación Android de manera que se establezca así una base sólida para el desarrollo del producto, garantizando que se satisfagan las necesidades del usuario y que se reduzcan riesgos y costos asociados a cambios durante el desarrollo.

### 3.2.1. Requisitos Funcionales

#### Gestión de usuarios

**RF-01**: La aplicación permitirá el registro de nuevos usuarios proporcionando un correo electrónico y una contraseña, o autenticarse por primera vez mediante su cuenta de Google.

**RF-02**: La aplicación permitirá a los usuarios iniciar sesión mediante correo electrónico y contraseña o con su cuenta de Google.

**RF-03**: La aplicación permitirá a los usuarios restablecer la contraseña asociada a su cuenta indicando su dirección de correo electrónico.

**RF-04**: Los usuarios podrán cerrar sesión en cualquier momento.

**RF-05**: Los usuarios podrán cambiar su foto de perfil en la aplicación.

**RF-06**: Los usuarios podrán modificar el nombre asociado a su perfil en la aplicación.

**RF-07**: La aplicación permitirá al usuario seleccionar si es un veterinario o cliente al registrarse.

#### Gestión de mascotas

**RF-08**: Los usuarios podrán registrar y gestionar uno o más perfiles de mascotas en la aplicación.

**RF-09**: Los usuarios podrán visualizar los datos de sus mascotas registradas en la aplicación.

**RF-10**: Los usuarios podrán editar los datos asociados a sus mascotas.

**RF-11**: Los usuarios podrán eliminar de forma permanente los perfiles de sus mascotas.

**RF-12**: Los usuarios podrán elegir el veterinario asociado a sus mascotas en el momento de crear el perfil de su primera mascota en la aplicación.

#### Seguimiento veterinario

**RF-13**: Los usuarios podrán solicitar citas para sus mascotas con su veterinario, indicando el motivo de la consulta y el tipo de cita.

**RF-14**: Los usuarios podrán solicitar citas para vacunación para sus mascotas, indicando el nombre de la vacuna.

**RF-15**: Los usuarios podrán ver un historial completo de todas las citas veterinarias pasadas de sus mascotas, incluyendo detalles como la fecha y hora, motivo de la consulta, coste, tratamiento y observaciones.

**RF-16**: Los usuarios podrán ver las próximas citas confirmadas de sus mascotas, incluyendo fecha y hora, motivo de la cita y su estado.

**RF-17**: Los usuarios podrán cancelar las citas confirmadas para sus mascotas, siempre que la cita aún no haya ocurrido.

**RF-18**: Los usuarios podrán ver un historial completo de todas las citas pasadas para vacunación de sus mascotas, con detalles como la fecha y hora de la cita, nombre y coste de la vacuna, y la retroalimentación proporcionada po el veterinario.

**RF-19**: Los usuarios podrán ver las próximas citas confirmadas para vacunación de sus mascotas, incluyendo nombre de la vacuna, fecha y hora de la cita y su estado.

**RF-20**: Los usuarios podrán cancelar las citas confirmadas para vacunación de sus mascotas, siempre que la cita aún no se haya producido.

#### Notas

**RF-21**: Los usuarios podrán crear notas asociadas a cada una de sus mascotas, como recordatorios, observaciones de salud, o cualquier otra información.

**RF-22**: Los usuarios podrán visualizar las notas creadas en el perfil de sus mascotas, con detalles como el título, contenido y la fecha y hora en que se modificó por última vez.

**RF-23**: Los usuarios podrán modificar la información de las notas, es decir, su título, contenido y categoría.

**RF-24**: Los usuarios podrán eliminar de forma permanente las notas asociadas a los perfiles de sus mascotas.

**RF-25**: La aplicación permitirá a los usuarios filtrar las notas asociadas a sus mascotas por título y por categoría.

#### Recuerdos

**RF-26**: Los usuarios podrán subir a la aplicación recuerdos asociados a sus mascotas, que serán fotografías o imágenes con un título y una descripción opcional.

**RF-27**: Los usuarios podrán eliminar de forma permanente los recuerdos asociados a sus mascotas.

**RF-28**: Los usuarios podrán ver las fotogrfías subidas en la sección de recuerdos asociadas a sus mascotas, junto con el título y la descripción (si tuviera).

**RF-29**: Los usuarios podrán modificar el título y la descripción de los recuerdos subidos a la aplicación.

### Gestión financiera

**RF-30**: Los usuarios podrán registrar en la aplicación los gastos económicos relacionados con sus mascotas.

**RF-31**: Los usuarios podrán ver un historial completo con todos los gastos asociados a sus mascotas que han registrado en la aplicación.

**RF-32**: Los usuarios podrán eliminar de forma permanente los gastos registrados en la aplicación.

**RF-33**: Los usuarios podrán modificar los datos de los gastos económicos relacionados con sus mascotas, tales como el nombre, la cantidad en euros, la categoría, la fecha y la frecuencia.

**RF-34**: Los usuarios podrán filtrar los gastos económicos por categoría, por frecuencia y por rango de precio (precio mínimo y máximo) para facilitar su búsqueda y organización.

**RF-35**: La aplicación permitirá a los usuarios consultar estadísticas acerca de sus gastos, permitiendo la interacción por parte de los usuarios de manera que puedan filtrar y buscar información específica según la categoría o la fecha.

**RF-36**: La aplicación presentará la información de los gastos de forma visual y númerica, incluyendo resúmenes cuantitativos y gráficos que faciliten la visualización e interpretación de los datos.

### Rastreo de paseos

**RF-37**: La aplicación permitirá el rastreo de paseos de los usuarios con sus mascotas en tiempo real.

**RF-38**: La aplicación permitirá a los usuarios pausar y reanudar un paseo.

**RF-39**: La aplicación permitirá a los usuarios finalizar y guardar un paseo.

**RF-40**: La aplicación registrará la ubicación de los usuarios (si tiene los permisos correspondientes) y pintará en el mapa el recorrido que vayan realizando durante los paseos en tiempo real.

**RF-41**: La aplicación permitirá a los usuarios tomar fotografías durante los paseos (siempre que el usuario haya aceptado los permisos de uso de la cámara a la aplicación).

**RF-42**: La aplicación calculará y registrará la duración de los paseos de los usuarios, iniciando la cuenta desde cero cuando el usuario inicie el paseo.

**RF-43**: La aplicación calculará y registrará la distancia recorrida por los usuarios durante sus paseos.

**RF-44**: La aplicación mostrará una ventana pop up al finalizar el usuario el paseo, incluyendo estadisticas del paseo como la distancia total recorrida y la duración del mismo.

**RF-45**: La aplicación permitirá a los usuarios eliminar un paseo realizado.

**RF-46**: La aplicación permitirá a los usuarios visualizar en un mapa pintado con líneas el recorrido realizado durante los paseos.

**RF-47**: Los usuarios podrán visualizar un historial completo de todos los paseos realizados con sus mascotas.

**RF-48**: La aplicación permitirá a los usuarios visualizar sólo los paseos realizados con la mascota en cuyo perfil se encuentra el usuario.

**RF-49**: La aplicación permitirá a los usuarios consultar estadísticas relacionadas con los paseos completados con cada uno de sus animales, permitiendo a los usuarios filtrar los datos según el mes y año en el que lo realizaron.

**RF-50**: La aplicación mostrará los datos de los paseos de forma visual y numérica, mediante resúmenes cuantitativos y diferentes tipos de gráficas que faciliten la visualización e interpretación de los datos.

### 3.2.2. Requisitos No Funcionales

**RNF-01**: La aplicación móvil debe ser compatible con dispositivos que ejecuten Android 7.0 (API 26) o superior.

**RNF-02**: La interfaz de usuario deberá ser clara, intuitiva y simple, orientada a facilitar la interacción con todo tipo de usuarios.

**RNF-03**: No hay establecido un límite de perfiles de mascotas para un mismo usuario.

**RNF-04**: Todas las mascotas de un mismo usuario tendrán asignado el mismo veterinario.

**RNF-05**: La aplicación móvil deberá ser capaz de adaptarse automáticamente al idioma configurado por el usuario en su dispositivo, soportando, como mínimo, español e inglés.

### 3.2.3. Casos de Uso

| **Nombre** | **Descripción** |
|---|---|
| Caso de uso | CU-01: Registro de un usuario |
| Requisito | RF-01 |
| Precondiciones | El usuario no está registrado en la aplicación. |
| Descripción | El usuario introduce todos los datos requeridos (nombre, correo electrónico, contraseña...) y pulsa el botón de registrarse. |
| Postcondiciones | La aplicación guarda los datos del usuario en la base de datos.<br>El usuario queda autenticado en la aplicación. |

Tabla 1: CU-01 - Registro

| **Nombre** | **Descripción** |
|---|---|
| Caso de uso | CU-02: Inicio de sesión |
| Requisito | RF-02 |
| Precondiciones | El usuario ya está registrado en la aplicación.<br>El usuario se encuentra con la sesión cerrada |
| Descripción | El usuario introduce su correo electrónico y contraseña y pulsa el botón de iniciar sesión. |
| Postcondiciones | La aplicación dirige al usuario a la pantalla principal de la aplicación. |

Tabla 2: CU-02 - Inicio de sesión

| Nombre | Descripción |
|---|---|
| Caso de uso | CU-03: Recuperar contraseña |
| Requisito | RF-03 |
| Precondiciones | El usuario ya está registrado en la aplicación. |
| Descripción | El usuario introduce el correo electrónico asociado a su cuenta registrada en la aplicación y pulsa el botón de recuperar. |
| Postcondiciones | El servidor de la aplicación envía un correo electrónico de recuperación de la contraseña a la dirección de correo indicada por el usuario. |

Tabla 3: CU-03 - Recuperar contraseña

| Nombre | Descripción |
|---|---|
| Caso de uso | CU-04: Cerrar sesión |
| Requisito | RF-04 |
| Precondiciones | El usuario se encuentra con la sesión iniciada. |
| Descripción | El usuario pulsa sobre el botón de la esquina superior izquierda y, en el menú desplegado, pulsa la opción Çerrar Sesión". |
| Postcondiciones | La aplicación cierra la sesión del usuario y lo dirige a la pantalla de inicio de sesión. |

Tabla 4: CU-04 - Cerrar sesión

| Nombre | Descripción |
|---|---|
| Caso de uso | CU-05: Editar perfil |
| Requisito | RF-05, RF-06 |
| Precondiciones | El usuario se encuentra con la sesión iniciada. |
| Descripción | El usuario modifica su nombre y/o foto de perfil. |
| Postcondiciones | Los cambios realizados por el usuario se actualizan en la base de datos. Los cambios realizados por el usuario se reflejan en la aplicación. |

Tabla 5: CU-05 - Editar perfil

| Nombre | Descripción |
|---|---|
| Caso de uso | CU-06: Añadir mascota |
| Requisito | RF-08 |
| Precondiciones | El usuario se encuentra con la sesión iniciada. |
| Descripción | El usuario registra el perfil de una nueva mascota en la aplicación. |
| Postcondiciones | La aplicación almacena en la base de datos los datos de la mascota creada.<br>El perfil de la mascota aparece en la pantalla principal de la aplicación. |

Tabla 6: CU-06 - Añadir mascota

| Nombre | Descripción |
|---|---|
| Caso de uso | CU-07: Editar mascota |
| Requisito | RF-10 |
| Precondiciones | El usuario ha iniciado sesión en la aplicación.<br>El usuario tiene al menos un perfil de mascota creado. |
| Descripción | El usuario modifica los datos que desee de su mascota. |
| Postcondiciones | La aplicación actualiza la base de datos con los cambios realizados.<br>Los cambios realizados se visualizan en la aplicación. |

Tabla 7: CU-07 - Editar mascota

| Nombre | Descripción |
|---|---|
| Caso de uso | CU-08: Eliminar mascota |
| Requisito | RF-11 |
| Precondiciones | El usuario ha iniciado sesión en la aplicación.<br>El usuario tiene al menos un perfil de mascota creado. |
| Descripción | El usuario elimina el perfil de una mascota de la aplicación. |
| Postcondiciones | La aplicación elimina todos los datos de la mascota y todos los asociados con esta de la base de datos. |

Tabla 8: CU-08 - Eliminar mascota

| Nombre | Descripción |
|---|---|
| Caso de uso | CU-09: Solicitar cita |
| Requisito | RF-13 |
| Precondiciones | El usuario ha iniciado sesión en la aplicación.<br>El usuario tiene al menos un perfil de mascota creado. |
| Descripción | El usuario envía una solicitud de cita para su mascota al veterinario asociado. |
| Postcondiciones | La aplicación crea una cita con estado pendiente y la almacena en la base de datos.<br>La cita se añade y se puede visualizar en la lista de citas pendientes. |

Tabla 9: CU-09 - Solicitar cita

| Nombre | Descripción |
|---|---|
| Caso de uso | CU-10: Solicitar vacuna |
| Requisito | RF-14 |
| Precondiciones | El usuario ha iniciado sesión en la aplicación.<br>El usuario tiene al menos un perfil de mascota creado. |
| Descripción | El usuario envía una solicitud de vacuna para su mascota al veterinario asociado. |
| Postcondiciones | La aplicación crea una vacuna con estado pendiente y la almacena en la base de datos.<br>La vacuna se añade y se puede visualizar en la lista de vacunas pendientes. |

Tabla 10: CU-10 - Solicitar vacuna

| Nombre | Descripción |
|---|---|
| Caso de uso | CU-11: Visualizar historial de citas |
| Requisito | RF-15, RF-16 |
| Precondiciones | El usuario ha iniciado sesión en la aplicación.<br>El usuario tiene al menos un perfil de mascota creado. |
| Descripción | El usuario puede visualizar el historial de citas próximas, pasadas y pendientes de confirmación para sus mascotas. |
| Postcondiciones | Ninguna |

Tabla 11: CU-11 - Visualizar historial de citas

| Nombre | Descripción |
|---|---|
| Caso de uso | CU-12: Visualizar historial de vacunas |
| Requisito | RF-18, RF-19 |
| Precondiciones | El usuario ha iniciado sesión en la aplicación.<br>El usuario tiene al menos un perfil de mascota creado. |
| Descripción | El usuario puede visualizar el historial de vacunas próximas, pasadas y pendientes para sus mascotas. |
| Postcondiciones | Ninguna |

Tabla 12: CU-12 - Visualizar historial de vacunas

| Nombre | Descripción |
|---|---|
| Caso de uso | CU-13: Cancelar cita |
| Requisito | RF-17 |
| Precondiciones | El usuario ha iniciado sesión en la aplicación.<br>El usuario tiene al menos un perfil de mascota creado.<br>El usuario tiene alguna mascota con una cita próxima o pendiente. |
| Descripción | El usuario cancela una cita próxima o pendiente. |
| Postcondiciones | La aplicación actualiza el estado de la cita en la base de datos.<br>Los cambios se reflejan en el historial de citas de la aplicación, es decir, la cita ya no aparece. |

Tabla 13: CU-13 - Cancelar cita

| Nombre | Descripción |
|---|---|
| Caso de uso | CU-14: Cancelar vacuna |
| Requisito | RF-20 |
| Precondiciones | El usuario ha iniciado sesión en la aplicación.<br>El usuario tiene al menos un perfil de mascota creado.<br>El usuario tiene alguna mascota con una vacuna próxima o pendiente. |
| Descripción | El usuario cancela una vacuna próxima o pendiente. |
| Postcondiciones | La aplicación actualiza el estado de la vacuna en la base de datos.<br>Los cambios se reflejan en el historial de vacunas de la aplicación, es decir, la vacuna ya no aparece. |

Tabla 14: CU-14 - Cancelar vacuna

| Nombre | Descripción |
|---|---|
| Caso de uso | CU-15: Añadir nota |
| Requisito | RF-21 |
| Precondiciones | El usuario ha iniciado sesión en la aplicación.<br>El usuario tiene al menos un perfil de mascota creado. |
| Descripción | El usuario crea y añade una nota asociada a una mascota en la aplicación. |
| Postcondiciones | La aplicación guarda los datos de la nota en la base de datos.<br>La nota se crea y aparece en el historial de notas. |

Tabla 15: CU-15 - Añadir nota

| Nombre | Descripción |
|---|---|
| Caso de uso | CU-16: Visualizar historial de notas |
| Requisito | RF-22 |
| Precondiciones | El usuario ha iniciado sesión en la aplicación.<br>El usuario tiene al menos un perfil de mascota creado. |
| Descripción | El usuario puede visualizar el historial de notas creadas asociadas a cada una de sus mascotas. |
| Postcondiciones | Ninguna |

Tabla 16: CU-16 - Visualizar historial de notas

| Nombre | Descripción |
|---|---|
| Caso de uso | CU-17: Editar nota |
| Requisito | RF-23 |
| Precondiciones | El usuario ha iniciado sesión en la aplicación.<br>El usuario tiene al menos un perfil de mascota creado.<br>El usuario tiene añadida al menos una nota. |
| Descripción | El usuario modifica la información de una nota (título, contenido y categoría). |
| Postcondiciones | La aplicación actualiza la nota en la base de datos.<br>Los cambios se reflejan en el historial de notas, viéndose la nota modificada con los nuevos datos. |

Tabla 17: CU-17 - Editar nota

| Nombre | Descripción |
|---|---|
| Caso de uso | CU-18: Eliminar nota |
| Requisito | RF-24 |
| Precondiciones | El usuario ha iniciado sesión en la aplicación.<br>El usuario tiene al menos un perfil de mascota creado.<br>El usuario tiene añadida al menos una nota. |
| Descripción | El usuario elimina una nota. |
| Postcondiciones | La aplicación elimina la nota de la base de datos.<br>La nota eliminada no aparece en el historial de notas. |

Tabla 18: CU-18 - Eliminar nota

| Nombre | Descripción |
|---|---|
| Caso de uso | CU-19: Filtrar notas |
| Requisito | RF-25 |
| Precondiciones | El usuario ha iniciado sesión en la aplicación.<br>El usuario tiene al menos un perfil de mascota creado. |
| Descripción | El usuario filtra las notas que aparecen en el historial según el título y la categoría que indique. |
| Postcondiciones | El historial de notas se actualiza con las notas que contengan en el título el texto indicado en el buscador, y que pertenezcan a la categoría seleccionada. |

Tabla 19: CU-19 - Filtrar notas

| Nombre | Descripción |
|---|---|
| Caso de uso | CU-20: Añadir recuerdo |
| Requisito | RF-26 |
| Precondiciones | El usuario ha iniciado sesión en la aplicación.<br>El usuario tiene al menos un perfil de mascota creado. |
| Descripción | El usuario crea y añade un recuerdo asociada a una mascota en la aplicación. |
| Postcondiciones | La aplicación guarda los datos del recuerdo en la base de datos.<br>El recuerdo se crea y se visualiza en el historial de recuerdos. |

Tabla 20: CU-20 - Añadir recuerdo

| Nombre | Descripción |
|---|---|
| Caso de uso | CU-21: Eliminar recuerdo |
| Requisito | RF-27 |
| Precondiciones | El usuario ha iniciado sesión en la aplicación.<br>El usuario tiene al menos un perfil de mascota creado.<br>El usuario tiene añadido al menos un recuerdo. |
| Descripción | El usuario elimina un recuerdo. |
| Postcondiciones | La aplicación elimina el recuerdo de la base de datos.<br>El recuerdo eliminado no aparece en el historial de recuerdos. |

Tabla 21: CU-21 - Eliminar recuerdo

| Nombre | Descripción |
|---|---|
| Caso de uso | CU-22: Visualizar recuerdos |
| Requisito | RF-28 |
| Precondiciones | El usuario ha iniciado sesión en la aplicación.<br>El usuario tiene al menos un perfil de mascota creado. |
| Descripción | El usuario puede visualizar los recuerdos creados asociados a cada una de sus mascotas. |
| Postcondiciones | Ninguna |

Tabla 22: CU-22 - Visualizar recuerdos

| Nombre | Descripción |
|---|---|
| Caso de uso | CU-23: Editar recuerdo |
| Requisito | RF-29 |
| Precondiciones | El usuario ha iniciado sesión en la aplicación.<br>El usuario tiene al menos un perfil de mascota creado.<br>El usuario tiene añadido al menos un recuerdo. |
| Descripción | El usuario modifica la información de un recuerdo (título y descripción). |
| Postcondiciones | La aplicación actualiza el recuerdo en la base de datos.<br>Los cambios se reflejan en la lista de recuerdos, viéndose el recuerdo modificado con los nuevos datos. |

Tabla 23: CU-23 - Editar recuerdo

| Nombre | Descripción |
|---|---|
| Caso de uso | CU-24: Añadir gasto |
| Requisito | RF-30 |
| Precondiciones | El usuario ha iniciado sesión en la aplicación.<br>El usuario tiene al menos un perfil de mascota creado. |
| Descripción | El usuario crea y añade un gasto asociado a una mascota en la aplicación. |
| Postcondiciones | La aplicación guarda los datos del gasto en la base de datos.<br>El gasto se crea y aparece en el historial de gastos. |

Tabla 24: CU-24 - Añadir gasto

| Nombre | Descripción |
|---|---|
| Caso de uso | CU-25: Visualizar historial de gastos |
| Requisito | RF-31 |
| Precondiciones | El usuario ha iniciado sesión en la aplicación.<br>El usuario tiene al menos un perfil de mascota creado. |
| Descripción | El usuario puede visualizar el historial de gastos creados asociados a cada una de sus mascotas. |
| Postcondiciones | Ninguna |

Tabla 25: CU-25 - Visualizar historial de gastos

| Nombre | Descripción |
|---|---|
| Caso de uso | CU-26: Eliminar gasto |
| Requisito | RF-32 |
| Precondiciones | El usuario ha iniciado sesión en la aplicación.<br>El usuario tiene al menos un perfil de mascota creado.<br>El usuario tiene añadido al menos un gasto. |
| Descripción | El usuario elimina un gasto. |
| Postcondiciones | La aplicación elimina el gasto de la base de datos.<br>El gasto eliminado no aparece en el historial de gastos. |

Tabla 26: CU-26 - Eliminar gasto

| Nombre | Descripción |
|---|---|
| Caso de uso | CU-27: Editar gasto |
| Requisito | RF-33 |
| Precondiciones | El usuario ha iniciado sesión en la aplicación.<br>El usuario tiene al menos un perfil de mascota creado.<br>El usuario tiene añadido al menos un gasto. |
| Descripción | El usuario modifica la información de un gasto. |
| Postcondiciones | La aplicación actualiza el gasto en la base de datos.<br>Los cambios se reflejan en el historial de gastos, viéndose el gasto modificado con los nuevos datos. |

Tabla 27: CU-27 - Editar gasto

| Nombre | Descripción |
|---|---|
| Caso de uso | CU-28: Filtrar gastos |
| Requisito | RF-34 |
| Precondiciones | El usuario ha iniciado sesión en la aplicación.<br>El usuario tiene al menos un perfil de mascota creado. |
| Descripción | El usuario filtra los gastos que aparecen en el historial según su categoría, frecuencia y rango de precio. |
| Postcondiciones | El historial de gastos se actualiza con los gastos que tengan las características indicadas por los filtros. |

Tabla 28: CU-28 - Filtrar gastos

| Nombre | Descripción |
|---|---|
| Caso de uso | CU-29: Empezar paseo |
| Requisito | RF-37 |
| Precondiciones | El usuario ha iniciado sesión en la aplicación.<br>El usuario tiene al menos un perfil de mascota creado.<br>El usuario no tiene ningún paseo empezado.<br>El usuario ha concedido permisos a la aplicación para registrar su ubicación.<br>El cronómetro está a 00:00:00.<br>La distancia recorrida está a 0.00 m. |
| Descripción | El usuario inicia un paseo con su mascota pulsando el botón de play. |
| Postcondiciones | La aplicación almacena en la base de datos la información asociada a este.<br>El botón de play cambiar a un icono rojo que representa parar el paseo.<br>El cronómetro se pone en marcha.<br>Se habilita el botón de finalizar paseo. |

Tabla 29: CU-29 - Empezar paseo

| Nombre | Descripción |
|---|---|
| Caso de uso | CU-30: Parar paseo |
| Requisito | RF-38 |
| Precondiciones | El usuario ha iniciado sesión en la aplicación.<br>El usuario tiene al menos un perfil de mascota creado.<br>El usuario ha empezado un paseo.<br>El paseo empezado no está detenido.<br>El usuario ha concedido permisos a la aplicación para registrar su ubicación.<br>El cronómetro está en marcha. |
| Descripción | El usuario para un paseo con su mascota pulsando el botón de parar. |
| Postcondiciones | El botón de parar cambia a un icono amarillo que representa reanudar el paseo.<br>El cronómetro se detiene. |

Tabla 30: CU-30 - Parar paseo

| Nombre | Descripción |
|---|---|
| Caso de uso | CU-31: Reanudar paseo |
| Requisito | RF-38 |
| Precondiciones | El usuario ha iniciado sesión en la aplicación.<br>El usuario tiene al menos un perfil de mascota creado.<br>El usuario ha empezado un paseo.<br>El paseo empezado está detenido.<br>El usuario ha concedido permisos a la aplicación para registrar su ubicación.<br>El cronómetro está detenido. |
| Descripción | El usuario reanuda un paseo con su mascota pulsando el botón de reanudar. |
| Postcondiciones | El paseo se vuelve a poner en marcha.<br>El botón de reanudar cambia a un icono rojo que representa parar el paseo.<br>El cronómetro se pone en marcha por donde iba. |

Tabla 31: CU-31 - Reanudar paseo

| Nombre | Descripción |
|---|---|
| Caso de uso | CU-32: Finalizar paseo |
| Requisito | RF-39 |
| Precondiciones | El usuario ha iniciado sesión en la aplicación.<br>El usuario tiene al menos un perfil de mascota creado.<br>El usuario ha empezado un paseo.<br>El usuario ha concedido permisos a la aplicación para registrar su ubicación. |
| Descripción | El usuario finaliza un paseo con su mascota pulsando el botón de finalizar paseo. |
| Postcondiciones | La aplicación muestra una ventana emergente con información sobre el paseo realizado.<br>El cronómetro se pone a 00:00:00.<br>La distancia recorrida se pone a 0.00 m.<br>Se deshabilita el botón de finalizar paseo. |

Tabla 32: CU-32 - Finalizar paseo

| Nombre | Descripción |
|---|---|
| Caso de uso | CU-33: Tomar fotografía en un paseo |
| Requisito | RF-41 |
| Precondiciones | El usuario ha iniciado sesión en la aplicación.<br>El usuario tiene al menos un perfil de mascota creado.<br>El usuario ha empezado un paseo.<br>El usuario ha concedido permisos a la aplicación para registrar su ubicación.<br>El usuario ha concedido permisos a la aplicación para hacer uso de la cámara.<br>El dispositivo móvil tiene cámara trasera o delantera. |
| Descripción | El usuario pulsa el botón de la cámara y toma una fotografía durante un paseo. |
| Postcondiciones | La aplicación guarda en *Firebase Storage* la foto realizada. |

Tabla 33: CU-33 - Tomar fotografía en un paseo

| Nombre | Descripción |
|---|---|
| Caso de uso | CU-34: Eliminar paseo |
| Requisito | RF-45 |
| Precondiciones | El usuario ha iniciado sesión en la aplicación.<br>El usuario tiene al menos un perfil de mascota creado.<br>El usuario ha terminado un paseo. |
| Descripción | El usuario elimina un paseo realizado. |
| Postcondiciones | La aplicación elimina de la base de datos el paseo y toda la información relacionada con este.<br>El paseo no aparece en la lista de paseos realizados. |

Tabla 34: CU-34 - Eliminar paseo

| Nombre | Descripción |
|---|---|
| Caso de uso | CU-35: Visualizar recorrido de un paseo |
| Requisito | RF-46 |
| Precondiciones | El usuario ha iniciado sesión en la aplicación.<br>El usuario tiene al menos un perfil de mascota creado.<br>El usuario ha terminado un paseo. |
| Descripción | El usuario visualiza el recorrido de un paseo representado por líneas pintadas sobre un mapa de Google Maps. |
| Postcondiciones | Ninguna. |

Tabla 35: CU-35 - Visualizar recorrido de un paseo

| Nombre | Descripción |
|---|---|
| Caso de uso | CU-36: Ver lista de paseos realizados |
| Requisito | RF-47, RF-48 |
| Precondiciones | El usuario ha iniciado sesión en la aplicación.<br>El usuario tiene al menos un perfil de mascota creado. |
| Descripción | El usuario puede ver una lista con los paseos realizados con sus mascotas. |
| Postcondiciones | Ninguna. |

Tabla 36: CU-36 - Ver lista de paseos realizados

| Nombre | Descripción |
|---|---|
| Caso de uso | CU-37: Registro de localizaciones en tiempo real |
| Requisito | RF-40 |
| Precondiciones | El usuario ha iniciado sesión en la aplicación.<br>El usuario tiene al menos un perfil de mascota creado.<br>El usuario ha empezado un paseo.<br>El paseo empezado no está detenido.<br>La aplicación tiene los permisos concedidos por el usuario para registrar su ubicación. |
| Descripción | La aplicación registra de forma continua las ubicaciones del usuario durante un paseo. |
| Postcondiciones | Las localizaciones del usuario se guardan en la base de datos vinculadas al usuario y al paseo. |

Tabla 37: CU-37 - Registro de localizaciones en tiempo real

| Nombre | Descripción |
|---|---|
| Caso de uso | CU-38: Cálculo de duración durante un paseo |
| Requisito | RF-42 |
| Precondiciones | El usuario ha iniciado sesión en la aplicación.<br>El usuario tiene al menos un perfil de mascota creado.<br>La aplicación tiene los permisos concedidos por el usuario para registrar su ubicación.<br>Hay un paseo iniciado, en curso. |
| Descripción | Durante un paseo, la aplicación activa un cronómetro y calcula la duración del paseo. |
| Postcondiciones | La duración queda registrada en la base de datos como un campo del documento correspondiente al paseo. |

Tabla 38: CU-38 - Cálculo de duración de un paseo

| Nombre | Descripción |
|---|---|
| Caso de uso | CU-39: Cálculo de distancia recorrida durante un paseo |
| Requisito | RF-43 |
| Precondiciones | El usuario ha iniciado sesión en la aplicación.<br>El usuario tiene al menos un perfil de mascota creado.<br>La aplicación tiene los permisos concedidos por el usuario para registrar su ubicación.<br>El usuario ha iniciado un paseo. |
| Descripción | Durante un paseo, la aplicación calcula la distancia entre dos coordenadas consecutivas del usuario y acumula el total recorrido. |
| Postcondiciones | La distancia queda registrada en la base de datos como un campo del documento correspondiente al paseo. |

Tabla 39: CU-39 - Cálculo de distancia recorrida de un paseo

## 3.3. Aplicación Web

### 3.3.1. Requisitos Funcionales

**RF-01**: Los veterinarios podrán visualizar todos sus clientes y, de cada cliente, sus mascotas registradas en la aplicación.

**RF-02**: La aplicación permitirá a los veterinarios filtrar a los clientes por su nombre.

**RF-03**: Los veterinarios podrán visualizar en un calendario integrado en la aplicación los eventos confirmados (citas y vacunas).

**RF-04**: Los veterinarios podrán completar los datos correspondientes a un evento que ya ha ocurrido, tales como su coste, diagnóstico, tratamiento o retroalimentación.

**RF-05**: Los veterinarios podrán confirmar citas y vacunas a los clientes asignándoles una fecha y hora.

**RF-06**: La aplicación sólo permitirá cambiar la fecha y hora de un evento futuro, es decir, siempre que todavía no haya ocurrido.

**RF-07**: Los veterinarios podrán rechazar una solicitud de cita o vacuna de un cliente.

**RF-08**: La aplicación enviará una notificación al cliente cuando su veterinario le asigne una fecha a una cita o vacuna pendiente de confirmación.

**RF-09**: La aplicación enviará una notificación al cliente cuando su veterinario rechace una cita o vacuna pendiente de confirmación.

**RF-10**: La aplicación enviará una notificación al cliente cuando su veterinario complete los datos correspondientes a una cita o vacuna que ya se ha producido.

**RF-11**: La aplicación enviará una notificación al cliente cuando su veterinario cambie la fecha/hora de alguna cita o vacuna.

**RF-12**: La aplicación permitirá a los veterinarios poder visualizar en el calendario su agenda mediante una vista mensual, semanal y diaria.

**RF-13**: La aplicación permitirá a los veterinarios visualizar, en un panel interactivo, estadísticas relacionadas con sus ingresos, clientes y animales a su cargo, citas atendidas y vacunas administradas. Estos datos se mostrarán en formato gráfico y numérico, facilitando su análisis e interpretación.

**RF-14**: Todos los días las 9:00 a.m. el sistema enviará automáticamente notificaciones recordatorio a los dispositivos de los dueños de los mascotas que tengan una cita o vacuna programada para el día siguiente.

### 3.3.2. Requisitos No Funcionales

**RNF-01**: La aplicación debe ser funcional en los principales navegadores modernos, incluyendo Google Chrome, Mozilla Firefox y Microsoft Edge.

**RNF-02**: El sistema debe limitar la visibilidad de los datos e información sensible únicamente a los dueños y a los veterinarios, es decir, a los usuarios registrados en la aplicación.

**RNF-03**: El sistema requerirá conexión permanente a Internet para poder funcionar correctamente, ya que todas las funcionalidades dependen del acceso en tiempo real a la base de datos remota.

### 3.3.3. Casos de Uso

| Nombre | Descripción |
|---|---|
| Caso de uso | CU-01: Ver clientes y mascotas |
| Requisito | RF-01 |
| Precondiciones | El usuario ha iniciado sesión en la aplicación. |
| Descripción | El usuario puede ver los clientes que tiene y sus mascotas, es decir, los animales que tiene asignados. |
| Postcondiciones | Ninguna. |

Tabla 40: CU-01 - Ver clientes y mascotas

| Nombre | Descripción |
|---|---|
| Caso de uso | CU-02: Filtrar clientes |
| Requisito | RF-02 |
| Precondiciones | El usuario ha iniciado sesión en la aplicación. |
| Descripción | El usuario puede ver sus clientes filtrados por su nombre. |
| Postcondiciones | La aplicación muestra los clientes cuyo nombre contenga la cadena de texto introducida por el usuario en el filtro. |

Tabla 41: CU-02- Filtrar clientes

| Nombre | Descripción |
|---|---|
| Caso de uso | CU-03: Visualizar los eventos confirmados |
| Requisito | RF-03 |
| Precondiciones | El usuario ha iniciado sesión en la aplicación. |
| Descripción | El usuario puede ver los eventos que tiene confirmados en un calendario integrado en la aplicación. |
| Postcondiciones | Ninguna. |

Tabla 42: CU-03 - Visualizar los eventos confirmados.

| Nombre | Descripción |
|---|---|
| Caso de uso | CU-04: Completar datos de una cita o vacuna |
| Requisito | RF-04 |
| Precondiciones | El usuario ha iniciado sesión en la aplicación.<br>El usuario ya ha atendido alguna cita o vacuna. |
| Descripción | El usuario puede completar los datos relacionados con una cita o vacuna pasada, como su coste, diagnóstico o tratamiento. |
| Postcondiciones | La aplicación actualiza los datos de la cita o vacuna en la base de datos.<br>La aplicación envía al dispositivo del cliente una notificación informando de que tiene datos disponibles para una cita o vacuna. |

Tabla 43: CU-04 - Completar datos de una cita o vacuna.

| Nombre | Descripción |
|---|---|
| Caso de uso | CU-05: Confirmar una cita o vacuna |
| Requisito | RF-05 |
| Precondiciones | El usuario ha iniciado sesión en la aplicación.<br>El usuario tiene una cita o vacuna pendiente. |
| Descripción | El usuario confirma una cita o vacuna pendiente, asignándole una fecha y hora en una franja disponible. |
| Postcondiciones | La aplicación actualiza la cita o vacuna en la base de datos.<br>La aplicación envía al dispositivo del cliente una notificación informando de que la cita o vacuna ha sido confirmada.<br>La cita o vacuna confirmada aparece en el calendario de eventos del usuario.<br>La cita o vacuna se elimina de la lista de citas o vacunas pendientes. |

Tabla 44: CU-05 - Confirmar una cita o vacuna.

| Nombre | Descripción |
|---|---|
| Caso de uso | CU-06: Reprogramar una cita o vacuna. |
| Requisito | RF-06 |
| Precondiciones | El usuario ha iniciado sesión en la aplicación.<br>El usuario tiene una cita o vacuna confirmada para el futuro. |
| Descripción | El usuario cambia la fecha y hora de una cita o vacuna próxima, esto es, que todavía no ha ocurrido. |
| Postcondiciones | La aplicación actualiza la cita o vacuna en la base de datos.<br>La aplicación envía al dispositivo del cliente una notificación informando de que la cita o vacuna ha sido reprogramada.<br>La cita o vacuna confirmada aparece en el calendario de eventos del usuario en la nueva fecha y hora. |

Tabla 45: CU-06 - Reprogramar una cita o vacuna

| Nombre | Descripción |
|---|---|
| Caso de uso | CU-07: Rechazar una cita o vacuna |
| Requisito | RF-07 |
| Precondiciones | El usuario ha iniciado sesión en la aplicación.<br>El usuario tiene una cita o vacuna pendiente. |
| Descripción | El usuario rechaza una cita o vacuna pendiente. |
| Postcondiciones | La aplicación actualiza la cita o vacuna en la base de datos.<br>La aplicación envía al dispositivo del cliente una notificación informando de que la cita o vacuna ha sido rechazada.<br>La cita o vacuna se elimina de la lista de citas o vacunas pendientes. |

Tabla 46: CU-07 - Rechazar una cita o vacuna.

| Nombre | Descripción |
|---|---|
| Caso de uso | CU-08: Visualizar estadísticas |
| Requisito | RF-13 |
| Precondiciones | El usuario ha iniciado sesión en la aplicación. |
| Descripción | El usuario despliega el menú lateral y abre una pantalla de estadísticas e informes. |
| Postcondiciones | La aplicación muestra las estadísticas y datos correspondientes. |

Tabla 47: CU-08 - Visualizar estadísticas.

| Nombre | Descripción |
|---|---|
| Caso de uso | CU-09: Enviar notificación de recordatorio |
| Requisito | RF-14 |
| Precondiciones | Un cliente tiene una cita o vacuna confirmada. |
| Descripción | El día anterior de la fecha de la cita o vacuna, a las 9:00 a.m., la aplicación envía al dispositivo del cliente una notificación para recordarle la cita o vacuna del día siguiente. |
| Postcondiciones | El cliente recibe en su dispositivo la notificación. |

Tabla 48: CU-09 - Enviar notificación de recordatorio.

# 4
# Diseño

## 4.1. Diseño del Sistema

En este capítulo se describe el proceso de diseño del sistema propuesto, partiendo de los requisitos funcionales y no funcionales identificados en la fase de análisis. El objetivo es definir la estructura y organización interna de la solución, estableciendo los modelos, arquitecturas y componentes que servirán de base para su posterior implementación.

El diseño parte de los resultados obtenidos en el capítulo de Análisis, de modo que cada decisión tomada — tanto a nivel arquitectónico como a nivel de interfaz — responde de forma específica a las necesidades y restricciones detectadas. De esta manera, se garantiza la coherencia entre lo que el sistema debe hacer y cómo se implementará para lograrlo, asegurando el cumplimiento de los objetivos y requisitos establecidos.

## 4.2. Arquitectura

Para el desarrollo de este proyecto se ha optado por una arquitectura de tipo **cliente–servidor**, lo que permite que la aplicación (cliente) realice peticiones a un servidor y reciba las respuestas correspondientes.

En el caso de la aplicación móvil, la interfaz de usuario actúa como cliente, gestionando la interacción directa con el usuario y enviando solicitudes a los servicios del servidor. El servidor, implementado sobre la plataforma de Google Firebase, asume el rol de proveedor de servicios, procesando las peticiones y devolviendo la información o ejecutando las acciones solicitadas.

Firebase centraliza diferentes funcionalidades necesarias para la aplicación, como la gestión de la configuración remota (*Remote Config*), la ejecución de funciones en la nube (*Cloud Functions*) y la comunicación en tiempo real entre cliente y servidor. Esto permite que cualquier cambio o actualización de configuración se distribuya automáticamente a todos los dis-

positivos cliente conectados, sin necesidad de actualizar la aplicación.

La Figura 15 muestra de forma esquemática esta arquitectura, donde las aplicaciones móviles se conectan a través de Internet al servicio de Remote Config de Firebase, el cual puede recibir datos tanto desde herramientas personalizadas como desde funciones en la nube o servidores propios, garantizando la sincronización y el control centralizado de la configuración.

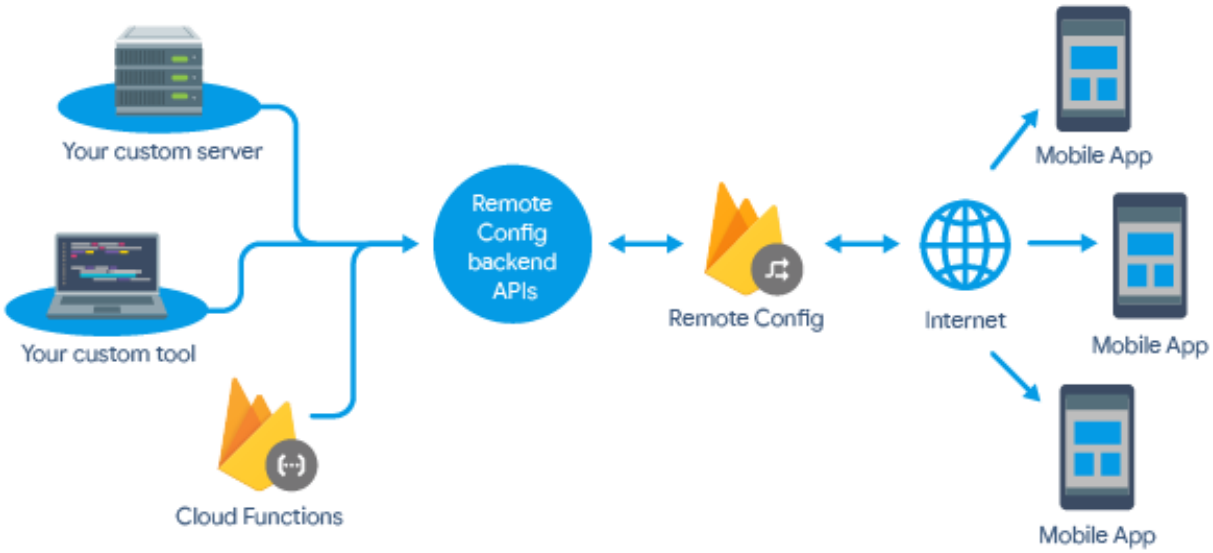



Figura 15: Arquitectura cliente-servidor con Firebase.

Fuente: https://firebase.google.com/docs/remote-config/automate-rc

En el caso de la aplicación web, la arquitectura cliente–servidor se implementa siguiendo un patrón de comunicación basado en *API REST*, como muestra la figura 16. El cliente está desarrollado con *Angular*, donde los distintos servicios encapsulan la lógica de comunicación y se encargan de realizar las peticiones HTTP a las rutas definidas en la API dentro del servidor.

El backend está construido con *Node.js* y el framework *Express*, actuando como servidor intermedio que recibe las solicitudes del cliente, procesa la lógica de negocio correspondiente y accede a la base de datos para obtener o modificar la información solicitada.

La base de datos utilizada es Cloud Firestore, un servicio de Firebase que ofrece almacenamiento NoSQL en la nube, sincronización en tiempo real y escalabilidad automática. El backend se conecta con Firestore mediante el SDK oficial de Firebase para Node.js, lo que permite realizar operaciones de lectura y escritura de forma eficiente.

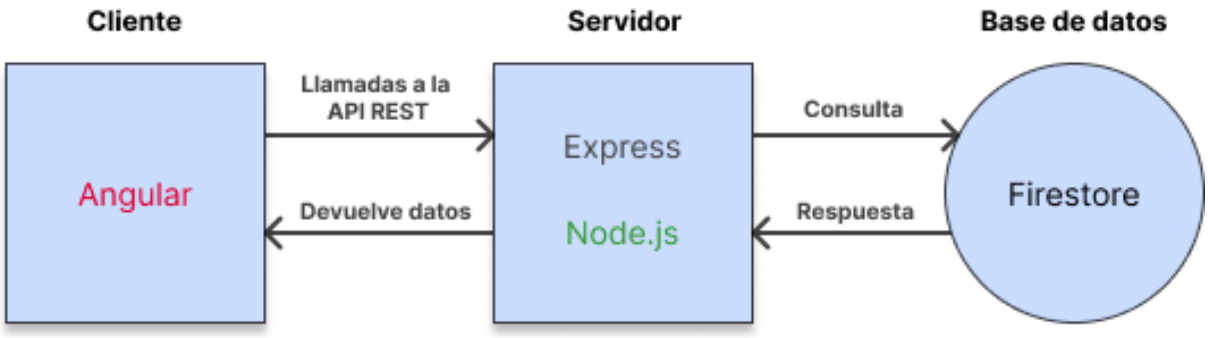



Figura 16: Arquitectura cliente-servidor de la aplicación web.

Fuente: Elaboración propia.

De este modo, tanto en la aplicación móvil como en la web, la arquitectura garantiza una clara separación entre el cliente (responsable de la interacción con el usuario) y el servidor (responsable de la gestión de datos y lógica de negocio), aprovechando las capacidades de la nube para asegurar disponibilidad, consistencia y mantenimiento centralizado.

## 4.3. Base de Datos

Como ya se ha mencionado, los datos del sistema completo se alojan en la nube, en concreto en *Cloud Firestore*. La elección de un esquema **NoSQL** responde a su facilidad de desarrollo, su funcionalidad y el rendimiento a gran escala. Algunas de las ventajas de las bases de datos NoSQL son (AWS, 2025):

- Flexibilidad: Las bases de datos NoSQL ofrecen esquemas flexibles que permiten un desarrollo más rápido e iterativo.
- Escalabilidad: Están diseñadas para escalar horizontalmente, lo que permite que puedan manejar grandes volúmenes de datos utilizando múltiples servidores.
- Mayor velocidad y rendimiento: Estas bases de datos están optimizadas para consultas rápidas, especialmente en grandes conjuntos de datos. Su diseño permite un mejor manejo de la información, lo que se traduce en tiempos de respuesta más rápidos.

La base de datos utilizada en el presente proyecto es Cloud Firestore, un sistema de gestión de bases de datos NoSQL orientado a documentos proporcionado por Firebase. Firestore organiza la información en colecciones y documentos, donde cada documento contiene campos de

datos estructurados en formato clave-valor, y puede almacenar subcolecciones, permitiendo así modelar jerarquías de datos complejas de manera flexible. Esta característica se adapta de manera óptima al sistema integral desarrollado en este TFG, que requiere manejar información diversa y dinámica relacionada con usuarios, mascotas, visitas veterinarias, vacunas, paseos, notas y multimedia.

La elección de Firestore se debe principalmente a su capacidad de sincronización en tiempo real, que garantiza que tanto la aplicación web como la móvil reflejen de manera inmediata cualquier actualización de los datos, mejorando la experiencia de usuario y facilitando la gestión coordinada por parte de dueños y profesionales veterinarios. Además, su escalabilidad automática y su modelo flexible de datos permite incorporar nuevas funcionalidades sin necesidad de realizar reestructuraciones complejas de la base de datos Finalmente, la integración nativa con Firebase Authentication y Firebase Storage simplifica la gestión segura de usuarios y el almacenamiento de imágenes, lo que contribuye a un desarrollo más ágil y coherente con los objetivos de centralización y seguimiento integral del bienestar de las mascotas.

A continuación se describirá con detalle el diseño de la base de datos, es decir, la estructura interna de colecciones y subcolecciones, y sus campos:

- **Citas**. Representa las citas veterinarias que tienen los usuarios con su veterinario. El id de los documentos de esta colección es generado automáticamente por la base de datos Firestore. Sus campos son:
  - coste: número
  - diagnostico: cadena de texto
  - duracion: número
  - estado: cadena de texto
  - fecha: marca de tiempo (timestamp)
  - fechaModificacion: marca de tiempo (timestamp)
  - fechaSolicitud: marca de tiempo (timestamp)
  - motivo: cadena de texto
  - observaciones: cadena de texto

- tipoCita: cadena de texto
- tratamiento: cadena de texto
- mascotaId: cadena de texto
- usuarioId: cadena de texto
- veterinarioId: cadena de texto

- **Usuarios**. Aunque Firebase Authentication ya gestiona los usuarios, se ha optado por añadir esta colección para almacenar sus datos más alguna información adicional. El id de estos documentos es el correo electrónico con el que se registra el propio usuario. Sus campos son:

  - email: cadena de texto
  - esVeterinario: booleano
  - fcmToken: cadena de texto
  - fechaRegistro: cadena de texto
  - nombre: cadena de texto

- **Gastos**. Representa los gastos internos que van añadiendo los usuarios en la aplicación móvil. El id de los documentos de esta colección es generado automáticamente por la base de datos Firestore. Sus campos son:

  - cantidad: número
  - categoria: cadena de texto
  - descripcion: cadena de texto
  - fecha: marca de tiempo (timestamp)
  - frecuencia: cadena de texto
  - mascotaId: cadena de texto

- **Mascotas**. Esta colección representa las mascotas que crean los usuarios. El id de los documentos de esta colección es generado automáticamente por la base de datos Firestore. Sus campos son:

- especie: cadena de texto
- fechaNacimiento: marca de tiempo (timestamp)
- fechaRegistro: marca de tiempo (timestamp)
- fotoPerfil: cadena de texto
- nombre: cadena de texto
- peso: número
- raza: cadena de texto
- sexo: cadena de texto
- usuarioId: cadena de texto
- veterinarioId: cadena de texto

- **Notas**. Los documentos de esta colección son cada una de las notas que crean los usuarios. El id de los documentos de esta colección es generado automáticamente por la base de datos Firestore. Sus campos son:
  - categoria: cadena de texto
  - contenido: cadena de texto
  - fechaCreacion: marca de tiempo (timestamp)
  - fechaModificacion: marca de tiempo (timestamp)
  - mascotaId: cadena de texto
  - titulo: cadena de texto

- **Vacunas**. Representa las visitas veterinarias para vacunación de los animales. El id de los documentos de esta colección es generado automáticamente por la base de datos Firestore. Sus campos son:
  - coste: número
  - estado: cadena de texto
  - fechaAdministracion: marca de tiempo (timestamp)

- fechaModificacion: marca de tiempo (timestamp)
- fechaSolicitud: marca de tiempo (timestamp)
- mascotaId: cadena de texto
- retroalimentacion: cadena de texto
- tipoVacuna: cadena de texto
- usuarioId: cadena de texto
- veterinarioId: cadena de texto

- **Publicaciones**. Representa cada una de los recuerdos que suben los usuarios, esto es, la fotografía con título, y descripción. Su id es generado automáticamente por la base de datos Firestore. Sus campos son:
  - descripcion: cadena de texto
  - fecha: marca de tiempo (timestamp)
  - mascotaId: cadena de texto
  - titulo: cadena de texto
  - urlFoto: cadena de texto
  - usuarioId: cadena de texto

- **Paseos**. Cada documento de esta colección representa un paseo realizado por un usuario. La estructura del id de estos documentos es el email del usuario seguido de la fecha y hora de inicio del paseo, sin espacios ni separadores de ningún tipo. Los campos que se guardan para cada paseo son:
  - altitudMaxima: número
  - altitudMinima: número
  - distanciaRecorrida: número
  - duracion: cadena de texto
  - fecha: cadena de texto

- fechaInicio: cadena de texto
- gpsActivado: booleano
- latitudCentro: número
- latitudMaxima: número
- latitudMinima: número
- longitudCentro: número
- longitudMaxima: número
- longitudMinima: número
- mascotaId: cadena de texto
- numeroFotos: número
- ultimaImagen: cadena de texto
- usuario: cadena de texto

- **Localizaciones**. En la raíz de la base de datos, la colección localizaciones almacena la información de los recorridos registrados durante los paseos de las mascotas.

  Cada documento de esta colección tiene como id el correo electrónico del usuario del que se obtiene de la información.

  - **Subcolección**. Dentro de cada documento de la colección Localizaciones (cada uno perteneciente a un usuario único) se almacenan múltiples subcolecciones, una por cada paseo registrado de dicho usuario, cuyo id de estas subcolecciones corresponde a la fecha y hora de inicio del paseo en formato yyyyMMddHHmmss.

    En el interior de cada una de estas subcolecciones, se guardan múltiples documentos, donde cada documento representa un punto geográfico registrado en un momento concreto del recorrido. El ID de cada documento corresponde al segundo exacto del paseo en el que se registró la localización (por ejemplo: 0013, 0017, 0027, etc.). En este nivel, cada documento contiene los siguientes campos:

    - altitud: número
    - hora: cadena de texto

- latitud: número
- longitud: número
- tieneAltitud: booleano

Este diseño permite almacenar de forma jerárquica y cronológica todas las posiciones registradas durante los paseos, facilitando su consulta y reconstrucción en tiempo real o en visualizaciones posteriores del recorrido.

## 4.4. Aplicación Móvil

### 4.4.1. Diseño del Servidor

- **Firebase Firestore Database**

  Firestore es el sistema de base de datos NoSQL empleado para almacenar y gestionar la información central del sistema completo. Se trata de una base de datos en la nube, orientada a documentos y con capacidad de sincronización casi en tiempo real, lo que favorece la escalabilidad y la integración entre distintas plataformas.

  Su acceso se ha realizado tanto desde ambos entornos del sistema, permitiendo gestionar de forma eficiente la información asociada a los diferentes módulos funcionales. Gracias a su modelo flexible y sin esquemas rígidos, ha sido posible representar entidades diversas como citas veterinarias, mascotas, usuarios o gastos, adaptándose fácilmente a los requisitos específicos del dominio.

  Además, Firestore ha permitido realizar consultas personalizadas con soporte para filtros, ordenación y paginación, cubriendo así las necesidades funcionales de presentación y navegación de datos. Su integración fluida con otros servicios del ecosistema Firebase ha contribuido a un desarrollo ágil y bien cohesionado entre cliente, servidor y almacenamiento.

  (Firebase, 2025d)

- **Firebase Authentication**

  Firebase Authentication ha sido la solución utilizada para gestionar el sistema de **autenticación de usuarios** tanto en la aplicación móvil como en la aplicación web. Este

servicio permite implementar flujos de inicio de sesión seguros y escalables, integrándose fácilmente con el resto de servicios de Firebase.

Se han habilitado dos métodos de acceso principales de autenticación: mediante correo electrónico y contraseña, y a través de cuentas de Google. Esto proporciona a los usuarios flexibilidad a la hora de acceder a la plataforma, manteniendo una experiencia de uso sencilla y coherente en ambos entornos. En cualquier momento el desarrollador puede habilitar/deshabilitar un proveedor de acceso, o agregar uno nuevo entre la gran variedad que ofrece Firebase, como se puede observar en la Figura 17.

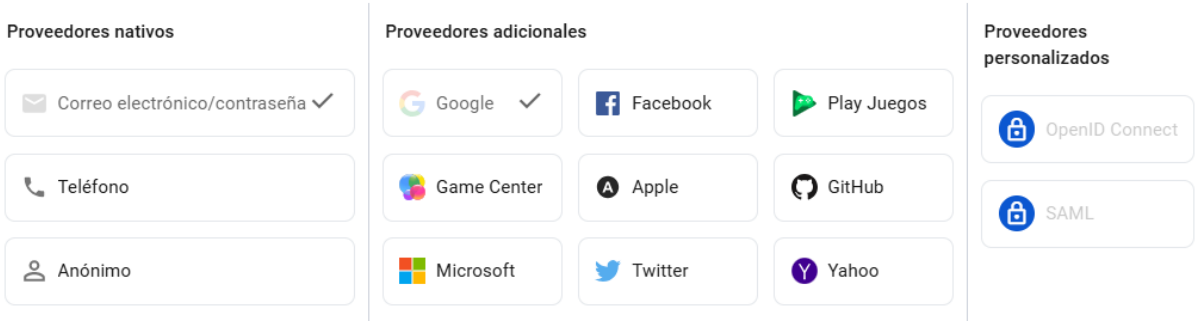



Figura 17: Proveedores de acceso que ofrece Firebase Authentication.

Fuente: https://console.firebase.google.com/

La autenticación no distingue entre distintos roles (veterinario o dueño). No obstante, esta información se recoge y almacena en el proceso de registro dentro de la base de datos, lo que permite posteriormente adaptar la experiencia de usuario según su perfil.

En materia de seguridad, el desarrollador no gestiona directamente las contraseñas de los usuarios ni son almacenadas en la base de datos.

(Firebase, 2025c)

- **Firebase Cloud Storage**

  Firebase Cloud Storage ha sido el servicio elegido que almacena y gestiona los **archivos multimedia** dentro del sistema. Esta herramienta en la nube permite almacenar imágenes y otros archivos de manera segura, escalable y con acceso sencillo desde las diferentes partes de la aplicación (Firebase, 2025b).

Cloud Storage se utiliza para guardar las fotos de perfil tanto de mascotas como de usuarios, las imágenes tomadas durante los paseos, y las publicaciones que los usuarios suben desde sus dispositivos móviles. La organización de los archivos sigue una estructura clara y ordenada en carpetas, facilitando su acceso y gestión:

- fotos_perfil/mascotas/mascotaId/ para las imágenes de perfil de las mascotas.
- fotos_perfil/usuarios/emailUsuario/ para las fotos de perfil de los usuarios.
- images/emailUsuario/idPaseo/ donde se almacenan las fotos capturadas durante cada paseo.
- publicaciones/emailUsuario/idMascota/ para las imágenes asociadas a las publicaciones.

El acceso a estas imágenes se realiza desde Android, donde se integran las librerías oficiales de Firebase para la subida, descarga y eliminación de archivos. En los documentos de la base de datos se guarda el enlace a las fotos almacenadas en Storage mediante un campo específico.

- **Firebase Cloud Messaging**

  *Firebase Cloud Messaging* (FCM) es una solución de mensajería multiplataforma que te permite enviar mensajes de forma segura (Firebaese, 2025). Con esta opción que ofrece Firebase, se ha podido implementar las **notificaciones** en el proyecto que se lanzan cuando un usuario realiza una acción. Concretamente, cuando un veterinario rechaza, confirma o cambia la fecha de una cita, así como cuando rellena los datos de un evento que ya se ha producido (coste, tratamiento, diagnóstico). En estos casos, con el token de FCM del usuario (que se almacena en su documento de la base de datos) se envía la notificación correspondiente al dueño de la mascota de la cita que ha cambiado de estado, o cuyos datos ya se encuentran disponibles.

- **Firebase Cloud Functions**

  También se ha hecho uso de *Cloud Functions*. Se trata de un framework sin servidores que permite ejecutar automáticamente código de backend (Firebase, 2025a). En el caso de este proyecto los eventos en segundo plano ejecutándose se corresponden con **Cloud**

**Scheduler**. Mediante código TypeScript almacenado en Google Cloud, las funciones implementadas con planificador se ejecutan cada día a las 9:00 a.m., hora central europea (CET), las cuales comprueban si entre todas las citas y vacunas confirmadas hay alguna agendada para el día siguiente, y envía una notificación automáticamente como recordatorio a los usuarios propietarios de mascotas que tengan cita o vacuna al día siguiente. Estas funciones se ejecutan en un entorno administrado, sin necesidad de administrar ni escalar un servidor propio.

### 4.4.2. Diseño del Cliente

En este contexto, entendemos por cliente como la parte de la arquitectura que se ejecuta en el dispositivo móvil del usuario. Esta es responsable de la interacción con la interfaz de usuario y la comunicación con el servidor (Firebase Storage, base de datos Firestore, etc.).

- **Arquitectura interna**

  En la implementación del cliente móvil se ha optado por una arquitectura basada en *Activities* y *Fragments*, siguiendo un enfoque próximo al patrón *MVC* (Modelo-Vista-Controlador, Model-View-Controller en inglés) en su adaptación clásica en Android. En este modelo encontramos dos partes diferenciadas:

  - **Activities y Fragments**, que cumplen la función de Vista (gestión de los layouts *XML* y actualización de la interfaz gráfica) y de Controlador (gestión de eventos de usuario, invocación a servicios y acceso a la base de datos).
  - **Modelo**, formado por las clases de datos que representan las distintas entidades de la aplicación (mascotas, citas, gastos, etc).

### 4.4.3. Interfaz de Usuario

La interfaz de usuario es un aspecto fundamental para la satisfacción de los usuarios con el producto software desarrollado. Su aspecto y usabilidad son clave para que el usuario se sienta cómodo con la aplicación, y pretenda seguir usándola o no. El diseño de la interfaz de usuario de la aplicación Android se ha llevado a cabo teniendo en cuenta diversos factores como la temática de la aplicación o su público objetivo. En concreto, se ha buscado que sea intuitiva, fácil de usar, accesible y coherente con la temática de cuidado de mascotas.

La integración en la aplicación de los distintos elementos de la interfaz se ha realizado siguiendo las guías de estilo de *Material Design*, lo que asegura compatibilidad con los estándares actuales de diseño en Android.

- **Tipografía**

  El tipo de fuente constituye un elemento esencial en el diseño de la interfaz de usuario, puesto que contribuye tanto a la legibilidad de los contenidos como a la estética de la aplicación. En el caso de la aplicación móvil desarrollada, se ha optado por la familia tipográfica **Geologica** en sus distintas variantes (Regular, Bold y Thin) con el fin de establecer una identidad visual y moderna.

  Se trata de un tipo de letra de fácil legibilidad debido a su espaciado y amplitud (Fonts, 2025). Sus diferentes tonos y estilos como la negrita o la delgada permite jerarquizar la información claramente, diferenciando títulos, subtítulos o anotaciones sin necesidad de recurrir a cambios excesivos de color. Su estilo geométrico y equilibrado transmite sensación de sencillez, precisión y claridad, lo que proporciona una estética moderna y profesional para un sistema así que gestiona tanto información médica como de uso personal para el usuario. La utilización de los distintos tipos de variantes de Geologica, además del tamaño de fuente y su color, han sido claves para garantizar una experiencia de usuario intuitiva, evitando la sobrecarga visual y facilitando la rápida identificación de los distintos niveles de información en las pantallas.

- **Iconos**

  Los iconos han sido añadidos a las diferentes partes de la interfaz de usuario de manera que facilite la comprensión de los diferentes campos, acciones o elementos disponibles y reduzca la carga cognitiva del usuario al interactuar con la aplicación.

  La biblioteca de iconos utilizada ha sido *Material Design Icons* (Pictogrammers, 2025). haciendo uso de la opción para cada icono de descargarlo en el formato XML para Android (véase Figura 18), lo cual ha permitido que su integración y utilización en el proyecto haya sido sencilla y directa.

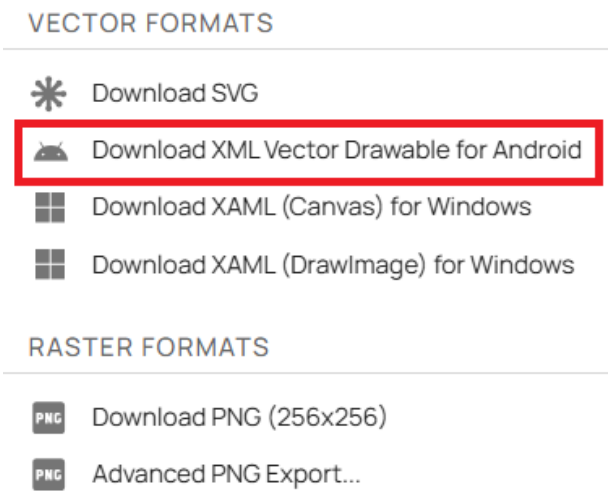



Figura 18: Descargar un icono como vector XML para Android.

Fuente: https://pictogrammers.com/library/mdi

Los iconos han seguido también una consistencia visual en cada una de las pantallas, de manera que mantienen un estilo y color similares entre ellos. Además, algunos se han combinado con texto para ayudar la comprensión e identificación del elemento al que hacen referencia, como se puede observar en el ejemplo de la Figura 19.

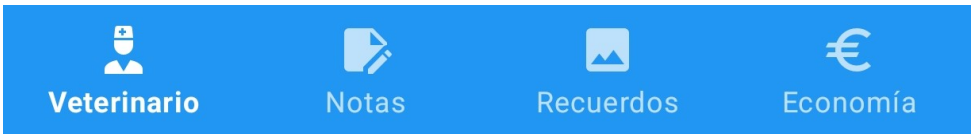



Figura 19: Barra inferior de navegación con iconos y texto.

Fuente: elaboración propia.

- **Paleta de colores**

  La elección y definición de una paleta cromática propia resulta esencial ya que influye directamente en la percepción estética y proporciona una identidad visual clara a la aplicación.

  La paleta de colores diseñada contiene los siguientes colores principales:

  - **Azul principal (#2196F3)**: color corporativo y base de la interfaz. Se emplea en la

barra de navegación, botones primarios y elementos de acción destacados, transmitiendo confianza, profesionalidad y cercanía.

- **Verde principal (#81C784)**: es el segundo color más importante después del azul principal. Asociado al bienestar y la naturaleza, se utiliza también en botones de acciones principales de la aplicación, como creación o guardado de datos.
- **Amarillo principal (#FFE082)**: aporta calidez y visibilidad sin ser intrusivo. Se emplea para ciertos detalles y, fundamentalmente, como color de fondo de las tarjetas de citas veterinarias debido a su asociación con la atencion y precaución, cualidades necesarias en la gestión de información sanitaria. Este recurso visual llama la atención de manera positiva, favoreciendo la usabilidad y seguridad al recordar al propietario de la mascota aspectos críticos como próximas citas o vacunas.
- **Lila (#C4B5FD)**: se reserva para pequeños detalles, aportanto un matiz diferenciador y reforzando la estética moderna de la aplicación.

A su vez, los colores escogidos para la tipografía y separadores de elementos han sido:

- **Text Primary (#212121)**: usada para el texto principal, con un contraste suficiente sobre los fondos claros.
- **Text Secondary (#757575)**: aplicado a textos secundarios, etiquetas y descripciones.
- **Divider (#BDBDBD)**: se ha utilizado para líneas divisoras de elementos y bordes.
- **Fondo (#F9FAFB):**: se trata del color de fondo de las pantallas de la aplicación. Un color claro y neutro que refuerza la legibilidad y contraste de los elementos sobrepuestos.

Por último se han escogido otros colores para adecuarlos mejor a su funcionalidad específica en la aplicación, como el tono de rojo #D32F2F para botones de Eliminar o Cancelar, el verde #4CAF50 para el color del icono de inicio de los paseos o el marrón #795548 para reflejar el estado pendiente de confirmación de una cita o vacuna.

## 4.5. Aplicación Web

### 4.5.1. Diseño del Servidor

Dentro de la arquitectura de web de la aplicación, el servidor (backend) actúa como un intermediario entre el cliente (frontend) y la base de datos. En el backend se ha implementado una API REST que gestiona la lógica de negocio y centraliza la comunicación entre la parte frontal que está en contacto directamente con el usuario y la base de datos, garantizando la seguridad, validación y consistencia necesarias en las operaciones. Se podría decir que el servidor es la capa intermedia que aporta seguridad en cuanto a aplicar validaciones antes de crear, leer, actualizar o eliminar documentos en las colecciones de Firestore.

El formato de los datos empleado para el envío y recepción de la información entre el servidor tanto con la base de datos como con el cliente Angular ha sido **JSON**. Este es muy sencillo de entender y de utilizar, además de que el uso de una base de datos noSQL orientada a documentos hace que JSON sea ideal para este proyecto por su flexibilidad y compatibilidad.

- **Organización del código**

  Como se puede observar en la Figura 20, esta parte del código de la aplicación se compone de tres carpetas. La primera corresponde al proyecto con las funciones implementadas para Firebase Cloud Functions. La segunda, node_modules, contiene todos los módulos instalados que han sido necesarios para implementar las distintas funcionalidades, y una tercera denominada src que es la principal. Esta carpeta contiene los archivos fundamentales para el correcto funcionamiento de la aplicación. Estos son:

  - app.js, que contiene la configuración principal de Express y la definición de las rutas principales.
  - firebase.js, que se encarga de configurar Firebase.
  - index.js, que configura el puerto de ejecución del servidor. En este caso, se trata del puerto 3000.

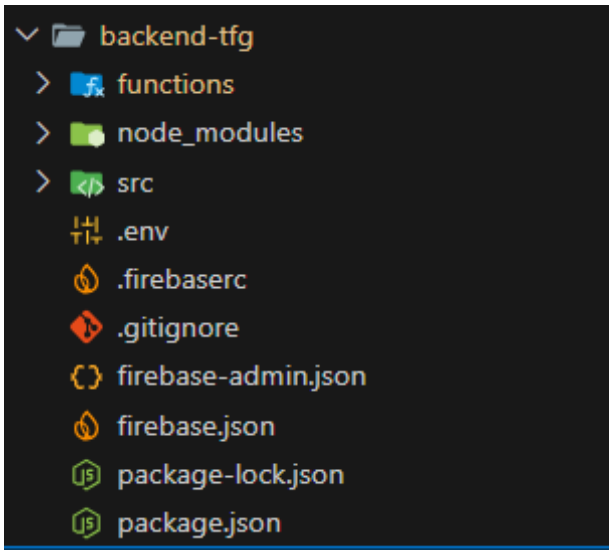



Figura 20: Estructura de carpetas y archivos del servidor.

Fuente: elaboración propia.

Por otro lado, la carpeta src divide el código correspondiente a la API en tres subcarpetas:

- routes: define las rutas concretas para cada una de las operaciones de cada endpoint.
- controllers: realiza validacioness de parámetros de entrada y gestiona las respuestas. Actúa como intermediario entre las llamadas http de las rutas y los modelos que contienen la lógica de negocio.
- models: responsable de acceder directamente a la base de datos y realizar las operaciones necesarias.

Con esto se pretende mantener una separación de responsabilidades en el código, además de facilitar y mejorar su mantenibilidad y escalabilidad.

- **Estructura de la API**

  La API del backend de la aplicación web se ha diseñado siguiendo el estilo **RESTful**, lo que implica que cada recurso se representa mediante una URL única, y las operaciones sobre estos recursos se realizan mediante los métodos HTTP estándar (GET, POST, PUT, DELETE). Los recursos principales de la aplicación son: mascotas, usuarios, citas

y vacunas. La elección de este tipo de API responde a la necesidad de mantener una arquitectura clara, coherente y fácilmente mantenible, así como de garantizar interoperabilidad entre distintos clientes que puedan consumir la API, como la aplicación web en Angular y posibles futuras integraciones. Además, es ideal por su simplicidad, escalabilidad y por el uso de JSON para el intercambio de datos.

En el archivo app.js se definen las rutas principales, correspondiendo cada una a un recurso específico de la API. Todas las rutas relacionadas con la API se encuentran precedidas por el prefijo /api/, lo que permite diferenciarlas claramente de las rutas utilizadas para la navegación de la aplicación cliente. Por ejemplo, las rutas asociadas a la gestión de usuarios se exponen como http://localhost:3000/api/usuarios.

### 4.5.2. Diseño del Cliente

La arquitectura del cliente de la aplicación web consiste en un modelo *Single Page Application* (SPA) con Angular. Como su nombre indica, se trata de una aplicación de una sóla página que es la encargada de interactuar con el usuario sin necesidad de cargar páginas nuevas, sino que es el contenido el que se muestra y se actualiza en una única página. Esto permite una mayor fluidez en la navegación y la recarga dinámica de los datos sin refrescar la página, lo cual se traduce a su vez en una mayor rapidez y mejor experiencia de usuario respecto a otro tipo de aplicaciones. Además, al cargar las páginas solo una vez, las webs SPA tienen un menor consumo de ancho de banda (omatech, 2025).

- **Estructura de la aplicación Angular**

  En cuanto a la división del código en diferentes carpetas en la parte del cliente destaca la carpeta src, encargada de organizar en diferentes archivos y subcarpetas el código principal de la aplicación. En esta carpeta se ecuentra la subcarpeta principal app, que modulariza el código en dos subcarpetas: components y services.

  El diseño de la parte frontal de la aplicación se centra en el uso de componentes reutilizables, que se encuentran dentro de la carpeta components, el uso de servicios (carpeta services) para la comunicación del frontend con el backend y en el enrutamiento o *routing* para gestionar la navegación del usuario entre las diferentes vistas.

- **Comunicación cliente-servidor**

  Tal y como se ha mencionado, la aplicación Angular consume la API REST a través de los servicios creados. Estos servicios se utilizan en los componentes, y funcionan de manera que cuando se requiere una comunicación con un endpoint del servidor, el componente hace uso del servicio correspondiente y, a través del servicio HttpClient, envía peticiones haciendo uso de un método http al servidor con la url correspondiente de la API. Los componentes utilizan los resultados devueltos por el servidor para mostrar los resultados o gestionar la información según la funcionalidad.

# 5 Implementación

Este apartado está dedicado a describir la etapa de implementación del sistema, es decir, cómo se ha desarrollado técnicamente cada una de las partes. Se mencionarán las librerías y servicios usados, así como detalles de la implementación enfocados en la seguridad o en la experiencia del usuario. Estará dividido en dos bloques, uno para la aplicación móvil y otro para la aplicación web.

## 5.1. Aplicación Móvil

### 5.1.1. Librerías y Dependencias

**Firebase Authentication**

Se ha utilizado para la implementación de la gestión de usuarios en la aplicación, un servicio que permite integrar de forma sencilla el registro e inicio de sesión mediante correo electrónico y contraseña, así como a través de cuentas de Google. Además, ofrece la funcionalidad de recuperación de contraseñas, lo que facilita al usuario restablecer su acceso en caso de olvido o modificación de la misma.

La integración se ha realizado mediante la dependencia de la propia librería:

implementation("com.google.firebase:firebase-auth")

Esta librería proporciona los métodos necesarios para la gestión del ciclo completo de autenticación de usuarios (registro, inicio de sesión y restablecimiento de contraseñas). En el flujo principal de autenticación de la aplicación se han desarrollado las siguientes actividades:

- **LoginActivity**: permite al usuario introducir sus credenciales, iniciar sesión con Google o navegar a las pantallas de registro y recuperación de contraseña. Por motivos de seguridad, cuando las credenciales introducidas no son correctas se muestra un mensaje

genérico (*"El email o la contraseña no son correctos"*) en lugar de detallar qué campo ha fallado, evitando dar información sensible a posibles atacantes.

Para la implementación del inicio de sesión con cuentas de Google se ha utilizado la dependencia com.google.android.gms:play-services-auth:21.0.0 de Google Play Services.

- **RegisterActivity**: gestiona el registro de nuevos usuarios en la aplicación. Se validan los datos introducidos (por ejemplo, el formato del correo electrónico y la longitud mínima de la contraseña) y, en caso de ser correctos, se crea la cuenta en Firebase Authentication. Adicionalmente, se registra un documento asociado en la colección usuarios de la base de datos, con información extra como el nombre, si es veterinario o no y su token para notificaciones.

- **ForgotPasswordActivity**: permite al usuario recuperar su acceso introduciendo la dirección de correo electrónico asociada a su cuenta. Mediante el método sendPasswordResetEmail(email) se envía automáticamente un mensaje de correo electrónico con el enlace para restablecer la contraseña.

Por otra parte, la **persistencia de la sesión** del usuario autenticado se ha implementado en el método del ciclo de vida onStart() de la actividad de inicio de sesión. Este método se ejecuta cada vez que la actividad pasa a estar visible para el usuario. En él se obtiene el usuario actualmente autenticado mediante FirebaseAuth.getInstance().currentUser. Si el resultado no es nulo, significa que el usuario ya tiene una sesión activa y, en consecuencia, se le redirige directamente a la pantalla principal (Home) sin necesidad de volver a introducir sus credenciales, mejorando así la experiencia de uso Figura 21.

```
public override fun onStart() {  Juaansanchez
    super.onStart()

    val currentUSer = FirebaseAuth.getInstance().currentUser
    if (currentUSer != null) {
        goHome(currentUSer.email.toString(), currentUSer.providerId)
    }
}
```

Figura 21: Método que valida la persistencia de sesión.

Fuente: elaboración propia.

**Firebase Firestore y Firebase Storage**

La gestión de la información de las mascotas y de los archivos asociados se ha implementado utilizando dos de los servicios principales de Firebase:

- **Firebase Firestore** (com.google.firebase:firebase-firestore).

  Esta librería ha servido para almacenar, actualizar, eliminar, hacer consultas y obtener los documentos de las distintas colecciones de la base de datos: mascotas, citas, vacunas, notas, etc. Por ejemplo, en cuanto a la gestión de las citas y vacunas veterinarias, cada vez que un usuario solicita una nueva cita o vacuna a su veterinario, se crea un documento en la colección correspondiente.

  Para la visualización de los distintos elementos (citas, vacunas, notas, paseos...), la aplicación obtiene los documentos asociados a la mascota seleccionada por el usuario en la pantalla principal. Estos se muestran en un RecyclerView, que permite visualizar dinámicamente estos recursos según los usuarios vayan eliminando, editando o añadiendo.

- **Firebase Storage** (com.google.firebase:firebase-storage)

  Esta dependencia ha sido fundamental para el almacenamiento de imágenes de la aplicación en este servicio del servidor en la nube. Ha permitido gestionar las fotografías de perfil tanto de los usuarios como de las mascotas, además de las imágenes subidas en la pantalla de recuerdos o las tomadas por los usuarios durante los paseos.

Por último, cabe destacar que a la hora de la eliminación de un perfil de una mascota, no se ha implementado simplemente la eliminación del documento de la mascota en su colección, sino que antes se procede a la eliminación de todas las imágenes relacionadas con esta en Storage (paseos que ha realizado, foto de perfil y publicaciones), así como de todos los documentos vinculados en Firestore (citas, vacunas, notas, etc). De este modo, se garantiza que no queden datos huérfanos ni archivos innecesarios, es decir, de una mascota que ya no esté registrada en la aplicación, preservando la coherencia y optimizando el espacio en la base de datos y el almacenamiento en la nube.

**Material Design Components**

La librería Material Design Components (for Developers, 2025e) (com.google.android.material:material) proporciona elementos de interfaz gráfica modernos, que enriquecen la experiencia de usuario y facilitan la implementación de interacciones consistentes y atractivas.

En la implementación de las distintas pantallas e interfaces de usuario de la aplicación, se han utilizado principalmente:

- **FloatingActionButton:** es un botón circular que se mantiene fijo en la interfaz y se superpone al resto de elementos de la vista. Su propósito es destacar una acción principal o prioritaria dentro del contexto de la pantalla, garantizando que el usuario pueda acceder a ella de manera rápida e intuitiva.

  En la aplicación se ha utilizado para facilitar la creación de nuevos recursos (notas, recuerdos o gastos), así como para permitir la solicitud inmediata de citas y vacunas veterinarias.

- **BottomSheetDialogFragment:** muestra información detallada proporcionada por el veterinario sin necesidad de abandonar el fragmento actual, optimizando la navegación y la experiencia de usuario.

- **MaterialCardView:** se emplea en la pantalla de estadísticas financieras y de paseos para presentar las tarjetas visuales de manera atractiva y consistente con la estética general de la aplicación.

El uso de esta librería permite que la interfaz de usuario sea coherente, moderna y accesible, mejorando la interacción y la percepción de calidad de la aplicación.

**RecyclerView**

La librería RecyclerView (for Developers, 2025f) (androidx.recyclerview:recyclerview) permite mostrar listas de elementos de forma eficiente, con soporte para vistas dinámicas y optimización de memoria mediante recliclado de vistas.

En la aplicación, el componente RecyclerView se utiliza para:

- Mostrar las notas, publicaciones y gastos asociados a cada mascota.

- Presentar el historial de paseos y citas veterinarias de manera dinámica.

- Implementar adaptadores que gestionan la visualización de cada elemento y permiten actualizaciones en tiempo real cuando se añaden, modifican o eliminan documentos de la base de datos.

El uso de RecyclerView contribuye a la fluidez de la interfaz y a la escalabilidad de la aplicación, facilitando la visualización de grandes volúmenes de información sin afectar al rendimiento.

**AndroidX AppCompat** (for Developers, 2025a)

Esta librería permite mantener compatibilidad hacia atrás con versiones anteriores de Android, proporcionando componentes de interfaz y funcionalidades modernas independientemente de la versión del sistema operativo del dispositivo.

En la aplicación móvil, AppCompat se utiliza como base para todas las Activities y Fragments, asegurando que elementos de UI como la barra de herramientas, los temas y estilos, los botones y menús funcionen correctamente en la mayor variedad de dispositivos posible.

En el apartado de las notas y de la economía se ha hecho uso del componente SearchView, incluido en esta librería, para implementar la barra de búsqueda en sus archivos layouts.

Esto permite al usuario filtrar las notas por título o categoría, facilitando su búsqueda cuando hay un gran número de elementos añadidos en la aplicación y la lista es extensa.

Con el mismo objetivo, en la gestión de los gastos se implementaron filtros más avanzados (por categoría, frecuencia, precio mínimo y precio máximo), que actualizan el contenido mostrado en el RecyclerView tras pulsar el botón de aplicar filtros.

En la Figura 22 podemos observar la implementación de la función que filtra los gastos, cambiando la consulta a la base de datos en función de los campos introducidos por el usuario y actualizándose la lista del RecyclerView y los elementos de la interfaz dependiendo de si se han obtenido resultados.


Figura 22: Función para filtrar los gastos.

Fuente: elaboración propia.

**MPAndroidChart**

La librería externa MPAndroidChart (Jahoda, 2025) (com.github.PhilJay:MPAndroidChart:v3.1.0) ha sido utilizada para la implementación de las gráficas de estadísticas financieras y de paseos.

En la pantalla de estadísticas de la economía, las gráficas muestran a los usuarios datos como:

- Gasto total en cada mes, pudiendo filtrar por todas las categorías o por una concreta (gráfico de líneas).
- Gasto total por categoría, pudiendo filtrar por fecha indicando mes y año concreto o todo un año completo (gráfico de barras horizontales).

Esta librería también ha facilitado la creación de gráficos en la pantalla de estadísticas de los paseos, en concreto:

- Distancia recorrida en metros por meses en el año actual (gráfica de barras verticales).
- Duración total de los paseos en minutos por meses en el año actual (gráfica de barras verticales).
- Duración promedio de los paseos en minutos por franja horaria (mañana, tarde y noche), siendo este un gráfico de barras horizontales.
- Cantidad de paseos realizados por semana del mes (pie chart, gráfico circular en inglés).

En la Figura 23 se presenta la función privada mostrarDatosEnGrafica(datos: Map<String, Float>) de la clase ExpenseStatsFragment.kt, cuya finalidad es configurar el componente gráfico de barras horizontales de MPAndroidChart para representar las categorías de gasto junto con sus valores correspondientes.

```
private fun mostrarDatosEnGrafica(datos: Map<String, Float>) {  Juaansanchez
    val entries = datos.entries.mapIndexed { index, entry ->
        BarEntry(index.toFloat(), entry.value)
    }

    val dataSet = BarDataSet(entries,  label: "Gastos")
    dataSet.color = ContextCompat.getColor(requireContext(), R.color.azul_principal)
    dataSet.valueTextSize = 12f

    val data = BarData(dataSet)
    data.barWidth = 0.8f

    horizontalBarChartExpensesByCategory.data = data

    // Configuramos etiquetas X con categorías
    horizontalBarChartExpensesByCategory.xAxis.valueFormatter = IndexAxisValueFormatter(datos.keys.toList())
    horizontalBarChartExpensesByCategory.xAxis.labelCount = datos.size

    horizontalBarChartExpensesByCategory.invalidate() // refrescar
}
```

Figura 23: Función para representar los gastos en una gráfica.

Fuente: elaboración propia.

**Google Maps SDK para Android**

Para la visualización y rastreo en tiempo real de los paseos de las mascotas, se ha incluido la librería Maps SDK para Android (for Developers, 2025d) (com.google.android.gms:play-services-maps:19.0.0) para integrar un mapa de Google directamente en la aplicación.

La aplicación implementa funciones como centrar la vista del mapa en la ubicación del usuario en tiempo real o pintar líneas que representan el recorrido realizado.

El uso de esta librería se complementa con la gestión de permisos de GPS mediante la implementacion de funciones privadas como allPermissionsGrantedGPS(), initPermissionsGPS() y requestPermissionsLocation(), que aseguran que la aplicación pueda acceder a la ubicación del dispositivo únicamente con el consentimiento del usuario. En caso de que los permisos estén desactivados, se redirige al usuario a la pantalla de ajustes de ubicación o se muestra un cuadro de diálogo solicitando permisos.

Respecto a los paseos, esta actividad principal gestiona la recopilación de localizaciones, el cálculo de la distancia recorrida en metros, la gestión del cronómetro y la persistencia de los datos del paseo en la base de datos, incluyendo la eliminación coherente de documentos asociados (imágenes, localizaciones) para mantener la consistencia de la información.

Para el cálculo de la distancia recorrida en los paseos (Figura 24) se emplea la **fórmula de Haversine** (Wikipedia, 2025), que calcula la distancia entre dos puntos de una esfera a partir de sus latitudes y longitudes. En la aplicación se traduce en la distancia entre dos coordenadas consecutivas del usuario.

Un detalle incluido en la implementación de esta función es que, con el fin de evitar sumar desplazamientos que no se corresponden con un paseo real (por ejemplo, en coche), se ha definido un criterio basado en la velocidad máxima de caminata aceptada.

La aplicación registra la localización cada 8 segundos (constante INTERVAL_LOCATION). A partir de este intervalo se ha fijado un límite máximo de distancia entre dos puntos consecutivos:

LIMIT_DISTANCE_ACCEPTED_WALKING = 12 * INTERVAL_LOCATION

Esto equivale a una velocidad de 12 m/s. Si la distancia entre dos puntos excede ese valor, se interpreta que el desplazamiento no se ha producido caminando y, por tanto, no se suma al total recorrido. Se ha establecido este valor considerablemente elevado como margen de seguridad frente a posibles errores de precisión del GPS.

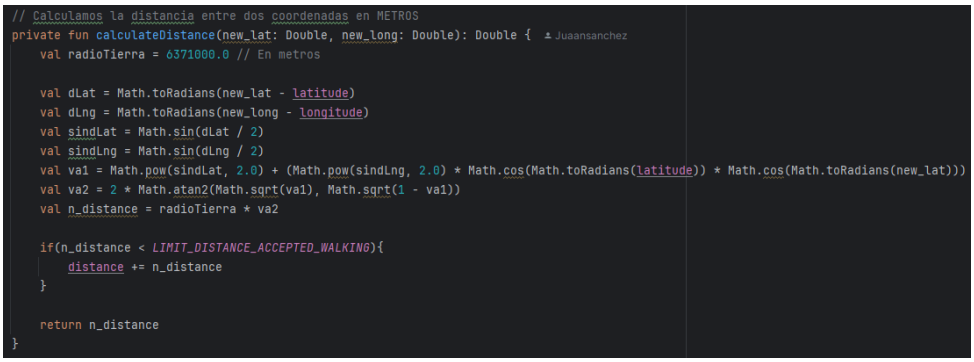



Figura 24: Función para calcular la distancia recorrida entre dos puntos.

Fuente: elaboración propia.

**CameraX**

Para la captura de fotografías durante los paseos, la aplicación utiliza la librería CameraX (Jetpack) (for Developers, 2025b) con las dependencias:

implementation("androidx.camera:camera-core:1.0.1")

implementation("androidx.camera:camera-camera2:1.0.1")

Esta librería simplifica la integración de la cámara en Android, gestionando automáticamente el ciclo de vida, la previsualización y la captura de imágenes en distintos dispositivos.

En la pantalla CameraActivity, CameraX se emplea para:

- Tomar fotografías durante los paseos y almacenarlas directamente en Firebase Storage.
- Cambiar entre cámara trasera y delantera según las capacidades del dispositivo.
- Gestionar permisos de cámara, garantizando el acceso seguro y controlado.

La integración de CameraX permite que la funcionalidad de fotografía sea robusta, compatible con la mayoría de dispositivos Android y totalmente integrada con la gestión de paseos y almacenamiento en la nube de la aplicación.

**Glide**

Para la carga y visualización eficiente de imágenes en la aplicación móvil se ha utilizado la librería Glide (Judd, 2025) (com.github.bumptech.glide:glide:4.12.0). Esta librería permite descargar, procesar y mostrar imágenes de forma optimizada, gestionando la memoria y el

almacenamiento en caché automáticamente, lo que garantiza un rendimiento fluido incluso al trabajar con numerosas imágenes. En la aplicación, Glide se emplea para mostrar los archivos de imagen de la aplicación (fotografías tomadas en los paseos y fotos de perfil de los usuarios y mascotas).

Durante el rastreo de paseos, cada recorrido puede incluir múltiples fotografías. La función privada loadImagesFromStorage() obtiene las URLs de las imágenes almacenadas en Firebase Storage y, mediante Glide, las carga dinámicamente en un contenedor de tipo LinearLayout, mostrando las imágenes correctamente escaladas y adaptadas al diseño de la interfaz.

Glide también ha permitido mostrar de forma instantánea la foto de perfil del usuario y de las mascotas. La aplicación actualiza la URL de la imagen en Firestore y, a continuación, Glide carga la foto en un ImageView aplicando transformaciones visuales como circle crop.

La Figura 25 muestra un ejemplo donde se carga la imagen de perfil del usuario (si existe) en un ImageView de forma circular usando Glide. La URL de la imagen se obtiene del campo fotoPerfil del documento del usuario en Firestore.

```
// Obtener datos de Firestore
firestore.collection( collectionPath: "usuarios").document(userEmail).get()
    .addOnSuccessListener { document ->
        if (document.exists()) {
            val userName = document.getString( field: "nombre")
            textViewUserName.text = userName

            // Obtener URL de la foto de perfil si existe
            val fotoPerfilUrl = document.getString( field: "fotoPerfil")
            if (!fotoPerfilUrl.isNullOrEmpty()) {
                buttonSeleccionarImagen.text = ...
                Glide.with(requireContext())
                    .load(fotoPerfilUrl)
                    .circleCrop()
                    .into(imageViewProfilePic)
```

Figura 25: Función para cargar una imagen en la aplicación.

Fuente: elaboración propia.

El uso de Glide asegura que la aplicación maneje de manera eficiente tanto la visualización de múltiples imágenes como la actualización en tiempo real de fotos de perfil, contribuyendo a una experiencia de usuario fluida y profesional.

## 5.2. Aplicación web

### Angular Material

La librería @angular/material (Angular, 2025a) proporciona un completo conjunto de componentes de interfaz gráfica basados en las guías de diseño de Google Material Design. En esta aplicación se ha empleado para implementar tablas, formularios, menús, cuadros de diálogo, mensajes informativos, botones y navegaciones laterales, garantizando una experiencia de usuario coherente, accesible y moderna. Además de facilitar la consistencia visual, el uso de estos componentes permitió acelerar el desarrollo y mantener buenas prácticas de usabilidad.

Un caso en el que se ha utilizado esta librería se encuentra en el componente HomeComponent, que constituye la interfaz principal de la aplicación web. En él se emplean diversos módulos de Angular Material para estructurar tanto el menú lateral como la barra de herramientas superior. Concretamente, se ha utilizado MatSidenav y MatList para implementar un menú lateral de navegación con accesos directos a las secciones de agenda, clientes, citas y vacunas pendientes, así como un submenú desplegable de estadísticas mediante MatExpansionPanel.

Por otro lado, la barra superior se ha implementado con MatToolbar, incorporando un botón de menú basado en MatIcon y un botón de cierre de sesión con MatButton. La acción de cierre de sesión abre un cuadro de diálogo de confirmación implementado con MatDialog.

De esta manera, Angular Material facilita no solo la coherencia visual de la interfaz, sino también la organización y accesibilidad de las principales funcionalidades del sistema.

### Angular Fire

La librería @angular/fire (Google, 2025) es la capa oficial de integración entre Angular y los servicios de Firebase, ofreciendo una API adaptada al ecosistema de Angular. En esta aplicación se ha utilizado principalmente para la gestión de la autenticación de usuarios mediante Firebase Authentication. Gracias a esta integración, ha sido posible implementar de manera sencilla y segura funcionalidades como el registro de nuevos usuarios, el inicio de sesión tanto con Google como con correo electrónico, y el envío de correos electrónicos automáticos para la recuperación o cambio de contraseña.

Además, la librería proporciona acceso directo a los datos del usuario autenticado (por

ejemplo, su correo electrónico), lo que permite inyectar dicha información en los distintos componentes de la aplicación y utilizarla en las peticiones a la API, garantizando que las operaciones realizadas correspondan siempre al veterinario autenticado.

**RxJS**

La librería RxJS (Angular, 2025b) constituye el motor de programación reactiva en Angular y proporciona un conjunto de operadores y estructuras para la gestión eficiente de flujos de datos asíncronos. Su integración es esencial en esta aplicación, ya que permite suscribirse a los cambios provenientes de la API REST y reaccionar automáticamente cuando se reciben nuevas respuestas o se actualiza el estado de la aplicación.

Un ejemplo concreto de su uso se encuentra en el servicio de citas y en el de vacunas, donde se utiliza un objeto de tipo BehaviorSubject para mantener un flujo reactivo de vacunas pendientes. Este patrón permite que cualquier componente suscrito reciba actualizaciones en tiempo real cada vez que se modifica el listado, sin necesidad de recargar manualmente los datos.

Gracias a este enfoque, la interfaz se mantiene consistente y reactiva: al actualizarse una vacuna en el servidor, los cambios se propagan automáticamente a todos los componentes suscritos, garantizando una experiencia de usuario fluida y sin recargas manuales.

**Angular Forms**

Los módulos @angular/forms y @angular/reactive-forms se han utilizado para la construcción de formularios interactivos, dinámicos y validados en la aplicación web.

En particular, ReactiveFormsModule ha permitido definir formularios basados en un modelo reactivo en el que cada campo se representa mediante un FormControl. Esto proporciona un control total sobre el estado y la validez de cada entrada, así como la posibilidad de aplicar validaciones síncronas y asíncronas de forma declarativa.

Ejemplos concretos incluyen los formularios de registro, inicio de sesión y cambio de contraseña, donde se aplican reglas de validación de campos como correo electrónico, contraseñas seguras y coincidencia de contraseñas.

Además, la naturaleza reactiva de estos formularios permite actualizar dinámicamente la interfaz de usuario en función del estado del formulario: mostrar mensajes de error en tiempo real, habilitar o deshabilitar botones de envío, o incluso aplicar estilos visuales diferenciados

según la validez de los campos.

**Angular Router** (Angular, 2025c)

Es el módulo responsable de gestionar la navegación interna de la aplicación web, permitiendo la organización de vistas y componentes mediante rutas definidas y jerarquizadas. Su uso ha sido fundamental para garantizar una experiencia fluida, estructurada y segura.

En esta aplicación, la navegación se ha organizado en tres niveles principales:

- **Rutas públicas para la autenticación**: gestionadas en AuthRoutingModule, incluyen las vistas de inicio de sesión, registro y recuperación de contraseña. Estas rutas están agrupadas bajo el path /auth y permiten la carga perezosa (*lazy loading*) de los módulos relacionados con la autenticación, optimizando la eficiencia de la aplicación.

- **Rutas protegidas**: definidas en DashboardRoutingModule, estas rutas están protegidas mediante un guard implementado con canActivate de Angular Fire Auth Guard. Solo los usuarios autenticados pueden acceder a estas secciones, que incluyen: calendario, gestión de clientes, citas y vacunas, y estadísticas de mascotas y citas. La ruta raíz del dashboard carga el HomeComponent, que actúa como contenedor de las vistas hijas, permitiendo una navegación jerárquica consistente.

- **Resolución de datos previa a la carga de componentes**: mediante el uso de UserResolver, se obtiene la información del usuario autenticado antes de cargar los componentes dependientes de estos datos. Esto garantiza que los componentes dispongan de toda la información necesaria al inicializarse, evitando llamadas adicionales a la API y posibles inconsistencias en la interfaz.

Además, Angular Router permite cerrar sesión y redirigir al usuario a la pantalla de login de forma controlada mediante Router.navigate.

**Full Calendar**

La librería FullCalendar (FullCalendar, 2025) se ha utilizado para implementar la agenda del veterinario, mostrando las citas y vacunas confirmadas. Esta herramienta permite al veterinario visualizar de forma clara y organizada los eventos, así como interactuar con ellos,

pudiendo reprogramarlas modificando sus fechas y completando datos relevantes de cada cita o vacuna.

En la aplicación, FullCalendar se integra mediante el módulo @fullcalendar/angular y los plugins dayGrid, timeGrid e interaction, que proporcionan vistas mensuales y semanales y la interacción con los eventos. Los eventos se cargan desde la API REST a través de los servicios de Angular.

Se han implementado funcionalidades específicas para mejorar la experiencia del usuario:

- Diferenciación de eventos mediante colores según el tipo (cita o vacuna).
- Modales emergentes para mostrar detalles completos de cada evento al hacer clic.
- Formularios reactivos para completar información de citas y vacunas pasadas, con validación de campos.
- Reprogramación de eventos también mediante un diálogo modal.

El código de la Figura 26 muestra la configuración de las diferentes opciones que proporcionan los calendarios de FullCalendar. Se establecen campos como la vista inicial del calendario (initialView), la disposición de los elementos de la cabecera del calendario (headerToolbar) o la duración de los eventos por defecto (defaultTimedEventDuration). Otros como events se asignan posteriormente, en concreto dentro del método ngOnInit(), mediante una llamada a la API del servidor.

```
calendarOptions: CalendarOptions = {
  plugins: [dayGridPlugin, interactionPlugin, timeGridPlugin],
  initialView: 'dayGridMonth',
  aspectRatio: 1.2,
  height: 'auto',
  contentHeight: 'auto',
  expandRows: true,
  locale: esLocale,
  headerToolbar: {
    left: 'prev,next today',
    center: 'title',
    right: 'dayGridMonth,timeGridWeek,timeGridDay'
  },
  defaultTimedEventDuration: '00:30:00',
  eventTimeFormat: { hour: '2-digit', minute: '2-digit', hour12: false },
  events: [],
  dateClick: undefined,
  eventDidMount: (info) => {
    if (info.event.extendedProps['tipo'] === "vacuna") {
      info.el.style.color = "#ca50ff";
    } else if (info.event.extendedProps['tipo'] === "cita") {
      info.el.style.color = "#33adff";
    }
  },
  eventClick: this.mostrarModalEvento.bind(this)
};
```

Figura 26: Configuración del calendario de eventos.

Fuente: elaboración propia.

De esta manera, FullCalendar se convierte en un componente central de la aplicación web, combinando visualización, interacción y sincronización en tiempo real con la información del sistema.

### Apex Charts

La librería ng-apexcharts (ApexCharts, 2025) se ha empleado para la representación gráfica de datos estadísticos dentro del panel de control de la aplicación web. En las pantallas de estadísticas del veterinario se han implementado diferentes tipos de gráficas (líneas, barras, donut, pie, heatmap, entre otras), adaptadas a la naturaleza de cada conjunto de datos.

Estas gráficas muestran información clave como el número de citas por tipo y por mes, ingresos mensuales, mascotas más frecuentes, nuevos clientes, porcentaje de ocupación semanal, promedio de duración de cada tipo de cita o el ranking de mascotas con más visitas. Los datos que alimentan los gráficos se obtienen dinámicamente mediante llamadas al backend, a través de los servicios de Angular, garantizando la sincronización con la información almacenada en la base de datos.

Asimismo, las gráficas se han configurado con etiquetas y *tooltips* interactivos que facilitan

la interpretación visual de los datos al pasar el puntero sobre cada elemento. También se han implementado filtros dinámicos (por fecha, categoría, especie...) en las gráficas, permitiendo realizar comparaciones ofreciendo una visión detallada y flexible. Todo ello está organizado en tres secciones principales accesibles desde el menú lateral: Resumen, Citas y Vacunas y Mascotas, lo que proporciona al veterinario un panel estadístico claro, estructurado e interactivo.

La Figura 27 muestra un ejemplo de mapa de calor (heatmap) generado en la aplicación web mediante ApexCharts. Este mapa representa la distribución de citas y vacunas por franjas horarias para un veterinario. Cada celda indica la cantidad de citas en un intervalo horario durante cada día de la semana y se ha utilizado una escala con 4 colores posibles para representar el número de citas y vacunas en cada celda: gris (sin citas), amarillo (de 1 a 6), narajna (de 7 a 12) y rojo (13 o más). Los datos se obtienen del servidor a través de la API correspondiente y se asignan dinámicamente al gráfico en el método ngOnInit() del componente.

```
if (this.user?.email) {
  this.citasService.obtenerHeatmapHorarioVeterinario(this.user.email).subscribe(data => {
    this.chartOptions = {
      series: data,
      chart: {
        height: 350,
        width: '100%',
        type: 'heatmap'
      },
      plotOptions: {
        heatmap: {
          shadeIntensity: 0.6,
          radius: 4,
          useFillColorAsStroke: true,
          colorScale: {
            ranges: [
              { from: 0, to: 0, color: '#e0e0e0', name: 'Sin citas' },
              { from: 1, to: 6, color: '#ffff00', name: 'Bajo' },
              { from: 7, to: 12, color: '#f0ac00', name: 'Medio' },
              { from: 13, to: 99999, color: '#ff0000', name: 'Alto' }
            ]
          }
        }
      },
      dataLabels: {
        enabled: true,
        style: { colors: ['#000'] }
      },
      title: {
        text: 'Mapa de calor - Citas y vacunas por franja horaria',
        align: 'center'
      },
      tooltip: {
        enabled: true,
        y: { formatter: val => `${val} citas` }
      },
      xaxis: {
        categories: this.horas
      }
    };
  });
}
```

Figura 27: Configuración del mapa de calor de citas y vacunas por día y hora.

Fuente: elaboración propia.

**Firebase y Firebase-admin**

Los módulos de firebase para node, firebase y firebase-admin (for Developers, 2025c), han permitido integrar fácilmente el servidor al proyecto Firebase que contiene los diferentes servicios. En este caso, el servidor no accede directamente con consultas SQL a la base de datos, sino que emplea los métodos que el SDK de Firebase Admin facilita para ello.

# 6

# Pruebas y Validación

Una vez concluida la fase de implementación se ha llevado a cabo de la validación del sistema con el objetivo de garantizar que la solución cumple con los requisitos definidos y funciona como se esperaba. Para ello se han realizado principalmente pruebas funcionales sobre la aplicación móvil y la aplicación web, verificando que cada una de las funcionalidades implementadas responde correctamente al ser utilizada por un usuario final. Estas pruebas se han centrado en los aspectos y casos de prueba definidos en la fase de análisis.

Adicionalmente, se han ejecutado pruebas de la API REST mediante la herramienta Postman, con el propósito de comprobar la correcta comunicación entre el servidor y el cliente web, así como la validez de las operaciones sobre la base de datos. Estas pruebas han permitido asegurar que los distintos endpoints responden de manera adecuada, gestionando correctamente peticiones, datos y posibles errores.

Estas pruebas tienen como objetivo identificar errores y poder corregirlos. Sin embargo, como en todo producto software, esto no demuestra la ausencia total de errores en el sistema.

## 6.1. Pruebas Funcionales

| **Identificador** | ERR-01 |
|---|---|
| **Descripción** | Al seleccionar el campo desplegable del sexo de la mascota no se muestra ninguna opción seleccionable |
| **Explicación** | El elemento Spinner de este campo no está configurado correctamente. El adaptador (ArrayAdapter) está configurado dentro de la función createNewUserPet, por lo que la actividad al abrirse todavía no lo reconoce |
| **Corrección** | Configurar el Spinner dentro del método onCreate para que las opciones del desplegable estén disponibles desde el inicio. |

Tabla 49: ERR-01 - Añadir mascota.

| **Identificador** | ERR-02 |
|---|---|
| **Descripción** | Al abrir la pantalla de Citas, no se abre el historial para visualizar las citas |
| **Explicación** | El id de la mascota no se le está enviando al fragmento encargado de mostrar las citas de la mascota |
| **Corrección** | Cargar el fragmento creando una instancia de este donde se le envíe el id de la mascota como parámetro |

Tabla 50: ERR-02 - Visualizar historial de citas.

| | |
|---|---|
| **Identificador** | ERR-03 |
| **Descripción** | Al eliminar la única nota que hay añadida, la aplicación no muestra el mensaje informando de que no hay notas. |
| **Explicación** | Cuando se elimina la última nota añadida en la aplicación, no se gestiona la visibilidad del mensaje |
| **Corrección** | Después de eliminar una nota hay que añadir la comprobación de si la lista ha quedado vacía, y en ese caso actualizar la visibilidad del texto en consecuencia. En este caso, el adaptador del RecyclerView tiene que informar al fragmento mediante un callback, y el fragmento maneja el callback haciendo visible el mensaje correspondiente. |

Tabla 51: ERR-03 - Mostrar mensaje.

| | |
|---|---|
| **Identificador** | ERR-04 |
| **Descripción** | Al realizar un usuario un paseo demasiado corto, la app se cierra y deja de funcionar. |
| **Explicación** | El paseo realizado tiene una duración excesivamente corta, por lo que durante ese tiempo a la aplicación no le ha dado tiempo de registrar ninguna ubicación del usuario correctamente, y los atributos como minLatitude y maxLatitude se quedan con valor null y la aplicación no puede calcular los valores centrales de logitud y latitud del paseo. |
| **Corrección** | Manejar los casos en los que estas variables son nulas y mostrar un diálogo informando de que el paseo fue demasiado corto. |

Tabla 52: ERR-04 - Realizar paseo.

| **Identificador** | ERR-05 |
|---|---|
| **Descripción** | Al editar los datos de una mascota, si el usuario deja vacía la fecha de nacimiento de la mascota, la aplicación falla y deja de funcionar. |
| **Explicación** | Esto pasa porque en el código se está haciendo la conversión del valor de cadena de texto a fecha para guardarlo en su campo en la base de datos, lo cual si esa cadena de texto es null (no se ha introducido ningún valor), la aplicación falla porque no se puede convertir un valor nulo a una fecha. |
| **Corrección** | Añadir comprobación de que, si no se ha introducido fecha de nacimiento, asignar valor null a ese valor para guardarlo así en la base de datos en lugar de intentar convertirlo a fecha. |

Tabla 53: ERR-05 - Editar mascota.

| **Identificador** | ERR-06 |
|---|---|
| **Descripción** | Cuando se selecciona una nueva foto de perfil para cambiarla, la vista de la interfaz en la aplicación no se actualiza con la nueva foto sino que se sigue visualizando la antigua, a pesar de que se muestra el mensaje toast de que la foto de perfil se ha actualizado. |
| **Explicación** | Esto ocurre porque hay un problema de sincronización entre la caché de Glide y el guardado de la url de la imagen en Firestore, ya que Glide está cargando la url antes de que ese campo se actualice en la base de datos. |
| **Corrección** | En la función guardarUrlEnFirestore, hacer que Glide cargue la url y la muestre en la pantalla sólo cuando ya se ha completado el guardado del nuevo valor de la url en la base de datos y no antes, es decir, dentro del bloque addOnSuccessListener. |

Tabla 54: ERR-06 - Editar mi perfil.

| **Identificador** | ERR-07 |
|---|---|
| **Descripción** | Cuando hay muchas notas añadidas, el botón de crear nota queda arrastrado por ellas hacia abajo de manera que desaparece de la vista de la pantalla y no hay manera de poner utilizarlo. |
| **Explicación** | El error radica en que ese botón en el layout xml estaba dispuesto dentro de un LinearLayout, con lo cual su posición depende directamente de los demás elementos de ese LinearLayout. |
| **Corrección** | Se saca ese Floating Action Button fuera del LinearLayout pero se mantiene dentro del FrameLayout. |

Tabla 55: ERR-07 - Crear nota o ver notas.

| **Identificador** | ERR-08 |
|---|---|
| **Descripción** | La aplicación permite crear un gasto sin haber introducido un dato para el campo fecha, que es obligatorio. |
| **Explicación** | Esto ocurre porque en la función donde se crea el gasto no hay ninguna validacion de este campo obligatorio. |
| **Corrección** | Inttroducir en el código la correspondiente validación de este campo, en la que se comprueba que si es null (no se ha seleccionado ninguna fecha), mediante un return dentro de este bloque del código se detiene la ejecución de la función y no se crea el gasto. |

Tabla 56: ERR-08 - Crear gasto.

| **Identificador** | ERR-09 |
|---|---|
| **Descripción** | La aplicación no permite editar los datos de un gasto por un error de tipo: Could not deserialize object. |
| **Explicación** | Este error se produce porque se estaba intentando guardar el campo cantidad del gasto como un String pero el campo cantidad en la base de datos lo espera como un number. |
| **Corrección** | Antes de guardar el documento correspondiente en su colección, es necesario convertir el valor de la cantidad a Double con la función de extensión de la clase String: toDoubleOrNull(). |

Tabla 57: ERR-09 - Editar gasto.

| **Identificador** | ERR-10 |
|---|---|
| **Descripción** | Al abrir la pantalla de Mi Perfil, se superpone la vista del Main Activity con los elementos del fragmento de Mi Perfil. |
| **Explicación** | Al abrir el fragmento, este simplemente se superpone sobre la vista que estaba abierta, en este caso en Main Activity. |
| **Corrección** | Se oculta la visibilidad del layout del Main cuando se abra el fragmento de Mi Perfil para que sólo se muestren los elementos de ese layout. Al volver al Main, se hace visible de nuevo su layout. |

Tabla 58: ERR-10 - Visualizar Mi Perfil.

| | |
|---|---|
| **Identificador** | ERR-11 |
| **Descripción** | En cada card que representa un paseo, el nombre de la mascota con la que se ha realizado ese paseo no se obtiene correctamente. |
| **Explicación** | La causa de este fallo reside en que las consultas a Firebase con asíncronas, pero la funcion encargada de buscar en la base de datos el nombre de la mascota dado su id intenta devolver el resultado antes de que Firebase devuelva el resultado de la consulta. |
| **Corrección** | La solución está en usar un callback, de manera que la función que obtiene el nombre de la mascota recibe un callback como parámetro y, cuando Firebase responde con el resultado llama al callback para que actualizar el TextView con el nombre de la mascota. |

Tabla 59: ERR-11 - Ver paseos.

## 6.2. Pruebas Postman

Para validar el correcto funcionamiento de la API REST desarrollada en Node.js y Express, se han realizado pruebas con la herramienta Postman. Estas pruebas permiten verificar tanto los códigos de estado HTTP devueltos por los endpoints como la estructura y el contenido de las respuestas JSON, garantizando que el backend cumple con los requisitos funcionales definidos y que dichas operaciones son válidas para la implementación de estos requisitos en la aplicación. A través de esta herramienta se podrán realizar peticiones HTTP para los endpoints de la API, comprobando la correcta comunicación entre el servidor y la base de datos, así como las respuestas JSON y si realiza una gestión adecuada de los errores.

A continuación se incluyen los casos de prueba diseñados, que prueban los principales endpoints de cada entidad:

- **Usuarios**

| **Identificador** | CP-01 |
|---|---|
| **Método** | GET |
| **Endpoint** | /api/usuarios/clientes/:veterinarioId |
| **Descripción** | Obtener los clientes de un veterinario |
| **Datos de entrada** | idVeterinario |
| **Respuesta esperada** | 20o OK + JSON con datos de los clientes |

Tabla 60: CP-01 - Obtener clientes.

| **Identificador** | CP-02 |
|---|---|
| **Método** | POST |
| **Endpoint** | /api/usuarios/enviarNotificacion |
| **Descripción** | Enviar notificación a un usuario |
| **Datos de entrada** | email, title, message |
| **Respuesta esperada** | 200 OK + notificación enviada |

Tabla 61: CP-02 - Enviar notificación.

| **Identificador** | CP-03 |
|---|---|
| **Método** | GET |
| **Endpoint** | /api/usuarios/veterinarios/:veterinarioId/resumen |
| **Descripción** | Obtener un resumen de estadísticas para el veterinario |
| **Datos de entrada** | idVeterinario |
| **Respuesta esperada** | 200 OK + JSON con datos de las estadísticas |

Tabla 62: CP-03 - Obtener resumen de estadísticas.

| **Identificador** | CP-04 |
|---|---|
| **Método** | GET |
| **Endpoint** | /api/usuarios/veterinarios/:veterinarioId/porcentajeOcupacion |
| **Descripción** | Obtener el porcentaje de ocupación en la semana actual |
| **Datos de entrada** | idVeterinario |
| **Respuesta esperada** | 200 OK + JSON con el porcentaje de opcupación |

Tabla 63: CP-04 - Obtener porcentaje ocupación.

| **Identificador** | CP-05 |
|---|---|
| **Método** | GET |
| **Endpoint** | /api/usuarios/veterinarios/nuevos-clientes/:veterinarioId |
| **Descripción** | Obtener los clientes nuevos de un veterinario en el mes actual |
| **Datos de entrada** | idVeterinario |
| **Respuesta esperada** | 200 OK + JSON con los datos de los nuevos clientes |

Tabla 64: CP-05 - Obtener nuevos clientes.

- **Citas**

| **Identificador** | CP-06 |
|---|---|
| **Método** | GET |
| **Endpoint** | /api/citas/veterinario/:veterinarioId/pendientes |
| **Descripción** | Obtener las solicitudes de citas pendientes por confirmar de un veterinario |
| **Datos de entrada** | idVeterinario |
| **Respuesta esperada** | 200 OK + JSON con los datos de las citas pendientes |

Tabla 65: CP-06 - Obtener citas pendientes.

| **Identificador** | CP-07 |
|---|---|
| **Método** | GET |
| **Endpoint** | /api/citas/:citaId |
| **Descripción** | Obtener una cita |
| **Datos de entrada** | idCita |
| **Respuesta esperada** | 200 OK + JSON con los datos de la cita |

Tabla 66: CP-07 - Obtener cita.

| **Identificador** | CP-08 |
|---|---|
| **Método** | PUT |
| **Endpoint** | /api/citas/ |
| **Descripción** | Actualizar los datos de una cita |
| **Datos de entrada** | JSON con los datos de la cita |
| **Respuesta esperada** | 200 OK + JSON con los datos de la cita actualizada |

Tabla 67: CP-08 - Actualizar cita.

| **Identificador** | CP-09 |
|---|---|
| **Método** | GET |
| **Endpoint** | /api/citas/veterinario/horas-ocupadas/:veterinarioId/:fecha |
| **Descripción** | Obtener las horas ocupadas por citas y vacunas en una fecha determinada |
| **Datos de entrada** | idVeterinario, fecha |
| **Respuesta esperada** | 200 OK + JSON con las horas no disponibles |

Tabla 68: CP-09 - Obtener horas no disponibles.

| **Identificador** | CP-10 |
|---|---|
| **Método** | GET |
| **Endpoint** | /api/citas/veterinario/citasCalendario/:veterinarioId |
| **Descripción** | Obtener las citas y vacunas confirmadas de un veterinario |
| **Datos de entrada** | idVeterinario |
| **Respuesta esperada** | 200 OK + JSON con los datos de las citas y vacunas junto con otros datos útiles para su visualización en el calendario |

Tabla 69: CP-10 - Obtener citas y vacunas para calendario.

| **Identificador** | CP-11 |
|---|---|
| **Método** | GET |
| **Endpoint** | /api/citas/estadisticas/por-mes/:veterinarioId |
| **Descripción** | Obtener el número de citas confirmadas de un veterinario en cada mes |
| **Datos de entrada** | idVeterinario |
| **Respuesta esperada** | 200 OK + JSON con el número de citas confirmadas por mes |

Tabla 70: CP-11 - Obtener número de citas por mes.

| **Identificador** | CP-12 |
|---|---|
| **Método** | GET |
| **Endpoint** | /api/citas/estadisticas/ingresos-por-mes/:veterinarioId |
| **Descripción** | Obtener el total de ingresos en citas y vacunas por mes de un veterinario |
| **Datos de entrada** | idVeterinario |
| **Respuesta esperada** | 200 OK + JSON con los ingresos por mes del veterinario |

Tabla 71: CP-12 - Obtener ingresos por mes.

| **Identificador** | CP-13 |
|---|---|
| **Método** | GET |
| **Endpoint** | /api/citas/estadisticas/mascotas-con-mas-citas/:veterinarioId |
| **Descripción** | Obtener las cinco mascotas con más citas de un veterinario |
| **Datos de entrada** | idVeterinario |
| **Respuesta esperada** | 200 OK + JSON con el nombre de las mascotas junto con el número de citas que han tenido con su veterinario |

Tabla 72: CP-13 - Obtener mascotas con más citas.

| **Identificador** | CP-14 |
|---|---|
| **Método** | GET |
| **Endpoint** | /api/citas/estadisticas/tiempo-promedio-confirmacion/:veterinarioId |
| **Descripción** | Obtener la media de tiempo que tarda un veterinario en confirmar las solicitudes de citas pendientes |
| **Datos de entrada** | idVeterinario |
| **Respuesta esperada** | 200 OK + JSON con los datos del promedio de tiempo de confirmación de la semana actual, de la semana anterior, la variación entre ambos tiempos y si ha mejorado, empeorado o sigue igual |

Tabla 73: CP-14. Obtener tiempo promedio de confirmación de citas.

| **Identificador** | CP-15 |
|---|---|
| **Método** | GET |
| **Endpoint** | /api/citas/estadisticas/promedio-duracion-tipo-cita/:veterinarioId |
| **Descripción** | Obtener la duración media de cada cita de un veterinario |
| **Datos de entrada** | idVeterinario |
| **Respuesta esperada** | 200 OK + JSON con las citas por su tipo y el valor de su duración media |

Tabla 74: CP-15 - Obtener duración promedio de citas por tipo.

| **Identificador** | CP-16 |
|---|---|
| **Método** | GET |
| **Endpoint** | /api/citas/estadisticas/cantidad-por-tipo-cita/:veterinarioId |
| **Descripción** | Obtener el número de citas de cada tipo que ha tenido un veterinario |
| **Datos de entrada** | idVeterinario |
| **Respuesta esperada** | 200 OK + JSON con cada tipo de cita junto a la cantidad correspondiente |

Tabla 75: CP-16 - Obtener número de citas por tipo.

| **Identificador** | CP-17 |
|---|---|
| **Método** | GET |
| **Endpoint** | /api/citas/estadisticas/heatmap-horario/:veterinarioId |
| **Descripción** | Obtener las citas y vacunas de un veterinario por franja horaria |
| **Datos de entrada** | idVeterinario |
| **Respuesta esperada** | 200 OK + JSON con el día de la semana, la hora y la cantidad correspondiente de citas y vacunas en esa franja |

Tabla 76: CP-17 - Obtener citas y vacunas por día de la semana y hora.

| **Identificador** | CP-18 |
|---|---|
| **Método** | GET |
| **Endpoint** | /api/citas/estadisticas/promedio-total-coste-tipo-cita/:veterinarioId |
| **Descripción** | Obtener el coste promedio y su total acumulado de todas citas de un veterinario según su tipo |
| **Datos de entrada** | idVeterinario |
| **Respuesta esperada** | 200 OK + JSON con los tipos de cita y su valor promedio y total acumulado del coste |

Tabla 77: CP-18 - Obtener coste promedio y total de citas por tipo.

- **Mascotas**

| **Identificador** | CP-19 |
|---|---|
| **Método** | GET |
| **Endpoint** | /api/mascotas/:id |
| **Descripción** | Obtener mascota por su identificador |
| **Datos de entrada** | Identificador de la mascota |
| **Respuesta esperada** | 200 OK + JSON con los datos de la mascota |

Tabla 78: CP-19 - Obtener mascota por id.

| **Identificador** | CP-20 |
|---|---|
| **Método** | GET |
| **Endpoint** | /api/mascotas/duenyo/:usuarioId |
| **Descripción** | Obtener las mascotas de un cliente |
| **Datos de entrada** | idUsuario |
| **Respuesta esperada** | 200 OK + JSON con los datos de las mascotas del usuario |

Tabla 79: CP-20 - Obtener mascotas de un usuario.

| **Identificador** | CP-21 |
|---|---|
| **Método** | GET |
| **Endpoint** | /api/mascotas/veterinario/mascotas-frecuentes/:veterinarioId?meses=:meses&filtroEspecie=:especie |
| **Descripción** | Obtener las mascotas que han visitado su veterinario una media de una o más veces al mes durante un determinado número de meses pasados, indicado como parámetro query. También se puede filtrar por especie, indicándola de igual manera en la url. |
| **Datos de entrada** | idVeterinario, número de meses y especie |
| **Respuesta esperada** | 200 OK + JSON con el nombre de las mascotas que superan el umbral de visitas y el total de visitas que han realizado en el período de tiempo indicado |

Tabla 80: CP-21 - Obtener mascotas frecuentes.

| **Identificador** | CP-22 |
|---|---|
| **Método** | GET |
| **Endpoint** | /api/mascotas/veterinario/mascotas-seguimiento/:veterinarioId |
| **Descripción** | Obtener las mascotas cuya última cita o vacuna que tuvo fue hace más de 1 año o nunca ha tenido, y además no tiene ninguna cita o vacuna programada |
| **Datos de entrada** | idVeterinario |
| **Respuesta esperada** | 200 OK + JSON con el nombre, especie y raza de las mascotas junto a la fecha de su última cita o vacuna (si la tuvo hace más de un año) |

Tabla 81: CP-22 - Obtener mascotas para seguimiento.

| **Identificador** | CP-23 |
|---|---|
| **Método** | GET |
| **Endpoint** | /api/mascotas/veterinario/mascotas-por-sexo/:veterinarioId |
| **Descripción** | Obtener el número de mascotas de cada sexo asignadas a un veterinario. |
| **Datos de entrada** | idVeterinario |
| **Respuesta esperada** | 200 OK + JSON con el número total de mascotas asignadas, el número de mascotas macho, el número de mascotas hembra y los porcentajes que representan respecto al total |

Tabla 82: CP-23 - Obtener mascotas por sexo.

| | |
|---|---|
| **Identificador** | CP-24 |
| **Método** | GET |
| **Endpoint** | /api/mascotas/veterinario/mascotas-por-especie/:veterinarioId |
| **Descripción** | Obtener el número de mascotas de cada especie asignadas a un veterinario. |
| **Datos de entrada** | idVeterinario |
| **Respuesta esperada** | 200 OK + JSON con el número total de mascotas asignadas junto con un objeto con el nombre de cada especie, el número de mascotas que atiende por cada especie y el porcentaje que representa respecto al total |

Tabla 83: CP-24 - Obtener mascotas por especie.

| | |
|---|---|
| **Identificador** | CP-25 |
| **Método** | GET |
| **Endpoint** | /api/mascotas/veterinario/especies/:veterinarioId |
| **Descripción** | Obtener las especies de los animales que están a cargo de un veterinario, es decir, los tipos de animales que atiende |
| **Datos de entrada** | idVeterinario |
| **Respuesta esperada** | 200 OK + JSON con un un array con las distintas especies asignadas a un veterinario |

Tabla 84: CP-25 - Obtener especies por veterinario.

- **Vacunas**

| **Identificador** | CP-26 |
|---|---|
| **Método** | GET |
| **Endpoint** | /api/vacunas/veterinario/:veterinarioId/pendientes |
| **Descripción** | Obtener las solicitudes de vacunas pendientes por confirmar de un veterinario |
| **Datos de entrada** | idVeterinario |
| **Respuesta esperada** | 200 OK + JSON con los datos de las vacunas pendientes |

Tabla 85: CP-26 - Obtener vacunas pendientes.

| **Identificador** | CP-27 |
|---|---|
| **Método** | GET |
| **Endpoint** | /api/vacunas/:vacunaId |
| **Descripción** | Obtener vacuna por su identificador |
| **Datos de entrada** | idVacuna |
| **Respuesta esperada** | 200 OK + JSON con los datos de la vacuna |

Tabla 86: CP-27 - Obtener vacuna por id.

| **Identificador** | CP-28 |
|---|---|
| **Método** | PUT |
| **Endpoint** | /api/vacunas/ |
| **Descripción** | Actualizar los datos de una vacuna |
| **Datos de entrada** | JSON con los datos de la vacuna |
| **Respuesta esperada** | 200 OK + JSON con los datos de la vacuna actualizada |

Tabla 87: CP-28 - Actualizar vacuna.

| | |
|---|---|
| **Identificador** | CP-29 |
| **Método** | GET |
| **Endpoint** | /api/vacunas/estadisticas/vacunas-por-mes/:veterinarioId |
| **Descripción** | Obtener el número de vacunas confirmadas de un veterinario en cada mes |
| **Datos de entrada** | idVeterinario |
| **Respuesta esperada** | 200 OK + JSON con el número de vacunas confirmadas por mes |

Tabla 88: CP-29 - Obtener número de vacunas por mes.

- **Ejemplos de ejecución.**

En la Figura 28 se muestra la prueba del endpoint GET /api/usuarios/clientes/{veterinarioId}. La API responde con el código de estado 200 OK junto con un objeto JSON que contiene un array de objetos con los datos de los clientes del veterinario, además de varias propiedades útiles para la implementación de la paginación en el cliente.

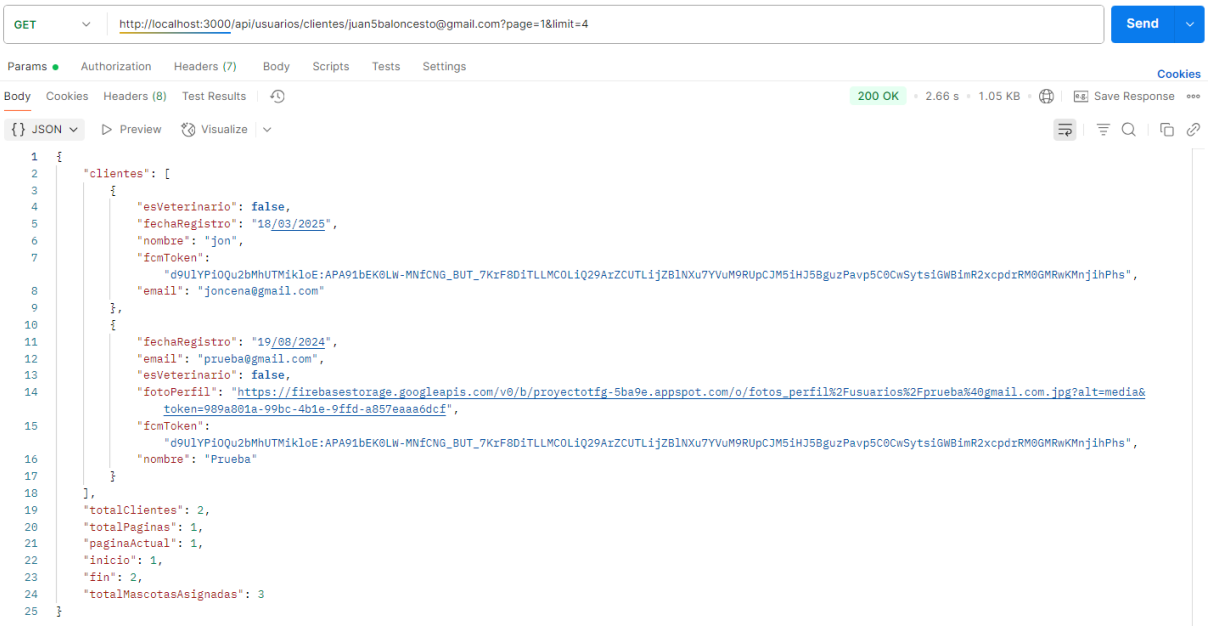



Figura 28: Ejecución de prueba de obtención de clientes.

Fuente: elaboración propia.

La siguiente ejecución muestra la prueba del enpoint de la operación http GET /api/usuarios/veterinarios/{veterinarioId}/resumen, que responde con el código 200 OK y el cuerpo de la respuesta en formato JSON con varios datos relacionados con el veterinario, como se puede observar en la Figura 29.

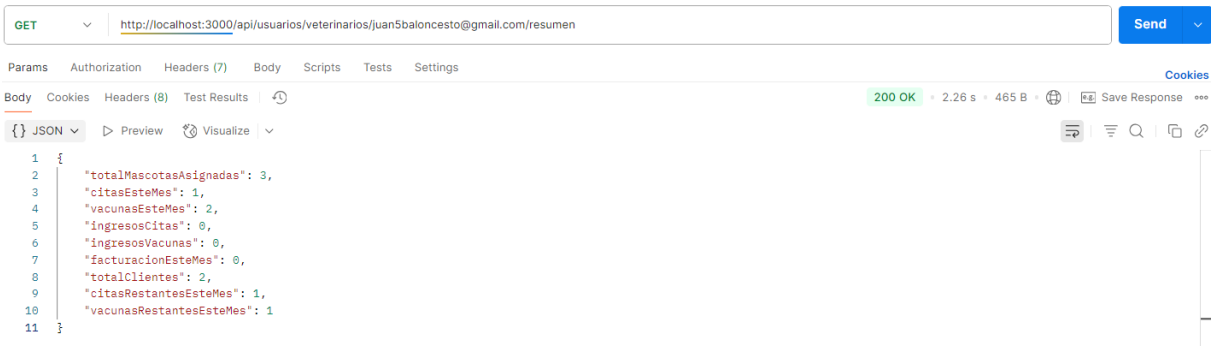



Figura 29: Ejecución de prueba de obtención de resumen de estadísticas.

Fuente: elaboración propia.

La Figura 30 refleja la ejecución del endpoint GET /api/citas/veterinario/{veterinarioId}/fechas-ocupadas, que devuelve las horas ocupadas ya por otras citas o vacunas en la fecha concreta pasada como parámetro en la url. Esta información se utiliza para prevenir que el veterinario no superponga dos citas/vacunas en la misma fecha y a la misma hora. Al abrir el selector de hora para agendar una cita/vacuna un día determinado, la aplicación deshabilita las horas ya ocupadas ese día de manera que no son seleccionables por el veterinario.

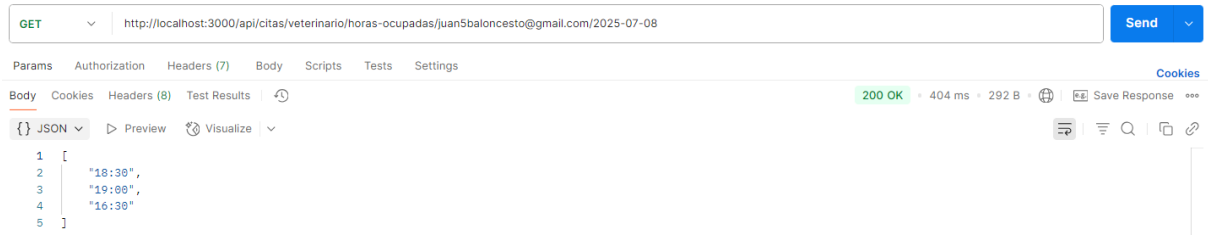



Figura 30: Ejecución de prueba de obtención de horas ocupadas de una fecha.

Fuente: elaboración propia.

A continuación, se incluye la prueba del endpoint GET /api/citas/estadisticas/ingresos-por-mes/{veterinarioId}. La respuesta JSON tiene la estructura que necesita el componente que implementa la gráfica que muestra estos datos en la aplicación e incluye el año y todos los meses junto con la cantidad de ingresos generados por citas y vacunas en cada uno de ellos.

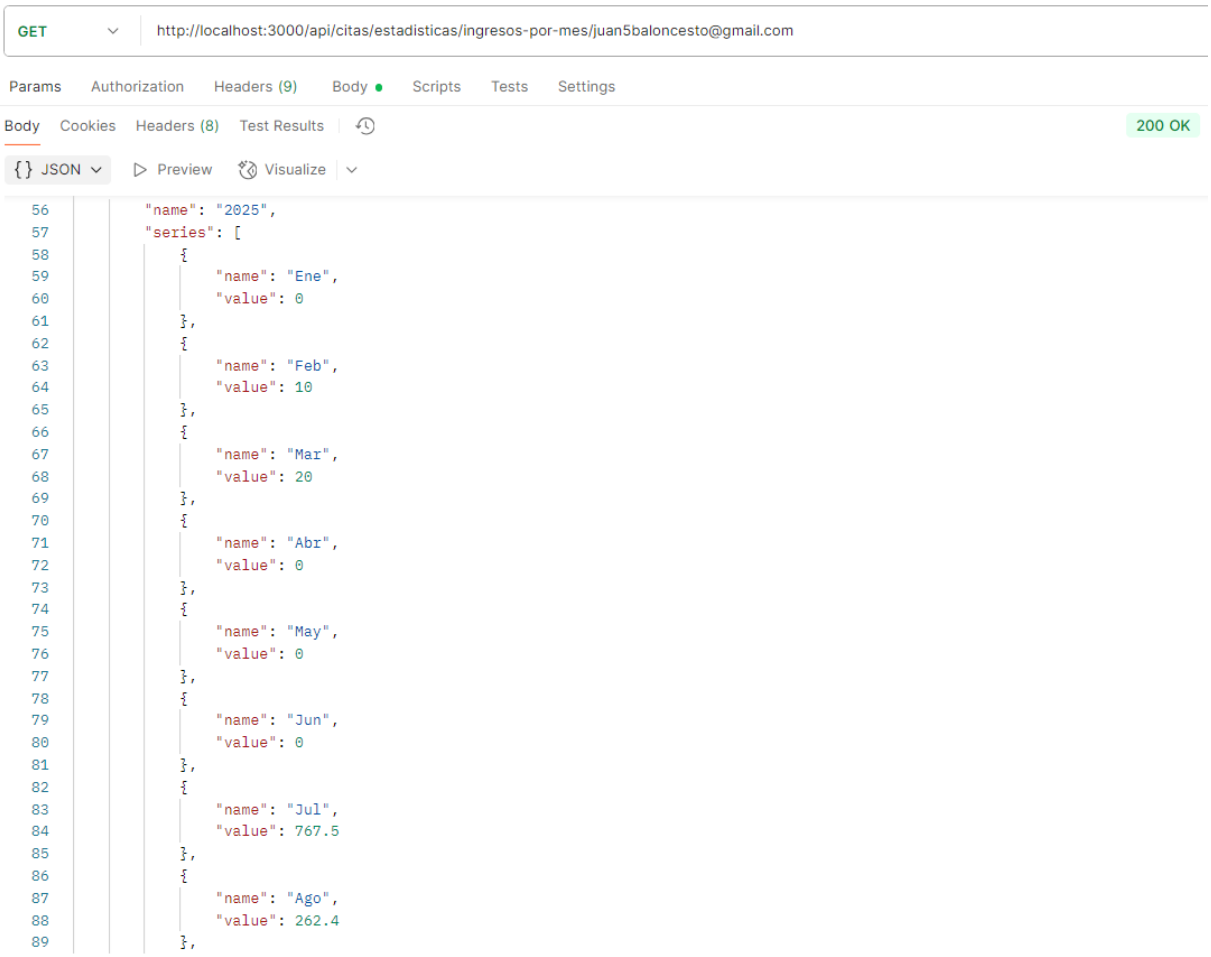



Figura 31: Ejecución de prueba de obtención de ingresos por mes.

Fuente: elaboración propia.

Respecto al recurso de las mascotas también se han realizado las pruebas en Postman. La Figura 32 muestra una de ellas, en la que el endpoint probado es GET /api/mascotas/veterinario/ mascotas-seguimiento/{veterinarioId}, que devuelve las mascotas del veterinario que llevan más de un año sin tener una cita o administrarse una vacuna (o nunca la han tenido) y no tienen cita o vacuna agendada. En esos casos, se considera que esas mascotas necesitarían una visita al veterinario. En el ejemplo, la mascota de nombre kuka no tiene fecha en el campo ultimaVacuna, por lo que su última vacuna fue hace menos de un año o ya tiene agendada una. Sin embargo, nunca ha ido a una cita con su veterinario, por lo que se muestra el mensaje informando de que necesita una revisión. Las mascotas que no necesitan ni una revisión ni una vacuna no son devueltas por este endpoint.

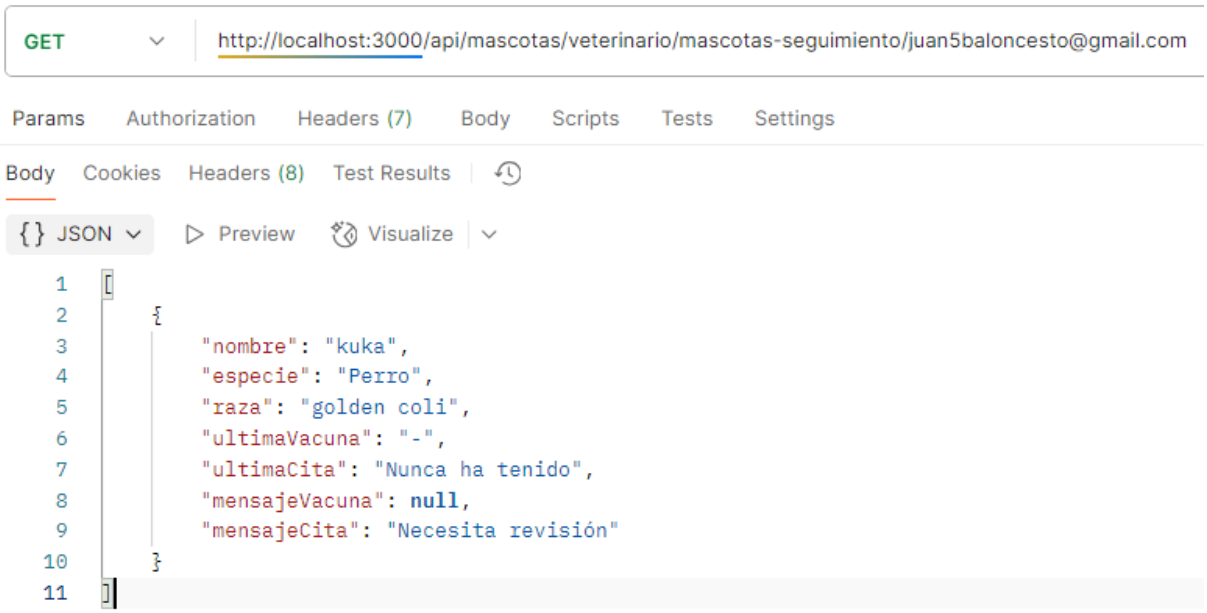



Figura 32: Ejecución de prueba de seguimiento de mascotas.

Fuente: elaboración propia.

Por último, la petición GET a la URL /api/mascotas/veterinario/mascotas-frecuentes/{veterinarioId}. Este endpoint devuelve las mascotas frecuentes de un veterinario. En concreto, se ha considerado que una mascota es frecuente si ha tenido una visita o más al mes con su veterinario en el rango de tiempo seleccionado en el filtro de la aplicación. En este caso, por defecto se implementa para los últimos 3 meses, por tanto las mascotas que devuelve han tenido 3 o más visitas entre citas y vacunas en ese período. Este endpoint permite también filtrar por especie a escoger entre las que atiende el veterinario. En este ejemplo, si no se indica nada por parámetro, por defecto no filtra por especie sino que busca entre todas ellas (Figura 33).

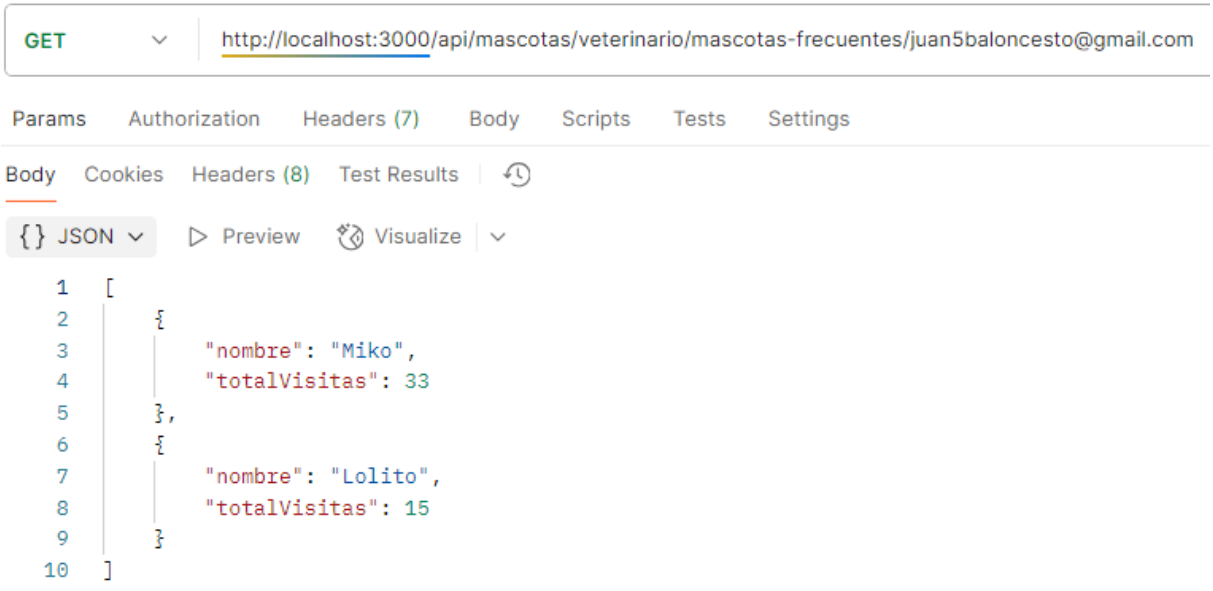



Figura 33: Ejecución de prueba de obtención de mascotas frecuentes.

Fuente: elaboración propia.

# 7

# Conclusiones y Trabajos Futuros

## 7.1. Conclusiones

Una vez terminado el proyecto en todas sus etapas, desde la definición del proyecto y sus objetivos hasta la redacción de la documentación final, pasando por todas las fases descritas y detalladas del ciclo de vida de desarrollo de software, se pueden sacar diferentes conclusiones y valoraciones finales.

Haber podido llevar a cabo desde cero este proyecto ha sido una satisfacción debido a que me generaba una gran ilusión el hecho de implementar esta idea de producto aplicable y usable totalmente por cualquier perfil de público, independientemente de su edad, sexo, nacionalidad, etc. Además, también pretendía desarrollar un sistema que tuviese un toque personal distinto a otros ya existentes, y que tuviera una mezcla de funcionalidades integradas que hiciesen que fuera lo más completo posible. Una vez visto el resultado final, puedo sacar conclusiones positivas respecto a esas pretensiones previas al inicio del desarrollo.

Las expectativas respecto a los objetivos iniciales propuestos se han cumplido. Se ha logrado centralizar la gestión del cuidado de animales en un sistema único, el cual a su vez conecta de alguna manera a los dueños de mascotas con sus veterinarios en tiempo real. Se han implementado las funcionalidades clave como son la gestión de citas y vacunas, las anotaciones de diversos tipos referentes a higiene, alimentación, etc., la gestión económica y el rastreo de paseos. Además, se ha conseguido integrar ambas versiones del sistema - tanto la aplicación móvil como la aplicación web - al mismo proyecto de Firebase, permaneciendo así ambas conectadas a la misma base de datos y servicios en la nube. No obstante, aunque he mencionado que he quedado satisfecho con el resultado de este proyecto, soy consciente de que es

suceptible de ser mejorable, por ejemplo en el diseño de la interfaz (en ambas aplicaciones), en términos de rendimiento o de código más limpio y mantenible.

La realización de este trabajo ha supuesto un reto importante, puesto que no había utilizado con anterioridad la plataforma Firebase, ni había desarrollado una aplicación móvil de esta envergadura, así como tampoco había usado Kotlin como lenguaje de programación ni Angular como framework de desarrollo web. Sin embargo, si bien es cierto que sí había manejado MongoDB, que se trata también de una base de datos NoSQL basada en colecciones y documentos. También el hecho de estar ya familiarizado con lenguajes como Java y JavaScript ha permitido que la curva de aprendizaje con los lenguajes Kotlin y TypeScript haya sido menos pronunciada por sus similitudes.

No obstante, haber ido consultando durante el desarrollo del proyecto distintos manuales, documentación y material online de diferentes comunidades y recursos sumado al hecho de haberlo puesto en práctica me ha permitido obtener una gran cantidad de aprendizajes tanto referentes a conceptos puramente técnicos como otras habilidades como la autogestión, autosuficiencia, responsabilidad individual o la planificación.

En definitiva, este proyecto ha sido muy enriquecedor tanto en lo académico como en lo personal. Me ha permitido afianzar mi orientación profesional hacia el desarrollo web fullstack y al desarrollo de aplicaciones móviles. Haber aprendido y haber abordado nuevos conceptos y tecnologías me ha permitido también aumentar y consolidar mi formación profesional en esta recta final de mi etapa universitaria.

## 7.2. Líneas Futuras de Trabajo

Algunos trabajos futuros que se podrían realizar en este sistema son:

- Permitir a los usuarios la **configuración y personalización de notificaciones**. Los usuarios podrían tener la opción de activar o desactivar notificaciones a modo de recordatorios según su tipo y seleccionar su frecuencia, es decir, cada cuánto tiempo y en qué hora del día desea recibirla. Por ejemplo, podrían activar y personalizar recordatorios de alimentación, de medicación, de higiene e incluso asociados a los paseos. Permitir que el usuario pueda activar notificaciones cuando vayan consiguiento ciertos hitos durante los paseos con sus mascotas, como recibir una alerta notificando cuando haya recorri-

do una cantidad de metros que establezca o cuando haya transcurrido un determinado tiempo de duración serían posibles mejoras a introducir.

- Aunque el sistema no distingue entre distintos tipos de roles al iniciar sesión (tanto dueños de mascotas como veterinarios pueden acceder a ambas aplicaciones), debido a la forma en que se encuentran actualmente desarrolladas y las funciones que tienen implementadas, la aplicación Android se encuentra enfocada a su uso por parte de los propietarios de mascotas y la aplicación web tiene utilidad para los profesionales veterinarios. Una propuesta de línea futura sería extender ambas aplicaciones con **funcionalidades y pantallas nuevas dedicadas exclusivamente a cada rol**. En concreto, en la aplicación móvil se podrían implementar nuevas pantallas correspondientes al rol de veterinario (se mostrarían si el usuario autenticado es un veterinario) y, por tanto, nuevas funcionalidades exclusivas para él (gestión de solicitudes de citas y vacunas, generación de informes, control de ingresos y gastos…). Por otro lado, en la aplicación web se podrían implementar funcionalidades destinadas a los clientes previa comprobación también de si el usuario que ha iniciado sesión es un veterinario o no. De esta manera, si no fuese veterinario el usuario que accede, sería interesante añadir a la aplicación web características y prestaciones similares a las que ya hay implementadas para dispositivos móviles.

- A pesar de que Android es el sistema operativo para dispositivos móviles más empleado, iOS también tiene gran peso en este mercado. Se podría desarrollar una versión que fuese funcional y operativa en **iOS** y abarcar así un mayor número de usuarios.

- Otras opciones suceptibles de ser añadidas a la aplicación móvil son:

    - Implementación de un **foro de debate** con intervenciones donde los usuarios intercambien opiniones, compartan sus experiencias y resuelvan dudas entre ellos.
    - Incorporación de **botones específicos en los paseos** para registrar eventos como la micción y la defecación de las mascotas, de los que se podría guardar el momento y el lugar exacto en el que se ha producido durante un paseo.
    - **Sistema de logros o insignias** por cuidados, paseos o rutinas completadas, fomentando así el compromiso de los dueños con el cuidado de sus animales.

# Referencias


Alliance, T. A. (2025). *Principios del manifiesto Ágil.* https://agilemanifesto.org/iso/es/principles.html.

Angular. (2025a). *Angular material ui component library.* https://material.angular.dev.

Angular. (2025b). *Angular - the rxjs library.* https://v17.angular.io/guide/rx-library.

Angular. (2025c). *Routing.* https://angular.dev/guide/routing.

ApexCharts. (2025). *Angular chart examples & samples demo - apexcharts.js.* https://apexcharts.com/angular-chart-demos.

AWS. (2025). *Bases de datos no relacionales.* https://aws.amazon.com/es/nosql/.

Express. (2025). *Express.js.* https://en.wikipedia.org/wiki/Express.js.

Firebaese, G. (2025). *Firebase cloud messaging.* https://firebase.google.com/docs/cloud-messaging.

Firebase, G. (2025a). *Cloud functions for firebase.* https://firebase.google.com/docs/functions.

Firebase, G. (2025b). *Cloud storage for firebase.* https://firebase.google.com/docs/storage.

Firebase, G. (2025c). *Firebase authentication.* https://firebase.google.com/products/auth.

Firebase, G. (2025d). *Firestore | firebase.* https://firebase.google.com/docs/firestore.

Fonts, G. (2025). *Geologica - google fonts.* https://fonts.google.com/specimen/Geologica.

for Developers, G. (2025a). *Appcompat | jetpack | android developers.* https://developer.android.com/jetpack/androidx/releases/appcompat.

for Developers, G. (2025b). *Camerax overview.* https://developer.android.com/media/camera/camerax.

for Developers, G. (2025c). *Javascript sdk | node.js (client) api reference | firebase.* https://firebase.google.com/docs/reference/node.

for Developers, G. (2025d). *Maps sdk for android overview.* https://developers.google.com/maps/documentation/android-sdk/overview.

for Developers, G. (2025e). *Material components | mobile | android developers.* https://developer.android.com/design/ui/mobile/guides/components/material-overview.

for Developers, G. (2025f). *Recyclerview | jetpack | android developers.* https://developer

.android.com/jetpack/androidx/releases/recyclerview.

FullCalendar. (2025). *Angular component - docs | fullcalendar.* https://fullcalendar.io/docs/angular.

Google. (2025). *angular/angularfire: Angular + firebase.* https://github.com/angular/angularfire.

Google. (2025a). *Angular developer guides.* https://v17.angular.io/guide/developer-guide-overview.

Google. (2025b). *Introducción a android studio.* https://developer.android.com/studio/intro?hl=es-419.

Jahoda, P. (2025). *Philjay/mpandroidchart.* https://github.com/PhilJay/MPAndroidChart.

JetBrains. (2025). *Kotlin programming language.* https://kotlinlang.org/.

Judd, S. (2025). *bumptech/glide.* https://github.com/bumptech/glide.

Microsoft. (2025). *Typescript: Javascript with syntax for types.* https://www.typescriptlang.org.

Microsoft Corporation. (2025). *Visual studio code - code editing. redefined.* https://code.visualstudio.com.

Node.js. (2025). *Introduction to node.js.* https://nodejs.org/es/learn/getting-started/introduction-to-nodejs.

omatech. (2025). *¿qué es una web spa?* https://www.omatech.com/blog/2022/08/31/que-es-una-web-spa/#:~:text=Ventajas%20de%20la%20web%20SPA&text=Las%20aplicaciones%20de%20una%20sola,completa%20en%20la%20primera%20petici%C3%B3n.

Pictogrammers. (2025). *Material design icons - icon library - pictogrammers.* https://pictogrammers.com/library/mdi/.

Postman, Inc. (2025). *Postman api platform.* https://www.postman.com.

WHATWG. (2025). *Html standard — living standard.* https://html.spec.whatwg.org.

Wikipedia. (2025). *Haversine formula - wikipedia.* https://en.wikipedia.org/wiki/Haversine_formula.

Wikipedia contributors. (2024). *Android - wikipedia.* https://es.wikipedia.org/wiki/Android.

# Apéndice A
# Manual de Instalación

## A.1. Instalación de la Aplicación Móvil

- **A través de archivo APK**

  A continuación se mencionan los pasos necesarios para proceder a la instalación de la aplicación movíl mediante su correspondiente archivo con extensión .apk.

  1. Descargar el archivo apk proporcionado en la carpeta de entrega del proyecto.
  2. Tranferir el archivo a un dispositivo Android.
  3. Ejecutar el archivo apk en el dispositivo y completar el proceso de instalación.

     Nota: Si al abrir el archivo Android muestra un aviso de que el teléfono no permite instalar apps desconocidas, seleccionar **Ajustes ->Permitir desde esta fuente**. A continuación, regresar y confirmar la instalación del APK.

- **A través del código fuente en Android Studio**

  En esta ocasión se detallan los pasos para instalar la aplicación cargando el proyecto en Android Studio y ejecutándola en un dispositivo móvil físico. Para ello, es necesario que previamente se haya descargado e instalado este entorno de desarrollo desde su página oficial (https://developer.android.com/studio).

  También es necesario previamente activar las opciones de desarrollador dentro de los ajustes del teléfono. Para ello:

  - Abrir **Ajustes ->Acerca del teléfono/Sobre el teléfono**
  - Pulsar sobre la opción **Versión del sistema operativo** siete veces.
  - Volver a **Ajustes** y pulsar **Ajustes adicionales ->Opciones de desarrollador**. En esta pantalla, activar las opciones **Depuración por USB** e **Instalar vía USB**.

1. Descargar carpeta del proyecto en un directorio del ordenador,
2. En Android Studio pulsar **File ->Open** y seleccionar la carpeta del proyecto.
3. Conectar el dispositivo Android al ordenador mediante un cable USB. Aceptar en el teléfono los permisos de depuración USB y pulsar la opción **Utilizar USB para Transferencia de archivos**.
4. A continuación aparecerá en la barra superior de Android Studio el nombre del dispositivo. Pulsar en el icono de **Run app** de la derecha.

## A.2. Instalación de la Aplicación Web

Para poner en marcha la aplicación web es necesario tener instalados en el sistema Node.js, npm y Angular CLI. Se recomienda utilizar las versiones más recientes con el objetivo de garantizar la compatibilidad y el correcto funcionamiento del proyecto.

Los pasos a seguir para la instalación y ejecución son los siguientes:

1. Descargar la carpeta con el código fuente y guardarla en un directorio local.
2. En una terminal de comandos, acceder al directorio del backend y descargar las dependencias:

   ```
   cd backend-tfg
   npm install
   ```

3. En la misma carpeta "backend-tfg", ejecutar el servidor con el siguiente comando:

   ```
   npm run dev
   ```

4. En otra terminal, acceder al directorio del frontend y descargar también sus dependencias:

   ```
   cd frontend-tfg
   npm install
   ```

5. Iniciar la aplicación cliente, también desde la misma carpeta “frontend-tfg” con el comando:

   ```
   ng serve
   ```

6. Finalmente, acceder desde un navegador a la dirección http://localhost:4200 para interactuar con la aplicación web.

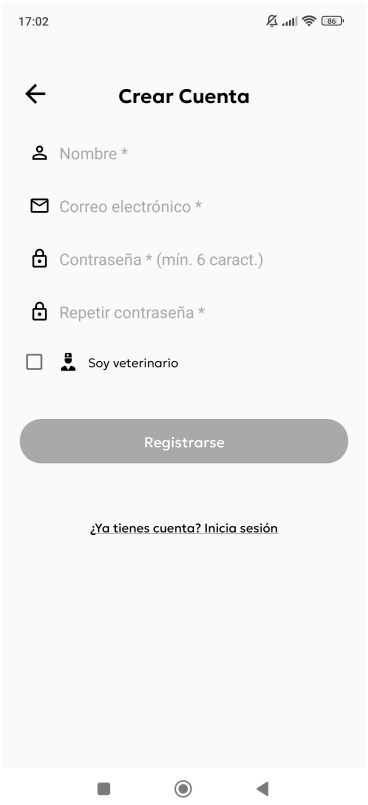
**Notas:**

- El servidor y el cliente deben estar simultáneamente en ejecución para el correcto funcionamiento de la aplicación web.
- El backend se ejecuta por defecto en el puerto 3000, quedando disponible en la URL http://localhost:3000/api/.

# Apéndice B
# Manual de Usuario

## B.1. Aplicación Móvil

### B.1.1. Registrarse



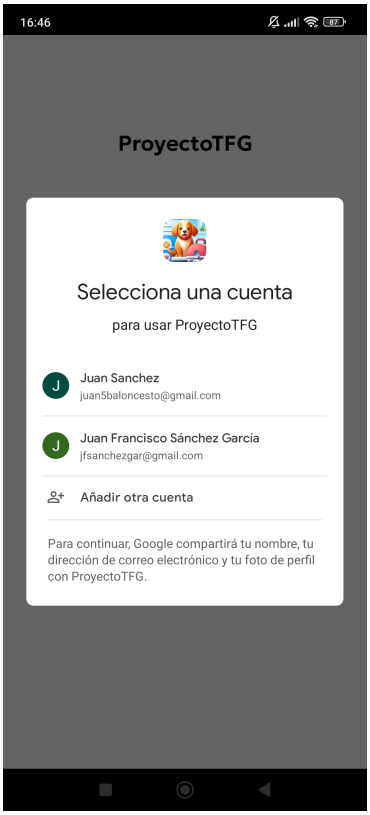

En la pantalla inicial de la aplicación, pulsar sobre el texto en negrita “Registrarse”. La aplicación abrirá la pantalla de registro con los campos a rellenar. Una vez introducidos los datos, pulsar en el botón “Registrarse”.

### B.1.2. Iniciar sesión

#### Con correo electrónico y contraseña


Al abrir la aplicación en el dispositivo móvil, introducir las credenciales y pulsar el botón de “Iniciar sesión”. Si existe una cuenta con esos datos, se abre la página principal con las mascotas registradas o con la opción de añadir la primera mascota a la aplicación.

### Con Google

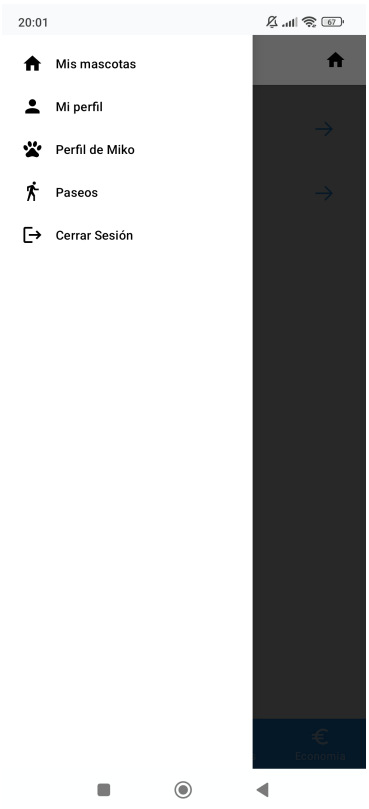



Al abrir la aplicación, pulsar sobre el botón “Continuar con Google” y escoger la cuenta con la que se desee acceder a la aplicación. A continuación, se cargará la página principal con las mascotas registradas o con la opción de añadir la primera mascota a la aplicación.

### B.1.3. Recuperar contraseña

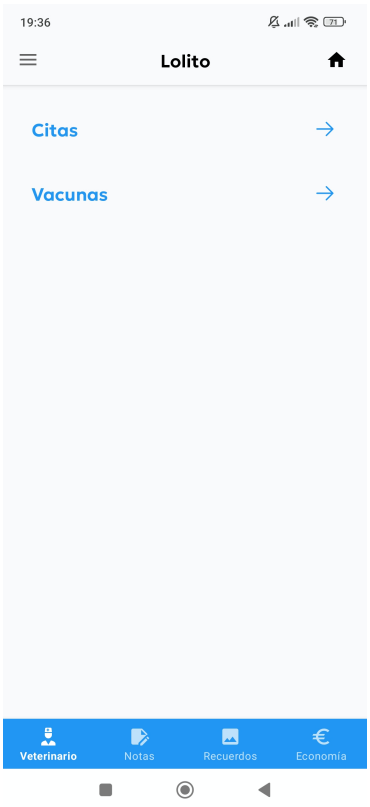

Para restablecer la contraseña se debe pulsar en el texto situado debajo del campo de la contraseña "¿Has olvidado tu contraseña?". A continuación, introducir el correo electrónico de la cuenta que se desee cambiar o recuperar la contraseña. Una vez introducido, se enviará un correo electrónico donde se puede indicar una nueva contraseña.

### B.1.4. Cerrar sesión



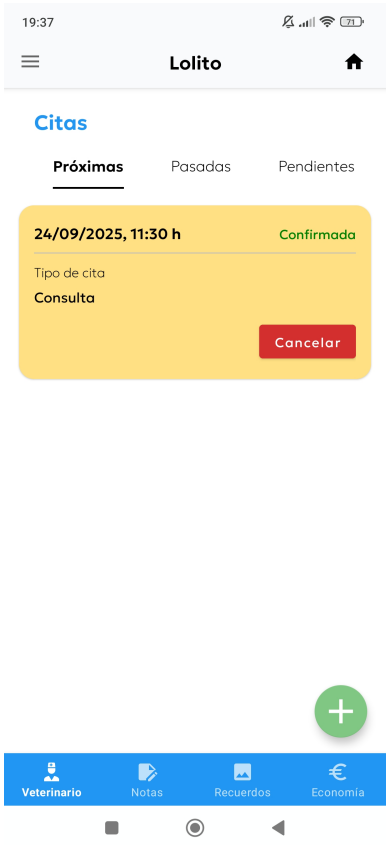

En la barra superior, desplegando el menú lateral a través del botón de la esquina superior izquierda, se puede cerrar sesión en la aplicación pulsando la opción “Cerrar sesión”. La aplicación volverá a mostrar la pantalla de incio de sesión.

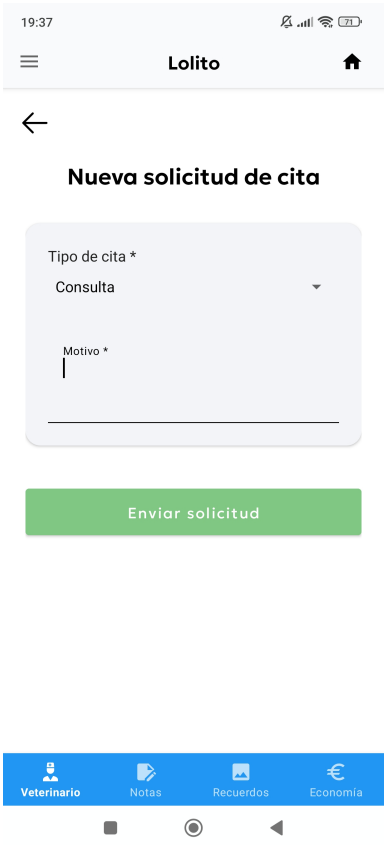
### B.1.5. Gestión de citas y vacunas




Pulsar en el perfil de una mascota y, a continuación, se abrira la primera opción del menú inferior denominada “Veterinario”. En ella, pulsar sobre “Citas” o “Vacunas” y se cargará el historial de citas o vacunas de la mascota.

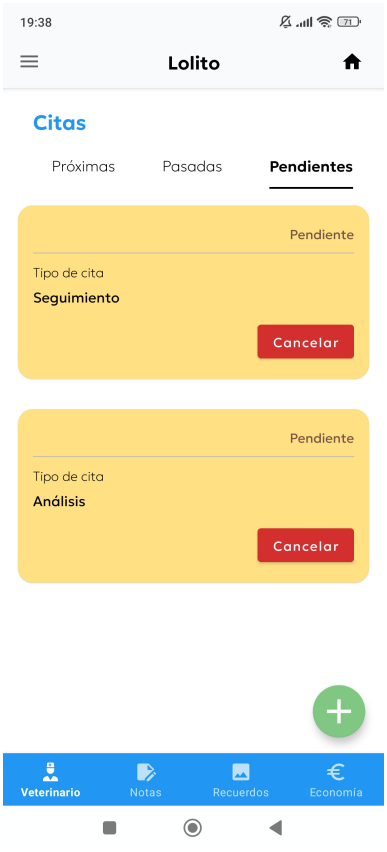
### Solicitar cita/vacuna




Una vez accedido a la pantalla de “Citas” o “vacunas”, pulsar sobre el botón redondo verde con una cruz blanca en la esquina inferior derecha. Se abrirá la pantalla de solicitud de cita/-vacuna. Una vez introducidos los datos, pulsar en el botón ´´Enviar solicitud” y la aplicación abrirá de nuevo el historial de visitas.

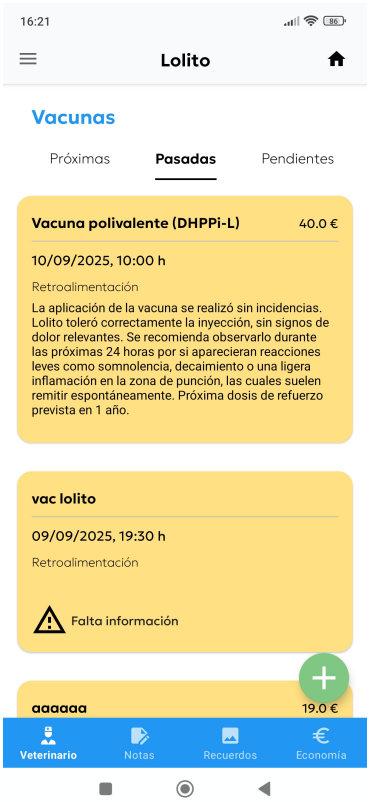
### Cancelar cita/vacuna





Para cancelar una cita/vacuna próxima o pendiente, pulsar sobre el botón rojo “Cancelar” en su correspondiente tarjeta amarilla.

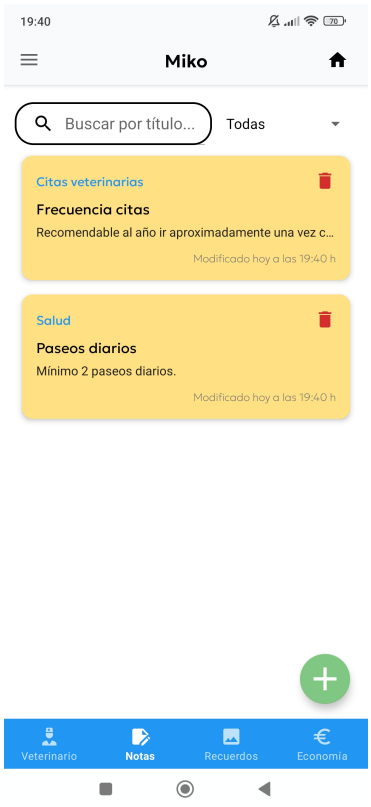
### Visualizar datos veterinarios de cita/vacuna




En el historial de citas pasadas, pulsar sobre la tarjeta amarilla de una de ellas y se abrirá una pestaña inferior en la misma pantalla con la información proporcionada por el veterinario tras la cita.

En el historial de vacunas pasadas, si el veterinario ha proporcionado la información correspondiente de una vacuna esta saldrá dentro de la misma tarjeta amarilla.

### B.1.6. Gestión de notas




Pulsar en el perfil de una mascota y, en el menú inferior de la pantalla que se abre, pulsar sobre “Notas”. La pantalla resultante mostrará la lista de notas ya creadas.

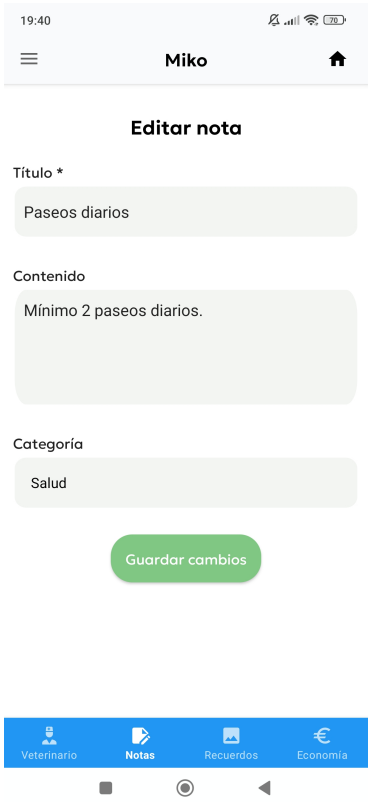
### Añadir nota



En el historial de notas pulsar sobre el botón redondo verde con una cruz blanca situado en la esquina inferior derecha. Introducir la información de la nota y pulsar en “Añadir nota”. Una vez creada con éxito, la aplicación mostrará la pantalla con todas las notas incluida la que se acaba de añadir.

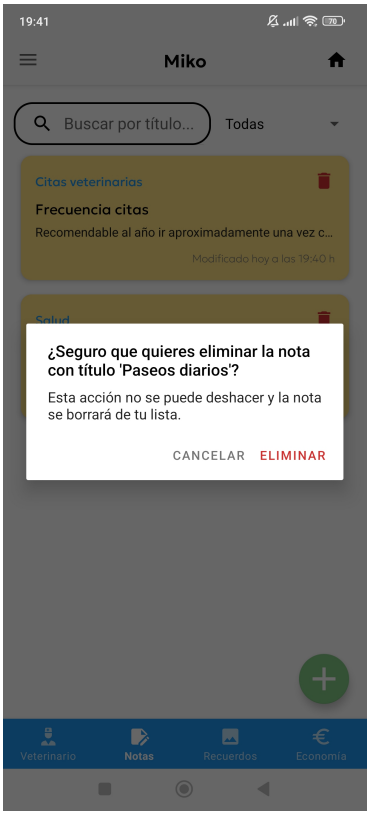
Una nota puede no pertenecer a ninguna categoría, en cuyo caso la categoría a seleccionar es “Ninguna”.

### Editar nota



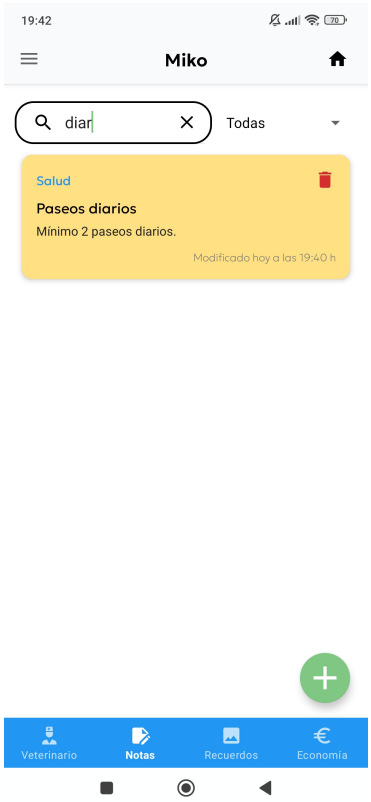

En el historial de notas, pulsar el rectángulo amarillo de una de ellas. A continuación se abrirá la pantalla donde se puede modificar el título, contenido y/o categoría de la nota. Una vez pulsado el botón de “Guardar cambios” tras editar la nota, se abrirá el historial de notas donde se reflejarán los cambios realizados en dicha nota.

### Eliminar nota




En la tarjeta amarilla de una nota, pulsar sobre el icono de una basura de color rojo en la esquina superior derecha, y confirmar pulsando en “Eliminar”.

### Filtrar notas




En la pantala del historial de notas, en la parte superior aparecen los dos campos a introducir. Una vez introducidos los datos como título y/o categoría, aparecerán todas las notas que coincidan con esos datos. En cuanto al título, aparecerán las notas que en su título contengan la cadena de texto introducida en el título, no sólo las que empiecen por dicho texto.

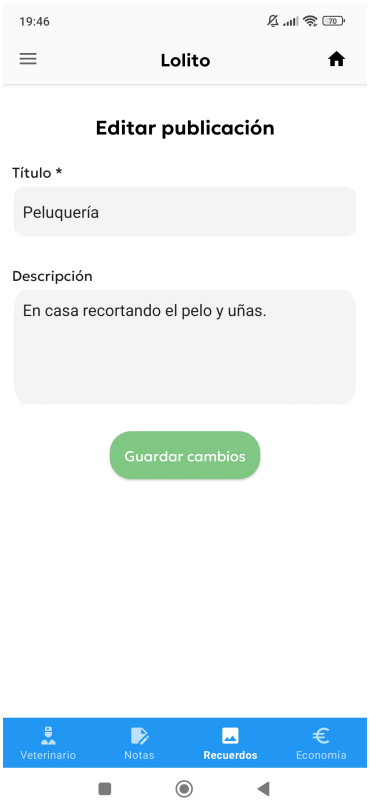
### B.1.7. Gestión de recuerdos


Pulsar en el perfil de una mascota y, en el menú inferior de la pantalla que se abre, pulsar sobre “Recuerdos”. La pantalla resultante mostrará la lista de recuerdos ya creados.

### Añadir recuerdo

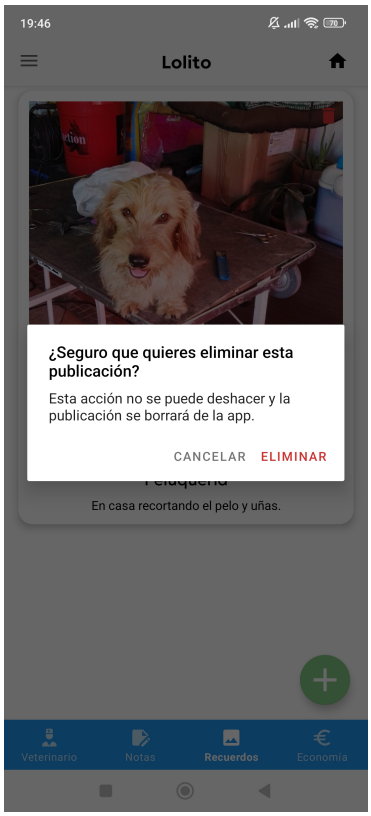

En el historial de recuerdos pulsar sobre el botón redondo verde con una cruz blanca situado en la esquina inferior derecha. Introducir la información que se pide y pulsar en “Añadir publicación”. Una vez creada con éxito, la aplicación mostrará la pantalla con todos los recuerdos con su título y descripción, incluido el que se acaba de añadir.

### Editar recuerdo




En el historial de recuerdos, pulsar la fotografía de cualquiera de ellos. A continuación se abrirá la pantalla donde se puede modificar el título y/o la descripción. Una vez pulsado el botón de “Guardar cambios” tras editar la publicación, se abrirá de nuevo el historial donde se reflejarán los cambios recién guardados.

### Eliminar recuerdo

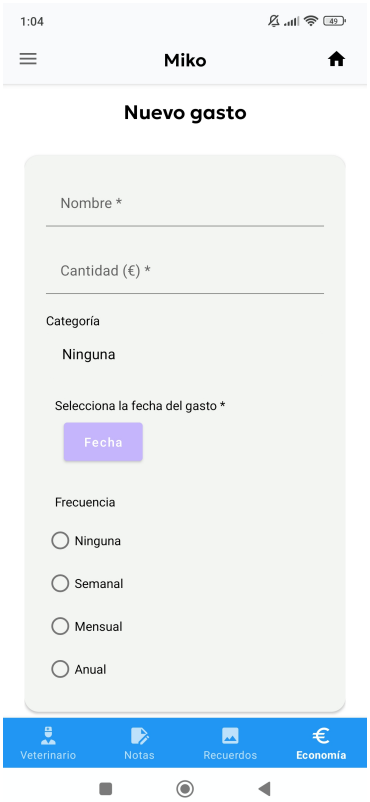



En la tarjeta blanca de un recuerdo, pulsar sobre el icono de una basura de color rojo en la esquina superior derecha, y confirmar pulsando en “Eliminar”.

### B.1.8. Gestión de economía


Pulsar en el perfil de una mascota y, en el menú inferior de la pantalla que se abre, pulsar sobre “Economía”. La pantalla resultante mostrará la lista de gastos ya añadidos.

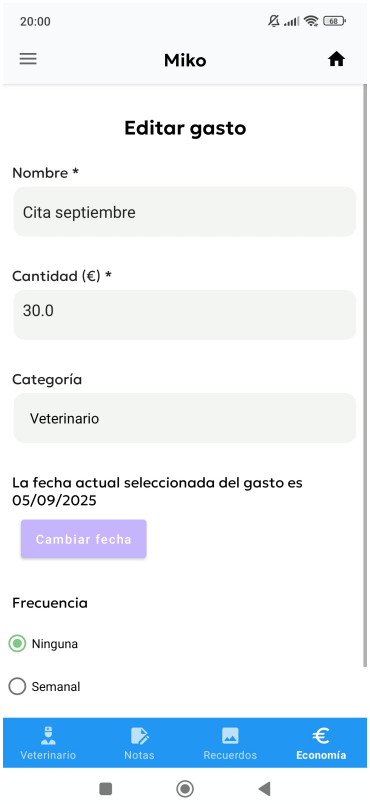
### Añadir gasto




Pulsar sobre el botón redondo de la esquina inferior derecha para abrir el formulario de datos del gasto. Rellenar y pulsar el botón de “Añadir gasto”. Si no se recuerda la fecha exacta del gasto, seleccionar cualquier día del mes. En este caso, el día no es relevante pero sí lo es el mes de cara a la generación de estadísticas.

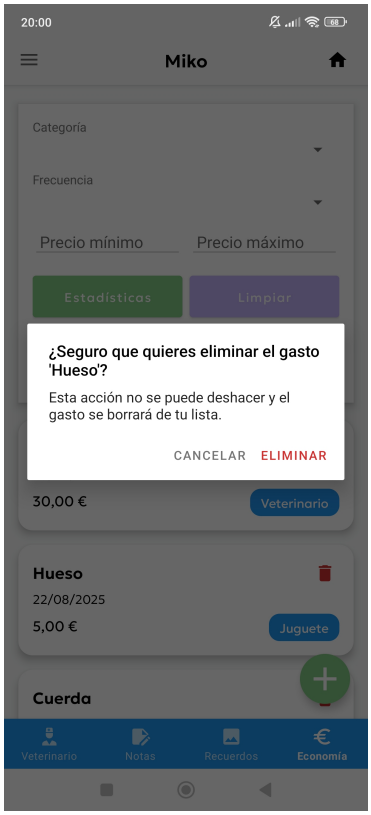
Al igual que con la categoría, en caso de que no se conozca la frecuencia de un gasto (o simplemente no tenga) se puede seleccionar la opción “Ninguna”.

### Editar gasto



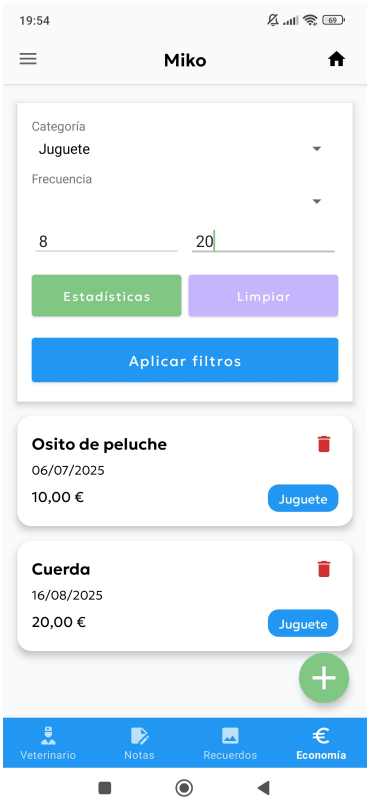

En el historial de gastos representados cada uno sobre un rectángulo blanco, pulsar sobre el que se quiera editar y se abrirá la pantalla donde, una vez modificados los datos y pulsado el botón “Guardar cambios”, se cargará de nuevo la pantalla con todos los gastos con el gasto que se acaba de modificar actualizado.

### Eliminar gasto



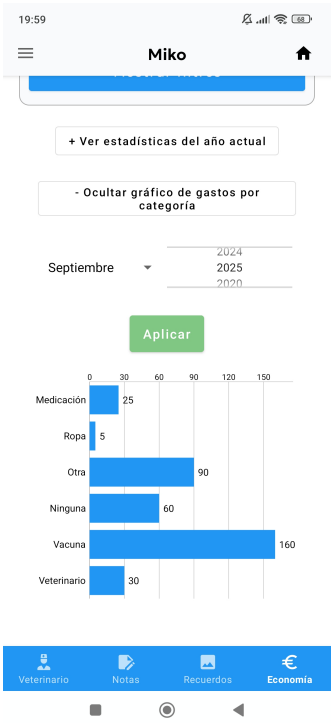

En un gasto, pulsar sobre el icono de una basura de color rojo en la esquina superior derecha, y confirmar pulsando en “Eliminar”.

### Filtrar gastos




En la pantala del historial de gastos, en la parte superior aparecen todos los campos para filtrar. Una vez introducidos los datos que se desee, sólo aparecerán en el historial los gastos que coincidan con dicha información.

### Ver estadísticas de gastos


En la pantalla del historial de gastos, pulsar sobre el botón verde “Estadísticas”. Entonces, la aplicación cargará la pantalla correspondiente con diferentes elementos. Las dos primeras tarjetas muestran la categoría con más y menos gasto respectivamente del total histórico, esto es, de entre todas las añadidas a la aplicación.

El siguiente elemento es un filtro donde se puede introducir una categoría y fecha, y una vez se pulsa el botón azul “Mostrar filtros” la aplicación muestra las estadísticas de los gastos de la categoría y período seleccionado. Para mostrar datos de nuevo pero con otra información introducida en los filtros, pulsar sobre “Ocultar filtros” y de nuevo en “Mostrar filtros” para que se actualicen correctamente las tarjetas con los nuevos datos.

Al pulsar sobre el penúltimo elemento, “Ver estadísticas del año acual”, la aplicación carga en la pantalla información sobre los gastos sólo del año actual.

Si se abre el último elemento (“Ver gráfico de gastos por categoría”), se muestra un filtro donde se puede seleccionar un mes (o todo el año) junto con un año determinado. En el caso de haber gastos registrados en esa fecha, la aplicación mostrará un gráfico de barras horizontales con los gastos totales por categoría en el mes y año seleccionados. Si no se encuentran gastos, no se carga ninguna gráfica y en su lugar aparece el mensaje “No hay gastos en esta fecha”.

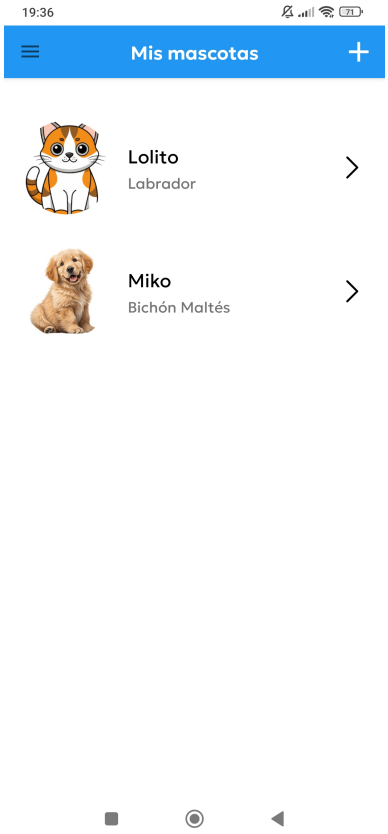
### B.1.9. Gestión de mi perfil

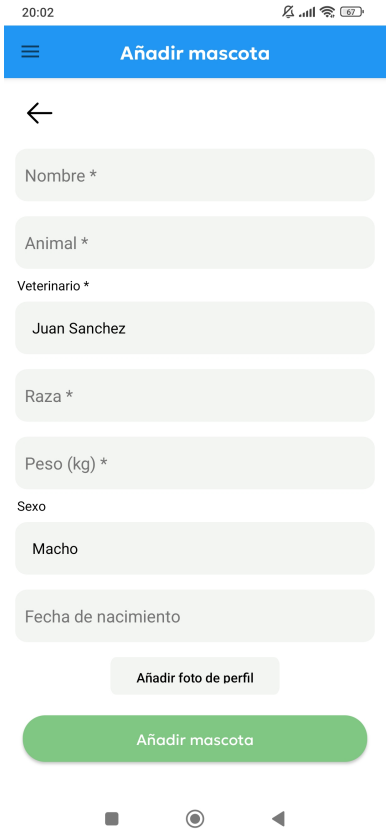

Una vez se accede al perfil de una mascota, abrir el menú lateral mediante el botón situado en la esquina superior izquierda de la pantalla representado con tres líneas horizontales. En este menú, pulsar la opción “Mi perfil”. Se abrirá la pantalla con el nombre y correo electrónico asociado a la cuenta que la que se ha iniciado sesión en la aplicación.

En esta pantalla se puede editar el nombre pulsando sobre él, introduciendo uno nuevo y pulsando en “Guardar”. Además, también se puede añadir una foto de perfil (botón “Añadir foto de perfil”) o cambiarla si ya se había añadido una (botón “Cambiar foto de perfil”) seleccionando una en el archivo del teléfono móvil.

### B.1.10. Gestión de perfiles de mascotas

#### Añadir mascota

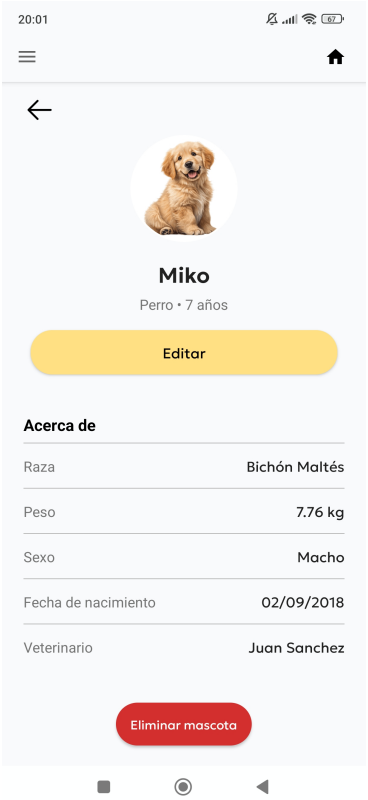



En la pantalla principal de la aplicación que se muestra al iniciar sesión, pulsar en el botón de la esquina superior derecha en forma de cruz blanca. Rellenar los datos correspondientes en la pantalla que se abre y pulsar sobre “Añadir mascota”. Una vez creada, la aplicación mostrará de nuevo la pantalla principal donde aparecerá el perfil de la mascota que se acada de añadir.

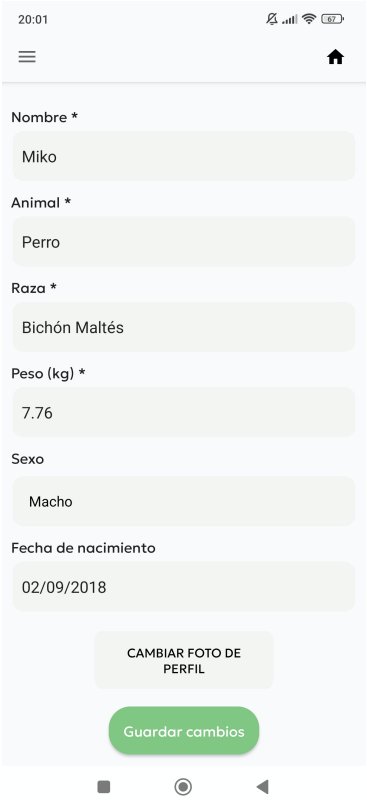
Si no se tiene ningún perfil de mascota registrado en la aplicación, también se puede añadir una mascota desde el botón azul “Añadir mascota” situado en el centro de la pantalla.

### Visualizar datos de mascota



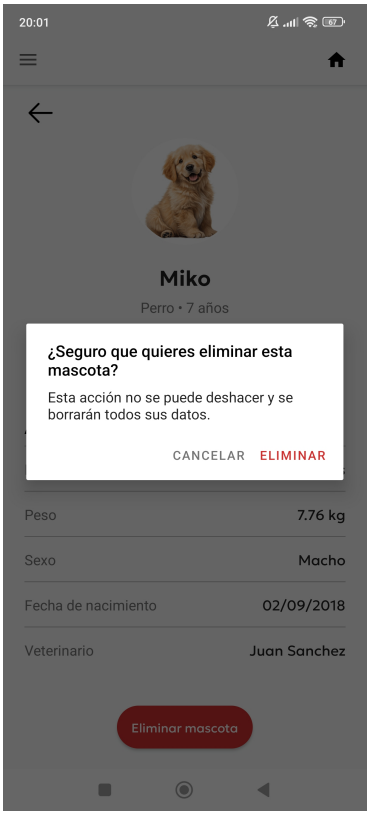

Una vez se accede al perfil de una mascota, abrir el menú lateral mediante el botón situado en la esquina superior izquierda de la pantalla representado con tres líneas horizontales. En este menú, pulsar la opción “Perfil de [nombre de mascota]”. Se abrirá la pantalla con la información asociada a la mascota como especie, edad, raza, peso, etc.

### Editar datos de mascota



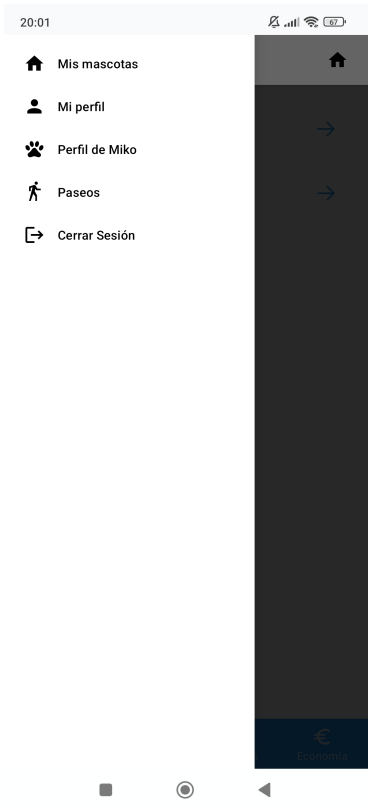

Dentro de la pantalla del perfil de una mascota, pulsar sobre el botón con fondo amarillo “Editar”. Se abrirá un formulario con los campos a editar. Una vez cambiados los datos que se desee, pulsar sobre “Guardar cambios”, tras lo cual la aplicación mostrará de nuevo la pantalla con la información actualizada del animal.

### Eliminar mascota



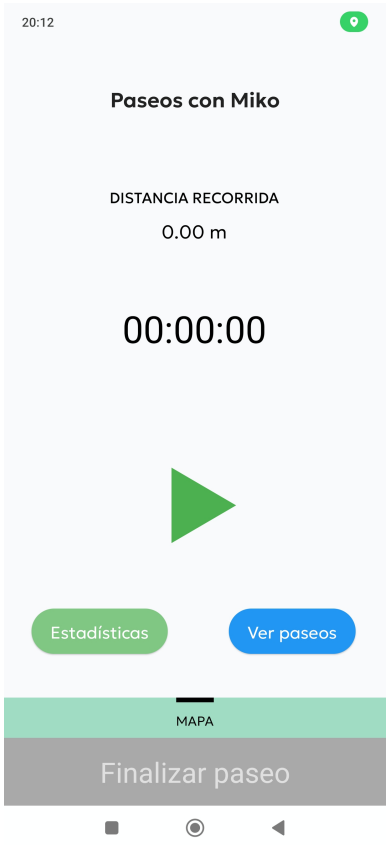

En la pantalla del perfil de una mascota, pulsar sobre el botón rojo inferior “Eliminar mascota” y confirmar la acción pulsando en “Eliminar”.

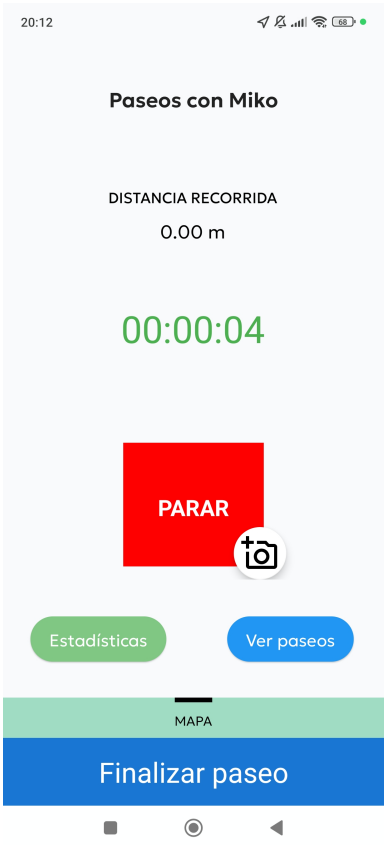
### B.1.11. Gestión de paseos




Para acceder a la pantalla de “Paseos” hay que pulsar en esa misma opción dentro del menú lateral desplegable desde la barra superior de la aplicación.

### Empezar/parar/reanudar/finalizar paseo




Para empezar un paseo se debe pulsar el botón verde en forma de “Play”. Si se desea parar o reanudar un paseo hay que pulsar sobre el botón cuadrado rojo o naranja respectivamente. Para finalizar un paseo en curso, pulsar el botón azul de la parte inferior “Finalizar paseo”.

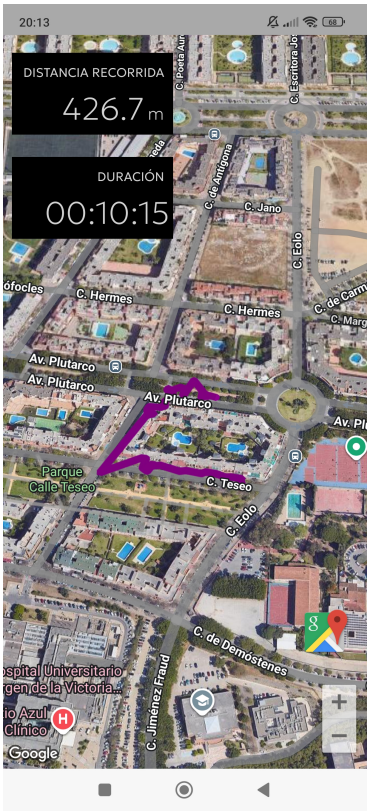
### Guardar/eliminar un paseo


Al finalizar un paseo, en caso de que se quiera guardar y mantener los datos del mismo en la aplicación, se debe cerrar la ventana emergente (pop up) que aparece tras finalizar el paseo pulsando la cruz de la esquina superior derecha. Tras esto, la aplicación cerrará la ventana emergente y el paseo quedará guardado automáticamente.

Para eliminar un paseo hay que pulsar el botón rojo “Eliminar” dentro de la ventana emergente del paseo terminado, y confirmar la acción pulsando de nuevo “Eliminar”.

También es posible eliminar un paseo posteriormente, accediendo al historial de paseos mediante el botón azul “Ver paseos” y pulsando sobre el botón “Eliminar” en el paseo que se desee eliminar de la aplicación.

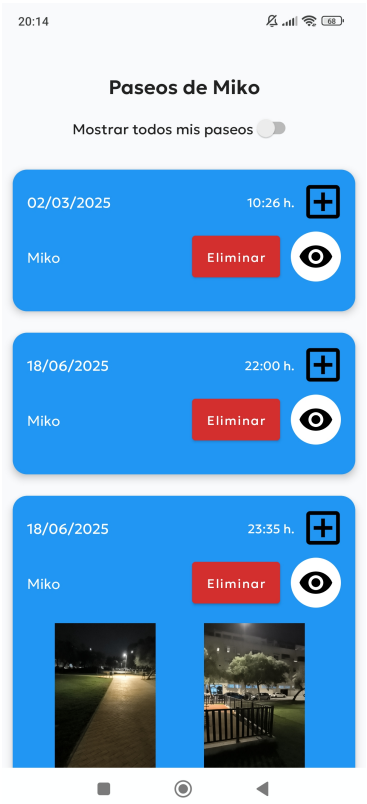
### Visualizar recorrido de un paseo en un mapa




Pulsando el botón con un ojo negro sobre fondo blanco en la parte inferior de la ventana emergente al finalizar un paseo, la aplicación abrirá un mapa de Google Maps con el recorrido realizado pintado con líneas de color morado sobre el mapa, además de mostrar la distancia recorrida y duración del paseo.

No obstante, también se puede acceder al recorrido de un paseo dibujado sobre un mapa pulsando el botón azul “Ver paseos” en la pantalla principal de gestión de paseos. La aplicación mostrará todos los paseos realizados. Para visualizar el recorrido de cada uno de ellos, pulsar sobre el botón con un ojo negro sobre fondo blanco al igual que al finalizar un paseo.

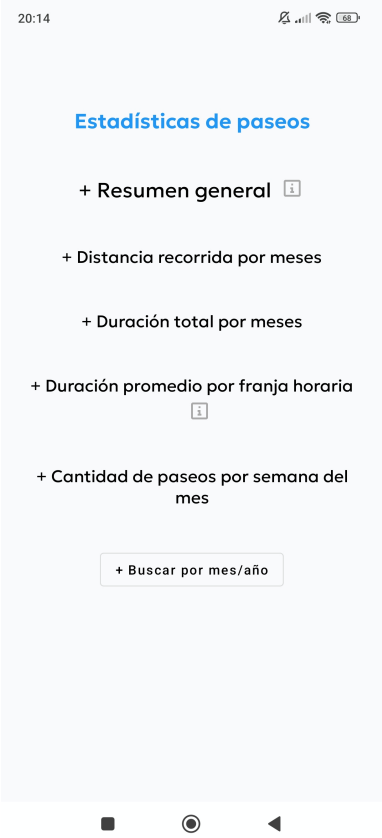
### Visualizar historial de paseos




Desde la pantalla principal de gestión de paseos, pulsar en el botón azul “Ver paseos”. Si uno o más paseos realizados y guardados, se mostrarán en una lista uno debajo de otro con información en cada uno como la fecha y hora de inicio y la mascota con la que se realizó el paseo.

Si se pulsa sobre el botón de la esquina superior derecha de cada tarjeta azul que representa un paseo, se podrá visualizar en una ventana emergente su distancia recorrida y su duración.

## Ver estadísticas de paseos

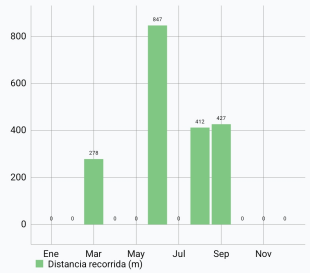



20:14

## Estadísticas de paseos

+ Resumen general

- Distancia recorrida por meses




+ Duración total por meses

+ Duración promedio por franja horaria

+ Cantidad de paseos por semana del

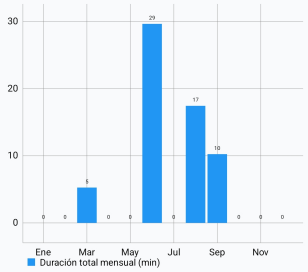
20:14

## Estadísticas de paseos

+ Resumen general

+ Distancia recorrida por meses

- Duración total por meses




+ Duración promedio por franja horaria

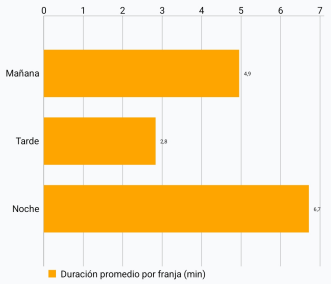
+ Cantidad de paseos por semana del

20:14

+ Distancia recorrida por meses

+ Duración total por meses

- Duración promedio por franja horaria




+ Cantidad de paseos por semana del mes

+ Buscar por mes/año

20:14

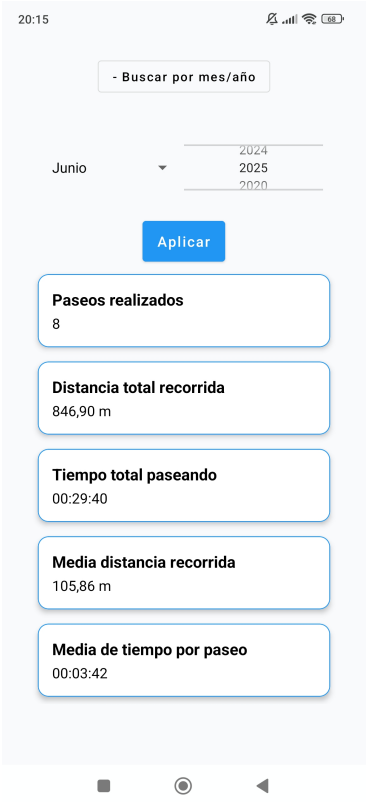
+ Distancia recorrida por meses

+ Duración total por meses

+ Duración promedio por franja horaria

- Cantidad de paseos por semana del mes (durante el año actual)


+ Buscar por mes/año




En la pantalla principal de los paseos, pulsar sobre el botón verde “Estadísticas”. Entonces, la aplicación cargará la pantalla de estadísticas de paseos con diferentes elementos. Al pulsar sobre cada uno, la aplicación mostrará diferente información como total de paseos, distancia y duración total y promedio, etc. También se puede filtrar por mes y año, mostrándose datos sobre los paseos realizados en esa fecha. Además, se pueden visualizar gráficas que representan estadísticas como la distancia total recorrida o duración total paseando en cada mes del año actual.

## B.2. Aplicación Web

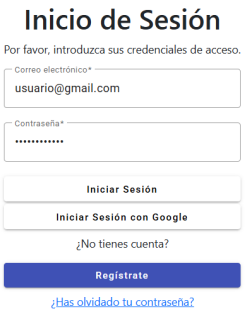
### B.2.1. Registrarse

#### Con correo electrónico y contraseña


Para registrarse con unas credenciales hay que introducir una dirección de correo electrónico y una misma contraseña dos veces. Después, pulsar el botón “Registrarse”.

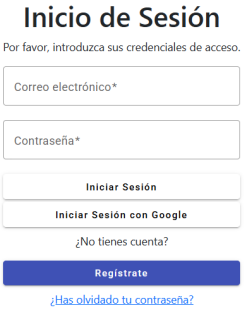
### B.2.2. Iniciar sesión

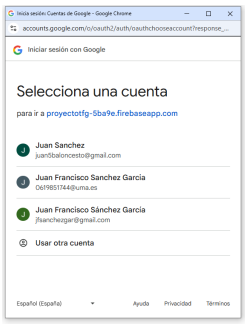
#### Con correo electrónico y contraseña




Introducir las credenciales y pulsar el botón de “Iniciar Sesión”. Si son correctas, se accede a la aplicación y se muestra la pantalla con el calendario de eventos.

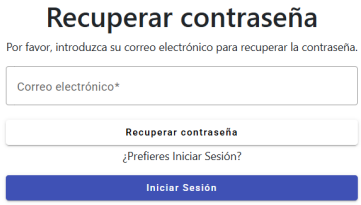
### Con Google



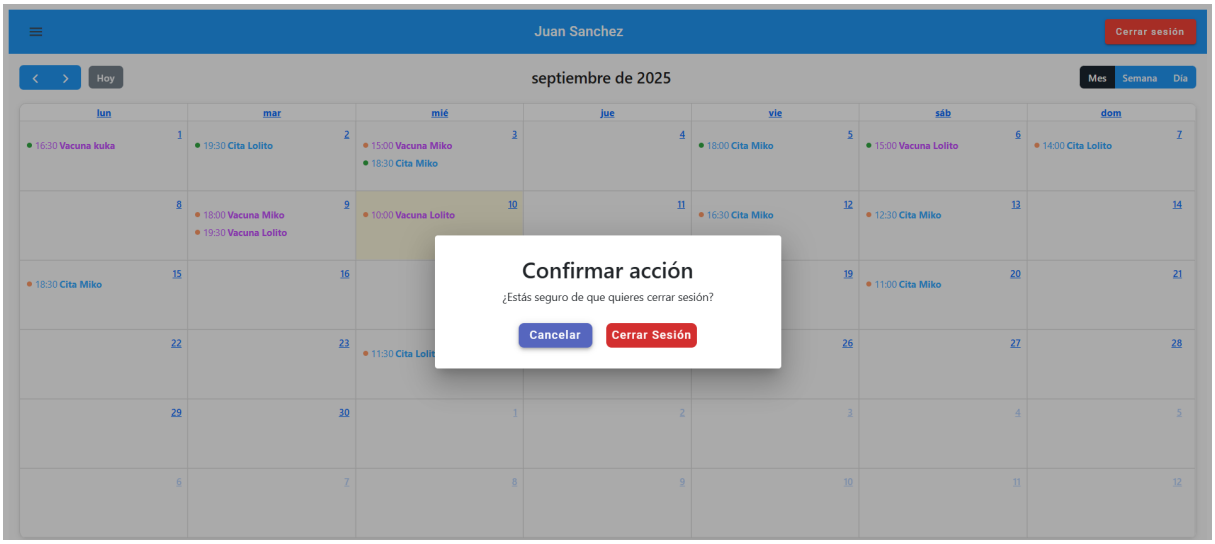

Pulsar directamente sobre el botón “Iniciar Sesión con Google” y, a continuación, seleccionar una cuenta. Se accederá a la aplicación con la cuenta de Google seleccionada y la aplicación mostrará la pantalla con el calendario de eventos.

### B.2.3. Recuperar contraseña



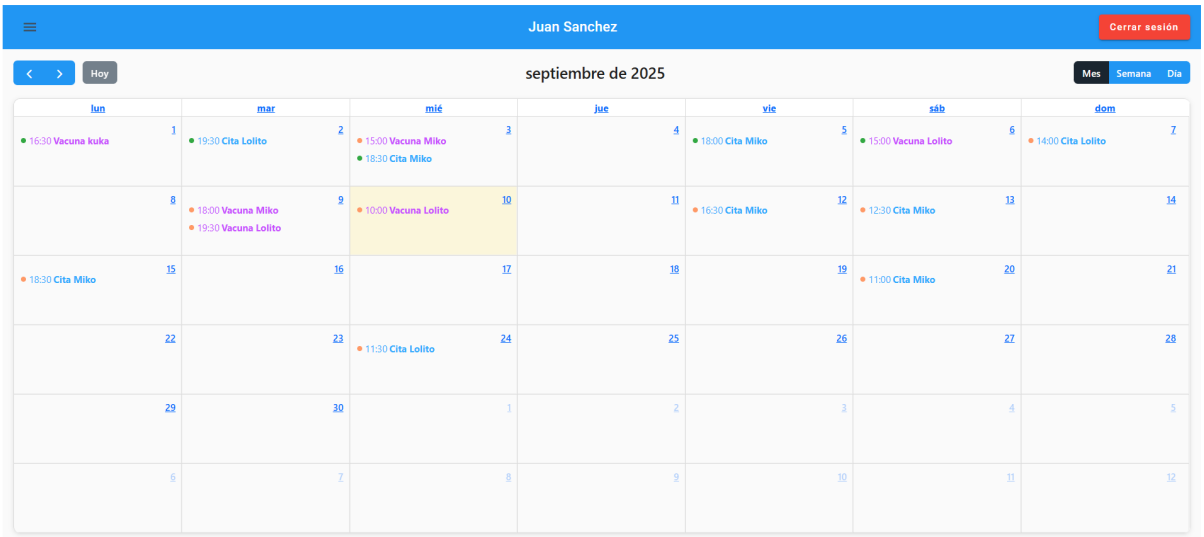

Pulsar sobre el texto "¿Has olvidado tu contraseña?". Se abrirá una pantalla donde se debe introducir una dirección de correo electrónico y pulsar el botón "Recuperar contraseña". Una vez hecho, se enviará automáticamente un mensaje de correo a la dirección indicada para poder cambiar la contraseña.

### B.2.4. Cerrar sesión



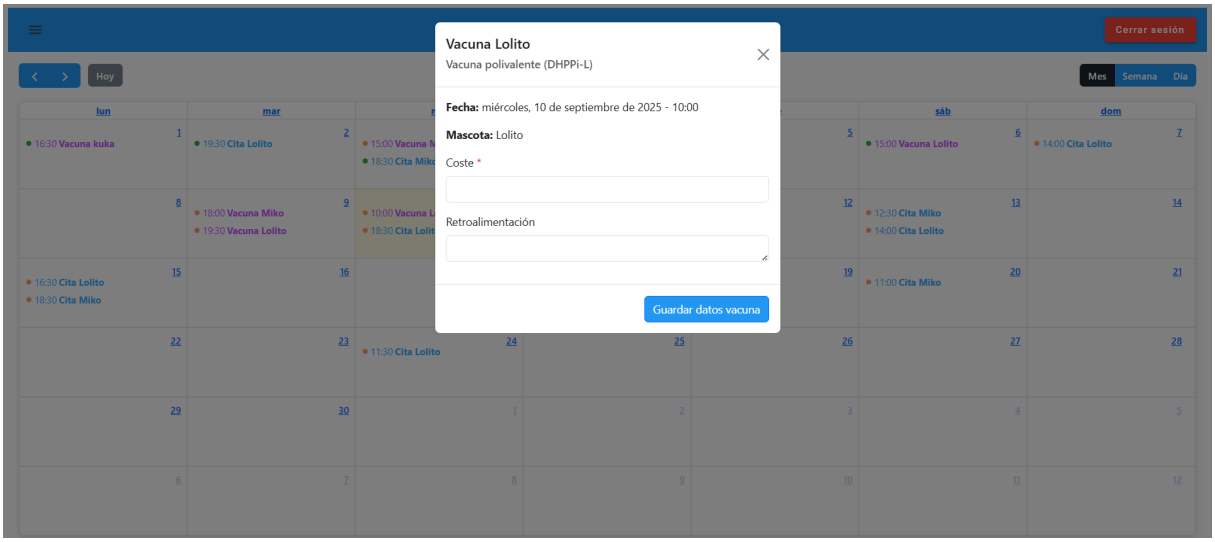

En cualquier página de la aplicación aparece en la parte superior una barra fija. Pulsar el botón rojo de la esquina superior derecha “Cerrar sesión” para cerrar la cuenta autenticada en la aplicación. Como consecuencia, la aplicación mostrará la pantalla de inicio de sesión.

### B.2.5. Gestión del calendario



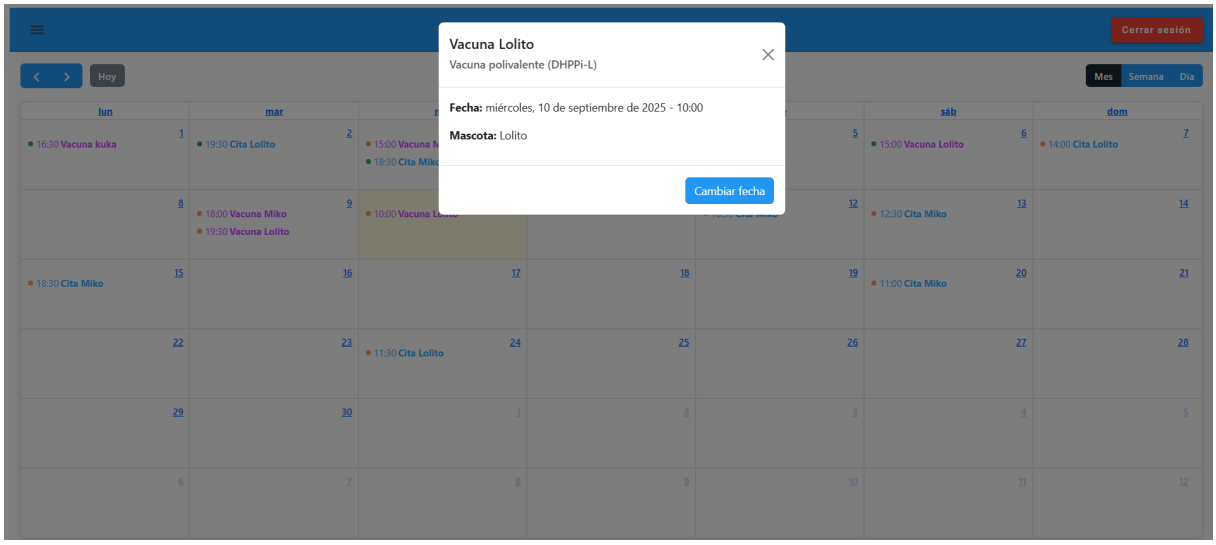

Tras iniciar sesión en la aplicación, la aplicación carga un calendario con todas las citas veterinarias y vacunas confirmadas del veterinario con sus clientes.

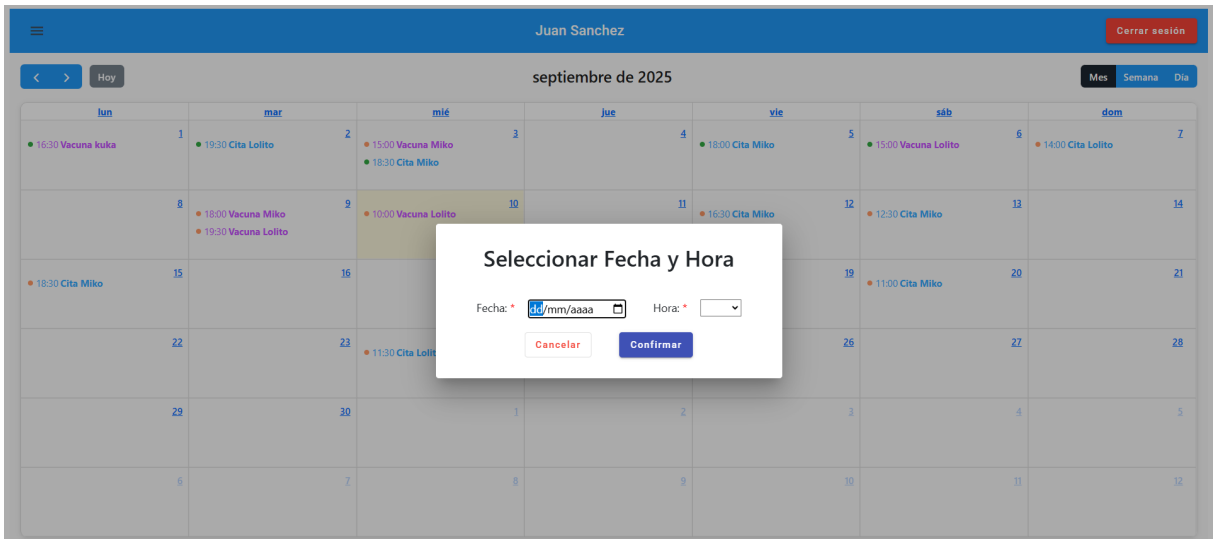
### Rellenar datos de citas y vacunas




Una vez se ha producido una cita con un animal, o bien se le ha administrado una vacuna, el veterinario puede hacer click en dicha cita o vacuna en el calendario y rellenar los distintos campos que aparecen en pantalla. Finalmente, pulsar el botón azul “Guardar datos cita” de la esquina inferior derecha para dejar guardada esa información en la aplicación., estando visible en cualquier momento seleccionando el evento en el propio calendario.

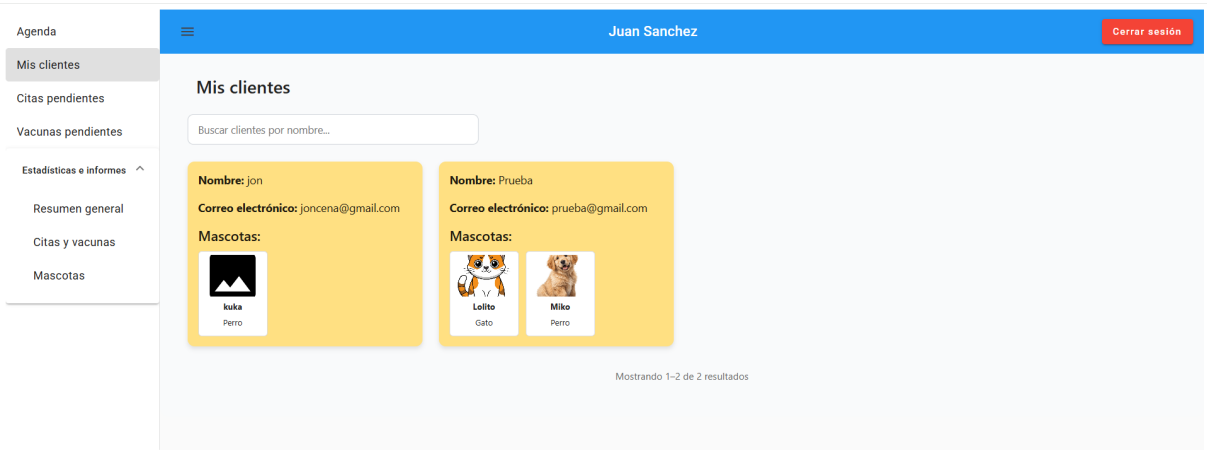
### Cambiar fecha de citas y vacunas

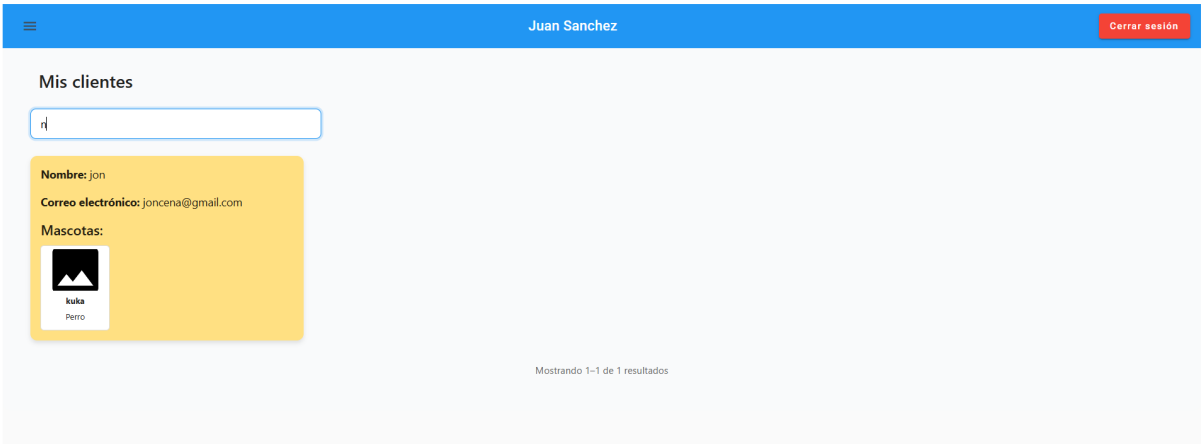



Si todavía no ha pasado la fecha de una cita o vacuna confirmada, es posible cambiarla. Para ello, hacer click en el evento en el calendario. En la ventana modal que se abrirá, pulsar el botón azul “Cambiar fecha”. Aparecerá otra ventana para seleccionar la nueva fecha y hora de la cita o vacuna. Finalmente, confirmar los cambios pulsando el botón “Confirmar”.

### B.2.6. Ver clientes



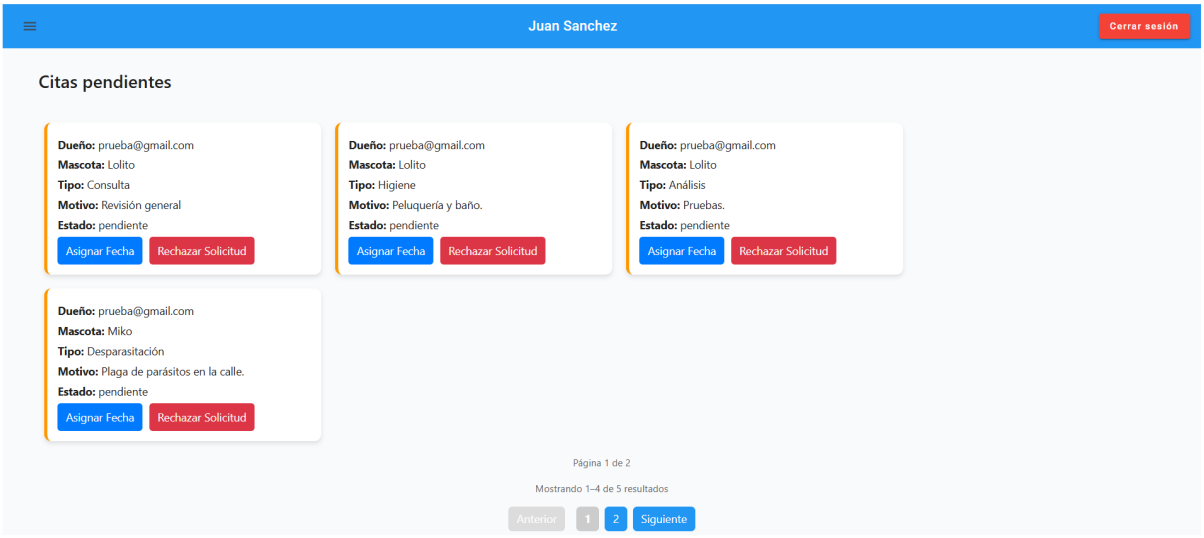



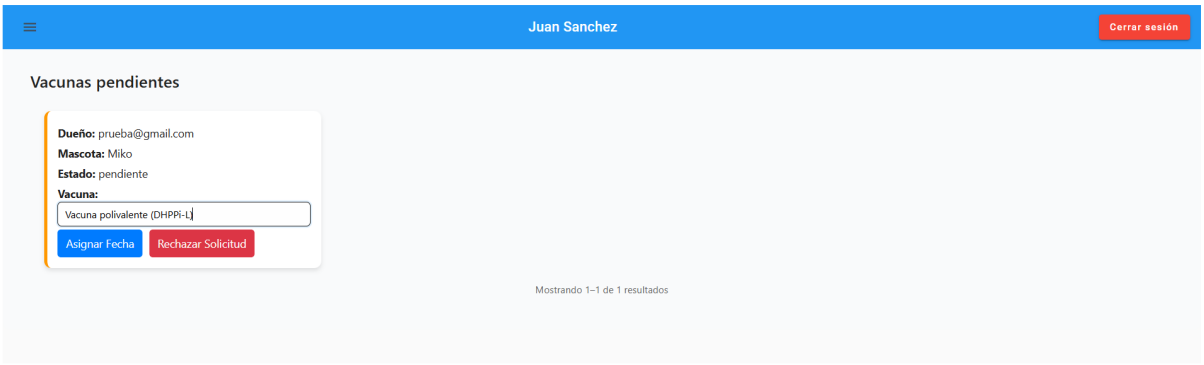

En el menú lateral desplegable desde el botón con tres líneas horizontales en la esquina superior izquierda, pulsar sobre la opción “Mis clientes”. A continuación, la aplicación mostrará los clientes de cuatro en cuatro, siendo posible pasar de página manualmente utilizando los botones numerados de la parte inferior.

También es posible filtrar los clientes por nombre, de manera que al introducir una cadena de texto en ese campo sólo se mostrarán los clientes cuyo nombre contenga el texto introducido.

### B.2.7. Gestión de solicitudes de citas y vacunas

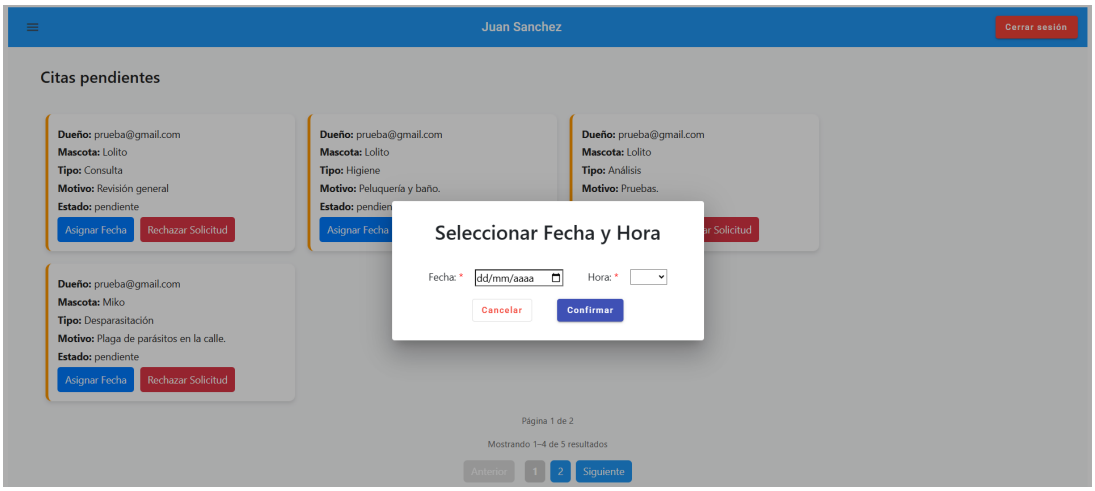
#### Ver citas y vacunas pendientes

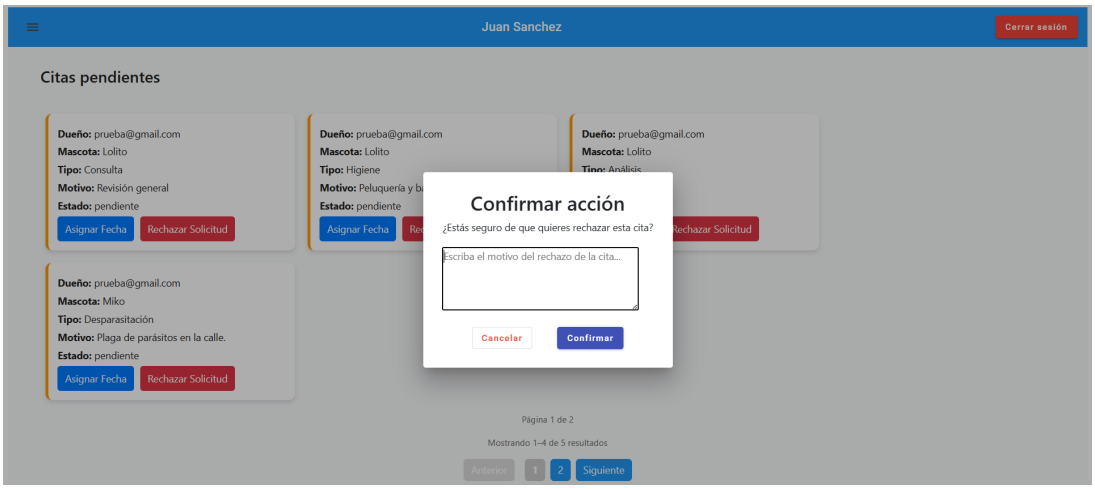



A través de las opciones “Citas pendientes” y “Vacunas pendientes” del menú lateral de navegación se puede acceder a las pantallas que muestran las solicitudes de citas y de vacunas pendientes de confirmación que han realizado los clientes.

### Confirmar/Rechazar citas y vacunas pendientes




Las solicitudes pueden ser aceptadas o rechazadas por el veterinario. Para aceptarla, pulsar el botón azul “Asignar Fecha”. Entonces, la aplicación abrirá una ventana modal con los campos para seleccionar la fecha y hora de la cita/vacuna. Para aceptarla, simplemente pulsar el botón rojo “Rechazar solicitud” y pulsar el botón “Confirmar”.

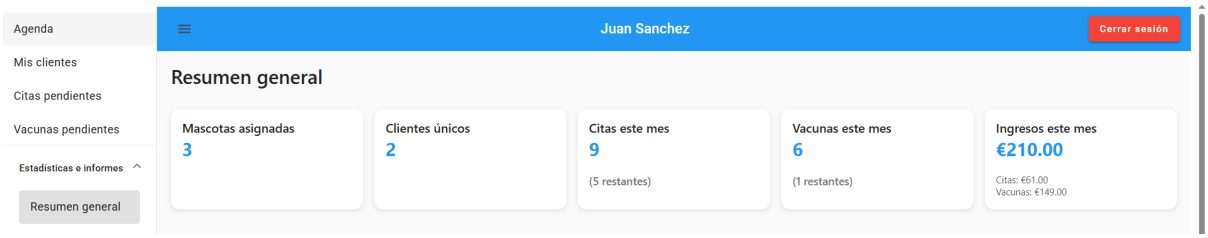
Las solicitudes de vacunas funcionan de manera idéntica a las de las citas, con la única direncia que antes de aceptarlas el veterinario puede indicar el nombre de la vacuna que se le va a administrar al animal en la fecha y hora que seleccione.

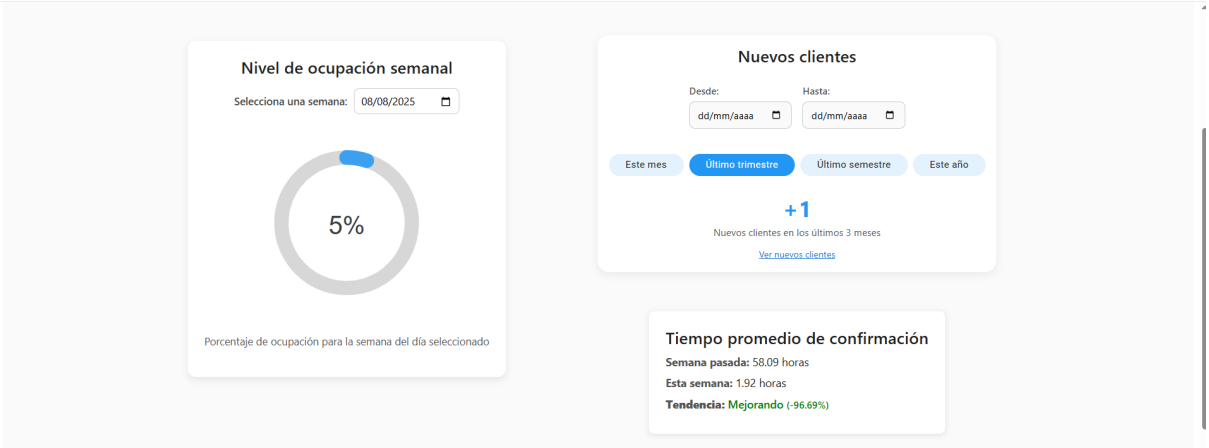
Las solicitudes de citas y vacunas se muestran ordenadas por fecha en la que se realizó la solicitud, de manera que las que más tiempo llevan sin ser respondidas aparecen las primeras, siendo las últimas las más recientes. De esta forma, el veterinario puede ir confirmándolas o rechazándolas siguiendo un orden.

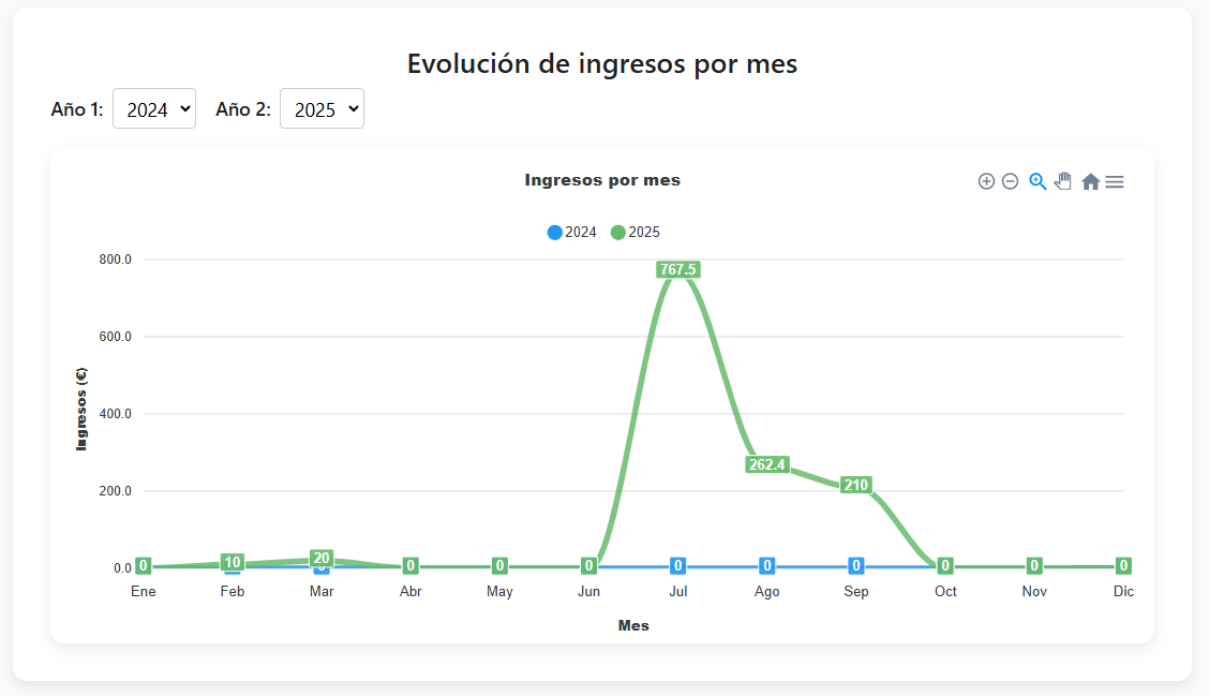
### B.2.8. Panel de control y estadísticas

Accesibles distintas pantallas a través de los botones “Resumen general“, “Citas y vacunas“ y “Mascotas“ dentro del submenú “Estadísticas e informes“ del menú lateral. En ellas se puede seleccionar distinta información en los distintos filtros disponibles, de manera que los datos que presentan los distintos componentes estadísticos cambian en función de los datos introducidos.

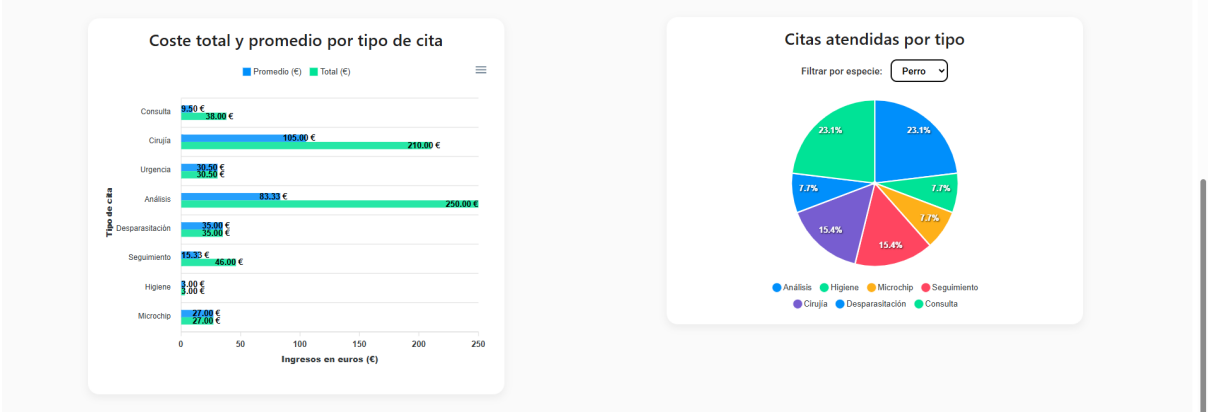
## Resumen general

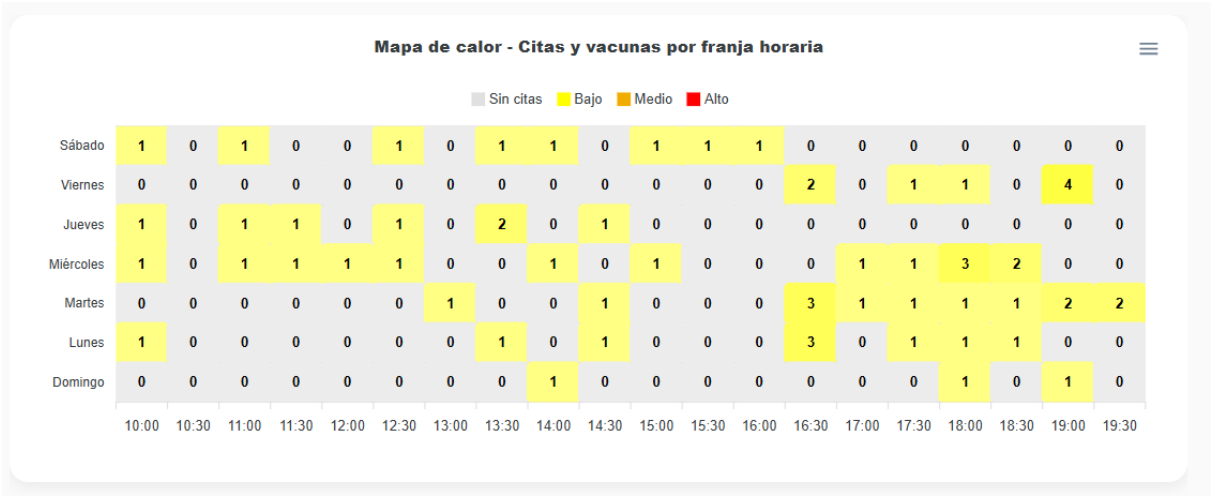


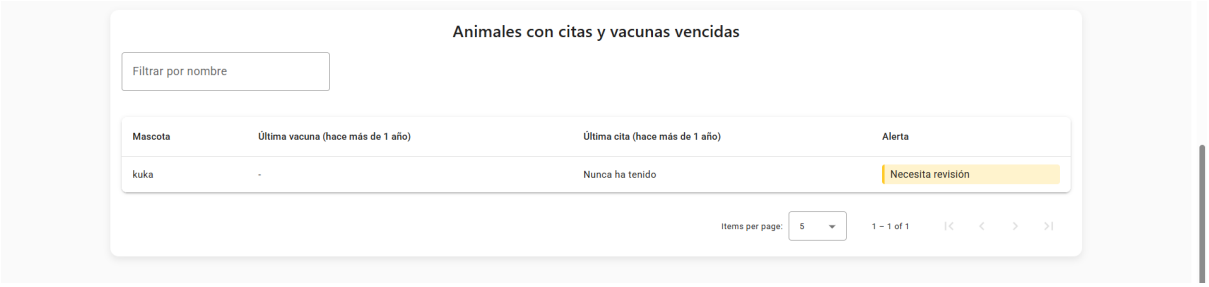
# Citas y vacunas

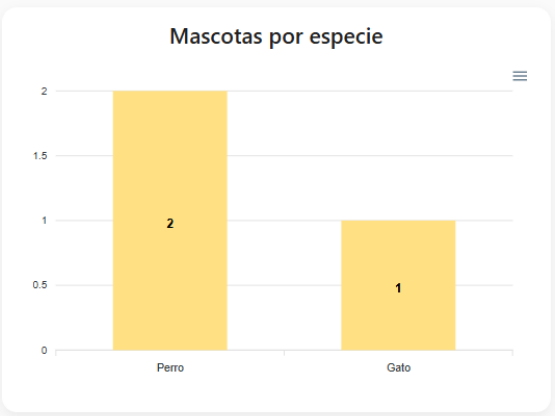



# Mascotas


Agenda
Mis clientes
Citas pendientes
Vacunas pendientes
Estadísticas e informes
Resumen general
Citas y vacunas
Mascotas
Juan Sanchez
Cerrar sesión
Mascotas frecuentes
Últimos 3 meses
Últimos 6 meses
Último año
Últimos 2 años
Filtrar por especie:
Todas
Miko
Visitas: 34
Lolito
Visitas: 16
Top 5 mascotas con más citas
Miko
21
Lolito
11
0
5
10
15
20
25


Animales con citas y vacunas vencidas
Filtrar por nombre
Mascota
Última vacuna (hace más de 1 año)
Última cita (hace más de 1 año)
Alerta
kuka
-
Nunca ha tenido
Necesita revisión
Items per page:
5
1 – 1 of 1


Mascotas por sexo
Total
3
33.3%
66.7%
Machos
Hembras


Mascotas por especie
2
1.5
1
0.5
0
2
1
Perro
Gato

UNIVERSIDAD DE MÁLAGA | uma.es

E.T.S de Ingeniería Informática
Bulevar Louis Pasteur, 35
Campus de Teatinos
29071 Málaga

E.T.S. DE INGENIERÍA INFORMÁTICA