Trabajo de Fin de Grado

Pawfind Pal gestión y búsqueda de mascotas

TITULACIÓN: Grado en Ingeniería Informática

AUTOR: Marcos Ismael Medina Castellano

TUTORIZADO POR:
María Dolores Afonso Suárez

Fecha Mayo 2024

Agradecimientos

Quiero agradecer a todos los profesores que he tenido, tanto en la universidad como fuera de ella, en especial a Pino Mendoza quien cuando estaba terminando la Formación Profesional del Ciclo Desarrollo de Aplicaciones Multiplataforma me animó a intentar el primer año de carrera. Sin esa conversación que tuvimos al final de la clase, nunca me lo hubiera planteado. También me gustaría hacer una mención a mi tutora de este TFG, Marilola Afonso por su amabilidad, tiempo, paciencia y ánimos. Gracias por guiarme en este proceso y por todos los consejos.

Y por último pero no menos importante, agradecer a mi familia y amigos, sobretodo a Alejandro Schmid y Rubén Tejera quienes han compartido conmigo su conocimiento y experiencia en el desarrollo del software, siempre con paciencia y amabilidad con el propósito de que mejore en cada momento. Y en especial, a mi madre, la persona que siempre ha estado dándome ánimos, apoyándome y acompañándome en este largo camino, gracias por siempre creer en mí.

Resumen

Pawfind Pal es una aplicación de búsqueda y gestión de animales que reúne a los amantes de las mascotas en un solo lugar, donde pueden almacenar toda la información médica y personal de sus queridos animales. Además, ofrece la posibilidad de adquirir un dispositivo hardware que permite localizar a la mascota en todo momento, brindando tranquilidad en caso de que se pierda. Los veterinarios también tienen acceso a la aplicación, lo que les permite consultar los datos de las mascotas durante las visitas y crear informes médicos. De esta manera, se puede mantener todo el historial clínico digitalizado y accesible en cualquier momento para ambos.

Abstract

Pawfind Pal is a pet search and management application that brings pet lovers together in one place, where they can store all the medical and personal information of their beloved animals. Additionally, it offers the option to acquire a hardware device that allows you to locate your pet at any time, providing peace of mind in case your pet goes missing. Veterinarians also have access to the application, enabling them to review pet data during visits and create medical reports. This ensures that the entire clinical history is digitized and accessible at any time for both parties.

Índice general

Índice de figuras

Índice de Algoritmos

Capítulo 1

Introducción

Pawfind Pal gestión y búsqueda de mascotas nace con la idea de solventar varios problemas que tienen los dueños de mascotas, uno de ellos es el miedo de un día llegar a casa y no encontrar a nuestra mascota, otro es la dificultad que implica la gestión del historial médico de las mascotas, poder consultar los datos de las citas clínicas y de cualquier otro aspecto concerniente a su salud.

Incluso para los veterinarios, puede ser complicado gestionar adecuadamente y acceder a la información completa de nuestras mascotas, especialmente si no se visita la misma clínica. Por lo general, los datos suelen quedar almacenados localmente en la clínica que se frecuenta, lo que dificulta la continuidad del acceso a la información médica.

Cada vez que deseamos acudir a una consulta médica diferente, nos enfrentamos a la necesidad de recordar todas las citas anteriores, lo cual no es una tarea sencilla. La dificultad radica en que podemos olvidar detalles importantes sobre el historial médico de nuestra mascota.

Afortunadamente, con la introducción de la aplicación Pawfind Pal, todos estos inconvenientes se solucionan. Esta aplicación ofrece acceso a dos tipos de usuarios:

1. Veterinarios
2. Propietarios de mascotas

De esta manera, independientemente del veterinario que atienda a nuestra mascota, siempre que se utilice Pawfind Pal, los datos médicos estarán disponibles. Tanto los veterinarios como los propietarios de mascotas podrán consultar en cualquier momento el historial médico completo. Esto incluye la posibilidad de revisar las citas anteriores, los medicamentos recetados y las intervenciones realizadas a la mascota.

Además, Pawfind Pal ofrece la opción de adquirir un dispositivo localizador por separado. Con este dispositivo, se puede conocer en todo momento la ubicación real de la mascota, eliminando así el temor de llegar a casa y descubrir que nuestra querida mascota no está

presente.

1.1. Elementos que componen el proyecto

Para llevar a cabo el proyecto de manera satisfactoria, se han desarrollado tres componentes fundamentales:

API - Desarrollo del Backend: La API actúa como el cerebro del sistema, gestionando la comunicación entre la interfaz de usuario y la Base de Datos. Este desarrollo backend se encarga de conectar y orquestar las operaciones de la Base de Datos, asegurando que la información fluya de manera eficiente y segura entre el servidor y las aplicaciones.

App iOS - Interfaz Móvil Nativa: La aplicación iOS ofrece una experiencia de usuario intuitiva y accesible, diseñada específicamente para usuarios de dispositivos Apple. Esta interfaz móvil permite a los distintos usuarios interactuar con las funcionalidades del sistema de manera nativa, brindando una experiencia fluida y adaptada a los requisitos de la plataforma iOS.

Localizador - Prototipo de Hardware: Como un componente esencial para la ubicación y seguimiento de mascotas, se ha desarrollado un prototipo de localizador utilizando un Arduino UNO y un módulo SIM900. Este dispositivo innovador permite la geolocalización en tiempo real, ofreciendo una solución práctica para monitorizar la ubicación de las mascotas mediante la integración de Hardware y Software.

1.2. Objetivos

- Desarrollar un prototipo hardware que funcione como localizador.

- Diseñar una app móvil que permita: al rol usuario, gestionar los datos de sus masctoas, así como ver su ubicación y los datos médicos; y al rol veterinario, poder hacer un Create Read Update and Delete de sus clientes y un Create Read Update and Delete de las citas médicas y el resto de información asociada a la mascota del cliente.

- Desarrollar una API que es consumida por el prototipo Hardware y la app móvil para el intermacbio de información.

- Implementar una Base de Datos donde se almacenarán los datos del sistema.

1.3. Atributos de cada elemento

1.3.1. API

El objetivo de la API será exponer unos endpoints para en todo momento poder tener acceso a los datos almacenados en la Base de Datos, además ofrece ciertos niveles de seguridad como JWT y el uso de criptografía en las contraseñas, con la idea de tener una capa extra de robustez y fiabilidad.

1.3.2. App iOS

La aplicación móvil intenta albergar las necesidades de dos tipos de usuarios bien definidos:

1. Usuarios propietarios de las mascotas.
2. Veterinarios profesionales en la salud de las mascotas.

Para cada rol se cumple una serie de objetivos:

Usuario:

1. Create Read Update and Delete de sus mascotas.
2. Asociar dispositivo localizador.
3. Modificar los datos personales del usuario.
4. Poder ver la ubicación de la mascota en tiempo real.
5. Ver el historial clínico de la mascota.

Veterinario:

1. Asociar usuarios como mis clientes.
2. Eliminar a un usuario como mi cliente.
3. Create Read Update and Delete de citas médicas a las mascotas de mis clientes.

1.3.3. Localizador

Se desarrolla un dispositivo hardware que comunica con la API para emitir la ubicación en tiempo real del dispositivo, para que el dueño de la mascota pueda desde el cliente visualizar de forma rápida la posición de su mascota.

Hardware

1. Arduino UNO
2. Módulo SIM900
3. Tarjeta sim
4. Adaptador de corriente 12v - 1A

Una vez completados todos los componentes mencionados anteriormente, contaremos con una solución integral que aborda múltiples desafíos enfrentados tanto por los dueños de mascota como por los profesionales de la salud veterinaria. Esta solución no solo facilita la interacción entre propietarios y veterinarios, sino que también centraliza toda la información relevante sobre los animales en un único lugar, proporcionando un acceso rápido y eficiente a los datos esenciales. Con esta herramienta, se mejora significativamente la gestión y el bienestar de las mascotas, simplificando la vida de quienes se encargan de su cuidado y atención.

Capítulo 2

Estado actual y objetivos

El proyecto se considera un **Producto mínimo viable**, debido a que en este momento ya se han desarrollado los tres componentes principales que se mencionaron previamente. Estos componentes no solo están implementados, sino que también están interconectados entre sí, lo que permite que trabajen en conjunto de manera efectiva.

Además, estos componentes cumplen con todas las funcionalidades básicas que se especificaron en los objetivos del proyecto. Ofreciendo un conjunto básico pero funcional de características esenciales que permiten a los usuarios comenzar a utilizarlo y beneficiarse de sus capacidades.

2.1. Objetivos

1. Desarrollar un prototipo hardware que funcione como localizador
2. Diseñar una app móvil que permita: al rol usuario, gestionar los datos de sus mascotas así como ver su ubicación y los datos médicos; al rol veterinario, poder hacer un Create Read Update and Delete de sus clientes y un Create Read Update and Delete de las citas médicas y el resto de información asociada a la mascota del cliente.
3. Desarrollar una API que es consumida por el prototipo Hardware y la app móvil para el intercambio de información.
4. Implementar una Base de Datos dondse se almacenarán los datos del sistema.

El funcionamiento actual para una demo, requiere exponer la API bien sea abriendo un puerto en nuestro router o publicándola en un servidor que se encuentre expuesto en internet.

2.2. Añadidos a los objetivos

Durante el desarrollo de los objetivos previstos en el TFT01, llegué a la conclusión de que hay ciertos aspectos que no se abordaron con la profundidad necesaria. A medida que avanzaba en la implementación de las metas establecidas, fue evidente que estos elementos requerían una atención más detallada para asegurar un producto más robusto y de mayor calidad.

Considerando esta observación, se decide implementar estos aspectos adicionales. La intención detrás de esta decisión es garantizar que el producto no solo cumpla con los requisitos básicos, sino que también ofrezca una experiencia más completa y satisfactoria para los usuarios. La inclusión de estos aspectos adicionales contribuirá significativamente a la mejora de la solidez y calidad del producto.

A continuación, se enumeran estos aspectos que se han identificado y que se implementarán para fortalecer el proyecto.

API

1. Se genera un token con JWT que es necesario para validar las peticiones mediante la cabecera Authorization.
2. Uso de Salt and Pepper para las contraseñas de los usuarios.
3. Las contraseñas se almacenan en forma de hash en la Base de Datos.
4. Arquitectura Hexagonal en todo el proyecto para mejorar la escalabilidad del mismo.
5. Patrón Builder para los modelos de datos y mejorar el manejo de los mismos.

iOS

1. MVC como patrón de diseño para mejorar la escalabilidad y robustez de la aplicación.
2. Optimización en la carga de imágenes redimensionando su tamaño antes de la carga.
3. Optimización de recursos utilizando hilos en segundo plano para la carga de elementos como las tablas y el mapa.

Capítulo 3

Competencias específicas y aportaciones del trabajo

3.1. Competencias específicas

En el desarrollo de este trabajo, se cubren algunas competencias que se detallan a continuación y cómo se cubren en este proyecto.

1. CI1: Desde el comienzo, se ha diseñado un sistema compuesto por múltiples partes, cada una con una función específica crucial para el funcionamiento global. En el proceso de desarrollo, se han integrado una serie de buenas prácticas, que abarcan desde metodologías de desarrollo ágil hasta técnicas avanzadas de optimización de recursos.
2. CI16: Incorporando buenas prácticas, patrones de diseño y patrones de arquitectura en diversas secciones del proyecto, se busca asegurar un desarrollo estructurado y eficiente. Estas prácticas no solo ayudan a mejorar la calidad del código y su mantenibilidad, sino que también garantizan que el sistema sea robusto y escalable.
3. TI6: Desarrollando una API robusta y una aplicación iOS que consuma dicha API, se facilita un intercambio de información eficiente y seguro. Ambos componentes están perfectamente integrados, ofreciendo una experiencia de usuario coherente y sin interrupciones.

3.2. Aporte a nuestro entorno

A nivel técnico aporta un conjunto de buenas prácticas y metodologías que intenta hacer el Software más flexible y escalable en el futuro. Además de intentar poner barreras de seguridad y autenticación con el objetivo de evitar filtrado de datos.

En una situación real, este proyecto tiene varias formas de obtener beneficios en función de los diferentes roles. A continuación se exponen las diferentes vías según el rol.

Usuario
Como usuario, se obtienen beneficios de las siguientes formas:

1. Con la compra del dispositivo localizador.
2. Suscripción del servicio localizador, ya que dicho dispositivo tiene integrada una tarjeta SIM para hacer las peticiones al servidor.
3. Características premium que se obtienen con una suscripción.

Veterinario
Como un veterinario se obtienen beneficios de las siguientes formas:

1. Suscripción premium que le permite tener acceso a herramientas avanzadas de la gestión del mismo.
2. Datos estadísticos avanzados sobre las tendencias de los clientes.

Capítulo 4

Desarrollo

4.1. Metodología

La metodología aplicada para el desarrollo del proyecto es SCRUM, utilizando la herramienta Trello para mantener organizados los diferentes tickets que se deben desarrollar y las características de cada Sprint. En la ilustración 4.1 se puede observar el estado final de los tickets desarrollados durante los diferentes Sprints que componen el desarrollo del proyecto.

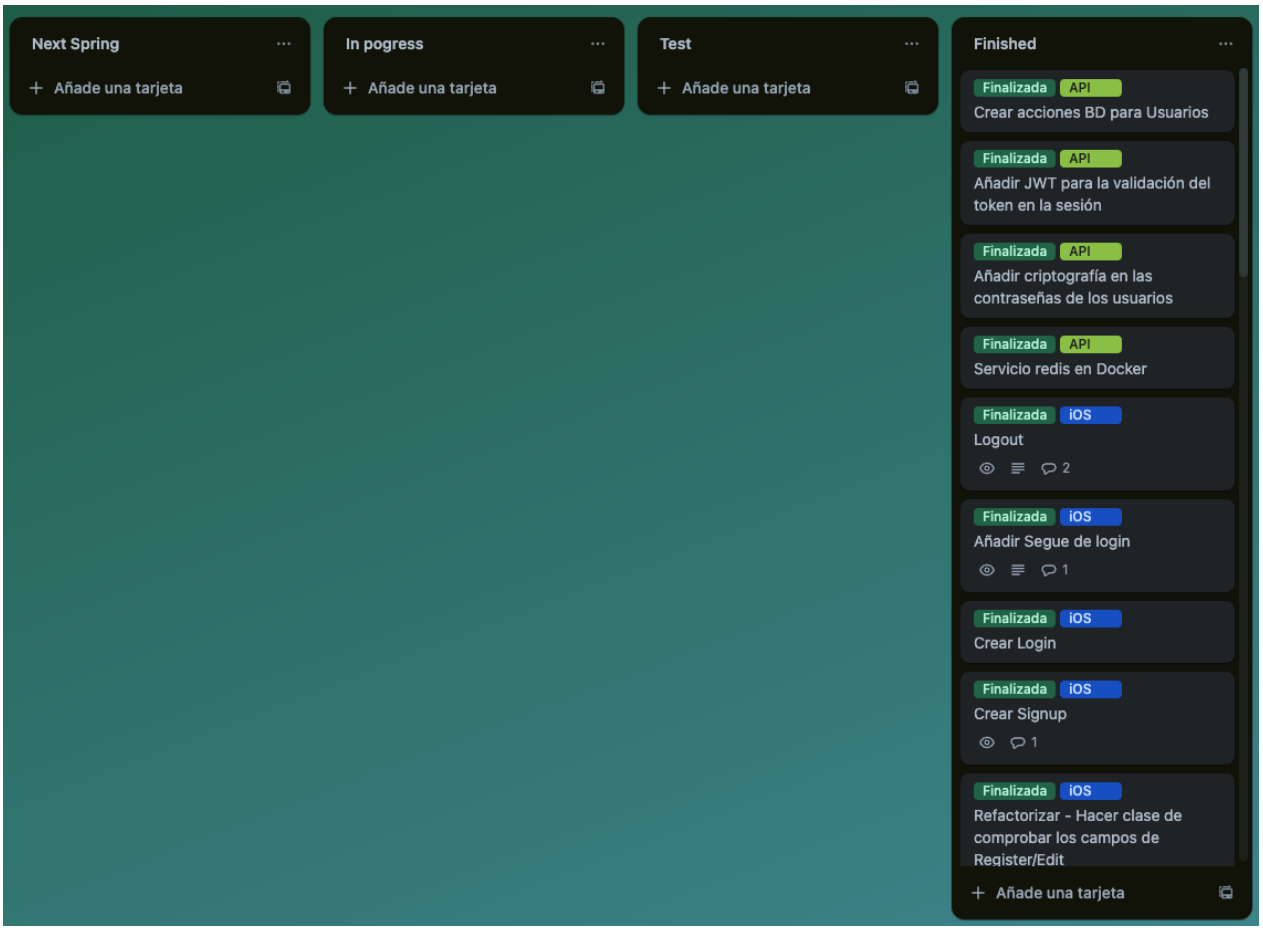


Ilustración 4.1: Tickets de los distintos Sprints desarrollados.

La duración de cada Sprint, no tuvo un tiempo previamente establecido, ya que algunos Sprints tenían más carga de desarrollo que otros, por lo que la duración ha sido variable. De forma concreta, se desarrollaron 4 Sprints que se describen en el siguiente apartado.

4.2. Análisis

Sprint 0: Se lleva a cabo un estudio de las diferentes tecnologías que serán aplicadas en el desarrollo de los módulos que componen el proyecto. Este estudio implica una búsqueda detallada de documentación relevante sobre cada tecnología, así como una investigación sobre cómo ejecutar ciertas acciones específicas que se implementan en el proyecto.

Además, se estudia el uso de un ORM para facilitar la gestión y accesos a la Base de Datos. Se ha utilizado Sequelize [20].

Sprint 1: Se dedicó al desarrollo de la API, lo que implicó la creación de todos los endpoints necesarios y la realización de pruebas de los endpoints así como de los métodos de

criptografía. Este proceso fue fundamental para garantizar la seguridad y funcionalidad de la API. Además, se implementó el ORM, lo que permitió una integración más fluida y eficiente entre la Base de Datos y la API.

Este fue el Sprint más largo debido a la gran carga de trabajo que requería. La complejidad de desarrollar y probar cada endpoint, junto con la implementación de métodos de criptografía requería gran parte del tiempo de desarrollo.

Sprint 2: Comenzó con el desarrollo de la aplicación móvil, centrado en el rol del usuario y en la implementación de todas las funciones necesarias definidas en el TFT01. Durante el Sprint, se trabajó intensamente para asegurarse de que la aplicación pudiera consumir eficazmente los endpoints expuestos por la API.

Se integraron todas las funcionalidades esenciales para el rol de usuario, lo que incluye tareas de autenticación, gestión de perfiles y visualización de datos. Integrando buenas prácticas para la optimización de recursos del dispositivo móvil.

Sprint 3: Se enfoca en el desarrollo del dispositivo localizador, una parte fundamental del proyecto. Simultáneamente, se completó la implementación del rol veterinario en la aplicación móvil.

Se dedicó tiempo para montar el prototipo con el hardware necesario, además de la implementación del código con las pruebas necesarias para hacer las conexiones a la API.

En paralelo, se finalizó la implementacióndel rol veterinario en la aplicación móvil. Esto incluyó la incorporación de funcionalidades específicas para los profesionales veterinarios, como la gestión de los perfiles de sus clientes, acceso al historial médico de sus mascotas y la capacidad de gestionar las citas médicas.

4.2.1. API

Para el desarrollo de la API la tecnología utilizada es ExpressJS [7] para facilitarnos la estructura y control de los endpoints.

Además, se estudian diversas tecnologías y patrones de diseño como:

1. REDIS [19]: Utilizada como Base de Datos virtual para el registro de nuevos usuarios.
2. Criptografía: Añade una capa de seguridad en las contraseñas de los usuarios con Salt and Pepper
3. Arquitectura Hexagonal [1]: Divide el proyecto en capas reduciendo así el acoplamiento entre capas y la implementación de las mismas.

4. Patrón Builder [2]: Facilitando la creación de los modelos de datos que se utiliza en el proyecto.
5. JWT [12]: Validando las peticiones con el campo 'Authorization' de las cabeceras HTTP.
6. Cookies [10]: Dando una sesión temporal para el proceso de registro de usuarios.
7. Sesiones [21]: Cuando se hace de forma exitosa un login, devuelve el token necesario para las demás peticiones.
8. Protocolo HTTP [9]: En los endpoints con el uso de los GET, POST, PUT, DELETE, OPTIONS.
9. Docker [5]: Usado como servidor donde se ejecuta Redis para el proceso de registro de usuarios.

Que en conjunto forman el desarrollo completo de la API. Entraremos en detalle en la sección de Implementación.

4.2.2. iOS

El objetivo era desarrollar la aplicación móvil de forma nativa en iOS, por lo que se dedicó tiempo al aprendizaje del lenguaje Swift [24], UIKit [27] y a la herramienta oficial de Apple, Xcode.

El desarrollo del mismo implica el estudio de:

1. Patrón MVC [15]: Patrón de diseño utilizado para desacoplar las distintas capas del proyecto y tener una mejor escalabilidad.
2. StoryBoards [23]: Definiendo las vistas de usuarios.
3. Hilos en segundo plano [25]: Técnica para optimizar los recursos en la carga de tablas y servicios que requieren actualización constante.
4. Implementación de mapas [14]: Donde se visualiza la ubicación de la masctoa y del usuario en tiempo real.
5. Ciclos de vida de las vistas [13]: Cargando los diferentes elementos de la vista en función del estado de la vista.
6. Almacenamiento en local con Core Data [3]: Base de Datos eficiente donde almacenar todos los datos de las mascotas del usuario.
7. Almacenamiento en local con UserDefaults [28]: Se almacenan en un fichero de texto los datos en formato JSON, ideal para cuando los datos no cambian mucho.
8. Optimización de recursos: Utilizando servicios creados para la optimización de la carga de imágenes y peticiones.

9. Protocols - Delegate [4]: Es el patrón de diseño que se utiliza en Apple para la comunicación de varias partes del código y poder notificar de forma asíncrona cuándo una tarea finaliza.
10. Implementación de QR: Para poder asociar fácilmente a los clientes y veterinarios y asignar los localizadores a las mascotas.
11. Desarrollo de servicios de validación: Con la finalizad de validar que los campos sean los correctos, tales como email o campos no vacíos.

4.2.3. Localizador

Para la implementación del prototipo, se analiza el manual oficial del módulo SIM900 para obtener información sobre cómo se establecen las comunicaciones HTTP, se conecta al APN.

4.2.4. Base de Datos

Para la Base de Datos se utiliza el ORM Sequelize [20], el cual nos ofrece una manera versátil de crear modelos, establecer relaciones y almacenar datos de forma sencilla y eficiente.

Para el desarrollo de la Base de Datos utilizando Sequelize [20], se llevó a cabo un estudio que incluye los siguientes aspectos:

1. La creación de modelos: Se diseñaron modelos que representan las diferentes entidades del sistema. Estos modelos fueron configurados para reflejar la estructura y los atributos necesarios de cada entidad, asegurando que la Base de Datos pudiera gestionar la información de manera coherente y organizada.
2. La asociaciones de los diferentes modelos de datos: Se establecieron relaciones entre los distintos modelos para garantizar que los datos pudieran interactuar adecuadamente entre sí. Esto incluyó la definición de asociaciones como uno a uno, uno a muchos y muchos a muchos, facilitando la integridad referencial y la consistencia de los datos.

Como podemos ver en la ilustración 4.2 se muestra el esquema del diseño de la Base de Datos, sus relaciones y campos.

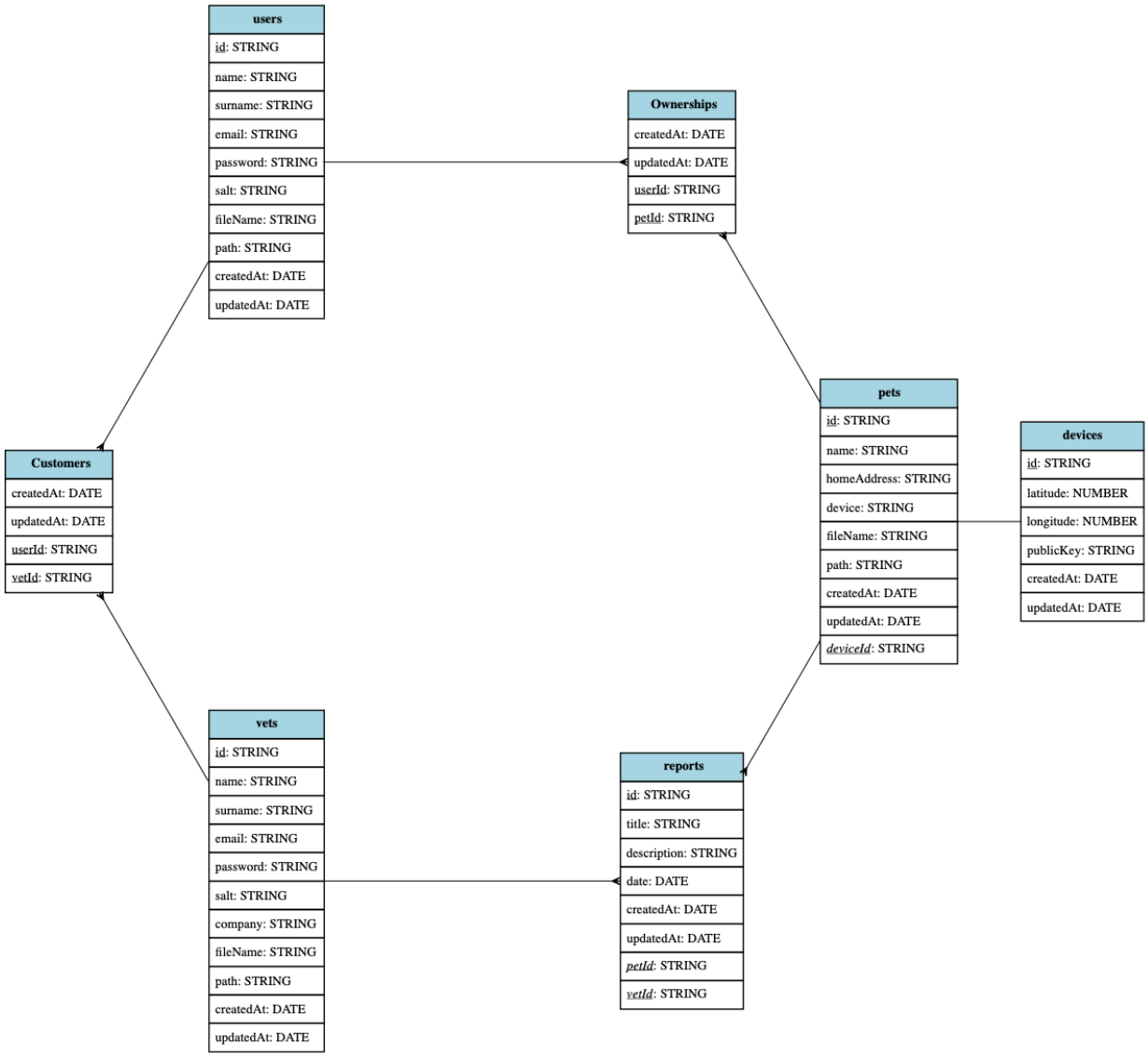


Ilustración 4.2: Diseño de la Base de Datos.

4.3. Diseño

Para el diseño se realizan los Mockups de la app iOS 4.3 haciendo uso de la aplicación 'Freeform' una herramienta propietaria de Apple. El diseño de la Base de Datos se hace también en dicha herramienta.

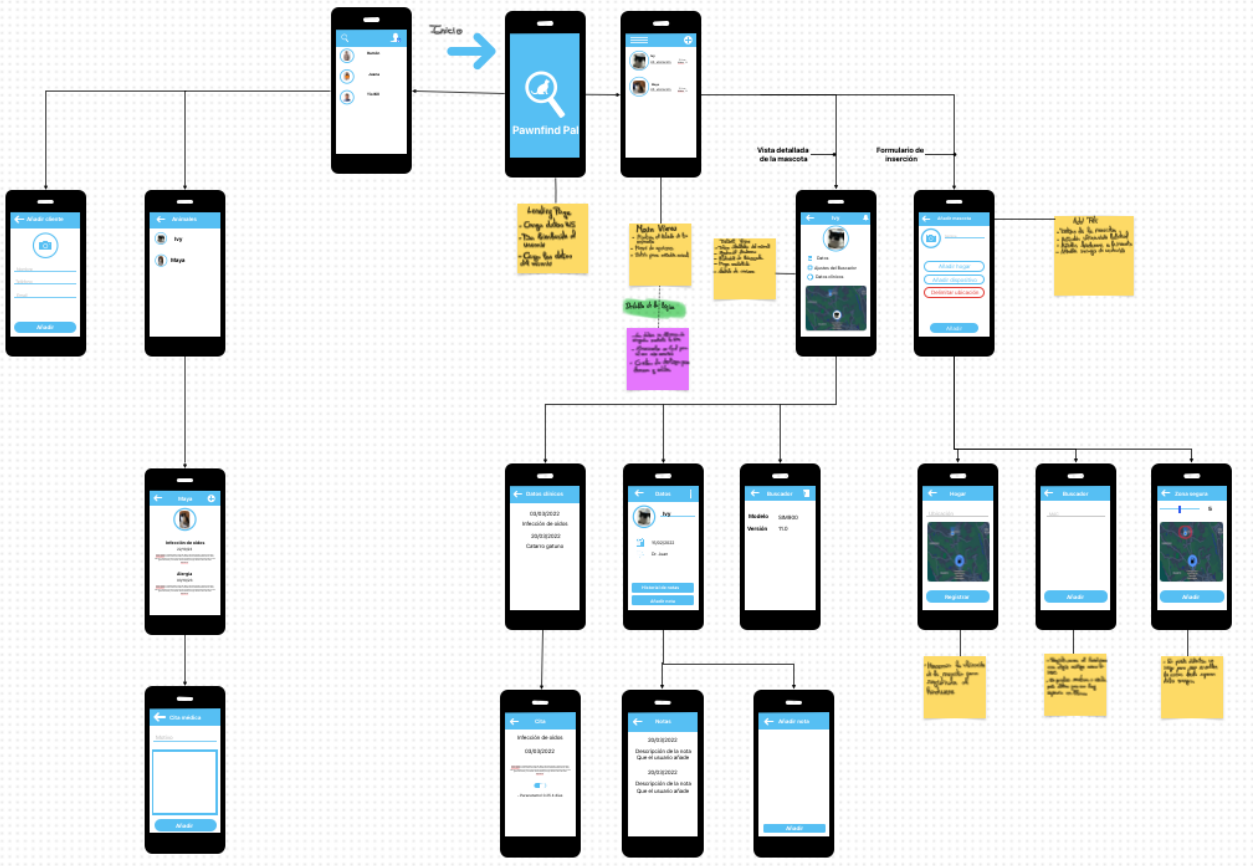


Ilustración 4.3: Mockup diseño aplicación iOS.

4.4. Estructura

Las distintas partes que conforman este proyecto mencionadas anteriormente se han desarrollado atendiendo a criterios de modularidad para facilitar la extensión de la aplicación en versiones futuras.

4.4.1. API

Como se menciona anteriormente, la tecnología en la que se basa la API es ExpressJS [7]. El IDE utilizado es WebStorm de IntelliJ. Se implementa la Arquitectura Hexagonal, en la ilustración 4.4 se observa la integración de cada capa de la arquitectura y la conexiones que existen entre ellas. A continuación se describe cada uno de los apartados:

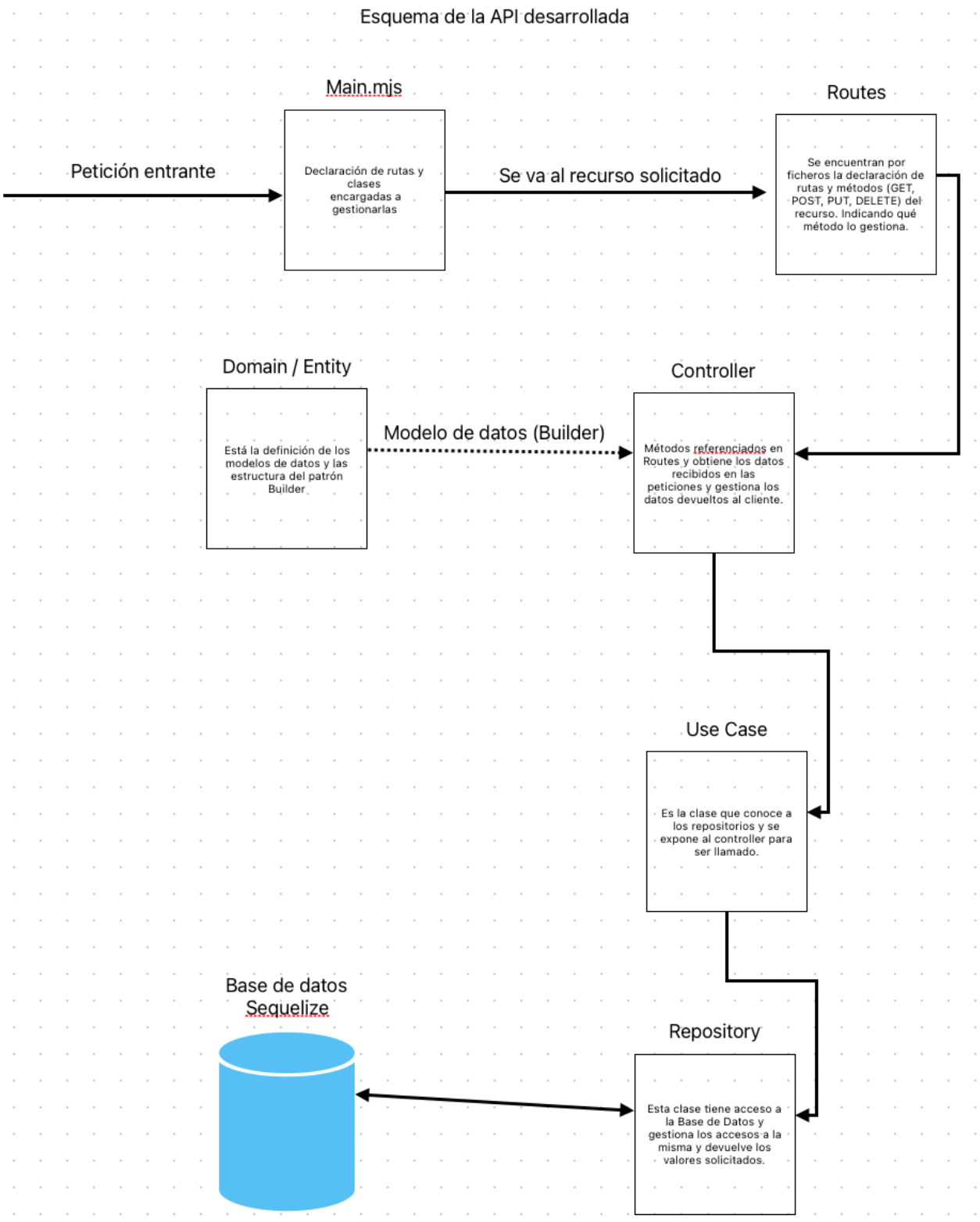


Ilustración 4.4: Esquema de la API.

Application
En dicha carpeta se encuentran todos los casos de uso de las diferentes entidades que componen la API, en dichos ficheros, se exponen las diferentes funciones que tiene permitido dicha entidad, de acuerdo con la Arquitectura Hexagonal, las entidades conocen a los 'repositorios' que son los que en última instancia tienen acceso a las conexiones con la Base de Datos.

Domain / Entity
Dentro de esta carpeta, se encuentran los diferentes modelos de datos que se usan en la API. Dentro de la misma se definen los distintos atributos que tiene cada entidad, así como los elementos necesarios para aplicar el Patrón Builder.

Infraestructura
En esta carpeta se encuentran otras dos:

1. Controller
2. Repository

Controller
Contiene los métodos que exporta y son referenciados en la carpeta Routes.

Dentro de esta carpeta, se encuentra la carpeta routes.

Routes
Aquí se encuentran todas las referencias a los métodos expuestos en la carpeta 'Controller', se indica qué tipo de Method se utiliza y el tipo y ruta que espera, asociado al método que se llamará para ser tratado.

Repository
Aquí se encuentran todos los ficheros de tipo 'Repositorio' para cada modelo de datos. Dichos repositorios, tienen los métodos necesarios y la lógica necesaria para acceder a las funciones Create Read Update and Delete de datos en las entidades correspondientes en el ORM Sequelize [20].

Además, el fichero 'model.mjs' define toda la estructura de datos y sus relaciones que utilizará Sequelize [20] para crear toda la estructura de datos del mismo.

config.mjs
Encontramos la configuración de ExpressJS.

db-conector.mjs

Define la conexión a la Base de Datos, indicando a sequelize [20] el tipo de dialecto de la Base de datos y la ruta de la misma.

main.mjs
En este fichero se define el objeto de Express, se inicia la Base de datos y también se enlazan las diferentes rutas con sus respectivos ficheros que se encuentran en 'routes'.

Además, se encuentra el endpoint de signups ya que debido a su complejidad al usar diferentes tecnologías es complicado de separar.

4.4.2. iOS

La aplicación móvil está estructurada siguiendo el patrón MVC, se implementan varios servicios para facilitar el uso de alertas, optimizar el renderizado de imágenes, servicios en segundo plano, validadores de campos. En la figura 4.5 podemos ver la forma que están estructuradas y se interconectan entre sí las diferentes capas que componen el proyecto.

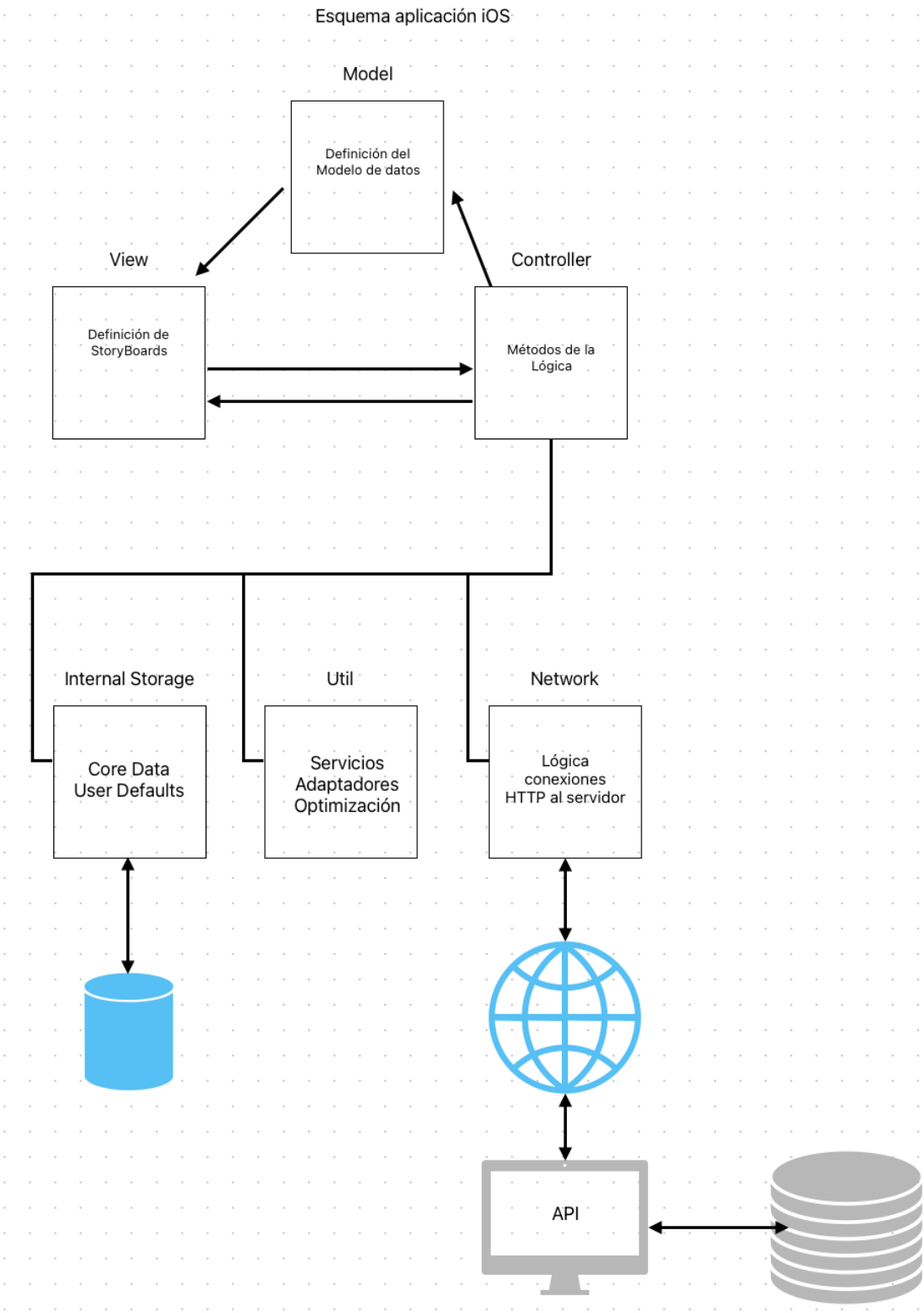


Ilustración 4.5: Esquema de la aplicación iOS.

Internal Storage
Para almacenar los datos de las mascotas, utilizamos CoreData, una forma fácil y eficaz de mantener en local una copia de los datos almacenados en el servidor.

Dentro de la carpeta, encontramos la clase 'PetStorage' donde se definen los distintos métodos que se pueden llevar a cabo, en este caso, un Create Read Update and Delete de los datos.

Util
Aquí se encuentran diferentes recursos que son necesarios en la aplicación, como servicios, adaptadores intermedios de los modelos para evitar el acoplamiento y clases encargada de la optimización de recursos.

Network
Se encuentra la clase encargada de hacer todas las peticiones a la API, para poder hacer las peticiones, nos apoyamos en AlamoFire 5.8.1, además existe una carpeta 'Data', que contiene los modelos de datos especificados como 'Decodable' para poder deserializar el JSON que nos devuelve la API.

Controller
Se encuentran todas las clases controladoras de las vistas, dichas clases registran las acciones de los diferentes elementos que se encuentran en la vista e interactúan con el Modelo y la Vista.

View
Se encuetran todos los archivos de las StoryBoards de Swift, por comodidad, se tienen cuatro elementos, aunque dentro se defienen las vistas relacionadas con ese apartado, identificadas y asociadas a su respectivo controlador.

Consta de:

1. LaunchScreen: LandingPage de la aplicación.
2. Main: Vista inicial para login o registro.
3. UserScreen: Conjunto de vistas relacionadas con el rol usuario.
4. VetScreen: Conjunto de vistas relacionadas con el rol veterinario.

Model
Se encuentran todas las clases Modelo de los datos que se utilizan en la aplicación, definiendo todos sus campos ya que se trabaja con OOP

4.4.3. Localizador

El localizador es simplemente un único fichero .ino de Arduino, ya que ahí se inicia y se lanza la rutina programada se explicará de forma detallada en la implementación, para tener una idea general del mismo, se puede observar el esquema de la Ilustración 4.6 4.6.

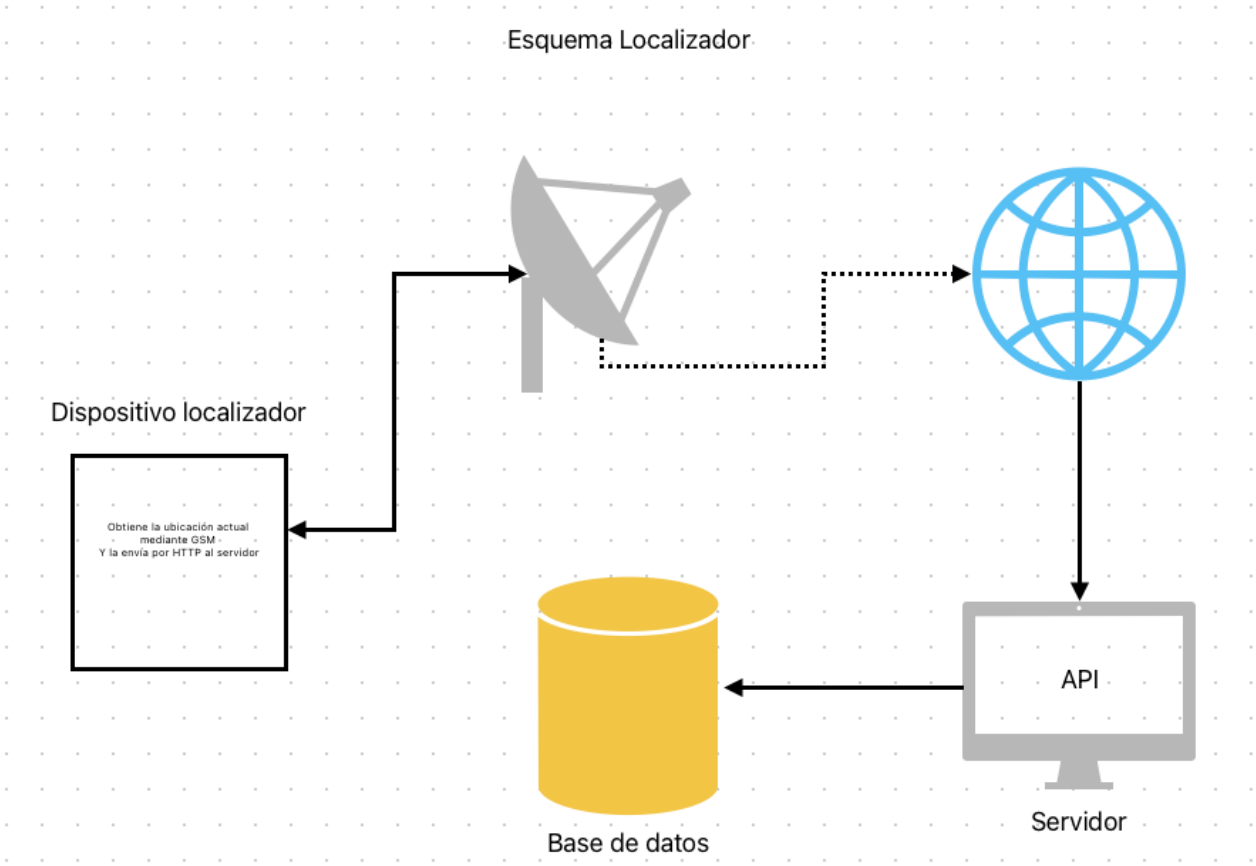


Ilustración 4.6: Esquema de la implementación del localizador.

4.4.4. Docker

En este proyecto se utiliza Docker para tener Redis desplegado, ya que como se explica en la implementación, es necesario en el proceso de Registro. El Docker contiene un contenedor 'redis-stack'.

4.4.5. Redis

Es una base de datos en memoria que se utilizará para el proceso de registro de los usuarios, permitiendo que el proceso de registro tenga una validez de 30 minutos mediante

Cookies, por lo que si se pierde conexión a internet, el proceso tendría vigencia para continuar en el estado que se quedó.

4.5. Implementación

En esta sección vamos a describir la implementación de cada apartado que se desarrolla en este trabajo, se explicará de forma detallada cada una de las partes que lo componen y el uso del mismo. Además se incluirán partes de código de las secciones más importantes del mismo.

4.5.1. Docker y Redis

El objetivo al usar 'Redis' es disponer de la versatilidad en el registro del usuario, pasando los datos poco a poco, ya que en una plataforma móvil, podemos quedarnos sin cobertura, por lo que redis hace una copia de la solicitud en memoria. A cambio envía una cookie de validación.

Mientras tanto, se hace todo el proceso de registro y una vez finalizado, se notifica y se registra el usuario en la Base de Datos.

Todo este conjunto de operaciones, se realiza por pasos utilizando las distintas cabeceras HTTP, para saber cuáles son las siguientes peticiones que se deben hacer.

Para aclarar un poco mejor, vamos a mostrar un esquema del flujo de peticiones en un registro de rol usuario. Ver la Ilustración 4.7.

```
> POST /signups/

< 200
  Location: /signups/0r8230rj29uf9823u42
  {"state": "in-progress"}

> GET /signups/0r8230rj29uf9823u42

< 200
  Link: </users/4i02k-3f90w4-09iuf0-230484>; rel="User"
  Link: </auths;user=4i02k-3f90w4-09iuf0-230484>; rel="Auth"

> OPTIONS /users/4i02k-3f90w4-09iuf0-230484

< 204
  Allow: OPTIONS, PUT

> PUT /users/4i02k-3f90w4-09iuf0-230484

  name: …
  surname: …


< 202
  Link: </signups/0r8230rj29uf9823u42>; rel="Signup"
  Link: </auths;user=4i02k-3f90w4-09iuf0-230484>; rel="Auth"

> OPTIONS /auths;user=4i02k-3f90w4-09iuf0-230484

< 204
  Allow: OPTIONS, PUT

> PUT

  email: …
  password: …

< 202
  Link: </signups/0r8230rj29uf9823u42>; rel="Signup"
  Link: </users/4i02k-3f90w4-09iuf0-230484>; rel="User"
  Link: </auths;user=4i02k-3f90w4-09iuf0-230484>; rel="self"
```

Ilustración 4.7: Ejemplo del flujo de peticiones para hacer un registro.

Para visualizar mejor cómo nos ayuda, vamos a registrar un nuevo usuario y veremos cómo se crea un elemento en Redis que posteriormente se queda almacenado en la Base de Datos.

Primero, vamos a registrar un usuario, por ejemplo directamente desde la app móvil. Ver Ilustración 4.8.

Ilustración 4.8: Registro de un nuevo usuario desde la aplicación móvil.

Una vez registrado, veremos que en Redis se nos crea un nuevo registro en la memoria virtual, podemos ver dicha Base de Datos con la apliación Redis Insight. Ver Ilustración 4.9.

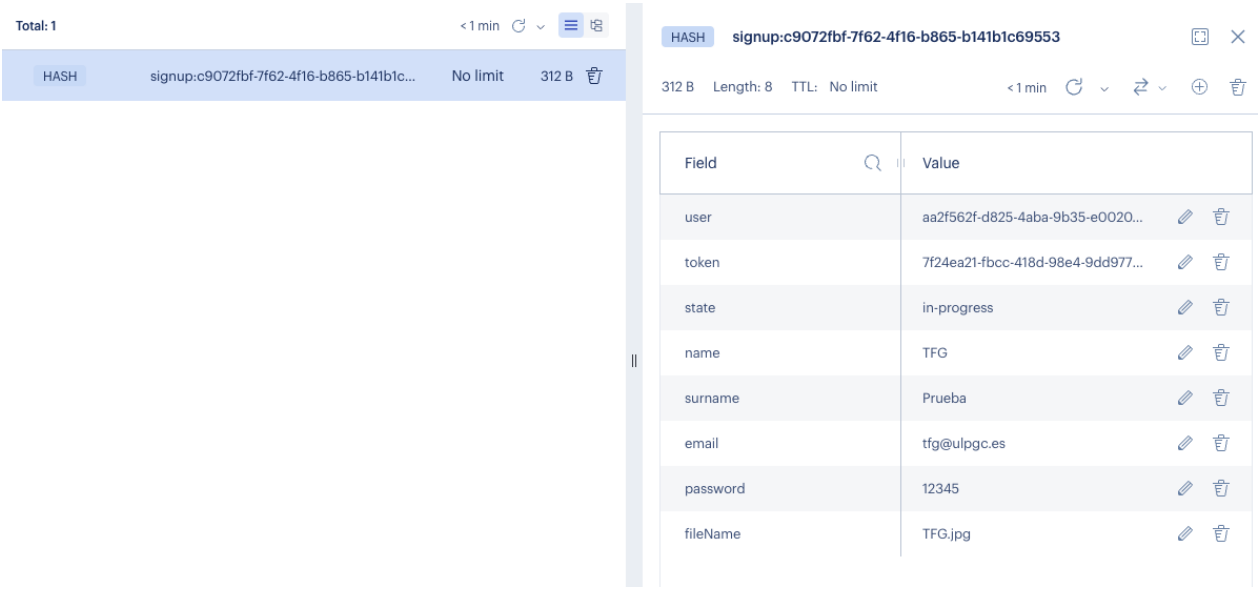


Ilustración 4.9: Datos en Redis del usuario registrado.

Se comprueba que el usuario también queda registrado en la Base de Datos. Ver Ilustración 4.10

Ilustración 4.10: Datos del registro almacenados en la Base de Datos.

4.5.2. API

Para el desarrollo de la misma, se sigue la Arquitectura Hexagonal y antes de continuar, debemos entender un poco el propósito de la misma y su funcionamiento.

Arquitectura Hexagonal

La idea detrás de dicha arquitectura, es la de permitir que una aplicación sea usada de la misma manera por los distintos clientes e independiente de las implementaciones que tiene, por lo que se promueve el desacoplamiento de las diferentes capas con el uso de interfaces. Con esto conseguimos que, si hoy por ejemplo la Base de Datos que estás utilizando es MySQL y toda la implementación y los requisitos del proyecto, se deben migrar a SQLite, simplemente se debe cambiar la parte del repositorio, ya que es la clase que tiene los métodos de cada

tecnología, pero fuera de esa capa, como lo que se exponen son interfaces, no se ven afectadas.

Si se siguiera un planteamiento diferente, este cambio implicaría el tener que cambiar las dependencias que existen en las demás capas, ya que el código estaría pensado para MySQL.

Cada capa conoce a la siguiente pero no al contrario, por lo que las capas saben que alguien las llama pero sin entrar en detalle de las implementaciones internas. Ver Ilustración 4.11.

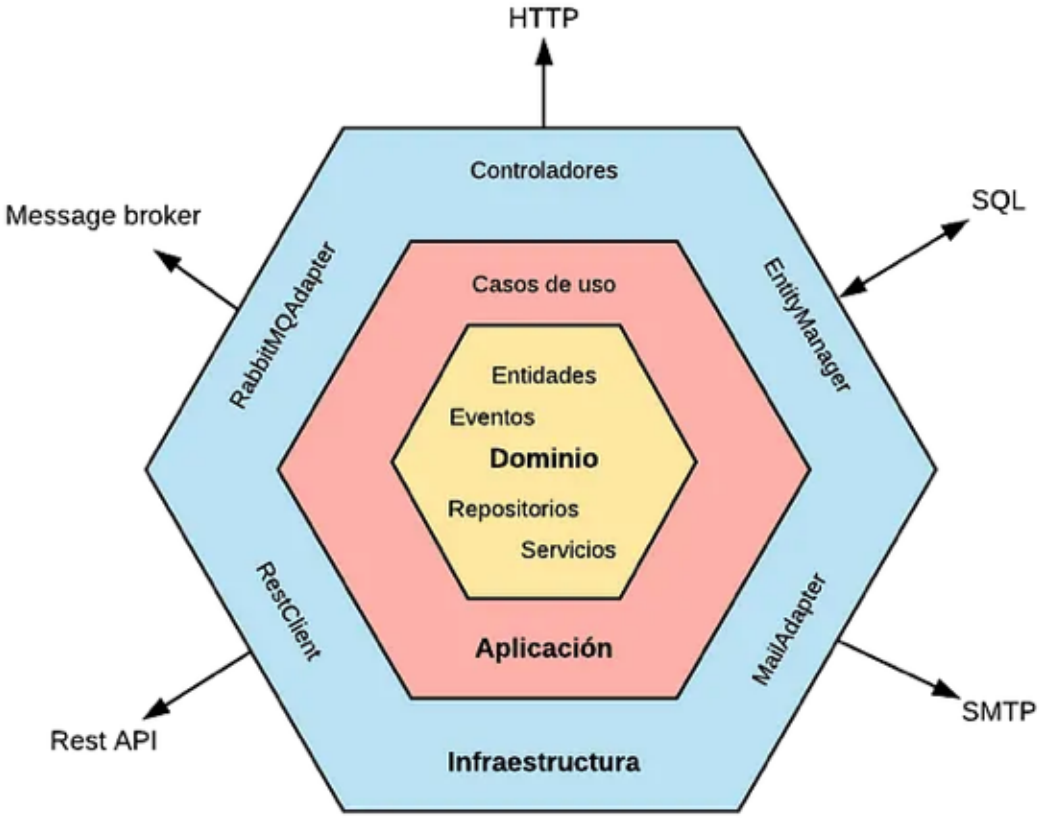


Ilustración 4.11: Ejemplo visual de la Arquitectura Hexagonal, [obtenida de la entrada medium de Edu Salguero [1].

Una vez explicado el concepto de Arquitectura Hexagonal, vamos a mostrar la implementación del mismo en la API.

Como ejemplo se muestra todo el proceso para obtener la ubicación de un dispositivo localizador.

Proceso para obtener la ubicación de un localizador

El fichero 'main.mjs', tendremos la declaración de la ruta y qué clase es la que responde a ellos, exponiendo los endpoints y los HTTP Methods correspondientes:

```
app.use(express.json())
app.use(cookieParser())
app.use('/users', userRoutes)
app.use('/devices', deviceRoutes)
app.use('/pets', petRoutes)
app.use('/vets', vetRoutes)
app.use('/reports', reportRoutes)
```

Algoritmo 4.1: Asignación de la ruta y la clase que la gestiona.

La clase 'deviceRoutes.mjs' contiene los HTTP Methods que espera y a qué método del 'deviceController.mjs' debe llamar para la gestión.

```
import express from 'express';
import { getDeviceLocation, postDevicePet, putDeviceLocation } from "../deviceController.mjs";

const router = express.Router();
router.get('/:id/location', getDeviceLocation)
router.post('/:id/location', putDeviceLocation)
router.post('/:id/:petId', postDevicePet)
export default router;

...
```

Algoritmo 4.2: Contenido de la clase 'deviceRouter.mjs'

Una vez llega una petición de tipo GET a la ruta '/:id/location' es el método 'getDeviceLocation' de la clase 'deviceController.mjs' el encargado de capturarlo.

```
import e from "express";
import DeviceUseCase from "../../application/deviceUseCase.mjs";
import PetUseCase from "../../application/petUseCase.mjs";
import DeviceModel from "../../domain/entities/DeviceModel.mjs";
import { Devices, Pets } from "../repository/models.mjs";

const useCase = new DeviceUseCase()
export const getDeviceLocation = async (req, res) => {
    const device = await useCase.getDevice(req.params['id'])
    if(device != null) {
            res.json({
                    'latitude': device.latitude,
                    'longitude': device.longitude
```

```
            })
    }
}
```

Algoritmo 4.3: Método 'getDeviceLocation' encargado de devolver la ubicación del dispositivo.

En primer lugar, antes de nada, se debe obtener el dispositivo en cuestión, los casos de uso son los encargados de ello, con el 'id' que se obtiene mediante la URL, si se obtiene un objeto 'device' se procede a devolver en formato JSON la latitud y longitud del mismo.

Antes de que se envíen los datos, es el caso de uso el encargado de comunicar al 'repository' que se quiere obtener información. Para ello, veremos cómo se comporta el caso de uso.

```
import DeviceRepository from "../infraestructure/repository/deviceRepository.mjs";

export default class DeviceUseCase {
    repository = new DeviceRepository()
    async getDevice(id) {
            return await  this.repository.getDevice(id)
    }
 ...
```

Algoritmo 4.4: Método 'getDevice(id)' del caso de uso.

Dicho caso de uso, conoce la referencia al repository, que no es más que la clase que sí tiene acceso a la Base de Datos. Este método simplemente informa de que hay una petición.

En última instancia, es la clase 'deviceRepository.mjs' la que sí conoce la implementación de la Base de Datos, por lo que es la encargada de hacer la solicitud y devolver el valor, en forma de promesa.

```
import { Devices } from "./models.mjs";

export default class DeviceRepository {
    async getDevice(id) {
            return await Devices.findByPk(id)
    }

    async putDeviceLocation(...data) {
            return await Devices.update({
                    id: data[0].id,
                    latitude: data[0].latitude,
                    longitude: data[0].longitude
            }, {where: {id: data[0].id}})
    }

 ...
```

Algoritmo 4.5: El objeto Devices que hace referencia al modelo de Sequelize y sus acciones.

Esta estructura se mantiene en todo el proyecto de la API, aplicando la Arquitectura Hexagonal.

Antes de continuar con la implementación y detalles de código de la API, es necesario hablar de la estructura de datos que utiliza Sequelize [20].

Dicha estructura está definida en la clase 'model.mjs', donde se definen los atributos de cada modelo (que se verán reflejados en las tablas) así como las relaciones que existen entre ambos.

Modelo de datos Sequelize

```
import { DataTypes } from 'sequelize'
import { sequelize } from '../../db-conector.mjs'

export const Users = sequelize.define('users', {
  id: {
    type: DataTypes.STRING,
    primaryKey: true,
    allowNull: false
  },
  name: {
    type: DataTypes.STRING,
    allowNull: false
  },
  surname: {
    type: DataTypes.STRING,
    allowNull: true
  },
  email: {
    type: DataTypes.STRING,
    allowNull: false
  },
  password: {
    type: DataTypes.STRING,
    allowNull: false
  },
  salt: {
    type: DataTypes.STRING,
    allowNull: false
  },
  fileName: {
    type: DataTypes.STRING,
    allowNull: true
  },
  path: {
    type: DataTypes.STRING,
    allowNull: true
  }
})

export const Devices = sequelize.define('devices', {
  id: {
    type: DataTypes.STRING,
    primaryKey: true
  },
  latitude: {
    type: DataTypes.FLOAT,
```

```
      allowNull: false
    },
    longitude: {
      type: DataTypes.FLOAT,
      allowNull: false
    },
    publicKey: {
      type: DataTypes.STRING,
      allowNull: false
    }
  })

  export const Pets = sequelize.define('pets', {
    id: {
      type: DataTypes.STRING,
      primaryKey: true
    },

    name: {
      type: DataTypes.STRING,
      allowNull:false
    },

    homeAddress: {
      type: DataTypes.STRING,
      allowNull: false
    },

    device: {
      type: DataTypes.STRING,
      allowNull: true
    },
    fileName: {
      type: DataTypes.STRING,
      allowNull: true
    },
    path: {
      type: DataTypes.STRING,
      allowNull: true
    }
  })

  export const Vets = sequelize.define('vets', {
    id: {
      type: DataTypes.STRING,
      primaryKey: true,
      allowNull: false
    },
    name: {
      type: DataTypes.STRING,
      allowNull: false
    },
    surname: {
      type: DataTypes.STRING,
      allowNull: true
    },
    email: {
      type: DataTypes.STRING,
      allowNull: false
```

```
  },
  password: {
    type: DataTypes.STRING,
    allowNull: false
  },
  salt: {
    type: DataTypes.STRING,
    allowNull: false
  },
  company: {
    type: DataTypes.STRING,
    allowNull: false
  },
  fileName: {
    type: DataTypes.STRING,
    allowNull: true
  },
  path: {
    type: DataTypes.STRING,
    allowNull: true
  }
})

export const Reports = sequelize.define('reports', {
  id: {
    type: DataTypes.STRING,
    primaryKey: true,
    allowNull: false
  },
  title: {
    type: DataTypes.STRING,
    allowNull: false
  },
  description: {
    type: DataTypes.STRING,
    allowNull: true
  },
  date: {
    type: DataTypes.DATE,
    allowNull: false
  }
})

Users.belongsToMany(Pets, {through:'Ownerships'})
Pets.belongsToMany(Users, {through: 'Ownerships'})
Devices.hasOne(Pets)
Users.belongsToMany(Vets, {through: 'Customers'})
Vets.belongsToMany(Users, {through: 'Customers'})
Pets.hasMany(Reports)
Reports.belongsTo(Pets)
Vets.hasMany(Reports)
Reports.belongsTo(Vets)
```

Algoritmo 4.6: Definición y relación de los datos en Sequelize.

Cada objeto de dicho modelo, tiene asociado una definición de métodos para poder hacer las operaciones básicas de Create Read Update and Delete, además de métodos especiales en función de las asociaciones que se han creado. (Documentación Sequelize [20]).

Actualizar campos de mascota y asociar / devincular 'Device'

Una de las características que nos ofrece Sequelize [20], es que ya tiene métodos para tratar las relaciones, por lo que simplemente se debe llamar. En el ejemplo expuesto a continuación, se muestra cómo el 'repository' de 'PetRepository.mjs' en función de los datos de entrada, actualiza los datos de la mascota y asocia o desactiva un dispositivo a la misma, el flujo previo del programa, es el mismo mostrado en el ejemplo anterior.

```
async postPet(data, user) {
            const deviceUseCase = new DeviceUseCase()
            const petUseCase = new PetUseCase()
            const device = await deviceUseCase.getDevice(data.device)
            const oldPet = await petUseCase.getPet(data.id)

            if(device != null) {
                    const useCase = new PetUseCase()
                    const pet = await useCase.getPet(data.id)
                    await deviceUseCase.postDevicePet(device, pet)
            } else if(oldPet.device != null && oldPet.device !== "") {
                    const oldDevice = await deviceUseCase.getDevice(oldPet.device)
                    await deviceUseCase.destroyDevicePet(oldDevice, oldPet)
            }
            return await Pets.update(data, {where: {id: data.id}})
    }
```

Primero, se obtiene un dispositivo al id que se le pasa, además se obtienen los datos de la mascota previo a la actualización.

Si el campo asociado al dispositivo no era vacío y existe, se obtiene la mascota y se le asocia el dispositivo a la misma con los métodos 'getPet(id)' y 'postDevicPet(device, pet)' respectivamente.

En el caso de no pasar ningún dispositivo y la mascota tener uno asociado, se procede a desvincularlo, con el método 'destroyDevicePet(device, pet).

Por último, se llama al método 'update(data, query)' al que se le pasan los nuevos datos de la mascota con el dispositivo asociado o no.

Authorization

La API está preparada para operar con el campo de 'Authorization' de la cabecera HTTP, por lo que cuando se inicia sesión, se devuelve un Token, el Token se genera con JWT (Documentación de JWT [12] https://jwt.io/introduction) y el payload del mismo contiene el id del usuario.

Antes de mostrar el código de cómo se genera el Token, es necesario explicar el contenido del mismo. Ver Ilutración 4.12.

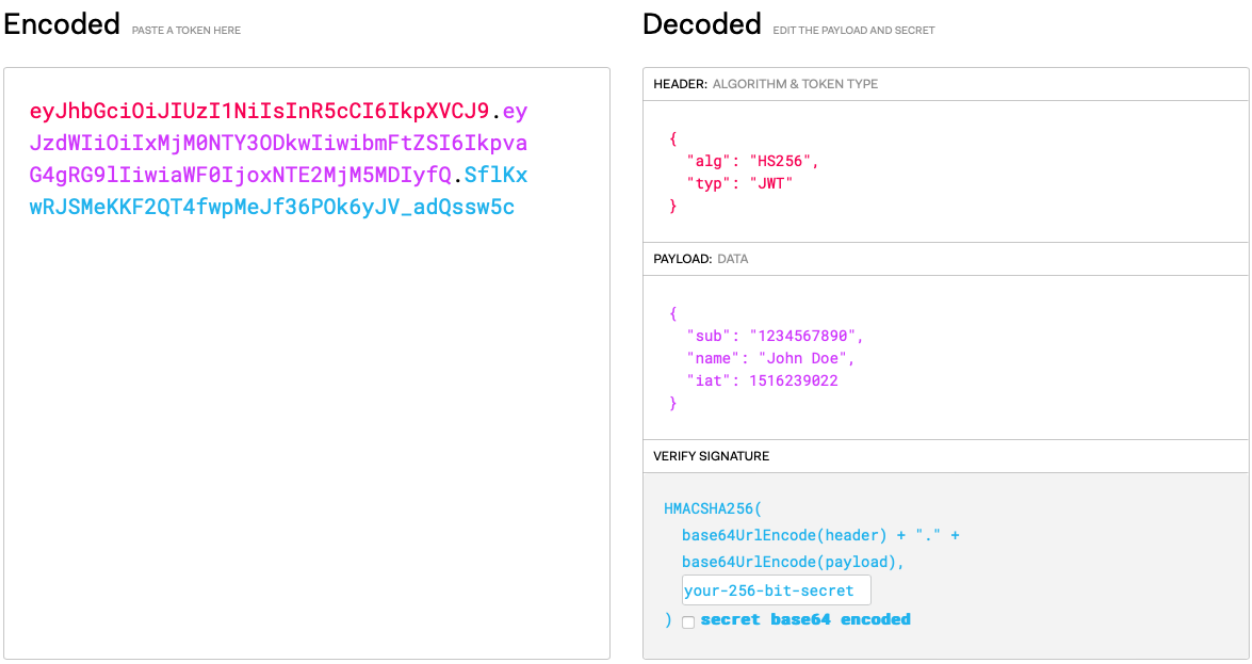


Ilustración 4.12: Ejemplo demostrativo de lo que recibe el usuario (encode) y los datos que tienen antes de ser generado (decoded) [imagen obtenida del debugger de la web [12]].

El secreto, en nuestro caso es una variable de entorno que es la que se utiliza para generar y codificar los Token.

Ahora se mostrará un ejemplo del inicio de sesión de un rol 'usuario' para el cuál se genera la firma de su JWT y se envía como respuesta en formato JSON.

```
        const userEmail = req.body['user-email']
        const user = await Users.findOne({where: {email: userEmail}})
        if (user !== null) {
                const password = req.body['user-password']
                const pepper = process.env.PEPPER
                const passwordHash = await bcrypt.hash(password + pepper, user.salt)
                if (passwordHash === user.password) {
                        const token = jwt.sign({'user-id': user.id}, process.env.JWT_SECRET)
                        res.send({token})
                }
   ...
```

Algoritmo 4.7: Ejemplo de la petición Signin del rol usuario.

Como se puede observar en el ejemplo, el objeto jwt realiza la 'firma' llamando al método sign(), que recibe como parámetros los campos que queremos introducir en nuestro payload y el secreto con el que se hace la firma. Posteriormente se envía como respuesta al cliente.

Cifrado

Un apartado importante es la privacidad y seguridad de los datos, por lo que se implementa Salt and Pepper para las contraseñas.

El funcionamiento es simple, consiste en intentar añadir ruido a la contraseña que introduce el usuario, mediante una salt que se genera de forma aleatoria y un pepper, que es una variable de entorno de la API, para el resultado se hace un hash y se almacena, teniendo la combinación de la contraseña original, la salt aleatoria y el pepper. Por supuesto, dicha salt debe ser almacenada, ya que la única manera de comprobar si la contraseña es correcta, es volver a hacer el proceso con dicha salt y el campo 'password' del usuario y ver si el hash resultante, coincide con el almacenado.

A continuación se muestra el código que realiza dicha función:

Esta responsabilidad está dentro de la clase 'UserModel.mjs' que usa un patrón Patrón Builder, ya que es la encargada de crear el objeto final:

```
    ...
    async build() {
      if(this.salt == null) {
        this.salt = getSalt()
        this.password = await hashPassword(this.password, this.salt, getPepper())
      }

      return new UserModel(this)
    }
  }
}

function getSalt() {
  return bcrypt.genSaltSync(10)
```

```
}

async function hashPassword(password, salt, pepper) {
  return await bcrypt.hash(password + (pepper ?? ''), salt)
}

function getPepper() {
  return process.env.PEPPER
}
```

En dicho código, se obtiene una salt aleatoria con el método 'getSalt(), después se llama a la función interna 'hashPassword(password, salt, pepper)', dicho resultado, es el que se almacenará en la Base de Datos. Ver Ilustración 4.13.

	id	name	surname	email	
	Filtro	Filtro	Filtro	Filtro	Filtro
1	6 8 4 4 0 b 7 a- 7 8	Marcos	Medina	markensi 9 @outlook.com	$ 2 b$ 1 0 $ 0
2	aa 2 f 5 6 2 f-...	TFG	Prueba	tfg@ulpgc.es	$ 2 b$ 1 0 $SifJ

Ilustración 4.13: Se muestra como queda almacenado el campo 'password' en la Base de Datos.

4.5.3. iOS

Para implementar la app móvil, se decide utilizar el patrón MVC, por lo que antes de mostrar la implementación y aspectos claves, se debe explicar en qué consiste el patrón MVC.

MVC

Este patrón de diseño consiste en tener de forma independiente y desacoplada las distintas partes del mismo, separando la lógica de la vista y teniendo un intermediario que se comunica entre ambos. Ver Ilustración 4.14.

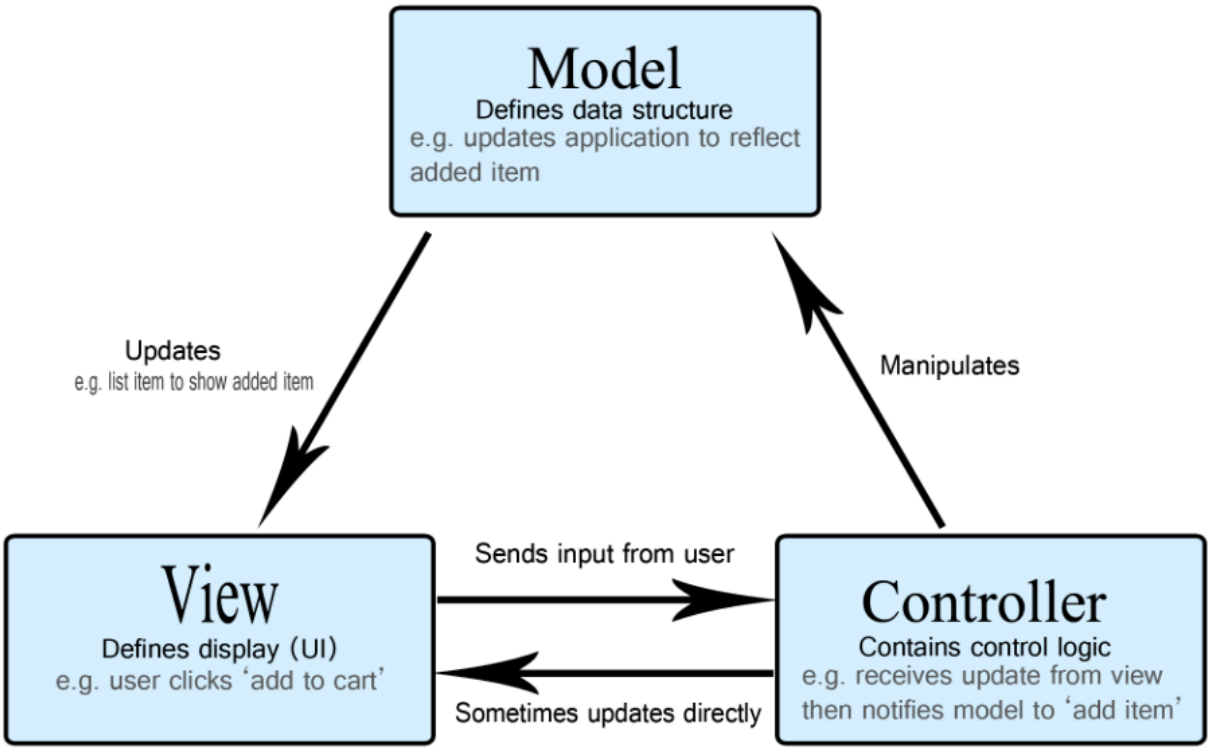


Ilustración 4.14: Ejemplo de comunicación del patrón MVC [obtenida de la entrada MDN sobre la explicación de dicho patrón [15]].

Esta aplicación distingue entre dos tipos de roles de usuario, por lo que primero vamos a centrarnos en la implementación del rol usuario, sus características y acciones. Después se centrará en el rol Veterinario y se finaliza con detalles de implementación de interés.

Usuario

Como usuario, se pueden hacer varias acciones, veremos en la lista las más comúnes y las que se mostrarán en detalle:

1. Create Read Update and Delete de usuario.
2. Create Read Update and Delete de mascotas.
3. Citas médicas de las mascotas.

Create Read Update and Delete de usuario

Una vez el usuario se registra e inicia sesión, se muestra la vista principal, Ilustración

4.15, donde se ve un listado de mascotas si tiene previamentes registradas, en caso contrario aparece vacío, además en la parte superior se encuentran los iconos que dan acceso a:

1. Crear una mascota.
2. Ver perfil del usuario.
3. Cerrar sesión.

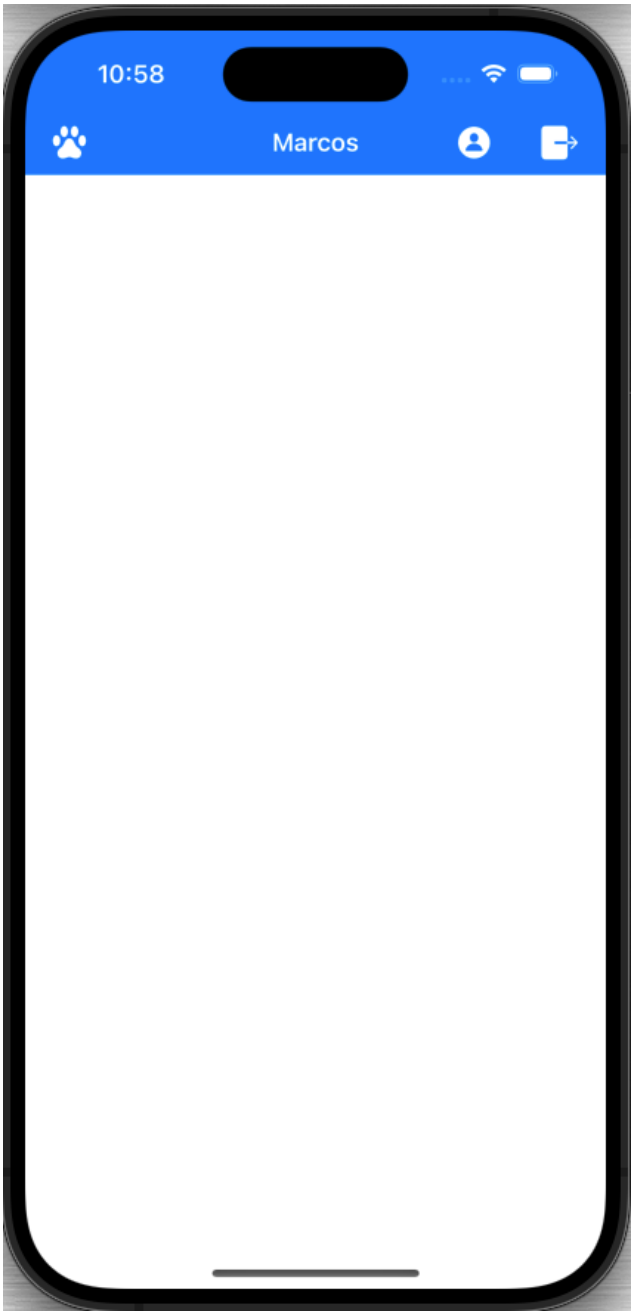


Ilustración 4.15: Vista principal del rol usuario sin mascotas asociadas.

Para las acciones Create Read Update and Delete del usuario, se debe dirigir al perfil del mismo, donde se muestran los datos de su perfil. Ver Ilustración 4.16.

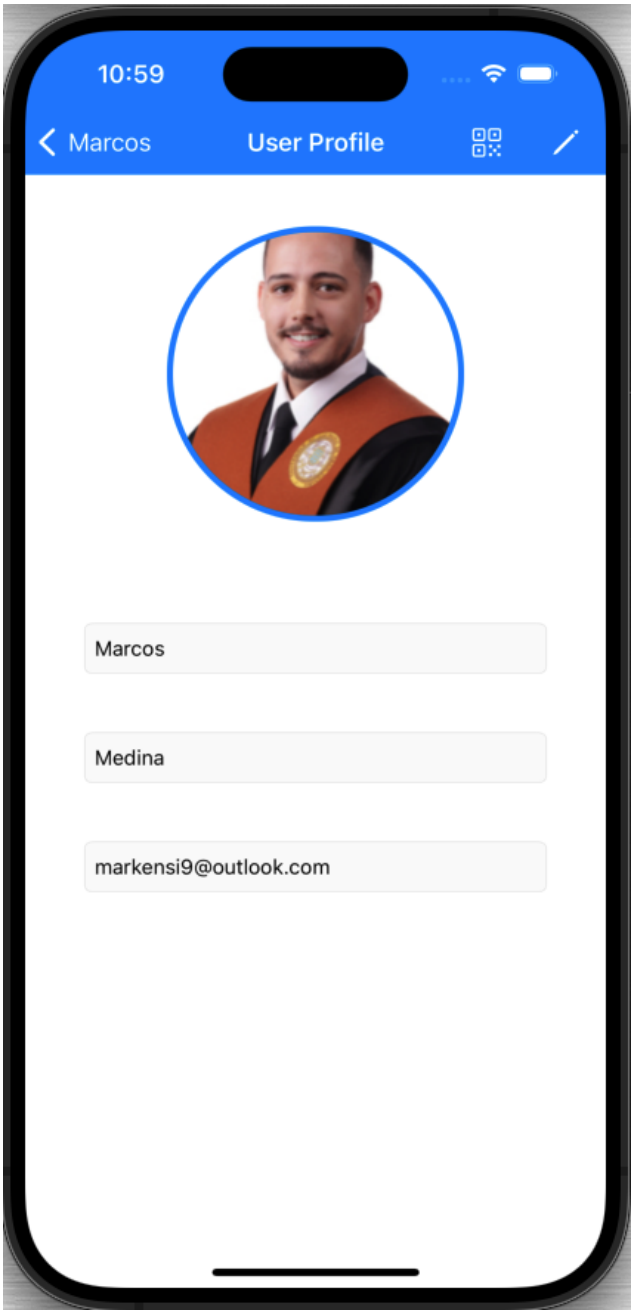


Ilustración 4.16: Vista del perfil del rol usuario.

En la parte superior tenemos varias opciones, si se pulsa sobre el lápiz se habilitan los campos y se puede cambiar la foto y contraseña. Se ilustrará un ejemplo de cómo se modifican y notifican dichos cambios. Ver Ilutración 4.17.

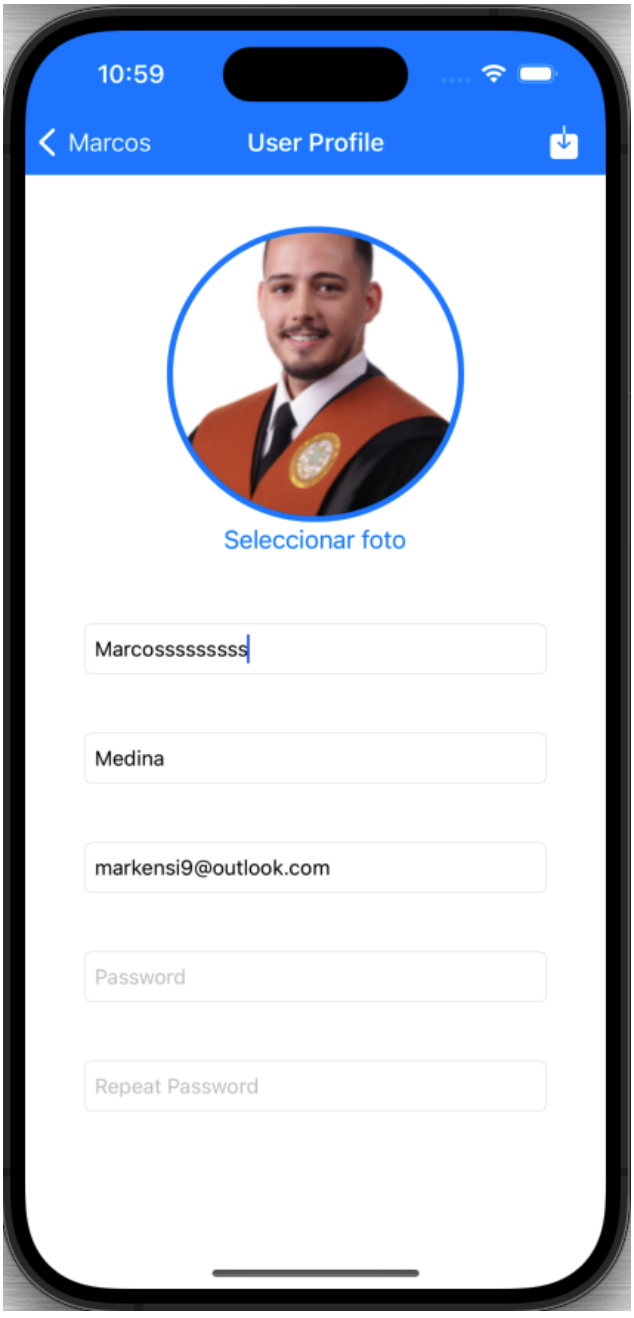


Ilustración 4.17: Vista del perfil del rol usuario editando el nombre.

Una vez modificados los campos, se pueden guardar con la opción de la barra superior. Una vez finalizado, se notifica al usuario el resultado de dicha acción. Ver Ilustración 4.18.

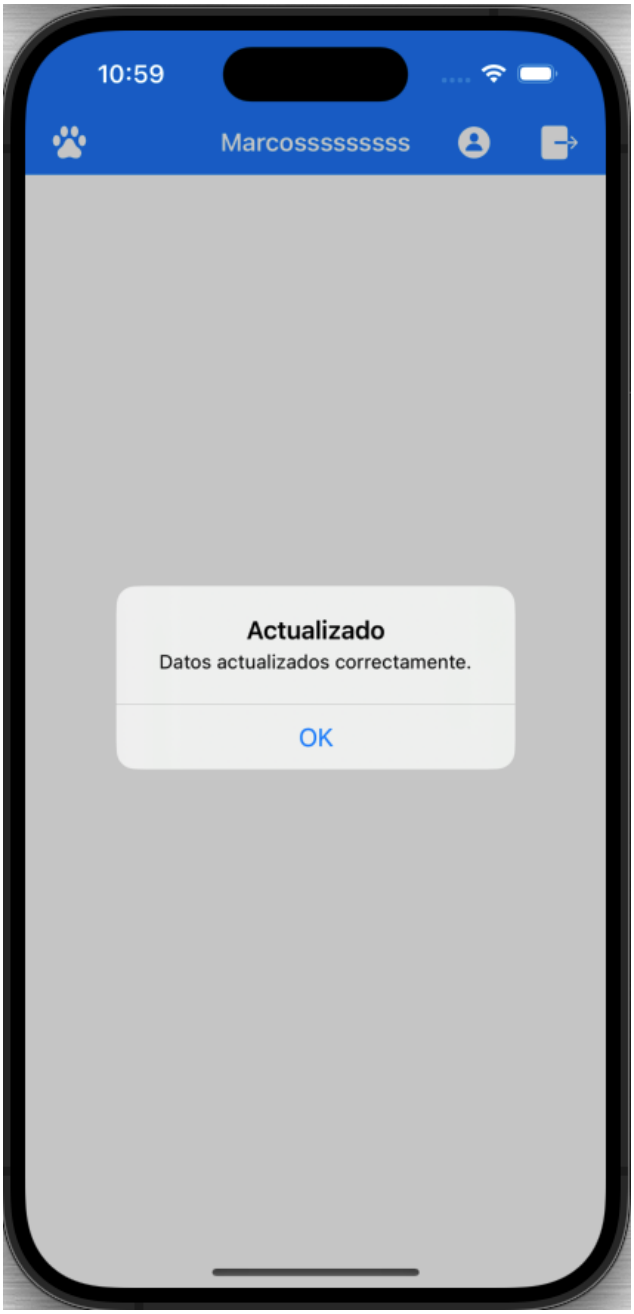


Ilustración 4.18: Vista del perfil del rol usuario editando el nombre.

Create Read Update and Delete de mascotas

Para añadir una nueva mascota asociada al usuario actual, se debe acceder al icono de la huella para abrir el formulario de registro de la misma. Ver Ilustración 4.19.

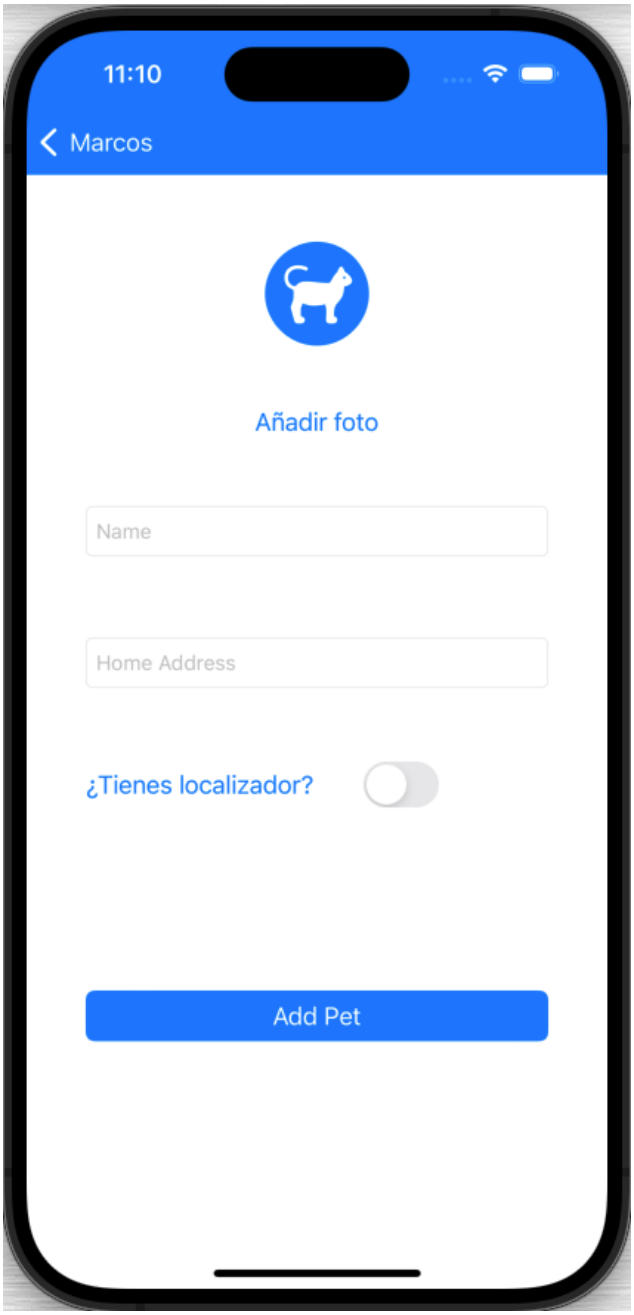
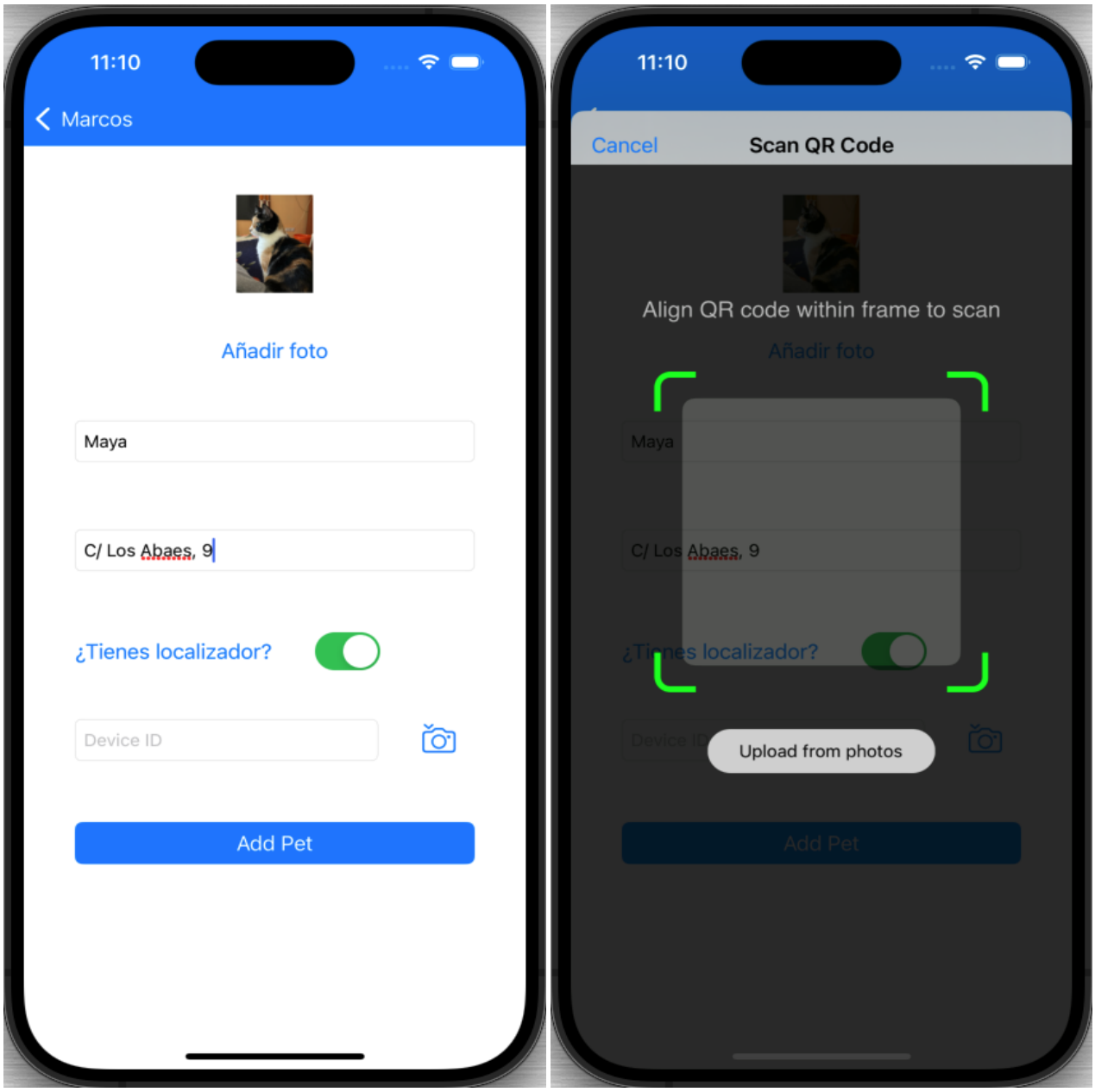


Ilustración 4.19: Vista del formulario para crear una mascota.



(a) Formulario mascota creando un nuevo registro. (b) Lector de QR.

Ilustración 4.20: Asociar dispositivo a mascota.

En dicho formulario puede añadir una foto de la mascota, sus datos y asociar un dispositivo localizador en caso de tener uno. Ver Ilustración 4.20a, para ello, se puede hacer uso del escaneo de código QR. Ver Ilustración 4.20b, a continuación se muestra los iconos desde los que ejecutar estas funcionalidades.

Ilustración 4.21: Vista del formulario con un dispositivo asociado.

Vemos como no podemos editar el campo una vez que se escanea. Ver Ilustración 4.21, para evitar errores del usuario, una vez tenemos todos los campos, podemos añadir la mascota y enviar una notificación al usuario. Ver Ilustración 4.22.

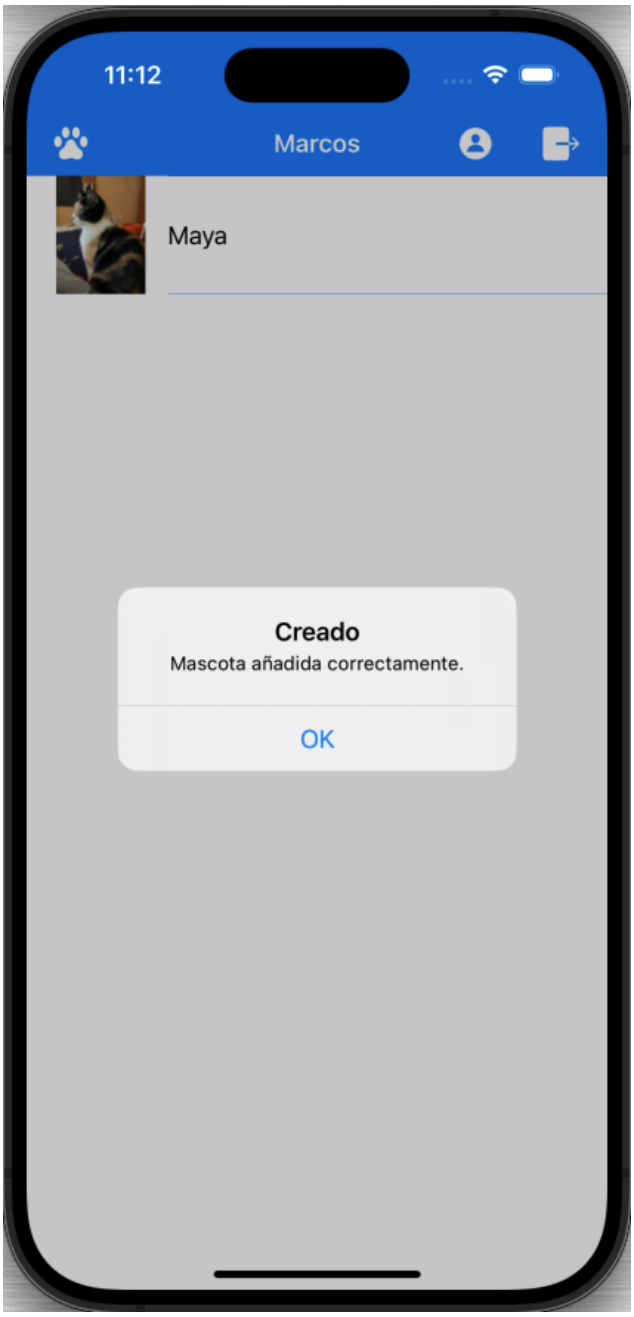


Ilustración 4.22: Notifiación al usuario de que se crea una nueva mascota.

Si se pulsa sobre una mascota, accedemos al perfil específico de la misma, en este caso como tiene asociado un dispositivo localizador, aparece en la mitad inferior de la vista el mapa con la ubicación de la mascota para poder localizarla. Ver Ilustración 4.23.

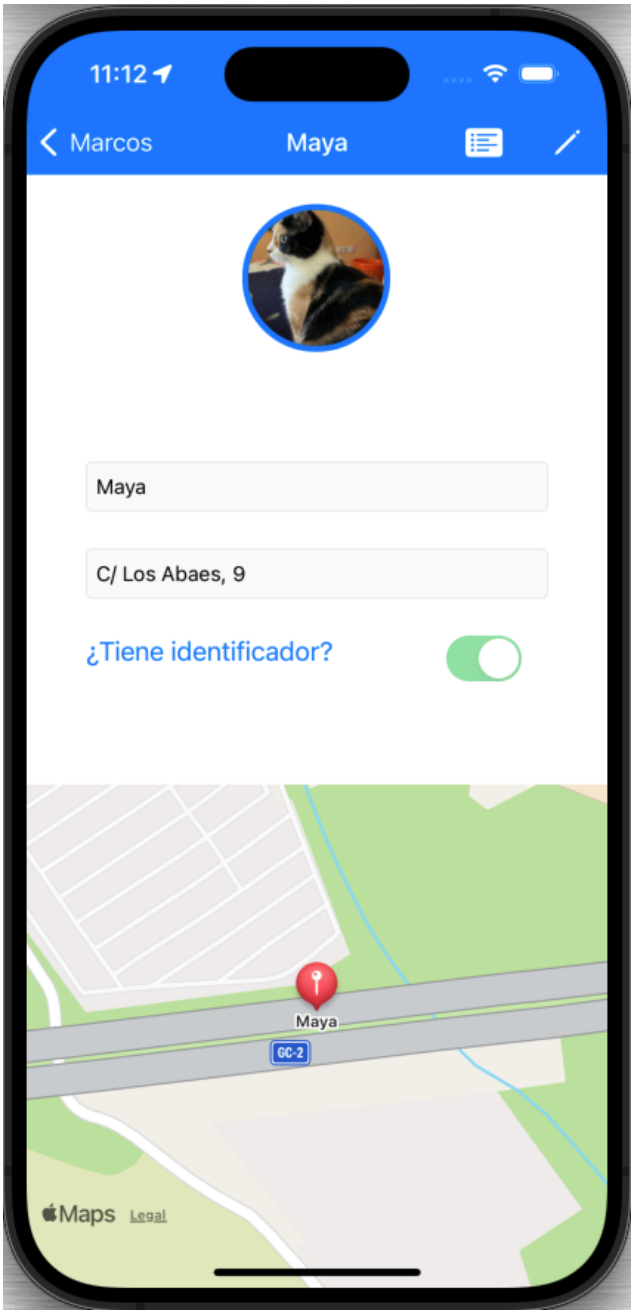
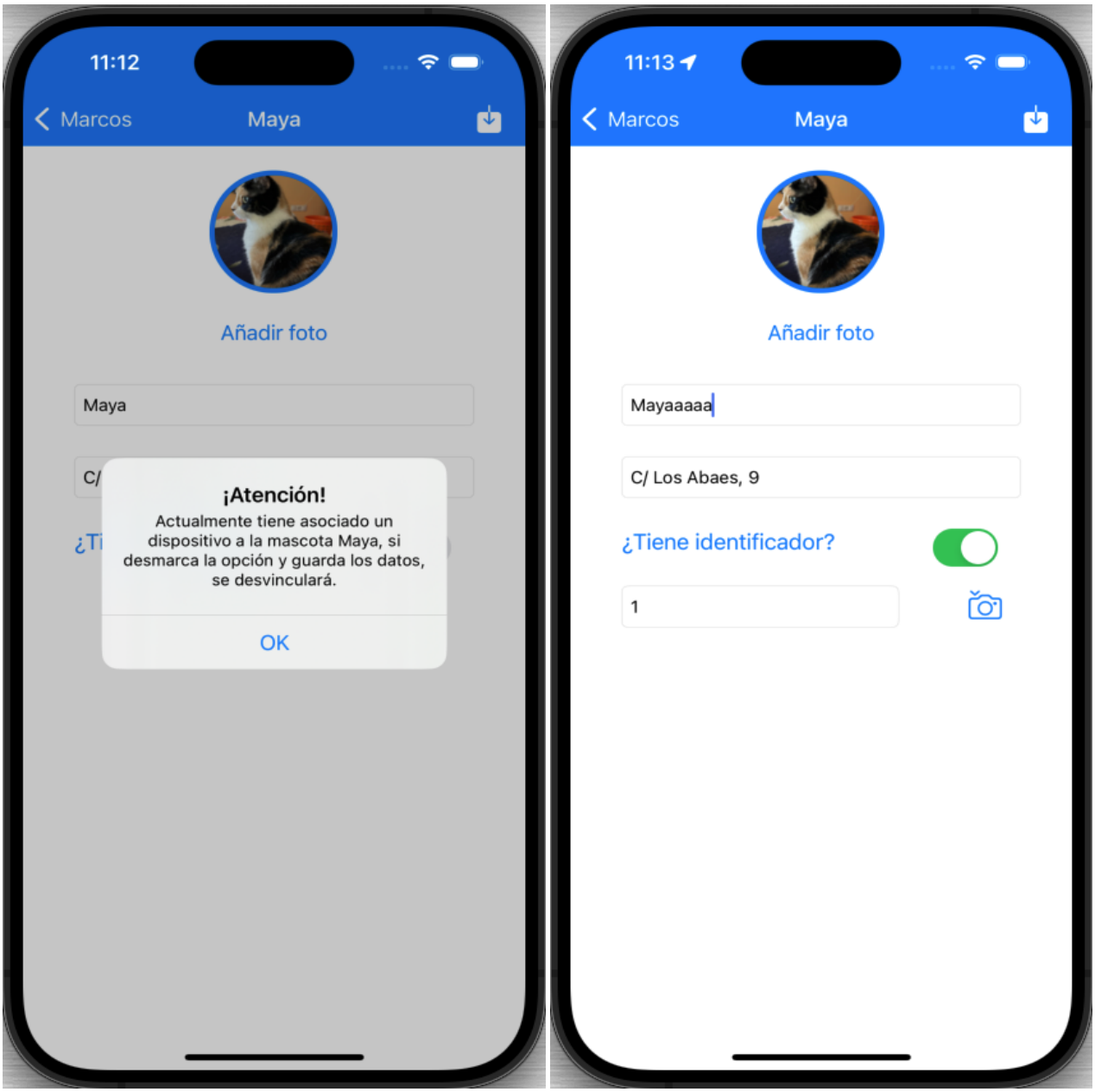


Ilustración 4.23: Vista del formulario para crear una mascota.

Siguiendo el mismo lenguaje de diseño, si se pulsa sobre el lápiz se tiene la posibilidad de editar los datos, tal y como pasa con el usuario, se habilitan los campos para ser editados y se oculta el mapa. Ver Ilustración 4.24.

Ilustración 4.24: Vista del formulario editar mascota.

En el caso en que se tenga asociado un dispositivo y se intenta desactivar, se le notifica al usuario para advertirle que dicha acción desvincula el localizador de la mascota. Ver Ilustración 4.25a.



(a) Alerta desvincular dispositivo. (b) Editando los datos de la mascota.

Ilustración 4.25: Acciones de edición mascota.

Una vez se actualizan los datos de la mascota, ver Ilustración 4.25b, se notifica al usuario y se ven reflejados los cambios 4.26.

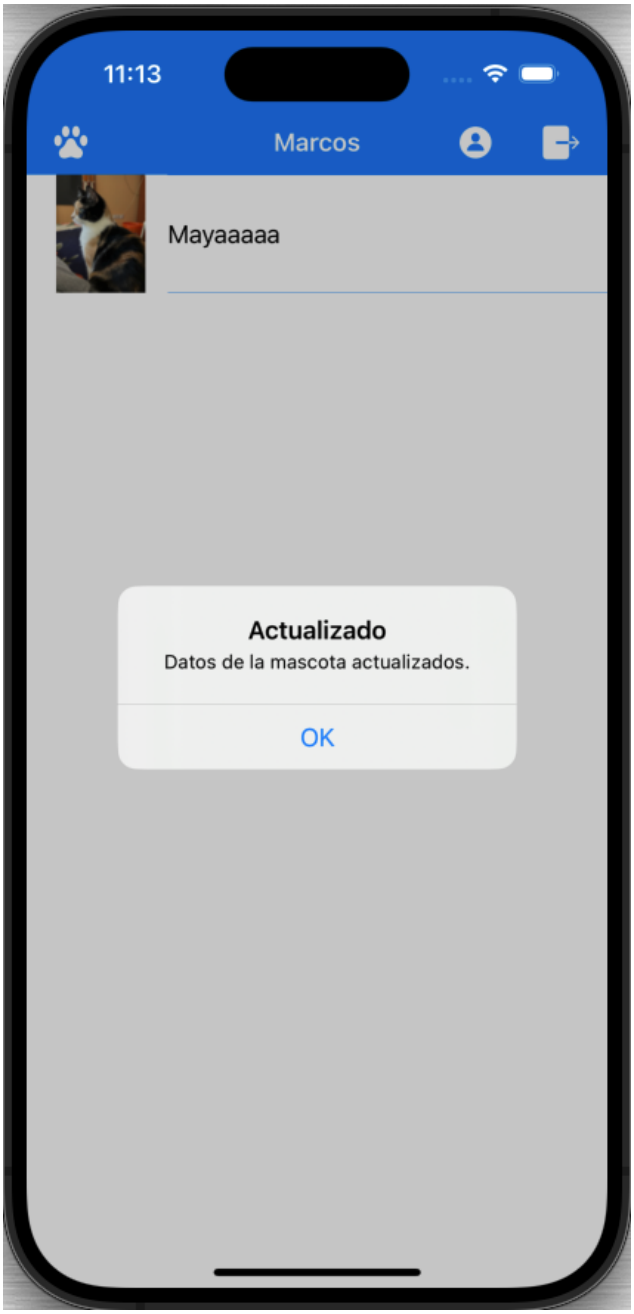
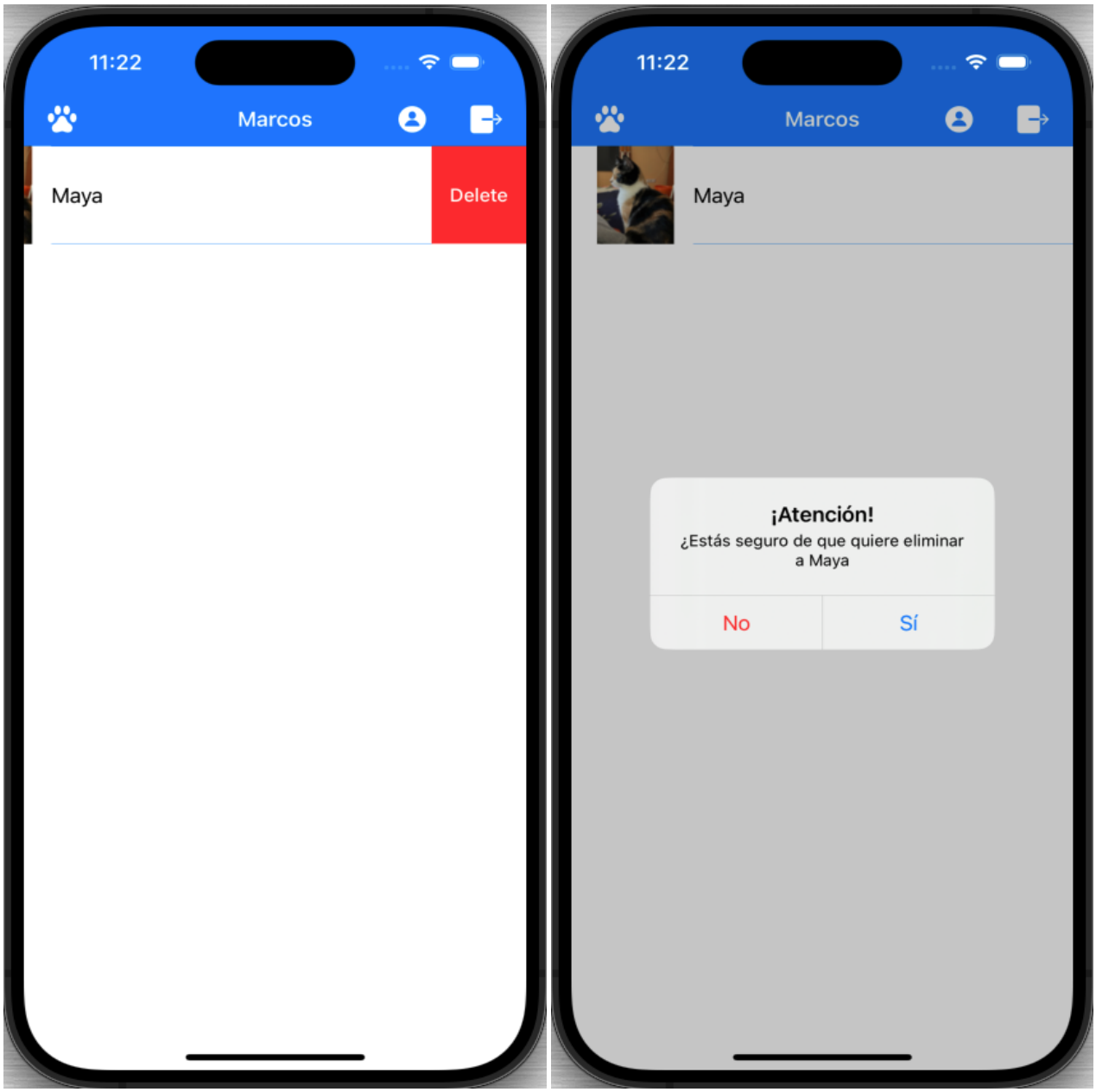


Ilustración 4.26: Alerta datos de mascota actualizados.

Por último, podemos eliminar la mascota deslizando hacia la izquierda desde la lista de mascotas, si después se pulsa sobre 'Delete'. Ver Ilustración 4.27a aparece un cuadro de diálogo en el que se notifica al usuario. Ver Ilustración 4.27b.

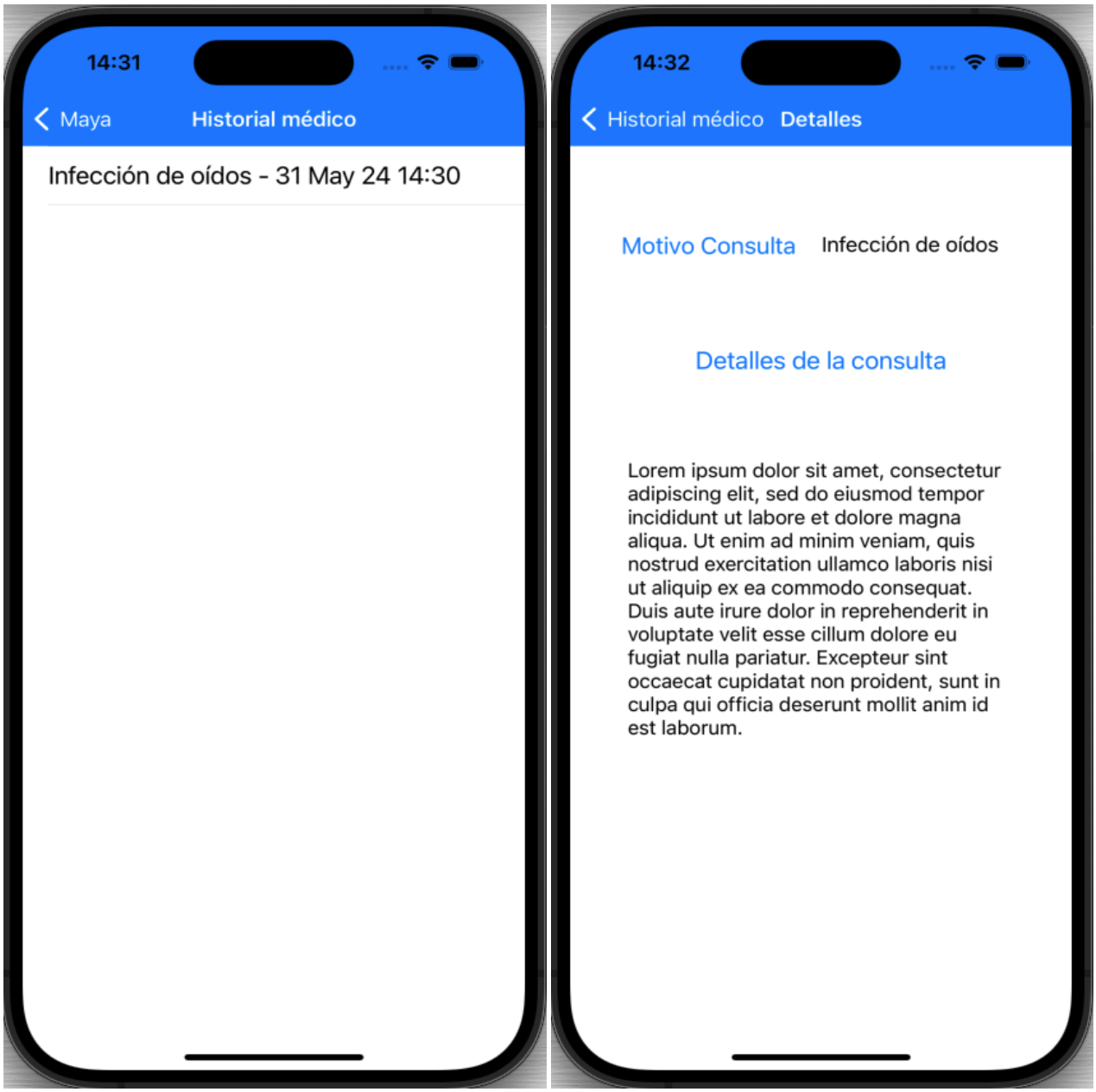


(a) Acción de borrar mascota. (b) Diálogo de confirmación.

Ilustración 4.27: Acción de eliminar una mascota.

Citas médicas de las mascotas

Cuando se accede al historial clínico, aparece un listado de las citas que tiene registradas la mascota con su título y fecha. Ver Ilustración 4.28a. Al acceder a cada una de ellas se muestra una vista detallada de la misma donde aparece todo lo que el veterinario anotó en la consulta. Ver Ilustración 4.28b.



(a) Historial médico de la mascota. (b) Vista detallada de la cita médica.

Ilustración 4.28: Acciones del historial médico de las mascotas.

Veterinario

Como veterinario, se pueden hacer varias acciones, veremos en la lista las más comúnes que posteriormente se detallarán:

1. Create Read Update and Delete de veterianrio.
2. Asociar cliente.
3. Create Read Update and Delete de citas médicas de las mascotas de un cliente.

Create Read Update and Delete de veterinario

Una vez que el veterinario inicia sesión, se muestra la vista principal. Ver Ilustración 4.29 donde se ve el listado de clientes que tenga asociados y si no tiene clientes, aparecerá vacío. En la parte superior se encuentran diferentes acciones como:

1. Añadir usuario.
2. Ver perfil del veterianrio.
3. Cerrar sesión.

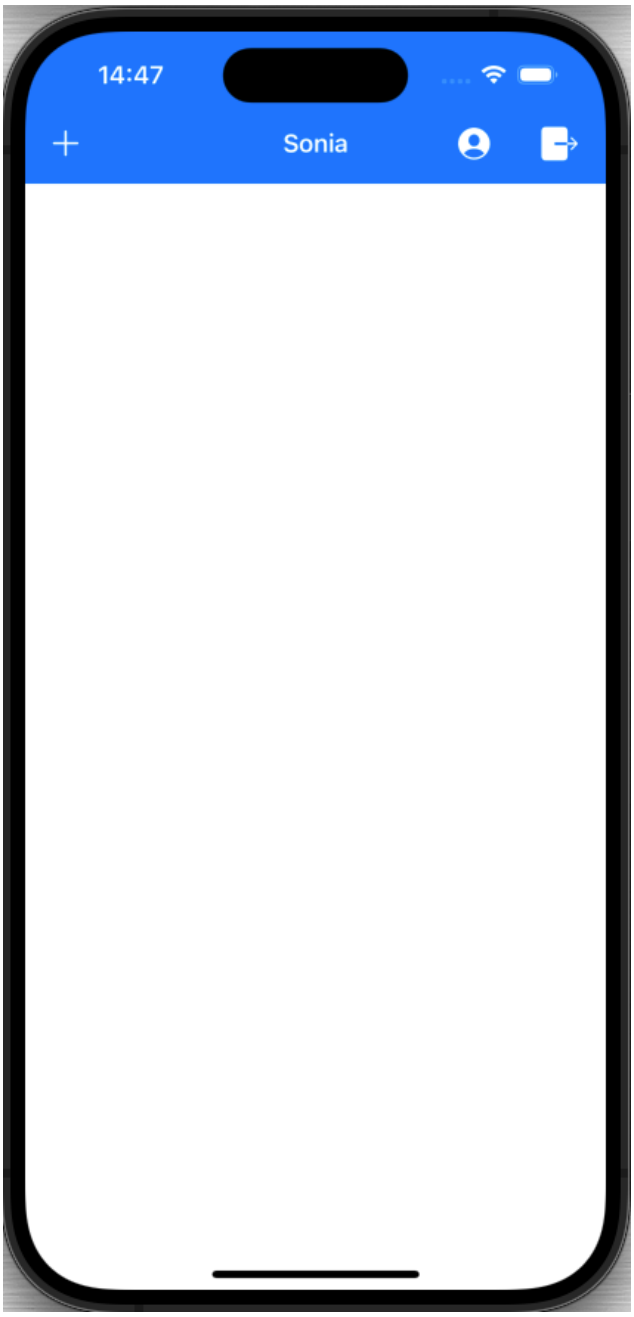


Ilustración 4.29: Vista principal del veterinario.

Para las acciones Create Read Update and Delete del veterinario, se debe acceder al perfil del mismo, donde se muestran los datos de su perfil. Ver Ilustración 4.30.

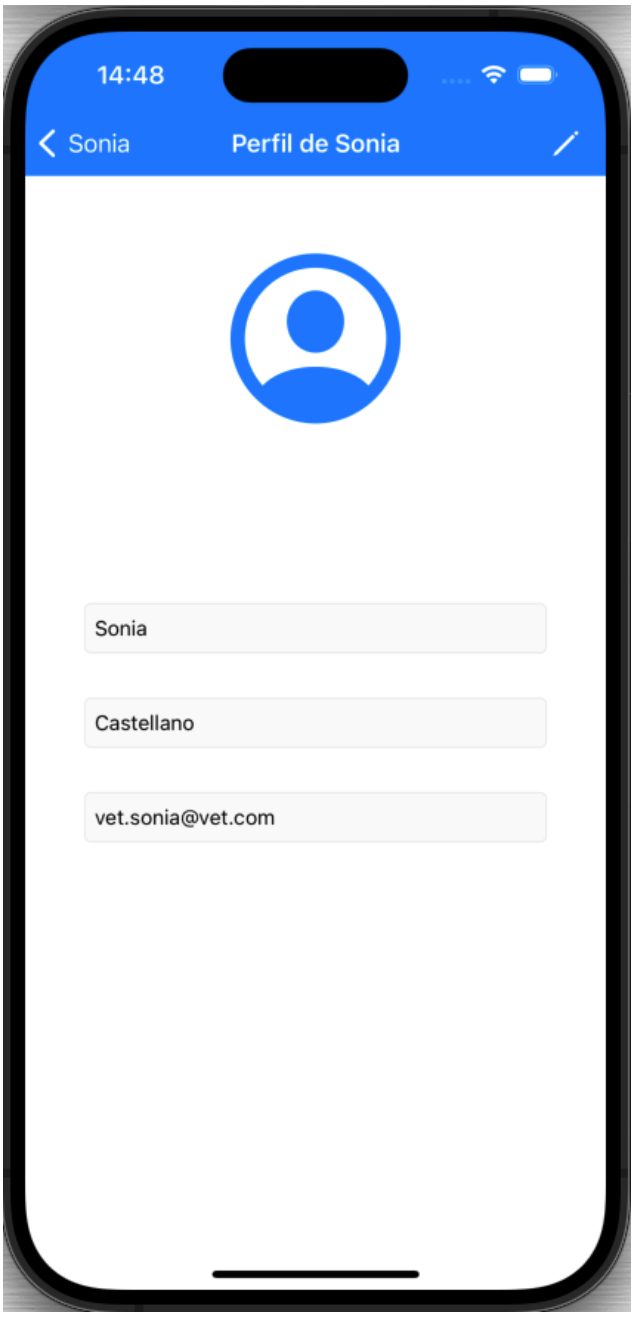


Ilustración 4.30: Vista principal del perfil veterinario.

En el menú superior, se pueden editar los datos si se pulsa sobre el lápiz, lo que habilita los campos y también el añadir una foto. Ver Ilustración 4.31.

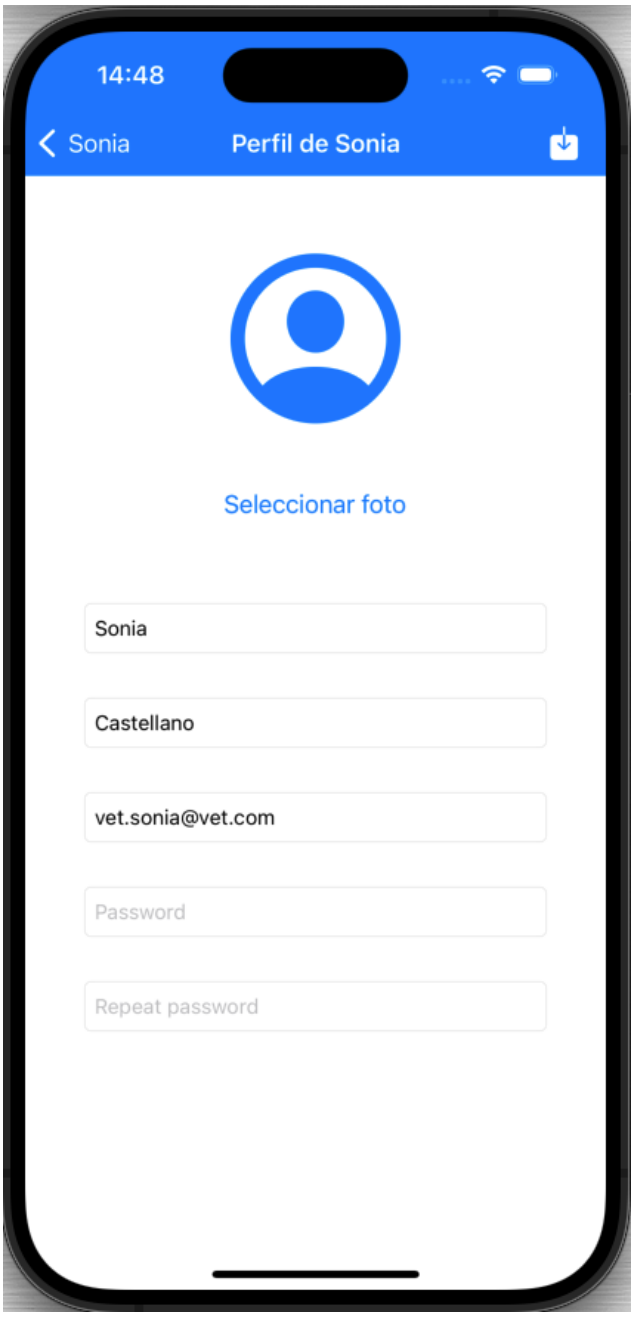
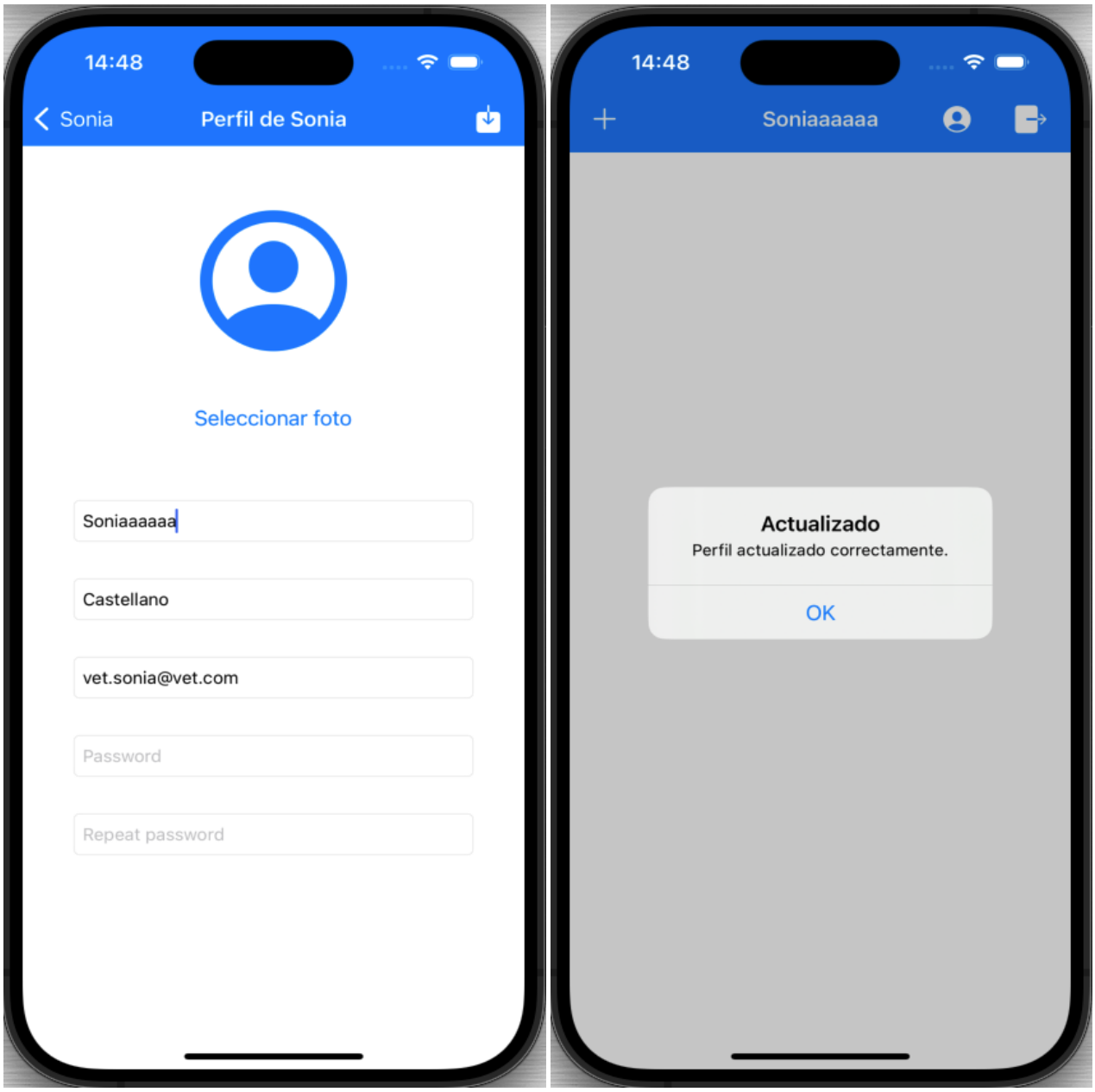


Ilustración 4.31: Vista principal del perfil veterinario para editar.

Una vez editados los campos, ver Ilustración 4.32a. Se guardan y se envían al servidor, también se notifica al usuario de que la acción se ha completado correctamente. Ver Ilustración 4.32b.



(a) Datos modificados. (b) Notificación de que se completa la acción.

Ilustración 4.32: Acciones de edición del perfil del veterinario.

Asociar cliente

Para poder asociar clientes en el menú superior de la 'main view'. Ver Ilustración 4.29. El icono '+' muestra un pop up que nos permite escribir el identificador, una forma alternativa es mediante el escaneo de códigos QR, con lo que se rellena directamente el campo.

Como rol 'usuario', en la sección de perfil, ver Ilustración 4.16. En la barra superior existe un icono 'QR' que genera un código con su id. Ver Ilustración 4.33 .

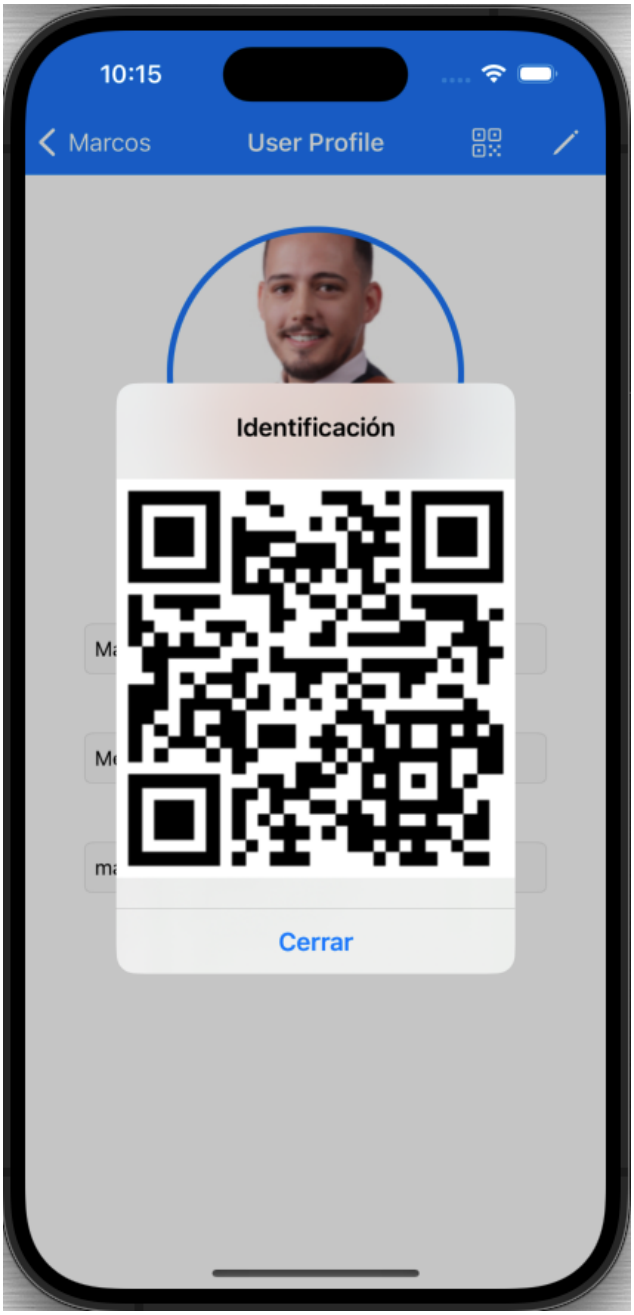
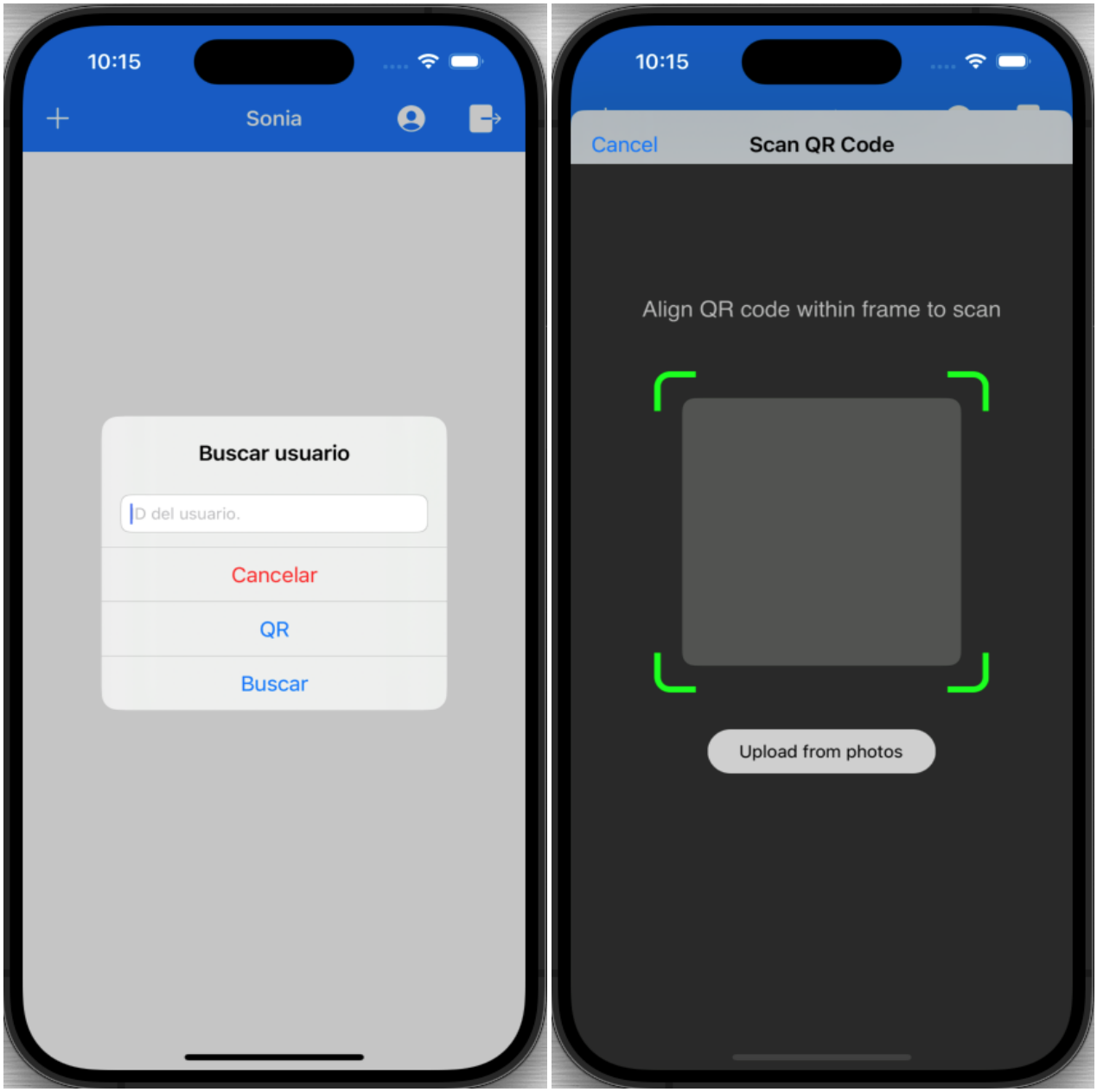


Ilustración 4.33: Vista principal del perfil usuario con el QR generado.

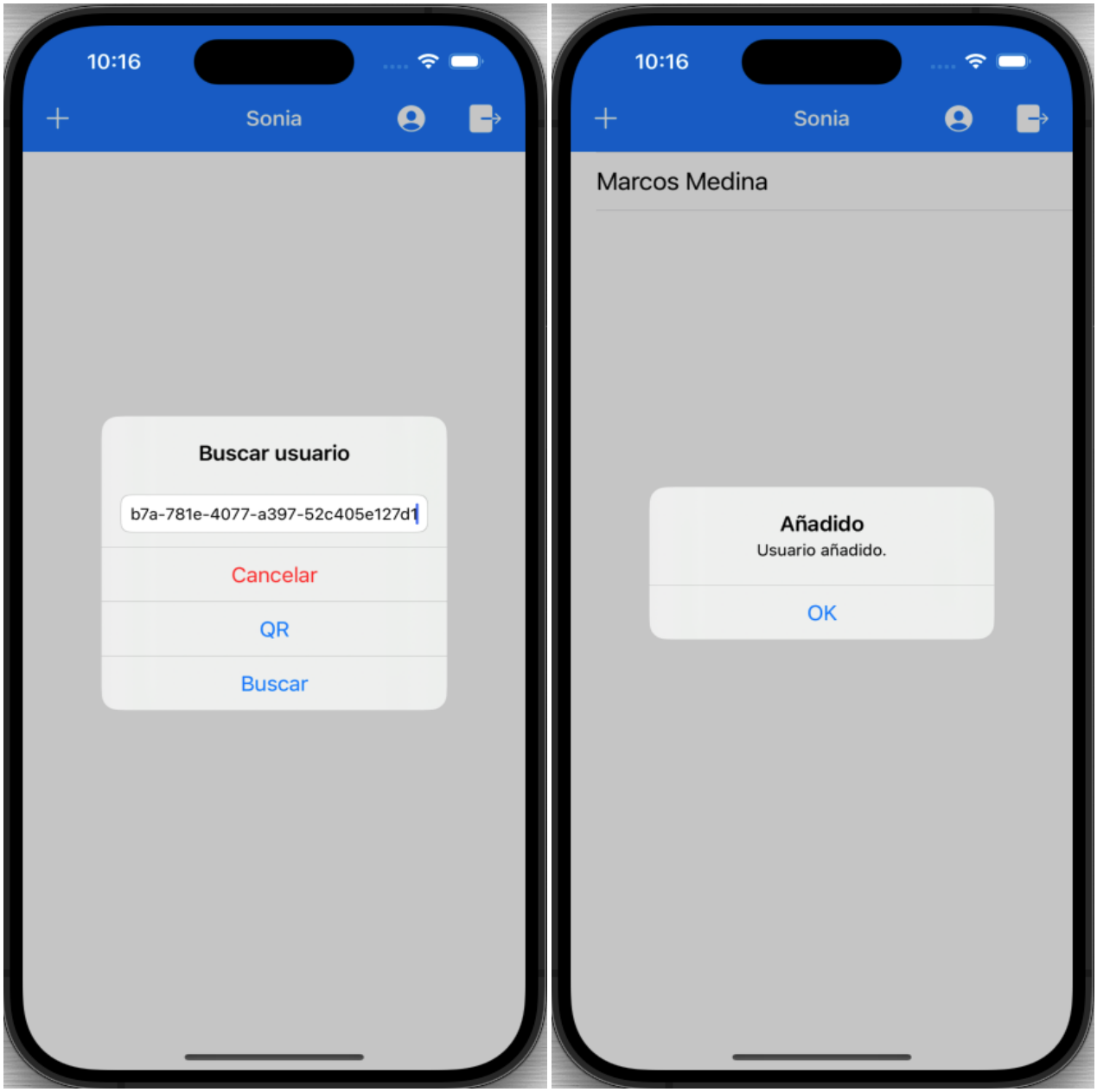
Desde la 'Main View' del veterinario, ver Ilustración 4.29 se accede al menú pop up del símbolo '+'. Ver Ilustración 4.34a



(a) Menú pop up. (b) Lector QR para asociar un usuario.

Ilustración 4.34: Acciones para asociar un cliente al veterinario.

Una vez se escanea el código QR, ver Ilustración 4.34b , en el menú pop up aparece dicho campo rellenado. Ver Ilustración 4.35a. A continuación, se pulsa sobre 'Buscar' y se asocia dicho usuario, notificando al veterinario y apareciendo en la lista. Ver Ilustración 4.35b .

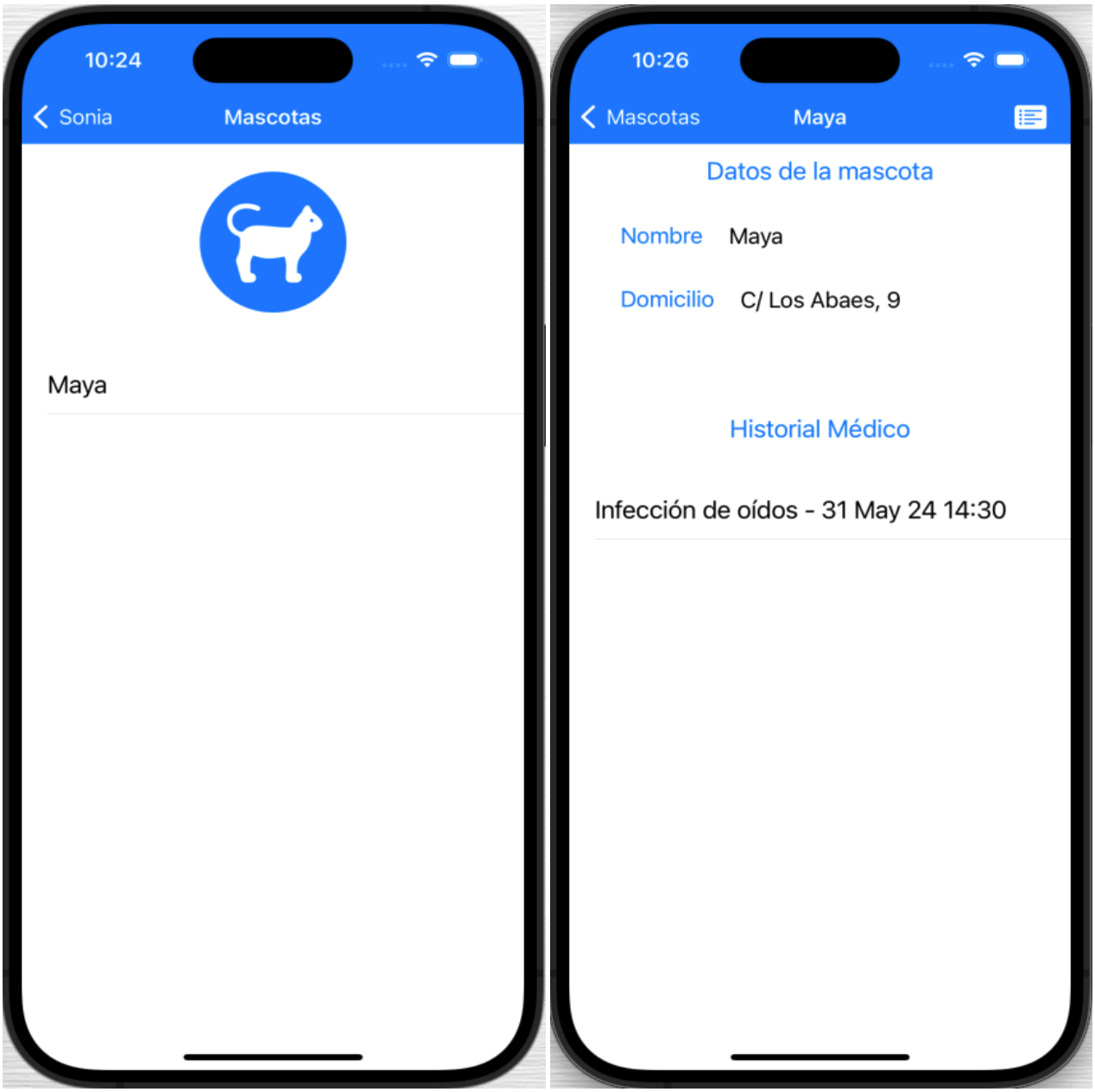


(a) ID puesto en el campo de búsqueda. (b) Notificación de que se completa la acción.

Ilustración 4.35: Acciones para asociar un cliente al veterinario.

Create Read Update and Delete de las citas médicas

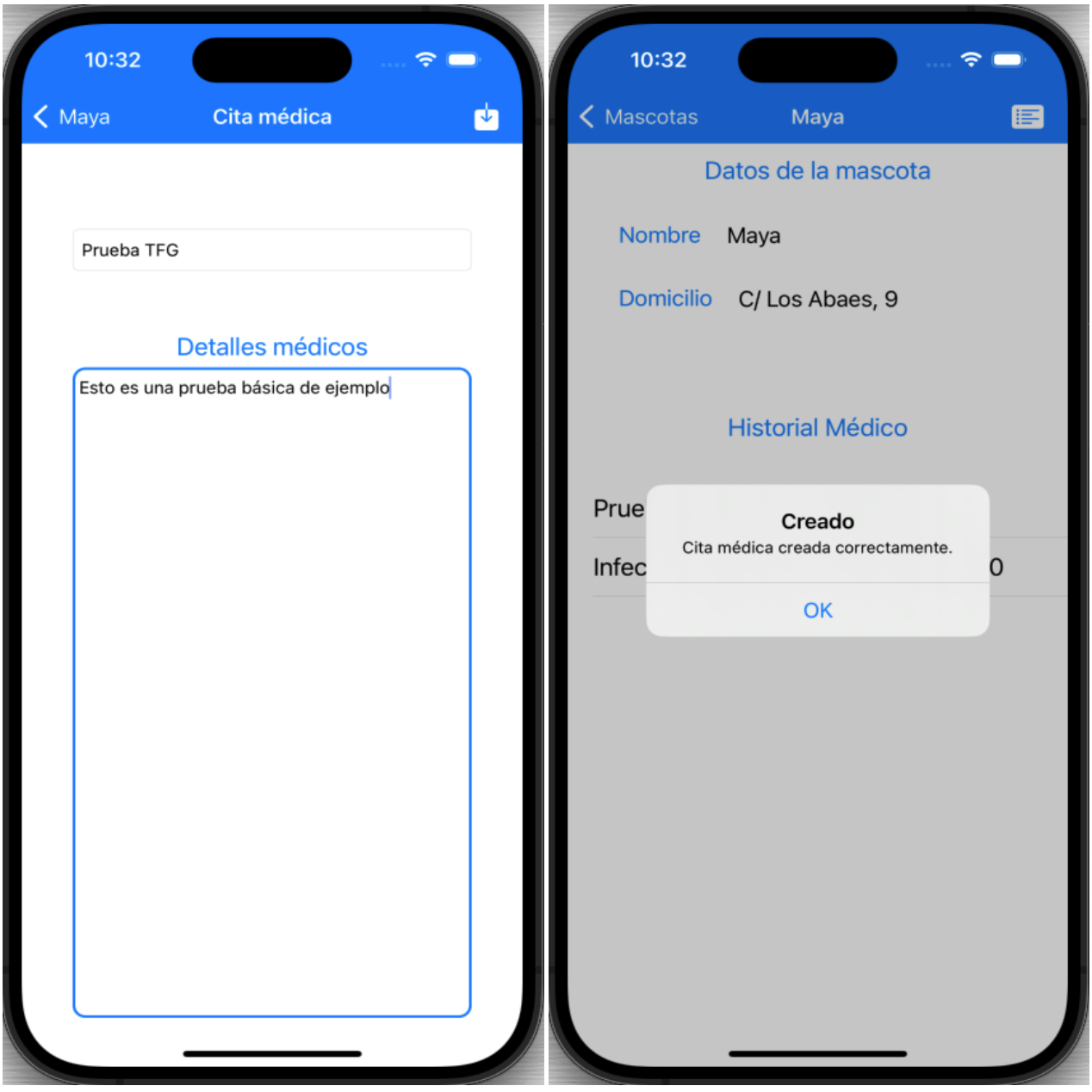
Una vez se accede a un cliente desde el listado principal, ver Ilustración 4.29. Aparece el listado de las mascotas que ese cliente tiene registradas, ver Ilustración 4.36a. Una vez seleccionamos la mascota a tratar, veremos los detalles básicos de la misma, así como el historial médico de la mascota. Ver Ilustración 4.36b .



(a) Listado de mascotas del usuario seleccionado. (b) Detalles básicos de la mascota seleccionada.

Ilustración 4.36: Selección de cliente y mascota.

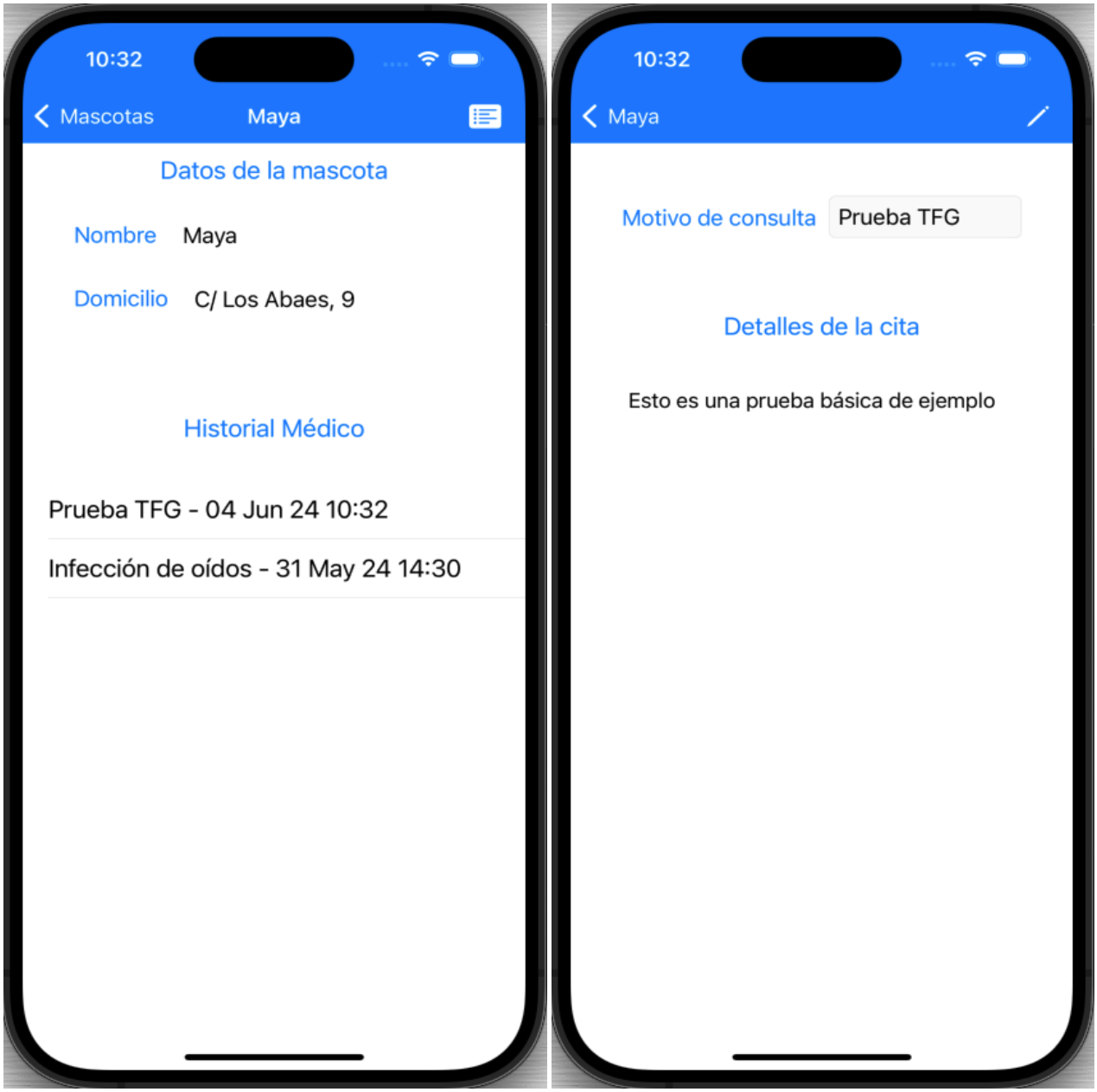
Para añadir un nuevo reporte médico a la mascota en cuestión, en la barra superior, se encuentra un icono de listado. Ver Ilustración 4.36b, si se accede, aparecen los campos para añadir la descripción y título de la misma, ver Ilustración 4.37a. Una vez se añaden, se notifica al usuario. Ver Ilustración 4.37b .



(a) Rellena el formulario para un reporte médico. (b) Notificación de que se completa la acción.

Ilustración 4.37: Acciones de añadir un reporte a una mascota.

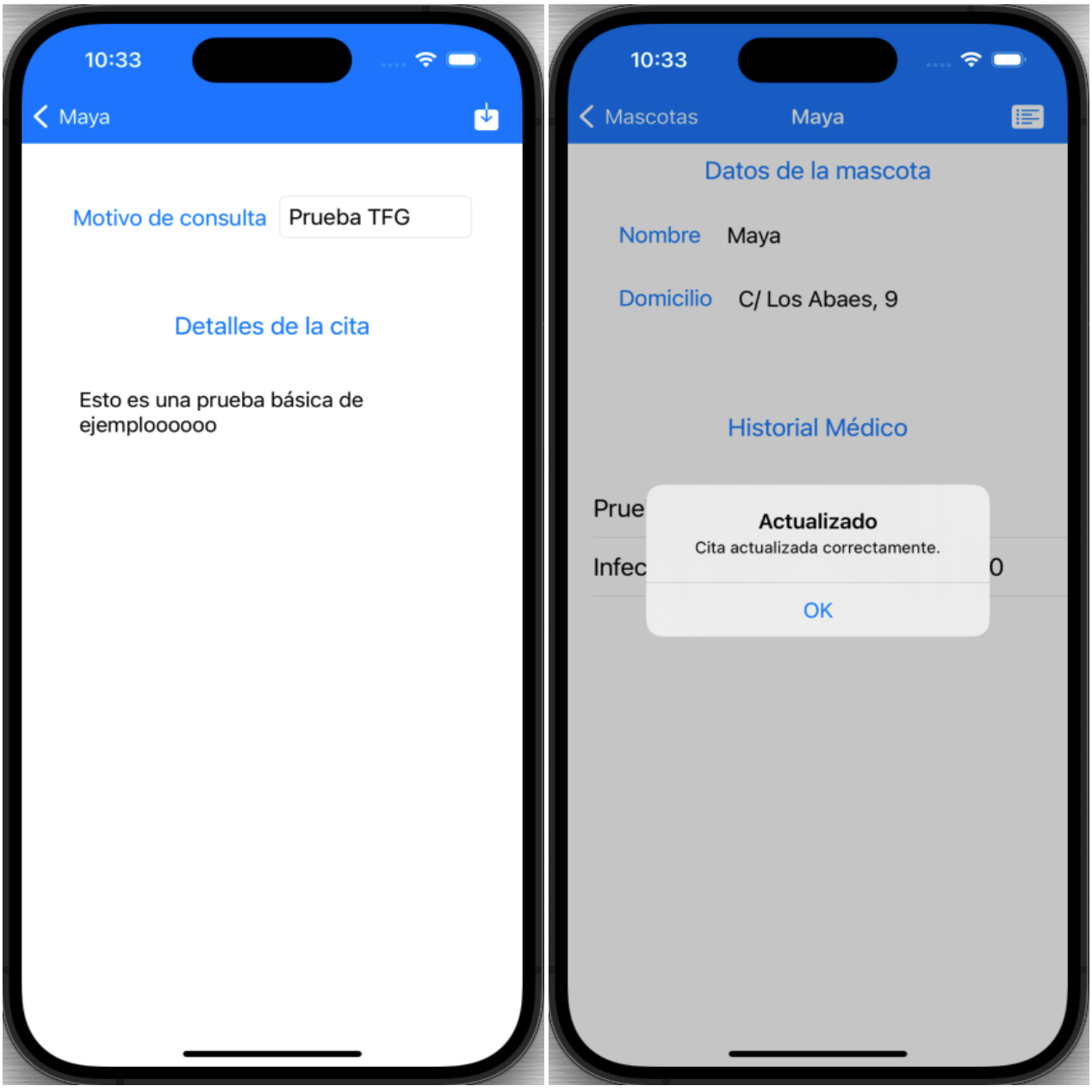
En el listado de la mascota aparece el reporte médico añadido. Ver Ilustración 4.38a, por lo que se puede ver la vista detallada de cualquiera de estos reportes. Ver Ilustración 4.38b .



(a) Listado de los reportes médicos de la mascota. (b) Vista detallada del reporte médico.

Ilustración 4.38: Acciones ver los datos de un reporte.

En la vista detallada del reporte en cuestión, se observa que en la barra superior existe un botón para editar dicho reporte, ver Ilustración 4.38b. Por lo que si se pulsa, se habilitan los campos para poder ser editados, ver Ilustración 4.39a. Una vez que se almacenan los datos, se notifica al usuario. Ver Ilustración 4.39b.



(a) Edición de un reporte médico. (b) Notificación de que se completa la acción.

Ilustración 4.39: Acciones de editar un reporte a una mascota.

Optimización para cargar imágenes
En la aplicación en todo momento se intenta optimizar los recursos que se tienen, ya que en un dispositivo móvil es muy importante.

Para mejorar la eficiencia en este aspecto, existe un servicio que recibe la imagen que se quiere cargar y los valores a los que se quiere redimencionar, esta función devuelve la imagen en el tamaño deseado, por lo que el consumo de memoria se ve reducido de forma significativa. Además, las llamadas a dicho servicio, se hacen mediante hilos en segundo plano para no bloquear el hilo principal de la aplicación.

```
func resizeImage(image: UIImage, targetSize: CGSize) -> UIImage {
    let size = image.size

    let widthRatio  = targetSize.width  / size.width
    let heightRatio = targetSize.height / size.height

    var newSize: CGSize
    if widthRatio > heightRatio {
        newSize = CGSize(width: size.width * heightRatio, height: size.height * heightRatio)
    } else {
        newSize = CGSize(width: size.width * widthRatio, height: size.height * widthRatio)
    }

    // Redimensiona la imagen
    let rect = CGRect(origin: .zero, size: newSize)

    UIGraphicsBeginImageContextWithOptions(newSize, false, 1.0)
    image.draw(in: rect)
    let newImage = UIGraphicsGetImageFromCurrentImageContext()
    UIGraphicsEndImageContext()

    return newImage!
}
```

Algoritmo 4.8: Función dentro del servicio que reduce el tamaño de la imagen.

```
        if let image = UIImage(contentsOfFile: imagePath) {
            let targetSize = CGSize(width: 300, height: 300)
            DispatchQueue.global().async {
                let imageResized = ResizeImageService.shared.resizeImage(image: image, targetSize: targetSize)
                DispatchQueue.main.async {
                    self.photoField.image = imageResized
                }
            }
        }
```

Algoritmo 4.9: Ejemplo de llamada al servicio

Cargar la ubicación en segundo plano
Otro de los aspectos que consumía mucha memoria es el tener que pintar constantemente un mapa e ir actualizando la ubicación (cada 60 segundos se consulta a la API). Para mejorar la eficiencia en dicho aspecto, se utilizan los hilos en segundo plano donde se lanza dicho hilo que hace las peticiones para obtener la ubicación y refrescar la información en el mapa. De

esta forma se consigue que no se bloquee el hilo principal de la aplicación y además se reduce el consumo de memoria.

Una vez el mapa desaparece de la vista, ya sea porque se edita la mascota o se sale de la misma, se deja de pintar dicho mapa y también se para el hilo, con lo que se optimizan los recursos de la aplicación.

También se tiene un servicio del mapa, donde se crea una única vez el objeto 'maker' que es el icono que se pinta en el mismo. Posteriormente se actualiza dicha referencia, con lo que se optimizan mejor los recursos.

A continuación, se muestra los códigos encargados de actualizar el mapa en segundo plano, así como de parar el código del mismo.

```
DispatchQueue.main.async {
    print("Antes␣de␣actualizat␣mapa.")
    //self.updateMap(latitude: latitude, longitude: longitude)
    print("Dentro␣del␣hilo␣de␣pintar␣mascota␣en␣mapa")

    if self.petAnnotation == nil {
        self.petAnnotation = MapService.shared.createMarker(data: data)
        self.petAnnotation!.title = self.pet?.name
        self.mapPetLocation.addAnnotation(self.petAnnotation!)
    }

    self.petAnnotation?.coordinate = CLLocationCoordinate2D(latitude: latitude, longitude: longitude)

    let region = MapService.shared.createRegion(data: data)
    self.mapPetLocation.setRegion(region, animated: true)
}
```

Algoritmo 4.10: Hilo donde se actualiza el mapa.

Si la respuesta recibida por la API para obtener la ubicación es favorable, se lanza en un hilo para refrescar los datos en el mapa. Accediendo al 'MapService' para actualizar el objeto 'maker' y actualizando en el mapa la anotación con los datos recibidos del servicio.

```
class MapService {
    static let shared = MapService()
    static var petLocation: CLLocationCoordinate2D? = nil
    var petAnnotation: MKPointAnnotation?

    public func createMarker(data:[String: Any]) -> MKPointAnnotation {
        let location = CLLocationCoordinate2D(latitude: data["latitude"] as! CLLocationDegrees, longitude: data["longitude
        MapService.petLocation = location
        let marker = MKPointAnnotation()
        marker.coordinate = location

        return marker
    }
```

```
    public func createRegion(data: [String:Any]) -> MKCoordinateRegion {
        let region = MKCoordinateRegion(center: CLLocationCoordinate2D(latitude: data["latitude"] as! CLLocationDegrees, l

        return region
    }
}
```

Algoritmo 4.11: Clase MapService

Dicho servicio nos devuelve la anotación para que se pueda añadir en el mapa, además de la región del mismo. De esta forma, ambos objetos se crean una única vez y se actualizan las referencias.

Dependencias
La aplicación utiliza tres librerías desarrolladas por terceros:

1. Alamofire 5.8.1 - Conexiones HTTP
2. Codescanner 2.4.1 - Lector de códigos QR
3. SwiftQRCodeScanner 1.16.0 - Generador de QR

4.5.4. Localizador

La implementación del localizador requiere accesorios hardware:

1. Arduino UNO
2. SIM900
3. Tarjeta sim
4. Tranformador de 12v - 1A

El objetivo del localizador es el de conectarse a la API y enviar la ubicación que tiene registrada actualmente para ir actualizando los valores y que estos sean consumidos por el cliente (app iOS).

Debido a las características de la placa SIM900, el acceso a la ubicación GSM mediante las posiciones de las antenas cercanas no está operativa, por lo que se requeriría de un segundo módulo GPS para poder obtener la ubicación actual, debido a esto, para la demostración, dicha ubicacón está escrita en el código a fin de poder acualizar los datos en la Base de Datos mediante la llamada al endpoint de la API.

Para que el código del localizador funcione correctamente, se deben actualizar los datos con el APN de la compañía del operador de la tarjeta sim utilizada. En este proyecto, la tarjeta sim es de la compañía O2, que pertenece a Telefónica.

```
// Configurar el APN
  sim900.println("AT+SAPBR=3,1,\"APN\",\"telefonica.es\"");
  delay(2000);
  while (sim900.available()) {
    Serial.write(sim900.read());
  }

  // Configurar el usuario
  sim900.println("AT+SAPBR=3,1,\"USER\",\"telefonica\"");
  delay(2000);
  while (sim900.available()) {
    Serial.write(sim900.read());
  }

  sim900.println("AT+SAPBR=3,1,\"PWD\",\"telefonica\"");
  delay(2000);
  while (sim900.available()) {
    Serial.write(sim900.read());
  }

  // Configurar contexto PDP
  sim900.println("AT+CSTT=\"telefonica.es\",\"telefonica\",\"telefonica\"");
  delay(2000);
  while (sim900.available()) {
    Serial.write(sim900.read());
  }
```

Algoritmo 4.12: Datos del APN de la tarjeta SIM en el setup

Una vez se inicia la placa un timer cada 60 segundos se activa para lanzar la petición HTTP a la API con las coordenadas actuales mediante un JSON. Después se descativa la antena para ahorrar energía y se duerme durante 60 segundos.

```
Serial.println("Iniciando␣HTTP...");
  sim900.println("AT+HTTPINIT");
  delay(2000);
  while (sim900.available()) {
    Serial.write(sim900.read());
  }

  Serial.println("Preparando␣URL...");
  sim900.println("AT+HTTPPARA=\"URL\",\"http://2.137.57.130:3000/devices/"+String(deviceId)+"/location\"");
  delay(2000);
  while (sim900.available()) {
    Serial.write(sim900.read());
  }
  sim900.println("AT+HTTPPARA=\"CONTENT\",\"application/json\"");
  delay(1000);
  while (sim900.available()) {
    Serial.write(sim900.read());
  }
  String postData = "{\"latitude\":␣28.126949083018815,␣\"longitude\":␣-15.450903430612021}";
  sim900.println("AT+HTTPDATA=" + String(postData.length()) + ",␣10000");
  delay(1000);
  while (sim900.available()) {
    Serial.write(sim900.read());
  }
```

```
  sim900.println(postData);
  delay(10000);
  while (sim900.available()) {
    Serial.write(sim900.read());
  }
  Serial.println("METHOD: POST");
  sim900.println("AT+HTTPACTION=1"); //0 get 1 post
  delay(10000);
  while (sim900.available()) {
    Serial.write(sim900.read());
  }

  sim900.println("AT+HTTPREAD");
  delay(2000);
  while (sim900.available()) {
    Serial.write(sim900.read());
  }

  sim900.println("AT+HTTPTERM");
  delay(2000);
  while (sim900.available()) {
    Serial.write(sim900.read());
  }

  Serial.println("Desactivando GPRS...");
  sim900.println("AT+SAPBR=0,1");
  delay(2000);
  while (sim900.available()) {
    Serial.write(sim900.read());
  }
}
```

Algoritmo 4.13: Código del envío de petición HTTP y JSON de datos a la API

Capítulo 5

Conclusiones y trabajo futuro

Con estas tres partes desarrolladas se alcanzan los objetivos marcados en el TFT01, dando como resultado un Producto mínimo viable. El desarrollo de este proyecto ha implicado la creación de varios componentes, incluyendo productos IoT, y ha requerido adquirir habilidades tanto en programación Backend como Frontend.

El uso de distintos tipos de arquitecturas dota a este proyecto de un carácter profesional en el área de la ingeniería del software. Además se han optimizado los recursos haciendo uso de buenas prácticas y herramientas nativas.

Una de las líneas futuras que se plantea es que el producto se pueda comercializar para lo cual es necesario realizar un estudio de mercado. Esta versión comercial se plantea estableciendo suscripciones además de la venta y mantenimiento del servicio del localizador.

Estas son las funcionalidades planteadas para cada rol de usuario por el pago de su suscripción:

Veterinario

1. Publicación de servicios que ofrece como peluquería, tienda, etc.
2. Acceso a análisis estadísticos de los datos recolectados del sistema.
3. Ofrecer otros servicios a los usuarios potenciales.

Usuario

1. Análisis de las rutas de sus mascotas si tiene el localizador.
2. Análisis de datos clínicos de las mascotas.

Además, en un futuro se pueden lanzar versiones mejoradas del localizador, versiones premium que incluyen más características como algún zumbador y conexión por Bluetooth

por si se encuentran en alguna zona sin cobertura pero sí dentro del alcance del Bluetooth. De esta forma se podrá guiar en la búsqueda de la mascota. Dicho Hardware supone una mayor inversión ya que es una versión superior.

Capítulo 6

Bibliografía

6.1. Proyecto iOS GitLab: https://gitlab.com/pawfind-pal/ios

6.2. Proyecto API GitLab: https://gitlab.com/pawfind-pal/backend

6.3. Proyecto Localizador GitLab: https://gitlab.com/pawfind-pal/arduino-location

6.4. Alamofire: https://github.com/Alamofire/Alamofire

6.5. Codescanner: https://github.com/masashi-sutou/CodeScanner

6.6. Swift QR Code Scanner: https://github.com/vinodiOS/SwiftQRCodeScanner

6.7. SIM900 Documentación: https://www.espruino.com/datasheets/SIM900_AT.pdf

Bibliografía

[1] Arquitectura Hexagonal - Medium (17 de Abril de 2024). https://medium.com/@edusalguero/arquitectura-hexagonal-59834bb44b7f.

[2] Builders - Refactoring Guru (22 de Abril de 2024). https://refactoring.guru/es/design-patterns/builder.

[3] Core Data Documentation - Apple (26 de Marzo de 2024). https://developer.apple.com/documentation/coredata.

[4] Delegate Documentation - Apple (26 de Marzo de 2024). https://developer.apple.com/documentation/uikit/uiapplication/1622936-delegate.

[5] Docker Documentation - Docker (24 de Marzo de 2024). https://docs.docker.com.

[6] El ORM como herramienta eficiente de trabajo - ESIC (28 de Abril de 2024). https://www.esic.edu/rethink/tecnologia/el-orm-como-herramienta-eficiente-de-trabajo.

[7] ExpressJS Documentation - Express (23 de Marzo de 2024). https://expressjs.com/es/4x/api.html.

[8] Formato JSON - IBM (30 de Abril de 2024). https://www.ibm.com/docs/es/baw/23.x?topic=format

[9] HTTP - MDN (17 de Abril de 2024). https://developer.mozilla.org/es/docs/Web/HTTP.

[10] HTTP cookies - MDN (22 de Abril de 2024). https://developer.mozilla.org/es/docs/Web/HTTP/Cookies.

[11] JSON Web Tokens - Auth0 (16 de Abril de 2024). https://auth0.com/docs/secure/tokens/json-web-tokens.

[12] JSON Web Tokens - JWT (09 de Abril de 2024). https://jwt.io.

[13] Life Cycle Documentation - Apple (26 de Marzo de 2024). https://developer.apple.com/documentation/uikit/app$_a nd_e nvironment/managing_y our_a pp_s l if e_c y$

[14] Map Documentation - Apple (26 de Marzo de 2024). https://developer.apple.com/documentation/mapkit.

[15] MVC - MDN (16 de Abril de 2024). https://developer.mozilla.org/es/docs/Glossary/MVC.

[16] Producto viable mínimo - Wikipedia (16 de Abril de 2024). https://es.wikipedia.org/wiki/Producto$_{v}iable_{m}ínimo$.

[17] Programación orientada a objetos - IBM (10 de Abril de 2024). https://www.ibm.com/docs/es/spss-modeler/saas?topic=language-object-oriented-programmi

[18] Qué es SCRUM - proyectosagiles (17 de Abril de 2024). https://proyectosagiles.org/que-es-scrum/.

[19] Redis Documentation - Redis (24 de Marzo de 2024). https://redis.io/docs/latest/.

[20] Sequelize Documentation - Sequelize (29 de Abril de 2024). https://sequelize.org.

[21] Sesión informática - Wikipedia (19 de Abril de 2024). https://es.wikipedia.org/wiki/Sesión$_{(}informática$).

[22] SQLite - SQLite (17 de Abril de 2024). https://sqlite.org/index.html.

[23] StoryBoard Documentation - Apple (25 de Marzo de 2024). https://developer.apple.com/documentation/uikit/uistoryboard.

[24] Swift Documentation - Swift (25 de Marzo de 2024). https://www.swift.org/documentation/.

[25] Thread Documentation - Apple (26 de Marzo de 2024). https://developer.apple.com/documentation/dispatch/dispatchqueue.

[26] Token informática - Wikipedia (04 de Abril de 2024). https://es.wikipedia.org/wiki/Token$_{(}informática$).

[27] UIkit Documentation - Apple (25 de Marzo de 2024). https://developer.apple.com/documentation/uikit.

[28] User Defaults Documentation - Apple (26 de Marzo de 2024). https://developer.apple.com/documentation/foundation/userdefaults.

[29] ¿Qué es MySQL - Oracle (10 de Abril de 2024). https://www.oracle.com/es/mysql/what-is-mysql/.

[30] ¿Qué es una API y cómo funciona? - RedHat (02 de Abril de 2024a). https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods.

[31] ¿Qué es una API y cómo funciona? - RedHat (02 de Abril de 2024b). https://www.redhat.com/es/topics/api/what-are-application-programming-interfaces.

[32] ¿Qué es una URL? - MDN (02 de Abril de 2024). https://developer.mozilla.org/es/docs/Learn/Common$_{q}uestions/Web_{m}echanics/What_{i}s_{a}URL$.

Glosario

API Traducido al español como "Interfaz de Programación de Aplicaciones, es una pieza de código que permite a diferentes aplicaciones comunicarse entre sí y compartir información y funcionalidades" [31]. iv, 2–5, 7, 10–12, 15, 17, 20, 25, 27–29, 32, 34, 63–66

Arquitectura Hexagonal Dicha arquitectura consiste en intentar tener de forma desacoplada las distintas capas del servicio. Con lo que se consgue tener independencia entre las tecnologías que tiene cada capa, por lo que si en un futuro se decide, por ejemplo, cambiar de MySQL a SQLite, simplemente se debe cambiar dicha implementación, el resto de capas no se ven afectadas ya que están desacopladas mediante interfaces. vi, 6, 11, 15, 17, 25–28

Base de Datos Una base de datos puede ser relacional o no relacional, aunque su objetivo es el mismo, almacenar los datos de forma que se mantengan aunque el dispositivo se apague. vi, 2, 3, 5, 6, 10–14, 17, 18, 22–25, 28, 35, 65

Cookies "Es una pequeña información enviada por el sitio web y almacenada en el navegador del usuario, de manera que el sitio web pueda consultar la actividad previa del navegador" [10]. 12, 22

Create Read Update and Delete Es un término utilizado para referirse a las acciones de Crear, Leer, Actualizar y Borrar a un conjunto de datos. 2, 3, 5, 17, 20, 31, 36, 38, 41, 51, 52, 59

Hardware Es el término que se acuña a cualquier parte física de las informática, como por ejemplo el CPU, RAM, etc.. 2, 4, 5, 69

HTTP Traducido como: Protocolo de Transferencia de Hipertexto, "es un protocolo de la capa de aplicación para la transmisión de documentos hipermedia, como HTML" [9]. 12, 13, 22, 32, 66

HTTP Methods "Define un conjunto de métodos de solicitud para indicar la acción deseada que se realizará para un recurso determinado" [30]. 27

JSON "JavaScript Object Notation es un formato ligero de datas, es de fácil lectura y escritura para los usuarios. JSON es fácil de analizar y generar por parte de las máquinas" [8]. 20, 28, 33, 66

JWT "JSON Web Token es un standard abierto (RFC 7519) que define una forma compacta y autónoma de transmitir información de forma segura entre partes como un objeto JSON" [11]. 3, 6, 12, 32, 33

MVC Traducido como: Modelo Vista Controlador, "es un patrón en el diseño de software comúnmente utilizado para implementar interfaces de usuario, datos y lógica de control, enfatizando la separación entre la lógica de negocios y su visualización" [15]. VI, 6, 12, 18, 35, 36

MySQL "Es un sistema de gestión de bases de datos relacional bajo licencia pública y comercial por Oracle Corporation" [29]. 25, 26

OOP "Traducido como Programación Orientada a Objetos, se basa en el concepto de crear un modelo del problema destino en los programas." [17] . 20

ORM "Del acrónimo Object Relational Mapping, es un modelo de programación cuya misión es transformar las tablas de una base de datos de forma que las tareas básicas estén simplificadas" [6]. 10, 11, 13, 17

Patrón Builder "Permite crear objetos complejos paso a paso, permitiendo producir distintos tipos y representaciones de un objeto empleando el mismo código de construcción" [2]. 6, 12, 17, 34

Producto mínimo viable "Es un producto con suficientes características para satisfacer a los clientes iniciales, y proporcionar retroalimentación para el desarrollo futuro" [16]. 5, 68

Salt and Pepper Son términos que se refieren a la criptografía y consiste en lo siguiente: Para una contraseña se genera una Salt nueva a la que se le añade a modo de 'ruido', además para añadir más 'ruido' se le suma a esa combinación la Pepper. Ese conjunto se le hace un Hash y se almacena dicho resultado. 6, 11, 34

SCRUM "Es un proceso en el que se aplican de manera regular un conjunto de buenas prácticas para trabajar colaborativamente, en quipo y obtener el mejor resultado posible de un proyecto" [18]. 9

Sesiones "Es el intercambio de información interactiva semipermanente, también conocido como diálogo, conversación o un encuentro, entre dos o más dispositivos de comunicación" [21]. 12

Software Son el conjunto de instrucciones que debe hacer el CPU del dispositivo en el que se ejecuta para llevar a cabo una tarea que ha sido diseñada previamente. El software no es algo tangible a diferencia del hardware. 2, 7

SQLite "Es una librería escrita en C que implementa un motor de base de datos SQL pequeño, rápida, autónomo, alta fiabilidad y con todas las características" [22]. 25

Token "Es un proceso de sustitución de un elemento de datos sensible por un equivalente no sensible denominado token, que no tiene ningún significado o valor exportable" [26]. 32, 33

URL "Uniform Resource Locator, es una dirección que es dada a un recurso único en la Web. En teoría cada URL válida apunta a un recurso único. Dicho recurso pueden ser páginas HTML, documento CSS, imagenes, etc. " [32]. 28