ULPGC
Universidad de Las Palmas de Gran Canaria

ESCUELA DE INGENIERÍA INFORMÁTICA

ULPGC

Escuela de Ingeniería Informática

Grado en Ingeniería Informática

Trabajo de Fin de Grado

Aplicación web y móvil de citas para mascotas

AUTOR: Daniel Cruz López

TURORIZADO POR:

Luis Miguel Hernández Acosta

Enero de 2025

Agradecimientos

Quería agradecer a mi familia por haber confiado en que acabaría la carrera de ingeniería y ser un gran apoyo a lo largo de estos años. Sin ellos, habría carecido de la confianza para seguir. Les debo todo.

También quería agradecer a mi pareja, por haberme apoyado desde el inicio de estos estudios. Ha estado en mis buenos y malos momentos y ha sido mi fuerza para acabar esta ingeniería. Gracias de todo corazón.

Por último, agradecer a mi tutor Luis Hernández y a todos los profesores que generaron en mí un mayor interés por el mundo de la informática. Sin ello, difícilmente habría podido estudiar con motivación.

Muchas gracias a todos.

Resumen

PawMatch [14] es una aplicación web de citas para mascotas. Los usuarios cuentan con un perfil para la mascota y otro para el dueño. Pueden visualizar e interactuar con otros perfiles guardándolos para comprobar más adelante si les interesan o iniciar un chat para concretar encuentros en persona con sus mascotas.

Además, los usuarios pueden filtrar los perfiles que buscan según características de la mascota, como raza, sexo, edad, localización y más; o también por características del dueño.

PawMatch está diseñada para facilitar que encuentre el mejor acompañante para su mascota, así como para conectar con personas que también tengan mascotas y quieran pasar un buen rato juntos.

Abstract

PawMatch [14] is a pet dating web app. Users have a profile for pet and another for the owner. They can view and interact with other profiles by saving them to check later if they are interested or start a chat to arrange in-person meetings with their pets.

Additionally, users can filter the profiles they search for based on pet characteristics, such as race, sex, age, location and more; or also by characteristics of the owner.

PawMatch is designed to make it easy for you to find the best companion for your pet, as well as to connect with people who also have pets and want to have a good time together.

Índice general

Índice de tablas

Índice de ilustraciones

1 Introducción

Actualmente, las mascotas desempeñan un papel fundamental en la vida de las personas. Brindan compañía, apoyo emocional y crean vínculos profundos con sus dueños. Cabe mencionar que este cambio de percepción hacia los animales frente al pasado ha sido bastante notorio. De ser considerados simples mascotas a un miembro más en la familia, lo cual ha generado nuevas necesidades en torno a su cuidado, socialización y bienestar. Sin embargo, muchos dueños de mascotas enfrentan dificultades a la hora de satisfacer necesidades efectivas.

Por un lado, encontrar pareja para la reproducción de las mascotas sigue siendo un desafío, ya que no existe un medio que facilite estas conexiones de forma segura y eficiente. Además, situaciones comunes, como la necesidad de cuidadores temporales de confianza durante viajes o emergencias, también presentan obstáculos debido a la falta de plataformas dedicadas a este propósito.

No obstante, los problemas no solo surgen por parte de la mascota, el ser humano cada vez presenta más dificultades a la hora de establecer relaciones personales con otros individuos. Antiguamente, era más sencillo establecer contacto con otras persona dueñas de animales, sin embargo, hoy en día el ser humano se ha vuelto más antisocial, lo cual dificulta establecer esas relaciones.

La creación de una plataforma que englobe estas funciones y permita conectar de manera eficiente a dueños de mascotas con necesidades comunes no solo respondería a estas necesidad, sino que también crearía una comunidad digital comprometida con el bienestar animal.

1.1 Motivación

La motivación principal para la creación de esta **PawMatch**, la aplicación web y móvil, surge de una experiencia personal que marcó mi círculo cercano. Mi pareja tenía una mascota que, a pesar de los esfuerzos, nunca logró establecer contacto con un perro de su misma raza. Esto suponía una dificultad considerable, ya que no existía una manera sencilla de encontrar personas con mascotas compatibles. Lamentablemente, el perro falleció sin haber podido tener esa interacción.

Debido a ello, me he sentido atraído por la idea de crear un proyecto donde pudiera facilitar a las personas encontrar animales ideales para conectar con sus mascotas.

1.2 Estudio de mercado

Para este apartado, se ha analizado el funcionamiento de tres aplicaciones móviles: **MatchDog**, **Fetchdate** y **Pawmates**. Estas plataformas están enfocadas en ofrecer servicios que facilitan la socialización y las conexiones románticas, ya sea entre mascotas o entre humanos amantes de los animales.

Estas aplicaciones comparten similitudes con otras plataformas, como **Tinder**, ya que se basan en un sistema de perfiles donde el usuario busca opciones compatibles con sus intereses. Cuando existe un interés mutuo entre dos usuarios, se habilita la posibilidad de iniciar una conversación y concretar un encuentro. Sin embargo, lo primordial en estas aplicaciones es la mascota, promoviendo tanto su socialización como la de sus dueños.

A pesar de estas similitudes, cada aplicación tiene un enfoque particular:

- **MatchDog**: Está centrada en facilitar la reproducción controlada de perros.
- **Fetchdate**: Se basa en citas humanas, utilizando a las mascotas como enlace para fomentar conexiones.
- **Pawmates**: Busca promover la socialización de perros y dueños en entornos locales.

Sin embargo, estas plataformas suelen enforcarse en necesidades específicas y carecen de una integración completa que facilite múltiples servicios en un solo lugar.

Plataforma	Descripción	Fortalezas	Debilidades
MatchDog	Ayuda a encontrar pareja para la reproducción de peros.	Enfoque claro en reproducción controlada,	Limitada a perros; no abarca otras especies ni servicios adicionales.
Fetchdate	Aplicación de citas que utiliza a las mascotas como enlace para conectar amantes de animales.	Combina citas humanas y amor por las mascotas; incluye perfiles tanto de dueños como de animales.	Solo conecta personas con intereses románticos, no ofrece servicios para mascotas como cuidado.
Pawmates	Permite encontrar compañeros de juego para perros y socializar con otros dueños.	Fácil de usar, orientada a la socialización en áreas locales.	Exclusiva para perros, no abarca reproducción ni cuidado.

***Tabla 1**: Comparativa de Aplicaciones Existentes*

Aunque las aplicaciones vistas ofrecen soluciones interesantes, cada una de ellas está centrada en un aspecto muy específico de la interacción entre dueños de mascotas. Ninguna integra múltiples servicios como la reproducción controlada, citas, socialización y cuidado en un solo lugar. Esto crea una oportunidad para desarrollar una plataforma que abarque todas estas necesidades, facilitando un servicio más completo y accesible para los usuarios.

1.3 Objetivos

El objetivo principal del proyecto es desarrollar una solución tecnológica que facilite las citas entre mascotas y promueva la interacción entre sus dueños, ayudando tanto en la socialización de los animales como la creación de conexiones entre personas con intereses comunes.

La aplicación web permitirá a los usuarios gestionar perfiles detallados de sus mascotas, realizar búsquedas personalizadas y comunicarse mediante un sistema interno de mensajería. Además, integrará herramientas como la geolocalización para encontrar coincidencias cercanas.

Para los usuarios registrados como dueños, se establecerá la posibilidad de crear perfiles específicos para su mascota, incluyendo detalles como raza, edad y características relevantes. Asimismo, se garantizará la seguridad en el manejo de contraseñas y datos personales, protegiendo la privacidad de los usuarios.

Con esta solución, se busca cubrir necesidades específicas relacionadas con la socialización de mascotas y sus dueños, mientras se asegura un diseño funcional, seguro y accesible.

1.4 Estructura del documento

El documento se organiza en varios capítulos que detallan el desarrollo del proyecto desde la concepción de la idea hasta su implementación final. Los capítulos y su contenido son:

- El **Capítulo 1** introduce el contexto que motivó la creación de la aplicación, describiendo las necesidades identificadas en el mercado y el enfoque elegido para la plataforma. Además, se incluye un análisis del mercado actual de aplicaciones de citas para mascotas, así como los objetivos del proyecto, los cuales guían las decisiones y el desarrollo a lo largo del trabajo.
- El **Capítulo 2** se enfoca en las tecnologías y herramientas utilizadas en el proyecto. Se ofrece una visión detallada sobre los lenguajes de

programación, frameworks y bibliotecas seleccionadas, así como las herramientas de desarrollo y diseño visual que soportan la aplicación. Asimismo, se explican las APIs externas utilizadas y la gestión del proyecto.

- En el **Capítulo 3**, se lleva a cabo el análisis y diseño de la plataforma. Este capítulo describe los requisitos funcionales y no funcionales que completan el desarrollo del sistema. Además, se incluyen los mockups de las pantallas clave de la aplicación, como el registro, el inicio de sesión y el perfil de las mascotas, y se detalla la estructura de la base de datos.
- El **Capítulo 4** se dedica a la implementación del proyecto, abarcando la metodología de trabajo y los detalles sobre la creación y organización de las tablas de la base de datos. También se analiza la estructura general del proyecto y la conexión de los diferentes componentes, así como las funcionalidades clave implementadas.
- El **Capítulo 5** está destinado a las pruebas realizadas a lo largo del proceso de desarrollo.
- El **Capítulo 6** ofrece una guía detallada para el usuario, explicando cómo interactuar con la aplicación desde el registro hasta la gestión de los perfiles, el uso del mapa, las búsquedas, la mensajería y las notificaciones.
- El **Capítulo 7** se centra en la guía para desarrolladores, sobre cómo pueden acceder al proyecto y ejecutarlo.
- En el **Capítulo 8**, se presentan las conclusiones del proyecto, reflexionando sobre los logros obtenidos, los retos superados y la relación entre los objetivos iniciales y los resultados alcanzados. Se identifican, además, posibles líneas de mejora a futuro.

2 Tecnologías y herramientas SW

Antes de desarrollar la aplicación web y móvil, se seleccionaron las tecnologías y herramientas se software más adecuadas, buscando garantizar funcionalidad, eficiencia y facilidad de integración. Se priorizaron opciones que aseguraran compatibilidad con el desarrollo tanto web como móvil.

2.1 Lenguajes de programación

- **HTML**:
  Para definir la estructura del contenido.

- **CSS**:
  Para definir el estilo del contenido.

- **TypeScript** [3]:
  Lenguaje de programación utilizado para facilitar el desarrollo lógico de la aplicación con características avanzadas y soporte para objetos complejos.

2.2 Frameworks y bibliotecas

- **Angular** [12]:
  Fue el framework utilizado para el proyecto. Bastante robusto y escalable, permitió la construcción de la interfaz y la lógica de la aplicación web y móvil. Node.js fue utilizado como entorno de ejecución para gestionar las herramientas necesarias durante el desarrollo. Permitió ejecutar la consola de Angular para construir, servir y probar la aplicación. Además, algunas dependencias como **@angular/forms**, **@angular/router** o **@angular/Google-maps** fueron clave para el manejo de formularios reactivos, navegación entre vistas o integrar mapas en la interfaz.

- **Ngx-sonner** [6]:
  Es una biblioteca de Angular que se utilizó para mostrar notificaciones simples en pantalla. Sobre todo, se utilizó como medio para mostrar mensajes emergentes de confirmación, error y advertencia al usuario.

2.3 Base de datos

El manejo de datos se realizó utilizando **Firebase**, específicamente su módulo **Firestore**, una base de datos en tiempo real que simplifica el almacenamiento y recuperación de datos. Además, para la gestión de usuarios, se empleó **Firebase Authentication**, que ofrece un método seguro de inicio de sesión.

2.4 Herramientas de desarrollo

Para la escritura del código, gestión y control de versiones y creación de componentes visuales se utilizaron una serie de herramientas:

- **Webstorm**:
  Es un entorno desarrollo diseñado específicamente para el desarrollo web. Es un buen soporte para Angular, lo que permitió organizar el proceso de desarrollo.

- **Git**:
  Es un sistema de control de versiones que permite registrar cambios en el código fuente.

- **Github**:
  Plataforma basada en Git que permite almacenar proyectos de forma remota. Además de servir como repositorio del proyecto, facilitó la integración de ramas y revisión de cambios.

- **TeleportHQ**:
  Herramienta de desarrollo visual que permite diseñar y generar código base para componentes web. En el proyecto, se utilizó para la creación inicial de la interfaz de usuario del sitio web. Fue muy útil para diseñar botones, formularios y otros componentes de la interfaz, proporcionando una base que se luego se adaptó y personalizó en el código.

2.5 Diseño visual

El diseño de la interfaz y los mockups se realizó con **Figma**, una herramienta que permite crear prototipos y obtener una visión clara del producto final antes de su desarrollo. Aunque el sitio web no quedó exactamente igual que los diseños hechos en Figma, la herramienta es muy útil para tener una guía de cómo hacer el diseño.

2.6 APIs externas

- **Google Maps JavaScript API y Google Places API [5]**:
  Estas APIs se utilizaron para ofrecer funcionalidades relacionadas con geolocalización y búsqueda de lugares. **Google Places API** fue fundamental para autocompletar campos de formulario relacionados con ciudades y direcciones. Por otro lado, **Google Maps JavaScript API** ofreció el servicio de geolocalización en el mapa de Google Maps, necesario para visualizar a los usuarios.

- **Firebase APIs**:
  Incluyen **Firebase Authentication**, **Firestore** y **Storage**.

2.7 Gestión de proyectos

La planificación y organización del proyecto se llevaron a cabo utilizando **Trello**, una herramienta ágil que permitió gestionar las tareas y seguir el progreso de las distintas Sprint del desarrollo.

3 Análisis y Diseño

Este capítulo se centra en la fase de análisis y diseño de la aplicación, definiendo las bases para el desarrollo del proyecto. A lo largo del capítulo, se definen los requisitos necesarios para garantizar que la aplicación web cumpla con los objetivos planteados anteriormente, se diseña la interfaz de usuario mediante los mockups y se detalla la estructura de la base de datos.

3.1 Requisitos

A continuación, se presentan los requisitos funcionales y no funcionales definidos para cumplir con las necesidades y expectativas de los usuarios.

3.1.1 Requisitos funcionales

Se presentan los requisitos funcionales que especifican las tareas y actividades que el sistema debe ser capaz de realizar. A medida que ha avanzado el desarrollo de la aplicación, se han añadido o modificado ciertos requisitos.

Número de requisito	**RF1**
Nombre de requisito	Gestión de perfiles
Fuente de requisito	Cliente/Usuario
Prioridad del requisito	Alta
Descripción del requisito	Los usuarios deben poder registrarse, crear y editar perfiles para ellos y para sus mascotas. Esto incluye cargar recursos multimedia, definir características (edad, raza, género) y actualizar información en cualquier momento.
Validación del requisito	- Comprobar que los usuarios puedan registrarse, crear y editar su perfil. - Verificar que los datos de la mascota (edad, raza, género)

	pueden ser introducidos y modificados. - Asegurar que es posible cargar recursos multimedia (fotos, vídeos). - Probar que los cambios en los perfiles se reflejan de forma inmediata en el sistema.

Tabla 2: Requisito Gestión de perfiles

Número de requisito	**RF2**
Nombre de requisito	Registro de usuario
Fuente de requisito	Cliente/Usuario
Prioridad del requisito	Alta
Descripción del requisito	Los usuarios deben poder registrarse en el sistema proporcionando su email y contraseña.
Validación del requisito	El sistema validará si los datos son correctos, mostrando un mensaje de error si hay problemas (email ya registrado, credenciales incorrectas, etc.).

Tabla 3: Requisito registro de usuario

Número de requisito	**RF3**
Nombre de requisito	Inicio de sesión de usuario
Fuente de requisito	Cliente/Usuario
Prioridad del requisito	Alta
Descripción del requisito	Los usuarios deben poder iniciar sesión en la plataforma utilizando su email y contraseña.
Validación del requisito	Las credenciales deben tener el formato correcto, el usuario sería redirigido a la página principal. En caso contrario se mostrará un mensaje de error.

Tabla 4: Requisito Inicio de sesión

Número de requisito	RF4
Nombre de requisito	Test de prioridad
Fuente de requisito	Cliente/Usuario
Prioridad del requisito	Baja
Descripción del requisito	Al registrarse, el sistema deberá hacer preguntas para determinar qué tipo de interacción busca el usuario (amor para su mascota, cuidador, etc.).
Validación del requisito	El sistema debe permitir que el usuario seleccione un interés y guardarlo en su perfil.

Tabla 5: Registro test de prioridad

Número de requisito	RF5
Nombre de requisito	Motor de búsqueda de coincidencias
Fuente de requisito	Cliente/Usuario
Prioridad del requisito	Alta
Descripción del requisito	El sistema debe proporcionar un motor de búsqueda que permita a los usuarios filtrar y buscar coincidencias según criterios como raza, ubicación, género y preferencias de la mascota o dueño.
Validación del requisito	- Verificar que los filtros funcionan correctamente. - Comprobar que las búsquedas devuelven resultados relevantes en función de los criterios seleccionados. - Asegurar que el sistema muestra un mensaje adecuado cuando no hay coincidencias.

Tabla 6: Registro motor de búsqueda

Número de requisito	RF6
Nombre de requisito	Mensajería interna
Fuente de requisito	Cliente/Usuario
Prioridad del requisito	Media
Descripción del requisito	El sistema debe permitir a los usuarios enviar y recibir mensajes entre sí, para coordinar encuentros o compartir

	información adicional sobre sus mascotas.
Validación del requisito	- Asegurar que los usuarios pueden enviar mensajes y que estos se reciben correctamente. - Comprobar que los mensajes son visibles en un formato organizado (chat y lista).

Tabla 7: Requisito mensajería interna

Número de requisito	**RF7**
Nombre de requisito	Geolocalización de coincidencias
Fuente de requisito	Cliente/Usuario
Prioridad del requisito	Media
Descripción del requisito	El sistema debe utilizar un servicio de geolocalización, como Google Maps, para mostrar las coincidencias cercanas a la ubicación del usuario. Esto permitirá a los dueños de mascotas visualizar posibles encuentros cercanos y facilitar la interacción.
Validación del requisito	- Comprobar que la ubicación del usuario se detecta correctamente. - Verificar que las coincidencias cercanas se muestran en el mapa.

Tabla 8: Requisito geolocalización

Número de requisito	**RF8**
Nombre de requisito	Notificación push y alertas
Fuente de requisito	Cliente/Usuario
Prioridad del requisito	Baja
Descripción del requisito	El sistema debe enviar notificaciones automáticas a los usuarios sobre nuevas propuestas de chats.
Validación del requisito	Comprobar que las notificaciones se envían automáticamente al generarse un nuevo chat.

Tabla 9: Requisito notificación

Número de requisito	**RF9**
Nombre de requisito	Perfiles de mascotas
Fuente de requisito	Cliente/Usuario
Prioridad del requisito	Alta
Descripción del requisito	Cada usuario podrá crear el perfil para su mascota.
Validación del requisito	Verificar que los usuarios puedan crear el perfil para su mascota.

Tabla 10: Requisito perfil mascota

Número de requisito	**R10**
Nombre de requisito	Perfiles de usuarios
Fuente de requisito	Cliente/Usuario
Prioridad del requisito	Alta
Descripción del requisito	Cada usuario podrá crear su propio perfil.
Validación del requisito	Verificar que los usuarios pueden crear su perfil.

Tabla 11: Requisito perfil dueño

Número de requisito	**R11**
Nombre de requisito	Sistema de valoración
Fuente de requisito	Cliente/Usuario
Prioridad del requisito	Baja
Descripción del requisito	Los usuarios podrán valorar interacciones con otros usuarios proporcionando feedback. Estas valoraciones serán almacenadas y podrán ser visibles para otros usuarios.
Validación del requisito	- Comprobar que los usuarios pueden calificar o dejar una valoración tras interactuar con otro usuario. - Validar que el promedio de valoraciones se calcula correctamente y se muestre en el perfil. - Asegurar que solo los usuarios registrados pueden realizar valoraciones.

Tabla 12: Requisito sistema de valoración

Número de requisito	R12
Nombre de requisito	Logout
Fuente de requisito	Cliente/Usuario
Prioridad del requisito	Alta
Descripción del requisito	El sistema debe permitir a los usuarios cerrar sesión en cualquier momento.
Validación del requisito	- Verificar que, al cerrar sesión, el usuario es redirigido a la pantalla de inicio o login. - Asegurar que, tras cerrar sesión, no se puede acceder a ninguna funcionalidad protegida sin volver a autenticarse.

Tabla 13: Requisito logout

Número de requisito	R13
Nombre de requisito	Mapa de usuarios cercanos
Fuente de requisito	Cliente/Usuario
Prioridad del requisito	Media
Descripción del requisito	El sistema debe mostrar un mapa con la ubicación de los usuarios cercanos al usuario autenticado, con opciones para filtrar por distancia (1, 5 y 15 km).
Validación del requisito	El mapa debe mostrar usuarios cercanos y permitir hacer click en ellos para ver más detalles de su perfil.

Tabla 14: Requisito mapa

Número de requisito	R14
Nombre de requisito	Mensajes de confirmación o error
Fuente de requisito	Cliente/Usuario
Prioridad del requisito	Baja
Descripción del requisito	El sistema debe mostrar mensajes emergentes (en formularios, búsquedas, etc.) que indiquen si la acción se ha completado correctamente o si ha habido un error.
Validación del requisito	El sistema debe mostrar un mensaje de éxito o error que desaparezca automáticamente.

Tabla 15: Requisito mensajes emergentes

Número de requisito	R15
Nombre de requisito	Guardado de perfiles favoritos
Fuente de requisito	Cliente/Usuario
Prioridad del requisito	Media
Descripción del requisito	Los usuarios deben poder guardar perfiles de otros usuarios para acceder a ellos fácilmente más tarde.
Validación del requisito	Al hacer click en el botón “guardar”, el perfil del usuario se añadirá a una lista de usuarios guardados, accesible en el apartado “Saves”.

Tabla 16: Requisito guardado perfil

3.1.2 Requisitos no funcionales

A continuación, se presentan los requisitos no funcionales, los cuales garantizan que el sistema sea eficiente, seguro y capaz de satisfacer las expectativas del usuario en cuanto a calidad y experiencia.

Nombre	Rendimiento
Descripción	La mayoría de las transacciones de búsqueda deberían completarse en menos de un segundo.
Validación	Verificar que el tiempo de respuesta promedio para las búsquedas sea menor de 1 segundo bajo condiciones normales de carga.

Tabla 17: Requisito rendimiento

Nombre	Seguridad
Descripción	- Las contraseñas de los usuarios deben estar cifradas antes de almacenarse en la base de datos.

	- El sistema debe implementar autentificación mediante Firebase Authentication.
Validación	Probar el flujo de autenticación mediante Firebase Authentication y verificar que solo usuarios autenticados acceden a las funcionalidades protegidas.

Tabla 18: Requisito seguridad

Nombre	**Fiabilidad**
Descripción	EL sistema no debe tener prácticamente tiempo de inactividad.
Validación	Simular fallos en el sistema para verificar su capacidad de recuperación sin afectar la experiencia del usuario.

Tabla 19: Requisito fiabilidad

Nombre	**Disponibilidad**
Descripción	El sistema debe estar disponible durante todo el día.
Validación	Configurar alertas automáticas para responder rápidamente a cualquier caída del servicio.

Tabla 20: Requisito disponibilidad

Nombre	**Mantenibilidad**
Descripción	El sistema debe ser modular (uso de Angular) para permitir actualizaciones de características sin afectar a la funcionalidad existente.
Validación	La arquitectura es modular y cada componente está correctamente desacoplado.

Tabla 21: Requisito mantenibilidad

Nombre	Portabilidad
Descripción	La aplicación debe ser compatible con los navegadores más utilizados y con SO móviles.
Validación	- La aplicación funciona en los navegadores más usados. - Lo mismo para los SO móviles.

Tabla 22: Requisito portabilidad

3.2 Mockups

Durante la fase inicial del proyecto, se desarrollaron mockups con el objetivo de definir una visión preliminar de la interfaz de usuario y su estructura. Inicialmente, se realizaron bocetos básicos a mano, los cuales sirvieron como base para elaborar diseños más detallados utilizando la herramienta **Figma**. En esta etapa, se definieron aspectos clave como la paleta de colores, las formas de los componentes y el uso de imágenes.

Los mockups facilitaron la planificación de la interacción entre las distintas vistas del sitio web. A continuación, se presentan algunos de los diseños realizados.

3.2.1 Mockups a papel

Tantos la Ilustración 1 como la Ilustración 2 muestran los primeros bocetos de algunas de las páginas principales del sitio web. Cada ilustración compone cuatro páginas. Son bocetos sencillos con el objetivo de plasmar en un papel las primeras ideas planteadas antes de decidir aspectos más visuales como la paleta de colores, la forma de componentes o la selección de imágenes.

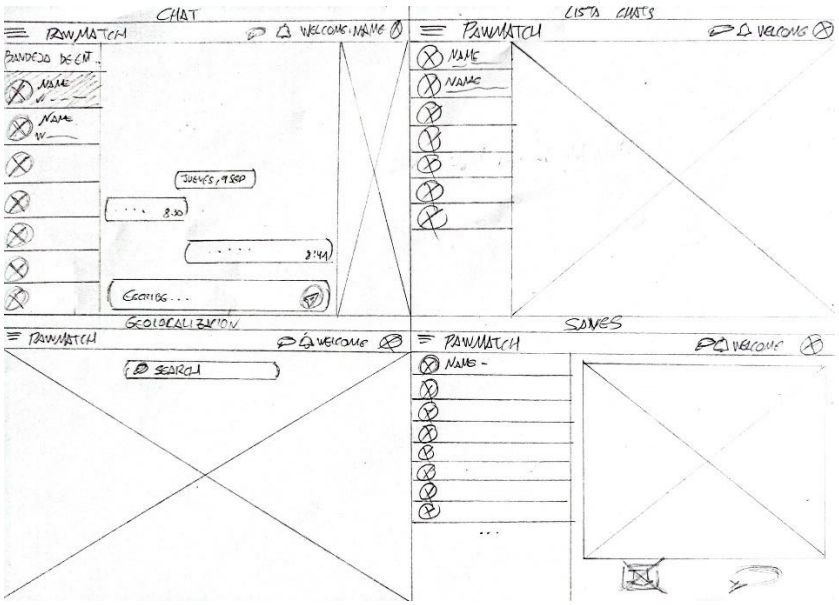

Ilustración 1: bocetos de páginas principales

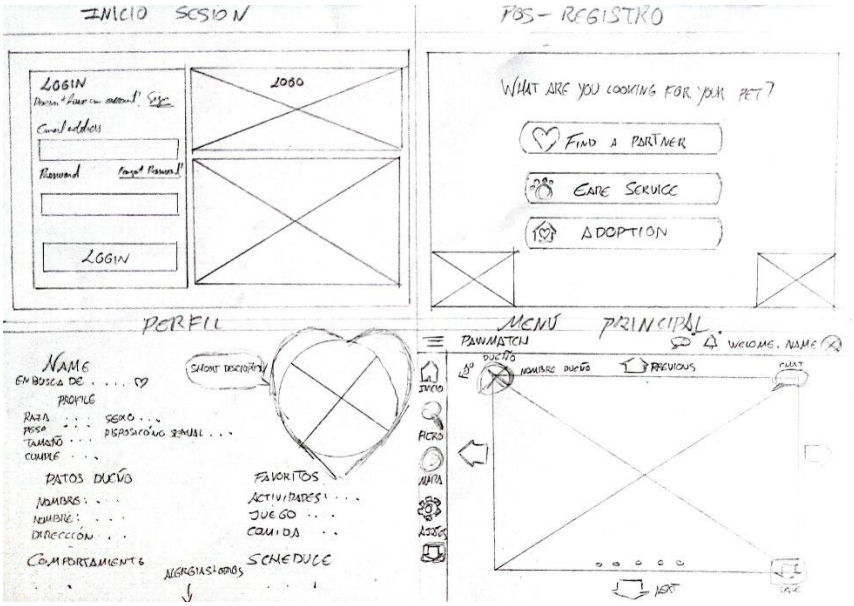

Ilustración 2: bocetos de páginas de formularios y perfiles

3.2.2 Mockup inicio de sesión

La página mostrada en la Ilustración 3 representa el inicio de sesión, que es la primera página que se le presenta al usuario. Si se quiere autenticar deberá rellenar el formulario.

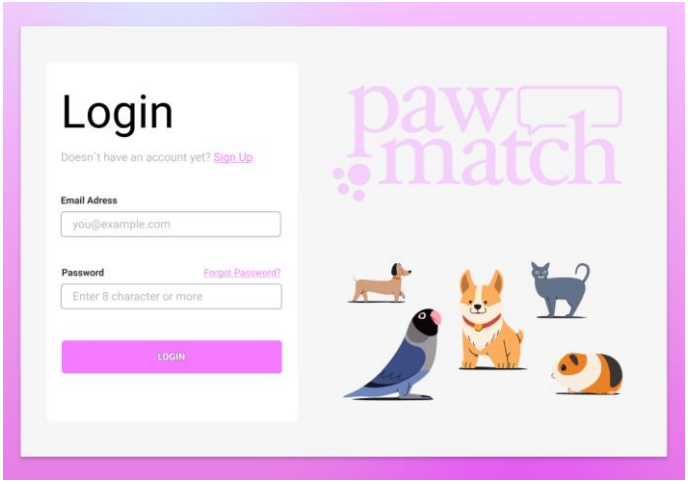

Ilustración 3: Mockup de la página de inicio de sesión

3.2.3 Mockup de formulario

Si el usuario se registra, será redirigido a la página mostrada en la Ilustración 4, la cual se basa en un formulario donde se deberá rellenar la información sobre la mascota y el dueño.

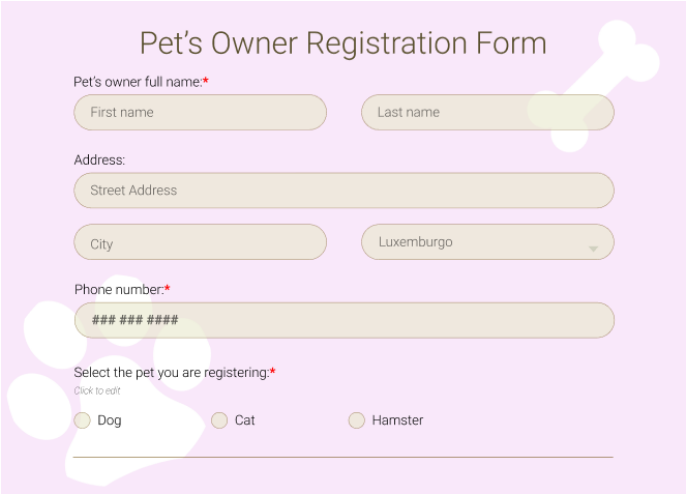

Ilustración 4: Mockup de la página del formulario

3.2.4 Mockup de home

En la Ilustración 5 se muestra la página de inicio, donde el usuario puede observar la información más relevante de un perfil, como el nombre, sexo, ubicación entre otros. Las flechas en los extremos permiten al usuario navegar para ver otros perfiles.

Además, los iconos en las esquinas de la imagen de perfil permiten al usuario chatear con la persona de dicho perfil o guardarlo en una sección personalizada.

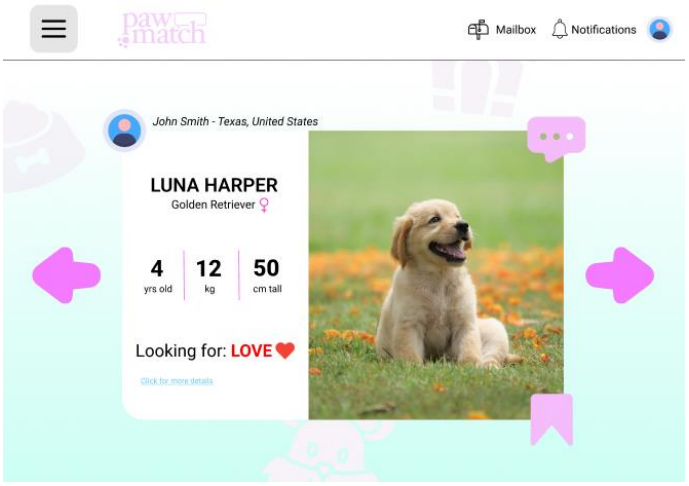

Ilustración 5: Mockup del home

3.2.5 Mockup de Perfiles Guardados

En la Ilustración 6 se muestra la página de perfiles guardados, donde se presenta una lista de perfiles guardados en el lado izquierdo. Si se selecciona uno de ellos, se despliega el contenido en el contenedor de la derecha.

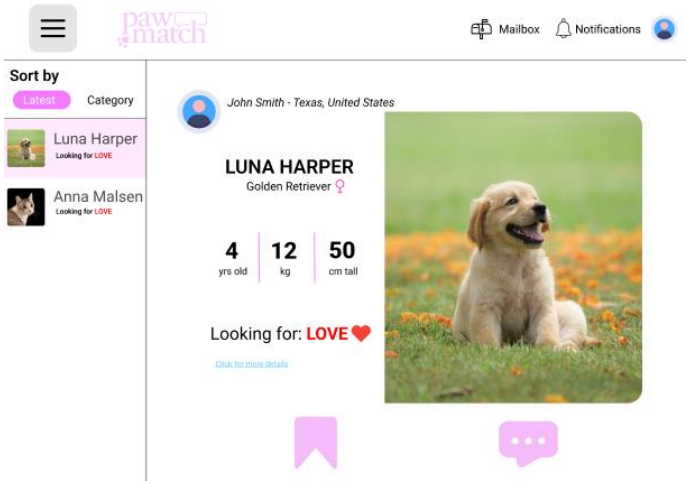

Ilustración 6: Mockup de Saves

3.2.6 Mockup de Mapa

En la Ilustración 7 se muestra la página del mapa. En ella, aparecen los perfiles cercanos a la posición del usuario. Si se selecciona un perfil, se despliega una pequeña ventana en la esquina inferior derecha.

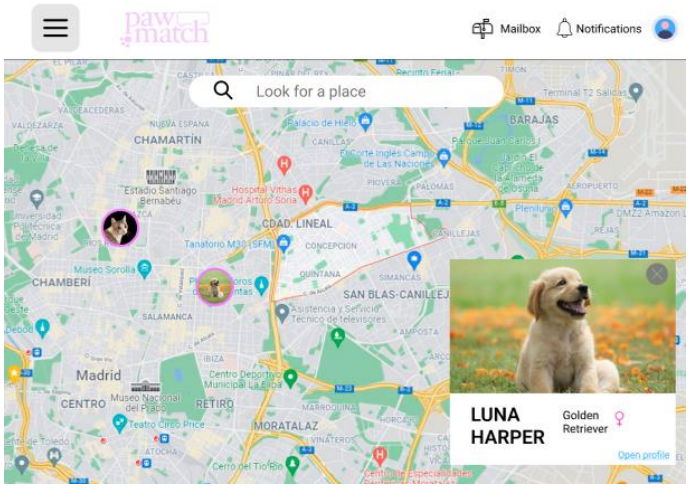

Ilustración 7: Mockup de Map

3.2.7 Mockup de Perfil Mascota

En la Ilustración 8 se muestra la página con los datos de la mascota del usuario. Permite modificar cualquiera de los campos deseados.

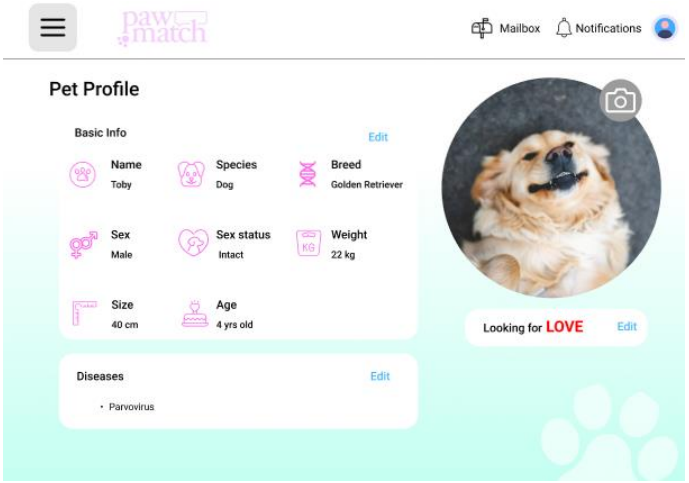

Ilustración 8: Mockup de Pet Profile

3.3 Estructura de BD

La elección de **Firebase** como base de datos se basa en su capacidad para ofrecer almacenamiento en tiempo real, lo cual es esencial para funcionalidades como el chat. Además, su naturaleza intuitiva y el hecho de que se ha utilizado en varias asignaturas de la carrera ofrecen cierta experiencia en su uso.

La estructura de la base de datos está representada por un archivo JSON, que muestra dos colecciones principales: **Users** y **Chats**, interconectadas mediante campos de identificación.

Colección Users

La colección **Users**, representada en la Ilustración 9, contiene varios campos de diferentes tipos, como cadenas de texto y números enteros. Un aspecto clave es el campo de **profilePerson**, que incluye subcampos con los datos del propietario del perfil. Además, **profilePet** es una lista que puede almacenar más de un perfil de mascota, y cada perfil tiene sus propios campos. También se incluye el campo **savedUsers**, que almacena los IDs de los perfiles guardados por el usuario.

Algunos campos en la colección **Users** son compartidos entre los perfiles de personas y mascotas, como la ubicación y el ID de usuario.

Colección Chats

La colección **Chats**, representada en la Ilustración 10, almacena información sobre las conversaciones entre los usuarios. En esta colección se encuentran campos que definen las características principales del chat, como el último mensaje enviado o si el chat está en espera o activo.

Dentro de **Chats**, también se encuentra una subcolección llamada **messages**, que guarda los mensajes individuales de cada chat.


```
{
  "Users": {
    "userId1": {
      "address": "123 Main St",
      "city": "New York",
      "country": "USA",
      "latitude": 40.7128,
      "longitude": -74.0060,
      "notification": "",
      "profilePerson": {
        "age": 30,
        "description": "Friendly and outgoing",
        "job": "Engineer",
        "name": "John Doe",
        "picture": "url_to_picture",
        "schedule": "Weekdays after 6 PM",
        "sex": "Male"
      },
      "profilePet": [
        {
          "age": 5,
          "breed": "Golden Retriever",
          "description": "Loyal and playful",
          "diseases": "None",
          "name": "Buddy",
          "picture": "url_to_picture",
          "search": "Companionship",
          "sex": "Male",
          "sexualStatus": "Neutered",
          "size": "Large",
          "species": "Dog",
          "weight": 30
        }
      ],
      "savedUsers": [
        "userId2",
        "userId3"
      ],
      "userId": "userId1"
    }
  },
```


Ilustración 9: JSON colección de usuarios

Ilustración 10: JSON colección de chats

4 Implementación

En este capítulo se explica el desarrollo del proyecto, empezando por la metodología que se seleccionó para organizar todo el trabajo. Más adelante se detalla el contenido de la base de datos, qué tipo de información guarda y cómo se organiza. Seguido se describe cómo está organizado el proyecto en general, explicando cómo se conectan los diferentes componentes, servicios y rutas. Por último, se realiza la explicación completa de cada uno de los componentes, sus funcionalidades y cómo se implementó para que todo funcione correctamente.

4.1 Metodología

En esta sección se detalla el proceso metodológico utilizado durante el desarrollo del proyecto. Para ello, se ha utilizado la metodología ágil **SCRUM**, que se adapta bien al desarrollo iterativo y al seguimiento continuo de las tareas. Además, ha sido una metodología bastante utilizada a lo largo de la carrera y se ajusta adecuadamente al proyecto.

Al inicio del proyecto, fue necesario definir las iteraciones que se llevarían a cabo a lo largo del desarrollo, la cuales se muestran en la Tabla 23:

Fases	Duración Estimada	Tareas
Planificación / Sprint 0	20h	Tarea 1.1: Definición de requisitos funcionales y no funcionales. Tarea 1.2: Análisis de tecnologías y herramientas a utilizar. Definición de backlog inicial.
Sprint 1 - Desarrollo inicial	80h	Tarea 2.1: Diseño de la arquitectura del sistema y base de datos. Tarea 2.2: Desarrollo inicial del backend (APIs básicas y estructura de datos). Daily Scrum para coordinar tareas.
Sprint 2 - Desarrollo y avance en funcionalidad	80h	Tarea 2.3: Desarrollo de funcionalidades específicas en frontend (IU y diseño responsivo). Daily Scrum y Sprint Review al final del sprint para revisión de avances.
Sprint 3 - Integración y validación	70h	Tarea 3.1: Pruebas unitarias y de integración de geolocalización y mensajería. Tarea 3.2: Validación de usabilidad. Sprint Retrospective al final para mejorar en el siguiente sprint.
Sprint 4 - Refinamiento y corrección	30h	Tarea 3.3: Corrección de errores detectados en pruebas y ajuste en la implementación. Daily Scrum para asegurar que los problemas se resuelvan ágilmente.
Documentación / Presentación	20h	Tarea 4.1: Redacción de la memoria final. Tarea 4.2: Creación de la presentación final para el funcionamiento del sitio web.

Tabla 23: Metodología SCRUM

Durante el desarrollo, se realizaron varias reuniones con el tutor para mostrar los avances, dudas y hacer correcciones. En estas reuniones se hacía un repaso de los requisitos cumplidos en cada sprint, los pendientes, y se definían las metas para la siguiente iteración.

Además de las reuniones con el tutor, se pidió la opinión de conocidos fuera del entorno académico para que probaran el prototipo y ofrecieran feedback sobre sus fortalezas y debilidades. Este feedback fue clave para guiar las mejoras en las siguientes iteraciones.

Para lograr el propósito de organizar el proyecto en base a la metodología SCRUM, se hizo uso de la herramienta **Trello**, que ofrece tableros Kanban para colocar los requisitos elaborados, así como posibles tareas extras. En la Ilustración 11 se puede observar el estado del Trello tras la finalización del desarrollo del proyecto. Como se puede observar, gran parte de los requisitos fueron completados; aunque otros quedaron por hacerse debido a la escasez de tiempo o porque se descartaron por no ser prioritarios.

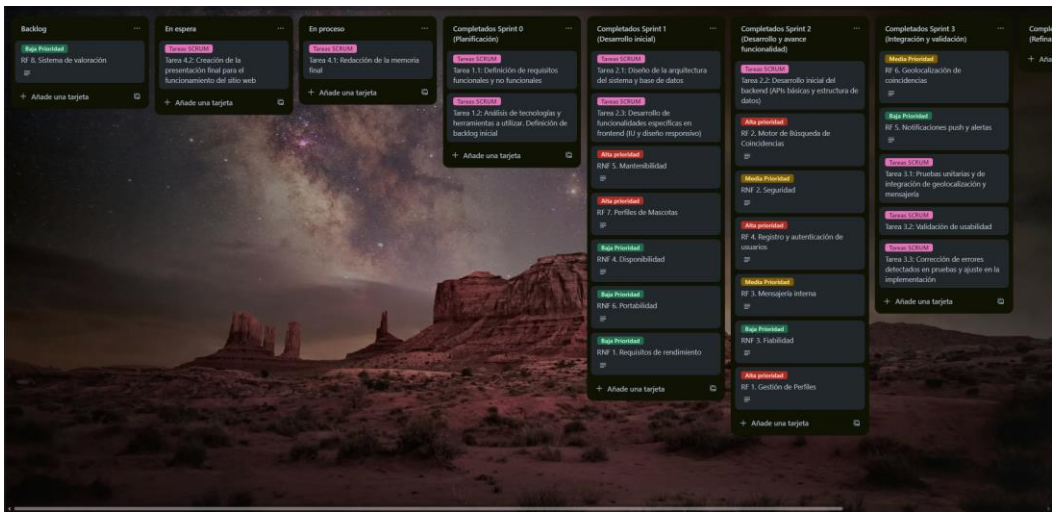

Ilustración 11: Tablero del proyecto en Trello

4.2 Contenido tablas BD

En esta sección se detalla el contenido y la organización de las colecciones y documentos dentro de la base de datos de *Firestore*, que es la utilizada para almacenar la información del sistema. Como se mencionó en la sección de estructura de la base de datos, esta se compone principalmente de dos colecciones: **Users** y **Chats**, las cuales están interconectadas mediante campos de identificación.

4.2.1 Colecciones y documentos

4.2.1.1 Usuarios

La colección **Users** contiene todos los datos relevantes de los usuarios y sus mascotas. En cada documento de esta colección se almacenan detalles tanto del propietario como de las mascotas, permitiendo la creación de perfiles completos.

Además, se incluyen los perfiles guardados por el usuario en el campo de **savedUsers**.

Los principales campos en esta colección son:

- **userId**: Identificador único generado por Firebase Authentication. Este campo permite identificar si el usuario actual está autenticado correctamente.
- **profilePerson**: Contiene los datos el propietario del perfil. Entre ellos el nombre, edad, sexo, ocupación, disponibilidad para quedar e imagen.
- **profilePet**: Lista que puede almacenar uno o más perfiles de mascotas para el usuario. Cada uno contiene sus propios campos, como el nombre, edad, sexo, raza, tamaño, peso, disposición sexual, descripción de enfermedades e imagen.
- **savedUsers**: Lista que almacena los perfiles guardados por el usuario. Este campo se compone de los IDs de documentos de otros usuarios.
- **Campos de ubicación**: Incluyen dirección, ciudad o país, latitud y longitud. Estos datos se obtienen mediante la API de Google Maps, facilitando la localización para funcionalidades como búsquedas cercanas.
- **Notificaciones**: Campo de tipo string que almacena el mensaje de la notificación. Si el campo está vacío, significa que no hay notificaciones pendientes. Este diseño es suficiente para el proyecto debido a que solo existe un único tipo de notificación en la plataforma.

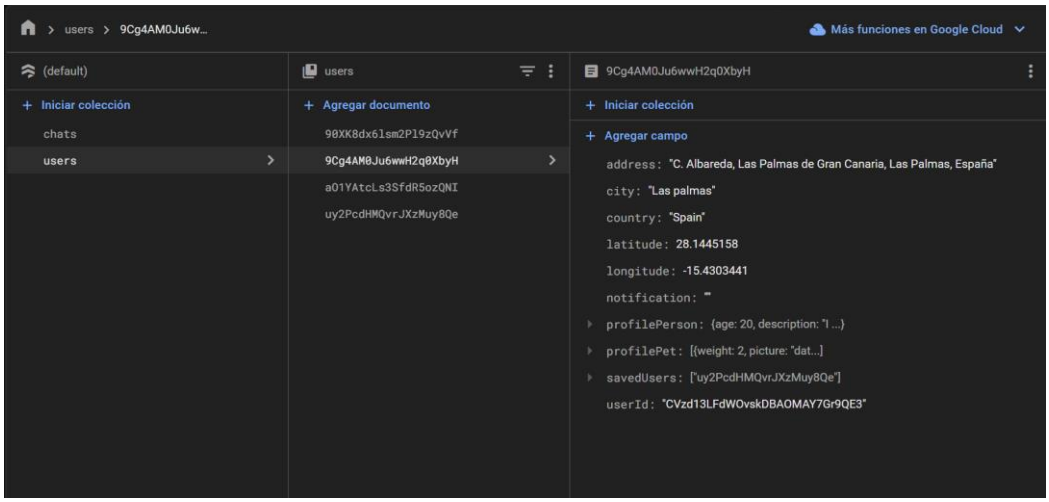

Ilustración 12: Estructura de la colección Users

4.2.1.2 Chats

La colección de **Chats** almacena la información sobre las conversaciones entre dos usuarios, ofreciendo los detalles más importantes para la interacción en tiempo real. En esta colección se encuentran los campos que definen el estado del chat y detalles sobre los mensajes asociados.

Los campos claves en esta colección son:

- **lastMessage:** campo que almacena el último mensaje enviado en la conversación. Este dato se utiliza principalmente con fines visuales.
- **lastMessageSender:** Campo que almacena el ID de documento del último usuario que envió un mensaje. Tiene el mismo fin que **lastMessage**.
- **state**: Indica el estado del chat, ya sea activo o no leído, lo que permite clasificar y facilitar la gestión de las conversaciones por parte del usuario.
- **lastMessageTimeStamp**: indica la fecha del último mensaje enviado.
- **participants**: Lista con los dos IDs de documento de los usuarios involucrados en la conversación.

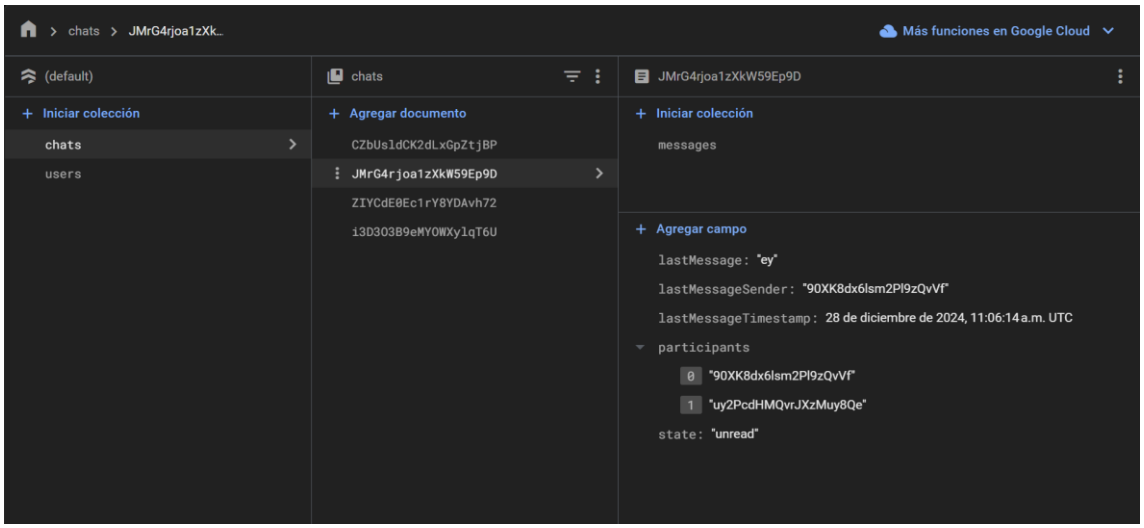

Ilustración 13: Estructura de la colección Chats

Además, dentro de cada documento de chat, se encuentra la subcolección messages, que almacena el conjunto de mensajes de la conversación, cada cual con sus respectivos campos:

- **message**: texto enviado en el mensaje.

- **timestamp:** Hora exacta en la que se envió el mensaje.
- **senderId**: el ID de documento del usuario que envió dicho mensaje.

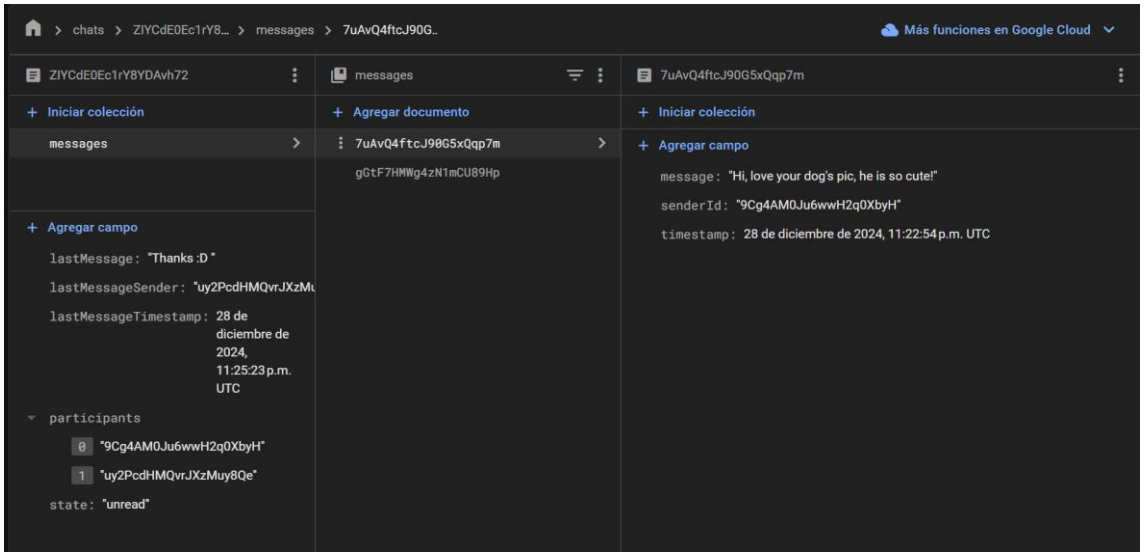

Ilustración 14: Estructura de la subcolección messages

4.2.1.3Relaciones entre colecciones

Las colecciones **Users** y **Chats** están interrelacionadas para garantizar una funcionalidad fluida del sistema:

- **Relación Usuario con Chats**: Los usuarios pueden mantener conversaciones con otros usuarios. Para esto, la colección Chats almacena en el campo participants los IDs de los documentos de los usuarios involucrados.
- **Relación Dueños y Mascotas**: Dentro de cada documento en la colección **Users**, los campos **profilePerson** y **profilePet** permiten estructurar la información del dueño y su/s mascotas. Esta organización facilita compartir información entre ambos perfiles, como la ubicación o los usuarios guardados, integrando una representación de un usuario como el conjunto de sus datos personales y los de sus mascotas.

4.3 Estructura y conexionado de componentes

A continuación, se describe la estructura del proyecto, la organización de los componentes, cómo interactúan entre sí y con el backend.

4.3.1 Estructura general del proyecto y arquitectura

Debido a cómo funciona Angular, el proyecto sigue una aproximación a la arquitectura **Model-View-View-Model (MVVM)** [43], aunque con ciertas variaciones. Esta arquitectura organiza el código en capas separadas:

- **Model**: En esta capa representamos los datos y su lógica. En este caso, se han definido las interfaces para **Users** y **Chats**, pero estas se encuentran dentro de los servicios de CRUD correspondientes. Esta decisión permite programar y acceder a los datos desde un único punto, facilitando la implementación y el mantenimiento del sistema.
- **View**: En la siguiente capa encontramos los componentes de la interfaz de usuario, encargados de mostrar los datos al usuario. Estos se encuentran en las carpetas **pages** y **componentes**.
    - **Pages**: Contiene los componentes que representan las páginas principales de la aplicación, como la página de registro, inicio de sesión, pantalla principal, perfil, entre otros. Aunque Angular utiliza un enfoque de aplicación de página única, facilita la organización y el trabajo con cada sección como si fueran páginas independientes.
    - **Components**: Incluye componentes reutilizables y más pequeños que no representan una página completa por sí solos, como el encabezado, notificaciones o cajas para mostrar perfiles. Estos componentes suelen integrarse dentro de los que se encuentran en la carpeta **pages**.
- **ViewModel**: En esta capa se encuentra la lógica de la aplicación y actúa como puente entre la vista y el modelo. En el proyecto, está representada por los servicios dentro de la carpeta **service** y **auth**, que encapsulan la lógica de negocio, interactúan con la base de datos y gestionan la autenticación.

Se puede observar en la Ilustración 15 cómo está representado el proyecto en carpetas. No es estrictamente la estructura de un MVVM, aunque es bastante aproximada. El motivo de la organización de carpetas fue debido al código generado por el software *TeleportHQ*.

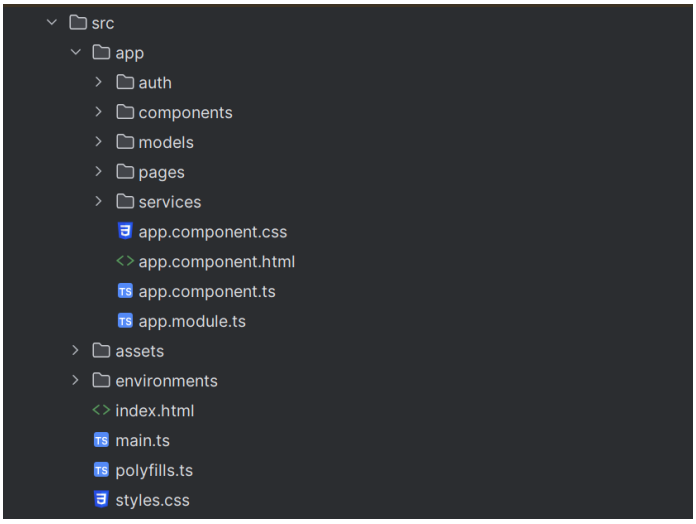

Ilustración 15: Estructura del proyecto

Entrando más en detalle en la estructura del proyecto, en la Ilustración 16 se muestran la carpeta **pages**, que contiene los componentes que representan las páginas principales de la aplicación.

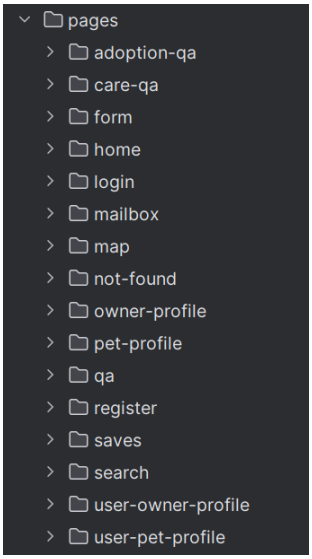

Ilustración 16: Estructura de pages

Luego nos encontramos con la carpeta de **components**, que contendrían los componentes reutilizables. Se muestra el contenido en la Ilustración 17.

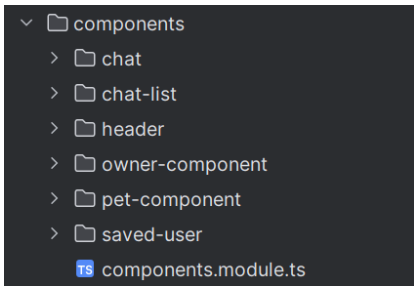

Ilustración 17: Estructura de Components

La estructura incluye una carpeta denominada **services**, mostrada en la Ilustración 18, que representa los servicios que gestionan la lógica de negocio y la interacción con la base de datos. Además, la carpeta de **auth** contiene los servicios específicos para la autenticación del usuario. Para mayor claridad en la representación del código, se ha decidido separar los servicios relacionados con *Firestore* con los relacionados con *Firebase Authentication*.

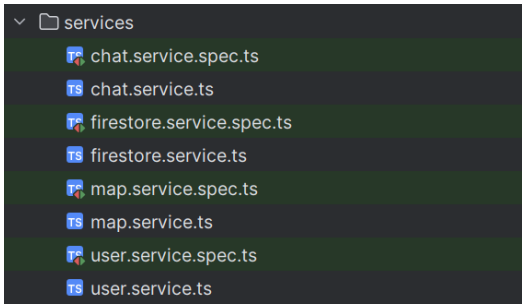

Ilustración 18: Estructura de servicios

Finalmente, se encuentran las carpetas de **models** y **assets**. La carpeta **models** incluye un archivo que enumera las razas de animales [8, 9, 10], organizado de manera que facilite su uso en la aplicación. La carpeta **assets** incluye todos los recursos utilizados por parte de la aplicación, como imágenes o iconos.

4.4 Código y funcionalidades

4.4.1 Funcionalidades principales

A continuación, se presentan las funciones principales que contiene la aplicación.

Registro e Inicio de Sesión

El componente **Register** y componente **Login** [4] tienen una funcionalidad bastante simular, ambos se encargan de gestionar el registro y la autenticación de usuarios.

El usuario rellena los campos de email y contraseña, los cuales son requeridos. Tras comprobar que los campos son válidos (en caso contrario se lanza un **Toast**, que es un mensaje emergente en pantalla), se llama al servicio de autenticación para registrar al usuario.

```
41
42    form : FormGroup<{email: FormControl<...  = this._formBuilder.group<FormRegister>( controls: {
43      email: this._formBuilder.control( formState: '',  validatorOrOpts: [
44          Validators.required,
45          Validators.email,
46          ]),
47      password: this._formBuilder.control( formState: '', Validators.required),
48    });
49
      1+ usages   Daniel Cruz López *
50    async register() {
51      if (this.form.invalid) return;
52      try {
53        const { email : string , password : string } = this.form.value;
54        if (!email || !password) return;
55        await this._authService.registerNewUser( user: {email, password});
56        toast.success( message: "User created successfully!");
57        await this._router.navigate( commands: ['/qa']);
58      } catch (error) {
59        toast.error( message: "Something was wrong with the registration");
60      }
61    }
62  }
```

Ilustración 19: Formulario de registro y login

Formulario de creación de perfil

El componente **Form** está disponible después del registro y permite crear el perfil de la persona y la mascota. Utiliza **FormBuilder** para recoger y validar los datos, asumiendo la gestión de campos adicionales y funcionalidades avanzadas debido a su mayor complejidad.

Uno de los campos más destacados del formulario es la ubicación, que utiliza la API de *Google Maps Places* para ofrecer autocompletado. En el código de la función que se aprecia en la Ilustración 20, se obtiene una instancia de la API que es guardada en la constante **cityAutocomplete** y se configura un evento que, cuando el usuario escriba en el campo de ciudad, se muestren las sugerencias y se guarde la ubicación seleccionada.

```
1+ usages  Daniel Cruz López
private getPlaceAutocomplete() {
  if (!google || !google.maps) {
    console.error('Google Maps API no está cargado correctamente');
    return;
  }
  const cityAutocomplete = new google.maps.places.Autocomplete(this.cityInput.nativeElement, {
    componentRestrictions: { country: 'es' },
    types: ['(cities)'],
  });

  cityAutocomplete.addListener('place_changed', () => {
    const place = cityAutocomplete.getPlace();
    if (place.geometry) {
      this.isAddressInputDisabled = false;
      this.setAddressAutocomplete(place.geometry.viewport);
    } else {
      console.log('No se han encontrado detalles para la ciudad seleccionada');
    }
  });
}
```

Ilustración 20: función de autocompletado de ciudades

Buscar coincidencias

Esta es la funcionalidad central de la aplicación: encontrar perfiles compatibles con las preferencias del usuario. El flujo de trabajo es el siguiente:

- En el componente **Home**, se cargan usuarios aleatorios mediante la función **loadUsers**, la cual se puede observar en la Ilustración 21. Esta función realiza una solicitud a la base de datos para obtener un número determinado de perfiles. Sin embargo, estos perfiles deben pasar un filtro antes de ser mostrados. Si ningún usuario pasa el filtro, la lista quedaría vacía.

- Si el usuario desea aplicar filtros en la búsqueda, puede ir a la página de **Search**. Allí encontrará un formulario con varios campos de filtro. Los datos que el usuario ingrese en este formulario se envían a **Home** y se aplican a los usuarios que se cargan nuevamente. Si un usuario no cumple con los filtros establecidos, es descartado automáticamente y no aparece en la lista final.

```
1+ usages  Daniel Cruz López
async loadUsers() {
  try {
    const userIds : string[]  = await this.firestoreService.getRandomUsers(this.usersList);
    if (!userIds) return;
    const filteredUsers : any[]  = [];
    for (const userId : string  of userIds) {

      const passesFilters : boolean  = await this.applyFilters(userId);
      if (passesFilters && userId !== this.userAuthDocId) {
        filteredUsers.push(userId);
      }
    }
    this.usersList.push(...filteredUsers);
    this.updateCurrentUser();
    this.isComponentLoading = false;
  } catch (error) {
    console.error('Error loading users: ', error);
  }
}
```

Ilustración 21: Función de carga de usuarios en pantalla

Guardar usuarios

Esta funcionalidad permite al usuario guardar perfiles de interés en una lista, que luego se visualiza en el componente **Saves**. Esto facilita mantener un registro de perfiles relevantes para futuras interacciones.

La lista de perfiles guardados no tiene un límite de capacidad en este momento, aunque esto podría ser una mejora futura. Además, los usuarios pueden eliminar los perfiles que ya no les interesen de la lista en cualquier momento.

Buscar por geolocalización

El proceso de geolocalización comienza obteniendo las coordenadas del usuario, necesarias para ubicarlo en el mapa. Este procedimiento se realiza en el componente **Map**, a través de la función **currentUserCoords**, donde guarda los datos que contienen los campos latitud y longitud del documento del usuario.

Posteriormente, la ubicación se visualiza en el mapa mediante la función **loadMap**, que utiliza las coordenadas obtenidas para centrar en el mapa en dicha posición. Además, se añade un marcador en la ubicación del usuario, empleando un color distintivo. Todo este procedimiento se realiza haciendo uso de la API de *Google Maps*.

En este componente también se generan marcadores para cada usuario registrado en la aplicación. Sin embargo, solo se muestran aquellos usuarios que se encuentran dentro de un radio determinado respecto a la ubicación central. La distancia entre los puntos se calcula en el servicio **mapService**, donde se implementa la *fórmula de harversine* [2].

En el código, la *fórmula de harversine* [2] toma las coordenadas geográficas de dos puntos (la latitud y la longitud) y las convierte a radianes mediante la función **degreesToRadians**. A continuación, calcula la diferencia de latitudes y longitudes entre los puntos y aplica la fórmula para obtener la distancia en kilómetros, teniendo en cuenta la forma esférica de la Tierra. Este valor se compara con el radio especificado para determinar si un marcador debe ser visible en el mapa. La implementación detallada de esta función se encuentra en la Ilustración 22.

```
1+ usages  Daniel Cruz López
calculateDistance(cord1: any, cord2: any): number {
  const R : 6371  = 6371; // Radio de la Tierra en km
  const dLat : number  = this.degreesToRadians( degrees: cord2.lat - cord1.lat);
  const dLng : number  = this.degreesToRadians( degrees: cord2.lng - cord1.lng);
  const a : number  =
      Math.sin( x: dLat / 2) * Math.sin( x: dLat / 2) +
      Math.cos(this.degreesToRadians(cord1.lat)) * Math.cos(this.degreesToRadians(cord2.lat)) *
      Math.sin( x: dLng / 2) * Math.sin( x: dLng / 2);
  const c : number  = 2 * Math.atan2(Math.sqrt(a), Math.sqrt( x: 1 - a));
  return R * c;
}
```

Ilustración 22: función para calcular distancia entre dos puntos geográficos

Chatear

El componente **Mailbox** gestiona las conversaciones de chat. Primero, obtiene el ID del usuario autenticado utilizando el servicio de autenticación **authStateService**. Con este ID, consulta las conversaciones asociadas mediante el **chatService**. Estas conversaciones se procesan para extraer información clave, como el último mensaje, su fecha, el estado del chat y el ID del otro participante.

La información de las conversaciones se gestiona mediante observables, lo que permite que cualquier cambio en la base de datos se refleje automáticamente en tiempo real. Por ejemplo, si el otro participante envía un mensaje, este se debería

mostrar instantáneamente en la interfaz de usuario, asegurando un chat en tiempo real. Este es uno de los mayores potenciales de usar *Firestore* como base de datos.

4.4.2Servicios

Autenticación

El servicio **authService** gestiona la autenticación de usuarios en la aplicación utilizando *Firebase Authentication*. Contiene dos métodos principales:

- **registerNewUser**: Crea una nueva cuenta de usuario con el correo electrónico y la contraseña proporcionados, usando el método **createUserWithEmailAndPassword** de *Firebase*.
- **loginExistingUser** [4]: Inicia sesión con un usuario existente mediante el correo electrónico y la contraseña, utilizando **signInWithEmailAndPassword** de *Firebase*.

El servicio se inyecta en los componentes donde se requiere funcionalidad de autenticación, y utiliza **AngularFireAuth** para interactuar con *Firebase*.

CRUD usuarios y mascotas

El servicio **userService** gestiona operaciones CRUD sobre los usuarios y sus mascotas en *Firestore*. Utiliza **AngularFirestore** para interactuar con la base de datos y **AuthStateService** para obtener el ID del usuario autenticado, asegurando que las operaciones estén vinculadas al usuario correspondiente.

Entre las responsabilidades clave del servicio, se encuentra la creación de usuarios, como se muestra en la Ilustración 23. Durante esta operación, se genera un nuevo documento en *Firestore*, asociándolo con el ID del usuario autenticado.

El servicio también incluye funciones como la actualización de los datos del perfil, donde se actualizan solo los campos proporcionados.

```
1+ usages  Daniel Cruz López
async createUser(user: UserCreate) {
  const currentUser : firebase.User  = await this._authState.currentUser;
  if (!currentUser) {
    throw new Error('No user is authenticated');
  }
  const newUser : {userId: string, city: string,...  = {
    ...user,
    userId: currentUser.uid,
  };
  return this._firestore.collection(PATH).add(newUser);
}
```

Ilustración 23: Función de creación del perfil

Aunque *Firestore* utiliza *observables* internamente para las operaciones de lectura, este servicio emplea *promesas* [1] en las operaciones de escritura, como la creación y actualización de usuarios. Estas acciones no requieren actualizaciones en tiempo real, y las *promesas* [1] permiten manejas dichas tareas asíncronas de forma directa y sencilla, dado que suelen ejecutarse una sola vez y no necesitan suscripción continua.

Servicio de consultas a la base de datos

Existen otros servicios como el **firestoreService** basado en observables [7], debido a que, la mayor parte de funciones de este se basan en operaciones de lectura, donde los datos de los usuarios pueden ir variando. En la Ilustración 24 se puede apreciar este comportamiento cuando se desea obtener los datos de la mascota o el dueño de un respectivo usuario pasándole el ID.

```
1+ usages  Daniel Cruz López *
getPets(userId: string): Observable<any[]> {
  return this.firestore.collection( path: 'users').doc(userId).valueChanges().pipe(
      map( project: (userData: any) => userData?.profilePet || [])
  );
}

1+ usages  Daniel Cruz López
getPerson(userId: string): Observable<any> {
    return this.firestore.collection( path: 'users').doc(userId).valueChanges().pipe(
        map( project: (userData: any | undefined) => userData?.profilePerson || null)
    );
}
```

Ilustración 24: Funciones de lectura de perfil de usuario

5 Tests

Para la realización de las pruebas de la aplicación, se utilizaron dos principales estrategias. En primer lugar, se emplearon herramientas básicas de depuración como **console.log** para verificar el correcto funcionamiento de las distintas funcionalidades del sistema y detectar posibles errores durante el desarrollo.

A lo largo del proceso de pruebas, se contó con la colaboración de usuarios cercanos, quienes probaron la aplicación en diferentes etapas de su desarrollo. Ellos brindaron retroalimentación sobre posibles fallos, problemas de usabilidad, errores que no se habían detectado previamente. Este proceso permitió realizar ajustes y mejorar la estabilidad y experiencia de usuario antes de continuar con el desarrollo.

6 Guía de Usuario

6.1.1 Registro

Para registrarse, el usuario debe completar un formulario sencillo, que se puede apreciar en la Ilustración 25, que consta de dos campos: correo electrónico y contraseña. El formato de las credenciales debe ser el indicado en el texto de ejemplo para poder acceder con éxito al formulario de creación de cuenta.

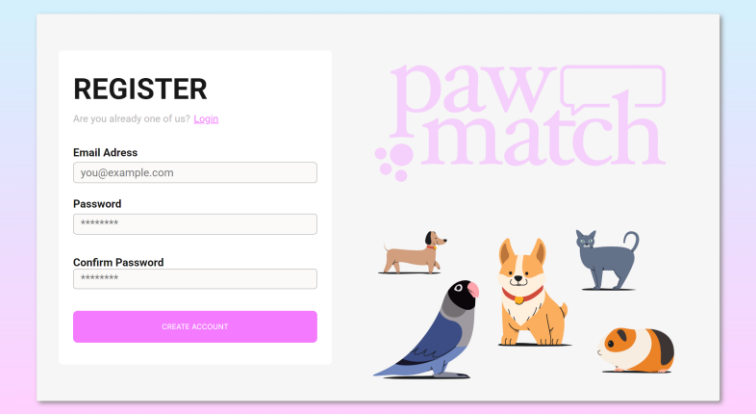

Ilustración 25: Página de registro

Cada página cuenta con su diseño responsivo, adaptado a todo tipo de pantallas: desktop, Tablet y móvil entre las más importantes. Para simplificar la presentación, se muestran únicamente los diseños para escritorio y móvil. La Ilustración 26 muestra cómo se visualiza el formulario de registro en dispositivos móviles.

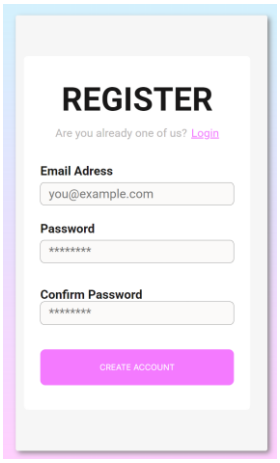

Ilustración 26: Página de registro en móvil

6.1.2 Formulario

En esta pestaña, el usuario debe completar una serie de campos para poder crear su perfil. Primero se solicitan los datos personales de la persona, como se muestran en la Ilustración 27. Algunos campos, como ciudad o dirección, cuentan con funcionalidades de autocompletado para facilitar el proceso y mejorar la experiencia del usuario.

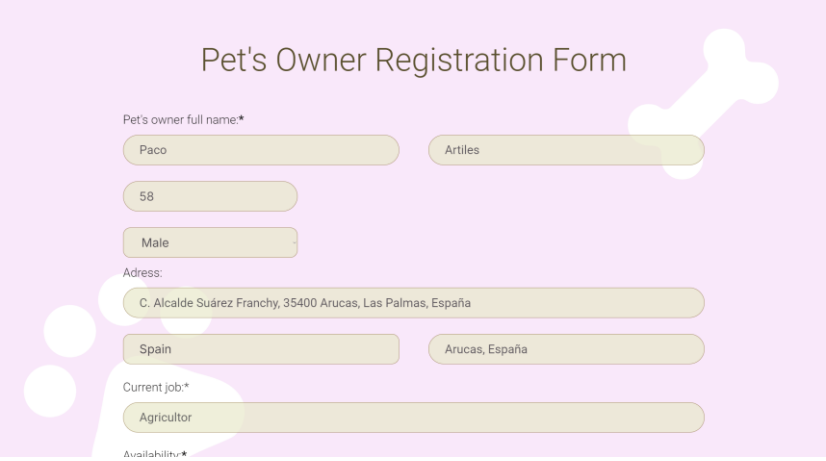

Ilustración 27: Página de formulario de creación de perfil

En la siguiente sección del formulario, se debe ingresar los datos de la mascota, como se puede observar en la Ilustración 28. No todos los campos son obligatorios, aquellos que sí lo son están marcados con un asterisco al lado del título correspondiente.

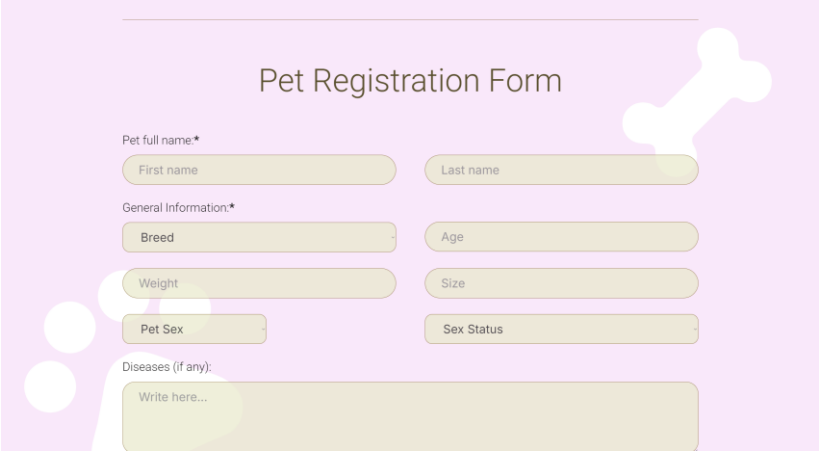

Ilustración 28: Página de formulario de creación de perfil

6.1.3Inicio

En la pestaña principal, se presentan los perfiles disponibles en la aplicación, uno a la vez. El usuario puede alternar entre visualizar perfiles de mascotas o de dueños utilizando un botón deslizante situado en la parte superior, como se muestra en la Ilustración 29.

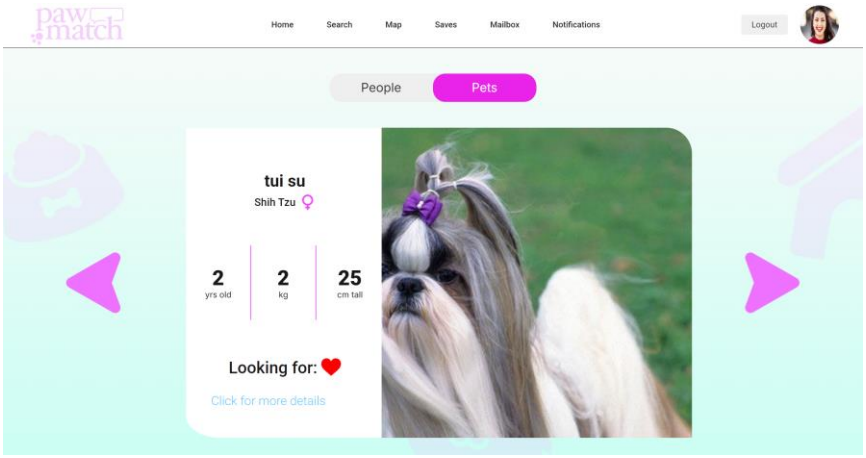

Ilustración 29: Página de inicio

Las flechas situadas en los extremos permiten navegar entre perfiles, ya sea hacia el siguiente o el anterior. Cada perfil muestra la información más relevante, aunque no todos los datos disponibles. Para acceder a más detalles sobre un perfil, se puede hacer click en el enlace ubicado en la parte inferior del componente.

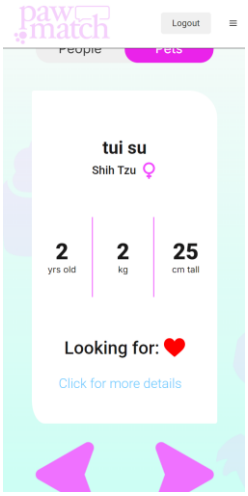

Ilustración 30: Página de inicio móvil

La navegación principal de la aplicación se encuentra en la cabecera de la pantalla, donde se incluye un menú interactivo. En dispositivos con pantallas de ancho reducido, el menú principal se oculta automáticamente, pero puede ser desplegado mediante un botón en la esquina superior derecha, como se ve en la Ilustración 30.

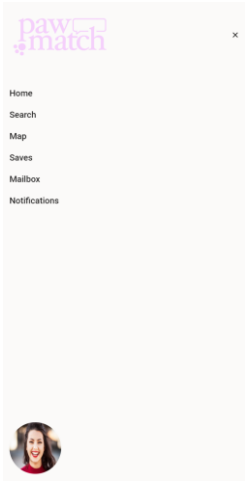

Ilustración 31: Menú de navegación móvil

6.1.4 Búsqueda

Para realizar un filtro de búsqueda más avanzado, el usuario puede acceder a la sección **Search**, mostrada en la Ilustración 32 en formato desktop y en la Ilustración 33 en formato móvil.

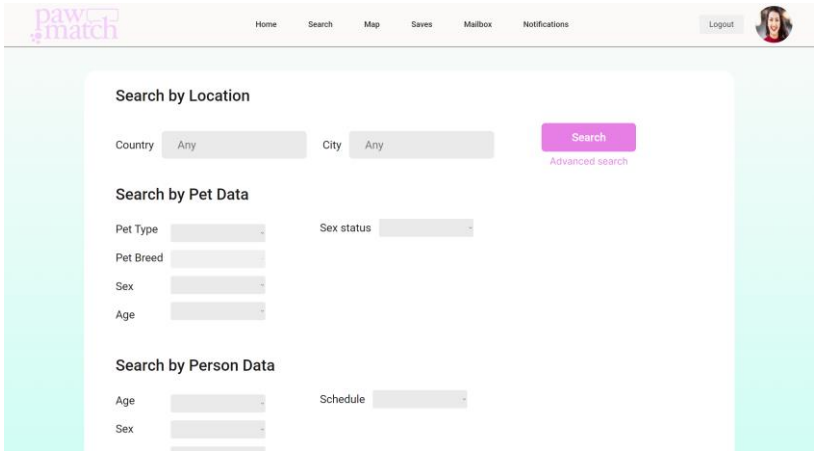

Ilustración 32: Página de filtros de búsqueda

En esta sección, el usuario tiene la posibilidad de seleccionar los campos deseados para aplicar un filtro. Actualmente, la aplicación permite establecer un único valor por cada campo de filtrado.

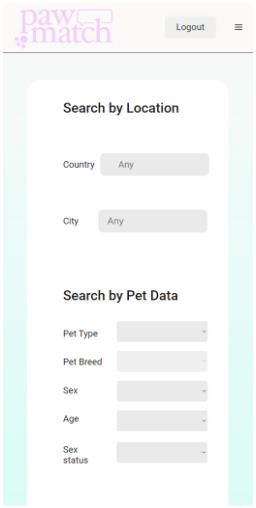

Ilustración 33: Página de filtros de búsqueda en móvil

Una vez completado el formulario de búsqueda, al presionar el botón correspondiente, el usuario será redirigido a la página principal, donde se mostrarán únicamente los perfiles que cumplan con los filtros aplicados. Si no se encuentran usuarios que cumplan con los criterios seleccionados, se mostrará un mensaje informando al usuario, como aparece en la Ilustración 34.

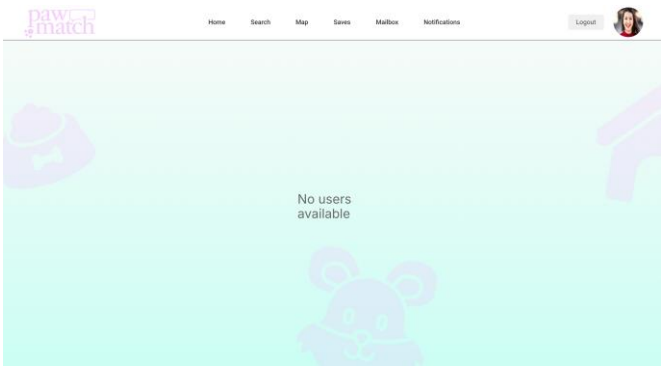

Ilustración 34: Página inicial sin usuarios disponibles

6.1.5 Perfil Persona

El usuario puede acceder a su perfil haciendo click en su imagen ubicada en la cabecera, tanto en formato desktop como móvil. En esta página, mostrada en la Ilustración 35 e Ilustración 36, se muestran todos los datos del usuario.

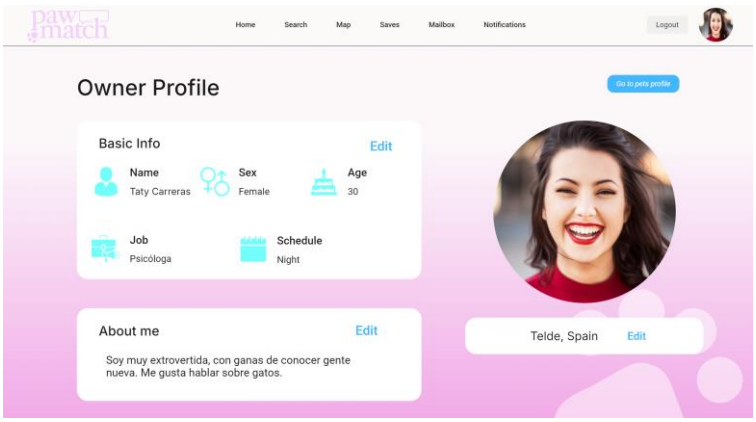

Ilustración 35: Página de perfil de la persona

Adicionalmente, el botón situado en la esquina superior derecha permite al usuario navegar hasta el perfil de su mascota.

Aunque los botones “Edit” están visibles en la interfaz, actualmente esta funcionalidad para modificar los datos aún no se encuentra implementada.

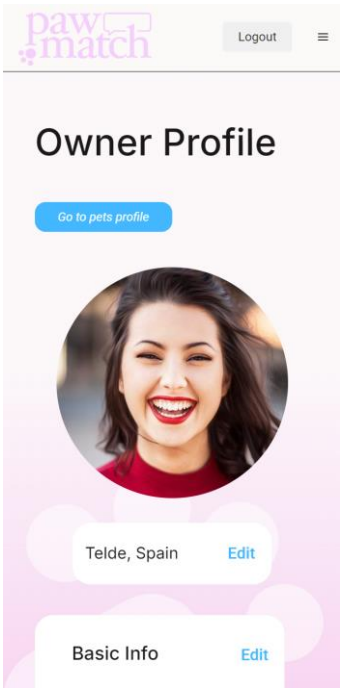

Ilustración 36: Página de perfil de la persona en móvil

6.1.6 Perfil Mascota

El usuario puede acceder al perfil de su mascota utilizando el botón situado en la esquina superior derecha de la página de su perfil personal. Esta funcionalidad está disponible en cualquier formato de pantalla.

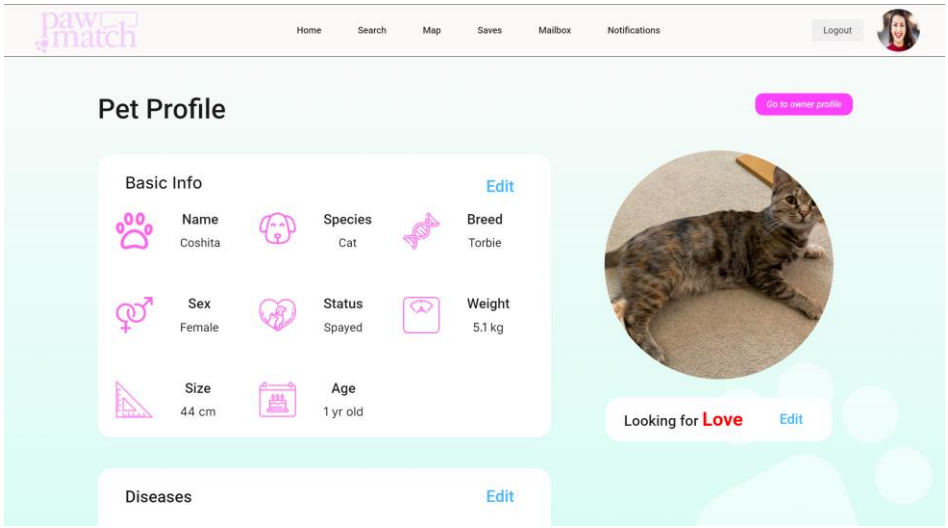

Ilustración 37: Página de perfil de mascota

En esta página, representada en la Ilustración 37 y la Ilustración 38, se presentan todos los datos de la mascota, incluyendo información relevante como nombre, raza, edad y características adicionales que el usuario haya proporcionado al completar el formulario.

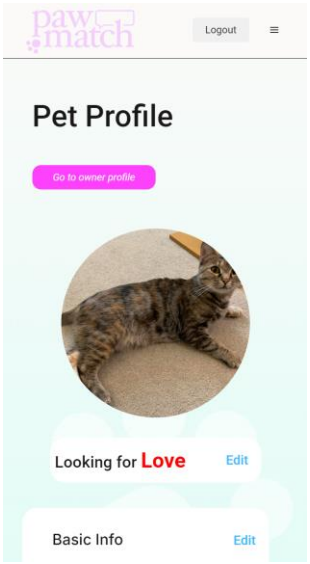

Ilustración 38: Página de perfil de mascota en móvil

6.1.7 Perfiles guardados

Cuando un usuario está interesado en un perfil, puede acceder a la página personal correspondiente. En esta página, se muestran todos los datos del perfil, además de dos iconos visibles debajo de la imagen de perfil de la mascota, como se observa en la Ilustración 39.

- Icono de la izquierda: Permite guardar el perfil en la lista personal de usuarios guardados.
- Icono de la derecha: Ofrece la posibilidad de iniciar un chat directo con la persona asociada al perfil, sin necesidad de haberlo guardado previamente.

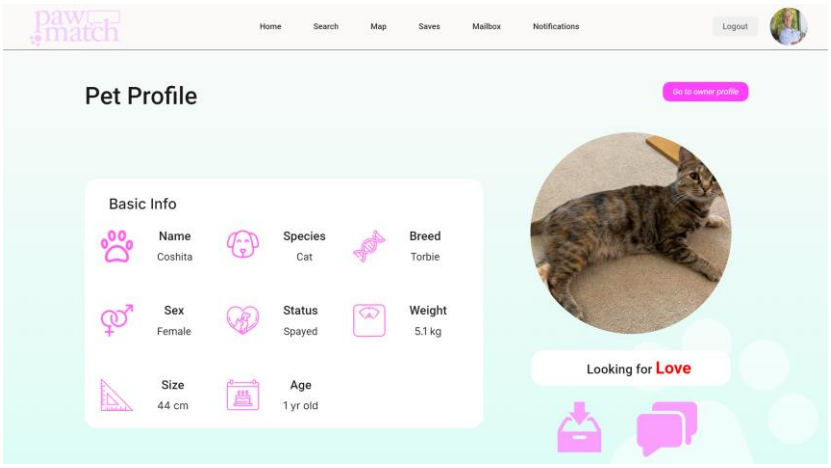

Ilustración 39: Página personal de un perfil

En la pestaña de guardados, el usuario puede gestionar sus perfiles guardados. En el diseño desktop, se presenta una lista en el lado izquierdo donde puede alternar entre ver perfiles de mascotas o personas utilizando los botones “Pets” y “Person”, visibles en la Ilustración 40.

En el lado derecho, se muestra el componente del perfil seleccionado, similar a lo visto en la página principal. Sin embargo, en este caso, aparece un icono adicional debajo del componente para chatear con la persona del perfil. Además, el usuario puede eliminar cualquier perfil guardado directamente desde esta página.

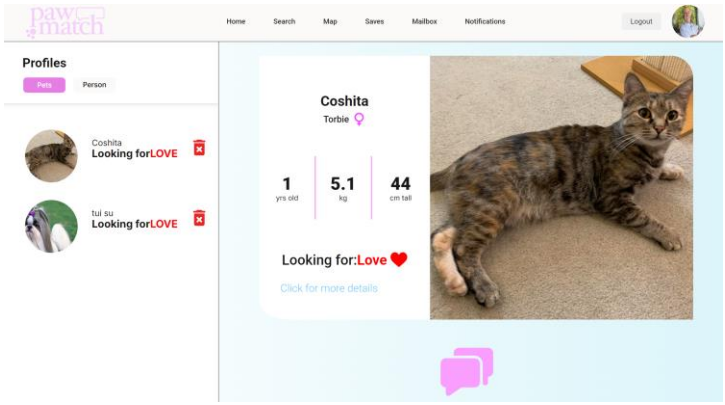

Ilustración 40: Página de perfiles guardados

En el formato móvil, el diseño es más compacto debido a las limitaciones de espacio. Solo se visualizan la lista de perfiles guardados y los iconos para chatear o eliminar perfiles, como se aprecia en la Ilustración 41. Para acceder a la página personal de un perfil guardado, el usuario puede hacer clic directamente en el espacio correspondiente dentro de la lista.

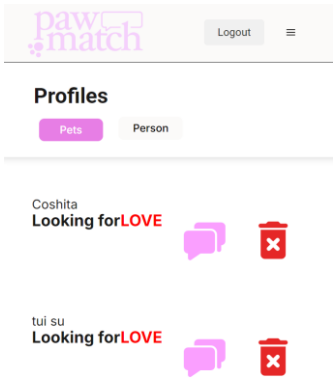

Ilustración 41: Página de perfiles guardados en móvil

6.1.8 Búsqueda por Localización en el mapa

La pestaña del mapa permite al usuario buscar perfiles en función de su localización. El mapa utiliza la ubicación del usuario como punto central, mientras que los marcadores en rojo alrededor representan a otros perfiles, como se muestran en la Ilustración 42. Al pulsar sobre uno de los marcadores, el componente del perfil correspondiente se despliega a la derecha del mapa, de manera similar a otras secciones de la aplicación.

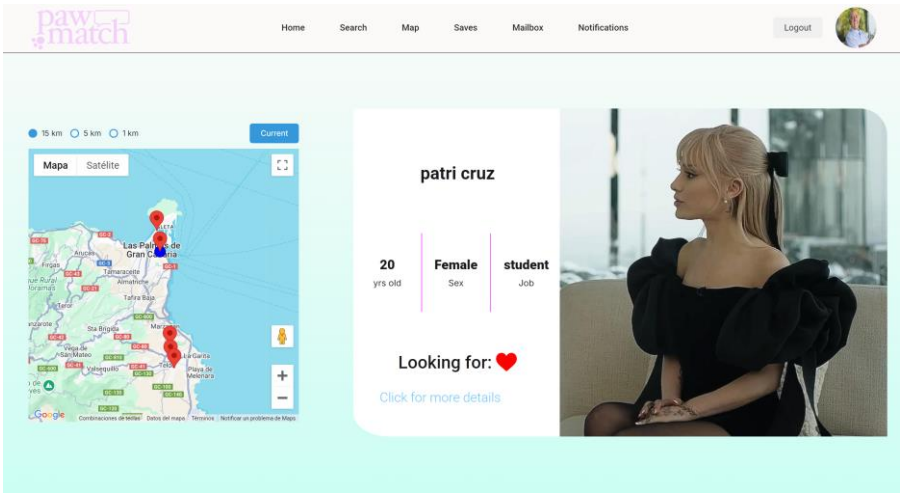

Ilustración 42: Página de búsqueda en mapa

Respecto a las posibilidades que posee el usuario en el mapa, se encuentran:

- Filtrado por distancia: Los marcadores visibles en el mapa pueden limitarse a un radio de 15 km, 5 km o 1 km.
- Botón de ubicación actual: El botón “Current” permite centrar nuevamente el mapa en la ubicación del usuario, en caso de que se haya desplazado por el mapa.
- Interacción con *Google Maps*: El mapa incluye las funcionalidades estándar de *Google Maps*, como zoom, cambio a vista Satélite y la opción de usar el muñeco para explorar visualmente una ubicación.

Además, el mapa puede ampliarse para facilitar la navegación, como se observa en la Ilustración 43.

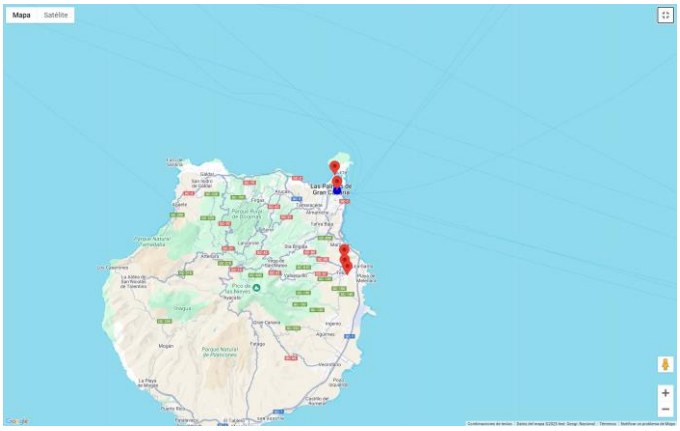

Ilustración 43: Mapa ampliado

En formato móvil, el diseño es más compacto, como se aprecia en la Ilustración 44. En este caso, el perfil seleccionado se muestra debajo del mapa, lo que requiere realizar un desplazamiento para visualizarlo.

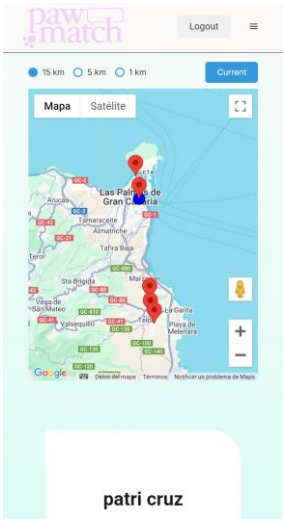

Ilustración 44: Página de búsqueda en mapa en móvil

6.1.9 Mensajería

El usuario puede acceder a la pestaña de mensajería, que se muestra en la Ilustración 45. En esta sección, a la izquierda aparece una lista de los chats activos del usuario. Además, tiene la opción de filtrar los chats para mostrar solo aquellos no leídos.

El usuario puede realizar las siguientes acciones en esta pestaña:

- Visualización del chat: Al hacer click en un chat de la lista, el chat se despliega en la zona derecha de la pantalla.
- Acceso al perfil: En la parte superior del chat, se muestra un perfil al que el usuario puede acceder haciendo click en él, lo que lo lleva a la página personal del perfil.
- Botón de retroceso: Este botón “Back” permite cerrar el componente de chat y regresar a la lista de chats.
- Chatear: Lógicamente, el usuario podrá chatear como en cualquier aplicación de mensajería.

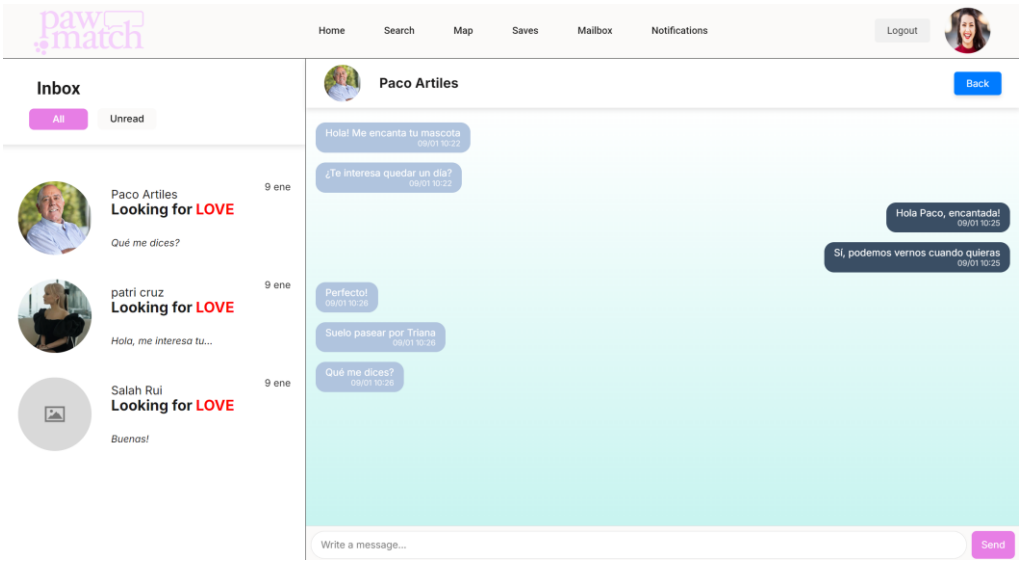

Ilustración 45: Página de mensajería

Dado que la pantalla en dispositivos móviles es más pequeña, solo se puede visualizar la lista de chats o el chat seleccionado, pero no ambos al mismo tiempo. Este diseño es común en aplicaciones de mensajería como *WhatsApp* o *Telegram*, como se muestra en la Ilustración 46 y la Ilustración 47.

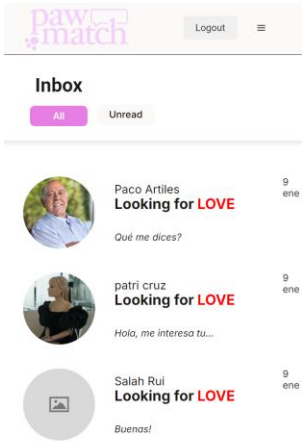

Ilustración 46: Lista de chats en móvil

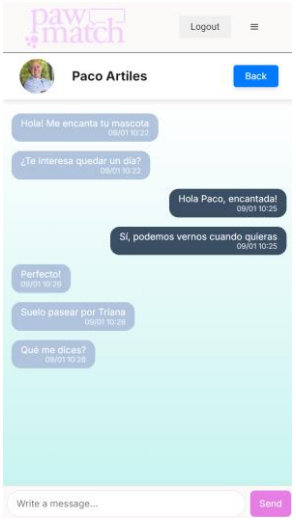

Ilustración 47: Chat en móvil

6.1.10 Notificaciones

Cuando alguien envía un nuevo mensaje al usuario, este recibirá una notificación en la aplicación, como se muestra en la Ilustración 48. La notificación se muestra de manera visible en la interfaz para alertar al usuario de la nueva actividad en su cuenta.

Si el usuario hace click en la notificación, esta desaparece y será redirigido automáticamente a la pestaña de **Mensajería**, donde podrá ver el mensaje recibido y empezar la conversación de inmediato.

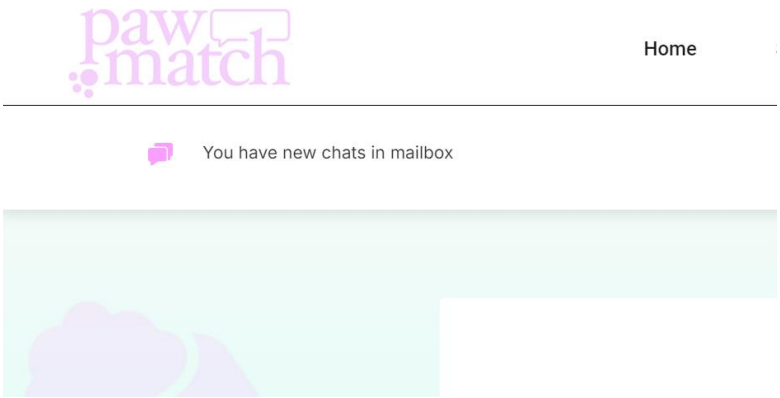

Ilustración 48: Notificación de nuevo chat

7 Guía de desarrollador

Instalación

Para ejecutar el proyecto localmente, se deben seguir los siguientes pasos:

Clonar el repositorio: git clone https://github.com/danicl01/pawMatch.git
2. Navegar a la carpeta del proyecto → cd pawmatch
3. Instalar las dependencias → npm install
4. Iniciar el servidor de desarrollo → ng serve
5. Abrir el navegador y acceder a la aplicación en http://localhost:4200 .

Configuración de Firebase

Para conectar la aplicación a Firebase, se deben seguir estos pasos:

1. Crear un proyecto en Firebase Console.
2. Agregar la configuración e Firebase al proyecto en el fichero enviroments.ts, donde se almacenan las variables de entorno, como la que se muestra en la siguiente Ilustración.

```
export const environment = {
  production: false,
  firebaseConfig: {
    apiKey: "TU_API_KEY",
    authDomain: "TU_AUTH_DOMAIN",
    projectId: "TU_PROJECT_ID",
    storageBucket: "TU_STORAGE_BUCKET",
    messagingSenderId: "TU_MESSAGING_SENDER_
    appId: "TU_APP_ID",
    measurementId: "TU_MEASUREMENT_ID"
  }
};
```

3. En Firebase Console, habilitar Firestore Database y la autenticación, que son los servicios que se utilizan en el proyecto. Se recomienda también habilitar Firebase Storage por posibles mejoras a futuro.

8 Conclusiones

8.1 Introducción

Este proyecto ha consistido en el desarrollo de una aplicación web y móvil que facilita la conexión entre dueños de mascotas y otros usuarios con intereses similares.

A lo largo del documento, se han tratado aspectos claves como el estudio del mercado, una descripción detallada de las tecnologías y herramientas utilizadas, y un listado completo de los requisitos para el análisis.

Posteriormente, se presentaron los diseños de los mockups y la estructura de la base de datos. Además, se detalló el proceso de desarrollo del proyecto, explicando desde el funcionamiento de la base de datos y sus colecciones, hasta la estructura de los componentes, sus funcionalidades y los servicios implementados.

Finalmente, se abordó el apartado de los tests y se ofreció una guía detallada tanto para el usuario y como para cualquier posible desarrollador.

8.2 Conclusiones

Este proyecto ha sido un reto personal significativo, ya que nunca en la carrera había desarrollado un proyecto desde cero, siendo lo máximo alcanzado la creación de prototipos simples y en grupos de mínimo tres personas. Ha sido tanto interesante como desafiante organizar cada etapa, desde el planteamiento de los requisitos y la creación de los primeros mockups hasta la selección de la metodología y las tecnologías más adecuadas para el desarrollo de la aplicación.

Una de las lecciones más importantes que he aprendido es no ser tan ambicioso al intentar cumplir todos los requisitos de inmediato o crear una aplicación demasiado grande. Es más valioso centrarse en la calidad que en la cantidad, y poner atención en los detalles siempre será más efectivo que apresurarse a terminas las tareas.

Uno de los mayores desafíos fue la deuda técnica acumulada durante el desarrollo. La falta de experiencia en la implementación de buenas prácticas de organización y gestión del tiempo tuvo un impacto negativo, pero fue un error que me ha permitido aprender y mejorar en este aspecto para futuros proyectos.

Además, siento que aún hay áreas que puedo seguir mejorando en cuanto a la programación, como la implementación de principios de *clean code* y la realización de tests más formales, en lugar de depender solo de **console.log.**

Al hacer una comparativa con la versión inicial de la aplicación, puedo afirmar que se ha logrado mantener un diseño coherente y se han implementado muchas de las funcionalidades previstas. Durante el proceso, han surgido nuevas ideas, como permitir que el usuario tenga múltiples perfiles para sus mascotas, añadir más filtros de búsqueda, crear grupos de chat y establecer un sistema de valoración de usuarios.

Algunas funcionalidades, como la edición y eliminación de cuentas de usuarios, la gestión del chat (eliminación y bloqueo de perfiles), así como la implementación de una lógica de búsqueda más avanzada según las necesidades del usuario (amor, cuidado o adopción), no pudieron ser implementadas debido a la falta de tiempo. Sin embargo, estas ideas son perfectamente viables para ser desarrollada sen futuras versiones de la aplicación.

8.3 Líneas futuras

A continuación, mencionan algunas de las principales áreas que podrían explorarse en el futuro:

- **Gestión de perfiles**: Permitir que los usuarios tengan múltiples perfiles para sus mascotas y añadir opciones de personalización avanzada.
- **Optimización de la búsqueda**: Introducir filtros adicionales y mejorar la lógica de búsqueda separando por categorías como amor, cuidado y adopción.
- **Mejoras en la mensajería y notificaciones**: Implementar un sistema de mensajería más completo con multimedia y notificaciones personalizadas.
- **Sistema de valoración**: Añadir un sistema de calificación para usuarios y mascotas, mejorando la confianza dentro de la comunidad.
- **Optimización del código**: Refinar la estructura del código siguiendo mejores prácticas de *clean code* y la realización de tests unitarios.
- **Seguridad**: mejorar la seguridad con cifrado y verificación con en dos pasos.

9 Bibliografía

[1] Angular / *Typescript Promises*. (s.f.). Obtenido de https://stackoverflow.com/questions/50513496/angular-typescript-promises

[Último acceso: 12/01/2025]

[2] *Calculate distance between two latitude-longitude points? (Haversine formula)*. (s.f.). Obtenido de https://stackoverflow.com/questions/27928/calculate-distance-between-two-latitude-longitude-points-haversine-formula

[Último acceso: 13/01/2025]

[3] Dayley, B. &. (2018). *Angular Development with TypeScript (2nd Edition)*. *Addison-Wesley*.

[4] *Getting a user profile after login with Angular and Firebase*. (s.f.). Obtenido de https://stackoverflow.com/questions/69031297/getting-a-user-profile-after-login-with-angular-and-firebase

[Último acceso: 14/01/2025]

[5] *How to load Google Maps in service in Angular 2*. (s.f.). Obtenido de https://stackoverflow.com/questions/65799280/how-to-load-google-maps-in-service-in-angular-2

[Último acceso: 13/01/2025]

[6] *ngx-sooner*. (s.f.). Obtenido de https://tutkli.github.io/ngx-sonner/

[Último acceso: 12/01/2025]

[7] *Observe firebase database on changes typescript*. (s.f.). Obtenido de https://stackoverflow.com/questions/49103969/observe-firebase-database-on-changes-typescript

[Último acceso: 14/01/2025]

[8] *Razas de gatos*. (s.f.). Obtenido de https://es.wikipedia.org/wiki/Anexo:Razas_de_gatos

[Último acceso: 13/01/2025]

[9] *Razas de hámster*. (s.f.). Obtenido de https://wikihamies.fandom.com/es/wiki/Categoría:Razas_de_hámster

[Último acceso: 13/01/2025]

[10] *Razas de perros*. (s.f.). Obtenido de https://es.wikipedia.org/wiki/Anexo:Razas_de_perros

[Último acceso: 13/01/2025]

[11] *Responsive design*. (s.f.). Obtenido de https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/CSS_layout/Responsive_Design

[Último acceso: 12/01/2025]

[12] *What is Angular?* (s.f.). Obtenido de https://angular.dev/overview

[Último acceso: 12/01/2025]

[13] *What Is MVVM (Model-View-ViewModel)?* (s.f.). Obtenido de https://builtin.com/software-engineering-perspectives/mvvm-architecture

[Último acceso: 12/01/2025]

[14] PawMatch. Obtenido de https://github.com/danicl01/pawMatch

[Último acceso: 14/01/2025]