UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

GRADO INGENIERÍA INFORMÁTICA

ADOPTA UNA PATITA

ADOPT A LITTLE PAW

Realizado por
FRANCISCO VELASCO ROMERO

Tutorizado por
JOSÉ FRANCISCO CHICANO GARCÍA

Departamento
LENGUAJES Y CIENCIAS DE LA COMPUTACIÓN

MÁLAGA, AGOSTO 2024

UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

GRADUADO EN INGENIERÍA INFORMÁTICA

ADOPTA UNA PATITA

ADOPT A LITTLE PAW

Realizado por

Francisco Velasco Romero

Tutorizado por

José Francisco Chicano García

Departamento

Lenguajes y ciencias de la computación

UNIVERSIDAD DE MÁLAGA

MÁLAGA, AGOSTO DE 2024

Fecha defensa: septiembre de 2024

Resumen

Adopta Una Patita es una aplicación web diseñada para facilitar la adopción de animales tanto por parte de particulares (por ejemplo, personas que encuentran animales abandonados) como de protectoras. A través de esta plataforma, los usuarios pueden crear perfiles detallados de los animales disponibles para adopción, donde pueden subir fotos, incluir información sobre la convivencia con el animal, historial clínico, datos del animal, ubicación, entre otros.

La plataforma ofrece múltiples formas de contacto, como chats en tiempo real y comunicación externa por correo electrónico, para facilitar la interacción entre adoptantes y quienes ofrecen a los animales en adopción.

Además, Adopta Una Patita se enfoca en garantizar el bienestar de los animales adoptados mediante un sistema de seguimiento post adopción. Este seguimiento, que es una práctica común en muchas protectoras, se extiende también a los particulares que utilizan la plataforma. Para estos usuarios individuales, se asignará un tutor, un empleado de la página, que se encargará de monitorear el bienestar del animal.

Además, para facilitar la navegación a través de la web se ofrecerá un sistema de filtros basado en las especies elegidas en la página principal, además de perfiles de usuarios y listado de animales en adopción y adoptados.

Palabras clave:

Contrato adopción, seguimiento adopción, bienestar animal, adopción animal, aplicación web

Abstract

Adopta Una Patita is a web application designed to facilitate the adoption of animals by both individuals (for example, people who find abandoned animals) and animal shelters. Through this platform, users can create detailed profiles of animals available for adoption, where they can upload photos, include information about living with the animal, medical history, animal data, location, among others.

The platform offers multiple forms of contact, such as real-time chats and external email communication, to facilitate interaction between adopters and those offering animals for adoption.

In addition, Adopta Una Patita focuses on ensuring the welfare of adopted animals through a post-adoption follow-up system. This follow-up, which is a common practice in many shelters, is also extended to individuals who use the platform. For these individual users, a tutor, an employee of the site, will be assigned to monitor the animal's welfare.

In addition, to facilitate navigation through the website, a filter system based on the chosen species will be offered on the home page, as well as user profiles and a list of adopted and adopted animals.

Keywords:

Adoption contract, adoption follow-up, animal welfare, animal adoption, web application

Índice

1

Introducción

1.1 Motivación

Adopta Una Patita surge como una solución necesaria ante la evidente falta de plataformas eficientes y seguras en el sector de la adopción animal. Actualmente, en España las opciones disponibles son limitadas, consistiendo principalmente en aplicaciones web desactualizadas, de difícil navegación y poco seguras. La alternativa quizás más empleada son las redes sociales que, aunque cumplen su propósito básico, no están diseñadas específicamente para facilitar el proceso de adopción. Además, en los últimos 8 años, las adopciones animales en España se han disparado, destacando sobre todo el número de perros que ha aumentado casi un 200%, lo que significa que actualmente hay más perros que niños menores de 14 años. Este aumento subraya la necesidad de una solución efectiva para gestionar la adopción de mascotas.

1.2 Objetivos

Adopta Una Patita busca llenar esta falta de aplicaciones web de adopción animal como una plataforma moderna, segura y centrada en mejorar la experiencia de adopción, con una interfaz amigable para todo tipo de público, que simplifica y optimiza este proceso.

Para ello hemos marcado 12 objetivos a cumplir:

1. **Interfaz intuitiva y amigable**

   Nuestra aplicación se ha diseñado con una interfaz intuitiva que facilita el acceso rápido a la información clave de cada animal. Los usuarios pueden navegar fácilmente a través de menús claros y aplicar filtros avanzados para encontrar el animal adecuado. La información se presenta de manera organizada y accesible, con botones de acción visibles que permiten completar tareas de manera eficiente, mejorando así la experiencia del usuario.

2. **Autenticación segura y gestión de usuarios**

   El objetivo es implementar un sistema de autenticación segura y gestión de usuarios utilizando JWT (JSON Web Token) en combinación con Spring Boot. Este sistema permitirá el registro de nuevos usuarios, el inicio de sesión con credenciales seguras y la protección de rutas específicas dentro de la aplicación. JWT se utilizará para generar y validar tokens de autenticación, proporcionando una forma robusta de verificar la

identidad de los usuarios sin necesidad de almacenar sesiones en el servidor. Spring Boot facilitará la configuración y la integración de este mecanismo de seguridad, asegurando que los datos del usuario estén protegidos y que solo los usuarios autenticados puedan acceder a recursos sensibles dentro de la aplicación.

3. **Elaboración de perfiles para usuarios y asociaciones**

   La aplicación permitirá la creación de perfiles tanto para usuarios individuales como para asociaciones, proporcionando una forma sencilla y organizada de gestionar la información relevante. Los perfiles incluirán datos personales como DNI, nombre y apellidos, así como información de contacto como correo electrónico y número de teléfono. Además, se integrará un sistema de localización en el mapa que permitirá visualizar la ubicación de los usuarios o asociaciones, facilitando el acceso a servicios o la gestión de adopciones en áreas específicas. Esta funcionalidad centralizará toda la información importante en un solo lugar, mejorando la eficiencia y el manejo de los datos.

4. **Elaboración de perfiles para animales**

   La aplicación permitirá la creación de perfiles detallados para cada animal, que incluirán toda la información necesaria para facilitar su adopción. Estos perfiles contendrán fotos y/o vídeos del animal, una descripción de su carácter, su sexo, tamaño, peso, raza, y edad. Además, se incluirá su historia, historial médico, y el estado actual del proceso de adopción. También se ofrecerán botones que permitirán acceder al perfil de su tutor actual y una opción de comunicación a través de un chat, para que los adoptantes potenciales puedan hacer preguntas o recibir actualizaciones directamente.

5. **Sistema de comunicación entre potenciales adoptantes y tutores**

   El sistema de comunicación entre potenciales adoptantes y tutores permitirá una interacción directa que facilitará el proceso de adopción. A través de un chat a tiempo real, se podrá realizar una entrevista manual para evaluar la idoneidad del adoptante o, alternativamente, se podrá utilizar un cuestionario automatizado que asegure que el adoptante cuenta con los conocimientos y capacidades básicas necesarias para cuidar del animal de manera adecuada. Esto garantizará que cada animal sea adoptado por personas que estén preparadas para asumir la responsabilidad de su bienestar.

6. **Sistema de búsqueda basado en filtros**

El sistema de búsqueda basado en filtros permitirá a los usuarios encontrar animales que se ajusten a criterios específicos, como raza, tamaño, edad, entre otros. Este sistema ofrecerá una interfaz intuitiva que permitirá a los usuarios aplicar múltiples filtros para refinar los resultados y encontrar rápidamente el animal que mejor se adapte a sus necesidades y preferencias. La búsqueda eficaz facilitará una adopción más precisa y satisfactoria, mejorando la experiencia general del usuario y aumentando las probabilidades de encontrar un hogar adecuado para cada animal.

7. **Sistema de seguimiento post adopción**

   El sistema de seguimiento post adopción garantiza el bienestar del animal después de la adopción mediante un monitoreo continuo.

   Inicialmente, se realizará un seguimiento diario, que se ajustará a visitas semanales después de la primera semana, luego será mensual, y concluirá al cabo de un año. Las protectoras serán responsables de este seguimiento, mientras que, en el caso de adopciones por individuos, un tutor autorizado por la aplicación se encargará de supervisar el bienestar del animal, asegurando una transición exitosa y el mantenimiento de un entorno adecuado.

8. **Difusión de un contrato de adopción**

   La difusión de un contrato de adopción obligatorio establece claramente las obligaciones y derechos tanto para la entidad responsable del animal como para el adoptante. Este contrato debe ser firmado antes de iniciar el proceso de adopción y detalla todos los aspectos esenciales relacionados con la custodia y el bienestar del animal. Asegura que ambas partes comprendan y acepten sus responsabilidades, creando un acuerdo formal que protege los intereses del animal y garantiza el cumplimiento de las normas y expectativas establecidas para su cuidado y adopción.

9. **Registros de autoridad**

   Permiten llevar un seguimiento detallado de todas las modificaciones realizadas en los perfiles, incluyendo la identificación de quién realizó cada cambio y la fecha y hora en que se efectuó. Este sistema proporciona una capa adicional de transparencia y responsabilidad, asegurando que todos los ajustes en la información sean rastreables y auditables. Facilita la supervisión del historial de cambios y asegura la integridad y de los datos almacenados en la plataforma.

10. **Historial de cambios en la aplicación**

Se implementarán tablas históricas para mantener un registro detallado del historial de cambios en la aplicación. Estas tablas almacenarán datos históricos sobre las modificaciones realizadas en los perfiles de usuarios y animales, así como en otras áreas clave de la aplicación. La información registrada incluirá detalles como el tipo de cambio, el usuario que realizó la modificación y la fecha y hora de la actualización. Este sistema asegura la integridad de los datos a lo largo del tiempo, permitiendo la recuperación de versiones anteriores y facilitando la auditoría y el análisis de los cambios.

11. Exportación de datos a Excel

Función de exportación de datos a Excel, restringida exclusivamente para administradores. Esta funcionalidad permitirá la exportación de información detallada sobre animales y usuarios en formatos de hojas de cálculo, facilitando el análisis y la gestión de los datos. El objetivo es proporcionar a los administradores una herramienta útil para estudiar tendencias, realizar auditorías y tomar decisiones informadas basadas en datos históricos y actuales.

12. Sistema de roles con varios permisos

El sistema incluirá un esquema de roles con distintos niveles de permisos para asegurar una gestión adecuada de la aplicación. Los roles definidos son:

Usuario Normal: Accede a funcionalidades básicas como buscar y visualizar perfiles de animales, enviar solicitudes de adopción, y gestionar su propio perfil.

Tutor: Tiene acceso adicional para gestionar la información de los animales bajo su cuidado, comunicarse con potenciales adoptantes, y recibir seguimiento post adopción.

Administrador: Posee permisos completos para gestionar todos los aspectos del sistema, incluyendo la administración de usuarios, la configuración de roles y permisos, la supervisión del proceso de adopción, y el acceso a herramientas de análisis y exportación de datos.

Este sistema de roles permitirá un control efectivo sobre las acciones y accesos dentro de la aplicación, garantizando que cada tipo de usuario tenga acceso solo a las funciones y datos pertinentes a su rol.

1.3 Metodología de trabajo

Inicialmente, se optó por emplear una metodología ágil basada en Scrum, estructurada en *sprints* de dos semanas de duración. En cada sprint, se establecían una serie de objetivos específicos, que debía ser cumplidos antes de detallar las tareas para el siguiente sprint. Esta metodología permitía una planificación flexible y la posibilidad de ajustar el desarrollo conforme se avanzaba en el proyecto.

Sin embargo, durante las primeras fases del desarrollo, se hizo presente una dificultad significativa en el cumplimiento de los objetivos dentro del tiempo previsto. Esto se debió a la falta de experiencia previa con Angular y el desarrollo *frontend*, lo que dificultaba realizar predicciones precisas sobre el tiempo necesario para completar las tareas asignadas. Como resultado, la metodología basada en *sprints* no resultó ser la más adecuada para este proyecto.

Ante esta situación, se decidió cambiar a una metodología iterativa, que ofrecía mayor flexibilidad y permitía centrarse en objetivos específicos sin la presión de plazos estrictos. Bajo este enfoque, los objetivos del proyecto se organizaron de manera lógica y en cascada, siguiendo el flujo funcional de la aplicación web.

El desarrollo se estructuró de la siguiente forma:

1. Registro de usuarios y gestión de autenticación: Se diseñó e implementó el sistema de registro, inicio de sesión y confirmación de cuentas.
2. Estructura de la página principal: A continuación, se estableció una estructura y patrón de diseño a seguir como la paleta de colores, *header* y *footer*.
3. Gestión de animales: Implementación de las funcionalidades relacionadas con el registro, modificación y gestión de perfiles de los animales en adopción.
4. Filtros y búsqueda: Se añadieron filtros para la página principal, facilitando la búsqueda de animales según diferentes criterios.
5. Perfiles de usuario y comunicación: Desarrollo de perfiles de usuario y las funcionalidades de chat, permitiendo la comunicación entre usuarios.
6. Proceso de adopción: Finalmente, se implementó el flujo completo del proceso de adopción, desde la solicitud, generación de contrato y finalización del proceso.

1.4 Tecnologías a utilizar

Para el desarrollo hemos seleccionado una serie de tecnologías tanto para el *backend* como para el *frontend* que nos van a permitir construir una aplicación web segura, robusta, escalable y fácil de mantener. En las siguientes secciones, se presentarán las tecnologías utilizadas.

1.3.1 Backend

Las tecnologías empleadas en la parte del backend son:

- Java: Lenguaje multiplataforma orientado a objetos desarrollado por Sun Microsystems. Es un lenguaje caracterizado por ser multiplataforma y ser el más popular entre los desarrolladores de aplicaciones web.

- IntelliJ: entorno de desarrollo integrado (IDE) desarrollado por JetBrains que vamos a emplear para trabajar con Java. Destaca sobre todo por sus potentes herramientas de refactorización y depuración entre muchas otras.

- Postman: Herramienta que realiza pruebas de API con una interfaz sencilla y fácil usar.

- Spring Boot: Framework de código abierto en Java que permite la creación de microservicios y proporciona las herramientas necesarias para el desarrollo de aplicaciones web.

- iText: Librería *Open Source* que permite crear y modificar archivos PDF, RTF y HTML.

- Apache POI: Librería *Open Source* para manipular formatos de archivo de Microsoft. En nuestro caso la vamos a usar exclusivamente para la creación y modificación de Excel.

- Lombok: Librería que a través de anotaciones nos reduce el código que codificamos, ahorrándonos tiempo y mejorando la legibilidad de este.

- Thymeleaf: Librería que implementa un motor de plantillas XML/XHML/HTML5. Herramienta que vamos a usar para crear las plantillas que se van a usar para enviar los correos electrónicos correspondientes.

- SonarLint: Extensión de código abierto que identifica y ayuda a solucionar problemas de calidad y seguridad a medida que se va escribiendo código como un corrector ortográfico a tiempo real.

- JSON Web Token (JWT): Estándar basado en JSON para la creación de tokens de acceso que permitan la propagación de identidad y privilegios. Va a permitir un flujo seguro de intercambio de información entre cliente y servidor.

- Maven: Herramienta de gestión de proyectos que se usa para la gestión de dependencias del proyecto. Además, se encarga de compilar el código fuente, empaquetarlo, instalar paquetes, generar documentación basada en el código fuente y gestionar las distintas fases del ciclo de vida del proyecto (procesamiento, generar recursos, tests…).

1.3.2 Frontend

Las tecnologías empleadas en la parte del *frontend* son:

- TypeScript: Lenguaje de programación libre desarrollado por Microsoft. Es un superconjunto de JavaScript que añade un fuerte tipado, modularización, tuplas, decoradores, interfaces y orientación a objetos, haciéndolo un lenguaje muy similar a Java o C#.

- WebStorm: entorno de desarrollo integrado (IDE) desarrollado por JetBrains, especializado en el desarrollo de aplicaciones web y tecnologías relacionadas. Diseñado para facilitar la programación en lenguajes como JS, TS, HTML, CSS y *frameworks* populares como Angular. Destaca sobre todo por sus potentes herramientas de refactorización y depuración entre muchas otras.

- Angular: Framework de desarrollo web de código abierto creado y mantenido por Google. Se utiliza para crear aplicaciones web dinámicas y de una sola página. Su arquitectura está basada en componentes reutilizables.

- Npm: Gestor de paquetes y dependencias para Node.js, encargado de instalar, publicar, y gestionar módulos, facilitando el desarrollo y distribución de aplicaciones y librerías JavaScript.

- NodeJS: Entorno de ejecución de JavaScript en el servidor, que permite construir aplicaciones escalables y de alto rendimiento utilizando JavaScript fuera del navegador.

- NgBootstrap: Librería que nos ofrece componentes de interfaz de usuario reactivos y sencillos de implementar usando únicamente Angular y TypeScript.

- Font Awesome: Framework de iconos vectoriales y estilos css.

- Leaflet: Librería de código abierto empleada para la creación de mapas interactivos de forma sencilla.

- Angular material: Librería que ofrece un conjunto de componentes visuales de angular.

- NgMultiselectDropdown: Librería que nos ofrece un componente de angular para poder crear desplegables de selección múltiple personalizados.

- NgToastr: Librería empleada para crear cuadros informativos para los eventos de éxito, error, información y de advertencia. Empleada en nuestra aplicación para mostrar al usuario errores informativos de la parte de *backend*.

- SockJS: Biblioteca que proporciona una API para la comunicación en tiempo real, bidireccional y de baja latencia similar a WebSockets pero con mayor compatibilidad y fiabilidad. Usado para el sistema de chat en tiempo real.

- SweetAlert2: Biblioteca para la creación y personalización de *popups* que sustituyen a los popups por defecto de JavaScript.

1.3.3 Base de datos

Las tecnologías empleadas en la parte de base de datos son:

- MySQL: Sistema de gestión de bases de datos relacional de código abierto desarrollado por Oracle. Permite almacenar, organizar y recuperar datos de manera eficiente. Es muy popular para todo tipo de aplicaciones.
- DBeaver: Potente software para la gestión de bases de datos de código abierto, libre y con una interfaz muy amigable. Gracias a esto permite la integración con la mayoría de las bases de datos, entre ellas, MySQL.

1.3.4 Tecnologías generales

Además de las tecnologías mencionadas anteriormente, hemos empleado un conjunto adicional de herramientas y tecnologías específicas para el desarrollo y despliegue de aplicaciones web, que son las siguientes:

- Docker: Plataforma de código abierto que automatiza el despliegue de aplicaciones en contenedores que incluyen todo lo necesario para que el software funcione de forma consistente en cualquier entorno.

- Git: Sistema de control de versiones distribuido. Mantiene un historial de cambios en el código entre otras herramientas de gestión del desarrollo del software lo que lo hace indispensable.
- GitHub: Plataforma de desarrollo colaborativo que permite alojar los proyectos en la nube usando el sistema de control de versiones Git.
- GitHub Copilot: Herramienta de programación colaborativa basada en inteligencia artificial desarrollada por GitHub y OpenAI. Está integrado en ambos IDEs usados y sugiere y completa código mejorando en gran medida la productividad y proporcionando buenas prácticas.

1.5 Estructura de la memoria

La memoria se ha estructurado en 6 partes principales: introducción, requisitos, diseño, implementación, pruebas y conclusión.

CAPÍTULO 1.- INTRODUCCIÓN

Se establecen las bases del proyecto, proporcionando el contexto necesario para entender el problema que se trata y los objetivos a alcanzar. Asimismo, se detalla las tecnologías y metodología de trabajo empleadas.

CAPÍTULO 2.- REQUISITOS

Detalla las características esenciales que debe cumplir la aplicación para asegurar su correcto funcionamiento. Los requisitos funcionales se centran en describir las funciones específicas que el sistema debe realizar, como la creación de perfiles de usuarios, la gestión de adopciones y la comunicación entre adoptantes y tutores. Por otro lado, los requisitos no funcionales abordan aspectos como la seguridad, el rendimiento, la escalabilidad y la usabilidad, garantizando que la aplicación sea eficiente, segura y fácil de usar para todos los involucrados.

CAPÍTULO 3.- DISEÑO

La sección de diseño abordará la estructura técnica y arquitectónica de la aplicación, describiendo cómo se organiza y conecta cada componente del sistema. Aquí se detallarán los patrones de diseño utilizados, la arquitectura general del sistema, y las decisiones clave sobre la disposición de la interfaz de usuario, la lógica de negocio y la gestión de datos. También se explicarán las integraciones entre el *frontend*, *backend* y la base de datos, así como las estrategias para garantizar escalabilidad, mantenimiento y facilidad de expansión en el futuro.

CAPÍTULO 4.- IMPLEMENTACIÓN

La sección de implementación detallará cómo se llevó a cabo el desarrollo concreto de las funcionalidades de la aplicación. Aquí se mostrarán ejemplos de código clave, explicando cómo se han materializado las funcionalidades descritas en la fase de diseño. También se cubrirá cómo se utilizaron las tecnologías previamente mencionadas para construir la aplicación, los retos encontrados durante el desarrollo y cómo se resolvieron. Se hará énfasis en cómo se pusieron en práctica las decisiones de diseño para lograr el producto final.

CAPÍTULO 5.- PRUEBAS

La sección de pruebas explicará cómo se ha validado el correcto funcionamiento de la aplicación. Aquí se describirán los distintos tipos de pruebas llevadas a cabo, como pruebas unitarias, de integración y funcionales, detallando las herramientas utilizadas para realizarlas (como JUnit, Mockito, etc.). Se analizarán los resultados obtenidos y se discutirá cómo estas pruebas garantizaron la estabilidad, seguridad y rendimiento de la aplicación. Además, se explicará cómo se abordaron los posibles errores o fallos encontrados durante el proceso de prueba y su resolución.

CAPÍTULO 6.- CONCLUSIONES

Esta última parte de la memoria ofrece una evaluación final del proyecto. Incluyendo un análisis de los objetivos planteados, líneas futuras de la aplicación y una reflexión final sobre la experiencia adquirida y el resultado final en base a los conocimientos iniciales.

2

Requisitos

En esta sección se va a identificar y describir los requisitos funcionales y no funcionales que guiarán el diseño y la implementación de la aplicación web.

2.1 Requisitos funcionales

REQUISITO FUNCIONAL	DESCRIPCIÓN DEL REQUISITO
RF1	Los usuarios deben poder registrarse, activar su cuenta e iniciar sesión introduciendo los datos requeridos
RF2	Los usuarios deben poder ver el perfil del resto de usuarios. Cada uno podrá consultar sus datos privados (tales como el DNI) en su propio perfil
RF3	En la aplicación encontramos dos tipos de usuarios, individual y protectora. El tipo de usuario "protectora" se diferencia del "individual" en que este debe proporcionar datos del animal en adopción que son opcionales para el usuario individual. Las protectoras estarán obligadas a subir la historia clínica del animal además de seleccionar obligatoriamente "SI" en las opciones sobre "tiene las vacunas al día", "esterilizado" y "tiene chip". Cuando una protectora da en adopción un animal, esta se convertirá en el tutor del seguimiento del animal adoptado mientras que a un usuario individual se le asignará un tutor disponible de la aplicación.
RF4	Los usuarios deben poder acceder a una lista donde se muestren todos los animales en adopción por parte de estos
RF5	Los usuarios deben poder acceder a su historial de chats
RF6	Los usuarios deben poder buscar animales disponibles en adopción a través de la página principal. Se proporcionará una serie de filtros para facilitar este proceso
RF7	Los usuarios podrán consultar los detalles de cualquier animal que se encuentre disponible
RF8	Se proporcionará a los usuarios un sistema de mensajería a tiempo real

RF9	Los usuarios podrán adoptar animales a través de un proceso automatizado y guiado con notificaciones del proceso a través de correo electrónico
RF10	En el proceso de adopción se generará automáticamente un contrato de adopción con los datos del animal, adoptante y propietario además de las bases legales y obligaciones
RF11	Se proporcionará un sistema de seguimiento post adopción en el que se asignará un tutor. Si el propietario es una protectora, esta misma será el tutor, en cambio, si es un individual, se escogerá a un tutor disponible de la aplicación
RF12	Implementación de tablas históricas y registros de auditoría
RF13	Los administradores podrán exportar datos de animales y usuarios a Excel
RF14	Los usuarios registrados y que hayan iniciado sesión podrán acceder a un panel de control que se encontrará en la cabecera de la página

Figura 1.- Requisitos funcionales

2.2 Requisitos no funcionales

REQUISITO NO FUNCIONAL	DESCRIPCIÓN DEL REQUISITO
RNF1	Implementación de una interfaz intuitiva y amigable. Con un diseño atractivo para todo tipo de usuario
RNF2	Los datos sensibles como como contraseñas serán cifrados
RNF3	Habrá un sistema de gestiones sesión y comunicaciones seguras. En nuestro caso emplearemos JWT (JSON Web Token)
RNF4	El sistema podrá interactuar con otras plataformas externas a través del estándar RESTful API

Figura 2.- Requisitos no funcionales

3

Diseño

En esta sección se detalla la arquitectura y diseño técnico de la aplicación web, cubriendo aspectos clave de su implementación.

3.1 Base de datos

Es una parte fundamental en el desarrollo de cualquier aplicación, ya que es donde se almacenan y gestionan los datos esenciales para su funcionamiento. En el caso de esta aplicación, la base de datos está diseñada para gestionar la información relacionada con los usuarios, los animales en adopción, los procesos de adopción y las interacciones entre estos elementos.

Para representar la estructura y relaciones entre las distintas entidades de la base de datos, se ha utilizado el modelo Entidad-Relación (ER). Este modelo gráfico facilita la visualización de las entidades que conforman el sistema (como usuarios, animales y adopciones), así como las relaciones entre ellas (por ejemplo, un usuario puede adoptar uno o más animales).

El modelo ER es una herramienta fundamental en la fase de diseño de bases de datos, ya que ayuda a identificar las tablas, los atributos y las claves primarias y foráneas que se utilizarán para organizar los datos de manera eficiente.

Una entidad es un objeto o concepto del mundo real que tiene una representación en la base de datos, como por ejemplo un usuario, un animal o una adopción. Cada entidad tiene un conjunto de propiedades o atributos que la describen, como el nombre o el correo electrónico en el caso de un usuario, o la raza y edad en el caso de un animal.

Las relaciones, por su parte, definen cómo las entidades están conectadas entre sí. Estas relaciones pueden ser de uno a uno, uno a muchos o muchos a muchos, y permiten organizar

los datos de manera coherente y eficiente. Además, en el modelo ER, se definen las claves primarias y foráneas que facilitan la identificación única de los registros y su interrelación.

A continuación, se presentará el diagrama ER de la base de datos, que describe cómo se estructuran las entidades y las relaciones dentro del sistema, asegurando la integridad y coherencia de los datos.

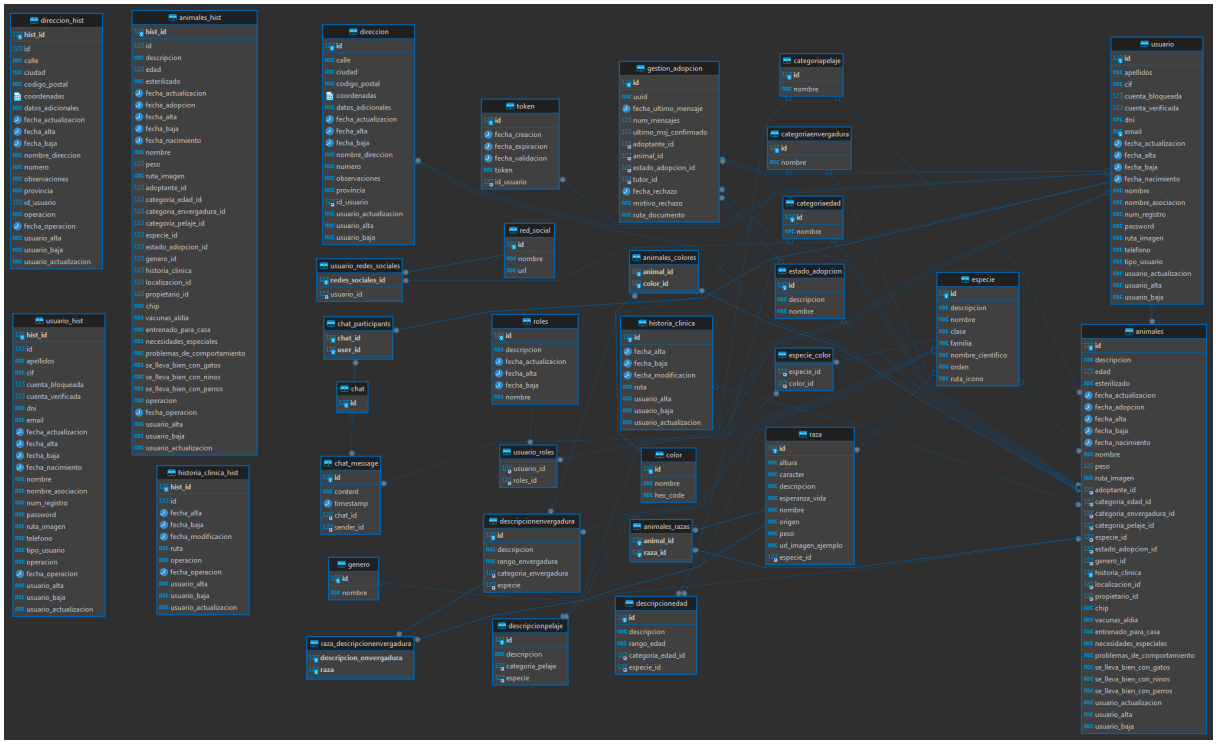


Figura 3.- Diagrama ER

TIPOS DE ENTIDADES

Vamos a hacer una diferenciación entre los tres tipos de entidades empleadas en la aplicación.

- **Entidades maestras**

  Consisten en aquellas que contienen los datos centrales o fundamentales de la aplicación. Describen las entidades principales del sistema. Tienen una estructura compleja y sus valores cambian con frecuencia.

- **Entidades catálogo**

Son aquellas que contienen valores estáticos o casi estáticos que se utilizan para clasificar, categorizar o parametrizar datos de otras tablas. Normalmente su contenido no cambia o lo hace rara vez.

- **Entidades históricas**

Están conformadas por tablas que no poseen ninguna relación directa con el resto. Su función es guardar únicamente los cambios en los registros de las tablas más sensibles cada vez que ocurre la creación, modificación o eliminación de estos. Esta función se lleva a cabo a través de *triggers*.

3.1.1 Entidades y relaciones

Vamos a detallar las relaciones más importantes del modelo:

Categoría - descripción - especie

Las tablas de categoría son unas tablas catálogo que marcan las distintas categorías que hay en cuanto a edad, envergadura y pelaje. La tabla descripción tiene una relación con la especie y la categoría correspondiente. Esto permite tener las mismas categorías para todas las especies, pero con descripciones propias de cada una.

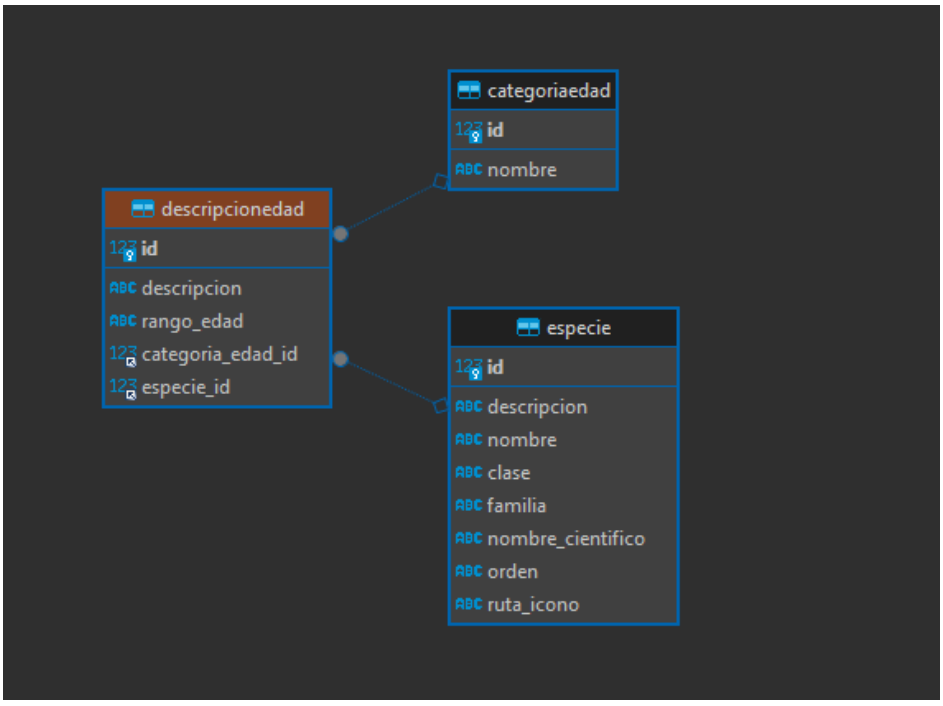


Figura 4.- Categoría - descripción -especie

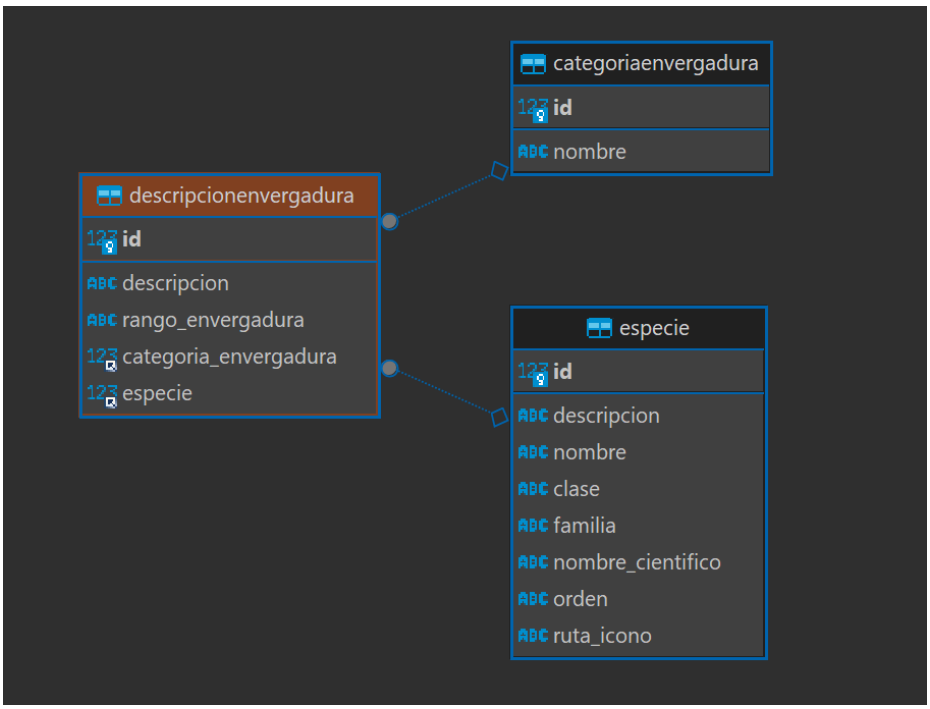


Figura 5.-Categoría - descripción - especie

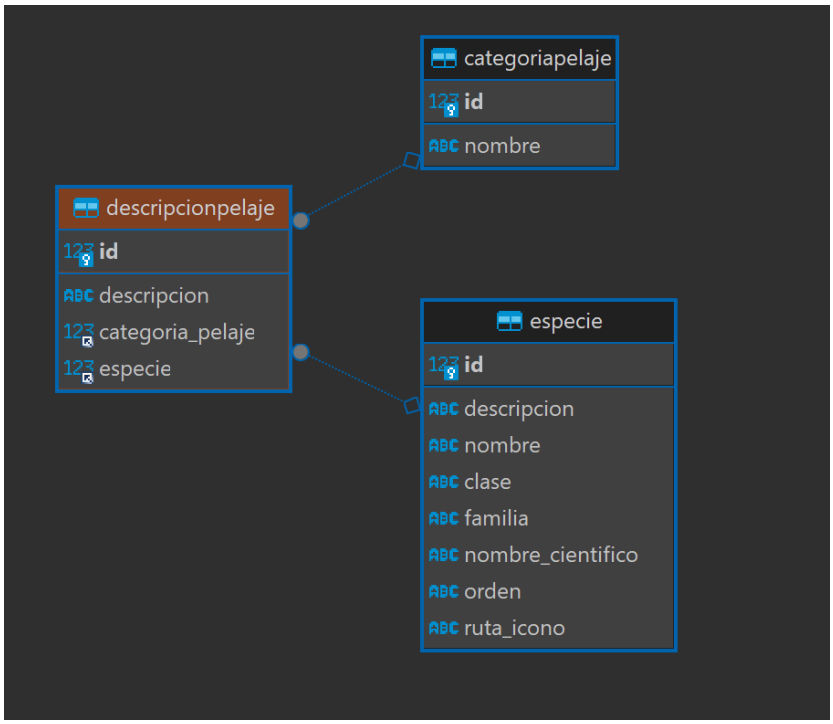


Figura 6.- Categoría - descripción - especie

Usuario

Constituye la parte esencial de la aplicación y participa en casi todas las relaciones. Presenta relaciones de muchos a muchos con roles. Esto se ha realizado con la intención de que un usuario pudiera tener más de un rol a futuro. También contaremos con un enumerado de "Tipo de usuario" cuyos valores pueden ser ("NORMAL", "PROTECTORA", "TUTOR", "ADMIN"). El resto de las relaciones son de uno a muchos, que incluyen direcciones, redes sociales y gestiones de adopción (equivale a estar en un proceso de adopción como adoptante o propietario) ya que se ha decidido que un usuario pueda tener varias direcciones, redes y procesos adoptivos respectivamente.

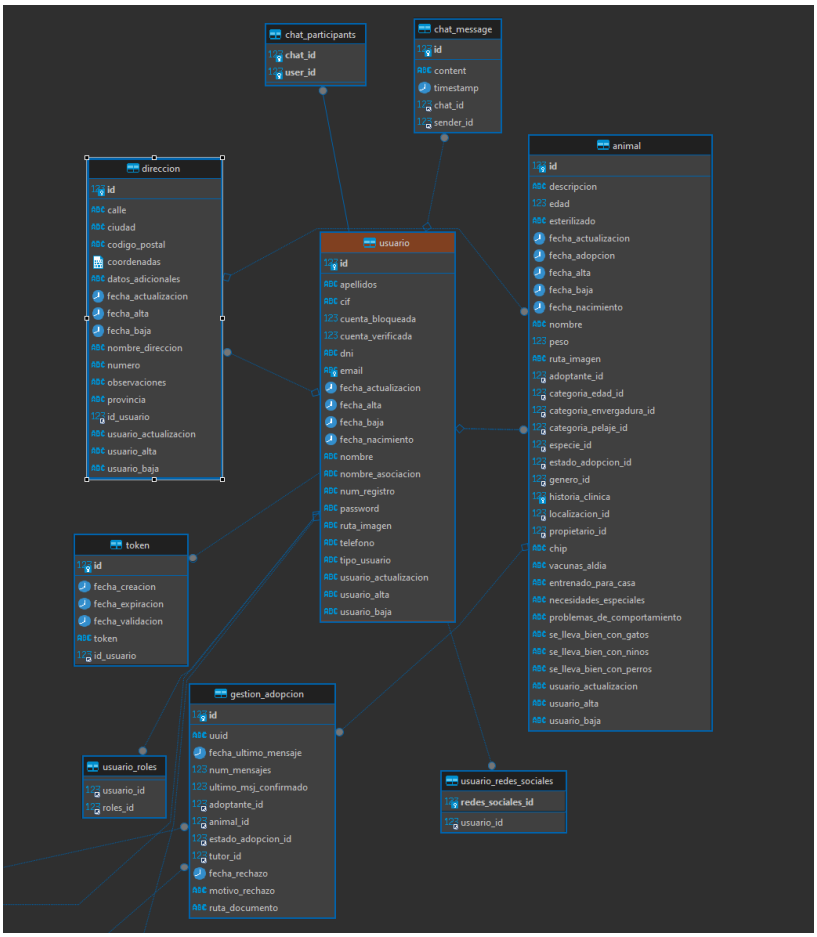


Figura 7.- Usuario

Animal

Es la segunda tabla más importante después de usuario y tiene una gran cantidad de relaciones importantes. Se establece una relación de muchos a uno con dirección ya que la ubicación del animal no tiene por qué coincidir con la del usuario, bien porque sea una

protectora distribuida o bien porque el usuario individual lo tenga a cargo de otra persona. También presentará relaciones de muchos a uno con las distintas categorías (edad, envergadura y pelaje), especie y género. Tiene un estado de adopción que es una tabla catálogo con el estado actual del animal. Presenta dos relaciones de muchos a uno con usuario, esto corresponde con el adoptante y el propietario del animal. Cuenta con una relación uno a uno con historia clínica, esta tabla mencionada se ha creado con el propósito de dar mayor seguridad e integridad a datos sensibles como estos. Finalmente, encontramos relaciones de muchos a muchos con las tablas color y razas, ya que estos pueden presentar más de un color y tener mezclas de razas.

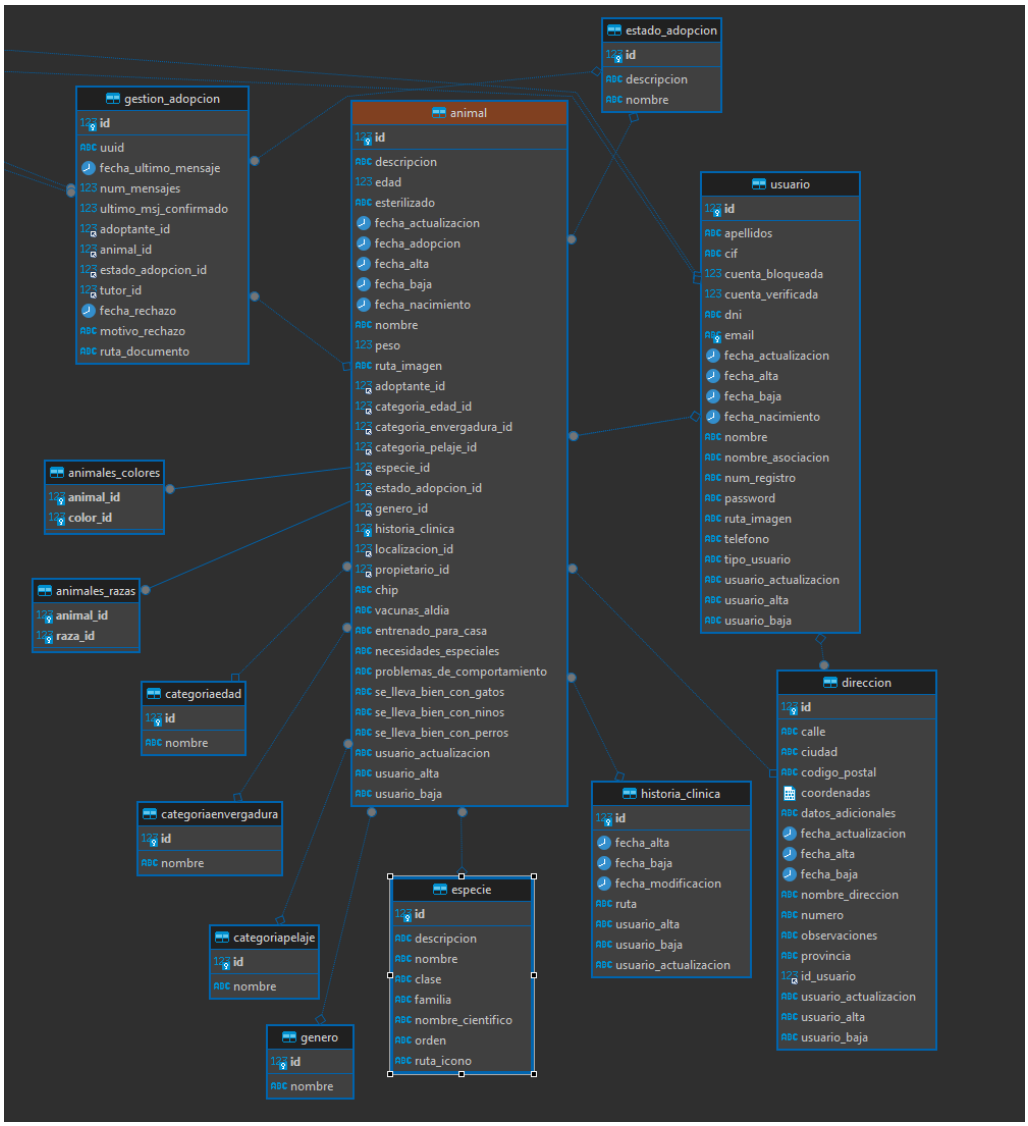


Figura 8.- Animales

Chat

Está conformado por un par de relaciones importantes. Una relación de muchos a muchos con usuarios ya que en un chat podrían llegar a participar varias personas como podrían ser dos usuarios y un tutor. La otra relación consiste en un uno a muchos con mensajes del chat que es una tabla encargada de tener un registro del mensaje enviado por el usuario y la fecha. Estas relaciones mencionadas permiten la existencia de un chat en tiempo real.

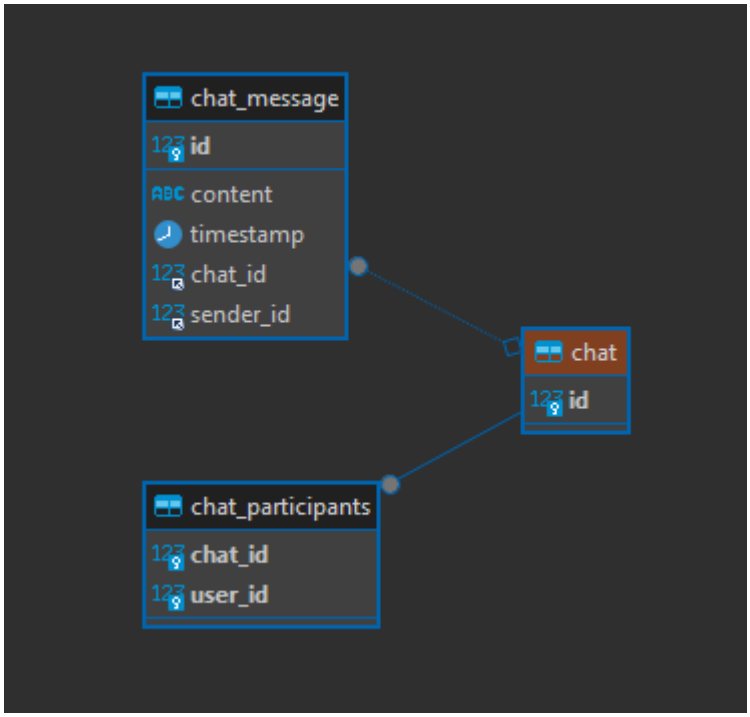


Figura 9.- Chat

Token

Es una entidad que se utiliza para activar la cuenta del usuario una vez se registra. Para ello, Tiene unos campos de fechas de creación, expiración y validación, un token de seis dígitos y el usuario asociado a este token. Consiste en una relación de muchos a uno con la entidad usuario, ya que se pueden generar varios tokens para un mismo usuario debido a que intente usar un token caducado.

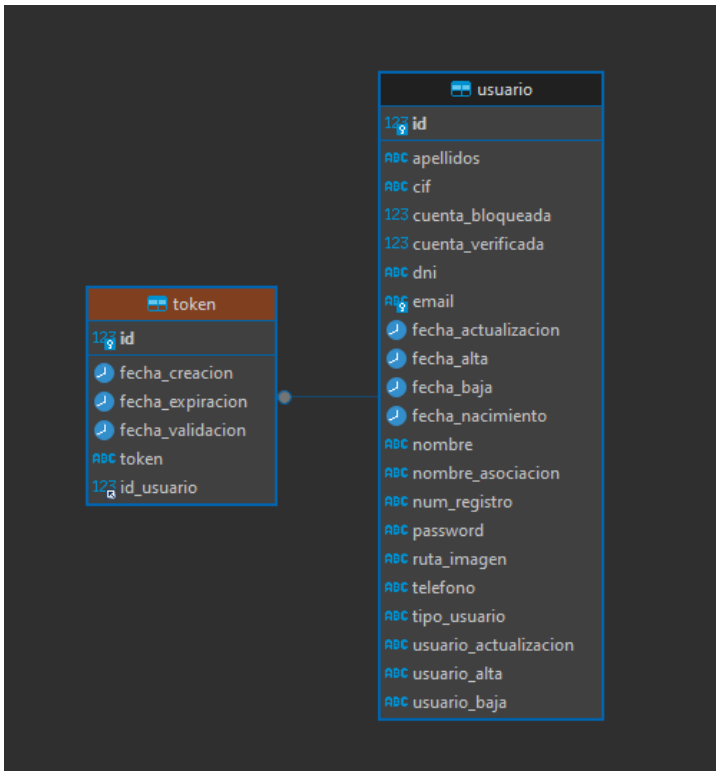


Figura 10.- Token

Dirección

Entidad que, como su nombre indica, está creada con la intención de poder guardar y representar una dirección real. Tiene campos típicos como la calle, ciudad, el código postal, provincia y observaciones. Además, contará con un campo coordenadas para poder ser representada de forma precisa en un mapa.

Con vistas a futuro se ha añadido que las direcciones puedan tener un nombre asociado elegido ya que queremos introducir que el usuario pueda guardar varias direcciones y pueda reutilizarlas y no tener que introducirlas manualmente cada vez (relación de muchos a uno).

En cambio, un animal solo podrá estar en una dirección concreta (relación de uno a muchos).

Figura 11.- Dirección

Raza

Representa una categoría importante a la hora de clasificar los animales. Su campo más importante es el nombre (obviando la clave primaria). El resto de datos simplemente tienen un carácter informativo para futuras funcionalidades de la aplicación. Está conformada por dos relaciones. Muchos a uno con especie, para poder mostrar las razas disponibles según la especie elegida y, muchos a muchos con animales ya que un animal puede tener varias razas.

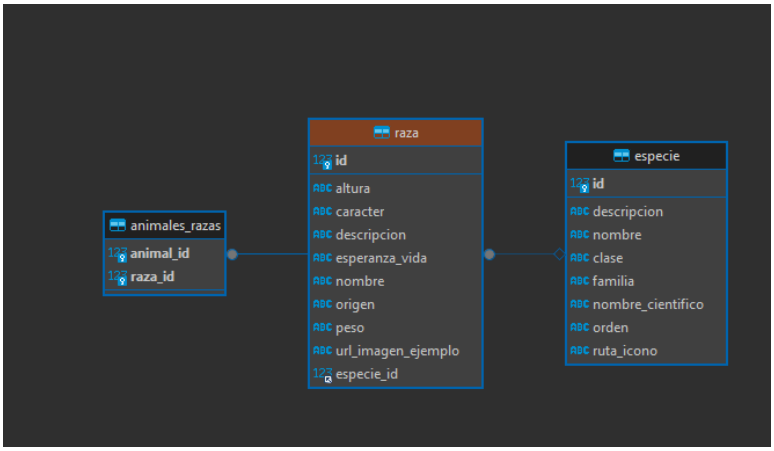


Figura 12.- Raza

Color

Tiene como finalidad representar los colores disponibles en la aplicación. Como característica especial podemos observar que tiene un campo "hex_code", esto se debe a que usaremos estos códigos para poder mostrar al usuario una representación de los colores elegidos en la creación del perfil del animal.

Presenta relaciones de muchos a muchos con animales y especie ya que un animal puede tener varios colores y, esta relación también se da con especie ya que se va a usar para poder filtrar los colores disponibles por especie elegida en los filtros.

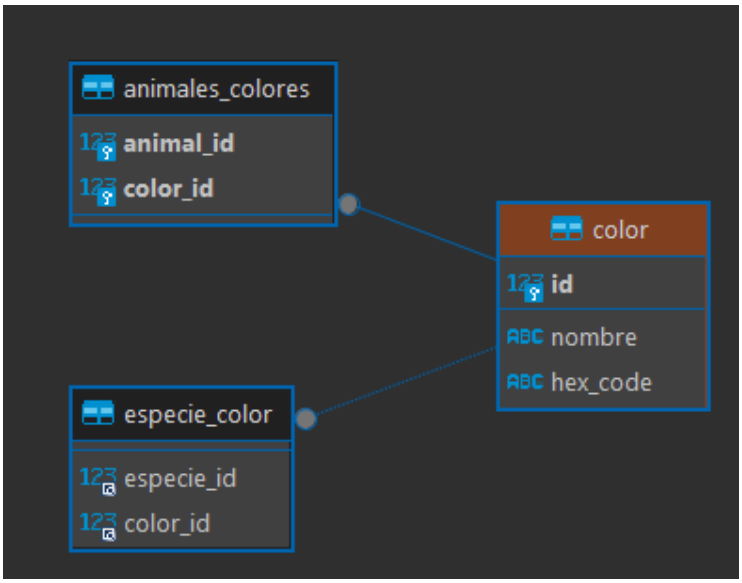


Figura 13.- Color

Roles

Entidad que almacena el catálogo disponible de roles en la aplicación. Estos roles van a determinar el acceso a distintas funcionalidades y parte del sistema, conformando una parte indispensable de la seguridad.

Tiene una relación de muchos a muchos con usuario porque para ciertas funcionalidades a futuro se ha planteado que un usuario pueda tener más de un rol.

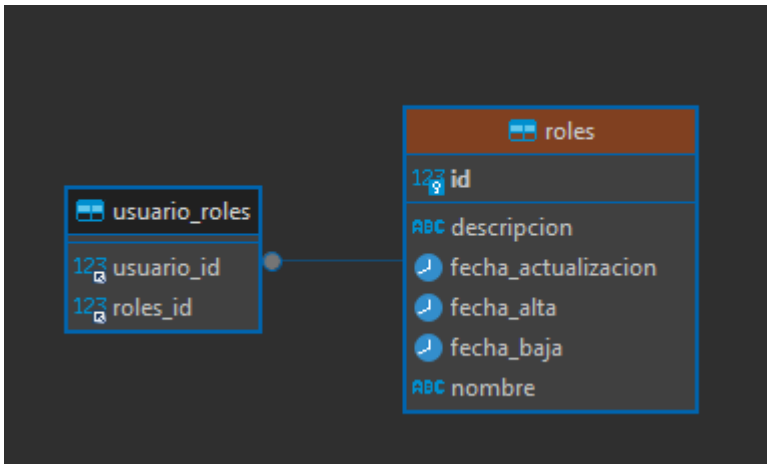


Figura 14.- Roles

Gestión adopción

Entidad indispensable para el proceso de adopción y el seguimiento post adopción. Contiene un identificador único para poder distinguir cada proceso, fecha del último mensaje por parte del adoptante, el número de mensajes para saber en qué periodicidad de mensajes se encuentra el proceso adoptivo (diaria o mensual), el adoptante, animal, estado de la adopción, tutor asignado, ruta del documento (contrato de adopción) y, en caso de rechazar la petición encontramos dos campos, la fecha en la que se rechazó y un motivo.
Encontramos relaciones de muchos a uno como el estado de adopción, el cuál indicará el momento en el que nos encontramos de la adopción (esperando firmas, en espera de recibir el animal…), la relación con usuario que, representa al usuario adoptante y, por último la relación con animal, que sirve para indicar campos referentes a este como su propietario.

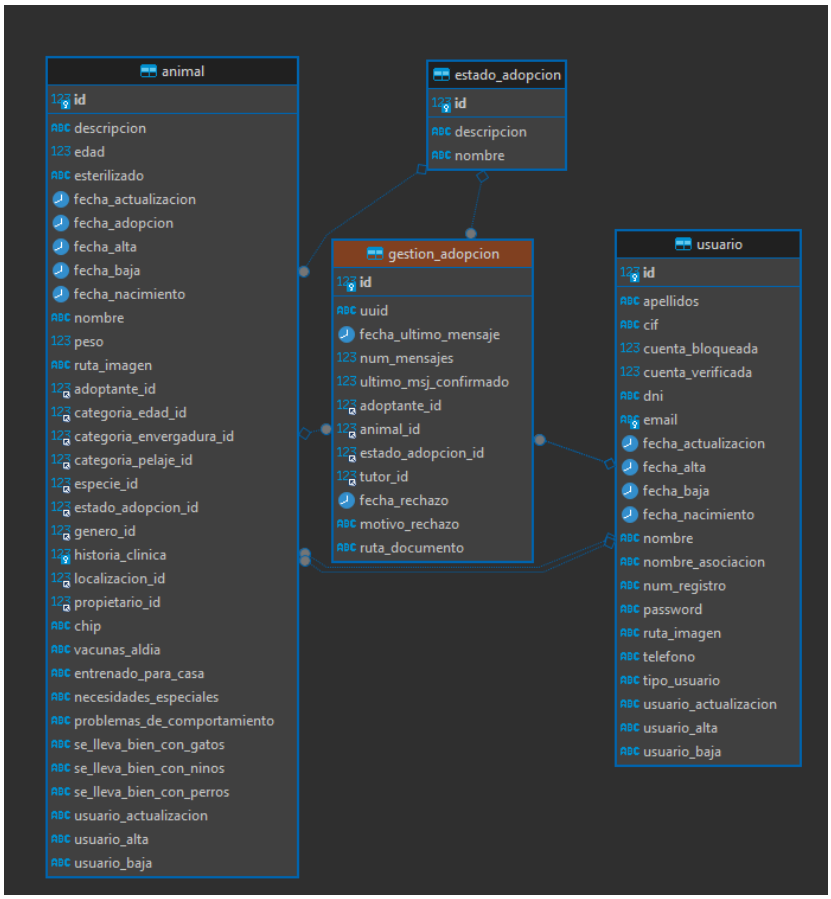


Figura 15.- Gestión adopción

Género

Su finalidad consiste en proporcionar los géneros disponibles a elegir a través del filtro de animales.

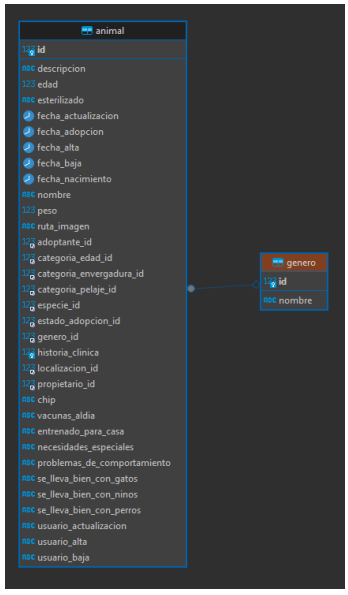


Figura 16.- Género

Red social

Esta entidad está pensada para usarse en el futuro cuando se permita añadir redes sociales a través de la edición del perfil del usuario.

Como dato a destacar posee un campo nombre el cuál es un enumerado en el que se puede elegir entre 5 redes sociales: youtube, instragram, facebook, X (twitter) y tiktok.

Tiene una relación de muchos a muchos con usuarios. Estos podrán elegir hasta 5 redes sociales para mostrar en su perfil.

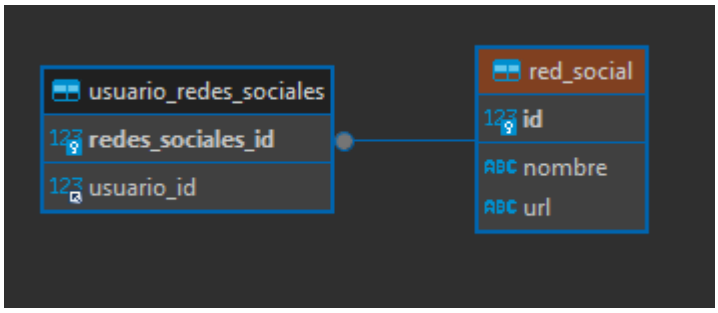


Figura 17.- Red social

Entidades históricas

Son entidades iguales que la original de la que provienen. Su objetivo es guardar el histórico de cambios sobre estas entidades originales que son la base de la aplicación por motivos de seguridad e integridad de datos.

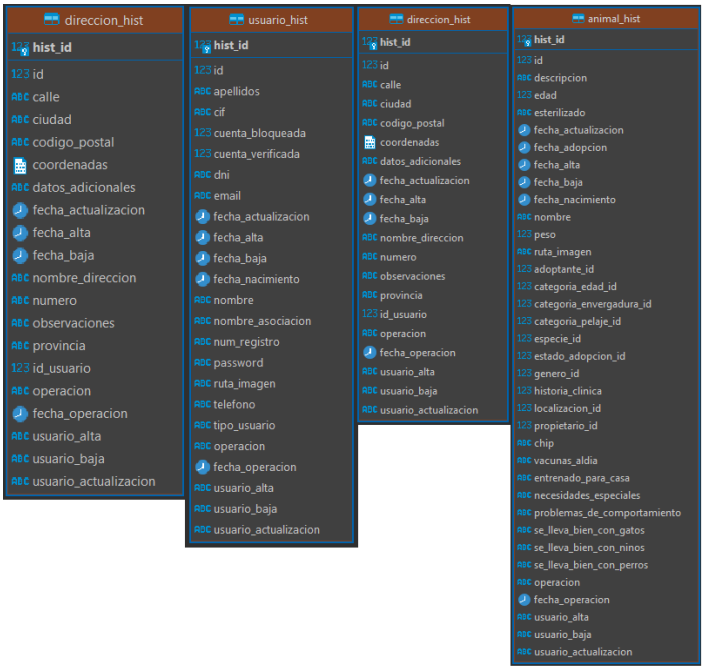


Figura 18.- Entidades históricas

3.1.2 Triggers

Los *triggers* son procedimientos automáticos en una base de datos que se ejecutan en respuesta a eventos específicos, como la inserción, actualización o eliminación de registros en una tabla. Los triggers permiten implementar reglas de negocio, mantener la integridad de los datos o realizar auditorías de manera automática sin intervención directa del usuario.

En nuestra aplicación, hemos implementado *triggers* para las tablas de usuario, dirección, historia clínica y animales. Estas tablas contienen datos críticos, y es esencial llevar un registro histórico de cada cambio. Por ello, cada vez que se crea, modifica o elimina un registro en estas tablas, se genera automáticamente un registro en sus respectivas tablas históricas. Esto asegura la trazabilidad y la integridad de la información más relevante en la aplicación.

A continuación, se muestra un ejemplo con la entidad dirección:

```
CREATE DEFINER=`TFGUSER`@`%` TRIGGER `direccion_after_insert` AFTER INSERT ON `direccion` FOR EACH ROW BEGIN
  INSERT INTO direccion_hist (
    `id`, `calle`, `ciudad`, `codigo_postal`, `coordenadas`,
    `datos_adicionales`, `fecha_actualizacion`, `fecha_alta`,
    `fecha_baja`, `nombre_direccion`, `numero`, `observaciones`,
    `provincia`, `id_usuario`, `operacion`, `usuario_alta`, `usuario_baja`, `usuario_actualizacion`
  )
  VALUES (
    NEW.`id`, NEW.`calle`, NEW.`ciudad`, NEW.`codigo_postal`, NEW.`coordenadas`,
    NEW.`datos_adicionales`, NEW.`fecha_actualizacion`, NEW.`fecha_alta`,
    NEW.`fecha_baja`, NEW.`nombre_direccion`, NEW.`numero`, NEW.`observaciones`,
    NEW.`provincia`, NEW.`id_usuario`, 'INSERT', NEW.`usuario_alta`, NEW.`usuario_baja`, NEW.`usuario_actualizacion`
  );
END
```

Figura 19.- Trigger de creación

```
CREATE DEFINER=`TFGUSER`@`%` TRIGGER `direccion_after_update` AFTER UPDATE ON `direccion` FOR EACH ROW BEGIN
  INSERT INTO direccion_hist (
    `id`, `calle`, `ciudad`, `codigo_postal`, `coordenadas`,
    `datos_adicionales`, `fecha_actualizacion`, `fecha_alta`,
    `fecha_baja`, `nombre_direccion`, `numero`, `observaciones`,
    `provincia`, `id_usuario`, `operacion`, `usuario_alta`, `usuario_baja`, `usuario_actualizacion`
  )
  VALUES (
    NEW.`id`, NEW.`calle`, NEW.`ciudad`, NEW.`codigo_postal`, NEW.`coordenadas`,
    NEW.`datos_adicionales`, NEW.`fecha_actualizacion`, NEW.`fecha_alta`,
    NEW.`fecha_baja`, NEW.`nombre_direccion`, NEW.`numero`, NEW.`observaciones`,
    NEW.`provincia`, NEW.`id_usuario`, 'UPDATE', NEW.`usuario_alta`, NEW.`usuario_baja`, NEW.`usuario_actualizacion`
  );
END
```

Figura 20.- Trigger de modificación

```
CREATE DEFINER=`TFGUSER`@`%` TRIGGER `direccion_after_delete` AFTER DELETE ON `direccion` FOR EACH ROW BEGIN
  INSERT INTO direccion_hist (
    `id`, `calle`, `ciudad`, `codigo_postal`, `coordenadas`,
    `datos_adicionales`, `fecha_actualizacion`, `fecha_alta`,
    `fecha_baja`, `nombre_direccion`, `numero`, `observaciones`,
    `provincia`, `id_usuario`, `operacion`, `usuario_alta`, `usuario_baja`, `usuario_actualizacion`
  )
  VALUES (
    OLD.`id`, OLD.`calle`, OLD.`ciudad`, OLD.`codigo_postal`, OLD.`coordenadas`,
    OLD.`datos_adicionales`, OLD.`fecha_actualizacion`, OLD.`fecha_alta`,
    OLD.`fecha_baja`, OLD.`nombre_direccion`, OLD.`numero`, OLD.`observaciones`,
    OLD.`provincia`, OLD.`id_usuario`, 'DELETE', OLD.`usuario_alta`, OLD.`usuario_baja`, OLD.`usuario_actualizacion`
  );
END
```

Figura 21.- Trigger de borrado

3.2 Backend

El backend es el componente responsable de procesar las solicitudes del usuario, aplicar las reglas de negocio y enviar las respuestas correspondientes. Para llevar a cabo esto hemos decidido elegir Spring Boot, un poderoso framework que facilita la creación de aplicaciones web robustas y escalables.

Se ha implementado una arquitectura MVC junto con una API RESTful en el sistema. Esto implica que cada funcionalidad está accesible a través de endpoints bien definidos, que siguen las convenciones de REST. Estos endpoints actúan como puntos de entrada y salida para las operaciones básicas del sistema, permitiendo realizar acciones como Crear, Leer, Actualizar y Eliminar (CRUD).

3.2.1 Seguridad

En la aplicación se utiliza JWT (JSON Web Token) en combinación con Spring Security para la autenticación y autorización. JWT es un estándar que permite transmitir información entre partes de manera segura, en forma de un token firmado. Este token contiene información del usuario y otros datos necesarios para verificar su identidad [1].

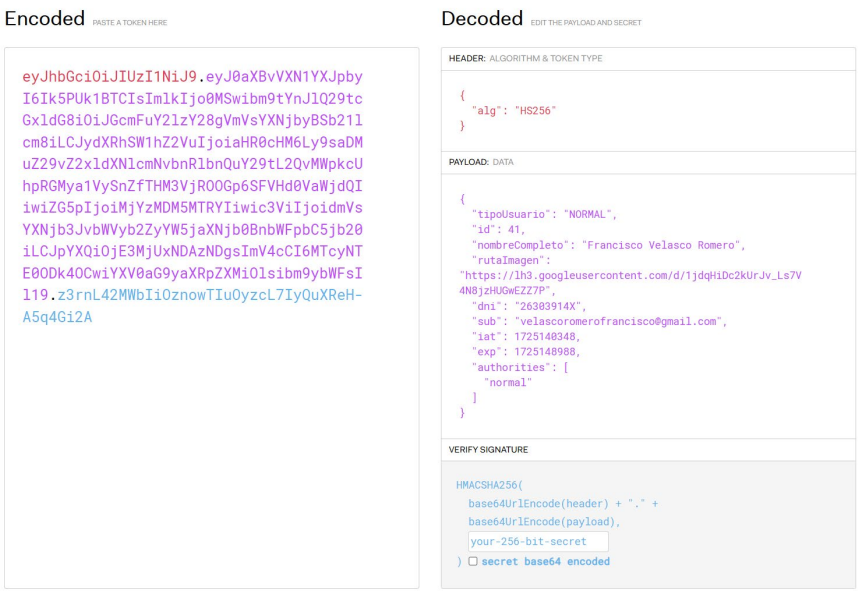


Figura 22.- Estructura JWT

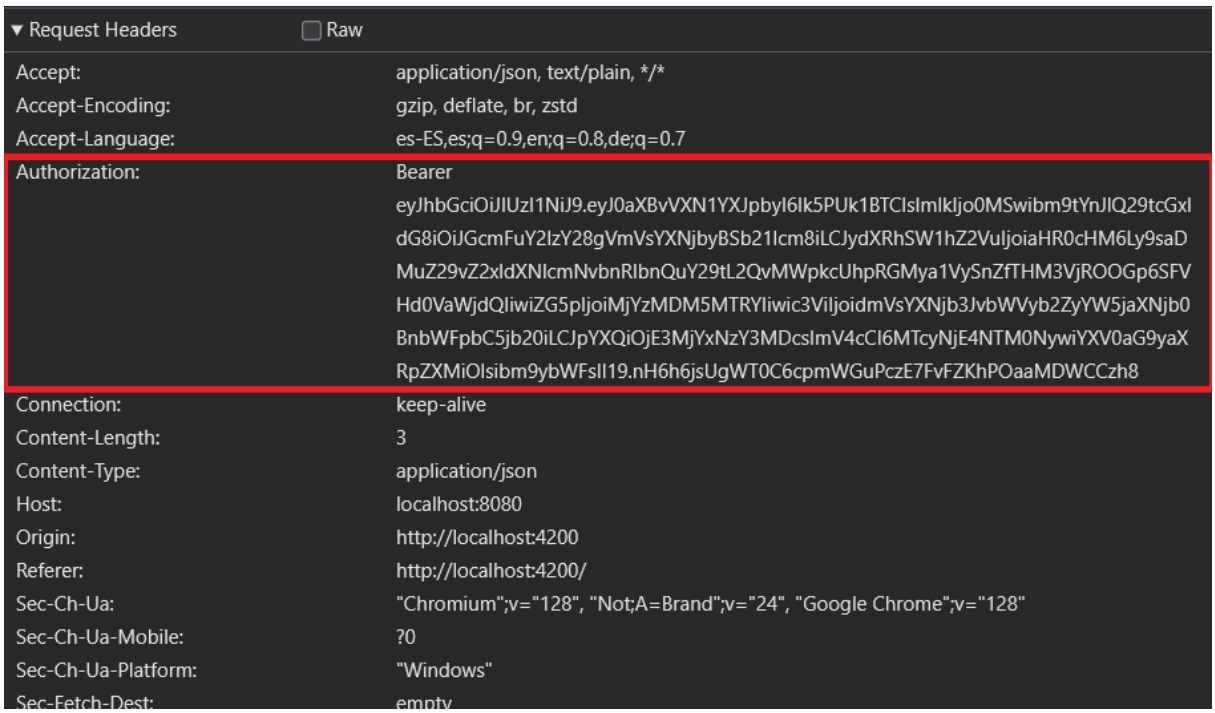
Como podemos observar, la primera parte del token (cabecera) indica el tipo de algoritmo empleado para firmar el token.

En la siguiente parte encontramos el payload, el cual está compuesto por una serie de *claims*. Hay una serie de *claims* predefinidos como sub (subject) que es el identificador único del usuario, iat (Issued At) que corresponde con el momento en el que se emitió el token, exp (Expiration) que conforma la fecha y la hora en la que expira el token. Adicionalmente, se pueden añadir *claims* opcionales que puedan ayudar en el proceso de identificación del usuario como puede ser en nuestro caso el tipo de usuario, id, nombre completo, ruta de la imagen y *authorities* que los empleamos para evitar llamadas adicionales al *backend* para obtener información sobre la sesión del usuario.

Finalmente, encontramos la firma, que consiste en una cadena generada por un algoritmo criptográfico empleado para verificar la integridad del documento JSON del *payload*.

Cuando un usuario se autentica en la aplicación, Spring Security genera un JWT que se envía al cliente. Este token se incluye en las cabeceras de las solicitudes posteriores, permitiendo al servidor identificar y autorizar al usuario sin necesidad de mantener una sesión en el servidor. Al recibir una solicitud, Spring Security se encarga de verificar la validez del JWT, garantizando que el usuario tiene los permisos necesarios para acceder al recurso solicitado.

Encontraremos dicho token en la cabecera de la petición HTTP, concretamente en el campo "Authorization" y este llevará un prefijo "Bearer" seguido de un salto de línea con el token.



▼ Request Headers	Raw
Accept:	application/json, text/plain, */*
Accept-Encoding:	gzip, deflate, br, zstd
Accept-Language:	es-ES,es;q=0.9,en;q=0.8,de;q=0.7
Authorization:	Bearer eyJhbGciOiJIUzI1NiJ9.eyJ0aXBvVXN1YXJpbyI6Ik5PUk1BTCIsImlkIjo0MSwibm9tYnJlIQ29tcGxl dG8iOiJGcmFuY2lzY28gVmVsYXNjbyBSb21lcm8iLCJydXRhSW1hZ2VuIjoiaHR0cHM6Ly9saD MuZ29vZ2xldXNlcmNvbnRlbnQuY29tL2QvMWpkcUhpRGMya1VySnZfTHM3VjROOGp6SFV Hd0VaWjdQIiwiZG5pIjoiMjYzMDM5MTRYIiwic3ViIjoidmVsYXNjb3JvbWVyb2ZyYW5jaXNjb0 BnbWFpbC5jb20iLCJpYXQiOjE3MjYxNzY3MDcsImV4cCI6MTcyNjE4NTM0NywiYXV0aG9yaX RpZXMiOlsibm9ybWFsIl19.nH6h6jsUgWT0C6cpmWGuPczE7FvFZKhPOaaMDWCCzh8
Connection:	keep-alive
Content-Length:	3
Content-Type:	application/json
Host:	localhost:8080
Origin:	http://localhost:4200
Referer:	http://localhost:4200/
Sec-Ch-Ua:	"Chromium";v="128", "Not;A=Brand";v="24", "Google Chrome";v="128"
Sec-Ch-Ua-Mobile:	?0
Sec-Ch-Ua-Platform:	"Windows"
Sec-Fetch-Dest:	empty

Figura 23.- Cabecera token

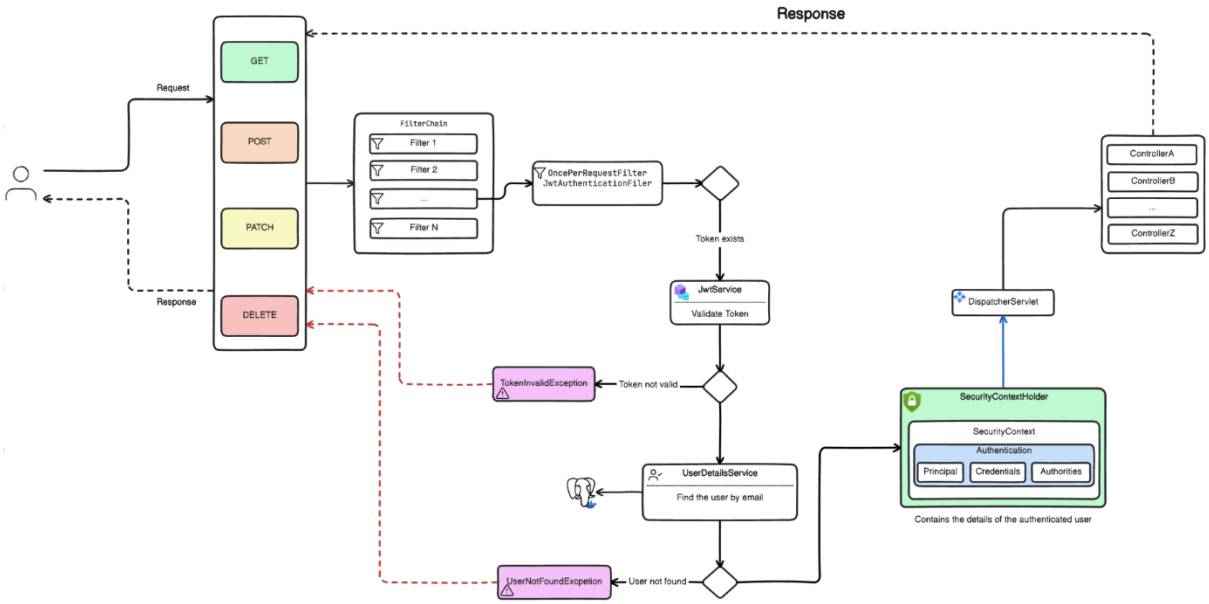


Figura 24.- Spring Security JWT [2]

3.2.2 Endpoints

Todos los endpoints de nuestra aplicación utilizan el prefijo "/api/v1", por lo que, en adelante, al describir los endpoints, se omitirá dicho prefijo para reducir espacio. A continuación vamos a ver cada uno según su controlador. Se va a proporcionar el método REST, una ruta y un cuerpo en formato JSON (si se requiere) o parámetro de URL. Se asume para los casos que se van a mostrar a continuación que, en caso de fallo, se va a devolver un objeto de error el cual contiene un código y un mensaje de error destinado al usuario.

Figura 25.- Ejemplo de mensaje de error REST

OpenAPI

OpenAPI, anteriormente conocido como Swagger, es una especificación que permite a los desarrolladores describir y documentar las interfaces de programación de aplicaciones (APIs) de manera estandarizada. La especificación OpenAPI se utiliza para crear una documentación interactiva y detallada de las APIs RESTful, facilitando la comprensión y el uso de estas interfaces tanto para desarrolladores como para consumidores de servicios.

A continuación, mostramos la especificación OpenAPI de nuestra aplicación que se puede consultar utilizando el prefijo "swagger-ui/index.html#".

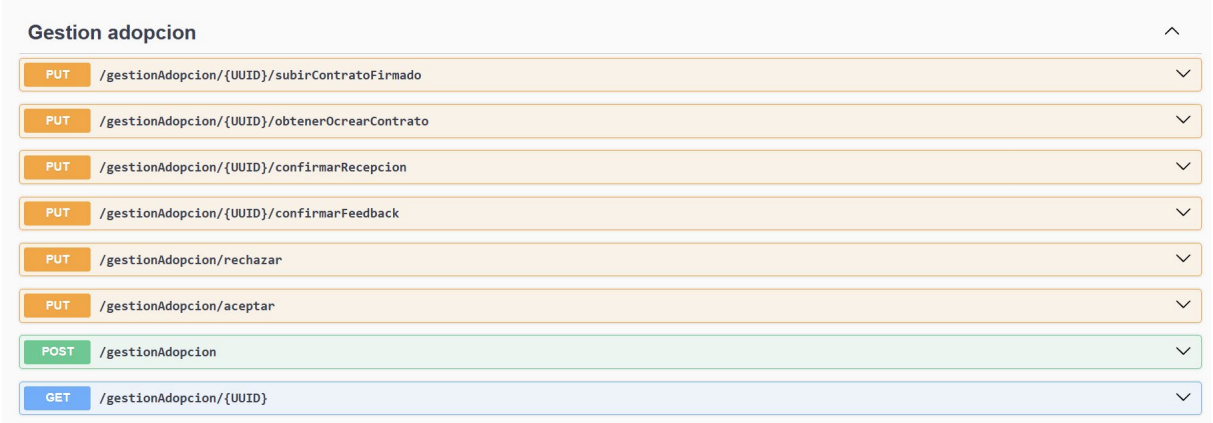


Figura 26.- Gestión adopción

Figura 27.- Animal

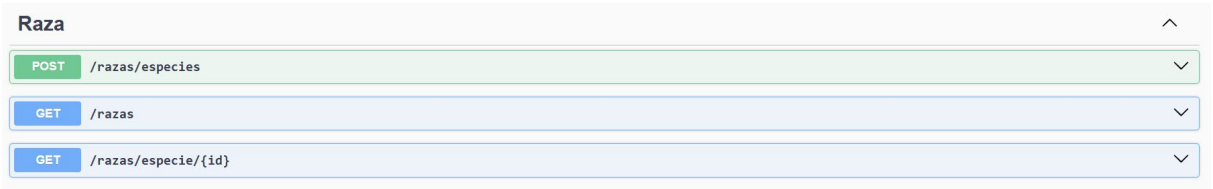


Figura 28.- Raza

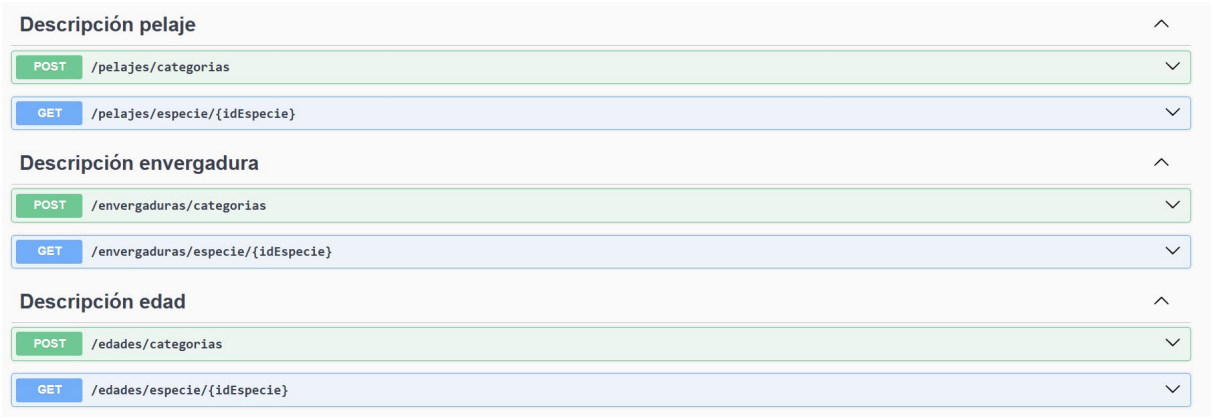


Figura 29.- Descripciones

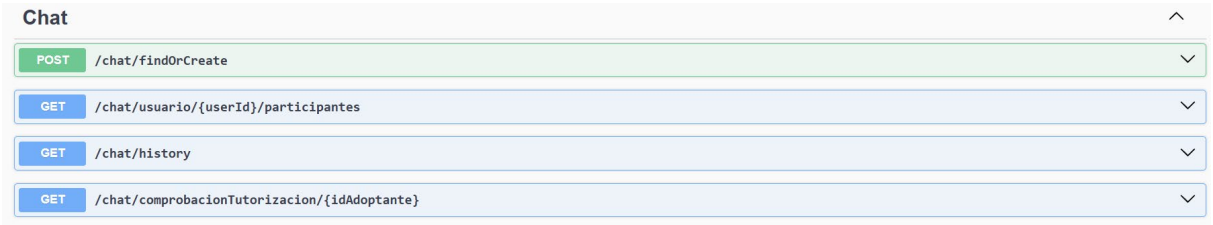


Figura 30.- Chat

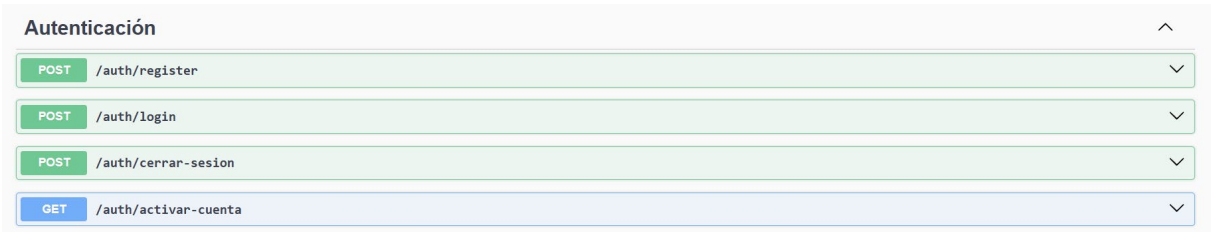


Figura 31.- Autenticación

Figura 32.- Múltiples catálogos

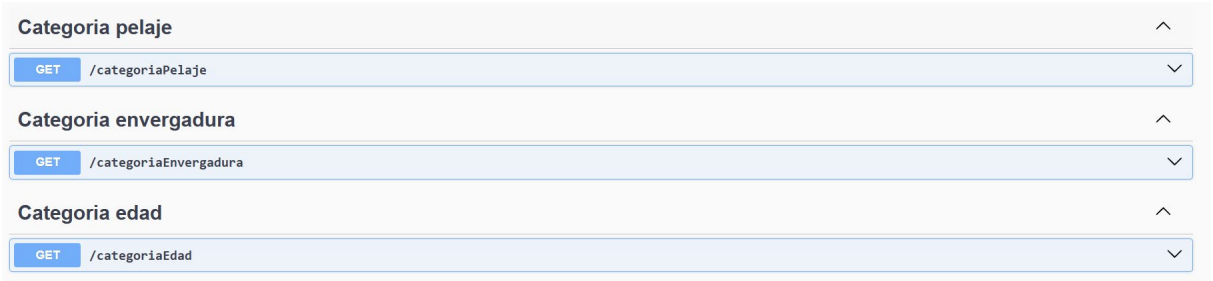


Figura 33.- Categorías

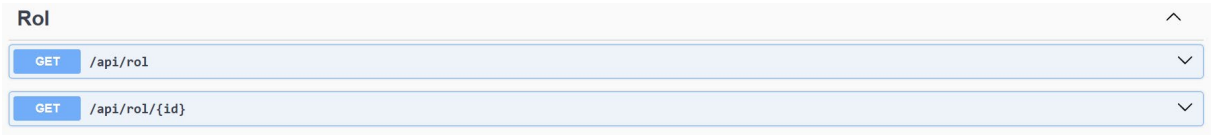


Figura 34.- Rol

Autenticación

POST	/auth/register	Datos necesarios para el registro de un usuario {

		"nombre": "string", "apellidos": "string", "password": "string", "dni": "string", "fechaNacimiento": date, "telefono": "string", "email": "string", "direcciones": ["string"], "nombreAsociacion": "string", "cif": "string", "numRegistro": "string", "tipoUsuario": "NORMAL" || "PROTECTORA" } Se pedirá "nombre", "apellidos" y "dni" o "nombreAsociacion", "cif" y "numRegistro" según si el "tipoUsuario" es normal o protectora.

Este endpoint se utiliza para poder registrar a un nuevo usuario en la aplicación. Se llevará a cabo una primera comprobación para confirmar que dicho usuario no exista ya en la aplicación. Los datos del usuario pasarán por una serie de comprobaciones que variarán según si es un individual o una protectora. Si todo ha sido correcto se creará el usuario en la base de datos con una cuenta no confirmada y se devolverá un token el cuál servirá para que el usuario pueda activar su cuenta. El código de activación se asociará al token y tendrá un tiempo de expiración, el usuario lo recibirá por email de forma automatizada.

PUT	/auth /activar-cuenta	

Nos permite activar la cuenta de un usuario que se acabe de registrar en la aplicación.
Una vez creada la cuenta, el usuario debe activarla con el token que se le generó en el registro.
Si el token se encuentra expirado, se generará otro código de activación y se enviará por

correo de nuevo. Introduciendo el código correcto y no expirado resultará en la activación de la cuenta.

POST	/auth /login	Credenciales de inicio de sesión
		{
		"email": "string",
		"password": "string"
		}

Permite al usuario iniciar sesión comprobando sus credenciales. Al introducir el correo y la contraseña correcta se devolverá el token del usuario.

POST	/auth /cerrar-sesion	

Este endpoint va a ser llamado cuando un usuario quiera cerrar sesión. Esta petición va a limpiar el contexto de seguridad en spring por lo que ya no se considerará que el usuario esté "logueado" y no podrá acceder a ciertos recursos protegidos hasta que vuelva a iniciar sesión. Se requiere que el usuario haya iniciado sesión previamente y por lo tanto tenga un token válido en la cabecera de la petición.

Animal

GET	/animales/	

Obtiene todos los animales disponibles para adopción. Se emplea en la página principal para mostrar todos los animales inicialmente.

GET	/animales/{id}	Id del animal a obtener

Obtiene el animal asociado al id pasado por parámetro. Se utiliza para obtener el animal del que se quiere consultar el perfil.

POST	/animales/archivos/subirImagenes	Las imagenes por MULTIPART. Multipart es un formato que envía el archivo en binario junto

		con cabeceras con el nombre del archivo y el tipo.

Se suben las imagenes del animal a través de la API de Google Drive. El nombre de cada imagen subida tiene el siguiente formato: id del usuario + "_" + fecha actual + "(" + contador del nº de imagen subida + ")" + extensión. Se devuelve un string con la url de cada imagen separada por ";".

POST	/animales/archivos/subirHistoriaClinica	La historia clínica por MULTIPART. Multipart es un formato que envía el archivo en binario junto con cabeceras con el nombre del archivo y el tipo.

Subida de la historia clínica del animal a través de la API de Google Drive. Se devuelve un string con la url de la historia clínica. El nombre de la historia clínica subida tiene el siguiente formato: id del usuario + "_" + fecha actual + extensión. Se devuelve un string con la url de la historia clínica.

POST	/animales/	Datos obligatorios para la creación del animal
		
		{
		"nombre": "string",
		"edad": number,
		"peso": number,
		"rutaImagen": "string",
		"localizacion": {
		"calle": "string",
		"codigoPostal": "string",
		"ciudad": "string",
		"provincia": "string",
		"datosAdicionales": "string",
		"observaciones": "string",
		"coordenadas": {

		"x": number,
		"y": number
		}
		},
		"descripcion": "string",
		"categoriaEdad": {
		"id": number
		},
		"categoriaEnvergadura": {
		"id": number
		},
		"categoriaPelaje": {
		"id": number
		},
		"fechaNacimiento": date,
		"especie": {
		"id": number
		},
		"genero": {
		"id": number
		},
		"colores": [
		{
		"id": number
		}
		],
		"razas": [
		{
		"id": number
		}
		]

		},
		"historiaClinica": {
		"ruta": "string"
		},
		"vacunasAldia": "SI" || "NO" || "DESCONOCIDO",
		"esterilizado": "SI" || "NO" || "DESCONOCIDO",
		"chip": "SI" || "NO" || "DESCONOCIDO",
		"entrenadoParaCasa": "SI" || "NO" || "DESCONOCIDO",
		"seLlevaBienConPerros": "SI" || "NO" || "DESCONOCIDO",
		"seLlevaBienConGatos": "SI" || "NO" || "DESCONOCIDO",
		"seLlevaBienConNinos": "SI" || "NO" || "DESCONOCIDO",
		"necesidadesEspeciales": "SI" || "NO" || "DESCONOCIDO",
		"problemasDeComportamiento": "SI" || "NO" || "DESCONOCIDO""
		}

Endpoint destinado a dar de alta a un animal en la aplicación y crear su perfil. Una vez se lleva a cabo la petición con los datos obligatorios, se crea el animal en la base de datos y se devuelve la URI del animal creado.

PUT	/animales/	Datos actuales del animal con campos actualizados
		
		{
		"nombre": "string",
		"edad": number,
		"peso": number,
		"rutaImagen": "string",
		"localizacion": {
		"calle": "string",
		"codigoPostal": "string",
		"ciudad": "string",
		"provincia": "string",
		"datosAdicionales": "string",

		"observaciones": "string",
		"coordenadas": {
		"x": number,
		"y": number
		}
		},
		"descripcion": "string",
		"categoriaEdad": {
		"id": number
		},
		"categoriaEnvergadura": {
		"id": number
		},
		"categoriaPelaje": {
		"id": number
		},
		"fechaNacimiento": date,
		"especie": {
		"id": number
		},
		"genero": {
		"id": number
		},
		"colores": [
		{
		"id": number
		}
		],
		"razas": [
		{
		"id": number

		}
		]
		},
		"historiaClinica": {
		"ruta": "string"
		},
		"vacunasAldia": "SI" || "NO" || "DESCONOCIDO",
		"esterilizado": "SI" || "NO" || "DESCONOCIDO",
		"chip": "SI" || "NO" || "DESCONOCIDO",
		"entrenadoParaCasa": "SI" || "NO" || "DESCONOCIDO",
		"seLlevaBienConPerros": "SI" || "NO" || "DESCONOCIDO",
		"seLlevaBienConGatos": "SI" || "NO" || "DESCONOCIDO",
		"seLlevaBienConNinos": "SI" || "NO" || "DESCONOCIDO",
		"necesidadesEspeciales": "SI" || "NO" || "DESCONOCIDO",
		"problemasDeComportamiento": "SI" || "NO" || "DESCONOCIDO""
		}

Endpoint destinado a la modificación de un animal ya existente. Se revisa nuevamente si siguen completados los campos obligatorios necesarios para la creación del animal y se lleva a cabo la actualización de este siempre que no esté dado de baja de la aplicación y sea el propietario o un administrador el que lanza la petición.

DELETE	/animales/archivos/eliminarImagenes	Un string con las urls a eliminar por un parámetro URL llamado Imagenes /animales/eliminarImagenes?Imagenes=imagen1Url

Su funcionalidad consiste en eliminar las imagenes que se le pasan por ruta, bien porque se haya producido un fallo en la creación del animal o bien porque se hayan editado. La cadena de texto está conformada por urls de las imagenes separadas por ";". Estas se formatean y se eliminan individualmente. Si el proceso ha sido correcto, se envía una respuesta de confirmación.

DELETE	/animales/archivos/eliminarHistoriaClinica	Un string consistente en la URL a eliminar a través de un parámetro URL llamado /animales/archivos/eliminarHistoriaClinica?HistoriaClinica=HistoriaUrl

Empleado para eliminar la historia clínica de un animal, ya sea por fallo al crearse o por modificación del usuario (subir una nueva).

Eliminación de la historia clínica a través de la URL recibida. Si el proceso ha sido correcto, se envía una respuesta de confirmación.

POST	/animales/especies	Una lista de ids correspondientes a las especies {[idEspecie1, idEspecie2, idEspecie3]}

Obtiene todos los animales cuyo id de especie coincida con alguno de los recibidos en la petición.

POST	/animales/filtrar	Un objeto FiltroRequest que contiene una serie de campos necesarios para poder realizar el filtrado de animales
		{
		"filtros": {[
		"nombreCampo": "string",
		"valor": object
		]},
		"orden": "ASC" || "DESC",
		campoOrden: "string"
		}
		
		CampoOrden corresponde al nombre del campo por el cuál se quiere ordenar, en nuestra aplicación actualmente siempre será "fechaAlta".

Obtiene todos los animales que cumplan con los filtros recibidos en la petición.

DELETE	/animales/{id}	Id del animal a dar de baja

El animal cuyo id coincida con el recibido en la petición será dado de baja en la aplicación. Consiste en un borrado lógico en el que se asigna una fecha de baja y el usuario que ha realizado la acción.

GET	/animales/adopcion/propietario/{id}	Id del propietario

Obtención de todos los animales que se encuentren en la página para ser adoptados cuyo propietario tenga el id recibido por parámetro.

GET	/animales/exportar	

Si el usuario que realiza la petición es un administrador, recibirá un Excel con los datos de todos los animales registrados en la aplicación que no estén dados de baja.

Chat

POST	/chat/findOrCreate	Id de dos usuarios entre los que se quiere crear un chat
		{
		"user1Id": number,
		"user2Id": number
		}

Se lleva a cabo una búsqueda para intentar encontrar un chat existente entre los dos usuarios. Si no existe ninguno, se crea y se devuelve los datos del chat.

GET	/chat/history	Id del chat por parámetro URL /chat/history?chatId=idChat

Recupera todos los mensajes del chat con el id correspondiente, se utiliza siempre que se cargue un chat existente.

Categoría edad

GET	/categoriaEdad/	

Recupera todas las categorías de edad existentes.

Categoría envergadura

GET	/categoriaEnvergadura/	

Recupera todas las categorías de envergadura existentes.

Categoría pelaje

GET	/categoriaPelaje/	

Recupera todas las categorías de pelaje existentes.

Color

GET	/color/	

Recupera todos los colores existentes.

GET	/color/especie/{idEspecie}	Id de la especie de interés

Recupera todos los colores existentes que tiene la especie que corresponde al id del parámetro recibido.

POST	/color/especies	Lista de ids de las especies de interés {[idEspecie1, idEspecie2, idEspecie3]}

Recupera todos los colores existentes los cuales estén asociados al menos a uno de los ids de las especies recibidos.

Descripcion edad

GET	/edades/especie/{idEspecie}	Id de la especie

Recupera todas las descripciones de edad disponibles que tengan asociado el id de esa especie y devuelve todas las categorías de edad asociadas a estas descripciones.

POST	/edades/categorias	Lista de Ids de las especies de interés {[idEspecie1, idEspecie2, idEspecie3]}

Recupera todas las descripciones de edad disponibles que tengan asociado algún id de la lista de especies y devuelve todas las categorías de edad asociadas a estas descripciones.

Descripción envergadura

GET	/envergaduras/especie/{idEspecie}	Id de la especie

Recupera todas las descripciones de envergadura disponibles que tengan asociado el id de esa especie y devuelve todas las categorías de envergadura asociadas a estas descripciones.

POST	/envergaduras/categorias	Lista de ids de las especies de interés {[idEspecie1, idEspecie2, idEspecie3]}

Recupera todas las descripciones de envergadura disponibles que tengan asociado algún id de la lista de especies y devuelve todas las categorías de envergadura asociadas a estas descripciones.

Descripción pelaje

GET	/pelajes/especie/{idEspecie}	Id de la especie

Recupera todas las descripciones de pelaje disponibles que tengan asociado el id de esa especie y devuelve todas las categorías de pelaje asociadas a estas descripciones.

POST	/pelajes/categorias	Lista de ids de las especies de interés {[idEspecie1, idEspecie2, idEspecie3]}

Recupera todas las descripciones de pelaje disponibles que tengan asociado algún id de la lista de especies y devuelve todas las categorías de pelaje asociadas a estas descripciones.

Especie

GET	/especies/	

Recupera todas las especies existentes.

Estado adopción

GET	/estadosAdopcion/	

Obtiene todos los estados de adopción existentes.

Genero

GET	/generos/	

Obtiene todos los géneros existentes.

Gestión adopción

POST	/gestionAdopcion/	Datos obligatorios para poder comenzar el proceso de adopción
		{
		"ultimoMsjConfirmado": boolean,
		"numMensajes": number,
		"fechaUltimoMensaje": date,
		"motivoRechazo": "string",
		"fechaRechazo": date,
		"rutaDocumento": "string",
		"uuid": "string",
		"estadoAdopcion": {
		"id": number
		},
		"adoptante": {
		"id": number
		},
		"animal": {
		"id": number,
		"propietario": {
		"id": number
		}
		}
		}

		"RutaDocumento" hace refencia a la ruta donde se almacena el contrato de adopción. "EstadoAdopcion" se emplea para poder seguir el estado del proceso. "UUID" es un identificador único. Propietario consiste en el dueño del animal que está en adopción. El resto de los campos se emplean para el seguimiento o rechazo del proceso de adopción.

Creación de la gestión de adopción, que es el objeto que se crea para poder llevar a cabo el proceso de adopción. A esta gestión de adopción se le genera un UUID para poder identificar cada proceso. Se envía un correo al propietario del animal para que pueda elegir si rechazar o aceptar el proceso.

GET	/gestionAdopcion/{UUID}	Identificador único (UUID)

Se obtiene el objeto directriz del proceso de adopción a partir del UUID asociado a este.

PUT	/gestionAdopcion/rechazar	El objeto relacionado con el proceso de adopción
		{
		"ultimoMsjConfirmado": boolean,
		"numMensajes": number,
		"fechaUltimoMensaje": date,
		"motivoRechazo": "string",
		"fechaRechazo": date,
		"rutaDocumento": "string",
		"uuid": "string",
		"estadoAdopcion": {
		"id": number
		},
		"adoptante": {
		"id": number
		},
		"animal": {
		"id": number,

		"propietario": {
		"id": number
		}
		}
		}
		
		En este caso el estado de adopción vendrá relleno con el id del estado correspondiente al estado "petición rechazada". También se rellenará la "fechaRechazo" y, opcionalmente, el motivo del rechazo.

Se rechaza el proceso de adopción. Se envía correo informativo al adoptante, se cancela el proceso y el animal sigue disponible para adoptar.

PUT	/gestionAdopcion/aceptar	El objeto relacionado con el proceso de adopción
		{
		"ultimoMsjConfirmado": true,
		"numMensajes": number,
		"fechaUltimoMensaje": date,
		"motivoRechazo": "string",
		"fechaRechazo": date,
		"rutaDocumento": "string",
		"uuid": "string",
		"estadoAdopcion": {
		"id": number
		},
		"adoptante": {
		"id": number
		},
		"animal": {
		"id": number,
		"propietario": {

		"id": number } } } En este caso el estado de adopción vendrá relleno con el id del estado correspondiente al estado "petición aceptada".

Se acepta el proceso de adopción. Se envía correo informativo al adoptante y además se le avisa de que tiene que firmar el contrato de adopción para poder seguir el proceso.

PUT	/gestionAdopcion/{UUID}/obtenerOcrearContrato	El UUID del proceso de adopción

Una vez aceptado el proceso de adopción se procede a crear el contrato de adopción. Una vez creado deben poder acceder a él adoptante y propietario obteniéndolo de la base de datos.

PUT	/gestionAdopcion/{UUID}/subirContratoFirmado	El UUID del proceso de adopción y el contrato de adopción por multipart. Multipart es un formato que envía el archivo en binario junto con cabeceras con el nombre del archivo y el tipo.

Una vez firmado el contrato, este se subirá de nuevo, sustituyendo al que se encuentre en ese momento en la base de datos. En la primera parte no tendrá ninguna firma, después le llegará al propietario el contrato firmado y este, firmará y se convertirá en el contrato final dando comienzo a parte final del proceso adoptivo.

Raza

GET	/razas/	

Obtiene todas las razas existentes.

GET	/razas/especie/{id}	Id de la especie interesada {[idEspecie1, idEspecie2, idEspecie3]}

Obtiene la especie asociada al id del parámetro recibido.

GET	/razas/especies	Lista de ids de las especies interesadas {[idEspecie1, idEspecie2, idEspecie3]}

Obtiene todas las especies cuyo id aparezca al menos una vez en la lista recibida.

Rol

GET	/rol/	

Obtiene todos los roles existentes.

GET	/rol/{id}	Id del rol a obtener

Obtiene el rol asociado a ese id

Usuario

GET	/usuario/{id}	Id del rol usuario a obtener

Obtiene el usuario asociado a ese id

GET	/usuario/exportar	

Este endpoint solo está permitido para los administradores. Devuelve un Excel con los datos de todos los usuarios registrados en la aplicación.

3.3 Frontend

El frontend es el componente responsable de gestionar la interacción del usuario, proporcionando una experiencia visualmente atractiva y fluida.

Se ha elegido Angular como framework debido a que nos permite construir interfaces de usuario dinámicas y eficientes, aprovechando componentes reutilizables. Además, facilita la integración con la API RESTful del backend.

El frontend actúa como intermediario entre el usuario y el backend, recoge la información introducida por parte del usuario, la valida, la envía al backend y presenta la respuesta de forma clara y accesible.

3.3.1 Seguridad

En la parte de seguridad del frontend, se implementa un mecanismo para proteger el acceso a las distintas rutas de la aplicación mediante el uso de AuthGuard en Angular. AuthGuard es un servicio que actúa como un guardián de las rutas, asegurándose de que solo los usuarios autenticados puedan acceder a determinadas páginas.

Cada vez que un usuario intenta navegar a una ruta protegida, AuthGuard verifica si el token JWT (JSON Web Token) almacenado en el navegador es válido. Si el token es válido y no ha expirado, el acceso a la ruta se permite. En caso contrario, el usuario es redirigido a la página de inicio de sesión.

Esta capa de seguridad garantiza que solo los usuarios autorizados, con un token activo, puedan interactuar con las áreas protegidas de la aplicación, protegiendo así los datos y funcionalidades sensibles del sistema.

DISEÑO DE LA INTERFAZ GRÁFICA DE USUARIO

Hemos decidido elegir un color esmeralda porque creemos que es un color que representa la naturaleza y, de forma más específica, la conexión con los animales. A este esmeralda se le ha añadido un color que pueda contrastar y ser fácil de leer como es el caso del color blanco.
El diseño de la página va a girar en torno a esta paleta de colores. La disposición va a tender a extenderse al ancho de la página siempre con unos márgenes externos mínimos. Los distintos módulos van a seguir una estructura cuadrada o rectangular en forma de tarjeta redondeada.

4

Implementación

Procedemos a ver las principales rutas de la aplicación web y sus funcionalidades:

Inicio de sesión(/login)

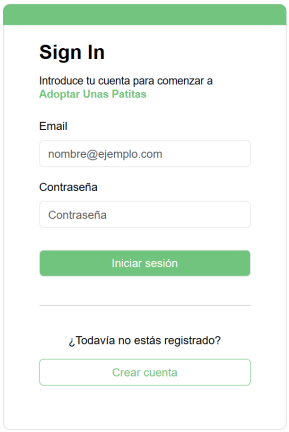


Figura 35.- Inicio de sesión

Ofrece las opciones de iniciar sesión introduciendo email y contraseña o bien un botón que lleve a la página de registro.

Registro(/register)

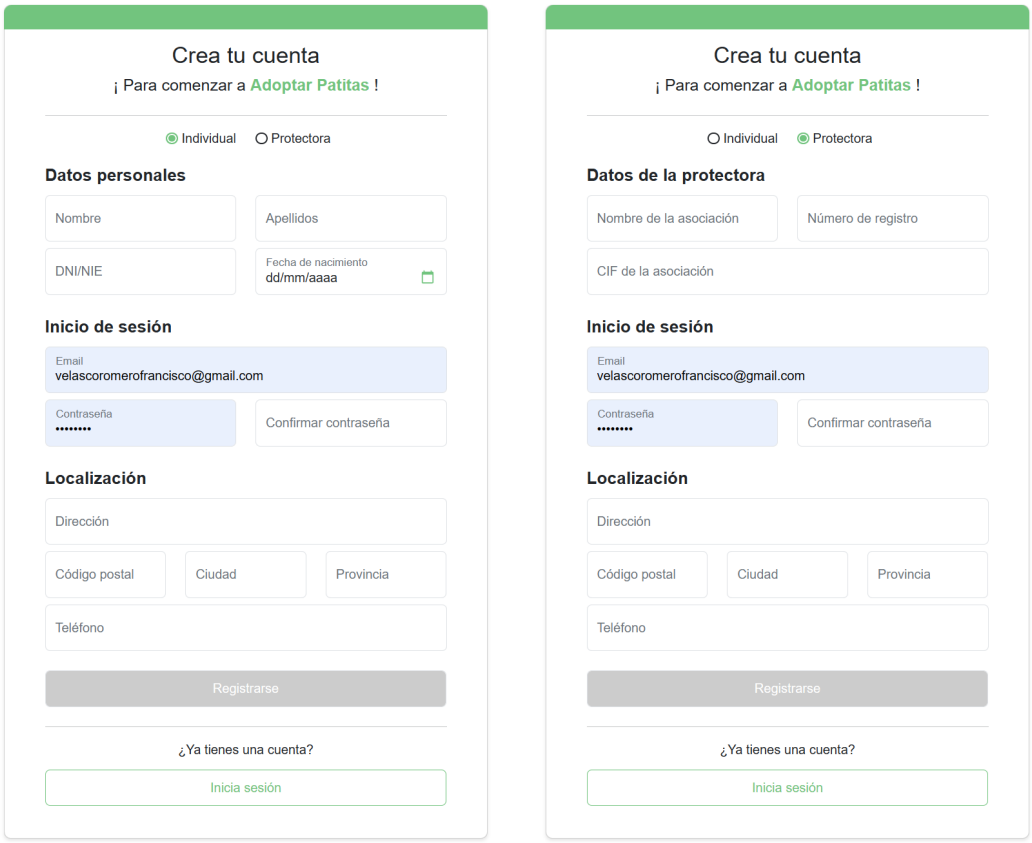


Figura 36.- Registro

Se ofrecen las opciones de registrar cualquiera de los dos tipos de usuarios, cambiando de forma dinámica la información requerida y las comprobaciones. El botón "Registrarse" no estará disponible hasta que todos los campos estén rellenos y correctos.

Abajo del todo se ofrece un botón para poder volver al inicio de sesión.

Activar cuenta(/activar-cuenta)

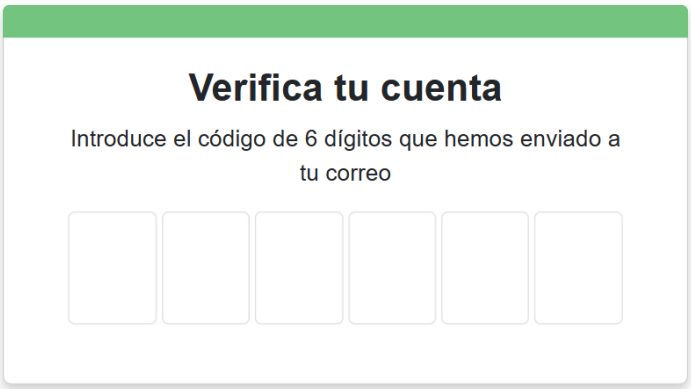
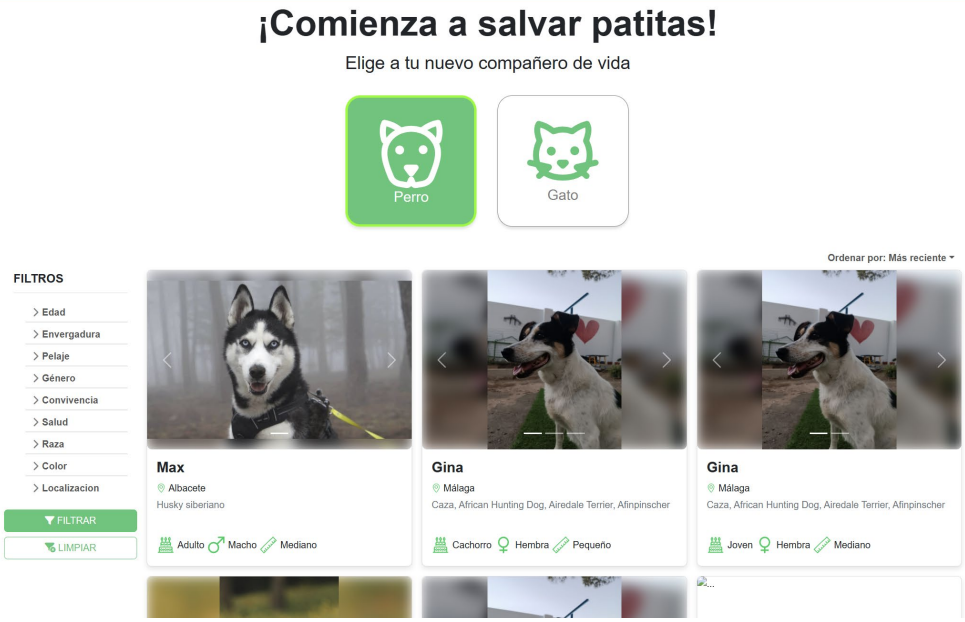


Figura 37.- Activar cuenta

Para activar la cuenta el usuario deberá introducir los 6 dígitos que recibirá en su correo.

Main(/main)



Página principal a la que se puede acceder sin necesidad de iniciar sesión. Ofrece la opción de elegir entre las especies disponibles. Una vez elegida la especie se mostrarán los filtros y las tarjetas de cada animal con su información más relevante.

Se mostrarán botones para aplicar los filtros, limpiarlos y cambiar el orden de los resultados.

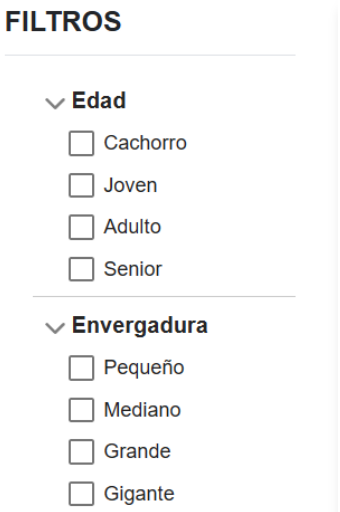


Figura 38.- Filtros (Parte 1)

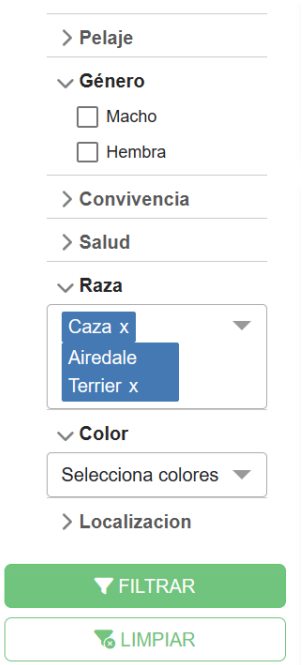


Figura 39.- Filtros (Parte 2)

Perfil animal(/animales/{id})

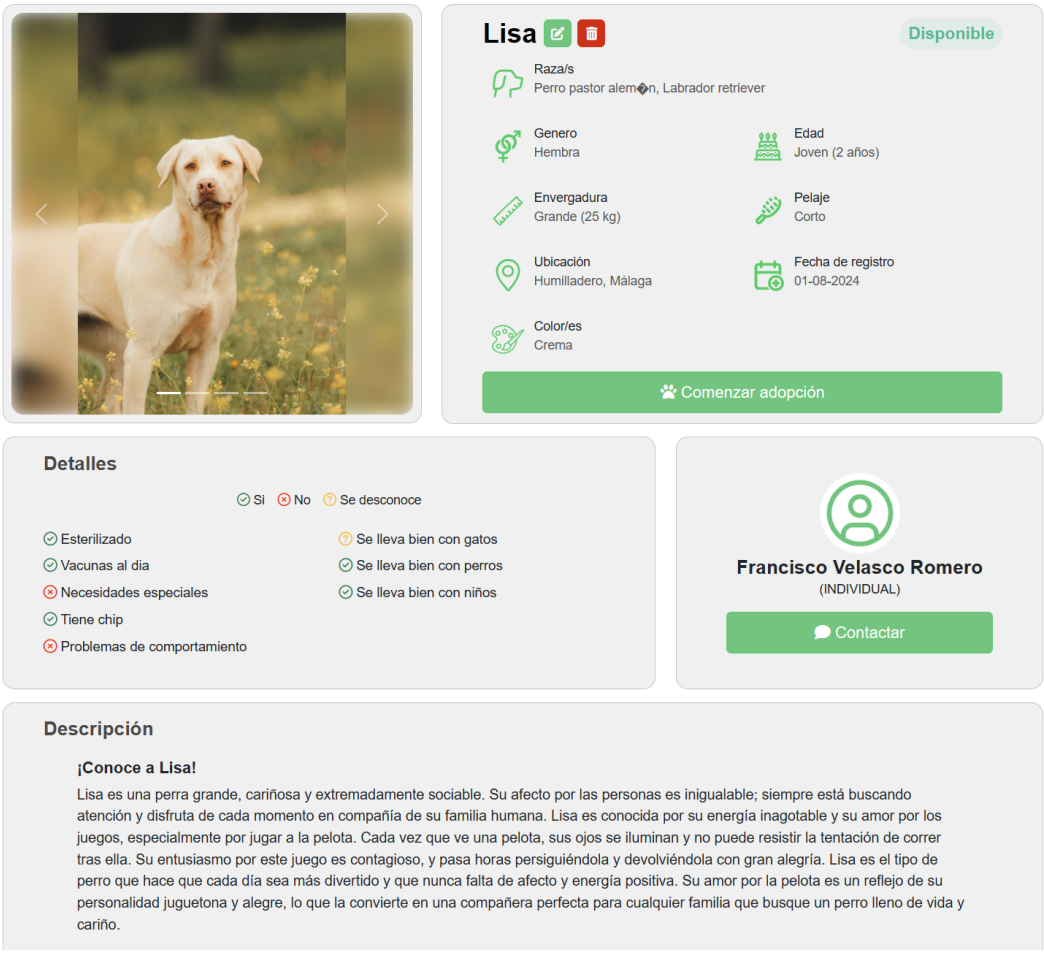


Figura 40.- Perfil animal (primera parte)

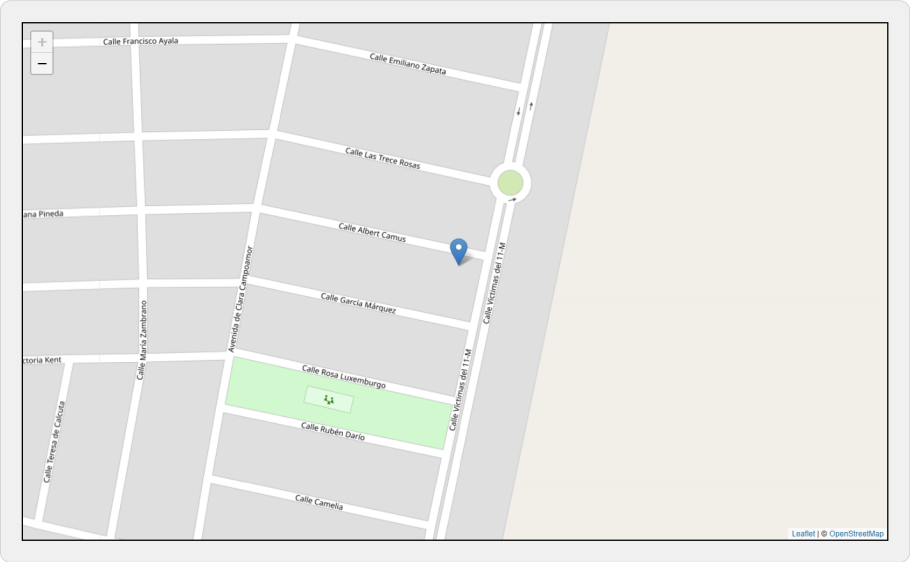


Figura 41.- Perfil animal (segunda parte)

Encontramos varias partes, un carrusel de fotos del animal, una parte con los datos más relevantes, detalles de convivencia, datos del propietario usuario, una descripción y la ubicación.

Se muestran botones para contactar con el usuario, comenzar el proceso de adopción y en el caso de ser el propietario, botones de editar y borrar.

Editar animal(/editar-animal/{id}) y añadir animal(/registrarAnimal)

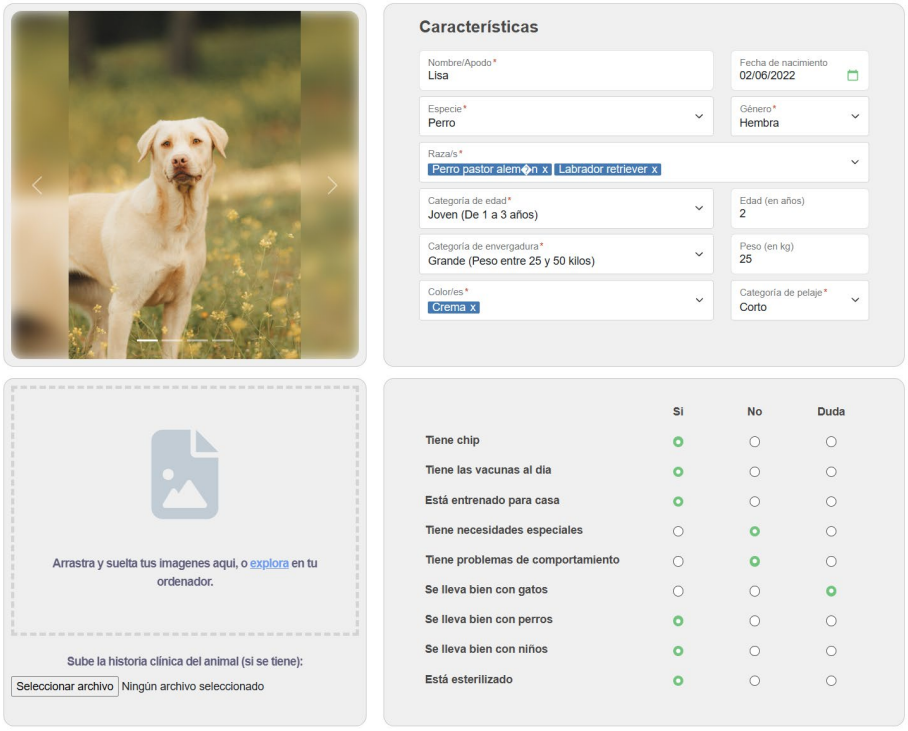


Figura 42.- Editar animal (primera parte)

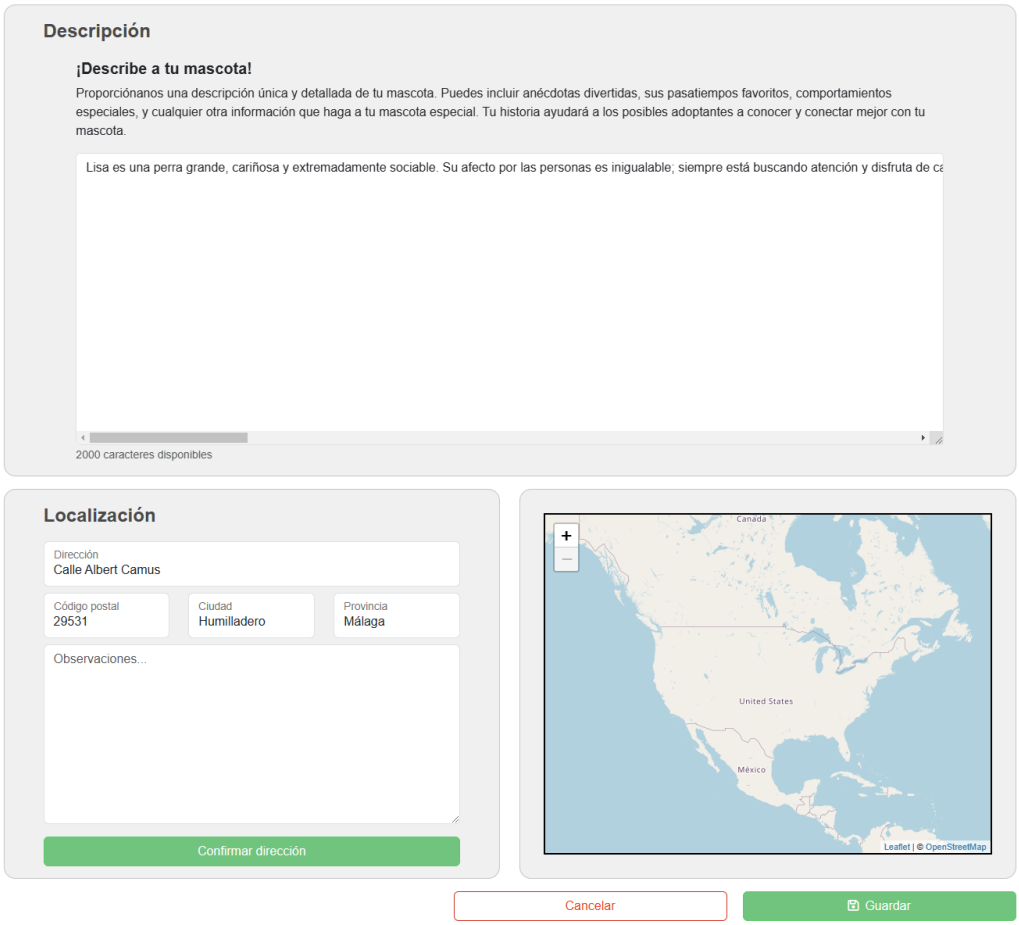


Figura 43.- Editar animal (segunda parte)

Encontramos un carrusel para visualizar las imágenes subidas, campos y desplegables disponibles a ser modificados para introducir la nueva información del animal, un área para poder arrastrar imágenes a la subida, un botón de seleccionar archivo para subir la historia clínica y unos botones de estilo "radio" para seleccionar una de las tres opciones disponibles del apartado de convivencia. A continuación, encontramos un campo donde se podrá introducir la descripción del animal, con un contador visual de palabras para que el usuario sepa en todo momento los caracteres restantes. Finalmente encontramos un campo donde introducir la dirección del animal que, al confirmar, se verá reflejado en el mapa. Se mostrarán dos botones, uno para cancelar la modificación y otro para confirmar.

La única diferencia entre editar el animal y crearlo será que en el caso de editar animal los valores ya vendrán rellenos según lo guardado en la base de datos.

Perfil usuario(/perfil-usuario/{id})

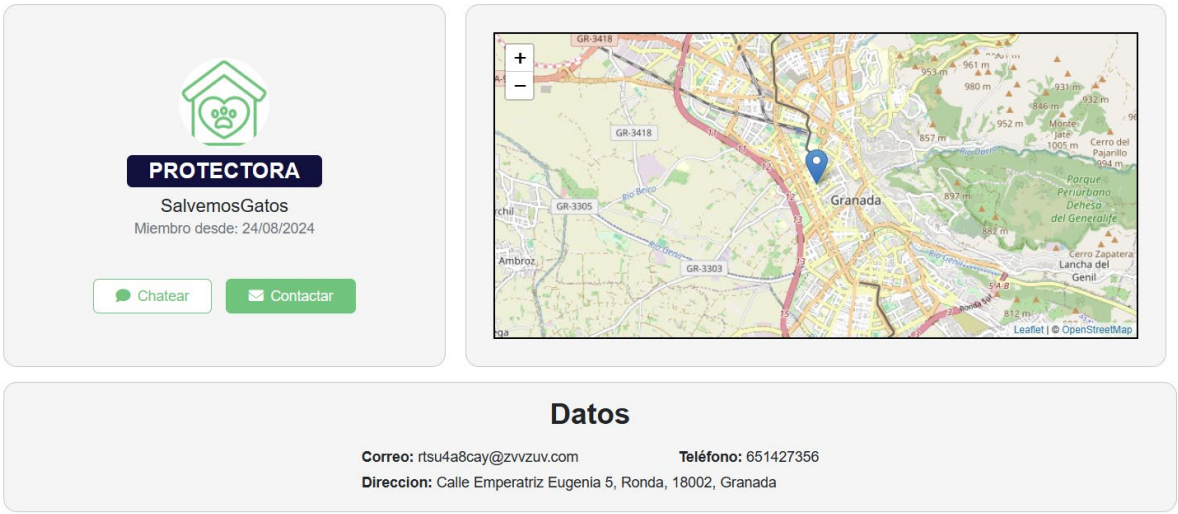


Figura 44.- Perfil usuario

Una simple vista del perfil del usuario en el que se ven reflejados sus datos, junto con las formas de contactar y la ubicación. La información sensible como DNI, CIF o dirección estará oculta y solo podrá acceder a ella el usuario correspondiente a dicho perfil.

Panel de usuario

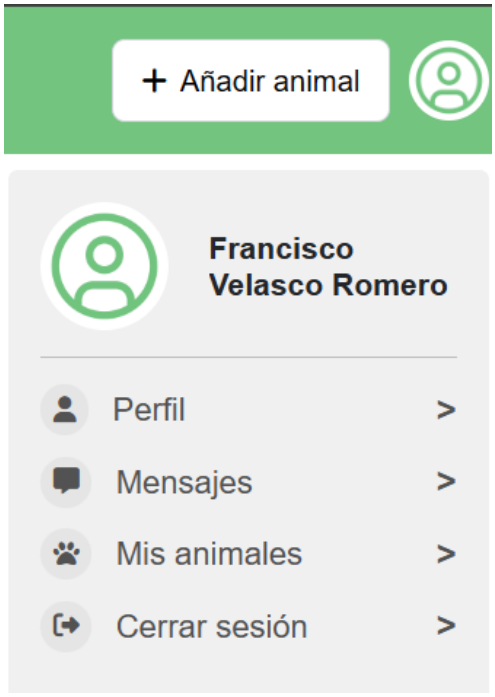


Figura 45.- Panel de control

Muestra los accesos directos básicos del usuario y una opción para cerrar sesión.

Animales en adopción del usuario actual(/animales-adopcion)

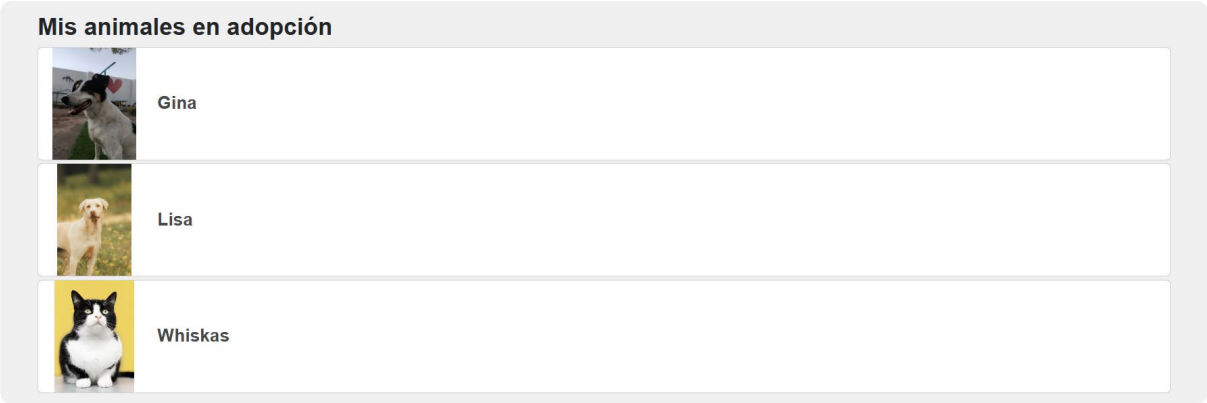


Figura 46.- Animales en adopción del usuario

Muestra una lista de animales en adopción por parte del usuario de la sesión. Sirven como enlaces directos a sus perfiles.

Chats del usuario actual(/chats)

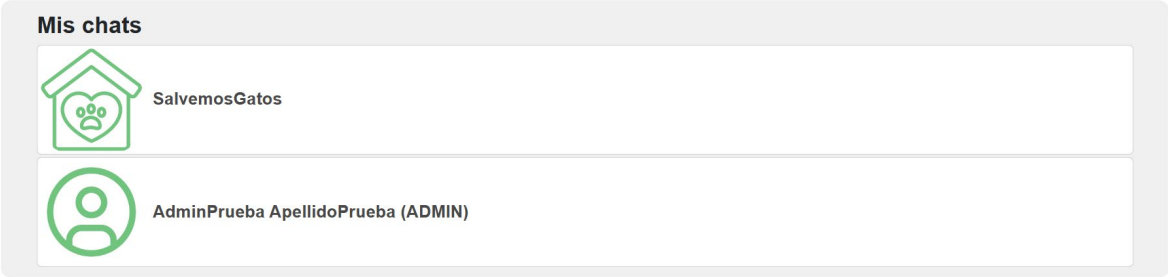


Figura 47.- Chats del usuario actual

Muestra una lista con los usuarios con los que tiene un chat abierto el usuario actual. Sirven como enlaces directos a su chat correspondiente.

Proceso de adopción(/gestion-adopcion/{UUID})

Figura 48.- Proceso adoptivo

Consiste en una página dinámica cuyo contenido va a variar en función de la fase adoptiva y del usuario que lo consulte. El proceso adoptivo utiliza un UUID y una serie de comprobaciones de usuario para diferenciar el proceso a reflejar para cada usuario implicado en este.
El proceso de adopción sigue el siguiente flujo:

1. Se inicia una petición de adopción por parte del adoptante. El propietario del animal será advertido a través de un correo electrónico automatizado.
2. En la página se ofrece la información sobre los detalles de la petición de adopción junto con dos botones de aceptar o rechazar adopción.
3. Si se rechaza el proceso acaba y el animal se mantiene en adopción. En caso de aceptarse, se genera un PDF con los datos del animal y primeramente se avisa al adoptante por correo para que firme su parte.
4. Una vez el adoptante sube su parte firmada, el propietario recibe un correo para firmar su parte y formalizar el proceso.
5. Una vez formalizado el proceso, se pide por correo al adoptante que confirme la recepción del animal.
6. Tras esto, el sistema de forma automática irá informando a través del correo al adoptante de cuándo tiene que dar el feedback del animal. Si pasado un día el tutor no ha recibido ninguna información, el sistema advertirá al adoptante y le informará de las repercusiones legales que podría haber sino se pone en contacto lo antes posible.

5

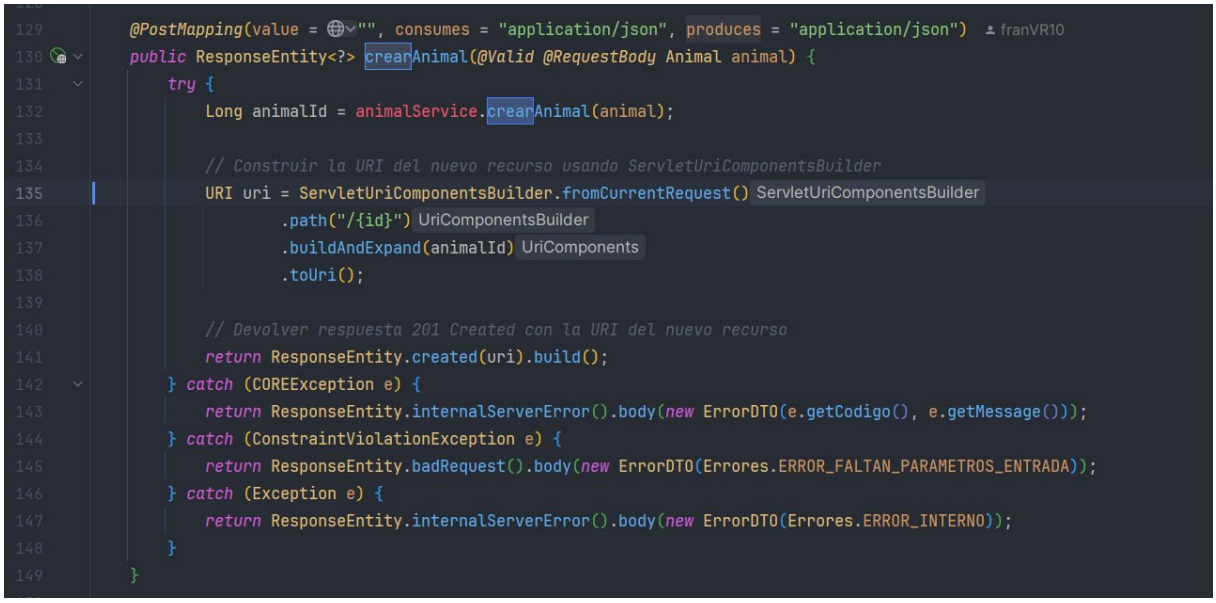
Tests

5.1 Backend

En la aplicación, utilizaremos JUnit y Mockito para realizar pruebas exhaustivas de los endpoints del backend. JUnit nos permitirá estructurar y ejecutar pruebas unitarias de manera eficiente, asegurando que cada componente del sistema funcione correctamente de forma aislada. Con Mockito, podremos simular comportamientos de dependencias externas y crear escenarios de prueba controlados, lo que facilita la verificación del correcto funcionamiento de los endpoints sin necesidad de interactuar con el entorno real o la base de datos. Esto nos ayudará a garantizar la calidad y fiabilidad del sistema antes de su despliegue.

Al ser una aplicación con una lógica de negocio compleja vamos a comentar la metodología de tests del endpoint de animales, ya que este mismo proceso se va a realizar con el resto de los módulos de la aplicación.

La estructura de estos tests siempre va a ser la misma: tenemos un endpoint el cuál puede devolver varias cosas, un mensaje de éxito, un error controlado, parámetros insuficientes y un error desconocido.



```
129    @PostMapping(value = "", consumes = "application/json", produces = "application/json")  franVR10
130    public ResponseEntity<?> crearAnimal(@Valid @RequestBody Animal animal) {
131        try {
132            Long animalId = animalService.crearAnimal(animal);
133
134            // Construir la URI del nuevo recurso usando ServletUriComponentsBuilder
135            URI uri = ServletUriComponentsBuilder.fromCurrentRequest() ServletUriComponentsBuilder
136                    .path("/{id}") UriComponentsBuilder
137                    .buildAndExpand(animalId) UriComponents
138                    .toUri();
139
140            // Devolver respuesta 201 Created con la URI del nuevo recurso
141            return ResponseEntity.created(uri).build();
142        } catch (COREException e) {
143            return ResponseEntity.internalServerError().body(new ErrorDTO(e.getCodigo(), e.getMessage()));
144        } catch (ConstraintViolationException e) {
145            return ResponseEntity.badRequest().body(new ErrorDTO(Errores.ERROR_FALTAN_PARAMETROS_ENTRADA));
146        } catch (Exception e) {
147            return ResponseEntity.internalServerError().body(new ErrorDTO(Errores.ERROR_INTERNO));
148        }
149    }
```

Figura 49.- Endpoint "crearAnimal"

Resultado correcto

Vamos a comprobar que, efectivamente, el endpoint devuelva los resultados esperados en cada caso (código http 2XX).

Figura 50.- Test crear animal con éxito

Primero de todo debemos crear una petición http, después deberemos simular los servicios como crear animal para que cuando llegue un animal (en este caso cualquiera) el servicio devuelva un id concreto.

Una vez se han llevado a cabo las simulaciones necesarias de los servicios y el objeto a devolver, se procede a llamar al controlador con un animal de ejemplo.

Finalmente, se comprueba si en la respuesta se ha recibido el estado 201 para saber que se ha creado correctamente. Adicionalmente, comprobaremos que en la respuesta http venga en la cabecera "location" la URI correspondiente a dicho animal.

Resultado incorrecto

Para aquellos casos en los que se produce un error ya sea bien, porque se ha enviado parámetros insuficientes o incorrectos o bien, se haya producido un error interno en el servidor se mostrará al usuario una respuesta de error (códigos 4XX o 5XX http) en cuyo cuerpo aparecerá el objeto "ErrorDTO".

Figura 51.- ErrorDTO

El error DTO es una excepción personalizada en la que se devuelve un código de error junto con un mensaje, ambos personalizados. Para mantener la homogeneidad en la aplicación hemos creado un enumerado "Errores" que consiste en una serie de códigos de errores y mensajes para cada situación.

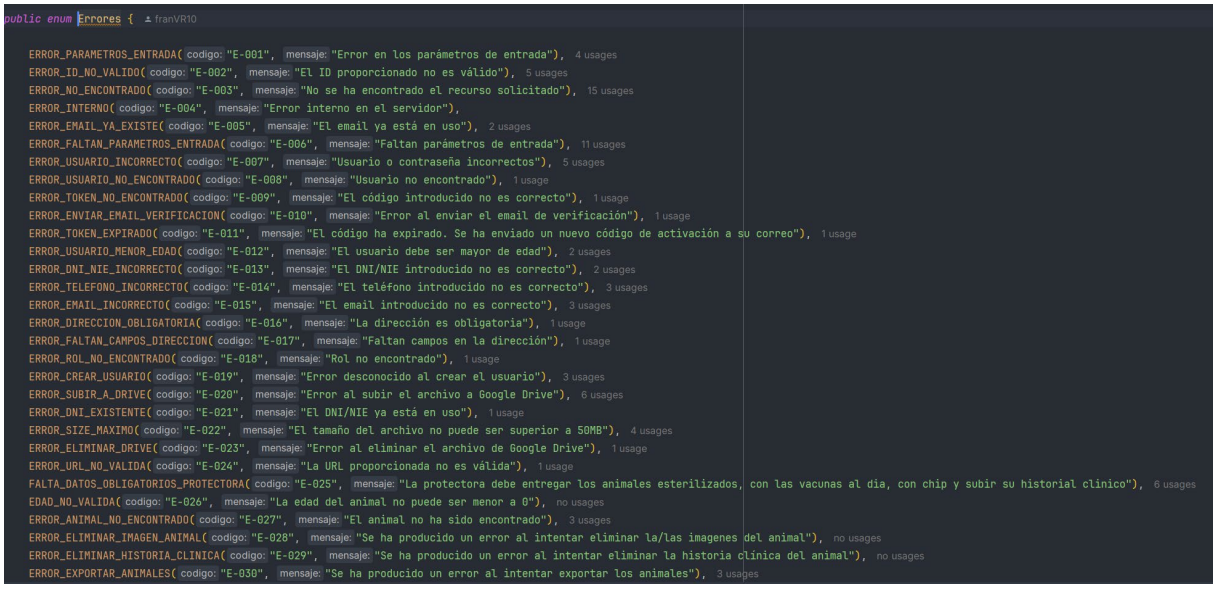


Figura 52.- Enumerado "Errores"

Parámetros de entrada incorrectos

Este error se produce debido a que el objeto a recibir por parte del controlador viene incorrecto o incompleto.

```java
@Test  franVR10 *
public void testCrearAnimalConstraintViolation() throws COREException {

    // Configura el mock para lanzar una excepción
    when(animalService.crearAnimal(any(Animal.class))).thenThrow(new ConstraintViolationException(null));

    // Llama al método del controlador
    ResponseEntity<?> response = animalController.crearAnimal(animal);

    // Verifica que la respuesta sea 500 Internal Server Error
    assertEquals(HttpStatus.BAD_REQUEST, response.getStatusCode());

    // Verifica que el cuerpo de la respuesta contenga el código de error
    ErrorDTO errorDTO = (ErrorDTO) response.getBody();
    assertEquals(Errores.ERROR_FALTAN_PARAMETROS_ENTRADA.getCodigo(), errorDTO.getCodigo());

}
```

Figura 53.- Error parámetros de entrada

Podemos observar que se devuelve una excepción "ConstraintViolationException", dicha excepción viene de las comprobaciones implementadas que se hacen sobre el objeto gracias a una librería del *framework* Spring Boot.

En este caso prepararemos el método para devolver dicha excepción cuando se llame al servicio de la creación del animal. Posteriormente, procederemos a llamar al controlador. Finalmente comprobaremos que el código de la respuesta http sea el de bad request (400) y que venga con el mensaje correspondiente.

Excepción controlada

Se produce cuando se da un fallo que está contemplado en la aplicación que pueda ocurrir.

```java
@Test  franVR10 *
public void testCrearAnimalCorexception() throws COREException {

    // Configura el mock para lanzar una excepción
    when(animalService.crearAnimal(any(Animal.class))).thenThrow(new COREException(Errores.ERROR_CREAR_ANIMAL));

    // Llama al método del controlador
    ResponseEntity<?> response = animalController.crearAnimal(animal);

    // Verifica que la respuesta sea 500 Internal Server Error
    assertEquals(HttpStatus.INTERNAL_SERVER_ERROR, response.getStatusCode());

    // Verifica que el cuerpo de la respuesta contenga el código de error
    ErrorDTO errorDTO = (ErrorDTO) response.getBody();
    assertEquals(Errores.ERROR_CREAR_ANIMAL.getCodigo(), errorDTO.getCodigo());

}
```

Figura 54.- Excepción controlada

Nuestras excepciones controladas son aquellas que se han capturado como "COREException" que es una excepción personalizada nuestra con código y error igual a "ErroresDTO" (Figura 51).

Estas llevarán un mensaje personalizado que se elevará hasta el usuario para poder advertirle del problema actual. En este caso, la excepción trata de un error al crear el animal y se incita al usuario a intentarlo más tarde.

Como en los casos anteriores simularemos el servicio, pero esta vez con la excepción personalizada correspondiente y llamaremos al controlador. Comprobaremos que el código http y la respuesta son las que deberían ser.

Excepción no controlada

```
@Test  franVR10
public void testCrearAnimalException() throws COREException {

    // Configura el mock para lanzar una excepción
    when(animalService.crearAnimal(any(Animal.class))).thenThrow(new RuntimeException());

    // Llama al método del controlador
    ResponseEntity<?> response = animalController.crearAnimal(animal);

    // Verifica que la respuesta sea 500 Internal Server Error
    assertEquals(HttpStatus.INTERNAL_SERVER_ERROR, response.getStatusCode());

    // Verifica que el cuerpo de la respuesta contenga el código de error
    ErrorDTO errorDTO = (ErrorDTO) response.getBody();
    assertEquals(Errores.ERROR_INTERNO.getCodigo(), errorDTO.getCodigo());

}
```

Figura 55.- Excepción no controlada

Simulamos una excepción en tiempo de ejecución al lanzar el servicio correspondiente. Llamamos al controlador correspondiente y, una vez más, comprobamos que las respuestas sean acordes a lo que esperamos. En este caso eso esperamos un mensaje de error desconocido ("Erorres.ERROR_INTERNO") ya que no se sabe qué es lo que ha podido pasar, es decir, no estaba controlado ese escenario.

Esta estructura de pruebas se repetirá para cada endpoint de la aplicación con los objetos que correspondan en cada caso. Tras hacer las pruebas para todos los casos de todos los endpoints hemos realizado un total de 135 tests.

```
[INFO] Results:
[INFO]
[INFO] Tests run: 135, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  9.276 s
[INFO] Finished at: 2024-09-13T20:13:00+02:00
[INFO] ------------------------------------------------------------------------

Process finished with exit code 0
```

Figura 56.- Resultados tests

5.2 Frontend

En el desarrollo del frontend de nuestra aplicación, hemos implementado pruebas unitarias utilizando Jasmine y Karma, dos herramientas clave para garantizar la calidad y el correcto funcionamiento de los componentes de Angular.

Jasmine es un framework de pruebas para JavaScript que permite escribir y ejecutar pruebas de forma sencilla y estructurada. Jasmine sigue un estilo de prueba llamado BDD (Behavior Driven Development), que se enfoca en describir el comportamiento de la aplicación en escenarios comprensibles. Las pruebas en Jasmine se organizan en "suites" (grupos de pruebas) y cada prueba, o "spec", verifica que una unidad de código cumpla con su comportamiento esperado. Con Jasmine, podemos crear afirmaciones (expect) que comparan los resultados obtenidos con los esperados.

Por otro lado, Karma es un test *runner* que se utiliza en combinación con Jasmine para ejecutar las pruebas en un entorno controlado. Karma se encarga de lanzar un servidor que ejecuta las pruebas en diferentes navegadores, asegurando que el código funcione correctamente en múltiples entornos. Esto nos permite obtener feedback inmediato al ejecutar las pruebas, detectar posibles errores y verificar el comportamiento en distintas plataformas.

En conjunto, Jasmine y Karma nos proporcionan una base sólida para asegurar que el código del frontend sea robusto y que las funcionalidades se comporten correctamente a medida que realizamos cambios y añadimos nuevas características. Las pruebas que hemos implementado

cubren tanto la lógica de los componentes como la interacción de los servicios, permitiendo una mayor confiabilidad y mantenibilidad del código.

Hemos realizado pruebas de todos los componentes sobre sus funcionalidades más relevantes ya que en este campo no tenemos tanta experiencia y, frontend, es un entorno que por su naturaleza se puede probar de forma manual en el propio uso de la aplicación.
Las pruebas siguen toda la misma estructura por lo que mostraremos un ejemplo y lo detallaremos. En este caso hemos elegido la vista *main*, que es aquella donde se muestran los filtros del animal y los animales disponibles en adopción.

```
it( expectation: 'debería cambiar el estado seleccionado de una especie al hacer clic', assertion: () : void => {
  component.isLoading = false;
  component.especies = [
    {id: 1, rutaIcono: 'random', nombre: 'Perros', seleccionado: false, rutaIconoSegura: 'path/to/icon' }
  ];
  fixture.detectChanges();

  const speciesElement : DebugElement = fixture.debugElement.query(By.css( selector: '.cardImagen'));
  speciesElement.triggerEventHandler( eventName: 'click', eventObj: null); // Simula un clic
  fixture.detectChanges(); // Actualiza la vista

  expect(component.especies[0].seleccionado).toBeTrue(); // Verifica que el estado cambió
});
```

Figura 57.- Ejemplo test

Observamos que para hacer esta prueba hemos preparado un par de cosas: se ha marcado que la aplicación ha terminado de cargar todos los campos y, que hay una especie disponible a elegir botón.

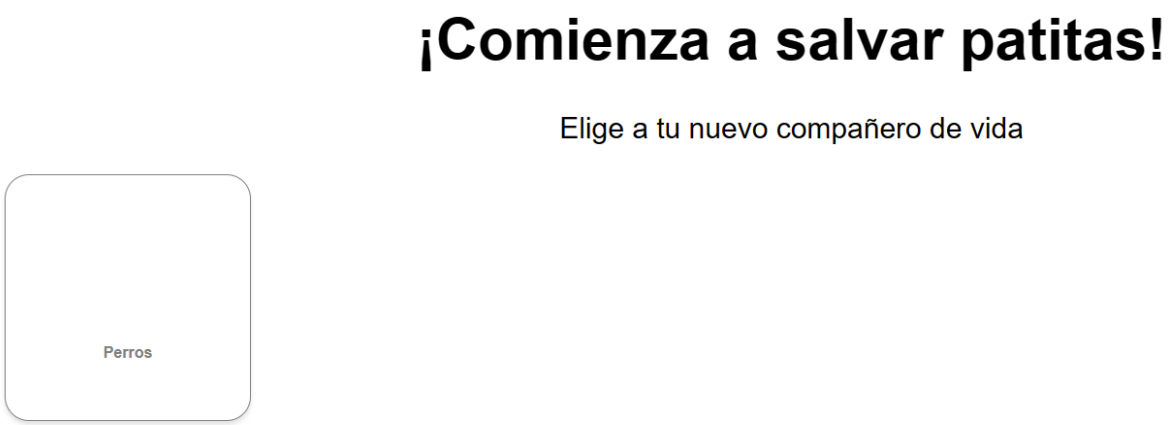


Figura 58.- Botón disponible

Posteriormente, se llama a una función que recoge los cambios producidos y actualiza la vista del navegador simulado por Karma. Tras recargar esta vista, simulamos un clic en el botón y observamos que realmente el componente está guardando la especie seleccionada.

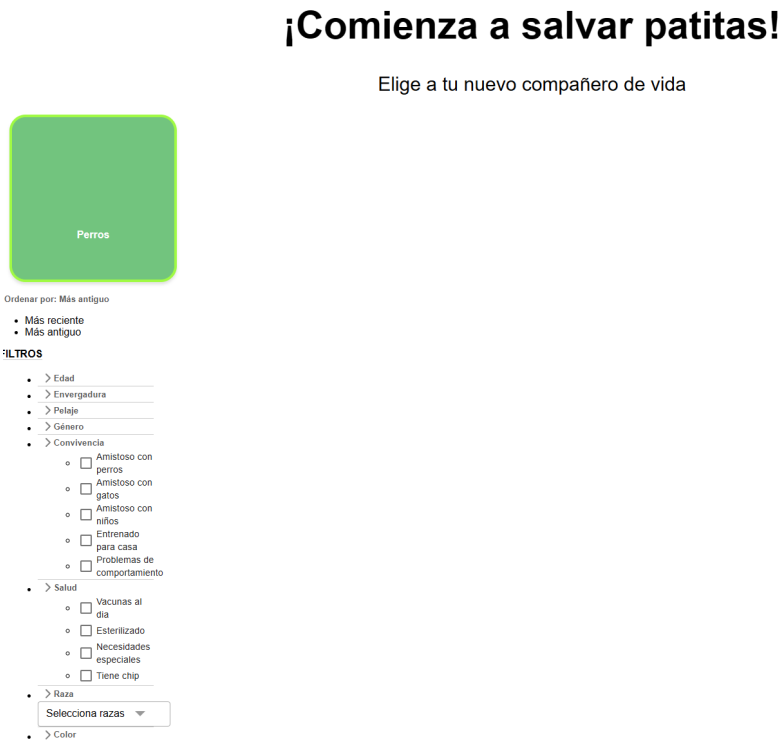


Figura 59.- Clic simulado

Esta estructura se va a repetir para todos los componentes y todo lo que se puede probar de cada uno como por ejemplo comprobar que aparezcan todos los filtros una vez pulsado el botón de la especie (Figura 59). Se realizó un total de 87 tests.

```
Chrome 128.0.0.0 (Windows 10): Executed 58 of 87 SUCCESS (0 secs / 0.472 secs)
LOG: 'Obtengo nuevos colores'
Chrome 128.0.0.0 (Windows 10): Executed 58 of 87 SUCCESS (0 secs / 0.472 secs)
LOG: 'Cambio en especies seleccionadas'
Chrome 128.0.0.0 (Windows 10): Executed 59 of 87 SUCCESS (0 secs / 0.48 secs)
LOG: 'Obtengo nuevos colores'
Chrome 128.0.0.0 (Windows 10): Executed 59 of 87 SUCCESS (0 secs / 0.48 secs)
LOG: 'Tipo de Usuario Cambiado:', 'NORMAL'
Chrome 128.0.0.0 (Windows 10): Executed 66 of 87 SUCCESS (0 secs / 0.68 secs)
LOG: 'Tipo de Usuario Cambiado:', 'PROTECTORA'
Chrome 128.0.0.0 (Windows 10): Executed 70 of 87 SUCCESS (0 secs / 0.721 secs)
Chrome 128.0.0.0 (Windows 10): Executed 80 of 87 SUCCESS (0 secs / 0.922 secs)
Chrome 128.0.0.0 (Windows 10): Executed 80 of 87 SUCCESS (0 secs / 0.922 secs)
Chrome 128.0.0.0 (Windows 10): Executed 81 of 87 SUCCESS (0 secs / 0.923 secs)
13 09 2024 21:48:28.869:WARN [web-server]: 404: /_karma_webpack_/imagen2.jpg
LOG: 'Participantes:', [Object{id: 1, nombre: 'John Doe', rutaImagen: 'path/to/image'}]
Chrome 128.0.0.0 (Windows 10): Executed 85 of 87 SUCCESS (0 secs / 0.928 secs)
Chrome 128.0.0.0 (Windows 10): Executed 86 of 87 SUCCESS (0 secs / 0.93 secs)
Chrome 128.0.0.0 (Windows 10): Executed 87 of 87 SUCCESS (1.042 secs / 0.931 secs)
TOTAL: 87 SUCCESS
```

Figura 60.- Tests frontend

6

Conclusiones

6.1 Objetivos alcanzados

El objetivo principal de este proyecto es desarrollar una alternativa competitiva a las actuales aplicaciones web de adopción de animales en España. Hemos identificado una carencia significativa en cuanto a la calidad de estas plataformas, las cuales a menudo no están a la altura de las expectativas de los usuarios. Actualmente, las redes sociales, a pesar de no estar diseñadas específicamente para este propósito, suelen ser la opción preferida debido a su accesibilidad y facilidad de uso, lo que afecta negativamente a la experiencia del usuario en términos de eficacia y especialización.

Aunque comenzamos sin experiencia previa en diseño o desarrollo *frontend*, creemos que hemos conseguido crear una aplicación web segura y robusta que puede competir con las alternativas existentes. Nuestra propuesta ofrece funcionalidades que otras plataformas no incluyen, y lo hace manteniendo una interfaz simple, moderna y amigable, que busca atraer a todo tipo de usuarios. Este enfoque intuitivo es precisamente lo que da a las redes sociales su ventaja, y es un aspecto que hemos intentado replicar en nuestra plataforma, adaptándola específicamente a las necesidades del proceso de adopción de animales.

Queremos destacar que nuestra aplicación ofrece una amplia gama de opciones, asegurando que ningún usuario sienta que le falta la capacidad de reflejar de manera realista el estado de su animal o que carece de herramientas adecuadas para buscar un animal de forma efectiva. Todo esto se ha logrado manteniendo un diseño sencillo y limpio, evitando la sobrecarga de información y garantizando una experiencia de usuario agradable y accesible, como hemos mencionado anteriormente.

6.2 Aspectos a mejorar

A continuación, vamos a detallar los principales aspectos que consideramos que deberíamos mejorar.

6.2.1 Código

Al ser nuestro primer proyecto en Angular, dedicamos una cantidad considerable de tiempo a estudiar y comprender este *framework*, lo que impactó en el cumplimiento del plazo estimado. Como resultado, no pudimos implementar todas las buenas prácticas ni reutilizar tantos componentes como hubiéramos deseado, a pesar de que la reutilización es una de las principales ventajas de Angular. Además, creemos que el código de los componentes en TypeScript tiene un amplio margen de mejora, ya que también es la primera vez que lo utilizamos. Mejorar estos aspectos en futuras iteraciones incrementaría significativamente la eficiencia y escalabilidad del proyecto, algo que consideramos fundamental.

En cuanto a la parte de Java, aunque la arquitectura está mejor estructurada y se han seguido buenas prácticas, reconocemos que todavía hay margen de mejora. Por ejemplo, hemos identificado nombres de variables y métodos que no son lo suficientemente descriptivos, además de inconsistencias en la nomenclatura, mezclando inglés y español en varias partes del código. Asimismo, hemos optado por el uso de *converters* como alternativa a los DTOs. Si bien esta solución permite desacoplar los objetos obtenidos de la base de datos de los que se devuelven al *frontend*, al modificar el formato de los campos directamente en lugar de mapear clases, el resultado es un código menos legible en comparación con la implementación de DTOs y *mappers*. Consideramos que el uso de DTOs y *mappers* hubiera mejorado la claridad y mantenibilidad del código.

6.2.2 Futuras funcionalidades a estudiar

Detallaremos a continuación las funcionalidades que hemos identificado como interesantes, pero quedaban fuera del ámbito de este TFG y que se podrían implementar en el futuro.

Firma electrónica de documentos

Para agilizar y dotar de mayor seguridad al proceso de confirmación de la adopción se añadirá una firma automática a través de un botón, esta irá asociada a algún dato único del usuario como podría ser el id, DNI, UUID, etc.

Lista de animales favoritos

Consideramos que es una funcionalidad esencial para facilitar la decisión del usuario en el momento de la adopción y poder retomarla en cualquier momento. Se añadirá una opción de

"Animales favoritos" al panel desplegable de usuario para poder consultarlos en cualquier momento.

Compartición de ficheros multimedia a través del chat en tiempo real

Añadir la posibilidad de poder enviar directamente a través del chat de la aplicación archivos multimedia como fotos y videos para evitar tener que recurrir a aplicaciones externas de mensajería.

Encriptación para el chat en tiempo real

Actualmente los mensajes se guardan en claro en la base de datos, lo que supone un potencial riesgo en caso de acceso no autorizado y una violación de los derechos de privacidad del usuario.

Opción de recuperar acceso

Agregar la indispensable posibilidad de poder cambiar la contraseña de forma segura en el caso de olvidar la actual.

Edición de perfil de usuario

Permitir editar los datos a través del perfil de usuario, así como la imagen de perfil y poder añadir distintas redes sociales.

Mayor automatización del proceso adoptivo

Con el objetivo de reducir la cantidad de operaciones por parte de los usuarios y reducir el número de correos electrónicos, se quiere introducir un sistema que reconozca automáticamente los cambios en el proceso y se muestre a través de notificaciones en la página.

Contadores de notificaciones de mensajes

Consiste en añadir al panel de usuario un número que indique el número de mensajes nuevos recibidos.

Contadores en los filtros

Por cada apartado del filtro queremos mostrar un número que indique la cantidad de resultados que coinciden para cada apartado del filtro en ese momento.

Implementación de Amazon S3

Actualmente el sistema que guarda y muestra los documentos y las imágenes subidas es la API gratuita de Google con el plan gratuito de Google Drive en el que se ofrecen 15 GB de almacenamiento gratuito. Este sistema, aunque funciona, no está realmente pensado para el mundo laboral y el rendimiento no es el esperado para una aplicación real.

Sistema de copias de seguridad periódicas

Un sistema de copias de seguridad periódicas es esencial para garantizar la integridad y disponibilidad de la base de datos en caso de fallos, desastres o ataques de seguridad. Estas copias permiten restaurar la base de datos a un estado reciente, minimizando la pérdida de datos y asegurando la continuidad del servicio. Al automatizarse, las copias se realizan regularmente sin intervención manual, y su almacenamiento tanto local como remoto ofrece una protección adicional frente a situaciones imprevistas que puedan comprometer la información.

Sistema de conflictos

Implementación de un sistema para facilitar al usuario ponerse en contacto con un administrador de la aplicación en caso de disputa o fraude.

Sistema de autenticación multifactor (MFA)

Un sistema de autenticación multifactor (MFA) añade una capa adicional de seguridad al requerir más de una forma de verificación para acceder a la aplicación. Esto significa que, además de la contraseña, se solicita al usuario un segundo factor, como un código enviado al teléfono o una aplicación de autenticación. Al combinar algo que el usuario sabe (la contraseña) con algo que tiene (un dispositivo de confianza), se reduce significativamente el riesgo de accesos no autorizados, incluso si la contraseña es comprometida. Este enfoque fortalece la protección de la cuenta frente a ataques y fraudes.

Mejoras en la accesibilidad

Adición de un modo nocturno, soporte para lectores de pantalla, navegación por teclado, tipografía para disléxicos, colores para daltónicos, etc.

Diseño reactivo

Realizar cambios en el diseño para que pueda ser visualizado de manera correcta desde cualquier tipo de dispositivo sin importar el tamaño de su pantalla.

6.3 Plan de futuro

El número de familias en España que optan por añadir un animal de compañía a su hogar está en constante aumento [3], impulsado en parte por una mayor concienciación sobre la adopción frente a la compra de mascotas. En respuesta a esta creciente demanda, creemos que se necesita una solución segura y moderna para la adopción de animales.

Nuestro objetivo es llevar esta aplicación al mundo real y hacerla accesible para todas las familias interesadas en adoptar. Para lograr esto, buscamos asociarnos con instituciones influyentes en el ámbito del bienestar animal, como la Junta de Andalucía, que podrían beneficiarse de nuestra plataforma y apoyar su implementación.

Estamos particularmente interesados en recibir orientación y mentoría de profesionales de estas instituciones. Su experiencia y apoyo serían cruciales para perfeccionar la aplicación y asegurar que cumpla con los estándares legales, éticos y tecnológicos necesarios, promoviendo así un avance significativo en el desarrollo y concienciación del bienestar animal en nuestro país.

Referencias

[1] JSON Web Tokens (s.f). Recuperado de: https://jwt.io/.

[2] Ali, B. (s.f) *Youtube* [Canal Bouali Ali]. Recuperado de: https://www.youtube.com/watch?v=BVdQ3iuovg0.

[3] Redacción de Cuatro. (2024, marzo 14). Crecen el número de mascotas y los gastos en España: más perros que niños. *Cuatro.com*. Recuperado de: https://www.cuatro.com/noticias/sociedad/20240314/crecen-numero-mascotas-gastos-espana-mas-perros-ninos_18_011965285.html.

[4] SVGREPO. (s.f). Recuperado de: https://www.svgrepo.com/.

[5] Fontawesome. (s.f). Recuperado de: https://fontawesome.com/search.

[6] Bootstrapdocs. (s.f). Recuperado de: https://getbootstrap.com/docs/5.3/getting-started/introduction/.

[7] Spring boot docs. (s.f). Recuperado de: https://docs.spring.io/spring-boot/index.html.

[8] Angular docs. (s.f). Recuperado de: https://v17.angular.io/docs.

[9] The Dog API. (s.f). Recuperado de: https://www.thedogapi.com/.

[10] The Cat API. (s.f). Recuperado de: https://thecatapi.com/.

Apéndice A

Manual de usuario

A.1 Requisitos técnicos mínimos

El proyecto consiste una aplicación web cuya intención es estar pública en la red por lo que los requisitos consistirán únicamente en disponer de un dispositivo que cuente con conexión a internet y un navegador web instalado.

A.2 Acceso y activación de cuenta

A.2.1 Registro

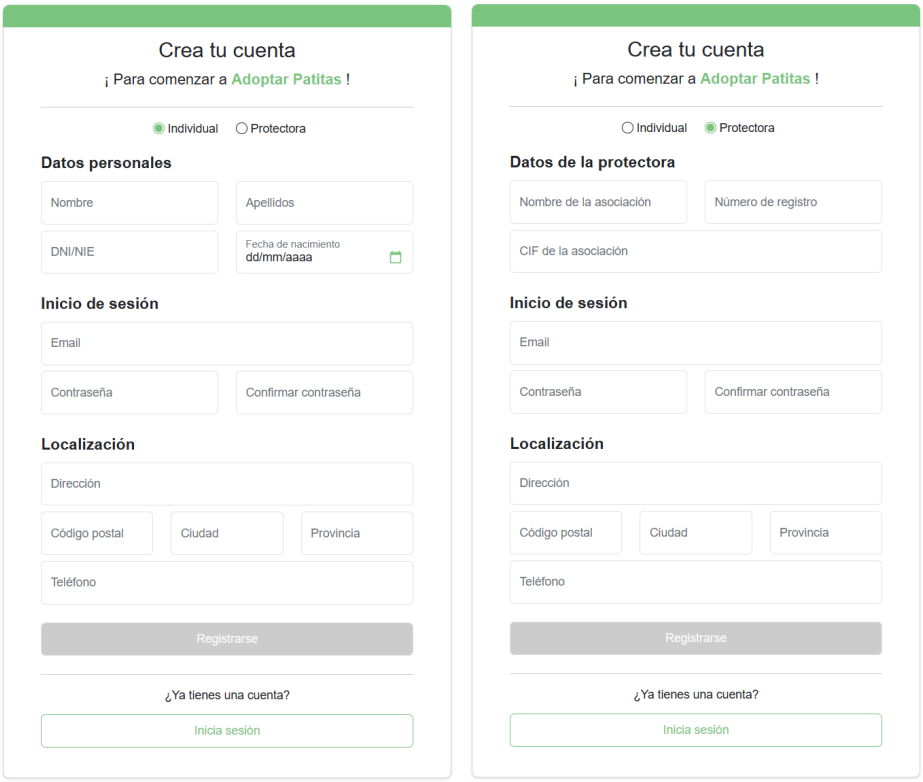


Figura 61.- Registro

El usuario deberá rellenar todos los datos requeridos en función del tipo de usuario elegido, que puede ser individual o protectora.

Todos los campos serán obligatorios, si alguno no se rellena o se hace de forma incorrecta como introducir un DNI incorrecto, esto se notificará en tiempo real al usuario. El botón de registro no se activará hasta que todos los campos estén rellenos y correctos.

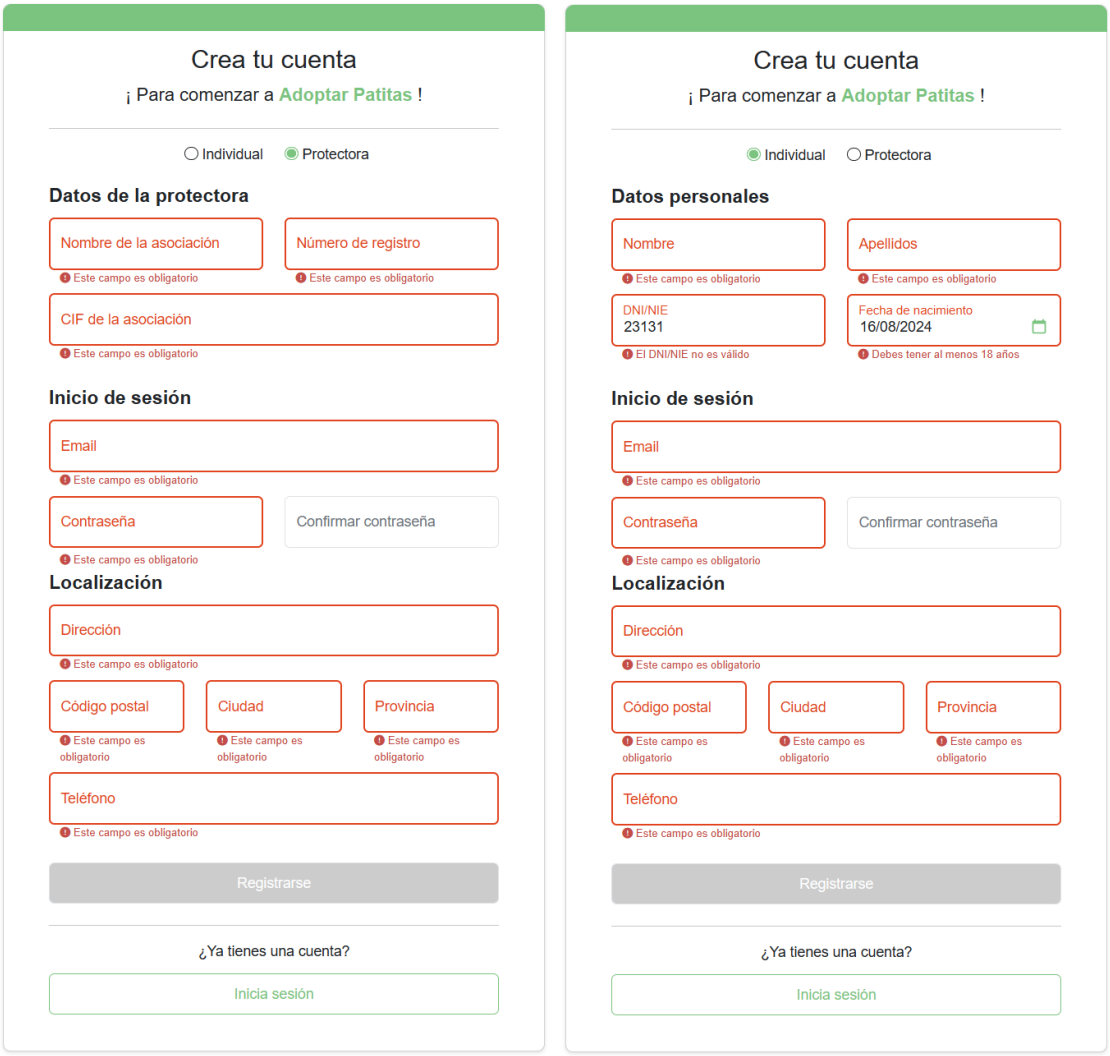


Figura 62.- Comprobaciones registro

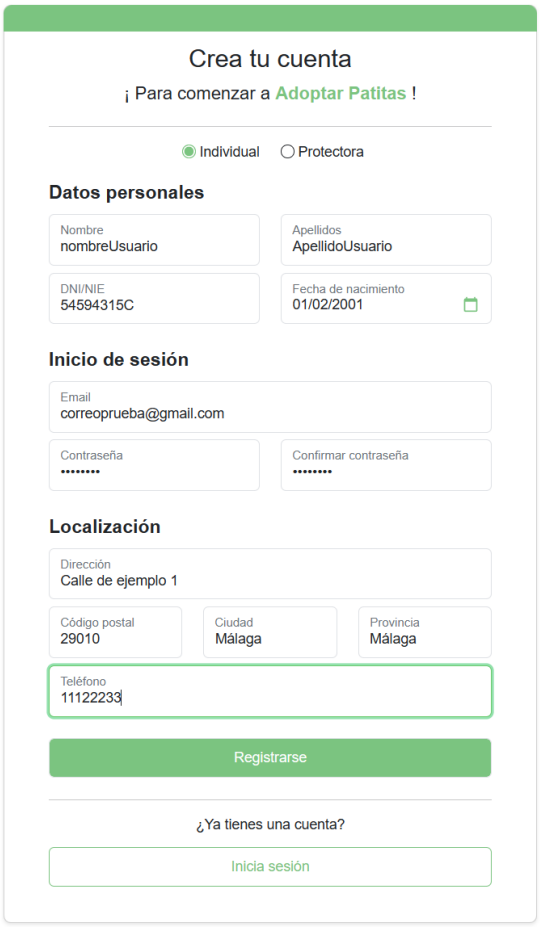


Figura 63.- Registro correcto

A.2.2 Activar cuenta

Una vez pulsado el botón registrarse se mostrará una vista donde activar la cuenta introduciendo el código de 6 dígitos que se enviará a la dirección de correo indicada.

Figura 64.- Correo de confirmación

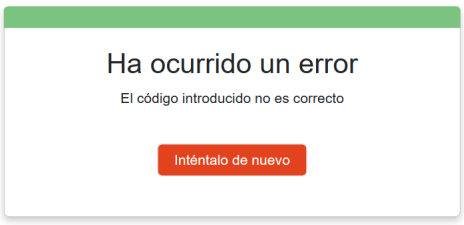


Figura 65.- Código incorrecto

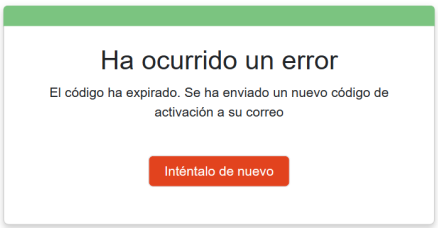


Figura 66.- Código expirado

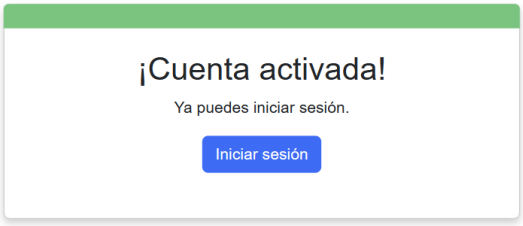


Figura 67.- Cuenta activada con éxito

En caso de que se introduzca un código incorrecto, se informará como tal. Si el código era correcto, pero ha expirado, se enviará un nuevo correo con un código actualizado. Una vez introducido el código correcto y no expirado la cuenta será activada.

A.2.3 Inicio de sesión

Para iniciar sesión únicamente se requerirá el correo y la contraseña empleados al crear la cuenta.

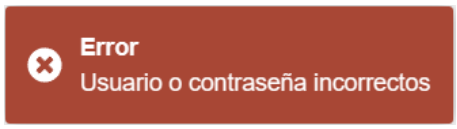


Figura 68.- Toast rojo en caso de error

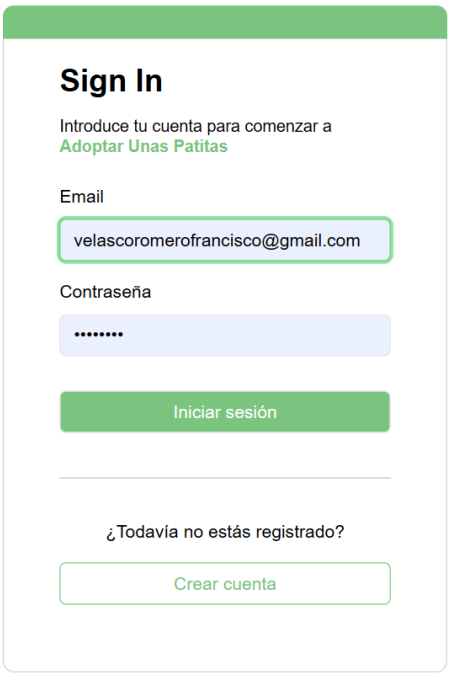


Figura 69.- Inicio de sesión

A.3 Página principal

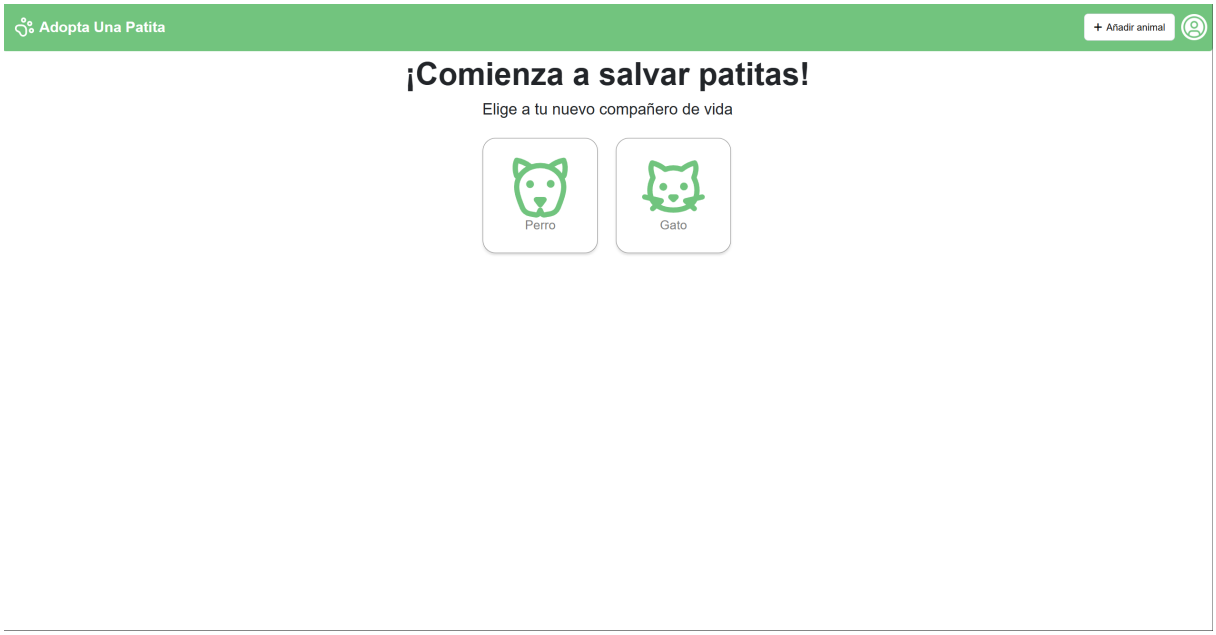


Figura 70.- Página principal inicial

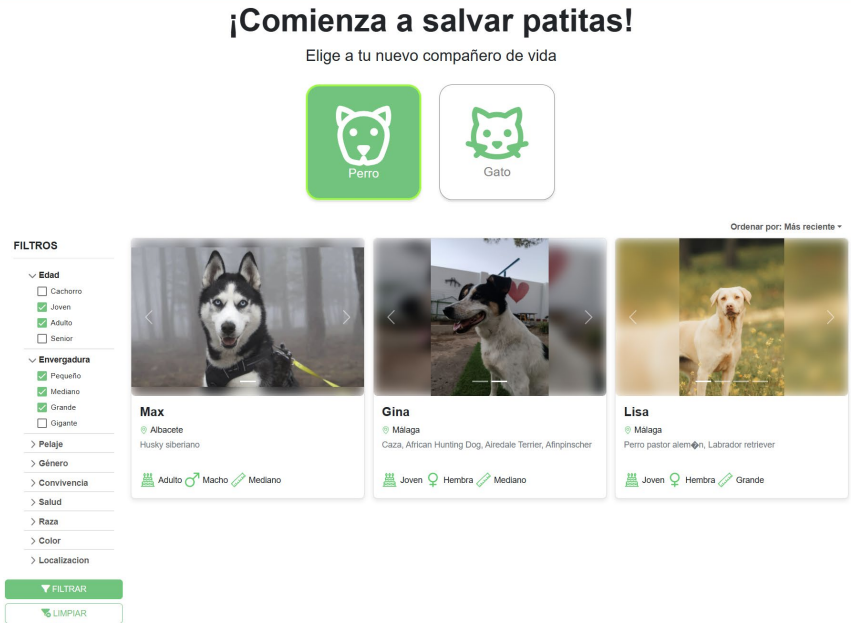


Figura 71.- Búsqueda por filtros

El usuario seleccionará los filtros que le interesen a través de la serie de desplegables proporcionados. Para que estos filtros aparezcan primero debe seleccionar alguna especie

de las ofrecidas. Se podrá buscar por más de una especie a la vez.

Pulsar el botón limpiar o cambiar la especie seleccionada causará que los filtros vuelvan a su estado inicial. Los filtros variarán según las especies seleccionadas, por ejemplo, como no existen gatos gigantes pues esta opción estará disponible para filtrar.

Figura 72.- No existen gatos gigantes

A.4 Animal

A.4.1 Perfil

Hacer clic en cualquiera de las tarjetas de la página principal llevará al perfil del animal donde podremos consultar todos sus datos.

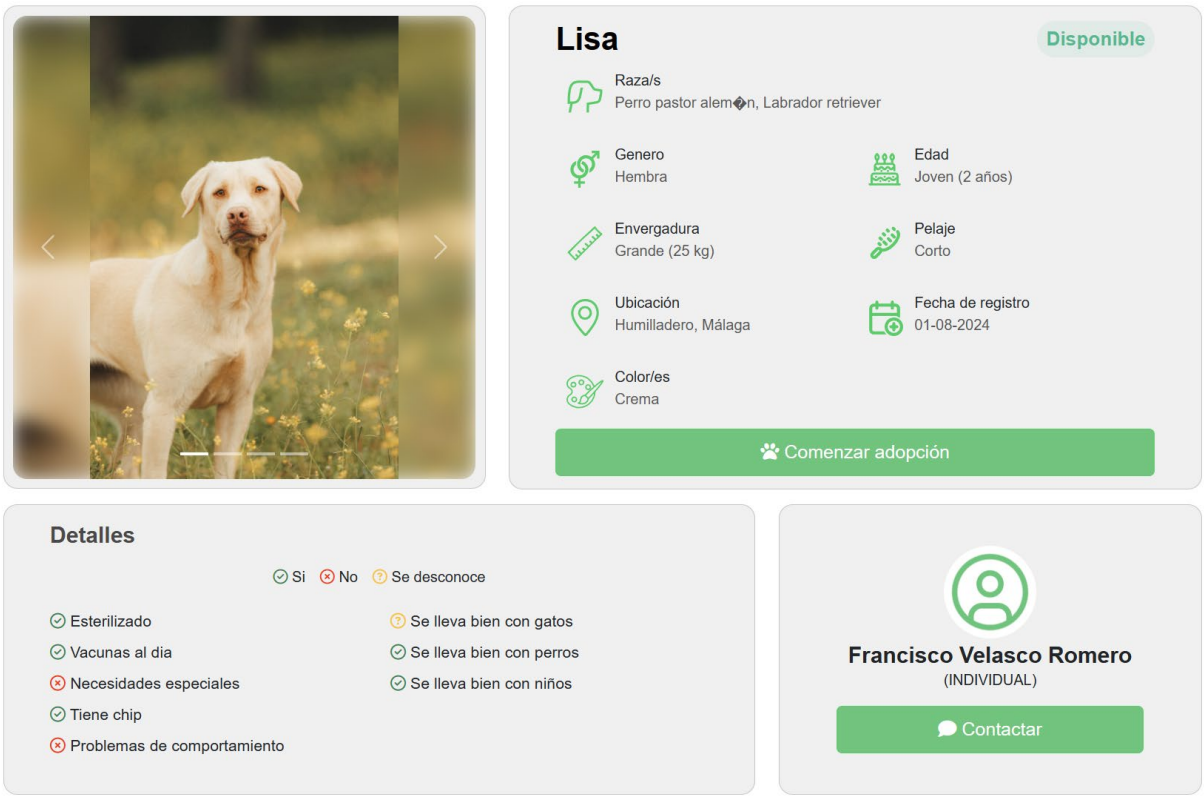


Figura 73.- Perfil (Parte 1)

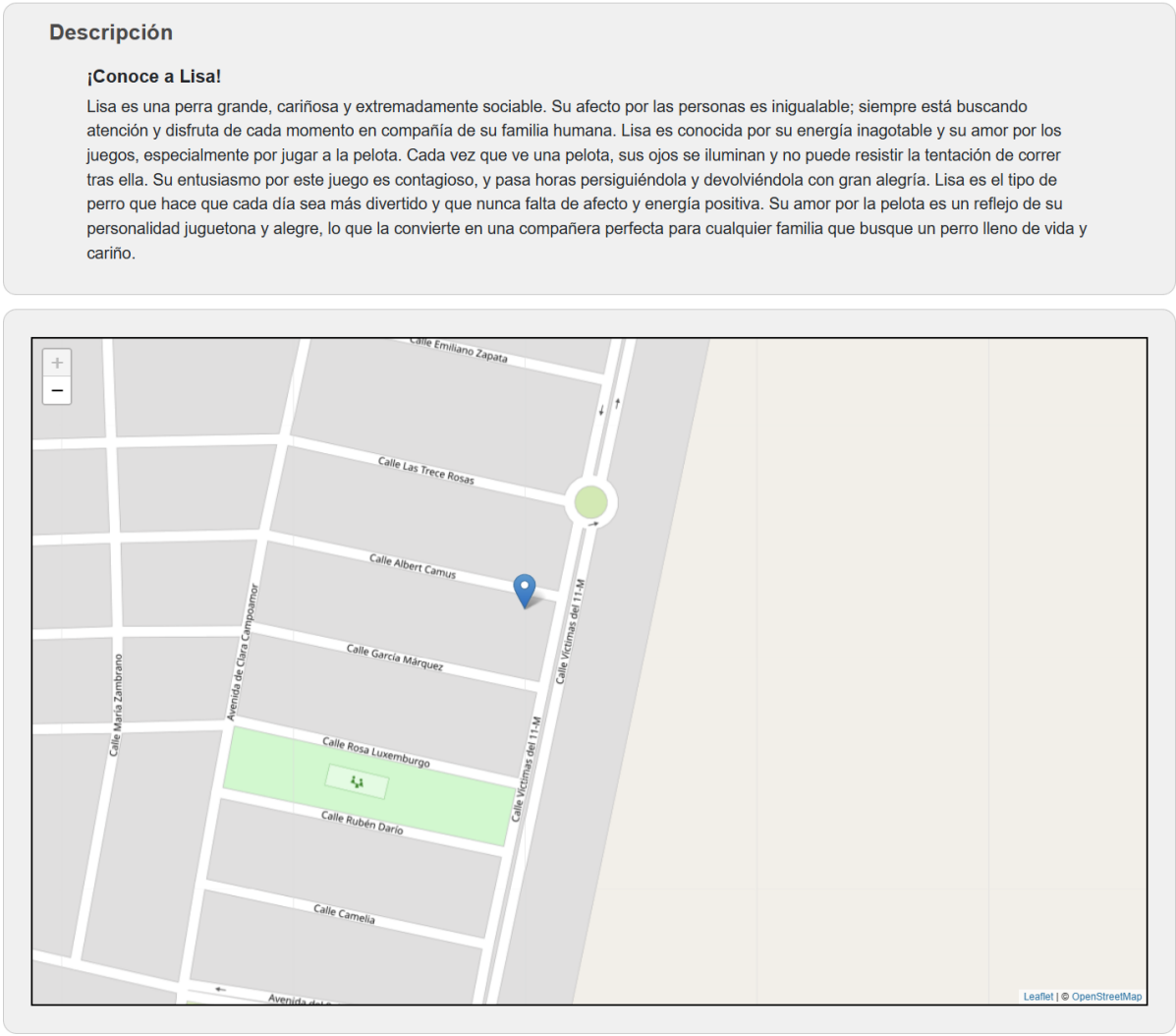


Figura 74.- Perfil (Parte 2)

En la primera parte podemos ver que está dividido en varios módulos: carrusel de fotos del animal, características, datos de convivencia y usuario propietario.

Hacer clic en el icono del usuario nos llevará a su perfil. Clicar en el botón contactar nos llevará al chat con esa persona.

En la segunda parte podemos ver que se muestra la descripción y, justo debajo, encontramos el módulo donde se muestra la ubicación del animal a través de leaflet.

Si somos el propietario del animal se mostrarán dos botones al lado de su nombre: editar y eliminar.

Figura 75.- Opciones animal propio

A.4.2 Crear animal

El animal podrá ser creado pulsando el botón "añadir animal" que se encuentra en la cabecera de la página. Si el usuario no ha iniciado sesión se le llevará al inicio de sesión.

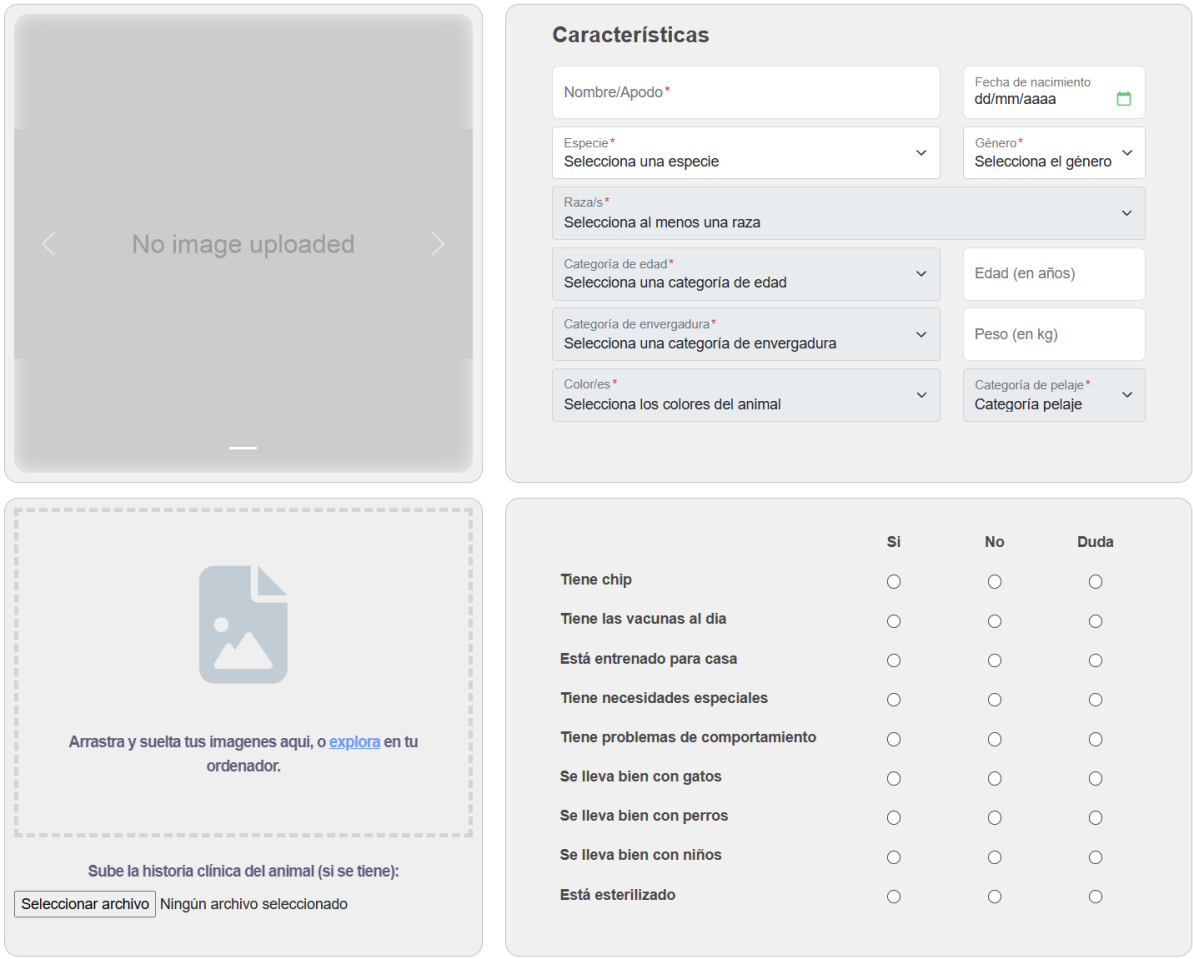


Figura 76.- Crear animal (Parte 1)

Mostrará todos los campos a rellenar. Todos los campos son obligatorios a excepción de la historia clínica (sino es una protectora) y aquellos campos de características que no tengan el asterisco rojo.

Inicialmente podemos ver que hay varios campos deshabilitados, esto se debe a que su contenido va a variar una vez se elija la especie por lo que, hasta no elegir la especie no estarán disponibles.

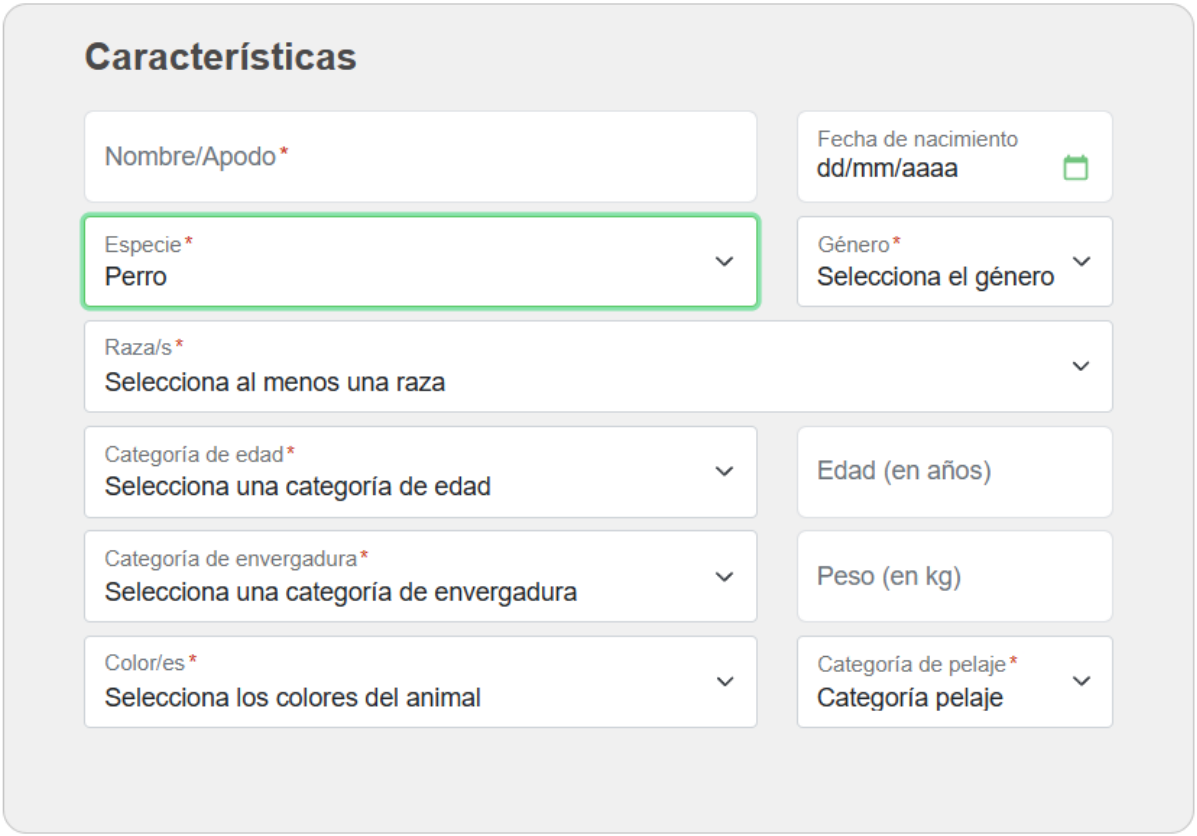


Figura 77.- Especie seleccionada

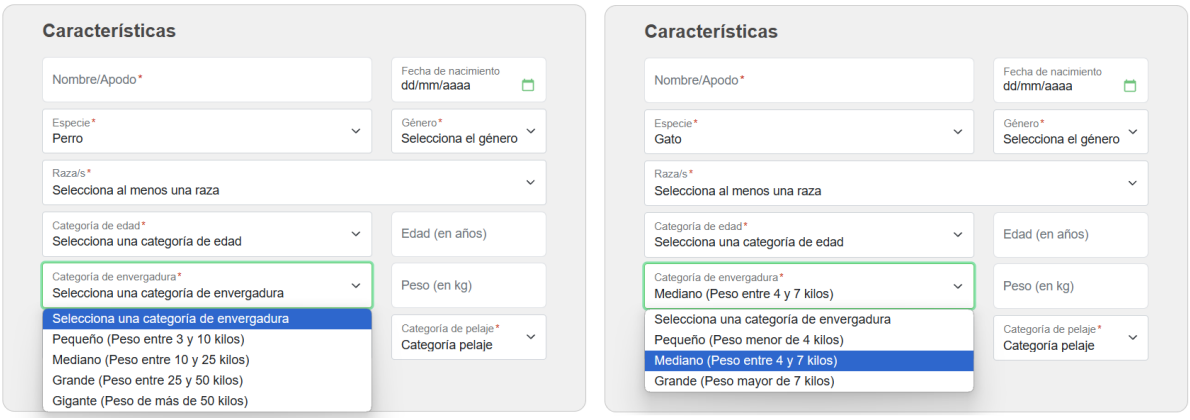


Figura 78.- Cambian las características según la especie

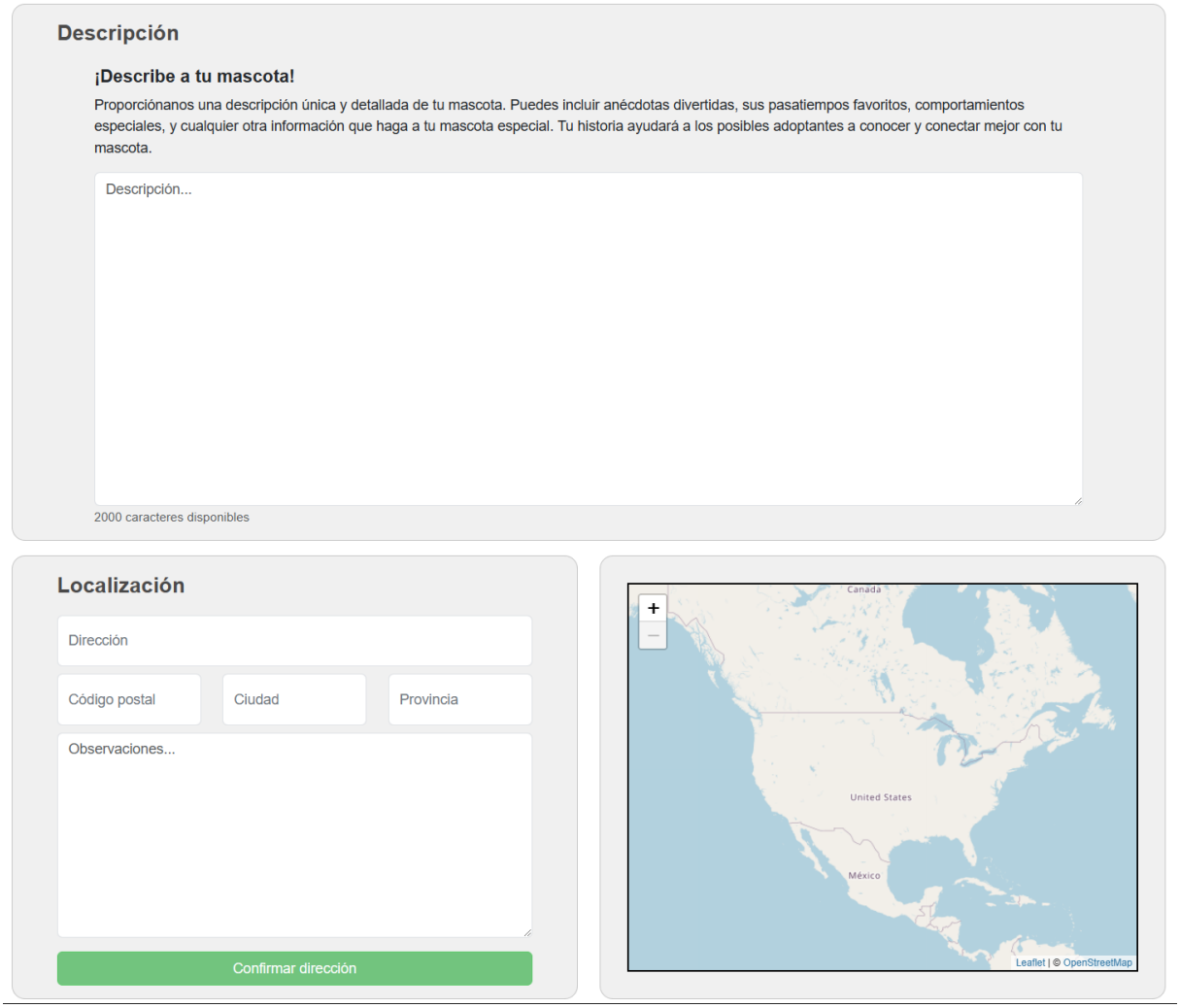


Figura 79.- Crear animal (Parte 2)

Se requerirá una descripción con una longitud no mayor a 2000 caracteres. En todo momento se visualizarán los caracteres disponibles.

Finalmente se ofrece un módulo donde introducir la información sobre la dirección y un botón para confirmarla, la cual, actualizará la ubicación del mapa.

Todos los campos incorrectos se mostrarán al pulsar en guardar o si son obligatorios, también se podrá ver al tiempo real al entrar en alguno y no rellenarlo.

Se ofrecerá al final botones de cancelar y guardar. Guardar llevará al perfil creado además de un mensaje de confirmación.

A.4.3 Editar animal

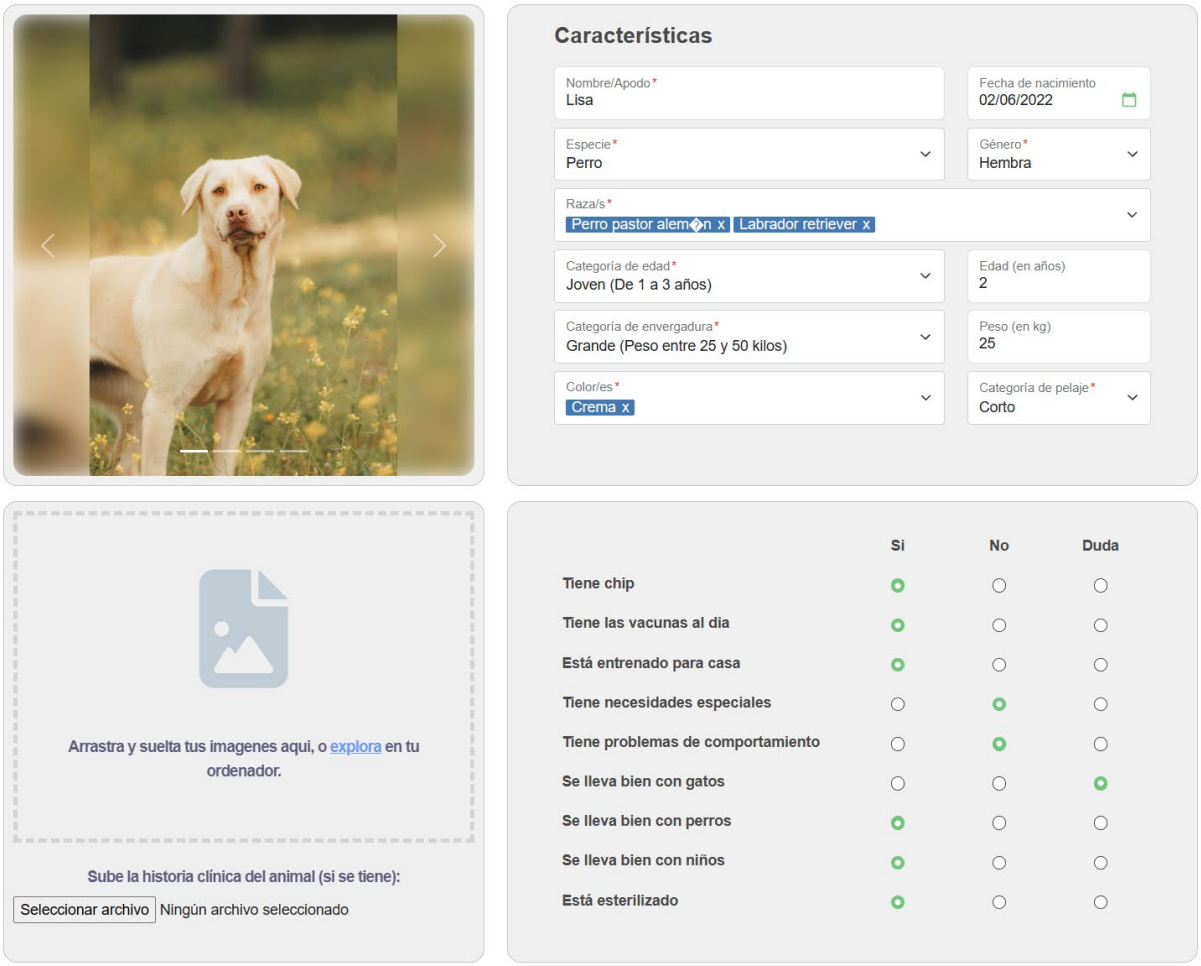


Figura 80.- Edición animal (Parte 1)

Podemos observar que se muestran las opciones disponibles a modificar con los datos que aparecían en su perfil.

Del módulo características solo será obligatorio rellenar o elegir datos del desplegable en el caso que tenga el asterisco rojo. En los módulos que se encuentran en la segunda mitad de la imagen, todo será obligatorio excepto la historia clínica, que será únicamente obligatorio en el caso que el propietario sea una protectora.

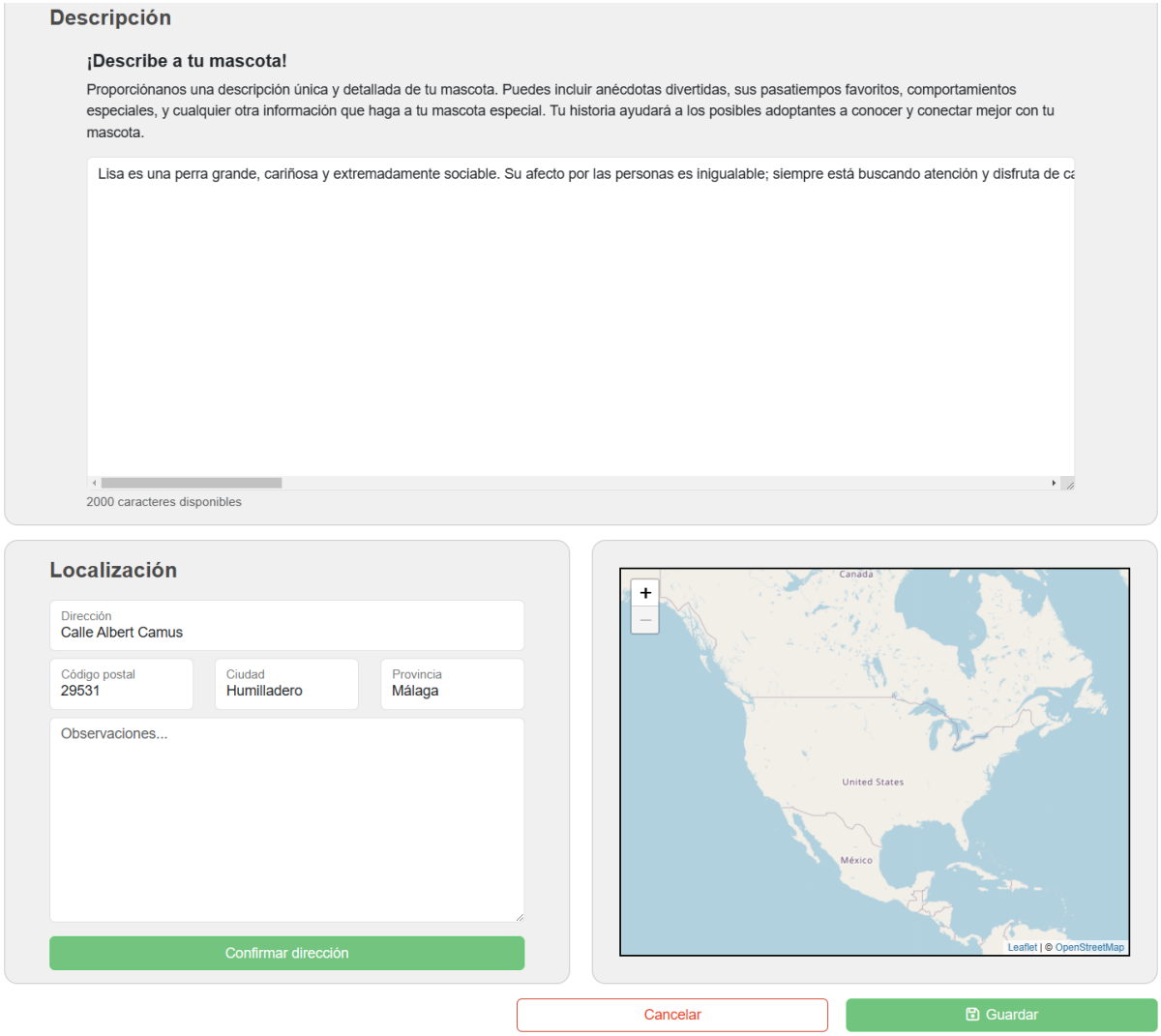


Figura 81.- Editar animal (Parte 2)

En esta segunda parte encontramos dos módulos, descripción y localización. La descripción será obligatoria y no podrá tener un contenido de más de 2000 caracteres. Se ofrecerá un contador para ver de manera visual cuantos caracteres quedan disponibles.

En la localización, se muestran dos cajas, a la izquierda se introducirá la dirección y, opcionalmente una observación asociada a esta. Una vez introducida, se deberá pulsar el botón confirmar dirección, el cuál actualizará el mapa a la derecha.

Los errores o campos no rellenos se indicarán al usuario bien cuando pulse el botón guardar o bien cuando haya estado dentro de algún campo obligatorio y no haya sido rellenado. Se marcará en rojo aquellos campos incorrectos o vacíos (siendo obligatorios).

Finalmente, se mostrarán las opciones de confirmar y cancelar.

A.4.4 Eliminar animal

Para eliminar el animal simplemente se preguntará si se está seguro de la opción elegida. Si se confirma el botón, el animal será dado de baja de la aplicación. Seguidamente, se llevará al usuario de vuelta a la página principal.

Figura 82.- Eliminar perfil animal

A.5 Usuario

A.5.1 Panel de usuario

En la cabecera de la página, si tenemos la sesión iniciada encontramos a la derecha del todo la foto de perfil del usuario de la sesión actual. Si pinchamos sobre ella obtendremos un menú desplegable donde tendremos varios accesos directos.

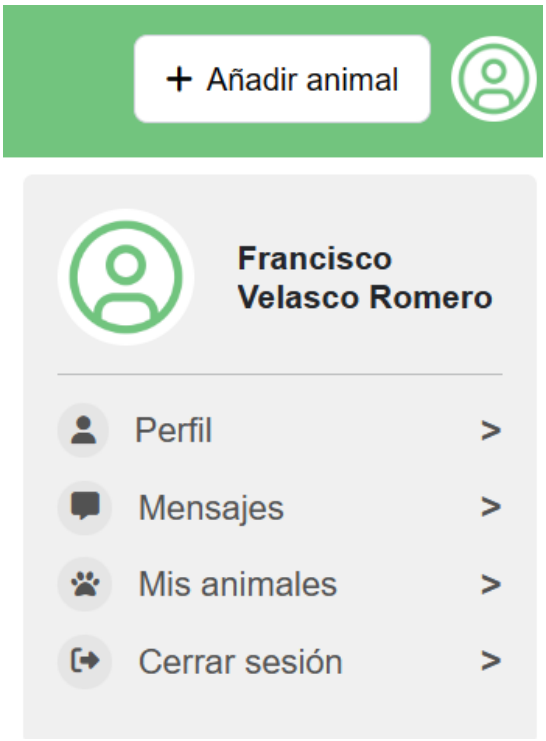


Figura 83.- Panel de usuario

A.5.2 Perfil

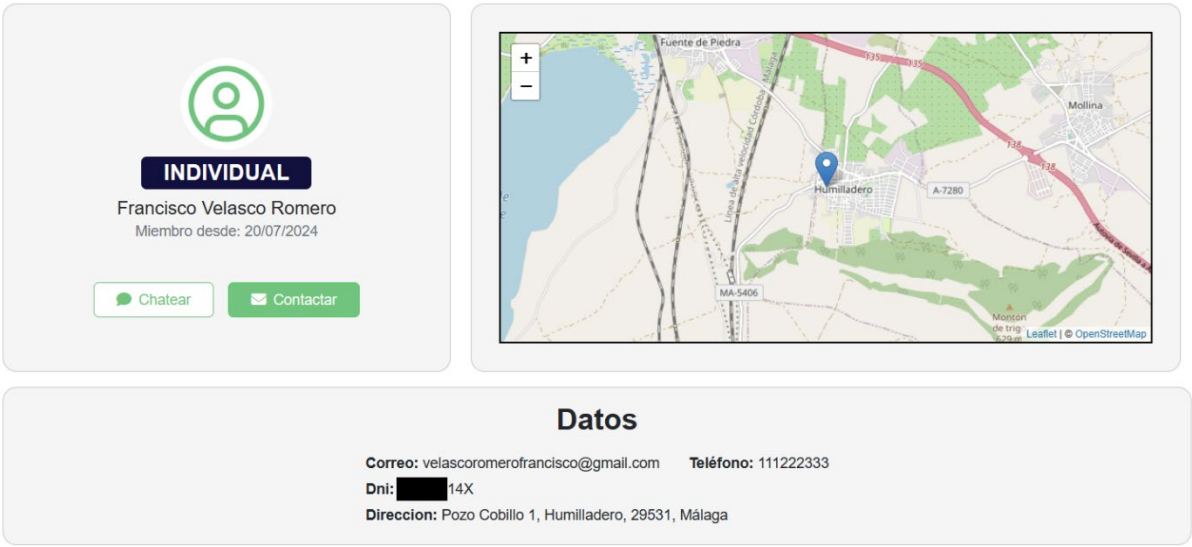


Figura 84.- Perfil usuario

Podemos ver datos básicos del usuario al que accedemos y su foto de perfil. Los datos sensibles como el DNI solo podrán ser observados por el propietario de la cuenta.

Se ofrece la posibilidad de poder contactar con el usuario vía chat o bien vía correo a través de los dos botones que se ofrecen.

También se proporciona un mapa con la ubicación del usuario.

A.5.3 Mensajes

Podemos ver un listado de usuarios con los que hemos abierto alguna vez un chat.

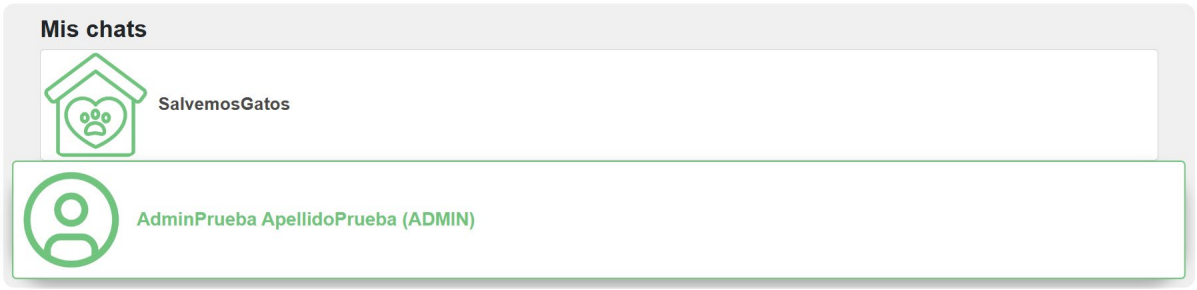


Figura 85.- Listado de chats

Pinchar en cualquiera de ellos nos llevará al chat correspondiente con dicha persona.

A.5.4 Mis animales

En este apartado podremos observar todos los animales que tenemos dados en adopción en la página.

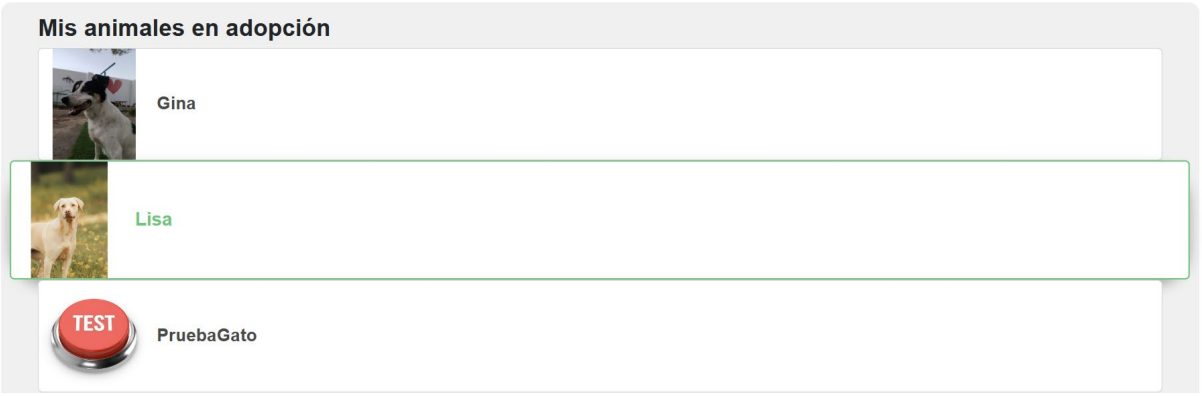


Figura 86.- Mis animales

Pinchar en cualquiera de ellos nos llevaría al perfil de dicho animal.

A.6 Chat

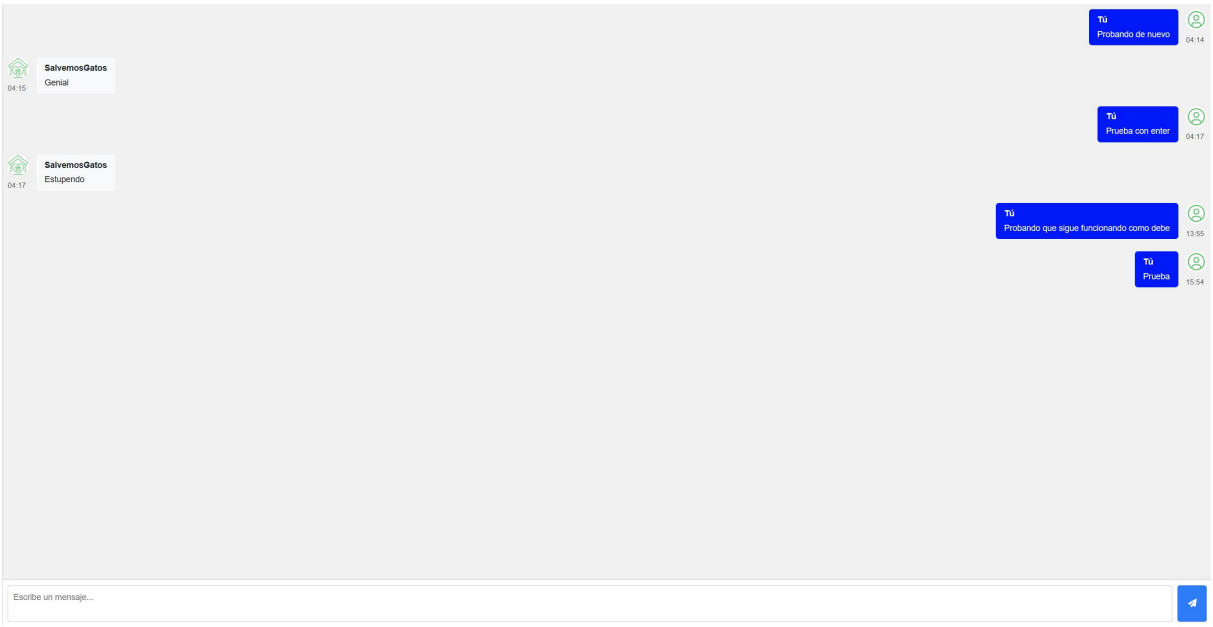


Figura 87.- Chat

Llegar al chat a través de cualquiera de las acciones proporcionadas en la página llevará a una vista en una nueva ventada donde podremos enviar mensajes con el otro usuario a tiempo real.

Nuestro usuario se encuentra en la parte de la derecha marcado en azul, mientras que el usuario con el que nos comunicamos será marcado en blanco en el lado izquierdo.

A.7 Proceso de adopción

Vamos a detallar cómo sería el proceso de adopción por parte del usuario.

Desde el perfil del animal se pulsará el botón de "comenzar adopción".

Figura 88.- Comenzar adopción

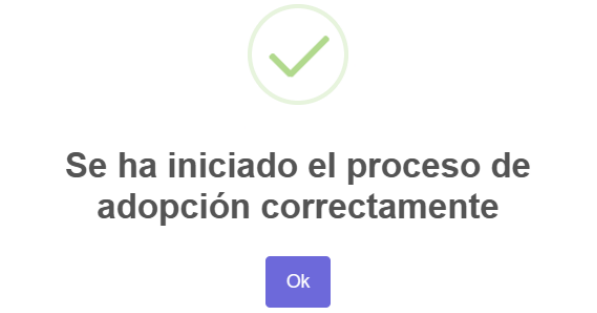


Figura 89.- Confirmación

El propietario recibirá los datos de la adopción y decidirá si aceptarla o rechazarla. En caso de ser rechazada recibiremos el siguiente correo:

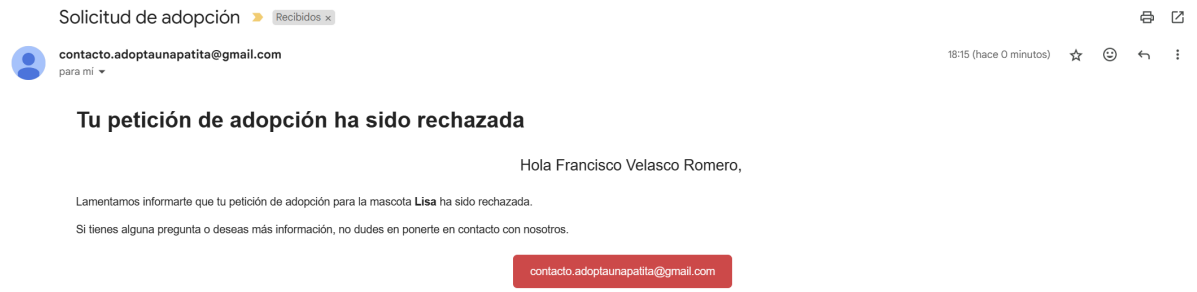


Figura 90.- Petición rechazada

En caso de ser aceptada recibiremos el mensaje de la figura 54.

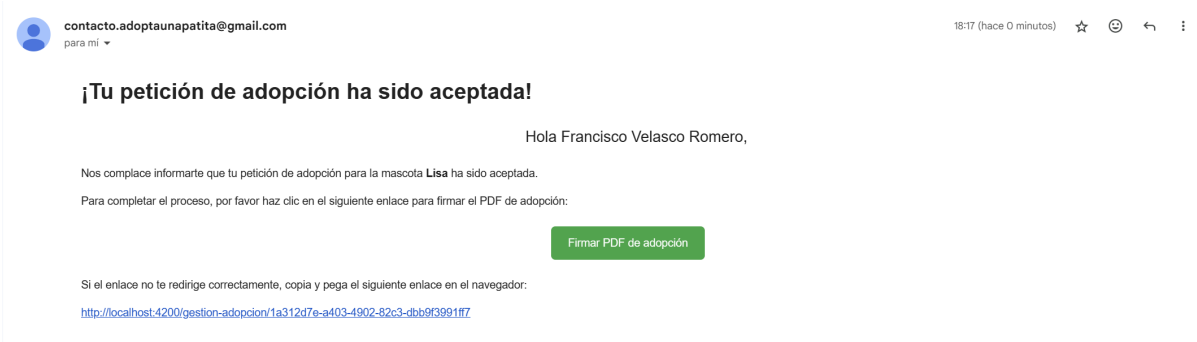


Figura 91.- Petición aceptada

El siguiente paso tal y como indica el correo es firmar el PDF de adopción. Para ello pinchamos en el enlace.

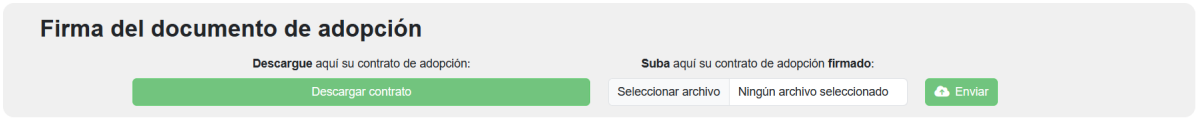


Figura 92.- Firma de contrato

Se proporcionarán dos opciones, un botón para descargar el contrato y así poder firmarlo y luego, una opción para subir dicho contrato firmado. Procedemos a enviar el contrato firmado.

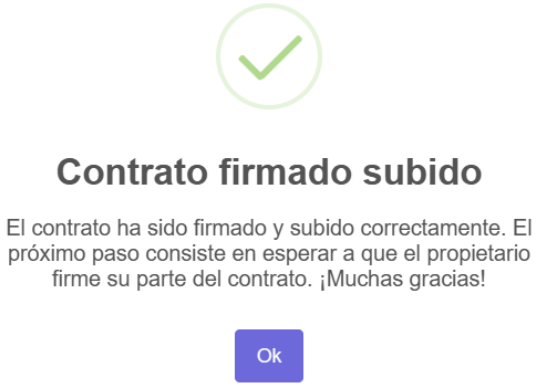


Figura 93.- Confirmación firma

Una vez nos llegue al correo la confirmación de que el propietario ha firmado su parte del contrato, entonces se dará por comenzada oficialmente la adopción.

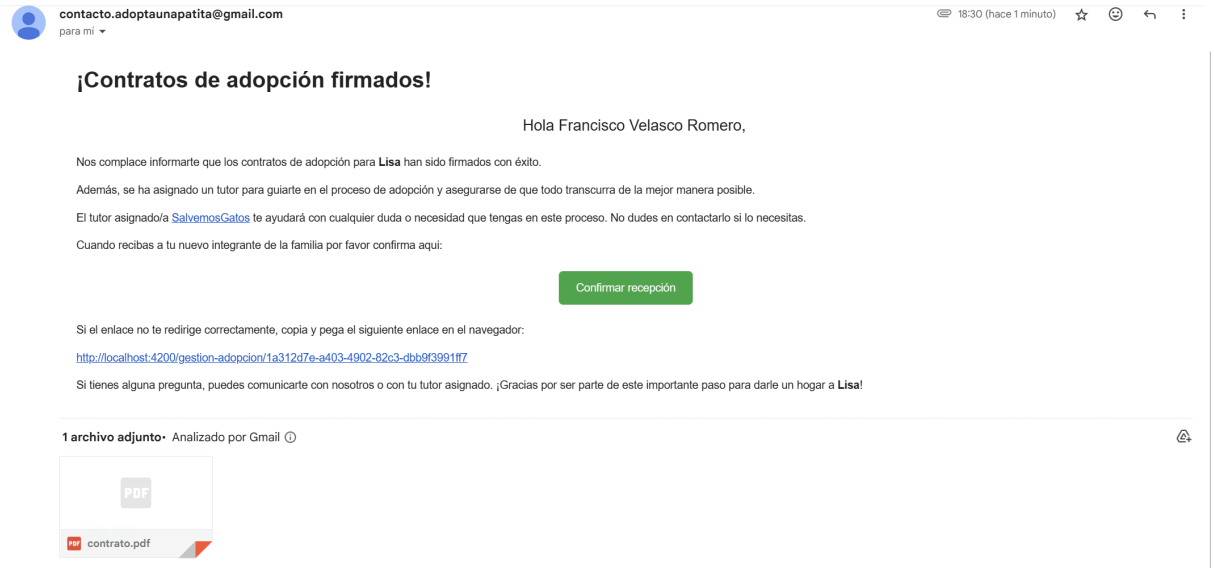


Figura 94.- Confirmación final adopción

Podemos ver datos útiles como el contrato adjunto firmado por ambas partes y el tutor asignado con hipervínculo a su chat.

En este punto el animal estará de camino a nuestro hogar por lo que debemos confirmar su recepción.

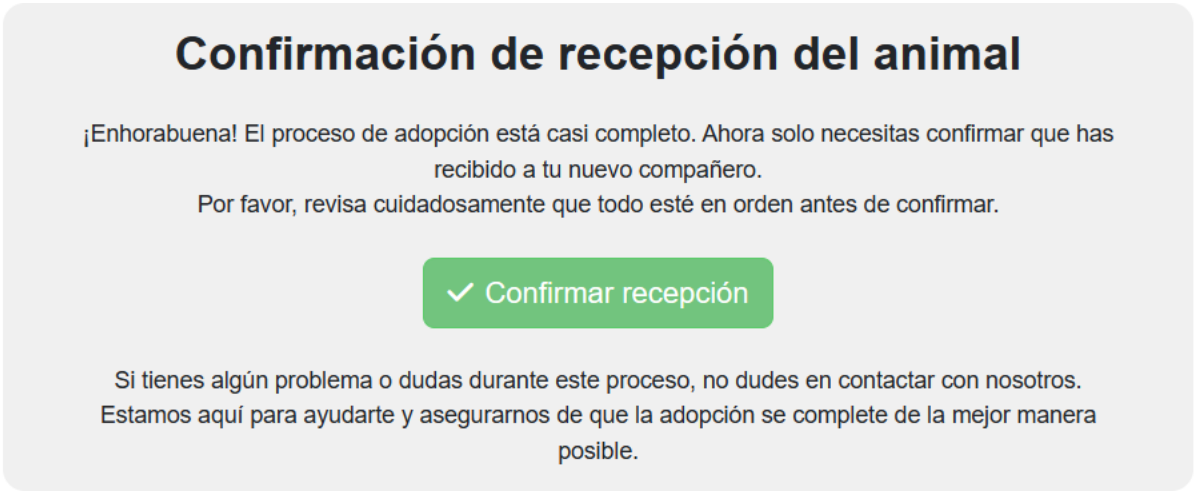


Figura 95.- Confirmar recepción

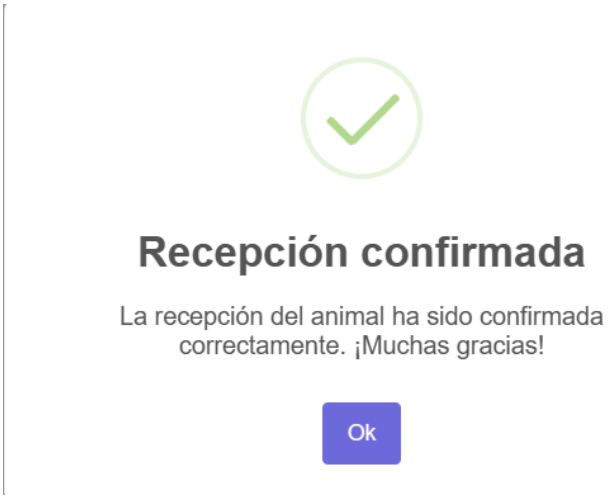


Figura 96.- Recepción confirmada

A partir de este momento el animal ya está adoptado, pero no definitivamente. Comienza el proceso de seguimiento en el que se pedirá feedback periódicamente hasta alcanzar el año, si el feedback ha sido positivo todas las veces entonces la adopción será finalizada con éxito.

Si no se da el feedback en el plazo correspondiente se mandará un mensaje de advertencia.

Recordatorio: ¡Es hora de dar tu feedback!

Hola **Francisco**,

Esperamos que todo esté yendo bien con **Lisa**.

Te recordamos que hoy es el día de compartir cómo está siendo la experiencia con tu nuevo compañero. Para ello, por favor ponte en contacto con tu tutor asignado, **SalvemosGatos**, y proporciónale el feedback necesario.

Contactar con mi tutor

Este recordatorio es parte de nuestro compromiso de asegurar que **Lisa** está recibiendo los cuidados necesarios y que todo está marchando bien en su nuevo hogar.

Si ya has proporcionado tu feedback, ¡muchas gracias! Si no, te pedimos que lo hagas a la mayor brevedad posible para que podamos seguir garantizando el bienestar de **Lisa**.

Estamos aquí para ayudarte en cualquier cosa que necesites durante este proceso.

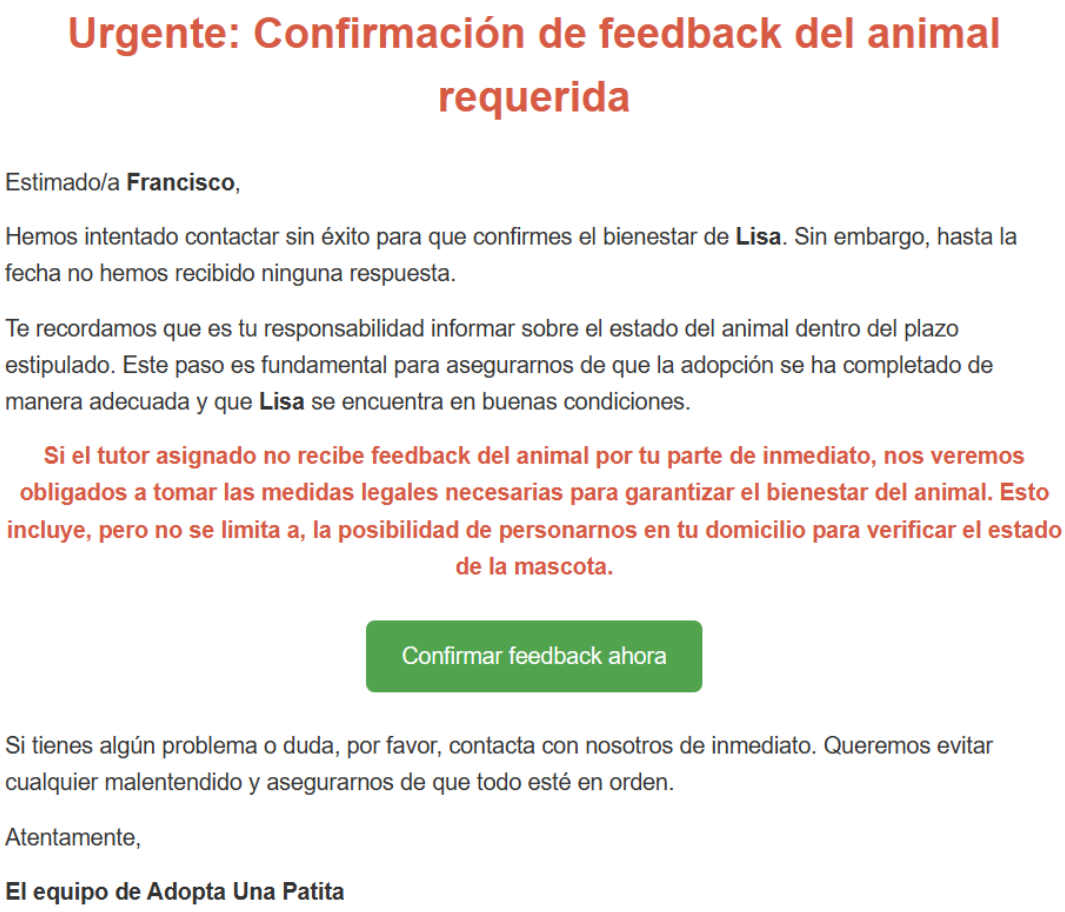
Atentamente,

El equipo de Adopta Una Patita

Figura 97.- Recordatorio feedback

En este momento el usuario al pinchar en el enlace sería llevado al chat con el tutor. Dicho tutor será el que confirme la recepción del mensaje y que todo está bien.

Si el feedback superase el periodo máximo se recibiría el siguiente mensaje de advertencia:



Urgente: Confirmación de feedback del animal requerida

Estimado/a **Francisco**,

Hemos intentado contactar sin éxito para que confirmes el bienestar de **Lisa**. Sin embargo, hasta la fecha no hemos recibido ninguna respuesta.

Te recordamos que es tu responsabilidad informar sobre el estado del animal dentro del plazo estipulado. Este paso es fundamental para asegurarnos de que la adopción se ha completado de manera adecuada y que **Lisa** se encuentra en buenas condiciones.

Si el tutor asignado no recibe feedback del animal por tu parte de inmediato, nos veremos obligados a tomar las medidas legales necesarias para garantizar el bienestar del animal. Esto incluye, pero no se limita a, la posibilidad de personarnos en tu domicilio para verificar el estado de la mascota.

Confirmar feedback ahora

Si tienes algún problema o duda, por favor, contacta con nosotros de inmediato. Queremos evitar cualquier malentendido y asegurarnos de que todo esté en orden.

Atentamente,

El equipo de Adopta Una Patita

Figura 98.- Advertencia feedback

Finalmente, cuando pasa un año y se completan todos los feedbacks satisfactoriamente, se da por finalizada la adopción definitiva del animal.

¡Proceso de adopción finalizado con éxito!

Hola **Francisco**,

¡Nos complace informarte que has completado exitosamente todos los feedbacks periódicos del proceso de adopción de **Lisa**!

Después de un año de seguimiento, estamos felices de darte la enhorabuena por haber finalizado todo el proceso de manera satisfactoria. Tu dedicación y compromiso para asegurar el bienestar de **Lisa** han sido ejemplares.

Desde este momento, damos por concluido el proceso de adopción, y **Lisa** es oficialmente un miembro permanente de tu familia. Sabemos que continuará recibiendo el amor y los cuidados que merece.

¡Felicitaciones a ti y a Lisa por este gran logro!

Te deseamos lo mejor en esta nueva etapa juntos y recuerda que siempre estaremos aquí para apoyarte en lo que necesites.

Gracias por confiar en nosotros y por darle a **Lisa** un hogar lleno de amor.

Atentamente,

El equipo de Adopta Una Patita

Figura 99.- Adopción finalizada

UNIVERSIDAD DE MÁLAGA | uma.es

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

E.T.S. DE INGENIERÍA INFORMÁTICA