Universitat Oberta de Catalunya

uoc.edu

Aplicación web para la gestión de mascotas

Daniel De Paz San José
Máster Universitario en Ingeniería de Telecomunicación
Smart Cities

Rubén Molina Casasnovas
Carlos Monzo Sánchez

30 de mayo de 2022

FICHA DEL TRABAJO FINAL

Título del trabajo:	*APLICACIÓN WEB PARA LA GESTIÓN DE MASCOTAS*
Nombre del autor:	*Daniel De Paz San José*
Nombre del consultor/a:	*Rubén Molina Casasnovas*
Nombre del PRA:	*Carlos Monzo Sánchez*
Fecha de entrega (mm/aaaa):	05/2022
Titulación:	*Máster Universitario en Ingeniería de Telecomunicación*
Área del Trabajo Final:	*Smart Cities*
Idioma del trabajo:	*Castellano*
Palabras clave	Mascotas, gestión, cuidados, aplicación, web

Resumen del Trabajo (máximo 250 palabras): *Con la finalidad, contexto de aplicación, metodología, resultados i conclusiones del trabajo.*

El objetivo de este proyecto es implementar una aplicación web, capaz de centralizar el cuidado de mascotas e interconectar a estas con la ciudad.

En un contexto en el que la tenencia de mascotas aumenta año tras año, y el cuidado de estas conlleva una organización mayor, se hacen imprescindibles mecanismos de gestión y organización. Tradicionalmente estos mecanismos han utilizado técnicas analógicas, siendo el objetivo de este proyecto generar un sistema de gestión online, capaz de automatizar las tareas necesarias, y capaz de interconectar a las mascotas con los servicios de la ciudad, a través de las ventajas de la interconexión digital.

Para realizar este proyecto se desarrollará una aplicación web, basada en Java y Angular, la cual permita realizar todas las funciones requeridas. Esta aplicación será accesible por cualquier usuario de forma online, permitiendo una gestión remota de la misma, y un acceso a los datos desde cualquier lugar.

Como resultado, la aplicación web generada permitirá centralizar los datos de las mascotas (nombre, fecha de nacimiento, vistas al veterinario, comidas etc), también permitirá almacenar información de la mascota (fotos, registro de gastos, de comidas etc), permitirá a su vez contactar con servicios de la ciudad (veterinarios, cuidadores etc.) y reservar servicios de la cuidad como peluquería, centros de lavado, parques etc.

Abstract (in English, 250 words or less):

The main objective of this project is the implementation of a web application which provides the capability to unify pet cares and maintainability in a single application. Furthermore, the application will permit to interconnect pets with cities amenities.

For the last years, the number of pets has been increasing all over the world. Pets' maintainability is based on certain repetitive actions, which require organizational abilities. Those organizational abilities have been performed, historically, by manual process like calendars.

The main objective of the designed project is the generation of a product with the ability to centralize all the maintenance actions, in one single web application. Furthermore, the designed product will automate some processes required by the maintenance actions.

To obtain the desired product, a web application, based on Java and Angular will be developed. This web application will be online, being accessible at any place and time by any user.

As a result, the developed web application will permit to centralize pets' data (name, birthdate, food). Also, it will permit to store pets' data (photos, financial expenses). As well, the connection with city professionals (veterinarians, pet carers etc.) and city amenities (parks, hairdressing, cleaning centers, etc)

Índice

Lista de figuras

Lista de tablas

1 Introducción

Este primer apartado de la memoria tiene como objetivo exponer el punto de partida del proyecto. Para ello, se cita el contexto y la justificación de este, así como los objetivos principales, el enfoque utilizado y la planificación.

1.1 Contexto y justificación del Trabajo

Actualmente, en España existen 29 millones de mascotas censadas, 300 millones si nos referimos a Europa [1]. La tenencia de estas mascotas supone para los dueños ciertas tareas periódicas de manutención, cuidados y mantenimiento. Un ejemplo es, en el caso de los perros, la aplicación trimestral de una pastilla antiparasitaria, o la aplicación de pipetas y collares antiparasitarios, así como visitas al veterinario, compra de comida etc. Todas estas acciones requieren cierta planificación y organización, la cual se suele gestionar con medios tradicionales como calendarios.

Por otro lado, según reflejan varios informes y noticias [2], actualmente en España hay el doble de mascotas que niños menores de 15 años. Este hecho se ha visto producido en parte por la pandemia del coronavirus, cuya crisis asociada ha conllevado que muchas familias no puedan afrontar la manutención de un niño, pero si la de una mascota, significativamente de menor coste, generando con ello un aumento importante en la tenencia de mascotas. Como resultado se ha generado un mercado con un número de clientes al alza, y con una necesidad por cubrir, la gestión digital.

1.2 Objetivos del Trabajo

El objetivo principal de este proyecto es generar una aplicación web, la cual permita centralizar y facilitar la realización de todas las tareas relacionadas con el cuidado de las mascotas.

Para cumplir dicho objetivo, se abordarán los siguientes tres puntos:

1. Centralización de datos de mascotas.
2. Gestión y planificación de cuidados y mantenimiento de las mascotas.
3. Interacción con espacios destinados a mascotas de la ciudad

De esta forma, la aplicación generada permitirá interconectar a los dueños de mascotas con los servicios que ofrece la ciudad relativos a las mismas. Como consecuencia, se obtendrá un producto que permitirá generar un “alias virtual” de cada mascota, introduciendo a estas en el mundo digital, aprovechándose de todos los beneficios que este aporta.

1.3 Enfoque y método seguido

Actualmente existen distintas aplicaciones capaces de centralizar los datos de las mascotas y los cuidados de estas (algunos ejemplos podrían ser 11pets [3] o ExpertoAnimal [4]). Además, existen aplicaciones para contratar servicios de cuidadores de mascotas (dogbuddy [5]).

Sin embargo, todas estas aplicaciones ofrecen funcionalidades concretas de forma aislada y sin interconexión. Es por ello por lo que el objetivo de este proyecto es implementar una nueva aplicación que englobe todas estas funcionalidades, pero de forma conjunta e integrada.
´

Una posible estrategia para alcanzar este objetivo podría ser la conexión de las aplicaciones existentes, mediante un servicio de interconexión entre ellas. El principal problema de esta propuesta es que existe un gran número de aplicaciones a interconectar, cada una con su implementación propia, y sin ninguna homogeneidad en la estructuración de sus datos. Por lo tanto, la conexión de estos servicios requeriría un análisis personalizado de cada uno, con el fin de alcanzar una estructura común.

Otra posible estrategia sería la realización desde cero de una aplicación propia, la cual ofrezca todos los servicios citados. La ventaja de esta estrategia es la capacidad total de decisión en la estructuración de los datos a almacenar, y la gestión de estos. Por otro lado, el principal inconveniente es el no aprovechamiento de la cartera de clientes de los que ya disponen las aplicaciones existentes.

Como resultado, y en el contexto en el que se enmarca este ***TFM***, con el fin de poder aprovechar la flexibilidad que ofrece realizar una aplicación desde cero, y con la intención de desarrollar servicios personalizados que permitan implementar los conocimientos adquiridos en el máster, se ha optado por utilizar la segunda estrategia, implementando una aplicación desde cero.

Debido a que actualmente no existe ningún competidor que destaque sobre los demás, en cuanto a número de clientes se refiere, la pérdida de la cartera de clientes producida al escoger la estrategia de realizar una aplicación desde cero no debería ser un gran inconveniente. Esto es debido a que el producto que ofrece este proyecto mejora y centraliza las funcionalidades ofrecidas por sus competidores. Por lo tanto, se espera que genere una fuerte atracción hacia la plataforma, aglutinando parte de la cartera de clientes de la que disponen el resto de los competidores.

1.4 Planificación del Trabajo

Para la realización del trabajo, a nivel de recursos, se necesitará un ordenador, desde el que se realizará la implementación del producto y la elaboración de la documentación.

A nivel de software, se utilizará Java para la realización del Back de la aplicación, y ***Angular*** para la realización del Front. La aplicación se desplegará inicialmente en un servidor web levantado en el propio ordenador de trabajo, pudiéndose llegar a desplegar en un servidor público de algún distribuidor comercial (***AWS***, Google Could etc). La aplicación utilizará, además, una base de datos PostgreSQL, la cual se alojará en la misma máquina en la que se despliegue la aplicación web.

La realización del proyecto se dividirá en cinco fases, coincidiendo con los hitos de entrega de la asignatura. Las tareas que componen cada una de las fases son:

- **Fase 1:**
  - Definición de Objetivos y alcance.
  - Planificación del proyecto.
  - Realización de la memoria de la PEC 1.
- **Fase 2:**
  - Definición del estado de arte.
  - Realización de la memoria de la PEC 2.
- **Fase 3:**
  - Implementación.
  - Realización de pruebas.
  - Realización de la memoria de la PEC 3.
- **Fase 4:**
  - Solución de errores.
  - Realización de la memoria completa del ***TFM***.
- **Fase 5:**
  - Realización de presentación del ***TFM***.

El diagrama de Gantt que comprende todas las tareas puede verse en la siguiente figura:

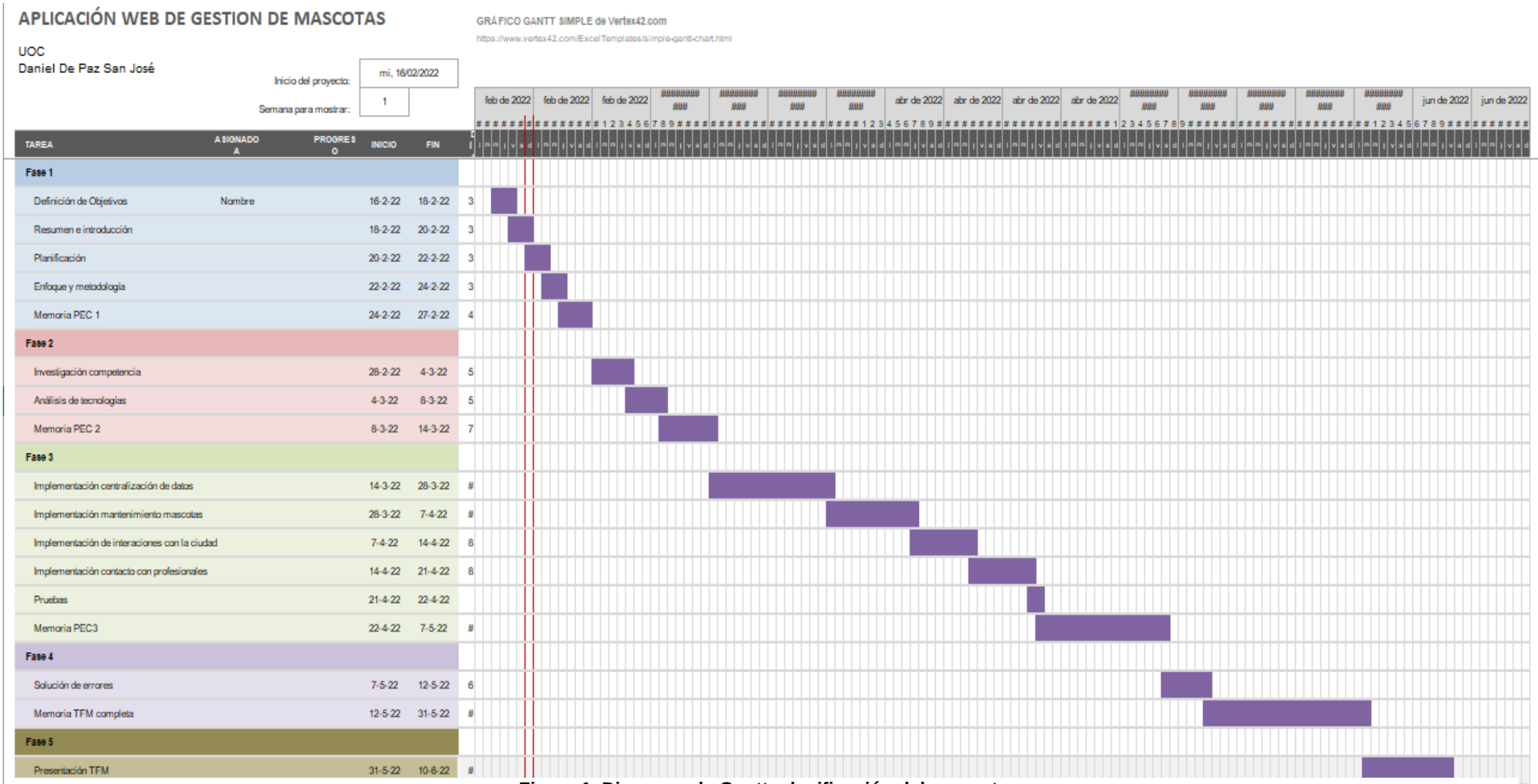

Figura 1. Diagrama de Gantt, planificación del proyecto.

1.5 Breve sumario de productos obtenidos

Como resultado de la elaboración de este proyecto, se pretende obtener una aplicación web, accesible desde cualquier navegador. Esta aplicación web estará compuesta por una parte ***front*** ,de visualización del usuario, y una parte ***back*** , la cual contendrá la lógica de negocio de la aplicación.

La aplicación web estará desplegada en un servidor de aplicaciones para que sea accesible desde cualquier lugar con conexión a internet.

1.6 Breve descripción de los otros capítulos de la memoria

El contenido de esta memoria se divide en varios apartados. El inicio de estos es el apartado 2, en el que se expone el estado del arte, poniendo en contexto la situación del sector de las mascotas a nivel nacional, y citando casos y proyectos previos relacionados con el ámbito del trabajo. La memoria continúa con las especificaciones y restricciones del proyecto en el apartado 3**¡Error! No se encuentra el origen de la referencia.**, donde también se definen los actores, y casos de uso implicados. Se continúa exponiendo, en el apartado 4, el diseño propuesto para cumplimentar los requisitos establecidos. Tras exponer este diseño desde un puesto de vista de alto nivel se muestra la implementación a más bajo nivel en el apartado 5. Finalmente se muestran los resultados obtenidos en el apartado 6 y las conclusiones y líneas de trabajo futuro en los apartados 7 y 8.

2 Estado del arte

Este segundo apartado de la memoria pondrá en contexto el estado actual del sector de las mascotas en nuestro país. Se analizará la volumetría que ofrece el sector, y el crecimiento de esta en los últimos años. Por otro lado, se compararán las distintas soluciones existentes, ofrecidas por las empresas del sector, a los problemas planteados en este proyecto.

2.1 Datos del sector

En los últimos años se ha producido un aumento significativo en la cantidad de mascotas registradas en nuestro país. Es importante destacar que en España no existe un censo único de animales de compañía, sino que cada comunidad autónoma mantiene su registro propio. Existen algunas asociaciones como ***REIAC*** [6], cuyo principal objetivo es generar una red informática que permita centralizar los censos de mascotas de cada comunidad. Por otro lado, distintas asociaciones relacionadas con empresas privadas del sector también realizan y mantienen sus censos propios. Un ejemplo sería ***ANFAAC*** [7], asociación enmarcada en la alimentación de mascotas, o Veterindustria [8], asociación Empresarial Española de la Industria de Sanidad y Nutrición Animal.

A nivel estatal, los censos de mascotas de las comunidades autónomas presentan un aumento generalizado en el registro de animales de compañía. Un ejemplo de ello es el Ayuntamiento de Madrid, el cual, en los últimos seis años presenta una media de incremento anual de un 8% en perros y gatos censados, aumentando en 91.640 el número de censos [9].

Figura 2. Censo de especies Canina y Felina del Ayuntamiento de Madrid. [9]

Otro ejemplo sería el Ayuntamiento de Barcelona, el cual en los últimos cuatro años ha aumento en 10.423 el número de mascotas censadas, lo que supone un incremento medio anual de un 5% [10]

Figura 3. Censo de mascotas Ayuntamiento de Barcelona. [10]

Los datos de las comunidades van en línea con los presentados en el informe conjunto de 2021 de ***ANFAAC*** y Veterindustria [7], según los cuales en España habría más de 29 millones de mascotas.

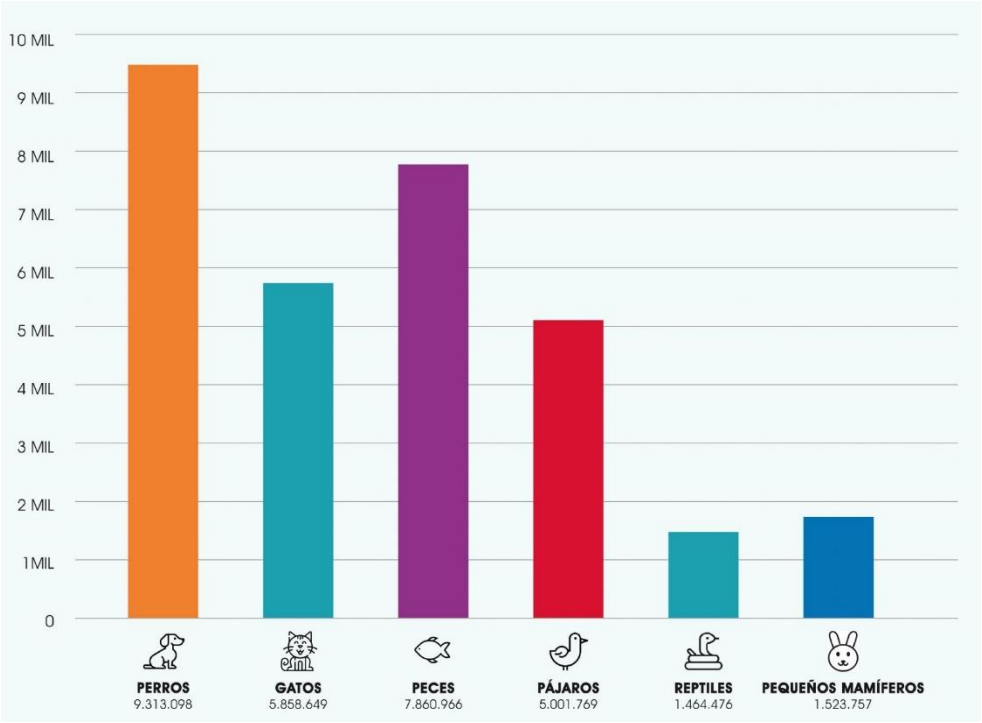

Figura 4. Estudio de Censos 2021 ANFAAC y Veterindustria. [7]

Todos estos censos, junto con otras muchas estimaciones, dejan claro que el número de mascotas en nuestro país está aumentando. Sin embargo, las cifras tomadas deben analizarse con cautela, dado que actualmente en España, no existe ningún mecanismo de control efectivo que asegure que los censos de las comunidades autónomas sean fieles a la realidad. Esto es así dado que no existen controles activos sobre mascotas sin censar, o mascotas fallecidas no dadas de baja. Por lo tanto, existe cierto margen de error en las cifras de los

censos. Además, los censos de las asociaciones privadas son estimaciones sobre volumetrías de mercado, por lo que también contienen cierto margen de error.

2.2 Crecimiento del mercado

Otro dato que demuestra el auge del sector es la tendencia de búsquedas en Google de los términos “adoptar perro” y “adoptar gato”. Como se puede ver en la Figura 5, a comienzos de 2020 en España, la tendencia de búsqueda de ambos términos aumentó, coincidiendo justo con el inicio del confinamiento debido a la pandemia del COVID-19.

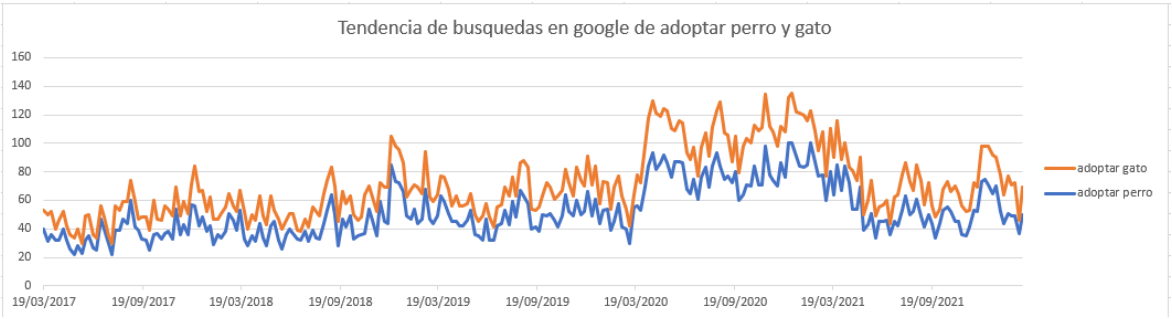

Figura 5: Tendencia en Google: adoptar perro y adoptar gato. [11]

Además, el estudio anual desarrollado por la ***Fundación affinity***, el cual se centra en analizar las adopciones y abandonos de mascotas en nuestro país, demuestra que la adopción nacional de animales de compañía se ha visto reforzada en los últimos años. Este estudio muestra, además, que la adopción nacional aumenta en detrimento de la adopción internacional [12].

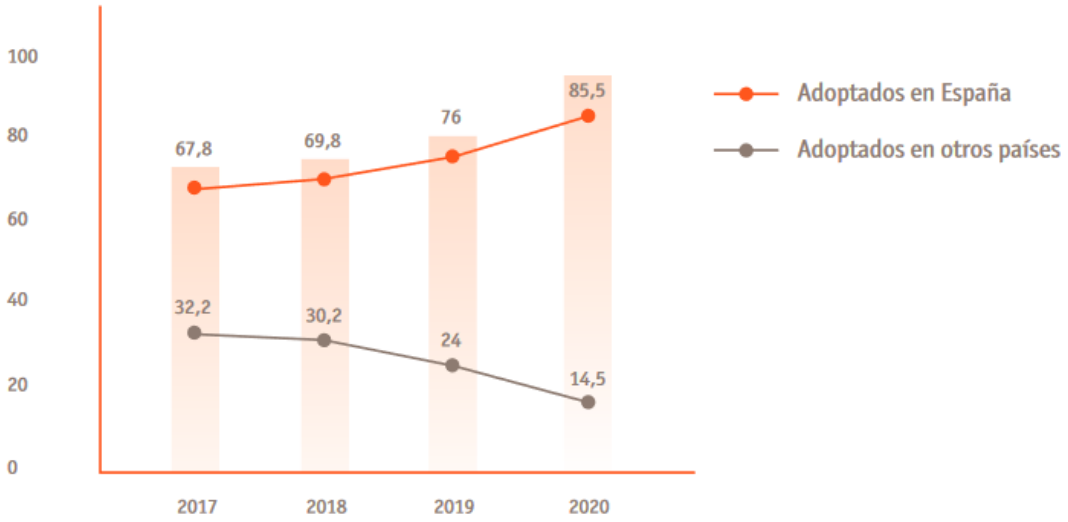

Figura 6. Porcentaje de perros adoptados en España. [12]

Aunque no existen estudios concluyentes, distintos artículos atribuyen este crecimiento a dos causas [13]:

Por un lado, la situación de confinamiento generada por la pandemia del COVID-19, obligó a muchas personas a permanecer en sus hogares. Este hecho generó el clima perfecto para tener una mascota, dado que muchos de los impedimentos existentes sobre la adopción de mascotas desaparecieron. Las personas pasaron a tener el tiempo suficiente para atender a sus mascotas, además, la situación de confinamiento favorecía el poder salir a la calle en caso de tener que pasear a una mascota. El confinamiento generó una necesidad ferviente de luchar contra la monotonía y el aburrimiento, lo cual en muchos casos se hizo por medio de las mascotas. Como resultado, uno de los efectos colaterales generados por el confinamiento fue que muchas personas se decidieran a adoptar una mascota.

Por otro lado, durante los últimos años se ha estado produciendo un cambio generacional, el cual, entre muchos otros aspectos, está cambiando los modelos de familia más habituales. Según un estudio del Instituto Nacional de Estadística [14], el número de hogares de una persona aumentó un 2,0% en 2020, el de dos un 0,6% y el de cinco o más creció un 1,3%. El informe también indica que en España hay 3,91 millones de hogares formados por parejas sin hijos, 2,89 millones formados por parejas con un hijo y 2,76 millones por parejas con dos hijos. Estos datos corroboran que la nueva tendencia de hogares es el formado por una pareja sin hijos.

Esto, unido al descenso de la natalidad en España, el cual según el ***INE***, ha sufrido una disminución de un 25% en los últimos seis años [15], ha generado que en los nuevos modelos de familia, la tendencia de tener una mascota aumente frente a la tendencia de tener un hijo.

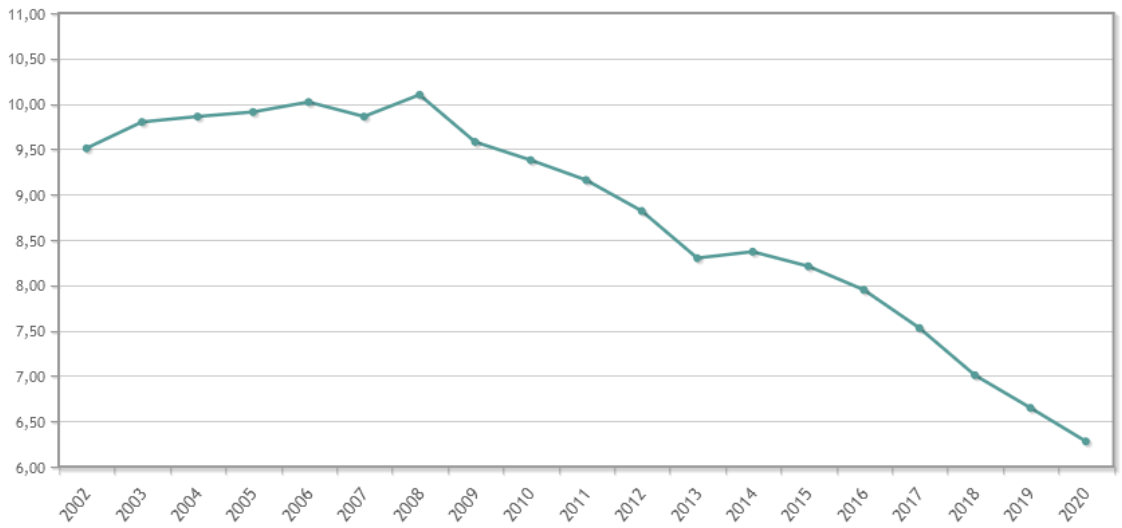

Figura 7. Tasa de Natalidad en España, nacidos por mil habitantes. [15]

2.3 Cuidados de mascotas

En este escenario, en el que queda claro que el mercado de las mascotas está en auge, es importante definir

para la elaboración de este proyecto, en qué consisten los cuidados y mantenimiento de una mascota.

Como se muestra en la Figura 4, las especies predominantes de mascotas en España son: perros, peces, gatos y pájaros. Es importante destacar, que cada especie requiere unos cuidados específicos, los cuales pasaremos a detallar a continuación:

- **Perros**:

Los perros son la especie de mascota por excelencia, su esperanza de vida se encuentra entre los 10 y 13 años, y se estima que en Europa en 2020 existían unos 90 millones (sin contar los perros callejeros) [16].

Los cuidados referentes a los perros se engloban en 5 grandes bloques [17]:

1. *Alimentación*: Hay que ofrecer a los perros alimentos de calidad en función de su etapa de vida, según si es cachorro, adulto o senior. La recomendación es que un perro adulto debe alimentarse dos veces al día.
2. *Educación*: Cobra especial importancia en las primeras etapas de vida de un perro. Consiste no solo en enseñar donde hacer sus necesidades, sino que implica la interacción del perro con su entorno, el trato con otros perros, y con otras personas.
3. *Higiene*: Consiste en el cepillado de pelo, baño, corte de uñas y limpieza de ojos y oídos. Es un cuidado periódico que debe realizarse con bastante frecuencia para evitar enfermedades.
4. *Actividad física*: Los perros necesitan moverse y ejercitarse cada día. Por ello, entre los cuidados de un perro está el sacarle a pasear al menos tres veces al día.
5. *Cuidados veterinarios*: Además de los cuidados por enfermedad, los perros disponen de un plan de vacunas, así como un plan de desparasitaciones. Las desparasitaciones consisten en la ingesta de una pastilla antiparasitaria interna cada 3 meses, así como la aplicación de pipetas antiparasitarias externas cada 6 meses.

- **Peces**:

Suelen denominarse las mascotas más fáciles de cuidar. Su esperanza de vida es de entre 2 y 3 años. Sus cuidados requieren únicamente dos acciones [18]:

1. *Alimentación*: Los peces deben ser alimentados una vez al día, con un tipo de alimento específico según el tipo de pez.
2. *Mantenimiento de la pecera*: Se debe mantener en correctas condiciones el agua de la pecera, midiendo el pH una vez a la semana y cambiando el filtro de agua y limpiando la pecera cada mes. Cada cierto tiempo se deberá, además realizar un remplazo del agua de la pecera.

- **Gatos**:

Una de las mascotas más apreciadas del mundo. Su esperanza de vida está entre los 12 y 18 años. Sus cuidados se parecen mucho a los de los perros, agrupándose estos en [19]:

1. *Alimentación*: Dependiendo de la etapa de vida en la que se encuentre el animal requerirá de un pienso específico. Un gato adulto debe alimentarse dos veces al día.
2. *Higiene*: Los gatos, por naturaleza, se realizan su propia higiene mediante el lamido, por lo que no es necesario bañarles. Sin embargo, el lamido suele generar bolas de pelo, las cuales se deben prevenir mediante el cepillado diario. Además, es necesaria una limpieza frecuente de los ojos.
3. *Arenero*: Los gatos domésticos realizan sus necesidades en areneros, los cuales deben ser limpiados diariamente. La arena de estos areneros se debe cambiar con una frecuencia semanal.
4. *Veterinario*: Al igual que los perros, los gatos tienen un plan de vacunas anual, además de los cuidados en caso de enfermedad.

- **Aves**:

Una de las especies de mascotas más variada. Su esperanza de vida depende de la especie, desde los canarios con 15 años a los loros con una media de entre 20 y 75 años dependiendo de la especie [20]. Los cuidados de las aves, por lo general, engloban los siguientes puntos: [21]

1. *Alimentación*: Se les debe proveer de alimento y agua, cambiando estos diariamente.
2. *Limpieza de la jaula*: Las jaulas de aves deben limpiarse y desinfectarse una vez a la semana
3. *Veterinario*: Se recomienda realizar un chequeo veterinario una vez al año.

Como podemos ver, cada especie de animal requiere un tipo de cuidados diferente, con acciones periódicas realizadas con distintas frecuencias, las cuales se deben planificar para el correcto mantenimiento del animal.

2.4 Precedentes en el mantenimiento

Tras analizar los distintos cuidados que requiere cada especie, es evidente que independientemente de la especie de mascota, se requiere de un sistema de organización que facilite la planificación de todas las acciones periódicas, necesarias para el mantenimiento del animal.

Tradicionalmente esta organización se ha realizado utilizando medios analógicos. Es muy frecuente que, con la compra de comida, o con las visitas al veterinario, las empresas ofrezcan de forma gratuita a los clientes distintos modelos de calendarios organizativos, pensados para controlar la frecuencia con la que se realiza el mantenimiento. Algunos ejemplos de estos medios analógicos pueden verse en la Figura 8.

Figura 8. Muestra de medios analógicos de organización de mascotas.

2.5 Actualidad del mantenimiento

En los últimos años han surgido diferentes aplicaciones web y aplicaciones para smartphones, las cuales han tenido como objetivo digitalizar y automatizar algunas de las tareas de mantenimiento citadas. Algunos ejemplos son:

11pets [3]	
Tecnología	Aplicación móvil Android e IOS
Características	• Control veterinario • Recordatorios de vacunas, lavados, peluquería, higiene etc.

iKibble Free [22]	
Tecnología	Aplicación móvil Android e IOS
Características	• Consejos de alimentación. • Dietas

Noblecan [23]	
Tecnología	Aplicación web
Características	• Contacto con adiestradores. • Clases online sobre mantenimiento.

Este conjunto de aplicaciones, además de muchas otras presentes en el sector, han digitalizo algunas de las tareas relacionadas con el cuidado de mascotas, generando con ello un entorno de mayor comodidad para los usuarios.

Sin embargo, el mayor problema existente en el sector es la especialización de todas estas aplicaciones. Dicho de otro modo, cada una de las aplicaciones digitaliza alguna de las acciones de cuidado de una mascota, pero ninguna digitaliza el total de las acciones necesarias. Esto genera un ambiente en el que son necesarias varias aplicaciones para poder digitalizar los cuidados de una mascota.

2.6 Futuro del mantenimiento

Como resultado al escenario descrito, este proyecto de fin de máster pretende implementar un producto capaz de unificar, en una única aplicación, todas las funcionalidades relativas al cuidado de mascotas. De esta forma, no solo se digitalizará el mantenimiento de las mascotas, sino que se centralizará, mejorando con ello la experiencia de los usuarios.

Por último, el producto desarrollado aportará valor añadido interconectando a las mascotas con la ciudad. De esta forma, desde la aplicación generada se permitirá reservar o consultar el estado de distintos servicios de la ciudad (parques, zonas de entrenamiento, centros de lavado), así como contactar con profesionales (adiestradores, cuidadores, veterinarios), además de almacenar los archivos digitales de las mascotas (imágenes y videos).

Las líneas de trabajo futuro de este proyecto se centrarían en desarrollar una nueva funcionalidad de red social, en la que los usuarios pudiesen compartir los archivos digitales de sus mascotas con otros usuarios, además de poder poner comentarios en las publicaciones de sus “amigos”, y valorar los servicios profesionales conectados con la aplicación.

3 Planteamiento del proyecto

En este apartado se realizará el planteamiento del proyecto previo al diseño. Se definirá con detalle: el alcance, los actores, los casos de uso y los distintos escenarios que abordará el proyecto.

3.1 Alcance

Dado el tiempo para realizar este trabajo fin de máster, el primer punto para realizar el planteamiento del proyecto consistirá en definir detalladamente el alcance de este.

Para definir lo que incluirá y lo que no incluirá el alcance, nos basaremos en los dos perfiles de uso que tendrá la aplicación.

- El perfil *Dueño*, destinado a las personas que quieran utilizar la aplicación para el mantenimiento y cuidado de su mascota.

- El perfil *Servicio*, destinado a los negocios o profesionales que quieran ofrecer sus servicios a través de la aplicación.

3.1.1 Requisitos contemplados en el alcance

Tal y como se indica en el apartado 1.2, el proyecto cuenta con tres objetivos de alto nivel, los cuales pasarán a detallarse para concretar la funcionalidad que incluye cada uno.

O1. Centralización de datos de mascotas.

- **R1.1**: La aplicación permitirá, al perfil *Dueño*, almacenar, visualizar y actualizar los **datos identificativos** de una mascota: nombre, fecha de nacimiento, especie, raza, sexo, nº de identificación.
- **R1.2**: La aplicación permitirá que el perfil *Dueño*: almacene, visualice y actualice los **datos sanitarios** de una mascota relacionados con vacunaciones y tratamientos (tipo de vacuna/tratamiento, fecha, centro veterinario, lote de vacuna/tratamiento, nº colegiado responsable y coste).
- **R1.3**: La aplicación permitirá, al perfil *Dueño*, almacenar, visualizar y actualizar los **datos relacionados con la alimentación** de la mascota: tipo de comida, marca, cantidad, fecha de compra, fecha de caducidad, coste, lugar de compra.

02. Gestión y planificación de cuidados y mantenimiento de las mascotas.

- **R2.1**: La aplicación permitirá, al perfil *Dueño*, configurar **recordatorios** periódicos relativos a: tratamientos médicos, compras de comida, visitas al veterinario, limpiezas, visitas a centros de estética y limpieza del hábitat de la mascota.

- **R2.3**: La aplicación proporcionará al perfil *Dueño,* **consejos** relativos al mantenimiento de la mascota: Tipos de alimentación según la edad, periodicidad de la limpieza del animal, periodicidad del mantenimiento del hábitat, campañas de tratamientos veterinarios y adiestramiento.
- **R2.4**: La aplicación proporcionará, al perfil *Dueño*, un registro histórico de **mantenimiento** según la mascota:
    - Peces: Registro histórico de PH de la pecera, cantidad de agua y temperatura.
    - Perros: Registro de collares y pipetas antiparasitarias.
    - Gatos: Registro del cambio de arena.
    - Aves: Registro de limpieza de la jaula

03. Interacción con espacios destinados a mascotas de la ciudad

- **R3.1**: La aplicación permitirá a el perfil *Servicio* dar de alta un *servicio* en la aplicación, almacenando: Tipo de servicio, localización, horario y datos de contacto. (Se entiende por servicio cualquier tipo de actividad profesional destinada al mundo animal: Veterinarios, adiestradores etc)
- **R3.2**: La aplicación ofrecerá a el perfil *Dueño,* información sobre los distintos *servicios* dados de alta en la aplicación.
- **R3.3**: La aplicación ofrecerá al perfil *Dueño*, información sobre servicios públicos de la ciudad: Parques y rutas de paseo.

3.1.2 Elementos no contemplados en el alcance

A continuación, se detallarán las funcionalidades que no estarán contempladas en el alcance. Cabe destacar que, a pesar de no implementarse las funcionalidades que se detallarán a continuación, se plantearán en el apartado de líneas futuras del proyecto.

RE_1: La aplicación solo permitirá el registro de las siguientes especies de mascotas: perros, peces, gatos y aves.
RE_2: La aplicación solo ofrecerá información sobre servicios dados de alta en la propia aplicación. No proporcionará ningún mecanismo de reserva directa.
RE_3: Los recordatorios generados por un usuario serán únicamente accesibles por el usuario que los generó.

3.2 Actores

El escenario concreto en el que se enmarca este proyecto requiere de la presencia de tres actores principales, los cuales se definen a continuación. Entendemos por actor a todo individuo que forme parte de un grupo, el cuál desee utilizar la aplicación con un objetivo concreto.

Tabla 1. Perfiles de la aplicación.

Nombre	*Descripción*
Dueño	Usuarios que utilizan el sistema para registrar y gestionar una o varias mascotas.
Servicio	Usuarios que utilizan el sistema para promocionar un servicio profesional.
Aplicación	Este usuario de gestión se encargará de incluir en la aplicación información relativa a consejos y servicios de mantenimiento de mascotas, la cuál será accesible para el usuario *Dueño*.

3.3 Casos de uso

Como consecuencia de las especificaciones y restricciones definidas, surgen una serie de casos de uso que el proyecto debe ser capaz de resolver, los cuales se pueden ver en la Figura 9. Estos casos de uso definen situaciones de uso concretas a más alto nivel.

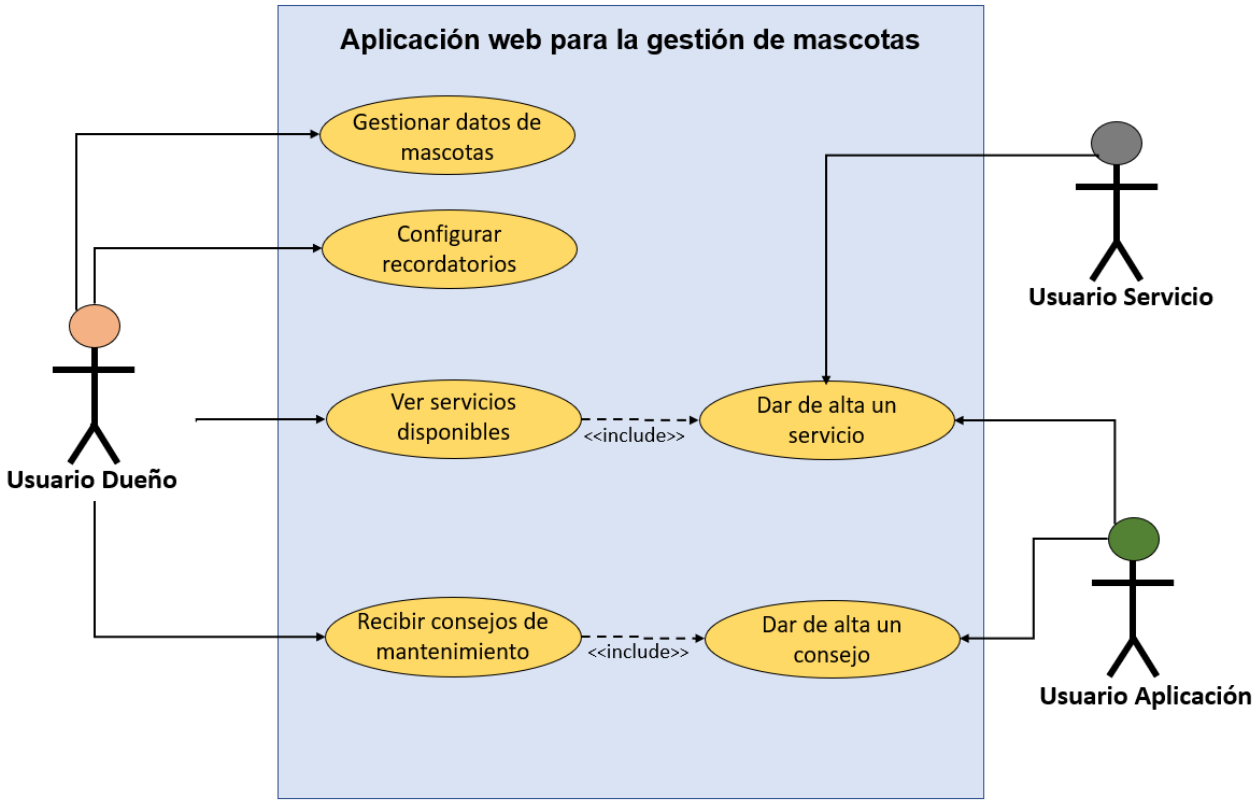

Figura 9. Diagrama de casos de uso de la solución propuesta.

A continuación, se muestra el detalle de cada uno de los casos de uso involucrados.

Tabla 2. Descripción del caso de uso 1.

Código	**CU1**
Nombre	Gestionar datos de mascotas
Actores	Usuario *Dueño*
Función	Almacenar, visualizar y actualizar toda la información relativa a una mascota
Descripción	El usuario *Dueño* almacena mediante la interfaz de la aplicación la información referente a los datos de sus mascotas, sus tratamientos, compras de comida o histórico de cuidados. Se permite al usuario, editar y visualizar la información introducida y existente, mediante varias ventanas agrupadas por categorías.
Requisitos	**R1.1**, **R1.2**, **R1.3**, **R2.4**
Restricciones	**RE_1**

Tabla 3. Descripción del caso de uso 2.

Código	**CU2**
Nombre	Configurar recordatorios
Actores	Usuario *Dueño*
Función	Generar recordatorios para el propio usuario
Descripción	El usuario *Dueño* podrá generar un recordatorio, indicando la fecha de activación, fecha de fin y periodicidad. Este recordatorio será individual para el propio usuario y no podrá ser compartido.
Requisitos	**R2.1**
Restricciones	**RE_3**

Tabla 4. Descripción del caso de uso 3.

Código	**CU3**
Nombre	Dar de alta un servicio
Actores	Usuario Servicio y Aplicación
Función	Registrar un servicio público o privado en la aplicación
Descripción	Los usuarios Servicio y Aplicación podrán registrar un servicio en la aplicación, introduciendo el nombre del servicio, su horario, tipo de servicio y localización. Estos servicios serán posteriormente mostrados a los usuarios *Dueño*.
Requisitos	**R3.1**
Restricciones	**RE_2RE_3**

Tabla 5. Descripción del caso de uso 4.

Código	**CU4**
Nombre	Ver servicios disponibles
Actores	Usuario *Dueño*
Función	Ver la información sobre los servicios dados de alta en la aplicación
Descripción	El usuario *Dueño* dispondrá de una ventana, en la que se le ofrecerá un listado con los servicios previamente dados de alta en la aplicación. Estos servicios podrán filtrarse por localización, horario de apertura y tipo de servicio.
Requisitos	**R3.2**, **R3.3**
Restricciones	**RE_2RE_2RE_3**

Tabla 6. Descripción del caso de uso 5.

Código	**CU5**
Nombre	Dar de alta un consejo
Actores	Usuario Aplicación
Función	Registrar un consejo en la aplicación
Descripción	El usuario Aplicación, personificado en el equipo de mantenimiento de la aplicación, podrá dar de alta un consejo indicando: tipo de consejo, especie destinataria, fecha de activación y texto del mensaje. Estos consejos serán enviados a todos los usuarios *Dueño*.
Requisitos	**R2.3**
Restricciones	**RE_2RE_3**

Tabla 7. Descripción del caso de uso 6.

Código	**CU6**
Nombre	Recibir consejos de mantenimiento
Actores	Usuario *Dueño*
Función	Visualizar los consejos de mantenimiento de una especie de mascota en concreto
Descripción	El usuario *Dueño* podrá consultar los consejos de mantenimiento generados por la aplicación, para un determinado tipo de mascota, siempre y cuando el usuario haya dado de alta previamente a ese tipo de mascota en su perfil.
Requisitos	**R2.3**
Restricciones	**RE_2RE_3**

4 Diseño

Una vez realizado el análisis de requisitos de la aplicación, y habiendo definido los casos de uso y tecnologías a utilizar, se está en disposición de realizar el diseño de la solución propuesta. Este diseño constará, por un lado, de la definición de la arquitectura a utilizar, junto con la definición de la estructura de base de datos.

4.1 Arquitectura

En el ámbito de las aplicaciones web una de las arquitecturas de software más extendidas es el modelo ***MVC*** (Modelo Vista Controlador). Este patrón de diseño especifica la estructura de alto nivel que debería tener una aplicación web. Esta estructura (Figura 10) está formada por tres componentes [24]:

- **Modelo**: Contiene la representación de los datos de la aplicación.
- **Vista**: Interfaz de usuario para el uso de la aplicación.
- **Controlador**: Conexión entre el Modelo y la Vista. Contiene la lógica de negocio de la aplicación.

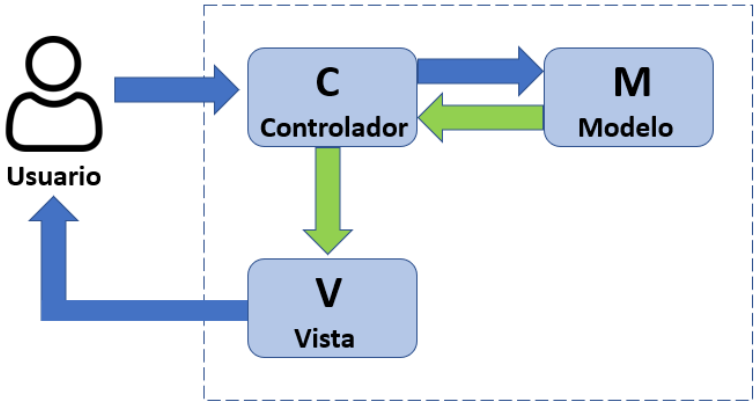

Figura 10. Arquitectura MVC (Modelo Vista Controlador)

Tomando como base este patrón de diseño, durante los años han surgido distintas implementaciones de este, siendo alguno de los modelos más extendidos la arquitectura monolítica y la arquitectura de microservicios.

La arquitectura monolítica se basa en el modelo MVC. Su característica principal es que la capa del controlador se implementa utilizando un único ejecutable lógico. Dicho de otra forma, toda la funcionalidad de la aplicación se centraliza en un único proyecto, el cual contiene toda la lógica de negocio y servicios necesarios [25].

La arquitectura de microservicios también se basa en el modelo MVC, pero a diferencia de la arquitectura monolítica el controlador se encuentra dividido en

múltiples proyectos. La lógica de negocio se divide en servicios independientes, que se comunican entre sí para ofrecer la funcionalidad conjunta de la aplicación [26].

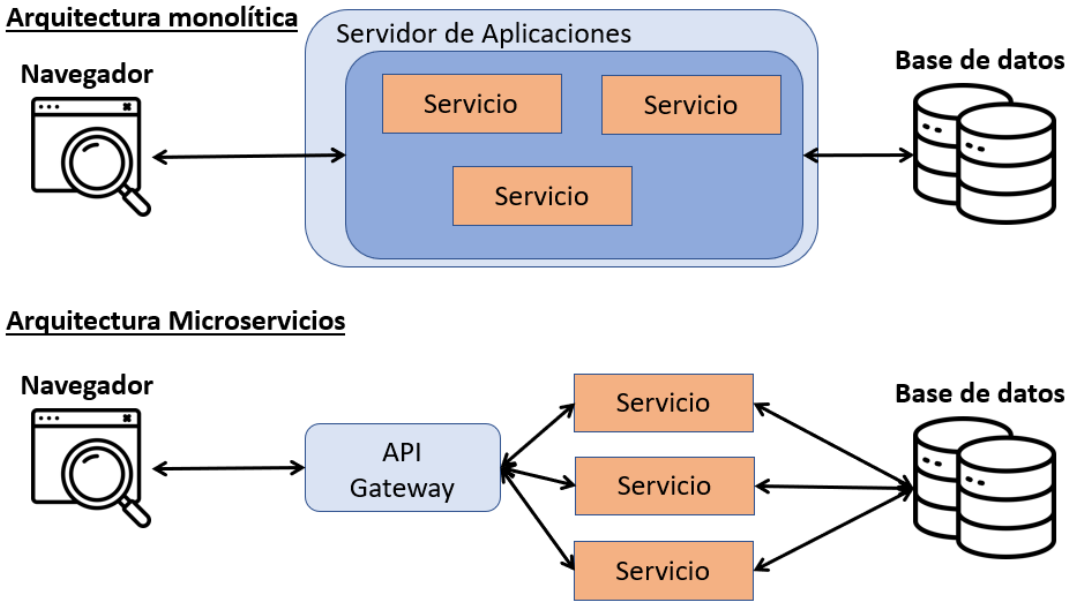

Figura 11. Comparación arquitectura monolítica vs microservicios.

Estas dos arquitecturas son actualmente los dos modelos más utilizados en el desarrollo web. Por un lado, la arquitectura monolítica ha sido la predomínate tradicionalmente hablando. Sin embargo, en los últimos años la arquitectura de microservicios ha tenido un gran auge, debido a su popularidad en el sector y a la aparición de numerosos productos enfocados en esta arquitectura (Docker, Kubernetes etc).

Realizando un análisis comparativo de ambas arquitecturas, podemos definir sus principales diferencias y similitudes:

Tabla 8. Comparativo monolito vs microservicios.

Categoría	***Monolítica***	***Microservicios***
Código	Un único lenguaje en toda la aplicación.	Cada servicio puede tener su propio lenguaje.
Despliegue	Instalación de un único componente. Es necesario parar la aplicación para el despliegue	Instalación de múltiples componentes. El despliegue se puede realizar sin apagar la aplicación
Escalabilidad	Difícilmente escalable, requiere un escalamiento completo de la aplicación	Fácilmente escalable vertical y horizontalmente. Cada servicio puede escalarse de forma individual.

Complejidad	Sencilla si se trata de aplicaciones pequeñas	Elevada si el número de servicios es pequeño. Requiere una configuración de la infraestructura mayor.

Analizando las ventajas e inconvenientes que presenta cada arquitectura, y teniendo en cuenta que este proyecto se enmarca en un ***TFM***, con tiempo y recursos limitados, una de las características que más importancia adquiere es la complejidad. Esta, unida al hecho de que en el momento de lanzamiento del proyecto no se prevé un gran número de usuarios inicial, por lo tanto, no será necesaria una gran escalabilidad, hace que la balanza se decante hacia una arquitectura monolítica.

Como resultado, la definición a alto nivel de la arquitectura que conformará el proyecto puede verse en la Figura 12

Figura 12. Arquitectura de la aplicación.

4.2 Marco tecnológico

Para la implementación de la arquitectura diseñada, es necesaria la utilización de una serie de tecnologías que permitan implementar cada una de las capas de la arquitectura. Utilizando el modelo MVC se plantean claramente tres capas de aplicación:

- Modelo: Capa de acceso a base de datos.
- Vista: Interfaz de usuario.
- Controlador: Lógica de negocio.

Como base para la creación del proyecto se va a utilizar ***JHipster.*** Este se identifica con un generador de proyectos, cuyo objetivo es crear la base para desarrollar una aplicación web, aportando con ello todo lo referente a: creación del proyecto, agregación de librerías y dependencias, configuración, despliegue de la aplicación en un entorno local etc. Dicho de otro modo, ***JHipster*** permite que, cumplimentando un breve cuestionario, indicando las tecnologías que se desea utilizar, se genere un proyecto completo, sobre el que un desarrollador puede comenzar a trabajar, sin preocuparse por toda la fase de creación y configuración del proyecto.

Debido al uso de ***JHipster*** como tecnología de creación del proyecto, existirán ciertas limitaciones en las tecnologías a escoger entra las distintas capas, lo cual se pasa a exponer a continuación.

4.2.1 Interfaz de usuario

Actualmente las dos tecnologías más utilizadas para el desarrollo de proyectos frontend, provistas por ***JHipster*** son ***Angular*** y ***React***. Ambas están basadas en JavaScript y ofrecen funcionalidades similares. Mientras que ***Angular*** ha sido desarrollada por Google, y aporta al programador un framework completo de trabajo para el desarrollo web; ***React*** ha sido desarrollada por Facebook, y se identifica con una librería que necesita de herramientas adicionales para aportar las mismas características que ***Angular***. [27]

Debido a que ***Angular*** ofrece un entorno de trabajo completo, sobre el que será más fácil realizar el desarrollo de una aplicación web compleja, se ha optado por este para el desarrollo del frontend.

Adicionalmente, en conjunto con ***Angular***, se ha optado por añadir ***Bootstrap*** como framework visualización. Este framework aporta a ***Angular*** una capa de visualización extra sobre los elementos mostrados, haciendo que la parte visual sea mucho más llamativa en cuanto a estilos.

4.2.2 Lógica de negocio

Entre los lenguajes de programación más conocidos para el desarrollo del backend, provistos por ***JHipster,*** están: Java, Python, PHP y C/C++. Debido a que el conocimiento de los desarrolladores de este proyecto se centra en ***Java***, y dado que se identifica con uno de los lenguajes más versátiles y portables, se ha optado por su uso en la capa de backend. Adicionalmente, y siguiendo la estructura que provee ***JHipster***, se han añadido los frameworks: ***Spring*** e ***Hibernate***.

Spring es uno de los frameworks más conocidos en el desarrollo web, dado que provee una capa de abstracción sobre ***Java***, estandarizando y simplificando numerosas operativas mediante el uso de etiquetas. Este framework permite a

los desarrolladores simplificar y agilizar los desarrollos, al incluir mediante la adición de etiquetas, operativas que serían difíciles de codificar. [27]

Hibernate es un framework de acceso a datos, que permite estandarizar y simplificar las operativas que se ejercen sobre las bases de datos. Actualmente se ha convertido prácticamente en un estándar para la comunidad. [28]

4.2.3 Acceso a datos

El almacenamiento de los datos en las aplicaciones web suele llevarse a cabo utilizando bases de datos. Existen numerosos tipos de bases de datos: Dinámicas, estáticas, relacionales, no relacionales etc. Sin embargo, siguiendo la estructura que provee ***JHipster,*** se ha optado por utilizar ***PostgreSQL.***

PostgreSQL es un sistema de gestión de bases de datos relacional de código abierto, el cual ofrece una muy buena aplicación de escritorio para la administración y mantenimiento de la base de datos. Una de sus principales características es que no requiere del uso de bloqueos de lectura al realizar transacciones, lo que aporta una gran escalabilidad. [30]

Además, siguiendo la estructura de ***JHipster,*** se utilizará ***Liquibase*** como librería de mantenimiento del modelo de base de datos. Esta librería está más enfocada al uso por parte de los desarrolladores, dado que aporta mecanismos para controlar y añadir cambios en el modelo de base de datos de forma simple y organizada. [31]

Con todo ello, en la Figura 13 se muestra el diseño de la arquitectura a implementar, indicando las tecnologías utilizadas en cada capa.

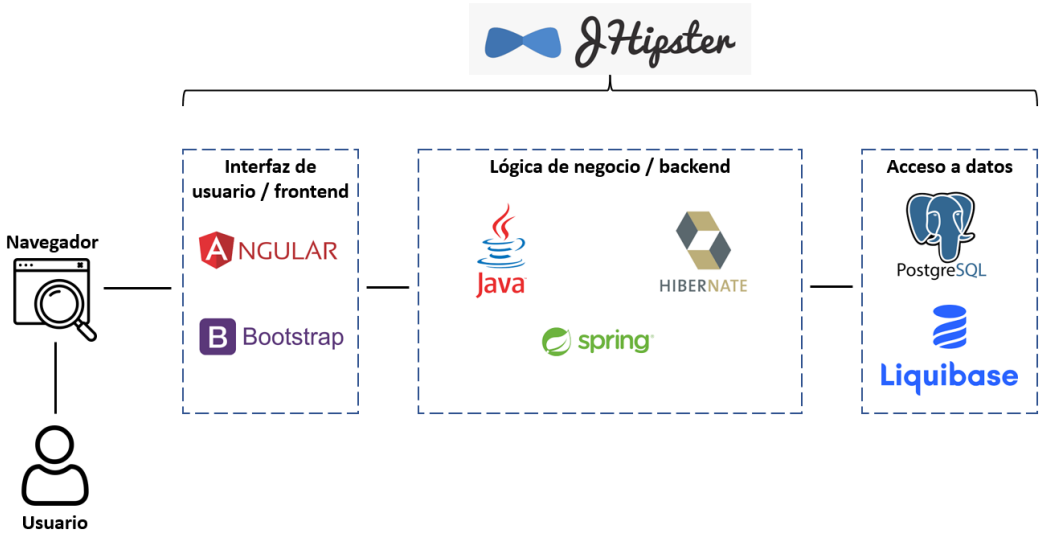

Figura 13. Diagrama de tecnologías utilizadas.

4.3 Interfaz de comunicación

El modelo ***MVC*** definido implementa implícitamente una arquitectura del estilo cliente-servidor. Este diseño es uno de los más comunes y utilizados en los últimos tiempos, debido a la versatilidad que ofrece en el entorno del desarrollo de aplicaciones web.

Entendemos el modelo cliente-servidor como una arquitectura en la que el servidor es un elemento centralizado, al que múltiples clientes pueden acceder realizando peticiones. Mientras que el servidor (o sus réplicas en caso de alta disponibilidad) es un elemento único, los clientes son elementos múltiples que pueden identificarse con distintos tipos de tecnologías (móviles, tablets, ordenadores etc). Debido a esta variedad de clientes, se hace necesario un mecanismo de comunicación cliente-servidor.

Este mecanismo de comunicación cliente-servidor es conocido como ***API*** (Application Programming Interface). Una ***API*** se puede entender como la definición del protocolo y estructura de comunicación entre dos aplicaciones. Las ***API*** definen que información se va a ofrecer y mediante que mecanismo se va a poder consultar [27]. A lo largo del tiempo han surgido diferentes propuestas sobre como implementar una ***API***, siendo las más utilizadas actualmente ***SOAP*** y ***REST***.

SOAP es un estándar creado con el fin de interconectar aplicaciones empresariales diseñadas con diferentes lenguajes de programación. La definición del protocolo ***SOAP*** ofrece una serie de reglas a la hora de establecer la comunicación entre aplicaciones, tales como la estandarización de los tokens identificatorios de los paquetes o el control de errores. Las comunicaciones ***SOAP*** pueden realizarse utilizando distintos protocolos como: ***HTTP***, ***SMTP*** o ***TCP***, y el intercambio de mensajes debe realizarse en formato ***XML***.

REST es un conjunto de principios arquitectónicos, por lo que no se identifica con un estándar. ***REST*** está diseñado para adaptarse a las necesidades de las aplicaciones móviles y servicios web. La comunicación entre los clientes y servidores se realiza utilizando el protocolo ***HTTP***, y como formatos de mensaje válidos encontramos: ***HTML***, ***XML*** y ***JSON*** [28].

Comparando ambas propuestas, ***SOAP*** ofrece un servicio más robusto y seguro, pero más pesado y difícil de implementar. Mientras que ***REST*** ofrece un servicio mucho más ligero y fácil de implementar, lo que lo hace perfecto para el desarrollo de aplicaciones web, siendo por tanto la opción escogida para este proyecto.

4.4 Base de datos

Una parte importante del diseño de muchas aplicaciones es el diseño de las entidades de base de datos y sus relaciones. Para la realización de este proyecto

se han definido un conjunto de tablas, las cuales almacenarán los datos necesarios para solventar todos los casos de uso planteados.

Por un lado, tenemos las entidades relacionadas con los usuarios, cuyo esquema puede verse en el Anexo 2. Para la definición del usuario *Dueño* se generará la tabla ***owner***, la cual contiene datos de identificación del usuario (nombre, apellidos, correo etc). Otra de las entidades definidas es la tabla ***pet***, la cual contiene la información genérica de registro de una mascota (nombre, número de identificación, tipo de mascota, etc). Cada *Dueño* podrá tener una o varias mascotas, por lo que existe una relación one-to-many entre las tablas ***owner*** y ***pet***. Por otro lado, existe la entidad ***service_user***, la cual será la encargada de almacenar los datos de registro de un usuario del tipo Servicio, estos datos contendrán: nombre de la empresa, localización, email, etc. Cada empresa registrada podrá dar de alta uno o varios servicios, los cuales se almacenarán en la tabla ***cuma_service***, que contendrá datos específicos del servicio (datos de contacto, horario, localización etc). Las tablas ***service_user*** y ***cuma_service*** mantendrán una relación one-to-many.

Por otra parte, la aplicación permite el registro de cuatro tipos de mascotas, cada una con sus peculiaridades, por lo que la tabla ***pet*** deberá tener especializaciones. Es por ello que se han diseñado las tablas: ***dog, bird, fish y cat***, Las cuales contienen datos identificativos específicos para cada tipo de animal, como pueden ser: pelaje, color, raza, tipo de agua del hábitat etc. Los detalles de cada una de estas tablas pueden verse en el Anexo 2.

Cada una de las tablas de especialización posee una relación one-to-one con la tabla ***pet***, dado que una mascota solo puede ser de un tipo específico. Además, dado que la aplicación debe proveer de un historial específico para cada tipo de mascota (**R2.4**), cada especialización de una mascota tendrá una relación one-to-many con una entidad de históricos específica para cada mascota (***bird_history, cat_history, dog_history, fish_history***). Las tablas de históricos contendrán los datos definidos en el caso de uso **R2.4**

Por último, existirán las entidades: ***food, treatements, advices*** y ***reminders***, las cuales se utilizarán para almacenar los datos necesarios para la gestión de comidas, tratamientos veterinarios, consejos y recordatorios, estas pueden verse en el Anexo 3.

5 Implementación

Para cumplimentar los requisitos definidos para este proyecto, la implementación se ha basado en los estándares del ***MVC***, definido en el apartado 4.1. Los estándares de este modelo definen una serie de buenas prácticas de estructuración y codificación de clases, las cuales se indicarán a continuación.

5.1 Capa de acceso a base de datos

Cualquier aplicación que utilice un lenguaje orientado a objetos ***POO***, y que requiera de la utilización de una base de datos, deberá implementar un mecanismo que conecte las entidades de su base de datos, con los objetos de su aplicación, este mecanismo es comúnmente conocido como ***ORM*** (mapeador relacional de objetos). Un ***ORM*** es el encargado de realizar las operaciones ***CRUD*** (Create, Read, Update, Delete) necesarias sobre la base de datos, y generar con ello los objetos requeridos por la aplicación [29].

Existen distintos tipos de ***ORM*** dependiendo del lenguaje utilizado, para el caso de Java en 2006 se definió ***JPA***. Este es un estándar, que define una serie de indicaciones sobre como elaborar un ***ORM*** para el caso específico de Java. Entre las indicaciones que aporta, está la nomenclatura de interfaces que se debe de utilizar para realizar el acceso a base de datos. Cabe destacar que ***JPA*** es simplemente la definición del estándar, cayendo la responsabilidad de su implementación sobre cada proyecto [30].

Dado que la implementación de ***JPA*** puede llegar a resultar tediosa, existen varios productos que ofrecen una capa de abstracción sobre esta. Como se expuso en el apartado 4.2, para este proyecto se utilizará **Hibernate**. Cabe destacar que, dado que en este proyecto se ha optado por el uso de **Spring**, el acceso a base de datos se realizará utilizando **Spring JPA**, el cuál es una capa de abstracción adicional sobre **Hibernate**.

Un ejemplo concreto de uso del **Spring JPA** puede verse en la Figura 14 y Figura 15. En la primera figura podemos ver la definición del ***DAO***, entendido como la clase que englobará las operaciones sobre base de datos para una determinada tabla. Una de las características de **Spring JPA** es el uso de la etiqueta `@Respository`, lo que identifica a la clase como un ***DAO*** dentro de Spring. Este tipo de clases deben extender la clase JpaRepository, con el fin de heredar y poder utilizar los métodos ***CRUD*** que implementa **Hibernate.**

```java
/**
 * Spring Data SQL repository for the Dog entity.
 */
@SuppressWarnings("unused")
@Repository
public interface DogRepository extends JpaRepository<Dog, Long> {
    List<Dog> findByPet_id(Long id);
}
```

Figura 14. Ejemplo de DAO.

```java
@Entity
@Table(name = "dog")
public class Dog implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "sequenceGenerator")
    @SequenceGenerator(name = "sequenceGenerator")
    @Column(name = "id")
    private Long id;

    @NotNull
    @Column(name = "i_d_dog", nullable = false, unique = true)
    private Integer iDDog;

    @Column(name = "breed")
    private String breed;

    @Column(name = "height")
    private Double height;

    @Column(name = "weight")
    private Double weight;

    @Column(name = "fur")
    private String fur;

    public Long getId() {
        return this.id;
    }
    public Dog id(Long id) {
        this.setId(id);
        return this;
    }
    public void setId(Long id) {
        this.id = id;
    }
}
```

Figura 15. Ejemplo entidad de acceso a base de datos.

La Figura 15, muestra la definición del objeto entidad de la tabla dog. Los objetos entidad son representaciones de una tabla de la base de datos. Estos objetos deben contener la etiqueta `@Entity`, y `@Table(name = tabla)`, indicando la tabla que mapean. A su vez, deben de definir tantas variables como capos tenga la tabla a mapear, siendo precedida la definición de cada variable por la etiqueta `@Column(…)`, indicando la columna a mapear.

Para la implementación de este proyecto, utilizaremos el mecanismo descrito anteriormente para mapear todas y cada una de las entidades de base de datos definidas en el apartado 4.4. Se generará por tanto un objeto entidad, por cada

una de las tablas definidas, así como un objeto ***DAO*** que incluya las consultas necesarias para el funcionamiento de la aplicación.

5.2 Capa de servicios.

A la hora de implementar el back de una aplicación, es recomendable seguir un diseño que desacople cada una de las capas de la aplicación, con el fin de facilitar posibles cambios y actualizaciones futuras. Es por ello, que a la hora de consumir los datos y métodos ofrecidos por la capa de acceso a base de datos (apartado 5.1), es recomendable implementar una capa de servicios.

Para desarrollar una capa de servicios es imprescindible comenzar implementando una interfaz. Esta interfaz se puede entender como la definición de métodos que estarán disponibles, definiendo a su vez parámetros y valores de vueltos. Un ejemplo de interfaz, para acceder al método de acceso a base de datos definido en el apartado anterior puede verse en la Figura 16.

```
/**
 * Service Interface for managing {@link com.tfm.cuma.domain.Dog}.
 */
public interface DogService {

    /**
     * Get all the entities with pet_id = id.
     *
     * @return the list of entities.
     */
    List<DogDTO> findByPet(Long id);
}
```

Figura 16. Definición de la interfaz de servicio.

Se puede apreciar, que la interfaz es simplemente la definición del “esqueleto” de los métodos a utilizar, definiendo únicamente: nombre del método, valore devuelto y parámetros de entrada.

Una vez definida la interfaz, es necesario implementarla, para ello se crea la implementación de la interfaz, la cual debe incluir la implementación de todos los métodos definidos en la interfaz. Un ejemplo de la implementación de la interfaz definida en la Figura 16, es la implementación mostrada en la Figura 17. Podemos ver como se implementa el método definido en la interfaz, utilizando para ello el método `findByPet_id(id)`, el cual pertenece al ***DAO*** de acceso a base de datos definido en la Figura 14.

Podemos ver que, una vez consultada la capa de acceso a base de datos, se devuelve una lista de objetos entidad de base de datos, como los mostrados en la Figura 15. La implementación del servicio convierte estos objetos a entidades ***DTO*** del tipo `DogDTO`, utilizando para ello el mapeador entidad-DTO `dogMapper`. La definición de mapeadores es muy sencilla de implementar si se utiliza Spring. En concreto, la implementación del mapeador utilizado en este ejemplo puede verse en la Figura 18. En los siguientes puntos veremos la importancia de realizar este mapeo entre objetos.

```java
/**
 * Service Implementation for managing {@link Dog}.
 */
@Service
@Transactional
public class DogServiceImpl implements DogService {

    private final Logger log = LoggerFactory.getLogger(DogServiceImpl.class);
    private final DogRepository dogRepository;
    private final DogMapper dogMapper;

    @Override
    @Transactional(readOnly = true)
    public List<DogDTO> findByPet(Long id) {
        log.debug("Request to get dog via pet_id");
        List<Dog> history = dogRepository.findByPet_id(id);
        List<DogDTO> foodDTO = new ArrayList<>();
        for (Dog food : history) {
            foodDTO.add(dogMapper.toDto(food));
        }
        return foodDTO;
    }
}
```

Figura 17. Implementación de interfaz de servicio.

```java
/**
 * Mapper for the entity {@link Dog} and its DTO {@link DogDTO}.
 */
@Mapper(componentModel = "spring")
public interface DogMapper extends EntityMapper<DogDTO, Dog> {
    @Mapping(target = "pet", source = "pet", qualifiedByName = "petId")
    DogDTO toDto(Dog s);

    @Named("petId")
    @BeanMapping(ignoreByDefault = true)
    @Mapping(target = "id", source = "id")
    PetDTO toDtoPetId(Pet pet);
}
```

Figura 18. Ejemplo de mapeador entidad-DTO.

Para la realización de este proyecto, implementaremos una capa completa de servicio, la cual consuma los métodos ofrecidos por la capa de acceso a base de datos, de la forma que se ha indicado en este apartado.

5.3 Capa de comunicación API REST en el servidor

Para implementar el diseño de la interfaz de comunicación en el lado servidor, definido en el apartado 4.3, se definirán una serie de endpoints tipo ***REST***, los cuales podrán ser consultados por las aplicaciones cliente.

Estos endpoints se implementarán utilizando Spring boot, el cual ofrece un alto nivel de abstracción en la creación de endpoints. Un ejemplo de ello puede verse en la Figura 19, en la que se define un endpoint para solicitar un listado de objetos del tipo Dog.

Para crear un endpoint con Spring Boot es necesario comenzar la definición del método con una etiqueta que identifique el tipo de operación ***HTTP*** requerida (`@GetMapping`, `@PostMapping`, `@PutMapping`, `@DeleteMapping`, `@PatchMapping`. [31]) En este caso, dado que se quiere solicitar una información, se utiliza el método get y su etiqueta `@GetMapping`, a la que se le indica la ruta (path) que recibirá el endpoint, en este caso *"/dogs"*. La implementación del endpoint prosigue con la llamada al método **findAll** del repositorio, el cual devolverá el listado de objetos requeridos. Por último, se compone una respuesta ***HTTP***, indicando que la operación ha sido satisfactoria (`ResponseEntity.ok()`), e introduciendo en el body de la respuesta el listado de objetos requeridos.

```
/**
 * {@code GET  /dogs} : get all the dogs.
 *
 * @param pageable the pagination information.
 * @return the {@link ResponseEntity} with status {@code 200 (OK)} and the list of dogs in body.
 */
@GetMapping("/dogs")
public ResponseEntity<List<DogDTO>> getAllDogs(
    @org.springdoc.api.annotations.ParameterObject Pageable pageable) {
        log.debug("REST request to get a page of Dogs");
        Page<DogDTO> page = dogService.findAll(pageable);
        HttpHeaders headers = PaginationUtil.generatePaginationHttpHeaders(
            ServletUriComponentsBuilder.fromCurrentRequest(), page);
        return ResponseEntity.ok().headers(headers).body(page.getContent());
}
```

Figura 19. Ejemplo de endpoint tipo REST.

Cabe destacar que el listado de objetos devueltos por el endpoint está compuesto por objetos del tipo ***DTO*** (`ResponseEntity<List<DogDTO>>`). Los objetos ***DTO*** son objetos utilizados para transmitir información entre los clientes y servidores en arquitecturas ***REST***. Estos objetos deben ser serializarles, para su correcto envío a través de la red, y de tipo solo lectura, para aumentar la seguridad de la aplicación.

Se considera una buena práctica de programación, no utilizar los objetos entidad de acceso a base de datos (definidos en el apartado 5.1), a la hora de realizar envíos al cliente. Por un lado, esto es debido a que los objetos entidad pueden contener más información de la que el cliente necesita. Por otro lado, la definición de los objetos entidad se diseña con el fin de adaptar los tipos de las variables, a los campos de base de datos, haciendo que a veces sea complicado serializar los tipos de dato escogidos. Es por ello por lo que surge la necesidad de realizar un mapeo de objetos entidad a ***DTO***, a la hora de elaborar una respuesta ***HTTP*** hacia un cliente.

Para la realización de este proyecto, se definirán un conjunto de endpoints, siguiendo la estructura anteriormente indicada. Estos endpoints serán consumidos por los clientes para obtener los datos necesarios para la visualización y operativa de la aplicación.

5.4 Consumo de API REST en los clientes

Una vez definida la API de endpoint en el back, es posible hacer uso de estos endpoints desde distintos clientes. Para la realización de este proyecto se ha

optado por el desarrollo de una aplicación web utilizando ***Angular***, por lo que las tecnologías y mecanismos de consulta de endpoints deberán estar basados en estas tecnologías.

En la Figura 20 se puede apreciar un ejemplo de consulta a un endpoint utilizando ***Angular***. Los primero que se debe hacer es definir una clase servicio en lenguaje TypeScript, en este caso `DogService`. Lo segundo, definir la URL sobre la que se desea hacer la consulta ***HTTP***. Esta ruta deberá coincidir con el path con el que se creó el endpoint a consultar. Una vez definida la ruta a consultar, se define un método de tipo `Observable`, que realizará la consulta e informará al cliente cuando le llegue un resultado, en este caso el método es `find(id)`.

Una vez definido el método, se utilizan la librería `http` de ***Angular*** para realizar una consulta ***HTTP***, en este caso un get, sobre el path requerido. La respuesta se mapeará automáticamente a un objeto del tipo `IDog`, el cual devolverá el método tras su finalización.

```
@Injectable({ providedIn: 'root' })
export class DogService {
  protected resourceUrl = this.applicationConfigService.getEndpointFor('api/dogs');

  constructor(protected http: HttpClient, protected applicationConfigService: ApplicationConfigService) {}

  find(id: number): Observable<EntityResponseType> {
    return this.http.get<IDog>(`${this.resourceUrl}/${id}`, { observe: 'response' });
  }

}
```

Figura 20. Ejemplo de consulta de endpoint desde el front.

Lo más importante de este mecanismo es el cumplimiento del "contrato" que impone la ***API***. En el apartado 5.3 se indicaba, que en la definición del endpoint se debía indicar, por un lado, la ruta sobre la que este debe ser consultado, y por otro el objeto ***DTO*** que el endpoint devuelve tras su llamada. A la hora de consultar un endpoint desde el front es importante respetar la configuración impuesta en el back.

Respetar el contrato de la ***API*** implica, por un lado, que la consulta desde el front debe hacerse sobre la misma ruta que fue definida en el back. Por otro lado, la implementación del front debe disponer de un objeto gemelo al ***DTO*** definido como respuesta en el back. De esta forma, se podrá deserializar el ***DTO***, y mapearlo a un objeto con sus mismos atributos en el front. Si esto no se cumpliera, el front obtendría objetos con información corrupta.

Los objetos gemelos a los ***DTO*** del back, en el front suelen denominarse modelos, y siguen una estructura parecida a la mostrada en la Figura 21

```
export class Dog implements IDog {
  constructor(
    public id?: number,
    public iDDog?: number,
    public breed?: string | null,
    public height?: number | null,
    public weight?: number | null,
    public fur?: string | null,
    public histories?: IDogHistory[] | null,
    public pet?: IPet | null
  ) {}
}
```

Figura 21. Ejemplo de modelo de objeto en el front.

Una vez el front ha generado un modelo con los datos que desea representar, este podrá ser representado en una página html en el navegador, generando con ello una página web.

5.5 Seguridad en la aplicación.

Dado que el producto diseñado es una aplicación web multiusuario, es importante durante la implementación elaborar mecanismos de seguridad que impidan que información de un usuario pueda ser visualizada por otro. Además, es importante implementar mecanismos de seguridad que eviten ataques malintencionados sobre la aplicación.

Para la implementación de estos mecanismos de seguridad se han definido tres roles dentro de la aplicación:

- **ROLE_OWNER**: Permite acceder a los endpoint y ventanas destinadas al usuario con perfil *Dueño*.
- **ROLE_SERVICE**: Permite acceder a los endpoint y ventanas destinadas al usuario con perfil Servicio.
- **ROLE_ADMIN**: Permite realizar labores de mantenimiento y monitorización sobre la aplicación, acceder a la gestión de usuarios y generar consejos.

Durante el proceso de registro de un usuario, apartado 6.1, la aplicación asignará el rol correspondiente al usuario generado. De forma que, entre la información de usuario almacenada en base de datos, se encontrará el rol.

5.5.1 Ciclo de vida JWT

La implementación de los mecanismos de seguridad se realizará utilizando ***JWT***. Este es un mecanismo de que permite compartir entre dos partes, de forma segura, la identidad de un determinado usuario. Normalmente estas dos partes se identifican con el ***back*** y el ***front***. El ciclo de vida y funcionamiento de ***JWT*** puede verse en la Figura 22. Este comienza con el login de un usuario, el cual, en caso de ser satisfactorio, produce la generación en el ***back*** de un ***JWT*** que

contiene la información de usuario y roles (secreto). El token es devuelto al ***front***, el cual lo almacena en la sesión del usuario, siendo utilizado en la cabecera de las peticiones ***HTTP*** sucesivas, para que el ***back*** pueda determinar la identidad del usuario que realiza la petición.

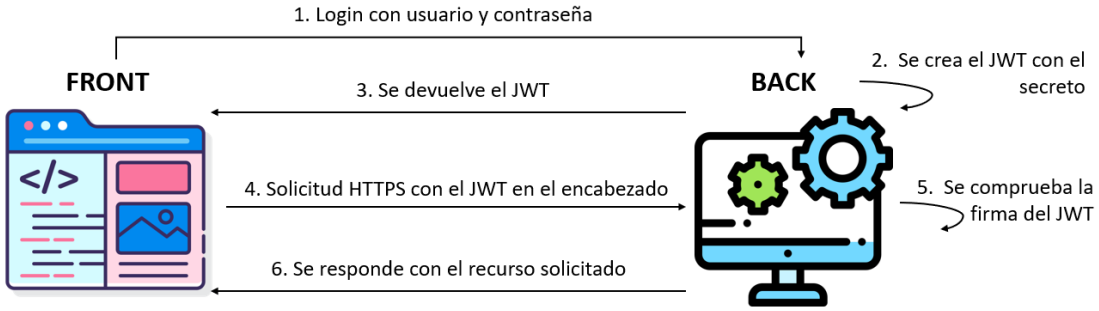

Figura 22. Ciclo de vida de un token JWT.

5.5.2 Composición de un token JWT

En la práctica, un token ***JWT*** es una cadena de texto codificada en Base64. Un ejemplo de un token codificado de la aplicación puede verse en la parte izquierda de la Figura 23. El token está compuesto por tres secciones separadas por un punto:

- **Header**: encabezado donde se identifica el algoritmo utilizado, en este caso HS512.
- **Payload**: donde se almacenan los datos de usuario y privilegios, en este caso usuario y ROLE_OWNER.
- **Signature**: Firma que permite identificar si el token es válido o ha sido modificado.

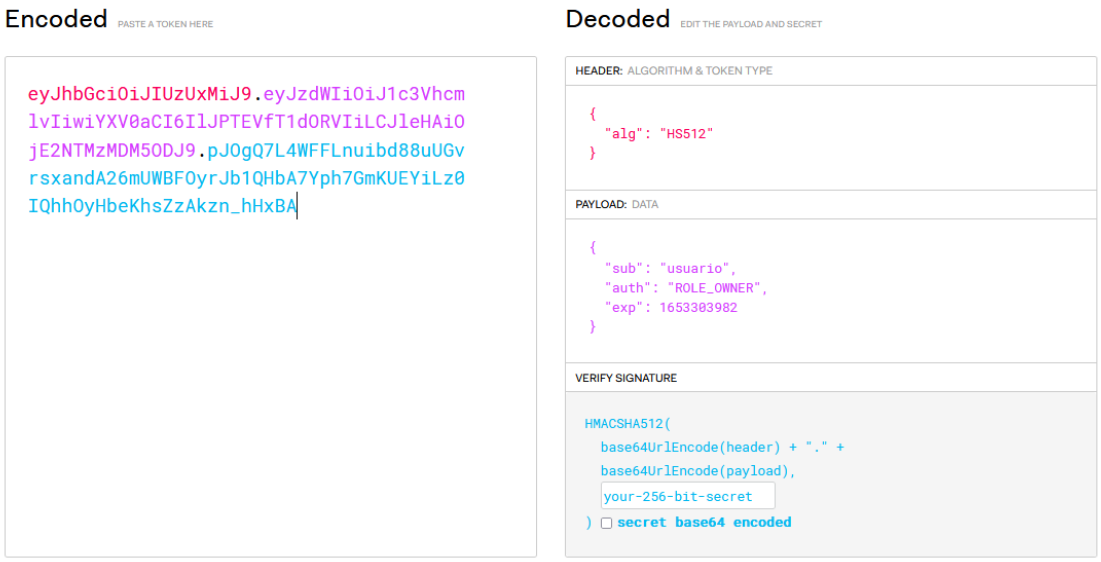

Figura 23. Ejemplo de token JWT de la aplicación.

Si observamos la parte derecha de la Figura 23, puede verse el contenido del token decodificado. Esto es debido a que el token de por si no está cifrado, por lo que su contenido puede ser visto por cualquiera. La firma del token está basada en una clave compartida entre el ***back*** y el ***front***. Esta firma asegura simplemente que el token no ha sido modificado por el camino. Para añadir una capa de encriptado en la transmisión del token se deben utilizar mecanismos de transferencia encriptada como HTTPS.

5.5.3 JWT en el front

Para limitar la navegación web a los usuarios que contengan un determinado rol, se utilizarán el elemento `RouterModule` de ***Angular***. Esta clase permite configurar la navegación a través de la aplicación. Un ejemplo resumido de la aplicación puede verse en la Figura 24, donde se definen las rutas: login, menu-owner y menu-service, indicando mediante el atributo `authorities`, que roles pueden acceder a cada una:

- login/*: cualquier rol.
- menu-owner/*: usuarios con el rol **ROLE_OWNER.**
- menu-service/*: usuarios con el rol **ROLE_SERVICE.**

La información sobre el rol de un determinado usuario se obtiene del ***JWT*** guardado en sesión para ese usuario una vez hecho el login, tal y como se explicó en el apartado 5.5.1

```
@NgModule({
  imports: [
    RouterModule.forRoot(
      [
        {
          path: 'login',
          loadChildren: () => import('./login/login.module').then(m => m.LoginModule),
        },
        {
          path: 'menu-owner',
          data: {
            authorities: [Authority.OWNER],
          },
          canActivate: [UserRouteAccessService],
          loadChildren: () => import('./menu-owner/menu-owner.module').then(m => m.MenuOwnerModule),
        },
        {
          path: 'menu-service',
          data: {
            authorities: [Authority.CUMASERVICE],
          },
          canActivate: [UserRouteAccessService],
          loadChildren: () => import('./menu-service/menu-service.module').then(m => m.MenuServiceModule),
        }
        navbarRoute,
        ...errorRoute,
      ],
      { enableTracing: DEBUG_INFO_ENABLED }
    ),
  ],
  exports: [RouterModule],
})
export class AppRoutingModule {}
```

Figura 24. Ejemplo de routing en el front de la aplicación.

5.5.4 JWT en el back, autenticación de roles

En términos de seguridad, en el ***back*** de la aplicación se comprueba, por un lado, que las peticiones ***HTTP*** recibidas han sido realizadas por un usuario con el rol adecuado, así como que no se suplanta la identidad de ningún usuario.

Para la protección de los endpoints definidos en la ***API,*** se utiliza **Spring** Security, el cual es el marco de autenticación y autorización de **Spring.** Para utilizar **Spring** Security es necesario definir una clase con la etiqueta `@EnableWebSecurity`, en nuestro caso será la clase `SecurityConfiguration.java` [32].

Para la implementación de este proyecto se utilizan los filtros de cadena de **Spring** Security, dado que son un mecanismo efectivo y fácil de escalar. Para ello, en la clase `SecurityConfiguration.java`, se definirá el método `filterChain(…)`, el cuál definirá una serie de filtros que se aplicarán sobre las peticiones ***HTTP*** de los clientes antes de redirigirlas a su endpoint correspondiente. En este caso, definiremos una serie de filtros relacionados con el endpoint a consultar y el rol requerido en el ***JWT*** recibido. Un ejemplo resumido de la configuración de este proyecto puede verse en la Figura 25, donde se define en el método `filterChain(…)`, algunos de los endpoint de la aplicación, y que roles tienen acceso a ellos.

```
@EnableWebSecurity
public class SecurityConfiguration {

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        http
            .csrf()
            .disable()
            .addFilterBefore(corsFilter, UsernamePasswordAuthenticationFilter.class)
            .exceptionHandling()
                .authenticationEntryPoint(problemSupport)
                .accessDeniedHandler(problemSupport)
        .and()
            .authorizeRequests()
            .antMatchers("/api/authenticate").permitAll()
            .antMatchers("/api/register").permitAll()
            .antMatchers("/api/account/reset-password/init").permitAll()
            .antMatchers("/api/admin/**").hasAuthority(AuthoritiesConstants.ADMIN)
            .antMatchers("/api/createPet").hasAuthority(AuthoritiesConstants.OWNER)
            .antMatchers("/api/pets/owner").hasAuthority(AuthoritiesConstants.OWNER)
            .antMatchers("/api/petTreatments/**").hasAuthority(AuthoritiesConstants.OWNER)
            .antMatchers("/api/petFood/**").hasAuthority(AuthoritiesConstants.OWNER)
            .antMatchers("/management/**").hasAuthority(AuthoritiesConstants.ADMIN)
        .and()
            .httpBasic()
        .and()
            .apply(securityConfigurerAdapter());
        return http.build();
    }
```

Figura 25. Ejemplo de configuración de Spring Security en el back.

5.5.5 JWT en el back, autenticación de usuarios

Por otro lado, para evitar la suplantación de usuarios en las llamadas a la ***API,*** se requiere de la verificación del usuario en cada endpoint, de forma que, si un usuario está solicitando, por ejemplo, la información de una mascota en base al id de esta, primero se debe comprobar que el id pertenece a una mascota del usuario. Con este procedimiento se evitan ataques del tipo man-in-the-middle, en el que usuarios malintencionados envían peticiones HTTP modificadas, solicitando datos de otros usuarios. Para evitar este tipo de ataques, la ***API*** de la aplicación se ha diseñado de forma que en la petición ***HTTP*** no se envíen datos identificativos del usuario, sino que siempre se utilicen los tokens ***JWT***. De este modo, si por ejemplo un usuario quiere solicitar su listado de mascotas, no debe enviar en la petición a la ***API*** ningún tipo de identificación más que su ***JWT***.

Para realizar esta implementación se utiliza en el ***back*** la clase `SecurityContext` de **Spring**, la cual permite acceder al token ***JWT*** recibido, accediendo con ello a la información del usuario que ha realizado la petición ***HTTP***. Seguidamente, en cada endpoint, se utiliza la información de usuario del token para realizar las peticiones de información. Un ejemplo de ello lo podemos ver en la Figura 26, donde se utiliza el servicio `userService`, que implementa el `SecurityContext`, para acceder a la información de usuario del ***JWT*** , y con ella crear al usuario de aplicación `currentOwner`, en caso de que este exista. Una vez obtenido el usuario, se realiza la petición de mascotas sobre este.

```
@GetMapping("/pets/owner")
public ResponseEntity<List<PetDTO>> getAllOwnerPets() {
    log.debug("REST request to get Pets of an owner");

    AdminUserDTO currentUser = userService
        .getUserWithAuthorities()
        .map(AdminUserDTO::new)
        .orElseThrow(() -> new BadRequestAlertException("user cannot be found", ENTITY_NAME, "idexists"));

    OwnerDTO currentOwner = ownerService.findByOwnerId(Math.toIntExact(currentUser.getId()));
    List<PetDTO> petList = petService.findPetByOwner(Math.toIntExact(currentOwner.getId()));
    return ResponseEntity.ok().body(petList);
}
```

Figura 26. Ejemplo de securización de usuarios con JWT.

5.6 Resultado de la implementación

Como resultado de aplicar los anteriores apartados al desarrollo de la implementación, el producto obtenido muestra una arquitectura como la expuesta en la Figura 27. Este diseño ofrece una implementación desacoplada, en la que cada capa sería fácilmente sustituible. Esto es muy útil a la hora de incluir cambios o actualizaciones tecnológicas, dado que, si por ejemplo se desease cambiar la base de datos utilizada, simplemente se tendría que actualizar la capa de acceso a base de datos, evitando el cambio en el resto de las capas de la estructura.

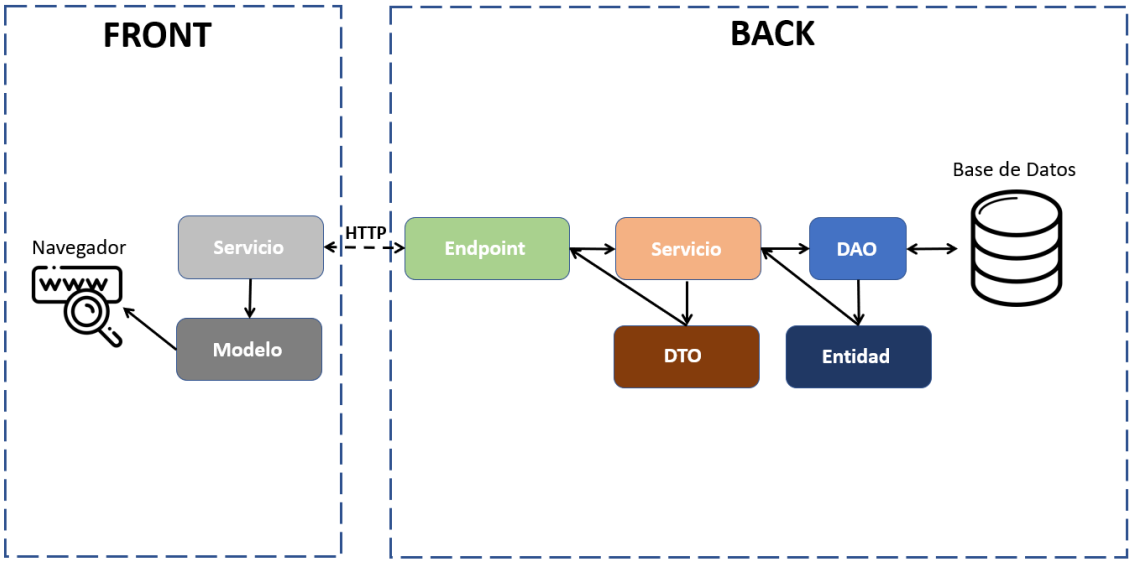

Figura 27. Resultado de la implementación.

6 Resultados

Como resultado de la implementación del proyecto, como producto se ha obtenido una aplicación web, accesible desde el navegador, que implementa todos los casos de uso definidos. La aplicación web dispone de distintas pantallas, cada una de las cuales ofrece una de las funcionalidades requeridas, las cuales se pasan a exponer a continuación:

6.1 Acceso a la aplicación.

En la Figura 28 puede verse la ventana de acceso a la aplicación, esta permite por un lado iniciar sesión si ya se dispone de una cuenta (Figura 29), o registrarse si no se dispone de cuenta (Figura 30 y Figura 31).

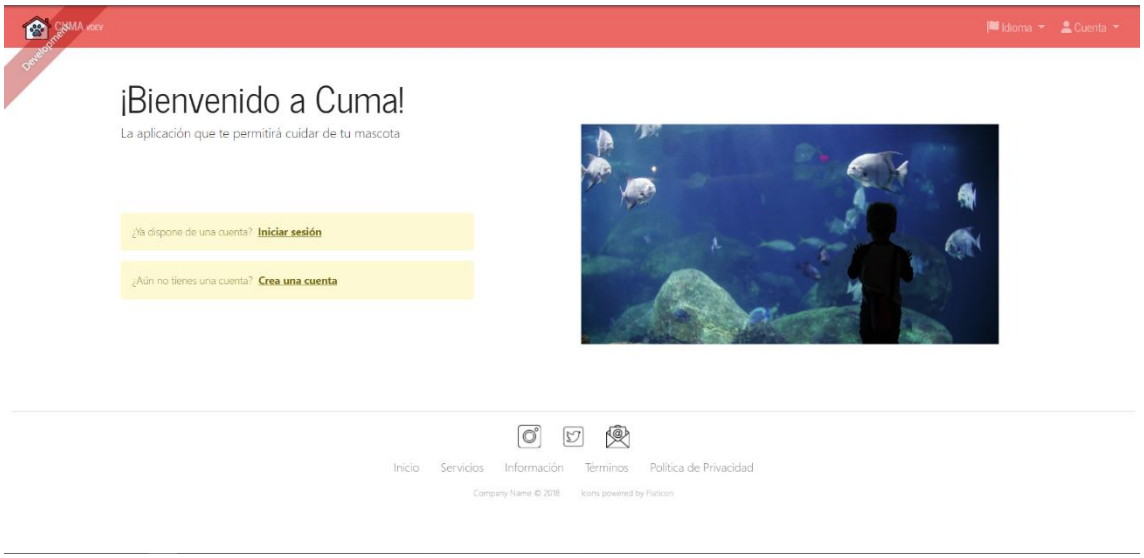

Figura 28. Ventana de acceso a la aplicación.

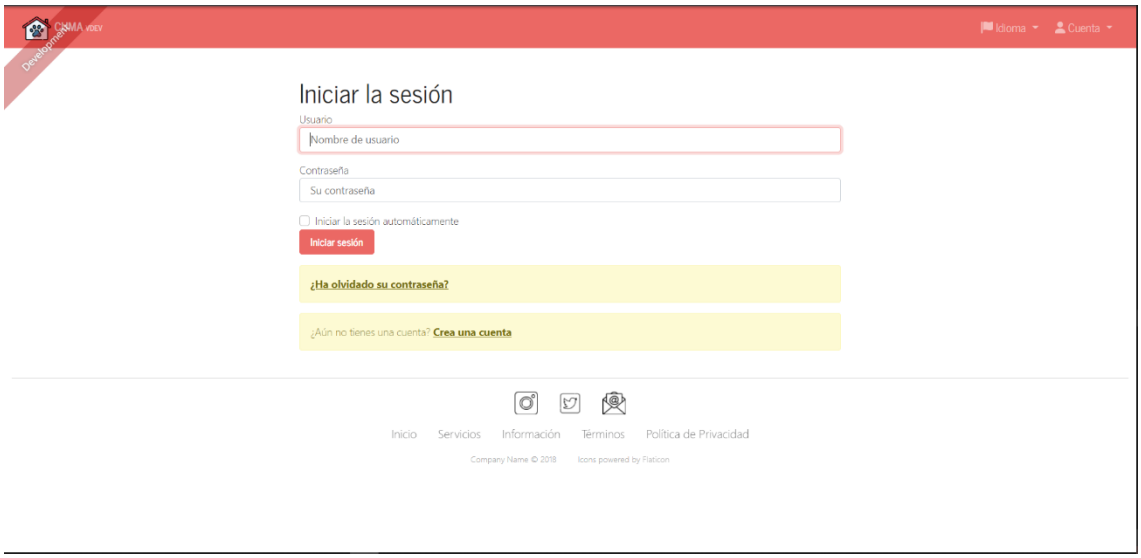

Figura 29. Ventana de inicio de sesión.

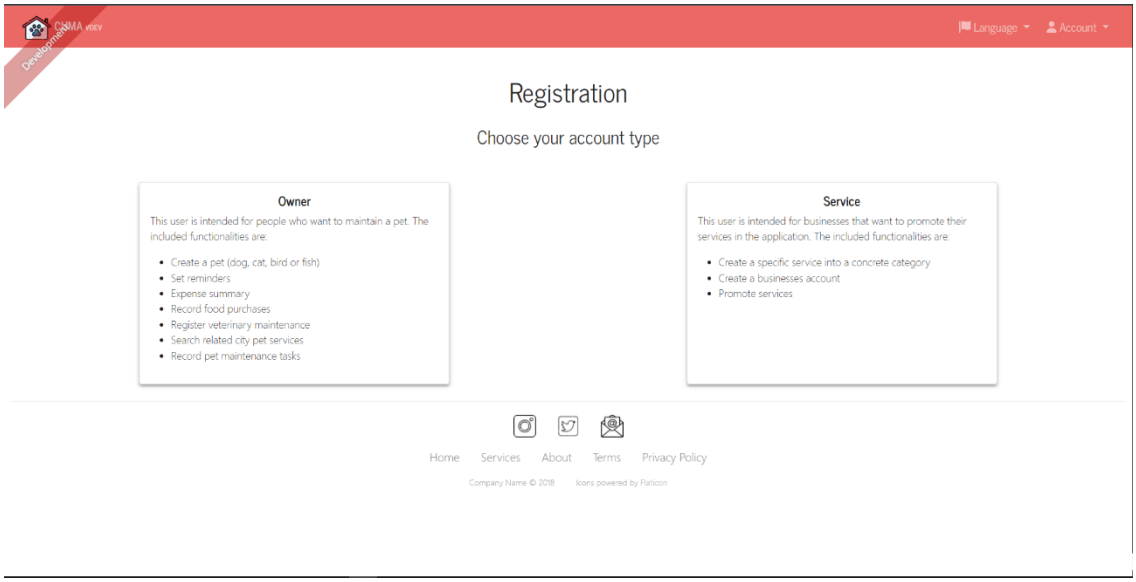

Figura 30. Ventana de creación de usuarios 1.

La creación de usuarios está dividida en dos ventanas, en la primera se nos describen los dos tipos de usuarios existentes (Figura 30), y en la segunda (Figura 31) se introducen los datos del usuario a crear.

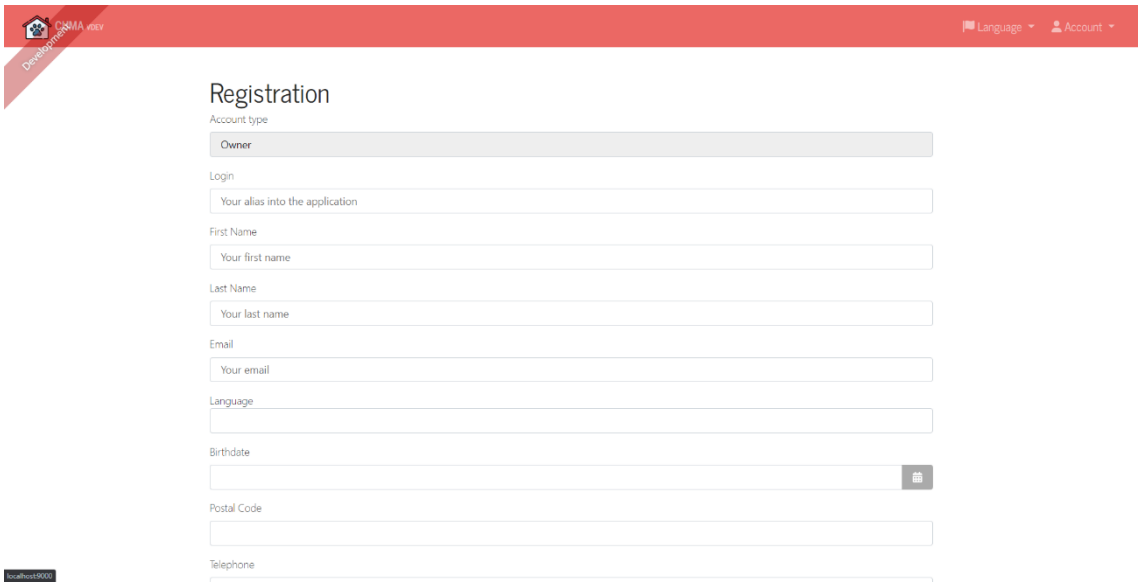

Figura 31. Ventana de creación de usuarios 2

6.2 Menú principal del usuario Dueño.

Una vez un usuario con el perfil *Dueño* ha iniciado sesión en la aplicación, se le mostrará el menú principal del perfil *Dueño*, Figura 32.

Esta ventana permite a un usuario con el perfil *Dueño* acceder a las distintas funcionalidades de las que dispone en la aplicación: registrar una mascota, registrar y visualizar información sobre la mascota, ver consejos publicados, ver servicios disponibles y generar recordatorios.

Cada una de las funcionalidades descritas será accesible mediante cada una de las tarjetas de la interfaz, las cuales se expondrán en los siguientes apartados.

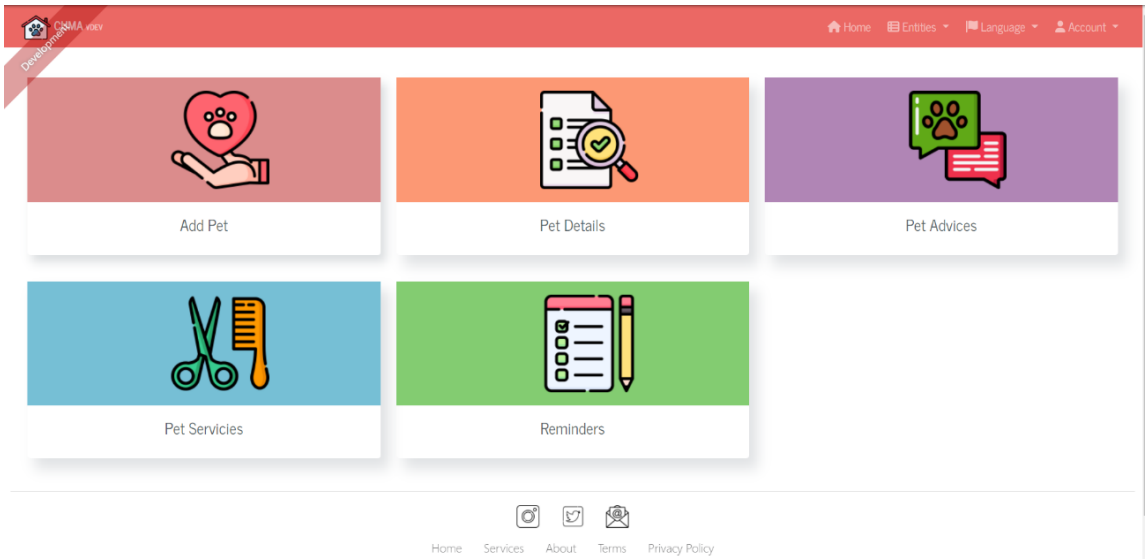

Figura 32. Menú principal del usuario Dueño.

6.2.1 Menú de creación de mascotas.

Tras pulsar el botón "*Add Pet*" de la Figura 32, se abrirá la ventana de creación de mascotas, la cual está compuesta por un formulario con los distintos campos necesarios para la creación de una mascota, Figura 33. Tras completar los datos del formulario, se almacenará la mascota creada en la base de datos, ligando esta al usuario que la ha creado.

Como resultado, la funcionalidad de esta ventana cumplimenta parcialmente el caso de uso **CU1**. Concretamente, se cumplimenta el apartado de creación de mascotas definido en el requisito **R1.1**.

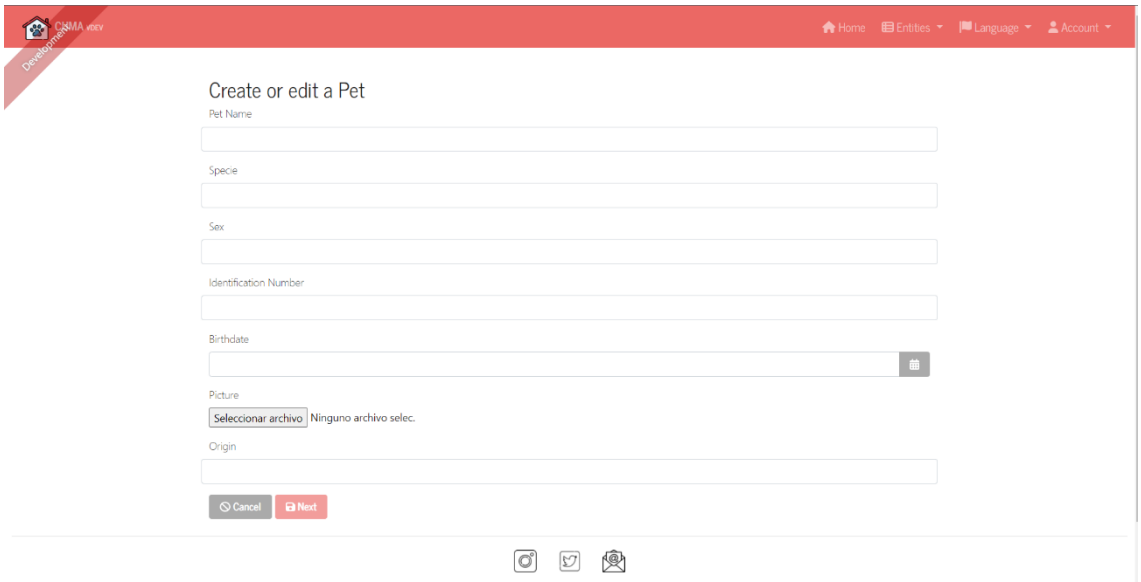

Figura 33. Ventana de creación de mascotas.

6.2.2 Menú de listado y registro de información de mascotas.

Esta ventana será accesible tras pulsar el botón "*Pet details"* de la Figura 32. El contenido de la ventana mostrará el listado de mascotas creadas por el usuario, junto con un resumen de la información sobre cada una (Figura 34).

De esta forma se cumplimenta parcialmente el caso de uso **CU1**. Concretamente, se cumplimenta el apartado de visualización de mascotas definido en el requisito **R1.1**.

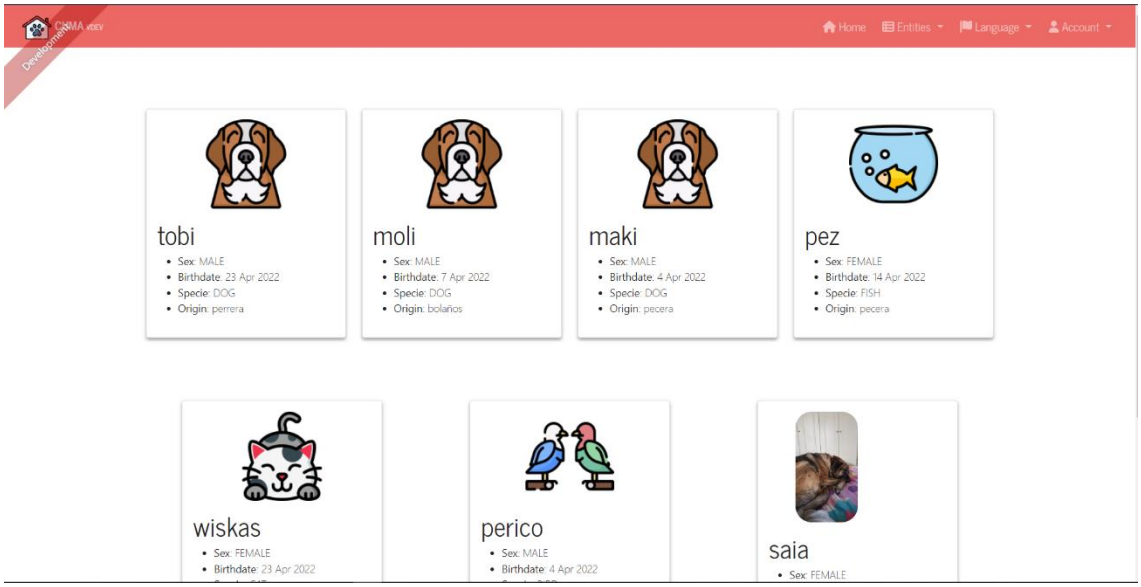

Figura 34. Listado de mascotas, menú de registro.

En esta misma ventana, si se pulsa alguna de las tarjetas de mascotas, se abrirá una nueva ventana mostrando información más específica de la mascota seleccionada.

Entre la información mostrada se encuentra: información de registro, tratamientos veterinarios guardados, registro de comida y registro de cuidados (Figura 35).

Pulsando sobre la tarjeta de cada uno de los apartados se mostrará el histórico de cada uno de ellos referente a la mascota. Para añadir un nuevo registro en cualquiera de las categorías, se deberá pulsar sobre el botón situado en la esquina superior derecha de cada tarjeta. Este redirigirá al usuario a la ventana de creación de cada recurso.

Mediante esta ventana se cumplimenta totalmente el caso de uso **CU1**, dado que se ofrece al usuario la visualización, almacenamiento y actualización de:

- Tratamientos veterinarios: Tarjeta "*Treatements*". Requisito **R1.2**.
- Gestión de comidas: Tarjeta "*Food*". Requisito **R1.3**.
- Gestión de cuidados: Tarjeta "*History*". Requisito **R2.4**

Figura 35. Ventana detalle de mascota.

6.2.3 Menú de visualización de consejos

Esta ventana será accesible tras pulsar el botón "*Pet Advices"* de la Figura 32. El contenido mostrado serán los consejos activos dados de alta en la aplicación por un usuario administrador. Estos consejos contienen varios campos por lo que el usuario puede filtrar como son: especie de mascota a la que va dirigido el consejo, fecha de creación o categoría del consejo.

Mediante esta ventana se ofrece la funcionalidad indicada en el caso de uso **CU6**, concretamente la funcionalidad referida a la visualización de consejos por parte del usuario *Dueño*, cumplimentándose parcialmente el requisito **R2.3**.

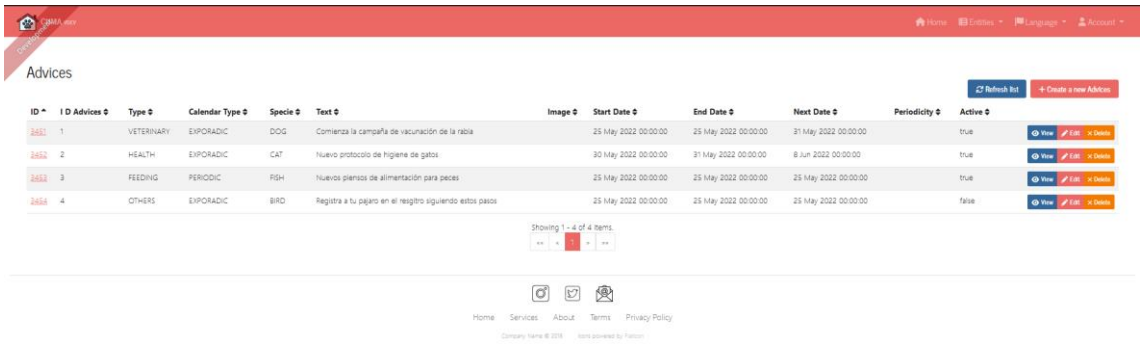

Figura 36. Ventana de visualización de consejos.

6.2.4 Menú de visualización de servicios.

Tras pulsar el botón "*Pet Services"* de la Figura 32, se abrirá la ventana de visualización de servicios, en la que el usuario podrá visualizar los distintos servicios dados de alta en la aplicación, pudiendo filtrar estos por categoría si fuera necesario (Figura 37).

Mediante esta ventana se cumplimenta totalmente el caso de uso **CU4**, ofreciendo la funcionalidad total de visualización de servicios en la aplicación. Con ello se cumplimentan los requisitos: **R3.2** y **R3.3**.

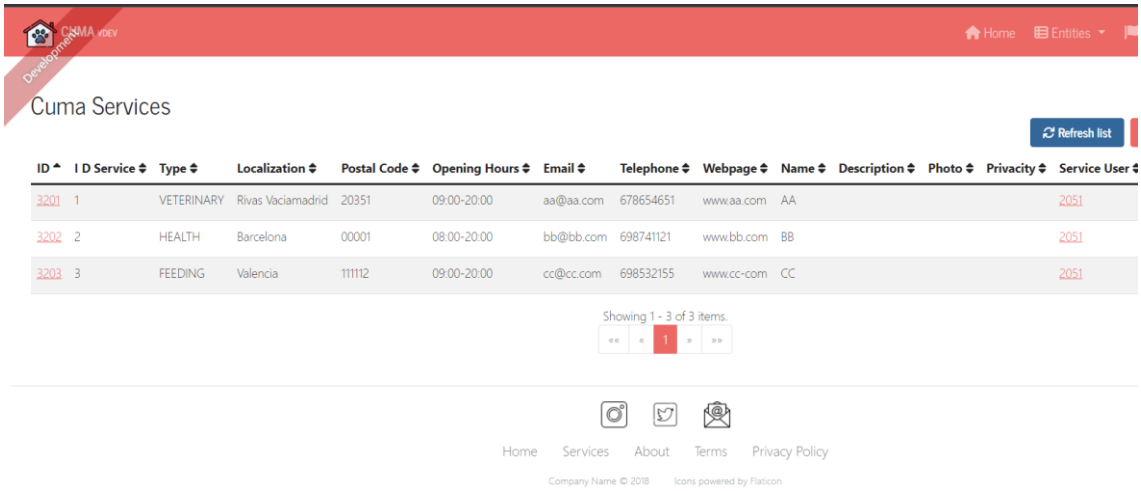

Figura 37. Ventana de visualización de servicios.

6.2.5 Menú de recordatorios de usuario

Esta ventana será accesible tras pulsar el botón "*Reminders"* de la Figura 32. En ella el usuario tendrá la capacidad tanto de crear, como de visualizar los recordatorios que ha dado de alta en la aplicación.

La Figura 38 muestra la ventana de visualización de recordatorios en modo de tabla, dado la posibilidad de editar o eliminar cada recordatorio mediante la botonera lateral de cada fila.

En la parte superior derecha de la ventana aparecen el botón de creación de un nuevo recordatorio, el cual redirigirá al usuario a la ventana mostrada en la Figura 39, donde el usuario podrá completar el formulario de creación de recordatorios.

Mediante esta ventana se cumplimenta totalmente el caso de uso **CU2**, dando cumplimiento al requisito **R2.1**, ofreciendo totalmente la funcionalidad de creación y visualización de recordatorios en la aplicación.

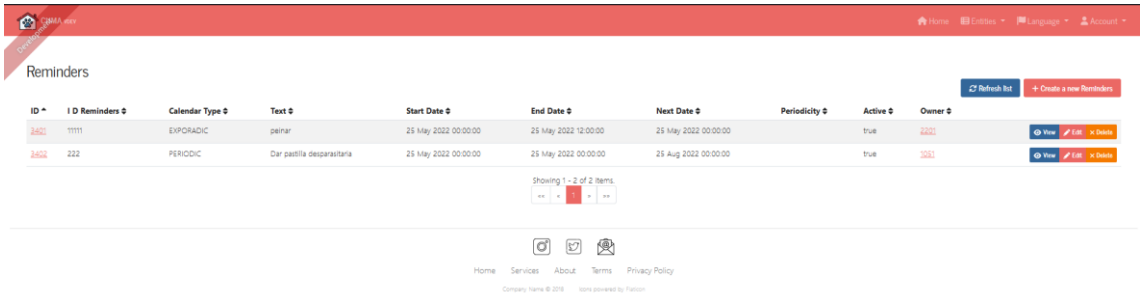

Figura 38. Ventana de visualización de recordatorios.

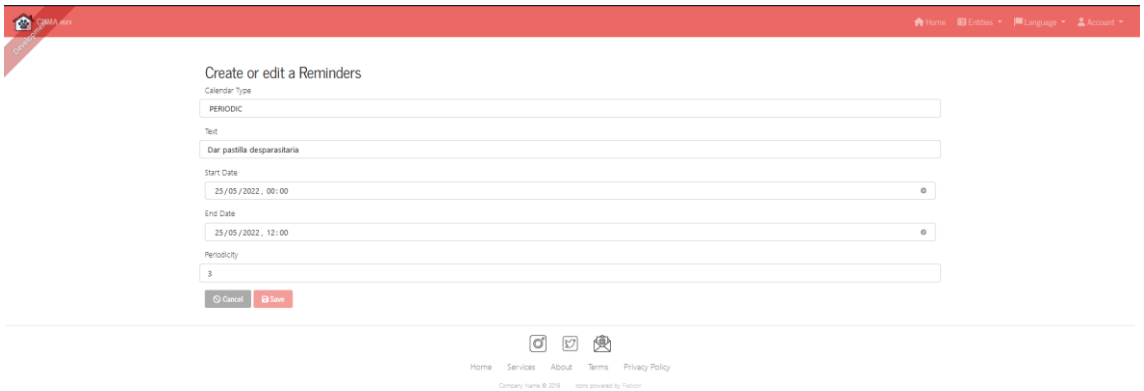

Figura 39. ventana de creación de recordatorios.

6.3 Menú principal del usuario Servicio

Si el acceso a la aplicación ha sido producido por un usuario con el perfil Servicio, el menú de inicio mostrado será el correspondiente al perfil Servicio, Figura 32.

Esta ventana permite a un usuario con el perfil Servicio acceder a las distintas funcionalidades de las que dispone en la aplicación: Creación de servicios y visualización de servicios existentes.

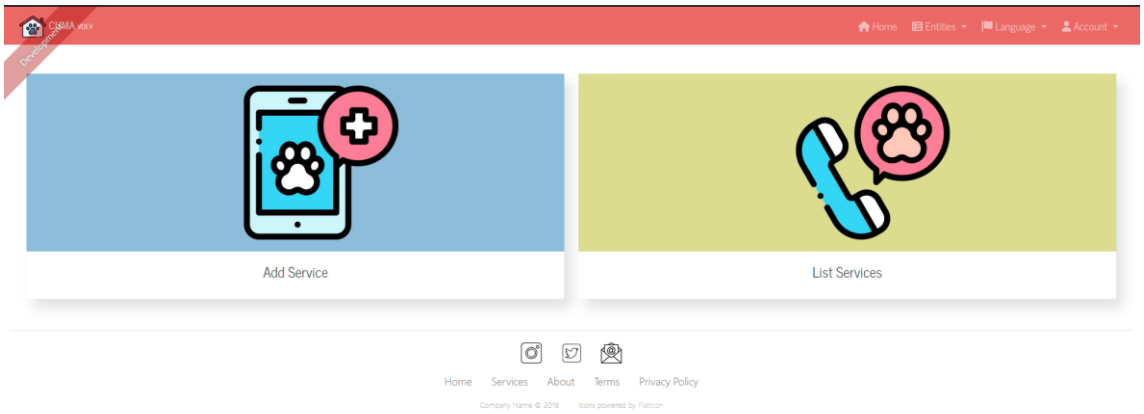

Figura 40. Menú principal usuario servicio.

6.3.1 Menú de creación de servicios

Esta ventana será accesible tras pulsar el botón "*Add Service*" de la Figura 40Figura 32. En ella se permitirá al usuario Servicio rellenar el formulario de creación de servicios, indicando campos como: localización, datos de contacto, descripción etc.

Mediante esta ventana se cumplimenta parcialmente el caso de uso **CU3**, ofreciendo la funcionalidad de creación de servicios privados en la aplicación definida en el requisito **R3.2**.

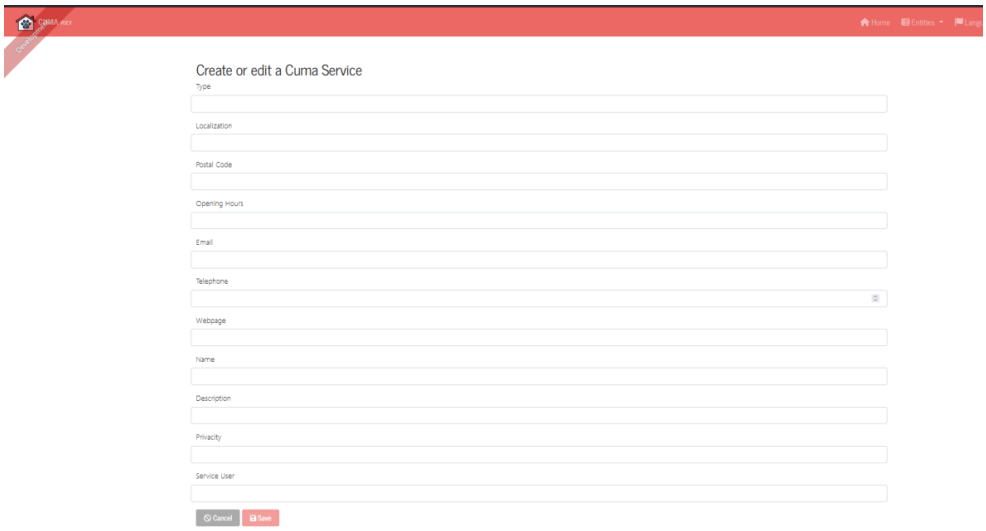

Figura 41. Ventana de creación de servicios privados.

6.3.2 Menú de visualización de servicios

Tras pulsar el botón "*List Services*" de la Figura 40, se abrirá la ventana de visualización de servicios, en la que el usuario podrá visualizar los distintos servicios dados de alta en la aplicación, pudiendo filtrar estos por categoría si fuera necesario.

Esta ventana no hace referencia a ningún caso de uso n requisito concreto, pero mejora la experiencia del usuario servicio durante la creación de nuevos servicios.

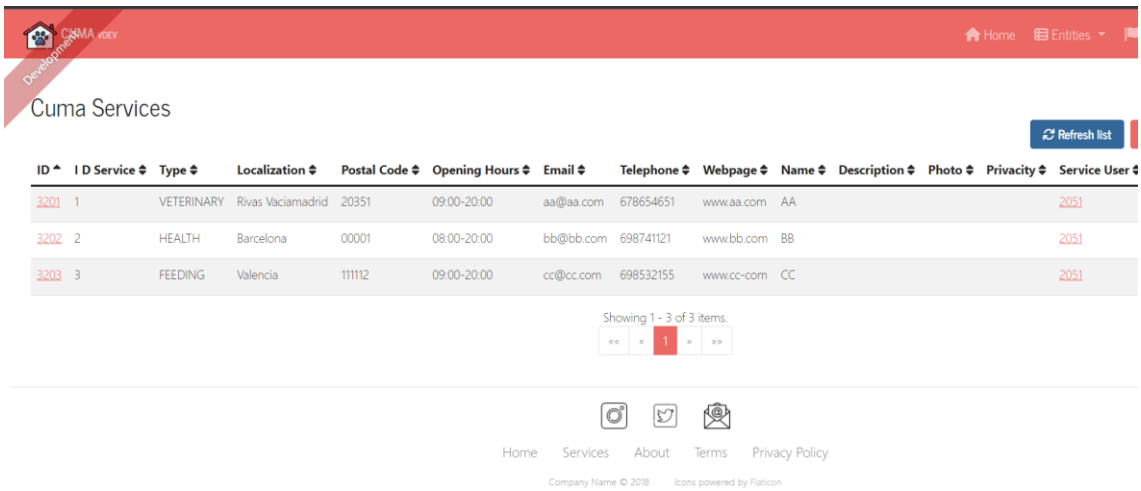

Figura 42. Ventana de visualización de servicios del usuario Servicio.

6.4 Menús de administración

Este conjunto de ventanas recoge la funcionalidad destinada a los usuarios con perfil Aplicación, la cual está destinada a tareas de mantenimiento y creación de consejos y servicios. Por lo tanto, solo los usuarios con perfil Aplicación podrán acceder a esta funcionalidad.

El menú principal del usuario Aplicación, dispone de una barra de navegación situada en la parte superior, la cual mediante listas desplegables ofrece el acceso a las distintas funcionalidades, Figura 43.

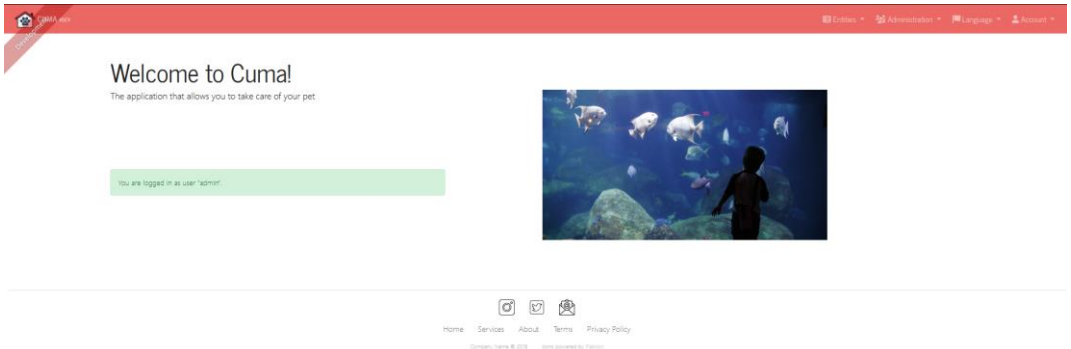

Figura 43. Menú principal del usuario Aplicación.

6.4.1 Menú de creación de consejos

Esta ventana será accesible mediante un submenú del desplegable "*Entities*" situado en la barra de navegación. En ella se permitirá al usuario Aplicación generar un consejo utilizando para ello el formulario expuesto en la Figura 44. Una vez generado el consejo este podrá ser visualizado por los usuarios con perfil *Dueño* mediante la ventana correspondiente, apartado 6.2.3.

Mediante esta ventana se ofrece la funcionalidad descrita en el caso de uso **CU5**, concretamente la creación de consejos en la aplicación. Con ello, y junto con la venta expuesta en el apartado 6.2.3, se cumplimenta totalmente el requisito **R2.3**.

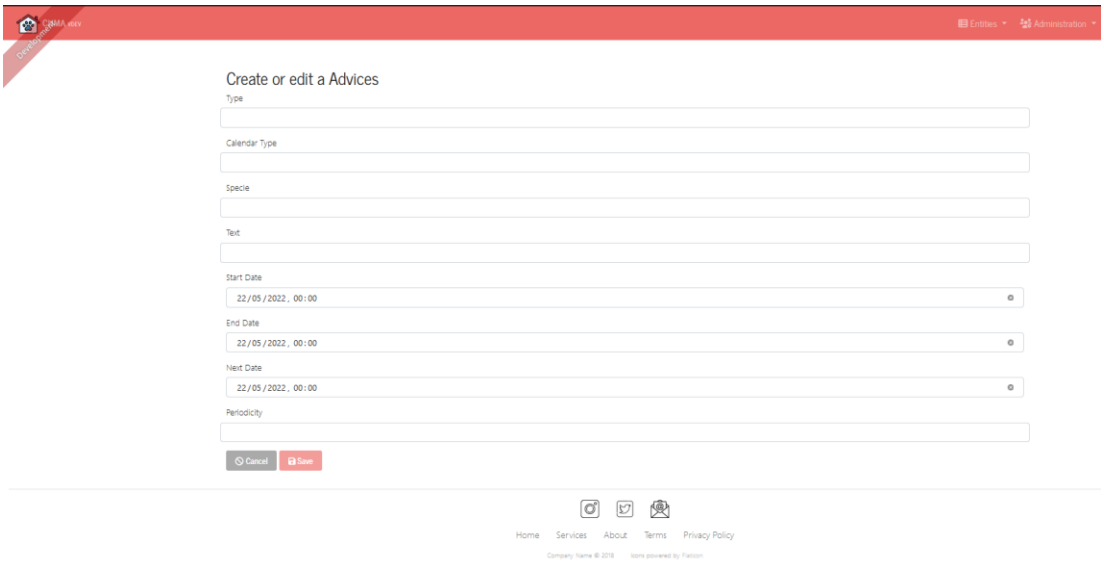

Figura 44. Menú de creación de consejos.

6.4.2 Menú de creación de servicios públicos.

Esta ventana será accesible mediante un submenú del desplegable "*Entities*" situado en la barra de navegación Figura 32. En ella se permitirá al usuario Servicio realizar la creación de un servicio del tipo público.

Mediante esta ventana, junto con las mostradas en los apartados 6.2.4 y 6.3.1, se completa totalmente el caso de uso **CU3**, ofreciendo la funcionalidad de creación y visualización de servicios públicos y privados en la aplicación. Con ello se cumplimentan además los requisitos: **R3.2** y **R3.3**.

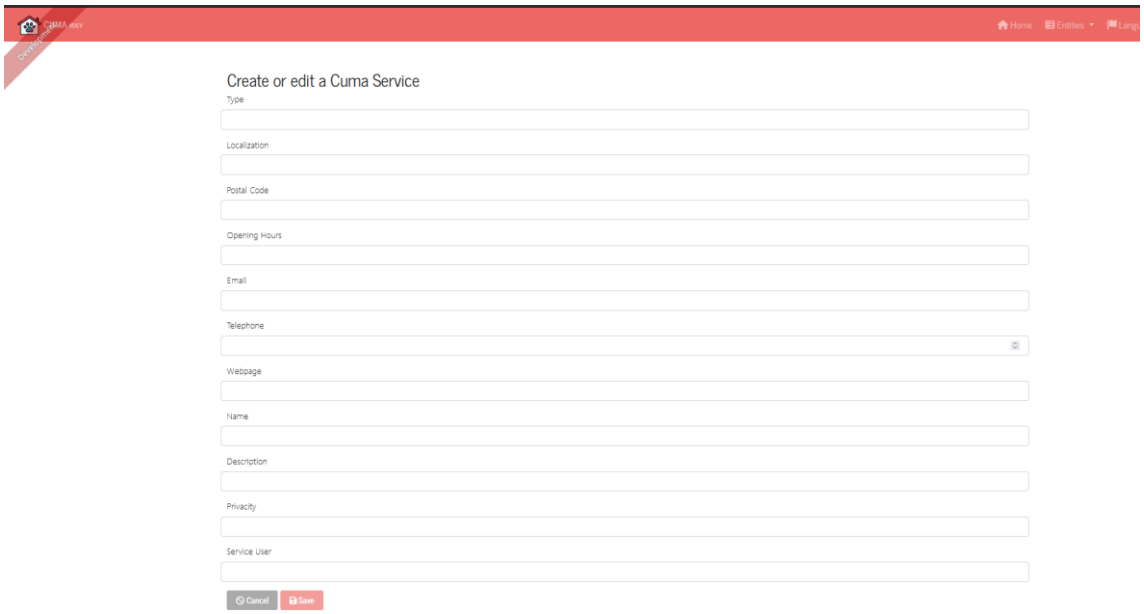

Figura 45. Ventana de creación de servicios públicos.

6.4.3 Soporte multilenguaje.

Parte de la funcionalidad añadida de la aplicación es el soporte multilenguaje. En todas las ventanas, en la esquina superior derecha, existe un botón llamado "*Language*", el cual permite cambiar el idioma en el que se visualiza la ventana entre: ingles, castellano y catalán. Un ejemplo de esto puede verse en la Figura 46

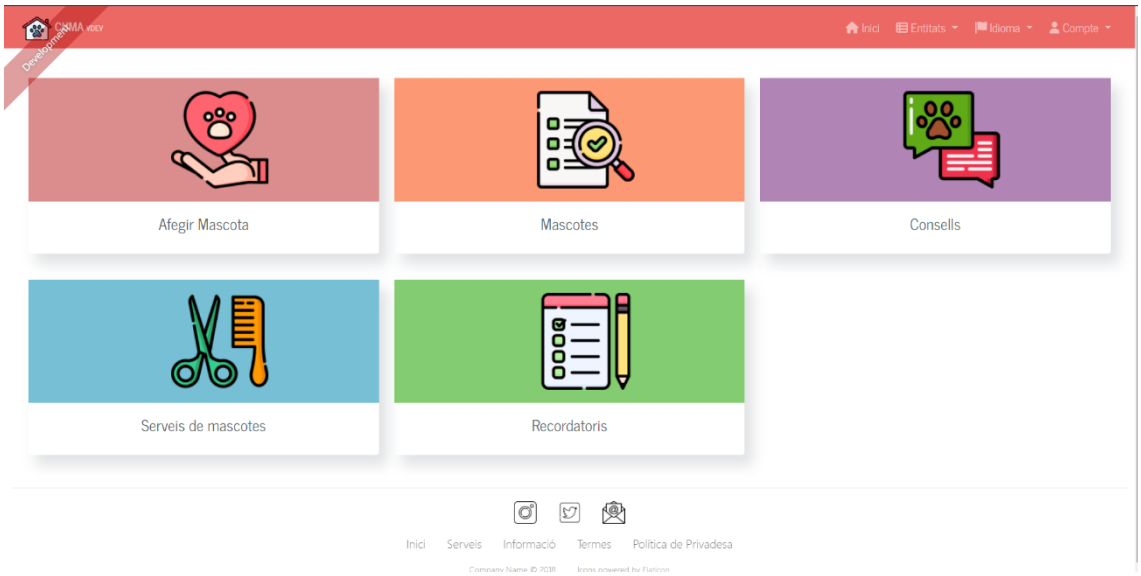

Figura 46. Menú de usuario Dueño en catalán.

6.4.4 Gestión de usuarios.

En la aplicación los usuarios administradores, identificados como el personal de mantenimiento de la aplicación.

Parte de la gestión de la aplicación implica la gestión de usuarios, esta puede realizarse a través de la ventana mostrada en la Figura 47, a la cual solo pueden acceder los usuarios con rol administrador. En ella se pueden visualizar los detalles de todos los usuarios registrados, así como modificar los mismos.

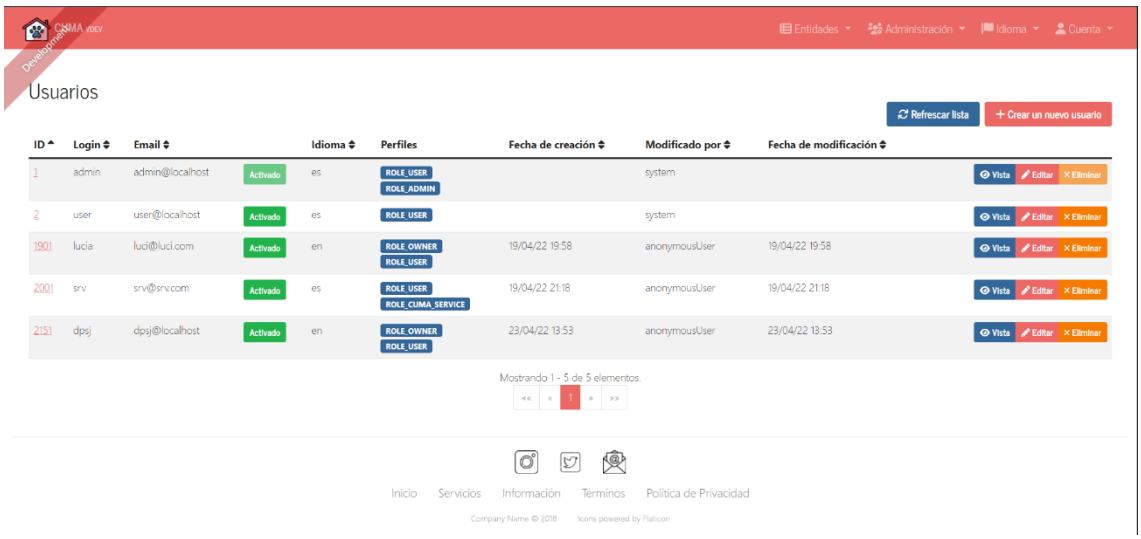

Figura 47. Ventana de gestión de usuarios.

7 Conclusiones

Para finalizar se analizan los objetivos propuestos y su cumplimiento, así como los resultados y conocimiento obtenido.

Como se expuso en el apartado 1.2, el objetivo principal del proyecto es generar una aplicación web, la cual permita centralizar y facilitar la realización de todas las tareas relacionadas con el cuidado de las mascotas. Para cumplir dicho objetivo, se definieron tres objetivos principales, los cuales fueron plasmados en distintos requisitos y casos de uso más concretos.

1. Centralización de datos de mascotas.
2. Gestión y planificación de cuidados y mantenimiento de las mascotas.
3. Interacción con espacios destinados a mascotas de la ciudad

Analizando los resultados expuestos en el apartado 6, puede observase que se han cumplido en su totalidad todos los requisitos definidos para la aplicación.

Por un lado, la aplicación ha sido capaz de gestionar los datos de mascotas referentes a: identificación, cuidados y mantenimiento, proporcionando a los usuarios facilidad para el registro de nuevos datos, y visualización del histórico de los mismos. Con esto se ha cumplimentado el eje central de la operativa requerida por la aplicación, ofreciendo a los usuarios todas las acciones necesarias para el cuidado básico de una mascota.

Por otro lado, se ha fomentado la interacción de la aplicación con la ciudad, proporcionando un fácil y centralizado acceso a información relativa a los servicios de mascotas de la ciudad. Con ello se ha cumplimentado el segundo gran eje de la aplicación, unificando en un único punto los servicios multidisciplinares que engloban el cuidado de una mascota.

Podemos ver que los tres objetivos planteados se han cumplido satisfactoriamente. Como consecuencia, se ha alcanzado la idea de generar una aplicación que englobe en un único punto las tareas relacionadas con el cuidado de mascotas, haciendo con ello que los usuarios ya no deban tener múltiples aplicaciones instaladas, facilitando con ello el cuidado de mascotas.

Sin embargo, siguiendo la idea que intenta aportar la aplicación, el escenario de interacción de la aplicación con la ciudad ha quedado ligeramente desviado del planteamiento deseado. Inicialmente el diseño de la aplicación incluía una interacción mayor con los servicios de la cuidad, planteando la posibilidad de hacer reservas sobre los mismos, o un mecanismo directo de mensajería, así como servicios de compras integrados. Finalmente, tras analizar la carga de trabajo que esto suponía, se decidió disminuir el alcance de esta funcionalidad, limitándola a la visualización de servicios disponibles en la cuidad. Es por ello, que, aunque el objetivo de interacción puede darse por completado, sería necesario desarrollar versiones futuras de la aplicación con mayor funcionalidad añadida.

Respecto a la planificación, definida en el apartado 1.4, se ha cumplido satisfactoriamente el diseño de fases establecido. Sin embargo, la Fase 3, dedicada a la implementación, ha requerido de ciertas alteraciones en el cronograma, reasignando tiempo a tareas de implementación que resultaron ser más costosas de lo previsto.

Para finalizar, cabe destacar las lecciones aprendidas durante la elaboración del proyecto. La principal lección aprendida es que en un proyecto de desarrollo software, la fase de planificación y diseño es mucho más importante de lo que parece. Normalmente los desarrolladores software tienden a comenzar la implementación de las aplicaciones lo más rápido posible, reduciendo el tiempo de las fases de planificación y diseño. Esto genera que muchas decisiones tengan que ser tomadas sobre la marcha durante la implementación, generando que estas decisiones suelan tener un enfoque centrado en la tarea que el desarrollador estaba realizando en ese momento, y no un enfoque global. Con ello se generan fallos de diseño, en los que los recursos están repetidos, o no están bien aprovechados, haciendo aplicaciones ineficientes y difíciles de mantener.

Como resultado he aprendido de la importancia de dedicarle tiempo al diseño de la aplicación. Es muy importante al iniciar un proyecto parase a pensar en todos los posibles escenarios, anotando los requerimientos de cada uno, teniendo en cuenta todas las posibles opciones. Una vez obtenida esta información, es el momento de comenzar el diseño de: tablas de base de datos, estructura de clases, interconexión entre servicios etc. Realizando los desarrollos de esta manera se generan aplicaciones mucho más sólidas y coherentes.

8 Líneas de trabajo futuro

Tras finalizar este proyecto y utilizando el conocimiento adquirido durante la realización de este, se han analizado los posibles trabajos futuros que ampliarían la cobertura y prestaciones implementadas.

Por un lado, tal como se mencionó en el apartado anterior, sería positivo mejorar la interacción de la aplicación con la ciudad, implementando de un sistema de reservas de citas directo, a través de la aplicación, entre los usuarios con perfil Dueño y Servicio. Para ello sería necesario implementar un sistema de mensajería, mediante el cual un usuario Dueño, al visualizar un servicio, tuviera un botón para enviar un mensaje al mismo.

Por otro lado, una mejora en la aplicación sería ampliar el número de especies que puede gestionar la aplicación, añadiendo: reptiles, roedores y pequeños mamíferos. Esto supondría modificar toda la capa de creación y visualización de información de mascotas, añadiendo nuevas opciones y ventanas para las nuevas especies.

Otro punto futuro de mejora sería orientar la aplicación no solo al ámbito de la gestión, sino al ámbito social. Para ello se podría implementar una funcionalidad de red social, en la que los usuarios con el perfil *Dueño* puedan compartir fotos y videos de sus mascotas con otros usuarios, al estilo de otras redes sociales como Instagram.

9 Glosario

Tabla 9. Definición de acrónimos utilizados.

Acrónimo	*Definición*
ANFAAC	*Asociación Nacional de Fabricantes de Alimentos para Animales de Compañía*
TFM	*Trabajo de Fin de Máster*
REIAC	Red Española de Identificación de Animales de Compañía
AWS	Amazon Web Services
INE	Instituto Nacional de Estadística
MVC	Modelo Vista Controlador
ORM	Object Relational Mapper
CRUD	Create, Read, Update, Delete
DAO	Data Access Object
JPA	Jakarta Persistence API
DTO	Data Transfer Object
POO	Programación Orientada a Objetos
API	Application Programming Interface
SOAP	Simple Object Access Protocol
REST	REpresentational State Transfer
XML	extensible markup language
HTTP	*Hypertext Transfer Protocol*
SMTP	Simple Mail Transfer Protocol
TCP	Transmission Control Protocol
JSON	JavaScript Object Notation
HTML	Hypertext Markup Language
JWT	Json Web Token

Tabla 10. Definición de términos utilizados.

Término	*Definición*
back	*Parte de una aplicación web, centrada en la lógica de negocio y el almacenamiento, es conocida como el lado servidor.*
front	*Parte de una aplicación web, centrada en la interacción con los usuarios, es conocida como el lado del cliente.*
Fundación affinity	*organización sin ánimo de lucro con el objetivo de «promocionar el rol de las mascotas en la sociedad».*
Angular	Framework (marco de trabajo) de JavaScript de código abierto escrito en TypeScript, utilizado para el desarrollo del front de aplicaciones web
React	Librería de JavaScript para el desarrollo UI (Interfaz de usuario), desarrollada por Facebook

10 Bibliografía

[1] ANFACC, «Asociación Nacional de Fabricantes de Alimentos para Animales de Compañia,» [En línea]. Available: https://www.anfaac.org/datos-sectoriales/. [Último acceso: 24 02 2022].

[2] C. B. Mases, «¿Por qué en los hogares españoles hay más mascotas que hijos?,» *La Vanguardia,* 29 07 2021.

[3] 11pets, [En línea]. Available: https://www.11pets.com/es. [Último acceso: 24 02 2022].

[4] ExpertoAnimal, [En línea]. Available: https://www.expertoanimal.com/. [Último acceso: 24 02 2022].

[5] «dogbuddy,» [En línea]. Available: https://es.dogbuddy.com/. [Último acceso: 24 02 2022].

[6] Red Española de Identificación de animales de compañía, «REIAC,» [En línea]. Available: https://www.reiac.es/. [Último acceso: 12 03 2022].

[7] ANFAAC, «Asociación Nacional de Fabricantes de Alimentos para Animales de Compañía,» 2021. [En línea]. Available: https://www.anfaac.org/datos-sectoriales/. [Último acceso: 12 03 2022].

[8] Veterindustria, «Veterindustria. Asociación Empresarial Española de la Industria de Sanidad y Nutrición Animal,» 2021. [En línea]. Available: https://www.veterindustria.com/estadisticas/veterindustria/el-sector-en-cifras_108_1_ap.html. [Último acceso: 12 03 2022].

[9] Ayuntamiento de Madrid, «Portal de datos abiertos del Ayuntamiento de Madrid,» 16 07 2021. [En línea]. Available: https://datos.madrid.es/portal/site/egob/menuitem.c05c1f754a33a9fbe4b2e4b284f1a5a0/?vgnextoid=3e573d68ae8a6410VgnVCM1000000b205a0aRCRD&vgnextchannel=374512b9ace9f310VgnVCM100000171f5a0aRCRD&vgnextfmt=default. [Último acceso: 09 03 2022].

[10] Gerència d' Ecologia, Urbanisme i Mobilitat, «Open Data BCN Servicio de datos abiertos del Ayuntamiento de Barcelona,» ajuntament.barcelona.cat, 2021. [En línea]. Available: https://opendata-ajuntament.barcelona.cat/data/es/dataset/cens-animals-companyia. [Último acceso: 12 03 2022].

[11] Google, «Google Trends,» [En línea]. Available: https://trends.google.es/trends/explore?date=today%205-y&geo=ES&q=adoptar%20perro,adoptar%20gato. [Último acceso: 12 03 2022].

[12] D. J. Fatjó, «Estudio “El nunca lo haría” de la Fundación Affinity sobre el abandono, la pérdida y la adopción de animales de compañía en España 2,» 2021.

[13] L. p. d. u. 4. e. n. d. a. d. compañía, «La pandemia dispara un 44% el número de animales de compañía,» *La vanguardia,* 25 01 2022.

[14] I. i. u. o. N. a. d. E. a. ica, «Encuesta Continua de Hogares (ECH),» 2021.

[15] I. N. d. Estadística, «INE,» 2020. [En línea]. Available: https://www.ine.es/jaxiT3/Datos.htm?t=1433#!tabs-grafico. [Último acceso: 12 03 2022].

[16] A. Orús, «statista,» 06 12 2021. [En línea]. Available: https://es.statista.com/estadisticas/567772/poblacion-canina-en-la-union-europea-por-pais/. [Último acceso: 12 03 2022].

[17] Escuela de postgrado veterinaria, «Escuela de postgrado veterinaria,» 21 07 2021. [En línea]. Available: https://postgradoveterinaria.com/cuidados-perro-recomendaciones-basicas/. [Último acceso: 12 03 2022].

[18] V. D. Mirón, «misanimales,» 08 02 2022. [En línea]. Available: https://misanimales.com/los-cuidados-basicos-de-los-peces/. [Último acceso: 13 03 2022].

[19] T. Baena, «animalcity,» 11 08 2021. [En línea]. Available: https://animalcity.es/clinica/cuidados-basicos-de-los-gatos/. [Último acceso: 12 03 2022].

[20] calcuworld, «calcuworld,» [En línea]. Available: https://es.calcuworld.com/cuantos/cuanto-vive-un-loro/. [Último acceso: 12 03 2022].

[21] wakyma, «wakyma,» [En línea]. Available: https://wakyma.com/blog/consejos-clave-cuidado-de-aves/. [Último acceso: 12 03 2022].

[22] iKibble, «iKibble,» Llamaface, [En línea]. Available: http://www.ikibble.com/. [Último acceso: 09 03 2022].

[23] Noblecan, «Noblecan Adiestramiento canino,» [En línea]. Available: https://www.noblecan.com/adiestrador-canino-online/. [Último acceso: 09 03 2022].

[24] J. M. Aguilar, «CampusMVP,» 15 10 2019. [En línea]. Available: https://www.campusmvp.es/recursos/post/que-es-el-patron-mvc-en-programacion-y-por-que-es-util.aspx. [Último acceso: 03 04 202].

[25] C. Recalde, I. Verón y C. Nuñez, «Estudio, diseño y comparación entre aplicaciones,» Facultad Politécnica, Universidad Nacional de Asunción , San Lorenzo, Paraguay.

[26] D. López y E. Maya, «Arquitectura de Software basada en Microservicios para,» Séptima Conferencia de Directores de Tecnología de Información, San José, 2017.

[27] Á. Rojas, «icentro,» 01 07 2021. [En línea]. Available: https://www.incentro.com/es-ES/blog/que-es-api-interfaz-programacion-aplicaciones. [Último acceso: 03 05 2022].

[28] Red Hat, «redhat,» 08 04 2019. [En línea]. Available: https://www.redhat.com/es/topics/integration/whats-the-difference-between-soap-rest. [Último acceso: 03 05 2022].

[29] J. M. Alarcón, «campusmvp,» 26 02 2018. [En línea]. Available: https://www.campusmvp.es/recursos/post/que-es-un-orm.aspx. [Último acceso: 03 05 2022].

[30] o. M. Alarcón, «campusmvp,» 23 03 2021. [En línea]. Available: https://www.campusmvp.es/recursos/post/la-api-de-persistencia-de-java-

que-es-jpa-jpa-vs-hibernate-vs-eclipselink-vs-spring-jpa.aspx. [Último acceso: 03 05 2022].

[31] spring, «docs.spring.io,» [En línea]. Available: https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/bind/annotation/RequestMapping.html. [Último acceso: 05 05 2022].

[32] spring, «spring.io,» [En línea]. Available: https://spring.io/guides/gs/securing-web/. [Último acceso: 22 05 2022].

11 Anexos

11.1 Anexo 1: Esquema de base de datos de la entidad mascota y sus relaciones.

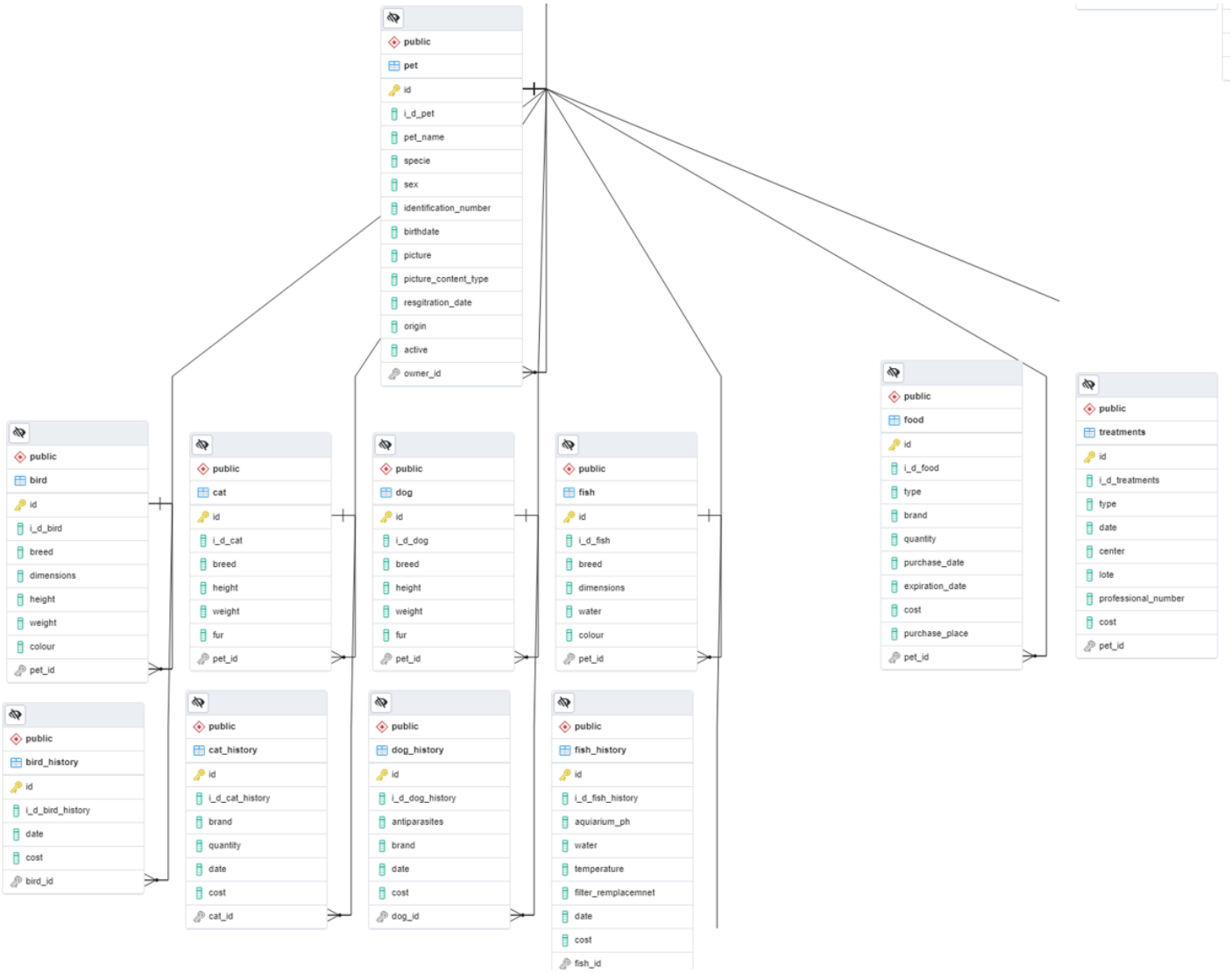

Figura 48. Esquema de entidad mascota y sus relaciones en base de datos.

11.2 Anexo 2: Esquema de base de datos de las entidades de usuarios.

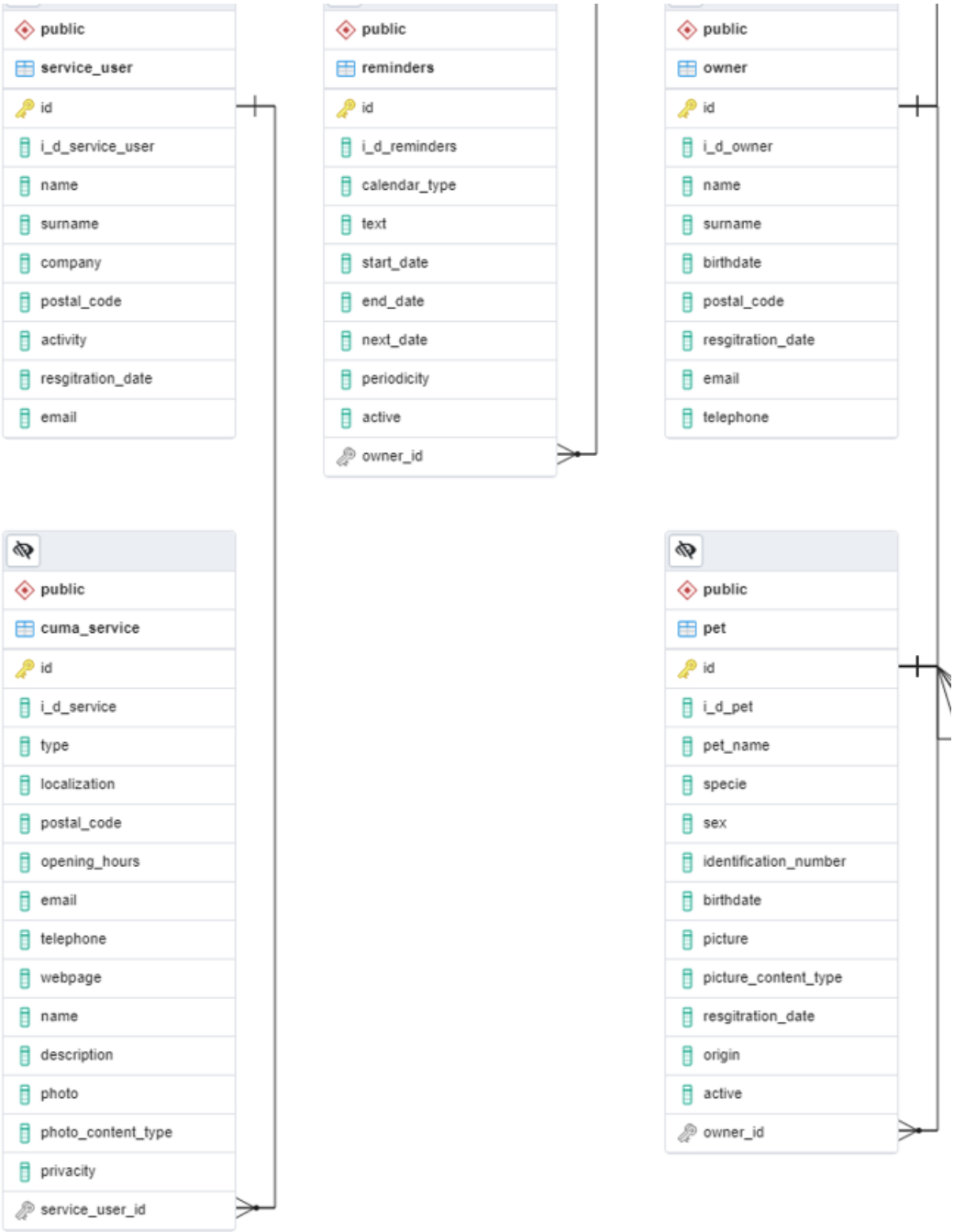

Figura 49. Esquema de las entidades de usuarios de la aplicación.

11.3 Anexo 3. Diagrama completo de base de datos.

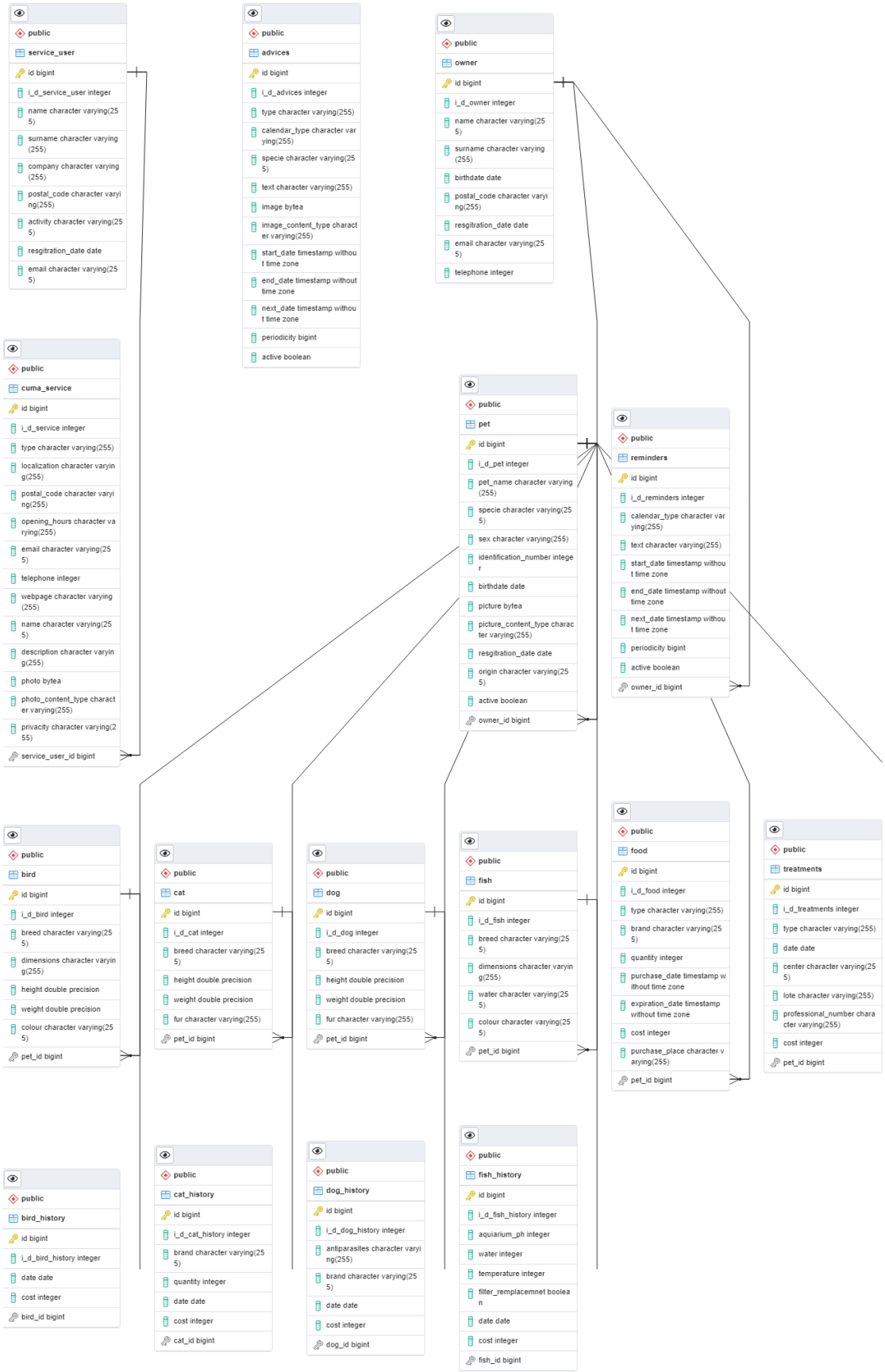

Figura 50. Diagrama completo de base de datos.