Facultade de Informática
UNIVERSIDADE DA CORUÑA

TRABALLO FIN DE GRAO
GRAO EN ENXEÑARÍA INFORMÁTICA
MENCIÓN EN TECNOLOXÍAS DA INFORMACIÓN

# Aplicación web para la ayuda a la adopción de una mascota

**Estudante:** Jose Miguel Alvarez-Campana Medina
**Dirección:** Carlos Fernández Lozano

A Coruña, setembro de 2021.

*Para mi familia y amigos*

**Agradecimientos**

**A mi tutor D. Carlos Fernández Lozano** por su ayuda, su amabilidad y paciencia.

**A mis padres** que siempre han estado ahí apoyándome y animándome durante toda la carrera.

**A mis compañeros** de fatiga, algunos siempre tendrán un lugar importante en mis recuerdos universitarios.

**Resumen**

El objetivo de este proyecto es el desarrollo de una aplicación web que permita al usuario registrarse y acceder a una base de datos donde tendría al alcance de su mano todos los datos de los animales disponibles para una posible adopción. La base de datos se nutriría de los registros de las protectoras que irían actualizando periódicamente los datos de los animales disponibles. Los futuros adoptantes, a través de diferentes filtros, podrían encontrar la mascota que mejor se adecue a sus necesidades. Además, podrán activar una alerta que les notificaría cuando un animal se añada a la base de datos que cumpla sus requisitos.

Para la creación de la aplicación se hace uso de herramientas de código abierto como Angular, Bootstrap, Java y Docker. Este servicio estará conectado a una base de datos creada en MySql.

**Abstract**

The objective of this project is the development of a web application that allows the user to register and access a database where he would have access to all the available animals. The database would save the registries of the protectors that would periodically update the data of the available animals.Future adopters, through different filters, could find the pet that best suits their needs. In addition, they will be able to activate an alert that would notify them when an animal is added to the database that meets their requirements.

Open source tools such as Angular, Bootstrap, Java and Docker are used to create the application. This service will be connected to a database created in MySql.

**Palabras clave:**

- Java
- SQL
- Angular
- JavaScript
- Docker
- Bootstrap
- Animales
- Adopciones

**Keywords:**

- Java
- SQL
- Angular
- JavaScript
- Docker
- Bootstrap
- Animals
- Adoptions

# Índice general

# Índice de figuras

# Índice de tablas

Capítulo 1

# Introdución

## 1.1 Motivación

El abandono de animales ha crecido en los últimos años y ronda alrededor de los 138.000 al año. España se encuentra entre los países del mundo con mayor tasa de abandonos según la Fundación Affinity. Con la llegada del COVID-19, los abandonos se han visto incrementados a causa de los fallecimientos.

Con la llegada del confinamiento, una de las excepciones que permitía el gobierno para salir de casa era sacar a las mascotas a pasear. Esta medida tuvo como consecuencia el aumento de la compra de una mascota, sobre todo cachorros. La real sociedad canina ha contabilizado el aumento en un 50%. Sin embargo, este aumento no se ha producido por igual en las adopciones, que, a diferencia de la compra de animales, ha disminuido un 40% según datos de la Fundación Affinity. Las protectoras han visto frenado estos procesos debido al confinamiento al no incluir la adopción como causa para poder salir del domicilio.

El estudio que ha realizado el departamento de psiquiatría de la Universidad Autónoma de Barcelona constata que aproximadamente el 74% de las personas que durante la pandemia tuvieron mascota en casa, esta les ayudo a superar el encierro.

Si a esto le sumamos que en Navidad hay una clara tendencia a regalar mascotas a los niños que después se abandonan, en concreto, un 30% del total lo que supone más de 100.000 animales al año. Esto nos lleva a tener protectoras con sobrepoblación de las cuales el 43% de los animales, según datos de Animal's Health, revista especializada, no son adoptados.

Actualmente, los adoptantes tienen que revisar distintas webs para ver los animales disponibles en las perreras y a la vez, estas organizaciones no tienen manera de contactar con los posibles candidatos salvo cuando ellos los contactan. Uno de los métodos más usados es Facebook como medio de difusión a través de las publicaciones de los usuarios.

Se propone el desarrollo de una aplicación web que permita al usuario registrarse y acceder a una base de datos donde tendría al alcance de su mano todos los datos de los animales

disponibles. La base de datos se nutriría de los registros de las protectoras que irían actualizando periódicamente los datos de los animales disponibles. Los futuros adoptantes a través de diferentes filtros podrían encontrar la mascota que mejor se adecue a sus necesidades. Además, podrán activar una alerta que les notificaría cuando un animal se añada a la base de datos que cumpla sus requisitos.

Cambios en la familia o una mudanza pueden servir de excusa (0,8%): Embarazo, nacimiento de un hijo o una alergia, enfermedad o muerte del cuidador del animal pueden hacer que la familia se deshaga del perro.

Falta de tiempo y falta de dedicación en la educación de las mascotas, no todos los propietarios están dispuestos a dedicar tiempo en ellos. Ya no quiero a mi mascota (10,8 %). Los cachorros hacen sus necesidades en casa, destrozan todo y no conocen ni las órdenes ni las normas de convivencia. Las mascotas crecen, hay que sacarlas a pasear, darles de comer, cuidarlos, llevarlos al veterinario, etc. Todos los días, sin importar el trabajo que tengamos, el tiempo que haga, ni si estamos mal de salud.

Causas económicas; no poder mantener a la mascota (6,4 %) es otro de los motivos para deshacerse de la mascota. Puede ser la perdida de empleo o un

empeoramiento puede llegar a no poder asumir el gasto que supone mantener a un animal (comida, veterinario...) También puede deberse a un cambio en los intereses personales que hace que se prefiera gastar en otras actividades antes que seguir manteniendo a la mascota (Figura 1.1 página 3).

Problemas de conducta o de comportamiento (13,2 %) puede ser una causa con la que se pretende justificar un abandono, que el perro ladre en exceso, que orine en casa, que no obedezca, etc. Un perro que no se ha socializado adecuadamente, no está bien educado ni estimulado o no tiene cubiertas sus necesidades básicas estará estresado y lo manifestará con comportamientos que nos resultarán desagradables.

Abandono de perros de caza (11,6 %). Es un problema en que debería tomar medidas las federaciones de caza.

Camadas no deseadas (21 %). La principal causa de abandono son las camadas sin control. Las perras acostumbran a tener un par de celos al año y pueden parir desde 2 a 8-10 cachorros.

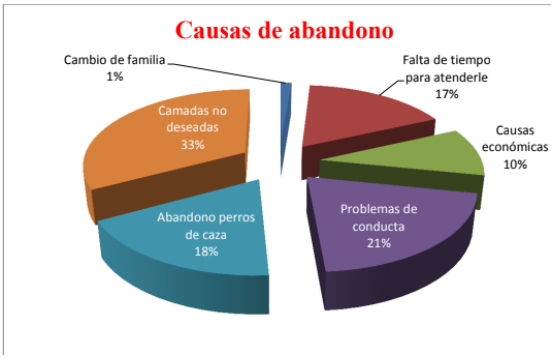


Figura 1.1: Causas del abandono de perros

Según datos de la Fundación Affinity y las Asociaciones Protectoras y de Defensa Animal (FAPAM), en España se abandonan tres perros y gatos cada cinco minutos. Tener una mascota acarrea una responsabilidad.

Las protectoras y los ayuntamientos recogen 400 perros y gatos de la calle al día. Solo 2 de cada 10 perros son llevados a los refugios por sus familias y el 60% de las mascotas abandonadas son encontradas por particulares y menos del 50% acaban adoptadas [1].

España posee una de las tasas de abandono animal más altas de Europa. En España es obligatoria la identificación de los animales de compañía, con microchip. En el caso de los perros, casi el 90 % está identificado. Sin embargo, la mitad de los gatos no lo están.

Solamente el 10 % de propietarios llevaron personalmente a su animal a la protectora. La gran mayoría fueron encontrados (63,3 %) o traídos por un tercero (26,9 %). La llegada a la protectora en el caso de los perros no es estacional, sí ocurre con los gatos, que llegan de abril a octubre a causa de su ciclo reproductivo.

El tiempo medio de permanencia es de 10 meses en animales adultos y de tres meses en cachorros. El 44 % de los perros y gatos que llegaron en 2019 a las protectoras fueron adoptados por una nueva familia. Un porcentaje muy similar al de años anteriores. Los datos se repiten de año en año y son escalofriantes.

Los perros atendidos en protectoras españolas en 2019 eran: Adultos (60,8 %), mestizos (60,6 %), de tamaño mediano - grande (78 %), con buen estado de salud (66,5 %) e identificados con microchip (25 %). En el caso de los gatos: Cachorros (48,1 %) y adultos (44,6 %), mestizos (94,8 %), en buen estado de salud (51,4%) y con microchip (4 %).

## 1.2 Objetivos

La finalidad del proyecto es la creación de una plataforma web que facilite tanto la labor de los usuarios a la hora de buscar una mascota que se adapte a ellos como la gestión de los animales por los refugios. A continuación, se detallarán las funcionalidades del proyecto:

- Gestión de usuarios: operaciones relacionadas con los dos tipos de usurarios, clientes y refugios, como son: dar de alta, actualizar alguno de sus parámetros e iniciar sesión.
- Funcionalidades de los refugios: Podrán añadir y ver los animales a su cargo, además de poder ver el historial de donaciones que los clientes han realizado.
- Gestión de animales: los refugios podrán modificar los atributos de los perros, así como consultar su historial de apadrinamientos y adopciones.
- Funcionalidades de los clientes: podrán realizar una búsqueda de los animales por campos. Además de ofrecer la opción de adoptarla podrán guardar la ficha del animal en su lista de favoritos y también ofrece la posibilidad de apadrinarlo. También permite la búsqueda de refugios por ciudad y la opción de realizar una donación. En su perfil podrán ver el historial de adopciones, apadrinamientos y donaciones realizados a través de la plataforma.

Capítulo 2

# Estado del arte

Aparte de los modelos tradicionales de buscar una mascota para adoptar, es decir a través de ir fisicamente a los refugios o preguntar a los familiares o amigos, han surgido otras maneras como son realizar adopciones a través de redes sociales o aplicaciones. A continuación, se expondrá tres ejemplos de estos último:

## 2.1 SalvaUnaMascota.es

En esta página se pueden registrar dos tipos de usuarios, el particular que busca adoptar y la protectora que ofrece la mascota [2].

La página ofrece la posibilidad de seleccionar la provincia, el tipo de mascota, perros o gatos en este caso, el tamaño y el sexo .

Seleccionando la foto de la mascota se puede conocer el tamaño, edad, vacunas, si está esterilizada, o desparasitada o con/sin chip, y una pequeña descripción como su nombre, raza y raza de sus padres.

Se puede contactar con la protectora por medio de un teléfono o escribiendo a una dirección de correo electrónico.

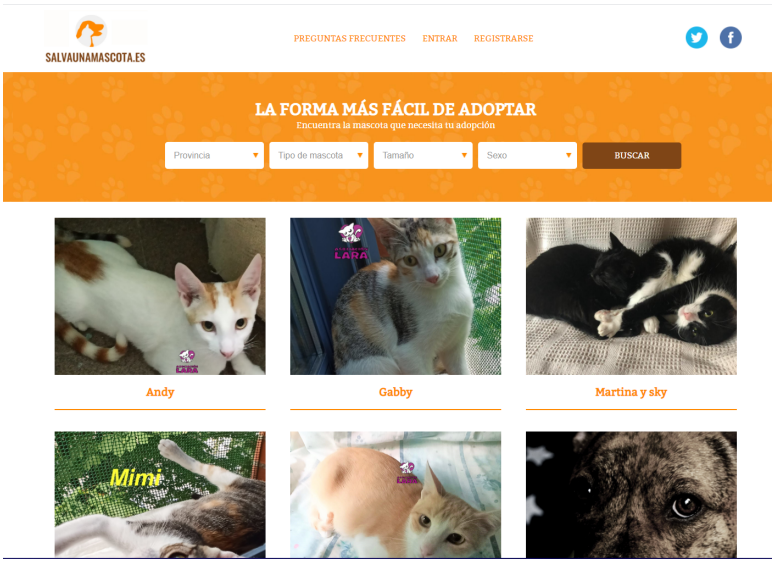


Figura 2.1: Captura de la interfaz de SalvaUnaMascota

## 2.2 AdoptaMe

Es una aplicación española que fue publicada en Play Store hace unos meses. Permite el registro de dos tipos de cuentas: usuarios y refugios [3].

Una vez se registra un usuario, se muestra una interfz con fotos de las mascotas disponibles con una breve descripción (Figura 2.2 página 7).

Ofrece la posibilidad de realizar búsquedas por provincia y raza y la opción de seleccionar favoritos, para que se pueda acceder a ellos de una forma más sencilla y rápida.

Esta app muestra la opción de que el usuario pueda ser voluntario en un refugio, cuando no sea posible adoptar una mascota.

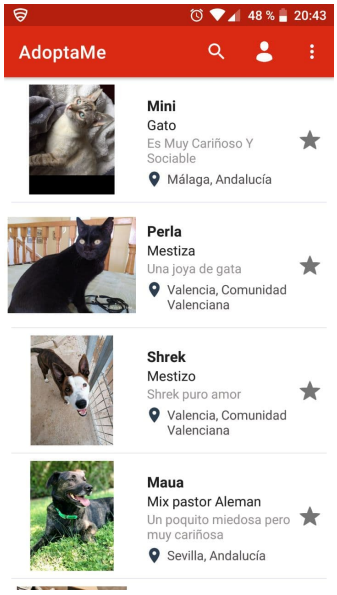


Figura 2.2: Captura de la interfaz de AdoptaMe

## 2.3 Cidade amigable cos animais

Sección dentro de la página del ayuntamiento de A Coruña [4] donde se publican fotos de perros y gatos, disponibles para adoptar, con una breve reseña sobre su edad y su raza. La persona interesada puede solicitar más información a través de un formulario de contacto (Figura 2.3 página 8).

Es un portal para favorecer la labor del refugio de animales y ofrece un espacio para que las personas que hayan perdido a su mascota puedan poner un aviso. En resumen esta sección está al servicio de la ciudadanía, no es un portal exclusivo para adopciones.

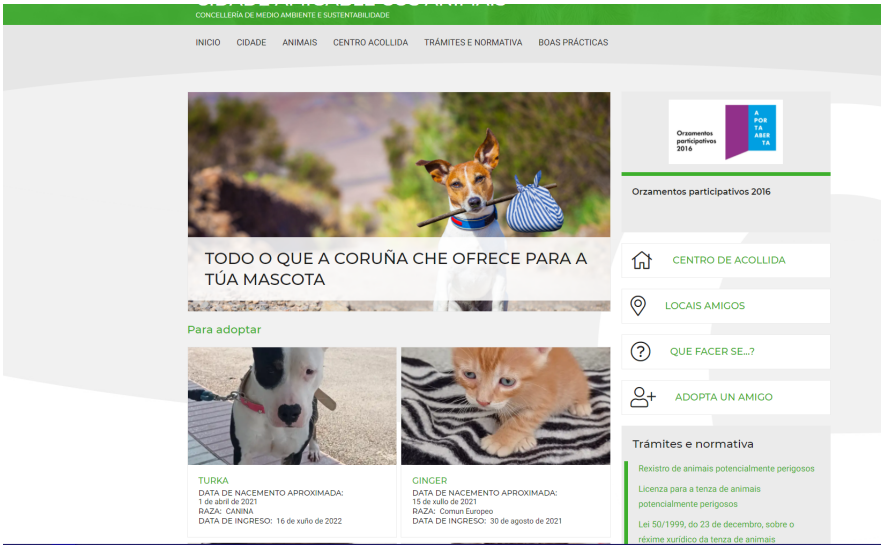


Figura 2.3: Captura de la interfaz de la página web del ayuntamiento

## 2.4 Conclusiones

Después de observar estos ejemplos con la misma temática, se ve la importancia de los filtros de búsqueda, estos deben ser amplios para que sea rápida y eficaz, para que la persona que está interesada en adoptar no se canse y desista en su empeño.

Para la buena realización de los filtros se debe poner atención a las característacas de la mascota que sean interesantes para el usuario, como pueden ser: raza, tamaño, sexo, edad, etc.

Se observa ,en general, la carencia en estas páginas de una sección donde se ofrezca a los usuarios solidarios, que no puedan adoptar, la opción de contribuir económicamente. Ayudando así en el buen cuidado y mantenimiento de los animales, cubriendo las necesidades hasta su adopción.

Por todo esto se propone con este TFG el desarrollo de una plataforma web que facilite la búsqueda de una mascota que se adapte a las preferencias del usuario y permita a los refugios gestionar las adopciones. También ofrecerá de una manera sencilla contribuir económicamente a aquellas personas solidarias que deseen colaborar con los refugios.

Capítulo 3

# Tecnologias y herramientas usadas

## 3.1 Lenguajes de programación

### 3.1.1 Java

Java es un lenguaje de alto nivel, orientado a objetos y basado en entidades llamadas clases creado por Sun Microsystems. Es un lenguaje imperativo con cierta similitud a C, pero a su vez incorpora herramientas como por ejemplo ,gestión de hilos o ejecución remota.

Es uno de los lenguajes más usados en el desarrollo de aplicaciones. Oracle ofrece además el Java Development Kit (JDK) que ofrece un entorno Java Runtime Environment (JRE), que contiene una máquina virtual de java ,Java Virtual Machine (JVM), y todas las librerías necesarias para el desarrollo de la aplicación. La facilidad que da este sistema, es que una vez compilado el programa se puede ejecutar en cualquier máquina que tenga una JVM [5].

En el desarrollo de este proyecto se ha usado Java JDK 11.

### 3.1.2 HTML

HyperText Markup Language (HTML) es un lenguaje sencillo basado en etiquetas, es decir cada una de ellas definira a través de sus atributos como se comporta. Las etiquedas describen cada uno de los componentes web [6].

HTML es el lenguaje estándar de la World Wide Consorcium (W3C), organización cuyo objetivo es la estandarización de todas las tecnologías relacionadas con la web. Este lenguaje es el usado para la creación de las páginas web, con apoyo de otras tecnologías que veremos a continuación.

### 3.1.3 CSS

W3C ante la necesidad de la creación de una tecnología que diera formato de manera consistente. Hubo dos propuestas, Cascading HTML Style Sheets (CHSS) y Stream-based Style

Sheet Proposal (SSP) que fueron unificadas en la Cascading Style Sheets (CSS) [6].

CSS es el lenguaje usado para dar presentación a las páginas webs creadas con HTML o Extensible Markup Language (XML). Aparte de definir los colores, fuentes y disposición, Nos permitirá adaptar la página web a diferentes tamaños de pantalla y dispositivos.

### 3.1.4 JavaScript

A principio de los años 90, se empezaron a desarrollar las primeras aplicaciones web. A medida que iban incluyendo cada vez formularios más complejos, la navegación se volvía mucho más lenta. Cabe destacar, que en aquella época la mayoría de los usuarios usaban un moden con 28.8 kbps de velocidad.

Gracias a Brendan Eich, ingeniero de Netscape, se introdujo un lenguaje basados en tecnologías existentes como ScriptEase, en el navegador. Años más tarde Netscape y Sun Microsystems desarrollaron conjuntamente JavaScript.

Es un lenguaje imperativo basado en objetos. Se usa principalmente desde el lado del usuario, la mayoría de los navegadores lo incluyen. JavaScript permite una notable mejoría en las interaces y aplicaciones web dinámicas [7].

Este lenguaje fue creado para la inclusión de contenido ejecutable en las páginas web. Es un lenguaje ligero, es decir no necesita preprocesado (compilación) antes de de ser ejecutado, que permite la creación de páginas web dinámicas.

### 3.1.5 SQL

Structured Query Lenguage (SQL) es un lenguaje estructurado de alto nivel estándar usado en el manejo de bases de datos relacionales. Este lenguaje fue creado por American Normalization and Standarization Institute (ANSI) junto con International Standarization Organism (ISO), es por este motivo que este lenguaje es el que está presente en la mayoría de las herramientas de manejo de las bases de datos relacionada.

El SQL engloba tres tipos de sentencias con finalidades distintas:

- Lenguaje de Manipulación de Datos (DML): Insert, Update, Delete y Select.
- Lenguaje de Definición de Datos (DDL): Create, Alter y Drop.
- Lenguaje de Control de Datos (DCL): Commit, Rollback, Grant y Revoke.

## 3.2 Librerías y frameworks

### 3.2.1 JDBC API

Java DataBase Connectivity (JDBC) es un driver de acceso a la base de datos SQL que ofrece un acceso uniforme y cómodo a una gran variedad de bases de datos, en nuestro caso será MySQL [8].

Usaremos JDBC para enviar las sentencias SQL definidas en Java según los parametros adecuados, permitiendo manipular los datos. Usando esta API nos permite acceder a la base de datos que crearemos para el almacenamiento persistnente de información.

### 3.2.2 Apache

Es un servidor HTTP que se ejecuta en segundo plano admitiendo múltiples peticiones simultáneas. En un primer momento fue diseñado para sólo ser usado en sistemas Unix, pero posteriormente se implementó para otros sistemas operativos [9].

Los servidores Apache son open-source que lo usan acerca del 40% de los sitios web en todo el mundo. Actúa como intermediario entre el servidor y el cliente. Extrae la información almacenada en el servidor y la envía a la web cuando el cliente la solicita.

Su fama no proviene de su núcleo, que es bastante sencillo, si no de la cantidad tan diversa de extensiones que ofrecen multitudes de funcionalidades diferentes. Además son muy fáciles de introducir, sólo hay que instalar la extensión y reiniciar el servidor.

### 3.2.3 Jetty

Jetty proporciona un servidor web y un contenedor de servlets. Está bajo una licencia dual de Apache y Eclipse. Esta tecnología es conocida por su fácil integración en aplicaciones, dispositivos y frameworks [10].

Una de sus propiedades es que es ligero, es decir, si incluimos el archivo central de Jetty en la aplicación será incluido en el .jar, permitiendo así usarlo como servidor integrado.

### 3.2.4 Javax Mail

Librería de java desarrollada por SUN que ofrece herramientas comunes de los servicios de mensajería electrónica. En nuestro caso lo usaremos para facilitar la automatización de envío de correos electrónicos a los usuarios [11].

### 3.2.5 Bootstrap

Es un framework CSS, JavaScript y HTML para facilitar el desarrollo web rápido y responsive . Incluye librerías de CSS y JavaScript para la creación de la interfaz web que será compatible con todos los navegadores y se adaptará a los distintos dispositivos [12].

Otra de sus ventajas, es que es un framework de código libre apoyado por una gran comunidad que no sólo crea nuevas funcionalidades, si no que también identifica y soluciona posibles problemas de sus versiones oficiales.

Se diferencia del resto de frameworks parecidos en que se basa en la división de la web en una rejilla que permite dividirla en filas y columnsa de distinto tamaño. Además da soporte a contenidos habituales en las webs como formularios, botonoes, menús, etc.

### 3.2.6 Angular

Desde su creación en el 2012 denominado Angular JS ha sido el framework preferido por los dearrolladores web. Debido al exito mucho ingenieros querían usar este framework para muchas cosas, exponiendo así sus limitaciones [13].

Actualmente solo se llama Angular, este paso fue significativo de su evolución como un framework más allá del JavaScript. Angular es un framework de código libre de JavaScript escrito en TypeScript, que son dos lenguajes compatibles.

Google le da soporte y su objetivo principal es desarrollar aplicaciones de una sola página. Es decir, la primera carga en la página es más lenta, pero a partir de ahí el resto de cargas casi van a ser instantaneas.

Como framework, Angular tiene claras ventajas al mismo tiempo que proporciona una estructura estándar para que los desarrolladores trabajen con ella, permitiendo a los usuarios crear grandes aplicaciones de forma sostenible (Figura 3.1 página 12).


Figura 3.1: Logo de Angular

### 3.2.7 MySql

Es un sistema de gestión de bases de datos relacional de código abierto muy extendido junto con Apache creado por MySQL AB. MySQL ofrece un servidor SQL rápido, multiusuario

y robusto, también ofrece un cliente y distintas APIs para dar soporte a diversos lenguajes, el más usado es Hypertext Preprocessor (PHP) [14].

## 3.3 Herramientas de diseño

### 3.3.1 Intellij IDEA

Es un Integrated development environments (IDE) diseñado en Java cuya funcionalidad es facilitar la labor de los programadores al desarrollar software, fue desarrollado por JetBrains. Ofrece un editor de código, un compilador y un intérprete [15].

Exsiten dos ediciones: community edition y edición comercial, que como su nombre indica es de pago, mientras que la primera es de código abierto. Este IDE puede hacer uso de diferentes complementos para aumentar sus funcionalidades.

### 3.3.2 Maven

Es una herramienta de código abierto para la construcción automática de proyectos, usada principalmente en Java. Facilita el uso e instalación de librerías externas al tenerlas en su repositorio. También estandariza los proyectos Java, facilitando el uso compartido de las compilaciones JAR [16].

Pero Maven es más que una herramienta que facilita las ”builds” en los proyectos, es capaz de gestionar todo un proyecto desde la validación del código, pasando por la ejecución de test hasta el despliegue de la aplicación. También facilita la gestión y dependencias de distintos módulos y librerías.

### 3.3.3 Docker

Docker es una herramienta de código libre muy usada actualmente, ofrece la posibilidad de ejecutar tu programa en un entorno controlado y definido, permitiendo así una ejecución idéntica en distintos sistemas operativos [17].

Además permite la conexión entre disitintos contenedores, permitiendo así la creación de una aplicación modular. Muchas empresas y apliaciones usan estos contenedores para irlos activando según la carga de trabajo que tengan. También es muy útil a la hora de actualizar un módulo, sólo has de desconectar el módulo desactualizado y activar el nuevo, reduciendo así la perdida de servicio (Figura 3.2 página 13).


Figura 3.2: Logo de Docker

Capítulo 4

# Metodología

El proyecto ha sido realizado siguiendo una arquitectura por capas, o también conocido como el patrón Modelo Vista Controlador (MVC). Este planteamiento divide la aplicación en tres partes diferentes, lo que permite una mayor flexibilidad y encapsulación.

La primera capa, llamada capa modelo, a construir es la encargada de gestionar la base de datos para que refleje las peticiones de los usuarios y contiene la lógica de negocio. La siguiente capa es el controlador que se encarga de procesar las peticiones de los clientes y comunicárselas a la capa modelo. Por último, tenemos la capa cliente, cuya función es presentar al cliente una interfaz sencilla e intuitiva para que realice sus peticiones y se reflejen las respuestas del servidor.

## 4.1 Metodología del desarrollo

Para la implementación del proyecto se ha optado por Scrum [18], una metodología ágil para el desarrollo de proyectos software. Esta metodología incremental se basa en definir una serie de iteraciones, llamadas sprints, en el ciclo de vida. Cada uno de estos sprints tiene como objetivo desarrollar una o varias funcionalidades de manera que pueden ser expuestas al cliente.

De esta manera, en cada sprint que realicemos iremos obteniendo el feedback del cliente, que nos permitirá modificar las iteraciones realizadas, así como, orientar las siguientes hacia el gusto del cliente.

Se ha elegido esta metodología, al tener claro las funcionalidades desde el principio, lo que permite definir de una manera sencilla las iteraciones. Al final de cada sprint se realizará una revisión para detectar los posibles fallos.

### 4.1.1 Organización del equipo

En una metodología scrum los integrantes asumirán uno o varios de los siguientes roles:

- Product Owner: es el encargado de optimizar y maximizar el valor del producto. El product owner registrará, en el Product Backlog, los requisitos del proyecto al inicio del ciclo de vida para que estén claros desde el principio.
- Scrum Master: tiene dos funcionalidades principales: gestionar el proceso scrum y eliminar los posibles impedimentos que comprometan el desarrollo.
- Equipo de desarrollo: formado por todos los empleados encargados de implementar el proyecto.

### 4.1.2 Conceptos relacionados

En este apartado revisaremos las definiciones de algunos conceptos de la metodología scrum:

- Sprint: Duración durante la cual se completa una iteración.
- Product Backlog: guía que recoge un listado ordenado por preferencias de los objetivos que debe de cumplir el proyecto. Todos los miembros pueden acceder al Product BackLog para revisar su contenido, pero sólo el Product Owner tiene los permisos de poder editarlo.
- Sprint Backlog: conjunto de elementos, seleccionados del Product Backlog, necesarios para la implementación de un sprint.
- Incremento: el conjunto de todos los elementos del Produck Backlog completados durante un sprint y de los incrementos anteriores.
- Revisión del sprint y reuniones planificadas del sprint: reuniones marcadas por la finalización de un incremento.
- Scrum diario: reuniones cortas donde el equipo se pone al día con el progreso realizado por sus compañeros.
- Retroespectiva del sprint: reunión organizada por el Scrum Master con el objetivo de mejorar las futuras iteraciones gracias a la experiencia de las anteriores.

### 4.1.3 Adaptación de la metodología

La metodología scrum está dirigida para grupos de trabajo entre 5 y 9 personas. En este caso, se simulará que el equipo de desarrollo está formado integramente por el alumno y el tutor asumirá los roles de scrum master y product owner.

Al principio de cada día, simularé un scrum diario, donde revisaré el trabajo anterior y cuales son mis próximos pasos. Si surgiera algún problema en el desarrollo se notificaría al scrum master, es decir al tutor.

Después de cada sprint, se haría una reunión donde se expondría una simulación del producto para verificar que cumple con los requisitos y se abordaría las posibles futuras optimizaciones basadas en la experiencia adquirida.

### 4.1.4 Requisitos

#### Requisitos funcionales

Los requisitos funcionales son los que especifican la funcionalidad del sistema. Se han identificado una serie de necesidades que el proyecto debe de cumplir:

- Como cliente poder filtrar los resultados de una búsqueda por unas características.
- Como cliente poder guardar los animales que me gusten para acceder a ellos más rápido.
- Como cliente poder recibir notificaciones de cuando exista un perro que se acomode a mis gustos sin tener que entrar a mirar en la aplicación los nuevos animales añadidos.
- Como cliente poder contribuir monetariamente a la labor de un refugio
- Como cliente poder apadrinar un animal.
- Como cliente poder cambiar las preerencias.
- Como refugio poder añadir con facilidad animales.
- Como refugio poder administrar los perros añadidos por mí.
- Como refugio poder revisar el historial de adopciones y apadrinaciones.
- Como refugio poder editar mi información.

#### Requisitos no funcionales

- Consistencia: El modelo siempre se mantiene actualizado.
- Fiabilidad: Gestiona los posibles fallos de manera controlada.
- Seguridad: las claves se almacenan cifradas en la base de datos.
- Simplicidad y usabilidad: La interfaz es simple e intuitiva para los usuarios.

### Usuarios

En este proyecto se contemplan dos tipos de usuarios en la aplicación:

- Cliente: usuario registrado en la plataforma, que puede realizar búsquedas por campos o por Id, adoptar un animal, apadrinarlo o guardarlo en su lista de preferidos para facilitar su acceso en el futuro. También podrá buscar refugios según la ciudad permitiendo saber así su ubicación además de ofrecer la posibilidad de realizar una donación. El cliente podrá establecer una serie de preferencias que permitirá al sistema avisarle automáticamente por email de cuando exista un perro disponible que se adapte a sus preferencias. El usuario también podrá acceder a su historial de donaciones, apadrinamientos y adopciones realizados a través de esta plataforma.

- Refugios: la plataforma pone a disposición de las perreras una serie de funcionalidades para la gestión de sus "inquilinos" tales como, añadir un nuevo animal, la posibilidad de modificar sus campos, así como ver el historial de adopciones y apadrinamientos. También ofrecerá la posibilidad de ver sus donaciones recibidas y la opción de editar alguno de sus atributos.

### Historias de usuario

Las historias de usuario son una explicación general de lo que al cliente le gustaría que ofreciese la aplicación. En nuestro caso serían:

Tabla 4.1: HU1-Registro de Cliente

| | |
|---|---|
| Descripción | Registro de un nuevo cliente en el sistema. |
| Usuario | Cliente. |
| Precondiciones | El correo del cliente no debe existir en la base de datos. |
| Flujo de acciones | 1. El sistema muestra los campos a rellenar por el cliente.<br>2. El cliente rellena todos los campos obligatorios.<br>3. El sistema valida que la información introducida es válida y la almacena en la base de datos. |
| Flujo alternativo de acciones | Si los datos no son válidos el sistema no los registra. |
| Postcondiciones | Se crea el nuevo cliente en el sistema. |

Tabla 4.2: HU2-Iniciar sesión

| | |
|---|---|
| Descripción | El cliente inicia sesión en la plataforma. |
| Usuario | Cliente. |
| Precondiciones | El cliente debe de estar registrado con anterioridad. |
| Flujo de acciones | 1. El sistema muestra los campos a rellenar por el cliente.<br>2. El cliente rellena todos los campos obligatorios.<br>3. El sistema valida que la información introducida es válida y redirige al cliente al área cliente. |
| Flujo alternativo de acciones | Si los datos no son válidos el sistema no lo redirige. |
| Postcondiciones | El cliente inicia sesión. |

Tabla 4.3: HU3-Buscar perro por campos

| | |
|---|---|
| Descripción | El cliente realiza una búsqueda según unos parámetros. |
| Usuario | Cliente. |
| Precondiciones | El cliente debe de haber iniciado sesión. |
| Flujo de acciones | 1. El sistema muestra los campos a rellenar por el cliente.<br>2. El cliente rellena todos los campos obligatorios.<br>3. El sistema valida que la información introducida es válida y muestra al cliente la lista de animales que cumplan los requisitos. |
| Flujo alternativo de acciones | - |
| Postcondiciones | El sistema ofrece un listado de los animales que cumplen las condiciones. |

Tabla 4.4: HU4-Buscar perro por id

| | |
|---|---|
| Descripción | El cliente realiza una búsqueda según el identificador de un perro. |
| Usuario | Cliente. |
| Precondiciones | El cliente debe de haber iniciado sesión. |
| Flujo de acciones | 1. El sistema muestra el campo a rellenar por el cliente.<br>2. El cliente rellena todos los campos obligatorios.<br>3. El sistema valida que la información introducida es válida y muestra al cliente el animal cuyo id coincide con el introducido. |
| Flujo alternativo de acciones | El sistema no muestra nada si no hay un perro con ese identificador. |
| Postcondiciones | El sistema muestra al animal que tiene ese identificador. |

Tabla 4.5: HU5-Añadir perro a la lista de favoritos

| | |
|---|---|
| Descripción | El cliente añade a su lista de favoritos un perro a su elección. |
| Usuario | Cliente. |
| Precondiciones | El cliente debe de haber realizado una búsqueda exitosa. |
| Flujo de acciones | 1. El cliente solicita su deseo de guardar la ficha del animal en su lista de favoritos.<br>2. El sistema añade a la lista de seguimientos del cliente el id del perro. |
| Flujo alternativo de acciones | - |
| Postcondiciones | La ficha del animal estará visible en la lista de favoritos del cliente. |

Tabla 4.6: HU6-Apadrinar un perro

| | |
|---|---|
| Descripción | El cliente apadrina un perro. |
| Usuario | Cliente. |
| Precondiciones | El cliente debe de haber realizado una búsqueda exitosa y el refugio ofrece la posibilidad de apadrinar ese animal. |
| Flujo de acciones | 1. El cliente solicita su deseo de apadrinar el animal.<br>2. El sistema muestra un formulario para completar por el cliente.<br>3. El cliente rellena el formulario con su información.<br>4. El sistema valida los datos introducidos y ejecuta la transacción. Redirige al cliente de vuelta al área clientes. |
| Flujo alternativo de acciones | - |
| Postcondiciones | La transacción es reflejada en el historial de apadrinamientos del cliente y del perro. |

Tabla 4.7: HU7-Adoptar un perro

| | |
|---|---|
| Descripción | El cliente adopta un perro. |
| Usuario | Cliente. |
| Precondiciones | El cliente debe de haber realizado una búsqueda exitosa y el refugio debe de ofrecer la posibilidad de adoptarlo. |
| Flujo de acciones | 1. El cliente solicita su deseo de adoptar el animal.<br>2. El sistema muestra un formulario para completar por el cliente.<br>3. El cliente rellena el formulario con su información.<br>4. El sistema valida los datos introducidos y ejecuta el procedimiento. Redirige al cliente de vuelta al área clientes. |
| Flujo alternativo de acciones | - |
| Postcondiciones | La operación es reflejada en el historial de adopciones del cliente y del perro. |

Tabla 4.8: HU8-Ver lista de favoritos

| | |
|---|---|
| Descripción | El cliente visualiza su lista de animales favoritos. |
| Usuario | Cliente. |
| Precondiciones | El cliente debe de haber iniciado sesión con anterioridad. |
| Flujo de acciones | 1.El cliente solicita su deseo de visualizar la lista de sus perros favoritos.<br>2. El sistema muestra la relación de animales que el cliente ha añadido a su lista de favoritos. |
| Flujo alternativo de acciones | El sistema no muestra nada si no hay ningún animal añadido a la lista. |
| Postcondiciones | El sistema muestra la relación de perros añadidos a esta lista. |

Tabla 4.9: HU9-Ver historial de apadrinamientos

| | |
|---|---|
| Descripción | El cliente visualiza su historial de apadrinamientos. |
| Usuario | Cliente. |
| Precondiciones | El cliente debe de haber iniciado sesión con anterioridad. |
| Flujo de acciones | 1. El cliente solicita su deseo de visualizar la lista de los apadrinamientos que ha realizado a través de la aplicación.<br>2. El sistema muestra la relación de apadrinamientos que el cliente ha realizado. |
| Flujo alternativo de acciones | El sistema no muestra nada si el cliente no ha apadrinado ningún perro mediante la aplicación. |
| Postcondiciones | El sistema muestra la relación de perros apadrinados a través de la aplicación. |

Tabla 4.10: HU10-Ver historial de donaciones

| | |
|---|---|
| Descripción | El cliente visualiza su historial de donaciones. |
| Usuario | Cliente. |
| Precondiciones | El cliente debe de haber iniciado sesión con anterioridad. |
| Flujo de acciones | 1. El cliente solicita su deseo de visualizar la lista de las donaciones que ha realizado a través de la aplicación.<br>2. El sistema muestra la relación de donaciones que el cliente ha realizado. |
| Flujo alternativo de acciones | El sistema no muestra nada si el cliente no ha donado a ningún mediante la aplicación. |
| Postcondiciones | El sistema muestra la relación de donaciones realizadas a través de la aplicación. |

Tabla 4.11: HU11-Ver historial de adopciones

| | |
|---|---|
| Descripción | El cliente visualiza su historial de adopciones. |
| Usuario | Cliente. |
| Precondiciones | El cliente debe de haber iniciado sesión con anterioridad. |
| Flujo de acciones | 1. El cliente solicita su deseo de visualizar la lista de los adopciones que ha realizado a través de la aplicación.<br>2. El sistema muestra la relación de animales adoptados que el cliente ha realizado. |
| Flujo alternativo de acciones | El sistema no muestra nada si el cliente no ha adoptado ningún perro mediante la aplicación. |
| Postcondiciones | El sistema muestra la relación de perros adoptados a través de la aplicación. |

Tabla 4.12: HU12-Editar perfil

| | |
|---|---|
| Descripción | El cliente actualiza ciertos parámetros de su perfil. |
| Usuario | Cliente. |
| Precondiciones | El cliente debe de haber iniciado sesión con anterioridad. |
| Flujo de acciones | 1. El cliente solicita su deseo de actuar su perfil.<br>2. El sistema muestra un formulario con los posibles campos a actualizar. |
| Flujo alternativo de acciones | - |
| Postcondiciones | El sistema actualiza la información del cliente. |

Tabla 4.13: HU13-Registro del Refugio

| | |
|---|---|
| Descripción | Registro de un nuevo refugio en el sistema. |
| Usuario | Refugio. |
| Precondiciones | El correo del refugio no debe existir en la base de datos. |
| Flujo de acciones | 1. El sistema muestra los campos a rellenar por el refugio.<br>2. El refugio rellena todos los campos obligatorios.<br>3. El sistema valida que la información introducida es válida y la almacena en la base de datos. |
| Flujo alternativo de acciones | Si los datos no son válidos el sistema no los registra. |
| Postcondiciones | Se crea el nuevo refugio en el sistema. |

Tabla 4.14: HU14-Inicio de sesión

| | |
|---|---|
| Descripción | El refugio inicia sesión en el sistema. |
| Usuario | Refugio. |
| Precondiciones | El refugio debe de haberse registrado con anterioridad. |
| Flujo de acciones | 1. El sistema muestra los campos a rellenar por el refugio.<br>2. El refugio rellena todos los campos obligatorios.<br>3. El sistema valida que la información introducida es válida y redirige al refugio al área refugio. |
| Flujo alternativo de acciones | Si los datos no son válidos el sistema no redirige al refugio. |
| Postcondiciones | El refugio inicia sesión. |

Tabla 4.15: HU15-Añadir perro

| | |
|---|---|
| Descripción | El refugio añade un nuevo animal. |
| Usuario | Refugio. |
| Precondiciones | El refugio debe de haber iniciado sesión con anterioridad. |
| Flujo de acciones | 1. El sistema muestra los campos a rellenar por el refugio.<br>2. El refugio rellena todos los campos obligatorios.<br>3. El sistema valida que la información introducida es válida y muestra un mensaje informando que ha sido añadido correctamente. |
| Flujo alternativo de acciones | Si los datos no son válidos el sistema no añade la información a la base de datos y no muestra el mensaje. |
| Postcondiciones | El sistema almacena la información del animal en la base de datos. |

Tabla 4.16: HU16-Ver los perros

| | |
|---|---|
| Descripción | El refugio visualiza la relación de caninos que ha añadido con anterioridad en la aplicación. |
| Usuario | Refugio. |
| Precondiciones | El refugio debe de haber iniciado sesión con anterioridad. |
| Flujo de acciones | 1. El refugio expresa el deseo de ver la lista de perros que ha añadido en la aplicación.<br>2. El sistema muestra la relación de animales almacenado. |
| Flujo alternativo de acciones | Si no se han añadido ningún animal no se muestra nada. |
| Postcondiciones | El sistema muestra el listado de animales añadidos por el refugio. |

Tabla 4.17: HU17-Acceder a la información del perro

| | |
|---|---|
| Descripción | El refugio visualiza información almacenada en el sistema sobre un perro particular. |
| Usuario | Refugio. |
| Precondiciones | El sistema debe de mostrar la lista animales que aloja el refugio. |
| Flujo de acciones | 1. El refugio expresa el deseo de ver la información total de un animal.<br>2. El sistema muestra toda la información acerca del perro. |
| Flujo alternativo de acciones | - |
| Postcondiciones | El sistema muestra la información detallada del animal. |

Tabla 4.18: HU18-Editar la información del perro

| | |
|---|---|
| Descripción | El refugio edita la información de un animal en particular. |
| Usuario | Refugio. |
| Precondiciones | El sistema debe de mostrar la información completa del animal. |
| Flujo de acciones | 1. El sistema muestra una serie de campos para que el refugio los pueda editar.<br>2. El refugio modifica los valores que necesite.<br>3. El sistema valida la información y la modifica en la base de datos. |
| Flujo alternativo de acciones | - |
| Postcondiciones | La información del perro es actualizada. |

Tabla 4.19: HU19-Ver las donaciones recibidas

| | |
|---|---|
| Descripción | El refugio visualiza la relación de donaciones recibidas por los clientes. |
| Usuario | Refugio. |
| Precondiciones | El refugio debe de haber iniciado sesión en la aplicación. |
| Flujo de acciones | 1. El refugio expresa el deseo de ver las donaciones recibidas.<br>2. El sistema muestra el listado de donaciones recibidas de los clientes a través de la aplicación. |
| Flujo alternativo de acciones | Si no ha recibido ninguna donación, no se muestra nada. |
| Postcondiciones | El sistema muestra la relación de donaciones. |

Tabla 4.20: HU20-Actualizar datos del refugio

| | |
|---|---|
| Descripción | El refugio actualiza alguno de sus atributos. |
| Usuario | Refugio. |
| Precondiciones | El refugio debe de haber iniciado sesión en la aplicación. |
| Flujo de acciones | 1. El refugio expresa el deseo de actualizar su información.<br>2. El sistema muestra un formulario con los posibles campos a actualizar.<br>3. El refugio completa los campos.<br>4.El sistema comprueba la validez de la información de los datos introducidos y actualiza la base de datos. |
| Flujo alternativo de acciones | - |
| Postcondiciones | El sistema actualiza los datos del refugio. |

Tabla 4.21: HU21-Cerrar sesión

| | |
|---|---|
| Descripción | El usuario cierra su sesión en la aplicación. |
| Usuario | Cliente o Refugio. |
| Precondiciones | El usuario debe de haber iniciado sesión. |
| Flujo de acciones | 1. El usuario expresa el deseo de cerrar la sesión.<br>2. El sistema cierra la sesión y le redirige a la página principal. |
| Flujo alternativo de acciones | - |
| Postcondiciones | El sistema cierra la sesión del usuario. |

## 4.2 Planificación

Después de planificar la base de datos y las historias de usuario, se procedió a seleccionar las tecnologías que se usarán para la implementación del proyecto. Se difinieron los siguientes 4 sprints de 3 semanas cada uno:

- Sprint 1 (11/06/2021 - 2/07/2021): Definido para la creación del núcleo del proyecto, creando la base datos y el servidor, además se implementará una interfaz sencilla que permita la creación de usuarios (clientes o refugio) y poder iniciar sesión (Figura 4.1 página 29).

  - Creación de la base de datos.
  - Definición de las entidades en la capa modelo.
  - Configuración del servidor.
  - Historias de uso realizadas en este sprint:
    - Registro de un cliente.
    - Inicio de sesión de un cliente.
    - Registro de un refugio.
    - Inicio de sesión de un refugio.

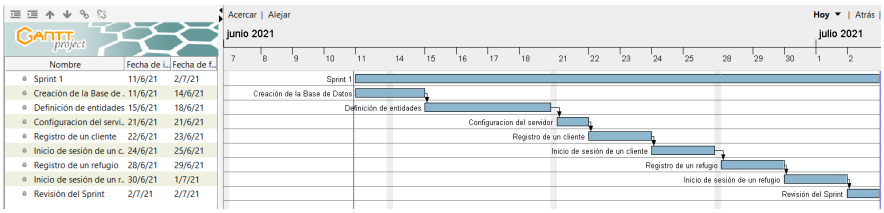


Figura 4.1: Diagrama de Gantt representando el sprint 1

- Sprint 2 (2/07/2021 - 23/07/2021): En esta etapa se definirán las operaciones principales de esta aplicación, es decir, aquellas relacionadas con los animales (Figura 4.2 página 30).

  - Cifrado de contraseñas.
  - Historias de uso realizadas en este sprint:
    - Cerrar sesión de un usuario.
    - Añadir un animal.
    - Búsqueda del animal por campos o por un identificador

* Poder editar las características del animal.
* Registrar una adopción.

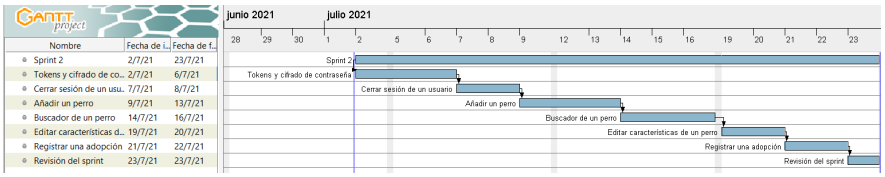


Figura 4.2: Diagrama de Gantt representando el sprint 2

- Sprint 3 (23/07/2021 - 13/08/2021): En esta iteración continuaremos con la implementación de las funcionalidades y adaptaremos la interfaz web para que acomode de manera intuitiva el resto de operaciones (Figura 4.3 página 30).

  – Notificación por email automático.
  – Mejora de la interfaz.
  – Historias de uso realizadas en este sprint:
    * Poder apadrinar un animal.
    * Poder ver el historial de adopciones y apadrinamientos.
    * Editar las preferencias de un cliente.
    * Editar las características del refugio.

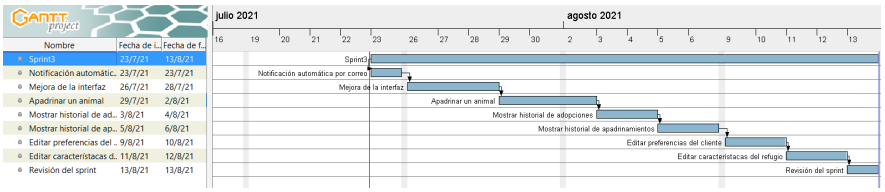


Figura 4.3: Diagrama de Gantt representando el sprint 3

- Sprint 4 (13/08/2021 - 3/09/2021): En esta iteración terminaremos de implementar el resto de historias de usuario así como adaptar el proyecto a docker (Figura 4.4 página 31).

  – Historias de uso realizadas en este sprint:
    * Poder realizar donaciones al refugio.
    * Búsqueda de refugios por ciudad.
    * Poder ver un histórico de donaciones.
    * Poder guardar perros en una lista separada para acceder más facilmente a ellos.

    * Editar las características del refugio.
  - Integrar la pasarela de pago Paypal.
  - Uso de docker para facilitar el despliegue del proyecto en diferentes sistemas operativos.

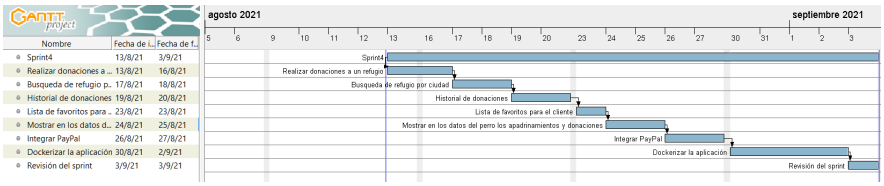


Figura 4.4: Diagrama de Gantt representando el sprint 4

## 4.3 Seguimiento

Durante el proceso de desarrollo del proyecto han surgido ciertas desviaciones, esto ha supuesto un impacto a la planificación temporal retrasando la fecha de finalización una semana más tarde:

- No se han tenido en cuenta en la planificación las tareas de corrección de errores que se han detectado durante las revisiones del sprint. Para corregir esto, se han incluido después de cada revisión una tarea de 3 horas para resolver los problemas.
- Durante la configuración de Docker se ha tenido que reorganizar la estructura del proyecto así como la modificación de ciertas configuraciones suponiendo un retraso de 3 horas.

A consecuencia de estas desviaciones se ha realizado una replanificación de la fecha de finalización siendo esta el 10 de septiembre y un aumento proporcianal del coste económico. En las figuras 4.5, 4.6, 4.7 y 4.8. Tras esta replanificación, el desarrollo del proyecto finaliza exitosamente en la fecha propuesta.

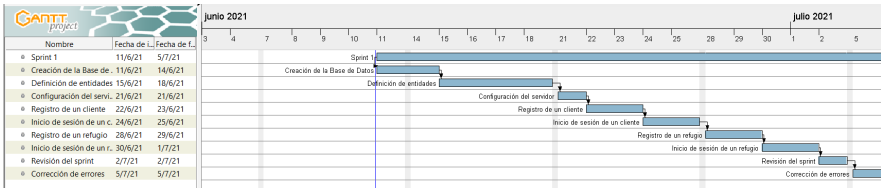


Figura 4.5: Diagrama de Gantt representando la replanificación el sprint 1

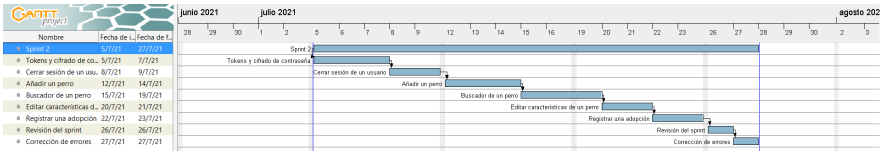


Figura 4.6: Diagrama de Gantt representando la replanificación el sprint 2

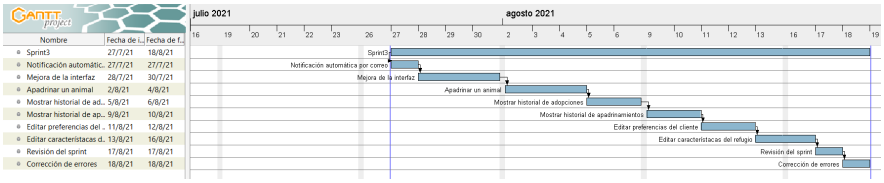


Figura 4.7: Diagrama de Gantt representando la replanificación el sprint 3

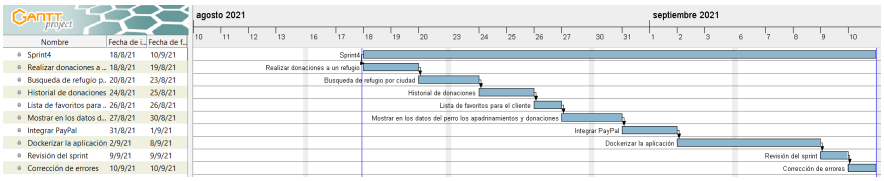


Figura 4.8: Diagrama de Gantt representando la replanificación sprint 4

## 4.4 Estimación del coste del proyecto

Al realizar este proyecto en el ordenador personal del estudiante con herramientas gratuitas, solo tenemos que considerar el gasto en recursos humanos:

- Analista-Programador (estudiante): Según el BOE de 2019[19], el saldo bruto al mes medio de este cargo es de 1712,42 € (14,69 €/h). Teniendo en cuenta que la implementación del proyecto duró 80 días y que la jornada eran de 3 horas diarias, el coste total ascendería a 3525,6 €.

- Jefe de proyecto: Según el BOE de 2019, el sueldo mensual medio es de 2500 € (19,53 €/h). El jefe de proyecto realizó una reunión de 2 horas después de cada iteración, más el tiempo de las reuniones iniciales y de las horas revisando la memoria. Conjuntamente son un total de 30 horas, por lo que el coste sería de 585,90€.

En total, el coste total de este proyecto es de 4111,50 €.

Capítulo 5

# Diseño de la aplicación

Ante la creación de la aplicación, se decidieron implementar una serie de patrones para facilitar el desarrollo y los posibles cambios en un futuro. Se ha optado por patrones creacionales (singleton, factoría), estructurales (fachada, DAO) y de comportamiento (MVC, DTO) [20] .

## 5.1 Modelo-Vista-Controlador

Es un patrón que consiste en dividir la aplicación consiste en modelado de datos, presentación de la información y control de los datos. Para que esta técnica sea efectiva es necesario establecer en diferentes capas cada una de las divisiones mencionadas. Esto permite una mayor flexibilidad y reutilización al poder cambiar una capa entera sin tener que modificar todo el código. Esta arquitectura es la más usada a la hora de desarrollar aplicaciones web.

- Modelo: esta capa es la encarga de establecer la representación de la información en el sistema, es decir, planifica los accesos a la información, como las búsquedas, actualizaciones y otros tipos de consultas. En esta división también estableceremos la lógica de negocio.
- Vista: se encarga de presentar una interfaz a los usuarios donde muestre la información almacenada del sistema y ofrezca una serie de operaciones sobre esta.
- Controlador: en esta capa se interpreta las instrucciones recibidas por la interfaz y se comunica las operaciones a la capa modelo.

En la Figura 5.1 página 34 podemos ver un diagrama simplificado de cómo se ha aplicado este patrón en el proyecto. La interfaz web sería la vista, el servidor que a través de sus servlets interpreta las peticiones sería el controlador y por último estaría la capa modelo que es donde se modifica la información persistente.

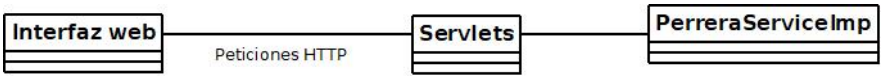


Figura 5.1: Representación simplificada del patrón MVC en el proyecto

## 5.2 Fachada

Es un patrón de diseño estructural que se basa en implementar una interfaz simplificada a un subsistema complejo. Esta fachada limitará las posibles opciones de los usuarios para interactuar con la aplicación. En la Figura 5.2 página 34 podemos ver como desde la clase PerreraServiceImpl controlamos todas las operaciones necesarias.

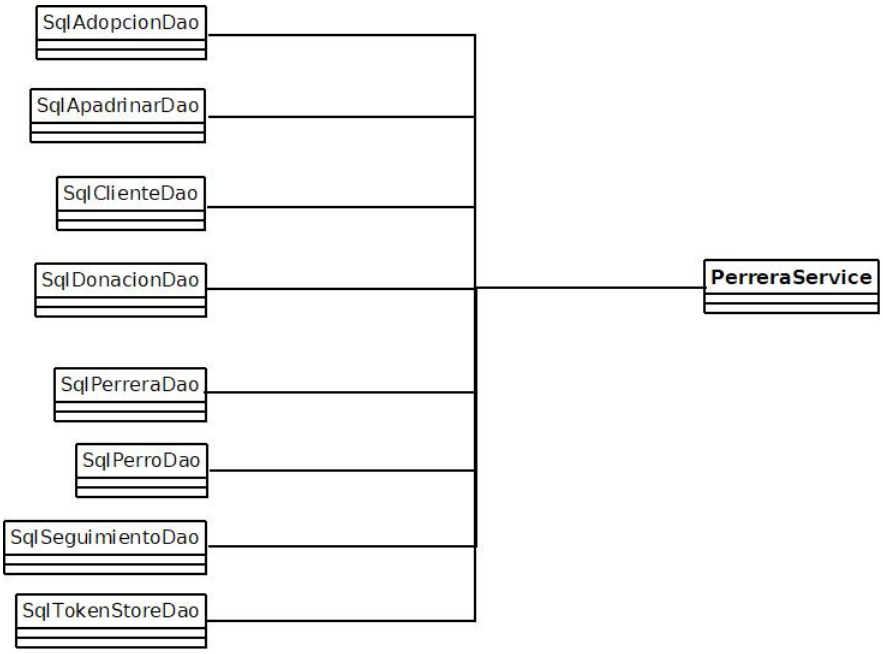


Figura 5.2: Representación simplificada del patrón fachada en el proyecto

## 5.3 Factoría

Es uno de los patrones más usados en Java. Al tener como objetivo facilitar el diseño es considerado un patrón estructural. Tiene como objetivo la definición de una interfaz para crear objetos al funcionar como una superclase, permitiendo que las subclases alteren el tipo de objetos que se crean (Figura 5.3 página 35).

```
public PerreraServiceImpl(){
    dataSource = new SimpleDataSource();
    perroDao = SqlPerroDaoFactory.getDao();
    donacionDao = SqlDonacionDaoFactory.getDao();
    perreraDao = SqlPerreraDaoFactory.getDao();
    clienteDao = SqlClienteDaoFactory.getDao();
    adopcionDao = SqlAdopcionDaoFactory.getDao();
    seguimientoDao = SqlSeguimientoDaoFactory.getDao();
    apadrinarDao = SqlApadrinarDaoFactory.getDao();
    tokenStoreDao = SqlTokenStoreDaoFactory.getDao();
}
```

Figura 5.3: Muestra de ejemplo de como se generan los DAOs desde una clase factoría

## 5.4 Singleton

Es uno de los patrones más sencillo que existen. Es un patrón diseñado para la optimización en la creación de un objeto, por lo que se cataloga como un patrón creacional.

Este patrón asegura la existencia única una instancia de cada clase y provee un acceso único para todas las instancias, es decir, existe una clase encargada de la creación del objeto y de que no se cree ninguna más (Figura 5.4 página 35).

```
public class SqlAdopcionDaoFactory {

    private final static String CLASS_NAME_PARAMETER = "SqlAdopcionDaoFactory.className";
    private static SqlAdopcionDao dao = null;

    private SqlAdopcionDaoFactory() {
    }

    /rawtypes/
    private static SqlAdopcionDao getInstance() {
        try {
            String daoClassName = ConfigurationParametersManager
                    .getParameter(CLASS_NAME_PARAMETER);
            Class daoClass = Class.forName(daoClassName);
            return (SqlAdopcionDao) daoClass.getDeclaredConstructor().newInstance();
        } catch (Exception e) {
            throw new RuntimeException(e);
        }

    }

    public synchronized static SqlAdopcionDao getDao() {

        if (dao == null) {
            dao = getInstance();
        }
        return dao;

    }

}
```

Figura 5.4: Representación del uso del singleton en el proyecto

## 5.5 DAO

El patrón Data Abstract Object (DAO) separa la lógica de negocio de la interactuación con la información almacenada en la base de datos mediante una capa, en ella se crean funciones para traducir la información a objetos mapeados y asegurarse de que los cambios en la base de datos son persistentes. Este patrón es implementado en la capa modelo, donde se definen las operaciones sobre la base de datos (Figura 5.5 página 36).

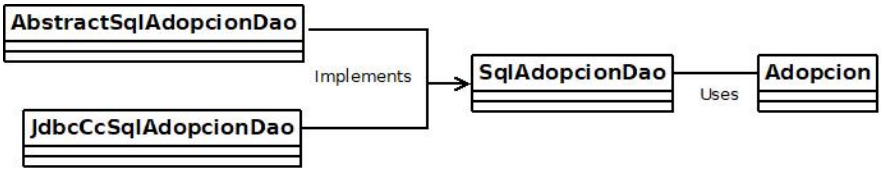


Figura 5.5: Representación simplificada en el proyecto del patrón DAO

## 5.6 DTO

Un Data Transfer Object (DTO) es un objeto cuyo propósito es el de ser intercambiado entre procesos como unidad de información. Esta estrategia se implementa para reducir el tiempo de las llamadas entre capas al enviar toda la información necesaria encapsulada a la vez. Los DTOs no deben de implementar ninguna operación de negocio, tan solo deben de contener los atributos y métodos necesarios para transportar la información (Figura 5.6 página 36).


Figura 5.6: Representación simplificada en el proyecto del patrón DTO

Capítulo 6

# Arquitectura del sistema

En esta sección se expondrá de manera detallada la arquitectura usada en la creación de este proyecto.

## 6.1 Capa Modelo

En el siguiente diagrama podemos observar el conjunto de entidades persistentes que representarán la información en la aplicación (Figura 6.1 página 37).

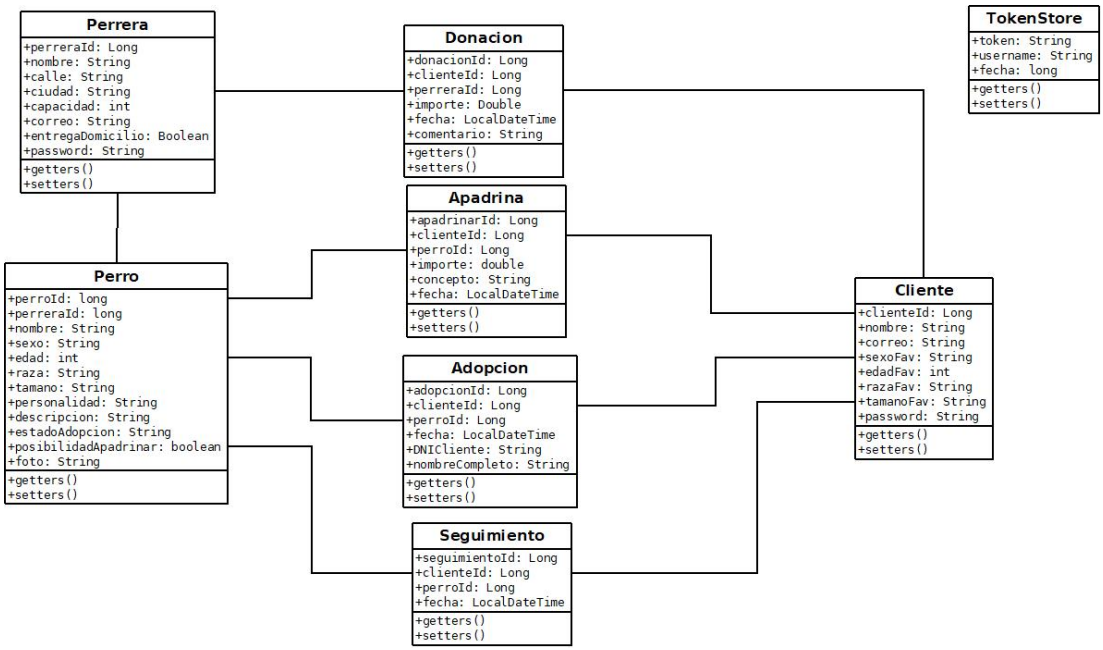


Figura 6.1: Diagrama UML de clases de la capa modelo

Aparte de las entidades principales (Cliente, Perro, Perrera), se ha creado un conjunto de entidades donde quedará reflejada las operaciones realizadas por el cliente, por ejemplo, la

entidad Seguimiento guardará un listado de animales en los que está interesado el cliente.

Además, se creará una entidad, TokenStore, que gestionará el acceso al perfil de los actores a través de tokens de acceso.

**Modelo conceptual de la base de datos**

- Perrera: Representará al refugio. En ella almacenaremos datos necesarios:
  - PerreraId: Identificador del refugio, será autogenerado.
  - Nombre: nombre por el que se identifica el refugio.
  - Calle: dirección del refugio.
  - Ciudad: localidad donde se encuentra la perrera.
  - Capacidad: cantidad de animales que puede albergar.
  - Correo: email de contacto de la perrera y necesario que esté asociado a una cuenta de PayPal para recibir donaciones.
  - Entrega domicilio: posibilidad de entrega del animal en el domicilio del cliente.
  - Password: contraseña establecida del cliente, se almacenará cifrada por seguridad.
- Perro: Representa la ficha de un animal creada por un refugio.
  - PerroId: Identificador del animal, será autogenerado.
  - PerreraId: Id de la perrera que creó la ficha del animal.
  - Nombre: nombre identificativo del animal.
  - Sexo: género del perro.
  - Edad: cantidad de años vividos por el animal, o una aproximación.
  - Raza: grupo que tienen características similares.
  - Tamano: altura aproximada que alcanzará el animal en su madurez.
  - Personalidad: carácter del perro.
  - Descripción: Comentarios de la perrera respecto al animal.
  - EstadoAdopcion: disposición a la hora de adoptar.
  - PosibilidadApadrinar: se establece la facultad de donar dinero para el cuidado del perro.
  - Foto: imagen del animal.
- Cliente: Representa al usuario que va en busca de adoptar un animal.
  - ClienteId: Identificador del usuario, será autogenerado.

  - Nombre: nombre por el que se siente identificado el usuario.
  - Correo: email de contacto del cliente y necesario que esté asociado a una cuenta de PayPal para recibir donaciones.
  - SexoFav: aquí se establecerá la preferencia del usuario respecto al sexo del animal.
  - EdadFav: aquí se establecerá la preferencia del usuario respecto a la edad del perro.
  - RazaFav: aquí se establecerá la preferencia del usuario respecto a la raza del animal.
  - TamanoFav: aquí se establecerá la preferencia del usuario respecto al tamaño del perro.

- Donación: Representa la acción del cliente de apoyar un refugio.
  - DonacionId: identificador de la donación, es autogenerado.
  - PerreraId: Identificador del refugio.
  - ClienteId: Identificador del cliente.
  - Importe: cantidad donada.
  - Fecha: momento cundo se realiza la transacción.
  - Comentario: explicación del usuario.

- Apadrinar: Representa cuando un cliente apadrina un animal.
  - ApadrinarId: identificación del apadrinamiento, es autogenerado.
  - PerroId: Identificador del perro.
  - ClienteId: Identificador del usuario.
  - Importe: cantidad donada.
  - Fecha: momento cundo se realiza la transacción.
  - Concepto: explicación del usuario para que se quiere usar el importe.

- Adopcion: Representa de la adopción de un animal.
  - AdopcionId: identificador de la adopción, es autogenerado.
  - PerroId: Identificador del perro.
  - ClienteId: Identificador del usuario.
  - Fecha: momento cuando se realiza la adopción.
  - NombreCompleto: nombre y apellidos del cliente.
  - DNICliente: número identificativo personal del usuario.

- TokenStore: Representa el acceso del usuario en la aplicación.
  - Token: cadena de texto aleatoria autogenerada.
  - Username: correo del usuario (refugio y cliente).
  - Fecha: momento cuando se realiza el inicio correcto de sesión.

### 6.1.1 Cliente

Esta clase tiene objetivo la representación de un usuario al que le gustaría adoptar o apadrinar un perro según sus gustos, por ello aparte de los datos mínimos se han añadido campos para almacenar sus gustos. Posteriormente estos campos se usarán para notificar al usuario de cuando hay un animal que concuerde con sus especificaciones (Figura 6.2 página 40).

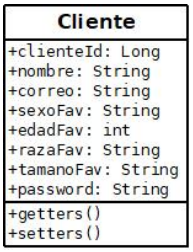


Figura 6.2: Diagrama UML de la clase Cliente

### 6.1.2 Perrera

Esta clase tiene como objetivo representar al actor refugio. Esta entidad tiene los atributos indispensables (Figura 6.3 página 40).

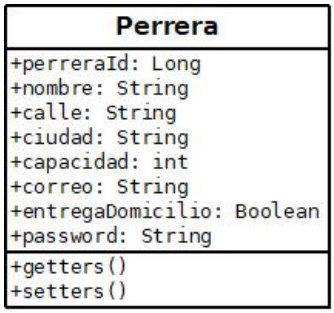


Figura 6.3: Diagrama UML de la clase Perrera

### 6.1.3 Perro

Aquí definiremos las características del animal (Figura 6.4 página 41).


Figura 6.4: Diagrama UML de la clase Perro

### 6.1.4 Donación

Representa la información acerca de las donaciones realizadas por el cliente a los refugios (Figura 6.5 página 41).

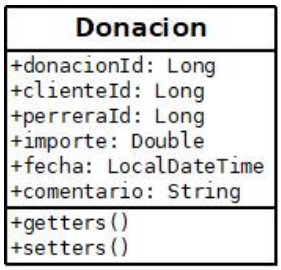


Figura 6.5: Diagrama UML de la clase Donación

### 6.1.5 Apadrinamiento

Aquí almacenaremos la información acerca de los apadrinamientos de los animales por parte de los clientes (Figura 6.6 página 41).

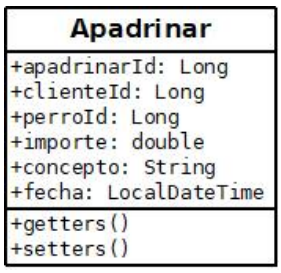


Figura 6.6: Diagrama UML de la clase Apadrinamiento

### 6.1.6 Seguimiento

Representa el momento donde el cliente quiere guardar un animal en su lista de favoritos (Figura 6.7 página 42).


Figura 6.7: Diagrama UML de la clase Seguimiento

### 6.1.7 Adopción

Representa la adopción de un animal por parte de un cliente (Figura 6.8 página 42).

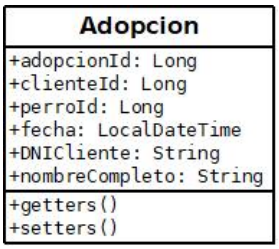


Figura 6.8: Diagrama UML de la clase Adopción

### 6.1.8 TokenStore

Utilizaremos esta clase para almacenar los tokens de inicio de sesión para gestionar las operaciones de los clientes (Figura 6.9 página 42).


Figura 6.9: Diagrama UML de la clase TokenStore

## 6.2 Capa de acceso a los datos

En esta subcapa de la modelo, definiremos las operaciones sobre la base de datos, así como la lógica de negocio. Como se mencionó en el apartado de patrones, se ha implementado siguiendo la arquitectura DAO. Aparte de las operaciones Create, Read, Update, Delete (CRUD), para cada item se han definido otras operaciones. Posteriormente se ha creado un servicio que será el que defina la lógica del sistema (Figura 6.10 página 43).

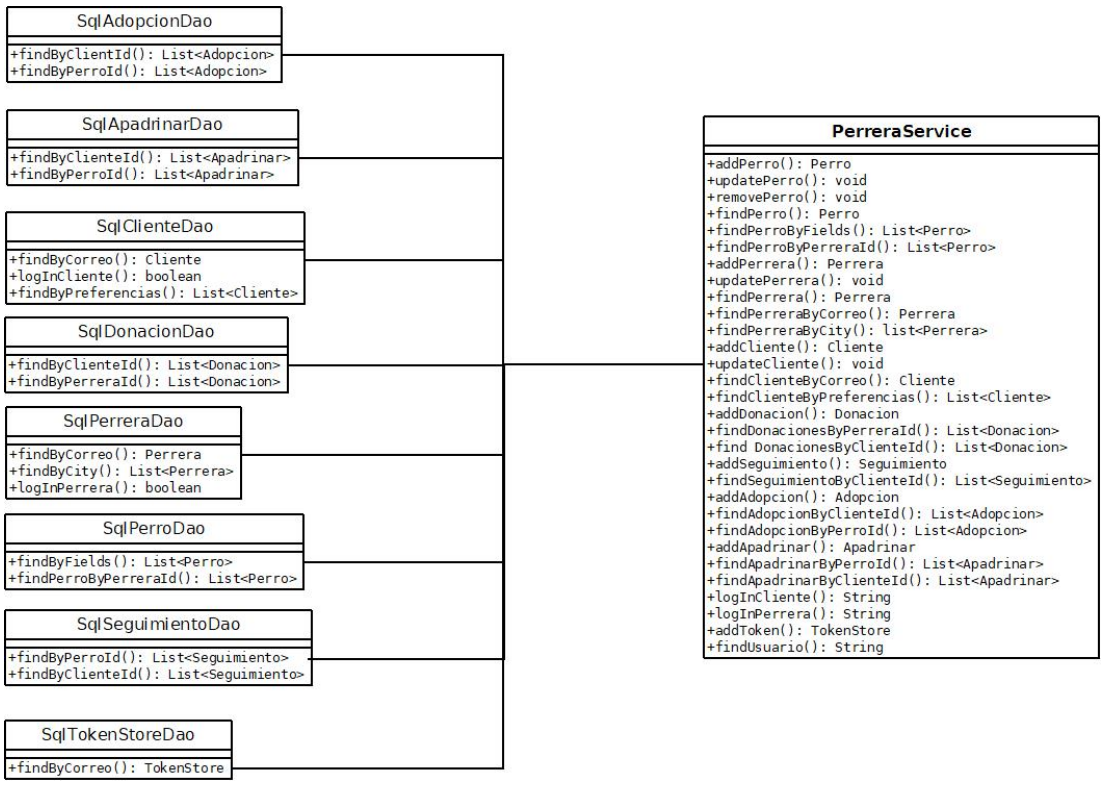


Figura 6.10: Diagrama UML de los DAOS

### 6.2.1 SqlAdoptarDao

En este DAO aparte de las operaciones CRUD se han implementado, dos métodos para realizar búsquedas por parte de los actores (Figura 6.11 página 44).

- FindByClienteId: operación que devuelve el listado de adopciones realizadas por el usuario a través de la aplicación.
- FindByPerroId: función que muestra la relación de adopciones del animal a través de la aplicación.

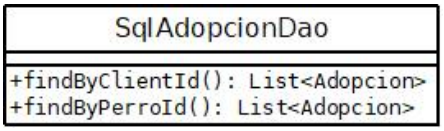


Figura 6.11: Diagrama UML de SqlAdoptarDao

### 6.2.2 SqlApadrinarDao

En este DAO aparte de las operaciones CRUD se han implementado, dos métodos para realizar búsquedas por parte de los actores (Figura 6.12 página 44).

- FindByClienteId: operación que devuelve el listado de apadrinamientosrealizadas por el usuario a través de la aplicación.
- FindByPerroId: función que muestra la relación de apadrinamientos del animal a través de la aplicación.

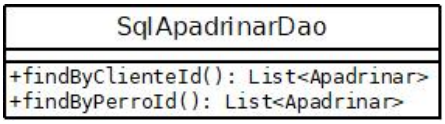


Figura 6.12: Diagrama UML de SqlApadrinarDao

### 6.2.3 SqlClienteDao

Para la gestión del cliente, además de las operaciones CRUD, se han implementado métodos relacionados con el inicio de sesión y sus gustos (Figura 6.13 página 44).

- FindByCorreo: devuelve el cliente cuyo correo es el que se ha introducido.
- LogInCliente: función para loguearse como cliente.
- FindByPreferencias: devuelve el conjunto de animales que cumplen con los requisitos del cliente.

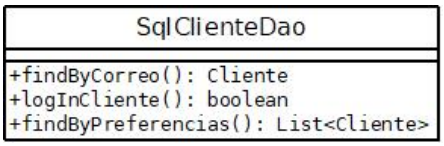


Figura 6.13: Diagrama UML de SqlClienteDao

### 6.2.4 SqlDonacionDao

Definiremos un par de búsquedas para uso de los actores, además de las operaciones CRUD (Figura 6.14 página 45).

- FindByClienteId: operación que devuelve el listado de donaciones realizadas por el usuario a través de la aplicación.
- FindByPerreraId: función que muestra la relación de donaciones recibidas por el refugio a través de la aplicación.

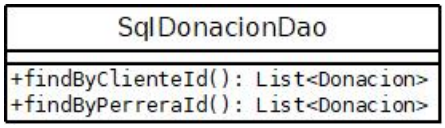


Figura 6.14: Diagrama UML de SqlDonacionDao

### 6.2.5 SqlPerreraDao

Aquí definiremos unas operaciones para la gestión de inicio de sesión de los refugios y una operación que va a usar el usuario para buscar por ciudad (Figura 6.15 página 45).

- FindByCorreo: Devuelve el refugio buscado por su correo.
- FindByCity: función que devuelve la relación de refugios que residen en una determinda ciudad.
- LogInPerrera: operación de iniciar sesión como un refugio.

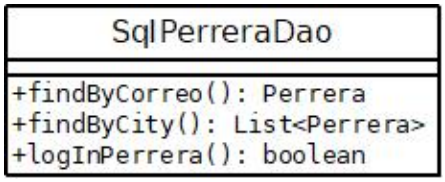


Figura 6.15: Diagrama UML de SqlPerreraDao

### 6.2.6 SqlPerroDao

Definiremos unas operaciones para filtrar la relación de los animales según sus atributos (Figura 6.17 página 46).

- FindByFields: búsqueda de los animales que encajan con los campos de búsqueda especificados.
- FindPerroByPerreraId: devuelve la relación de animales refugiados en la perrera.

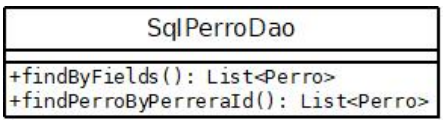


Figura 6.16: Diagrama UML de SqlPerroDao

### 6.2.7 SqlSeguimientoDao

A parte de las operaciones CRUD, programaremos unas funciones para localizar los seguimientos según el identificador del cliente o del perro (Figura **??** página **??**).

- FindByClienteId: operación que devuelve el listado de seguimientos realizadas por el usuario a través de la aplicación.
- FindByPerroId: función que muestra la relación de seguimientos recibidas por el animal a través de la aplicación.

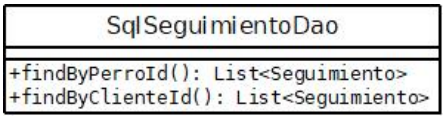


Figura 6.17: Diagrama UML de SqlSeguimientoDao

### 6.2.8 SqlTokenStoreDao

Definiremos una operación que nos ayudará posteriormente con el inicio y mantenimiento de la sesión (Figura 6.18 página 46).

- FindByCorreo: funcion que devuelve el token relacionado con el correo.


Figura 6.18: Diagrama UML de SqlTokenStoreDao

## 6.3 Capa Servicio

Desde esta capa se llamarán a los distintos daos para realizar las operaciones. Esta clase definirá la lógica de negocio de nuestra aplicación.

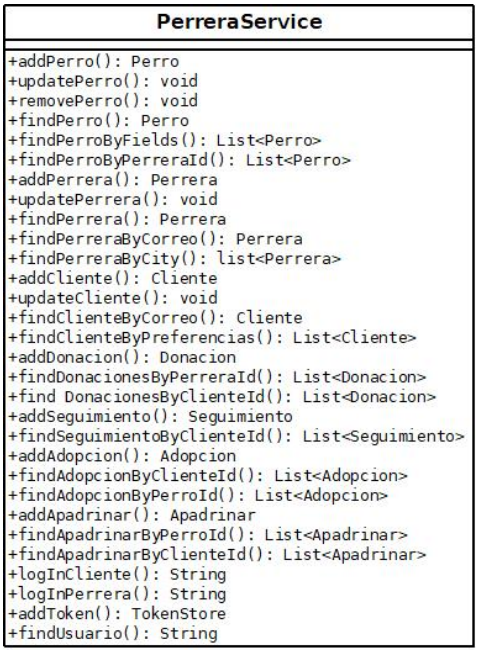


Figura 6.19: Diagrama UML del servicio

Aquí definimos todas las operaciones necesarias como son las creaciones, búsquedas, actualizaciones y eliminaciones de los datos persistentes (Figura 7.3 página 58) .

- AddPerro : El refugio añade un animal.
- UpdatePerro: Se actualiza los datos del perro.
- RemovePerro: Se borra el animal de la base de datos.
- FindPerro: Devuelve el animal cuyo animal coincide con el ID introducido.
- FindPerroByFields: Se busca los animales que coincidan con los campos introducidos.
- FindPerroByPerreraId: Devuelve los animales de una perrera.
- AddPerrera: Se añade una perrera.
- UpdatePerrera: Se actualiza los datos de un refugio.
- FindPerreraByCorreo: Se busca un refugio por su correo.

- FindPerreraByCity: Se busca los refugios ubicados en una ciudad.
- AddCliente: Se añade un cliente.
- UpdateCliente: Se actualiza los datos del cliente.
- FindClienteByCorreo: Se busca un usuario por su correo.
- FindClienteByPreferencias: Se busca los clientes con gustos similares.
- AddDonacion: Se añade una donación.
- FindDonacionesByPerreraId: Devuelve las donaciones recibidas por el refugio.
- FindDonacionesByClienteId: Devuelve las donaciones realizadas por el cliente.
- AddSeguimiento: El cliente añade un perro a su lista de favoritos.
- FindSeguimientoByClienteId: Devuelve los animales favoritos de un cliente.
- AddAdopcion: Se añade una adopción.
- FindAdopcionByClienteId: Devuelve las adopciones realizadas por el cliente.
- FindAdopcionByPerroId: Devuelve las veces que un perro ha sido adoptado.
- AddApadrinar: Se realiza un apadrinamiento.
- FindApadrinarByClienteId: Devuelve los apadrinamientos realizados por el cliente.
- FindApadrinarByPerroId: Devuelve las veces que un perro ha sido apadrinado.
- LogInCliente: Un usuario inicia sesión.
- LogInPerrera: Un refugio inicia sesión.
- LogOut: Se cierra la sesión.
- AddToken: Se crea un token al inicio de sesión correcto.
- FindUsuario: Devuelve el token activo de un usuario.

## 6.4 Capa Controlador

Esta capa tendrá como función traducir las operaciones del cliente lanzadas desde la vista para ser ejecutadas en la capa modelo. Se implementará un servicio web REST, donde cada petición desde la interfaz irá a una URL determinada donde habrá un servlet configurado esperando las peticiones http (POST, PUT, GET, DELETE) que las convertirá en instrucciones para la capa modelo. Además, configuraremos el servidor Jetty para que acepte las peticiones desde la interfaz web que más tarde implementaremos.

Los servlets harán uso de la tecnología JSON para mapear los objetos recibidos y traducirlos a diferentes DTOs, uno por cada clase de la capa modelo. Estos DTOs serán usados como forma de intercambiar información entre las subcapas, permitiendo una mayor independencia.

A continuación, se explicará las funcionalidades de los servlets.

### 6.4.1 AdopcionServlet (/adopciones/)

| /adopciones/ | |
|---|---|
| Descripción | Añade una adopción. |
| Tipo petición HTTP | POST |
| Parámetros | - |

| /adopciones?clienteId={clienteId} | |
|---|---|
| Descripción | Búsqueda de todas las adopciones realizadas por el cliente. |
| Tipo petición HTTP | GET |
| Parámetros | clienteId |

| /adopciones?perroId={perroId} | |
|---|---|
| Descripción | Búsqueda de todas las veces que un perro fue adoptado por los clientes. |
| Tipo petición HTTP | GET |
| Parámetros | perroId |

### 6.4.2 ApadrinamientoServlet (/apadrinamientos/)

| /apadrinamientos/ | |
|---|---|
| Descripción | Añade un apadrinamiento. |
| Tipo petición HTTP | POST |
| Parámetros | - |

| /apadrinamientos/apadrinamientos?clienteId={clienteId} | |
|---|---|
| Descripción | Búsqueda de todos los apadrinamientos realizados por el cliente. |
| Tipo petición HTTP | GET |
| Parámetros | clienteId |

| /apadrinamientos/apadrinamientos?perroId={perroId} | |
|---|---|
| Descripción | Búsqueda de todas las veces que un perro fue apadrinado por los clientes. |
| Tipo petición HTTP | GET |
| Parámetros | perroId |

### 6.4.3 ClienteServlet (/clientes/)

| /clientes/ | |
|---|---|
| Descripción | Añade un cliente. |
| Tipo petición HTTP | POST |
| Parámetros | - |

| /clientes?correo={correo} | |
|---|---|
| Descripción | Busca un cliente por su ID. |
| Tipo petición HTTP | GET |
| Parámetros | correo |

| /clientes?clienteId={clienteId} | |
|---|---|
| Descripción | Actualiza los parámetros de un cliente. |
| Tipo petición HTTP | PUT |
| Parámetros | clienteId |

### 6.4.4 DonacionServlet (/donaciones/)

| /donaciones/ | |
|---|---|
| Descripción | Añade una donacion. |
| Tipo petición HTTP | POST |
| Parámetros | - |

| /donaciones?perreraId={perreraId} | |
|---|---|
| Descripción | Filtra las donaciones por el ID de un refugio. |
| Tipo petición HTTP | GET |
| Parámetros | perreraId |

| /donaciones?clienteId={clienteId} | |
|---|---|
| Descripción | Filtra las donaciones por el ID de un cliente. |
| Tipo petición HTTP | GET |
| Parámetros | clienteId |

### 6.4.5 PerreraServlet (/perreras/)

| /perreras/ | |
|---|---|
| Descripción | Añade un refugio. |
| Tipo petición HTTP | POST |
| Parámetros | - |

| /perreras?perreraId={perreraId} | |
|---|---|
| Descripción | Actualización de los atributos de un refugio. |
| Tipo petición HTTP | PUT |
| Parámetros | perreraId |

| /perreras?perreraId={perreraId} | |
|---|---|
| Descripción | Búsqueda de un refugio por su ID. |
| Tipo petición HTTP | GET |
| Parámetros | perreraId |

| /perreras?ciudad={ciudad} | |
|---|---|
| Descripción | Búsqueda de los refugios presentes en una ciudad. |
| Tipo petición HTTP | GET |
| Parámetros | ciudad |

| /perreras?correo={correo} | |
|---|---|
| Descripción | Búsqueda de un refugio por su correo. |
| Tipo petición HTTP | GET |
| Parámetros | correo |

### 6.4.6 PerreraServlet (/perro/)

| /perros/ | |
|---|---|
| Descripción | Añade un perro. |
| Tipo petición HTTP | POST |
| Parámetros | - |

| /perros?perroId={perroId} | |
|---|---|
| Descripción | Actualiza los parámetros de un animal. |
| Tipo petición HTTP | PUT |
| Parámetros | perroId |

| /perros?perroId={perroId} | |
|---|---|
| Descripción | Realiza la búsqueda de un animal cuyo ID corresponde con el parámetro. |
| Tipo petición HTTP | GET |
| Parámetros | perroId |

| /perros?perreraId={perreraId} | |
|---|---|
| Descripción | Realiza la búsqueda todos los animales pertenecientes a un refugio. |
| Tipo petición HTTP | GET |
| Parámetros | perreraId |

| /perros?edad={edad}&sexo={sexo}&raza={raza} &tamano={tamano}&personalidad={personalidad} | |
|---|---|
| Descripción | Realiza la búsqueda todos los animales que coinciden con las características deseadas. |
| Tipo petición HTTP | GET |
| Parámetros | Edad, sexo, raza, tamaño y personalidad deseadas por el cliente. |

| /perros?perroId={perroId} | |
|---|---|
| Descripción | Elimina el perro cuyo ID coincida. |
| Tipo petición HTTP | DELETE |
| Parámetros | perroId |

### 6.4.7 SeguimientoServlet (/seguimientos/)

| /seguimientos/ | |
|---|---|
| Descripción | Añade un seguimiento. |
| Tipo petición HTTP | POST |
| Parámetros | - |

| seguimientos?clienteId=clienteId | |
|---|---|
| Descripción | Realiza la búsqueda de los seguimientos realizados por el cliente. |
| Tipo petición HTTP | GET |
| Parámetros | clienteId |

### 6.4.8 TokenStoreServlet (/tokens/)

| /tokens/ | |
|---|---|
| Descripción | Crea un token de inicio de sesión. |
| Tipo petición HTTP | POST |
| Parámetros | - |

| /tokens?tokenCliente={token} | |
|---|---|
| Descripción | Confirma la sesión activa de un cliente. |
| Tipo petición HTTP | GET |
| Parámetros | Token de un cliente. |

| /tokens?tokenPerrera={token} | |
|---|---|
| Descripción | Confirma la sesión activa de un refugio. |
| Tipo petición HTTP | GET |
| Parámetros | Token de un refugio. |

| /tokens?username={username} | |
|---|---|
| Descripción | Cierra la sesión del usuario. |
| Tipo petición HTTP | DELETE |
| Parámetros | Correo del usuario. |

## 6.5 Vista

Para la creación de la interfaz, se ha optado por la creación de una página web que hará las peticiones a nuestra aplicación REST. Para la construcción del sitio web se ha usado el conocido lenguaje HTML combinado con CSS para darle forma. También se ha usado JavaScript para crear las funcionalidades necesarias. Para facilitar la implementación se optado por el uso del framework Angular. Esta herramienta está basada en una arquitectura formada por componentes, encapsulando en cada uno una vista y unos servicios llamados desde los componentes de las vistas, permitiendo un mayor modularidad.

En el diseño de las vistas se ha usado las librerías de Bootstrap para diseñar más rápido, fácil y responsive los componentes de las páginas web.

Hemos usado la librería i18n de angular para facilitar la internacionalización de nuestra web. Actualmente está disponible en 3 idiomas: español, gallego e inglés

### 6.5.1 Componentes de la vista

- Area-clientes: componente que dispondrá las funcionalidades de registrarse o iniciar sesión como un usuario.
- Area-refugios: componente que dispondrá las funcionalidades de registrarse o iniciar sesión como un refugio.
- Detalles-adopcion: componente que ejecutará la función de adopción.
- Detalles-apadrinamiento: componente que ejecutará la función de apadrinar un animal.
- Detalles-donacion: componente que ejecutará la función de donación.
- Detalles-perro: componente que muestra la información detallada del perro.
- Funciones-cliente: componente que dispondrá al usuario sus funcionalidades.
- Funciones-refugio: componente que dispondrá al refugio sus funcionalidades.

Para el desarrollo de la interfaz se ha optado por un diseño simple y que transmita tranquilidad. A continuación se mostrarán unos pequeños ejemplos de la aplicación:

- En la Figura 6.20 página 55 vemos como es lo que un cliente vería cuando quisiera crearse una cuenta o loguearse. En el caso de la creación de un nuevo usuario se le pregunta de manera opcional por sus preferencias, de esta manera desde el principio podrá recibir los correos automáticos.
- En la Figura 6.21 página 55 podemos ver la interfaz del cliente, en específico el buscador de animales. A la izquierda se pueden filtrar por las características buscadas, a la derecha aa un buscador por un identificador que le llegaría al cliente de un perro que se ajuste a sus gustos.
- En la Figura 6.22 página 55 podemos ver la interfaz que presenta las múltiples herramientas de las que dispone un refugio. A la hora de añadir un perro, al tiempo que se van rellenando los campos del formulario a la izquierda se va actualizando su ficha.

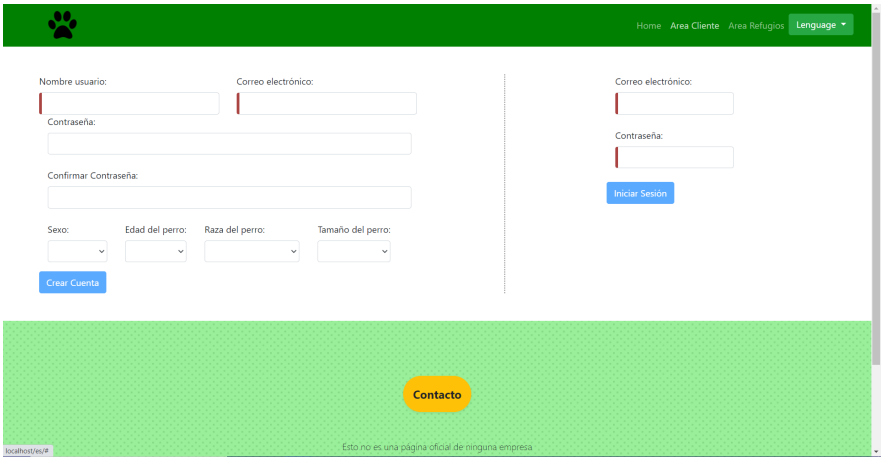


Figura 6.20: Visualización de la pantalla de crear cuenta o inicio de sesión

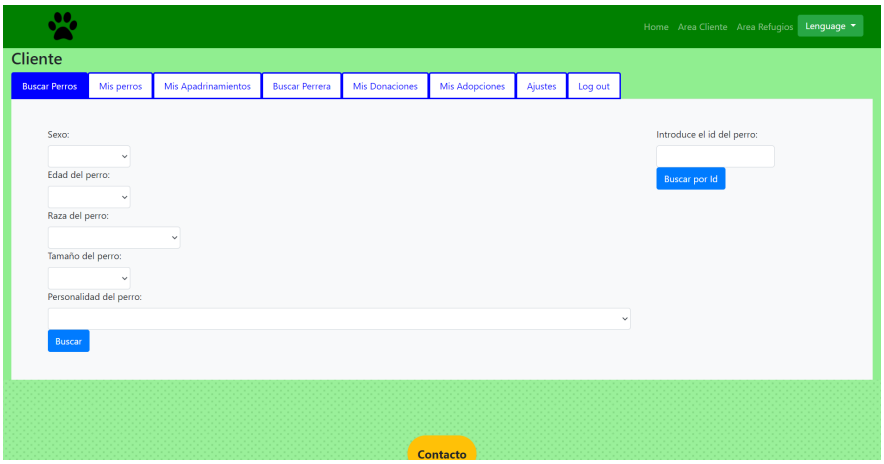


Figura 6.21: Funciones del cliente

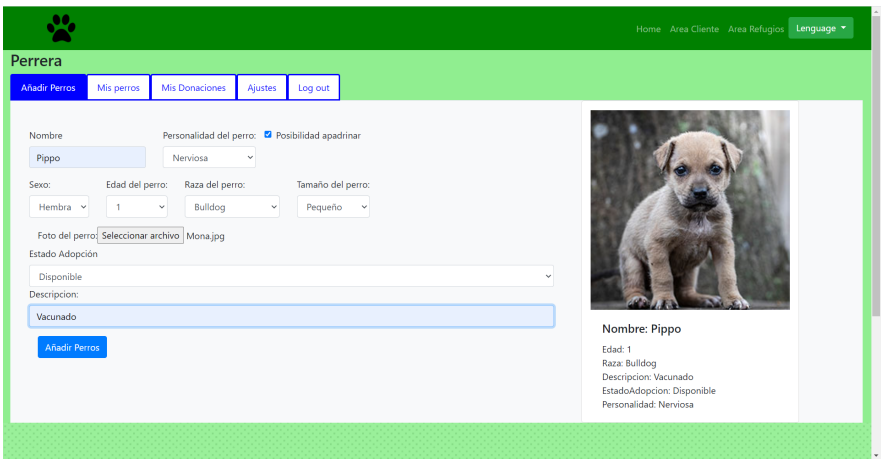


Figura 6.22: Funciones del refugio

Capítulo 7

# Implementación

## 7.1 Prerrequisitos software

Para la implementación de la aplicación se han usado las siguientes versiones del software:

- Intellij IDEA version: ultimate 2020.3.2.
- Java OpenJDK version: 11.0.10.
- Sql version: MySqlServer 8.0.
- Apache Maven: 3.6.3.
- Bootstrap version: 5.0.1.
- Angular CLI version: 12.0.2.
- Node.js version: 14.17.0.
- Package Mangager npm version: 6.14.13.
- Docker Engine version: v20.10.7.

## 7.2 Estructura del proyecto

En este apartado explicaremos la estructura tanto del servidor y el cliente, siguiendo la arquitectura explicada anteriormente.

### 7.2.1 Modelo

En la Figura 7.1 página 58, se puede observar la estructura del modelo, como hemos dicho antes se han diseñado en esta capa las entidades necesarias para satisfacer los requisitos del

proyecto. En cada una de ellas (Figura 7.2 página 58) se ha encapsulado sus operaciones que serán llamadas desde la clase PerreraServiceImpl (Figura 7.3 página 58).

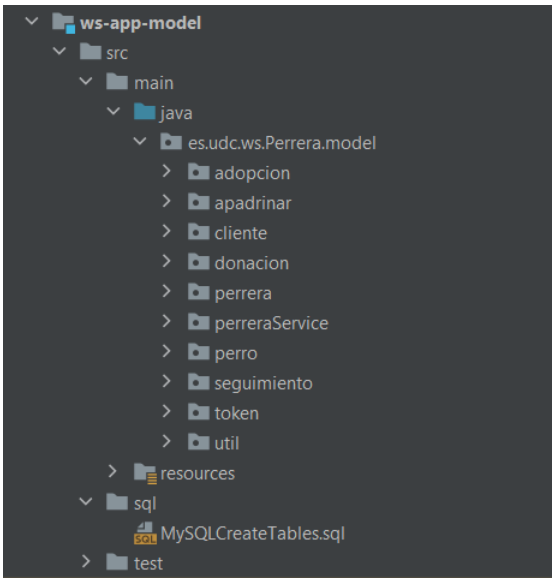


Figura 7.1: Capa Modelo

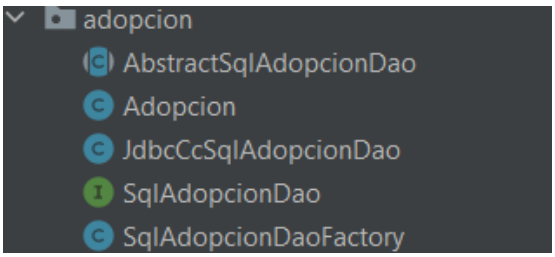


Figura 7.2: Ejemplo de entidad

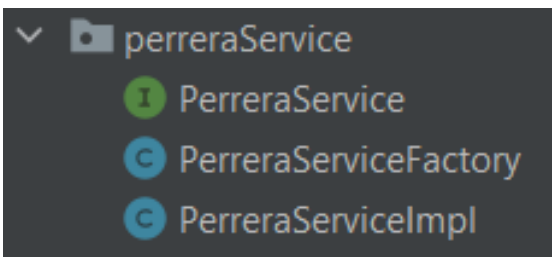


Figura 7.3: Servicio

### 7.2.2 Servidor

En la Figura 7.4 página 59, vemos como se ha estructurado el servidor. Aparte de la creación de los servlets, hemos generado tanto los dtos como los conversores para facilitar la comunicación con la vista. Además, en esta capa tambien implentaremos la autenificación y cifrado de las contraseñas, así como, la automatización de alertas a través de JavaxMail.

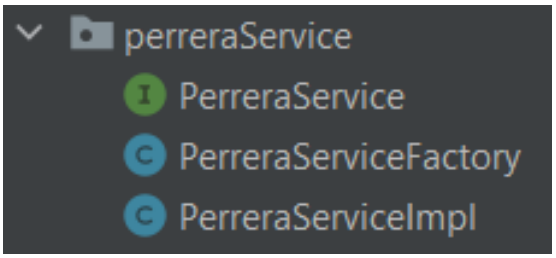


Figura 7.4: Servidor

### 7.2.3 Vista

En la Figura 7.5 página 60, se muestra la estructura de la interfaz de la aplicación generada en Angular. Aquí se configura la pasarela de pago PayPal pare realizar tanto donaciones a perreras como apadrinamientos.

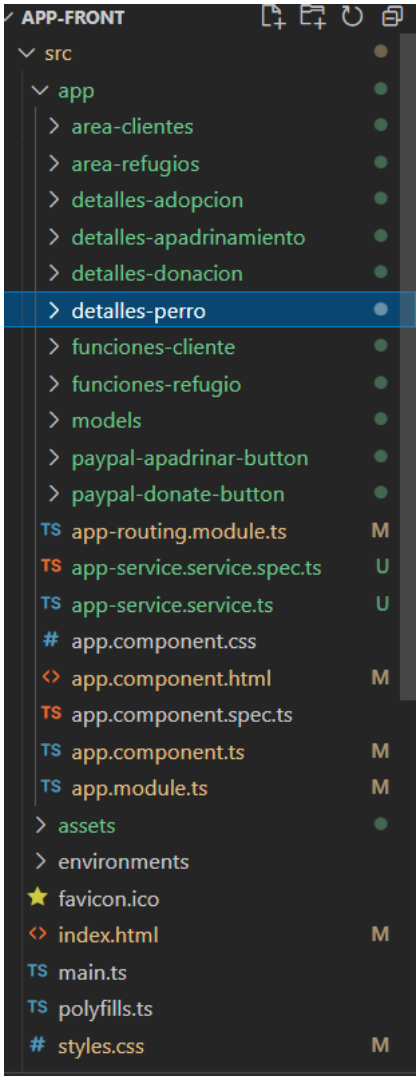


Figura 7.5: Vista

## 7.3 Docker

Para una mayor facilidad a la hora de desplegar la aplicación se ha optado por usar esta tecnología en auge recientemente. Esta plataforma ofrece la posibilidad de empaquetar aplicaciones en entornos aislados, llamados contenedores. Estos contienen el software mínimo necesario para la ejecución del programa a través de la virtualización reusables en múltiples sistemas operativos.

Docker mejora la eficiencia y la rapidez al eliminar la capa de virtualización del sistema operativo. También asegura al compartir el contenedor el mismo funcionamiento de la aplicación para todos (Figura 7.6 página 61).

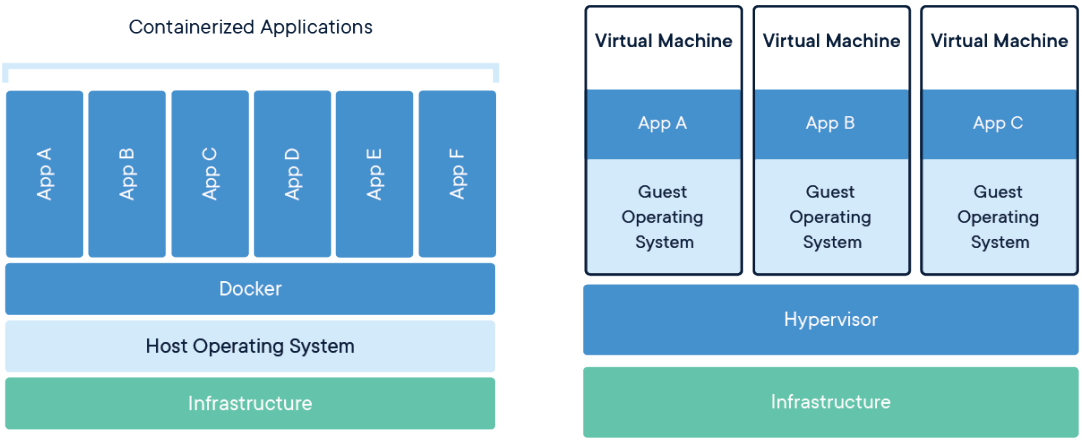


Figura 7.6: Comparativa entre Docker y virtualización

En caso del proyecto se ha optado por la creación de dos contenedores diferentes, el primero albergará la base de datos con el servidor jetty, mientras el segundo ejecutará la interfaz web. A través de Docker-Compose podemos encapsular ambos contenedores y configurar las conexiones entre ellos.

En cada DockerFile se ha creado lo contenedores usando distintas capas, cada línea del fichero. Esto permite que cada vez que se modifique el fichero docker comprueba que líneas se han modificado e implanta las nuevas capas. También en este fichero exponemos los puertos necesarios para las conexiones que luego se usarán para conectar ambos contenedores.

## 7.4 Docker-Compose

Docker-Compose permite enlazar distintos contenederos, ofreciendo herramientas para la creación de redes internas y el uso compartido de volúmenes. Esta tecnología permite el desarrollo modular de grandes aplicaciones, además de las ventajas obvias permite una optmización de los recursos, se pueden ir ajustando los contenedores desplegados según el tipo

y el tamaño de la carga. También permite la disminución de perdida de servicio ante las actualizaciones.

## 7.5 Instrucciones de uso

Para la ejecución del programa es necesario comprobar que el sistema donde lo estemos ejecutando no tengo ocupados los puertos 3306 y el 80, que serán los usados por el gestor de base de datos y la interfaz en el localhost.

Para la ejecución correcta del proyecto primero hemos de iniciar la base de datos antes del resto del programa. Los comandos a ejecutar son los siguientes:

- “docker-compose up –d db”
- “docker-compose up”
- Abrir en un navegador la url: ”localhost/es”

Para actuar como un usuario y un refugio a la vez es necesario usar dos navegadores distintos ya que ciertos elementos que el programa necesita estarán almacenados en el local store y puede dar lugar a fallos al mezclar la información.

## 7.6 Pruebas

Las pruebas son fundamentales en cualquier desarrollo de un proyecto. Estas validan el trabajo en sus dferentes etapas. El objetivo de las pruebas no consiste en comprobar que funciona correctamente si no en encontrar errores para que sean corregidos.

Como queda establecido, al seguir una metodología scrum, al final de cada sprint es necesaria realizar comprobaciones para asegurar el correcto funcionamiento de las funcionalidades implementadas.

## 7.7 Pruebas unitarias

Estas pruebas son las encargadas de verificar que cada una de las funcionalidades cumpla con todos los objetivos. Estas pruebas se realizaron sobre todas las entidades y operaciones en la capa modelo, verificando así unos cimientos robustos. Para la generación de estas pruebas se usa JUnit 5, un framework que facilita la evaluación correcta del comportamiento específico de los métodos de una clase (Figura 7.7 página 63).

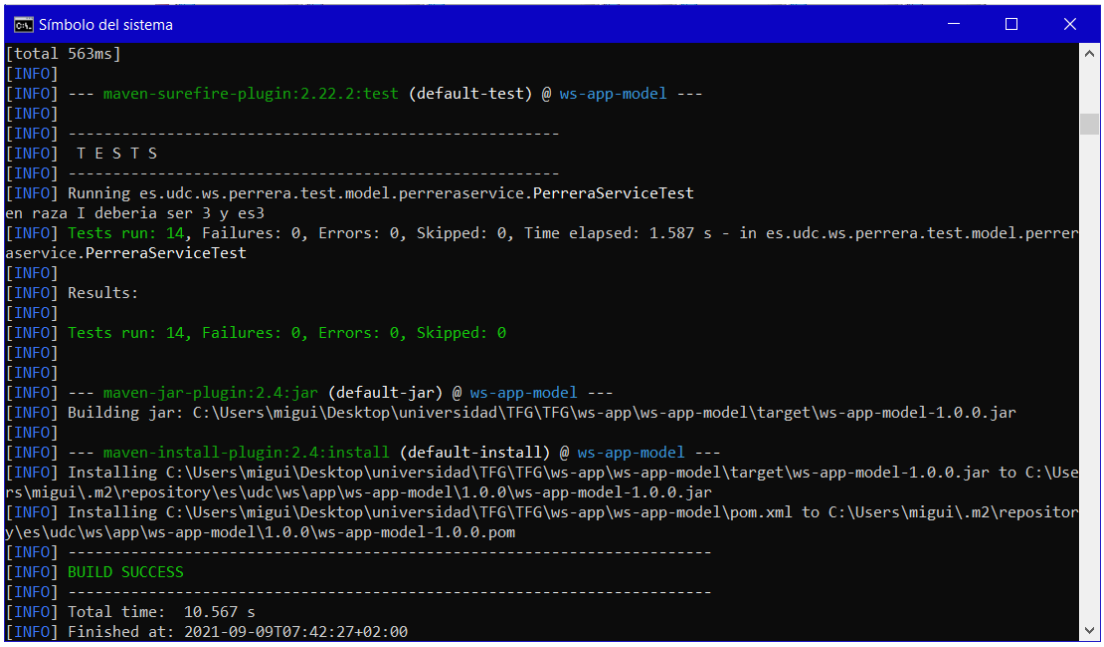


Figura 7.7: Comprobación de los tests

## 7.8 Pruebas de integración

Estas pruebas son las encargadas de detectar errores de interacción entre las distintas capas. También verificará la comunicación de distintos componentes:

- La correcta comuncicación con Javax Mail para la correcta generación automática de correos.
- La comunicación entre el servidor y Postman.
- La comunicación entre la base de datos y la aplicación.
- La comunicación entre el backend y frontend.
- La comunicación con la plataforma de pago Paypal

## 7.9 Pruebas de sistema

Aquí ponemos a prueba la correcta interacción de la aplicación desde el punto de vista de todos los usuarios y que se devuelva la información pertinente. También se comprueba la seguridad de la aplicación con los tokens autogenerados y el cifrado de contraseñas.

## 7.10 Pruebas de aceptación

En estas pruebas hemos comprobado la seguridad de la aplicación, verificando el correcto cifrado de la contraseña y de que esta se alamacene así. También se ha comprobado que el diseño de la interfaz web se acomode a diferentes tamaños de pantalla, dispositivos y navegadores web.

# Capítulo 8

# Conclusiones

El proyecto cumple con todos los requisitos indicados en el anteproyecto. Se ha creado una aplicación web REST que proporciona las herramientas necesarias para facilitar la labor de los refugios de animales, a la vez también ayuda a las personas que desean adoptar una mascota que se adapte a ellos, solventando así uno de los motivos más frecuentes en el abandono de los animales.

Se ha intentado realizar una interfaz lo más cercana posible a una aplicación real usando el framework de Angular con la ayuda de Bootstrap.

Al utilizar Docker, he asegurado el mismo funcionamiento de la aplicación en distintos sistemas operativos sin la necesidad de instalar programas adicionales.

Tras realizar este proyecto he aprendido nuevas herramientas y técnicas, además de adquirir una mayor profundidad de las habilidades aprendidas durante la carrera.

En este proyecto se ha intentado solucionar los problemas que tenían otras aplicaciones o portales web de la misma temática. La mayoría de ellos no ofrecía la posibilidad de realizar donaciones o apadrinamientos. Existen usuarios solidarios que no tienen la posibilidad de adoptar un animal pero quieren ayudar a la causa.

He comparado mi proyecto con una de las últimas apliaciones publicadas en la PlayStore, llamada AdoptaMe, trabajo de fin de grado de un alumno español, pienso que mi trabajo cubre los defectos de la app como la falta de filtros de búsqueda más específicos y la ausencia de poder ayudar de manera económica.

## 8.1 Futuras líneas de trabajo

Este proyecto sirve como una buena base, a partir de la cuál se puede ir implementando más funcionalidades o mejorar las existentes. Algunas de las posibles mejoras podrían ser:

- La inclusión dentro del portal de diferentes animales, como por ejemplo gatos, roedores, reptiles, etc.

- La posibilidad de incluir una provincia o municipio en el filtro de búsqueda y como un campo para las notificaciones automáticas a los usuarios.
- Inclusión de las redes sociales, ejemplo: la posibilidad de hacer una publicación automática en el facebook del refugio cuando este añade un nuevo animal para adoptar.
- Añadir un tercer tipo de usuario, como aquel que quiera ejercer como casa de acogida temporal.
- La inclusión de otras opciones de pago como Bizum, Stripe, etc.
- Ampliar la internacionalización de la web para poder llegar a un público mayor.

# Lista de acrónimos

**ANSI** American Normalization and Standarization Institute. 10

**CHSS** Cascading HTML Style Sheets. 9

**CRUD** Create, Read, Update, Delete. 43–46

**CSS** Cascading Style Sheets. 10, 12, 53

**DAO** Data Abstract Object. 33, 36, 43, 44

**DTO** Data Transfer Object. 33, 36, 49

**FAPAM** Fundación Affinity y las Asociaciones Protectoras y de Defensa Animal. 3

**HTML** HyperText Markup Language. 9, 10, 12, 53

**HTTP** Hypertext Transfer Protocol. 11

**IDE** Integrated development environments. 13

**ISO** International Standarization Organism. 10

**JDBC** Java DataBase Connectivity. 11

**JDK** Java Development Kit. 9

**JRE** Java Runtime Environment. 9

**JVM** Java Virtual Machine. 9

**MVC** Modelo Vista Controlador. v, 15, 33, 34

**PHP** Hypertext Preprocessor. 13

**SQL** Structured Query Lenguage. 10–12

**SSP** Stream-based Style Sheet Proposal. 9, 10

**W3C** World Wide Consorcium. 9

**XML** Extensible Markup Language. 10

# Bibliografía


[1] "Noticia sobre las causas y posibles soluciones," 2021. [En línea]. Disponible en: https://www.lavanguardia.com/participacion/cartas/20210630/7564421/causas-soluciones-abandono-mascotas.html

[2] "Página web oficial de salvaunamascota.es," 2021. [En línea]. Disponible en: https://www.SalvaUnaMascota.es

[3] "Enlace a la aplicación de adoptame," 2021. [En línea]. Disponible en: https://play.google.com/store/apps/details?id=com.adoptame&hl=es_419&gl=US

[4] "Cidade amigable cos animais," 2021. [En línea]. Disponible en: https://www.coruna.gal/mascotas/es

[5] "The java language especication," 2021. [En línea]. Disponible en: https://docs.oracle.com/javase/specs/jls/se15/html/index.html

[6] "Documentación oficial de html y css," 2021. [En línea]. Disponible en: https://www.w3.org/standards/webdesign/htmlcss

[7] "Página web oficial de javascript," 2021. [En línea]. Disponible en: https://www.javascript.com

[8] "Qué es jdbc." [En línea]. Disponible en: https://www.ibm.com/support/knowledgecenter/es/SSGU8G_11.70.0/com.ibm.jdbc_pg.doc/ids_jdbc_011.html

[9] "Apache," 2021. [En línea]. Disponible en: https://httpd.apache.org/docs/

[10] "Documentación oficial de jetty," 2021. [En línea]. Disponible en: https://www.eclipse.org/jetty/documentation.php

[11] "Documentación oficial de javax.mail," 2021. [En línea]. Disponible en: https://docs.oracle.com/javaee/7/api/javax/mail/package-summary.html

[12] "Documentación oficial de bootstrap," 2021. [En línea]. Disponible en: https://getbootstrap.com/docs/5.0/

[13] "Documentación oficial de angular," 2021. [En línea]. Disponible en: https://angular.io/docs

[14] "Documentación oficial de mysql," 2021. [En línea]. Disponible en: https://dev.mysql.com/doc/

[15] "Documentación oficial de intellij idea," 2021. [En línea]. Disponible en: https://www.jetbrains.com/help/idea/discover-intellij-idea.html

[16] "Documentación oficial de maven," 2021. [En línea]. Disponible en: https://maven.apache.org/guides/

[17] "Documentación oficial de docker," 2021. [En línea]. Disponible en: https://docs.docker.com/

[18] "Página web oficial de scrum," 2021. [En línea]. Disponible en: https://www.scrum.org/resources/what-is-scrum

[19] "Boletín oficial del estado núm. 251, de 18 de octubre de 2019, páginas 114772 a 114801." 2019. [En línea]. Disponible en: https://www.boe.es/diario_boe/txt.php?id=BOE-A-2019-14977

[20] E. Freeman and E. Robson, *Head First Design Patterns: Building Extensible and Maintainable Object-Oriented Software*, 2nd ed. O'Reilly Media, 2021.