UNIVERSIDAD MIGUEL HERNÁNDEZ DE ELCHE

ESCUELA POLITÉCNICA SUPERIOR DE ELCHE

GRADO EN INGENIERÍA INFORMÁTICA EN
TECNOLOGÍAS DE LA INFORMACIÓN

DESARROLLO APLICACIÓN WEB PARA LA ASOCIACIÓN PATAS SIN FRONTERAS

TRABAJO FIN DE GRADO

Febrero - 2026

AUTOR: Antonio Angosto Magdaleno
DIRECTOR/ES: Alejandro Martínez Gracia

RESUMEN

El trabajo realizado consiste en una aplicación web desarrollada con el *framework* Angular enfocada principalmente en la difusión de la adopción de animales, la captación de nuevos colaboradores, la obtención ayuda gracias a donaciones y a la gestión interna de la asociación mediante la digitalización y automatización de gran parte de los procesos recurrentes y repetitivos.

Al tratarse de una asociación, se ha hecho uso de herramientas *Open Source* y otras de bajo coste para que no repercutan negativamente en la misión principal de la asociación, que es destinar todos los recursos disponibles a los animales.

AGRADECIMIENTOS

Quiero agradecer a Blanca, mi pareja, por motivarme a dar ese último esfuerzo para concluir esta etapa universitaria, así como por todo el apoyo y la ayuda que me ha brindado para terminar este trabajo.

ÍNDICE GENERAL

Contenido

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

Capítulo 1

Introducción

1.1.- EMPRESA/ENTORNO DE APLICACIÓN (ASOCIACIÓN DE BIENESTAR ANIMAL “PATAS SIN FRONTERAS”)

Este Trabajo de Fin de Grado (TFG) se basa en el desarrollo de una plataforma web funcional para la Asociación de Bienestar Animal “Patas sin Fronteras”, una entidad sin ánimo de lucro registrada en Crevillente (Alicante).

Esta asociación, que en 2025 celebra su décimo aniversario, se encuentra instituida en el Sector de Bienestar Animal y su principal labor es la asistencia en rescates, atención veterinaria, gestión de colonias felinas en el municipio de Crevillent, y gestión de las adopciones de los animales a su cargo.

Paralelamente, “Patas sin fronteras” dedica gran parte de su tiempo en realizar actividades para la captación de fondos y recursos, para poder costear tanto las facturas veterinarias, como la manutención de todos estos animales rescatados. Entre estas actividades, se encuentran campañas como recogidas de alimentos, puntos de venta en mercadillos del municipio de Crevillent, ferias de la adopción, o en contraposición, la venta de artículos de diseño propio, como pueden ser los calendarios y agendas.

La asociación opera en un entorno de personas voluntarias con un escaso tiempo debido a la alta demanda social de su labor. Esta realidad subraya que cualquier ayuda para la gestión favorecerá significativamente el mejor funcionamiento de la entidad. Este hecho además se contrasta con una limitación de recursos económicos para poder realizar la contratación de un profesional para el desarrollo de una plataforma web, obligando a la organización a depender únicamente de sus redes sociales.

El desarrollo de esta aplicación web se debe, por tanto, a la necesidad de digitalización de este sector, que de forma histórica se ha organizado de forma manual, mediante documentación en físico, sin fácil trazabilidad, organización de voluntarios, contratos de acogida y de adopción, y la gestión de recursos.

El tema principal de esta plataforma es la implementación de una solución tecnológica de fácil manejo para un sector que no está altamente familiarizado con la tecnología. Por ello, los conocimientos de informática de los usuarios de la web son básicos. En consecuencia, la plataforma debe ser diseñada teniendo en cuenta que debe ser sencilla, de fácil mantenimiento por personal no técnico, y con equipos informáticos estándar.

1.2.- JUSTIFICACIÓN DEL PROYECTO

El desarrollo de esta plataforma web se ha llevado a cabo por una combinación de motivaciones: social, potencial explotación y académica. Este trabajo aúna la necesidad de digitalización de esta asociación sin ánimo de lucro, con la aplicación de los conocimientos técnicos adquiridos para la obtención del Grado. El objetivo principal es ofrecer una solución real a una problemática actual en un sector con escasa actualización tecnológica, logrando centralizar y recopilar toda la información de uso interno, desvinculándose de la única herramienta hasta el momento: las redes sociales.

1.2.1.- Justificación Social

La necesidad de llevar a cabo esta plataforma web se encuentra determinada por la manera tan desactualizada de llevar a cabo la gestión de la información en esta asociación. Tal y

como se ha reflejado en el apartado anterior, históricamente se ha llevado a cabo la gestión de manera manual, con documentación física y procesos descentralizados.

Esta metodología genera una dificultad operativa, que se agrava por la naturaleza del trabajo: la intermitencia y poca continuidad de participación de los voluntarios. Esto resulta una gran dificultad para la persona que toma el relevo en la gestión de cada animal, a poder llevar un correcto seguimiento, y consume innecesariamente el limitado tiempo del voluntario.

Este proyecto se justifica por la capacidad de generar un impacto social y operativo, de forma directa. Se busca la modernización y organización, de forma centralizada en una sola aplicación web, incluyendo todas las áreas críticas de la asociación: seguimiento de vacunas, historiales de adopción y archivo de contratos. Esto se traduce, de forma directa, en una gestión más eficaz, controlada y transparente, lo cual, es muy difícil conseguir trabajando únicamente con medios físicos.

Además, la implantación de esta plataforma web se verá una optimización del tiempo del voluntario de forma directa, al automatizar tareas repetitivas y administrativas, permitiendo así que el equipo pueda dedicar el tiempo utilizado en las tareas realmente necesarias.

Por último, debido a la falta de recursos económicos de la asociación, este TFG se convierte en la única vía de obtención de una herramienta de este nivel. El trabajo se justifica, por tanto, como una ayuda tecnológica directa y gratuita para una entidad que lo necesita urgentemente.

1.2.2.- Justificación de Potencial de Explotación

La creación de este proyecto se justifica, además, en la necesidad de adaptación de la asociación a la tecnología y el entorno digital. En el ámbito de las asociaciones, la máxima difusión de las labores realizadas es un factor clave en la eficacia de las campañas, siendo la plataforma web el vehículo más eficaz para alcanzar este objetivo.

En cuanto a la futura viabilidad de explotación comercial, el desarrollo de esta aplicación ofrece un alto potencial, dado que el diseño responde a una necesidad real de la asociación, pudiendo ser un producto escalable y pudiendo adaptarse a otras asociaciones de forma personalizada.

El módulo de gestión centralizada privada y la parte pública de la aplicación se han creado con la visión de configurar una plataforma de bajo mantenimiento que pueda ser comercializada o licenciada a otras asociaciones y entidades sin ánimo de lucro que operen bajo la misma situación y/o sector.

En consecuencia, el proyecto genera un beneficio doble. Por una parte, aporta una herramienta diseñada explícitamente pensando en las protectoras, abordando sus necesidades de gestión digital a un coste que sería significativamente menor al de un desarrollo personalizado y, por otro lado, presenta un potencial beneficio al transformar un trabajo académico en un servicio tecnológico con necesidad y demanda real en el sector.

1.2.3.- Justificación Académica

La realización de este proyecto se justifica en el plano académico como el reto final que permite finalizar los estudios de grado, y constituye un desafío técnico a la altura del desarrollo de un TFG.

Este desarrollo permite aplicar el conocimiento técnico adquirido durante los estudios de Grado, permitiendo la puesta en marcha de la plataforma web desde la idea base.

La plataforma exige la aplicación de metodologías de Ingeniería de software, para un desarrollo completo, desde la idea, pasando por el diseño y finalizando con el desarrollo completo de un proyecto aplicado a una necesidad real. Por otro lado, la necesidad de diseñar, generar, e implementar una arquitectura de bases de datos capaz de gestionar información sensible, como son contratos e historiales de animales, con fácil acceso y uso.

Por último, la demostración de las competencias adquiridas tanto en la parte del servidor (*Backend*) como en el navegador (*Frontend*), construyendo una interfaz de usuario accesible e intuitiva, fácilmente usable para un perfil de usuario no técnico.

1.3.- OBJETIVOS

El objetivo principal de este TFG es desarrollar e implementar una plataforma web funcional, que optimice el tiempo de los voluntarios de la Asociación de Bienestar Animal “Patas sin Fronteras”, que centralice la gestión interna de expedientes y documentación. Y que además sirva como portal de comunicación y difusión de las formas de ayuda y de los animales que están para adopción.

1.3.1.- Objetivos específicos

Para obtener el objetivo principal de este TFG, vamos a dividirlo en varios objetivos específicos, a los cuales le daremos una solución personalizada para cada uno de ellos.

- **Gestión de expedientes:** Implementar un módulo de gestión que permita la creación de fichas individuales para cada animal, incluyendo la información más relevante que debe saber una persona que está interesada en adoptar, como puede ser nombre, sexo, fecha de nacimiento o fotos del animal. Este sistema debe servir tanto para asegurar una correcta trazabilidad interna del estado de cada animal, como para servir para el público, como fuente de datos de todos los animales que hay en cada en adopción en tiempo real.
- **Trazabilidad sanitaria con sistema de alarmas:** Desarrollar un módulo de trazabilidad sanitaria dentro del área de gestión, de cada expediente individual de animal, un registro de salud que no sea visible para el público, que registre tanto datos del animal, como son el número de chip y pasaporte, como fechas de vacunas, desparasitaciones y esterilización. Este módulo, además, integrará un sistema de alarmas automatizado para notificar a cada voluntario sobre las fechas de vencimiento de estos tratamientos.
- **Gestión de documentación:** Implementar un módulo de gestión documental automatizada que, al adjuntar documentación a cada expediente individual, genere automáticamente una carpeta en **Google Drive** y un enlace directo correspondiente, permitiendo el acceso rápido desde la ficha del animal.
- **Captación de fondos y colaboración:** Diseñar e integrar un apartado público específico de "Ayuda" en la web que recoja y redirija a las todas las vías de colaboración económica de la asociación (hacerse socio, *teaming*, apadrinamiento, Bizum, etc.), facilitando así la información al visitante de la web.
- **Tienda Online:** Desarrollar un módulo de tienda online para la venta de productos de la asociación, el cual, al incluir los productos en el carrito, te deriva con un mensaje automático de **WhatsApp** al voluntario responsable de la tienda, detallando el contenido del carrito por el cual te has interesado y pudiendo coordinar así la entrega.
- **Gestión de candidatos a adopción:** Desarrollar un sistema que recoja las respuestas generadas por **Google Forms**, y las muestre en una tabla interna de manejo intuitivo. Este módulo generado en el área de gestión debe permitir a los voluntarios encargados de leerlos, añadir anotaciones, compartir impresiones y establecer una valoración final como "apto" o "no apto". En función de esa última valoración, debe generar mensajes automáticos a **WhatsApp** para iniciar o dar por finalizado el proceso de optar a la adopción de un animal.
- **Gestión de colonias felinas (Método CER):** Diseñar e implementar un módulo personalizado para la gestión del protocolo C.E.R. (Captura, esterilización y retorno) que permita la trazabilidad individualizada de los gatos callejeros de Crevillent, agrupándolos por colonias, y vinculando cada una de las colonias con su dirección y voluntario responsable. Cada ficha de gato deberá recoger información individual (nombre, código de identificación, color de capa, estado de la esterilización y fecha, datos de chip y pasaporte, etc.), Además, con la información aportada de cada animal felino, se debe automatizar la generación de un informe de este en el formato aportado por el ayuntamiento, y gestionar su almacenamiento en una carpeta de

Google Drive, con acceso restringido a los voluntarios de las colonias y los responsables municipales.

1.4.- LÍMITES DEL PROYECTO

El presente TFG aborda la fase actual de desarrollo de la plataforma, y pese a que actualmente es completamente funcional, se ve delimitada por la duración del TFG y de los recursos, al tratarse de un proyecto de naturaleza académica.

No obstante, se trata de un proyecto en continuo desarrollo y disponible para la adaptación de futuras necesidades operativas de la asociación. Por lo tanto, aunque el alcance del TFG es finito, la vida útil del proyecto no lo es el proyecto.

Pese a esto, ciertas funcionalidades quedan excluidas para garantizar un alcance realista y viable en la actualidad. Estos límites vienen dados tanto por los recursos de la asociación como por la complejidad de mantener ese uso en el tiempo:

- **Comercio electrónico:** El módulo de Tienda Online no incluirá la pasarela de pago directa (TPV), dado que la baja demanda de productos y el coste de contratación del servicio, no lo hacen viable para la asociación. La transacción se realiza mediante conversación vía WhatsApp y el pago se llevará a cabo mediante **Bizum** o efectivo, quedando el pago y el envío fuera de la plataforma web.
- **Integración con sistemas externos:** La funcionalidad de creación y modificación de documentos se realizará mediante **Google Drive** y se llevará a cabo a nivel de creación de carpetas y enlaces directos a la web. Este proyecto no presentará funcionalidades de visor ni editor de documentos dentro de la plataforma.
- **Gestión sanitaria:** El sistema de alarmas se realiza mediante una notificación visual a través del apartado de “notificaciones” dentro de la plataforma web. Esta no presenta un envío correos o mensajes a los responsables.
- **Seguridad avanzada:** El método de acceso de los voluntarios a la aplicación web será mediante el proveedor de identificación de **Google**, con las cuentas personales de cada uno de los usuarios, sin implementar seguridad extra como el doble factor de autenticación.
- **Módulo de usuario:** Esta plataforma no presentará un sistema de gestión de usuarios externos, con acceso de forma individual, como pueden ser los socios o colaboradores externos, ya que es una herramienta enfocada en los voluntarios y gestión de información interna.

Capítulo 2

Antecedentes y estado de la cuestión

2.1.- SITUACIÓN ACTUAL Y RETOS DE LA DIGITALIZACIÓN DEL SECTOR

La gestión operativa realizada dentro de las asociaciones de bienestar animal se ha caracterizado por un desfase tecnológico. La gran mayoría de las asociaciones carecen de recursos y conocimiento técnico para implementar un sistema propio informatizado. Esta situación, ha provocado la dependencia de herramientas y métodos ineficientes y no centralizados.

2.1.1.- Gestión tradicional y descentralizada

Tradicionalmente, la gestión de expedientes, y recopilación de datos sanitarios (vacunas, chip, número de pasaporte y otros datos clínicos), se han registrado en fichas individuales, mayormente de forma manual. Esta metodología provoca duplicidad en los datos, baja trazabilidad y en general, una organización deficiente al tratarse de una gestión distribuida entre múltiples voluntarios.

2.2.- HERRAMIENTAS MODERNAS GENÉRICAS

Actualmente, el camino que han tomado muchas de las asociaciones para actualizarse y comenzar a digitalizarse es el uso de herramientas generales que no están diseñadas para la logística de gestión de animales, ni flujos de trabajo:

- **Hojas de cálculo:** Se utilizan para clasificar y ordenar los animales, realizando un seguimiento de sus fechas de vacunación y registrar números de pasaporte y chip. No obstante, carecen de un sistema de alertas o campos establecidos explícitamente para la gestión que se requiere.
- **Google Drive:** Esta herramienta se emplea para guardar imágenes, documentos de identificación, informes veterinarios, o contratos de adopción, pudiendo clasificarse por carpetas. En cambio, esta herramienta no ofrece ninguna trazabilidad del expediente al que está vinculado ni un control de duplicación de documentos.

A continuación, se recogen las herramientas que hay actualmente disponibles para gestionar la problemática del ámbito de las asociaciones de bienestar animal, y porque no se han implantado cada una de ellas.

2.2.1.- Software especializado de gestión de refugios (SaaS)

Existen plataformas comerciales diseñadas específicamente para el sector, principalmente en el mercado anglosajón, como **ShelterLuv** o **PetPoint**, y algunas opciones en el mercado español como **Gpro**.

Suelen ofrecer una gestión completa que incluye registros médicos, microchips y procesos de adopción.

La principal limitación para las asociaciones es el coste de suscripción, elevado para una asociación que depende de donaciones. Además, al ser sistemas cerrados, no permiten la personalización de informes específicos para normativas locales (como el formato exacto del **Ayuntamiento de Crevillent** para el método C.E.R.) ni la integración directa con los flujos de trabajo en **WhatsApp** que requiere la asociación.

2.2.2.- Plataformas de difusión y adopción

Plataformas como **Miwuki** o **Petfinder** son muy comunes e implantadas en el sector para dar visibilidad a los animales.

Permiten centralizar la búsqueda de mascotas y facilitan el contacto entre adoptantes y protectoras.

Estas herramientas están enfocadas casi exclusivamente a la difusión. Carecen de un módulo de gestión interna que permita llevar el control sanitario detallado, la gestión de colonias felinas o la administración de una tienda propia. Son un complemento, pero no una solución a la gestión operativa diaria.

2.2.3.- Soluciones basadas en integraciones "Ad-hoc"

Es común encontrar asociaciones que intentan paliar su falta de software propio mediante la combinación de herramientas gratuitas, tal y como se ha mencionado anteriormente.

El uso conjunto de **Google Forms** y **WhatsApp** es la solución más extendida para gestionar candidatos. Sin embargo, el salto entre la recepción del formulario y la comunicación por **WhatsApp** es totalmente manual. El voluntario debe copiar el número de teléfono, guardarlo en la agenda y escribir el mensaje, lo que fragmenta la información y consume un tiempo valioso que este proyecto pretende automatizar.

2.2.4.- Resumen comparativo

Tabla 1: Comparativa de herramientas

Herramienta	Enfoque principal	Ventajas	Inconvenientes
SaaS Específicos (Gpro, ShelterLuv)	Gestión integral profesional	Alta potencia y trazabilidad sanitaria	Coste elevado y rigidez ante normativas locales
Portales de Adopción (Miwuki)	Difusión y visibilidad	Gran alcance de adoptantes potenciales	Nula gestión interna y operativa diaria
Herramientas Genéricas (Excel, Google Drive)	Almacenamiento y listas	Coste cero y facilidad de uso	Procesos manuales, duplicidad y falta de

		básico	alertas
Integraciones Ad-hoc (Google Forms, WhatsApp)	Comunicación básica	Rapidez de contacto inicial	Fragmentación; requiere intervención humana constante
Propuesta TFG (Patas Sin Fronteras)	Gestión Integral y Automatización	Bajo coste, automatización de WhatsApp, gestión C.E.R. local y centralización	Requiere desarrollo inicial y mantenimiento técnico

2.3.- VALORACIÓN Y JUSTIFICACIÓN DEL DESARROLLO

Tras el estudio de la situación actual del sector y el análisis de las soluciones presentes en el mercado, se justifica la necesidad técnica y social de este proyecto basándose en las siguientes conclusiones:

2.3.1.- Análisis de la carencia tecnológica

La principal conclusión de este estudio es la ausencia de herramientas a bajo coste que se adapten a las protectoras:

- **Herramientas genéricas (Excel/Drive):** Son accesibles, pero implican una carga de trabajo manual elevada, con pérdida de trazabilidad y alto riesgo de error humano.
- **Sistemas profesionales:** Ofrecen herramientas concretas para las necesidades de las asociaciones, pero con un coste elevado y sin posibilidad de adaptarse a las burocracias específicas.

2.3.2.- Valor de la integración y la automatización

La propuesta de generar esta herramienta trata de eliminar la fragmentación de la información. Actualmente, el voluntario debe de trabajar de forma aislada, por un lado con el formulario de **Google**, por otro lado una hoja de cálculo y por otro, redactar manualmente los mensajes en **WhatsApp**.

Además, el seguimiento de la información sanitaria, como las vacunas veterinarias, se sigue realizando mediante métodos tradicionales, como son papel y bolígrafo, en fichas físicas. Debido a toda esta problemática, supone que ninguna otra herramienta soluciona los problemas de forma completa. La plataforma desarrollada en el marco del TFG surge, por tanto, para dar solución a la unificación de todos estos procesos y garantizar que los voluntarios trabajen sobre la misma documentación e información.

Capítulo 3
Hipótesis de trabajo

El desarrollo de la aplicación web y el uso de las distintas tecnologías utilizadas han sido condicionadas por la experiencia y conocimientos en todas ellas, ya que por motivos personales y laborales se han desarrollado varios proyectos con ellas.

Al tratarse de una aplicación que no va a tener un gran tráfico ni mucha carga de datos, se ha escogido el servicio en la nube de **Firebase** de **Google**, que ofrece una capa gratuita, hasta cierto umbral de tráfico, y una vez sobrepasado ese límite, se factura según su uso. Esta plataforma simplifica las tareas complejas de *backend* y bases de datos.

La selección del *stack* tecnológico en **JavaScript** se debe a la eficiencia y escalabilidad que proporciona, además de la reutilización de recursos entre la parte del *frontend* y el *backend*. Esta uniformidad del lenguaje facilita el desarrollo, ya que los conocimientos son los mismos para ambas partes del desarrollo.

3.1.- Lenguajes de programación

La aplicación está construida íntegramente con **JavaScript**. Al hacer uso del *framework* **Angular** y **Node.js**, se hace uso de las siglas **MEAN** (**MongoDB**, **Express.js**, **Angular**, **Node.js**), para indicar el *stack* tecnológico.

Aunque en este caso, al servirse de la plataforma **Firebase,** en vez de **MongoDB** y **Express.js**, es más correcto llamarlo *stack* **MEAN** con **Firebase**.

3.1.1.- HTML/CSS/JavaScript

Estos tres lenguajes son la base fundamental de la web y son los que entienden los navegadores:

- **HTML:** Proporciona la estructura y contenido de las distintas secciones de la web.
- **CSS:** Permite aplicar presentación y estilos a la aplicación.
- **JavaScript:** Permite crear interacciones dinámicas en la aplicación.

3.1.2.- TypeScript

TypeScript es un lenguaje en el desarrollo web moderno, mantenido por **Microsoft** [1]. Se utiliza para manejar la mantenibilidad y escalabilidad del proyecto. Al ser un superconjunto de **JavaScript**, todo el código de **JavaScript** es válido en **TypeScript**, pero añade funcionalidades adicionales:

- **Estructuras de datos estáticas:** permite especificar los tipos de datos para variables, argumentos en funciones, propiedades de objetos, resultados de funciones, etc. El tipado estático permite definir tanto tipos de datos primitivos (*string, number, boolean, array*), como estructuras de datos más complejas y compuestas.
- **Estructuras de datos genéricas:** permite crear estructuras de datos reutilizables sin limitación de las estructuras de datos estáticos, ya que, mediante argumentos, permite crear funciones o estructuras de datos que se adaptan a los datos de entrada.
- **Decoradores:** es una característica que permite modificar variables, funciones, clases, etc. Se ejecutan antes de la ejecución del elemento y permite su modificación.

El código en **TypeScript** es *transpilado* a **JavaScript**, para que cualquier navegador pueda usarlo. El proceso de *transpilación* convierte el código fuente a otro de un nivel comparable, teniendo una funcionalidad y legibilidad similar. A diferencia de la compilación, que convierte el código a un nivel inferior, como por ejemplo el código binario.

La extensión de los archivos de **TypeScript** es *.ts*. La configuración de un proyecto que usa **TypeScript** se define en el archivo *tsconfig.json*. Este archivo le indica al compilador de

TypeScript como debe generar los archivos en **JavaScript**, para que sean ejecutables por el navegador.

Una característica importante de la configuración es la versión de salida de **JavaScript**, ya que define el estándar **ECMAScript** que será generado. Si se usa un estándar más antiguo, se garantizará la compatibilidad con navegadores menos modernos (**Internet Explorer**, versiones antiguas de ciertos navegadores, versiones específicas de **Node.js**, etc.). En caso contrario, usar un estándar más moderno, permitirá el acceso a las funcionalidades, mejoras de rendimiento y nuevas herramientas que se van añadiendo al lenguaje.

3.1.3.- SASS

SASS (*Syntactically Awesome Style Sheets*) es un lenguaje de preprocesamiento de **CSS** que, de forma similar a **TypeScript**, extiende las funcionalidades de **CSS**. Este lenguaje no es entendible por los navegadores, por lo que también necesita de un preprocesamiento y transformación a **CSS**, para que el navegador pueda ejecutarlo.

Existen otro tipo de alternativas a **SASS**, que actúan de manera similar, pero varían en la sintaxis o en las herramientas que proporcionan. Entre esas alternativas podemos encontrar **LESS** y **Stylus**.

Entre las características principales de **SASS**, se incluyen el uso de lógica de programación y definir estructuras de datos simples.

Existen dos sintaxis de código en **SASS** diferentes:

- **SCSS (*.scss*):** Se trata de la sintaxis más común y flexible, ya que cualquier archivo **CSS** válido es también válido en **SCSS**. Utiliza una sintaxis con *{}* para la anidación de clases y funcionalidades.
- **SASS (*.sass*):** Es el estado original, utiliza una sintaxis indentada y con saltos de línea para estructurar el código.

Hoy en día, **CSS** ha adoptado muchas funcionalidades que aporta **SASS** a su estándar, y sigue evolucionando, siendo cada vez menos relevante el uso de preprocesadores para el desarrollo web.

3.1.4.- Frontend (Angular)

Angular es un *framework* de código abierto, mantenido por **Google**. Este *framework* ha ido evolucionando durante los años, sufriendo un gran cambio en 2016. En este año, se cambió

totalmente el paradigma de desarrollo del *framework*, pasando a estar fuertemente ligado a **TypeScript** y cambiando su arquitectura a la generación de módulos y componentes.

Se pueden distinguir dos versiones de **Angular**:

- **AngularJS:** Se trata de la versión inicial. Su desarrollo es con el lenguaje **JavaScript**, y su mantenimiento fue descontinuado y cerrado en 2022.
- **Angular2+:** Engloba todas las versiones posteriores a **AngularJS**, se simplifica a esta nomenclatura para poder diferenciarlo fácilmente, ya que existen muchas versiones y está en continuo desarrollo.

Angular sigue el enfoque de creación de aplicaciones **Single Page Application (SPA)**. Este tipo de aplicaciones están compuestas por un único archivo *index.html*, que contiene todas las referencias a scripts de **JavaScript** y estilos de **CSS**.

3.1.4.1. Arquitectura basada en componentes

La aplicación sigue los estándares que recomienda **Angular** y está estructurada en una Arquitectura Basada en *Componentes* [2]. Esta arquitectura divide la interfaz de usuario en unidades pequeñas y reutilizables, manteniendo sus funcionalidades encapsuladas. Podemos encontrar distintos tipos de funcionalidades (*Componentes*, *Pipes* o *Directivas*), teniendo un objetivo específico cada una de ellas.

Los componentes gestionan una parte de la vista y pueden tener lógica asociada. Se define mediante el decorador *@Component* en el archivo **TypeScript**. Este decorador enlaza:

- **HTML:** estructura visual.
- **SCSS:** presentación y estilos de la parte visual.
- **TypeScript:** contiene la lógica y los datos del componente.

Existe otra unidad de encapsulamiento de funcionalidades llamada *Módulo*. Los módulos permiten agrupar distintos componentes, servicios y otras estructuras de la aplicación, de manera que, a la hora de cargar la aplicación se pueden encapsular en distintos paquetes, y mediante tecnologías como carga perezosa (*Lazy Loading)*, el usuario de la aplicación solo descarga los paquetes que esté usando, optimizando así los tiempos de carga de la pantalla.

En las versiones más recientes de **Angular**, el uso de los *Módulos* ha sido desaconsejado, ya que el *Framework* busca cada vez más la simplicidad, mejor rendimiento y un código más fácil de mantener y menos repetitivo. Esto se debe a que cada funcionalidad que se desarrollaba en **Angular**, debía ser declarada en un *Módulo*, lo cual añadía complejidad y líneas de código innecesarias.

La evolución ha sido los *Standalone Components*. Estos tipos de componentes aplican un comportamiento parecido al de los *Módulos*, pudiendo agrupar distintas estructuras de en el mismo componente.

Angular no tiene fuertemente definido un estándar de desarrollo, pero si tiene una guía de diseño y estructuración del desarrollo que ha ido evolucionando por la comunidad de desarrolladores. Esta falta de estándar de desarrollo otorga a los desarrolladores una gran libertad de estructurar el código como más les convenga, pero a su vez, provoca que se desarrollen aplicaciones difíciles de mantener o muy complicadas.

3.1.4.2. Inyección de dependencias y servicios

Angular implementa un patrón de inyección de dependencias (**DI**) de manera jerárquica. Este patrón permite el desacoplamiento del código y mantiene cierta lógica no relacionada con la vista en otros archivos.

Los servicios son un tipo de clases específicas de **TypeScript**, se definen con el decorador *@Injectable()*, y deben contener, preferiblemente, todo el código relacionado con la lógica de negocio, gestión de estado y almacenamiento de datos y la comunicación con el *backend*.

Los servicios son reutilizables entre los distintos componentes u otras herramientas que ofrece **Angular**. Al encapsular tareas específicas, los servicios mantienen a los componentes enfocados únicamente en la presentación de los datos y la gestión de interacciones de usuarios.

Los servicios se pueden configurar de manera que generen una instancia única para toda la aplicación y que sea reutilizada y compartida por los que la requieran, o bien, creando varias instancias de un mismo servicio, siendo los datos y variables independientes unos de otros.

3.1.5.- Backend

Para el backend, se ha optado por seguir una arquitectura **REST (*Representational State Transfer*)**. Esta arquitectura provoca que el cliente (*frontend*) se ocupe únicamente de la capa presentacional de los datos, y el servidor (*backend*) se encargue de servir los datos o ejecutar acciones sobre base de datos.

Debido a que se ha utilizado **Firebase** como plataforma de servicios en la nube, la lógica del *backend* se implementa mediante **Firebase Cloud Functions**. Esta herramienta permite crear y desplegar funciones en la plataforma de **Firebase** y que puedan ser utilizadas por el *frontend*.

3.1.5.1.- Firebase Cloud Functions

Las **Cloud Functions** son fragmentos de código que se despliegan en contenedores y se ejecutan de manera independiente. Estas funciones se mantienen desactivadas mientras no se hace uso de ellas, y una vez inicializadas, después de recibir un evento, se mantienen activas por un periodo corto de tiempo. A este tipo de ejecución se le llama ***Cold Start*** y ***Warm Container***, provocando una pequeña latencia en la petición cuando se inician, y una ejecución instantánea una vez están en ejecución.

En este proyecto, se distinguen dos tipos principales de funciones que gestionan la lógica del *backend* y son activadas por eventos concretos:

- **Funciones HTTP (API REST):** Estas funciones se exponen a través de una URL y actúan como *endpoints* de un **API REST** tradicional [3]. Permiten que el *frontend* interactúe con el *backend* y la base de datos mediante peticiones (**GET**, **POST**, etc.).
- **Funciones programadas (*Scheduled Functions*):** Estas funciones se configuran para que se ejecuten en intervalos de tiempo definidos [4]. De manera que permiten ejecutar tareas de mantenimiento, generación de informes o realizar copias de seguridad. Permite definir la ejecución programada mediante la sintaxis de **Unix Crontab** o mediante **App Engine** de **Google**.

3.2.- Gestor de paquetes y librerías de terceros

En el desarrollo de aplicaciones de **JavaScript**, se utilizan gestores de paquetes para manejar las dependencias del proyecto, permitiendo la instalación, actualización y gestión de las librerías de terceros que se utilizan tanto en el *frontend* como en el *backend*.

Existen distintos gestores de paquetes, cada uno con sus pros y sus contras. Podemos encontrar **npm**, **yarn**, **pnpm**, **bun**, etc., entre los más conocidos y utilizados.

3.2.1.- NPM (Node Package Manager)

En este proyecto se ha utilizado **NPM** como gestor de paquetes. Se trata del gestor de paquetes por defecto a la hora de instalar **Node.js**. Debido a la poca cantidad de paquetes instalados en la aplicación, no se ha barajado el uso de utilizar alguna de las alternativas.

3.3.- Gestor de base de datos

La gestión de los datos y el almacenamiento se realiza a través de los servicios que proporciona **Firebase**. Se utilizan principalmente dos herramientas de esta plataforma:

- **Firestore:** Es una base de datos *NoSQL* flexible y escalable, orientada a documentos, que se utiliza para almacenar los datos de la aplicación (animales, productos, colonias, etc.) [5]. Ofrece sincronización de los datos en tiempo real gracias al uso integrado de **WebSockets**.
- **Firebase Storage:** Se utiliza para almacenar datos binarios, como imágenes de animales y productos, documentos en PDF, archivos estáticos en formato JSON, etc. El acceso a estos archivos se hace a través de una URL.

Además, ambas estructuras de datos permiten la definición de reglas de seguridad para controlar el acceso a los datos dependiendo del usuario que los solicita o del rol asignado.

3.4.- Arquitectura y metodología de desarrollo

En cuanto a la arquitectura, se han utilizado los patrones y estándares más comunes en el desarrollo de aplicaciones modernas *frontend* y **Angular**. Manteniendo así el código con una estructura lógica y mantenible.

3.4.1.- Patrones de Arquitectura

La arquitectura del *frontend* implementada con **Angular** sigue el patrón **Modelo-Vista-Modelo de Vista (MVVM)** [6] [7].

- **Modelo (Model):** Representa los datos y la lógica de negocio. En este proyecto, el **Modelo** es gestionado principalmente por los **Servicios** de **Angular** y la comunicación con **Firestore**.
- **Vista (View):** Es la interfaz de usuario visible para el usuario final. Está representada por las plantillas **HTML** y **CSS** de los componentes de **Angular**.
- **Modelo de la Vista (ViewModel):** Actúa como un intermediario entre la **Vista** y el **Modelo**, conteniendo la lógica de presentación. Está representado por las clases **TypeScript** de los componentes de **Angular**, también llamados controladores. Estos controladores, contienen todo el código relacionado con la interacción dinámica de los usuarios y la comunicación con los servicios.

Además, **Angular** se fundamenta en la arquitectura basada en componentes. Esta estructura divide la interfaz de usuario en unidades pequeñas, reutilizables y con funcionalidades

encapsuladas, llamadas **Componentes**. Cada componente gestiona una parte de la vista y su lógica asociada, pudiendo ser reutilizados en distintos lugares de la aplicación.

A su vez, se pueden diferenciar dos tipos de componentes por sus funcionalidades y su finalidad:

- **Smart Components:** Manejan la lógica de la aplicación, como la gestión de estado, la obtención de datos y la comunicación con los servicios.
- **Dumb Components:** Se enfocan en cómo se ven las cosas. Se encargan de mostrar la interfaz de usuario a través de los datos recibidos de los **Smart Components** y de emitir eventos cuando el usuario interactúa.

3.4.2.- Principios de Diseño

Se ha adoptado la aplicación de los principios **SOLID** [8], un acrónimo de cinco principios fundamentales de la programación orientada a objetos y diseño de software:

- **Single Responsibility Principle (SRP):** El código de un componente o servicio debe tener una única razón para cambiar.
- **Open/Closed Principle (OCP):** Las entidades de software deben estar abiertas a la extensión, pero cerradas a la modificación.
- **Liskov Substitution Principle (LSP):** Los objetos de un programa deben poder ser reemplazados por instancias de sus subtipos sin alterar la corrección de ese programa.
- **Interface Segregation Principle (ISP):** Es mejor tener muchas interfaces específicas que una única interfaz general.
- **Dependency Inversion Principle (DIP):** Se debe depender de abstracciones, no de concreciones.

La aplicación de **SOLID** en la capa de **Servicios** de **Angular** garantiza la mantenibilidad y la escalabilidad del proyecto a largo plazo.

3.4.3.- Seguridad y Autenticación

Para gestionar la seguridad y el control de acceso a los recursos de la aplicación, se implementos las siguientes herramientas en el *frontend*:

- **Guards:** Son clases que utiliza **Angular** para decidir si un usuario tiene permiso para navegar a una determinada ruta o si puede salir de ella. Se utilizan para proteger las rutas privadas, asegurando que solo los usuarios autenticados o con un rol concreto pueden acceder a ellas.
- **Interceptors:** Son clases que se ejecutan antes y después de cada petición HTTP. Pueden inspeccionar y transformar las peticiones HTTP antes de que sean enviadas al *backend* y las respuestas antes de que sean procesadas por la aplicación.

Principalmente se usan para añadir credenciales de autenticación a las peticiones salientes y para manejar errores cuando se ha recibido respuesta del *backend*.

- **Módulo de inicio de sesión:** Implementa la lógica de autenticación del usuario. Este proceso gestiona la comunicación segura con el *backend* (**Firebase Authentication** [9]) para verificar las credenciales del usuario y almacenar de forma segura el estado de la sesión.

3.4.4.- Integración *frontend* y *backend*

Toda esta lógica de integración *frontend* con **Angular** y *backend* con **Firebase** se ha realizado usando la librería **@angular/fire** [10]. Es una librería oficial mantenida por **Google** y el equipo de **Angular**.

Esta librería proporciona una capa de abstracción entre **Firebase** y todos los métodos que proporciona de forma nativa, y el patrón de desarrollo de **Angular**. Esta abstracción simplifica enormemente el uso de las distintas herramientas de **Firebase**, como el uso de autenticación, la conexión con la BBDD, la obtención de los archivos de **Firebase Storage** sin necesidad de una URL directa o las llamadas a **Cloud Functions** sin necesidad de conocer la URL del *endpoint*.

3.5.- Herramientas de desarrollo

En cuanto a las herramientas de desarrollo de software y utilidades para el desarrollo de este proyecto se incluyen **WebStorm**, sistemas de control de versiones y herramientas de calidad de código.

3.5.1.- IDE

El entorno de desarrollo integrado (**IDE**) utilizado para la construcción, depuración y gestión del proyecto es **WebStorm** [11].

WebStorm es un **IDE** comercial, desarrollado por la empresa **JetBrains**, diseñado específicamente para el desarrollo en **JavaScript**. Esta empresa dispone de distintos **IDEs** especializados en distintos lenguajes de programación, como por ejemplo **Intellij IDEA** para **Java**, **PyCharm** para **Python**, **PhpStorm** para **PHP**, etc. Al tratarse de una herramienta comercial, su uso requiere una licencia pagada.

Este **IDE** proporciona opciones de autocompletado inteligente, herramientas de refactorización y detección de errores en tiempo real para código **TypeScript**, **JavaScript** y plantillas de **Angular**.

Está perfectamente ligado e integrado con **Git**, ya que permite, mediante una interfaz gráfica personalizada, gestionar commits, ramas, merges y la interacción con el repositorio donde se encuentra almacenado el proyecto.

Permite configurar herramientas de calidad de código como **ESLint** y **Prettier**, indicando los errores reportados de manera directa en tiempo de desarrollo y pudiendo hacer las correcciones que se recomiendan tras guardar los archivos.

Además, **WebStorm** ofrece herramientas de depuración de código **JavaScript** y **TypeScript** mediante el establecimiento de puntos de control para interrumpir la ejecución del programa en ciertas líneas de código y así examinar más detalladamente su estado.

3.5.2.- Control de versiones

Para la gestión y seguimiento del desarrollo de la aplicación se ha utilizado el sistema de control de versiones de **Git** junto con la plataforma **GitHub**. El uso de estas herramientas es fundamental para proyectos grupales que trabajan de manera colaborativa o bien para poder mantener un registro de las funcionalidades que se han ido añadiendo durante el desarrollo del proyecto.

Existen otras opciones populares de control de versiones como son el caso de **Mercurial** y **Subversion (SVN)**. Pero por motivos de simpleza y conocimientos profesionales, se ha hecho uso de **Git**.

3.5.2.1.- Git

Git es una herramienta de control de versiones que permite registrar y tener un seguimiento de todos los cambios realizados en el código durante su desarrollo.

Permite mantener un seguimiento detallado de los cambios en el tiempo, mantener los desarrollos incompletos en distintas ramas de manera aislada y volver a versiones anteriores de código en caso de ser necesario.

Además, se ha seguido el patrón **Gitflow** para gestionar la nomenclatura y creación de las distintas ramas que contienen los desarrollos de nuevas funcionalidades o correcciones [12].

3.5.2.2.- GitHub

GitHub es una plataforma destinada a almacenar repositorios de código en la nube. Actúa como el servidor donde se sincronizan los cambios aplicados por los distintos desarrolladores.

Además, en este proyecto se utiliza la herramienta **GitHub Actions**, ofrecida por **GitHub**. Esta es una herramienta de **Integración Continua/Despliegue Automático (CI/CD)**, que permite automatizar tareas de despliegue y ejecución de *scripts*.

3.5.3.- Estandarización y calidad de código

Para asegurar la mantenibilidad, escalabilidad y consistencia del código en el proyecto, se han integrado dos herramientas fundamentales que trabajan en conjunto para estandarizar el estilo y mejorar la calidad de código: **ESLint** y **Prettier**.

3.5.3.1.- ESLint

ESLint es una herramienta de análisis estático de código, se utiliza como inspector de la calidad del código detectando potenciales errores y sugiriendo buenas prácticas basadas en unas reglas configuradas.

Las reglas utilizadas para su configuración pueden ser totalmente personalizadas o bien utilizar conjuntos de reglas proporcionadas por la comunidad mediante plugins. Esta configuración se define en un archivo *.eslintrc.json* o *eslint.config.js* (versión más moderna).

3.5.3.2.- Prettier

Prettier es un formateador de código automático. Su propósito es, mediante unas reglas definidas, aplicar un formato uniforme a todo el código.

A diferencia de **ESLint**, **Prettier** se enfoca únicamente en el estilo visual, como pueden ser los espaciados, las comillas utilizadas, saltos de línea y puntos y coma.

Al aplicar un estilo uniforme se garantiza que todos los desarrolladores, independientemente de sus preferencias personales en el desarrollo, sigan el mismo patrón de desarrollo. De esta

forma, cada usuario puede enfocarse en el desarrollo de la lógica o funcionalidades, y dejar de lado el formato y estilo de código.

La configuración de **Prettier** se define en un archivo *.prettierrc*.

3.5.4.- Auditoría de código

Se ha utilizado **Lighthouse** como herramienta para auditar la calidad, el rendimiento y la accesibilidad de la aplicación [13]. Está desarrollada por **Google** y es accesible desde cualquier navegador **Chrome**.

Esta herramienta analiza la aplicación y genera un informe con puntuaciones y sugerencias de mejora. Se basa en distintas métricas para evaluar la aplicación:

- **Rendimiento:** Analiza la velocidad de carga de la aplicación, evaluando métricas como *First Content Paint* (FCP) y *Largest Content Paint* (LCP). Estas métricas indican el tiempo transcurrido desde que la aplicación comienza a cargarse hasta que cualquier parte del contenido de la página es visible por pantalla, y el tiempo que tarda en cargar el elemento más grande visible en pantalla, como por ejemplo imágenes, vídeos, etc.
- **Accesibilidad:** Revisa que la aplicación es usable para personas con problemas visuales, auditivos, etc., comprobando el contraste de textos, el uso atributos **ARIA** y una correcta estructura del **DOM**.
- **Mejores prácticas:** Busca problemas ocasionados por el uso de librerías obsoletas o vulnerabilidades de seguridad básicas.
- **SEO (Optimización para motores de búsqueda):** Comprueba si la aplicación cumple con los requisitos básicos para ser indexada por los distintos motores de búsqueda, requiriendo el uso de metadatos y títulos.
- **Aplicaciones Web Progresivas (PWA):** Si la aplicación está desarrollada para ser una **PWA**, verifica que se cumplen los criterios para ser instalada y su funcionamiento *offline*.

3.6.- Plataforma de despliegue y servicios en la nube

La plataforma **Firebase** también permite el despliegue de aplicaciones, además de un dominio asociado, así como una integración con la herramienta de **Google Analytics** para monitorizar el uso de la aplicación por parte de los usuarios y los posibles errores que pueden ocurrir por problemas de código.

3.6.1.- Firebase Hosting

Los archivos estáticos de la aplicación compilada se alojan en el servicio de **Firebase Hosting**, que se sirven a los usuarios a través de un dominio generado por **Firebase** de forma gratuita.

Además, **Firebase Hosting** tiene un historial de versiones desplegadas de la aplicación web, pudiendo restaurar un despliegue anterior de forma directa y sencilla.

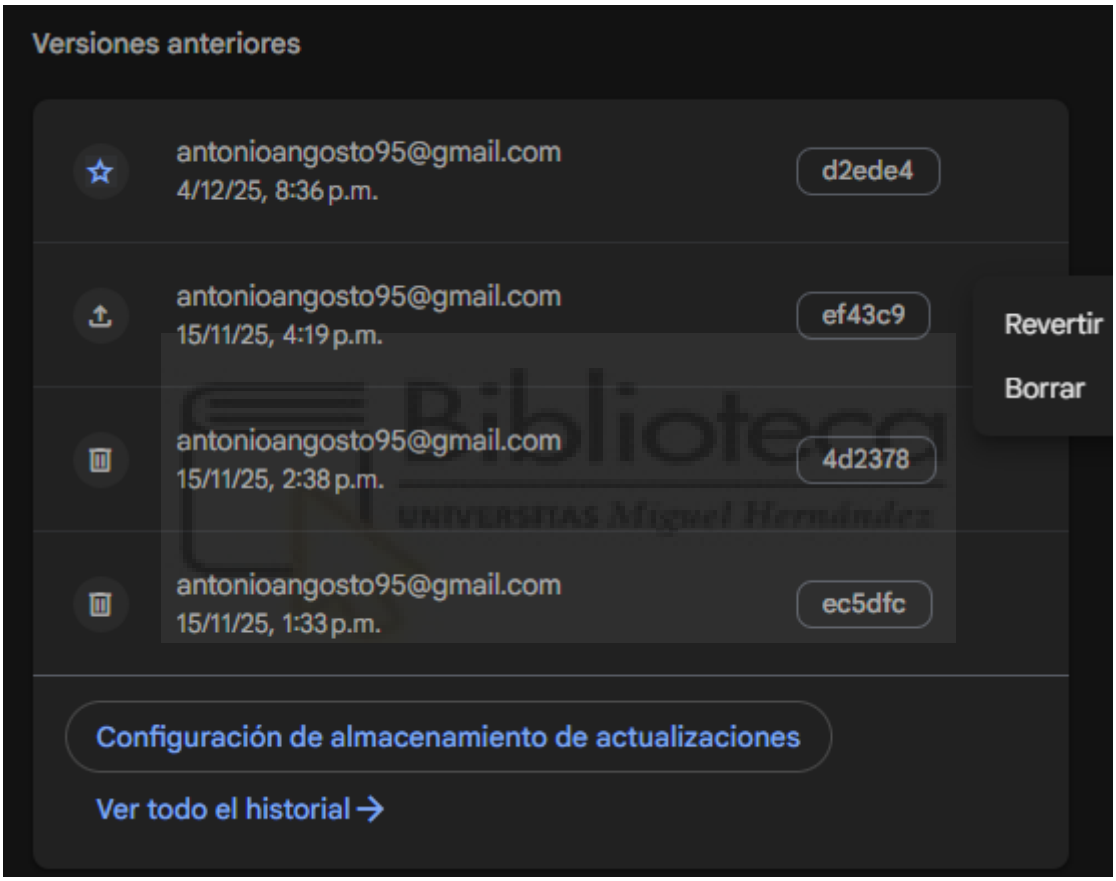

Ilustración 1: Control de versiones Firebase Hosting

3.6.2.- Firebase Analytics

La plataforma también incluye un panel de recopilación de datos sobre el uso y el rendimiento de la aplicación integrado con **Google Analytics**.

Esta herramienta permite visualizar las interacciones y comportamientos de los usuarios en la aplicación, cantidad de usuarios que están ejecutando la aplicación en el momento o el país de los usuarios que acceden.

3.7.- Estilos y experiencia de usuario (UX/UI)

Para la interfaz de usuario y la experiencia de usuario se han considerado aspectos como el **modo oscuro**, el **contraste de textos** y la **accesibilidad (a11y)**.

Para lograr estos aspectos con mayor facilidad, se ha hecho uso de herramientas que facilitan la estructuración de la vista en **CSS** y el uso de componentes visuales preconstruidos.

3.7.1.- Tailwind CSS

Tailwind CSS es un *framework* de **CSS** que utiliza un enfoque de CSS de utilidades y que permite construir diseños complejos directamente en el HTML, fomentando la rapidez en el desarrollo y la consistencia del diseño [14].

Este *framework*, ofrece un sistema de diseño predefinido que garantiza la consistencia en los estilos de la aplicación, unificando espaciados, colores y tamaños de textos.

3.7.2.- NG-ZORRO

Para componentes de interfaz de usuario más complejos y estandarizados, se integra **NG-ZORRO** [15]. Este *framework* es un conjunto de componentes de interfaz de usuario para **Angular**, basado en el sistema de diseño de **Ant Design**. Proporciona elementos preconstruidos que aceleran el desarrollo y garantizan una experiencia de usuario familiar y profesional.

3.7.3.- Internacionalización (i18n)

La aplicación también cuenta con un sistema de internacionalización, para poder gestionar el cambio de idioma dependiendo de las preferencias del usuario.

Se utiliza la librería **Transloco** para gestionar y servir los textos de la interfaz en múltiples idiomas [16]. **Transloco** se integra nativamente con **Angular** y permite cargar los archivos de traducción bajo demanda, optimizando el rendimiento y simplificando el mantenimiento de las traducciones.

En este momento, se da soporte a los idiomas español e inglés, pudiendo ser configurado cualquier otro idioma de forma sencilla, ya que solo hay que añadir un nuevo archivo con sus traducciones estáticas.

Capítulo 4
Metodología y resultados

El desarrollo de software no es un proceso lineal, donde todo se encuentra bien definido y no se producen errores durante el desarrollo. Para ello se debe tener un seguimiento continuo con los usuarios finales de la aplicación para poder satisfacer las necesidades y que hagan un uso real de la aplicación para obtener posibles casos no resueltos y errores imprevistos.

4.1.- Planificación del proyecto

Debido a que el proyecto se ha realizado únicamente con un desarrollador técnico, con recursos limitados y la validación de las funcionalidades con un cliente no técnico (los voluntarios de la asociación), se ha optado por un ciclo de vida **iterativo e incremental**.

Con este modelo de ciclo de vida, se ha podido ir creando partes de la aplicación de forma aislada y con funcionalidades pequeñas, y evolucionando partes ya creadas, adaptándose a las necesidades de los voluntarios y usuarios de la aplicación.

Gracias a las entregas parciales de funcionalidades, se permite mostrar a los usuarios los prototipos funcionales, facilitando la retroalimentación y la mejora continua.

Una vez creada la base y la estructura de las clases y entidades, el primer paso es crear la parte pública y accesible por cualquier usuario de la aplicación, para más adelante continuar con la parte privada de gestión y controlada por roles de usuario.

4.1.1.- Ciclo de vida iterativo e incremental

Al ser un ciclo de vida iterativo, el primer paso es la toma de requisitos y requerimientos de la funcionalidad a desarrollar. Tras haber realizado todo el desarrollo de la funcionalidad y haberlo implementado y desplegado para su uso, el cliente puede pedir cambios y mejoras de esta funcionalidad, reproduciendo de nuevo todo el ciclo.

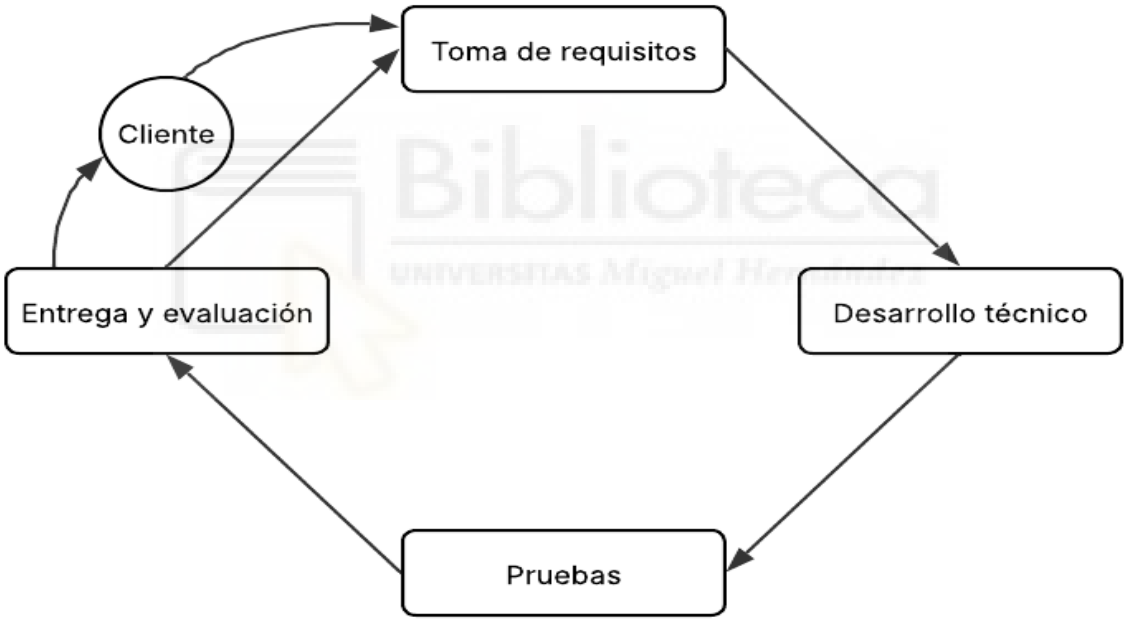

Ilustración 2: Ciclo de vida iterativo

4.1.2.- Planificación temporal (Diagrama de Gantt)

Como se observa a continuación, el proyecto se ha dividido en distintas etapas siguiendo una metodología incremental, teniendo que iterar tareas ya realizadas en periodos distintos debido a la retroalimentación con los voluntarios y la inclusión de nuevas funcionalidades.

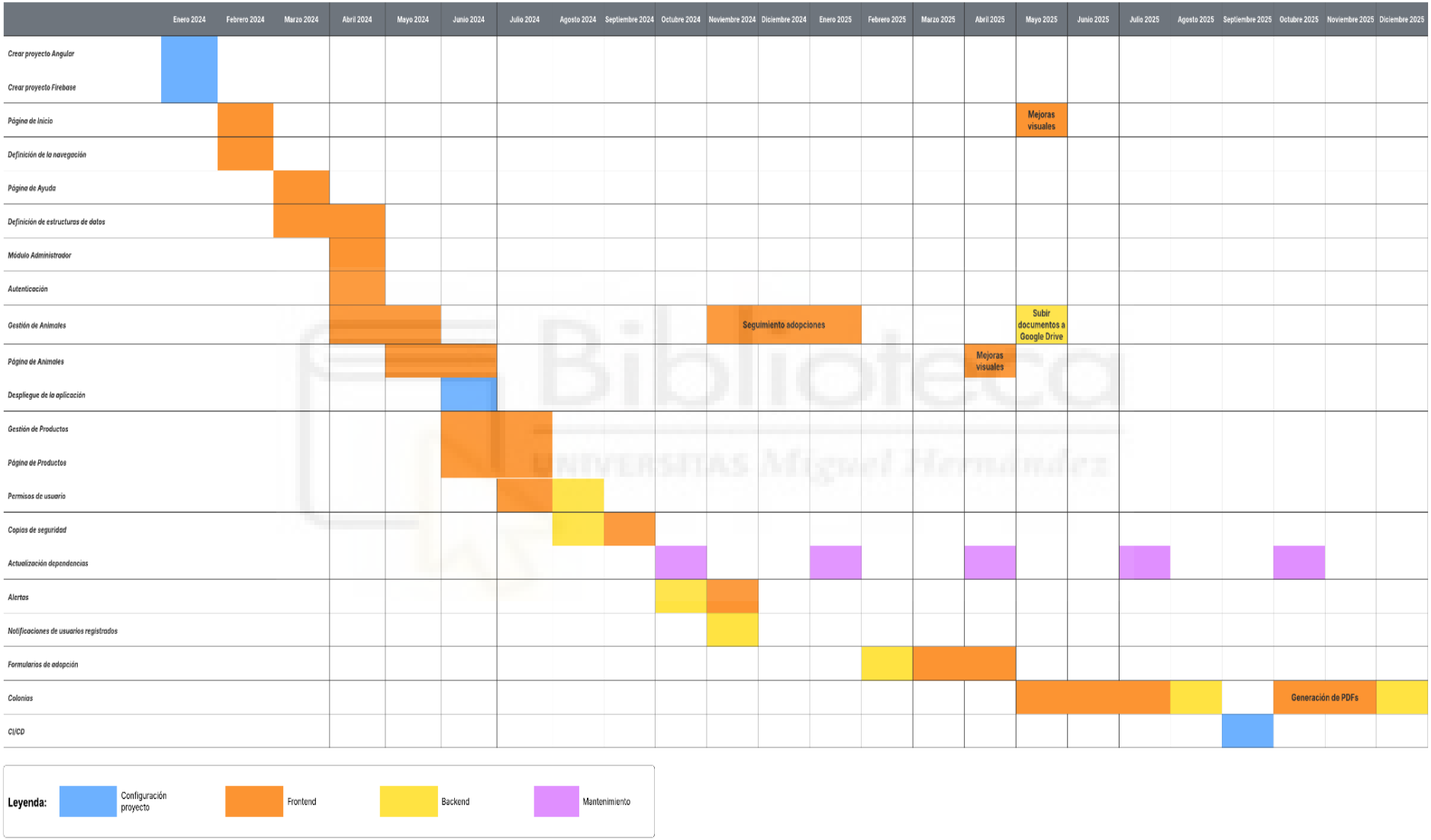

Ilustración 3: Diagrama de Gantt

4.2.- Captura de requisitos

Para garantizar que la aplicación solvente las necesidades de la asociación, se ha llevado a cabo un proceso de diálogo continuo con los usuarios y voluntarios que van a utilizar la plataforma.

4.2.1.- Identificación de Actores

Se han identificado dos actores principales que interactúan con la aplicación, con niveles de acceso y privilegios claramente diferenciados.

- **Visitante (Usuario público):** Representa a cualquier persona que accede a la web sin necesidad de estar autenticado. Su interacción se limita a la consulta de información (animales en adopción, formas de colaborar con la asociación y ver productos en venta).
- **Voluntario (Usuario gestor):** Representa a los usuarios que tienen permisos y roles para acceder a la parte privada de gestión. Este rol tiene permisos para crear, editar y eliminar registros de los distintos módulos de gestión. A su vez, existen distintos roles para tener acceso y modificar solo ciertas partes de los datos:
    - **Admin:** Tiene acceso total a todos los módulos y modificación de los registros.
    - **Editor:** Tiene acceso a los módulos de gatos, perros, exóticos y productos. Pueden modificar todos los registros.
    - **Perros:** Es un rol que hereda de **Editor**. Tiene acceso adicional a los formularios de adopción de perros y exóticos.
    - **Gatos:** Es un rol que hereda de **Editor**. Tiene acceso adicional a los formularios de adopción de exóticos.
    - **Gestor Colonias:** Tiene total acceso al listado de colonias y la modificación de los registros.
    - **Alimentador Colonias:** Tiene acceso al módulo de colonias y la modificación de los registros, pero solo de las colonias a las que se le ha dado acceso. Este acceso solo lo puede proporcionar un usuario con rol **Admin**.

Todos los roles son gestionados a través del módulo de Roles de Usuario. Este módulo únicamente es accesible para los usuarios **Admin**, y pueden añadir y modificar los usuarios con permisos de acceso.

4.2.2.- Requisitos funcionales

Al haber dos tipos de usuarios diferenciados por el tipo de acceso y el uso que van a hacer de la aplicación, se han agrupado los requisitos funcionales en dos bloques.

4.2.2.1.- Módulo público

El objetivo de este módulo es la difusión y la captación de adoptantes.

- **Tipos de animales en adopción:** Se debe mostrar los distintos tipos de animales que se pueden adoptar, navegando al listado del tipo seleccionado.
- **Visualización de animales:** Se debe mostrar el listado de los animales, con la imagen y los detalles e información de cada animal. Además, se debe habilitar un botón para navegar al formulario de adopción de cada animal.
- **Tienda solidaria:** Se debe mostrar un listado de productos, con la información de cada uno y una galería de imágenes. Al no disponer de pasarela de pago, el proceso de compra se realizará mediante un botón que genera un mensaje automático de **WhatsApp** al encargado de la asociación, rellenado con los datos de los productos seleccionados.
- **Donaciones y colaboración:** Página informativa con las formas de colaborar y ponerse en contacto con la Asociación.

4.2.2.2.- Módulo privado

El objetivo de este módulo es la gestión y la modificación de los registros que van a ser visibles para la parte pública y para gestionar documentación privada de manera sencilla. Para acceder a este módulo es requerida la autenticación con un usuario permitido y con roles.

- **Autenticación:** Se permitirá el acceso mediante **OAuth** de **Google**, permitiendo únicamente el acceso a las cuentas definidas en una lista blanca.
- **Gestión de animales:** Los voluntarios y gestores podrán dar de alta, modificar y eliminar animales, así como el estado de adopción del animal (“Adoptado”, “Reservado”, “En adopción”, etc.). Estos datos se verán reflejados en el listado de animales de acceso público.
- **Gestión del estado y seguimiento del animal:** Se permitirá el registro de datos de seguimiento de los animales adoptados, como vacunas, número de chip y documentos de adopción. Estos datos son internos y no se verán reflejados en la parte pública.

- **Alertas del animal:** A partir de la información registrada en el seguimiento de los animales, se generarán alertas para cada responsable dentro de la aplicación, indicando las fechas estimadas de la próxima vacuna y la fecha de castración recomendada.
- **Gestión de colonias felinas:** Se permitirá la creación de colonias y los gatos integrantes dentro de cada una de ellas para su gestión. Se podrán generar documentos en PDF a modo de resumen para presentar en subvenciones a nivel de ayuntamiento.
- **Gestión de documentación de las colonias:** Se permitirá generar las fichas de los animales de las colonias en PDF con toda la información registrada por parte de los voluntarios. Estas fichas se subirán a la cuenta de **Google Drive** de la Asociación.
- **Gestión de formularios de adopción:** Tras el completado de un formulario de adopción en **Google Forms** su registro en Google Sheets, la plataforma permitirá la lectura de todos los formularios registrados en **Google Sheets**, ofreciendo una visualización más accesible y sencilla. También se permitirá la confirmación o denegación de un formulario mediante mensajes automatizados.
- **Gestión de permisos de usuario:** Se permitirá la modificación de permisos y roles a los usuarios mediante correo electrónico.
- **Gestión de copias de seguridad:** Se generarán copias de seguridad de la base de datos mensualmente, pudiendo descargar y restaurar tras algún fallo en los datos.

4.2.3.- Casos de Uso

Para ilustrar las interacciones del apartado anterior, se presentan los diagramas de casos de uso en el que los voluntarios y gestores tienen acceso total a todo el sistema o parte de él mediante roles, y los visitantes, que se limita su acceso solo a visualización.

4.2.3.1.- Caso de uso Visitantes

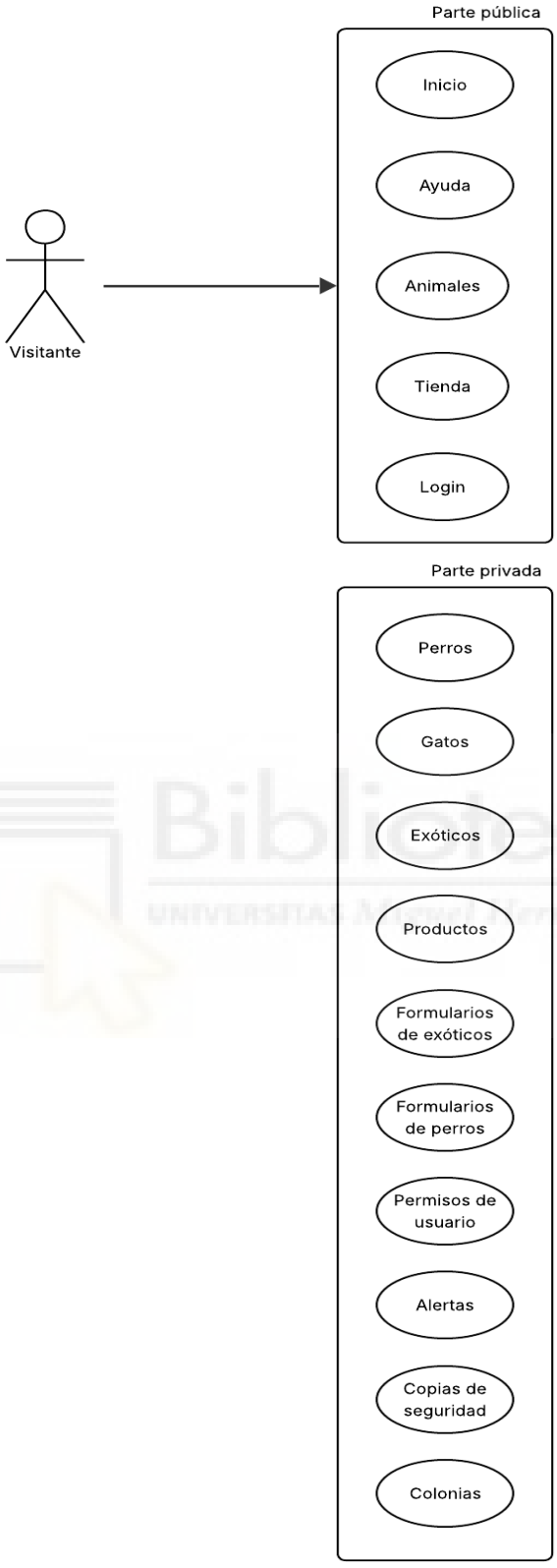

Ilustración 4: Caso de uso Visitantes

4.2.3.2.- Caso de uso Admin

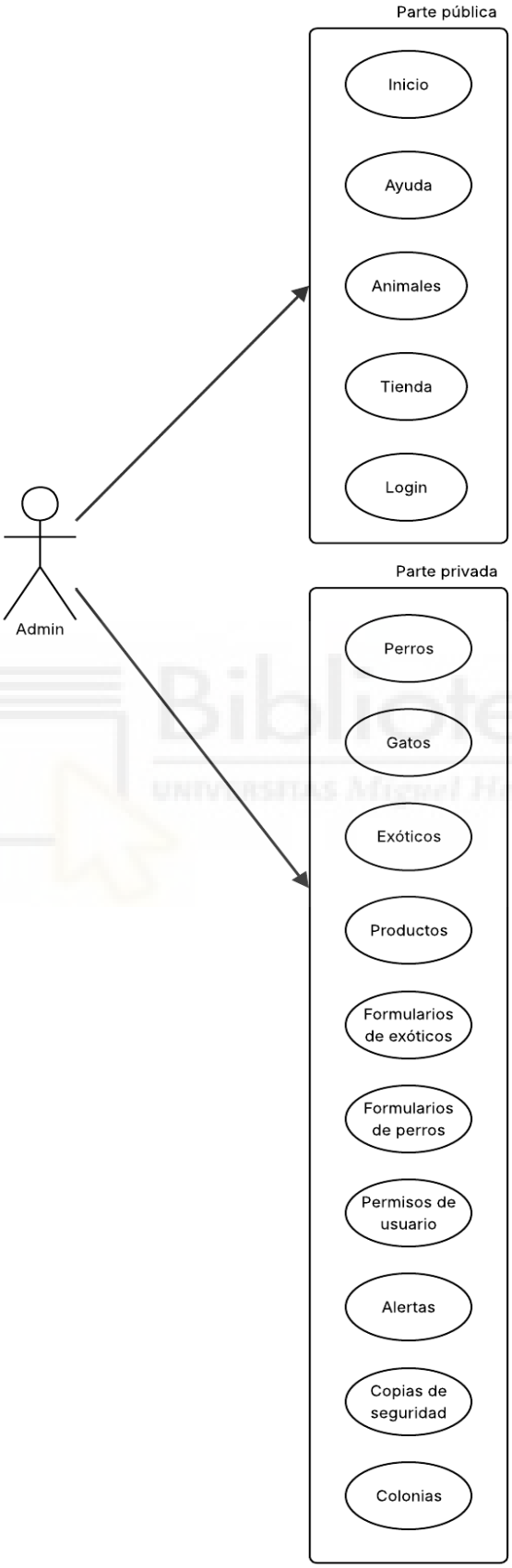

Ilustración 5: Caso de uso rol Admin

4.2.3.3.- Caso de uso Editor

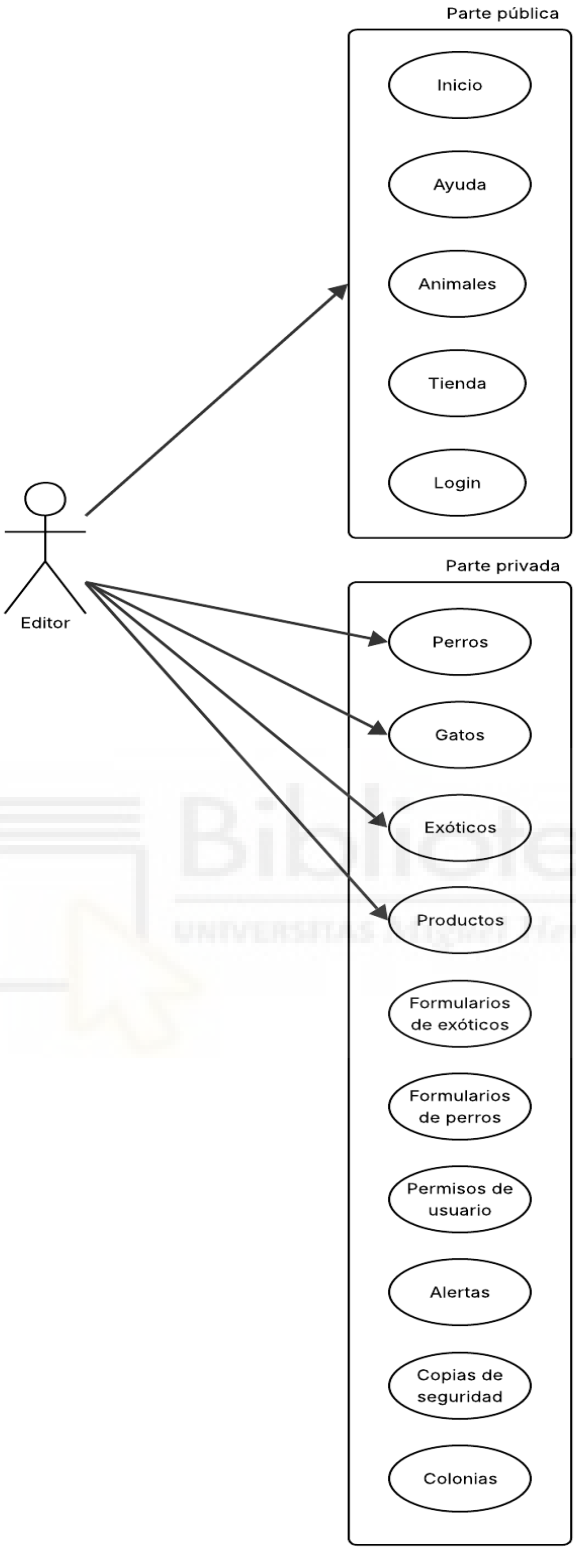

Ilustración 6: Caso de uso rol Editor

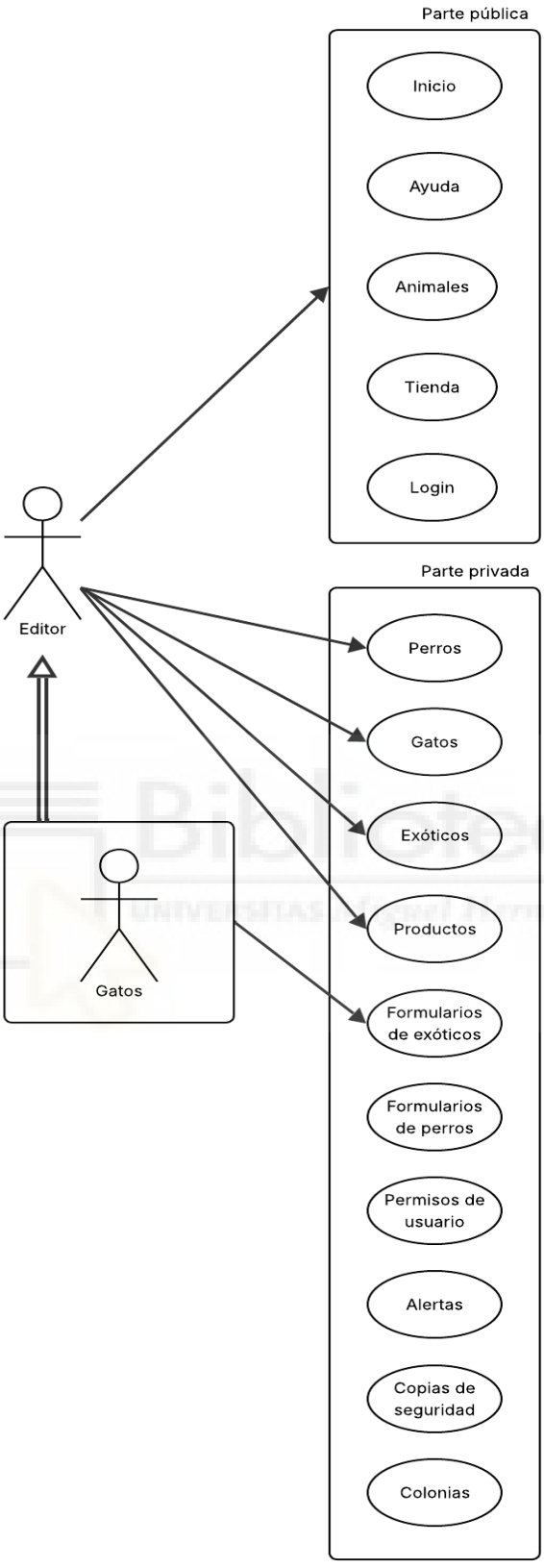

Ilustración 7: Caso de uso rol Gatos

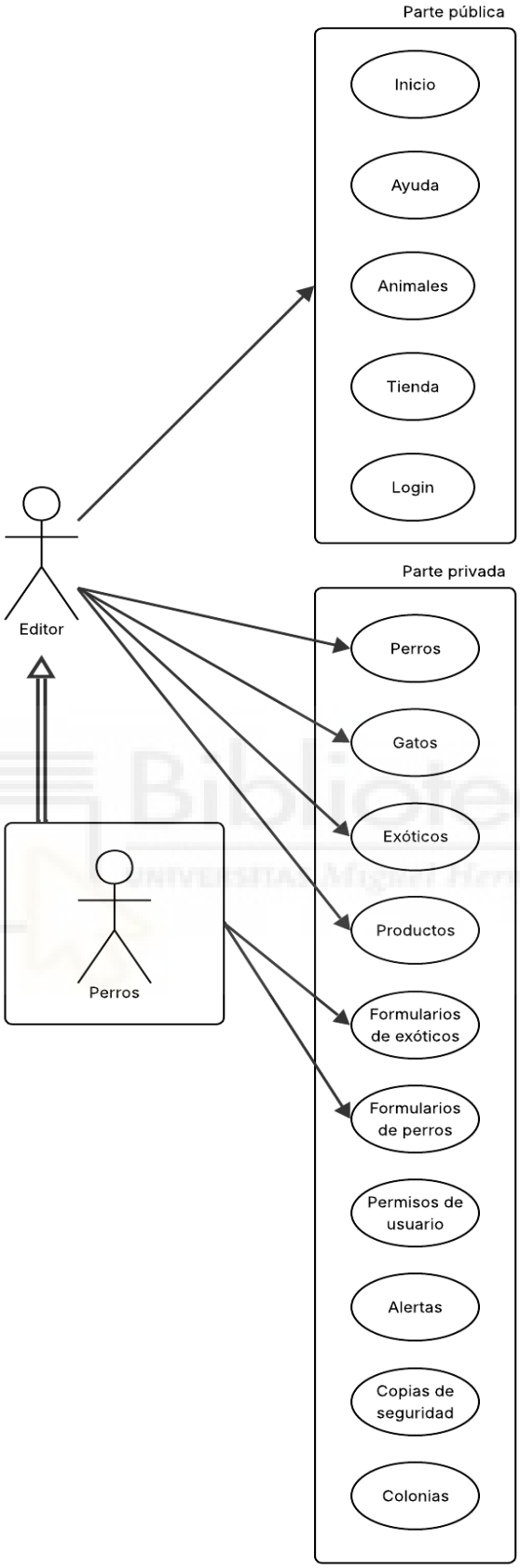

Ilustración 8: Caso de uso rol Perros

4.2.3.4.- Caso de uso Colonias

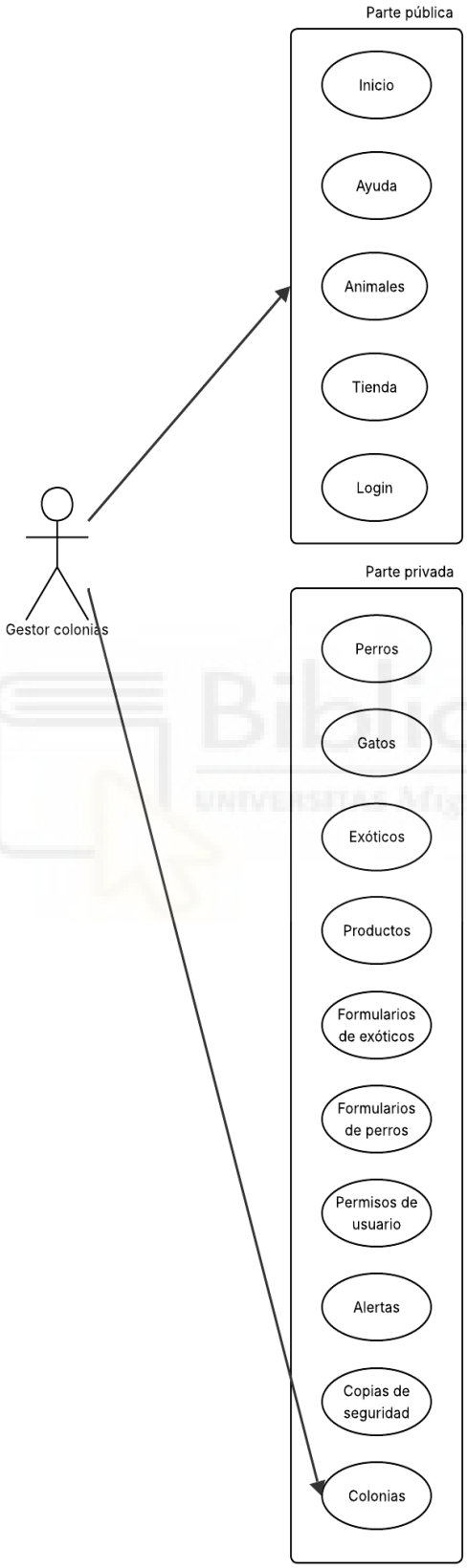

Ilustración 9: Caso de uso rol Gestor colonias

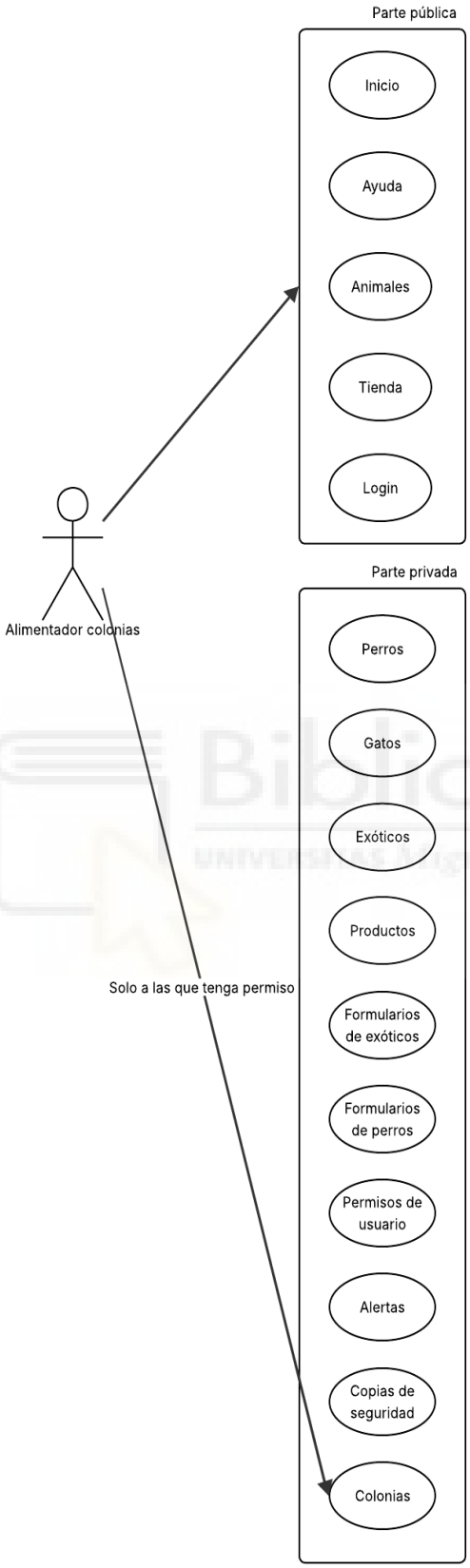

Ilustración 10: Caso de uso rol Alimentador colonias

4.2.4.- Requisitos no funcionales

En cuanto a los requisitos no funcionales, encontramos los que definen la calidad del sistema, siendo necesarios para los perfiles no técnicos para el uso amigable de la aplicación.

- **Usabilidad y accesibilidad:** La interfaz debe ser intuitiva y sencilla, utilizando un sistema de diseño unificado y con coherencia entre las distintas páginas. Además, debe tener un diseño *responsive*, asegurando que tenga una correcta visualización en las distintas resoluciones de pantalla.
- **Rendimiento:** Se utilizarán tecnologías de *Lazy Loading* para optimizar la carga de las distintas pantallas y descargar únicamente los bloques de código y páginas que se visualicen. También se optimizará el uso de eventos y animaciones para no sobrecargar las páginas reduciendo su rendimiento.
- **Seguridad y autenticación:** Mediante el uso de reglas en **Firebase Firestore** [17], se configurará que solo los usuarios con el rol correcto puedan realizar operaciones de escritura en la base de datos, mientras que los demás usuarios sólo realizarán operaciones de lectura.
- **Escalabilidad y costes:** La arquitectura de **Firebase** permite escalar y aumentar los recursos destinados a la aplicación de manera automática, permitiendo dar mayor soporte en casos de mucho uso. Mediante la optimización de las consultas y operaciones que se realizan, se puede incurrir en unos costes de los servicios muy bajos.
- **Mantenibilidad:** Se deben seguir los estándares de desarrollo de **Angular** y utilizar herramientas de calidad de código (**ESLint** y **Prettier**) para unificar el desarrollo y facilitar el desarrollo a futuro por terceras personas.

4.3.- Diseño

A continuación, se detalla el modelado de datos y la arquitectura de uso de la aplicación.

4.3.1.- Modelo de datos

Aunque la aplicación utiliza una base de datos **NoSQL**, se ha definido una estructura de modelos y relaciones entre distintos tipos de datos para mejorar la mantenibilidad de la aplicación.

4.3.1.1.- Animales

Los tres tipos de animales que están disponibles para adoptar (perros, gatos y exóticos) comparten la misma estructura de datos, menos la estructura de perros, que añade la propiedad `size`.

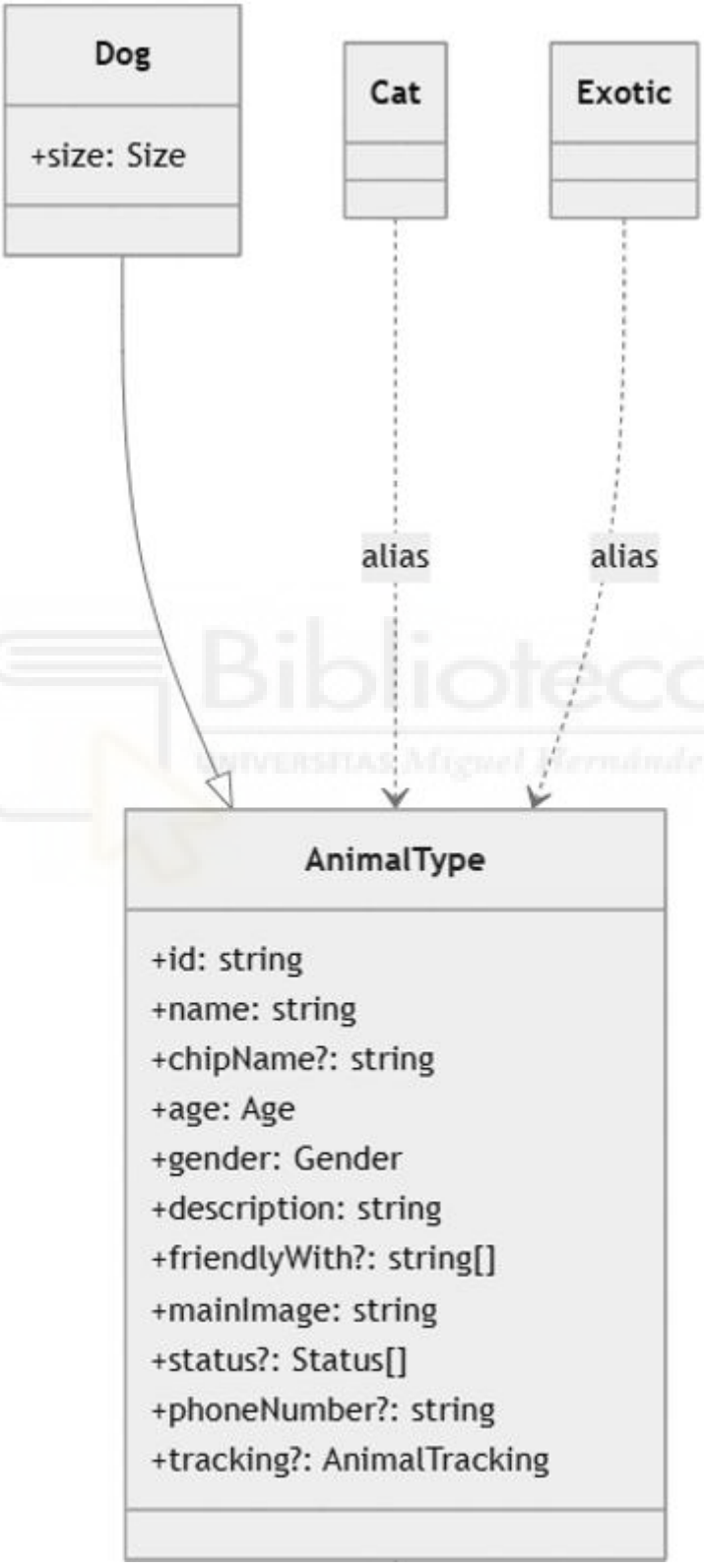

Ilustración 11: Modelo de datos de animales

Ilustración 12: Constantes de los modelos de datos de animales

La propiedad `tracking` almacena todos los datos requeridos para el seguimiento de las adopciones de los animales.

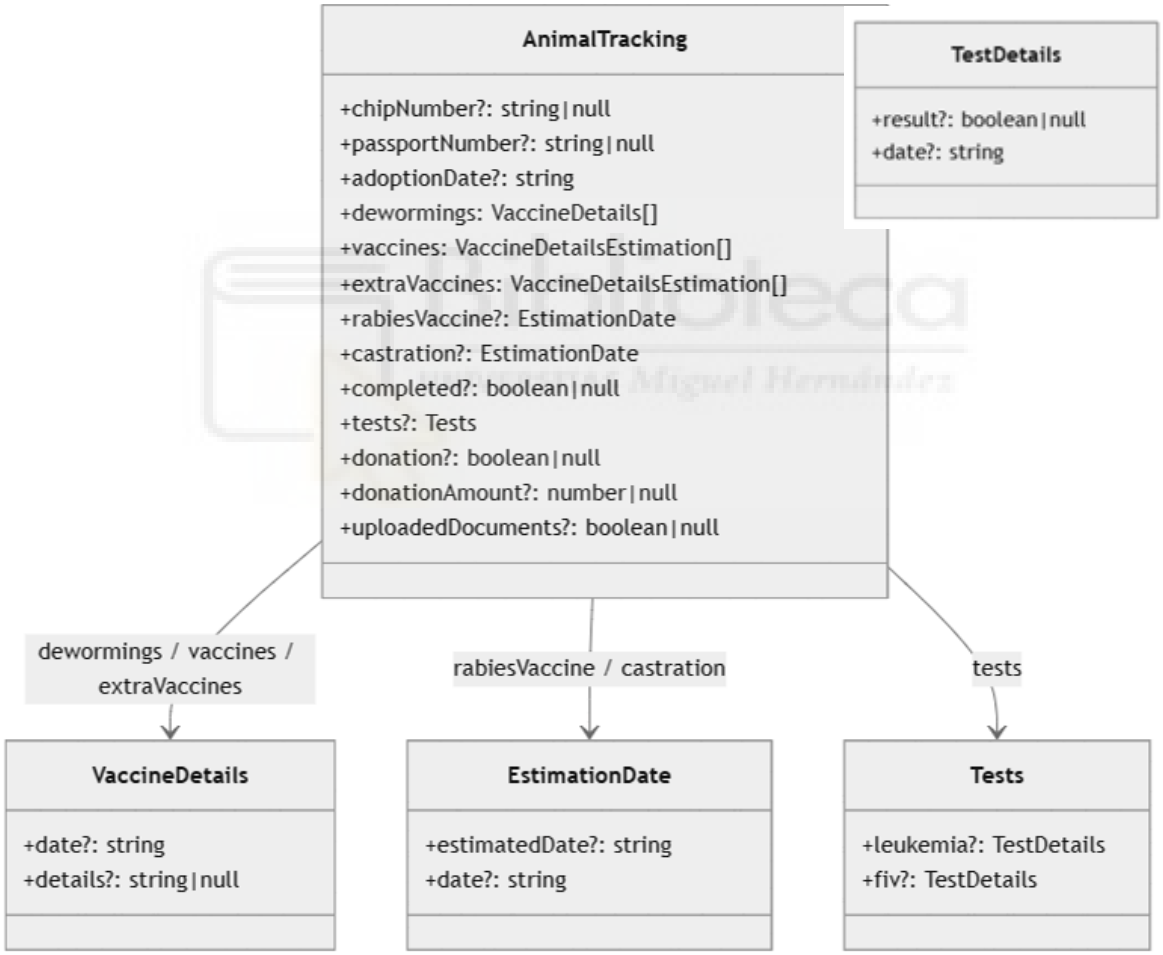

Ilustración 13: Modelo de datos de seguimiento de animales

4.3.1.2.- Productos

Los datos de los productos que se almacenan en la base de datos siguen el modelo de `Product`. Para los usuarios de la aplicación que visitan la tienda, todos los productos almacenan también la propiedad `quantity` de la interfaz `CartItem`.

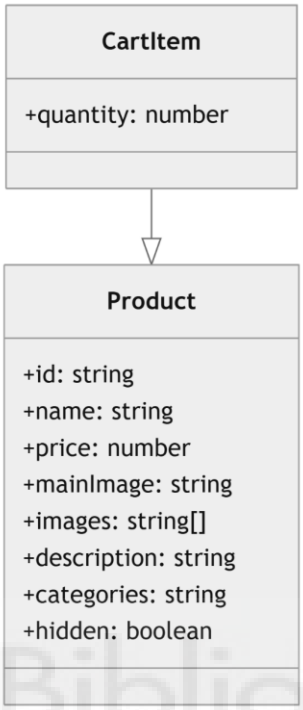

Ilustración 14: Modelo de datos de productos

4.3.1.3.- Alertas

Las alertas contienen información resumida del seguimiento y próximas vacunas de los animales que se han dado en adopción. Como cada animal tiene una persona asociada a su seguimiento, se almacena un `userId`. Además, se permite la navegación directa a la tabla de gestión con los detalles del animal desde la alerta, ya que se almacena el tipo de animal, el identificador y el nombre.

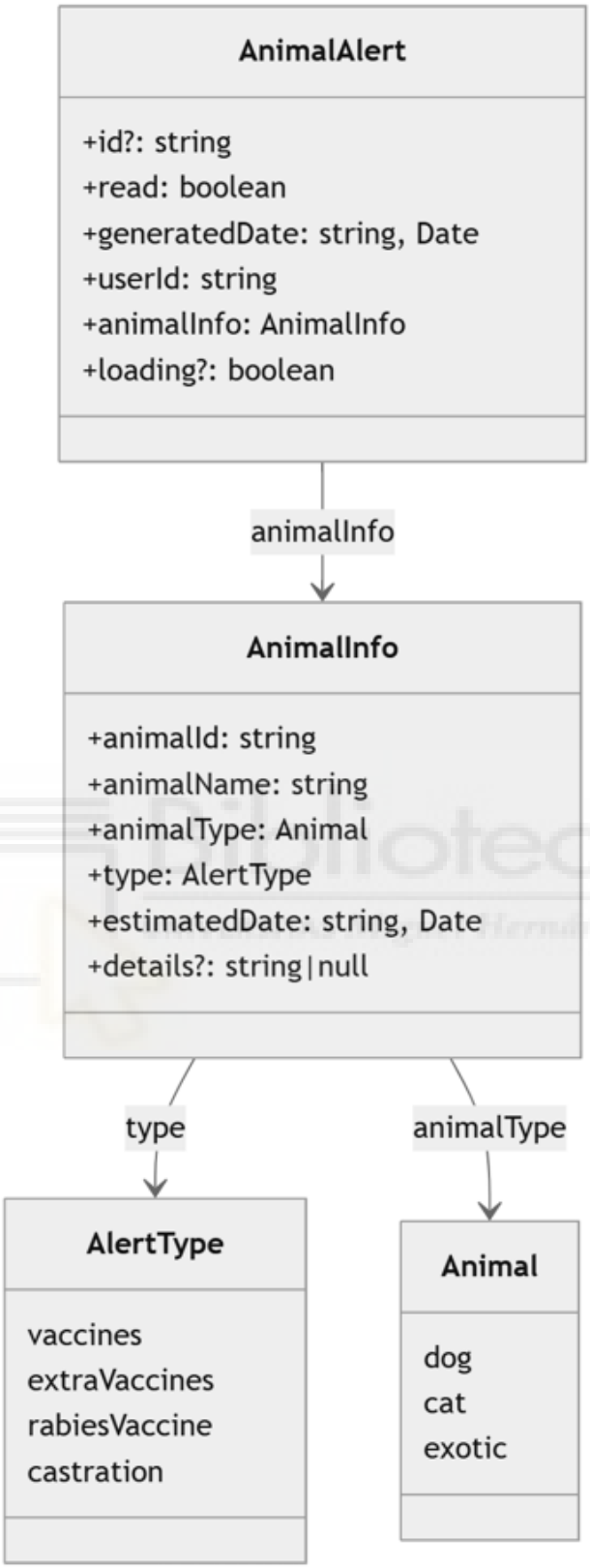

Ilustración 15: Modelo de datos de alertas

4.3.1.4.- Colonias

Las colonias almacenan todos los datos requeridos para la generación de fichas con los datos requeridos por los ayuntamientos que llevan un seguimiento de las colonias felinas de su territorio.

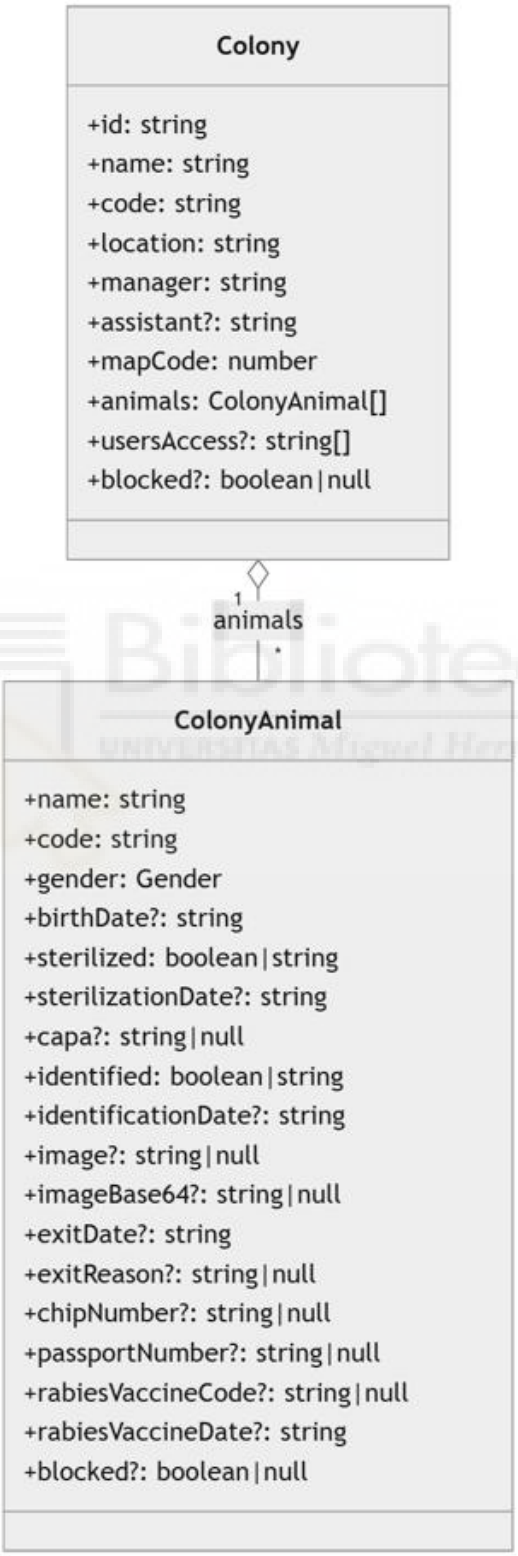

Ilustración 16: Modelo de datos de colonias

4.3.2.- Diagramas de secuencia

El proceso de autenticación y para la navegación a la página de administración se divide en tres etapas: la solicitud de inicio, la validación con **Google** y la autenticación con **Firebase**. Una vez acabado el proceso de inicio de sesión, el sistema comprueba si el usuario contiene roles permitidos para redirigir o no a la página de gestión.

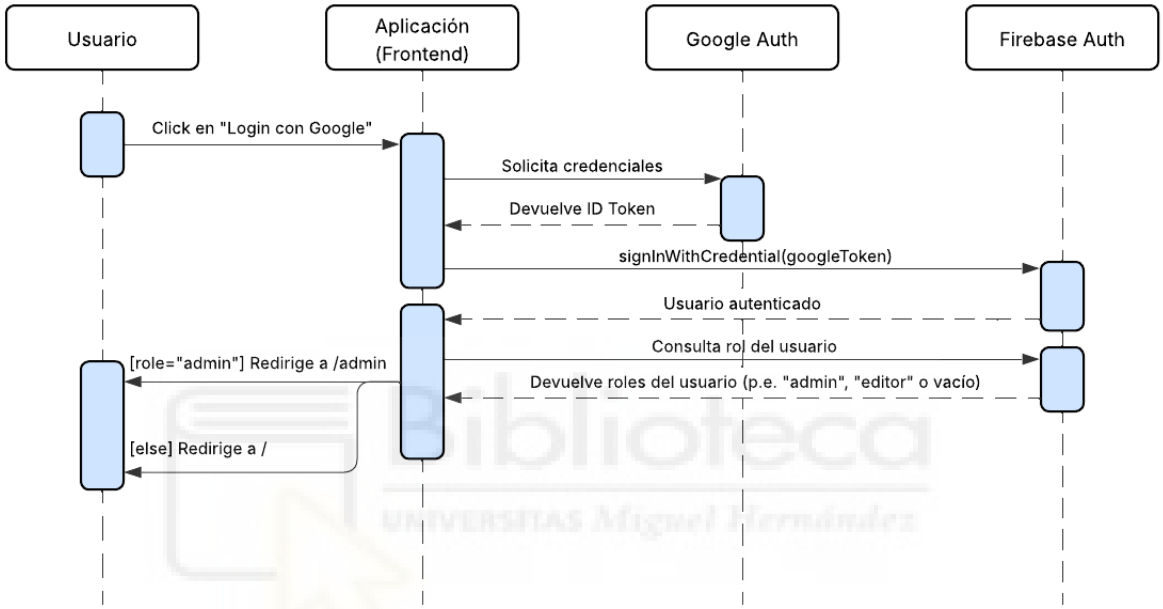

Ilustración 17: Diagrama de secuencia de autenticación

La obtención de los formularios de adopción se realiza mediante una llamada a *backend* y este, a su vez, se comunica con la **API** de **Google Sheets** para obtener los datos y devolverlos formateados a la aplicación *frontend*. A la hora de guardar, el proceso es similar, pero se realiza únicamente una actualización de las celdas modificadas en **Google Sheets**.

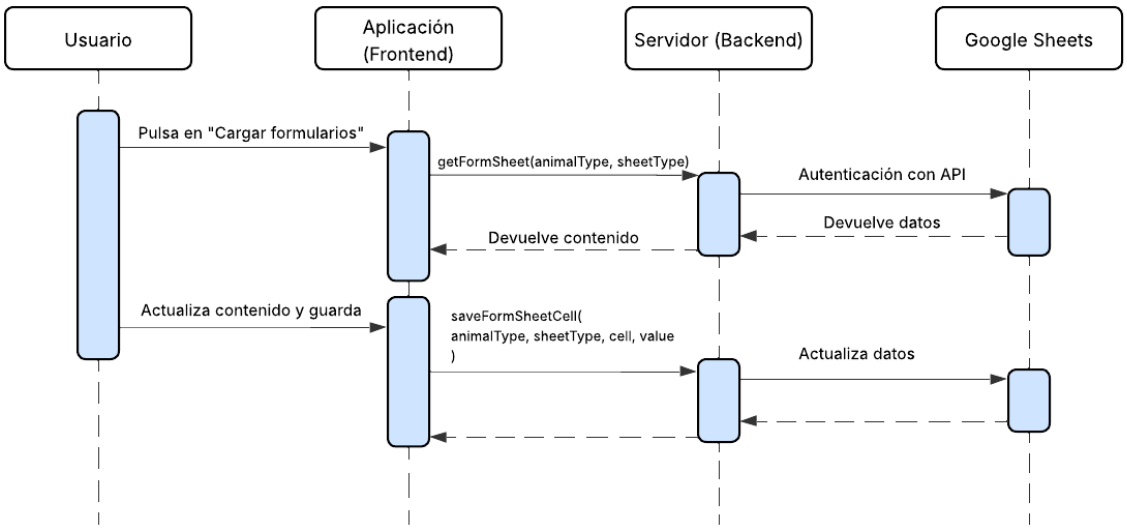

Ilustración 18: Diagrama de secuencia de gestionar formularios de adopción

La generación de fichas de los animales de las colonias se realiza tras la acción del usuario, indicando sobre qué animales de la colonia quiere generarlas. El *backend* se encarga de la renderización de las fichas a través de una plantilla y rellenando los campos que haya en la base de datos. Una vez completado el proceso de renderización de fichas, se autentica con la **API** de **Google Drive** y procede a la subida de los documentos.

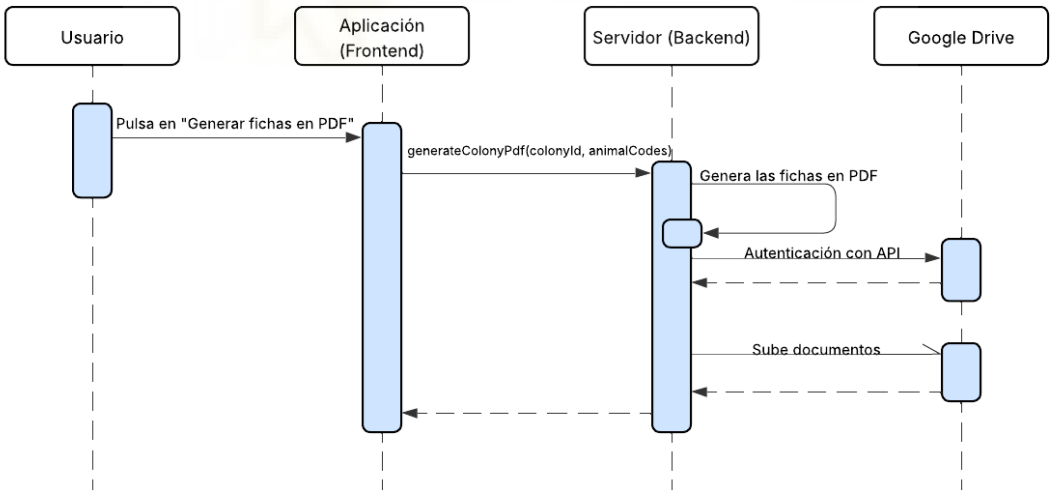

Ilustración 19: Diagrama de secuencia de generar fichas de colonias

La generación de alertas de seguimiento de los animales adoptados es un proceso que se ejecuta mediante **CRON** en el *backend*. El proceso obtiene los animales que tienen datos de seguimiento, comprueba si la fecha actual coincide con alguna de las fechas estimadas para la siguiente vacunación y, en caso de ser correcto, genera un nuevo documento en la base de datos.

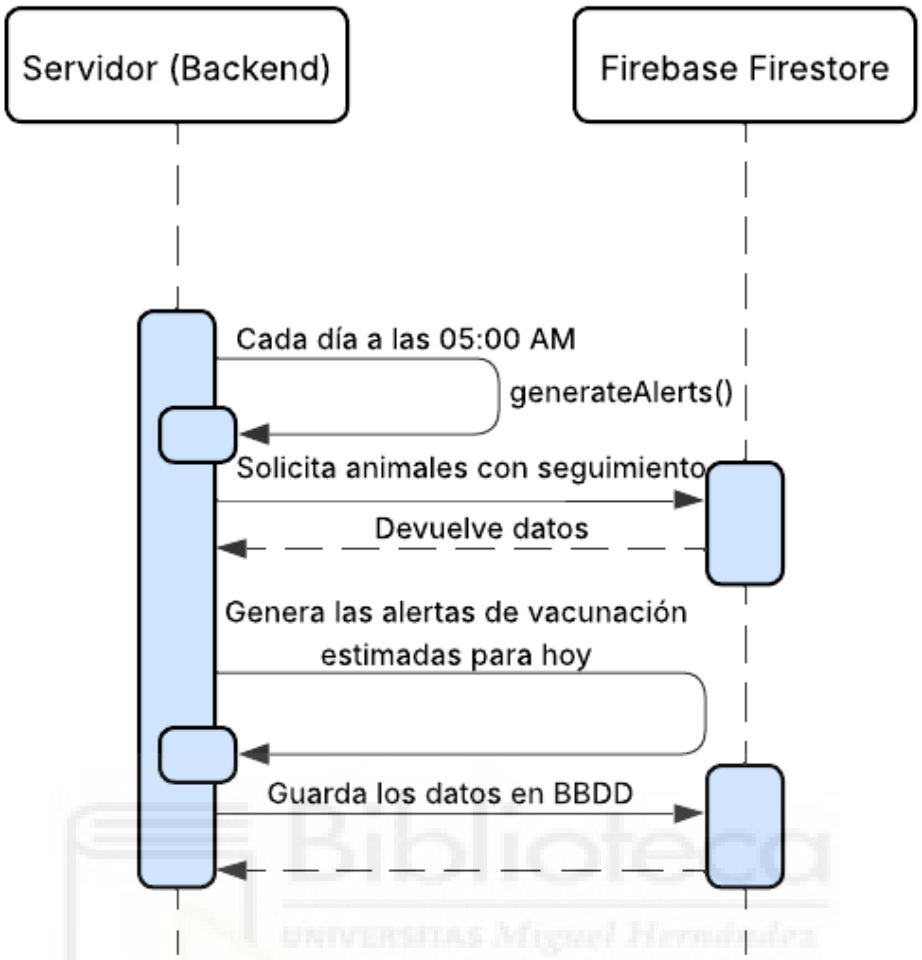

Ilustración 20: Diagrama de secuencia de generar alertas automáticas

4.3.3.- Diagramas de actividad

A continuación, se muestran los diagramas de actividad y procesos más relevantes de la aplicación, teniendo en cuenta los distintos caminos que pueden tomar dependiendo de la acción o estado de la aplicación.

El *guard* de autenticación permite la navegación a la página de administración si el usuario que ha iniciado sesión tiene los permisos necesarios. Este *guard* se ejecuta en cada intento de navegación a la ruta de */admin*.

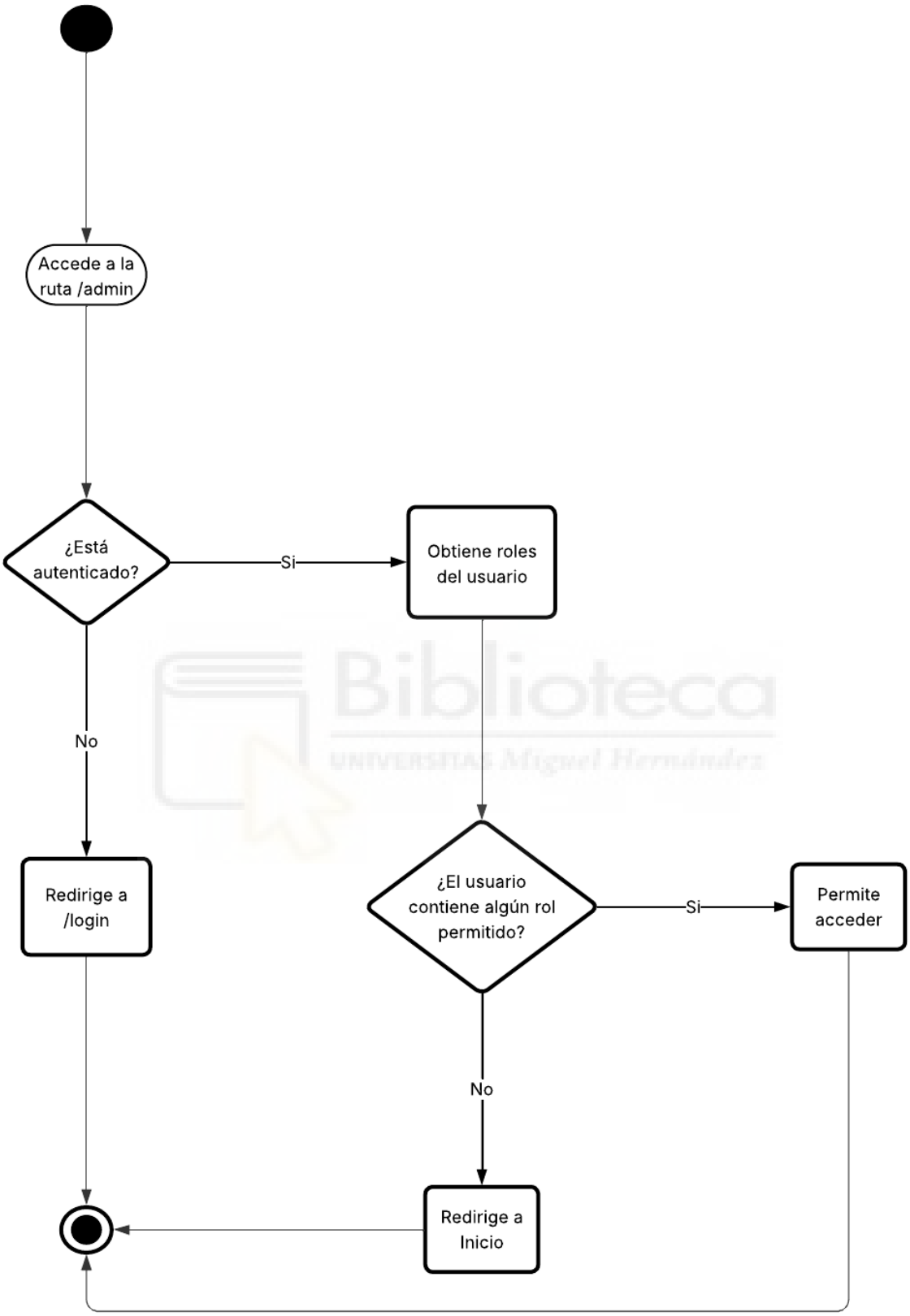

Ilustración 21: Diagrama de actividad de protección de rutas

La gestión de formularios de los adoptantes se realiza en varios marcos de responsabilidad. Primeramente, es el usuario o interesado en la adopción el que realiza el formulario de **Google Forms**, y este formulario es almacenado en una **hoja de cálculo Excel de Google**.

Seguidamente, son los voluntarios encargados de las adopciones quienes se encargan de leer y valorar cada una de las propuestas de adopción, pudiendo marcarlas como **No válidos** o bien continuar con el proceso de adopción poniéndose en contacto con la persona interesada y decidir si se realiza la adopción o no.

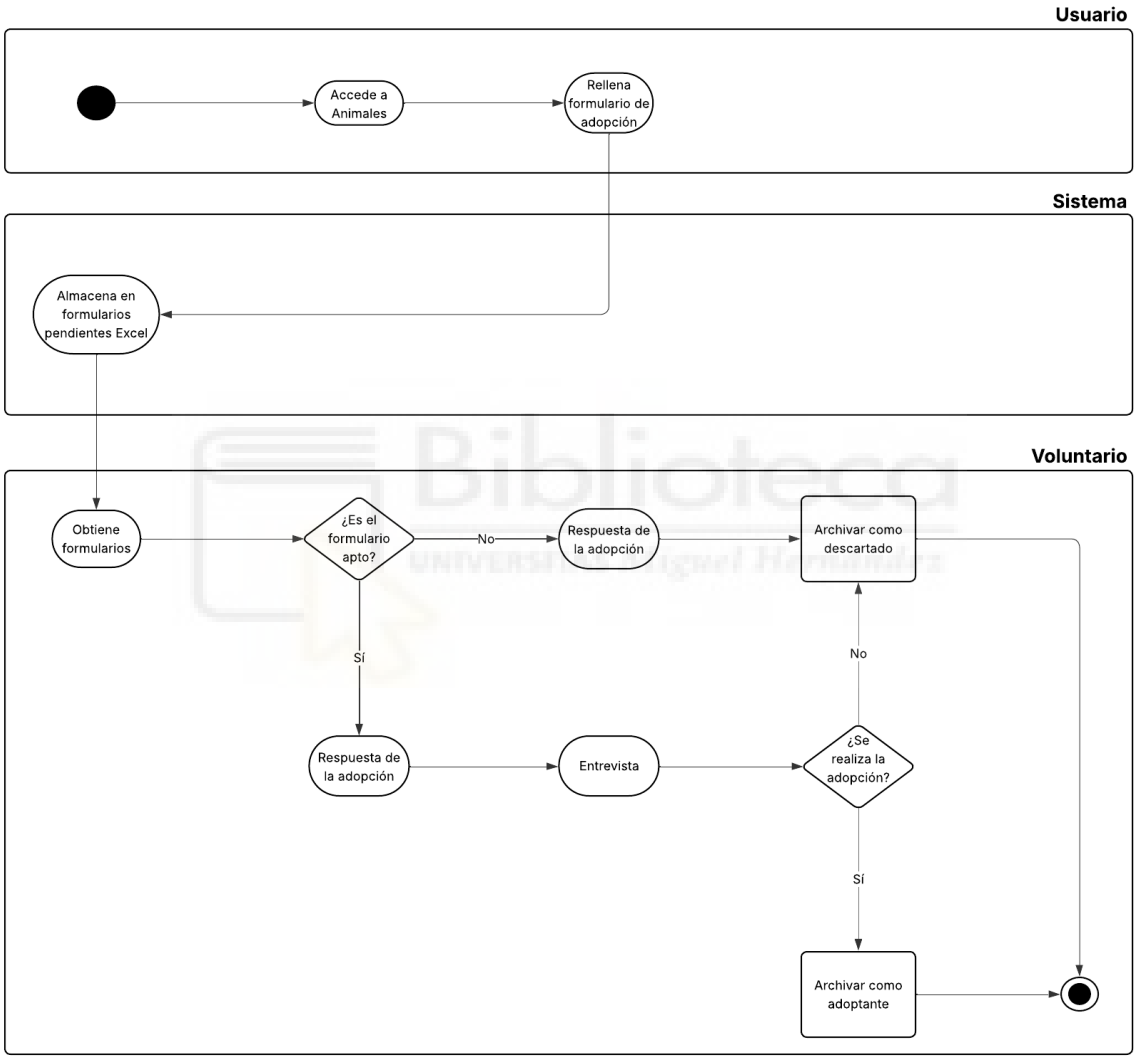

Ilustración 22: Diagrama de actividad de proceso de adopción

Añadir un nuevo animal se realiza a través de las tablas de animales, donde, una vez rellenado el formulario con los campos requeridos y si se decide confirmar la acción, se almacenarán los datos en la base de datos.

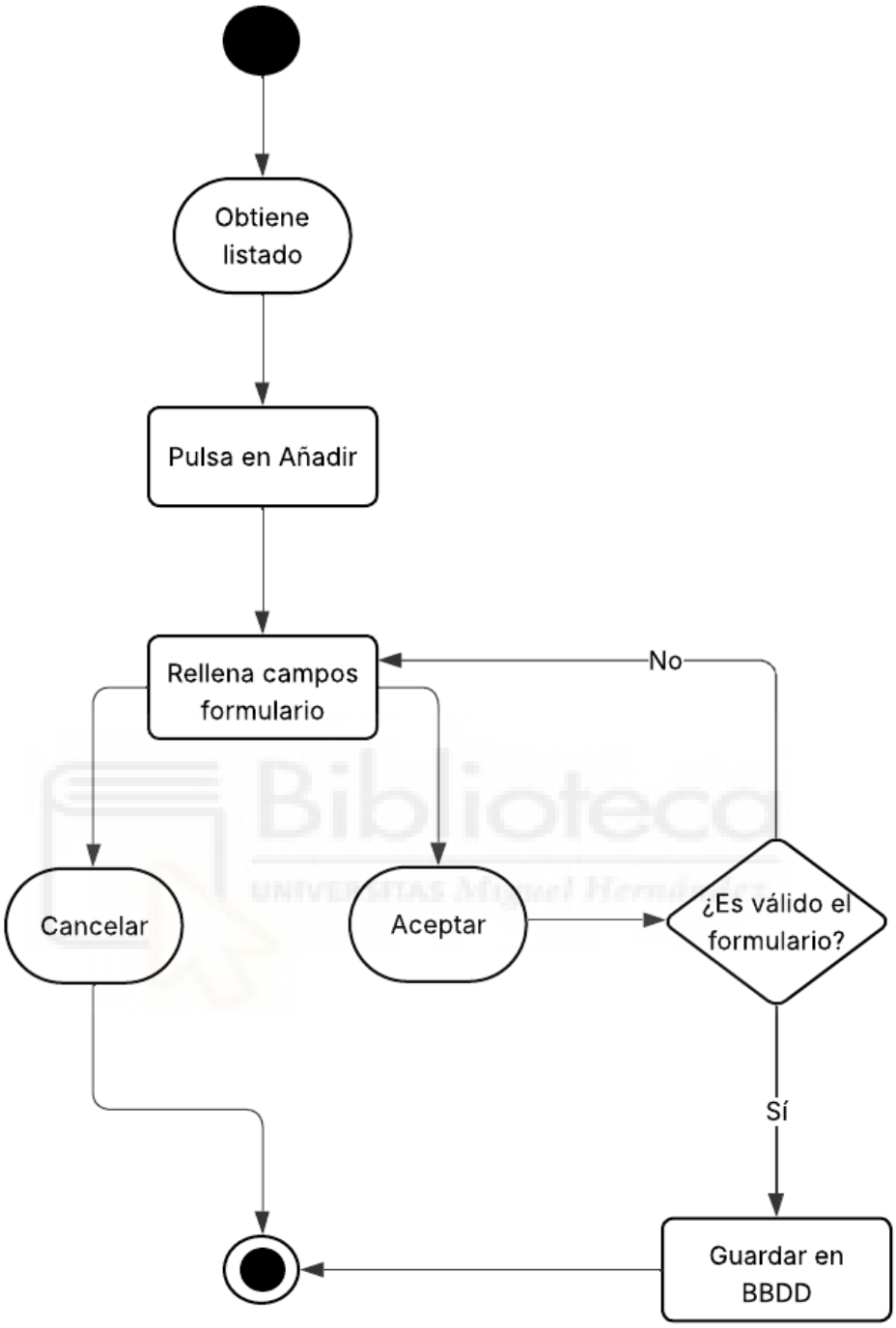

Ilustración 23: Diagrama de actividad de añadir un nuevo animal

El flujo de añadir animales de forma masiva a las colonias consiste en una acción que puede ahorrar mucho tiempo a los voluntarios. Una vez abierto el diálogo de carga masiva, se podrán realizar varias acciones con las que modificar o añadir animales en lote.

Se puede utilizar código en formato **CSV** con un formato concreto, definido por un texto de ayuda, para obtener todos los datos requeridos por cada animal, y si el formato obtenido es correcto se permitirá guardar en base de datos. Además, también se permite la obtención de

imágenes de los **PDF** y asignación a cada animal, en este caso para poder relacionar la imagen con el animal ya creado, el título del documento **PDF** debe coincidir con el identificador del animal.

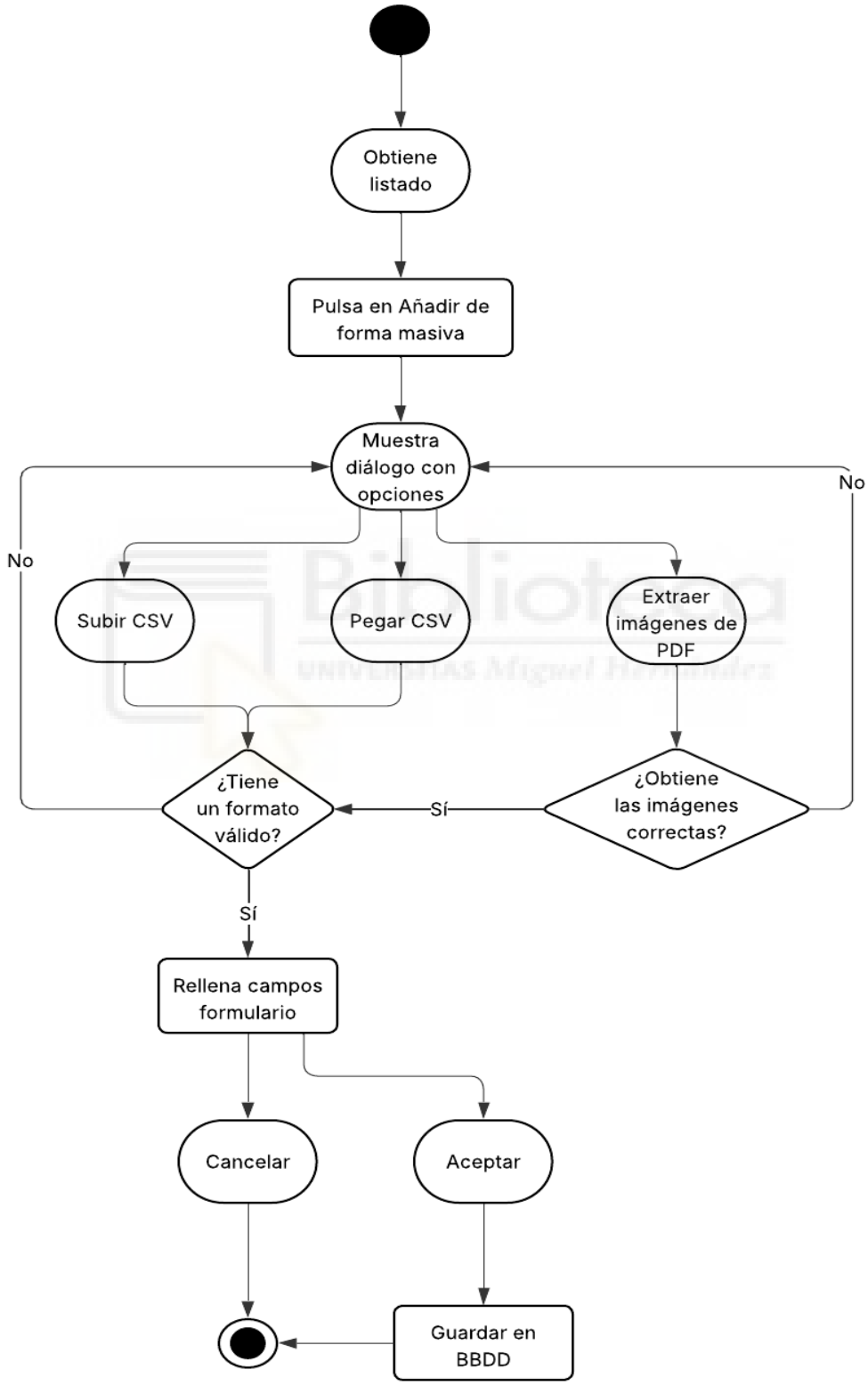

Ilustración 24: Diagrama de actividad de añadir animales a una colonia de forma masiva

4.3.4.- Diagramas de componentes

Para representar los diagramas de componentes, se ha escogido por los principales módulos y sus relaciones con clases abstractas, servicios y otros componentes.

El listado de animales presenta un componente común para los tres tipos (Perros, Gatos y Exóticos) y, dependiendo de la ruta a la que se navegue, una clase o servicio que obtiene los datos de *backend*.

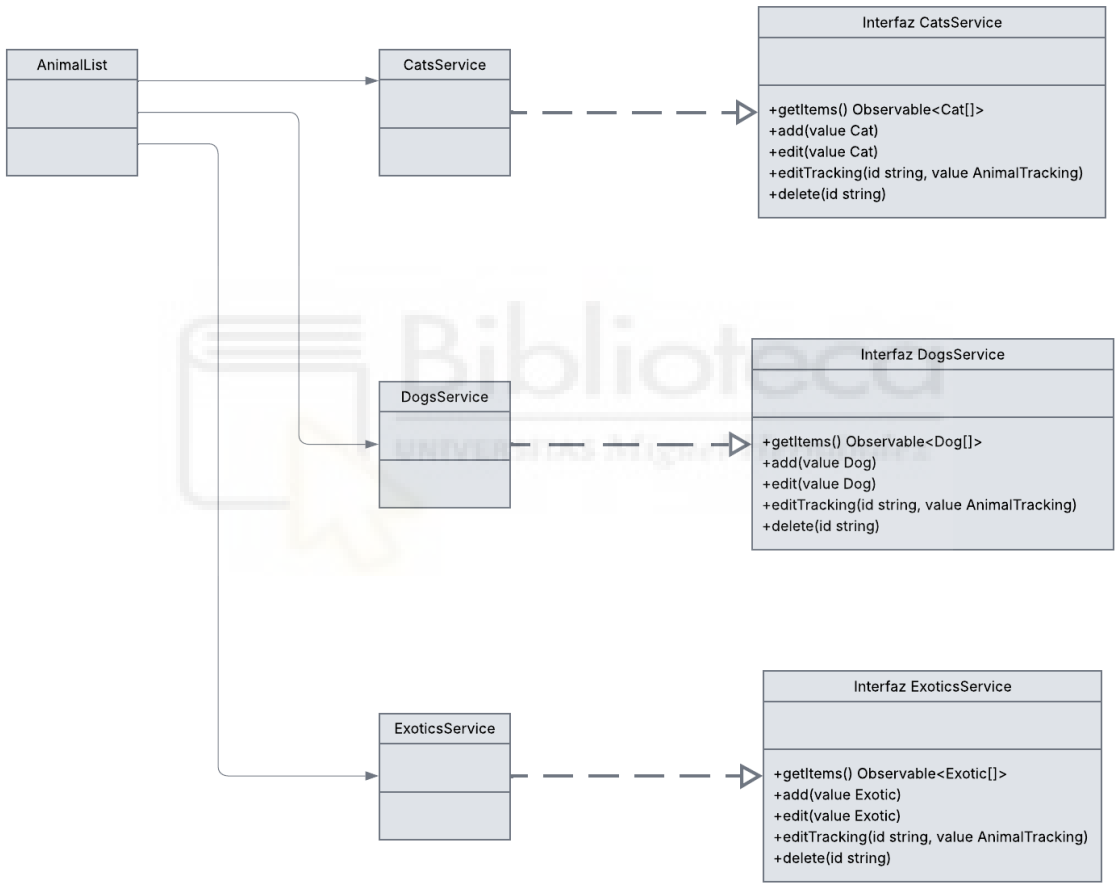

Ilustración 25: Diagrama de componentes de listado de animales

En cuanto a la parte de administración de animales, se ha implementado una clase abstracta con métodos y variables comunes para los tres tipos de animales. A su vez, cada tipo de animal implementa el servicio donde realizar las operaciones de lectura y escritura y los componentes desde donde se pueden modificar los detalles de cada animal.

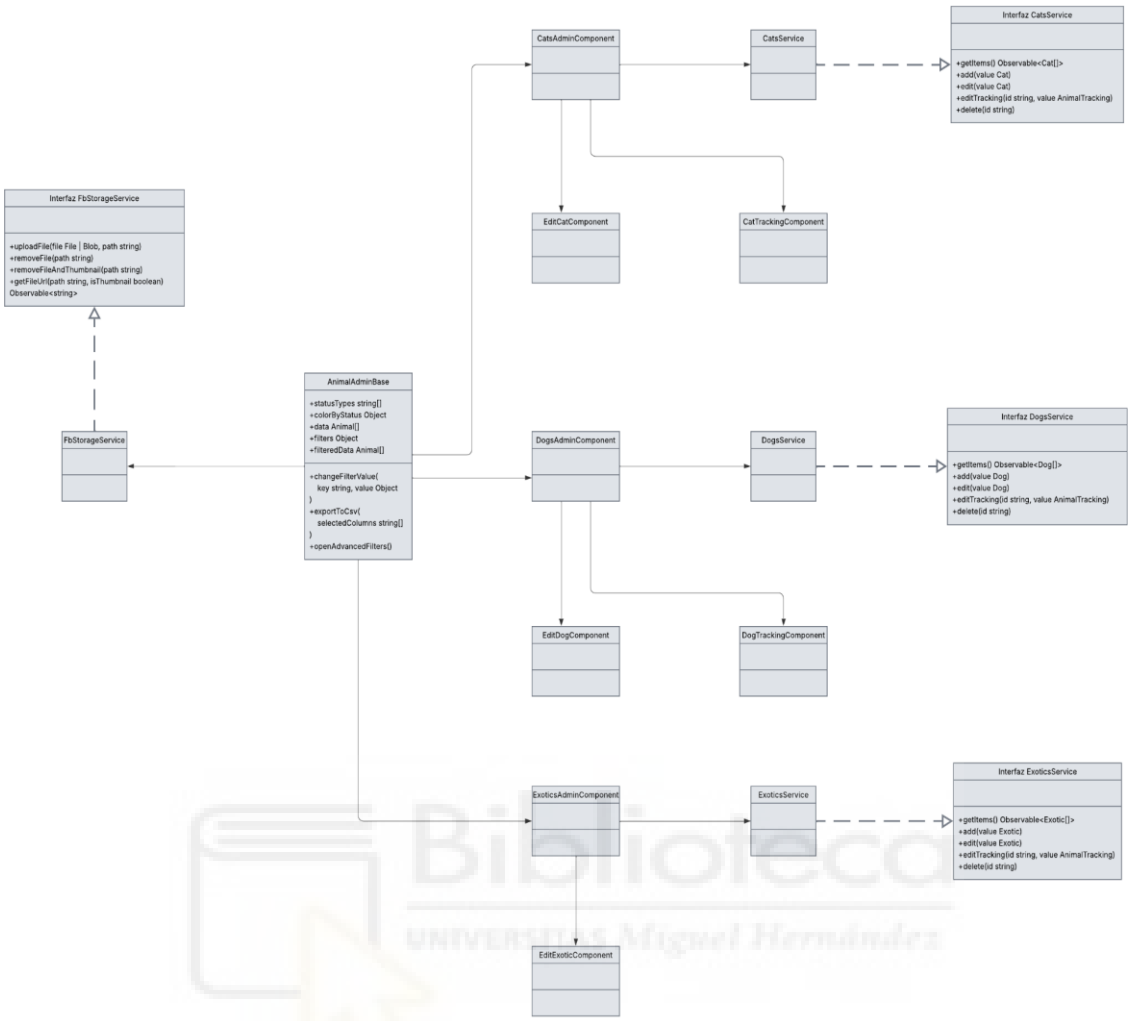

Ilustración 26: Diagrama de componentes de gestión de animales

El módulo de la tienda contiene un componente *Store*, que es la página pública de los productos, y un componente *ProductsAdmin*, que permite modificar los productos listados. Además, se implementa el uso del servicio *FbStorageService* que permite el almacenamiento de imágenes.

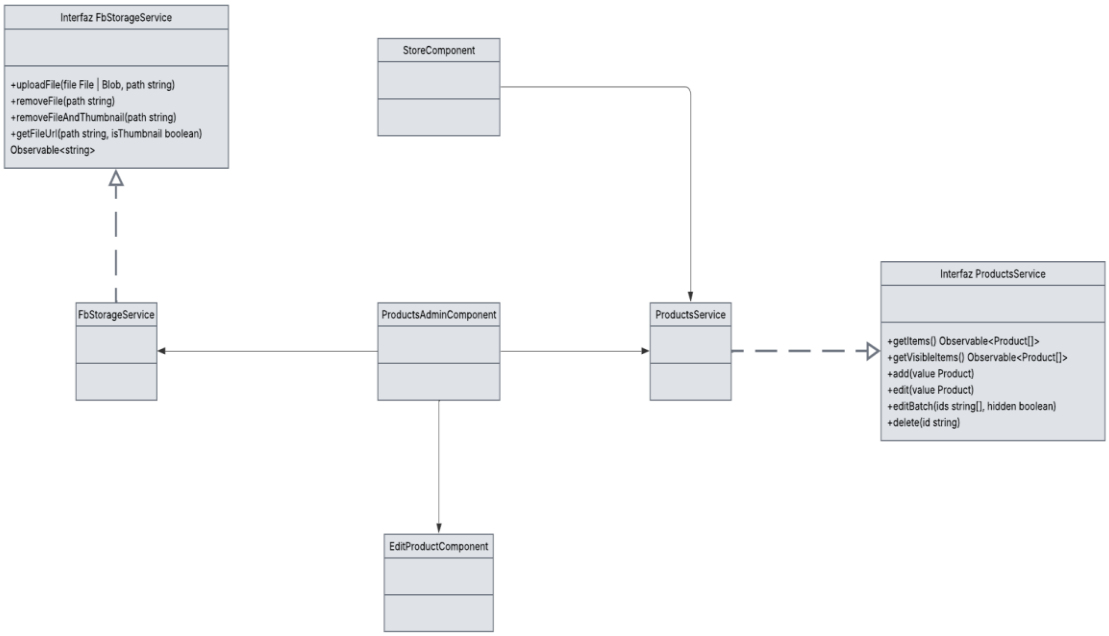

Ilustración 27: Diagrama de componentes de tienda

El módulo de gestión de colonias implementa su servicio específico *ColoniesService* para realizar operaciones de lectura y escritura en la base de datos y el servicio *FbStorageService* para el almacenamiento de imágenes.

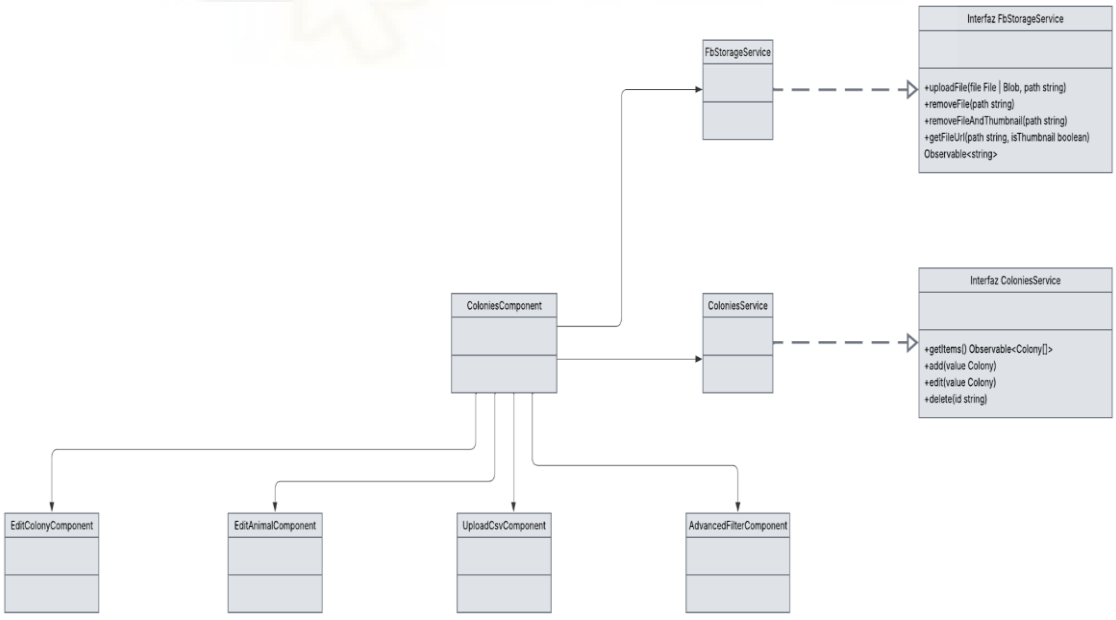

Ilustración 28: Diagrama de componentes de gestión de colonias

4.3.5.- Diagramas de distribución y despliegue

El sistema se basa en las siguientes partes:

- **Cliente:** Usuario que accede a la aplicación desde un navegador.
- **Firebase:** Plataforma intermediaria sobre los servicios de Google Cloud, utilizada para simplificar el desarrollo.
- **Google Cloud:** Plataforma de infraestructura donde se alojan todos los datos y código desplegado.

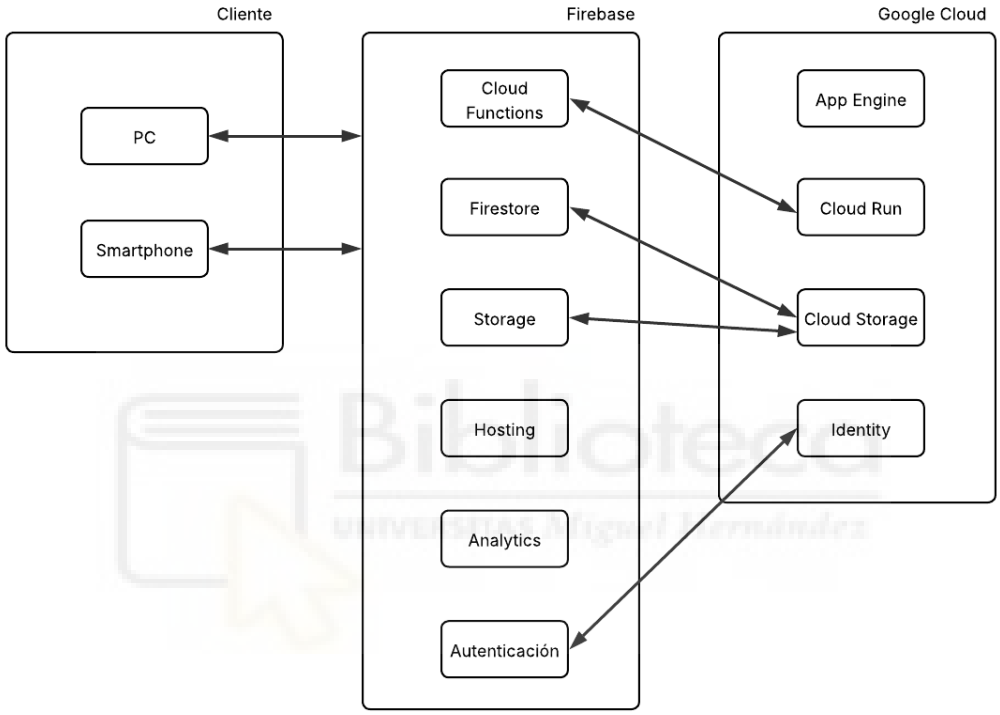

Ilustración 29: Diagrama de distribución en Firebase y Google Cloud

4.3.6.- Diagrama de navegación global

La navegación entre páginas de la aplicación se realiza a través de la cabecera, donde se pueden acceder a todas las páginas públicas y, si ha iniciado sesión, a la parte de gestión privada.

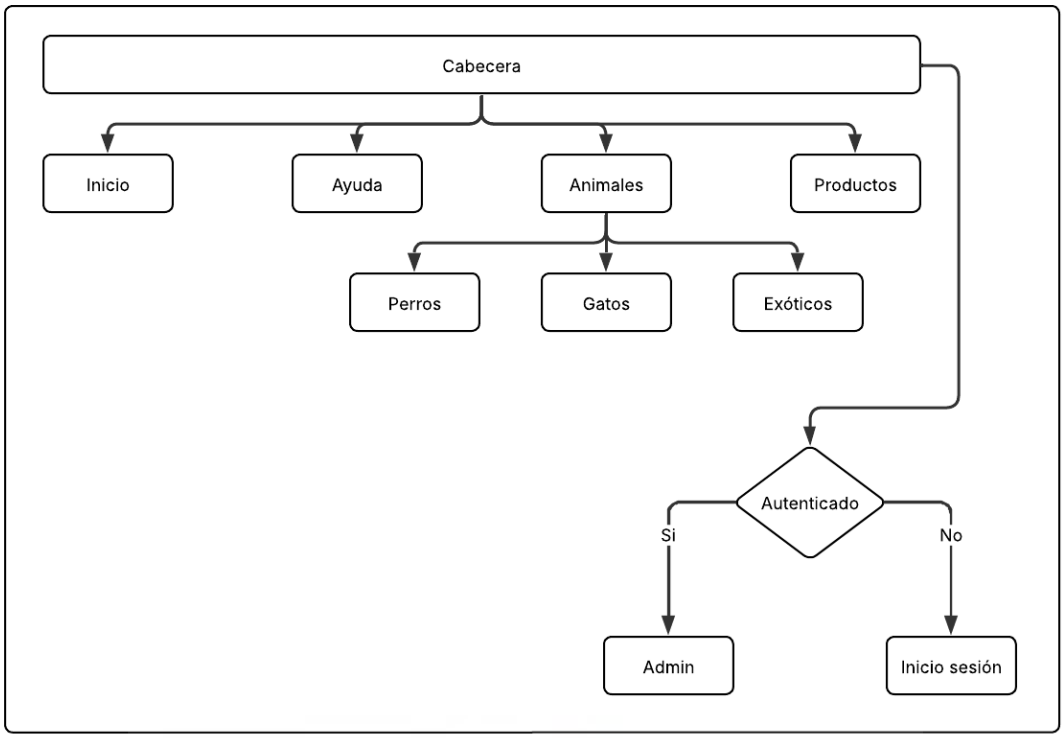

Ilustración 30: Diagrama de navegación en la aplicación

4.4.- Implementación

La aplicación se diferencia en dos partes fundamentales, la parte pública y la parte privada. De la parte pública se puede destacar el listado de animales disponibles, ya que es el escaparate para los visitantes. Y de la parte privada, se puede destacar la gestión de animales en adopción, los formularios de interesados para adoptar, la gestión de colonias y los permisos de usuarios.

4.4.1.- Parte pública

La parte pública se compone de cuatro páginas con información estática (Inicio y Ayuda) e información dinámica (Animales y Productos).

Se pueden ver los tres tipos de animales que hay en adopción (https://patas-sin-fronteras.web.app/animals).

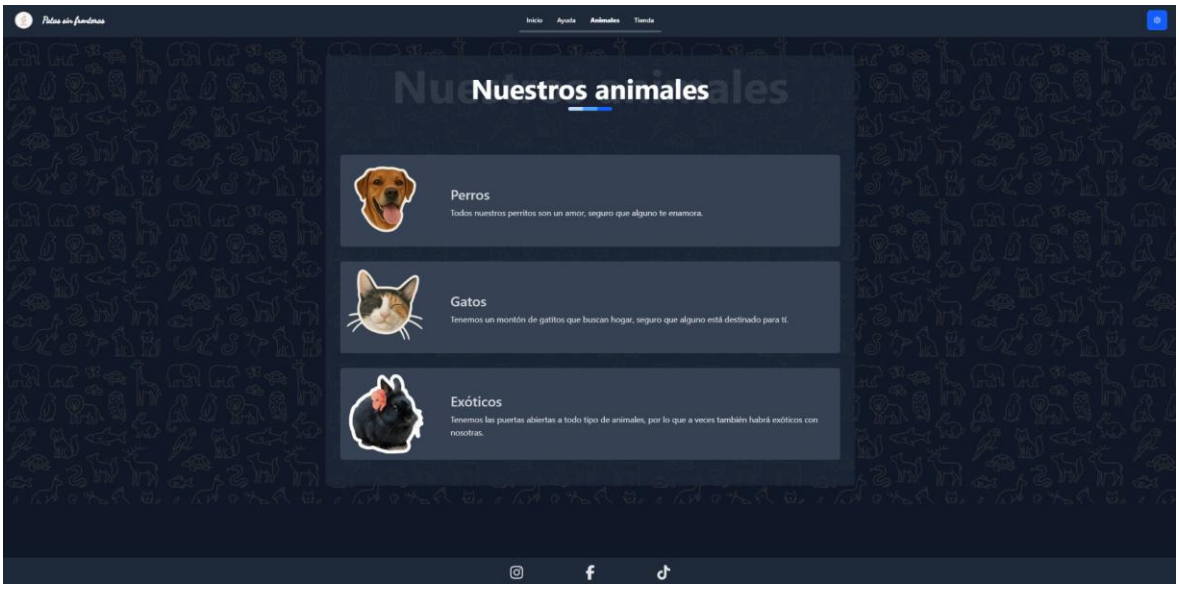

Ilustración 31: Página de tipos de animales

Al pulsar sobre alguno de los elementos, navegaremos al listado de todos los animales en adopción y adoptados, donde se puede ver toda la información específica de cada uno y los enlaces para rellenar los formularios de adopción.

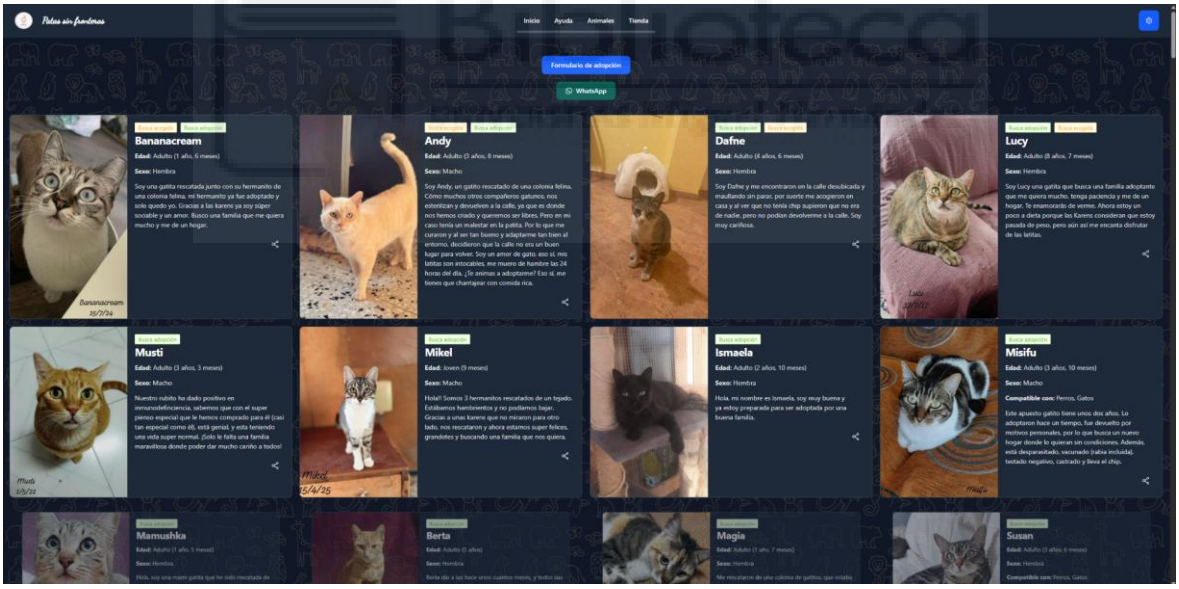

Ilustración 32: Página de listado de gatos

La ordenación de los animales se ha llevado a cabo mediante un algoritmo donde se dan distintas prioridades según los estados que tengan asignados.

```
export const getStatusPriority = (item: AnimalType): number => {
  const statusList = item.status;
  const completed = item.tracking?.completed;
  const statusWeights: Record<Status, number> = {
    urgent: -10,
    looking_for_reception: -5,
    looking_for_adoption: -4,
    recovery: -3,
    nursling: -3,
    reserved: 5,
    adopted: 10
  };

  return (statusList?.reduce((acc, status) => acc + statusWeights[status], 0) ?? 0) +
(completed ? 10 : 0);
};
```

Como se puede observar, un valor final más bajo, mostrará los animales antes en el listado. Al poder tener varios estados, se hace un sumatorio de todos los valores, y en caso de que se encuentre ya adoptado el animal, su prioridad baja para que no se muestre entre los primeros.

4.4.2.- Parte privada

Todas las gestiones de los datos de la aplicación se hacen mediante tablas, donde se muestra resumidamente los datos más importantes, y mediante diálogos contextuales se pueden todos los datos relacionados.

La gestión de los animales visibles para su adopción se hace a partir de la siguiente tabla. Los tres tipos de animales (Perros, Gatos y Exóticos) muestran la misma interfaz y funcionalidades casi idénticas. En las tablas se muestran, a modo de resumen, los datos más prioritarios.

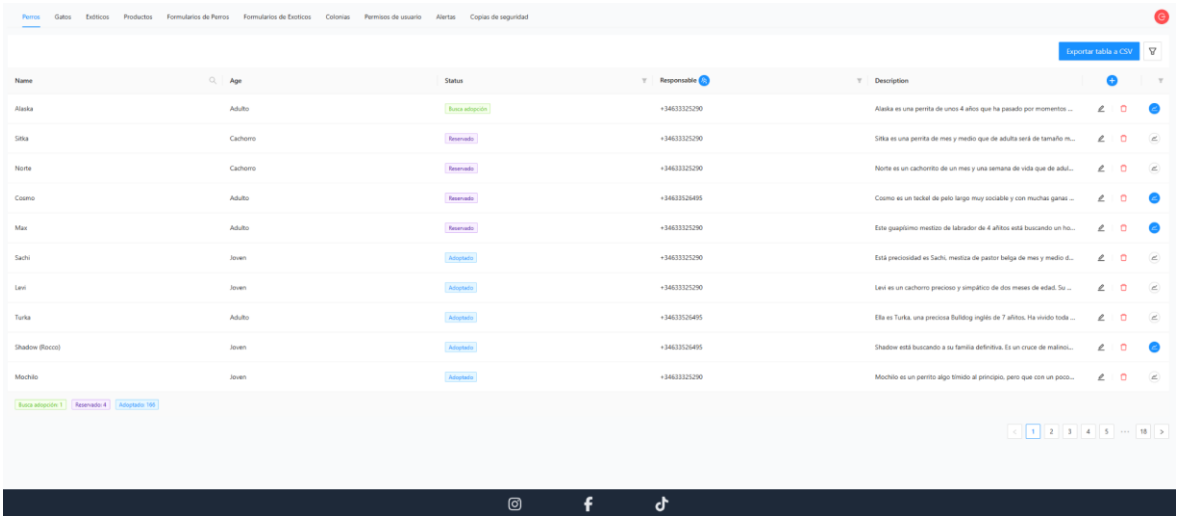

Ilustración 33: Página de gestión de animales

Si editamos algún animal, podremos ver todos los datos disponibles y poder modificarlos.

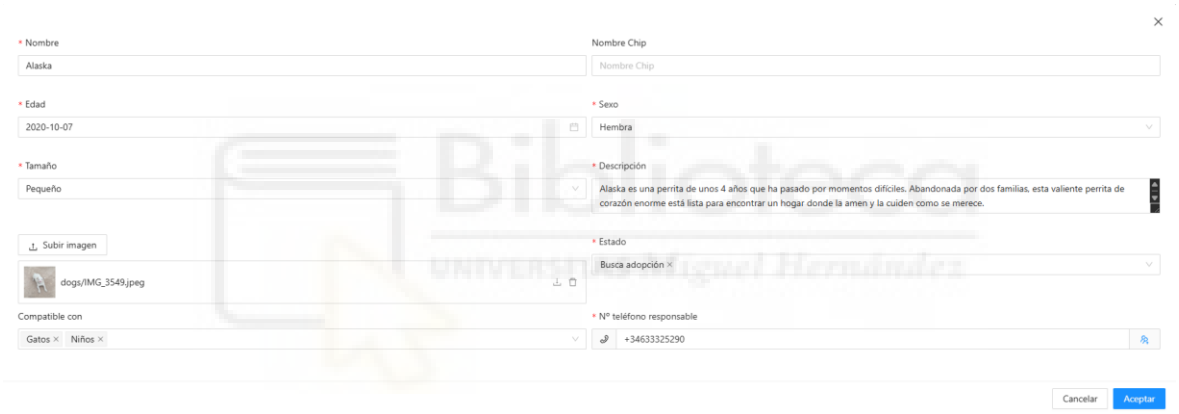

Ilustración 34: Página de editar animal

También se puede acceder a las opciones de seguimiento, donde podremos indicar próximas vacunas, vacunaciones ya realizadas o datos como número de chip o pasaporte.

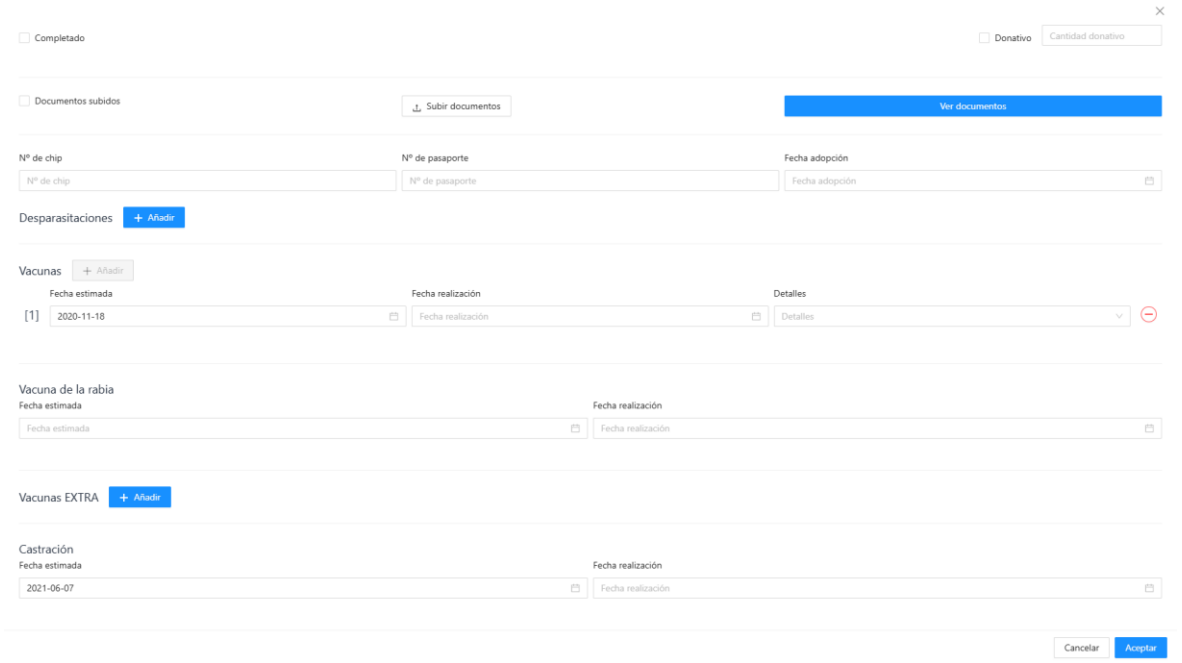

Ilustración 35: Página de editar seguimiento de animal

La lectura de los formularios de adopción se muestra en una tabla. Los datos de esta tabla son extraídos del documento **Excel** donde los vuelca **Google Forms**. Se han definido los colores verde y rojo para poder visualizar rápidamente qué formularios de adoptantes son válidos o no, pudiendo realizar acciones sobre cada uno de ellos, como por ejemplo enviar un mensaje predefinido por **WhatsApp**, mostrar el formulario completo o añadir comentarios tras las interacciones entre los voluntarios y los adoptantes interesados.

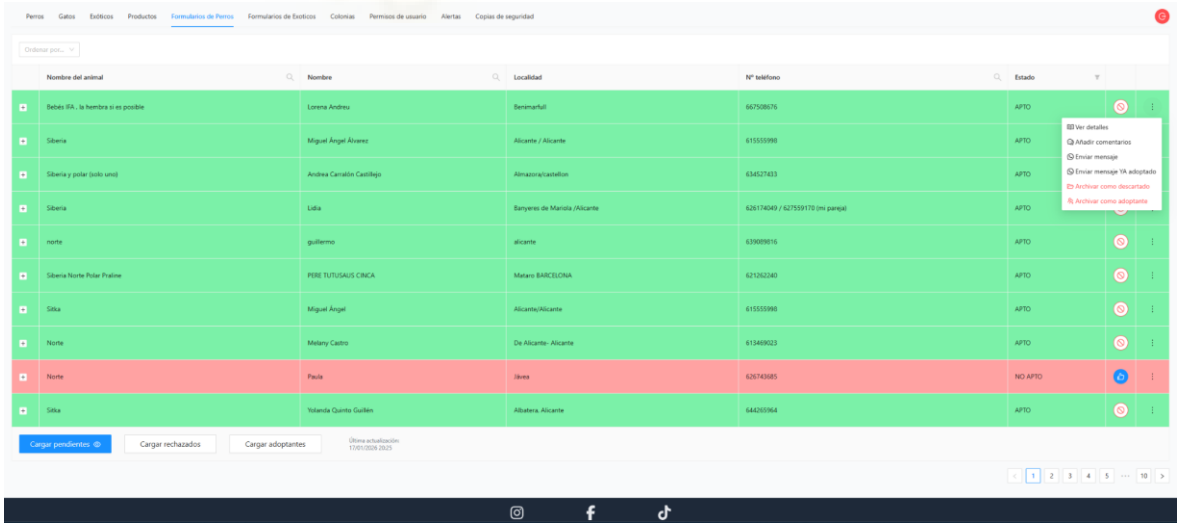

Ilustración 36: Página de gestión de formularios de adopción

Las colonias se muestran en tablas anidadas y expandibles, donde cada colonia contiene varios gatos. Se pueden modificar de manera rápida los datos más susceptibles a cambio, mientras que los demás datos se pueden modificar mediante un menú contextual.

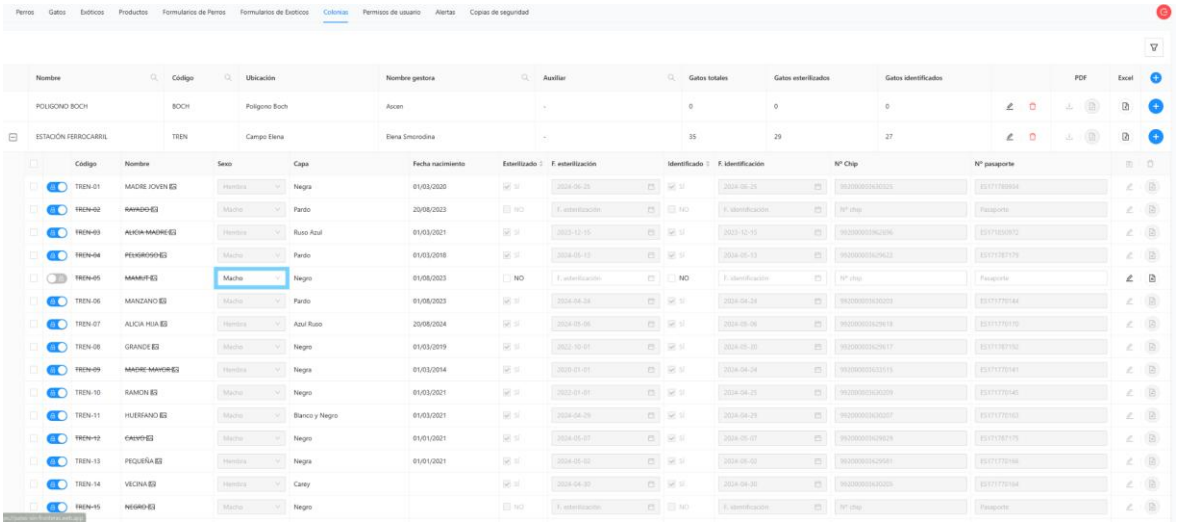

Ilustración 37: Página de gestión de colonias

Con anterioridad a la creación de este módulo, todos los documentos se generaban en **PDF** escribiendo cada campo uno a uno con algún editor de **PDFs**. Tras la creación del módulo, era interesante añadir una funcionalidad para poder volcar todos esos datos y almacenarlos en la base de datos de manera rápida y sencilla.

Se creó un diálogo donde poder subir un documento **CSV** con los datos obtenidos de los **PDFs**.

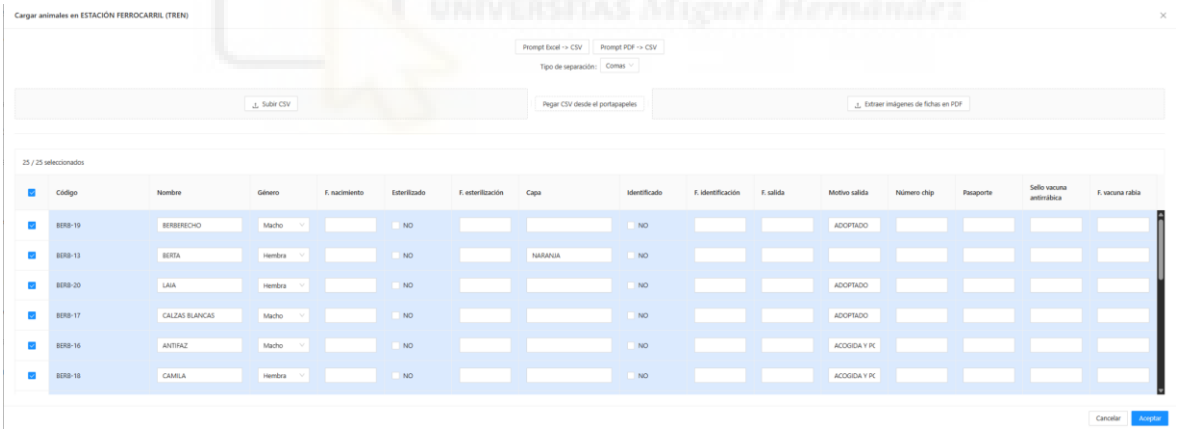

Ilustración 38: Página de actualizar colonias de forma masiva

Con ayuda de la **IA** (como por ejemplo **ChatGPT** o **Gemini**), se preparó un *prompt* con toda la información necesaria y cómo generar cada **clave/valor** de los datos de los PDFs y obtener un resultado en **CSV**, dando muy buenos resultados y acelerando el proceso de volcado de datos de forma masiva.

```
readonly promptPdfToCsv =
    'A partir de los archivos PDF adjuntos, genera una tabla que contenga las siguientes columnas:\n' +
    '-  name: Apartado "Nombre". Si no existe el valor, dejar el campo vacío;\n' +
    '-  code: Apartado "CÓDIGO". Si no existe el valor, dejar el campo vacío;\n' +
    '-   gender: Apartado "SEXO". Si el valor es "macho", traducirlo a "male", si es "hembra" traducirlo a "female" y en caso de estar vacío poner como valor "unknown";\n' +
    '-   birthDate: Apartado "Fecha nacimiento". La fecha está en formato dd/MM/yyy, transfórmala a yyyy/MM/dd. Si no hay fecha, dejar el campo vacío. Si solo hay año, poner mes 01 y día 01.;\n' +
    '-  sterilized: Checkbox del apartado "Esterilizado". Si está marcado, poner valor "true"; si no, valor "false". Deben ser en minúsculas;\n' +
    '-   sterilizationDate: Apartado "Fecha esterilización". La fecha está en formato dd/MM/yyy, transfórmala a yyyy/MM/dd. Si no hay fecha, dejar el campo vacío. Si solo hay año, poner mes 01 y día 01.;\n' +
    '-  capa: Apartado "Capa". Si no existe el valor, dejar el campo vacío;\n' +
    '-  identified: Si existe imagen, poner valor "true"; si no, valor "false". Deben ser en minúsculas.;\n' +
    '-   identificationDate: Apartado "Fecha identificación". La fecha está en formato dd/MM/yyy, transfórmala a yyyy/MM/dd. Si no hay fecha, dejar el campo vacío. Si solo hay año, poner mes 01 y día 01.;\n' +
    '-  exitDate: Apartado "Fecha salida de la colonia". La fecha está en formato dd/MM/yyy, transfórmala a yyyy/MM/dd. Si no hay fecha, dejar el campo vacío. Si solo hay año, poner mes 01 y día 01.;\n' +
    '-  exitReason: Apartado "Causa de su salida". Si no existe el valor, dejar el campo vacío;\n' +
    '-  chipNumber: Apartado "Microchip". Si no existe el valor, dejar el campo vacío;\n' +
    '-   passportNumber: Apartado "Pasaporte". Si no existe el valor, dejar el campo vacío;\n' +
    '-   rabiesVaccineCode: En la tabla "Vacunaciones", extraer el valor de la columna "Producto". Si no existe el valor, dejar el campo vacío;\n' +
    '-   rabiesVaccineDate: En la tabla "Vacunaciones", extraer el valor de la columna "Fecha". La fecha está en formato dd/MM/yyy, transfórmala a yyyy/MM/dd. Si no hay fecha, dejar el campo vacío. Si solo hay año, poner mes 01 y día 01.;';
```

Las fichas de los animales también contienen imágenes, por lo que era crucial automatizar la obtención de estas. Con la ayuda de la librería **pdf-lib** [18], se pudo gestionar los elementos que contienen los **PDFs** y obtener las imágenes para poder almacenarlas en la base de datos.

```
const pdfImages: { name: string; image: Blob }[] = [];
for (const file of event) {
  if (!file.originFileObj) {
    continue;
  }

  const pdfDoc = await PDFDocument.load(await file.originFileObj.arrayBuffer());

  const [firstPage] = pdfDoc.getPages();
  const images: { name: string; image: Blob }[] = [];

  for (const [, object] of firstPage.doc.context.enumerateIndirectObjects()) {
    const obj = object as PDFRawStream;
    if (!obj?.dict) {
      continue;
    }

    const subtype = obj.dict.get(PDFName.of('Subtype'));
    if (!subtype || subtype.toString() !== '/Image') {
      continue;
    }

    const contents = obj.contents;

    if (!contents) {
      continue;
    }

    const filter = obj.dict.get(PDFName.of('Filter'));
    const filterStr = filter?.toString?.() ?? '';

    if (filterStr.includes('DCTDecode')) {
      images.push({
        name: file.originFileObj.name,
        image: new Blob([contents as BlobPart], { type: 'image/jpeg' })
      });
    }
  }

  const image = images.at(-1);
  if (image) {
    pdfImages.push(image);
  }
}
```

El algoritmo busca los elementos de tipo imagen en el **PDF**, almacenando únicamente la que encuentra en último lugar, ya que siempre coincide con la imagen del animal. Este algoritmo es mejorable si se buscase entre las imágenes obtenidas la que mayor peso de almacenamiento tenga, ya que los **PDFs** contienen logos y puede dar falsos positivos.

La administración y asignación de los roles permite añadir un nuevo usuario a través de su correo electrónico o bien modificar los usuarios que ya tengan algún rol.

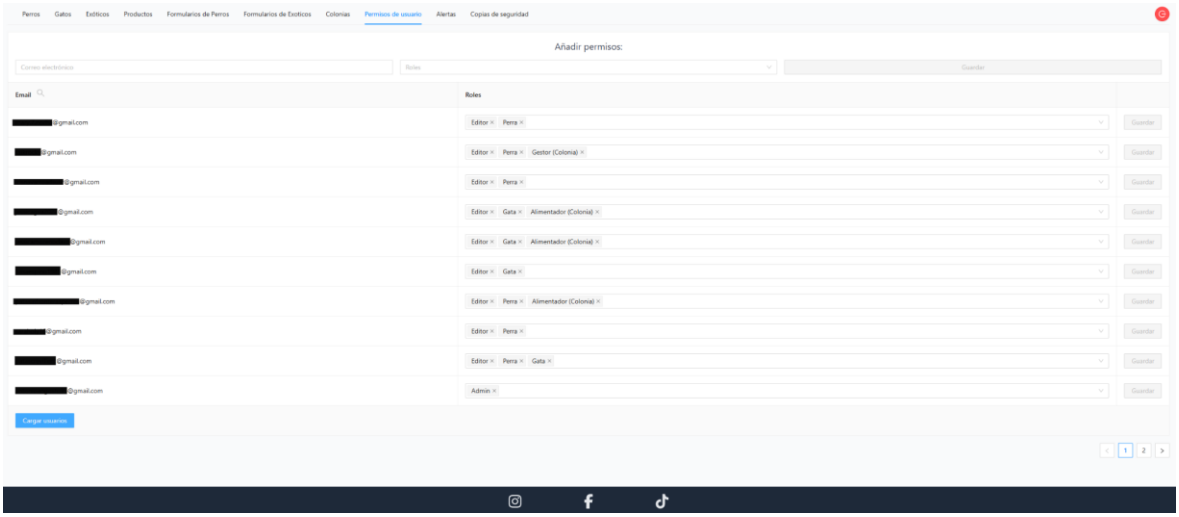

Ilustración 39: Página de permisos de usuario

4.5.- Pruebas de rendimiento y accesibilidad

Como se comentó en apartado 3.5.4.- Auditoría de código, se ha utilizado la herramienta **Lighthouse** para auditar la calidad, rendimiento y la accesibilidad de la aplicación. Se han ejecutado y optimizado las páginas de mayor tráfico y de acceso público.

4.5.1.- Auditoría página de Inicio

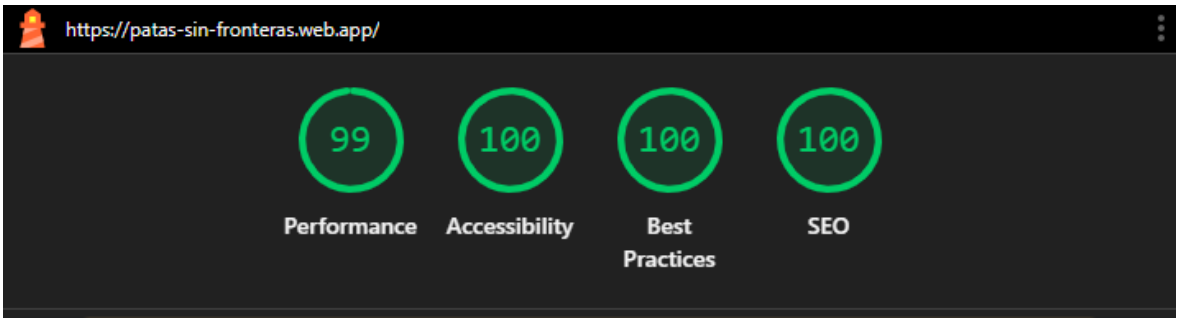

Ilustración 40: Resultados Lighthouse página inicio

La página de inicio muestra una optimización óptima en todos los parámetros medidos. Es especialmente importante esta optimización en la página de Inicio ya que es la puerta de entrada principal a la aplicación por los usuarios y los robots de motores de búsqueda.

4.5.2.- Auditoría página de Ayuda

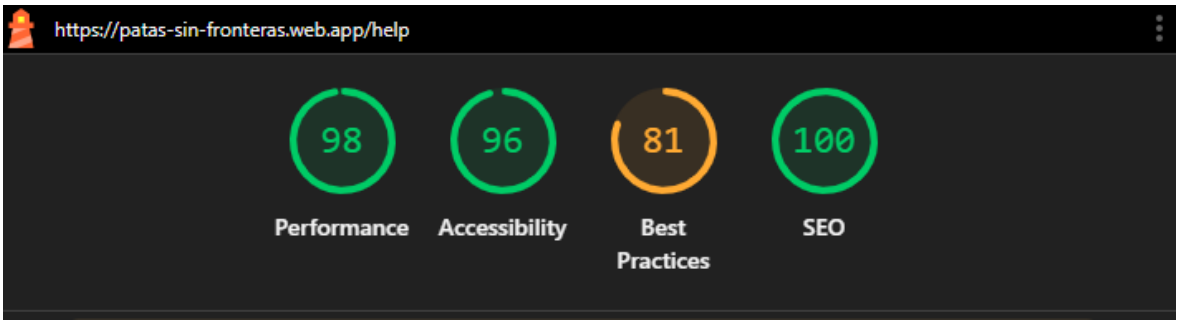

Ilustración 41: Resultados Lighthouse página ayuda

En la página de Ayuda también se obtienen valores óptimos en casi todos los parámetros. El apartado de mejores prácticas baja debido al uso de la dependencia **PayPal**, indicando que se utilizan **APIs** desactualizadas.

4.5.3.- Auditoría páginas de Animales

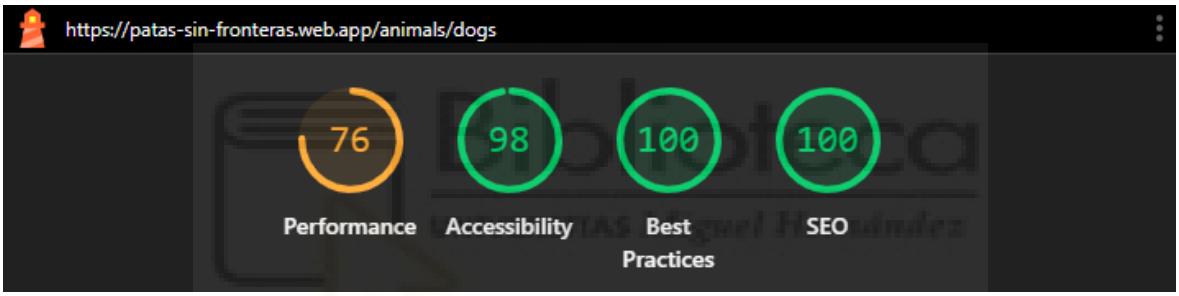

Ilustración 42: Resultados Lighthouse página listado de perros

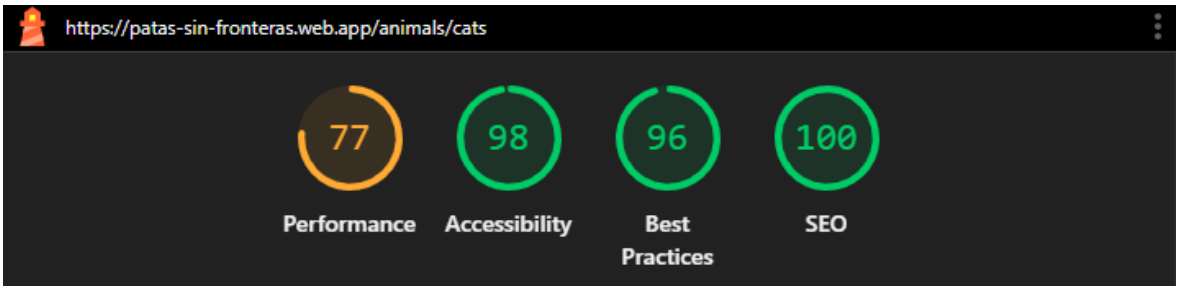

Ilustración 43: Resultados Lighthouse página listado de gatos

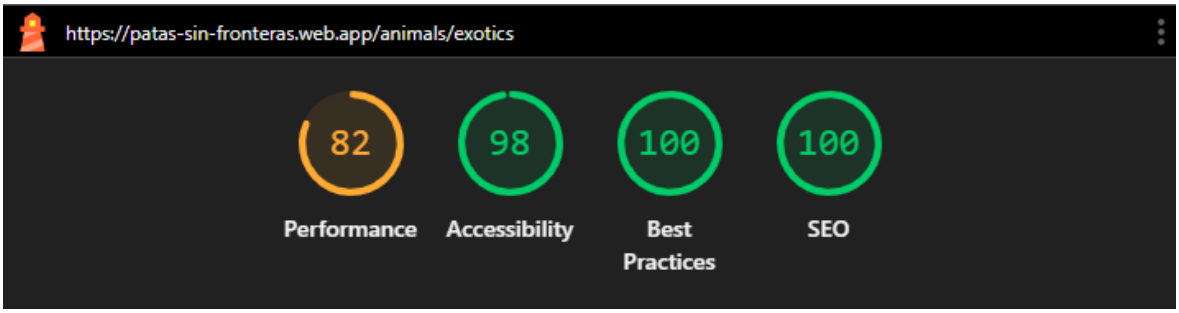

Ilustración 44: Resultados Lighthouse página listado de exóticos

En las páginas de listado de los distintos tipos de animales, se puede observar una bajada en la nota del rendimiento, esto es debido a la cantidad de imágenes que se muestran. Mientras que los demás parámetros siguen obteniendo una buena puntuación.

4.6.- Implantación y desarrollo

En cuanto a la implantación de la aplicación, actualmente está desplegada y funcionando en el dominio **https://patas-sin-fronteras.web.app/**. A la hora de desarrollar y probar localmente los cambios, se hace mediante comandos de **npm**:

- **npm run start:** Compila y ejecuta localmente la aplicación *frontend*. Se puede acceder a través de la **URL** **http://localhost:4200/**. Este modo permite que, si se realiza cualquier cambio en el código, automáticamente se refresca la aplicación con los nuevos cambios.
- **npm run serve:functions** / **npm run serve:functions:windows:** Compila y ejecuta la aplicación *backend* localmente. Se distinguen dos tipos de comandos debido a que se necesita declarar una variable de entorno en el sistema, ya que en entornos **UNIX** se declara con el comando ***`export`*** y en **Windows** con el comando ***`set`***.
- **npm run lint:** Inicia la ejecución de **Prettier** y **ESLint** en todo el proyecto, modifica los errores encontrados que puedan autocorregirse y muestra un listado de los problemas encontrados no corregidos.

También se disponen de comandos para compilar y desplegar la aplicación:

- **npm run build:** Compila la aplicación *frontend* en modo producción, generando un directorio con todos los archivos listos para desplegar.
- **npm run build:functions:** Compila la aplicación *backend* en modo producción, generando un directorio con todos los archivos listos para desplegar.
- **npm run deploy:** Ejecuta internamente el comando de compilar el *frontend* y después sube los archivos generados al *hosting* de **Firebase**. Además, incrementa la versión del **package.json** para que quede reflejado en **Git** que se ha desplegado una versión.
- **npm run deploy:functions:** Ejecuta internamente el comando de compilar el *backend* y después sube todos los archivos de funciones al servidor. Además, incrementa la versión del **package.json** del *backend* para que quede reflejado en **Git** que se ha desplegado una versión.

Capítulo 5
Conclusiones y trabajo futuro

5.1.- CONCLUSIONES

La conclusión más relevante de este trabajo es que se han alcanzado al completo los objetivos generales y específicos definidos en el apartado 1.3.- OBJETIVOS de esta memoria. La plataforma no es solo funcional, sino que se adapta a las necesidades particulares de la **Asociación "Patas Sin Fronteras"**. Así se ha conseguido cumplir cada uno de los objetivos específicos:

- **Gestión de expedientes:** Se ha desarrollado una base de datos centralizada que sustituye a los registros físicos, permitiendo el acceso de todos los voluntarios al historial completo del animal desde cualquier dispositivo.
- **Trazabilidad sanitaria con sistema de alarmas:** Se ha desarrollado un sistema de alarmas que genera una notificación cuando se llega a la fecha de vencimiento de las vacunas o fecha recomendada de esterilización.
- **Gestión de documentación:** La integración de la **API** de **Google Drive** ha facilitado la organización automática de la documentación legal correspondiente a cada animal, clasificándolos por carpetas individuales [19].

- **Captación de fondos y colaboración:** La web presenta un apartado de captación de recursos, recogiendo las diferentes maneras de realizar donaciones para los colaboradores externos.
- **Tienda Online:** Se ha creado un apartado en la web correspondiente a los productos de venta, y debido a que se buscaba evitar los costes de pasarela de pago, se ha generado un flujo desde la web hasta el **WhatsApp** de una de las voluntarias, que genera el pedido, pudiendo realizarse el pago de forma externa a la web.
- **Gestión de candidatos a la adopción:** La selección de adoptantes se ha facilitado generando un flujo de recogida de respuestas de solicitud de adopción. Tras recibir la solicitud, se clasifica y se responde a través de **WhatsApp** mediante mensajes predefinidos con el chat correspondiente, reduciendo el tiempo de espera y el esfuerzo por parte de los voluntarios.
- **Gestión de colonias felinas (Método CER):** Se ha diseñado una herramienta específica para la gestión de colonias, que genera de manera automatizada desde la propia web las fichas individuales de cada animal. Además, almacena de forma organizada y con el formato específico de la administración pública en **Google Drive**.

Por otra parte, en cuanto a conclusiones técnicas, al no tener un cliente definido que realizase aportaciones monetarias ni una planificación estricta y definida, este proyecto se ha utilizado también para mejorar los conocimientos técnicos, pudiendo utilizar novedades que se incluyen en los lenguajes web modernos, nuevas funcionalidades que los *frameworks* aportan tras cada versión y tecnologías de peso en el mercado y que no se hacen uso de ellas en el ámbito laboral.

5.2.- POSIBLES DESARROLLOS FUTUROS

Dado que se trata de un proyecto escalable y no está atado a unas líneas de entrega estrictas y definidas, este proyecto se utiliza para poner en práctica nuevas funcionalidades y desarrollos de manera subjetiva. A continuación, se definen algunos de los desarrollos más viables a futuro que aportan valor a la aplicación.

5.2.1.- Evolución técnica y estética

- **Revisión de estilos:** Ajustar y alinear todos los estilos de la aplicación para hacerlos más uniformes, teniendo en cuenta la parte pública y la privada. Mejorar la página de presentación para hacerla más atractiva y visual.
- **Funcionalidades de Angular:** Al tratarse de un *Framework* en continua evolución, **Angular** implementa nuevas funcionalidades que mejoran el rendimiento y la

experiencia de desarrollo, por lo que hacer uso de estas nuevas herramientas es una buena línea de trabajo para seguir aprendiendo.
- **Mejoras y adopción de nuevas animaciones:** Implementar nuevas transiciones que aparecen con la evolución de **CSS** y los navegadores modernos.
- **Mejoras de accesibilidad:** Implementar las etiquetas, atributos y las distintas herramientas ofrecidas de accesibilidad para que la aplicación pueda ser utilizada por cualquier tipo de usuario.

5.2.2.- Nuevas funcionalidades de gestión

- **Mapas de localización de colonias:** Implementar el uso de mapas como **Google Maps** o **Leaflet** para la visualización de las colonias.
- **Gráficas de métricas de adopciones y seguimiento:** Implementar el uso de gráficos con **ECharts** que muestren a modo resumen las adopciones anuales, el seguimiento sanitario de las adopciones, evolución de animales en proceso de adopción y adoptados durante los años, etc.
- **Mejoras en las notificaciones:** Evolucionar el sistema de alertas a notificaciones por correo electrónico y **notificaciones push** para recibir alertas a tiempo real y sin necesidad de acceder a la plataforma.

5.2.3.- Digitalización de integraciones

- **Pasarela de pago:** Evaluar la viabilidad de implementar la pasarela de pago con **Bizum** para evitar el flujo actual de **WhatsApp**.
- **Gestión de documentos:** Implementar un visor y editor de documentos en **PDF** para agilizar las adopciones y firma de contratos.

5.2.4.- Nuevas funcionalidades de promoción y visibilidad

- **Calendario de eventos de participación:** Añadir una página con un calendario o cronograma para mostrar los eventos públicos en los que la asociación interviene.
- **Blog con novedades:** Añadir página a modo de blog donde se mostrarán las entradas con información relevante relacionada con la adopción de animales.

5.2.5.- Iteraciones técnicas

Una posible iteración en cuanto a la estructuración del proyecto para prepararse a un futuro donde se puede obtener beneficio económico es modularizar el sistema y extraer las distintas funcionalidades en librerías independientes. De esta manera, se podría vender el servicio por paquetes para distintos clientes.

Para este desarrollo se utilizaría la librería **NX** [20]. Esta herramienta permite gestionar múltiples proyectos en un mismo repositorio, además de poder crear librerías con código compartido entre las aplicaciones, como pueden ser páginas visuales, modelos de datos o funciones genéricas.

Por otra parte, **NX** utiliza la misma carpeta de paquetes y dependencias instalados para todos los proyectos, simplificando enormemente la creación de nuevas aplicaciones, el mantenimiento y corrección de errores entre las aplicaciones ya creadas.

Teniendo en cuenta toda esta información, para cada nuevo cliente, tan solo habría que generar una aplicación dentro del repositorio y configurar sus credenciales personalizadas. Todo lo demás, podrá ser reutilizable y añadir únicamente los módulos que se requieran.

Bibliografía

[1] TypeScript
www.typescriptlang.org/docs/
Fecha de consulta 21 12 2025

[2] Guía de desarrollo de Angular
https://angular.dev/style-guide
Fecha de consulta 18 12 2025

[3] Documentación oficial de Cloud Functions
https://firebase.google.com/docs/functions/callable?hl=es-419
Fecha de consulta 8 01 2026

[4] Documentación oficial de Scheduled Functions
https://firebase.google.com/docs/functions/schedule-functions?hl=es-419
Fecha de consulta 14 01 2026

[5] Documentación oficial de Cloud Firestore
https://firebase.google.com/docs/firestore
Fecha de consulta 8 01 2026

[6] Patrones de arquitectura Angular
https://medium.com/front-end-world/mvc-and-mvvm-patterns-in-angular-7397e0bc7b07
Fecha de consulta 4 12 2025

[7] Arquitectura en Angular
https://dev.to/vanessamarely/arquitectura-en-angular-5c23
Fecha de consulta 4 12 2025

[8] Principios SOLID
https://www.freecodecamp.org/espanol/news/los-principios-solid-explicados-en-espanol/
Fecha de consulta 21 12 2025

[9] Documentación oficial de Firebase Authentication
Fecha de consulta 8 01 2026
https://firebase.google.com/docs/auth

[10] Documentación oficial de AngularFire
https://github.com/angular/angularfire
Fecha de consulta 8 01 2026

[11] IDE WebStorm
https://www.jetbrains.com/es-es/webstorm/features/
Fecha de consulta 20 12 2025

[12] Git y Gitflow
https://www.atlassian.com/es/git/tutorials/comparing-workflows/gitflow-workflow
Fecha de consulta 27 12 2025

[13] Documentación oficial de Lighthouse
https://developer.chrome.com/docs/lighthouse/overview?hl=es-419
Fecha de consulta 20 01 2026

[14] Documentación oficial de Tailwind
https://tailwindcss.com/docs
Fecha de consulta 16 01 2026

[15] Documentación oficial de NG-Zorro
https://ng.ant.design/docs/introduce/en
Fecha de consulta 16 12 2025

[16] Documentación oficial de Transloco
https://jsverse.gitbook.io/transloco
Fecha de consulta 16 01 2026

[17] Documentación oficial de seguridad y reglas de Firebase
https://firebase.google.com/docs/rules
Fecha de consulta 14 01 2026

[18] Librería edición de PDFs pdf-lib
https://pdf-lib.js.org/
Fecha de consulta 18 01 2026

[19] Documentación oficial de Google Drive API
https://developers.google.com/drive/api/guides/about-sdk
Fecha de consulta 10 01 2026

[20] Herramienta monorepos NX
https://nx.dev/
Fecha de consulta 27 01 2026