Escuela Politécnica Superior

Dogizzy

App destinada a las relaciones entre perros

Grado en Ingeniería Multimedia

Trabajo Fin de Grado

Autor:
Sergio Espinosa Zaragoza

Tutora:
Estela Saquete Boro

Universitat d'Alacant
Universidad de Alicante

Resumen

Las relaciones personales cada vez se están modernizando más y hoy en día podemos llegar a sentir un fuerte vínculo con personas que hemos conocido online, algo que hace unos años sería muy difícil de conseguir por la falta de medios.

Un tema relacionado que todavía está en medio de la modernización es las relaciones entre mascotas. Es obvio decir que las propias mascotas no pueden usar un teléfono inteligente para conocer a otras de manera online, pero los dueños de estas sí podrían, y es algo que actualmente está poco explorado.

Todas las aplicaciones relacionadas con las mascotas suelen estar enfocadas a su entrenamiento y a controlar sus dietas, actitudes, etc. De igual forma, están muy desactualizadas y descuidadas, y no suelen tener buenas respuestas de los usuarios que las utilizan, fomentando así un desinterés sobre las aplicaciones de este sector al tener malas experiencias constantes.

Por lo tanto, el objetivo principal que plantea este proyecto es poder juntar las bases de una aplicación de chat en línea con perfiles que busquen relacionar a sus mascotas de manera personal, pero con las facilidades y mejoras de una aplicación bien estructurada y fluida que mejore la experiencia de usuario.

Para poder ayudar a que estos usuarios interactúen entre ellos, se ha planteado un algoritmo base que relacione los perfiles entre sí y se los sugiera al usuario dependiendo de la compatibilidad de sus intereses. Desde estos perfiles sugeridos será donde se pueda iniciar una conversación en línea entre los dueños de las mascotas.

Palabras clave: aplicación móvil, mascotas caninas, chat online, relaciones entre perros

Motivación, justificación y objetivo general

La idea de este TFG nació después de la asignación del tema general que publicó mi tutora en el listado de temas para este año. Una vez asignado el tema de aplicación para mascotas, tuve bastante claro desde el principio lo que quería hacer.

Bastantes conocidos míos tienen una mascota, la mayoría perros, y a principio de curso daba la casualidad de que varios buscaban una pareja para sus perros y poder criar así a sus mascotas, aunque sea una vez antes de castrarlos por los problemas que el celo produce, sobre todo a las perros hembras.

Aun viviendo en grandes ciudades, siempre les ha costado relacionar a sus perros con otros, y mucho más criarlos, y aprovechando que mi tema asignado era sobre mascotas, la idea que un día surgió como una broma de hacer una aplicación que consiga que sus mascotas puedan criar se hizo realidad.

Por otra parte, quería explorar el mundo del desarrollo móvil ya que en el grado de Ingeniería Multimedia se da muy por encima en ciertas asignaturas, y específicamente me interesaba aprender cómo desarrollar una aplicación nativa más que usando alguna interfaz web que permita trasladarlo a móvil.

De esta manera, con todo el apoyo de mis conocidos al ser una aplicación que les beneficiaria enormemente y con los consejos de algunos compañeros en el tema de desarrollar una aplicación de teléfono, decidí desarrollar este proyecto de forma nativa para *Android,* al ser una experiencia totalmente nueva que me motiva completamente porque de cara al futuro, es un sector en el que podría verme trabajando después de graduarme como ingeniero.

Agradecimientos

Primeramente, agradecer a todo el mundo que ha compartido estos años conmigo en esta larga aventura de cinco años que ha durado mi paso por el grado de Ingeniería Multimedia.

Por una parte, a mis padres y familia, que siempre me han apoyado en todo lo que hago y han confiado plenamente en mí, aunque me haya tomado un año más de universidad para acabar el grado. Y en especial a mi hermana Isabel, que siempre ha estado ahí para ayudarme en los momentos más agobiantes del proyecto y me ha sabido guiar en todos los aspectos en los que me sentía perdido.

Por otra parte, a todos los compañeros de carrera y amigos que he conocido y me han hecho crecer como persona y como ingeniero. También al profesorado de la carrera, que siempre ha estado ahí para ayudarnos incluso después de años sin haber dado clase de manera conjunta, y a la Universidad de Alicante por haberme dado esta oportunidad de explorarme a mí mismo y llegar a ser la persona que soy hoy en día.

Por último, agradecer a mi amigo y compañero Izadi, que desde el día que lo conocí me ha estado apoyando y ayudando en todo lo que hago, independientemente de si era un problema personal o académico. Incluso a miles de kilómetros de distancia, siempre ha sabido sacar lo mejor de mí y he conseguido superar todos los retos que me he propuesto gracias a él.

Nuestros compañeros perfectos nunca tienen menos de cuatro patas.

Colette

Si no tienes paciencia, no puedes hacer cerveza.

Proverbio ovambo

Índice de contenidos

Índice de ilustraciones

Índice de tablas

Listado de abreviaturas

ADM	Administrador
ANR	Application Not Responding
API	Application Programming Interfaces (Interfaz de programación de aplicaciones)
ASO	App Store Optimization
CAME	Corregir, Afrontar, Mantener y Explotar
CV	Curriculum Vitae
DAFO	Debilidades, Amenazas, Fortalezas y Oportunidades
FC	Forced Closure
FCM	Firebase Cloud Messaging
HTML	HyperText Markup Language
i.e.	is est (por ejemplo)
ID	Identification
IDE	Integrated Development Environment
KPI	Key Performance Indicator (Identificador Clave de Actuación)
RF	Requisito funcional
RNF	Requisito no funcional
SDK	Software Development Kit (Kit de desarrollo sofware)
SMART	Specific, Measurable, Assignable, Realistic Time-Related
SYS	Requisito del sistema
TFG	Trabajo de Fin de Grado
UID	User identification
USR	Usuario
XML	Extensible Markup Language

1 Introducción

En la actualidad, la manera en la que las personas interactuamos ha cambiado mucho respecto a como ocurría hace escasos diez años, incluso alguno menos. Tenemos toda nuestra vida en nuestros bolsillos, y desde entonces somos incapaces de imaginarnos una vida sin nuestros dispositivos móviles.

Nuestras relaciones interpersonales han cambiado a ser algo mucho más online por comodidad e innovación, lo que no quita que sigan siendo relaciones válidas como cualquier otra, pero quizá sean más difíciles de comprender.

Por otro lado, si reflexionamos sobre cómo se relacionan nuestras mascotas, sigue siendo una realidad muy convencional y dependiente de situaciones externas, como pasear, ir a parques o conocer a alguien que resulte que tenga mascota.

Por lo tanto, aprovechando la cualidad de poder interactuar con todo el mundo a través de un dispositivo, es conveniente expandir horizontes y crear nuevas formas de interacción social que sean beneficiosas aprovechando sectores que todavía siguen sin estar modernizados totalmente como podría ser el sector de las mascotas.

Teniendo en cuenta que las redes sociales y la interacción online entre usuarios están en auge, es interesante ver como interactuarían las personas cuando el centro de atención no es la misma persona, sino su mascota.

La creación de una aplicación que se centre en una idea es lo suficientemente complicado por sí sola como para intentar que abarque varias, por lo tanto, es importante centrar las ideas fundamentales que desarrollará la aplicación para conseguir que se cumpla el objetivo principal de este TFG: interacciones online centradas en las mascotas.

Aquí es donde surge este proyecto, *Dogizzy*, una aplicación que mediante un algoritmo de comparación de intereses facilitará la experiencia de usuario para relacionarse con otros dueños de mascotas. Específicamente, la aplicación se centrará en los dueños de perros.

El eje central e innovador de la aplicación será el algoritmo mencionado anteriormente, pero tampoco nos podemos olvidar de otros dos factores importantes para la aplicación: el chat online y los perfiles de usuario.

Para los usuarios es importante mostrarse al público de una manera visualmente atractiva y que incite las interactuaciones entre los usuarios, y qué mejor manera de interactuar que directamente enviando mensajes.

Con estos tres ejes en mente, se propone crear una aplicación que ayude a los usuarios a relacionarse entre ellos y así facilitar a que sus mascotas se relacionen entre ellas, pudiendo concretar citas online con objetivos claros (como, por ejemplo, salir a pasear, ir a un parque canino o tener crías). Es una aplicación que ayuda a los usuarios a compaginar el día a día con las necesidades sociales de nuestra mascota ya que simplifica el proceso de conocer otros dueños que tengan mascotas compatibles.

Dogizzy es una aplicación que, tras hablar con varios conocidos míos que tienen dificultad a la hora de relacionar a sus mascotas, puede ser muy útil a la hora de encontrar amigos y criar perros, ya que, en lugares como pueblos o ciudades pequeñas, es difícil coincidir con otros dueños de manera natural y que, además, sean de la misma raza si tu objetivo es tener crías, por lo que creo que es una aplicación que podría tener bastante futuro.

2 Estudio de viabilidad

Antes de empezar a desarrollar el proyecto, se va a analizar un poco el proyecto en sí mismo realizando un estudio de viabilidad que nos ayudará a tomar las decisiones necesarias una vez sabiendo si los objetivos del proyecto son viables, pertinentes y necesarios.

A su vez, es conveniente realizar un análisis de riesgos con el objetivo de poder identificar, analizar y crear un plan de respuesta para cualquier riesgo que pudiese surgir durante el ciclo de desarrollo.

2.1 Análisis DAFO

El DAFO (de las iniciales de Debilidades, Amenazas, Fortalezas y Oportunidades) es una metodología de estudio de la situación de un proyecto, analizando sus características internas (i.e. Debilidades y Fortalezas) y su situación externa (i.e. Amenazas y Oportunidades) en una matriz cuadrada como muestra la *Ilustración 1*.

Ilustración 1. Esquema de un análisis DAFO.

(Martinez, 2015)

En cuanto al proyecto *Dogizzy* los cuadrantes serían los siguientes:

- **Debilidades:**
  - Falta de experiencia en proyectos grandes.
  - Uso de nuevas tecnologías.
  - Tiempo limitado de 300 horas para desarrollar el proyecto.
  - Incertidumbre en el tamaño del proyecto, y cuanto se podrá realizar como parte del TFG.
  - Desarrollo poco guiado.
  - Falta de experiencia en organización y distribución de tiempo en proyectos.
  - Presupuesto monetario nulo para el marketing una vez el proyecto esté lo suficientemente desarrollado.

- **Amenazas**

- Varias aplicaciones de relaciones entre perros en el mercado.
- Desarrollar una aplicación que no sea lo suficientemente completa y robusta.
- Circunstancias personales que podrían afectar al tiempo dedicado en el proyecto.

- **Fortalezas**
  - Facilidad de aprender nuevas tecnologías.
  - Experiencia mínima en desarrollo para *Android.*
  - Interés personal en el desarrollo del proyecto.
  - Capacidad de hacer frente a las adversidades.
  - Determinación en cumplir los objetivos propuestos.
- **Oportunidades**
  - Aprender tecnologías recientes para el desarrollo de aplicaciones.
  - Darme a conocer.
  - Llegar a mucha gente.
  - Poder demostrar mi capacidad de gestionar un proyecto.
  - Proyecto de grandes rasgos para añadir a mi CV.

Como conclusión, y teniendo en cuenta la naturaleza del *TFG* de ser un proyecto nuevo que llega al mercado, deberíamos optar por una estrategia ofensiva siguiendo el análisis CAME[1].

Con el análisis de riesgos que realizamos en el punto 2.3, crearemos medidas de contención que nos ayudarán a corregir y afrontar las debilidades y amenazas expuestas de manera efectiva, potenciando así al máximo las fortalezas y las oportunidades.

2.2 Lean Canvas

El *Lean Canvas* es otra de las herramientas que podemos utilizar para analizar un producto que está siendo creado y consta de carácter innovador. La idea del *Lean Canvas* es analizar el proyecto desde diferentes perspectivas de interés, no solo desde el desarrollo y los costes.

[1] *CAME*: "Herramienta de negocio que funciona como complemento al análisis DAFO que brinda información esencial para analizar estrategias y se puedan tomar decisiones correctas para el modelo de negocio" (Galiana, 2021)

El *Lean Canvas* del proyecto *Dogizzy* sería el siguiente que se muestra a continuación en la Ilustración 2:

Ilustración 2. Cuadro de Lean Canvas de Dogizzy

(Fuente propia)

De esta forma, podemos ver una idea más clara de lo aspectos que el proyecto puede ofrecer, lo que soluciona y como lo hará de una manera visual y fácil de ver. Igualmente, es conveniente analizar en detalle cada uno de los aspectos expuestos en el *Lean Canvas* para profundizar la idea del proyecto.

2.2.1 Segmento de Clientes

En este apartado del *Lean Canvas* se ha dividido a los clientes en cuatro grupos principales. Teniendo en cuenta los objetivos que estos tenían al descargarse la aplicación: (1) adoptar perros, (2) buscar una pareja para la mascota y poder criar, (3) conocer dueños de otras mascotas y, por último, (4) compañía para la mascota.

Podrá haber más usuarios con diferentes intenciones que utilicen *Dogizzy* pero la gran mayoría tendrá como interés alguno de esas cuatro objetivos, estimando que los usuarios que busquen compañía para sus mascota, o directamente pareja para esta sean los usuarios predominantes en la aplicación.

Unos posibles arquetipos de usuarios de *Dogizzy* podrían ser los siguientes:

- Daniel, opositor de 27 años, se dedica a prepararse para los exámenes de su oposición a la vez que cuida de su mascota, que se trata de un perro de raza Shiba Inu. Daniel en su tiempo libre pasea y cuida a su mascota, y le gustaría encontrar mascotas que quisieran jugar con su perro. Tampoco descarta la idea de encontrar una pareja para su perro y poder criar.

- Anastasia, enfermera de 23 años, tiene de mascota un Caniche de pelo corto. Anastasia tiene largos turnos de trabajo, por lo que cuando ella no puede sus padres cuidan de su mascota. Los padres de Anastasia presentan a su mascota a concursos de belleza, de los cuales ha ganado varios de los que se ha presentado. Por este motivo, los padres de Anastasia la han impulsado a que busque una pareja para su mascota que tenga pedigrí para poder seguir presentando a las siguientes generaciones en concursos.

2.2.2 Problemas

Las partes de la problemática del proyecto se pueden resumir en la falta de recursos a la hora de conocer a otros dueños de mascotas si se hace de la manera habitual (conocerlos por la calle cuando paseen a su mascota o en algún parque para perros). Además, esta manera de conocer a otros dueños es bastante ineficiente si buscas algo más concreto para tu mascota, como es criar una raza de perro en específico para criar, y te impide conocer a otras mascotas que no vivan en tu ciudad.

2.2.3 Propuesta única de valor

Crear una aplicación orientada hacia las relaciones entre perros que facilite las interacciones entre los dueños que tengan los mismos objetivos para sus mascotas y consigan crear vínculos de valor que beneficien a ambos usuarios.

2.2.4 Solución

Buscando una solución a los problemas comentados en el apartado 2.2.2 se ha llegado a la siguiente solución: una interfaz amigable y usable que permita la comunicación por chat

entre los dueños, alcanzando así relaciones entre usuarios de distintas ciudades, además del uso de filtros de usuarios registrados en la aplicación que facilita la posibilidad de encontrar a un dueño con los mismos intereses y objetivos que tú. Gracias a esta solución obtenemos nuestra Ventaja diferencial como se expone en el punto 2.2.9.

2.2.5 Canales

Al ser un proyecto que nace junto a un TFG no se dispone de ningún presupuesto monetario de marketing, por lo que los canales de distribución se limitarán a aquellos que se puedan tener sin tener que invertir dinero. Estos canales son, principalmente las redes sociales (*Instagram, Facebook, Twitter*, entre otros), el boca a boca y tiendas como la *Play Store*, en la cual será importante hacer una buena optimización del ASO (*App Store Optimization*) para poder posicionar bien a *Dogizzy* para que la gente pueda encontrarlo.

2.2.6 Flujos de ingreso

La descarga de la aplicación será totalmente gratuita y en principio no constará de compras dentro de la aplicación ni de publicidad. Esto se debe a que la monetización no forma parte de los objetivos del TFG. Sin embargo, no se descarta que en un futuro si se continua el proyecto se añadan estas características.

2.2.7 Estructura de costes

Teniendo en cuenta que la app funciona totalmente en el dispositivo del usuario, no accede a ninguna *API* o servicio propio, y los servicios externos que se van a utilizar son totalmente gratuitos, el coste se limitará a los recursos humanos, que se reduce a mí. De todas maneras, no se descarta que en un futuro se puede hacer uso de alguna *API* de terceros con coste, así como desarrollar uno propio, lo que supondría añadir el coste del *hosting*, del dominio y de los demás gastos que acarrearía.

2.2.8 Métricas clave

Las métricas claves estarán en constante evolución dependiendo del desarrollo del proyecto, así como las necesidades que se vayan detectando. De todas formas, unas métricas iniciales para el inicio del proyecto podrían ser las siguientes:

- Instalaciones totales e instalaciones activas.
- Retención del usuario.
- Valoraciones de las tiendas de aplicaciones como la *Google Play Store*.
- Número y frecuencia de cierres inesperados y ANRs. (*Application Not Responding*)

2.2.9 Ventaja especial

Finalmente, la ventaja especial que nos diferenciaría de otras aplicaciones actualmente en el mercado sería la facilidad de encontrar a otro usuario con los mismos intereses que tú y poder interactuar con dicho usuario de una manera cómoda, totalmente online y sin ningún compromiso, y a partir de ahí decidir si te interesa establecer conexiones en persona para conocer a las mascotas.

2.3 Análisis de riesgos

A lo largo de un proyecto de estas características, suele ser adecuado que antes de iniciar nada se tenga en consideración los riesgos que puedan suceder en el transcurso del desarrollo. Para ello, es conveniente crear un análisis de riesgos para poder prevenir lo que pueda pasar y así poder interactuar de manera efectiva en cada caso, es lo ideal. Por lo tanto, se ha realizado una tabla listando los posibles riesgos junto a un plan de contingencia en cada caso que se encuentra a continuación:

Posible Riesgo	Probabilidad	Efectos	Plan de Contingencia
Tecnologías y herramientas			
Desconocimiento de las tecnologías a usar	Muy alta	Soportable	Empezar a estudiar las tecnologías antes de empezar la fase de desarrollo.
Perdida de datos del servidor	Baja	Soportable	Tener copias de seguridad del servidor para poder restaurar los datos y poner todo en funcionamiento lo más pronto posibles

Corrupción de la última versión de la copia de seguridad	Baja	Soportable	Volver a la última versión disponible y estable.
Corrupción de todas las copias de seguridad	Muy baja	Muy costoso	Reinstalar la última versión del código y volver a empezar a introducir los datos. En caso de falta de tiempo habría que introducir los datos mínimos para poder mostrar el mínimo producto viable.
Caída de servicios externos de los que dependa la aplicación	Muy baja	Costoso	Desactivar la funcionalidad afectada e informar a los usuarios de lo que está sucediendo. Si el problema se alarga o persiste, buscar servicios alternativos para restablecer dicha funcionalidad.
En el proceso de desarrollo, romper una funcionalidad mientras se implementa otra	Muy alta	Costoso	Usar un sistema de control de versiones como Git, donde poder ir a cualquier versión anterior donde ese problema no era presente, y poder probar que diferentes funciones funcionan bien entre sí antes de pasar el código a desarrollo.
Personales			
Posible Riesgo	**Probabilidad**	**Efectos**	**Plan de Contingencia**
Enfermedad o lesión leve	Media	Soportable	Realizar tareas más llevaderas y aplazar las más costosas para cuando este recuperado.

Enfermedad o lesión grave	Baja	Muy costoso	Recortar tareas o incluso funcionalidades de la aplicación hasta poder adaptarse a la situación en la que me encuentre.
Desmotivación	Media	Soportable	Apoyarme en familiares y amigos para hablar con ellos y despejarme la mente.
Trabajos externos	Muy alta	Costoso	Organizarme las tareas y el tiempo lo mejor posible para compaginar el trabajo con el proyecto. En caso de que no se pueda realizar, recortar tareas e incluso funcionalidades para poder sacar un mínimo producto viable.
Organizacionales y de estimaciones			
Posible Riesgo	**Probabilidad**	**Efectos**	**Plan de Contingencia**
Duda sobre algún apartado de la memoria	Muy alta	Soportable	Seguir las guías de TFG proporcionadas por mi tutora. En el caso de no llegar a ninguna conclusión, preguntarle al tutor al respecto.
Atasco en alguna parte del desarrollo de la aplicación	Alta	Costoso	En el caso de que no lo consiga solucionar por mi cuenta, preguntar a algún profesor o compañero.
Sobreestimación de tareas	Baja	Soportable	En caso de sobreestimar las horas, dedicar las horas restantes a otras tareas

Infraestimación de tareas	Alta	Soportable	Dividir las tareas y aplazar lo necesario para seguir un ritmo constante.
Requerimientos			
Posible Riesgo	**Probabilidad**	**Efectos**	**Plan de Contingencia**
Cambio de los requerimientos básicos	Baja	Costoso	Cambio en la estructuración parcial o total de la aplicación, con los cambios/recortes en funcionalidades que esto pueda ocasionar.
Cambio de los requerimientos avanzados	Media	Costoso	Modificar la documentación y añadir las tareas necesarias. Comprobar y modificar los requerimientos que se verían afectados

Tabla 1. Análisis de riesgos

3 Planificación

Para poder llevar a cabo el proyecto y la memoria bajo el tiempo previsto, es necesario realizar una pequeña planificación de las diferentes secciones de trabajo para así ponernos unas fechas límites de cuando cada tarea tiene que estar finalizada.

Procedemos a dividir el trabajo en secciones como indica la Tabla 2.

Contenidos	Tiempo total	Fecha límite
- Motivación, justificación y objetivo general	2 meses	1 de febrero

- Planificación - Objetivos - Estudio de la viabilidad - Introducción		
- Metodología - Análisis y especificación - Estado del arte	1 mes y 1 semana	9 de marzo
- Diseño - Implementación	3 meses y 3 semanas	29 de junio
- Pruebas y validación - Resultados - Resumen - Índices - Citas - Agradecimientos - Conclusiones y trabajo futuro - Referencias bibliografía y apéndices	2 semanas	13 de julio

Tabla 2. Planificación

Este curso solo he tenido que cursar dos asignaturas y el TFG, por lo que puedo dedicarme a ello a lo largo del segundo semestre y terminarlo a tiempo para la convocatoria de julio, a la que he decidido ir al tener también que compaginarme el TFG con las prácticas de empresa y otro trabajo que tengo.

Para lograr esto, he decidido planificar el proyecto, los objetivos y su viabilidad antes de desarrollar la aplicación, y dedicarle la gran mayoría del tiempo al diseño del proyecto y al propio desarrollo, finalizando los últimos toques de la memoria en las últimas semanas.

He dividido el trabajo en meses y semanas al tener como control reuniones más o menos mensuales con mi tutora del TFG, en las cuales me ponía objetivos que completar para cada reunión. De esta forma, siempre controlo el progreso que llevo y cuento con retroalimentación de mi tutora.

4 Estado del arte

A la hora de proceder al desarrollo de cualquier proyecto o resolver cualquier problema es necesario analizar la situación actual del mismo, las soluciones ya existentes o tendencias del mercado.

Por ello, a continuación, se detallarán varias secciones donde se analizan la situación actual de la problemática, antecedentes, así como las tecnologías y fuentes de información que se podrían usar para el desarrollo.

4.1 Análisis del mercado

En el 2021, se estima que haya 461 millones de perros como mascota, lo que supone que un 33% de los hogares a nivel mundial cuidan al menos de un perro (PFMA, 2021). Estos datos siguen en auge desde la aparición de la COVID-19, a la vez que crece el número de consumidores de aplicaciones para sus mascotas.

Actualmente (19/07/2022), las *apps* más populares destinadas para perros en base al número de descargas en la tienda de aplicaciones de *Android* son las que se encuentran plasmadas en la Tabla 3.

Nombre	Descargas	Valoración	Características
Dogo[2]	1M+	4.7☆	-Aplicación enfocada en el adiestramiento de perros. -Cuenta con lecciones de texto, imágenes y video, además de un mural donde publicar imágenes de tu mascota con "me gustas" y comentarios.

[2] https://play.google.com/store/apps/details?id=app.dogo.com.dogo_android [Último acceso 19/07/2022]

Puppr[3]	100k+	4.3☆	-Aplicación enfocada en el adiestramiento de perros. -Parecida a Dogo, cuenta con las mismas funcionalidades, pero intercambia el mural por un chat.
Dog Walk[4]	100k+	4.0☆	-Aplicación para trackear los paseos de tu perro. -Utiliza el GPS para dibujar la ruta que has realizado con tu perro y lo guarda en un registro de paseos -Puedes compartir tus paseos realizados.
DogLog[5]	50k+	No consta	-Aplicación para monitorizar y tener un seguimiento de las actividades de tu perro -Opciones de seguimiento de alimentación, actividades del paseo, medicinas y vacunas entre otros.
MatchDog[6]	100k+	1.9☆	-Aplicación destinada a las relaciones entre perros.

[3] *https://play.google.com/store/apps/details?id=com.chinandcheeks.dogtrainer* [Último acceso 19/07/2022]
[4] *https://play.google.com/store/apps/details?id=com.tractive.android.walk* [Último acceso 19/07/2022]
[5] *https://play.google.com/store/apps/details?id=com.mobikode.dog* [Último acceso 19/07/2022]
[6] *https://play.google.com/store/apps/details?id=co.tech.matchdog* [Último acceso 19/07/2022]

			-Perfiles de perros con descripción de éstos. -Sistema de *likes* y *match* -Buscador de perfiles

Tabla 3. Aplicaciones similares del mercado

Como se puede observar, la mayoría de las aplicaciones no se parecen entre sí al diferenciarse en su funcionalidad principal, pero se puede inferir que las aplicaciones más populares son precisamente las que permiten adiestrar a tu mascota de forma más fácil, ya que puede llegar a ser una tarea bastante complicada.

Del mismo modo, también observamos otras aplicaciones destinadas al seguimiento de la salud de tu perro, de sus actividades y sus paseos, y por último una aplicación destinada a las citas entre perros, que es la aplicación que más se asemeja a *Dogizzy*.

4.2 Tecnologías para el desarrollo

Una de las primeras cosas que hay que tener en cuenta a la hora de empezar a desarrollar el proyecto es qué tecnología vamos a utilizar, ya que podemos tomar la vía natural de un desarrollo de aplicación nativo, o probar con una de las muchas herramientas que nos permiten crear aplicaciones multiplataforma.

La primera idea era desarrollar la aplicación con *Flutter* [7]aunque al final ha sido desarrollada en nativo, por lo que veo interesante exponer en este apartado las diferentes opciones que tuve en cuenta a la hora del desarrollo.

[7] *https://flutter.dev/* [Último acceso 10/07/2022]

4.2.1 Desarrollo nativo

El desarrollo nativo de aplicaciones permite conseguir el máximo provecho de la arquitectura de cada sistema operativo al tener disponible un acceso directo a todas las *API* que ofrece cada plataforma consiguiendo así una mejor experiencia de usuario.

La desventaja que existe es que necesitas desarrollar todo el código para cada plataforma independientemente. Hoy en día, aunque mucho se delegue en servicios web, seguimos encontrando mucha lógica de la aplicación directamente en la misma, por lo que supone bastante más trabajo, alargando así el proceso de desarrollo, haciéndolo más caro y necesitando aprender diferentes tecnologías propias de cada plataforma, como son: *Swift* [8]para *iOS* y *Kotlin*[9] (o *Java*) para *Android*.

4.2.2 Jetpack Compose

Jetpack Compose[10] es un kit de herramientas moderno (*SDK*) para compilar interfaces nativas y es la implementación del sistema *Compose* para *Android*. Consiste en un modelo de programación reactivo que utiliza *Kotlin.*

Una de las ventajas de usar *Jetpack Compose* es que es un *SDK* completamente declarativo y podemos describir toda la interfaz de usuario utilizando funciones predefinidas, además de que es completamente compatible con todas las vistas de *Android* existentes. Asimismo, al ser un *SDK* y no un *framework,* contamos con todas las ventajas de rendimiento y acceso a *APIs* del desarrollo nativo.

El aspecto más destacable es el uso único de *Kotlin* al desarrollar la aplicación, ya que anteriormente los desarrolladores de *Android* tenían que programar archivos *XML* a la vez que archivos *Java* (o *Kotlin* en los más recientes), lo que hace que la aplicación sea mucho más concisa y fácil de mantener. Actualmente, es utilizado en aplicaciones como *Google Play Store, Twitter* y *Pinterest*.

Por el contrario, la principal desventaja de *Jetpack Compose* es que es un SDK bastante reciente, por lo que todavía sigue en fase de desarrollo y muchas opciones cambian de una

[8] https://www.apple.com/es/swift/ [Último acceso 10/07/2022]
[9] https://kotlinlang.org/ [Último acceso 10/07/2022]
[10] https://developer.android.com/jetpack/compose [Último acceso 10/07/2022]

versión de *Compose* a otra, lo que puede hacer que tengas que cambiar tu código al tener funciones obsoletas. De igual modo, muchas funciones están en fase experimental y están sujetas a constantes cambios e incluso la completa eliminación.

4.2.3 React Native

React Native[11] se trata de la respuesta de *Facebook* (en la actualidad renombrado a *Meta*) para la creación multiplataforma de aplicaciones para *iOS* y *Android*. El objetivo es usar los mismos conceptos que se usan en *React* (bibliotecas de *JavaScript* para la creación de interfaces web) y trasladarlos a una aplicación móvil.

No es de extrañar que *React Native* se use en grandes proyectos como en las propias aplicaciones de *Facebook, Instagram, o Skype*, ya que, si tu aplicación web esta creada con *React* o si tu equipo de desarrollo ya conoce la tecnología, no es muy complicado trasladar la experiencia a una aplicación móvil.

Sin embargo, la desventaja que nos podemos encontrar suele ser que el rendimiento no es el más óptimo y nos podemos encontrar con una navegación poco suave, además de que es bastante más complicado aprender *React Native* que *React* sin conocimientos previos, y muchas veces se necesita acceder a *APIs* que no están disponibles desde las bibliotecas de *JavaScript,* por lo que necesitaríamos tener experiencia en desarrollo nativo para acceder a estas.

4.2.4 Ionic

Similar a *React Native*, *Ionic* [12]nos permite trasladar la experiencia del *framework* Angular de web a móvil. La diferencia entre estas dos propuestas es que *Ionic* simplemente muestra elementos *HTML* que emulan el funcionamiento de *widgets* nativos, mientras que en *React Native* estamos haciendo aplicaciones que se acercan a ser nativas.

Por un lado, la ventaja de usar estos elementos *HTML* es que compartimos una sola base de código para todas las plataformas compatibles (web y móvil), y si contamos con

[11] *https://reactnative.dev/* [Último acceso 10/07/2022]
[12] *https://ionicframework.com/* [Último acceso 10/07/2022]

conocimientos previos de programación web podemos acelerar el proceso de desarrollo con una curva de aprendizaje de la tecnología bastante asequible.

Por otro lado, la desventaja que nos encontramos de nuevo es el rendimiento, ya que al final *Ionic* abre una vista web (*WebView*) para ejecutarse, sumado a las limitaciones de funciones específicas de cada sistema operativo que no podemos obtener utilizando este *framework*.

4.2.5 NativeScript

NativeScript[13] es otra alternativa para el desarrollo de aplicaciones móviles mediante el uso de *Angular*, además de también ser compatible con el *framework Vue.js*[14]

Por una parte, la principal diferencia entre *Ionic* y *NativeScript* es que en este caso si se consigue un rendimiento parecido al de las aplicaciones nativas gracias al uso de componentes propios de los sistemas de despliegue de *Android* y *iOS*. Además, aunque la comunidad que utiliza *NativeScript* no sea tan grande como la de *React Native*, gracias a su compatibilidad con dos frameworks diferentes y su buena documentación lo convierten en una buena opción que considerar a la hora de desarrollar una aplicación móvil.

Por otra parte, la desventaja principal es el tener que seguir usando un motor de *JavaScript* para ejecutar la lógica de la aplicación *(JavaScriptCore* en *iOS* y *V8* en *Android*), por lo que no conseguimos el rendimiento y estabilidad de un desarrollo nativo.

4.2.6 Xamarin

A diferencia de las anteriores opciones, *Xamarin*[15] no se trata de un *framework* o *SDK* que permite usar tecnologías web en una aplicación móvil, sino que permite compartir la lógica detrás de las aplicaciones entre las tres plataformas más importantes (*iOS, Android y Windows UWP*), desarrollándolo en *C#* y *.NET*.

De esta manera, la mayoría del código es común para todas las plataformas y solo hay que implementar el código restante específico de cada uno, incluyendo las llamadas a *APIs*

[13] https://nativescript.org/ [Último acceso 10/07/2022]
[14] https://vuejs.org/ [Último acceso 10/07/2022]
[15] https://docs.microsoft.com/es-es/xamarin/get-started/what-is-xamarin [Último acceso 10/07/2022]

propias o alguna característica no común al resto. Además, utilizando el módulo de *Xamarin.Forms* para crear las interfaces de usuarios, podemos ampliar la cantidad de código compartido entre plataformas.

Los proyectos de Xamarin se compilan a nivel nativo de cada plataforma y utiliza widgets específicos de cada, consiguiendo así un rendimiento muy alto que se asemeja a una solución totalmente nativa. Como añadido, cuenta con el apoyo de *Microsoft*, por lo que se puede integrar con los *IDEs* de este mismo, como *Visual Studio* y otros de sus servicios los cuales son muy familiares para una gran cantidad de desarrolladores.

La desventaja de *Xamarin* es el propio de lenguaje de programación siendo *C#* y *.NET*, que supone una curva de aprendizaje para los que no estén familiarizados con los mismos, además de que, aunque el rendimiento sea casi nativo, aun se ejecuta sobre su propio entorno de ejecución de *Xamarin,* alargando los tiempos de carga y engordando el tamaño final de la aplicación.

4.2.7 Flutter

Flutter es la respuesta de *Google* a la creación de una única aplicación multiplataforma, en este caso para *iOS* y *Android*. Utiliza el lenguaje de programación *Dart* a la vez que un motor de renderización propio de alto rendimiento basado en *C++* y *widgets* propios, consiguiendo así un rendimiento similar al nativo.

Entre sus ventajas se incluye la función de recarga caliente (*hot reload* en *inglés*) lo cual permite ver el resultado del código en tiempo (casi) real, widgets ya creados que permiten agilizar todo el proceso de desarrollo y acceso a funciones nativas con opción incluso a reutilización de código ya nativo. Además, cuenta con el soporte oficial de *Google* asegurando su suporte, un equipo de desarrollo dedicado, fácil instalación y comienzo de desarrollo, etc.

Sin embargo, al ser un *framework* relativamente nuevo, no cuenta con demasiadas bibliotecas de terceros o aplicaciones de ejemplo. Aunque *Google* disponga de una holgada variedad de bibliotecas desarrolladas, el desarrollador puede que tenga que implementar por sí mismo muchas de las funciones que en otras plataformas podría estar utilizando ya creadas y mantenidas por otras personas.

5 Objetivos

En el siguiente apartado se exponen las metas que se han propuesto para este TFG. Para ello se ha utilizado el principio SMART[16], asegurándonos así que los objetivos propuestos aporten valor y nos ayuden a definir el proyecto.

El objetivo principal es la creación de una red social que permita la comunicación entre dos usuarios mediante un chat online con el fin de relacionar a sus mascotas entre ellas. Como subobjetivos u objetivos específicos se plantea:

- Ofrecer una interfaz de usuario fácil y usable que permita ver los perfiles de los demás usuarios de manera ágil y eficaz con el uso de filtros.

- Personalizar de manera total del perfil del usuario con el objetivo de facilitar la aparición de su perfil en la interfaz mencionada anteriormente, además de para customizar el perfil al gusto propio del usuario.

En cuanto al desarrollo de la aplicación, también se han marcado otros subobjetivos como metas a cumplir. El primero sería aprender de cero una nueva tecnología como es *Jetpack Compose* y demostrar en la aplicación todo lo que he aprendido durante el transcurso del TFG.

Siguiendo al subobjetivo anterior, otro subobjetivo más que me gustaría cumplir es el demostrar mi capacidad de organización y gestión del tiempo compaginando vida diaria, estudios y trabajo con el desarrollo de un proyecto a larga escala para así demostrar lo que soy capaz de hacer con todo lo aprendido en el grado de Ingeniería Multimedia.

[16] *Principio SMART:* "del inglés Specific, Measurable, Assignable, Realistic Time-Related, son características que un objetivo debería tener para cumplir este principio". (Doran, 1981).

Dicho esto, cabe también mencionar que el resultado final de la aplicación sería una demo por la cantidad de funcionalidades y optimizaciones que se tendrían que añadir para que fuese un producto viable en el mercado, al poder categorizar mi proyecto como una especie de red social. Por lo tanto, el último objetivo a cumplir sería continuar desarrollando el proyecto una vez acabado el periodo del TFG, para seguir aprendiendo desarrollo en *Android* a la vez que expando mis límites y capacidades en un proyecto propio que me ayudará en mi portfolio a que las empresas puedan verme como un buen candidato.

6 Metodología

La metodología que se seguirá para desarrollar este proyecto será una metodología ágil[17] basada en *Scrum*[18]. No será fielmente una metodología *Scrum* ya que no se van a aplicar todas las restricciones que se deberían, al ser un único miembro del equipo y al utilizar en varias partes otros tipos de metodologías como el desarrollo en cascada[19] o filosofías como *Lean*.[20]

Por lo tanto, se establecerán iteraciones o *sprints* que utilizarán tareas de la pila de tareas por hacer o *product backlog* para realizarlos durante dicha iteración. Para esto, se utilizará la herramienta online Trello, y se irán marcando también que tareas quedan pendientes, cuales están en marcha y cuales están terminadas. Adicionalmente, también hago uso de un bloc de notas para llevar constancia de lo que hacer cada semana de forma más cómoda, al ser los *sprints* normalmente de entre un mes o mes y medio.

[17] *Metodología ágil:* "las metodologías ágiles son aquellas que permiten adaptar la forma de trabajo a las condiciones del proyecto, consiguiendo flexibilidad e inmediatez en la respuesta para amoldar el proyecto y su desarrollo a las circunstancias específicas del entorno". (Garrido, 2021)
[18] *Scrum:* "metodología ágil basado en historias de usuarios, o cosas que los usuarios quieren tener". (ScrumGuides, s.f.)
[19] *Metodología de desarrollo en cascada*: "metodología de desarrollo lineal donde se realiza por completo una fase del proyecto antes de proceder a la siguiente". (Digital Guide IONOS, s.f.)
[20] *Lean*: "filosofía de desarrollo basado en el valor percibido del cliente donde se enfocan los procesos principales a que este valor vaya incrementando continuamente sin tener ningún residuo. La filosofía ágil nace a partir de esta filosofía" (RealtimeBlog, s.f.).

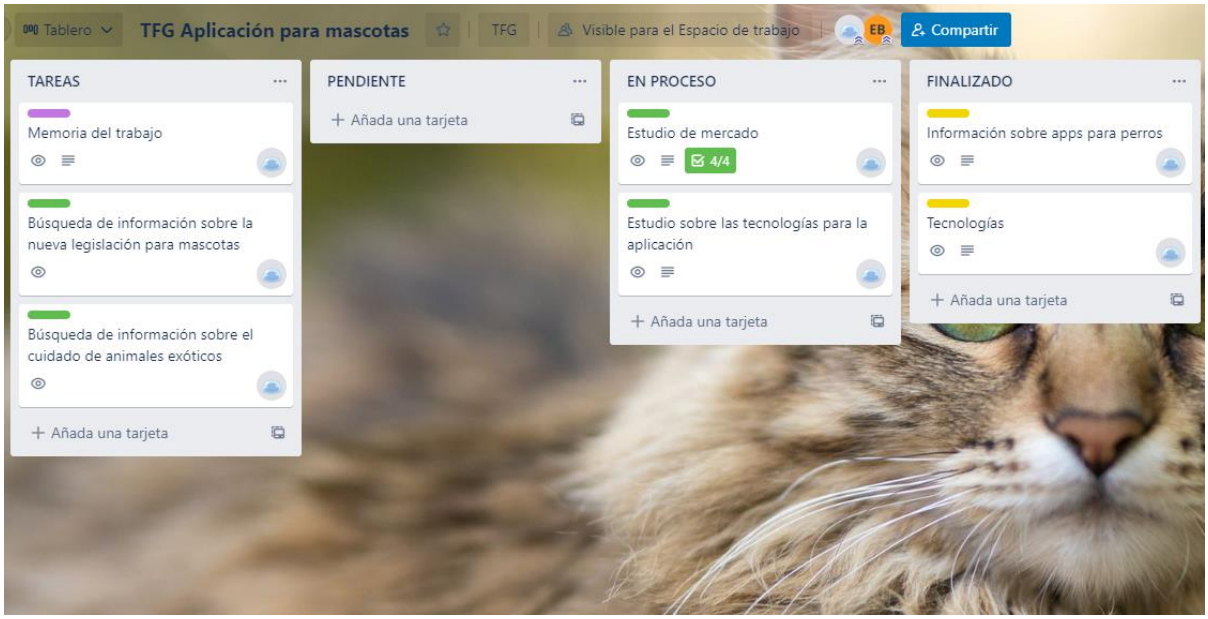


Ilustración 3. Imagen de trello en el primer sprint

(Fuente propia)

Al final de cada iteración —que será cada vez que haya reunión con la tutora— se analizará como ha ido cada *sprint*, los problemas que ha habido y cómo mejorar el proceso para ser más eficientes. Estos *sprints* serán definidos en la sección de Planificación de esta misma memoria, aunque se subdividirán en diferentes subtareas.

Entre las características de *Scrum* que no se seguirán, una de ellas es la separación de los tres roles que suele haber: (1) *Product Owner* (el cliente o quien tiene la visión del proyecto), (2) *Scrum Master* (el facilitador del proceso de Scrum) y (3) el equipo. Esto se debe a que es un proyecto individual y, por lo tanto, todos los roles recaen sobre mí. Además, no se crearán gráficas de tareas terminadas (*burndown*) para optimizar el tiempo.

Otras características que usaremos de otras metodologías y filosofías es el poder añadir tareas a un *sprint* ya en curso, de la filosofía *Lean.* Del mismo modo, siguiendo la metodología de desarrollo en cascada, se implementarán las características básicas que se requiere que el producto mínimo viable tenga, especificando primero los requisitos del sistema, seguido de una fase análisis y planificación y otra de diseño y especificación del sistema. De esta manera, nos ahorramos la incertidumbre que puede traer un proyecto ágil al ser un proyecto individual y tener que aprender una tecnología completamente desde cero.

7 Análisis y especificación

Para poder concretar nuestra solución del problema analizado hasta el momento es esencial analizar, definir y detallar los requerimientos y especificaciones que habrá que diseñar e implementar en las fases siguientes.

De esta manera quedará claro las funciones y restricciones que contará el proyecto y nos dará una mejor idea de cómo diseñar la solución adecuada.

Para la realización de esta parte de la memoria se seguirá, de una forma simplificada, el sistema IEE830; diseñado para asegurar que se recoge siguiendo buenas prácticas estos requerimientos.

7.1 Características de los usuarios

Tipo de usuario	Administrador
Descripción	Propietarios de la plataforma y de mayor nivel de acceso. Responsable de que el sistema funcione correctamente.
Actividades	Acceso y manejo de cuadros de mando que reportan el estado de la aplicación para supervisar que todo funciona como esperado.
Habilidades	Alto grado educativo y técnico, experto en el sistema desarrollado y ser capaz de entender los diferentes KPI[21] y saber analizar el estado del sistema en base a estos.

Tabla 4. Características del usuario Administrador

Tipo de usuario	Usuario
Descripción	Gran masa de usuarios que usan la plataforma.
Actividades	Usar la aplicación para interaccionar con otros perfiles y personalizar el suyo propio.

[21] *KPI*: "del inglés *Key Performance Indicator* o *Indicador Clave de Rendimiento* hace referencia a una serie de métricas que se utilizan para sintetizar la información sobre la eficacia y productividad de las acciones que se lleven a cabo en un negocio con el fin de poder tomar decisiones y determinar aquellas que han sido más efectivas a la hora de cumplir con los objetivos marcados en un proceso o proyecto concreto." (Porras, 2017)

Habilidades	Conocimientos informáticos básicos. Saber usar un teléfono móvil y estar familiarizados con las aplicaciones de estas.

Tabla 5. Características del usuarios

7.2 Requisitos

En esta sección se detallan todos los requerimientos del sistema propuesto. Se distinguen dos tipos de requisitos: los requisitos funcionales y los no funcionales.

Como todos los requisitos han de ser identificados con diferentes IDs, se seguirá el formato de "*tipo-usuario-número*". El *tipo* hará referencia a si es un requisito funcional *RF* o requisito no funcional *RNF*, el *usuario* al principal usuario para el cual se establece el requisito, *ADM* para administrador, *USR* para un requisito de usuarios finales y finalmente *SYS* si son inherentes al sistema como los requisitos no funcionales. Finalmente, el *número* permitirá enumerar todos los requisitos para una fácil identificación.

7.2.1 Requisitos funcionales

Son los requisitos que hacen referencia a funcionalidades del sistema. Definen un sistema o algún componente. En definitiva, son el conjunto de entrada, comportamiento y salida que el sistema tiene que realizar.

Identificador	RF-ADM-001
Requisito	Cuadros de mando
Descripción	Integrar un completo sistema de estadísticas y logs[22] para el análisis y monitorización del sistema.

Tabla 6. Requisito funcional de administrador 1

Identificador	RF-USR-001
Requisito	Registro
Descripción	Interfaz de registro del usuario para poder acceder a la aplicación

Tabla 7. Requisito funcional de usuario 1

Identificador	RF-USR-002
Requisito	Inicio de sesión
Descripción	Interfaz para que el usuario pueda acceder a la aplicación con su cuenta.

Tabla 8. Requisito funcional de usuario 2

Identificador	RF-USR-003

[22] *Logs:* "información de más o menos bajo nivel reportada por el sistema operativo o una aplicación concreta que sirve para identificar qué está haciendo, incluyendo errores, problemas o avisos menores, y cuando ha sucedido eso, indicando la fecha, hora y segundo". (Lerena, 2020)

Requisito	Pantalla principal de perfiles
Descripción	Interfaz a la que el usuario accederá una vez iniciado sesión y que mostrará los perfiles más compatibles para chatear al usuario.

Tabla 9. Requisito funcional de usuario 3

Identificador	RF-USR-04
Requisito	Lista de perfiles compatibles
Descripción	Lista de perfiles que se mostrará en la interfaz principal que constará de "cartas" que contendrán la foto de perfil del usuario, su nombre, su fecha de nacimiento y su lugar de residencia. Pulsando en estas cartas podremos acceder al perfil del usuario deseado.

Tabla 10. Requisito funcional de usuario 4

Identificador	RF-USR-005
Requisito	Algoritmo de compatibilidad de perfiles
Descripción	Algoritmo funcional que ayudará a que los perfiles más compatibles con el usuario se muestren antes en la interfaz principal de la aplicación.

Tabla 11. Requisito funcional de usuario 5

Identificador	RF-USR-006
Requisito	Perfil del usuario
Descripción	Interfaz que mostrará el perfil del propio usuario, que consistirá en una foto de perfil, nombre del usuario, fecha de nacimiento, lugar de residencia, una pequeña biografía, una lista de etiquetas sobre los intereses del usuario y un apartado donde publicar fotos. La interfaz de perfil de usuario permite la edición de estos atributos.

Tabla 12. Requisito funcional de usuario 6

Identificador	RF-USR-007
Requisito	Perfil de los demás usuarios
Descripción	Interfaz casi idéntica a la del perfil del propio usuario, salvo que intercambiando la posibilidad de editar con una opción de iniciar un chat online.

Tabla 13. Requisito funcional de usuario 7

Identificador	RF-USR-008
Requisito	Edición de la foto de perfil del usuario

Descripción	Sistema que permitirá editar la foto de perfil del usuario, realizando una foto en directo o eligiendo la foto desde la galería.

Tabla 14. Requisito funcional de usuario 8

Identificador	RF-USR-009
Requisito	Cámara de fotos
Descripción	Sistema que permitirá abrir la cámara de fotos del teléfono para sacar una foto.

Tabla 15. Requisito funcional de usuario 9

Identificador	RF-USR-010
Requisito	Galería de fotos
Descripción	Sistema que permitirá abrir la galería de fotos del teléfono para escoger una foto.

Tabla 16. Requisito funcional de usuario 10

Identificador	RF-USR-011
Requisito	Edición de los atributos principales del perfil del usuario
Descripción	Interfaz que permitirá la edición del nombre del usuario, la fecha de nacimiento de su mascota y su lugar de residencia.

Tabla 17. Requisito funcional de usuario 11

Identificador	RF-USR-012
Requisito	Edición de las etiquetas de intereses del usuario
Descripción	Interfaz que permitirá la edición de las etiquetas del usuario con un sistema parecido al "*drag & drop*[23]"

Tabla 18. Requisito funcional de usuario 12

Identificador	RF-USR-013
Requisito	Edición de las fotos del usuario
Descripción	Interfaz que permitirá editar la galería de fotos del usuario, de manera que podrá subir fotos, tomando una foto directamente con la cámara o escogiendo una de la galería del teléfono, o eliminarlas.

Tabla 19. Requisito funcional de usuario 13

Identificador	RF-USR-014
Requisito	*Pop-up*[24] de las fotos de galería del usuario

[23] Sistema *Drag & Drop*: "es una técnica que facilita la interacción del usuario con un programa o aplicación de manera intuitiva para llevar elementos de un lugar a otro". (NeoAttacks, s.f.)
[24] Ventana *Pop-Up*: "es un elemento o ventana emergente que se despliega de manera repentina sobre el contenido de una página web o aplicación móvil para mostrar un contenido determinado". (NeoAttacks, s,f,)

Descripción	Sistema que mostrará una ventana emergente con la foto seleccionada de la galería.

Tabla 20. Requisito funcional de usuario 14

Identificador	RF-USR-015
Requisito	Chat en línea
Descripción	Interfaz que permitirá comunicar a dos usuarios mediante un chat en línea.

Tabla 21. Requisito funcional de usuario 15

Identificador	RF-USR-016
Requisito	Lista de perfiles con un chat abierto
Descripción	Interfaz que permitirá al usuario ver los perfiles con los que ha tenido una conversación y seleccionar el chat para acceder a la interfaz de chat en línea.

Tabla 22. Requisito funcional de usuario 16

Identificador	RF-USR-017
Requisito	Buscador de chats
Descripción	Barra de búsqueda que permitirá buscar chats abiertos si buscamos por el nombre de la mascota del usuario.

Tabla 23. Requisito funcional de usuario 17

Identificador	RF-USR-018
Requisito	Navegación entre interfaces
Descripción	Sistema que permitirá una navegación entre interfaces con una transición suave.

Tabla 24. Requisito funcional de usuario 18

Identificador	RF-USR-019
Requisito	Barra de navegación inferior
Descripción	Barra de navegación inferior que nos permitirá navegar entre las tres interfaces principales: lista de chats, pantalla principal de perfiles y perfil del usuario.

Tabla 25. Requisito funcional de usuario 19

Identificador	RF-USR-020
Requisito	Barra de navegación inferior
Descripción	Barra de navegación inferior que nos permitirá navegar entre las tres interfaces principales: lista de chats, pantalla principal de perfiles y perfil del usuario.

Tabla 26. Requisito funcional de usuario 20

Identificador	RF-USR-021

Requisito	Interfaz de configuración
Descripción	Interfaz de configuración de la aplicación que nos permitirá configurar la aplicación a nuestro gusto, mostrará las licencias de las librerías y nos dará la posibilidad de cerrar la sesión del usuario.

Tabla 27. Requisito funcional de usuario 21

7.2.2 Requisitos no funcionales

Los requisitos no funcionales, también conocidos como "atributos de calidad" son los requisitos inherentes del sistema. Definen el comportamiento del producto y permiten imponer restricciones en el diseño del sistema para asegurar que se cumplen con ciertas características deseadas.

Identificador	RNF-SYS-001
Requisito	Diseño de tamaño adaptable
Descripción	La aplicación deberá estar adaptada para una gran cantidad de dispositivos con pantallas de tamaño y relación de aspecto diferentes.

Tabla 28. Requisito no funcional del sistema 1

Identificador	RNF-SYS-002
Requisito	Diferentes temas de la aplicación
Descripción	La aplicación contará con dos temas (claro y oscuro), para facilitar su uso en diferentes ambientes y siguiendo las preferencias de los usuarios.

Tabla 29. Requisito no funcional del sistema 2

Identificador	RNF-SYS-003
Requisito	Rendimiento
Descripción	La aplicación deberá funcionar de una forma fluida, a 60FPS el máximo tiempo posible, y con menos de un 0.5% de bloqueos de aplicación (*ANR*) y 1% de cierres inesperados (*FC*) siguiendo la recomendación de *Google* para los desarrolladores de *Android*.

Tabla 30. Requisito no funcional del sistema 3

Identificador	RNF-SYS-004
Requisito	Escalabilidad
Descripción	Minimizar el tamaño máximo de la aplicación para asegurar que se pueda descargar en un tiempo razonable bajo conexiones lentas.

Tabla 31. Requisito no funcional del sistema 4

Identificador	RNF-SYS-005
Requisito	Legalidad
Descripción	Todo material usado, tanto software como recursos, deberán tener una licencia válida para poder ser usados en un proyecto de esta índole. También se deberá asegurar el cumplimiento del Reglamento General de Protección de datos (*GDPR*) y facilitar a los usuarios sus derechos.

Tabla 32. Requisito no funcional del sistema 5

Identificador	RNF-SYS-006
Requisito	Mantenimiento
Descripción	El código tendrá que ser lo más modular posible para facilitar modificaciones sin implicar cambios en el resto, así como la expansión de futuras funcionalidades. Se deberá usar un sistema de control de versiones en todo momento para poder iterar y avanzar en el desarrollo del proyecto.

Tabla 33. Requisito no funcional del sistema 6

8 Diseño

En este apartado se aporta la solución a todos los requisitos vistos en el apartado anterior *Análisis y Especificación*, especificando como serán resueltos.

Dogizzy será una aplicación que residirá única y exclusivamente en los dispositivos móviles y que no requerirá ningún servidor propio. Todos los servicios externos utilizados actuarán para mantener la información de la aplicación actualizada.

8.1 Diseño de la arquitectura conceptual

Dogizzy no contará con *API* propia, por lo que toda la lógica de la aplicación estará centrada en si misma e interactuará con diferentes elementos externos, como se muestra en la siguiente ilustración:

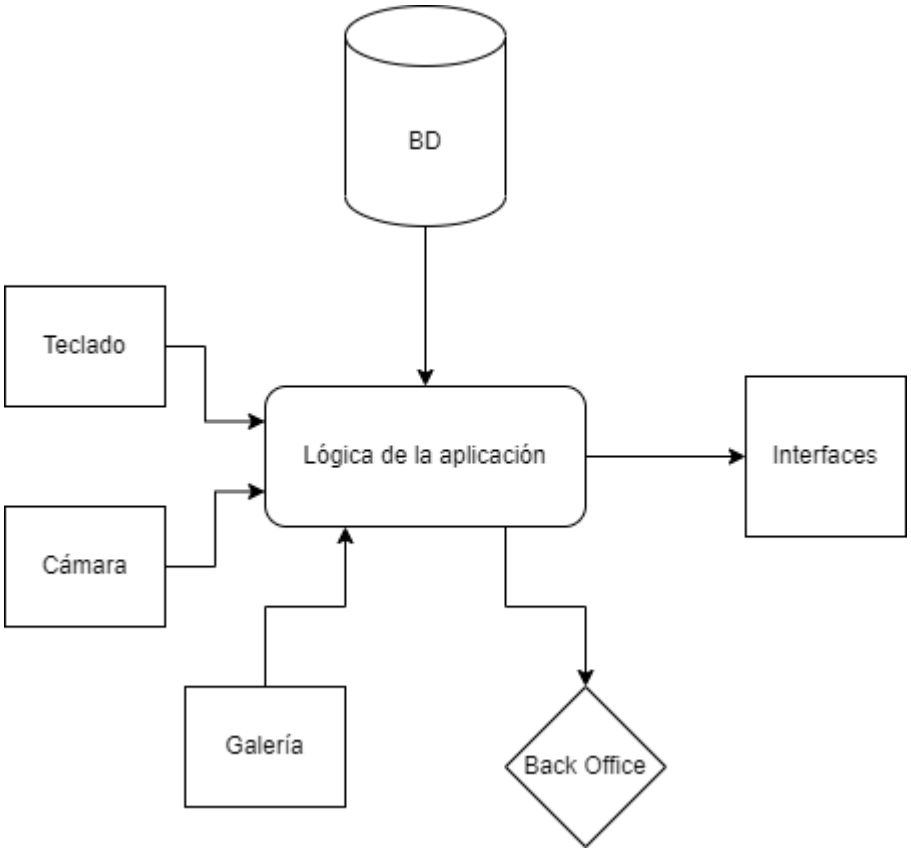


Ilustración 4. Esquema conceptual de Dogizzy

(Fuente propia)

Los elementos principales con los que se interactuará serán los siguientes:

- Cámara, teclado y galería, que serán las entradas de información a la aplicación para que pueda ser procesado por esta.
- Base de datos que formará parte de la persistencia de la aplicación, donde se almacenará la información para su posterior consulta y uso.
- Back Office que será un servicio externo el cual se hará cargo de monitorizar el estado de la aplicación, su logs y cuadros de mando con el fin de optimizar y mejorar la aplicación.
- Interfaces las cuales se actualizarán conforme el usuario haya producido información y esta haya sido procesada por la aplicación.

8.2 Diseño de la arquitectura tecnológica

Para satisfacer las condiciones expuestas en la arquitectura conceptual se ha decidido usar el siguiente *stack* o pila tecnológica.

La tecnología base de la aplicación será nativa, específicamente para *Android*, para poder utilizar así *APIs* propias del sistema y tener un mejor rendimiento. Se utilizará el lenguaje de programación *Kotlin*, desarrollado y mantenido por *JetBrains* mayoritariamente, y con el kit de herramientas de *Android: Jetpack Compose*.

La persistencia de la información se gestionará mediante el uso de *Firebase*[25], el cual se explicará más detalladamente en el apartado *8.3 Diseño de la persistencia*.

Finalmente, para el *back office (RF-ADM-001)* se hará uso de los cuadros de mando de *Firebase*, incluyendo *Crashlytics*[26] para el seguimiento y análisis de errores, *Performance Monitoring* [27]para monitorizar y optimizar el rendimiento de *Dogizzy*, *Test Lab* [28]para realizar pruebas en dispositivos reales y *Google Analytics*, para explorar y analizar la cantidad y el comportamiento de usuarios en la aplicación.

Al usar *Firebase* como base de datos principal, es idóneo utilizar estos cuadros de mando para obtener información útil para mejorar la aplicación y ahorrarnos también el tener que desarrollar un back office propio con todo el coste temporal que eso supondría.

8.3 Diseño de la persistencia

[25] *https://firebase.google.com/* [Último acceso 10/07/2022]
[26] *https://firebase.google.com/products/crashlytics* [Último acceso 10/07/2022]
[27] *https://firebase.google.com/products/performance* [Último acceso 10/07/2022]
[28] *https://firebase.google.com/products/test-lab* [Último acceso 10/07/2022]

Todos los datos de *Dogizzy* se almacenarán en el servidor de *Google Firebase,* una base de datos en la nube que facilita el desarrollo y creación de aplicaciones al gestionar todos los datos de nuestra aplicación que queramos.

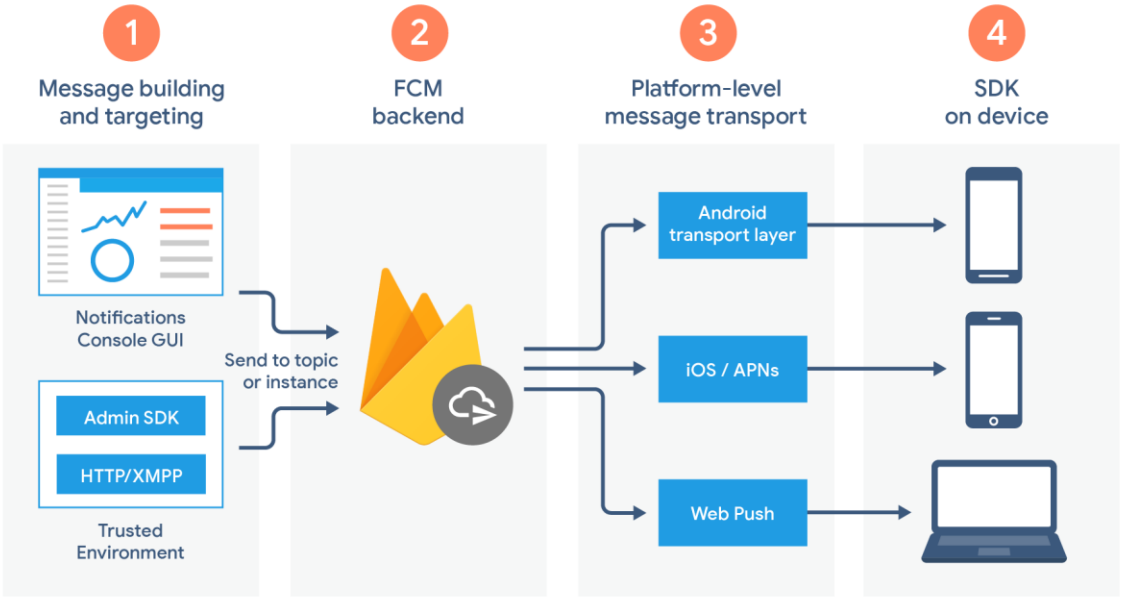


Ilustración 5. Descripción general de la arquitectura de FCM

(firebase.google.com)

Al ser *Dogizzy* un proyecto que en el tiempo de la entrega se quedará en fase demo, usaré el plan de suscripción *Spark*, el cual nos proporciona las opciones básicas que ofrece *Firebase* de forma gratuita. Concretamente, usaremos las funciones de *Authentication, Firestore Database* y *Storage*. En *Authetication* almacenaremos los usuarios, en *Firestore Database* todos los datos de los usuarios registrados y con *Storage* almacenaremos todo el contenido multimedia que suban.

En primer lugar, en *Authentication* guardaremos los emails y contraseñas de cada usuario, a la vez que la fecha de creación de la cuenta, la última vez que accedieron a la cuenta de la aplicación y un UID autogenerado que identificará al usuario. Las contraseñas están cifradas y guardadas de manera que no podemos acceder a ellas desde el cuadro de mandos de *Firebase*.

En segundo lugar, en *Firestore Database* almacenaremos todos los datos del usuario, en una distribución que será por colecciones y documentos. Cada colección puede tener múltiples documentos, y cada documento podrá tener múltiples colecciones, y así infinitamente. Nuestra base de datos constará de una primera colección llamada *Usuarios* que tendrá dentro documentos identificados con el UID de cada usuario. Dentro de estos documentos estará la siguiente información: el nombre del usuario, su fecha de nacimiento, su lugar de residencia, una biografía del usuario y una lista de sus intereses. Además, dentro de cada documento habrá una colección llamada *Chats*, donde se almacenarán los chats de cada usuario. Similar a la colección *Usuarios*, en *Chats* habrá varios documentos con los UID de los usuarios con los que el usuario identificado en la colección anterior haya mantenido una conversación, y dentro de *Chats* habrá otra colección llamada *Mensajes*, que como indica el nombre, contendrá documentos que serán los mensajes entre los dos usuarios. Estos mensajes estarán identificados con IDs generados automáticamente, y contendrán la siguiente información: el mensaje enviado, el usuario que ha enviado el mensaje y cuando se ha enviado el mensaje.

En último lugar, los datos multimedia que el usuario suba a la plataforma se almacenan en *Storage*. Actualmente, el usuario solo podrá subir imágenes, las cuales se dividirán en dos tipos: (1) imágenes de perfil e (2) imágenes del usuario. *Storage* divide su información por carpetas, por lo tanto, tendremos dos carpetas identificativas para cada tipo de imagen. Dentro de estas dos carpetas dispondremos de carpetas identificadas por el UID de cada usuario, las cuales contendrán las fotos de cada respectiva clase.

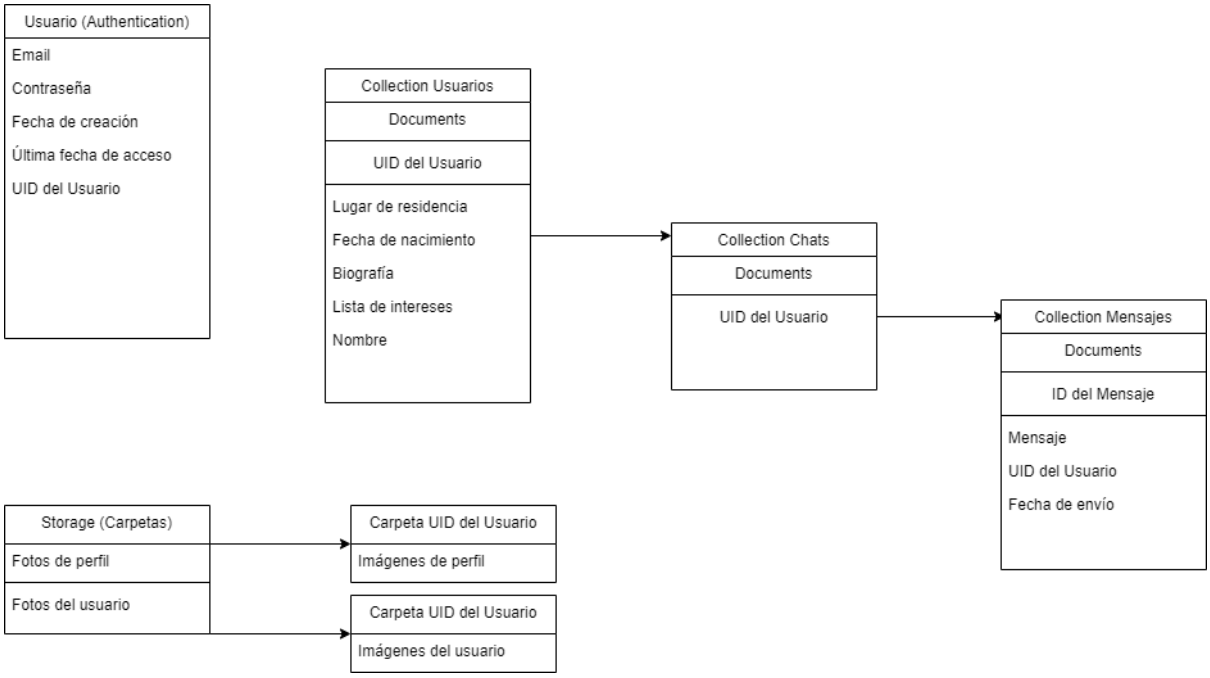


Ilustración 6. Esquema de base de datos Authentication, Firestore Database y Storage de Firebase (Fuente Propia)

8.4 Guía de estilos

Para conseguir un proceso de desarrollo eficiente es muy conveniente definir una guía de estilos la cual seguir y ahorrar así incertidumbres a la hora de diseñar las interfaces de la aplicación.

8.4.1 Filosofía de diseño

Dogizzy consta de interfaces simples y limpias con interacción que ayudan al flujo de navegación del usuario con la idea de hacer la experiencia del usuario más atractiva y elegante. Para conseguirlo, se ha seguido el sistema de *Material Design* de *Google*, incluyendo su última revisión: *Material Design 3*[29].

Material Design 3 nos ayudará a incorporar un sistema de lenguaje visual que sintetiza los principios clásicos de un buen diseño con la innovación de la tecnología. Además, *Material Design 3* es compatible por medio de librerías con el kit de herramientas *Jetpack Compose*

[29] https://m3.material.io/ [Último acceso 10/07/2022]

que vamos a utilizar, y también se puede utilizar en herramientas de diseño como *Figma*[30] que utilizaremos para diseñar las interfaces.

8.4.2 Logo e iconografía

El logo de *Dogizzy* es la imagen de cuatro huellas caninas sobre un degradado de fondo con los colores principales de la aplicación.

Ilustración 7. Logo de Dogizzy

(Fuente propia)

[30] https://www.figma.com/ [Último acceso 10/07/2022]

Por otro lado, para las funciones generales y destacar algún elemento se utilizarán los iconos de la librería *Zest* de *Figma.*

Ilustración 8. Ejemplos de iconos Zest de Figma

(Fuente propia)

8.4.3 Colores

Siguiendo las directrices marcadas por *Material Design 3* y teniendo en cuenta la psicología del color, en concreto, en qué transmite cada color, se han establecido los siguientes tres colores principales: *Yellow Orange, Pastel Red* y *Radical Red*, como se puede observar en la Ilustración 9.

Ilustración 9. Colores de Dogizzy

(Fuente propia)

Por lo tanto, *Yellow Orange* es el color principal, *Pastel Red* es el color secundario y, en última instancia, *Radical Red* el color de acento. Esta combinación de colores está contrastada de una manera muy suave, con *Pastel Red* siendo el intermedio de *Yellow Orange* y *Radical Red.* Se han escogido estas tonalidades más cálidas para trasmitir una sensación de cercanía, alegría y familiaridad, atributos que ayudarán a definir el objetivo de *Dogizzy*.

8.4.4 Tipografía

La tipografía que se usará como representante de la marca será la familia de fuentes *Gilroy*, más concretamente en sus variantes *Light*, *Bold* and *ExtraBold,* distribuidas de forma gratuita.

La jerarquía que se seguirá está detallada en la Ilustración 10. Dicha jerarquía se tomará de referencia y se adecuará el tamaño de la tipografía dependiendo del contexto en el que nos encontremos.

Ilustración 10. Jerarquía de tipografías de Dogizzy

(Fuente propia)

8.4.5 Diseño de interacción o experiencia de usuario

La idea principal de *Dogizzy* es facilitar las relaciones entre dueños y mascotas, por lo tanto, diseñar una buena interacción entre el usuario y la aplicación va a ser un elemento importante para su éxito.

Para ello, utilizaremos un *User Journey Map* basado en experiencias propias con aplicaciones similares de chat online para poder ver así cuales son las experiencias positivas y negativas al utilizar una aplicación de este tipo.

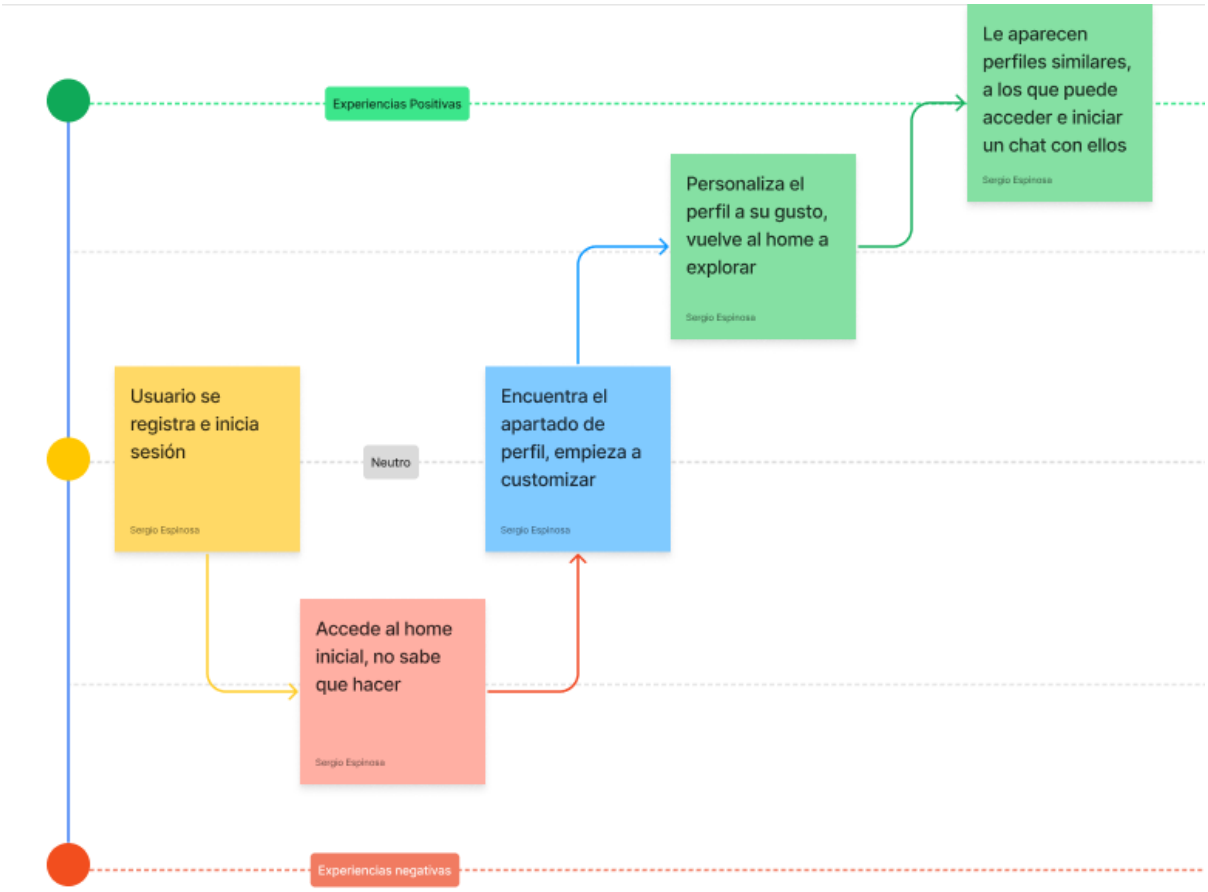


Ilustración 11. Demo de User Journey Map de Dogizzy

(Fuente propia)

Como se puede ver en la Ilustración 11 anterior, lo que se espera del primer uso de *Dogizzy* es que nos encontremos con una serie de experiencias neutras-negativas hasta llegar al momento de interactuar con otros perfiles.

Por lo tanto, lo que más mejorará la sensación del usuario al usar la aplicación será que tenga que realizar el mínimo número de pasos posibles al principio para poder así interactuar cuanto antes con los demás usuarios de la aplicación.

Dogizzy se centra en esto mostrando principalmente los perfiles de los demás usuarios conforme accedes a la aplicación, además de poder editar tu perfil de forma intuitiva y rápida para minimizar la sensación de pesadez que uno puede obtener al estar mucho tiempo personalizando su perfil.

8.4.6 Diseño de interfaces

A continuación, se mostrarán diseños de las interfaces mediante *mockups* realizados en la aplicación *Figma*, donde se habían diseñado tres interfaces principales las cuales han sufrido cambios durante el proceso de desarrollo.

La aplicación se abre en el inicio de sesión de esta, y se ha dejado el diseño intacto durante todo el proceso de desarrollo.

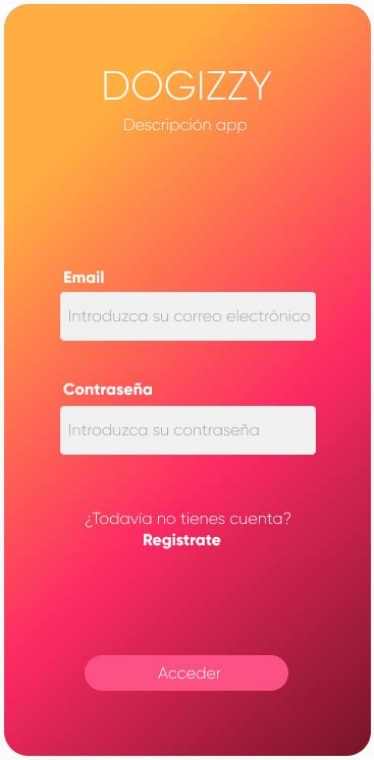


Ilustración 12. Mockup de Inicio de Sesión

(Fuente propia)

En el momento de diseñar estas interfaces, todavía no tenía muy clara cuál iba a ser la pantalla principal una vez iniciado sesión, por lo que en lo siguiente que me centré fue en el diseño del perfil, que sí que ha sufrido algún cambio menor a la hora de implementarlo.

Ilustración 13. Mockup del perfil del usuario

(Fuente propia)

Las partes principales se han desarrollado e implementado igual que en el *mockup*, sin embargo, al hacer uso de dos tipos diferentes de perfiles, ha habido varios cambios a la hora de interactuar con las partes de la interfaz a la hora de editar el perfil y chatear con otros usuarios.

En el perfil del propio usuario, la opción de chatear se ha cambiado por la opción de poder editar la cabecera del perfil, y se ha añadido una opción de configuración también. Todos estos cambios a las interfaces los veremos en el apartado 9 *Implementación*.

Por otra parte, los perfiles de otros usuarios contarán con la opción de chatear como muestra el *mockup*, pero la opción de seguir los perfiles se quitó completamente al no cuadrar con la idea principal de *Dogizzy*. Además, los iconos situados al lado del título de cada apartado se retiran en esta interfaz, al ser estos los que permitirán editar cada sección.

Por último, el último *mockup* realizado se ha cambiado completamente para simplificar la aplicación y sus funcionalidades. La primera idea de la aplicación contaba con un buscador de perfiles, pero al suprimir la opción de seguir perfiles, y con esto suprimir un identificador de cada perfil que se pueda buscar, se eliminó también esta idea. A cambio, se idealizó la pantalla principal de la aplicación: la lista de perfiles con los mismos intereses al usuario, simplificando así estas dos primeras ideas en una más concisa que se centra más en el objetivo principal de *Dogizzy*: las relaciones entre dueños y mascotas.

Ilustración 14. Mockup de la idea de buscador de Dogizzy.

(Fuente propia)

Como se puede observar en el *mockup*, se utilizaba un sistema de filtros para buscar perfiles con los intereses del usuario, el cual se utiliza ahora automáticamente en la interfaz principal con los intereses que tenga puesto el usuario en su perfil.

Al simplificar esta interfaz en otra diferente juntando dos ideas, también se simplificó la barra de navegación inferior que se ve en las tres ilustraciones, dejando solo las interfaces

principales accesibles desde la navegación: la lista de chats, la lista de perfiles de usuarios y el perfil del usuario.

8.4.7 Diseño de pruebas y validación

Para asegurar un correcto funcionamiento de la aplicación, aparte de la realización de pruebas manuales, se hará uso de la herramienta *Firebase Test Lab* con su funcionalidad *Robo*.

La funcionalidad *Robo* nos permite subir nuestra aplicación a un servidor de *Google* para que desde ahí se pruebe la aplicación en varios dispositivos físicos. La ventaja de utilizar *Robo* es que la posibilidad de ahorrar el diseñar pruebas individuales, ya que *Robo* es capaz de analizar la interfaz y navegar automáticamente entre ellas.

Al finalizar las pruebas en *Robo*, la herramienta nos devolverá un reporte que incluye logs, capturas de pantalla y videos que nos permitirán detectar errores de manera fácil y visual.

Por otra parte, se usarán los servicios de *Firebase Crashlytics* para el seguimiento y análisis de errores y *Firebase Performance Monitoring* para comprobar el rendimiento de la aplicación entre los diferentes usuarios.

9 Implementación

A continuación, se documenta el proceso de desarrollo siguiendo la metodología explicada en el apartado 6 *Metodología* y la planificación realizada en el apartado 3 *Planificación*. Al estar planificados *sprints* de bastante duración, se subdividirán los *sprints* en este apartado en función a las funcionalidades y objetivos conseguidos para una mejor explicación del proceso de desarrollo.

9.1 *Sprint* 1: Preproducción

El entorno de desarrollo se ha configurado en un portátil con Windows 10 donde se han instalado los diferentes programas y utilidades para poder llevar a cabo la implementación de *Dogizzy*.

Como *IDE* utilizaremos *Android Studio*[31], la adaptación por parte de *Google* de *IntelliJ IDEA* de *JetBrains* para el desarrollo de *Android*. Es la herramienta por excelencia del mundo del desarrollo Android gracias a su cantidad de utilidades para el desarrollo, testeo y *debugging*[32], facilitándonos así el proceso de desarrollo.

En *Android Studio* configuramos un emulador en el cual haremos las pruebas (Google Pixel 4) con versión de *Android 9.0.* También llevaremos a cabo un control de versiones de la aplicación con *Git* junto a *GitHub*.

9.2 *Sprint* 2: Especificación de requisitos y diseño de interfaces

Una vez todo preparado para empezar a desarrollar, es importante saber qué vamos a desarrollar y de qué manera.

Tras un estudio de las principales aplicaciones sobre mascotas (la mayoría mascotas caninas y felinas) como se puede ver en la Tabla 3 del apartado 4 *Estado del Arte*, se decide la dirección en la que desarrollaré la aplicación.

Primero, se especifican los requisitos para saber que tenemos que desarrollar (Apartado 7 *Análisis y especificación*), de manera que las funcionalidades básicas queden claras, aunque pueden estar sujetas a cambios a medida que desarrollamos la aplicación.

Una vez definidos los requisitos, diseñamos varias interfaces de la aplicación para saber el estilo que va a tener, logos, tipografías y colores. De esta manera, tenemos una idea más clara de cómo empezar a desarrollar la aplicación ya que estos requisitos con el diseño de interfaces nos sirven de guía para asentar una primera base de la aplicación.

9.3 *Sprint* 3: Creación de la estructura base de la aplicación

Lo primero que se ha realizado una vez tenido el entorno de trabajo configurado y una pequeña base conceptual es la estructura base de la aplicación. En este *sprint* se

[31] https://developer.android.com/studio [Último acceso 10/07/2022]
[32] *Debugging*: proceso de búsqueda y solución de fallos o bugs (Hostgator, s.f)

implementan las interfaces de Inicio de sesión, Registro y Perfil, al ser las interfaces que estaban diseñadas anteriormente en la herramienta *Figma*.

Para comenzar a desarrollar estas pantallas lo primero es crear por lo menos un *Activity*, clases especiales que usa *Android* para referirse a las pantallas de las aplicaciones. Lo ideal sería crear varios *Activity*, en los cuales dividir el código de cada interfaz principal, sin embargo, por falta de experiencia se desarrollará la aplicación con un solo *Activity* que utilizará la navegación de *Jetpack Compose* para navegar entre las interfaces de la aplicación. Dicho esto, la estructura de ficheros del proyecto se dividirá aun así por interfaces y/o funcionalidades principales, siguiendo las recomendaciones de programación de *Android* y asegurar así un alto nivel de modularidad y limitar el código duplicado. Asimismo, para la estructura de las interfaces se ha hecho uso de diferentes archivos de parametrización que ofrece *Android*, como *Color.kt, Shape.kt, Type.kt y Theme.kt*.

Gracias a utilizar estos parámetros, se facilita la personalización de las interfaces además de poder crear un tema oscuro de manera muy sencilla, al solo tener que sobrescribir valores que *Android* puede identificar automáticamente cuando el usuario tiene su dispositivo móvil con tema claro u oscuro, sin tener que cambiar los temas de la aplicación de forma manual.

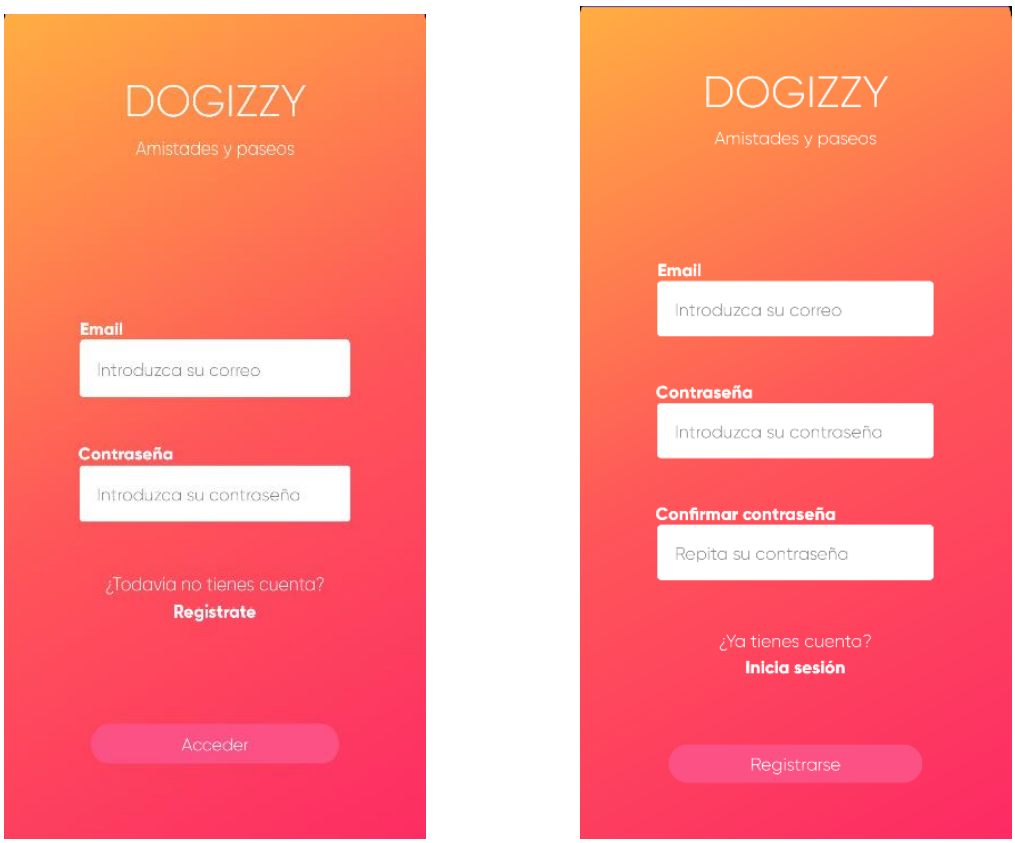


Ilustración 15. Interfaces de Inicio de sesión y Registro

(Fuente propia)

Como se puede apreciar en la Ilustración 12, las dos interfaces son prácticamente idénticas al cumplir funcionalidades que se apoyan entre sí. Estas interfaces desde el primer momento fueron desarrolladas al completo con *Firebase Database* para la gestión de usuarios registrados para así comenzar a familiarizarnos con *Firebase* y poder probar la aplicación en funcionamiento y no solo ver las vistas previas de las interfaces desde *Android Studio*.

Para el desarrollo de esta interfaz se han utilizado elementos *layout* de *Jetpack Compose* básicas como *Surface* para envolver todo el contenido, *Columns* y *Rows* para distrubir el contenido y alinearlo y *Texts, TextFields y Buttons* para mostrar e introducir contenido. Todos estos elementos facilitan el proceso de desarrollo al poder crearlas con muy pocas líneas en comparación de cómo sería con *Android* puro en *XML* y se utilizarán en todas las interfaces que desarrollemos, por lo que en las siguientes interfaces mencionaré los elementos *layout* utilizados que destaquen y no se utilicen generalmente.

Ilustración 16. Interfaz de perfil del usuario registrado

(Fuente propia)

En la Ilustración 16 se muestra la interfaz del perfil del usuario registrado en su versión final, sin embargo, en este *sprint* solo se implementó la interfaz de manera estática, por lo que las opciones de editar el perfil se implementaron más adelante.

Para la lista de intereses se ha utilizado una *FlowRow* propia, ya que la *Row* que nos proporciona *Jetpack Compose* no se adaptaba al añadir nuevos intereses, y a que *Jetpack Compose* ofrece la posibilidad de hacer tus propias funciones de *layout* para que se ajusten a lo que buscas mostrar en la interfaz. Esta *FlowRow* muestra una lista de elementos *Card* con los intereses del usuario.

También mencionar que, para mostrar las imágenes del usuario, en un principio se usó el elemento *layout* de *LazyGrid*, pero tener dos elementos *layout* anidados que permiten *scroll* no está soportado por *Android*, por lo cual también hemos podido aprovechar nuestra *FlowRow* con una lista de elementos *Image* para simular un *LazyGrid.*

Finalmente, también se desarrolló la barra de navegación utilizando el elemento *NavigationBar* en este *sprint*, aunque solo tengamos una interfaz disponible hasta el momento que sería la que estamos observando ahora.

9.4 *Sprint* 4: Chat online e interfaz de buscador

En el siguiente *sprint* se implementa la funcionalidad de chat online y sus interfaces, además de la interfaz de buscador (véase la Ilustración 11 del apartado 8.4.6 *Diseño de interfaces*) que luego se sustituirá y se utilizará su funcionalidad en la interfaz principal descrita en el siguiente *sprint.*

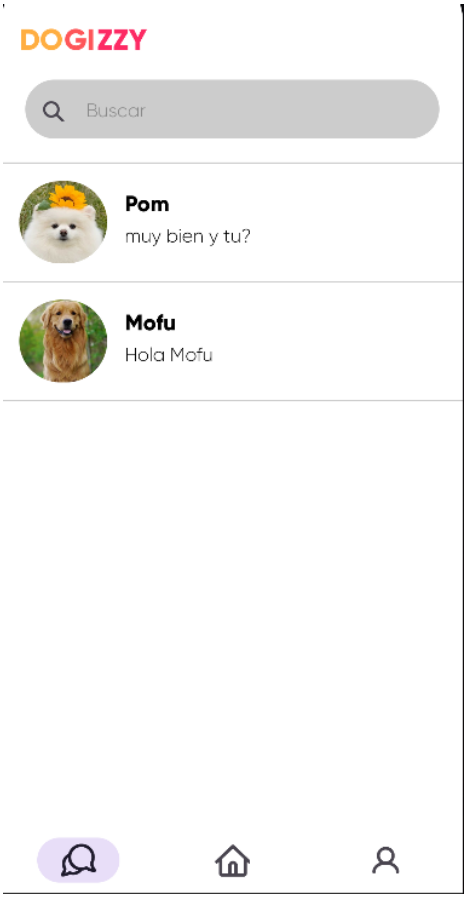


Ilustración 17. Interfaz lista de chats

(Fuente propia)

En la interfaz que vemos en la Ilustración 17, se muestra una lista de chats en los cuales el usuario ha mantenido una conversación anteriormente, con la opción de buscar los chats por el nombre de la mascota del otro usuario. Los chats están distribuidos en una lista con la foto de perfil de cada usuario. De esta manera, se puede acceder a los chats de manera fácil y rápida sin tener que acceder desde el perfil del otro usuario, que es desde donde tenemos que iniciar la conversación previamente.

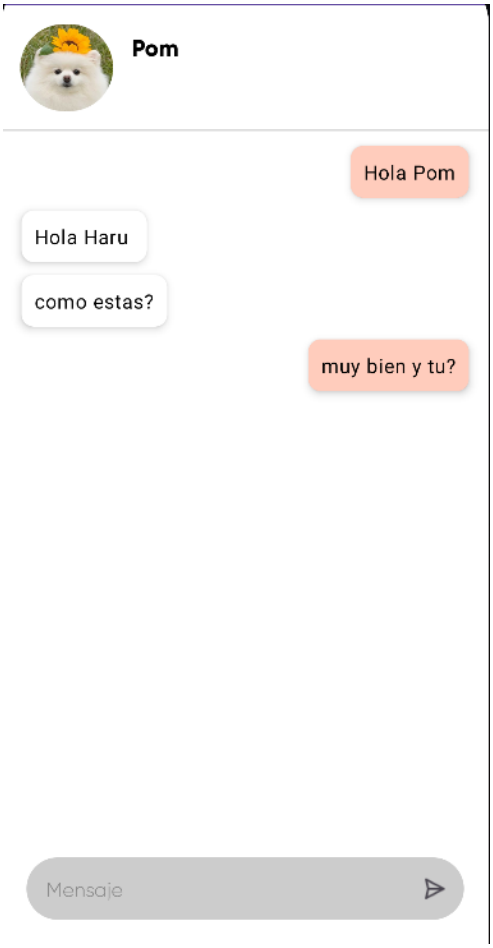


Ilustración 18. Interfaz chat

(Fuente propia)

Una vez dentro del chat, se haya una barra superior con el nombre y la foto de perfil del usuario con el que estamos manteniendo una conversación, los mensajes intercambiados y la barra que abre el teclado del teléfono. Para el desarrollo del chat se ha hecho uso de un *ViewModel* que encapsulará toda la lógica que utilice el chat y las conexiones a la base de datos, separando así el diseño de la interfaz y la lógica de esta. Al ser una versión muy inicial la que se va a presentar, cabe mencionar que los mensajes del chat no estarán cifrados por falta de tiempo y conocimiento, pero lo necesario para que la aplicación se pudiese publicar sería que se cifraran los mensajes de extremo a extremo.

9.5 *Sprint* 5: Interfaz principal y algoritmo de compatibilidad de mascotas.

En este *sprint* se desarrollará la parte principal de la aplicación, la cual cuenta con un algoritmo básico que ordenará los perfiles registrados en la aplicación en orden de

compatibilidad con el usuario, apareciendo en la parte superior de la lista de perfiles los más compatibles y en la inferior los menos compatibles. En el apartado de Anexos se explicará con más detalle las características del algoritmo para relacionar perfiles entre sí en base a lo que publiquen en su perfil.

Ilustración 19. Interfaz principal de la aplicación.

(Fuente propia)

Como se puede apreciar, los perfiles se muestran de dos en dos con su foto de perfil, nombre, edad y lugar de residencia. Se muestra esta información al considerarse esta la principal de cada perfil, ya que la edad y la ciudad en la que viva el usuario pueden ser factores importantes a la hora de relacionarse con otros usuarios. No obstante, desde esta interfaz se accede a los perfiles de los demás usuarios, donde podemos ver toda la información de este.

Como se ha mencionado antes, en esta lista también de elementos *Card* se ha hecho uso también de nuestro *layout* personalizado *FlowRow*.

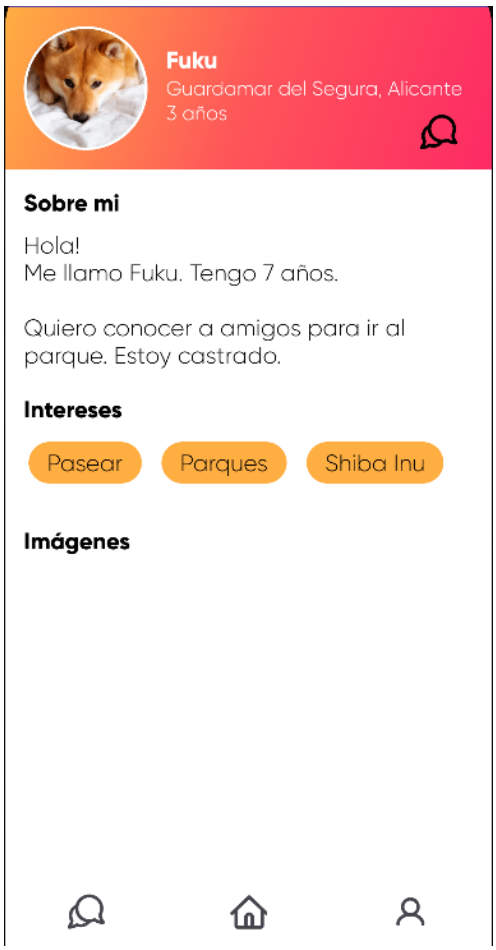


Ilustración 20. Interfaz perfil de otro usuario

(Fuente propia)

Esta interfaz de perfil de usuario, como ya se comentó en el apartado 8.4.6 *Diseño de interfaces* solo se diferencia de la del perfil del usuario propio por las opciones de editar y la opción de iniciar un chat, que como también se mencionó antes, para que un chat aparezca en la lista de chats del usuario es necesario que primero se inicie la conversación desde el perfil del usuario con el que queramos conversar.

9.6 *Sprint* 6: Edición de perfil del usuario y funcionalidad de cámara

Una vez desarrolladas las interfaces principales, lo siguiente que se va a desarrollar son las funciones que permitirán al usuario editar la información de su perfil, factor importante a la hora de relacionar los perfiles con el algoritmo.

Ilustración 21. Interfaz del perfil de usuario con las funciones de editar resaltadas

(Fuente propia)

Las funciones de editar se dividen en tres. La primera, en la parte superior a la derecha, que permite editar la foto de perfil del usuario, el nombre, el lugar de residencia, la edad y la biografía. La segunda, al lado de intereses, que permite editar los intereses del usuario y la tercera, al lado de imágenes que permite editar las imágenes del usuario.

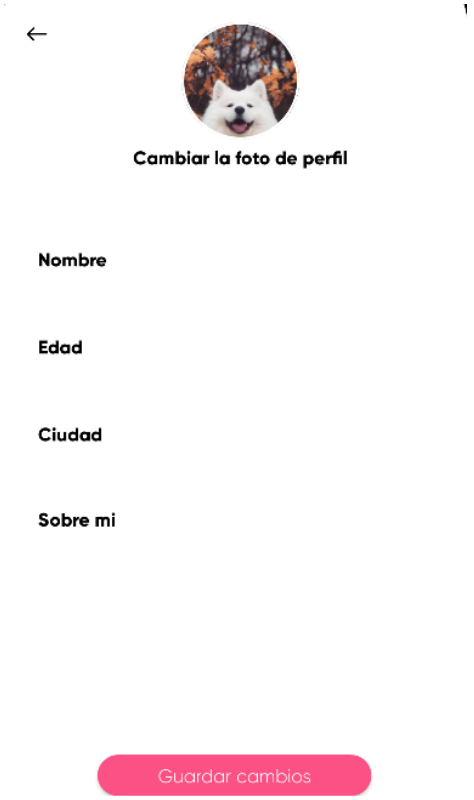


Ilustración 22. Interfaz de edición del perfil 1

(Fuente propia)

Para editar los parámetros nombre, edad, ciudad y sobre mí se utilizan elementos *TextField*, y para editar un elemento *Image* seleccionable, que abre la cámara del dispositivo móvil. Para toda la lógica del usuario se ha utilizado un *ViewModel* como en para el chat online, salvo que este *ViewModel* tendrá todas las funciones relacionadas con obtener y modificar la información del usuario, y por lo tanto aparte de usarse exhaustivamente en estas interfaces, también se utiliza en todas las interfaces de la aplicación. Se puede modificar la información del usuario independientemente, aunque estas características estén disponibles para editar en la misma interfaz. Para no modificar una característica solo hace falta dejar el *TextField* en blanco.

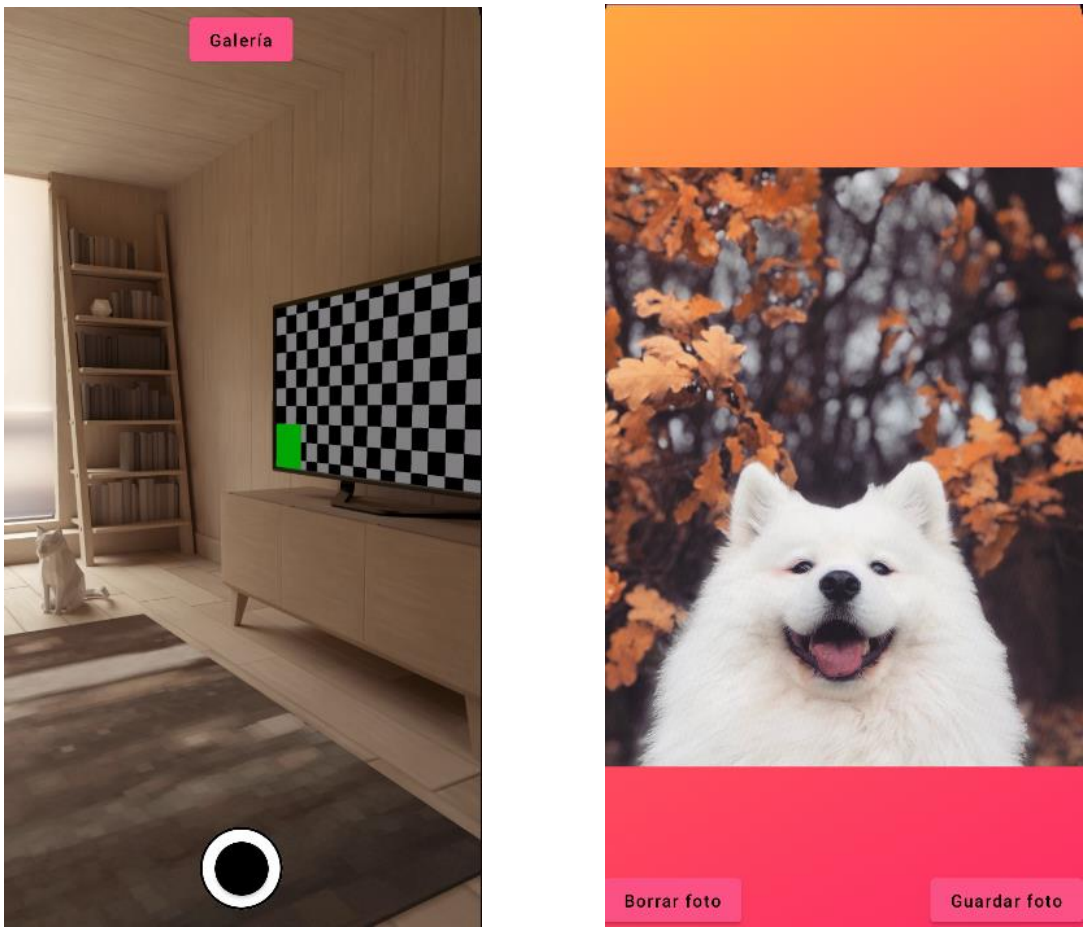


Ilustración 23. Funcionalidad de cámara

(Fuente propia)

Para modificar la foto de perfil, como podemos observar en la Ilustración 23, primero se abrirá la cámara de nuestro dispositivo móvil, dándonos la opción de tomar una foto directamente o de acceder a la galería tras el botón que se muestra en la parte superior.

Una vez tomada la foto o habiendo escogido la foto desde la galería de nuestro móvil, se nos dará la opción o bien de guardar la foto o bien de borrar la foto. Por un lado, si borramos la foto volveremos a la cámara de nuestro teléfono móvil y, por otro lado, si la guardamos aparecerá modificada en la interfaz que en la que nos encontrábamos anteriormente.

Ilustración 24. Interfaz de edición de los intereses del usuario

(Fuente propia)

Siguiendo con la edición de los intereses del usuario, para ello se ha realizado una interfaz que muestre todas las etiquetas disponibles en la aplicación (son un ejemplo, siempre se pueden poner más) y las etiquetas que está mostrando el usuario. Como se puede observar en la Ilustración 24, el usuario tendrá un máximo de diez intereses para mostrar.

Para modificarlos, simplemente tiene que pulsar en la etiqueta que quiera que se muestre en su perfil o, al contrario, si tiene una etiqueta que quiere dejar de mostrar también se realizará el mismo procedimiento. Las listas se reorganizarán solas gracias a los estados de *Jetpack Compose*, que permiten crear un observable manejado por un *ViewModel* que dibujará otra vez la interfaz cada vez que se modifique un valor de las dos listas de etiquetas. Para las listas se utilizará de nuevo la *custom layout FlowRow*.

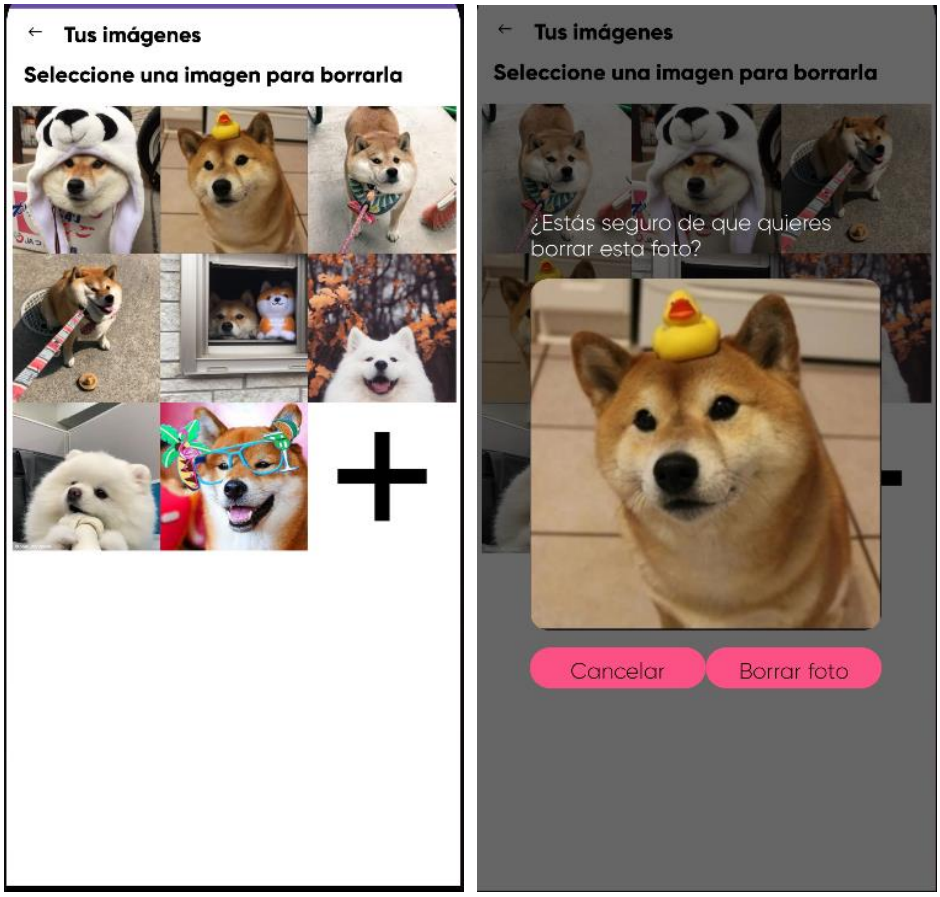


Ilustración 25. Interfaz de edición de imágenes del usuario

(Fuente propia)

Finalmente, para modificar las imágenes del usuario, se han utilizado funciones utilizadas en las dos interfaces anteriores para modificar la información del usuario, como son la funcionalidad de la cámara para añadir una foto pulsando en el símbolo "+" al lado de la última foto, y la funcionalidad de lista observable que se ha explicado en la anterior interfaz. Para eliminar una foto de la lista, simplemente basta con pulsar la imagen que desees eliminar. Para que no se borren imágenes por descuido, se mostrará una alerta para asegurar que es dicha foto la que se quiere borrar. Esta interfaz no cuenta con botón de guardado ya que las fotos se actualizan en la base de datos directamente al modificar la lista.

9.7 *Sprint* 7: Configuración y retoques

Una vez finalizadas las funciones de edición del perfil, ya solo faltarían las opciones de configuración e información de la aplicación y algún que otro retoque estético para llamar la atención del usuario.

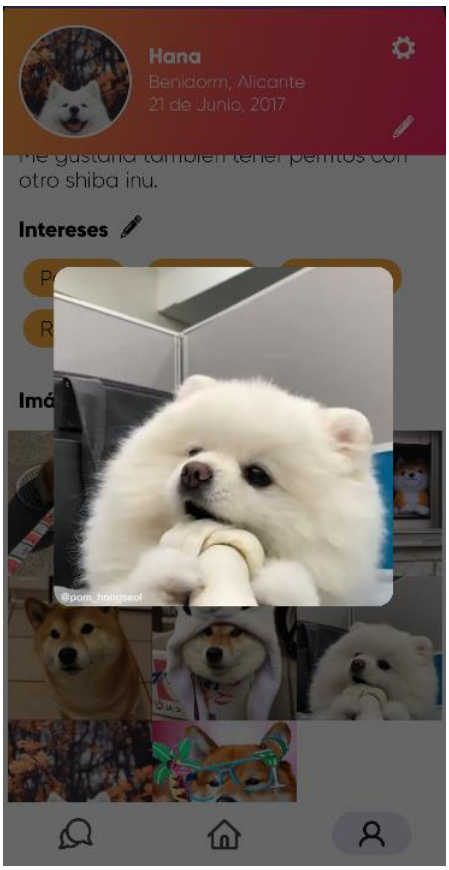
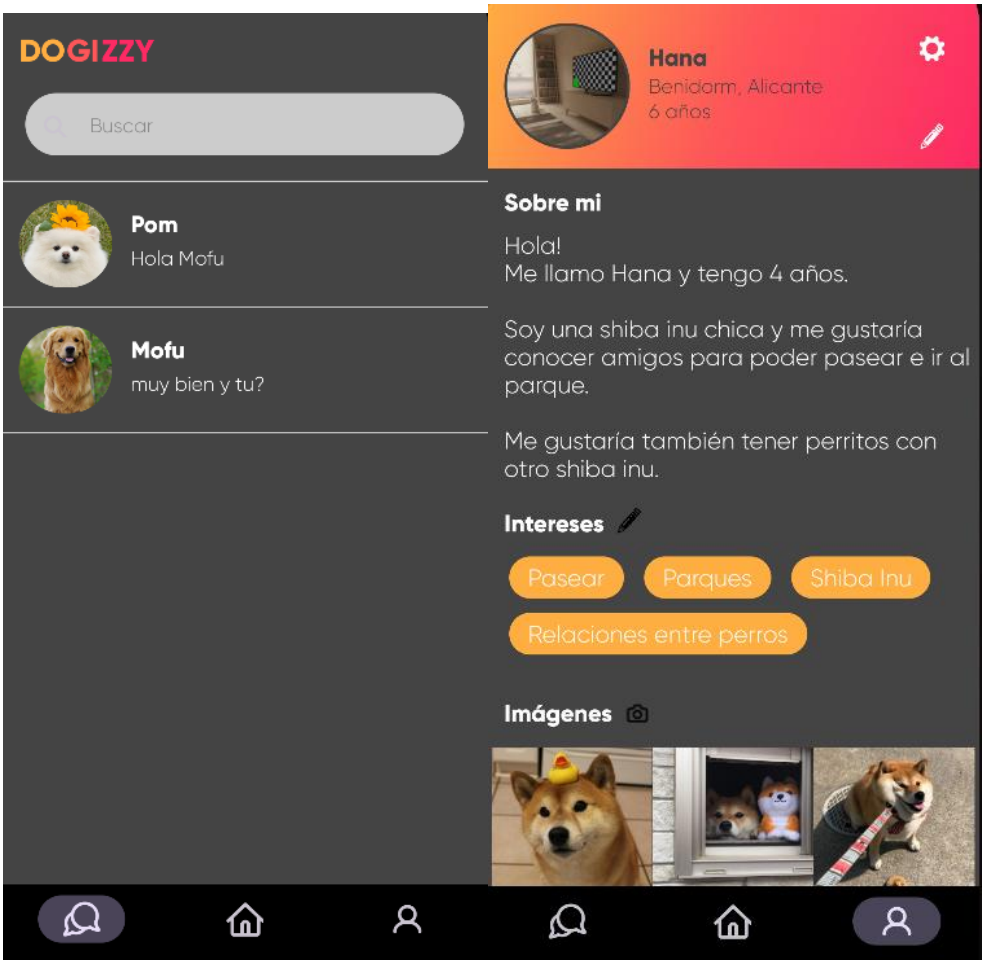


Ilustración 26. Pop-up de las imágenes del perfil

(Fuente propia)

Para que el usuario pueda ver con claridad las fotos de la mascota, se ha desarrollado una ventana *pop-up* con la imagen seleccionada. Para salir de esta ventana solamente haría falta pulsar en cualquier lado gris de la pantalla.


DOGIZZY
Buscar
Pom
Hola Mofu
Mofu
muy bien y tu?
Hana
Benidorm, Alicante
6 años
Sobre mi
Hola!
Me llamo Hana y tengo 4 años.
Soy una shiba inu chica y me gustaría conocer amigos para poder pasear e ir al parque.
Me gustaría también tener perritos con otro shiba inu.
Intereses
Pasear
Parques
Shiba Inu
Relaciones entre perros
Imágenes

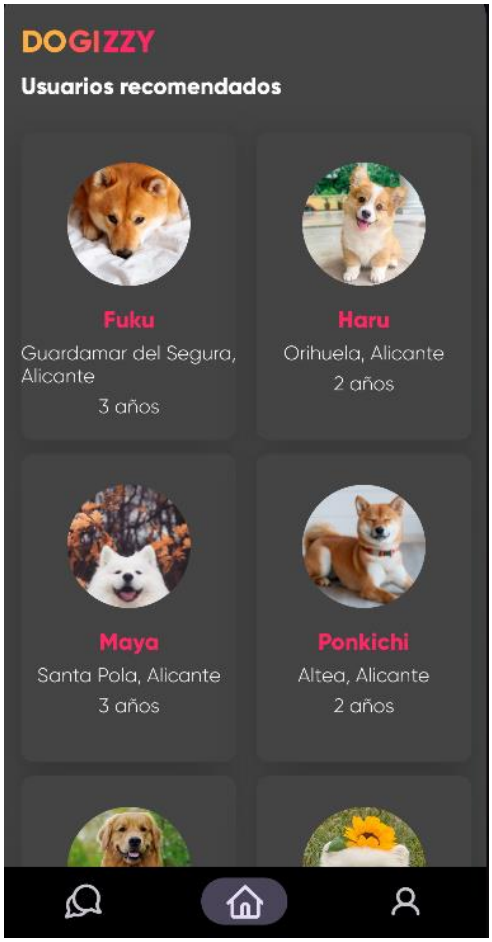

DOGIZZY
Usuarios recomendados
Fuku
Guardamar del Segura, Alicante
3 años
Haru
Orihuela, Alicante
2 años
Maya
Santa Pola, Alicante
3 años
Ponkichi
Altea, Alicante
2 años

Ilustración 27. Interfaces principales con tema oscuro

(Fuente propia)

Por último, en las opciones de configuración de la aplicación de momento solo contamos con una opción que muestra las licencias de las librerías usadas en nuestra aplicación, y una opción de cerrar sesión. También se muestra abajo del todo la versión de *Dogizzy* que se tiene descargada actualmente.

Ilustración 28. Configuración de la aplicación

(Fuente propia)

10 Pruebas y validación

Durante el periodo de desarrollo de la aplicación se ha ido probando su funcionamiento en un dispositivo que estaba disponible en los emuladores de *Android Studio: Google Pixel 4* para así comprobar rápidamente si los nuevos cambios funcionaban correctamente y no interferían con funcionalidades anteriormente implementadas.

Al finalizar la implementación, las pruebas se han extendido a otros dispositivos móviles que se encontraban al alcance, como un *Xiaomi Mi 8*, un *Redmi 9* y un *LG k 40S*. En ninguno de los dispositivos se ha notado apenas ralentización, asegurando un correcto y fluido funcionamiento independientemente del modelo de teléfono.

Además de pruebas directas de gente conocida con sus propios dispositivos, se han realizado pruebas automáticas en el mencionado *Firebase Test Lab*, usando el test automático mediante la herramienta *Robo*.

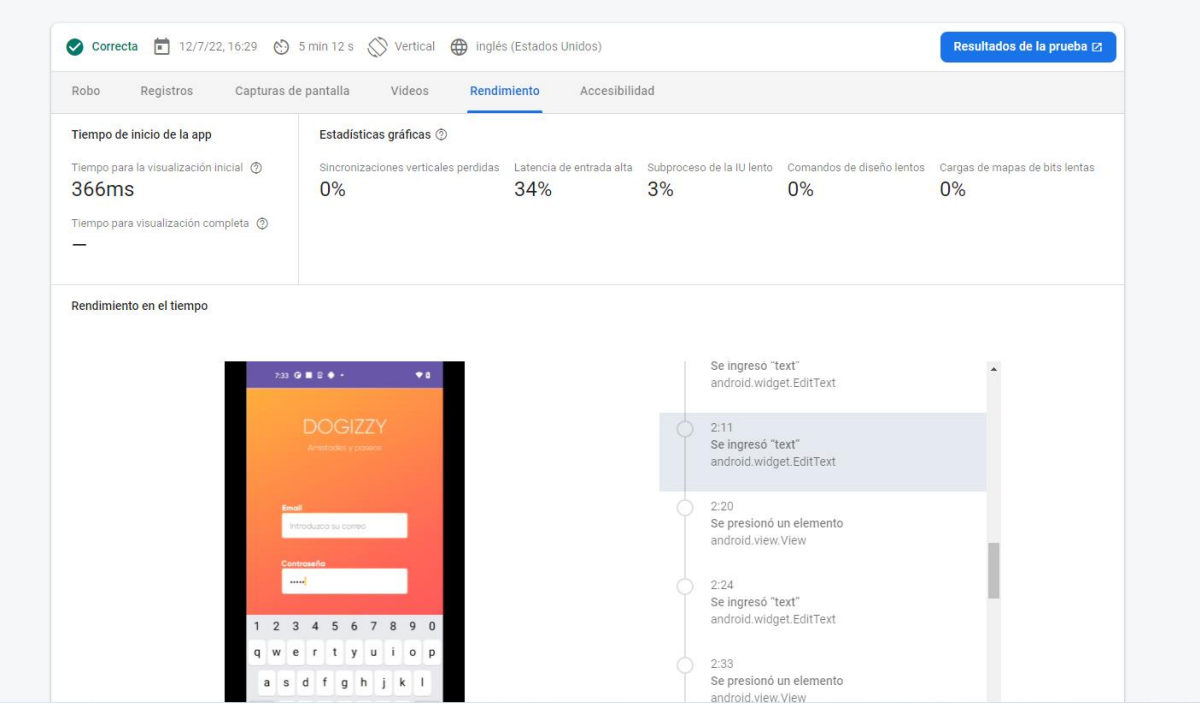


Ilustración 29. Captura de test automático Robo 1

(Fuente Propia)

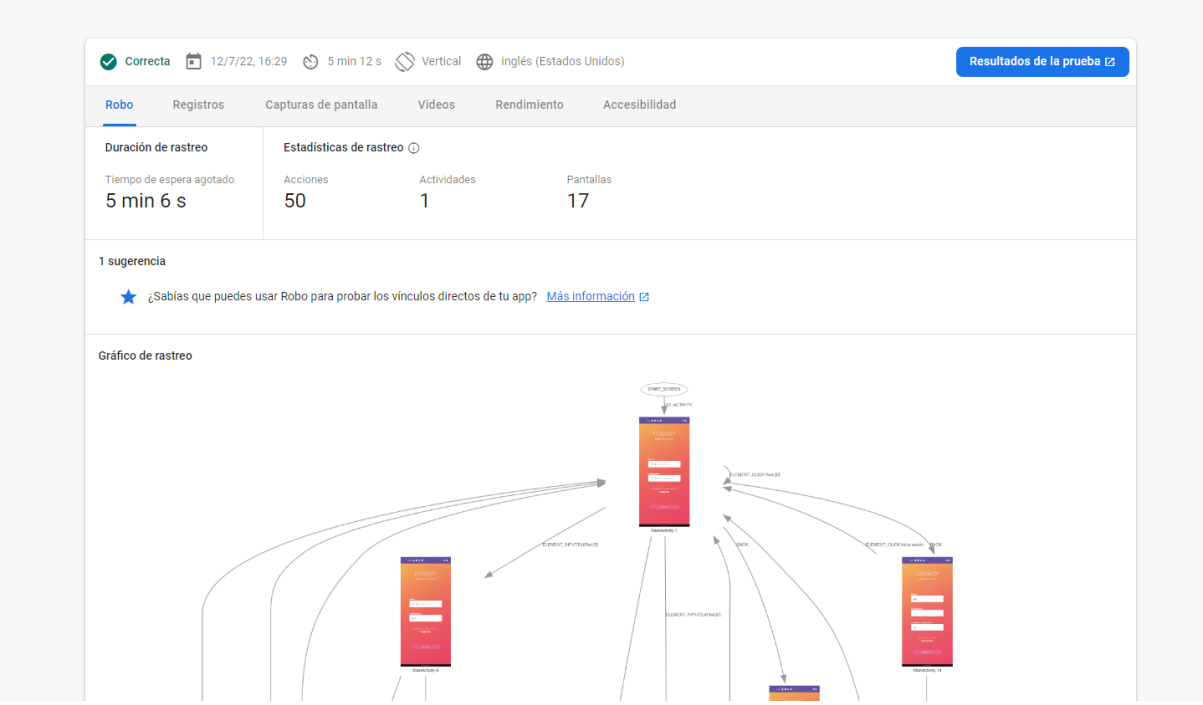


Ilustración 30. Captura de test automático Robo 2

(Fuente propia)

Todos los test realizados se han pasado con éxito y sin ningún fallo de funcionamiento. Se puede observar que, en cuanto al rendimiento, la aplicación se beneficia de haber sido desarrollada en nativo. Sin embargo, podemos observar que hay bastante latencia de entrada en los inputs y es algo que podríamos mejorar.

Otras herramientas que vamos a utilizar para recoger reportes sobre la aplicación serán *Firebase Crashlytics* y *Performance Monitoring*. Con *Firebase Crashlytics* podemos recoger información sobre los cierres inesperados de la aplicación y en qué línea de código se sitúa dicho error para poder corregirlo.

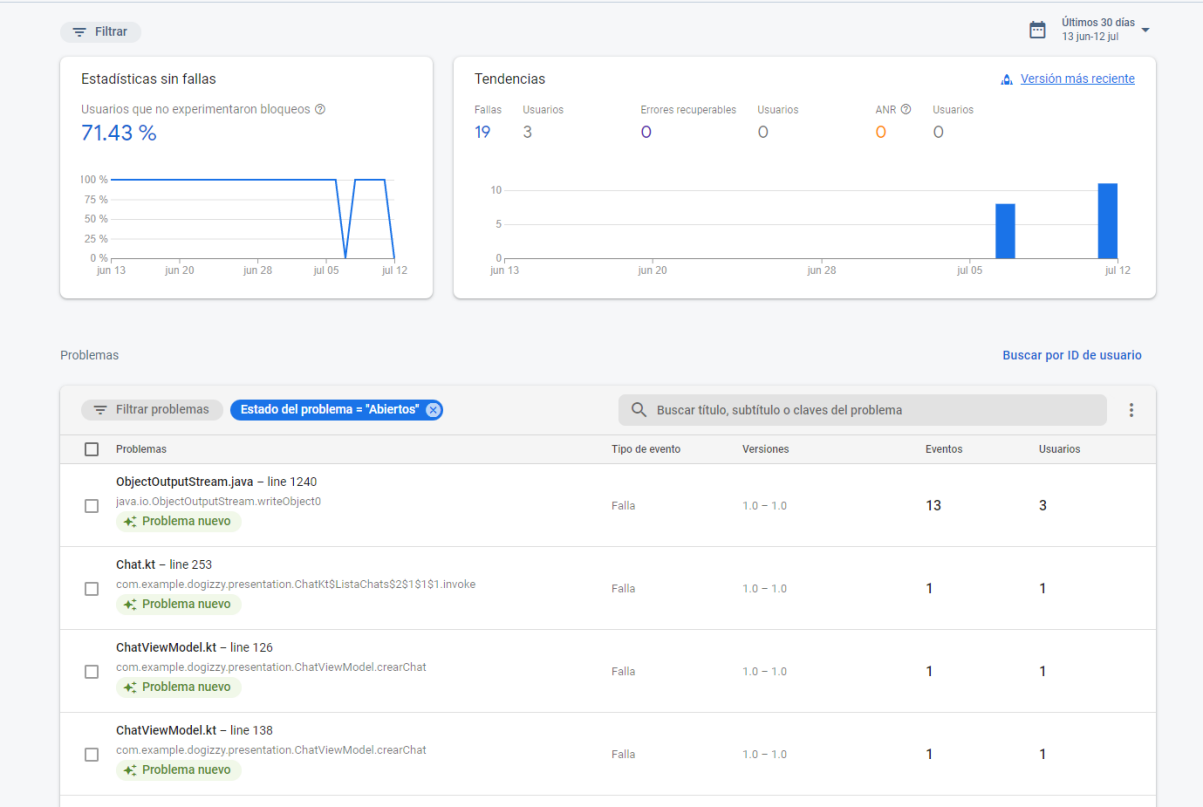


Ilustración 31. Captura de los cuadros de mando de Firebase Crashlytics.

(Fuente propia)

En el siguiente ejemplo podemos ver que desde que se incluyó *Firebase Crashlytics* a la aplicación ha habido diversas fallas que provocaban que la aplicación dejase de funcionar, pero gracias a esta herramienta se pueden identificar fácilmente las líneas en la que la aplicación falla para poder solucionarlo.

Similarmente, los cuadros de mando de *Performance Monitoring* nos permiten observar si hay algún problema de rendimiento, ya sea de la aplicación en sí, la conexión a la base de datos o a la red. De esta manera podremos saber cuándo y dónde hay ralentizaciones en el funcionamiento de la aplicación.

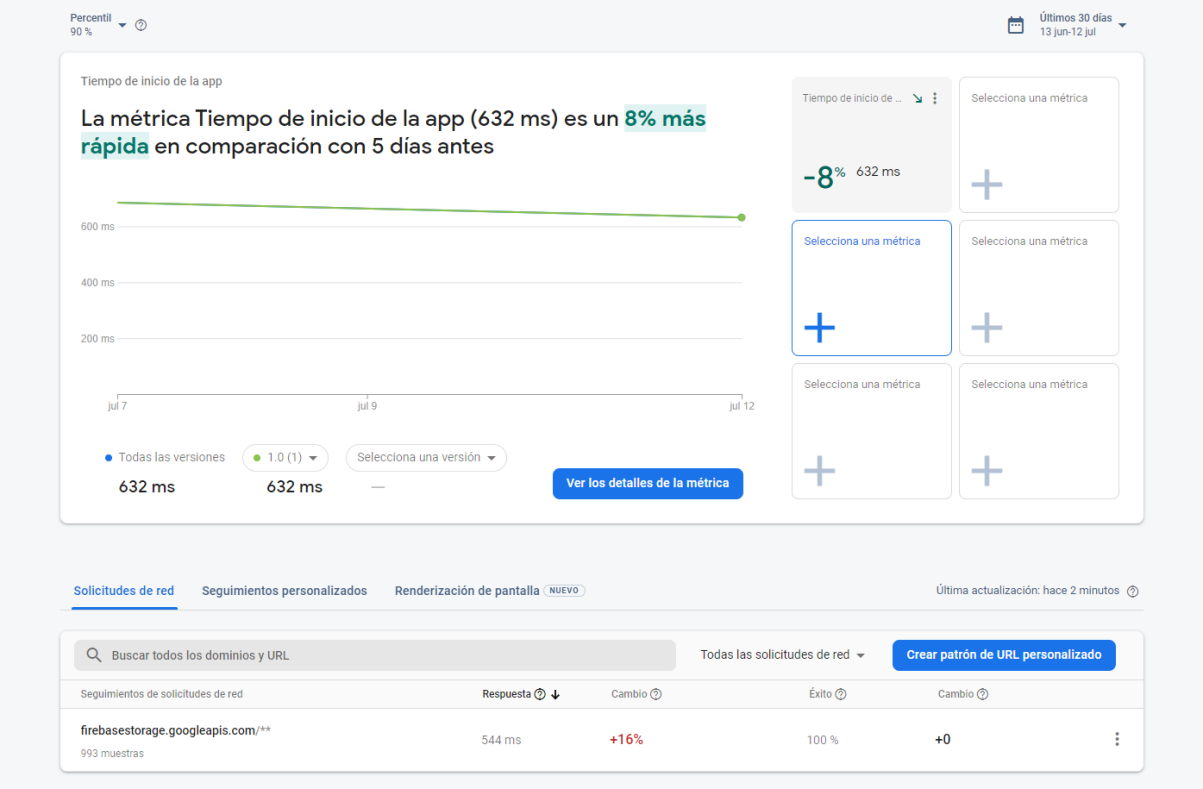


Ilustración 32. Captura de los cuadros de mando de Performance Monitoring.

(Fuente propia)

Como se puede observar en la siguiente Ilustración 32, podemos ver el tiempo que tardaría la aplicación en iniciarse y el tiempo que tardaría la base de datos *Firebase* en responder a una petición.

Analizando las pruebas realizadas manualmente y con los test automáticos, nos encontramos con problemas y detalles que habrá que pulir en futuras iteraciones:

- Alta latencia en la entrada de valores.
- Carga de recursos optimizable.
- Navegación entre pantallas mejorable.
- Parpadeo al navegar entre pantallas.

Todos estos problemas se podrán ir solucionando con mayor o menor facilidad en las próximas versiones. La mayoría de estos surgen por la inexperiencia al programar en *kotlin* para *Android* y *Jetpack Compose,* por lo que con el aprendizaje gradual que se obtenga con el paso del tiempo se conseguirá una aplicación mucho más fluida y limpia.

11 Resultados

En esta sección se analiza el producto final y mostraremos los resultados del proyecto *Dogizzy*.

11.1 Producto Final

El producto final se puede observar en las ilustraciones utilizadas en el apartado 9 *Implementación,* concretamente desde la Ilustración 15 hasta la Ilustración 28. Utilizaremos estas ilustraciones para simular en este apartado un recorrido de las interfaces simple que realizaría un usuario que se ha registrado recientemente.

El usuario al abrir la aplicación se encontraría con las interfaces de inicio de sesión y registro mostradas en la Ilustración 15. Al realizar estas tareas para acceder completamente a la aplicación, lo primero que se le mostraría sería la Ilustración 19, que corresponde con la interfaz principal de la aplicación. Desde esta interfaz puede acceder a los demás perfiles directamente, pero lo esperado es que configure primero su perfil antes de interactuar con los demás usuarios. Esta interfaz es la mostrada en la Ilustración 21.

En este momento, el usuario estaría editando su perfil como le plazca, rellenando los atributos que le convenga en ese instante. Utilizará las pantallas mostradas desde la Ilustración 22 hasta la Ilustración 25. Al terminar de personalizar su perfil, volvería a la interfaz principal buscando interactuar con algún perfil.

11.2 Coste temporal

Gracias a la detallada fase de análisis y especificación además del diseño, se ha contado con bastante tiempo de implementación, pudiendo solucionar los problemas que han ido surgiendo al implementar las funcionalidades, como cambiar interfaces y funcionalidades completamente conforme hemos ido desarrollando *Dogizzy* para que se ajuste más al objetivo principal de la aplicación.

En cuanto a la memoria en general, también se encuentra dentro del plazo planificado incluso con cuestiones personales que han implicado que se trabaje más ciertos días que se tenían menos horas planificadas.

12 Conclusiones y trabajo futuro

En este apartado se recogen las conclusiones extraídas a lo largo de todo el desarrollo del proyecto.

12.1 Estado actual

Actualmente *Dogizzy* se podría considerar como MVP o mínimo producto viable completado. Se han cumplido los objetivos y funcionalidades requeridas, aunque para desplegar la aplicación en plataformas como *Google Play Sto*re todavía quedaría por realizar bastante trabajo para que *Dogizzy* cumpla los estándares que necesita una aplicación para ser publicada.

La aplicación es capaz de realizar con éxito su función principal, ofrecer una plataforma con la que interactuar con otros usuarios con preferencias similares a las tuyas en torno a sus mascotas.

Por otra parte, el diseño de la aplicación se ha hecho de manera modular. De esta manera, en un futuro se podrá ampliar de forma más sencilla ya sean diferentes funcionalidades, actualizaciones de contenido o lo que se considere oportuno.

Finalmente, gracias a las herramientas de *Firebase* se puede monitorizar el comportamiento de los usuarios, así como los problemas de rendimiento y errores que vayan teniendo para solucionar las fallas a tiempo y ofrecer así la mejor experiencia de usuario posible.

12.2 Mejoras y ampliaciones

Como todo trabajo académico, siempre hay espacio para mejorar en el futuro. En cuanto a mejoras y ampliaciones de cara a trabajo futuro, durante el desarrollo del proyecto se han

encontrado con varias características que para un despliegue en *Google Play Store* necesitarían ser mejorados/implementados:

- Seguridad: habría que implementar normas de seguridad para la base de datos *Firestore Firebase* para restringir el acceso de datos a los usuarios de mejor manera. Asimismo, los mensajes de los chats deberían estar cifrados *end to end* [33] para asegurar la confidencialidad y la intimidad de los usuarios, al ser un estándar que cualquier aplicación debe tener.

- Rendimiento de la base de datos: al tener pocos datos por el momento, no se observan muchos fallos en el rendimiento de las llamadas a la base de datos, pero se podría optimizar el código utilizado para estas llamadas y utilizar caches que guarden la información del usuario para no realizar peticiones una y otra vez.

- Optimización del algoritmo de compatibilidad y filtros: el algoritmo de compatibilidad se encuentra en una fase muy beta, siendo este dependiente de la manera en la que se introduzcan los datos. Dicho esto, lo idóneo sería implementar un sistema de filtros que el usuario pueda configurar para así ordenar los perfiles que uno quiera que se muestren primero, a la vez que utilizar un sistema de ubicación por GPS que permita mostrar los perfiles cercanos independientemente de los intereses de cada usuario, etc.

- Acabado de la aplicación: el acabado de la aplicación considero que es correcto, pero siempre se podrá mejorar añadiendo animaciones, ajustando tonalidades del tema oscuro, notificaciones de chat, etc.

12.3 Nociones aprendidas

Personalmente, nunca me había enfrentado solo a un trabajo de estas características y sobre todo de estas dimensiones. Pasar de observar una problemática a diseñar una solución e implementarla no es una tarea sencilla y requieren de mucho tiempo y dedicación.

[33] *Cifrado end-to-end:* "El cifrado end-to-end (E2EE) es un proceso de comunicación seguro que evita que terceros accedan a los datos transferidos de un extremo a otro". (IBM, s.f.)

Los mayores problemas a los que me he enfrentado han sido debidos a la poca definición o haber centrado suficiente la idea. Gracias a esto he podido darme cuenta de lo importante que son las primeras fases del desarrollo ya que marcan una pequeña guía y se acaba agradeciendo mientras se va desarrollando la solución. Una vez la idea clara y definida todo funciona mucho más deprisa y se ha podido realizar todo de manera exitosa.

Gracias a la utilización de fuentes externas y librerías existentes se ha podido implementar la aplicación de una manera más sencilla, modular y con tecnología actual que se usa en el mercado que nos ayuda a incrementar nuestro CV y facilitar así mi entrada en el mundo laboral. También nos ha permitido centrarnos en la creación de una interfaz usable, características y funcionalidades que beneficien y mejoren la experiencia del usuario.

En conclusión, estoy muy satisfecho con el resultado al haber tenido momentos en el que pensaba que no iba a sacar la proyecto adelante por la cantidad de trabajo que conllevaba realizar una aplicación de este estilo, por lo que es gratificante ver que todo esfuerzo da sus frutos.

Referencias

Android Developers (s.f) Jetpack Compose. Obtenido de Android Developers: https://developer.android.com/jetpack/compose?hl=es-419

Android Developer. (22 de julio de 2019). *Launch checklist.* Obtenido de Android Developers: https://developer.android.com/distribute/best- practices/launch/launch-checklist

Android Developers. (07 de mayo de 2019). *Distribution dashboard.* Obtenido de Android Developers:https://developer.android.com/about/dashboards

Facebook. (s.f.). *React Native · A framework for building native apps using React*. Obtenido de Github: https://facebook.github.io/react-native/

Ionic. (s.f.). *Build Amazing Native Apps and Progressive Web Apps with Ionic Framework and Angular*. Obtenido de Ionic Framework: https://ionicframework.com/

JetBrains. (s.f.). *Kotlin on Android. Now official*. Obtenido de Kotlin Blog: https://blog.jetbrains.com/kotlin/2017/05/kotlin-on-android-now-official

Wikipedia. (s.f.). *Xamarin*. Obtenido de Wikipedia: https://en.wikipedia.org/wiki/Xamarin

Apple Inc. (s.f.). *Swift - Apple Developer*. Obtenido de Apple Developer: https://developer.apple.com/swift/

Firebase (s.f). Firebase Documentation. Obtenido de Firebase: https://firebase.google.com/docs?hl=es-419

Egizabal Alkorta, I. (2019). Exploratu-App para la conversión de divisas usando OCR y AR. TFG (Universidad de Alicante)

Martínez, I. (11 de agosto de 2015). *El análisis DAFO.* Obtenido de Comunicae Blog: https://blog.comunicae.es/el-analisis-dafo/

Doran, G. T. (1981). En *There's a S.M.A.R.T Way to Write Managemetn's Goals and Objectives* (págs. 35-36).

Galiana, P. (11 de junio de 2021). Qué es un análisis CAME y como se hace. Obtenido de IEBSchool: https://www.iebschool.com/blog/que-es-un-analisis-came-y-como-se-hace-marketing-digital

PFMA (s.f). Pet population 2021. Obtenido de PFMA: https://www.pfma.org.uk/pet-population-2021

Digital Guide IONOS. (s.f). El modelo en cascada: desarrollo secuencial de software. Obtenido de IONOS: https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/el-modelo-en-cascada/

Scrum Guides (s.f.) What is Scrum?. Obtenido de Scrum Guides: https://scrumguides.org/

Garrido Sotomayor, S. (9 de diciembre de 2019). Las metodologías ágiles más utilizadas y sus ventajas dentro de la empresa. Obtenido de IEBSchool: https://www.iebschool.com/blog/que-son-metodologias-agiles-agile-scrum

RealtimeBlog. (s.f.). *How to choose between Agile and Lean, Scrum and Kanban — which methodology is the best?* Obtenido de RealtimeBlog: https://realtimeboard.com/blog/choose-between-agile-lean-scrum-kanban/

Porras Blanco, M. (29 de septiembre de 2017) KPI's ¿Qué son, para qué sirven y por qué y cómo utilizarlos? Obtenido de Logicalis:

https://blog.es.logicalis.com/analytics/kpis-qu%C3%A9-son-para-qu%C3%A9-sirven-y-por-qu%C3%A9-y-c%C3%B3mo-utilizarlos

Lerena, S (3 de diciembre de 2020) Logs: qué son y por qué monitorizarlos. Obtenido de Pandorafms: https://pandorafms.com/blog/es/logs/

NeoAttacks (s.f) Drag & Drop. Obtenido de NeoAttacks: https://neoattack.com/neowiki/drag-drop/

NeoAttacks (s.f) Ventana Pop Up. Obtenido de NeoAttacks: https://neoattack.com/neowiki/pop-up/

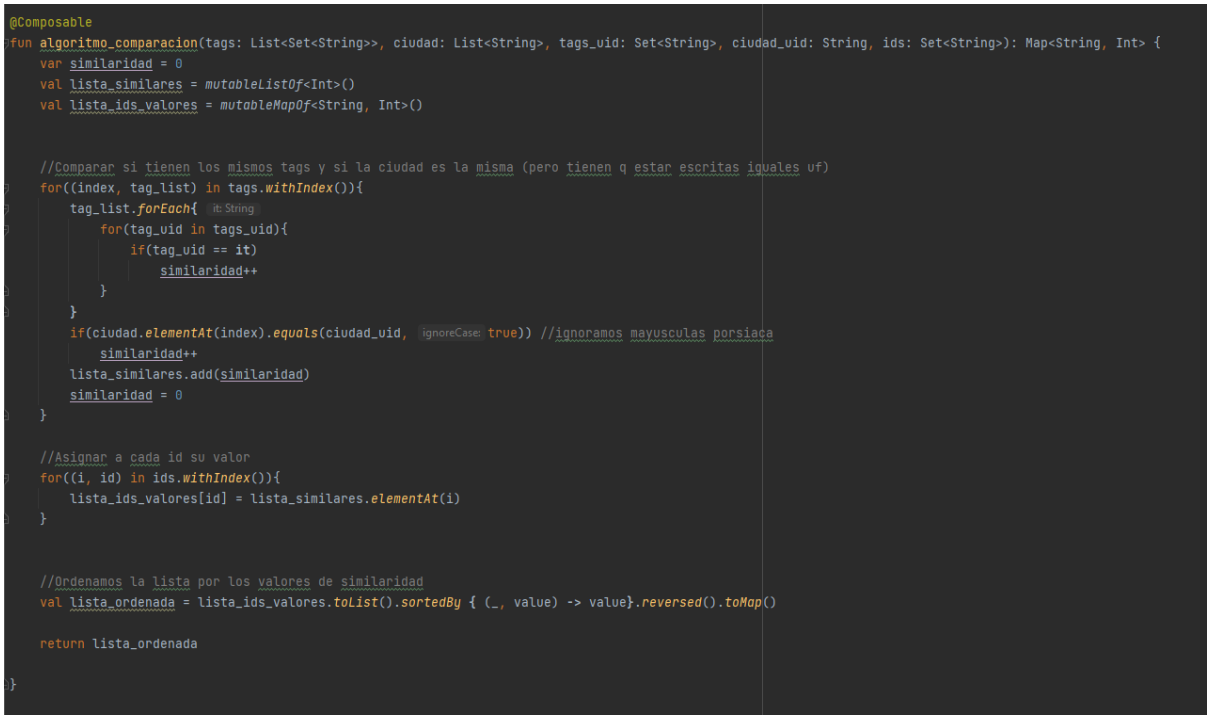
IBM (s.f) ¿Qué es el cifrado end-to-end?. Obtenido de IBM: https://www.ibm.com/es-es/topics/end-to-end-encryption

Hostgator (s.f) ¿Qué es debug en programación y para qué sirve? Obtenido de Hostagor: https://www.hostgator.mx/blog/que-es-debug-en-programacion/

Apéndice I – Algoritmo de compatibilidad de *Dogizzy*

Para cumplir con el objetivo principal de *Dogizzy* se ha creado un algoritmo de compatibilidad para los usuarios, y así mostrar en primer lugar dichos perfiles que sean más compatibles con el usuario.

A continuación, explicaremos el funcionamiento del algoritmo que se encuentra en la Ilustración 30:



```
@Composable
fun algoritmo_comparacion(tags: List<Set<String>>, ciudad: List<String>, tags_uid: Set<String>, ciudad_uid: String, ids: Set<String>): Map<String, Int> {
    var similaridad = 0
    val lista_similares = mutableListOf<Int>()
    val lista_ids_valores = mutableMapOf<String, Int>()


    //Comparar si tienen los mismos tags y si la ciudad es la misma (pero tienen q estar escritas iguales uf)
    for((index, tag_list) in tags.withIndex()){
        tag_list.forEach{ it: String
            for(tag_uid in tags_uid){
                if(tag_uid == it)
                    similaridad++
            }
        }
        if(ciudad.elementAt(index).equals(ciudad_uid, ignoreCase: true)) //ignoramos mayusculas porsiaca
            similaridad++
        lista_similares.add(similaridad)
        similaridad = 0
    }

    //Asignar a cada id su valor
    for((i, id) in ids.withIndex()){
        lista_ids_valores[id] = lista_similares.elementAt(i)
    }


    //Ordenamos la lista por los valores de similaridad
    val lista_ordenada = lista_ids_valores.toList().sortedBy { (_, value) -> value}.reversed().toMap()

    return lista_ordenada

}
```

Ilustración 33. Algoritmo de comparación de perfiles

(Fuente propia)

- Se recogen todos los perfiles de la aplicación y los datos de sus intereses y lugar de residencia.

- Se diferencia entre el usuario que está usando la aplicación y los usuarios los cuales mostraremos los perfiles en la interfaz principal.

- A continuación, comprobamos si el usuario y los demás perfiles tiene algún interés en común, además de si viven en el mismo lugar.

- Con estos datos, se crea un contador de “similaridad” que se almacena en un mapa de datos que relaciona cada perfil con su resultado en la comparación de datos.

- Por último, dicha lista se ordena de mayor compatibilidad a menor y se muestran por pantalla los perfiles en la interfaz principal.

Como se ha mencionado en el apartado de mejoras, el algoritmo está en una versión muy primeriza, al depender de datos estáticos que el usuario introduzca, en vez de datos como la

ubicación por GPS, filtros personalizados, etc. Aun así, al ser la parte principal de *Dogizzy* he optado por hacer este apéndice mostrando la lógica del algoritmo.