



TRABALLO FIN DE GRAO  
GRAO EN ENXEÑARÍA INFORMÁTICA  
MENCIÓN EN ENXEÑARÍA DO SOFTWARE



## **Aplicación Android con Flutter para la edición y publicación de imágenes**

**Estudiante:** Jose Luis Pardo López

**Dirección:** José Losada Pérez

A Coruña, junio de 2024.

*Dedicado a mi familia y amigos, quienes siempre están ahí para motivarme a seguir avanzando.*

## **Agradecimientos**

Gracias a mi tutor, José Losada, por proporcionarme el soporte necesario a lo largo de todo el desarrollo del proyecto. Por otro lado, siento también la necesidad de agradecer a todos mis compañeros de carrera, por hacer el aprendizaje más sencillo y mostrarse colaborativos desde el primer hasta el último día en la universidad.

## **Resumen**

A lo largo de la última década las redes sociales se han expandido de forma continuada hasta convertirse en parte de la vida de millones de personas. Algunas de las redes sociales más conocidas se basan en la publicación de imágenes por parte de los usuarios, lo que ha conseguido acercar a estos al mundo de la edición de imágenes.

Por ello, se plantea el desarrollo de una aplicación móvil para Android que proporcione a sus usuarios una plataforma de edición de imágenes, pero que también actúe como una red social en sí misma para poder visualizar y valorar las ediciones de otros usuarios.

Los usuarios de la aplicación tendrán que registrarse para poder utilizarla, y una vez registrados podrán navegar por las distintas publicaciones de ejemplo que ofrece la aplicación, editar y subir sus propias publicaciones y valorar las publicaciones de otros usuarios.

Las publicaciones estarán divididas en dos clases: aquellas que la propia aplicación ofrece de ejemplo y aquellas publicadas por los usuarios. Antes de publicar una imagen, se ofrecerá al usuario la posibilidad de editarla añadiéndole distintos filtros y texto, además de recortarla. Además, los usuarios podrán decidir el nivel de privacidad de cada una de sus publicaciones, lo que determinará qué usuarios podrán verlas.

También se permitirá a los usuarios establecer relaciones de seguimiento entre ellos a través de sus perfiles. Cuando un usuario decide seguir a otro, tendrá que hacerlo desde el perfil de este último. De esta manera, el usuario que realiza el seguimiento pasará a estar en la lista de amigos del segundo, quien recibirá una notificación al respecto. Esto también ayudará a definir un nivel de privacidad en el cual una publicación podrá ser vista sólo por los amigos del usuario que la publica.

La aplicación seguirá la arquitectura cliente-servidor. Los encargados de la parte servidor serán: Firebase; ofreciendo una base de datos NoSQL alojada a la nube, y Unsplash, que nos proporcionará una [API](#) con la que acceder a una enorme colección de imágenes. Por otro lado, para la implementación de la parte cliente se utilizará el lenguaje Dart junto con el framework Flutter para construir la interfaz de usuario.

Para el desarrollo de este proyecto se utilizará una versión adaptada y simplificada de Scrum, una metodología en la cual se dividirá el proyecto en pequeñas subtareas, las cuales se irán completando de manera gradual a lo largo de las distintas iteraciones. Cada iteración constará de sus propias fases de planificación, diseño, desarrollo y pruebas.

## **Abstract**

Over the past decade, social media has continuously expanded, becoming a part of millions of people's lives. Some of the most well-known social networks are based on user image posts, which has brought users closer to the world of image editing.

Therefore, the development of a mobile application for Android is proposed, providing its users with an image editing platform that also acts as a social network itself, allowing them to view and rate the edits of other users.

Users of the application will need to register to use it. After logging into the system, users will be able to browse through various example posts offered by the application, edit and upload their own posts, and rate the posts of other users.

Posts will be divided into two categories: those offered as examples by the application and those published by users. Before publishing an image, users will have the opportunity to edit it by adding various filters and text, as well as cropping it. Additionally, users can decide the privacy level of each of their posts, which will determine which users can view them.

Users will also be allowed to connect with each other through their profiles. When a user decides to connect with another, he will need to do so from the latter's profile. This way, the user sending the request will be added to the second user's friend list, and this second user will receive a notification about it. This will also help define a privacy level where a post can be seen only by the owner's friends.

The application will follow a client-server architecture. Firebase will handle the server side, providing a NoSQL database hosted in the cloud. On the other hand, the client side will be implemented using the Dart language along with the Flutter framework to build the user interface.

The development of this project will use Scrum, a methodology that divides the project into small subtasks, which will be gradually completed over different iterations or Sprints. Each Sprint will consist of its own phases of planning, design, development and testing.

### **Palabras clave:**

- Publicación.
- Amigo.
- Valoración.
- Privacidad.
- Dart.
- Firebase.

### **Keywords:**

- Publication.
- Friend.
- Rating.
- Privacy.
- Dart.
- Firebase.



# Índice general

---

<b>1</b>	<b>Introducción</b>	<b>1</b>
1.1	Contexto y motivaciones . . . . .	1
1.2	Objetivos . . . . .	2
<b>2</b>	<b>Estado del arte</b>	<b>3</b>
2.1	Alternativas clásicas . . . . .	3
2.2	Aplicaciones móviles . . . . .	4
<b>3</b>	<b>Tecnologías, lenguajes y herramientas</b>	<b>7</b>
3.1	Tecnologías y lenguajes . . . . .	7
3.1.1	Lado servidor ( <i>Back-End</i> ) . . . . .	7
3.1.2	Lado cliente ( <i>Front-End</i> ) . . . . .	9
3.2	Herramientas . . . . .	10
<b>4</b>	<b>Introducción al desarrollo</b>	<b>12</b>
4.1	Metodologías . . . . .	12
4.2	Scrum . . . . .	13
4.2.1	Roles . . . . .	13
4.2.2	<i>Sprint</i> . . . . .	15
4.2.3	Artefactos . . . . .	15
4.2.4	Integración de Scrum en el proyecto . . . . .	16
<b>5</b>	<b>Viabilidad del proyecto</b>	<b>17</b>
5.1	Viabilidad técnica . . . . .	17
5.2	Viabilidad económica . . . . .	18
5.2.1	Recursos humanos . . . . .	18
5.2.2	Recursos materiales . . . . .	19
5.2.3	Costes totales . . . . .	19

5.3	Planificación de los <i>sprints</i> . . . . .	19
5.3.1	Sprint 1 (15/01/2024 - 30/01/2024) . . . . .	20
5.3.2	Sprint 2 (31/01/2024 - 29/02/2024) . . . . .	20
5.3.3	Sprint 3 (01/03/2024 - 21/03/2024) . . . . .	20
5.3.4	Sprint 4 (22/03/2024 - 15/04/2024) . . . . .	21
5.3.5	Sprint 5 (16/04/2024 - 30/04/2024) . . . . .	21
5.3.6	Sprint 6 (01/05/2024 - 06/06/2024) . . . . .	21
5.4	Comparativa entre resultados estimados y reales . . . . .	21
<b>6</b>	<b>Análisis</b>	<b>23</b>
6.1	Actores . . . . .	23
6.2	Requisitos . . . . .	23
6.2.1	Requisitos funcionales . . . . .	23
6.2.2	Requisitos no funcionales . . . . .	25
6.3	Historias de usuario . . . . .	25
<b>7</b>	<b>Diseño</b>	<b>38</b>
7.1	Arquitectura general . . . . .	38
7.2	Diseño del lado servidor . . . . .	40
7.2.1	Firebase Authentication . . . . .	40
7.2.2	Firebase Storage . . . . .	40
7.2.3	Firestore Database . . . . .	41
7.3	Diseño del lado cliente . . . . .	43
7.3.1	Patrones y principios . . . . .	43
7.3.2	Comunicación con el lado servidor . . . . .	46
<b>8</b>	<b>Implementación y pruebas</b>	<b>49</b>
8.1	Estructura del proyecto . . . . .	49
8.1.1	Paquete <i>assets</i> . . . . .	50
8.1.2	Paquete <i>repositories</i> . . . . .	50
8.1.3	Paquete <i>models</i> . . . . .	50
8.1.4	Paquete <i>controllers</i> . . . . .	50
8.1.5	Paquete <i>util</i> . . . . .	51
8.1.6	Paquete <i>views</i> . . . . .	51
8.1.7	Archivo <i>main.dart</i> . . . . .	51
8.2	Implementación . . . . .	51
8.2.1	Modelo . . . . .	51
8.2.2	Repositorios . . . . .	53

---

8.2.3	Controladores . . . . .	56
8.2.4	Vistas . . . . .	57
8.3	Pruebas . . . . .	60
<b>9</b>	<b>Conclusiones y futuras líneas de trabajo</b>	<b>64</b>
<b>A</b>	<b>Manual de usuario</b>	<b>67</b>
A.1	Inicio de sesión y registro . . . . .	68
A.1.1	Inicio de sesión . . . . .	68
A.1.2	Registro . . . . .	69
A.1.3	Cerrar sesión . . . . .	70
A.2	Feed . . . . .	71
A.2.1	Publicaciones de usuarios seguidos . . . . .	71
A.2.2	Todas las publicaciones . . . . .	72
A.2.3	Buscar usuarios . . . . .	73
A.3	Publicaciones de ejemplo . . . . .	74
A.3.1	Listado de publicaciones . . . . .	74
A.3.2	Detalle de publicación . . . . .	75
A.4	Editor de imágenes . . . . .	76
A.4.1	Escoger imagen . . . . .	76
A.4.2	Editar imagen . . . . .	77
A.4.3	Recortar imagen . . . . .	81
A.4.4	Previsualización . . . . .	85
A.5	Perfiles de usuario . . . . .	89
A.5.1	Perfil propio . . . . .	89
A.5.2	Perfil ajeno . . . . .	98
A.5.3	Amigos . . . . .	100
<b>Lista de acrónimos</b>		<b>101</b>
<b>Glosario</b>		<b>102</b>
<b>Bibliografía</b>		<b>103</b>

# Índice de figuras

---

2.1	Interfaz de la versión inicial de <i>Adobe Photoshop</i> . . . . .	4
2.2	Interfaz de la versión inicial de <i>Microsoft Paint</i> . . . . .	4
2.3	Interfaz de <i>VSCO</i> . . . . .	5
2.4	Interfaz de <i>PicsArt</i> . . . . .	6
4.1	Roles Scrum . . . . .	14
4.2	Roles Scrum . . . . .	16
7.1	Arquitectura general del sistema . . . . .	39
7.2	Diagrama de entidades . . . . .	41
7.3	Diagrama entidad-relación . . . . .	43
7.4	Patrón de diseño Modelo-Vista-Controlador . . . . .	45
8.1	Diagrama de paquetes del proyecto en Android Studio . . . . .	50
A.1	Inicio de sesión . . . . .	68
A.2	Registro de nueva cuenta de usuario . . . . .	69
A.3	Cerrar sesión . . . . .	70
A.4	Publicaciones de usuarios seguidos y filtro por valoraciones . . . . .	71
A.5	Sección "Explorar" . . . . .	72
A.6	Buscar usuarios por nombre de usuario . . . . .	73
A.7	Listado de publicaciones de ejemplo . . . . .	74
A.8	Detalle de publicación de ejemplo . . . . .	75
A.9	Editor de imágenes . . . . .	76
A.10	Filtros . . . . .	77
A.11	Añadir y editar textos . . . . .	78
A.12	Añadir capa de color . . . . .	79
A.13	Aplicación de capa de color . . . . .	80

---

A.14 Recortar . . . . .	81
A.15 Girar imagen . . . . .	82
A.16 Ajustar escala . . . . .	83
A.17 Resultado de recorte . . . . .	84
A.18 Añadir descripción . . . . .	85
A.19 Previsualización de la publicación . . . . .	86
A.20 Añadir <i>links</i> . . . . .	87
A.21 Definir privacidad . . . . .	88
A.22 Perfil propio . . . . .	89
A.23 Editar publicación . . . . .	90
A.24 Editar perfil . . . . .	91
A.25 Configurar cuenta . . . . .	92
A.26 Cambiar contraseña . . . . .	93
A.27 Eliminar cuenta . . . . .	94
A.28 Notificaciones activas . . . . .	95
A.29 Notificaciones antiguas . . . . .	96
A.30 Notificaciones de valoración . . . . .	97
A.31 Perfil de otro usuario al que seguimos . . . . .	98
A.32 Dejar de seguir usuario . . . . .	99
A.33 Listado de amigos . . . . .	100

# Índice de tablas

---

5.1	Costes estimados de los recursos humanos . . . . .	18
5.2	Costes reales de los recursos humanos . . . . .	18
5.3	Costes reales de los recursos humanos . . . . .	19
5.4	Comparativa entre esfuerzo real y estimado de cada <i>sprint</i> . . . . .	22
6.1	HU1: Registrarse . . . . .	26
6.2	HU2: Iniciar sesión . . . . .	26
6.3	HU3: Explorar publicaciones de usuario . . . . .	27
6.4	HU4: Filtrar publicaciones de usuario . . . . .	27
6.5	HU5: Buscar perfiles . . . . .	28
6.6	HU6: Ver perfil . . . . .	28
6.7	HU7: Seguir perfil . . . . .	29
6.8	HU8: Editar perfil . . . . .	29
6.9	HU9: Editar cuenta . . . . .	29
6.10	HU10: Cambiar contraseña . . . . .	30
6.11	HU11: Eliminar cuenta . . . . .	30
6.12	HU12: Ver amigos . . . . .	30
6.13	HU13: Ver notificaciones de amistad . . . . .	31
6.14	HU14: Ver notificaciones de valoración . . . . .	31
6.15	HU15: Obtener imagen de la galería . . . . .	31
6.16	HU16: Sacar fotografía . . . . .	32
6.17	HU17: Añadir filtro a imagen . . . . .	32
6.18	HU18: Añadir texto a imagen . . . . .	32
6.19	HU19: Añadir capa de color a imagen . . . . .	33
6.20	HU20: Recortar imagen . . . . .	33
6.21	HU21: Añadir descripción a publicación . . . . .	33
6.22	HU22: Añadir links a publicación . . . . .	34

---

6.23 HU23: Definir privacidad de publicación . . . . .	34
6.24 HU24: Realizar publicación . . . . .	34
6.25 HU25: Editar descripción de publicación . . . . .	35
6.26 HU26: Editar links de publicación . . . . .	35
6.27 HU27: Modificar nivel de privacidad de publicación . . . . .	36
6.28 HU28: Eliminar publicación . . . . .	36
6.29 HU29: Valorar publicación de usuario . . . . .	36
6.30 HU30: Visualizar publicaciones de ejemplo . . . . .	37
6.31 HU31: Ordenar publicaciones de ejemplo . . . . .	37
6.32 HU32: Cerrar sesión . . . . .	37
8.1 Plan de pruebas del <i>sprint 2</i> . . . . .	63

# Capítulo 1

## Introducción

---

### 1.1 Contexto y motivaciones

En los últimos años las redes sociales se han convertido en una parte integral de la vida de gran parte de la población mundial, y su contenido tiene un impacto cada vez mayor tanto a nivel social como en el ámbito laboral.

El auge de las redes sociales, junto con el continuo crecimiento de la cantidad de información que se mueve a través de medios digitales, ha impulsado el desarrollo de muchas otras áreas, entre las que destaca el mundo de la edición de imágenes. Especialmente en Instagram, pero prácticamente en cualquier red social, cualquier creador de contenido necesita imágenes de calidad para sus publicaciones.

En este contexto, la demanda de herramientas de edición ha ido creciendo cada vez más, pues esta demanda no proviene exclusivamente de profesionales de la fotografía, sino que también los usuarios cotidianos de las redes sociales buscan tener perfiles con una estética cuidada. Aunque pueda debatirse sobre las causas y consecuencias de estas nuevas necesidades entre la población y especialmente entre los jóvenes, es innegable que han influido directamente en el avance de numerosos sectores que complementan el uso de las plataformas online. En definitiva, las aplicaciones de edición de imágenes se han convertido, prácticamente, en un requisito básico para todos los teléfonos.

Una aplicación móvil enfocada en un público general, inexperto en fotografía y acostumbrado a las redes sociales, proporcionaría a sus usuarios una zona de pruebas perfecta para aprender a crear un perfil con *engagement* a través de la publicación de imágenes que encajen en los estándares de este tipo de plataformas. La edición de imágenes, su publicación y, sobre todo, la interacción con otros perfiles, supondrían las bases necesarias para el aprendizaje autónomo del usuario.

En este trabajo intentan abordarse todas estas necesidades para culminar en el diseño de una aplicación móvil con las características mencionadas, que busca ofrecer el funcionamiento

de una red social básica pero consistente, centrada en las imágenes.

La idea proviene del alumno, pues en su entorno no son pocas las personas dedicadas a crear contenido de forma profesional o semiprofesional. Por ello, conoce de primera mano el funcionamiento de estos estándares no escritos que surgen en las redes sociales y de la necesidad de un período de aprendizaje cuando se busca crear un perfil con proyección.

## 1.2 Objetivos

La meta principal de este proyecto es proporcionar a todas las personas que creen contenido fotográfico o para redes sociales (o que quieran aprender) una aplicación móvil que suponga una red social alternativa para ellos. La aplicación busca ofrecer una herramienta en la que puedan editar imágenes, publicarlas, navegar por otras publicaciones e interaccionar con otros usuarios, todo ello de forma sencilla e intuitiva. Los objetivos a la hora de implementar la aplicación son los siguientes:

- Gestión de los usuarios.
- Gestión de las publicaciones de los usuarios.
- Visualización de publicaciones de ejemplo ofrecidas por la propia aplicación.
- Gestión de las valoraciones sobre las publicaciones de los usuarios.
- Gestión de los amigos de los usuarios.
- Edición de imágenes.

Más allá de los requisitos funcionales, se busca crear una comunidad de creadores de contenido, fomentar la interacción respetuosa entre ellos y facilitar el aprendizaje para aquellos que estén iniciándose.

## Capítulo 2

# Estado del arte

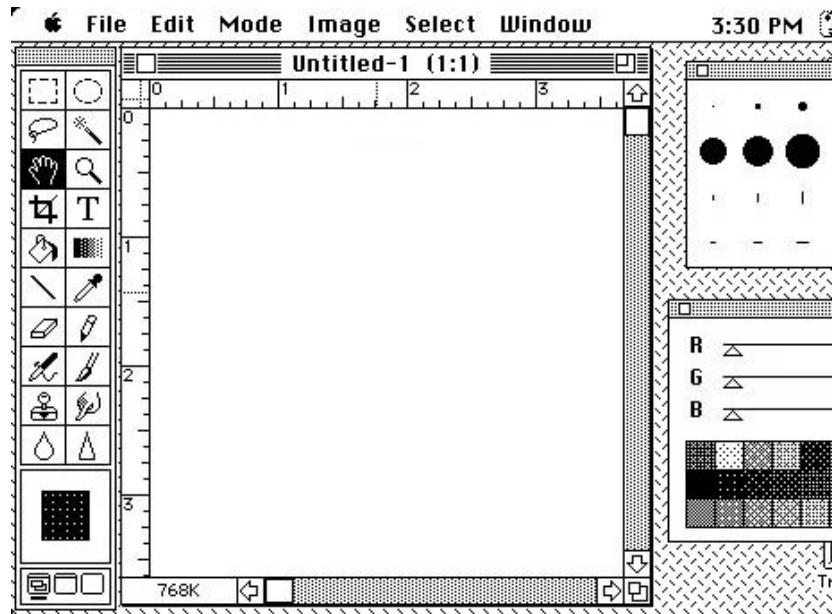
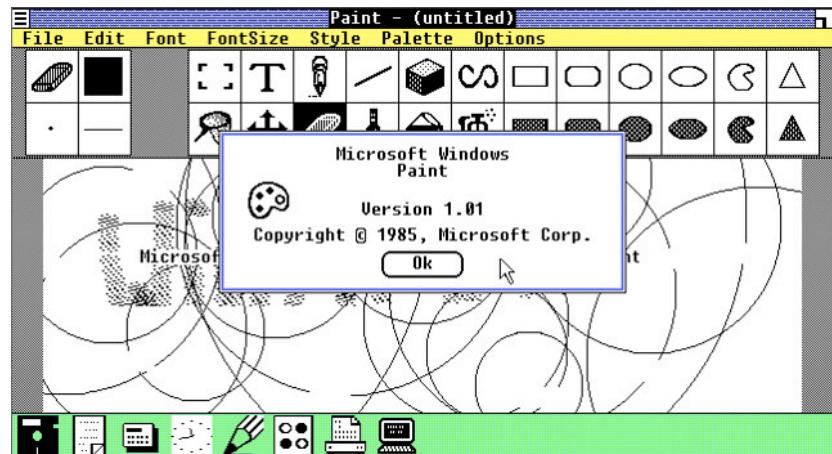
---

Aunque en la actualidad la edición de imágenes está al alcance de todos, no hace mucho tiempo estaba reservada casi exclusivamente para profesionales. La transformación se ha producido como consecuencia de la creciente necesidad de generar contenido visual atractivo para redes sociales, pues esto ha provocado la aparición de aplicaciones y programas sencillos que han democratizado el acceso a las herramientas de edición.

### 2.1 Alternativas clásicas

Antes de que el primer *smartphone* saliera al mercado ya se habían lanzado programas de edición de imágenes para ordenador como *Microsoft Paint* (1985) [1] y *Adobe Photoshop* (1990) [2]. Ambos programas continúan utilizándose a día de hoy y se encuentran entre los más conocidos en su ámbito, pero no están tan centrados en ofrecer una solución sencilla para creadores de contenido para redes sociales.

Estos programas, aunque también tienen algunas opciones básicas de edición, fueron diseñados principalmente para profesionales o aficionados con un nivel avanzado de conocimientos técnicos. No buscan tener una interfaz completamente intuitiva, sino proporcionar herramientas precisas, complejas y especializadas que requieren un tiempo significativo de aprendizaje y, sobre todo, de práctica. A continuación, en las figuras 2.1 y 2.2 se muestran las interfaces de usuario de *Photoshop* y *Paint* en sus versiones originales, respectivamente.

Figura 2.1: Interfaz de la versión inicial de *Adobe Photoshop*Figura 2.2: Interfaz de la versión inicial de *Microsoft Paint*

## 2.2 Aplicaciones móviles

Actualmente, existen aplicaciones móviles de edición que se centran en un público más general y que proporcionan una red social en sí mismas, entre las que destacan VSCO [3] y *PicsArt* [4].

VSCO, que se enfoca en imágenes fotográficas, ofrece una amplia variedad de filtros y herramientas de edición como el ajuste de la saturación, la aplicación de difuminado, la crea-

ción de collages, etc. Otra de sus características más destacables consiste en las sugerencias personalizadas (basadas en nuestra actividad reciente dentro de la aplicación) que nos hace al navegar por las publicaciones de otros usuarios. El principal problema de VSCO es que se obliga al usuario a pagar por utilizar la mayoría de sus filtros y herramientas, lo que la hace menos accesible para el usuario promedio. En la figura 2.3 se muestra parte de la interfaz de usuario de la aplicación.

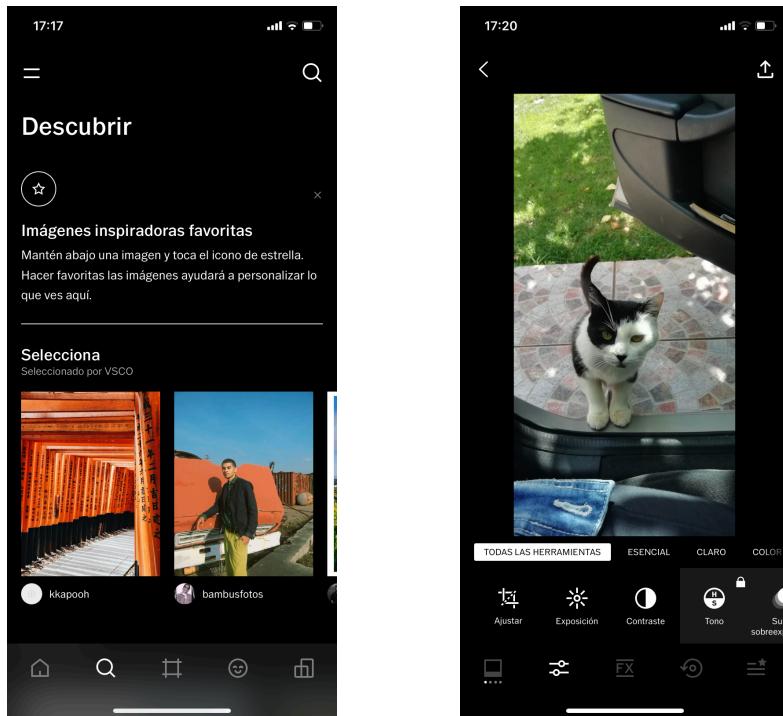


Figura 2.3: Interfaz de VSCO

Las funcionalidades que ofrece *PicsArt* son bastante similares a las de VSCO, pero además nos da la posibilidad de usar inteligencia artificial para editar nuestras imágenes (por ejemplo, para eliminar o añadir fondos). Se trata de una herramienta muy atractiva teniendo en cuenta que el mundo se encuentra en pleno auge de la IA, pero también provoca que se consuman muchos recursos de nuestro dispositivo. Para utilizar *PicsArt* necesitaremos comprar una suscripción, lo que también aleja la aplicación del público más general, aunque una vez dentro no tendremos que pagar para desbloquear ninguna herramienta o filtro. En la figura 2.4 se puede ver parte de la interfaz de usuario de la aplicación.

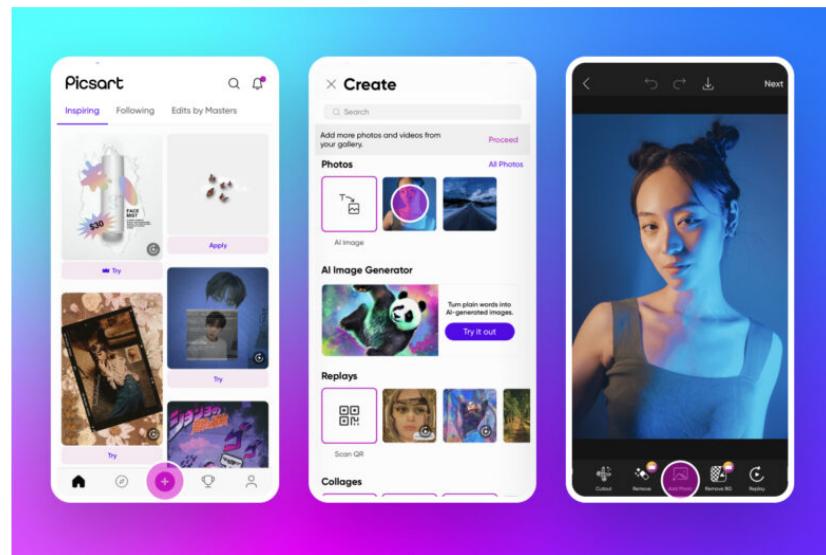


Figura 2.4: Interfaz de *PicsArt*

Tras analizar las diferentes opciones se llega a la conclusión de que es necesario implementar una solución que evite al usuario la problemática de tener que pagar una suscripción o comprar por separado diferentes herramientas de edición.

## Capítulo 3

# Tecnologías, lenguajes y herramientas

---

EN este capítulo se detallan las distintas tecnologías y herramientas utilizadas en el desarrollo del proyecto, además de los lenguajes de programación empleados para cada subsistema del software construido.

## 3.1 Tecnologías y lenguajes

A continuación se enumeran las tecnologías y lenguajes empleadas en el proyecto divididas en dos apartados: lado servidor y lado cliente.

### 3.1.1 Lado servidor (*Back-End*)

Para el *backend* de la aplicación se emplean dos grupos principales de tecnologías. Uno de ellos está compuesto por las herramientas de la plataforma Firebase, destinadas a la creación de una base de datos segura y con una interfaz sencilla. Gracias a esta plataforma podrán almacenarse todos los datos relativos a los usuarios incluyendo sus publicaciones, notificaciones, valoraciones y amistades.

Por otra parte, para la obtención y el manejo de las imágenes que la aplicación ofrecerá como ejemplo, se emplea la [API](#) destinada exclusivamente al mundo de la fotografía que nos ofrece la plataforma Unsplash.

#### Firestore Database

Firestore Database [5] es la base de datos NoSQL que nos ofrece Google a través de su plataforma Firebase. Se enfoca hacia el desarrollo de aplicaciones móviles y web, pues proporciona un servicio en la nube que permite a sus usuarios almacenar y sincronizar datos en

tiempo real.

Esta base de datos está basada en documentos [JSON](#), organizados a su vez en colecciones. Cada documento (que equivaldría a una fila en una base de datos relacional) contiene datos en forma de pares clave-valor, pero no es necesario que todos los documentos de una colección tengan los mismos campos. Las colecciones son simples contenedores de documentos, y un documento puede contener subcolecciones, facilitando así la creación de estructuras jerárquicas. Otra característica a destacar de Firestore es la posibilidad de definir reglas de seguridad para controlar el acceso a los datos por parte de los usuarios.

Se ha escogido Firestore Database debido principalmente a tres factores: la sencillez de su uso, la gran escalabilidad que proporciona y la integración con otros servicios de Firebase, los cuales facilitarán enormemente el desarrollo del lado servidor.

### **Firebase Authentication**

Firebase Authentication [6] es una herramienta que se encarga de proporcionar servicios de *backend*, [SDK](#) fáciles de utilizar y bibliotecas de interfaz de usuario, todo ello destinado a la autenticación de los usuarios en la aplicación. Se integra con otras herramientas de Firebase, especialmente a la hora de definir reglas de seguridad, pues podremos tener en cuenta el estado de la autenticación del usuario a la hora de decidir si tiene acceso o no a determinados datos.

Permite la autenticación mediante correo electrónico y contraseña, autenticación telefónica y el acceso a través de proveedores de identidad federados como Google, Twitter, Facebook y GitHub. En la aplicación podremos integrar uno o varios de estos métodos de inicio de sesión. También proporciona otros servicios como el envío automático de correos de verificación de cuenta o de verificación de cambio de correo electrónico, el restablecimiento de la contraseña o el envío de códigos de verificación vía [SMS](#).

Se ha decidido emplear Firebase Authentication en base a la simplificación que hace de todo el proceso de autenticación de usuarios y a la hora de definir reglas de acceso a nuestra base de datos.

### **Firebase Storage**

Firebase Storage [7] se trata del servicio de almacenamiento en la nube de la plataforma Firebase. En él se pueden configurar sus propias reglas de acceso, y su diseño se enfoca en almacenar y proporcionar contenido generado por los usuarios como imágenes, videos o cualquier otro tipo de archivo de gran tamaño. Además, utiliza contenedores de objetos o *buckets*, que pueden ser archivos individuales o directorios. Los archivos individuales reciben el nombre de "objetos", y están identificados de manera única por la ruta que ocupan dentro del

*bucket*. Es posible utilizar la referencia que se genera hacia una ruta para subir, descargar o modificar su contenido.

En definitiva, se trata de una estructura jerárquica de archivos en la que hay un contenedores superior que contiene todos los demás contenedores y objetos.

La decisión de emplear Firebase Storage se debe, además de a su seguridad y a su eficiencia, a su capacidad de integración con el resto de herramientas de Firebase empleadas en el proyecto. Gracias a Authentication es posible fortalecer las reglas de acceso, y a través de los datos en Firestore Database los usuarios de la aplicación podrán acceder a los archivos almacenados en Storage.

### API Unsplash

Esta [API](#) permite a los desarrolladores acceder a la enorme colección de fotografías de la plataforma Unsplash [8]. Esta colección ha sido creada y es actualizada gracias a la contribución de fotógrafos de todo el mundo, quienes envían sus fotografías a la plataforma de forma gratuita.

La [API](#) ofrece varios *endpoints* a los que se pueden realizar distintos tipos de peticiones [HTTP](#) con varios parámetros. Tras realizar una petición a la [API](#) esta enviará una respuesta en formato [JSON](#) con una colección de imágenes. Cada imagen incluirá entre sus datos la [URL](#) a través de la cual podremos acceder a su visualización.

Para la autenticación la [API](#) utiliza OAuth 2.0. Los desarrolladores tendrán que registrarse en Unsplash para obtener una clave de acceso a la [API](#) y una clave secreta, pues serán necesarias para realizar las peticiones a los distintos *endpoints*.

Los motivos por los cuales se ha decidido emplear la Unsplash son la gran calidad de sus imágenes y, sobre todo, la sencillez en el uso de la [API](#).

#### 3.1.2 Lado cliente (*Front-End*)

Para el desarrollo del subsistema cliente de la aplicación se ha empleado el lenguaje de programación Dart junto con el [framework](#) Flutter.

##### Dart

Dart [9] se trata de un lenguaje de programación desarrollado por Google y diseñado para crear aplicaciones (web, móviles o escritorio) de manera ágil. Dart es un lenguaje orientado a objetos que emplea el tipado estático aunque también soporta el tipado dinámico para una mayor flexibilidad. El tipado estático permite conocer el tipo de las variables ya en tiempo de compilación, lo que se traduce en una mayor capacidad de detección de errores sin necesidad de ejecutar la aplicación y un autocompletado más preciso.

Dart incluye un gestor de paquetes llamado "Pub", que proporciona las herramientas necesarias para gestionar las dependencias de un proyecto Dart y administrar los paquetes utilizados en él y sus versiones. A través de los archivos pubspec.yaml y pubspec.lock que se generan en un proyecto Dart se pueden definir las dependencias del proyecto y sus versiones, respectivamente.

La elección de emplear Dart se basa, principalmente, en que es el único lenguaje en el que el autor del proyecto tiene experiencia para el desarrollo de aplicaciones Android, pero también en la gran colección de librerías que se pueden encontrar en el repositorio oficial de Dart (pub.dev) y a su buena documentación.

### Flutter

Flutter [10], desarrollado también por Google, supone un *framework* de código abierto para el desarrollo de interfaces de usuario. Aunque se usa principalmente para el desarrollo de aplicaciones móviles, actualmente también es compatible con el desarrollo de aplicaciones web y de escritorio para diferentes sistemas operativos, por lo que es muy útil a la hora de desarrollar aplicaciones multiplataforma.

Flutter utiliza el lenguaje Dart, que está optimizado para la creación de imágenes de usuario con este *framework*, motivo por el cual se ha incluido en el proyecto.

### HTTPS

HTTP es un protocolo de comunicación para el intercambio de información entre cliente y servidor. HTTPS [11], por su parte, es la versión segura de HTTP, pues garantiza que los datos transmitidos estén encriptados y protegidos frente amenazas externas. El intercambio de recursos o peticiones entre cliente y servidor consiste en el envío de un mensaje de solicitud por parte del cliente y el posterior envío de un mensaje de respuesta por parte del servidor.

El uso de HTTPS en el proyecto es necesario para enviar las peticiones a la API Unsplash. Además, la comunicación con Firebase también se produce a través de este protocolo de forma interna.

## 3.2 Herramientas

### Android Studio

Android Studio [12] es un IDE basado en IntelliJ IDEA (IDE desarrollado por JetBrains), especializado en proporcionar un entorno para el desarrollo de aplicaciones Android. La principal ventaja de Android Studio frente a otros IDEs es la facilidad a la hora de probar el software durante el proceso de desarrollo. A través de este IDE se puede ejecutar un emulador

de cualquier dispositivo Android, en el cual se puede instalar y utilizar la aplicación. Con la opción *hot reload* nos da la posibilidad de visualizar en la *app* los cambios realizados en el código sin necesidad de reiniciarla.

Otro de los puntos fuertes de Android Studio es la posibilidad de activar el modo depuración para la ejecución de la aplicación, que nos permite aislar con facilidad en el código los *bugs* que van apareciendo.

### Google Chrome

En el proyecto se ha empleado Chrome [13] para poder probar las peticiones a la API de Unsplash. A pesar de que existen otras herramientas dedicadas exclusivamente al lanzamiento de peticiones HTTP se ha decidido usar Chrome debido a la facilidad con la que operan los endpoint de Unsplash. Además, las herramientas para desarrolladores que nos ofrece Chrome nos permiten una visualización cómoda de las respuestas junto a todos los datos que se necesitan para evaluar los resultados de las pruebas.

### Git

Git [14] es el sistema de control de versiones más conocido, empleado habitualmente en proyectos colaborativos de desarrollo software. Se trata de un sistema de control de versiones distribuido, lo que significa que un clon local del proyecto es un repositorio de control de versiones completo. Estos repositorios locales permiten trabajar sin conexión y facilitan el trabajo en paralelo. Primero, los desarrolladores confirman su trabajo localmente y, a continuación, sincronizan la copia del repositorio con la del servidor.

Se emplea en el proyecto para llevar un control de versiones de la parte *front-end*.

### Overleaf

Overleaf [15], una plataforma en línea para la creación de documentos escritos en L<sup>A</sup>T<sub>E</sub>X, es la herramienta utilizada para la redacción de esta memoria.

## Capítulo 4

# Introducción al desarrollo

---

Se propone para este proyecto la creación de una aplicación Android para la edición y publicación de imágenes por parte de los usuarios. Antes de empezar con el desarrollo del proyecto en sí mismo se debe definir un objetivo claro y establecer una serie de pautas que permitan alcanzarlo. Para ello existen las distintas metodologías de desarrollo software.

Dentro de este ámbito, una metodología puede definirse de manera amplia como el conjunto de técnicas y métodos que se emplean para diseñar una solución de software informático. Estas técnicas y métodos se aplican a lo largo de todo el proceso de gestión y ejecución del proyecto.

### 4.1 Metodologías

Las metodologías [16] de desarrollo software pueden dividirse en dos vertientes: las metodologías ágiles y las no ágiles. A la hora de decidir cuál es la mejor opción a aplicar en nuestro proyecto se deben tener en cuenta las características de una y otra:

- **Metodologías no ágiles:**

- El proyecto se divide en fases. Es necesario terminar y validar cada una de las fases para poder pasar a la siguiente.
- Inicialmente se hace una toma de requisitos del proyecto y se establece una secuencia de trabajo. Es importante iniciar con un riguroso proceso de captura de requisitos, análisis y diseño.
- La estimación se realiza una única vez y al inicio del proyecto, por lo que tendrá mucha importancia.
- No se esperan modificaciones en los requisitos una vez iniciado el proceso de desarrollo.

- **Metodologías ágiles:**

- También se divide el proyecto en fases, pero en este caso cada fase se divide en fases aún más pequeñas llamadas *sprints* o iteraciones. Cada una de estas iteraciones se puede completar de manera independiente a la fase en la que se encuentra y nos permite comenzar con otras fases, aunque la suya propia no esté finalizada por completo.
- Distintos equipos de trabajo pueden trabajar a la vez en distintas fases del proyecto y se puede ir iterando entre los distintos *sprints*.
- Son metodologías adaptativas, de forma que los requisitos y la planificación del proyecto pueden ir evolucionando a lo largo del desarrollo del mismo.

Para este proyecto se ha optado por el uso una metodología ágil, específicamente *Scrum*, debido a la necesidad de monitorizar de forma periódica el estado de la aplicación en desarrollo. Dado que el equipo consta únicamente de dos integrantes se harán algunas adaptaciones en la metodología para ajustarla a la situación.

## 4.2 Scrum

*Scrum* [17] constituye un marco de trabajo en el cual se realizan entregas parciales y regulares del producto, priorizadas por el beneficio que aportan al receptor del proyecto. Estas entregas incrementales aportan una mayor adaptabilidad al proyecto, por lo que es muy útil emplear *Scrum* en proyectos en los que los requisitos son cambiantes o están poco definidos.

### 4.2.1 Roles

En un proyecto en el que se aplica *Scrum* se deben distinguir tres roles [18]: el *Scrum Master*, el *Product Owner* y el equipo de desarrollo.

#### Scrum Master

El *Scrum Master* es el encargado de gestionar todo el proceso *Scrum* y de asegurar que se lleva a cabo de acuerdo con lo estipulado. También tiene un papel importante en la eliminación de los constantes impedimentos que puedan surgir en la organización del proyecto. Se ocupa de las labores de formación del equipo y de facilitar la comunicación entre los distintos miembros del mismo.

### Product Owner

El Product Owner es quien debe optimizar y maximizar el valor del producto a través de la gestión del Product Backlog, además de comunicar al resto del equipo las peticiones y requerimientos del cliente. El Product Backlog es la lista de características que debe incluir el producto, y el Product Owner debe elaborarla y mantenerla de forma estructurada, detallada y priorizada; por lo que tendrá potestad para controlar la organización de los *sprints* del proyecto. Cada elemento del Product Backlog debe estar definido en una historia de usuario.

### Equipo de desarrollo

Los integrantes de este equipo son los encargados de desarrollar el producto de manera autogestionada para conseguir entregar al final de cada *sprint* el software con las nuevas funcionalidades. Este producto entregable se denomina incremento. Para crear el incremento de un *sprint* se emplean los elementos seleccionados del Product Backlog para el *sprint* en cuestión, que reciben el nombre de Sprint Backlog.

En la figura 4.1 se muestra cómo se comunican los distintos roles de un proyecto Scrum.

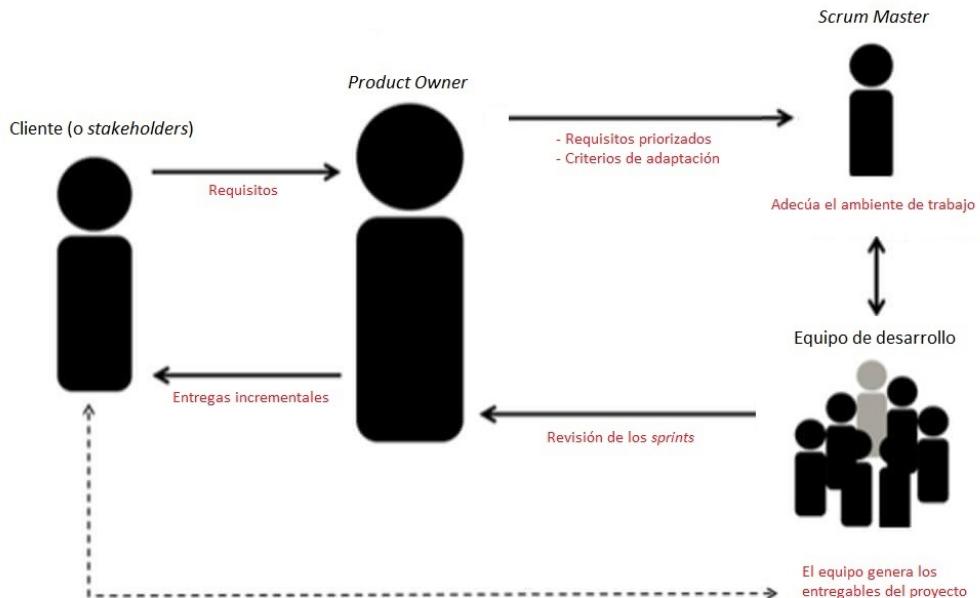


Figura 4.1: Roles Scrum

### 4.2.2 Sprint

Se llama *sprint* [19] a cada uno de los ciclos o iteraciones de desarrollo que se van a producir en nuestro proyecto Scrum. La duración habitual de un *sprint* suele ser de entre una y seis semanas, y en él se debe conseguir un software entregable o incremento que aporte valor al cliente. El incremento de un *sprint* supondrá una nueva versión del producto que incluirá todas las características contenidas en el Sprint Backlog, decididas por el Product Owner. Cada sprint se divide en cuatro fases:

- **Planificación del sprint o Sprint Planning**, en la que se definen las funcionalidades y características que debe incluir el incremento y la prioridad de las mismas.
- **Fase de desarrollo**, en la cual se implementan estas funcionalidades. Para dar apoyo al desarrollo y conocer el avance en todo momento, diariamente se realiza una breve reunión entre todos los miembros del equipo conocida como Scrum Diario.
- **Revisión del sprint**, que se produce al final del mismo con el objetivo de corroborar la correcta entrega del incremento.
- **Retrospectiva del sprint**, que sirve para valorar el desempeño del equipo a lo largo del ciclo y detectar posibles mejoras a aplicar en el siguiente.

### 4.2.3 Artefactos

En este ámbito se denomina "artefactos" a aquellos elementos físicos que se producen como resultado de la aplicación de Scrum, y que están destinados a aportar valor o transparencia al trabajo realizado en el proyecto. Los tres artefactos que se generan ya han sido mencionados y definidos previamente: el Product Backlog, los Sprint Backlog y los incrementos.

A continuación, en la figura 4.2 se muestra el flujo habitual de los procesos de Scrum, incluyendo en él estos artefactos.

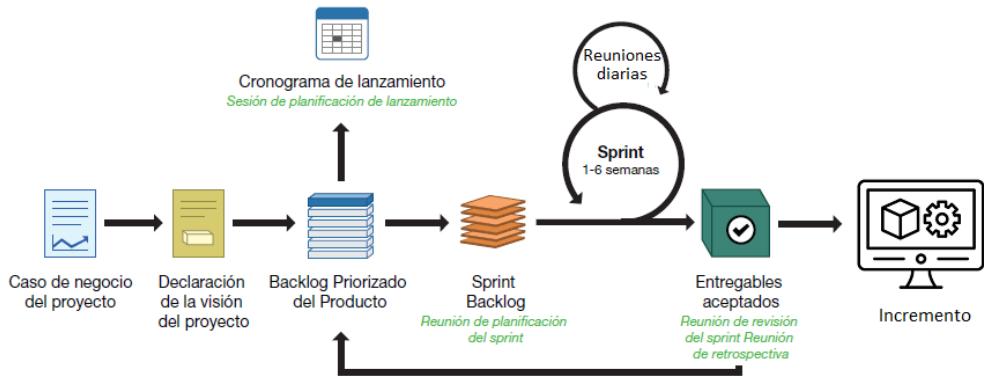


Figura 4.2: Roles Scrum

#### 4.2.4 Integración de Scrum en el proyecto

Dadas las dimensiones del equipo, compuesto exclusivamente por el autor y el tutor del proyecto, es necesario ajustar la metodología Scrum para obtener una versión simplificada que nos permita trabajar eficientemente. El autor desempeñará tanto el rol de equipo de desarrollo como el de Product Owner, ocupándose de la definición, priorización e implementación de los elementos del Product Backlog. El tutor del proyecto, por otro lado, será quien ejerza de Scrum Master, pues será el encargado de asegurar el cumplimiento de los principios de la metodología ágil a lo largo del desarrollo del proyecto.

Se ha decidido aumentar la periodicidad de las reuniones diarias, pues su importancia y necesidad se ven reducidas como consecuencia de que el Product Owner constituya también el equipo de desarrollo. Con esta modificación se pretende evitar entorpecer el proceso de desarrollo, pero sí que se establecerán reuniones de forma regular en cada *sprint* para que todos los miembros del proyecto conozcan el avance y puedan identificar posibles mejoras en el proceso.

## Capítulo 5

# Viabilidad del proyecto

---

ANTES de iniciar con el desarrollo del proyecto es necesario estimar y estudiar los costes económicos y técnicos de su realización. Gracias a este estudio se podrá determinar si el proyecto es viable en distintos aspectos, además de que colaborará con la toma de decisiones a lo largo del desarrollo.

### 5.1 Viabilidad técnica

Las herramientas y tecnologías empleadas en este proyecto han sido seleccionadas teniendo en cuenta la capacidad de integración entre ellas y con el entorno de desarrollo, que en este caso será el ordenador del alumno. Gran parte de las tecnologías del proyecto emplean el procesamiento en la nube, por lo que no requieren un gran consumo de requisitos en el ordenador. Por otra parte, el lanzamiento de peticiones HTTPS, el uso de Google Chrome (siempre que el número de pestañas abiertas no sea excesivo) y las solicitudes a un repositorio Git tampoco suponen una carga de trabajo ni hacen un alto consumo de la RAM del equipo. En cambio, la ejecución de un emulador de un dispositivo móvil Android a través de Android Studio sí puede suponer una traba si el entorno no es lo suficientemente potente. Tras realizar pruebas para las distintas herramientas se llegó a la conclusión de que el entorno de desarrollo tiene capacidad para emplear todas ellas de manera fluida.

También ha sido necesario analizar la capacidad del autor del proyecto para realizar su trabajo asignado. Las soluciones aportadas por su parte para el desarrollo de los distintos subsistemas de la aplicación han sido acordes a su nivel de formación y no extremadamente pretenciosas. Además, el autor cuenta con cierto conocimiento previo en tecnologías de desarrollo móvil (tanto iOS como Android), por lo que habría adquirido previamente determinada capacidad para implementar estas soluciones de manera precisa y anticipar y atajar los distintos problemas que fueran surgiendo.

## 5.2 Viabilidad económica

A continuación, se analizan por separado los análisis de costes estimados de los recursos humanos y materiales y se comparan con los costes reales finales. No se tienen en cuenta los recursos técnicos, ya que las herramientas y tecnologías empleadas son completamente gratuitas.

### 5.2.1 Recursos humanos

El equipo se compone de dos integrantes: el tutor del proyecto, que ocupa el rol de Scrum Master, y el autor, que se encarga de las tareas del Product Owner y del equipo de desarrollo simultáneamente. El autor del proyecto es un programador *junior*, por lo que para estimar los costes totales de los trabajadores se toman los sueldos medios en España de un programador *junior* [20] y de un jefe de proyecto [21], además del porcentaje de impuestos a pagar por cada trabajador en nómina. En la tabla 5.2 se pueden ver los costes estimados totales de los recursos humanos para el proyecto, mientras que en la tabla 5.1 se pueden ver los costes reales tras la finalización del proyecto.

	Coste medio	Trabajo estimado	Coste estimado
<i>Scrum master</i>	28,27€/hora	28 horas	805,56€
<i>Programador junior</i>	10,77€/hora	400 horas	4308€
<b>Total</b>			<b>5113,56€</b>
<b>Total con impuestos</b>			<b>6647,63€</b>

Tabla 5.1: Costes estimados de los recursos humanos

	Coste medio	Trabajo real	Coste real
<i>Scrum master</i>	28,27€/hora	28 horas	805,56€
<i>Programador junior</i>	10,77€/hora	430 horas	4631,10€
<b>Total</b>			<b>5436,66€</b>
<b>Total con impuestos</b>			<b>7067,66€</b>

Tabla 5.2: Costes reales de los recursos humanos

Como podemos comprobar, los costes finales fueron **420,03€** mayores de lo esperado. Esto se debe a que fueron necesarias más horas de trabajo de las estimadas para el desarrollo de

determinados *sprints* del proyecto. Por otro lado, se cumplió la estimación de trabajo del Scrum Master, pues no fue necesaria su intervención más allá de la organización incial del proyecto y las reuniones acordadas.

### 5.2.2 Recursos materiales

Para la realización del proyecto son necesarios exclusivamente dos elementos: un ordenador en el cual realizar la implementación y la documentación, y un teléfono móvil el cual instalar y probar la aplicación. Como aproximación, se toman los valores medios de un teléfono móvil y un ordenador portátil para determinar los costes finales:

- Valor medio de un teléfono móvil: **400€**
- Valor medio de un ordenador portátil: **750€**

Por ende, se puede determinar que los costes de los recursos materiales empleados en el proyecto son de **1150€**.

### 5.2.3 Costes totales

Tras haber analizado los costes de los distintos tipos de recursos, se puede ver en la tabla 5.3 la comparativa final entre los costes estimados previos a la realización del proyecto y los costes reales finales.

	<b>Coste estimado</b>	<b>Coste real</b>
<i>Recursos humanos</i>	6647,63€	7067,66€
<i>Recursos materiales</i>	1150€	1150€
<b>Total</b>	<b>7797,63</b>	<b>8217,66€</b>

Tabla 5.3: Costes reales de los recursos humanos

Como conclusión del análisis realizado se obtiene que los costes finales superaron en **420,03€** a los estimados, lo cual supone un aumento del **5,39%**. Este aumento, aunque es significativo, no supuso un impedimento a la hora de alcanzar los objetivos definidos para el proyecto.

## 5.3 Planificación de los sprints

Para este proyecto se han planificado los *sprints* de manera que en cada uno de ellos se complete la implementación de un módulo del sistema. La duración de los *sprints* oscilará

entre las 1 y 4 semanas, en función de la complejidad del desarrollo de las funcionalidades incluidas en el Sprint Backlog. A continuación se muestra la planificación de los *sprints* junto a las tareas a completar en cada uno.

#### 5.3.1 Sprint 1 (15/01/2024 - 30/01/2024)

A lo largo de este *sprint* inicial se atravesará la etapa de análisis de requisitos y la configuración inicial del entorno de desarrollo (incluyendo la configuración de la base de datos). Esto incluye:

- Análisis de las funcionalidades del sistema.
- Creación y configuración del proyecto Flutter en Android Studio.
- Creación y configuración del proyecto en Git.
- Creación de una cuenta en Unsplash y obtención de las claves para el lanzamiento de peticiones a su API.
- Creación y configuración del proyecto en Firebase, incluyendo Firestore Database, Storage y Authentication.
- Diseño e implementación de la base de datos en los servicios de Firebase.

#### 5.3.2 Sprint 2 (31/01/2024 - 29/02/2024)

En este *sprint* se implementará el módulo de gestión de usuarios, que se divide en las siguientes tareas:

- Implementación del módulo de gestión de usuarios en el *front-end* incluyendo inicio de sesión, configuración de cuenta y visualización de perfiles.
- Realización de pruebas del módulo.

#### 5.3.3 Sprint 3 (01/03/2024 - 21/03/2024)

En este *sprint* se implementará el módulo de edición de imágenes, dividido en las siguientes tareas:

- Implementación del módulo de gestión de edición de imágenes en el *front-end*.
- Realización de pruebas del módulo.

### 5.3.4 Sprint 4 (22/03/2024 - 15/04/2024)

En este *sprint* se implementará el módulo de gestión de imágenes de usuario, dividido en las siguientes tareas:

- Implementación del módulo de gestión imágenes de usuario en el *front-end*, incluyendo publicar, visualización de publicaciones y valoraciones.
- Realización de pruebas del módulo.

### 5.3.5 Sprint 5 (16/04/2024 - 30/04/2024)

En este *sprint* se implementará el módulo de gestión de imágenes de ejemplo, dividido en las siguientes tareas:

- Implementación del módulo de gestión de las imágenes de ejemplo que ofrece la aplicación.
- Realización de pruebas del módulo.
- Revisión general de la aplicación.

### 5.3.6 Sprint 6 (01/05/2024 - 06/06/2024)

En el *sprint* final, que también será el de mayor duración, se llevará a cabo la redacción de la memoria.

## 5.4 Comparativa entre resultados estimados y reales

Tras decidir cómo se divide el Product Backlog entre los distintos *sprints*, se ha estimado el esfuerzo en horas necesario para completar las tareas de cada uno de ellos. En la tabla 5.4 se puede ver la comparativa para cada *sprint* entre su esfuerzo estimado y el esfuerzo que finalmente fue necesario.

Finalmente, el proyecto requirió **430 horas** de trabajo por parte del equipo de desarrollo, lo que supera en 30 horas lo previsto. Dado que se ha utilizado una aproximación de unas 3 horas de trabajo diarias para dar una duración determinada a los *sprints*, esto supone un retraso de 10 días en la finalización del proyecto. Aunque es una diferencia notable, no supuso un problema a la hora de coordinar el trabajo junto con el Scrum Master ni de completar cada tarea dentro de su *sprint* asignado inicialmente.

	Esfuerzo estimado	Esfuerzo real
<i>Sprint 1</i>	50 horas	40 horas
<i>Sprint 2</i>	80 horas	90 horas
<i>Sprint 3</i>	70 horas	80 horas
<i>Sprint 4</i>	70 horas	70 horas
<i>Sprint 5</i>	40 horas	50 horas
<i>Sprint 6</i>	90 horas	100 horas
<b>Total</b>	<b>400</b>	<b>430</b>

Tabla 5.4: Comparativa entre esfuerzo real y estimado de cada *sprint*

## Capítulo 6

# Análisis

---

Las probabilidades de éxito del proyecto dependen en gran medida de la realización de un análisis de requisitos detallado y de una correcta identificación de los actores que podrán interactuar con el sistema.

## 6.1 Actores

En este caso, hay un solo tipo de usuario en la aplicación, lo que se traduce en un único actor. Este usuario tendrá que registrarse antes de poder acceder al resto de funcionalidades, de forma que su información de inicio de sesión quede almacenada en el sistema. Si este usuario ya se ha registrado previamente, tendrá que iniciar sesión de nuevo para usar el resto de funcionalidades.

## 6.2 Requisitos

Seguidamente, se detalla el conjunto de requisitos que deberá cumplir el sistema, divididos en dos apartados: requisitos funcionales y no funcionales.

### 6.2.1 Requisitos funcionales

Los requisitos funcionales son aquellos que definen las funcionalidades que debe ofrecer el software a implementar. Para este proyecto se han definido los siguientes requisitos funcionales:

- **Gestión de usuarios:**

- Registro de nuevas cuentas de usuario.
- Autenticación de usuarios previamente registrados en el sistema.

- Búsqueda de usuarios.
- Visualización de perfiles de usuario.
- Edición del perfil de usuario (datos públicos).
- Configuración de la cuenta de usuario (datos privados).
- Eliminación de la cuenta de usuario.
- Establecimiento de relaciones de amistad con otros usuarios.
- Envío y recepción de notificaciones al establecerse relaciones de amistad.
- Cierre de sesión.

- **Gestión de imágenes de usuario:**

- Publicación de imágenes.
- Edición de publicaciones.
- Eliminación de publicaciones.
- Visualización de imágenes publicadas por usuarios.
- Visualización de las imágenes publicadas por un usuario en su perfil.
- Definición de un nivel de privacidad para las publicaciones.

- **Gestión de imágenes de ejemplo:**

- Visualización de imágenes de ejemplo.
- Ordenación de la lista de imágenes de ejemplo.

- **Gestión de valoraciones:**

- Valoración de publicaciones de usuario.
- Filtrado del listado de publicaciones de usuario en función del número de valoraciones.
- Envío y recepción de notificaciones al realizarse valoraciones en publicaciones de usuario.

- **Edición de imágenes:**

- Obtención de imágenes de la galería del usuario.
- Obtención de imágenes de la cámara del usuario.
- Recorte de imágenes.
- Adición de filtros a las imágenes.
- Adición de textos a las imágenes.
- Adición de capas de color a las imágenes.

### 6.2.2 Requisitos no funcionales

Los requisitos no funcionales establecen cómo debe comportarse el software en términos de rendimiento, seguridad, accesibilidad y otros aspectos más allá de sus funcionalidades básicas. Los requisitos no funcionales que aplican a nuestro producto son los siguientes:

- **Seguridad.** El software debe garantizar la privacidad y la integridad de los datos sensibles de las cuentas de usuario, así como de los datos sensibles almacenados en las preferencias de la aplicación.
- **Usabilidad.** La interfaz de usuario debe ser sencilla e intuitiva para un público general sin grandes conocimientos sobre edición de imágenes.
- **Escalabilidad.** La aplicación debe estar preparada para un aumento significativo en la carga de trabajo a realizar. Además, debe favorecer la implementación de nuevas funcionalidades que puedan querer añadirse tras la finalización del proyecto
- **Rendimiento.** El tiempo de respuesta de la interfaz ante las acciones del usuario debe ser siempre lo suficientemente bajo como para no provocar esperas incómodas al usuario.

## 6.3 Historias de usuario

Las historias de usuario [22], empleadas en Scrum, son descripciones sencillas y genéricas de funcionalidades software desde el punto de vista del usuario final. Las historias de usuario son muy útiles a la hora de priorizar el trabajo para poder aumentar el valor de producto para el cliente. Otro punto importante es la flexibilidad que ofrecen, permitiendo su adaptación a medida que se modifican los requisitos del proyecto, lo que suele ser habitual en proyectos con metodologías ágiles. Además, contribuyen a mejorar la precisión de las estimaciones, pues es más sencillo estimar tareas pequeñas que grandes funcionalidades. En definitiva, las historias de usuario dejan de lado la perspectiva del equipo de desarrollo para así centrarse en las necesidades de los usuarios finales de la aplicación.

En cada historia de usuario debe ser posible identificar tres áreas: qué es lo que se necesita, quién lo necesita y por qué lo necesita. Para ello, las historias suelen contener un título conciso, una descripción breve, el usuario final o actor, la acción a realizar por parte de este y el resultado que espera de ella. A continuación se muestra el listado de historias de usuario definidas para nuestro proyecto.

HU-01	
<i>Título</i>	Registrarse
<i>Descripción</i>	El usuario desea poder registrarse en el sistema creando un perfil con el que iniciar sesión
<i>Actor</i>	Usuario
<i>Acción</i>	Proporcionar correo electrónico, fecha de nacimiento, contraseña, nombre de usuario y, opcionalmente, imagen de perfil para a continuación completar el registro
<i>Objetivo</i>	Crear una perfil de usuario que quede registrado en el sistema para poder iniciar sesión y acceder al resto de funcionalidades

Tabla 6.1: HU1: Registrarse

HU-02	
<i>Título</i>	Iniciar sesión
<i>Descripción</i>	El usuario desea poder iniciar sesión en su cuenta previamente creada para acceder al resto de funcionalidades del sistema
<i>Actor</i>	Usuario (previamente registrado)
<i>Acción</i>	Introducir email y contraseña de la cuenta previamente creada para acceder a los datos de la misma y, con ello, al resto de la aplicación
<i>Objetivo</i>	Obtener acceso al conjunto de funcionalidades que ofrece el sistema

Tabla 6.2: HU2: Iniciar sesión

HU-03	
<i>Título</i>	Explorar publicaciones de usuario
<i>Descripción</i>	El usuario desea poder ver el listado de publicaciones de los usuarios, esté él en su lista de amigos o no.
<i>Actor</i>	Usuario (previamente iniciada sesión)
<i>Acción</i>	Navegar por el conjunto de publicaciones que han realizado previamente otros usuarios
<i>Objetivo</i>	Estar al tanto de las nuevas publicaciones que van realizando otros usuarios y conocer sus detalles, incluyendo imagen, descripción, links, fecha de publicación, número de valoraciones y el perfil que hizo la publicación.

Tabla 6.3: HU3: Explorar publicaciones de usuario

HU-04	
<i>Título</i>	Filtrar publicaciones de usuario
<i>Descripción</i>	El usuario desea poder filtrar el listado de publicaciones de usuario en función del número de valoraciones
<i>Actor</i>	Usuario (previamente registrado)
<i>Acción</i>	Introducir un número de valoraciones mínimo para obtener el listado de aquellas publicaciones que lo cumplan
<i>Objetivo</i>	Obtener un listado de publicaciones más corto en el cual se vean sólo aquellas publicaciones que tengan el reconocimiento que busca el usuario

Tabla 6.4: HU4: Filtrar publicaciones de usuario

HU-05	
<i>Título</i>	Buscar perfiles
<i>Descripción</i>	El usuario desea poder realizar búsquedas por nombre de usuario para encontrar perfiles
<i>Actor</i>	Usuario (previamente iniciada sesión)
<i>Acción</i>	Introducir un texto para realizar una búsqueda y así obtener el listado de perfiles cuyo nombre de usuario contenga el texto introducido
<i>Objetivo</i>	Encontrar perfiles de forma rápida en función del nombre de usuario

Tabla 6.5: HU5: Buscar perfiles

HU-06	
<i>Título</i>	Ver perfil
<i>Descripción</i>	El usuario desea poder visualizar los detalles (nombre de usuario, imagen de perfil, número de amigos y publicaciones) de los perfiles de otros usuarios y del suyo propio
<i>Actor</i>	Usuario (previamente iniciada sesión)
<i>Acción</i>	Entrar al perfil de usuario para obtener la información actualizada del mismo
<i>Objetivo</i>	Conocer la información pública de las cuentas de otros usuarios y de la suya propia

Tabla 6.6: HU6: Ver perfil

HU-07	
<i>Título</i>	Seguir perfil
<i>Descripción</i>	El usuario desea poder introducirse en la lista de amigos de otros usuarios
<i>Actor</i>	Usuario (previamente iniciada sesión)
<i>Acción</i>	Activar el seguimiento a un perfil de usuario para introducirse en su lista de amigos
<i>Objetivo</i>	Estar en la lista de amigos de un usuario para poder ver sus publicaciones definidas con una privacidad que excluye a los usuarios fuera de esa lista

Tabla 6.7: HU7: Seguir perfil

HU-08	
<i>Título</i>	Editar perfil
<i>Descripción</i>	El usuario desea poder modificar en cualquier momento su nombre de usuario y su imagen de perfil
<i>Actor</i>	Usuario (previamente iniciada sesión)
<i>Acción</i>	Introducir un nuevo nombre o una nueva imagen para actualizar los datos del perfil
<i>Objetivo</i>	Mantener la estética y la información pública del perfil de acuerdo a las preferencias del usuario

Tabla 6.8: HU8: Editar perfil

HU-09	
<i>Título</i>	Editar cuenta
<i>Descripción</i>	El usuario desea poder modificar los datos privados de su cuenta
<i>Actor</i>	Usuario (previamente iniciada sesión)
<i>Acción</i>	Introducir una nueva fecha de nacimiento o un nuevo correo electrónico para actualizar la información de la cuenta
<i>Objetivo</i>	Actualizar los datos privados de la cuenta de acuerdo con las preferencias del usuario

Tabla 6.9: HU9: Editar cuenta

HU-10	
<i>Título</i>	Cambiar contraseña
<i>Descripción</i>	El usuario desea poder actualizar su contraseña de inicio de sesión
<i>Actor</i>	Usuario (previamente iniciada sesión)
<i>Acción</i>	Introducir y verificar una nueva contraseña para modificar la clave de inicio de sesión en la cuenta
<i>Objetivo</i>	Modificar clave de seguridad de inicio de sesión en la cuenta

Tabla 6.10: HU10: Cambiar contraseña

HU-11	
<i>Título</i>	Eliminar cuenta
<i>Descripción</i>	El usuario desea poder eliminar por completo la información de su cuenta guardada en el sistema
<i>Actor</i>	Usuario (previamente iniciada sesión)
<i>Acción</i>	Eliminar la cuenta para borrar los datos del usuario en el sistema
<i>Objetivo</i>	Eliminar los datos de la cuenta de usuario para que no sea posible utilizarla nunca más

Tabla 6.11: HU11: Eliminar cuenta

HU-12	
<i>Título</i>	Ver amigos
<i>Descripción</i>	El usuario desea poder visualizar el listado de perfiles de usuario que se encuentran dentro de su lista de amigos y el listado de amigos de otros usuarios
<i>Actor</i>	Usuario (previamente iniciada sesión)
<i>Acción</i>	Entrar a la lista de amigos y visualizar el conjunto de perfiles de usuario incluidos
<i>Objetivo</i>	Poder navegar por la lista de amigos propia y las listas de amigos de otros usuarios para encontrar perfiles.

Tabla 6.12: HU12: Ver amigos

HU-13	
<i>Título</i>	Ver notificaciones de amistad
<i>Descripción</i>	El usuario desea ser notificado cuando algún perfil se incluye en su lista de amigos y desea poder ver tanto las nuevas notificaciones como las ya leídas
<i>Actor</i>	Usuario (previamente iniciada sesión)
<i>Acción</i>	Visualizar la lista de notificaciones de amistad de forma que las nuevas notificaciones queden marcadas como leídas
<i>Objetivo</i>	Estar al tanto de los nuevos perfiles que se incluyen en la lista de amigos del usuario

Tabla 6.13: HU13: Ver notificaciones de amistad

HU-14	
<i>Título</i>	Ver notificaciones de valoración
<i>Descripción</i>	El usuario desea ser notificado cuando algún perfil valora una publicación suya y desea poder ver tanto las nuevas notificaciones como las ya leídas
<i>Actor</i>	Usuario (previamente iniciada sesión)
<i>Acción</i>	Visualizar la lista de notificaciones de amistad de forma que las nuevas notificaciones queden marcadas como leídas
<i>Objetivo</i>	Estar al tanto de los nuevos perfiles que valoren las publicaciones del usuario

Tabla 6.14: HU14: Ver notificaciones de valoración

HU-15	
<i>Título</i>	Obtener imagen de la galería
<i>Descripción</i>	El usuario desea poder utilizar imágenes de su galería para posteriormente publicarlas o para establecer su imagen de perfil
<i>Actor</i>	Usuario (previamente iniciada sesión)
<i>Acción</i>	Entrar a la galería para seleccionar una imagen y así emplearla en la aplicación
<i>Objetivo</i>	Usar como imagen de perfil o en sus publicaciones imágenes que el usuario tenga guardadas en su galería previamente

Tabla 6.15: HU15: Obtener imagen de la galería

HU-16	
<i>Título</i>	Sacar fotografía
<i>Descripción</i>	El usuario desea poder acceder a su cámara para sacar fotografías y posteriormente publicarlas o para establecer su imagen de perfil
<i>Actor</i>	Usuario (previamente iniciada sesión)
<i>Acción</i>	Acceder a la cámara del teléfono para sacar una fotografía y así emplearla en la aplicación
<i>Objetivo</i>	Usar como imagen de perfil o en sus publicaciones fotografías que el usuario saque con la cámara de su teléfono sin necesidad de estar previamente guardadas en su galería de imágenes

Tabla 6.16: HU16: Sacar fotografía

HU-17	
<i>Título</i>	Añadir filtro a imagen
<i>Descripción</i>	El usuario desea poder editar sus imágenes aplicando filtros predefinidos por el sistema
<i>Actor</i>	Usuario (previamente iniciada sesión)
<i>Acción</i>	Seleccionar el filtro a aplicar en la imagen y visualizar el resultado
<i>Objetivo</i>	Editar imágenes de forma sencilla con el uso de filtros predefinidos

Tabla 6.17: HU17: Añadir filtro a imagen

HU-18	
<i>Título</i>	Añadir texto a imagen
<i>Descripción</i>	El usuario desea poder editar sus imágenes de manera que pueda agregar uno o varios textos "flotantes" en cualquier parte de la imagen
<i>Actor</i>	Usuario (previamente iniciada sesión)
<i>Acción</i>	Introducir uno o varios textos para que aparezcan encima de la imagen y moverlos por la superficie de esta
<i>Objetivo</i>	Incrustar textos en la imagen que no tenía previamente

Tabla 6.18: HU18: Añadir texto a imagen

HU-19	
<i>Título</i>	Añadir capa de color imagen
<i>Descripción</i>	El usuario desea poder editar sus imágenes aplicando capas de color escogiendo el color y su opacidad
<i>Actor</i>	Usuario (previamente iniciada sesión)
<i>Acción</i>	Escoger un color, definir su opacidad y visualizar el resultado
<i>Objetivo</i>	Aplicar capas de cualquier color durante la edición de imágenes

Tabla 6.19: HU19: Añadir capa de color a imagen

HU-20	
<i>Título</i>	Recortar imagen
<i>Descripción</i>	El usuario desea poder recortar y girar según sus preferencias la imagen que está editando
<i>Actor</i>	Usuario (previamente iniciada sesión)
<i>Acción</i>	Ajustar el recorte y la orientación de la imagen y visualizar el resultado
<i>Objetivo</i>	Obtener como resultado la imagen recortada y con la orientación que desee.

Tabla 6.20: HU20: Recortar imagen

HU-21	
<i>Título</i>	Añadir descripción a publicación
<i>Descripción</i>	El usuario desea poder añadir una descripción en la imagen que esté editando para que aparezca esta sección en su publicación
<i>Actor</i>	Usuario (previamente iniciada sesión)
<i>Acción</i>	Introducir un texto que suponga la descripción de la imagen y previsualizar la publicación
<i>Objetivo</i>	Incluir un apartado descriptivo en la publicación del usuario

Tabla 6.21: HU21: Añadir descripción a publicación

HU-22	
<i>Título</i>	Añadir links a publicación
<i>Descripción</i>	El usuario desea poder añadir una descripción en la imagen que esté editando para que aparezca esta sección en su publicación
<i>Actor</i>	Usuario (previamente iniciada sesión)
<i>Acción</i>	Introducir hasta tres links para que acompañen a la imagen y previsualizar la publicación
<i>Objetivo</i>	Incluir en la publicación del usuario una sección con enlaces externos

Tabla 6.22: HU22: Añadir links a publicación

HU-23	
<i>Título</i>	Definir privacidad de publicación
<i>Descripción</i>	El usuario desea poder definir la privacidad de sus publicaciones para decidir si la puede ver todo el mundo, sólo sus amigos o sólo él mismo
<i>Actor</i>	Usuario (previamente iniciada sesión)
<i>Acción</i>	Seleccionar el tipo de privacidad de la publicación y aplicar los cambios
<i>Objetivo</i>	Decidir qué conjunto de usuarios pueden visualizar cada publicación del usuario

Tabla 6.23: HU23: Definir privacidad de publicación

HU-24	
<i>Título</i>	Realizar publicación
<i>Descripción</i>	El usuario desea poder publicar las imágenes que edita y para las que define una descripción, una serie de links y un nivel de privacidad.
<i>Actor</i>	Usuario (previamente iniciada sesión)
<i>Acción</i>	Realizar publicación para que se registre en el sistema
<i>Objetivo</i>	Incluir nuevas publicaciones en el perfil del usuario y que podrán ver otros usuarios

Tabla 6.24: HU24: Realizar publicación

HU-25	
<i>Título</i>	Editar descripción de publicación
<i>Descripción</i>	El usuario desea poder modificar la descripción de sus publicaciones o añadir una si no lo había hecho a la hora de publicar
<i>Actor</i>	Usuario (previamente iniciada sesión)
<i>Acción</i>	Introducir un texto para que sustituya a la descripción previamente guardada para la publicación y visualizar los cambios
<i>Objetivo</i>	Mantener actualizada la descripción de las publicaciones del usuario de acuerdo a sus preferencias

Tabla 6.25: HU25: Editar descripción de publicación

HU-26	
<i>Título</i>	Editar links de publicación
<i>Descripción</i>	El usuario desea poder modificar los links de sus publicaciones o añadirlos si no lo había hecho a la hora de publicar
<i>Actor</i>	Usuario (previamente iniciada sesión)
<i>Acción</i>	Introducir nuevos links que sustituyan a los previamente guardados para la publicación y visualizar los cambios
<i>Objetivo</i>	Mantener actualizados los links de las publicaciones del usuario de acuerdo a sus preferencias

Tabla 6.26: HU26: Editar links de publicación

HU-27	
<i>Título</i>	Modificar nivel de privacidad de publicación
<i>Descripción</i>	El usuario desea poder modificar la privacidad de sus publicaciones
<i>Actor</i>	Usuario (previamente iniciada sesión)
<i>Acción</i>	Seleccionar un nuevo nivel de privacidad que sustituya al previamente almacenado para la publicación y aplicar los cambios
<i>Objetivo</i>	Mantener actualizado el nivel de privacidad de las publicaciones del usuario de acuerdo a sus preferencias

Tabla 6.27: HU27: Modificar nivel de privacidad de publicación

HU-28	
<i>Título</i>	Eliminar publicación
<i>Descripción</i>	El usuario desea poder eliminar publicaciones que haya realizado previamente
<i>Actor</i>	Usuario (previamente iniciada sesión)
<i>Acción</i>	Eliminar publicación y visualizar su desaparición del perfil
<i>Objetivo</i>	Mantener actualizado el conjunto de publicaciones del usuario de acuerdo a sus preferencias

Tabla 6.28: HU28: Eliminar publicación

HU-29	
<i>Título</i>	Valorar publicación de usuario
<i>Descripción</i>	El usuario desea poder valorar las publicaciones que realicen otros perfiles o el suyo propio
<i>Actor</i>	Usuario (previamente iniciada sesión)
<i>Acción</i>	Valorar la publicación y visualizar el aumento en su número de valoraciones
<i>Objetivo</i>	Enviar a otros usuarios valoraciones en sus publicaciones como muestra de interés

Tabla 6.29: HU29: Valorar publicación de usuario

HU-30	
<i>Título</i>	Visualizar publicaciones de ejemplo
<i>Descripción</i>	El usuario desea poder visualizar la lista de publicaciones de ejemplo que ofrece la aplicación y los detalles de cada una de ellas
<i>Actor</i>	Usuario (previamente iniciada sesión)
<i>Acción</i>	Visualizar y navegar por la lista de publicaciones de ejemplo y acceder al detalle de ellas.
<i>Objetivo</i>	Obtener un listado de publicaciones de las cuales el usuario pueda obtener ideas que aplicar a sus propias publicaciones

Tabla 6.30: HU30: Visualizar publicaciones de ejemplo

HU-31	
<i>Título</i>	Ordenar publicaciones de ejemplo
<i>Descripción</i>	El usuario desea poder ordenar el listado de publicaciones de ejemplo de diferentes formas: más recientes primero, más antiguas primero, más populares primero y de forma aleatoria
<i>Actor</i>	Usuario (previamente iniciada sesión)
<i>Acción</i>	Seleccionar un modo de ordenación y visualizar los cambios en el listado de publicaciones de ejemplo
<i>Objetivo</i>	40 horas

Tabla 6.31: HU31: Ordenar publicaciones de ejemplo

HU-32	
<i>Título</i>	Cerrar sesión
<i>Descripción</i>	El usuario desea poder cerrar la sesión que ha iniciado previamente
<i>Actor</i>	Usuario (previamente iniciada sesión)
<i>Acción</i>	Salir de la parte de la aplicación destinada a los usuarios autenticados
<i>Objetivo</i>	Eliminar del dispositivo la cuenta actual o iniciar sesión con otra cuenta

Tabla 6.32: HU32: Cerrar sesión

# Capítulo 7

# Diseño

---

**L**a fase de diseño consiste en la planificación y estructuración de la implementación de las funcionalidades de la aplicación a partir de los requisitos previamente definidos para el proyecto. Con el objetivo de conseguirlo de manera organizada, debe definirse previamente la arquitectura general del producto y de cada uno de sus subsistemas *software*, además de los patrones de diseño y principios empleados durante el desarrollo.

## 7.1 Arquitectura general

El sistema sigue una arquitectura cliente-servidor. Este lado servidor, integrado por los servicios de Firebase y la [API](#) de Unsplash, tiene las siguientes responsabilidades:

- Almacenamiento y recuperación de datos, asegurando su correcta organización y su integridad.
- Procesamiento de consultas.
- Registro de acceso a los datos.
- Autenticación de los usuarios y control de acceso.
- Encriptación de datos sensibles.
- Gestión de sesiones de usuario.
- Procesamiento de tareas programadas, como el envío automático de *emails*.

Por otra parte, el lado cliente es una aplicación Android desarrollada empleando el lenguaje Dart y el *framework* Flutter, y se ocupa de las siguientes tareas:

- Comunicación con el lado servidor a través del envío de consultas y el procesamiento de las respuestas para obtener los datos.

- Validación de los datos y manejo de errores.
- Presentación de la interfaz de usuario.
- Gestión del estado de la aplicación.
- Gestión de las preferencias de usuario.

En la figura 7.1 se muestra la representación global de la arquitectura del sistema, dividida en subsistemas e incluyendo las comunicaciones entre ellos. Como sucede siempre en la arquitectura cliente-servidor, el cliente se encarga de proporcionar un punto de acceso a los usuarios para interactuar con las funcionalidades de la aplicación y de esa forma con el servidor, encargado de la gestión de los datos.

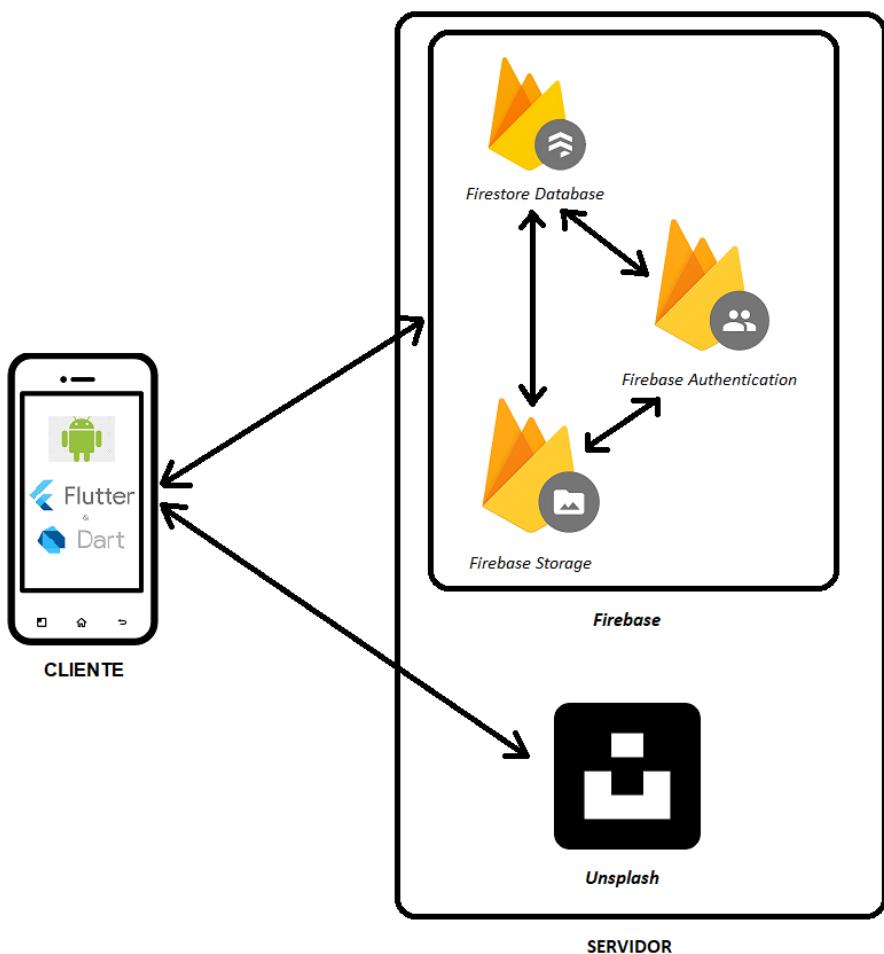


Figura 7.1: Arquitectura general del sistema

## 7.2 Diseño del lado servidor

El *backend* de esta aplicación, como ya se ha mencionado previamente, está compuesto por dos subsistemas principales: la API Unsplash y Firebase. Para el uso de la API no es necesario realizar ningún tipo de desarrollo ni configuración adicional en el *backend*, pues Unsplash tiene sus propias reglas de seguridad y su propio almacenamiento, de forma que se emplean en el *frontend* los *endpoints* que proporciona la API para obtener los resultados referentes a las imágenes de ejemplo. Firebase, en cambio, sí necesita que se configuren las herramientas a utilizar en el proyecto, que en este caso son: Firebase Authentication, Firebase Storage y Firestore Database.

### 7.2.1 Firebase Authentication

Este módulo simplifica en gran medida la gestión de la autenticación y las sesiones de usuario en la aplicación. Se ha configurado el inicio de sesión por medio de correo electrónico y contraseña, y con ello el envío automático de correos de verificación. Cuando un usuario crea una cuenta recibe un correo de verificación al *email* introducido, y si decide modificarlo recibirá otro correo al nuevo *email* introducido. Este correo contendrá un enlace de verificación al que el usuario tendrá que acceder para poder utilizar su cuenta.

Al crear una cuenta a través de Firebase Authentication, esta herramienta genera un UID para el usuario que se emplea para identificarlo de manera única y realizar consultas sobre sus datos y el estado de su sesión. Con esta información se definen reglas de acceso para el resto de servicios de Firebase.

### 7.2.2 Firebase Storage

El almacenamiento en la nube que proporciona Storage ha sido utilizado para el guardado de todas las imágenes creadas por los usuarios. Estas imágenes pueden dividirse en dos categorías: aquellas que pertenecen a las publicaciones de los usuarios y aquellas que actúan como imagen de perfil. Para organizar estos dos grupos se ha decidido estructurar los *buckets* de la siguiente forma:

- Para las imágenes de perfil se ha creado una carpeta llamada "*profileImage*", y dentro de ella se encuentran todos los objetos con las imágenes de perfil, identificado cada uno de ellos por el UID del usuario dueño.
- Para las imágenes de las publicaciones se crea una carpeta por cada usuario, cuyo nombre coincide con el UID de este. Dentro de cada carpeta se encuentran todos los objetos con las imágenes de las publicaciones de ese usuario, identificado cada uno de ellos por el *timestamp* de la fecha en que se realizó la publicación.

Para este servicio se han definido unas reglas de acceso que establecen que cualquier usuario autenticado puede ver cualquier imagen, pero sólo puede modificar imágenes si, además de estar autenticado, esas imágenes son relativas a su perfil (esto se consigue gracias al UID del usuario).

### 7.2.3 Firestore Database

Para la configuración de Firestore Database ha sido necesario concretar el diseño de la base de datos del sistema. Para ello, se ha realizado un diagrama en el que se muestran todas las entidades identificadas para este sistema y las relaciones entre ellas, como se puede ver en la figura 7.2.

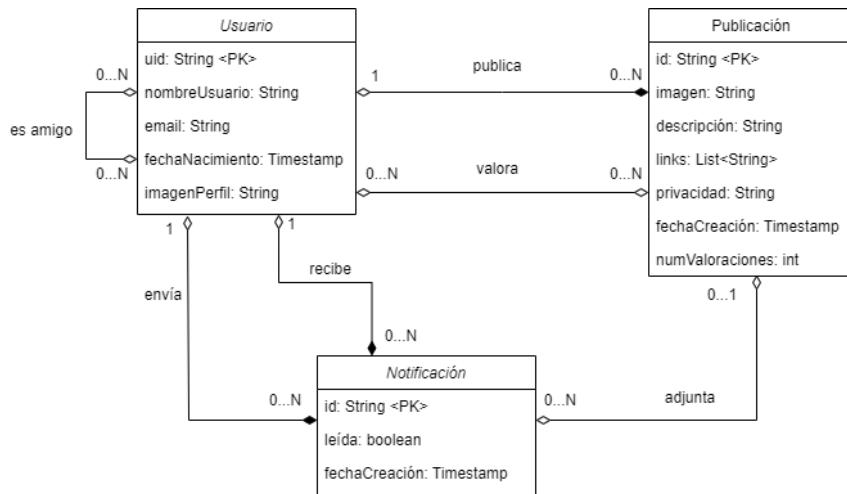


Figura 7.2: Diagrama de entidades

Se pueden diferenciar en el diagrama las siguientes entidades:

- **Usuario.** Representa los usuarios del sistema. Su clave primaria es el UID generado por Firebase Authentication, lo que ayuda a la obtención de datos del usuario en el manejo de las sesiones.
- **Publicación.** Representa las publicaciones que realizan los usuarios del sistema.
- **Notificación.** Representa las notificaciones que reciben los usuarios cuando otros usuarios comienzan a seguirlos o valoran sus publicaciones.

Además de las entidades, pueden verse también las siguientes relaciones:

- **Publica.** Un usuario puede realizar todas las publicaciones que desee. Una publicación siempre tiene como dueño a un único usuario.

- **Valora.** Un usuario puede valorar todas las publicaciones que deseé. Una publicación puede estar valorada por cualquier conjunto de usuarios o ninguno.
- **Es amigo.** Un usuario puede hacerse amigo de todos los usuarios que deseé. Un usuario puede tener en su lista de amigos cualquier conjunto de usuarios o ninguno.
- **Recibe.** Un usuario puede recibir cualquier número de notificaciones. Una notificación siempre está destinada a que la reciba un único usuario.
- **Envía.** Un usuario puede enviar todas las notificaciones que deseé. Una notificación siempre será enviada por un único usuario.
- **Adjunta.** Una notificación puede adjuntar una o ninguna publicación. Una publicación puede estar adjuntada en cualquier conjunto de notificaciones o ninguna.

Para manejar esta situación en Firestore Database se han creado tres colecciones de documentos correspondientes a las tres entidades del diagrama. El manejo de las relaciones se ha realizado dividiéndolas en dos apartados:

- Las relaciones muchos a muchos, como son "valora" y "es amigo", se han implementado como colecciones adicionales. En el caso de la colección de valoraciones, cada documento contendrá una referencia al id del usuario que realiza la valoración y otra al id de la publicación valorada. En el caso de la colección de amistades, cada documento contendrá una referencia al id del usuario seguidor y otra al id del usuario seguido.
- El resto de relaciones se implementan como claves foráneas en el lado no propietario de la relación. Por ejemplo, para la relación entre Usuario y Publicación se añade un atributo a la publicación que contendrá una referencia al id del usuario. El caso aislado de la relación entre Notificación y Publicación, que no tiene lado propietario, se implementa añadiendo una clave foránea a la notificación que se encargue de referenciar el id de la publicación, pero puede aparecer vacía en el caso de que la notificación no adjunte una publicación.

Este diseño de la base de datos concuerda con el diagrama entidad-relación derivado del diagrama de clases previamente definido, como se puede ver en la figura 7.3.

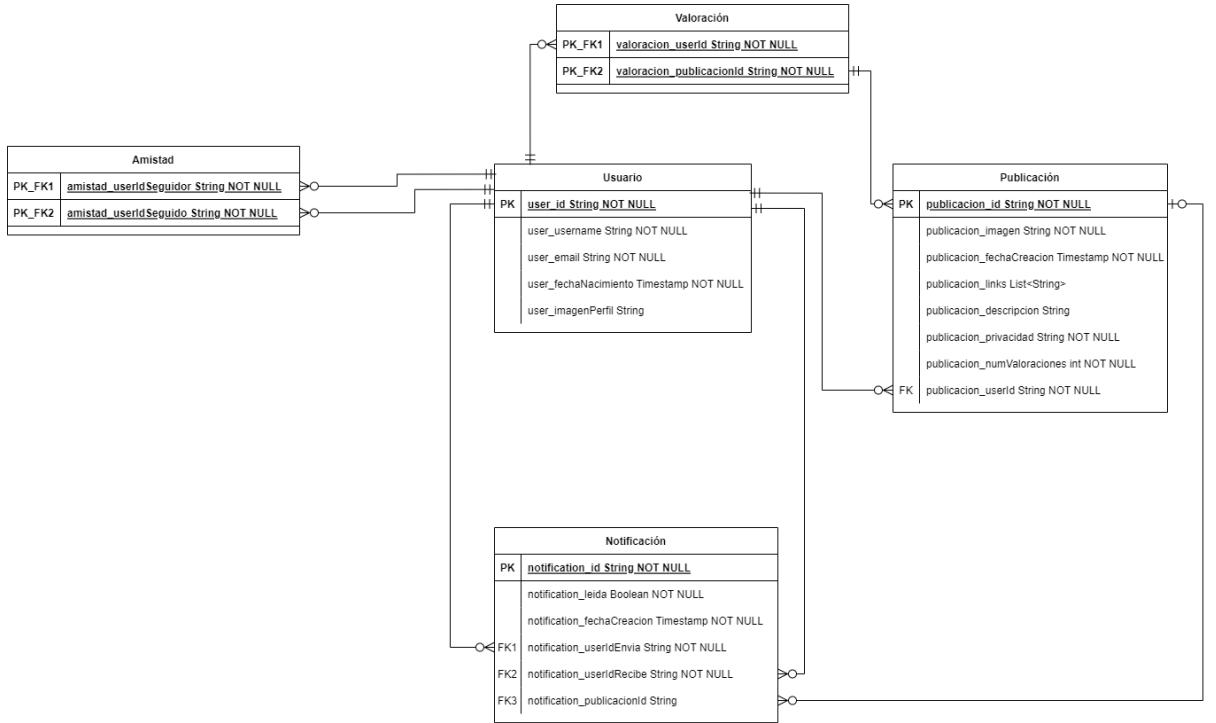


Figura 7.3: Diagrama entidad-relación

Las reglas de acceso aplicadas para Firestore Database se definen de forma diferente para cada colección según sus necesidades. El diseño de estas reglas se centra en excluir accesos innecesarios a las distintas colecciones y en solicitar siempre la autenticación del usuario.

### 7.3 Diseño del lado cliente

En este caso, el *frontend* supone la aplicación que utilizarán los usuarios, y es importante para cumplir los requisitos desarrollarlo siguiendo una organización concreta. También es importante una buena división de las responsabilidades, y con ella una estructuración del proyecto en la que esas responsabilidades sean fácilmente reconocibles.

#### 7.3.1 Patrones y principios

Para facilitar la escalabilidad y el mantenimiento de nuestro producto *software*, es necesaria una buena estructuración. Con este fin existen los patrones de diseño y principios de desarrollo *software* [23]. Los principios de diseño se definen como una serie de reglas y recomendaciones que los programadores deberían seguir para obtener un código limpio, comprensible y mantenable. Los patrones de desarrollo, en cambio, suponen soluciones generales, reutilizables y aplicables a problemas habituales del diseño de software, por lo que están más

relacionados con la estructuración del código.

Se describen en este apartado el conjunto de patrones de diseño y principios de desarrollo empleados en el proyecto.

### Patrones de diseño software

El patrón de diseño principal empleado para el código fuente del subsistema cliente fue [MVC](#) [24], estructurado en tres componentes: modelo, vista y controlador. Cada componente tiene sus propias responsabilidades:

- **Modelo.** Se encarga de la gestión de los datos de la aplicación, sin contener ninguna lógica relativa a la presentación de los mismos.
- **Vista.** Presenta al usuario estos datos y le permite el uso de las distintas funcionalidades, pero no conoce directamente el significado de los datos del modelo
- **Controlador.** Es el responsable de comunicar los demás componentes. Recibe las peticiones de la vista y obtiene las respuestas del modelo, las cuales traduce y envía a la vista para que pueda mostrarlos.

En este caso, se ha añadido adicionalmente un componente "repositorio", destinado al envío de peticiones a la [API](#) Unsplash y a Firebase. El controlador llama al repositorio cuando es necesario solicitar datos al *backend*, y el repositorio se los devuelve al controlador empleando las estructuras de datos del modelo. Este repositorio puede considerarse un subcomponente especializado del modelo, pues este abarca todo lo relacionado con la gestión de datos, incluyendo la obtención de datos de fuentes externas.

En la figura 7.4 se puede ver la estructuración del patrón [MVC](#) y la comunicación entre sus componentes.

Para el manejo de las variables en memoria que necesita la aplicación, como los datos del usuario y el estado de su sesión, se ha empleado también el patrón de diseño *Observer* [25]. Este patrón permite definir un mecanismo de suscripción para notificar a varias clases sobre cualquier modificación que sufra la variable a la que están suscritos. En este caso, la vista está suscrita a las variables contenedoras de los datos necesarios a lo largo de toda la ejecución de la aplicación (que forman parte del modelo), de forma que recibe una notificación que le indica que debe actualizarse cuando se produce cualquier modificación en estos datos.

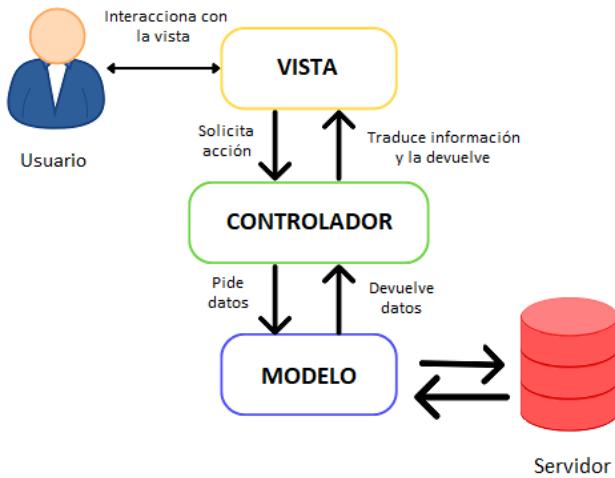


Figura 7.4: Patrón de diseño Modelo-Vista-Controlador

### Principios de desarrollo software

Los principios aplicados en el desarrollo del lado cliente son los siguientes:

- **Single Responsibility Principle.** El principio de responsabilidad única establece que cada clase debe estar destinada a una sola tarea. Esto evita la creación de módulos incompatibles en proyectos en los que varios desarrolladores trabajan simultáneamente y facilita el control de versiones.

Este es el primero de los conocidos principios *SOLID* [26], que suponen un conjunto de reglas y prácticas a seguir para un diseño de clases orientado a objetos.

- **Open-Closed Principle.** El principio de apertura y cierre establece que las clases deben estar cerradas a la modificación y abiertas a la extensión. Esto significa que debe ser posible agregar nuevas funcionalidades al sistema, pero sin modificar el código existente.

Es el segundo de los principios *SOLID*.

- **YAGNI.** El principio *YAGNI* [27] o *You Aren't Gonna Need It* busca la simplificación del desarrollo al establecer que cada componente debe añadirse sólo cuando sea estrictamente necesario. Forma parte de la filosofía de la llamada "programación extrema", que trata de evitar los excesos y la ineficiencia en el código.

- **DRY.** El principio *DRY* [28] o *Don't Repeat Yourself* trata de evitar la duplicación de código de forma innecesaria, promoviendo la reutilización del mismo. Esto afectará directamente sobre la mantenibilidad y escalabilidad del código, pues si el código está

lleno de duplicaciones, cuando haya que realizar cambios es muy probable que sea necesario aplicarlos en más de una parte del código.

- **KISS.** El principio KISS [29] o *Keep It Simple, Stupid* establece que tanto el diseño como la implementación de los distintos componentes del software deben ser simples y directos con el objetivo de facilitar la comprensión, la mantenibilidad y la escalabilidad del código.

### 7.3.2 Comunicación con el lado servidor

Como se ha mencionado previamente, la comunicación con la parte servidor de este producto software se realiza a través de una capa repositorios adicional incluida en el patrón MVC. Para el diseño y la implementación de esta capa se deben identificar previamente todos los accesos a los datos del *backend* que serán necesarios para llevar a cabo las funcionalidades de la *app*. Se enumeran a continuación todos los accesos en este proyecto, divididos en repositorios en función de los datos a los que acceden.

#### Repository de publicaciones de ejemplo

Este repositorio se encarga del manejo de los datos relativos a las publicaciones de ejemplo que ofrece la aplicación. Por tanto, es el encargado de realizar la comunicación con la API Unsplash a través de peticiones HTTP.

Dado que los usuarios no tienen la capacidad de hacer ningún tipo de modificación en este conjunto de publicaciones, los únicos accesos a estos datos deben darse para obtener listas ordenadas de publicaciones.

La API proporciona el endpoint `/search/photos`, al que se le puede añadir el parámetro `order_by` para explicitar el tipo de ordenación que queremos en la lista de respuesta. Esto puede traducirse en el repositorio como la definición de un único tipo de acceso que reciba por parámetro el tipo de ordenación a obtener: `List<Example> getExamples(String order)`

#### Repository de usuarios

Este repositorio debe ocuparse del manejo de los datos sobre los usuarios (incluyendo amistades) y sobre estado de la sesión. Teniendo en cuenta todas las funcionalidades del sistema, este repositorio debe ofrecer los siguientes accesos al *backend*:

- **User loginWithEmailAndPassword(String email, String password).** Inicia sesión en el sistema y devuelve los datos del usuario ya *loggeado*.
- **User signUp(User user).** Crea una cuenta en el sistema y devuelve los datos del usuario ya *loggeado*.

- **User editProfile(String? userName, String? profileImage)**. Actualiza el nombre de usuario y/o la imagen del perfil y devuelve los datos actualizados del usuario.
- **User updateAccount(String? email, DateTime? birthDate)**. Actualia el email y/o la fecha de nacimiento del usuario y devuelve los datos actualizados de este.
- **void deleteAccount(String userId)**. Elimina la cuenta del usuario.
- **void updatePassword(String password)**. Trata de actualizar la contraseña del usuario.
- **void logOut()**. Cierra la sesión del usuario ya *loggeado*.
- **User getUserData(String userId)**. Obtiene los datos de un usuario a partir de un UID.
- **List<User> getFriends(String userId)**. Obtiene todos los amigos de un usuario a partir de un UID.
- **List<User> getFollowedUsers(String userId)**. Obtiene todos aquellos a los que sigue a un usuario a partir de un UID.
- **void followOrUnfollow(String followerId, String followedUserId)**. Incluye o excluye a un usuario de la lista de amigos de otro, en función de si ya está o no en la lista.
- **List<User> getUsers(String username)**. Obtiene todos aquellos usuarios cuyo nombre de usuario contenga el texto especificado por parámetro.

### Repository de publicaciones de usuario

Este repositorio se ocupa del manejo de los datos relativos a las publicaciones de los usuarios, incluyendo las valoraciones realizadas en ellas. Este repositorio debe ofrecer los siguientes accesos:

- **Publication publish(Publication publication)**. Reañliza una nueva publicación y devuelve sus datos una vez incluida en el sistema.
- **Publication editPublication(Publication publication)**. Actualiza los datos de una publicación y devolver la publicación ya actualizada.
- **void deletePublication(Publication publication)**. Elimina una publicación del sistema.
- **List<Publication> getUserPublications(String userId)**. Obtiene todas las publicaciones realizadas por un usuario a partir de un UID.

- ***Publication getPublication(String publicationId)***. Trata de obtener los datos de una publicación con a partir de un id.
- ***List<Publication> getAllPublications(int minLikes)***. Obtiene todas las publicaciones que hay en el sistema filtradas por un número mínimo de valoraciones.
- ***List<Publication> getFollowedUsersPublications(String userId, int minLikes)***. Obtiene todas las publicaciones realizadas por los usuarios a los que sigue un usuario a partir de un UID, filtradas por un número mínimo de valoraciones.
- ***void likeOrDislikePublication(String userId, String publicationId)***. Añade o elimina una valoración a una publicación (con un id concreto) por parte de un usuario (a partir de un UID), en función de si ya existe o no esa valoración.

### Repository de notificaciones

Por último, es necesario un repositorio encargado del manejo de las notificaciones que reciben los usuarios. Este repositorio debe proporcionar los siguientes accesos:

- ***void createNotification(String senderId, String recipientId, String? publicationId)***. Crea una nueva notificación, enviada por un usuario a otro y, opcionalmente, relativa a una publicación. De esta forma, puede crear tanto notificaciones de amistad como de valoración.
- ***void deleteNotification(String senderId, String recipientId, String? publicationId)***. Elimina una notificación del sistema, enviada por un usuario a otro y, opcionalmente, relativa a una publicación. Aunque un usuario no pueda eliminar las notificaciones que recibe, se puede dar el caso que un usuario deje de seguir a otro que elimine su valoración de una de sus publicaciones. Si se da una de esas situaciones, es adecuado eliminar la notificación que se habría creado previamente.
- ***void deactivateNotification(String notificationId)***. Marca como leída una notificación.
- ***List<Notification> getUserNotifications(String userId)***. Obtiene todas las notificaciones recibidas por un usuario a partir de un UID, tanto la leídas como las no leídas.

## Capítulo 8

# Implementación y pruebas

---

DURANTE la fase de implementación del proyecto, en cada *sprint* se han realizado primero el desarrollo y después las pruebas de las nuevas funcionalidades. Siguiendo este enfoque, se convierte en una obligación escribir un código de calidad desde el principio si se busca ser eficiente, de forma que el objetivo de las pruebas se reduzca a reafirmar el cumplimiento de los requisitos predefinidos.

En este capítulo se detallan algunos de los desafíos a los que se enfrentó el desarrollador a lo largo de este proceso, profundizando en los aspectos más técnicos del código desarrollado y en el procedimiento de pruebas. También se incluye la jerarquía de los paquetes creados para organizar los archivos del proyecto.

### 8.1 Estructura del proyecto

Una vez conocidos los subcomponentes del *frontend* y sus responsabilidades, se puede definir una estructura para el proyecto en Android Studio. En la figura 8.1 se muestra el diagrama de paquetes de este proyecto.

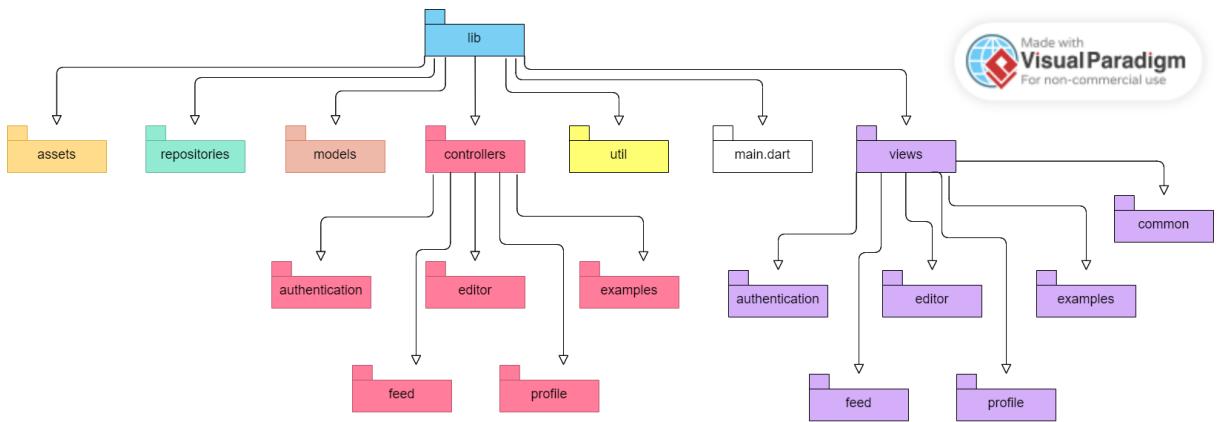


Figura 8.1: Diagrama de paquetes del proyecto en Android Studio

### 8.1.1 Paquete *assets*

Este paquete contiene archivos estáticos empleados en la aplicación, como pueden ser imágenes o logos.

### 8.1.2 Paquete *repositories*

En este paquete se encuentran los repositorios mencionados en el apartado anterior, destinados al acceso a los datos contenidos en el *backend* del sistema.

### 8.1.3 Paquete *models*

Contiene las clases que representan las entidades del lado servidor, así como los datos persistentes en la aplicación. Los repositorios utilizan estas clases para traducir las respuestas en formato **JSON** que obtiene del servidor, de forma que el resto de componentes de la aplicación puedan comprender esos datos.

### 8.1.4 Paquete *controllers*

Este paquete contiene los controladores del **MVC** aplicado a la aplicación, la aplicación, de los que se crean tantos como pantallas hay en la aplicación. Se divide a su vez en varios paquetes, cada uno de ellos destinado a un módulo del sistema desde el punto de vista del usuario final. Estos paquetes son:

- **Authentication.** Destinado al manejo de la autenticación del usuario durante el proceso de registro o inicio de sesión.
- **Editor.** Engloba todo lo relativo al proceso de edición y publicación de imágenes.

- **Feed.** Se ocupa del manejo de las acciones relacionadas con la búsqueda de publicaciones y usuarios.
- **Profile.** Se ocupa de los procesos relativos a los perfiles de usuario.
- **Examples.** Destinado al módulo de las publicaciones de ejemplo que ofrece la aplicación.

### 8.1.5 Paquete *util*

Se emplea este paquete para recopilar funciones estáticas, clases, *enums* y cualquier otro tipo utilidades necesarias en varios puntos del código.

### 8.1.6 Paquete *views*

Este paquete contiene toda la vista de la aplicación, es decir, es el encargado de que se muestre la interfaz de la aplicación al usuario. Su organización en subpaquetes es similar a la del paquete *controllers*, exceptuando que el paquete *views* tiene un subpaquete adicional llamado *common*, en el que se encuentran determinados *widgets* que se utilizan en varias pantallas de la aplicación, como por ejemplo el botón de seguir o las listas de publicaciones.

### 8.1.7 Archivo *main.dart*

Este archivo contiene el componente raíz de toda la aplicación, siendo lo primero que se ejecuta cuando esta se inicia. En él se define el tema de la aplicación, se suscribe la vista a las modificaciones de los datos persistentes (patrón *Observer*), se inicializa la conexión con Firebase y se lanzan todos los procesos de inicio de la interfaz de usuario.

## 8.2 Implementación

Para dar un entendimiento general de la construcción de la aplicación, es necesario destacar por igual las responsabilidades de todas las capas en las que se ha estructurado el código.

### 8.2.1 Modelo

Al modelo se le asignan dos tareas principales con el objetivo de dar soporte a los requisitos funcionales del proyecto: proporcionar una traducción al cliente de los datos del servidor y guardar las variables en memoria de la aplicación. Para ejemplificar y posteriormente poder explicar con mayor detalle las responsabilidades de los componentes del modelo, se muestra a continuación el código de la clase *Image*, que representa las publicaciones realizadas por los usuarios.

```

1 class Image {
2     String id?;
3     String? image;
4     String? userId;
5     String? description;
6     DateTime? creationDate;
7     List<String>? likes;
8     Privacy? privacy;
9     Uint8List? decode;
10    List<Map<String, String>>? links;
11
12    Image({this.decode, this.image, this.userId, this.description, this.creationDate, this.likes,
13          this.privacy, this.links});
14 }

```

Listing 8.1: Clase *Image*

Como se puede observar, la clase *Image* contiene todos los atributos necesarios para las publicaciones de usuario. Para los *links* se emplea el tipo *Map<String, String>*, permitiendo dar la capacidad al usuario de poner títulos a los *links* de sus publicaciones. Para las valoraciones, en cambio, se ha decidido emplear el tipo *List<String>*, pues de esta manera se puede almacenar en una publicación el conjunto de los UID de los usuarios que la han valorado. Esto simplifica el acceso a los datos de esos usuarios y favorece la posibilidad de incorporar nuevas funcionalidades a este módulo, como por ejemplo la visualización del listado de usuarios que han valorado una publicación.

Adicionalmente, se ha incluido un atributo llamado *decode* de tipo *Uint8List*, el cual no se encuentra en la base de datos. Este es un atributo auxiliar destinado a almacenar los datos binarios de una imagen previo a su publicación. Es útil, por ejemplo, para que el usuario pueda previsualizar el resultado de una publicación. Cuando el usuario decide subir la publicación, se emplean estos datos binarios para generar un archivo que pueda subirse a Firebase Storage. Una vez existe la referencia al archivo en Storage, esta es almacenada en Firestore Database a través del atributo *image*.

Para enfocarse ahora en la parte del modelo dedicada al manejo de las variables en memoria de la aplicación, puede verse en el trozo de código 8.2 parte de la clase *AppModel*. Esta clase no trabaja en la comunicación con el lado servidor, sino que almacena determinados datos relevantes durante toda la ejecución de la aplicación y cuya modificación afecta directamente al estado de la misma.

```

1 class AppModel extends ChangeNotifier {
2
3     bool? _isLoggedIn;
4     bool? get isLoggedIn => _isLoggedIn;
5     set isLoggedIn(bool? isLoggedIn) {
6         _isLoggedIn = isLoggedIn;
7         notifyListeners();
8     }
9
10    ...
11 }
12

```

Listing 8.2: Clase *AppModel*

Como se explica en el capítulo 7.3.1, la vista de la aplicación está suscrita a las modificaciones en estos datos a través del patrón *Observer*, por lo que esta clase hará de notificador.

El valor de `_isLoggedIn` indica el estado de la sesión del usuario dentro de la aplicación. Se actualiza cuando el usuario inicia o cierra sesión con una cuenta registrada en el sistema. Los cerrados de sesión pueden producirse de forma voluntaria o involuntaria, pues el manejo de la sesión se realiza desde el *backend*. A continuación se puede ver la parte del código en la que se realiza esa suscripción de la vista a los datos persistentes del modelo, situada en el archivo `main.dart` dentro del componente raíz de la aplicación.

```

1 class MyApp extends StatelessWidget {
2     const MyApp({super.key});
3
4     @override
5     Widget build(BuildContext context) {
6         return MultiProvider(
7             providers: [
8                 ChangeNotifierProvider(create: (c) => AppModel()),
9                 ChangeNotifierProvider(create: (c) => UserModel()),
10                Provider(create: (c) => UserRepository()),
11                Provider(create: (c) => ImageRepository()),
12                Provider(create: (c) => NotificationRepository())
13            ],
14            child: Builder(builder: (context) {
15
16                ...
17
18            }),
19        );
20    }
21

```

Listing 8.3: Suscripción a las modificaciones en los datos persistentes del modelo

### 8.2.2 Repositorios

La capa repositorios es la encargada de comunicarse con la base de datos para enviar peticiones y recibir respuestas, las cuales se devuelven a los controladores contenidas en las entidades del modelo. Para exemplificar el funcionamiento de esta capa se adjuntan a continuación partes del código de dos repositorios de los sistemas.

```

1 class ExampleRepository {
2
3     Future<dynamic> _getMethod(String url) async {
4         Dio dio = Dio();
5         dio.options.headers['content-type'] = 'application/json';
6         return await dio.get(url,
7             options: Options(
8                 responseType: ResponseType.json,
9                 method: 'GET')).then((value) {
10            return value;
11        });
12    }
13
14    Future<List<Example>?> getExamples(String order) async {
15        dynamic response;
16        if (order == Order.random.name) {
17            response = await _getMethod('https://api.unsplash.com/photos/random?count=30
18                &client_id=[claveAcceso]');
19        }
20        else {
21            response = await _getMethod('https://api.unsplash.com/photos?per_page=30
22                &order_by=$order&client_id=[claveAcceso]');
23        }
24        if (response.statusCode == 200) {
25            List<Example> examples = [];
26            response.data.forEach((elm) {
27                Example example = Example.fromJSON(elm);
28                examples.add(example);
29            });
30            return examples;
31        }
32        return null;
33    }
34 }

```

Listing 8.4: Repositorio *ExampleRepositoty*

El código que aparece en el *listing 8.4* corresponde al repositorio destinado a la obtención de publicaciones de ejemplo, lo que significa que se encarga de la comunicación con la API Unsplash. Este repositorio es diferente a los demás, que se comunican con Firebase, y solo necesita un método. Gracias al paquete *Dio* este método puede realizar peticiones [HTTP](#), recibiendo una respuesta en formato JSON que será traducida a la entidad *Example* del modelo gracias al método *fromJSON*. Cuando un controlador llame a este repositorio tendrá que especificar el orden en el que desea visualizar el listado de publicaciones de ejemplo, pues los parámetros de la petición [HTTP](#) son diferentes en función del tipo de orden.

```

1 class UserRepository {
2
3     final FirebaseAuth.FirebaseAuth _auth = FirebaseAuth.FirebaseAuth.instance;
4     final _db = FirebaseFirestore.instance;
5     final FirebaseStorage _storage = FirebaseStorage.instance;
6
7     ...
8
9     Future<User?> signUp(User user) async {
10         try {
11             FirebaseAuth.UserCredential credential = await _auth.createUserWithEmailAndPassword(email:
12                 user.email!, password: user.password!);
13             if (credential.user != null) {
14                 FirebaseAuth.User authUser = credential.user!;
15                 user.id = authUser.uid;
16                 if (user.profileImageFile != null) {
17                     user.profileImage = await _setProfileImage(user.profileImageFile!, user.id!);
18                 }
19
20                 bool success = false;
21                 await _db.collection("usuarios").add(user.toFirestore()).whenComplete(() => success = true);
22                 return success ? user : null;
23             }
24         } catch (e) {
25             return null;
26         }
27     }
28 }
29
30 Future<String> _setProfileImage(Uint8List image, String user) async {
31     Reference reference = _storage.ref().child("profileImage").child(user);
32     UploadTask uploadTask = reference.putData(image);
33     TaskSnapshot taskSnapshot = await uploadTask;
34     return await taskSnapshot.ref.getDownloadURL();
35 }
36
37 ...
38 }
39

```

Listing 8.5: Repositorio *UserRepository*

Por otra parte, el código del *listing 8.5* representa al repositorio destinado al manejo de los datos sobre los usuarios y la sesión en la aplicación. Este repositorio tiene que hacer uso de las tres herramientas de Firebase configuradas para el proyecto:

- **Firebase Authentication** para manejar la sesión del usuario. En la función *signUp* del ejemplo se llama a Authentication para crear una cuenta a través de un *email* y una contraseña y así obtener el UID del usuario.
- **Firebase Storage** para obtener y actualizar los datos sobre las fotos de perfil. Dentro de *signUp*, en el caso de que el usuario insertara una imagen de perfil durante el registro, se suben a Storage los datos binarios de esa imagen y se obtiene la referencia al objeto.
- **Firestore Database** para obtener y actualizar los datos relativos a los usuarios del sistema. En la función *signUp* se puede ver cómo se emplea la información generada por Authentication y Storage y la introducida por el usuario para crear un nuevo documento en la base de datos que suponga un nuevo usuario.

En ambos ejemplos las funciones devuelven *null* en caso que se produzca un error, lo que contribuye a que tanto el controlador como la vista puedan diferenciar con facilidad casos de éxito y fallo en las peticiones. Además, ambas funciones llevan la palabra clave *async*, que indica que la ejecución de la aplicación no se queda esperando por la respuesta a la petición, sino que esta se procesa en un hilo de ejecución o *thread* paralelo.

### 8.2.3 Controladores

Los controladores del sistema se limitan a realizar la comunicación entre vistas y repositorios y, cuando es necesario, a actualizar los datos persistentes del modelo. Dado que todas las pantallas de la aplicación tienen sus propias funcionalidades y necesidades de información, se ha creado un controlador por cada pantalla. Otro enfoque habitual es el de crear sólo un controlador por cada repositorio, pero este se suele aplicar cuando la aplicación tiene pocas pantallas o cuando, a pesar de que tenga varias pantallas, las peticiones que se realizan se repiten en varias de ellas.

Un controlador es llamado de forma asíncrona por su vista correspondiente y a continuación llama al repositorio o repositorios pertinentes. Cuando el controlador recibe la respuesta de los repositorios, este realiza en ella las comprobaciones y ajustes necesarios antes de trasladarla a la vista. En determinados casos, tendrá que decidir si es necesario actualizar datos persistentes de la aplicación, como por ejemplo tras realizar un intento de inicio de sesión.

Se muestra en el *listing 8.6* uno de los métodos del controlador de la vista del perfil, la cual se emplea tanto para el perfil del usuario propio como para perfiles de otros usuarios.

```

1 class ProfileController extends BaseController {
2
3     ...
4
5     Future<List<Image>> getUserImages(String? userId) async {
6         List<Image>? userImages;
7
8         if (userId != null) {
9             userImages = await userRepository.getUserImages(userId);
10        } else {
11            userImages = await userRepository.getUserImages(userModel.user!.id!);
12        }
13
14        userImages?.sort((a, b) => (b.creationDate ?? DateTime.now()).compareTo(a.creationDate ??
15            DateTime.now()));
16        return userImages;
17
18    ...
19 }

```

Listing 8.6: Controlador *ProfileController*

En la función *getUserImages*, que trata de recuperar las publicaciones de un perfil de usuario, se diferencian dos casos: cuando la vista nos proporciona un UID para realizar la petición y cuando no. De esta manera se distingue cuándo estamos obteniendo las publicaciones de

nuestro perfil de usuario y cuándo las de otro. En el caso de obtener nuestras publicaciones, se obtiene el UID de nuestro usuario de las variables en memoria de la aplicación. Una vez obtenida la lista de publicaciones, ésta se ordena en función de la fecha de publicación más reciente antes de devolverla a la vista.

#### 8.2.4 Vistas

La capa vista de la aplicación es la encargada de presentar la interfaz de usuario y a través de ella todas las funcionalidades e información que ofrece nuestro sistema. El uso del *framework* Flutter ha sido primordial durante todo el desarrollo de la parte gráfica.

Recogiendo las interacciones del usuario con la interfaz, la vista realizará solicitudes a su controlador asociado y recogerá sus respuestas para cambiar su estado. Para manejar estos cambios de estado de las vistas se ha decidido utilizar los conocidos *StatefulWidget* de Flutter; *widgets* que permiten definir variables de estado cuya modificación provoque una actualización de la interfaz.

El envío de peticiones desde la vista se realiza de manera asíncrona, pues esta no puede dejar de responder a las interacciones del usuario. Algunas pantallas, como por ejemplo los perfiles, necesitan cargar datos incluso antes de que el usuario realice ninguna interacción. Con este objetivo se hace uso de *FutureBuilder*, un *widget* de Flutter que permite construir una interfaz de usuario basándose en el estado de un *Future* (que se trata de un valor estará disponible en algún momento del futuro).

En el ejemplo 8.7 se muestra el primer *widget* al que referencia el componente raíz del proyecto. Gracias al *widget Future Builder* se puede mostrar una cosa u otra en la interfaz de usuario en función del estado de *isLogged*. Cuando se renderice el componente *App* comenzará a ejecutarse la función asíncrona *future()*, y durante su ejecución se mostrará un indicador de carga en la pantalla. Dentro de *future()* se llama a la función *getIsLogged()* del controlador, la cual accede a las preferencias de usuario almacenadas en el teléfono para recuperar sus anteriores datos de inicio de sesión (si existen) y trata de iniciar sesión de nuevo, de forma que el usuario no tenga que iniciar sesión manualmente cada vez que abre la aplicación. Una vez se obtiene la respuesta, en función de si el resultado es positivo o negativo se llevará al usuario al *feed* o al inicio de sesión, respectivamente.

```

1 class App extends StatefulWidget {
2   const App({super.key});
3
4   @override
5   _AppState createState() => _AppState();
6 }
7
8 class _AppState extends State<App> {
9
10   final controller = BaseController();
11   late Future<bool?> isLoggedIn;
12
13   Future<bool?> future() async {
14     await Future.delayed(const Duration(seconds: 3));
15     return controller.getIsLogged();
16   }
17
18   @override
19   void initState() {
20     isLoggedIn = future();
21     super.initState();
22   }
23
24   @override
25   Widget build(BuildContext context) {
26     return FutureBuilder(
27       future: isLoggedIn,
28       builder: (context, snapshot) {
29         if (snapshot.connectionState == ConnectionState.done) {
30           if (snapshot.hasError) {
31             return const LoginPage();
32           } else if (snapshot.hasData) {
33             if (snapshot.data == false) {
34               return const LoginPage();
35             }
36             else {
37               return const Home();
38             }
39           } else {
40             return const LoginPage();
41           }
42         } else {
43           return Container(
44             height: double.infinity,
45             width: double.infinity,
46             color: Colors.white,
47             child: const Center(child: CircularProgressIndicator(color: Colors.deepOrange),),
48           );
49         }
50       },
51     );
52   }
53 }
```

Listing 8.7: Uso de *FutureBuilder*

El desafío más complejo al que se ha enfrentado el alumno en cuanto al desarrollo de la vista ha sido conseguir el correcto funcionamiento y una buena estructuración en la pantalla del editor, concretamente en el proceso de adición de filtros, textos y capas de color. En esta vista se producen muchos cambios de estado sin realizar peticiones al controlador, pues no es necesario obtener datos del *backend*. Esto genera que se concentre gran parte de la lógica en la vista, lo que complica la consecución de un código limpio y sencillo. Además, el hecho de que puedan cargarse imágenes de cualquier tamaño obliga a que deba aplicarse lógica también al

renderizado de la pantalla y no sólo a la funcionalidad propiamente dicha. En el *listing 8.8* se puede observar el *widget* que muestra en pantalla la imagen aplicando las ediciones que va añadiendo el usuario.

```

1  SizedBox(
2    height: imageHeight,
3    width: changedWidth ? screenWidth : snapshot.data!,
4    child: RepaintBoundary(
5      key: _globalKey,
6      child: Stack(
7        children: [
8          GestureDetector(
9            onTap: () {
10              if (selectedText != null) {
11                setState(() {
12                  texts[selectedText!].isSelected = false;
13                  selectedText = null;
14                });
15              }
16            },
17            child: Stack(
18              children: [
19                ColorFiltered(
20                  colorFilter: filter,
21                  child: Image.memory(image, fit: BoxFit.fitHeight)
22                ),
23                Container(
24                  color: imageColor,
25                  width: changedWidth ? screenWidth : snapshot.data!,
26                  height: imageHeight,
27                ),
28              ],
29            ),
30          ),
31          ...textWidgets,
32        ],
33      ),
34    ),
35 )

```

Listing 8.8: *Stack* con imagen en el editor

El filtro, la capa de color y los textos aplicados a la imagen están contenidos junto a la propia imagen en una pila con una anchura y altura determinados (previamente calculados en función de las dimensiones originales de la imagen). *RepaintBoundary* es el *widget* que permitirá exportar la imagen con todas las ediciones aplicadas. La lista *textWidgets* contiene todos los textos añadidos por el usuario a la imagen, cada uno de ellos con una posición definida. La inicialización de este listado se muestra a continuación en el *listing 8.9*.

```

1 List<Widget> textWidgets = texts.asMap().entries.map((entry) {
2     int index = entry.key;
3     var textInfo = entry.value;
4     return Positioned(
5         left: textInfo.left,
6         top: textInfo.top,
7         child: _selectedIndex == 1
8             ? Draggable(
9                 feedback: ImageText(textInfo: textInfo),
10                child: ImageText(textInfo: textInfo),
11                onDragEnd: (drag) {
12                    final renderBox = context.findRenderObject() as RenderBox;
13                    Offset off = renderBox.globalToLocal(drag.offset);
14                    double paddingLateral = MediaQuery.of(context).padding.left;
15                    double paddingSuperior = MediaQuery.of(context).padding.top;
16                    double spaceLateral = (screenWidth - snapshot.data!) / 2;
17                    if (changedWidth) {
18                        spaceLateral = 0;
19                    }
20
21                    setState(() {
22                        textInfo.top = off.dy - appBarHeight - spaceHeight - paddingSuperior;
23                        textInfo.left = off.dx - spaceLateral - paddingLateral;
24                        textInfo.isSelected = true;
25                        if (selectedText != null && selectedText != index) {
26                            texts[selectedText!].isSelected = false;
27                        }
28                        selectedText = index;
29                    });
30                },
31            )
32            : ImageText(textInfo: textInfo)
33        );
34    }).toList();

```

Listing 8.9: Creación de *textWidgets* en el editor

El *widget Draggable* permite crear un componente desplazable para que el usuario pueda arrastrarlo por la pantalla. Gracias a él podemos determinar la posición en la que se encuentra cada texto y actualizarla. A su vez hace uso de un widget *ImageText*, el cual simplemente muestra los textos y los actualiza a petición del usuario.

### 8.3 Pruebas

Las pruebas se han realizado de manera manual a través del simulador proporcionado por Android Studio y un teléfono móvil personal del alumno, tratando de corroborar el correcto funcionamiento de la aplicación actuando como los usuarios finales. Para cada *sprint* se ha diseñado un plan de pruebas, el cual se ejecuta una vez finalizado el desarrollo de las funcionalidades. Además, previo a esa ejecución, se deben preparar datos de prueba con el objetivo de poder probar funcionalidades relacionadas con datos externos a la sesión del usuario. Esta preparación de datos de prueba incluye la adición (desde la web de Firebase) de publicaciones, usuarios, notificaciones, valoraciones y amistades a la base de datos del sistema.

Se proporciona como ejemplo en la tabla 8.3 el plan de pruebas del segundo *sprint* del proyecto.

	<b>Acción</b>	<b>Resultado esperado</b>	<b>Datos necesarios</b>
<i>T1</i>	Crear cuenta de usuario.	Se accede a todas las funcionalidades de la aplicación.	
<i>T2</i>	Iniciar sesión en una cuenta de usuario	Se accede a todas las funcionalidades de la aplicación.	Cuenta de usuario.
<i>T3</i>	Visualizar perfil de usuario propio	Se muestran correctamente en el perfil propio los datos de la cuenta de usuario en la que se ha iniciado sesión.	Cuenta de usuario.
<i>T4</i>	Editar perfil de usuario.	Se visualiza en el perfil de usuario propio los nuevos datos de nombre de usuario y de imagen de perfil.	Cuenta de usuario.
<i>T5</i>	Editar cuenta de usuario.	Se actualiza la fecha de nacimiento, se cierra la sesión y se envía un correo al nuevo <i>email</i> del usuario para que verifique su email y pueda acceder a su cuenta de nuevo.	Cuenta de usuario.
<i>T6</i>	Actualizar contraseña.	Se actualiza la contraseña del usuario necesaria para acceder a su cuenta.	Cuenta de usuario.

	<b>Acción</b>	<b>Resultado esperado</b>	<b>Datos necesarios</b>
T7	Eliminar cuenta de usuario.	Se cierra la sesión y se eliminan por completo los datos de la cuenta de usuario, imposibilitando volver a utilizarla.	Cuenta de usuario.
T8	Cerrar sesión.	Se muestra la pantalla de inicio de sesión.	Cuenta de usuario.
T9	Visualizar lista de amigos.	Se muestra el listado de todos los perfiles de usuario que se encuentran en nuestra lista de amigos.	Varias cuentas de usuario. Relaciones de amistad entre esas cuentas de usuario.
T10	Visualizar notificaciones.	Se muestran destacadas las notificaciones no leídas y al salir y volver a entrar a la pantalla de notificaciones estas aparecen como leídas	Cuenta de usuario. Notificaciones no leídas para esa cuenta de usuario.
T11	Buscar usuarios por nombre de usuario.	Se muestra la lista de usuarios que contienen en su nombre de usuario el texto introducido en la búsqueda.	Varias cuentas de usuario.
T12	Buscar usuarios seguidos por nombre de usuario.	Se muestra la lista de usuarios seguidos que contienen en su nombre de usuario el texto introducido en la búsqueda.	Varias cuentas de usuario. Relaciones de amistad entre esas cuentas de usuario.

	<b>Acción</b>	<b>Resultado esperado</b>	<b>Datos necesarios</b>
<i>T13</i>	Visualizar perfil de usuario ajeno.	Se muestran correctamente los datos del perfil del usuario.	Varias cuentas de usuario.
<i>T14</i>	Seguir perfil de usuario.	El perfil de usuario aparece al realizar una búsqueda sobre los perfiles de usuario seguidos.	Varias cuentas de usuario.
<i>T15</i>	Dejar de seguir perfil de usuario.	El perfil de usuario no aparece al realizar una búsqueda sobre los perfiles de usuario seguidos.	Varias cuentas de usuario. Relaciones de amistad entre esas cuentas de usuario.

Tabla 8.1: Plan de pruebas del *sprint 2*

Además de estas pruebas manuales realizadas por el alumno, se han realizado al final de cada iteración del proyecto pruebas de aceptación conjuntamente con el tutor. Estas pruebas de aceptación tienen el objetivo de confirmar que las funcionalidades y la interfaz de la aplicación se ajustan a lo que busca el usuario final. También se trata de pruebas manuales, pero en este caso alumno y tutor colaboran para dar un enfoque menos técnico y más práctico a la hora de determinar los resultados de las pruebas.

## Capítulo 9

# Conclusiones y futuras líneas de trabajo

---

EN este proyecto se ha desarrollado una aplicación Android que actúa como una red social basada en la edición y la publicación de imágenes. Esta aplicación busca proporcionar a los usuarios inexpertos en la creación de contenido para redes sociales una plataforma en la que puedan instruirse en este ámbito. Estos usuarios pueden aprender visualizando una galería de publicaciones de ejemplo o las publicaciones de otros usuarios para posteriormente crear sus propias publicaciones.

Se puede considerar que el proyecto ha sido exitoso, pues se han cumplido los requisitos funcionales y no funcionales iniciales, presentando al usuario todas las características requeridas a través de una interfaz de usuario diseñada para tratar de ser intuitiva.

Para llegar al objetivo marcado ha sido necesario aplicar gran parte de los conocimientos adquiridos a lo largo del grado, tanto conocimientos de programación como aquellos subyacentes, relacionados con la construcción y documentación de proyectos *software*. El desarrollo de una aplicación móvil de principio a fin por parte del alumno supone la puesta en práctica de estos conocimientos, pero también un constante aprendizaje, especialmente a la hora de emplear un nuevo lenguaje y nuevas herramientas como Firebase.

Aplicar una metodología ágil como procedimiento de desarrollo prepara al alumno para enfrentarse a entornos de desarrollo reales, adaptándose a las necesidades del mercado laboral. El uso de metodologías ágiles está muy extendido actualmente debido a que se busca adoptar un enfoque más flexible.

El diseño de la aplicación se ha visto influido en gran medida por el interés en la escalabilidad del sistema. En cuanto al *backend*, algunas de las razones por las que se escogió Firebase son la gran cantidad de servicios que proporciona más allá del alojamiento de datos y la flexibilidad para modificar las configuraciones de estos servicios, lo que abre las puertas a incorporar nuevas funcionalidades a la aplicación. Por otro lado, el diseño del *frontend* ha adoptado el

patrón [MVC](#), un gran aliado de la mantenibilidad y la escalabilidad de los productos software.

Gracias a estas características, que se han potenciadas por el diseño del sistema, podemos pensar en nuevas líneas de trabajo en las cuales se incorporen nuevas funcionalidades a la aplicación o se mejoren funcionalidades existentes para el beneficio de los usuarios. Algunas de las posibles mejoras a incluir en el sistema son:

- Implementación de un modo oscuro para la interfaz de usuario.
- Añadir internacionalización para que cada usuario visualice la aplicación en el idioma que tenga configurado en su teléfono.
- Añadir la opción de escoger el color de los textos incrustados en las imágenes durante la edición.
- Añadir al editor de imágenes opciones de ajuste de brillo y opacidad.
- Permitir al usuario guardar publicaciones, tanto de ejemplo como de otros usuarios, de forma que se incluya una nueva sección en su perfil en la que aparezcan todas sus publicaciones guardadas.
- Permitir incluir más de una imagen por publicación.
- Permitir visualizar el listado de usuarios que han valorado una publicación.

# **Apéndices**

## Apéndice A

# Manual de usuario

---

**E**n este apartado se adjunta un manual de usuario completo para todas las funcionalidades de la aplicación. El manual se divide en las siguientes secciones:

- Inicio de sesión y registro.
- *Feed.*
- Publicaciones de ejemplo.
- Editor de imágenes.
- Perfiles de usuario.

Estas secciones se dividirán a su vez en apartados más pequeños para poder aislar cada funcionalidad de la aplicación y explicar correctamente su uso.

## A.1 Inicio de sesión y registro

### A.1.1 Inicio de sesión



Figura A.1: Inicio de sesión

Para iniciar sesión en la aplicación con una cuenta de usuario debemos introducir correctamente el *email* y la contraseña asociados a esa cuenta. Tras introducirlos y pulsar sobre "Iniciar Sesión" se nos redirigirá al *feed* de la aplicación.

Si pulsamos sobre "Créala ahora!" aparecerá la pantalla de registro de una nueva cuenta.

### A.1.2 Registro

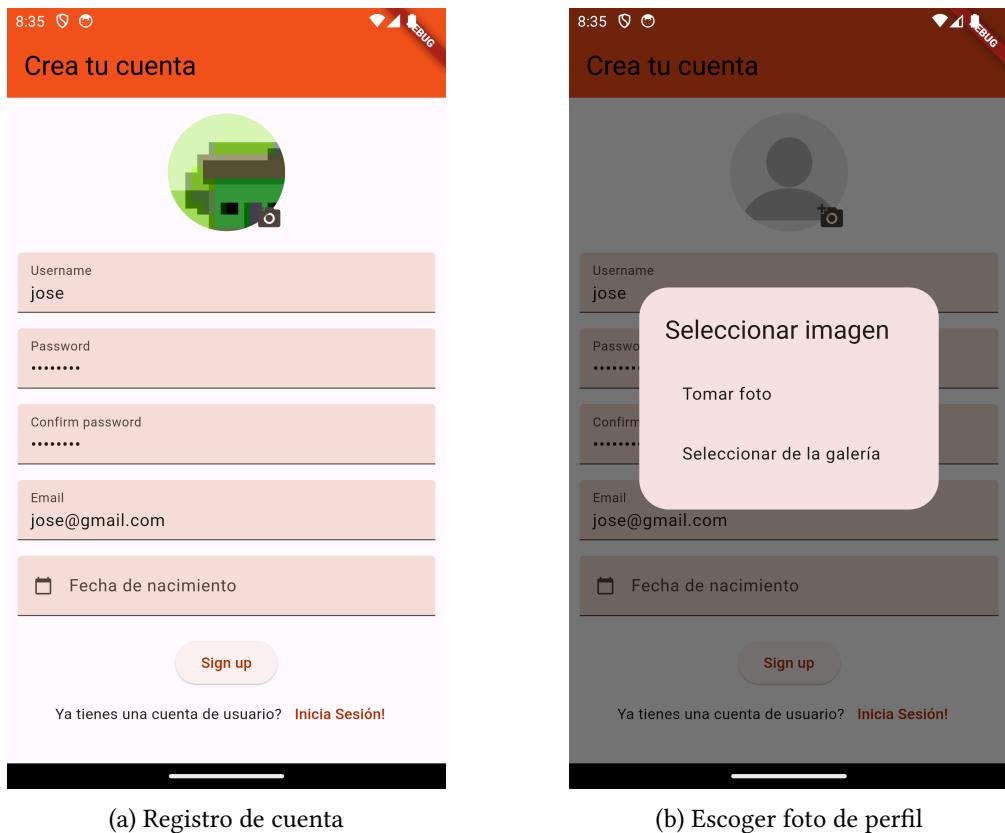


Figura A.2: Registro de nueva cuenta de usuario

Para crear una cuenta es necesario introducir todos los datos solicitados en los campos de texto. El único campo opcional es la imagen de perfil. Una vez introducidos los datos, si pulsamos en "Sign up" se creará la cuenta y accederemos al *feed* de la aplicación.

Si queremos asignar una imagen de perfil debemos pulsar sobre el ícono de la cámara situado en el círculo con la imagen y se abrirá un *pop up* en el que podremos decidir si obtener esa imagen de la galería o de la cámara del teléfono.

Para volver a la pantalla de inicio de sesión pulsaremos sobre "Inicia sesión!".

### A.1.3 Cerrar sesión

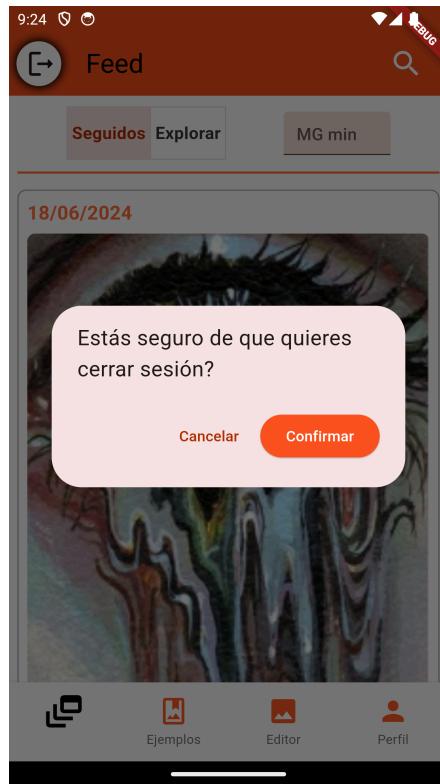


Figura A.3: Cerrar sesión

Una vez iniciada la sesión, en cualquier momento se podrá pulsar sobre el ícono de la esquina superior izquierda de la pantalla para cerrar la sesión. Al pulsar en este botón aparecerá un *pop up* que nos pedirá confirmar el cierre de sesión. Si lo confirmamos se cerrará la sesión y seremos redirigidos a la pantalla de inicio de sesión.

## A.2 Feed

Se llama *feed* a un flujo de contenido desplazable. Este contenido aparece en forma de bloques que se repiten uno después de otro. En nuestra aplicación llamamos *feed* a la primera pantalla que visualiza el usuario cuando ha iniciado sesión, en la cual aparece en este formato el listado de publicaciones hechas por otros usuarios, ordenadas de más reciente a más antigua.

### A.2.1 Publicaciones de usuarios seguidos

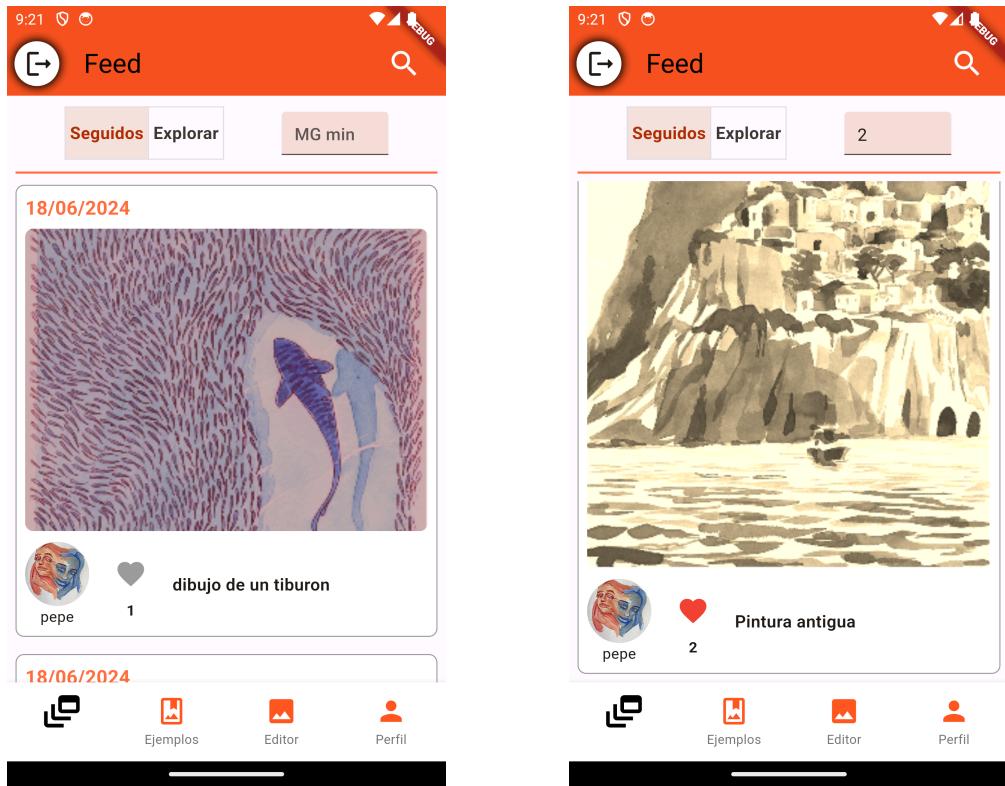


Figura A.4: Publicaciones de usuarios seguidos y filtro por valoraciones

Al iniciar sesión en la aplicación veremos en el *feed* el listado de publicaciones (públicas o para amigos) realizadas por aquellos usuarios a los que seguimos, es decir, aquellos que nos tienen incluidos en su lista de amigos. La lista se ordena de fecha de publicación más reciente a más antigua, y es posible filtrarla por número de valoraciones o "me gusta" introduciendo un número en el campo de la esquina superior derecha.

### A.2.2 Todas las publicaciones

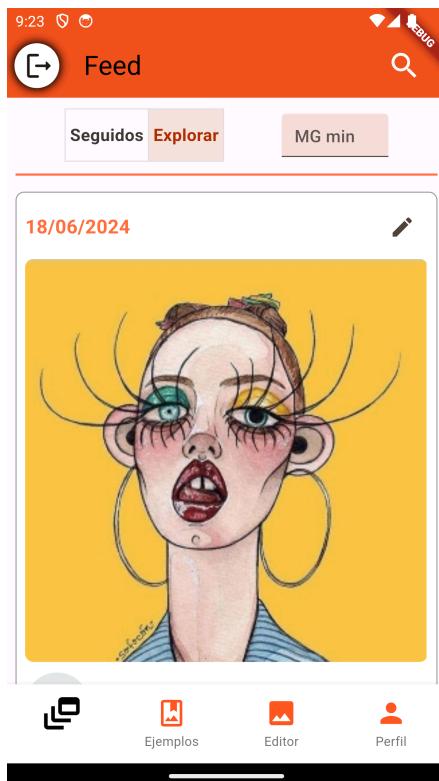


Figura A.5: Sección "Explorar"

Si pulsamos en el selector de la parte superior izquierda sobre la opción "Explorar", podremos ver ordenadas de más reciente a más antigua todas las publicaciones realizadas por los usuarios, estemos en su lista de amigos o no. Podremos ver las publicaciones públicas de todos los usuarios y aquellas para amigos de los usuarios a los que seguimos. También pueden filtrarse estas publicaciones por número de valoraciones. Si pulsamos sobre la opción "Seguidos", volveremos a ver sólo aquellas publicaciones de los usuarios a los que seguimos

### A.2.3 Buscar usuarios

Pulsando sobre el símbolo de la lupa que aparece en la parte derecha del *banner* superior, iremos a la sección de búsqueda de usuarios.

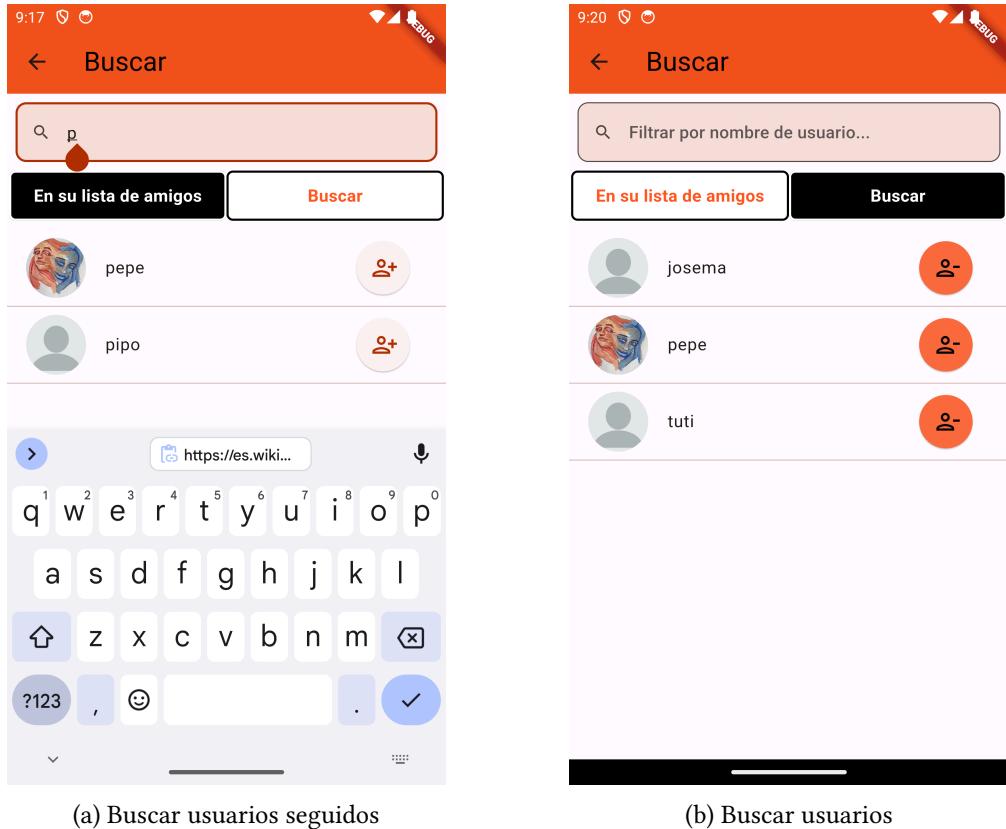


Figura A.6: Buscar usuarios por nombre de usuario

Al entrar a esta pantalla, veremos la lista de usuarios a los que seguimos, la cual podremos filtrar por nombre de usuario introduciendo un texto en el campo editable de la parte superior de la pantalla. Si pulsamos en la opción "Buscar", podremos realizar búsquedas en ese campo para encontrar usuarios entre todos los existentes en el sistema. Se puede acceder a cualquier perfil de usuario pulsando sobre los elementos de la lista, y seguirlos o dejarlos de seguir usando el botón de la parte derecha de los elementos.

Para volver al *feed* pulsaremos sobre la flecha que aparece en la parte izquierda del *banner* superior. Esta flecha siempre tiene la misma funcionalidad.

## A.3 *Publicaciones de ejemplo*

Para acceder a la sección de publicaciones de ejemplo que ofrece la aplicación tendremos que pulsar sobre la opción "Ejemplos" de la barra de navegación inferior.

### A.3.1 Listado de publicaciones

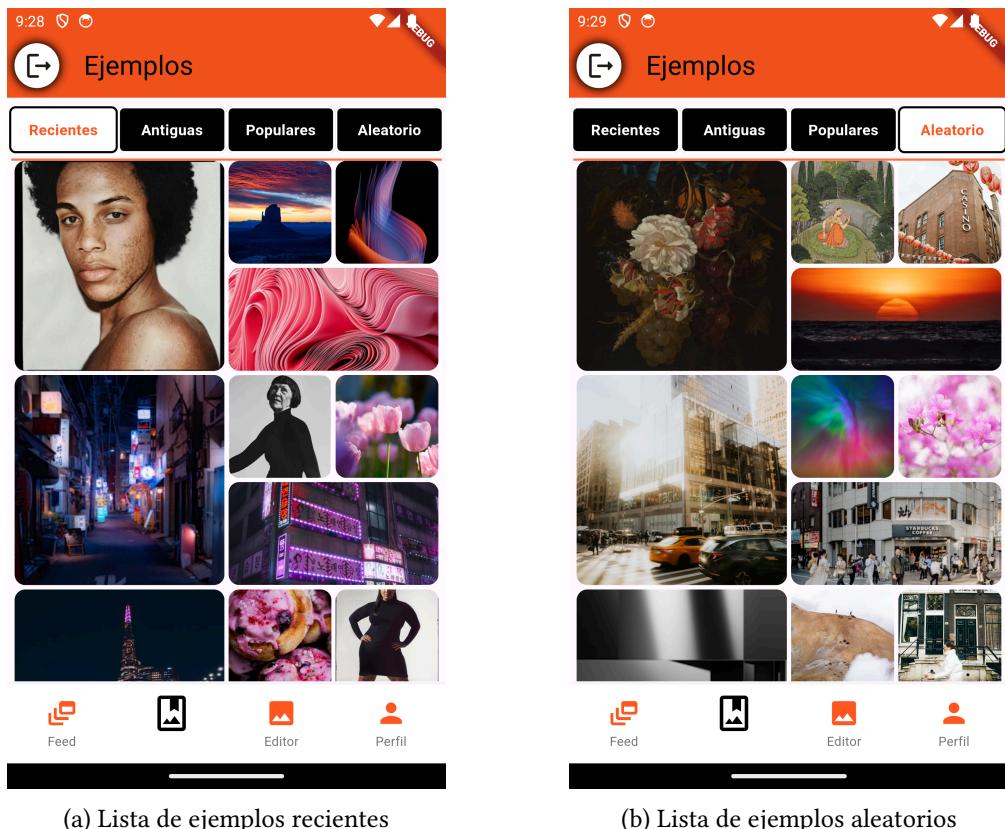


Figura A.7: Listado de publicaciones de ejemplo

Al entrar a esta sección veremos un listado de ejemplos desplazable verticalmente. Pulsando sobre las opciones de la parte superior de la lista ejemplos podremos modificar la ordenación de los ejemplos.

### A.3.2 Detalle de publicación



Figura A.8: Detalle de publicación de ejemplo

Si pulsamos sobre uno de los ejemplos del lista accederemos a esta pantalla, en la que podremos ver al completo la publicación.

## A.4 Editor de imágenes

Para acceder al editor, tendremos que pulsar sobre la opción "Editor" de la barra de navegación inferior.

### A.4.1 Escoger imagen

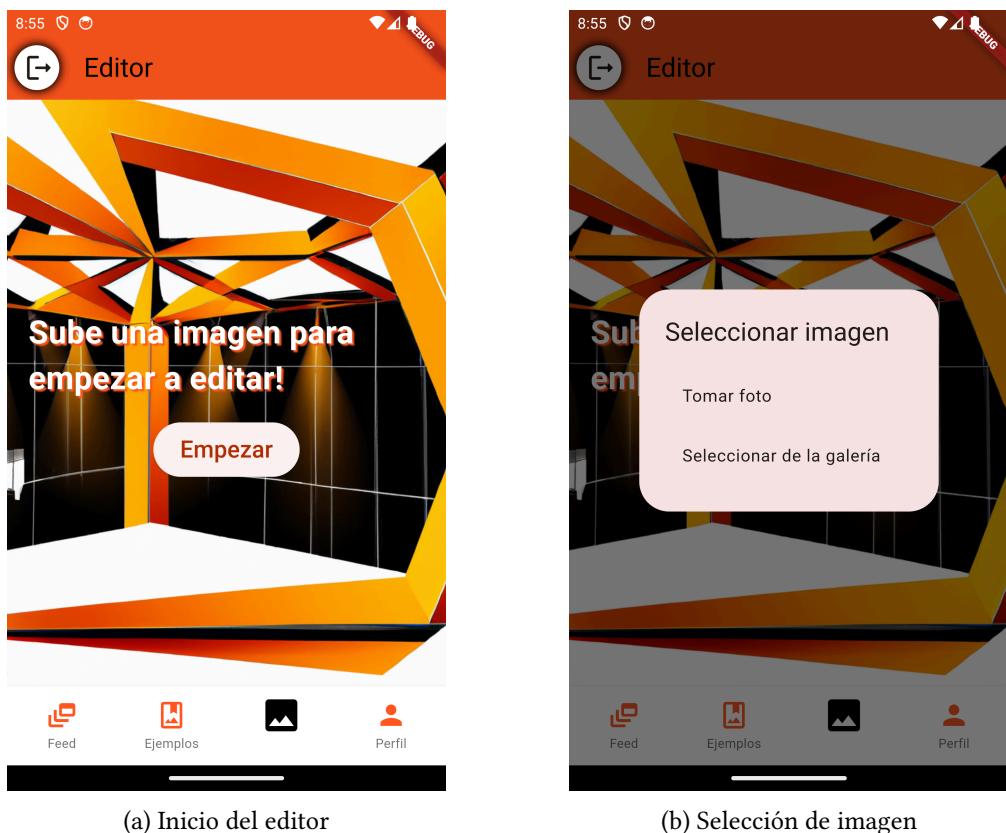


Figura A.9: Editor de imágenes

Al entrar a la sección del editor veremos la pantalla de inicio, en la que podremos escoger una imagen para editar si pulsamos en el botón "Empezar". A continuación, podremos seleccionar si queremos escoger una imagen de la galería o si queremos sacar una foto con la cámara de nuestro teléfono.

### A.4.2 Editar imagen

Una vez seleccionada la imagen accederemos al editor, donde podremos realizar distintas configuraciones sobre la imagen.

#### Filtros

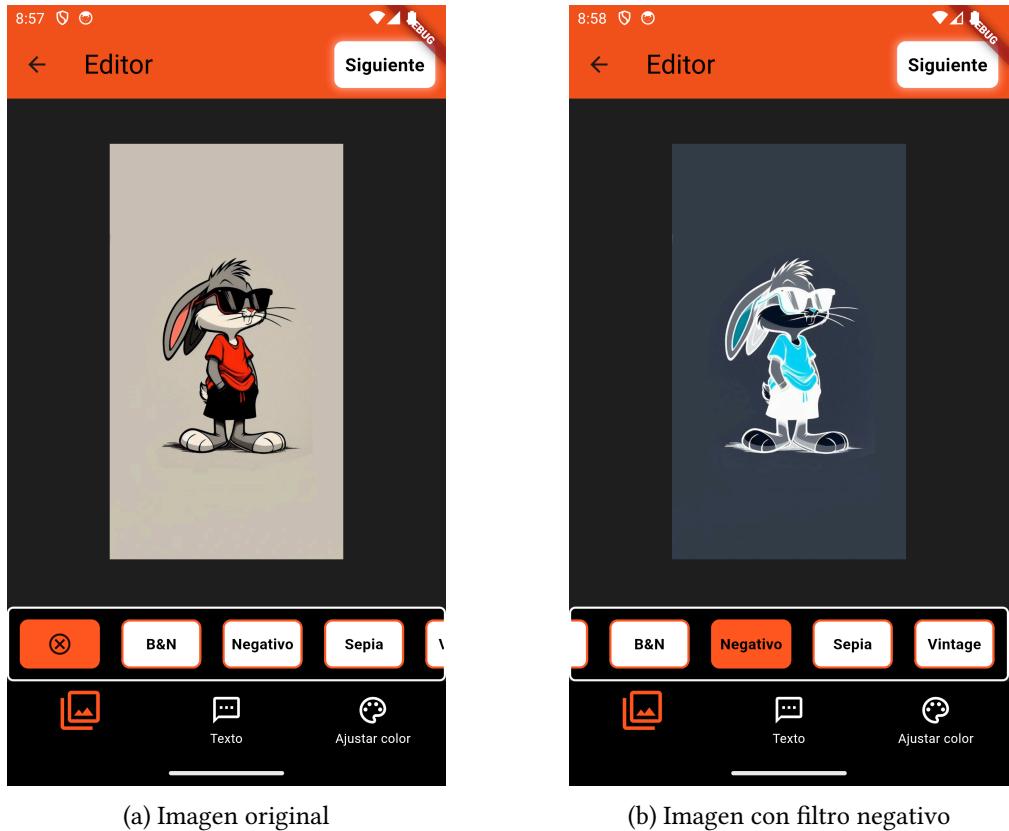


Figura A.10: Filtros

El primer apartado del editor consiste en la aplicación de un filtro a la imagen. Desplazándonos por la barra de navegación desplazable inferior podremos seleccionar el filtro a aplicar, o por el contrario no aplicar ninguno seleccionando la primera opción.

### Añadir textos

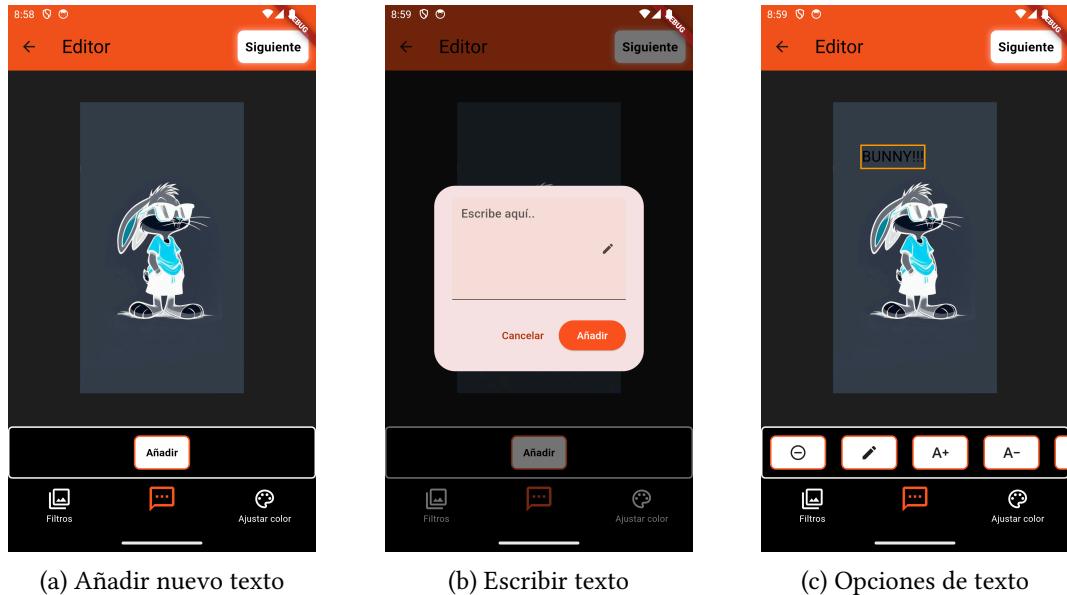


Figura A.11: Añadir y editar textos

Pulsando sobre la opción "Texto" en la parte inferior de la pantalla accederemos a la adición de textos sobre la imagen. Si pulsamos en "Añadir" aparecerá un *pop up* en el que podremos escribir nuestro texto. Una vez confirmado el texto, veremos que este aparece seleccionado encima la imagen, sobre la cual podremos desplazarlo

Mientras el texto esté seleccionado, podremos editarlo con las opciones de la barra de navegación desplazable inferior, en la que se nos permite editar el texto, hacerlo más grande o más pequeño, ponerlo en negrita y en cursiva y eliminarlo. Para deseleccionar un texto simplemente tendremos que pulsar sobre la imagen fuera del texto, lo que permitirá añadir otro texto. Para volver a seleccionar un texto tendríamos que pulsar sobre él mientras estamos en la opción de "Texto".

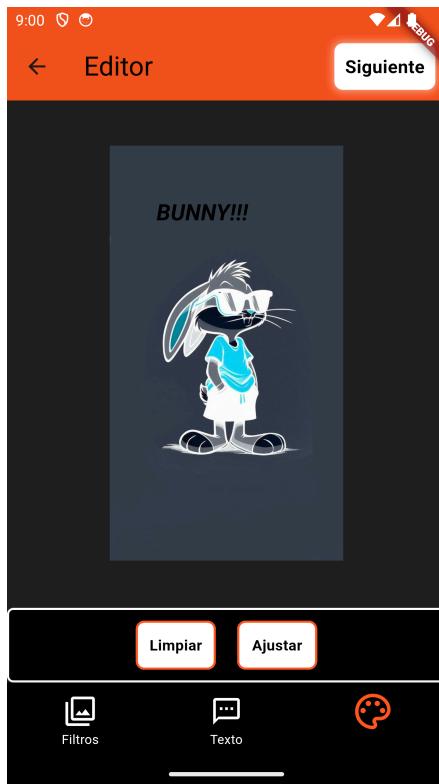
**Capa de color**

Figura A.12: Añadir capa de color

Para añadir una capa de color a la imagen tendremos que seleccionar la opción "Ajustar color" en la parte inferior derecha de la pantalla. En esta sección pulsaremos "Limpiar" si queremos eliminar la capa de color que hayamos añadido previamente, y si queremos añadir una nueva capa pulsaremos en "Ajustar".

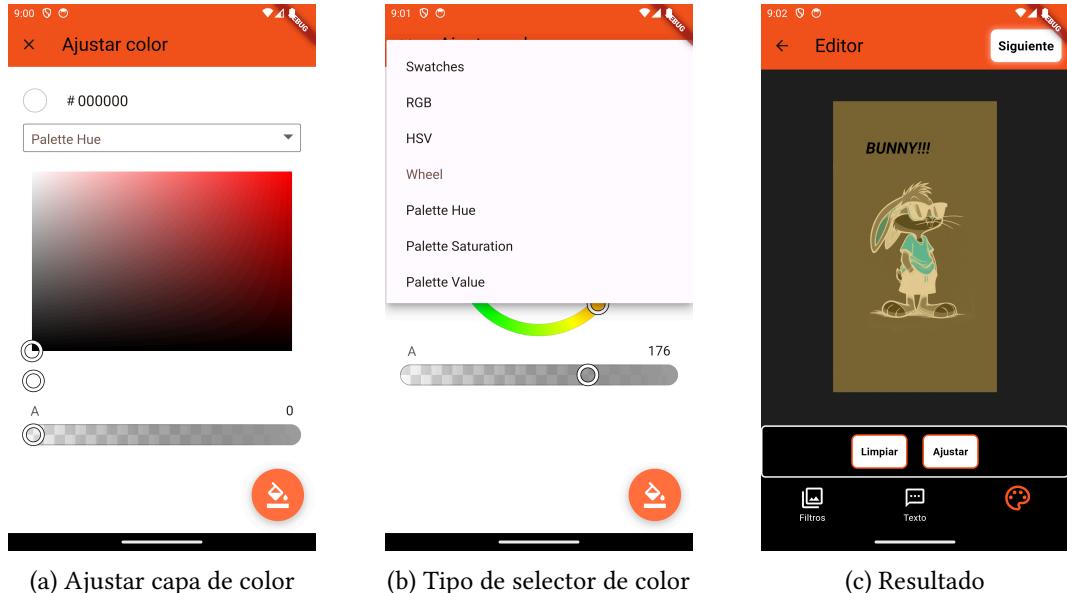


Figura A.13: Aplicación de capa de color

Tras pulsar en "Añadir" se nos llevará a una pantalla donde podremos escoger el color de la capa y su opacidad. En el selector desplegable de la parte superior podremos escoger el tipo de selector de color que debe aparecer en pantalla, habiendo varias opciones. Podemos definir ese color manualmente a través del selector escogido o introducir su código hexadecimal en el campo de texto en la parte superior de la pantalla. Para aplicar la capa de color pulsaremos en el símbolo flotante de la parte inferior derecha de la pantalla.

### A.4.3 Recortar imagen

Para pasar a la sección de recorte de la imagen pulsaremos en "Siguiente" en la parte derecha del *banner* superior.

#### Recortar

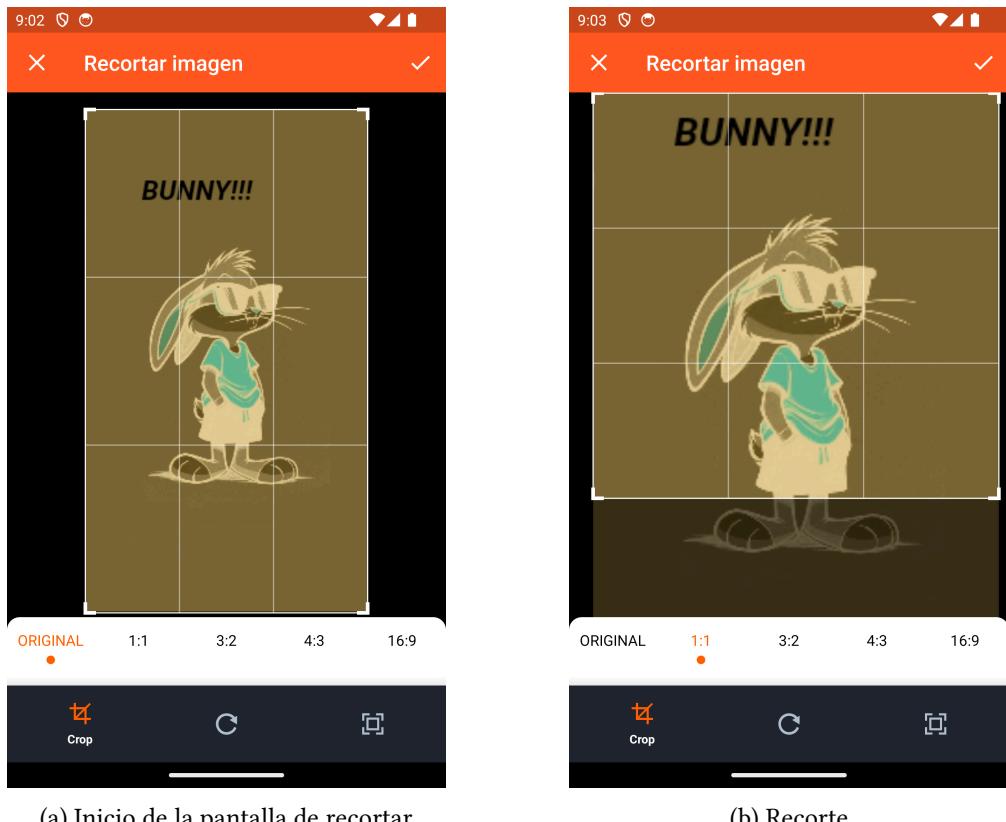


Figura A.14: Recortar

Al entrar a esta sección veremos que tenemos varias opciones en la parte inferior de la pantalla. La primera opción que aparece seleccionada nos permite recortar la imagen de forma libre o siguiendo una proporción de dimensiones predeterminada, en función del tipo de recorte seleccionado en la barra de navegación desplazable.

**Girar**

Figura A.15: Girar imagen

En la segunda opción de la parte inferior de la pantalla se nos permite girar la imagen en cualquier sentido.

### Ajustar escala

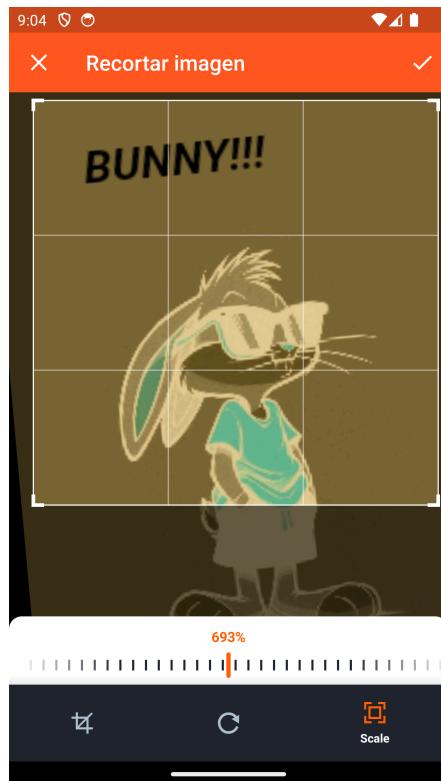


Figura A.16: Ajustar escala

En la última opción de la parte inferior de la pantalla se nos permite hacer y deshacer zoom en la imagen para ajustar la escala de la misma.

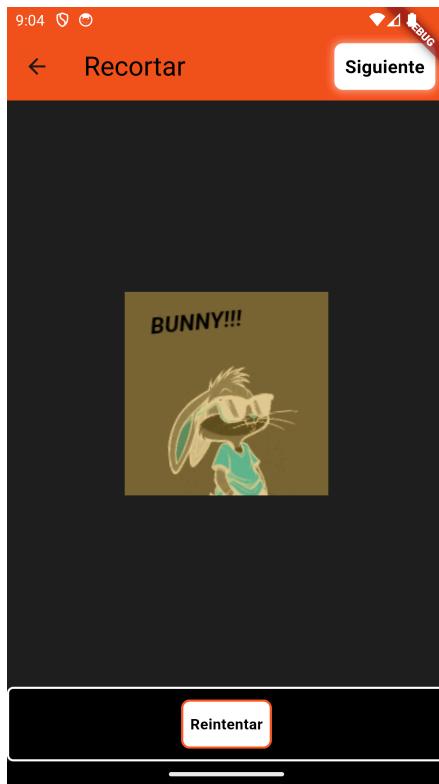
**Reintentar**

Figura A.17: Resultado de recorte

Una vez hayamos acabado de recortar la imagen pulsaremos en el ícono de la parte superior izquierda de la pantalla y se nos redirigirá a la visualización del resultado del recorte (figura A.17). En esta pantalla podremos pulsar en "Reintentar" para rehacer el recorte de la imagen o en "Siguiente" para pasar al siguiente apartado.

#### A.4.4 Previsualización

##### Añadir descripción

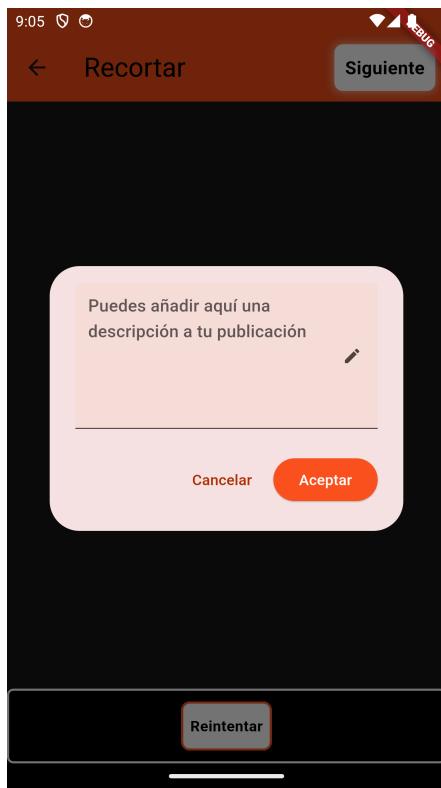


Figura A.18: Añadir descripción

Antes de poder previsualizar la publicación se nos solicita que introduzcamos una descripción para la misma, la cual podemos dejar vacía.

### Previsualizar publicación



Figura A.19: Previsualización de la publicación

Tras confirmar la descripción se nos redirige a la pantalla de previsualización de la publicación. Aquí podremos pulsar en "Editar descripción" para cambiar la descripción de la publicación, en "Links" para añadir y modificar los links asociados, o en el ícono de la parte derecha del banner superior para confirmar la publicación.

**Añadir *links***Figura A.20: Añadir *links*

Tras pulsar sobre "Links" aparecerá un *pop up* en el que podemos asociar hasta tres enlaces a la publicación. Cada enlace cuenta con la posibilidad de tener un título.

**Definir privacidad**

Figura A.21: Definir privacidad

Tras pulsar en el ícono de la parte derecha del banner superior aparecerá un *pop up* en el cual podremos seleccionar la privacidad de nuestra publicación. Si pulsamos en "Publicar" se nos redirigirá a nuestro perfil y podremos ver la imagen publicada.

## A.5 Perfiles de usuario

Para acceder a nuestro perfil tendremos que pulsar sobre la opción "Perfil" de la barra de navegación inferior. En cambio, para acceder a perfiles de otros usuarios podemos hacerlo desde la búsqueda de usuarios, la referencia al perfil en las publicaciones, desde las listas de amigos o desde las notificaciones. Cómo se visualiza el perfil propio es diferente a cómo se visualizan perfiles ajenos.

### A.5.1 Perfil propio

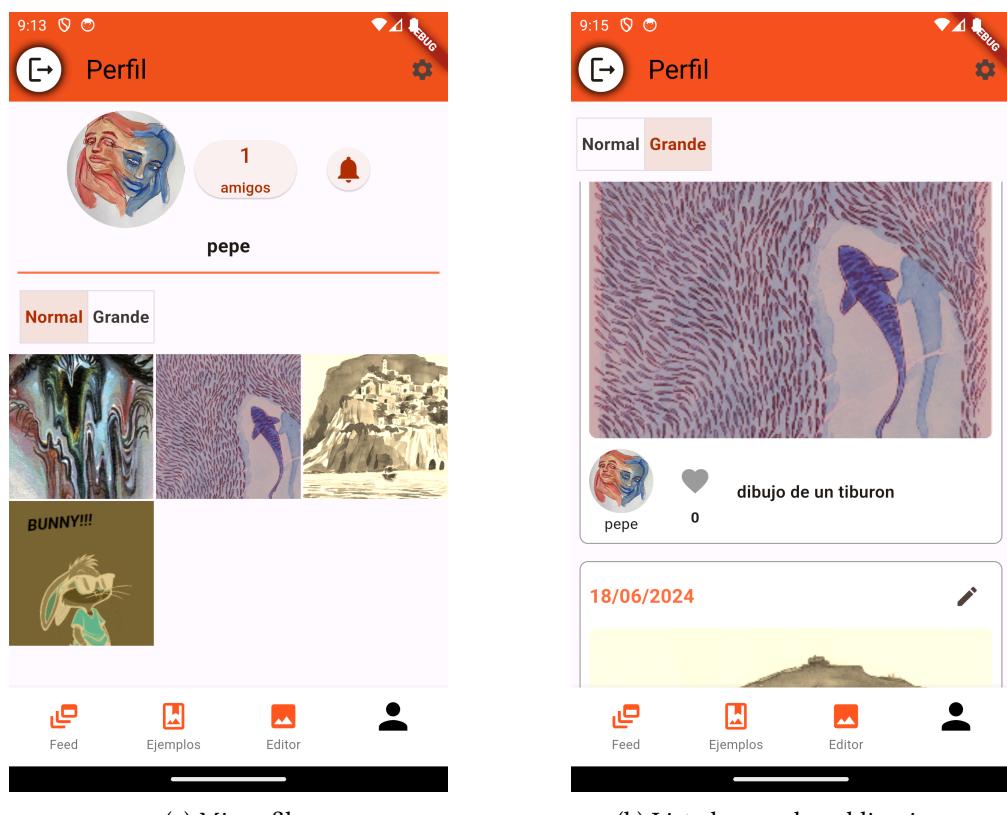


Figura A.22: Perfil propio

Al entrar a nuestro perfil podemos ver nuestra información y acceder a diferentes secciones. Hay un selector de modo de visualización dos opciones: "Normal" y "Grande", siendo "Normal" la opción predeterminada. Si pulsamos sobre la opción "Grande" veremos el listado de publicaciones en el mismo formato que el *feed* de la aplicación.

## Editar imagen

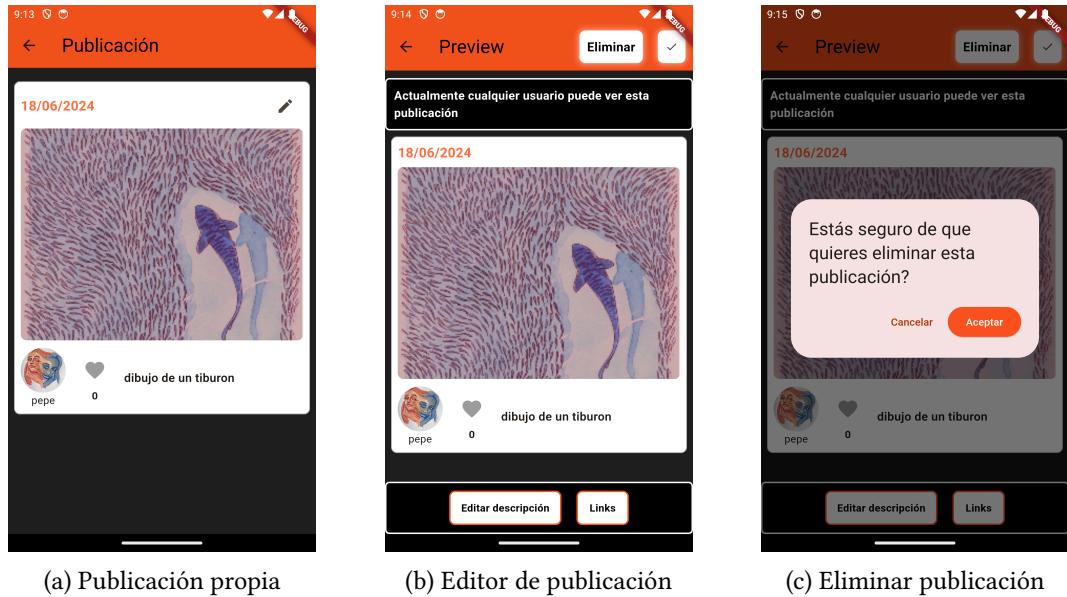


Figura A.23: Editar publicación

Si pulsamos sobre una publicación teniendo el perfil en formato "Normal" se nos redirigirá a la pantalla de la figura A.23a, donde podremos ver los detalles de esa publicación.

Pulsando en el icono del lápiz que aparece en la parte superior derecha de nuestras publicaciones podremos editarlas. En la pantalla de edición se permiten las mismas configuraciones que en la pantalla de previsualización de publicación. Adicionalmente, se puede eliminar la publicación pulsando en el botón "Eliminar" que aparece en el *banner* y posteriormente confirmando la eliminación.

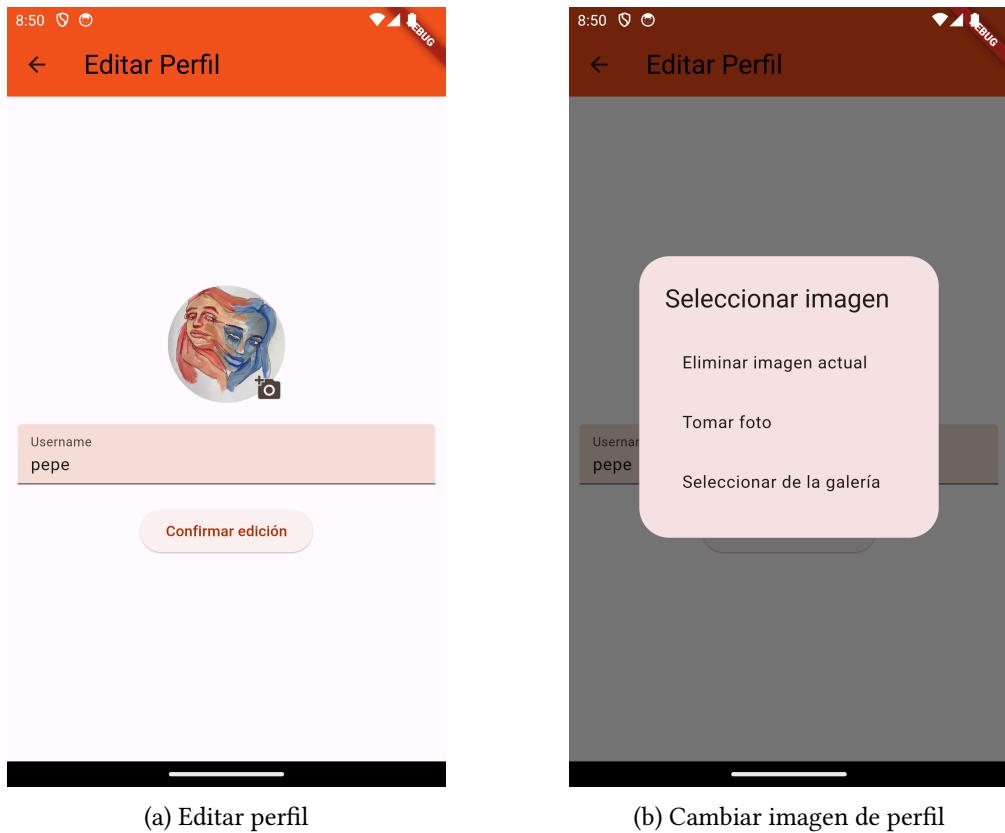
**Editar perfil**

Figura A.24: Editar perfil

Si en nuestro perfil pulsamos sobre el ícono del engranaje en la parte derecha del *banner* superior se desplegarán dos opciones: "Editar perfil" y "Configurar cuenta". Entrando en la opción "Editar perfil" podremos editar nuestra imagen de perfil (pulsando sobre el ícono de la cámara en la parte inferior derecha de la imagen de perfil) y nuestro nombre de usuario (editando el campo de texto).

### Configurar cuenta

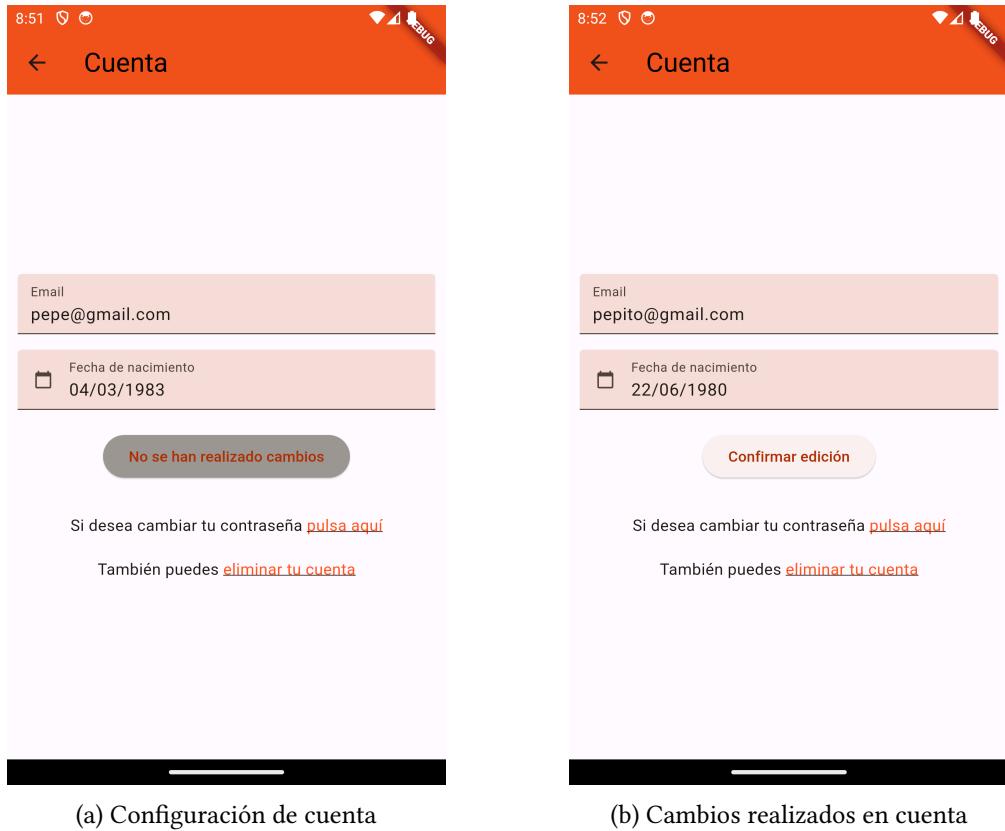


Figura A.25: Configurar cuenta

Pulsando en la opción "Configurar cuenta" del ícono del engranaje accedemos a la pantalla en la que podemos modificar nuestro correo electrónico y nuestra fecha de nacimiento. El botón de confirmar se activará cuando realicemos cambios en estos campos. Si hemos modificado el *email* se nos llevará a la pantalla de inicio de sesión, y no podremos entrar en nuestra cuenta con el nuevo *email* hasta que no lo verifiquemos.

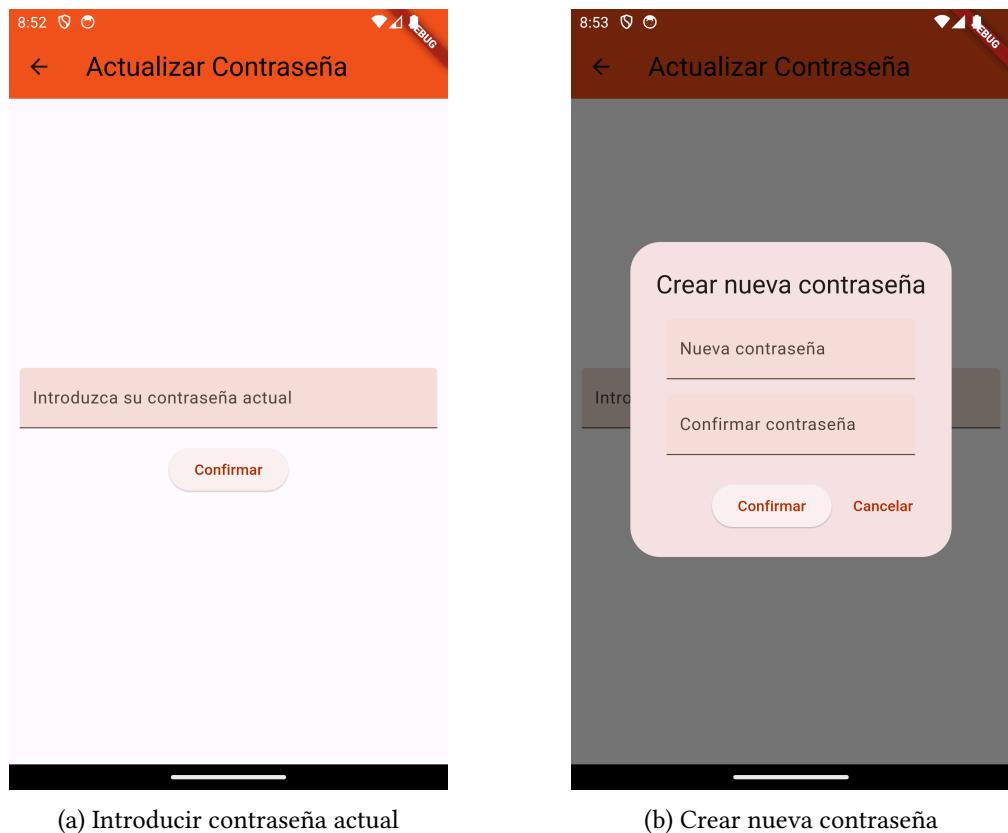
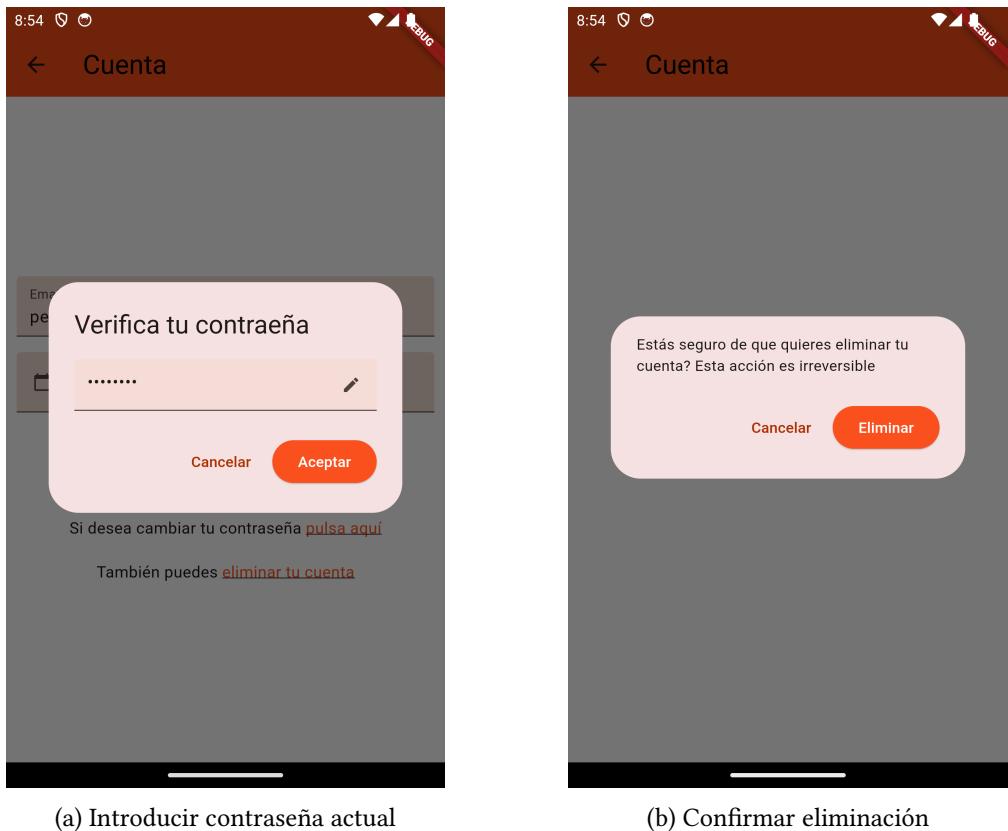


Figura A.26: Cambiar contraseña

Si en la pantalla de configuración de cuenta pulsamos sobre el texto "pulsa aquí" accederemos a otra pantalla en la que inicialmente se nos solicitará nuestra contraseña actual. A continuación, si la contraseña es correcta, aparecerá un *pop up* en el que podremos definir nuestra nueva contraseña.



(a) Introducir contraseña actual

(b) Confirmar eliminación

Figura A.27: Eliminar cuenta

Si por el contrario en la pantalla de configuración de cuenta pulsamos sobre "eliminar tu cuenta" aparecerá un *pop up* que nos solicita nuestra contraseña actual y a continuación, si la contraseña es correcta, aparecerá otro pop up para confirmar la eliminación de la cuenta.

## Notificaciones

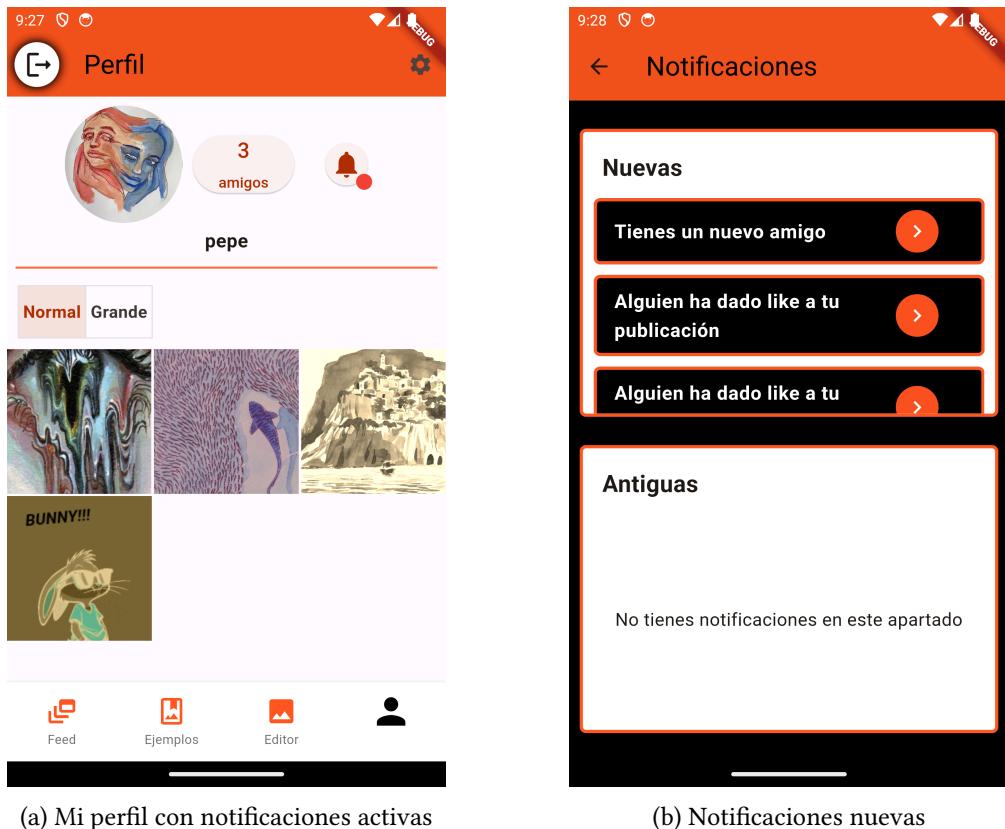


Figura A.28: Notificaciones activas

Para entrar a la sección de notificaciones tendremos que pulsar sobre el ícono de la campana en la parte superior derecha del perfil. Cuando tengamos nuevas notificaciones sin leer aparecerá un círculo rojo al lado del ícono de las notificaciones. La pantalla de notificaciones se divide en dos apartados: *notificaciones nuevas* y *antiguas*. Al entrar a esta pantalla, las notificaciones nuevas se marcarán como leídas.

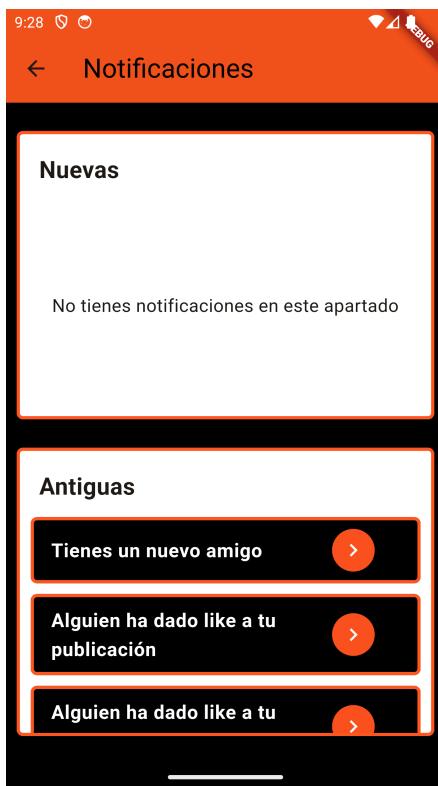


Figura A.29: Notificaciones antiguas

Al leer las notificaciones nuevas, si volvemos a nuestro perfil habrá desaparecido el círculo rojo del ícono de notificaciones, y si volvemos a entrar a la sección de notificaciones todas ellas aparecerán en el apartado de notificaciones antiguas.



Figura A.30: Notificaciones de valoración

Si pulsamos sobre una notificación de seguimiento, se nos redirige al perfil del usuario que ha empezado a seguirnos. En cambio, si pulsamos sobre una notificación de valoración se desplegará un *pop up* en el que podremos seleccionar si ir al perfil del usuario que hizo la valoración o a la publicación valorada.

### A.5.2 Perfil ajeno

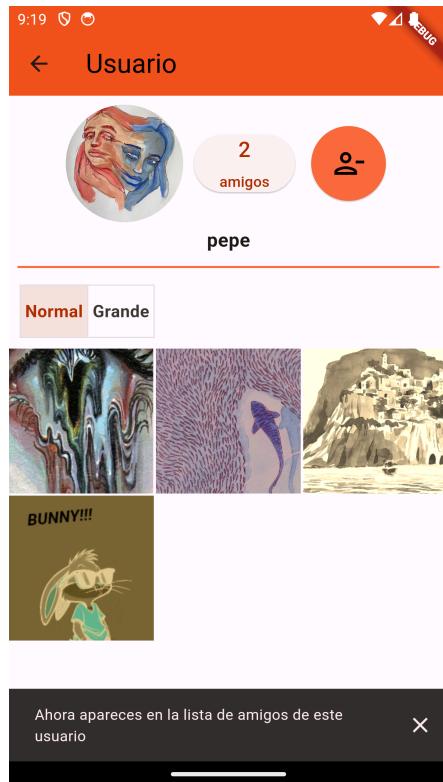


Figura A.31: Perfil de otro usuario al que seguimos

Al entrar al perfil de otro usuario, no veremos el botón para ir a notificaciones ni el de ajustes, pero sí veremos el botón para seguirlo o dejarlo de seguir.

### Seguir y dejar de seguir

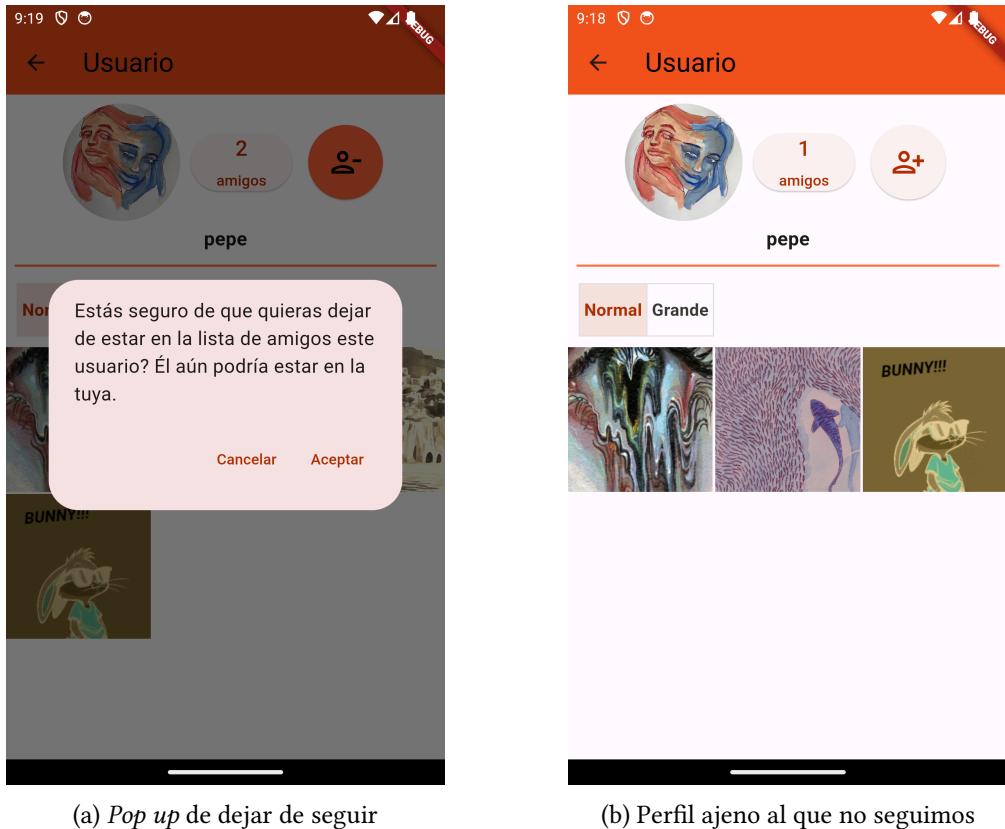


Figura A.32: Dejar de seguir usuario

Si seguimos al usuario del perfil, al pulsar sobre el ícono de la parte superior derecha aparecerá un *pop up* para confirmar o cancelar. Si por el contrario no seguimos al usuario, al pulsar en el ícono simplemente empezaremos a seguirlo. Sólo podremos ver las publicaciones del usuario cuya privacidad está definida como "para amigos" si seguimos al usuario.

### A.5.3 Amigos

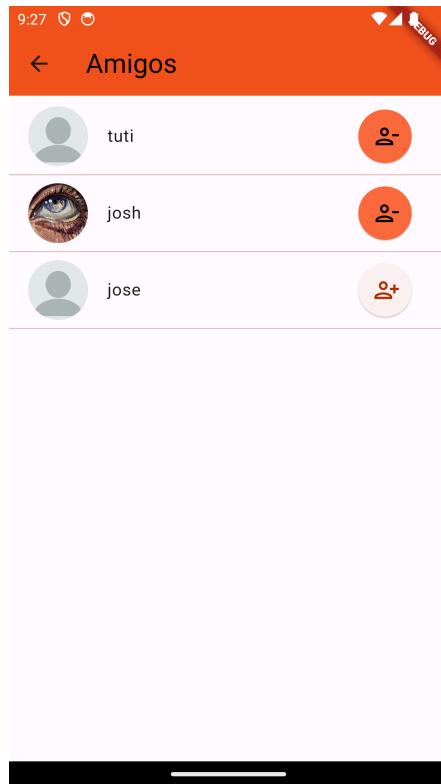


Figura A.33: Listado de amigos

Si en un perfil, sea el nuestro o el de otro usuario, pulsamos sobre el botón de "amigos", accederemos a su listado de amigos o usuarios que le siguen. Desde este listado desplazable verticalmente podremos acceder a los perfiles de usuario y seguirlos o dejarlos de seguir.

# **Lista de acrónimos**

---

**API** *Application Programming Interface.* 1, 7, 9–11, 38, 40, 44, 46

**HTTP** *HyperText Transfer Protocol.* 9–11, 46, 54

**IA** Inteligencia Artificial. 5

**IDE** *Integrated Development Environment.* 10

**JSON** *JavaScript Object Notation.* 8, 9, 50

**MVC** *Model-View-Controller.* 44, 50, 65

**RAM** *Random Access Memory.* 17

**SDK** *Software Development Kit.* 8

**SMS** *Short Message Service.* 8

**URL** *Uniform Resource Locators.* 9

# Glosario

---

**bugs** Problema en un programa de computadora o sistema de software que desencadena un resultado indeseado.. [11](#)

**endpoint** Dirección de una API, o bien un backend que se encarga de dar respuesta a una petición.. [11](#)

**framework** Esquema o marco de trabajo que ofrece una estructura base para elaborar un proyecto con objetivos específicos, una especie de plantilla que sirve como punto de partida para la organización y desarrollo de software.. [9](#)

# Bibliografía

---

- [1] “Lanzamiento de microsoft paint.” [En línea]. Disponible en: <https://blogthinkbig.com/la-historia-de-paint-de-paintbrush-a-paint-3d#:~:text=El%20nacimiento%20de%20Paint,no%20era%20un%20programa%20nuevo>.
- [2] “Lanzamiento de adobe photoshop.” [En línea]. Disponible en: [https://es.wikipedia.org/wiki/Adobe\\_Photoshop](https://es.wikipedia.org/wiki/Adobe_Photoshop)
- [3] “VSCO.” [En línea]. Disponible en: <https://www.vSCO.co/>
- [4] “Picsart.” [En línea]. Disponible en: <https://picsart.com/es/>
- [5] “Firestore database.” [En línea]. Disponible en: <https://firebase.google.com/docs/firestore?hl=es>
- [6] “Firebase authentication.” [En línea]. Disponible en: <https://firebase.google.com/docs/auth>
- [7] “Firebase storage.” [En línea]. Disponible en: <https://firebase.google.com/docs/storage>
- [8] “Unsplash.” [En línea]. Disponible en: <https://unsplash.com/>
- [9] “Dart.” [En línea]. Disponible en: <https://dart.dev/>
- [10] “Flutter.” [En línea]. Disponible en: <https://aws.amazon.com/es/what-is/flutter/>
- [11] “Https.” [En línea]. Disponible en: <https://www.cloudflare.com/es-es/learning/ssl/what-is-https/>
- [12] “Android studio.” [En línea]. Disponible en: <https://developer.android.com/studio?hl=es-419>
- [13] “Google chrome.” [En línea]. Disponible en: [https://www.googleadservices.com/pagead/aclick?sa=L&ai=DChcSEwivhs3wseqGAxXxmIMHHCNsC\\_0YABAAGgJlZg&](https://www.googleadservices.com/pagead/aclick?sa=L&ai=DChcSEwivhs3wseqGAxXxmIMHHCNsC_0YABAAGgJlZg&)

ase=2&gclid=CjwKCAjwps-zBhAiEiwALwsVYapPIKAPHhZhmKyzZA9av\_IYE7SQeG7N-GdJAKVmsK5YGahUQWk18BoCf7sQAvD\_BwE&ohost=www.google.com&cid=CAESVuD2o-F2QaUegIpkgDj0opKGo8hqe\_EjD4aBx-qmqA7faUG\_BxE2rQqcLhxNic6D4t2QqQ7\_h2h2bIxzyQIHADGfck3V-nPU47zGjOl1W9hBGo7noGdX&sig=AOD64\_09f65Vf\_txeiz1EpWQRF2\_ovKc\_Q&q&nis=4&adurl&ved=2ahUKEwig-MXwseqGAXUPhP0HHbiBCSIQ0Qx6BAgOEAE

- [14] “Git.” [En línea]. Disponible en: <https://www.git-scm.com/>
- [15] “Overleaf.” [En línea]. Disponible en: <https://es.overleaf.com/>
- [16] “Metodologías ágiles y no ágiles.” [En línea]. Disponible en: <https://www.positivo.pro/blog/metodologias-agiles-vs-tradicionales/>
- [17] “Scrum.” [En línea]. Disponible en: <https://proyectosagiles.org/que-es-scrum/>
- [18] “Roles en scrum.” [En línea]. Disponible en: <https://www2.deloitte.com/es/es/pages/technology/articles/roles-y-responsabilidades-scrum.html>
- [19] “Sprints en scrum.” [En línea]. Disponible en: <https://openwebinars.net/blog/que-es-un-sprint-scrum/>
- [20] “Sueldo medio desarrollador en españa.” [En línea]. Disponible en: <https://es.talent.com/salary?job=programador+junior#:~:text=El%20salario%20programador%20junior%20promedio,hasta%20%E2%82%AC%202027.000%20al%20a%C3%B3lo.>
- [21] “Sueldo medio jefe de proyecto en españa.” [En línea]. Disponible en: <https://es.jooble.org/salary/scrum-master#:~:text=%C2%BFCu%C3%A1nto%20dinero%20puedo%20ganar%20como%20Scrum%20master%20por%20hora%20en,de%20habilidades%20de%20Scrum%20master.>
- [22] “Historias de usuario.” [En línea]. Disponible en: <https://miro.com/es/agile/que-es-historia-usuario/>
- [23] “Patrones y principios software.” [En línea]. Disponible en: [https://www.autentia.com/wp-content/uploads/libros/SoftwareDesign\\_PrincipiosyPatrones-Autentia.pdf](https://www.autentia.com/wp-content/uploads/libros/SoftwareDesign_PrincipiosyPatrones-Autentia.pdf)
- [24] “Patrones mvc.” [En línea]. Disponible en: <https://codigofacilito.com/articulos/mvc-model-view-controller-explicado>
- [25] “Patrón observer.” [En línea]. Disponible en: <https://refactoring.guru/es/design-patterns/observer>

- [26] “Principios solid.” [En línea]. Disponible en: <https://profile.es/blog/principios-solid-desarrollo-software-calidad/>
- [27] “Patrón yagni.” [En línea]. Disponible en: <https://codeyourapps.com/principio-yagni-no-vas-a-necesitarlo/>
- [28] “Patrón dry.” [En línea]. Disponible en: <https://codeyourapps.com/el-principio-dry-no-te-repitas/>
- [29] “Patrón kiss.” [En línea]. Disponible en: <https://www.quelinka.com/blog/post/principio-kiss-ejemplos>