



2021

Desarrollo de Aplicaciones Multiplataforma

Proyecto Fin de Grado

CIFP Virgen de Gracia

Instadroid

Carlos González Díez

Índice de contenidos

PRESENTACIÓN DEL PROYECTO	3
1. Explicación resumida	3
2. Estudio de mercado y valor del producto.....	4
PLANIFICACIÓN DE TAREAS Y ESTIMACIÓN DE COSTES	8
1. Planificación y organización de tareas.....	8
1.1. Planificación	8
1.2. Uso de GitFlow como flujo de trabajo	11
1.3. Wiki del proyecto.....	12
2. Estimación de costes, gestión de riesgos y herramientas utilizadas	13
2.1. Estimación de costes	13
2.2. Herramientas a utilizar	15
2.3. Prevención de riesgos	17
ANÁLISIS DE LA SOLUCIÓN.....	20
1. Análisis y especificación de requisitos	20
1.1. Requisitos funcionales	20
1.2. Requisitos no funcionales	22
1.3. Requisitos de información	23
2. Análisis de escenarios (casos de uso)	24
DISEÑO DE LA SOLUCIÓN	27
1. Diseño de la interfaz de usuario y prototipos.....	27
1.1. Introducción	27
1.2. Prototipado de Instadroid	29
2. Diseño de la persistencia de la información.....	46
2.1. Diagrama de clases	46
2.2. Diagrama de clases de Instadroid.....	48
2.3. Diseño de la persistencia de la información	50
2.4. Diagrama de la persistencia de la información Instadroid.....	55
3. Diseño de la arquitectura del sistema.....	56
3.1. Arquitectura de un sistema.....	56
3.2. Arquitectura del sistema Instadroid	57
IMPLEMENTACIÓN DE LA SOLUCIÓN	60
1. Justificación tecnológica.....	60
2. Aspectos esenciales de la implementación	70
TESTEO Y PRUEBAS DE LA SOLUCIÓN.....	101
1. Plan de pruebas.....	101
1.1. Plan de pruebas Instadroid	103
1.2. Realización de pruebas	105
3. Solución a problemas encontrados	121
LANZAMIENTO Y PUESTA EN MARCHA	121
1. Aspectos relevantes del despliegue.....	121
2. El despliegue de Instadroid.....	122
3. Manual de uso	126
VALORACIÓN Y CONCLUSIONES	126
BIBLIOGRAFÍA/WEBGRAFÍA.....	128

PRESENTACIÓN DEL PROYECTO

1. Explicación resumida

La solución que voy a implementar es Instadroid, una red social para compartir fotografías y momentos de una persona para cualquier dispositivo móvil del mercado, por lo que se deberá utilizar cualquier herramienta que nos permita desarrollar una aplicación multiplataforma (Android o IOS).

La aplicación consiste principalmente en una línea de tiempo en la que podremos ver las fotografías que siguen los otros usuarios a los que seguimos. En esta fotografía podremos realizar acciones como dar un me gusta o añadir un comentario. De igual manera, podremos ver las interacciones que los demás usuarios de la aplicación hacen sobre la fotografía. Estas interacciones se actualizarán en tiempo real.

De igual manera, podremos subir nuestras fotos (ya sea echándolas con la cámara o eligiéndolas de nuestra galería). Además, podremos añadir una serie de retoques en forma de filtros a la hora de subir nuestra foto.

De todas las fotos, incluidas las nuestras, podremos ver la localización desde la que ha sido subida en un mapa.

Además, podremos acceder a nuestro perfil, el que podremos cambiar distintos aspectos como nuestro nombre de usuario, descripción y foto de perfil. De igual forma, podremos mostrar un código QR correspondiente a nuestro perfil para que otros usuarios puedan escanearlo y agregarnos como amigos.

Si además queremos encontrar usuarios o publicaciones tendremos un buscador disponible que nos ayudará a encontrar lo que necesitemos. De igual manera podremos tener una conversación (chat) con otros usuarios.

La aplicación nos mandará notificaciones para que siempre estemos al día con las últimas publicaciones que suben nuestros amigos, o si nos mandan un mensaje, etc.

Además, no tendremos que preocuparnos por crearnos un usuario con contraseña, ya que la aplicación contará con un sistema de autenticación mediante otras redes como Google, Facebook o Twitter.

Para terminar, en términos de diseño la aplicación móvil va a seguir los patrones de diseño actuales (Material Design e IOS).

Para poder llevar a cabo todo esto de forma correcta necesitamos un backend que nos provea de los datos que necesitamos en la aplicación. Para ello, la aplicación servidor deberá almacenar de forma remota todos los datos de imágenes, usuarios, etc. Estos datos pueden almacenarse en una base de datos relacional o No-SQL.

Las peticiones hacia el servidor (ya sea una API REST o Firebase) deben ser todas seguras con mecanismos como JWT o el propio sistema de autenticación de Firebase.

Finalmente, ambas aplicaciones van a estar desplegadas (en el caso de la aplicación servidor si fuera necesario con Docker, AWS o Heroku y en el caso de cliente se encontrarán publicadas en las App Store), para poder ser accesibles desde cualquier momento y lugar.

2. Estudio de mercado y valor del producto

Un estudio de mercado consiste en una iniciativa empresarial con el fin de hacerse una idea sobre la viabilidad comercial de una actividad económica¹. Este estudio tiene un objetivo económico, es decir, su objetivo es generar beneficio económico con la actividad económica cuya validez queremos probar en el mercado.

Esta investigación busca anticipar la respuesta de los clientes potenciales y la competencia ante un producto o servicio concreto, por lo que no solo se hacen estudios de mercado cuando queremos emprender una aventura empresarial, si no siempre que queramos probar nuestros productos o servicios, saber cómo podemos mejorarlo, de qué manera posicionarlos en el mercado, etc.

De esta manera, con un estudio de mercado bien realizado, conoceremos el perfil y el comportamiento de nuestros clientes, la situación del mercado o industria a la que nos dedicamos, cómo trabaja nuestra competencia, etc. Incluso podemos llegar a descubrir nuevas necesidades que no conocíamos de nuestro público objetivo.

Un estudio de mercado se apoya sobre 4 pilares fundamentales²:

- La información del sector
- Conocer a nuestro target (público objetivo)
- Conocer a nuestra competencia

¹ <https://blog.mailrelay.com/es/2018/09/06/estudio-de-mercado>

² <https://obsbusiness.school/es/blog-investigacion/marketing-y-comunicacion/como-hacer-un-estudio-de-mercado-en-4-pasos>

- Análisis DAFO (Debilidades, Amenazas, Fortalezas y Oportunidades)

De acuerdo con esto, voy a realizar un estudio de mercado sobre la aplicación Instadroid, pasando por cada uno de estos puntos:

■ Sector

Está claro que el sector en el que cae por completo la aplicación Instadroid es el sector de las redes sociales, un sector que ha cambiado de forma transversal el sistema económico mundial. Analizando este sector³, puedo concluir que las redes sociales se han consolidado como una herramienta básica para el manejo de transacciones, pero también en una fuente de fraude y corrupción debido a la todavía poca confianza que existe en este medio.

Pero lo realmente destacable es como estas herramientas se han integrado en casi todos los ámbitos del proceso productivo empresarial (marketing y comunicación, gestión del conocimiento empresarial, internacionalización, precios, captación de recursos humanos, mejora de la competitividad), etc.

En conclusión, es un sector que ofrece una serie de oportunidades increíbles a las empresas y aplicaciones que realmente innovan sobre las ya conocidas redes sociales que están establecidas.

■ Público objetivo (target)

Se trata de un recorte demográfico, socioeconómico y comportamental de un grupo compuesto por los futuros consumidores del producto o servicio de la empresa.⁴ Para crear un público objetivo se toman en cuenta los siguientes datos:

- Edad
- Sexo
- Ubicación
- Formación educativa
- Poder adquisitivo
- Clase social

³ Análisis de la lectura Carlos Merino Moreno, *Consecuencias sociales y económicas de las redes sociales*, Universitat Oberta de Catalunya (www.uoc.edu). Ubicación del recurso:
http://openaccess.uoc.edu/webapps/o2/bitstream/10609/54384/2/Consecuencias%20sociales%20y%20economicas%20de%20las%20redes%20sociales_M%C3%B3dulo1.pdf

⁴ <https://www.rdstation.com/es/blog/publico-objetivo-cliente-ideal-buyer-persona/>

- Hábitos de consumo

En nuestro caso, podemos definir el público objetivo como personas sin distinción de sexos de entre 15 y 35 años, de cualquier parte del mundo, sin importar la información educativa, poder adquisitivo o clase social.

En realidad, podría decirse que se trata de cualquier persona con un smartphone y que esté acostumbrado a manejarse en redes sociales (ya que cada vez intentar acotar ese rango de edad es más complicado).

■ Análisis de la competencia

No hay que realizar un análisis muy exhaustivo para saber que el sector al que queremos entrar está plagado de competencia. Y esta competencia son gigantes de las redes sociales.

En primer lugar, tenemos a nuestro competidor más directo, que sería Instagram, ya que las funcionalidades que ofrecemos son similares. En este caso el cliente ha hecho énfasis en la noción de cercanía que debe transmitir la App, por lo que podremos diferenciarnos de Instagram a través de este valor.

En segundo lugar, no como competidores directos, pero si como competidores secundarios, tenemos:

- ➔ WhatsApp. Su funcionalidad de mensajería puede perjudicar a nuestra función de mensajes privados entre los distintos usuarios.
- ➔ Facebook. Por razones obvias. Un gigante de las redes totalmente asentado.
- ➔ Pinterest. Aunque sea para compartir más que fotos, la gente utiliza mucho Pinterest para buscar inspiración, por lo que una importante cuota de mercado se va en Pinterest.

■ Análisis DAFO



Se trata de una sencilla herramienta de análisis estratégico muy extendida en la toma de decisiones de todo tipo de organizaciones y empresas⁵. Sus siglas significan:

- Debilidades: Se trata de las desventajas que tenemos respecto a nuestros competidores, cosas que deberíamos mejorar.
- Amenazas: Entrada de nuevos competidores al mercado o un mercado saturado son amenazas para la empresa.
- Fortalezas: Qué es lo que hacemos mejor que nuestros competidores, en qué nos diferenciamos de ellos.
- Oportunidades: Nichos del mercado no ocupados, estrategias para mejorar la eficiencia y reducir los costes.

En el caso de Instadroid nuestro análisis DAFO, sería el siguiente:

➔ **Debilidades:**

La principal debilidad con la que nos encontramos es el poco conocimiento del negocio con respecto a la competencia y que somos algo nuevo, en un mercado gobernado por gigantes, por lo que costará ganarse una cuota en el mercado.

➔ **Amenazas:**

La principal amenaza para Instadroid es la saturación del mercado al que nos dirigimos, debido a que hay muchas alternativas que ofrecen lo mismo.

➔ **Fortalezas:**

Nos diferenciamos de nuestros competidores por nuestro valor de cercanía, por lo que ese valor es una fortaleza que ellos no tienen y que hace que el público

⁵ <https://www.infoautonomos.com/plan-de-negocio/analisis-dafo/>

objetivo nos perciba como una aplicación más cercana y que les acerca a sus amigos.

➔ **Oportunidades:**

En el sector de las aplicaciones hay oportunidades constantes de mejora, por lo que tendremos que ir actualizando tanto nuestra aplicación como a nuestros programadores constantemente para ofrecer lo último al cliente

Conclusión extraída del análisis de mercado

Gracias a este análisis de mercado podemos saber cómo debe actuar Instadroid en el mercado, es decir, el valor que vamos a ofrecer a nuestros usuarios:

Instadroid debe de ser una aplicación que fomente el valor de la cercanía de sus usuarios, alejándose de sus competidores. Debe de ser una aplicación que ponga al cliente en el centro y siempre se renueve para ofrecerle a éste lo mejor. Ese es nuestro valor y nuestra posición en el mercado.

PLANIFICACIÓN DE TAREAS Y ESTIMACIÓN DE COSTES

1. Planificación y organización de tareas

1.1. Planificación

Para la planificación y organización de tareas he utilizado, principalmente 2 herramientas. En primer lugar, GitKraken boards⁶, que ofrece la posibilidad de crear un tablero Kanban para poder organizar las tareas del proyecto. Además, estas tareas se pueden descomponer en tareas más pequeñas en modo de CheckList. Lo mejor de GitKraken es que podemos establecer una fecha en la que queremos terminar cada tarea, pudiendo luego acceder a una vista Timeline en la que podemos ver de forma muy clara como van a avanzar nuestras tareas en el tiempo. Este es el timeline para el proyecto Instadroid:



⁶ <https://www.gitkraken.com/boards>

Como se puede ver las tareas de presentación del proyecto ya realizadas ya han pasado su fecha de entrega.

A continuación, este es un detalle de la estimación temporal de las tareas (de momento está hecho para una hipotética entrega el 8 de Enero, pudiendo ser modificado si finalmente esta entrega es posteriormente)

29 de octubre

Hitos que finalizar:

- Planificación de tarareas y estimación de costes. Detalle de la tarea:

2. PLANIFICACIÓN DE TAREAS Y ESTIMACIÓN DE COSTES
In Progress

Description

Carlos González Díez created on 18 oct. 18:40 • edited

- Planificación y organización de tareas
- Estimación de costes y recursos
- Herramientas usadas
- Gestión de riesgos

- Análisis de la solución. Detalle de la tarea:

3. ANÁLISIS DE LA SOLUCIÓN
In Progress

Description

Carlos González Díez created on 18 oct. 18:42 • edited

- Requisitos
- Casos-de-uso
- Escenarios

2 de diciembre

Hito que finalizar:

- Diseño de la solución. Detalle de la tarea:

4. DISEÑO DE LA SOLUCIÓN
To Do

Description

Carlos González Díez created on 18 oct. 18:44 • edited

- Diseño y prototipado de interfaz
- Diseño persistencia de datos
- Diseño arquitectura

31 de diciembre

Hito que finalizar:

- Implementación de la solución. Detalle de la tarea:

5. IMPLEMENTACIÓN DE LA SOLUCIÓN
To Do

Description

Carlos González Díez created on 18 oct. 18:45 • edited

- Justificación tecnológica
- Aspectos esenciales
- Desarrollo de la funcionalidad indicada

4 de enero

Hito que finalizar:

- Testeo y pruebas de la solución. Detalle de la tarea:

The screenshot shows a task detail view titled "6. TESTEO Y PRUEBAS". The "Description" section lists two items: "Plan de pruebas" and "Solución a bugs". Both items have checkboxes next to them, indicating they are tasks.

5 de enero

Hito que finalizar:

- Lanzamiento. Detalle de la tarea:

The screenshot shows a task detail view titled "7. LANZAMIENTO". The "Description" section lists three items: "Despliegue", "Manual de uso", and "Manual de uso". The last item is repeated. All three items have checkboxes next to them.

7 de enero

Hito final:

- Valoración conclusiones y bibliografía. Detalle de la tarea:

The screenshot shows a task detail view titled "8. VALORACIÓN, CONCLUSIONES Y BIBLIOGRAFÍA". The "Description" section lists two items: "Valoración y conclusiones" and "Bibliografía y recursos". Both items have checkboxes next to them.

Finalmente, este es el **aspecto final** del tablero:

The screenshot shows the final state of the GitKraken board. It has four columns: "To Do", "In Progress", "Done", and "Add Column".

- To Do:** Contains cards for "4. DISEÑO DE LA SOLUCIÓN" (SB-5), "5. IMPLEMENTACIÓN DE LA SOLUCIÓN" (SB-6), "6. TESTEO Y PRUEBAS" (SB-7), "7. LANZAMIENTO" (SB-8), and "8. VALORACIÓN, CONCLUSIONES Y BIBLIOGRAFÍA" (SB-9). Each card shows progress counts: 8/11, 29/11, 6/12, 14/12, and 16/12 respectively.
- In Progress:** Contains cards for "2. PLANIFICACIÓN DE TAREAS Y ESTIMACIÓN DE COSTES" (SB-3) and "3. ANÁLISIS DE LA SOLUCIÓN" (SB-4). Each card shows progress counts: 25/10 and 25/10 respectively.
- Done:** Contains cards for "1. PRESENTACIÓN DEL PROYECTO" (SB-2) and "Add a card". The first card shows a progress count of 16/10.
- Add Column:** A button to add a new column to the board.

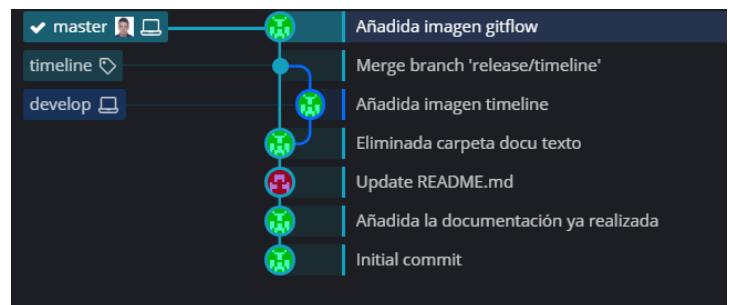
Este tablero es para **organización personal**, ya que me gusta más que todas las demás alternativas, de igual manera se puede consultar su versión pública en el repositorio en GitHub del proyecto⁷:

1.2. Uso de GitFlow como flujo de trabajo

El flujo de trabajo GitFlow se basa principalmente en que nuestro repositorio tendrá dos ramas, una llamada develop y la máster. La rama develop es donde convergen todas las ramas de desarrollo y la rama máster es nuestra rama principal o de producción⁸.

En el repositorio de Instadroid igualmente se utiliza GitFlow como flujo de trabajo:

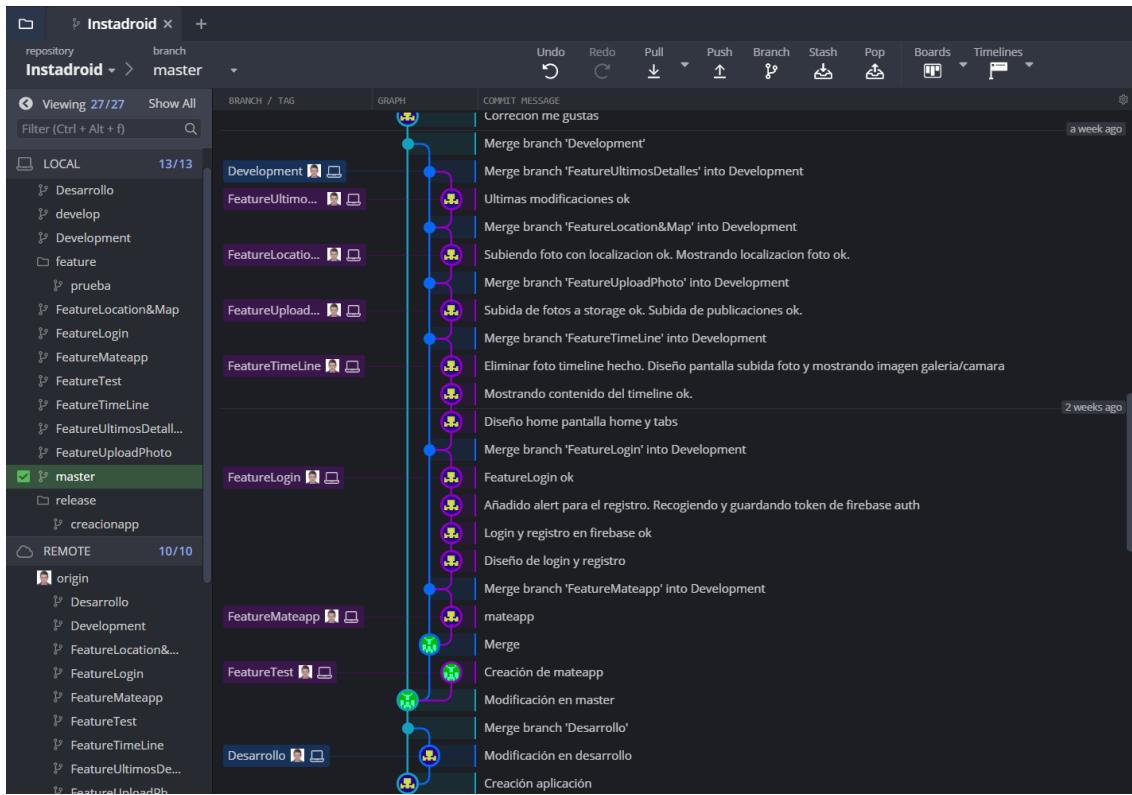
Como se puede observar, se ven las dos ramas de trabajo (develop y master).



⁷ Enlace al tablero Kanban del repositorio: <https://github.com/CarlosDez23/Instadroid/projects/1>

⁸ <https://gfourmis.co/gitflow-sin-morir-en-el-intento/>

Esta es una captura de mi árbol de Git, siguiendo este flujo de trabajo:



Como se puede observar, se crea una rama llamada Development y a partir de ella se van realizando otras en función de las funcionalidades que vamos añadiendo a la aplicación (Las distintas ramas Feature seguido del nombre de la funcionalidad). Cuando estas funcionalidades se finalizan se mezclan con la rama Development y al finalizar el proyecto, la rama Development se mezcla con la rama master, que es la rama principal del repositorio. Estas ramas están presentes tanto en mi repositorio local como en el repositorio remoto en Github.

1.3. Wiki del proyecto⁹

Para finalizar, como herramienta para documentar el proyecto, he utilizado la Wiki de GitHub que nos permite añadir información acerca del proyecto en el propio repositorio. Esta herramienta que ofrece GitHub se trata de un espacio en el que podemos alejar la documentación¹⁰, por lo que toda la documentación hecha y por hacer estará reflejada ahí.

⁹ Dirección de la wiki del repositorio de Instadroid: <https://github.com/CarlosDez23/Instadroid/wiki>

¹⁰ <https://docs.github.com/es/free-pro-team@latest/github/building-a-strong-community/about-wikis>

2. Estimación de costes, gestión de riesgos y herramientas utilizadas

2.1. Estimación de costes

La estimación de costes es un proceso que consiste en desarrollar una aproximación a los recursos monetarios necesarios para completar las actividades del proyecto. Se trata de una predicción basada en la información que tenemos disponible en un momento dado.¹¹

En el marco que nos atañe, el desarrollo de software es muy complicado determinar el tiempo y esfuerzo que tomará realizar un proyecto. Esto se debe a que la estimación de costes de software es esencialmente difícil¹², y los seres humanos somos bastante malos prediciendo resultados absolutos. No hay dos proyectos software iguales, por lo que no hay una “Biblia” acerca de cómo realizar estimaciones de costes sobre estos proyectos. Hay que tener en cuenta la cantidad de parámetros que conforman la existencia de los proyectos software, que muchas veces lo que parecen problemas simples se vuelven mucho más problemáticos cuando se los aborda desde el punto de vista técnico.

Con la llegada de las metodologías ágiles se ha cambiado la forma en la que se monetizan las aplicaciones, ya que estas no buscan establecer un coste inicial acerca de cuánto va a costar realizar x aplicación, si no que van estimando el coste del proyecto conforme el proyecto se va creando. De manera que, los contratos ágiles se enfocan en lo siguiente¹³:

- **Paquetes de trabajo a precio fijo:** Como bien sabemos el proyecto se divide en subtareas que se van lanzando, por lo que a cada subtarea se le establece un precio en función del esfuerzo que le ha costado a la empresa desarrollarla (no cuesta lo mismo realizar un chat que realizar un formulario)
- **Terminación anticipada:** De forma general, significa que el cliente puede terminar el proyecto antes de lo esperado si ya se ha entregado un gran porcentaje del producto y se encuentra satisfecho con el resultado.
- **Cambios flexibles:** En un proyecto ágil, el cambio debe de ser un elemento transversal. De forma que, cualquier cambio que el equipo de desarrollo acuerde hacer sobre el producto (mientras tenga el mismo valor) debe ser correctamente

¹¹ <https://capitulo7pmbok.wordpress.com/gestion-de-los-costos-del-proyecto/>

¹² <https://www.toptal.com/agile/estimacion-de-costos-de-software-en-gestion-de-proyectos-agiles>

¹³ <https://www.toptal.com/agile/estimacion-de-costos-de-software-en-gestion-de-proyectos-agiles>

estimado entrega en cuestión que en ese momento el equipo de desarrollo esté completando.

Como Instadroid se va a llevar a cabo siguiendo la metodología ágil, necesitamos tener esto muy en cuenta para realizar las distintas estimaciones sobre las entregas que se van a ir remitiendo a cliente.

En primer lugar, partimos de la base de que el coste de una aplicación móvil de tipo red social oscila entre los 15.000 € y 25.000€¹⁴.

Pero eso son equipos de desarrolladores ya profesionales y con experiencia. En este caso y siendo realista con mi situación, voy a cobrar 20 €/hora (la media de un programador multiplataforma freelance en España es de 25€/hora¹⁵). Mi estimación de tiempo son 70 horas (más las ya echadas entre formación y documentación ya presentada que son 40, 110 horas).

20€/hora x 110 horas	2200 €
----------------------	--------

Esta estimación sería para una parte de la App (la que vamos a implementar). Si hubiera que desarrollar la App entera yo estimaría unas 200 horas, lo que situaría el precio de mi salario en los 4000 €.

Las licencias ya dispongo de ellas (VScode es open source, GitHub ya lo tengo pagado, al igual que el Office 365).

Solo sería necesario que aparte del coste de mis horas el cliente pagará:

- 25€ por un emulador de IOS que necesito para que la aplicación sea multiplataforma
- El despliegue de la aplicación en las tiendas de aplicaciones. Esto irá a cuenta del cliente, especialmente en Google que puede pagar para toda la vida porque puede que su empresa quiera seguir subiendo aplicaciones bajo la misma identidad corporativa. En este caso: 25 € para subir la app al Play Store de Google¹⁶ y 95€/año para subirla y mantenerla en el AppStore de IOS¹⁷.

¹⁴ <https://www.yeeply.com/blog/cuanto-cuesta-crear-una-app/>

¹⁵ <https://www.masquenegocio.com/2017/07/19/desarrollar-app-espana/>

¹⁶ https://cincodias.elpais.com/cincodias/2015/02/01/lifestyle/1422792260_243066.html

¹⁷ <https://www.yeeply.com/blog/cuanto-cuesta-crear-una-app/>

- En el caso de despliegue en AWS y de Firebase es complicado de cuantificar porque te cobran en función de las peticiones, lecturas, escrituras, autenticaciones, por lo que adjunto el enlace de la tabla de precios como referencia¹⁸.

COSTES TOTALES	2345 €
COSTES TOTALES + IVA	2.837,45 €

Los costes totales están estimados en cuanto al desarrollo de una parte de la aplicación (como vamos a hacer), los costes totales del desarrollo de la aplicación completa serían de: **4145 €**.

Obviamente esto sin contar los costes que tendría el cliente por sus peticiones a AWS y/o Firebase, pero eso es algo que se podría ir estableciendo solo cuando la app esté ya un tiempo en el mercado y se pueda establecer una media mensual de peticiones.

■ Fuentes de financiación

Para financiar este proyecto, como no debo pagar a nadie nada más que mi sueldo solo me financiaré del presupuesto pedido al cliente. Alternativamente, si por ejemplo quisiéramos comprarnos un ordenador nuevo para poder trabajar también con IOS (un Mac), como Freelances podríamos acudir al crowdfunding que son pequeños micromecenazgos, es decir personas anónimas que aportan dinero a nuestro proyecto. Para ello debemos compartir el proyecto en una de las plataformas de crowdfunding¹⁹ que hay por internet. Pero estas personas necesitarán algo a cambio, por lo que les daremos acceso anticipado a la app, convirtiéndose ellos en nuestros testers.

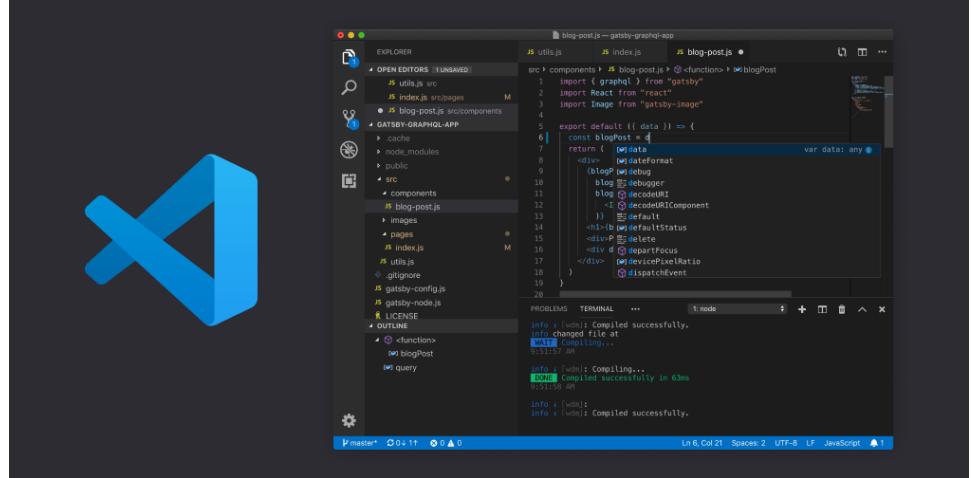
2.2. Herramientas a utilizar

Las herramientas que voy a utilizar son las siguientes:

¹⁸ <https://aws.amazon.com/es/api-gateway/pricing/> y Firebase
<https://firebase.google.com/docs/firestore/billing-example?hl=es>

¹⁹https://es.gofundme.com/start?utm_source=google&utm_medium=cpc&utm_campaign=SE_NonBrand_ES_SP_Tier1&utm_content=Crowdfunding&utm_term=crowdfunding_e_c_&gclid=Cj0KCQjw59n8BRD2ARIsAAmgPmKjQZb86o5AS4Ou6fHTVCDAxu1bgE08UCz9TddQBb7GX_xu_pV_G5UaAh3IEALw_wcB

Para el desarrollo de código, vamos a utilizar Visual Studio Code, es una solución de código abierto, fácilmente extensible y muy ligero debido a que se trata de un editor de texto y no de un IDE. Está desarrollado para Windows, Linux y macOS. Incluye depuración, control integrado con Git y se puede extender con diversas extensiones para soportar los lenguajes que necesitemos. Este editor de texto se basa en Electron un framework que permite utilizar Chromium y Node.js para desarrollar aplicaciones de escritorio²⁰.



Para todas las tareas de ofimática utilizaremos Office 365, un conjunto de programas que incluyen Word, Excel, Outlook (cliente de correo) y demás herramientas de sobra conocidas²¹.



Para poder probar las aplicaciones utilizaremos los emuladores de iPhone y Android.

Para el control de código fuente utilizaremos Git y GitHub. Para gestionar estas herramientas utilizaremos GitKraken, que además incorpora GitKraken boards, que es un gestor de incidencias a través de un tablero Kanban. Esto nos ayudará a descomponer el proyecto en pequeñas tareas y poder planificarlas todas de forma muy ordenada. Para

²⁰ https://es.wikipedia.org/wiki/Visual_Studio_Code

²¹ https://es.wikipedia.org/wiki/Microsoft_Office_365

ponernos un poco en contexto, GitKraken es utilizado por empresas como Netflix, Tesla y Apple²².

Otras herramientas tradicionales son un cliente de correo (cliente de Outlook en Windows 10) y un navegador (Mozilla Firefox).

Además, utilizaré draw.io para realizar los diagramas necesarios y JustInMind para el prototipado de la interfaz de la aplicación (ambas herramientas gratuitas).

2.3. Prevención de riesgos

■ Riesgos laborales

Se entiende por prevención el conjunto de actividades o medidas adoptadas en todas las fases de la actividad de la empresa con el fin de evitar o disminuir los riesgos derivados del trabajo. Estos riesgos del trabajo son riesgos laborales, que es la posibilidad de que un trabajador sufra un determinado daño derivado del trabajo. Estos daños pueden ser enfermedades, patologías o lesiones²³. Para más información acudir a la Ley 31/1995, de 8 de noviembre de 1995, de prevención de riesgos laborales o LPRL²⁴.

En este caso, al ser una empresa de informática hay que prestar especial atención a²⁵:

- Dolores de espalda y otros trastornos musculoesqueléticos
- Fatiga visual

Nuestra acción prevención de riesgos laborales se centrará, principalmente:

- Tener un puesto de trabajo adecuado, silla adecuada, escritorio a altura correcta, teclado separado del monitor, buena iluminación y ventilación. La luz debe ser en su mayoría natural y debe entrar por un lado del trabajador, ya que no es bueno que le dé por detrás y se refleje en el monitor.
- Utilizar herramientas ergonómicas: pantallas con eye-care, teclados con una franja para apoyar correctamente las muñecas.
- Obligación de levantarse y estirar se cada hora.

²² <https://blog.desdelinux.net/gitkraken-un-excelente-cliente-de-git-multiplataforma-para-tu-escritorio/>

²³ <http://www.invassat.gva.es/es/que-es-prevencion-de-riesgos-laborales>

²⁴ <https://www.boe.es/buscar/act.php?id=BOE-A-1995-24292>

²⁵ <https://prevencionugtandalucia.es/riesgos-laborales-en-el-sector-de-la-informatica/>

Fuera de los riesgos específicos del sector hay que prestar siempre especial atención a los sistemas de sistemas para evacuación de incendios (extintores) de acuerdo con las normativas vigentes²⁶.

■ Riesgos específicos de proyectos software

La gestión de riesgos es una **actividad de protección** dentro de la gestión de proyectos, encargada de identificar, mitigar y monitorizar los riesgos que pudieran afectar a la ejecución y viabilidad del proyecto²⁷.

Estos riesgos pueden ser de 3 tipos:

- Riesgos del proyecto: Ponen en peligro al plan, si se producen supondrá un mayor esfuerzo y dinero.
- Riesgos técnicos: Ponen en peligro la calidad del producto final.
- Riesgos del negocio: Ponen en peligro la realización del proyecto, si se cumplen, el proyecto se cancela.

Las principales causas que incrementan el nivel de riesgo en un proyecto son²⁸:

- Caer en alguno de los errores típicos
- Desarrollar sin metodología
- No tener una correcta estimación, evaluación y administración de los riesgos

En Instadroid, se ha establecido una política clara en cuanto a los riesgos a los que nos enfrentamos:

RIESGO	ACCIÓN
Falta de conocimiento de la metodología o sobre algún paso para la consecución del proyecto	Se garantizará la contratación de personal que conozca las tecnologías a implementar. De igual manera se distribuirá material formativo.
Desarrolladores inefficientes	Se procederá a su despido por circunstancias económicas

²⁶ <https://www.preverlab.com/nuevo-reglamento-de-instalaciones-de-proteccion-contra-incendios/>

²⁷ https://es.wikiversity.org/wiki/Gesti%C3%B3n_de_riesgos_de_proyectos_software

²⁸ <http://www.luiscarlosaceves.com/udem/ppd/riesgossoftware.pdf>

Falta de organización del equipo	Se van a utilizar herramientas como GitKraken Boards que favorecen la organización del trabajo en equipo
Poca retroalimentación del cliente	Se le instará a contestar las preguntas que sean necesaria. Utilizaremos metodología ágil para tener este feedback continuo del cliente.
Mala elección de herramientas	Se intentará estar al día sobre nuevas herramientas que nos pudieran ayudar de mejor forma a las actuales. Tendremos segundas opciones, como en el caso del editor cambiar de visual studio code a Atom o Sublime Text
Pérdida de código fuente o archivos	Disponemos de un servidor NAS, en el queharemos copia de todo nuestro trabajo, además lo tendremos en la nube y el servidor hará copia de seguridad todos los días
Conflictos entre miembros del equipo	Se instará a su profesionalidad y en caso contrario se les sancionará siguiendo el artículo 58 del Estatuto de los Trabajadores ²⁹ .
Cambio de la plataforma sobre la que funcionará el software	Habrá que ver qué ha cambiado y cómo nos afecta, por eso mejor no utilizar código nativo y utilizar algún framework
El cliente insiste en involucrarse en cuestiones técnicas	Se le debe explicar amablemente hasta dónde pueden ser útiles sus conocimientos. Si el cliente tiene formación técnica mejor, estará más involucrado
Mala calidad del producto final	Se pospone el lanzamiento y si es necesario se rehace el equipo para ofrecer siempre la máxima calidad en nuestros productos

²⁹ <https://www.cuestioneslaborales.es/estatuto-los-trabajadores/#58>

No hay suficiente gente para el tamaño del proyecto	Se procederá a realizar más contrataciones
Retraso en una entrega	Se debe trabajar hasta terminarla (siempre dentro de lo razonable y dentro de los límites legales). Habrá que descontar del precio ofertado al cliente debido a nuestro retraso. Siempre cumpliremos nuestra palabra.
El usuario final insiste en nuevos requisitos	Se aceptará el mismo y se le informará de los cambios en cuanto al precio acordado. Si el requisito no se puede integrar se le explicará igualmente
Interfaz o código mal implementada/o	Se vuelve a diseñar e implementar
El producto depende las normativas del gobierno	Estar siempre al tanto de estas normativas para estar siempre bajo el amparo de la ley

ANÁLISIS DE LA SOLUCIÓN

1. Análisis y especificación de requisitos

1.1. Requisitos funcionales

Un requisito funcional³⁰ define una función del sistema de software o sus componentes. Una función es descrita como un conjunto de entradas, comportamientos y salidas. Se trata de funcionalidades específicas que un sistema debe cumplir. Estos requisitos establecen los comportamientos que tiene el software que vamos a desarrollar.

En el caso de la aplicación Instradoid, los requisitos funcionales serían los siguientes:

RF1 → El sistema debe permitir al usuario ver una línea de tiempo con las fotografías de los usuarios a los que sigue.

RF2 → El usuario debe poder “dar me gusta” a la fotografía, así como efectuar un comentario sobre la misma.

³⁰ https://es.wikipedia.org/wiki/Requisito_funcional

RF3 → El usuario debe poder ver en tiempo real los comentarios y me gustas que otros usuarios han realizado sobre la publicación.

RF4 → El usuario debe poder ver en un mapa la localización en la que fue subida la fotografía.

RF5 → El usuario debe poder subir una foto a Instadroid, echándola con la cámara o subiéndola desde la galería de su teléfono.

RF6 → El usuario debe poder editar la fotografía antes de subirla, pudiendo aplicar distintos filtros o efectos.

RF7 → El sistema debe permitir al usuario ver una sección con la información de su perfil.

RF8 → El usuario debe poder cambiar su información del perfil, como su nombre, fotografía de perfil y contraseña.

RF9 → El usuario debe poder mostrar su código QR único de usuario.

RF10 → El usuario debe poder agregar a otros usuarios a su lista de amigos a través del escaneo del código QR único de usuario de los demás usuarios.

RF11 → El sistema debe contar con una sección de búsqueda.

RF12 → El usuario debe poder buscar publicaciones por hashtag o usuarios por nombre de usuario.

RF13 → El usuario debe poder agregar a otro usuario mediante su búsqueda y seguimiento.

RF14 → El usuario debería poder tener una conversación privada con otros usuarios.

RF15 → El usuario debe poder identificarse con sus credenciales de otras redes sociales o servicios.

RF16 → El usuario debe poder recibir notificaciones desde la aplicación.

RF17 → El usuario debe poder aceptar las peticiones de amistad que se le envían.

RF18 → El usuario debe poder usar la aplicación en cualquier momento y lugar.

RF19 → El sistema debe controlar que las peticiones son todas seguras.

RF20 → El sistema debe facilitar un sistema de login.

RF21 → El sistema debe facilitar un sistema de registro para los usuarios que no quieran identificarse con los medios expuestos anteriormente.

RF22 → El usuario debe poder añadir un título y hashtags a su publicación.

RF23 → La aplicación cliente del sistema debe seguir los cánones de diseño marcados por cada plataforma.

RF24 → La aplicación debe funcionar en cualquier dispositivo móvil del mercado, sea cual sea su sistema operativo.

RF25 → La experiencia del usuario debe de ser perfecta y debe sentirse cómodo usando la aplicación.

RF26 → El sistema debe de ser escalable, por lo que se debe de actualizar y mejorar constantemente mientras dure el ciclo de vida de la aplicación.

RF27 → La navegación en la aplicación cliente debe incluir una pantalla de publicaciones, una pantalla de perfil, una pantalla en la que se pueda subir una publicación y, por último, una pantalla para realizar búsquedas de usuarios y publicaciones.

RF28 → El usuario debe poder editar (cambiar el título) a una publicación subida.

RF29 → El usuario debe poder borrar una publicación subida.

1.2. Requisitos no funcionales

Un requisito no funcional³¹ es aquel que impone restricciones en el diseño o la implementación como restricciones en el diseño o estándares de calidad. Son propiedades o cualidades que el producto debe tener.

En el caso de Instradoid, los requisitos no funcionales son los siguientes:

RNF1 → El cliente móvil del sistema debe ser desarrollado con algún lenguaje o framework que le permita ser una aplicación multiplataforma, tales como React-Native, Native Script, Electron o similares.

RNF2 → Todos los datos que maneja la aplicación deberán almacenarse remotamente en un servidor.

RNF3 → El consumo de estos datos se hará a través de una API REST propia desarrollada con SpringBoot o similares. También se puede usar Firebase para esta parte de backend.

³¹ https://www.ecured.cu/Requisitos_no_funcionales

RNF4 → El acceso al consumo de estos datos del servidor debe estar securizado bien con JWT si se opta por una API REST propia o mediante el sistema de autenticación de Firebase si se opta por esta última opción.

RNF5 → El almacenamiento de datos se realizará en una base de datos que podrá ser SQL (Postgres), NoSQL (MongoDB o Firebase) o ambas.

RNF6 → El acceso a estos datos y su publicación en las vistas debe de ser lo más rápido posible.

RNF7 → En el caso de optar por una API REST propia, la misma debe de ser completamente testeable sin necesidad de una aplicación cliente, con soluciones como POSTMAN o similares.

RNF8 → El servicio web debe estar desplegado pudiendo utilizar Heroku, Docker, AWS o similar con dicho fin.

RNF9 → La gama de colores de la aplicación especificada por el cliente debe contener azul, blanco y verdes agua sin saturación.

RNF10 → Los métodos de autenticación del usuario son su email y contraseña, número de teléfono, Google, Facebook y Twitter.

1.3. Requisitos de información

Los requisitos de información son aquellos que representan entidades e información relevante con las que el producto software va a operar³².

En el caso de Instadroid, los requisitos de información con los que vamos a trabajar son los siguientes:

RI1 → *Usuario*: Representa a cada una de las personas que utilizan la aplicación. Como información importante del mismo encontramos:

- Id de usuario
- Nombre
- Apellidos
- Fecha de nacimiento
- Nick
- Token de autenticación

³² <https://www.infor.uva.es/~mlaguna/is1/apuntes/2-requisitos.pdf>

- Email

RI2 → *Publicación*: Representa las fotografías que suben los usuarios a Instadroid, junto con sus comentarios y me gustas. De las mismas, vamos a trabajar con estos datos:

- Id de publicación
- Fotografía
- Comentarios (se relaciona con el RI5)
- Número de me gustas
- Posición
- Datos del usuario que la ha subido

RI3 → *Mensaje*: Representa los mensajes que se mandan los usuarios en sus conversaciones privadas. De los mismos la información que vamos a tratar es:

- Id de mensaje
- Emisor (se relaciona con el RI1)
- Destinatario (se relaciona con el RI1)

RI4 → *Amigo*: Representa las relaciones de amistad entre usuarios, de tal forma que los datos a tratar son:

- Id usuario 1 (se relaciona con el RI1)
- Id usuario 2 (se relaciona con el RI1)

RI5 → *Comentario*: Representa un comentario que hace un usuario en una publicación, por lo que los datos a tratar en este caso son:

- Id publicación (se relaciona con el RI2)
- Contenido

2. Análisis de escenarios (casos de uso)

Un caso de uso es la descripción de una acción o actividad. Un diagrama de casos de uso es una descripción de las actividades que deberá realizar alguien o algo con nuestro sistema para llevar a cabo algún proceso.

En nuestro contexto de desarrollo de una aplicación un diagrama de casos de uso representa a un sistema o subsistema como un conjunto de interacciones que se desarrollarán entre casos de uso y entre estos y sus actores en respuesta a un evento que inicia un actor principal. Los diagramas de casos de uso sirven para especificar la

comunicación y el comportamiento de un sistema mediante su interacción con los usuarios u otros sistemas³³.

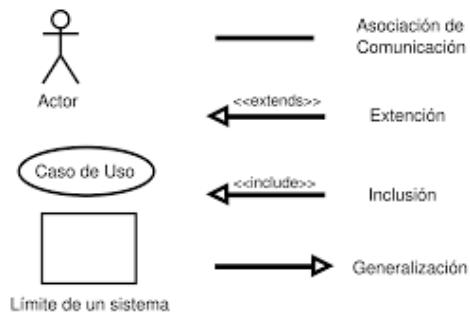
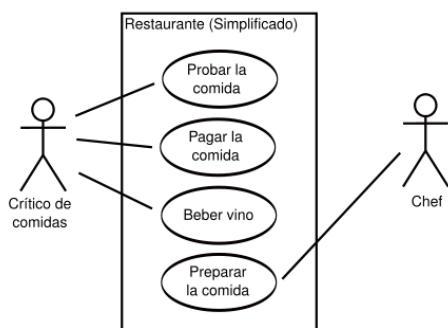


Ilustración: Notación de caso de uso³⁴

Lo que vamos a intentar abordar con nuestro diagrama son los requisitos funcionales obtenidos en la fase de análisis, es decir, cada requisito funcional debería ser un caso de uso o formar parte de uno.

Ilustración: Ejemplo básico de diagrama de casos de uso³⁵

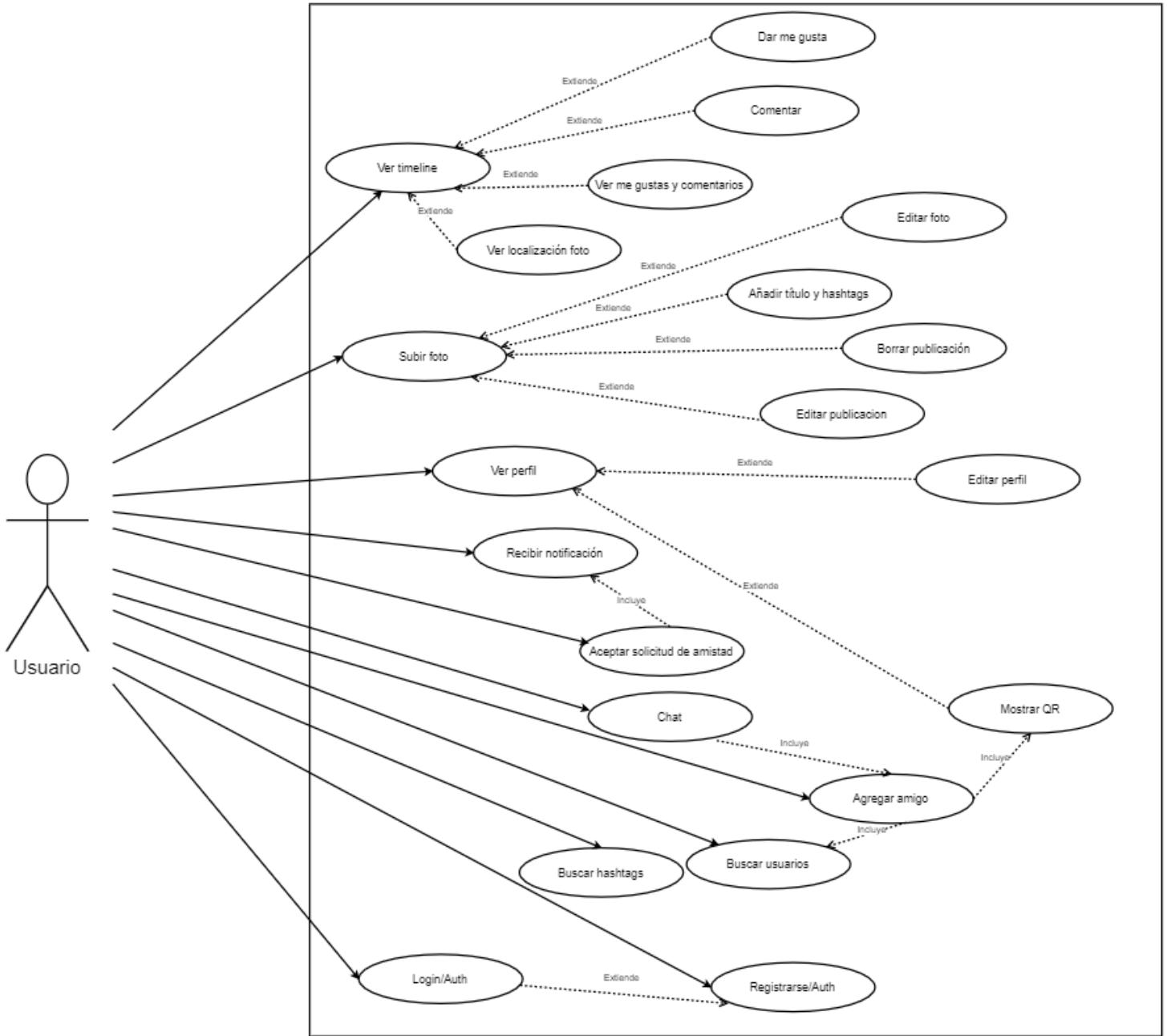


³³ https://es.wikipedia.org/wiki/Caso_de_uso

³⁴ https://upload.wikimedia.org/wikipedia/commons/thumb/9/9e/Notacion_Caso_de_Uso.svg/1200px-Notacion_Caso_de_Uso.svg.png

³⁵ https://upload.wikimedia.org/wikipedia/commons/thumb/8/80/UML_diagrama_caso_de_uso.svg/498px-UML_diagrama_caso_de_uso.svg.png

En el caso de Instadroid, este sería el diagrama de casos de uso para el sistema:



DISEÑO DE LA SOLUCIÓN

1. Diseño de la interfaz de usuario y prototipos

Enlace al vídeo de YouTube donde se muestra el prototipo →
https://www.youtube.com/watch?v=_RwGjg4G_dE

A partir de este punto empezamos con el diseño de la solución. Este proceso juega un papel determinante en el desarrollo, ya que permite a los ingenieros de software producir diversos modelos que:

- Caracterizan la solución a implementar
- Pueden ser analizados y evaluados con el fin de determinar si se satisfacen los requisitos
- Facilitan el examen y evaluación de alternativas
- Sirven para planificar las siguientes actividades de desarrollo

En definitiva, este proceso es la actividad del ciclo de vida del software en la cual se analizan los requisitos de la solución (*¿Qué?*) para producir una descripción de la estructura interna del software que sirva de base para su construcción (*¿Cómo?*)³⁶.

1.1. Introducción

En esta primera parte de nuestro proceso de diseño, nos centraremos en el diseño de la interfaz de usuario o ingeniería de interfaz. Se trata del resultado de definir la forma, función, utilidad, ergonomía, imagen de marca y otros aspectos que afectan a la apariencia externa de las interfaces de usuario en sistemas de todo tipo³⁷.

Se trata de una actividad multidisciplinar que abarca a varias ramas del diseño y del conocimiento³⁸:

- Diseño gráfico
- Diseño industrial
- Diseño web
- Diseño de software

³⁶ <https://www.ctr.unican.es/asignaturas/is1/is1-t04-trans.pdf>

³⁷ https://es.wikipedia.org/wiki/Dise%C3%B1o_de_interfaz_de_usuario

³⁸ <https://www.efectodigital.online/post/2018/04/18/dise%C3%B1o-de-interfaz-de-usuario-ui>

- Diseño de ergonomía

El objetivo de la interfaz de usuario es mantener la interacción con los mismos de la forma más atractiva, centrando el diseño en ellos. Las herramientas principales son los recursos gráficos, los pictogramas, los estereotipos y la simbología.

Es por ello, que toda interfaz de usuario sigue estos 6 principios:

- Familiaridad con el usuario: Se utilizan términos y conceptos que se toman de la experiencia de las personas que más utilizan el sistema
- Consistencia: La interfaz debe ser consistente en el sentido de que las operaciones comparables se activan de la misma forma
- Mínima sorpresa: El comportamiento del sistema no debe provocar sorpresa en los usuarios
- Recuperabilidad: La interfaz tiene que incluir mecanismos para permitir a los usuarios recuperarse de los errores (como la confirmación de actividades destructivas)
- Guía al usuario: Cuando los errores ocurren, la interfaz debe dar feedback significativo al usuario
- Diversidad de usuarios: La interfaz debe proveer características de interacción apropiada para los diferentes tipos de usuarios.

Normalmente, el diseño de interfaces se realiza a través de la elaboración de prototipos (ejecutables o no ejecutables). En nuestro caso vamos a utilizar la herramienta Just In Mind³⁹ de cara a la realización del prototipado para nuestra aplicación Instadroid.

Para ello, en este documento se expondrán los casos de uso más importantes del sistema junto con su navegabilidad reflejada en el prototipo. De esta manera, los requisitos funcionales del cliente se verán quedará bien plasmados en la interfaz de usuario.

³⁹https://www.justinmind.com/?utm_medium=cpc&utm_source=google&utm_campaign=1063145459&utm_term=just%20in%20mind_e&gclid=Cj0KCQiAwMP9BRCzARIsAPWTJ_Hcqj5e74vQ_sa9ZTJ63onwlUPrAMiM_OhzAttpbITmmU3iDLU-MqkaAhQVEALw_wcB

1.2. Prototipado de Instadroid

Antes de mostrar el prototipo realizado para la UI voy a mostrar la paleta de colores utilizada y acordada con el cliente, así como las fuentes utilizadas.

■ Paleta de colores

Se estableció con el cliente que la paleta de colores tendría tonos azules/verdosos marinos, por lo que he utilizado el color “Aquamarine” que se acerca a este requisito de color azul-verdoso:

#5AFFE7

Este color junto con el blanco corriente conforma los dos colores más presentes en la aplicación.

De forma complementaria para acentuar el menú de navegación se ha utilizado:

#37465B

Además, en el login y splash screen se ha utilizado una imagen de fondo que pretende unificar esta paleta de colores (siguiendo una tendencia de diseño muy actual que son los paisajes vectoriales):



Para el texto de la aplicación se ha utilizado un negro común. Como se puede observar (y a petición del cliente, se trata de una paleta de colores suaves y poco sobrecargada.

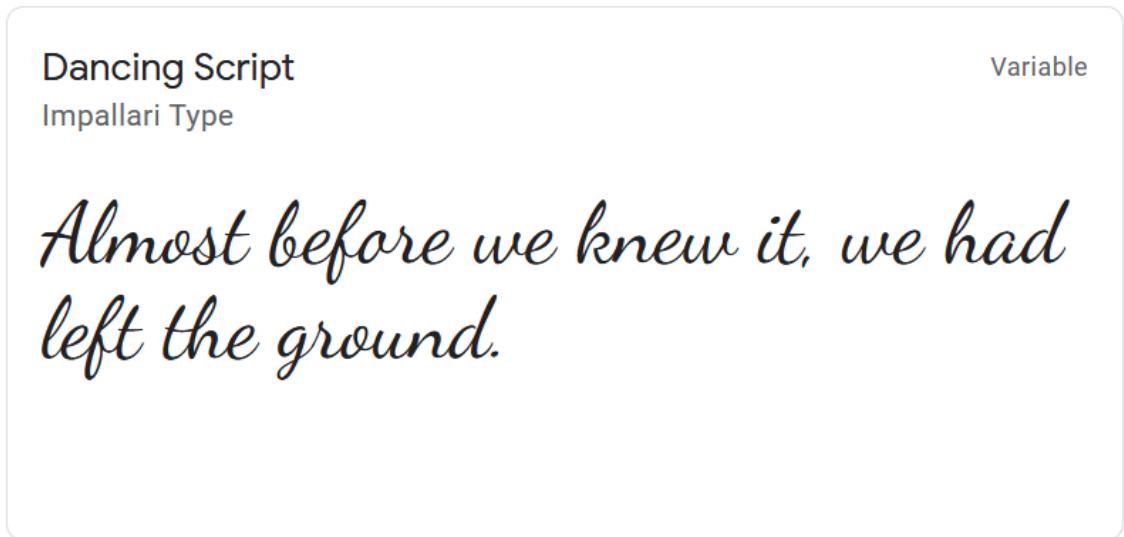
■ Fuente

Como fuentes de la aplicación se han utilizado dos:

- Para los textos normales de la aplicación, la fuente Roboto⁴⁰:



- Para el logotipo de la aplicación que aparece en el login y en el timeline de la aplicación, la fuente Dancing Script⁴¹:



Una vez especificados los recursos de los que se vale el diseño de la aplicación, podemos empezar a ver el prototipo de UI de la misma.

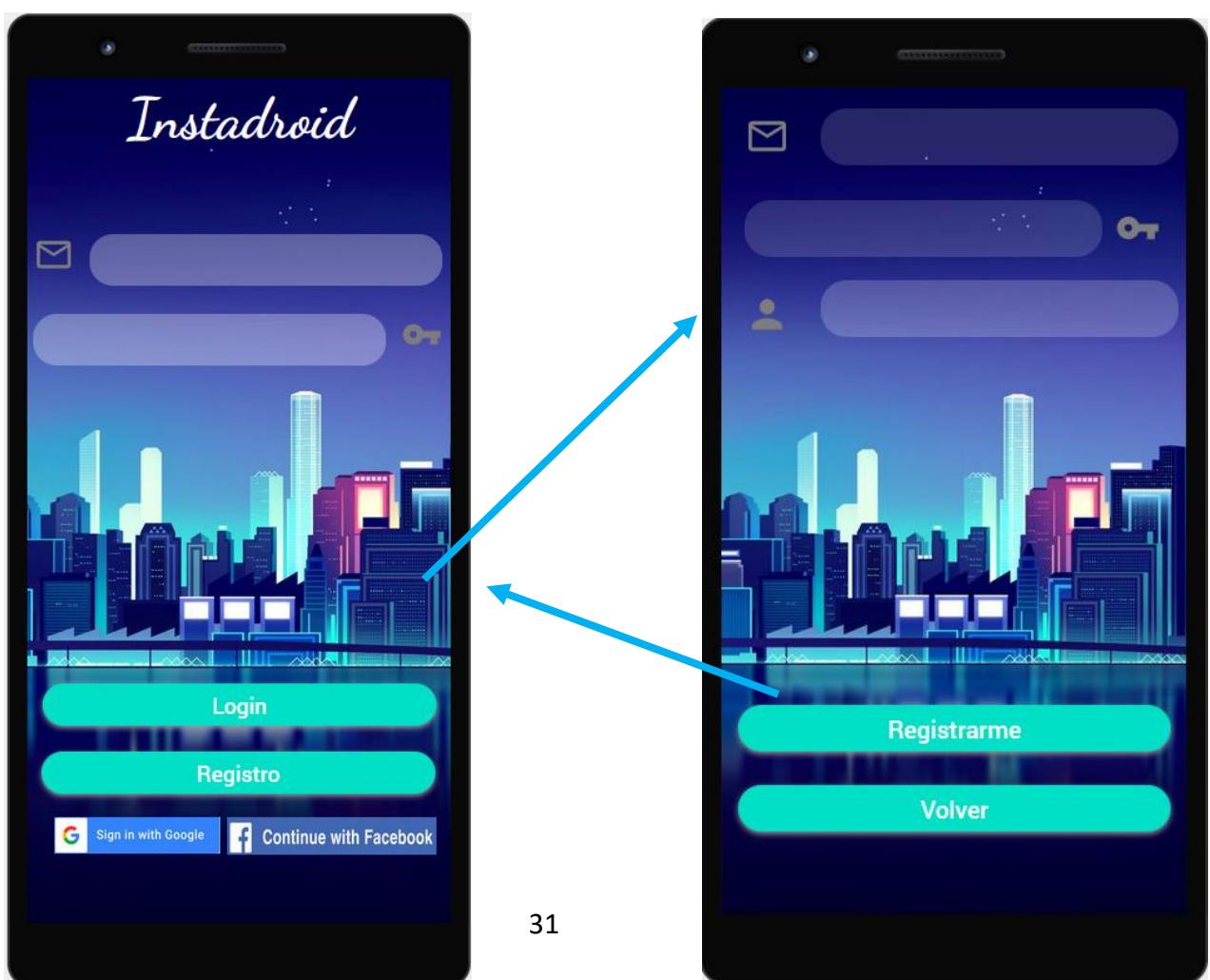
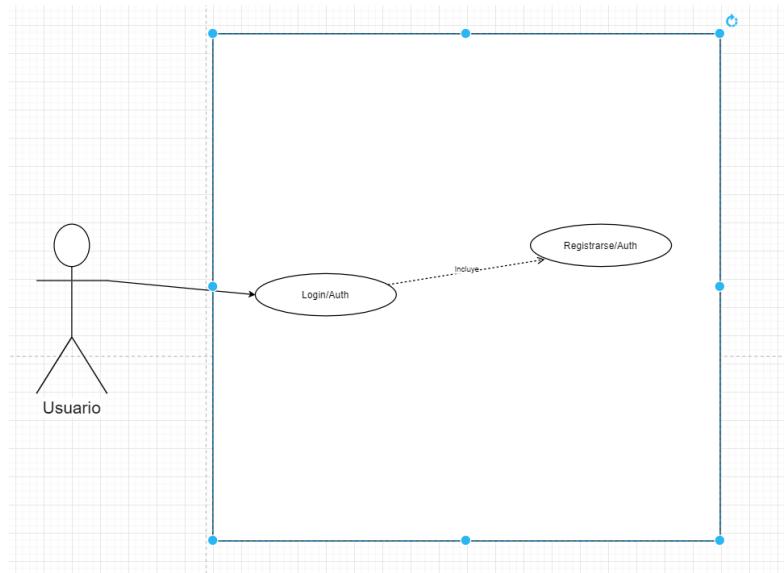
⁴⁰ <https://fonts.google.com/specimen/Roboto>

⁴¹ <https://fonts.google.com/specimen/Dancing+Script?query=dancing>

■ Prototipo de UI de Instadroid

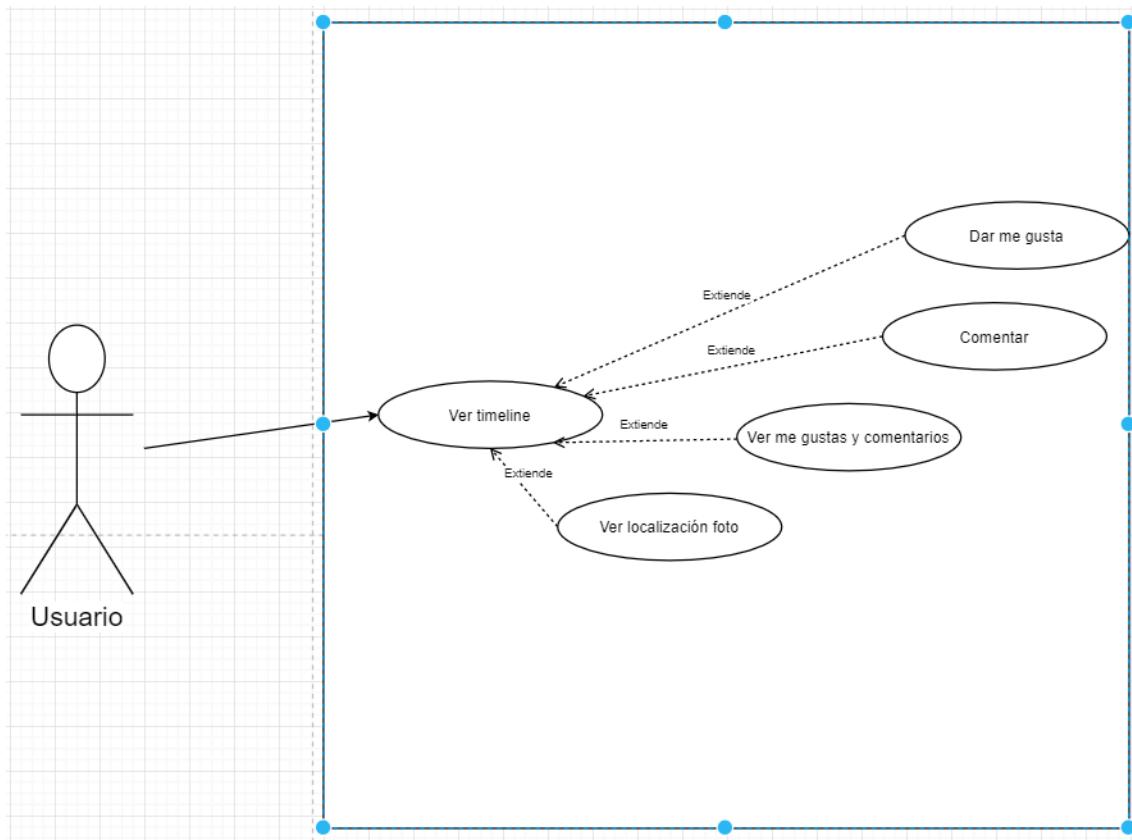
Para poder mostrar el prototipo de forma ordenada voy a realizarlo siguiendo los casos de uso más importantes de la aplicación:

Caso de uso login y registro



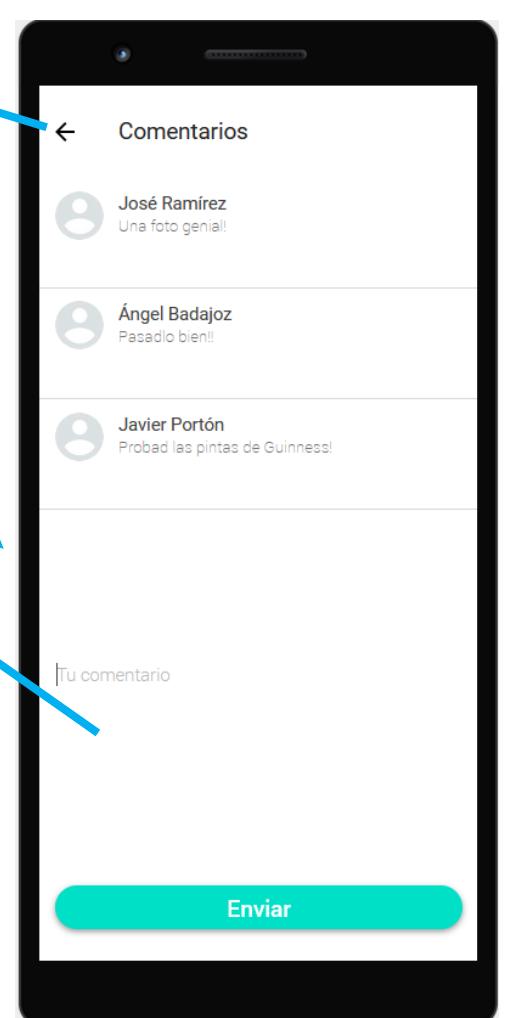
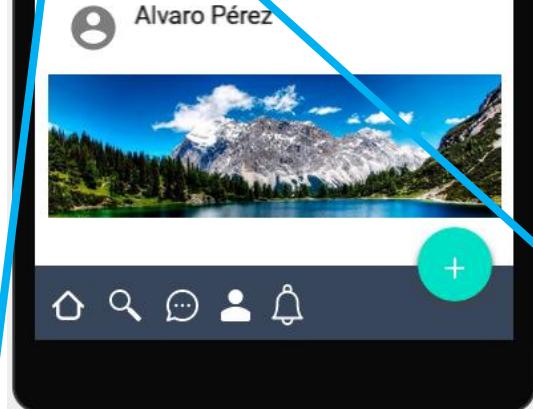
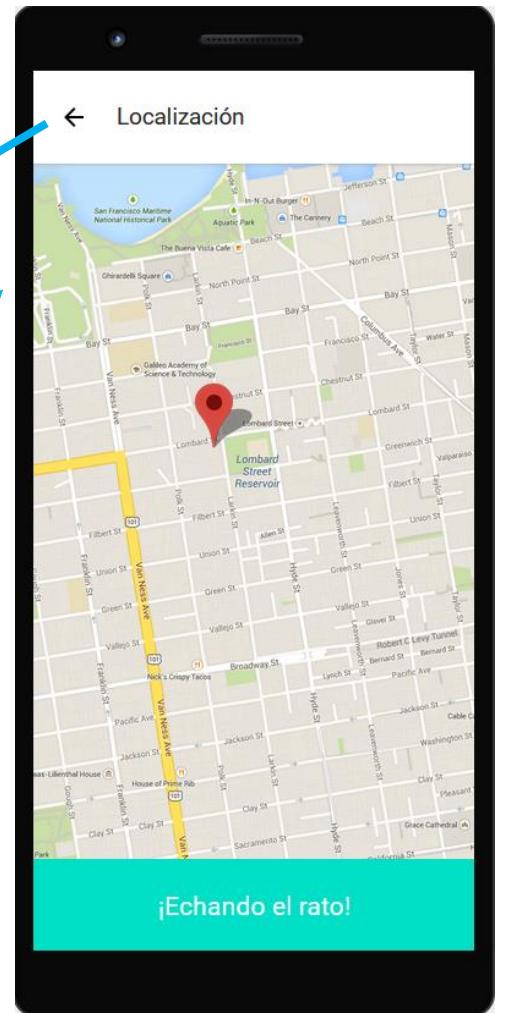
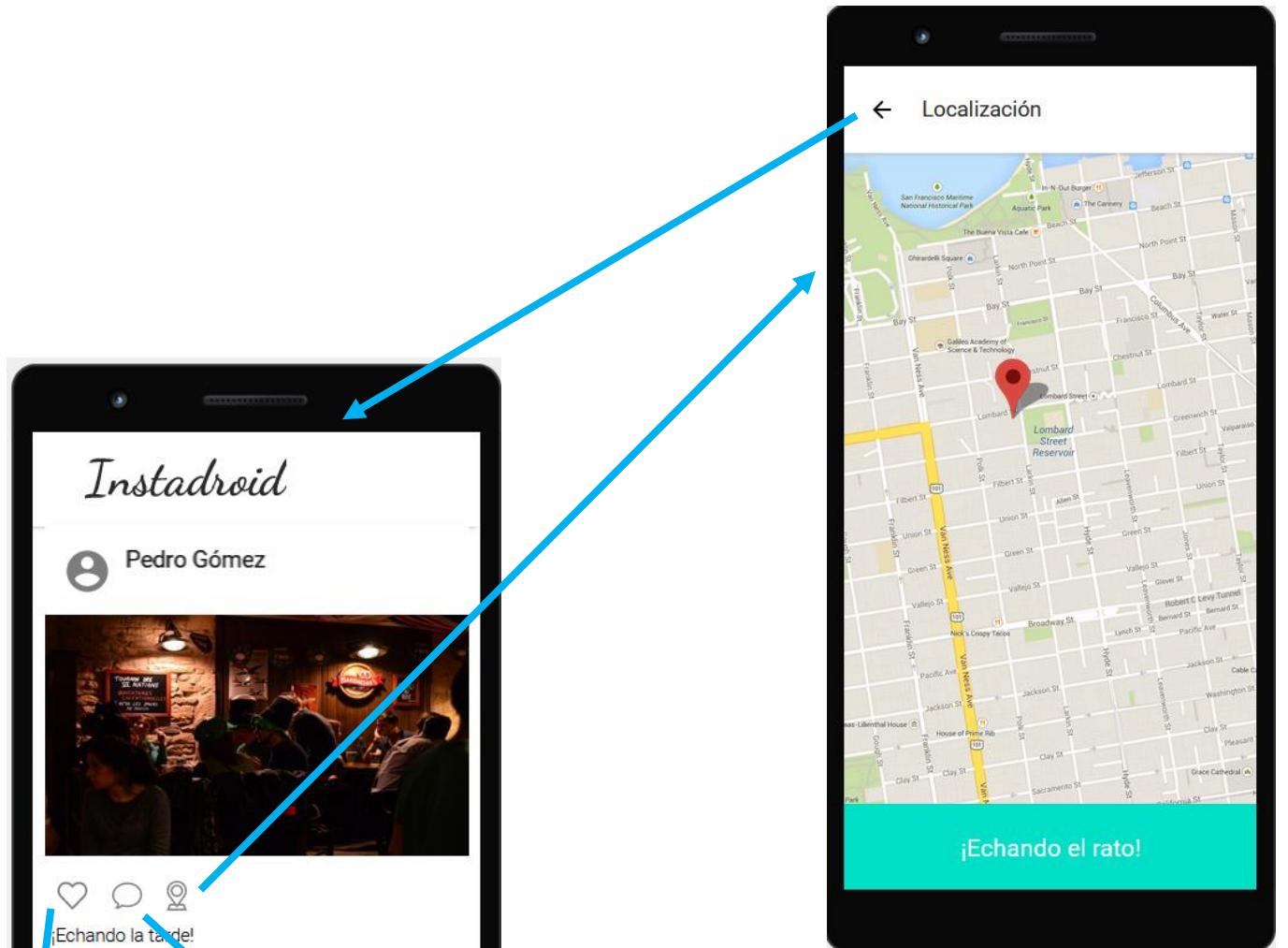
En estas dos pantallas se realizan los casos de uso de logearse y registrarse. De igual manera se facilitan otras formas de acceso a la aplicación con Google y Facebook. El usuario debe pulsar en el botón de registro para registrarse por primera vez, y accederá a la pantalla de registro. En la misma tendrá un botón de volver para regresar a la pantalla de login. Para el registro tendrá que introducir sus credenciales de correo, contraseña y nombre de usuario.

Caso de uso ver timeline y extensiones



Se trata de la pantalla principal de la aplicación, en la que el usuario podrá ver las fotografías subidas por las personas a las que sigue y de forma complementaria podrá darles me gusta, comentarlas, ver sus me gusta y comentarios, así como ver la localización de la foto.

Además, desde esta pantalla tendrá el menú de navegación desde el que podrá acceder al resto de opciones que ofrece la aplicación. Para la presentación de las imágenes se ha utilizado un “Card” un elemento de diseño muy recurrido en todas las aplicaciones para presentar la información al usuario.

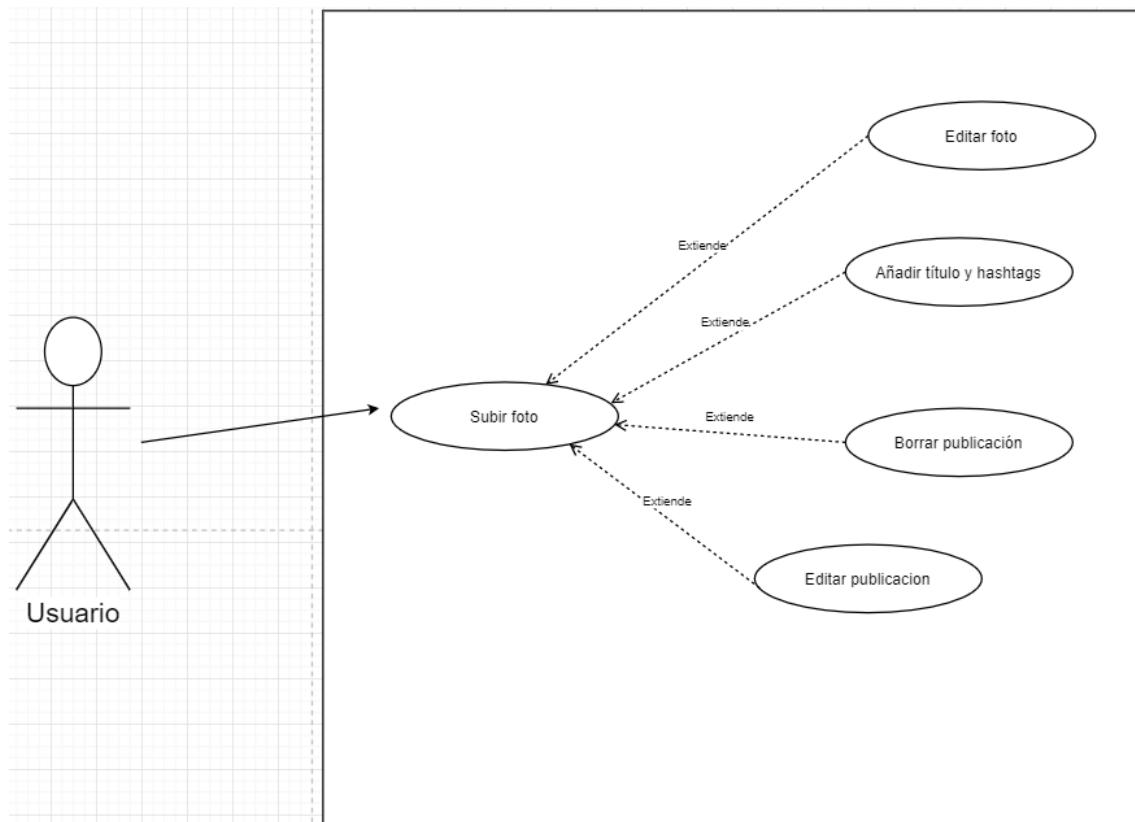


De esta manera se efectúa un comentario, el usuario escribe en el área de texto y pulsa en el botón de enviar

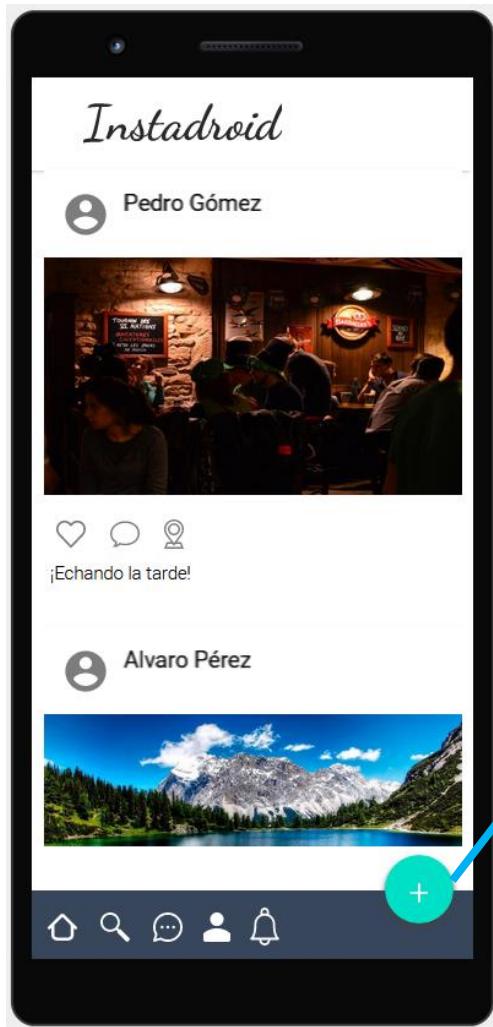
El botón para dar like a una publicación

Como se puede ver la navegación es bastante intuitiva, en cada publicación tenemos un ícono para dar me gusta, un ícono para acceder a los comentarios de la publicación (pantalla en la cuál contamos con un cuadro de texto en el que efectuar nuestro comentario dándole a enviar) y un ícono para consultar la ubicación de la aplicación, cumpliendo con el caso de uso antes expuesto (y sus posibles extensiones).

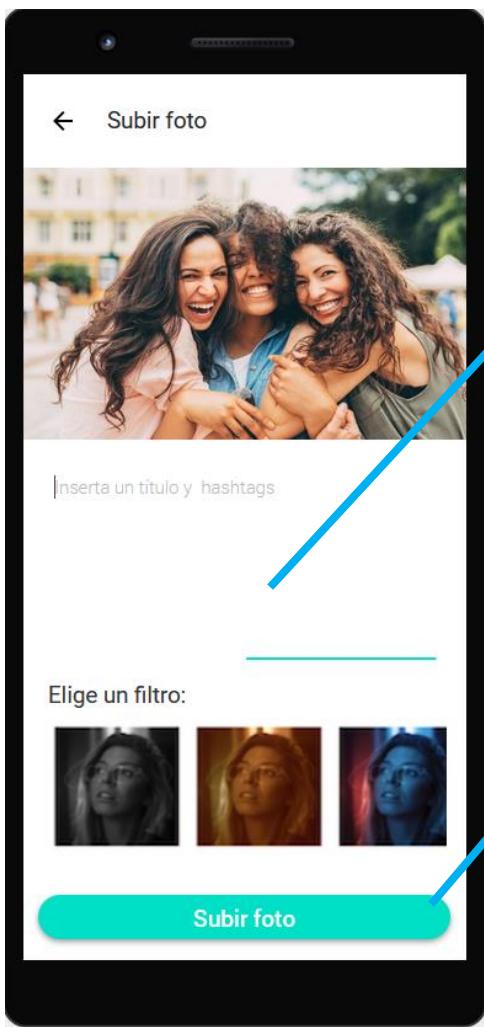
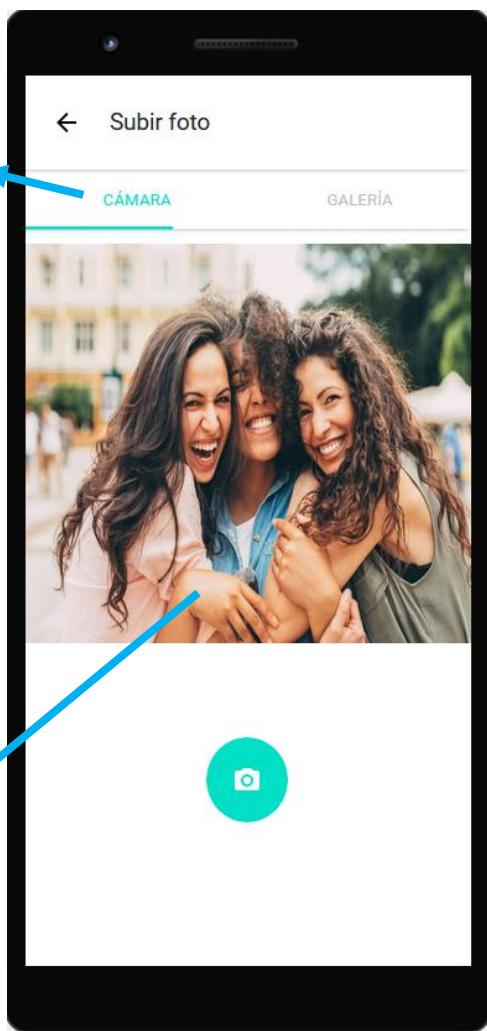
Caso de uso subir foto y extensiones



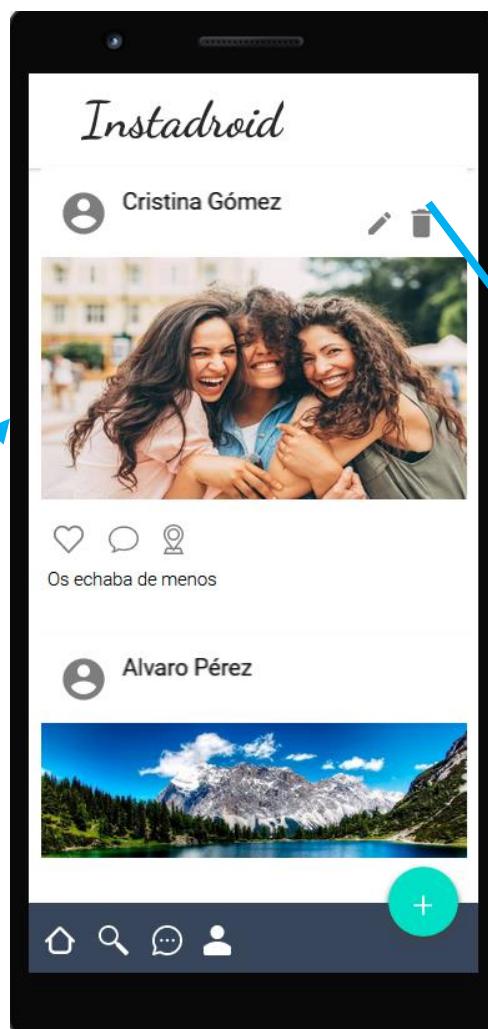
Se trata de la acción del usuario que conlleva subir una fotografía a Instadroid. Además, el usuario podrá editar la foto (añadiéndole filtros) así como dar un título y hashtags a la misma. Una vez subida la foto, el usuario debe poder contar con la opción de editar la publicación (cambiarle el título) o borrarla.



El usuario puede escoger si hacer la foto en el momento o escogerla de la galería



Inserción de título y hashtags.
Elección de filtro



Una vez subida la foto aparece en el timeline

Opciones de editar y borrar la publicación

Los pasos a seguir para subir una publicación son sencillos:

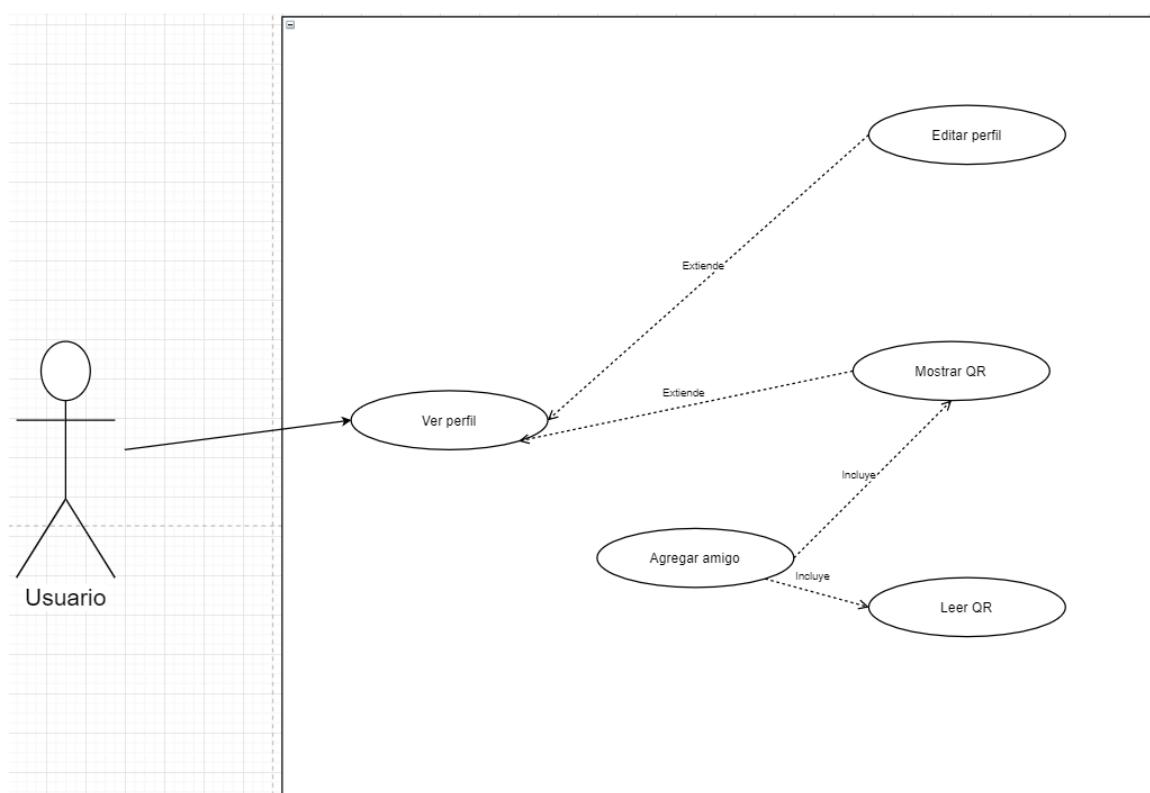
- El usuario pulsa en el botón con el símbolo + en el menú de navegación, lo que le llevará a la pantalla de subir foto
- En la pantalla de subir foto, el usuario puede elegir si hacerla con la cámara en ese momento o elegirla de la galería
- A continuación, el usuario es llevado a la pantalla de inserción de título y hashtags. Cuando pulsa en subir foto, la foto será subida a Instadroid
- El usuario es redireccionado de nuevo al timeline, donde aparecerá su foto con las opciones de borrar y editar.

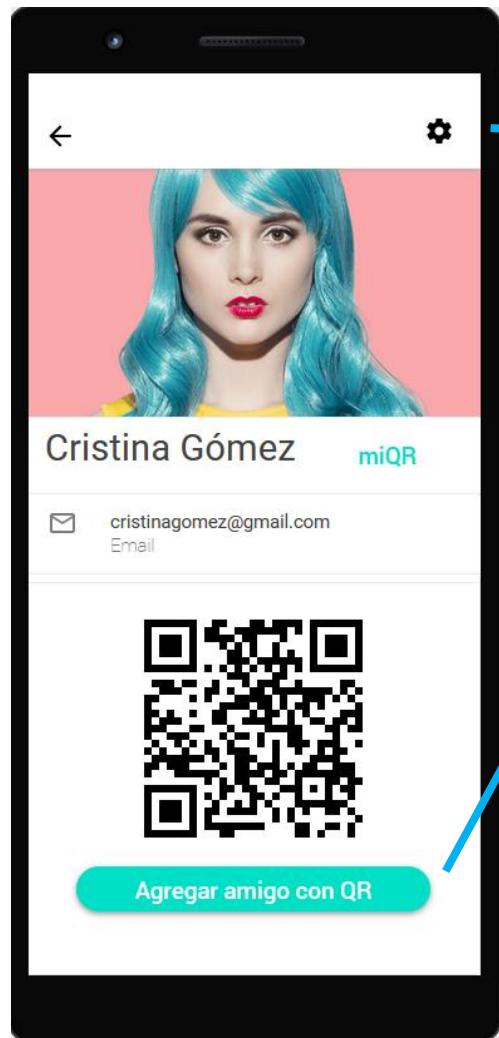
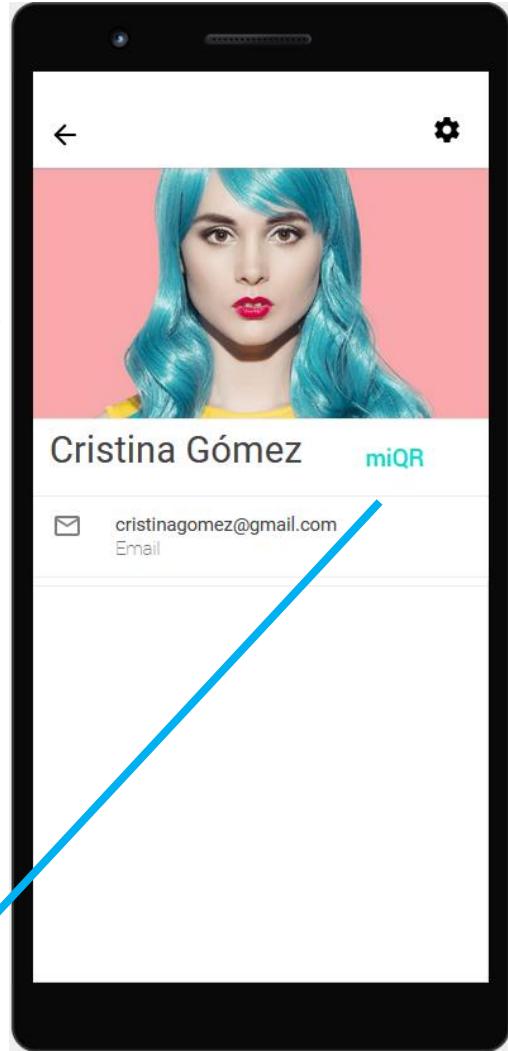
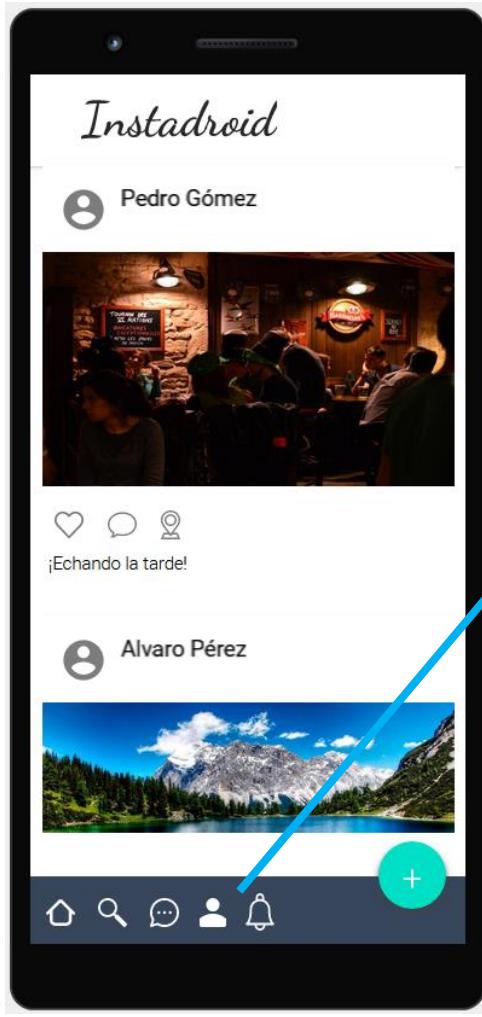
Caso de uso ver perfil y extensiones

Se trata de la acción que realiza el usuario para visualizar la información de su perfil. Además, desde aquí podrá acceder a mostrar su código QR para ser agregado por otras personas como amigo.

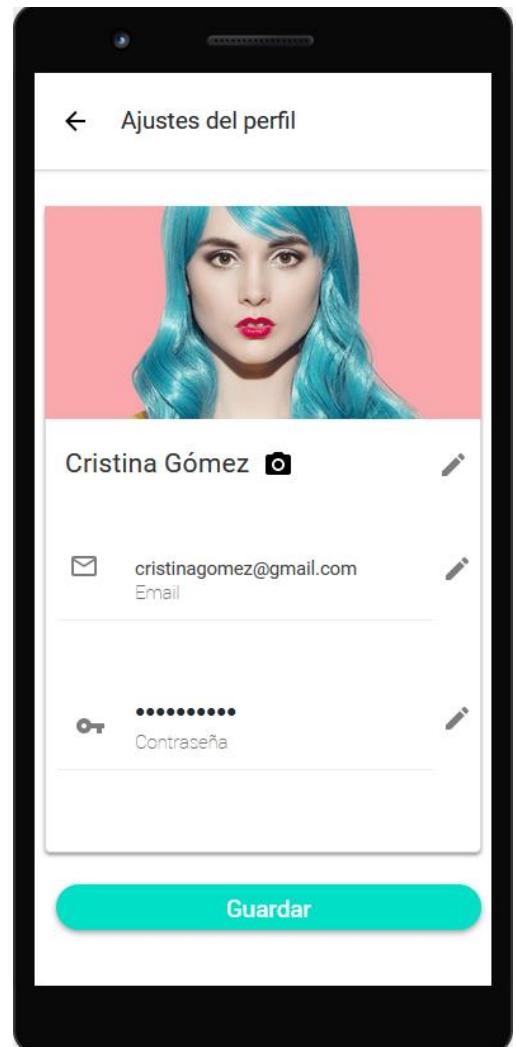
De igual manera también podrá acceder a una pantalla de configuración de su perfil, en la que podrá modificar la información de su perfil (su nombre de usuario, su email, su contraseña y su foto de perfil).

Finalmente, también contará con un botón para poder agregar amigos a través del código QR de los mismos. Este botón llevará a la cámara del teléfono, desde la que podrá leer un el código QR.

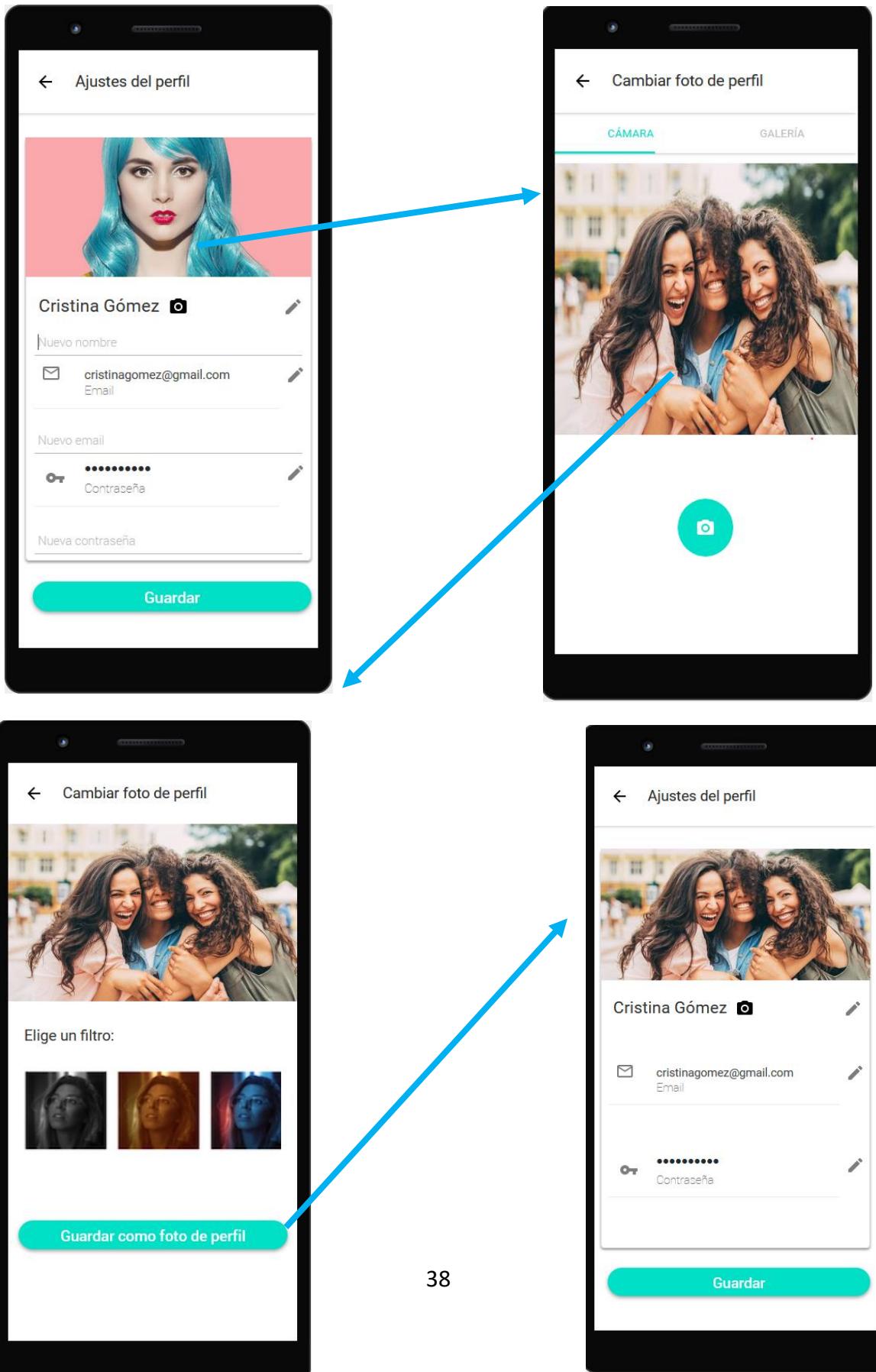




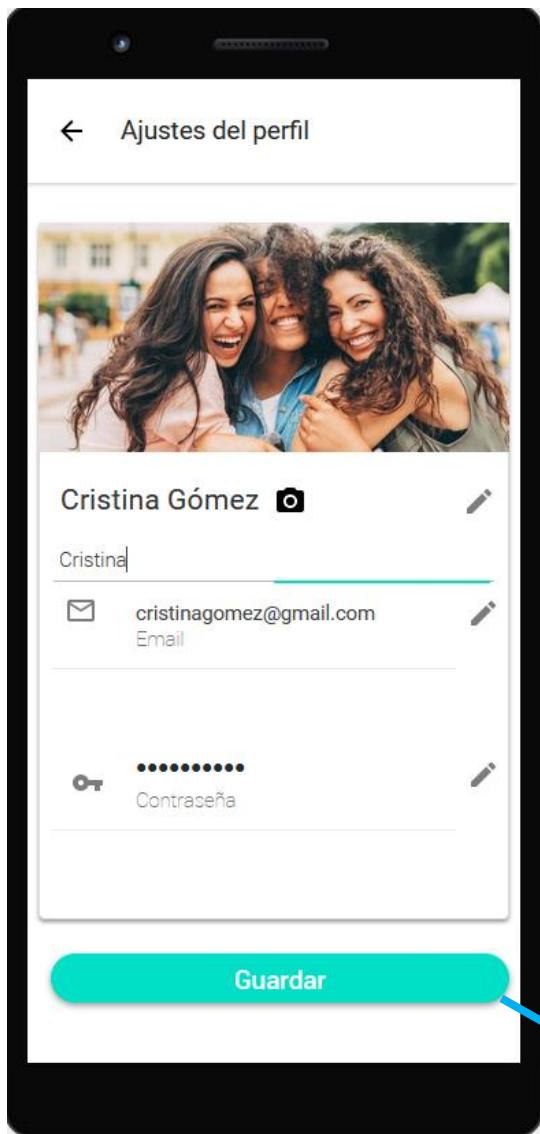
Pulsando en este botón, el usuario accederá a escanear el QR de otro usuario



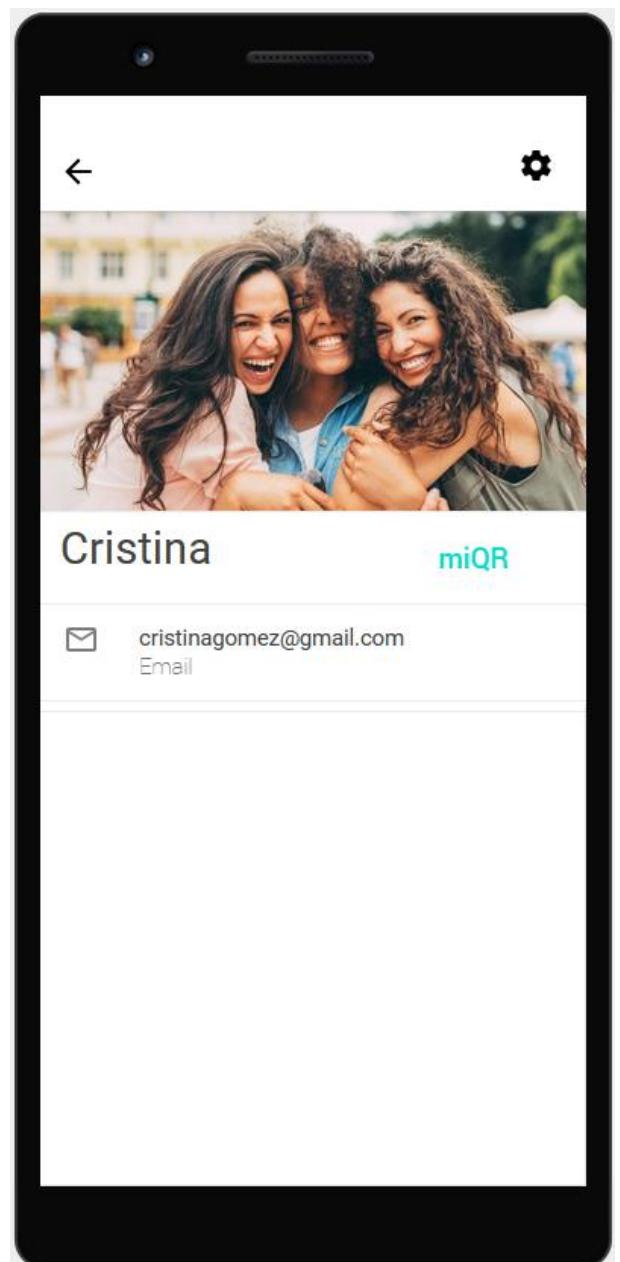
Una vez nos encontramos en la pantalla de ajustes del perfil, el usuario podrá editar los campos pulsando el icono del lápiz que hay al lado de ellos. Para cambiar la foto perfil, tendrá que pulsar al icono de la cámara. El proceso para cambiar la foto de perfil es igual que el de subir una foto:



Para cambiar la información de usuario:



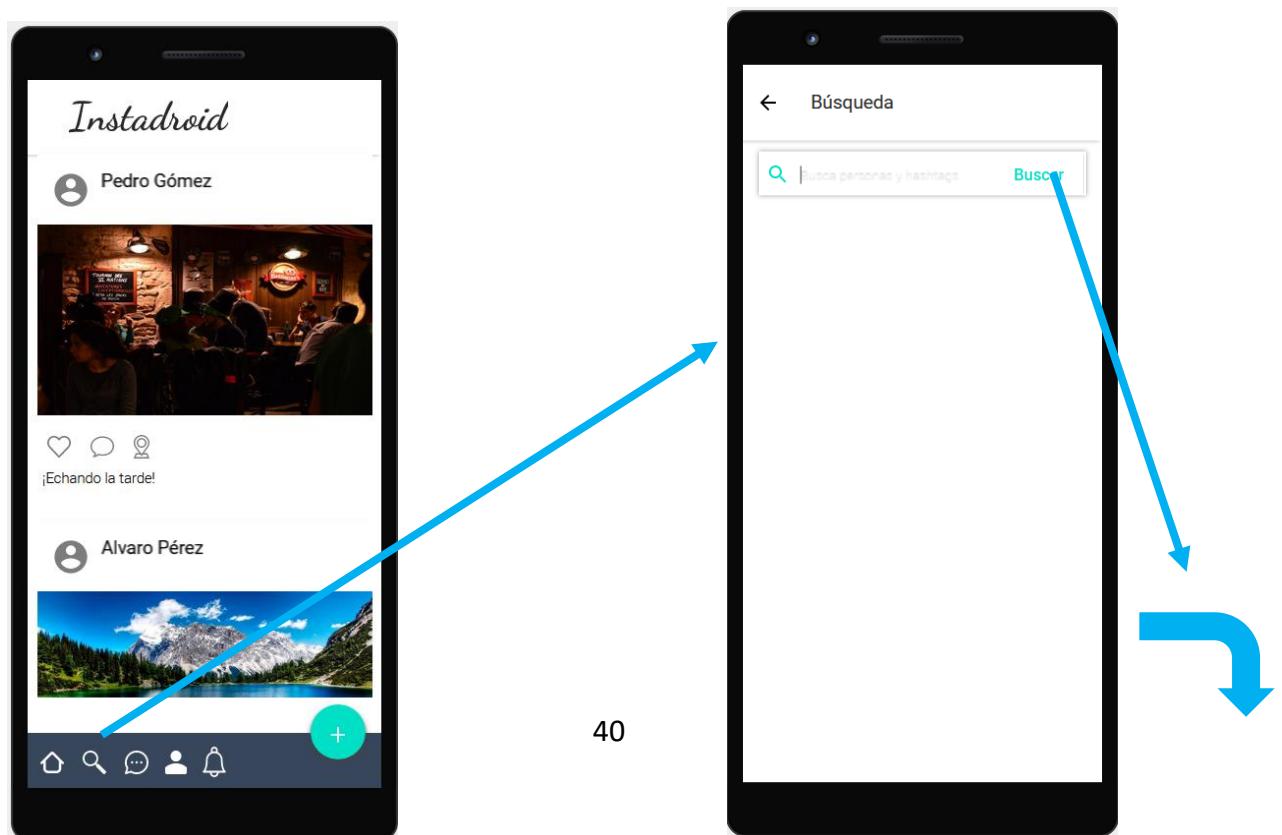
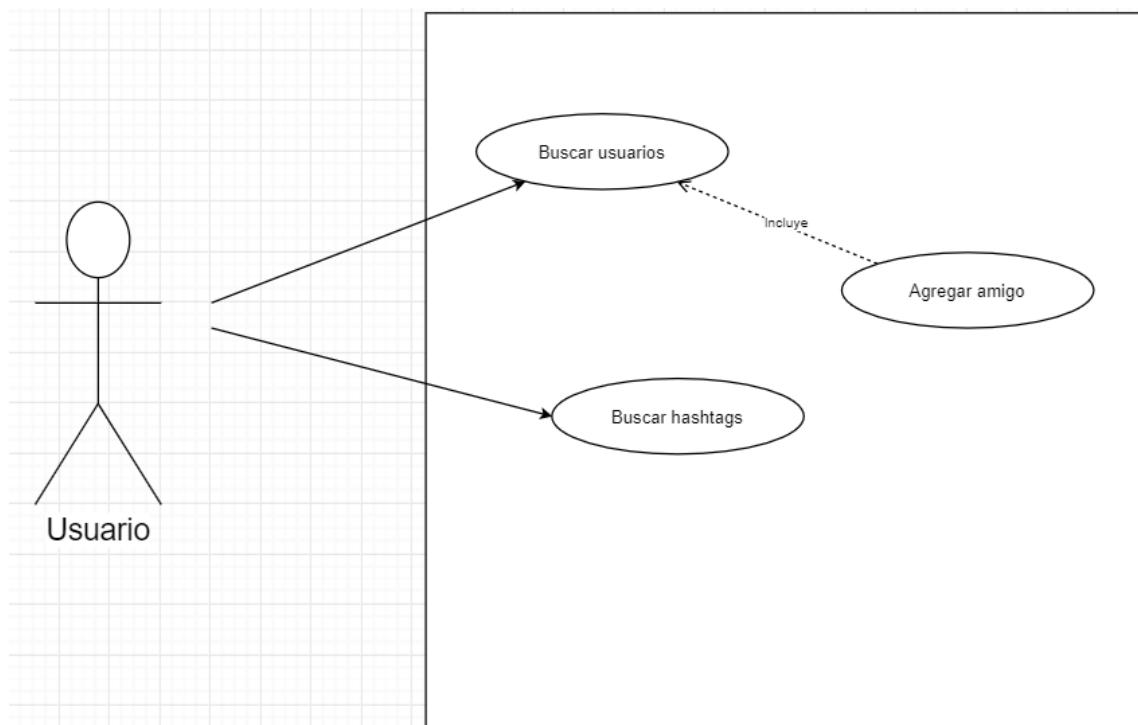
Información del perfil (foto de perfil y nombre de usuario) cambiada

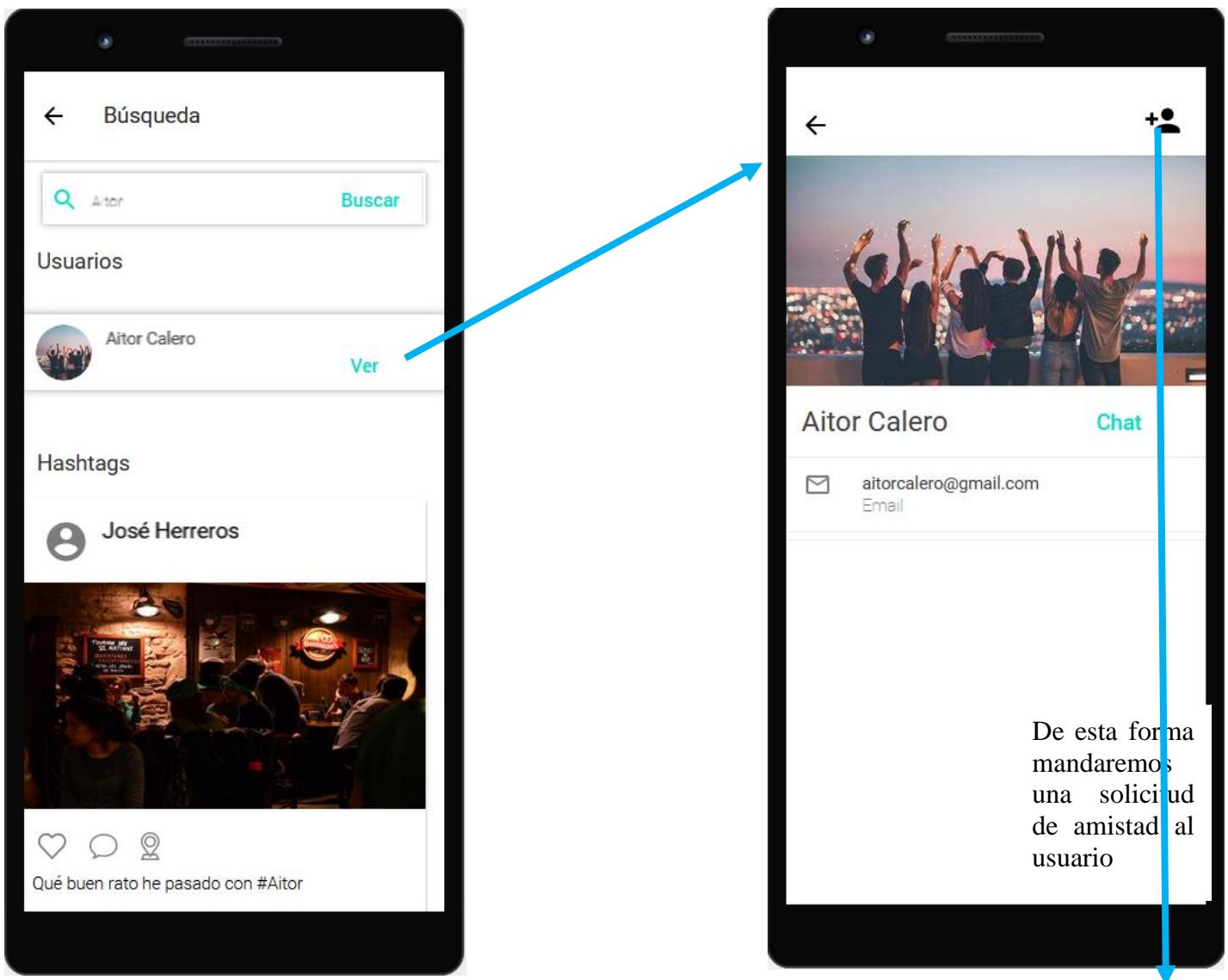


Caso de uso buscar usuarios y hashtags (y extensiones)

Se trata de la acción que realiza el usuario que responde al caso de uso de contar con un buscador en el que se puedan realizar búsquedas de usuarios y hashtags. En este caso, en Instadroid el usuario contará con una pestaña de búsqueda en la que con la introducción de cualquier texto buscará usuarios y publicaciones que contengan el texto especificado.

Cuando encontramos usuarios en nuestra búsqueda, podemos ver su perfil y podemos enviarle una solicitud de amistad.





Como se puede observar, se realiza la búsqueda escribiendo cualquier texto en el campo de tipo input e Instadroid nos mostrará usuarios, así como una lista de publicaciones que contengan ese hashtag.

Pulsando en el botón ver que sale en cada usuario, se podrá acceder al perfil del usuario en cuestión, desde el que podrá enviarse una petición de amistad o empezar una conversación privada con el mismo.

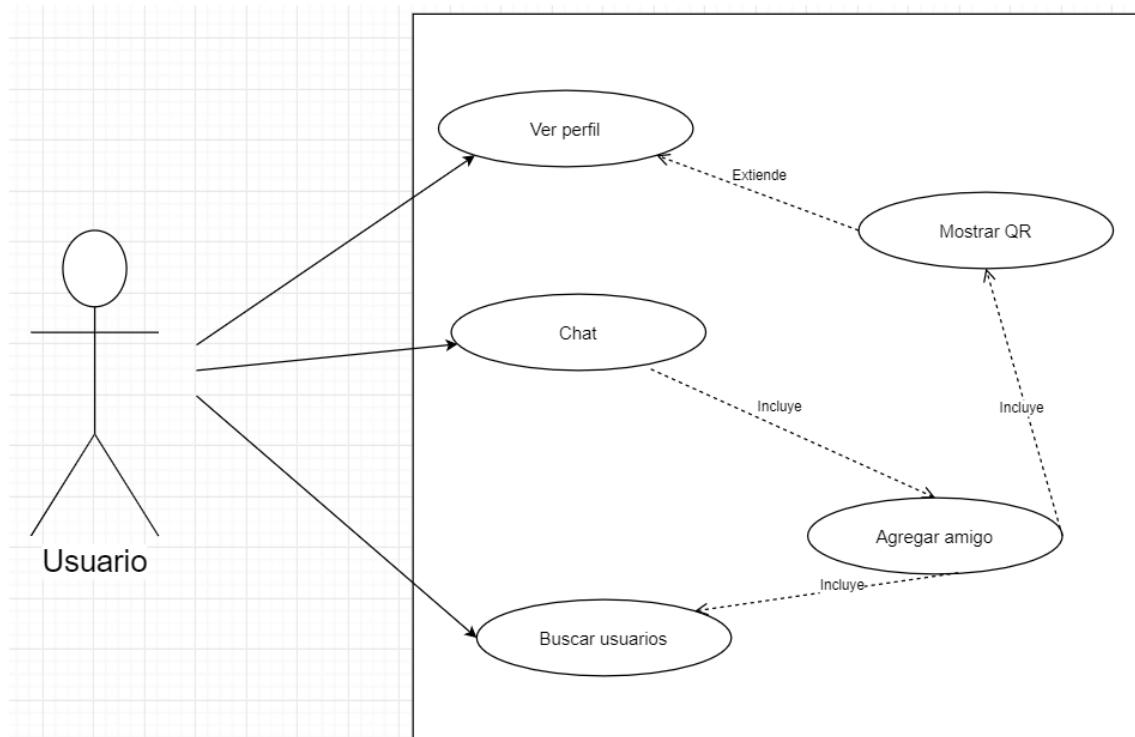
(La conversación privada será mostrada a continuación en el caso de uso “Chat”).

Caso de uso “Chat”

El caso de uso de chatear con otro usuario es el más especial, ya que requiere, en primer lugar, que dos usuarios sean amigos entre sí, por lo que como precondición de estos casos

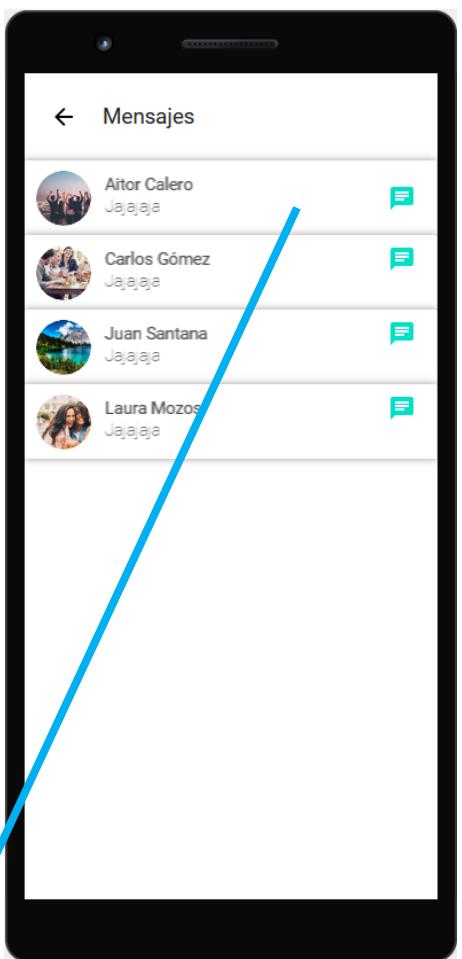
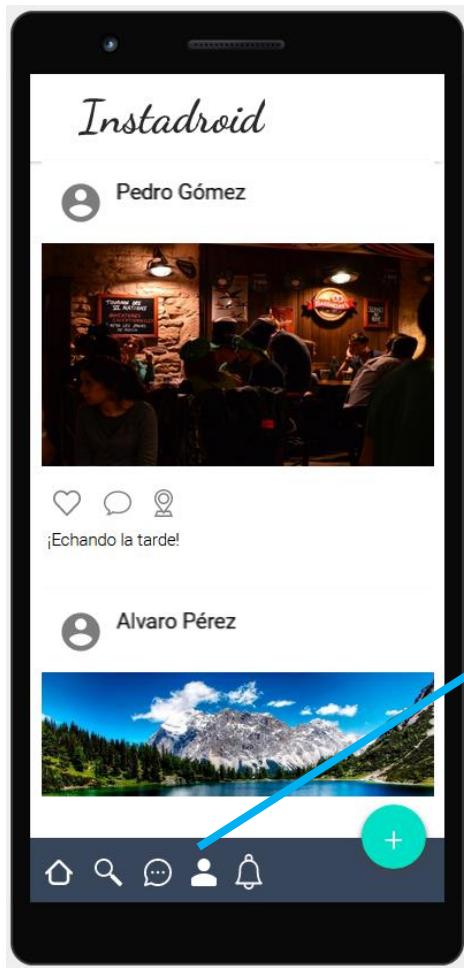
de uso los usuarios deben antes haber establecido una relación de amistad de cualquiera de las dos formas que ofrece la aplicación para hacerlo:

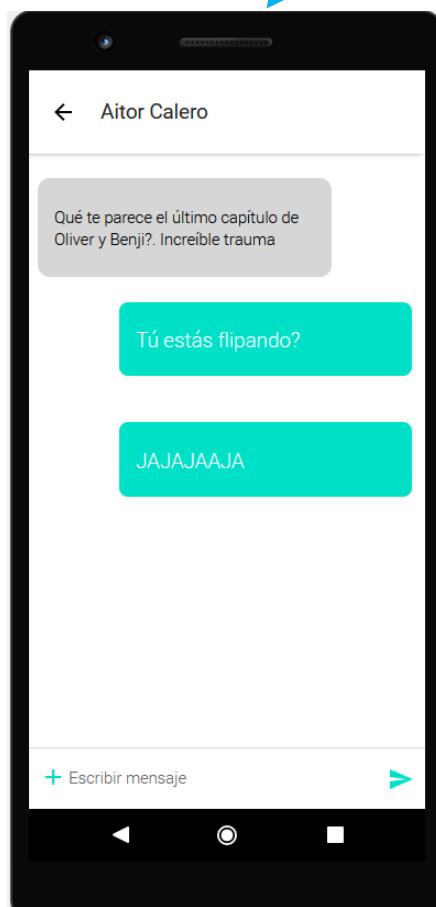
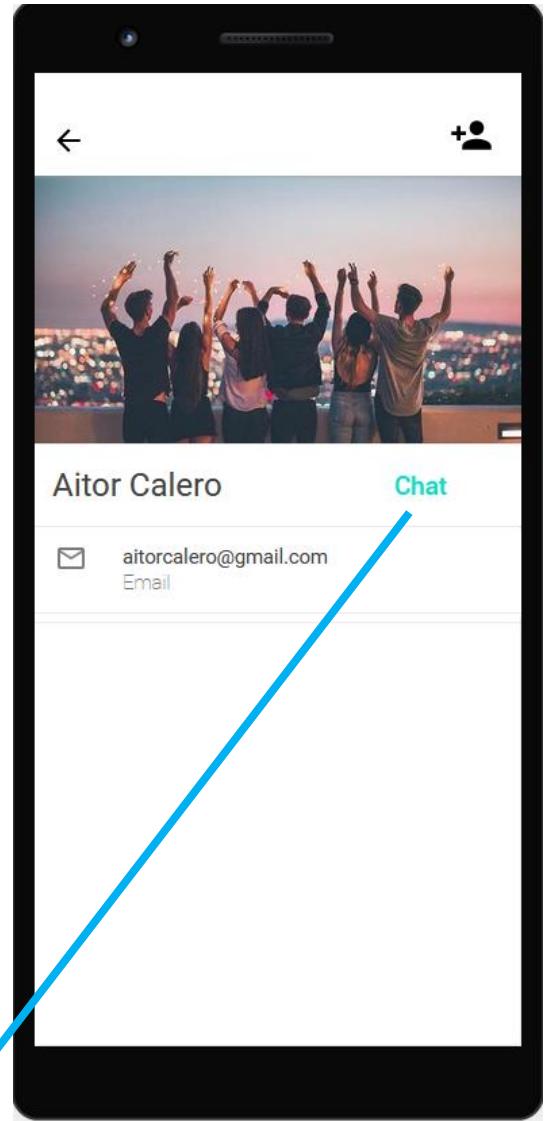
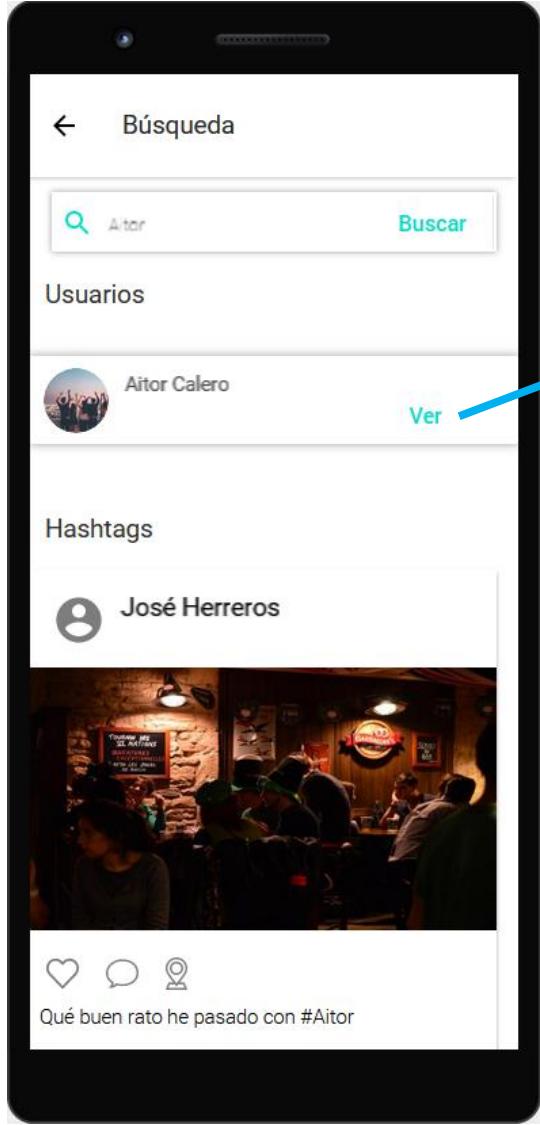
- Mediante el escaneo del QR de otro usuario
- Mediante la búsqueda de un usuario y el envío de una petición de amistad



Como se puede observar para poder chatear se deben haber cumplido antes una serie de condiciones dentro de la aplicación.

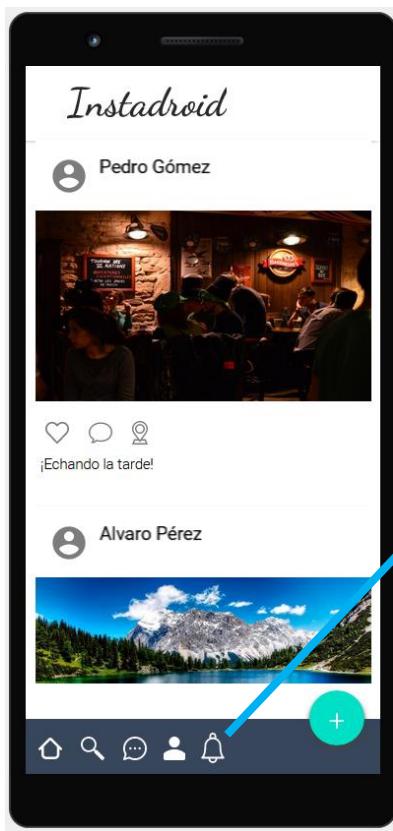
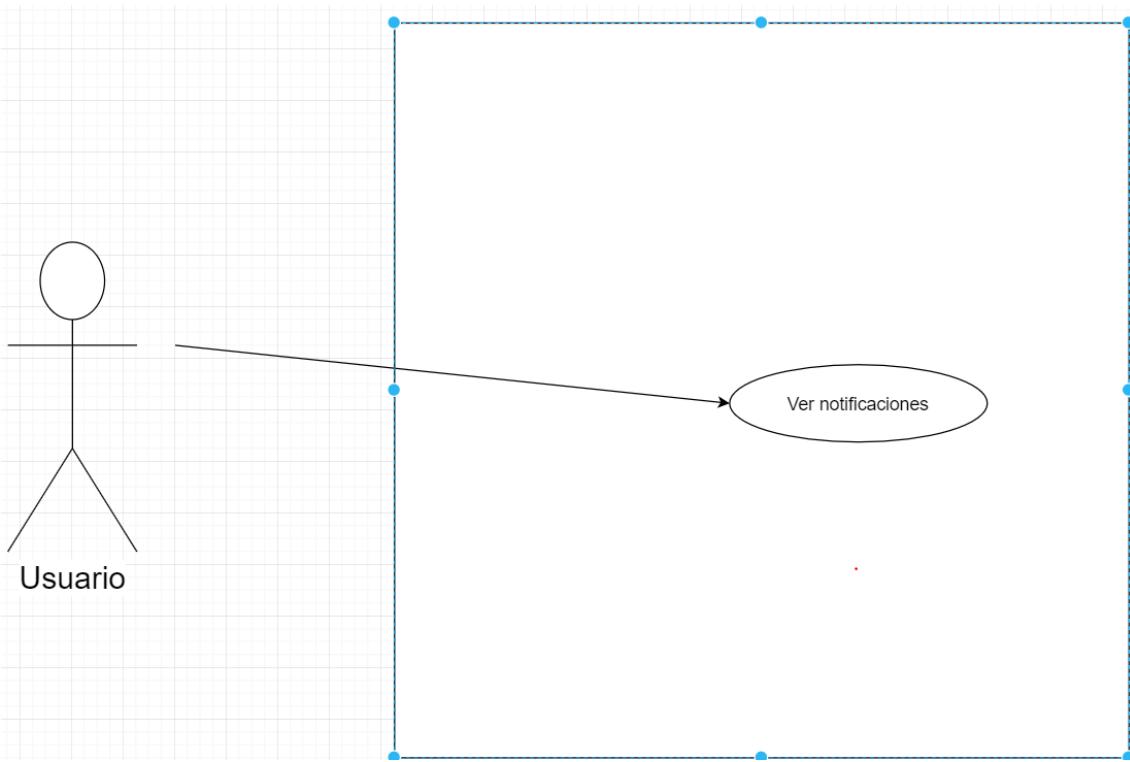
La navegación de estas condiciones (ver perfil, mostrar QR, buscar usuarios, agregar usuarios) por la interfaz ya la hemos visto, por lo que teniendo en cuenta eso y suponiendo que ya hay una relación de amistad con el usuario podemos ver la navegabilidad del caso de uso de realizar un chat.



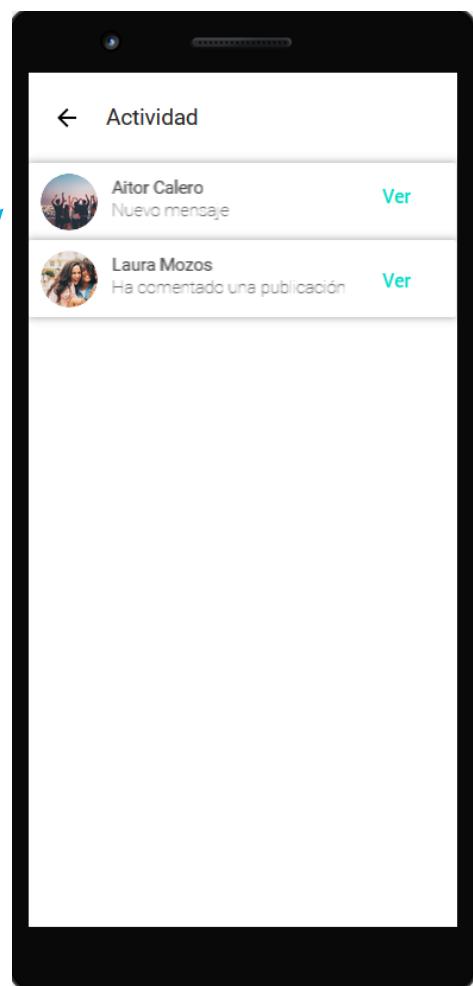


Caso de uso recibir y ver notificaciones

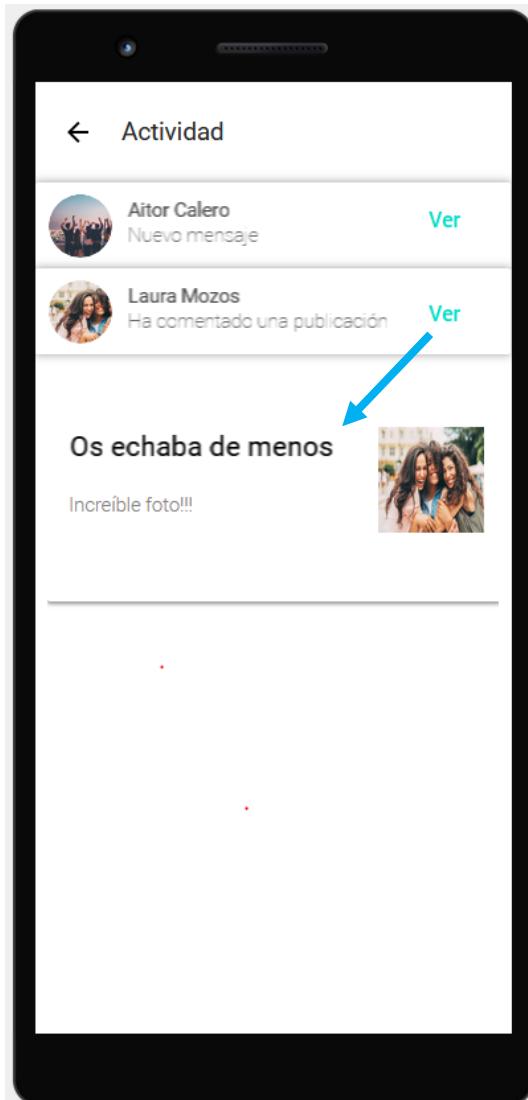
Para este caso de uso el usuario podrá ver las notificaciones que ha recibido en una pantalla a la cuál puede acceder desde la barra de navegación.



Accedemos a la pantalla de actividad en la que podemos ver las notificaciones rebidas. Las notificaciones de nuevo mensaje nos llevarán a la pantalla del chat de ese mensaje



En el caso de que le diéramos a una notificación sobre una publicación se nos desplegaría una “Card” con la información de la publicación en cuestión y mostrando el comentario realizado por otro usuario:



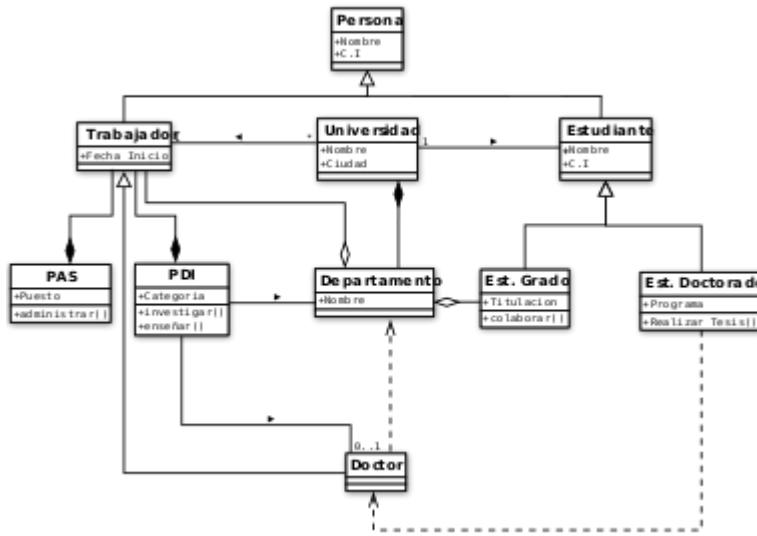
2. Diseño de la persistencia de la información

2.1. Diagrama de clases

Un diagrama de clases en Lenguaje Unificado de Modelado (UML) es un tipo de estructura estática que describe la estructura de un sistema mostrando las clases de este, sus atributos, operaciones (o métodos) y las relaciones entre los objetos⁴².

⁴² https://es.wikipedia.org/wiki/Diagrama_de_clases

Diagrama de Clases

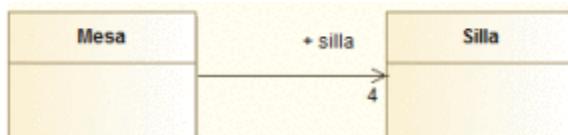


En cuanto a las clases, podemos definir las mismas como una plantilla para la creación de objetos de datos según un modelo predefinido. Cada clase es un modelo que define un conjunto de variables (estado) y métodos para operar con dichos datos (comportamiento). Cada objeto creado a partir de la clase se conoce como instancia de la clase⁴³.

En cuanto a las relaciones entre los objetos, son el tercer pilar fundamental del diagrama de clases. Pueden ser binarias o de orden superior. Si dos clases están relacionadas significa que esas clases tienen algo que ver entre sí⁴⁴.

Hay varios tipos de relación:

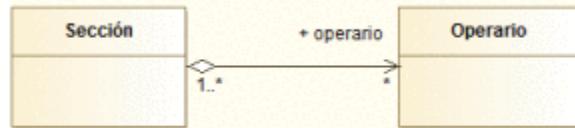
- Asociación: Se utiliza para expresar simplemente que dos clases están vinculadas entre sí.



- Pertenencia: Se apoya en el punto de vista del binomio [PARTE] – [TODO]. El componente [PARTE] se caracteriza porque es una pieza del componente [TODO], teniendo este último la capacidad de albergar al componente [PARTE], integrándolo dentro de sí mismo.

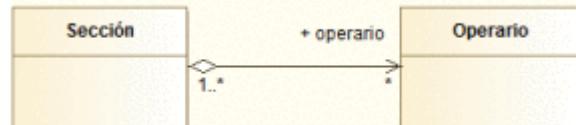
⁴³ [https://es.wikipedia.org/wiki/Clase_\(inform%C3%A1tica\)](https://es.wikipedia.org/wiki/Clase_(inform%C3%A1tica))

⁴⁴ <https://joanpaon.wordpress.com/2013/06/06/uml-diagramas-de-clases-relacion/>



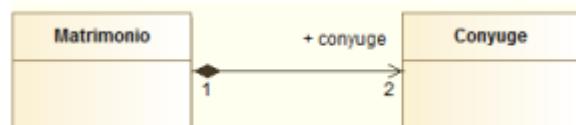
- Autonomía: La autonomía define qué ciclo de vida tienen los objetos de la clase [TODO] y cuál tienen los objetos de la clase [PARTE]. En concreto, la regla para determinar el tipo de asociación es fijarse en el ciclo de vida de los objetos de la clase [PARTE] cuando la clase [TODO] se destruye. ¿Qué ocurre con los objetos de la clase [PARTE] cuando la clase [TODO] desaparece?, la respuesta da lugar a dos tipos de asociación:

➔ Agregación: Cuando el objeto [TODO] se destruye, los objetos [PARTE] pueden seguir existiendo de forma autónoma.



En la agregación, la clase [TODO] se representa con un rombo en blanco y la clase [PARTE] se identifica con una flecha de navegación.

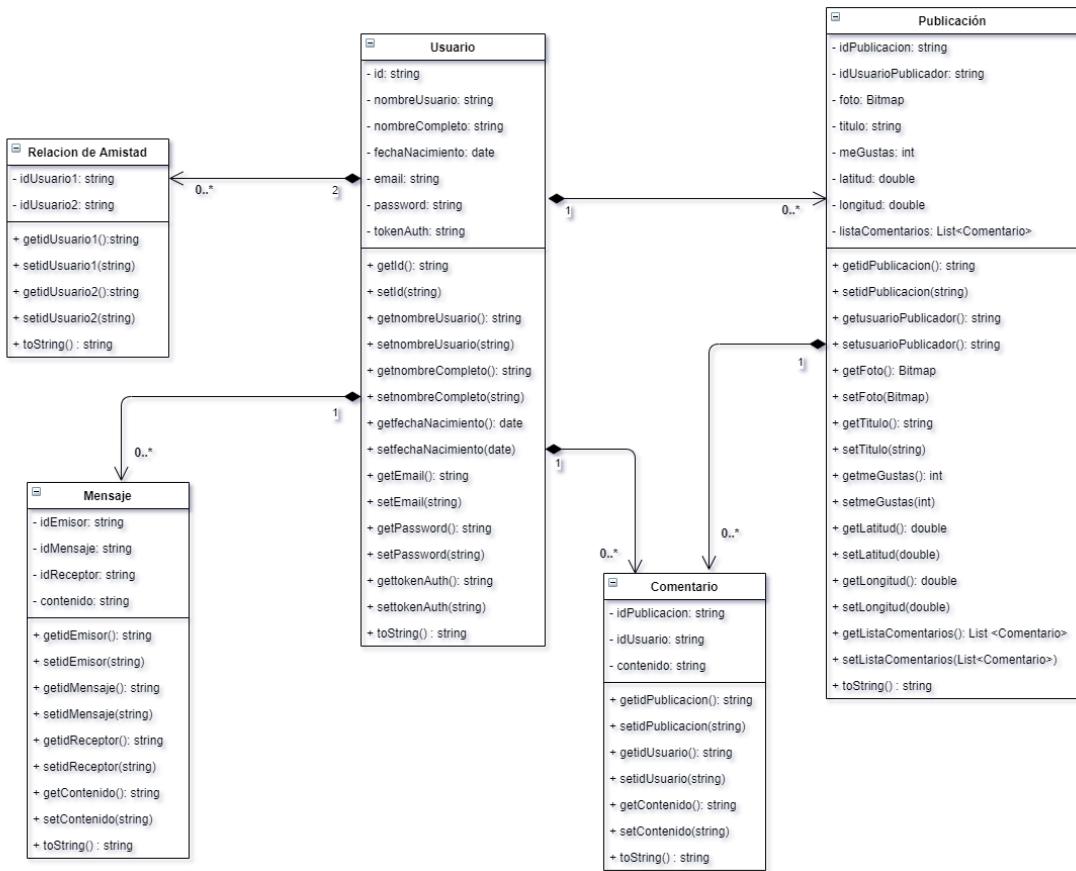
➔ Composición: Cuando el objeto [TODO] se destruye, los objetos [PARTE] desaparecen igualmente, ya que su existencia no tiene sentido.



En la composición, la clase [TODO] se representa con un rombo en negro y la clase [PARTE] se identifica con una flecha de navegación.

2.2. Diagrama de clases de Instadroid

A continuación, muestro el diagrama de clases de la aplicación Instadroid:



Como se puede observar, todas las relaciones de la aplicación Instadroid son relaciones de composición, porque ¿qué sentido tiene que permanezcan en la aplicación las publicaciones de un usuario, sus comentarios, sus mensajes y sus relaciones de amistad si el usuario es eliminado?

Voy a explicar una por una la naturaleza de estas relaciones.

Usuario ↔→ publicación

En Instadroid, los usuarios suben publicaciones a la aplicación, por lo que la publicación está asociada al usuario que la sube.

- Tipo de asociación: Composición, ya que, si el usuario desaparece, sus publicaciones también.
- Cardinalidad: Una publicación pertenece a un usuario, pero un usuario puede subir (o no) varias publicaciones $1 \rightarrow 0..*$

Usuario ↔ Comentario ↔ Publicación

Los usuarios comentan las publicaciones subidas a Instadroid.

- Tipo de asociación: Composición, ya que, si el usuario o la publicación comentada desaparecen, los comentarios también.
- Cardinalidad: Un comentario es realizado por un usuario, pero un usuario puede realizar (o no) varios comentarios $1 \rightarrow 0..*$. Un comentario pertenece a una publicación, y una publicación puede tener ninguno o N comentarios $1 \rightarrow 0..*$.

Usuario ↔ Mensaje

En Instadroid, los usuarios envían y reciben mensajes (pueden ser emisores o receptores en un mensaje concreto).

- Tipo de asociación: Composición, ya que, si el usuario desaparece, sus mensajes también.
- Cardinalidad: Un mensaje es enviado/recibido por un usuario y un usuario puede enviar/recibir ninguno o varios mensajes $1 \rightarrow 0..*$.

Usuario ↔ relación de amistad

En Instadroid, los usuarios pueden enviar y recibir solicitudes de amistad. Las relaciones de amistad se componen por las id's de los dos usuarios que son amigos.

- Tipo de asociación: Composición, ya que, si el usuario desaparece, sus relaciones de amistad, también.
- Cardinalidad: Una relación de amistad pertenece a dos usuarios y los usuarios pueden tener a su vez varias relaciones de amistad $2 \rightarrow 0..*$.

2.3. Diseño de la persistencia de la información

Se denomina persistencia a la capacidad de guardar la información de un programa para poder volver a utilizarla en otro momento⁴⁵. Esto puede significar guardar los datos en un fichero (de texto, binario, csv) o guardar los datos en una base de datos (y sus distintas alternativas)

⁴⁵ <https://uniwebsidad.com/libros/algoritmos-python/capitulo-11/persistencia-de-datos>

Para el desarrollo de Instadroid, distintos factores me han llevado a elegir la opción de utilizar la base de datos NoSQL de Google Firebase.

Antes que nada, debo contextualizar las cosas.

En primer lugar **¿qué es una base de datos NoSQL?**

NoSQL significa “not only SQL”, es decir, no sólo SQL. No es un modelo antagónico, si no un enriquecimiento y complemento útil de las tradicionales bases de datos SQL relacionales⁴⁶. NoSQL plantea modelos de datos específicos de esquemas flexibles que se adaptan a los requisitos de las aplicaciones más modernas⁴⁷.

Este tipo de bases de datos surgió debido a las limitaciones y problemas de las bases de datos relacionales, que no son capaces de hacer frente a las exigencias del desarrollo moderno. En cambio, las bases de datos NoSQL utilizan novedades, como los servidores en la nube y estructuras de datos muy potentes y flexibles.

El funcionamiento de estas bases de datos parte de la premisa de no usar tablas tradicionales y rígidas para almacenar los datos. En su lugar, organizan grandes volúmenes de datos con técnicas más flexibles como por ejemplo documentos y pares de clave valor.

Una de las particularidades de los sistemas NoSQL es el escalamiento horizontal. Para entender este concepto es necesario saber que las bases de datos tradicionales escalan de manera vertical, es decir, toda su capacidad de rendimiento se basa en un solo servidor, por lo que para aumentar su capacidad hay que invertir en un servidor más potente (una opción muy cara en el largo plazo). El escalamiento horizontal supone que las soluciones NoSQL distribuyen sus datos en varios servidores, por lo que si necesitamos manejar más volumen de datos con un servidor austero sería suficiente. De esta manera, pueden almacenar grandes cantidades de datos a un precio menor.

Para terminar de entender bien el concepto de base de datos NoSQL, debemos atender a sus cuatro conceptos más importantes:

- Bases de datos orientadas a documentos

Los datos se almacenan directamente en documentos de diferentes longitudes, sin ser necesario que los datos estén estructurados. Se les asigna distintos atributos

⁴⁶ <https://www.ionos.es/digitalguide/hosting/cuestiones-tecnicas/nosql/>

⁴⁷ <https://www.grapheverywhere.com/bases-de-datos-nosql-marcas-tipos-ventajas/>

sobre los cuales se puede rastrear los contenidos de los documentos. Son especialmente indicados para gestión de contenidos y blogs. Utilizan JSON como formato de archivo ya que permite el intercambio de datos más rápido entre aplicaciones.

■ Bases de datos de grafos

Una base de datos de grafos forma relaciones entre datos mediante nodos. Así con contenidos de información muy entrelazada, ofrecen mejor rendimiento que las SQL relacionales. Se usan sobre todo en el ámbito de las redes sociales para representar relaciones entre seguidores.

■ Bases de datos clave-valor

Guardan los datos en pares clave-valor. Los valores individuales están asignados a claves específicas y el propio juego de datos funciona como clave y representa un valor. La clave crea un índice que se puede usar para buscar en la base de datos. Estas claves son únicas y se pueden comparar a las Primary Key de las bases de datos relacionales.

■ Base de datos orientadas a columnas

A diferencia de SQL, no se guardan los registros en líneas, sino en columnas. Este cambio implica procesos de lectura mucho más cortos y una mayor capacidad de rendimiento. Este modelo se utiliza para minería y análisis de datos.

En segundo lugar **¿por qué decido utilizar una base de datos NoSQL?**

Para ello vamos a realizar un análisis de los conceptos más importantes de la persistencia de datos⁴⁸.

■ Integridad de datos

Es la garantía de que los datos almacenados mantendrán su exactitud y consistencia en el tiempo.

- ➔ En SQL las tablas tienen estructuras rígidas, donde cada dato es de un tipo previamente definido, por lo que, si tu código funciona en un registro, debe funcionar en todos los demás

⁴⁸ <https://medium.com/@eugeniomendoza/c%C3%B3mo-saber-si-necesitas-una-base-de-datos-nosql-b6cfdb5bb7d9b>

- ➔ En NoSQL no hay que definir el tipo de dato a almacenar, en un momento puedes almacenar un número y después almacenar una string o un array. Más que en los propios datos, NoSQL da la prioridad a cómo vamos a acceder a los mismos.

Si necesitamos que nuestros datos se mantengan exactos y consistentes, utilizaremos SQL. Esto es ideal en sistemas intolerantes a los fallos (por ejemplo, bancos, empresas).

Si nuestras estructuras de datos son propensas a cambiar, puede ser problemático en SQL ya que modificar la estructura rígida de la base de datos puede darnos más de un problema.

En este caso será mejor utilizar NoSQL, ya que añadir claves nuevas a un documento no supone ningún problema y es sencillo.

■ Operaciones atómicas

Se trata de cuando realizamos un cambio que afecta a múltiples entidades de la base de datos al mismo tiempo. Se acompaña del concepto de transacciones y los famosos commits y rollbacks.

- ➔ En SQL, como las tablas están conectadas, pueden “ponerse de acuerdo” para no aceptar nuevos cambios hasta que termine la transacción.
- ➔ NoSQL, al no existir las fuertes relaciones entre los datos que hay en SQL, no hay posibilidad de hacer una transacción atómica. Cuando queremos cambiar algo en 5 entidades diferentes, tendremos que hacer 5 llamadas a la base de datos.

Si nuestra aplicación maneja operaciones críticas como pagos a la vez del mismo producto, por ejemplo, SQL nos cubre las espaldas mejor, pero si no es el caso es mejor acudir a NoSQL porque la atomicidad no siempre es crucial... o ¿acaso es importante que dos usuarios difieran en la cantidad de likes de una foto?

■ Escalabilidad

Se trata de lo que he expuesto justo antes de la escalabilidad horizontal de las bases de datos NoSQL. Esto viene muy bien principalmente en las primeras fases de crecimiento de una aplicación ya que expandir el volumen de datos a manejar será barato.

■ Velocidad

¿Cómo de rápido leemos y escribimos en la base de datos?

- ➔ En SQL, las garantías que otorgan las relaciones conllevan un precio. En cuanto comenzamos a hacer consultas con muchas joins que involucran múltiples entidades aumentamos el tiempo de búsqueda.
- ➔ En NoSQL, se suele contar con mecanismos de búsqueda sumamente rápida para conseguir un dato específico entre millones. La principal ventaja en la velocidad es que puedes diseñar la base de datos en función de las consultas que le haremos.

■ Consistencia vs redundancia

- ➔ En SQL, consiste en asegurarse de que un único dato esté una única vez en toda la base de datos.
- ➔ En NoSQL, la redundancia es repetir adrede los datos a conveniencia en varias partes de la base de datos. Por ejemplo, en mi caso yo tendré un documento con usuarios y otro con publicaciones. Cuando se muestran las publicaciones en la app también tienen datos de los usuarios, por lo que podría repetir los datos del usuario que ha subido la publicación (nombre de usuario, foto de perfil) y meterlos también como datos de la propia publicación. De esta forma gano velocidad porque no tengo que realizar dos consultas para simular una “relación”.

Conclusión

Después de esto, creo que se puede ver claro y justificado el porqué de mi elección.

- Requisitos del propio proyecto. Si necesito ver información en tiempo real la mejor alternativa es Firebase y su Cloud Storage, por lo que aquí ya estoy condicionado a elegir NoSQL.
- No vamos a realizar operaciones sensibles y críticas en la aplicación por lo que la integridad no es un factor de principal importancia (priorizamos velocidad). Además, como estamos en etapas tempranas del desarrollo los tipos de datos pueden cambiar en algún momento y NoSQL nos dará facilidades para ello.
- No requerimos de operaciones atómicas que cambien muchas entidades, si cambia una publicación solo cambia la publicación, si el usuario edita su nombre, solo cambia el usuario, etc.

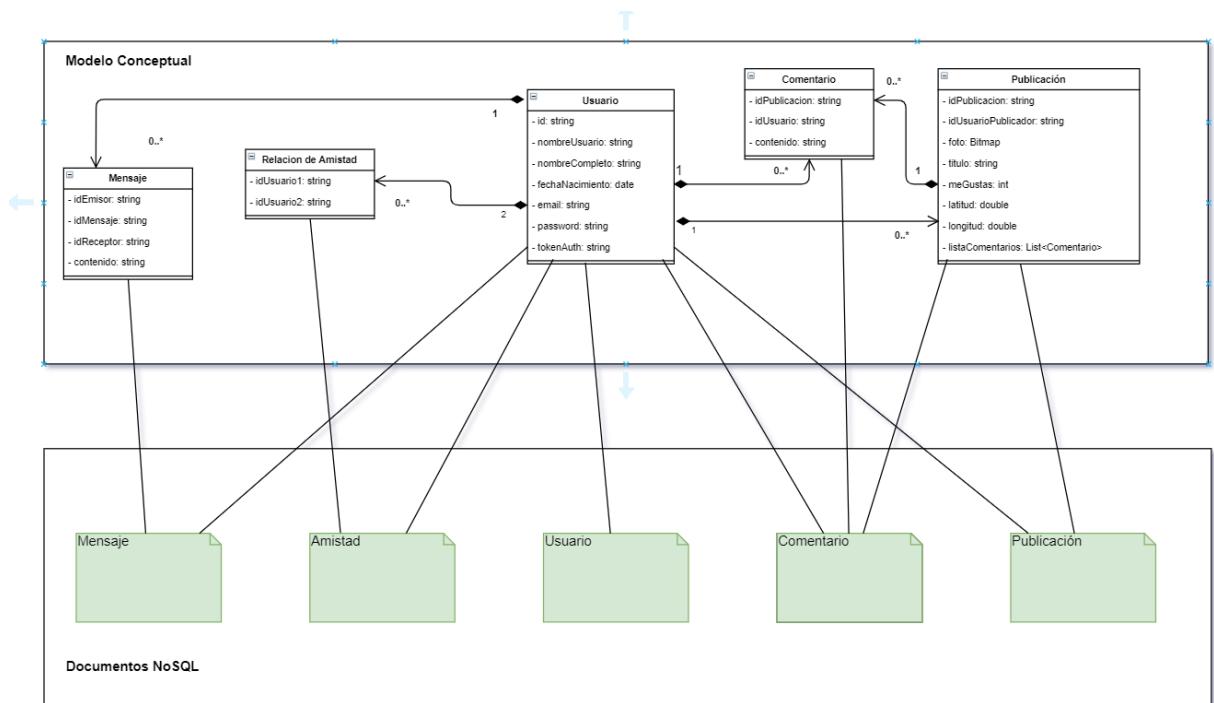
- Si la aplicación tiene buena acogida y necesitamos manejar más volumen de datos podremos hacerlo de forma austera gracias a la escalabilidad horizontal que nos ofrece NoSQL, comprando servidores baratos y aumentando la base de datos
- Con el objetivo de alcanzar la máxima velocidad posible de lectura y escritura nos serviremos de los mecanismos que ofrece NoSQL (recuerdo que debemos mostrar datos en tiempo real).

2.4. Diagrama de la persistencia de la información Instadroid

Este diagrama hay que entenderlo en 2 dimensiones. Por un lado, tenemos la dimensión de negocio que describe a los propios objetos/clases que se manejan a nivel de aplicación. Por otro lado, tenemos el grupo de documentos NoSQL físicos que se van a crear, indicando los datos de qué clases van a almacenar⁴⁹.

Las flechas muestran las clases de las que los documentos guardan datos, cuando hacen referencia a dos clases en este caso significa que el documento tendrá los datos de una y la referencia a otra (no quiero que por ejemplo cada vez que me traiga un usuario de la base de datos, forzadamente me tenga que traer sus publicaciones, por ejemplo, si no que en la publicación guardaré una referencia al usuario, por lo que tendré que realizar dos consultas a la base de datos, pero será más rápido).

Diagrama:



⁴⁹ <https://eaminds.com/2018/08/03/modelando-nosql-data-bases/>

Como se puede observar, se reutiliza el propio diagrama de clases con los datos de cada clase para marcar qué es lo que va a contener cada documento.

Relaciones

De izquierda a derecha:

- Mensaje: Contendrá los elementos de la clase mensaje y el idEmisor y el idReceptor serán referencias a datos de la clase Usuario, que tendremos en el documento de usuarios.
- Amistad: Contendrá dos referencias a datos de la clase Usuario que estarán en el documento de usuarios.
- Usuario: Contendrá los datos del usuario sin más referencias.
- Comentario: Contendrá los datos propios del comentario además de una referencia a datos de la clase Usuario (usuario que realiza el comentario), que estarán en el documento de usuarios y una referencia a los datos de la clase Publicación (publicación comentada), que estarán en el documento de publicaciones.
- Publicación: Contendrá los datos propios de la publicación y una referencia a la clase Usuario (usuario que sube la publicación) cuyos datos estarán en el documento de usuarios.

3. Diseño de la arquitectura del sistema

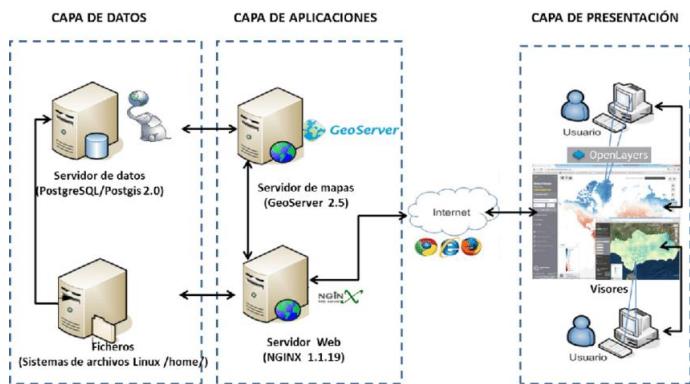
3.1. Arquitectura de un sistema

En los inicios de la informática, la programación era bastante difícil para la mayoría de las personas, pero con el tiempo se han ido descubriendo y desarrollando formas y guías generales para resolver problemas. A las mismas se les ha denominado arquitectura de software, por la semejanza con los planos de un edificio, debido a que la arquitectura indica la estructura, funcionamiento e interacción entre las distintas partes del software⁵⁰.

Abordada desde el punto de vista técnico, la arquitectura de software es el diseño de más alto nivel de la estructura de un sistema:

⁵⁰ https://es.wikipedia.org/wiki/Arquitectura_de_software

- También denominada arquitectura lógica, integra un conjunto de patrones y abstracciones coherentes que proporcionan un marco definido y claro para interactuar con el código fuente del software.
- Se selecciona y diseña con base en requisitos y restricciones.
 - Objetivos: Prefijados por el sistema, no solo son de tipo funcional, sino que también incluyen otros aspectos como el mantenimiento, la auditoría, flexibilidad e interacción con otros sistemas.
 - Restricciones: Limitaciones derivadas de las tecnologías disponibles.
- Define de manera abstracta los componentes que llevan a cabo tareas de cómputo, sus interfaces y la comunicación entre los mismos. La arquitectura general del sistema especifica las distintas particiones físicas del mismo, su descomposición lógica en subsistemas de diseño y la ubicación de cada subsistema en cada partición⁵¹.



3.2. Arquitectura del sistema Instadroid

Decisión de la arquitectura y justificación

Como bien he descrito a la hora de mostrar el diseño de la persistencia de datos, Instadroid necesita que los datos sean mostrados en tiempo real, por lo que cualquier cambio respecto a ellos deberá reflejarse automáticamente en la interfaz del cliente de la aplicación.

En este sentido, como ya he descrito, utilizaré Firebase ya que me otorga las herramientas necesarias a tal fin (se explicará Firebase de forma detallada en el apartado de herramientas de la implementación del proyecto).

⁵¹ <https://manuel.cillero.es/doc/metodologia/metrica-3/procesos-principales/dsi/actividad-1/>

Al utilizar Firebase, la arquitectura de Instadroid será una arquitectura en tiempo real que además utiliza arquitecturas basadas en servicios web y servicios.

Una **arquitectura en tiempo real** es un software cuyo correcto funcionamiento depende de los resultados producidos por el mismo y del instante de tiempo en el que se producen esos resultados⁵². Hay dos tipos:

- Soft: Es aquel que se degrada si los resultados no se producen correctamente de acuerdo con los requisitos especificados.
- Hard: Su funcionamiento será incorrecto si los resultados no se producen de acuerdo con la especificación temporal.

Firebase ofrece estos servicios de consulta de datos en tiempo real, por lo que, en nuestra implementación, cuando un usuario comenta una foto, otro usuario que esté viendo los comentarios de la misma foto verá el comentario aparecer al momento, sin necesidad de “refrescar” la pantalla.

Una **arquitectura orientada a servicios (SOA)** es una forma de pensar en servicios, su construcción y resultados. Cada servicio es una representación lógica de una actividad de negocio que tiene un resultado de negocio específico, por ejemplo, comprobar el crédito de un cliente⁵³.

Firebase, ofrece distintos servicios, si un usuario quiere autenticarse lo hará contra el servicio de autenticación de Firebase. Si un usuario quiere subir una publicación los datos de esta se subirán al servicio de base de datos (Realtime database) y la imagen se subirá a el servicio de almacenamiento (Storage).

Finalmente, una **arquitectura orienta a servicios web** es muy similar a la anterior, utilizando un conjunto de protocolos y estándares que sirven para intercambiar datos entre aplicaciones. Nos permiten intercambiar datos entre aplicaciones programadas en distintos lenguajes. Estos estándares utilizados han sido muchos, pero principalmente en los últimos años se utiliza el estándar REST (Representational State Transfer), que utiliza los verbos del protocolo HTTP proporcionando una API que utiliza los verbos para poder realizar distintas operaciones entre la aplicación que ofrece el servicio web y el cliente⁵⁴.

⁵² <https://ittgweb.wordpress.com/2016/05/29/3-7-diseno-de-software-de-arquitectura-de-tiempo-real/>

⁵³ https://es.wikipedia.org/wiki/Arquitectura_orientada_a_servicios

⁵⁴ https://es.wikipedia.org/wiki/Servicio_web

Todo el conjunto de herramientas de Firebase se puede mirar como un servicio Web⁵⁵, que expone una API Rest a la que nosotros le enviamos peticiones, solo que Firebase facilita librerías a los distintos lenguajes de programación con métodos que se encargan de realizar estas peticiones. Por lo que, en realidad, la comunicación con Firebase es con el protocolo HTTP, aunque nosotros no lo “escribamos” específicamente.

Así, por ejemplo, cuando hacemos un set a una colección de datos en realidad estamos mandando esos datos vía POST.

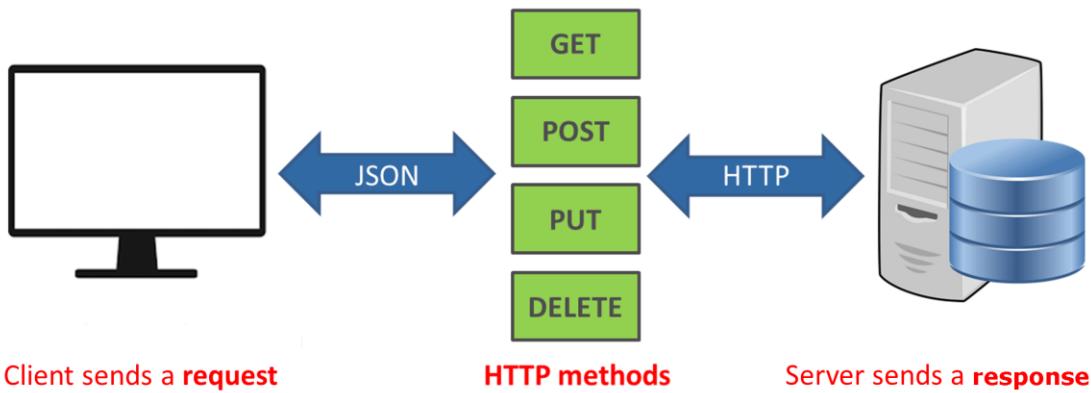
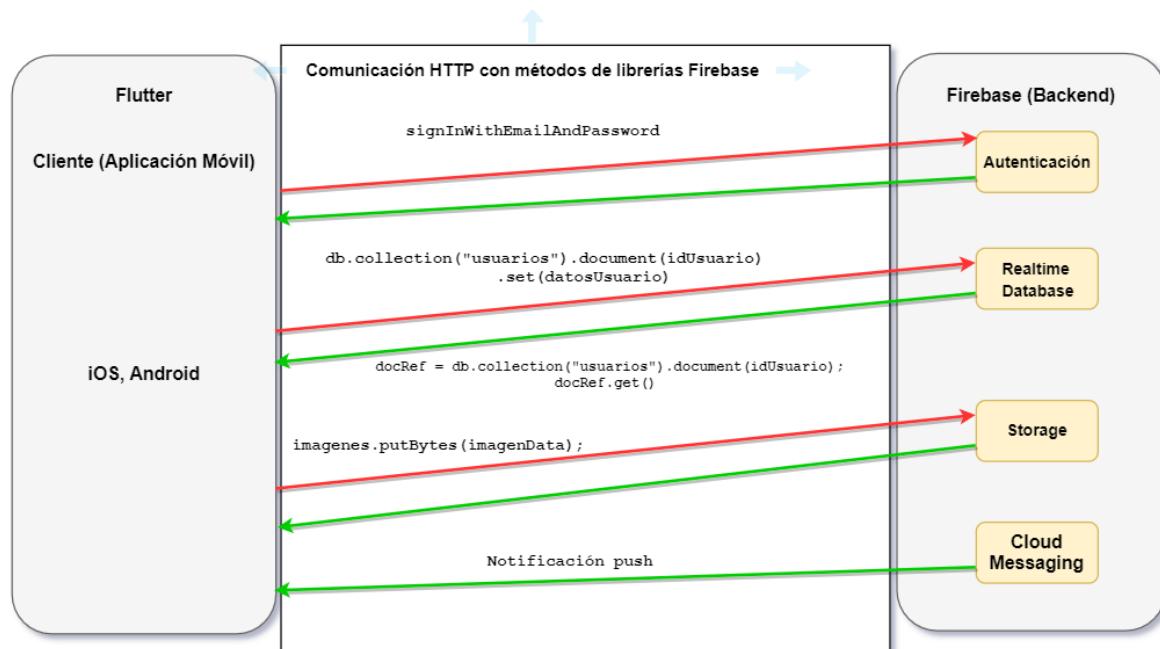


Diagrama de la arquitectura Instadroid



⁵⁵ <https://es.stackoverflow.com/questions/267497/explicaci%C3%B3n-comunicaci%C3%B3n-cliente-android-con-la-servidor-en-firebase>

Instadroid está conformado por un cliente (aplicación móvil), válido para ambos sistemas operativos (Android y iOS) y que utiliza como Backend Firebase. Como podemos ver, la parte de Firebase está dividida entre los distintos servicios que utilizaremos en el desarrollo de nuestra aplicación:

- Autenticación

Cuando el usuario se autentica lo hace contra el servicio de autenticación de Firebase, como por ejemplo `signInWithEmailAndPassword`, que autenticará a un usuario con su email y contraseña.

- Realtime database

Se trata de la base de datos en tiempo real (ya entraremos en detalles en el punto oportuno). Cuando el usuario suba una publicación o actualice su información de usuario se registrará en esta base de datos NoSQL.

- Storage

Es el servicio de almacenamiento de ficheros. Aquí almacenaremos todas las imágenes de la aplicación en vez de guardarlas en los propios documentos de la base de datos como cadena codificada a base 64.

La comunicación utilizada por las dos partes del sistema es el protocolo HTTP, como ya expliqué antes. He añadido una serie de métodos que utiliza Firebase en sus librerías para mostrar como cada uno de ellos es en realidad una petición HTTP.

IMPLEMENTACIÓN DE LA SOLUCIÓN

1. Justificación tecnológica

Hemos llegado a la parte de la implementación de la solución. En esta etapa del ciclo de vida de nuestro sistema, hay que proceder a la creación de las aplicaciones correspondientes, someterlas a pruebas, crear la documentación pertinente y capacitar a los usuarios⁵⁶.

Antes de meternos de lleno en el desarrollo de la solución, debemos exponer qué tecnologías vamos a utilizar y lo más importante, por qué vamos a utilizarlas.

⁵⁶ <https://blog.powerdata.es/el-valor-de-la-gestion-de-datos/ciclo-de-vida-de-un-sistema-de-information-fases-y-componentes>

Análisis tecnológico

Para poder realizar un análisis más ordenado de las tecnologías a utilizar, voy a dividir las tecnologías usadas entre la parte del cliente (aplicación móvil) y la parte del backend.

■ Tecnología del cliente

Partimos desde la premisa de que nuestra aplicación debe de ser multiplataforma, es decir, debe funcionar en los dos sistemas operativos principales del mercado: iOS y Android. Con esta restricción, debemos dejar de lado la idea del desarrollo totalmente nativo, por lo que Java y Swift quedan descartados.

Debemos acudir, pues, a tecnologías que nos permitan realizar un desarrollo de una solución multiplataforma. Llegados a este punto tenemos varias opciones, entre las que destacan:

■ React Native⁵⁷

Es un framework de Javascript, creado por Facebook, y que permite crear aplicaciones reales nativas para iOS y Android, basado en la librería de Javascript React. En vez de realizar una aplicación web híbrida o en HTML lo que se obtiene con este framework es una aplicación real nativa.

Utiliza el paradigma fundamental de construcción de bloques de interfaz de usuario (componentes visuales con los que interacciona el usuario) y gestiona la interacción entre los mismos utilizando las capacidades de Javascript y React.

Las características principales de este framework son:

- Compatibilidad: Permite crear aplicaciones que pueden ser ejecutadas tanto en iOS como en Android con el mismo código fuente.
- Funcionalidad nativa: Las aplicaciones creadas con este framework funciona de la misma manera que una aplicación nativa real sin el uso de un webview.
- Hot-reload: Podemos ver los cambios que ejecutamos al instante, debido a la gran velocidad de compilación.

⁵⁷ <https://www2.deloitte.com/es/es/pages/technology/articles/que-es-react-native.html>

- Sencillo de aprender: Muy fácil de leer y comprender tanto si conoces Javascript como si eres principiante en ese lenguaje de programación.
- Experiencia positiva para el desarrollador: Funcionalidades como el hot-reload y el flexbox layout engine (nos permite abstraernos de los detalles de cada layout en iOS y Android) hacen que el desarrollo sea más satisfactorio para el programador.

Otra de las cosas a destacar de este framework es el gran soporte que tiene y la extensa documentación que facilita⁵⁸.

Su principal desventaja es que utiliza un bridge de Javascript que es el que se encarga de “convertir” nuestro código Javascript en código nativo de la plataforma de ejecución, lo que lo hace un poco menos veloz que otras alternativas.

■ Ionic⁵⁹

Se trata de un framework de código abierto que se utiliza destinado al desarrollo de aplicaciones híbridas, es decir, en la que se utilizan HTML, Javascript y CSS para generarar interfaces y se “envuelven” en una aplicación nativa que la muestra en un WebView. Incluso el despliegue se realiza en plataformas nativas (Play Store y App Store).

Lo bueno, es que, junto a este framework, puedes utilizar otro framework de desarrollo front-end web como es Angular, que agilizará el desarrollo mucho más que si se realiza con Vanila Javascript (Javascript puro).

Como principales desventajas, destaca que su rendimiento puede ser ligeramente menor en comparación a una aplicación nativa⁶⁰.

■ Kotlin Native

Se trata de una tecnología que permite compilar el código escrito en Kotlin directamente a archivos binarios nativos, por lo que puede ser utilizado sin la necesidad de una máquina virtual (como sería el caso de Java). Actualmente ofrece soporte para plataformas como iOS, MacOS, Android, Windows, Linux y WebAssembly.

⁵⁸ <https://reactnative.dev/>

⁵⁹ <https://www.qualitydevs.com/2019/05/31/que-es-ionic-desarrollador-web/>

⁶⁰ <https://openwebinars.net/blog/ionic-framework-ventajas-desventajas/>

De momento es algo experimental pero que vale la pena mencionar ya que se está apostando mucho por ello y está en constante cambio⁶¹.

■ Native Script

Se trata de una alternativa bastante similar a Ionic, que permite construir aplicaciones para Android y iOS utilizando lenguajes de programación independientes del dispositivo (Javascript o Typescript). También soporta desarrollo directamente con Angular y con Vue.js mediante un complemento desarrollado por la comunidad. Además, se pueden utilizar bibliotecas de terceros como Maven y npm en las aplicaciones móviles⁶².

Como desventajas de esta herramienta encontramos⁶³:

- No tiene una curva de aprendizaje sencilla
- No se puede reutilizar buena parte del código de la web
- No tiene un conjunto de componentes de interfaz muy destacable, siendo superado por el de Ionic.

■ Xamarin⁶⁴

Es una plataforma de código abierto para crear aplicaciones multiplataforma (iOS, Android y Windows) con .NET. Se ejecuta en un entorno administrado que proporciona ventajas como la asignación de memoria y la recolección de elementos no utilizados.

Permite a los desarrolladores compartir un promedio de un 90% del código entre plataformas.

Entre sus principales desventajas (que son unas cuantas) es su coste para uso profesional empresarial. Por ejemplo, es necesario comprar una licencia de Visual Studio Professional que cuesta 1.199 dólares el primer año y 799 por renovación⁶⁵.

⁶¹ <https://www.bbvanexttechnologies.com/kotlin-native-ayer-hoy-y-manana/>

⁶² <https://es.wikipedia.org/wiki/NativeScript>

⁶³ <https://desarrolloweb.com/articulos/ionic-vs-nativescript.html>

⁶⁴ <https://docs.microsoft.com/es-es/xamarin/get-started/what-is-xamarin>

⁶⁵ <https://inmediatum.com/blog/ingenieria/ventajas-y-desventajas-de-apps-desarrolladas-en-xamarin/>

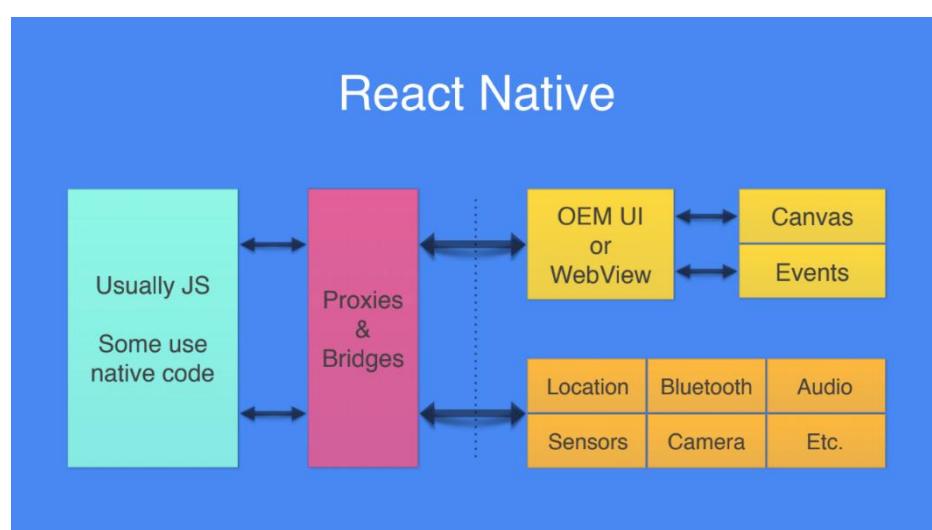
Además, personalmente, pienso que cualquiera de las alternativas aquí mencionadas es mejor.

■ Flutter⁶⁶

Es un framework de código abierto desarrollado por Google para crear aplicaciones nativas de forma fácil, rápida y sencilla. Su principal ventaja radica en que el código que genera es 100% nativo para cada plataforma, consiguiendo un rendimiento y una UI idénticos a las aplicaciones nativas tradicionales. Pese a las promesas de otros frameworks con el mismo propósito, se trata del único en generar código 100% nativo y ejecutable en ambas plataformas.

Otras ventajas:

- Experiencia de usuario. Flutter incluye Material Design de Google y Cupertino de Apple por lo que la experiencia de usuario es óptima y las interfaces idénticas a las de las aplicaciones desarrolladas de forma nativa.
- Tiempo de carga. Con Flutter se experimentan tiempos de carga por debajo de un segundo en cualquiera de las plataformas.
- Desarrollo ágil y rápido: Gracias al hot-reload.
- Se puede programar desde cualquier sistema operativo.
- Rendimiento superior a su máximo competidor, React Native, ya que no utiliza un bridge⁶⁷.



⁶⁶ <https://www.qualitydevs.com/2019/07/05/que-es-flutter/>

⁶⁷ <https://www.bbvanexttechnologies.com/es-flutter-el-framework-del-futuro/>

Además, se está viendo que este Framework tiene un gran futuro por delante, debido a que Google está desarrollando su sistema operativo Fucsia, que si finalmente ve la luz sería un salto enorme para Flutter.

Finalmente, es necesario hacer mención del lenguaje que utiliza Flutter. Se trata de Dart, un lenguaje creado también por Google, sencillo de aprender ya que guarda similitudes con C++, C# o Java.

Lo bueno de Dart es que funciona de forma similar a Java, ejecutándose sobre una máquina virtual (Dart VM) y permite a Flutter evitar la necesidad de tener otra capa de lenguaje declarativo como XML para realizar las interfaces, porque el propio layout es definido en Dart y es fácil de entender y rápido de desarrollar⁶⁸.

Conclusión tecnológica para la parte del cliente

Para desarrollar Instadroid, voy a utilizar Flutter. Me ha costado decidirme entre Flutter y React Native, pero finalmente me convenció Flutter porque parece ser que ofrece una mejor performance y tiene detrás una gran comunidad y al mismísimo Google⁶⁹.

Además, estos son otros de los motivos:

- No necesito utilizar ningún lenguaje para definir las vistas, ni HTML, ni XML, utilizando el propio Dart y de forma sencilla podré hacerlo.
- Ofrece muchas facilidades para aplicaciones que realizan cargas de datos en tiempo real como los Widget Future
- Muestra unas interfaces increíbles, totalmente iguales a las de aplicaciones nativas
- Al ser un producto de Google como Firebase tiene una buenísima integración con esta última tecnología
- Pienso que mi desarrollo va a ser mucho más ágil y rápido
- Puedo utilizar el Hot Reload para ver cambios y realizar pruebas al momento
- Por gusto personal de la tecnología

⁶⁸ <https://alexmarket.medium.com/flutter-o-nativescript-707b7ffdbceb>

⁶⁹ <https://flutter.dev/>

- Porque al estar en la parte de móviles en mis prácticas me gustaría saber y entender las tecnologías que invadirán el desarrollo móvil en los próximos años



■ Tecnología del servidor

Del lado del servidor serían muchas las tecnologías que podríamos utilizar para construir una API REST, como:

- SpringBoot
- Django
- Laravel
- Node js con express
- Deno
- .NET

Incluso podríamos utilizar sockets de cualquier lenguaje de programación que trabaje con ellos como Java o Python.

Además, estas tecnologías podrían conectar con cualquier tipo de base de datos tanto SQL como NoSQL, como MySQL, Oracle o Mongo.

Nuestra principal limitación a la hora de elegir una tecnología para el backend viene marcada por la necesidad de la aplicación de mostrar datos en tiempo real, lo que nos hace acudir a distintas soluciones que ofrecen un backend ya realizado en vez de realizar nosotros el nuestro propio.

Entre estas alternativas destacan⁷⁰:

■ Back4App

Una solución sencilla y diseñada para aplicaciones móviles y sitios web. Ofrece una manera más rápida y sencilla de alojar y administrar aplicaciones. Es una solución de código abierto que ofrece múltiples herramientas e integraciones que nos ayudan como desarrolladores a construir aplicaciones de una forma mucho más rápida. Algunas de sus características son:

- Modelo de datos
- APIs GraphQL
- Base de datos en tiempo real
- Autenticación de dos factores
- Copias de seguridad automatizadas
- Posibilidad de utilizarlo directamente como una API Rest

■ Parse

Se trata de una plataforma que ofrece funcionalidades similares a la anterior, pero es más fácil de utilizar en comparación con otras opciones similares. De tal manera que ofrece características y herramientas para desarrollar y administrar servicios de backend:

- Base de datos en tiempo real
- Integración con redes sociales
- Autenticación a través de correo electrónico y contraseña
- Funciones de seguridad
- Alojamiento SSL

■ AWS Amplify

De código abierto, Amazon ha creado un gran entorno que permite a los desarrolladores a crear aplicaciones sin servidor, fácilmente integrable con cualquier frontend.

⁷⁰ <https://blog.back4app.com/es/las-mejores-alternativas-a-firebase/>

Destaca su interfaz de usuario fácil de usar y navegar y sus extensas librerías que tienen la capacidad de permitir que los desarrolladores conecten sus soluciones backend con interfaces móviles.

Se trata de la alternativa y competitora más fuerte de Firebase.

Algunos de los servicios que ofrece:

- Notificaciones push
- Marketing vía correo electrónico
- Base de datos en tiempo real
- Geolocalización

■ Firebase⁷¹

Se trata de una plataforma móvil creada por Google cuya principal función es desarrollar y facilitar la creación de apps de elevada calidad de forma rápida. La plataforma se encuentra en la nube y está disponible para distintas plataformas como iOS, Android y Web. Contiene diversas funciones para que cualquier desarrollador pueda combinar y adaptar la plataforma a sus necesidades.

Características principales:

- Desarrollo: Firebase permite la creación de mejores apps minimizando el tiempo de optimización y desarrollo mediante diferentes funciones, entre las que destacan la detección de errores y el testeo. Poder almacenar todo en la nube o poder configurarla de forma remota son características destacables de la plataforma.
- Analítica: Tener un control máximo del rendimiento de la app mediante métricas analíticas desde un único panel y de forma gratuita.
- Poder de crecimiento: Permite gestionar de manera fácil todos los usuarios de las aplicaciones, con el añadido de captar nuevos usuarios mediante invitaciones o notificaciones.
- Rapidez: Implementar Firebase puede ser fácil y rápido gracias a su API, que es muy intuitiva. De esta manera, podremos invertir más

⁷¹ <https://www.iebschool.com/blog/firebase-que-es-para-que-sirve-la-plataforma-desarrolladores-google-seo-sem/>

tiempo en resolver los problemas de las apps cliente y evitar perder tiempo en la creación de una infraestructura compleja.

- Agilidad: Firebase ofrece apps multiplataforma con APIs integradas a SDK individuales como iOS, Android y Javascript
- Extensa documentación. La documentación⁷² de Firebase es muy extensa y sencilla de entender y aplicar. Además, cuenta con canal de YouTube⁷³ y sus desarrolladores participan activamente en GitHub y StackOverflow dando soporte al que lo necesite.

¿Qué servicios nos ofrece Firebase?⁷⁴. Varios. Entre los que destacan:

- Realtime Database / Cloud Firestore

Son las bases de datos en tiempo real, se alojan en la nube y son NoSQL. Cloud Firestore es la nueva opción.

Firebase envía automáticamente eventos a las aplicaciones cuando los datos cambian.

- Autenticación de usuarios

Firebase ofrece un sistema de autenticación que permite el registro, como el acceso utilizando otras plataformas externas (como Google o Twitter).

- Almacenamiento en la nube

Firebase cuenta con un sistema de almacenamiento, donde los desarrolladores pueden guardar ficheros y sincronizarlos. Es personalizable mediante reglas. Es muy útil para guardar fotografías, por ejemplo.

- Cloud Messaging

Servicio de envío de notificaciones y mensajes a diversos usuarios en tiempo real y a través de varias plataformas.

⁷² <https://firebase.google.com/docs/auth/android/start?hl=es-419>

⁷³ <https://www.youtube.com/user/Firebase>

⁷⁴ <https://www.digital55.com/desarrollo-tecnologia/que-es-firebase-funcionalidades-ventajas-conclusiones/>

Conclusión tecnológica para la parte del servidor

Para el backend de Instadroid, como bien ya he venido exponiendo, voy a utilizar Firebase. Me parece la mejor alternativa y además debemos cumplir el requisito de carga de datos en tiempo real.

De igual manera, estos son los servicios a utilizar en Firebase:

- Servicio de autenticación de usuarios: Nuestros usuarios tendrán la opción de autenticarse con Google y Twitter, algo que ya nos ofrece Firebase.
- Realtime Database: Será nuestra base de datos NoSQL, en la que almacenaremos los datos de la aplicación.
- Cloud Storage: Lo utilizaremos para guardar las distintas fotos que suban los usuarios.
- Cloud Messaging: Para enviar las notificaciones.



2. Aspectos esenciales de la implementación

Las funcionalidades por implementar en mi proyecto son: un login y registro, listar las publicaciones que suben los usuarios a Instadroid, subir, editar, borrar y visualizar una publicación, así como poder ver la localización desde la que esa publicación se ha subido y la localización de todas las publicaciones subidas.

Como extra, he añadido la opción de poder dar un me gusta a la publicación.

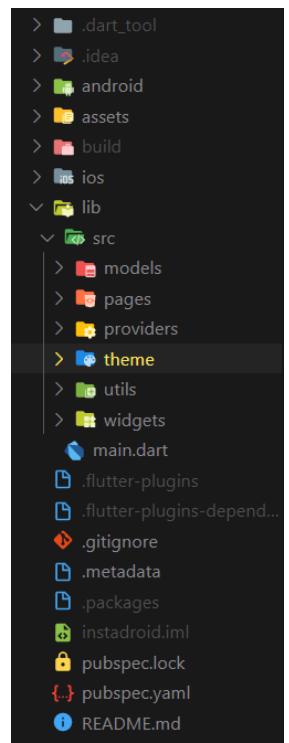
Vamos a ir analizando uno a uno estos aspectos y explicando los puntos más relevantes de su implementación.

Antes de empezar con todo, debemos comprender el funcionamiento básico de Flutter. Flutter se basa en widgets, un widget es la forma que tiene Flutter de definir los componentes visuales de una aplicación y el comportamiento de estos. Se trata de un concepto muy parecido a los componentes de React⁷⁵. Cualquier componente de Flutter es un widget, desde un texto hasta una pantalla entera (ya que los widgets suelen tener otros widgets dentro).

Hay dos tipos de widgets: los stateless y los stateful. Los stateless no mantienen el estado cuando se redibujan y los stateful si lo hacen y nosotros podemos elegir que elementos guardar o cambiar en un cambio de estado (también podemos forzar cambios de estado).

La estructuración de los proyectos Flutter sigue más o menos este aspecto:

Tenemos los modelos de datos, las pages que son las pantallas, los providers o servicios que serán las clases que nos traerán los datos necesarios para que los widgets de las pages muestren. Tenemos una carpeta para nuestro tema personalizado de la aplicación, los utils con métodos de utilidad para la aplicación. Además, tenemos la carpeta de widgets que al ser repetidos en la aplicación se crean de forma individual para no repetir información. De igual manera tenemos nuestro main y a destacar el pubspec.yaml, en el que se incluyen las dependencias del proyecto. El aspecto de ese archivo es este:



⁷⁵ <https://flutter.dev/docs/development/ui/widgets-intro>

```

# Use with the CupertinoIcons class for iOS
cupertino_icons: ^1.0.0
font_awesome_flutter: ^8.11.0
http: ^0.12.2
shared_preferences: ^0.5.12+4
provider: ^4.3.2+3
image_picker: ^0.6.7+17
firebase_core: ^0.5.3
firebase_storage: ^5.2.0
permission_handler: ^5.0.1+1
geolocator: ^6.1.13
google_maps_flutter: ^1.0.6

dev_dependencies:
  flutter_test:
    | sdk: flutter

# For information on the generic Dart part of
# following page: https://dart.dev/tools/pub/p

# The following section is specific to Flutter
flutter:

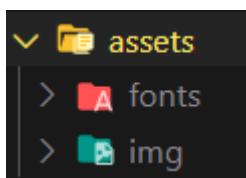
  # The following line ensures that the Material
  # included with your application, so that you
  # the material Icons class.
  uses-material-design: true

  # To add assets to your application, add an
  assets:
    | - assets/img/
    #   - images/a_dot_ham.jpeg

    # An image asset can refer to one or more resolution-specific
    # files where an suffix for the file denotes the
    # resolution/density: px, @2x, @3x, etc. For example, values/深色疑惑表情包.png
    # will be loaded
    # as values/深色疑惑表情包@2x.png at double the resolution
    # of values/深色疑惑表情包.png.
    # A density-metric specific file (e.g. values-deep-orange-600dp.png)
    # will be loaded
    # as values-deep-orange-600dp.png.
    # Additionally, a regular file (e.g. values-deep-orange.png)
    # will be loaded
    # as values-deep-orange-480dp.png.
    # Note that if multiple files share the same base name, the most
    # recent version will overwrite the previous one.
    # See https://flutter.dev/assets-and-images/#multiple-sizes

```

Como se puede observar en este archivo se incluyen las dependencias de la aplicación, así como si queremos incluir recursos locales como imágenes o fuentes (la carpeta assets):



En esta carpeta incluiremos recursos como las fuentes y las imágenes.

Una vez hecha esta idea más o menos general, es necesario comprender que cada aplicación en Flutter tiene un archivo main, con su método main que se encarga de ejecutar la aplicación. En este método main se ejecuta myApp que es el widget padre de toda la aplicación (dónde se van a encontrar todos los demás widgets).

Dentro de este widget myApp se definen aspectos como el tema de la aplicación y las rutas de esta (podemos tratar una aplicación Flutter como una aplicación web, especificando las rutas de la aplicación mediante cadenas de texto, cada una será una pantalla de nuestra aplicación).

```

void main() async{
    //Arrancamos las preferencias de usuario para guardar el token
    WidgetsFlutterBinding.ensureInitialized();
    final prefs = new UserPreferences();
    await prefs.initPrefs();
    runApp(MyApp());
}

class MyApp extends StatelessWidget {
    @override
    Widget build(BuildContext context) {
        return MaterialApp(
            debugShowCheckedModeBanner: false,
            title: 'Instadroid',
            //Ruta inicial de nuestra aplicación
            initialRoute: 'login',
            /*Pantallas que va a tener la aplicación, en flutter se pueden manejar
            como rutas, similar al desarrollo web*/
            routes: {
                'login'      : (BuildContext context) => LoginPage(),
                'registro'   : (BuildContext context) => RegistroPage(),
                'home'       : (BuildContext context) => HomePage(),
                'foto'        : (BuildContext context) => UploadPhotoPage(),
                'localizacion': (BuildContext context) => PhotoLocationPage(),
                'mapa'        : (BuildContext context) => LocalizacionFotosPage(),
            },
            //Tema de la aplicación
            theme: myTheme,
        );
    }
}

```

Como se puede ver, se definen las rutas que va a tener la aplicación, así como la ruta inicial de la misma, que será el login. También se define el tema, que es este:

```

import 'package:flutter/material.dart';
//Tema a nivel global de la aplicación
final myTheme = ThemeData.light().copyWith(
    primaryColor: Color.fromRGBO(90, 255, 231, 1.0),
    buttonColor: Color.fromRGBO(8, 198, 171, 1.0),
);

```

Copiamos el tema claro por defecto y modificamos el color primario y el color de los botones.

Una vez entendido de forma general como funciona un proyecto flutter, podemos pasar a examinar los aspectos relevantes de la implementación de la funcionalidad acordada.

■ Aspectos relevantes del login y registro

En primer lugar, suponiendo que el usuario se encuentra ya registrado, podrá hacer login.

Este login se realiza utilizando el servicio de autenticación de Firebase, que registrará a los usuarios y nos devolverá un token de registro único para cada usuario.

Sobre la interfaz, se trata de una pantalla que consta de una distribución en columna dentro de la cual se encuentra un formulario, que consta de dos cajas de texto y dos botones (uno para realizar el login y otro para acceder a la pantalla de registro):

```
@override
Widget build(BuildContext context) {
  return Scaffold(
    resizeToAvoidBottomPadding: false,
    body: Stack(
      children: <Widget> [
        BackgroundImage(),
        SafeArea(
          child: Form(
            key: formKey,
            child: Column(
              mainAxisAlignment: MainAxisAlignment.center,
              children: <Widget> [
                AppTitle(
                  color: Colors.white,
                  size: 50.0,
                ), // AppTitle
                SizedBox(height: 20),
                Container(
                  padding: EdgeInsets.symmetric(horizontal: 15),
                  child: _emailInput(),
                ), // Container
                SizedBox(height: 20),
                Container(
                  padding: EdgeInsets.symmetric(horizontal: 15),
                  child: _passwordInput(),
                ), // Container
                SizedBox(height: 100),
                (_cargando)
                  ? CircularProgressIndicator(backgroundColor: myTheme.primaryColor)
                  : Container(),
                SizedBox(height: 150),
                _loginButton(context),
                SizedBox(height: 20),
                registerButton(context),
              ],
            ),
          ),
        ),
      ],
    ),
  );
}
```

Esta sería la distribución a nivel general, y a nivel particular las cajas de texto y botones siguen el siguiente aspecto (igual para todos)

```

Widget _emailInput(){
  return Container(
    decoration: BoxDecoration(
      color: Color.fromRGBO(255,255,255, 0.2),
      borderRadius: BorderRadius.circular(20),
    ), // BoxDecoration
    padding: EdgeInsets.symmetric(horizontal: 5, vertical: 6),
    child: TextFormField(
      keyboardType: TextInputType.emailAddress,
      decoration: InputDecoration(
        icon: Icon(Icons.mail_outline, color: myTheme.primaryColor),
        labelText: 'email',
        labelStyle: TextStyle(
          color: myTheme.primaryColor,
        ), // TextStyle
        border: OutlineInputBorder(
          borderRadius: BorderRadius.circular(20.0)
        ), // OutlineInputBorder
        enabledBorder: OutlineInputBorder(
          borderRadius: BorderRadius.circular(25.0),
          borderSide: BorderSide(
            color: myTheme.primaryColor,
            width: 2.0,
          ), // BorderSide
        ), // OutlineInputBorder
      ), // InputDecoration
      onSaved: (value) => user.email = value,
      validator: (value){
        if(utils.isAEmail(value)){
          return null;
        }else{
          return 'Introduce un email válido';
        }
      }
    ), // TextFormField
  ); // Container
}

```

Como podemos ver es muy sencillo crear cualquier elemento en Flutter, ya que muchos vienen ya definidos y solo tenemos que personalizarlos. En este caso las cajas de texto de los formularios (text input), aparte de tener definido su aspecto pueden tener definido que tipo de teclado van a mostrar (email, numérico) así como que valores vamos a poder obtener de ellas cuando hagamos “submit” del formulario (onSaved). También podemos añadir un validador que devolverá null si lo introducido es válido y si no devolverá el texto que mostrará al usuario en pantalla.

En cuanto a los botones, tienen el siguiente aspecto:

```

Widget _loginButton(BuildContext context){
  return Container(
    width: 300,
    child: RaisedButton(
      child: Container(
        padding: EdgeInsets.symmetric(horizontal:80.0, vertical: 15.0),
        child: Text('Login'),
      ), // Container
      shape: RoundedRectangleBorder(
        borderRadius: BorderRadius.circular(20.0),
      ), // RoundedRectangleBorder
      elevation: 0.0,
      color: myTheme.buttonColor,
      textColor: Colors.white,
      onPressed: () => _submit(context),
    ), // RaisedButton
  ); // Container
}

```

Poco que decir respecto a ello, solo que se personaliza el botón y se realiza su evento onPressed.

Cuando el usuario ha escrito sus credenciales y pulsa en el botón de login, se realiza el “submit” del formulario:

```

void _submit(BuildContext context) async {
  final loginProvider = LoginAuthProvider();
  if(!formKey.currentState.validate()){
    return;
  }
  formKey.currentState.save();
  setState(() {
    _cargando = !_cargando;
  });
  bool isLoggedIn = await loginProvider.loginFirebase(user);
  setState(() {
    _cargando = !_cargando;
  });
  if(isLoggedIn){
    Navigator.pushReplacementNamed(context, 'home');
  }else{
    utils.showAlert(
      context,
      [
        FlatButton(
          child: Text('Ok'),
          onPressed: () {
            Navigator.of(context).pop();
          }
        ) // FlatButton
      ],
      'Login',
      'Usuario no encontrado, por favor crea una cuenta',
    );
  } //Realizar login en firebase
}

```

Como se puede ver primero se valida el formulario (se llama a los validators que hemos visto antes que incluyen los text input). Si es válido se realiza el save que llama al onSave de los text input. En este caso se crea un objeto usuario general en la pantalla de login al que se le van seteando los valores obtenidos de los text input (en su método onSave).

Clase usuario⁷⁶:

```
1 import 'dart:convert';
2
3 Usuario usuarioFromJson(String str) => Usuario.fromJson(json.decode(str));
4
5 String usuarioToJson(Usuario data) => json.encode(data.toJson());
6
7 class Usuario {
8
9   String idFire;
10  String nombreUsuario;
11  String email;
12  String password;
13  String fotoUrl;
14
15  Usuario({
16    this.idFire,
17    this.nombreUsuario,
18    this.email,
19    this.password,
20    this.fotoUrl,
21  });
22
23 //Aquí si tenemos el id porque firebase nos lo mandará con id
24 factory Usuario.fromJson(Map<String, dynamic> json) => Usuario(
25   idFire      : json["idFire"],
26   nombreUsuario : json["nombreUsuario"],
27   email       : json["email"],
28   fotoUrl     : json["fotoUrl"],
29 );
30
31 //Cuando mapeamos a Json es para enviarlo a firebase y en firebase solo queremos tener estos datos
32 Map<String, dynamic> toJson() => {
33   "nombreUsuario" : nombreUsuario,
34   "email"        : email,
35   "fotoUrl"      : fotoUrl,
36 };
37 }
```

Una vez llamado al saved, ponemos nuestra variable de estado cargando a verdadero, para mostrar un indicador en la interfaz de que se está realizando el login. Mientras tanto de forma asíncrona hacemos login contra Firebase (por eso el método submit es async porque hace await del proceso de login).

Para realizar el login y registro en Firebase, he decidido centralizar todo en un provider de login. En ese provider el método de login es el siguiente:

⁷⁶ Como podemos ver las clases en dart permiten realizar constructores con nombre. En este caso tenemos uno que mapea de json a clase usuario y un método que hace lo contrario (de modelo a json)

```

//Login contra firebase
Future<bool> loginFirebase(Usuario usuario) async {
  final url = '${_baseUrl}${_loginServiceToken}${_apiKey}';
  final userData = {
    'email' : usuario.email,
    'password' : usuario.password,
    'returnSecureToken' : true
  };
  return await _handleResponse(url, json.encode(userData));
}

//La petición http y su procesamiento
Future<bool> _handleResponse(String url, String requestBody) async{
  final resp = await http.post(url, body: requestBody);
  Map<String, dynamic> decodedResponse = json.decode(resp.body);
  //Guardamos el token en las preferencias de usuario
  userPreferences.token = decodedResponse['idToken'];
  return decodedResponse.containsKey('idToken');
}

```

Lo primero destacable es el objeto Future que devuelve el método. Se trata de algo específico de dart que marca que para recibir ese método se debe hacer de forma asíncrona. Lo que devolverá el método será lo que está entre llaves del future, en este caso un booleano. Se suele decir “un future que resuelve un booleano”.

Como se puede ver la url se forma concatenando:

```

final String _apiKey           = 'AIzaSyA6R47Ui5Wep9u44y4BKc3cYk3ZX69J6LY';
final String _baseUrl          = 'https://identitytoolkit.googleapis.com/v1/accounts:';
final String _registerServiceToken = 'signUp?key=';
final String _loginServiceToken   = 'signInWithPassword?key=';

```

Formamos los datos de nuestro usuario y hacemos un post a la url. Si la respuesta contiene el idToken, entonces el usuario está logueado correctamente.

Para el registro, la forma de proceder es exactamente la misma, los text input tienen el mismo aspecto y los botones son iguales. El método del provider es el siguiente:

```

//Damos de alta un usuario en Firebase
Future<bool> createUserFirebase(Usuario usuario) async {
  final url = '${_baseUrl}${_registerServiceToken}${_apiKey}';
  final userData = {
    'email' : usuario.email,
    'password' : usuario.password,
    'returnSecureToken' : true
  };
  return await _handleResponse(url, json.encode(userData));
}

```

El `_handleResponse` es el mismo que el del login por lo que funciona igual un post en el que si recibimos el token, el usuario está correctamente registrado.

Como se puede ver en el `_handleResponse`, se guardan el token en las preferencias del usuario. Esto no es nada más que las shared prefs, una memoria persistente que nos valdrá para guardar algunos datos de nuestra aplicación y poder acceder a ellos de forma global.

En este caso guardamos el token:

```
/*Clase para las preferencias de usuario, nos servirá para almacenar
en el almacenamiento del dispositivo algunos datos relevantes, en nuestro
caso el token de auth de firebase y el id que el usuario tiene en la bbdd de
firebase */
class UserPreferences{

    //Hacemos un singleton para tener una instancia en toda la aplicación
    static final UserPreferences _instance = new UserPreferences._internal();

    factory UserPreferences(){
        return _instance;
    }

    UserPreferences._internal();

    SharedPreferences _prefs;

    //Lista de publicaciones gustadas
    List<String> _publisMeGusta;

    initPrefs() async {
        this._prefs = await SharedPreferences.getInstance();
        this._publisMeGusta = List();
    }

    // Get y sets para los datos a almacenar
    // ?? → significa que si el elemento es nulo que devuelva en este caso un carácter
    get token{
        return _prefs.getString('token') ?? '';
    }

    set token(String token){
        _prefs.setString('token', token);
    }
}
```

Las shared prefs se inician justo al iniciar la aplicación, en su método main:

```
void main() async{
    //Arrancamos las preferencias de usuario para guardar el token
    WidgetsFlutterBinding.ensureInitialized();
    final prefs = new UserPreferences();
    await prefs.initPrefs();
    runApp(MyApp());
}
```

La única diferencia que tiene el registro con el login es que, en el submit del registro, además, insertaremos el usuario recién creado en realtime database de Firebase, nuestra base de datos NoSQL:

```
void _submit(BuildContext context) async {
    final registerProvider = LoginAuthProvider();
    final userProvider = UserProvider();
    String registerResult;
    if(!formKey.currentState.validate()){
        return;
    }
    formKey.currentState.save();
    setState(() {
        _cargando = !_cargando;
    });
    bool registrado = await registerProvider.createUserFirebase(user);
    user.photoUrl = '';
    bool guardadoBd = await userProvider.agregarUsuario(user);
    setState(() {
        _cargando = !_cargando;
    });
    if(registrado && guardadoBd){
        registerResult = 'Correctamente registrado';
    }else{
        registerResult = 'Ha ocurrido un problema con el registro';
    }
    _giveFeedback(registerResult, registrado);
}
```

Como podemos ver, la variable `guardadoBd` llama al `userProvider` (otro fichero con otro provider que se encarga de manejar a los usuarios en realtime database) para que agregue al usuario (de forma asíncrona):

```
//Insertamos los datos del usuario en realtime database de firebase
Future<bool> agregarUsuario (Usuario usuario) async {
    final url = '${_mainUrl}/usuarios.json?auth=${_prefs.token}';
    final resp = await http.post(url, body: usuarioToJson(usuario));
    final Map<String, dynamic> decodedData = json.decode(resp.body);
    //Si lo que nos devuelve contiene name ese es el id que firebase le ha dado
    if(decodedData.containsKey('name')){
        //Lo almacenamos en las shared prefs
        _prefs.idUsuarioLogueado = decodedData['name'];
        //Borramos los me gusta del usuario anterior
        _prefs.publicacionesGustadasGuardadas = false;
    }
    return decodedData.containsKey('name');
}
```

Como se puede observar, aquí ya utilizamos el guardado en nuestras shared prefs para autorizarnos a realizar una escritura en la base de datos de Firebase (ya que así lo hemos definido en nuestras reglas):

```

instadroid-45787-default-rtdb
  +-- publicaciones
  +-- usuarios
      +-- -MPTcgnreztzb_hs9s6j
      +-- -MPTEwK0kpU-TdfjimF
      +-- -MPU9kxQTwVhUl3-sh
      +-- -MPUQBovw_YCtdlWcShJ
      +-- -MPkqNQoScPNsJYd0CQ
      +-- -MQ7sdQ4ioeNTEx93DzA
      +-- -MQ8SXa7xVeGmOW-okUt
      +-- -MQ8bfPey40o_1Ma-ys
      +-- -MQ8c0LxF2Gase9yTaO
      +-- -MQ8gbPpQfTomjXynLi
      +-- -MQ8npM0eg5QydinTCKE3
  
```

Ubicación de la base de datos: Bélgica (europe-west1) [beta]

<https://console.firebaseio.google.com/u/2/project/instadroid-45787/database/instadroid-45787-default-rtdb/data/>

(Nuestra base de datos)

```

rules
  "rules": {
    ".read": "auth != null",
    ".write": "auth != null"
  }
  
```

(Sus reglas, no podemos ni leer ni escribir si no estamos auth)

Como podemos ver en el método del registro, si Firebase devuelve un json que contiene “name” será que habrá registrado correctamente al usuario y ese será su id en Firebase.

Guardamos ese id en las shared prefs (así cuando subamos una publicación, tendremos el id del usuario que la ha subido):

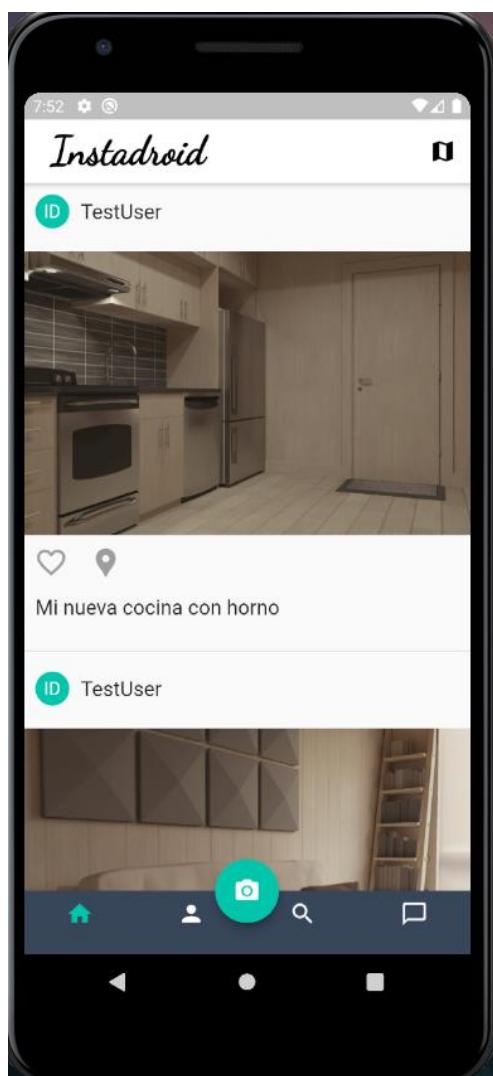
```
get idUsuarioLogueado{
    return _prefs.getString('id') ?? '';
}

set idUsuarioLogueado(String id){
    _prefs.setString('id', id);
}
```

■ Aspectos relevantes del timeline de publicaciones

Se trata de la pantalla en la que el usuario va a ver las publicaciones subidas a Instadroid.

Adjunto foto para poner un poco más en contexto:



Primero, uno de los aspectos relevantes es la presencia del menú inferior y de la barra superior, ya que se trata de un widget que contiene a las demás páginas de la aplicación y que siempre va a estar presente:

```
Widget build(BuildContext context) {
  return ChangeNotifierProvider(
    create : (_) => _Navegacion(),
    child: Scaffold(
      appBar: AppBar(
        title: Transform(
          transform: Matrix4.translationValues(-80.0, 0.0, 0.0),
          child: AppTitle(
            color: Colors.black,
            size: 35.0,
          ), // AppTitle
        ), // Transform
        backgroundColor: Colors.white,
        centerTitle: false,
        actions: [
          IconButton(
            icon: Icon(Icons.map, color: Colors.black),
            onPressed: () async {
              final publicacionesProvider = PublicacionesProvider();
              final List<Publicacion> listaPublis = await publicacionesProvider.lista();
              Navigator.pushNamed(context, 'mapa', arguments: listaPublis);
            },
          ), // IconButton
        ],
      ), // AppBar
      body: _Pages(),
      floatingActionButton: FloatingActionButton(
        child: Icon(Icons.camera_alt, color: Colors.white),
        backgroundColor: myTheme.buttonColor,
        onPressed: () {
          // Navegación a la página de subir foto
          Navigator.pushNamed(context, 'foto');
        },
      ), // FloatingActionButton
      floatingActionButtonLocation: FloatingActionButtonLocation.miniCenterDocked,
      bottomNavigationBar: _BottomNavigation() ,
    ), // Scaffold
  ); // ChangeNotifierProvider
```

Como se puede observar es un Scaffold (el Widget que utiliza flutter para definir páginas con cabecera, cuerpo, fabs, etc), que tiene su appBar como hemos visto y su body, que tiene dentro un widget y su bottomNavigationBar que tiene dentro otro widget. Además, incluye el fab.

El widget del body:

```
class _Pages extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    final navegacion = Provider.of<_Navegacion> (context);

    return PageView(
      controller: navegacion.pageController,
      physics: NeverScrollableScrollPhysics(),
      children: [
        TimelinePage(),
        ProfilePage(),
      ],
    ); // PageView
  }
}
```

Se trata de un pageview, en el que incluiremos las distintas páginas que vamos a mostrar como pantallas de nuestra aplicación.

El widget del bottomNavigation:

```
@override  
Widget build(BuildContext context) {  
  final navegacion = Provider.of<_Navegacion>(c  
  
  return Theme(  
    data: Theme.of(context).copyWith(  
      canvasColor: Color.fromRGBO(55, 70, 91, 1.  
    ),  
    child: BottomNavigationBar(  
      type: BottomNavigationBarType.fixed,  
      currentIndex: 0,  
      onTap: (newIndex){  
        setState(() {  
          _index = newIndex;  
        });  
        navegacion.currentPage = newIndex;  
      },  
      items: [  
        BottomNavigationBarItem(  
          icon: Icon(  
            Icons.home,  
            color: (_index == 0)  
              ? myTheme.buttonColor  
              : Colors.white,  
          ), // Icon  
          label: '',  
        ), // BottomNavigationBarItem  
        BottomNavigationBarItem(  
          icon: Icon(  
            Icons.person,  
            color: (_index == 1)  
              ? myTheme.buttonColor  
              : Colors.white,  
          ), // Icon  
          label: '',  
        ), // BottomNavigationBarItem
```

Se trata del que va a dibujar

Nuestro menú inferior. Cada

Item es un elemento del menú.

Cuando tocamos cada ítem, se dispara el onTap, que cambiará el color del ítem seleccionado.

Finalmente, para realizar la lógica de navegación de una pantalla a otra, utilizaremos el paquete de Flutter denominado provider. Se trata, en muy resumidas cuentas, de un elemento que nos permite comunicar a los widgets entre sí, manteniendo un “estado global en nuestra aplicación”. Para ello he creado una clase que tiene el controlador de la navegación y el índice que tienen el bottomNavigationBar, para comunicar al bottomNavigation con el PageView que tenemos en widgets separados y que reaccionen a los eventos que se producen en ellos:

```

class _Navegacion with ChangeNotifier {

    int _currentPage = 0;
    PageController _pageController = PageController();

    int get currentPage => this._currentPage;

    set currentPage (int newIndex){
        this._currentPage = newIndex;
        this._pageController.animateToPage(
            newIndex,
            duration: Duration(milliseconds:250),
            curve: Curves.easeOut
        );
        notifyListeners();
    }

    PageController get pageController => this._pageController;
}

```

Por eso tanto el widget del pageview como el del bottom navigation tienen esta línea:

```

final navegacion = Provider.of<_Navegacion> (context);

```

Como esta clase tiene un mixing (similar a una interfaz Java) con change notifier, podemos hacer que sea el provider de esos dos widgets y poder comunicarlos. Es un concepto un poco abstracto, pero es muy simple de implementar.

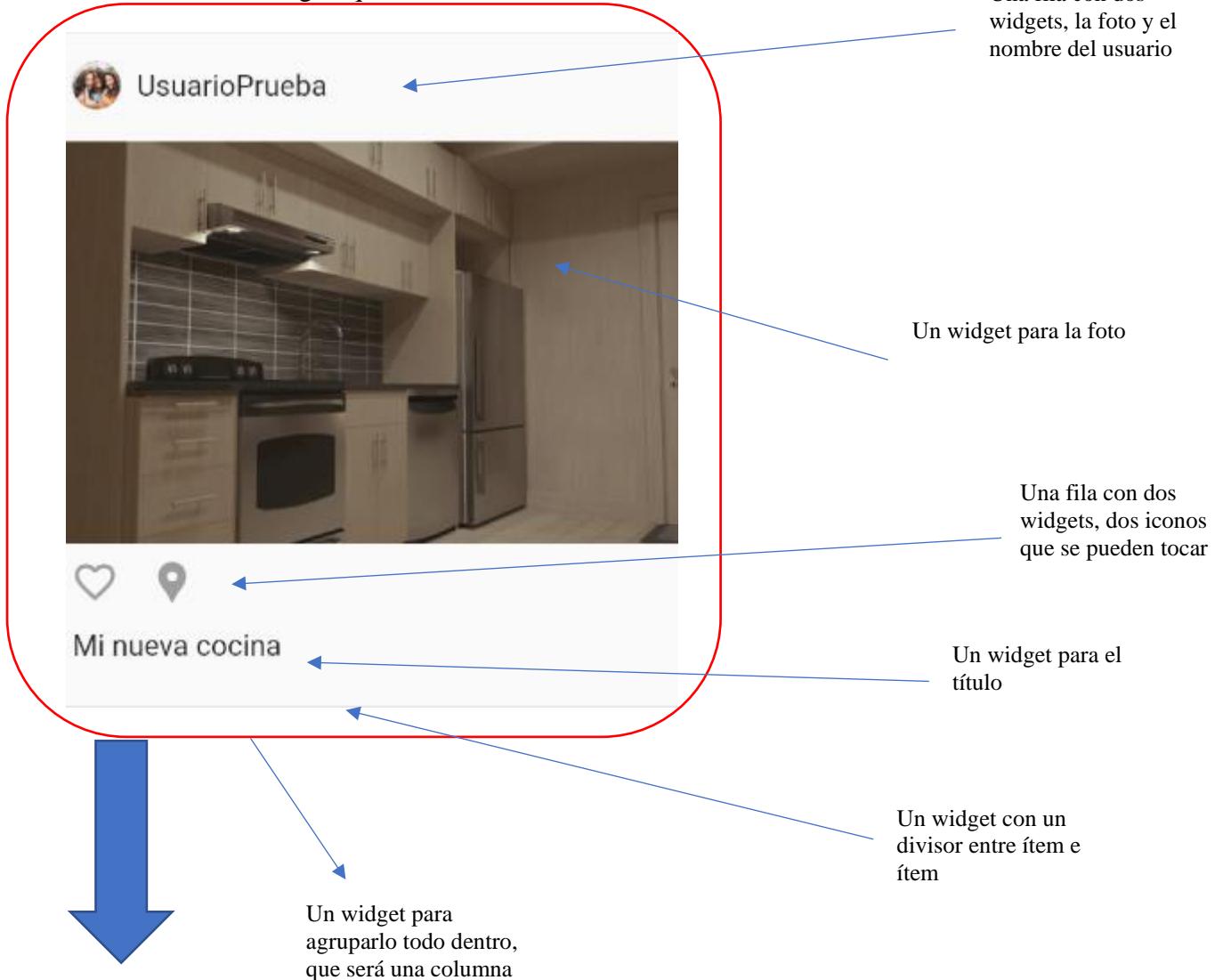
Volviendo de nuevo con la pantalla del timeline propiamente dicho, hay que primero dejar claro que se trata de un listview que nos va a ir mostrando un ítem por cada publicación que haya en nuestra base de datos.

```

class TimelinePage extends StatelessWidget {
    @override
    Widget build(BuildContext context) {
        return Scaffold(
            body: _PublicacionesList(),
        ); // Scaffold
    }
}

```

Cada ítem es un widget, que a su vez contiene otros dentro:



```
@override  
Widget build(BuildContext context) {  
  return Container(  
    padding: EdgeInsets.symmetric(horizontal: 15.0, vertical: 20.0),  
    child: Column(  
      children: [  
        _PublicacionItemHeader(publicacion: this.publicacion),  
        SizedBox(height: 10),  
        _PublicacionImagen(publicacion: this.publicacion),  
        SizedBox(height: 0.0),  
        _PublicacionActions(publicacion: this.publicacion),  
        SizedBox(height: 5.0),  
        _PublicacionTitulo(publicacion: this.publicacion),  
        SizedBox(height: 20.0),  
        Divider(thickness: 1.0),  
      ],  
    ), // Column  
  ); // Container  
}
```

Obviamente, para poder construir esta lista tenemos antes que realizar la consulta a nuestra base de datos para poder mostrar las publicaciones:

```
class _PublicacionesList extends StatelessWidget {  
  
    final publicacionesProvider = PublicacionesProvider();  
  
    @override  
    Widget build(BuildContext context) {  
        return FutureBuilder(  
            future: publicacionesProvider.listarPublicaciones(),  
            builder: (BuildContext context, AsyncSnapshot<List<Publicacion>> snapshot) {  
                if(snapshot.hasData){  
                    return ListView.builder(  
                        itemCount: snapshot.data.length,  
                        itemBuilder: (context, index) => _PublicacionItem(publicacion: snapshot.data[index]),  
                    ); // ListView.builder  
                }else{  
                    return Container(  
                        height: 400.0,  
                        child: Center(  
                            child: CircularProgressIndicator(  
                                backgroundColor: myTheme.primaryColor,  
                            ), // CircularProgressIndicator  
                        ), // Center  
                    ); // Container  
                }  
            },  
        ); // FutureBuilder  
    }  
}
```

Para ello nos servimos del FutureBuilder, un widget de flutter que sirve para realizar cargas de datos asíncronas. Le decimos que su fuente va a ser un método de un provider de publicaciones que traerá la lista de estas de la base de datos. Mientras que no haya datos va a mostrar una barra de carga, cuando los haya crearemos un ListView builder, en el que cada ítem será el ítem de publicación que he mostrado antes.

El modelo de publicación es el siguiente:

```
class Publicacion {  
  
    String idPublicacion;  
    String foto;  
    String idUsuario;  
    double latitud;  
    double longitud;  
    int meGustas;  
    String titulo;  
  
    Publicacion({  
        this.idPublicacion,  
        this.foto,  
        this.idUsuario,  
        this.latitud,  
        this.longitud,  
        this.meGustas,  
        this.titulo,  
    });  
  
    factory Publicacion.fromJson(Map<String, dynamic> json) => Publicacion(  
        idPublicacion : json["idPublicacion"],  
        foto : json["foto"],  
        idUsuario : json["idUsuario"],  
        latitud : json["latitud"].toDouble(),  
        longitud : json["longitud"].toDouble(),  
        meGustas : json["meGustas"],  
        titulo : json["titulo"],  
    );  
  
    Map<String, dynamic> toJson() => {  
        "idPublicacion" : idPublicacion,  
        "foto" : foto,  
        "idUsuario" : idUsuario,  
        "latitud" : latitud,  
        "longitud" : longitud,  
    };
```

Este es el método que nos trae las publicaciones:

```
//Listamos las publicaciones que tenemos en Firebase
Future<List<Publicacion>> listarPublicaciones() async {
    //auth es el método de autenticación de firebase, por lo que mandamos el token que firebase
    //nos manda cuando el usuario se loguea
    final url = '${_baseUrl}/publicaciones.json?auth=${_prefs.token}';
    final List<Publicacion> list = List<Publicacion>();
    final resp = await http.get(url);
    final Map <String, dynamic> decodedResp = json.decode(resp.body);

    if(decodedResp == null){
        return [];
    }
    decodedResp.forEach((id, publicacion){
        final Publicacion publi = Publicacion.fromJson(publicacion);
        publi.idPublicacion = id;
        list.add(publi);
    });
    return list;
}
```

Como vemos, es un future que resuelve una lista de publicaciones. Una vez realiza la petición y la tiene, recorre el json (que es un mapa String, Dynamic) y por cada elemento crea una publicación, la clave del elemento será la id de la publicación y el valor será el resto del objeto. Lo añade a una lista por cada iteración y devuelve la lista.

Para realizar el encabezado de la publicación, en el que aparecen tanto la foto del usuario que la ha subido como su nombre, he utilizado otro FutureBuilder, que consulta la id del usuario que ha subido la publicación:

```
@override
Widget build(BuildContext context) {
    final usuariosProvider = UserProvider();
    return FutureBuilder(
        future: usuariosProvider.findUserById(publicacion.idUsuario),
        builder: (BuildContext context, AsyncSnapshot<Usuario> snapshot) {
            if(snapshot.hasData){
                return Container(
                    padding: EdgeInsets.symmetric(horizontal:10.0, vertical: 10.0),
                    child: Row(
                        children: [
                            Container(
                                child: (snapshot.data.fotoUrl == '')
                                    ? CircleAvatar(
                                        backgroundColor: myTheme.buttonColor,
                                        foregroundColor: Colors.white,
                                        maxRadius: 15.0,
                                        child: Text('ID'),
                                    ) // CircleAvatar
                                    : CircleAvatar(
                                        backgroundImage: NetworkImage(snapshot.data.fotoUrl),
                                        maxRadius: 15.0,
                                    ), // CircleAvatar
                            ), // Container
                            SizedBox(width: 10.0),
                            Text(
                                snapshot.data.nombreUsuario,
                                style: TextStyle(fontSize: 18)
                            ), // Text
                        ],
                    ),
                );
            }
        },
    );
}
```

Si el usuario tiene foto la pondrá y si no pondrá una por defecto.

Este es el método que nos trae los datos del usuario en el provider de usuarios:

```
//Buscamos un usuario por su id
Future<Usuario> findUserById(String id) async {
  final url = '${_mainUrl}/usuarios/$id.json?auth=${_prefs.token}';
  final resp = await http.get(url);
  return usuarioFromJson(resp.body);
}
```

En cuanto a la imagen de la publicación, simplemente utilizamos un widget de Flutter que es el FadeInImage, que mientras la imagen está cargando mostrará un gif de carga y cuando cargue mostrará la imagen.

```
class _PublicacionImagen extends StatelessWidget {

  final Publicacion publicacion;

  const _PublicacionImagen({@required this.publicacion});

  @override
  Widget build(BuildContext context) {
    return Container(
      width: double.infinity,
      height: 250.0,
      child: FadeInImage(
        placeholder: AssetImage('assets/img/loading.gif'),
        image: NetworkImage(publicacion.foto),
        fit: BoxFit.cover,
      ), // FadeInImage
    ); // Container
  }
}
```

Para los iconos de me gusta, localización, editar y borrar (sólo se mostrarán en las publicaciones que haya subido el usuario logueado), utilizamos distintos métodos que crearán estos botones. Cada botón tendrá su método que disparará un evento al tocarlo. Como vemos otra vez utilizamos las shared preferences, en las que tenemos guardado en el id del usuario logueado para saber si esa publicación es de ese usuario.

```
    @override
    Widget build(BuildContext context) {
      return Row(
        children: [
          _createLikeButton(),
          _createPositionButton(context),
          Expanded(
            child: Container(),
          ), // Expanded
          //Si la publicación la hemos subido el usuario logueado, se muestran las opciones de
          //editar y eliminar
          (_prefs.idUsuarioLogueado == publicacion.idUsuario)
            ? _createEditButton(context)
            : Container(),
          (_prefs.idUsuarioLogueado == publicacion.idUsuario)
            ? _createDeleteButton(publicacion.idPublicacion, context)
            : Container(),
        ],
      ); // Row
    }
  
```

Finalmente, para el título de la publicación este ha sido el layout escogido:

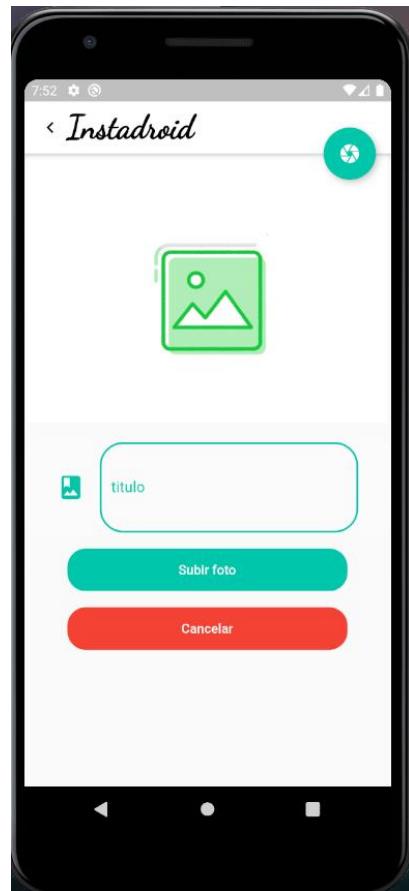
```
class _PublicacionTitulo extends StatelessWidget {

  final Publicacion publicacion;

  const _PublicacionTitulo({@required this.publicacion});
  @override
  Widget build(BuildContext context) {
    return Container(
      padding: EdgeInsets.symmetric(horizontal: 10.0),
      alignment: Alignment.bottomLeft,
      child: Text(
        publicacion.titulo,
        style: TextStyle(
          fontSize: 18
        ), // TextStyle
      ), // Text
    ); // Container
  }
}
```

■ Aspectos relevantes de la creación/edición/borrado de una publicación

Este es el aspecto que muestra la pantalla desde la que se puede subir o editar una publicación subida:



Como podemos ver se trata de un Scaffold, con su appBar, que tiene un FAB desde el que podremos elegir el método de subida de la foto (galería o cámara), que todo su body es una columna en la que encontramos un layout para la foto, un text input para el título. Además, cuenta con dos botones, uno para subir la publicación y otro para cancelar y volver al timeline. Adicionalmente, en el appBar también contamos con una flecha hacia atrás, que nos regresará al timeline.

```
return Scaffold(  
    appBar: AppBar(  
        title: Transform(  
            transform: Matrix4.translationValues(-110.0, 0.0, 0.0),  
            child: AppTitle(  
                color: Colors.black,  
                size: 35.0  
            ), // AppTitle  
        ), // Transform  
        backgroundColor: Colors.white,  
        centerTitle: false,  
        leading: IconButton(  
            icon: Icon(Icons.chevron_left, color: Colors.black),  
            onPressed: () => Navigator.of(context).pop(),  
        ), // IconButton  
    ), // AppBar  
    body: SingleChildScrollView(  
        child: Form(  
            key: formKey,  
            child: Column(  
                mainAxisAlignment: MainAxisAlignment.start,  
                children: [  
                    _createPhotoLayout(context),  
                    SizedBox(height: 15.0),  
                    _createTitleInput(),  
                    SizedBox(height: 10.0),  
                    _createUploadPhotoButton(context),  
                    SizedBox(height: 15.0),  
                    _createCancelButton(context),  
                ],  
            ), // Column  
        ), // Form  
    ), // SingleChildScrollView
```

Para comprender esta pantalla debemos comprender primero que se utiliza con dos fines, tanto como para subir una foto como para editar una foto que ya está subida. Para conseguir hacer esta distinción hago lo siguiente:

Tengo una publicación como atributo de la clase:

```
//La publicación que vamos a ir construyendo para subir  
Publicacion publicacion = Publicacion();
```

Tengo una variable booleana para saber si estamos editando o subiendo una publicación:

```
//Para saber si estamos editando o creando  
bool editando = false;
```

Si vamos a editar, se mandará una publicación como argumento desde la pantalla de timeline (parecido a un Intent de Android nativo). Comprobamos. Si la publicación que nos llega por argumentos no es nula, querrá decir que estamos editando y la publicación, variable de la clase será igual a esa publicación que nos llega.

```
//Si recibimos una publicación como argumento entonces estamos editando una publicación existe  
final Publicacion publicacionArgs = ModalRoute.of(context).settings.arguments;  
//Si no es nula la publicación mandada desde la pantalla anterior, nuestra publicación global  
//porque la estamos editando  
if(publicacionArgs != null){  
    publicacion = publicacionArgs;  
    editando = true;  
}
```

Para mostrar la imagen tenemos que comprobar una serie de cosas. Primero si la foto de la publicación no es nula, mostraremos la foto de la publicación. Si nuestra imagen que es un File (la que subimos desde cámara o galería al momento de echarla) no es nula, la mostramos. Si esto no se cumple tampoco, mostraremos un gif.

```

Widget _createPhotoLayout(BuildContext context) {
    //Consultamos el tamaño de la pantalla
    final screenSize = MediaQuery.of(context).size;
    if(publicacion.foto != null){
        //Mostramos foto del producto que nos viene del timeline
        return Container(
            //Ocupa todo el ancho disponible
            width: double.infinity,
            //Será la mitad de la altura de la pantalla
            height: screenSize.height * 0.5,
            child: FadeInImage(
                image: NetworkImage(publicacion.foto),
                placeholder: AssetImage('assets/img/loading.gif'),
                fit: BoxFit.cover,
            ), // FadeInImage
        ); // Container
    }else if(imagen != null){
        return Container(
            width: double.infinity,
            height: screenSize.height * 0.5,
            child: Image.file(
                imagen,
                fit: BoxFit.cover,
            ), // Image.file
        ); // Container
    }else{
        return Container(
            width: double.infinity,
            height: screenSize.height * 0.38,
            child: Image(
                image: AssetImage('assets/img/photoupload.gif'),
            ), // Image
        ); // Container
    }
}

```

El resto de la pantalla es un formulario como los que ya hemos visto para el login y el registro, a la hora de enviar el formulario, se validará que este es correcto y que tiene imagen.

Para subir la publicación, primero subimos la imagen al Storage de Firebase, a través del siguiente provider de subir fotos que he creado:

```

//Subimos imagen al storage de firebase, el método devuelve la url de la foto subida

Future<String> uploadImageToFirebase(File image, String publiTitle) async {
    final collection = 'images';
    await Firebase.initializeApp();
    final storage = FirebaseStorage.instance;
    var snapshot = await storage.ref()
        .child('$collection/$publiTitle')
        .putFile(image);
    var url = await snapshot.ref.getDownloadURL();
    print(url.toString());
    return url.toString();
}

```

Una vez tenemos la url, lo siguiente que haremos será detectar la localización del dispositivo, en otro provider que he creado para obtener la localización:

```

Future<Position> getCurrentLocation() async{
    Position position = await Geolocator.getCurrentPosition(desiredAccuracy: LocationAccuracy.high);
    print('Localización: $position');
    return position;
}

```

Del objeto position que devuelve, sacaremos la latitud y longitud.

Con todos los datos, comprobamos si estamos editando o no y realizamos la subida/edición a Firebase:

```

if(editando){
    bool publiEditada = await publicacionesProvider.editarPublicacion(publicacion);
} else{
    bool publiSubida = await publicacionesProvider.insertarPublicacion(publicacion);
}

```

Los métodos para crear y editar una publicación son:

```

//Insertamos publicación
Future<bool> insertarPublicacion(Publicacion publicacion) async {
    final url = '${_baseUrl}/publicaciones.json?auth=${_prefs.token}';
    final resp = await http.post(url, body: publicacionToJson(publicacion));
    final decodedResp = json.decode(resp.body);
    print(decodedResp);
    return true;
}

//Editamos publicación
Future<bool> editarPublicacion(Publicacion publicacion) async {
    final url = '${_baseUrl}/publicaciones/${publicacion.idPublicacion}.json?auth=${_prefs.token}';
    final resp = await http.put(url, body: publicacionToJson(publicacion));
    final decodedResp = json.decode(resp.body);
    return true;
}

```

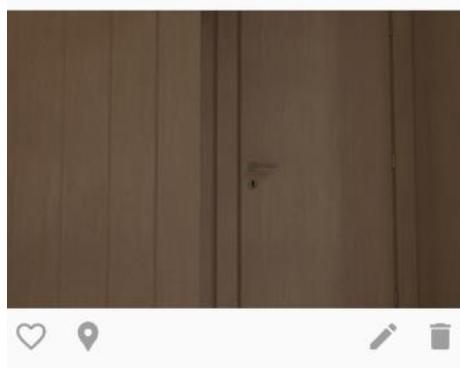
Para crear una publicación podemos tocar en el FAB que se encuentra en el menú de navegación:

```

floatingActionButton: FloatingActionButton(
    child: Icon(Icons.camera_alt, color: Colors.white),
    backgroundColor: myTheme.buttonColor,
    onPressed: (){
        //Navegación a la página de subir foto
        Navigator.pushNamed(context, 'foto');
    },
), // FloatingActionButton

```

Para editarla, desde el timeline, si hemos subido la publicación nos aparecerá el ícono para hacerle click y acceder a la pantalla. Igualmente nos aparecerá la opción de borrar.



```

Widget _createEditButton(BuildContext context) {
  return IconButton(
    icon: Icon(Icons.edit, color: Colors.grey),
    onPressed: (){
      //Editamos la publicación, vamos a la pantalla de subir foto pero en modo edición
      Navigator.pushNamed(context, 'foto', arguments: publicacion);
    },
    iconSize: 30,
  ); // IconButton
}

```

Como se puede ver y ya mencioné antes, aquí se le pasa como argumento la publicación.

Para borrar un producto, haremos click en el ícono de la papelera y se nos mostrará un mensaje de confirmación antes de continuar. Una vez aceptemos la publicación se borrará de Firebase.

```

Widget _createDeleteButton(String idPublicacion, BuildContext context) {
  return IconButton(
    icon: Icon(Icons.delete, color: Colors.grey),
    onPressed: () {
      utils.showAlert(
        context,
        [
          FlatButton(
            child: Text('Aceptar'),
            onPressed: () async{
              //Eliminamos la publicación
              final publicacionesProvider = PublicacionesProvider();
              bool publiEliminada = await publicacionesProvider.borrarPublicacion(idPublicacion);
              if(publiEliminada){
                Navigator.of(context).pop();
                Navigator.pushNamed(context, 'home');
              }
            }
          ), // FlatButton
          FlatButton(
            child: Text('Cancelar'),
            onPressed: () => Navigator.of(context).pop(),
          ) // FlatButton
        ],
        'Borrar publicación',
        '¿Seguro/a de borrar la publicación?',
      );
    },
    iconSize: 30,
  ); // IconButton
}

```

En el provider de publicaciones, este es el método para eliminar la publicación:

```

//Borramos una publicación de Firebase
Future<bool> borrarPublicacion (String idPublicacion) async {
  final url = '${_baseUrl}/publicaciones/${idPublicacion}.json?auth=${_prefs.token}';
  await http.delete(url);
  return true;
}

```

■ Aspectos relevantes de ver la localización de una publicación

Para mostrar la localización de una publicación, hacemos click en el botón de localización que hay en cada publicación:

```
Widget _createPositionButton(BuildContext context) {
    return IconButton(
        icon: Icon(Icons.location_pin, color: Colors.grey),
        onPressed: () {
            //Navegación a la pantalla de localización
            Navigator.pushNamed(context, 'localizacion', arguments: publicacion);
        },
        iconSize: 30,
    ); // IconButton
}
```

Este le pasará a la página de localización la publicación en concreto para que pueda mostrar su localización.

En la página de localización mostramos el mapa de Google (se ha tenido que modificar el manifest de Android, para incluir el api key):

```
//Recibimos la publicación de la página anterior
final Publicacion publicacion = ModalRoute.of(context).settings.arguments;
//Posición de la cámara
final CameraPosition posicionInicial = CameraPosition(
    target: LatLng(publicacion.latitud, publicacion.longitud),
    zoom: 17.5,
    tilt: 50,
); // CameraPosition
//Creación de marcadores
Set<Marker> markers = new Set<Marker>();
markers.add(Marker(
    markerId: MarkerId('geo-location'),
    position: LatLng(publicacion.latitud, publicacion.longitud),
    infoWindow: InfoWindow(
        title: publicacion.titulo
    ), // InfoWindow
)); // Marker
```

Obtenemos la publicación que se nos pasa desde el timeline, fijamos la posición de la cámara y añadimos un marcador, que se pondrá en esa posición en el mapa y si le hacemos click, mostrará el título de la publicación.

Para mostrar el mapa, he utilizado en Scaffold, en el que muestro el appBar y en el body muestro el mapa de Google:

```

return Scaffold(
  appBar: AppBar(
    title: Transform(
      transform: Matrix4.translationValues(-100.0, 0.0, 0.0),
      child: AppTitle(
        color: Colors.black,
        size: 35.0
      ), // AppTitle
    ), // Transform
    backgroundColor: Colors.white,
    centerTitle: false,
    leading: IconButton(
      icon: Icon(Icons chevron_left, color: Colors.black),
      onPressed: () => Navigator.of(context).pop(),
    ), // IconButton
  ), // AppBar
  body: GoogleMap(
    mapType: mapType,
    markers: markers,
    initialCameraPosition: posicionInicial,
    onMapCreated: (GoogleMapController controller) {
      _controller.complete(controller);
    },
  ), // GoogleMap
); // Scaffold

```

El controlador del mapa y el tipo de mapa se definen como variables de clase:

```

Completer<GoogleMapController> _controller = Completer();
MapType mapType = MapType.normal;

```

■ Aspectos relevantes de la visualización de la localización de todas las publicaciones

Se trata de una pantalla muy similar a la anterior, pero esta nos mostrará la localización de todas las fotos que haya guardadas en Firebase. En este caso, accederemos a esta pantalla desde el navbar general de nuestra aplicación:

Instadroid



Pulsando ese ícono, haremos una llamada a nuestro provider para que nos liste las publicaciones y pase esa lista como argumento a la pantalla en la que veremos todas las localizaciones:

```

actions: [
    IconButton(
        icon: Icon(Icons.map, color: Colors.black),
        onPressed: () async {
            final publicacionesProvider = PublicacionesProvider();
            final List<Publicacion> listaPublis = await publicacionesProvider.listarPublicaciones();
            Navigator.pushNamed(context, 'mapa', arguments: listaPublis);
        },
    ), // IconButton
],

```

Ya en la pantalla de visualizar la localización, recibimos la lista, y fijamos la posición de la cámara en la localización de la primera publicación. Además creamos un set de marcadores y para rellenarlo, recorremos nuestra lista y vamos creando un marker por cada publicación:

```

final List<Publicacion> listPublicaciones = ModalRoute.of(context).settings.arguments;

final CameraPosition posicionInicial = CameraPosition(
target: LatLng(listPublicaciones[0].latitud, listPublicaciones[0].longitud),
zoom: 17.5,
tilt: 50,
); // CameraPosition

//Creación de marcadores
Set<Marker> markers = new Set<Marker>();

listPublicaciones.forEach((publicacion) {
    markers.add(Marker(
        markerId: MarkerId('geo-location${publicacion.titulo}'),
        position: LatLng(publicacion.latitud, publicacion.longitud),
        infoWindow: InfoWindow(
            title: publicacion.titulo
        ), // InfoWindow
    )); // Marker
});

```

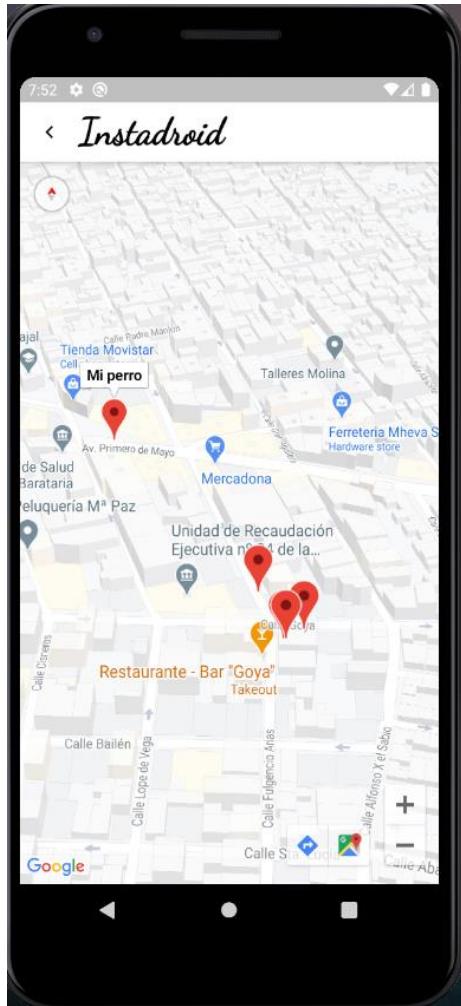
Finalmente, de igual manera que antes, pintaremos el mapa:

```

return Scaffold(
    appBar: AppBar(
        title: Transform(
            transform: Matrix4.translationValues(-100.0, 0.0, 0.0),
            child: AppTitle(
                color: Colors.black,
                size: 35.0,
            ), // AppTitle
        ), // Transform
        backgroundColor: Colors.white,
        centerTitle: false,
        leading: IconButton(
            icon: Icon(Icons chevron_left, color: Colors.black),
            onPressed: () => Navigator.of(context).pop(),
        ), // IconButton
    ), // AppBar
    body: GoogleMap(
        mapType: mapType,
        markers: markers,
        initialCameraPosition: posicionInicial,
        onMapCreated: (GoogleMapController controller) {
            _controller.complete(controller);
        },
    ), // GoogleMap
); // Scaffold

```

Así se visualizan las localizaciones en nuestra aplicación:



■ Funcionalidad extra: Dar «me gusta» a una publicación

Se trata de pulsar en el ícono del corazón y que este cambie de estado y se vea de color rojo. Además, he guardado las publicaciones que le gustan a el usuario en los shared prefs, de manera que se recordarán a qué aplicaciones se ha dado me gusta.

La lógica del botón es sencilla, utilizamos un stateful widget, de manera que cuando haya un cambio del estado el widget se redibujará en consecuencia:

```
return IconButton(
  icon: (_isLiked)
    ? Icon(Icons.favorite)
    : Icon(Icons.favorite_border),
  onPressed: () {
    setState(() {
      _isLiked = !_isLiked;
    });
    if(_isLiked){
      userPreferences.publicacionesGustadas.add(publicacion.idPublicacion);
      userPreferences.publicacionesGuardadas = true;
    }else{
      userPreferences.publicacionesGustadas.remove(publicacion.idPublicacion);
    }
  },
  color: (_isLiked)
    ? Colors.red
    : Colors.grey,
  iconSize: 30,
```

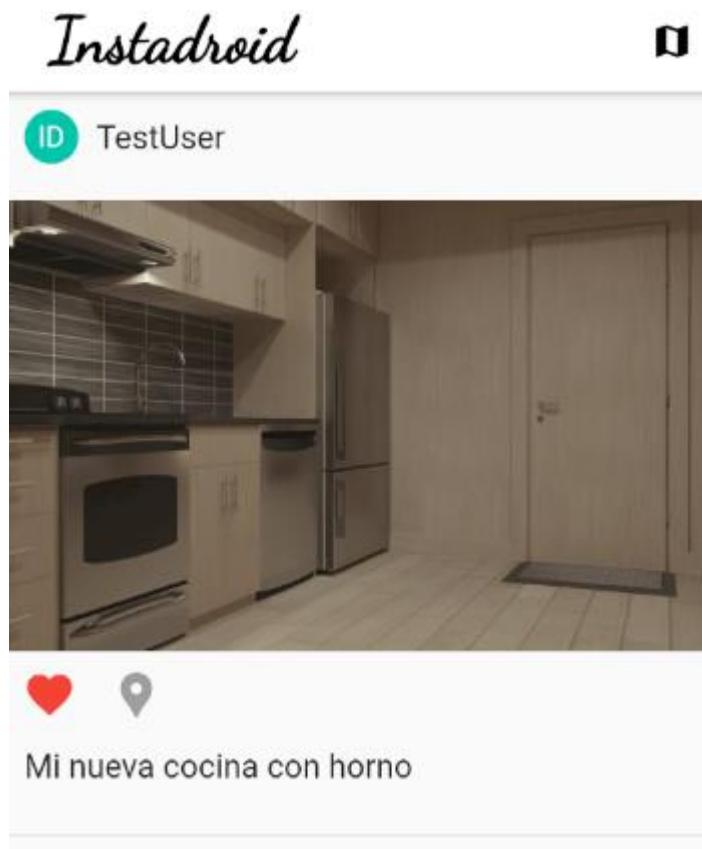
En las shared prefs tenemos una lista de likes que va dando el usuario y que se van añadiendo o quitando de la lista si le quita el like.

```
List<String> get publicacionesGustadas => this._publisMeGusta;

set publicacionesGustadasGuardadas(bool setear){
  if(setear){
    List<String> gustadas = _prefs.getStringList('megustas') ?? List<String>();
    gustadas.addAll(this._publisMeGusta);
    _prefs.setStringList('megustas', gustadas);
  }else{
    //Cambiamos de usuario por lo que borramos sus me gusta de los shared prefs
    _prefs.setStringList('megustas', new List<String>());
  }
}

List<String> get publicacionesMeGustaGuardadas => _prefs.getStringList('megustas') ?? List<String>()
```

Cuando queremos guardarlas simplemente en el setter de las publicaciones guardadas le pasamos true y añadirá la lista como variable de clase a la lista ya existente en las shared prefs. De esa manera lograremos una persistencia de las publicaciones gustadas.



TESTEO Y PRUEBAS DE LA SOLUCIÓN

1. Plan de pruebas

Es el producto formal que define los objetivos de la prueba de un sistema, establece y coordina una estrategia de trabajo, y provee del marco adecuado para elaborar una planificación paso a paso de las actividades de prueba⁷⁷.

Se plantean los siguientes niveles de prueba:

■ Pruebas unitarias⁷⁸

Tienen como objetivo verificar la funcionalidad y estructura de cada componente individualmente una vez ha sido programado.

Constituyen la prueba inicial del sistema, apoyándose las demás pruebas en ellas.

Hablamos de dos enfoques principales:

- Estructural o de caja blanca: Se verifica la estructura interna del componente con independencia de la funcionalidad que debería tener el mismo.
- Funcional o de caja negra: Se comprueba que los componentes del sistema funcionen de forma adecuada. Se realiza un análisis de las entradas y salidas y se verifica la adecuación del resultado a lo esperado. Se consideran entradas y salidas sin preocuparse por la estructura interna.

Los pasos necesarios para llevar a cabo pruebas unitarias son:

- Ejecutar los casos de uso, esperando que los casos de prueba contemplen todas las condiciones válidas y esperadas como las inválidas e inesperadas.
- Corregir los errores o defectos encontrados y repetir las pruebas que los detectaron.

La prueba unitaria está finalizada cuando se cumplan todas las verificaciones y no se encuentre ningún defecto.

■ Pruebas de integración⁷⁹

⁷⁷ <https://manuel.cillero.es/doc/metodologia/metrica-3/procesos-principales/asi/actividad-10/>

⁷⁸ <https://manuel.cillero.es/doc/metodologia/metrica-3/tecnicas/pruebas/unitarias>

⁷⁹ <https://manuel.cillero.es/doc/metodologia/metrica-3/tecnicas/pruebas/integracion/>

Verifican el correcto ensamblaje entre los distintos componentes una vez se han probado unitariamente.

Hay dos tipos de integración:

- Integración incremental: Se combina el siguiente componente a probar con los otros componentes que ya están probados y así se va incrementando el número de componentes sobre los que realizar pruebas. De esta manera, cuando una de la integración de componentes falle, sabremos que son los mecanismos de comunicación entre este componente y los demás lo que está fallando.

Dentro de esta integración incremental hay tres tipos de estrategias de integración:

- ➔ De arriba abajo (top-down): El primer componente que se desarrolla y prueba es el primero de la jerarquía. La ventaja de elegir esta estrategia es que los componentes se prueban en etapas tempranas y con frecuencia.
 - ➔ De abajo arriba (bottom-up): Se crean los componentes de más bajo nivel primero y a continuación los de más último nivel. Con esta estrategia tendremos menos dificultades a la hora de planificar y gestionar.
 - ➔ Estrategias combinadas: Se aplican las anteriores de forma conjunta.
- Integración no incremental: Se prueba cada componente por separado y posteriormente se integran todos cuando se han realizado sobre cada uno las pruebas necesarias.

■ Pruebas del sistema⁸⁰

Su objetivo es ejercitar el sistema comprobando la integración de este de forma global, comprobando si funciona correctamente el sistema y la comunicación de este con otros sistemas necesarios para su funcionamiento.

Como pruebas del sistema podemos encontrar:

- Pruebas funcionales: Comprobando que el sistema cumple los requisitos establecidos
- Pruebas de comunicaciones: Se aseguran de que los componentes y su comunicación funcionan correctamente.

⁸⁰ <https://manuel.cillero.es/doc/metodologia/metrica-3/tecnicas/pruebas/sistema/>

- Pruebas de rendimiento: Determinan los tiempos de respuesta y si están dentro de las especificaciones.
- Pruebas de volumen: Examinan el funcionamiento del sistema cuando está trabajando con grandes volúmenes de datos.
- Pruebas de sobrecarga: Comprueban el funcionamiento del sistema rozando el límite de sus recursos. Lo que se busca es saber el punto a partir del cual el sistema comienza a trabajar por debajo de los requisitos establecidos.
- Pruebas de facilidad de uso: Comprueban que el sistema se adapta a las necesidades de los usuarios (si se acomoda a su modo habitual de trabajar, si es fácil para ellos introducir datos en el sistema y obtener resultados)
- Pruebas de seguridad: Verifican los mecanismos de control de acceso al sistema para evitar alteraciones indebidas.

■ **Pruebas de implantación⁸¹**

Comprueban el correcto funcionamiento del sistema integrado en el entorno de producción.

■ **Pruebas de aceptación⁸²**

Validan que el sistema cumple con el funcionamiento esperado. Son definidas por el usuario y preparadas por el equipo de desarrollo. Su aceptación final está en manos del usuario.

■ **Pruebas de regresión⁸³**

Consisten en comprobar si los cambios sobre un componente introducen un comportamiento no deseado o errores adicionales en otros componentes.

1.1. Plan de pruebas Instadroid

El plan de pruebas de nuestra aplicación va a consistir en los siguientes grupos de pruebas:

⁸¹ <https://manuel.cillero.es/doc/metodologia/metrica-3/tecnicas/pruebas/implantacion/>

⁸² <https://manuel.cillero.es/doc/metodologia/metrica-3/tecnicas/pruebas/aceptacion/>

⁸³ <https://manuel.cillero.es/doc/metodologia/metrica-3/tecnicas/pruebas/regresion/>

■ Pruebas unitarias

- *Caso de prueba 1 (Caso de Uso registro):* Registro de usuario (alta en servicio de autenticación de Firebase y alta en la colección de usuarios de realtime database de Firebase). Se comprobará también que el sistema no nos deja registrar a dos usuarios con el mismo correo electrónico. Finalmente se comprobará la correcta cumplimentación del formulario.
- *Caso de prueba 2 (Caso de Uso login):* Login del usuario registrado (comprobamos que el usuario se loguea correctamente y que si no se introducen credenciales válidas no deja hacer login). Se comprobará la correcta cumplimentación del formulario.
- *Caso de prueba 3 (Caso de Uso ver timeline):* Listado correcto de las publicaciones presentes en realtime database.
- *Caso de prueba 4 (Caso de Uso subir fotos):* Creación de una publicación. Se comprobará que se sube correctamente tanto la imagen (a Firebase Storage) como el resto de los datos de la publicación a realtime database. También verificaré que se añade al timeline de publicaciones con las opciones de eliminar y editar. Igualmente se comprobará que la aplicación no dejará subir una publicación sin título y sin foto.
- *Caso de prueba 5 (Caso de Uso editar fotos):* Edición de la publicación. Se comprobará que la publicación se actualiza correctamente en realtime database cambiando todos sus datos.
- *Caso de prueba 6 (Caso de Uso ver localización foto):* Visualización de la localización. Se comprobará que la ubicación en la que se realiza la publicación es correcta.
- *Caso de prueba 7 (Caso de Uso ver localización de todas las fotos):* Visualización de la localización de la publicación en el mapa de todas las localizaciones. Se comprobará que se añade la publicación en esta vista.
- *Caso de prueba 8 (Caso de Uso eliminar publicación):* Eliminación de la publicación: Se comprobará que la aplicación nos emite un mensaje de confirmación y una vez aceptemos, se verificará que la publicación se elimina de realtime database.

■ Pruebas del sistema

➔ Prueba de facilidad de uso

Caso de prueba 9: Se comprobará si la aplicación es intuitiva y fácil de usar con un usuario no especializado

➔ Prueba de seguridad

Caso de prueba 10: Se comprobará si un usuario no registrado puede acceder a la aplicación.

Caso de prueba 11: Se comprobará si quitando el token de autorización a la hora de realizar peticiones a Firebase obtenemos resultados en nuestras consultas al backend.

1.2. Realización de pruebas

Caso de prueba 1 (Caso de Uso registro)

Registraremos a un usuario.

En primer lugar, vamos a introducir un correo no válido y una contraseña no válida.

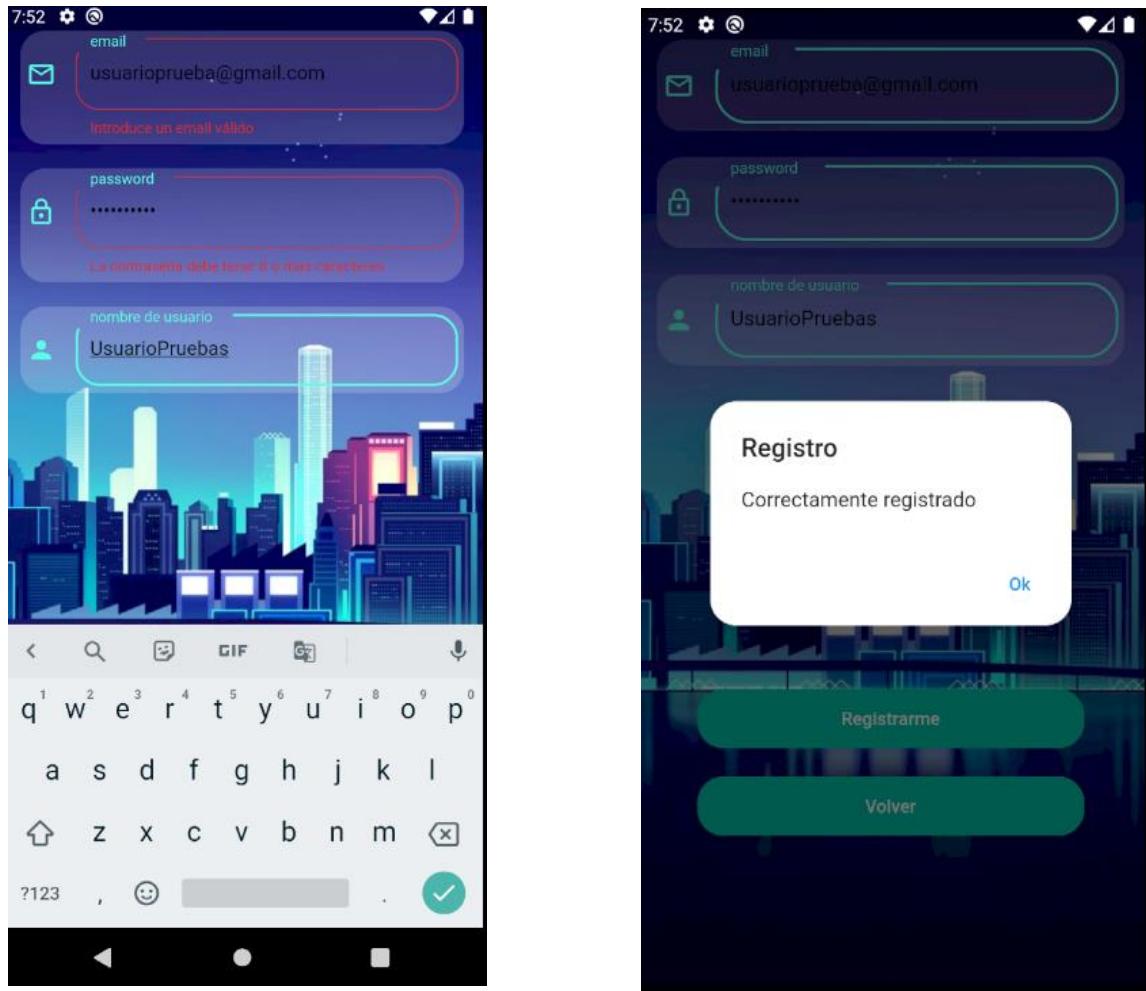
Esperamos ➔ Que la interfaz muestre que el formulario no se ha cumplimentado correctamente.

Resultado ➔ Ok



A continuación, registramos un usuario de forma correcta y vamos a ver si se ha guardado tanto en el servicio de autenticación como en nuestra colección de usuarios de realtime database:

Resultado → Ok



Se agrega al servicio de autenticación y a realtime database:

Firebase

Instadroid ▾ Authentication Ir a la documentación

Compilación

- Authentication
- Cloud Firestore
- Realtime Database
- Storage
- Hosting
- Functions
- Machine Learning

Lanzamiento y monitorización

Crashlytics, Performance, Test L...

Analytics

Dashboard, Events, Conversions...

Interacción

- Extensions

Spark Gratis 0 USD/mes Actualizar

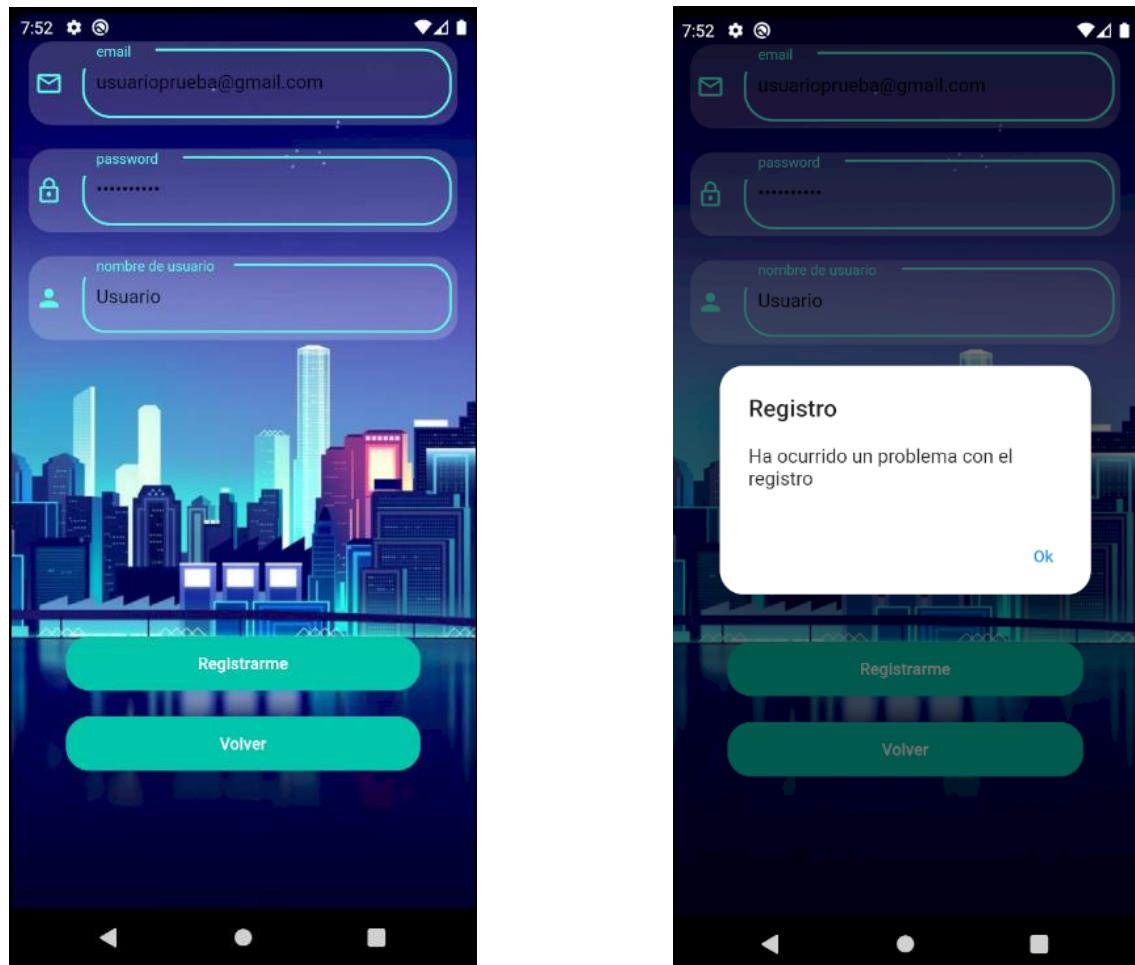
Crea prototipos y pruébalos por completo con la Suite del Emulador Local, que ahora también incluye Firebase Authentication. [Empezar](#)

Identificador	Proveedores	Fecha de creación	Inicio de sesión	UID de usuario ↑
adrian@gmail.com	✉	3ene.2021	3ene.2021	8rWny6AnBsXM1RxG6pSjDZerCxP2
prueba@prueba.com	✉	3ene.2021	3ene.2021	Bwpimu5imnT9MiXE3l7gLD9YES03
test2@test.com	✉	30dic.2020	30dic.2020	Bz0AuqxGUWfpWdxuT2fPnPwkatf1
aitor@gmail.com	✉	26dic.2020	26dic.2020	C4Ktr71TCIgwh79on7Gx1lQeMC13
megustas@gmail.com	✉	4ene.2021	4ene.2021	DInArNIVauQfl4avcnkLkpuiV2
cgonzalezd10@gmail.com	✉	26dic.2020	3ene.2021	PUbTK2oFmedVHycsmL20n6dBm...
test@test.com	✉	26dic.2020	3ene.2021	Z5agzu01PPUwrrB2JY2AbJnCtQ2
josecar@gmail.com	✉	3ene.2021	3ene.2021	beqBN3ivT5YmUlnmDXo6omFTA...
pilar@gmail.com	✉	3ene.2021	3ene.2021	dj6816dixaMPpRqODnxnf4y6KAm2
usuario@prueba@gmail.com	✉	4ene.2021	4ene.2021	hl3cryDbZ3Um8r0P0Mbndu4leJ2

The screenshot shows the Firebase Realtime Database console. On the left, there's a sidebar with various services like Authentication, Cloud Firestore, and Storage. The main area is titled 'Realtime Database' with tabs for 'Datos', 'Reglas', 'Copias de seguridad', and 'Uso'. A note at the top says 'Crea prototipos y pruébalo por completo con la Suite del Emulador Local, que ahora también incluye Firebase Authentication.' and a 'Empezar' button. The database structure under 'usuarios' is shown as a tree view with many nodes, one of which is highlighted in yellow and has the email 'usuarioprueba@gmail.com' displayed below it.

Finalmente, vamos a comprobar que no nos deja registrar a un usuario ya existente:

Resultado → Ok



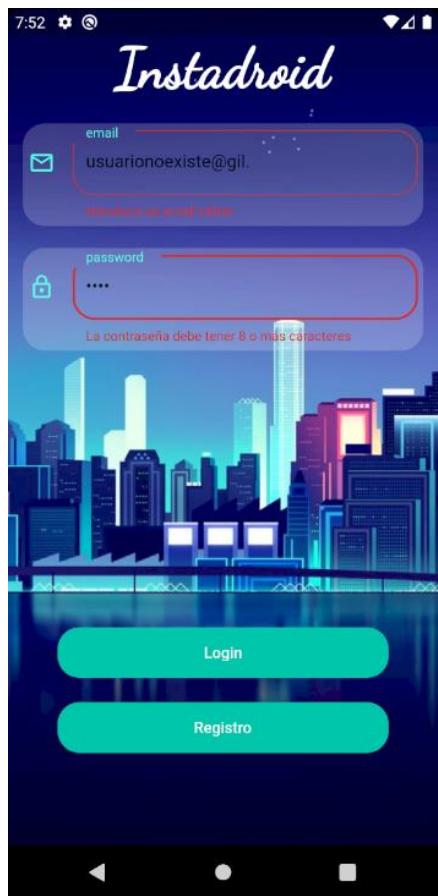
Caso de prueba 2 (Caso de Uso login)

En primer lugar, vamos a probar a hacer login con un usuario que no existe en Firebase:

Resultado: Ok

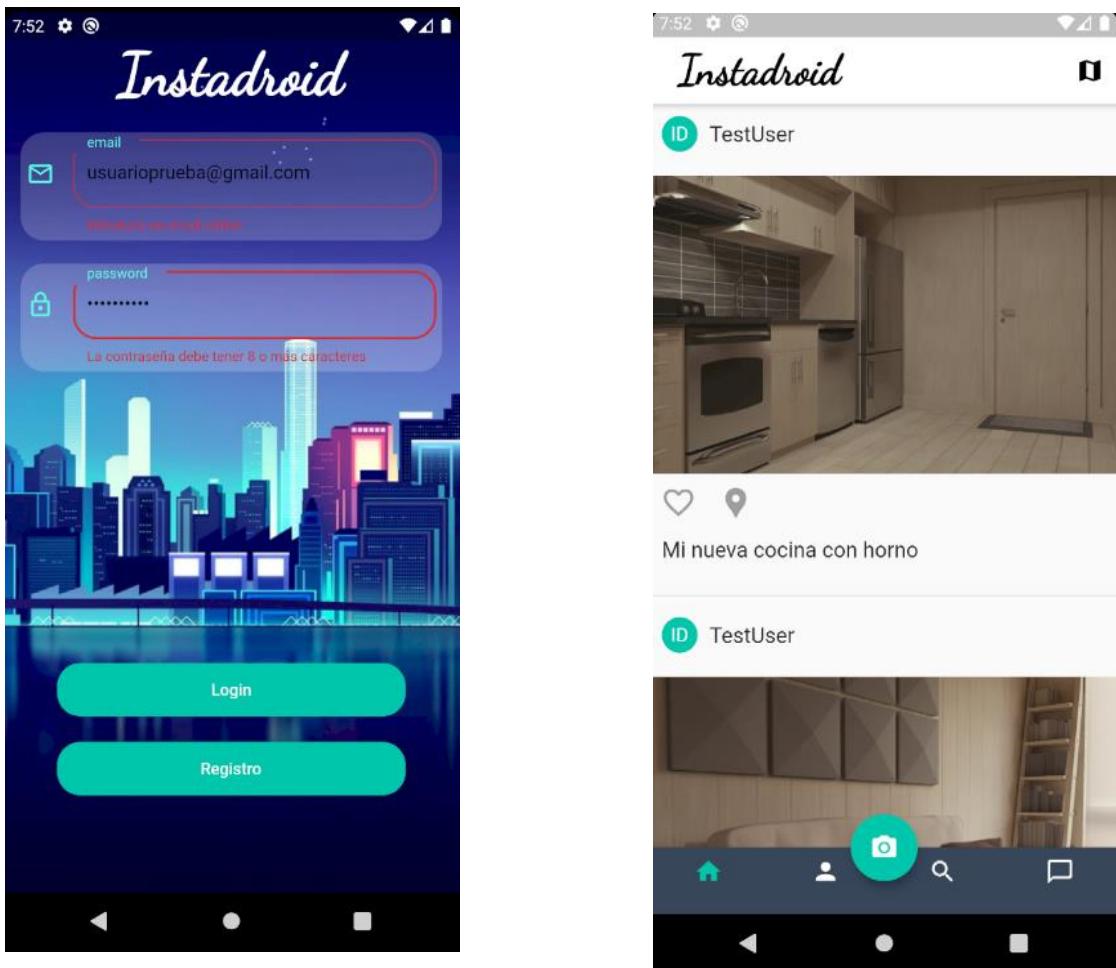
A continuación, vamos a llenar el formulario de login de forma incorrecta para ver si la aplicación nos manda feedback sobre los errores de nuestro formulario:

Resultado: Ok



Finalmente, vamos a hacer login con el usuario que acabamos de registrar:

Resultado: Ok → Entramos en la aplicación



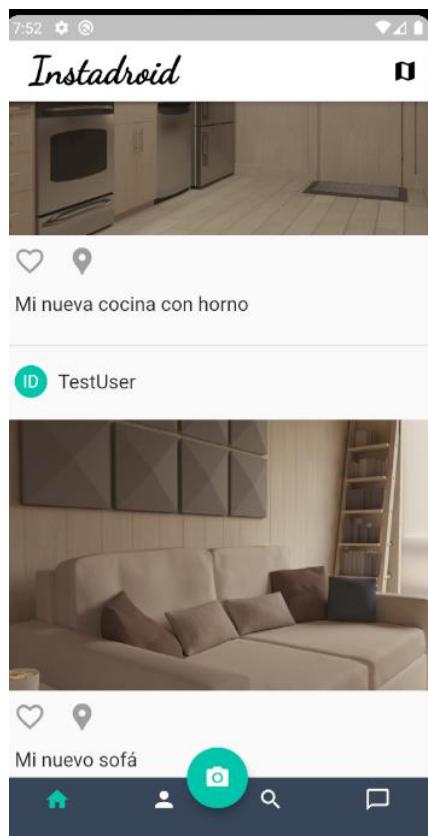
Caso de prueba 3 (Caso de Uso ver timeline)

Comprobamos que las publicaciones que hay en realtime database se nos muestran en forma de lista en nuestra aplicación Instadroid:

Resultado: Ok → Las publicaciones que están en la base de datos se muestran en nuestra aplicación (fijarse en los títulos):

```

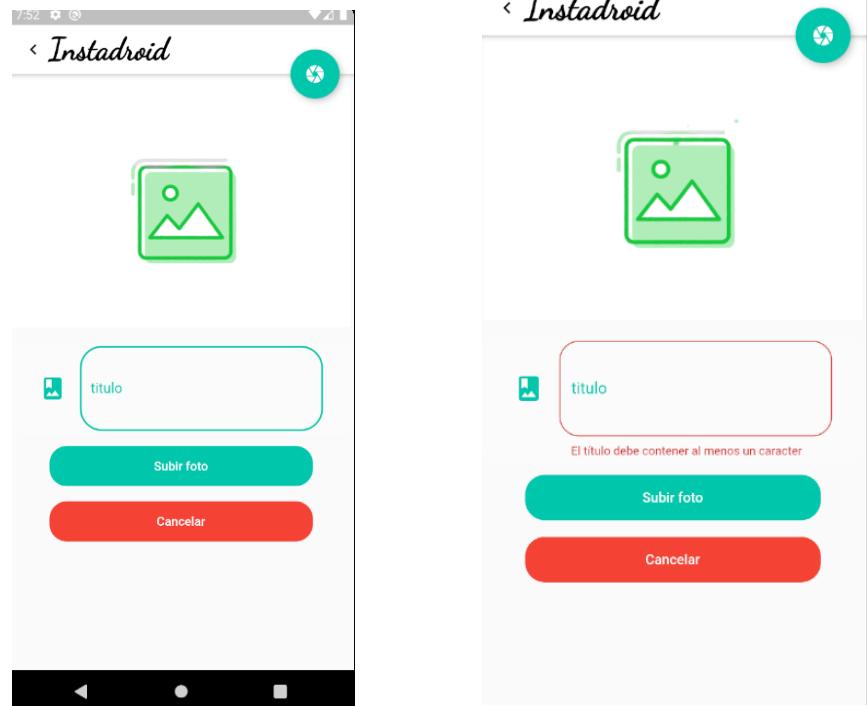
https://instadroid-45787-default-rtdb.firebaseio.com/publicaciones.json
+-- -MQ74u8Sw5aGf7IX-gde
    |   +-- foto: "https://firebasestorage.googleapis.com/v0/b/instadroid-45787.appspot.com/o/publicaciones%2F-MQ74u8Sw5aGf7IX-gde.jpg?alt=media&token=... "
    |   +-- idPublicacion: "-MQ74u8Sw5aGf7IX-gde"
    |   +-- idUsuario: "-MPTeWKXspU-TdFjivnF"
    |   +-- latitud: 38.685125
    |   +-- longitud: -4.1026833
    |   +-- meGustas: 0
    |   +-- titulo: "Mi nueva cocina con horno"
+-- -MO7qmgeZCtvFTSaA5Q
    |   +-- foto: "https://firebasestorage.googleapis.com/v0/b/instadroid-45787.appspot.com/o/publicaciones%2F-MO7qmgeZCtvFTSaA5Q.jpg?alt=media&token=... "
    |   +-- idPublicacion: "-MO7qmgeZCtvFTSaA5Q"
    |   +-- idUsuario: "-MPTeWKXspU-TdFjivnF"
    |   +-- latitud: 38.685125
    |   +-- longitud: -4.1026833
    |   +-- meGustas: 0
    |   +-- titulo: "Mi nuevo sofá"
  
```



Caso de prueba 4 (Caso de Uso subir fotos)

En primer lugar, comprobamos que la interfaz nos advierte de que no podemos subir una foto sin título ni sin imagen:

Resultado → Ok



Como se puede ver, nos dice que debe tener al menos un carácter, y si probamos a subirlo con título, pero sin foto el resultado es:

A continuación, subimos la publicación y debemos comprobar que se añade tanto a realtime database (colección de publicaciones), como la foto se sube como un fichero al servicio de storage de firebase. Además, comprobaremos que se añade en el timeline de publicaciones de Instadroid y si aparece con las opciones de borrar y editar publicación (ya que la hemos subido nosotros, el usuario logueado)

Resultado: Ok → La foto se sube al servicio de storage, la publicación (con la url a la foto en el storage) se sube igualmente y aparece en el timeline de Instadroid como una nueva publicación.

En este caso he echado una foto con la cámara del emulador (pero también tenemos la opción de escogerla desde la galería):

Subimos la foto.



La publicación se ha subido a Firebase realtime database:



The screenshot shows the Realtime Database interface. At the top, there are tabs for Datos, Reglas, Copias de seguridad, and Uso. Below the tabs, a message encourages prototyping and testing with the Local Emulator Suite. The main area displays a list of post keys under a specific path. Each key has a detailed view showing the following fields:

- foto: "https://firebasestorage.googleapis.com/v0/b/instadroid-45787.appspot.com/o/images%2FPublicación%20de%20prueba.jpg?alt=media&token=..."
- idUser: "-MQDF73V6u2vrII1MNRV"
- latitud: 37.4219983
- longitud: -122.084
- meGustas: 0
- título: "Publicación de prueba"

La foto se ha subido Firebase Storage:

The screenshot shows the Storage interface. On the left, the navigation bar includes Información general, Compilación, Lanzamiento y monitorización, Analytics, Interacción, and Spark. The Storage section is selected. The main area shows a list of files under the path gs://instadroid-45787.appspot.com/images. One file, "Publicación de prueba", is highlighted with a blue arrow pointing to its preview thumbnail. A detailed view panel on the right provides the following information for this file:

- Nombre: Publicación de prueba
- Tamaño: 194.079 bytes
- Tipo: image/jpeg
- Fecha y hora de creación: 5 ene 2021 8:56:27
- Actualizada: 5 ene 2021 8:56:27

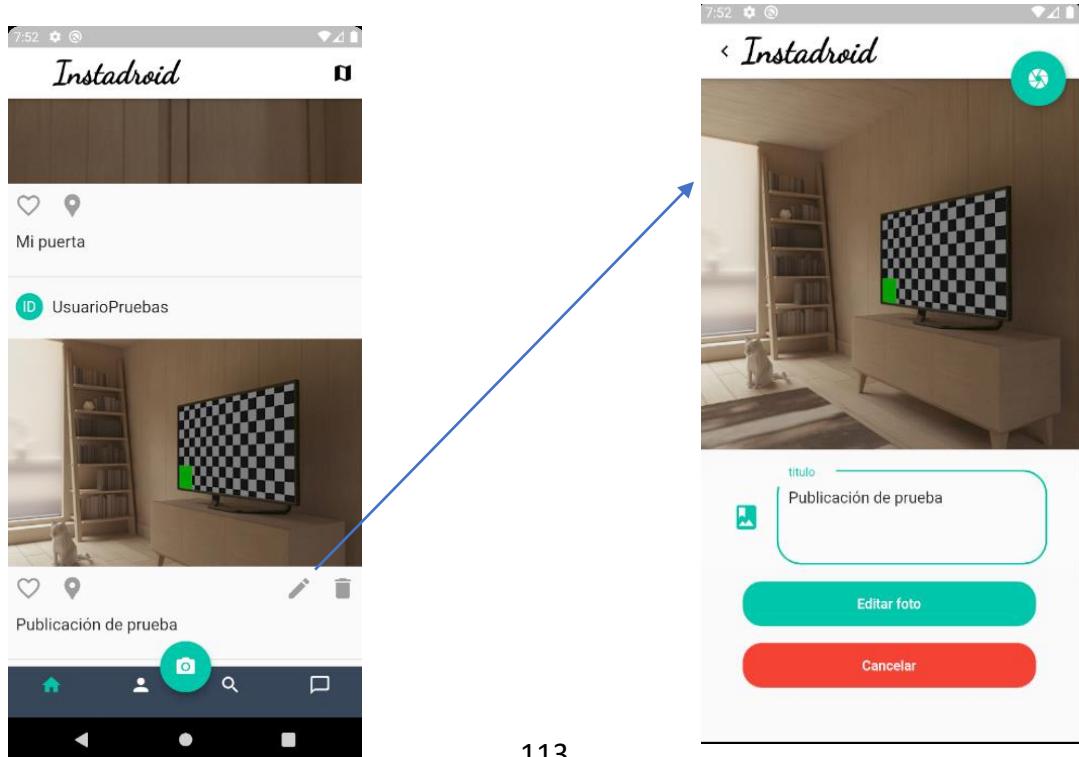
Finalmente, la foto aparece en el timeline de Instadroid, con las opciones de editar y eliminar:



Caso de prueba 5 (Caso de Uso editar fotos):

Comprobamos que al editar la publicación esta se actualiza correctamente en realtime database y en el timeline:

Resultado: Ok → La publicación se actualiza y aparece actualizada





Se actualiza en realtime database (si se comparan las ids con la publicación subida son los mismos)

ID	Título	Foto URL	ID Publicación	ID Usuario	Latitud	Longitud	Me Gustas
-MQ8LATHmDwnvKAIRGqi	Edición de publicación de prueba	https://firebasestorage.googleapis.com/v0/b/instadroid-45787.appspot.com/.../image.jpg	-MQGY0vtVZNrs1rq49d9	-MQDF73V6u2vrII1MNRV	37.4219983	-122.084	0
-MQ8T58fY2cvNY_buhHg							
-MQ8ZjewDusV2ln7ench							
-MQ8_7kv-tOuayD-c3Ci							
-MQ8gHq0mdGJ4SRKLav8							
-MQ8gqePPt_XoubiMkew							
-MQ8o2MGBjBdrqkvdHhT							
-MQGY0vtVZNrs1rq49d9							

La publicación aparece actualizada en el timeline:

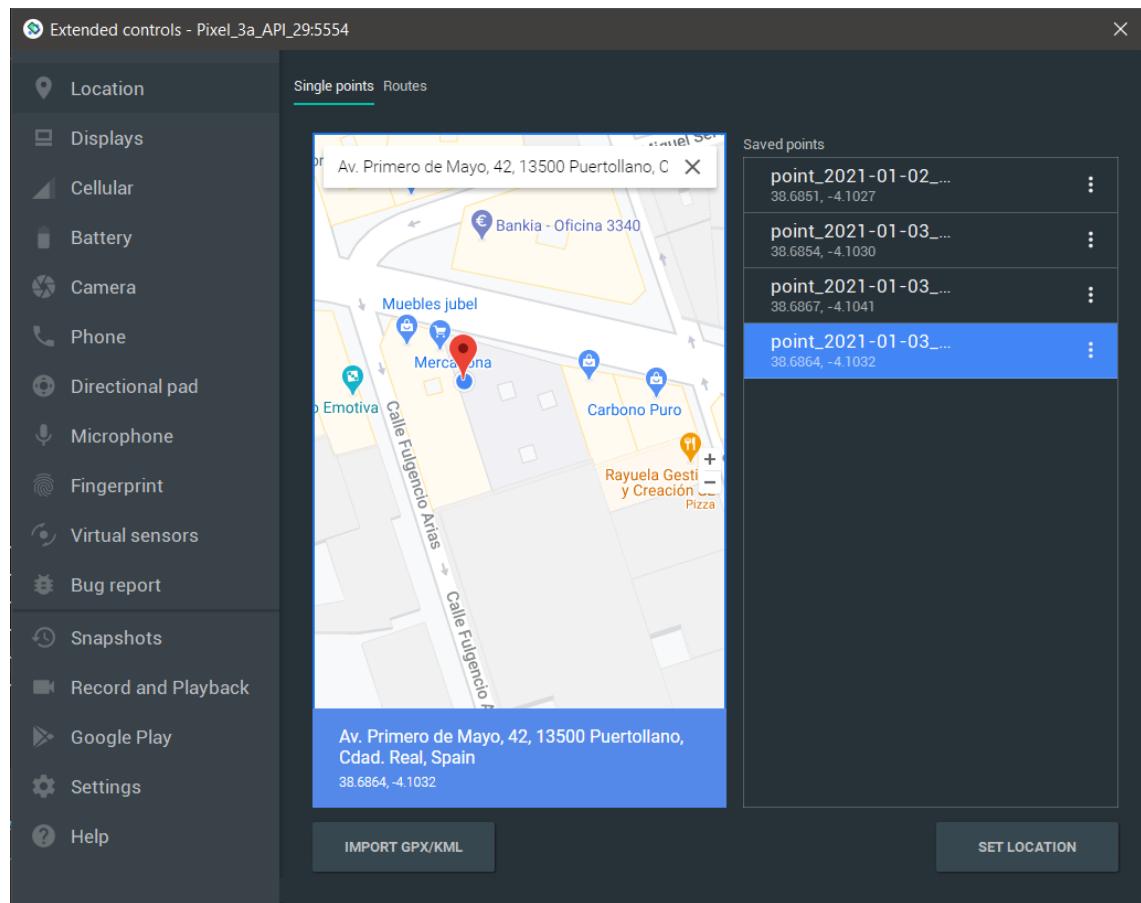


Caso de prueba 6 (Caso de Uso ver localización foto)

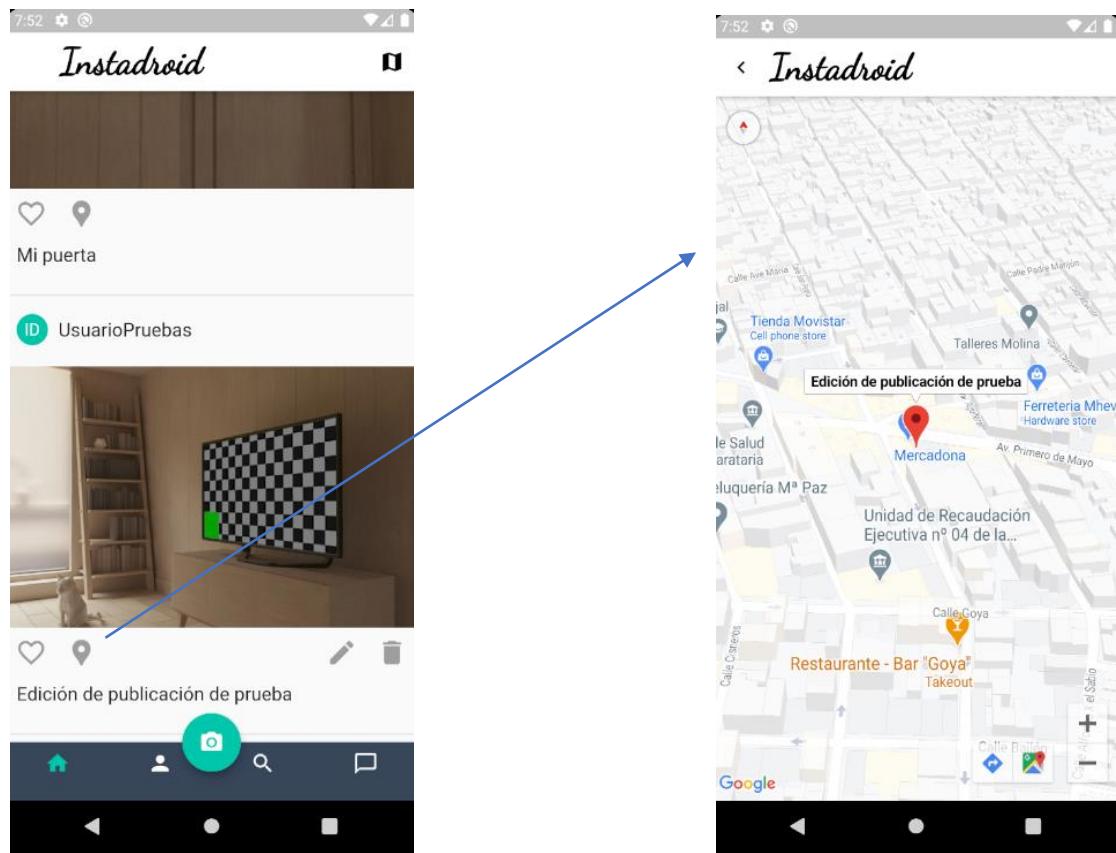
En este caso de prueba voy a verificar que la localización desde la que se realizó la publicación se muestra correctamente.

Resultado: Ok → La localización de la publicación se muestra correctamente.

Esta es la localización que he fijado en el emulador:



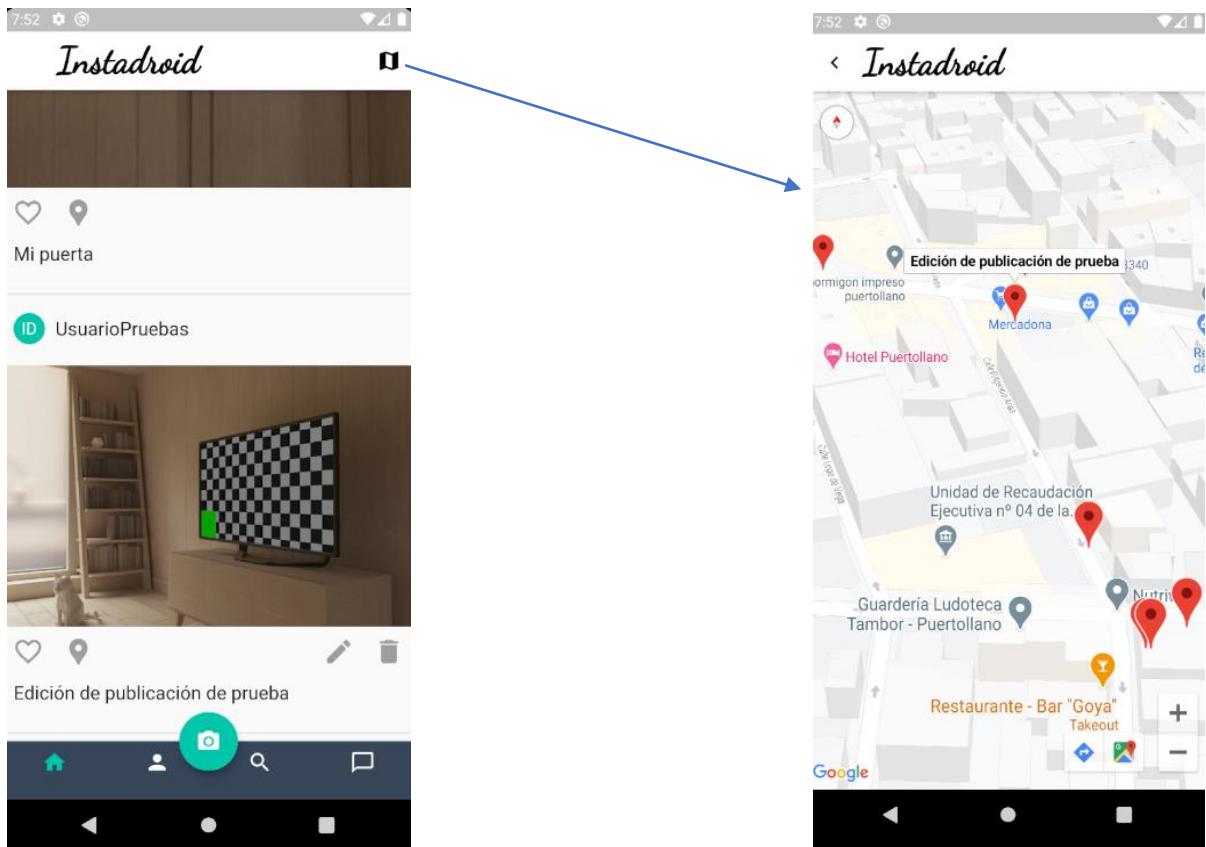
Esta es la localización en la que se muestra la publicación:



Caso de prueba 7 (Caso de Uso ver localización de todas las fotos)

Comprobamos que la publicación que acabamos de crear y editar aparece junto con todas las demás publicaciones situada en el mapa:

Resultado: Ok → La publicación se muestra en el mapa de todas las publicaciones.



Caso de prueba 8 (Caso de Uso eliminar publicación)

Primero comprobamos que se le pide una confirmación al usuario cuando quiera borrar una publicación:

Resultado: Ok → Este es el resultado si pulsamos en el ícono de la papelera



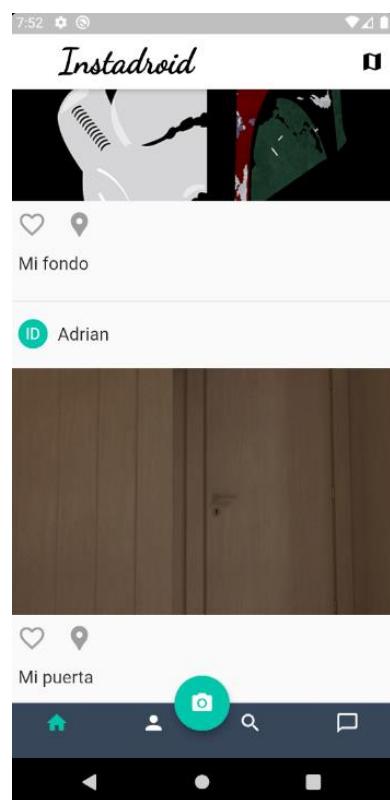
Una vez aceptamos, comprobamos que la publicación se elimina de realtime database y también desaparece de nuestro timeline de publicaciones:

La última publicación de realtime database ya no es esa:

The screenshot shows the Firebase Realtime Database interface. On the left, there's a sidebar with various services like Authentication, Firestore, Realtime Database, Storage, Hosting, Functions, and Machine Learning. The main area is titled "Realtime Database" and has tabs for "Datos", "Reglas", "Copias de seguridad", and "Uso". Below the tabs, there's a note about creating prototypes with the Local Emulator Suite. A preview window shows a URL: <https://instadroid-45787-default-rtbd.firebaseioapp.com/>. The database structure is displayed as a tree, with one node expanded to show its properties: foto, idUsuario, latitud, longitud, meGustas, and titulo.

Key	Value
foto	"https://firebasestorage.googleapis.com/v0/b/ins..."
idUsuario	"-MQ8npM0eg5QydnTCkE3"
latitud	37.4219983
longitud	-122.084
meGustas	0
titulo	"Mi puerta"

Y ya ha desaparecido del timeline:



2. Caso de prueba 9: Se comprobará si la aplicación es intuitiva y fácil de usar con un usuario no especializado

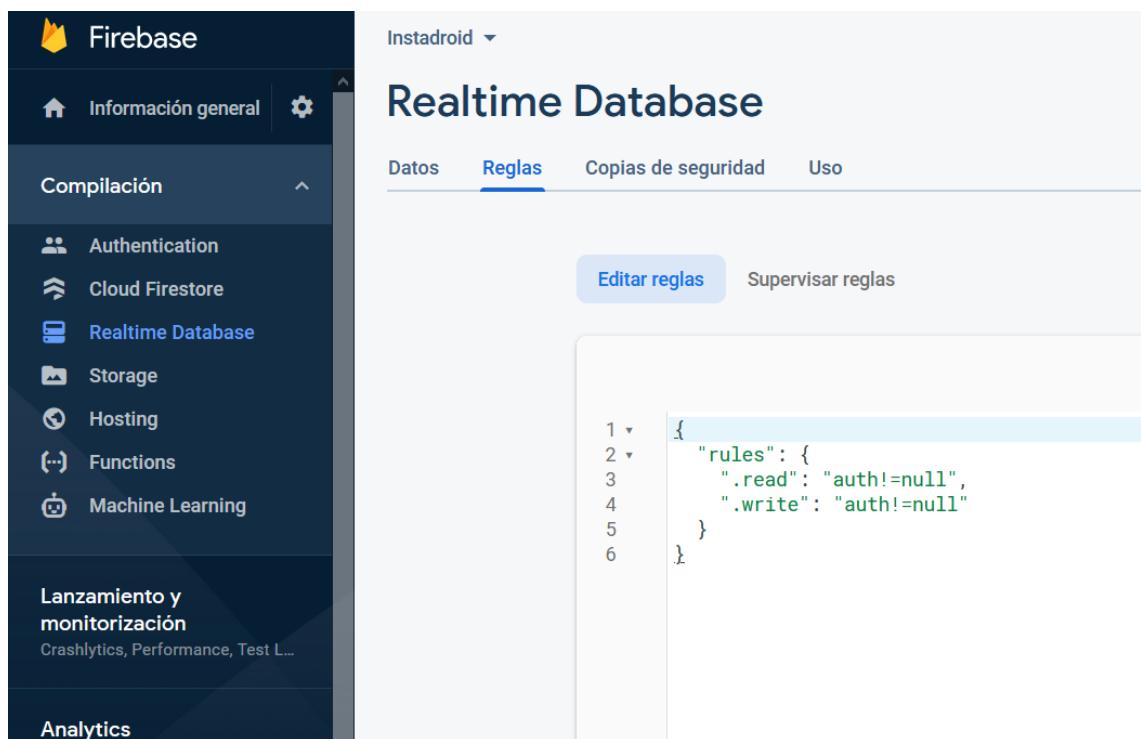
Se realizó un recorrido por la aplicación con un usuario no especializado, todo fue bastante intuitivo solo que antes en la página de subir foto no tenía el botón para desplegar el menú de elegir de cámara o galería, había que pulsar en el gif para hacerlo. Apunté esa conclusión y la solucioné posteriormente.

3. Caso de prueba 10: Se comprobará si un usuario no registrado puede acceder a la aplicación

Como ya hemos visto, no se deja acceder a un usuario que no esté registrado en la aplicación, por lo que el acceso es seguro (remito al caso de prueba 2).

4. Caso de prueba 11: Se comprobará si quitando el token de autorización a la hora de realizar peticiones a Firebase obtenemos resultados en nuestras consultas al backend

Cuando nos logueamos contra el servicio de autenticación de Firebase, el mismo nos devuelve un token que nosotros utilizaremos para autenticarnos en nuestras peticiones hacia Firebase (por ejemplo, tenemos que editar las reglas de acceso a realtime database para que solo deje escribir y leer si hay autenticación):



The screenshot shows the Firebase Realtime Database rules editor. On the left, there's a sidebar with navigation links like 'Información general', 'Compilación', 'Authentication', 'Cloud Firestore', 'Realtime Database' (which is selected), 'Storage', 'Hosting', 'Functions', and 'Machine Learning'. Below that is a section for 'Lanzamiento y monitorización' (Deployment and monitoring) with links to Crashlytics, Performance, and Test Lab. At the bottom is an 'Analytics' section. The main area is titled 'Realtime Database' and has tabs for 'Datos', 'Reglas' (which is selected), 'Copias de seguridad', and 'Uso'. There are buttons for 'Editar reglas' (Edit rules) and 'Supervisar reglas' (Monitor rules). The rules editor itself shows the following code:

```
1  {
2    "rules": {
3      ".read": "auth != null",
4      ".write": "auth != null"
5    }
6 }
```

En este caso y como ya vimos en los aspectos relevantes del código de nuestra aplicación, Instadroid realiza sus peticiones siempre mandando el token:

```
//Borramos una publicación de Firebase
Future<bool> borrarPublicacion (String idPublicacion) async {
    final url = '${_baseUrl}/publicaciones/$idPublicacion.json?auth=${_prefs.token}';
    await http.delete(url);
    return true;
}
```

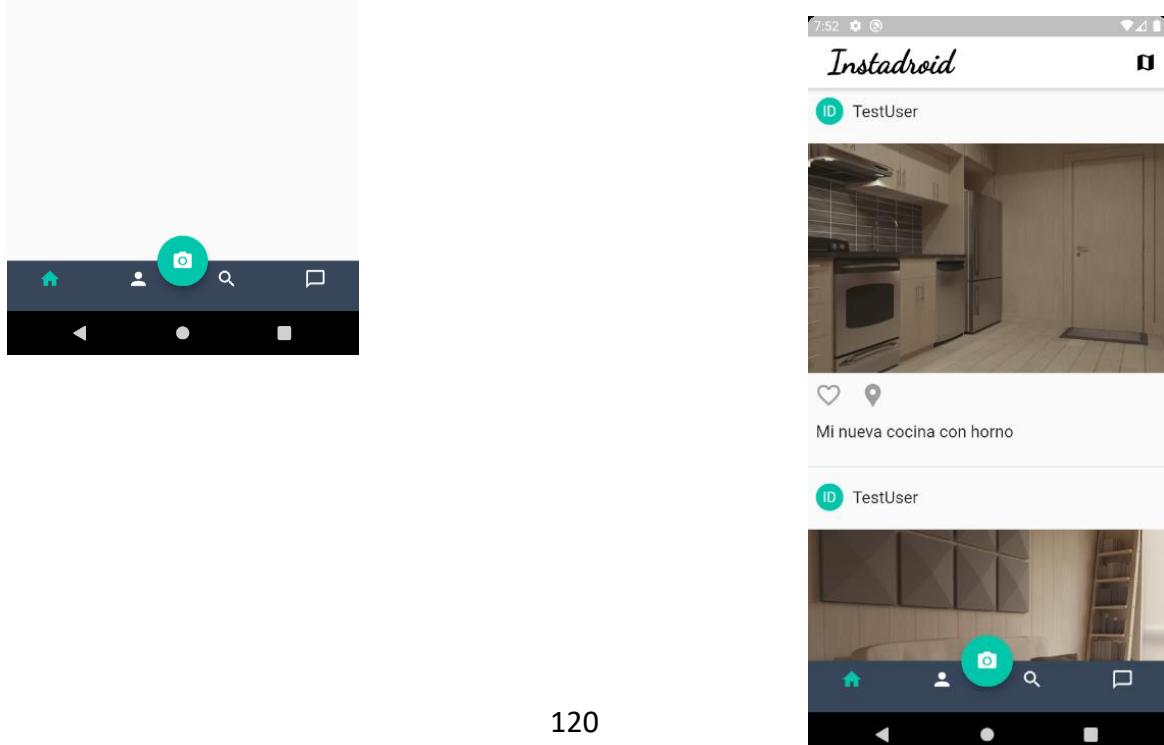
Vamos a realizar el caso de prueba de que, si le quito el token a la petición de listar las publicaciones, no me debería mostrar ninguna:

```
//Listamos las publicaciones que tenemos en Firebase
Future<List<Publicacion>> listarPublicaciones() async {
    //auth es el método de autenticación de firebase, por lo que mandamos
    //nos manda cuando el usuario se loguea
    final url = '${_baseUrl}/publicaciones.json';
```

Efectivamente, se queda cargando la lista, pero nunca llega a mostrarla. Si lo vuelvo a poner:



Entonces si carga la lista:



3. Solución a problemas encontrados

Siendo totalmente sincero, solo me he encontrado con un problema durante el desarrollo de la funcionalidad indicada para Instadroid. Y es el que he comentado, realizando las pruebas de usuario respecto a lo intuitiva y fácil de manejar que es la aplicación, todo estaba correcto salvo que no sabía cómo subir una foto porque antes tenía que pulsar en el gif que hay a la hora de subir una publicación para que aparezca el menú de cámara/galería.

Simplemente corregí el layout de esa pantalla para que quedara como se encuentra ahora mismo.

Durante el desarrollo problemas no han surgido, me formé bien en Flutter y no ha habido problemas al respecto (la aplicación solo me falló una vez porque me faltó utilizar un Future Builder). El principal problema, si se puede llamar así, ha sido tener que formarme, y adaptarme a cómo trabaja flutter y a manejar la asincronía. Todo lo demás ha salido como esperaba.

LANZAMIENTO Y PUESTA EN MARCHA

1. Aspectos relevantes del despliegue

El despliegue de productos software es el conjunto de actividades que hacen que un sistema de software esté disponible para su uso. Estas actividades pueden ocurrir en el lado del desarrollador o en el lado del consumidor. Al ser un sistema software diferente de los demás, los procesos concretos dentro de cada actividad del despliegue difícilmente pueden definirse⁸⁴.

Generalmente, un despliegue suele contar con las siguientes actividades:

- Lanzamiento: Las operaciones necesarias para preparar un sistema para el compilado y su transferencia a los sistemas informáticos en los que se ejecutará en producción.
- Instalación y activación: La instalación implica establecer algún tipo de comando, script o configuración del sistema para ejecutar el software. La activación consiste en iniciar el software por primera vez.

⁸⁴ https://es.wikipedia.org/wiki/Despliegue_de_software

- Desactivación: Es la inversa de la activación, se cierra un software que ya se está ejecutando en un sistema.
- Actualización: Reemplaza una versión anterior del software en tu totalidad o en parte por una versión más reciente.
- Seguimiento de versiones: Ayudan al usuario a encontrar e instalar actualizaciones a los sistemas de software.
- Adaptación: Es el proceso para modificar un sistema que se ha instalado previamente. Se diferencia de la actualización en que la adaptación es iniciada por eventos locales (como cambiar el entorno del sitio del cliente), mientras que la actualización es consecuencia de la disponibilidad de una nueva versión.

2. El despliegue de Instadroid

Debemos comprender Instadroid como dos partes bien definidas:

- La parte del backend, que es la que se encarga de proveernos de los datos
- La parte de la aplicación móvil, que es la que recibe esos datos y los muestra y permite al usuario interactuar con ellos

Estas dos partes, se despliegan de una manera distinta.

■ Parte del servidor/backend

Por norma general, los servidores de datos de las aplicaciones, cuando son creados por los propios desarrolladores, se despliegan utilizando distintos proveedores, como Amazon Web Services⁸⁵ o Heroku⁸⁶ (entre muchos otros).

⁸⁵ [⁸⁶ <https://www.heroku.com/>](https://aws.amazon.com/es/free/?all-free-tier.sort-by=item.additionalFields.SortRank&all-free-tier.sort-order=asc&awsf.Free%20Tier%20Types=*all&awsf.Free%20Tier%20Categories=categories%23comput&e&trk=ps_a134p000004f2ohAAA&trkCampaign=acq_paid_search_brand&sc_channel=PS&sc_campaig&n=acquisition_EMEA&sc_publisher=Google&sc_category=Cloud%20Computing&sc_country=EMEA&&sc_geo=EMEA&sc_outcome=acq&sc_detail=aws%20cloud&sc_content=Cloud%20Hosting_e&sc_matc&htype=e&sc_segment=467752261647&sc_medium=ACQ-P|PS-GO|Brand|Desktop|SU|Cloud%20Computing|Solution|EMEA|EN|Text|xx|EU&s_kwcid=AL!4422!3!467752261647!e!!g!!aws%20cloud&ef_id=CjwKCAiAudD_BRBXEiwAudakXxiHvwSP3fHJSUSircqAFwfuuVz-qeYBasBix_OBqKzlBX7_O0tRZxoCVm0QAvD_BwE:G:s&s_kwcid=AL!4422!3!467752261647!e!!g!!aws%20cloud</p>
</div>
<div data-bbox=)

Esos son servidores en la nube, igualmente podríamos tener una máquina física nosotros y utilizar otros métodos como Weblogic⁸⁷ de Oracle o Apache HTTP server⁸⁸.

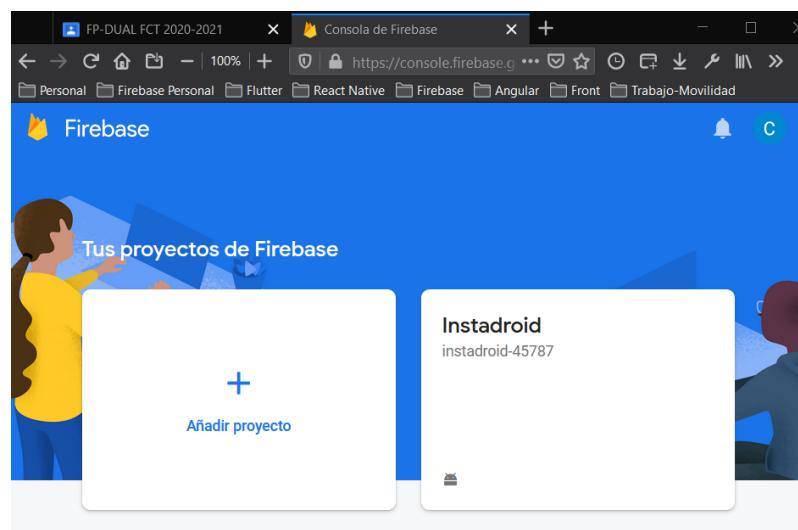
Además, lo normal en estos casos es envolver nuestra aplicación en un contenedor Docker para que pueda ejecutarse de manera independiente⁸⁹ y eso sería lo que desplegamos en uno de los servidores anteriores.

En resumen, hay muchos métodos para realizar un despliegue. Pero nosotros no tenemos que hacer nada de eso.

Gracias a nuestra elección de Firebase, no tenemos que preocuparnos en realizar nosotros un despliegue porque sus servicios ya se encuentran desplegados desde el momento en el que creamos un proyecto (de hecho, elegimos la ubicación de la base de datos, del almacenamiento, etc).

Por lo que, no es que nuestro backend no esté desplegado, si no que no lo tenemos que hacer nosotros. Como puede observarse, la decisión de utilizar Firebase es, bajo mi punto de vista, totalmente acertada. No tenemos que preocuparnos de realizar un backend, ya que todo lo realiza Firebase por nosotros.

Toda esta “configuración” de Firebase se realiza cuando creamos el proyecto. Para ello simplemente accedemos a la consola de Firebase⁹⁰:



⁸⁷ <https://www.oracle.com/es/java/weblogic/>

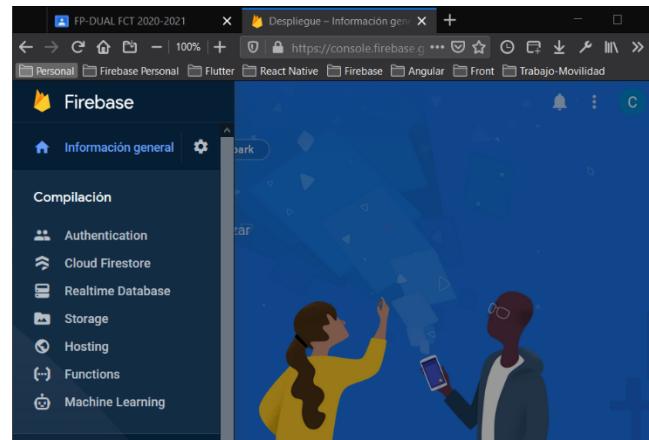
⁸⁸ <https://httpd.apache.org/>

⁸⁹ <https://www.redhat.com/es/topics/containers/what-is-docker>

⁹⁰ <https://console.firebaseio.google.com/u/2/>

Le damos un nombre al proyecto y seguimos los pasos. Ya tenemos nuestro proyecto creado.

En el menú lateral escogemos un servicio que queramos añadir, por ejemplo, el que hemos utilizado nosotros como base de datos, realtime database.



Una vez le demos a crear base de datos, se nos pedirá elegir dónde queremos que esté ubicada:

A screenshot of the 'Realtime Database' configuration screen. At the top, it says 'Realtime Database'. Below that, a blue header bar contains the title 'Configurar base de datos' and two numbered steps: '1 Opciones de base de datos' and '2 Reglas de seguridad'. The main content area has a heading 'Tus ajustes de ubicación indican dónde se guardarán los datos de Realtime Database.' Below this, there is a section titled 'Ubicación de Realtime Database' with a dropdown menu. The menu shows two options: 'Estados Unidos (us-central1)' which is highlighted in blue, and 'Bélgica (europe-west1) [beta]'. At the bottom of the screen, there are navigation arrows for the wizard.

Seleccionamos y ya tenemos nuestra base de datos desplegada. Y esto es tan solo un ejemplo, los demás servicios también se despliegan (como autenticación y storage) dónde nosotros queramos entre las opciones disponibles.

■ Parte del cliente (aplicación móvil Instadroid)

Para la puesta en marcha y despliegue de la aplicación cliente, debemos poner la misma a disposición de los usuarios en las tiendas de aplicaciones (PlayStore para Android y AppStore para IOS, recordemos que Instadroid es una aplicación multiplataforma).

Para subir una aplicación al PlayStore de Google debemos⁹¹:

- Crearnos una cuenta de desarrollador: Necesaria para acceder a Google Play. Deberemos realizar un único pago de 25 dólares.
- Accedemos a Google play console, seleccionamos todas las aplicaciones y le damos a crear aplicación. Aquí insertaremos el título de la aplicación, así como los idiomas predeterminados. Le damos a crear.
- Rellenamos la ficha de la aplicación en la que especificaremos detalles de la app, pondremos fotos para que se vean en el PlayStore, le daremos una categoría, etc.
- Generamos nuestra APK. Hay muchas formas de hacerlo (los propios IDE ofrecen esa funcionalidad). Como único requisito es que tiene que estar firmada.
- Establecemos precio y distribución (en qué países va a estar nuestra app disponible).
- Enviamos la app a revisión y esperamos, ya que puede tardar un par de días en validarse la aplicación y estar disponible en el Play Store.

Para subir una aplicación a la App Store de Apple tenemos que⁹²:

- Registrarnos como particulares en el programa de desarrolladores de Apple, debemos introducir nuestra tarjeta de crédito (el pago es de 99 dólares anuales).
- Crear un certificado de Apple para la aplicación. Se hace a través de Xcode, que es el IDE de Apple para desarrollar aplicaciones para IOS.
- Registrar un dispositivo móvil de Apple en nuestra cuenta de desarrollador.

⁹¹ <https://www.yeeply.com/blog/guia-subir-app-google-play-store/>

⁹² <https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/publicar-una-app-en-la-apple-store-crear-una-app/>

- Generar un id para la aplicación.
- Crear un perfil de aprovisionamiento, que combina todos los ajustes anteriores en un único perfil.
- Publicar la aplicación a través de Itunes Connect, para ello desde ahí, click en myapps, y luego en añadir. Rellenamos con los datos de nuestra aplicación y le damos a créate. Aquí también calificaremos la edad de nuestra aplicación, así como si es gratuita o de pago.
- Diseñamos la página de descripción de la app. Aquí también cargamos el archivo build de nuestra aplicación, que generamos desde Xcode y definimos su ícono y otros aspectos.
- Esperamos dos días para que la aplicación se revise y pase a estar disponible.

3. Manual de uso

El manual de uso se encuentra disponible en: <https://carlosdez23.github.io/Instadroid/>

Utilizando GitHub Pages se ha realizado un manual de usuario web en el que se explica de forma breve y concisa cómo se utiliza la app.

VALORACIÓN Y CONCLUSIONES

Cuando me llegó el enunciado del proyecto, la verdad es que me eché las manos a la cabeza. No podíamos utilizar Java, que era lo que habíamos estado utilizando todo el año anterior para realizar aplicaciones móviles.

Entonces, caí en cuenta de las palabras que siempre nos han dicho respecto a nuestro futuro oficio “esto avanza muy rápido, hay que estar en continua reinvenCIÓN y siempre formándose”. El proyecto final era un claro ejemplo de ello.

Al principio, como solo teníamos que documentar, es un poco pesado, porque son cosas que no habíamos hecho nunca, para las que hay que informarse mucho y que cuesta sacar adelante. Pero ha sido bueno aprenderlo ya que ya contamos con formación para conocer en profundidad las fases más tempranas de un proyecto software, y por lo menos cuando veamos unos diagramas de casos de uso no nos vamos a echar a temblar y los vamos a comprender mucho mejor.

También tuvimos que sacar lo aprendido en la asignatura de EIE y hacer cuentas para ver cómo financiábamos nuestra aplicación y además tuvimos que examinar el mercado y

nuestra competencia, para ver cómo puede entrar la aplicación por el ojo del cliente. E incluso tuvimos que tirar de FOL para la prevención de riesgos del proyecto.

Para la realización del análisis tuvimos que tirar de habilidades sociales para sacarle los requisitos al cliente y después aplicar lo aprendido en entornos de desarrollo para hacer los diagramas de casos de uso.

De igual manera, tuvimos que prototipar la interfaz de nuestra aplicación (ahí si que me eché las manos a la cabeza), pero más o menos con los conocimientos de la asignatura de Desarrollo de Interfaces, pude sacar algo decente.

También tuvimos que diseñar diagramas de bases de datos (aunque sea NoSQL), diagramas de clases, de arquitectura. Cosas que nunca había creído que iba a poder hacer, pero gracias a la “caña” que hemos recibido en estos dos años he podido sacar adelante.

A la hora de la implementación de la solución he disfrutado mucho, porque ha sido una buenísima oportunidad para aprender dos tecnologías que me han gustado mucho: Flutter y Firebase. Una de las cosas que más destaco de mi formación en este instituto es que me ha hecho “buscarme la vida”, por lo que todo lo aprendido ha sido por mi cuenta y estoy orgulloso de ello.

Para implementar, he notado mucho cómo las asignaturas de PSP (asincronía en las peticiones), Programación Multimedia (desarrollo de una aplicación móvil propiamente dicha) y Acceso a Datos (consultar datos de donde sea hasta de un XML) me han ayudado mucho a desempeñar una correcta implementación. También la asignatura SGE, al aprender Python me ha enseñado a comprender mejor las estructuras de datos como los mapas, que se utilizan mucho en Flutter.

En resumen, ha sido una experiencia en la que me he dado cuenta de cómo toda mi formación encajaba perfectamente para desempeñar esta labor.

Como valoración al proyecto, la verdad, me gustaría que las cosas cambiaron. Pero no depende ni de mí ni del instituto. Debería ser algo más parecido al currículo de DAW, en el que todos trabajan en conjunto para un mismo proyecto que es una situación mucho más cercana a la realidad laboral de nuestra profesión. No sé hasta qué punto esto podrá cambiar, porque no es la realidad a no ser que seas freelance que tú te enfrentes sólo a todos y cada uno de los pasos del ciclo de vida de un proyecto software.

Creo que ese sería un avance en la dirección correcta.

BIBLIOGRAFÍA/WEBGRAFÍA

Recursos web utilizados en el desarrollo de este proyecto:

<https://blog.mailrelay.com/es/2018/09/06/estudio-de-mercado>

<https://obsbusiness.school/es/blog-investigacion/marketing-y-comunicacion/como-hacer-un-estudio-de-mercado-en-4-pasos>

Análisis de la lectura Carlos Merino Moreno, Consecuencias sociales y económicas de las redes sociales, Universitat Oberta de Catalunya (www.uoc.edu). Ubicación del recurso:

http://openaccess.uoc.edu/webapps/o2/bitstream/10609/54384/2/Consecuencias%20sociales%20y%20econ%C3%B3micas%20de%20las%20redes%20sociales_M%C3%B3dulo1.pdf

<https://www.rdstation.com/es/blog/publico-objetivo-cliente-ideal-buyer-persona/>

<https://www.infoautonomos.com/plan-de-negocio/analisis-dafo/>

<https://www.gitkraken.com/boards>

Enlace al tablero Kanban del repositorio:

<https://github.com/CarlosDez23/Instadroid/projects/1>

<https://gfourmis.co/gitflow-sin-morir-en-el-intento/>

Dirección de la wiki del repositorio de Instadroid:

<https://github.com/CarlosDez23/Instadroid/wiki>

<https://docs.github.com/es/free-pro-team@latest/github/building-a-strong-community/about-wikis>

<https://capitulo7pmbok.wordpress.com/gestion-de-los-costos-del-proyecto/>

<https://www.toptal.com/agile/estimacion-de-costos-de-software-en-gestion-de-proyectos-agiles>

<https://www.toptal.com/agile/estimacion-de-costos-de-software-en-gestion-de-proyectos-agiles>

<https://www.yeeply.com/blog/cuanto-cuesta-crear-una-app/>

<https://www.masquenegocio.com/2017/07/19/desarrollar-app-espana/>

https://cincodias.elpais.com/cincodias/2015/02/01/lifestyle/1422792260_243066.html

<https://www.yeeply.com/blog/cuanto-cuesta-crear-una-app/>

<https://aws.amazon.com/es/api-gateway/pricing/> y Firebase

<https://firebase.google.com/docs/firestore/billing-example?hl=es>

https://es.gofundme.com/start?utm_source=google&utm_medium=cpc&utm_campaign=SE_NonBrand_ES_SP_Tier1&utm_content=Crowdfunding&utm_term=crowdfunding_e_c_&gclid=Cj0KCQjw59n8BRD2ARIsAAmgPmKjQZb86o5AS4Ou6fHTVCDAxu1bgE08UCz9TddQBb7GX_xu_pV_G5UaAh3IEALw_wcB

https://es.wikipedia.org/wiki/Visual_Studio_Code

https://es.wikipedia.org/wiki/Microsoft_Office_365

<https://blog.desdelinux.net/gitkraken-un-excelente-cliente-de-git-multiplataforma-para-tu-escritorio/>

<http://www.invassat.gva.es/es/que-es-prevencion-de-riesgos-laborales>

<https://www.boe.es/buscar/act.php?id=BOE-A-1995-24292>

<https://prevencionugtandalucia.es/riesgos-laborales-en-el-sector-de-la-informatica/>

<https://www.preverlab.com/nuevo-reglamento-de-instalaciones-de-proteccion-contra-incendios/>

https://es.wikiversity.org/wiki/Gesti%C3%B3n_de_riesgos_de_proyectos_software

<http://www.luiscarlosaceves.com/udem/ppd/riesgossoftware.pdf>

https://es.wikipedia.org/wiki/Requisito_funcional

https://www.ecured.cu/Requisitos_no_funcionales

<https://www.infor.uva.es/~mlaguna/is1/apuntes/2-requisitos.pdf>

https://es.wikipedia.org/wiki/Caso_de_uso

https://upload.wikimedia.org/wikipedia/commons/thumb/9/9e/Notacion_Caso_de_Uso.svg/1200px-Notacion_Caso_de_Uso.svg.png

https://upload.wikimedia.org/wikipedia/commons/thumb/8/80/UML_diagrama_caso_de_uso.svg/498px-UML_diagrama_caso_de_uso.svg.png

<https://www.ctr.unican.es/asignaturas/is1/is1-t04-trans.pdf>

https://es.wikipedia.org/wiki/Dise%C3%B1o_de_interfaz_de_usuario

<https://www.efectedigital.online/post/2018/04/18/dise%C3%B1o-de-interfaz-de-usuario-ui>

https://www.justinmind.com/?utm_medium=cpc&utm_source=google&utm_campaign=1063145459&utm_term=just%20in%20mind_e&gclid=Cj0KCQiAwMP9BRCzARIAPWTJ_Hcqj5e74vQ_sa9ZTJ63onwlUPrAMiM_OhzAttpblTmmU3iDLU-MqkaAhQVEALw_wcB

<https://fonts.google.com/specimen/Roboto>

<https://fonts.google.com/specimen/Dancing+Script?query=dancing>

https://es.wikipedia.org/wiki/Diagrama_de_clases

[https://es.wikipedia.org/wiki/Clase_\(inform%C3%A1tica\)](https://es.wikipedia.org/wiki/Clase_(inform%C3%A1tica))

<https://joanpaon.wordpress.com/2013/06/06/uml-diagramas-de-clases-relacion/>

<https://uniwebsidad.com/libros/algoritmos-python/capitulo-11/persistencia-de-datos>

<https://www.ionos.es/digitalguide/hosting/cuestiones-tecnicas/nosql/>

<https://www.grapheverywhere.com/bases-de-datos-nosql-marcas-tipos-ventajas/>

<https://medium.com/@eugeniomendoza/c%C3%B3mo-saber-si-necesitas-una-base-de-datos-nosql-b6cf5bb7d9b>

<https://eaminds.com/2018/08/03/modelando-nosql-data-bases/>

https://es.wikipedia.org/wiki/Arquitectura_de_software

<https://manuel.cillero.es/doc/metodologia/metrica-3/procesos-principales/dsi/actividad-1/>

<https://ittgweb.wordpress.com/2016/05/29/3-7-diseno-de-software-de-arquitectura-de-tiempo-real/>

https://es.wikipedia.org/wiki/Arquitectura_orientada_a_servicios

https://es.wikipedia.org/wiki/Servicio_web

<https://es.stackoverflow.com/questions/267497/explicaci%C3%B3n-comunicaci%C3%B3n-cliente-android-con-la-servidor-en-firebase>

<https://blog.powerdata.es/el-valor-de-la-gestion-de-datos/ciclo-de-vida-de-un-sistema-de-informacion-fases-y-componentes>

<https://www2.deloitte.com/es/es/pages/technology/articles/que-es-react-native.html>

<https://reactnative.dev/>

<https://www.qualitydevs.com/2019/05/31/que-es-ionic-desarrollador-web/>

<https://openwebinars.net/blog/ionic-framework-ventajas-desventajas/>

<https://www.bbvnexttechnologies.com/kotlin-native-ayer-hoy-y-manana/>

<https://es.wikipedia.org/wiki/NativeScript>

<https://desarolloweb.com/articulos/ionic-vs-nativescript.html>

<https://docs.microsoft.com/es-es/xamarin/get-started/what-is-xamarin>

<https://inmediatum.com/blog/ingenieria/ventajas-y-desventajas-de-apps-desarrolladas-en-xamarin/>

<https://www.qualitydevs.com/2019/07/05/que-es-flutter/>

<https://www.bbvnexttechnologies.com/es-flutter-el-framework-del-futuro/>

<https://alexmarket.medium.com/flutter-o-nativescript-707b7ffdbceb>

<https://flutter.dev/>

<https://blog.back4app.com/es/las-mejores-alternativas-a-firebase/>

<https://www.iebschool.com/blog/firebase-que-es-para-que-sirve-la-plataforma-desarrolladores-google-seo-sem/>

<https://firebase.google.com/docs/auth/android/start?hl=es-419>

<https://www.youtube.com/user/Firebase>

<https://www.digital55.com/desarrollo-tecnologia/que-es-firebase-funcionalidades-ventajas-conclusiones/>

<https://flutter.dev/docs/development/ui/widgets-intro>

<https://manuel.cillero.es/doc/metodologia/metrica-3/procesos-principales/asi/actividad-10/>

<https://manuel.cillero.es/doc/metodologia/metrica-3/tecnicas/pruebas/unitarias>

<https://manuel.cillero.es/doc/metodologia/metrica-3/tecnicas/pruebas/integracion/>

<https://manuel.cillero.es/doc/metodologia/metrica-3/tecnicas/pruebas/sistema/>

<https://manuel.cillero.es/doc/metodologia/metrica-3/tecnicas/pruebas/implantacion/>

<https://manuel.cillero.es/doc/metodologia/metrica-3/tecnicas/pruebas/aceptacion/>

<https://manuel.cillero.es/doc/metodologia/metrica-3/tecnicas/pruebas/regresion/>

https://es.wikipedia.org/wiki/Despliegue_de_software

https://aws.amazon.com/es/free/?all-free-tier.sort-by=item.additionalFields.SortRank&all-free-tier.sort-order=asc&awsf.Free%20Tier%20Types=*all&awsf.Free%20Tier%20Categories=categories%23compute&trk=ps_a134p000004f2ohAAA&trkCampaign=acq_paid_search_brand&sc_channel=PS&sc_campaign=acquisition_EMEA&sc_publisher=Google&sc_category=Cloud%20Computing&sc_country=EMEA&sc_geo=EMEA&sc_outcome=acq&sc_detail=aws%20cloud&sc_content=Cloud%20Hosting_e&sc_matchtype=e&sc_segment=467752261647&sc_medium=ACQ-P|PS-GO|Brand|Desktop|SU|Cloud%20Computing|Solution|EMEA|EN|Text|xx|EU&s_kwcid=AL!4422!3!467752261647!e!!g!!aws%20cloud&ef_id=CjwKCAiAudD_BRBXEiwA_udakXxiHvwSP3fHJSUSircqAFwfuuVz-qeYBasBix_OBqKzlBX7_O0tRZxoCVm0QAvD_BwE:G:s&s_kwcid=AL!4422!3!46752261647!e!!g!!aws%20cloud

<https://www.heroku.com>

<https://www.oracle.com/es/java/weblogic/>

<https://httpd.apache.org/>

<https://www.redhat.com/es/topics/containers/what-is-docker>

<https://console.firebaseio.google.com/u/2/>

<https://www.yeeply.com/blog/guia-subir-app-google-play-store/>

<https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/publicar-una-app-en-la-apple-store-crear-una-app/>