
UNIVERSITAT DE BARCELONA

**Trabajo de Final de Grado**

**GRADO DE INGENIERÍA INFORMÀTICA**

**Facultad de Matemáticas y Informática**
**Universidad de Barcelona**

---

# Implementación de la aplicación móvil MyBoards

---

**Pedro Pizarro Huertas**

Director: Oscar Amoros Huguet
Realizado en: Departamento de Matemáticas y Informática

Barcelona, 20 de junio de 2021

# Index

# Abstract (Castellano)

Estar expuesto en las redes sociales puede acarrear fuertes consecuencias que pueden determinar la vida de las personas. Hoy en día puedes encontrar trabajo en una red social, vender tu coche, encontrar pareja o arruinar tu carrera. El ser humano no ha sabido adaptar esta nueva revolución a sus vidas, pues muchas personas son influenciadas por este gran gigante el cual puede llegar a generar un vínculo con el usuario que puede llegar a ser toxico. Además, no hay ningún impedimento real que proteja al usuario de sufrir estas consecuencias.

En este TFG hablaremos de MyBoards, una “red social” diferente en este aspecto. Se tratará de una aplicación móvil implementada únicamente para Android en un principio, que pretende desvincular al usuario del contenido y presentar a este de una forma en la que:
No hay ninguna jerarquía entre los usuarios, ninguno vale más que otro. El contenido será renovado por la propia comunidad de forma periódica. La anonimidad le brindará la protección para expresarse libremente además de dejarle a nadie más que al propio usuario el derecho a juzgarse a sí mismo.

En este documento presentamos el diseño e implementación de un prototipo de aplicación que cumpla con las principales características descritas.

# Abstract (English)

Being exposed on social networks can have strong consequences that can determine people's lives. Nowadays you can find work on a social network, sell your car, find a partner or ruin your career. Human beings have not been able to adapt this new revolution to their lives, since many people are influenced by this great giant, which can make connections with the user that could become toxic. There is no real impediment that protects the user from suffering these consequences.

In this TFG we will talk about MyBoards, a different "social network" in this regard. MyBoards is a mobile application which aims to unlink the user from the content and present it in a way that there is no hierarchy between users, none is worth more than another. The content will be renewed by the community itself periodically. Anonymity will give you the protection to express yourself freely in addition to leaving no one but the user himself the right to judge himself.

In this document will be presented the design and implementation of an application prototype that meets the main characteristics described.

# Abstract (Català)

Estar exposat en les xarxes socials pot implicar fortes conseqüències que poden determinar la vida de les persones. Avui dia pots trobar treball en una xarxa social, vendre el teu cotxe, trobar parella o arruïnar la teva carrera. L'ésser humà no ha sabut adaptar aquesta nova revolució a les seves vides, perquè moltes persones són influenciades per aquest gran gegant el qual pot arribar a generar un vincle amb l'usuari que pot arribar a ser toxic. A més, no hi ha cap impediment real que protegeixi a l'usuari de sofrir aquestes conseqüències.

En aquest TFG parlarem de MyBoards, una “xarxa social” diferent en aquest aspecte. Es tractarà d'una aplicació mòbil implementada únicament per a Android, en principi, que pretén desvincular a l'usuari del contingut i presentar aquet d'una forma en la qual:
No n’hi ha cap jerarquia entre els usuaris, cap val més que el següent. El contingut serà renovat per la pròpia comunitat de manera periòdica. L'anonimat li brindarà la protecció per a expressar-se lliurement a més de deixar-li a ningú més que al propi usuari el dret a jutjar-se a si mateix.

En aquest document presentem el disseny i implementació d'un prototip d'aplicació que compleixi amb les principals característiques descrites.

# 1. Introducción

Las redes sociales han sido una verdadera revolución para la sociedad, es por eso que grandes empresas han desplegado un gran abanico de metodologías para analizar las tendencias de los usuarios, mercantilizando sus intereses llegando a transformar los datos personales de sus usuarios en el producto. Además, existen muchos problemas psicológicos como la depresión, inseguridad emocional o ansiedad que se han podido ver acentuadas en la población a medida que las redes sociales pasaban a ser una pieza esencial en la vida de las personas. En este documento se concretarán los problemas que han supuesto las redes sociales hasta el momento y se mostrará cómo se ha llevado a cabo el desarrollo de una nueva red social en la cual el usuario este protegido frente a estos problemas.

## 1.1. Motivación

Existen dos aspectos que han motivado la realización de este proyecto. El primero es mi interés personal en el mundo de desarrollo de aplicaciones móviles. Ya cuento con dos años de experiencia laboral en el sector y aún sigo viendo como las aplicaciones móviles son una demanda muy significativa en el mercado.
El segundo es que, las redes sociales popularmente han generado un rechazo entre mis círculos sociales y por lo que puedo advertir en otros medios de comunicación es un rechazo más generalizado que particular. Existe una problemática en el modo en el que las redes sociales afectan a las personas.

## 1.2. Contenido de la memoria.

Este documento explicará cómo se han tratado todos los aspectos de la aplicación desde la investigación hasta la primera versión del proyecto. Para ello, el documento se ha estructurado de la siguiente forma:
Primero veremos los antecedentes el en mundo de las redes sociales. Haremos una pequeña apreciación de su servicio y se expondrán problemas que han podido ser consecuencia de su uso. Una vez planteados los problemas acontecidos en las redes sociales, veremos cuales son las soluciones a estas problemáticas que podría proporcionar MyBoards. Posteriormente hablaremos de como se ha organizado el flujo de trabajo a lo largo del desarrollo de la aplicación. Entonces, entraremos en la implementación, donde primero se hablará la idea de MyBoards más en detalle y veremos cómo pondrá solución a los problemas que plantearon el resto de redes sociales. Entonces hablaremos de la metodología que ha seguido el código, más tarde, hablaremos de todas las tecnologías y veremos cómo van a poder ser útiles al proyecto. También veremos que reglas se han seguido para realizar el diseño de la aplicación y finalmente hablaremos de los puntos más relevantes de la realización del código del proyecto. A continuación, veremos las pruebas realizadas en usuarios y los resultados extraídos. También se ha querido plasmar en el proyecto que aspectos de la aplicación no han podido llevarse a la ejecución. También hablaremos de ideas que en un futuro podrían implementarse en este proyecto. finalmente, encontraremos el anexo y la webgrafía con la información que respaldará en algunos puntos a este documento.

# 2. Objetivos

El principal objetivo de este proyecto es desarrollar una versión MVP (Producto viable mínimo) de una red social para la plataforma Android en la cual cumpla dos condiciones. La primera, que los usuarios sean anónimos, si los usuarios no tienen el miedo de que sus círculos o grandes entidades no puedan identificarle podrán expresarse con total libertad.
La segunda, que no exista una jerarquía entre los usuarios, la aplicación no dará más valor a una cuenta que otra, en otras plataformas podemos ver como el contenido de personas con un mayor número de seguidores adquieren una mayor visibilidad y más crédito. Si en una red social no existe un valor público que otorga un estatus superior de un usuario frente a otro, lo único que queda es el contenido. De esta forma, si ambos usuarios pueden publicar por igual y valorar por igual, el contenido publicado será el principal producto que consumirá el usuario. Dando este poder por igual podemos evitar que las personas puedas sentirse insatisfechas por no lograr el crédito que logran otros usuarios dentro de esta red social.

Para el desarrollo de esta aplicación existe la necesidad de llevar a cabo un código que gestione una base de datos de la cual se extraerá la información necesaria por la aplicación Android. Por lo tanto, debemos desarrollar un servicio que almacene y gestione la información que los usuarios publiquen en la aplicación. Además, también tendremos que desarrollar el cliente de Android, la cual tendrá que cumplir los estándares de usabilidad para sentirse cómodos navegando por la aplicación.

# 3. Antecedentes en las redes sociales

Hablaremos aquí de los problemas que un usuario puede sufrir, en tres de las grandes plataformas más utilizadas en el contexto actual: Facebook, Instagram y Twitter.

## 3.1. Facebook

Facebook es un servicio de redes y medios sociales en línea de origen estadounidense con sede en Menlo Park, California. Es una red social que fue creada para poder mantener en contacto a personas, y que éstos pudieran compartir información personal, noticias y contenidos audiovisuales con amigos y familiares.

### 3.1.1. Efectos Psicológicos en el individuo

Facebook ha sido el responsable de efectos adversos a la sociedad muy significativos, esto incluso está reflejado en fuentes como Wikipedia:
*"Facebook cuenta con más de 2700 millones de usuarios activos mensuales a fecha de marzo de 2018.4 Su popularidad ha supuesto a una ingente cobertura mediática de la compañía, como un escrutinio significativo sobre la privacidad y los efectos psicológicos que tiene en los usuarios. En los últimos años, la compañía se ha enfrentado con una intensa presión sobre la cantidad de fake news, la incitación al odio y las representaciones de violencia que prevalecen en sus servicios, aspectos que está intentando contrarrestar."*

Y es cierto que existe una gran cantidad de estudios que defienden que Facebook puede afectarnos psicológicamente. Hasta el punto de llegar a tener consecuencias serias en nuestra vida privada, podemos ver en algunos estudios:

Un estudio bajo la regulación de la Universidad de Surrey y la Sociedad Britanica de psicología concluye que a pesar de que Facebook promueve la conexión social, este contribuye a la creciente prevalencia de la soledad [1].

Un famoso estudio realizado por Julia Brailovskaia, Julia Velten y Jürgen Margaf llamado "Cyberpsichology, Behavior, and Social Networking" habla de cómo las interacciones sociales en Facebook pueden generar desordenes psicológicos en los individuos [2].

### 3.1.2. Impacto político

También podemos ver el impacto político que sus herramientas pueden facilitar a organizaciones o individuos en el famoso caso de Cambridge Analytica [3].
En este caso la consultora Cambridge Analytica adquirió de forma indebida información de 50 millones de usuarios de la red social en Estados Unidos. Esos datos privados fueron luego utilizados para manipular psicológicamente a los votantes en las elecciones de EE. UU. de 2016, donde Donald Trump resultó electo presidente.

### 3.1.3. Organizaciones

Por si los efectos psicológicos y sus implicaciones en ideologías fuesen poco también existe información que respalda, a pesar de ser algo de fácil percepción, que esta plataforma empuja a las personas hacia una menor privacidad.
*"Compartir información de cualquier naturaleza con una cantidad de personas posee una cierta vulnerabilidad para los usuarios y se requieren de métodos para controlar esos riesgos" [4]*

Esta falta de privacidad nos lleva a que otros tipos de organizaciones lleguen a beneficiarse de este vínculo entre las personas que construye Facebook:
*"Según informa Bloomberg, al menos una docena de grupos terroristas identificados por los EE.UU. mantiene una presencia en Facebook. Entre estos están Hamas y Hezbollah en Medio Oriente, Boko Haram en África Occidental y las FARC de Colombia. Estos grupos utilizan la plataforma para reclutar nuevos adeptos y difundir sus macabros contenidos." [5]*

Con todo esto no quiero señalar que Facebook es una plataforma con carácter destructivo, sino que es una plataforma que, a pesar de perseguir conductas de esta naturaleza como organización, no van a poder evitar que estas cosas sigan sucediendo, porque la propia idea de negocio es mostrar información y estrechar vínculos con cualquier usuario. Esto puede resultar imposible de controlar.

## 3.2. Instagram

Instagram es una aplicación principalmente orientada para móvil y red social de origen estadounidense, propiedad de Facebook, cuya función principal es poder compartir fotografías y vídeos con múltiples efectos fotográficos y en una variedad amplia de formatos entre usuarios.

### 3.2.1. Efectos psicológicos en el público joven

Es una red es puramente visual, por lo que se da mucha importancia a la calidad del contenido que suben los usuarios. En general, las personas la utilizan para compartir su lado más personal.
En Instagram el 71% de sus usuarios están comprendidos entre los 16 y los 34 años, muchos la denominan la aplicación de los jóvenes. Pero parece ser que según un estudio realizado por RSPH (Royal Society for Public Healt) y YHM (Young Health Movement) sitúan Instagram como la peor aplicación para la salud mental de la gente joven [6].
En el estudio mencionado anteriormente se mencionan problemas como: Ansiedad, Depresión, Soledad, problemas de expresión, problemas de identidad y bullying entre los 14 problemas identificados.
¿No es curioso que la aplicación con más público joven sea tan perjudicial para estos? Además, se ha creado la figura del "Influencer":
*"Persona que destaca en una red social u otro canal de comunicación y expresa opiniones sobre un tema concreto que ejercen una gran influencia sobre muchas personas que la conocen."*

En muchas ocasiones estos "influencers" se transforman en ídolos con un gran público y construyendo así una falsa expectación de como la vida de estas personas son e idealizándolas hasta el punto de que sus admiradores pueden llegar a sentirse miserables en comparación a ellos [7].

Pero de nuevo, estos “influencers” no tendrían estos efectos en una plataforma que no valorase a las personas en base a su número de seguidores.

## 3.3. twitter

Twitter es un servicio de micro blogueo, con sede en San Francisco, California, EE. UU., con filiales en San Antonio (Texas) y Boston (Massachusetts) en Estados Unidos. Su éxito reside en el envío de mensajes cortos llamados “tweets”.

### 3.3.1. Consecuencias reales en el mundo virtual

Con el tiempo twitter se ha utilizado por personajes públicos de gran peso en la sociedad con objetivos de marketing, propaganda, influencia, campaña política… Hasta llegar a tal punto en el que puede llegar a ser une enorme escandalo a nivel internacional sucesos como la expulsión de la cuenta de Donald Trump en este mismo 2021 [8]. Y como este se han llegado a dar numerosos escándalos a nivel internacional en esta plataforma [9].
Estos escándalos no son simples sucesos ajenos al usuario. Twitter incluso ha podido ser el canal por el que se han llegado a acarrear consecuencias penales a usuarios por el motivo de haber publicado un twitter el cual el gobierno no consideraba apto.
Por lo tanto, si existe el riesgo de que una entidad pueda enviarte a la cárcel o penalizarte de alguna forma por un “tweet” ¿En esta plataforma puedes expresarte de forma libre?

# 4. Análisis

El perfil de usuario al que se dirigirá esta aplicación será a personas ya con una mayoría de edad, suficientes adultas como para tener un contexto de lo que está sucediendo en las redes sociales actualmente, entre los 20 y 35 años. Es inevitable que en esta franja de edad los usuarios no fuesen veteranos debido a la basta cantidad de personas que usan las redes sociales, pero, aun así, un usuario que haya tenido malas experiencias en las otras redes sociales valorará más una red social como la de este proyecto. Por lo tanto, los usuarios hacia los que va dirigida esta aplicación serán veteranos y tener entre 20 y 35 años.

Los usuarios en esta aplicación solo podrán jugar un rol, todos con las mismas acciones. Definiremos los casos de uso de un usuario a continuación (Los Boards y Posts mencionados son entidades dentro de la aplicación que se definirán en el apartado 6.4 de este documento).

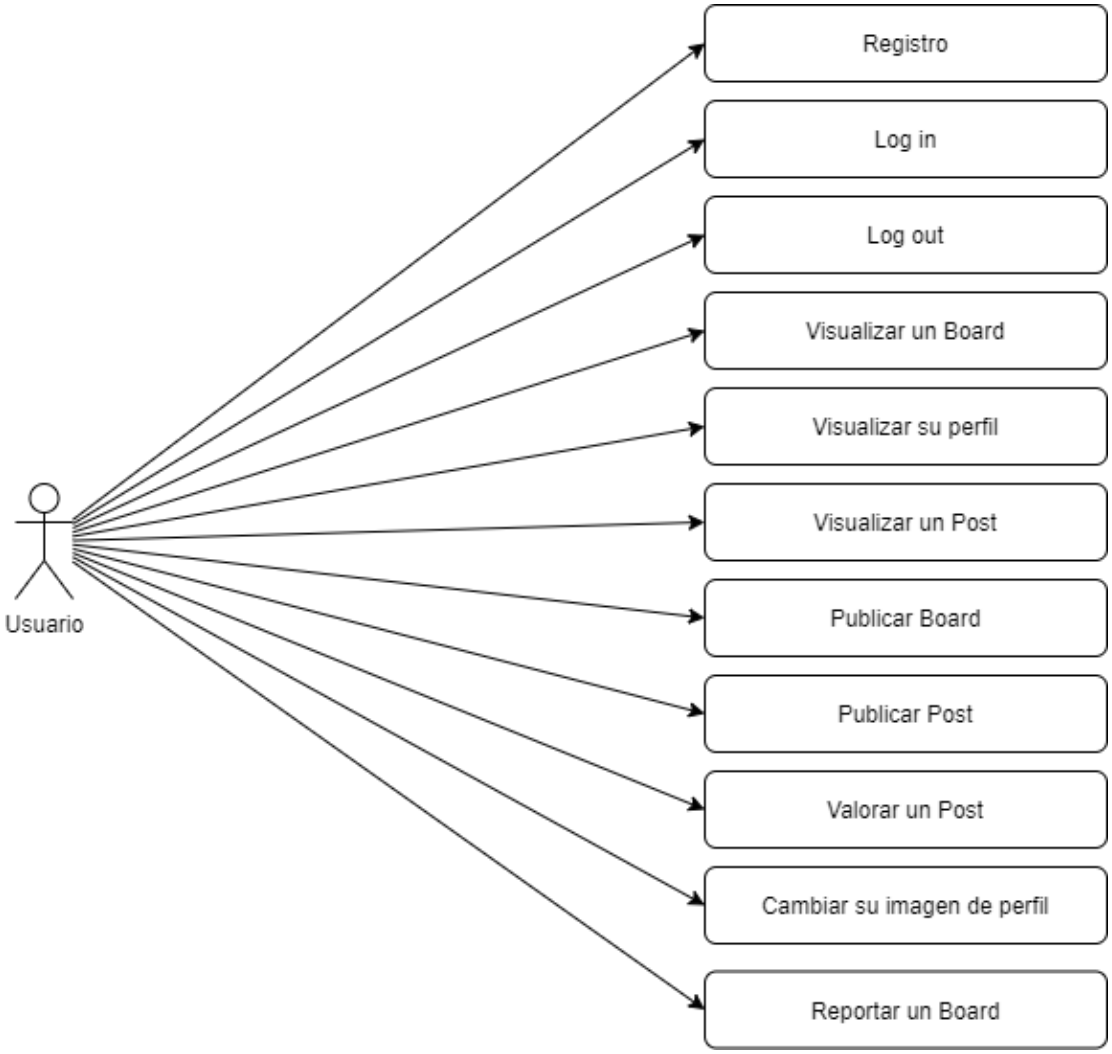



*Figura 1 Casos de uso de un usuario*

| UC1. Registro | |
|---|---|
| **Objetivo** | Registrarse en la plataforma. |
| **Actor** | Usuario. |
| **Precondiciones** | El usuario tiene instalada la aplicación. |
| **Secuencia** | 1. El usuario presiona el botón “I dont have an account…”<br>2. El usuario rellena el campo de nombre de usuario.<br>3. El usuario rellena el campo de email.<br>4. El usuario rellena el campo de contraseña.<br>5. El usuario presiona el botón de registro.<br>6. El sistema valida los campos y dirige al usuario a la pantalla principal de la aplicación. |
| **Extensiones** | 6.1. El nombre de usuario no es válido y la aplicación muestra el error pertinente manteniéndolo en la pantalla de registro.<br>6.2. El email no es válido y la aplicación muestra el error pertinente manteniéndolo en la pantalla de registro.<br>6.3. La contraseña no es válida y la aplicación muestra el error pertinente manteniéndolo en la pantalla de registro. |

| UC2. Log in | |
|---|---|
| **Objetivo** | Identificarse para entrar en la aplicación con una cuenta previamente registrada en el sistema. |
| **Actor** | Usuario. |
| **Precondiciones** | El usuario tiene instalada la aplicación.<br>El usuario ya ha realizado el registro previamente de una cuenta. |
| **Secuencia** | 1. El usuario rellena el campo de nombre de usuario.<br>2. El usuario rellena el campo de contraseña.<br>3. El usuario presiona el botón de log in.<br>4. El sistema valida los campos y dirige al usuario a la pantalla principal de la aplicación. |
| **Extensiones** | 4.1. El nombre de usuario no es válido y la aplicación muestra el error pertinente manteniéndolo en la pantalla de log in.<br>4.2. La contraseña no es válida y la aplicación muestra el error pertinente manteniéndolo en la pantalla de log in. |

| UC3. Log out | |
|---|---|
| **Objetivo** | Desconectarse de la aplicación desvinculándose con la identificación activa. |
| **Actor** | Usuario |
| **Precondiciones** | El usuario tiene instalada la aplicación.<br>El usuario ha hecho log in o registro previamente. |
| **Secuencia** | 1. El usuario se dirige a la opción donde visualizará su perfil presionando en botón de la barra de navegación correspondiente.<br>2. El sistema pide los datos del usuario al servicio y muestra la pantalla que dispone esta información.<br>3. El usuario presiona el botón de Log out.<br>4. El sistema borra los datos locales que contienen la información del usuario y muestra la pantalla de log in. |

| UC4. Visualizar un Board |
|---|

| | |
|---|---|
| **Objetivo** | Visualizar un Board dentro de la plataforma. |
| **Actor** | Usuario. |
| **Precondiciones** | El usuario tiene instalada la aplicación.<br>El usuario ha hecho log in o registro previamente. |
| **Secuencia** | 1. El usuario desde la pantalla principal toca cualquiera de los Boards que se presentan en pantalla.<br>2. El sistema pide los datos del Board seleccionado y muestra la pantalla que contiene la información del Board. |

| **UC5. Visualizar su Perfil** | |
|---|---|
| **Objetivo** | Visualizar la información del usuario. |
| **Actor** | Usuario |
| **Precondiciones** | El usuario tiene instalada la aplicación.<br>El usuario ha hecho log in o registro previamente. |
| **Secuencia** | 1. El usuario se dirige a la opción donde visualizará su perfil presionando en botón de la barra de navegación correspondiente.<br>2. El sistema pide los datos del usuario al servicio y muestra la pantalla que dispone esta información. |

| **UC6. Visualizar un Post** | |
|---|---|
| **Objetivo** | Visualizar un Post publicado en un Board. |
| **Actor** | Usuario |
| **Precondiciones** | El usuario tiene instalada la aplicación.<br>El usuario ha hecho log in o registro previamente. |
| **Secuencia** | 1. El usuario desde la pantalla principal toca cualquiera de los Boards que se presentan en pantalla.<br>2. El sistema pide los datos del Board seleccionado y muestra la pantalla que contiene la información del Board.<br>3. El usuario toca uno de los Posts que se muestran en la pantalla de la presentación del Board.<br>4. El sistema muestra por pantalla la información del Post seleccionado. |

| **UC7. Publicar un Board** | |
|---|---|
| **Objetivo** | Publicar un Board en la plataforma para su posterior visualización por los otros usuarios. |
| **Actor** | Usuario. |
| **Precondiciones** | El usuario tiene instalada la aplicación.<br>El usuario ha hecho log in o registro previamente. |
| **Secuencia** | 1. El usuario presiona el botón de la pantalla principal “Nuevo Board”.<br>2. El sistema presenta en pantalla el formulario a rellenar con la información necesaria para la creación del Board.<br>3. El usuario rellena el campo del nombre del Board.<br>4. El usuario rellena el campo de tags.<br>5. El usuario toca el botón “seleccionar archivo” del campo de icono del Board.<br>6. El sistema muestra la galería multimedia del dispositivo del usuario.<br>7. El usuario selecciona un archivo de la galería. |

| | |
|---|---|
| | 8. El sistema cierra la galería multimedia y muestra en el formulario el archivo seleccionado.<br>9. El usuario presiona el botón “Crear Board”<br>10. El sistema valida la información, guarda la información y muestra al usuario la pantalla de visualización del Board que acaba de crear. |
| **Extensiones** | 10.1. El campo de nombre de Board está vacío y el sistema muestra el error pertinente sin realizar la publicación y manteniendo en pantalla el formulario de creación de Board.<br>10.2. El campo de tags es invalido y el sistema muestra el error pertinente sin realizar la publicación y manteniendo en pantalla el formulario de creación de Board.<br>10.3. El archivo subido como icono del Board no es una imagen y el sistema muestra el error pertinente sin realizar la publicación y manteniendo en pantalla el formulario de creación de Board. |

| **UC8. Publicar un Post** | |
|---|---|
| **Objetivo** | Publicar un Post, dentro de un Board, en la plataforma para su posterior visualización por los otros usuarios. |
| **Actor** | Usuario |
| **Precondiciones** | El usuario tiene instalada la aplicación.<br>El usuario ha hecho log in o registro previamente. |
| **Secuencia** | 1. El usuario desde la pantalla principal toca cualquiera de los Boards que se presentan en pantalla.<br>2. El sistema pide los datos del Board seleccionado y muestra la pantalla que contiene la información del Board.<br>3. El usuario presiona el botón de “Nuevo Post”<br>4. El sistema muestra la galería multimedia del dispositivo del usuario.<br>5. El usuario selecciona un archivo de la galería.<br>6. El sistema cierra la galería multimedia y muestra una pantalla de previsualizado de la información del Post seleccionada.<br>7. El usuario presiona el botón “confirmar”.<br>8. El sistema cierra el previsualizado del Post, guarda los datos del Post y actualiza la pantalla del Board en la que se encuentra el usuario para poder ver el nuevo Post creado. |
| **Extensiones** | 7.1. El archivo seleccionado de la galería no es válido, el sistema cierra el formulario y muestra una pantalla de error. |

| **UC9. Valorar un Post** | |
|---|---|
| **Objetivo** | Añadir un valor positivo o negativo a un Post previamente publicado. |
| **Actor** | Usuario |
| **Precondiciones** | El usuario tiene instalada la aplicación.<br>El usuario ha hecho log in o registro previamente. |
| **Secuencia** | 1. El usuario desde la pantalla principal toca cualquiera de los Boards que se presentan en pantalla.<br>2. El sistema pide los datos del Board seleccionado y muestra la pantalla que contiene la información del Board.<br>3. El usuario toca uno de los Posts que se muestran en la pantalla de la presentación del Board. |

| | |
|---|---|
| | 4. El sistema muestra por pantalla la información del Post seleccionado.<br>5. El usuario presiona el botón para “valorar”.<br>6. El sistema ilumina el botón presionado y guarda la información. |
| **Extensiones** | 5.1. El usuario ya había valorado previamente esta publicación, por lo tanto el sistema no hará nada. |

| **UC10. Cambiar su imagen de perfil** | |
|---|---|
| **Objetivo** | Cambiar la imagen que muestra la pantalla del perfil de usuario. |
| **Actor** | Usuario |
| **Precondiciones** | El usuario tiene instalada la aplicación.<br>El usuario ha hecho log in o registro previamente. |
| **Secuencia** | 1. El usuario se dirige a la opción donde visualizará su perfil presionando en botón de la barra de navegación correspondiente.<br>2. El sistema pide los datos del usuario al servicio y muestra la pantalla que dispone esta información.<br>3. El usuario presiona el botón “cambiar foto de perfil”<br>4. El sistema muestra la galería multimedia del dispositivo del usuario.<br>5. El usuario selecciona un archivo de la galería.<br>6. El sistema cierra la galería multimedia y muestra el nuevo archivo seleccionado como imagen de perfil. |
| **Extensiones** | 6.1. El archivo seleccionado no valido y el sistema cierra la galería y muestra una ventana de error. |

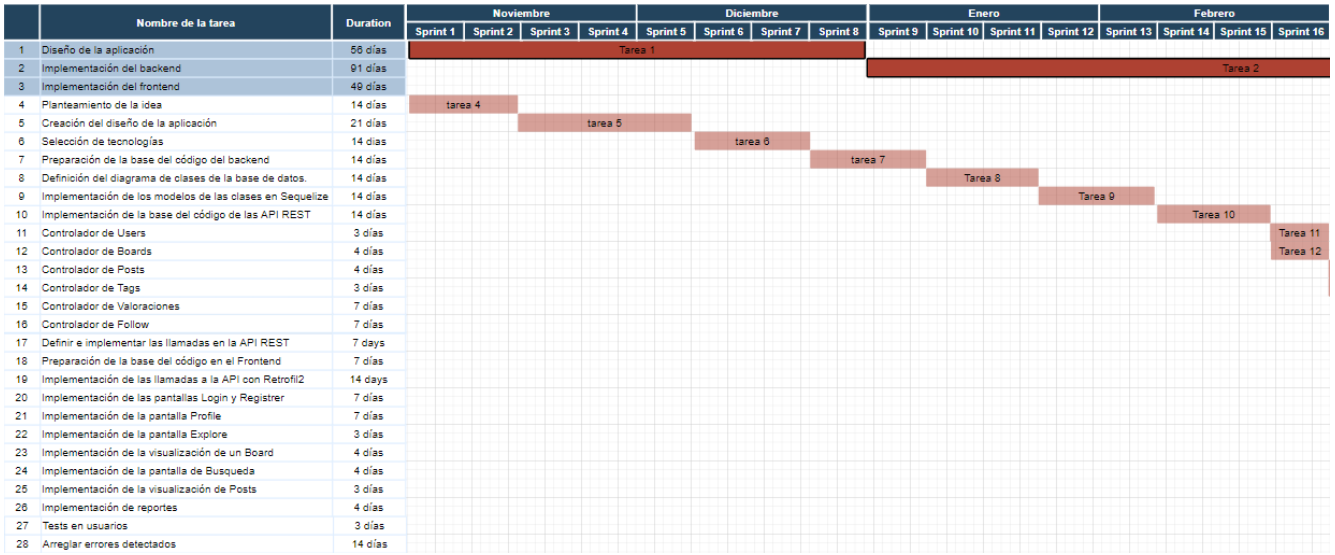
| **UC11. Reportar un Board** | |
|---|---|
| **Objetivo** | Enviar un reporte al sistema de un contenido inapropiado en un Board. |
| **Actor** | Usuario |
| **Precondiciones** | El usuario tiene instalada la aplicación.<br>El usuario ha hecho log in o registro previamente. |
| **Secuencia** | 1. El usuario desde la pantalla principal toca cualquiera de los Boards que se presentan en pantalla.<br>2. El sistema pide los datos del Board seleccionado y muestra la pantalla que contiene la información del Board.<br>3. El usuario presiona el botón “Reportar”<br>4. El sistema muestra un formulario de reporte en pantalla.<br>5. El usuario rellena el campo de texto del reporte.<br>6. El usuario presiona el botón “enviar”<br>7. En sistema cierra el formulario y muestra un mensaje confirmando que la información ha sido envíada. |

# 5. Metodología

## 5.1. Plan de trabajo efectuado

Para realizar un proyecto se debe organizar un flujo de trabajo con fechas de entrega y con la programación de todas las tareas a realizar. En el caso de este proyecto he decidido organizarlo según la metodología Scrum, una de las metodologías más conocidas en el mundo del desarrollo de aplicaciones con la cual, personalmente, estoy muy familiarizado.
En mi caso solo soy un trabajador, así que no he programado ninguna de las reuniones. Además, yo jugaré todos los roles a lo largo de la implementación del proyecto.

Los Sprints definidos han sido de una semana, para permitir más flexibilidad. A continuación, en la Figura 2 y Figura 3 podemos ver el diagrama de Gantt definido inicialmente.



| | Nombre de la tarea | Duration |
|---|---|---|
| 1 | Diseño de la aplicación | 56 días |
| 2 | Implementación del backend | 91 días |
| 3 | Implementación del frontend | 49 días |
| 4 | Planteamiento de la idea | 14 días |
| 5 | Creación del diseño de la aplicación | 21 días |
| 6 | Selección de tecnologías | 14 días |
| 7 | Preparación de la base del código del backend | 14 días |
| 8 | Definición del diagrama de clases de la base de datos. | 14 días |
| 9 | Implementación de los modelos de las clases en Sequelize | 14 días |
| 10 | Implementación de la base del código de las API REST | 14 días |
| 11 | Controlador de Users | 3 días |
| 12 | Controlador de Boards | 4 días |
| 13 | Controlador de Posts | 4 días |
| 14 | Controlador de Tags | 3 días |
| 15 | Controlador de Valoraciones | 7 días |
| 16 | Controlador de Follow | 7 días |
| 17 | Definir e implementar las llamadas en la API REST | 7 days |
| 18 | Preparación de la base del código en el Frontend | 7 días |
| 19 | Implementación de las llamadas a la API con Retrofil2 | 14 days |
| 20 | Implementación de las pantallas Login y Registrer | 7 días |
| 21 | Implementación de la pantalla Profile | 7 días |
| 22 | Implementación de la pantalla Explore | 3 días |
| 23 | Implementación de la visualización de un Board | 4 días |
| 24 | Implementación de la pantalla de Busqueda | 4 días |
| 25 | Implementación de la visualización de Posts | 3 días |
| 26 | Implementación de reportes | 4 días |
| 27 | Tests en usuarios | 3 días |
| 28 | Arreglar errores detectados | 14 días |



*Figura 2 Diagrama de Gantt inicial 1/2*

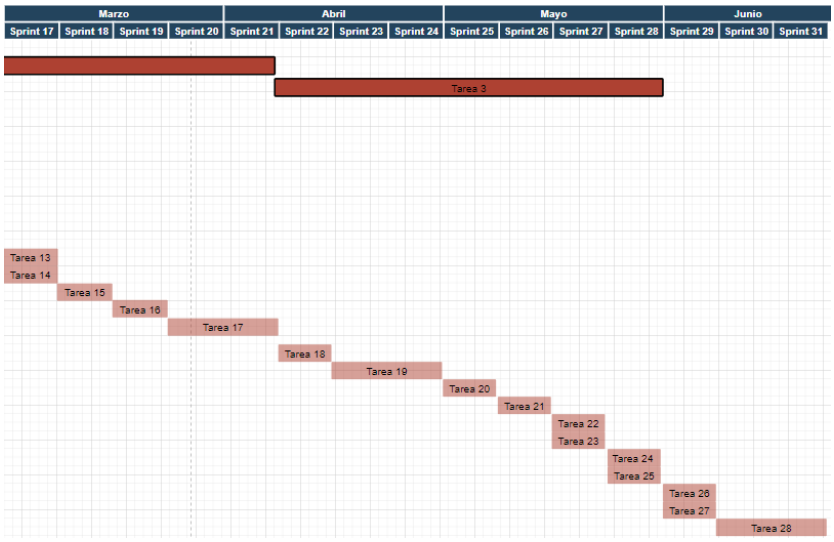
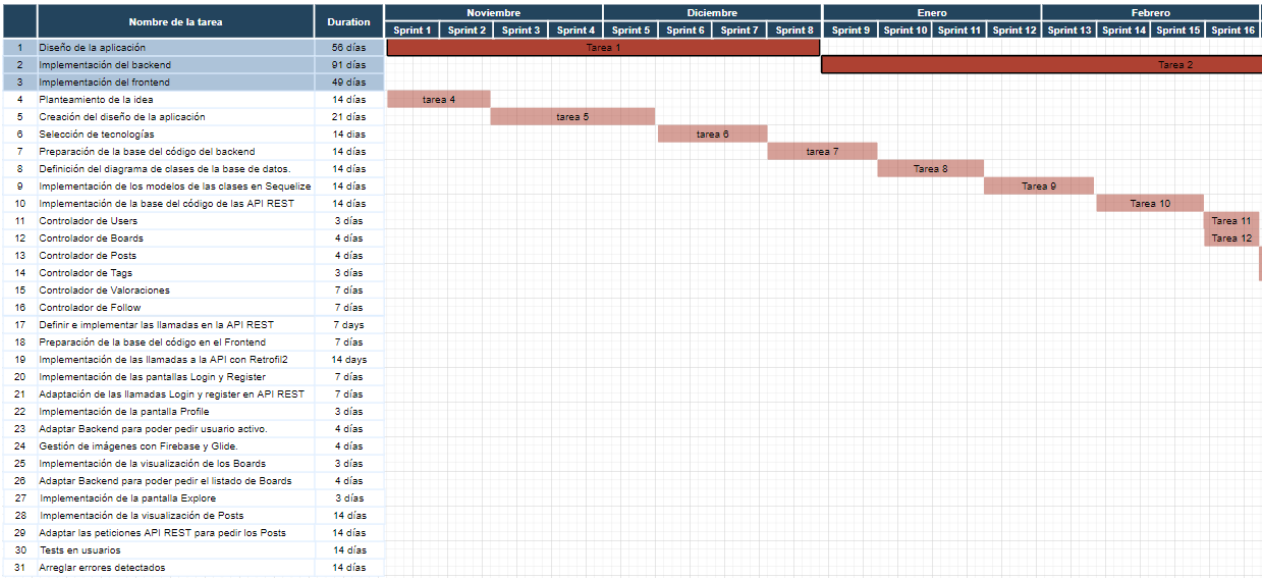




*Figura 3 Diagrama de Gantt inicial 2/2*

El proyecto inicialmente era suficiente para poder acabar con todas las tareas, pero debo decir que sobreestime la dificultad de la ejecución de muchas de ellas. Cuando llegué a la implementación del Frontend me di cuenta de que la implementación que hice del backend no era adaptable al frontend que estaba diseñando. Por lo tanto, tuve que estirar el periodo de tiempo con el que desarrollar las implementaciones de las distintas pantallas acortando el número de servicios que ofrecería la primera versión del MVP (Producto viable mínimo).
A partir de el sprint 25 tenía que trabajar juntamente con el Backend y el Frontend (Términos de los cuales definiremos en el siguiente apartado 5.2) para poder verificar su correcta funcionalidad. En un final, el diagrama de Gantt fue puede verse en la Figura 4 y Figura 5.



| | Nombre de la tarea | Duration |
|---|---|---|
| 1 | Diseño de la aplicación | 56 días |
| 2 | Implementación del backend | 91 días |
| 3 | Implementación del frontend | 49 días |
| 4 | Planteamiento de la idea | 14 días |
| 5 | Creación del diseño de la aplicación | 21 días |
| 6 | Selección de tecnologías | 14 días |
| 7 | Preparación de la base del código del backend | 14 días |
| 8 | Definición del diagrama de clases de la base de datos. | 14 días |
| 9 | Implementación de los modelos de las clases en Sequelize | 14 días |
| 10 | Implementación de la base del código de las API REST | 14 días |
| 11 | Controlador de Users | 3 días |
| 12 | Controlador de Boards | 4 días |
| 13 | Controlador de Posts | 4 días |
| 14 | Controlador de Tags | 3 días |
| 15 | Controlador de Valoraciones | 7 días |
| 16 | Controlador de Follow | 7 días |
| 17 | Definir e implementar las llamadas en la API REST | 7 days |
| 18 | Preparación de la base del código en el Frontend | 7 días |
| 19 | Implementación de las llamadas a la API con Retrofit2 | 14 days |
| 20 | Implementación de las pantallas Login y Register | 7 días |
| 21 | Adaptación de las llamadas Login y register en API REST | 7 días |
| 22 | Implementación de la pantalla Profile | 3 días |
| 23 | Adaptar Backend para poder pedir usuario activo. | 4 días |
| 24 | Gestión de imágenes con Firebase y Glide. | 4 días |
| 25 | Implementación de la visualización de los Boards | 3 días |
| 26 | Adaptar Backend para poder pedir el listado de Boards | 4 días |
| 27 | Implementación de la pantalla Explore | 3 días |
| 28 | Implementación de la visualización de Posts | 14 días |
| 29 | Adaptar las peticiones API REST para pedir los Posts | 14 días |
| 30 | Tests en usuarios | 14 días |
| 31 | Arreglar errores detectados | 14 días |



*Figura 4 Diagrama de Gantt final 1/2*

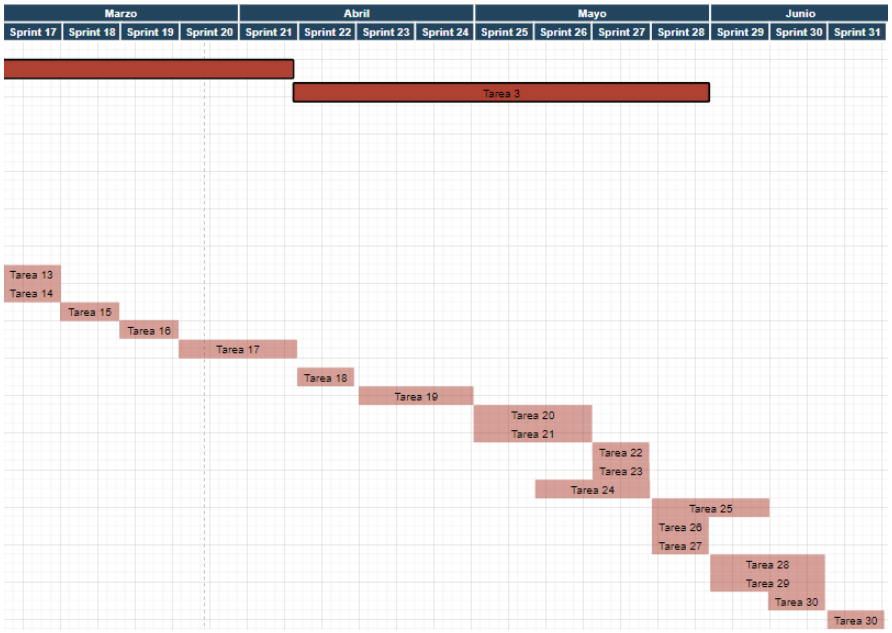




*Figura 5 Diagrama de Gantt final 2/2*

## 5.2. Estructura de la aplicación

### 5.2.1. Backend

Backend es la capa de acceso a datos de un software o cualquier dispositivo, que no es directamente accesible por los usuarios, además contiene la lógica de la aplicación que maneja dichos datos. El Backend también accede al servidor, que es una aplicación especializada que entiende la forma como el navegador solicita cosas.
He decidido que para llevar a cabo la implementación del backend para este proyecto una de las piezas indispensables será una API-REST.

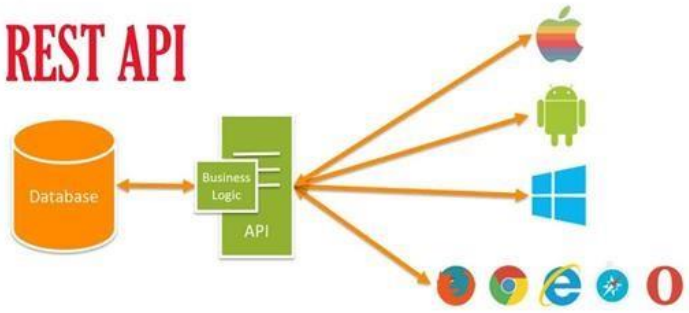



*Figura 6 REST API*

Un servicio de API REST, como vemos en la Figura 6, es un estilo de arquitectura de software que se utiliza para describir cualquier interfaz entre diferentes sistemas que utilice HTTP para comunicarse. REST significa Representational State Transfer (transferencia de estado representacional), lo que quiere decir que, entre dos llamadas cualquiera, el servicio no guarda los datos, esta responsabilidad queda delegada al backend, que será el que almacenará, buscará y actualizará los datos encontrados en la base de datos.
El cliente de una API REST puede ser una aplicación Android o iOS o un navegador web, incluso puede ser cualquier otro dispositivo o servicio como televisiones, Alexa o incluso un microondas. De esta forma con el mismo servicio de API REST implementado se podría reutilizar para implementar MyBoards en cualquier plataforma.
Una vez hemos hablado de la API REST podremos entender mejor cómo funciona la estructura de este proyecto, el backend de MyBoards está compuesto por tres módulos principales:

- Modelo: Se encargará de presentar la estructura de los datos, se definirán estructuras de datos que representarán las diferentes entidades que componen la base de datos.
- Controlador: Se encargará de gestionar los Modelos para llevar a cabo las diferentes funcionalidades que ofrecerán las llamadas API.
- API: Definiría las distintas peticiones que se pueden llevar a cabo desde el lado del Cliente y delegaría a los respectivos controladores para llevar a cabo cada funcionalidad.

De esta forma, la aplicación funcionaria tal que:

1. El cliente lanzaría una petición para recibir todos los Boards, esta petición la recogería la API.
2. La API tendría que enviarle al cliente un listado de Boards pero para eso le comunicará al controlador respectivo que le envíe esta información.
3. El controlador hará la petición de los modelos de Boards necesaria y los gestionará para transformarlo en los datos útiles que se requieren y entonces estos datos serían enviados de nuevo a la API para que esta los envíe hacia el cliente.

Para desarrollar el Backend también tenemos que decidir el Framework (Anexo A) que utilizaremos
En este proyecto he elegido utilizar el framework Node Express. Node Express (Figura 7) es un framework muy popular utilizado entre otros por Netflix, Paypal, Uber, Linkedin y Ebay. Este Framework está basado en la versión de Javascriot por la antonomasia para backend Node.js y nos ofrece una ligera cobertura de los aspectos más importantes para una aplicación sin eclipsar la potencia que nos ofrece Node. Las principales ventajas de Express que considero relativas para este proyecto serían:

- Velocidad (Ya se ejecuta en el motor JS de Google).
- Transmisión de datos (solicitudes y respuestas HTTP como un solo evento).
- Codificación fácil y rápida.
- Ciclos de desarrollo rápidos.
- Lógica de negocio en el servidor.


*Figura 7 Node Express*

### 5.2.2. Frontend

El Frontend es el lado del cliente, será nuestra aplicación para Android que mostrará la información y recogerá los eventos y los gestionara juntó al servidor (Backend) para llevar a cabo las funcionalidades implementadas que el usuario quiera llevar a cabo.
La arquitectura del proyecto de Frontend está estructurada usando el patrón MVVM, que se caracteriza por tratar de desacoplar lo máximo posible la interfaz de usuario de la lógica de la aplicación.

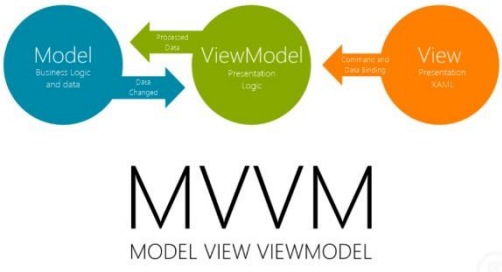



*Figura 8 Model-View-ViewModel*

El patrón MVVM se divide en tres capas:

- **View**: Únicamente la parte visual, en nuestro caso serán los ficheros XML de Android que definen los colores, la forma y la distribución de la vista entre otros atributos puramente estéticos que el usuario final verá reflejado en la pantalla.
- **ViewModel**: Es la encargada de interactuar entre el modelo y la vista, gestionará los eventos y estará constantemente atenta a cualquier cambió que deba producirse en la vista.
- **Model**: Dónde vamos a tener toda la lógica de datos que mostraremos finalmente en la View.

También habría que destacar que como se ve en la Figura 8, la View solo se comunica en una dirección, solo comunicará con el viewModel, observando datos y reportando eventos a este.
El ViewModel contiene los datos que la View presenta, la View únicamente los observa para que cuando haya algún cambio, este cambio también se produzca en la View, el ViewModel jamás le comunicara un cambio a la View, sino que es la View quien verá ese cambio. En el ViewModel podemos ver como este únicamente pedirá los datos al Model, el Model se los enviará y el ViewModel los guardará, entonces la View verá que este dato ha cambiado y mostrará el nuevo dato.

Otro aspecto muy importante que caracterizará a la implementación de este Frontend es el uso de los principios de Clean Architecture. Clean architecture es un conjunto de principios cuya finalidad principal es ocultar los detalles de la implementación a la lógica de la aplicación. Teniendo así la lógica aislada a lo que sería la implementación de la presentación de los datos, conseguimos tener una lógica mucho más mantenible y escalable en el tiempo.
En Clean Architecture, una aplicación se divide en responsabilidades y cada una de estas responsabilidades se representa en forma de capa. De esta forma tenemos capas exteriores y capas interiores:

- La capa más exterior representa los detalles de implementación
- Las capas más interiores representan el dominio incluyendo lógica de aplicación y lógica negocio empresarial.

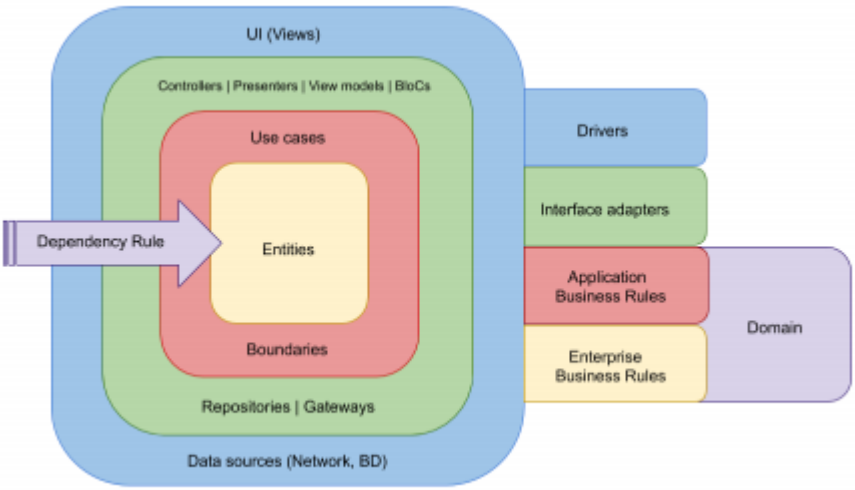



*Figura 9 Diagrama de Clean Architecture*

La regla de dependencia, visible en la Figura 9, nos dice que un círculo interior nunca debe conocer nada sobre un círculo exterior. Pero los círculos exteriores si pueden conocer círculos interiores, de esta manera conseguimos que las diferentes capas en esta arquitectura sean fácilmente reemplazables y flexibles. Las capas en las que se divide Clean Architecture son tres: Dominio, adaptadores y detalles de implementación.

**Dominio**
El dominio es el corazón de una aplicación y tiene que estar totalmente aislado de cualquier dependencia ajena a la lógica o los datos de negocio, entendiendo estos como los datos de las diferentes entidades que forman la aplicación.

- **Entidades**: Es el modelo que forma cada objeto en la aplicación. Por ejemplo: Podríamos tener una entidad silla, la cual definiríamos con un color, un material, un identificador del fabricante… De esta forma cuando vayamos a hacer cualquier topo de silla utilizaríamos esta entidad para definir la silla en particular.
- **Casos de uso**: Los casos de uso representan la lógica de aplicación, que existe principalmente debido a la automatización de procesos que se llevan a cabo para realizar las funciones solicitadas por el usuario.

**Adaptadores**
Los adaptadores se van a encargar de transformar la información como se entiende y es representada en los detalles de implementación o frameworks, drivers a como la entiende el dominio.
Es habitual utilizar en este punto junto con Clean Architecture diferentes patrones de presentación como MVC, MVP, MVVM o BloC donde los presenters, controllers, view models o blocs serían los adaptadores encargados de transformar la información de las vistas a información que necesitan los casos de uso. En el caso de este proyecto el “Model” mencionado anteriormente jugará el rol de adaptador.

**Detalles de implementación**
Los detalles de implementación en Clean Architecture son todas aquellas librerías que se suelen utilizar en una aplicación para toda aquella lógica paralela a nuestra aplicación.

## 5.3. Tecnologías utilizadas

Una vez hemos hablado del de la idea de MyBoards y sus motivaciones tendremos que hablar de como esto se va a llevar a cabo desde un punto de vista más técnico.
Y para llevar a cabo los aspectos tecnológicos de la aplicación tendremos que hablar de los dos hemisferios que compondrán la aplicación, en Backend y el Frontend.

### 5.3.1. Backend

#### *5.3.1.1. Base de datos*

Una base de datos es un conjunto de datos, si la redundancia da lugar a la duda, es la información necesaria para el modelo de negocio almacenada de forma ordenada para su futura recuperación y gestión.
En nuestro caso la base de datos se ha llevado a cabo en el sistema de gestión MySQL2: MySQL2 es un sistema de gestión de bases de datos, es decir, es un conjunto de programas

que permiten la almacenación, modificación y extracción de la información de una base de datos. En particular, MySQL2 es la continuación de MySQL-Native, es el resultado de haber creado el protocolo desde cero y la API se cambió para que coincidiese con el popular MySQL, por lo tanto, es completamente compatible con MySQL.
Ya que está basado en MySQL principalmente comparte sus características que le da ventajas que lo hacen muy interesante para los desarrolladores. La más evidente es que
trabaja con bases de datos relacionales, es decir, utiliza tablas múltiples que se interconectan entre sí para almacenar la información y organizarla correctamente. Con esta herramienta podremos construir una base de datos sólida e interconectada.

#### *5.3.1.2. Visual Studio code*

Visual Studio Code es un editor de código fuente desarrollado por Microsoft que soporta las plataformas Windos, Linux y MacOS. Incluye:

- Soporte para la depuración: proceso por el cual se facilita identificar y corregir errores de Programación.
- Control integrado de Git: Integra un servicio con el que poder mantener tu código en la nube con Git, un software de control de versiones.
- Resaltado de sintaxis: Apoyo visual para poder identificar distintos elementos que componen el código fuente (clases, métodos, variables…)
- Finalización inteligente de código: Puede autocompletar el código a medida que se va escribiendo.
- Refactorización de código: Herramienta para reestructurar el código alterando su estructura dejando intacta la funcionalidad.

Este ha sido la elección para desarrollar el código fuente del backend debido a que es gratuito, de código abierto y es el entorno en el que, personalmente, me siento más cómodo trabajando.

#### *5.3.1.3. Hat*

Es una herramienta para generar IDs aleatorias evitando las colisiones entre estas. Esta herramienta ha sido seleccionada por su apoyo en la comunidad, será usada para asignar una ID a cada objeto que se introduzca a la base de datos.

#### *5.3.1.4. Bcrypt*

En nuestra aplicación nos interesa tener las contraseñas encriptadas en nuestra base de datos, de esta forma si alguien pudiera acceder de alguna forma a la visualización de la base de datos, la contraseña del usuario sería inútil para el intruso. En busca de esta finalidad he recurrido a Bcrypt, una herramienta que en el pasado me ha sido muy útil, además de ser compatible con el entorno de Node tiene un gran apoyo por la comunidad.
Es una librería que ayuda para encriptar las contraseñas que utilizaremos para los usuarios. Básicamente una función Bcrypt se define con la siguiente cadena:


*$2b$[cost]$[22 character salt][31 character hash]*

- **2a:** Identifica el algoritmo de Bcrypt utilizado.

- **[cost]:** Es el factor de coste, se utilizan 2^10 iteraciones de la función de derivación de claves, es decir, el algoritmo utilizado por la librería es utilizado 2^10 veces.
- **[22 Character salt]:** Son datos aleatorios que se utilizan como una entrada adicional a una función unidireccional que codifica datos. En este caso de 22 caracteres (16-byte)
- **[33 Character salt]:** Igual que el anterior, pero de 31 caracteres (24-byte)

Un ejemplo de una cadena encriptada de esta forma sería:

$2a$10$N9qo8uLOickgx2ZMRZoMyeIjZAgcfl7p92ldGxad68LJZdL17lhWy

#### *5.3.1.5. Sequelize* 

Cuando hacemos algún desarrollo del lado del backend una de las tareas que tenemos que realizar es manipular la base de datos (Insertar, buscar, actualizar y borrar), para esto generalmente se escribe directamente la consulta SQL en el lenguaje de programación y así conseguir los datos, pero para hacer este proceso más cómodo y optimizado he utilizado Sequelize. Sequelize es un ORM (Object-Relational mapping) que nos permite convertir tablas de una base de datos en entidades en un lenguaje de programación orientado a objetos, lo cual agiliza bastante el acceso a estos datos. De esta forma conseguiremos que los objetos que manipulamos de nuestro modelo estén directamente contactados con su respectiva tabla en la base de datos. Hablaremos de esta herramienta más adelante, en la implementación.

#### *5.3.1.6. Heroku* 

Heroku es una plataforma como servicio (PaaS), una plataforma que permite lanzar programas en la nube que fue desarrollada en 2007 soportando únicamente Ruby. Posteriormente Heroku amplío su soporte a otras tecnologías, entre estas se encuentra Node.js, que es el entorno en el que trabajaremos para desarrollar nuestro Backend. Heroku es necesario en este proyecto debido a que necesitamos que el programa de nuestro Backend esté corriendo en la nube para poder ser accesible desde la aplicación móvil en todo momento.
Además, utilizaremos el plugin: ClearDB MySQL, una base de datos como servicio que contendrá nuestra base de datos. En particular, Heroku ha sido seleccionado para este proyecto porque es la única plataforma como servicio que se ha encontrado que soporta Node.js de forma gratuita.

### 5.3.2. Frontend

#### *5.3.2.1. Android studio* 

Android Studio es el entorno de desarrollo integrado (IDE) oficial para la plataforma Android. Fue anunciado el 16 de mayo de 2013 en la conferencia Google I/O, y reemplazó a Eclipse como el IDE oficial para el desarrollo de aplicaciones para Android. Entre las características que incluye esta IDE destacaría:

- Soporte para construcción basada en Gradle: Utiliza Gradle para automatizar la construcción del código y transformarlo en una aplicación ejecutable.
- Refactorización: Visto ya en Visual Studio Code en el apartado 5.3.1.2, pero, en este caso este proceso está orientado al desarrollo de aplicaciones móviles en el entorno de Android.

- Herramientas Lint: Herramienta para detectar problemas de rendimiento, usabilidad, combatividad de versiones y otros problemas.
- Plantillas para crear diseños comunes de Android y otros componentes.
- Un editor de diseño interactivo, permite al usuario arrastrar y soltar componentes de la interfaz del usuario.
- Soporte integrado para Google Cloud Platform: Esto nos permitirá la integración de Firebase que veremos a continuación.
- Renderizado a tiempo real: Podemos ver al momento los cambios en la parte visual de la aplicación.
- Simulación de dispositivos: Permite ejecutar y probar aplicaciones en un entorno virtual, sin necesidad de dispositivo físico. Esto permite probar distintos tamaños de pantalla y distintos aspectos de hardware entre otros aspectos físicos de los dispositivos.

También encontramos IDES para el desarrollo de aplicaciones Android como NetBeans, Intellij, Aide o Eclipse, pero Android Studio esta únicamente orientada al desarrollo de aplicaciones Android.
El editor de diseño interactivo, la simulación de dispositivos y el renderizado a tiempo real son facilidades a la hora del desarrollo de una aplicación móvil que hacen que Android Studio sea mucho más cómoda frente al resto de IDES. Por lo tanto, Android Studio será la IDE que se usa para el desarrollo de esta aplicación.

#### *5.3.2.2. Kotlin* 

Kotlin es el lenguaje de programación que se ha utilizado en la implementación del Frontend de proyecto. Es un lenguaje de programación de tipado estético, esto quiere decir que la comprobación de tipificación (el correcto nombramiento de los elementos del código) se realizará durante la compilación y no durante la ejecución.
Este lenguaje de programación corre sobre la máquina virtual de Java, esto da la ventaja de que el código Kotlin puede ser convertido a código en Java y viceversa, son perfectamente compatibles.
En Android Studio, hoy en día, Kotlin es la opción preferida para la programación de aplicaciones móvil. Gran parte de las librerías, entre ellas las desarrolladas por Google, están escritas en Kotlin y es por eso (Además de mi declinación personal por este lenguaje de programación) por lo que este ha sido el lenguaje de programación escogido.

#### *5.3.2.3. Dagger hilt*

Es la herramienta que nos ayuda a respetar la regla de dependencia de Clean Architecture, es una herramienta de inyección de dependencias.
La inyección de dependencias es suministrar objetos a una clase en lugar de ser la propia clase la que cree dichos objetos. Esos objetos cumplen contratos que necesitan nuestras clases para poder funcionar (de ahí el concepto de dependencia).
Conociendo ya lo que es una inyección de dependencias podemos entender que llevar un objeto de una clase a otra puede ser una tarea complicada dependiendo de la estructura de nuestra aplicación.
Por eso utilizamos Dagger Hilt, Dagger Hilt está basado en Dagger el cual te libera libera de escribir código estándar para hacer una inyección de dependencias, lo que es tedioso y propenso a errores.

En Dagger Hilt tu declaras una clase Provider. La clase Provider se encargará de hacer una instancia de esa propia clase y proporcionarlo a la clase desde la cual hagas una llamada a esta clase Provider, desde cualquier clase de la aplicación.
Además, una entre muchas de las funcionalidades de Dagger Hilt, puedes declarar el Provider como Singleton (Que solo existiría una instancia de esta clase) o LazySingleton (Lo mismo que el Singleton, pero esta instancia se crea si no se había creado antes en el caso que requieras cualquier atributo o método de esta instancia) con solo una notación.

#### *5.3.2.4. Retrofit2*

Retrofit2 es un intermediario seguro para clientes REST desarrollado por Square, nos permitirá hacer consultas a la API-REST que hemos definido en nuestro backend desde nuestra app de Android.
La librería proviene un poderoso framework para autentificar e interactuar con las API enviando peticiones con OKHttp.
Cuando hacemos una petición recibiremos la respuesta en formato JSON y hacernos así más fácil el proceso para transformar esta respuesta en el objeto que deseemos cuando estemos trabajando en la capa de Model de nuestro MVVM

#### *5.3.2.5. Firebase*

Firebase es una plataforma para el desarrollo de aplicaciones web y aplicaciones móviles lanzada en 2011 y adquirida por Google en 2014 la cual ofrece una serie de herramientas multiplataforma.
En la implementación de nuestra aplicación solo hemos usado esta tecnología para aprovecharnos de una de las herramientas de esta plataforma:

- **Firebase Storage**

Esta tecnología es un servicio de almacenamiento de objetos construido para el escalamiento de Google. Con esta tecnología podemos almacenar imágenes, audios, videos y cualquier tipo de contenido generado por el usuario a la base de datos de Firebase.
Este servicio ideológicamente debería darse por una base de datos propia. En el caso de esta aplicación, debemos tener en cuenta que nuestro Backend está usando Heroku (Apartado 5.3.1.6) y como comentamos se está utilizando ClearDB MySQL para contener la base de datos. El problema es que esta base de datos es únicamente para la base de datos de MySQL, no puede contener las Imágenes, que es algo necesario para llevar a cabo está aplicación. Por eso Firebase Storage juega un papel muy importante en esta aplicación.
Además, también tiene agregado la seguridad de Google en las interacciones con esta tecnología. Eso quiere decir que a petición de Google tenemos que autentificar nuestras peticiones al servicio de Firebase Storage por medio de la herramienta Firebase Auth. Es por eso que la aplicación también contiene Firebase Auth, pero no se identificará con sus datos personales, sino que se ha configurado el servicio para permitir llamadas anónimas al servicio.

#### *5.3.2.6. Glide*

Glide es la herramienta para aplicaciones móvil más conocida para cargar imágenes sobre los recursos visuales de Android. Estas imágenes pueden ser tanto archivos como URL.

Esta herramienta es muy rápida, muy eficiente y fácil de utilizar, además, permite hacer modificaciones de presentación de las imágenes.
La base de datos de la aplicación contendrá únicamente la URL de las imágenes guardadas en Firebase, por lo tanto, esta tecnología nos permite presentar las imágenes desde la base de datos de Firebase de una forma muy rápida y cómoda.

#### *5.3.2.7. Jetpack*

Android Jetpack es un conjunto de bibliotecas que ayudan a reducir el código estándar y así poder realizar funciones que de otra forma llevarían a implementar un codigo repetitivo además de utilizar tecnologías que cada vez se están quedando anticuadas en el desarrollo de Android. En especial, este conjunto de Bibliotecas ha sido añadido para la hacer uso de dos de sus librerías:

- **Navigation:**

Navigation es un framework que facilita el hecho de navegar entre “destinos” dentro de una aplicación de Android. Estos destinos pueden ser cualquier tipo de componente de Android que represente una vista. Además, esta librería influye una herramienta llamada **Safe Args,** la cual facilitar el tráfico de información entre los componentes conectados por medio de Navigation.

- **Databinding:**

DataBinding es una librería que vincula los componentes de la vista de las aplicaciones Android en tus diseños con las fuentes de datos de tu app en un formato declarativo, es decir, desde el código asignado al componente de la vista podremos referenciar directamente los subcomponentes definidos en esta.

# 6. Diseño

## 6.1. Diagrama de clases de la aplicación

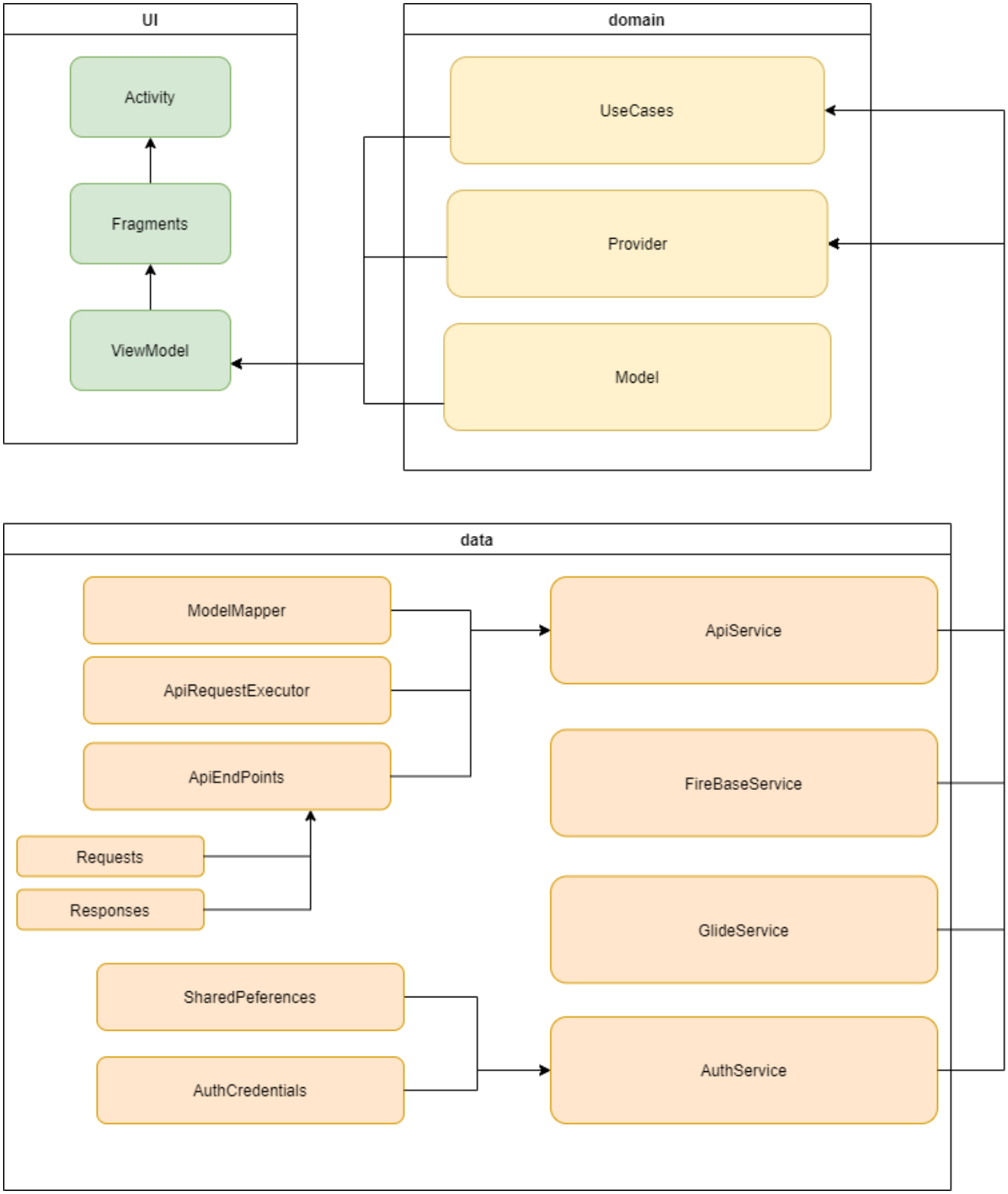



*Figura 10 Diagrama de clases de la aplicación*

El diseño realizado para la implementación de la aplicación del proyecto puede verse de una forma simplificada en la Figura 10, para ver la información completa del diagrama de clases de la aplicación ver el Anexo E. Podemos ver en el diagrama de la Figura 10 tres grandes subgrupos.

UI
La UI es la interfaz, los elementos que estarán directamente relacionados con las interacciones con el usuario y presentación de la información. Las Activity y Fragments son elementos visuales de Android que contienen la lógica de la navegación por la aplicación, las interacciones y la presentación de la información. Los ViewModel serán los componentes que irán ligados a cada Fragment, serán los que contendrán y solicitarán la información que presentarán los otros dos elementos de la UI.

**Domain**
La capa que se encargará de proveer a la vista de los servicios y modelos de datos necesarios para la realización de las acciones que demande el usuario será Domain. Al único que proveerá de los servicios será al ViewModel pero los modelos definidos en Model serán utilizados a lo largo del proyecto.
Existirá un UseCase por acción que pueda realizar el usuario, este use case estará directamente con el servicio cenesario para llevar a cabo esa acción.
Model contiene las estructuras de datos que se utilizarán a lo largo del proyecto.
Se hablará en profundidad del Provider en los detalles de la implementación (Apartado 7.2.4). El Provider será el que proveerá de cualquier servicio a los elementos de la UI respetando la regla de la inyección de dependencias de Clean Architecture, el cual hemos comentado en el

**Data**
Data contiene los servicios del proyecto, todo aquello que requiera una implementación que no esté relacionada con la vista
ApiService realizará las comunicaciones con la API REST que definiremos en el proyecto. Podemos ver que está utilizando varios componentes para su realización. ApiEndPoints será el fragmento de código junto con Requests y Responses que conocerá las rutas y métodos de la API REST además de los campos necesarios para cada petición y la forma de la respuesta. Por otro lado, tenemos el ApiRequestExecutor que contendrá la lógica necesaria para lanzar las peticiones una vez construidas contra el servicio de API REST. Una vez tengamos la respuesta de una petición previamente lanzada el ModelMapper transformará la información recibida en una información que podamos utilizar en nuestro proyecto, esta información será la que tenemos definida en los modelos definidos previamente en Model.
FireBaseService realizará las interacciones con el servicio de Firebase (Apartado 5.3.2.5).
Hemos hablado anteriormente de Glide (Apartado 5.3.2.6), GlideService contendrá el código para hacer funcionar esta herramienta correctamente.
AuthService se encargará de guardar la información localmente para la gestión del usuario. AuthCredentials será la estructura de datos en la que se guardará a la información del usuario y se utilizará esta estructura para guardar estos datos en SharedPreferences.

## 6.2. Reglas de diseño

Antes de realizar el diseño se deben de definir unas reglas sobre las que trabajar para lograr una aplicación que aporte una experiencia positiva a los usuarios.
Para empezar, comenzaremos definiendo cual será la orientación de la pantalla en la aplicación:
El contenido principal de la aplicación serán los Boards, su símil en la realidad sería un tablón de corcho, como si se tratase de un tablón de anuncios o un tablón personal en el que poner recuerdos a la vista. Por lo tanto, esta idea sugeriría que la orientación del móvil fuese Horizontal.
Pero tenemos que entender el entorno en el que vamos a desarrollar la aplicación. Por lo general los usuarios utilizan el móvil de una forma perezosa, ellos prefieren una aplicación que les resulte cómoda a la hora de navegar que algo estéticamente correcto.
Pues ya se ha estudiado el caso de la orientación de la pantalla [10] en los cuales se ha concluido que el 94% del tiempo que los usuarios usan el móvil lo hacen verticalmente, por lo tanto, sería ir en contra corriente si decidiéramos optar por hacerlo en horizontal, así que la aplicación se diseñará en vertical.
Existiría la posibilidad de hacer un diseño para uso vertical y otro para el uso horizontal, pero para la versión de este proyecto se considera que se hará solo en una orientación debido a que realizar dos diseños de la aplicación llevaría un trabajo demasiado extenso para una primera versión de la aplicación.
Del estudio previo también podemos extraer cual es la postura preferida de los usuarios, visible en la Figura 11, esto también deberá tenerse en cuenta. Una vez definida la orientación definiremos unos valores, unas reglas, en las que se basará el diseño para este proyecto.

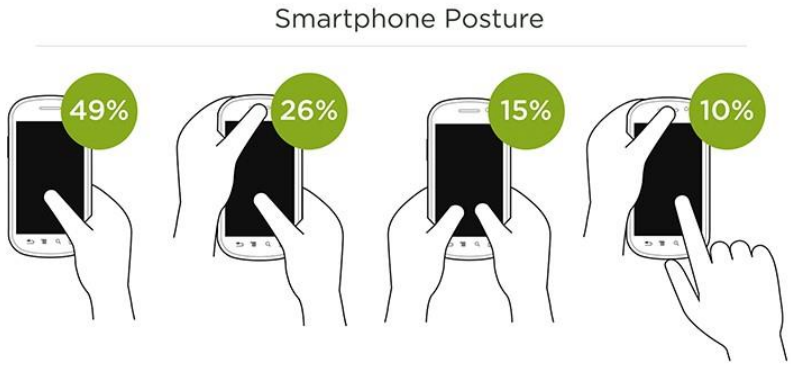



*Figura 11 Posturas en un dispositivo móvil*

## 6.3. Navegación simple

Los usuarios de móvil no siempre tienen el tiempo para buscar lo que necesitan en una aplicación, cuan más fácil sea para el usuario estar donde quieren estar mejor valorarán la experiencia con la aplicación. A continuación, analizaremos tres factores principales a tener en cuenta para realizar una navegación simple.

*1 - Infraestuctura de nivel superior*

La infraestructura de nivel superior es el cómo gestiona nuestra aplicación que la navegación por la aplicación permita lleva al usuario al punto que el desea de una forma fácil e intuitiva. Hay muchas formas de estructurar la aplicación, he decidido llevar a cabo la más común para diseños orientados a aplicaciones móviles: Nested Doll Navigation.
Tenemos que comprender que la navegación de nuestra aplicación se entenderá como un árbol dividido en jerarquías, cada vez que accedemos a una sección dentro de otra sección entenderemos la primera como una subsección que se encuentra por debajo de la sección de la que emerge.
Y emerger es de lo que va Nested Doll Navigation (Figura 12), cada vez que descendemos en este árbol de jerarquías veremos que esta pantalla por encima de la anterior y cuando queramos navegar a una jerarquía superior veremos como la pantalla se encontraba por debajo de la anterior, como si de la típica muñeca rusa, matriosca, se tratase.

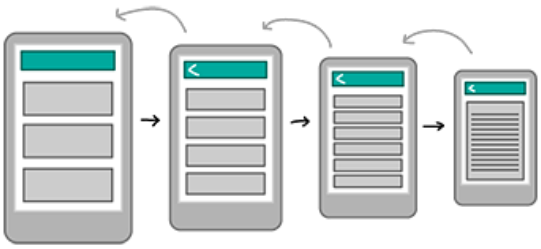


*Figura 12 Nested Doll Navigation*

*2 - Señales locales*

Las señales locales están ahí para orientar al usuario en el espacio en el que se encuentra. Estas señales son:

- Logo o botón home.
- Cajas de búsqueda.
- Sección headers/banners (Figura 13)
- Sección de herramientas de navegación (Figura 14)
- Desplegables/galerias/calendarios/etc.

A parte también debemos controlar bien los márgenes de la pantalla para indicar al usuario donde debe y donde no debe tocar.
Es mejor utilizar estas señales con moderación para que cuando el usuario las encuentre esté seguro de lo que significan, si el usuario llegase a encontrarse muchos, podría llevarle al conflicto.

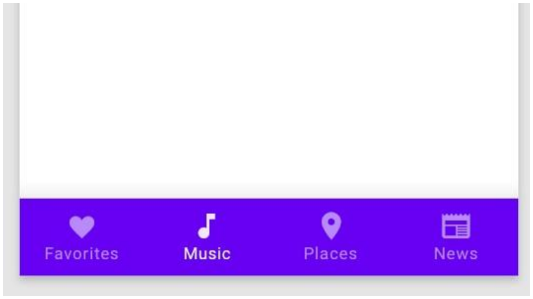



*Figura 14 Herramientas de navegación*

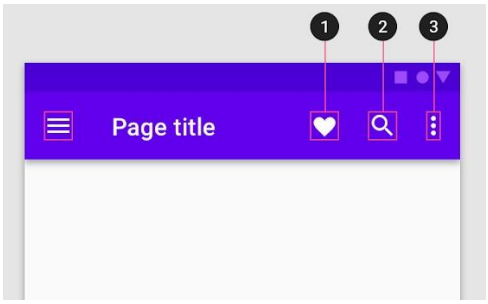



*Figura 13 Header*

*3- Guías*

Las guías son las señales visuales que ayudan al usuario con la toma de decisiones, se tiene que dar importancia a el uso de las dos siguientes guías:

- **Iconos**: Los íconos deben, siempre que sea posible, ajustarse al modelo mental del usuario de ese ícono, suelen estar ya estandarizados. Los iconos que no cumplen los criterios comprendidos suelen dar lugar a confusiones, por lo tanto, los iconos deben ser lo más intuitivos posibles y encontrarse en un contexto que les dé sentido.

- **Etiquetas**: Las etiquetas dan un valor explicativo a las tomas de decisiones determinantes para el usuario. Las imágenes y texto juntos son muy descriptivos, pero por su cuenta, un texto es más descriptivo que una imagen.

Una vez conocemos estos 3 factores que definirán el comportamiento de la aplicación debemos tener en cuenta otros aspectos que facilitan al usuario el uso de la aplicación y mejoran la experiencia en su uso:

- **Áreas de toque grandes:**

Simplemente debemos tener en cuenta que cuan más grandes sean las áreas en la que se puede tocar un botón mayor será la facilidad en las que el usuario podrá emplearlos.

- **Disposición de información necesaria:**

En las distintas pantallas de la aplicación, nunca que tenemos que dar información que sea inútil para el usuario, hay que organizar las presentaciones para conseguir una distribución de la información desenfadada y espaciada.

- **Textos grandes:**

Los textos a lo largo de la aplicación tienen que ser lo más grande posible respetando los espaciados y márgenes, no queremos cansar la vista del usuario.

- **Uso de controles apropiados:**

Debemos usar los controles apropiados a cada contexto, debemos tener en cuenta que, por ejemplo, un ordenador tiene un teclado y un ratón para facilitar su uso y por lo tanto las webs están diseñadas de otra forma que un móvil no puede estarlo, en un móvil disponemos únicamente de una pantalla táctil y utilizar el teclado obliga al usuario a usar las dos manos. Por lo tanto, debemos usar las herramientas necesarias para disponer al usuario de toda comodidad.
Por ejemplo: Para poner una fecha, en un ordenador se hace uso del teclado porque resulta cómodo para el usuario, pero usar el teclado en el caso del diseño en móvil para rellenar una fecha exige más del usuario, lo correcto sería un menú rotativo.

- **Posición del pulgar:**

Se han hecho numerosos artículos en los que se habla sobre el comportamiento de los usuarios en móvil y uno de los casos más comentados es “La zona del pulgar”. Se muestra en la Figura 15 la zona en la pantalla que requiere de un esfuerzo del usuario para llegar con su pulgar (Zona naranja y roja) y le fuerza a adoptar otra posición, por esto interesa que los elementos que requieren de un acceso más frecuente o de un acceso necesario para la

navegación de la aplicación se encuentren en la zona más accesible para el pulgar (zona verde).

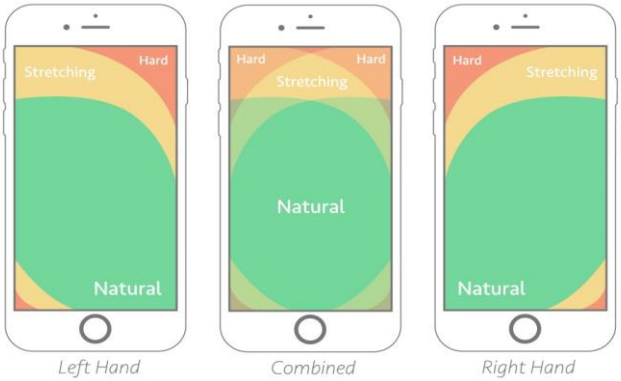



*Figura 15 Posición del pulgar en dispositivo móvil*

## 6.4. Componentes principales

### 6.4.1. Post

Un Post (Figura 16) es una publicación de un usuario, para la primera versión de la aplicación solo contendrán imágenes y se podrán valorar con un “Up” o un “Down” que son el apoyo positivo o negativo de los usuarios, respectivamente. La información que componen los Posts será:

- Imagen: La imagen que contiene dicho Post.
- Apoyo: Todos los Posts tienen un valor numérico que refleja el apoyo de la comunidad a este contenido, este es proporcional al número de Up y Down recibido y puede ser negativo en caso de que los Down sean más que los Up o positivo si el caso es el contrario.

Por supuesto un Post no mostrará nunca a los usuarios quien ha sido el autor, y lo mismo podemos decir de los Up y Down de cada post. Cuanto mayor sea el balance positivo entre los “up” y “down” de un Post más visibilidad se le otorgara.


*Figura 16 Visualización de un Post*

### 6.4.2. Board

Los Boards (Figura 17) se mostrará como si de un tablón se tratase y sobre estos estarán colocados de forma aleatoria los Post, superponiéndose unos a otros dependiendo del balance positivo que ostenten, los Posts que queden tapados por el resto y lleguen a un balance positivo que quede por debajo de un número determinado de Posts de este Board serán eliminados. La información que componen los Boards será:

- *Apoyo: Todos los Boards, de la misma forma que los Posts, tienen un valor numérico que refleja el apoyo de la comunidad, a diferencia que, en el Post, este apoyo es proporcional al número de Up y Down recibido en todos los Posts que contiene el Board y, como en el Post, puede ser negativo en caso de que los Down sean más que los Up o positivo si el caso es el contrario*
- *Icono: Los Boards tienen un icono que es una imagen que se utiliza para dar refuerzo a la temática del Board.*
- *Título: Es el enunciado del Board.*
- *Tags: Los tags serán palabras o compuestos de palabras que ayudarán tanto a dar refuerzo a la temática del Board como para clasificarlo y ayudar en la exploración de estos.*
- *Posts: Se verán los posts distribuidos de forma aleatoria junto al apoyo de cada uno de ellos y la imagen del contenido de este en forma de miniatura.*

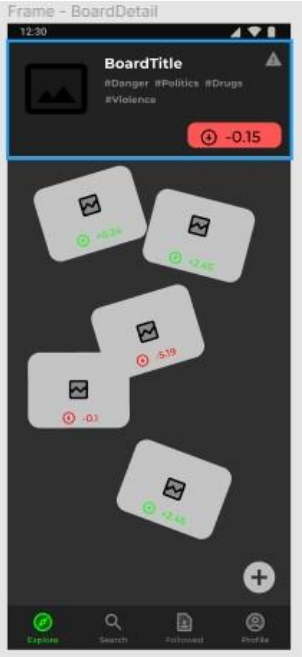



*Figura 17 Visualización de un Board*

No se mostrará el autor de un Board, a menos que un usuario siga al autor de ese Board, en este caso este autor si será visible para el usuario.
El listado de Posts que recibirá cada Board es limitado, digamos que esa limitación es de 50 Posts, si un Post llega a una valoración que lo sitúa en la posición 50, este no será mostrado y después de un tiempo será eliminado de la base de datos.
El usuario estará limitado a añadir un Post a cada Board cada día, de esta forma evitaremos el problema de que pueda llegar a crear demasiados Posts en un Board y así tapar todo el contenido.

### 6.4.3. Profile

Un Profile (Figura 18) es la información del usuario a la cual solo podrá acceder este mismo desde la barra de navegación de la aplicación. La información a la cual se tendrá acceso:

- Icono: Una imagen seleccionada por el usuario.
- Nombre: El nombre del usuario.
- Apoyo: Es el valor numérico que representa el resultado del cálculo pertinente que tiene en cuenta los votos positivos y negativos recibidos en sus Boards creados y en sus Posts publicados.
- Boards: Se mostrarán junto a su apoyo un listado de los Boards creados por el usuario en cuestión.

El Profile no será visible para nadie que no conozca el nombre completo, en la búsqueda, un usuario puede buscar a otro usuario introduciendo su nombre exacto y decidir seguirle para así ver los Boards que ha subido dicho usuario.

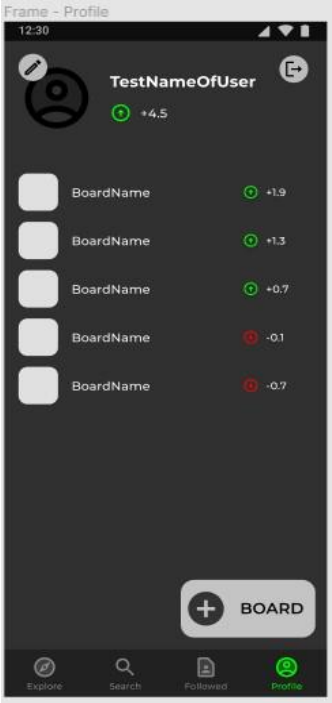



*Figura 18 Visualización de Profile*

## 6.5. Flujo de la aplicación

El flujo de la aplicación está definido como los pasos que el usuario da y lo que verá a lo largo que interactúa con tu aplicación, teniendo en cuenta todas las opciones que este podrá tomar y mostrando a cada paso lo que este se encontrará.
En mi caso he utilizado la plataforma Figma para realizar mi diseño, el cual es público y se puede ver en el siguiente enlace [12].
En este documento trataremos caso a caso los flujos de las distintas funcionalidades de las que proviene la aplicación.

### 6.5.1. Login y register

Es una disposición muy simple, en la parte superior podemos ver el título de la aplicación aislado del resto de objetos de esta pantalla para darle más protagonismo.
Podemos ver que, en el centro de ambas pantallas, tenemos el formulario para el registro, al tocar en cualquier campo se nos abrirá el teclado para rellenarlos y cuando hayamos acabado daremos al botón del formulario pertinente, el cual se muestra con icono y etiqueta, respetando la norma de las Guías explicado en el apartado 6.3.
Cuando haya un error en el proceso de inicio o registro de sesión: El mensaje de error será mostrado debajo del título de la ventana Login o Register como se puede ver en la ventana de Login de la Figura 19.
En la parte inferior de la pantalla de Login podemos ver el botón para acceder a la pantalla de Register.

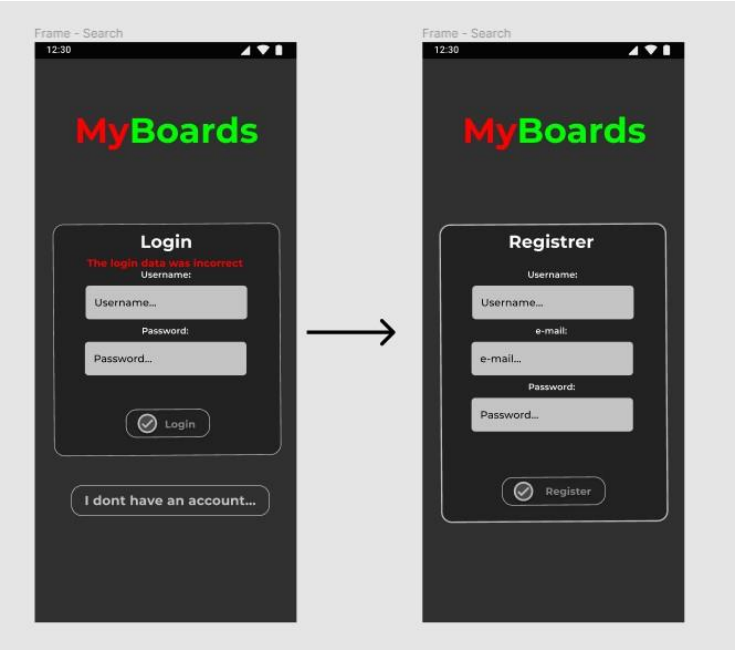



*Figura 19 Flujo de Login y Register*

### 6.5.2. Search

El campo de búsqueda (Figura 20) no sigue las reglas estipuladas en el diseño, esto ha sido debido a que, cuando el teclado se abra por abajo cuando el usuario presione en el campo de texto se desea que pueda ver el máximo de objetos encontrados a medida que busca. Si el campo de búsqueda se coloca en la parte inferior de la pantalla este quedaría tapado por el

teclado.  Esta ha sido una decisión que se ha tomado después de revisar el diseño de Instagram.

En la parte inferior del campo de búsqueda vemos la selección de tres opciones representadas con iconos descriptivos, que son: Boards, perfiles y hashtags. El seleccionado queda resaltado con el color principal de la aplicación y será el tipo de contenido que se buscará y se mostrará en el contenedor inferior.
En el diseño completo se puede ver que este menú también puede llevar a un Board si esta toca sobre cualquier Board encontrado por medio de la búsqueda por hashtag o por listado, pero en la imagen de la se puede ver el caso en el que se toque a un Profile, aparecerá por encima la ventana de dialogo que dará la opción de seguir y hará una breve presentación del perfil en cuestión. Ya en esta pantalla podemos ver en la parte inferior la barra de navegación, la cual siempre será visible y mantendrá la opción seleccionada con el color principal de la aplicación.

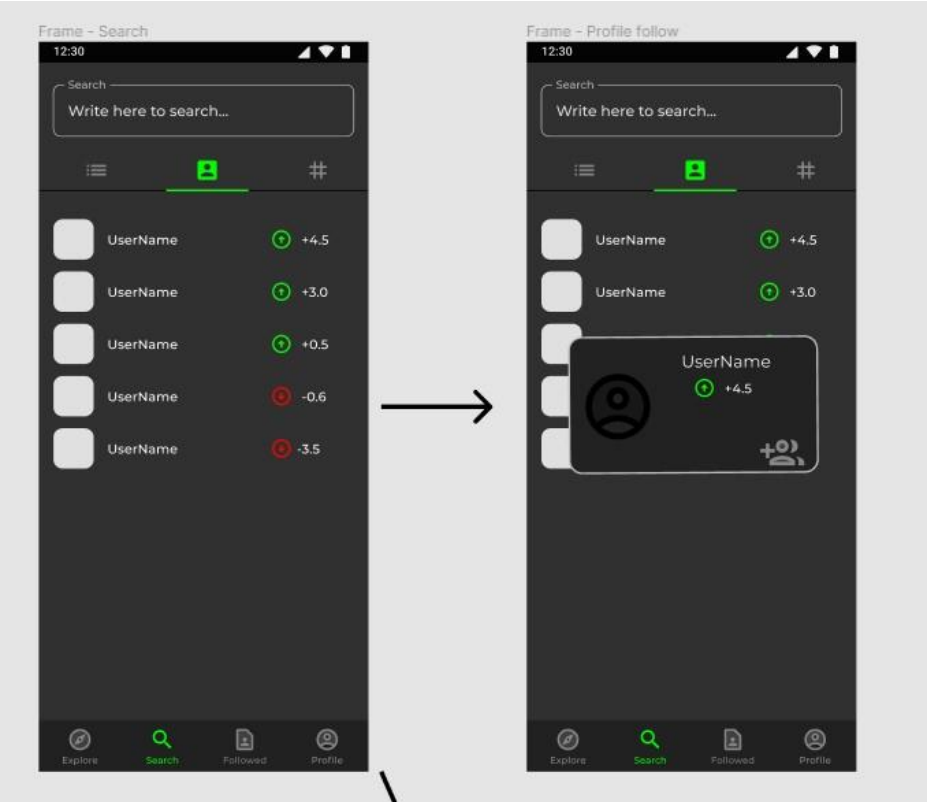



*Figura 20 Flujo de opción Search*

### 6.5.3. Explore

En la Figura 21, en el “Frame-Explore”, podemos ver que hay una disposición de filas separadas por tags de varios Boards, cada uno con su icono como protagonista, con su título en la parte inferior de este. En el extremo inferior derecho tenemos la opción de añadir un Board (veremos en el caso de Profile que es lo que sucede en este punto) está vez, para poder diferenciarlo de la opción de añadir un Post se compone de un botón que cuenta con icono y etiqueta.
En el caso de acceder a un Board veremos la pantalla del Board, el “Frame-BoardDetail” de la Figura 21, con la información que describimos en la descripción de componentes distribuida en un encabezado que separa el contenido principal de la información descriptiva del post.

Podemos ver en la esquina inferior, en la derecha, encontramos el botón para añadir un Post al Board.

Acciones desde un Board:

- Crear un Post: Abrir la ventana correspondiente que facilitará el proceso de añadir un Post al contenido del Board.
- Reportar: Si un usuario no considera oportuno de alguna forma el contenido del Board puede reportarlo para su consecuente inspección (A continuación, en el punto 5.2.3.4 veremos como sucede esto).
- Cerrar: Cerrar el Board y volver al punto de la aplicación en la que se encontraba previamente con el botón back de Android.

Si accedemos a cualquier Post tocándolo se superpone visualización del Post manteniendo la información del cabecero del Board en pantalla para evitar que el usuario pierda de vista la temática del contenido (Frame-PostDetail en Figura 21).

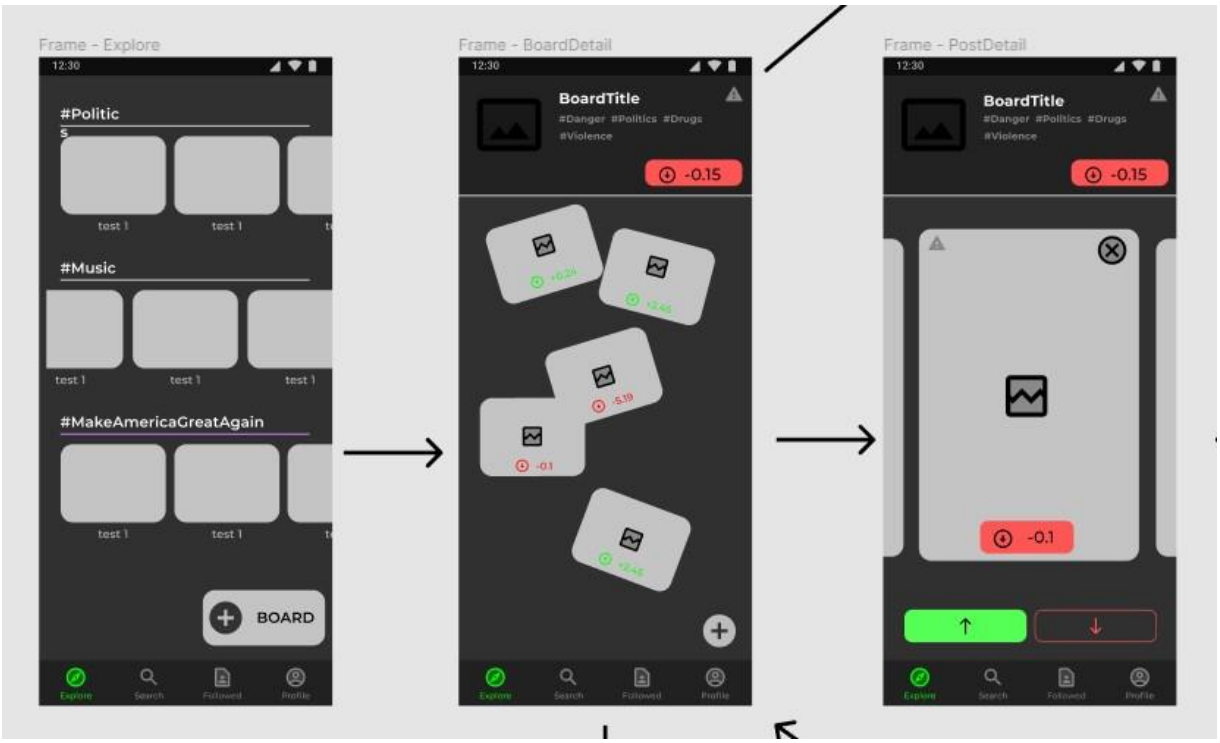



*Figura 21 Flujo de opción Explore, visualización de Board y Post*

También a destacar que como se puede apreciar, en la vista del Post podremos ver también un pequeño fragmento de otros Posts para dar a entender al usuario que si desplaza el dedo podrá pasar a ver el siguiente Post. Acciones desde un Post:

- Up: Dar un voto positivo al Post.
- Down: Dar un voto negativo al Post.
- Reportar: Si un usuario no considera oportuno de alguna forma el contenido del Post puede reportarlo para su siguiente inspección.
- Navegar: Si el usuario desplaza el dedo hacia la derecha podrá intercambiar la vista del Post actual por la del consecuente o el anterior.

- Cerrar: Cerrar la visualización del Post y volver al respectivo Board con el botón back de Android.

En el caso de presionar un Up o un Down podemos ver que el color resalta como el fondo del botón en caso de estar seleccionado para dar a entender al usuario cuan ha sido su opción.

### 6.5.4. Report

Tanto como cuando accedemos a un reporte desde el Board o desde el Post se nos abrirá la ventana pertinente por encima del contenido en el cual nos encontremos como vemos en la Figura 22. Se trata de un campo de texto que pide el motivo del reporte, encabezado por un título y un icono para poner en contexto al usuario y al extremo inferior podemos ver el botón para enviar el reporte.


*Figura 22 Visualización de ventana de Report*

### 6.5.5. Creación de un post

Desde cualquier visualización de un Board, como vemos en la Figura 23, veremos que existe la opción de presionar el botón del extremo inferior a la derecha de la pantalla, una vez presionado se nos abrirá una ventana por encima del contenido el cual nos hará seleccionar el origen del contenido que deseamos subir, sea cual sea la decisión que tome el usuario, se manejará con la herramienta nativa de Android.
Haremos captura o seleccionaremos desde la galería y entonces veremos una previsualización de la imagen antes de aceptar con el botón del extremo inferior de la ventana para publicar

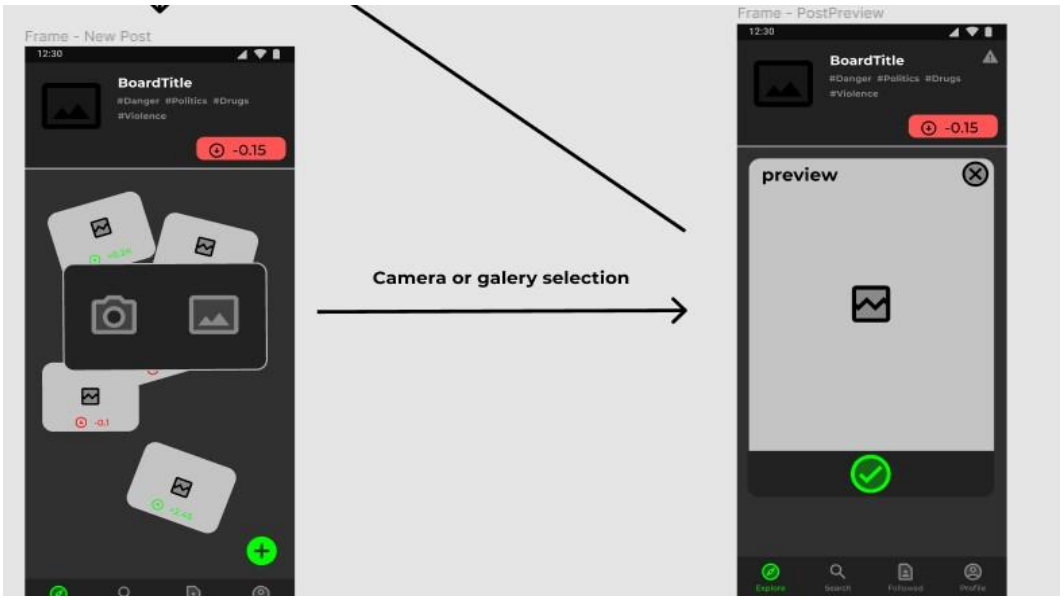



*Figura 23 Flujo de creación de un Post*

este Post en una posición aleatoria en el Board y entonces esta ventana se cerrará mostrando de nuevo la visualización del Board.

### 6.5.6. Profile y Creación de Boards

En la parte superior de “Frame-Profile” de Figura 24 podemos ver el cabecero, que se dispone de un icono, que podrá ser editado por el usuario presionando el botón que se encuentra en la parte superior de este. También podemos ver el nombre del usuario y justo debajo la puntuación un botón compuesto únicamente de un icono que corresponde a la acción de Logout.
En el contenedor inferior al cabecero veremos un listado de los Boards que este usuario ha creado y sus valoraciones medias.

Acciones desde Profile:

- Crear un Board: Nos abrirá la ventana de dialogo que podemos ver en “Frame-New Board”. Por medio de este dialogo podremos crear un nuevo Board.
- Logout: Presionando el botón Logout situado en el extremo superior a la derecha de la pantalla de Profile, podremos desvincular la aplicación de nuestra cuenta volviendo a la pantalla de Login vista en el apartado 6.5.1.
- Editar icono: Se seleccionará una imagen de la galería para que esta sea la nueva imagen visible en el icono de Profile.

Una vez es visible el dialogo “Frame-New Board” de la Figura 24, el usuario verá el formulario a rellenar con la información del Board.

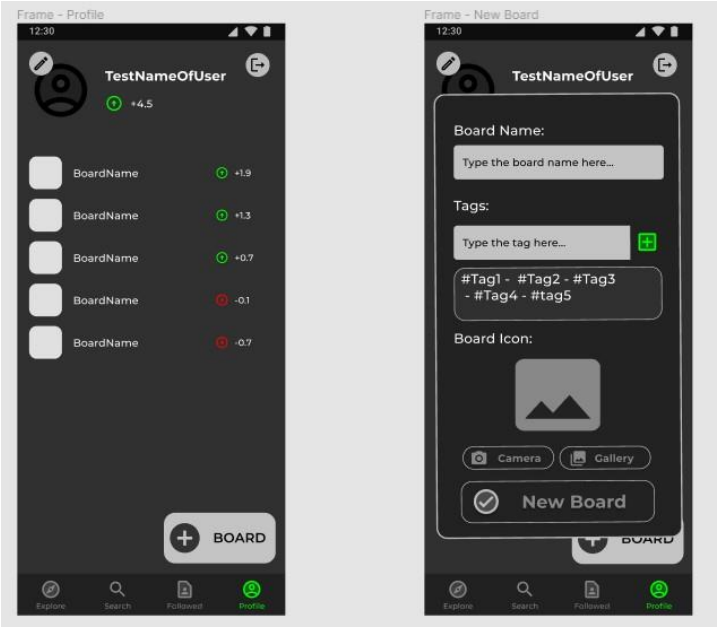



*Figura 24 Flujo de creación de un Board desde Profile*

El Formulario contiene cuatro campos fácilmente identificables:

- Nombre del Board: Es un campo de texto con una etiqueta descriptiva en la parte superior. Este será el nombre del nuevo Board. Si al intentar crear el nuevo Board este texto está vacío se mostrará un testo rojo encima del campo del texto indicando el error.

- Tags: Vemos la misma composición que en el nombre del Board pero en este caso tenemos un contenedor con un texto en la parte inferior y un botón. Este botón será para añadir el tag que el usuario esté escribiendo en el cuadro de texto y lo añadirá al contenedor de la parte inferior. El botón no hará su función y se mostrará el error de la misma forma que en el nombre del Board en el caso en el que el texto tenga algún espacio o haya más de cinco Tags.
- Icono del Board: Vemos como protagonista una imagen predefinida y en su parte inferior dos botones, el botón de galería y de cámara. Con los botones se accederá a la galería o a la cámara para seleccionar o capturar la imagen que sería el icono del Board y será mostrada en lugar de la imagen actual encima de los botones.
- Botón “New Board”: Con este botón se recogerá la información rellenada por el usuario. No es necesario añadir tags o seleccionar un icono, pero si lo será añadir un título. Una vez presionado este botón dirigirá la vista a la visualización del nuevo Board vista en el apartado 6.5.3.

### 6.5.7. Followed

Podemos ver en la Figura 25 que esta vista tendrá únicamente un contenedor de los Boards de las personas que seguimos ordenados por fecha de creación. Estos Boards se presentan en esta imagen con su icono como principal recurso visual con el título del Board en la parte inferior.  Está presentación los Boards se distribuirán de una forma distinta en esta pantalla, además esta presentación tiene la particularidad de que podremos ver el icono del creador del Board en la parte superior a la izquierda de cada uno de los Boards. Si tocamos cualquier Board del listado iremos a la visualización del Board seleccionado de la misma forma explicada en el apartado 6.5.3.

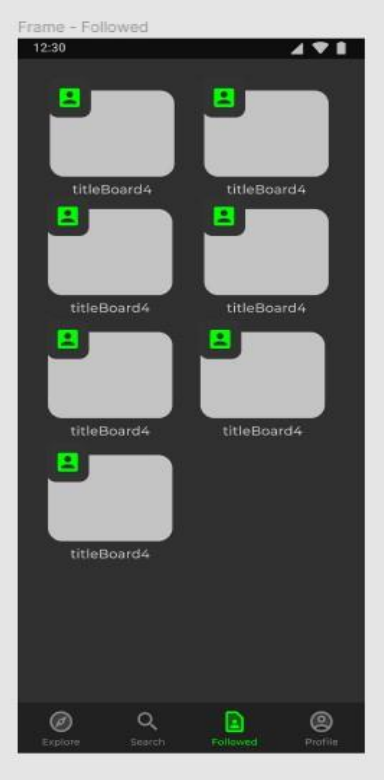



*Figura 25 Visualización de la función Followed*

# 7. Implementación

## 7.1. Definición del proyecto

La idea principal de MyBoards es dar el protagonismo y la responsabilidad al contenido, en esta plataforma podrás crear “Boards”. Los “Boards” se presentan como un tablón de corcho en el que el usuario puede poner fotos o cualquier cosa en formato 2D. Dichos objetos 2D se llaman “Posts”. Los Posts se distribuirán de forma aleatoria sobre el Board. Los Posts no mostrarán de ninguna forma al autor. Todo el contenido es anónimo frente a la comunidad y es el contenido el que será sujeto de aprobación de los demás usuarios. Cuanto mayor sea el apoyo de la comunidad a cada elemento de cada tablón, mayor será su visibilidad.
Cada uno de los Posts que pongamos en un tablón se verán por encima del resto durante un breve periodo de tiempo esperando la valoración de los usuarios. Cada Board tendrá una valoración que se entenderá como la exigencia de calidad del contenido del Board. Por lo tanto, si mi Post durante este periodo no logra llegar a esa valoración, o recibe una cantidad de valoraciones negativas que superen un límite, este post comenzará a quedar por detrás del resto. Dicho límite será proporcional a la valoración del Board.
De esta forma logramos que el contenido sea valorado y elegido por la comunidad, pero como hemos dicho el perfil del usuario será algo ajeno a su contenido.

MyBoards protegerá al usuario limitando a un acceso directo la relación entre este y los otros usuarios. Se acotará la búsqueda de otros perfiles a una búsqueda por nombre exacto de usuario y sin existir una comunicación directa entre ellos.
Es posible seguir el contenido de amigos, pero no directamente. Si yo conozco el nombre exacto de otro usuario podré buscarle y seguirle, una vez el haya aprobado esta relación yo podría ver sus Boards pero nunca sabré cuales han sido sus Posts. De esta forma podemos hacer que los amigos se ayuden entre si a llenar sus Boards, pero nunca se sabrá quién es el responsable del contenido, por lo tanto, esto sigue protegiendo a cada usuario con el manto del anonimato. Además, estos Boards jamás podrán ser privados, todo contenido siempre está expuesto a la opinión pública.
Partiendo de estas reglas creamos un entorno en el que no existan “influencers”, personas con millones de seguidores apoyando un contenido idealizado, de vidas perfectas que puede hacer al resto sentirse inferiores. No existirá ese espejo falso con el que comparar tu vida con la de otros. Es un entorno en el que la política, las grandes empresas, los famosos, los poderosos tendrán la misma fuerza para decidir que cualquier otra persona en esta plataforma.
Este proyecto busca solucionar los problemas vistos en otras redes sociales no con medidas sobre un sistema a modo de parches, si no, partiendo de un proyecto de red social que se construye en base a lo aprendido en la experiencia con las redes sociales actuales. Si tenemos en cuenta los problemas planteados en antecedentes (Apartado 3):

- **Efectos Psicológicos**

Esta plataforma no incentiva al usuario a conseguir un gran número de seguidores, sí que en esta plataforma los usuarios reciben un valor positivo o negativo en función de las valoraciones recibidas por la comunidad, pero estas valoraciones no reflejan un número de personas de la comunidad que apoyan este contenido, este número sería una proporción de la valoración positiva y negativa, sin dar más valor a un gran número de usuarios.
Con esto quedaría eliminada la figura de el “influencer”, de esta forma sería imposible en una plataforma como esta que los usuarios sintiesen esa necesidad de crear una vida virtual publica e idílica que podría influenciar la vida del resto de usuarios.

Además, la eliminación de la comunicación directa es un factor que ayudará mucho en el caso del bullyng.

- **Organizaciones**

Al no existir una comunicación directa entre los usuarios, como ya hemos comentado, no podrían darse casos como el caso de las unidades terroristas captando nuevos adeptos en Facebook.
Además, la forma que tiene MyBoards de presentar el contenido haría que el contenido de cualquier tipo de organización quedase expuesta a toda la comunidad, en esta aplicación no existe el contenido privado, por lo tanto, cualquier caso detectado de terrorismo o de cualquier actividad ilícita podría ser fácilmente detectada, reportada y eliminada.

- **Impacto político**

No se podrían dar casos como el que afectó a las elecciones de estados unidos por medio de las herramientas de Facebook (El caso de Cambridge Analytica), ya que la visualización del contenido está en las manos de los usuarios, cada uno de ellos tiene la libre decisión de apoyar y visualizar el contenido que ellos deseen, la responsabilidad y el poder caen en las manos de cada individuo.
Se han podido ver en muchos casos como la visibilidad de los partidos políticos es algo muy importante, y seguramente en cualquier plataforma que permita cualquier tipo de comunicación va a haber contenido de carácter político. Pero en esta aplicación se expondría este contenido a una valoración más cruda y real, me atrevería a decir, de la comunidad.

- **Consecuencias reales en el mundo virtual**

Otro de los problemas presentados anteriormente son las consecuencias políticas que llevarían a penalizaciones por terceros, ya sean gobiernos, empresas o cualquier tipo de entidad. Este problema no existiría en esta aplicación, ya que no tendrían acceso a la persona que ha publicado el contenido.
La anonimidad protege a cada uno del contenido, si un partido político o cualquier otra entidad no está contenta con el contenido que hay relacionado con ellos, no les queda otra cosa que entender que esta valoración no tiene nombres que perseguir y que quizás esta opinión generalizada tenga algo de real y debería tenerse en cuenta.

## Conclusión

Con todo esto concluyo en que la solución que presenta MyBoards no está basada en algoritmos de control, ni medidas y políticas dentro de la plataforma que pretendan regular el comportamiento dentro de esta. La solución a estos problemas en MyBoards está en que las capacidades de los usuarios dentro de este sistema, no exista la posibilidad de llevar a cabo estas prácticas.
Quizás frente una plataforma más transparente e intocable, las grandes entidades quizás se replanteen la naturaleza de sus actos y podemos hacer de este un mundo un poco mejor.

## 7.2. Detalles de la implementación

En este apartado se verán los detalles más relevantes del diseño, arquitectura y tecnología que se han llevado a la practica en el proceso de creación de esta aplicación.

### 7.2.1. Base de datos

Se trata de una base de datos en el lenguaje MySQL, como hablamos anteriormente en el apartado 5.3.1.1.
Las distintas tablas que componen la base de datos están relacionadas por medio de claves foráneas. Estas claves hacen referencia a otra columna id de una tabla ajena, de esta forma podemos conocer, por ejemplo, con que Board está relacionado un Post. El Post contendría la id del Board. La id de cada uno de los elementos se generará de forma automática.
En el proyecto del backend, existe una carpeta llamada “models” que contiene los datos que componen cada uno de estos objetos que forman la base de datos. Los modelos mencionados, están construidos con Sequelize, tecnología de la que hablamos en el Apartado 5.3.1.5.

Gracias a Sequelize, a la hora de realizar búsquedas sobre la base de datos, no tendríamos que hacerlo como se haría naturalmente en una base de datos (Escribiendo una “query”). Con esta tecnología simplemente tenemos que hacer referencia al modelo pertinente de la tabla sobre la que queramos actuar y realizar cualquiera de las llamadas que facilita Sequelize.
Realizando las llamadas estandarizadas de Sequelize el código queda mejor estructurado y además, podemos aprovecharnos de la seguridad que nos otorga este Framework evitando problemas como la conocida inyección SQL. Podemos ver la documentación de la base de datos a continuación, en la Figura 26.

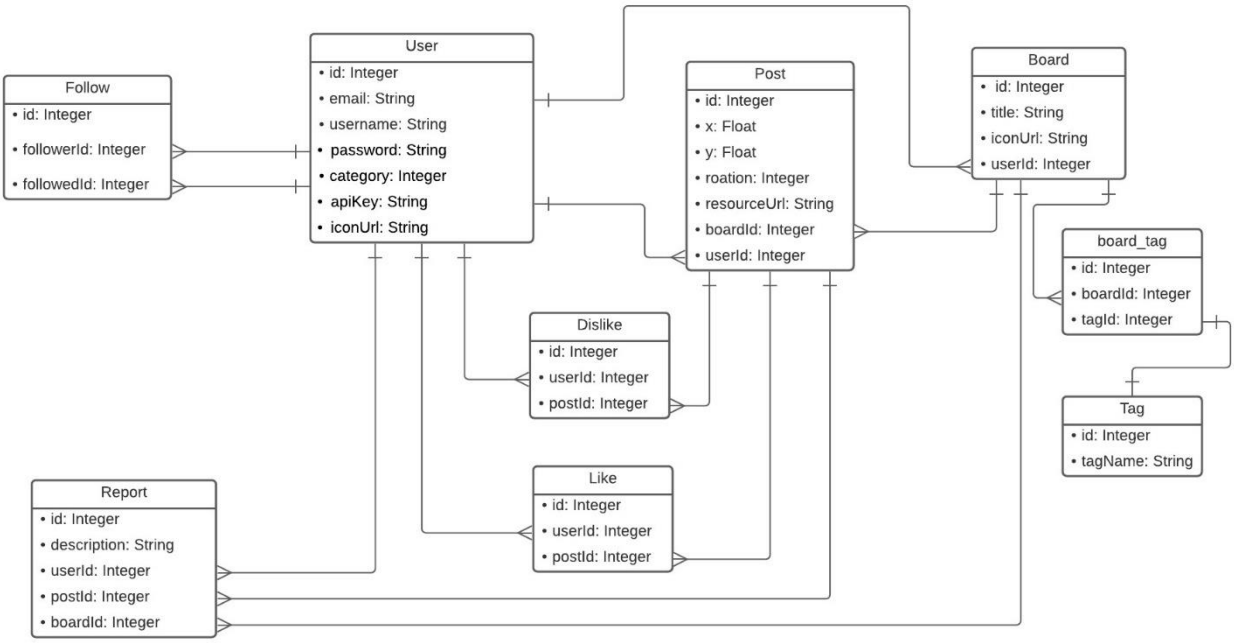



*Figura 26 Diagrama de clases de la base de datos*

- User

| Nombre de la columna | Tipo | Permite valores nulos | Descripción |
|---|---|---|---|
| **id** | Integer | no | Id del usuario. |
| **email** | String | no | Correo del usuario. |
| **username** | String | no | Nombre del usuario. |
| **password** | String | no | Contraseña encriptada por medio de Bcrypt (Apartado 5.3.1.4). |
| **category** | Integer | no | Es el número que indica la categoría del usuario. En caso de ser 0 se trata de un super usuario y en caso de ser 1 es un cliente de la aplicación. |
| **apiKey** | String | no | Es la clave que el cliente utilizará para identificar al usuario en las llamadas a la API REST. |
| **iconUrl** | String | si | La url de la imagen guardada en Firebase Storage (Apartado 5.3.2.5) que el usuario tiene en su perfil. |

- Board

| Nombre de la columna | Tipo | Permite valores nulos | Descripción |
|---|---|---|---|
| **id** | Integer | no | Id del Board. |
| **title** | String | no | Título del Board. |
| **iconUrl** | String | si | La url de la imagen guardada en Firebase Storage (Apartado 5.3.2.5) que el Board mostrará en su cabecera. |
| **userId** | Integer | no | El id del usuario que ha creado el Board. |

- Post

| Nombre de la columna | Tipo | Permite valores nulos | Descripción |
|---|---|---|---|
| **Id** | Integer | no | Id del Post. |
| **x** | Float | no | Posición en el eje de las X dentro del Board, comprendida entre los valores entre 0 y 1. |
| **y** | Float | no | Posición en el eje de las Y dentro del Board, comprendida entre los valores entre 0 y 1. |
| **rotation** | Integer | no | Rotación con la que el Post se mostrará, comprendida entre los valores 0 y 360. |
| **resourceUrl** | String | no | La url de la imagen guardada en Firebase Storage (Apartado 5.3.2.5) que será el contenido del Post. |
| **boardId** | Integer | no | La id del Board que contiene el Post. |
| **userId** | Integer | no | La id del usuario que ha creado el Post. |

- Like

| Nombre de la columna | Tipo | Permite valores nulos | Descripción |
|---|---|---|---|
| **id** | Integer | no | Id del Like |
| **userId** | Integer | no | El id del usuario que ha realizado el Like |
| **postId** | Integer | no | El id del post en el que el usuario ha realizado el Like. |

- Dislike

| Nombre de la columna | Tipo | Permite valores nulos | Descripción |
|---|---|---|---|
| **id** | Integer | no | Id del Dislike. |
| **userId** | Integer | no | El id del usuario que ha realizado el Dislike |
| **postId** | Integer | no | El id del post en el que el usuario ha realizado el Dislike. |

- Board_tag

| Nombre de la columna | Tipo | Permite valores nulos | Descripción |
|---|---|---|---|
| **id** | Integer | no | Id del Board_tag. |
| **boardId** | Integer | no | Id del Board que relaciona el tag con el Board. |
| **tagId** | Integer | no | Id del tag que relaciona el Board con el tag. |

- Tag

| Nombre de la columna | Tipo | Permite valores nulos | Descripción |
|---|---|---|---|
| **id** | Integer | no | Id del tag. |
| **tagName** | String | no | El nombre del tag, un texto que jamás se formará por más de una palabra. |

- Report

| Nombre de la columna | Tipo | Permite valores nulos | Descripción |
|---|---|---|---|
| **id** | Integer | no | Id del Report. |
| **descripción** | String | si | Descripción que el usuario envía sobre el reporte. Este puede ser vacío y así ser simplemente una muestra de descontento infundada. |
| **userId** | Integer | no | Id del usuario que ha realizado el reporte. |
| **postId** | Integer | si | Id del Post al que se le ha realizado el reporte. Puede ser nulo porque el reporte puede haberse hecho a un Board o a un Post. |
| **boardId** | Integer | si | Id del Board al que se le ha realizado el reporte. Puede ser nulo porque el reporte puede haberse hecho a un Board o a un Post. |

- Follow

| Nombre de la columna | Tipo | Permite valores nulos | Descripción |
|---|---|---|---|
| **id** | Integer | no | Id del Follow. |
| **followerId** | Integer | no | Id del usuario que ha realizado el Follow. |
| **followedId** | Integer | no | Id del usuario al que se le ha realizado el Follow. |

### 7.2.2. API REST

Para poder hacer uso de la API REST hemos definido un nombre de url para cada una de las peticiones que se podrán hacer, estas peticiones, en este proyecto podrán ser o POST o GET. Con cada una de las peticiones, en excepción de Login y Register, irán acompañadas de un "header" (Un cabecero). Este "header" será siempre el valor de la "apiKey" del usuario, será el valor que el cliente guardará y enviará en las llamadas a las peticiones de la API REST para autentificar al usuario y comprobar que tiene capacidades para poder realizar las peticiones.
La API diseñada en este proyecto está ejecutándose en Heroku (Recordemos el apartado 5.3.1.6) por lo tanto las peticiones que se vayan a hacer a esta API REST, como hemos visto ya en el ejemplo de la petición anterior, se harán partiendo de la dirección que proporciona Heroku:

*https://media-board.herokuapp.com/*

Añadiremos al final "/api" al final de esta dirección para referirnos a las peticiones de la APIREST, así se ha definido en la implementación de las API REST.
Entonces, las peticiones POST van acompañadas de un cuerpo en formato JSON y las respuestas, tanto de POST o GET, son también cuerpos en JSON.
En el caso de las peticiones GET los valores que se requieran para poder hacer la petición estarán contenidos en la propia dirección de la petición.
Un ejemplo podría ser:

*https://media-board.herokuapp.com/api/board?boardId=1*

En esta petición se puede ver:

- https://media-board.herokuapp.com: La dirección que nos proporciona Heroku donde la API-REST está ejecutándose.
- /api: Indicamos que vamos a hacer una petición a la API-REST.
- /board: Añadimos a la dirección el nombre de la petición.
- ?boardId=1: En este caso le estamos insertando un parámetro llamado "boardId" de valor 1 para poder realizar la petición.

Los errores de las llamadas de la API REST se recibirán como mensajes de error de la petición, no contendrán ningún cuerpo, únicamente se recibirán con un código de estado 500 con un mensaje de error. Podemos ver la documentación más detallada sobre las llamadas que ofrece la API REST en el Anexo D.

### 7.2.3. ViewModel y view

En la implementación del cliente hemos implementado un patrón MVVM (Explicado en la estructura del frontend en el apartado 5.2.2). En este apartado aclararé como esto se ha llevado a la práctica.
Podemos ver que en el proyecto de Frontend existe un directorio llamado "ui", en la raíz de este directorio veremos los distintos directorios que hacen referencia a las distintas pantallas definidas en el diseño que podemos ver en el apartado 6.5, cada una de es una pantalla en la aplicación. Cada uno de estos directorios contiene un Fragment y un ViewModel, recordamos que en el patrón MVVM la lógica queda delegada al ViewModel, mientras que la vista (En este caso los Fragment) sería la encargada de presentar los datos. Los Fragments harán uso de la herramienta Databinding (Apartado 5.3.2.7), por lo tanto, incluyen un método que se gestionará de forma nativa:

```
override fun onBindingCreated(
```

Dentro de este método podremos utilizar el valor “binding” para aplicar los cambios en la vista haciendo referencia a los distintos componentes de estos Fragments (Textos, imágenes, otros fragments…). Además, al inicio de este método siempre asignaremos el Viewmodel correspondiente al Fragment.
Los Viewmodel que se asignan a cada Fragment serán los únicos objetos dentro del directorio “ui” que tendrán acceso a los demás directorios del proyecto, el viewmodel es el que conocerá que código debe ejecutar para recibir los datos que obtendrá la vista.
Los viewmodels contendrán además un objeto llamado “state”, este objeto es el contenedor de la información que se mostrará en la vista de forma directa o bien será gestionado por el fragment para poder presentarlo de la forma correcta. Para ver exactamente cuál es el flujo de ejecución podemos ver el flujo de este patrón en la Figura 27, además explicaré punto por punto la secuencia utilizando como ejemplo el caso del botón Login a continuación.

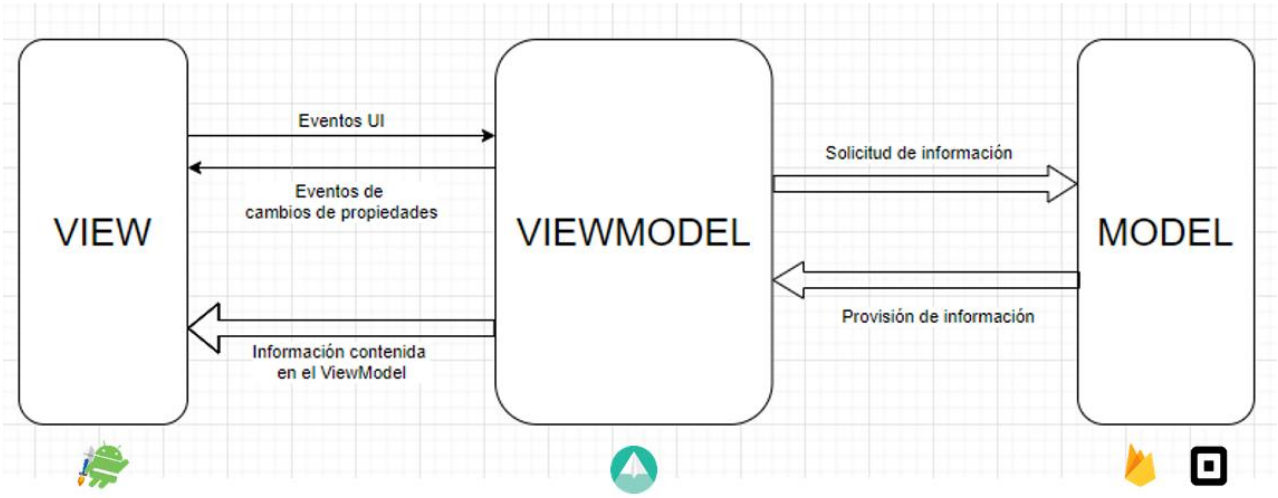



*Figura 27 Flujo de un Model-View-ViewModel*

1. **El usuario presiona el botón Login. El Fragment ha definido un comportamiento para este botón:**

Estableceremos que cuando se haya realizado un clic en este botón se hará la llamada al Viewmodel (Que en el fragmento de código sería “vm”).

2. **El Viewmodel recibe la orden del Fragment y pone el proceso en marcha:**

Este será el flujo que utilizaremos con todos los Viewmodel de la aplicación para hacer peticiones a la API REST. Se trata de un objeto de tipo “MutableLiveData” contenido en los valores de la vista que almacena el ViewModel, es un objeto que hereda de LiveData (Anexo B). Si asignamos a este objeto cualquier valor, aún que sea un valor nulo, su observador lanzará un evento. Por lo tanto, lo que haremos será actualizar el valor de este objeto “MutableLiveData” dentro del ViewModel para que este sepa que código debe ejecutar.

3. **Se realizan las llamadas a los servicios externos a la ui.**

Una vez ejecutado el evento, el ViewModel hará las llamadas a los servicios externos de la UI para pedir a la API REST (O cualquier otro servicio externo) los datos necesarios para llevar el Login a cabo. Estas llamadas se harán utilizando la función “transformLatest” de los LiveData.

4. **Esperamos la respuesta del servicio externo a la ui.**

El ViewModel una vez emita la llamada al servicio pertinente para realizar una petición, debe quedarse esperando una respuesta. En el caso de este proyecto, como hemos visto en el punto anterior, hemos utilizado la función “transformLatest”, la cual modificará el valor del MutableLiveData por el resultado obtenido posteriormente por el servicio externo.

5. **Actualizamos la vista dependiendo del resultado obtenido por el ViewModel.**

Una vez se ha modificado el valor del MutableLiveData con la respuesta del servicio externo, también tenemos que haber definido un observador en el MutableLiveData que lance un evento, contra el objeto loginResult en este caso, en el momento que el MutableLiveData cambie sus valores. El fragmento estará a la espera del evento para realizar los cambios en la vista. Por lo tanto, aquí es donde finalmente el Fragment, si es necesario, realizará los cambios en la vista si la petición se ha llevado con éxito.

### 7.2.4. Inyección de dependencias

A lo largo de la aplicación existe la necesidad del uso de clases que provienen de otras capas del proyecto. Y la inyección de dependencias, en este proyecto, se implementado de la forma en la que se define en Clean Architecture, del cual hablamos al definir la estructura de la aplicación en el apartado 5.2.2. Para poder llevar a cabo esta inyección de dependencia se ha utilizado la herramienta Dagger Hilt (De la que hablamos ya en el apartado 5.3.2.3). Y ahora hablaremos de como esto se ha llevado a cabo:
En el directorio “domain”, podemos ver que existe un subdirectorio llamado “provider”. Dentro de este directorio encontraremos los objetos “Module”, que serán los que se encargarán de proveer desde cualquier parte del código (Aún que esto solo sucederá en una dirección como indica Clean Architecture) los servicios. Vemos que los módulos se inicializan con dos notaciones:

```
@Module
@InstallIn(ApplicationComponent::class)
```

Con “@Module” indicamos que esto será un bloque de código que contribuye a la creación de un grafo de objetos. Un módulo puede definirse como un grafo en el que los distintos servicios se inyectan entre sí para poder ser utilizados en conjunto.
Encontramos también la notación “@InstallIn(ApplicationComponent::class)” es para indicar a Dagger Hilt en que componente de la aplicación se instalará este módulo en el momento de la compilación del código. En este caso está instalándose en “ApplicationComponent” por lo tanto el módulo proveerá siempre que la aplicación este en ejecución.
Una vez definimos el módulo, pasamos a ver como este creara los servicios que le sean demandados. Cada uno de estos tendrá las notaciones:

```
@Provides
@Singleton
```

Indicando que este método será un proveedor que podrá ser accedido desde cualquier punto del proyecto con “@Provides”. Además, con “@Singleton” aclaramos que solo existirá una instancia de este servicio.
Entre los distintos proveedores declarados en el interior de estos módulos pueden existir dependencias de otros servicios del mismo modulo, o incluso de otro modulo declarado en el proyecto. Pero de estas dependencias se encargará internamente Dagger Hilt, nosotros simplemente declararemos el tipo de servicio que se requiere como atributos de entrada de la construcción de la clase del servicio.
Una vez declarados estos proveedores podemos hacer uso de ellos en cualquier clase. Utilizamos la notación “@Inject” para que automáticamente los valores referenciados por esta notación sean proveídos por el módulo pertinente.

### 7.2.5. Gestión de imágenes

Debido a que el almacenamiento de una gran cantidad imágenes supondría un gran espacio en la base de datos y que eso supondría la necesidad tener que pagar por este servicio. Se ha realizado el almacenamiento del contenido multimedia de esta aplicación por medio del servicio Firebase Storage. Como podemos ver en la Figura 28, desde el cliente de Android enviamos la imagen a FireBase y este servicio devolverla a URL donde tiene contenida la imagen. Posteriormente esta URL será la que enviaremos al servicio Backend para que realmente lo que se guarde en la base de datos sea la URL de la imagen contenida en Firebase. Es por eso por lo que podemos que hay campos en la base de datos (Podemos verlo en el apartado 7.2.1) que almacenan una dirección URL al fichero pertinente dentro del servicio de FirebaseStorage. Por lo tanto, podemos ver un efecto retardado en el muestreo de las imágenes una vez se crea un Post o un Board.
El hecho de gestionar las imágenes por medio de este servicio es el causante de este efecto en la experiencia del usuario y en un futuro se tiene planteado expandir la base de datos para tener la capacidad de almacenaje como para poder asumir este servicio.

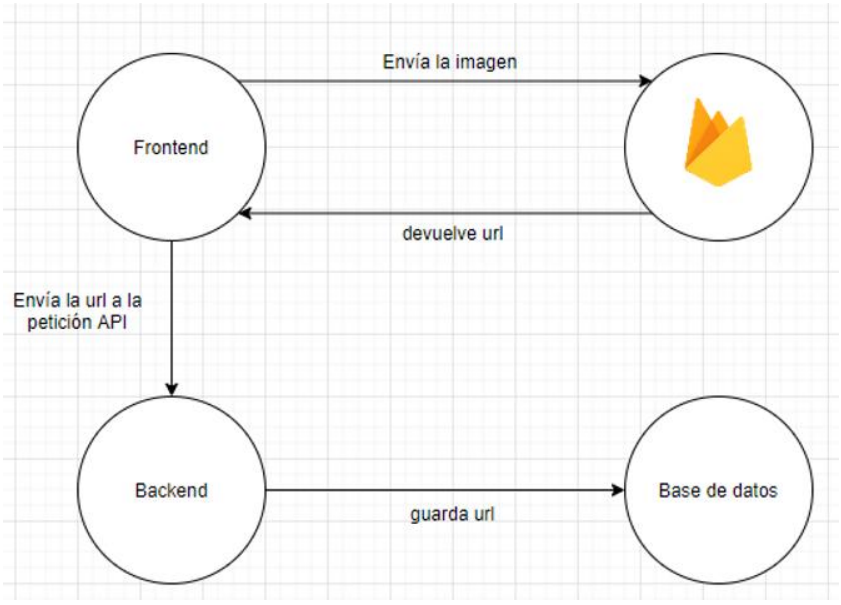



*Figura 28 Flujo de gestión de imágenes*

# 8. Pruebas en usuarios

Debido a que nos encontramos en el desarrollo de una aplicación móvil la usabilidad de esta es muy importante. Por lo tanto, se han realizado pruebas en usuarios finales para saber si la navegación de la aplicación es intuitiva y cómoda, y conocer mejor el efecto que causaría la aplicación en los usuarios finales.

## 8.1. Perfil de los voluntarios y método de reclutamiento

Para la realización de las pruebas se ha definido un perfil muy parecido al que se han definido en otras redes sociales, el cual será el objetivo principal de la aplicación que se desarrolla en este proyecto. Los voluntarios se encontrarán entre los 20 y 30 años, serán ya conocedores y usuarios habituales de otras redes sociales y se harán las pruebas en 10 voluntarios.
El reclutamiento de los voluntarios de prueba se ha realizado entre mis círculos personales, he hecho correr la voz entre mis amistades y les he informado de que día tendrían que realizar la prueba y donde. Aquellas personas que quisieron llevar las pruebas a cabo.

## 8.2. Metodología

El tipo de prueba llevado a cabo será de validación, se buscará que el producto cumple con los objetivos del diseño.
Las tareas se llevarán a cabo en un espacio cerrado donde se disponga de una conexión estable. Las pruebas se realizarán de forma moderada, únicamente se encontrarán en este espacio el voluntario y el moderador. Se le explicará al sujeto la idea de la aplicación, pero en ningún momento se le explicará cómo funciona, únicamente debe conocer el concepto del proyecto y realizará una serie definida de tareas a realizar en la aplicación que se llevarán a cabo de forma secuencial. Se contarán el número de pasos que ha realizado para llegar a realizar la tarea satisfactoriamente y se compararan con el numero ideal de acciones que se requieren para llevarlas a cabo. El listado de tareas a realizar es el siguiente:

1. **Registro:** Realizar el registro de una cuenta.
2. **Logout:** Hacer logout.
3. **Login:** Hacer login.
4. **Profile:** Cambiar la imagen de su perfil.
5. **Nuevo board:** Crear un nuevo Board.
6. **Ver board:** Acceder a un Board.
7. **Nuevo post:** Añadir los Posts que desee al Board.
8. **Valorar:** Valorar los posts que desee el usuario.

Para la realización de esta tarea se dispone de un dispositivo móvil personal, pero en caso de que el usuario quiera utilizar su dispositivo móvil y este soporte la plataforma Android se dispone de la libertad de realizar las pruebas en su dispositivo personal. Los dispositivos en los que se han llevado pruebas han sido:

- Xiaomi Mi note 10 pro (Mi dispositivo móvil personal)
- Xiaomi Mi note 10
- Samsung Galaxy S10
- Samsung Galaxy A32
- Nokia 3.4
- Huawei P20

Finalmente, se realizarán una serie de preguntas, un aspecto más exploratorio de estas pruebas habrá un espacio en el que se pedirán observaciones y mejoras que los sujetos creen que podrían aumentar la calidad de la aplicación. El listado de preguntas que se harán en búsqueda de esta información será el siguiente:

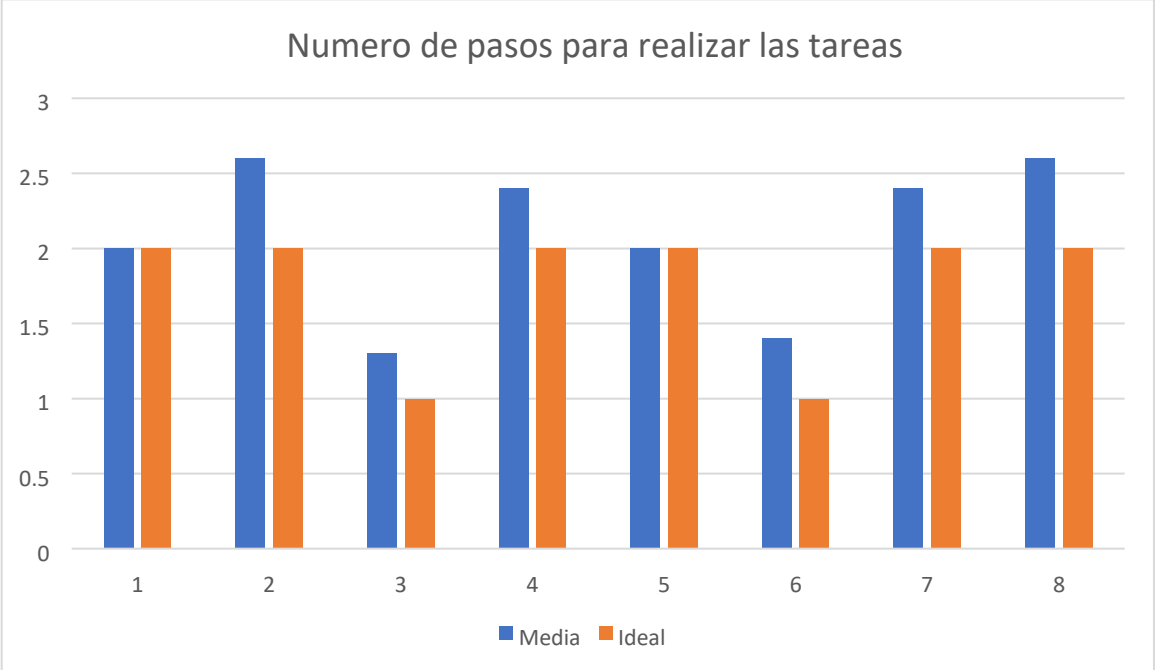
- ¿Crees que ha sido fácil encontrar los elementos de la aplicación que se han demandado?
- ¿Como valorarías estéticamente hablando la aplicación?
- ¿La aplicación responde de forma rápida?
- ¿Qué crees que podría mejorar la calidad de la aplicación, o que crees que le falta?

En la prueba se contabilizarán las acciones que ha realizado el usuario, cada una de las siguientes acciones añadirían un valor de 1 a esta métrica:

- El usuario se desplaza a un nuevo punto de la aplicación.
- El usuario presiona un botón (Si este desplaza a otro punto de la aplicación no se acumulará con el primero).

Empezamos a contabilizar con esta métrica desde el punto en el que terminó la prueba previa. No se ha tenido en cuenta las acciones dentro de la galería, además tampoco se ha contado como una acción que el usuario presione el botón para acceder a la galería desde dentro del formulario de creación de Board, debido a que, en el caso, podría crearse sin ninguna imagen y esto no resultaría útil para valorar la navegación por la aplicación.
En las tareas “Nuevo post” y “Valorar” solo se contabilizará el número de acciones realizados hasta añadir el primer Post o valorar el primer Post.

## 8.3. Resultados




*Grafica 1 Resultados de las acciones llevadas a cabo por los voluntarios*

En la Grafica 1 podemos ver la gráfica resultante de las acciones que ha tomado a los usuarios realizar las tareas en comparación con un valor ideal representado con el color naranja, este

valor ideal es el número de acciones mínimo para realizar la tarea. Las tareas han sido realmente satisfactorias en este aspecto, los usuarios han encontrado los elementos para realizar las tareas de una forma muy fácil e intuitiva. Además, se ha tomado nota de algunas observaciones a destacar observadas durante la realización de estas tareas:

- En la prueba “Logout” los voluntarios han navegado hacia exploración cuando les he mencionado que realizaran el Logout. En una ocasión he preguntado a un voluntario porque ha ido hacia allí y me respondió que normalmente hay un botón de “opciones” para estas acciones, pensaba que se encontraría allí.
- En la prueba “Nuevo Board” un voluntario pudo crear tags vacíos.
- En la prueba “Ver Board” algunos de los voluntarios han accedido a un Board desde explorar y no desde el menú del perfil, el caso ideal de esta prueba únicamente el caso en el que accedan desde el perfil, así que esto no debe entenderse como un problema de usabilidad.
- En la prueba “Valorar” se han acumulado acciones debido a que los sujetos han dado por error un toque fuera de la ventana sobre la que actuaban y el listado de posts se ha cerrado.

Referente a las respuestas de los sujetos a las preguntas realizadas:

- *¿Crees que ha sido fácil encontrar los elementos de la aplicación que se han demandado?*

Por general todos los usuarios han manifestado que les ha resultado fácil realizar las tareas, han comentado que es intuitiva y fácil de entender. Pero como ya hemos comentado anteriormente, dos de los sujetos han puntualizado que en el listado de posts cuando se realiza un toque entre dos posts el listado se cierra.

- *¿Como valorarías estéticamente hablando la aplicación?*

Todos los usuarios han respondido de manera positiva a este aspecto. Se ha tenido que insistir para extraer as observaciones, las más destacadas han sido que las imágenes podrían tener bordes y que la aplicación es muy oscura.

- *¿La aplicación responde de forma rápida?*

En esta respuesta es donde se ha visto una negatividad, pues cuando se crea un Board o un Post la imagen toma bastante tiempo en cargar, esto ha sido manifestado por ocho de los diez sujetos.

- *¿Qué crees que podría mejorar la calidad de la aplicación, o que crees que le falta?*

Siete de los diez sujetos han manifestado que la funcionalidad de búsqueda sería algo muy útil, aunque para la primera versión de la aplicación esta funcionalidad no estará disponible.

Tres de los sujetos han manifestado que les gustaría que cuando tocasen un Tag los llevase a un listado de Boards que contengan ese tag.

Ocho de los diez los sujetos han querido resaltar que las imágenes tardan mucho en cargarse una vez son creadas.

## 8.4. Conclusiones de las pruebas

Por medio de estas pruebas se han podido identificar errores en la aplicación, como hemos podido ver, un voluntario pudo crear tags vacíos en la prueba de “Nuevo Board”, también hemos podido ver que cerraban la visualización de los Posts tocando en el espacio entre estos por error. Estos errores pueden afectar a la usabilidad y se archivarán para su posterior solución.

Además, los voluntarios han manifestado también la necesidad de la funcionalidad de búsqueda, estoy de acuerdo, es una funcionalidad que debe ser desarrollada en un futuro próximo. Además, han dado ideas que no se habían planteado en el diseño, como poder buscar directamente los Boards relacionados con un tag con el simple hecho de tocarlo. Las quejas sobre el retraso de carga de las imágenes es algo que ya prevenía, pero debido a los recursos disponibles para el desarrollo de esta aplicación no es posible llevar a cabo una mejora en este aspecto de la aplicación en este momento.

Para terminar, revisando los resultados de las tareas, la diferencia entre las acciones realizadas en las pruebas para realizar las tareas es muy negligible en comparación con el número de acciones idea. Podemos concluir a partir de las pruebas que, efectivamente, la aplicación consta de una navegación que resulta fácil para los usuarios ya conocedores de otras redes sociales. Ninguno de ellos ha manifestado dudas cuando las tareas se le eran encomendadas.

# 9. Conclusiones

En el apartado 2 definimos unos objetivos a realizar por la aplicación. Después de llevar a cabo el desarrollo de la aplicación podemos concluir que las dos condiciones que la aplicación debía cumplir se han respetado por completo en la aplicación. La aplicación brinda a los usuarios de anonimidad y no hay una jerarquía entre estos usuarios. Pero si hablamos de los aspectos más técnicos de la aplicación hay algunos objetivos definidos en el diseño que no han llegado a cumplirse. Debido a la gran cantidad de trabajo que ha supuesto la implementación del código y las correcciones que han tenido que llevarse a cabo, tres de los casos de uso los cuales son Search, Report y Followed no han podido llevarse a la práctica. En el caso del Followed no resulta un problema debido que al replantearme este caso de uso después del desarrollo de la aplicación considero que es una idea que ataca a las dos condiciones que debe respetar la aplicación. En el caso del Search y Report sí que considero una necesidad de la aplicación. Algunos de los voluntarios de las pruebas en usuarios manifestaron que estas funcionalidades deberían ser parte de la aplicación. Yo también lo considero así.
En aspectos de usabilidad las pruebas en usuarios han reflejado que se trata de un aspecto muy logrado en la aplicación. Resulta muy fácil e intuitivo navegar por la aplicación. El problema principal en este caso serían los prolongados tiempos de carga de las imágenes.

En conclusión, el desarrollo de la aplicación no ha podido llevarse a cabo en su totalidad, pero la calidad de las funcionalidades que actualmente ofrece la primera versión de la aplicación ha respetado las dos condiciones que impusimos en los objetivos. Además de reflejar unos resultados positivos en los resultados de las pruebas en usuarios. Para saber si las medidas tomadas podrían realmente acabar con los problemas identificados en las redes sociales o reducirlos, ahora mismo, solo podríamos especular, la aplicación debería tener un público comparable con el resto de las redes sociales para poder realizar pruebas más realistas. Pero con las condiciones definidas para la realización de la aplicación se elimina la posibilidad de que algunos de los problemas comentados a lo largo de este documento puedan llevarse a cabo.

# 10. Trabajo futuro

## 10.1. Control de contenido

Como hemos visto, uno de los objetivos del proyecto es crear un entorno en el que los usuarios puedan expresarse con libertad. Esta libertad es dada por el anonimato, esto puede suponer un problema si no existe un control.

Lo usuarios en este momento, podrían publicar cualquier tipo de contenido en la aplicación. Esto puede dar lugar a publicaciones con contenido inapropiado. En el diseño actual ya está contemplada una gestión de la comunidad por medio de reportes, pero esto es algo que está atado al subjetivismo y además es un proceso que solo funcionará una vez el contenido ya esté presente en la aplicación.
Por lo tanto, esta aplicación necesitaría un proceso por el cual, antes de publicar cualquier contenido, realizase un análisis en busca de violencia, pornografía, propaganda... etc. Este proyecto se ha realizado en un tiempo en el cual esto no ha podido llevarse a la ejecución, pero si se ha podido buscar algunas tecnologías que podrían realizar la función.
Este proceso podría llevarse a cabo en el Frontend, pero realizar un proceso como este en el Frontend es algo que podría llevar a problemas de seguridad. Además, que este análisis no podría ser trasladado a otras plataformas, si se deseara desarrollar un cliente para iOS o para Windows esta funcionalidad tendría que ser replanteada para esta nueva plataforma.

Entonces, la opción más inteligente es realizar el control en la implementación del Backend.
Para realizar esta implementación podríamos apoyarnos en otras tecnologías:
Cloud Vision (API de Google), Amazon Rekognition o VISUA son tres de las tecnologías líderes en el campo de reconocimiento de imágenes. Pueden analizar contenido en búsqueda de objetos, texto y todas ellas además permiten realizar búsquedas personalizadas.
Por supuesto, la otra opción sería realizar el desarrollo de una Inteligencia artificial para realizar este proceso. Alcanzar la calidad de las tecnologías mencionadas podría resultar algo inasumible si no hay un equipo cualificado involucrado enteramente a este desarrollo.

Estas tecnologías mencionadas suponen un precio y la realización de este proceso para este trabajo de final de grado hubiese sido poco realista, por estos motivos, este trabajo no incluye tal control de contenido, pero soy enteramente consciente que para un proyecto como este sería algo necesario.

## 10.2. Seguridad

En este proyecto hemos hablado de Bcrypt (Apartado 5.3.1.4), el cual se está utilizando para guardar la contraseña de los usuarios encriptada en la base de datos. Por lo tanto, si de alguna forma se llegase a abrir una brecha en la aplicación que diese acceso a alguien a la base de datos, esta información le resultaría inútil.

Además, utilizamos Sequelize para acceder a la base de datos. Utilizando esta tecnología podemos evitar la inyección de SQL (Descrito en el Anexo C). Pero soy consciente que existen más brechas de seguridad en esta aplicación. Un ejemplo sería que no existe un control de validez de e-mail, por lo cual, se podrían realizar cuentas falsas. Para este caso en particular, como ya he comentado, debería existir un proceso de validez del correo electrónico.
Veo necesario un estudio de las debilidades de la aplicación y aplicar las medidas necesarias.

## 10.3. Monetización

Si la aplicación fuese lanzada al mercado sería deseable que produjese ingresos económicos, por lo tanto, es deseable realizar un plan de monetización.
La idea de poner un precio económico a la adquisición de la aplicación podría suponer un problema para la captación de clientes, para una aplicación sin un público y a la sombra de la enorme competencia en el mundo de las redes sociales, esta opción queda descartada.
También existiría la posibilidad de ofrecer un servicio de subscripciones para otorgar privilegios a usuarios. Pero ya se ha plasmado en este documento numerosas veces que todos los usuarios deben encontrarse en igualdad de condiciones, esta posibilidad tampoco sería contemplada para satisfacer esta característica.
Definitivamente, el plan de monetización que sería más atractivo para el cliente, en una aplicación de esta naturaleza, sería la publicidad. Con una cantidad de usuarios que convenciese a otras empresas para realizar pagos por la publicación de anuncios en la plataforma de MyBoards podría ser posible crear una fuente de ingresos.
Pero sin una base de usuarios solida la monetización por medio de publicidad sería imposible. Es por eso que en un principio habría que asumir que la aplicación no producirá un valor económico. Si esto cambiase en algún momento se procedería con un plan de monetización por medio de publicidad.

# 11. Anexo

## A- Framework

Un framework es una estructura conceptual y tecnológica de soporte definido, normalmente con artefactos o módulos de software concretos, que puede servir de base para la organización y desarrollo de software. En otras palabras, aprovechas una implementación previa, un compuesto de estructuras tecnológicas, para llevar a cabo implementaciones más sencillas y eficientes.

## B- Livedata

LiveData es una clase de contenedor de datos observable de Android.
A diferencia de un observable regular, LiveData está optimizado para responder correctamente a los ciclos de vida de los componentes de Android, como actividades, fragmentos o servicios.

LiveData solo actualiza observadores de componentes de apps que tienen un estado de ciclo de vida activo, es decir, los LiveData solo serán funcionales mientras el componente al que están asignados exista. En el momento que el componente asignado al LiveData sea eliminado el LiveData dejará de cumplir su función.

## C- Inyección SQL

Inyección SQL es una vulnerabilidad en la seguridad de una base de datos.

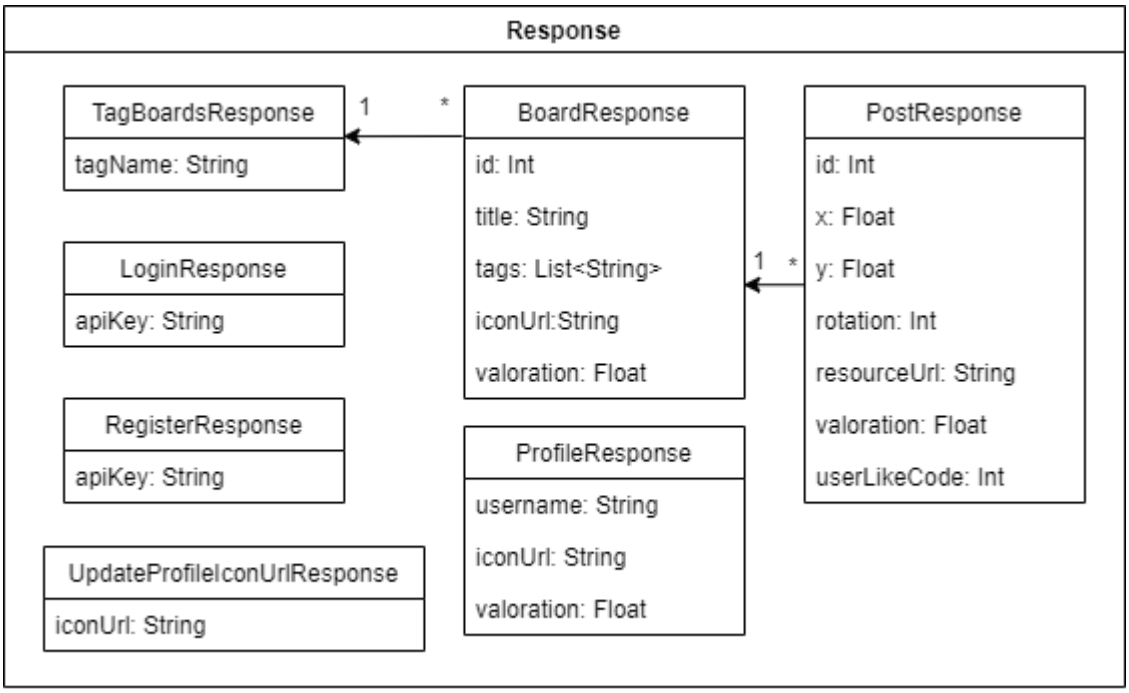
Se entiende como una inyección SQL el proceso por el cual atacante, puede utilizar cualquier proceso de una aplicación por el cual pueda filtrar código SQL hasta la base de datos y así ver o modificar información a la cual no debería tener acceso.

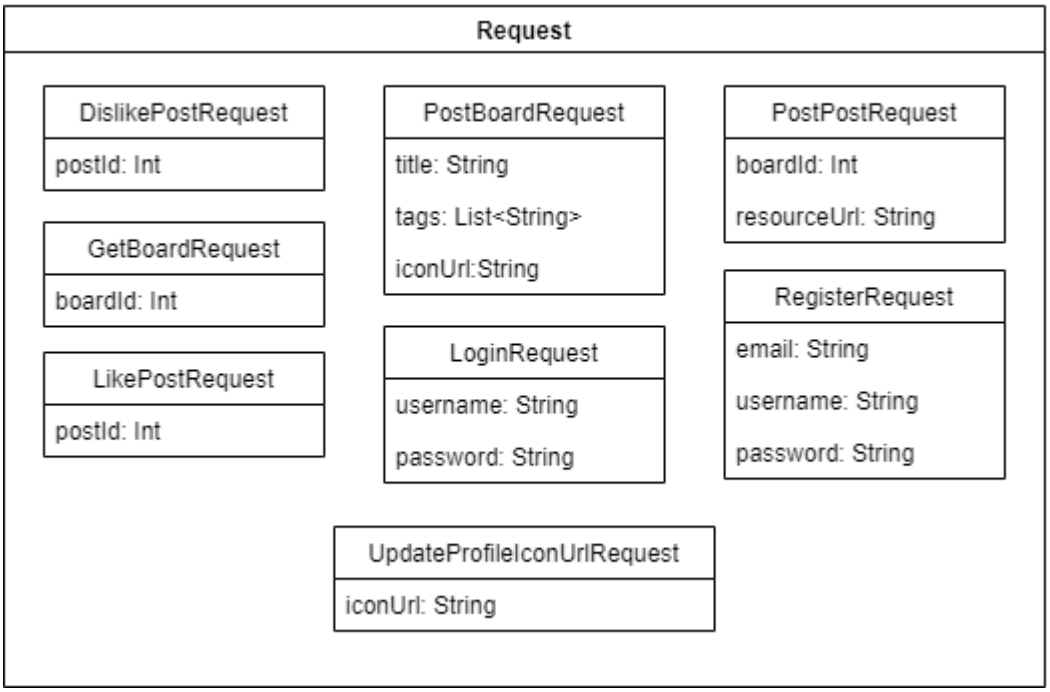
## D- Documentación de la API REST

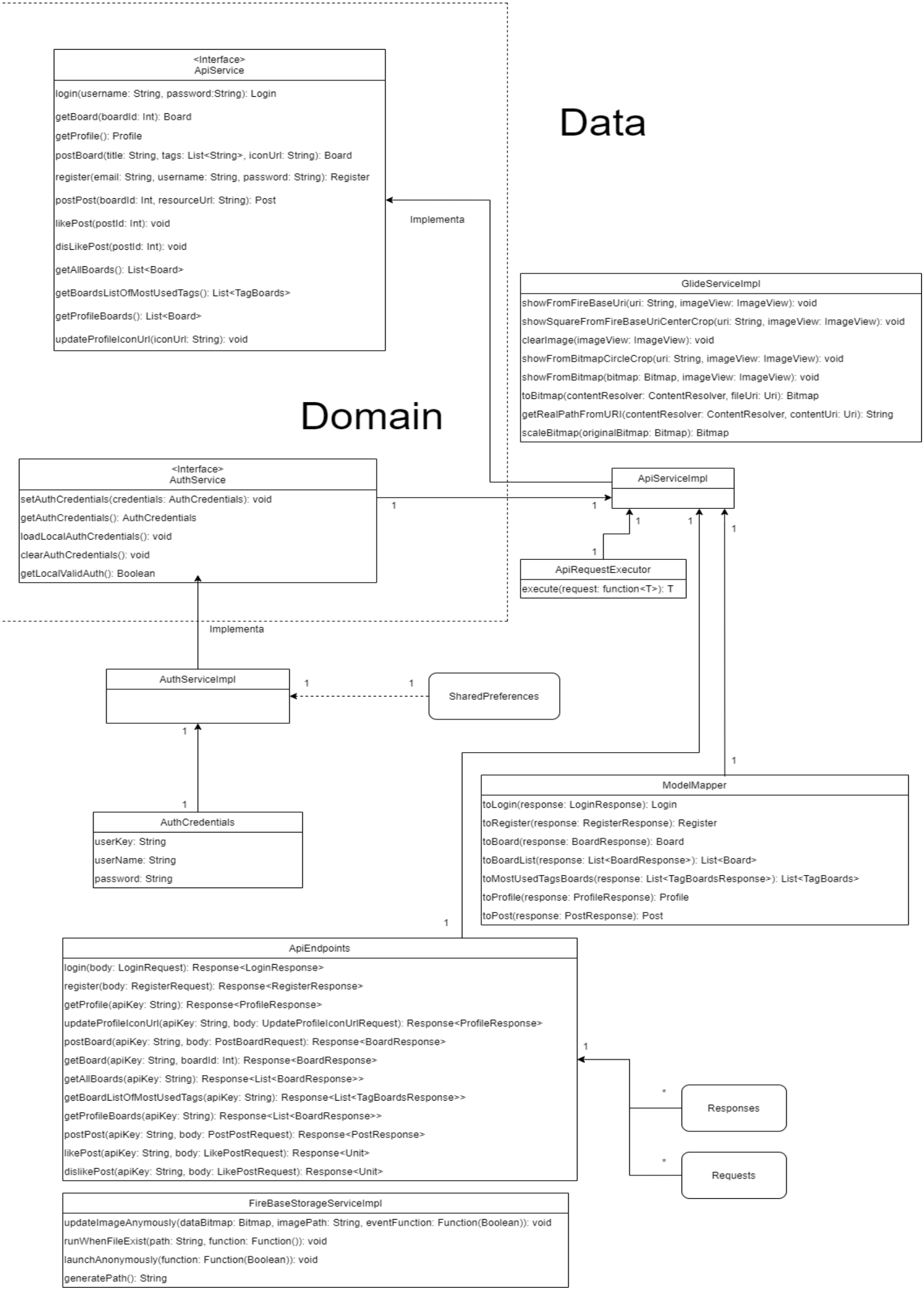
| Nombre de la petición | Método | Cuerpo de la petición (JSON) | Cuerpo de la respuesta (JSON) | Descripción |
|---|---|---|---|---|
| /login | POST | { "username": String, "password": String } | { "apiKey": String, } | Realizará el login del usuario y si el login se realiza satisfactoriamente devolverá la apiKey del usuario. |
| /signup | POST | { "email": String, "username": String, "password": String, } | { "apiKey": String, } | Realizará el registro del nuevo usuario a la aplicación, si el signup se realiza satisfactoriamente devolverá la apiKey del usuario. |
| /profile | GET | Vacío | { "username": String "iconUrl": String, "valoration": Float } | Recibimos la información del perfil del Usuario que realiza la petición identificándolo con la apiKey del header. |
| /profile/iconUrl | POST | { "iconUrl": String } | { "iconUrl": String } | Enviaremos una dirección que apunta a la imagen de Firebase de la nueva imagen del usuario y esta será sobre escrita en la base de datos. Devolverá la nueva dirección de nuevo como respuesta. |
| /board?boardId =X | GET | Vacío | { "id": Integer, "title": String, "tags": List[String], "iconUrl": String, "valoration": Integer, "postList": List[Post] } | Pedimos el Board con la id del parámetro del nombre de la petición. Donde X es la id del Board. La lista de "postList" será una lista de objetos con la misma estructura que veremos en la petición /board/post |
| /board | POST | { "title": String, "tags": List[String], "iconUrl": String, } | { "id": Integer, "title": String, "tags": List[String], "iconUrl": String, | Creamos un Board con la información que se encuentra en la petición, este Board se creará sin ningún Post |

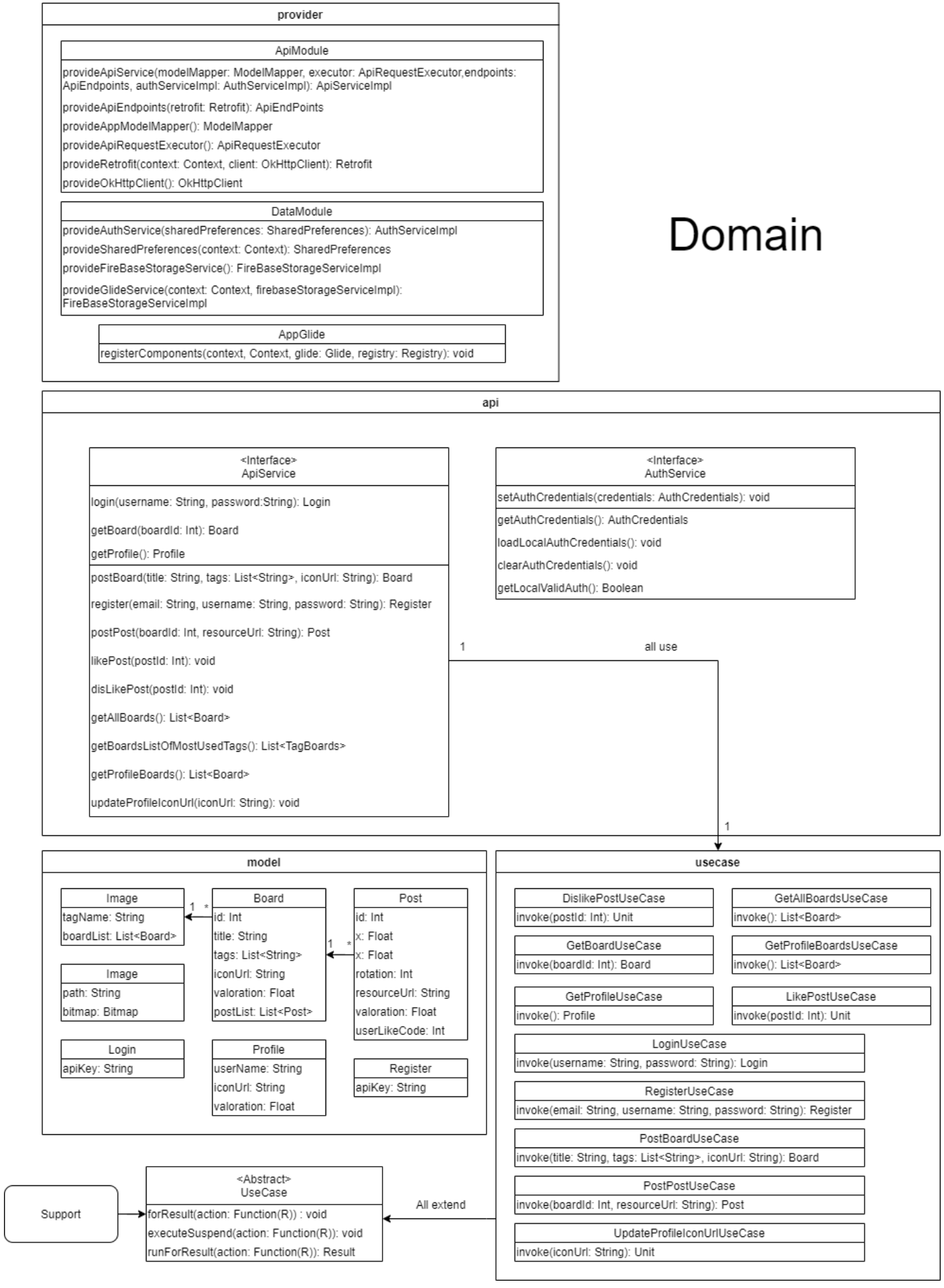
| | | | | |
|---|---|---|---|---|
| /boards | GET | Vacío | List[Board] | Recibimos todos los Boards de la aplicación. Cada Board contendrá la misma estructura mostrada en /board |
| /tags/boards | GET | Vacío | { "tagName": String "boardList":List[Board] } | Se recibe una lista con el contenido de la response que vemos en la table, con el nombre del tag y el listado de la información de todos los Boards que contienen el tag. |
| /board/post | POST | { "boardId": Integer, "resourceUrl": String, } | { "id": Integer, "x": Float, "y": Float, "rotation": Integer, "resourceUrl": String, "valoration": Float, "userLikeCode": Integer, } | Creamos un Post en el Board con boardId. Los valores x, y y rotation se crearán en el controlador de User |
| /board/post/like | POST | { "postId": Integer } | Vacío | Se realiza un Like en el Post indicado en postId, se usa la apiKey del usuario para saber de qué usuario proviene el Like. La respuesta se hará con una simple respuesta de confirmación. |
| /board/post/dislike | POST | { "postId": Integer } | Vacío | Se realiza un DisLike en el Post indicado en postId, se usa la apiKey del usuario para saber de qué usuario proviene el DisLike. La respuesta se hará con una simple respuesta de confirmación. |

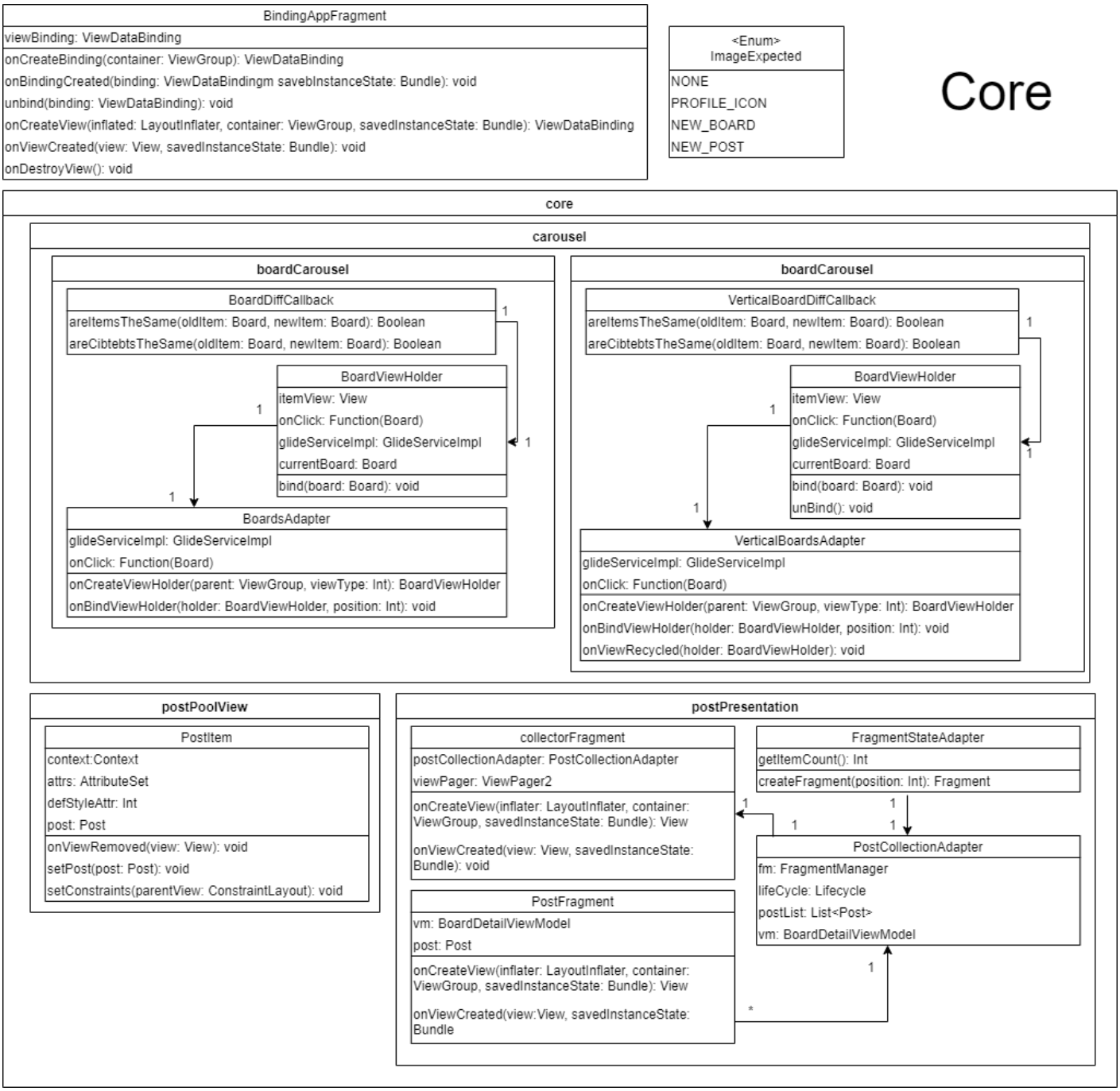
# E- Diagrama de clases de la aplicación Android

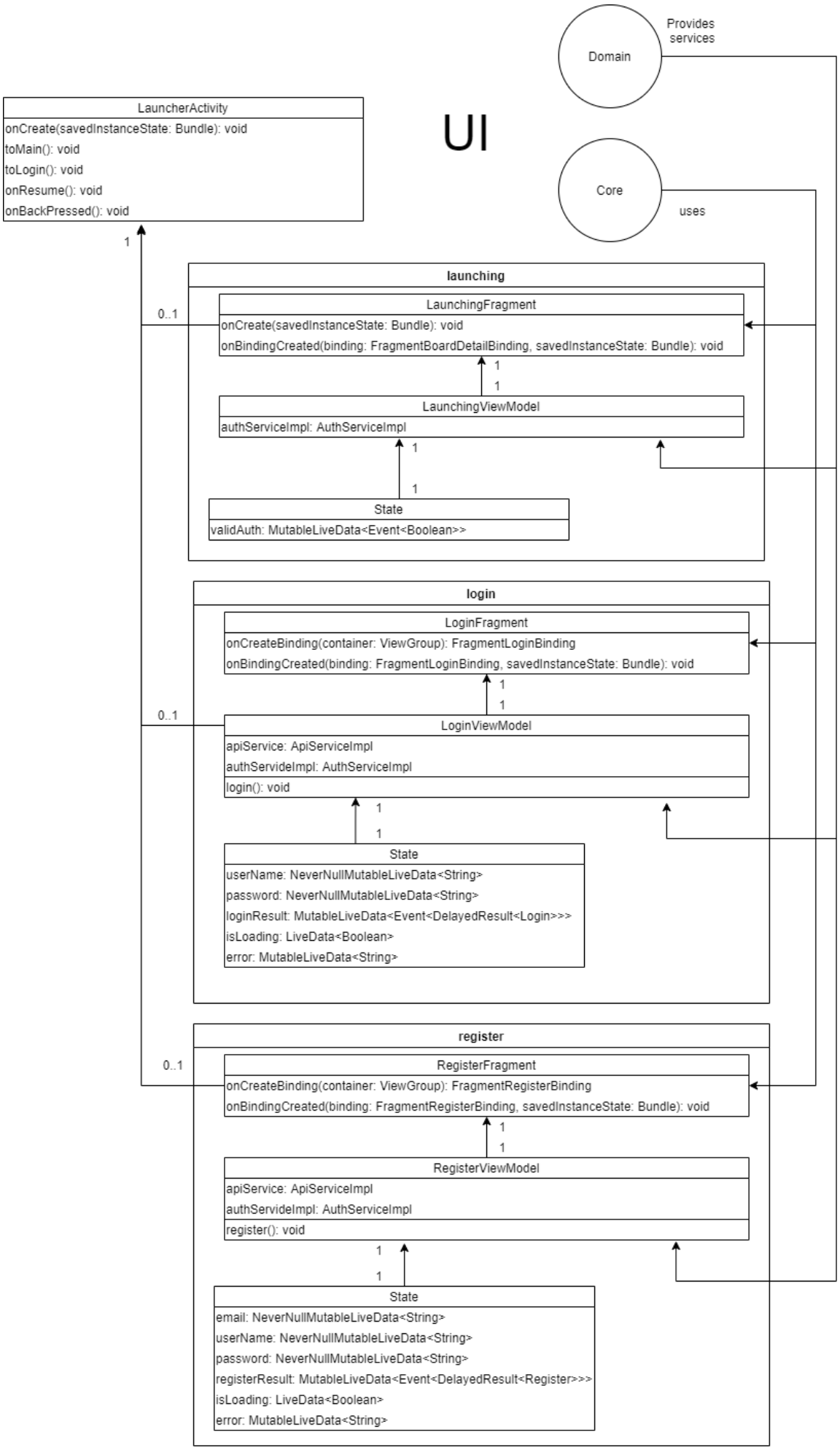


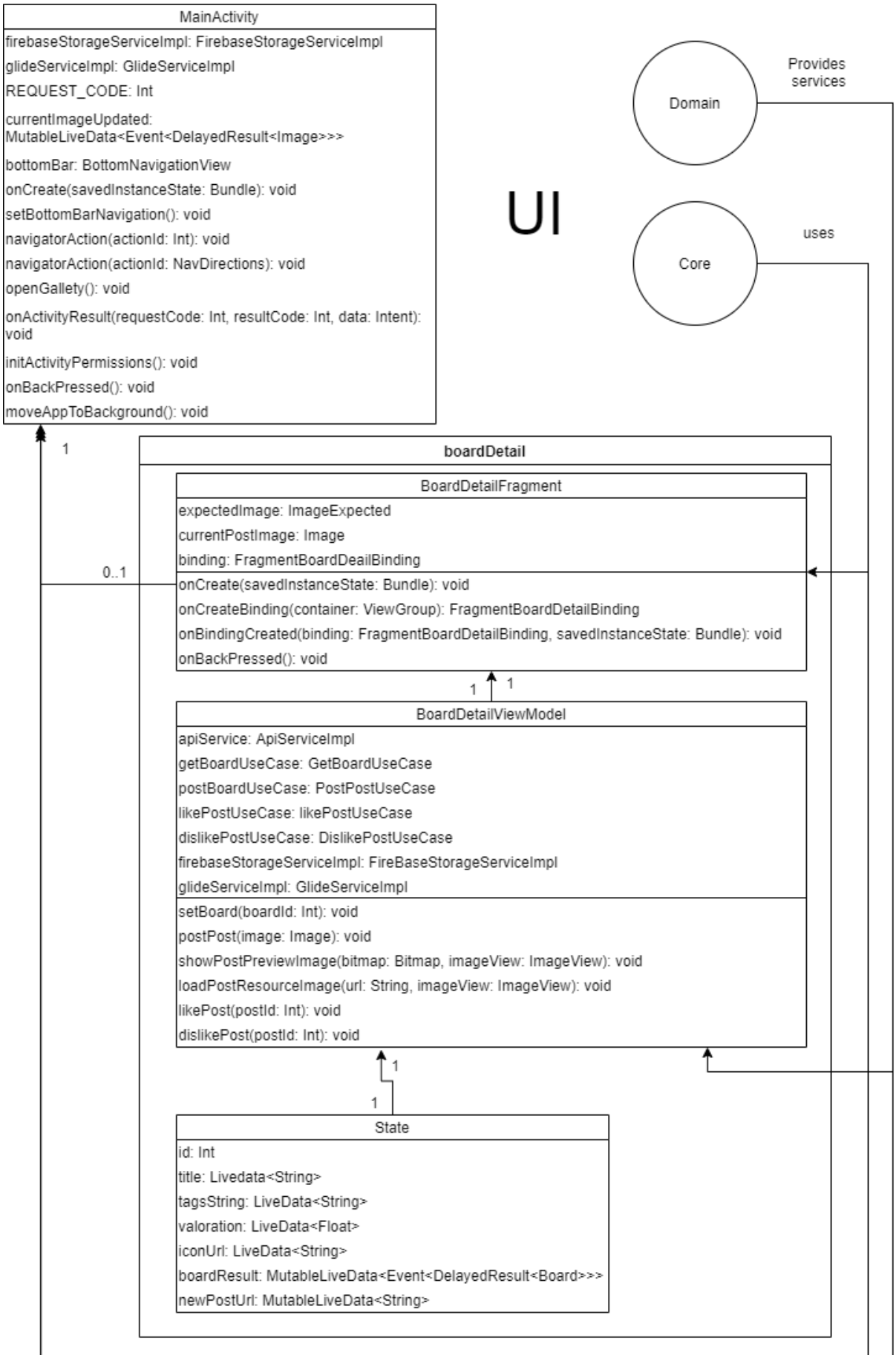



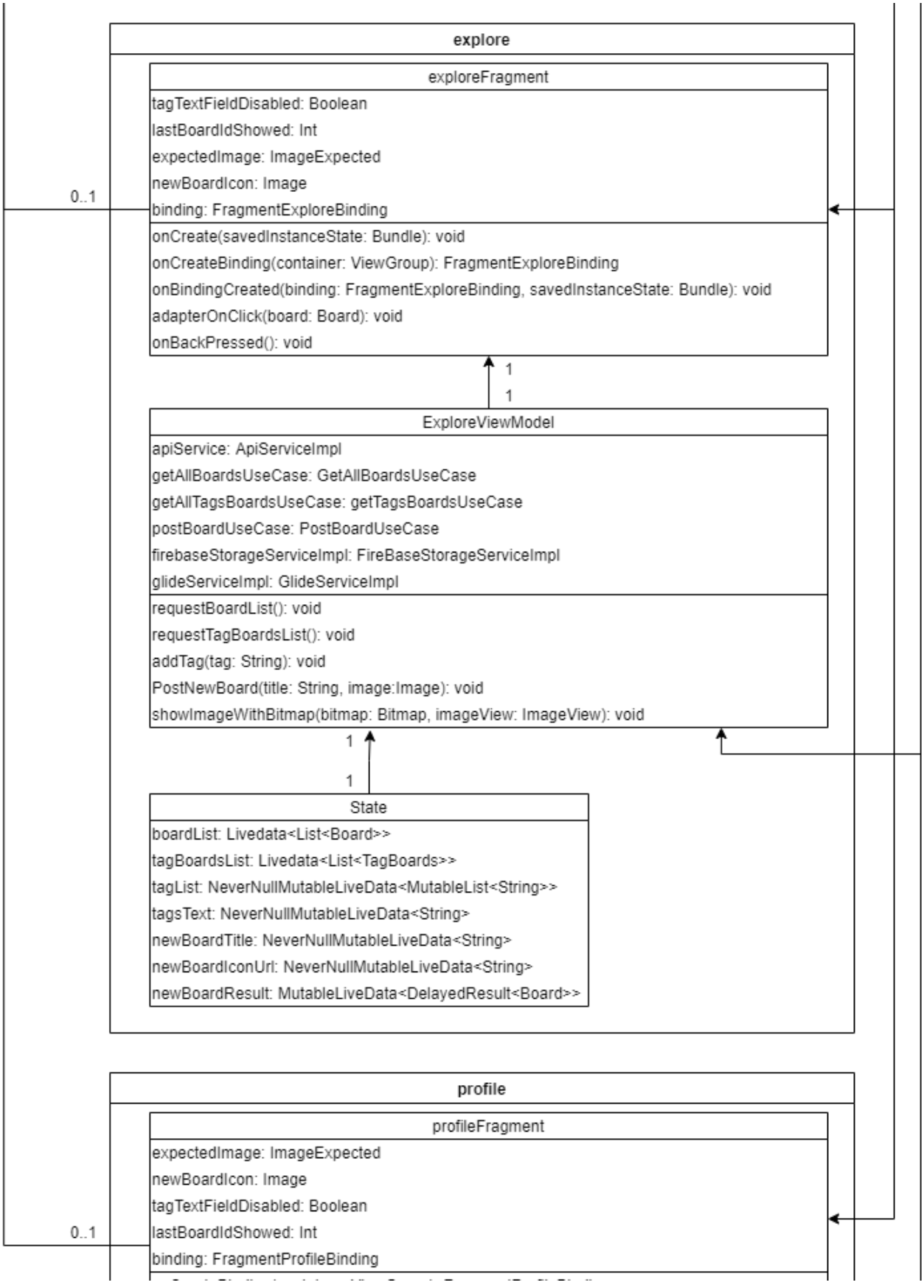

<Interface>
ApiService
login(username: String, password:String): Login
getBoard(boardId: Int): Board
getProfile(): Profile
postBoard(title: String, tags: List<String>, iconUrl: String): Board
register(email: String, username: String, password: String): Register
postPost(boardId: Int, resourceUrl: String): Post
likePost(postId: Int): void
disLikePost(postId: Int): void
getAllBoards(): List<Board>
getBoardsListOfMostUsedTags(): List<TagBoards>
getProfileBoards(): List<Board>
updateProfileIconUrl(iconUrl: String): void
Data
Implementa
GlideServiceImpl
showFromFireBaseUri(uri: String, imageView: ImageView): void
showSquareFromFireBaseUriCenterCrop(uri: String, imageView: ImageView): void
clearImage(imageView: ImageView): void
showFromBitmapCircleCrop(uri: String, imageView: ImageView): void
showFromBitmap(bitmap: Bitmap, imageView: ImageView): void
toBitmap(contentResolver: ContentResolver, fileUri: Uri): Bitmap
getRealPathFromURI(contentResolver: ContentResolver, contentUri: Uri): String
scaleBitmap(originalBitmap: Bitmap): Bitmap
Domain
<Interface>
AuthService
setAuthCredentials(credentials: AuthCredentials): void
getAuthCredentials(): AuthCredentials
loadLocalAuthCredentials(): void
clearAuthCredentials(): void
getLocalValidAuth(): Boolean
ApiServiceImpl
ApiRequestExecutor
execute(request: function<T>): T
Implementa
AuthServiceImpl
SharedPreferences
AuthCredentials
userKey: String
userName: String
password: String
ModelMapper
toLogin(response: LoginResponse): Login
toRegister(response: RegisterResponse): Register
toBoard(response: BoardResponse): Board
toBoardList(response: List<BoardResponse>): List<Board>
toMostUsedTagsBoards(response: List<TagBoardsResponse>): List<TagBoards>
toProfile(response: ProfileResponse): Profile
toPost(response: PostResponse): Post
ApiEndpoints
login(body: LoginRequest): Response<LoginResponse>
register(body: RegisterRequest): Response<RegisterResponse>
getProfile(apiKey: String): Response<ProfileResponse>
updateProfileIconUrl(apiKey: String, body: UpdateProfileIconUrlRequest): Response<ProfileResponse>
postBoard(apiKey: String, body: PostBoardRequest): Response<BoardResponse>
getBoard(apiKey: String, boardId: Int): Response<BoardResponse>
getAllBoards(apiKey: String): Response<List<BoardResponse>>
getBoardListOfMostUsedTags(apiKey: String): Response<List<TagBoardsResponse>>
getProfileBoards(apiKey: String): Response<List<BoardResponse>>
postPost(apiKey: String, body: PostPostRequest): Response<PostResponse>
likePost(apiKey: String, body: LikePostRequest): Response<Unit>
dislikePost(apiKey: String, body: LikePostRequest): Response<Unit>
Responses
Requests
FireBaseStorageServiceImpl
updateImageAnymously(dataBitmap: Bitmap, imagePath: String, eventFunction: Function(Boolean)): void
runWhenFileExist(path: String, function: Function()): void
launchAnonymously(function: Function(Boolean)): void
generatePath(): String

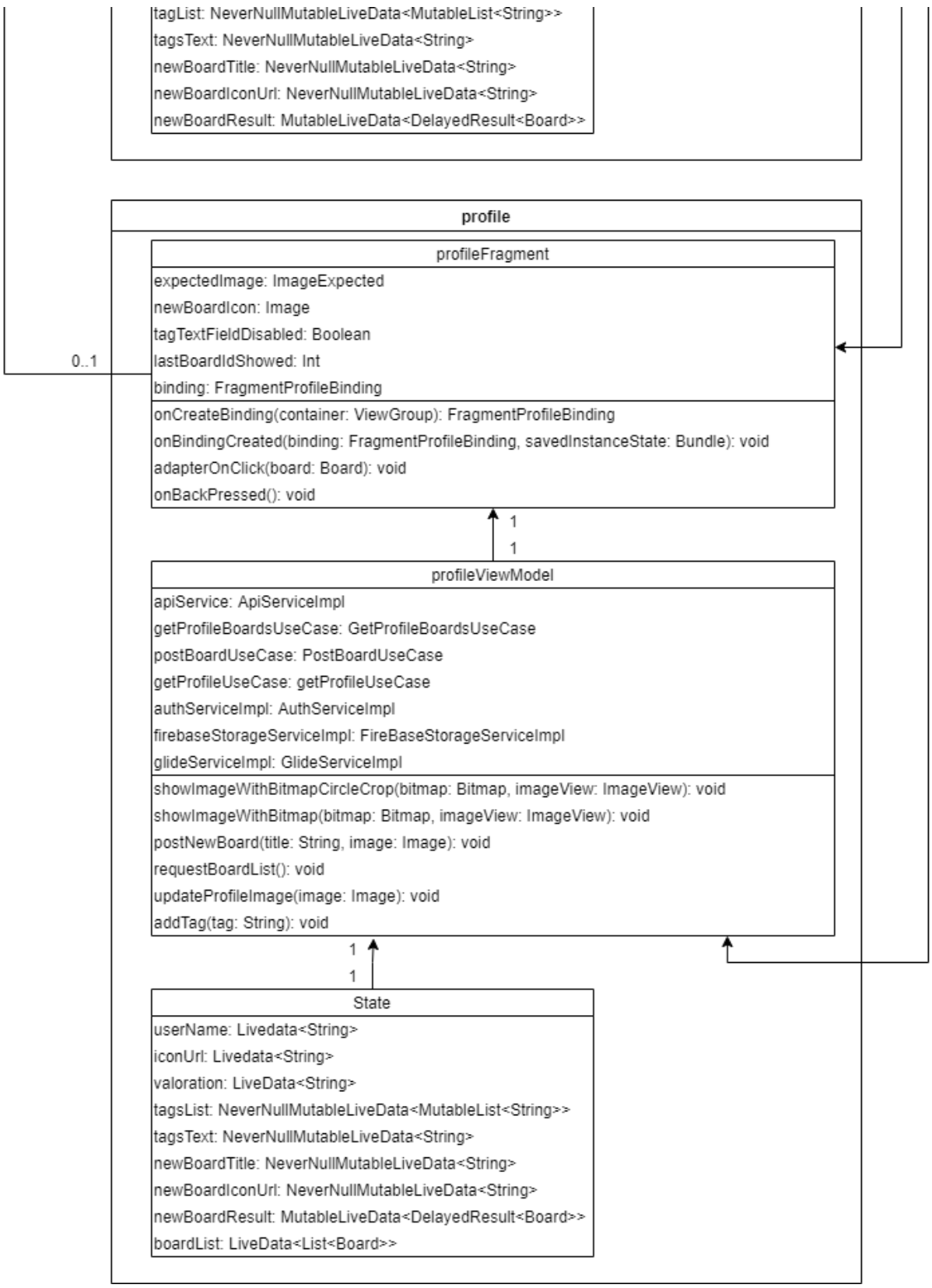

provider
ApiModule
provideApiService(modelMapper: ModelMapper, executor: ApiRequestExecutor,endpoints: ApiEndpoints, authServiceImpl: AuthServiceImpl): ApiServiceImpl
provideApiEndpoints(retrofit: Retrofit): ApiEndPoints
provideAppModelMapper(): ModelMapper
provideApiRequestExecutor(): ApiRequestExecutor
provideRetrofit(context: Context, client: OkHttpClient): Retrofit
provideOkHttpClient(): OkHttpClient
DataModule
provideAuthService(sharedPreferences: SharedPreferences): AuthServiceImpl
provideSharedPreferences(context: Context): SharedPreferences
provideFireBaseStorageService(): FireBaseStorageServiceImpl
provideGlideService(context: Context, firebaseStorageServiceImpl): FireBaseStorageServiceImpl
AppGlide
registerComponents(context, Context, glide: Glide, registry: Registry): void
Domain
api
<Interface>
ApiService
login(username: String, password:String): Login
getBoard(boardId: Int): Board
getProfile(): Profile
postBoard(title: String, tags: List<String>, iconUrl: String): Board
register(email: String, username: String, password: String): Register
postPost(boardId: Int, resourceUrl: String): Post
likePost(postId: Int): void
disLikePost(postId: Int): void
getAllBoards(): List<Board>
getBoardsListOfMostUsedTags(): List<TagBoards>
getProfileBoards(): List<Board>
updateProfileIconUrl(iconUrl: String): void
<Interface>
AuthService
setAuthCredentials(credentials: AuthCredentials): void
getAuthCredentials(): AuthCredentials
loadLocalAuthCredentials(): void
clearAuthCredentials(): void
getLocalValidAuth(): Boolean
1
all use
1
model
Image
tagName: String
boardList: List<Board>
1
*
Board
id: Int
title: String
tags: List<String>
iconUrl: String
valoration: Float
postList: List<Post>
1
*
Post
id: Int
x: Float
x: Float
rotation: Int
resourceUrl: String
valoration: Float
userLikeCode: Int
Image
path: String
bitmap: Bitmap
Login
apiKey: String
Profile
userName: String
iconUrl: String
valoration: Float
Register
apiKey: String
usecase
DislikePostUseCase
invoke(postId: Int): Unit
GetAllBoardsUseCase
invoke(): List<Board>
GetBoardUseCase
invoke(boardId: Int): Board
GetProfileBoardsUseCase
invoke(): List<Board>
GetProfileUseCase
invoke(): Profile
LikePostUseCase
invoke(postId: Int): Unit
LoginUseCase
invoke(username: String, password: String): Login
RegisterUseCase
invoke(email: String, username: String, password: String): Register
PostBoardUseCase
invoke(title: String, tags: List<String>, iconUrl: String): Board
PostPostUseCase
invoke(boardId: Int, resourceUrl: String): Post
UpdateProfileIconUrlUseCase
invoke(iconUrl: String): Unit
Support
<Abstract>
UseCase
forResult(action: Function(R)) : void
executeSuspend(action: Function(R)): void
runForResult(action: Function(R)): Result
All extend


BindingAppFragment
viewBinding: ViewDataBinding
onCreateBinding(container: ViewGroup): ViewDataBinding
onBindingCreated(binding: ViewDataBindingm savebInstanceState: Bundle): void
unbind(binding: ViewDataBinding): void
onCreateView(inflated: LayoutInflater, container: ViewGroup, savedInstanceState: Bundle): ViewDataBinding
onViewCreated(view: View, savedInstanceState: Bundle): void
onDestroyView(): void
<Enum>
ImageExpected
NONE
PROFILE_ICON
NEW_BOARD
NEW_POST
Core
core
carousel
boardCarousel
BoardDiffCallback
areItemsTheSame(oldItem: Board, newItem: Board): Boolean
areCibtebtsTheSame(oldItem: Board, newItem: Board): Boolean
BoardViewHolder
itemView: View
onClick: Function(Board)
glideServiceImpl: GlideServiceImpl
currentBoard: Board
bind(board: Board): void
BoardsAdapter
glideServiceImpl: GlideServiceImpl
onClick: Function(Board)
onCreateViewHolder(parent: ViewGroup, viewType: Int): BoardViewHolder
onBindViewHolder(holder: BoardViewHolder, position: Int): void
boardCarousel
VerticalBoardDiffCallback
areItemsTheSame(oldItem: Board, newItem: Board): Boolean
areCibtebtsTheSame(oldItem: Board, newItem: Board): Boolean
BoardViewHolder
itemView: View
onClick: Function(Board)
glideServiceImpl: GlideServiceImpl
currentBoard: Board
bind(board: Board): void
unBind(): void
VerticalBoardsAdapter
glideServiceImpl: GlideServiceImpl
onClick: Function(Board)
onCreateViewHolder(parent: ViewGroup, viewType: Int): BoardViewHolder
onBindViewHolder(holder: BoardViewHolder, position: Int): void
onViewRecycled(holder: BoardViewHolder): void
postPoolView
PostItem
context:Context
attrs: AttributeSet
defStyleAttr: Int
post: Post
onViewRemoved(view: View): void
setPost(post: Post): void
setConstraints(parentView: ConstraintLayout): void
postPresentation
collectorFragment
postCollectionAdapter: PostCollectionAdapter
viewPager: ViewPager2
onCreateView(inflater: LayoutInflater, container: ViewGroup, savedInstanceState: Bundle): View
onViewCreated(view: View, savedInstanceState: Bundle): void
FragmentStateAdapter
getItemCount(): Int
createFragment(position: Int): Fragment
PostCollectionAdapter
fm: FragmentManager
lifeCycle: Lifecycle
postList: List<Post>
vm: BoardDetailViewModel
PostFragment
vm: BoardDetailViewModel
post: Post
onCreateView(inflater: LayoutInflater, container: ViewGroup, savedInstanceState: Bundle): View
onViewCreated(view:View, savedInstanceState: Bundle


Domain
Provides services
UI
Core
uses
LauncherActivity
onCreate(savedInstanceState: Bundle): void
toMain(): void
toLogin(): void
onResume(): void
onBackPressed(): void
1
launching
0..1
LaunchingFragment
onCreate(savedInstanceState: Bundle): void
onBindingCreated(binding: FragmentBoardDetailBinding, savedInstanceState: Bundle): void
1
1
LaunchingViewModel
authServiceImpl: AuthServiceImpl
1
1
State
validAuth: MutableLiveData<Event<Boolean>>
login
LoginFragment
onCreateBinding(container: ViewGroup): FragmentLoginBinding
onBindingCreated(binding: FragmentLoginBinding, savedInstanceState: Bundle): void
1
1
0..1
LoginViewModel
apiService: ApiServiceImpl
authServideImpl: AuthServiceImpl
login(): void
1
1
State
userName: NeverNullMutableLiveData<String>
password: NeverNullMutableLiveData<String>
loginResult: MutableLiveData<Event<DelayedResult<Login>>>
isLoading: LiveData<Boolean>
error: MutableLiveData<String>
register
0..1
RegisterFragment
onCreateBinding(container: ViewGroup): FragmentRegisterBinding
onBindingCreated(binding: FragmentRegisterBinding, savedInstanceState: Bundle): void
1
1
RegisterViewModel
apiService: ApiServiceImpl
authServideImpl: AuthServiceImpl
register(): void
1
1
State
email: NeverNullMutableLiveData<String>
userName: NeverNullMutableLiveData<String>
password: NeverNullMutableLiveData<String>
registerResult: MutableLiveData<Event<DelayedResult<Register>>>
isLoading: LiveData<Boolean>
error: MutableLiveData<String>


MainActivity
firebaseStorageServiceImpl: FirebaseStorageServiceImpl
glideServiceImpl: GlideServiceImpl
REQUEST_CODE: Int
currentImageUpdated: MutableLiveData<Event<DelayedResult<Image>>>
bottomBar: BottomNavigationView
onCreate(savedInstanceState: Bundle): void
setBottomBarNavigation(): void
navigatorAction(actionId: Int): void
navigatorAction(actionId: NavDirections): void
openGallety(): void
onActivityResult(requestCode: Int, resultCode: Int, data: Intent): void
initActivityPermissions(): void
onBackPressed(): void
moveAppToBackground(): void
Domain
Provides services
UI
Core
uses
1
0..1
boardDetail
BoardDetailFragment
expectedImage: ImageExpected
currentPostImage: Image
binding: FragmentBoardDeailBinding
onCreate(savedInstanceState: Bundle): void
onCreateBinding(container: ViewGroup): FragmentBoardDetailBinding
onBindingCreated(binding: FragmentBoardDetailBinding, savedInstanceState: Bundle): void
onBackPressed(): void
1
1
BoardDetailViewModel
apiService: ApiServiceImpl
getBoardUseCase: GetBoardUseCase
postBoardUseCase: PostPostUseCase
likePostUseCase: likePostUseCase
dislikePostUseCase: DislikePostUseCase
firebaseStorageServiceImpl: FireBaseStorageServiceImpl
glideServiceImpl: GlideServiceImpl
setBoard(boardId: Int): void
postPost(image: Image): void
showPostPreviewImage(bitmap: Bitmap, imageView: ImageView): void
loadPostResourceImage(url: String, imageView: ImageView): void
likePost(postId: Int): void
dislikePost(postId: Int): void
1
1
State
id: Int
title: Livedata<String>
tagsString: LiveData<String>
valoration: LiveData<Float>
iconUrl: LiveData<String>
boardResult: MutableLiveData<Event<DelayedResult<Board>>>
newPostUrl: MutableLiveData<String>


explore
exploreFragment
tagTextFieldDisabled: Boolean
lastBoardIdShowed: Int
expectedImage: ImageExpected
newBoardIcon: Image
binding: FragmentExploreBinding
onCreate(savedInstanceState: Bundle): void
onCreateBinding(container: ViewGroup): FragmentExploreBinding
onBindingCreated(binding: FragmentExploreBinding, savedInstanceState: Bundle): void
adapterOnClick(board: Board): void
onBackPressed(): void
0..1
1
1
ExploreViewModel
apiService: ApiServiceImpl
getAllBoardsUseCase: GetAllBoardsUseCase
getAllTagsBoardsUseCase: getTagsBoardsUseCase
postBoardUseCase: PostBoardUseCase
firebaseStorageServiceImpl: FireBaseStorageServiceImpl
glideServiceImpl: GlideServiceImpl
requestBoardList(): void
requestTagBoardsList(): void
addTag(tag: String): void
PostNewBoard(title: String, image:Image): void
showImageWithBitmap(bitmap: Bitmap, imageView: ImageView): void
1
1
State
boardList: Livedata<List<Board>>
tagBoardsList: Livedata<List<TagBoards>>
tagList: NeverNullMutableLiveData<MutableList<String>>
tagsText: NeverNullMutableLiveData<String>
newBoardTitle: NeverNullMutableLiveData<String>
newBoardIconUrl: NeverNullMutableLiveData<String>
newBoardResult: MutableLiveData<DelayedResult<Board>>
profile
profileFragment
expectedImage: ImageExpected
newBoardIcon: Image
tagTextFieldDisabled: Boolean
lastBoardIdShowed: Int
binding: FragmentProfileBinding
0..1


tagList: NeverNullMutableLiveData<MutableList<String>>
tagsText: NeverNullMutableLiveData<String>
newBoardTitle: NeverNullMutableLiveData<String>
newBoardIconUrl: NeverNullMutableLiveData<String>
newBoardResult: MutableLiveData<DelayedResult<Board>>
profile
profileFragment
expectedImage: ImageExpected
newBoardIcon: Image
tagTextFieldDisabled: Boolean
0..1
lastBoardIdShowed: Int
binding: FragmentProfileBinding
onCreateBinding(container: ViewGroup): FragmentProfileBinding
onBindingCreated(binding: FragmentProfileBinding, savedInstanceState: Bundle): void
adapterOnClick(board: Board): void
onBackPressed(): void
1
1
profileViewModel
apiService: ApiServiceImpl
getProfileBoardsUseCase: GetProfileBoardsUseCase
postBoardUseCase: PostBoardUseCase
getProfileUseCase: getProfileUseCase
authServiceImpl: AuthServiceImpl
firebaseStorageServiceImpl: FireBaseStorageServiceImpl
glideServiceImpl: GlideServiceImpl
showImageWithBitmapCircleCrop(bitmap: Bitmap, imageView: ImageView): void
showImageWithBitmap(bitmap: Bitmap, imageView: ImageView): void
postNewBoard(title: String, image: Image): void
requestBoardList(): void
updateProfileImage(image: Image): void
addTag(tag: String): void
1
1
State
userName: Livedata<String>
iconUrl: Livedata<String>
valoration: LiveData<String>
tagsList: NeverNullMutableLiveData<MutableList<String>>
tagsText: NeverNullMutableLiveData<String>
newBoardTitle: NeverNullMutableLiveData<String>
newBoardIconUrl: NeverNullMutableLiveData<String>
newBoardResult: MutableLiveData<DelayedResult<Board>>
boardList: LiveData<List<Board>>

# 12. Bibliografía


1. Dibb, B., and M. Foster. Loneliness and Facebook use: the role of social comparison and rumination. Heliyon 7.1 (2021)

2. Brailovskaia, Julia, Julia Velten, and Jürgen Margaf. Relationship between daily stress, depression symptoms, and Facebook addiction disorder in Germany and in the United States. Cyberpsychology, Behavior, and Social Networking 22.9 (2019): p. 610-614.

3. González, M. Qué ha pasado con Facebook: del caso Cambridge Analytica al resto de polémicas más recientes. Xataka. [Consulta 14-04-2018] (2018).

4. Nixon, Paul G., Vanessa P. Dennen, and Rajash Rawal, eds. Reshaping International Teaching and Learning in Higher Education: Universities in the Information Age. Routledge, (2021): p.7-17

5. Barbieri A. Facebook y su complicada lucha contra el terrorismo La Vanguardia (14/05/2018)

6. Cramer, S. (2017). Instagram ranked worst for young people's mental health. RSPH Special Reports on Mental Health. (19/05/2017)

7. Bastouni, Jana. Self-Objectification of "Picture Perfect" Images on Instagram. FirstYear Reader: p167.

8. AFP. Expulsión de Trump de Twitter es definitiva: directivo de la red social. Milenio. (10/02/2021)

9. Auteur. La red social más polémica 15 años de Twitter: grandes escándalos a lo largo de su historia. Clarín.com Tecnología (21/03/2021)

10. L. Wroblewski. Defining Mobile: 4-5.5 Inches, Portrait & One-Thumb. Lukew (28/04/2015)

11. S.Ingram. The Thumb Zone: Designing For Mobile Users. Smashingmagazine (19/09/2016)

12. https://www.figma.com/file/TNl3LKPtvT7ipGAfcfreqF/MyBoards