UNIVERSIDAD DE LAS PALMAS
DE GRAN CANARIA

ESCUELA DE INGENIERÍA DE TELECOMUNICACIÓN Y ELECTRÓNICA

Trabajo Fin de Grado

Plataforma software web y móvil para notificación instantánea de información basada en un sistema publicar/subscribir mediante canales con soporte para filtrado selectivo y canales privados (NotifyMe).

Titulación: Grado en Ingeniería en Tecnologías de la Telecomunicación

Mención: Telemática

Autor: Cunwang Guo

Tutor: Dr. Luis Hernández Acosta

Fecha: Agosto 2021

UNIVERSIDAD DE LAS PALMAS
DE GRAN CANARIA

ESCUELA DE INGENIERÍA DE TELECOMUNICACIÓN Y ELECTRÓNICA

Trabajo Fin de Grado

Plataforma software web y móvil para notificación instantánea de información basada en un sistema publicar/subscribir mediante canales con soporte para filtrado selectivo y canales privados (NotifyMe).

HOJA DE EVALUACIÓN

Calificación:_____

Presidente

Fdo.:

Vocal **Secretario**

Fdo.: Fdo.:

Fecha: Agosto 2021

Agradecimientos

Primero que nada, me gustaría agradecer a mi familia por estar a mi lado y ofrecerme ánimos en todo momento, de lo contrario hubiera sido mucho más complicado llegar tan lejos. A mis padres por ofrecerme los medios materiales y los ánimos necesarios para poder realizar las diferentes tareas a lo largo de la carrera. A mis abuelos, aunque no estén este país, me han proporcionado voluntad a través de videoconferencias para seguir hacia adelante.

A los amigos y compañeros, por apoyarme en los mejores y peores momentos. Horas incontables de diversión y risas aseguradas.

A mi tutor, por ayudarme a elegir y en realizar el proyecto sin mayores obstáculos.

Resumen

El uso de dispositivos móviles o Smartphones y las aplicaciones de mensajería instantánea como WhatsApp, Telegram o Twitter han crecido de forma espontánea en esta última década. Esto es debido a eficacia y sencillez que es comunicarse con otra persona o recibir notificaciones y/o noticias en cualquier parte y en cualquier momento. No obstante, aunque las aplicaciones de mensajería bidireccional son ampliamente utilizadas, no a todos nos convencen, ya sea porque algunos solamente queremos conocer la información del entorno, por ejemplo, los torneos de basquetbol de un polideportivo.

La aplicación desarrollada en Android *NotifyMe*, presenta ambas funcionalidades: comunicación bidireccional y unidireccional. La primera opción tiene interés, por ejemplo, para los padres que quieran estar al tanto de su hijo/a y del colegio. Con esta App los padres recibirán por el chat o través de notificaciones las últimas noticias del colegio como las fiestas, excursiones, eventos etcétera, y aparte de esto podrán conocer las notas de expediente y los registros de asistencia a clase de su hijo/a. Por otro lado, está la comunicación unidireccional, en la que los usuarios pueden leer las últimas publicaciones públicas de sus canales favoritos y recibir notificaciones, estos canales podría ser el nombre de un instituto, ayuntamiento, polideportivo etc.

Abstract

The use of mobile devices or smartphones and instant messaging applications such as WhatsApp, Telegram or Twitter have grown spontaneously in the last decade. This is due to efficiency and simplicity that is to communicate with another person or receive notifications and/or news anywhere and at any time. However, although bidirectional messaging applications are widely used, not all of them convince us, either because some of us only want to know the information of the environment, for example, basketball tournaments from a sports center.

The application developed in Android *NotifyMe*, presents both functionalities: bidirectional and unidirectional communication. The first option is of interest, for instance, to parents who want to be aware of their child and the school their child is attending. On the one hand, with this app parents will receive by chat or through notifications the latest news from the school such as parties, excursions, events and so on, and apart from this they will be able to know the marks and the attendance records to class of their child. On the other hand, there is unidirectional communication, in which users can read the latest public publications of their favorite channels and receive notifications, these channels could be the name of an institute, city hall, sports center etc.

Tabla de contenido

Índice de Figuras

Capítulo I. Introducción

1. Contexto y motivación

En la actualidad existe una gran cantidad de personas que emplean las aplicaciones de mensajería instantánea como WhatsApp, Telegram o Twitter. Sin embargo, una parte de ellas prefieren disponer de una herramienta que permita tanto la comunicación bidireccional como unidireccional. La comunicación unidireccional tiene interés ya sea porque solo queremos recibir notificaciones de los temas que nos interesa y nos permite la búsqueda de la información de forma más rápida y eficaz.

Las aplicaciones de comunicación unidireccional que hay en el mercado son pocas y una de las que destaca es Konvoko [1], en la cual se basará nuestro proyecto. Esta aplicación se basa en los conceptos de canales y filtros, lo que nos permite suscribirnos a los diferentes canales que existen y recibir las notificaciones de nuevas publicaciones realizadas en estos canales de forma similar a las notificaciones de WhatsApp. Estos canales están estructurados de forma jerárquica donde un canal puede tener a su vez uno o varios subcanales asociados y donde cada uno de estos canales (o subcanales) puede tener asociados varios filtros. De esta forma, un usuario puede apuntarse a las notificaciones de uno o varios canales y, luego, si quiere, filtrar sobre subcanales o filtros específicos de esos canales de forma que pueda concretar la información en la que está interesado exactamente.

Para conseguir una mayor funcionalidad en este tipo de casos también se va a permitir la comunicación bidireccional y para ello se dará la opción a los usuarios de poder crear canales privados donde será necesario registrarse previamente con la intención de conseguir acceso a dichos canales, y así, no solo poder recibir información desde el canal sino también poder enviarla.

La idea podría ser de gran utilidad, por ejemplo, en el ámbito de la educación ya sea en los colegios o institutos donde los padres puedan recibir las notificaciones de las calificaciones, tareas, asistencias y evaluaciones de sus hijos. De tal forma que permita a estos estar al tanto del progreso y necesidades de sus hijos en cada momento.

Entonces, en el caso planteado antes, los padres podrán interaccionar con el colegio de forma activa, por ejemplo, confirmando o no la asistencia a una reunión convocada por el colegio, o mediante la solicitud de alguna tutoría al colegio.

La plataforma software está compuesta de un Back-end y un Front-end que puede ser la misma web o una aplicación móvil que nos permita recibir notificaciones de todo tipo temas: deporte, educación, cultura etc. Por otra parte, también existirá la opción para la búsqueda rápida de un tema específico (basada en los conceptos de canales, subcanales y filtros ya comentados), por ejemplo, el nombre de un instituto de educación secundaria. El propósito esencial del desarrollo de esta plataforma software es la creación de una aplicación móvil que permita al usuario la opción de suscribirse a los canales que le interesa y recibir las notificaciones (todas o solo aquellas filtradas previamente) como si fueran notificaciones de WhatsApp.

Hay que indicar que para la parte Back-end de esta plataforma software se usará el conocidísimo gestor de contenidos WordPress [2], y que el objetivo de este proyecto para esta parte sólo será la configuración de este para permitir que el administrador de la plataforma pueda crear canales públicos y publicar en los mismos. Para ello se configurará WordPress de forma que se permita la clasificación de todo lo publicado usando la jerarquía de categorías, subcategorías y etiquetas propias de este gestor, y que esta jerarquización concuerde entonces con la que necesitamos para nuestro caso concreto que se basa en canales, subcanales y filtros.

Finalmente, el Back-end que se usará para la parte privada será Firebase [3] de Google que es una plataforma en la nube para el desarrollo de aplicaciones web y móvil. En ella los usuarios registrados crearán sus propios canales privados teniendo la opción de poder elegir la creación de un canal con contraseña o sin contraseña. Si el canal creado es sin contraseña quiere decir que cualquier usuario puede suscribirse, por el contrario, si es con contraseña entonces solamente aquellos usuarios que posean la contraseña podrán acceder. Esto evita que cualquier individuo se pueda suscribir al canal y visualizar la información privada, como podría ser el canal de un instituto de educación secundaria donde los profesores convocan reuniones para los padres.

2. Comparativa de Apps

2.1. Konvoko

La idea del proyecto se basa principalmente en la App Konvoko y en las Apps de mensajería instantánea como WhatsApp, Telegram o Twitter focalizado en el chat en grupo.

Konvoko es una App móvil tanto para Android como para iOS que se basa en el concepto de canales y subcanales, en el cual el usuario (por defecto, ya que no existe registro de usuario) puede suscribirse a los diferentes canales que proporciona la App. Un canal podría ser el nombre de un colegio, un instituto o una universidad si estamos dentro de la categoría de Educación.

Considerando que estamos en la categoría de educación, el usuario podrá acceder a los subcanales y visualizar las noticias, por ejemplo, los eventos como podrían ser los carnavales, las olimpiadas, el periodo de prematricula etc. Por otra parte, si el usuario está realmente interesado de un canal y quiere conocer las últimas novedades, este podrá suscribirse haciendo clic en el botón *seguir* que aparece al lado del nombre del canal como se puede observar en la **Figura 1.** Una vez subscrito al canal, el nombre del botón pasará a *siguiendo* como se ve en la **Figura 2**. Por desgracia esta App no proporciona filtrado de notificaciones, es decir, si el usuario se ha suscrito a un canal y tiene habilita la notificación de ese canal recibirá todas las noticias.

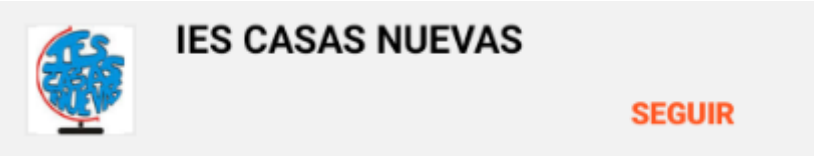


Figura 1. Canal IES Casas Nuevas (Konvoko)

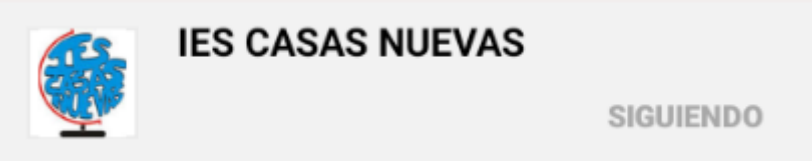


Figura 2. Canal IES Casas Nuevas *siguiendo* (Konvoko)

2.2. Dinantia

Otras de las Apps en la que se inspira nuestro proyecto es Dinantia [4], que es una plataforma de comunicación web y móvil dirigida a colegios, profesores, padres y alumnos. Esta App a diferencia de Konovoko el usuario tendrá que registrarse ya sea utilizando el correo electrónico o simplemente introduciendo el número de teléfono. Con Dinantia los padres se pueden comunicar con los profesores o el colegio de manera sencilla, en la que pueden preguntar las diferentes dudas que tengan acerca de una reunión o autorizar al hijo/a a ir a una excursión o preguntar por la asistencia de su hijo/a como aparece en la **Figura 3**. Por otra parte, los alumnos tendrán la posibilidad de utilizar la herramienta *Stop Bullying,* que permite reportar al centro cualquier caso de acoso escolar y recibir asesoramiento a través del chat.

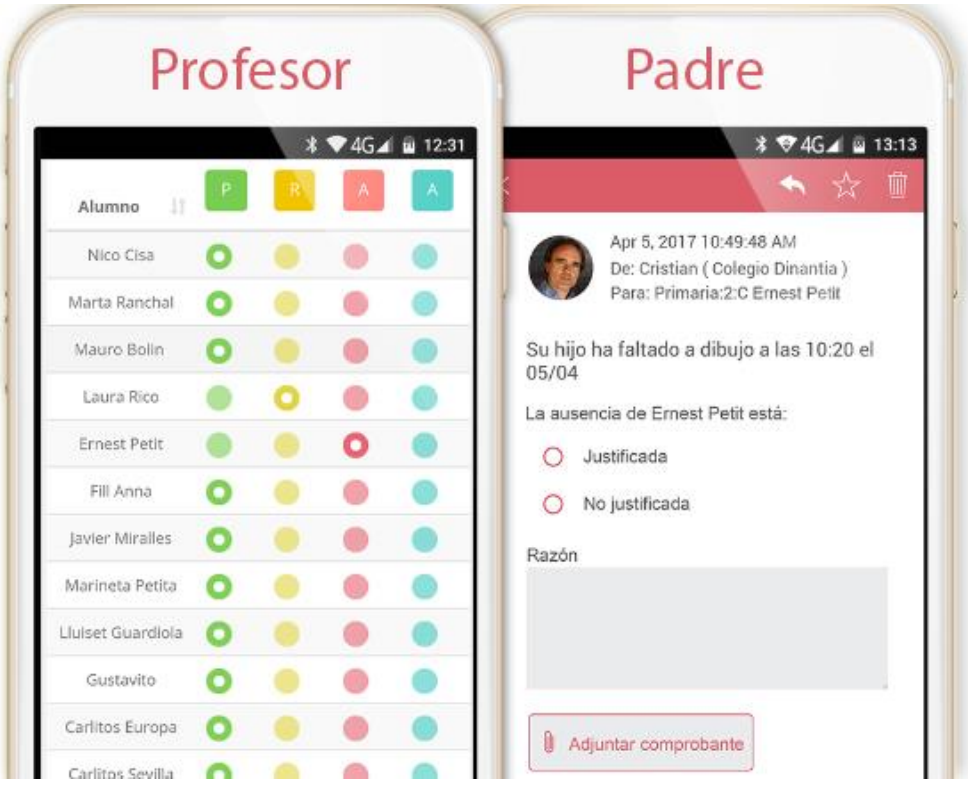


Figura 3. Control de asistencia (Dinanatia)

2.3. ECYL

Por último, la app ECYL [5] también similar a Konvoko pero centrado en el servicio público de empleo. En ella el usuario podrá habilitar las notificaciones para recibir ofertas de empleo, cursos de formación y noticias de interés sobre la formación y el empleo. En la **Figura 4** se puede observar la pantalla de inicio de la app ECYL.

Figura 4. Ofertas de empleo (ECYL)

3. Objetivos

Objetivo 1.

El objetivo principal del proyecto consiste en desarrollar una aplicación móvil para Android con el lenguaje de programación Java, que nos permita recibir notificaciones de los canales a los que estamos suscritos. En la pantalla de inicio de la App existirán

diferentes categorías como educación, deporte, cultura etc. De modo que si hacemos clic en una de ellas nos dirigirá al detalle de esa categoría, por ejemplo, en el caso de la categoría de educación tendrá como subcategorías los diferentes colegios e institutos a los cuales nos podemos suscribir dándole al botón *follow*. Una vez suscrito, aparte de poder recibir las noticias de esos canalales también se podrá visualizar estos en la sección de *following* (guardados localmente), y si pulsamos en una de ellas nos llevará al detalle permitiéndonos leer todas las noticias existentes de ese canal.

Objetivo 2.

Desarrollar una comunicación bidireccional, en la que los usuarios deberán registrarse desde el dispositivo móvil para poder suscribirse a los canales privados. En estos canales todos los usuarios que sean miembros del grupo, es decir que están suscritos, podrán enviar mensajes de forma similar a los grupos de WhatsApp.

Objetivo 3.

Emplear el gestor de contenido WordPress para crear posts, categorías, tags y otros de forma sencilla, para posteriormente extraer estas publicaciones en formato JSON a través de la API REST de WordPress y acceder a él desde el móvil.

Objetivo 4.

Desplegar WordPress (WP) fuera del ámbito local utilizando una infraestructura PaaS como Heroku [6] y gestionar las imágenes de WP mediante el almacenamiento en la nube Amazon S3 ASW [7].

Objetivo 5.

Dotar la aplicación móvil de una arquitectura MVP (Modelo, Vista y Presentador), para conseguir una mayor escalabilidad y mantenibilidad del código en el futuro.

4. Estructura del documento

El documento de este Proyecto Fin de Carrera está dividido en 8 capítulos:

- **Capítulo I. Introducción.** En este primer capítulo se introducirá el tema a tratar justificando su necesidad y utilidad, así como el back-end y front-end a utilizar en base a unos objetivos. Asimismo, se realizará una comparativa de las aplicaciones móviles similares a la App a desarrollar.

- **Capítulo II. Tecnologías Software.** En este segundo capítulo se explica las tecnologías software empleadas para el desarrollo del proyecto.

- **Capítulo III. Análisis.** En este capítulo se presenta las características de la aplicación móvil a lograr y los back-ends a emplear.

- **Capítulo IV. Diseño.** En el capítulo cuatro se explica los mockups desarrollados, el diseño de las bases de datos y el funcionamiento de una arquitectura limpia MVP.

- **Capítulo V. Implementación.** En este capítulo se explica cómo se ha desarrollado cada una de las bases de datos y la aplicación móvil utilizando una arquitectura limpia.

- **Capítulo VI. Guía de usuario.** En este capítulo se explica de forma precisa la interfaz de usuario de cada una de las pantallas.

- **Capítulo VII. Conclusiones.** En este capítulo se explica las conclusiones obtenidas al desarrollar el proyecto y las posibles líneas futuras.

- **Capítulo VIII. Presupuesto.** En el capítulo ocho se detalla el presupuesto necesario para la realización de Proyecto Fin de Grado.

- **Capítulo IX. Bibliografía.** En este capítulo se detalla las bibliografías utilizadas.

- **Capítulo X. Anexos.** En este último capítulo se explica cómo desplegar WordPress fuera del ámbito local, el pliego de condiciones y el detalle de las clases implementadas.

Capítulo II. Tecnologías software

En este segundo capítulo se explicará las tecnologías software utilizadas.

1. Android Studio

Android Studio [8] es el entorno de desarrollo de Apps para Android basado en IntelliJ IDEA [9], que nos ofrecerá un editor de códigos eficaz y las herramientas para desarrolladores de IntelliJ, además de una variedad de funciones para aumentar la productividad al crear Apps para dispositivos móviles Android.

IntelliJ IDEA, es un entorno de desarrollo integrado escrito en Java para elaborar programas informáticos. Fue creado por JetBrains [10], una compañía checa de desarrollo de software que fabrica herramientas para desarrolladores de software y gestores de proyectos.

La realización del proyecto se ha desarrollado utilizando el lenguaje de programación de propósito general Java [11]. Este lenguaje de programación fue comercializado por primera vez en 1995 por Sun Microsystems, una compañía americana encargada en vender ordenadores, componentes de ordenador, softwares y servicios tecnológicos de la información.

El lenguaje Java está basado en clases y orientado a objetos, diseñado para tener menores dependencias de implementación. Es un lenguaje rápido, seguro y confiable, por tanto, es ampliamente utilizado para el desarrollo de aplicaciones móviles, centros de datos, consolas de videojuegos etc.

2. GitKraken

Durante la realización del trabajo se ha utilizado el software GitKraken [12] para guardar el proyecto en el repositorio en línea GitHub [13]. En ella podemos visualizar todas las actualizaciones del proyecto que se han ido realizado en formato de “commits” [14]. Un “commit” es una operación que envía los últimos cambios del código fuente al

repositorio, haciendo que estos cambios formen parte de la revisión principal del repositorio.

Normalmente en un repositorio en línea participan más de una persona, pero en este caso solo se ha utilizado para guardar las diferentes versiones del proyecto y poder acceder a ella desde cualquier lugar e incluso compartirlo con otras personas que estén interesadas.

3. Figma

Antes de empezar a implementar la aplicación es necesario realizar un boceto de la App también conocido por el nombre Mockup para visualizar el aspecto que tendría esta. Esto es esencial a la hora de desarrollar una aplicación ya sea móvil o web, dado que nos ofrece una visión general de cómo podría ser nuestra App y ver las diferentes partes como las pantallas que tendrá, las interfaces de usuario, y las distintas funcionalidades.

La herramienta Figma [15] nos permite realizar exactamente esto, podemos crear todo tipo de trabajo de diseño gráfico desde wireframing de sitios webs, diseños de interfaces de aplicaciones móviles, diseños de prototipos y mucho más de una forma sencilla e intuitiva.

4. Trello

Una vez que tengamos realizado el boceto o mockup es necesario también realizar una planificación de las tareas a desarrollar. La herramienta que utilizaremos para este fin se conoce como Trello [16] y con el podemos gestionar las tareas a realizar de forma clara, ordenada y eficiente. Este tablero de tareas tiene un nombre especial conocido como tablero Kanban.

En la **Figura 5** se muestra una estructura básica de la tabla de tareas en Trello:

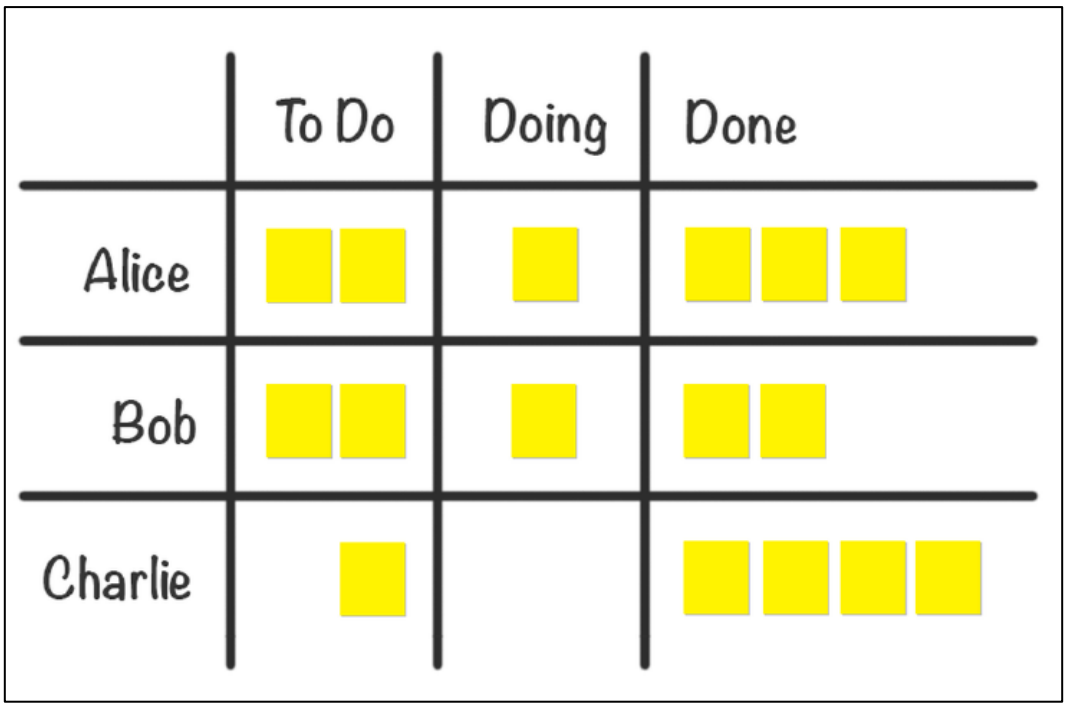


Figura 5. Tablero Kanban básico

Está formado por tres columnas principales:

- **To do.** Aquí se añadirán todas las tareas a realizar, desde el diseño de la interfaz de usuario hasta la realización de tests de implementación. Una vez que empecemos a realizar la tarea, esta pasará a la columna **Doing**.
- **Doing.** En esta segunda columna se presentarán las tareas que se están realizando en el momento. Cuando finalicemos con una tarea, esta pasará a la columna **Done**.
- **Done.** En la última columna se registrarán todas aquellas tareas finalizadas.

Pueden existir también otras columnas, por ejemplo, una columna en la que se propone nuevas ideas, funcionalidades u otra columna en la que se añaden las ideas descartadas.

5. Postman

Postman [17] es una herramienta dirigida a desarrolladores que nos permite realizar peticiones HTTP a cualquier API. Es una herramienta bastante útil, dado que nos ofrece la posibilidad de comprobar el correcto funcionamiento de nuestros proyectos de forma simple y rápida. Postman es usada por más de 7 millones de desarrolladores y más de 300 000 empresas en todo el mundo.

Si entramos en un poco más de detalle, la plataforma de colaboración Postman permite a los usuarios diseñar, simular, depurar, probar, documentar, supervisar y publicar una API y todo esto en un mismo lugar. Además, las aplicaciones nativas de Postman para MacOS, Windows, Linux ofrecen funciones aún más avanzadas.

En nuestro proyecto utilizaremos esta herramienta para dos propósitos: recuperar de la REST API [18] los canales, subcanales, tags y demás, y para crear notificaciones push enviando peticiones POST a la API de Firebase Cloud Messaging [19]. En el primer caso se ha empleado para comprobar la estructura de la REST API y ver el formato del JSON y en el segundo caso para testear las notificaciones push en la aplicación móvil.

Capítulo III. Análisis

1. Descripción general

La aplicación móvil *NotifyMe* ha sido desarrollada para dispositivos móviles Android y tiene como propósito ofrecer al usuario información útil sobre las noticias que se publican en los diferentes canales. Estos canales pueden ser el nombre de un colegio, ayuntamiento, polideportivo etcétera, a los que se puede suscribir el usuario y recibir notificaciones. Estas notificaciones, se puede filtrar por *tags* de forma que solo se reciba aquellas noticias que sean de interés para el usuario.

Existirá una sección pública y una sección privada. En la sección pública cualquier usuario puede acceder ya sea un usuario registrado o invitado. En esta sección se muestra todos los canales públicos, así como las noticias de los diferentes colegios, institutos, ayuntamientos etc. Cada uno de los canales públicos tendrá a su vez varios subcanales a los que el usuario puede acceder y visualizar la información que hay en ella y podrá también compartir la noticia simplemente dándole al botón *share* a sus compañeros, amigos o familiares.

Por otra parte, está la sección privada en la que solo los usuarios registrados pueden acceder. Estos canales privados están formados por un chat grupal similar a las Apps de mensajería instantánea como WhatsApp o Telegram, donde los usuarios pueden conversar enviando mensajes al grupo. Esto puede ser de gran utilidad, por ejemplo, en un canal de educación primaria o secundaria en la que los padres se comunican con el profesorado. Los padres también recibirán las últimas noticias del colegio que asiste su hijo/a, ya sea las excursiones que se vayan a realizar, eventos (carnavales, Halloween, olimpiadas...) o las notas de expediente. La idea también podría ser conveniente en el caso que queremos encontrar, por ejemplo, un grupo para realizar alpinismo (o cualquier deporte). Entonces buscamos en la categoría de *Deporte* un canal dedicado a este propósito, nos suscribimos al canal y por el chat grupal podemos concordar un día y una hora con los miembros del grupo para quedar juntos. En el peor de los casos si no podemos asistir a la actividad convocada, podremos enviar fácilmente un mensaje por chat avisando de que no podemos presentarnos debido a una incidencia que nos ha ocurrido o simplemente porque estamos enfermos.

2. Gestor de contenido WordPress

Hace algunos años crear tu propia página web era algo inviable si no teníamos conocimientos de programación. Pero hoy en día casi cualquier usuario puede crear su página web desde cero de forma simple y eficiente con WordPress. Este gestor de contenido posee miles de plugins para facilitar al usuario a la hora de construir su página web.

En nuestro caso el WordPress lo emplearemos como back-end para crear estos canales públicos como podrían ser el nombre de un colegio, instituto, ayuntamiento etcétera, tal y como habíamos mencionado anteriormente. Una vez que hayamos creado estos canales en WordPress las recuperamos desde el dispositivo móvil Android haciendo uso de la REST API de WordPress. Con esta API podemos utilizar el contenido que hemos creado en la página web de forma sencilla mediante peticiones HTTP. La información que recuperamos a través de la REST API hay que filtrarla de forma adecuada, dado que no solo incluye información que nos interesa, sino que también proporciona muchos datos irrelevantes. Esto se profundizará más en el apartado 2.1.

Para que WordPress pueda funcionar fuera del ámbito local es necesario disponer de una plataforma en la nube como Heroku. Este es uno de los PaaS (Plataform as a Service) más utilizados hoy en día sobre todo en entornos profesionales y empresariales por sus buenas características a la hora de realizar el despliegue de una aplicación. Por otra parte, los plugins que añadamos y los cambios que realicemos en el sitio de WordPress las actualizamos utilizando la consola de Heroku. Estos cambios se suben a la plataforma de Heroku en formato de “commits” explicada en el punto 2.2. del capítulo II. En el capítulo de Anexos apartado 1.3 se explica de forma más detallada este proceso.

Otros de los puntos a tener en cuenta son las imágenes, como WordPress no almacena las imágenes que añadamos a una publicación tendremos que hacerlo a través de Amazon S3 AWS. Al igual que Heroku es un servicio totalmente gratuito si no superamos un cierto umbral, por lo que es excelente para la realización de nuestro proyecto, ya que tiene pocos datos. Para facilitar el mapeo de las imágenes que utilicemos en WordPress, existe un plugin que nos guarda automáticamente todas las imágenes que empleemos en WordPress en el contenedor de Amazon S3 AWS.

Por desgracia WordPress es un poco tedioso de manejar si queremos crear canales privados o registrar usuarios. Por lo que utilizaremos Firebase de Google para este propósito.

2.1. REST API

La REST API de WordPress proporciona una interfaz para que las aplicaciones puedan interactuar con la base de datos de WordPress enviando y recibiendo datos como objetos JSON.

Se trata de un servicio orientado a desarrolladores. Proporciona acceso a los datos del contenido del sitio de WordPress e implementa las mismas restricciones de autenticación. Esto quiere decir que, si el contenido es público, será accesible por todas las personas luego es accesible a través de la REST API, mientras que si el contenido es privado solo será visible para aquellos usuarios autenticados y no se podrá recuperar la información mediante la REST API. Es por esto por lo que no podemos crear canales privados en WordPress ya que no nos permite recuperarlos con la REST API.

La REST API es la unión de dos palabras: API o Application Programming Interface y REST o Representational State Transfer. La primera de ellas se refiere a un conjunto de protocolos que se utilizan para desarrollar e integrar el software de las aplicaciones, mientras que el segundo se refiere a un conjunto de conceptos para modelar y acceder a los datos de nuestra aplicación como objetos y colecciones interrelacionados.

La REST API proporciona puntos finales (endpoints) que representan los posts, páginas, taxonomías y otros tipos de datos de WordPress. De esta forma podemos desde la aplicación móvil enviar y recibir datos JSON a estos puntos finales para consultar, modificar y crear contenido en la página de WordPress. Los endpoints [20] utilizados los veremos en el capítulo V.

Hay que tener cuenta también el aspecto de la paginación. Los sitios de WordPress pueden tener una gran cantidad de contenido y es por esto por lo que no podemos recuperar directamente todas las categorías en una sola petición. WordPress por defecto devuelve 10 ítems (categorías, tags, posts ...) por petición [21]. En la aplicación móvil se adaptará a este caso para no saturar la página de WordPress con las peticiones.

3. Firebase

Firebase de Google es una plataforma en la nube para el desarrollo de aplicaciones web y móvil de una forma efectiva, rápida y sencilla, por lo que es ampliamente utilizada por desarrolladores cuando necesitan almacenar datos en la nube o utilizar los servicios de este como la autenticación. Firebase es un conjunto de herramientas para construir, mejorar y escalar nuestra App, y estas las herramientas que proporciona cubren una gran parte de los servicios que los desarrolladores, ahorrando así horas y horas de trabajo.

En nuestro proyecto Firebase actuará como el segundo back-end y en ella guardaremos todos los canales privados, el historial del chat y demás. Estos canales privados son similares a los canales de WordPress, puesto que podemos darle al botón seguir y recibir notificaciones, sin embargo, estos canales privados no tendrán subcanales, sino que existirá un chat grupal que nos permite comunicarnos con los miembros del grupo para establecer planes y compartir información. Además, no solo realizaremos la acción de recuperar datos de Firebase como ocurría con WordPress, sino que los canales privados que cree el usuario se añadirán a Firebase y el resto de los usuarios podrá verlos.

Por su puesto que no solo utilizaremos esta plataforma para guardar canales privados, dado que un usuario que quiera acceder a los canales privados tendrá que registrarse previamente y después autenticarse. En un principio se intentó realizar esta parte privada también en WordPress, pero era complicado y tedioso, mientras que con Firebase la autenticación se ha completado de forma rápida y eficaz.

Finalmente, la aplicación móvil también dispone de notificaciones push y para ello en un inicio se ha utilizado OneSignal [22] que incluso existe un plugin en WordPress para para facilitar el envío de mensajes al dispositivo Android. OneSignal es fácil de utilizar e instalar tan solo con tres líneas de código podemos empezar a recibir notificaciones, sin embargo, proporciona poco control sobre estas notificaciones en la App Android. El usuario recibirá todas las notificaciones y tampoco puede filtrarlas por los *tags*, por lo que se terminó utilizando el servicio de Firebase FCM (Firebase Cloud Messaging). Este servicio proporciona una conexión fiable y eficiente, en la que nos permite recibir mensajes en iOS, Android y Web sin coste alguno.

4. Funcionamiento

A continuación, en la **Figura 6** se explica el funcionamiento del proyecto. Como se puede observar está formado por tres módulos: WordPress, Firebase y el dispositivo móvil Android.

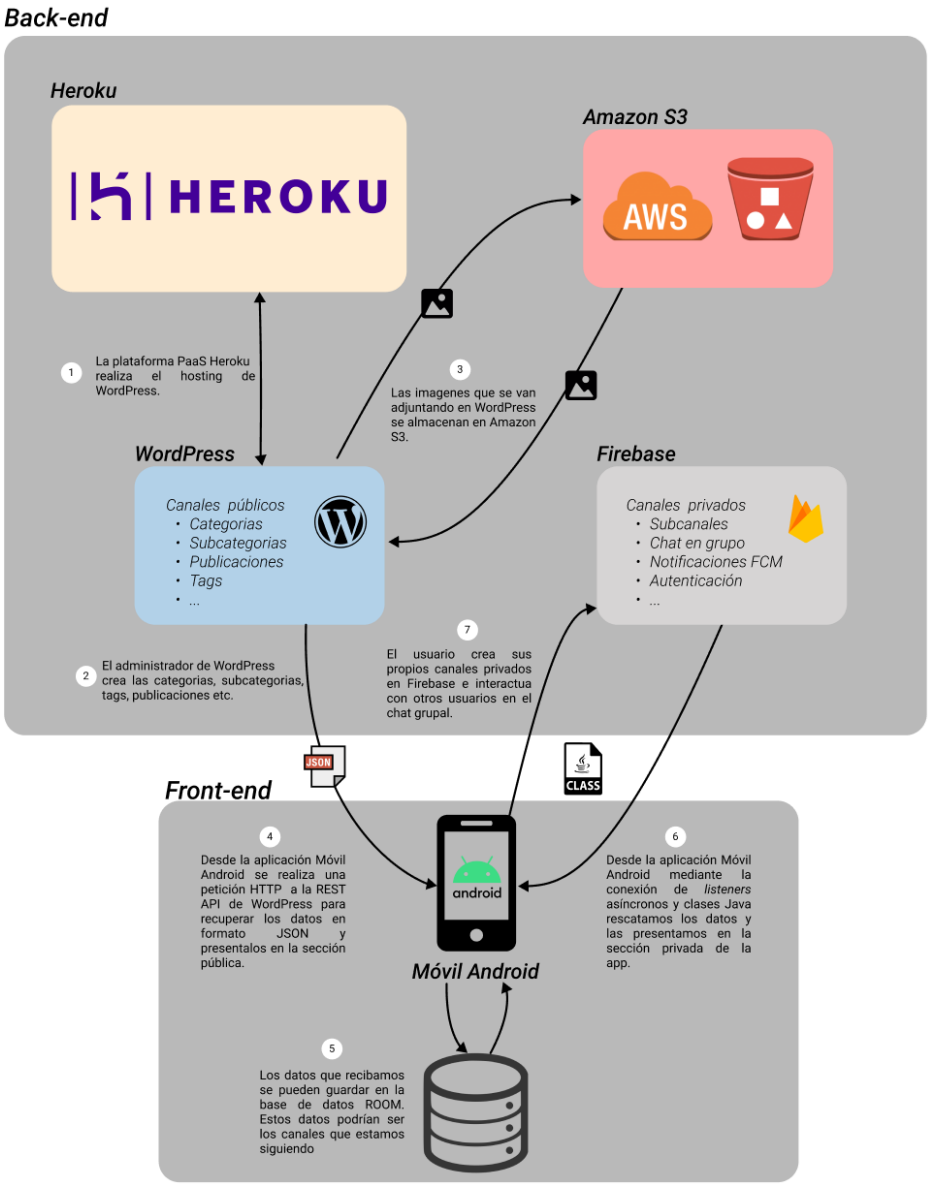


Figura 6. Ejemplo de funcionamiento

Capítulo IV. Diseño

En el capítulo 4 se mostrarán los diseños iniciales o mockups utilizando el software gratuito Figma.

1. Mockups

Como se había comentado anteriormente en el apartado de tecnologías software, para empezar a desarrollar una aplicación ya sea web o móvil es necesario empezar por unos bocetos llamados mockups. Los mockups nos ayudan a visualizar el aspecto final que tendría nuestra aplicación, así como las distintas funcionalidades va a tener, las diferentes pantallas, la interfaz de usuario y demás. A través de la herramienta de Figma, que es gratuita podemos obtener este propósito de forma cómoda y rápida.

1.1. Splash

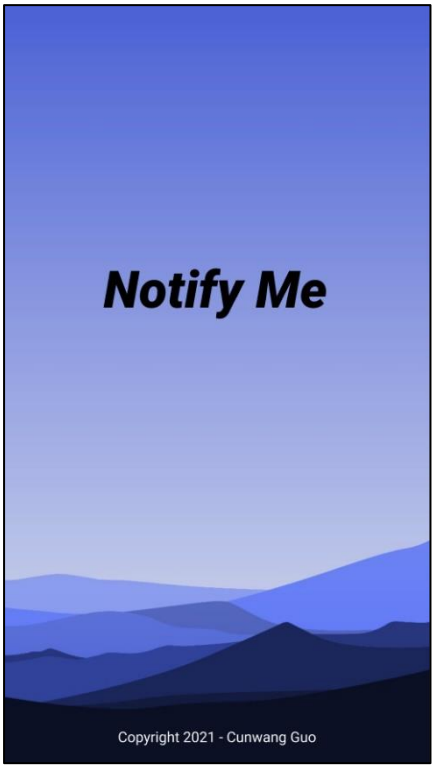


Figura 7. Pantalla Splash mockup

Esta es la pantalla inicial de la App, la que se ilustra en la **Figura 7**, en la que nos mostrará un fondo de pantalla y el nombre de la aplicación. La pantalla en si no realizará ningún tipo de acción solamente estará de ilustración.

1.2. Login y Sign up

La pantalla Login será nuestra pantalla principal después de Splash, donde el usuario podrá acceder como usuario registrado a través del botón *Login,* y que para ello tendrá que registrarse previamente haciendo clic en *Sign up*. Si el usuario no quiere registrarse, puede entrar como usuario registrado o *Enter as guest,* sin embargo, poseerá menos privilegios que un usuario registrado. En la **Figura 8** se puede observar el aspecto de la pantalla *Login*.

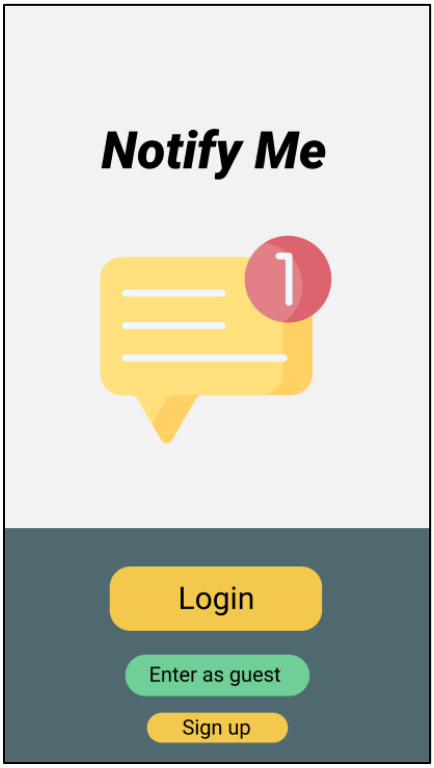


Figura 8. Pantalla Login mockup

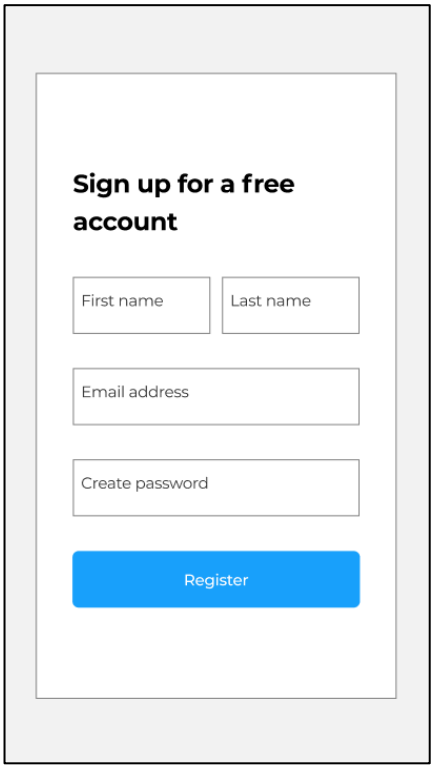


Figura 9. Pantalla Sign up mockup

Cuando el usuario haga clic en *Sign up* desde la pantalla *Login*, se abrirá una nueva vista formada por un formulario tal y como se muestra en la **Figura 9**. Éste tendrá que introducir su nombre, apellido, email y una contraseña. Una vez que se registre satisfactoriamente se redirigirá automáticamente a la pantalla *Login*.

1.3. Index

Una vez que el usuario acceda a través de *Login* o *Enter as guest* pasara a la pantalla *Index,* la que se muestra en la **Figura 10**. En esta pantalla el usuario se encontrará con una lista de las categorías principales como educación o deporte en la que podrá hacer clic y dirigirse a la subcategoría de dicha categoría. Además, tendrá un Toolbar formado por un menú, un buscador de canales y un botón con forma de campana. Este último se utiliza para visualizar las noticias no leídas.

En la **Figura 11** se aprecia el menú desplegado por el usuario como *guest* y como vemos solamente podrá acceder a los canales públicos a los que se ha suscrito y las notificaciones públicas. También existirá una opción para salir de la cuenta y volver a la pantalla *Login*.

Figura 10. Pantalla Index mockup

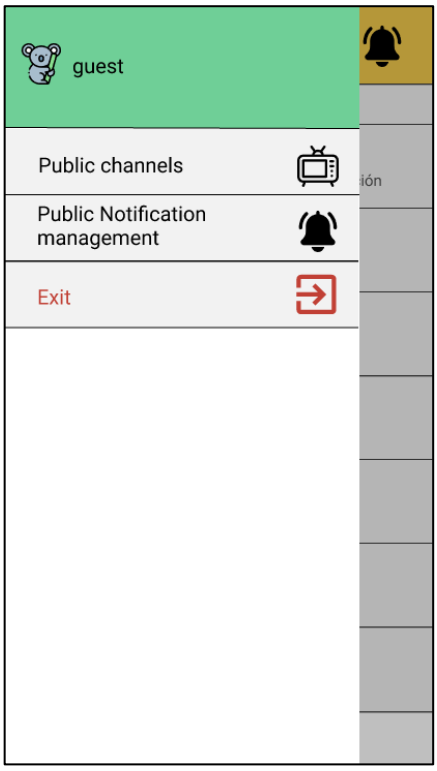


Figura 11. Pantalla Index mockup menú desplegado (guest)

Si el usuario accede como usuario registrado, se le mostrará también la sección privada en la barra de navegación lateral. Esto se puede contemplar en la **Figura 12**.

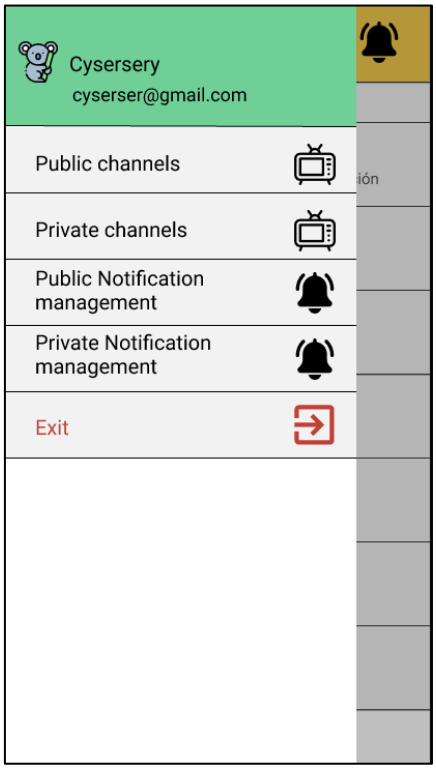


Figura 12. Pantalla Index menú desplegado (user)

1.4. Subcategory

En la pantalla *Subcategory* aparecerán todos las subcategorías o canales de una categoría. Cada ítem de la lista estará formado por una imagen, el nombre del canal y un botón *follow*. Esto se ve ilustrado en la **Figura 13**.

El usuario podrá acceder a las publicaciones simplemente pulsando sobre el canal deseado. Además, se podrá suscribir a los canales de interés a través del botón *follow* y así recibir notificaciones de las últimas noticias.

Al igual que la pantalla de *Index* también tendrá un Toolbar con un buscador de canales, un botón con forma de campana para visualizar las últimas notificaciones y una flecha para volver a la pantalla anterior.

Figura 13. Pantalla SubCategory mockup

1.5. Notification history

Figura 14. Pantalla Notification history mockup

Si el usuario quiere conocer las últimas notificaciones de los canales en los que está suscrito puede hacer clic en el icono en forma de campana que aparece en el Toolbar.

Esta vista está formada por dos secciones: una para las notificaciones públicas y otra para las privadas. Una vez que hayamos terminado de leer podremos eliminar el aviso pulsando en el botón *dismiss* tal y como se observa en la **Figura 14**.

En el Toolbar existirá un botón de *back* para volver a la pantalla anterior.

1.6. Subcategory detail

La vista *Subcategory detail* se encargará de mostrar al usuario todos los subcanales de un canal público. Cada ítem de la lista está formado por una imagen y el título de la publicación.

Al igual que las demás vistas, tendrá un Toolbar con un botón para retroceder, un buscador de canales y un icono en forma de campana para visualizar las últimas noticias.

Figura 15. Pantalla Subcategory detail mockup

1.7. Posts

Figura 16. Pantalla Posts mockup

En esta pantalla se mostrará al usuario el contenido del ítem seleccionado en la **Figura 15**. Cada publicación está compuesta, por una imagen, la misma que aparece en la pantalla *Subcategory* (pero en un formato más grande), la fecha en la que publicó, el título y la descripción de la noticia. Esto se puede observar en la **Figura 16**.

En la descripción no solamente se incluye texto, sino que puede existir hiperenlaces para acceder a un PDF, una localización o la misma publicación (en la web).

1.8. Public channels

La vista *Public channels* que aparece en la **Figura 17** mostrará al usuario registrado o como invitado los canales públicos que este está siguiendo. Los ítems de la lista son exactamente iguales que los ítems de la lista de la pantalla *Subcategory* salvo que ahora el texto del botón aparece como *unfollow* para darse de baja.

Si pulsamos en uno de los ítems accederemos al detalle de ese canal al igual que en la vista *Subcategory*.

Figura 17. Pantalla Public channels mockup

1.9. Private channels y Create channel

Figura 18. Pantalla Private channels mockup

Si nos damos de alta como usuario registrado se nos presentará la en la barra de navegación lateral la opción *Private channels*. En ella, existirá dos secciones: una sección donde aparecerán todos los canales privados que estamos siguiendo y otra sección, en la se presentarán aquellos canales creados por el usuario. Esto se puede observar en la **Figura 18**.

De forma similar a los canales públicos el usuario podrá darse de baja de aquellos canales creados por otros usuarios. Por otro lado, el usuario tiene la opción de crear sus propios canales pulsando en el botón *Create channel* y además podrá eliminar estos canales cliqueando en el icono de papelera.

Una vez que el usuario pulse en *Create channel* se presentará la pantalla que se muestra en la **Figura 19** donde este tendrá que introducir el nombre, el tema y la descripción del grupo. Al crearse el grupo satisfactoriamente se redirigirá automáticamente al usuario a la pantalla *Private channels*.

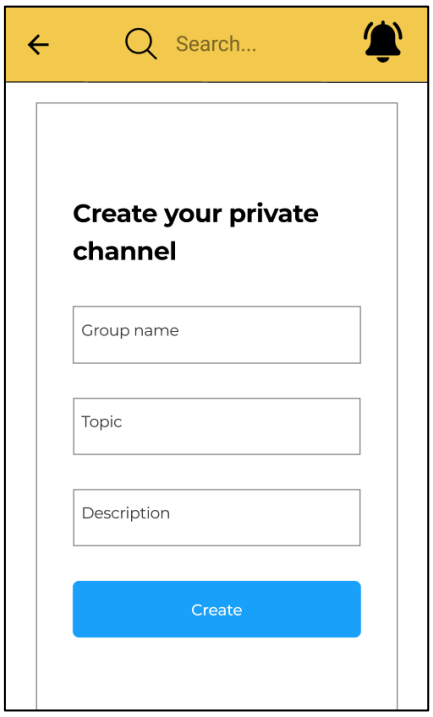


Figura 19. Pantalla Create channel mockup

1.10. Private posts y Participants

La pantalla *Private posts* está formada por un chat grupal en el que todos los miembros del grupo pueden participar mandando mensajes de texto. Para diferenciar los mensajes del propio usuario de otros se ha separado en dos tipos: mensajes con fondo de color azul y mensajes con fondo de color verde claro. Este segundo tipo serán los mensajes que nosotros mandemos al grupo y aparecerán por la parte derecha del chat tal y como se puede observar en la **Figura 20**.

Por otro lado, el usuario podrá acceder a la lista de participantes haciendo clic en el menú con tres puntos localizado en la parte derecha del Toolbar. En la **Figura 21**, se puede observar todos aquellos usuarios que están participando en el canal junto con su foto y nombre.

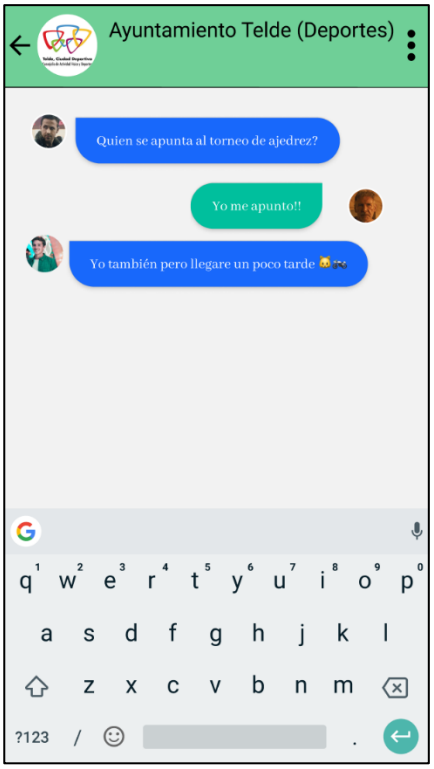


Figura 20. Pantalla Private posts mockup

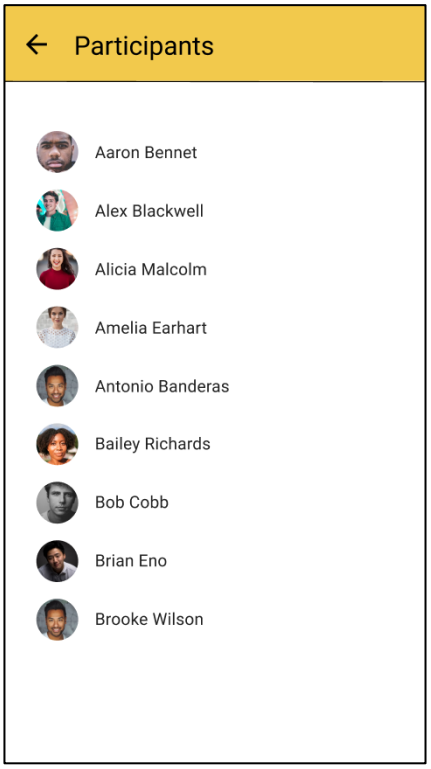


Figura 21. Pantalla Participants mockup

1.11. Public notifications y Private notifications

En la pantalla *Public notifications* el usuario podrá filtrar las notificaciones por los *tags*, como podría ser las competiciones, noticias generales, eventos, ciclismo etcétera o simplemente desactivar las notificaciones de un canal. Esto se puede observar en la **Figura 22**.

Asimismo, en la pantalla *Private notifications* se podrá filtrar tanto las notificaciones de los canales creados por el propio usuario como los canales creados por otros usuarios tal y como se puede visualizar en la **Figura 23**.

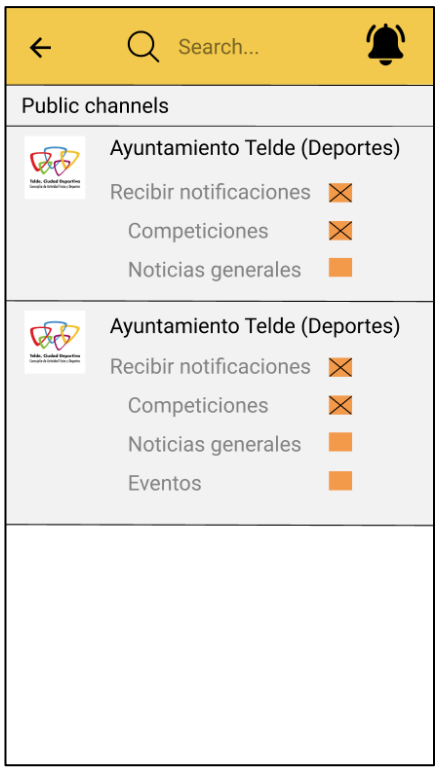


Figura 22. Pantalla Public notifications mockup

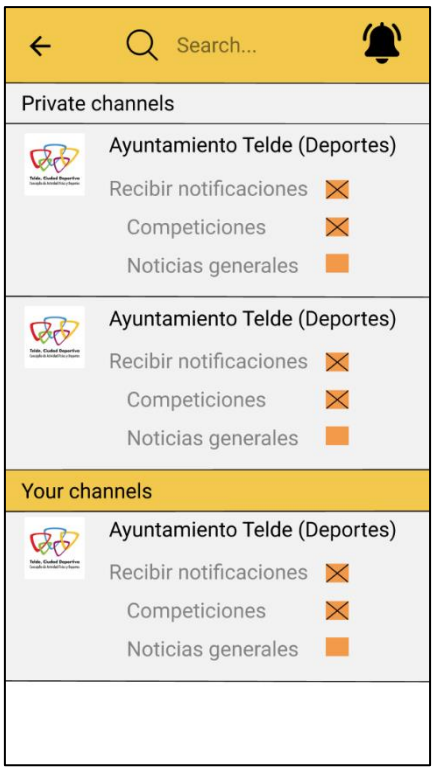


Figura 23. Pantalla Private notifications mockup

2. Diseño de las bases de datos

En este proyecto se van a utilizar tres tipos de base de datos: WordPress, Room [23] y Firebase. La primera de ellas, WordPress se empleará para crear los canales públicos, mientras que Firebase se utilizará para crear los canales privados. Por último, la base de datos Room se empleará para guardar los datos de los canales localmente en el móvil.

2.1 WordPress

La base de datos de WordPress está formada por 12 tablas que vienen por defecto y que emplea una base de datos MySQL [24]. En la **Figura 24**, se presenta un diagrama de la base de datos y sus relaciones entre ellas [25]:

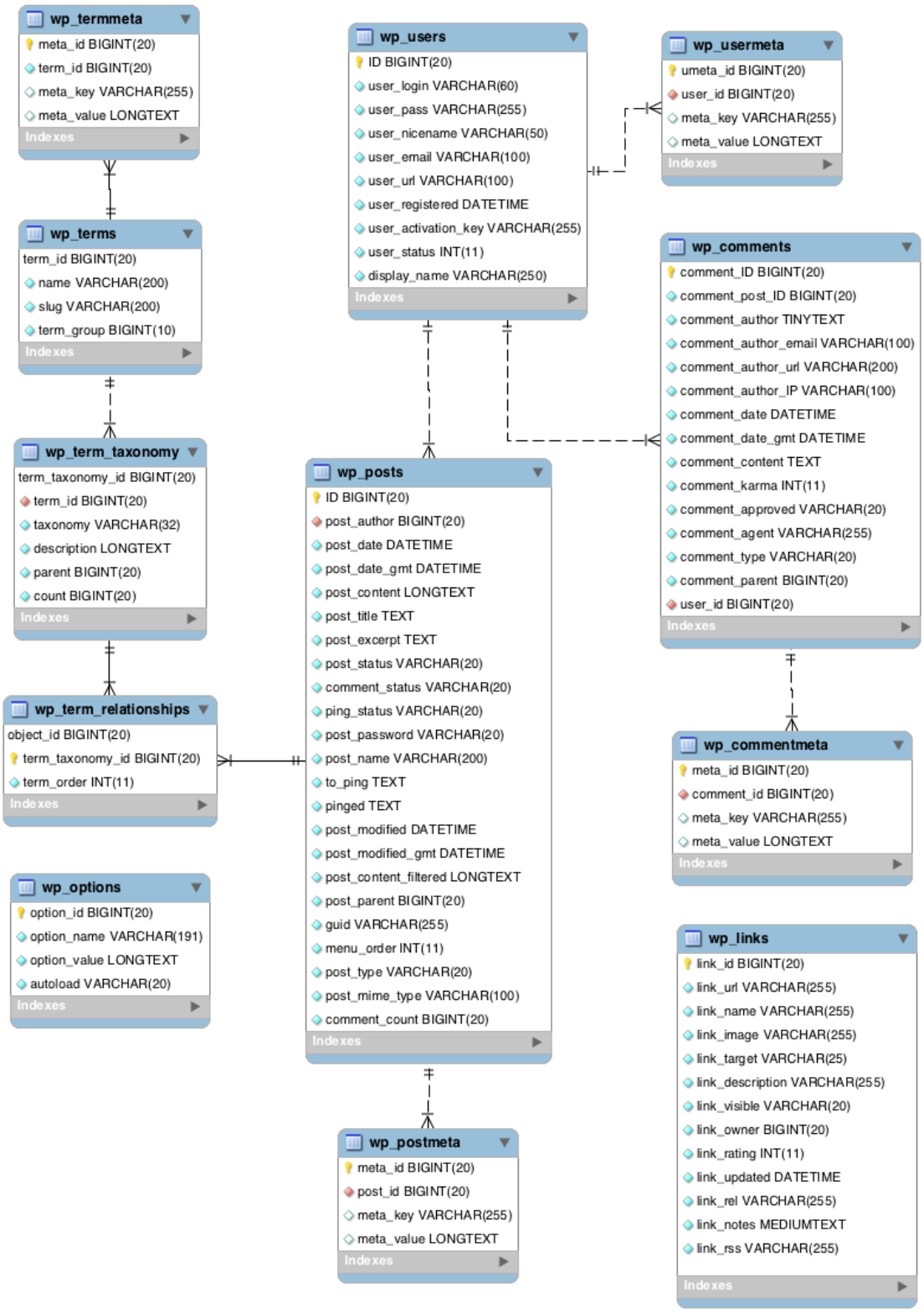


Figura 24. Diagrama de la base de datos de WordPress

2.2 Firebase

En este proyecto se va a utilizar la base de datos de Firebase Realtime Database [26]. Se trata de una base de datos NoSQL [27] (no relacional) alojada en la nube que almacena los datos en formato JSON y se sincronizan en tiempo real.

Figura 25. Firebase Realtime Database JSON

En la **Figura 25** podemos visualizar el aspecto del fichero JSON de Firebase y como vemos existe una raíz principal *Categories* de donde parten todas las categorías (que son las mismas que las de WordPress).

2.3 Room

La base de datos Room se empleará para guardar los datos de forma local en el dispositivo móvil Android. Estará formada por cinco tablas como se puede observar en la **Figura 26**. Esto se detallará más en el capítulo V.

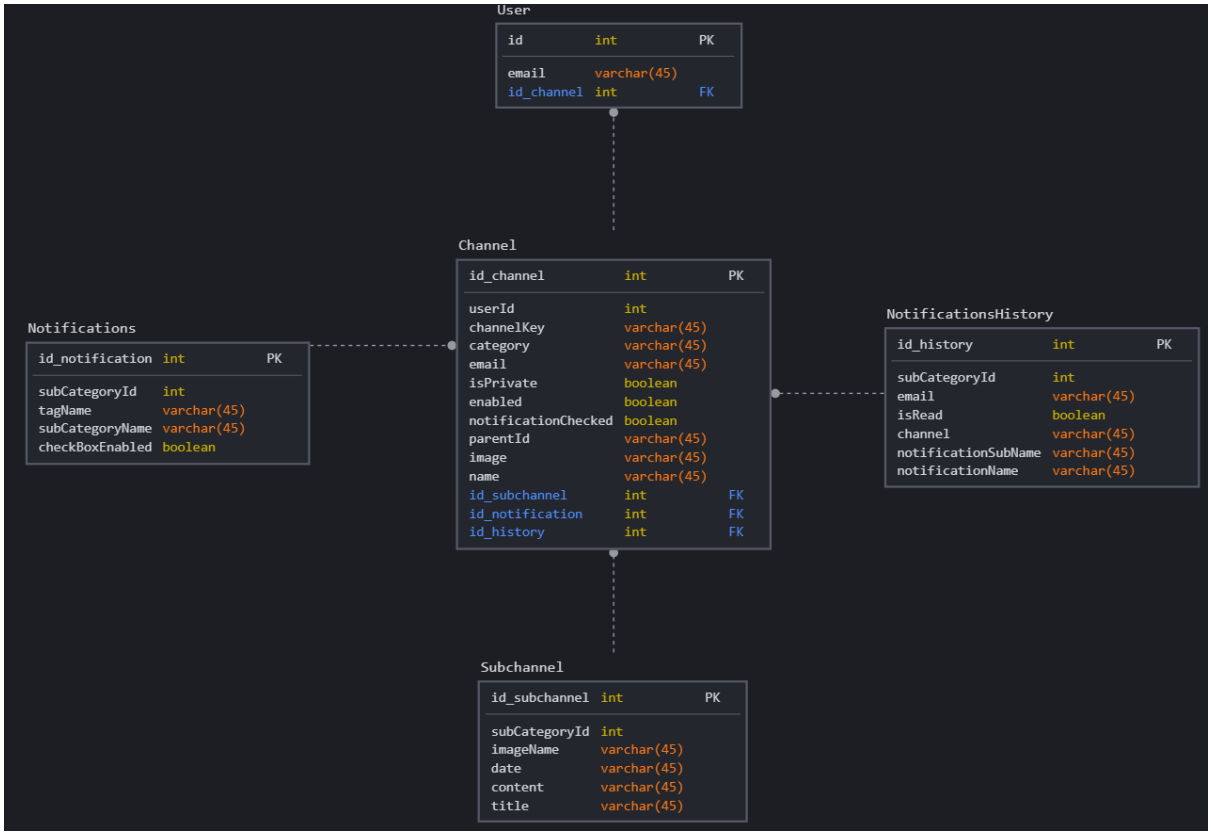


Figura 26. Tablas de la base de datos Room y sus relaciones.

3. Arquitectura modelo vista presentador (MVP)

Al diseñar una aplicación web o móvil es vital emplear un patrón de diseño software, ya que ayudan a estandarizar el código de forma que sea comprensible para otros programadores. Además de esto, nos proporciona mayor escalabilidad, robustez y mantenibilidad del código y al mismo tiempo ayudarnos a cumplir los famosos principios SOLID ilustrada en la **Figura 27**.

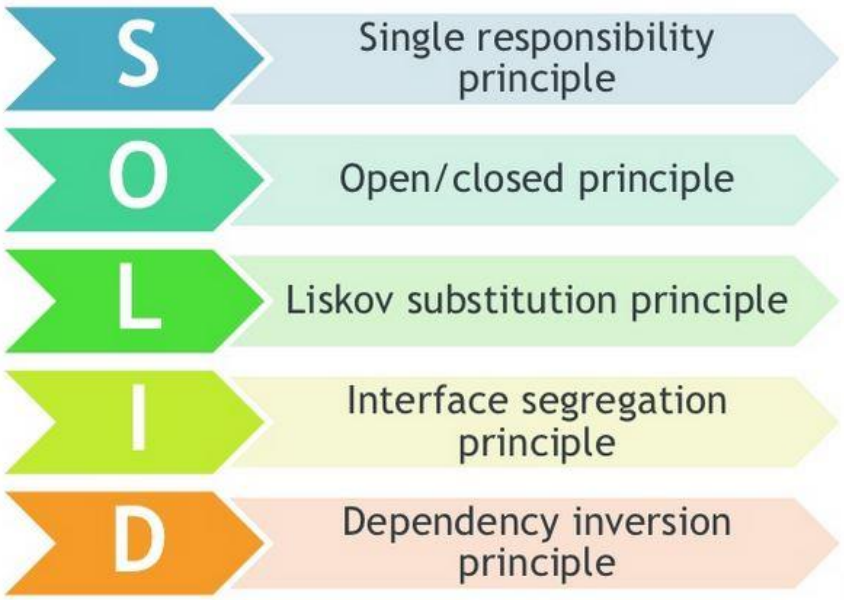


Figura 27. Principios SOLID

Para la realización este proyecto nos basaremos en el patrón de diseño *modelo vista presentador*, el cual es una de las arquitecturas de diseño software más empleadas que tiene como propósito separar la interfaz de usuario de la lógica de las aplicaciones, garantizando así el desacoplamiento entre las diferentes capas que componen una aplicación. De esta forma nos permite mantener un proyecto limpio, escalable, mantenible y lo más importante testeable.

En la **Figura 28** se muestra la arquitectura MVP formada por tres capas:

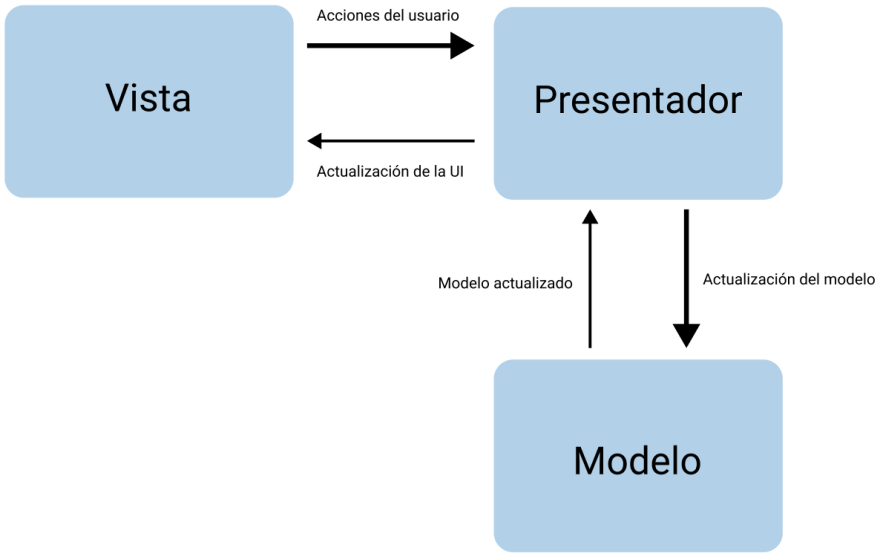


Figura 28. Arquitectura MVP

- **View:** Esta capa se encarga de mostrar los datos, y es donde se encontrarán nuestros fragmentos y vistas, es decir gestiona la interfaz de usuario.
- **Presenter:** Es el intermediario entre la vista y el modelo, con esta capa conectaremos la interfaz gráfica con los datos.
- **Model:** En esta capa es donde se lleva a cabo toda la lógica de negocio, o sea es la capa que gestiona y se almacenan los datos.

En este proyecto se empleará una variante de este patrón utilizando *clean code,* la que se muestra en la **Figura 29**, en la que se incluyen más clases aparte de la vista, el presentador y el modelo. En esta variante se incluyen las siguientes clases:

- **Screen:** Esta clase se encargará de inyectar las dependencias entre el resto de las clases.
- **Contract:** Clase que definirá las interfaces de la vista, del presentador y del modelo.
- **ViewModel:** Esta clase se encarga de guardar el estado de la vista.
- **State:** Tiene como propósito guardar los estados del modelo.
- **Mediator:** Clase singleton que se encarga de gestionar la forma en que un conjunto de clases se comunican entre sí, formando una capa de comunicación bidireccional.

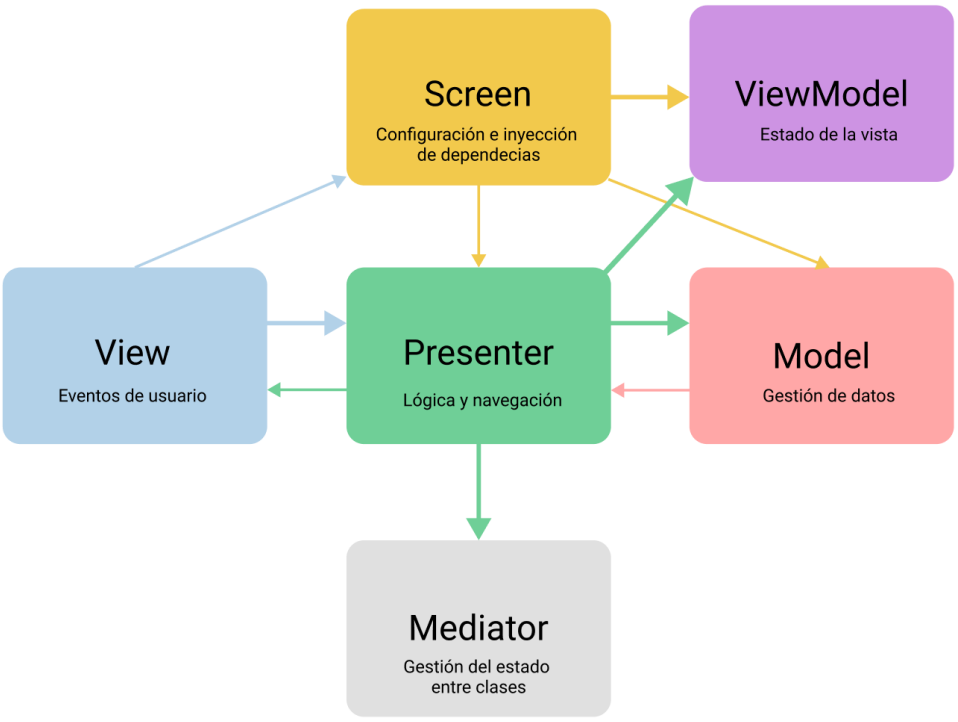


Figura 29. Arquitectura MVP – clean code

A continuación, se explicarán cada uno de los campos y métodos de la plantilla MVP.

Activity

Se encarga de la gestión de la interfaz de usuario.

Campos

presenter

private [nombre de la clase]Contract.Presenter

Define al presentador asociado a la vista.

TAG

private static String

Cadena de caracteres que define el nombre de la clase.

Métodos

inyectPresenter

public void inyectPresenter ([nombre de la clase]Contract.Presenter presenter)

Inyecta al presentador en la vista.

navigateToNextScreen

public void navigateToNextScreen ()

Navega a la siguiente pantalla.

onBackPressed

public void onBackPressed ()

Navega a la pantalla anterior.

onCreate

protected void onCreate (Bundle savedInstanceState)

Crea la actividad e inicializa las configuraciones.

onDataUpdated

public void onDataUpdated ([nombre de la clase]ViewModel viewModel)

Actualiza la interfaz de usuario.

onDestroy

protected void onDestroy ()

Llama al método padre *onDestroy* y se encarga el presentador de realizar la lógica una vez que se destruya la actividad.

onPause

protected void onPause ()

Llama al método padre *onPause* y se encarga el presentador de realizar la lógica una vez que la actividad entra en segundo plano.

onResume

protected void onResume ()

Llama al método padre *onResume* y se encarga el presentador de realizar la lógica una vez la actividad reanuda.

Contract

Interfaz se encarga de definir la comunicación entre los módulos (vista, presentador y modelo) de la arquitectura.

Interfaces

[nombre de la clase]Contract.Model

public static interface [nombre de la clase]Contract.Model

Define los métodos que deben estar presente en el modelo.

[nombre de la clase]Contract.Presenter

public static interface [nombre de la clase]Contract.Presenter

Define los métodos que deben estar presente en el presentador.

[nombre de la clase]Contract.View

public static interface [nombre de la clase]Contract.View

Define los métodos que deben estar presente en la vista.

Presenter

Esta clase se encarga de recibir y gestionar las acciones de usuario procedentes de la vista y de notificar al modelo para procesar dichas acciones.

Campos

mediator

private AppMediator

Objeto mediador empleado para pasar los estados entre las diferentes pantallas de la App.

model

private [nombre de la clase]Contract.Model

Define el modelo asociado al presentador.

state

private [nombre de la clase]State

Objeto que se encargará de almacenar el estado de la pantalla [nombre de la clase].

TAG

private static String

Cadena de caracteres que define el nombre de la clase.

view

private WeakReference<[nombre de la clase]Contract.View>

Define la vista asociada al presentador.

Constructores

[nombre de la clase]Presenter

public [nombre de la clase]Presenter (AppMediator mediator)

Inicializa el objeto mediator y actualiza la variable *state ([nombre de la clase]State)* con la información almacenada en el mediador.

Métodos

getStateFromNextScreen

private NextTo[nombre de la clase]State getStateFromNextScreen ()

Recupera los estados de la siguiente pantalla.

getStateFromPreviousScreen

private PreviousTo[nombre de la clase]State getStateFromPreviousScreen ()

Recupera los estados de la pantalla anterior.

injectModel

public void injectModel ([nombre de la clase]Contract.Model model)

Inyecta el modelo en el presentador.

injectView

public void injectView (WeakReference<[nombre de la clase]Contract.View> view)

Inyecta la vista en el presentador.

onBackPressed

public void onBackPressed ()

Indica a la vista de navegar a la pantalla anterior.

onDestroy

public void onDestroy ()

El presentador se encarga de tratar la lógica cuando la actividad se destruye.

onPause

public void onPause ()

El presentador se encarga de tratar la lógica cuando la actividad entra en segundo plano.

onRestart

public void onRestart ()

El presentador se encarga de tratar la lógica cuando la actividad regresa del modo onStop.

onResume

public void onResume ()

El presentador se encarga de tratar la lógica cuando la actividad entra de nuevo en primer plano.

onStart

public void onStart ()

Inicializa el state *([nombre de la clase]State)* si este es nulo y actualiza las variables del *ViewModel*.

passStateToNextScreen

private void passStateToNextScreen ([nombre de la clase]ToNextState state)

Llama al mediador para que actualice el estado *[nombre de la clase]ToNextState*.

passStateToPreviousScreen

private void passStateToNextScreen ([nombre de la clase]ToPreviousState state)

Llama al mediador para que actualice el estado *[nombre de la clase]ToPreviousState*.

Model

Esta clase se encargará de la lógica de negocio.

Campos

data

private String data

Variable String que se encarga de almacenar el dato.

TAG

public static String TAG

Define el nombre de la clase

Constructores

[nombre de la clase]Model

public [nombre de la clase]Model (Repository repository)

Constructor que inicializa el repositorio.

Métodos

getStoredData

public String getStoredData ()

Devuelve el valor almacenado en la variable data.

onDataFromNextScreen

public void onDataFromNextScreen (String data)

Actualiza los datos del modelo con los datos de la siguiente pantalla.

onDataFromPreviousScreen

public void onDataFromPreviousScreen (String data)

Actualiza los datos del modelo con los datos de la pantalla anterior.

onRestartScreen

public void onRestartScreen (String data)

Actualiza los datos cuando la actividad entra en modo *onRestart*.

Screen

Esta clase se encarga de configurar y de inyectar las dependencias en los diferentes módulos, es decir se encarga de crear los objetos como el mediador, repositorio, presentador etcétera e inyectarlos en las clases correspondientes.

Métodos

configure

public static void configure ([nombre de la clase]Contract.View view)

Configura e inyecta las dependencias en las diferentes clases.

State

La clase State tiene objetivo guardar el estado de las variables del modelo. Aquí es donde declararemos cada una de las variables a utilizar por el modelo.

ViewModel

La clase ViewModel tiene objetivo guardar el estado de las variables de la vista. Aquí es donde declararemos cada una de las variables a utilizar por la vista.

AppMediator

Esta clase se encarga de gestionar la forma en que un conjunto de clases se comunican entre sí. En el proyecto se encargará de guardar los estados de una pantalla para emplearlas en otra pantalla.

Campos

INSTANCE

private static AppMediator INSTANCE

Define la instancia de la clase.

m[nombre de la clase]State

private [nombre de la clase]State m[nombre de la clase]State

Define el objeto estado de una pantalla.

Constructores

AppMediator

private AppMediator ()

Constructor que crea los diferentes objetos de estado (por ejemplo, *PruebaState*).

Métodos

getInstance

public static AppMediator getInstance ()

Método que se encarga de crear la clase *AppMediator* solo si no es nulo, es decir se crea una vez solo. De esta forma todas las clases del proyecto emplean el mismo *AppMediator*.

getNext[nombre de la clase]ScreenState

public NextTo[nombre de la clase]State getNext[nombre de la clase]ScreenState ()

Obtiene el estado de la pantalla siguiente de la clase [nombre de la clase].

getPrevious[nombre de la clase]ScreenState

public PreviousTo[nombre de la clase]State getPrevious[nombre de la clase]ScreenState ()

Obtiene el estado de la pantalla anterior de la clase [nombre de la clase].

get[nombre de la clase]State

public [nombre de la clase]State get[nombre de la clase]State ()

Obtiene el estado de la pantalla [nombre de la clase].

resetInstance

public static void resetInstance ()

Restablece la instancia de la clase, es decir todo se crea de nuevo.

setNext[nombre de la clase]ScreenState

public void setNext[nombre de la clase]ScreenState ([nombre de la clase]ToNextState state)

Actualiza el estado del objeto *NextTo[nombre de la clase]ScreenState*.

setPrevious[nombre de la clase]ScreenState

public void setPrevious[nombre de la clase]ScreenState ([nombre de la clase]ToPreviousState state)

Actualiza el estado del objeto *PreviousTo[nombre de la clase]ScreenState*.

Capítulo V. Implementación

El capítulo cinco consta de dos partes: en la primera de ellas explicaremos la utilidad de cada una de las bases de datos mencionadas en el capítulo IV, y luego precisaremos en la aplicación móvil desarrollada en Android Studio.

1. Bases de datos

En esta sección nos centraremos en las bases de datos empleadas, así como la utilidad de cada una de las tablas y de sus campos.

1.1. WordPress

La base de datos de WordPress es de tipo SQL y se genera automáticamente durante la instalación. Las tablas que emplea WordPress son las siguientes:

1.1.1. Wp termmeta

Proporciona información adicional sobre la taxonomía, como podría ser si una categoría lleva un icono (imagen).

1.1.2. Wp terms

Proporciona información sobre la taxonomía, es decir las categorías, etiquetas (tags) y demás.

1.1.3. Wp term taxonomy

Esta tabla se encarga de relacionar los tipos de taxonomía entre ellos, esto se refiere a que una categoría puede tener varias etiquetas y viceversa.

1.1.4. Wp term relationship

Se encarga de relacionar la taxonomía con los tipos de posts, es decir una publicación puede tener asociado diferentes categorías y tags.

1.1.5. Wp options

En esta tabla es donde se almacenan todas las configuraciones de WordPress como la URL, el título, los plugins instalados etc.

1.1.6. Wp posts

Se encarga de almacenar todas las publicaciones y las páginas que tengan el sitio web WordPress.

1.1.7. Wp postmeta

Esta tabla contiene metainformación sobre los posts y las páginas.

1.1.8. Wp users

Contiene información de los usuarios como el nombre de *login,* apodo, la contraseña encriptada y el email.

1.1.9. Wp usermeta

Proporciona información extra (metadatos) sobre los usuarios.

1.1.10. Wp comments

Tabla que contiene información sobre los comentarios de cada post.

1.1.11. Wp commentmeta

Tabla que contiene información adicional sobre los comentarios, como podría ser las valoraciones (estrellas).

1.1.12. Wp links

Tabla que almacena información relacionada con los enlaces.

1.2.1 Categorías, posts y Taxonomía

En este apartado se comentará cómo se han creado las categorías, los canales públicos y cuáles son los plugins utilizados.

1.2.1.1 Categorías y canales

Para crear y clasificar las categorías y los canales se harán en la sección *Posts* → *Categories*. Una categoría principal es aquella categoría que no extiende de otra categoría, mientras que un canal es aquella categoría que extiende de una categoría padre. En la **Figura 30**, se muestra una tabla con las categorías y canales de WP y sus campos:

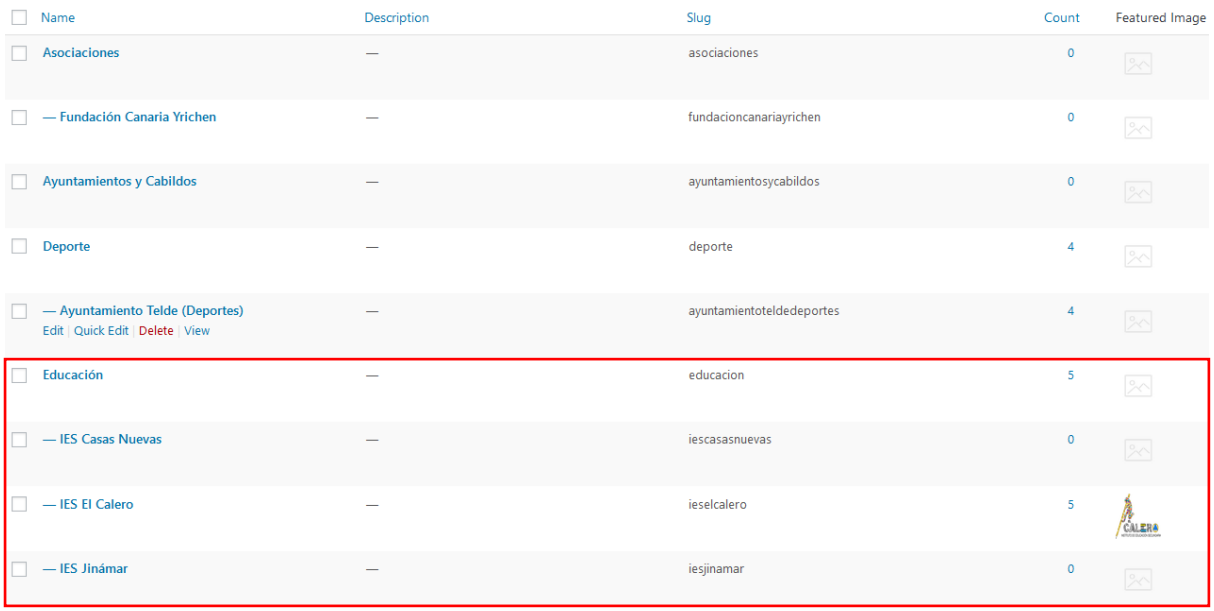


Figura 30. WordPress – Categories

Tal y como se puede observar en la **Figura 30**, la categoría educación tiene tres canales: IES Casas Nuevas, IES El Calero y IES Jinámar. Los campos que aparecen en la tabla son las siguientes:

- **Name:** representa el nombre de la categoría (o canal).
- **Description:** representa la descripción de la categoría (o canal). Aunque, en este proyecto no se harán uso de ellas.
- **Slug:** representa el texto que viene después del nombre de dominio, es decir es la versión amigable del nombre para las URLs.
- **Count:** define la cantidad de publicaciones que tiene la categoría (o canal).
- **Featured Image:** define la imagen utilizada para el canal. Esta columna solo se consigue instalando el plugin **Easy Add Thumbnail** que se explicará más en detalle en el apartado 1.2.2 de este capítulo.

1.2.1.2 Posts

Las publicaciones de los canales no se crean en las categorías sino en la sección *Posts* → *All Posts* donde se muestran todas las publicaciones de todos los canales. En la **Figura 31** y **Figura 32** se puede contemplar el aspecto de dicha sección:

Figura 31. WordPress – All Posts I

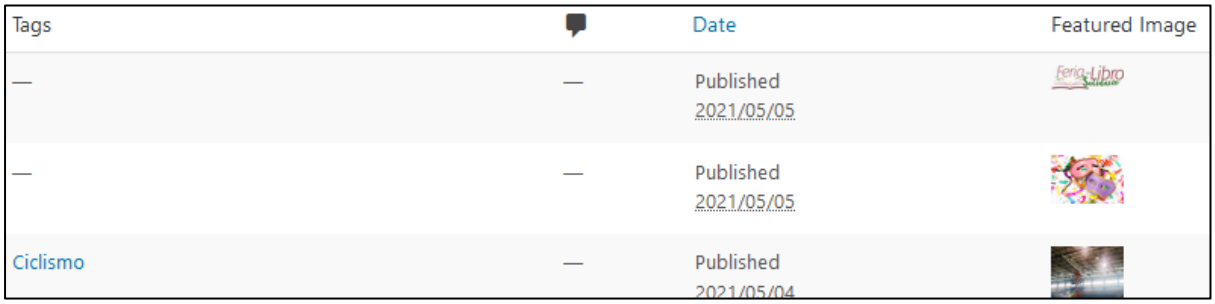


Figura 32. WordPress – All Posts II

Los campos que aparecen son los siguientes:

- **Title:** define el título de la publicación.
- **Author:** representa el autor que ha creado la publicación.
- **Categories:** indica a que categorías o canales pertenece la publicación.
- **Tags:** representa las etiquetas, aunque no serán las que utilicemos en la aplicación móvil para filtrar las notificaciones push.
- **Comments:** esta columna representa los comentarios que dejen los usuarios en la publicación.
- **Data:** representa la fecha en la que se publicó la noticia.
- **Featured Image:** Al igual que antes, define la imagen de la publicación y se logra con el plugin **Easy Add Thumbnail.**

1.2.1.3 Tags

Las etiquetas o *tags* representan los filtros de un canal en la aplicación móvil, de forma que permita al usuario personalizar las notificaciones *push*. Estas etiquetas se crean en la sección *Posts* → *Tags*.

Name ▲	Description	Slug	Count	Featured Image
Baloncesto	—	baloncesto	0	
Ciclismo	—	ciclismo	1	
Competición	—	competicion	1	
Eventos	—	eventos	2	
Fútbol	—	futbol	0	
Noticias generales	—	noticiasgenerales	4	

Figura 33. WordPress – Tags

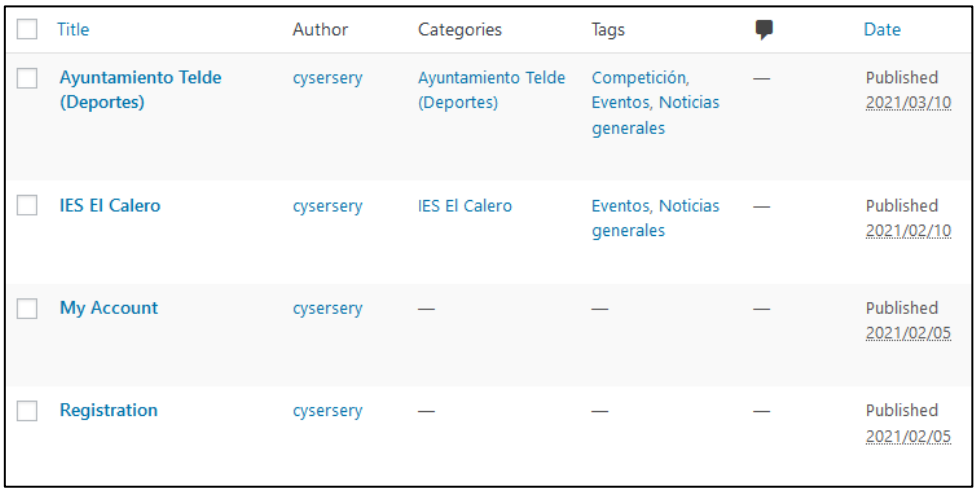
Los campos que aparecen en la **Figura 33** son:

- **Name:** define el nombre de la etiqueta.
- **Description:** define la descripción de la etiqueta.
- **Slug:** define la versión amigable del nombre para las URLs.
- **Count:** representa el número de categorías que hacen uso de la etiqueta.
- **Featured Image:** igual que antes, define la imagen de la etiqueta y se logra con el plugin **Easy Add Thumbnail**.

1.2.1.4 Pages

Las páginas o *pages* en WordPress representan un "canal" en la aplicación móvil y en ellas definimos las diferentes etiquetas y categorías. Para que se muestren las publicaciones de un canal en una página se ha empleado un plugin para generar lo que conoce como *shortcodes*, que son fragmentos de códigos que nos permite realizar funcionalidades con poco esfuerzo. Esto se detallará más en el apartado 1.2.2 de este capítulo.

En la **Figura 34**, se muestra la tabla *Pages* y sus campos:



Title	Author	Categories	Tags	💬	Date
Ayuntamiento Telde (Deportes)	cysersery	Ayuntamiento Telde (Deportes)	Competición, Eventos, Noticias generales	—	Published 2021/03/10
IES El Calero	cysersery	IES El Calero	Eventos, Noticias generales	—	Published 2021/02/10
My Account	cysersery	—	—	—	Published 2021/02/05
Registration	cysersery	—	—	—	Published 2021/02/05

Figura 34. WordPress - Pages

- **Title:** define el nombre de la página.
- **Author:** representa el autor que ha creado la página.

- **Categories:** representa las categorías que posee la página. En nuestro caso siempre se utilizará la categoría que tiene el mismo nombre que el título de la página.
- **Tags:** representa las etiquetas que tiene la página. En este caso estos tags si son las que aparecen en la aplicación móvil.
- **Comments:** representa los comentarios.
- **Date:** indica la fecha en la que se ha creado la página.

1.2.2. Plugins

Un *plugin* de WordPress es una unidad de software que contiene un grupo de funciones para aumentar la funcionalidad de nuestra página web. Los plugins que comentaremos en este apartado son esenciales para el funcionamiento correcto de nuestro proyecto.

1.2.2.1 Better REST API Featured Images

Este *plugin* es necesario para poder recuperar de la REST API de WordPress la imagen que se ha utilizado en una publicación desde la aplicación móvil. Entre la **Figura 35** y la **Figura 36** se puede contemplar la potencialidad de este *plugin* [28].

Este sería el aspecto que tendría en el JSON sin emplear el *plugin*.

```
"featured_media": 13,
```

Figura 35. WordPress REST API – Better REST API Featured Image I

Si empleamos el *plugin*, convierte el JSON de la **Figura 35** en un formato más detallado como aparece en la **Figura 36**.

```
"featured_media": 13,
"better_featured_image": {
    "id": 13,
    "alt_text": "Hot Air Balloons",
    "caption": "The event featured hot air balloon rides",
    "description": "The hot air balloons from the big event",
    "media_type": "image",
    "media_details": {
      "width": 5760,
      "height": 3840,
      "file": "2015/09/balloons.jpg",
      "sizes": {
        "thumbnail": {
          "file": "balloons-150x150.jpg",
          "width": 150,
          "height": 150,
          "mime-type": "image/jpeg",
          "source_url": "http://api.example.com/wp-content/uplo
```


Figura 36. WordPress REST API – Better REST API Featured images II

Ahora podemos ver que al final aparece una clave llamada *source_url,* de donde podemos extraer la imagen utilizada. En nuestro proyecto las imágenes se guardan en el *bucket* [29] de Amazon S3 AWS por lo que el aspecto sería como la que aparece en la **Figura 37**:


```
source_url:          "https://notifyme47bucket.s3.eu-west-3.amazonaws.com/app/uploads
                     /feriadellibrosolidario.png"
```


Figura 37. WordPress REST API – Better REST API Featured images III

1.2.2.2 Easy Add Thumbnail

Este *plugin* se encargará de añadir en las tablas *categories*, *posts* y *tags* una nueva columna llamada *Featured Image,* que nos permitirá visualizar la imagen que hemos elegido [30]. Esto es útil para el caso de las categorías (canales) dado que WordPress no permite asignarle una imagen a una categoría. Entonces, con la ayuda de este *plugin* podremos desde la aplicación móvil cargar la imagen almacenada en el *bucket* de Amazon S3 AWS, ya que al asignar una imagen a una categoría ésta se guarda automáticamente en el *bucket* con el plugin WP Offload Media Lite [31].

1.2.2.3 Pages with category and tag

En el apartado 2.1.4 explicamos que eran los *pages* en WordPress y las columnas que tenía, sin embargo, las columnas *categories* y *tag* realmente no son propios de la tabla *pages* sino que se consigue utilizando este *plugin* [32]. Si no utilizáramos este *plugin* no sería posible asignarle los *tags* a un canal en la aplicación móvil, por lo que es esencial si queremos conseguir la funcionalidad de filtrado de notificaciones.

1.2.2.4 Shortcodes Ultimate

Este *plugin* nos permite mostrar todas las publicaciones de un canal en una página de WordPress (aunque se puede acceder por categoría y sería lo mismo, pero es más tedioso y menos claro) [33]. De esta forma cualquier usuario que acceda a WordPress puede seleccionar una página (canal) de forma rápida y visualizar las publicaciones de ese canal. En la **Figura 38** podemos observar las diferentes páginas que tiene el proyecto: IES El Calero, Ayuntamiento Telde (Deportes), My Account y Registration.

Figura 38. WordPress – Portada con las diferentes páginas

1.2.2.5 WP Offload Media Lite

Se trata de un *plugin* que se encarga de guardar automáticamente las imágenes que utilicemos en WordPress en el *bucket* de Amazon S3 AWS. De esta forma podemos recuperar desde la aplicación móvil la imagen que se ha empleado para un canal o para

una publicación. Si no empleáramos este *plugin* habría que añadir las imágenes primero en WordPress y después en el *bucket* de Amazon S3 AWS de forma manual.

Hay que mencionar que para poder utilizar este *plugin* es necesario configurar previamente Amazon S3 AWS (esto se detalla más en el Capítulo X apartado 1.3. dado que necesita de la clave secreta y el nombre del *bucket*.

1.2.3. EndPoints

En este apartado veremos los puntos finales empleados para recuperar el contenido de la REST API de WordPress. Todos los partirán de la siguiente URL:

https://notifyme47.herokuapp.com/wp-json

1.2.3.1 Categories

Para recuperar las categorías principales de WordPress se hará uso de la siguiente ruta:

https://notifyme47.herokuapp.com/wp-json/wp/v2/categories

Los canales de una categoría se recuperan utilizando la misma ruta, pero hay que filtrarlas adecuadamente.

1.2.3.2 Posts

Para recuperar las publicaciones de un canal de WordPress se hará uso de la siguiente ruta:

https://notifyme47.herokuapp.com/wp-json/wp/v2/posts?categories={id del canal}

1.2.3.3 Tags

Para recuperar las etiquetas que hemos creado en WordPress se hará uso de la siguiente ruta:

https://notifyme47.herokuapp.com/wp-json/wp/v2/tags

1.2.3.4 Pages

Para recuperar las páginas que hemos creado en WordPress se hará uso de la siguiente ruta:

https://notifyme47.herokuapp.com/wp-json/wp/v2/pages?categories={id del canal}

1.2.3.5 Paginación

Como se había mencionado anteriormente en el capítulo III, la REST API devuelve por defecto 10 ítems por petición, sí queremos recuperar más ítems habrá que realizar lo que se conoce como *paginación*. Esto quiere decir que las rutas anteriores se modificarán ligeramente y pasarán a tener los siguientes aspectos:

https://notifyme47.herokuapp.com/wp-json/wp/v2/{tipo}?page={nº de página}

https://notifyme47.herokuapp.com/wp-json/wp/v2/{tipo}?categories={id del canal}&page={nº de página}

1.2. Firebase Realtime Database

En el caso de Firebase emplearemos la base de datos Realtime Database en la cual los datos se almacenan en formato JSON, se trata de una base de datos NoSQL.

Si nos fijamos en la **Figura 25** del capítulo IV podemos extraer las siguientes tablas:

1.2.1. Categories

Esta tabla se encargará de almacenar las diferentes categorías principales, que son las mismas que las de WordPress.

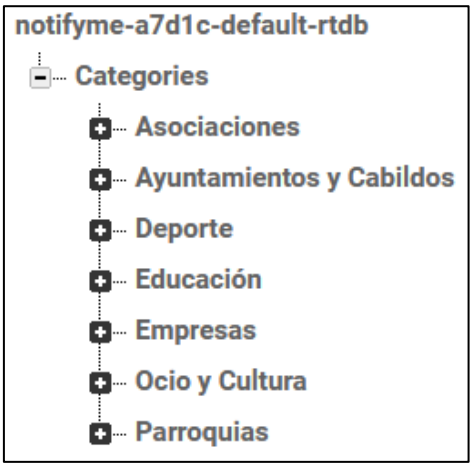


Figura 39. Realtime Database – Categories

1.2.2. Category

Category se refiere a las categorías que aparecen en la **Figura 39** y se encarga de almacenar los diferentes canales que pertenecen a esa categoría. En la **Figura 40** podemos observar que la categoría *Asociaciones* tiene tres canales cuyo identificador es generado por Firebase con el método *Push* (creación de un canal desde el móvil).

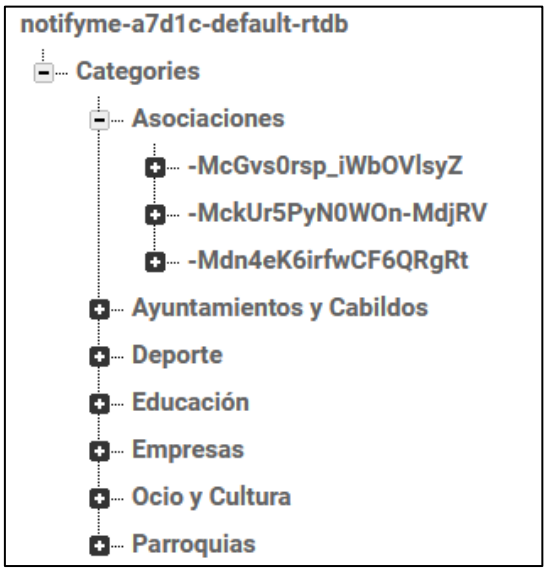


Figura 40. RealTime Database - Category

1.2.3. Channel

Cada canal en Firebase representa en la aplicación móvil un canal privado que solo es accesible para aquellos usuarios registrados. Los campos que aparecen en la **Figura 41** son los siguientes:

- **author:** representa el autor, es decir el usuario que ha creado el canal.
- **category:** indica a que categoría pertenece el canal. Cuando el usuario crea un canal privado desde la aplicación móvil tiene que elegir en que categoría quiere que aparezca su canal y este campo representa eso.
- **checked:** este campo representa el estado del CheckBox, es decir si este valor es igual a true significa que el CheckBox está marcado y si está a false entonces estaría desmarcado. Se emplea para la recepción de notificaciones push.

- **content:** se refiere a la descripción para indicar a qué se dedica el canal, de forma que los usuarios que se suscriban a dicho canal tengan una cierta idea.
- **date:** indica la fecha en la que se creó el canal.
- **hasPassword:** indica si el canal tiene contraseña (true) o no (false).
- **imgName:** representa el identificador de la imagen que se ha guardado en el *storage* de Firebase.
- **name:** representa el nombre del canal o el título del canal.

Figura 41. RealTime Database - Channel

1.2.4. Chat

En todos los canales existirá una tabla *Chat* en la que se almacena los mensajes que se envía al grupo. Los campos que aparecen en la **Figura 42** son los siguientes:

- **dateHour:** indica la fecha en la que se envió el mensaje.
- **message:** representa el mensaje que se ha enviado al grupo.
- **name:** representa el email del usuario que ha enviado el mensaje al grupo.

Figura 42. RealTime Database – Chat

1.2.5. Users

Al igual que la tabla *Chat*, todos los canales poseerá una tabla *Users* en la que se guardan todos los usuarios que están suscritos al canal. Esto se puede observar en la **Figura 43**.

- **name:** representa el email del usuario

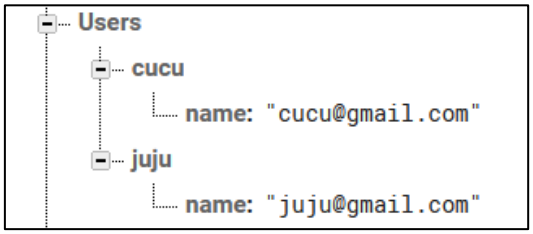


Figura 43. RealTime Database – Users

1.3. Room

Para guardar los datos localmente se ha utilizado la librería persistente Room. Esta librería proporciona una capa de abstracción sobre SQLite [34] permitiéndonos acceder a la base de datos de forma eficiente, rápida y al mismo tiempo aprovechar toda la potencia de SQLite.

Room está formado por tres componentes principales:

- **Base de datos:** Contiene el titular de la base de datos y sirve como punto de acceso principal para la conexión subyacente a los datos persistentes y relacionales de la aplicación.
- **Entidad:** Representa una tabla dentro de la base de datos.
- **DAO:** Contiene los métodos utilizados para acceder a la base de datos.

A continuación, en la **Figura 44** se muestra un esquema de la relación entre los diferentes componentes de Room con la aplicación.

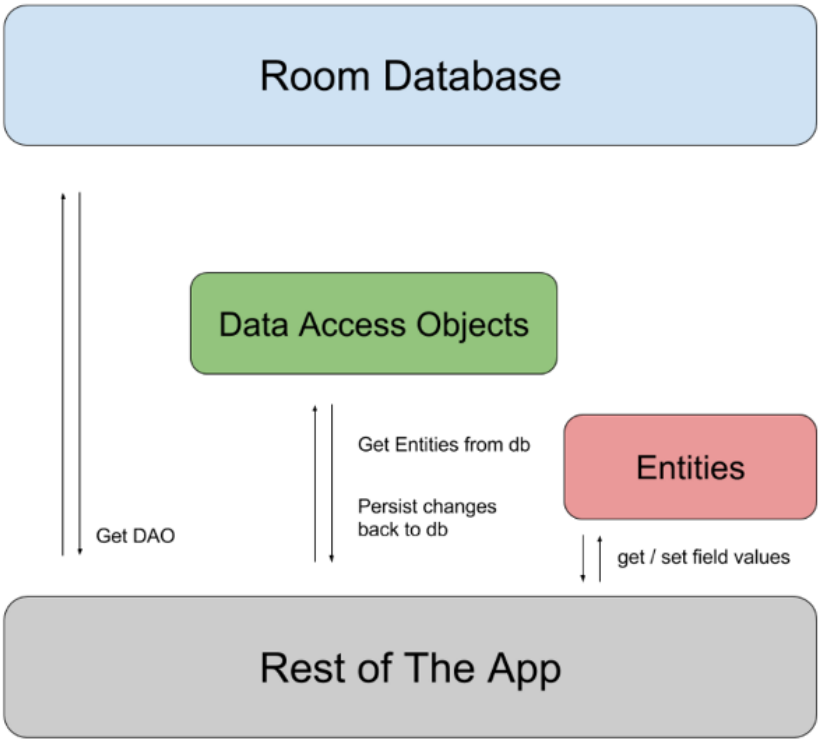


Figura 44. Diagrama de la arquitectura de Room

Como se puede observar en la **Figura 44** , la aplicación obtiene los objetos de acceso a los datos (DAO) [35] interactuando con la base de datos de Room. Una vez obtenida los DAOs, el siguiente paso es utilizar cada uno de estos para obtener entidades [36] de la base de datos, realizar los cambios y guardarlos nuevamente en la base de datos. Por último, la aplicación usa una entidad para obtener y configurar valores en las columnas de las tablas dentro de la base de datos.

1.3.1. Entidades

Como se había mencionado anteriormente, una entidad es aquella clase que representa una tabla dentro de la base de datos. A través de las entidades podemos modificar, añadir o borrar datos en la base de datos SQLite.

A continuación, se mostrarán las diferentes tablas que componen la base de datos Room utilizando el plugin UML Generator [37].

1.3.1.1. Entidad UserItem

La entidad *User* es la tabla principal de la base de datos, la que se muestra en la **Figura 45**, y está formada por:

- **email:** variable String que guarda el email del usuario.
- **id:** clave primaria.

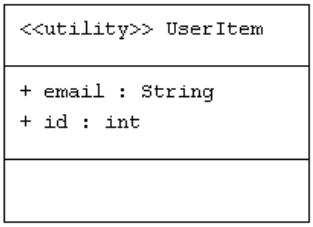


Figura 45. Entidad User

1.3.1.2. Entidad SubcategoryItem

Esta entidad se encarga de mapear un objeto canal en la base de datos y tendrá como clave foránea el identificador de la tabla *User,* tal y como se puede observar en la **Figura 46**, de forma que podamos distinguir los canales entre los diferentes usuarios. Los campos que aparecen en la entidad son:

- **userId:** clave foránea de la clase UserItem.
- **channelKey:** identificador del canal.
- **category:** nombre de la categoría, por ejemplo, educación.

- **email:** email del usuario.
- **isPrivate:** diferencia entre canales públicos o privados.
- **enabled:** habilitar o deshabilitar el botón de las opciones extras.
- **notificationChecked:** habilita o deshabilita el checkbox.
- **parentId:** variable String que define el identificador padre de una categoría. En WordPress las categorías que tengan parentId cero son las categorías principales, por tanto, aquellas categorías que tengan un valor diferente a cero en el campo parentId será una subcategoría.
- **image:** nombre de la imagen del canal.
- **name:** nombre del canal.
- **id:** clave primaria.

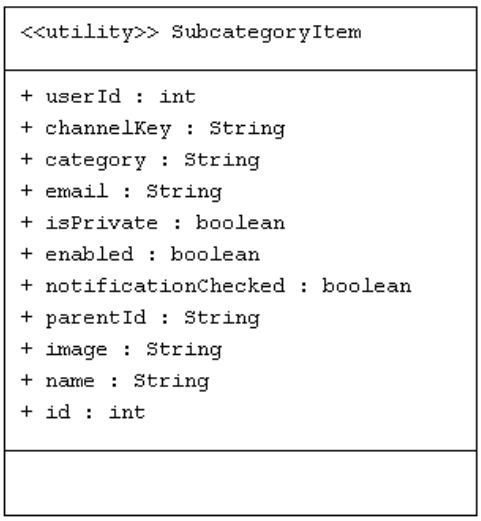


Figura 46. Entidad SubcategoryItem

1.3.1.3. Entidad SubcategoryDetailItem

Esta entidad, la de la **Figura 47**, almacena el detalle de un canal como el título, el nombre de la imagen, la fecha y el contenido. Tendrá como clave foránea el *id* de la clase *SubcategoryItem*. Los campos de esta entidad son:

- **subCategoryId:** clave foránea de la clase SubcategoryItem.
- **imageName:** nombre de la imagen.
- **date:** fecha en la que se ha publicado la noticia.

- **content:** contenido de la noticia.
- **title:** título de la noticia.
- **id:** clave primaria.

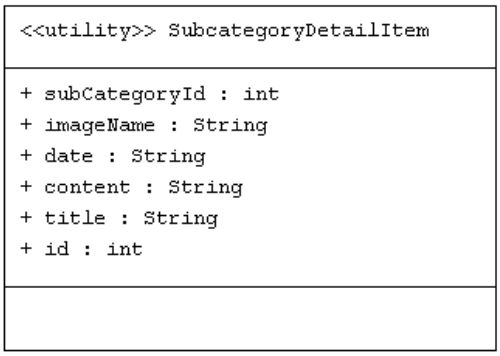


Figura 47. Entidad SubcategoryDetailItem

1.3.1.4. Entidad SubNotificationsItem

La entidad *SubNotificationsItem* se encargará de guardar los estados del checkbox referidos a los *tags* o filtros. Los campos que componen esta entidad se pueden observar en la **Figura 48**:

- **subCategoryId:** clave foránea de la clase SubcategoryItem.
- **tagName:** nombre del tag o filtro.
- **subCategoryName:** nombre de la subcategoría o del canal.
- **checkBoxEnable:** estado del checkbox.
- **id:** clave primaria.

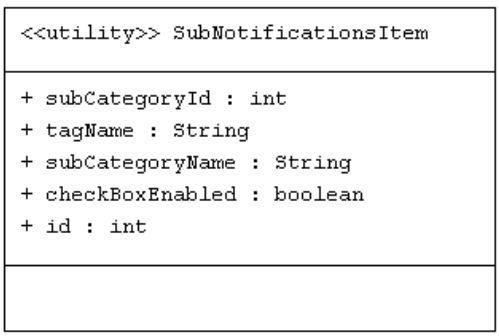


Figura 48. Entidad SubNotificationsItem

1.3.1.5. Entidad SubNotificationsHistoryItem

La clase *SubNotificationsHistoryItem*, la de la **Figura 49**, tiene como objetivo almacenar las notificaciones que el usuario reciba de los diferentes canales en la base de datos Room. Los campos que posee esta entidad son los siguientes:

- **subCategoryId:** clave foránea de la clase SubcategoryItem.
- **email:** email del usuario.
- **isRead:** si esta leído o no una notificación del historial de notificaciones.
- **channel:** nombre del canal.
- **notificationsSubName:** cuerpo de la notificación.
- **notificationsName:** título de la notificación.
- **id:** clave primaria.

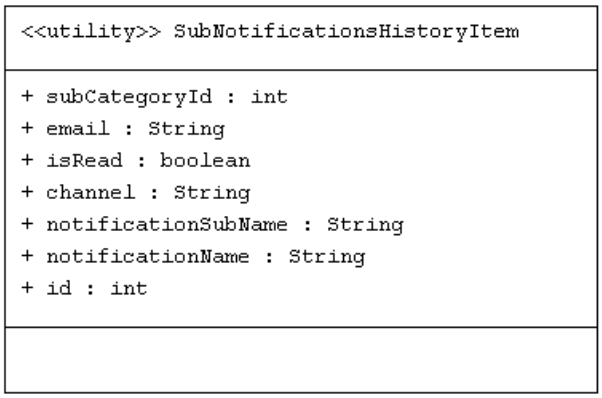


Figura 49. Entidad SubNotificationsHistoryItem

1.3.1.6. Esquema completo de las entidades.

En la **Figura 50** se muestra un esquema completo de las entidades mencionadas anteriormente y sus relaciones entre ellas. Si observamos la entidad *UserItem* está relacionada con la entidad *SubcategoryItem* y ésta a su vez está relacionada con las entidades *SubcategoryDetailItem*, *SubNotificationsItem* y *SubNotificationsHistoryItem*.

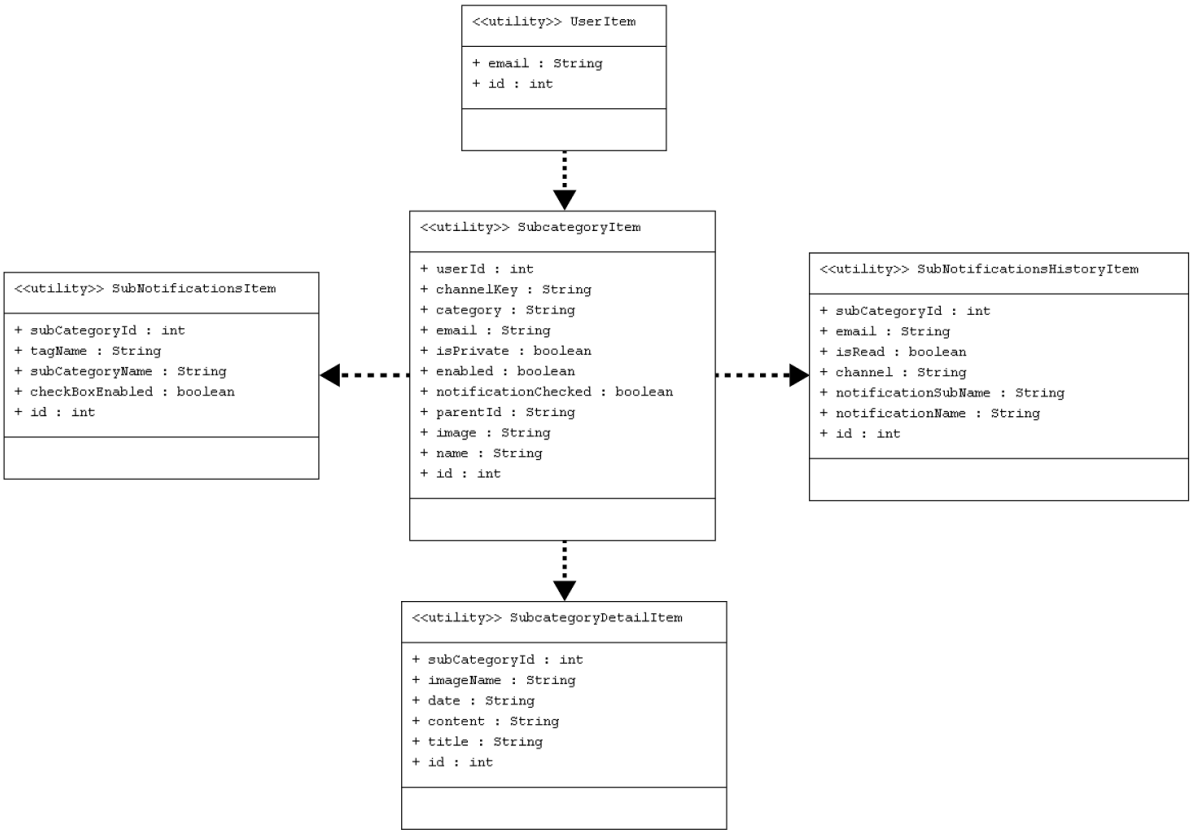


Figura 50. Esquema de las entidades de la base de datos Room

1.3.2. DAOs

La biblioteca de persistencias Room trabaja con objetos de acceso a datos o *DAO* para acceder a los datos de la aplicación. Cada *DAO* incluye métodos que ofrecen acceso abstracto a la base de datos. Las clases *DAO* se anotan con la nomenclatura *@Dao* y las consultas se pueden apreciar en la **Tabla 1**:

@Insert	Inserta un conjunto de entidades
@Update	Modifica un conjunto de entidades
@Delete	Quita un conjunto de entidades
@Query	Realiza operaciones de lectura/escritura

Tabla 1. Anotaciones de una clase DAO

1.3.2.1. UserDao

DAO que define las interacciones con los usuarios de la base de datos.

Métodos

---

insertUser

public void insertUser (UserItem userItem)

Inserta un usuario en la base de datos.

loadUsers

public List<UserItem> loadUsers ()

Carga todos los usuarios de la base de datos.

1.3.2.2. SubcategoryDao

DAO que define las interacciones con las categorías y subcategorías de la base de datos.

Métodos

---

deleteSubCategory

public void deleteSubCategory (SubcategoryItem subcategory)

Elimina un ítem *SubcategoryItem*.

insertSubCategory

public void insertSubCategory (SubcategoryItem subcategory)

Inserta un ítem *SubcategoryItem*.

loadSubCategories

public List<SubcategoryItem> loadSubCategories ()

Devuelve una lista *SubCategoryItem* con todos los ítems.

loadSubCategoriesByUser

public List<SubcategoryItem> loadSubCategoriesByUser (int userId)

Devuelve una lista *SubCategoryItem* con todos los ítems cuyo *userId* coincida con el *userId* pasado por parámetro.

updateCheckBoxEnable

public void updateCheckBoxEnable (int id, boolean enabled)

Actualiza el valor *enabled* en función del *id* del canal, es decir se encarga de guardar el estado del *checkbox* en la base de datos (*enabled* = *true* → checkbox activado)

updateNotifications

public void updateNotifications (int id, boolean notificationChecked)

Actualiza el valor *notificationChecked* en función del *id* del canal, es decir se encargará de guardar el estado del *checkbox* en la base de datos (*notificationChecked* = *true* → checkbox marcado).

1.3.2.3. SubcategoryDetailDao

DAO que define las interacciones con el detalle de las subcategorías de la base de datos.

Métodos

---

deleteAllItems

public void deleteAllItems (int subcategoryId)

Borra todos los ítems de la tabla cuyo *subcategoryId* coincidan con el valor del parámetro *subcategoryId*.

insertSubcategoryDetail

public void insertSubcategoryDetail (SubcategoryDetailItem subcategoryDetail)

Inserta un ítem *subcategoryDetail*.

loadSubCategoriesDetail

public List<SubcategoryDetailItem> loadSubCategoriesDetail (int subcategoryId)

Devuelve una lista con todos los ítems cuyo *subcategoryId* coincidan con el valor del parámetro *subCategoryId*.

1.3.2.4. SubNotificationsDao

DAO que define las interacciones con los tags o filtros de los canales de la base de datos.

Métodos

---

checkAllNotifications

public void checkAllNotifications (int subcategory_id, boolean checkBoxEnabled)

Actualiza todos los checkboxes al valor *checkBoxEnabled*, o sea marca o desmarca todos los *tags* del canal cuyo *subcategoryId* corresponde con el *subcategory_id* pasado por parámetro.

insertNotifications

public void insertNotifications (SubNotificationsItem subNotificationsItem)

Inserta un ítem *SubNotificationsItem*.

loadNotifications

public List<SubNotificationsItem> loadNotifications ()

Devuelve una lista *SubNotificationsItem* con todos los ítems de ese tipo de la base de datos.

loadNotifications

public List<SubNotificationsItem> loadNotifications (int subcategoryId)

Devuelve una lista *SubNotificationsItem* de todos los ítems de ese tipo de la base de datos cuyo *subcategoryId* coincida con el valor *subcategoryId* pasado por parámetro.

updateNotifications

public void updateNotifications (int id, boolean checkBoxEnabled)

Actualiza el valor *checkBoxEnabled* de un canal, es decir se encargará de guardar el estado del *checkbox* (*tags*) en la base de datos (*checkBoxEnabled* = *true* → checkbox marcado).

1.3.2.5. SubNotificationsHistoryDao

DAO que define las interacciones con las notificaciones de los canales de la base de datos.

Métodos

---

deleteNotificationsHistory

public void deleteNotificationsHistory (SubNotificationsHistoryItem subNotificationsHistoryItem)

Elimina un ítem *subNotificationsHistoryItem*.

insertNotificationsHistory

public void insertNotificationsHistory (SubNotificationsHistoryItem subNotificationsHistoryItem)

Inserta un ítem *subNotificationsHistoryItem*.

loadNotificationsByUser

public void loadNotificationsByUser(int subCategoryId)

Devuelve una lista de objetos *SubNotificationsHistoryItem* solo si el valor *subcategoryId* coincide con el valor *subCategoryId* pasado por parámetro.

2. Aplicación móvil

En esta última sección del capítulo V se comentará en primer lugar las incorporaciones y modificaciones que se han añadido a la aplicación, después se detallará la lógica de aplicación, así como las clases generales y las diferentes actividades (*Activities*). Y finalmente, se explicará cómo se ha realizado el testing.

2.1. Incorporaciones y modificaciones

2.1.1. Pantalla Login y Register

En la pantalla *Login*, la de la **Figura 51**, se muestra directamente el formulario de forma que sea más sencillo para el usuario registrarse. Por otra parte, en la pantalla *Register* o *Sign up* no es necesario introducir el nombre ni el apellido, sin embargo, para ayudar al usuario a memorizar la contraseña y para evitar posibles conflictos, tendrá que introducirla dos veces. En la **Figura 52** se ilustra el aspecto de esta pantalla.

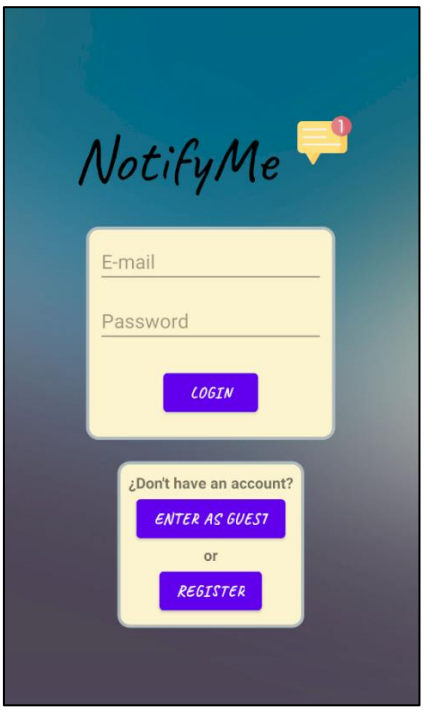


Figura 51. Pantalla Login

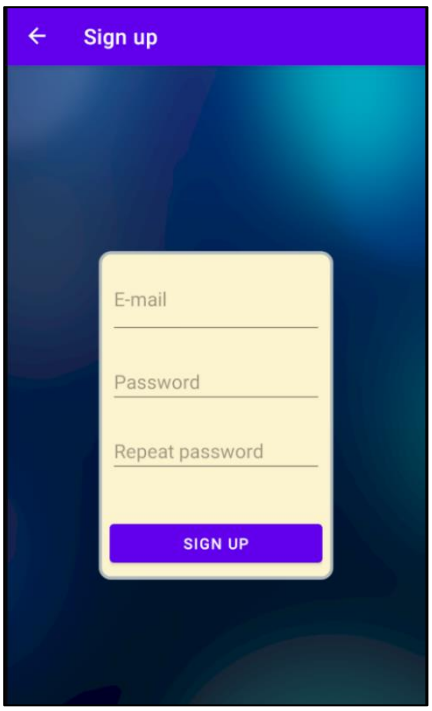


Figura 52. Pantalla Register

2.1.2. Pantalla PublicNotifications y PublicNotificationsDetail.

En la pantalla *Index* y en el resto de las pantallas que tengan el botón con forma de campana localizado en el Toolbar para ver las notificaciones, ese botón se ha trasladado a la pantalla *PublicNotifications*. Si no estamos suscritos a ningún canal aparecerá un texto junto con una imagen indicando de que la lista está vacía. Esto se puede observar en la **Figura 53**.

Si existen elementos en la lista como aparece en la **Figura 54**, podremos al igual que los *mockups* habilitar las notificaciones con la singularidad de que ahora para añadir filtros, el usuario tendrá que pulsar en el botón *extra options* para ver los *tags*. En el detalle de las notificaciones se han añadido dos botones para que el usuario pueda activar o desactivar los filtros de forma rápida.

Figura 53. Pantalla PublicNotifications sin canales

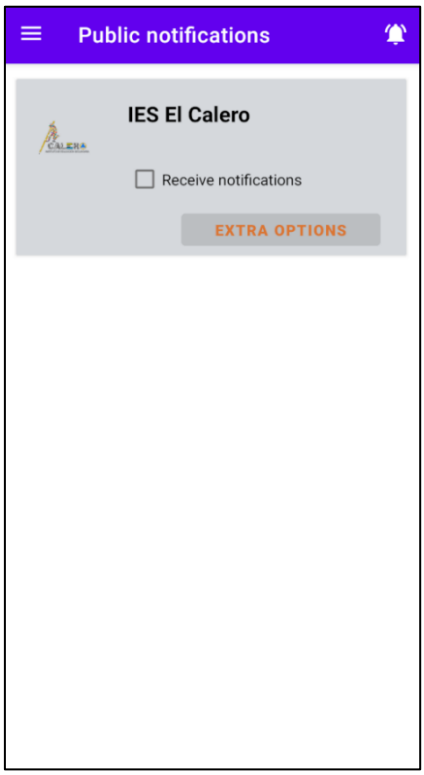


Figura 54. Pantalla PublicNotifications con canales

2.1.3. Pantalla Posts

En la pantalla *Posts* se ha añadido un botón *share* en el Toolbar que nos permite compartir la publicación a través de las redes sociales como WhatsApp, esto se puede ver en la **Figura 55** y **Figura 56**. Si otro usuario accede al enlace compartido se le redirigirá a la página de WordPress de la publicación.

Figura 55. Pantalla Posts

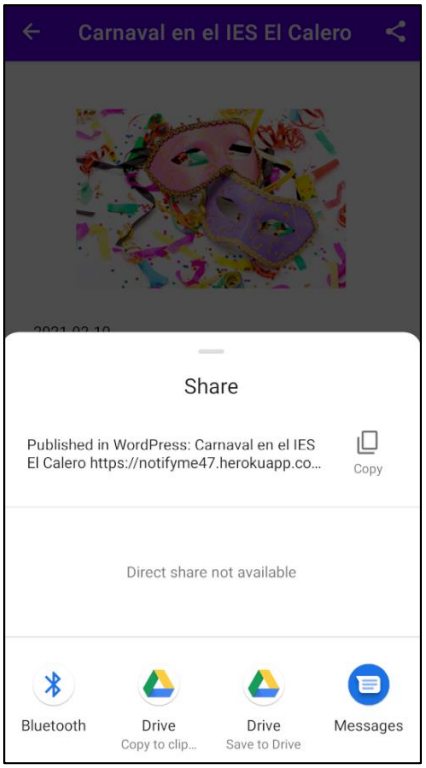


Figura 56. Pantalla Posts compartiendo publicación

2.1.4. Private channel y Private notifications

Se han realizado varios cambios en las pantallas privadas y la primera de ellas es *Private channels*, la de la **Figura 58**. En los *mockups* se mostraban los canales privados propios y de otros usuarios junto con el botón *crear canal*. Ahora en la pantalla *Private channels* se muestran simplemente los canales privados de otros usuarios.

Por otra parte, al igual que los canales públicos nos avisará si no estamos suscritos a ningún canal. Esto se puede ver en la **Figura 57**.

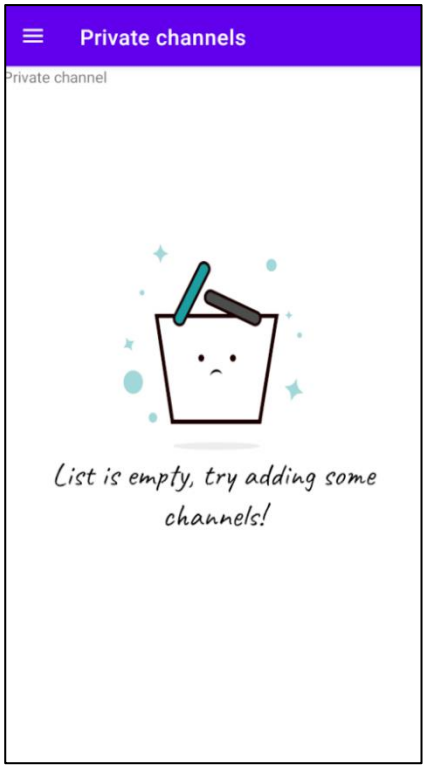


Figura 57. Pantalla Private channels vacía

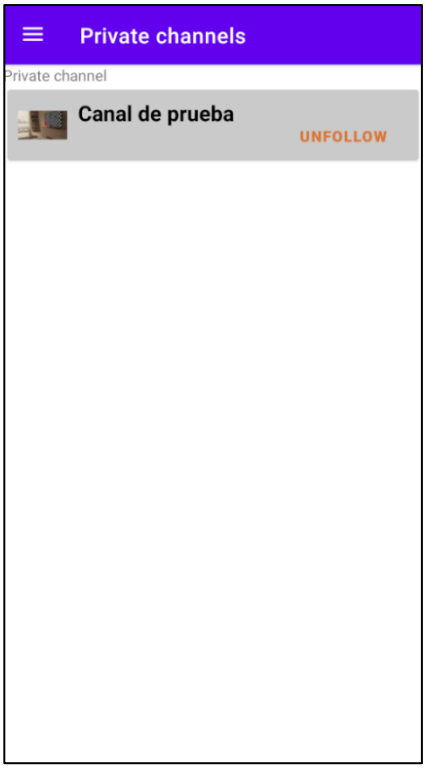


Figura 58. Pantalla Private channels

En el caso de las notificaciones de los canales privados, el de la **Figura 59**, solamente aparecerán aquellos canales privados creados por otros usuarios.

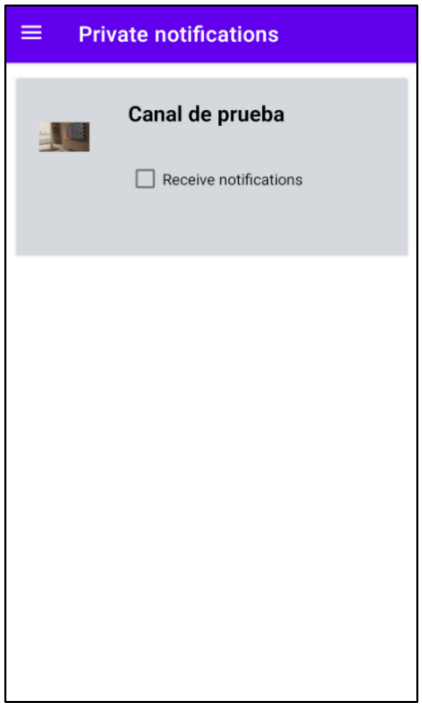


Figura 59. Pantalla Private notifications

2.1.5. Pantalla My channels

En esta nueva pantalla aparezcan los canales que el usuario haya creado. Cada ítem está formado por el nombre del canal, una imagen seleccionada de la gallería, un checkbox para recibir notificaciones y el botón *delete*. Esto se puede ver en la **Figura 60**.

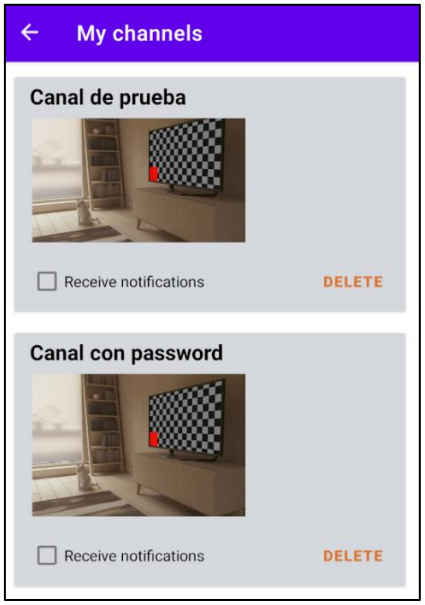


Figura 60. Pantalla MyChannels

2.1.6. Pantalla Create channels

Al principio esta opción aparecía en la pantalla *PrivateChannels*, sin embargo, se ha decidido incorporarlo dentro de un *FloatingActionButton* [38] y en la barra de navegación, tal y como se puede observar en la **Figura 61** y **Figura 62**. El usuario podrá acceder a la creación del canal por amabas formas y tendrá la opción de elegir si quiere crear el canal con contraseña o sin contraseña. La primera opción (canal con contraseña) es útil si no queremos que otros usuarios se suscriban a nuestro canal, ya sea porque es un canal que solo queremos compartirlo con amigos o familiares o porque el canal fue creado por un colegio y es únicamente accesible para los padres. Finalmente, en la pantalla *CreateChannels* se ha incluido para que el usuario pueda elegir una imagen para el canal desde la gallería, como el de la **Figura 65**, si no se elige se empleará la imagen por defecto. En la **Figura 63** y **Figura 64** se puede observar el aspecto de la pantalla *CreateChannels*.

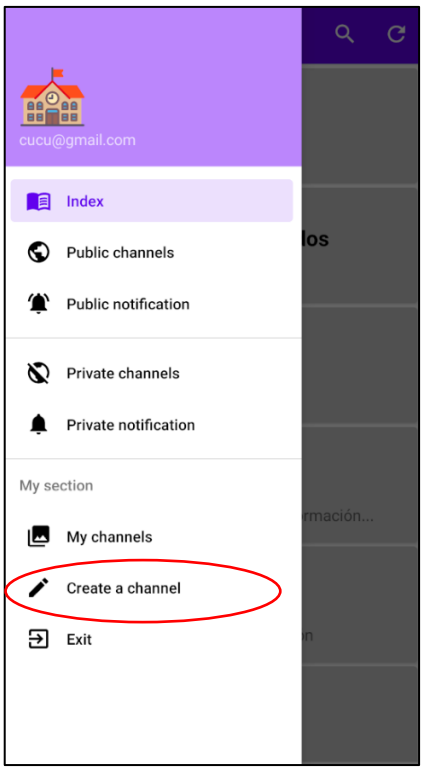


Figura 61. Barra de navegación privada

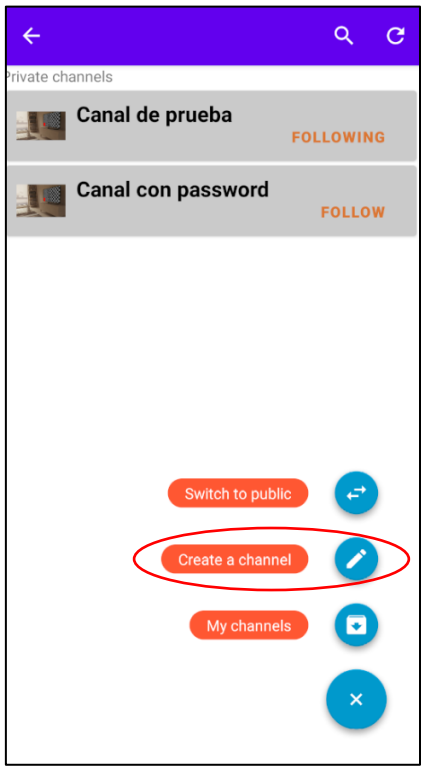


Figura 62. Pantalla Subcategory

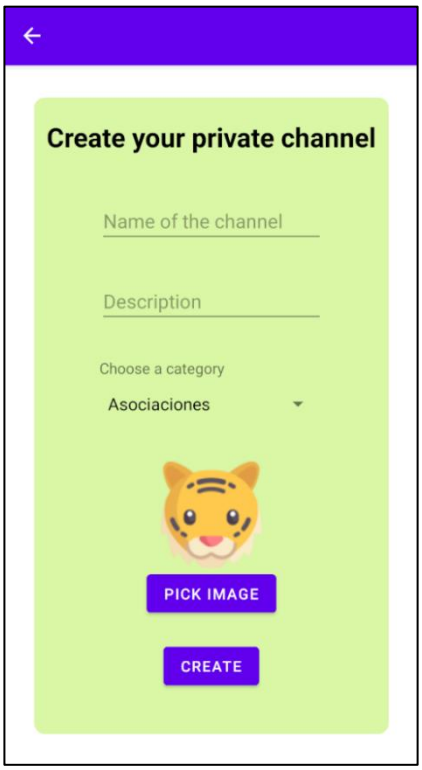


Figura 63. Pantalla Create channel sin contraseña

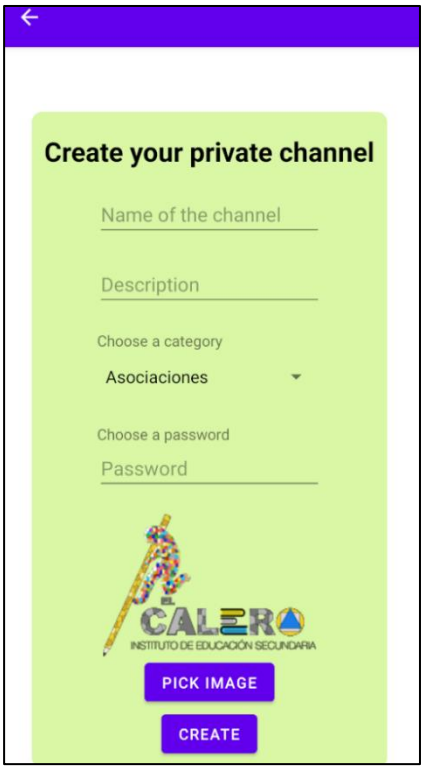


Figura 64. Pantalla Create channel con contraseña

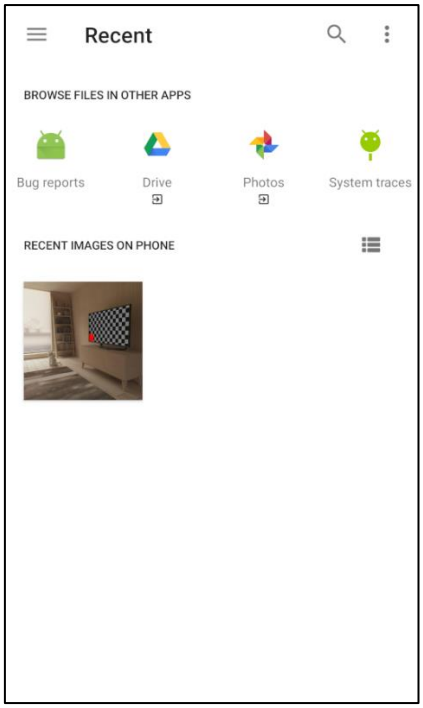


Figura 65. Seleccionando una imagen de la gallería

2.1.7. Pantalla ChannelInformation

Figura 66. Pantalla Channels information

Se ha añadido también la pantalla *ChannelInformation* que se encargará de mostrar información extra de un canal privado, así como la imagen, la fecha en la que se ha creado, el nombre del canal y una breve descripción acerca del canal. De forma que, si un usuario se suscribe a un canal privado éste pueda saber a qué se dedican los usuarios del grupo. Esto se puede ver en la **Figura 66**.

2.2. Lógica de la aplicación

En esta sección comentaremos primero las clases generales que intervienen y después detallaremos la lógica de cada una de las pantallas que conforman la aplicación. Se incluirán las clases generales (repositorio, objetos ...) y las actividades (*Activities*). Hay que mencionar que los campos y los métodos de las clases generales y *Activitivies* se han movido al capítulo X. Anexos apartado 3.

2.2.1. Clases generales

En este apartado se explicará todas las clases generales desde los métodos del repositorio, el mediador, hasta los objetos utilizados para almacenar información de los canales, subcanales, *tags* etcétera.

2.2.1.1 ChatClass

Este objeto se emplea principalmente en la pantalla PrivatePosts para mapear los mensajes que enviamos al chat y mensajes que recibidos de otros miembros.

2.2.1.2 FetchCategorias

Este objeto se emplea en varias pantallas para mapear un objeto canal o subcanal.

2.2.1.3 FetchFirebaseData

Este objeto define un canal de Firebase.

2.2.1.4 FetchPages

Este objeto se encarga de almacenar los datos de una página de WordPress. En este proyecto solo necesitaremos el campo tag y el nombre.

2.2.1.5 FetchTags

Este objeto define un tag de WordPress.

2.2.1.7 MyFirebaseMessagingService

Esta clase extiende de FirebaseMessagingService [39] y se emplea para generar y tratar las notificaciones push. Es una clase que siempre está a la escucha (servicio) incluso si la aplicación esta destruida.

Como se había mencionado en el capítulo de tecnologías software, emplearemos Postman para enviar notificaciones push. La estructura de una notificación es la siguiente:

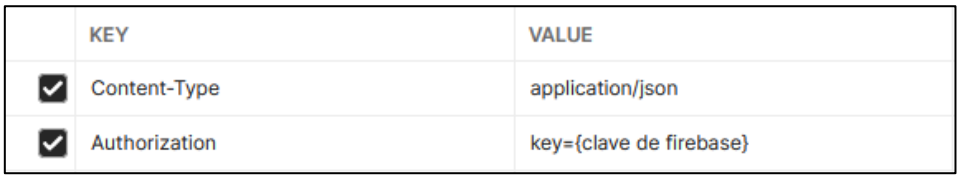
Figura 67. JSON de una notificación push

Los campos que aparecen en la **Figura 67** son los siguientes:

- **to:** este campo indica a que grupo va dirigido la notificación. Los usuarios que empleen la aplicación se auto suscribirán al grupo *general*.
- **data:** contiene los datos de una notificación.
- **title:** representa el título de la notificación.
- **body:** representa la descripción de la notificación.
- **channel:** representa el canal, es decir a que canal va dirigido la notificación.
- **tag:** representa la etiqueta o las etiquetas que tiene la notificación, de forma que los usuarios las pueda filtrar en la aplicación móvil.

Una vez descrita la estructura de la notificación es necesario proporcionarle en la cabecera los campos que aparecen en la **Figura 68**:

- **Content-Type:** aquí le indicaremos que se trata de un contenido JSON
- **Authorization:** aquí le proporcionamos la clave de Firebase.



	KEY	VALUE
☑	Content-Type	application/json
☑	Authorization	key={clave de firebase}

Figura 68. Cabecera de la notificación push

2.2.1.9 NotifyMeRepository

Esta es la clase singleton repositorio, donde están definidos todos los métodos para interactuar con las bases de datos Room y Firebase. Además, se encargará de almacenar y actualizar las preferencias como el email y de realizar peticiones HTTP.

En el proyecto utilizaremos esta clase en la mayoría de los modelos para recuperar, actualizar e insertar datos en la base de datos Room y Firebase.

2.2.2. Activities

Una *Activity* en Android se corresponde con una pantalla de la aplicación, es decir la interfaz de usuario. Se compone de una clase, un *layout* y una definición de uso (*AndroidManifest*). Tal y como se había mencionado en el punto 2.2. los campos y los métodos de cada pantalla se explican en el capítulo X. Anexos apartado 3.2.

2.2.2.1 SplashScreen

La pantalla *SplashScreen* es la pantalla principal de la aplicación y su función es mostrar una pantalla de carga con una portada. Para realizar esta funcionalidad se ha utilizado la librería *EasySplashScreen* [40] que nos permite generar una pantalla splash de forma sencilla y personalizable.

En esta pantalla, aunque se ha empleado la plantilla de *clean code*, solamente se ha modificado la clase *SplashScreenActivity* para generar la pantalla splash.

2.2.2.2 Login

En la pantalla *Login* se realizan dos funciones: acceder al contenido como usuario registrado o invitado y permitir a estos registrarse haciendo clic en el botón *register*. Si el usuario se da de alta introduciendo las credenciales se comprobará la autenticación contra Firebase.

2.2.2.3 Register

En esta pantalla permitirá al usuario registrarse en el proyecto de Firebase introduciendo un email y una contraseña. De esta forma los usuarios se podrán darse de alta como usuario registrado y suscribirse a los canales privados.

2.2.2.4 Index

La pantalla *Index* se encargará de mostrar al usuario las diferentes categorías recuperadas de WordPress a través de la REST API utilizando la librería *Volley* [41] de Google. Por otra parte, también existirá una barra de navegación lateral que permitirá al usuario navegar entre pantallas de forma rápida. Finalmente, se ha incluido un buscador de canales en el Toolbar de forma que el usuario pueda buscar un canal específico.

En la **Figura 69** podemos observar el proceso llevado a cabo para recuperar la información de los canales en WordPress. En primer lugar, la aplicación realiza una petición HTTP a la REST API de WordPress empleando la librería de *Volley*. Si la petición fue satisfactoria la página devolverá un fichero JSON el cual es necesario realizar un filtrado para obtener los datos de interés. Este fichero *Volley* lo trata como un objecto JSONArray [42] de forma que para obtener la información que necesitamos hay que crear un objeto JSONObject [43] y recorrer el array de JSON y extraer dicha información relevante.

Hay que tener en cuenta que la petición a la REST API con *Volley* es asíncrona de forma que para saber que ha terminado de ejecutar la función se empleará de un *callback* [44].

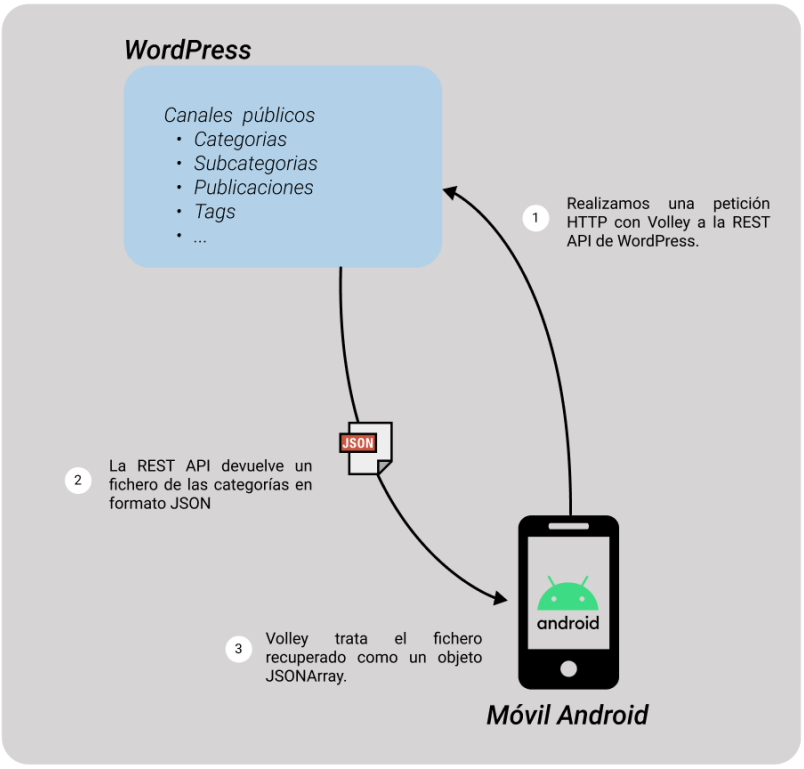


Figura 69. Petición a la REST API de WP con Volley

2.2.2.5 SubCategory

La pantalla *SubCategory* se encargará de mostrar los canales asociados a la categoría que ha elegido el usuario en la pantalla *Index*. Los canales públicos se recuperan de la REST API de WordPress empleando la librería *Volley*, sin embargo, los canales privados se obtendrán de Firebase adjuntando *listeners* asíncronos a una referencia de base de datos. En la **Figura 70** podemos ver la interacción de la app con la base de datos de Firebase.

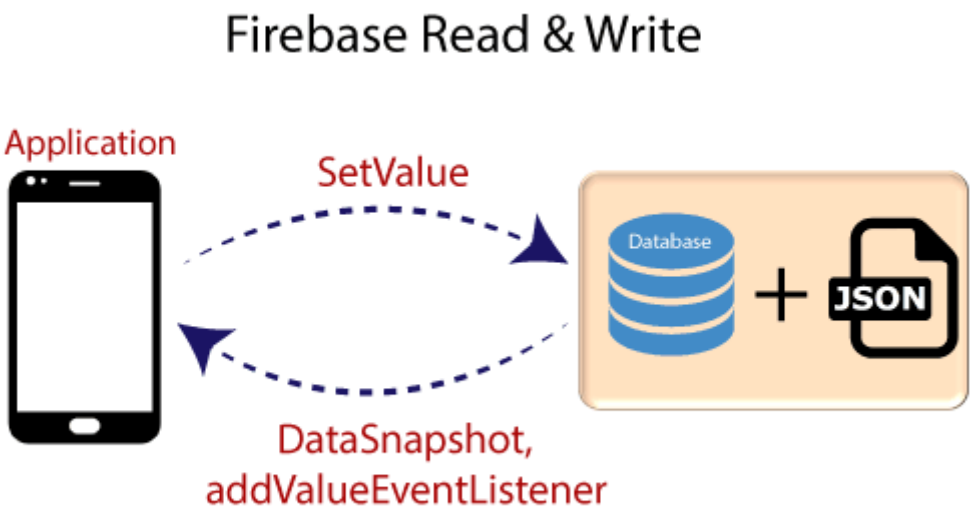


Figura 70. Ilustración que representa la interacción entre la App Android y Firebase [45]

2.2.2.6 SubCategoryDetail

Esta pantalla tendrá como propósito mostrar el detalle del canal seleccionado en la vista *SubCategory*, o sea nos presentará un listado con todas las publicaciones relacionadas con dicho canal. Estas publicaciones también se recuperarán de la REST API de WordPress haciendo uso de la librería *Volley*, sin embargo, si el canal que hemos elegido era privado, entonces recuperará el chat de Firebase.

2.2.2.7 Posts

La pantalla *Posts* es responsable de mostrar el detalle de una publicación, así como la imagen, el título, el contenido y la fecha en la que se publicó. También dispondrá de un botón *share* situado en el Toolbar que nos permite compartir la noticia con otros usuarios.

2.2.2.8 PublicChannels

En esta pantalla se mostrarán todos los canales públicos en los que está suscrito el usuario. La información de los canales y de las publicaciones se recuperan de la base de datos Room. Asimismo, si un canal ya no es de interés el usuario puede darse de baja pulsado en el botón *unfollow*.

2.2.2.9 PublicChannelsDetail

En la pantalla *PublicChannelsDetail* se presentarán todas las publicaciones del canal elegido en la pantalla *PublicChannel*. Estas publicaciones se recuperarán de la base de datos Room, y si se pulsa en una de las publicaciones se dirigirá a la vista *Posts* para mostrar el detalle de la publicación.

2.2.2.10 Search

La pantalla *Search* tiene objetivo mostrar aquellos canales cuyos nombres coincidan (o parte de ella) con la *query* introducida por el usuario en el buscador de canales (*SearchView*). Esto es conveniente cuando queremos buscar un canal específico de forma rápida y eficaz.

2.2.2.11 PublicNotifications

En la pantalla *PublicNotifications* permitirá al usuario configurar las notificaciones de los canales públicos, así como el filtrado por *tags* (eventos, competiciones...) de forma que solo se reciban las noticias que nos sean de interés.

2.2.2.12 PublicNotificationsDetail

En la pantalla *PublicNotificationsDetail* se mostrarán los *tags* (eventos, competición...) de un canal, que permitirá al usuario personalizar las notificaciones y recibir aquellas noticias que le sea de interés.

2.2.2.13 PublicNotificationsHistory

En la pantalla *PublicNotificationsHistory* se presentarán las notificaciones de las noticias de los diferentes canales a los que está suscrito el usuario. Estas notificaciones se pueden eliminar pulsando en el botón *dismiss*.

2.2.2.14 PrivateChannel

En esta pantalla se mostrarán todos los canales privados a los que está suscrito el usuario. La información de los canales se recuperará de la base de datos Room, mientras que el chat asociado a cada canal privado se extraerá de Firebase. Asimismo, si un canal ya no es de interés el usuario puede darse de baja pulsado en el botón *unfollow*.

2.2.2.15 PrivateNotifications

En la pantalla *PrivateNotifications* permitirá al usuario activar o desactivar las notificaciones de los canales privados en los que está suscrito.

2.2.2.16 PrivatePosts

En la pantalla *PrivatePosts* se presentará un chat grupal para que los participantes (usuarios suscritos al canal) se puedan comunicar entre sí. Podrán compartir todo tipo de noticias e información relevante relacionado con el tema del canal. Todos los mensajes que se envíen por el chat quedarán almacenados en la base de datos de Firebase.

2.2.2.17 MyChannels

En la pantalla *MyChannels* se presentarán los canales privados creados por el propio usuario en Firebase, de forma que éste no tenga que estar buscando sus canales en la pantalla *PrivateChannels* o *SubCategory* (privado). Asimismo, podrá acceder al chat, activar o desactivar las notificaciones del canal e incluso eliminar dichos canales.

2.2.2.18 CreateChannel

En esta pantalla el usuario podrá crear sus propios canales privados con contraseña o sin contraseña. Los canales creados se guardarán en la base de datos de Firebase y será visible y accesible por otros usuarios.

Un canal sin contraseña tiene interés cuando queremos que cualquier usuario se pueda unir, por ejemplo, un canal que se centra en los juegos de mesa como el ajedrez. Por otro lado, un canal con contraseña es útil cuando no queremos que cualquier usuario se pueda suscribir, esto podría ser un club deportivo privado.

2.2.2.19 ChannelInformation

La pantalla *ChannelsInformation* es similar a la pantalla Posts en la que se mostrará el detalle de un canal privado como la fecha de creación, el título, la descripción y la imagen del canal.

La descripción del canal tiene interés por ejemplo si un usuario se suscribe a un canal privado, pero no sabe a qué se dedican exactamente, éste puede ver la descripción para hacerse una idea.

2.2.2.20 Participants

Esta pantalla se encargará de mostrar un listado con todos los usuarios que están suscritos al canal de forma similar a la sección participantes en los grupos de WhatsApp. Se mostrará una imagen de perfil y el email del usuario. Esta pantalla se comentará más en el apartado de líneas futuras.

2.3. Tests de instrumentación

2.3.1. Espresso

Para realizar los tests de instrumentación se utilizará la herramienta Espresso [46] que nos permitirá escribir pruebas de la interfaz de usuario en Android de forma concisa, eficaz y confiable.

Las pruebas escritas en Espresso se ejecutan con una rapidez óptima y nos permite dejar atrás esperas, sincronizaciones, tiempos inactivos y sondeos, mientras se manipula y se afirma en la UI cuándo están en reposo.

2.3.2. Librerías Barista y UI Automator

La herramienta Espresso es bastante sólida, sin embargo, existen algunos tests que no los puede manejar completamente por lo que nos ayudaremos de algunas librerías externas como Barista y UI Automator.

Barista [47] hace que el desarrollo de pruebas de la UI sea más rápido, más fácil y predecible. Esta construido sobre Espresso y proporciona una API simple y detectable, eliminando la mayor parte de la sobrecarga y la verbosidad de las tareas comunes de Espresso.

Por otra parte, UI Automator [48] [49] proporciona un conjunto de APIs para compilar pruebas de la interfaz de usuario que realizan interacciones en las apps del usuario y las apps del sistema. Las API de UI Automator nos permite realizar operaciones como abrir el menú de configuración o el lanzador de apps en un dispositivo de prueba.

2.3.3. Testing

A continuación, mostraremos los tests y las clases generales.

2.3.3.1 Clases generales

En esta sección se explica de forma simplificada las clases auxiliares empleadas durante el testeo.

RecyclerViewMatcher

public class RecyclerViewMatcher

Esta clase auxiliar [50] nos permitirá comprobar el correcto funcionamiento de los *RecyclerViews*, así como los *TextViews*, *Buttons*, *Checkboxes* etc. de un ítem de la lista y la propia lista, como el tamaño (la cantidad de ítems).

ResetChat

public class ResetChat

Esta clase se encargará de borrar todo el contenido de la tabla chat de Firebase. Se emplea durante el testing para comprobar de la funcionalidad del chat en grupo.

SendNotification

public class SendNotification

Esta clase se encargará de crear y enviar una notificación *push* pública o privada para testear la funcionalidad de estas notificaciones. La API de UI Automator es la que se encargará de detectar estas notificaciones *push* en la interfaz de usuario.

2.3.3.2 Tests

En la **Tabla 2** se muestra un resumen de los tests desarrollados en Android utilizando Espresso y las librerías mencionadas anteriormente.

void	checkLoginLayoutExists() Test simple que comprobará si el layout del *Login* es correcto.
void	onEnterAsGuestClicked() Test para comprobar que los datos recuperados de WordPress (categorias) son las que esperamos.
void	onEnterAsGuestClickedAndCategoryClickedAndSearch2Clicked() El usuario hace clic en una categoría por ejemplo Asociaciones y en esta pantalla busca "IES".
void	onEnterAsGuestClickedAndCategoryClickedAndSearch2ClickedAndBack() El usuario hace clic en una categoría por ejemplo Asociaciones y en esta pantalla busca "IES".
void	onEnterAsGuestClickedAndCategoryClickedAndSearchClicked() El usuario hace clic en una categoría por ejemplo Asociaciones y en esta pantalla busca "casas".
void	onEnterAsGuestClickedAndEducationClicked() Test para comprobar que los datos recuperados de WordPress (subcategorias) educación son las que esperamos.

void	onEnterAsGuestClickedAndExitNoClicked() Test para comprobar que al pulsar *Exit*, y pulsando en "NO" no quedamos en la pantalla *Index*.
void	onEnterAsGuestClickedAndExitYesClicked() Test para comprobar que al pulsar *Exit*, y pulsando en "YES" volvemos a la vista *Login*.
void	onEnterAsGuestClickedAndNavBarClicked() Test para comprobar que la barra de navegación corresponde con las opciones que tiene el usuario invitado.
void	onEnterAsGuestClickedAndRefreshClicked() El usuario hace clic en refrescar pantalla.
void	onEnterAsGuestClickedAndSearch1Clicked() El usuario busca un canal público llamado IES Casas Nuevas introduciendo "Casas".
void	onEnterAsGuestClickedAndSearch2Clicked() El usuario introduce las iniciales IES.
void	onEnterAsGuestClickedAndSearch3Clicked() El usuario introduce "telde"
void	onEnterAsGuestClickedAndSportClicked() Test para comprobar que los datos recuperados de WordPress (subcategorias) deportes son las que esperamos.
void	onGuestClickedFollowAndOnLoginClickedFollow() En este test nos suscribiremos al canal IES El Calero utilizando la cuenta por defecto (guest).
void	onLoginClicked() El usuario entra dándole al botón *Login*.
void	onLoginClickedAndCheckChannel()

	El usuario hace clic en la categoría educación y se suscribe al canal IES El Calero y comprueba que aparece en *public channels* y *public notifications*.
void	onLoginClickedAndCreateChannelWithFloatingActionButton() El usuario entra en subcategorías, despliega el *Floating Action Button* y pulsa en *Create Channel* sin contraseña.
void	onLoginClickedAndCreateChannelWithFloatingActionButtonToasts() El usuario entra en subcategorías, despliega el *Floating Action Button* y pulsa en *Create Channel* sin contraseña y luego sin contraseña.
void	onLoginClickedAndCreateChannelWithPassWithFloatingActionButton() El usuario entra en subcategorías, despliega el *Floating Action Button* y pulsa en *Create Channel* con contraseña.
void	onLoginClickedAndEducationClicked() El usuario hace clic en la categoría educación y comprobamos también las subcategorías, subcategorías *detail* y el primer post del IES El Calero.
void	onLoginClickedAndEducationClickedAndFloatingActionButtonCheck() Test para comprobar el texto del *Floating Action Bar*.
void	onLoginClickedAndEducationClickedIESCaleroFollowClickedCheckOnROOM() El usuario hace clic en la categoría educación y se suscribe al canal IES El Calero comprueba que los datos se guardan de forma correcta localmente.
void	onLoginClickedAndFollowPublicAndPrivateChannel() El usuario se suscribe a un canal privado y a un canal público y comprueba de que cada canal está en sus secciones correspondientes.
void	onLoginClickedAndFollowTwoPrivateChannels() El usuario se suscribe a dos canales privados Canal de prueba y Canal de prueba con contraseña.
void	onLoginClickedAndFollowTwoPublicChannels() El usuario se suscribe a dos canales públicos IES El Calero y Ayuntamiento Telde (Deportes).
void	onLoginClickedAndNotReceivePrivateNotification()

	En este test el usuario se suscribe al canal privado "Canal de prueba" y espera a que le envíen un mensaje.
void	onLoginClickedAndPrivateSearchClicked() En este test el usuario en la pantalla de *Index* pulsa en buscar el canal Canal de prueba, de forma que tiene que pulsar en buscar canales privados.
void	onLoginClickedAndPublicSearchClicked() En este test el usuario en la pantalla de *Index* pulsa en buscar el canal IES El Calero, de forma que tiene que pulsar en buscar canales públicos.
void	onLoginClickedAndReceivePrivateNotificationEnteringViaPrivateChannel() En este test el usuario se suscribe al canal privado "Canal de prueba" y espera a que le envíen un mensaje.
void	onLoginClickedAndSportClicked() El usuario hace clic en la categoría deportes y comprobamos también las subcategorías, subcategorías *detail* y el primer post.
void	onLoginClickedAndSubscribeToPrivateNoPassword() El usuario accede a los canales privados e intenta entrar al canal privado sin suscribirse previamente.
void	onLoginClickedAndSubscribeToPrivateWithPassword() El usuario accede a los canales privados e intenta entrar al canal privado sin suscribirse previamente.
void	onLoginClickedAndSubscribeToPublicIESCaleroAndNotificationEnabled() El usuario registrado se suscribe al canal IES El Calero y espera a recibir una notificación.
void	onLoginClickedAndSubscribeToPublicIESCaleroAndNotificationNotEnabled() El usuario registrado se suscribe al canal IES El Calero y espera a recibir una notificación sin habilitar las notificaciones.
void	onLoginClickedTestingFloatingActionButtons() En este test se prueba el funcionamiento del *Floating action button* de la pantalla *Subcategories*.
void	onLoginClickedTestingNavBarMySection()

	En este test se prueba la sección *My section* de la barra de navegación.
void	onLoginEmptyCredentials() El usuario registrado hace clic en el botón *Login* sin introducir las credenciales.
void	onLoginEmptyPassword() El usuario registrado hace clic en el botón *Login* sin introducir la contraseña.
void	onLoginWrongCredentials() El usuario se equivoca al introducir las credenciales
void	onRegisterClicked() Test simple que comprobará que navegamos a la vista *Register*.
void	onRegisterSignUpEmailExists() Test que comprobará que el email introducido ya existe
void	onRegisterSignUpEmptyEmail() Test que comprobará que pasaría si no introducimos el email.
void	onRegisterSignUpEmptyPassword() Test que comprobará que pasaría si no introducimos la contraseña.
void	onRegisterSignUpEmptyPasswordRepeat() Test que comprobará si las contraseñas introducidas son coinciden.
void	onRegisterSignUpEmptyPasswordShort() Test que comprobará que pasaría si no introducimos una contraseña con menos de 6 caracteres.
void	onRegisterSignUpNotValidEmail() Test que comprobará si el email introducido es válido.
void	onSubCategoryEducationClickedAndIESCaleroCarnavalClicked() El usuario hace clic en Educación en IES El Calero y después en Carnaval en el IES El Calero para ver el post.
void	onSubCategoryEducationClickedAndIESCaleroClicked()

	El usuario hace clic en Educación en IES El Calero para ver las subcategorías.
void	onSubCategorySportClickedAndAyuntamientoTelde3Clicked() El usuario hace clic en Deportes en Ayuntamiento Telde y después en El Ayuntamiento mejora la iluminación del Rita Hernández para ver el post
void	onSubCategorySportClickedAndAyuntamientoTeldeClicked() El usuario hace clic en Deportes en Ayuntamiento Telde para ver las subcategorías.
void	subscribeToChannelIESCalero() El usuario se suscribe al canal IES El Calero y comprueba que aparece en *Public Channels* y después se da de baja.
void	subscribeToChannelIESCaleroAndCheckTagsButton() Esta prueba se encargará de comprobar que los botones *checkAll* y *uncheckAll* de la vista *public notifications detail* funciona correctamente.
void	subscribeToChannelIESCaleroAndIESCasasNuevas() El usuario se suscribe al canal IES El Calero e IES Casas Nuevas y comprueba que aparece en *Public Channels* y después se da de baja de esos dos canales.
void	subscribeToChannelIESCaleroAndNotReceivesNotification() El usuario se suscribe al canal IES El Calero y espera a recibir una notificación sin habilitar las notificaciones.
void	subscribeToChannelIESCaleroAndReceivesNotification() El usuario se suscribe al canal IES El Calero y espera a recibir una notificación.
void	subscribeToChannelIESCaleroAndSearchIESCalero() El usuario se suscribe al canal IES El Calero y comprueba que aparece como canales suscritos al buscarlos en *searchView* y después se da de baja.
void	subscribeToChannelIESCaleroAndUnchecksTag() El usuario se suscribe al canal IES El Calero y habilita las notificaciones, sin embargo, desactiva el tag Eventos por lo que no recibirá las notificaciones de ese tipo.
void	subscribeToChannelIESCaleroViaSearchView()

	El usuario en la pantalla de *Index* introducce en el *searchView* el nombre IES El Calero, se suscribe y comprueba que aparece en *public channels*.
void	subscribeToChannelIESCaleroViaSearchView2() El usuario en la pantalla *Subcategory* introduce en el *searchView* el nombre IES El Calero, se suscribe y comprueba que aparece en *public channels*.
void	subscribeToChannelIESCaleroViaSearchView3() El usuario en la pantalla *Index* introduce en el *searchView* el nombre IES El Calero, sin embargo, cambia de opinión e introduce Telde y se suscribe a ese canal. Luego comprueba de que esta en *public channels*.

Tabla 2. Resumen de los Tests de implementación

Capítulo VI. Guía de usuario

1. Pantallas de la aplicación móvil

En este apartado se explicarán las diferentes pantallas de la aplicación y su funcionalidad.

1.1. Splash

Esta es la primera pantalla de la aplicación, en la que se mostrará una pantalla de carga con el nombre de la aplicación y un fondo de pantalla. En la **Figura 71** se puede ver el aspecto que tiene.

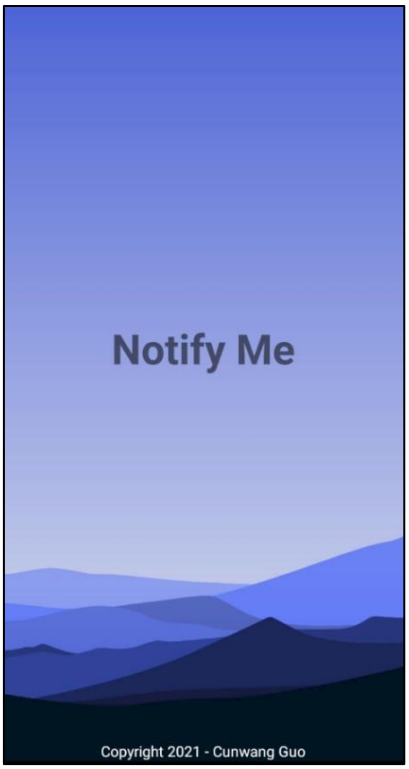


Figura 71. Pantalla Splash

1.2. Login

En esta pantalla es donde el usuario podrá darse de alta y existen dos formas: como usuario registrado pulsado en el botón *Login* o como usuario invitado pulsado en el botón *Enter as Guest*. Para entrar como usuario registrado es necesario registrarse previamente vía *Register* (el botón localizado al final).

Una vez registrado se podrá acceder introduciendo el email y la contraseña en el formulario *Login*. Si el usuario entra como usuario invitado no tendrá los mismos privilegios que un usuario registrado. Por ejemplo, no podrá visualizar ni crear canales privados.

En *Login* existen tres excepciones: email vacío, contraseña vacía y credenciales incorrectos tal y como se puede observar en las Figuras **Figura 73**, **Figura 74** y **Figura 75**.

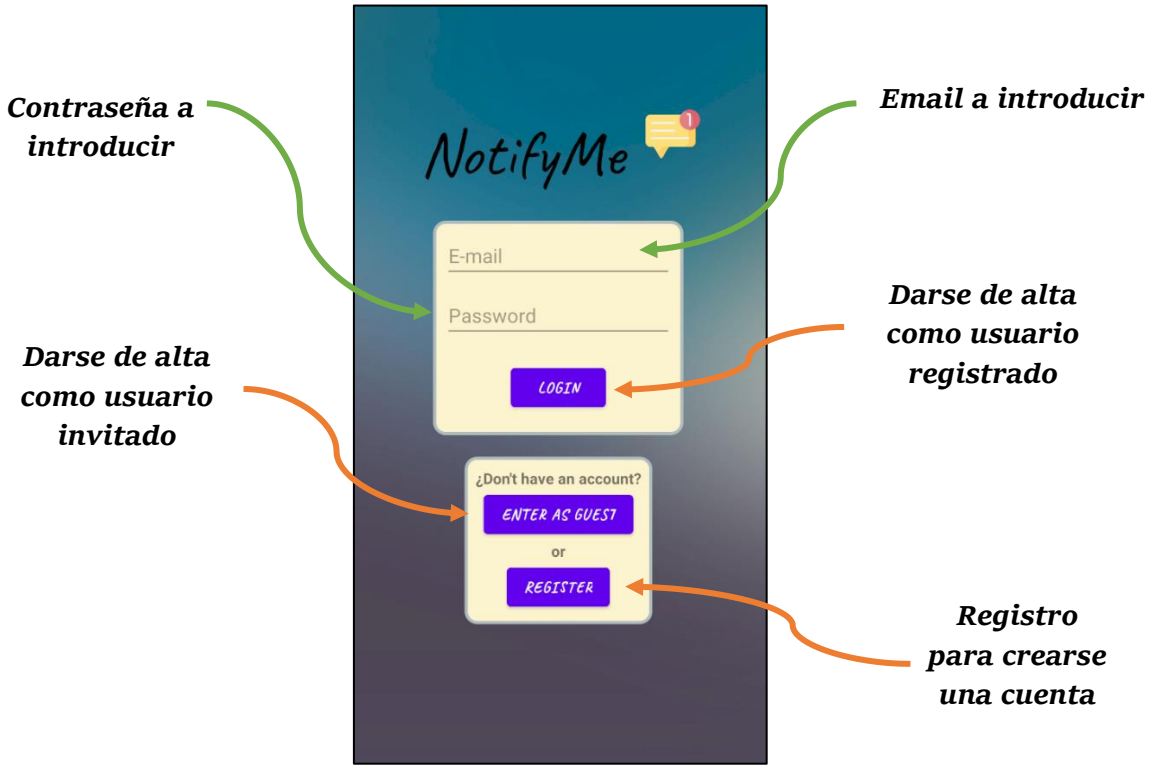


Figura 72. Pantalla Login

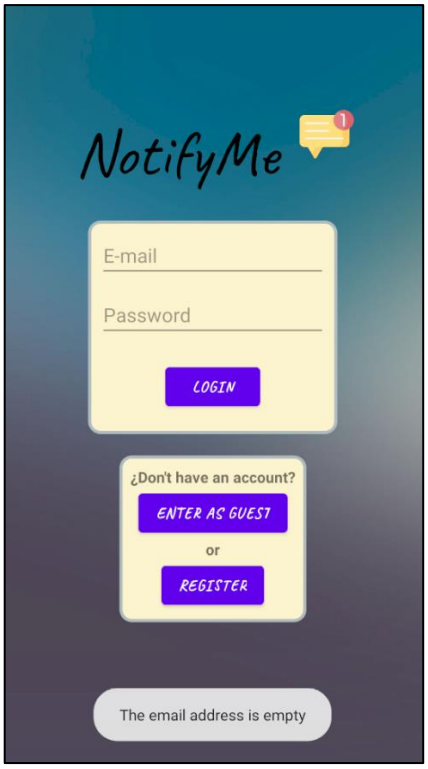


Figura 73. Pantalla Login – Email vacío

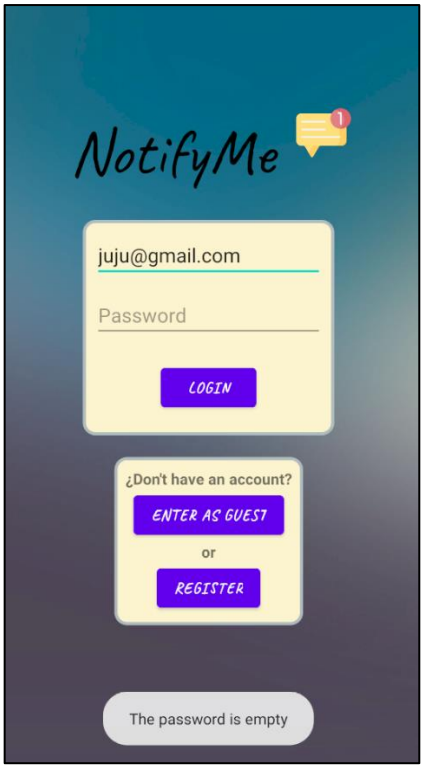


Figura 74. Pantalla Login – Contraseña vacía

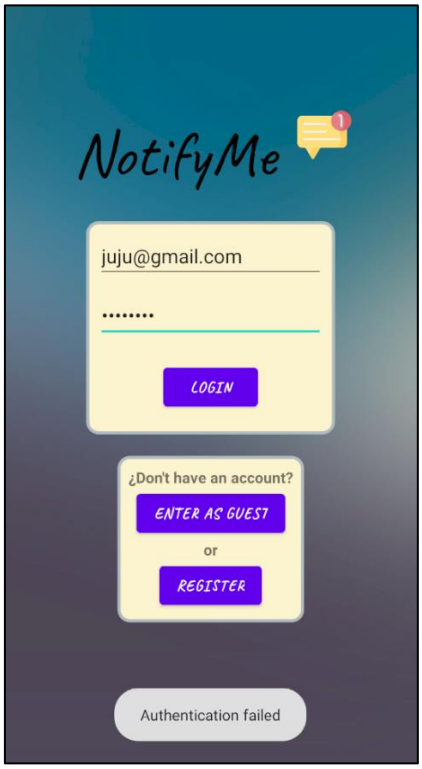


Figura 75. Login – Credenciales incorrectas

1.3. Register

En la pantalla *Register* el usuario podrá darse de alta contra Firebase introduciendo un email, y una contraseña el cual hay que repetirla para evitar posibles conflictos. El aspecto de la pantalla se puede ver en la **Figura 76**.

Hay que tener en cuenta ciertos aspectos al rellenar el formulario de registro: no introducir un email repetido, la contraseña debe tener al menos 6 caracteres de tipo alfanumérico y hay que repetir la contraseña para evitar problemas. Una vez introducido los datos, lo único que falta es pulsar en el botón *Sign up* para registrarse, y si ha sido satisfactoria la petición se le redirigirá automáticamente a la pantalla *Login*.

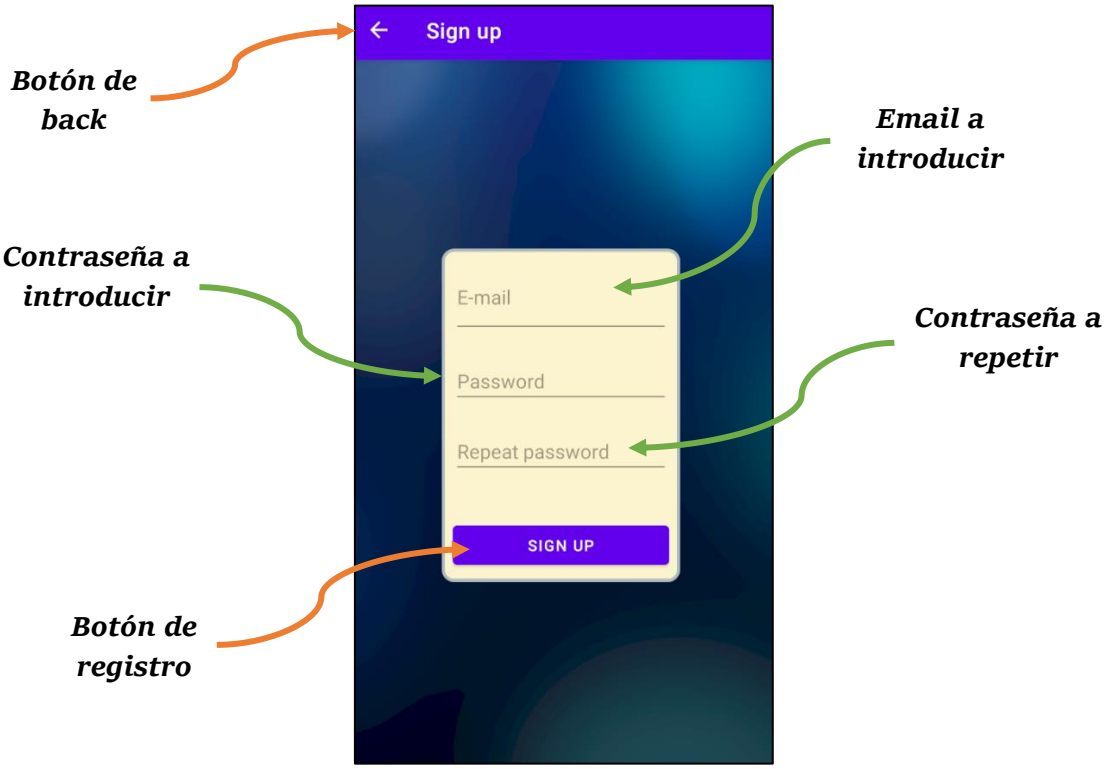


Figura 76. Pantalla Register

En las Figuras **Figura 77**, **Figura 78**, **Figura 79**, **Figura 80**, **Figura 81**, **Figura 82** y **Figura 83** se muestran las excepciones de la pantalla *Register*:

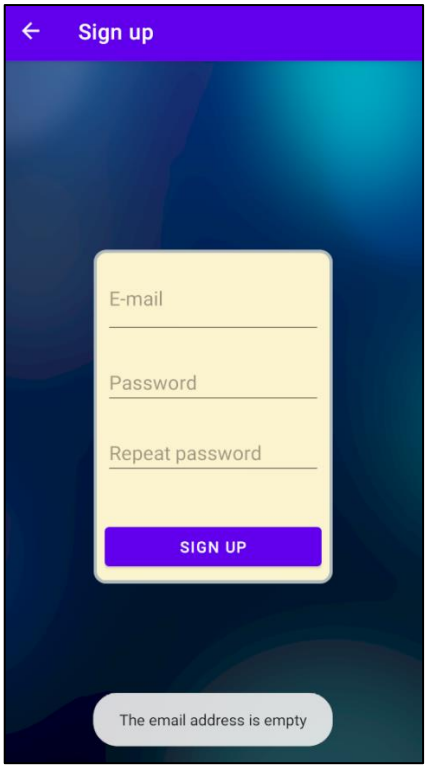


Figura 77. Pantalla Register – email vacío

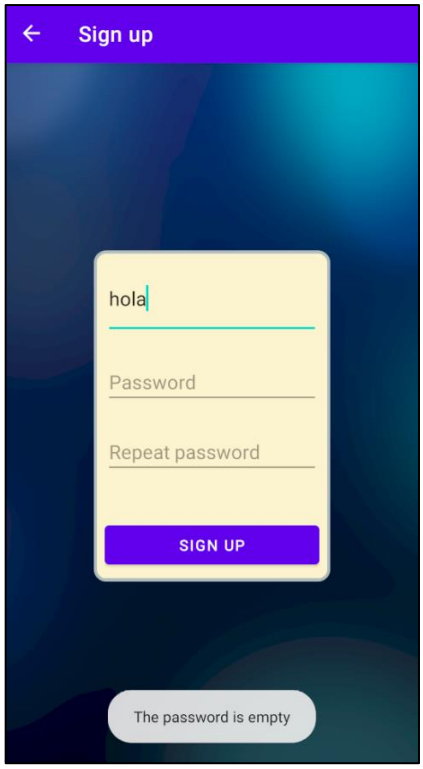


Figura 78. Pantalla Register – contraseña vacía

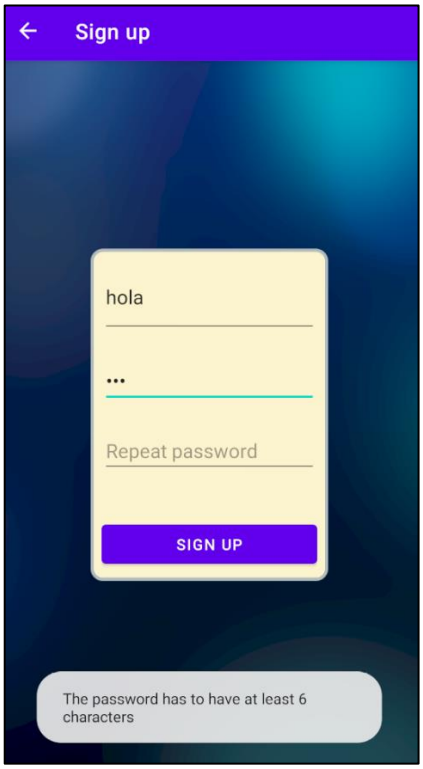


Figura 79. Pantalla Register – la contraseña no cumple con los requisitos

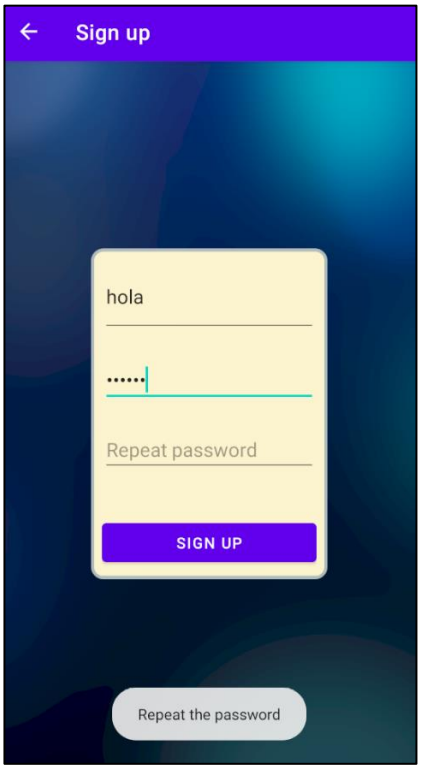


Figura 80. Pantalla Register – no se ha repetido la contraseña

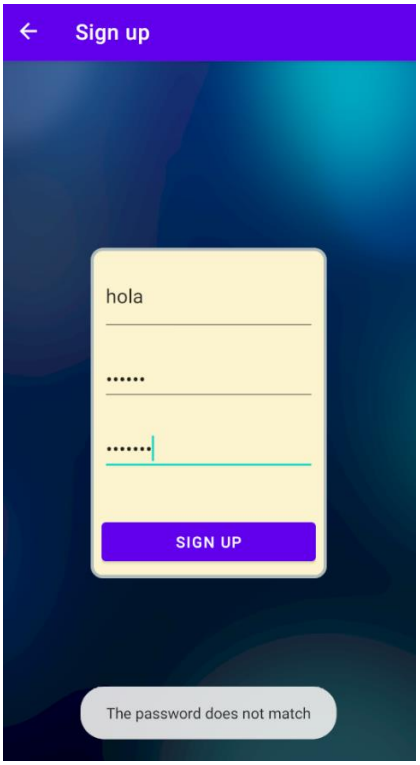


Figura 81. Pantalla Register – las contraseñas no coinciden

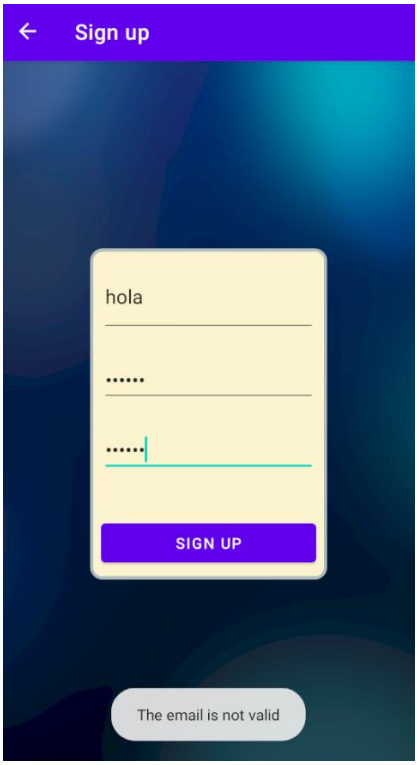


Figura 82. Pantalla Register – el email no es valido

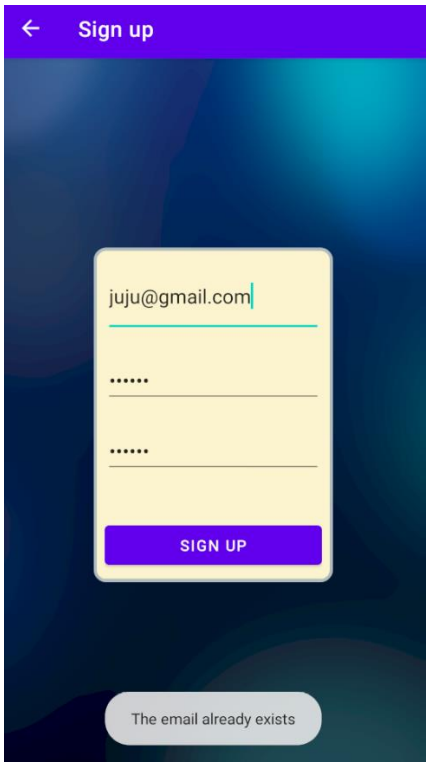


Figura 83. Pantalla Register – el email ya existe

1.4. Index

En la pantalla *Index* es donde se mostrará todas las categorías principales: Asociaciones, Ayuntamiento y Cabildos, Deporte, Educación, Empresas, Ocio y Cultura y Parroquias, tal y como se puede visualizar en la **Figura 84**.

Cada una de las categorías mencionadas anteriormente tendrá sus propios canales y subcanales, por ejemplo, en el caso de educación los canales podrían ser el nombre de un colegio, instituto, universidad etcétera.

El usuario en esta pantalla puede acceder a cualquier categoría, por ejemplo, si está interesado en el tema de los deportes puede pulsar en la opción *Deporte*. Dentro de ésta se encontrará con todo tipo de polideportivos, competiciones deportivas etc.

La pantalla *Index* también tendrá un Toolbar y en él podemos destacar la barra de navegación lateral, que nos permitirá navegar a las diferentes pantallas de forma rápida y eficaz. Por otro lado, está el buscador de canales que nos posibilitará buscar un canal específico velozmente, y por último la opción de refrescar que tiene objetivo actualizar la pantalla *Index*.

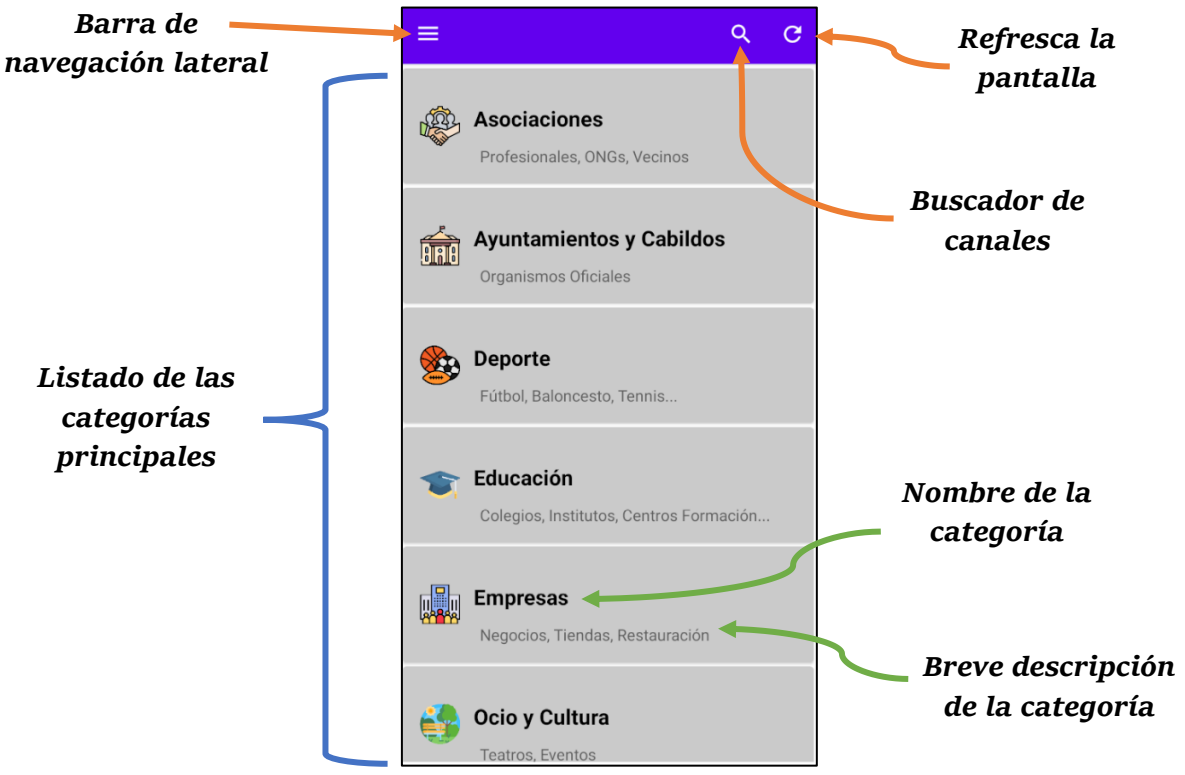


Figura 84. Pantalla Index

Tal y como se había mencionado anteriormente el usuario no registrado tendrá menos privilegios y por tanto la barra de navegación lateral difiere del usuario registrado. En las Figuras **Figura 85** y **Figura 86**, se puede apreciar la diferencia:

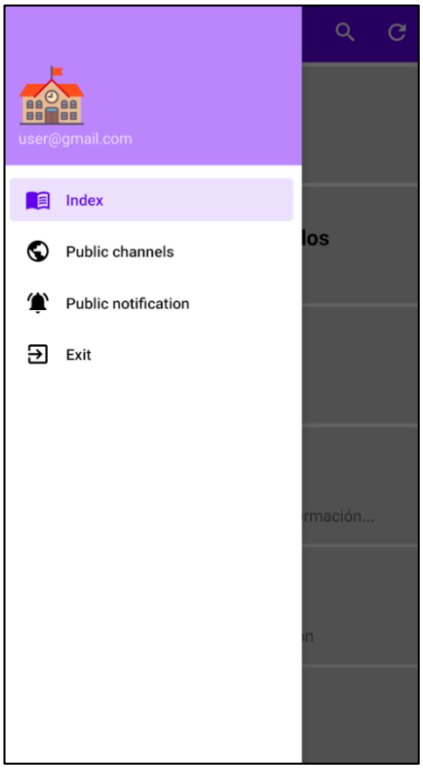


Figura 85. Pantalla Index – barra de navegación lateral (usuario invitado)

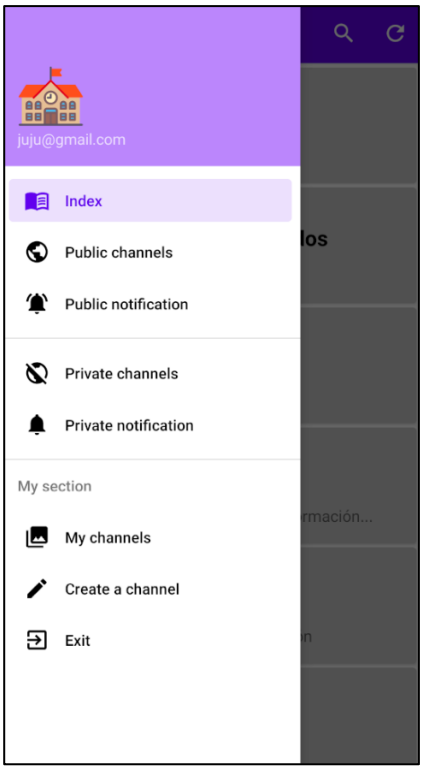


Figura 86. Pantalla Index – barra de navegación lateral (usuario registrado)

Se puede observar que usuario invitado solo tiene 4 opciones mientras que el usuario registrado posee 8 opciones. Las opciones del usuario registrado son:

- **Index:** pantalla donde se presentan las categorías principales.
- **Public channels:** pantalla donde se almacenan los canales públicos.
- **Public notifications:** pantalla donde se configuran las notificaciones de los canales públicos.
- **Private channels:** pantalla donde se almacenan los canales privados.
- **Private notifications:** pantalla donde se configuran las notificaciones de los canales privados.
- **My channels:** pantalla donde se mostrará los canales que ha creado el usuario.
- **Create channels:** pantalla en la que se crean los canales privados.
- **Exit:** vuelve a la pantalla Login.

Si analizamos la **Figura 86**:

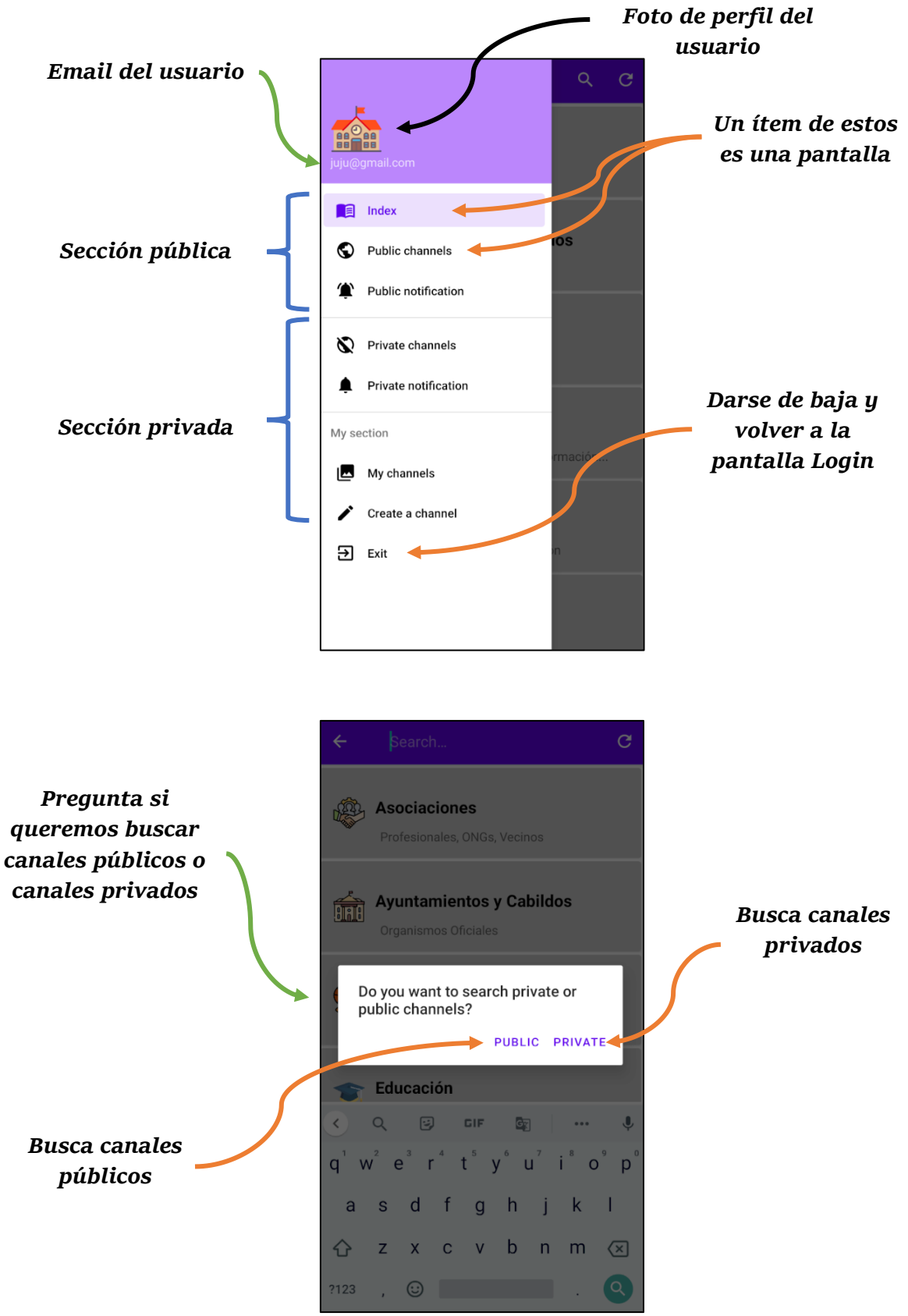


Figura 87. Pantalla Index – buscador de canales

En la **Figura 87** se muestra el AlertDialog cuando se pulsa en el icono de lupa del *Toolbar* que nos preguntará si queremos buscar canales públicos o canales privados. Si el usuario ha entrado como usuario invitado entonces no se mostrará el AlertDialog y se buscará directamente en los canales públicos, puesto que un usuario invitado no tiene acceso a los canales privados.

Ahora, si el usuario ha elegido la opción privada entonces se mostrará otro AlertDialog, el de la **Figura 88**, preguntando en que categoría se quiere buscar ese canal:

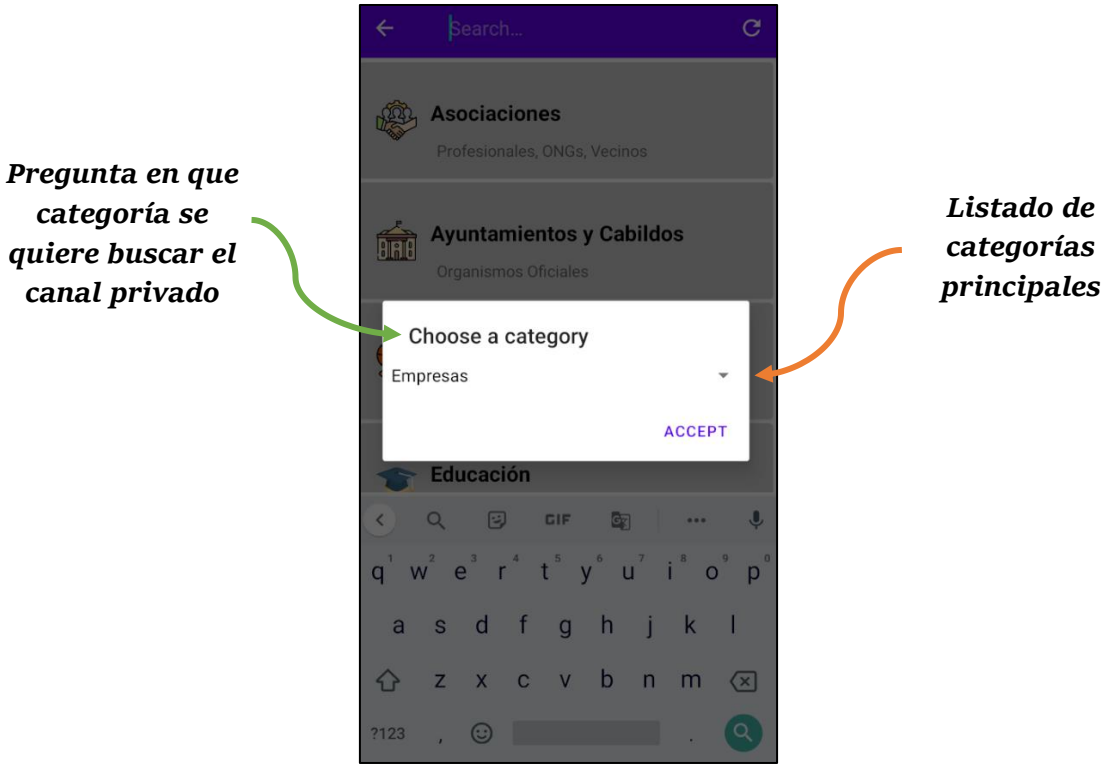


Figura 88. Pantalla Index – buscador de canales privados

En la **Figura 89** se introduce un texto de ejemplo en el buscador de canales:

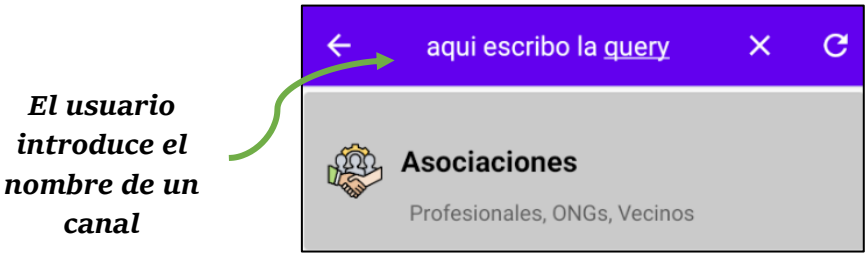


Figura 89. Pantalla Index – introduciendo texto en el buscador de canales

Finalmente, si el usuario quiere darse de baja puede hacerlo de dos formas: pulsando en el botón de *back* o pulsando en *Exit* (localizado en la barra de navegación). En la **Figura 90** se muestra el AlertDialog para salir de la cuenta.

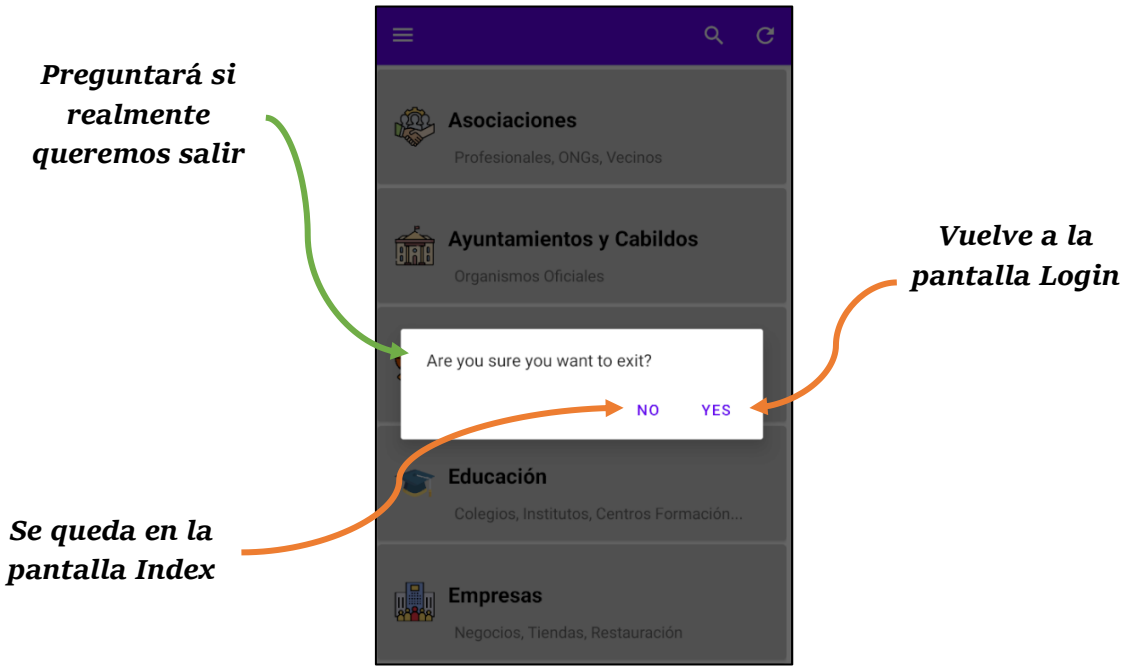


Figura 90. Pantalla Login – saliendo de la cuenta

1.5. SubCategory

En esta pantalla se mostrarán todos los canales de una categoría, y en ella podremos seguir a los canales que nos sean de interés. Estos canales que sigamos se mostrarán en la pantalla *PublicChannels* o *PrivateChannels* si el canal es privado.

Al igual que *Index* tendrá un Toolbar en el que existirá un buscador de canales para buscar un canal específico y estará también la función de refrescar. Asimismo, hay un FAB (Floating ActionButton) que nos permite intercambiar de modo, es decir si estamos visualizando los canales públicos y pulsamos en el botón de *switch,* entonces se mostrarán los canales privados y viceversa. Aparte del cambio de modo el FAB nos permitirá también crear canales privados con contraseña o sin contraseña y navegar a la pantalla *MyChannels*. Estas últimas dos opciones realizan la misma funcionalidad que los dos botones que aparecen en la barra de navegación lateral *My section* de la **Figura 86**.

El funcionamiento del botón *Switch* se puede ver entre las Figuras **Figura 91** y **Figura 92**.

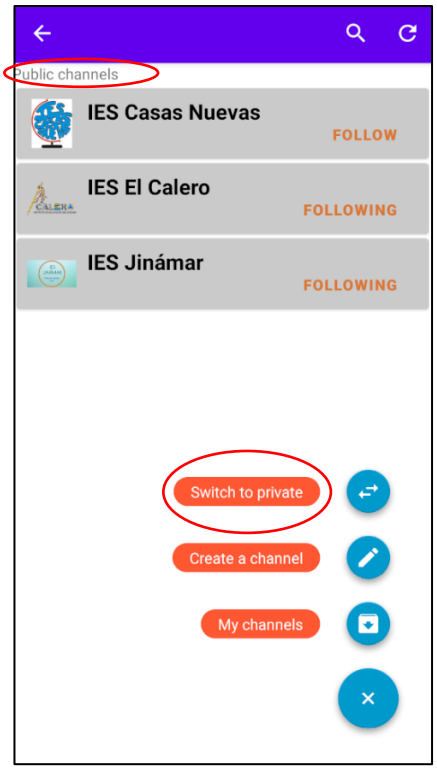


Figura 91. Pantalla SubCategory – canales públicos

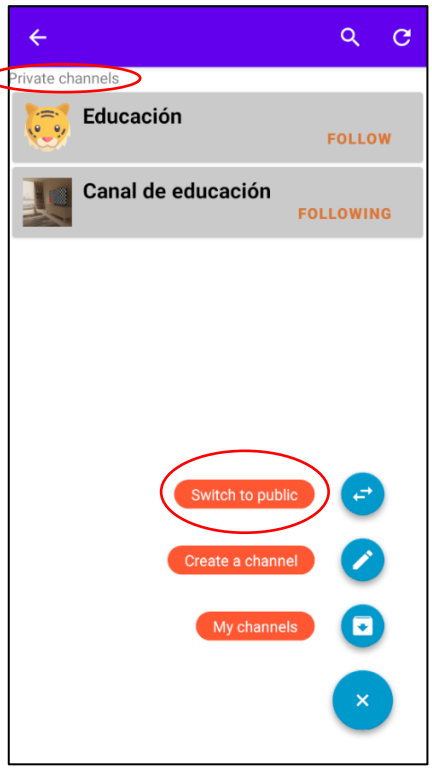
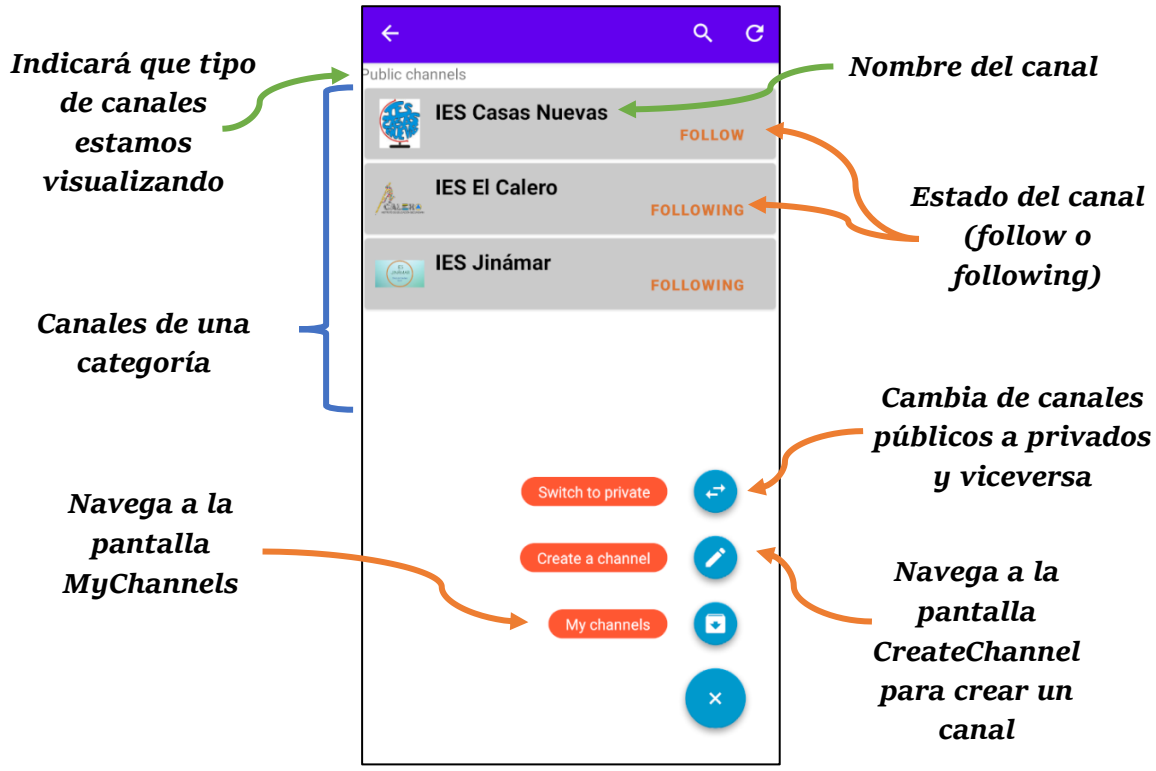


Figura 92. Pantalla SubCategory – canales privados



Ahora, si pulsamos en el botón del FAB *Create a channel* se mostrará un AlertDialog, la de la **Figura 93** indicando si queremos crear un canal con contraseña o un canal sin contraseña.

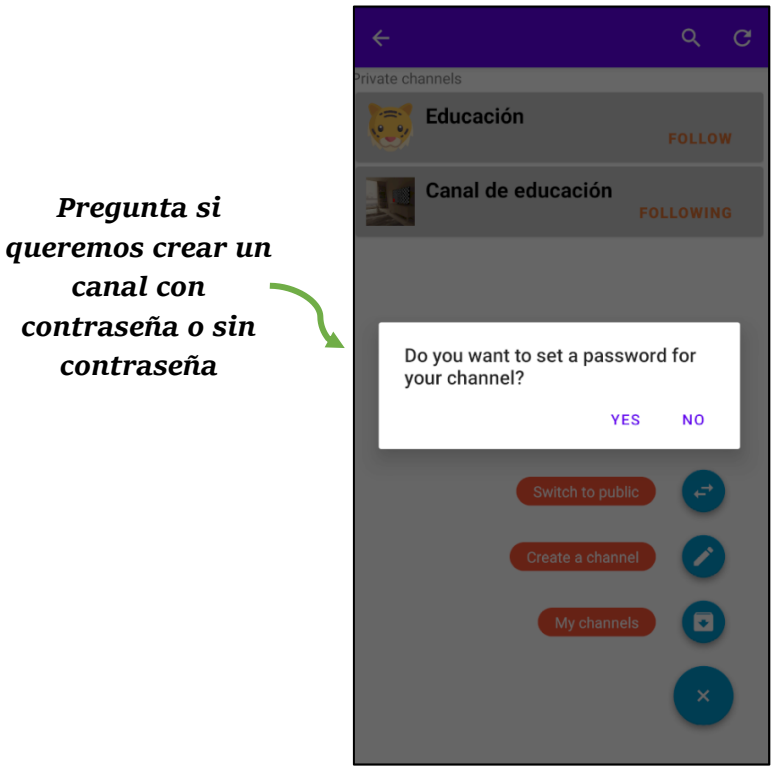


Figura 93. Pantalla SubCategory – AlertDialog para crear un canal con contraseña o sin contraseña

Finalmente, si nos queremos suscribir a un canal privado y esta tiene contraseña aparecerá el AlertDialog de la **Figura 94**.

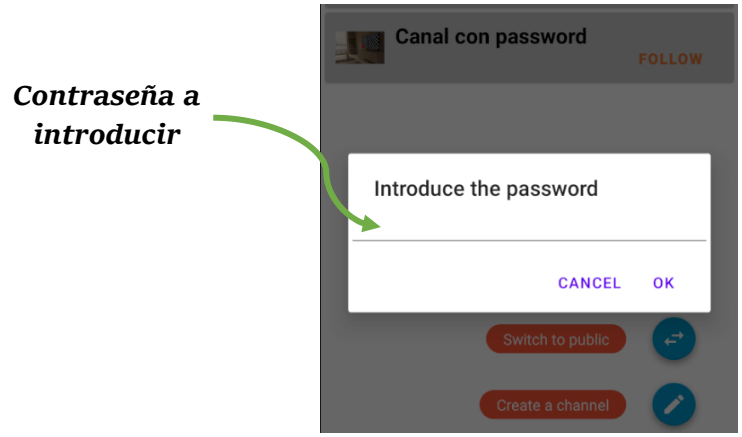


Figura 94. Pantalla SubCategory – AlertDialog canal con contraseña

Si la contraseña introducida es incorrecta se mostrará un mensaje Toast como el de la **Figura 95**:

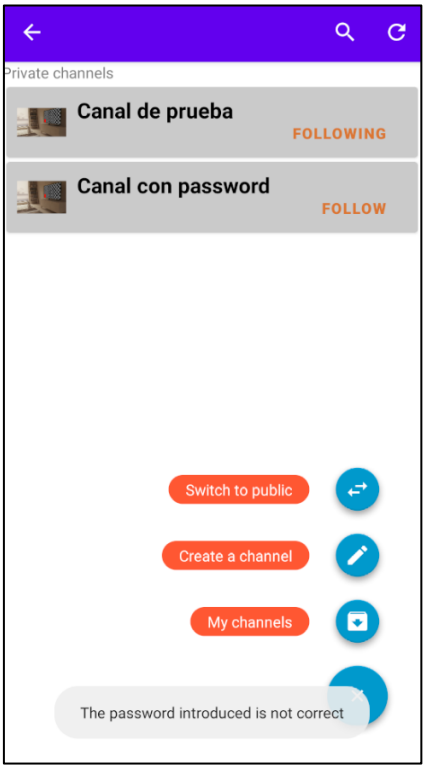


Figura 95. Pantalla SubCategory – contraseña incorrecta

1.6. SubCategoryDetail

La pantalla *SubCategoryDetail* se encargará de mostrar todas las publicaciones o subcanales de un canal. Si pulsamos en una de las publicaciones nos llevará a la pantalla *Posts* donde se contemplará el contenido de dicha publicación.

Al igual que las pantallas anteriores existirá un Toolbar con el botón de *back* y el botón de *refrescar* la pantalla.

En la **Figura 96** se muestra las publicaciones del canal IES El Calero:

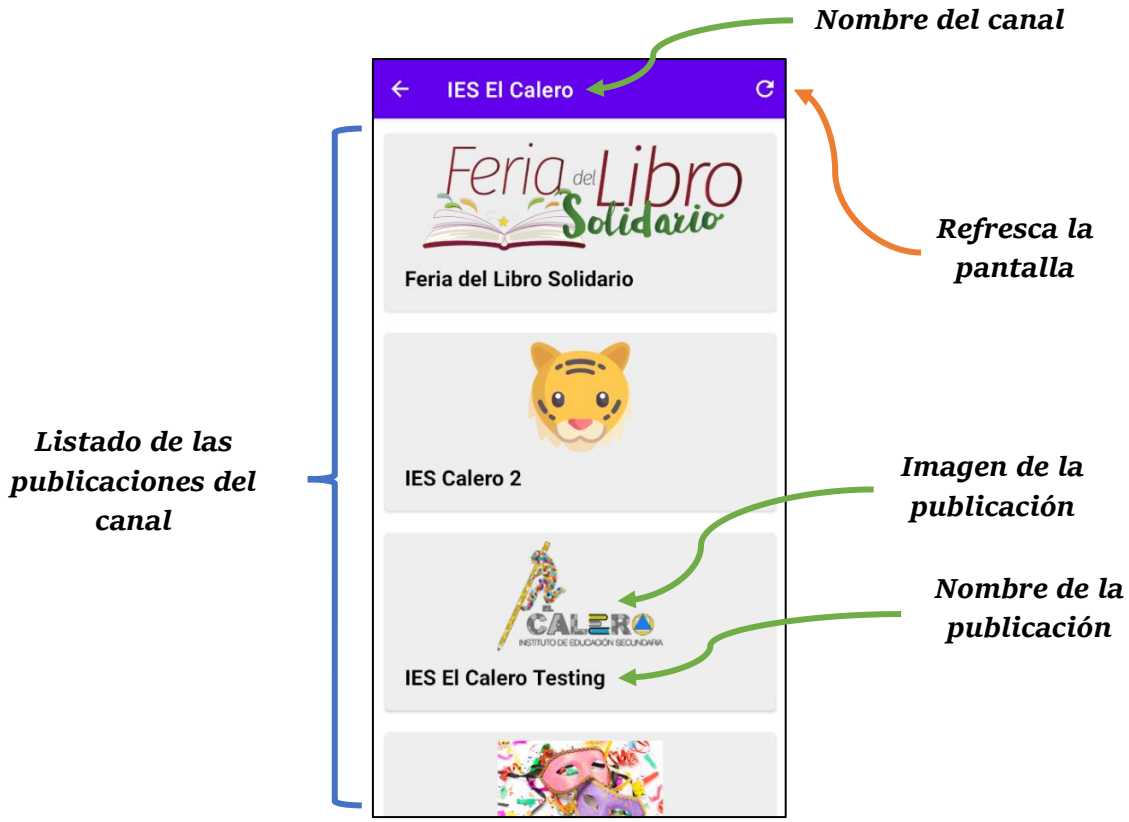


Figura 96. Pantalla SubCategoryDetail

1.7. Posts

En la pantalla *Posts* se mostrará el detalle de una publicación, así como el título de la publicación, la fecha en la que se ha creado, la imagen y el contenido de la publicación. Si el cuerpo de la publicación es muy largo se podrá realizar un *scrolling*. Además, existirá en el Toolbar un botón con forma *share* para compartir la publicación con otros usuarios, amigos familiares etc.

En el cuerpo de la publicación podrá aparecer hipervínculos a PDFs o a otras páginas (como las páginas oficiales de donde se extrajo la publicación).

Entre las Figuras **Figura 97** y **Figura 98** se puede ver el efecto del *scrolling*:

Figura 97. Pantalla Posts I

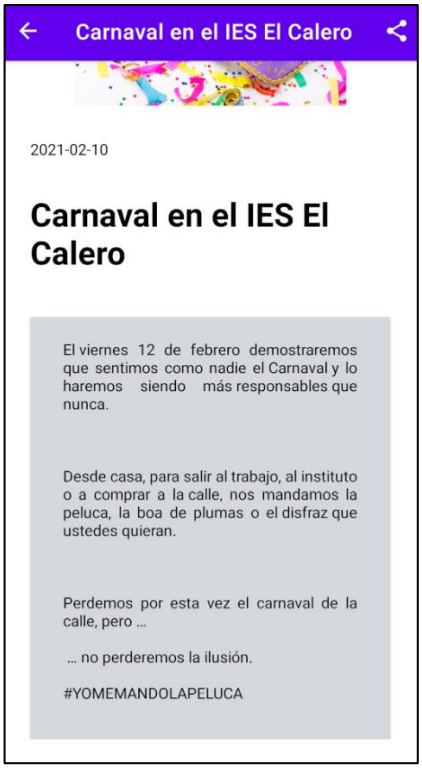


Figura 98. Pantalla Posts II

Los componentes de la pantalla son:

En la **Figura 99** se muestra la funcionalidad del botón *share*.

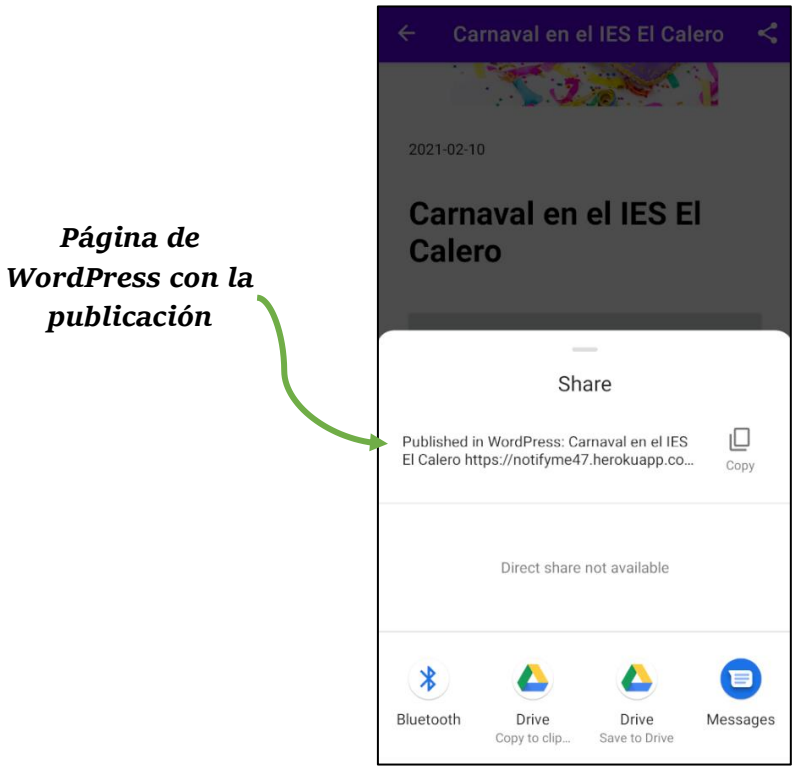


Figura 99. Pantalla Posts – compartiendo la publicación

1.8. PublicChannels

En la pantalla *PublicChannels* se mostrarán todos los canales públicos que estamos siguiendo, es decir todos aquellos canales en el que hemos pulsado *follow* desde la pantalla *SubCategory* aparecerán aquí. Si ya no nos interesa un canal podemos darnos de baja dándole al botón *unfollow*. En la **Figura 100** se puede ver el aspecto de esta pantalla.

Como se había visto anteriormente, esta pantalla se accede a través de la barra de navegación lateral, por ejemplo, desde la pantalla *Index*.

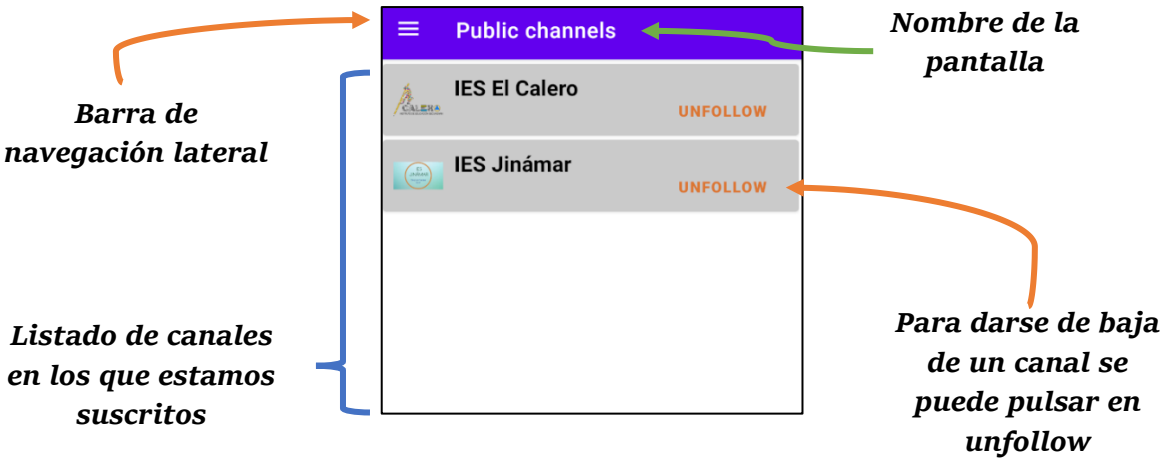


Figura 100. Pantalla PublicChannels

1.9. PublicChannelsDetail

Si pulsamos en uno de los canales de la pantalla *PublicChannels* navegaremos a esta pantalla, la de la **Figura 101**, donde se mostrará las publicaciones de ese canal. El aspecto es igual que la pantalla *SubCategoryDetail*, sin embargo, los datos son recuperados de la base de datos Room y no de WordPress.

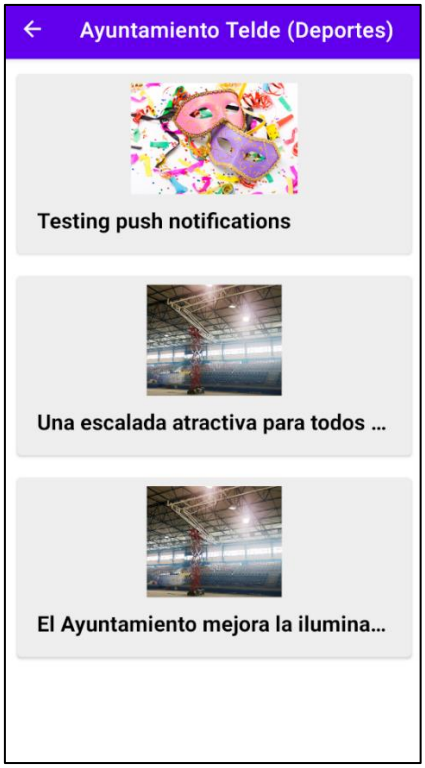


Figura 101. Pantalla PublicChannelsDetail

1.10. Search

Si buscamos un canal ya sea desde la pantalla *Login* o desde la pantalla *SubCategory* se navegará a la pantalla *Search* el cual tiene el mismo aspecto que la pantalla *SubCategory*.

En este ejemplo, la de la **Figura 102**, se ha buscado por el nombre *Calero* desde la pantalla *Index* y como se puede observar aparece únicamente ese canal, ya que no existen otros canales con ese nombre en WordPress. Asimismo, se podrá buscar canales desde la pantalla *Search*.

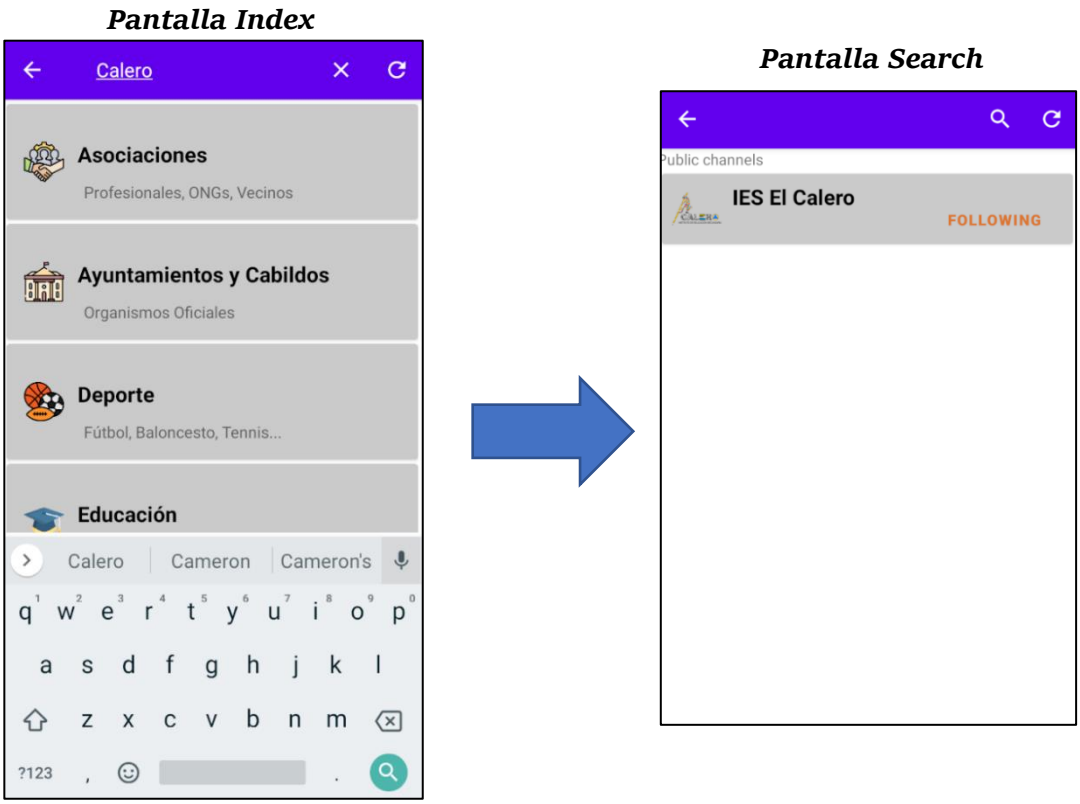


Figura 102. Buscando el canal IES El Calero

1.11. PublicNotifications

En la pantalla *PublicNotifications* se podrá configurar las notificaciones públicas, es decir podremos activar o desactivar las notificaciones de un canal, tal y como se puede ver en la **Figura 103**.

Existirá también un icono en forma de campana en el Toolbar que si pulsamos en ella nos dirigirá a la pantalla *PublicNotificationsHistory* para visualizar las notificaciones recibidas almacenadas.

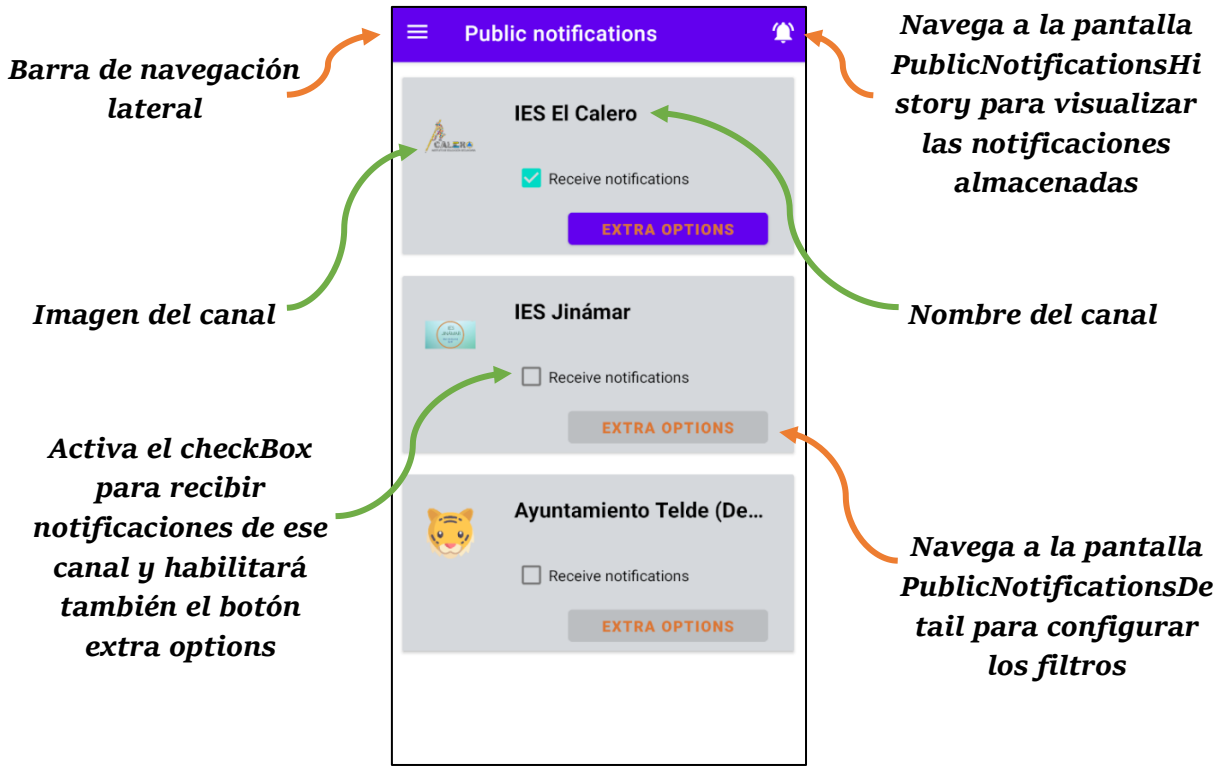


Figura 103. Pantalla PublicNotifications

1.12. PublicNotificationsDetail

Si pulsamos en el botón *extra options* (debemos tener marcado el checkBox previamente) desde la pantalla *PublicNotifications* nos dirigirá a esta pantalla donde se mostrará un listado de las etiquetas o *tags* que tiene el canal. Esto permite al usuario

personalizar las notificaciones que reciba, por ejemplo, un usuario puede estar interesando en el IES El Calero, pero solo de aquellas noticias relacionadas con los torneos (ajedrez, fútbol, lucha canaria ...). El aspecto de esta pantalla se puede observar en la **Figura 104**.

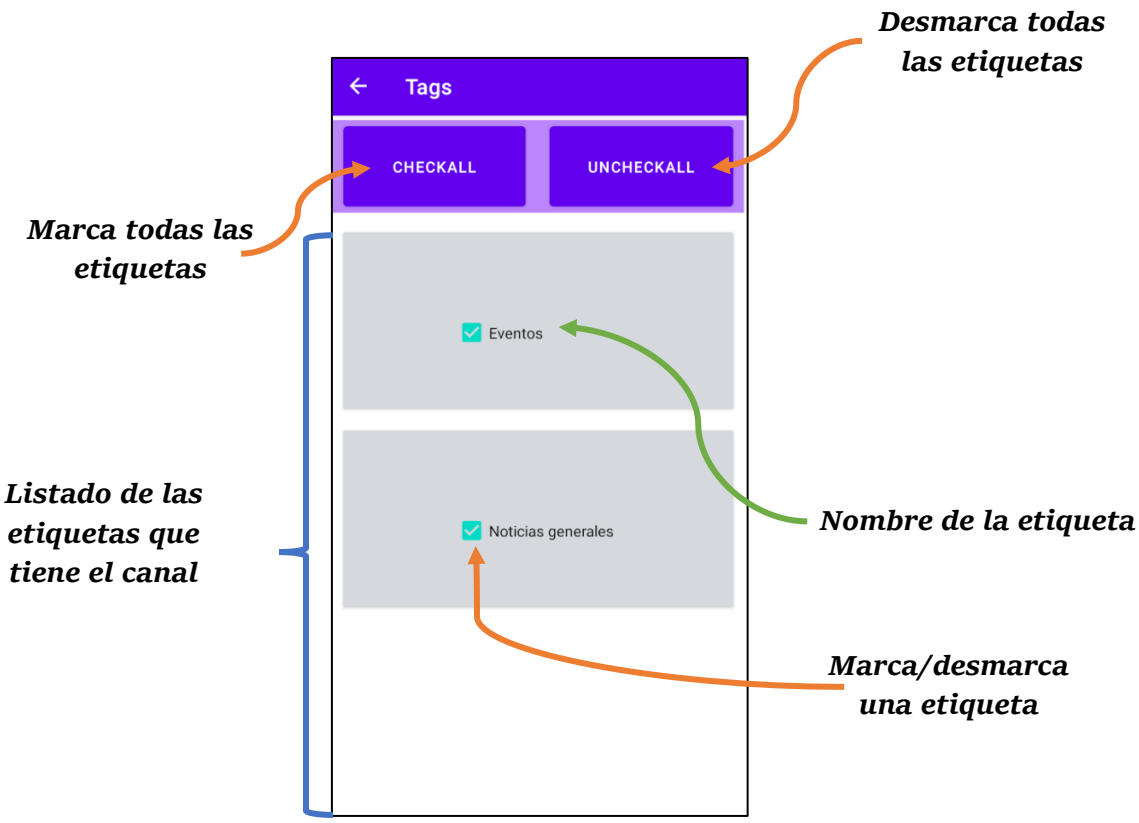


Figura 104. Pantalla PubicNotificationsDetail

1.13. PublicNotificationsHistory

En esta pantalla, la de la **Figura 105**, se almacenarán las notificaciones que recibamos de los canales públicos en los que estamos suscritos solo si tenemos las notificaciones activadas y las etiquetas marcadas. Asimismo, se puede eliminar una notificación pulsando en el botón *dismiss*.

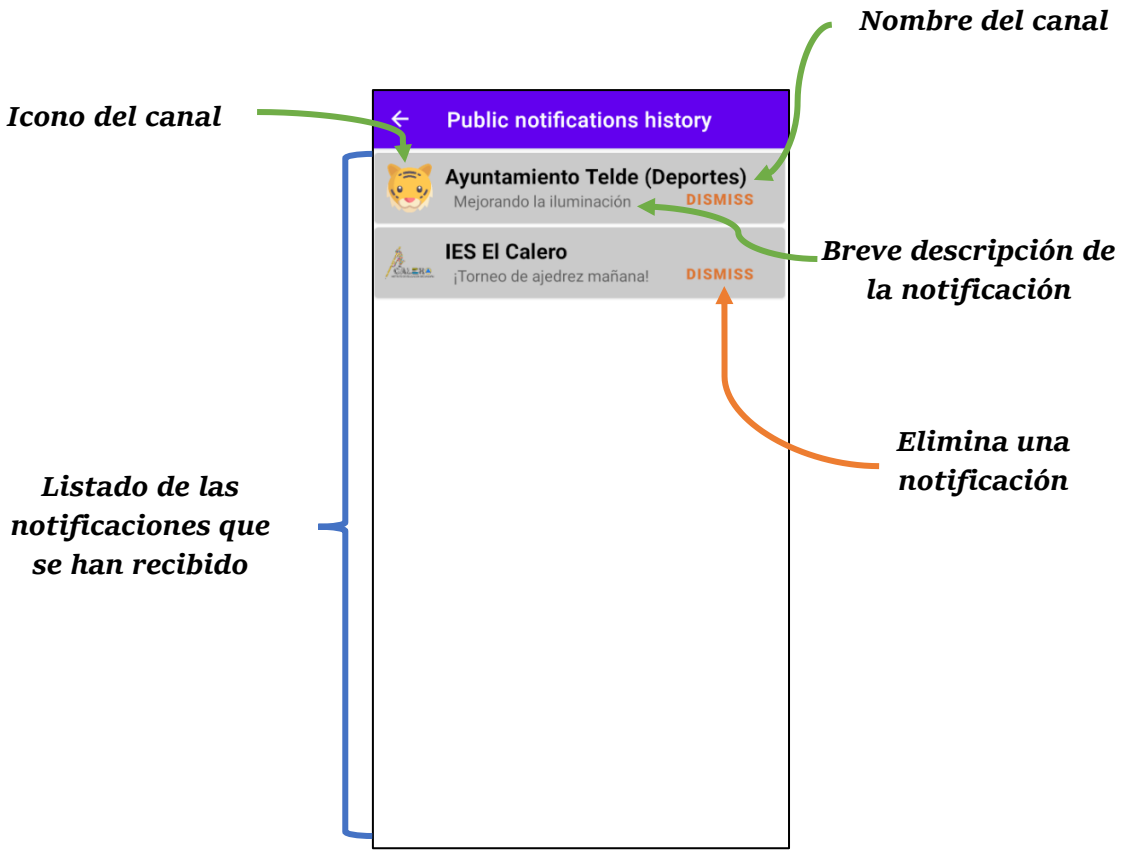


Figura 105. Pantalla PublicNotificationsHistory

1.14. PrivateChannel

Esta pantalla es casi igual que la pantalla *PublicChannels* salvo que ahora se muestran los canales privados de Firebase, es decir si nos hemos suscrito a un canal privado desde la pantalla *SubCategory* entonces aparecerán en esta pantalla.

Al igual que la pantalla *PublicChannels* se puede visualizar el nombre y la imagen del canal y también nos permitirá cancelar la suscripción si ya no nos interesa ese canal. El aspecto de esta pantalla se ilustra en la **Figura 106**.

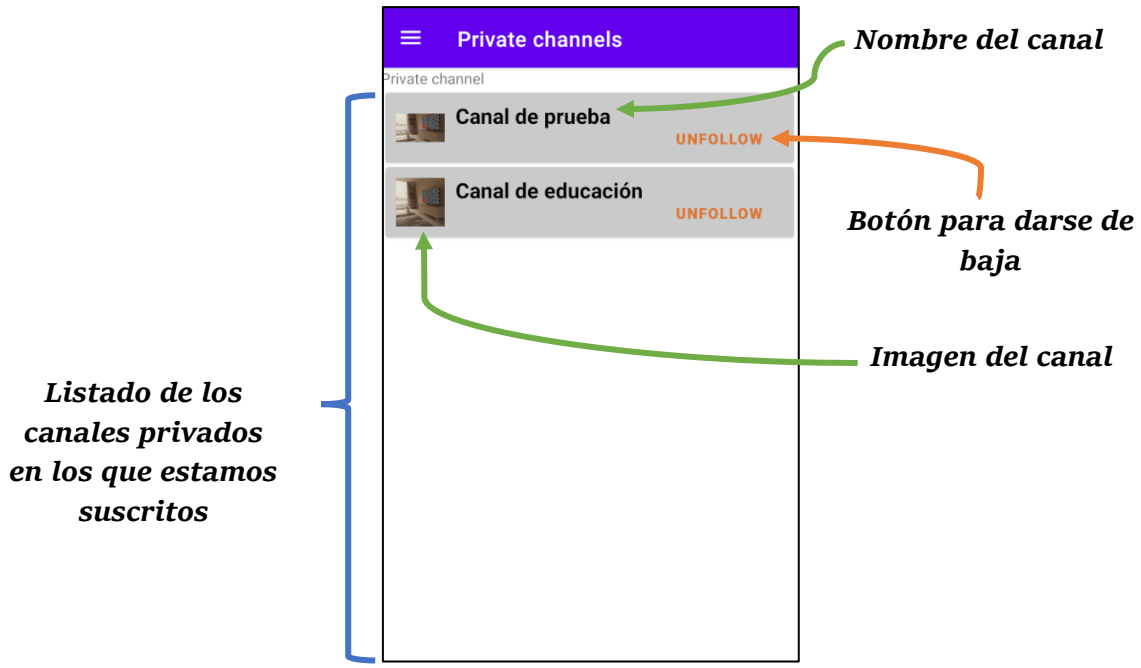


Figura 106. Pantalla PrivateChannels

1.15. PrivateNotifications

Los canales privados a diferencia de los canales públicos no tienen *tags,* solamente podemos activar o desactivar las notificaciones del canal, tal y como aparece en la **Figura 107**. Esto es debido a que los canales privados están formados por un chat grupal que comentaremos más en detalle en el punto 1.16.

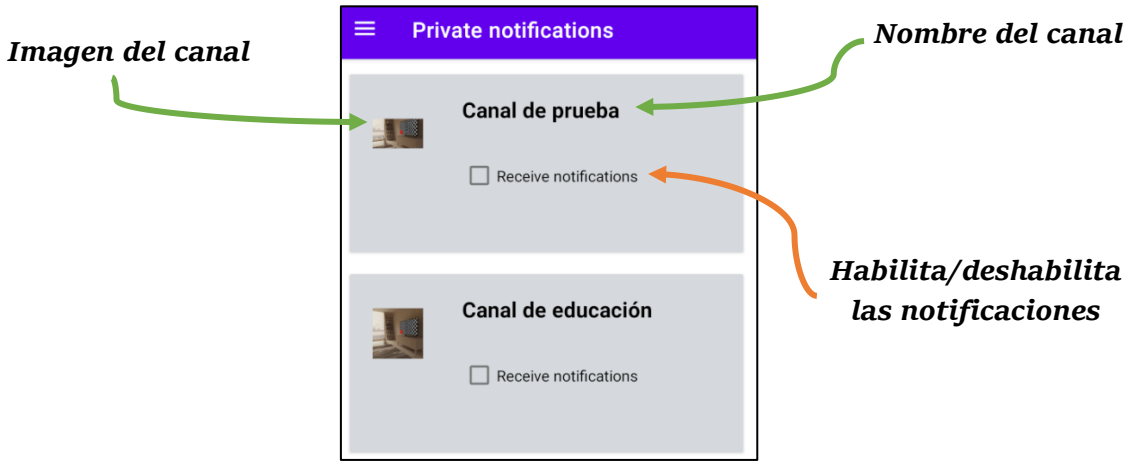


Figura 107. Pantalla PrivateNotifications

1.16. PrivatePosts

En esta pantalla, la de la **Figura 108**, se muestra un chat grupal que nos permite comunicarnos con otros usuarios que están suscritos al canal. En el chat grupal se puede conversar sobre todo tipo de asuntos relacionados con el tema del canal.

A esta pantalla se puede acceder de tres formas: desde la pantalla *SubCategory* (canales privados), desde *PrivateChannels* (si estamos suscritos) y desde *MyChannels* (Si el canal es creado por nosotros).

Por otra parte, en el Toolbar existe el menú con tres puntos que permitirá al usuario navegar a la pantalla *ChannelsInformations* para visualizar el detalle del canal o para navegar a la pantalla *Participants* para ver el listado de usuarios que están participando. En la **Figura 109** se puede apreciar le menú.

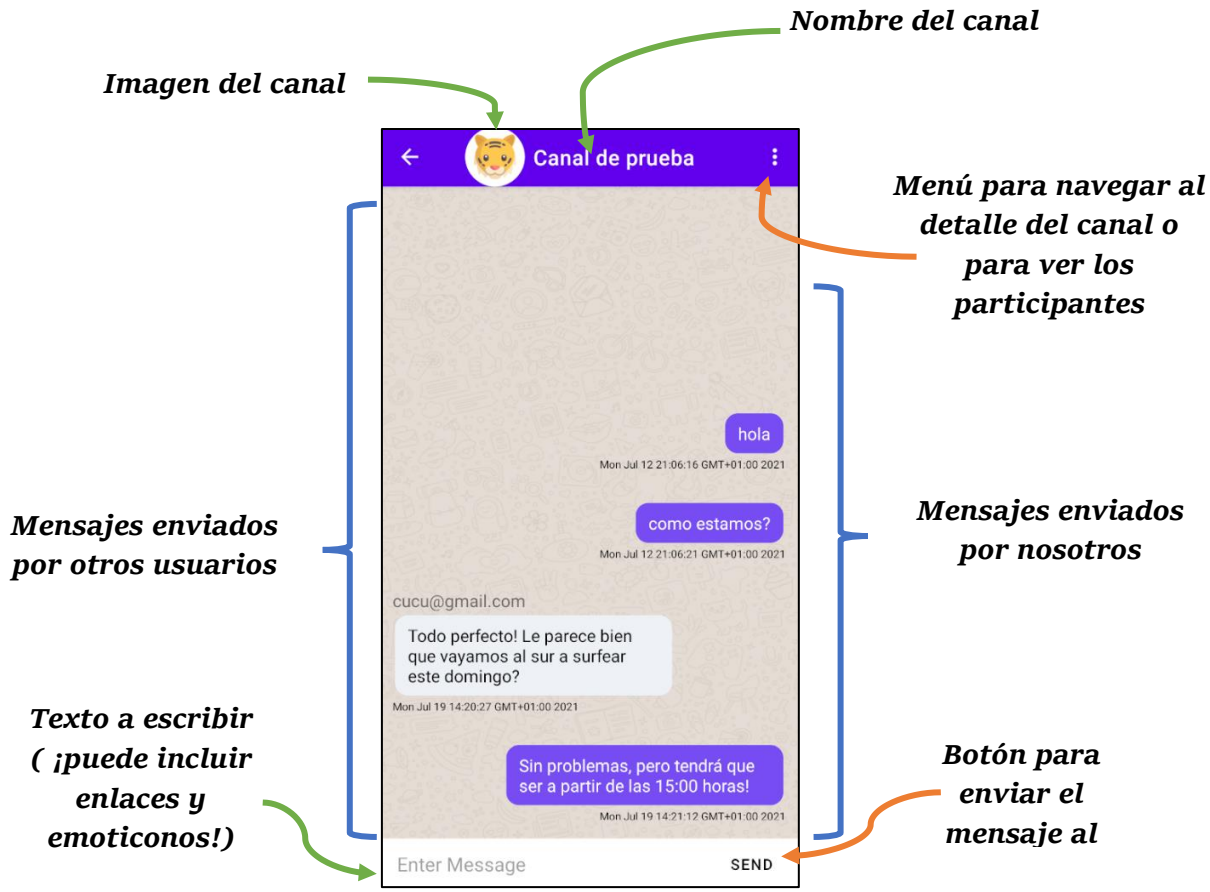


Figura 108. Pantalla PrivatePosts

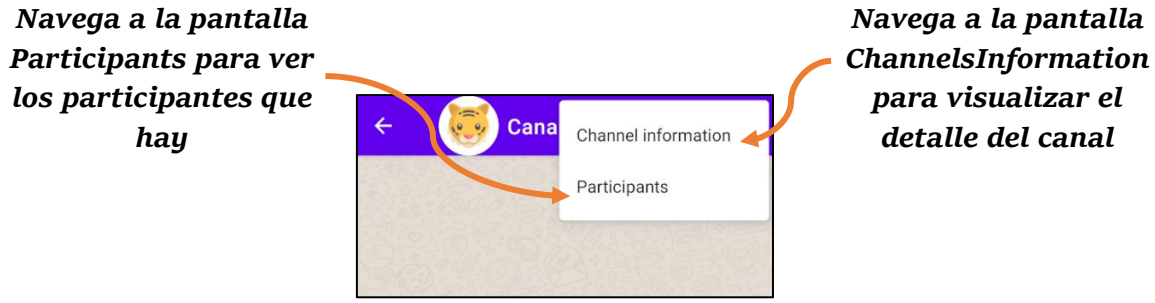


Figura 109. Pantalla Private Posts – menú desplegado

1.17. MyChannel

En la pantalla *MyChannels* se presentarán todos los canales (privados) creados por el propio usuario desde la pantalla *CreateChannels*. Si pulsamos en uno de los canales no dirigirá a la pantalla *PrivatePosts* y se mostrará el chat grupal.

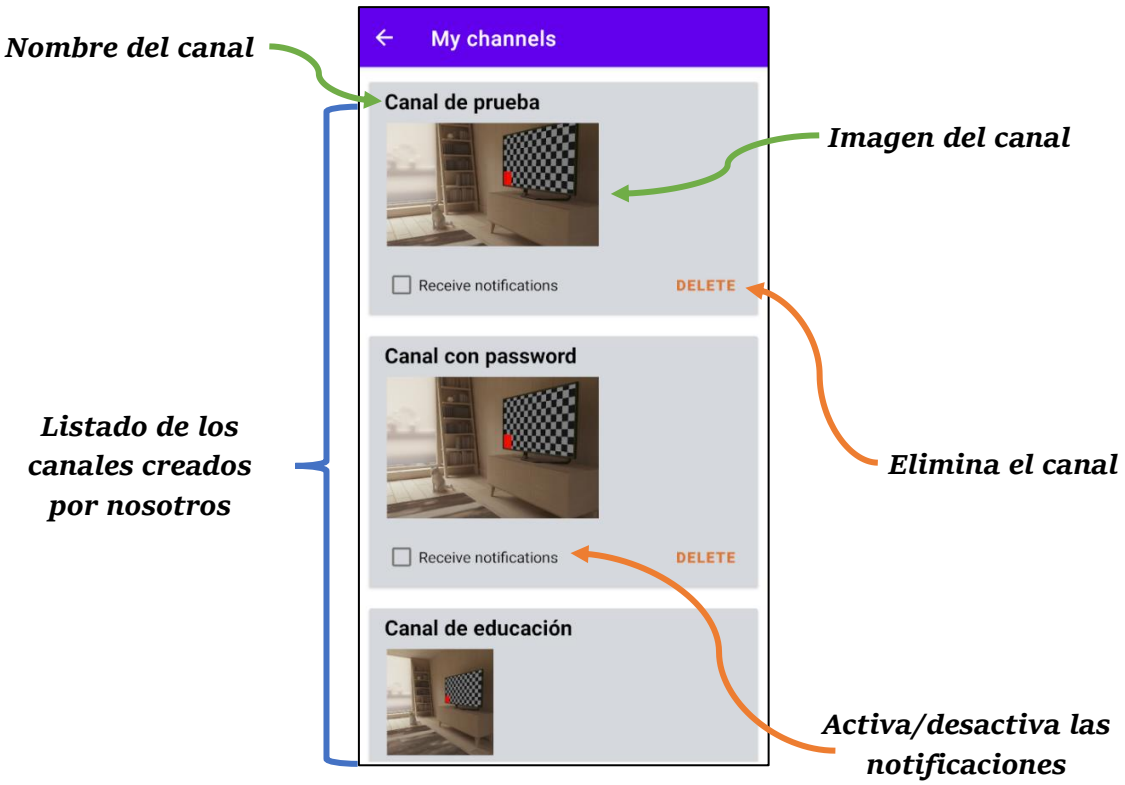


Figura 110. Pantalla MyChannels

Tal y como se puede observar en la **Figura 110** podremos activar o desactivar las notificaciones de los diferentes canales e incluso podemos eliminar un canal. Sin embargo, al pulsar en *delete* nos aparecerá en la pantalla un AlertDialog, la de la **Figura 111**, preguntando si realmente queremos eliminar ese canal.

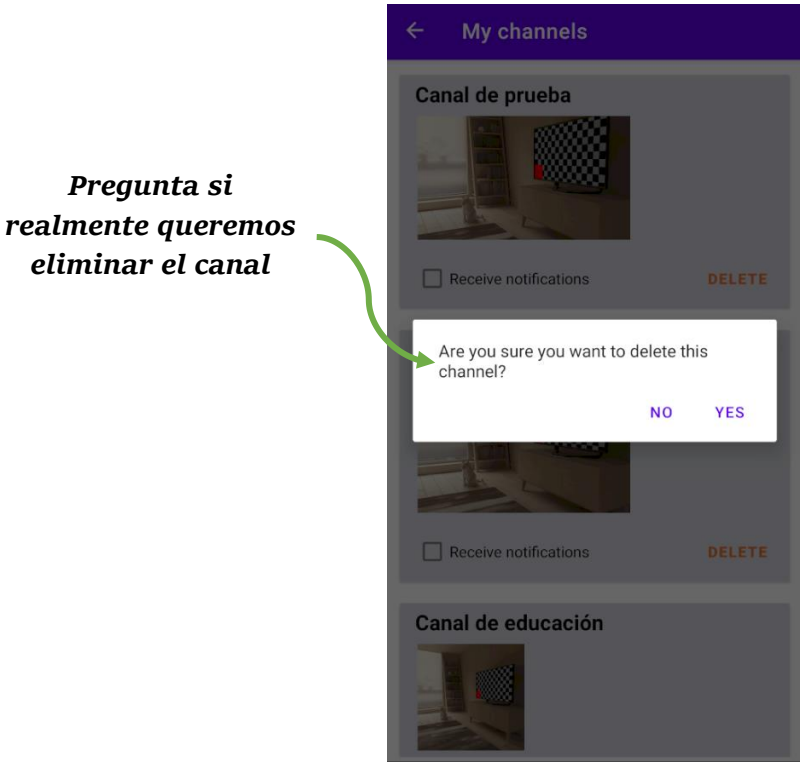


Figura 111. Pantalla MyChannels – AlertDialog eliminación de un canal

1.18. CreateChannel

En esta pantalla el usuario podrá crear su propio canal privado en la que tendrá que introducir el nombre, la descripción, la categoría, la contraseña (si ha elegido crear un canal con contraseña) y la imagen (si no se elige se utiliza uno por defecto) tal y como se puede observar en las Figuras **Figura 112** y **Figura 113**. Por otra parte, en la **Figura 114** se puede ver la selección de una imagen de la gallería.

Hay que mencionar también, que existen tres excepciones (o dos si el canal no tiene contraseña) que son nombre vacío, descripción y contraseña vacías.

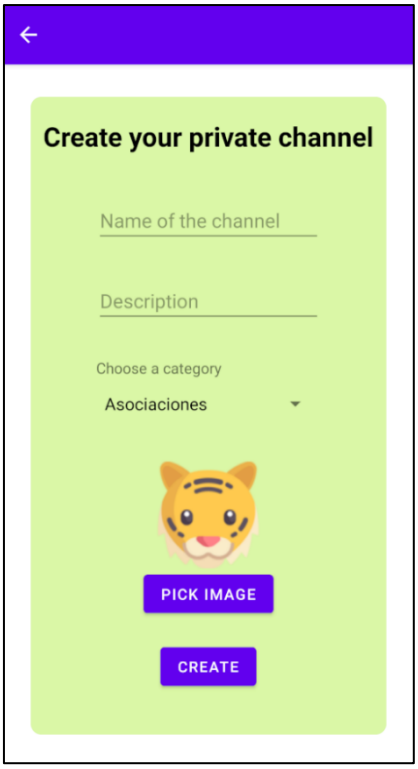


Figura 112. Pantalla CreateChannel – sin contraseña

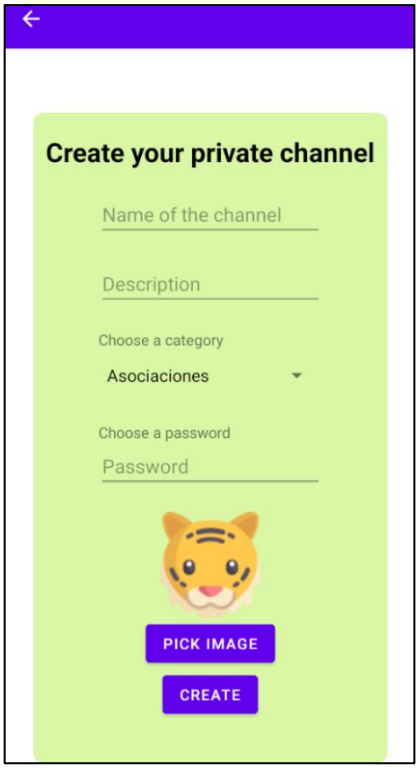


Figura 113. Pantalla CreateChannel – con contraseña

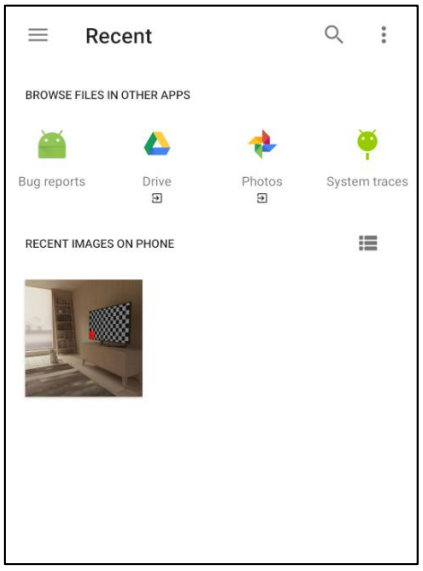
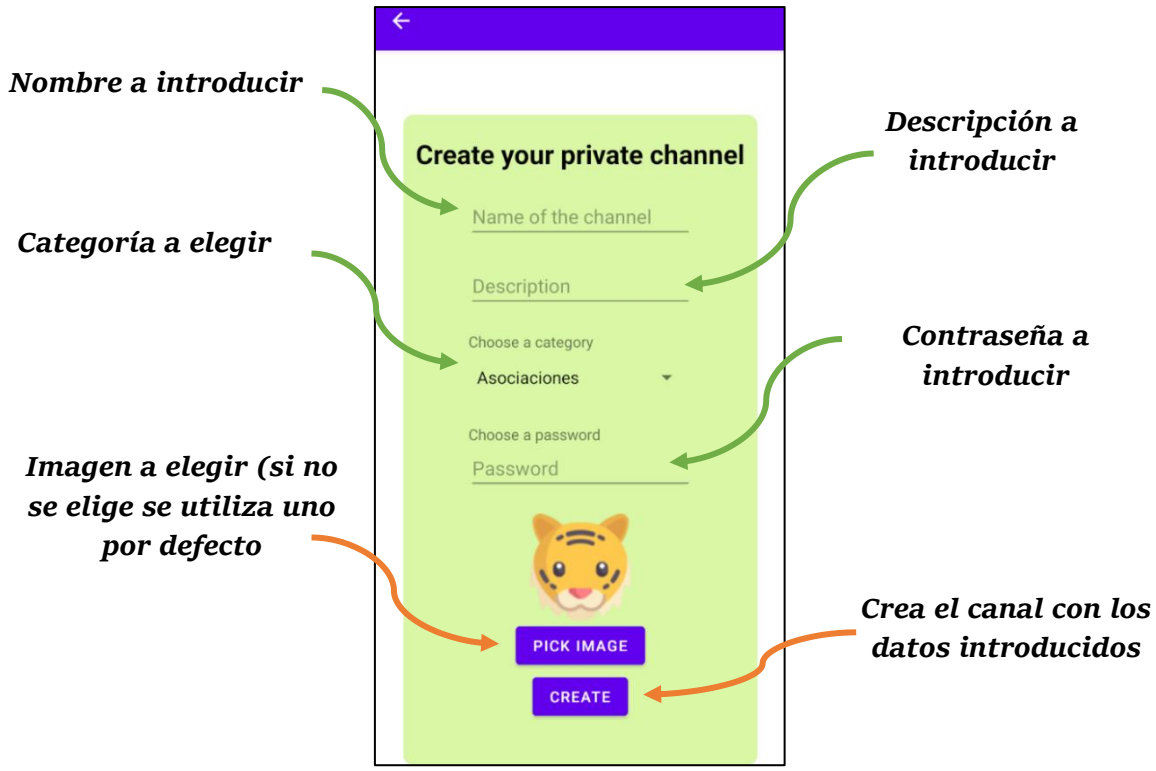


Figura 114. Seleccionado una imagen de la gallería

Si entramos en detalle de la **Figura 113** tenemos lo siguiente:



En las Figuras **Figura 115**, **Figura 116** y **Figura 117** se muestran las diferentes excepciones de esta pantalla:

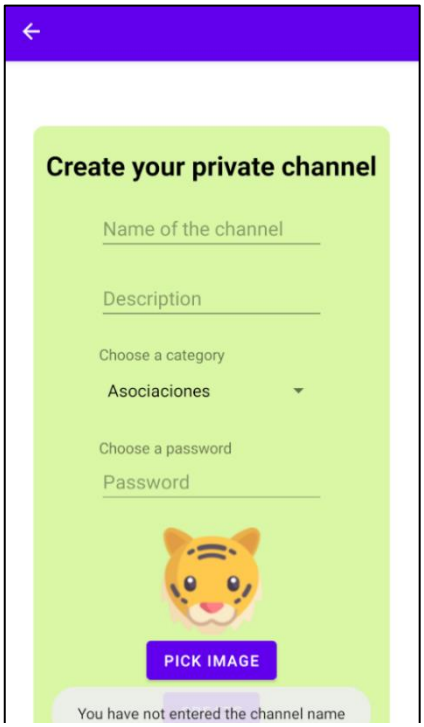


Figura 115. Pantalla CreateChannel – nombre vacío

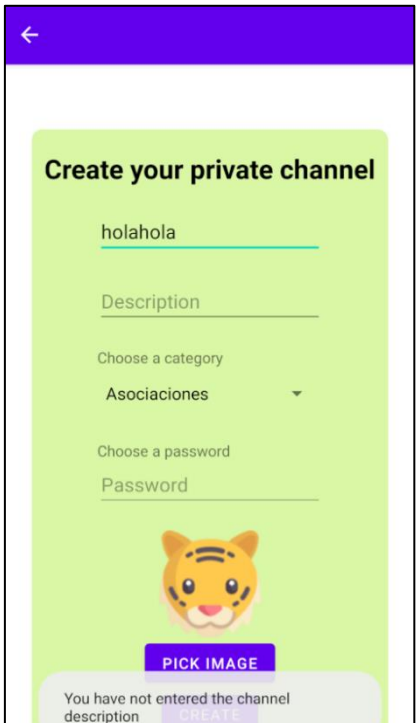


Figura 116. Pantalla CreateChannel – descripción vacía

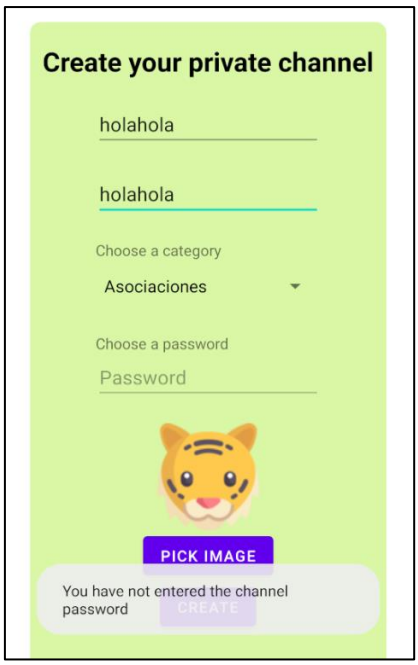


Figura 117. Pantalla CreateChannel – contraseña vacía

1.19. ChannelInformation

En la pantalla *ChannelInformation* se muestra el detalle de un canal privado, así como la fecha en la que se ha creado y la descripción del canal, tal y como se puede ver en la **Figura 118**.

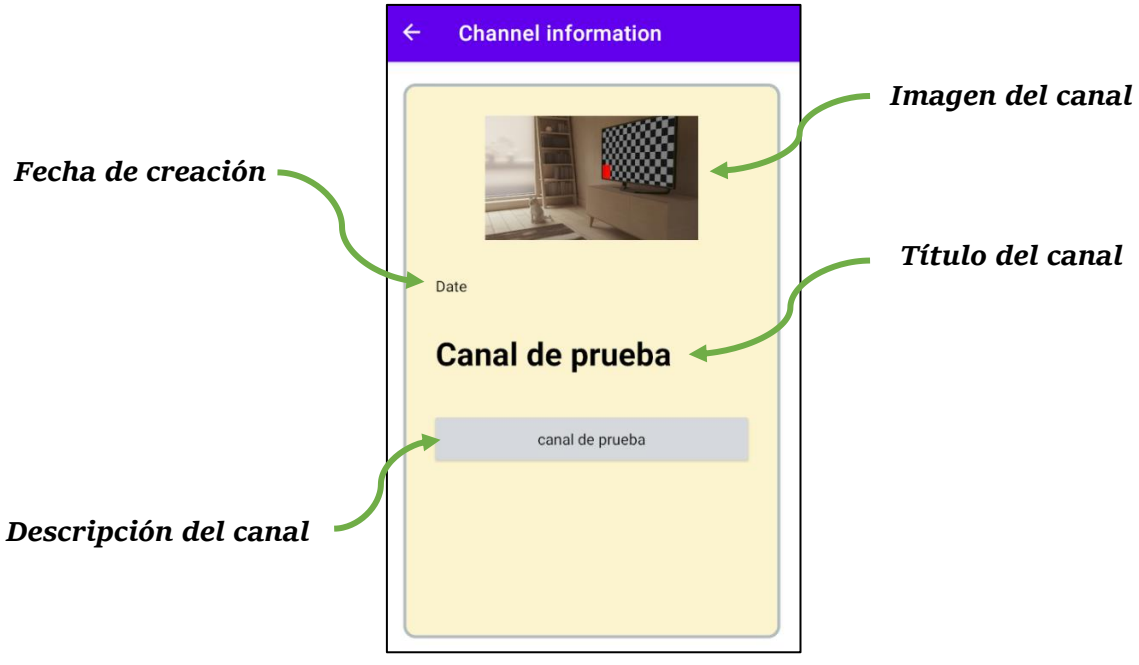


Figura 118. Pantalla ChannelInformation

1.20. Participants

Si queremos ver quiénes son los usuarios que están participando en el canal podemos acceder a la pantalla *Participants* a través de la pantalla *PrivatePosts*. Asimismo, cada usuario de la lista se identificará por su imagen de perfil y por el email. Esto se puede apreciar en la **Figura 119**.

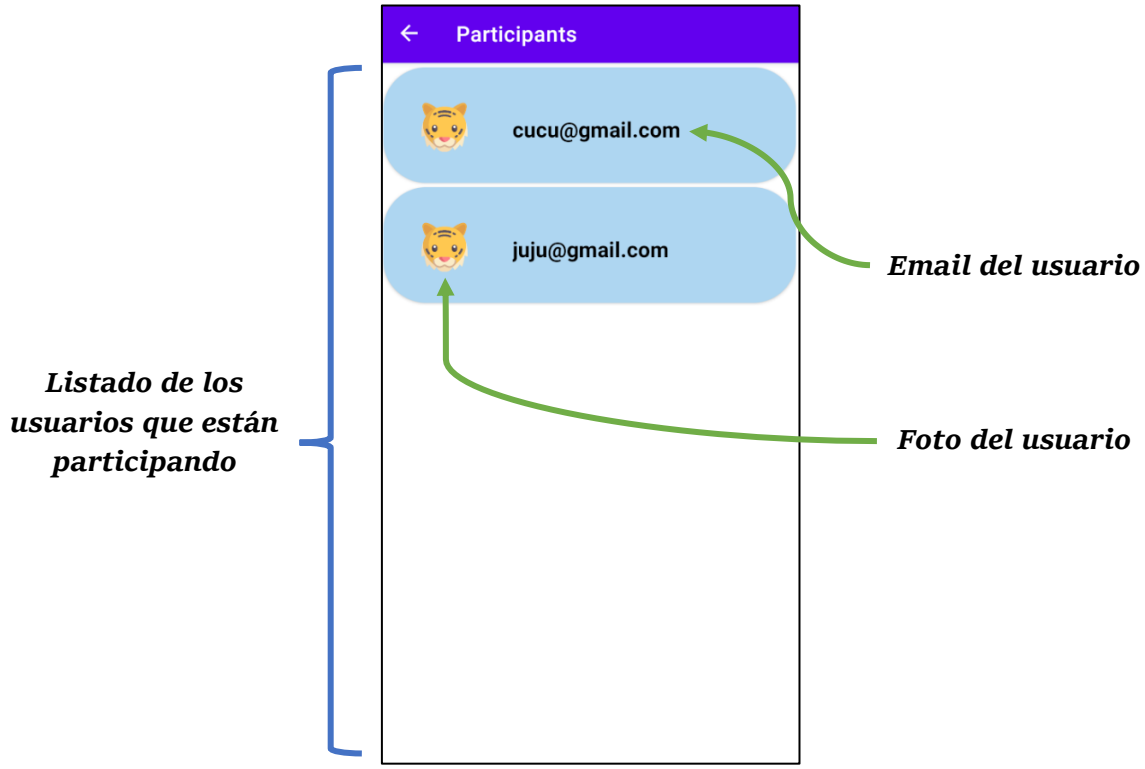


Figura 119. Pantalla Participants

1.21. Lista vacía

Si nos damos de alta ya sea como invitado o usuario registrado y nos dirigimos a la pantalla *PublicChannels*, *PublicNotifications*, *PrivateChannels* etcétera, sin suscribirnos previamente a un canal nos aparecerá una imagen con un texto indicando de que la lista está vacía y que intentemos añadir o crear (si estamos en *MyChannels*) algunos canales.

En la **Figura 120** se puede visualizar el aspecto de la lista vacía:

Figura 120. Pantalla PublicChannels – lista vacía

Capítulo VII. Conclusiones

En este capítulo se describe las conclusiones obtenidas, así como los objetivos alcanzados, además se proponen posibles líneas futuras en base al trabajo realizado.

1. Introducción

Tal y como se mencionó en el capítulo de introducción, hoy en día existe una gran cantidad de usuarios que emplean las aplicaciones de mensajería instantánea como WhatsApp, Telegram o Twitter. Sin embargo, a ciertos usuarios les parece más interesante disponer de ambas funcionalidades, es decir tanto la comunicación bidireccional como unidireccional. Por esta razón se presenta la aplicación móvil desarrollada en Android *NotifyMe* para satisfacer esta necesidad.

La aplicación *NotifyMe* presenta las siguientes funcionalidades principales:

- Parte pública.
  - Canales y subcanales públicos.
  - Sistema de suscripción.
  - Sistema de notificaciones con filtrado.
  - Almacenamiento de notificaciones.
- Parte privada
  - Canales privados.
  - Sistema de chat en grupo.
  - Sistema de suscripción.
  - Sistema de notificaciones.

La comunicación unidireccional tiene interés cuando el usuario solamente quiere recibir las notificaciones de sus canales favoritos sin la necesidad de interactuar, solamente recibe. Por otra parte, la comunicación bidireccional es útil cuando queremos comunicarnos con un grupo de personas para establecer planes, convocar reuniones, impartir información particular etc.

Para lograr este objetivo se ha empleado el gestor de contenidos WordPress para la parte pública y Firebase de Google para la parte privada, que son los dos módulos del back-end del proyecto.

2. Conclusiones

Los objetivos iniciales planteados en este proyecto fueron cinco:

- ✓ Desarrollar una aplicación móvil para Android utilizando el lenguaje de programación Java que nos permita suscribirnos a los diferentes canales públicos y recibir notificaciones con la opción de filtrado.
- ✓ Desarrollar una comunicación bidireccional donde los usuarios se puedan suscribir a los canales privados y comunicarse enviando mensajes al chat grupal.
- ✓ Emplear el gestor de contenidos WordPress para crear los canales públicos.
- ✓ Desplegar WordPress fuera del ámbito local utilizando Heroku y Amazon S3 AWS para el almacenamiento de imágenes.
- ✓ Dotar la aplicación móvil de una arquitectura MVP.

De los cuales se han cumplido todos, sin embargo, hubo que realizar ciertos cambios por ejemplo en el caso de los canales privados no era posible crearlos en WordPress dado que no era visible en el JSON devuelto por la REST API de WordPress. Es por esto por lo que se ha utilizado Firebase de Google para crear los canales privados junto con el sistema de chat en grupo. Por otra parte, tenemos las notificaciones *push* que inicialmente se implementó utilizando la plataforma de OneSignal, pero a causa de su poca manejabilidad sobre las notificaciones, se terminó utilizando Firebase Cloud Messaging.

Con la realización de este proyecto se ha aprendido a crear un sitio web de WordPress fuera del ámbito local haciendo uso de la plataforma de Heroku y de la plataforma de almacenaje en la nube Amazon S3 AWS. También se ha logrado consolidar los conceptos de categoría, tags, taxonomías y de cómo funciona la REST API de WordPress. Por otro lado, se han afianzado los conceptos relacionados con las notificaciones *push* así como

la creación, la estructura que tiene y de cómo se envía está utilizando FCM (Firebase Cloud Messaging).

Finalmente, se ha profundizado todavía más en la programación en Android, así como las peticiones HTTP para recuperar la información de la REST API, las llamadas asíncronas (ya que la información devuelta por la librería Volley o Firebase es de forma asíncrona), el almacenamiento en la base de datos Room para guardar la información localmente y el empleo de una arquitectura limpia MVP (mediador, repositorio, inyección de dependencias ...) para obtener un código limpio, bien estructurado y escalable.

3. Líneas futuras

En este apartado comentaremos el futuro del proyecto, así como las ideas y funcionalidades que no se han añadido debido a la falta de tiempo.

A continuación, se presenta un listado de las ideas:

- **Permitir al usuario elegir su imagen de perfil.** En el proyecto todos los usuarios tienen la misma imagen por defecto por lo que sería interesante poder elegir. Para ello habrá que crear una nueva pantalla donde el usuario pueda modificar su email, nombre, imagen de perfil, edad, así como la eliminación de la cuenta (derecho al olvido).

- **Permitir al usuario dirigirse a una publicación después de pulsar en la notificación.** Cuando recibimos una notificación de un canal y pulsamos sobre esa notificación, se ha programado para que se realice una función u otra dependiendo del estado de la App:

    - **Sí la App está en Foreground.** No realiza ninguna acción al pulsar sobre la notificación.
    - **Sí la App está en Background.** Pasa a *foreground* la App.
    - **Sí la App está Killed.** Se lanza nuevamente la aplicación.

- **Permitir al usuario registrase con otros medios.** En esta aplicación el usuario solamente puede registrarse introduciendo un email y repitiendo las contraseñas, sin embargo, sería más conveniente si los usuarios se pudieran registrar utilizando el número de teléfono, la cuenta de Facebook, de Apple, de Google etc. Esto es factible ya que Firebase en su apartado de autenticación lo permite como vemos en la **Figura 121**:

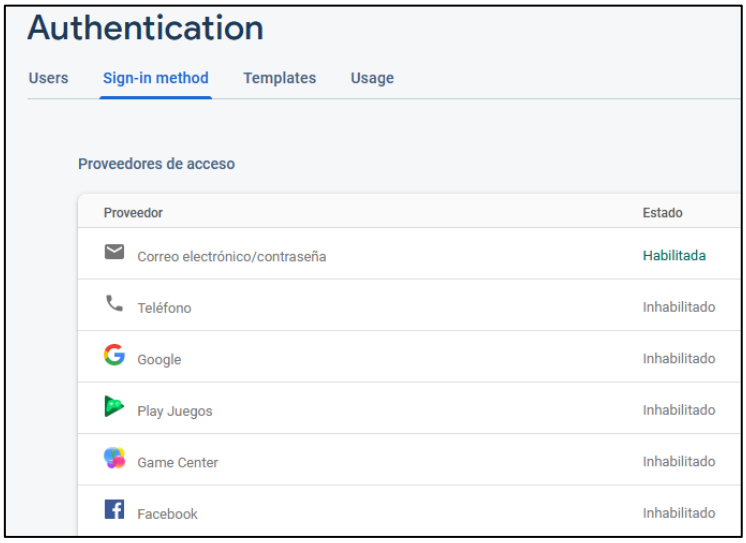


Figura 121. Firebase - Autenticación

- **Permitir al usuario valorar una publicación y filtrar las publicaciones por fecha, visita etc.** En la pantalla *SubCategoryDetail* el usuario solamente puede pulsar en una de las publicaciones para dirigirse al detalle, pero sería buena idea añadir un botón de valoración (estrellas del 0 al 5) de forma que el usuario pueda ordenar las publicaciones por valoración, por fecha de publicación o por visitas.

- **Mejorar del diseño de algunas pantallas.** Ya que en este proyecto nos hemos centrado más en la parte de la funcionalidad, en un futuro se pretende mejorar la interfaz de usuario.

Capítulo VIII. Presupuesto

En este capítulo se explica los gastos estimados generados por este proyecto basándonos de las indicaciones del COITT (Colegio Oficial de Ingenieros Técnicos de Telecomunicación). Las pautas establecidas por el COITT se desglosan en los siguientes apartados:

- Recursos materiales.
- Trabajo tarifado por tiempo empleado.
- Costes asociados a la redacción del documento.
- Derecho del visado del COITT.
- Aplicación de impuestos.

1. Recursos materiales

COITT establece que para la estimación económica de un proyecto se encuentran los recursos materiales empleados durante el diseño, desarrollo, testeo y la ejecución del proyecto, distinguiéndose en dos tipos: recursos hardware y software.

Para estimar el coste de amortización se estipula un periodo de tres años, y suponiendo un sistema de amortización lineal en el que el material inmovilizado se deprecia de forma lineal durante el periodo de tiempo evaluado. Por lo que, la expresión a utilizar es la siguiente:

$$Coste\ de\ amortización = \frac{Valor\ de\ adquisición - Valor\ residual}{Años\ de\ vida\ útil}$$

Ecuación 1. Coste de amortización

Hay que mencionar que se ha empleado alrededor de setecientas sesenta y ocho horas en un periodo de cinco meses para la realización del trabajo.

2. Recursos software

Las herramientas software empleadas para el desarrollo de este proyecto son las siguientes:

- Android Studio
- Microsoft Office.
- GitKraken.
- Figma.
- WordPress
- Firebase
- Heroku.
- Amazon S3 AWS.

Todas las herramientas software mencionadas anteriormente son gratuitas salvo Microsoft Office el cual hay que pagar por una licencia para su uso. El precio de esta licencia para un año es de 105,6 euros. Por otro lado, Firebase, Heroku y Amazon S3 AWS son sin cargo siempre y cuando no superemos el límite establecido por el plan gratuito. En conclusión, el coste total de los recursos software utilizados (coste de amortización calculado para un año) ha sido **ciento cinco con seis euros** (105,6€).

3. Recursos hardware

Entre los recursos hardware empleados para el desarrollo del proyecto se encuentran:

- Ordenador sobremesa.
- Teléfono móvil Huawei P9 Lite.
- Teléfono móvil Xiaomi Mi 10T Lite.

Recurso	**Valor de adquisición (€)**	**Valor residual (€)**	**Coste de amortización(€)**
Ordenador sobremesa	1616,59	1500	38.86

Huawei P9 Lite	329	95	78
Xiaomi Mi 10T Lite	349	299	16,7
		Total	133,56

Tabla 3. Coste de amortización de los recursos hardware

El coste total de recursos hardware utilizados en el proyecto es de **ciento treinta y tres euros con cincuenta y seis céntimos** (133,56€)**.**

4. Trabajo tarifado por tiempo empleado

En este proyecto se han empleado alrededor de 768 horas en un periodo de 5 meses para la búsqueda de información, análisis del gestor de contenidos WordPress, Firebase, Heroku y Amazon S3 AWS, así como el diseño, la implementación, el testeo de la aplicación móvil y la elaboración de la documentación.

Siguiendo las recomendaciones del COITT, el importe percibido por las horas trabajadas de un Ingeniero de Telecomunicaciones se calcula mediante la siguiente expresión:

$$H = C_t \cdot 74{,}88 \cdot H_n + C_t \cdot 96{,}72 \cdot H_e$$

Ecuación 2. Honorarios

Donde:

- H son los honorarios por tiempo.
- C_t indica un factor de corrección en función del número de horas trabajadas.
- H_n indica las horas trabajadas dentro de la jornada laboral.
- H_e indica las horas trabajadas fuera de la jornada laboral.

Según el COITT, el valor del coeficiente C_t varía en función del número de horas trabajadas de acuerdo con la siguiente tabla:

Horas empleadas	**Factor de corrección C_t**
Hasta 36 horas	1,00
36 a 72 horas	0,90
72 a 108 horas	0,80
108 a 144 horas	0,70
144 a 180 horas	0,65
180 a 360 horas	0,65
360 a 540 horas	0,55
540 a 720 horas	0,50
720 a 1080 horas	0,45
Más de 1080 horas	0,40

Tabla 4. Factor de corrección en función de las horas trabajadas

En base a la tabla se determina que el factor de corrección C_t tiene un valor igual a 0,45 horas, dado que la duración del proyecto está comprendida entre las 720 horas y 1080 horas. Cabe mencionar que no se ha trabajado fuera del horario laboral en la mayor parte de los días, por lo que supondremos unas 20 horas aproximadamente.

Si aplicamos este factor de corrección a la ecuación mencionada anteriormente tenemos:

$$H = 0{,}45 \cdot 74{,}88 \cdot 768 + \ 0{,}45 \cdot 96{,}72 \cdot 20 = 26.749{,}008\ €$$

Ecuación 3. Estimación de los honorarios

Por tanto, la tarifa por tiempo de empleado es de **veintiséis mil setecientos cuarentainueve euros** (26.749,008 €)**.**

5. Costes asociados a la redacción del documento

El importe de la redacción del proyecto se calcula de acuerdo con la siguiente expresión:

$$R = 0{,}07 \cdot P \cdot C_n$$

Ecuación 4. Coste de la redacción del documento

Donde:

- P indica el presupuesto del documento.
- C_n indica el coeficiente de ponderación en función del presupuesto.

Según el COITT C_n está determinado por el presupuesto del proyecto, recordemos que hasta el momento tenemos un total de (26.988,168 €) que es la suma del trabajo tarifado por tiempo empleado, la amortización del hardware y la amortización del software. COITT establece que para un presupuesto inferior a los 30.050 €, el coeficiente de ponderación tiene un valor igual a 1, y si sustituimos los valores en la ecuación anterior se obtiene lo siguiente:

$$R = 0{,}07 \cdot P \cdot C_n = 0{,}07 \cdot 26.988{,}168 \cdot 1 = 1.889{,}17\ €$$

Ecuación 5. Estimación del coste de redacción del documento

Entonces, los costes de redacción del proyecto son **mil ochocientos ochenta y nueve con diecisiete euros**.

6. Derecho del visado del COITT

Los gastos de visado del COITT se calculan con la siguiente expresión:

$$V = 0{,}006 \cdot P \cdot C_v$$

Ecuación 6. Gastos de visado COITT

Donde:

- P indica el presupuesto del proyecto.
- C_v es el coeficiente reductor en función del presupuesto del trabajo.

Como nuestro proyecto no dispone de gastos asociados a los materiales fungibles, el presupuesto hasta el momento será la suma de los recursos hardware, trabajo tarifado por tiempo empleado y costes asociados a la redacción del documento, por lo que el presupuesto actual es:

$$P = 26.988{,}168 + 1.889{,}17 = 28.877{,}338\ €$$

Ecuación 7. Estimación del presupuesto hardware, tiempo empleado y redacción

Al igual que antes como el presupuesto no supera los 30.050 € el coeficiente de ponderación tendrá un valor igual 1, por lo tanto, el coste de los derechos de visado del trabajo es de:

$$V = 0{,}006 \cdot P \cdot C_v = 0{,}006 \cdot 28.877{,}338 \cdot 1 = 173{,}26\ €$$

Ecuación 8. Estimación del coste de visado COITT

Luego, los derechos de visado son **ciento setenta y tres y con veintiséis euros** (173,26 €).

7. Gastos de tramitación y envío

Los gastos de tramitación y envío se estipulan en seis euros y un céntimo (6,01 €).

8. Aplicación de impuestos

El coste total del proyecto hasta el momento es de 28.942,94 €. A esta cantidad hay que sumarle los impuestos y como estamos en canarias hay añadirle el 7% del IGIC (Impuesto General Indirecto Canario).

Recurso	**Coste (€)**
Recursos materiales	133,56
Recursos software	105,6
Trabajo tarifado por tiempo empleado	26.749,01
Costes asociados a la redacción del documento	1.881,78
Derechos de visado del COITT	172,58
Gastos de tramitación y envío	6,01
Subtotal	29.048,54
Aplicación de impuestos (IGIC 7%)	2.033,4
Total	31.081,94

Tabla 5. Costes totales del proyecto

Por lo tanto, el presupuesto total de este Trabajo de Fin de Grado asciende a **treinta y un mil ochenta y uno con noventa y cuatro euros** (31.081,94).

Las Palmas de Gran Canaria a 22 de julio de 2021

Firma:

Cunwang Guo

Capítulo IX. Bibliografía

[1] «Konvoko,» [En línea]. Disponible en: https://www.konvoko.com/. [Último acceso: 2 feb 2021].

[2] «WordPress,» [En línea]. Disponible en: https://wordpress.com/. [Último acceso: 30 ene 2021].

[3] «Firebase,» [En línea]. Disponible en: https://firebase.google.com/. [Último acceso: 19 jul 2021].

[4] «Dinantia,» [En línea]. Disponible en: https://www.dinantia.com/es/. [Último acceso: 19 jul 2021].

[5] «ECYL,» [En línea]. Disponible en: http://empleo.jcyl.es/web/jcyl/Empleo/es/Plantilla66y33/1284383527269/_/_/_. [Último acceso: 19 jul 2021].

[6] «Heroku,» [En línea]. Disponible en: https://www.heroku.com/. [Último acceso: 17 jul 2021].

[7] «Amazon S3,» [En línea]. Disponible en: https://aws.amazon.com/s3/. [Último acceso: 17 jul 2021].

[8] «Android Studio,» [En línea]. Disponible en: https://developer.android.com/studio. [Último acceso: 1 feb 2021].

[9] «IntelliJ IDEA,» [En línea]. Disponible en: https://www.jetbrains.com/idea/. [Último acceso: 19 jul 2021].

[10] «Essential tools for software developers and teams,» [En línea]. Disponible en: https://www.jetbrains.com/. [Último acceso: 19 jul 2021].

[11] «Java Programming Language,» [En línea]. Disponible en: https://docs.oracle.com/javase/7/docs/technotes/guides/language/. [Último acceso: 19 jul 2021].

[12] «GitKraken,» [En línea]. Disponible en: https://www.gitkraken.com/. [Último acceso: 1 feb 2021].

[13] «GitHub,» [En línea]. Disponible en: https://github.com/. [Último acceso: 19 jul 2021].

[14] «Learn Git: Commit,» [En línea]. Disponible en: https://www.gitkraken.com/learn/git/commit. [Último acceso: 19 jul 2021].

[15] «Figma,» [En línea]. Disponible en: https://www.figma.com/. [Último acceso: 1 feb 2021].

[16] «Trello,» [En línea]. Disponible en: https://trello.com/en. [Último acceso: 19 jul 2021].

[17] «POSTMAN,» [En línea]. Disponible en: https://www.postman.com/. [Último acceso: 19 jul 2021].

[18] «REST API Handbook,» [En línea]. Disponible en: https://developer.wordpress.org/rest-api/. [Último acceso: 17 jul 2021].

[19] «Firebase Cloud Messaging,» [En línea]. Disponible en: https://firebase.google.com/docs/cloud-messaging. [Último acceso: 19 jul 2021].

[20] «REST API Developer Endpoint Reference,» [En línea]. Disponible en: https://developer.wordpress.org/rest-api/reference/. [Último acceso: 14 jul 2021].

[21] «Pagination,» [En línea]. Disponible en: https://developer.wordpress.org/themes/functionality/pagination/. [Último acceso: 19 jul 2021].

[22] «OneSignal,» [En línea]. Disponible en: https://onesignal.com/. [Último acceso: 19 jul 2021].

[23] «Save data in a local database using Room,» [En línea]. Disponible en: https://developer.android.com/training/data-storage/room. [Último acceso: 19 jul 2021].

[24] «MySQL,» [En línea]. Disponible en: https://www.mysql.com/. [Último acceso: 19 jul 2021].

[25] «Database Description,» [En línea]. Disponible en: https://codex.wordpress.org/Database_Description. [Último acceso: 16 jul 2021].

[26] «Firebase Realtime Database,» [En línea]. Disponible en: https://firebase.google.com/docs/database. [Último acceso: 19 jul 2021].

[27] «What is NoSQL?,» [En línea]. Disponible en: https://www.mongodb.com/nosql-explained. [Último acceso: 19 jul 2021].

[28] «Better REST API Featured Images,» [En línea]. Disponible en: https://wordpress.org/plugins/better-rest-api-featured-images/. [Último acceso: 16 jul 2021].

[29] «Amazon S3 bucket,» [En línea]. Disponible en: https://searchaws.techtarget.com/definition/AWS-bucket. [Último acceso: 19 jul 2021].

[30] «Easy Add Thumbnail,» [En línea]. Disponible en: https://wordpress.org/plugins/easy-add-thumbnail/. [Último acceso: 16 jul 2021].

[31] «WP Offload Media Lite for Amazon S3, DigitalOcean Spaces, and Google Cloud Storage,» [En línea]. Disponible en: https://wordpress.org/plugins/amazon-s3-and-cloudfront/. [Último acceso: 16 jul 2021].

[32] «Pages with category and tag,» [En línea]. Disponible en: https://wordpress.org/plugins/pages-with-category-and-tag/. [Último acceso: 16 jul 2021].

[33] «WordPress Shortcodes Plugin — Shortcodes Ultimate,» [En línea]. Disponible en: https://wordpress.org/plugins/shortcodes-ultimate/. [Último acceso: 16 jul 2021].

[34] «Save data using SQLite,» [En línea]. Disponible en: https://developer.android.com/training/data-storage/sqlite. [Último acceso: 19 jul 2021].

[35] «Accessing data using Room DAOs,» [En línea]. Disponible en: https://developer.android.com/training/data-storage/room/accessing-data. [Último acceso: 19 jul 2021].

[36] «Defining data using Room entities,» [En línea]. Disponible en: https://developer.android.com/training/data-storage/room/defining-data. [Último acceso: 19 jul 2021].

[37] «simpleUMLCE,» [En línea]. Disponible en: https://plugins.jetbrains.com/plugin/4946-simpleumlce. [Último acceso: 19 jul 2021].

[38] «FloatingActionButton,» [En línea]. Disponible en: https://github.com/zendesk/android-floating-action-button. [Último acceso: 19 jul 2021].

[39] «Receive messages in an Android app,» [En línea]. Disponible en: https://firebase.google.com/docs/cloud-messaging/android/receive. [Último acceso: 19 jul 2021].

[40] «Easy Splash Screen Android,» [En línea]. Disponible en: https://github.com/pantrif/EasySplashScreen. [Último acceso: 19 jul 2021].

[41] «Volley overview,» [En línea]. Disponible en: https://developer.android.com/training/volley. [Último acceso: 19 jul 2021].

[42] «JSONArray,» [En línea]. Disponible en: https://developer.android.com/reference/org/json/JSONArray. [Último acceso: 19 jul 2021].

[43] «JSONObject,» [En línea]. Disponible en: https://developer.android.com/reference/org/json/JSONObject. [Último acceso: 19 jul 2021].

[44] «Callback,» [En línea]. Disponible en: https://developer.android.com/reference/javax/security/auth/callback/Callback. [Último acceso: 19 jul 2021].

[45] «Firebase: Realtime Database Reading and Writing,» [En línea]. Disponible en: https://www.javatpoint.com/firebase-realtime-database-reading-and-writing. [Último acceso: 19 jul 2021].

[46] «Espresso,» [En línea]. Disponible en: https://developer.android.com/training/testing/espresso. [Último acceso: 21 jul 2021].

[47] «Barista,» [En línea]. Disponible en: https://github.com/AdevintaSpain/Barista. [Último acceso: 21 jul 2021].

[48] «UI Automator,» [En línea]. Disponible en: https://developer.android.com/training/testing/ui-automator. [Último acceso: 21 jul 2021].

[49] «Espresso & UIAutomator - the perfect tandem,» [En línea]. Disponible en: https://testyour.app/blog/espresso-uiautomator. [Último acceso: 21 jul 2021].

[50] dannyroa, «Android Espresso Samples,» [En línea]. Disponible en: https://github.com/dannyroa/espresso-samples/tree/master/RecyclerView/app/src/androidTest/java/com/dannyroa/espresso_samples/recyclerview. [Último acceso: 21 jul 2021].

[51] R. Boyer, Android 9 Development CookBook, 3ª ed., Packt, oct. 2018 [En línea]. Disponible en: https://www.packtpub.com/product/android-9-development-cookbook-third-edition/9781788991216

[52] D. Griffiths y D. Griffiths, *Head First Android Development*, 3a ed., O'Reilly, dic. 2021 [En línea]. Disponible en: https://www.oreilly.com/library/view/head-first-android/9781492076513/

[53] «Complete Guide To Hosting Wordpress On Heroku With SSL Certification,» [En línea]. Disponible en: https://dev.to/aryaziai/complete-guide-to-hosting-wordpress-on-heroku-with-ssl-certification-4f2l. [Último acceso: 22 jul 2021].

[54] «StackOverFlow,» [En línea]. Disponible en: https://stackoverflow.com/. [Último acceso: 22 jul 2021].

Capítulo X. Anexos

1. Despliegue de WordPress

En este apartado se explicará cómo se ha instalado y configurado WordPress fuera del ámbito local utilizando la plataforma PaaS Heroku y Amazon S3 AWS para almacenar las imagenes. Asimismo, se comentará la REST API de WordPress de la que se extraen los canales públicos para mostrarlos en la aplicación móvil.

1.1. Heroku

El primer paso es obviamente registrarse en Heroku para poder hacer uso de las herramientas que proporciona. Una vez registrado nos dirigiremos al repositorio de PhilppHeuer que contiene un *template* para instalar y correr WordPress en Heroku. Se trata un proyecto que está basado en Bedrock desarrollado por Roots que nos permite repensar la manera en la que desarrollamos los sitios Webs y de poder llegar a utilizar mejores prácticas de desarrollo, es decir se refiere a un WordPress *stack* moderno que nos ayuda a empezar con las mejores herramientas de desarrollo y una estructura de proyecto moderna.

En este repositorio nos dirigiremos a la sección *Getting Started* y pulsamos en el botón *Deploy to Heroku* el cual nos redirigirá a la página de Heroku. En ella introduciremos el nombre la aplicación, la región (Europa) y le damos a *deploy app* para que se configure y se construya el proyecto en Heroku.

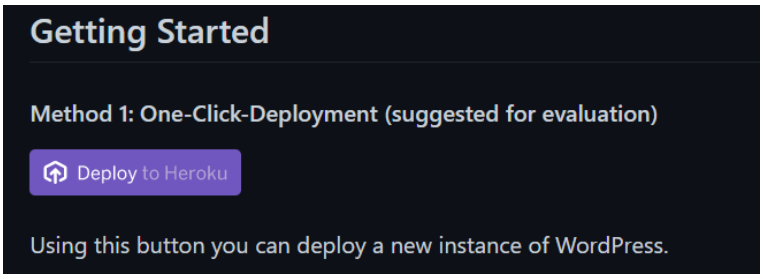


Figura 122. Github PhilippHeuer repository

Una vez que se haya configurado nos dirigiremos al dashboard de Heroku, donde podremos ver nuestro proyecto desplegado si no ha habido ningún tipo de problema. Después de esto, en la sección opciones o *settings* copiamos la URL git que se empleará para añadir todos los cambios al proyecto WordPress utilizando la consola cmd, el cual será necesario disponer de la CLI de Heroku.

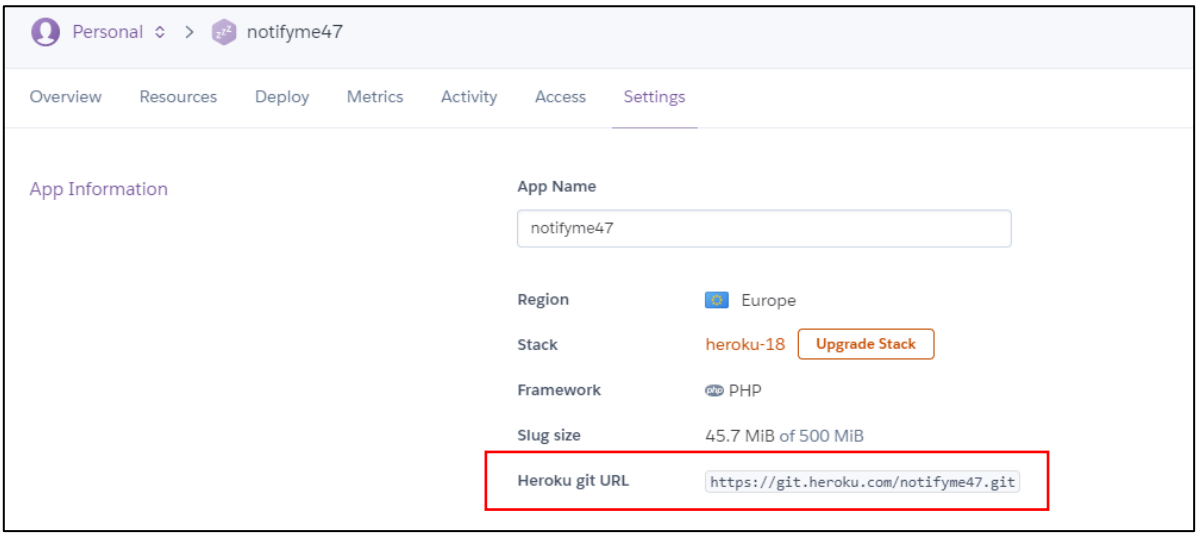


Figura 123. Heroku - Settings

El siguiente paso es clonar el proyecto de WordPress de PhilppHeuer en nuestro ordenador en un directorio adecuado como podría ser *Local Disk (C:)* → *Users* → *nombre del usuario* → *directorio*. En este punto abrimos la consola cmd para acceder a la CLI de Heroku introduciendo *heroku login*. Una vez que nos hayamos dado de alta realizaremos un *commit* del proyecto de PhilppHeuer a Heroku.

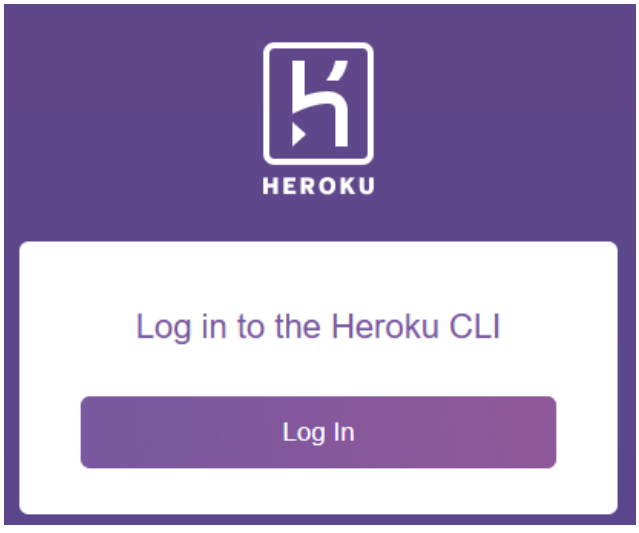


Figura 124. Heroku CLI Log in

1.2. Amazon S3 AWS

Al terminar de configurar el proyecto en Heroku es necesario crear el bucket en Amazon S3 AWS para guardar las imágenes que empleemos en WordPress. Para ello, acedemos a la página oficial de Amazon S3 AWS y lo primero que habrá que realizar es crearse una cuenta de Amazon (será necesario disponer de una tarjeta de crédito).

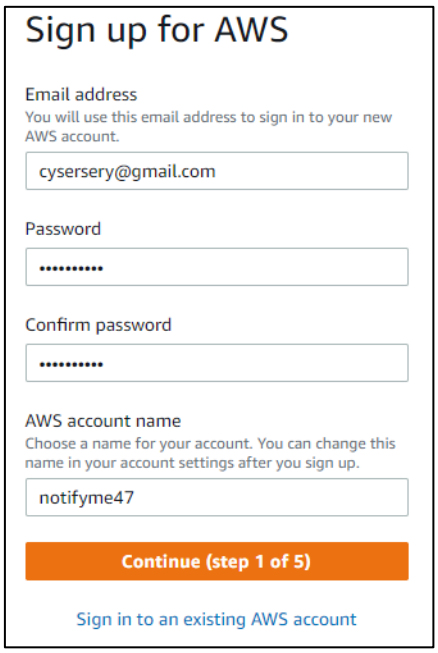


Figura 125. Amazon S3 AWS - Registro

Una vez registrado nos preguntará el tipo de plan, en nuestro elegiremos la versión gratuita, ya que nos proporciona un servicio que es suficiente para nuestro proyecto.

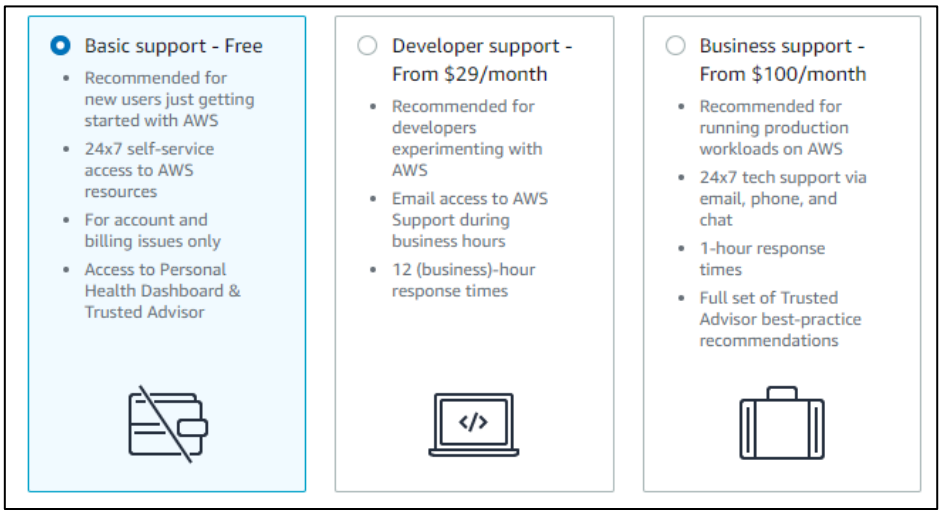


Figura 126. Amazon S3 AWS – Tipo de plan

Hecho esto podremos darnos de alta en la pantalla *sign in* como usuario root. Tras acceder escribiremos en el buscador *buckets* y elegimos la opción S3.

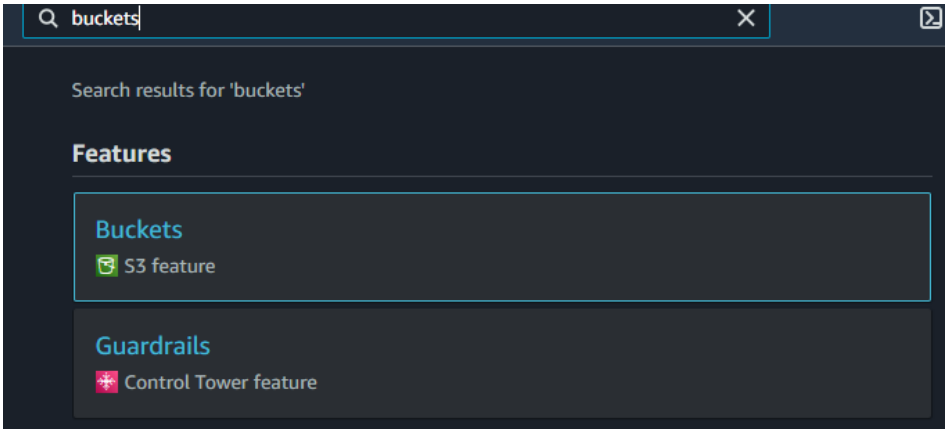


Figura 127. Amazon S3 AWS – Buscador de servicios

Para crear un bucket simplemente pulsamos en *Create bucket:*

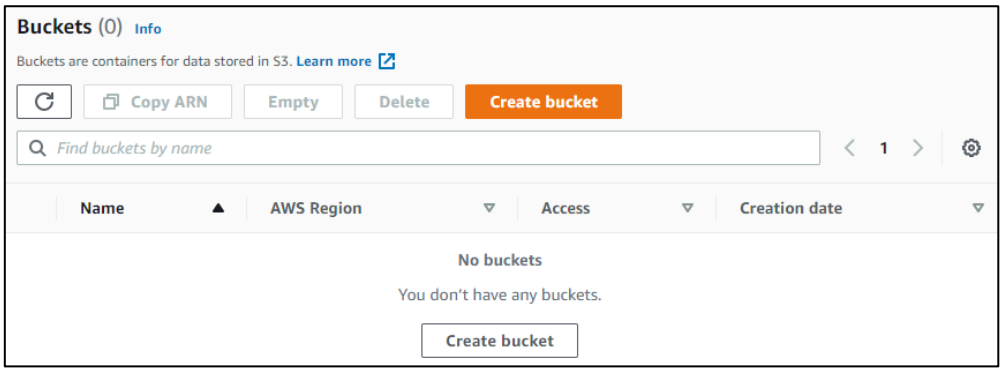


Figura 128. Amazon S3 AWS – Inicio

Introducimos el nombre del bucket y seleccionamos una región europea:

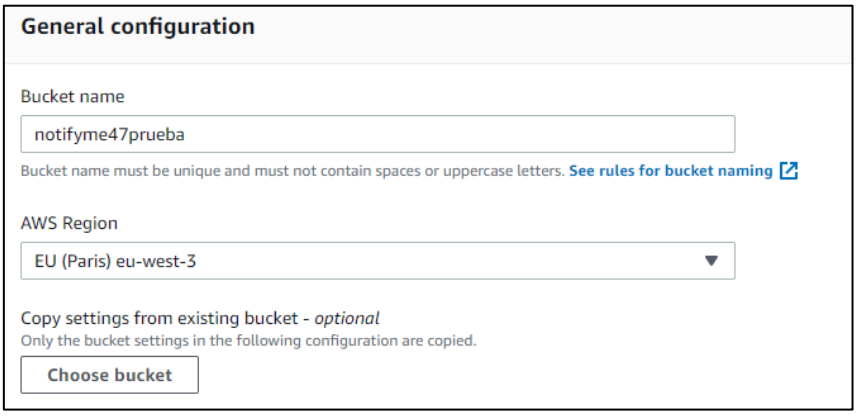


Figura 129. Amazon S3 AWS – Creando un bucket

1.3. WordPress

Ahora que tenemos desplegado correctamente el proyecto en Heroku y creado el bucket en Amazon S3 AWS, tenemos que añadir los plugins esenciales y configurarlos. Para ello, lo primero que tenemos que realizar es localizar el proyecto de WordPress en nuestro ordenador *Local Disk (C:)* → *Users* → *nombre del usuario* → *directorio*.

Una vez localizado el proyecto lo abrimos y accedemos a la carpeta de los plugins *web* → *app* → *plugins*. En este punto añadimos todos los plugins que nosotros queramos y para guardar los cambios para que se refleje en la página de WordPress debemos hacerlo a través de los *commits*.

Para subir los cambios a Heroku abrimos la consola cmd, localizamos el proyecto utilizando el comando *cd* e introducimos la orden *heroku login* (explicada en el punto 1.1 de este capítulo). Una vez autenticado introducimos las siguientes ordenes:

```
$ git add .
$ git commit -am “Una descripción del plugin o de los cambios”
$ git push heroku master
```

En el caso del plugin WP Offload Media debemos de modificar el fichero *wp-config* localizado en *web*. En ella debemos proporcionar el proveedor, la clave de acceso y la clave secreta:

```
require_once(dirname( path: __DIR__) . '/vendor/autoload.php');
require_once(dirname( path: __DIR__) . '/config/application.php');

define( 'AS3CF_SETTINGS', serialize( array(
    'provider' => 'aws',
    'access-key-id' => '',
    'secret-access-key' => '',
) ) );
```

Figura 130. Fichero wp-config

2. Pliego de condiciones

En este apartado especifican las condiciones bajo las que se ha desarrollado este Proyecto Fin de Carrera. Se detallará los requisitos hardware y software, así como las características y versiones.

2.1. Requerimientos software

Las herramientas software utilizadas son las siguientes:

- Microsoft Windows 10 Pro.
- Android Studio 4.1.1.
- Microsoft Office 2019.
- GitKraken 7.7.0.
- Postman 8.8.0.
- Figma.
- WordPress 4.9.2.
- Firebase (FCM, autenticación, Realtime Database).
- Heroku.
- Amazon S3 AWS.

2.2. Requerimientos hardware

El ordenador sobremesa utilizado para el desarrollo de este Trabajo de Fin de Grado presenta las siguientes características:

Procesador	Intel Core i7-10700K 3.80GHz
Tarjeta gráfica	Sapphire AMD Radeon RX 6800 16GB GDDR6
Placa base	MSI MAG Z490 TOMAHAWK

Memoria RAM	Kingston HyperX Fury Black DDR4 3200Mhz PC-25600 32GB 4x8GB CL16
Disco SSD	Samsung SSD 850 EVO 500 GB
Disco NVMe SSD	Samsung SSD M.2 NVMe 970 EVO Plus 500 GB
Fuente de alimentación	Corsair RM750W 80 Plus Gold Full Modular

Tabla 6. Características del ordenador sobremesa

En cuanto a los teléfonos móviles se han empleado los siguientes modelos:

	Huawei P9 Lite Gold	**Xiaomi Mi 10T**
Resolución	1080 x 1920 px FHD	1080 x 2400 px FHD+
Pulgadas	5.2”	6.67”
Tipo de pantalla	LCD IPS	LCD IPS
Procesador	Huawei HiSilicon KIRIN 955	Qualcomm Snapdragon 750G
Memoria RAM	3 GB	6 GB LPDDR4X
Almacenamiento	64 GB	128 GB

Tabla 7. Características de los dispositivos móviles

3. Detalle de las clases implementadas

En este apartado comentaremos en detalle los campos y métodos de las clases generales y de las *Activities*.

3.1. Clases generales

A continuación, explicaremos cada uno de los campos y métodos de las clases generales.

3.1.1. ChatClass

Campos

---

dateHour

private String dateHour

Variable String que define la fecha en la que se envió el mensaje.

message

private String message

Variable String que define el mensaje que se envía el usuario.

name

private String name

Variable String que define el email del usuario.

Métodos

Getters y setters asociados a cada uno de los campos anteriores.

3.1.2. FetchCategorias

Campos

---

buttonText

private String buttonText

Variable String que define el texto del botón.

category

private String category

Variable String que define el nombre de un canal o subcanal.

categoryReal

private String categoryReal

Variable String que define la categoría (educación, deporte...)

content

private String content

Variable String que define el contenido de un subcanal.

date

private String date

Variable String que define la fecha en la que se publicó la noticia.

imgName

private String imgName

Variable String que define el nombre de la imagen.

isPrivate

private boolean isPrivate

Variable boolean empleado para distinguir entre canales públicos y privados.

key

private String key

Variable String que define la referencia de un canal de Firebase.

parent

private String parent

Variable String que define el id de una categoría padre.

subcategory

private String subcategory

Variable String que define el subtítulo de una categoría.

Métodos

Getters y setters asociados a cada uno de los campos anteriores.

3.1.3. FetchFirebaseData

Campos

---

author

private String author

Variable String que define el autor del canal.

category

private String category

Variable String que define la categoría del canal.

checked

private boolean checked

Variable boolean que define el estado del checkbox en Firebase.

content

private String content

Variable String que define el contenido del canal.

date

private String date

Variable String que define la fecha en la que se creó el canal.

hasPassword

private String hasPassword

Variable boolean que define si tiene contraseña o no el canal.

imgName

private String imgName

Variable String que define el nombre de la imagen del canal.

keyValue

private String keyValue

Variable String que define la referencia de un canal.

name

private String name

Variable String que define el nombre del canal.

password

private String password

Variable String que define la contraseña del canal (solo para aquellos canales con contraseña).

Métodos

Getters y setters asociados a cada uno de los campos anteriores.

3.1.4. FetchPages

Campos

tagId

private String tagId

Variable String que define los tags de una página.

pageName

private String pageName

Variable String que define el nombre de la página.

Métodos

Getters y setters asociados a cada uno de los campos anteriores.

3.1.5. FetchTags

Campos

tagName

private String tagName

Variable String que define el nombre de un tag.

tagId

private String tagId

Variable String que define el id de un tag.

Métodos

Getters y setters asociados a cada uno de los campos anteriores.

3.1.6. MyFirebaseMessagingService

Campos

data

private Map<String, String> data

Objeto que almacena datos en formato clave valor.

GROUP_KEY_WORK_EMAIL

private String GROUP_KEY_WORK_EMAIL

String que define la clave empleada para las notificaciones en grupo. De esta forma si recibimos múltiples notificaciones estas se mostrarán en formato pila.

repositoryContract

private RepositoryContract repositoryContract

Objeto que define el repositorio del proyecto.

subNotificationsHistoryItemList

private List<SubNotificationsHistoryItem> subNotificationsHistoryItemList

Variable que almacena una lista de objetos SubNotificationsHistoryItem. Se emplea para guardar las notificaciones que recibamos.

TAG

private static String TAG

Variable String que define el nombre de la clase. Se emplea en los Logs de Android.

Métodos

---

fetchDataFromDataBase

public void fetchDataFromDataBase ()

Este método se encarga de comprobar las configuraciones que tiene el usuario de los canales y de las notificaciones en la base de datos, es decir si el usuario está suscrito al canal y tiene habilitada las notificaciones entonces llamará al método *getNotification*.

fetchFirebaseDataListNotification

public void fetchFirebaseDataListNotification ()

Este método sigue la misma lógica que el método anterior salvo que ahora se encargará de los canales de Firebase.

fetchSubCategoryListDataByUser

public void fetchSubCategoryListDataByUser (RepositoryContract.GetSubCategoryListCallBack callback)

Recupera de la base de datos el listado de los canales en los que está suscrito el usuario. Se utiliza en los métodos *fetchDataFromDataBase* y *fetchFirebaseDataListNotification* para comprobar si mostrar una notificación push.

fetchSubNotificationsListData

public void fetchSubNotificationsListData (RepositoryContract.GetSubNotificationsListCallBack callback)

Recupera de la base de datos el listado de los tags de un canal. Se utiliza en el método *fetchDataFromDataBase* para comprobar si mostrar una notificación push. Se utiliza para filtrar las notificaciones.

getNotification

private void getNotification (Map<String, String> data)

Este método se encarga de generar la notificación push y de llamar al método *saveNotifications* para guardar la notificación en la base de datos.

insertSubNotificationsHistoryToDataBase

public void insertSubNotificationsHistoryToDataBase (List<SubNotificationsHistoryItem> subNotificationsHistoryItemList)

Guarda una lista de objetos SubNotificationsHistoryItem en la base de datos, en otras palabras, guarda las notificaciones que recibamos en la base de datos.

onMessageReceived

public void onMessageReceived (RemoteMessage remoteMessage)

Este método se encarga de recibir las notificaciones sin importar en que ciclo se encuentre la aplicación (destruida, en background, en foreground ...). Se encarga de la lógica, es decir distingue notificaciones privadas de notificaciones públicas, actualiza las preferencias etc.

saveNotifications

private void saveNotifications (Map<String, String> data)

Se encarga de crear un objeto SubNotificationsHistoryItem a partir de los datos del parámetro data. Una vez creado el objeto llama al método *insertSubNotificationsHistoryToDataBase* para insertar el ítem en la base de datos.

3.1.7. NotifyMeRepository

Campos

DB_FILE

public static final String DB_FILE

Variable String que define el nombre del archivo de la base de datos.

SHARED_PREF

public static final String SHARED_PREF

Variable String que se emplea en la función SharedPreferences para guardar las preferencias.

EMAIL

public static final String EMAIL

Variable String que se emplea en la función SharedPreferences para guardar en las preferencias el valor del email. Este valor se actualiza cada vez que se realice login desde la pantalla Login.

ACTIVE

public static final String ACTIVE

Variable String que almacena ***"yes"*** o ***"no"***, y se emplea en la función SharedPreferences para comprobar si el usuario está en la pantalla PrivatePosts. Esto se detalla más en el punto 3.2.2.16 de este capítulo.

spinnerList

private final List<String> spinnerList

Almacena una lista de Strings. Se emplea para guardar las categorías principales y posteriormente mostrarlos en un spinner.

categoryList

private final List<String> categoryList

Almacena una lista de Strings. Se emplea en los bucles *for* para recorrer una tabla de la base de datos de Firebase.

fetchFirebaseDataList

private final List<FetchFirebaseData> fetchFirebaseDataList

Almacena una lista de objetos FetchFirebaseData. Se emplea para recuperar las categorías de Firebase.

fetchAllFirebaseDataList

private final List<FetchFirebaseData> fetchAllFirebaseDataList

Almacena una lista de objetos FetchFirebaseData. Se emplea para recuperar todos los canales de Firebase.

fetchFirebaseUserList

private final List<FetchFirebaseData> fetchFirebaseUserList

Almacena una lista de objetos FetchFirebaseData. Se emplea para recuperar todos los usuarios de un canal de Firebase.

chatClassList

private final List<ChatClass> chatClassList

Almacena una lista de objetos ChatClass. Se emplea para recuperar de un canal de Firebase el listado de los mensajes del chat.

subcategoryDetailItem

private final SubcategoryDetailItem subcategoryDetailItem

Almacena un objeto SubcategoryDetailItem.

subNotificationsItem

private final SubNotificationsItem subNotificationsItem

Almacena un objeto SubNotificationsItem.

subNotificationsHistoryItem

private final SubNotificationsHistoryItem subNotificationsHistoryItem

Almacena un objeto SubNotificationsHistoryItem.

email

private String email

Variable que se emplea para almacenar el email del usuario.

firebaseDataBase

private final FirebaseDataBase firebaseDataBase

Objeto que define el punto de entrada para acceder a la base de datos de Firebase.

mDataBase

private DatabaseReference mDataBase

Objeto que representa una referencia a una ubicación particular de la base de datos de Firebase.

mFirebaseStorage

private final FirebaseStorage mFirebaseStorage

Objeto que define el servicio de carga y descarga de objetos en Google Cloud Storage.

mStorage

private final StorageReference mStorage

Objeto que representa una referencia a Google Cloud Storage object.

INSTANCE

private static NotifyMeRepository INSTANCE

Define la instancia de la clase.

database

private final NotifyMeDataBase database

Define la base de datos Room.

context

private final Context context

Define el contexto de la clase.

Constructores

NotifyMeRepository

private NotifyMeRepository (Context context)

Constructor que inicializa las diferentes listas, los objetos y construye la base de datos Room.

Métodos

getInstance

public static RepositoryContract getInstance (Context context)

Método que se encarga de crear el repositorio solo si no es nulo, es decir se crea una vez solo. De esta forma todos los modelos del proyecto emplean el mismo repositorio.

getSubCategoryDao

private SubCategoryDao getSubCategoryDao ()

Retorna de la base de datos el objeto SubCategoryDao.

getSubCategoryDetailDao

private SubCategoryDetailDao getSubCategoryDetailDao ()

Retorna de la base de datos el objeto SubCategoryDetailDao.

getSubNotificationsDao

private SubNotificationsDao getSubNotificationsDao ()

Retorna de la base de datos el objeto SubNotificationsDao.

getSubNotificationsHistoryDao

private SubNotificationsHistoryDao getSubNotificationsHistoryDao ()

Retorna de la base de datos el objeto SubNotificationsHistoryDao.

getUserDao

private UserDao getUserDao ()

Retorna de la base de datos el objeto UserDao.

loadNotifyMe

public void loadNotifyMe (boolean clearFirst, final FetchNotifyMeDataCallback callback)

Método utilizado para acceder a las funciones del repositorio desde los modelos.

getSubNotificationsDetailList

public void getSubNotificationsDetailList (SubcategoryItem subcategoryItem, GetSubNotificationsListCallback callback)

Llama al método *getSubNotificationsDetailListDos* para pasarle el id del objeto SubcategoryItem y así recuperar los tags de un canal.

getSubNotificationsDetailListDos

public void getSubNotificationsDetailListDos (int subCategoryId, GetSubNotificationsListCallback callback)

Recupera de la base de datos Room una lista de objetos SubNotificationsItem cuyo identificador coincida con el identificador pasado por parámetro.

getSubNotificationsDetailListTres

public void getSubNotificationsDetailListTres (GetSubNotificationsListCallback callback)

Recupera de la base de datos Room una lista de objetos SubNotificationsItem.

getSubCategoryDetailList

public void getSubCategoryDetailList (SubcategoryItem subcategoryItem, GetSubCategoryDetailListCallback callback)

Llama al método *getSubCategoryDetailListDos* para pasarle el id del objeto SubcategoryItem y así recuperar los subcanales de un canal.

getSubCategoryDetailListDos

public void getSubCategoryDetailListDos (int subCategoryId, GetSubCategoryDetailListCallback callback)

Recupera de la base de datos Room una lista de objetos SubcategoryDetailItem cuyo identificador coincida con el identificador pasado por parámetro.

getSubcategoryListByUser

public void getSubcategoryListByUser (GetSubCategoryListCallback callback)

Recupera de la base de datos Room una lista de objetos SubcategoryItem cuyo email coincida con el email almacenado en SharedPreferences.

getSubNotificationsHistoryListByUser

public void getSubNotificationsHistoryListByUser (GetSubNotificationsHistoryListCallback callback)

Recupera de la base de datos Room una lista de objetos SubNotificationsHistoryItem cuyo email del usuario coincida con el email almacenado en SharedPreferences.

getUserList

public void getUserList (GetUserListCallback callback)

Recupera de la base de datos Room una lista con todos los usuarios.

insertSubCategory

public boolean insertSubCategory (FetchCategorias fetchCategorias)

Inserta en la base de datos un canal público fetchCategorias.

removeSubCategory

public void insertSubCategory (SubcategoryItem item)

Elimina de la base datos un canal público item.

insertSubCategoryDetail

public boolean insertSubCategoryDetail (FetchCategorias fetchCategorias)

Inserta en la base de datos un subcanal fetchCategorias.

insertSubCategoriesDetail

public boolean insertSubCategoriesDetail (List<SubcategoryDetailItem> subcategoryDetailItemList)

Inserta en la base de datos una lista de subcanales subcategoryDetailItemList.

insertExistingSubCategoryDetail

public boolean insertExistingSubCategoryDetail

(List<SubcategoryDetailItem> subcategoryDetailItemList)

Inserta en la base de datos una lista existente de subcanales subcategoryDetailItemList.

getForeignKeyValue

public int getForeignKeyValue ()

Obtiene el valor de la clave primaria de la tabla SubCategoryItem

nukeTable

public void getForeignKeyValue (int subCategoryId)

Elimina de la tabla SubCategoryDetail todos los ítems cuyo subCategoryId coincida con el valor pasado por parámetro.

checkNotification

public void checkNotification (int subCategoryId, boolean isChecked)

Actualiza el campo isChecked de la tabla SubCategoryItem (si el campo id de la tabla coincide con el valor subCategoryId) a true si isChecked es falso y a false si isChecked es verdadero.

insertSubNotifications

public void insertSubNotifications (List<SubNotificationsItem> subNotificationsItemList)

Inserta en la base de datos una lista de objetos SubNotificationsItem.

checkNotificationTag

public void checkNotificationTag (SubNotificationsItem item)

Actualiza el campo checkBoxEnabled de la tabla SubNotifications a true si ésta era falsa y viceversa. Solo si el campo id coincide con el valor pasado por parámetro.

checkAllTags

public void checkAllTags (int subCategoryId)

Actualiza el campo checkBoxEnabled de todos los ítems de la tabla SubNotifications a verdadero.

unCheckAllTags

public void unCheckAllTags (int subCategoryId)

Actualiza el campo checkBoxEnabled de todos los ítems de la tabla SubNotifications a falso.

insertSubNotificationsHistory

public void insertSubNotificationsHistory (List<SubNotificationsHistoryItem> subNotificationsHistoryItemList)

Inserta en la base de datos una lista de objetos SubNotificationsHistoryItem.

deleteSubNotificationsHistory

public void deleteSubNotificationsHistory (SubNotificationsHistoryItem> subNotificationsHistoryItem)

Elimina de la base de datos (de la tabla SubNotificationsHistoryItem) un ítem subNotificationsHistoryItem.

isOnline

public boolean isOnline ()

Comprueba si tenemos conexión a internet.

getNumberOfPages

public void getNumberOfPages (String type, GetNumberOfPagesCategory callback)

Realiza una petición HTTP a la REST API de WordPress para recuperar del header el número de página que tiene una petición de tipo type.

insertUser

public void insertUser (UserItem userItem)

Inserta en la base de datos un usuario userItem solo si dicho usuario no existe en la base de datos.

checkIfEmailExists

public boolean checkIfEmailExists ()

Método que se encarga de comprobar si el usuario que se va a insertar ya existe en la base de datos. Devuelve false si no existe y true si ya existe. Se emplea en el método *insertUser*.

getEmail

public String getEmail ()

Getter de la variable email.

setEmail

public void setEmail ()

Setter de la variable email.

setSharedPreferences

public void setSharedPreferences ()

Este método se encarga de guardar en la variable EMAIL (como preferencias) el valor del email del usuario que esta logueado. Se emplea principalmente para recibir notificaciones push cuando la aplicación esta *killed* (ya que no podemos recibir una notificación push si el valor del email es nulo).

fetchCategoriesFromFirebase

public void fetchCategoriesFromFireBase (String category, GetFirebaseChannelsListCallback callback)

Recupera de Firebase todos los canales de una categoría category.

createSubCategories

public void createSubCategories (String category, FetchFirebaseData fetchFirebaseData)

Crea un canal en Firebase con los datos del parámetro fetchFirebaseData en la categoría category.

autoSuscribe

public void autoSuscribe (String category, String key)

Este método se encargar de añadir al usuario que ha creado el canal en la tabla de Users de Firebase. Se emplea en la función *createSubCategories*.

setSpinnerChannel

public void setSpinnerChannel (GetSpinnerListCallback callback)

Añade en la lista spinnerList las categorías principales (asociaciones, deporte, educación ...).

fetchMyCategoriesFromFirebase

public void fetchMyCategoriesFromFirebase

(GetFirebaseChannelsListCallback callback)

Recupera de Firebase todos los canales de todas las categorías. Posteriormente esta lista se filtrará en el presentador de la pantalla MyChannels para obtener los canales creados por el usuario que esta logueado.

setFirebaseCategories

public void setFirebaseCategories ()

Añade en la lista categoryList las categorías principales (asociaciones, deporte, educación ...).

deleteSubCategoryFromFirebase

public void deleteSubCategoryFromFirebase (FetchFirebaseData fetchFirebaseData)

Elimina de Firebase un canal fetchFirebaseData.

subscribeToChannel

public void subscribeToChannel (String category, String key)

Añade a Firebase al usuario almacenado en la variable email al canal con referencia key de la categoría category.

deleteSubscription

public void deleteSubscription (FetchCategorias fetchCategorias)

Elimina de la tabla Users de Firebase el usuario almacenado en la variable email.

getFirebaseChannelPassword

public void getFirebaseChannelPassword (String category, String key, GetFirebaseChannelPasswordCallback callback)

Recupera de Firebase la contraseña de un canal que tiene contraseña con referencia key de la categoría category. Se emplea para comprobar la contraseña que ha introducido el usuario en el AlertDialog.

fetchChatList

public void fetchChatList (String category, String key, GetFirebaseChatListCallback callback)

Recupera de Firebase el historial del chat de un canal con referencia key de la categoría category.

sendMessage

public void sendMessage (String category, String key, String text)

Añade a la tabla Chat de Firebase del canal con referencia key de la categoría category el mensaje text.

deleteChat

public void deleteChat (String category, String key)

Elimina de Firebase la tabla chat del canal con referencia key de la categoría category. Se emplea solamente durante el testing.

setFirebaseNotification

public void setFirebaseNotification (String channel, String text) throws JSONException

Se encarga de generar una notificación FCM (Firebase Cloud Messaging). Se emplea cuando el usuario envía un mensaje al chat grupal.

setHeader

public void setHeader (Map<String, String> params)

Configura la cabecera de la notificación FCM. Se emplea en el método *setFirebaseNotification*.

setJsonObject

public void setJsonObject (String channel, String text, SharedPreferences sharedPreferences, JSONObject jsonObject, JSONObject jsonData) throws JSONException

Este método se encarga crear el fichero JSON como aparece en la **Figura 67** del apartado 3.2.1.7. Se emplea en el método *setFirebaseNotifications*.

fetchUserList

public void fetchUserList (FetchFirebaseData data, GetFirebaseUserListCallback callback)

Recupera de Firebase todos los usuarios de la tabla Users de un canal.

uploadImageToFirebase

public void uploadImageToFirebase (String firebaseImage, ImageView imageView)

Este método se encarga de subir la imagen imageView al *storage* de Firebase en la ruta firebaseImage.

setCheckBoxFirebase

public void setCheckBoxFirebase (FetchFirebaseData item)

Actualiza el campo boolean checked de un canal de Firebase (CheckBox marcado → True y CheckBox no marcado → False).

3.2. Activities

A continuación, nos centraremos en explicar los métodos del presentador y del modelo de cada Activity (sin incluir los métodos de la plantilla MVP). Una Activity en Android se corresponde con una pantalla de la aplicación, es decir la interfaz de usuario. Se compone de una clase, un layout y una definición de uso (AndroidManifest).

3.2.1. SplashScreen

SplashScreenActivity

Esta clase se encarga de la gestión de la interfaz de usuario de la pantalla *SplashScreen*.

Métodos

onCreate

protected void onCreate (Bundle savedInstanceState)

Crea la vista y carga las configuraciones de la librería EasySplashScreen para que se muestre una pantalla Splash con el título de la App y un fondo de pantalla.

3.2.2. Login

LoginPresenter

Esta clase se encarga de recibir y gestionar las acciones de usuario procedentes de la vista y de notificar al modelo para procesar dichas acciones de la pantalla Login.

Métodos

generalTopicSubscribing

public void generalTopicSubscribing ()

Para recibir notificaciones push es necesario estar suscrito en un canal y este método se encarga de auto suscribir al usuario (solo si no está suscrito previamente) al topic *general*.

getFirebaseToken

public void getFirebaseToken ()

Método para obtener el token de autenticación de Firebase. Con el token obtenido se puede realizar una petición GET utilizando el agente Postman por ejemplo, para obtener los *topics* en los que estamos suscritos (en este proyecto siempre saldrá *general* como topic).

navigateToNextScreen

public void navigateToNextScreen (String email)

Indica a la vista de navegar a la pantalla Index pasando como estado el texto del botón de forma que, en la siguiente pantalla se cargue el layout (privado o público) correspondiente.

onCompleteSignIn

public void onCompleteSignIn (Task<AuthResult> task, FirebaseAuth mAuth, String email)

Comprueba sí la autenticación contra Firebase es satisfactoria o no.

onLoginClicked

public void onLoginClicked (String email, String password, String buttonText)

Comprueba si el usuario accede como invitado o como registrado. Asimismo, se insertará en la base de datos Room al usuario si ha entrado como registrado (solo si no existe en la base de datos).

onRegisterClicked

public void onRegisterClicked (String buttonText)

Indica a la vista de navegar a la pantalla Register y al mismo tiempo que muestre un mensaje Toast del texto del botón buttonText.

LoginModel

LoginModel se encarga de guardar el valor del email en el repositorio y de insertar usuarios a la base de datos.

Métodos

insertUser

public void insertUser (UserItem userItem)

Inserta en la base de datos Room un usuario que no exista.

setRepositoryEmail

public void setRepositoryEmail (String email)

Guarda el valor del email del usuario en el repositorio.

3.2.3. Register

RegisterPresenter

Esta clase se encarga de recibir y gestionar las acciones de usuario procedentes de la vista y de notificar al modelo para procesar dichas acciones de la pantalla Register.

Métodos

checkEmailExists

public void checkEmailExists (Task<SignInMethodQueryResult> task, FirebaseAuth mAuth, String email, String password)

Comprueba si el email introducido existe en Firebase. Si el email no existe creará una nueva cuenta con ese email, y por el contrario si ya existe indicará a la vista de mostrar un mensaje Toast de que el email ya está utilizado.

createAccount

public void createAccount (Task<AuthResult> task, mAuth, String email, String password)

Registra a un nuevo usuario en el proyecto de Firebase y volverá a la pantalla Login.

navigateToNextScreen

public void navigateToNextScreen ()

Indica a la vista de navegar a la pantalla Login.

onRegisterClicked

public void onRegisterClicked (String etEmail, String etPassword,
String etPassword2)

Comprueba las excepciones que se producen cuando el usuario hace clic en el botón Sign up. Existe cinco excepciones: email vacío, contraseña vacía, contraseña demasiada corta (mínimo seis caracteres), repite la contraseña, las contraseñas no coinciden.

setSupportActionBar

public void setSupportActionBar(ActionBar supportActionBar)

Configura el ActionBar.

3.2.4. Index

IndexPresenter

Esta clase se encarga de recibir y gestionar las acciones de usuario procedentes de la vista y de notificar al modelo para procesar dichas acciones de la pantalla Index.

Métodos

checkListIsEmpty

public void checkListIsEmpty ()

Comprueba si la lista que contiene las categorías de WordPress está vacía. Si no está vacía, es decir que se ha podido recuperar los datos de WordPress entonces avisará a la vista de eliminar el LoadingDialog.

fetchCategoriesFromURL

public void fetchCategoriesFromURL ()

Llama al modelo para recupere el número de páginas que tiene la petición (paginación) para posteriormente pasárselo por parámetro al método del modelo, para que recupere de la API de WP las categorías. Si la lista está vacía esconderá el RecyclerView y mostrará una imagen con un texto indicando que hubo un error.

navigateToCreateChannel

public void navigateToCreateChannel ()

Indica a la vista de navegar a la pantalla CreateChannel para crear un canal sin contraseña.

navigateToCreateChannelWithPassword

public void navigateToCreateChannelWithPassword ()

Indica a la vista de navegar a la pantalla CreateChannel para crear un canal con contraseña.

onCategoryClicked

public void onCategoryClicked (FetchCategorias category)

Indica a la vista de navegar a la pantalla SubCategory y que pase el estado del canal elegido de forma que en la pantalla siguiente se carguen los canales de esa categoría.

onCategoryName

public void onCategoryName (String spinner)

Guarda el estado de la categoría elegida en el spinner.

onNavigationItemSelected

public void onNavigationItemSelected (MenuItem item)

Se encarga de navegar a la vista correspondiente en función del ítem pulsado en la barra de navegación.

onPrivateSelected

public void onPrivateSelected ()

Indica a la vista de mostrar un AlertDialog con un spinner para que el usuario puede elegir la categoría en la que quiere buscar el canal privado. Finalmente actualiza el estado *searchType* a privado.

onPublicSelected

public void onPublicSelected ()

Actualiza el estado *searchType* a público.

onQueryTextSubmit

public void onQueryTextSubmit ()

Recupera la query introducida en el buscador de canales (SearchView) e indica a la vista de pasar a la pantalla Search junto con los estados (público o privado).

onSearchType

public void onSearchType ()

Se encarga de averiguar si buscamos canales privados o canales públicos en el SearchView.

reloadData

public void reloadData ()

Refresca la pantalla.

setAdapter

public void setAdapter (Spinner spinner)

Asigna el adaptador para el spinner del AlertDialog.

setSearchView

public void setSearchView (SearchView searchView)

Guarda el estado del SearchView, de forma que pueda restablecer el menú en caso necesario.

setSpinnerChannel

public void setSpinnerChannel ()

Guarda en la lista spinnerList del ViewModel la lista de categorías principales.

setUpLayout

public void setUpLayout (NavigationView navigationView, View headerView, TextView tv_userEmail)

Carga un menú u otro dependiendo si el usuario ha entrado como registrado o invitado.

IndexModel

IndexModel se encargará principalmente de obtener los datos de la REST API de WordPress empleando la librería Volley.

Métodos

extractCategoriesFromURL

public void extractCategoriesFromURL (int pages, IndexModel.VolleyCallBack myCallBack)

Utiliza Volley para extraer la información (categorías principales) del fichero JSON devuelto por la REST API de WordPress.

fetchSpinnerList

public void fetchSpinnerList (RepositoryContract.GetSpinnerListCallback callback)

Recupera del repositorio la lista de categorías principales para utilizarlas posteriormente en el spinner del AlertDialog.

getNumberOfPages

public void getNumberOfPages (String type, RepositoryContract.GetNumberOfPagesCategory callback)

Llama al repositorio para que recupere el número de páginas que tiene la REST API en cuanto a las categorías para realizar la paginación.

isOnline

public boolean isOnline ()

Comprueba si tenemos conexión a internet.

3.2.5. SubCategory

SubCategoryPresenter

Esta clase se encarga de recibir y gestionar las acciones de usuario procedentes de la vista y de notificar al modelo para procesar dichas acciones de la pantalla SubCategory.

Métodos

checkListIsEmpty

public void checkListIsEmpty ()

Comprueba si la lista que contiene las subcategorías de WordPress está vacía. Si no está vacía, es decir que se ha podido recuperar los datos de WordPress entonces avisará a la vista de eliminar el LoadingDialog.

fetchCategoriesFromURL

public void fetchCategoriesFromURL ()

Llama al modelo para recupere el número de páginas que tiene la petición (paginación), para posteriormente pasárselo por parámetro al método del modelo que recupera de la API de WP las categorías. Si la lista está vacía esconderá el RecyclerView y se mostrará una imagen con un texto indicando que hubo un error.

fetchDataFromDataBase

public void fetchDataFromDataBase ()

Llama al modelo para que recupere de la base de datos Room los canales en los que está suscrito el usuario y actualiza la lista.

fetchFirebaseData

public void fetchFirebaseData ()

Llama al modelo para que recupere de Firebase los canales privados para comprobar que están en la base de datos. Si coinciden los nombres se cambiará el texto del botón de *follow* a *following* del ítem de la lista.

fetchSubCategoriesDetailFromURL

public void fetchSubCategoriesDetailFromURL ()

Llama al modelo para que recupere de WordPress el detalle del canal en el que el usuario ha pulsado seguir. Estos datos se almacenarán en la base de datos Room en la tabla SubCategoryDetailItem.

fetchTagsFromURL

public void fetchTagsFromURL ()

Llama al modelo para que recupere de WordPress los *tags* (Eventos, noticias generales, concursos etc.) y todas las páginas (una página en WordPress corresponde con el nombre del canal y en ella aparecerán todas las publicaciones relacionado con ese canal).

Por otra parte, mapeará el id de un tag de una página de WordPress con el nombre del tag. Esto es debido a que si hacemos una petición a la URL para extraer las páginas de la REST API devuelve el valor del tag en valor numérico, el cual no nos sirve. Por eso necesitamos realizar primero a una petición a la URL de WordPress para recuperar todos los tags con su nombre y valor numérico y después mapearlos.

navigateToCreateChannel

public void navigateToCreateChannel ()

Indica a la vista de navegar a la pantalla CreateChannel para crear un canal sin contraseña.

navigateToCreateChannelWithPassword

public void navigateToCreateChannelWithPassword ()

Indica a la vista de navegar a la pantalla CreateChannel para crear un canal con contraseña.

onCategoryName

public void onCategoryName (String spinner)

Actualiza el estado *categoryName* al valor elegido en el spinner.

onFloatingButtonClicked

public void onFloatingButtonClicked (FloatingActionButton fab,
TextView tv_typeChannel)

Reemplaza los canales públicos por los canales privados y viceversa. Se podría decir que actúa como un “switch” para cambiar entre canales públicos y canales privados.

onFloatingButtonCreate

public void onFloatingButtonCreate ()

Indica a la vista de navegar a la pantalla CreateChannels.

onFloatingButtonMyChannels

public void onFloatingButtonCreate ()

Indica a la vista de navegar a la pantalla MyChannels.

onPrivateSelected

public void onPrivateSelected ()

Indica a la vista de mostrar un AlertDialog con un spinner para que el usuario puede elegir la categoría en la que quiere buscar el canal privado. Finalmente actualiza el estado *searchType* a privado.

onPrivateSubCategoryFollowClicked

public void onPrivateSubCategoryFollowClicked (String password, FetchCategorias channel)

Comprueba que la contraseña introducida en el AlertDialog coincide con la contraseña guardada en el ViewModel. Si es incorrecta avisa a la vista de mostrar un mensaje Toast indicando que la contraseña es incorrecta. Por el contrario, si es correcta avisará al modelo de añadir al usuario en la base de datos de Firebase.

onPrivateSubCategoryFollowClickedNoPass

public void onPrivateSubCategoryFollowClickedNoPass (FetchCategorias channel)

Llama al modelo para que inserte directamente al usuario en la base de datos de Firebase, sin la necesidad de introducir una contraseña para suscribirse al canal privado.

onPublicSelected

public void onPublicSelected ()

Actualiza el estado *searchType* a public.

onQueryTextSubmit

public void onQueryTextSubmit ()

Recupera la query introducida en el buscador de canales (SearchView) y pasa a la pantalla Search junto con los estados (público o privado).

onSearchType

public void onSearchType ()

Se encarga de averiguar si buscamos canales privados o canales públicos en el SearchView.

onSubCategoryClicked

public void onSubCategoryClicked (FetchCategorias subCategory)

Indica a la vista de pasar a la pantalla SubCategoryDetail junto con el estado (el canal que ha pulsado el usuario) para que se muestren las publicaciones o un chat dependiendo si el canal era público o privado.

onSubCategoryFollowClicked

public void onSubCategoryFollowClicked (FetchCategorias subCategory)

Este método se encarga de realizar varias funciones:

1. Si el canal es privado el usuario se estará suscribiéndose a un canal de Firebase.

1.1. Si el canal no tiene contraseña, el usuario se suscribe directamente.

1.2. Si el canal tiene contraseña, avisará a la vista de mostrar un AlertDialog para que el usuario introduzca la contraseña.

2. Si el canal es público el usuario se estará suscribiéndose a un canal de WordPress.

reloadData

public void reloadData ()

Refresca la pantalla.

setAdapter

public void setAdapter (Spinner spinner)

Asigna el adaptador para el spinner del AlertDialog.

setSpinnerChannel

public void setSpinnerChannel ()

Guarda en la lista spinnerList del ViewModel la lista de categorías principales.

setSupportActionBar

public void setSupportActionBar(ActionBar supportActionBar)

Configura el ActionBar.

setUpLayout

public void setUpLayout (NavigationView navigationView, View headerView, TextView tv_userEmail)

Carga un menú u otro dependiendo si el usuario ha entrado como registrado o invitado.

suscribingToFirebaseChannel

private void suscribingToFirebaseChannel (FetchCategorias channel)

Avisa al modelo de añadir al usuario a la base de datos de Firebase.

SubCategoryModel

SubCategoryModel se encargará de obtener los canales y subcanales desde la REST API de WordPress empleando la librería Volley o de Firebase si el canal es privado.

Constructores

SubCategoryModel

public SubCategoryModel (RepositoryContract repository)

Constructor que inicializa el repositorio, y las diferentes listas de objetos.

Métodos

extractCategoriesFromURL

public void extractCategoriesFromURL (int pages, SubCategoryModel.VolleyCallBack myCallBack)

Utiliza Volley para extraer los canales del fichero JSON devuelto por la REST API de WordPress.

La URL se vería de la siguiente forma:

https://notifyme47.herokuapp.com/wp-json/wp/v2/categories?&page=" + page

Variables:

page representa el número de página (paginación).

extractPages

public void extractPages (int pages, SubCategoryModel.VolleyCallBackPages myCallBack)

Utiliza Volley para extraer las páginas del fichero JSON devuelto por la REST API de WordPress.

La URL se vería de la siguiente forma:

https://notifyme47.herokuapp.com/wp-json/wp/v2/pages?categories=" + subCategoryId + "&page=" + page

Variables:

page representa el número de página (paginación).

subCategoryId representa el id de una categoría y lo actualiza el presentador.

extractSubCategories

public void extractSubCategories (int pages, SubCategoryModel.VolleyCallBack2 myCallBack)

Utiliza Volley para extraer las publicaciones de un determinado canal del fichero JSON devuelto por la REST API de WordPress.

La URL se vería de la siguiente forma:

https://notifyme47.herokuapp.com/wp-json/wp/v2/posts?categories=" + subCategoryId + "&page=" + page

Variables:

page representa el número de página (paginación).

subCategoryId representa el id de una categoría y lo actualiza el presentador.

extractTags

public void extractTags (int pages, SubCategoryModel.VolleyCallBackTags myCallBack)

Utiliza Volley para extraer los tags del fichero JSON devuelto por la REST API de WordPress.

La URL se vería de la siguiente forma:

https://notifyme47.herokuapp.com/wp-json/wp/v2/tags?page=" + page

Variables:

page representa el número de página (paginación).

fetchFirebaseData

public void fetchFirebaseData (String categoryName, RepositoryContract.GetFirebaseChannelsListCallback callback)

Llama al repositorio para que recupere de Firebase los canales de la categoría pasada por parámetro.

fetchSpinnerList

public void fetchFirebaseData (RepositoryContract.GetSpinnerListCallback callback)

Llama al repositorio para que recupere las categorías principales. Estas categorías se emplean para rellenar el spinner del AlertDialog.

fetchSubCategoryListDataByUser

public void fetchSubCategoryListDataByUser (RepositoryContract.GetSubCategoryListCallback callback)

Llama al repositorio para que recupere la lista de los canales en los que está suscrito el usuario.

getFirebaseChannelPassword

public void getFirebaseChannelPassword (String category, String key, RepositoryContract.GetFirebaseChannelPasswordCallback callback)

Llama al repositorio para que recupere de Firebase la contraseña de un canal.

getNumberOfPages

public void getNumberOfPages (String type, RepositoryContract.GetNumberOfPagesCategory callback)

Recupera el número de páginas que tiene la REST API en cuanto a los canales, tags y pages para realizar la paginación.

insertSubCategoriesDetailToDataBase

public void insertSubCategoriesDetailToDataBase ()

Llama al repositorio para que inserte en la base de datos una lista de objetos SubCategoryDetailItem.

insertSubCategoryDetailToDataBase

public void insertSubCategoryDetailToDataBase (FetchCategorias fetchCategorias)

Llama al repositorio para que inserte en la base de datos el objeto fetchCategorias pasado por parámetro.

insertSubCategoryToDataBase

public void insertSubCategoryToDataBase (FetchCategorias fetchCategorias)

Llama al repositorio para que inserte en la base de datos el objeto fetchCategorias pasado por parámetro.

insertSubNotificationsItem

public void insertSubNotificationsItem (List<SubNotificationsItem> subNotificationsItemList)

Llama al repositorio para que inserte en la base de datos una lista de objetos SubNotificationsItem.

isOnline

public boolean isOnline ()

Comprueba si tenemos conexión a internet.

renameCategoriesFromURL

public void renameCategoriesFromURL (String name)

Se encarga de comparar los canales que están en la base datos con los canales recuperados de WordPress. Si los nombres coinciden cambia el texto del botón de *follow* a *following*.

subscribeToChannel

public void setSubCategoryId (String category, String key)

Llama al repositorio para que añada al usuario a la base de datos de Firebase.

3.2.6. SubCategoryDetail

SubCategoryPresenter

Esta clase se encarga de recibir y gestionar las acciones de usuario procedentes de la vista y de notificar al modelo para procesar dichas acciones de la pantalla SubCategoryDetail.

Métodos

checkListIsEmpty

public void checkListIsEmpty ()

Comprueba si la lista que contiene las subcategorías de WordPress está vacía. Si no está vacía, es decir que se ha podido recuperar los datos de WordPress entonces avisará a la vista de eliminar el LoadingDialog.

fetchSubCategoriesFromURL

public void fetchSubCategoriesFromURL ()

Llama al modelo para recupere el número de páginas que tiene la petición (paginación) para posteriormente pasárselo por parámetro al método del modelo, para que recupere de la API de WP las subcategorías. Si la lista está vacía esconderá el RecyclerView y mostrará una imagen con un texto indicando que hubo un error.

onSubCategoryDetailClicked

public void onSubCategoryDetailClicked (FetchCategorias subCategoryDetail)

Indica a la vista de pasar a la pantalla Posts junto con el estado (el subcanal que ha pulsado el usuario) para que se muestre el detalle de la publicación.

reloadData

public void reloadData ()

Refresca la pantalla.

setSupportActionBar

public void setSupportActionBar(ActionBar supportActionBar)

Configura el ActionBar.

SubCategoryDetailModel

SubCategoryDetailModel se encargará de comprobar si existe conectividad a internet y recuperará de WordPress los subcanales o publicaciones pertenecientes a un canal.

Métodos

extractSubCategories

public void extractSubCategories (int pages, SubCategoryDetailModel.VolleyCallBack myCallBack)

Utiliza Volley para extraer los subcanales del fichero JSON devuelto por la REST API de WordPress.

La URL se vería de la siguiente forma:

https://notifyme47.herokuapp.com/wp-json/wp/v2/posts?categories=" + subCategoryId + "&page=" + page

Variables:

subCategoryId: representa el id de una un canal o subcategoría.

page: representa el número de página (paginación).

isOnline

public boolean isOnline ()

Comprueba si tenemos conexión a internet.

3.2.7. Posts

PostsPresenter

Esta clase se encarga de recibir y gestionar las acciones de usuario procedentes de la vista y de notificar al modelo para procesar dichas acciones de la pantalla Posts.

Métodos

onShare

public void onShare ()

Construye la URL para pasárselo a la vista y que construya el intent hacia la pantalla share de Android.

setSupportActionBar

public void setSupportActionBar (ActionBar supportActionBar)

Configura el ActionBar.

3.2.8. PublicChannels

PublicChannelPresenter

Esta clase se encarga de recibir y gestionar las acciones de usuario procedentes de la vista y de notificar al modelo para procesar dichas acciones de la pantalla PublicChannel.

Métodos

fetchDataDetailFromDataBase

public void fetchDataDetailFromDataBase ()

Llama al modelo para que actualice la variable *subCategoryIdTable* y que recupere de la base de datos Room las subcategorías de un canal.

fetchDataFromDataBase

public void fetchDataFromDataBase ()

Llama al modelo para que recupere de la base de datos Room los canales públicos en los que está suscrito el usuario.

fetchSubCategoriesDetailFromURL

public void fetchSubCategoriesDetailFromURL ()

Llama al modelo para que recupere de WordPress las publicaciones o subcanales de un canal de forma que, si hay nuevas publicaciones se actualizará la base de datos Room añadiéndose estas nuevas publicaciones.

navigateToCreateChannel

public void navigateToCreateChannel ()

Indica a la vista de navegar a la pantalla CreateChannel para crear un canal sin contraseña.

navigateToCreateChannelWithPassword

public void navigateToCreateChannelWithPassword ()

Indica a la vista de navegar a la pantalla CreateChannel para crear un canal con contraseña.

onAlertDialogClick

public void onAlertDialogClick ()

Llama al modelo para eliminar de la base de datos el canal en el que se ha pulsado *unfollow*.

onNavigationItemSelected

public void onNavigationItemSelected (MenuItem item)

Se encarga de manejar los eventos de los ítems de la barra de navegación

onPublicChannelClicked

public void onPublicChannelClicked (SubcategoryItem item)

Indica a la vista de pasar a la pantalla PublicChannelDetail junto con el estado llamando al método passDataToPublicChannelDetailScreen, de forma que se carguen las publicaciones correspondientes al canal pulsado.

onUnFollowClicked

public void onUnFollowClicked (SubcategoryItem item)

Indica a la vista de mostrar un AlertDialog preguntado si realmente quiere eliminar el canal. Si la respuesta es sí entonces se llamará al método onAlertDialogClick.

passDataToPublicChannelDetailScreen

private void passDataToPublicChannelDetailScreen (SubcategoryItem item)

Este método se encarga de llamar al mediador para realizar un set del estado ítem (SubCategoryItem).

setSupportActionBar

public void setSupportActionBar(ActionBar supportActionBar)

Configura el ActionBar.

setUpLayout

public void setUpLayout (NavigationView navigationView, View headerView, TextView tv_userEmail)

Carga un menú u otro dependiendo si el usuario ha entrado como registrado o invitado.

PublicChannelModel

PublicChannelModel se encargará de recuperar de la base de datos los canales públicos a los que está suscrito el usuario. También actualizará la base de datos si hay nuevas publicaciones del canal en WordPress.

Métodos

extractSubcategories

public void extractSubcategories (int pages, PublicChannel.VolleyCallBack myCallBack)

Utiliza Volley para extraer los subcanales del fichero JSON devuelto por la REST API de WordPress.

La URL se vería de la siguiente forma:

https://notifyme47.herokuapp.com/wp-json/wp/v2/posts?categories=" + subCategoryId + "&page=" + page

Variables:

subCategoryId representa el id del canal.

page representa el número de página (paginación).

fetchSubCategoryDetailListData

public void fetchSubCategoryDetailListData (SubcategoryItem subcategoryItem, RepositoryContract.GetSubCategoryDetailListCallback callback)

Llama al repositorio para que recupere las publicaciones de un canal. Se utiliza para comparar el tamaño de la lista de las publicaciones extraídas de la base de datos con la lista de las publicaciones recuperadas de WordPress.

fetchSubCategoryListDataByUser

public void fetchSubCategoryListDataByUser (RepositoryContract.GetSubCategoryListCallback callback)

Llama al repositorio para que recupere la lista de los canales en los que está suscrito el usuario.

getNumberOfPages

public void getNumberOfPages (String type, RepositoryContract.GetNumberOfPagesCategory callback)

Recupera el número de páginas que tiene la REST API en cuanto a los canales para realizar la paginación.

getSubCategoryId

public String getSubCategoryId ()

Getter de la variable subCategoryId.

getSubCategoryIdTable

public int getSubCategoryIdTable ()

Getter de la variable subCategoryIdTable.

insertExistingSubCategoriesDetailToDataBase

public void insertExistingSubCategoriesDetailToDataBase ()

Llama al repositorio para que inserte en la base de datos una lista de objetos SubCategoryDetailItem. Solo se llama si hay nuevas publicaciones en WordPress.

onUnFollowClicked

public void onUnFollowClicked (SubcategoryItem item)

Llama al repositorio para eliminar en la base de datos un ítem SubcategoryItem.

resetLocalDatabase

public void resetLocalDatabase (int subCategoryId)

Llama al repositorio para eliminar en la base de datos todos aquellos ítems cuyo subCategoryId coincidan con el subCategoryId pasado por parámetro.

setSubCategoryId

public void setSubCategoryId (String subCategoryId)

Setter de la variable subCategoryId.

setSubCategoryIdTable

public void setSubCategoryIdTable (int subCategoryIdTable)

Setter de la variable subCategoryIdTable.

3.2.9. PublicChannelsDetail

PublicChannelDetailPresenter

Esta clase se encarga de recibir y gestionar las acciones de usuario procedentes de la vista y de notificar al modelo para procesar dichas acciones de la pantalla PublicChannelsDetail.

Métodos

fetchDataDetailFromDataBase

public void fetchDataDetailFromDataBase ()

Llama al modelo para que recupere de la base de datos Room las subcategorías de un canal.

onDetailClicked

public void onDetailClicked (SubcategoryDetailItem item)

Indica a la vista de navegar a la pantalla Posts junto con los datos (estados) a mostrar.

setSupportActionBar

public void setSupportActionBar(ActionBar supportActionBar)

Configura el ActionBar.

PublicChannelDetailModel

PublicChannelsDetailModel se encargará únicamente de extraer de la base de datos las publicaciones de un determinado canal (canal que se haya elegido en PublicChannel) y mostrarlos en una lista RecyclerView.

Métodos

fetchSubCategoryDetailListData

public void fetchSubCategoryDetailListData (SubcategoryItem subcategoryItem, RepositoryContract.GetSubCategoryDetailListCallback callback)

Llama al modelo para que recupere las publicaciones de un canal.

3.2.10. Search

SearchPresenter

Esta clase se encarga de recibir y gestionar las acciones de usuario procedentes de la vista y de notificar al modelo para procesar dichas acciones de la pantalla Search.

Métodos

checkListIsEmpty

public void checkListIsEmpty ()

Comprueba si la lista que contiene las subcategorías de WordPress está vacía. Si no está vacía, es decir que se ha podido recuperar los datos de WordPress entonces avisará a la vista de eliminar el LoadingDialog.

fetchCategoriesAllFromURL

public void fetchCategoriesFromURL ()

Llama al modelo para que recupere el número de páginas que tiene la petición (paginación), para posteriormente pasárselo por parámetro al método del modelo que recupera de la API de WP todos los canales. Si la lista está vacía esconderá el RecyclerView y se mostrará una imagen con un texto indicando que hubo un error.

fetchDataFrom

public void fetchDataFrom ()

Este método se encarga se de comprobar si el usuario ha entrado como invitado o registrado, de forma que si ha entrado como invitado no mostrará un AlertDialog preguntado si quiere buscar canales públicos o privados (se emplea a la hora de buscar un canal), dado que el usuario invitado solo puede buscar canales públicos.

fetchDataFromDataBase

public void fetchDataFromDataBase ()

Compara la lista recuperada de WordPress con la lista recuperada de la base datos. Si los nombres coinciden significa que el usuario está suscrito a esos canales y por tanto se actualizará el texto de los botones de *follow* a *following* de los ítems de la lista.

fetchFirebaseData

public void fetchFirebaseData ()

Llama al modelo para que recupere de Firebase los canales privados para comprobar que están en la base de datos. Si coinciden los nombres se cambiará el texto del botón de *follow* a *following* del ítem de la lista.

fetchSubCategoriesDetailFromURL

public void fetchSubCategoriesDetailFromURL ()

Llama al modelo para que recupere de WordPress el detalle del canal en el que el usuario ha pulsado seguir. Estos datos se almacenarán en la base de datos Room en la tabla SubCategoryDetailItem.

fetchTagsFromURL

public void fetchTagsFromURL ()

Llama al modelo para que recupere de WordPress los *tags* (Eventos, noticias generales, concursos etc.) y todas las páginas (una página en WordPress corresponde con el nombre del canal y en ella aparecerán todas las publicaciones relacionado con ese canal).

Por otra parte, mapeará el id de un tag de una página de WordPress con el nombre del tag. Esto es debido a que si hacemos una petición a la URL para extraer las páginas de la REST API devuelve el valor del tag en valor numérico, el cual no nos sirve. Por eso necesitamos realizar primero a una petición a la URL de WordPress para recuperar todos los tags con su nombre y valor numérico y después mapearlos.

onCategoryName

public void onCategoryName (String spinner)

Actualiza el estado *categoryName* al valor elegido en el spinner.

onPrivateSelected

public void onPrivateSelected ()

Indica a la vista de mostrar un AlertDialog con un spinner para que el usuario puede elegir la categoría en la que quiere buscar el canal privado. Finalmente actualiza el estado *searchType* a privado.

onPrivateSubCategoryFollowClicked

public void onPrivateSubCategoryFollowClicked (String password,
FetchCategorias channel)

Comprueba que la contraseña introducida en el AlertDialog coincide con la contraseña guardada en el ViewModel. Si es incorrecta avisa a la vista de mostrar un mensaje Toast indicando que la contraseña es incorrecta. Por el contrario, si es correcta avisará al modelo de añadir al usuario en la base de datos de Firebase.

onPrivateSubCategoryFollowClickedNoPass

public void onPrivateSubCategoryFollowClickedNoPass (FetchCategorias channel)

Llama al modelo para que inserte directamente al usuario en la base de datos de Firebase, sin la necesidad de introducir una contraseña para suscribirse al canal privado.

onPublicSelected

public void onPublicSelected ()

Actualiza el estado *searchType* a public.

onQueryTextSubmit

public void onQueryTextSubmit ()

Recupera la query introducida en el buscador de canales (SearchView) y pasa a la pantalla Search junto con los estados (público o privado).

onSearchType

public void onSearchType ()

Se encarga de averiguar si buscamos canales privados o canales públicos en el SearchView.

onSubCategoryClicked

public void onSubCategoryClicked (FetchCategorias subCategory)

Indica a la vista de pasar a la pantalla SubCategoryDetail junto con el estado (el canal que ha pulsado el usuario) para que se muestren las publicaciones o un chat dependiendo si el canal era público o privado.

onSubCategoryFollowClicked

public void onSubCategoryFollowClicked (FetchCategorias subCategory)

Este método se encarga de realizar varias funciones:

1. Si el canal es privado el usuario se estará suscribiéndose a un canal de Firebase.
1.1. Si el canal no tiene contraseña, el usuario se suscribe directamente.
1.2. Si el canal tiene contraseña, avisará a la vista de mostrar un AlertDialog para que el usuario introduzca la contraseña.

2. Si el canal es público el usuario se estará suscribiéndose a un canal de WordPress.

reloadData

public void reloadData ()

Refresca la pantalla.

setAdapter

public void setAdapter (Spinner spinner)

Asigna el adaptador para el spinner del AlertDialog.

setSpinnerChannel

public void setSpinnerChannel ()

Guarda en la lista spinnerList del ViewModel la lista de categorías principales.

suscribingToFirebaseChannel

private void suscribingToFirebaseChannel (FetchCategorias channel)

Avisa al modelo de añadir al usuario a la base de datos de Firebase.

SearchModel

SearchModel se encargará de obtener los canales, subcanales y demás desde la REST API de WordPress empleando la librería Volley. Por otro parte, obtendrá los canales privados de Firebase e insertará también los usuarios en la base de datos Room o en Firebase dependiendo del tipo de canal (público o privado).

Métodos

extractAllCategoriesFromURL

public void extractCategoriesFromURL (int pages, SearchModel.VolleyCallBack myCallBack)

Utiliza Volley para extraer todos los canales del fichero JSON devuelto por la REST API de WordPress.

La URL se vería de la siguiente forma:

https://notifyme47.herokuapp.com/wp-json/wp/v2/categories?&page=" + page

Variables:

page representa el número de página (paginación).

extractPages

public void extractPages (int pages, SearchModel.VolleyCallBackPages myCallBack)

Utiliza Volley para extraer las páginas del fichero JSON devuelto por la REST API de WordPress.

La URL se vería de la siguiente forma:

https://notifyme47.herokuapp.com/wp-json/wp/v2/pages?categories=" + subCategoryId + "&page=" + page

Variables:

page representa el número de página (paginación).

subCategoryId representa el id de una categoría y lo actualiza el presentador.

extractSubCategories

public void extractSubCategories (int pages, SearchModel.VolleyCallBack2 myCallBack)

Utiliza Volley para extraer las publicaciones de un determinado canal del fichero JSON devuelto por la REST API de WordPress.

La URL se vería de la siguiente forma:

https://notifyme47.herokuapp.com/wp-json/wp/v2/posts?categories=" + subCategoryId + "&page=" + page

Variables:

page representa el número de página (paginación).

subCategoryId representa el id de una categoría y lo actualiza el presentador.

extractTags

public void extractTags (int pages, SearchModel.VolleyCallBackTags myCallBack)

Utiliza Volley para extraer los tags del fichero JSON devuelto por la REST API de WordPress.

La URL se vería de la siguiente forma:

https://notifyme47.herokuapp.com/wp-json/wp/v2/tags?page=" + page

Variables:

page representa el número de página (paginación).

fetchFirebaseData

public void fetchFirebaseData (String categoryName, RepositoryContract.GetFirebaseChannelsListCallback callback)

Llama al repositorio para que recupere de Firebase los canales pertenecientes a la categoría categoryName.

fetchSpinnerList

public void fetchFirebaseData (RepositoryContract.GetSpinnerListCallback callback)

Llama al repositorio para que recupere las categorías principales. Estas categorías se emplean para rellenar el spinner del AlertDialog.

fetchSubCategoryListDataByUser

public void fetchSubCategoryListDataByUser (RepositoryContract.GetSubCategoryListCallback callback)

Llama al repositorio para que recupere la lista de los canales en los que está suscrito el usuario.

getFirebaseChannelPassword

public void getFirebaseChannelPassword (String category, String key, RepositoryContract.GetFirebaseChannelPasswordCallback callback)

Llama al repositorio para que recupere de Firebase la contraseña de un canal (key).

getNumberOfPages

public void getNumberOfPages (String type, RepositoryContract.GetNumberOfPagesCategory callback)

Recupera el número de páginas que tiene la REST API en cuanto a los canales, tags y pages para realizar la paginación.

getSubCategoryId

public String getSubCategoryId ()

Getter de la variable subCategoryId.

insertSubCategoriesDetailToDataBase

public void insertSubCategoriesDetailToDataBase ()

Llama al repositorio para que inserte en la base de datos una lista de objetos SubCategoryDetailItem.

insertSubCategoryToDataBase

public void insertSubCategoryToDataBase (FetchCategorias fetchCategorias)

Llama al repositorio para que inserte en la base de datos el objeto fetchCategorias pasado por parámetro.

insertSubNotificationsItem

public void insertSubNotificationsItem (List<SubNotificationsItem> subNotificationsItemList)

Llama al repositorio para que inserte en la base de datos una lista de objetos SubNotificationsItem.

isOnline

public boolean isOnline ()

Comprueba si tenemos conexión a internet.

renameCategoriesFromURL

public void renameCategoriesFromURL (String name)

Se encarga de comparar los canales que están en la base datos con los canales recuperados de WordPress. Si los nombres coinciden cambia el texto del botón de *follow* a *following*.

setSubCategoryId

public void setSubCategoryId (String subCategoryId)

Setter de la variable subCategoryId.

subscribeToChannel

public void setSubCategoryId (String category, String key)

Llama al repositorio para que añada al usuario a la base de datos de Firebase.

3.2.11. PublicNotifications

PublicNotificationsPresenter

Esta clase se encarga de recibir y gestionar las acciones de usuario procedentes de la vista y de notificar al modelo para procesar dichas acciones de la pantalla PublicNotifications.

Métodos

fetchDataFromDataBase

public void fetchDataFromDataBase ()

Llama al modelo para que recupere de la base de datos Room los canales públicos en los que está suscrito el usuario. Para filtrar los canales públicos se ha realizado por el método iterator.

navigateToCreateChannel

public void navigateToCreateChannel ()

Indica a la vista de navegar a la pantalla CreateChannel para crear un canal sin contraseña.

navigateToCreateChannelWithPassword

public void navigateToCreateChannelWithPassword ()

Indica a la vista de navegar a la pantalla CreateChannel para crear un canal con contraseña.

onBellClicked

public void onBellClicked ()

Indica a la vista de navegar a la pantalla PublicNotificationsHistory. Esta opción solo se puede utilizar si estamos suscrito a un canal o más.

onExtraClicked

public void onExtraClicked (SubcategoryItem item)

Indica a la vista de navegar a la pantalla PublicNotificationsDetail, donde el usuario podrá personalizar las notificaciones que se vayan a recibir.

onNavigationItemSelected

public void onNavigationItemSelected (MenuItem item)

Se encarga de manejar los eventos de los ítems de la barra de navegación

onNotificationClicked

public void onNotificationClicked (SubcategoryItem item)

Indica al modelo de que se ha pulsado en el checkbox para que actualice el valor booleano de la tabla (true a false y viceversa) de la base datos.

setSupportActionBar

public void setSupportActionBar(ActionBar supportActionBar)

Configura el ActionBar.

setUpLayout

public void setUpLayout (NavigationView navigationView, View headerView, TextView tv_userEmail)

Carga un menú u otro dependiendo si el usuario ha entrado como registrado o invitado.

PublicNotificationsModel

PublicNotificationsModel se encargará de recuperar de la base de datos los canales públicos en los que está suscrito el usuario para que pueda configurar las notificaciones. Por otro lado, todos los cambios que se realicen los actualizará en la base de datos Room.

Métodos

fetchSubCategoryListDataByUser

public void fetchSubCategoryListDataByUser (RepositoryContract.GetSubCategoryListCallback callback)

Llama al repositorio para que recupere la lista de los canales en los que está suscrito el usuario.

notificationChecked

public void notificationChecked (int subCategoryId, boolean isChecked)

Llama al repositorio para que actualice el ítem de la base de datos, es decir si el usuario ha pulsado en el checkbox se cambiará también en la base de datos (true → marcado y false → no marcado). En resumidas cuentas, guarda el estado del checkbox en la base de datos.

3.2.12. PublicNotificationsDetail

PublicNotificationsDetailPresenter

Esta clase se encarga de recibir y gestionar las acciones de usuario procedentes de la vista y de notificar al modelo para procesar dichas acciones de la pantalla PublicNotificationsDetail.

Métodos

fetchSubNotificationsTagsFromDataBase

public void fetchSubNotificationsTagsFromDataBase ()

Llama al repositorio para recuperar de la base de datos Room los *tags* de un canal.

onNotificationDetailAllClicked

public void onNotificationDetailAllClicked ()

Indica al modelo de que se han marcado todos los *tags* del canal.

onNotificationDetailAllClickedReverse

public void onNotificationDetailAllClickedReverse ()

Indica al modelo de que se han desmarcado todos los *tags* del canal.

onNotificationDetailClicked

public void onNotificationDetailClicked (SubNotificationsItem item)

Indica al modelo de que se ha marcado un checkbox, es decir un *tag*.

setSupportActionBar

public void setSupportActionBar (ActionBar supportActionBar)

Configura el ActionBar.

PublicNotificationsDetailModel

Este modelo tendrá como trabajo recuperar de la base de datos todos los *tags* de un canal y de actualizar el estado de los ítems (PublicNotificationsDetailItem) en los que se ha marcado o desmarcado en la base de datos.

Métodos

checkAllNotifications

public void checkAllNotifications (int id)

Llama al repositorio para que actualice el campo checkBoxEnabled de los ítems PublicNotificationsDetail a verdadero (marcado).

fetchSubNotificationsListData

public void fetchSubNotificationsListData (SubcategoryItem item, RepositoryContract.GetSubNotificationsListCallback callback)

Recupera del repositorio todos los *tags* de un determinado canal..

notificationTagChecked

public void notificationTagChecked (SubNotificationsItem item)

Llama al repositorio para que actualice el estado del checkbox de un ítem PublicNotificationsDetail en la base de datos (marcado → true y desmarcado → false).

uncheckAllNotifications

public void unCheckAllNotifications (int id)

Llama al repositorio para que actualice el campo checkBoxEnabled de todoslos ítems PublicNotificationsDetail a falso (desmarcados).

3.2.13. PublicNotificationsHistory

PublicNotificationsHistoryPresenter

Esta clase se encarga de recibir y gestionar las acciones de usuario procedentes de la vista y de notificar al modelo para procesar dichas acciones de la pantalla PublicNotificationsHistory.

Métodos

fetchNotificationsHistoryFromDataBase

public void fetchNotificationsHistoryFromDataBase ()

Llama al modelo para que recupere de la base de datos Room las notificaciones de aquellos canales en los que está suscrito el usuario.

onDismiss

public void onDismiss (SubNotificationsHistoryItem item)

Indica al modelo de eliminar de la base de datos un ítem SubNotificationsHistoryItem.

setSupportActionBar

public void setSupportActionBar (ActionBar supportActionBar)

Configura el ActionBar.

PublicNotificationsHistoryModel

PublicNotificationsHistoryModel tiene como propósito recuperar de la base de datos Room las notificaciones de las noticias de los canales a los que está suscrito el usuario y de eliminar dichas notificaciones de la base de datos.

Métodos

deleteNotificationHistoryItem

public void deleteNotificationHistoryItem (SubNotificationsHistoryItem item)

Llama al repositorio para que elimine de la base de datos un ítem SubNotificationsHistoryItem.

fetchSubNotificationsHistoryListDataByUser

public void fetchSubNotificationsHistoryListDataByUser (RepositoryContract.GetSubNotificationsListCallback callback)

Recupera del repositorio todas las notificaciones de las publicaciones de los canales en los que está suscrito el usuario.

3.2.14. PrivateChannel

PrivateChannelPresenter

Esta clase se encarga de recibir y gestionar las acciones de usuario procedentes de la vista y de notificar al modelo para procesar dichas acciones de la pantalla PrivateChannel.

Métodos

fetchDataFromDataBasePrivate

public void fetchDataFromDataBasePrivate ()

Llama al modelo para recuperar de la base de datos Room los canales privados en los que está suscrito el usuario.

navigateToCreateChannel

public void navigateToCreateChannel ()

Indica a la vista de navegar a la pantalla CreateChannel para crear un canal sin contraseña.

navigateToCreateChannelWithPassword

public void navigateToCreateChannelWithPassword ()

Indica a la vista de navegar a la pantalla CreateChannel para crear un canal con contraseña.

onAlertDialogClick

public void onAlertDialogClick ()

Llama al modelo para eliminar de la base de datos el canal privado en el que se ha pulsado *unfollow*.

onItemClick

public void onItemClick (SubcategoryItem item)

Indica a la vista de navegar a la pantalla PrivatePosts junto con los estados de forma que se cargue de forma correcta el chat grupal.

onNavigationItemSelected

public void onNavigationItemSelected(MenuItem item)

Se encarga de manejar los eventos de los ítems de la barra de navegación

onUnFollowClicked

public void onUnFollowClicked (SubcategoryItem item)

Indica a la vista de mostrar un AlertDialog preguntado si realmente quiere eliminar el canal. Si la respuesta es sí entonces se llamará al método onAlertDialogClick.

setSupportActionBar

public void setSupportActionBar(ActionBar supportActionBar)

Configura el ActionBar.

setUpLayout

public void setUpLayout (NavigationView navigationView, View headerView, TextView tv_userEmail)

Carga un menú u otro dependiendo si el usuario ha entrado como registrado o invitado.

PrivateChannelModel

PrivateChannelModel tendrá como tarea principal recuperar de la base de datos Room los canales privados en los que está suscrito el usuario. Asimismo, eliminará de la base de datos aquellos canales en los que el usuario ha pulsado *unfollow*.

Métodos

deleteSubscription

public void deleteSubscription (FetchCategorias fetchCategorias)

Llama al repositorio para que elimine al usuario de un canal de Firebase, de esta forma éste no aparecería en el listado de participantes de la pantalla Participants.

fetchSubCategoryDetailList

public void fetchSubCategoryDetailList (SubcategoryItem item, RepositoryContract.GetSubCategoryDetailListCallback callback)

Llama al repositorio para que recupere los detalles de un canal privado como el título, contenido, fecha y demás para posteriormente pasarlo como estado a la pantalla PrivatePosts.

fetchSubCategoryListDataByUser

public void fetchSubCategoryListDataByUser (RepositoryContract.GetSubCategoryListCallback callback)

Llama al repositorio para que recupere la lista de los canales privados en los que está suscrito el usuario.

onUnFollowClicked

public void onUnFollowClicked (SubcategoryItem item)

Llama al repositorio para eliminar en la base de datos un ítem SubcategoryItem.

3.2.15. PrivateNotifications

PrivateNotificationsPresenter

Esta clase se encarga de recibir y gestionar las acciones de usuario procedentes de la vista y de notificar al modelo para procesar dichas acciones de la pantalla PrivateNotifications.

Métodos

fetchPrivateNotificationsFromDataBase

public void fetchPrivateNotificationsFromDataBase ()

Llama al modelo para recuperar de la base de datos Room los canales privados en los que está suscrito el usuario y de actualizar la vista.

navigateToCreateChannel

public void navigateToCreateChannel ()

Indica a la vista de navegar a la pantalla CreateChannel para crear un canal sin contraseña.

navigateToCreateChannelWithPassword

public void navigateToCreateChannelWithPassword ()

Indica a la vista de navegar a la pantalla CreateChannel para crear un canal con contraseña.

onNavigationItemSelected

public void onNavigationItemSelected (MenuItem item)

Se encarga de manejar los eventos de los ítems de la barra de navegación

onNotificationClicked

public void onNotificationClicked (SubcategoryItem item)

Indica al modelo de que se ha pulsado en el checkbox para que actualice el valor booleano de la tabla (true a false y viceversa) de la base datos.

setSupportActionBar

public void setSupportActionBar(ActionBar supportActionBar)

Configura el ActionBar.

setUpLayout

public void setUpLayout (NavigationView navigationView, View headerView, TextView tv_userEmail)

Configura el layout.

PrivateNotificationsModel

PrivateNotificationsModel se encargará de recuperar de la base de datos los canales privados en los que está suscrito el usuario para que pueda configurar las notificaciones. Por otro lado, todos los cambios que se realicen los actualizará en la base de datos Room.

Métodos

fetchSubCategoryListDataByUser

public void fetchSubCategoryListDataByUser (RepositoryContract.GetSubCategoryListCallback callback)

Llama al repositorio para que recupere la lista de los canales en los que está suscrito el usuario.

notificationChecked

public void notificationChecked (int subCategoryId, boolean isChecked)

Llama al repositorio para que actualice el ítem de la base de datos, es decir si el usuario ha pulsado en el checkbox se cambiará también en la base de datos (true → marcado y false → no marcado). En resumidas cuentas, guarda el estado del checkbox en la base de datos.

3.2.16. PrivatePosts

PrivatePostsPresenter

Esta clase se encarga de recibir y gestionar las acciones de usuario procedentes de la vista y de notificar al modelo para procesar dichas acciones de la pantalla PrivatePosts.

Métodos

autoRefreshChat

public void autoRefreshChat ()

Crea un *timer* para que llame al método fetchFirebaseChatListNoRefresh cada dos segundos para actualizar el chat (solo si hay nuevos mensajes).

fetchFirebaseChatList

public void fetchFirebaseChatList ()

Llama al modelo para que recupere de Firebase la lista de objetos ChatClass correspondiente al canal actual. Este método se llama solamente en el onCreate y cuando el usuario envía un mensaje por el chat.

fetchFirebaseChatListNoRefresh

public void fetchFirebaseChatListNoRefresh ()

Llama al modelo para que recupere de Firebase la lista de objetos ChatClass correspondiente al canal actual. Esta lista se comparará con otra lista del mismo tipo, pero temporal, y si el tamaño es mayor entonces se actualizará la vista.

onChannelInformation

public void onChannelInformation ()

Indica a la vista de navegar a la pantalla ChannelInformation. Sin embargo, tendrá que filtrar primero desde que pantalla se ha llegado a PrivatePosts para pasar los estados correctamente.

onParticipants

public void onParticipants ()

Indica a la vista de navegar a la pantalla Participants. Sin embargo, tendrá que filtrar primero desde que pantalla se ha llegado a PrivatePosts para pasar los estados correctamente.

onSendMessageClicked

public void onSendMessageClicked (String text)

Indica al modelo de insertar en la base de datos de Firebase el nuevo mensaje enviado por el chat y que genere al mismo tiempo una notificación para notificar aquellos miembros que no estén en línea.

resetEditText

public void resetEditText(EditText et_send)

Vacía el texto del EditText.

setSharedPreferences

public void setSharedPreferences (String value)

Guarda en la preferencia ACTIVE el valor value. Esto se emplea a la hora de enviar notificaciones push, es decir si el valor ACTIVE es igual a *no* entonces significa que el usuario está en la pantalla PrivatePosts (utilizando o visualizando el chat), y por tanto no recibirá una notificación si otro usuario envía un mensaje por el chat. Si por el contrario el valor de ACTIVE es igual a *yes* entonces quiere decir que el usuario no está en la pantalla PrivatePosts, puede estar en otra pantalla o simplemente tiene la App en background y entonces si recibiría la notificación al igual que WhatsApp.

setSupportActionBar

public void setSupportActionBar(ActionBar supportActionBar)

Configura el ActionBar.

PrivatePostsModel

PrivatePostsModel se ocupará por una parte de recuperar de Firebase el listado de los mensajes enviados por otros usuarios y los propios del usuario. Y, por otra parte, guardará los nuevos mensajes que se envíe en la base de datos de Firebase y generará al mismo tiempo una notificación push si los usuarios no están en la vista PrivatePosts.

Métodos

fetchFirebaseChatData

public void fetchFirebaseChatData (String category, String key, RepositoryContract.GetFirebaseChatListCallback callback)

Llama al repositorio para que recupere la lista de mensajes del canal que tiene como identificador el valor *key* y que pertenece a la categoría *category*.

sendMessage

public void sendMessage (String category, String key, String text)

Llama al repositorio para que inserte el mensaje *text* en el canal cuyo identificador es igual a *key* y que está situado en la categoría *category*.

setFirebaseNotification

public void setFireBaseNotification (String channel, String text)

Llama al repositorio para que se generé una notificación push que tenga como título el email del usuario que envió el mensaje y cuerpo *text*. El parámetro *channel* se refiere al nombre del canal.

3.2.17. MyChannels

MyChannelsPresenter

Esta clase se encarga de recibir y gestionar las acciones de usuario procedentes de la vista y de notificar al modelo para procesar dichas acciones de la pantalla MyChannels.

Métodos

checkListIsEmpty

public void checkListIsEmpty ()

Indica a la vista de comprobar si la lista está vacía.

fetchDataFromDataBase

public void fetchDataFromDataBase ()

Llama al modelo para recuperar de la base de datos Room los canales privados en los que está suscrito el usuario. Se emplea en este caso para comprobar las notificaciones push.

fetchMyFirebaseData

public void fetchMyFirebaseData ()

Llama al modelo para recuperar de la base de datos de Firebase todos los canales y mediante un iterador filtramos los canales creados por el usuario.

onAlertDialogClick

public void onAlertDialogClick ()

Llama al modelo para eliminar de la base de datos el canal privado en el que se ha pulsado *delete*.

onBackPressed

public void onBackPressed ()

Indica a la vista de navegar a la pantalla anterior.

onDeleteClicked

public void onDeleteClicked (FetchFirebaseData item)

Indica a la vista de mostrar un AlertDialog preguntando si realmente se quiere eliminar el canal.

onItemClicked

public void onItemClicked (FetchFirebaseData item)

Indica a la vista de navegar a la pantalla PrivatePosts junto con los estados de forma que se cargue de forma correcta el chat grupal.

onNotificationClicked

public void onNotificationClicked (FetchFirebaseData item)

Llama al modelo para que guarde el estado del checkbox en la base de datos Room y en Firebase.

reloadData

public void reloadData ()

Indica a la vista de refrescar la pantalla.

setSupportActionBar

public void setSupportActionBar (ActionBar supportActionBar)

Configura el ActionBar.

MyChannelsModel

MyChannelsModel se encargará principalmente de recuperar todos los canales privados de Firebase. Otras de las tareas que realiza es comprobar si hay conexión a internet, guardar el estado del checkbox en la base de datos Room y en Firebase, borrado de un canal etc.

Métodos

deleteItemInFirebase

public void deleteSubscription (FetchFirebaseData fetchFirebaseData)

Llama al repositorio para que elimine de Firebase un determinado canal.

fetchMyFirebaseData

public void fetchMyFirebaseData (RepositoryContract.GetFirebaseChannelsListCallback callback)

Llama al repositorio para que recupere todos los canales de Firebase.

fetchSubCategoryListDataByUser

public void fetchSubCategoryListDataByUser (RepositoryContract.GetSubCategoryListCallback callback)

Llama al repositorio para que recupere la lista de los canales privados en los que está suscrito el usuario.

isOnline

public boolean isOnline ()

Comprueba si tenemos conexión a internet.

notificationChecked

public void notificationChecked (int subCategoryId, boolean isChecked)

Llama al repositorio para que guarde el estado del checkbox en la base de datos Room.

onNotificationClick

public void onNotificationClick (FetchFirebaseData item)

Llama al repositorio para que actualice el estado del checkbox en Firebase.

onUnFollowClicked

public void onUnFollowClicked (SubcategoryItem item)

Llama al repositorio para eliminar en la base de datos Room un ítem SubcategoryItem.

3.2.18. CreateChannel

CreateChannelPresenter

Esta clase se encarga de recibir y gestionar las acciones de usuario procedentes de la vista y de notificar al modelo para procesar dichas acciones de la pantalla CreateChannel.

Métodos

loadLayout

public void loadLayout ()

Recupera el estado de la pantalla anterior para indicarle a la vista que cargue el layout correspondiente.

onCreateChannelClicked

public void onCreateChannelClicked (String name, String description, String spinner, ImageView imageView)

Llama al modelo para que se creé un canal en Firebase con los datos pasados por parámetro, asimismo se guardará la imagen en el *storage* de Firebase.

onCreateChannelPasswordClicked

public void onCreateChannelClicked (String name, String description, String password, String spinner, ImageView imageView)

Llama al modelo para que se creé un canal con contraseña en Firebase con los datos pasados por parámetro, asimismo se guardará la imagen en el *storage* de Firebase.

setAdapter

public void setAdapter (Spinner spinner)

Asigna el adaptador para el spinner del AlertDialog.

setSpinnerChannel

public void setSpinnerChannel ()

Guarda en la lista spinnerList del ViewModel la lista de categorías padres.

setSupportActionBar

public void setSupportActionBar (ActionBar supportActionBar)

Configura el ActionBar.

CreateChannelModel

CreateChannelModel añadirá los canales (con contraseña o sin contraseña) que se vayan a crear a Firebase. Además, si el usuario ha elegido una imagen para el canal, entonces guardará dicha imagen en el *storage* de Firebase.

Métodos

fetchSpinnerList

public void fetchSpinnerList (RepositoryContract.GetSpinnerListCallback callback)

Recupera del repositorio la lista de categorías principales para utilizarlas posteriormente en el spinner.

insertChannelToFirebase

public void insertChannelToFirebase (String category, FetchFirebaseData fetchFirebaseData)

Llama al repositorio para que inserte en la base de datos de Firebase el canal *fetchFirebaseData* perteneciente a la categoría *category*.

uploadImageToFirebase

public void uploadImageToFirebase (String firebaseImage, ImageView imageView)

Llama al repositorio para que inserte en el *storage* de Firebase la imagen *imageView* con referencia *firebaseImage*.

3.2.19. ChannelInformation

MyChannelsPresenter

Esta clase se encarga de recibir y gestionar las acciones de usuario procedentes de la vista y de notificar al modelo para procesar dichas acciones de la pantalla ChannelInformation.

Métodos

setData

public void setData ()

Actualiza las variables del ViewModel con la información almacenada en el mediador (getStateFromPreviousScreen).

setSupportActionBar

public void setSupportActionBar (ActionBar supportActionBar)

Configura el ActionBar.

3.2.20. Participants

ParticipantsPresenter

Métodos

fetchFirebaseUserList

public void fetchFirebaseUserList ()

Llama al modelo para que recupere de Firebase todos los usuarios que están suscritos a ese canal.

setSupportActionBar

public void setSupportActionBar(ActionBar supportActionBar)

Configura el ActionBar.

ParticipantsModel

ParticipantsModel se ocupará únicamente de recuperar de la base de datos de Firebase un listado con todos los usuarios que están suscritos al canal.

Métodos

fetchFirebaseUserList

public void fetchFirebaseUserList (FetchFirebaseData fetchFirebaseData, RepositoryContract.GetFirebaseUserListCallback callback)

Llama al repositorio para que recupere un listado con todos los usuarios que están suscritos al canal.