
PROYECTO DE CICLO DE G. S. DE DAM

KEEP MOVING (RED SOCIAL)



**FERNANDO MANUEL
CUADROS HORNOS
CURSO 2017/18**

INDICE

1. Resumen	3
2. Introducción	4
2.1. ¿Por qué Android?	4
2.2. ¿Qué es Firebase y cómo funciona?	5
2.3. ¿Qué es Google Maps?	8
2.4. ¿Qué es Material Design?	8
2.5. ¿Qué es Git?	9
2.6. ¿Qué es GitLab?	10
3. Objetivos	11
3.1. Fase inicial	11
3.2. Tareas iniciales	13
3.3. Modificaciones en los objetivos a lo largo del proyecto	15
3.4. Organización de los datos en Firebase	17
4. Hipótesis de Trabajo	18
5. Material	20
5.1. Hardware	20
5.2. Software	21
5.3. Configuraciones del hardware y el software	22
6. Método	29
6.1. Creación de Actividad principal, implementación de las librerías de Material Design y Creación de los modelos	29
6.2. Gestión de Usuarios	33
6.3. Gestión de Quedadas	51
6.4. Mejoras en la funcionalidad y el diseño	81
6.5. Exportando la Aplicación	82
7. Tiempo de Ejecución	83
8. Resultados	86
9. Conclusiones	90
10. Bibliografía	90

1. RESUMEN

Desarrollo de una aplicación nativa para Android que nos permite organizar quedadas para practicar cualquier deporte. Estas quedadas se publican con un número de plazas limitado y con todos los datos necesarios para que el usuario interesado pueda asistir a esa quedada, una vez publicada, los usuarios podrán apuntarse solicitando una reserva de plazas que el usuario autor podrá confirmar o rechazar.

La aplicación también cuenta con un sistema de logueo y registro de usuarios en el cual el usuario deberá registrarse y posteriormente loguearse para poder acceder a las quedadas publicadas.

Una vez logueado el usuario también podrá crear sus propias quedadas, las cuales se podrán modificar posteriormente, o eliminarse cuando el usuario lo crea oportuno.

- Objetivos iniciales:

-Objetivos a nivel de desarrollo:

- Implementar un sistema de gestión de usuarios en el cual se puedan dar de alta nuevos usuarios y loguearse para acceder a sus datos.

- Implementar un sistema de gestión de quedadas en el cual el usuario logueado pueda dar de alta nuevas quedadas, así como modificar o eliminar sus quedadas.

- Implementar un sistema de peticiones en el cual los usuarios puedan enviar solicitudes para participar en quedadas, las cuales el usuario autor podrá aceptar o rechazar.

- Implementar la API de Firebase para la autenticación de usuarios, el almacenamiento de datos y el almacenamiento de datos multimedia como imágenes.

- Implementar la API de Google Maps para la ubicación de las quedadas.

- Implementar Material Design para obtener una interfaz más atractiva.

-Objetivos a nivel personal:

- Promover un estilo de vida saludable y la práctica de deporte.

- Realizar una red social completa empaquetada en una aplicación nativa en Android.

- Obtener un beneficio tras una posible comercialización de la aplicación en Google Play.

2. INTRODUCCIÓN

Como ya se ha explicado anteriormente, la intención de este proyecto es crear una aplicación nativa en Android la cual consiste en una red social diseñada con el fin de organizar quedadas para practicar cualquier deporte.

Esta tarea, como es lógico, requiere del uso de APIs y librerías externas que nos permitan almacenar los datos de nuestra aplicación y además obtener como resultado una aplicación plenamente funcional y elegante, para obtener este resultado he basado el desarrollo de mi aplicación en el uso de Android combinado con las API s de Google Firebase, Google Maps y Material Design.

En cuanto a la parte de organización, control de versiones y tareas he utilizado la herramienta Git combinada con el repositorio remoto de GitLab tratando de aprovechar al máximo todas las posibilidades de que esta plataforma nos ofrece.

2.1. ¿POR QUÉ ANDROID?



En los últimos años el uso del sistema operativo android se ha incrementado en niveles inimaginables, dejando fuera del mercado a sistemas operativos tan importantes como Windows phone o Symbian y en la actualidad es, junto con IOS, uno de los sistemas operativos móviles más importantes del mundo gracias a su amplia gama de posibilidades y personalización, además de las posibilidades que nos ofrece Google play siendo el mayor mercado de aplicaciones móviles del mundo en 2018.

Todos estos factores junto con los conocimientos de los que ya disponía sobre la programación nativa en Android OS han hecho que me termine decantando por basar este proyecto en una aplicación nativa para Android.

2.2. ¿QUÉ ES FIREBASE Y CÓMO FUNCIONA?



Firebase es una plataforma gratuita de desarrollo web y desarrollo móvil creada por Google y lanzada en 2011, que nos ofrece una serie de funciones y utilidades como Authentication, Database, Storage o Hosting, las cuales nos permiten guardar y sincronizar los datos de nuestras aplicaciones (Android, IOS o Web) en tiempo real con una gran facilidad.

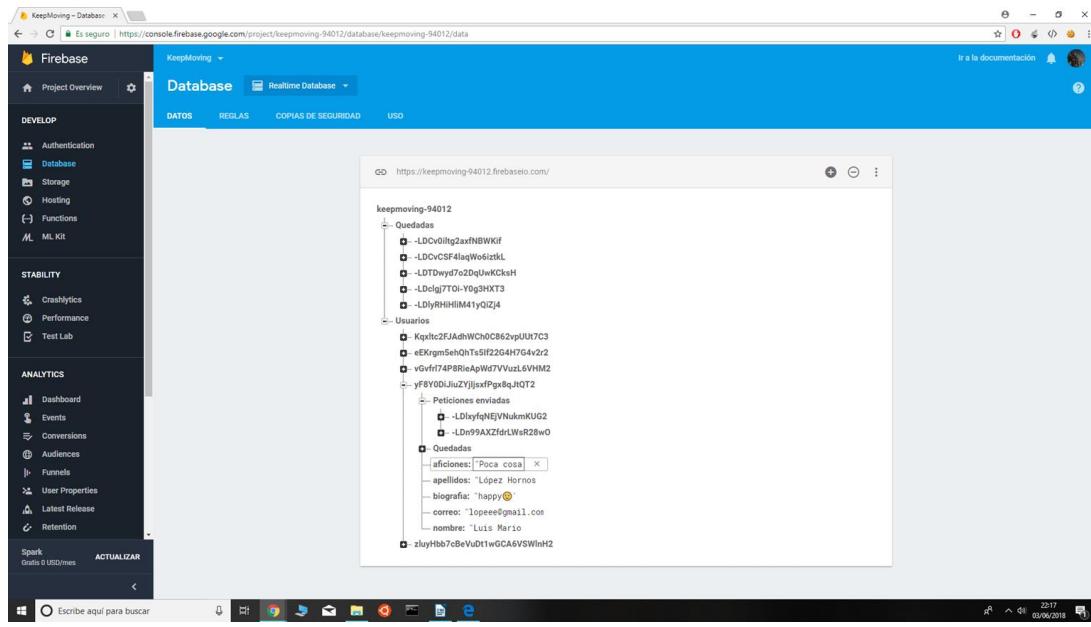
- Firebase Authentication:

Esta utilidad nos facilita las tareas de gestión de Usuarios en nuestras aplicaciones o páginas web, para ello utiliza un sistema de alta y baja de usuarios e inicio y cierre de sesión de estos con una amplia gama de posibilidades de inicio de sesión y una impecable gestión y recuperación de contraseñas:

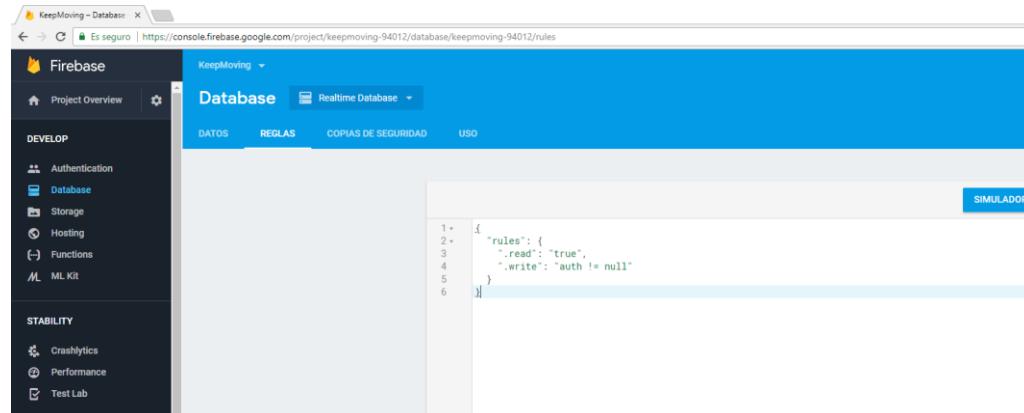
The screenshot shows the Firebase Authentication interface in the Firebase console. The left sidebar includes sections for DEVELOP (Authentication, Database, Storage, Hosting, Functions, ML Kit), STABILITY (Crashlytics, Performance, Test Lab), and ANALYTICS (Dashboard, Events, Conversions, Audiences, Funnels, User Properties, Latest Release, Retention). The main content area is titled 'Authentication' and shows the 'USUARIOS' tab selected. It displays a table of sign-in providers: Correo electrónico/contraseña (Enabled), Teléfono (Disabled), Google (Disabled), Play Juegos (Disabled), Facebook (Disabled), Twitter (Disabled), GitHub (Disabled), and Androíno (Enabled). Below this is a section for 'Dominios autorizados' (Authorized domains) with entries for localhost (Default) and keepmoving-94012.firebaseioapp.com (Default). A blue 'AÑadir dominio' (Add domain) button is visible.

- Firebase Database:

Esta utilidad nos ofrece un servicio de base de datos no relacional y a tiempo real, es decir, en la cuál los datos se actualizan al momento en nuestras aplicaciones cuando nosotros hacemos alguna modificación de estos, si necesidad de refrescar los datos constantemente, esto se consigue mediante escuchadores que se activan automáticamente cuando nuestros datos son modificados ya sea desde la aplicación o desde la propia consola de Firebase.

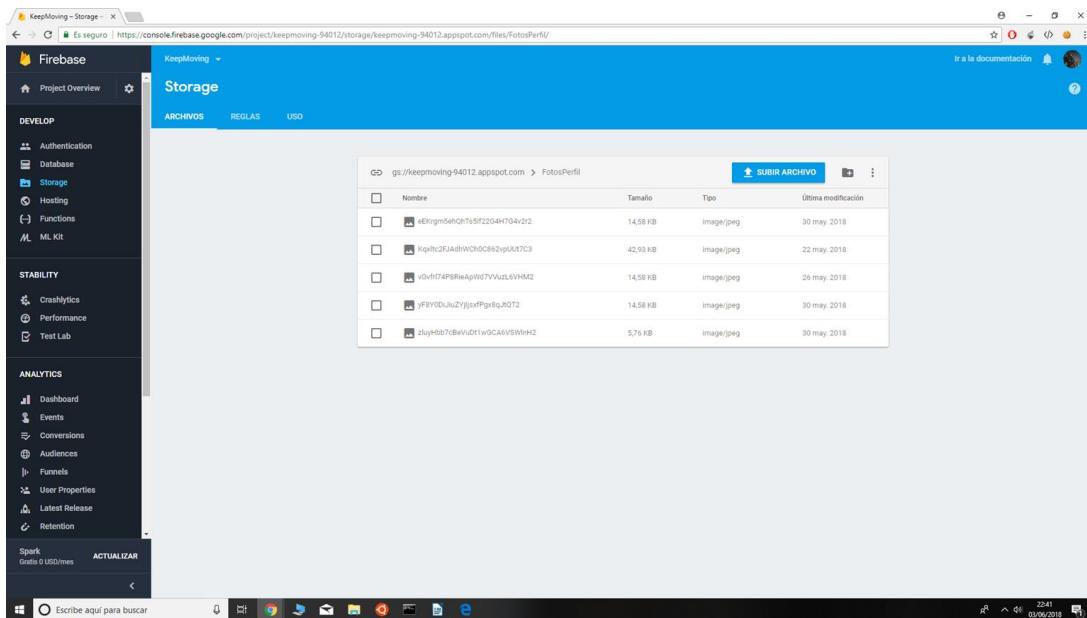


Además Firebase Database nos permite crear nuestras propias reglas de subida o descarga de datos desde la propia consola, lo que nos ofece una mayor seguridad y un mayor control sobre nuestra base de datos.



- Firebase Storage:

Esta utilidad nos permite subir y almacenar archivos en la nube para después usarlos en nuestras aplicaciones y obtenerlos de forma sencilla sin necesidad de almacenarlos directamente en la base de datos, para acceder a nuestro storage solo tendremos que usar nuestro usuario y la referencia donde están almacenados los datos, en mi caso es FotosPerfil:



The screenshot shows the Firebase Storage console for the 'KeepMoving' project. The left sidebar includes sections for Authentication, Database, Storage, Hosting, Functions, ML Kit, Crashlytics, Performance, and Test Lab. The main area is titled 'Storage' and shows a list of files under the reference 'gs://keepmoving-94012.appspot.com/FotosPerfil'. The table lists six files, each with a preview thumbnail, name, size, type, and last modified date. A 'SUBIR ARCHIVO' button is visible at the top right of the list.

Nombre	Tamaño	Tipo	Última modificación
e0rgm5eh0hT5f220A4t704vZ2	14,58 KB	image/jpeg	30 may. 2018
Kqrlfc2fJaJdhWWh0c08d2y0t8t7C3	42,93 KB	image/jpeg	22 may. 2018
v0VfI74P8RieApWd7VVzL6VhM2	14,58 KB	image/jpeg	29 may. 2018
y8Y0DUJu2YjjnxFPg8q,ROT2	14,58 KB	image/jpeg	30 may. 2018
zlojHbd7BeVuDt1wGCA6VSWlnH2	5,76 KB	image/jpeg	30 may. 2018

Además, al igual que Firebase Database, Firebase Storage nos ofrece la posibilidad de configurar nuestras propias reglas para mejorar la seguridad y el control sobre nuestros datos.

2.3. ¿QUÉ ES GOOGLE MAPS?



Google Maps API es un servidor de aplicaciones creado por Google que nos ofrece una serie de aplicaciones para trabajar con mapas, fotografías por satélite, ubicaciones y rutas en nuestras propias aplicaciones y páginas webs.

El uso de esta API está basado en el uso de coordenadas de ubicación y posicionamiento (longitud y latitud), las cuales nos permiten localizar cualquier parte del mundo y plasmarla en un mapa.

Además, nos permite plasmar nuestra propia posición a tiempo real utilizando los sensores de posicionamiento de nuestros dispositivos combinados con sus mapas y librerías, así como establecer rutas desde nuestra posición actual hasta cualquier ubicación a la que queramos ir.

2.4. ¿QUÉ ES MATERIAL DESIGN?



Material Design es un concepto o filosofía de diseño creada por Google en 2014 y que es válido tanto para el desarrollo en Android como para el desarrollo Web.

Material Design recibe su nombre por estar basado en objetos materiales reales, es decir, piezas colocadas en un espacio con un movimiento o animaciones determinadas, además de un gran nivel de detalle en cuanto a profundidad, bordes, sombras y colores, empaquetando todo esto en una serie de librerías gratuitas y fácilmente implementables.

2.5. ¿QUÉ ES GIT?



Es un software de control de versiones pensado para mejorar la eficiencia y la confiabilidad del mantenimiento de versiones de aplicaciones cuando éstas tienen un gran número de archivos de código fuente. Su propósito es llevar registro de los cambios en archivos de computadora y coordinar el trabajo que varias personas realizan sobre archivos compartido.

Git nos permite crear varias ramas (Branch) en nuestro proyecto con el fin de que no se creen conflictos entre los diferentes usuarios y roles que acceden y actualizan mediante commits el código de nuestro proyecto.

Conceptos básicos de Git:

Commit	Se guardan los cambios realizados en el proyecto de nuestro repositorio con un mensaje que contiene información sobre los últimos cambios
Add	Sirve para añadir ficheros a nuestro al proyecto, el cual tenemos almacenado en nuestro repositorio Git
Push	Actualiza nuestro proyecto en nuestro servidor remoto de Git
Clone	Copia todo lo que hay en el repositorio remoto de tu proyecto a tu repositorio local
Pull	Sirve para sincronizar nuestro repositorio local con los datos que hay en nuestro repositorio Git remoto
Branch	Es una copia de nuestro proyecto en nuestro repositorio Git la cual hace referencia a una versión de nuestro proyecto. A la hora de hacer cambios se pueden configurar para que solo se guarden en una de las ramas indicándole la rama al hacer un push.

2.6. ¿QUÉ ES GITLAB?



Es un servicio web de control de versiones y desarrollo de software colaborativo basado en Git. Además de gestor de repositorios, el servicio ofrece también alojamiento de wikis, un sistema de control de tareas y un sistema de seguimiento de errores, todo ello publicado bajo una Licencia de código abierto.

Para facilitar la gestión de nuestros proyectos, GitLab nos ofrece un sistema de gestión de tareas basado en el uso de milestones, que a su vez pueden, o no, contener Issues que hacen referencia a las tareas pendientes en nuestro proyecto. Para esta gestión además nos ofrece una interfaz basada los diferentes estados que puede tener una tarea (ToDo, Doing, Closed).

Conceptos básicos de GitLab:

Milestone	Traducido al español como hito, hace referencia a un evento significativo que ocurre durante el desarrollo de nuestro proyecto y que generalmente coincide con la terminación de un entregable principal del proyecto o una versión funcional del mismo. Los milestones o hitos están compuestos de tareas más pequeñas o issues.
Issues	Es una tarea u objetivo que queremos implementar en nuestro proyecto, normalmente esta comprendida dentro de un Milestone junto con otras muchas tareas.
Etiqueta	Hace referencia al estado de una tarea o issue, puede estar en tres fases, to do (por hacer), doing (haciéndose) o Closed (cerrada o terminada)

3. OBJETIVOS

3.1. FASE INICIAL

Al inicio del desarrollo de la Aplicación se partía de una idea inicial y unos bocetos iniciales, los cuales son los siguientes:

Documento inicial del proyecto:

The screenshot shows a Microsoft Word document window with the title bar 'doc: Bloc de notas'. The content of the document is as follows:

Archivo Edición Formato Ver Ayuda
Nombre: Keep Moving

BBDD: Firebase

Interfaz: Material, Cardviews

Patrón: repositorio, Modelo Vista Presentador

Adicionesles: Maps, Chat, Auth (usuarios)

Colecciones: Usuarios

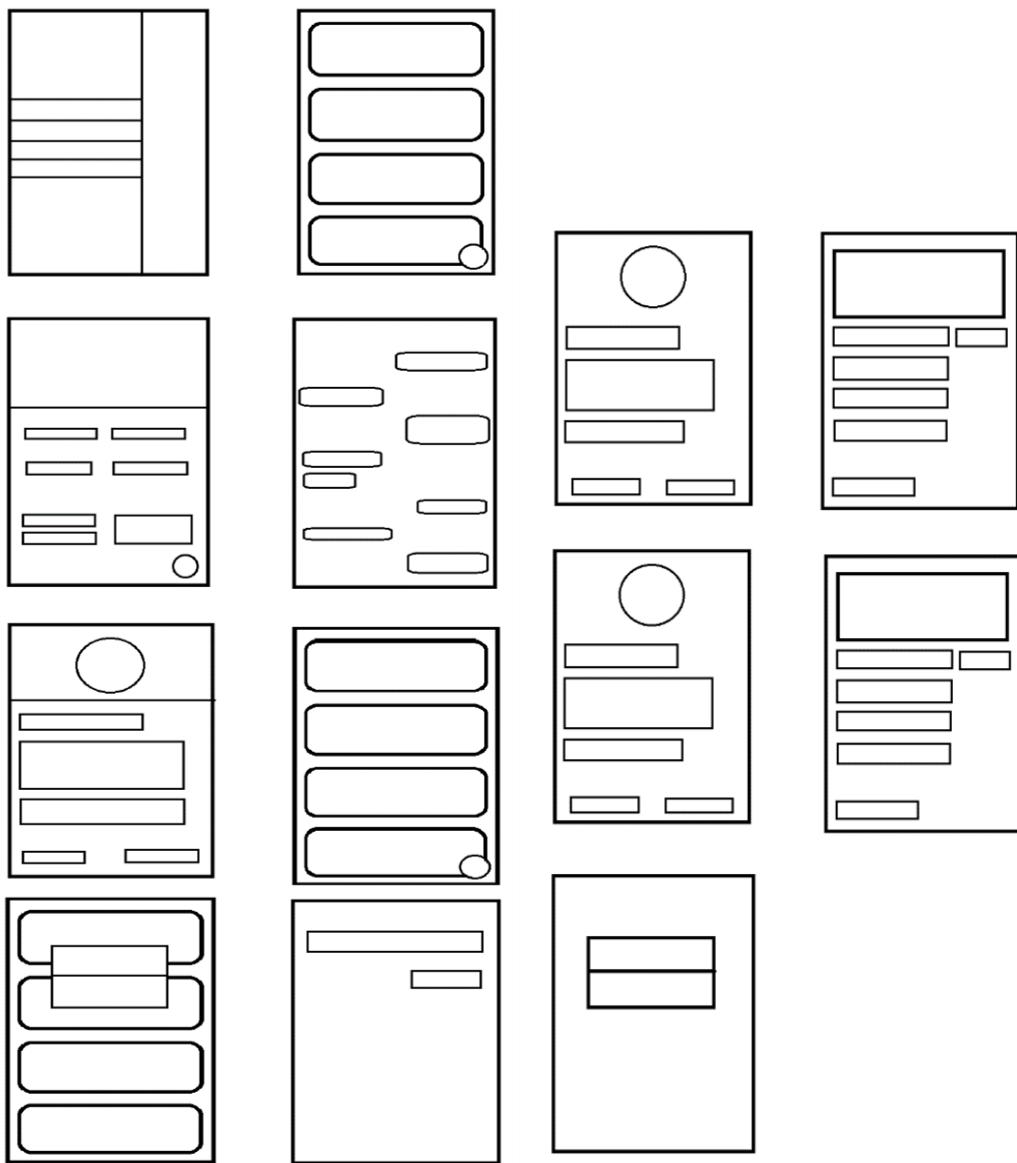
```
    |  
    |__ Quedadas  
    |__ Chats  
        |__ Mensajes
```

Descripción:
Aplicación que nos permita publicar quedadas deportivas en un lugar y con unas plazas y en la cual los usuarios se puedan apuntar, ver detalle o hablar con el publicante.

Modelos:

```
/*Principal*/  
-Usuario: nombre, foto, descripción, deportes que practica, valoración?, contacto?  
-Quedada: fecha publicación, fecha , hora, lugar, ubicación, deporte, plazas, autor(usuario)  
  
/*Chat ?? --> aún no se como hacerlo */  
-Chat: autor(nombre), autor(id)?, receptor (nombre), receptor (id), fecha ,mensajes[]  
-mensaje : mensaje , fecha
```

Bocetos iniciales del proyecto:



El proyecto constaría de las siguientes pantallas:

- Navigation Drawer
- Pantalla detalle quedada
- Pantalla perfil
- Pantalla con listado de mis quedadas y menú de edición/borrado
- Listado con quedadas publicadas
- Pantalla Chat
- Pantalla editar perfil
- Pantalla registro con datos
- Pantalla inicio de sesión
- Pantalla editar quedada
 - Pantalla crear quedada
- Pantalla de petición quedada

Una vez que ya se tenía una idea y unos bocetos, el siguiente paso fue crear un repositorio del proyecto en GitLab en el cual se crearon varios Milestones , con sus correspondientes Tareas (Issues) y sus correspondientes plazos y fechas de entrega al cuál iba subiendo las actualizaciones del proyecto con las tareas que iba completando.

3.2. TAREAS INICIALES

Ahora que ya sabemos como funciona Git y GiLab , voy a explicar los objetivos y las tareas en las que he dividido el desarrollo de mi proyecto junto con los plazos de entrega o finalización de las tareas:

- Milestone 1 (Del **22/Marzo/2018** hasta **22/abril/2018**):

En este hilo se pretende crear una primera versión funcional de la Aplicación que tuviera una interfaz para crear usuarios y mostrar el perfil con el detalle.

Este Milestone está compuesto de las siguientes tareas:

- Se crea la vista de la pantalla de loggin
- Se crea la vista de la pantalla de detalle de usuario
- Se instancia nuestra base de datos Firebase
- Se loguean los usuarios desde la pantalla de loggin
- Se muestra el detalle del usuario
- Se crea el Navigation Drawer
- Se crean las pantallas de registro
- Los usuarios se pueden registrar con sus datos
- Se accede al Storage para seleccionar la foto de perfil
- Se implementa Google Maps

- Milestone 2 (Del **27/Abril/2018** hasta **20/Mayo/2018**):

En este hilo se pretende obtener una versión en la que se gestionan las quedadas implementando que el usuario pueda crear ver y borrar nuevas quedadas, además de incluir ubicación.

Este Milestone está compuesto de las siguientes tareas:

- Pantalla para añadir quedadas
- Se implementa Ubicación
- Se añaden las quedadas a Firebase
- Se permite la modificación y el borrado de quedadas

- Milestone 3 (Del **6/Mayo/2018** hasta **22/Mayo/2018**):

En este hilo se pretende obtener una versión en la que visualicen los listados con las quedadas publicadas.

Este Milestone está compuesto de las siguientes tareas:

- Se visualizan las quedadas del usuario
- Se visualizan todas las quedadas publicadas por cualquier usuario

- Milestone 4 (Del **15/Mayo/2018** hasta **3/Junio/2018**):

En este hilo se crea listado de peticiones y solicitudes para que los usuarios se apunten a las quedadas reservando plazas.

Este Milestone está compuesto de las siguientes tareas:

- Se crea la pantalla desde la cual se publica una solicitud
- Se crean las solicitudes en Firebase correctamente
- Se muestra la lista con tus solicitudes
- Se muestra la lista con las peticiones a tus quedadas desde la cual el autor puede confirmar la reserva o descartarla
- Se cambia el estado de la solicitud y se visualizan los diferentes estados correctamente

- Milestone 5 (Del **22/Mayo/2018** hasta **20/Junio/2018**):

En este hilo se pretende mejorar la aplicación puliendo el diseño y añadiendo las ultimas utilidades y mejorando la funcionalidad de nuestra aplicación.

Este Milestone está compuesto de las siguientes tareas:

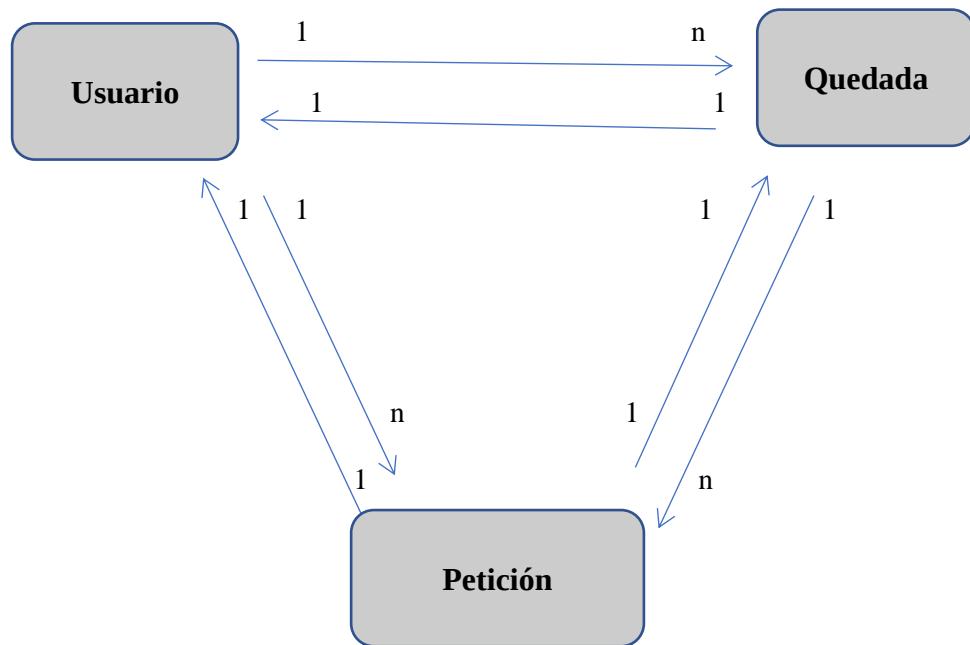
- Se crea un buscador en la lista para filtrar las quedadas
- Se visualiza el perfil de los usuarios al click-ar en su foto
- Se visualizan los detalles del usuario en la petición y en la quedada
- Se crea una pantalla con ayuda sobre la app
- Se mejora el diseño de la app
- Se mejora el control sobre los edit texts
- Se controla la conexión a internet
- El usuario puede borrar su cuenta
- Se implementa la app en varios idiomas
- Se implementa la verificación con el correo electrónico
- Se implementa la logueo con Facebook
- Se controla la fecha en las quedadas de la lista, mostrando solo las que están aun pendientes
- Se controla el Toolbar en cada pantalla
- Se implementa el cambio de contraseña
- Se comenta el código
- Se implementa la logueo con Facebook

3.3. MODIFICACIONES DE LOS OBJETIVOS A LO LARGO DEL DESARROLLO

Conforme al desarrollo de la aplicación fue avanzando y se completaban las tareas, en función al tiempo y, con el fin de obtener una aplicación lo más eficaz, funcional y productiva posible se fueron haciendo algunas modificaciones con respecto a la idea inicial y a las tareas iniciales.

Debido al tiempo del que disponía para terminar el Milestone 4 decidi cambiar el chat inicial plateado, por un sistema de solicitudes y peticiones en el cual un usuario enviara una solicitud a una quedada para que luego el autor de esa quedada la aceptase o rechazase.

Es decir:



Una vez planteado este nuevo esquema se creó un nuevo boceto del sistema de peticiones:

BOCETO SOLICITUD/PETICION QUEDADA



También se modificó la estructura de la base de datos dividiendo está en dos grupos principales, quedadas y usuarios y dentro de los usuarios tendríamos las quedadas del usuario, las peticiones enviadas y las peticiones recibidas, es decir, a la hora de subir una quedada esta se añadiría a una colección de quedadas con todas las quedadas, y a su vez, la misma quedada se añadiría dentro del usuario creador en la colección de quedadas de este usuario:

BBDD:

- Quedadas
- Usuarios
 - _Quedadas usuario
 - _Peticiones enviadas
 - _Peticiones recibidas

Los datos de cada uno de los modelos también cambiaron conforme avanzaba el desarrollo debido a las necesidades que iban surgiendo, además se quedaron nuevos modelos con respecto a los propuestos en la fase inicial como “Petición Quedada” o “Petición quedada recibida”:

Usuario	Quedada	Petición Quedada	Petición quedada recibida
correo; contraseña; nombre; apellidos; biografía; aficiones; foto;	id; autor; autor_uid; lugar; fecha; hora; deporte; info; plazas; longitud; latitud;	id_peticion; id; autor; autor_uid; lugar; fecha; hora; deporte; info; plazas; longitud; latitud; autor_peticion; autor_peticion_nombre; num_plazas_solicitadas; estado;	id_peticion; id; autor; autor_uid; lugar; fecha; hora; deporte; info; plazas; longitud; latitud; autor_peticion; autor_peticion_nombre; num_plazas_solicitadas; estado; foto;

Con estos cambios, finalmente la base de datos quedaría de la siguiente forma:

```

keepmoving-94012
  - Quedadas
    - LERGmbt3yqjHtDok7VW
    - LERJ1okFnhoZRWpHdI
    - LEUO_SHP-9GuJzxKnJ
    - LEURE_9kzRKPhnPjveS
  - Usuarios
    - AG1wYXSe6NOFXQQGczusXOFacWjI
      - Peticiones enviadas
      - Peticiones recibidas
        - aficiones: "Fútbol, Fitness, Senderismo, Running"
        - apellidos: "Cuadros Hornos"
        - biografia: "Nunquam Desperare"
        - correo: "f.manu_97@hotmail.co"
        - nombre: "Fernando M"
      - 53iHHRGwvMRQatLgw9pmAL8mn2
      - WHjGE5JGSHMbzxmz9SrOcxISQOT2
      - Wvc5tlj3u0Tp4vQJIKr263eRNl2
      - F2zAIElyJdg0Vn1etT3JzPvB52
      - j0VS0BL0ziLUuJOQ0diWYQSuKC2
      - rR NzWUSA4CgRNLN4bllhMr3hUud2

```

3.4. ORGANIZACIÓN DE LOS DATOS EN FIREBASE

Refiriéndome a la organización de los datos en Firebase, traté de aprovechar al máximo todas las posibilidades que este repositorio remoto nos ofrece:

- **Firebase Database:** Aquí es donde se almacenarán los datos de las quedadas, peticiones de quedadas, peticiones de quedadas recibidas y usuarios que se van subiendo, excepto la imagen y los datos de la autenticación los cuales se almacenan en otro el storage y la autenticación de Firebase.

Cuando se almacenan los datos de los usuarios como las aficiones, nombre, apellidos o biografía en la base de datos se almacenan usando como referencia el uid del usuario para conectarlos con el usuario autenticado y la referencia a la foto de perfil del usuario en el Storage de Firebase.

- **Firebase Storage:** Aquí es donde se almacenan las imágenes de perfil de los usuarios con el uid del usuario como referencia.

- **Firebase Autenticación:** Aquí es donde se almacena el email y la contraseña de los usuarios que se van loqueando en la aplicación y guardándose como usuarios registrados.

4. HIPOTESIS DE TRABAJO

A la hora de desarrollar el código se ha usado el conocido patrón de desarrollo **Modelo-Vista-Presentador** para mejorar la capacidad de reciclaje de nuestro código y el **patrón repositorio** el cual crea una clase repositorio que se conecta con la API, los modelos y la conexión a la base de datos y almacena los datos para luego enviárselos al presentador que se los enviará a su vez a la interfaz, el MVP se divide de la siguiente forma:

- Modelo:

Es donde se almacena la lógica de la aplicación, los modelos y la conexión a la base de datos, en mi caso contiene la siguiente información:

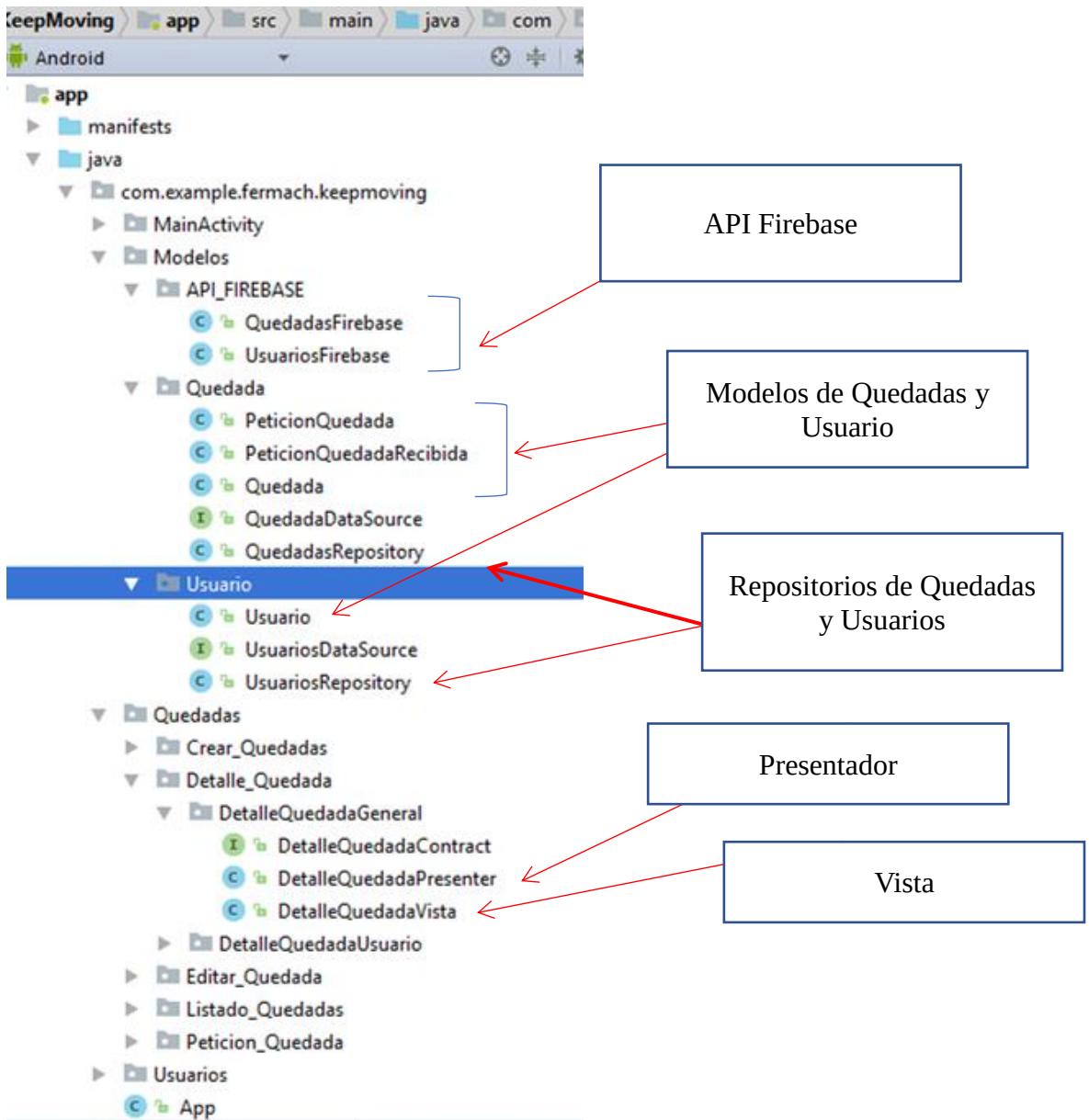
- Los modelos POJO de los objetos que tendrá mi aplicación, los cuales son Quedada, Usuario, Petición quedada y Petición quedada recibida,
- La API que se conecta con el repositorio y la base de datos. En mi caso la he dividido en API de quedadas y API de usuarios
- El Repositorio que almacena los datos y es el que comunica la API con el Presentador.
- La interfaz que contiene todos los métodos que implementara la API.

- Vista:

Está compuesta por la interfaz y los controles que implementa nuestra aplicación y que verá el usuario a través de los distintos fragmentos y pantallas.

- Presentador:

Es el encargado de comunicar el repositorio de datos con la vista de la aplicación, en mi caso utiliza una interfaz intermedia que comunica la vista de la interfaz con el presentador para agilizar la comunicación y obtener una versión lo más clara posible de nuestro código.



5. MATERIAL EMPLEADO

5.1. HARDWARE

-Ordenador Personal (Desarrollo):

- Windows 10 (Sistema Operativo)
- Procesador: AMD FX-8350 4.0Ghz 8X
- RAM: G.Skill Ripjaws X DDR3 1600 PC3-12800 8GB 2x4GB CL9
- GPU: MSI GeForce GTX 970 Gaming OC 4GB GDDR5
- Placa Base: MSI 970 Gaming
- Disco Duro: WD Blue 1TB SATA3
- Disco SSD: [Samsung 850 Evo SSD Series 250GB S..ATA3](#)
- Periféricos: Teclado, monitor, ratón, auriculares...



-Mi Smartphone (Testing) :

- Samsung Galaxy A5 2017

- SO: Android 8.0
- Procesador: Exynos7880 8 nucleos 1,9Ghz
- RAM: 3GB
- Memoria Interna: 32 GB
- Pantalla: 5,2 "



5.2. SOFTWARE

- JDK Java (Desarrollo)

- Git (Control de versiones)
- Versión: 2.14.2.2 x64

- Android Studio (Desarrollo)
- Versión: 3.0.1 x64

- Google Chrome

- Navegador Web
- Búsqueda de información
- Acceso al repositorio de Firebase
- Acceso al repositorio de GitLab
- Acceso a la consola de Google

- PicsArt Studio PC version (Edición y retoque de imágenes):

5.3. CONFIGURACIÓN DEL HARDWARE Y EL SOFTWARE

-Habilitar modo de desarrollo y emulación en mi Smartphone

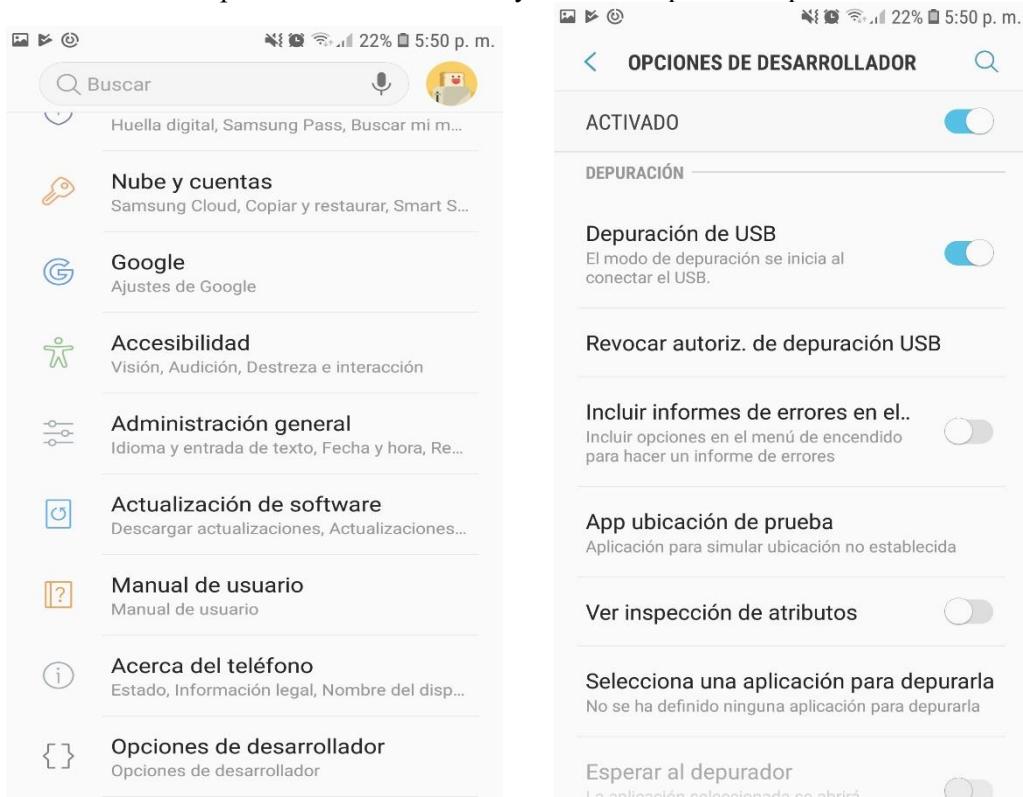
Para poder utilizar mi propio teléfono móvil como emulador con el que testear mi aplicación es necesario configurarlo en modo desarrollador y activar la depuración USB. Para ello seguir los siguientes pasos:

1. Debes dirigirte a los ajustes de tu teléfono > Acerca del Teléfono.
2. Clickar sobre el Número de Compilación repetidas veces hasta que salga un mensaje que diga que has activado el modo de desarrollador, en mi caso ya lo tenía activado.

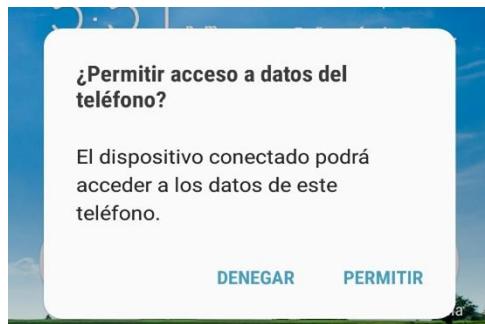


3. A continuación se te creará un nuevo hijo en el menú de ajustes que será “Opciones de desarrollador”

4. Clicar sobre “Opciones de desarrollador y activas la opción “Depuración USB”



5. Conectas el teléfono al PC y le das a permitir acceso cuando salga el siguiente mensaje:



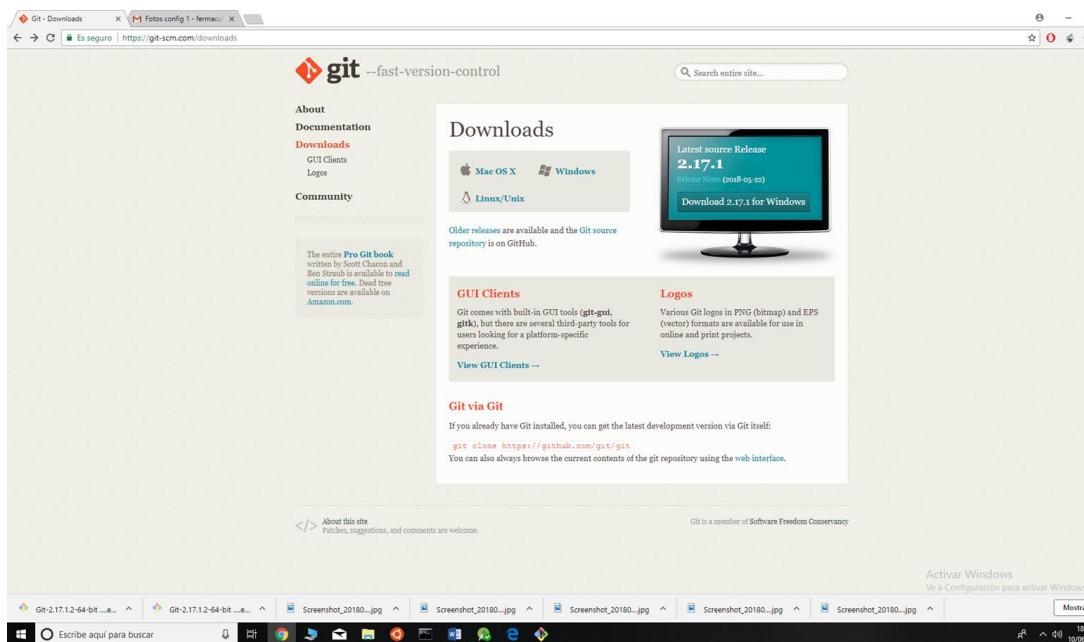
- Instalación de Git

Debido a que en mi caso estoy utilizando un sistema operativo Windows 10, los pasos de instalación serán adaptados para este sistema.

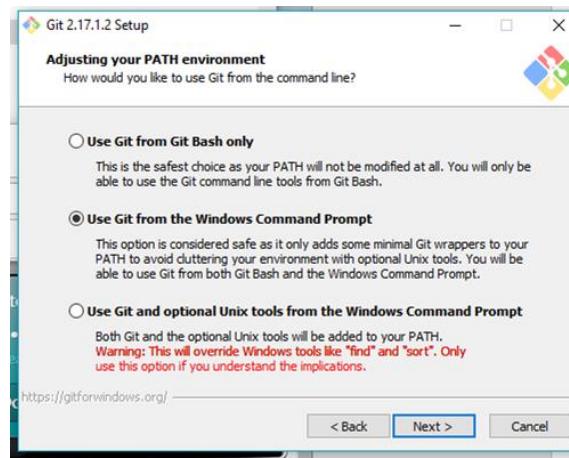
Para instalar Git en tu ordenador debes realizar los siguientes pasos:

1. Debes dirigirte a la página web de Git y descargar el .exe de la última versión utilizando el siguiente enlace:

<https://git-scm.com/downloads>



2. Una vez descargado tan solo debes ejecutar el exe y seguir las instrucciones de instalación y automáticamente se te instalará el software de control de versiones Git y se te añadirá al PATH de Windows para que puedas ejecutar sus comandos desde cualquier ruta en la consola del sistema.



Para añadir algún cambio de nuestro proyecto a nuestro repositorio remoto de Git, que en este caso sería GitLab, se haría realizando los siguientes pasos:

1. Una vez instalado Git y añadido al PATH (Variables de entorno) de nuestro sistema, abrimos una consola o terminal en nuestro pc y nos dirigimos a la ruta donde tenemos nuestro proyecto.
 2. Si es nuestro primer commit debemos inicializar un repositorio de git en la ruta de nuestro proyecto, para ello utilizamos el comando “git init”
 3. Añadimos los archivos de nuestro proyecto a nuestro repositorio git utilizando el comando:
 - git add . -> para añadir todos los archivos
 - git add “ruta del archivo que deseamos añadir” -> para añadir un solo archivo
1. Guardamos los cambios con un mensaje utilizando el comando:
 - git commit -m “mi mensaje”
 1. Añadimos los cambios a nuestro repositorio remoto, que en este caso sería GitLab indicándole la rama:
 - git push “Ruta de mi repositorio git remoto” “rama”

En mi caso seria

```
git push https://gitlab.com/Fermach/KeepMoving.git master
```

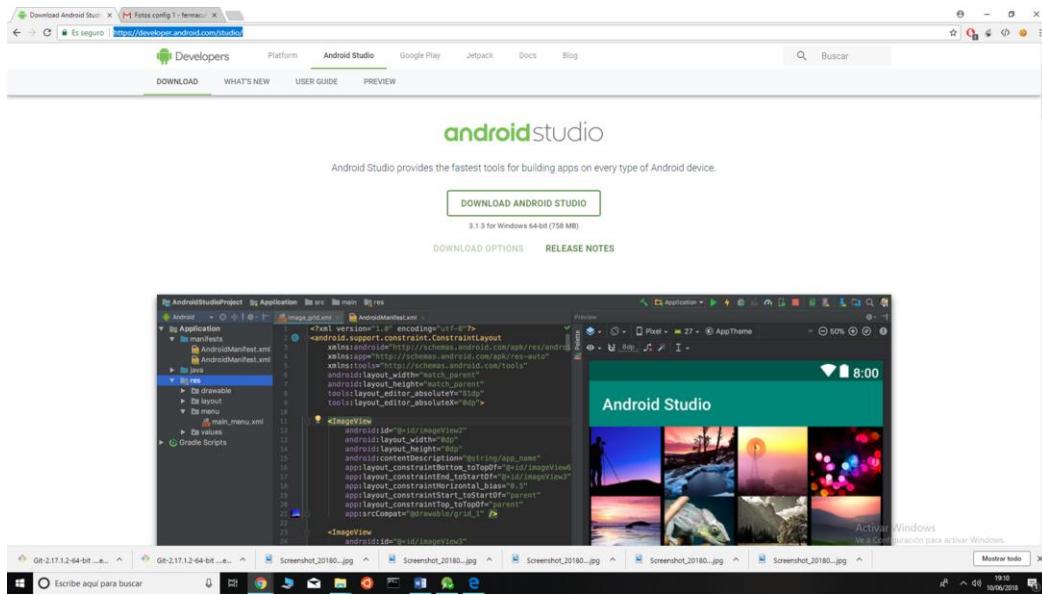
Aquí si no tenéis una sesión iniciada os pedirá que introduzcáis vuestro usuario y contraseña.

- Instalación de Android Studio

Para instalar Android Studio en tu ordenador debes realizar los siguientes pasos:

1. Debes dirigirte a la página web de Android Studio y descargar el .exe de la última versión utilizando el siguiente enlace:

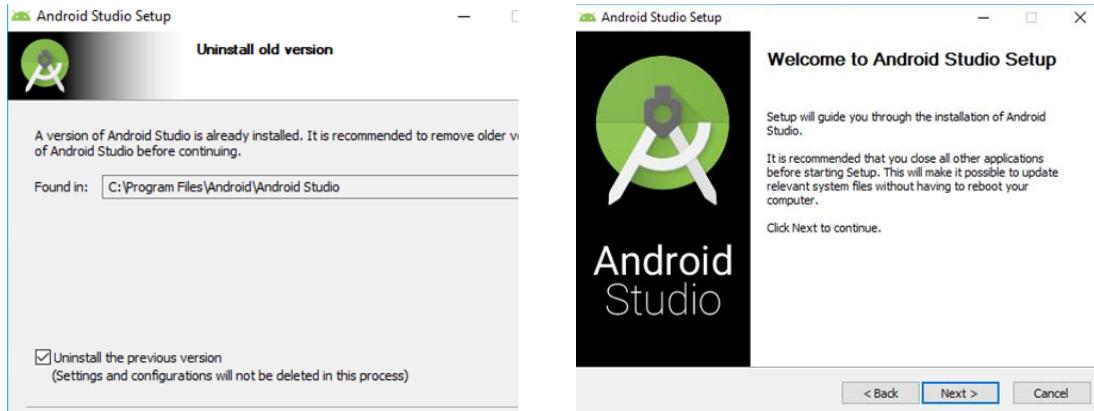
<https://developer.android.com/studio/>



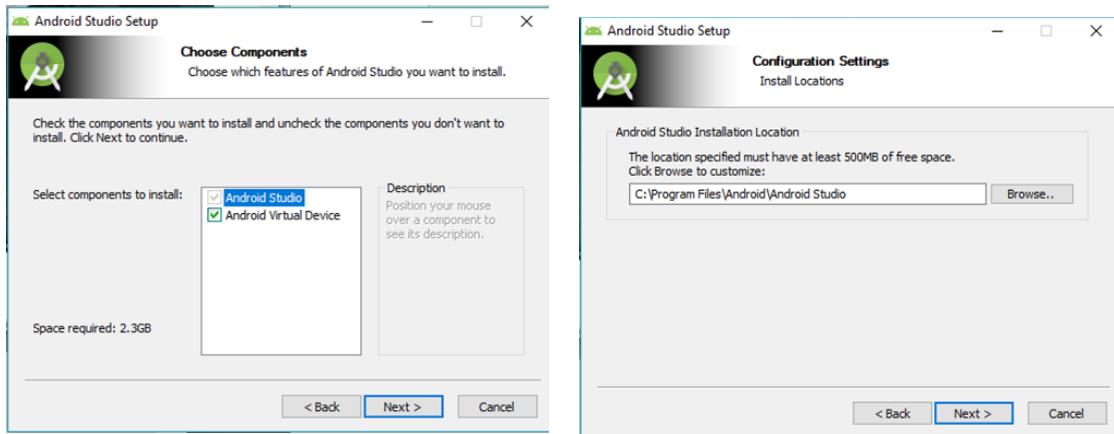
El archivo ejecutable de Android Studio tiene un peso considerable, por lo que es recomendable que uséis una buena conexión a internet.

2. Ejecutamos el archivo .exe y esto nos abrirá el asistente de instalación.

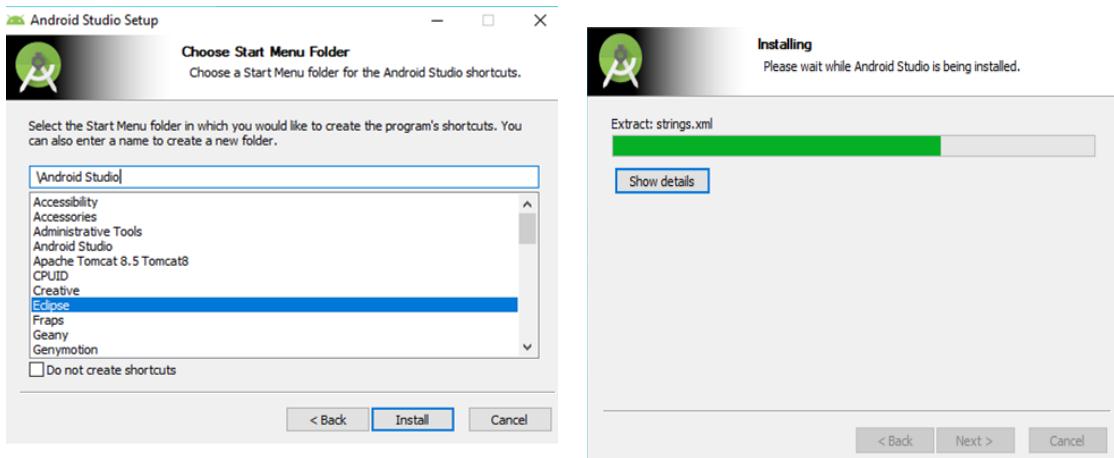
3. Seleccionamos la ruta de instalación



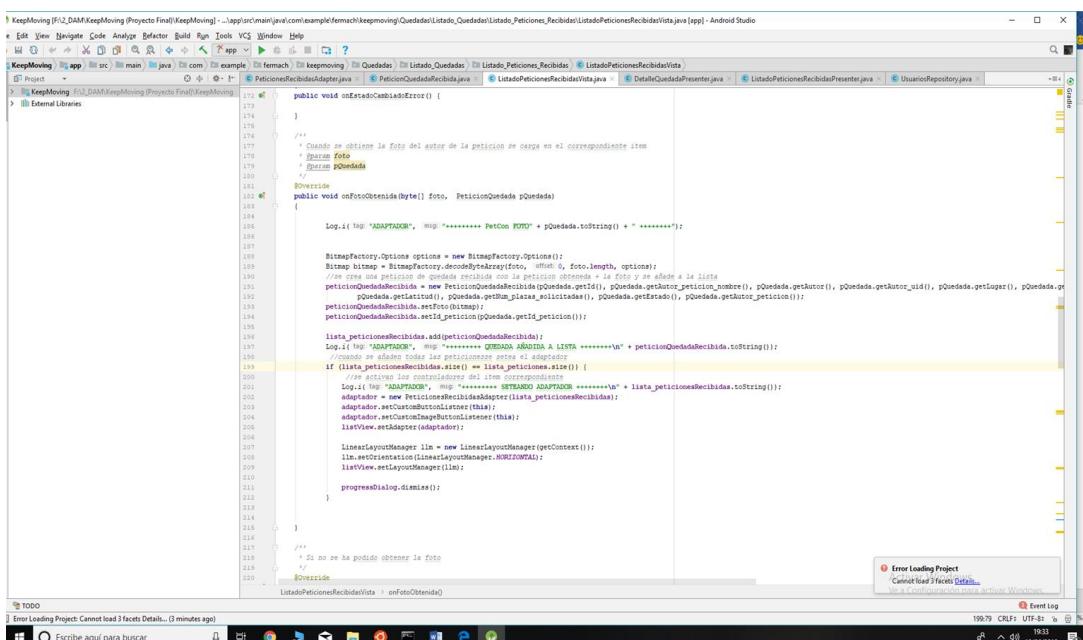
4. Seleccionamos los componentes que deseamos instalar y la ruta del SDK.



5. Seleccionamos instalar.



6. Una vez seguidos los anteriores pasos tendremos instalado Android Studio y listo para programar.

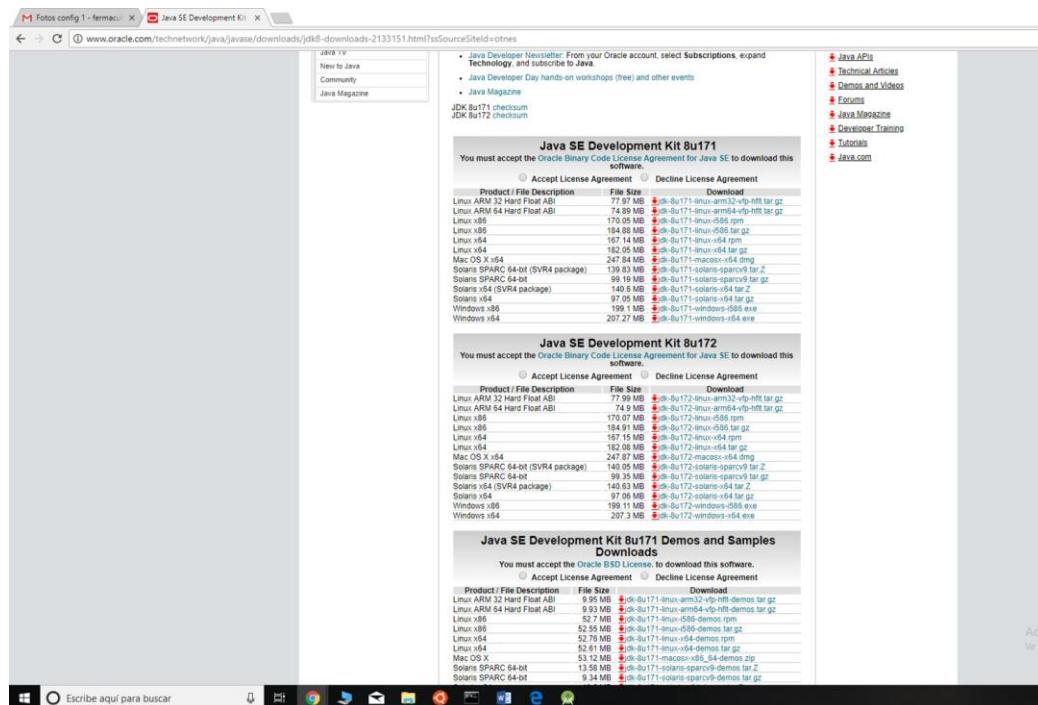


Instalación del JDK de Java.

Para instalar el Java Development Kit en tu ordenador debes realizar los siguientes pasos:

1. Debes dirigirte a la página web de Android Studio y descargar el .exe de la última versión utilizando el siguiente enlace:

<http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html?ssSourceSiteId=otnes>



2. Aceptamos los términos de licencia y clickamos sobre la versión que deseemos descargar. En mi caso sería la siguiente:

[Windows x64](#) [207.27 MB](#) [jdk-8u171-windows-x64.exe](#)

3. Ejecutamos el archivo y seguimos las instrucciones del asistente de instalación indicándole la ruta de instalación en mi caso “Program Files/Java/”, esto nos instalará el JDK de Java para que podamos desarrollar en este lenguaje nuestras aplicaciones.

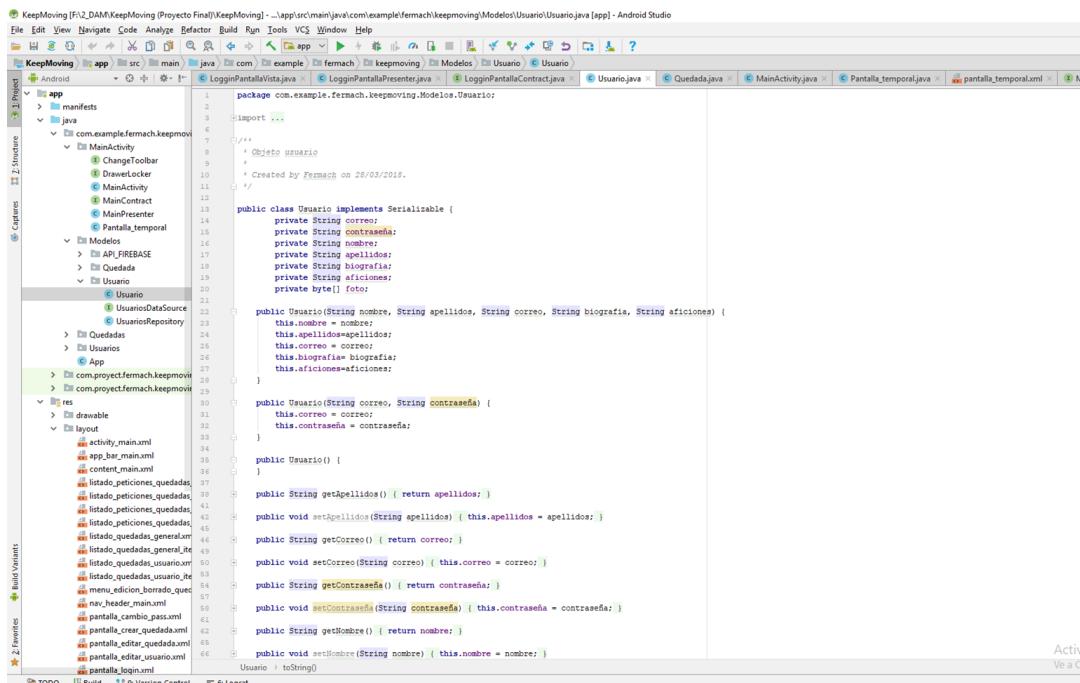
6. MÉTODO

Debido a la gran cantidad de clases que tiene el proyecto solo detallare aquellas partes del código que considere realmente relevantes.

Para explicar el desarrollo de mi proyecto voy a dividir este en distintas fases:

6.1. CREACIÓN DE LA ACTIVIDAD PRINCIPAL, IMPLEMENTACIÓN DE MATERIAL DESIGN Y CREACIÓN DE LOS MODELOS

En primer lugar, comencé creado mis modelos de usuario y quedada junto a sus geters, seters y constructores.



```

package com.example.fermach.keeppoving.Modelos.Usuario;

import java.io.Serializable;

public class Usuario implements Serializable {
    private String nombre;
    private String apellido;
    private String correo;
    private String biografia;
    private String aficiones;
    private byte[] foto;

    public Usuario(String nombre, String apellido, String correo, String biografia, String aficiones) {
        this.nombre = nombre;
        this.apellido = apellido;
        this.correo = correo;
        this.biografia = biografia;
        this.aficiones = aficiones;
    }

    public Usuario(String correo, String contraseña) {
        this.correo = correo;
        this.contraseña = contraseña;
    }

    public Usuario() {
    }

    public String getApellido() { return apellido; }

    public void setApellido(String apellido) { this.apellido = apellido; }

    public String getCorreo() { return correo; }

    public void setCorreo(String correo) { this.correo = correo; }

    public String getContraseña() { return contraseña; }

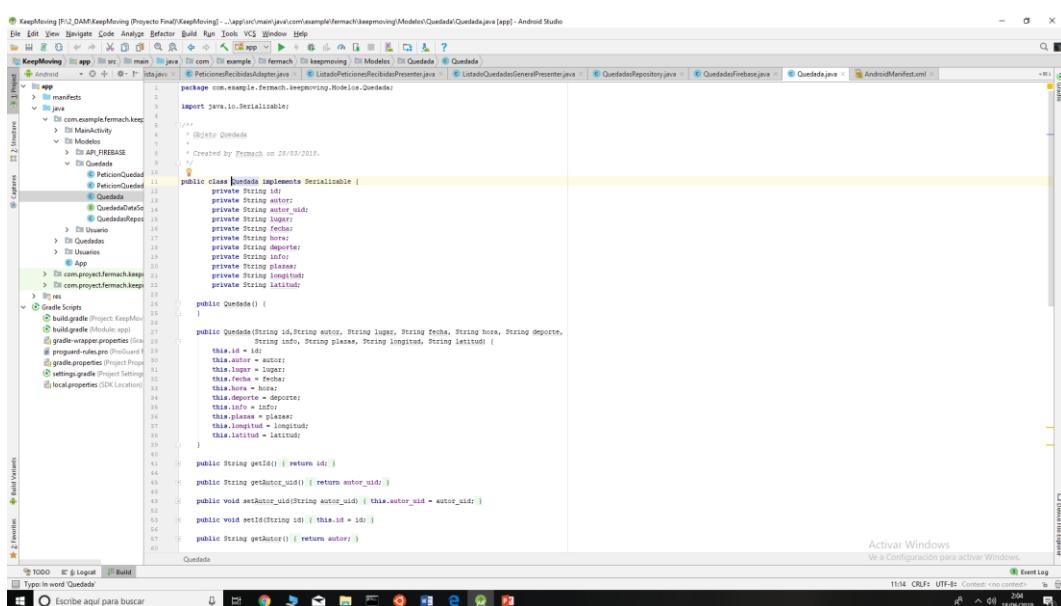
    public void setContraseña(String contraseña) { this.contraseña = contraseña; }

    public String getNombre() { return nombre; }

    public void setNombre(String nombre) { this.nombre = nombre; }

    @Override
    public String toString() { return "Usuario"; }
}

```



```

package com.example.fermach.keeppoving.Modelos.Quedada;

import java.io.Serializable;

public class Quedada implements Serializable {
    private String id;
    private String autor;
    private String autor_uid;
    private String lugar;
    private String fecha;
    private String hora;
    private String deporte;
    private String info;
    private String longitud;
    private String latitud;

    public Quedada() {
    }

    public Quedada(String id, String autor, String lugar, String fecha, String hora, String deporte, String info, String longitud, String latitud) {
        this.id = id;
        this.autor = autor;
        this.lugar = lugar;
        this.fecha = fecha;
        this.hora = hora;
        this.deporte = deporte;
        this.info = info;
        this.longitud = longitud;
        this.latitud = latitud;
    }

    public String getId() { return id; }

    public String getAutor_uid() { return autor_uid; }

    public void setAutor_uid(String autor_uid) { this.autor_uid = autor_uid; }

    public void setId(String id) { this.id = id; }

    public String getAutor() { return autor; }

    @Override
    public String toString() { return "Quedada"; }
}

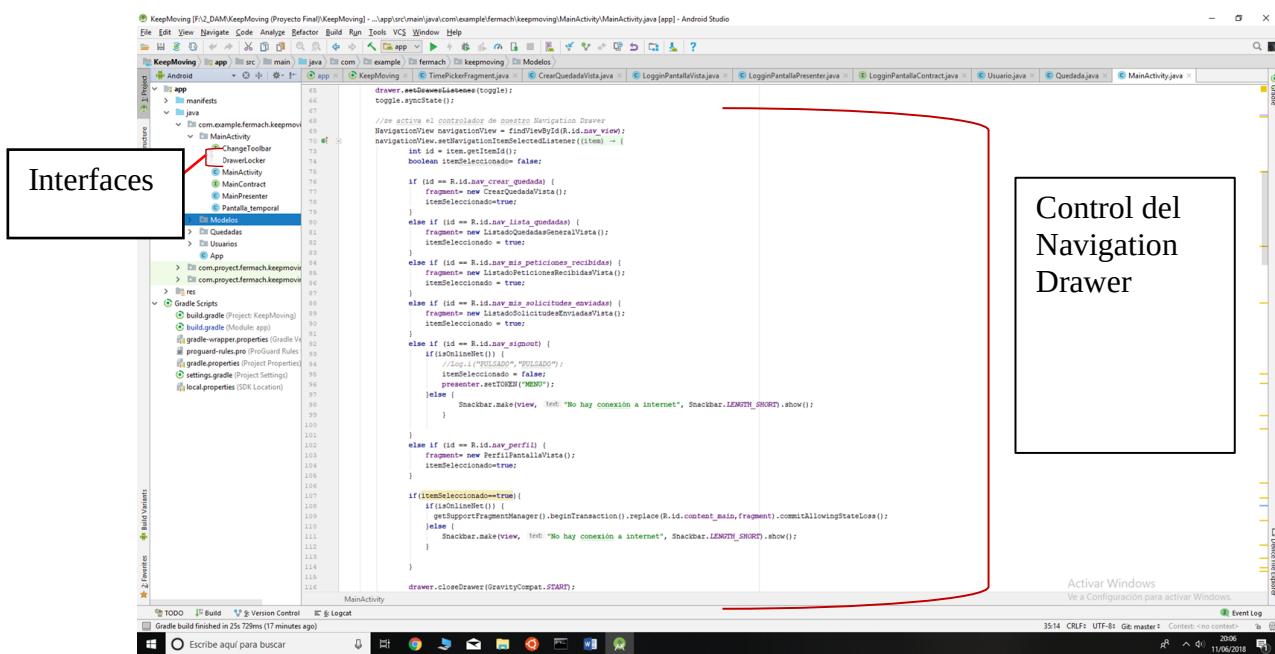
```

A continuación, implementé las librerías de Material Desing incluyendo en el gradle la siguiente línea:

implementation 'com.android.support:design:26.1.0'

Una vez que ya tenía los modelos POJO creados y las librerías implementadas, el siguiente paso fue crear la actividad principal donde se irían alternando los fragmentos accediendo al SupportFragmentManager, se instancia el navigation drawer, y el toolbar, además de controlar la acción de pulsar el botón de retroceder entre otras cosas.

Para poder controlar el Nav Drawer y El Toolbar cree dos interfaces a las que llamar desde los fragmentos para esconder estas vistas, mostrarlas o modificarlas.



La clase MainActivity también dispone de una interfaz propia, ya que es la que inicia el “Listener” que te dice si un usuario está o no registrado, para poder llamar a un fragmento u al otro en el caso de que el usuario esté o no registrado:

```

public interface MainContract {
    interface View {
        void onSesionCerrada();
        void onSesionCerradaError();
        void onUsuarioNoRegistrado(String TOKEN);
        void onUsuarioRegistrado(String TOKEN);
        void onTOKENseleccionado();
    }
    interface Presenter {
        void cerrarSesion();
        void setTOKEN(String TOKkEN);
        void iniciarListenerFire();
    }
}

```

Como podéis ver tiene dos interfaces, una que es implementada por el Presentador, el cual se comunica con el repositorio y la API para recibir y enviar datos, y la otra que la implementa la MainActivity y que es la que se encargará de llamar a unos métodos u otros en función de la respuesta que reciba el presentador como en el siguiente ejemplo:

```
@Override
public void iniciarListenerFire() {
    usuariosRepository.iniciarListener(new UsuariosDataSource.IniciarListenerCallback() {
        @Override
        public void onUsuarioRegistrado(String TOKEN) {
            view.onUsuarioRegistrado(TOKEN);
        }

        @Override
        public void onUsuarioNoRegistrado(String TOKEN) {
            view.onUsuarioNoRegistrado(TOKEN);
        }
    });
}
```

Firebase API:

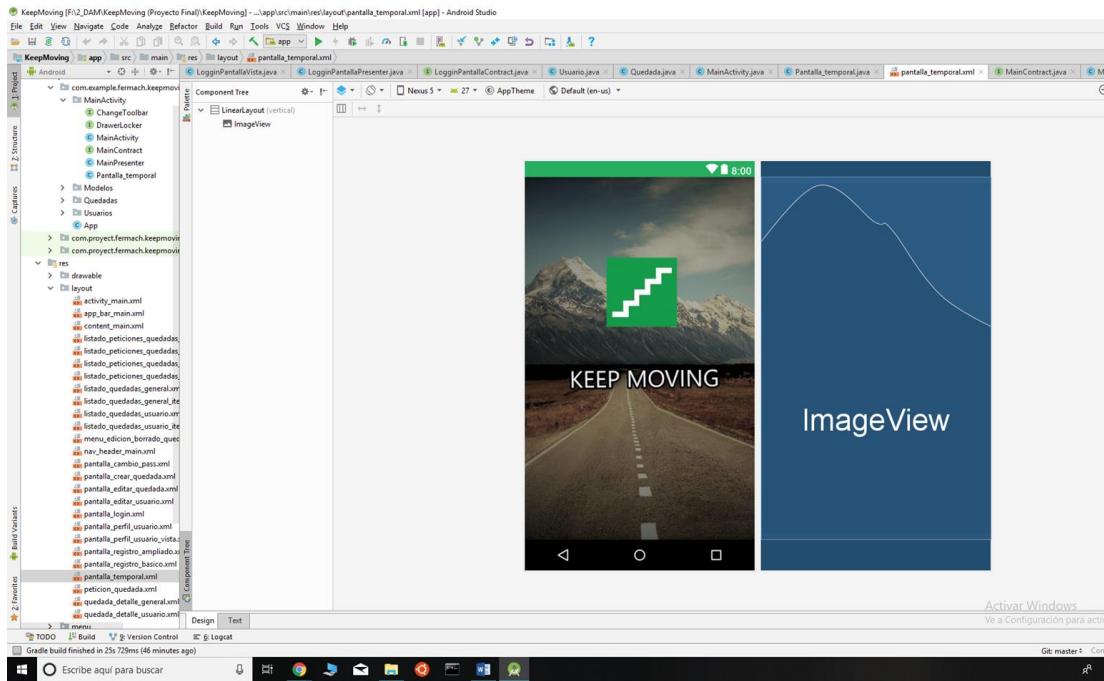
```
/**
 * Inicia el escuchador de control de usuario registrado
 * @param callback
 */
@Override
public void iniciarListener( final IniciarListenerCallback callback) {
    Log.i("LISTENER","USUARIOS FIRE 1" );

    mAuth.addAuthStateListener(new FirebaseAuth.AuthStateListener() {
        @Override
        public void onAuthStateChanged(@NonNull FirebaseAuth firebaseAuth) {
            user= firebaseAuth.getCurrentUser();
            Log.i("LISTENER","USUARIOS FIRE 2" );
            if(user!=null){
                Log.i("LISTENER","USUARIO REGISTRADO" );
                callback.onUsuarioRegistrado(TOKEN);
            }else{
                Log.i("LISTENER","USUARIO NO REGISTRADO" );
                callback.onUsuarioNoRegistrado(TOKEN);
            }
        }
    });
}
```

El método setToken() sirve para decirle a la API la pantalla en la que estamos, es decir, al método se le pasa una cadena u otra, y cuando retorne los datos, también retornará la cadena, por lo cual sabremos desde que pantalla se ha iniciado sesión , o desde que pantalla se ha cerrado la sesión.

De lo contrario siempre que el usuario inicie o cierre sesión realizaría la misma acción y ese no es el objetivo en este caso.

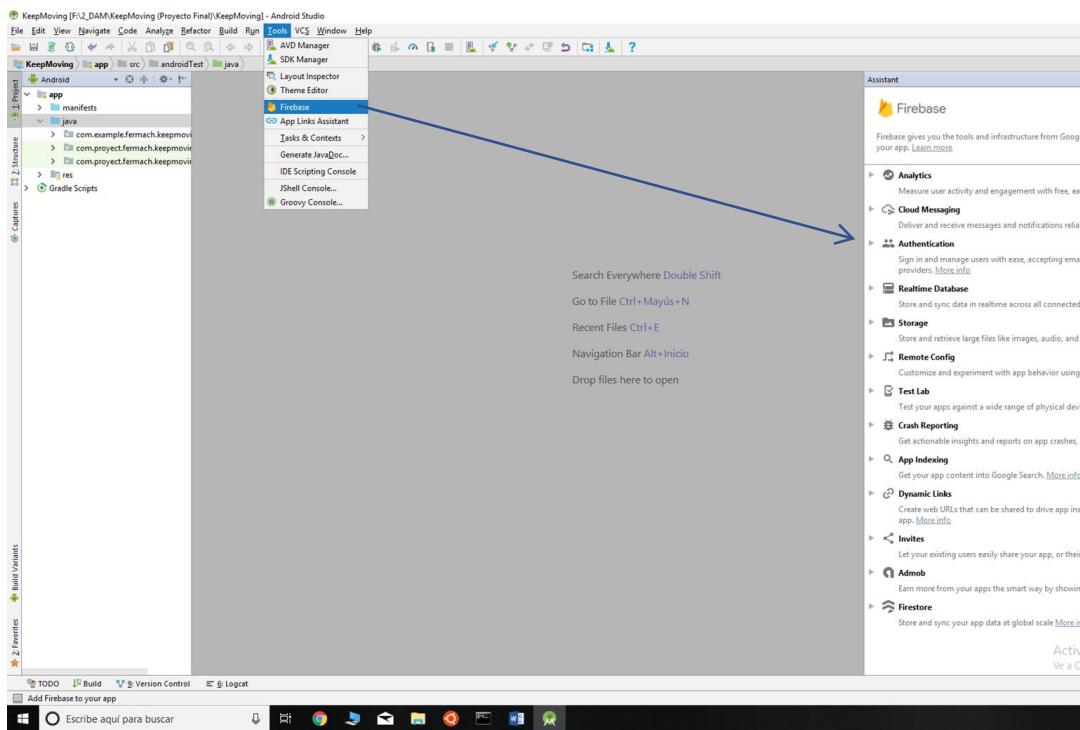
La clase pantalla temporal simplemente es un fragmento que muestra una imagen durante unos 4 segundos cuando se inicia la aplicación:



6.2. GESTIÓN DE USUARIOS

- Obtencion de la API KEY de Firebase.

En mi caso, para obtener el api key de firebase lo hice desde el propio Android Studio siguiendo los siguientes pasos:



Seleccionamos Authentication, a continuación, seleccionamos las siguientes opciones:

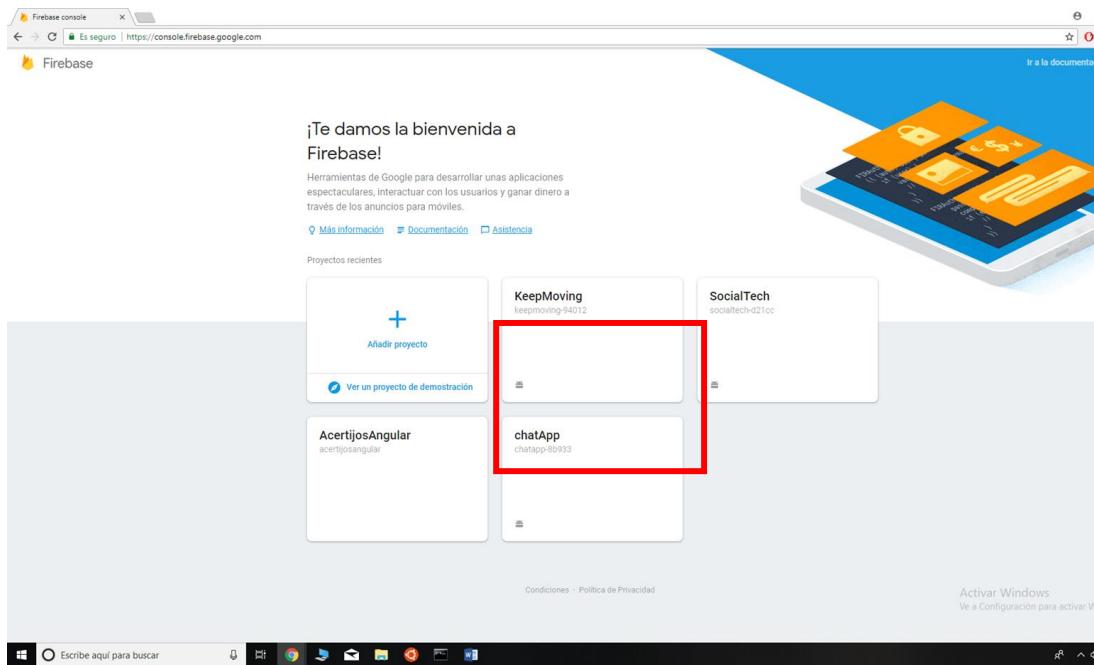


La primera de ellas nos pedirá que elijamos el nombre de nuestro proyecto, si ya teníamos un proyecto creado previamente en nuestra consola de Firebase seleccionamos elegir un repositorio existente, esto conectará nuestra aplicación con nuestro repositorio de Firebase.

En mi caso debido a que ya he conectado mi proyecto me aparece en conectado como estado.

La segunda opción nos permitirá acceder a “FireBase Authentication” con la cual gestionaremos los usuarios de nuestra aplicación, implementando en nuestro gradle las librerías que sean necesarias de forma automática.

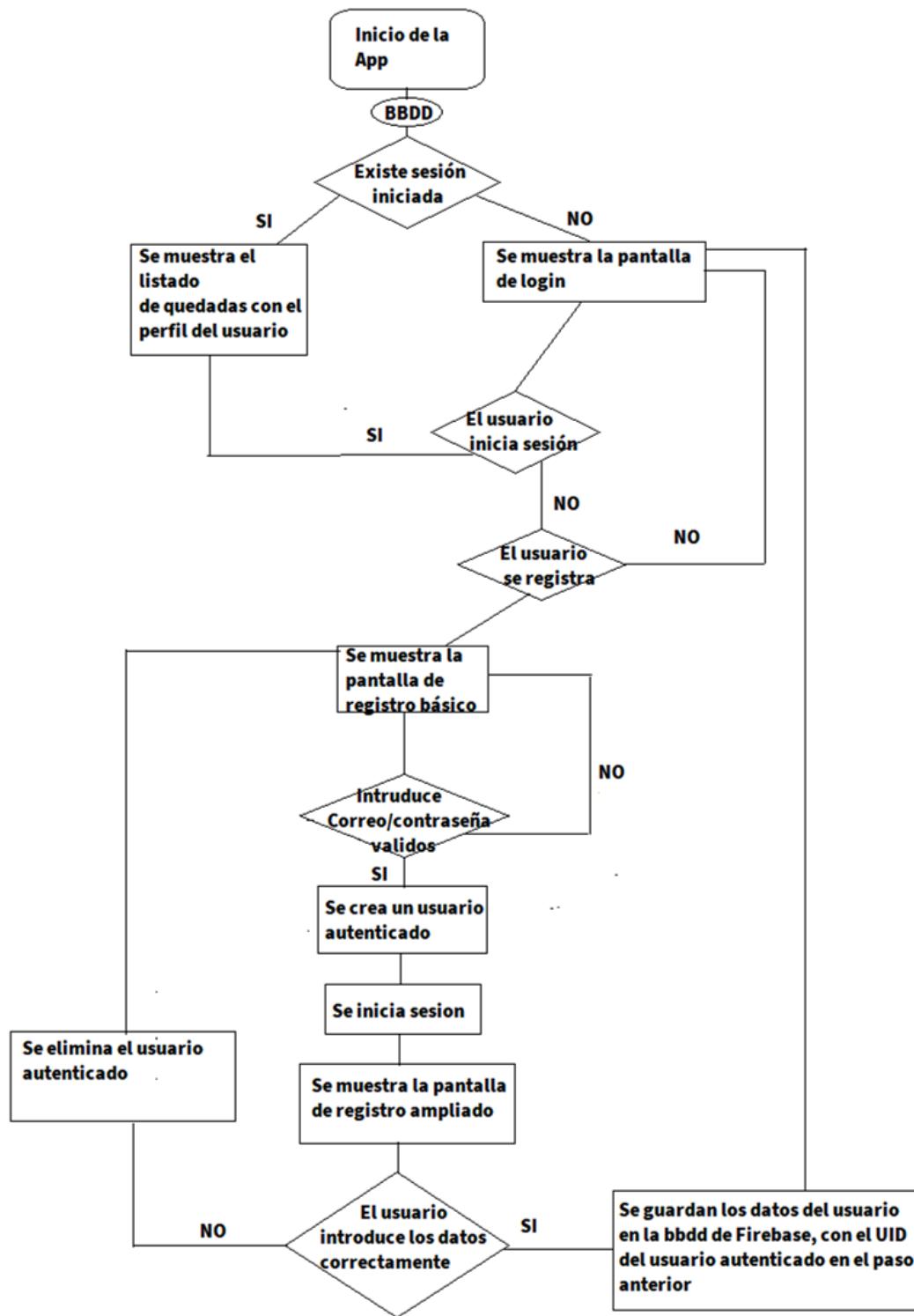
A continuación, si nos vamos a nuestra consola de FireBase veremos que se nos ha creado el nuevo repositorio con nuestro proyecto automáticamente:



Una vez hecho esta ya podremos acceder a nuestro repositorio

The screenshot shows the 'KeepMoving' project overview page. The left sidebar includes sections for 'DEVELOP' (Authentication, Database, Storage, Hosting, Functions, ML Kit), 'STABILITY' (Crashlytics, Performance, Test Lab), and 'ANALYTICS' (Dashboard, Events, Conversions, Audiences, Funnels, User Properties). The main content area displays 'Visión general' with a summary: 'Hay 1 aplicación en el proyecto'. It shows metrics for 'KeepMoving': 'com.project.fernach.keepmoving'. The metrics are: 'Usuarios activos diarios' (1), 'Usuarios activos mensuales' (27), 'Usuarios sin fallos' (74.1%), and 'Fallos' (129). Below this, there are three cards: 'Analytics' (with a graph showing growth), 'Authentication' (with a user icon), and 'Database' (with a server icon). At the bottom right, there is a link 'Activar Windows'.

El siguiente paso fue la crear el CRUD de usuarios y el sistema de gestión de usuarios, para ello me basé en el siguiente diagrama de flujo:



Como ya he explicado antes la clase MainActivity era la encargada de comprobar si un usuario estaba registrado o no, por lo cual en función de esto se mostraría la pantalla de login o la pantalla con las quedadas publicadas y el usuario registrado.

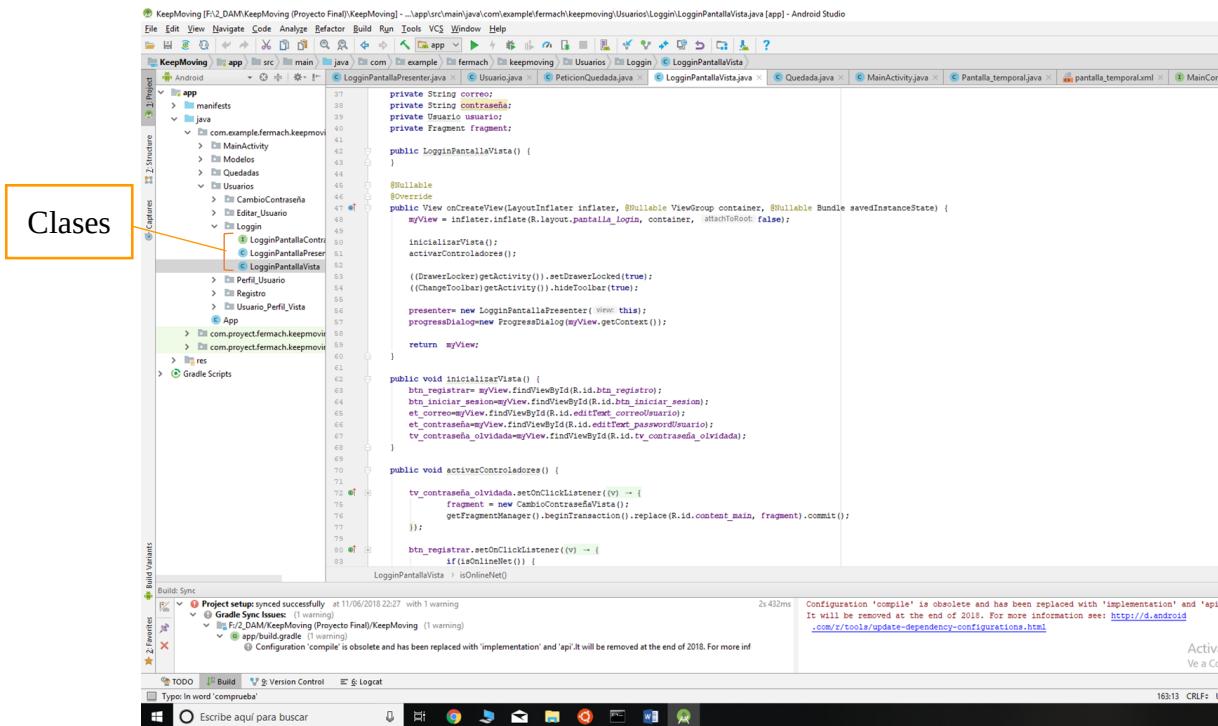
- Pantalla de Inicio de sesión:

Una vez que ya tenemos creados los modelos básicos y la conexión con Firebase, el siguiente paso fue crear la pantalla de login en la cual el usuario puede iniciar sesión o registrarse, para ello usé una interfaz con dos campos de texto y dos botones como se muestra en la imagen:



Esta pantalla a su vez estaba controlada por una vista que contenía los controles, un presentador que se comunicaba con el repositorio y la API de Firebase, y una interfaz que conectaba la vista y el presentador.

```
public interface LogginPantallaContract {  
    interface View {  
        void onSesionIniciadaError();  
        void onSesionIniciada();  
        void onTOKENseleccionado();  
    }  
    interface Presenter {  
        void loggearUsuario(Usuario usuario);  
        void comprobarRegistroDeUsuario();  
        void setTOKEN(String TOKKEN);  
    }  
}
```



Al pulsar sobre el inicio de sesión el presentador manda los datos del usuario a la API para que se inicie sesión con ese usuario, lo que hace que el listener que tenemos en la actividad principal muestre la pantalla de la lista de quedadas publicadas si se inicia la sesión correctamente o, por el contrario, si no se puede iniciar sesión, el callback le diría a la vista que mostrará un mensaje de error.

```
public void loguearUsuario(Usuario usuario, final LoguearUsuarioCallback callback) {
```

```
    String email= usuario.getCorreo();
    String password=usuario.getContraseña();

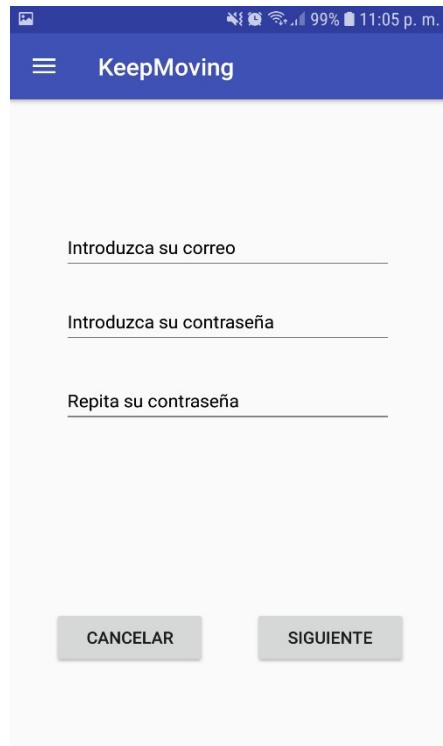
    Log.i("LOGGIN","EMAIL: "+email+", CONTRASEÑA: "+password);
```

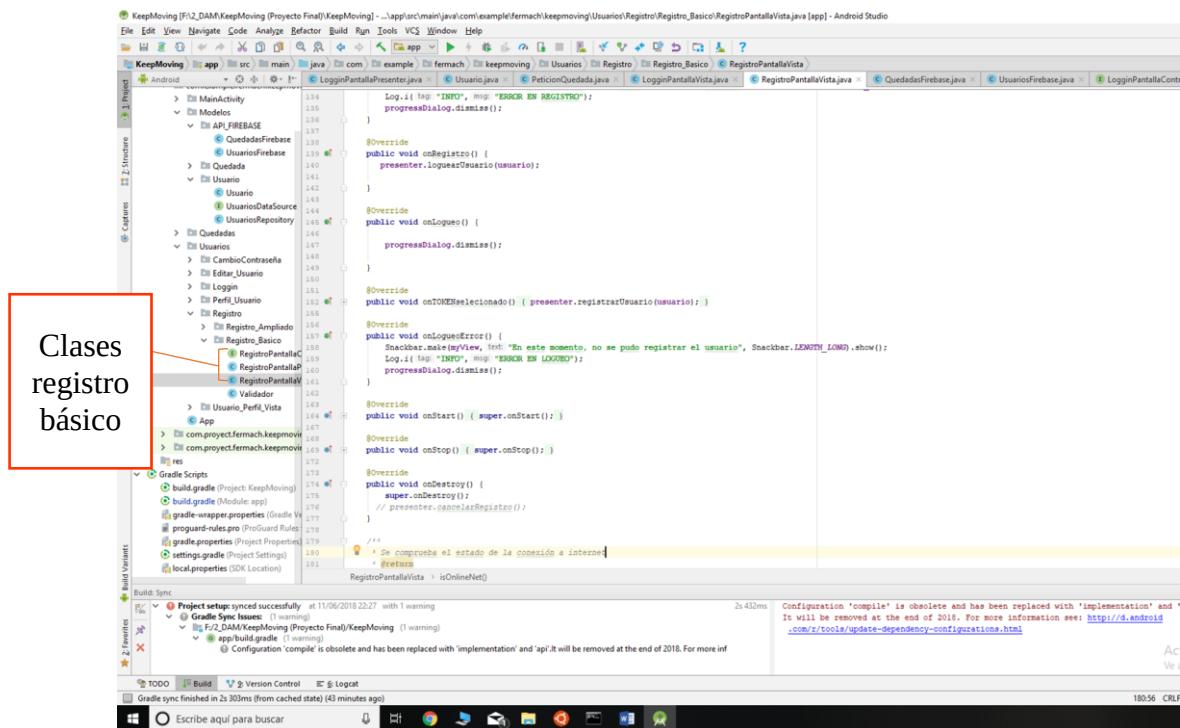
```
mAuth.signInWithEmailAndPassword(email,password)
    .addOnCompleteListener(new OnCompleteListener<AuthResult>() {
        @Override
        public void onComplete(@NonNull Task<AuthResult> task) {
            if(task.isSuccessful()){
                Log.i("LOGGIN","SUCESFUL");
                user = mAuth.getCurrentUser();
                callback.onUsuarioLogueado();
            }else{
                Log.i("LOGGIN","ERROR");
                callback.onUsuarioLogueadoError();
            }
        }
    });
}
```

Por el contrario, si pulsamos sobre registro, este botón nos llevará al fragmento de registro, para dar de alta a un nuevo usuario.

-Pantalla de registro básico:

Esta pantalla esta formada por dos botones (cancelar y siguiente) y tres EditText, uno de ellos para introducir el email y los otros dos para introducir la contraseña de forma repetida, a su vez al igual que el login, tiene sus correspondientes clases vista, presentador e interfaz.





Además, tiene una cuarta clase que sirve para validar un email o una fecha contra una expresión regular, en este caso la usamos para validar el email de nuestro usuario.

```
public class Validador {
```

```

// email y fecha expresiones de validacion
private static final String EMAIL_REGEX = "^(\\w+)(\\.(\\w+))*@((\\w+)(\\.(\\w+))*)\\.(\\w{2,3})$";
private static final String FECHA_REGEX= "^(?:(?:31(\\/[\\d]{1,2})|(?:(?:29|30)(\\/[\\d]{1,2})|(?:(?:0[1-9]|1[0-2])\\/[2]))|(?:(?:1[6-9]|([2-9]\\d)|\\d{2}))$|^(?:(?:29(\\/[\\d]{1,2})|0?2|3(?:(?:1[6-9]|([2-9]\\d)|(?:(?:0[48]|2468)[048]|([13579][26]))|(?:(?:16[2468][048]|([3579][26])00))))))$|^(?:(?:1[1-9]|1\\d{2}|[0-8])(\\/[\\d]{1,2})|(?:(?:0?|[1-9])|(?:(?:1[0-2])|\\4(?:(?:1[6-9]|([2-9]\\d)|\\d)?\\d{2})))$";
private static Pattern pattern_email;
private static Pattern pattern_fecha;

```

```
private Matcher matcher;
```

```
public Validador() {
    pattern_email = Pattern.compile(EMAIL_REGEX, Pattern.CASE_INSENSITIVE);
    pattern_fecha = Pattern.compile(FECHA_REGEX, Pattern.CASE_INSENSITIVE);
}
```

```
/**  
 * Este método valida un email  
 *  
 * @param email  
 * @return boolean
```

```

/*
public boolean validateEmail(String email) {
    matcher = pattern_email.matcher(email);
    return matcher.matches();
}

/**
 * Este metodo valida una fecha
 * @param fecha
 * @return
 */
public boolean validateFecha(String fecha) {
    matcher = pattern_fecha.matcher(fecha);
    return matcher.matches();
}
}

```

Hablando de los controladores en este caso el botón de cancelar nos dirige a la pantalla de login, mientras que el botón de siguiente lo que hace es:

1. Manda el token para decirle a el listener que estamos en la pantalla de registro, de lo contrario nos llevaría a la panta de listado de quedadas como si hubiéramos iniciado sesión desde la pantalla de login.
2. Una vez seteado el token registra un usuario contra la API de autentificación de usuarios de Firebase, en caso de error se muestra un mensaje de error.

```

@Override
public void registrarUsuario(Usuario usuario, final RegistrarUsuarioCallback callback) {
    // _____
    String email= usuario.getCorreo();
    String password=usuario.getContraseña();

    Log.i("REGISTRO","EMAIL: "+email +", CONTRASEÑA: "+password);
    mAuth.createUserWithEmailAndPassword(email,password)
        .addOnCompleteListener( new OnCompleteListener<AuthResult>() {
            @Override
            public void onComplete(@NonNull Task<AuthResult> task) {

                if(task.isSuccessful()){
                    Log.i("REGISTRO_FIRE","SUCCEFUL");
                    callback.onUsuarioRegistrado();
                }else{
                    Log.i("REGISTRO_FIRE","ERROR " +task.getException().getMessage());
                    callback.onUsuarioRegistradoError(""+task.getException().getMessage());
                }
            }
        });
}

```

3. Una vez registra el usuario se inicia sesión con ese usuario. Como le hemos pasado el token a la API, cuando el listener detecte que se ha iniciado sesión con un usuario sabrá que estamos en la pantalla de registro y por ello nos llevará a la siguiente pantalla de registro.

Main Activity.java (listener)

```
public void onUsuarioRegistrado(String TOKEN) {  
    this.TOKEN=TOKEN;  
  
    if(isOnlineNet()) {  
        if (TOKEN == "LOGGIN") {  
            Log.i("TOKEEN MAIN", TOKEN);  
  
            fragment = new ListadoQuedadasGeneralVista();  
            getSupportFragmentManager().beginTransaction().replace(R.id.content_main,  
fragment).commitAllowingStateLoss();  
        } else {  
            Log.i("TOKEEN MAIN", TOKEN);  
  
            fragment = new RegistroAmpliadoPantallaVista();  
            getSupportFragmentManager().beginTransaction().replace(R.id.content_main,  
fragment).commitAllowingStateLoss();  
  
        }  
    } else{  
  
        fragment = new LogginPantallaVista();  
        getSupportFragmentManager().beginTransaction().replace(R.id.content_main, frag-  
ment).commitAllowingStateLoss();  
        Snackbar.make(view,"No hay conexión a internet", Snackbar.LENGTH_  
SHORT).show();}}}
```

- Pantalla de registro ampliado:

Esta pantalla contiene unos EditText para introducir los datos del usuario, dos botones (atrás y registrarse) y un CircleImageView, el cual implementé utilizando el siguiente enlace añadido a mi Gradle:

implementation 'de.hdodenhof:circleimageview:2.1.0'

y así es como lo tengo implementado en la vista:

```
<RelativeLayout
    android:layout_width="match_parent"
    android:layout_height="140sp"
    android:background="@color/azul_claro"
    >

    <de.hdodenhof.circleimageview.CircleImageView
        android:id="@+id/fab_usuarioImagen_registro"
        android:layout_margin="16sp"
        android:layout_width="match_parent"
        android:src="@drawable/user"
        android:layout_height="match_parent"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        app:civ_border_color="@android:color/black"
        app:civ_border_width="2sp"
        android:elevation="10sp"
    />
</RelativeLayout>
```



La clase, al igual que las demás, cuenta con vista y su presentador, conectados por su correspondiente interfaz.

Clases
del
registro
ampliado

```
KeepMoving [F:\2_DAM\KeepMoving (Proyecto Final)\KeepMoving] - \app\src\main\java\com\example\fernach\keepmoving\Usuarios\Registro_Ampliado\RegistroAmpliadoPantallaVista.java [app] - Android Studio
```

```
public void activarControladores() {
    btn_Registrarse.setOnClickListener(v) -> {
        if (isOnlineNet()) {
            //VALIDAR CORREO
            nombre+=" "+et_nombre.getText().toString().trim();
            apellido+=" "+et_apellido.getText().toString().trim();
            aficiones+=" "+et_aficiones.getText().toString().trim();
            biografia+=" "+et_biografia.getText().toString().trim();

            // se comprueba los datos introducidos
            if(!nombre.isEmpty() && !apellido.isEmpty() ) {
                progressDialog.setMessage("Se están validando los datos");
                progressDialog.setCancelable(false);
                progressDialog.show();

                //se controla si se ha introducido foto o no
                if(foto !=null) {
                    usuario = new Usuario(nombre, apellido, correo+"@correo.biografia,aficiones);
                    presenter.registrarUsuarioConFoto(usuario,foto_bytes);

                }else{
                    usuario = new Usuario(nombre, apellido, correo+"@correo.biografia,aficiones );
                    presenter.registrarUsuario(usuario);
                }

            }else{
                Snackbar.make(myView, "Debe introducir el nombre y los apellidos", Snackbar.LENGTH_SHORT).show();
            }
        }else {
            Snackbar.make(myView, "No hay conexión a internet", Snackbar.LENGTH_SHORT).show();
        }
    });
    btn_atras.setOnClickListener(v) -> {
        progressDialog.dismiss();
    });
}
```

Hablando de los controles de esta clase tenemos por un lado el CircleImageView, el cuál al pulsarlo nos abre la galería para que seleccionemos una foto de perfil, al seleccionar esta cogemos la imagen en forma de Array de bytes y la transformamos a Bitmap para posteriormente establecerla como imagen de fondo en el CircleImageView:

```

foto_registro.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        if (isOnlineNet()) {
            cargarImagenGaleria();

        }else {
            Snackbar.make(myView, "No hay conexión a internet", Snackbar.LENGTH_SHORT).show();
        }
    }
});

//se carga una imagen de perfil de la galeria
public void cargarImagenGaleria(){
    Intent intent= new Intent(Intent.ACTION_PICK, MediaStore.Images.Media.EXTERNAL_CONTENT_URI);
    intent.setType("image/");
    startActivityForResult(intent.createChooser(intent,"Seleccione la Aplicación"),10);

}

@Override
public void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    if(resultCode==1){
        foto=data.getData();

        Bitmap bitmap = null;
        try {
            bitmap = MediaStore.Images.Media.getBitmap(App.getAppContext().getContentResolver(),foto);
            ByteArrayOutputStream baos = new ByteArrayOutputStream();
            bitmap.compress(Bitmap.CompressFormat.JPEG, 25, baos);
            foto_bytes = baos.toByteArray();
        } catch (IOException e) {
            e.printStackTrace();
        }
        foto_registro.setImageBitmap(bitmap);

    }
}

```

Por otro lado, tenemos los dos botones, el botón de cancelar lo que hace es:

1. Elimina el usuario que hemos registrado en el registro básico con el email y la contraseña, ya que si no se ha completado el registro con todos los datos obligatorios que son email, contraseña, nombre y apellidos, el usuario no es valido y por lo tanto no debe existir.
2. Retrocede al fragmento de registro básico.

El botón registrar, por el contrario, lo que hace es que registra un usuario en la base de datos con los datos introducidos y el email de el usuario registrado en el repositorio de usuarios autenticados, y se establece el UID del usuario autenticado como referencia en la base de datos al usuario autenticado.

Para guardar la foto de perfil lo hago en el Storage de Firebase usando el UID del usuario como referencia de la foto.

Identificador	Proveedores	Fecha de creación	Inicio de sesión	UID de usuario ↑
f.manu_97@hotmail.com		6 jun. 2018	9 jun. 2018	4G1wYXSs6NOFXGQGczusXOFacWj1
fitohornos@hotmail.com		7 jun. 2018	7 jun. 2018	53ihHRGw0vMRQatLgw9pmAL8...

keepmoving-94012

- Quedadas
- Users
- 4G1wYXSs6NOFXGQGczusXOFacWj1

Peticiones enviadas

Peticiones recibidas

Quedadas

aficiones: "Fútbol, Fitness, Senderismo, Running"

apellidos: "Cuadros Hornos"

biografia: "Nunquian Desperare 🏃"

correo: "f.manu_97@hotmail.co"

nombre: "Fernando M"

- 53ihHRGw0vMRQatLgw9pmAL8mn2

WHjGE5JGSHMbzxma9sRoxHSQOT

gs://keepmoving-94012.appspot.com > FotosPerfil

Nombre	Tamaño	Tipo	Última modificación
4G1wYXSs6NOFXGQGczusXOFacWj1	0...	image/jpeg	6 jun. 2018
fZ64fELYjdgiQvn1eT73zjPuB...	51,6...	image/jpeg	6 jun. 2018
JDV80BL0zIzIluJOQOdiWYQ5...	310...	image/jpeg	8 jun. 2018
iRN2WUSA4CgRNLN4bILMr3...	85,3...	image/jpeg	8 jun. 2018

4G1wYXSs6NOFXG...

Nombre: 4G1wYXSs6NOFXGQGczusXOFacWj1

Tamaño: 15,450 bytes

Tipo: image/jpeg

Fecha de creación: 6 jun. 2018 21:37:04

Fecha y hora de actualización: 6 jun. 2018 21:37:04

Ubicación del archivo:

Este es el código de la API de Firebase que utilzo para guardar el usuario con su foto de perfil:

```
@Override
public void registrarUsuarioAmpliadoConFoto(final Usuario usuario, final byte[] foto,
final RegistrarUsuarioConFotoCallback callback) {

    myfileStoragePath=myStorageRef.child("FotosPerfil").child(user.getUid());

    Log.i("USUARIO_PUSH",usuario.toString());
    UsuariosRef.child(user.getUid()).setValue(usuario).addOnCompleteListener(new On-
CompleteListener<Void>() {
        @Override
        public void onComplete(@NonNull Task<Void> task) {
            if (task.isSuccessful()){
                myfileStoragePath.putBytes(foto).addOnCompleteListener(new OnComplete-
Listener<UploadTask.TaskSnapshot>() {
                    @Override
                    public void onComplete(@NonNull Task<UploadTask.TaskSnapshot> task)
{
                        if(task.isSuccessful()){
                            callback.onUsuarioRegistrado();
                            Log.i("REGISTRO_FIRE_DB","SUCCEFUL");
                        }else{
                            //borrar usuario de la BBDDs

                            Log.i("REGISTRO_FIRE_DB","ERROR");
                            callback.onUsuarioRegistradoError();
                        }
                    }
                });
            }else{
                Log.i("REGISTRO_FIRE_DB","ERROR");
                callback.onUsuarioRegistradoError();
            }
        }
    });
}
```

- Pantalla de Perfil del Usuario.

Esta pantalla está formada principalmente por una serie de TextView que indican los datos del usuario, un CircleImageView para mostrar la foto de perfil y dos botones (Modificar datos y Mis quedadas), además, cuenta con su correspondiente vista y presentador, y su interfaz para conectar ambas.



Clases
Pantalla
de
perfil

```

KeepMoving [F:\2_DAM\KeepMoving (Proyecto Final)\KeepMoving] - ...\\app\\src\\main\\java\\com\\example\\fermack\\keepmoving\\Usuarios\\Perfil_Perfil\\PerfilPantallaVista.java - Android Studio
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
KeepMoving app src main java com example fermack keepmoving Usuarios Perfil_Perfil PerfilPantallaVista
Project Cursos Favorites
KeepMoving app manifest AndroidManifest.xml
src main Java com example fermack keepmoving Usuarios MainActivity
Models API_FIREBASE
QuedadasFirebase UsuariosFirebase
Quedadas Editar_Usuario
Users Usuarios
UsersDataSource UsuariosRepository
Quedadas Usuarios CambioContraseña Login
Perfil_Perfil PerfilPantallaVista
Registros Usuarios_Perfil_Vista App
com.example.fermack.keepmoving com.example.fermack.keepmoving
res drawable layout
activity_main.xml app_bar_main.xml content_main.xml listado_perfiles_quedadas
listado_perfiles_quedadas

```

```

public void inicializarVista() {
    presenter = new PerfilPantallaPresenter( myView );
    progressDialog = new ProgressDialog( myView.getContext() );
    progressDialog.setMessage("Obteniendo datos");
    progressDialog.setCancelable(false);
    progressDialog.show();
}

//se obtienen los datos del usuario actual
presenter.ObtenerUsuarioActual();
presenter.ObtenerFotoUsuarioActual();

return myView;
}

public void inicializarVista() {
    btn_mis_quedadas.setOnClickListener( v ) -> {
        if (isOnlineNet()) {
            fragment = new ListadoQuedadasUsuarioVista();
            getSupportFragmentManager().beginTransaction().replace(R.id.content_main, fragment).commit();
        } else {
            Snackbar.make( myView, "No hay conexión a internet", Snackbar.LENGTH_SHORT ).show();
        }
    };
}

btn_editor_datos.setOnClickListener( v ) -> {
    if (isOnlineNet()) {
        progressDialog = new ProgressDialog( myView );
        progressDialog.show();
        presenter.modificar_perfil();
    } else {
        Snackbar.make( myView, "No hay conexión a internet", Snackbar.LENGTH_SHORT ).show();
    }
};


```

Project setup: synced successfully at 12/06/2018 17:52 with 1 w 2s 681ms Configuration 'compile' is obsolete and has been replaced with 'implementation' and 'api'. It will be removed at the end of 2018. For more information see: <http://d.android.com/r/tools/update-dependency-configurations.html>

Esta pantalla carga los TextView y la foto (CircleImageView) con los datos del usuario registrado, para ello utiliza los siguientes métodos:

```

@Override
public void obtenerFotoPerfilUsuario(String uid, final PeticionQuedada pQuedada, final
ObtenerFotoPerfilUsuarioCallback callback) {
    myfileStoragePath=myStorageRef.child("FotosPerfil/").child(uid);

    myfileStoragePath.getBytes(ONE_MEGABYTE).addOnSuccessListener(new On-
SuccessListener<byte[]>() {
        @Override
        public void onSuccess(byte[] bytes) {
            Log.i("OBTENER FOTO US FIRE","SUCESFUL" );
            callback.onFotoUsuarioPerfilObtenida(bytes, pQuedada);
        }
    }).addOnFailureListener(new OnFailureListener() {
        @Override
        public void onFailure(@NonNull Exception e) {
            Log.i("OBTENER FOTO US FIRE","ERROR" );
            callback.onFotoUsuarioPerfilObtenidaError(pQuedada);
        }
    });
}

@Override
public void obtenerUsuarioActual(final ObtenerUsuarioActualCallback callback) {
    UsuariosRef.child(user.getUid()).addListenerForSingleValueEvent(new ValueEventListener() {
        @Override
        public void onDataChange(DataSnapshot dataSnapshot) {
            String nombre= dataSnapshot.child("nombre").getValue(String.class);
            String apellidos= dataSnapshot.child("apellidos").getValue(String.class);
            String correo= dataSnapshot.child("correo").getValue(String.class);
            String biografia= dataSnapshot.child("biografia").getValue(String.class);
            String aficiones= dataSnapshot.child("aficiones").getValue(String.class);
            usuarioActual= new Usuario(nombre,apellidos,correo,biografia,aficiones);

            Log.i("OBTENER USUARIO FIRE","SUCESFUL -- "+usuarioActual);
            callback.onUsuarioObtenido(usuarioActual);
        }
    });

    @Override
    public void onCancelled(DatabaseError databaseError) {
        callback.onUsuarioObtenidoError();
        Log.i("OBTENER USUARIO FIRE","ERROR -- "+usuarioActual);
    }
}

```

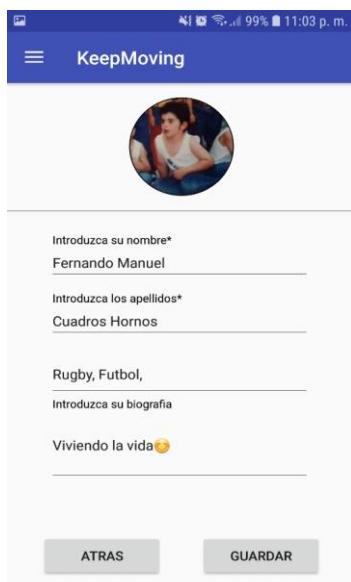
En cuanto a los controles de la vista, el botón de Mis quedadas nos lleva a la pantalla con un listado de todas las quedadas que ha publicado el usuario para que este pueda modificarlas, editarlas, o visualizar el detalle.

En cuanto al botón de Modificar datos, lo que hace es que nos lleva a la pantalla de edición de usuario para modificar sus datos, pasándole como parámetros los datos del usuario registrado.

- Pantalla de edición de usuario.

Esta pantalla cuenta con un CircleImageView para cargar la foto del usuario, unos EditTexts que se llenan con la información del usuario registrado, la cual le hemos pasado desde la pantalla de perfil y dos botones (cancelar y guardar).

Además, también cuenta con su correspondiente vista y presentador, y su interfaz para conectarlos.



Clases de la pantalla de edición de usuario

The screenshot shows the Android Studio interface with the following details:

- Project Structure:** The left sidebar displays the project structure under the 'KeepMoving' app module. It includes the 'app' module with its manifest and Java files, and other modules like 'com.example.fermach.KeepMoving' and 'com.fernach.KeepMoving'. The 'res' and 'layout' folders are also visible.
- Code Editor:** The main window shows the 'EditarUsuarioVista.java' file. The code handles user input validation and initializes a view. A red box highlights the code block where the view is initialized.
- Build Sync:** At the bottom, a message indicates a successful sync operation at 12/06/2018 17:52.
- Logs:** The bottom right corner shows '666 CRFL'.

```
if(args!=null) {
    if(args[0]==null) {
        //Preparar formulario de perfil
        foto_bytes=(byte[]) args.getSerializable(key: "FOTO");
        //Recoger argumentos
        usuario_ref = (Usuario) args.getSerializable(key: "USUARIO");
        Log.i("tag", "Argumentos: " + usuario_ref.toString());
    }else{
        Log.i("tag", "Argumentos: " + "NULLOS" );
    }
}
initializarVista();
activarControladores();

presenter=new EditarUsuarioPresenter(view: this);

return myView;
}

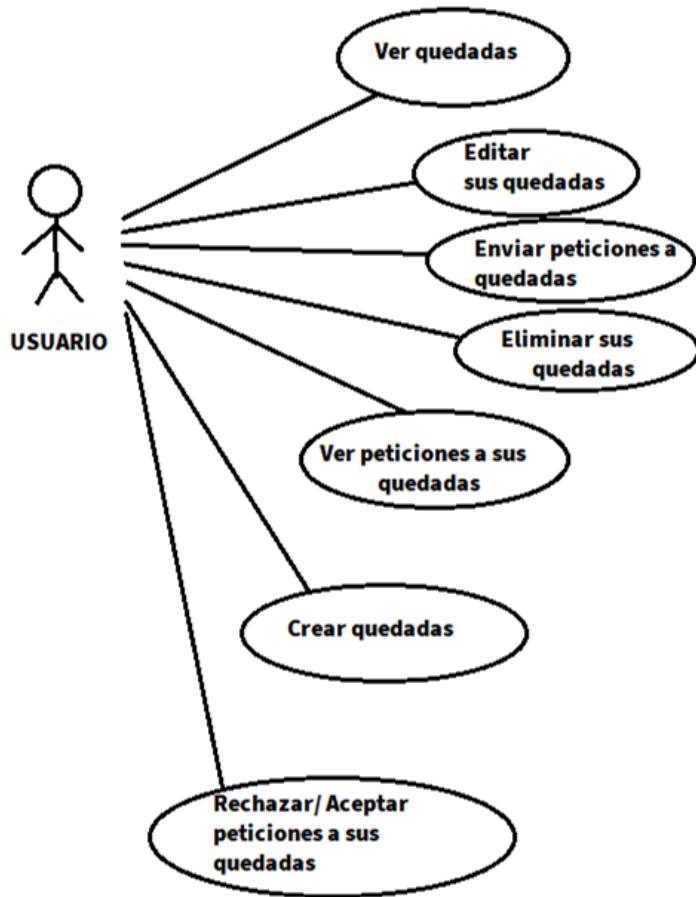
public void inicializarVista() {
    btn_guardar=myView.findViewById(R.id.btn_guardar_editar_usuario);
    btn_atrasasmyView.findViewById(R.id.btn_atras_editar_usuario);
    et_apellidosmyView.findViewById(R.id.editText_apellido_usuario_editar_usuario);
    et_nombremyView.findViewById(R.id.editText_nombre_usuario_editar_usuario);
    et_fotoPerfilmyView.findViewById(R.id.editText_biotografia_usuario_editar_usuario);
    et_biotografiamyView.findViewById(R.id.editText_biotografia_usuario_editar_usuario);
    et_aficionesmyView.findViewById(R.id.editText_aficiones_usuario_editar_usuario);
    foto_editar_usuario=myView.findViewById(R.id.fab_editar_usuario);
}
```

Hablando de los controles, el botón de cancelar simplemente retorna a la pantalla principal del usuario registrado y el botón de guardar lo que hace es actualizar los datos del usuario registrado con la foto y los datos introducidos en la pantalla, para ello utiliza el siguiente método en la API de Firebase.

```
@Override  
public void editarUsuario(final Usuario usuario, final byte[] foto, final EditarUsuario-  
Callback callback) {  
    Query query = UsuariosRef.child(user.getUid());  
  
    query.addListenerForSingleValueEvent(new ValueEventListener() {  
        @Override  
        public void onDataChange(DataSnapshot dataSnapshot) {  
            dataSnapshot.getRef().child("aficiones").setValue(usuario.getAficiones());  
            dataSnapshot.getRef().child("apellidos").setValue(usuario.getApellidos());  
            dataSnapshot.getRef().child("nombre").setValue(usuario.getNombre());  
            dataSnapshot.getRef().child("biografia").setValue(usuario.getBiografia());  
  
            myfileStoragePath=myStorageRef.child("FotosPerfil/").child(user.getUid());  
            myfileStoragePath.putBytes(foto).addOnCompleteListener(new OnCompleteListener<UploadTask.TaskSnapshot>() {  
                @Override  
                public void onComplete(@NonNull Task<UploadTask.TaskSnapshot> task) {  
                    if(task.isSuccessful()){  
                        callback.onUsuarioEditado();  
                        Log.i("EDITAR_FIRE_DB","SUCESFUL");  
                    }else{  
                        Log.i("EDITAR_FIRE_DB","ERROR");  
                        callback.onUsuarioEditadoError();  
                    }  
                }  
            });  
        }  
    }  
  
    @Override  
    public void onCancelled(DatabaseError databaseError) {  
    }  
});}
```

6.3. GESTIÓN DE QUEDADAS

Una vez que ya tenía controlada la parte de inicio de sesión y registro de los usuarios, el siguiente paso fue organizar la gestión de las quedadas, para que los usuarios pudieran crear, editar y eliminar quedadas, además de hacer peticiones a estas, las cuales podían ser aceptadas o rechazadas.

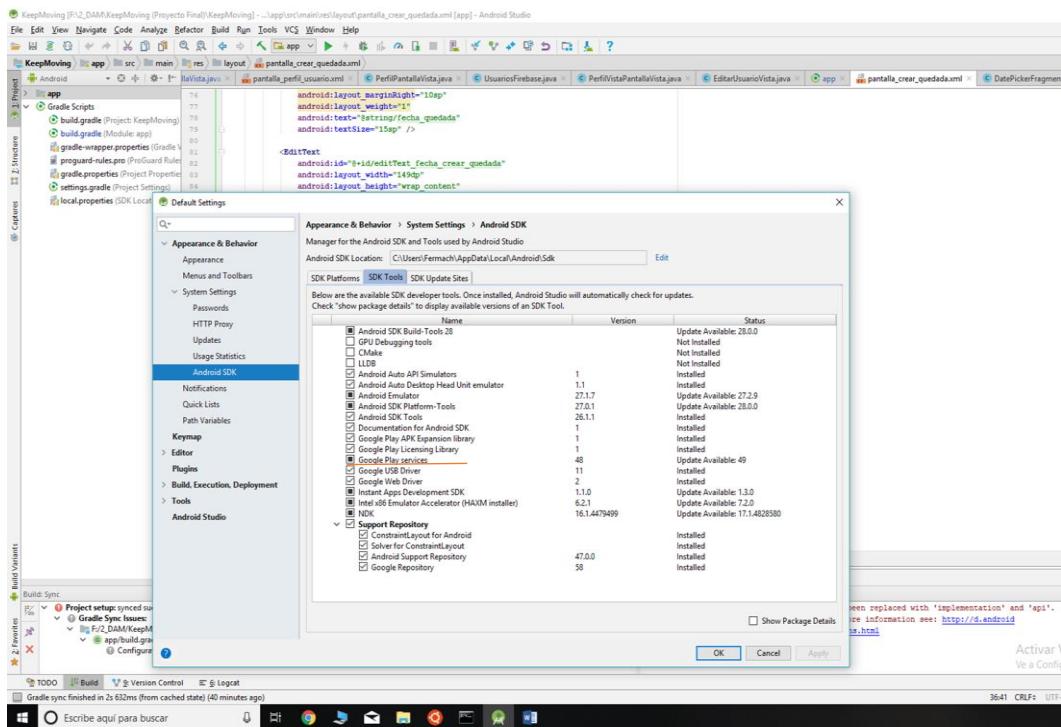


- Implementación de Google Maps.

Para explicar cómo he implementado los mapas de Google lo haré siguiendo una serie de pasos:

1. Obtener la API, acceder a la clave y agregar los atributos necesarios a tu manifiesto de Android.

En primer lugar, debemos dirigirnos al SDK manager y asegurarnos de que tenemos implementado Google Play Services.



Además, debemos implementar en el gradle las librerías correspondientes, en mi caso he usado la versión 11.0.4:

```
implementation 'com.google.android.gms:play-services:11.0.4'
implementation 'com.google.android.gms:play-services-maps:11.0.4'
implementation 'com.google.android.gms:play-services-location:11.0.4'
implementation 'com.google.android.gms:play-services-places:11.0.4'
```

En segundo lugar, debemos obtener una clave API de la consola de Google, para ello debemos ir al siguiente enlace:

<https://developers.google.com/maps/documentation/android-api/signup?hl=es-419#key>

Una vez ahí, seleccionamos obtener una clave

The screenshot shows a web browser displaying the Google Developers website for the Google Maps API. The URL is https://developers.google.com/maps/documentation/android-api/signup?hl=es-419#key. The page title is "Obtener una clave de API". The left sidebar has a "Primeros pasos" section with links like "Configuración de proyectos", "Información general", and "Obtener una clave de API" (which is highlighted). The main content area has a heading "Obtener una clave de API" and a sub-section "Guía rápida para obtener una clave". It contains two steps: "Paso 1: Obtener una clave de API desde la Google API Console" and "Paso 2: Agrega la clave de la API a tu aplicación". A sidebar on the right lists various documentation links related to Google Maps API.

Esto nos dará la opción de crear un nuevo proyecto en la consola de Google, en mi caso, como ya tengo algunos creados me salen para que pueda elegirlos.

The screenshot shows a modal dialog titled "Enable Maps SDK for Android". It asks "Select or create project" and lists existing projects: "+ Create a new project", "KeepMoving", "AcertijosAngular", "chatApp", and "KeepMoving" (which is selected). At the bottom are "CANCEL" and "NEXT" buttons.

Esto nos dará una clave API la cual usaremos para conectar nuestro proyecto a Google Maps

You're all set!

You're ready to start developing with Maps SDK for Android

YOUR API KEY

AIzaSyCBAF3 [REDACTED]



i To improve your app's security, restrict this key's usage in the [API Console](#).

A continuación,

debemos incluir en el AndroidManifest.xml el siguiente código:

```
<meta-data  
    android:name="com.google.android.geo.API_KEY"  
    android:value="YOUR_API_KEY"/>
```

Y debemos cambiar el valor de *YOUR_API_KEY* por la clave obtenida en el anterior paso.

Esto establecerá la clave `com.google.android.geo.API_KEY` en el valor de tu clave de API.

2. Crear el fragmento que extienda de SupportMapFragment

```
<RelativeLayout  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    android:id="@+id/rel_layout"  
    xmlns:android="http://schemas.android.com/apk/res/android"  
>  
  
<fragment xmlns:android="http://schemas.android.com/apk/res/android"  
    xmlns:map="http://schemas.android.com/apk/res-auto"  
    xmlns:tools="http://schemas.android.com/tools"  
    android:id="@+id/map"  
    android:name="com.google.android.gms.maps.SupportMapFragment"  
    android:layout_width="match_parent"  
    android:layout_height="149dp"  
    android:layout_margin="5sp"  
    tools:context="com.example.fermach.keepmovingMapsActivity" />  
  
</RelativeLayout>
```

3. A continuación, debemos de añadir los permisos que sean necesarios en nuestro Manifest, en mi caso son los siguientes:

```
<uses-permission android:name="android.permission.INTERNET" />  
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />  
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />  
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />  
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />  
<uses-permission android:name="com.google.android.providers.gsf.permission.READ_GSERVICES" />  
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />  
<uses-permission android:name="your.package.name.permission.MAPS_RECEIVE" />
```

4. Por último, debemos implementar los métodos necesarios para cargar el MapFragment en las clases que los vayan a utilizar.

Los principales métodos son OnMapReadyCallback, onMapReady(GoogleMaps) y getMapAsync().

- Pantalla de crear quedada.

Para iniciar la gestión de quedadas comenzaré por la pantalla de crear quedada.

Esta pantalla está compuesta de cuatro EditText para introducir los datos de la ubicación, fecha, hora y la información adicional, un Spinner para seleccionar el deporte, un SupportMapFragment y un ScrollableNumberPicker.

En cuanto a clases, la pantalla de crear quedada está formada por una vista con los controles y las vistas con las que interactuará el usuario, un presentador para comunicarse con el repositorio de quedadas, una interfaz que conecta la vista con el presentador, y dos clase de que extienden de DialogFragment las cuales utilizo para abrir un fragmento en el cuál el usuario pueda seleccionar la fecha o la hora de la quedada.

```
public class TimePickerFragment extends DialogFragment{  
  
    private TimePickerDialog.OnTimeSetListener listener;  
  
    public static TimePickerFragment newInstance(TimePickerDialog.OnTimeSetListener  
        listener) {  
        TimePickerFragment fragment = new TimePickerFragment();  
        fragment.setListener(listener);  
        return fragment;  
    }  
  
    public void setListener(TimePickerDialog.OnTimeSetListener listener) {  
        this.listener = listener;  
    }  
  
    @Override  
    public Dialog onCreateDialog(Bundle savedInstanceState) {  
  
        Calendar c= Calendar.getInstance();  
        int hour = c.get(Calendar.HOUR);  
        int minutes = c.get(Calendar.MINUTE);  
  
        return new TimePickerDialog(getActivity(),listener,hour,minutes,false);  
    }  
  
    public class DatepickerFragment extends DialogFragment{  
  
        private DatePickerDialog.OnDateSetListener listener;  
  
        public static DatepickerFragment newInstance(DatepickerDialog.OnDateSetListener  
            listener) {  
    }
```

```

listener) {
    DatePickerFragment fragment = new DatePickerFragment();
    fragment.setListener(listener);
    return fragment;
}

public void setListener(DatePickerDialog.OnDateSetListener listener) {
    this.listener = listener;
}

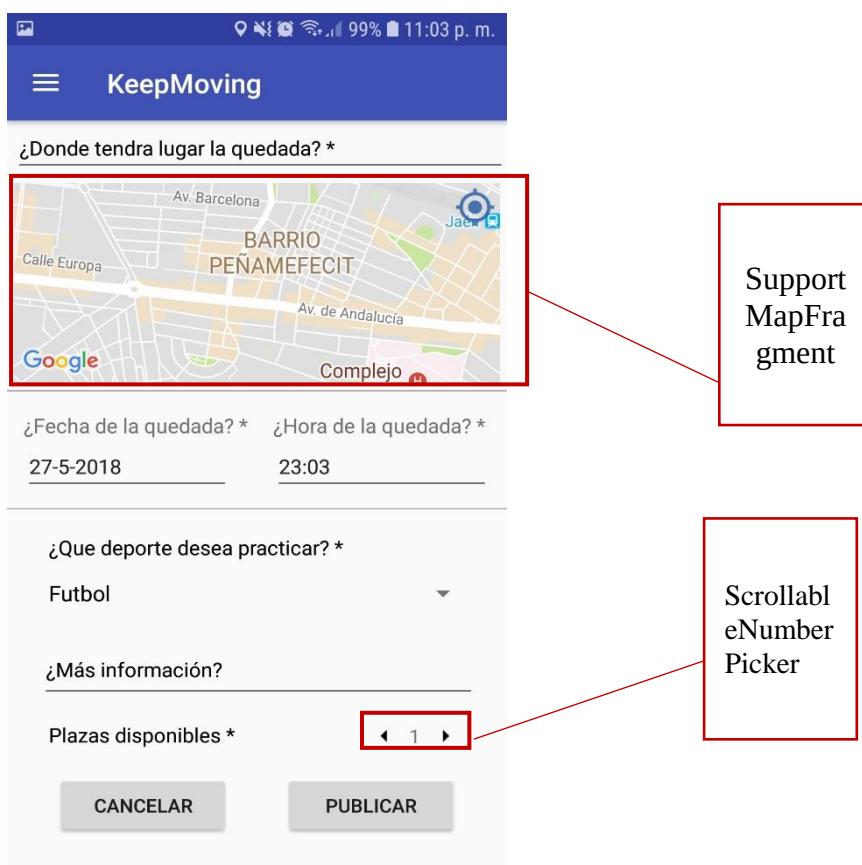
@Override
public Dialog onCreateDialog(Bundle savedInstanceState) {

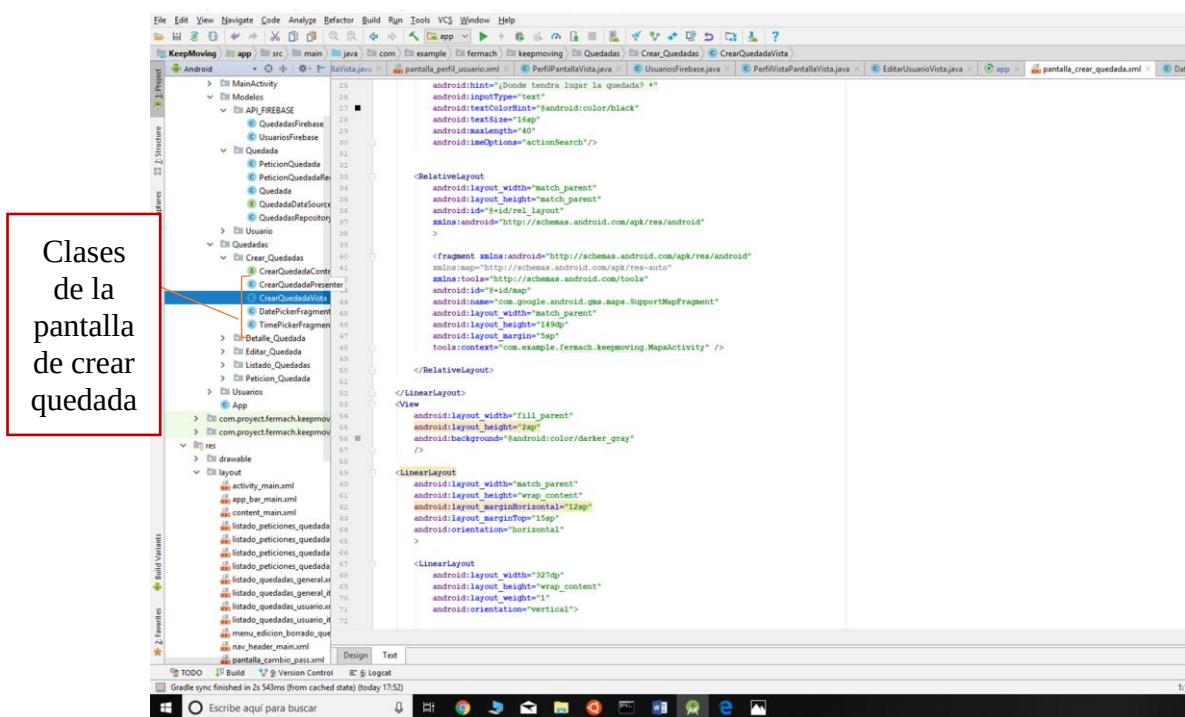
    Calendar c= Calendar.getInstance();
    int year = c.get(Calendar.YEAR);
    int month = c.get(Calendar.MONTH);
    int day = c.get(Calendar.DAY_OF_MONTH);

    return new DatePickerDialog(getActivity(), listener, year, month, day);
}

}

```





Clases de la pantalla de crear quedada

Ya que algunas de la vista que utilice en la pantalla de crear quedada provienen de librerías externas, comenzaré por explicar cómo las he implementado.

En primer lugar, comenzaré explicando como he cargado el mapa utilizando las librerías y los métodos de Google Maps.

Para empezar, necesito pedir los permisos de ubicación para poder establecer por defecto la ubicación actual.

```

String[] permisos = {Manifest.permission.ACCESS_FINE_LOCATION,
                    Manifest.permission.ACCESS_COARSE_LOCATION};
if (ContextCompat.checkSelfPermission(getApplicationContext(), FINE_LOCATION) == PackageManager.PERMISSION_GRANTED) {
    if (ContextCompat.checkSelfPermission(getApplicationContext(), COURSE_LOCATION) == PackageManager.PERMISSION_GRANTED) {
        permisosConcedidos = true;
    }
}
iniciarMaps();
} else {
    ActivityCompat.requestPermissions(getActivity(), permisos, LOCATION_PERMISSIONS_REQUEST_CODE);
}
}
else {
    ActivityCompat.requestPermissions(getActivity(), permisos, LOCATION_PERMISSIONS_REQUEST_CODE);
}

return myView;
}

```

```
@Override
public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions, @NonNull int[] grantResults) {
    permisosConcedidos = false;

    switch (requestCode) {
        case LOCATION_PERMISSIONS_REQUEST_CODE: {
            if (grantResults.length > 0) {
                for (int i = 0; i < grantResults.length; i++) {
                    if (grantResults[i] != PackageManager.PERMISSION_GRANTED) {
                        permisosConcedidos = false;
                        Log.i("MAPS---", "PERMISOS FALSE");
                        Toast.makeText(getApplicationContext(), "No se pudo acceder a la ubicación actual",Toast.LENGTH_SHORT).show();
                        break;
                    }
                }
            }

            Toast.makeText(getApplicationContext(), "Permisos concedidos!",Toast.LENGTH_SHORT).show();

            Log.i("MAPS---", "PERMISOS TRUE");
            permisosConcedidos = true;
            //si se han concedido permisos inicia maps con la ubicacion
            iniciarMaps();
        }else{
            permisosConcedidos = false;
        }
    }
}
```

Una vez pedidos los permisos inicializamos el mapa, en caso de que se hayan concedido nos iniciará el mapa con nuestra ubicación, en el caso contrario nos pondrá una ubicación por defecto.

```
public void iniciarMaps(){

    SupportMapFragment mapFragment = (SupportMapFragment) getChildFragmentManager()
        .findFragmentById(R.id.map);

    mapFragment.getMapAsync(new OnMapReadyCallback() {
        @Override
        public void onMapReady(GoogleMap googleMap) {
            mMap = googleMap;

            if (permisosConcedidos) {
                Log.i("MAPS", "PERMISOS UBICACION CONCEDIDOS");

                if (ActivityCompat.checkSelfPermission(getActivity(),
                    Manifest.permission.ACCESS_FINE_LOCATION) != PackageManager.PERMISSION_GRANTED &&
                        ActivityCompat.checkSelfPermission(getActivity(),
                            Manifest.permission.ACCESS_COARSE_LOCATION) != PackageManager.PERMISSION_GRANTED) {
                    ActivityCompat.requestPermissions(getActivity(),
                        new String[]{Manifest.permission.ACCESS_FINE_LOCATION,
                            Manifest.permission.ACCESS_COARSE_LOCATION}, 1);
                }
            }
        }
    });
}
```

```

Manifest.permission.ACCESS_FINE_LOCATION) != PackageManager.PERMISSION_GRANTED
    && ActivityCompat.checkSelfPermission(getApplicationContext(), Manifest.permission.ACCESS_COARSE_LOCATION) != PackageManager.PERMISSION_GRANTED) {

        return;
    }
    mMap.setMyLocationEnabled(true);
    mMap.getUiSettings().setMyLocationButtonEnabled(true);
    obtenerUbicacion();

} else {

    Toast.makeText(getApplicationContext(), "No se pudo acceder a la ubicación", Toast.LENGTH_SHORT).show();
    Log.i("MAPS", "PERMISOS UBICACION NO CONCEDIDOS");
    LatLng sydney = new LatLng(-34, 151);
    mMap.addMarker(new MarkerOptions().position(sydney).title("Marker in Sydney"));
    mMap.moveCamera(CameraUpdateFactory.newLatLng(sydney));

}
});
```

}

Como se puede apreciar en el método de arriba seleccionamos el botón de ubicación habilitado, para que el usuario pueda acceder a su ubicación en todo momento.

```

public void obtenerUbicacion(){
    mFusedLocationProviderClient=LocationServices.getFusedLocationProviderClient(getApplicationContext());

    try{
        Log.i("MAPS---", "1 ");
        Task ubicacion = mFusedLocationProviderClient.getLastLocation();
        ubicacion.addOnCompleteListener(new OnCompleteListener() {
            @Override
            public void onComplete(@NonNull Task task) {
                if (task.isSuccessful()){
                    Log.i("MAPS---", "1");
                    Location ubicacionActual=(Location)task.getResult();
                    mMap.moveCamera(CameraUpdateFactory.newLatLngZoom(new
LatLang(ubicacionActual.getLatitude(),ubicacionActual.getLongitude()),15f));
                }else{
                    Toast.makeText(getApplicationContext(), "No se pudo acceder a la ubicación actual", Toast.LENGTH_SHORT).show();
                    Log.i("MAPS---", "2");
                }
            }
        });
    }
}
```

```
        }
    });
}

}catch ( SecurityException e){
    Log.e("MAPS_ERROR","getDeviceLocation: "+e.getMessage());
}
}
```

Para buscar la ubicación introducida en el EditText utilizo el siguiente método

En cuanto al ScrollableNumberPicker, sirve para elegir el número de plazas que tendrá mi quedada.

Para implementarlo he utilizado la siguiente librería, implementada en mi gradle:

```
implementation 'com.michaelmuenzer.android:ScrollableNumberPicker:0.2.2'
```

Esta librería me ha permitido poder usar el Picker en el diseño de mi pantalla:

```
<com.michaelmuenzer.android.scrollablenumberpicker.ScrollableNumberPicker
    android:id="@+id/plazas_crear_quedada"
    android:layout_width="match_parent"
    android:layout_height="22dp"
    android:layout_weight="1"
    app:snp_buttonIconLeft="@drawable/menu_left"
    app:snp_buttonIconRight="@drawable/menu_right"
    app:snp_maxValue="50"
    app:snp_minValue="0"
    app:snp_orientation="horizontal"
    app:snp_stepSize="1"
    app:snp_updateInterval="1"
    app:snp_value="1" />
```

Por último, los controles de los botones realizan las siguientes acciones, en caso del botón de cancelar nos lleva de vuelta al fragmento del listado con todas nuestras quedadas, por el contrario, el botón de crear quedada lo que hace es que recoge los datos de los EditText, del Spinner, del Picker, y las coordenadas de la ubicación, y los envía a la API de quedadas de Firebase, pasando por el presentador y el repositorio, para que esta añada una nueva quedada en la base de datos.

```
public void crearQuedada(final Quedada quedada, final CrearQuedadaCallback callback) {
    user = mAuth.getCurrentUser();

    // obtenemos el usuario actual para cargar el nombre al crear la quedada
    UsuariosRef.child(user.getUid()).addListenerForSingleValueEvent(new ValueEventListener() {
        @Override
        public void onDataChange(DataSnapshot dataSnapshot) {
            String nombre = dataSnapshot.child("nombre").getValue(String.class);
            String apellidos = dataSnapshot.child("apellidos").getValue(String.class);
            String correo = dataSnapshot.child("correo").getValue(String.class);
            String biografia = dataSnapshot.child("biografia").getValue(String.class);
            String aficiones = dataSnapshot.child("aficiones").getValue(String.class);
            usuarioActual = new Usuario(nombre, apellidos, correo, biografia, aficiones);
```

```

Log.i("OBTENER USUARIO FIRE", "SUCCESFUL -- " + usuarioActual);
quedada.setAutor("'" + usuarioActual.getNombre() + ", " + usuarioActual.ge-
tApellidos());
quedada.setAutor_uid("'" + user.getId());

//se sube la quedada la quedada
 UsuariosRef.child(user.getId()).child("Quedadas").push().setValue(que-
dada).addOnCompleteListener(new OnCompleteListener<Void>() {
    @Override
    public void onComplete(@NonNull Task<Void> task) {
        if (task.isSuccessful()) {

            Log.i("CREAR_QUEDADA_FIRE1", "EXITO");
            QuedadasRef.push().setValue(quedada).addOnCompleteListener(new
OnCompleteListener<Void>() {
                @Override
                public void onComplete(@NonNull Task<Void> task) {
                    if (task.isSuccessful()) {

                        Log.i("CREAR_QUEDADA_FIRE2", "EXITO");
                        callback.onQuedadaCreada();
                    } else {
                        Log.i("CREAR_QUEDADA_FIRE", "ERROR");
                        callback.onQuedadaCreadaError();
                    }
                }
            });
        } else {
            Log.i("CREAR_QUEDADA_FIRE", "ERROR");
            callback.onQuedadaCreadaError();
        }
    }
});

@Override
public void onCancelled(DatabaseError databaseError) {
    callback.onQuedadaCreadaError();
    Log.i("OBTENER USUARIO FIRE", "ERROR -- " + usuarioActual);
}
});
```

Todas las quedadas se crean con un id único, en mi caso el id de la quedada se crea con la siguiente formula:

```
id = "" + (localizacion.getLongitude() + localizacion.getLatitude()) + "" + fecha + hora;
```

Como podéis observar en el método la quedada se crea tanto en la colección de quedadas como en la colección de quedadas del usuario para que sea más sencillo listar las quedadas posteriormente.

En caso de que la quedada se cree correctamente se nos dirigirá al fragmento con nuestras quedadas, en caso de que no se cree la quedada, nos dirigirá al fragmento de nuestra lista de quedadas y mostrará un mensaje de error.

Pantalla de Editar quedada.

Esta pantalla está formada, al igual que la pantalla de crear quedada, por cuatro EditText para introducir los datos de la ubicación, fecha, hora y la información adicional, un Spinner para seleccionar el deporte, un SupportMapFragment y un ScrollableNumberPicker.

Además también tiene dos botones (Cancelar y Guardar) y sus correspondientes Presentador, Vista e Interfaz.



Clases
De la
pantalla
de editar
quedada

```

if (isEmailValid()) {
    lugar = "" + tv_lugar.getText().toString().trim();
    hora = "" + tv_hora.getText().toString().trim();
    fecha = "" + tv_fecha.getText().toString().trim();
    mas_info = "" + tv_mas_info.getText().toString().trim();
    plazas = "" + picker_plazas.getValue();
    deporte = "" + spinner_deporte.getSelectedItem().toString().trim();

    // Verificar los datos
    if (!lugar.isEmpty() && hora.isEmpty() &&
        !fecha.isEmpty() && !deporte.isEmpty() && !plazas.isEmpty()) {
        // Guardar quedada

        Validator.validador= new Validator();
        if(validator.validateFecha(fecha)){
            if(compararFechaActualCon(fecha_obtenida)) {

                buscarLugar();
                btn_guardar.setEnabled(false);

                // Si la ubicación introducida es valida
                if (ubicacionEncontrada == true) {

                    progressDialog.setMessage("Se están actualizando los datos de la quedada");
                    progressDialog.setCancelable(false);
                    progressDialog.show();

                    longitud = "" + localizacion.getLongitude();
                    latitud = "" + localizacion.getLatitude();
                    // Lugar = (localizacion.getLongitude() + localizacion.getLatitude()) + " " + fecha + hora;

                    quedada = new Quedada(quedada.getId(), quedada.getAutor(), lugar, fecha, hora, deporte, mas_info, plazas,
                        longitud, latitud);

                    presenter.editarQuedada(quedada);
                } else {
                    btn_guardar.setEnabled(true);
                    Snackbar.make(myView, "No se pudo encontrar la ubicación seleccionada!", duration: 4000).show();
                }
            } else {
                AlertDialog.Builder myBuild = new AlertDialog.Builder(getContext());
                myBuild.setMessage("La fecha propuesta de la quedada ya ha expirado.\n\nIntroduzca una fecha válida!");
                myBuild.setTitle("Alerta");
                myBuild.show();
            }
        }
    }
}

```

Al cargar esta pantalla lo que hacemos es que recibimos un objeto de tipo Quedada, con la cual rellenamos el SupportMapFragment partiendo de su longitud y la latitud, y los demás datos de la pantalla con los datos de esta quedada.

En cuanto a los botones de esta pantalla, el botón de cancelar lo que hace es que nos lleva al fragmento con el perfil del usuario. En el caso del botón de guardar, nos actualiza la quedada que hemos recibido, utilizando su id, los datos introducidos y las coordenadas de la ubicación que hay en el fragmento.

```

@Override
public void cambiarEstadoQuedada(Final PeticionQuedada peticionQuedada, Final CambiarEstadoCallback callback) {
    listaPeticionesRef.addListenerForSingleValueEvent(new ValueEventListener() {
        user = mAuth.getCurrentUser();
        Query query = UsuariosRef.child(user.getUserId()).child("Peticiones recibidas").orderByChild("id_peticion")
            .equalTo(peticionQuedada.getId_peticion());
        query.addValueEventListener(new ValueEventListener() {
            @Override
            public void onDataChange(DataSnapshot dataSnapshot) {
                dataSnapshot.getRef().child("estado").setValue(peticionQuedada.getEstado());
                Query query2 = UsuariosRef.child(peticionQuedada.getAutor_peticion()).child("Peticiones enviadas").orderByChild("id_peticion")
                    .equalTo(peticionQuedada.getId_peticion());
                Log.i(tag, "CAMBIO ESTADO", msg: "1");
                query2.addValueEventListener(new ValueEventListener() {
                    @Override
                    public void onDataChange(DataSnapshot dataSnapshot) {
                        dataSnapshot.getRef().child("estado").setValue(peticionQuedada.getEstado());
                        Query query3 = UsuariosRef.child(user.getUserId()).child("Quedadas").orderByChild("id");
                        .equalTo(peticionQuedada.getId());
                        Log.i(tag, "CAMBIO ESTADO", msg: "2");
                        query3.addValueEventListener(new ValueEventListener() {
                            @Override
                            public void onDataChange(DataSnapshot dataSnapshot) {
                                dataSnapshot.getRef().child("plazas").setValue(plazasActualizadas);
                                dataSnapshot.getRef().child("plazas").setVal(plazasActualizadas);
                                Query query4 = UsuariosRef.child(user.getUserId()).child("Quedadas").orderByChild("id")
                                    .equalTo(peticionQuedada.getId());
                                Log.i(tag, "CAMBIO ESTADO", msg: "3");
                                query4.addValueEventListener(new ValueEventListener() {
                                    @Override
                                    public void onDataChange(DataSnapshot dataSnapshot) {
                                        dataSnapshot.getRef().child("plazas").setVal(plazasActualizadas);
                                        Log.i(tag, "CAMBIO ESTADO", msg: "4");
                                        callback.onEstadoCambiado();
                                    }
                                });
                            }
                        });
                    }
                });
            }
        });
    }
}

@Override
public void onChildChanged(DataSnapshot dataSnapshot, String s) {
}

@Override
public void onChildAdded(DataSnapshot dataSnapshot, String s) {
    final String plazasActualizadas = "" + (Integer.parseInt(peticionQuedada.getNumPlazas())
        - Integer.parseInt(peticionQuedada.getNumPlazas_solicitadas()));
    dataSnapshot.getRef().child("plazas").setValue(plazasActualizadas);

    Query query4 = UsuariosRef.child(user.getUserId()).child("Quedadas").orderByChild("id")
        .equalTo(peticionQuedada.getId());
    Log.i(tag, "CAMBIO ESTADO", msg: "3");

    query4.addValueEventListener(new ValueEventListener() {
        @Override
        public void onDataChange(DataSnapshot dataSnapshot) {
            dataSnapshot.getRef().child("plazas").setVal(plazasActualizadas);
            Log.i(tag, "CAMBIO ESTADO", msg: "4");
            callback.onEstadoCambiado();
        }
    });
}

@Override
public void onChildRemoved(DataSnapshot dataSnapshot) {
}

```

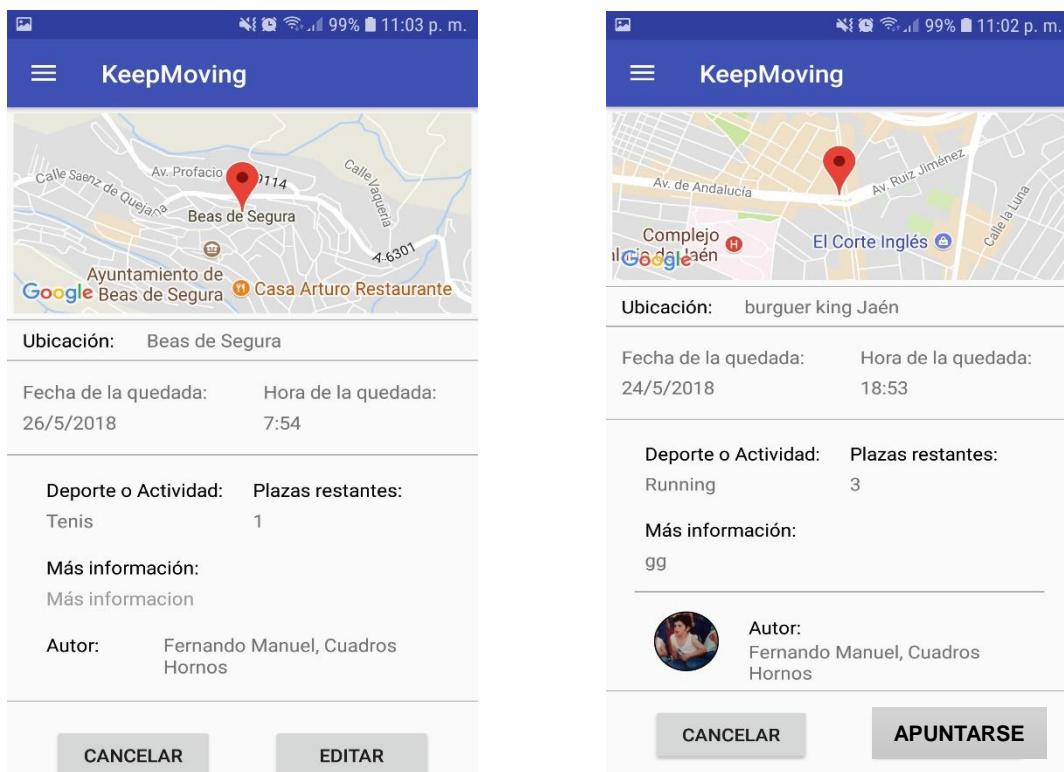
Como podemos ver se modifica la quedada en todas las colecciones en las que esté incluida.

- Pantallas de detalle de la quedada

En ese apartado existen dos pantallas de detalle, el detalle de una quedada del usuario, y el detalle de una quedada de otro autor ajeno.

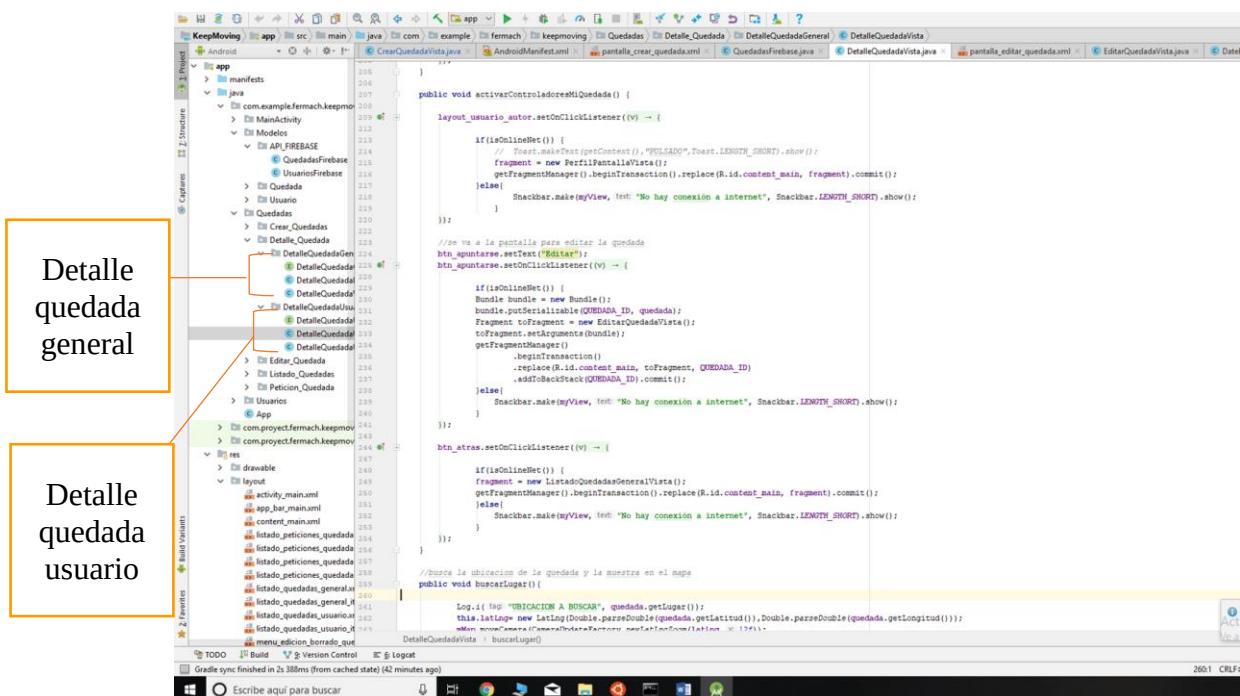
Estas pantallas están divididas en tres clases al igual que las de más que son el Presentador, la Vista y la interfaz que los conecta.

Además, ambas están compuestas de un SupportFragmentManager, el cual se carga con las coordenadas de la quedada que reciben como argumentos en un Bundle, unos TextView para cargar los datos, un Layout con datos sobre el autor de la quedada y dos botones.



*Detalle quedada usuario

*Detalle quedada general



Las principales diferencias entre estas pantallas son las siguientes.

1. La pantalla de detalle general, a diferencia de la pantalla de detalle usuario, implementa en el Layout con datos sobre el autor una foto de este.
2. En el detalle usuario, el botón derecho te lleva a la pantalla para editar la quedada, pasándole como parámetro la quedada.
3. En el detalle general, el botón derecho te lleva a la pantalla de petición quedada pasándole la quedada, aunque antes de esto verifica en la API que ese usuario no tenga ya peticiones a esa misma quedada.

```

public void verificarPeticionQuedada(Quedada quedada, final VerificarPeticionQuedadaCallback callback) {
    user = mAuth.getCurrentUser();
    //final boolean[] existePeticion = {false};

    Query query = UsuariosRef.child(user.getUid()).child("Peticiones enviadas").orderByChild("id")
        .equalTo(quedada.getId());

    Log.i("VERIFICACION A BUSCAR", "QUEDADA ID: "+quedada.getId());

    query.addListenerForSingleValueEvent(new ValueEventListener() {
        @Override
        public void onDataChange(DataSnapshot dataSnapshot) {
            if (dataSnapshot.exists()){
                callback.onQuedadaOcupada();
            }else{

```

```

        callback.onQuedadaLibre();
    }
}

@Override
public void onCancelled(DatabaseError databaseError) {
    }

);
}

}

```

Pantalla petición quedada.

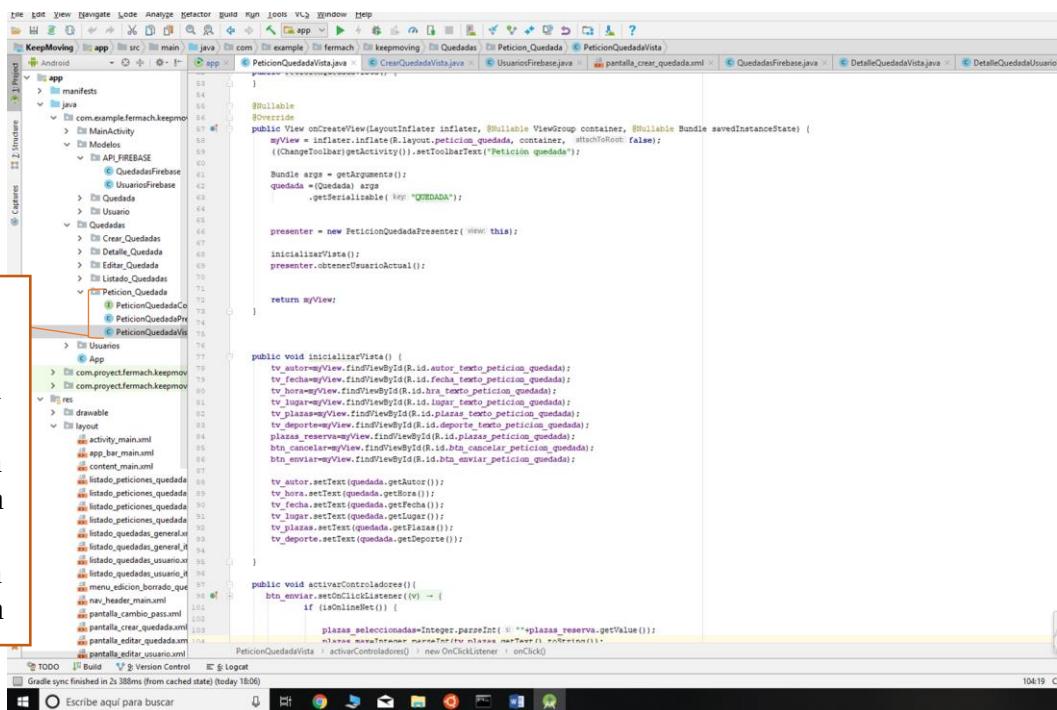
Esta pantalla se utiliza para decir las plazas que quiere reservar un usuario al apuntarse a una quedada y esta compuesta de el ítem de una quedada con los datos básicos sobre esta, enmarcados en TextView, además debajo tiene otro TextView con el autor de la quedada y un ScrollableNumberPicker para que el usuario selecciones las plazas que desea reservar.

Por último, tiene dos botones (cancelar y enviar solicitud).

Esta pantalla, al igual que el detalle de la quedada, recibe un objeto de tipo quedada como parámetro y también cuanta con su interfaz, su vista y su presentador.



Clases de la pantalla de petición quedada de petición quedada



En cuanto a los controles de esta clase, el botón de cancelar nos lleva a la pantalla con el detalle de la quedada, pasándole la quedada como argumento. Por el contrario, el botón de enviar solicitud recoge los datos, crea un objeto petición quedada con un id único y algunos detalles sobre el autor de la petición y lo envía a la API para que cree una nueva petición en la base de datos.

```

public void enviarSolicitud(final PeticionQuedada peticionQuedada, final EnviarSolicitudCallback callback) {
    user = mAuth.getCurrentUser();
    peticionQuedada.setAutor_peticion(user.getUid());
    String peticion_id= peticionQuedada.getAutor_peticion()+peticionQuedada.getId();

    peticionQuedada.setId_peticion(peticion_id);
    UsuariosRef.child(user.getUid()).child("Peticiones enviadas").push().setValue(peticionQuedada).addOnCompleteListener(new OnCompleteListener<Void>() {
        @Override
        public void onComplete(@NonNull Task<Void> task) {
            if (task.isSuccessful()) {
                UsuariosRef.child(peticionQuedada.getAutor_uid()).child("Peticiones recibidas").push().setValue(peticionQuedada).addOnCompleteListener(new OnCompleteListener<Void>() {
                    @Override
                    public void onComplete(@NonNull Task<Void> task) {
                        if (task.isSuccessful()) {
                            Log.i("ENVIAR_SOLICITUD_FIRE", "SUCCESFUL");
                            callback.onSolicitudEnviada();
                        } else {
                            ...
                        }
                    }
                });
            }
        }
    });
}

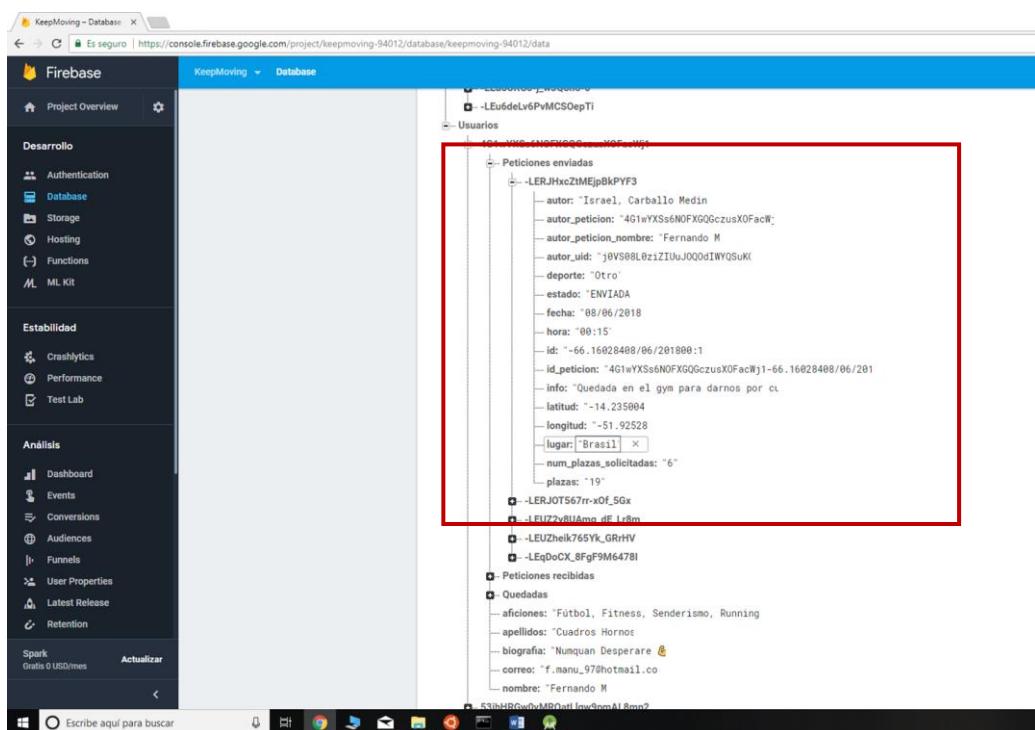
```

```
        Log.i("ENVIAR_SOLICITUD_FIRE", "ERROR_2");
        callback.onSolicitudEnviadaError();
    }
}
});
```

};

```
} else {
    Log.i("ENVIAR_SOLICITUD_FIRE", "ERROR_1");
    callback.onSolicitudEnviadaError();
}
}
});
```

}



Posteriormente nos lleva a la pantalla con el listado de nuestras peticiones enviadas.

-Pantallas de listados

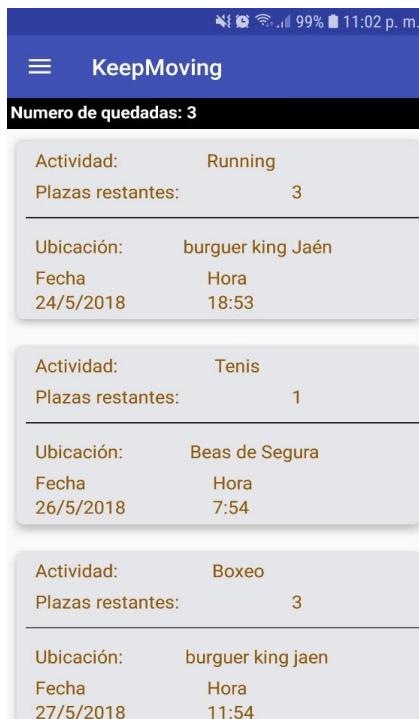
Estas pantallas las he metido en un mismo bloque ya que todas, excepto la pantalla de listado de quedadas recibidas, son bastante similares ya que están basadas en un ListView con su adaptador personalizado para cada una de las quedadas.

- Listado de quedadas generales y listado de quedadas del usuario

Listos listados son prácticamente iguales, la diferencia principal entre estos está en los controles, ya que el listado de quedadas de los usuarios tiene un FloatingActionButton, que sirve para añadir una nueva quedada y un fragmento que se muestra al hacer LongClick sobre la quedada para borrarla o editarla.

El listado de quedadas del usuario sirve para listar todas las quedadas publicadas por el usuario, mientras que el listado de quedadas generales sirve para listar todas las quedadas de todos los usuarios.

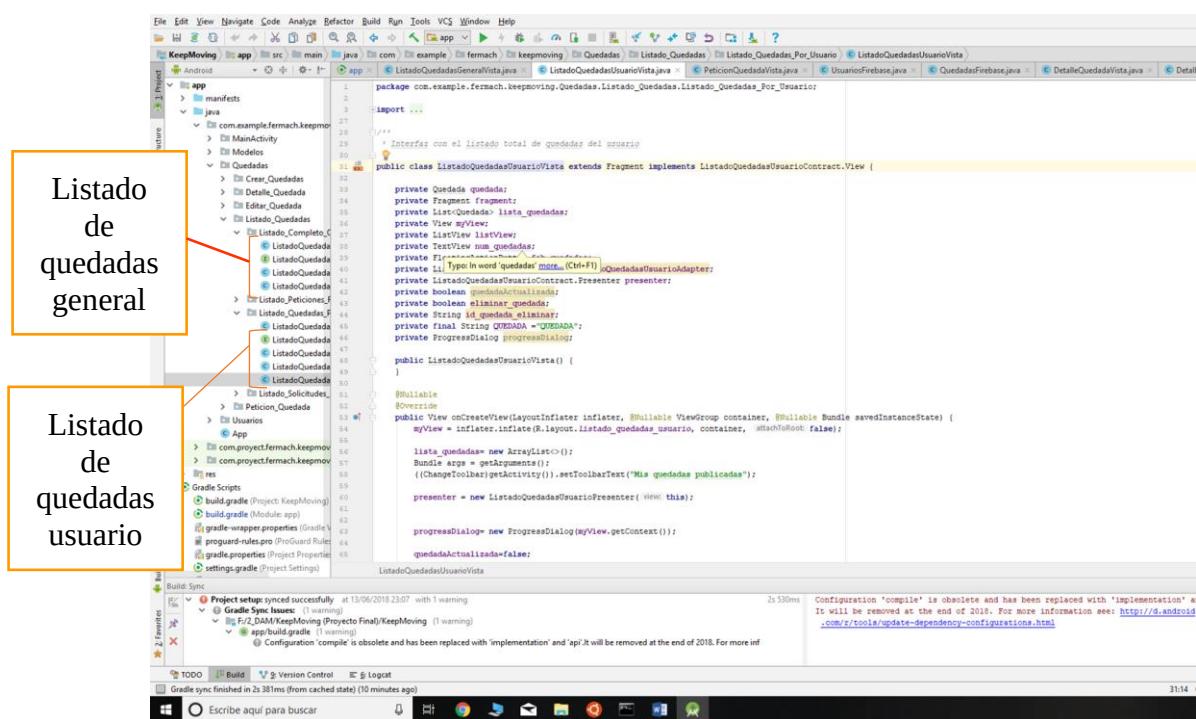
Además, cada uno de estos listados cuenta con sus correspondientes tres clases (vista, presentador e interfaz) y el adaptador para cada una de las quedadas de su lista.



*Listado de quedadas general



*Listado de quedadas del usuario



El siguiente es el adaptador de las quedadas, en ambas listas es prácticamente igual:

```
public class ListadoQuedadasGeneralAdapter extends ArrayAdapter<Quedada> {

    public ListadoQuedadasGeneralAdapter(@NonNull Context context, List<Quedada>
quedadas) {
        super(context, 0, quedadas);
    }

    @NonNull
    @Override
    public View getView(int position, @Nullable View convertView, @NonNull
ViewGroup parent) {
        Quedada quedada = getItem(position);

        if(convertView == null){
            convertView = LayoutInflater.from(getContext()).inflate(R.layout.listado_queda-
das_general_item, parent, false);
        }

        TextView deporte = convertView.findViewById(R.id.deporte_texto_quedada_ge-
neral_item);
        TextView plazas = convertView.findViewById(R.id.plazas_texto_quedada_gene-
ral_item);
        TextView lugar = convertView.findViewById(R.id.lugar_texto_quedada_gene-
ral_item);
        TextView fecha = convertView.findViewById(R.id.fecha_texto_quedada_gene-
ral_item);
        TextView hora = convertView.findViewById(R.id.hra_texto_quedada_gene-
ral_item);

        deporte.setText(quedada.getDeporte());
        plazas.setText(quedada.getPlazas());
        lugar.setText(quedada.getLugar());
        fecha.setText(quedada.getFecha());
        hora.setText(quedada.getHora());

        return convertView;
    }
}
```

En cuanto a los controles, en el listado de quedadas general, al pulsar sobre una quedada, dependiendo de si la quedada la ha publicado el usuario registrado o la ha publicado otro autor nos mostrará el detalle de la quedada del usuario o el detalle de la quedada del autor.

En el listado de quedadas del usuario tenemos un FloatingActionButton, el cual es uno de los controles más populares de las librerías de Material Design, que al pulsarlo no lleva a la pantalla para crear una quedada.

A parte de este FloatingActionButton, tenemos el menú que se muestra al hacer click prolongado sobre el ítem de la quedada, este menú extiende de DialogFragment, y nos muestra dos botones, uno para eliminar la quedada y el otro para editarla.

```

public class ListadoQuedadasUsuarioMenuLClick extends DialogFragment {

    private Quedada mQuedada;
    private ListadoQuedadasUsuarioContract.Presenter presenter;
    private final String QUEDADA_ID = "QUEDADA_ID";
    private final String ELIMINAR_QUEDADA = "ELIMINAR_QUEDADA";
    private final String QUEDADA_A_ELIMINAR_ID = "QUEDADA_A_ELIMI-
NAR_ID";

    public static ListadoQuedadasUsuarioMenuLClick newInstance(Quedada quedada) {

        Bundle args = new Bundle();

        //mandamos la quedada clickeada en nuestra lista
        args.putSerializable("QUEDADA", quedada);
        ListadoQuedadasUsuarioMenuLClick fragment = new ListadoQuedadasUsuarioMe-
nuLClick();
        fragment.setArguments(args);
        return fragment;
    }

    @Override
    public void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        presenter= new ListadoQuedadasUsuarioPresenter();

        //cuando se inicia el menu se recoge la quedada clickeada de nuestra lista
        mQuedada =(Quedada) getArguments()
            .getSerializable("QUEDADA");
    }

    @Nullable
    @Override
    public View onCreateView(LayoutInflater inflater, @Nullable ViewGroup container,
    @Nullable Bundle savedInstanceState) {
        final View v = inflater.inflate(R.layout.menu_edicion_borrado_quedada, container,
        false);

        Button borrar_rutina= v.findViewById(R.id.btn_borrar_quedada_menu);
        Button editar_rutina= v.findViewById(R.id.btn_editar_quedada_menu);

        borrar_rutina.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {

                //Al pulsar en el boton de borrar rutina se crea un dialogo que pregunte
                //si estamos seguros de su borrado
                AlertDialog.Builder myBuild = new AlertDialog.Builder(view.getContext());
                myBuild.setMessage("¿Estás seguro de que deseas eliminar esta que-
                dada?");
                myBuild.setTitle("Eliminar Quedada");
            }
        });
    }
}

```

```

myBuild.setPositiveButton("Sí", new DialogInterface.OnClickListener() {
    @Override
    public void onClick(DialogInterface dialogInterface, int i) {
        //si elegimos Si se cierra el dialogo y el fragmento
        //y realizamos un transaccion al fragmento de la lista de quedada pasan-
dole
        //el id de la quedada a eliminar y una llave para saber que queremos eli-
minar una quedada

        dialogInterface.cancel();
        getDialog().dismiss();
        Bundle args_eli = new Bundle();
        args_eli.putSerializable(QUEDADA_A_ELIMINAR_ID, mQue-
dada.getId());
        args_eli.putSerializable(ELIMINAR_QUEDADA, true);
        Fragment toFragment = new ListadoQuedadasUsuarioVista();
        toFragment.setArguments(args_eli);
        getFragmentManager()
            .beginTransaction()
            .replace(R.id.content_main, toFragment)
            .addToBackStack(ELIMINAR_QUEDADA).addToBa-
ckStack(QUEDADA_A_ELIMINAR_ID).commit();
    }
});
myBuild.setNegativeButton("No", new DialogInterface.OnClickListener() {
    @Override
    public void onClick(DialogInterface dialogInterface, int i) {

        //si elegimos no se cierra el dialogo, y se mantiene el menu abierto
        dialogInterface.cancel();
    }
});
myBuild.show();

}

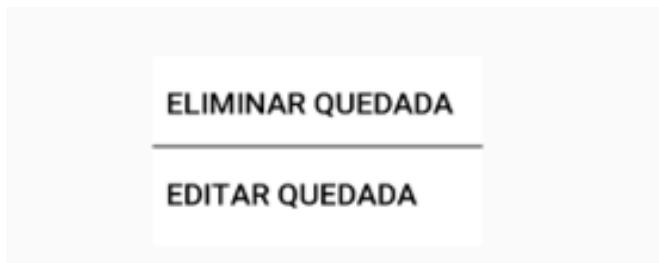
);

editar_rutina.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {

        //Si seleccionamos "Editar quedada" nos lleva al fragmento de editar quedada
        //pasandole el id de la quedada a editar
        getDialog().dismiss();
        Bundle bundle = new Bundle();
        bundle.putSerializable(QUEDADA_ID, mQuedada);
        Fragment toFragment = new EditarQuedadaVista();
        toFragment.setArguments(bundle);
        getFragmentManager()
            .beginTransaction()
            .replace(R.id.content_main, toFragment, QUEDADA_ID)
            .addToBackStack(QUEDADA_ID).commit();
    }
});
return v;
}

```

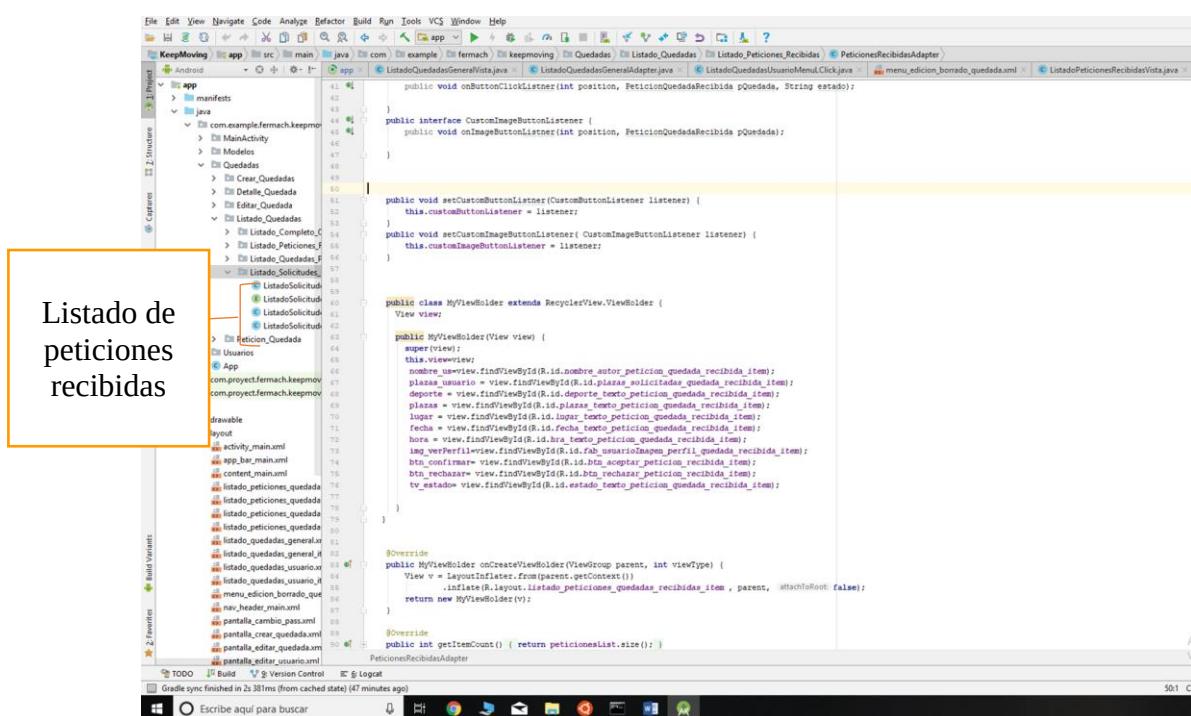
Como podemos comprobar en el adaptador del menú se le pasa como parámetro un objeto de tipo quedada, para posteriormente devolverlo a la vista y que esta llame al método de eliminar o borrar esa quedada, en función del botón clicado, utilizando el Presentador.



- Listado de peticiones enviadas

Este listado simplemente contiene un ListView con sus correspondientes ítems de objetos de tipo PeticionQuedada y en este adaptador, el apartado del estado de cada uno de los ítems se rellena de un color u otro en función de si esta petición se encuentra aceptada, rechazada o enviada.





Listado de peticiones recibidas

Este es el adaptador que utilzo para cada uno de los items de este listado:

```
public class ListadoSolicitudesEnviadasAdapter extends ArrayAdapter<PeticionQuedada> {
```

```
    public ListadoSolicitudesEnviadasAdapter(@NonNull Context context, List<PeticionQuedada> peticionQuedadas) {
```

```
        super(context, 0, peticionQuedadas);  
    }
```

```
    @NonNull  
    @Override  
    public View getView(int position, @Nullable View convertView, @NonNull ViewGroup parent) {  
        PeticionQuedada peticionQuedada= getItem(position);
```

```
        if(convertView == null){  
            convertView = LayoutInflater.from(getContext()).inflate(R.layout.listado_peticiones_quedadas_enviadas_item, parent, false);  
        }
```

```
        TextView deporte = convertView.findViewById(R.id.deporte_texto_peticion_quedada_item);  
        TextView plazas = convertView.findViewById(R.id.plazas_texto_peticion_quedada_item);  
        TextView lugar = convertView.findViewById(R.id.lugar_texto_peticion_quedada_item);  
        TextView fecha = convertView.findViewById(R.id.fecha_texto_peticion_quedada_item);
```

```

    TextView hora = convertView.findViewById(R.id.hra_texto_peticion_quedada_item);
    TextView estado = convertView.findViewById(R.id.estado_texto_peticion_quedada_item);
    LinearLayout layoutEstado=convertView.findViewById(R.id.layoutEstado);

    deporte.setText(peticionQuedada.getDeporte());
    plazas.setText(peticionQuedada.getPlazas());
    lugar.setText(peticionQuedada.getLugar());
    fecha.setText(peticionQuedada.getFecha());
    hora.setText(peticionQuedada.getHora());
    estado.setText(peticionQuedada.getEstado());

    //se modifica el texto y color del layout estado dependiendo del estado de la peticion'
    switch (" "+peticionQuedada.getEstado()){
        case "ENVIADA":
            layoutEstado.setBackgroundColor(Color.YELLOW);
            break;
        case "ACEPTADA":
            layoutEstado.setBackgroundColor(Color.GREEN);
            break;
        case "RECHAZADA":
            layoutEstado.setBackgroundColor(Color.RED);
            break;
    }
    return convertView;
}
}

```

-Listado de peticiones recibidas

Este es el más complejo de los cuatro listados esta formado por un RecyclerView horizontal compuesto por objetos de tipo PeticiónQuedadaRecibida, estos objetos tienen los mismos campos que un objeto de tipo PeticiónQuedada, añadiéndole un Bitmap que será la foto de perfil del autor de la petición.

```

private Bitmap foto;
private String id_peticion;
private String id;
private String autor;
private String autor_uid;
private String lugar;
private String fecha;
private String hora;
private String deporte;
private String info;
private String plazas;
private String longitud;
private String latitud;
private String autor_peticion;
private String autor_peticion_nombre;
private String num_plazas_solicitadas;
private String estado;

```

Esta pantalla está formada por la vista, un presentador para comunicarse con el repositorio y la API, una interfaz y el adaptador para el listado

En cuanto al adaptador de cada uno de los objetos, está formado por una serie de TextView para cada uno de los datos que se muestran en la petición, un CircleImageView, para mostrar la foto del autor de la petición y dos botones para aceptar o rechazar la petición.

Además, en la zona superior tiene un texto que indica el estado de la petición, cuyo color cambia en función del estado.



Clases de la pantalla de peticiones recibidas

```

package com.example.fernach.KeepMoving.Quedadas.Listado_Quedadas.Listado_Peticiones_Recibidas;
import ...
public class PeticionesRecibidasAdapter extends RecyclerView.Adapter<PeticionesRecibidasAdapter.MyViewHolder> {
    private List<PeticionQuedadasRecibida> peticionesList;
    CustomImageButtonListener customImageButtonListener;
    CustomImageButtonListener customImageImageButtonListener;
    Button btn_rechazar;
    TextView tv_estado;
    TextView tv_descrip;
    TextView plazas_usuario;
    TextView deporte ;
    TextView plazas ;
    TextView lugar ;
    TextView fecha ;
    TextView hora ;
    CircleImageView img_verPerfil;
    public interface CustomButtonListener {
        public void onButtonClickListener(int position, PeticionQuedadasRecibida pQuedada, String estado);
    }
    public interface CustomImageButtonListener {
        public void onImageClickListener(int position, PeticionQuedadasRecibida pQuedada);
    }
    public void setCustomButtonListener(CustomButtonListener listener) {
        this.customButtonListener = listener;
    }
    public void setCustomImageButtonListener( CustomImageButtonListener listener) {
        this.customImageImageButtonListener = listener;
    }
}
public void setCustomButtonListener(CustomButtonListener listener) {
    this.customButtonListener = listener;
}
public void setCustomImageButtonListener( CustomImageButtonListener listener) {
    this.customImageImageButtonListener = listener;
}

```

Este es el adaptador que utilzo:

```
public class PeticionesRecibidasAdapter extends RecyclerView.Adapter<PeticionesRecibidasAdapter.MyViewHolder>{

    private List<PeticionQuedadaRecibida> peticionesList;
    CustomButtonListener customButtonListener;
    CustomImageButtonListener customImageButtonListener;
    Button btn_confirmar;
    Button btn_rechazar;
    TextView tv_estado;
    TextView nombre_us;
    TextView plazas_usuario;
    TextView deporte ;
    TextView plazas ;
    TextView lugar ;
    TextView fecha ;
    TextView hora ;
    CircleImageView img_verPerfil;

    //interfaces para saber que item de la lista se esta clickando
    public interface CustomButtonListener {
        public void onButtonClickListner(int position, PeticionQuedadaRecibida pQuedada, String estado);

    }
    public interface CustomImageButtonListener {
        public void onImageButtonListner(int position, PeticionQuedadaRecibida pQuedada);
    }

    public void setCustomButtonListner(CustomButtonListener listener) {
        this.customButtonListener = listener;
    }
    public void setCustomImageButtonListener( CustomImageButtonListener listener) {
        this.customImageButtonListener = listener;
    }

}

public class MyViewHolder extends RecyclerView.ViewHolder {
    View view;

    public MyViewHolder(View view) {
        super(view);
        this.view=view;
        nombre_us=view.findViewById(R.id.nombre_autor_peticion_quedada_recibida_item);
        plazas_usuario = view.findViewById(R.id.plazas_solicitadas_quedada_recibida_item);
        deporte = view.findViewById(R.id.deporte_texto_peticion_quedada_recibida_item);
        plazas = view.findViewById(R.id.plazas_texto_peticion_quedada_recibida_item);
        lugar = view.findViewById(R.id.lugar_texto_peticion_quedada_recibida_item);
        fecha = view.findViewById(R.id.fecha_texto_peticion_quedada_recibida_item);
    }
}
```

```

hora = view.findViewById(R.id.hra_texto_peticion_quedada_recibida_item);
img_verPerfil=view.findViewById(R.id.fab_usuarioImagen_perfil_quedada_re-
cibida_item);
btn_confirmar= view.findViewById(R.id.btn_aceptar_peticion_recibida_item);
btn_rechazar= view.findViewById(R.id.btn_rechazar_peticion_recibida_item);
tv_estado= view.findViewById(R.id.estado_texto_peticion_quedada_reci-
bida_item);

}

@Override
public MyViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
    View v = LayoutInflater.from(parent.getContext())
        .inflate(R.layout.listado_peticiones_quedadas_recibidas_item , parent, false);
    return new MyViewHolder(v);
}

@Override
public int getItemCount() {
    return peticionesList.size();
}

public PeticionesRecibidasAdapter(List<PeticionQuedadaRecibida> peticionesList) {
    this.peticionesList = peticionesList;
}

@Override
public void onBindViewHolder(final MyViewHolder holder, final int position) {
    final PeticionQuedadaRecibida p = peticionesList.get(position);
    Log.i("HOLA",p.toString());
    nombre_us.setText(p.getAutor_peticion_nombre());
    plazas_usuario.setText(p.getNum_plazas_solicitadas());
    deporte.setText(p.getDeporte());
    plazas.setText(p.getPlazas());
    lugar.setText(p.getLugar());
    fecha.setText(p.getFecha());
    hora.setText(p.getHora());
    tv_estado.setText(p.getEstado());

    if(p.getFoto()!=null) {
        img_verPerfil.setImageBitmap(p.getFoto());
    }

    btn_confirmar.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            if (customButtonListener != null) {
                customButtonListener.onButtonClickListner(position, p, "ACEPTADA");
            }
        }
    });

    btn_rechazar.setOnClickListener(new View.OnClickListener() {
        @Override

```

```

public void onClick(View v) {
    if(customButtonListener!=null){
        customButtonListener.onButtonClickListner(position,p,"RECHAZADA"
    );
    }
};

img_verPerfil.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        if(customImageButtonListener!=null){
            customImageButtonListener.onImageButtonListner(position,p );
        }
    }
);
// Log.i("HOLAAAAAAA","HOLAA");

// Dependiendo del estado setea unos colores y un texto diferentes en el item de la peti-
cion
switch (""+p.getEstado()) {
    case "ENVIADA":
        tv_estado.setBackgroundColor(Color.YELLOW);
        btn_confirmar.setEnabled(true);
        btn_rechazar.setEnabled(true);
        btn_confirmar.setBackgroundColor(holder.view.getResources().getColor(android.R.color.darker_gray));
        btn_rechazar.setBackgroundColor(holder.view.getResources().getColor(android.R.color.darker_gray));
        break;
    case "ACEPTADA":
        tv_estado.setBackgroundColor(Color.GREEN);
        btn_confirmar.setEnabled(false);
        btn_confirmar.setBackgroundColor(Color.GREEN);
        btn_rechazar.setBackgroundColor(android.R.drawable.btn_default);
        btn_rechazar.setEnabled(false);
        break;
    case "RECHAZADA":
        tv_estado.setBackgroundColor(Color.RED);
        btn_confirmar.setEnabled(false);
        btn_rechazar.setBackgroundColor(Color.RED);
        btn_rechazar.setEnabled(false);
        btn_confirmar.setBackgroundColor(android.R.drawable.btn_default);
        break;
}
}
}

```

Para controlar cada uno de los ítems de mi lista de forma independiente, es decir, que cuando el usuario interactúe con uno la interfaz sepa que ítem ha seleccionado, utilizo unas interfaces intermedias, las cuales creo en el adaptador, como se puede ver arriba, posteriormente al seleccionar un elemento desde el listado llamo a es interfaz.

```

@Override
public void onButtonClickListner(int position, PeticionQuedadaRecibida pQuedada,
String estado) {
    if(isOnlineNet()) {

        peticionQuedada=new PeticionQuedada(pQuedada.getId(),pQuedada.getAutor_peticion_nombre(),pQuedada.getAutor(),pQuedada.getAutor_uid(),pQuedada.getLugar(),pQuedada.getFecha(),pQuedada.getHora(),pQuedada.getDeporte(),pQuedada.getInfo(),pQuedada.getPlazas(),pQuedada.getLongitud(),
        pQuedada.getLatitude(),pQuedada.getNum_plazas_solicitadas(),estado );
        peticionQuedada.setAutor_peticion(pQuedada.getAutor_peticion());
        peticionQuedada.setId_peticion(pQuedada.getId_peticion());

        Log.i("CAMBIANDO ESTADO ", "NUEVA PETICION CON CAMBIO DE ESTADO: " + peticionQuedada.toString());

        String fecha_obtenida= ""+peticionQuedada.getFecha()+" "+peticionQuedada.getHora();
        if(compararFechaActualCon(fecha_obtenida)) {
            presenter.cambiarEstadoQuedada(petitionQuedada);

        }else{
            AlertDialog.Builder myBuild = new AlertDialog.Builder(getContext());
            myBuild.setMessage("La fecha propuesta de la quedada ya ha expirado.\n\nNo es posible confirmar o rechazar esta quedada!");
            myBuild.setTitle("Alerta");

            myBuild.setNegativeButton("Cerrar", new DialogInterface.OnClickListener() {
                @Override
                public void onClick(DialogInterface dialog, int which) {
                    dialog.cancel();
                }
            });
            myBuild.show();
        }

    }else{
        Snackbar.make(myView,"No hay conexión con internet", Snackbar.LENGTH_SHORT).show();
    }
}

@Override
public void onImageButtonListner(int position, PeticionQuedadaRecibida pQuedada) {
    if(isOnlineNet()) {

        Bundle bundle = new Bundle();
        bundle.putSerializable(UID_USUARIO, pQuedada.getAutor_peticion());
    }
}

```

```

Fragment toFragment = new PerfilVistaPantallaVista();
toFragment.setArguments(bundle);
getFragmentManager()
    .beginTransaction()
    .replace(R.id.content_main, toFragment, UID_USUARIO)
    .addToBackStack(UID_USUARIO).commit();
}else{
    Snackbar.make(myView, "No hay conexión con internet", Snackbar.LENGTH_SHORT).show();
}

}

```

Al clicar en el botón de aceptar o rechazar la quedada se comprueba que la fecha de la quedada peticionada no halla expirado, y a continuación cambiamos el estado de la petición a aceptado o rechazado y actualizamos el listado.

Al seleccionar la imagen del autor de la petición nos lleva a el fragmento con su perfil.

6.4. MEJORAS EN LA FUNCIONALIDAD Y DISEÑO

En esta ultima etapa del desarrollo de la aplicación lo que se hizo fue que se implementaron me nuevos controles de los textos y las vistas, junto con otras mejoras con el fin de obtener una aplicación más atractiva de cara al usuario.

Estas son algunas de las mejoras más significativas:

- Se implementa una interfaz que esconde el Navigatión Drawer en las pantallas de logueo y registro del usuario.
- Se implementa una interfaz que esconde la barra de ajustes en las pantallas de logueo y registro del usuario y, además, cambia el texto de la barra de tareas dependiendo de la pantalla donde se encuentre el usuario.
- Se implementan las librerías de pullrefreshlayout para refrescar la vista al deslizar el listado de quedadas publicadas hacia abajo.

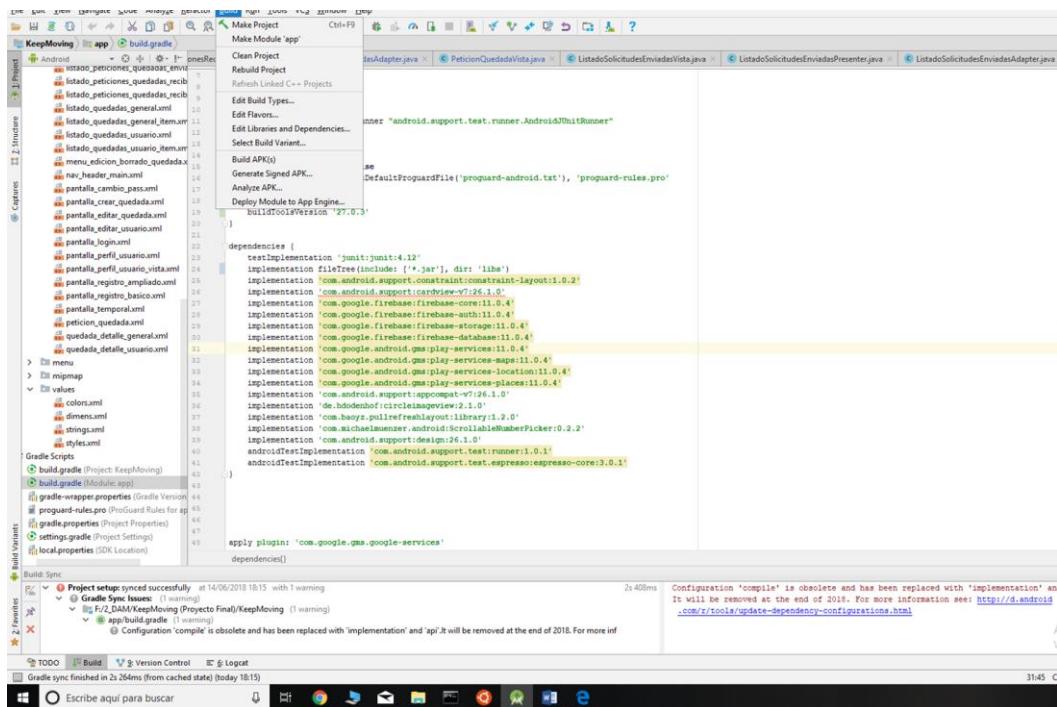
implementation 'com.baoyz.pullrefreshlayout:library:1.2.0'

- Se mejora el control sobre los EditText y los datos, implementando nuevas expresiones regulares y condicionales.
- Se implementa una pantalla para recuperar contraseña en caso de que el usuario halla olvidado su contraseña.
- Me mejora el diseño general de la aplicación, sacándole más potencial a las librerías de Material Design.

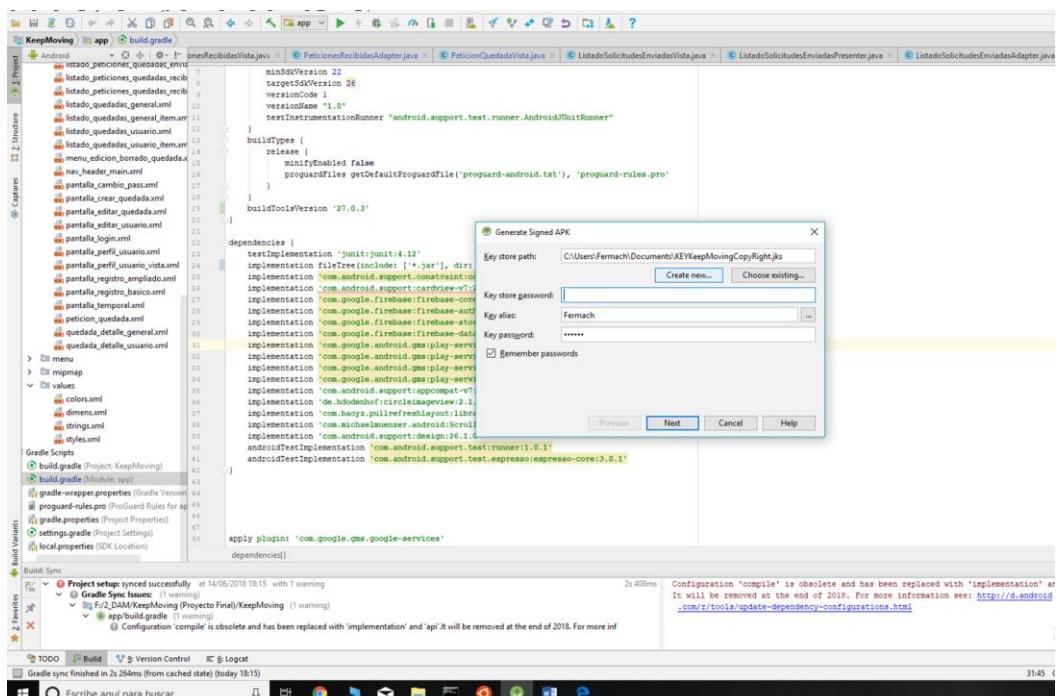
6.5. EXPORTANDO LA APLICACIÓN

Una vez que teníamos la primera versión de la aplicación, el último paso es exportar la apk con todas sus librerías, para ello que seguidos los siguientes pasos:

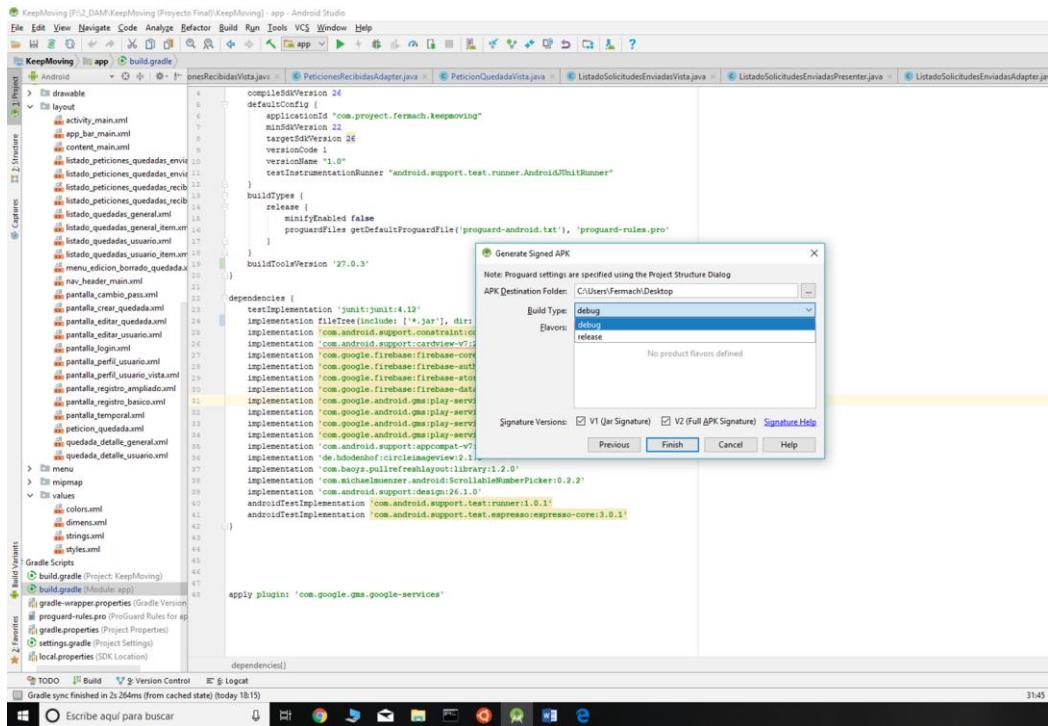
1. Crear la Key Store para poder compartir la aplicación.
Para ello debemos seleccionar *Build> Generate Signed APK*



Una vez hecho esto nos abrirá el siguiente menú donde debemos seleccionar “Create new...” y seleccionar el repositorio a continuación donde deseamos guardarla, en mi caso ya tengo una creada.



2. Seleccionamos siguiente y a continuación la ruta donde queremos guardar la APK. También se nos pedirá que escojamos el Build Type, aquí existen dos opciones, Debug y Release, la principal diferencia entre ambas que la versión Debug no puedes subirla a Google Play y a Release sí.



3. Una vez hecho esto nos generará una .apk en la ruta indicada el cual podremos instalar en nuestro teléfono móvil.

7. TIEMPO DE EJECUCIÓN

Debido a que he organizado el desarrollo de mi proyecto basándome en la metodología Scrum combinada con el repositorio GitLab, utilizaré los imágenes de los Milestones y los Issues, es decir las tareas programadas, para visualizar el tiempo empleando en el desarrollo de la aplicación.

El primer commit del proyecto se hizo el día 30 de marzo, aun que realmente el proyecto se inició antes, ya que antes de ese commit ya se inicio el desarrollo, algunos bocetos y alguna documentación.

A continuación, voy a detallar el tiempo invertido en cada uno de los bloques principales de tareas o Milestones:

La fecha de inicio y fin se solapan en ocasiones ya que a veces estaba desarrollando dos bloques simultáneamente.

Milestone	Logear usuarios y detalle
Fecha Inicio	22/03/2018
Fecha Fin	27/04/2018
Dificultades	La principal dificultad para desarrollar este bloque de tareas fue la gestión de usuario, y la forma de conectar la referencia del mismo usuario tanto en la base de datos, como en la autenticación y el storage.

Milestone	Gestión de quedadas
Fecha Inicio	27/04/2018
Fecha Fin	20/05/2018
Dificultades	La principal dificultad para desarrollar este bloque de tareas fue la creación de las quedadas y la implementación de Google Maps

Milestone	Listado de quedadas
Fecha Inicio	6/05/2018
Fecha Fin	22/05/2018
Dificultades	La principal dificultad para desarrollar este bloque de tareas fue la implementación de los CardViews y la gestión de las quedadas en la base de datos

Milestone	Solicitudes de plazas quedadas/usuarios
Fecha Inicio	15/05/2018
Fecha Fin	3/06/2018
Dificultades	La principal dificultad para desarrollar este bloque de tareas fueron los adaptadores de los listados, destacando el adaptador de peticiones recibidas.

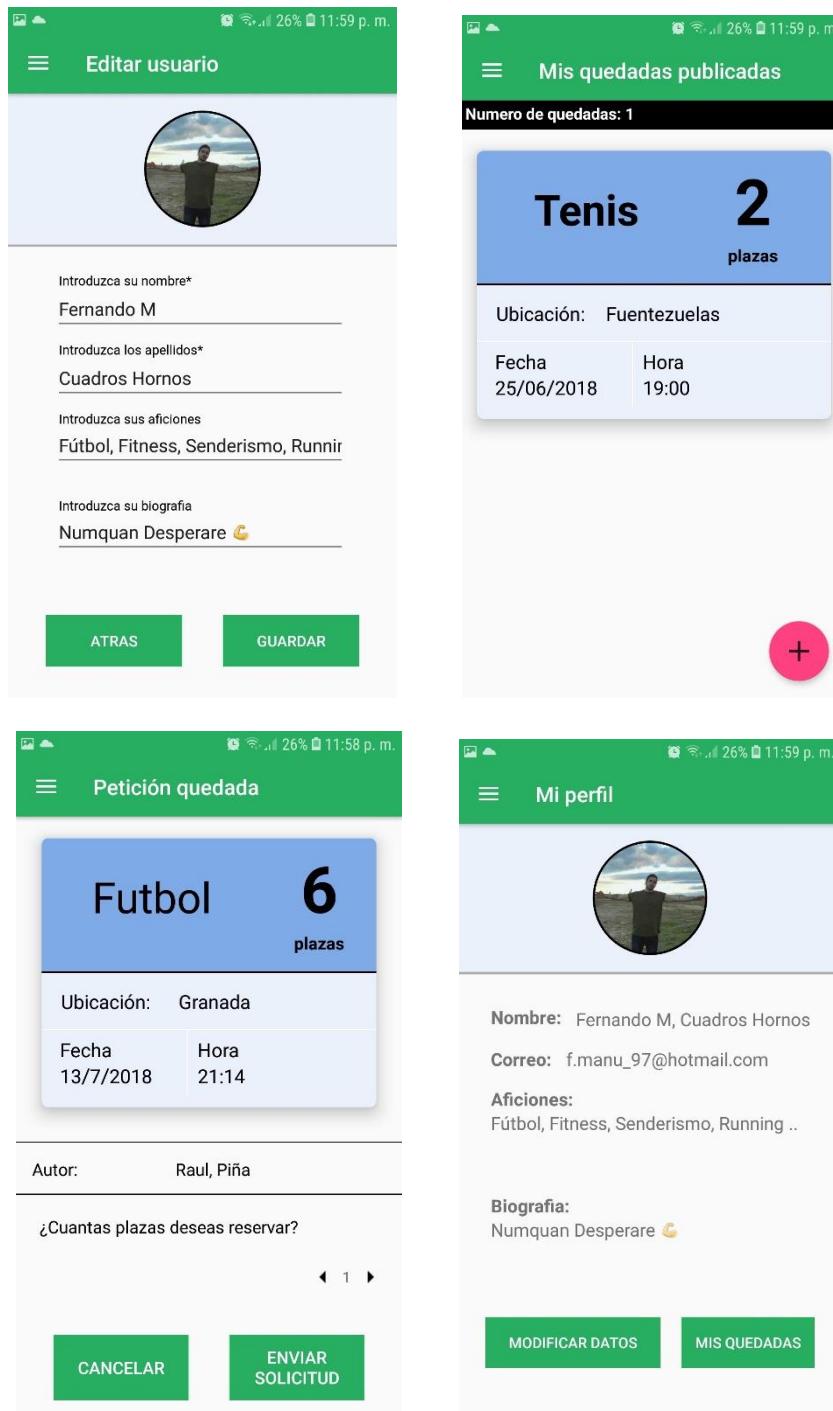
*En este bloque aun quedan tareas pendientes que implementaré en un futuro como una nueva versión.

Milestone	Loguear usuarios y detalle
Fecha Inicio	22/05/2018
Fecha Fin	20/06/2018 – Aún en desarrollo
Dificultades	Las principales dificultades para desarrollar este bloque de tareas fueron las mejoras en el diseño de la aplicación

La versión actual del proyecto es a **versión 1.5.0** y su desarrollo ha sido finalizado el día **17/Junio/2018**.

8. RESULTADOS

Estas son algunas capturas de la primera versión de mi aplicación KeepMoving:





Screenshot 1: Publicar quedada

¿Donde tendra lugar la quedada? *


¿Fecha de la quedada? * 14/06/2018 ¿Hora de la quedada? * 23:59

¿Que deporte deseas practicar? * Futbol

¿Más información?

Plazas disponibles * 1

CANCELAR PUBLICAR

Screenshot 2: Peticiones enviadas

Numero de solicitudes enviadas: 6

Actividad:	Skate
Plazas restantes:	36
Ubicación:	Los prados
Fecha	Hora
08/06/2018	20:15

ENVIADA

Actividad:	Otro
Plazas restantes:	5
Ubicación:	Jaén
Fecha	Hora
9/6/2018	10:49

ACEPTADA

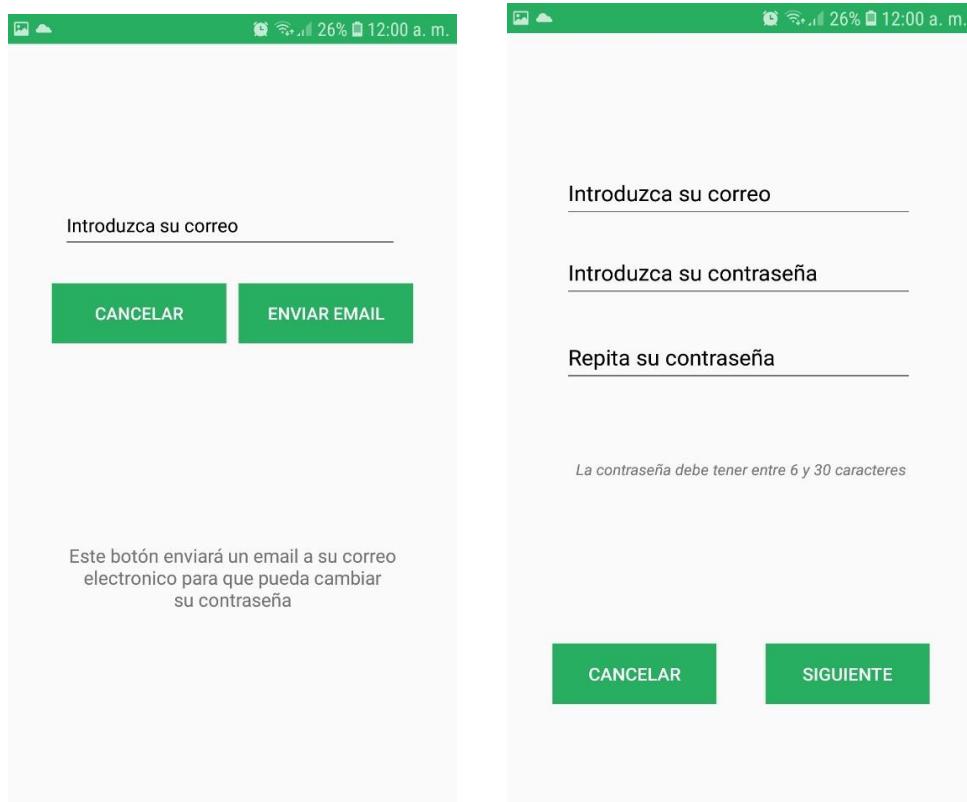
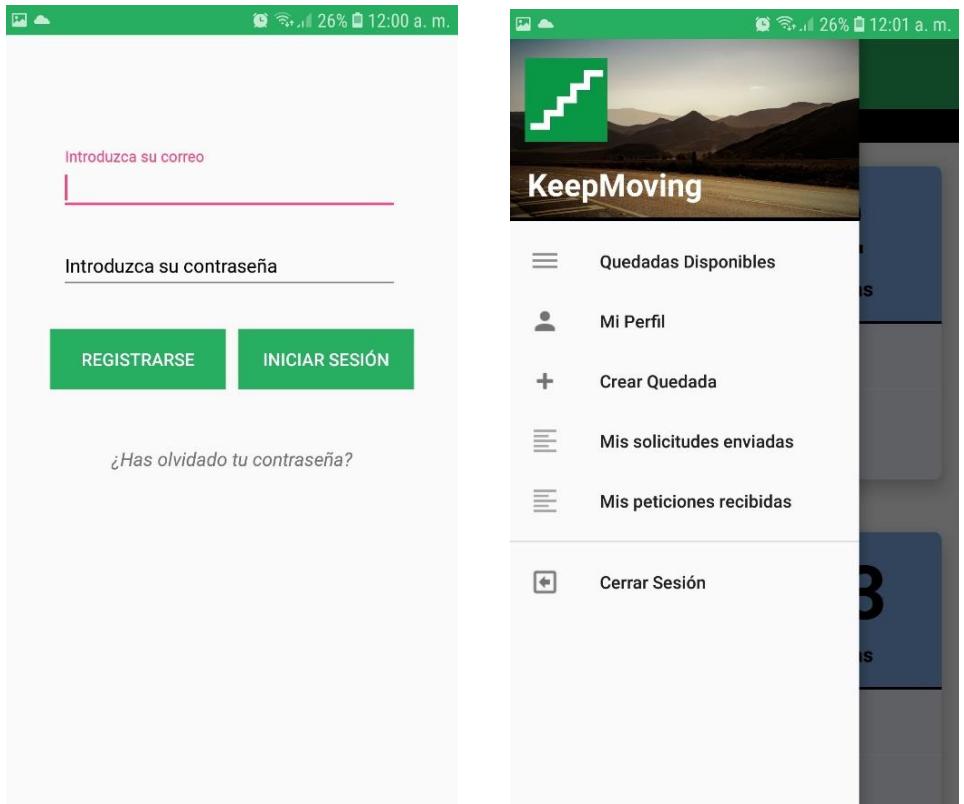
Introduzca su nombre*


Introduzca los apellidos*

Introduzca sus aficiones

Introduzca su biografia

ATRAS REGISTRARSE



9. CONCLUSIONES

Cuando comencé el proyecto tenía la idea de hacer una red social en la que los usuarios pudieran organizar quedadas, practicar deportes y conocer gente, todo esto sacándole el máximo potencial a las librerías de Firebase, combinándolas con las librerías de Material Design y de Google Maps.

Finalmente he obtenido lo que quería, cumpliendo con la gran mayoría de las características que, en el inicio, quería que implementase mi aplicación.

Por falta de tiempo no he podido perfeccionar la aplicación todo lo que me hubiera gustado, pero a pesar de ello, en esta primera versión de la aplicación he obtenido un resultado muy satisfactorio y que sin duda cumple con mis expectativas, aparte de todo lo aprendido en el desarrollo de este proyecto.

Principales características pendientes para futuras versiones:

- Chat de usuarios
- Varios Idiomas
- Registro con Facebook
- Filtro de búsqueda en el listado de quedadas
- Mostrar la distancia a la que se encuentra una quedada
- Optimizar más el código

10. BIBLIOGRAFÍA

- Documentación de Google

<https://developers.google.com/maps/android/?hl=es-419>
<https://firebase.google.com/docs/android/setup?hl=es-419>

- StackOverFlow

<https://es.stackoverflow.com/>

- ElAndroidLibre

<https://elandroidelibre.elespanol.com/2010/06/los-foros-de-android-en-espanol.html>

- Documentación de Android

<https://developer.android.com/design/material/?hl=es-419>
<https://developer.android.com/?hl=es-419>

- YouTube