Ejemplo de implementación en Android con Firebase.

Objetivos del proyecto: 2
- Sistema de autenticación con Firebase. 2
- Manejo de Realtime Database. 4
- Manejo de Cloud Storage 6

Problemas durante la implementación 8
Conclusión. 8

Turno de mañana. Grupo de trabajo 11.

Componentes: Pablo Borreguero Llorente.

Objetivos del proyecto:

Los objetivos del proyecto consisten en la implementación de las siguientes funcionalidades.

- Sistema de autenticación con Firebase.
  - Creación de cuenta e inicio de sesión.
  - Cerrar sesión.
  - Borrado de cuenta.

- Manejo de Realtime Database.
  - CRUD de objetos JSON

- Manejo de Cloud Storage.
  - CRUD de imágenes.

Sistema de autenticación con Firebase.

Firebase proporciona un sistema de autenticación haciendo uso bien de una cuenta de correo y una contraseña o bien de una cuenta gmail y un token asociado.
En este proyecto he optado por la utilización de una cuenta gmail para su implementación.

Para hacer uso de ésta tecnología, primero debemos crear un proyecto en Firebase y conectar nuestro proyecto de Android Studio siguiendo el asistente que encontramos en la consola de Firebase (https://console.firebase.google.com/). Tras ésto, podemos encontrar en la documentación que nos proporcionan todo el código necesario para implementar nuestro sistema de autenticación (https://firebase.google.com/docs/auth/android/start?authuser=1).

Para empezar, debemos inicializar en nuestro método **onCreate** el asistente de Google que nos permitirá elegir con qué cuenta queremos iniciar sesión en nuestra aplicación y una instancia de **FirebaseAuth** (mAuth en nuestro proyecto) con la que comprobaremos si ya hemos iniciado sesión antes.

En nuestro método **onStart**, mAuth.getCurrentUser() devolverá null si no tenemos usuario creado o si hemos cerrado sesión, lo cual nos mostrará la vista de Login. En caso de que tengamos usuario activo, nos llevará a nuestra vista principal de la aplicación.

En nuestra vista de Login, el botón **Sign in** nos lanza el asistente de Google para elegir cuenta y una vez elegida, obtenemos un ID que el método **firebaseAuthWithGoogle** intercambia por una credencial de Firebase que usamos para iniciar sesión y obtener los

datos del usuario que inicia sesión. Ésto nos servirá para añadir su usuario a nuestra base de datos.

Cuando cerramos la aplicación sin cerrar sesión, ésta credencial es la que comprueba el método onStart para comprobar si existe el usuario. En caso de cerrar sesión, deberemos obtener otra para iniciar sesión.

Si queremos borrar nuestra cuenta, tanto la credencial de Firebase como el ID de nuestra cuenta Google serán borrados.

Manejo de Realtime Database.

Realtime Database ofrece un sistema de almacenamiento de objetos JSON cuyos valores pueden ser de tipo String, Long, Double, Boolean, Map<String, Object> y List<Object>.
Es muy versátil y cómoda de usar y ofrece multitud de funciones con las que diseñar tus queries, además de la opción de añadir lógica interna a la query.

En este proyecto abordaremos las funciones de creación, lectura con y sin filtro, actualización y borrado de datos.

Lo primero que necesitamos es una referencia a nuestra base de datos. Ésto lo obtenemos con una instancia de DatabaseReference y los métodos .getInstance() y .getReference(). Nuestra referencia quedaría:

```
DatabaseReference ref = FirebaseDatabase.getInstance().getReference("nodoInicial");
```

- **Creación de objeto**

  Una vez creada nuestra referencia, podemos crear entradas a la misma usando el método **ref.push()**, el cual nos generará una entrada con clave aleatoria ordenada lexicográficamente.
  Con nuestra entrada creada, el método **ref.push().setValue(objeto)** nos añadiría nuestro objeto a dicha entrada.
  Si queremos especificar la clave de nuestra entrada deberemos hacer uso del método **ref.child("claveElegida").setValue(objeto)**.

- **Lectura de objetos**

  Si queremos obtener los datos de una entrada determinada, tenemos dos opciones. Podemos optar por definir un agente de escucha activa a la localización del objeto o un agente de escucha para una sola vez.

  Si queremos escucha activa, cada vez que se produzca un cambio en nuestra entrada, se ejecutará la query y nos devolverá los datos actualizados. Éste método es de la forma:
  ref.child("claveElegida").addValueEventListener(new ValueEventListener() {//...});
  y deberemos implementar el método onDataChange() y onCancelled(), los cuales nos devolverán respectivamente un snapshot de nuestros datos o un mensaje de error.

  Si por el contrario queremos ejecutar la query una sola vez, usaremos la forma

ref.addListenerForSingleValueEvent(new ValueEventListener() {//…}); debiendo implementar los mismos métodos onDataChange() y onCancelled().
Tanto en el agente de escucha activo como en el de un sólo uso, podemos añadir la lógica que creamos conveniente al margen de las funciones facilitadas por Firebase para adaptar las búsquedas a nuestras necesidades

- **Actualizar objetos**

  Si deseamos actualizar un objeto existente en nuestra base de datos, podemos hacerlo usando la función
  ref.child("claveElegida").updateChildren(Map<String, Object> updates)
  la cual contendrá un mapa con los nuevos valores para las claves secundarias del objeto.

- **Eliminar Objeto**

  Para eliminar objetos tenemos 3 opciones:

  ref.child("claveElegida").removeValue()
  ref.child("claveElegida").updateChildren(Map<String, null> updates)
  ref.child("claveElegida").setValue(null)

Tanto para la creación, actualización o borrado de objetos, podemos añadir a nuestros métodos agentes de escucha que nos den feedback sobre si la operación tuvo éxito o no. Estos son los métodos
.addOnSuccessListener(new OnSuccessListener<Void>() {//...});
.addOnFailureListener(new OnFailureListener() {//...});

Manejo de Cloud Storage

Otra base de datos utilizada en este proyecto es **Cloud Storage**, la cual he usado para almacenar imágenes asociadas a los objetos de Realtime Database. Éstas colecciones de imágenes estarán contenidas en carpetas, cada una con la ID del usuario que las subió.

Para la creación, lectura, actualización y borrado de datos necesitaremos, al igual que on Realtime Database, una referencia a nuestra BD:

StorageReference **ref** = FirebaseStorage.*getInstance*().getReference("nodoInicial");

- **Creación de objetos**

  Para crear objetos en Cloud Storage basta con usar el método **ref.child(userId).child("idObjeto").putFile(objeto).**

  Para recuperar la url del objeto creado, antes debemos guardar la operación en una variable a la que añadiremos un agente de escucha que nos avise del éxito de la operación. Dentro de este agente, deberemos implementar los métodos necesarios para recuperar la url, la cual podremos guardar en una variable para adjuntarla más tarde a nuestro objeto de Realtime Database.

- **Lectura de objetos**

  Para este proyecto, no he usado los métodos explicados en la documentación de Firebase para guardar las imágenes en el dispositivo. En lugar de eso, he hecho uso de la librería **Glide** de Google, la cual recupera datos de una url proporcionada y los guarda en el contenedor que le especifiquemos.

  Glide.*with*(contexto).load(Uri.*parse*(uriDelObjeto)).into(contenedor);

- **Actualización de objetos**

  Para actualizar objetos, podemos hacer uso del mismo método para la creación de objetos.
  La única diferencia sería cambiar los campos de .child("idObjeto") y .putFile(objeto) con los valores nuevos.

- **Borrado de objetos**

  De modo similar a Realtime Database, nuestra petición para borrar objetos de una localización concreta sería:

  ref.child(userId).child(“idObjeto”).delete();

Problemas durante la implementación

Durante el desarrollo del proyecto, tuve problemas en lo referente al borrado de usuarios al no ejecutarse en orden las queries.
El problema consistía en que al llegar al borrado de imágenes del usuario, la función no disponía de las ids de los objetos ya que el método encargado de obtenerlas aún no había terminado de ejecutarse, por lo que se borraba todo menos las imágenes en Cloud Storage.
Esto era debido a que son funciones asíncronas y se resolvió separando en un método aparte la recopilación de ids de imágenes y dentro de su método onSuccess, hacer la llamada al resto de métodos de borrado de datos.

Conclusión.

Al haber estado usando estas tecnologías, me he dado cuenta de la inmensa cantidad de opciones que ofrece Firebase y que aún habiendo contemplado los casos básicos, con más tiempo pueden llegar a hacerse cosas realmente interesantes, tales como:

- Configurar funciones que se ejecuten en el back-end cuando determinados eventos se produzcan en tu BD.
- Enviar emails a usuarios cuando se produzca un intento de inicio de sesión inesperado en su cuenta.
- Uso de stripe para pagos.

También dispone de la base de datos **Firestore**, la cual presentan como la “hermana mayor” de Realtime Database al haber arreglado algunos problemas de latencia, aunque no he investigado mucho este tema.

Realtime Database me ha parecido una BD muy útil para aplicaciones que supongan interacción continua entre muchos usuarios debido a la rapidez con la que trabaja.
Otro aspecto interesante ha sido el hecho de que si no tienes conexión, guarda en local la información y la envía cuando ésta esté otra vez disponible.

Sin duda es una tecnología que volveré a usar tratando de aprovecharla al máximo.