

A dark blue vertical bar on the left side of the page. A blue arrow points to the right from the bar, containing the date.

23-05-2022

ALCLOUD

Proyecto Fin de Ciclo en Desarrollo
de Aplicaciones Multiplataforma.
S2DAM

Several thin, curved, light blue lines that sweep upwards from the bottom left towards the center of the page.

Alberto Luján Cadenas
I.E.S AZARQUIEL

Índice

Introducción.....	2
Especificación de requisitos	2
Planificación, tareas y evaluación de costes	3
Estudio de la aplicación.....	4
Arquitectura de la aplicación.....	5
Costes del desarrollo y despliegue de la aplicación	6
Coste para el servicio <i>Authentication</i>	7
Coste para el servicio <i>Realtime Database</i>	7
Coste para el servicio <i>Cloud Storage</i>	7
Tecnologías principales utilizadas	8
<i>Firebase Authentication</i>	8
<i>BoM</i> de <i>Firebase</i> para <i>Android</i>	9
Librería para Autenticación de <i>Firebase</i>	9
<i>Sign-in method</i>	10
<i>Firebase Realtime Database</i>	14
<i>Firebase Storage</i>	16
<i>Google Cloud</i>	19
Otras tecnologías.....	22
<i>Picasso</i>	22
<i>Kotlin Coroutines</i>	23
<i>Retrofit-Gson</i>	23
<i>ArchLifecycle</i>	23
Desarrollo e implementación.....	23
Código de autenticación <i>Firebase Authentication</i>	24
Código de autenticación por <i>Google</i>	24
Código para la subida de archivos a <i>Firebase Storage</i>	28
Conclusiones y líneas futuras	31
Conclusiones	31
Líneas futuras.....	31
Referencias bibliográficas	32

Introducción

ALCLOUD es una aplicación Android que se basa en *Firebase* de *Google*, que es una plataforma destinada a desarrolladores web y de aplicaciones en la que facilitan el trabajo a los mismos para desplegar sus propias aplicaciones.

En este caso se han utilizado dichos servicios, concretamente: *Firebase Authentication*, *Firebase Realtime Database* y *Firebase Storage*. Así, todo esto dota a ALCLOUD de ser una aplicación rápida, segura y consistente en tiempo real, cuyo uso se basa en una nube.

Será esta nube donde los clientes podrán subir sus archivos a los servidores de *Google*, en el lugar que el desarrollador los ubique y los trate para que el usuario tenga sus documentos, ficheros, imágenes y mucho más en cualquier lugar y en cualquier momento.

Siguiendo en esta línea, es este el motivo por el que me he decidido a este proyecto: todas las aplicaciones más populares y destacadas por cualquier tipo de público están basadas en la nube. Desde cualquier servicio de *Google*, hasta el propio *WhatsApp*, se podría considerar como una nube. Es por esto por lo quiero realizar esta *app*; para conocer cómo funcionan, cómo se tratan los datos, cómo se manejan estas aplicaciones desde la perspectiva de un programador y la complejidad que podría llevar este tipo de proyectos. Porque, aunque es cierto que, si pensamos en la funcionalidad como tal de estas *apps*, podría parecernos sencilla, ya que solo es subir archivos y bajarlos, desde el punto de vista de un programador, sabemos que no solo es eso; también es necesario saber manejar el sistema, saber manejar esos datos y saber hacer, sobre todo, que el cliente tenga la sensación de fluidez y seguridad que todos queremos para nuestra información, puesto que al fin y al cabo, lo que obtiene uno mismo al usar estas aplicaciones es la gratificación y comodidad de que nuestros datos están a salvo.

Por tanto, la finalidad última de ALCLOUD será transmitir al usuario estas sensaciones y ofrecer la garantía de que todo estará a buen recaudo y disponible siempre que se desee. Todo ello, para usuarios *Android*, por el momento.

Especificación de requisitos

Los requisitos son una parte fundamental de la aplicación, ya que, en función de estos, podremos obtener un mayor o menor número de clientes para el proyecto. A continuación, se citarán algunos de ellos más importantes para el uso de ALCLOUD:

- Sistema operativo basado en *Android* → Versión 5.0 en adelante. Es importante destacar que aún no se sabe la compatibilidad con la última API que lanzó *Google* (*Android* 13), puesto que actualmente se encuentra en fase Beta y no ha sido testada para esta última versión.
- Conexión continua a internet, ya sea para su uso o visualización de los datos.
- Acceso al almacenamiento del dispositivo para todo tipo de archivos.
- Servicios de *Google* para autenticación del usuario.

Planificación, tareas y evaluación de costes

Teniendo como objetivo entregar mi proyecto para el día 10 de junio, he planteado dividir el tiempo restante en diferentes etapas con determinadas finalidades. Así, la primera de ellas tenía como objetivo la elección del propio PFC. Para ello comencé con una “*BrainStorm*”, que fui desarrollando posteriormente. De esta manera tenía varias opciones posibles:

- **AlcFy:** fue la primera idea que planteé para realizar mi PFC, consistía en una aplicación *Android* con posibilidad de realizarla web, en la que podía reproducirse música en *streaming* conectándose a un servidor, el cual contenía los archivos de las canciones en formato mp3. De esta forma, existirán dos modalidades de reproducción: bien de manera aleatoria o bien que sea el propio usuario quien elija la canción a reproducir.
- **App para negocios de restauración:** con motivo de la terrible pandemia que todos sufrimos desde 2020 ocasionada por el virus COVID-19, la población de todo el mundo ha sufrido numerosas restricciones para poner freno a su propagación, lo que ha hecho que tuviésemos que adaptar nuestros hábitos a dichas limitaciones. Es así como se me ocurrió una nueva idea para mi PFC: aprovechar un aparato electrónico, el cual consistía en una placa de Arduino y varios sensores de temperatura, de CO₂ y de humedad, y una pantalla LCD capacitiva, que mostraba los valores que recogían dichos sensores. Con este dispositivo y a través de Arduino, lo conectaría a un servidor y enviaría los datos correspondientes para que, mediante una *app Android* o web se mostraran en tiempo real los valores obtenidos. Así, de esta manera podría colocarse este aparato en locales cerrados como, por ejemplo, establecimientos del ámbito de la restauración, y controlar esos valores mediante la aplicación, estableciendo unos

parámetros que sirvieran para cuantificar el riesgo de contagio por CO₂. La finalidad de esto sería adoptar una serie de medidas encaminadas a regular estos datos, como, por ejemplo, mayor ventilación, menor densidad de personas, etc. avisando de ello al personal y a la clientela para una correcta acción conjunta.

- **WhatsFire:** esta idea fue la que casi utilicé de base para realizar mi PFC, ya que era el proyecto que más avanzado tenía y en el que más confiaba, pero por problemas en el desarrollo de la *app* no pude continuarlo. Se trataba de una réplica de la aplicación *WhatsApp*, con mensajería instantánea y subida de archivos.
- **ALCLOUD:** este proyecto surgió de la convicción de que actualmente la mayoría de aplicaciones se encuentran estrechamente relacionadas con la nube. De esta manera, observé cómo los usuarios valoraban positivamente el hecho de que sus datos estuviesen asegurados frente a posibles complicaciones, como problemas de *software* del teléfono móvil (virus, por ejemplo), o accidentes que dañaran el *hardware* del mismo (rotura de pantalla, pérdida de la función táctil, etc.), que impidiesen el acceso a sus archivos. Por tanto, decidí crear una nueva aplicación basada en *Firebase* de *Google*, con similitudes del comportamiento de su análoga en dicha compañía (*Google Drive*).

Como desde un principio la planificación no fue más que desarrollar la aplicación *Android* e ir avanzando en su lógica día tras día en mis tiempos libres después de trabajar en las prácticas y esto lo hacía con todas las ideas que fueron surgiendo en mi “*BrainStorm*”, tuve que replantearme la estrategia y realizar un *planning*, ya que si no iba a ser un desbarajuste y no podía continuar de la forma en que lo realizaba por los costes de tiempo que suponía.

Por ello, después de decidir cuál sería el proyecto que quería llevar a término, resolví destinar dos horas al desarrollo de la aplicación, con quince minutos de descanso entre hora, durante todos los días de la semana, para así poder realizar el estudio y arquitectura de la misma y, a continuación, trabajar en ello.

Estudio de la aplicación

1. Mi punto de partida fue observar las aplicaciones más populares que se encontraban en el sector *Android* similares a ALCLOUD, comparando estas con mis ideas

predefinidas mentalmente y así poder realizar un pequeño estudio del mercado para saber qué demandaban los clientes.

2. A continuación, tras obtener los resultados de estas pesquisas, procedí a la esquematización de los mismos, clasificándolos en dos columnas: ventajas e inconvenientes. De esta forma, conseguí una visión rápida y global de los datos a tener en cuenta para el desarrollo e incorporación a mi aplicación.
3. Después, realicé un diseño general en el que estructuraba las funcionalidades que iba a tener ALCLOUD según lo aplicado. Así mismo, lo especificaba posteriormente, describiendo las conexiones entre las distintas funciones que incluye la *app*.
4. Una vez acabada la evaluación de los datos obtenidos, procedí al diseño técnico. Aquí lo que pretendía era que esas funciones que detallaba fueran capaces de funcionar de una manera correcta y eficiente, para que así la aplicación se ejecutara de una manera consistente en la mayoría de los dispositivos compatibles.
5. Finalmente, cuando acabé el estudio procedí a la implementación. En este punto del proyecto, el objetivo era que, a través de la estructura conseguida, se aplicase la lógica de negocio al código; es decir, programar esas funcionalidades requeridas para la *app*.

Una vez completado el estudio de la aplicación, es cuando me dispongo a la arquitectura de la misma.

Arquitectura de la aplicación

1. Esta es la fase de desarrollo, en la cual tengo que traducir todos los datos obtenidos anteriormente al código que dará vida a la aplicación. Así, esta parte del proceso consistirá en implementar las etapas anteriores, dándose comienzo a las pruebas. Por tanto, en este apartado me dedicaré a crear los diferentes paquetes que tendrá la aplicación, adaptándose así el estudio realizado previamente a la misma. En mi caso concreto, decidí usar el paradigma *Model View-View Model*, ya que se utilizan mucho los hilos, lo que dota de mayor eficacia a la aplicación.
2. En dichas pruebas se espera que cada una de las funciones se ejecuten correctamente para ir descartando las posibles incongruencias con las demás unidades; así de esta forma, podía volver a realizar el estudio de las demás funciones sin tener que volver a empezar.

En el caso concreto de ALCLOUD, esto se traduce en realizar las funciones una a una (aunque algunas son subsidiarias de otras), para comprobar que todo se está implementando correctamente. Es por ello que, cuando una fallaba, solo tenía que estudiar esa unidad concreta para corregirla, no siendo necesario reiniciar todo el proyecto.

3. Una vez que la anterior fase haya tenido éxito es la hora de la integración, donde todos los elementos funcionales se combinan y se coordinan en un mismo conjunto para consolidar la aplicación.

En la práctica, esto quiere decir que, una vez tenga todas las funciones comprobadas y ejecutándose correctamente, procederé a su integración con la aplicación. Así, tendrán lugar funciones como realizar los controles encargados de ejecutar una unidad en concreto, las pantallas de carga, las relaciones entre unidades para obtener un mismo resultado, etc.

Costes del desarrollo y despliegue de la aplicación

En cuanto al coste del desarrollo de la aplicación, solo cabe mencionar que ha sido nulo; no ha habido coste alguno para llevar a cabo la aplicación debido a que las diversas tecnologías utilizadas ofrecían un servicio de prueba gratuito que me ha resultado accesible por estar en posesión de una cuenta educativa/corporativa. Así, se me ha permitido realizar este proyecto en dicho formato de prueba, y aunque por ser gratuito contiene algunas limitaciones, ha sido más que suficiente para llevar a cabo todo lo esperado y hacer posible ALCLOUD.

Pese a que estas tecnologías se describirán más adelante, aquí expongo una tabla con los costes reales si la aplicación hubiese llegado a su proceso de producción y mantenimiento. Del mismo modo, aunque existen más planes, he tomado como referencia el que a continuación expongo debido a que considero que es el que más se ajustaría a la realidad. Aun así, en función del crecimiento de la aplicación se podrían escoger otras alternativas.

Coste para el servicio *Authentication*

AUTHENTICATION	Plan Spark (Educativo/Corporativo)	Plan Baze (Prepago)
Autenticación telefónica: EE.UU., India y Canadá	10.000 por mes	\$0.01 por verificación
Autenticación telefónica: Todos los demás países	10.000 por mes	\$0.06 por verificación
Otros servicios de autenticación	Incluido sin coste	Incluido sin coste

Coste para el servicio *Realtime Database*

REALTIME DATABASE	Plan Spark (Educativo/Corporativo)	Plan Baze (Prepago)
Conexiones simultaneas	100	200,000 por base de datos
GB Almacenados	1GB	\$5 por GB
GB Descargados	10GB por mes	\$1 por GB
Varias bases de datos por proyecto	No incluido	Incluido sin coste

Coste para el servicio *Cloud Storage*

CLOUD STORAGE	Plan Spark (Educativo/Corporativo)	Plan Baze (Prepago)
GB Almacenados	5GB	\$0.026 por GB
GB Descargados	1GB por día	\$0.12 por GB
Operaciones de carga	20.000 por día	\$0.05 por cada 10.000
Operaciones de descarga	50.000 por día	\$0.04 por cada 10.000
Varios buckets por proyecto	No incluido	Incluido sin coste

Tecnologías principales utilizadas

Como he comentado anteriormente, las tecnologías que he utilizado para llevar a cabo este proyecto han sido varias, aunque destaca sobre las demás por su papel fundamental *Firebase* y sus distintos servicios como: *Firebase Authentication*, *Firebase Cloud Storage* y *Firebase Realtime Database*.

Desde su inicio, me decanté por estos servicios debido a la facilidad y a la gran documentación que tiene *Google* para poder dotar a las aplicaciones de los desarrolladores de un conjunto de técnicas avanzadas, las cuales otorgarán la utilidad al programador, y proporcionarán al cliente unas experiencias ricas en cuanto al uso de la aplicación y al desarrollo.

Además, decidí sumarle el lenguaje *Kotlin* debido a que me ayudaría más a usar estas tecnologías, ya que para la programación *Android* se usa dicho lenguaje hoy en día, pese a que anteriormente el más utilizado era *Java*. *Kotlin* es un lenguaje fácil de aprender porque su sintaxis se asemeja mucho a *Java* (básicamente deriva de él, ya que fue inventado para solventar ciertos problemas que tenía *Java* a la hora de desarrollar aplicaciones *Android*).

De este modo, a continuación, describiré la función de cada uno de los servicios implementados y su uso en el proyecto:

Firebase Authentication

Como es obvio, es obligatorio usar un servicio de autenticación en este tipo de proyectos porque lo que se quiere conseguir es que cada cliente tenga su espacio personal y asegurado en la *app* y así cumplir con los objetivos de dicho proyecto.

Pueden usarse varios tipos de autenticación, desde el clásico correo electrónico hasta el número de teléfono y *apps* integradas. En esta ocasión la aplicación cuenta con el método de correo electrónico y contraseña asociado a él y la autenticación directa por *Google*, haciendo así más rápida y segura la verificación para la identificación de la aplicación.

Los datos proporcionados por el usuario para el registro son asegurados por *Google* y sus servicios, cumpliendo de esta manera con las normativas vigentes, como, por ejemplo, que el desarrollador cuando acceda al *backend* no puedan ver datos sensibles (como las contraseñas de cada usuario).

Para la implementación de *Firebase Authentication* en este proyecto se han utilizado las librerías propias de *Google*, las cuales se encuentran en su portal web de *Firebase*. Dichas librerías son:

BoM de Firebase para Android

Se trata de la librería principal de *Firebase*, que permite administrar las versiones de las bibliotecas de los distintos servicios de *Firebase*. Dicha librería hay que colocarla en el archivo **app/build.gradle** a nivel de aplicación de nuestro proyecto, tal como lo indica la propia documentación. Esta es la parte de código que nos facilita el portal web:

```
dependencies {  
    // Import the BoM for the Firebase platform  
    implementation platform('com.google.firebase:firebase-bom:30.0.0')  
  
    // Declare the dependencies for the desired Firebase products without  
    // specifying versions  
    // For example, declare the dependencies for Firebase Authentication  
    // and Cloud Firestore  
    implementation 'com.google.firebase:firebase-auth'  
    implementation 'com.google.firebase:firebase-firestore'  
}
```

Librería para Autenticación de Firebase

Este paquete proporcionado por el servicio principal nos permite hacer uso de las distintas funciones que hay que utilizar para hacer posible el registro de usuarios y posteriores accesos de los mismos a nuestra aplicación, de tal manera que cuando el cliente realiza cualquiera de las dos acciones, estas se quedan reflejadas en la consola del proyecto para el desarrollador. Para poder usar estas librerías y programar nuestras funciones dentro del desarrollo del proyecto, se necesita la implementación de la librería que se coloca en el archivo **app/build.gradle**, *Firebase* nos sugiere lo siguiente:

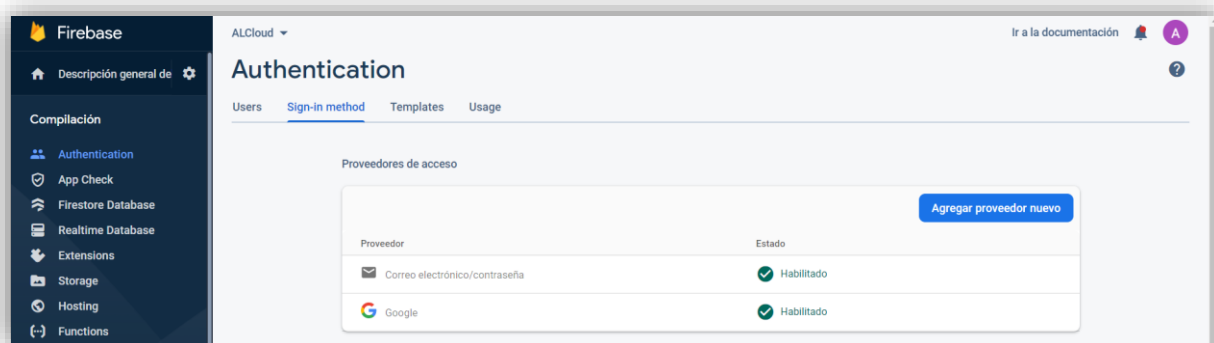
```
dependencies {  
    // Declare the dependency for the Firebase Authentication library  
    // When using the BoM, you don't specify versions in Firebase  
    library dependencies  
    implementation 'com.google.firebase:firebase-auth-ktx'  
}
```

Una vez que se declaran las librerías de *Autenticación* y *BoM*, tenemos que configurar en la consola del proyecto los distintos métodos de autenticación que tendrá las aplicaciones

asociadas al servicio. Para ello debemos acceder a través de la web a nuestra consola y manipular estos apartados:

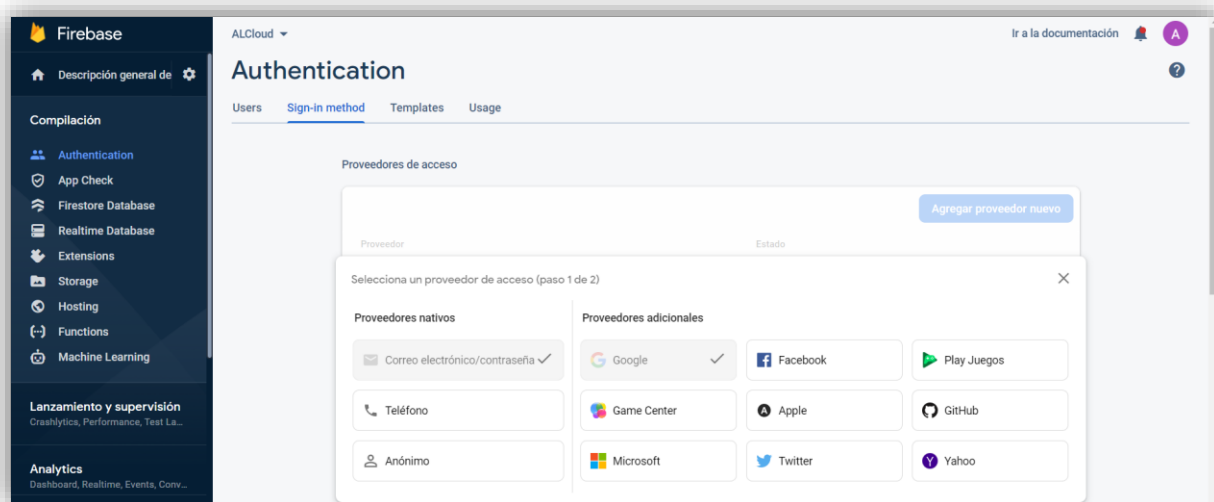
Sign-in method

Es el apartado que nos permite seleccionar qué proveedor de acceso tendrá el proyecto; en este caso, para ALCLOUD se ha usado dos proveedores, el de *Google* y Correo electrónico/contraseña.

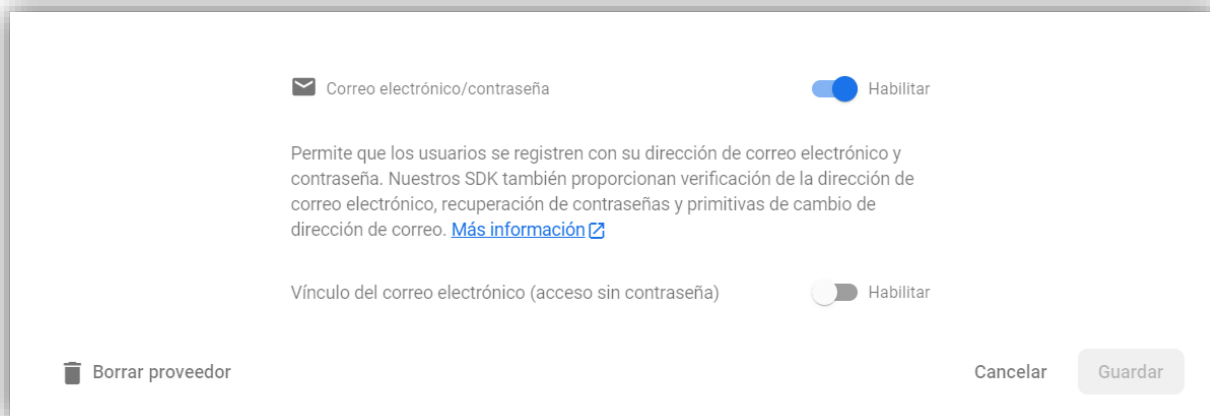


Firebase Auth 1 Configuración Proveedores

Para agregar proveedores nuevos, tenemos que pulsar en el botón de **Agregar proveedor nuevo** y nos mostrará un cuadro con las distintas opciones con las que cuenta Firebase.




Firebase Auth 2 Agregar Proveedor



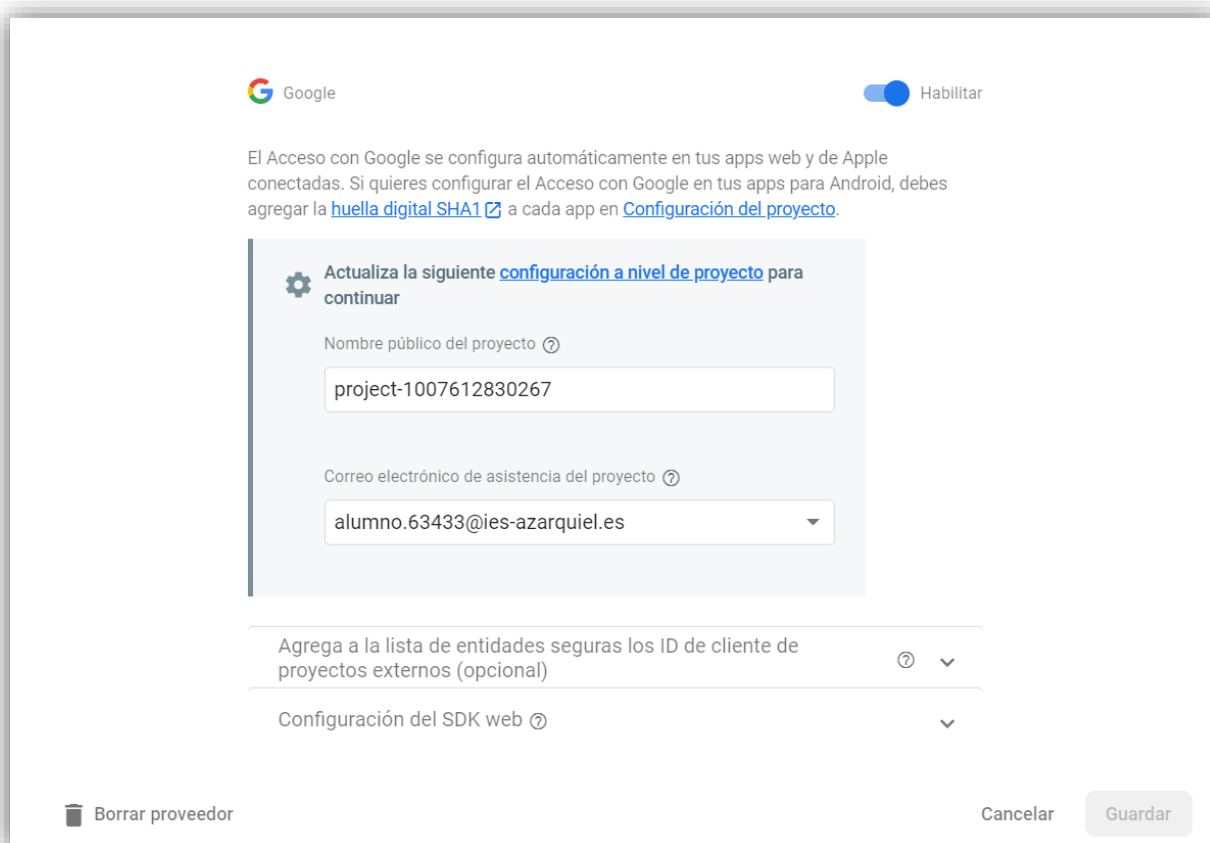
✉ Correo electrónico/contraseña Habilitar


Permite que los usuarios se registren con su dirección de correo electrónico y contraseña. Nuestros SDK también proporcionan verificación de la dirección de correo electrónico, recuperación de contraseñas y primitivas de cambio de dirección de correo. [Más información](#)

Vínculo del correo electrónico (acceso sin contraseña) Habilitar


 Borrar proveedor Cancelar Guardar


Firestore Auth 3 Editar opciones de proveedor Correo electrónico/contraseña






 Google Habilitar



El Acceso con Google se configura automáticamente en tus apps web y de Apple conectadas. Si quieres configurar el Acceso con Google en tus apps para Android, debes agregar la [huella digital SHA1](#) a cada app en [Configuración del proyecto](#).


 Actualiza la siguiente [configuración a nivel de proyecto](#) para continuar

Nombre público del proyecto 
project-1007612830267

Correo electrónico de asistencia del proyecto 
alumno.63433@ies-azarquiel.es

Agrega a la lista de entidades seguras los ID de cliente de proyectos externos (opcional)  

Configuración del SDK web  

 Borrar proveedor Cancelar Guardar

Firestore Auth 4 Editar opciones de proveedor Google

Después de configurar los parámetros requeridos para la autenticación, es importante y obligatorio registrar la huella digital SHA1 de la aplicación en nuestro proyecto, ya que, sin ella, los proveedores no podrían asociar las cuentas de los clientes al proyecto y con lo cual no funcionarían los métodos asociados a la autenticación en la programación de la *app*. Para

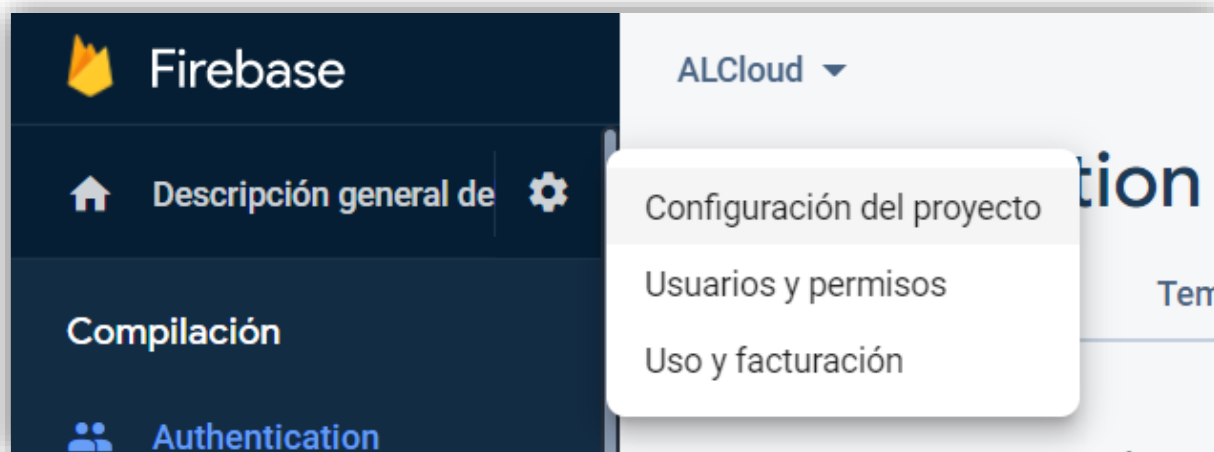
conseguir la huella digital SHA1 debemos introducir una serie de comandos en la terminal situada en el propio *IDE Android Studio*, en nuestro proyecto. Aquí el siguiente comando utilizado:

```
keytool -list -v -keystore "%USERPROFILE%\android\debug.keystore" -  
alias androiddebugkey -storepass android -keypass android
```

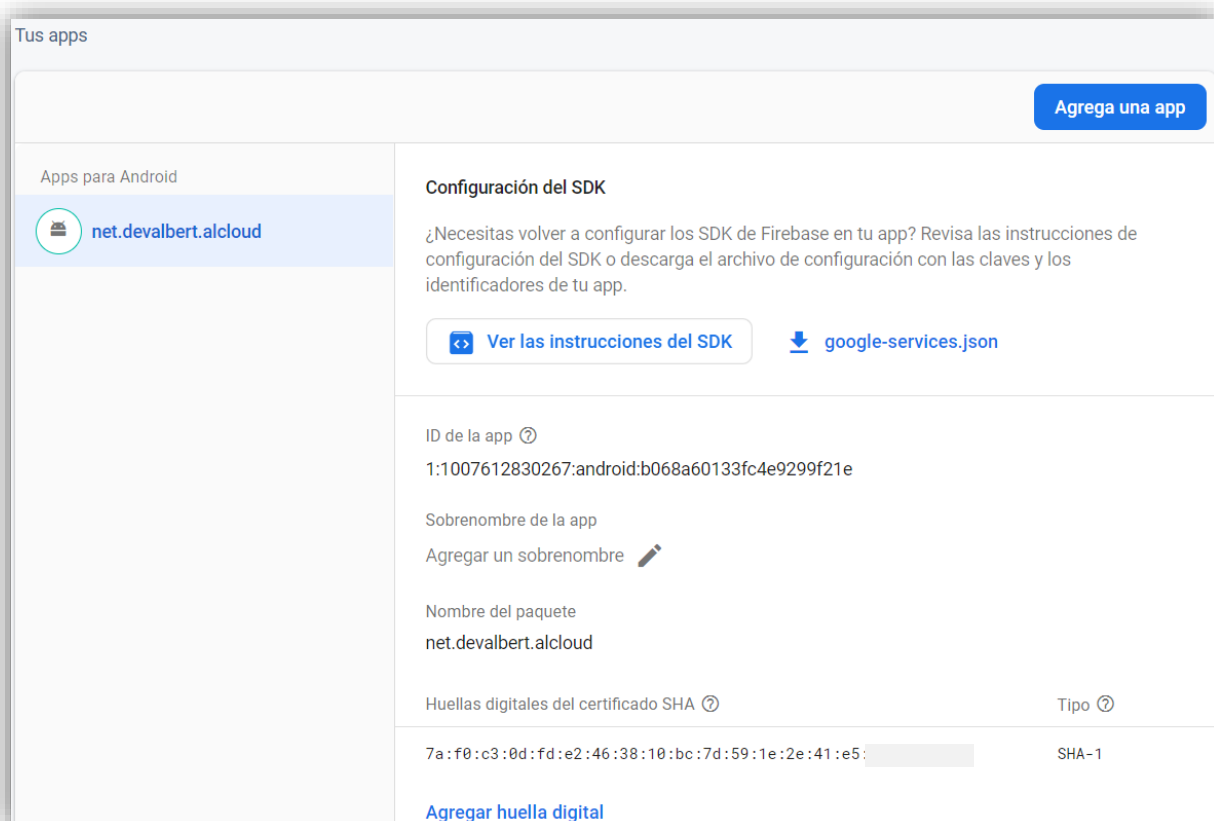
Cuando se introducen dichos comandos en la terminal, la salida es la siguiente (los valores pueden cambiar en función de cada proyecto):

```
E:\TFG (Provisional)\ALCloud>keytool -list -v -keystore  
"%USERPROFILE%\android\debug.keystore" -alias androiddebugkey -  
storepass android -keypass android  
  
Alias name: androiddebugkey  
Creation date: 18 mar. 2022  
Entry type: PrivateKeyEntry  
Certificate chain length: 1  
Certificate[1]:  
Owner: C=US, O=Android, CN=Android Debug  
Issuer: C=US, O=Android, CN=Android Debug  
Serial number: 1  
Valid from: Fri Mar 18 15:55:20 CET 2022 until: Sun Mar 10  
15:55:20 CET 2052  
Certificate fingerprints:  
    SHA1:  
7A:F0:C3:0D:FD:E2:46:38:10:BC:7D:59:1E:2E:41:E5:  
    SHA256:  
C6:C3:6D:64:72:18:D3:5A:21:EB:68:DC:68:FC:DB:B4:B3:E3:35:D2:68:68:  
2D:F3:98:D6:8F:  
Signature algorithm name: SHA1withRSA  
Subject Public Key Algorithm: 2048-bit RSA key  
Version: 1
```

A continuación, hay que introducir la huella obtenida en el proyecto. Para ello debemos ir a la consola de *Firebase*, **Configuración del proyecto** y buscar el apartado **Tus apps** para después pulsar en el botón **Agregar huella digital** e introducir la nuestra.



Firebase Auth 5 Configuración Proyecto



Firebase Auth 6 Agregar Huella Digital

Cuando hayamos hecho lo comentado anteriormente, ya sería posible realizar registros y *logins* en nuestra aplicación y que estos se reflejaran en la consola del proyecto.

Firestore Realtime Database

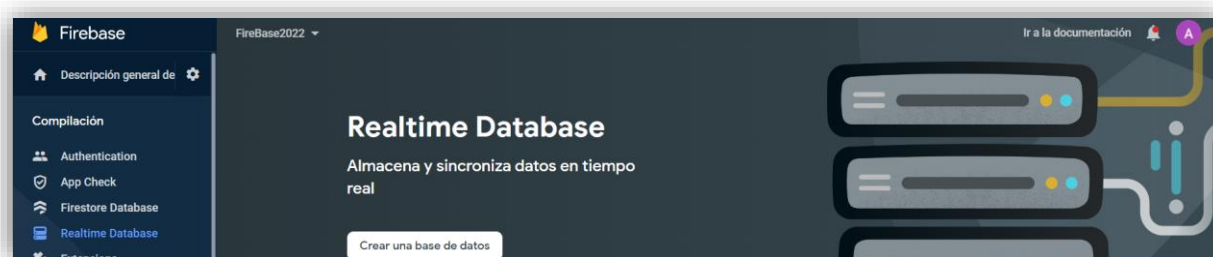
Este servicio es el encargado de almacenar y sincronizar todos los ficheros que tenemos alojados en nuestra base de datos NoSQL, que se encuentra situada en la nube. Como su propio nombre indica, esto se realiza en tiempo real, es decir, toda la ejecución de dicha prestación se realiza al momento según se necesita (por ejemplo, si un cliente sube a la nube que contiene este servicio un fichero, este se mostrará al instante según termine la operación de subida). Es por este motivo por el que decidí usar esta utilidad para ALCLOUD, así de esta manera se consigue que el usuario aprecie que la aplicación es rápida.

Para poder usar *Firestore Realtime Database* es necesario usar la librería principal como es *BoM* y a continuación, el paquete individual de dicho servicio. Como siempre, *Firestore* nos lo proporciona:

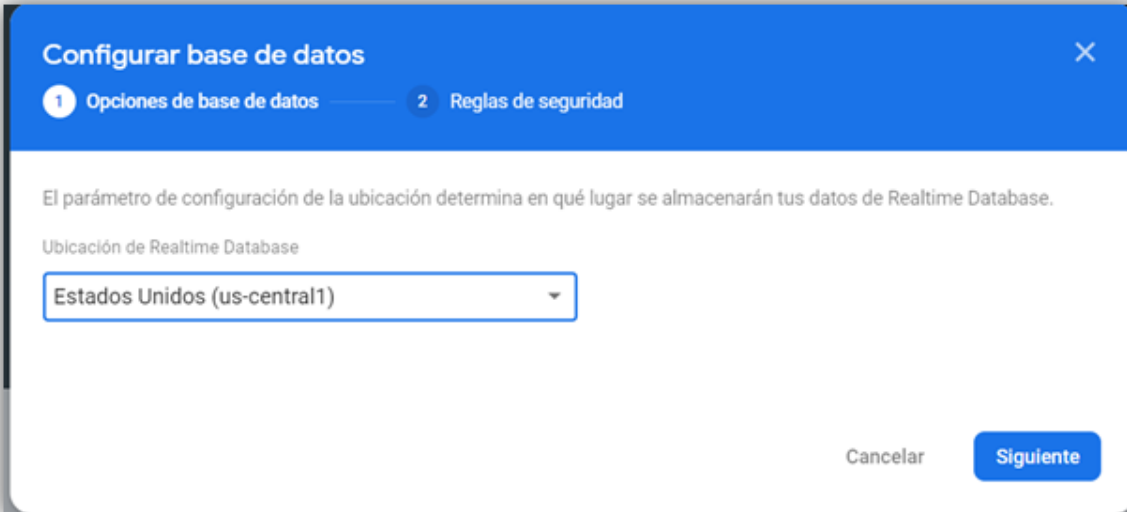
```
dependencies{  
    // Declare the dependency for the Realtime Database library  
    // When using the BoM, you don't specify versions in Firestore  
    library dependencies  
        implementation 'com.google.firebase:firebase-database-ktx'  
}
```

Después de implementar y sincronizar el *Gradle* del proyecto para obtener esta librería, ahora toca realizar toda la configuración de lo que es el servicio en sí, el cual nos proporciona unas mejoras para nuestro almacenamiento y, por consiguiente, a nuestra *app*. Dichas mejoras son la actualización de ficheros en tiempo real y una serie de normas para poder ejecutar este servicio.

Lo primero es realizar una BBDD que esté asociado a nuestro proyecto. Para ello debemos dirigirnos a la consola de *Firestore* de nuestro proyecto y seleccionar el servicio de *Realtime Database* para crear una BBDD.



Ahora se nos abrirá un diálogo en el cual debemos seleccionar dónde se alojará esa BBDD, por defecto nos elige la opción de **Estados Unidos (us-central1)**



Configurar base de datos

1 Opciones de base de datos 2 Reglas de seguridad

El parámetro de configuración de la ubicación determina en qué lugar se almacenarán tus datos de Realtime Database.

Ubicación de Realtime Database

Estados Unidos (us-central1)

Cancelar Siguiente

Firestore Realtime Database 2 Seleccionar Alojamiento de BBDD

Cuando seleccionemos el alojamiento de la BBDD nos pedirá la configuración de la misma, que esta trata de 2 opciones, aunque solo podemos seleccionar el modo de prueba, ya que tenemos una cuenta educativa y por lo cual está limitado. Hay que tener en cuenta que, al escoger esta opción, disponemos de 30 días para cambiar las reglas del servicio.



Configurar base de datos

1 Opciones de base de datos 2 Reglas de seguridad

Cuando definas la estructura de los datos, deberás crear reglas para protegerlos.
[Más información](#)

☐ Comenzar en **modo bloqueado**
De forma predeterminada, tus datos son privados. El acceso de lectura/escritura de los clientes solo se otorgará como se indica en tus reglas de seguridad.

☒ Comenzar en **modo de prueba**
Para permitir una configuración rápida, los datos se abren de forma predeterminada. Sin embargo, debes actualizar las reglas de seguridad en un plazo de 30 días a fin de habilitar el acceso de lectura/escritura a largo plazo para los clientes.

```
{
  "rules": {
    ".read": "now < 1656712800000", // 2022-7-2
    ".write": "now < 1656712800000", // 2022-7-2
  }
}
```

! Las reglas de seguridad predeterminadas del modo de prueba permiten que cualquier usuario con acceso a tu referencia de base de datos pueda ver, editar y borrar todos los datos durante los siguientes 30 días.

Cancelar Habilitar

Firestore Realtime Database 3 Seleccionar opción para la BBDD

Cuando se habilita la BBDD, es obligatorio cambiar las reglas de la misma para obtener una mayor seguridad en los archivos y, por ende, en la aplicación. Para ello tenemos que dirigirnos al apartado de *Realtime Database* y seleccionar en el menú superior **Reglas**. Aquí podremos establecer durante cuánto tiempo la base de datos estará disponible para realizar operaciones de lectura y escritura.



Firebase Realtime Database 4 Reglas del servicio

Este servicio ya estaría disponible y ejecutándose en cualquier momento en tiempo real.

Firestore Storage

Firestore Storage es uno de los servicios más potentes que tiene *Google*, ya que es capaz de almacenar objetos de manera simple y con capacidad para escalar dicha funcionalidad. Lo mejor que puede tener este servicio es como no, la seguridad de *Google*, puesto que cuando se realiza una operación de subida o bajada de archivos, están totalmente cifradas y no es necesario disponer de una red de alta velocidad.

Aquí se encuentra la piedra angular del proyecto, donde hay que configurar, aparte de las bibliotecas como en servicios anteriores, los servidores de *Google*, mediante *Google Cloud*, que es otro paquete de *Google* separado de *Firebase* pero que, en este caso, los he unido para hacer posible ALCLOUD.

Así, empezamos nuevamente añadiendo los SDK a nuestro proyecto con las líneas necesarias que *Firebase* nos proporciona:

```
dependencies{

// Declare the dependency for the Cloud Storage library
// When using the BoM, you don't specify versions in Firebase
library dependencies
    implementation 'com.google.firebase:firebase-storage-ktx'
}
```

Después de implementar el SDK es hora de crear el *bucket*. Para ello solo tenemos que seguir los pasos de configuración que nos facilita el portal *Firebase*.



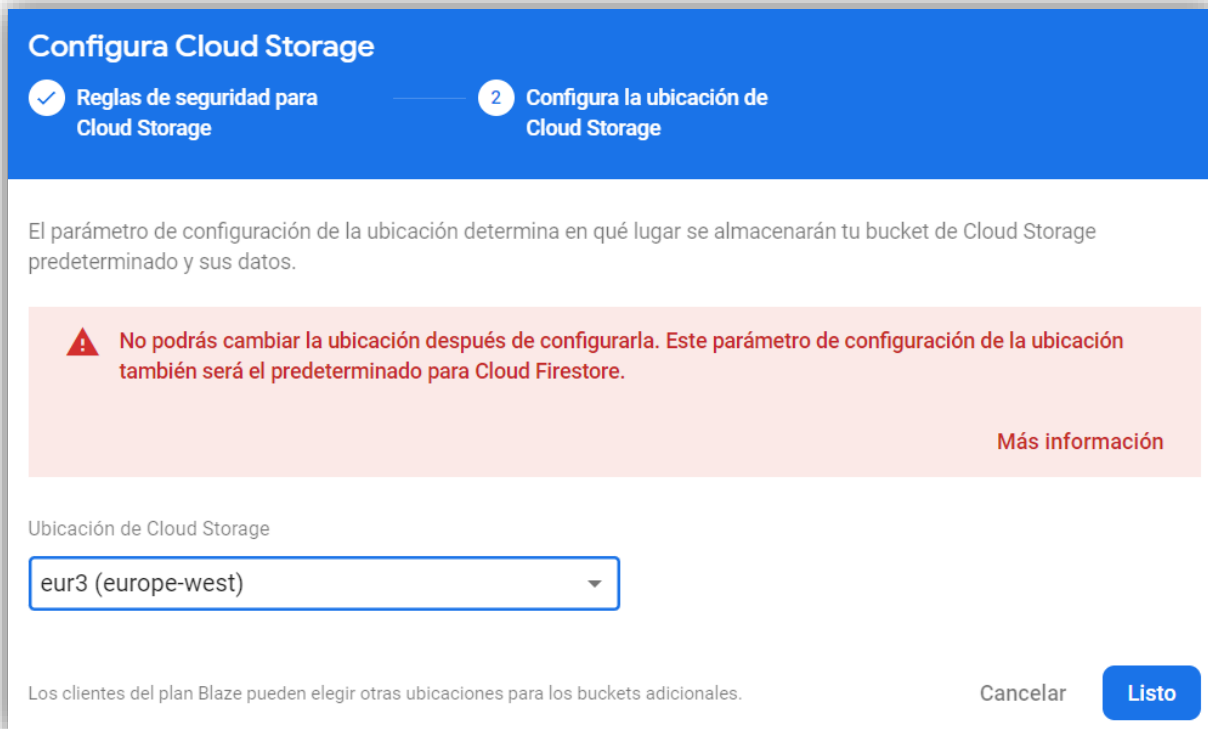
Firebase Storage 1 Iniciar servicio

Cuando seleccionemos **Comenzar** se nos abrirá un diálogo, en el cual debemos seleccionar en qué modo se va a inicializar el servicio. Una vez más, solo podemos escoger el modo de prueba.



Firebase Storage 2 Configurar modo del servicio

Y como es obligatorio, tenemos que seleccionar la ubicación de los servidores. En este caso es recomendable seleccionar Europa Oeste para que así nuestra aplicación tenga una menor latencia a la hora de realizar operaciones.



Configura Cloud Storage

✓ Reglas de seguridad para Cloud Storage 2 Configura la ubicación de Cloud Storage

El parámetro de configuración de la ubicación determina en qué lugar se almacenarán tu bucket de Cloud Storage predeterminado y sus datos.

⚠ No podrás cambiar la ubicación después de configurarla. Este parámetro de configuración de la ubicación también será el predeterminado para Cloud Firestore.

[Más información](#)

Ubicación de Cloud Storage

eur3 (europe-west)

Los clientes del plan Blaze pueden elegir otras ubicaciones para los buckets adicionales.

Cancelar **Listo**

Firestore Storage 3 Ubicación Servidor

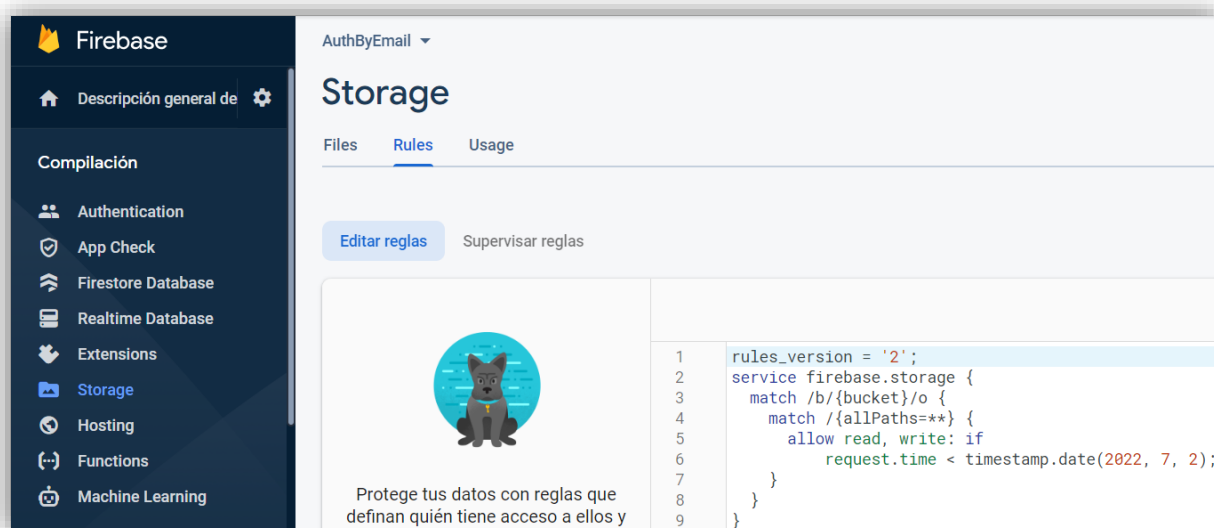
Cuando terminemos estos pasos se nos creará el *bucket*, que es lo que la aplicación necesita para poder tener la esencia de ser una nube. Lo primero después de todo esto, es supervisar y modificar las reglas si es necesario como hicimos con *Realtime Database* para asegurar nuestro servidor y configurar como actuará cuando se realicen peticiones.

Las reglas que se establecen por defecto significan que, para el servicio de *Firestore Storage* en el *bucket* predeterminado, para todas las rutas que se permitan las operaciones de lectura y escritura si la petición es anterior a la fecha indicada, en el caso de ALCLOUD solo estará disponible hasta el día 07/06/2023.

Hay que destacar que para obtener una mayor seguridad tanto en la aplicación como en los servidores deberíamos de establecer las reglas de esta manera:

```
rules_version = '2';
service firebase.storage {
  match /b/{bucket}/o {
    match /{allPaths=**} {
      allow read, write: if
        request.auth == true
    }
  }
}
```

Pero como he comentado antes, las reglas por defecto son estas:



Firebase Storage 4 Reglas del servicio

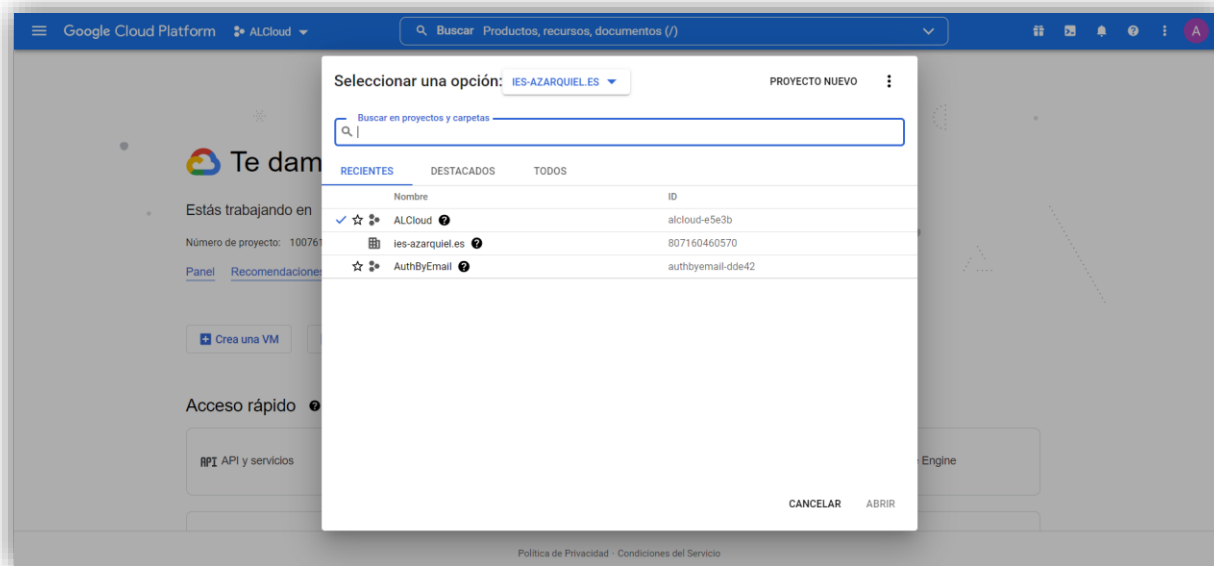
No es recomendable establecer de esta manera las reglas porque así permitimos que cualquier usuario autenticado o no pueda tener acceso al servidor, con lo cual dejaríamos la puerta abierta a intrusos que no queremos.

Google Cloud

Ahora viene la parte donde hay que rebuscar en la documentación de *Google* para hacer que el *bucket* sea accesible para poder escribir en el servidor, porque, aunque en las reglas del servicio establezcamos unas determinadas, no sirven de nada si no van acompañadas de las

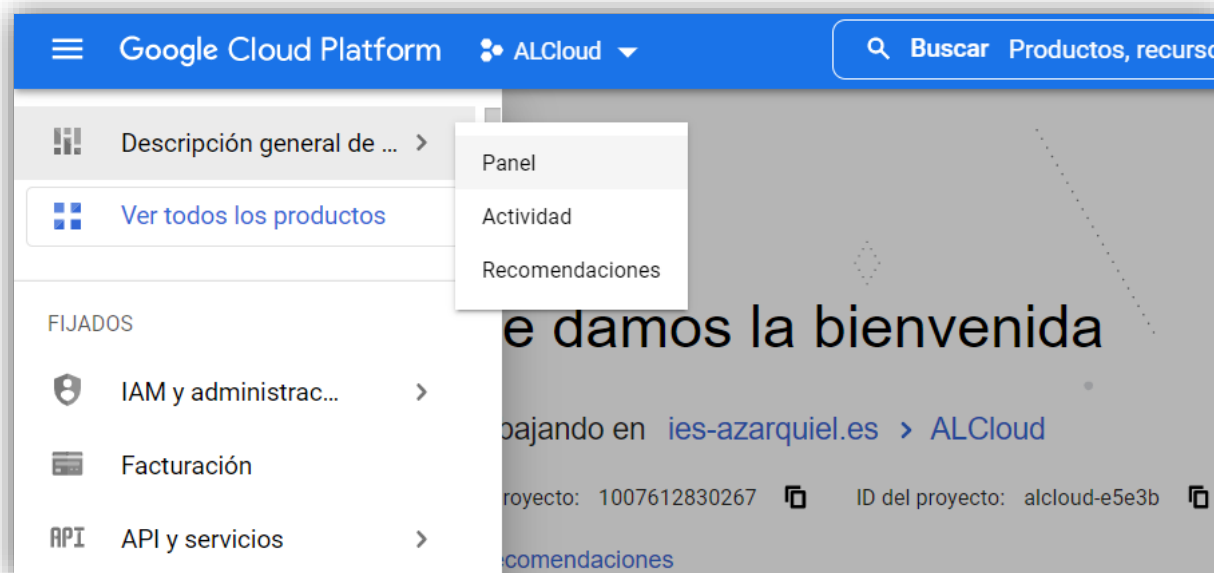
reglas que debemos instanciar en *Google Cloud* para nuestro proyecto. De nuevo, nos dirigimos a la consola de *Google Cloud* y a la configuración del proyecto que queramos.

Nada más acceder al portal web nos pedirá qué proyecto queremos manejar:



Google Cloud 1 Selección de proyecto

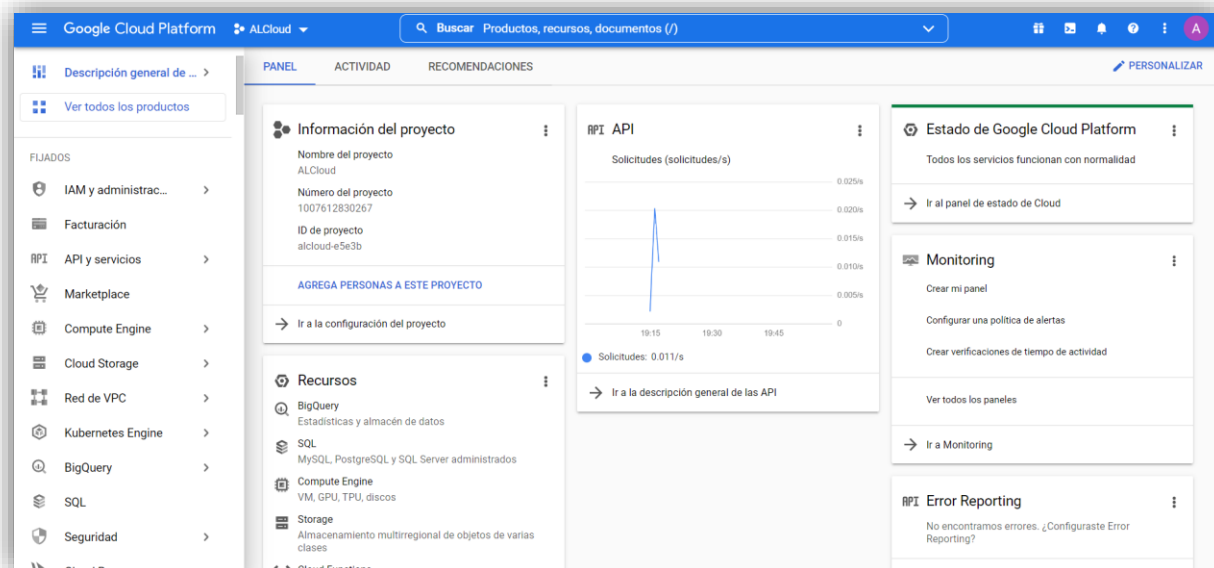
En el menú de navegación situado a la izquierda, después de abrir el proyecto que queramos seleccionamos en **Descripción general de Cloud > Panel**



Google Cloud 2 Panel del proyecto

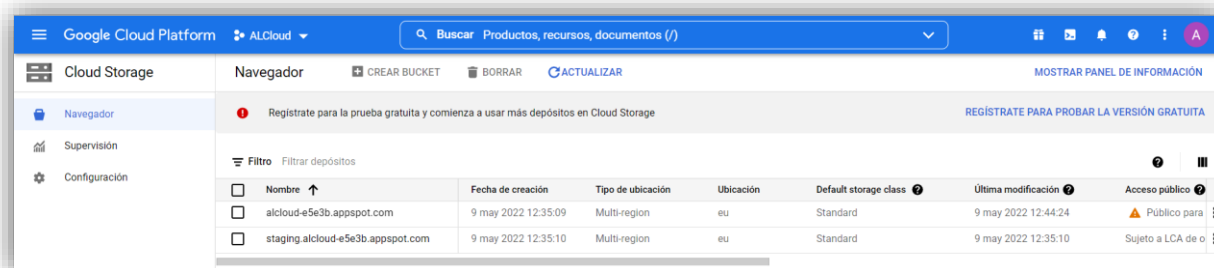
Como indica su nombre, el panel nos muestra las estadísticas de nuestro proyecto, indicando el estado de *Google Cloud*, la información del proyecto, un seguimiento de las acciones en el

bucket, seguimiento de peticiones a la API, etc. En este caso, para ALCLOUD nos interesa que cada acción que realizamos en la *app* se registre en buen estado en el panel del proyecto para saber que estamos haciendo peticiones correctas a la API de nuestro *bucket* y no a otro.



Google Cloud 3 Panel del proyecto

Cuando accedamos al panel de nuestro proyecto debemos ir a la opción del menú lateral izquierdo y seleccionar **Cloud Storage > Navegación** para poder establecer los permisos del bucket y aceptar o denegar las peticiones correspondientes.



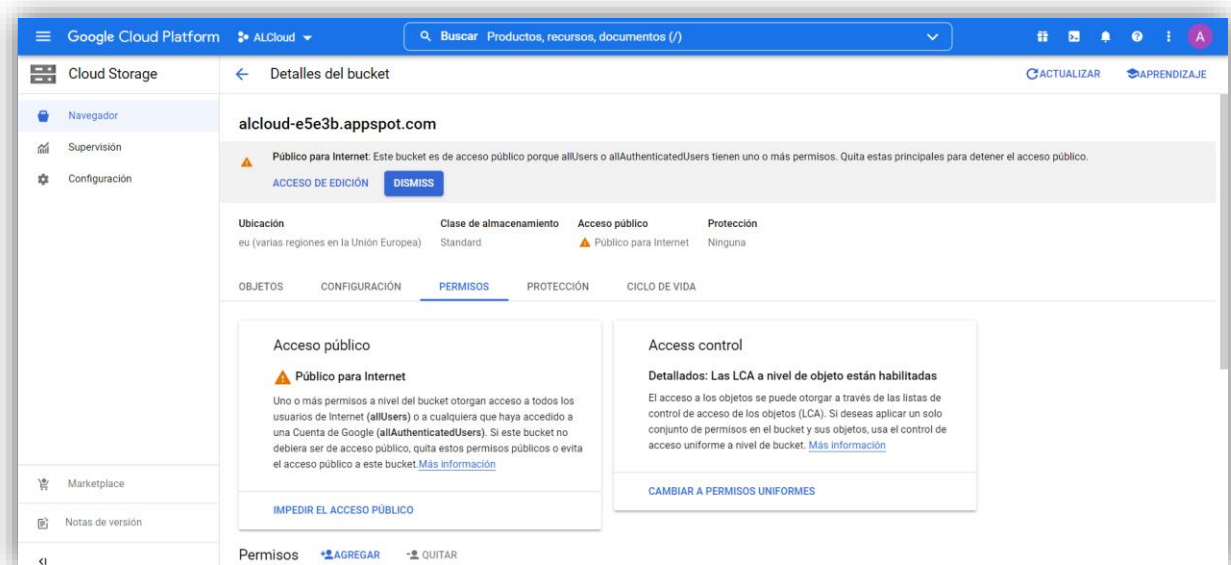
The screenshot shows the 'Navegador' (Browser) view of the Cloud Storage service. It displays a table of buckets. The table has columns for 'Nombre' (Name), 'Fecha de creación' (Creation Date), 'Tipo de ubicación' (Location Type), 'Ubicación' (Location), 'Default storage class', 'Última modificación' (Last Modified), and 'Acceso público' (Public Access). Two buckets are listed: 'alcloud-e5e3b.appspot.com' and 'staging.alcloud-e5e3b.appspot.com', both created on May 9, 2022, and located in the 'eu' region.

Nombre	Fecha de creación	Tipo de ubicación	Ubicación	Default storage class	Última modificación	Acceso público
alcloud-e5e3b.appspot.com	9 may 2022 12:35:09	Multi-region	eu	Standard	9 may 2022 12:44:24	Público para
staging.alcloud-e5e3b.appspot.com	9 may 2022 12:35:10	Multi-region	eu	Standard	9 may 2022 12:35:10	Sujeto a LCA de o

Google Cloud 4 Configurar Bucket

En este apartado debemos seleccionar el *bucket* correspondiente al que queremos editar las reglas, normalmente suele ser el primero. Entonces lo seleccionamos y al entrar nos aparecerá los distintos archivos subidos por los usuarios de nuestra aplicación. En el caso de ALCLOUD tenemos 3 carpetas que corresponden a los usuarios que he ido utilizando para comprobar el funcionamiento de los servicios y de la app en sí.

Cuando entremos en ese *bucket* seleccionamos la pestaña **Permisos** que se encuentra un poco más arriba de los *bucket* y es ahí donde debemos establecer los permisos.



Google Cloud 5 Establecer permisos Bucket

Seleccionamos **Agregar** y tenemos que poner los permisos deseados, pero en este caso para mi PFC he decidido usar el permiso de que cualquiera pueda leer los objetos que se encuentran dentro del *bucket* para facilitar el acceso y mostrar como funcionaría si estuviera en un entorno real.

Una vez que hayamos establecido todos los permisos que queramos, nuestra base de datos ya estaría disponible para su uso y todos los servicios corriendo de manera sincronizada.

Otras tecnologías

También debo documentar otras tecnologías que no solo sirven para la funcionalidad completa de la aplicación y sus objetivos, sino que hay otros paquetes que he utilizado, por ejemplo, para realizar el diseño de la muestra de los ficheros dentro de la *app*.

Picasso

Se trata de una librería que permite insertar imágenes en nuestras aplicaciones *Android* de una manera super sencilla, ya que su sintaxis no tiene complicación.

```
Picasso.with(context).load("http://i.imgur.com/DvpvklR.png").into(
    imageView)
```

Kotlin Coroutines

Este paquete es fundamental en el proyecto, puesto que como necesitamos obtener datos a través de la red, es necesario que se realice de una manera simultánea y paralela al hilo principal de la aplicación. Así el propio sistema *Android* no nos finalizará la ejecución por estar inactiva durante determinados segundos.

Retrofit-Gson

Otra de las librerías más útiles es *Retrofit-Gson*, porque nos permite serializar objetos a formato JSON para poder trabajar de una manera más eficaz con ellos. Por ejemplo, yo lo he usado para el paquete API del proyecto, el cual se encarga de realizar las llamadas a la API mediante interfaces que devuelven los datos, creando un objeto con unos parámetros indicados en la clase.

ArchLifecycle

Se utiliza para los ciclos de vida de algunos componentes de nuestra aplicación, pudiéndose usar en este proyecto para poder hacer otra acción en otro método a la respuesta de la API. Mayormente se usa en el paradigma MVVM (*Model-View, View-Model*)

Desarrollo e implementación

Como he utilizado varias tecnologías y explicado como sería su uso desde el *backend* para arrancar esos servicios y sincronizarlos entre sí, ahora es necesario mostrar también cómo se usan dichos servicios desde la vista del código, que es lo que realmente interesa para saber a ciencia cierta cómo funciona la aplicación. Cabe destacar que la mayoría del código usado viene dado por la documentación de *Google* de *Firebase*, pero muchas de las partes fundamentales y funcionales de ALCLOUD han tenido que ser desarrolladas de tal manera que sirvieran para la *app*, ya que en la propia documentación no se encuentra cómo realizar dichos apartados, como, por ejemplo, la subida de archivos al *bucket* y demás.

Así, empezaré por señalar cómo se instancian y se utilizan las librerías que he comentado antes, comenzando por el paquete de autenticación, puesto que es lo primero que se puede ver al iniciar ALCLOUD.

Lo más normal es que cuando entremos a la aplicación nos tengamos que registrar, ya que somos usuarios nuevos y tendremos que darnos de alta en la base de datos para luego poder usar los servicios de la *app*.

Código de autenticación *Firebase Authentication*

Primeramente, según la documentación de *Firebase*, tenemos que realizar una instancia del servicio de autenticación. Para ello he usado las siguientes líneas de código:

```
FirebaseAuth.getInstance().createUserWithEmailAndPassword(  
    "parametros necesarios")  
    .addOnCompleteListener(  
        "parametros si la operación fue satisfactoria"  
    )  
}
```

La función de estas es conectarse con la base de datos del servicio de *Firebase Authentication* y mediante la función ***createUserWithEmailAndPassword*** crea un usuario con correo y contraseña (el que haya introducido el cliente en la aplicación). Sin embargo, para la función de *login*, es decir, para entrar a la aplicación, si ya tenemos unas credenciales tan solo hay que llamar a la función de ***signInWithEmailAndPassword*** con la misma instancia que antes.

Así quedaría dicha función en ALCLOUD:

```
FirebaseAuth.getInstance().signInWithEmailAndPassword(  
    "parametros necesarios")  
    .addOnCompleteListener(  
        "parametros si la operación fue satisfactoria"  
    )  
}
```

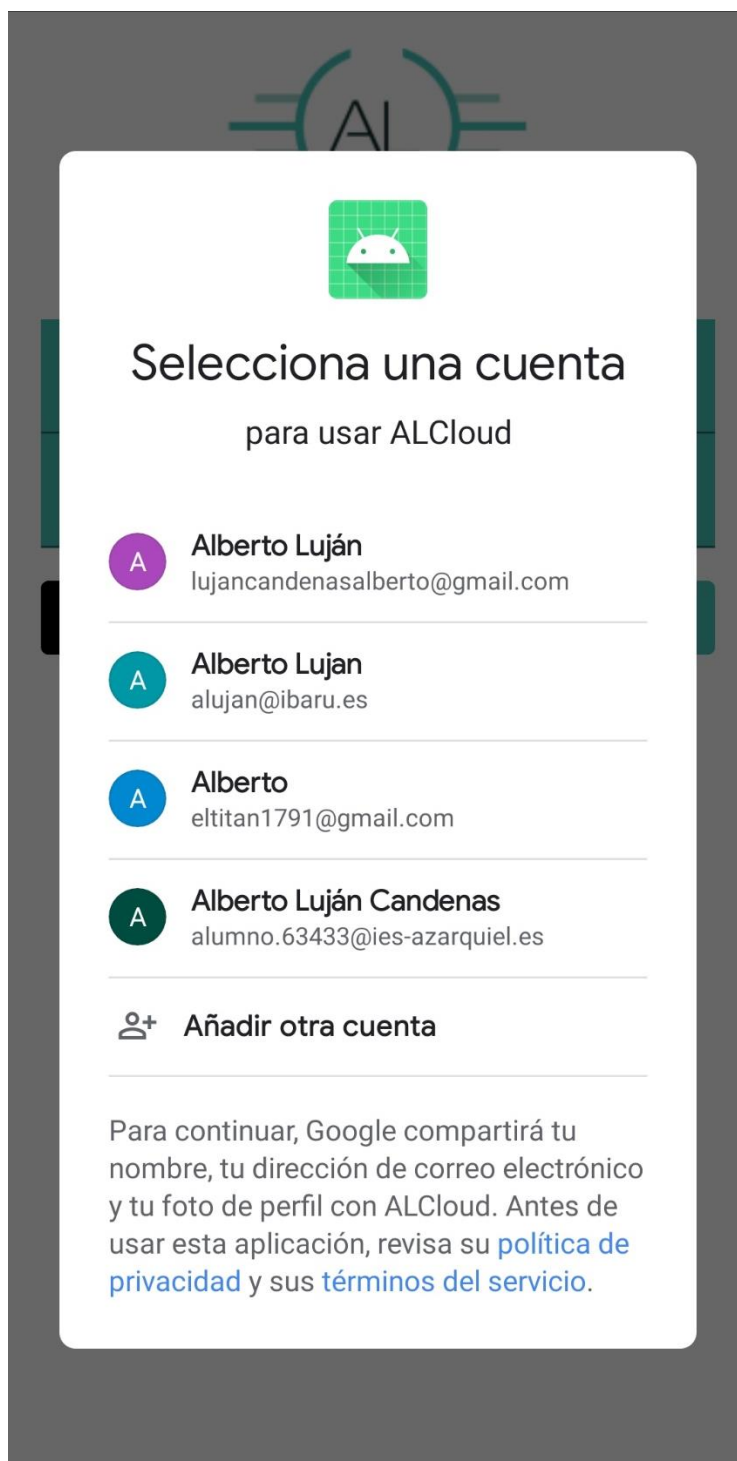
Código de autenticación por Google

Ahora mostraré algo realmente interesante y es cómo registrase en la aplicación con la *app* integrada de *Google*. Esta parte de código ha sido fruto de la documentación y desarrollo por mí mismo, ya que muchas de las funciones que se podía encontrar no me cuadraban de ninguna manera en la aplicación y tuve que usar como recurso *Java* y que el propio *IDE Android Studio* me lo compilara a *Kotlin*; es por eso que algunos apartados aparecen como ***deprecated***, porque no encontraba la manera de hacerlo "moderno".

Así quedaría dicho código:

```
googleBtn.setOnClickListener{  
    //Configuracion  
    // Configure Google Sign In  
    val gso =  
        GoogleSignInOptions.Builder(GoogleSignInOptions.DEFAULT_SIGN_IN)  
            //No es un error, el string esta dentro de  
            res/generated) que se hace automaticente por Google  
            .requestIdToken(getString(R.string.default_web_client_id))  
            .requestEmail()  
            .build()  
  
    val googleClient:GoogleSignInClient =  
        GoogleSignIn.getClient(this, gso)  
        googleClient.signOut()  
  
        startActivityForResult(googleClient.signInIntent,  
        GOOGLE_SIGN_IN)  
}
```

Lo que se consigue en esta parte de la aplicación es llamar al constructor de la clase **GoogleSignInOptions** para poder obtener el *idToken* de la aplicación que se encuentra dentro de los archivos propios, que se generan a la hora de la ejecución e instanciación de las librerías correspondientes. Así, de esta manera, procedemos a obtener el *email* del usuario que utilice y llamar a la función **startActivityForResult (actualmente deprecated)** para que el dispositivo muestre una pantalla de selección de cuentas de *Google*:



Inicio de sesión 1 Opción Google

Cuando seleccionamos una cuenta de *Google* que tengamos en nuestro dispositivo, la aplicación se desarrolló para que guardara el *email* para futuros inicios de sesión. Pero la funcionalidad que tiene el realizar la autenticación por *Google* es el siguiente:

```
override fun onActivityResult(requestCode: Int, resultCode: Int,
data: Intent?) {

    super.onActivityResult(requestCode, resultCode, data)

    if(requestCode == GOOGLE_SIGN_IN){

        val task: Task<GoogleSignInAccount> =
        GoogleSignIn.getSignedInAccountFromIntent(data)

        try{

            val account: GoogleSignInAccount =
            task.getResult(ApiException::class.java)

            if (account != null){

                val credential: AuthCredential =
                GoogleAuthProvider.getCredential(account.idToken, null)

                FirebaseAuth.getInstance().signInWithCredential(credential)

                .addOnCompleteListener{

                    if(it.isSuccessful){

                        showMain(account.email?: "",
                        account.displayName?: "", ProviderType.GOOGLE)

                        //en caso de no existir el email
                        devolvera un string vacio

                    }else{

                        showAlert()

                    }

                }

            }

        }

        catch (e: ApiException){

            showAlert()

        }

    }

}
```

Esta parte de código es la completa funcionalidad del *login* por *Google*, ya que realiza las llamadas correspondientes a *Google* mediante la librería de *Firebase Authentication* y recoge de la cuenta seleccionada el *token* mediante una tarea que se realiza en el sistema (pantalla de selección) para luego pasarlo a la instancia de *Firebase* y saber que se ha registrado con una cuenta de *Google*.

Estas son las partes de la autenticación más interesantes que tiene ALCLOUD, ya que es algo que nunca hemos podido contemplar en nuestro horario lectivo y ha sido un gran reto para el proyecto como otros servicios (*Cloud Storage*).

Código para la subida de archivos a *Firebase Storage*

Lo más normal al usar *Firebase* en nuestras aplicaciones Android es que para poder utilizar los distintos servicios, es obligatorio las librerías y la creación de instancias para usar ese servicio, ya que, si no realizamos una de estas antes, la librería no nos permite usar las distintas funciones que contienen para poder realizar en este caso la subida de archivos. Considero esta parte del proyecto muy interesante porque aquí es donde se ve cómo se programa esa subida de los ficheros: pero claro, antes de subirlos debemos seleccionarlos y para ello hay que notificar al sistema de que queremos abrir el gestor de archivos que tengamos por defecto instalado para poder realizar el recuento de los archivos y a partir de ahí usar *Firebase* para la subida.

```
private fun fileManager() {  
    val intent = Intent(Intent.ACTION_GET_CONTENT)  
    if (android.os.Build.VERSION.SDK_INT >=  
android.os.Build.VERSION_CODES.JELLY_BEAN_MR2) {  
        intent.putExtra(Intent.EXTRA_ALLOW_MULTIPLE, true)  
    }else{  
        Toast.makeText(this, "Se necesita una version de  
override fun onActivityResult(requestCode: Int, resultCode: Int,  
data: Intent?) {  
    super.onActivityResult(requestCode, resultCode, data)  
    if (requestCode == fileResult) {  
        if (resultCode == RESULT_OK && data != null) {  
            val clipData = data.clipData  
  
            if (clipData != null) {  
                for (i in 0 until clipData.itemCount) {  
                    val uri = clipData.getItemAt(i).uri  
                    uri?.let { fileUpload(it) }  
                }  
            } else {  
                val uri = data.data  
                uri?.let { fileUpload(it) }  
            }  
        }  
    }  
}
```

Como vemos, lo que se hace en estas líneas de código es crear un *Intent*, el cual está asociado a lo comentado anteriormente (mostrar la pantalla de selección de ficheros del sistema); pero todo este se puede realizar si la versión de *Android* es mayor o igual a *Lollipop*, es decir, la versión 5.0. Después de la comprobación procedemos a iniciar la *activity* para los ficheros, en la cual realizamos la selección múltiple de esta manera.

Con este apartado lo que se quiere conseguir es que el usuario pueda subir varios tipos de archivos, ya sean como fotos, vídeos, documentos, etc. y poder seleccionarlos de una vez. De esta forma, conseguimos uno de los objetivos principales de la aplicación: la velocidad de interacción.

Así, el código final de este apartado donde se realiza ya la subida en cuestión de los ficheros es el siguiente:

```
private fun fileUpload(mUri: Uri) {  
    //Crear carpeta para cada uno de los usuarios  
    val folder: StorageReference =  
        FirebaseStorage.getInstance().reference.child(username)  
    val path = mUri.lastPathSegment.toString()  
    val fileName: StorageReference =  
        folder.child(path.substring(path.lastIndexOf('/') + 1))  
    progreso.visibility = View.VISIBLE  
    llFotos.visibility = View.INVISIBLE  
    subtitulo.text = "Subiendo archivos"  
    fileName.putFile(mUri).addOnSuccessListener {  
        progreso.visibility = View.INVISIBLE  
        subtitulo.text = "Tus archivos"  
        initRV()  
        llFotos.visibility = View.VISIBLE  
    }.addOnFailureListener {  
        progreso.visibility = View.INVISIBLE  
        subtitulo.text = "Tus archivos"  
        Toast.makeText(this, "Error en la subida de archivos",  
            Toast.LENGTH_SHORT).show()  
    }  
}
```

Como se puede apreciar, lo que requiere *Firebase* para poder subir el fichero es la URI de este mismo. Con lo cual, creamos la referencia a la base de datos añadiéndole como último componente de la ruta el nombre del usuario de quien lo subió para después realizar una comprobación donde se mostrará solo archivos subidos por el mismo, es decir, el usuario

registrado. Entonces es cuando llamamos a la función de *Firebase Storage* ***putFile(uri)***. Este método nos obliga a realizarlo como una tarea para el sistema añadiéndole como un sensor que nos indica si ha ido todo correcto o no (***addOnSuccessListener***).

Conclusiones y líneas futuras

Conclusiones

En conclusión, podría decir que ALCLOUD me ha aportado bastante como desarrollador, ya que ha sido un gran reto manejar *Firebase*. Esta es una aplicación para desarrolladores prácticamente nueva y he de decir que me ha sorprendido la facilidad con la que puedes manejar el *backend* y cómo se integra de manera total con *Android* y las aplicaciones que podemos realizar. Aprendes muchísimo sobre cómo manejar bien el sistema y, sobre todo, al menos en este PFC, cómo manipular los ficheros para hacerlos capaces de navegar por Internet, cómo integrarlos en tu aplicación, etc.

Gracias a la formación recibida en *Android* en este grado, he obtenido una base consistente que me ha ayudado bastante con este proyecto; aunque, por otra parte, como no hemos trabajado con estos servicios en concreto (es decir, *Firebase Storage*, *Firebase Authentication* y *Firebase Realtime Database*), ha sido una manera de llevar a la práctica algo aprendido en clase de una forma totalmente diferente.

Además, una ventaja de este PFC es que no acaba aquí, puesto que puede servir como base para el desarrollo de otras aplicaciones, aprovechando así no solo los servicios utilizados (*Firebase*), sino también la experiencia que he adquirido en su manejo.

Líneas futuras

Esta aplicación podría y debería expandirse aún más, ya que puede contener muchísimos más servicios, como por ejemplo los de mensajería en campaña; es decir, el uso de notificación por parte de la compañía para recordar a los usuarios que tengan en sus dispositivos ALCLOUD, las nuevas actualizaciones que tendrá la misma y sus nuevas funciones, recordar que pueden salvaguardar sus ficheros, eventos de membresía y demás...

También podría incluirse la posibilidad de una aplicación web que realizara lo mismo que la aplicación *Android* para que así, de esta manera, todos aquellos usuarios que no dispongan de un *smartphone* compatible puedan acceder desde cualquier ordenador a través de su navegador favorito, convirtiéndose así ALCLOUD en una aplicación más expandible y versátil.

Otra de las funciones que me gustaría haber añadido es la posibilidad de integrar la aplicación con el sistema, es decir, que, a través de otras aplicaciones nativas en el móvil, como por ejemplo en la galería, existiese la posibilidad de elegir las fotos y/o vídeos y subirlas a ALCLOUD directamente desde ahí. Esto, además, conllevaría la realización de servicios en segundo plano que permitieran usar algunas funciones de la aplicación sin tener que estar ejecutándola y así poder realizar otras tareas en nuestro teléfono móvil.

De esta manera, poco a poco, ALCLOUD podría convertirse en una aplicación senior llena de multitud de opciones y demás funcionalidades que la llevarían a alcanzar un mayor público y a tener más utilidades.

Referencias bibliográficas

06/05/2022 → Usar *Cloud Storage*:

<https://firebase.google.com/docs/storage/android/start?authuser=0&hl=es>

06/05/2022 → Usar *Realtime Database*:

<https://firebase.google.com/docs/database?authuser=0&hl=es>

09/05/2022 → Documentación de la API de Google:

https://cloud.google.com/storage/docs/json_api/v1/objects/list?apix_params=%7B%22bucket%22%3A%22fir-storage-906bf.appspot.com%22%7D#response

09/05/2022 → Descargar archivos desde *Firebase Storage*:

<https://firebase.google.com/docs/storage/android/download-files?hl=es-419>

10/05/2022 → Acceder a los datos públicos del *bucket*:

<https://cloud.google.com/storage/docs/access-public-data?hl=es-419>

14/05/2022 → Documentación para hacer el *bucket* público:

<https://cloud.google.com/storage/docs/access-control/making-data-public?hl=es-419#prereq-console>

20/05/2022 → Usar *authentication*:

<https://firebase.google.com/docs/auth/android/start?authuser=0>

https://www.youtube.com/watch?v=xjsgRe7FTCU&t=288s&ab_channel=MoureDevbyBraisMoure

https://www.youtube.com/watch?v=dpURgJ4HkMk&t=10s&ab_channel=MoureDevbyBrais
[Moure](#)

03/06/2022 → *BoM Firebase* (biblioteca principal):

<https://firebase.google.com/docs/android/learn-more?authuser=0&hl=es#bom>

04/06/2022 → Librería *Picasso*:

<https://desarrollador-android.com/librerias/square/picasso/>

04/06/2022 → *Kotlin Coroutines*:

<https://developer.android.com/jetpack/androidx/releases/lifecycle?hl=es-419>