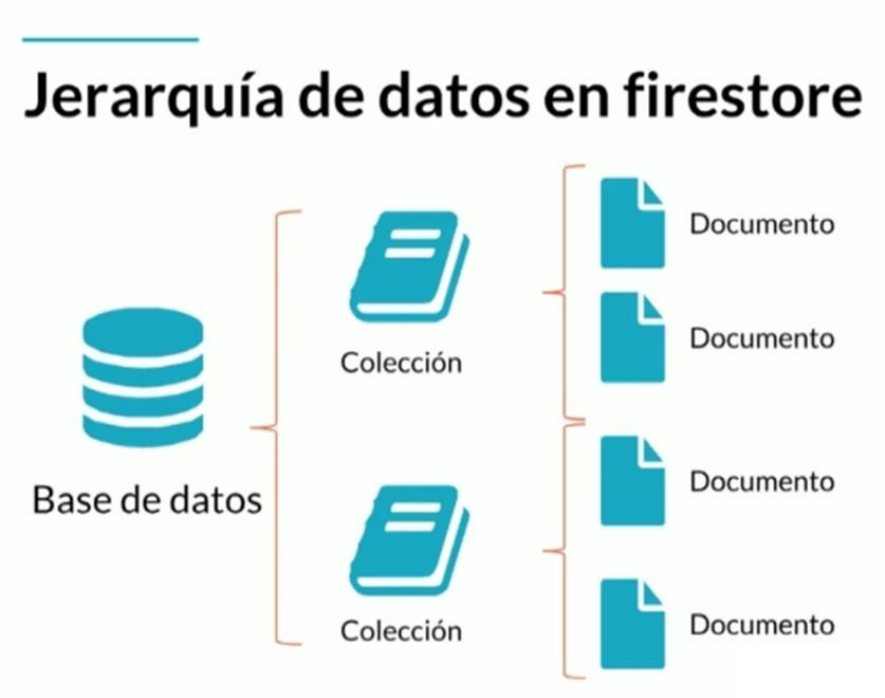
FireBase

FireBase es un recurso ampliar las posibilidades de las aplicaciones. Surgió en 2014 y es propiedad de Google. Ofrece diferentes servicios, desde autenticación a BBDD, servicio almacenamiento, hosting, configuración remota, etc. Es gratuita, aunque como siempre, ofrece una versión de pago.

Nosotros vamos a servirnos de FireBase como una Base de Datos On-Line (Cloud FireBase). A diferencia de otras bases de datos a las que estamos acostumbrados, **no es SQL** sino orientada a **documentos**.

La jerarquía de una BBDD en FireBase es la siguiente:

- Una **Base de Datos** es una agrupación de **colecciones**,
- Una **Colección** es una entidad, una "carpeta" y se las identifica con nombres en plural.
- Cada registro que queremos guardar es un Documento.

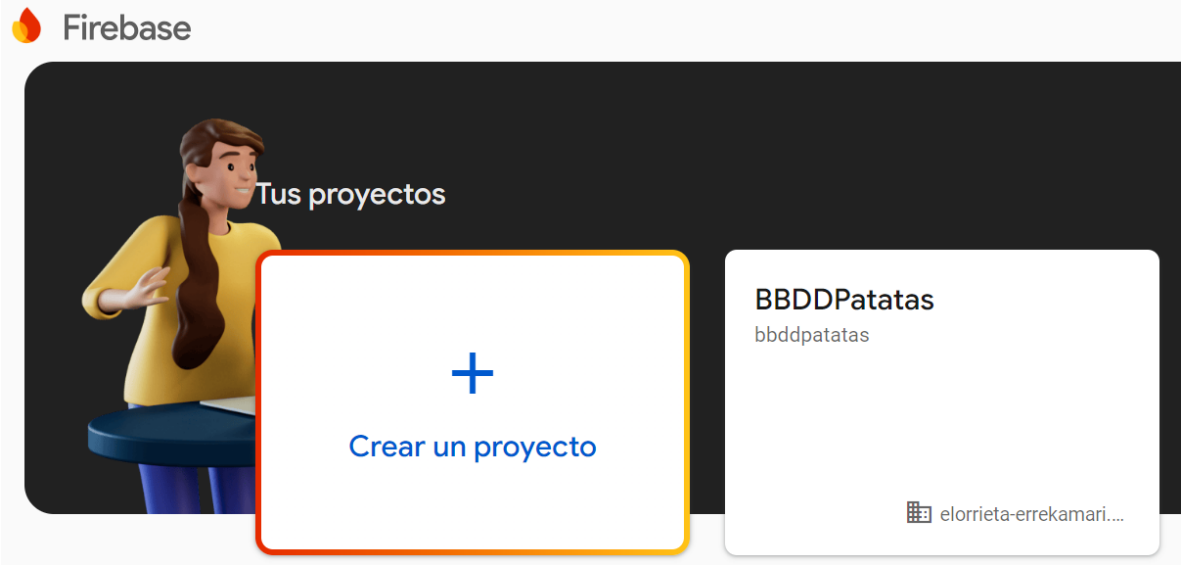


Vamos a ver los pasos que hay que seguir para usar FireBase desde un dispositivo Android.

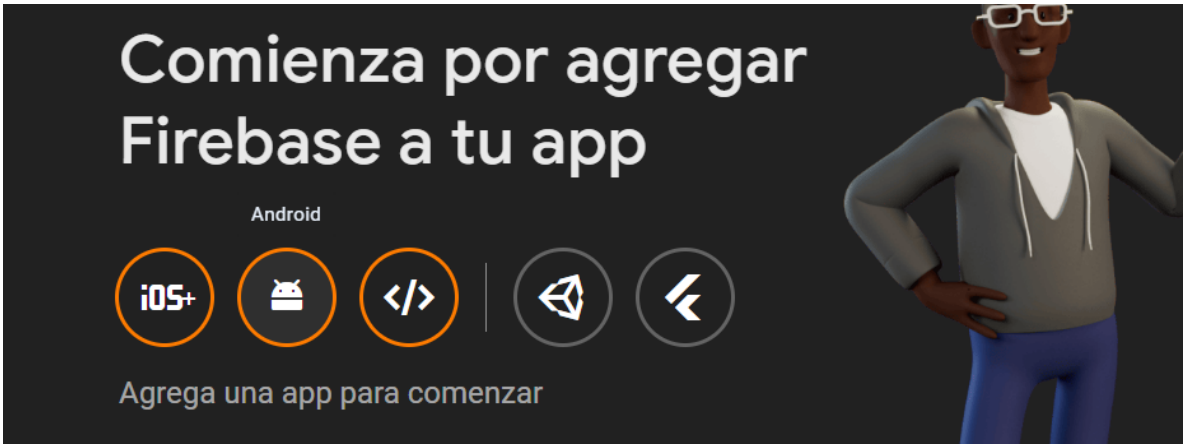
Crear y añadir FireBase al Android Studio

Lo primero es crear un proyecto en FireBase. Acudimos a la **FireBase Console** y agregamos un nuevo proyecto. En las opciones que nos salen, simplemente aceptamos las condiciones de uso, el uso de Google Analytics, y que para ello se utilice la cuenta por defecto.

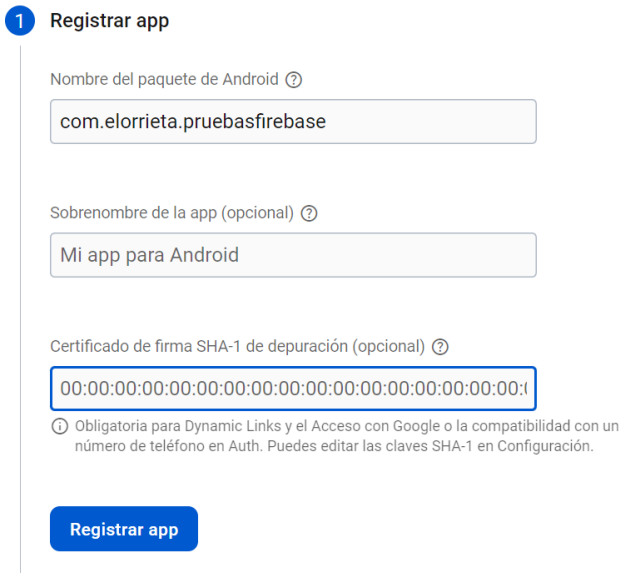
Para este ejemplo, el proyecto se va a llamar **BBDDPatatas**, puesto que deseamos inventariar nuestro almacén de productos alimentarios.



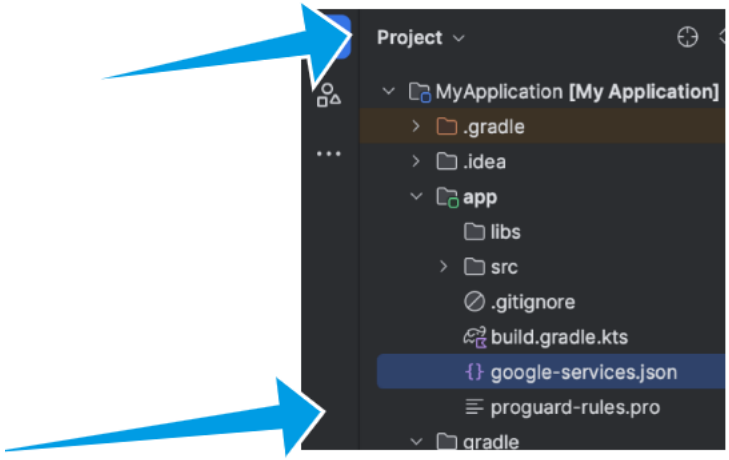
A continuación, necesitamos registrar nuestra App en con el proyecto FireBase. Para eso, tenemos que 'agregar' la app al proyecto. Dentro del proyecto BBDDPatatas pulsamos al icono de Android (o en **Agregar app**).



En la siguiente pantalla, necesitamos configurar los parámetros de nuestra App. Nótese que el **nombre del paquete** es el mismo que el de nuestra App en el Android Studio.



Al registrar la App, nos permite descargar un fichero llamado **google-services.json** que debemos instalar en nuestro proyecto Android. Contiene la configuración de FireBase que necesitamos. Tiene que estar incluido en:



Ahora, tenemos que añadir el SDK de FireBase. Para añadirlo a un proyecto **Kotlin**, abrimos el fichero **build.gradle.kts**. Este fichero es utilizado para compilar la App en el Android Studio, luego ten cuidado al modificarlo.

```
plugins {
    alias(libs.plugins.android.application) apply false
    alias(libs.plugins.jetbrains.kotlin.android) apply false

    id("com.google.gms.google-services") version "4.4.2" apply false
}
```

Tras sincronizar de nuevo el proyecto, acudimos al build.gradle.kts de nivel de aplicación (app) y agregamos los complementos google-services y cualquier SDK de FireBase que queramos usar.

```
plugins {
    alias(libs.plugins.android.application)
    alias(libs.plugins.jetbrains.kotlin.android)

    id("com.google.gms.google-services")
}
```

Y en dependencias:

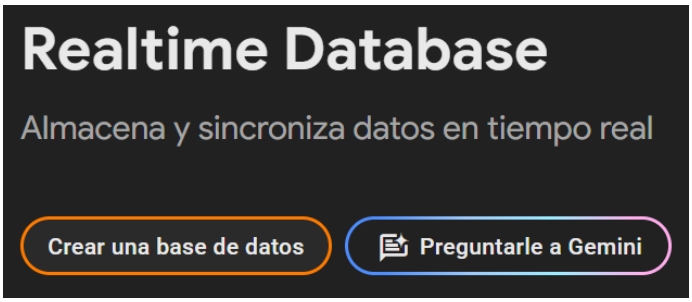
```
// Import the Firebase BoM
implementation(platform("com.google.firebase:firebase-bom:33.4.0"))

// TODO: Add the dependencies for Firebase products you want to use
// When using the BoM, don't specify versions in Firebase dependencies
implementation("com.google.firebase:firebase-analytics")
```

Tras sincronizar de nuevo el proyecto, ya estamos listos para trabajar con FireBase. Android Studio se habrá descargado todas la dependencias que necesita para funcionar, y con el fichero **google-services.json** ya tiene la configuración necesaria de acceso a FireBase.

Crear la BBDD en FireBase

En la consola de nuestro proyecto, hacemos **Todos los productos > Realtime Database** para empezar a configurar mi BBDD. Pulsamos en Crear, y seleccionamos modo de prueba.



Y ya tenemos la BBDD lista para ser utilizada. Ahora, debes recordar que esto NO es una base de datos SQL, por lo que no hay tablas ni nada parecido. Es una base de datos **jerárquica**. Ahora, vamos a definir una colección y un documento en nuestra Base de Datos.

En la pestaña de **reglas**, asegúrate de que tengas permisos de lectura y escritura:

```
{
  /* Visit https://firebase.google.com/docs/database/security to learn more about security rules. */
  "rules": {
    ".read": true,
    ".write": true
  }
}
```

Una vez completado, vamos a programar un acceso a esta BBDD desde nuestra App móvil.

Consulta desde tu App

Las consultas a una base de datos en FireBase son razonablemente sencillas de hacer. En primer lugar, hay que dar permisos a la App para acceder a Internet, si no lo has hecho ya, en el **AndroidManifest.xml**

```
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
```

Vamos hacer un primer ejemplo en el que recuperamos los elementos que dependen de la colección T_PATATAS.

```
class MainActivity : AppCompatActivity() {

    // Database reference
    private lateinit var database: DatabaseReference
```

Y ahora hacemos:

```
try {
    // We get the database reference
    database = Firebase.database.reference

    // Get the 'T_PATATAS' node. If the query is a success, we log the values returned.
    // If it is a failure, we log the error
    database.child( pathString: "T_PATATAS").get().addOnSuccessListener {
        Log.i( tag: "firebase",  msg: "Got value ${it.value}")
    }.addOnFailureListener{
        Log.e( tag: "firebase",  msg: "Error getting data", it)
    }
} catch (e: Exception) {
    // Something went wrong
    Log.e( tag: "exception", e.localizedMessage)
}
```

Este ejemplo es un poco **limitado**. Simplemente lanza la consulta contra FireBase y le solicita que le retorne todo lo que depende de T_PATATAS. Si la operación tiene éxito, se mostrará un mensaje de log; si falla, se mostrará un error. Nótese que el comportamiento es similar al de un evento. En nuestro caso, con el ejemplo del punto anterior, nos mostrará lo siguiente en la pestaña Logcat del Android Studio. Si no hubiese nada que devolver, se devolvería 'null'

```
.lorrieta.pruebasfirebase    D  tagSocket(1027) with statsTag=0xffffffff, statsUid=-1
.lorrieta.pruebasfirebase    I  Got value {cantidad=100, id=0, nombre=patata 0}
.lorrieta.pruebasfirebase    W  Verification of void androidx.profileinstaller.Prof
```

Expandiendo la consulta: Sacar dato a dato

Obviamente, con esto no vamos a ninguna parte. Vamos a dar formato a la respuesta de la base de datos FireBase para que nos cargue la respuesta en un Listado.

```
try {
    // We get the database reference, the 'root' of the hierarchy
    database = Firebase.database.reference

    // We get the data
    database.child( pathString: "T_PATATAS").get().addOnSuccessListener { document ->
        if (null != document){

            Log.i( tag: "firebase",  msg: "Got values: ${document.value}")

            val id = document.child( path: "id").getValue(Integer::class.java)
            val cantidad = document.child( path: "cantidad").getValue(Integer::class.java)
            val nombre = document.child( path: "nombre").getValue(String::class.java)

            Log.i( tag: "firebase",  msg: "Id: ${id}")
            Log.i( tag: "firebase",  msg: "Cantidad: ${cantidad}")
            Log.i( tag: "firebase",  msg: "Nombre: ${nombre}")
        } else {
            Log.e( tag: "firebase",  msg: "No data")
        }
    }.addOnFailureListener{
        Log.e( tag: "firebase",  msg: "Error getting data", it)
    }
} catch (e: Exception) {
    // Something went wrong
    Log.e( tag: "exception", e.localizedMessage)
}
```

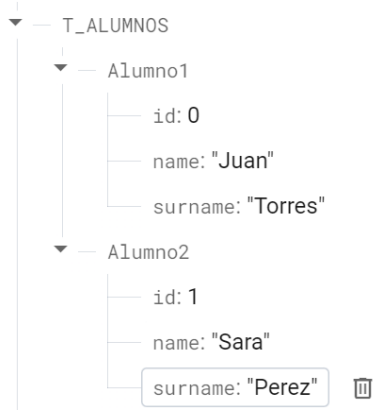
Esta modificación permite extraer de la colección sus hijos, lo que arroja el siguiente resultado por la pestaña de log:

```
orrieta.pruebasfirebase       I  Got value id: {cantidad=100, id=0, nombre=patata 0}
orrieta.pruebasfirebase       I  Got value id: 0
orrieta.pruebasfirebase       I  Got value cantidad: 100
orrieta.pruebasfirebase       I  Got value nombre: patata 0
```

Expandiendo la consulta: Múltiples elementos en la colección

Vamos a complicar las cosas. Hasta el momento, nuestra colección sólo tenía tres elementos: el id, nombre y cantidad de una única patata. Esto no tiene **ningún sentido**.

Vamos a construir otro ejemplo con alumnos, de los que tendremos a dos individuos: Juan y Sara. Para ello, añadimos al FireBase el siguiente esquema:



Para acceder a esta colección, hacemos:

```
// We get the data
database.child( pathString: "T_ALUMNOS").get().addOnSuccessListener { alumno ->
    if (null != alumno){

        Log.i( tag: "firebase",  msg: "Got values: ${alumno.value}")

        // We iterate the students...
        for (alum in alumno.children) {
            Log.i( tag: "firebase",  msg: "--- Value: ${alum.value}")

            val id = alum.child( path: "id").getValue(Integer::class.java)
            val name = alum.child( path: "name").getValue(String::class.java)
            val surname = alum.child( path: "surname").getValue(String::class.java)

            Log.i( tag: "firebase",  msg: "------ id: ${id}")
            Log.i( tag: "firebase",  msg: "------ name: ${name}")
            Log.i( tag: "firebase",  msg: "------ surname: ${surname}")
        }
    } else {
        Log.e( tag: "firebase",  msg: "No data")
    }
}.addOnFailureListener{
    Log.e( tag: "firebase",  msg: "Error getting data", it)
}
```

```
se        I  Got values: {Alumno1={surname=Torres, name=Juan, id=0}, Alumno2={surname=Perez, name=Sara, id=1}}
se        I  --- Value: {surname=Torres, name=Juan, id=0}
se        I  ------ id: 0
se        I  ------ name: Juan
se        I  ------ surname: Torres
se        I  --- Value: {surname=Perez, name=Sara, id=1}
se        I  ------ id: 1
se        I  ------ name: Sara
se        I  ------ surname: Perez
```

Expandiendo la consulta: Listar en el Android

Vamos a listar los contenidos de la base de datos en nuestra app móvil. Primero, generamos un POJO para guardar la información:

```
@IgnoreExtraProperties
data class Alumno(val id: Integer?, val nombre: String? = null, val surname: String? = null) {
    // Null default values create a no-argument default constructor, which is needed
    // for deserialization from a DataSnapshot.
}
```

Y a continuación, convertimos la información retornada por FireBase en POJOs. Una vez tengamos un listado de alumnos, enviarlo a una **ListView** es trivial.

```
findViewById<Button>(R.id.button).setOnClickListener {
    try {
        // We get the database reference, the 'root' of the hierarchy
        database = Firebase.database.reference
        // We get the data
        database.child( pathString: "T_ALUMNOS").get().addOnSuccessListener { alumno ->
            val myList = findViewById<ListView>(R.id.listView)
            val list = arrayListOf<Alumno>()
            if (null != alumno){
                // We iterate the students...
                for (alum in alumno.children) {
                    // We create the POJO
                    val al = Alumno(alum.child( path: "id").getValue(Integer::class.java),
                        alum.child( path: "name").getValue(String::class.java),
                        alum.child( path: "surname").getValue(String::class.java))
                    // Add to the list
                    list.add(al)
                }
```

```
                // Load the ListView
                val adapter = ArrayAdapter ( context: this, android.R.layout.simple_list_item_1, list)
                myList.adapter = adapter
            } else {
                Log.e( tag: "firebase",  msg: "No data")
            }
        }.addOnFailureListener{
            Log.e( tag: "firebase",  msg: "Error getting data", it)
        }
    } catch (e: Exception) {
        // Something went wrong
        Log.e( tag: "exception", e.localizedMessage)
    }
}
```

Expandiendo la consulta: Filtros, order by y demás movidas…

Para añadir este tipo de asuntos a la consulta, toca mirar en Google, pero es razonablemente fácil de mirar. Por ejemplo, para devolver por orden de id, primero añadimos una regla a nuestra base de datos:

```
{
  /* Visit https://firebase.google.com/docs/database/security to learn more about security rules. */
  "rules": {
    ".read": true,
    ".write": true,
    "T_ALUMNOS": {
      ".indexOn": "id"
    }
  }
}
```

Y luego modificamos una línea:

```
    // We get the database reference, the 'root' of the hierarchy
    database = Firebase.database.reference
    // We get the data
    database.child( pathString: "T_ALUMNOS").orderByChild( path: "id")
        .get().addOnSuccessListener { alumno ->
```