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.

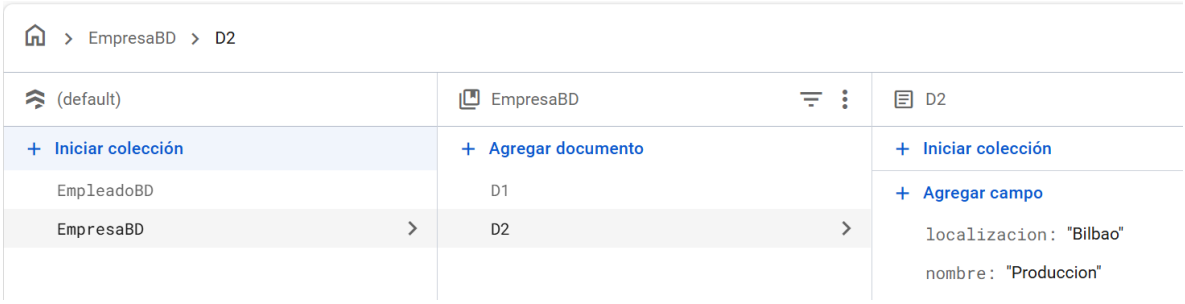
Nosotros vamos a servirnos de FireBase como una Base de Datos On-Line. A diferencia de otras bases de datos, **no es SQL** sino orientada a **documentos**. Tienes más información sobre Firebase en la documentación de la **UD05 – Firebase** de la asignatura de **ADT**. Nos basaremos en esa teoría y explicaciones para generar los ejemplos y apuntes necesarios para usar Firebase desde Kotlin en Android.

Configurar el proyecto en Android Studio

Vamos a suponer, para este ejemplo, que tienes ya creado el proyecto en Firebase:

https://firebase.google.com/

Nuestra Firestore Database es similar a esta:



Antes de hacer nada, vamos a preparar el proyecto de Android para que utilice Firestore Database. Por favor, mucho cuidado y paciencia, que si metes la pata, es difícil arreglarlo después y tendrás que empezar de cero.

Gradle es un sistema de automatización de compilación. Su función principal es gestionar *cómo se construye* tu aplicación, incluyendo:

- **Compilación del código** (Java, Kotlin, etc.)
- **Gestión de dependencias** (bibliotecas externas como Firebase o Retrofit)
- **Empaquetado de la app** (APK o AAB)
- **Automatización de tareas**

En Android Studio, tú no te limitas a darle al Run porque ya se ha compilado por detrás el código de tu proyecto. Literalmente ejecutas un script (para entendernos) que va leyendo diferentes ficheros de configuración que le dice lo que tiene que ir haciendo. Por ejemplo, si tú le indicas una dependencia de "usar el GSON", el Gradle lo va a buscar, lo va a descargar, y te lo va a dejar disponible en tu código para que lo uses.

En Android Studio, Gradle utiliza archivos de configuración llamados **build.gradle** (si usas Groovy) o **build.gradle.kts** (Si usas Kotlin DSL). Nosotros en este ejemplo usamos lo segundo. Esto lo puedes elegir cuando creas un proyecto nuevo.

En un proyecto Android típico hay dos niveles de Gradle:

- **A nivel de proyecto**: fichero **settings.gradle.kts**, define los módulos del proyecto y configuraciones globales

- **A nivel de App**: fichero **build.gradle.kts**, que se encuentra dentro de app/. Especifica cómo se compila ese módulo, sus dependencias, plugins, y variantes de build.

Vamos por pasos.

PASO 1- build.gradle.kts (Proyecto)

Hay que decirle que vamos a usar los **servicios de Google**. El fichero te tiene que quedar así:

```
// Top-level build file where you can add configuration options common to all sub-projects/modules.
plugins {
    alias(libs.plugins.android.application) apply false
    alias(libs.plugins.kotlin.android) apply false

    // Google services
    alias(libs.plugins.google.services) apply false
}
```

Le das a sincronizar.

Y te da error. Eso es porque te falta añadir entradas al fichero **libs.versions.toml**. Este fichero se utiliza para tener todas las versiones de las librerías etc. en un mismo sitio de forma centralizada. Si lo abres, tendrás que añadir:

```
[versions]
agp = "8.13.0"
kotlin = "2.2.20"
googleServices = "4.4.4"
```

Al principio del **libs.versions.toml** y esto el final:

```
[plugins]
android-application = { id = "com.android.application", version.ref = "agp" }
kotlin-android = { id = "org.jetbrains.kotlin.android", version.ref = "kotlin" }
google-services = { id = "com.google.gms.google-services", version.ref = "googleServices"}
```

Le das a sincronizar.

Y no debería darte más errores. Esto NO es necesario, pero es una ayuda para organizar las cosas. Puedes toquetear a mano lo que quieras, e incluso el propio Android Studio te va dando sugerencias a modo de warnings si detecta que no estás usando las últimas versiones etc.

Al darle a sincronizar, el Android Studio descargará todas las dependencias que necesites de la versión que necesites, y podrás usarlas desde tu código.

PASO 2- build.gradle.kts (App)

Ahora, abrimos el otro fichero y le añadimos las dependencias. Primero, arriba del todo:

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

    // Add the Google services Gradle plugin
    id("com.google.gms.google-services")
}
```

En este caso NO estamos haciendo uso del **libs.versions.toml**. No porque no se pueda, sino a modo ilustrativo. A continuación, abajo, añadimos las dependencias:

```
dependencies {

    implementation(libs.androidx.core.ktx)
    implementation(libs.androidx.appcompat)
    implementation(libs.material)
    implementation(libs.androidx.activity)
    implementation(libs.androidx.constraintlayout)
    testImplementation(libs.junit)
    androidTestImplementation(libs.androidx.junit)
    androidTestImplementation(libs.androidx.espresso.core)

    // Import the Firebase BoM
    implementation(platform( notation = "com.google.firebase:firebase-bom:33.0.0"))

    // Firestore KTX (para Kotlin)
    implementation("com.google.firebase:firebase-firestore-ktx")

    // Now you can add the dependencies for Firebase products you want to use
    // When using the BoM, don't specify versions in Firebase dependencies
    //implementation(libs.firebase.analytics) // Optional
    implementation("com.google.firebase:firebase-analytics") // Optional

}
```

De nuevo, no estamos usando el **libs.versions.toml**.

Le das a sincronizar.

Y todo debería ir bien.

NOTA

Por motivos que se me escapan, los de Firebase han dejado de sacar nuevas versiones de los módulos KTX, y han quitado las librerías KTX de Firebase Android BoM (v34.0.0).

Por tanto, si NO pones la 33.0.0, no tendrás ningún error de Gradle, pero en la MainActivity no te deja importar la librería "FirebaseFirestore" si queremos hacer:

private val db = FirebaseFirestore.getInstance().

¿Qué está mal? Nada en tu código ni en el Gradle. Simplemente, que no se descarga la librería "FirebaseFirestore" porque la versión 34.0.0 o superior no la tiene. Entonces, ¿Qué hay que usar entonces? Su API. Pero como el cambio lo han hecho en **Julio del 2025** hay que apañarse con una versión anterior para que todo funcione... en fin…

PASO 3- settings.gradle.kts

Asegúrate de que tienes los repositorios de Google y Maven Central puestos. Deberías ver algo así:

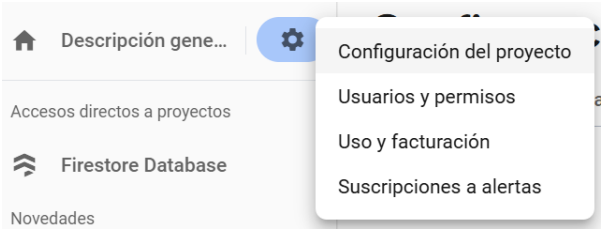
```
pluginManagement {
    repositories {
        google {
            content {
                includeGroupByRegex( groupRegex = "com\\.android.*")
                includeGroupByRegex( groupRegex = "com\\.google.*")
                includeGroupByRegex( groupRegex = "androidx.*")
            }
        }
        mavenCentral()
        gradlePluginPortal()
    }
}
dependencyResolutionManagement {
    repositoriesMode.set(RepositoriesMode.FAIL_ON_PROJECT_REPOS)
    repositories {
        google()
        mavenCentral()
    }
}

rootProject.name = "FireBaseEmpresas"
include( ...projectPaths = ":app")
```

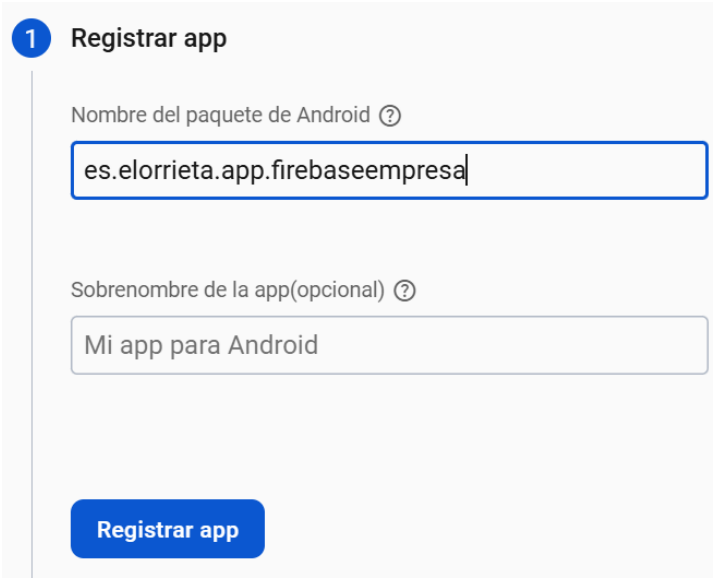
Credenciales

Para poder acceder desde tu app a la Base de Datos es necesario primero **registrarla** en tu proyecto de Firebase. Esto nos genera un fichero json que añadiremos a nuestro proyecto Android, y que funcionará como identificador.

En la web de Firebase, vamos a **Configuración del Proyecto**, en la pestaña **General** bajamos hasta encontrar **Tus apps**. Allí le damos a **Agregar App.**

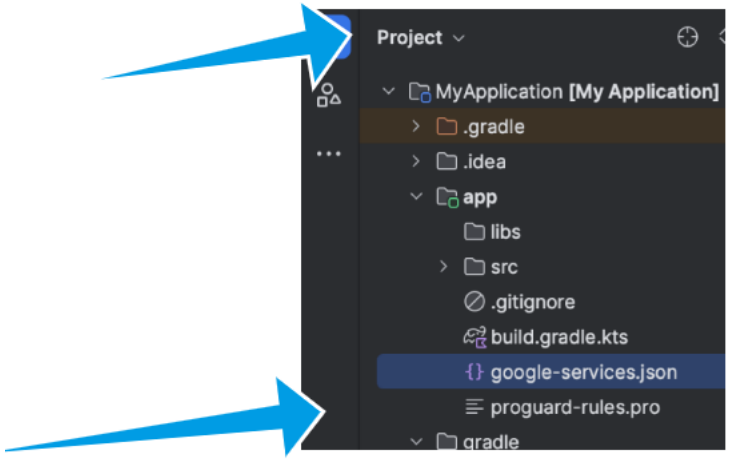


En la ventana emergente, le damos a Android y después indicaremos la ruta de paquetes de nuestra app de Android en la que se encuentra nuestra Actividad principal.



La siguiente ventana nos permitirá descargar un fichero **google-services.json** que deberemos incluir en nuestro proyecto Android. Verás que al completar este registro, nos aparecerá una entrada informativa en **Tus apps**.

El fichero **google-services.json** tiene que añadirse en una ruta específica: dentro de la carpeta app de tu proyecto, como ves en la imagen.



No te preocupes si lo añades, pero no lo ves en el Android Studio. Estar, está. Y si no lo tienes, te dará error. Tranquilo.

Acceso a Internet

Esto ya lo sabrás, pero para que tu App tenga salida a Internet hay que darle permisos. A modo de recordatorio, vas al **AndroidManifest** y le añades:

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

Consulta desde tu App

Las consultas a una base de datos en FireBase son razonablemente sencillas de hacer. Tienes un ejemplo funcional explicado con este documento, aquí solamente mencionaremos las partes importantes.

Para acceder a la base de datos, hay que instanciarla:

```
// Firestore Instance
private val db = FirebaseFirestore.getInstance()
```

Vamos a añadir un evento a un botón para que nos cargue en un RecyclerView nos cargue todas las Empresas:

```
findViewById<Button>( id = R.id.buttonFindAllEnterprises).setOnClickListener {
    // A Mutable List in Kotlin is a list whose contents can be changed after it is created
    val enterpriseList = mutableListOf<Enterprise>()

    // Access to DB to retrieve "EmpresaBD" Documents. Note there are TWO possible events
    // for the call: OnSuccess y OnFailure. We must provide behaviour for both
    db.collection( collectionPath = "EmpresaBD")
        .get()
        .addOnSuccessListener { result ->
            // So, everything went OK. This DOES NOT MEAN we found any Documents
            // We loop and retrieve each Document individually
            // NOTE the Document ID must be assigned manually
            for (document in result) {
                val enterprise = document.toObject( valueType = Enterprise::class.java)
                enterprise.id = document.id // The ID must be assigned manually
                enterpriseList.add(enterprise)
                Log.d( tag = "Firestore",  msg = "Enterprise: $enterprise")
            }
            // We update the adapter
            enterprisesAdapter.update( newEnterprises = enterpriseList)
        }
        .addOnFailureListener { exception ->

            // Something went wrong. This DOES NOT MEAN there was no Documents. This means
            // something serious, like server errors, etc.
            Toast.makeText(
                context = this,
                text = "Error when reading enterprises - $exception",
                duration = Toast.LENGTH_SHORT
            ).show()
            Log.e( tag = "Firestore",  msg = "Error when reading enterprises",  tr = exception)
        }
}
```

El código es extremadamente simple. Hacemos la petición con db.collection("EmpresaBD"). Esto nos va a devolver una serie de Documentos. Lo raro es cómo se tratan las respuestas de la Base de Datos: mediante **eventos**.

Si todo va bien, entonces se va a ejecutar la parte del **addOnSuccessListener**. Se recorren los resultados uno a uno, se convierten en objetos Enterprise, y se le pasan al adapter para cargar la tabla. Ten cuidado, que las ID (referencias internas de Firebase) se tienen que cargar explícitamente a mano.

```
for (document in result) {
    val enterprise = document.toObject( valueType = Enterprise::class.java)
    enterprise.id = document.id // The ID must be assigned manually
    enterpriseList.add(enterprise)
    Log.d( tag = "Firestore",  msg = "Enterprise: $enterprise")
}
```

Y recuerda: que no haya Documentos en la respuesta NO SIGNIFICA que haya algún error. Simplemente, que no hay Documentos. Es una respuesta válida.

Si algo ha pasado, entonces se va a ejecutar el **addOnFailureListener**. En este caso, nso limitaremos a informar del Error.

Enterprise

Un último detalle. Cómo definamos la clase Enterprise que usaremos para gestionar los Documentos es muy importante. Hay que inicializar TODOS sus atributos a sus valores por defecto, dado que Firestore necesita constructores vacíos para serializar y deserializar correctamente las cosas.

```
// Los valores por defecto (= "", = 0) son necesarios porque Firestore
// necesita un constructor vacío para deserializar.
data class Enterprise(var id: String = "",  6 Usages
                      var nombre: String = "",
                      var localizacion: String = "")
```