

FIREBASE con MVVM

1. Crea un proyecto: File→New Project → Empty Views Activity , lo llamaremos, por ejemplo, **FirestoreEjemplo**.

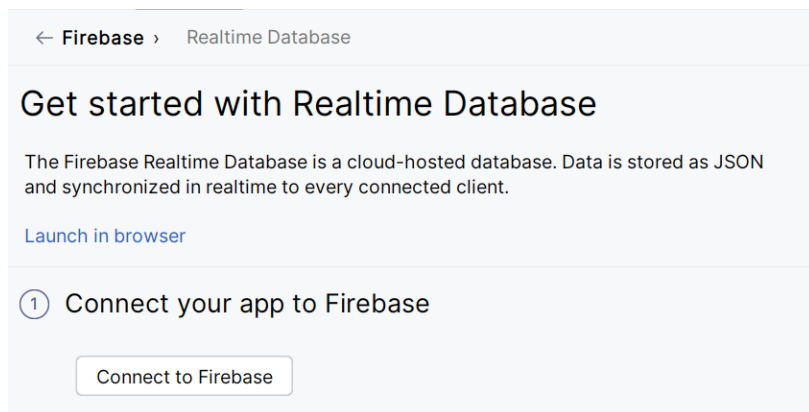
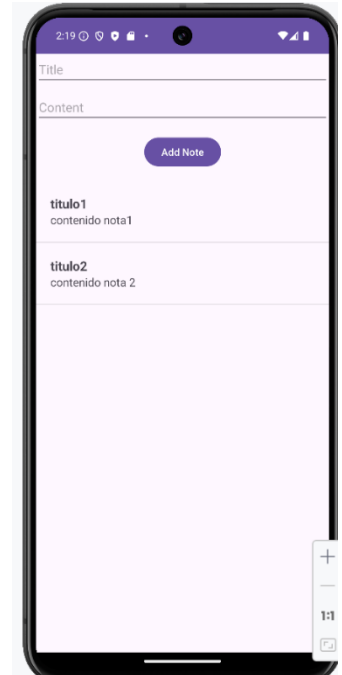
Vamos a crear una app, que añada y recupere datos en tiempo real de una base de datos Firebase, usando el patrón MVVM.

Firebase Realtime Database: Almacena y sincroniza datos en una base de datos NoSQL alojada en la nube. Los datos se almacenan en formato JSON y se sincronizan con todos los clientes en tiempo real y se mantienen disponibles cuando la app no tiene conexión.

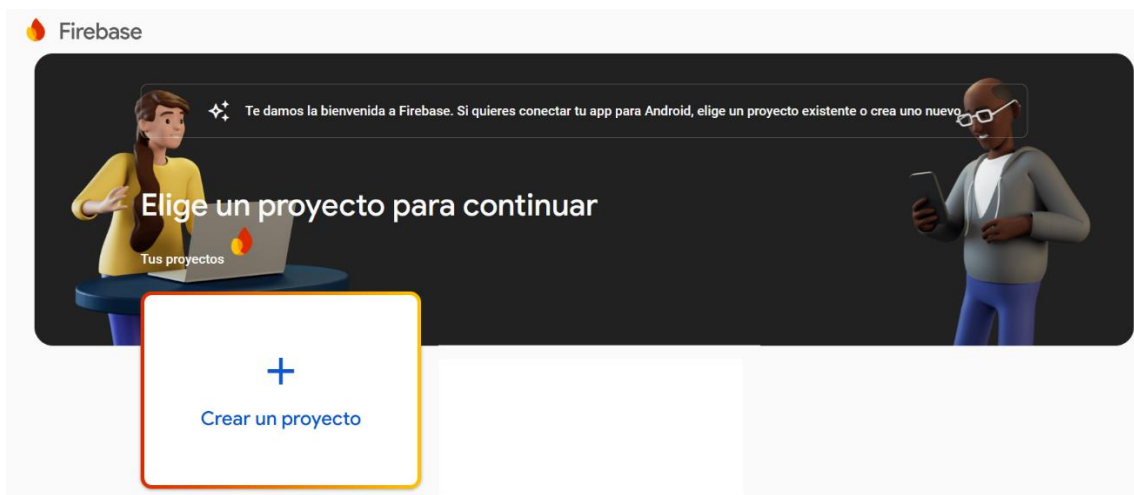
Lo primero que vamos a hacer es en el archivo *build.gradle* de la app poner `compileSdk = 35`

Más Información en: https://firebase.google.com/docs/database?utm_source=studio&hl=es-419

2. Como vamos a usar Firebase, ir a **Tools/Firebase**, selecciona **Realtime Database / Get Started with Realtime Database** donde deberás realizar los pasos que se indican a continuación:



- Pulsa en **Connect Your App to Firebase**. Se abrirá una página web que da acceso a Firebase Console: <https://console.firebase.google.com/> desde la cual te pedirán crear un proyecto



Una vez que has pulsado en **Crear un proyecto**, debes poner exactamente el mismo nombre de Proyecto que creaste en Android Studio y pulsas en **Continuar**, hasta pulsar el botón **Crear Proyecto**.



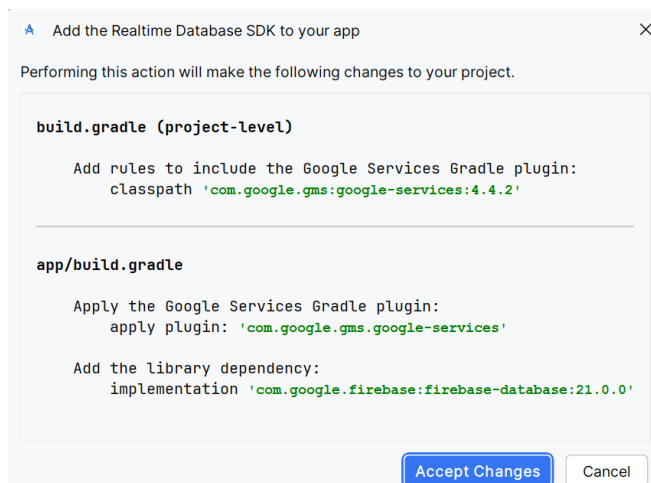
Cuando el proyecto esté creado, te aparecerá el siguiente mensaje. Pulsar en **Conectar**. Veremos cómo se abre una página web en <http://localhost:62521/>, a partir de ese momento podemos usar Firebase.



Continuamos con los pasos que nos indicaban desde Tools/Firebase. Pulsaremos en **“Add the Realtime Database SDK to your app”**, donde nos añadirán el plugin y dependencias necesarias a nuestro proyecto, para lo que debemos pulsar en **Accept Changes**.

2 Add the Realtime Database to your app

Add the Realtime Database SDK to your app



```
build.gradle (project-level)

Add rules to include the Google Services Gradle plugin:
classpath 'com.google.gms:google-services:4.4.2'

app/build.gradle

Apply the Google Services Gradle plugin:
apply plugin: 'com.google.gms.google-services'

Add the library dependency:
implementation 'com.google.firebase:firebase-database:21.0.0'
```

Como además queremos usar la librería de **Firestore Android BOM**, para manejar más fácil la librería de Firestore, debemos también añadir la siguiente dependencia:

```
implementation platform('com.google.firebase:firebase-bom:33.7.0')
```

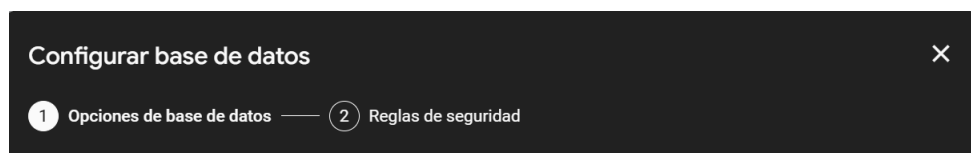
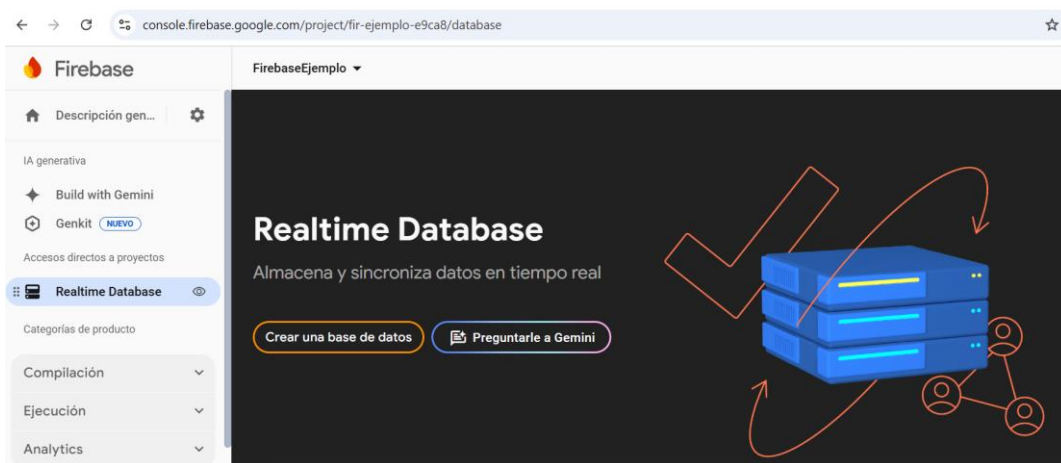
Tras estos cambios, el archivo `build.gradle` de la app tendrá un nuevo plugin:

```
plugins {  
    alias(libs.plugins.android.application)  
    alias(libs.plugins.kotlin.android)  
    alias(libs.plugins.google.gms.google.services)  
}
```

Nuevas dependencias:

```
implementation(libs.firebase.database)  
implementation(platform(libs.firebase.bom))
```

El siguiente paso que debemos hacer es **crear la base de datos**, para lo cual debes acceder a **Firestore Console** (<https://console.firebase.google.com/>), selecciona el proyecto que acabas de crear, pulsa en el apartado **Realtime Database** y después en **Crear una base de datos** y seguir cada uno de los pasos que van indicando.



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

Ubicación de Realtime Database

Bélgica (europe-west1)

Cancelar

Siguiente

Configurar base de datos



Opciones de base de datos



Reglas de seguridad

Cuando definas la estructura de los datos, **deberás crear reglas para protegerlos**.

[Más información](#)



Comenzar en modo bloqueado

Tus datos son privados de forma predeterminada. 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 dentro de 30 días a fin de habilitar el acceso de lectura/escritura a largo plazo para los clientes.

```
{
  "rules": {
    ".read": false,
    ".write": false
  }
}
```



Se denegarán todas las operaciones de lectura y escritura de terceros

Cancelar

Habilitar

Tras pulsar **Habilitar**, vamos a modificar las reglas, poniendo el **acceso de lectura y escritura público**, para que no tengamos problemas. Tras poner lectura y escritura a **true**, pulsamos en **Publicar** para guardar los cambios.

Más información sobre las reglas de seguridad:
<https://firebase.google.com/docs/database/security?hl=es&authuser=0>

3. Crea una data class para representar los datos que deseas almacenar en Firebase (modelo). Debe tener un constructor vacío, de ahí que admitan null y se inicialicen con null si no se pasa ningún valor.

```
package model

data class Note(var idNote: String? = null,
                var titulo: String? = null,
                var contenido: String? = null)
```

4. Crear la capa lógica, para manejar las interacciones de datos con Firebase:

Esta clase **NoteRepository** sirve como intermediaria entre ViewModel y la fuente de datos, en este caso Firebase. En esta clase, crearemos un método para leer las notas y otro para añadir.

Al hacer `FirebaseDatabase.getInstance` debes pasar la URL de acceso a tu base de datos, cópiala de Firebase console / Realtime database.

Más información sobre como Leer y Escribir datos en Firebase:
https://firebase.google.com/docs/database/android/read-and-write?utm_source=studio&hl=es-419

```
package persistence

import androidx.lifecycle.MutableLiveData
import com.google.firebase.database.DatabaseReference
import com.google.firebase.database.FirebaseDatabase
import model.Note

class NoteRepository {
    private var databaseReference: DatabaseReference
    init {
        databaseReference =
        FirebaseDatabase.getInstance(URL_REFERENCIA_REALTIME_DATABASE_FIREBASE).reference
    }

    fun addNote(note: Note) {
        val key = databaseReference.child("notes").push().key
        note.idNote = key
        databaseReference.child("notes").child(key!!).setValue(note)
    }

    fun getNotes(): MutableLiveData<List<Note>> {
        val notes = MutableLiveData<List<Note>>()

        // Cada vez que los datos cambien, se llamará al evento onDataChange con
        // la nueva lista de datos
        val firebaseDataListener = FirebaseDataListener(notes)

        databaseReference.child("notes").addValueEventListener(firebaseDataListener)

        return notes
    }
}
```

Clase `FirestoreDataListener`: clase que implementa `ValueEventListener` para actualizar datos con `Firestore Realtime Database`.

```
package persistence

import android.util.Log
import androidx.lifecycle.MutableLiveData
import com.google.firebase.database.DataSnapshot
import com.google.firebase.database.DatabaseError
import com.google.firebase.database.ValueEventListener
import model.Note

class FirestoreDataListener(var dataList: MutableLiveData<List<Note>>) :
    ValueEventListener {
    override fun onDataChange(snapshot: DataSnapshot) {
        val notes = mutableListOf<Note>()
        for (childSnapshot in snapshot.children) {
            val note = childSnapshot.getValue(Note::class.java)
            if (null != note) {
                notes.add(note)
            }
        }
        dataList.postValue(notes)
    }

    override fun onCancelled(error: DatabaseError) {
        // Handle error, e.g., log or show a message
        Log.e("FirestoreDataListener", "Error: ${error.message}")
    }
}
```

Clase `NoteViewModel`: Actúa como intermediario entre el `Repository` y la IU.

```
package viewmodel

import android.app.Application
import androidx.lifecycle.AndroidViewModel
import androidx.lifecycle.LiveData
import model.Note
import persistence.NoteRepository

class NoteViewModel(application: Application) : AndroidViewModel(application) {
    private val repository: NoteRepository
    lateinit var notesLiveData: LiveData<List<Note>>

    init {
        repository = NoteRepository()
        getNotes()
    }

    fun addNote(titulo: String, contenido: String) {
        val note = Note(null, titulo, contenido)
        repository.addNote(note)
    }

    private fun getNotes() {
        notesLiveData = repository.getNotes()
    }
}
```

5. Crear la parte de la vista o capa de Interfaz de Usuario:

Como último punto se debe crear la parte de Interfaz de Usuario como Activity/Fragment desde donde se deberá llamar a NoteViewModel siempre que se quiera realizar una llamada a base de datos.