Wiki.Local

Firebase Real Time Database

OBJETIVO

En esta guía aprenderemos a manejar una Base de Datos en tiempo real usando Firebase Real Time Database, para que a futuro puedas implementarlo en tus aplicaciones.

Firebase

Firebase es una plataforma en la nube para desarrollar aplicaciones web y aplicaciones móviles.

Al utilizar esta plataforma en nuestros proyectos tendremos una serie de ventajas:

- Sincronización, los datos de nuestro proyecto se sincronizará casi de forma automática sin necesidad de usar código complejo.
- Multiplataforma, podremos integrar fácilmente con plataformas web como con otras aplicaciones móviles, es compatible con IOS, Android Unity y C++ y Javascript.
- Infraestructura Google, al usar Google nuestro proyecto será fácilmente escalable.
- No se necesita servidor, al estar la plataforma en la nube no es necesario la creación de un servidor para nuestro proyecto.

Herramientas de Firebase

Firebase dispone un conjunto de soluciones, entre las cuales podemos destacar aquellos servicios que se utilizan para el desarrollo de un proyecto de tipo app móvil o web. Los cuales son:

Realtime Database

Es la herramienta más destacada de Firebase que permite implementar una Base de datos en tiempo real, el cual esta alojado en la nube y es una Base de Datos NoSQL que almacena sus datos en formato JSON.

Firebase dispara eventos automáticamente a las aplicaciones cuando los datos cambian, inclusive cuando estas no estuviesen disponibles Firebase Real Time implementará mecanismos para actualizar los datos una vez restablecida la conexión.

Cloud Firestore

Es una nueva Base de Datos NoSQL flexible, escalable y en la nube para el desarrollo de aplicaciones móviles y aplicaciones web. Con Cloud Firestore se puede realizar búsquedas, transacciones y ordenamientos avanzados además que los documentos estan organizados en colecciones a diferencia de un arbol JSON simple que viene en Realtime Database.

En el siguiente enlace puede encontrar las consideraciones para elegir entre Realtime Database o Cloud Firestore Encuesta consideraciones clave [https://firebase.google.com/docs/firestore/rtdb-vs-firestore?hl=es-419#key_considerations]

Autenticación de Usuarios

Firebase ofrece un sistema de autenticación que permite el autenticar usando proveedores de otras plataformas (como Facebook, Google, Microsoft y Otros) como tambien mediante el uso de usuario y contraseña.

Almacenamiento en la nube

Destinado para el almacenamiento de ficheros de las aplicaciones

Cloud Messaging

Herramienta destinada para el envio de notificaciones y mensajes en tiempo real a diversas aplicaciones o plataformas.

Hosting

Firebase ofrece un servidor para alojar app basadas en js con las funcionalidades de CDN.

Pasos para implementar Firebase Realtime

La ruta de implementación que se debe seguir para implementar Firebase Realtime en nuestro proyecto android consiste en:

1. Crear una Base de Datos
2. Agregar el SDK de Firebase Realtime Database al proyecto
3. Crear la referencia de Realtime Database
4. Realizar operaciones de escritura
5. Realizar operaciones de lectura

1. Crear una Base de Datos Realtime

Para este paso se necesita crear un proyecto desde el Firebase Console [https://console.firebase.google.com/project/_/database?hl=es-419] con una cuenta Google (@gmail.com). Hacemos clic en agregar proyecto y seguimos las instrucciones hasta finalizar la operación

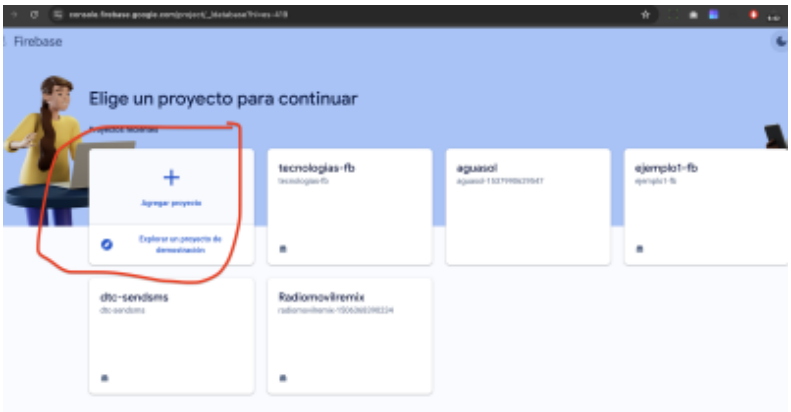


Figure 1: Pantalla inicial de Firebase Console

Nombre del proyecto

Al hacer clic en agregar proyecto se abrirá el formulario de creación del proyecto, en el primer paso debemos escribir el nombre de nuestro proyecto en este caso lo llamaremos **dis413-db**, luego hacemos clic en continuar

Figure 2: Paso 1 de la creación de la Base de Datos

google Analytics

En el siguiente paso se nos pedirá habilitar Google Analytics, para este proyecto lo deshabilitaremos y hacemos clic en crear proyecto.

==== Creación de la Base de Datos ====

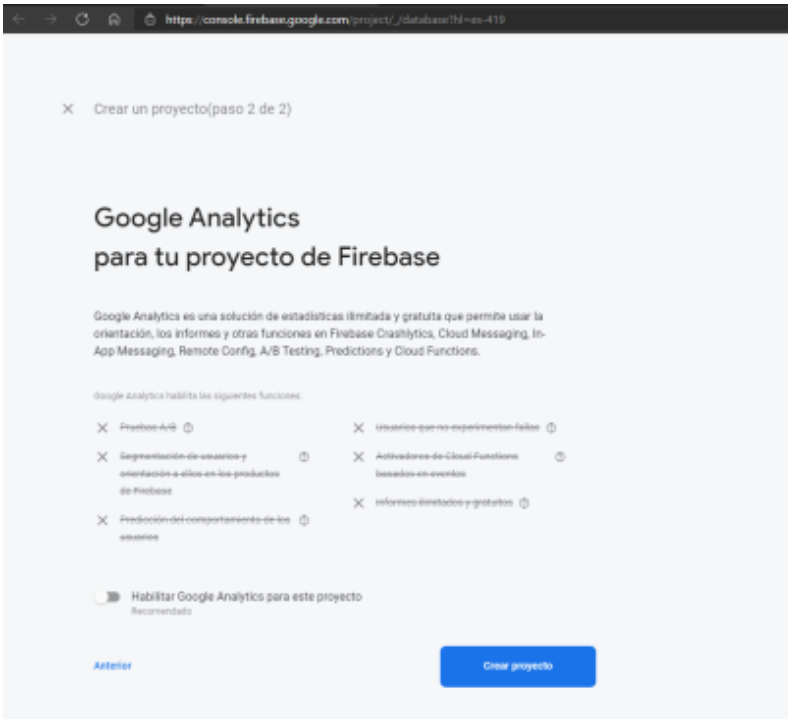


Figure 3: Paso 2 de la creación de la Base de Datos

Pantalla principal del proyeto

Una vez que termine el proceso de creación ya podremos acceder a la consola principal de administración de nuestro proyecto, desde el cual podemos acceder a los diferentes servicios que ofrece Firebase para asociar a nuestro proyecto.

Figure 4: Pantalla principal del proyecto **dis413-db**

RealTime Database

En el menú de la izquierda hacemos clic en **Realtime Database** y la primera vez hacemos clic en crear una Base de Datos

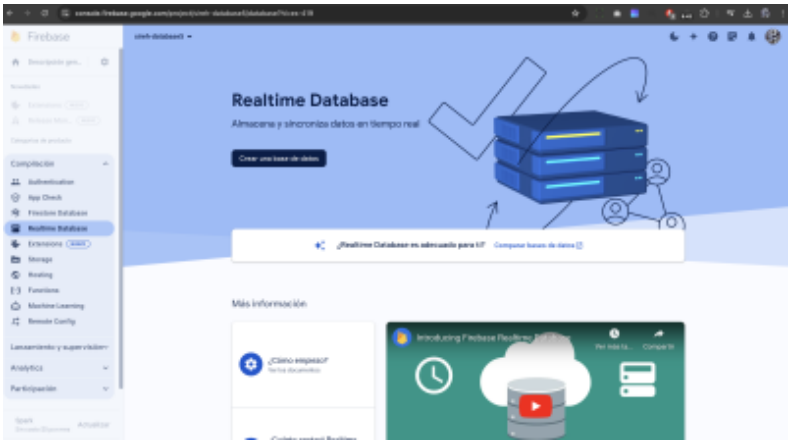


Figure 5: Pantalla principal del Proyecto

Selección de región

Al momento de crear la base de datos nos pedirá la ubicación de nuestra Base de datos, seleccionamos Estados Unidos (us-central) y las reglas de seguridad seleccionamos *Comenzar en* ***modo de prueba*** y finalmente hacemos clic en habilitar

Figure 6: Pantalla de configuración de la Base de Datos

Finalmente nuestro proyecto ya esta configurado para usar Realtime Database.

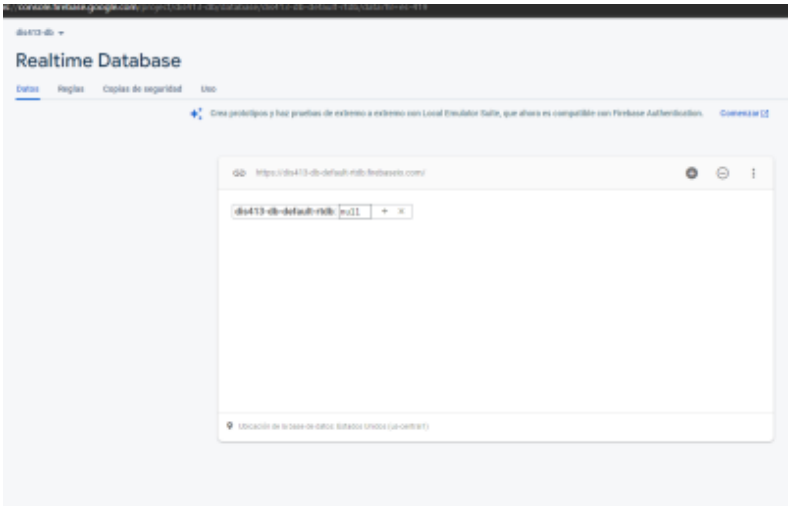


Figure 7: Base de Datos Realtime Database que se acaba de crear

2. Agregar el SDK de Firebase Realtime Database al proyecto

Este paso se lo puede hacer de forma manual agregando las dependencias a Android Studio o usar la herramienta Tools→Firebase. La segunda opción realizará las configuraciones necesarias de forma automática en nuestro proyecto Android Studio, siguiendo unos sencillos pasos

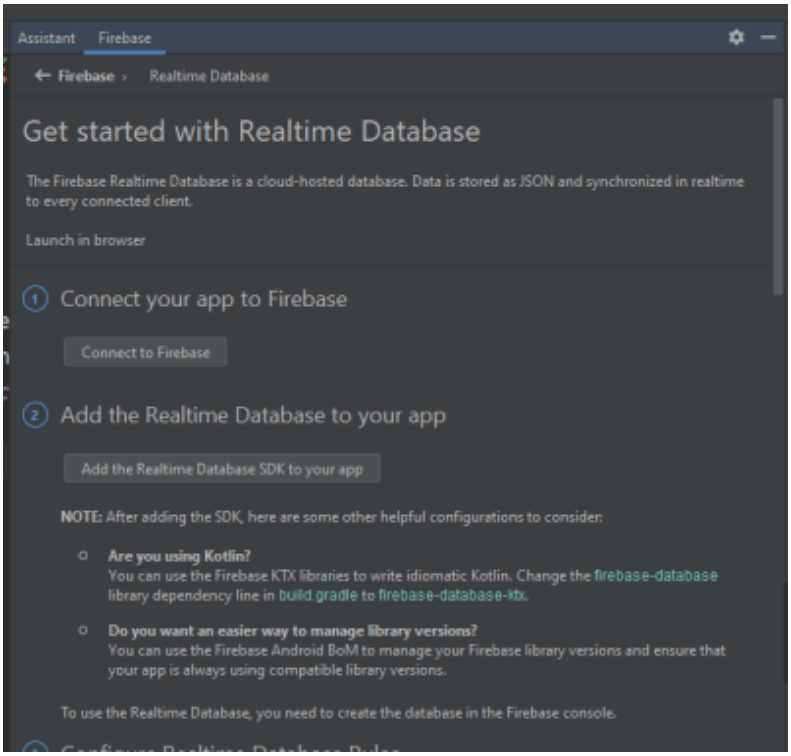
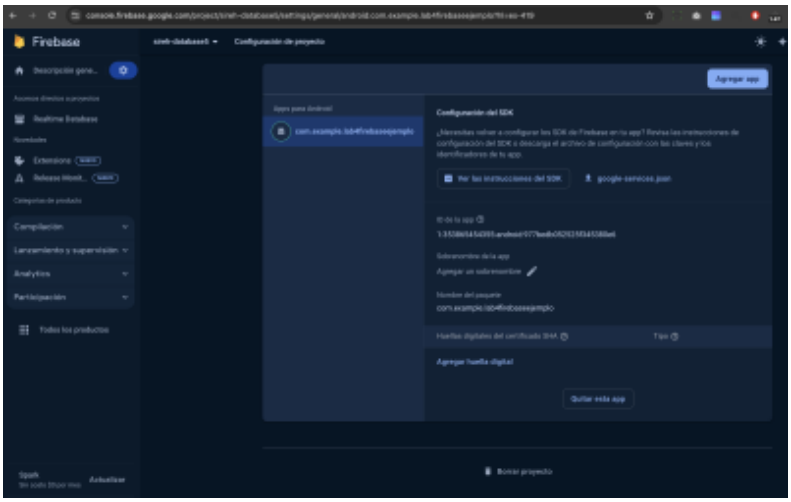


Figure 8: Asistente Firebase de android Studio

Si se añadio correctamente firebase a nuestro proyecto android podemos ver la lista de proyectos agregados en la descripción del proyecto desde la consola del firebase:



3. Crear la referencia de Realtime Database

Para leer o escribir en la base de datos, se necesita una instancia de `DatabaseReference`

```
val : DatabaseReference = Firebase.database.reference
```

4. Realizar operaciones de escritura

Para ejecutar operaciones básica de escritura, se utiliza el método `setValue()` de una referencia específica. Por ejemplo si queremos agregar un nuevo nodo a la referencias mensajes, previamente debemos tener definido un objeto de tipo Mensajes:

```
data class Pais(val id:Int,val nombre:String,val id_estado:String)
```

Agregar

Para generar un id, se utiliza el método `dbRef.push().getKey()`

```
val id: String? = dbRef.push().key
val p:Pais = Pais(id!!,"ARGENTINA","A")
dbRef.child("paises").child(id).setValue(p)
```

Modificar

Para modificar debemos conocer el id del nodo que queremos actualizar:

```
val m:Pais = Pais("-NvkrVZl_vsKp3cAjti-","PERU MODIFICADO","A")
       dbRef.child("paises").child(m.id).setValue(m).addOnSuccessListener {
           Log.i("TEOTEO","Se actualizo correctamente")

       }.addOnFailureListener {
           Log.i("TEOTEO","Se produjo un error al actualizar "+it.message)

       }
```

Eliminar

Para realizar un borrado de un nodo se invoca el método `removeValue()`

```
dbRef.child("paises").child("-NvkrRFTAh3FxqS-noCo").removeValue()
```

5. Realizar operaciones de lectura

Para leer datos de una ruta y detectar cambios, se usa el método `addValueEventListener()`, el cual tiene un método `onDataChange()` que lee y detecta cambios en el contenido de la referencia. La devolución de llamada del evento recibe una instantanea (`DataSnapshot`) que contiene los datos actuales incluido los secundarios.

```
val paises = mutableListOf<Pais>()
    fun listar(){
        val paisesListener = object : ValueEventListener {
            override fun onDataChange(dataSnapshot: DataSnapshot) {
                //val post = dataSnapshot.getValue()
                var paisesText = ""
                for (snapshot in dataSnapshot.children){
                    val p = snapshot.getValue(Pais::class.java)

                    if (p != null) {
                        Log.i("TEOTEOLS","AGREGANDO A LA LISTA "+p.nombre)
                        paises.add(p)
                        paisesText += "${p?.nombre}\n"
                    }

                    //Log.i("TEOTEOLS",snapshot.key.toString())
//                    val p = snapshot.child(snapshot.key.toString()).value
//                    Log.i("TEOTEOLS",p.toString())
                }
                jsonText.text = paisesText




            }

            override fun onCancelled(databaseError: DatabaseError) {
                Log.i("TEOTEOLS","CANCELED")
            }
        }
        dbRef.child("paises").addValueEventListener(paisesListener)
    }
```

En general ValueEventListener se debe utilizar a fin de leer datos para recibir notificaciones sobre las actualizaciones de los datos del backend. Si se necesita leer datos una sola vez, puede usar `addListenerForSingleValueEvent`.

```
val paises = mutableListOf<Pais>()
fun listarSinEscucharCambios(){
        dbRef.child("paises").addListenerForSingleValueEvent(object : ValueEventListener {
            override fun onDataChange(dataSnapshot: DataSnapshot) {
                paisesList.clear()
                var paisesText = ""
                for (snap in dataSnapshot.children) {
                    val m = snap.getValue(Pais::class.java)
                    if(m != null){
                        paisesList.add(m)
                        paisesText += "${m?.nombre}\n"
                    }

                }
                jsonText.text = paisesText
            }

            override fun onCancelled(databaseError: DatabaseError) {
                // Manejar la cancelación, si es necesario
            }
        })
    }
```

Cuando se trabaja con listas una aplicación debe escuchar otros eventos secundarios como ser (cuando se agrega un nodo, cuando cambia un nodo, cuando se elimina un nodo). Para escuchar estos eventos secundarios se deba añadir un `ChildEventListener` a la referencia de la base de datos:

```
paisesList.clear()
dbRef.child("paises").addChildEventListener(object : ChildEventListener {
            override fun onChildAdded(dataSnapshot: DataSnapshot, previousChildName: String?) {
                val m = dataSnapshot.getValue(Pais::class.java)
                if(m != null){
                    paisesList.add(m)
                    jsonText.text = jsonText.text.toString()+"${m.nombre}\n"
                }
```

```
            }

            override fun onChildChanged(dataSnapshot: DataSnapshot, previousChildName: String?) {
                val m = dataSnapshot.getValue(Pais::class.java)
                Toast.makeText(applicationContext, "Se actualizó ${m?.nombre}", Toast.LENGTH_SHORT).show()
            }

            override fun onChildRemoved(dataSnapshot: DataSnapshot) {
                Toast.makeText(applicationContext, "Se eliminó", Toast.LENGTH_SHORT).show()
            }

            override fun onChildMoved(dataSnapshot: DataSnapshot, previousChildName: String?) {
                Toast.makeText(applicationContext, "Se ordenó", Toast.LENGTH_SHORT).show()
            }

            override fun onCancelled(databaseError: DatabaseError) {
                // Manejar la cancelación, si es necesario
            }
        })
    }
```

ideas de proyectos

1. Aplicación de chat en tiempo real
2. Plataforma de colaboración
3. Aplicación de seguimiento de ubicación en tiempo real
4. Aplicación de lista de tareas colaborativa
5. Juego multijugador en tiempo real
6. Plataforma de encuestas en tiempo real
7. Aplicación de seguimiento de inventario en tiempo real
8. Aplicación de seguimiento de eventos en tiempo real
9. Plataforma de seguimiento de actividad física en tiempo real
10. Plataforma de intercambio de recetas en tiempo real

Otras alternativas

1. **Couchbase Server:** Aunque se basa en CouchDB, Couchbase Server ofrece capacidades de tiempo real a través de su funcionalidad de change data capture (CDC) y sus capacidades de clustering.
2. **Kuzzle:** Kuzzle proporciona almacenamiento y sincronización de datos en tiempo real a través de su motor de eventos. Los clientes pueden suscribirse a eventos específicos y recibir actualizaciones en tiempo real cuando ocurren cambios en los datos.
3. **Realtime Database de Firebase (versión Open Source):** Aunque la versión Open Source de Firebase Realtime Database puede no tener todas las características de la versión alojada, aún proporciona capacidades de sincronización en tiempo real que puedes instalar y ejecutar en tu propio servidor.
4. **Supabase:** Si bien Supabase se basa en PostgreSQL, ofrece funcionalidades en tiempo real a través de WebSockets. Esto permite a los clientes suscribirse a cambios en los datos y recibir actualizaciones en tiempo real.

Repositorio del laboratorio

https://github.com/F1852D160/Lab4FirebaseEjemplo [https://github.com/F1852D160/Lab4FirebaseEjemplo]

Práctica de Laboratorio nro. 5

Desarrollar un proyecto Android que se integre con Firebase y permita registrar y listar paises utilizando listview/recyclerview

---