





UNIDAD 5: FIREBASE



Firebase es una plataforma para el desarrollo de aplicaciones web y aplicaciones móviles de Google que permite a los desarrolladores gestionar los datos sin necesidad de administrar la infraestructura: hosting, servidores web, sistemas gestores de bases de datos, almacenamiento, etc.

Compila mejores apps	Improve app quality	Grow your business
 Cloud Firestore <small>BETA</small> Store and sync app data at global scale	 Crashlytics Prioritize and fix issues with powerful, realtime crash reporting	 In-App Messaging Engage active app users with contextual messages
 ML Kit <small>BETA</small> Machine learning for mobile developers	 Supervisión del rendimiento Obtén estadísticas sobre el rendimiento de tu app	 Google Analytics Obtén datos de analítica ilimitados sobre tu app sin cargo
 Cloud Functions Ejecuta código de back-end para dispositivos móviles sin administrar servidores	 Test Lab Prueba la app en dispositivos alojados en Google	 Predictions <small>BETA</small> Define dynamic user groups based on predicted behavior
 Authentication Autentica usuarios de forma simple y segura		 A/B Testing <small>BETA</small> Optimize your app experience through experimentation
 Hosting Entrega recursos de aplicaciones web con velocidad y seguridad		 Cloud Messaging Envía notificaciones y mensajes orientados
 Cloud Storage Almacena y envía archivos a la escala de Google		 Remote Config Modifica tu app sin implementar una versión nueva
 Realtime Database Almacena y sincroniza datos de app en milisegundos		 Dynamic Links Usa vínculos directos con atribución para impulsar el crecimiento
		 App Indexing Dirige el tráfico de búsqueda a tu app

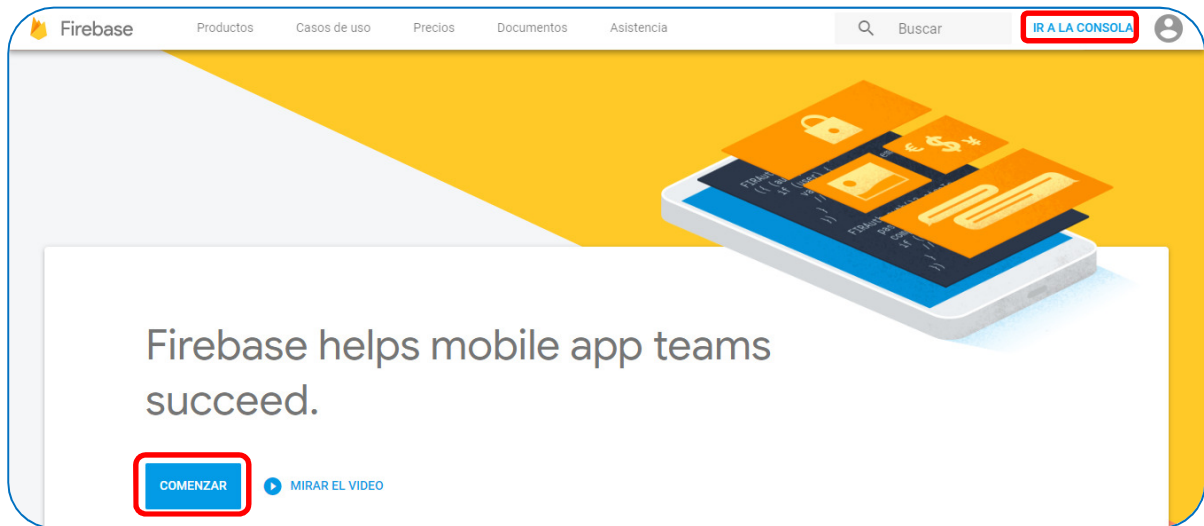
Firebase integra varias herramientas, siendo las más destacadas:

- **Realtime Database:** base de datos JSON (NoSQL) alojada en la nube que permite almacenar y sincronizar datos entre usuarios en tiempo real. Además, utiliza un sistema de caché que permite el uso de datos en dispositivos sin conexión y sincroniza dichos datos de forma automática una vez que el dispositivo tiene conexión.
- **Cloud Storage:** permite almacenar y compartir contenido generado por el usuario como imágenes, audio, video, etc. Los SDK de *Firebase* para el almacenamiento en la nube agregan seguridad de *Google* para las cargas y descargas de archivos desde las aplicaciones, independientemente de la calidad de la red.
- **Authentication:** permite gestionar los usuarios de forma sencilla y segura, ya sea con métodos de autenticación propios como a través del uso directo de cuentas existentes en proveedores de terceros como *Google* o *Facebook*.
- **In-App Messaging:** permite interactuar con usuarios que usan la app activamente mediante mensajes orientados y contextuales que les sugieren completar acciones clave en la app, como finalizar un nivel en un juego, comprar un artículo o suscribirse a contenido.
- **Cloud Firestore:** base de datos de documentos NoSQL que permite almacenar, sincronizar y consultar fácilmente datos para tus apps móviles y web a escala global.

COMENZAR A UTILIZAR FIREBASE

Para empezar a utilizar *Firebase* sólo tenemos que acceder a la web con nuestra cuenta de Google y pulsar el botón “Comenzar” o “Ir a la consola”:

<https://firebase.google.com/>



Desde la consola podremos crear y gestionar nuestros proyectos:



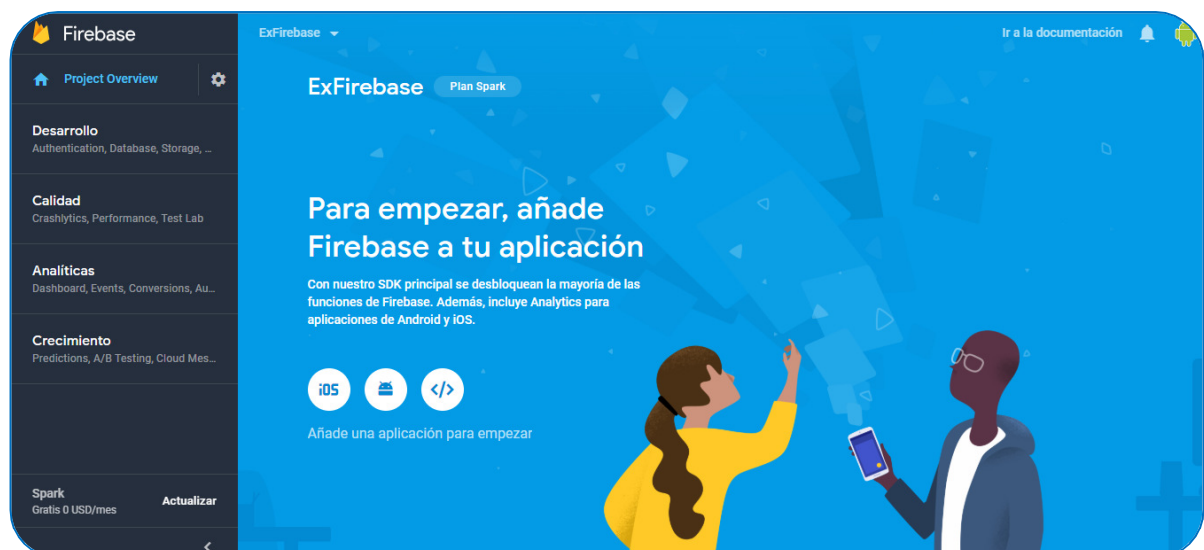
Aunque desde esta consola podemos crear nuevos proyectos, no es necesario hacerlo ya que al vincular nuestra aplicación con *Firebase* nos va a dar la opción de crearlo y lo va a hacer con las opciones requeridas para nuestro tipo de proyecto.

En cualquier caso, si quisiéramos crear un proyecto desde aquí, bastaría con establecer el nombre el proyecto, la ubicación de los servidores y aceptar las condiciones para el tratamiento de datos.

La imagen muestra dos paneles de la consola de Firebase para crear un nuevo proyecto. El panel izquierdo, titulado "Añadir un proyecto", contiene los siguientes campos: "Nombre del proyecto" con un menú desplegable que muestra "ExFirebase"; "ID del proyecto" con un campo de texto que contiene "exfirebase-oht"; "Ubicación de Analytics" con un menú desplegable que muestra "España"; y "Ubicación de Cloud Firestore" con un menú desplegable que muestra "us-central". Hay una nota que dice: "Nota: Los proyectos asocian las aplicaciones de distintas plataformas". El panel derecho contiene una lista de condiciones de uso de Analytics for Firebase, todas marcadas con una casilla de verificación. Al final del panel derecho hay dos botones: "Cancelar" y "Crear proyecto".

Tras unos instantes nuestro proyecto estaría listo para poder ser utilizado.

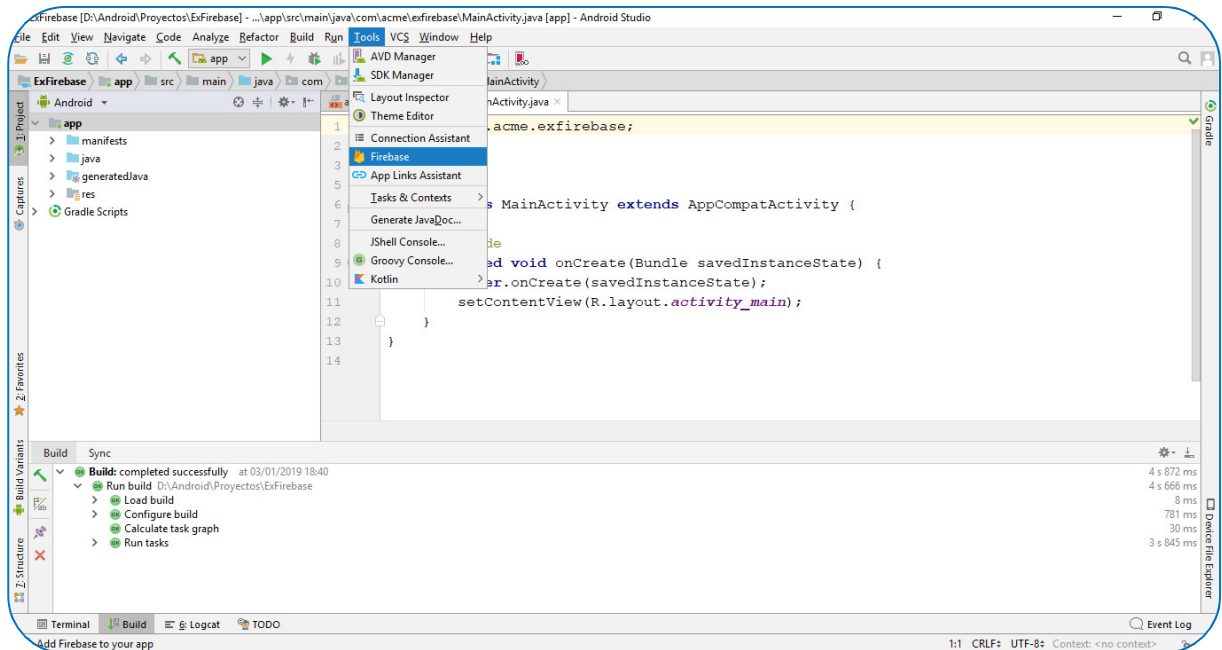
Fíjate en que estamos utilizando un plan denominado “*Spark*” que está orientado a pruebas y no tiene ningún coste.



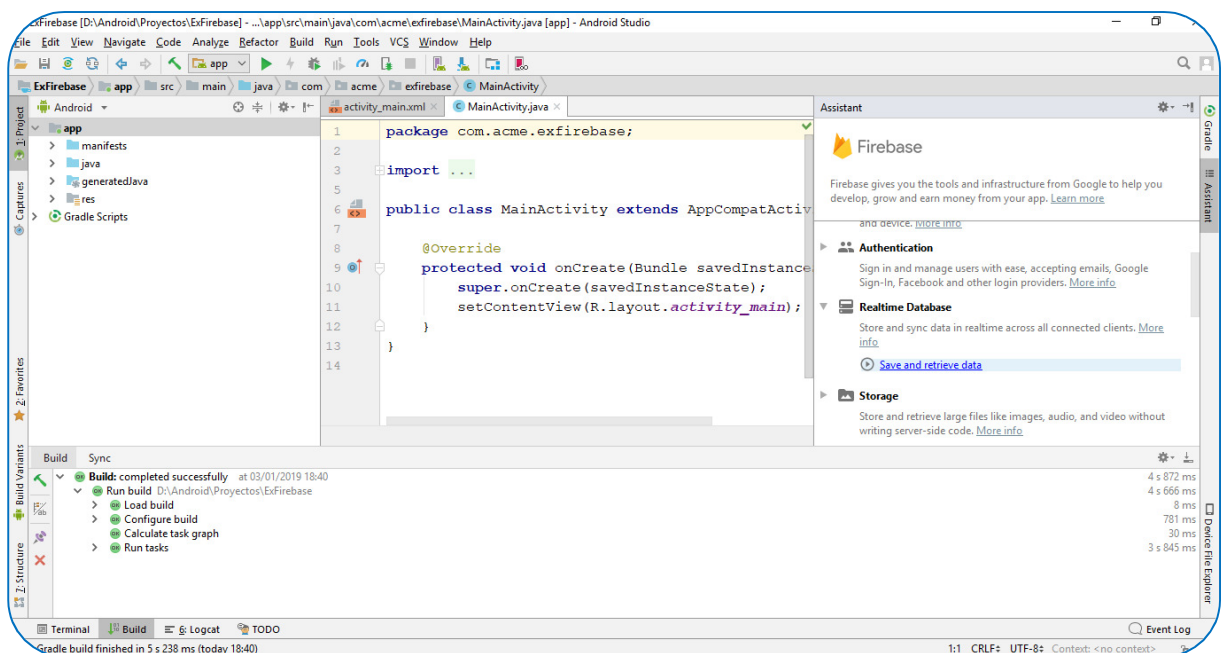
AGREGAR FIREBASE A UN PROYECTO

El siguiente paso será añadir *Firebase* a nuestra aplicación, así que antes de nada deberemos crear un proyecto nuevo en Android Studio, al que denominaremos “PruebaFirebase” y a continuación agregaremos *Firebase* utilizando *Firebase Assistant*.

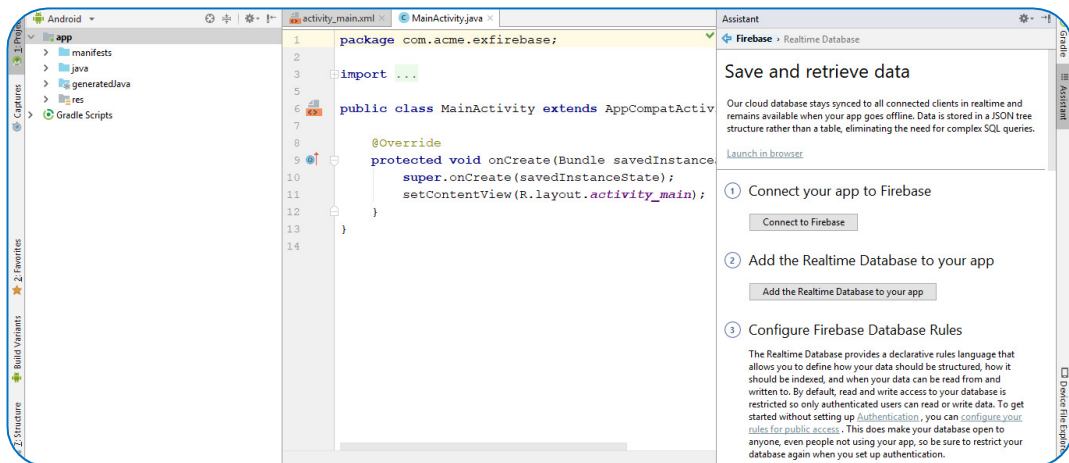
Con el proyecto abierto en Android Studio, seleccionamos “Herramientas” o “Tools”, “*Firebase*”:



Esto lanzará “*Firebase Assistant*”, donde tendremos que seleccionar la herramienta que queremos utilizar, que en este proyecto será “*Realtime Database*”, “*Save and retrieve data*”.



Al seleccionar esta opción accederemos al asistente que nos guiará a lo largo del proceso:

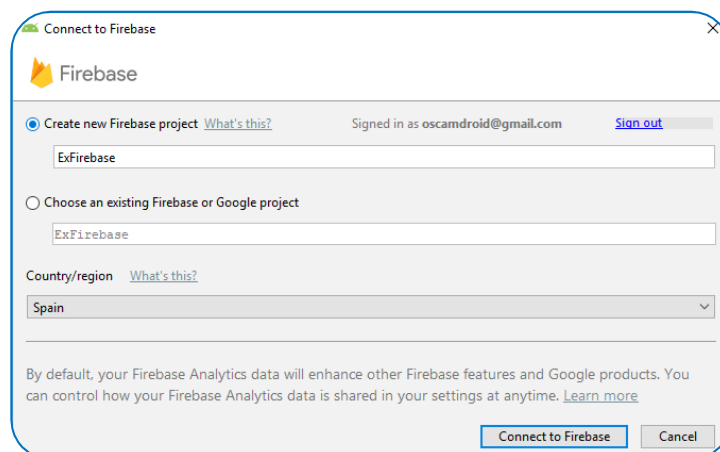


- 1) Conectar con *Firebase*: se abrirá el navegador y tendremos que iniciar sesión con la cuenta de *Google* que hemos utilizado en *Firebase*.

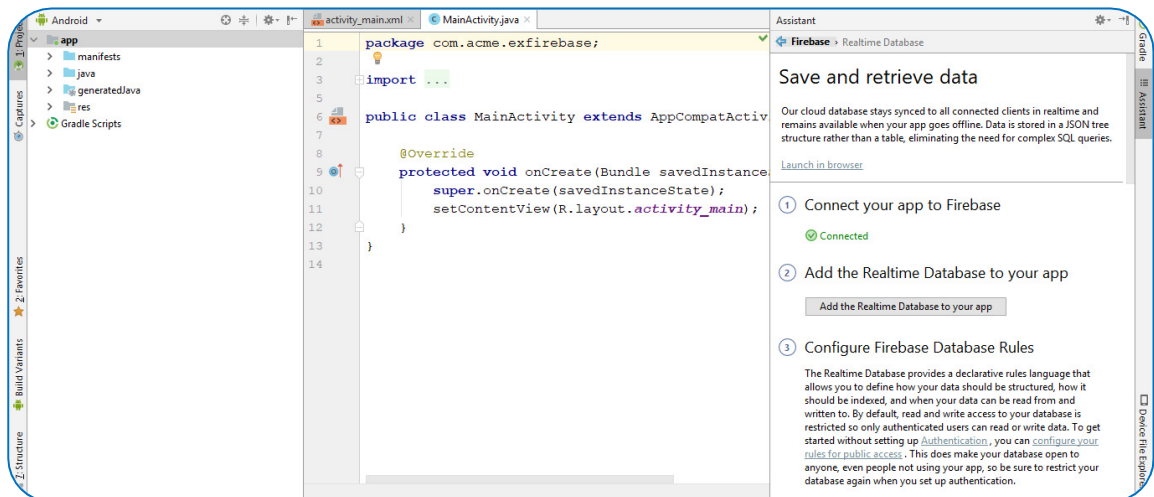


A continuación, asignaremos un proyecto *Firebase* a nuestra aplicación.

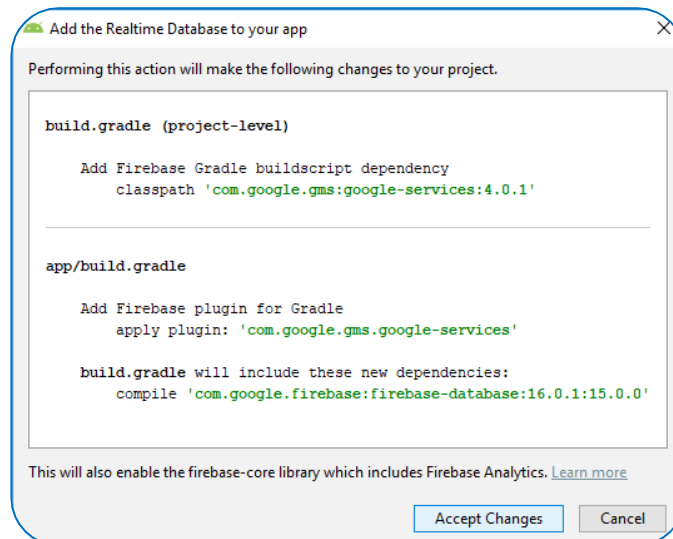
Podemos optar por crear un proyecto nuevo o utilizar uno existente, y en este caso vamos a crear uno nuevo ya que no tenemos ninguno creado previamente.



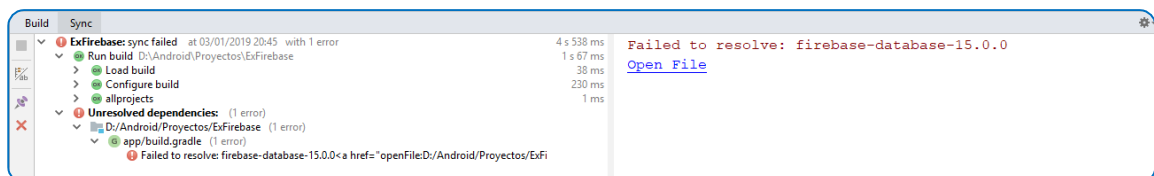
Tras esto, ya tendríamos nuestra app conectada a un proyecto *Firebase*:



- 2) Añadir la base de datos al proyecto: pulsamos el botón y se nos mostrará una nueva ventana indicándonos los cambios que se van a realizar en *gradle*.



Si al sincronizar el proyecto obtenemos un mensaje de error indicando que no se puede resolver una dependencia:



Tendremos que sustituir la línea:

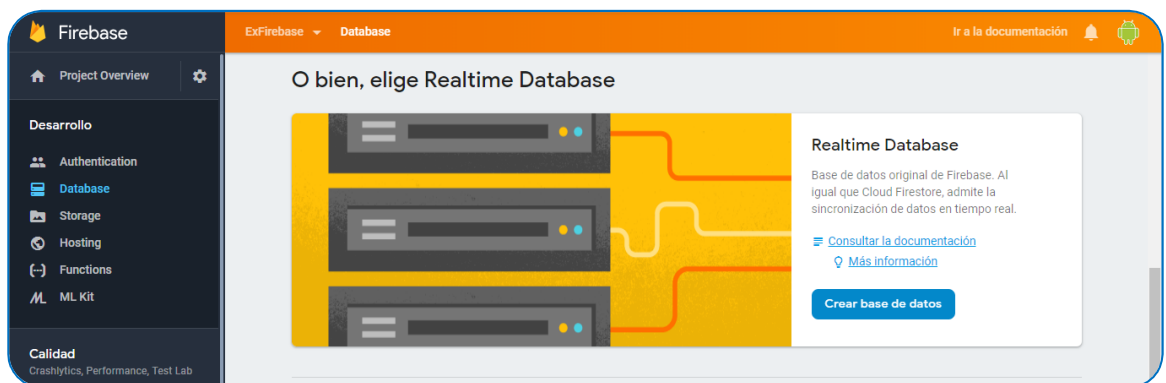
```
implementation 'com.google.firebase:firebase-database:16.0.1:15.0.0'
```

Por las siguientes, y volver a sincronizar:

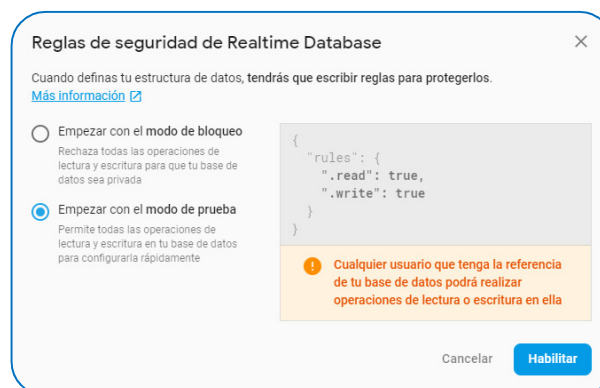
```
implementation 'com.google.firebase:firebase-database:16.0.1'
implementation 'com.google.firebase:firebase-core:16.0.1'
```


- 3) Configurar reglas de acceso: deberemos establecer que usuarios pueden acceder a la base de datos para leer y/o escribir información.

Se puede restringir de manera que los datos sólo puedan ser modificados por usuarios autenticados o permitir que los usuarios de la app (acceso público). Dado que no tenemos configurada la autenticación de usuarios y que queremos realizar pruebas vamos a configurar nuestra BD con acceso público y más adelante ya restringiremos el acceso. Para ello, tendremos que ir a la consola de *Firebase*, seleccionar “*Database*” en el panel de la izquierda, ir a “*Realtime Database*” y pulsar el botón “*Crear base de datos*”.



En la ventana que aparece seleccionamos “Empezar con el modo de prueba”



Y automáticamente se establecerán las reglas de acceso de lectura y escritura permitido para todos los usuarios.



ESTRUCTURA DE LOS DATOS

En “*Realtime Database*” los datos se estructuran en forma de árbol (JSON), y tendremos que tener claro el diseño de dichos datos antes de empezar a almacenar información.

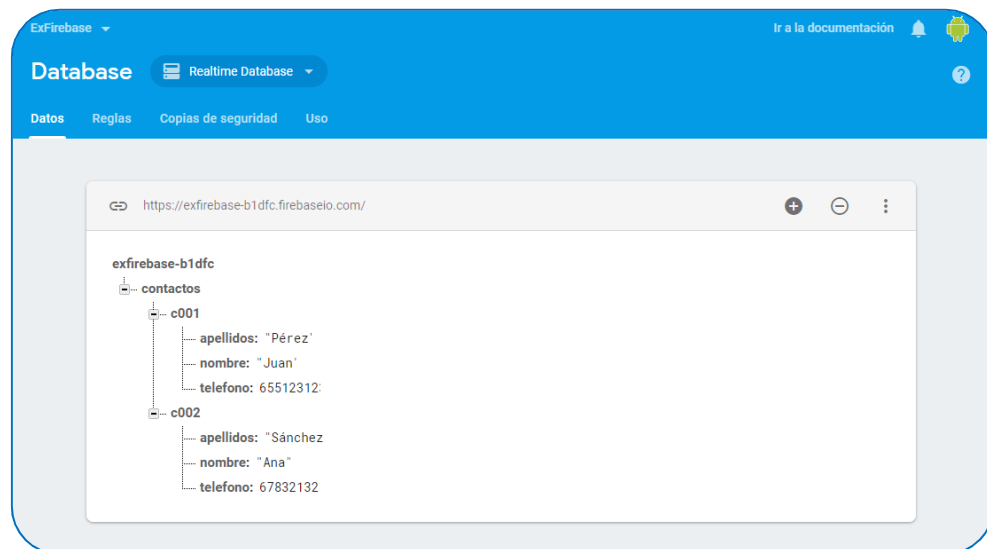
Los siguientes artículos explican los principales conceptos a tener en cuenta en a la hora de estructurar los datos mediante bases de datos jerárquicas:

<https://firebase.google.com/docs/database/android/structure-data>

<https://docs.microsoft.com/es-es/azure/cosmos-db/modeling-data>

Para este ejemplo vamos a partir de un modelo muy sencillo en el que vamos a almacenar datos de los contactos, compuestos por un id, nombre, apellido y teléfono.

La estructura de nuestra base de datos, estará compuesta por un nodo raíz, del que desciende el nodo “contactos”, y a partir de él un nodo por cada contacto con su “id”, y dentro de él el resto de campos del contacto (nombre, apellido y teléfono).



ESCRIBIR DATOS EN LA BASE DE DATOS

Escribir sobre nuestra base de datos es un proceso muy sencillo.

En primer lugar, obtenemos una referencia a la base de datos:

```
FirebaseDatabase fireDB = FirebaseDatabase.getInstance();  
DatabaseReference mDB = fireDB.getReference();
```

O lo que es lo mismo:

```
DatabaseReference mDB = FirebaseDatabase.getInstance().getReference();
```

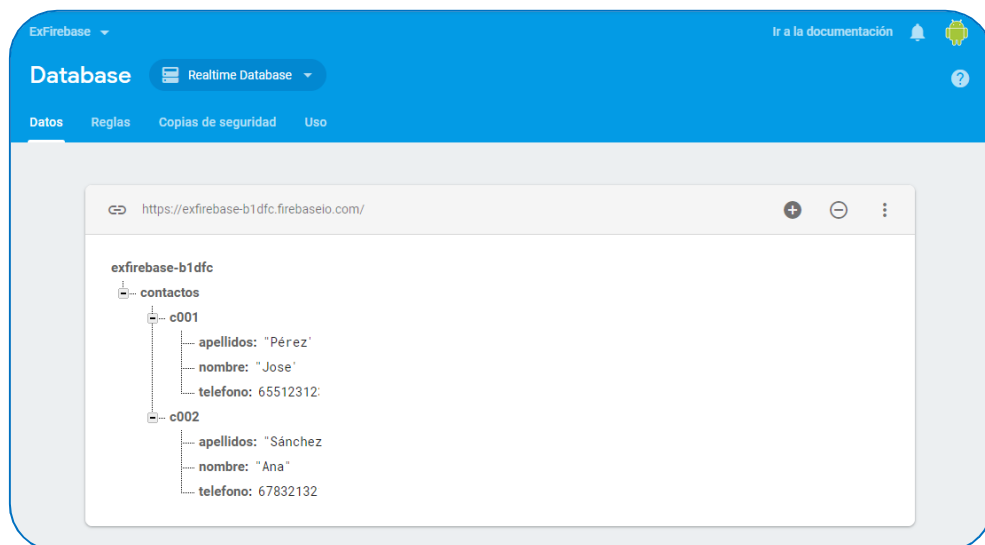
En ambos casos el método *getReference()* nos proporciona una referencia a la raíz del árbol, pero podemos establecer la ruta del nodo concreto que queremos obtener:

```
DatabaseReference mDB = fireDB.getReference("/contactos/c003");
```

A partir de la referencia obtenida podemos navegar por la estructura de árbol mediante el método *child* pasándole como valor el nombre del nodo hijo al que queremos desplazarnos, y una vez localizado el nodo que queremos modificar establecemos su valor mediante *setValue*.

```
mDB.child("contactos").child("c001").child("nombre").setValue("Jose");
```

Si vamos a la base de datos (actualizar página), veremos que el valor del nombre del contacto “c001” se ha modificado y en lugar de “Juan” ahora el valor es “Jose”:



Si queremos eliminar un nodo, el proceso para seleccionar el nodo es idéntico, y una vez seleccionado utilizaremos el método *removeValue()*.

```
FirebaseDatabase fireDB = FirebaseDatabase.getInstance();  
DatabaseReference mDB = fireDB.getReference("/contactos/c003");  
mDB.removeValue();
```

Escribir varios campos de forma simultánea

Para poder escribir en una sola operación toda la información de un “contacto” basta con definir una clase “Contacto” que además de los atributos necesarios contenga un constructor vacío.

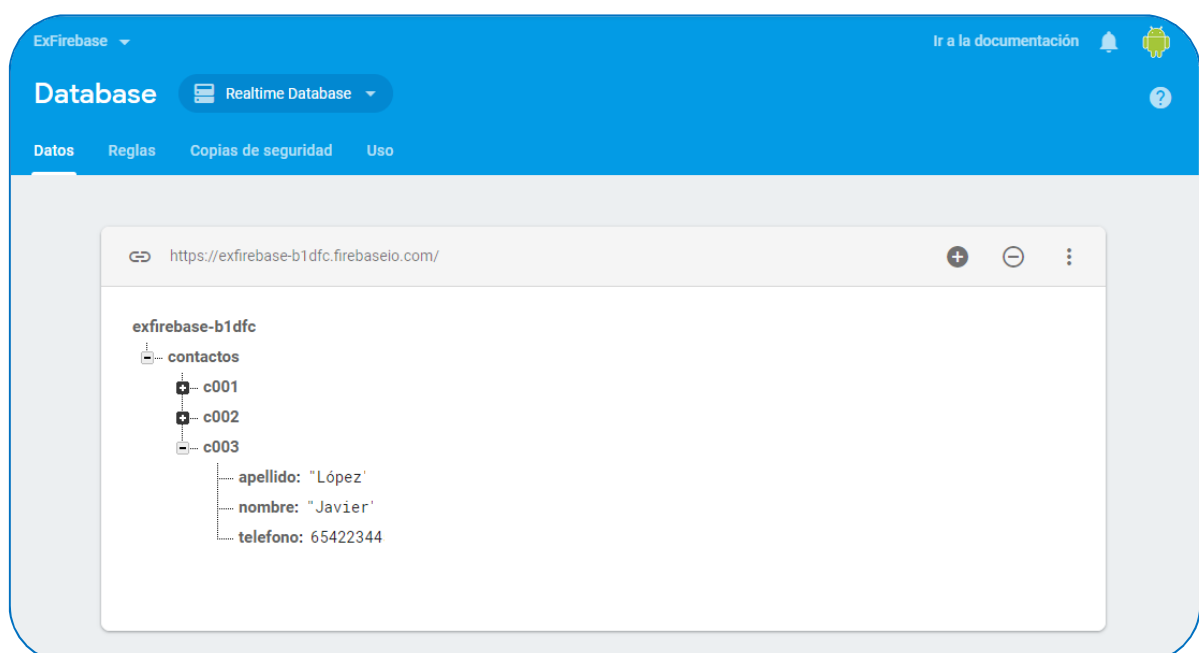
La clase “Contacto” podría quedar así:

```
public class Contacto {  
    public String nombre;  
    public String apellido;  
    public long telefono;  
  
    public Contacto() {  
        // Constructor requerido  
    }  
  
    public Contacto(String nom, String ape, long tlf) {  
        this.nombre = nom;  
        this.apellido = ape;  
        this.telefono = tlf;  
    }  
}
```

Una vez definida la clase, creamos un objeto de tipo “Contacto” con los valores a almacenar, navegamos hasta la posición de su “id” y le pasamos el objeto a *setValue*.

```
Contacto contacto = new Contacto("Javier", "López", 654223445);  
DatabaseReference mDB = FirebaseDatabase.getInstance().getReference();  
mDB.child("contactos").child("c003").setValue(contacto);
```

Y ya tendríamos almacenado el nuevo contacto en nuestra BD:



En los ejemplos anteriores hemos creado el id de nuestros registros de forma manual estableciendo los valores “c001”, “c002”, etc. Pero también podemos dejar que sea *Firebase* quien cree los id de forma automática haciendo uso de la función *push()*.

```
Contacto contacto = new Contacto("Marta", "Hernández", 661224335);
FirebaseDatabase fireDB = FirebaseDatabase.getInstance();
DatabaseReference mDB = fireDB.getReference("contactos");
mDB.push().setValue(contacto);
```



El método *push()* genera una clave única basada en una marca de tiempo, lo que permite que diversos usuarios puedan trabajar sobre los mismos datos sin que se generen claves duplicadas.

Además, al tratarse de una clave basada en una marca de tiempo, los elementos se organizan de forma cronológica.

Podemos obtener el valor de la clave generada utilizando el método *getKey()* sobre la referencia obtenida a la base de datos tras realizar el *push()*.

```
Contacto contacto = new Contacto("Manuel", "Beltrán", 612443890);
FirebaseDatabase fireDB = FirebaseDatabase.getInstance();
DatabaseReference mDB = fireDB.getReference("contactos").push();
mDB.setValue(contacto);
String contactoId = mDB.getKey();
```

Gestionar los errores de escritura

Cuando escribimos o modificamos datos en una base de datos es muy importante notificar al usuario si la operación se ha realizado con éxito o si se ha producido algún error.

Para comprobar los posibles errores hemos de crear un objeto *CompletionListener* que se encargue de gestionar los errores.

```
DatabaseReference.CompletionListener cl =
    new DatabaseReference.CompletionListener() {
        @Override
        public void onComplete(DatabaseError dbError, DatabaseReference dbRef){
            if (dbError != null) {
                // Error de escritura
                int dur = Toast.LENGTH_SHORT;
                String mensa = dbError.getMessage();
                Toast.makeText(MainActivity.this, mensa, dur).show();
            } else {
                // Escritura correcta
                int dur = Toast.LENGTH_SHORT;
                String mensa = "Contacto almacenado";
                Toast.makeText(MainActivity.this, mensa, dur).show();
            }
        }
    };
```

Y en la llamada al método *setValue*, además del dato a escribir incluiremos el objeto *CompletionListener*:

```
Contacto contacto = new Contacto("María", "Gómez", 656112233);
DatabaseReference mDB = FirebaseDatabase.getInstance().getReference();
mDB.child("contactos").child("c004").setValue(contacto, cl);
```

LEER DATOS DE LA BASE DE DATOS

El proceso de lectura de datos desde *Firebase Realtime Database* es mucho más complejo que su escritura y difiere mucho de la lectura de datos de otras bases de datos.

En *Realtime Database* no hay ningún mecanismo para leer el valor asignado a un nodo concreto, sino que se utiliza un *listener* de tipo *ValueEventListener* asociado a la base de datos que detectará los cambios producidos en el nodo referenciado y sus descendientes.

Existen dos métodos para añadir el *listener* a la base de datos:

- *addValueEventListener()*: se notificarán todos los cambios que se produzcan en el nodo especificado y sus descendientes.
- *addListenerForSingleValueEvent()*: se notificará únicamente el primer cambio que se produzca y después se eliminará el *listener*.

Tanto uno como otro, son llamados una vez al ser añadidos, además de cuando se produzcan cambios en la base de datos, de manera que esa primera llamada puede ser utilizada para leer dos datos actuales (antes de que sean modificados).

Por tanto, el proceso consistirá en crear un *ValueEventListener* que implemente los métodos *onDataChange* y *onCancelled*, y posteriormente asociar ese *listener* a la referencia del nodo que queremos (*addListenerForSingleValueEvent* o *addValueEventListener*).

```
ValueEventListener changeListener = new ValueEventListener() {  
    @Override  
    public void onDataChange(DataSnapshot dataSnapshot) {  
        // Se han producido cambios  
    }  
    @Override  
    public void onCancelled(DatabaseError databaseError) {  
        // Se ha producido algún error  
    }  
};  
  
DatabaseReference mDB = FirebaseDatabase.getInstance().getReference();  
mDB.addValueEventListener(changeListener);
```

Cuando dejemos de necesitar el *listener* lo podemos eliminar:

```
mDB.removeEventListener(changeListener);
```

Trabajar con DataSnapshot

El método *onDataChange* recibe un objeto *DataSnapshot* a partir del cual podemos acceder a los datos del subárbol que tengamos referenciado a través de diversos métodos:

- *child* ("ruta"): navega por la ruta indicada a partir del nodo.
- *hasChild* ("nodo"): comprueba si tiene un hijo cuyo nombre sea "nodo".
- *hasChildren*(): comprueba si tiene hijos.
- *getRef*(): devuelve la ruta del nodo referenciado.

```
String path = dataSnapshot.getRef().toString();  
"https://exfirebase-bldfc.firebaseio.com"
```
- *getChildrenCount*(): devuelve el número de hijos del nodo seleccionado.
- *getChildren*(): devuelve una lista de los nodos hijos sobre la que podemos iterar.
- *getKey*(): devuelve el nombre (*key*) del nodo seleccionado.
- *getValue*(*class*): recibe la clase del contenido seleccionado (*Integer*, *String*, etc.) y devuelve el contenido del nodo seleccionado.

En nuestro ejemplo, si la referencia se realizó a la raíz de la base de datos y queremos obtener un contacto concreto, deberemos navegar a partir del objeto *dataSnapshot* hasta dicho contacto mediante *child* y obtener el objeto mediante *getValue*.

```
Contacto c = dataSnapshot.child("contactos/c003").getValue(Contacto.class);
```

Si en lugar de acceder al contacto completo, quisiéramos acceder al nombre:

```
String nom =  
    dataSnapshot.child("contactos/c003/nombre").getValue(String.class);
```

También podemos procesar todos los nodos hijos de un determinado nodo utilizando el método *getChildren* dentro de un bucle *for*:

```
for (DataSnapshot contacto: dataSnapshot.child("contactos").getChildren()) {  
    Contacto c = contacto.getValue(Contacto.class);  
    String texto = c.nombre + " " + c.apellido;  
    Toast.makeText(MainActivity.this, texto, Toast.LENGTH_SHORT).show();  
}
```