**CENTRO DE ESTUDIOS ESPECIALIZADOS CESDE**

**TECNICO EN DESARROLLO DE SOFTWARE**

**PROGRAMACION DE MOVILES II**

**FIREBASE**

**Profesor: Freddy Moscoso Ceballos**

DEFINICIONES

Firebase

Es una plataforma digital desarrollada por Google, su principal función es facilitar la creación de aplicaciones web o apps (móviles) de una manera rápida, con calidad y un mejor rendimiento. Se encuentra en la nube y se puede usar en diferentes plataformas como iOS, Android o web.

Bases de datos en tiempo real (RealTime database)

Realtime Database (bases de datos en tiempo real) es un servicio que permite almacenar y sincronizar datos con una base de datos NoSQL(no relacional) alojada en la nube. Los datos están sincronizados con los clientes en tiempo real y seguirá estando disponible cuando la app no esté conectada.

Se usa para plataformas iOS y Android, principalmente, en formato JSON utilizando SDK de JavaScript y están sincronizadas con los usuarios en tiempo real lo que permite actualizaciones en forma inmediata.

Sus características principales son:

* Base de datos con alojamiento en la nube, lo que libera al usuario de servidores y permite su uso desde cualquier lugar
* Sincronización en tiempo real (Ya que no utiliza peticiones tipo HTTP) Las actualizaciones se reflejan inmediatamente permitiendo una información más veraz.
* Permite apps multiplataforma iOS, Android y Web (JavaScript). Todos los clientes pueden compartir la misma base de datos y recibir actualizaciones de forma automática con los datos más nuevos independientemente de su plataforma. Además, disponemos de un API REST para acceder a la base de datos desde otras plataformas.
* Si se pierde la conexión a Internet la aplicación sigue funcionando, trabajando desde la memoria caché (Cuando se recupere la conexión, la información será actualizada).
* Escalabilidad. Solución especialmente indicada para bases de datos de gran tamaño (bigdata).
* Accesible desde dispositivos del cliente: Se puede acceder de forma directa a la base de datos desde un dispositivo móvil o desde el navegador web, no se necesita un servidor de apps. La seguridad y la validación de datos están disponibles a través de las Security Rules de Firebase Realtime Database, reglas basadas en expresiones que se ejecutan cuando se leen o se escriben los datos.

¿Cómo funciona?:

La Firebase Realtime Database permite crear apps ricas y colaborativas permitiendo un acceso seguro a la base de datos de forma directa desde el código del cliente. Los datos persisten de manera local, e incluso cuando no hay conexión, los eventos en tiempo real se siguen activando, brindándole al usuario final una experiencia receptiva. Cuando el dispositivo recupera la conexión, la Realtime Database sincroniza las modificaciones locales de datos con las actualizaciones remotas que ocurrieron mientras el cliente no tenía conexión, fusionando cualquier conflicto de forma automática.

La Realtime Database proporciona un lenguaje de reglas flexibles basadas en expresiones llamado Security Rules de Firebase Realtime Database , para definir el modo en que tus datos se deben estructurar y el momento en que se pueden someter a lectura y escritura. Cuando se integra con Firebase Authentication, los desarrolladores pueden definir quién accede a qué datos y cómo pueden acceder al mismo.

La Realtime Database es una base de datos NoSQL y como tal tiene diferentes optimizaciones y funcionalidad en comparación con una base de datos relacional. La API Realtime Database está diseñada para permitir operaciones que se pueden ejecutar de forma rápida. Esto te permite crear una gran experiencia en tiempo real que le puede servir a millones de usuarios sin comprometerse con la capacidad de respuesta. Debido a esto, es importante pensar en el modo en que los usuarios necesitan acceder a tus datos y luego estructurarlos de forma adecuada.

Bases de datos no relacionales (NoSQL)

Las bases de datos NoSQL (Not Only SQL) se diseñan para modelos de datos específicos y vienen a suplir a las bases de datos tradicionales (relacionales) en el manejo de grandes cantidades de información que se están manejando desde la llegada de la web 2.0.

Las bases de datos relacionales están diseñadas para aplicaciones de procesamiento de transacciones online (OLTP) altamente coherentes y transaccionales, y son buenas para el procesamiento analítico online (OLAP).

El modelo relacional normaliza los datos en tablas conformadas por filas y columnas. Un esquema define estrictamente las tablas, las filas, las columnas, los índices, las relaciones entre las tablas y otros elementos de las bases de datos. La base de datos impone la integridad referencial en las relaciones entre tablas.

Las bases de datos NoSQL proporcionan una variedad de modelos de datos, como clave-valor, documentos y gráficos, que están optimizados para el rendimiento y la escala.

¿Cómo funciona una base de datos NoSQL (no relacional)?

Las bases de datos NoSQL utilizan una variedad de modelos de datos para acceder y administrar datos. Estos tipos de bases de datos están optimizados específicamente para aplicaciones que requieren grandes volúmenes de datos, baja latencia y modelos de datos flexibles, lo que se logra mediante la flexibilización de algunas de las restricciones de coherencia de datos en otras bases de datos.

En una base de datos relacional, un registro de libros a menudo se enmascara (o "normaliza") y se almacena en tablas separadas, y las relaciones se definen mediante restricciones de claves primarias y externas. En este ejemplo, la tabla Libros tiene las columnas ISBN, Título del libro y Número de edición, la tabla Autores tiene las columnas IDAutor y Nombre de autor y, finalmente, la tabla Autor-ISBN tiene las columnas IDAutor e ISBN. El modelo relacional está diseñado para permitir que la base de datos aplique la integridad referencial entre tablas en la base de datos, normalizada para reducir la redundancia y, generalmente, está optimizada para el almacenamiento.

En una base de datos NoSQL, el registro de un libro generalmente se almacena como un documento JSON. Para cada libro, el elemento, ISBN, Título del libro, Número de edición, Nombre autor y IDAutor se almacenan como atributos en un solo documento. En este modelo, los datos están optimizados para un desarrollo intuitivo y escalabilidad horizontal.

¿Por qué debería usar una base de datos NoSQL?

* Las bases de datos NoSQL se adaptan perfectamente a muchas aplicaciones modernas, como dispositivos móviles, web y juegos, que requieren bases de datos flexibles, escalables, de alto rendimiento y altamente funcionales para proporcionar excelentes experiencias de usuario.
* Flexibilidad: las bases de datos NoSQL generalmente ofrecen esquemas flexibles que permiten un desarrollo más rápido y más iterativo. El modelo de datos flexible hace que las bases de datos NoSQL sean ideales para datos semiestructurados y no estructurados.
* Escalabilidad: las bases de datos NoSQL generalmente están diseñadas para escalar usando clústeres distribuidos de hardware en lugar de escalar añadiendo servidores caros y sólidos. Algunos proveedores de la nube manejan estas operaciones en segundo plano, como un servicio completamente administrado.
* Alto rendimiento: la base de datos NoSQL está optimizada para modelos de datos específicos y patrones de acceso que permiten un mayor rendimiento que el intento de lograr una funcionalidad similar con bases de datos relacionales.
* Altamente funcional: las bases de datos NoSQL proporcionan API altamente funcionales y tipos de datos que están diseñados específicamente para cada uno de sus respectivos modelos de datos.

JSON

JSON (JavaScript Object Notation, «notación de objeto de JavaScript») es un formato de texto sencillo para el intercambio de datos. Se trata de un subconjunto de la notación literal de objetos de JavaScript, Una de las ventajas de JSON sobre XML como formato de intercambio de datos es que resulta mucho más sencillo y es reconocido casi por cualquier navegador

JSON define seis tipos de valores: cadenas, números, booleanos, nulo, matrices y objetos y una de las características de JSON, y es independiente de cualquier lenguaje de programación (la fuente y el emisor pueden estar en lenguajes diferentes)

Los modelos de programación más utilizados para tratar con JSON son:

Modelo de objeto. - El JSON completo es almacenado en memoria en un formato de árbol. Este árbol es navegado, analizado y modificado con las API apropiadas. Como lo carga todo en memoria y luego lo procesa este modelo consume muchos recursos. Sin embargo, es muy flexible para manipular el contenido. Este modelo es permitido por ejemplo en Java por la JSR 353 y por la biblioteca Jackson.

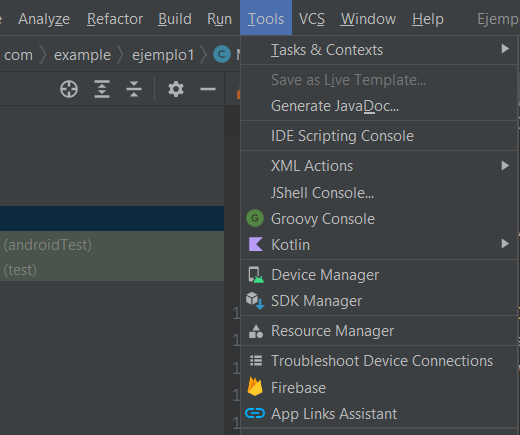
Modelo de flujo: Los datos son leídos o escritos en bloques. Por ejemplo, cada vez que se lee un bloque, el analizador genera eventos apropiados para indicar el tipo de bloque de que se trata. El cliente puede procesar el contenido escuchando los eventos apropiados. Además, es el cliente el que decide cómo se va leyendo el JSON permitiendo parar o saltar contenidos en mitad del proceso. El proceso de escritura tiene propiedades análogas.

Convirtiendo los objetos JSON en objetos del lenguaje. En Java esto es realizado por ejemplo por las bibliotecas Jackson y Gson.

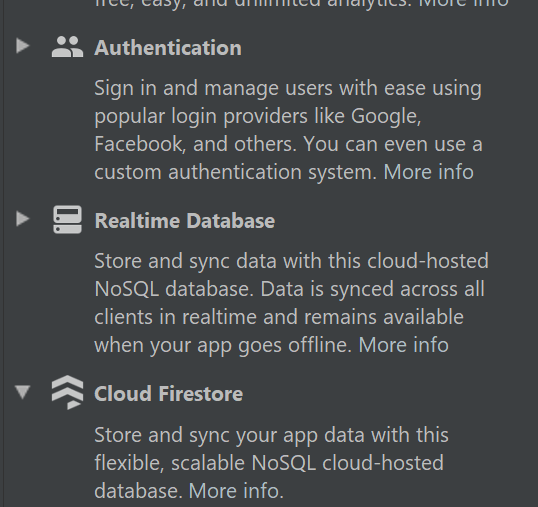
¿Cómo crear una base de datos en FireBase desde Android Studio?

FireBase desde Android Studio

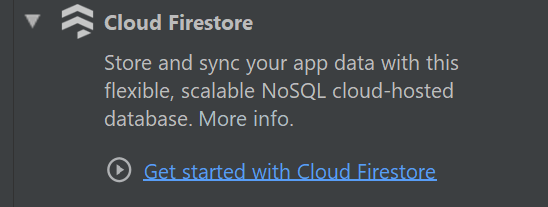
Ingresar al Android Studio y crear una app (no puede ser una sdk menor a 28), ir a la opción herramientas (Tools) y seleccionar Firebase



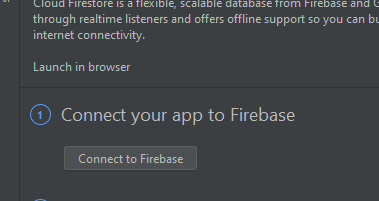
En la parte derecha se abre una ventana con las diferentes opciones del Firebase y vamos a seleccionar Cloud Firestore o solo Firestone



Click en Get started with Cloud Firestore

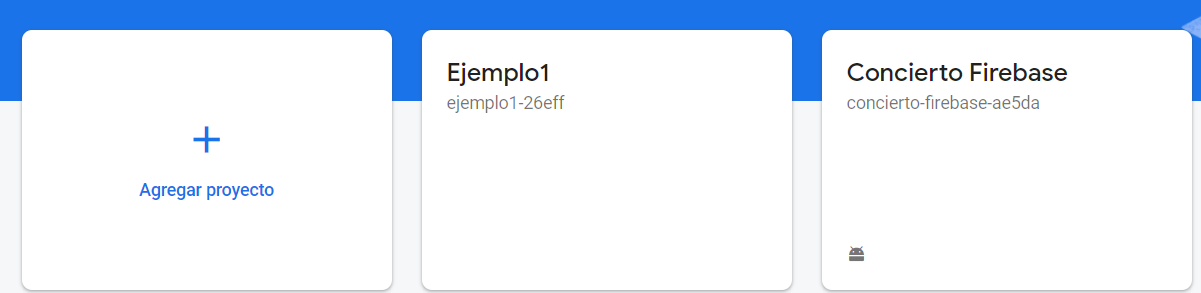


Click en Connect to FireBase

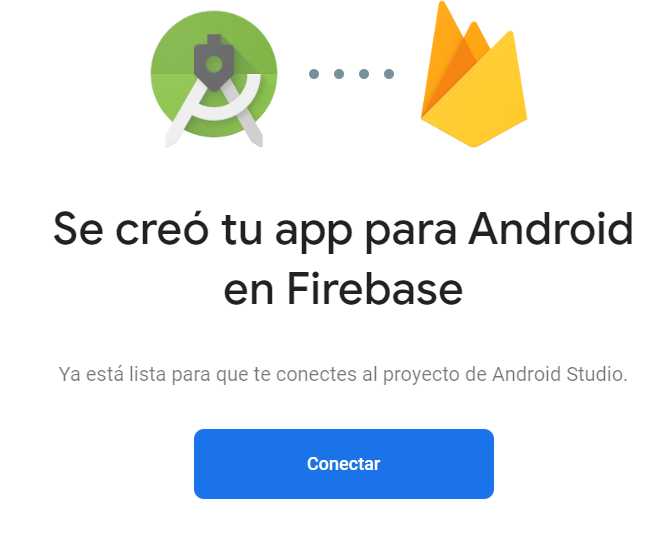


Nota: Si no tiene conectado el Firebase de Android Studio con un correo de Google él lo pedirá.

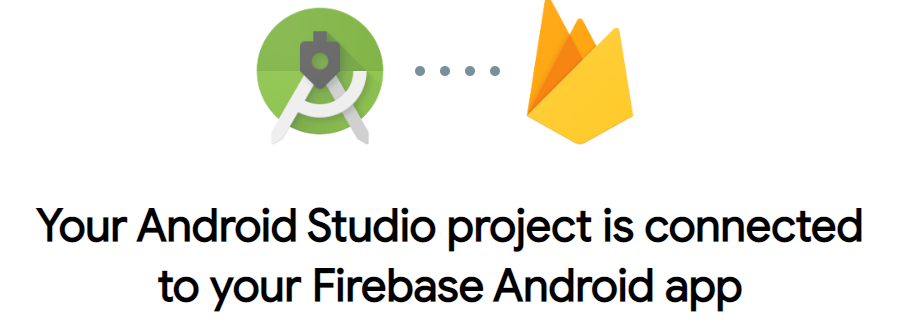
Si todo es correcto lo conectara con FireBase donde puede elegir un proyecto nuevo o uno ya existente



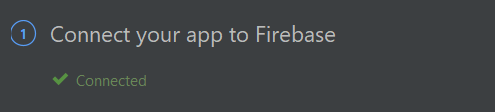
Y click en conectar



Sale el mensaje de que Firebase está conectado con Android Studio

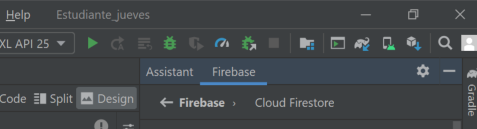


Y en Android Studio Aparece que ya hay conexión

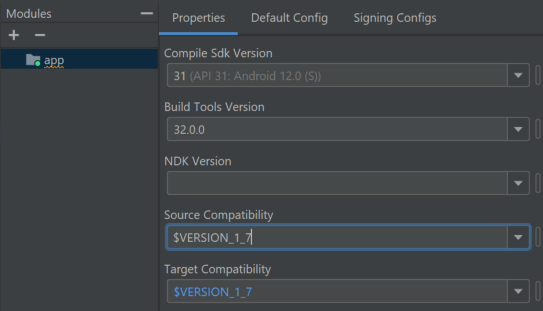


Nota: Si no se conecta seguir los siguientes pasos (diferencia de versiones):

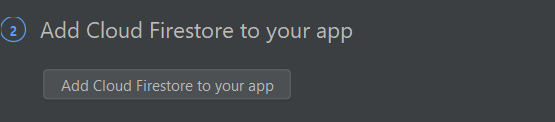
* Click en el icono Project Structure



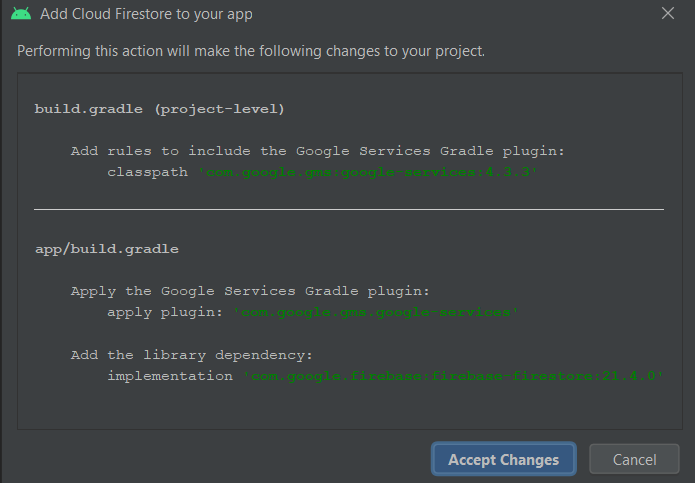
* Click en Properties y seleccionar las siguientes opciones



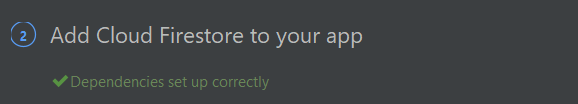
Seguir con el paso dos



Y Aceptar cambios

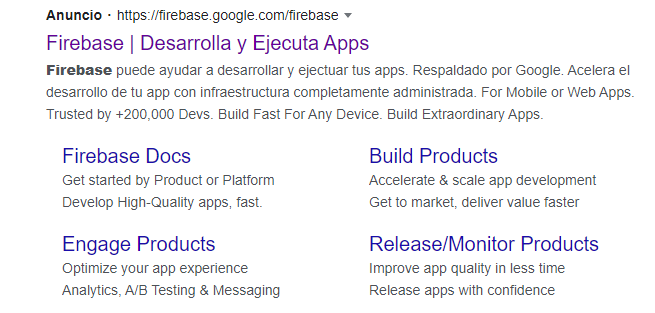


Aparece que las dependencias ya fueron conectadas

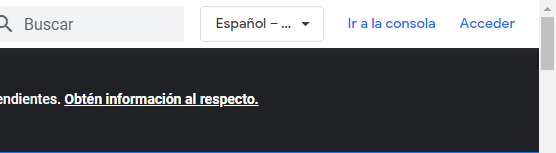


Se construye la App

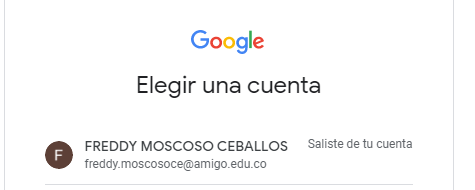
Para verificar la conexión se debe ingresar a un navegador digitamos Firebase



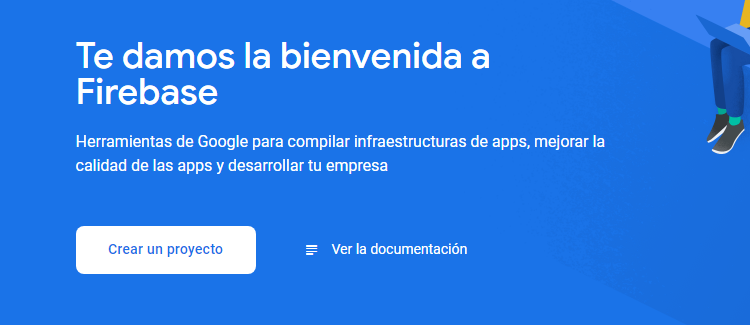
En la página que sale se selecciona la opción ir a la consola (parte superior derecha)



Elegir una cuenta de google



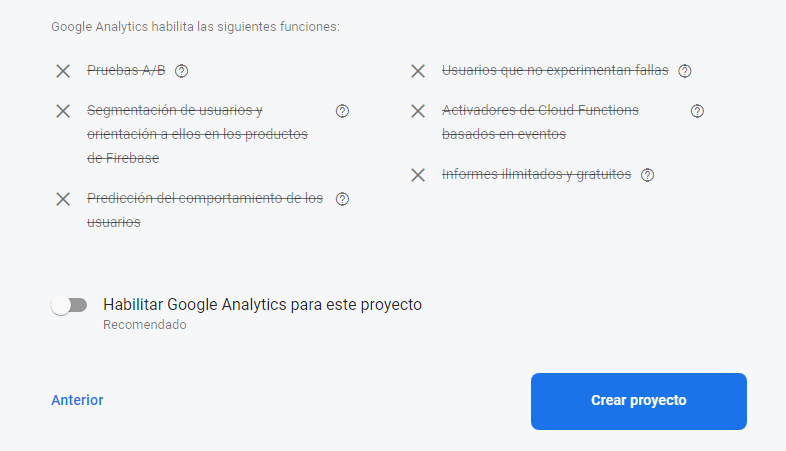
Si es la primera vez, no aparecen proyectos t debemos ir por crear un proyecto



Damos un nombre al proyecto y continuar

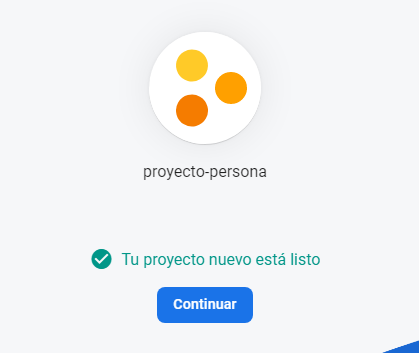


Nota: Se puede, o no, activar la herramienta de Google Analytics que es una plataforma online para medir y analizar lo que ocurre en un sitio Web o en una aplicación móvil.

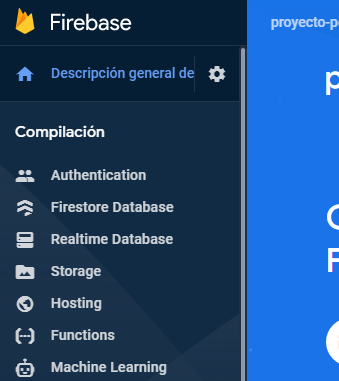


Click en Crear proyecto

Esperar que cree el nuevo proyecto y continuar



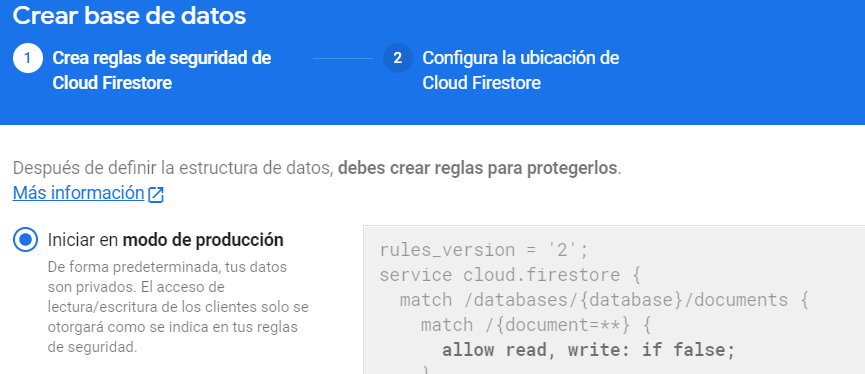
Click en Compilación para mostrar el siguiente menú y luego Firestone Database



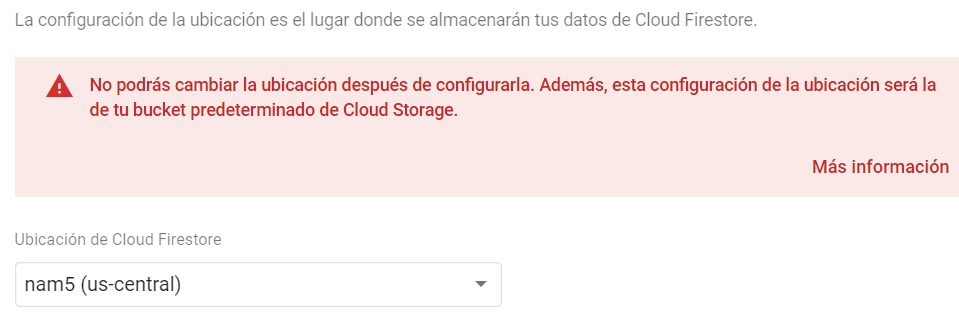
Crear la base de datos



Elegir por seguridad modo de producción (Revisar las reglas)

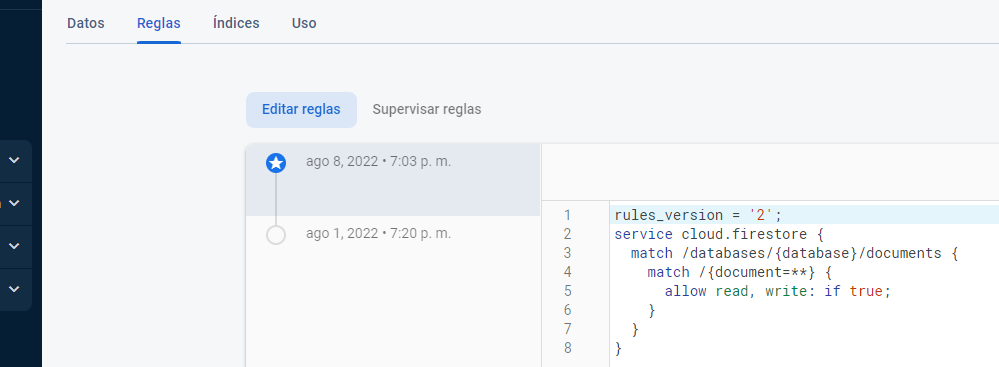


Siguiente y seleccionar la ubicación del lugar de almacenaje (Dejar la que ofrece por defecto)



Click en Habiltar

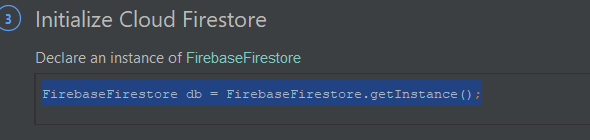
Nota: habilitar las reglas para que me permita la lectura y escritura de documentos en las colecciones (cambiar false por true)



Ahora se deben continuar los pasos desde Android Studio

Se construye la App

Se debe instanciar la clase FireBaseFirestone (A nivel global)



Definir objetos y asociarlos en el método Create

Código Java

public void Adicionar\_datos(){  
  
 identificacion=jetidentificacion.getText().toString();  
 nombre=jetnombre.getText().toString();  
 direccion=jetdireccion.getText().toString();  
 telefono=jettelefono.getText().toString();  
 if (identificacion.isEmpty() || nombre.isEmpty() || direccion.isEmpty() || telefono.isEmpty()){  
 Toast.*makeText*(this, "Todos los datos son requeridos", Toast.*LENGTH\_SHORT*).show();  
 jetidentificacion.requestFocus();  
 }  
 else{  
 // Create a new user with a first and last name  
 Map<String, Object> estudiante = new HashMap<>();  
 estudiante.put("identificacion", identificacion);  
 estudiante.put("nombre", nombre);  
 estudiante.put("direccion", direccion);  
 estudiante.put("telefono", telefono);  
  
// Add a new document with a generated ID  
 db.collection("estudiantes")  
 .add(estudiante)  
 .addOnSuccessListener(new OnSuccessListener<DocumentReference>() {  
 @Override  
 public void onSuccess(DocumentReference documentReference) {  
 // Log.d(TAG, "DocumentSnapshot added with ID: " + documentReference.getId());  
 Limpiar\_datos();  
 Toast.*makeText*(MainActivity.this, "Datos guardados", Toast.*LENGTH\_SHORT*).show();  
 }  
 })  
 .addOnFailureListener(new OnFailureListener() {  
 @Override  
 public void onFailure(@NonNull Exception e) {  
 // Log.w(TAG, "Error adding document", e);  
 Toast.*makeText*(MainActivity.this, "Error guardando datos", Toast.*LENGTH\_SHORT*).show();  
 }  
 });  
 }  
 }  
  
 public void Consultar\_dato(){  
 identificacion=jetidentificacion.getText().toString();  
 if (identificacion.isEmpty()){  
 Toast.*makeText*(this, "La identificacion es requerida", Toast.*LENGTH\_SHORT*).show();  
 jetidentificacion.requestFocus();  
 }  
 else {  
 db.collection("estudiantes")  
 .whereEqualTo("identificacion",identificacion)  
 .get()  
 .addOnCompleteListener(new OnCompleteListener<QuerySnapshot>() {  
 @Override  
 public void onComplete(@NonNull Task<QuerySnapshot> task) {  
 if (task.isSuccessful()) {  
 for (QueryDocumentSnapshot document : task.getResult()) {  
 idestudiante=document.getId();  
 jetnombre.setText(document.getString("nombre"));  
 jetdireccion.setText(document.getString("direccion"));  
 jettelefono.setText(document.getString("telefono"));  
 // Log.d(TAG, document.getId() + " => " + document.getData());  
 }  
 } else {  
 Toast.*makeText*(MainActivity.this, "Error consultando datos", Toast.*LENGTH\_SHORT*).show();  
 // Log.w(TAG, "Error getting documents.", task.getException());  
 }  
 }  
 });  
 }  
 }

public void Modificar\_datos(){  
  
 identificacion=jetidentificacion.getText().toString();  
 nombre=jetnombre.getText().toString();  
 direccion=jetdireccion.getText().toString();  
 telefono=jettelefono.getText().toString();  
 if (identificacion.isEmpty() || nombre.isEmpty() || direccion.isEmpty() || telefono.isEmpty()){  
 Toast.*makeText*(this, "Todos los datos son requeridos", Toast.*LENGTH\_SHORT*).show();  
 jetidentificacion.requestFocus();  
 }  
 else{  
 // Create a new user with a first and last name  
 Map<String, Object> estudiante = new HashMap<>();  
 estudiante.put("identificacion", identificacion);  
 estudiante.put("nombre", nombre);  
 estudiante.put("direccion", direccion);  
 estudiante.put("telefono", telefono);  
  
 db.collection("estudiantes").document(idestudiante)  
 .set(estudiante)  
  
 .addOnSuccessListener(new OnSuccessListener<Void>() {  
 @Override  
 public void onSuccess(Void aVoid) {  
 Toast.*makeText*(MainActivity.this,"Estudiante actualizado correctmente...",Toast.*LENGTH\_SHORT*).show();  
 Limpiar\_datos();  
 }  
 })  
 .addOnFailureListener(new OnFailureListener() {  
 @Override  
 public void onFailure(@NonNull Exception e) {  
 Toast.*makeText*(MainActivity.this,"Error actualizando estudiante...",Toast.*LENGTH\_SHORT*).show();  
 }  
 });  
 }  
}

public void Eliminar\_documento(){  
 identificacion=jetidentificacion.getText().toString();  
 if (identificacion.isEmpty()){  
 Toast.*makeText*(this,"Identificacion es requerida",Toast.*LENGTH\_LONG*).show();  
 jetidentificacion.requestFocus();  
 }  
 else {  
 db.collection("estudiantes").document(idestudiante)  
 .delete()  
 .addOnSuccessListener(new OnSuccessListener<Void>() {  
 @Override  
 public void onSuccess(Void aVoid) {  
 Limpiar\_datos();  
 Toast.*makeText*(MainActivity.this,"Estudiante eliminado correctamente...",Toast.*LENGTH\_SHORT*).show();  
 }  
 })  
 .addOnFailureListener(new OnFailureListener() {  
 @Override  
 public void onFailure(@NonNull Exception e) {  
 Toast.*makeText*(MainActivity.this,"Error eliminando estudiante...",Toast.*LENGTH\_SHORT*).show();  
 }  
 });  
 }  
}

public void Limpiar\_datos(){  
 jettelefono.setText("");  
 jetdireccion.setText("");  
 jetnombre.setText("");  
 jetidentificacion.setText("");  
 jetidentificacion.requestFocus();  
}

}

Binding 🡪 revisar

Bibliografía

* <https://ascenso.org/categoria/actualidad-digital/que-es-firebase-realtime-database/>
* <http://www.androidcurso.com/index.php/recursos/tutoriales/89-firebase/unidad-1-introduccion/695-base-de-datos-en-tiempo-real>
* <https://aws.amazon.com/es/nosql/>
* <https://www.acens.com/wp-content/images/2014/02/bbdd-nosql-wp-acens.pdf>
* <https://es.wikipedia.org/wiki/JSON>
* <https://www.nextu.com/blog/que-es-json/>