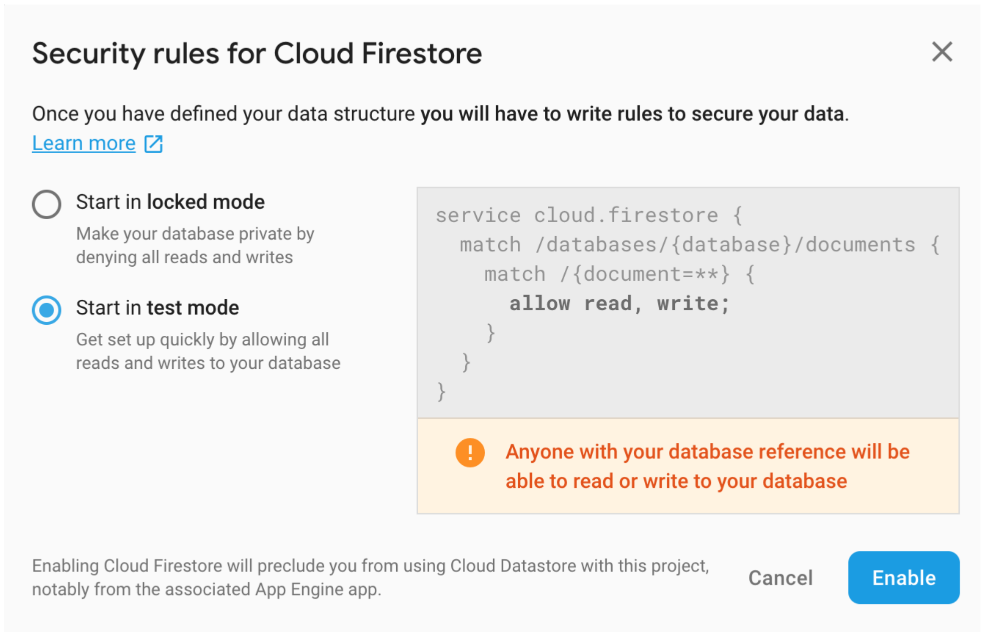
Using Google Firestore & Android Studio

Google Firestore - a cloud-based NoSQL database, which Google positions as a replacement for the Realtime Database. Cloud Firestore allows you to store data on a remote server, easily get access to them and monitor changes in real time.

Creation and connection to the project:

In the Firebase console, select Database and click on Create database. Next, select the access settings.



Security rules for Cloud Firestore

Once you have defined your data structure **you will have to write rules to secure your data**.
Learn more

Start in **locked mode**
Make your database private by denying all reads and writes

[✓ Start in **test mode**]
Get set up quickly by allowing all reads and writes to your database

```
service cloud.firestore {
  match /databases/{database}/documents {
    match /{document=**} {
      allow read, write;
    }
  }
}
```

Anyone with your database reference will be able to read or write to your database

Enabling Cloud Firestore will preclude you from using Cloud Datastore with this project, notably from the associated App Engine app.

To configure the project, we have to make the following steps:

- Add Firebase to the project according to the instructions from here.
- Add dependency to app/build.gradle

```
implementation 'com.google.firebase:firebase-firestore:21.0.0'
```

<u>Data storage structure:</u>

Firestore uses collections and documents to store data. A document is an entry that contains any fields. Documents are combined into collections. Also, the document may contain nested collections, but on Android it is not supported. If we draw an analogy with the SQL database, the collection is a table, and the document is an entry in this table. One collection may contain documents with a different set of fields.

<u>Receive and write data:</u>

In order to get all the documents of a collection, the following code is enough:

```
remoteDB.collection("Tasks")
    .get()
    .addOnSuccessListener { querySnapshot ->
       // Successfully received data. List in querySnapshot.documents
    }
    .addOnFailureListener { exception ->
       // An error occurred while getting data
    }
```

Here we request all documents from the Tasks collection.

The library allows you to create queries with parameters. The following code shows how to get documents from the collection by condition:

```
remoteDB.collection("Tasks")
    .whereEqualTo("title", "Task1")
    .get()
    .addOnSuccessListener { querySnapshot ->
       // Successfully received data. List in querySnapshot.documents
    }
    .addOnFailureListener { exception ->
       // An error occurred while getting data
    }
```

Here we request all documents from the Tasks collection, in which the title field corresponds to the value of Task1.

While getting the documents, they can be immediately converted into our data classes.

```
remoteDB.collection(“Tasks”)
    .get()
    .addOnSuccessListener { querySnapshot ->
       // Successfully received data. List in querySnapshot.documents
      val taskList: List<RemoteTask> =
querySnapshot.toObjects(RemoteTask::class.java)
    }
    .addOnFailureListener { exception ->
       // An error occurred while getting data
    }
```

For writing, you have to create a Hashmap with data (where the name of the field acts as a key, and the value of this field as a value) and transfer it to the library. You can see that in the following code:

```
val taskData = HashMap<String, Any>()
taskData["title"] = task.title
taskData["created"] = Timestamp(task.created.time / 1000, 0)

remoteDB.collection("Tasks")
    .add(taskData)
    .addOnSuccessListener {
        // Successful write
    }
    .addOnFailureListener {
        // There was an error while writing
    }
```

In this example, a new document will be created and the Firestore will generate an id for it. To set your own id you need to do the following:

```
val taskData = HashMap<String, Any>()
taskData["title"] = task.title
taskData["created"] = Timestamp(task.created.time / 1000, 0)

remoteDB.collection("Tasks")
    .document("New task")
    .set(taskData)
    .addOnSuccessListener {
        // Successful write
    }
    .addOnFailureListener {
        // There was an error while writing
    }
```

In this case, if there is no document with id equal to new task, then it will be created, and if it is, then the specified fields will be updated.

Another option to create/update a document.

```
remoteDB.collection("Tasks")
    .document("New task")
    .set(mapToRemoteTask(task))
    .addOnSuccessListener {
        // Successful write
    }
    .addOnFailureListener {
        // There was an error while writing
    }
```

<u>**Loading large amounts of data:**</u>

Realtime Database provides more or less convenient mechanism for loading large amounts of data, which based on manually editing a json file and loading it. Firestore does not provide anything. It was very inconvenient to add new documents until I found a way to download a large amount of information as easily as possible. So that you do not have such issues as me, I will attach the instructions below on how to quickly and easily load a large amount of data. The instruction was found on the Internet.

Install Node.js and npm

Install the firebase-admin package by running the command

npm install firebase-admin — save

3. Generate a json file with collection data. An example can be found in the Tasks.json file.

4. For loading we need an access key. How to get it is well described in this article.

5. In the export.js file fill with your data

require('./firestore_key.json') — file with access key. I put it in the folder with the script

<YOU_DATABASE> — the name of your firestore database

"./json/Tasks.json" — the path to the file in which the data lie [‘created’] — list of field names with type Timestamp

6. Run script

```
node export.js
```

References:

https://proandroiddev.com/cloud-firestore-android-is-easy-c940cb82715c

https://firebase.google.com/docs/firestore/quickstart