

# Lesson 10: Firebase Authentication and Database

<b>Introduction.....</b>	<b>10-1</b>
<b>What is the JSON? .....</b>	<b>10-2</b>
<b>How does Firebase Database work? .....</b>	<b>10-2</b>
<b>Firebase Authentication (Signup and Login Android App).....</b>	<b>10-3</b>
<b>Configure your App to use Firebase Services.....</b>	<b>10-8</b>
<b>Configuring Firebase Authentication.....</b>	<b>10-15</b>
Login to App Using a Firebase User Accounts .....	10-23
Logout Configuration .....	10-24
<b>Using Firebase Assistant with Android Studio.....</b>	<b>10-26</b>
<b>Firebase Database.....</b>	<b>10-27</b>
<b>Real Time Database.....</b>	<b>10-28</b>
<b>Cloud Firestore Database .....</b>	<b>10-37</b>
 <b>Lab 10: Firebase Authentication and Database .....</b>	 <b>10-44</b>
<b>Configure your App to use Firebase Services .....</b>	<b>10-52</b>
<b>Adding Firebase to your Android App .....</b>	<b>10-54</b>
<b>Configuring User Authentication Using Firebase Authentication .....</b>	<b>10-63</b>
<b>Creating a Firebase Cloud Database.....</b>	<b>10-64</b>
<b>Retrieving Data   Firebase Cloud Database .....</b>	<b>10-70</b>

## Introduction

**Firebase** is a NoSQL document database that simplifies storing, syncing, and querying data for your Android, iOS and web apps at a global scale. Its client libraries provide live synchronization and offline support, while its security features and integrations with the Firebase and Google Cloud platforms accelerate building truly server-less apps.

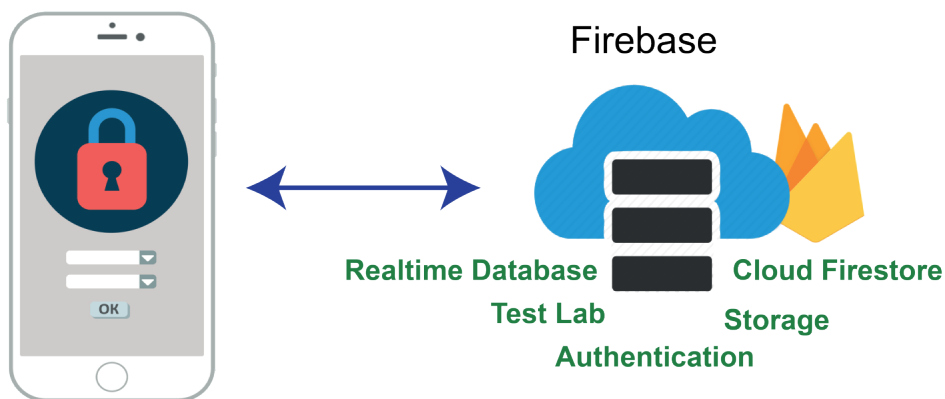
Firebase provides many products which are as follows:

### 1- Authentication

Most apps require to know the identity of the user. Firebase Authentication provides backend services, easy-to-use SDKs and ready-made UI libraries to authenticate users to your app. It supports authentication using passwords, phone numbers, popular federated identity providers like Google, Facebook, Twitter and more. In Firebase Authentication, developers can define who has access to what data and how they can access it. In this lesson, you will learn how to configure your Flutter app by using this feature in order to create user accounts on Firebase and how to login to your app using these accounts.

### 2- Realtime Database

The Firebase Realtime Database is a cloud-hosted database. Data is stored as JSON and is synchronized in real time to every connected client. In this lesson, you will configure your app to use Firebase real time database to save app data and retrieve (query) the existing Firebase data.



### 3- Cloud Firestore

Cloud Firestore is a flexible, scalable database for mobile, web, and server development from Firebase and Google Cloud Platform. Like Firebase Realtime Database, Cloud Firestore keeps your data in sync across client apps through real time listeners and offers offline support for mobile and web applications so you can build responsive apps that work regardless of network latency or Internet connectivity. Cloud Firestore also offers seamless integration with other Firebase and Google Cloud Platform products, including Cloud Functions. In this lesson, you will learn how to create a cloud Firestore database and how to use this data in an Android app.

## 4- Cloud Storage

Cloud Storage for Firebase is a powerful, simple, and cost-effective object storage service built for Google scale. The Firebase SDKs for Cloud Storage adds Google security to file uploads and downloads for your Firebase apps, regardless of network quality. You can use our SDKs to store images, audio, video or other user-generated content.

Firebase provides other services and products which are all important for mobile and web development domains such as cloud messaging, hosting, cloud functions, test lab, ML Kit, analytics, predictions, and others. To get more details about other Firebase products, check the following web site: <https://firebase.google.com/products#develop-products>

The Firebase Data is stored as JSON and is synchronized in real-time to every connected client. Here is some information about JSON.

### What is the JSON?

**JSON** (JavaScript Object Notation) is a lightweight data-interchange format. People can easily learn to read and write JSON. It is based on a subset of the JavaScript Programming Language. JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including Java, JavaScript, Kotlin, Dart and many others. These properties make JSON an ideal data-interchange language.

Google allows web sites, mobile apps and other media to connect to its database and import it in JSON format. For example, Google maps database is available to import or use by iOS, Android, and web apps in JSON format. All that the app needs to establish a connection with Google database is using https connection and Google API key as you will see in the next lesson (Using GPS and Google Maps).

You can customize your import of the Google database by selecting whatever object you want it to appear on your app content.

JSON objects are surrounded by curly braces {} and they are written in key/value pairs.

Keys must be string and values must be a valid JSON data type (string, number, array, Boolean or null).

Later in this lesson, you will learn in details about how to configure JSON in your app in order to access the Firebase database.

### How does Firebase Database work?

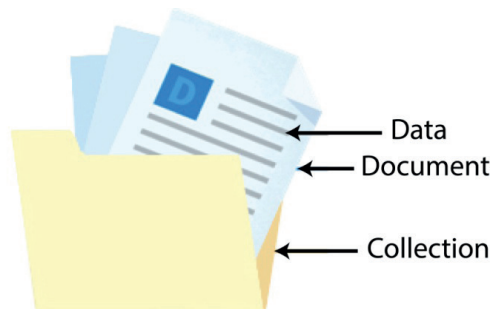
Firebase has many products and services. In this lesson, we will focus on Firebase authentication and database because these services are mostly used with mobile apps.

In the connection between the mobile app and the Firebase database, the Firebase uses data synchronization instead of typical HTTP requests every time data changes. Any connected device receives that update within milliseconds.

Firebase apps remain responsive even when they are offline because the Firebase Database SDK keeps your data on the device storage. When the device regains connection, the Database synchronizes the local data changes with the remote updates that occur while the client was offline and merge any conflicts automatically.

The Firebase Database can be accessed directly from a mobile device or web browser. Therefore, there is no need for an application server. Security and data validation are available through the Firebase Database Security Rules and expression-based rules that are executed when data is read or written.

Unlike an SQL database, in the Firebase database, there are no tables or rows. Instead, you store data in *documents*, which are organized into *collections*. Each document contains a set of key-value pairs (data) as you will see later in this lesson.



The Firebase Database lets you build rich, collaborative applications by allowing secure access to the database directly from the client-side code.

The Firebase Database is a NoSQL database and as such has different optimizations and functionality compared to a relational database. The Firebase Database API is designed to only allow operations that can be executed quickly. This enables you to build great real-time experience that can serve millions of users without compromising on responsiveness. Because of this, it is important to think about how users need to access your app data and then structure it accordingly.

When you work with the Firebase database, you will find that there are two types of Firebase database, **Realtime Firebase database** and **Cloud Firestore Database**. Later in this lesson, you will know in detail about the difference between the Firebase Realtime database and Cloud Firestore.

In this lesson, we are mostly going to focus on the Cloud Firestore database which is the latest way of using Firebase to store data in the cloud. We are also going to be using the authentication package so that you can design an app, which allows app users to register their accounts (Signup) and login to this app using their credentials (User Login).

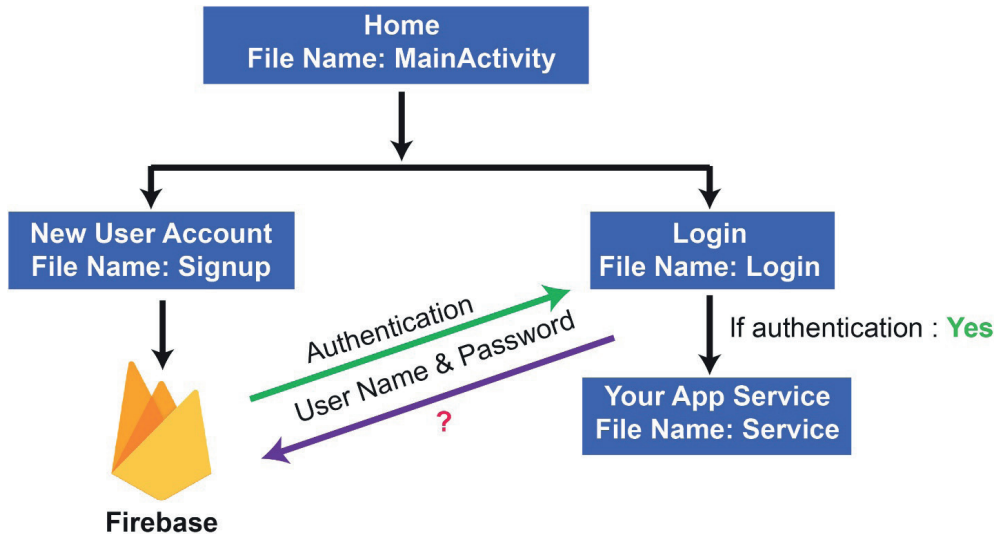
## Firebase authentication (Signup and Login Android App)

In this lesson, you will create a simple Android app using the Firebase authentication package to allow the app users to create a user account (signup) on Firebase and use this account to login to a specific part of this app such as use a specific service.

Most apps need to know the identity of a user. Knowing a user's identity allows an app to securely save his/her data in the cloud and provide the same personalized experience across all of the user's devices.

Firestore Authentication provides backend services, easy-to-use SDKs and ready-made UI libraries to authenticate users to your app. It supports authentication using passwords, phone numbers, popular federated identity providers like Google, Facebook and Twitter and more.

The following figure includes a diagram of the Android app which you will create:



As you see in the previous diagram, you will create a home interface (**MainActivity**) which includes two buttons. The first button is: **New User Account** to create a new user account. This account consists of a user name (email) and password which will be created at the Firestore web server. The second button in the **MainActivity** file will be for user login (**User Login**). When the app user taps the **User Login** button, he/she will be asked to enter his/her user name and password which the app user has created before. Then, when the app user enters his/her user name and password, then taps the **Login** button, his/her credential information (user name & password) will be sent to Firestore to check the authentication. If the app user's credential information is correct, then the user can login or move to another app interface (Service Activity) to use your app services.

To create this Android app, perform the following steps:

- 1- Open Android Studio, and then click **File → New → New Project**
- 2- Select **Empty Activity**, and click **Next**
- 3- Type: **Lesson10\_Authentication** for the application name, then click **Finish**.
- 4- Before you start with typing the Kotlin code in your app, check the **build.gradle (Module: Lesson10\_Authentication.app)** file and be sure it has the Kotlin plugin. If not, add the following code: **id 'kotlin-android-extensions'**

And click the **Sync Now**. The configuration should be as follows:

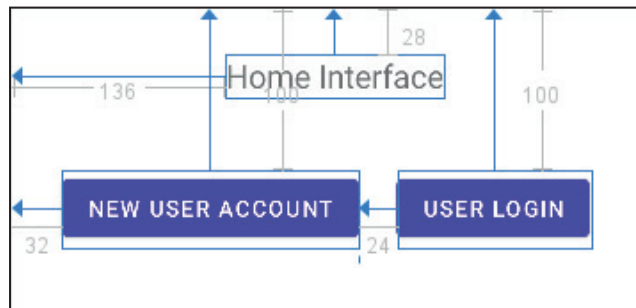
```

1  plugins {
2      id 'com.android.application'
3      id 'kotlin-android'
4      id 'kotlin-android-extensions'
5  }

```

5- Open the **activity\_main.xml** file in Design mode, and then **delete** the “Hello World!” text.

6- You should design the **activity\_main.xml** interface with the following design:



To do this, add a **TextView** widget from the **Palette** panel, set its constraints and set its attributes values as follows:

<b>text:</b> Home Interface	<b>textSize:</b> 20sp
-----------------------------	-----------------------

As illustrated in the previous figure, from the **Palette** panel, add two buttons, set their constraints and set their attributes values as follows:

<b>id:</b> signupBtn	<b>text:</b> New User Account
----------------------	-------------------------------

<b>id:</b> loginBtn	<b>text:</b> User Login
---------------------	-------------------------

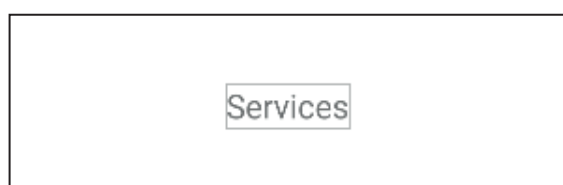
7- Create the **Service** activity. This activity should have the app services which the app user entered using his/her credentials (user name & password) to login to this activity. To do this, right click **layout** → **New** → **Activity** → **Empty Activity**

Type: **Service** for the **Activity Name** and click **Finish**

8- Open the **activity\_service.xml** file in the **Design** mode, add a **TextView** widget, set its constraints and set its attributes values as follows.

<b>text:</b> Services	<b>textSize:</b> 20sp
-----------------------	-----------------------

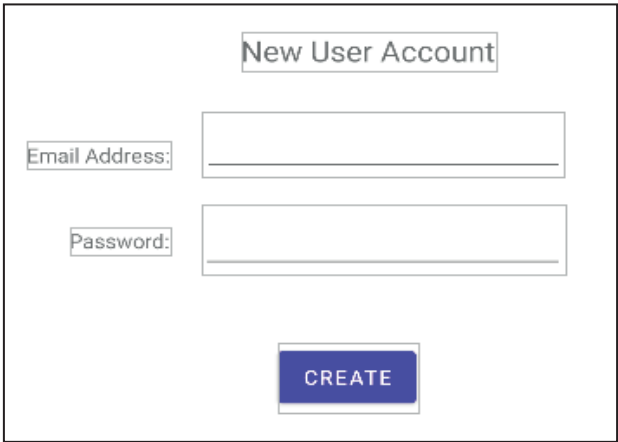
Your **activity\_service.xml** interface should look like the following design.



9- Now, create your Signup activity. To do this, right click **layout** → **New** → **Activity** → **Empty Activity**

Type: **Signup** for the **Activity Name** and click **Finish**.

10- Open the **activity\_signup.xml** file in the **Design** mode and add and configure the widgets which will give you the following design:



To do this, add a **TextView** widget at the top of your activity, set its constraints and attributes values as follows:

<b>text:</b> New User Account	<b>textSize:</b> 20sp
-------------------------------	-----------------------

Add two **TextView** widgets (email address and password), set their constraints and set their attributes values as follows:

<b>text:</b> Email Address	<b>textSize:</b> 14sp
----------------------------	-----------------------

<b>text:</b> Password	<b>textSize:</b> 14sp
-----------------------	-----------------------

Add two Plain Text widgets, set their constraints and set their attributes values as follows:

<b>id:</b> emailSignupId	Delete the attribute value of the <b>text</b> attribute (keep it empty)
--------------------------	---

<b>id:</b> passwordSignupId	Delete the attribute value of the <b>text</b> attribute (keep it empty)
-----------------------------	---

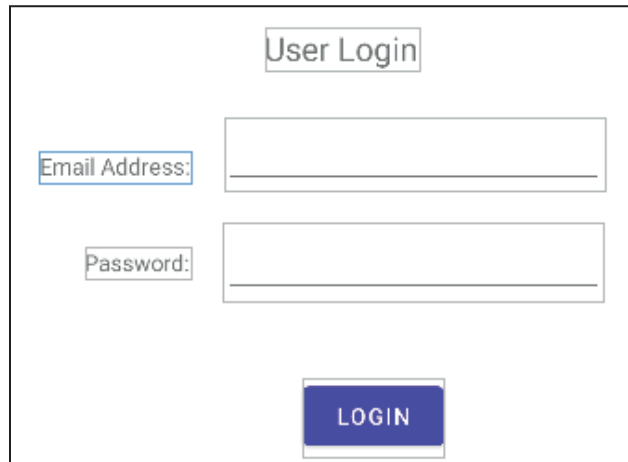
Add a **Button** widget, set its constraints and set its attributes values as follows:

<b>id:</b> createBtn	<b>text:</b> Create
----------------------	---------------------

9- Now, create your Login activity. To do this, right click **layout** → **New** → **Activity** → **Empty Activity**

Type: **Login** for the **Activity Name** and click **Finish**.

10- Your login activity should have the following design:

A wireframe of a 'User Login' activity. At the top center is a title 'User Login' in a rounded rectangle. Below it, on the left, are two labels: 'Email Address:' and 'Password:'. To the right of each label is a corresponding text input field. At the bottom center is a blue button with the text 'LOGIN' in white capital letters.

As you see, it has a similar design to the signup activity. To save your time and effort, open the **activity\_signup.xml** file in the **Code** mode, copy all the widgets code, then open the **activity\_login.xml** file in the **Code** mode and paste it in the same location.

Now, change the attributes values as follows:

<b>text:</b> User Login	<b>textSize:</b> 20sp
-------------------------	-----------------------

Modify the **id** attribute value for each **Plain Text** (`EditText`) widget as follows:

<b>id:</b> emailLoginId	<b>id:</b> passwordLoginId
-------------------------	----------------------------

Modify the **Button** widget attributes values as follows:

<b>id:</b> loginBtn	<b>text:</b> Login
---------------------	--------------------

11- Open the **MainActivity** file and use the Intent class to configure that when the app user taps the: **New User Account** button (id: signupBtn), he/she will go to the **Signup** activity. Also, when the app user taps the **User Login** button (id: loginBtn), he/she will go to the **Login** activity. The code is as follows:

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        signupBtn.setOnClickListener{
            startActivity(Intent(this, Signup::class.java))
        }

        loginBtn.setOnClickListener{
            startActivity(Intent(this, Login::class.java))
        }
    }
}
```



In the code above, double click **signupBtn**, click the red pop-up lamp and click **Import**

In the code above, double click **Intent**, click the red pop-up lamp and click **Import**

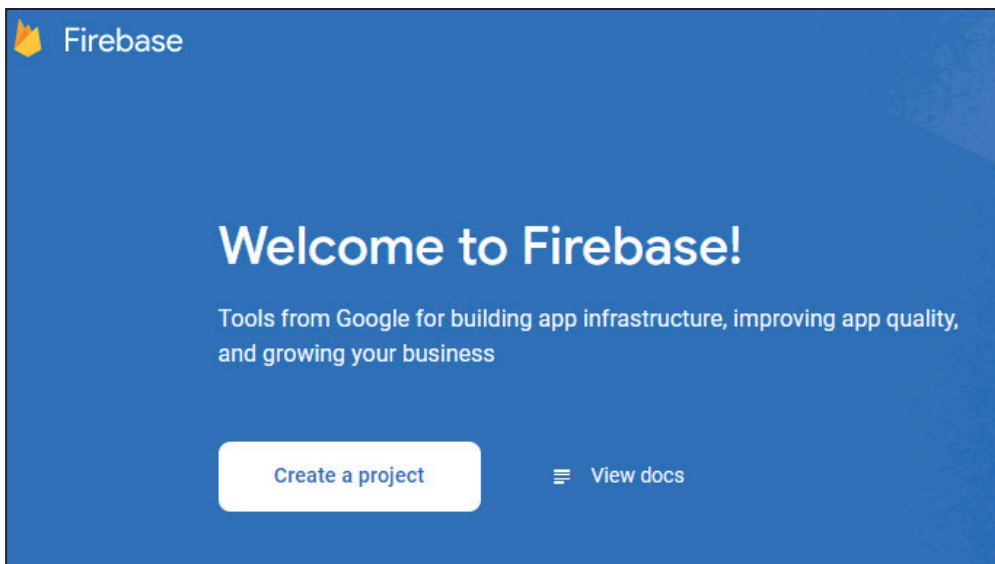
12- Now your Android app is ready for the authentication configurations step. Just, run your app to be sure the navigation from the MainActivity to other activities using the navigation buttons is working fine.

## Configure your App to use Firebase Services

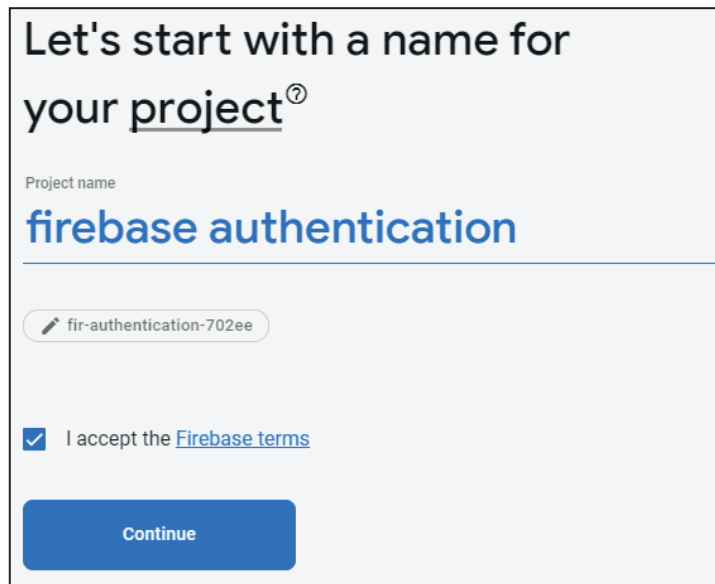
13- Go to: **<https://console.firebase.google.com>**

14- Sign into Firebase using your Google account (Gmail).

15- You will get the following web page:




16- Click **Create a project**. Fill out your project name "firebase authentication" or any other name. Check **I accept the Firebase terms** and then click **Continue** as illustrated in the following figure:



Let's start with a name for  
your project<sup>?</sup>

Project name

firebase authentication

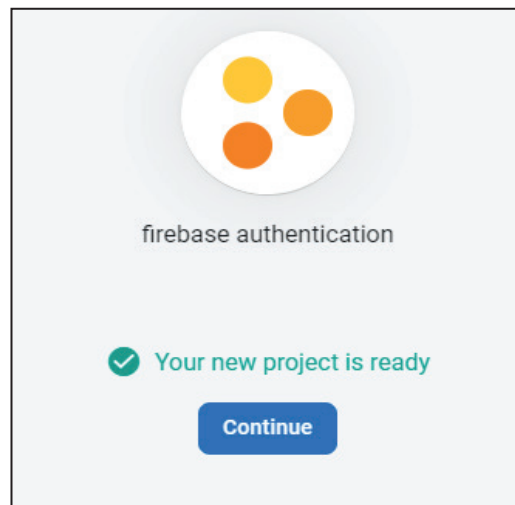
 fir-authentication-702ee

☒ I accept the [Firebase terms](#)

Continue

17- Click **Enable Google Analytics for this project** to **disable** this feature, because this app is just for testing how Firebase authentication works. However, these Google features are important for releasing a report about your live apps in the future.

Click **Create Project** button. Then, after seconds, you should get the following message as illustrated in the following figure telling you that your project has been created at Google Firebase. Then, click **Continue**



Now, as you see in the figure below, your project name: **firebase authentication** is created at the Firebase web server.



If you forgot your app name, go to: **Android Studio** and open the following path:

## Gradle Scripts → build.gradle (Module:Lesson10\_Authentication.app)

Then, scroll down the file content, as illustrated in the figure below, the **application Id** is your app id. Here, in this example, it is: **com.androidatc.lesson10\_authentication**

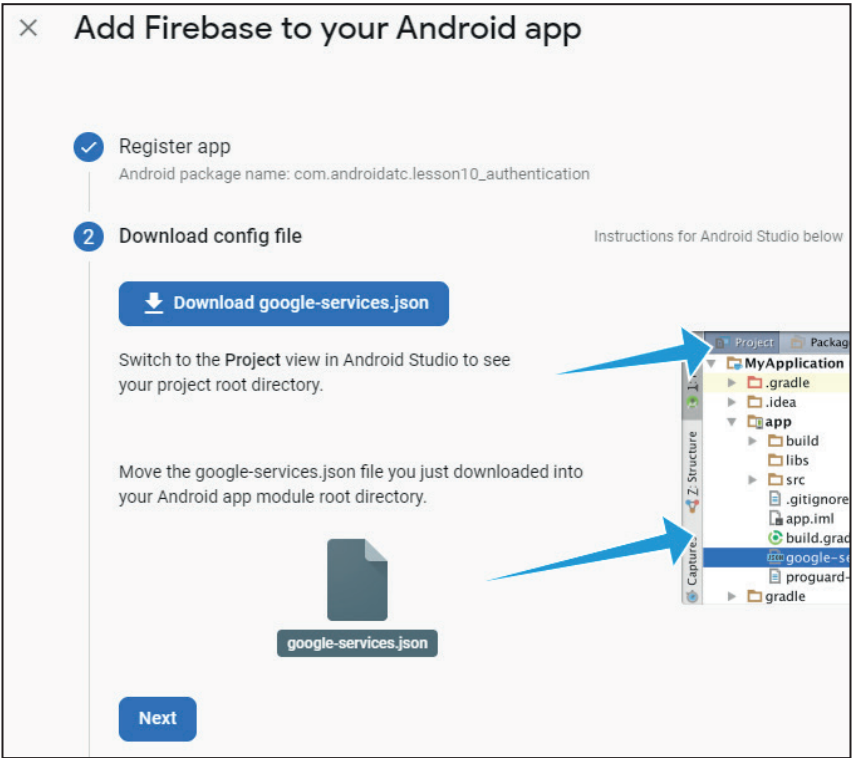
```
defaultConfig {
    applicationId "com.androidatc.lesson10_authentication"
    minSdk 29
    targetSdk 30
    versionCode 1
    versionName "1.0"
}
```

19- Copy the application Id, go to the [Firebase website](#), paste it in your **Android package name**, and then click **Register app** button as illustrated in the following figure:

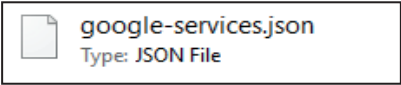
[illegible]

20- In this step, as illustrated in the following figure, you should click:

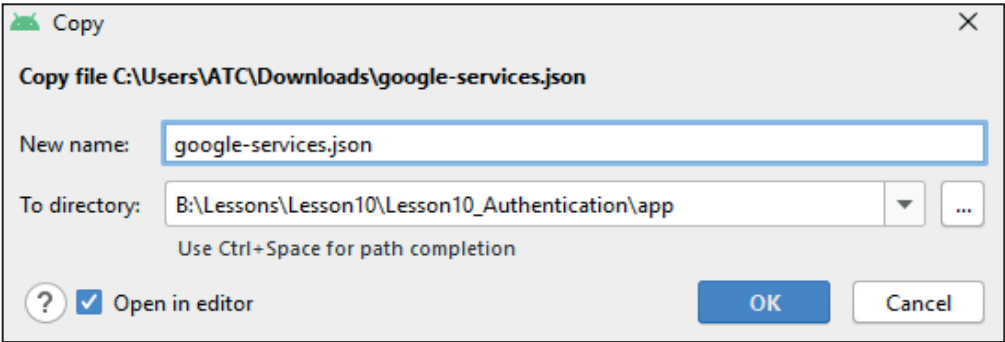
**Download google-service.json** button to download the JSON configuration file. This configuration file includes your **https** connection settings between your Android app and the Firebase services. If you download this file many times, you should use the file which has exactly this name: **google-services.json** without any number.



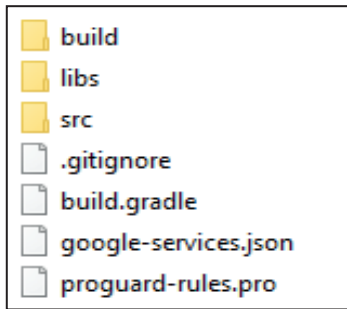
21- Open your download folder. You should find the following file:



Copy this file: **google-services.json** from your download folder, open your Android Studio, right click your **app** folder and select **Paste**. You should get the following dialog box, select **OK**



You should get the content of the **google-services.json** file in a new tab in Android Studio. This file includes some important information you should use in the next steps. If you happen to close this file, you may open it again with Notepad from your **app** folder as illustrated in the figure below:



22- Now, return to the Firebase web site to complete the setup steps. Click **Next**.

23- In this step you should add some Google services plugin configuration in your app.

Open **build.gradle (Project: Lesson10\_Authentication)** file and check if it has the gray highlighted part of the following code or not. If it does not, add it. Then click **Sync Now**

```
buildscript { repositories {  
    google()  
}  
dependencies {  
    classpath 'com.google.gms:google-services:4.3.10'  
}  
}
```

My configuration for this file was as follows:

```
// Top-level build file where you can add configuration options common to all sub-projects  
buildscript {  
    repositories {  
        google()  
        mavenCentral()  
    }  
    dependencies {  
        classpath "com.android.tools.build:gradle:7.0.0"  
        classpath "org.jetbrains.kotlin:kotlin-gradle-plugin:1.5.21"  
        classpath 'com.google.gms:google-services:4.3.10'  
  
        // NOTE: Do not place your application dependencies here; they belong  
        // in the individual module build.gradle files  
    }  
}  
  
task clean(type: Delete) {  
    delete rootProject.buildDir  
}
```

24- Open **build.gradle (Module:Lesson10\_Authentication.app)**, and add to the **plugins** list the following:

```
id 'com.google.gms.google-services'
```

Your code above should be added as is illustrated in the following figure, then click **Sync Now**.

```
plugins {
    id 'com.android.application'
    id 'kotlin-android'
    id 'kotlin-android-extensions'
    id 'com.google.gms.google-services'
```

Also, at the end of the file add the following code to the **dependencies**:

```
implementation platform('com.google.firebase:firebase-bom:28.3.1')
implementation 'com.google.firebase:firebase-auth-ktx'
```

Your last code should be added as is illustrated in the following figure.

```
dependencies {

    implementation 'androidx.core:core-ktx:1.6.0'
    implementation 'androidx.appcompat:appcompat:1.3.1'
    implementation 'com.google.android.material:material:1.4.0'
    implementation 'androidx.constraintlayout:constraintlayout:2.1.0'
    testImplementation 'junit:junit:4.13.2'
    androidTestImplementation 'androidx.test.ext:junit:1.1.3'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.4.0'
    implementation platform('com.google.firebase:firebase-bom:28.3.1')
    implementation 'com.google.firebase:firebase-auth-ktx'
```

Click **Sync Now**.

25- Now, return to the Firebase web site to complete the setup steps. Click **Next**. You should get the step which is illustrated in the following below. Click the **Continue to console** button.

Make sure to check out the [documentation](#) to learn how to get started with what you want to use in your app.

You can also explore [sample Firebase apps](#).

Or, continue to the console to explore Firebase.

Previous

Continue to console

26- Now, stop your app and then run it again to apply the changes in your Android Studio Gradle files.

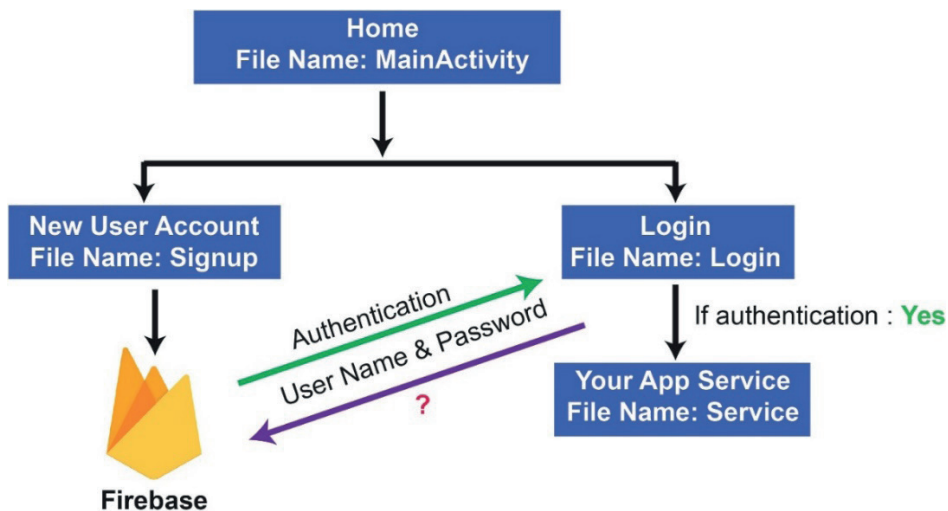
Now, as you see, your Android app has been configured with Firebase and it is ready to exchange or use the Firebase plug-in services, such as database or user authentication services.

## Configuring Firebase Authentication

To sign a user into your app, you should first get authentication credentials from the user (Sing-up or Create a New User). These credentials can be the user's email address and password. Then, pass these credentials to the Firebase Authentication SDK. Firebase backend services will then verify those credentials and return a response to the app user.

After a successful sign in, you can access the user's basic profile information and you can control the user's access to the data stored in other Firebase products. Also, you can use the authentication tokens provided to verify the identity of users in your own backend services.

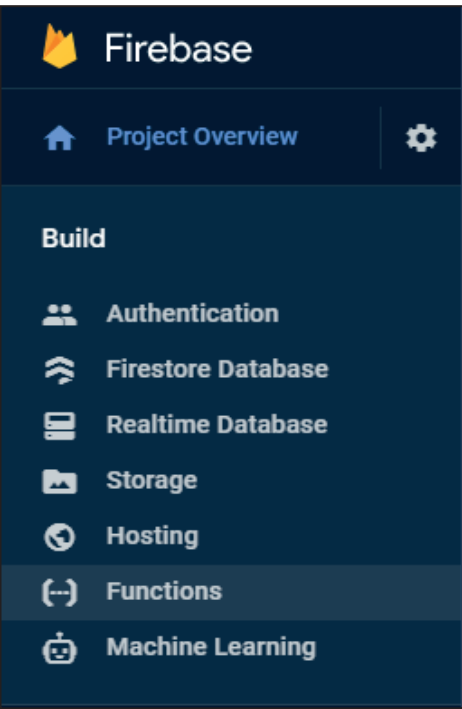
Remember, the figure below explains how your Android app in this example works. In the next step, you will configure the **Signup** file (Kotlin code) to allow each app user to create a new user account (user name & password) at your Firebase Authentication web server.



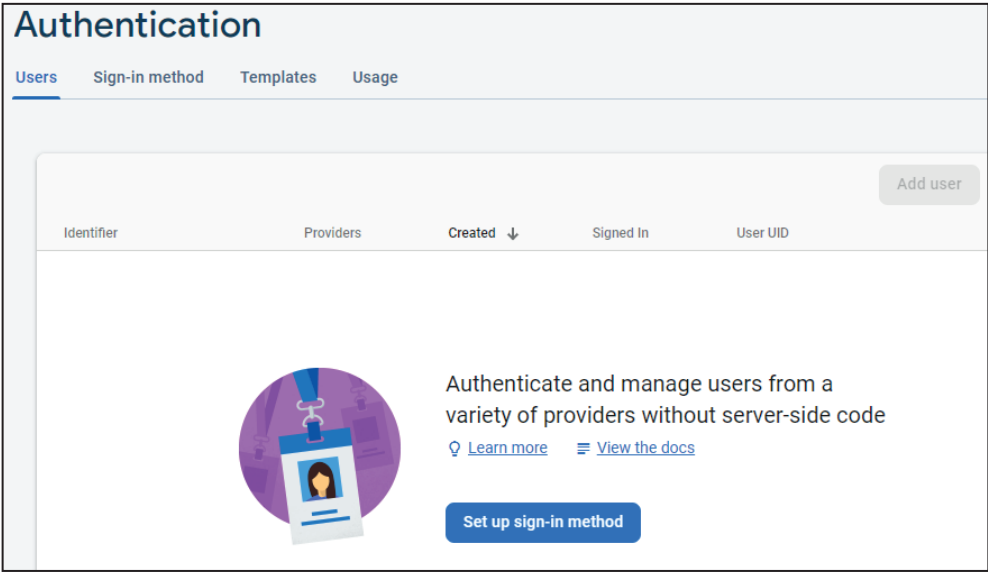
To do this, you should add some Firebase plugins to your app as illustrated in the next steps.

27- Return to the Firebase web site, click the **Build** in the left panel, then as is illustrated in the figure below, click **Authentication**.



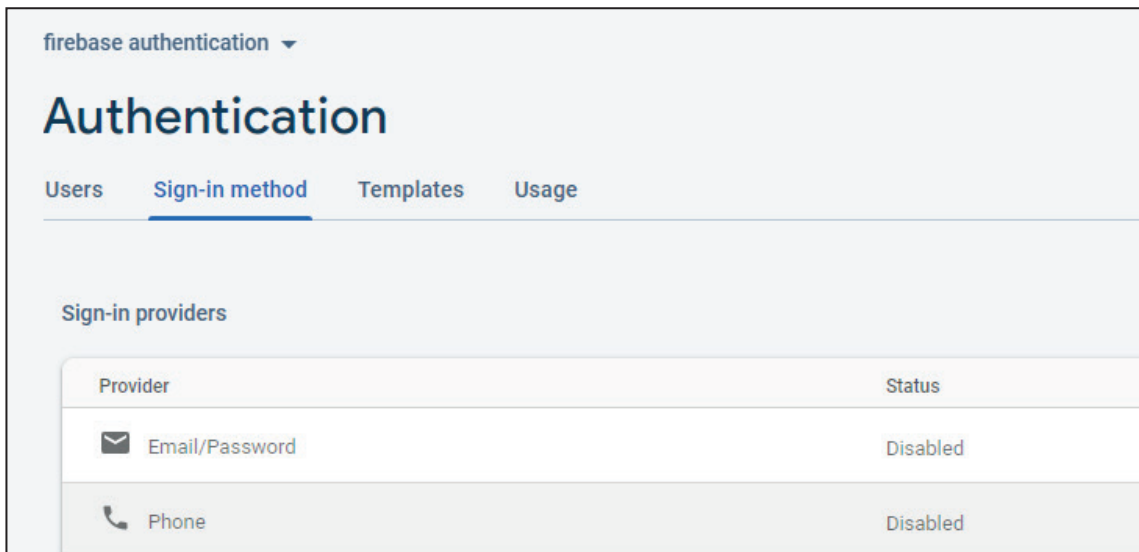


28- You should get the following Sign-in setup web page.

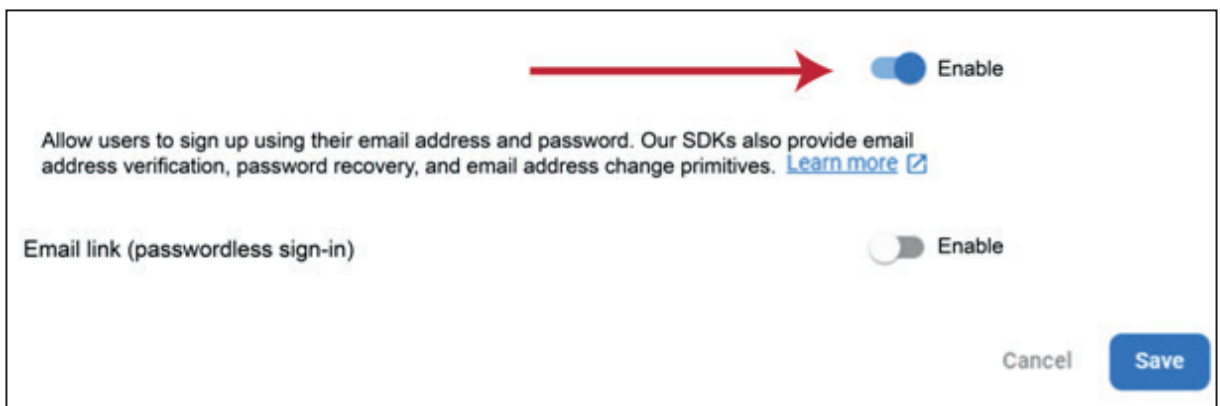


As you see in the figure above, the **Users** tab is still empty because no user account has been created through your **Signup** activity yet.

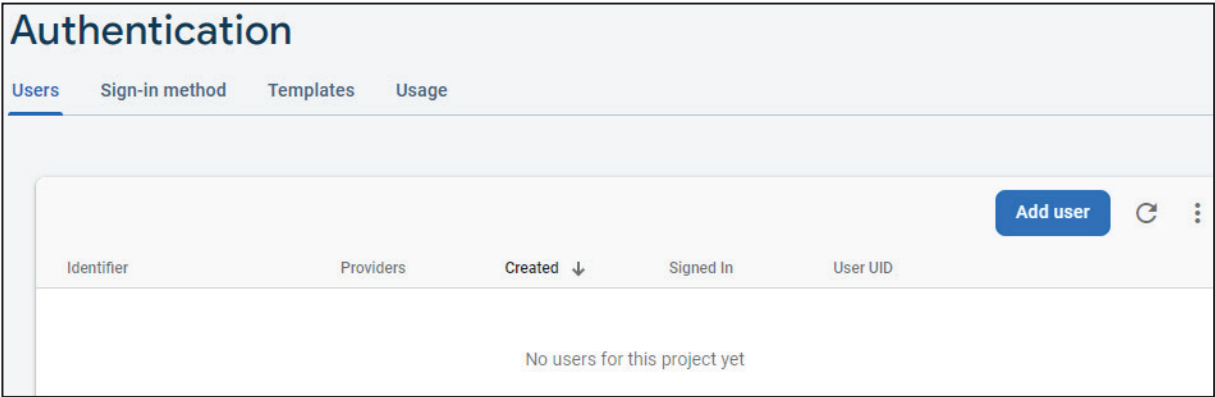
29- In your current **Firebase Authentication** web page, click the **Sign-in method** tab, and as illustrated in the figure below, click the **Email/Password**.



30- As illustrated in the figure below, click **Enable** to allow users to sign up using their email addresses and passwords, then click **Save**.



Now, if you click the Users tab, you should have the settings below. We will not create the app users manually because that will happen through the **Signup** activity as you will see in the next steps.



31- Back to your Android Studio. Before you make any configurations, you should know that your app needs to connect to the Internet to send the app user information (email addresses and passwords) to the Firebase web server; therefore, you should give your app permission to connect to the Internet by adding the following tag to your **AndroidManifest.xml** file:

```

<uses-permission android:name="android.permission.INTERNET" />

```

Add this tag as is illustrated in the following figure:

```

1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3      package="com.androidatc.lesson10_authentication">
4
5      <uses-permission android:name="android.permission.INTERNET" />
6

```

32- When your app user taps the **Signup** button, he/she should create his/her account at Firebase authentication service. However, you should forward this app user to another activity by including a welcome text message. But if app user information (user name and password) is incorrect or invalid, the app user should get a message (Snackbar message) telling him/her to enter a valid user name or password. Now, you should create a new activity which displays a welcome message. You will use this activity later in your configuration. To create this activity, right click **layout** → **New** → **Activity** → **Empty Activity**

Type **welcome** for the **Activity Name** and click **Finish**.

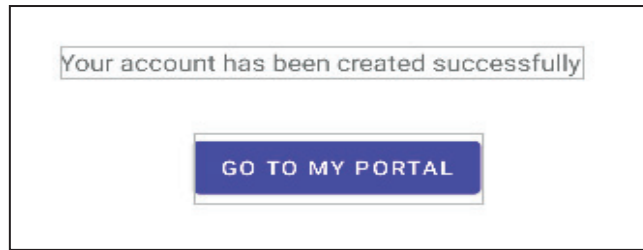
Open the **activity\_welcome.xml** file in the **Design** mode, add a **TextView** widget to this activity, set its constraints and set its attributes values as follows:

<b>text:</b> Your account has been created successfully.	<b>textSize:</b> 16sp
--	-----------------------

As illustrated in the figure below, add a **Button** widget, set its constraints, and set its attributes values as follows:

<b>id</b> : goToPortalBtn	<b>text:</b> Go to my Portal
---------------------------	------------------------------

Your activity should look like the following figure:



33- When the app user taps the: **GO TO MY PORTAL** button, he/she should go to the **Service** activity.

To do this, open the **welcome** Kotlin file and add the following code:

```
class welcome : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_welcome)

        goToPortalBtn.setOnClickListener() {
            startActivity(Intent(this, Service::class.java))
        }
    }
}
```

Double click **goToPortalBtn**, click the red pop-up lamp and select **Import**.

Double click **Intent**, click the red pop-up lamp and select **Import**.

34- Now, it is time to use Firebase authentication plug-in service in your app. Open **Signup** Kotlin file, and add the following code:

```
class Signup : AppCompatActivity() {

    private lateinit var auth: FirebaseAuth

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_signup)

        auth = FirebaseAuth.getInstance()
    }
}
```

Double click **FirebaseAuth**, click the red pop-up lamp and select **Import**.

In the previous code, you have configured **auth** as a static instance object of the **FirebaseAuth** class. **auth** in your code will represent your app user name (email) and password. Until now, you have initialized the **auth** Firebase instance which will be used later in

creating and authenticating in your Kotlin code.

35- In the same file (**Signup** Kotlin file), create the **createUser** function as is illustrated in the code below:

```
class Signup : AppCompatActivity() {

    private lateinit var auth: FirebaseAuth

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_signup)

        auth = FirebaseAuth.getInstance()
    }

    fun createUser(email: String, password: String) {
        auth.createUserWithEmailAndPassword(email, password)
            .addOnCompleteListener(this) { task ->

                if (task.isSuccessful) {
                    startActivity(Intent(this, welcome::class.java))
                }

                else {
                    Snackbar.make(
                        findViewById(R.id.createBtn),
                        "Enter a valid username and password (6 characters)",
                        Snackbar.LENGTH_LONG
                    ).show()
                }

            }
    }
}
```

Double click, **Intent** click the red pop-up lamp, and select **Import**.

Double click **Snackbar**, click the red pop-up lamp, and select **Import**.

In the **createUser** function above, you added an **IF** statement configuration about if the user name and password have been created successfully, your app will forward your app user to the **activity\_welcome.xml** layout which should include the welcome message. Otherwise, your app user will get a Snackbar message telling him/her to check their input and enter a valid user name and password.

36- On the same **Signup** Kotlin file, add the following code which will use **createUser** function to create the app user's user name and password at the Firebase authentication database when the app user taps the Create button. The code as is illustrated in the gray highlighted code below:

```

class Signup : AppCompatActivity() {

    private lateinit var auth: FirebaseAuth

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_signup)

        auth = FirebaseAuth.getInstance()

        createBtn.setOnClickListener() {
            if (emailSignupId.text.trim().toString().isEmpty()
                || passwordSignupId.text.trim().toString().
isEmpty()) {

                createUser(emailSignupId.text.trim().
toString(), passwordSignupId.text.trim().toString())
            }

            else {
                Snackbar.make(findViewById(R.id.createBtn),
                    "check your username and password then try again",
                    Snackbar.LENGTH_LONG).show()
            }
        }

    }

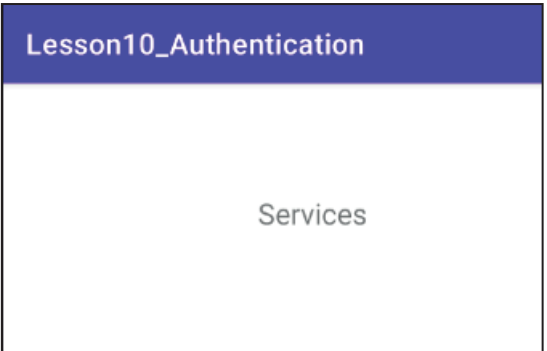
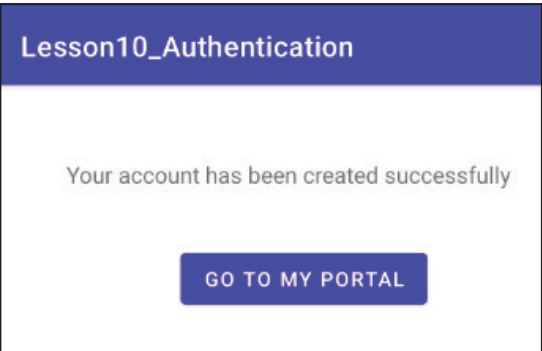
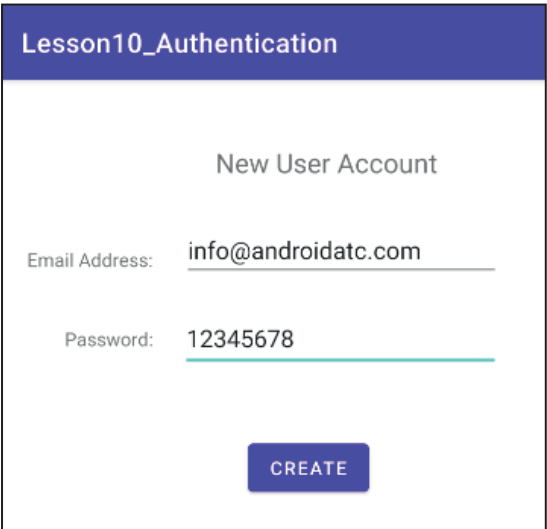
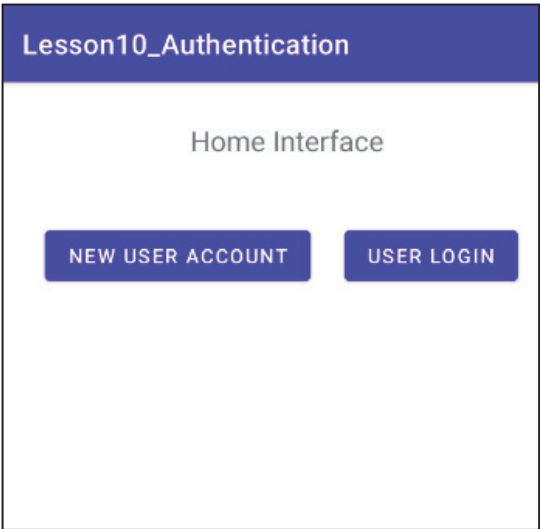
    fun createUser(email: String, password: String) {
        auth.createUserWithEmailAndPassword(email, password)
            .addOnCompleteListener(this) { task ->
                if (task.isSuccessful) {
                    startActivity(Intent(this, welcome::class.java))
                } else {
                    Snackbar.make(
                        findViewById(R.id.createBtn),
                        "Enter a valid username and password (6 characters)",
                        Snackbar.LENGTH_LONG
                    ).show()
                }
            }
    }
}

```

Double click `emailSignupId`, click the red pop-up lamp and select **Import**.

37- **Stop**, then **Run** your app. Tap the **New User Account** button, type your email address and any password (at least 6 characters) and tap the **CREATE** button. Your account should be created and the Intent class will move you directly to the **Welcome** activity. Tap the **GO TO MY PORTAL** button to open the Service activity.

**Note:** This code is run using the Pixel 5 API 30 emulator. Also, it is tested using a physically connected Samsung phone. However, it did not work when tested using Pixel 3 XL API 30 emulator.



Back to your Firebase web site, in the authentication part, click the **Users** tab and click **Refresh**. As is illustrated in the figure below, you should find a new user account that has been created.

Search by email address, phone number, or user UID					Add user	↺	⋮
Identifier	Providers	Created ↓	Signed In	User UID			
info@androidatc.com	✉	Aug 16, 2021	Aug 16, 2021	xNgmdAtDjwU938wL5Qq4NWmn...			
Rows per page: 50					1 - 1 of 1	⏪	⏩

**Note:** If you want your password to appear as an asterisk (\*) when your app user enters his/her password, open the **activity\_signup.xml** file in the **Code** mode, and then replace the attribute for the password **EditText** widget: **android:inputType="textPersonName"** with: **android:inputType="textPassword"**

## Login to App Using a Firebase User Accounts

Now, you will configure your **Login** (Kotlin file) to allow your app user who has an account at your Firebase authentication database to login to your app. Also, if this user is able to login successfully, then he/she will be moved to the **activity\_service.xml** interface.

You will use almost the same code of the **Signup** file with few changes such as the button id would be: **loginBtn**. Also, the **Plain Text** username id would be: **emailLoginId**, and the **Plain Text** password id would be: **passwordLoginId**

38- Open the **Login** file and type the following code:

```
class Login : AppCompatActivity() {

    lateinit var auth: FirebaseAuth

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_login)

        auth = FirebaseAuth.getInstance()

        loginBtn.setOnClickListener() {

            if (emailLoginId.text.trim().toString().isEmpty() || passwordLoginId.text.trim().toString().isEmpty()) {

                login(emailLoginId.text.trim().toString(), passwordLoginId.text.trim().toString()) }

            else {
                Snackbar.make(findViewById(R.id.loginBtn),
                    "check your username or password then try again",
                    Snackbar.LENGTH_LONG).show()
            }
        }
    }

    fun login(email: String, password: String) {

        auth.signInWithEmailAndPassword(email, password)
```



```

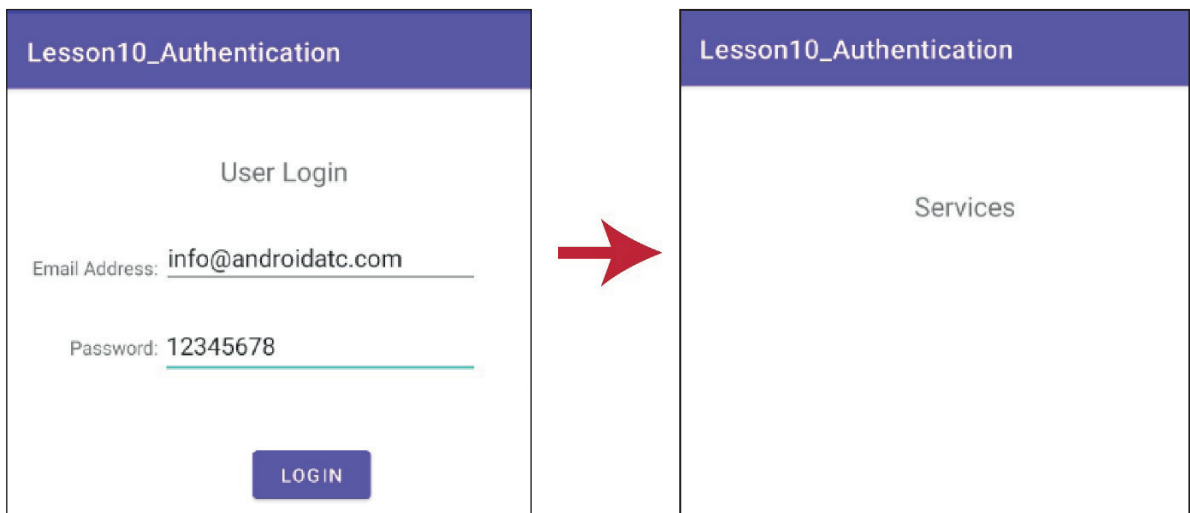
        .addOnCompleteListener(this) { task ->
            if (task.isSuccessful) {
                startActivity(Intent(this, Service::class.java))

            } else {
                Snackbar.make(
                    findViewById(R.id.loginBtn),
                    "Enter a valid username or password",
                    Snackbar.LENGTH_LONG
                ).show()
            }
        }
    }
}

```

Double click **Intent**, click the red pop-up lamp and select **Import**.

39- Stop, then Run your app. Tap the **User Login** button, enter your user name and password for the user account which you have created in the previous topic without any space before or after your email address or your password, and then tap the **LOGIN** button. You should be able to login to the service interface.



Check your Users tab at the Firebase authentication web site, the **Signed In** field will display the date of the user's sign in.

## Logout Configuration

Now, after completing the login step successfully, the app user will get the **Service** interface where all your app services are. Keep in mind, your app user will expect to find a button or icon that has a logout action added to this service interface. In this topic, you will add a logout button to your **activity\_service.xml** file, then add the code which will logout and move your app users to the **MainActivity** interface.

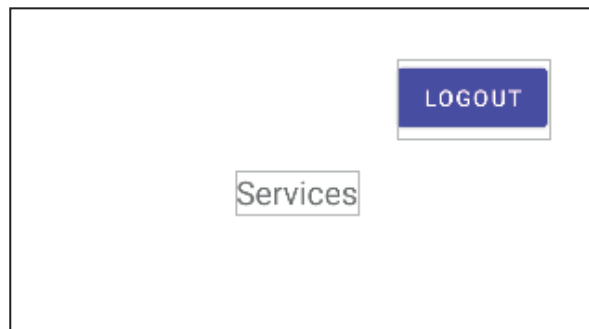
This topic is not related to Firebase database; however, it is important to add this option to apps that have authentication configuration. However, when you design your app, it is better to add the logout choice to your Drawer list or menu icon.

The steps are as follows:

40- Open **activity\_service.xml** in the **Design** mode, add a **Button** widget using drag and drop, set its constraints, and set its attributes values as follows:

<b>id:</b> logoutBtn	<b>text:</b> Logout
----------------------	---------------------

Your service interface should look like the following design:

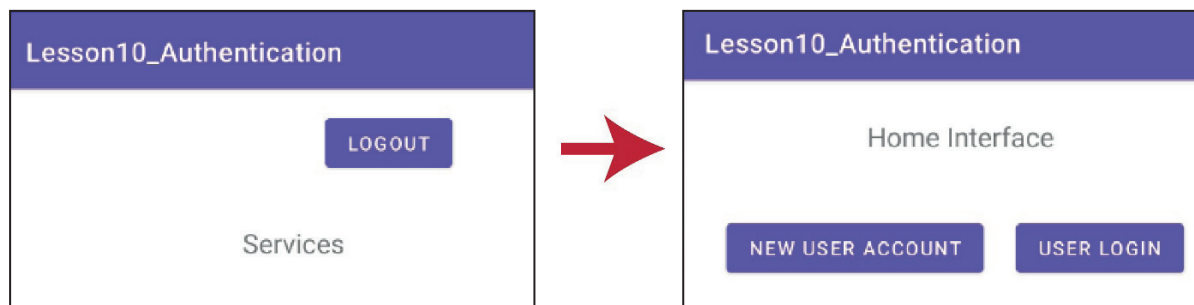


41- Open Service (Kotlin file) and add the following code:

```
class Service : AppCompatActivity() {  
  
    private lateinit var auth: FirebaseAuth  
  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContentView(R.layout.activity_service)  
  
        auth = FirebaseAuth.getInstance()  
  
        logoutBtn.setOnClickListener {  
            auth.signOut()  
            startActivity(Intent(this, MainActivity::class.java))  
        }  
    }  
}
```

Double click **Intent**, click the red pop-up lamp, and select **Import**.

42- Stop, then Run your app again. Tap the **User Login** button, enter your user name and password, you will login to the service interface. As is illustrated in the following figure, when you tap the Logout button, the app will close the session, and open the MainActivity interface. If you tap the USER LOGIN button again, the app will ask you to enter the user name and password.

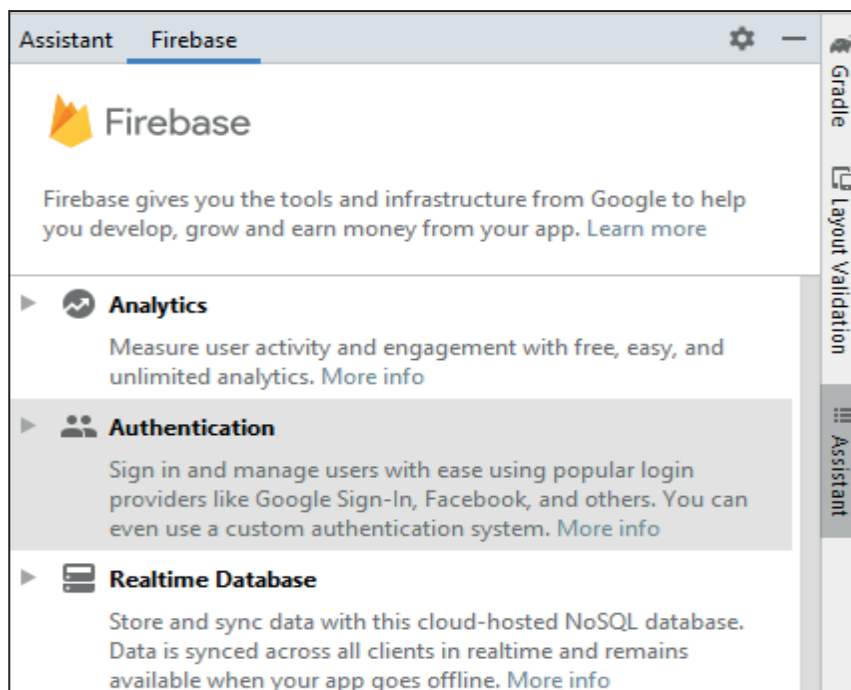


## Using Firebase Assistant with Android Studio

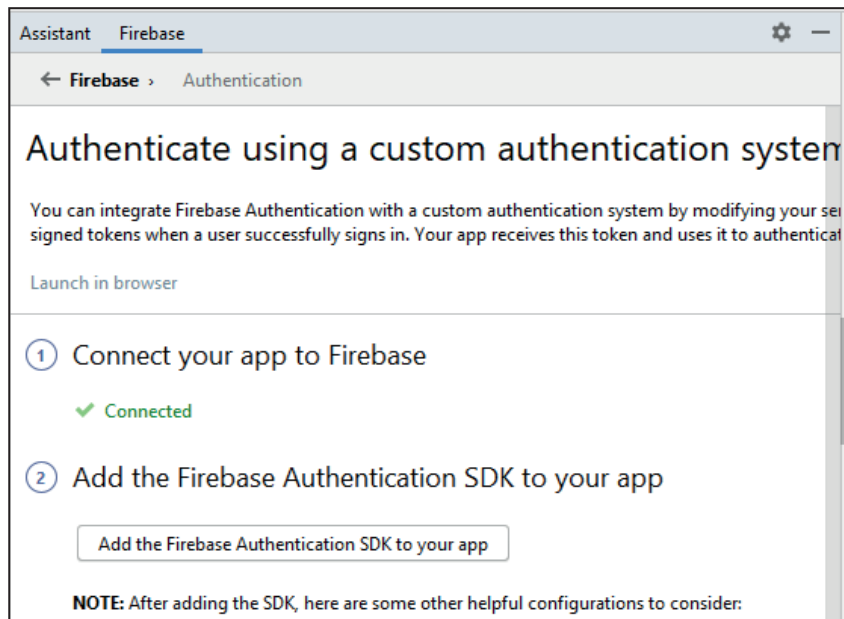
You may use Android Studio assistance to do some of the tasks which you have completed with the previous Firestore authentication example without the need to type a lot of code.

The Firestore Assistant registers your app with a Firestore project and adds automatically the necessary Firestore files, plugins, and dependencies to your Android project— all from within Android Studio!

To get the Firestore assistance in your Android Studio, from the menu bar, click **Tools → Firestore → Firestore**. As you see in the figure below, you will get a new panel on the right side of your Android Studio:



Click Authentication, select: Authentication using a custom authentication system. If you already added the file: google-services.json to your app and configured your app **AndroidManifest.xml** file to give your app permission to connect to the Internet, you can click the web link : **Connect your app to Firestore** as is illustrated in the following figure:



Also, in the same Firebase assistance panel above, if you click the web link: **Add the Firebase Authentication SDK to your app**, all the Firebase authentication dependency configurations which you have added to your app **build.gradle** files will be added automatically.

This assistance is very important because you may use it to add or to have an idea about the code configuration for each type of the Firebase services.

## Firestore Database

Firestore offers two cloud-based, client-accessible database solutions that support real-time data syncing:

**a- Realtime Database:** is Firestore's original database. It's an efficient, low-latency solution for mobile apps that require synced states across clients in real-time. Data is synced across all clients in real-time, and remains available when your app goes offline.

**b- Cloud Firestore:** is Firestore's newest database for mobile app development. It builds on the successes of the Realtime Database with a new, more intuitive data model. Cloud Firestore also features richer, faster queries and scales further than the Realtime Database.

## Which database is right for your project?

Google recommends Cloud Firestore for most developers starting a new project. Cloud Firestore offers additional functionality, performance, and scalability on an infrastructure designed to support more powerful features for future releases. Expect to see new query types, more robust security rules and improvements in performance among the advanced features planned for Cloud Firestore.

Both Realtime Database and Cloud Firestore are **NoSQL** Databases.

As you are choosing between database solutions, consider the following differences between Cloud Firestore and the Realtime Database.

Realtime Database	Cloud Firestore
Stores data as one large JSON tree.	Stores data as a collection of documents which is very similar to JSON.
Offline support for iOS and Android clients.	Offline support for iOS, Android, and web clients.
It can record client connection status(online or offline) and provide updates every time the client's connection state changes.	Presence is not supported natively.
Queries can sort or filter a property, but not both.	You can chain filters and combine filtering and sorting on a property in a single query.
Databases are limited to zonal availability in a single region.	Cloud Firestore is a multi-region solution that scales automatically.
Scaling requires splitting your data across multiple database instances in the same Firebase project.	Scaling is automatic.
Security: <ul style="list-style-type: none"> <li>• Reads and writes from mobile SDKs secured by Realtime Database Rules.</li> <li>• Reads and writes rules cascade.</li> <li>• Can validate data separately using the validate rule.</li> </ul>	Security: <ul style="list-style-type: none"> <li>• Reads and writes from mobile SDKs secured by Cloud Firestore Security Rules.</li> <li>• Reads and writes from server SDKs secured by Identity and Access Management (IAM).</li> <li>• Rules don't cascade unless you use a wildcard.</li> <li>• Rules can constrain queries: If a query's results might contain data the user doesn't have access to, the entire query fails.</li> </ul>
Charges only for bandwidth and storage, but at a higher rate.	Charges primarily on operations performed in your database (read, write, delete) and, at a lower rate, bandwidth and storage.

## Real Time Database

The Firebase Realtime Database is a cloud-hosted database. Data is stored as JSON and synchronized in real-time to every connected client. When you build cross-platform apps with our iOS, Android, and JavaScript SDKs, all of your clients share one Realtime Database instance and automatically receive updates with the newest data.

The Realtime Database provides a flexible, expression-based rules language called Firebase

Realtime Database Security Rules, to define how your data should be structured and when the data can be read from or written to. When integrated with Firebase Authentication, developers can define who has access to what data and how they can access it.

Here, you will continue using the same Android project which you used to configure the Firebase authentication. This Android project is already configured with Firebase web site which you configured in the previous topics.

By using the **Firebase Android BoM** (Bill of Materials), your app will always use compatible versions of the Firebase Android libraries. Here is how to use the Firebase Android BoM to declare dependencies in your module (app-level) **Gradle** file (usually app → build.gradle). When using the BoM, you don't specify individual library versions in the dependency lines. You must be sure your app has the following dependencies in your **build.gradle** (Module level):

```
dependencies {  
    // Import the BoM for the Firebase platform  
    implementation platform('com.google.firebase:firebase-bom:28.3.1')  
  
    // Declare the dependency for the Realtime Database library  
    // When using the BoM, you don't specify versions in Firebase  
    //library dependencies  
    implementation 'com.google.firebase:firebase-database-ktx'  
}
```

Also, there are other dependencies' configuration that must be added to your **build.gradle** file such as the following dependency configuration for Firebase authentication:

```
implementation 'com.google.firebase:firebase-auth'
```

Also, you already downloaded your project Firebase JSON file: **google-service.json**, and you added it to your project Android files.

If you want to review these previous settings from the beginning, you will need to go back to step 20 of the previous example.

Now, you will learn how to use the Firebase real time database in an Android project. In this example, you will use this Firebase service in the **Service** file. To do this, perform the following steps which will take you back to the previous steps:

43- In the previous app, add the following dependency to your **build.gradle** (Module level):

```
implementation 'com.google.firebase:firebase-database-ktx'
```

Then, click **Sync Now**.

44- Open the **activity\_signup.xml** file in the Design mode to add the following new TextView and Text Palin widgets:

Lesson10\_Authentication

New User Account

Name:

City:

Country:

Email Address:

Password:

CREATE

To do that, move down the existing form fields a little bit and add three **TextView** widgets to this activity, set their constraints and set their attributes values as follows:

<b>text:</b> Name:	<b>text:</b> City:	<b>text:</b> Country
--------------------	--------------------	----------------------

Then, add three **Plain Text** widgets, set their constraints and set their attributes values as follows:

<b>id:</b> nameld	<b>id:</b> cityId	<b>id:</b> countryId
-------------------	-------------------	----------------------

Delete the text attribute values for all these three **Plain Text** widgets.

Now, your **activity\_signup.xml** file should have the design in the previous figure.

45- Open the **Signup** file (Kotlin file), and add the following gray highlighted code:

**My Users**: will be your real-time database name at Firebase web server.

```

class Signup : AppCompatActivity() {

    private lateinit var auth: FirebaseAuth
    private lateinit var mydatabase:FirebaseDatabase
    private lateinit var myreference:DatabaseReference
    
```

```

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_signup)

    auth = FirebaseAuth.getInstance()
    mydatabase = FirebaseDatabase.getInstance()
    myreference = mydatabase.getReference("My Users")

    createBtn.setOnClickListener() {

```

46- At the end of your **Signup.kt** file (before the last bracket of your code), add the following database model class:

```

class DatabaseModel (var name:String, var city:String, var
country:String){}

```

47- Add the following code of the function which is responsible to send your data to the Firebase web server:

```

private fun sendData(name: String, city: String, country: String){
    var name=nameId.text.toString().trim()
    var city=cityId.text.toString().trim()
    var country=countryId.text.toString().trim()

    if(name.isNotEmpty() && city.isNotEmpty() && country.
isNotEmpty()){

        var model=DatabaseModel(name, city, country)
        var id=myreference.push().key

        // this part of code is responsible to send your date to Firebase
        myreference.child(id!!).setValue(model)

    }
}

```

48- Add to your previous **createBtn.setOnClickListener()** method, the settings of the **sendData** function as illustrated in the gray highlighted code below. This action will call or run your **sendData** function when your app user taps the **Create** button.



```
createBtn.setOnClickListener() {

    if (emailSignupId.text.trim().toString().isEmpty()
        || passwordSignupId.text.trim().toString().isEmpty()) {

        createUser(emailSignupId.text.trim().
toString(), passwordSignupId.text.trim().toString())

        sendData(nameId.text.toString().trim(), cityId.text.toString().
trim(), countryId.text.toString().trim())

    }
}
```

The full code of the **Signup.kt** file is as follows:

```
class Signup : AppCompatActivity() {

    private lateinit var auth: FirebaseAuth
    private lateinit var mydatabase: FirebaseDatabase
    private lateinit var myreference: DatabaseReference

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_signup)

        auth = FirebaseAuth.getInstance()
        mydatabase = FirebaseDatabase.getInstance()
        myreference = mydatabase.getReference("My Users")

        createBtn.setOnClickListener() {

            if (emailSignupId.text.trim().toString().isEmpty()
                || passwordSignupId.text.trim().toString().
isEmpty()) {

                createUser(emailSignupId.text.trim().
toString(), passwordSignupId.text.trim().toString())

                sendData(nameId.text.toString().trim(), cityId.text.toString().
trim(), countryId.text.toString().trim())

            }

            else{
                Snackbar.make(findViewById(R.id.createBtn),
```

```

        "check your username and password then try again",
        Snackbar.LENGTH_LONG).show()

    }

}

fun createUser(email: String, password: String) {
    auth.createUserWithEmailAndPassword(email, password)
        .addOnCompleteListener(this) { task ->
            if (task.isSuccessful) {
                startActivity(Intent(this, welcome::class.java))
            } else {
                Snackbar.make(
                    findViewById(R.id.createBtn),
                    "Enter a valid username and password (6 characters)",
                    Snackbar.LENGTH_LONG
                ).show()
            }
        }
}

private fun sendData(name: String, city: String, country: String){

    var name =nameId.text.toString().trim()
    var city =cityId.text.toString().trim()
    var country =countryId.text.toString().trim()

    if(name.isNotEmpty() && city.isNotEmpty() && country.isNotEmpty()){

        var model= DatabaseModel(name, city, country)
        var id = myreferance.push().key

        // this part of code is responsible to send your date to Firebase
        myreferance.child(id!!).setValue(model)

    }

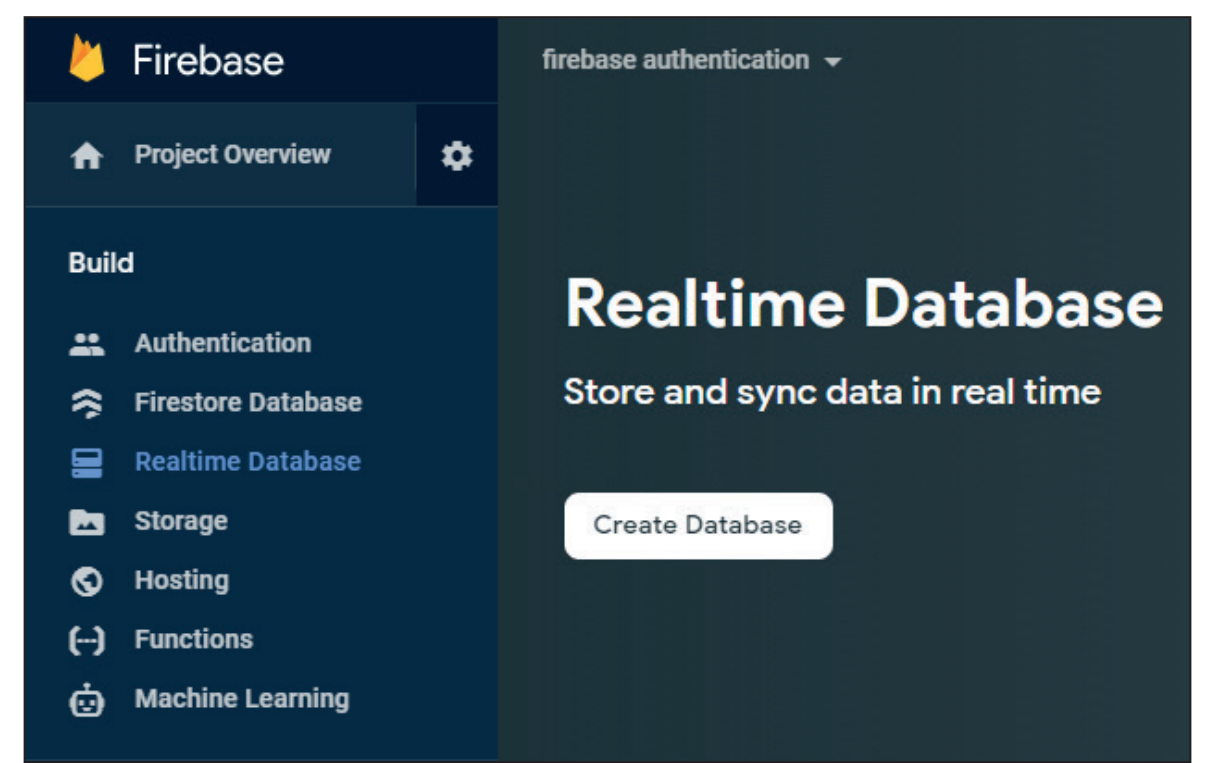
}

class DatabaseModel (var name:String, var city:String, var
country:String){}

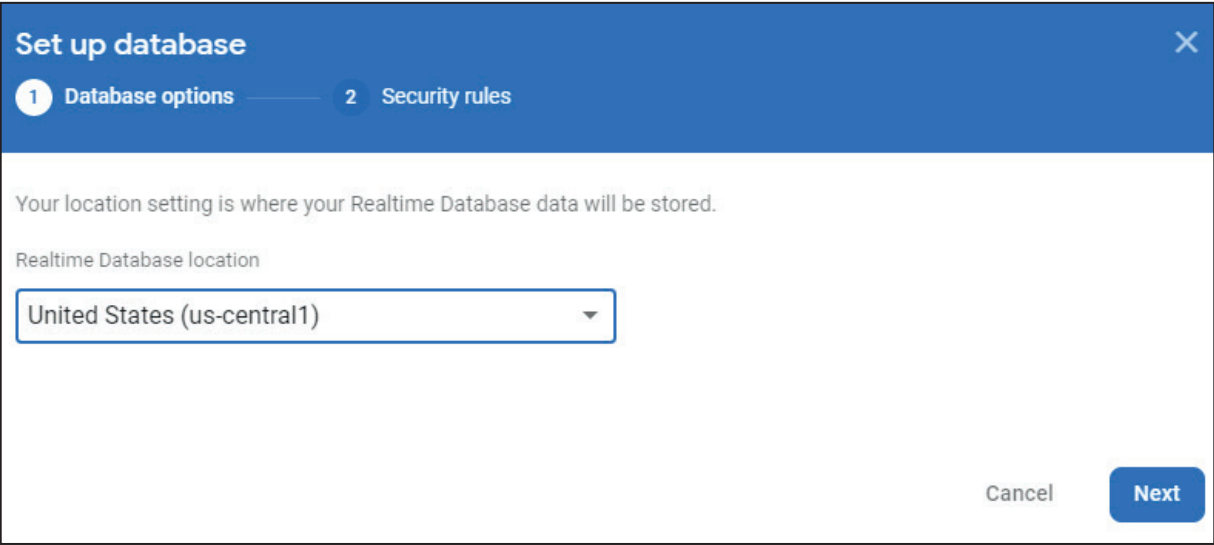
}

```

49- Back to your Firebase web configuration, as is illustrated in the figure below, go to the Build panel, click **Realtime Database**, then click the **Create Database** button.



50- As is illustrated in the figure below, select **United States (us-central1)** for the Firebase web server location and click **Next**



51- In this step and as is illustrated in the figure below, select **Start in test mode**. This means any one can send or retrieve data from your database (write and read) for 30 days. Then, you should configure the security rules as you will see in the next steps. Click **Enable**.

### Set up database

1 Database options

2 Security rules

Once you have defined your data structure you will have to write rules to secure your data.  
[Learn more](#)

☐ Start in **locked mode**  
Your data is private by default. Client read/write access will only be granted as specified by your security rules.

☒ Start in **test mode**  
Your data is open by default to enable quick setup. However, you must update your security rules within 30 days to enable long-term client read/write access.

```
{  "rules": {    ".read": "now < 1631851200000",    // 2021-9-17    ".write": "now < 1631851200000",    // 2021-9-17  } }
```

! The default security rules for test mode allow anyone with your database reference to view, edit and delete all data in your database for the next 30 days

Cancel **Enable**

52- Now as is illustrated in the figure below, your database has been created and ready to receive your app data.

firebase authentication

## Realtime Database

Data

Rules

Backups

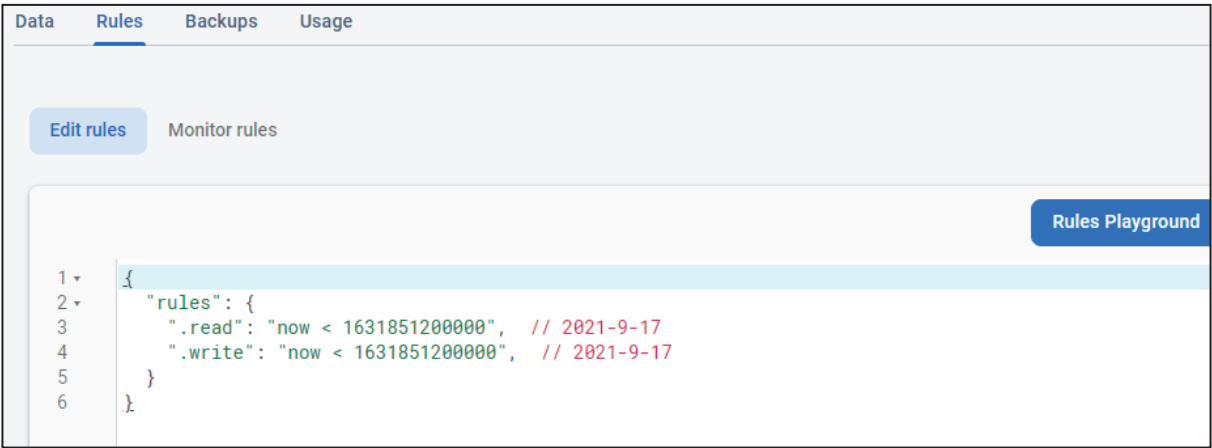
Usage

Protect your Realtime Database resources from abuse, such as billing fraud or phishing

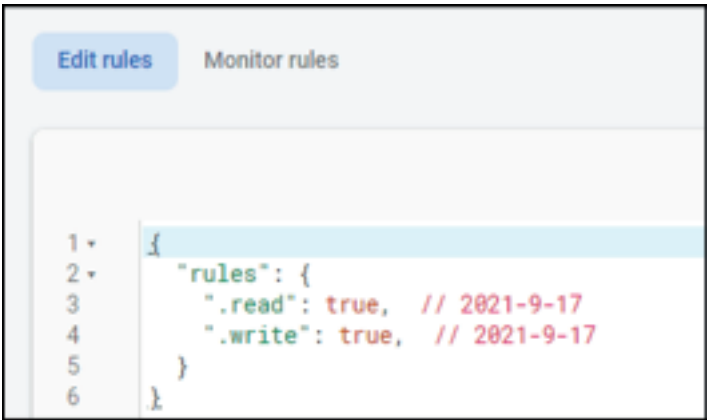
<https://fir-authentication-702ee-default-rtdb.firebaseio.com/>

**fir-authentication-702ee-default-rtdb**: null

53- Click the **Rules** tab, you will find the following default settings:



To give anyone read and write permissions, double click the **"now < 1631851200000"** as illustrated in the following figure and change these settings to: **true**, then click **Publish** button



54- Back to your Android Studio. **Stop**, then **Run** your app. Tap **New User Account** button. As illustrated in the figure below, enter your app user information, then tap the **CREATE** button.

Lesson10\_Authentication

New User Account

Name:Joe Alex

City:Toronto

Country:Canada

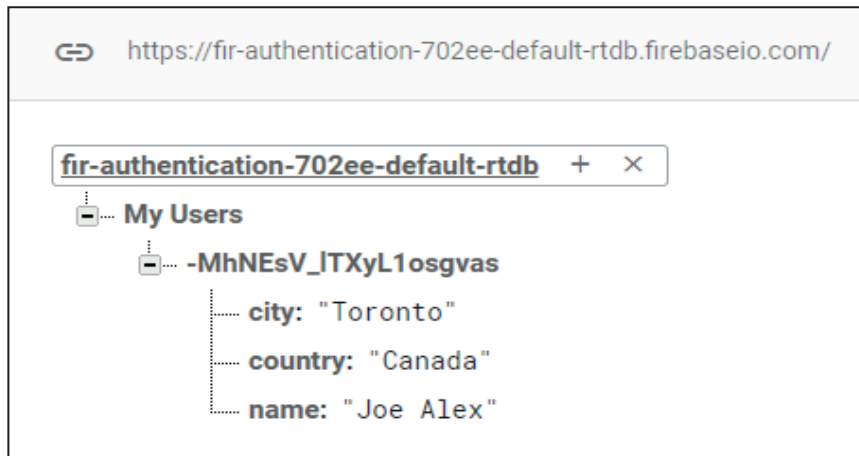
Email Address:joe@androidatc.com

Password:\*\*\*\*\*

CREATE

You should login to your **Welcome** activity. Check your data at the Firebase web server.

Click the **Refresh** button on your web browser and make sure that the user account has been created. Then click the **Realtime Database** on the left side. You should find your app user data has been added as illustrated in the figure below:



This format of database is the same as JSON database and if you click the three vertical ellipsis button which is illustrated in the following figure, you will get a submenu including export and import JSON options.



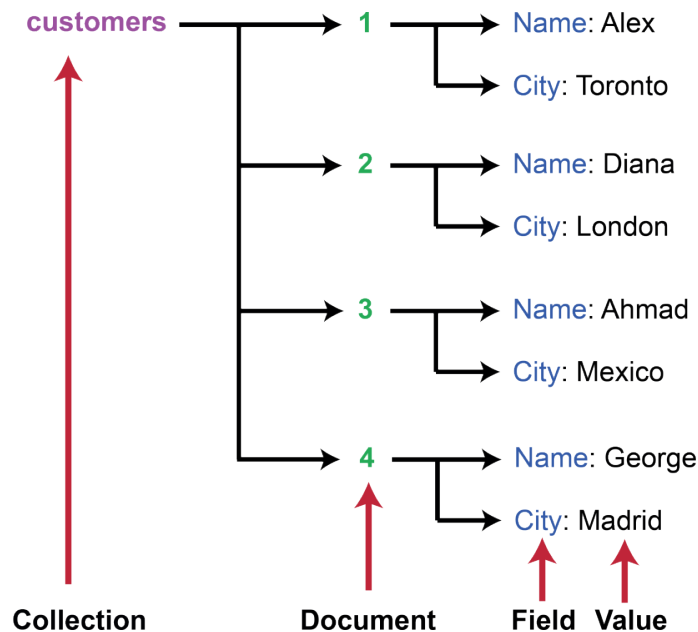
## Cloud Firestore Database

One of the most common data structures is a database table. A database table consists of columns (fields) and rows (records or values) as illustrated in the table below:

Customer Id	Name	City
1	Alex	Toronto
2	Diana	London
3	Ahmad	Mexico
4	George	Madrid

In the cloud Firestore stores, data structure is very similar to JSON tree as a collection of documents.

For example, the previous table will be stored in the cloud Firestore as illustrated in the following figure:

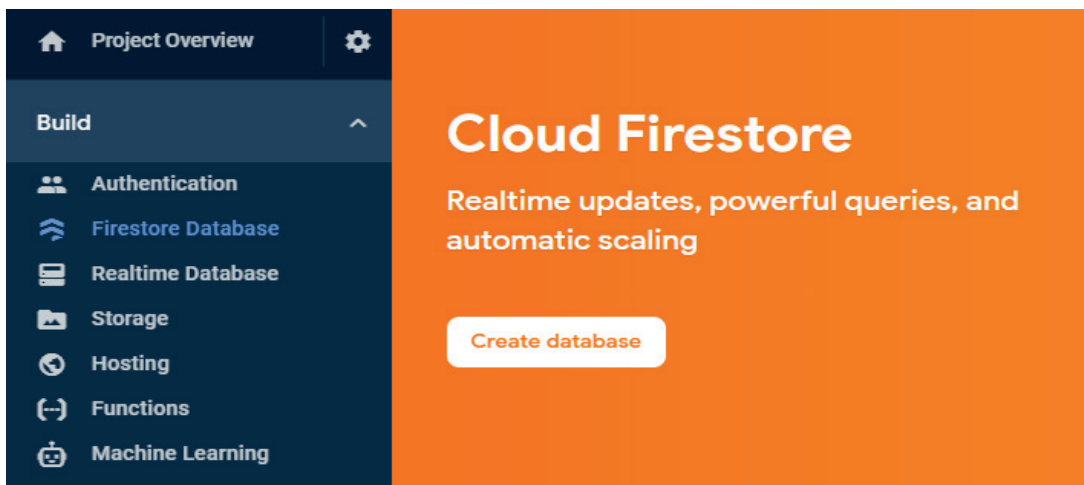


The container of data is called **collection** and the database may include one or more collections (tables). The primary key or id for each record (row) is called **document** and each document includes a group of field values that belong to the same document. For example, in the previous database diagram, Alex is the value of the Name field and Toronto is the value of City field. Also, all of these values belong to the same document which is : 1

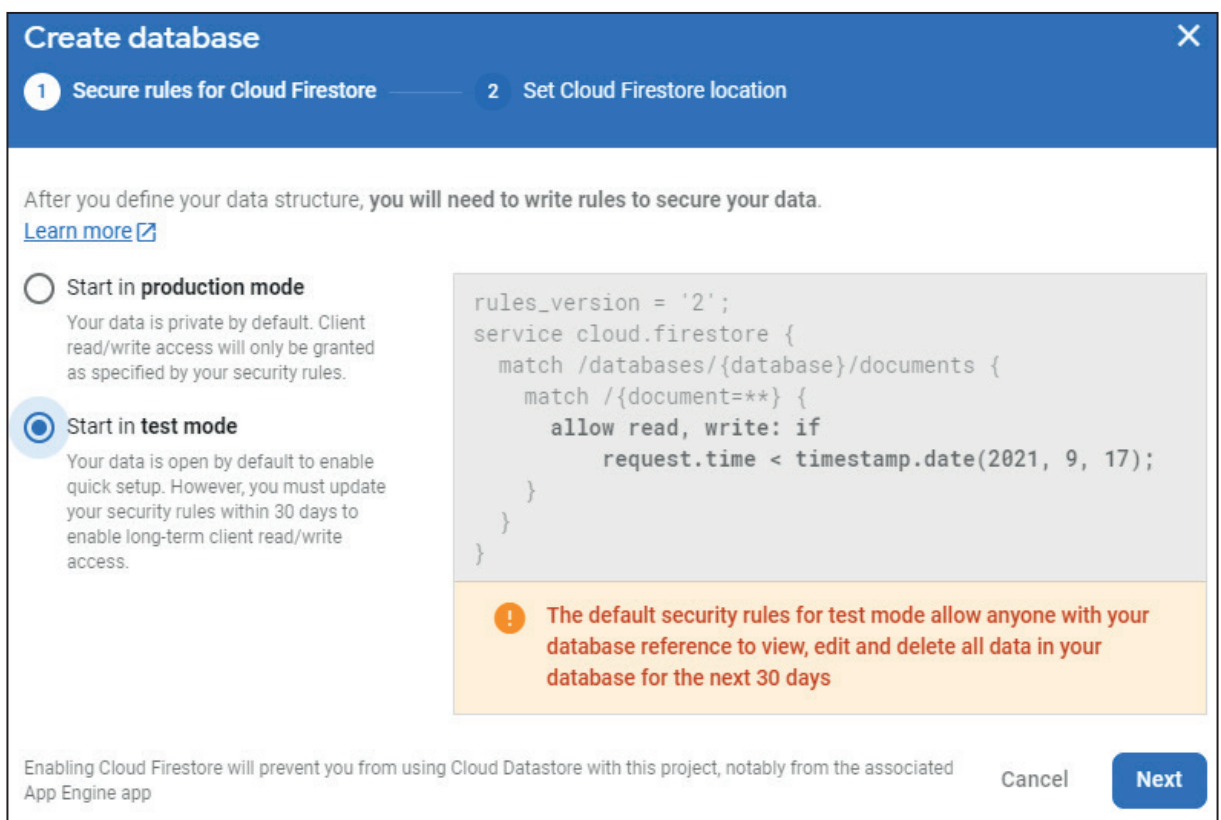
The document value can be auto and a random value or you can enter the values that you want, but each document value must be a unique value in the same collection.

Now, you will continue using your Android project (**Lesson10\_authentication**) as used previously. In the following part, you will create a **cloud Firestore database** using your **Firestore** web account, then retrieve and use the content of this database in your app . To create and retrieve or make a query for your Firestore database, perform the following steps:

55- At your Firestore web site, and as is illustrated in the figure below, click **Firestore Database** on the left panel.

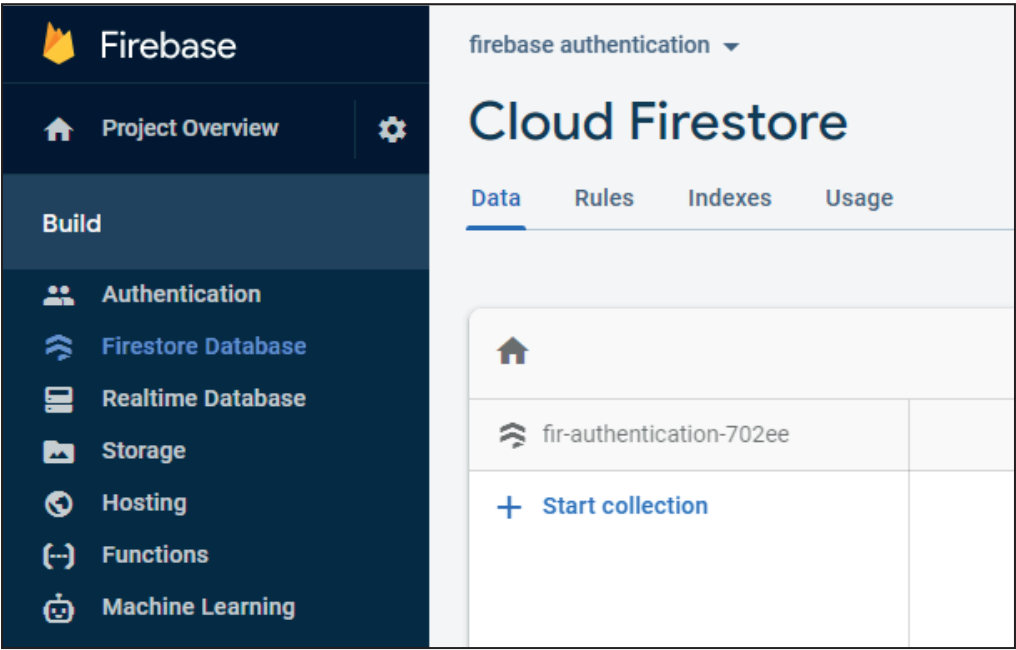


56- Click **Create database** button, select the : **Start in test mode**, and click **Next**.



57- Google has more than one cloud server center (or data center) world wide. You should select the cloud Firebase location which is the nearest for your app users' locations because they will connect to this database. Then click **Enable**. You should wait a few seconds, then you should get the following:





58- Remember, you want to create a database structure at the cloud Firebase for the following:

Customer Id	Name	City
1	Alex	Toronto
2	Diana	London
3	Ahmad	Mexico
4	George	Madrid

The data base table name (Collection) is: **customers**. To create this collection click **Start collection**, and as is illustrated in the following figure, type **customers\_table** for the **Collection ID** and click **Next**.

Start a collection

1 Give the collection an ID

2 Add its first document

Parent Path

/

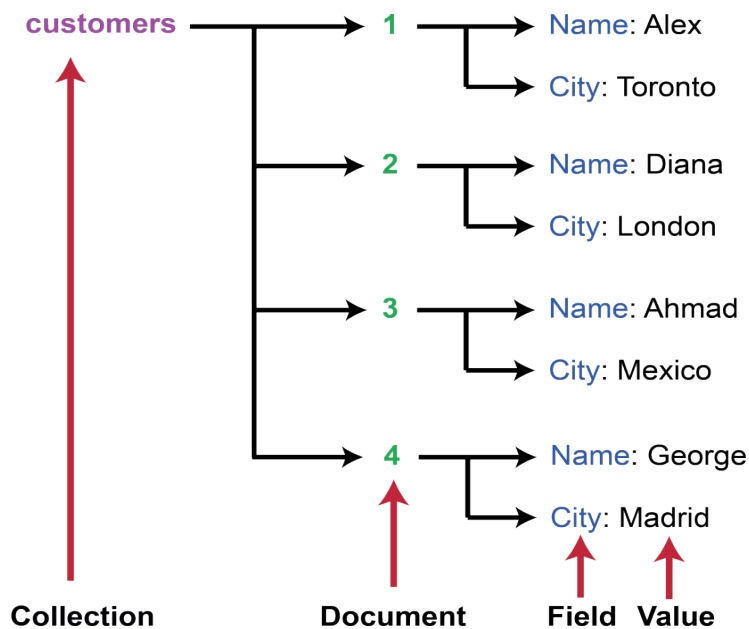
Collection ID

customers\_table

Cancel

Next

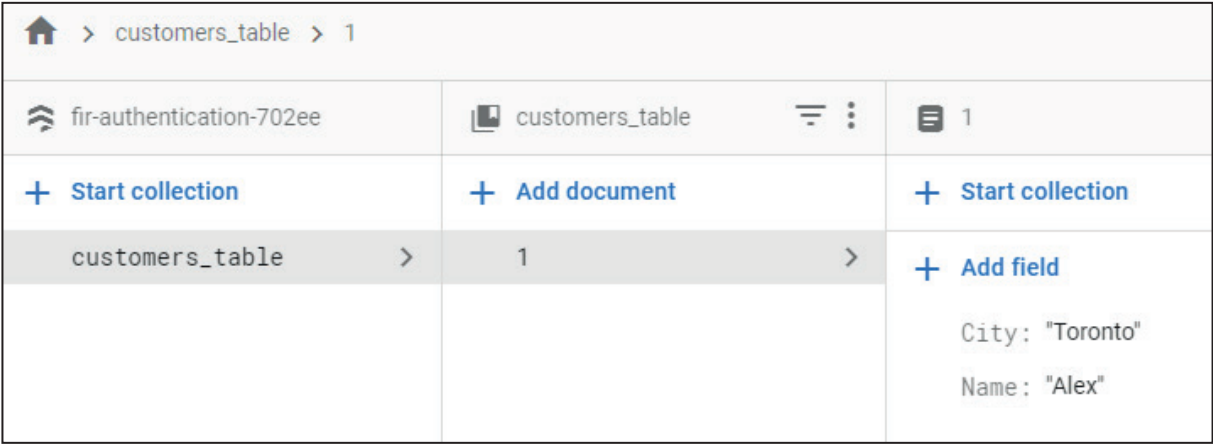
59- Now, you should add to your cloud Firestore collection (customers) its **Documents** one by one to create the following data structure:



For the first document, type **Document ID: 1**, **Name** for the first field, **Alex** for the first field value, then click the plus sign to add the second field **City** type **Toronto** for the second field value, then click **Save** as illustrated in the following figure:

The screenshot shows the 'Start a collection' dialog box. The first step, 'Give the collection an ID', is completed. The second step, 'Add its first document', is active. The 'Document parent path' is set to `/customers_table`. The 'Document ID' is set to `1`. Below this, there are two field entries. The first entry has 'Name' as the field, 'string' as the type, and 'Alex' as the value. The second entry has 'City' as the field, 'string' as the type, and 'Toronto' as the value. A plus sign (+) is visible at the bottom left of the field list, indicating that more fields can be added. At the bottom right, there are 'Cancel' and 'Save' buttons.

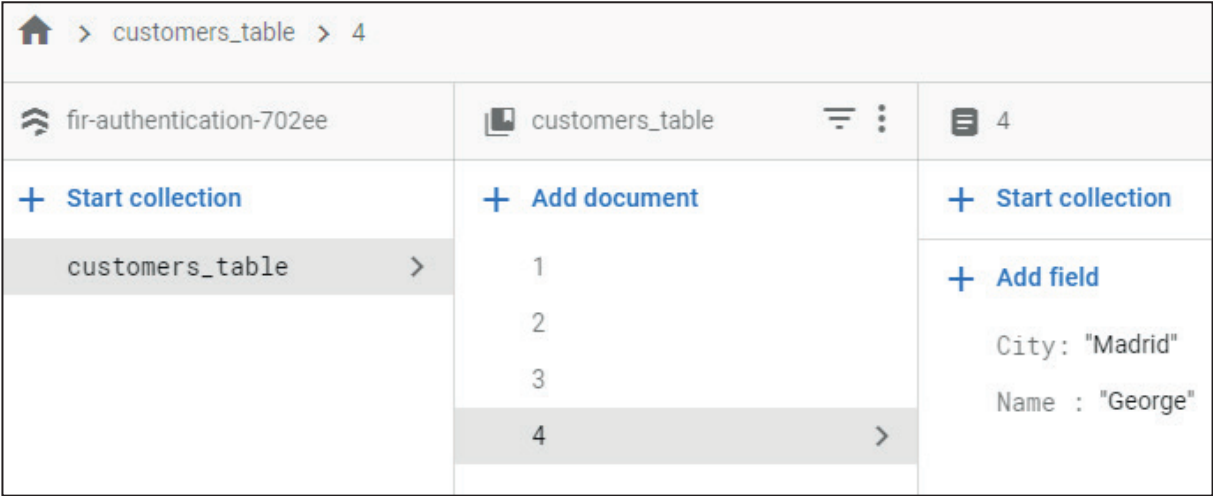
60- You should get the following:



Click: **+Add document** to repeat the previous step three more times to add the remaining 3 Documents as illustrated in the following table:

Document ID	1st Filed	2nd Filed
2	Name : Diana	City : London
3	Name : Ahmad	City : Mexico
4	Name : George	City : Madrid

When you finish, you should have the following figure:



If you click the **Rules** tab, you will get the security configuration below. Because you selected **test mode** for your database security settings, and as is illustrated in the following figure your app users have allow read and write permission for one month of your database creation date. Then, you must configure your security rules.

```
1 rules_version = '2';
2 service cloud.firestore {
3     match /databases/{database}/documents {
4         match /{document=**} {
5             allow read, write: if
6                 request.time < timestamp.date(2021, 9, 18);
7         }
8     }
9 }
```

You have already created a cloud Firebase database at Firebase web server; however, this database has been created manually. You should configure your app to create this database automatically when the app user sends or creates his profile information. This what you will do in the lab of this lesson.