

Lab 10:

Firestore Authentication and Database

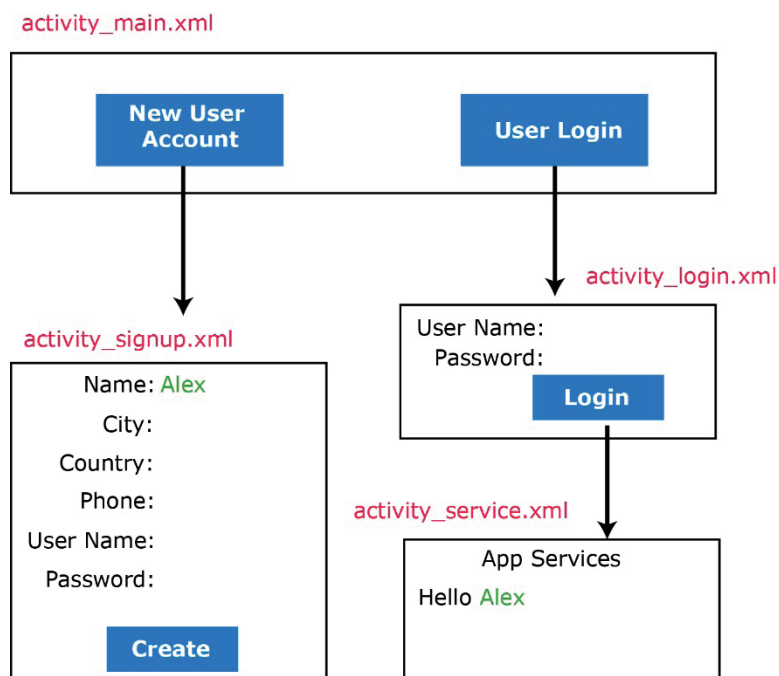
- **Configure your App to use Firestore Services**
- **Adding Firestore to your Android App**
- **Configuring User Authentication Using Firestore Authentication**
- **Creating a Firestore Cloud Database**
- **Retrieving Data | Firestore Cloud Database**

Lab Scenario:

In this lab, you will create an Android app that will allow the app user to create his/her account (user name & password) to access an app service. You will create an authentication procedure depending on the Firestore authentication service. In addition, within creating the app user's username and password, the app user will create his/her profile at Firestore database, then in a specific app activity you can retrieve this app user's profile information into the app interface.

The following diagram displays the app activities which you will create in this app. You will add two buttons (create a user account & user login) in your **activity_main.xml** file. When your app user clicks the **Create a User Account** button, the app user will move to the **activity_signup.xml** file which includes a form asking the app user to fill out his/her profile information, and his/her username and password. The app user's username and password will be sent to the Firestore authentication users accounts; however, the other app user's information will be sent to the Firestore database.

Also, when the app user taps the User Login, he/she will move to the **activity_login.xml**, if the user enters a valid user name and password, he/she will login to the **activity_service.xml**, and get a welcome message including his name, which is retrieved from the Firestore database.



To achieve the lab scenario, perform the following steps:

1- Open Android Studio, and then click **File → New → New Project**

2- Select **Empty Activity**, and click **Next**

3- Type: **Lab10** for the application name, then click **Finish**.

4- Before starting with typing the Kotlin code in your app, check the **build.gradle (Module: Lab10.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  }

```

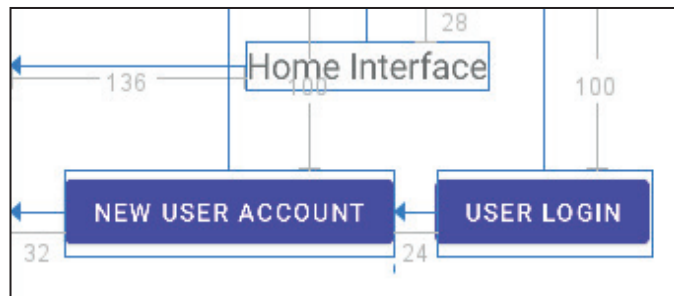
5- Open the **activity_main.xml** file in Design mode, and then **delete** the “Hello World!” text.

6- Add three new activities by right clicking **layout → New → Activity → Empty Activity**.

Type: **Signup** for the **Activity Name**, then click **Finish**.

7- Repeat the previous step to add the other two activities: **Login** and **Service**.

8- You should design the **activity_main.xml** interface with the following design:



To do that, open the **activity_main.xml** in the Design mode, add a **TextView** widget from the **Palette** panel, set its constraints, and set its attributes values as follows:

text: Home Interface	textSize: 20sp
-----------------------------	-----------------------

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

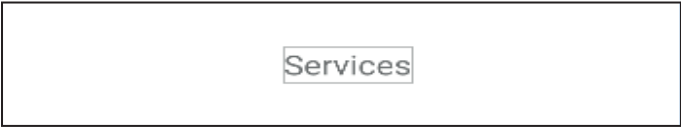
id: signupBtn	text: New User Account
----------------------	-------------------------------

id: loginBtn	text: User Login
---------------------	-------------------------

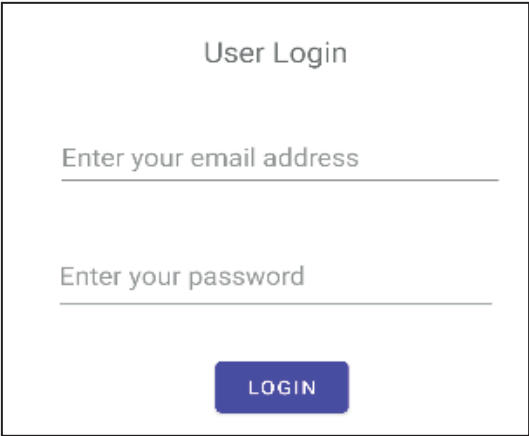
9- Open the **activity_service.xml** file in the **Design** mode, add a **TextView** widget, set its constraints, and set its attributes values as follows.

text: Services	textSize: 20sp
-----------------------	-----------------------

Your **activity_service.xml** interface should look like the following design.



10- Now, open the **activity_login.xml** file in the Design mode. Your login activity should have the following design:



As you see in the figure above, there is no need to add a **TextView** widget before the **EditText** widget which is assigned to enter the app users email or the password. It is enough to add a **hint** attribute to this **EditText** widget to display for what this **EditText** widget is assigned for. This will give you more space on the width of the mobile screen. To design this login interface, add a **TextView** widget to the top of this activity, set its constraints, and set its attributes values as follows:

text: User Login	textSize: 20sp
-------------------------	-----------------------

Add two **Plain Text** widgets, set their constrains as illustrated in the previous figure, and modify the **id** attribute value for each **Plain Text** (EditText) widget as follows:

id: emailLoginId	id: passwordLoginId
-------------------------	----------------------------

11- Open the **strings.xml** file (**app** → **res** → **values** → **strings.xml**) and add the configuration below.

```
<resources>
    <string name="app_name">Lab10</string>
    <string name="email_address">Enter your email address</string>
    <string name="password">Enter your password</string>
</resources>
```

These texts (**email_address** and **password**) are the hint sentences which you will add in the **EditText** fields of the Login interface. You may ask, why it is better to configure each text in this **strings.xml** file? I can add a text that I need directly as text within double quotations. The answer is, you are right, you may add the text in your app without the need to configure it in your app **strings.xml** before use; however, if any of your app users use a build-in translator software, this app user translator software can't translate these texts. Also, configuring your text in the **strings.xml** file makes your app more dynamic for the users who use auto reader software.

12- Now, to add the hint texts to your login interface, open the **activity_login.xml** file again in the **Code** mode, and add only the gray highlighted attributes in the code below:

```
<EditText
    android:id="@+id/emailLoginId"
    android:hint="@string/email_address"
    android:layout_width="297dp"
    android:layout_height="50dp"
    android:layout_marginTop="112dp"
    android:layout_marginEnd="52dp"
    android:ems="10"
    android:inputType="textPersonName"
    android:minHeight="48dp"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintTop_toTopOf="parent"/>

<EditText
    android:id="@+id/passwordLoginId"
    android:hint="@string/password"
    android:layout_width="295dp"
    android:layout_height="58dp"
    android:layout_marginTop="192dp"
    android:layout_marginEnd="56dp"
    android:ems="10"
    android:inputType="textPersonName"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    tools:ignore="SpeakableTextPresentCheck" />
```

13- Open the **activity_login.xml** in the **Design** mode, add a Button widget, set its constraints, and set its attributes as follows:

id: loginBtn	text: Login
---------------------	--------------------

Your **activity_login.xml** file should have the following design:

User Login

Enter your email address

Enter your password

LOGIN

14- Open the **activity_signup.xml** file in the **Design** mode, and add **6 Plain Text** widgets using the drag and drop technique. Set their constraints to have the following design:

Name

Name

Name

Name

Name

Name

Set the **id** attribute values for these **EditText** fields as follows:

id: nameId	id: cityId	id: countryId	id: phoneId	id: emailSignupId	id: passwordSignupId
-------------------	-------------------	----------------------	--------------------	--------------------------	-----------------------------

15- Open the **strings.xml** file (**app** → **res** → **values** → **strings.xml**) and add the last four lines of the XML code below:

```
<resources>
    <string name="app_name">Lab10</string>
    <string name="email_address">Enter your email address</string>
    <string name="password">Enter your password</string>
    <string name="name">Enter your full name</string>
    <string name="city">Enter your city</string>
    <string name="country">Enter your country</string>
    <string name="phone_number">Enter your phone number</string>
</resources>
```

16- Open the **activity_signup.xml** file in the **Design** mode, and delete the **text** attribute value for all these 6 **EditText** tags (keep the **text** attribute value empty).

17- Open the **activity_signup.xml** file in the **Code** mode, and add the **hint** attribute values below to the corresponding **EditText** tags:

<code>android:hint="@string/name"</code>	<code>android:hint="@string/city"</code>
<code>android:hint="@string/country"</code>	<code>android:hint="@string/phone_number"</code>
<code>android:hint="@string/email_address"</code>	<code>android:hint="@string/password"</code>

You should have the following design for your Signup interface:

18- Open the **activity_signup.xml** file in the **Design** mode, add a Button widget, set its constraints, and set its attributes values as follows:

id: createBtn	text: Create
---------------	--------------

The Signup interface should have the following design:

19- Open the **MainActivity** file, and use the Intent class to configure this interface where when the app user taps the: **New User Account** button (**signupBtn**), he/she will move to the **Signup** activity. Also, when the app user taps the **User Login** button (**loginBtn**), he/she will move 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**

20- **Run** your app just to be sure if the navigation from the **MainActivity** to other activities (**Login** or **Signup**) using the navigation buttons is working fine.

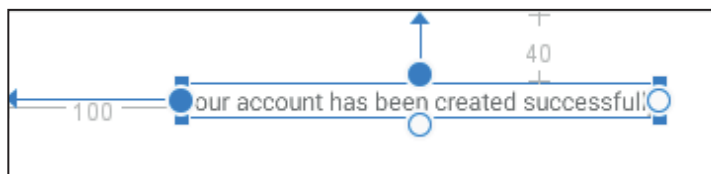
21- You should add a new activity including a thank you message only. The app user can go to the **Signup** interface, enter his/her profile information, username and password, then tap the **Create** button, this app user should move to an interface that displays his/her account which has now been created successfully.

Now, you should create this thank you activity which including a thank you message. To do that, right clicking **layout** → **New** → **Activity** → **Empty Activity**

Type: **Thankyou** for the activity name, then click **Finish**.

22- Open the **activity_thankyou.xml** file in the **Design** mode, add a **TextView** widget, set its constraints, and set its **text** attribute value to: **Your account has been created successfully**

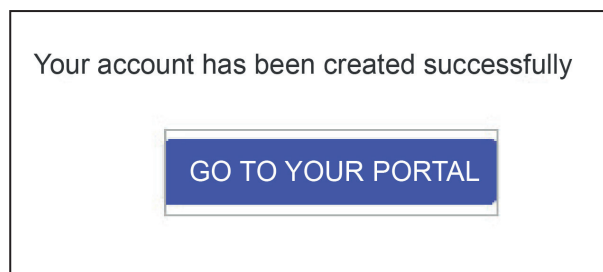
This interface should have the following design:



Also, add a **Button** widget below the text directly, set its constraints, and set its attributes values as follows:

id: portalBtn	text: Go to your portal
----------------------	--------------------------------

The interface design of this activity_thankyou.xml should be as follows:



23- You should configure your **Thankyou** Kotlin file in a way when your app user taps the button: **GO TO YOUR PORTAL**, he/she will move to the **Login** activity. To do this, open the **Thankyou** Kotlin file, and add the following configuration:


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

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

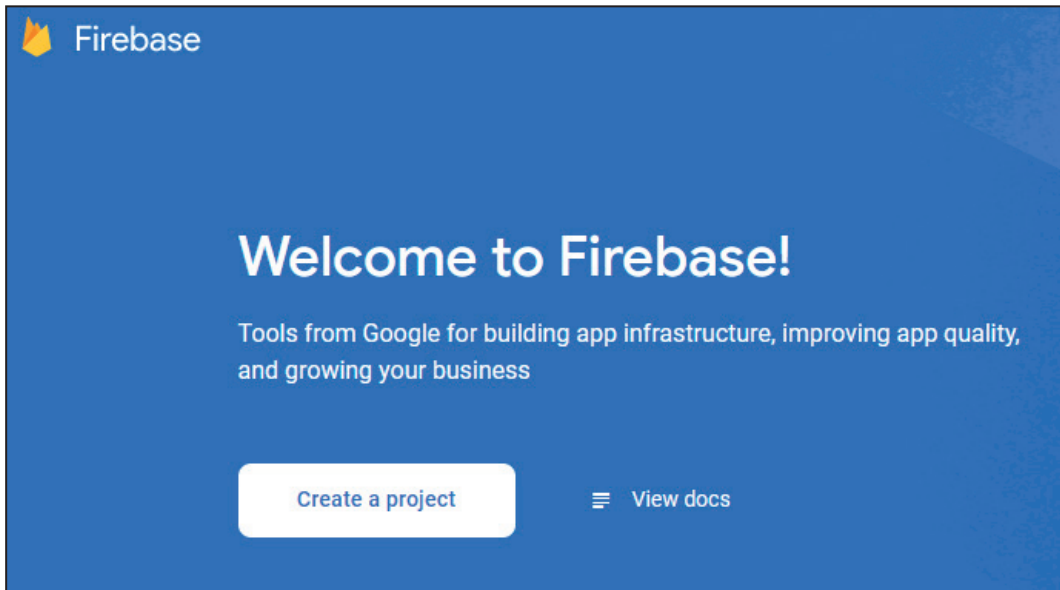
In the code above, double click `portalBtn`, click the red pop-up lamp, and click **Import**

Configure your App to use Firebase Services

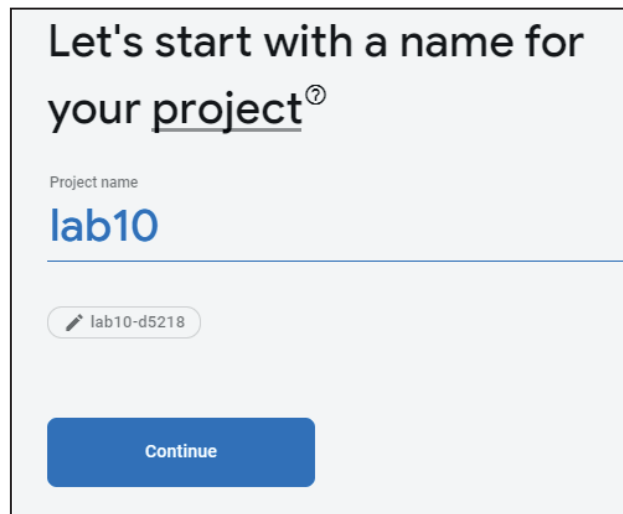
Now your Android app is ready for the Firebase authentication and database configuration steps.

24- First, your app should have an account at Firebase web server. To create an account at Firebase, Go to: <https://console.firebase.google.com>, sign into Firebase using your Google account (Gmail).

You will get the following web page:



25- Click **Create a project**. Fill out your project name "lab10" 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?

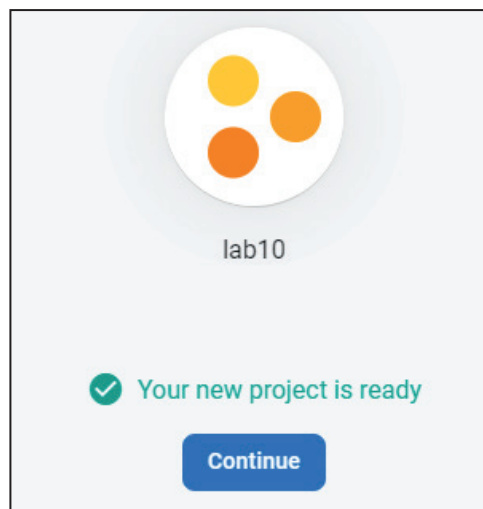
Project name

lab10

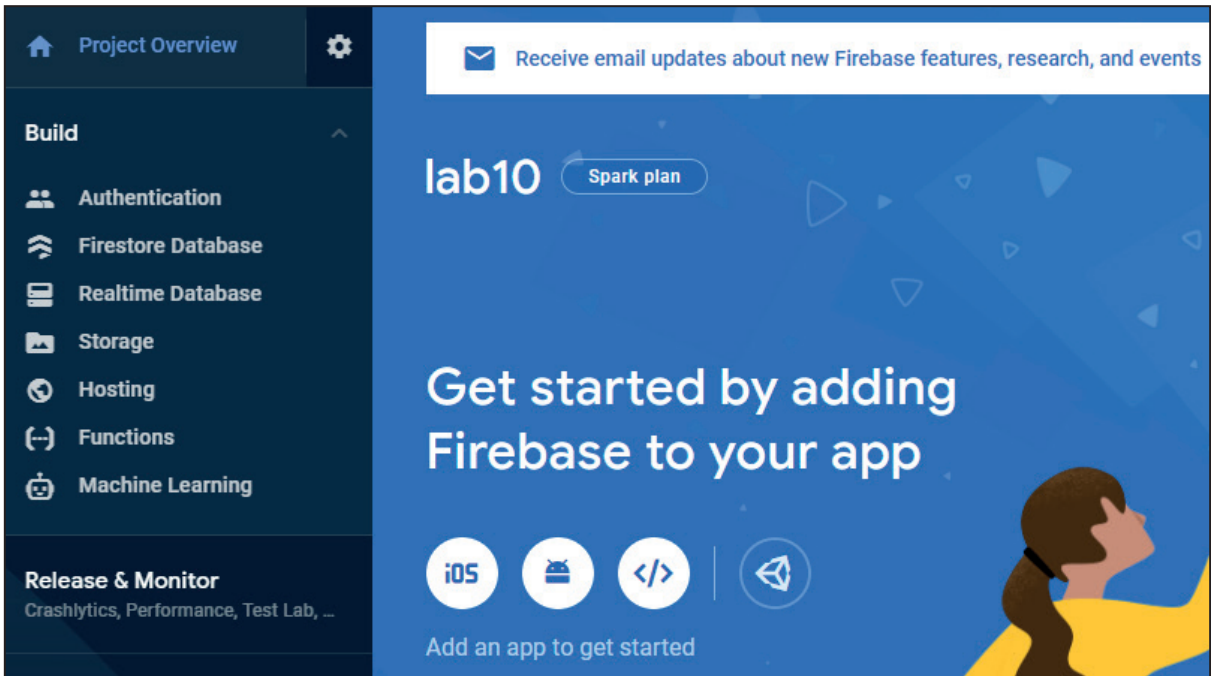
lab10-d5218

Continue

26- Click **Enable Google Analytics for this project** to disable this feature. Click **Create Project** button. Then, after a few 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**

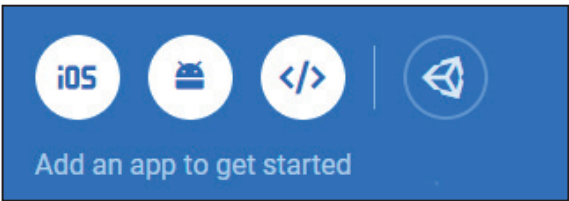


Now, as you see in the figure below, your project name: **lab10** is created at the Firebase web server.



Adding Firebase to your Android App

27- Click the **Android icon** to configure Firebase with your Android app.



You should get the following dialog box to register your app. Your project name must be a unique name at Google Play store. To get your app name, open the **MainActivity** file, the package name which exists at the first line of Kotlin code is your Android project name. The first line of the Kotlin code is illustrated in the code below,

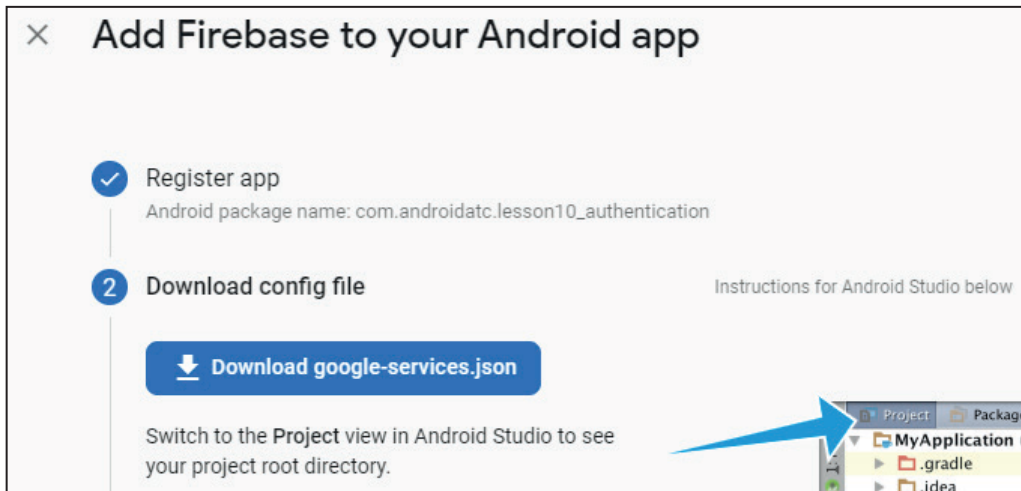
```
package com.androidatc.lab10
```

Enter your project name as illustrated in the figure below, and click **Register app**

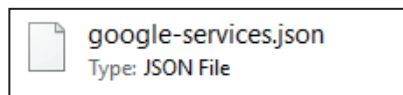


28- In this step, as illustrated in the following figure, you should click:

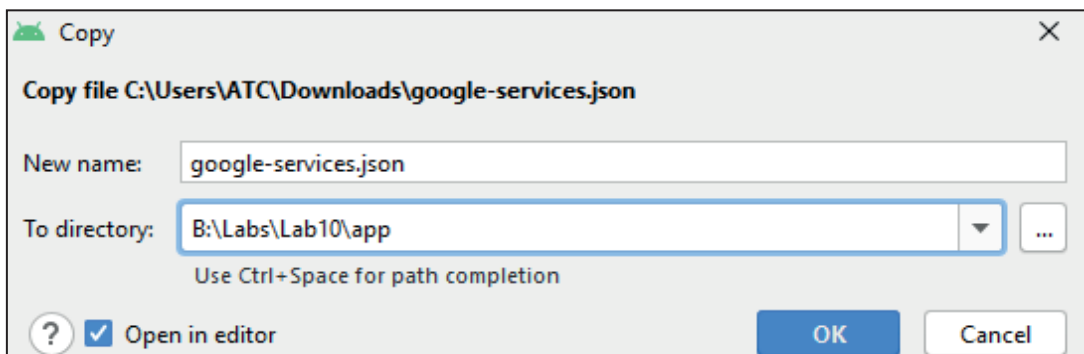
Download google-service.json button to download the JSON configuration file. This configuration file includes your **https** connection settings between your Android app and the Firebase services. If you download this file many times, you should use the file which has exactly this name: **google-services.json**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



29- Open your download folder in your web browser. 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**



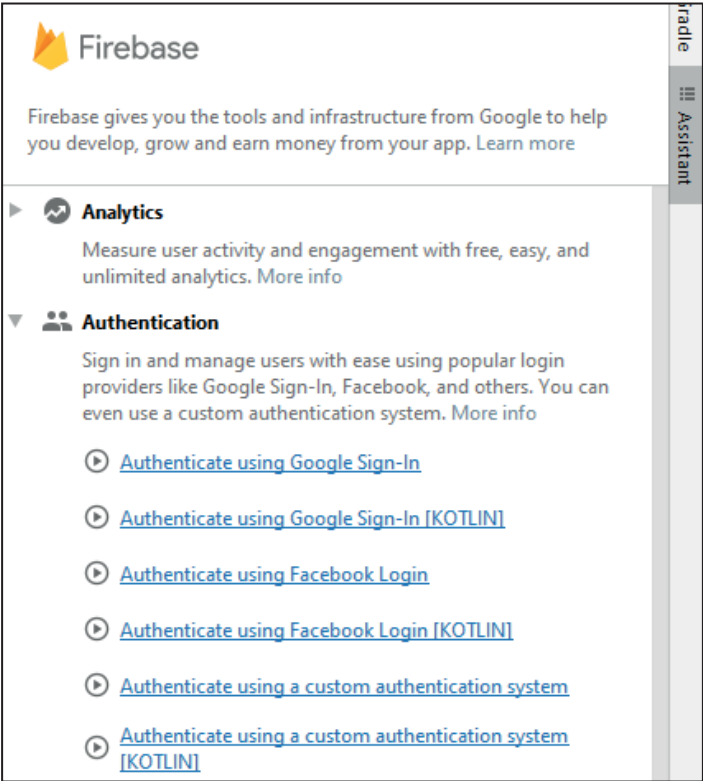
30- Now, return to the Firebase web site to complete the setup steps. Click **Next**.

In this step no need to follow the traditional method to configure your app **build.gradle** file or project-level build.gradle by adding the dependency libraries or classes which are required for your app to make your app SDK compatible with Firebase. You will add the Firebase SDK configuration in a different and easier way as you will see in the next step.

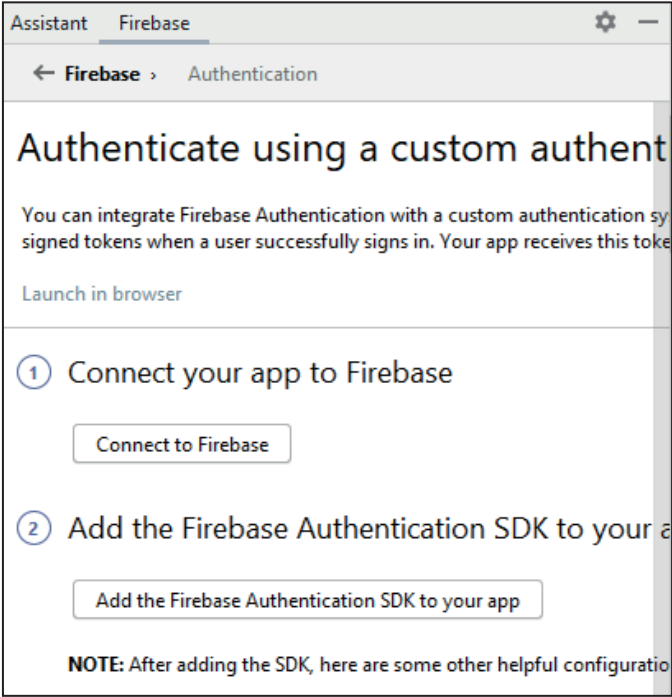
Click **Next**, then click **Continue to console**

31- Back to your Android Studio. To configure the Firebase authentication in your Android app, click **Tools → Firebase**

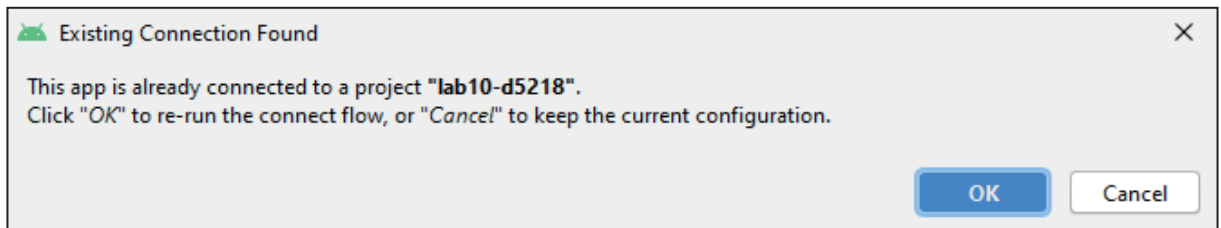
As illustrated in the figure below, click **Authentication**, then click **Authentication using a custom authentication system [Kotlin]**



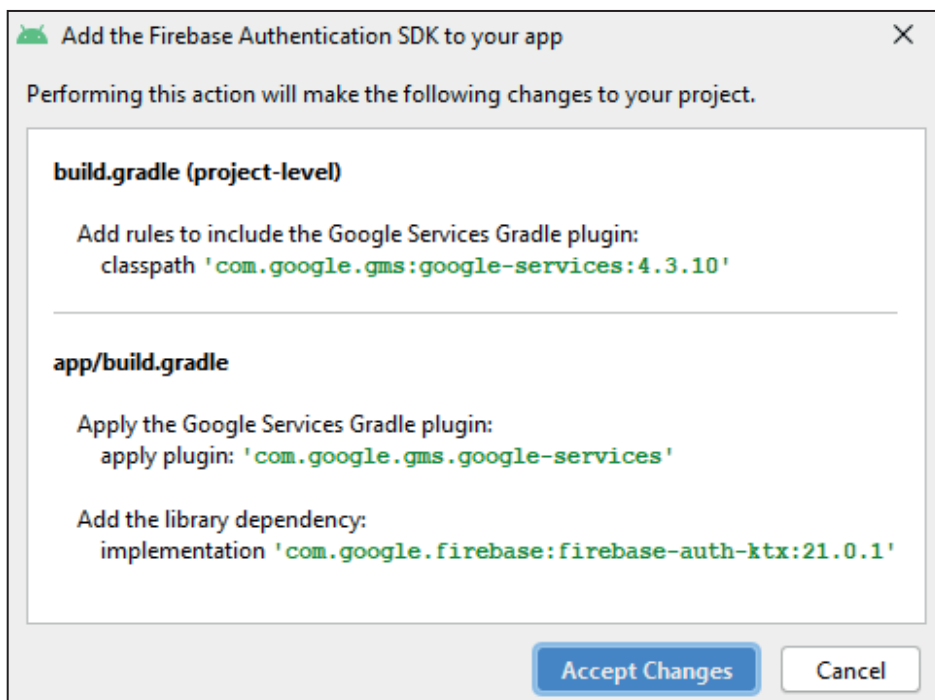
32- In the configuration wizard below, click Connect Firebase button.



Because your app is still connected, you should get a dialog box similar to the following. Click Cancel.

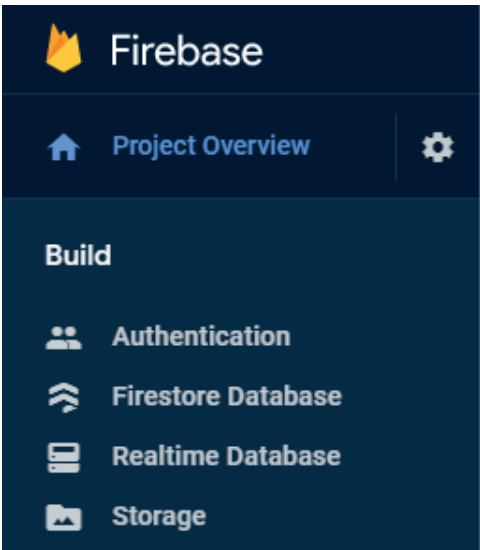


33- Click Add the Firebase Authentication SDK to your app. You should get the following dialog box which displays the dependency configurations and plugins which will be added to your app to make its SDK compatible to work with Firebase authentication service. Click **Accept Changes**.

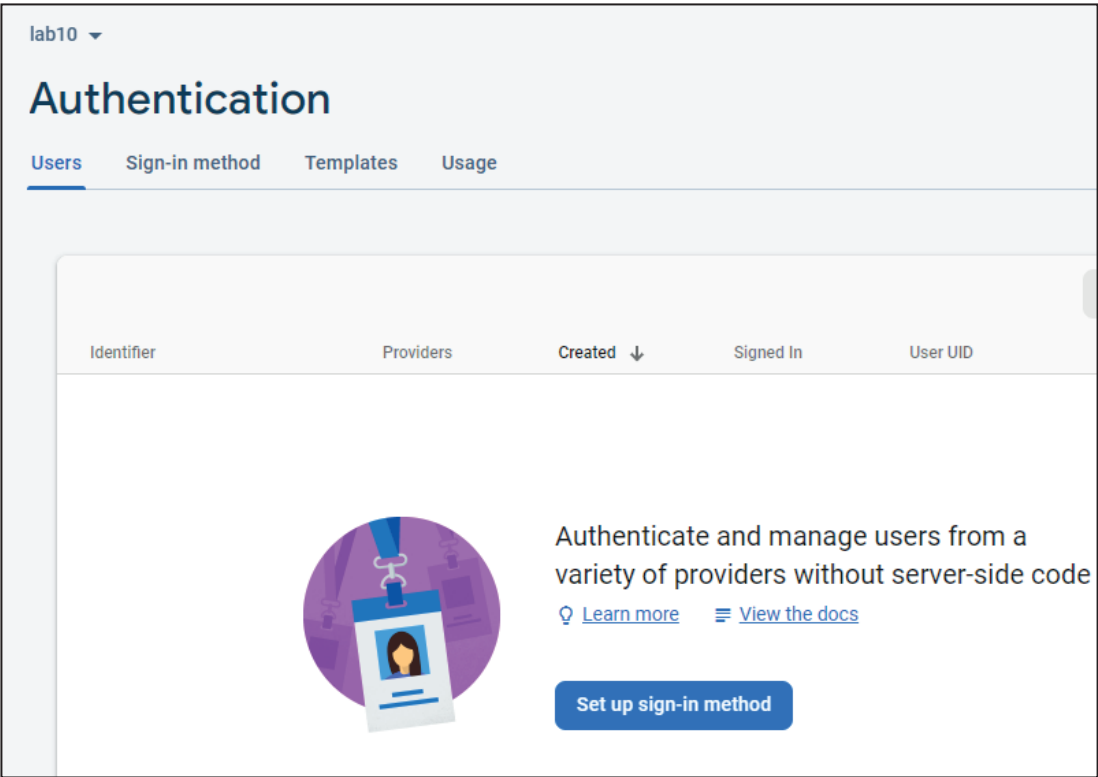


If you check the build.gradle file, you will find these configurations have been added successfully. As you saw, it's easier than adding them manually as we already did before in this lesson.

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



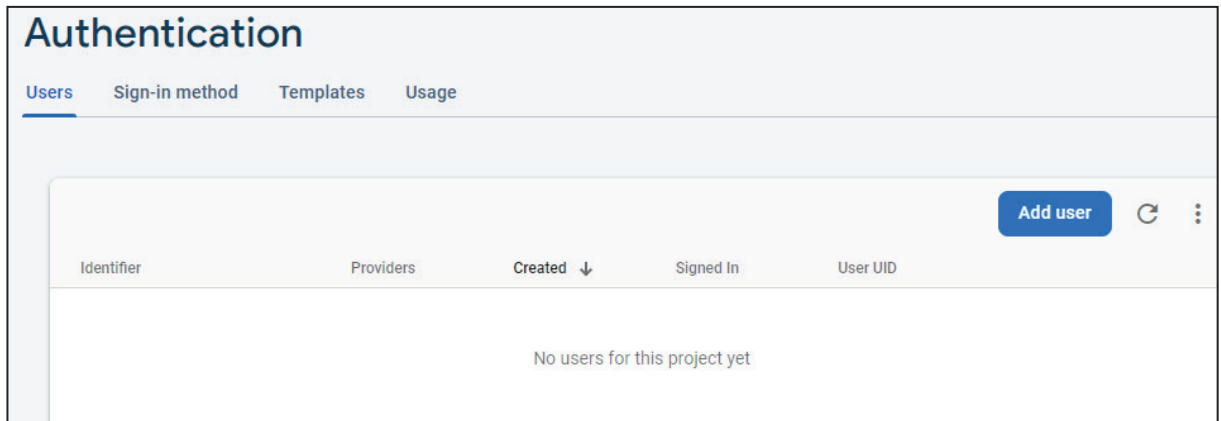
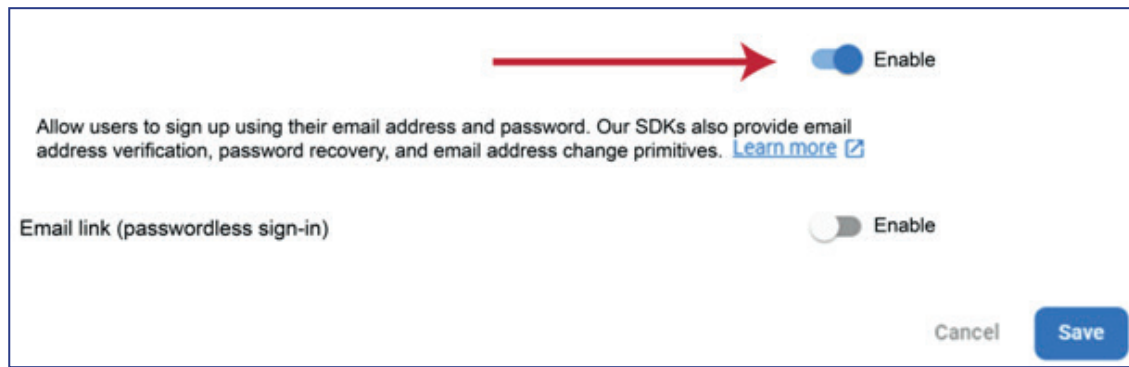
You should get the following authentication setup web page.



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

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

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



37- Back to your Android Studio. Before 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 illustrated in the following figure:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.androidatc.lab10">
    <uses-permission android:name="android.permission.INTERNET" />
```

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

39- In the same **Signup** Kotlin file, create the **createUser** function as 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, Thankyou::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**.

40- 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 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()
            }
        }

    }

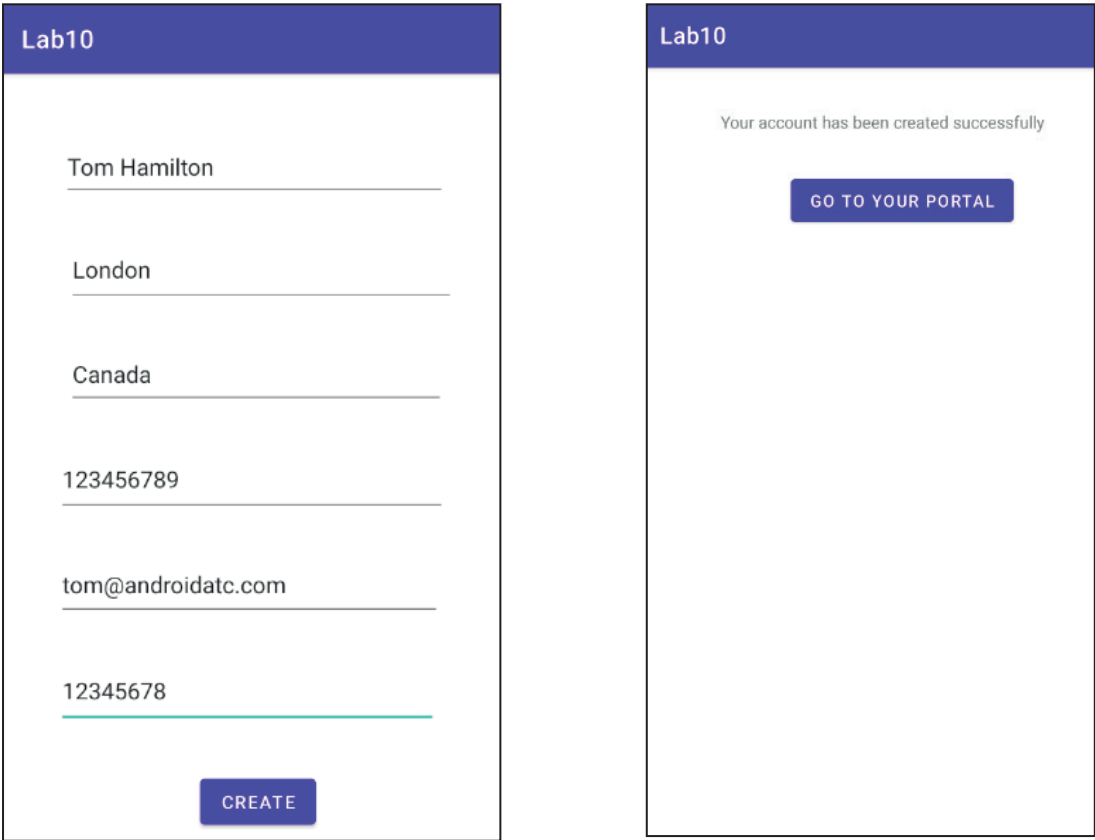
    fun createUser(email: String, password: String) {
        auth.createUserWithEmailAndPassword(email, password)
            .addOnCompleteListener(this) { task ->

                if (task.isSuccessful) {
                    startActivity(Intent(this, Thankyou::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**.

41- **Stop**, then **Run** your app. Tap the **New User Account** button. As illustrated in the following figures, fill out your profile data, username and password, then tap **CREATE** button. Your account should be created and the Intent class will move you directly to the **Thankyou** activity. Tap the **GO TO MY PORTAL** button to open the **Login** activity.



42- To be sure that your account (username and password) has been created successfully at the Firebase authentication (Users), go back to your Firebase web site, in the Authentication part, click the **Users** tab, and click **Refresh**. Your user account should look like the following:

Search by email address, phone number, or user UID					Add user	↺	⋮
Identifier	Providers	Created ↓	Signed In	User UID			
tom@androidatc.com	✉	Aug 22, 2021	Aug 22, 2021	d3CJ34E3I2OnjYTD9o2qn5sPBkk2			
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"`

Configuring User Authentication Using Firebase Authentication

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 logs in 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 is: **loginBtn**. Also, the **Plain Text** username id is: **emailLoginId**, and the **Plain Text** password id is: **passwordLoginId**

43- 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().isNotEmpty()
                || passwordLoginId.text.trim().toString().isNotEmpty()) {

                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()
        }
    }
}

```

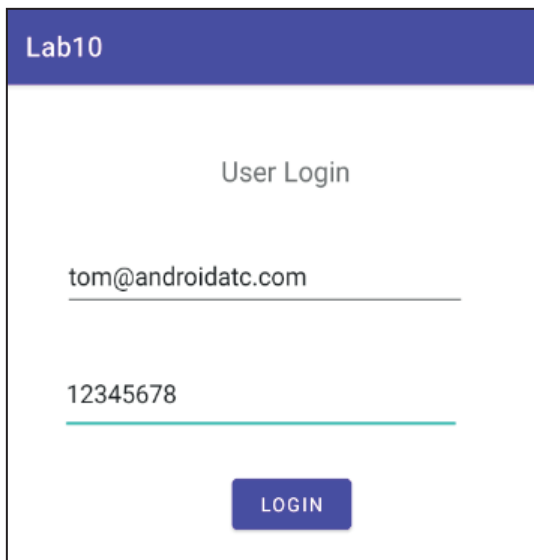
Double click **FirebaseAuth**, click the red pop-up lamp, and select **Import**.

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

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

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

44- **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 **LOGIN** button. You should login to the **Service** interface.

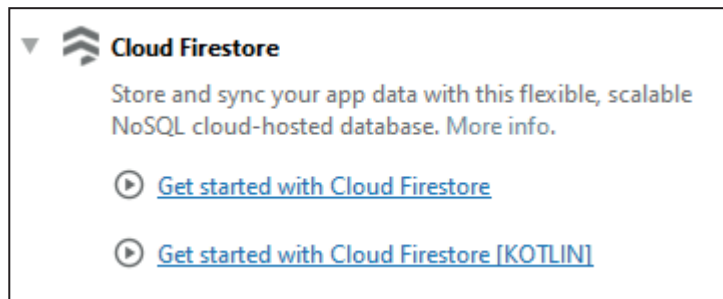


Creating a Firebase Cloud Database

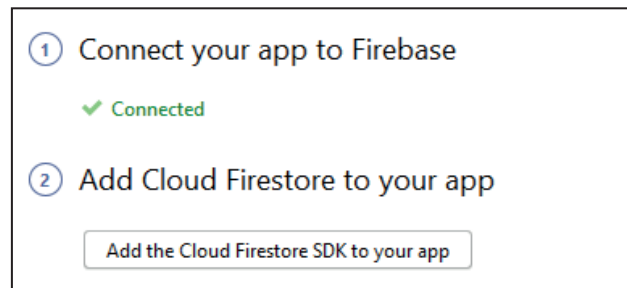
In the following steps, you will create a Firebase cloud database to save the app user information (name, city, country, and phone number).

45- To add the prerequisites plugins and dependency libraries for the Firebase cloud database for your app, click **Tools → Firebase**

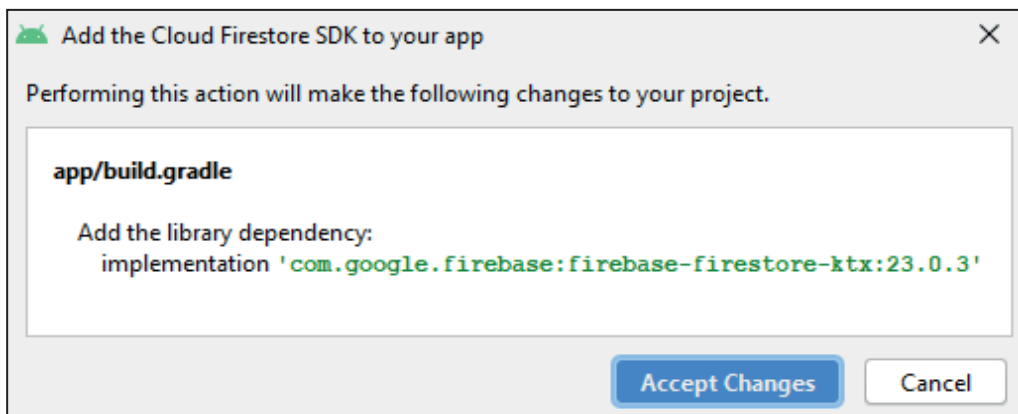
As illustrated in the figure below, click **Cloud Firestore**, then click **Get started with Cloud Firestore [Kotlin]**



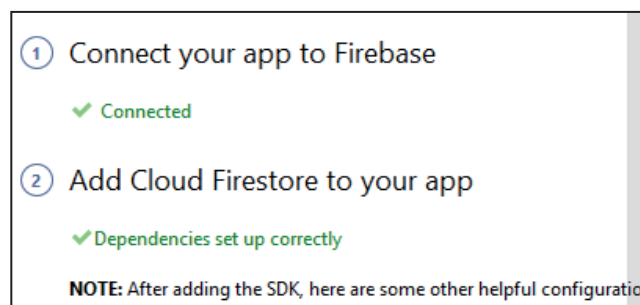
You should find that your app is already connected to Firebase as illustrated in the figure below:



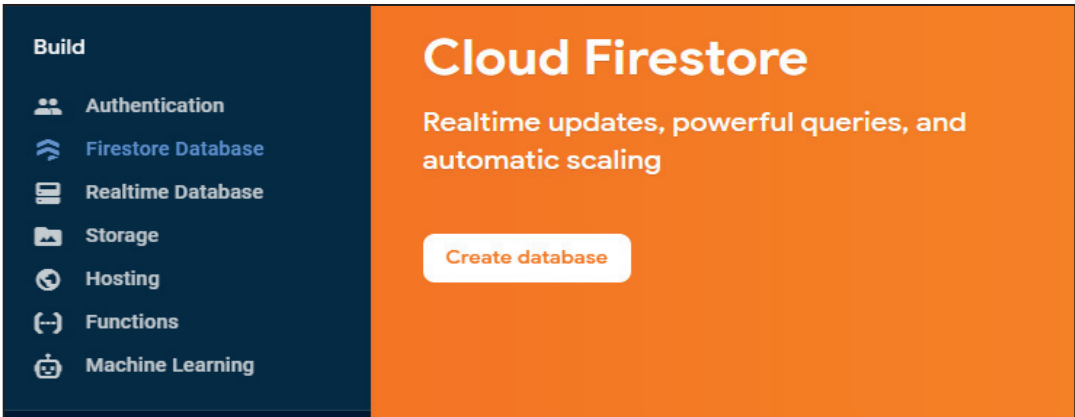
Then, click **Add the Cloud Firestore SDK to your app**, you should get the dialog box below. Click **Accept Changes**.



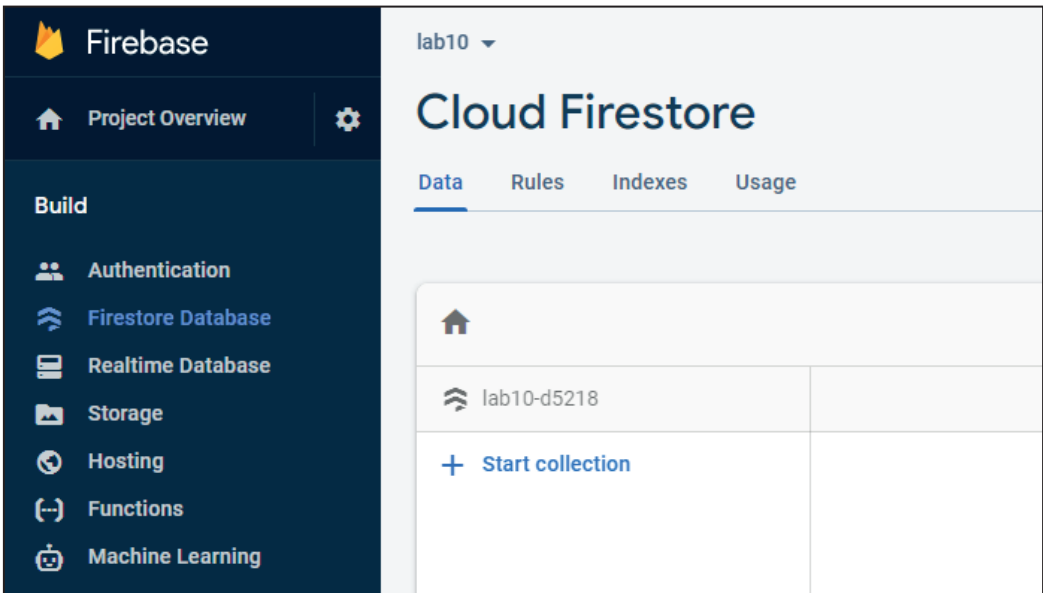
Then wait some seconds until your app files completes the synchronization process. Then you should get the following figure which displays that the dependencies' files are set up correctly. This means your app is ready to save data at the Firebase database.



46- Back to your Firebase web site. As illustrated in the following figure, click **Firestore Database**, then click **Create database** button



47- Select **Start in test mode**, click **Next**, then click **Enable**. You should have the following database (empty database).



If you click the **Rules** tab, you will find that your database security permissions has been configured with allow read and write permissions for all app users for one month only.

48- Now, back to your Android Studio, open your **Signup** Kotlin file, and add the gray highlighted code which is responsible to save the app user data (name, city, country, phone number) in the Firebase database and inside a **Collection** is called: **Customer**

```

class Signup : AppCompatActivity() {

    private lateinit var auth: FirebaseAuth

    val db = Firebase.firestore

    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())

                newCustomer()

            }
            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, Thankyou::class.java))
                }

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

```



```

    }
}

private fun newCustomer() {
    // Create a new customer
    val customer = hashMapOf(
        "Name" to nameId.text.toString().trim(),
        "City" to cityId.text.toString().trim(),
        "Country" to countryId.text.toString().trim(),
        "Phone Number" to phoneId.text.toString().trim()
    )
    // Add a new document with a generated ID.
    db.collection("Customer")
        .add(customer)
        .addOnSuccessListener { documentReference ->
            Log.d(TAG, "DocumentSnapshot added with ID:
${documentReference.id}") }
        .addOnFailureListener { e ->
            Log.w(TAG, "Error adding document", e)
        }
}
}
}

```

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

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

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

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

49- Stop, then Run your app. Tap **NEW USER ACCOUNT** button, fill out the app user information as illustrated in the figure below, then tap **Create** button. Your app should create this user account for authentication and at the same time, this user's name, city, country, and phone number will be saved at the Firestore database in a collection called Customer.

Lab10

Alex Alex

Mississauga

Canada

99998888

alex@androidatc.com

123456789

CREATE

Lab10

Your account has been created successfully

GO TO YOUR PORTAL

The following figure displays the user’s data which has been created or saved at the Firebase database:

<div><div>🏠 > Customer > wMmgVbxsNbP...</div></div>		
<div><div>lab10-d5218</div><div>+ Start collection</div></div>	<div><div>Customer</div><div>+ Add document</div></div>	<div><div>wMmgVbxsNbP5K2q2VjWP</div><div>+ Start collection</div></div>
Customer >	wMmgVbxsNbP5K2q2VjWP >	<div><div>+ Add field</div><div>City: "Mississauga"</div><div>Country: "Canada"</div><div>Name: "Alex Alex"</div><div>Phone Number: "99998888"</div></div>

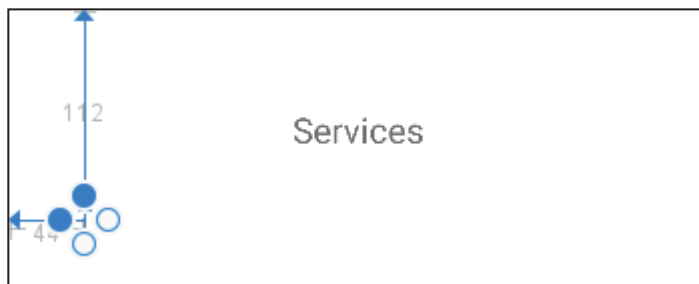
Retrieving Data | Firestore Cloud Database

To retrieve the content of a specific field of your Firestore database such as countries (Country) in your **Service** activity, follow the following steps:

50- Open **activity_service.xml** file in the Design mode, add a **TextView** widget, set its constraints, delete its **text** attribute value (keep it empty value), and set its **id** attribute value as follows:

id: readCountryId

Your Service interface should look like the following:



51- Open the **Service** file, and add the following code:

```
class Service : AppCompatActivity() {

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

        readFirestoreData()

    }

    fun readFirestoreData() {
        val db = FirebaseFirestore.getInstance()

        // Customer: is your database collection name. It is case sensitive.

        db.collection("Customer")
            .get()
            .addOnCompleteListener {
                val result: StringBuffer = StringBuffer()
                if (it.isSuccessful) {
                    for (document in it.result!!) {
```

```
// for "Country" use the same spelling which you have used before in
//the Signup file.
// append("      "): to make a space between country values' results.

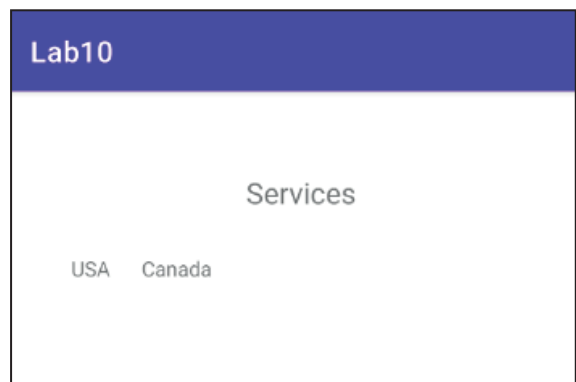
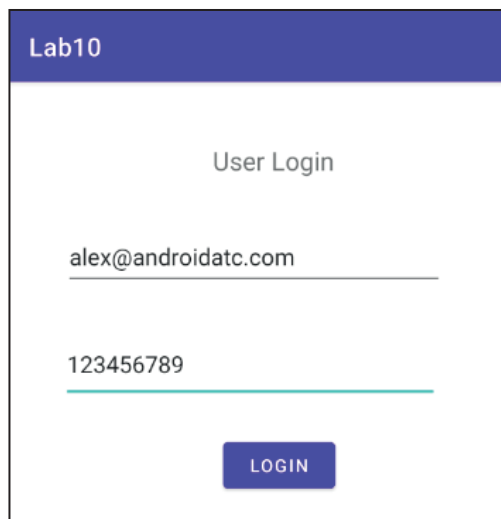
result.append(document.data.getValue("Country")).append("      ")

        }
        readCountryId.setText(result)
    }

}

}
```

52- **Stop**, then **Run** your app. Tap the **USER LOGIN** button, enter a username that already has a data at Firebase database, then tap the **LOGIN** button. The run result is as follows. Your result depends on the country values which you have in your "Country" field of your "Customer" collection.



At the end of this lesson, we want to share with you the following:

- Firebase topic has a lot of details, and one lesson is not enough to explain how you may use Firebase services in your Android apps. One of the interesting topics you may search about how you may use RecyclerView widget to display or retrieve texts and images in your app interface from a Firebase database. Also, there are a lot of amazing topics that you can search about when you have a free time.
- For more details about Firebase database users, database query, security, and other details, check the following web site: <https://firebase.google.com/docs/guides>
- Also, one of the best web sites which we recommend to find a lot of suggestions and solutions for your Android development is: <https://stackoverflow.com>