

Firebase Authentication (Kotlin)

In this lesson on Firebase, we will learn how to use Firebase Authentication API to store user data and allow them to login into our app.

Some services Firebase provides are:

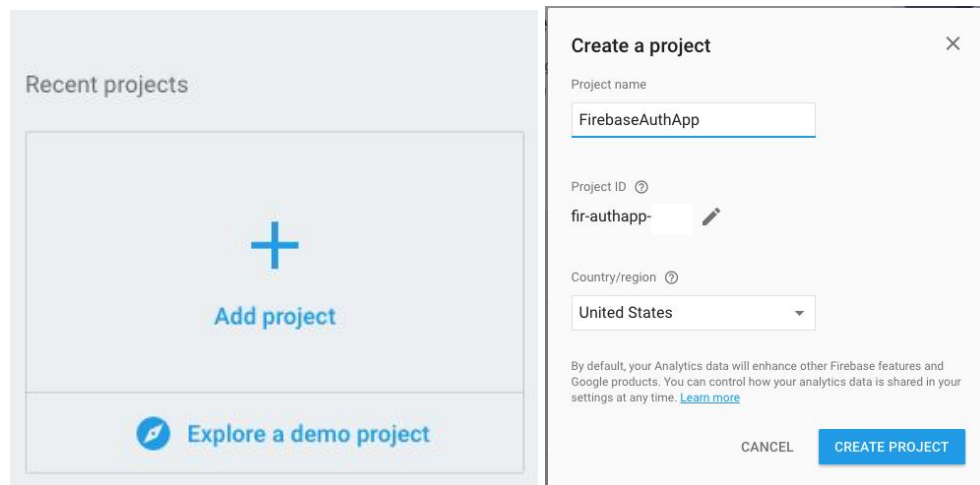
- Email and social app authentication like Facebook login
- Real Time Database
- Cloud Storage
- Analytics
- Notification Services

In this lesson with Firebase, we will:

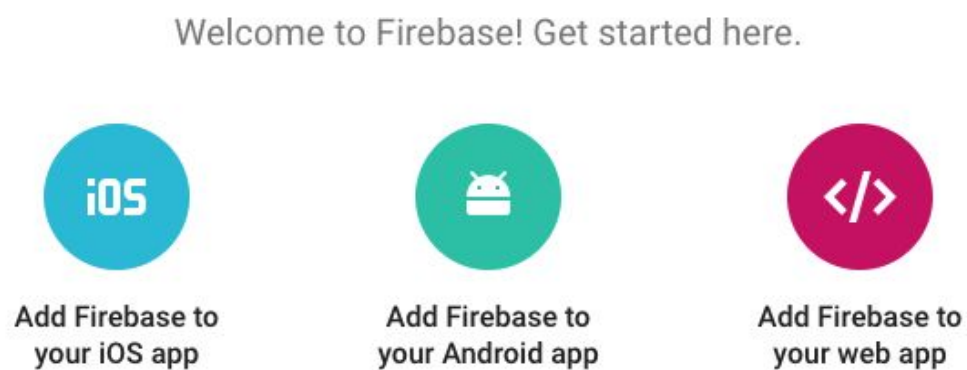
- Allow User registration
- User Login
- Retrieve User specific details from Firebase Database

While there are only few, you can have a look at them [here](#). We will create a demo with Firebase in this section. Visit [here](#).

Although, there is no need to visit Firebase console to integrate, we will still show it to you. Create a new project in Firebase:



Once you hit the the 'Create project' button, we're through with the initial process. Now, we can move on to adding Android specific details. Once we're on the project page, you can see:



Click on the Android option:

Add Firebase to your Android app

1

2

3

Register appDownload config fileAdd Firebase SDK

Android package name ?

mypackage

App nickname (optional) ?

Freemium Android App

Debug signing certificate SHA-1 (optional) ?

00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00

Required for Dynamic Links, Invites and Google Sign-In or phone number support in Auth. Edit SHA-1s in Settings.

CANCEL

REGISTER APP

in project **Firebase Intro**

Fill in your package details, click 'Register App':

Add Firebase to your Android app

1

2

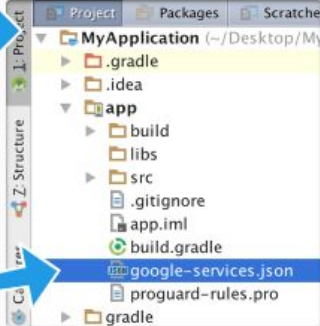
3

Register appDownload config fileAdd Firebase SDK

Android Studio instructions

Alternatives: [Unity](#) [C++](#)

- Download google-services.json
- Switch to the **Project** view in Android Studio to see your project root directory.
- Move the `google-services.json` file you have just downloaded into your Android app module root directory.



google-services.json

Already added the dependencies?
[Skip to the console](#)

CONTINUE

Download the JSON file inside the 'app' folder of your project and click 'Continue':

Add Firebase to your Android app

1

2

3

Register appDownload config fileAdd Firebase SDK

downloaded. Modify your build.gradle files to use the plugin.

- Project-level build.gradle** (<project>/build.gradle):

```
buildscript {
  dependencies {
    // Add this line
    classpath 'com.google.gms:google-services:3.1.0'
  }
}
```
- App-level build.gradle** (<project>/<app-module>/build.gradle):

```
...
// Add to the bottom of the file
apply plugin: 'com.google.gms.google-services'
```

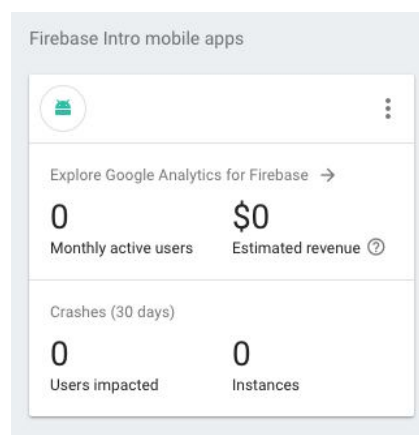
includes Analytics by default ?
- Finally, press "Sync now" in the bar that appears in the IDE:

Gradle files have changed since last syncSync now

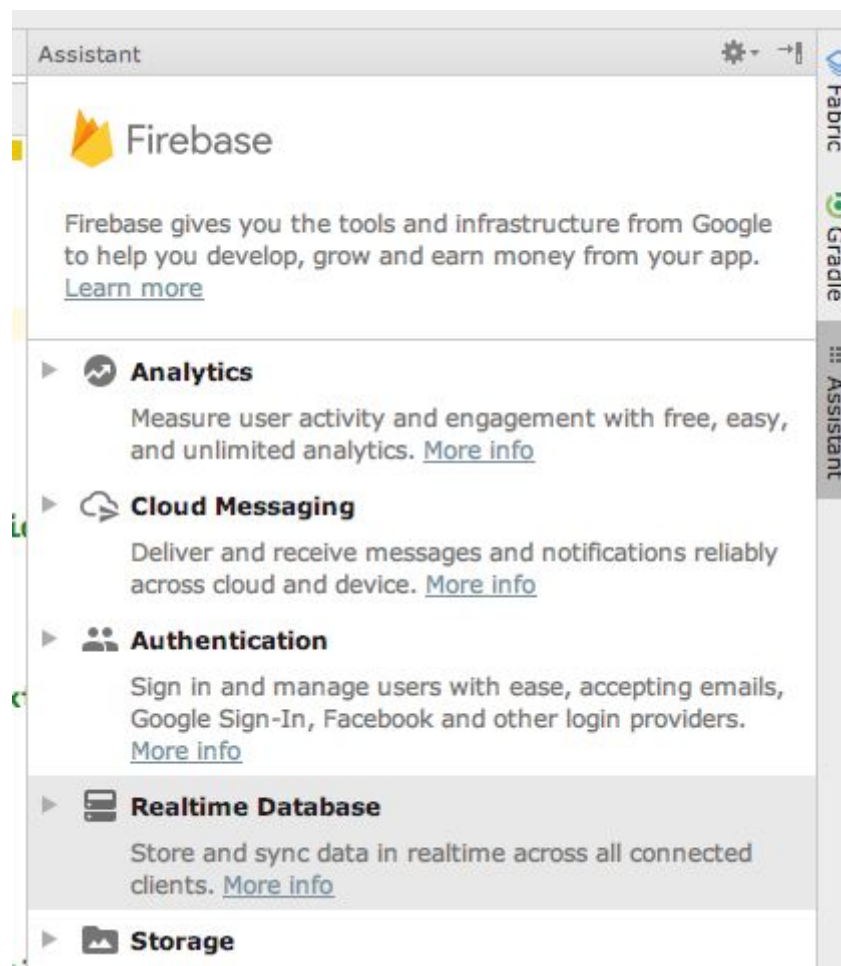
FINISH

Do the changes in gradle file as mentioned and click 'Finish'. Once this is done, sync the gradle file and wait for it to finish.

Once everything is done, you'll see the Android app appear in Firebase console as:



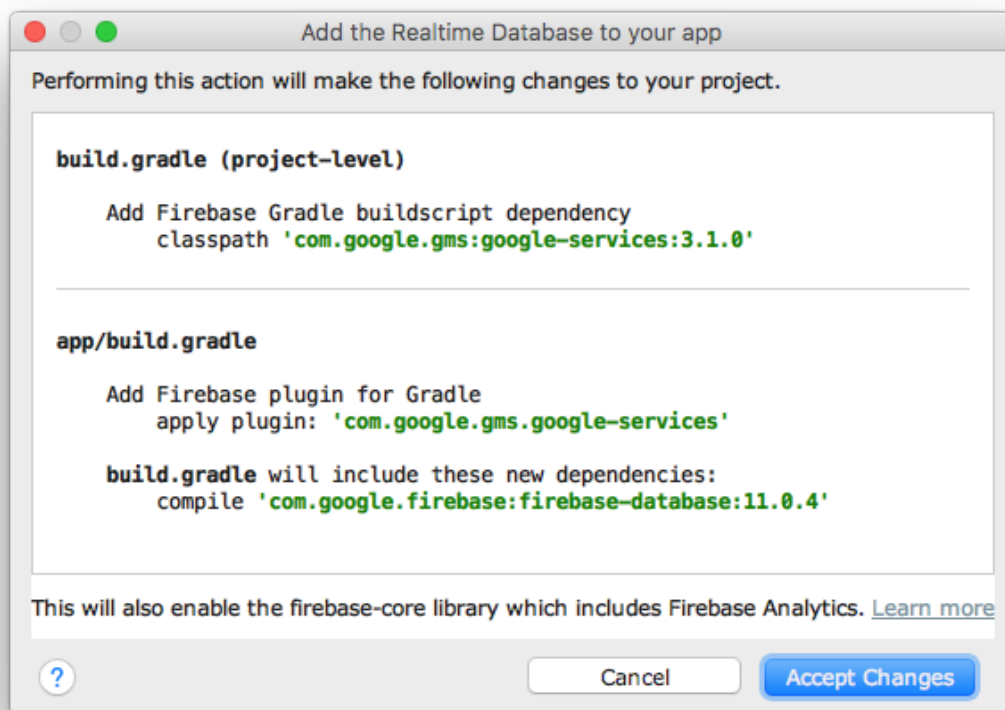
Next, in Android Studio, click on Tools > Firebase. This will open a dialog:



Click on 'Realtime Database' option and follow along as it says to add dependencies. It will make changes to Gradle file itself.

Make sure that in AS, you have 'Google Play Services' installed.

With other changes, it is clearly shown in the dialog Android Studio presented us:

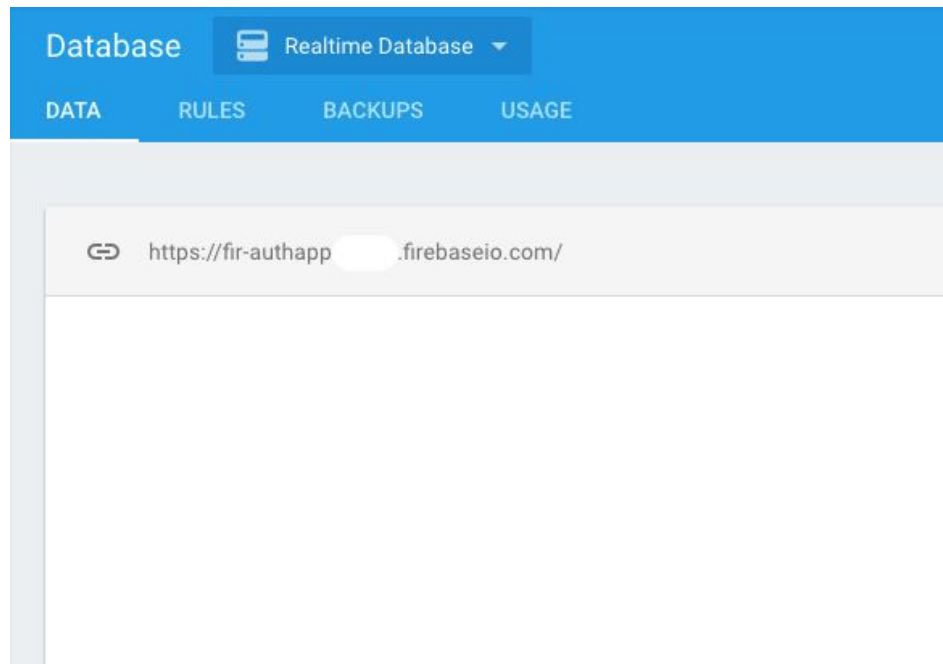


Now that the dependencies are set up correctly, we can start to work on our exciting app which will communicate to the firebase server. Make sure you have a project made in [Firebase Console](#). You will be needing its ID later to be configured in the project too.

Also, please note that as this lesson is meant for authentication purpose, we're accessing the same database on the basis of the user identity. So, to make our database private, we will add the following script in our Firebase project access rules:

```
{
  "rules": {
    ".read": "auth != null",
    ".write": "auth != null"
  }
}
```

If you visit the Database section in Firebase console, you will see that it is empty right now,



Now we can start working in our app.

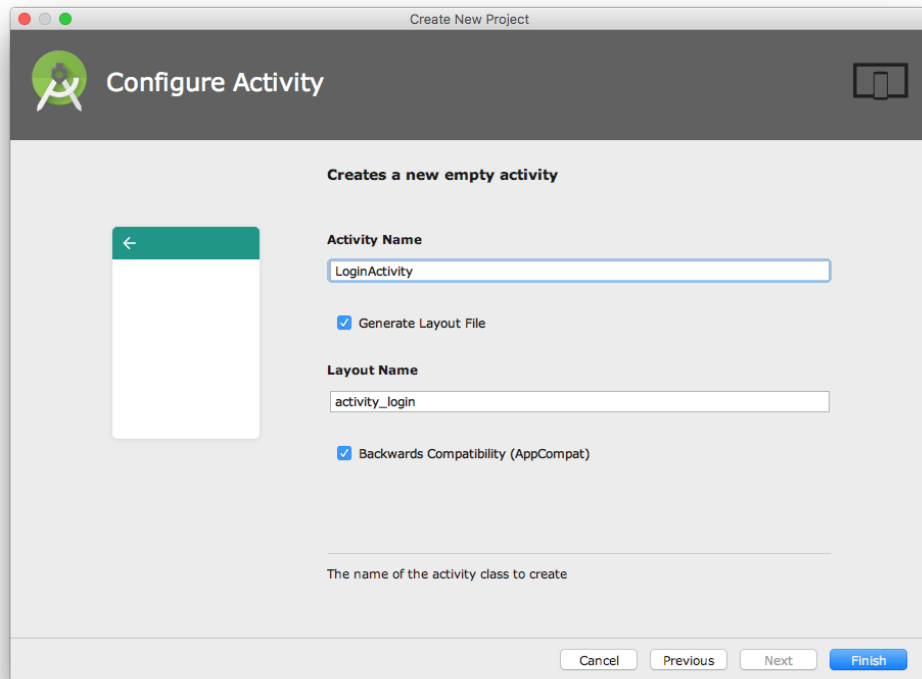
Adding Firebase Authentication

It is always good to authenticate users. Go back to Tools Firebase dialog and add 'Authentication' module. Go through the process and the dependencies should be added automatically.

All in all, I have following dependencies now:

```
implementation 'com.google.firebase:firebase-database:11.0.4'  
implementation 'com.google.firebase:firebase-auth:11.0.4'
```

While creating our project, we named our First activity as *LoginActivity*:



For introduction, we created three activities in our app, named as:

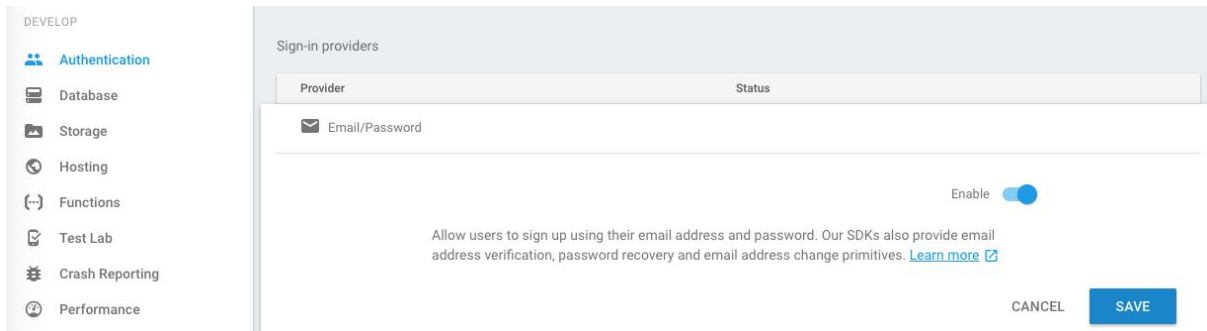
- CreateAccountActivity
- LoginActivity
- ForgotPasswordActivity

Purpose of each activity is clear with their names. Let us start with user registration with Firebase.

User Registration

In Firebase, we can register a new user with their email and password. Any additional information will have to be mapped manually and stored in Database ourself.

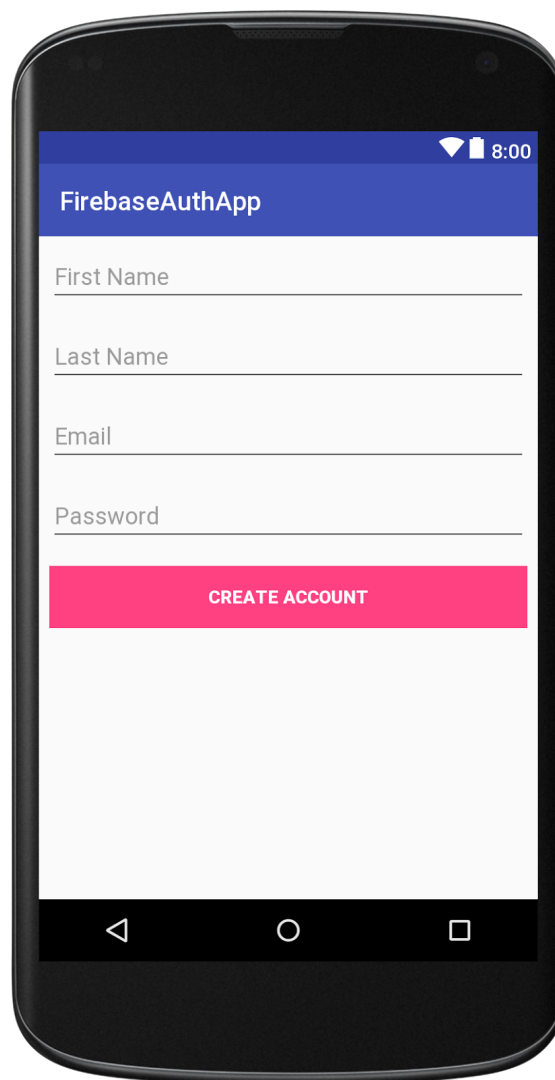
Also, to allow user registration with email and password, we must enable this service in Firebase console:



Unless we enable this in console, we will get a Firebase exception stating the reason:

```
com.google.firebase.auth.FirebaseAuthException: This operation is not allowed. You must enable this service in the console.
```

To create a user account, we created a simple UI as:



We simply used a simple UI elements to design this interface. Our XML looks like:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    ...>

    <EditText
        android:id="@+id/et_first_name"
        style="@style/viewCustom"
        android:ems="10"
        android:hint="@string/first_name"
        android:inputType="textPersonName" />

    <EditText
        android:id="@+id/et_last_name"
        style="@style/viewCustom"
        android:layout_below="@id/et_first_name"
        android:ems="10"
        android:hint="@string/last_name"
        android:inputType="textPersonName" />

    <EditText
        android:id="@+id/et_email"
        style="@style/viewCustom"
        android:ems="10"
        android:layout_below="@id/et_last_name"
        android:hint="@string/email"
        android:inputType="textEmailAddress" />

    <EditText
        android:id="@+id/et_password"
        style="@style/viewCustom"
        android:layout_below="@id/et_email"
        android:ems="10"
        android:hint="@string/password"
        android:inputType="textPassword" />

    <Button
        android:id="@+id/btn_register"
        style="@style/viewCustom"
        android:layout_below="@id/et_password"
```

```
        android:background="@color/colorAccent"
        android:text="@string/create_act"
        android:textColor="@android:color/white"
        android:textStyle="bold" />

</RelativeLayout>
```

We have used a simple Style in this file as well:

```
<resources>

    <!-- Base application theme. -->
    <style name="AppTheme"
parent="Theme.AppCompat.Light.DarkActionBar">
        <!-- Customize your theme here. -->
        <item name="colorPrimary">@color/colorPrimary</item>
        <item
name="colorPrimaryDark">@color/colorPrimaryDark</item>
        <item name="colorAccent">@color/colorAccent</item>
    </style>

    <style name="viewCustom">
        <item name="android:layout_width">match_parent</item>
        <item name="android:layout_height">wrap_content</item>
        <item name="android:layout_margin">8dp</item>
    </style>

</resources>
```

Finally, as of now, we have current Strings:

```
<resources>
    <string name="app_name">FirebaseAuthApp</string>

    <string name="email">Email</string>
    <string name="password">Password</string>
    <string name="first_name">First Name</string>
    <string name="last_name">Last Name</string>
    <string name="login_title">Login</string>
    <string name="create_act">Create Account</string>
</resources>
```

To start, let us define variables for our UI elements:

```
//UI elements
private var etFirstName: EditText? = null
private var etLastName: EditText? = null
private var etEmail: EditText? = null
private var etPassword: EditText? = null
private var btnCreateAccount: Button? = null
private var mProgressBar: ProgressDialog? = null
```

We will also define Firebase references for Database and Authorisation:

```
//Firebase references
private var mDatabaseReference: DatabaseReference? = null
private var mDatabase: FirebaseDatabase? = null
private var mAuth: FirebaseAuth? = null
```

Finally, we will also define some global variables for our usage in our program:

```
private val TAG = "CreateAccountActivity"

//global variables
private var firstName: String? = null
private var lastName: String? = null
private var email: String? = null
private var password: String? = null
```

Now, we will initialise these references in *onCreate* method:

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

    initialise()
}

private fun initialise() {
```

```

etFirstName = findViewById<View>(R.id.et_first_name) as EditText
etLastName = findViewById<View>(R.id.et_last_name) as EditText
etEmail = findViewById<View>(R.id.et_email) as EditText
etPassword = findViewById<View>(R.id.et_password) as EditText
btnCreateAccount = findViewById<View>(R.id.btn_register) as Button
mProgressBar = ProgressDialog(this)

mDatabase = FirebaseDatabase.getInstance()
mDatabaseReference = mDatabase!!.reference!!.child("Users")
mAuth = FirebaseAuth.getInstance()

btnCreateAccount!!.setOnClickListener { createNewAccount() }
}

```

Note the points:

- We initialise UI elements by calling *findViewById* method with appropriate IDs
- *mDatabaseReference* is initialised with database location with key 'Users'. Our user information will be stored at this location in database
- *mAuth* is the Firebase authentication service reference which will we will use to authenticate and create a new user
- We set a listener on our button and we will create a new method named as *createNewAccount*

Let us define this new method *createNewAccount*:

```

private fun createNewAccount() {
    ...
}

```

In this method, we will first get current String values present in EditText:

```

firstName = etFirstName?.text.toString()
lastName = etLastName?.text.toString()
email = etEmail?.text.toString()
password = etPassword?.text.toString()

```

Then, we will validate this text and show appropriate error message if any value is empty:

```
if (!TextUtils.isEmpty(firstName) && !TextUtils.isEmpty(lastName)
    && !TextUtils.isEmpty(email) && !TextUtils.isEmpty(password)) {

    ...
} else {
    Toast.makeText(this, "Enter all details", Toast.LENGTH_SHORT).show()
}
```

As Firebase connection and validation can take a little time, we will show the Progress bar with appropriate message before we make the authorisation:

```
mProgressBar!!.setMessage("Registering User...")
mProgressBar!!.show()
```

Finally, we make use of createUserWithEmailAndPassword function to create a new user with the mentioned email and password:

```
mAuth!!
.createUserWithEmailAndPassword(email!!, password!!)
.addOnCompleteListener(this) { task ->
    mProgressBar!!.hide()

    if (task.isSuccessful) {
        // Sign in success, update UI with the signed-in user's information
        Log.d(TAG, "createUserWithEmail:success")

        val userId = mAuth!!.currentUser!!.uid

        //update user profile information
        val currentUserDb = mDatabaseReference!!.child(userId)
        currentUserDb.child("firstName").setValue(firstName)
        currentUserDb.child("lastName").setValue(lastName)

        updateUserInfoAndUI()
    } else {
        // If sign in fails, display a message to the user.
```

```

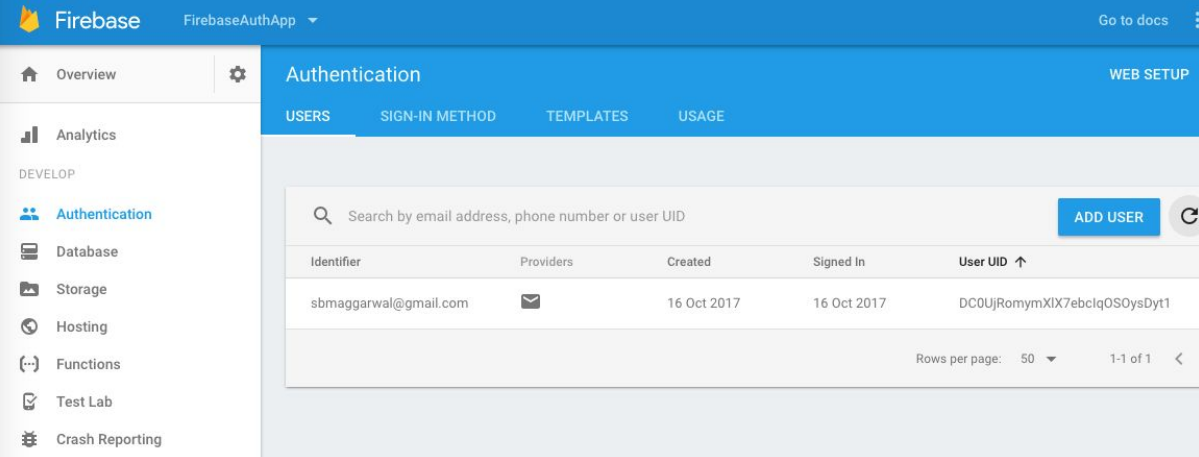
    Log.w(TAG, "createUserWithEmail:failure", task.exception)
    Toast.makeText(this@CreateAccountActivity, "Authentication failed.",
        Toast.LENGTH_SHORT).show()
}
}

```

We try creating a new user and using `addOnCompleteListener`, we get to know if the task was success or not.

If the task is a success, we create a new key and insert `firstName` and `lastName` key-values in it.

If you run the app right now and try registering, you can see the users getting registered:



The screenshot shows the Firebase Authentication console. The left sidebar contains navigation links for Overview, Analytics, Authentication (selected), Database, Storage, Hosting, Functions, Test Lab, and Crash Reporting. The main area displays the 'Authentication' section with tabs for USERS, SIGN-IN METHOD, TEMPLATES, and USAGE. A search bar is present above a table of users. The table has columns for Identifier, Providers, Created, Signed In, and User UID. One user is listed with the email 'sbmaggarwal@gmail.com', created on '16 Oct 2017', and signed in on '16 Oct 2017'. The User UID is 'DC0UjRomymXIX7ebclqOS0ysDyt1'. There is an 'ADD USER' button and a refresh icon in the top right of the table area.

Identifier	Providers	Created	Signed In	User UID ↑
sbmaggarwal@gmail.com	📧	16 Oct 2017	16 Oct 2017	DC0UjRomymXIX7ebclqOS0ysDyt1

In the last method call, we just start a new Activity once user is authenticated:

```

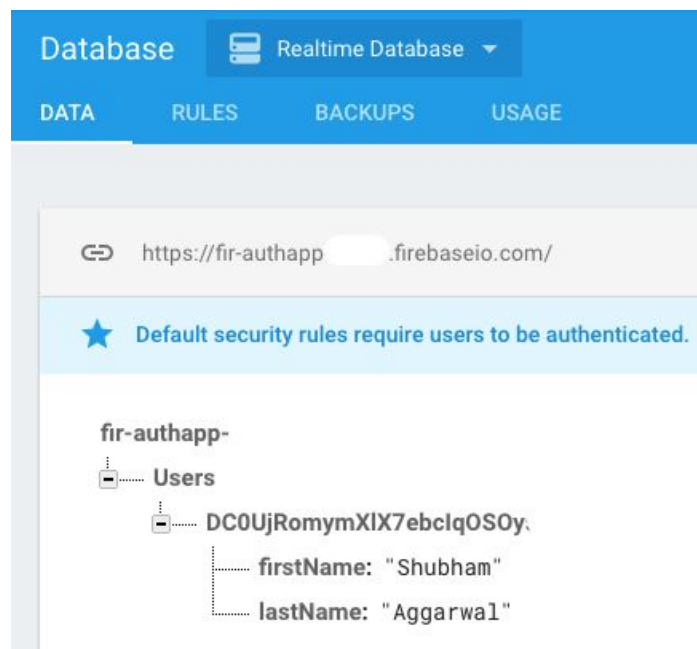
private fun updateUserInfoAndUI() {

    //start next activity
    val intent = Intent(this@CreateAccountActivity,
        MainActivity::class.java)
    intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP)
    startActivity(intent)
}

```

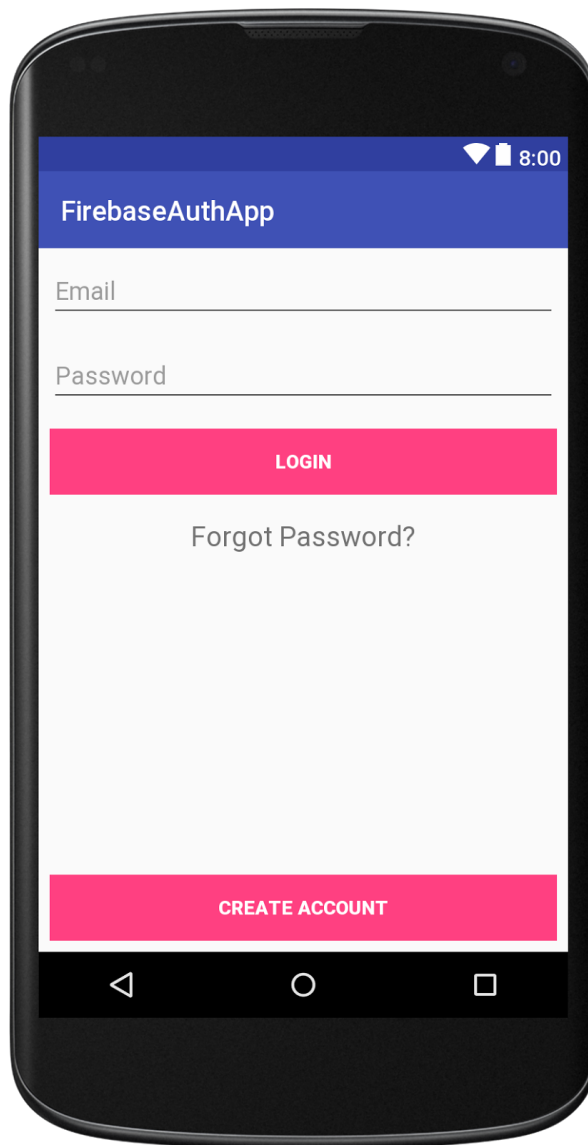

The FLAG_ACTIVITY_CLEAR_TOP flag clears the CreateAccountActivity from stack so that if user press back from MainActivity, he should not be taken back to CreateAccountActivity.

Also, as the user is now logged in, we can also see Database values being inserted in the database:



Allow User Login

For User login, we have created a similar UI:



We have basically three options in this screen:

- User can login
- Using 'Create Account' button, user will taken to CreateAccountActivity we defined earlier
- User can also reset his password if he submits his email in ForgotPasswordActivity, which will be opened when user hits 'Forgot Password?' view

Here is the XML interface which we designed for this layout:

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <EditText
        android:id="@+id/et_email"
        style="@style/viewCustom"
        android:ems="10"
        android:hint="@string/etEmail"
        android:inputType="textEmailAddress" />

    <EditText
        android:id="@+id/et_password"
        style="@style/viewCustom"
        android:layout_below="@id/et_email"
        android:ems="10"
        android:hint="@string/etPassword"
        android:inputType="textPassword" />

    <Button
        android:id="@+id/btn_login"
        style="@style/viewCustom"
        android:layout_below="@id/et_password"
        android:background="@color/colorAccent"
        android:text="@string/login_title"
        android:textColor="@android:color/white"
        android:textStyle="bold" />

    <TextView
        android:id="@+id/tv_forgot_password"
        style="@style/viewCustom"
        android:layout_below="@id/btn_login"
        android:clickable="true"
        android:gravity="center_horizontal"
        android:text="Forgot Password?"
        android:textSize="20sp" />

    <Button
        android:id="@+id/btn_register_account"
        style="@style/viewCustom"
```

```
        android:layout_alignParentBottom="true"
        android:background="@color/colorAccent"
        android:text="@string/create_act"
        android:textColor="@android:color/white"
        android:textStyle="bold" />

</RelativeLayout>
```

In this activity as well, we will define similar references to our UI elements and Firebase APIs:

```
private val TAG = "LoginActivity"

//global variables
private var email: String? = null
private var password: String? = null

//UI elements
private var tvForgotPassword: TextView? = null
private var etEmail: EditText? = null
private var etPassword: EditText? = null
private var btnLogin: Button? = null
private var btnCreateAccount: Button? = null
private var mProgressBar: ProgressDialog? = null

//Firebase references
private var mAuth: FirebaseAuth? = null
```

Let us initialise these references as well:

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

    initialise()
}

private fun initialise() {
    tvForgotPassword = findViewById<View>(R.id.tv_forgot_password) as TextView
    etEmail = findViewById<View>(R.id.et_email) as EditText
    etPassword = findViewById<View>(R.id.et_password) as EditText
```



```

        // If sign in fails, display a message to the user.
        Log.e(TAG, "signInWithEmail:failure", task.exception)
        Toast.makeText(this@LoginActivity, "Authentication failed.",
            Toast.LENGTH_SHORT).show()
    }
}
} else {
    Toast.makeText(this, "Enter all details", Toast.LENGTH_SHORT).show()
}
}

```

Once the user has logged in, he will be taken to the MainActivity:

```

private fun updateUI() {
    val intent = Intent(this@LoginActivity, MainActivity::class.java)
    intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP)
    startActivity(intent)
}

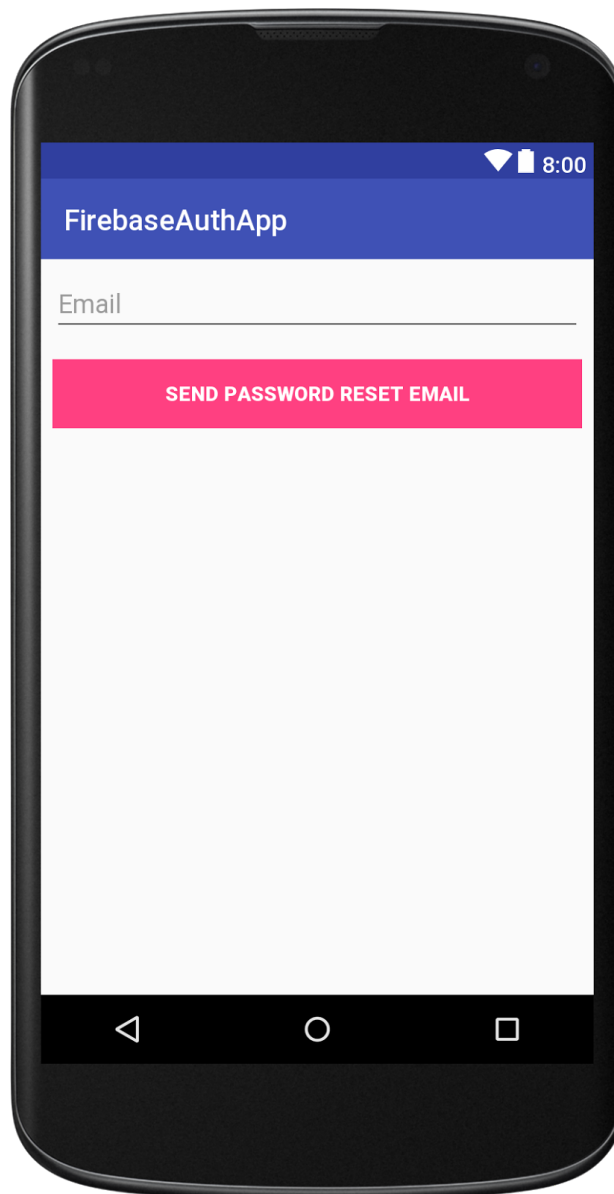
```

Using `signInWithEmailAndPassword`, we can login a user with email and password. Once a user logs in, we will take him to MainActivity to show user specific details. We will define this activity in a later section.

Sending password reset Email

Firebase allows us to reset a user's password with its own API itself. To demonstrate this, we created a `ForgotPasswordActivity`.

Here is a simple UI:



Here is the XML layout we used:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    ...>

    <EditText
        android:id="@+id/et_email"
        style="@style/viewCustom"
        android:ems="10"
        android:hint="@string/etEmail"
        android:inputType="textEmailAddress" />

    <Button
```

```
        android:id="@+id/btn_submit"
        style="@style/viewCustom"
        android:layout_below="@id/et_email"
        android:background="@color/colorAccent"
        android:text="@string/send_password_reset_email"
        android:textColor="@android:color/white"
        android:textStyle="bold" />
```

```
</RelativeLayout>
```

We defined references and initialised them:

```
private val TAG = "ForgotPasswordActivity"

//UI elements
private var etEmail: EditText? = null
private var btnSubmit: Button? = null

//Firebase references
private var mAuth: FirebaseAuth? = null

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

    initialise()
}

private fun initialise() {
    etEmail = findViewById<View>(R.id.et_email) as EditText
    btnSubmit = findViewById<View>(R.id.btn_submit) as Button

    mAuth = FirebaseAuth.getInstance()

    btnSubmit!!.setOnClickListener { sendPasswordResetEmail() }
}
```

For our rescue, we have a simple function call to make, `sendPasswordResetEmail`. Let us see it in action:


```

private fun sendPasswordResetEmail() {

    val email = etEmail?.text.toString()

    if (!TextUtils.isEmpty(email)) {
        mAuth!!
            .sendPasswordResetEmail(email)
            .addOnCompleteListener { task ->
                if (task.isSuccessful) {

                    val message = "Email sent."
                    Log.d(TAG, message)
                    Toast.makeText(this, message, Toast.LENGTH_SHORT).show()
                    updateUI()
                } else {
                    Log.w(TAG, task.exception!!.message)
                    Toast.makeText(this, "No user found with this email.",
Toast.LENGTH_SHORT).show()
                }
            }
    } else {
        Toast.makeText(this, "Enter Email", Toast.LENGTH_SHORT).show()
    }
}

private fun updateUI() {
    val intent = Intent(this@ForgotPasswordActivity, LoginActivity::class.java)
    intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP)
    startActivity(intent)
}

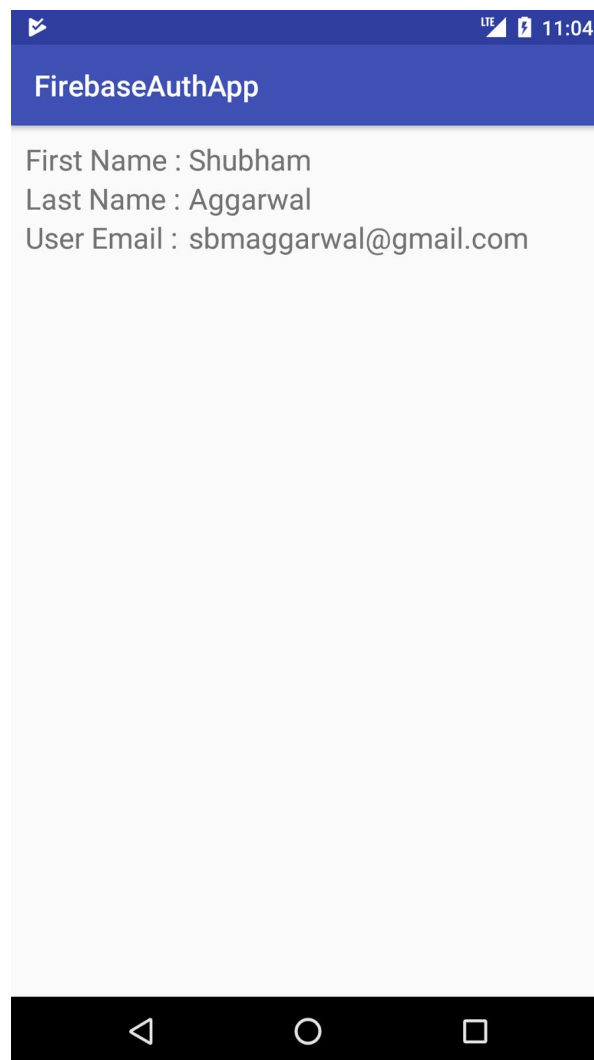
```

Firestore API simply tell us if the user with specified email even exist or not.

Access User specific data

In this final section, we will access some user specific data and display it in our MainActivity.

We will have following, very simple UI at the end:



We won't be showing any XML design as it is very simple.

As done before, we will define and initialise our references:

```
//Firebase references
private var mDatabaseReference: DatabaseReference? = null
private var mDatabase: FirebaseDatabase? = null
private var mAuth: FirebaseAuth? = null

//UI elements
private var tvFirstName: TextView? = null
private var tvLastName: TextView? = null
private var tvEmail: TextView? = null
```

```

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

    initialise()
}

private fun initialise() {
    mDatabase = FirebaseDatabase.getInstance()
    mDatabaseReference = mDatabase!!.reference!!.child("Users")
    mAuth = FirebaseAuth.getInstance()

    tvFirstName = findViewById<View>(R.id.tv_first_name) as
    TextView
    tvLastName = findViewById<View>(R.id.tv_last_name) as TextView
    tvEmail = findViewById<View>(R.id.tv_email) as TextView
}

```

Now, in onStart method, we will fetch user data from Firebase database.

```

override fun onStart() {
    super.onStart()

    val mUser = mAuth!!.currentUser
    val mUserReference = mDatabaseReference!!.child(mUser!!.uid)

    tvEmail!!.text = mUser.email

    mUserReference.addValueEventListener(object : ValueEventListener {
        override fun onDataChange(snapshot: DataSnapshot) {
            tvFirstName!!.text = snapshot.child("firstName").value as String
            tvLastName!!.text = snapshot.child("lastName").value as String
        }

        override fun onCancelled(databaseError: DatabaseError) {}
    })
}

```

That's it. According to our database structure in Firebase, we need to parse down the tree to reach our values.

Note that to get a database value, you must add a `addValueEventListener` listener to get a `DataSnapshot` reference from which database values can be fetched.

We never showed you how to logout a user. Create a simple UI element and call:

```
mAuth.signOut()
```

We leave this part as an exercise, although what to is just 1 line mentioned above.

Conclusion

In this lesson on Firebase Authentication, we learned how to take leverage from the power of Firebase Authentication and Database to store user data and personalise our app.

This app can be extended to a full feature app of user specific details and data. Go on and play with the code.