Firebase Tutorial

By Maksim Sundarev

#300307504

I am confirming that I have completed this assignment completely based on the requirements and it is working and fully functional.

In this tutorial you will familiarize yourself with Firebase and its services.

“Firebase” is a mobile and web application development platform developed by Firebase, Inc in 2011. It was acquired by Google in 2014, and as of today it has 19 products which are used by more than 1.5 million apps. (Wikipedia)

The following services (or products) are going to be covered in this tutorial:

- Authentication
- Realtime Database
- Storage
- Cloud Firestore
- Cloud Messaging

After completing the tutorial, you should have a solid foundation on Firebase and its products, and be able to continue your development, creating more advanced apps using Firebase.

So, let’s get started and dive into the world of Firebase.

1. Getting Started

First of all, create a new Android project in Android Studio, selecting all default settings. Call the project "*Firebase Tutorial*".

Now go to https://firebase.google.com and log-in into your Google account if not logged-in yet.

Then go to the Firebase console (click '*Go to console*' in the top right corner).

Now create a new project by clicking '*Add project*' button and call it same as the Android project you've just created.

You can disable Google Analytics, since we don't need it for the project.

Now you are in '*Project Overview'* section. Let's add our Android project to the Firebase.

You can do it either from Firebase Console or Android Studio. Let's do it from Android studio, since it is easier for us.

In the top Menu Bar in Android Studio select *Tools* > *Firebase*; Firebase assistant will be opened on the side.
Select *Authentication* > *Email and password authentication* > *Connect to Firebase.*
In the opened browser window choose a project we've just created. Then follow the steps with all defaults.

Now our projects are connected with each other and we can start do the *real work* here!

2. Setting up Layout and Other Resources

Now let's set up the layout file and all required resources.

Replace the default text view with the following code in your layout file '*activity_main.xml*':

```
<EditText
    android:id="@+id/signIn_email_editText"
    android:layout_width="350dp"
    android:layout_height="wrap_content"
    android:ems="10"
    android:hint="@string/email"
    android:inputType="textEmailAddress"
    android:textSize="20sp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.491"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.341" />

<Button
    android:id="@+id/signIn_button"
    android:layout_width="175dp"
    android:layout_height="60dp"
    android:background="@drawable/round_button"
    android:shadowColor="#000000"
    android:text="@string/sign_in_with_email"
    android:textColor="#FFFFFF"
    android:textColorHint="#FFFFFF"
    android:textSize="18sp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.636" />

<Button
    android:id="@+id/signUp_button"
    android:layout_width="175dp"
    android:layout_height="49dp"
    android:background="@drawable/round_button"
    android:text="@string/sign_up"
    android:textColor="#FFFFFF"
    android:textColorHint="#FFFFFF"
    android:textSize="18sp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.747" />
```

```
<EditText
    android:id="@+id/signIn_password_editText"
    android:layout_width="350dp"
    android:layout_height="wrap_content"
    android:layout_marginBottom="75dp"
    android:ems="10"
    android:hint="@string/password"
    android:inputType="textPassword"
    android:textSize="20sp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.491"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.542" />

<TextView
    android:id="@+id/signIn_TextViewTitle"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/welcome_to_firebase_tutorial"
    android:textSize="30sp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.485"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.105" />
```

To make our layout look prettier create two *Drawable Resource Files* in '*drawable*' folder: *round_button* and *round_button_dark,* and insert the following code for each file respectively:

round_button

```
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="rectangle">
    <solid android:color="@color/colorAccent" />
    <corners android:radius="50dp" />
</shape>
```

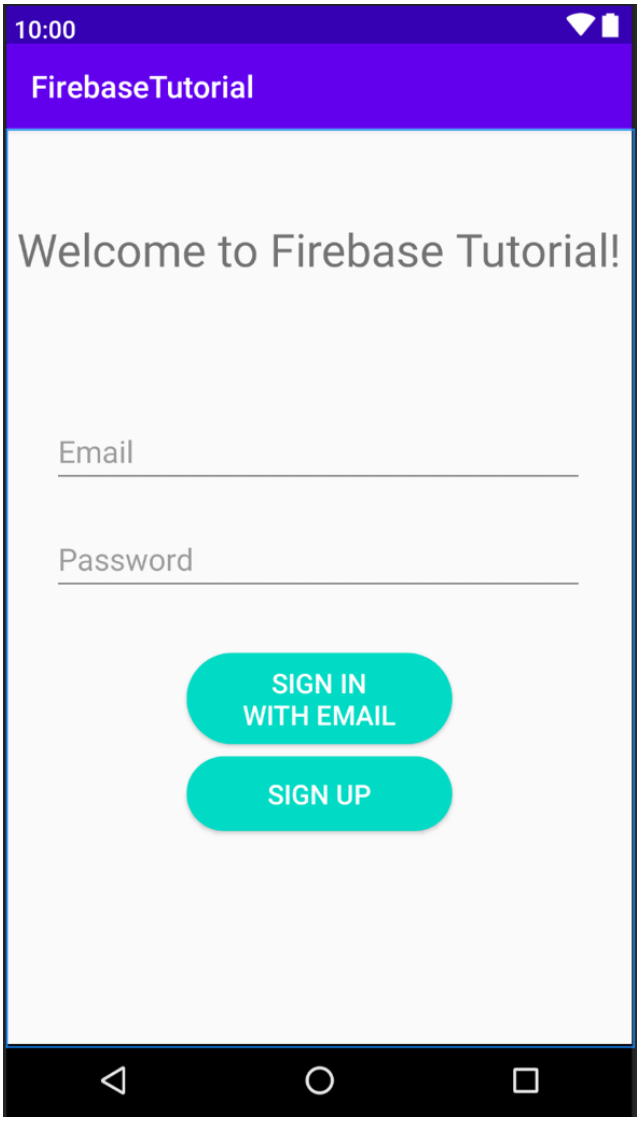
round_button_dark

```
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="rectangle">
    <solid android:color="@color/colorPrimaryDark" />
    <corners android:radius="50dp" />
</shape>
```

And finally add the following to your *strings.xml* file under *res > values*

```
<string name="sign_up">Sign Up</string>
<string name="sign_in">Sign In</string>
<string name="password">Password</string>
<string name="email">Email</string>
<string name="welcome_to_firebase_tutorial">Welcome to Firebase Tutorial!</string>
<string name="create_a_new_account">Create a New Account</string>
<string name="create_account">Create Account</string>
<string name="confirm_password">Confirm Password</string>
<string name="sign_in_with_email">Sign in\nwith Email</string>
<string name="save_settings">Save</string>
<string name="status">Status</string>
```

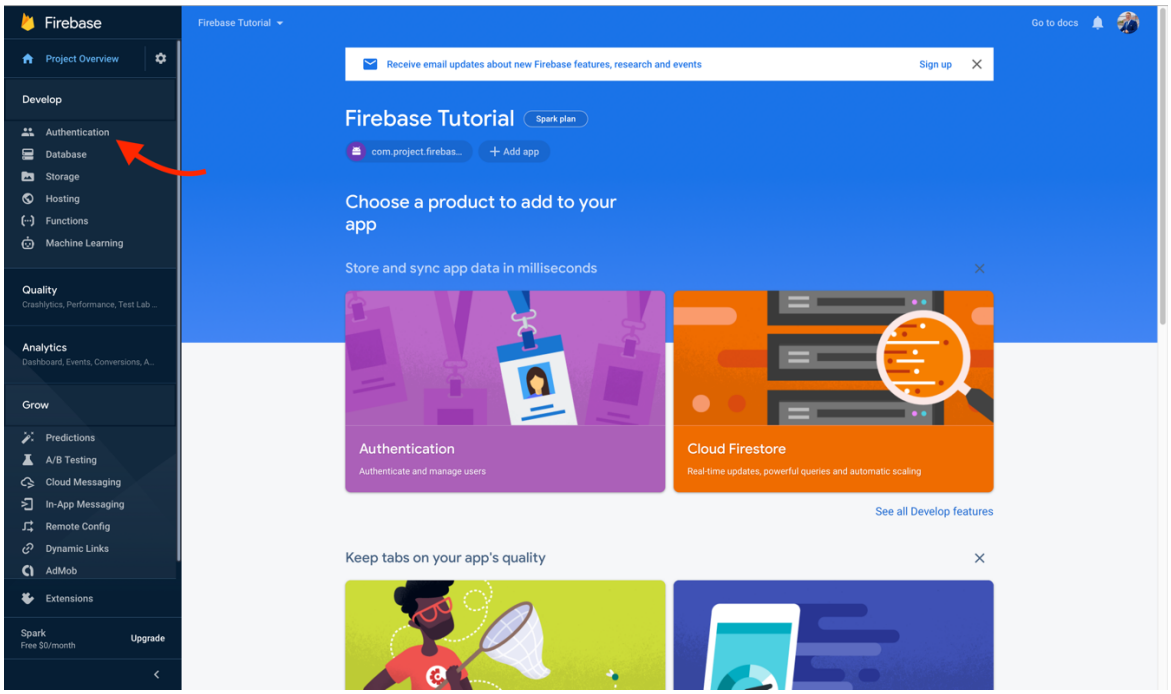
Now the layout looks like this:



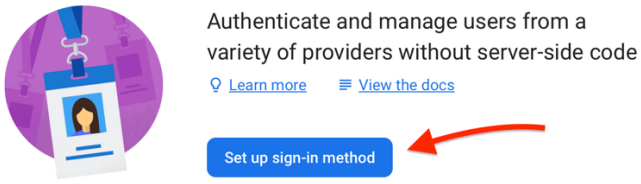
3. Implementing Sign-In/Sign-up with Email and Password

As you can already see from the layout, we are going to implement a user sign-in and user registration with email and password. Before start coding, let's do a little bit of extra set up:

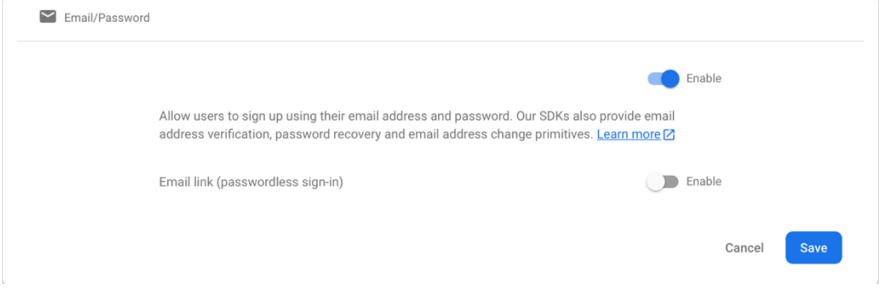
In your Firebase Console go to *Authentication* as shown on the screenshot



Click '*Set up sign-in method*'



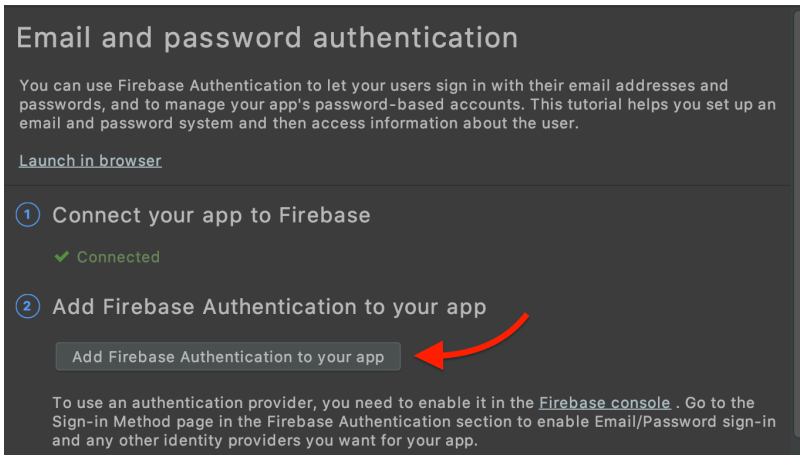
Click on '*Email/Password*', enable first option and click '*Save*'



We have enabled user sign-up with their email address and password, providing extra address verification, password recovery and email change functions, which we will not cover in this tutorial. Email link allows passwordless sign-in, but we also won't cover it, that is why we didn't enable it, though you might want to try this function on your own.

Now, let's jump into Android Studio to complete our set up.

In top Menu bar go to *Tools > Firebase;* Select *Authentication > Email and password authentication;* And now click '*Add Firebase Authentication to your app*'



Click '*Accept Changes*' in the pop-up dialog window.

Now we are ready to start actual coding! Follow step 3 in the Firebase Assistant.

After that your MainActivity.java should look like the following:

```java
package com.project.firebasetutorial;

import ...

public class MainActivity extends AppCompatActivity {

    // FirebaseAuth instance to log/sign in users
    private FirebaseAuth mAuth;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        mAuth = FirebaseAuth.getInstance();
    }

    @Override
    public void onStart() {
        super.onStart();
        // Check if user is signed in (non-null) and update UI accordingly.
        FirebaseUser currentUser = mAuth.getCurrentUser();
        updateUI(currentUser);
    }
}
```

You can notice on line 28 call to the method *updateUI* with *currentUser* as a parameter. This is the call to method that should update user interface according to if the user is logged in or not, though the implementation is left for the developer. Since the activity we are currently working on should be displayed only when the user is not logged in or registered, we are going to create a conditional statement to check if the user is logged in or not but will leave the inner logic for later. This is how our *onStart* method should look for now:

```
@Override
public void onStart() {
    super.onStart();
    // Check if user is signed in (non-null) and update UI accordingly.
    FirebaseUser currentUser = mAuth.getCurrentUser();
    if (currentUser != null) {
        //TODO implement logic after checking user log-in status
    }
}
```

Now we can implement *onClickListener's onClick* method for *Sign In* button. Insert the following into your *onCreate* method after *mAuth* initialization:

```
Button signInButton = findViewById(R.id.signIn_button);
signInButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {

    }
});
```

You can refer to Step 5 in Firebase Assistant that shows the example code for signing in existing users, but we also have to retrieve String values for email and passwords from corresponding EditTexts, and we also need to check for empty strings, since we don't want to allow empty values.

Insert the following before *signInButton* declaration:

```
EditText emailEditText = findViewById(R.id.signIn_email_editText);
EditText passwordTEditText = findViewById(R.id.signIn_password_editText);
```

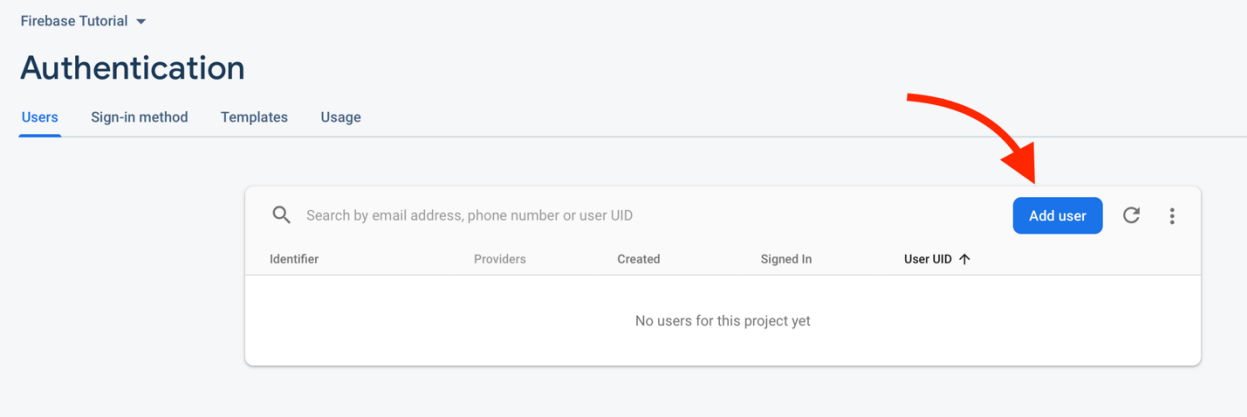
We call *signInWithEmailAndPassword* with retrieved email and password strings on *FirebaseAuth* instance to sign in the user. We also add *OnCompleteListener* and override *onComplete* method to check if the sign in task was successful or not.

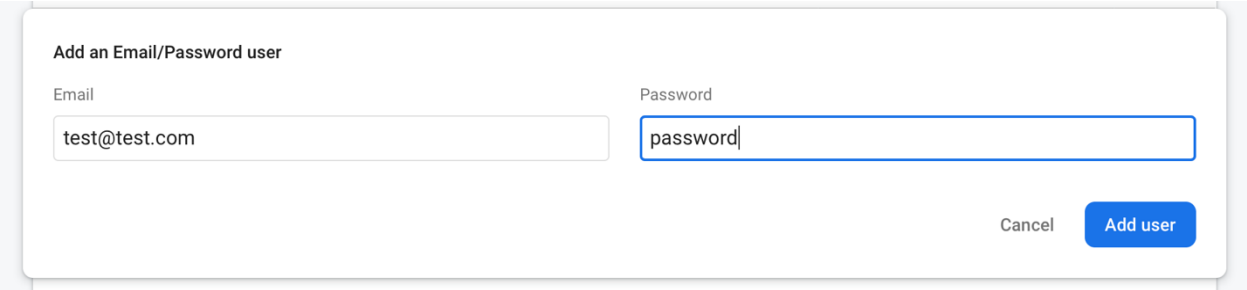
In *onComplete* the parameter *task* has *isSuccessful()* method that returns a Boolean value representing success of a given task, in our case – sign in.

Insert the following code into your *onClick* method for *Sign In* button:

```
// Retrieve String values
String email = emailEditText.getText().toString();
String password = passwordTEditText.getText().toString();

// Check for empty strings
if (!email.equals("") && !password.equals("")) {
    mAuth.signInWithEmailAndPassword(email, password)
            .addOnCompleteListener(new OnCompleteListener<AuthResult>() {
                @Override
                public void onComplete(@NonNull Task<AuthResult> task) {
                    // Successful sign in
                    if (task.isSuccessful()) {
                        Toast.makeText(MainActivity.this,"Login Successful",
                                Toast.LENGTH_SHORT).show();

                    // Unsuccessful sign in
                    } else {
                        String message = task.getException().toString();
                        Toast.makeText(MainActivity.this,
                                "Login Failed " + message,
                                Toast.LENGTH_SHORT).show();
                    }
                }
            });
} else {
    Toast.makeText(getApplicationContext(),
            "Email or Password cannot be empty!", Toast.LENGTH_SHORT).show();
}
```

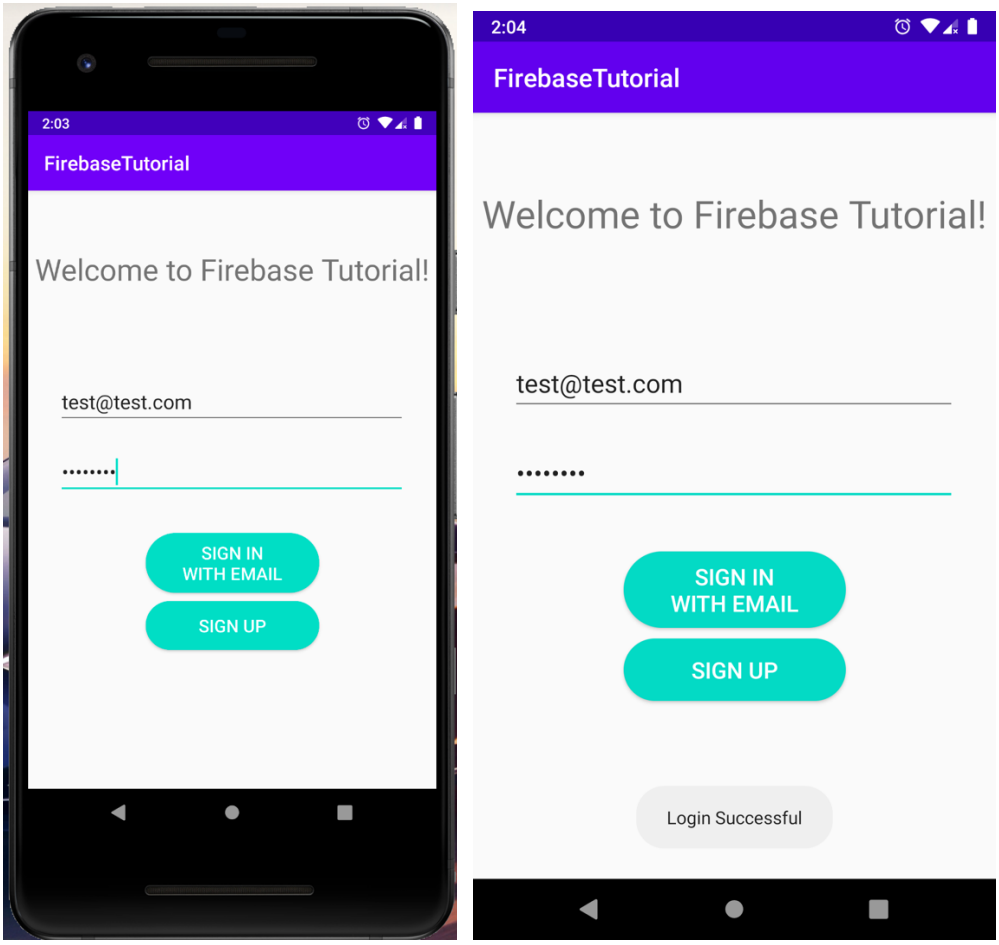
For now, we could potentially sign in a user, but we don't have any yet, and we cannot register one in our app too. Still, we can create a user manually in Firebase Console in our Authentication panel by clicking '*Add user*' button:





Create a test user and take a note of the email and password. Click '*Add user*'.

Now you can run the app, enter user details that you have just created. That is what you are going to see:



Since it is our first run, it might take a little while for Firebase to sign in our test user. After approximately 15 seconds you will receive the following toast message, indicating successful sign in.

Now we are ready to implement our registration for users.
Create a new empty activity called *RegistrationActivity* and insert the following code into its layout xml file:

```
<TextView
    android:id="@+id/registration_title_textView"
    android:layout_width="316dp"
    android:layout_height="46dp"
    android:layout_marginBottom="75dp"
    android:text="@string/create_a_new_account"
    android:textSize="25sp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.168"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.196" />

<Button
    android:id="@+id/registration_createNewAccount_button"
    android:layout_width="184dp"
    android:layout_height="55dp"
    android:background="@drawable/round_button"
    android:text="@string/create_account"
    android:textColor="#FFFFFF"
    android:textColorHint="#FFFFFF"
    android:textSize="18sp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.863"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.775" />

<EditText
    android:id="@+id/registration_email_editText"
    android:layout_width="350dp"
    android:layout_height="wrap_content"
    android:ems="10"
    android:hint="@string/email"
    android:inputType="textEmailAddress"
    android:textSize="20sp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.491"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.348" />

<EditText
    android:id="@+id/registration_password_editTex"
    android:layout_width="350dp"
    android:layout_height="wrap_content"
    android:layout_marginBottom="75dp"
    android:ems="10"
    android:hint="@string/password"
    android:inputType="textPassword"
    android:textSize="20sp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.491"
    app:layout_constraintStart_toStartOf="parent"
```

```
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.529" />


<EditText
    android:id="@+id/registration_confirm_password_editTex"
    android:layout_width="350dp"
    android:layout_height="wrap_content"
    android:layout_marginBottom="75dp"
    android:ems="10"
    android:hint="@string/confirm_password"
    android:inputType="textPassword"
    android:textSize="20sp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.491"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.669" />
```

Paste the following code after *signInButton's setOnClickListener* method in MainActivity.java file:

```
Button signUpButton = findViewById(R.id.signUp_button);
signUpButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        startActivity(new Intent(MainActivity.this, RegistrationActivity.class));
    }
});
```

In RegistrationActivity.java complete step 3 of Firebase Assistant for Authentication and copy *onStart* method from MainActivity.java.

Add the following code after mAuth declaration:

```
EditText emailEditText = findViewById(R.id.registration_email_editText);
EditText passwordEditText = findViewById(R.id.registration_password_editTex);
EditText confirmPasswordEditText =
findViewById(R.id.registration_confirm_password_editTex);

Button registerButton = findViewById(R.id.registration_createNewAccount_button);
registerButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {

    }
});
```

Finally add this code to the onClick method:

```
// If passwords match we retrieve strings and continue process
String email = emailEditText.getText().toString();
String password = passwordEditText.getText().toString();
String confirmPassword = confirmPasswordEditText.getText().toString();

if (password.equals(confirmPassword)) {
    // Check for all strings to be not empty
    if (!email.equals("") && !password.equals("")) {
        // Create user with email and password
        mAuth.createUserWithEmailAndPassword(email, password)
                .addOnCompleteListener(new OnCompleteListener<AuthResult>() {
                    @Override
                    public void onComplete(@NonNull Task<AuthResult> task) {
                        if (task.isSuccessful()) {
                            startActivity(new Intent(
                                    RegistrationActivity.this,
                                    MainActivity.class
                            ));
                            // We finish this activity, since we don't want users
                            // to return to this activity after successful
                            // registration process
                            finish();
                        } else {
                            Toast.makeText(RegistrationActivity.this,
                                    "Sign Up Failed: " +
                                            Objects.requireNonNull(task
                                                    .getException()
                                                    .getMessage()),
Toast.LENGTH_SHORT)
                                    .show();
                        }
                    }
                });
    } else {
        Toast.makeText(getApplicationContext(),
                "Email and Password Cannot be empty", Toast.LENGTH_SHORT).show();
    }
} else {
    Toast.makeText(view.getContext(), "Passwords Do Not Match!",
            Toast.LENGTH_SHORT).show();
}
```

For now, after successful registration we will send the user to Sign In (MainActivity) to check our registration, otherwise show the toast message with an error.

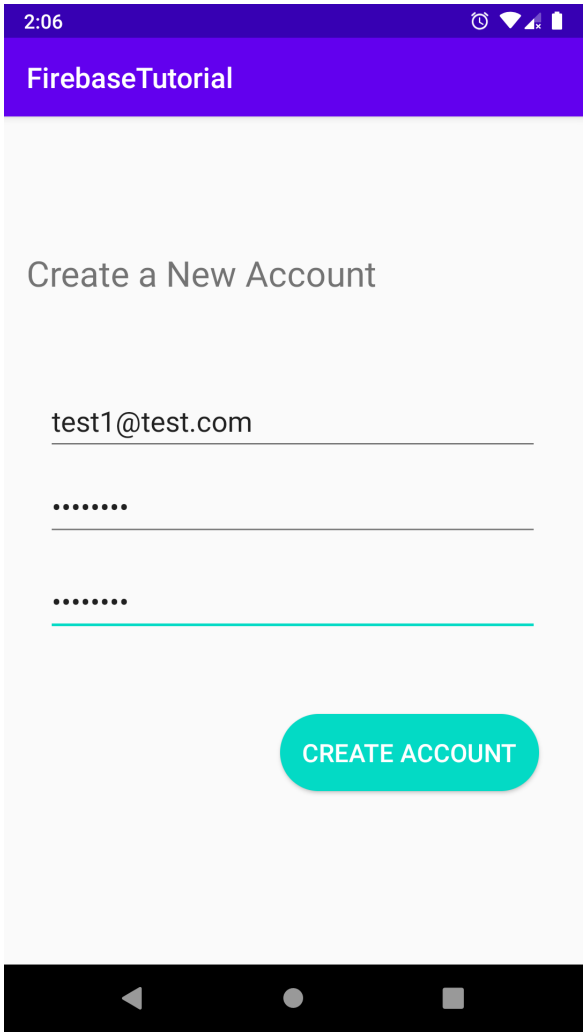
To allow Internet access for our app, add the following line into AndroidManifest.xml file:

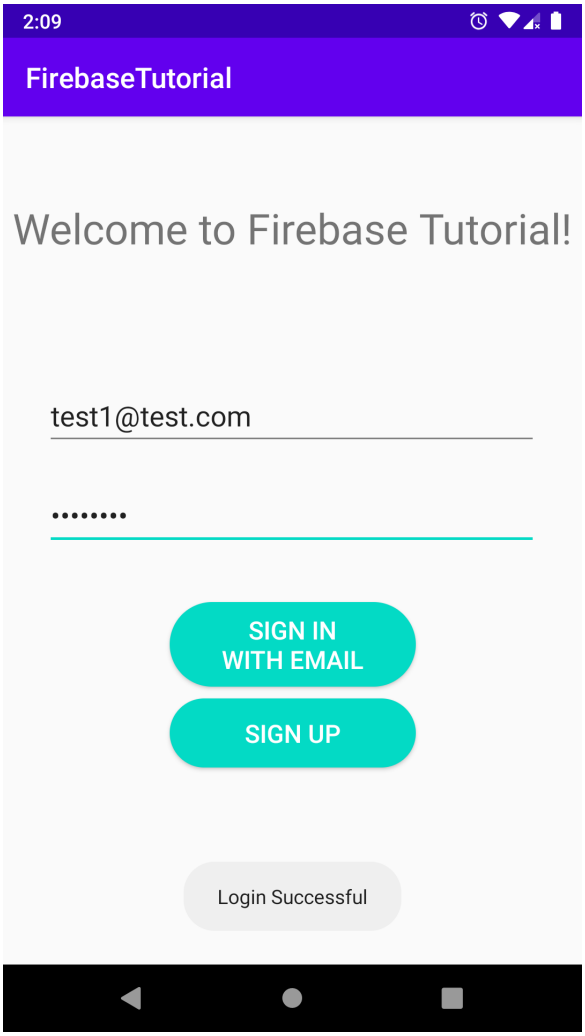
```
<uses-permission android:name="android.permission.INTERNET" />
```

The manifest file should now look like the following:

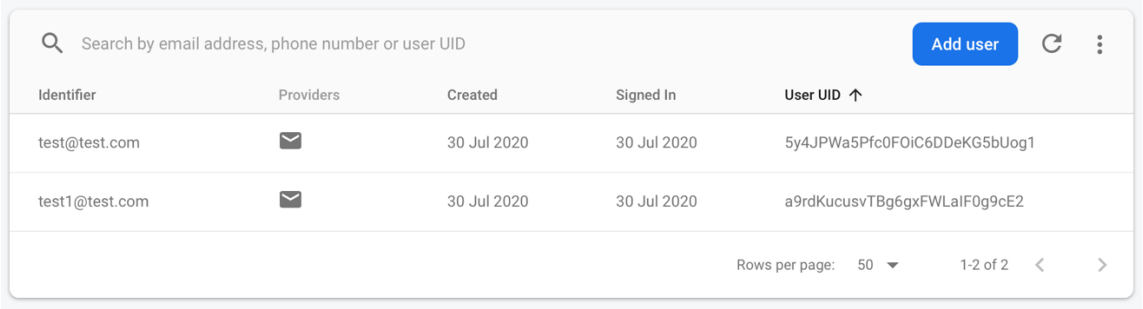
```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.project.firebasetutorial">

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

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="FirebaseTutorial"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".RegistrationActivity"></activity>
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Now we can run the app, create a new account and try to sign in.





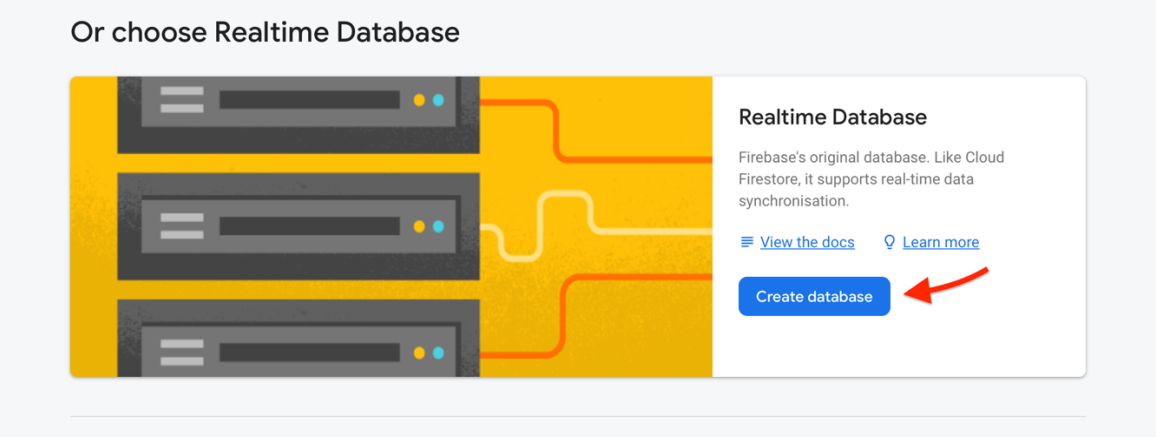
In the Firebase Console in Authentication control panel under *Users* you can also see our created users, notice that for each user we can see the provider, which is a method how user has authenticated, date created and date last signed in, also notice that each time a user is created unique user UID is assigned to that user.



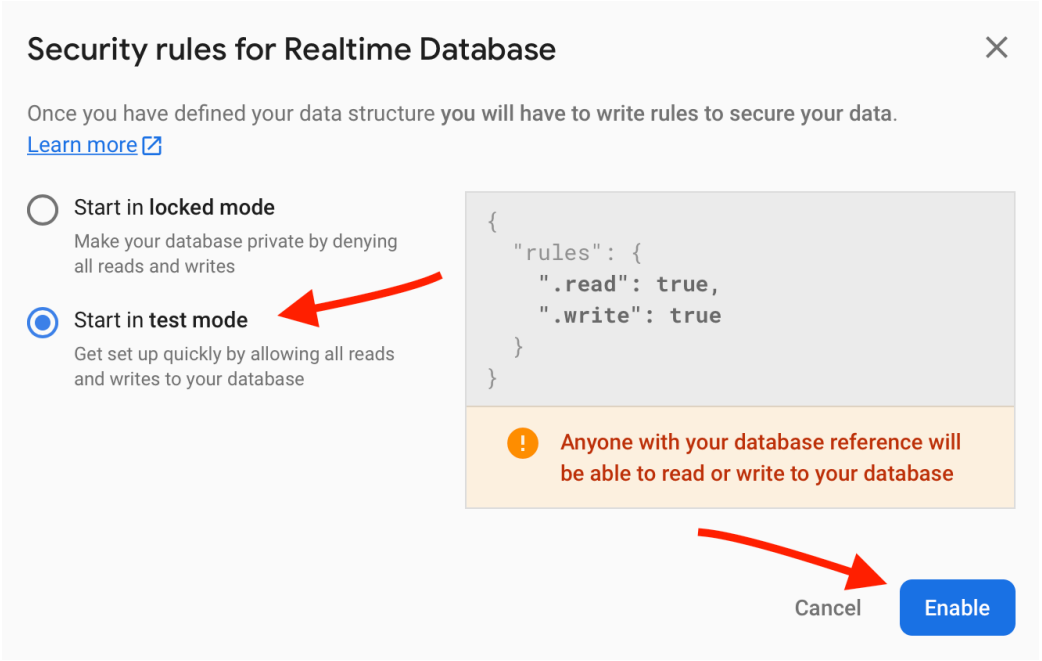
4. Implementing Realtime Database

Now we can successfully sign in or register our users. But we might also want to create a database entry for each newly registered user to store all information about them (should it be email, nickname, profile image, etc.) For this purpose, we are going to use *Firebase Realtime Database*.

For this, first go to Firebase Console and select *Database* under *Develop* tab on the left panel. Scroll down to '*Or choose Realtime Database*' and click '*Create database*':

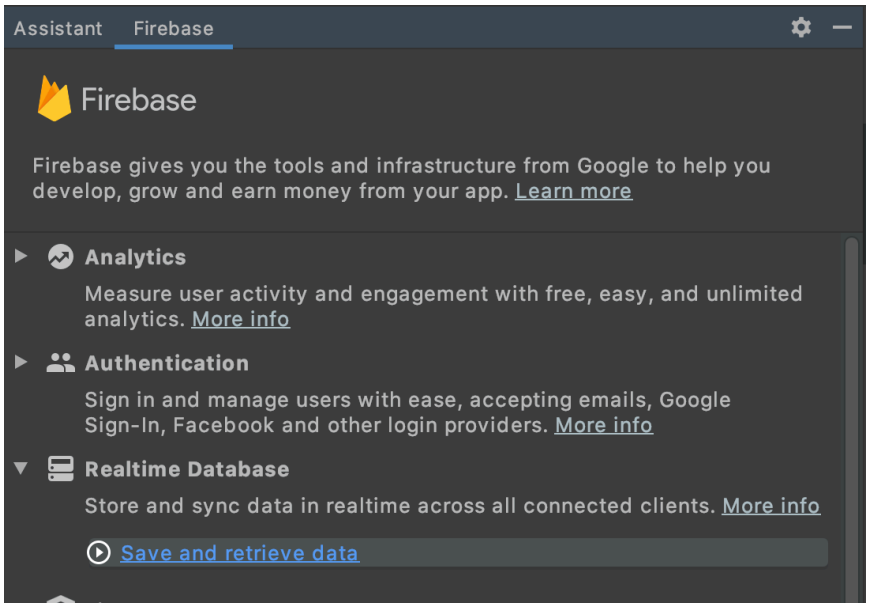


Set '*test mode*' security rule for now and click '*Enable*':



After that, you will see your Database entry. Data in Firebase Realtime Database is stored as JSON objects, which are objects having their fields as key -> value pairs. When data is added to this JSON tree, it becomes a node in the existing JSON structure with an associated key. You can learn more about Firebase DB structure at this guide.

Now let's go to our project in Android Studio to complete Database set up.
In Firebase Assistant go to *Realtime Database > Save and retrieve data:*



Click '*Add the Realtime Database to your app*' > '*Accept changes*'.

Now we are finally ready to code our DB interactions. Keep in mind that you can always refer to sample code in Firebase Assistant. Our goal for now is to create a DB entry each time a new user is registered, so we will work in RegistrationActivity.java file.

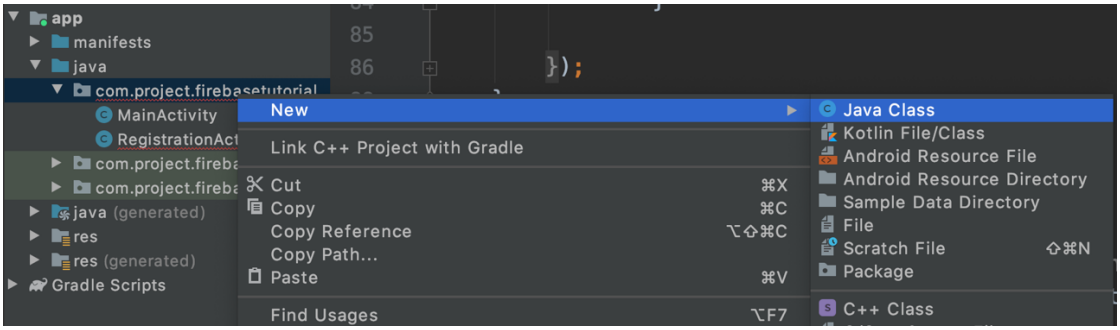
Create the following method stub where we will perform database interaction:

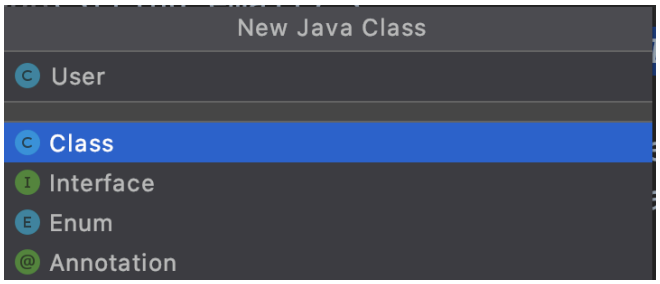
```
private void addUserToDB(String email) {}
```

Inside method paste the following code:

```
DatabaseReference mDatabase = FirebaseDatabase.getInstance().getReference("users");
```

We are creating a *DatabaseReference* instance which will point to our newly created DB (we call *getInstance()* for it) and we want to create a node with key '*users*'. Actually, this means that we create a subdivision in our DB where we will store all information of all user objects.
To create an object into our DB we need to have something to insert. A good practice is to insert Java objects directly into DB. That means that we need to create a new Java class called *User:*





Create the following class fields in User class:

```
private String id;
private String email;
private String userName;
private String status;
private String profileImage;
```

Place your cursor within class boundaries and press Ctrl + N
(Command + N on Mac) and generate constructor with all fields selected.
Generate getters and setters for all fields the same way.
Now your class should have constructor and getters/setters for all created fields.

We can now return to our *RegistrationActivity* and *addUserToDB* method.

Although we have created a number of fields, we are not going to use all of them right now. Those that we really need now are *id* and *email.* We can now set *username* to email address, *status* to some default status, and *profileImage* to default.
Insert the following code after creating Database reference:

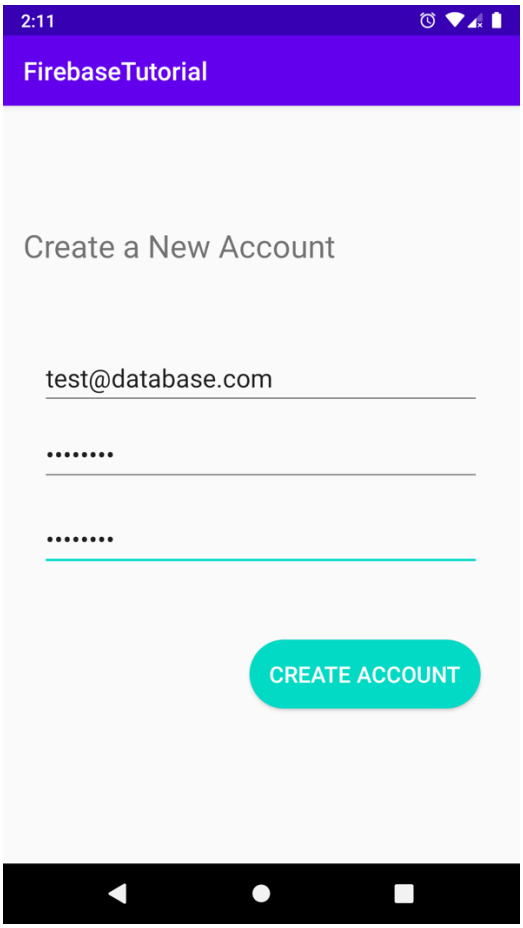
```
// Get ID from Firebase Auth
String userID = FirebaseAuth.getInstance().getCurrentUser().getUid();
// If there is no Firebase Auth, generate new UUID
final String id = userID != null ? userID : UUID.randomUUID().toString();
// Create a new User object
User newUser = new User(id, email, email, "default status", "default");
// Create a user entry in DB
mDatabase.child(id).setValue(newUser);
```

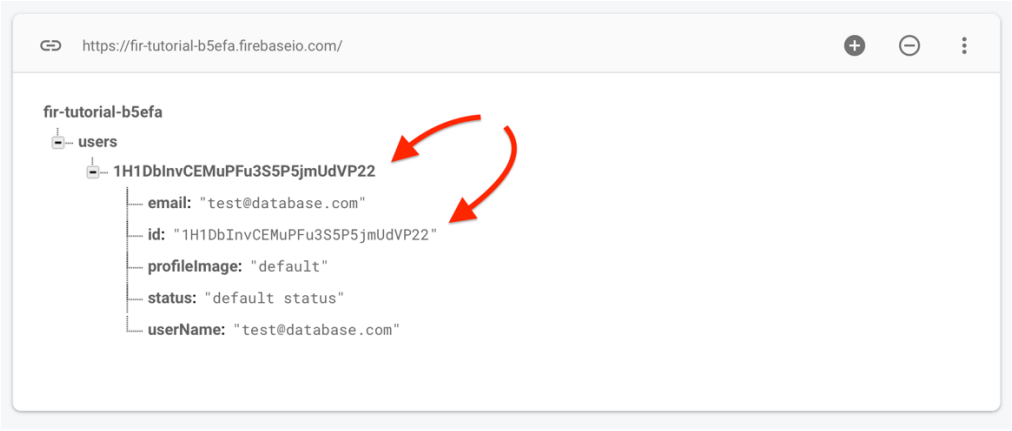
We want to get user ID from *Firebase Auth* because user is registered there before we create a database entry, but if something goes wrong, we assign this user a new *UUID* and use it to create our *User* object. After that, we create a new node with our *id* value and place in it our *newUser* object. And this is pretty much all for creating a user entry in the database. Now we need to call this method from our onCLickListener for registration button, but make sure that you call this method inside the conditional for *task.isSuccessful()*, otherwise we don't want to create a user entry in DB. Call method before *startActivity()* call:

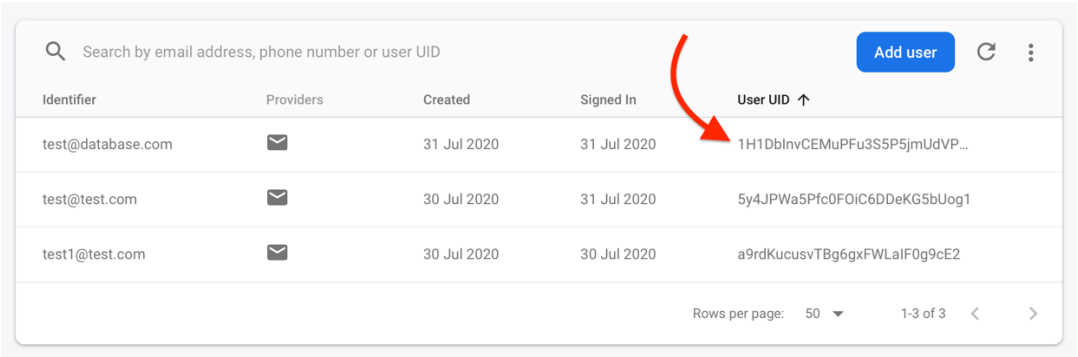
```
50              if (password.equals(confirmPassword)) {
51                  // Check for all strings to be not empty
52                  if (!email.equals("") && !password.equals("")) {
53                      // Create user with email and password
54                      mAuth.createUserWithEmailAndPassword(email, password)
55                              .addOnCompleteListener((task) → {
58                                      if (task.isSuccessful()) {
59                                          // Create a DB entry for a newly registered user
60                                          addUserToDB(email);
61                                          // Start the following activity
62                                          startActivity(new Intent(
63                                                  packageContext: RegistrationActivity.this,
64                                                  MainActivity.class
65                                          ));
```

Now let's run our app and try to register a new user.

Pay attention to the ID values we get after successful registration. And also notice that field names are absolutely same as we created them in the *User* class:


2:11
FirebaseTutorial
Create a New Account
test@database.com
••••••••
••••••••
CREATE ACCOUNT


https://fir-tutorial-b5efa.firebaseio.com/
fir-tutorial-b5efa
users
1H1DbInvCEMuPFu3S5P5jmUdVP22
email: "test@database.com"
id: "1H1DbInvCEMuPFu3S5P5jmUdVP22"
profileImage: "default"
status: "default status"
userName: "test@database.com"


Search by email address, phone number or user UID
Add user
Identifier
Providers
Created
Signed In
User UID
test@database.com
31 Jul 2020
31 Jul 2020
1H1DbInvCEMuPFu3S5P5jmUdVP...
test@test.com
30 Jul 2020
31 Jul 2020
5y4JPWa5Pfc0FOiC6DDeKG5bUog1
test1@test.com
30 Jul 2020
30 Jul 2020
a9rdKucusvTBg6gxFWLaIF0g9cE2
Rows per page: 50
1-3 of 3

Now we want to have some main activity where we can interact more with the app, besides just logging in and registering. Let's create a new activity called *MainPageActivity* and use the following xml for its layout:

```
<Button
    android:id="@+id/db_read_button"
    android:layout_width="184dp"
    android:layout_height="55dp"
    android:background="@drawable/round_button"
    android:shadowColor="#000000"
    android:text="Read From DB"
    android:textColor="#FFFFFF"
    android:textColorHint="#FFFFFF"
    android:textSize="18sp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.497"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.233" />

<Button
    android:id="@+id/logout_button"
    android:layout_width="184dp"
    android:layout_height="55dp"
    android:background="@drawable/round_button_dark"
    android:shadowColor="#000000"
    android:text="Log Out"
    android:textColor="#FFFFFF"
    android:textColorHint="#FFFFFF"
    android:textSize="18sp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.497"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.618" />

<Button
    android:id="@+id/profileButton"
    android:layout_width="184dp"
    android:layout_height="55dp"
    android:background="@drawable/round_button"
    android:shadowColor="#000000"
    android:text="Open Profile Settings"
    android:textColor="#FFFFFF"
    android:textColorHint="#FFFFFF"
    android:textSize="18sp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.497"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.425" />
```

For now, let's implement logic for '*Read from DB*' button. Clicking on it will result in displaying a toast message with user's ID and Email address.
Let's define this button and add *onClickListener:*

```
Button readFromDbButton = findViewById(R.id.db_read_button);
readFromDbButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {

    }
});
```

Also define *Firebase Auth* object as a class field, and copy *onStart()* method from any previous activities, we are going to implement it soon.

To read from any database we need some sort of a query to find desired data. So, let's create one for Firebase. But first we need to determine what upon are we going to build our query. Since we assume each user has unique email, let's query based on current user's email.

Make sure you have the following code above button declaration:

```
// Initialize Firebase Auth
FirebaseAuth mAuth = FirebaseAuth.getInstance();
// Get current user object
final FirebaseUser user = mAuth.getCurrentUser();
// Get database reference that will point to the DB root
final DatabaseReference mDatabaseReference =
FirebaseDatabase.getInstance().getReference();
```

We retrieve user's email and create a query inside *onClick()* method:

```
// Retrieve user email from user object
String userEmail = user.getEmail();
// Create a query for Firebase Realtime DB based on email
Query queryUser =
        mDatabaseReference.child("users").orderByChild("email").equalTo(userEmail);
```

Well, we now have a query, then we want to execute it somehow and receive our data back. To execute a query, we need to add *ValueEventListener* for this query. In a callback *onDataChanged()* we have an instance of *DataSnapshot* passed as a parameter. It is a kind of collection that contains child objects in it (might be one or many), which we retrieve by iterating through them.

In our case, we expect only one object to be in, and if there is one, we will display a toast message with user's email and ID. If this snapshot doesn't exist, means that there is no object that satisfies our query. The second callback that is required to be overridden is *onCancelled().* It usually called when some error occurs, for example query has a mistake in it, and in it we will display a message that will show a cause of an error.

Paste the following code after the query:

```
// Execute query and get the result back
queryUser.addValueEventListener(new ValueEventListener() {
    @Override
    public void onDataChange(@NonNull DataSnapshot snapshot) {
        // Check if snapshot that is returned to us exists
        if (snapshot.exists()) {
            // Snapshot is a collection of objects returned to us,
            // you can assume that it's sort of a cursor in some DBMSs
            // We have to iterate through it and get our single result
            for (DataSnapshot snapshotData : snapshot.getChildren()) {
                // Here is our data, we will display a Toast with data
                // Notice that snapshotData is our desired object from the DB
                Toast.makeText(
                        MainPageActivity.this,
                        "Your Email is: " + snapshotData.child("email")
                                .getValue() + "\n\n" +
                                "Your ID is: " + snapshotData.child("id")
                                .getValue(),
                        Toast.LENGTH_LONG
                ).show();
            }
        } else {
            // Here is the case when our query was unsuccessful
            // and we could not find any results
            // Display a Toast message indicating it
            Toast.makeText(MainPageActivity.this,
                    "Could not find desired entry",
                    Toast.LENGTH_LONG).show();
        }
    }

    @Override
    public void onCancelled(@NonNull DatabaseError error) {
        // This is the case when some error occurred while querying DB
        // We will display an error message here
        Toast.makeText(
                MainPageActivity.this,
                error.toException().getMessage(),
                Toast.LENGTH_LONG
        ).show();
    }
});
```

Change the line where we declare current user in *onStart* method in all activities:

```
FirebaseUser currentUser = FirebaseAuth.getInstance().getCurrentUser();
```

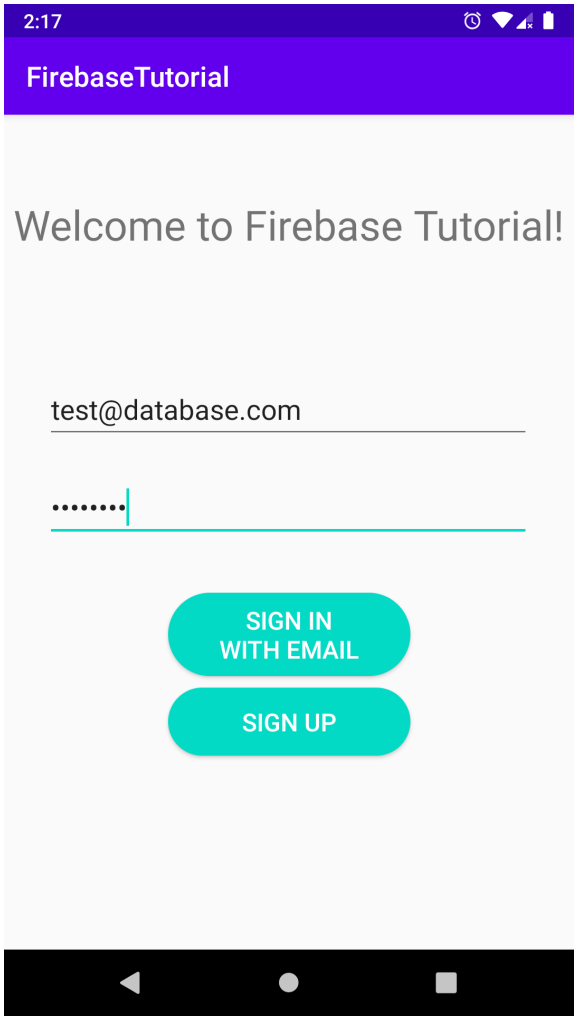
In MainActivity.java comment out or remove Toast message which was displayed if the sign in is successful and paste the following code to start new activity:

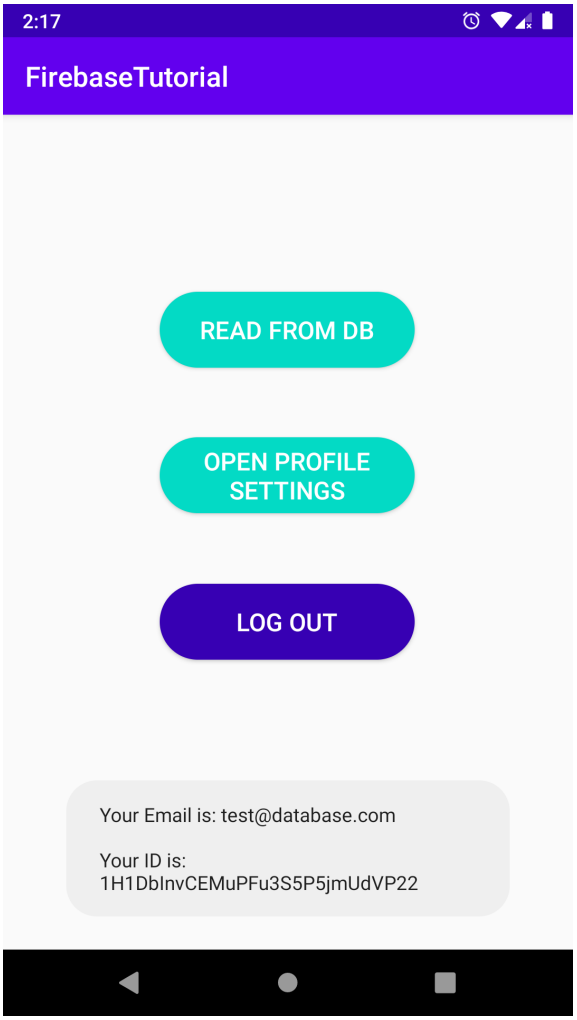
```
// Start next activity and finish current
startActivity(new Intent(MainActivity.this,
        MainPageActivity.class));
finish();
```

And replace *startActivity* method in RegistrationActivity.java with the following:

```
// Start next activity and finish current
startActivity(new Intent(
        RegistrationActivity.this,
        MainPageActivity.class));
```

Now we should be ready to run our app. Here is what we should see:



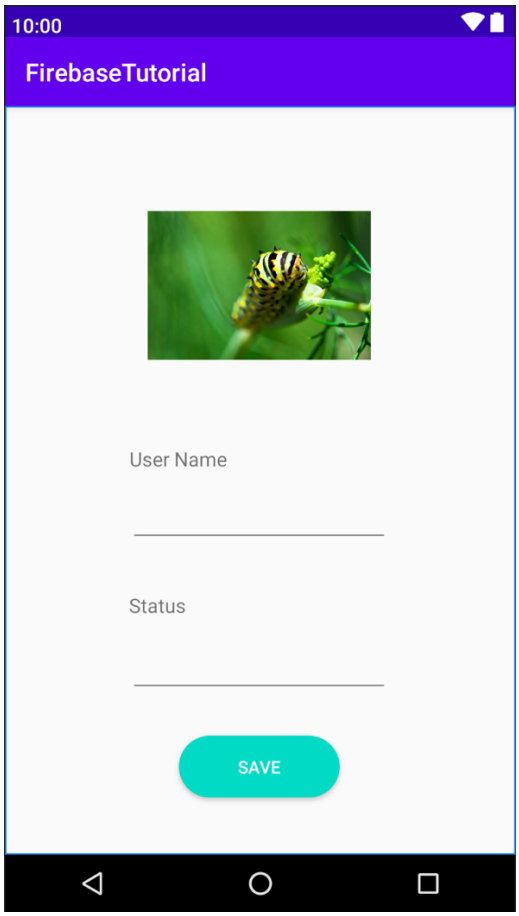


Well, that is not the end. We also want to implement profile settings activity where we can change username, status, and profile image (this in particular we will live for the next part).
Let's create one more activity called *ProfileActivity* and paste the following code into its layout file:

```
<EditText
    android:id="@+id/editTextSettingsUserName"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:ems="10"
    android:inputType="textPersonName"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.497"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/settingTxtName"
    app:layout_constraintVertical_bias="0.053" />

<EditText
    android:id="@+id/editTextSettingsStatus"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:ems="10"
    android:inputType="textPersonName"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.497"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/textView2"
    app:layout_constraintVertical_bias="0.12" />

<TextView
    android:id="@+id/settingTxtName"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="86dp"
    android:layout_marginBottom="396dp"
    android:contentDescription="User Name"
    android:text="User Name"
    android:textSize="16sp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.301"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/profileImage"
    app:layout_constraintVertical_bias="0.347" />

<TextView
    android:id="@+id/textView2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:contentDescription="@string/status"
    android:text="@string/status"
    android:textSize="16sp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.273"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/editTextSettingsUserName"
    app:layout_constraintVertical_bias="0.164" />
```

```
<Button
    android:id="@+id/btnSettingsUpdate"
    android:layout_width="130dp"
    android:layout_height="50dp"
    android:background="@drawable/round_button"
    android:contentDescription="@string/save_settings"
    android:text="@string/save_settings"
    android:textColor="#FFFFFF"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.498"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/editTextSettingsStatus"
    app:layout_constraintVertical_bias="0.416" />

<ImageView
    android:id="@+id/profileImage"
    android:layout_width="180dp"
    android:layout_height="180dp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.497"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.128"
    tools:srcCompat="@tools:sample/backgrounds/scenic" />
```

That's how it should look like:



We won't touch the ImageView for now. In both EditTexts we want the data from our database, and if there are any changes, we want to save them on clicking *Save* button.

Declare the following class fields and copy *onStart* method to this activity:

```
// Database reference
private DatabaseReference mDatabaseReference;
// FirebaseAuth instance to log/sign in users or get info
private FirebaseAuth mAuth;
// Instance of current user
FirebaseUser user;

// Views
ImageView profileImageView;

EditText userNameEditText;
EditText statusEditText;
```

```
@Override
public void onStart() {
    super.onStart();
    // Check if user is signed in (non-null) and update UI accordingly.
    FirebaseUser currentUser = FirebaseAuth.getInstance().getCurrentUser();
    if (currentUser != null) {
        //TODO implement logic after checking user log-in status
    }
}
```

Declare and instantiate the following variables in *onCreate()* method:

```
// Initialize Firebase Auth
mAuth = FirebaseAuth.getInstance();
// Get current user object
user = mAuth.getCurrentUser();
// Get database reference that will point to the DB root
mDatabaseReference = FirebaseDatabase.getInstance().getReference();

// Get all the views
profileImageView = findViewById(R.id.profileImage);

userNameEditText = findViewById(R.id.editTextSettingsUserName);
statusEditText = findViewById(R.id.editTextSettingsStatus);

Button saveButton = findViewById(R.id.btnSettingsUpdate);
```

In this activity we are going to both read and write to the database. So, let's create two methods, one will read data and place it into EditTexts, the other will upload new data to DB if the *Save* button is clicked.

Create two method stubs as shown below:

```
private void readAndDisplayData() {}

private void updateData() {}
```

Let's now implement *readAndDisplayData* method. In general the process of reading data from DB would be same as we already did, except for some minor changes like setting values into EditTexts instead of displaying a ToastMessages.

Paste the following at the beginning of *readAndDisplayData* method:

```
// Get current user's email
String userEmail = user.getEmail();
// Create a query to get current user's data
Query queryUser =
        mDatabaseReference.child("users").orderByChild("email").equalTo(userEmail);
```

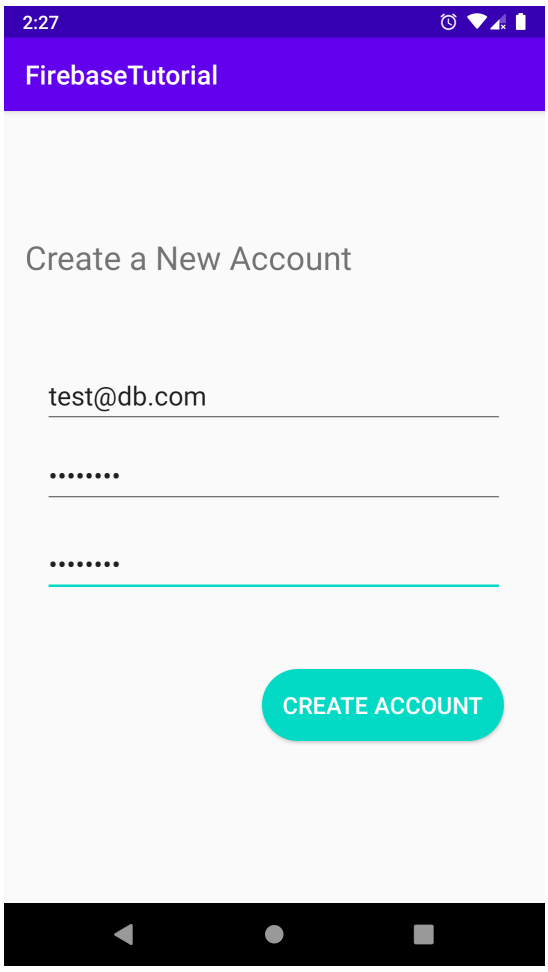
Now we are ready to execute query and here is the rest of the code:

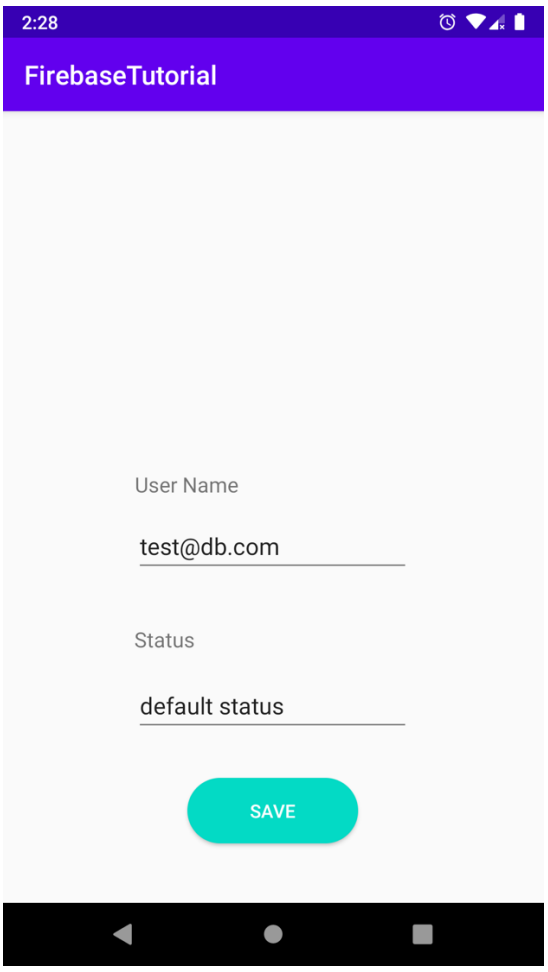
```
queryUser.addValueEventListener(new ValueEventListener() {
    @Override
    public void onDataChange(@NonNull DataSnapshot snapshot) {
        // Check if snapshot that is returned to us exists
        if (snapshot.exists()) {
            // Snapshot is a collection of objects returned to us,
            // you can assume that it's sort of a cursor in some DBMSs
            // We have to iterate through it and get our single result
            for (DataSnapshot snapshotData : snapshot.getChildren()) {
                // Get Username and Status from snapshot object by referencing child
field
                String userName = (String) snapshotData.child("userName").getValue();
                String status = (String) snapshotData.child("status").getValue();
                // Setting values to EditTexts
                userNameEditText.setText(userName);
                statusEditText.setText(status);
            }
        } else {
            // Here is the case when our query was unsuccessful
            // and we could not find any results
            // Set appropriate values in EditTexts
            userNameEditText.setText("No matching entry!");
            statusEditText.setText("No matching entry!");
        }
    }

    @Override
    public void onCancelled(@NonNull DatabaseError error) {
        // This is the case when some error occurred while querying DB
        // We will set appropriate values in EditTexts
        userNameEditText.setText("Error");
        statusEditText.setText("Error");
    }
});
```

In MainPageActivity add the following code to start Profile activity when clicking a button at the end of *onCreate* method:

```
Button openProfileButton = findViewById(R.id.profileButton);
openProfileButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        startActivity(new Intent(MainPageActivity.this,
ProfileActivity.class));
    }
});
```

Now, seems like we are ready to run the app and here what we should get:





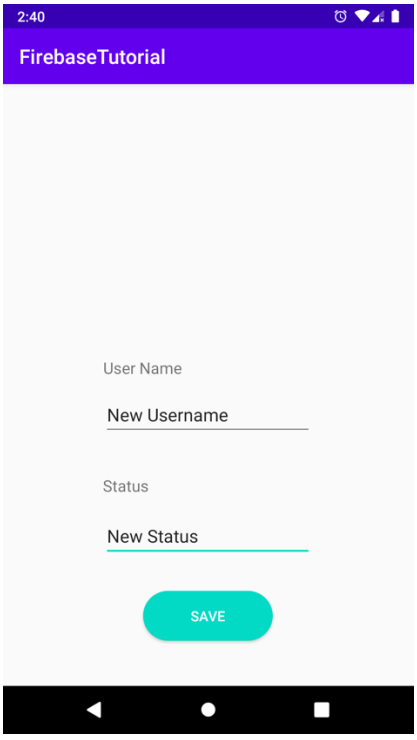
Now let's implement *updateData* method. We will call it if the user clicks '*Save*' button. Let's set up an onClickListener and call the method from there:

```
saveButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        // Update data for current user
        updateData();
        // Finish the activity since we are done here
        finish();
    }
});
```

Implementation of *updateData* is pretty simple: we get the reference to the object in DB by UID and set new values to each child fields:

```
// Get reference to current user object in DB by UID
DatabaseReference currentUserReference =
        mDatabaseReference.child("users").child(user.getUid());
// Get values from EditTexts
String newUserName = userNameEditText.getText().toString();
String newStatus = statusEditText.getText().toString();
// Set value to each 'field'
currentUserReference.child("userName").setValue(newUserName);
currentUserReference.child("status").setValue(newStatus);
```

And here is what we should get:



5. Signing Off and checking for current user's authentication

Signing out the user is a very simple process, we just need to call *signOut()* method on FirebaseAuth instance. We also want the UI to change on main Sign In activity and finish the previous one, since we don't want the user to be able to return. Here is the code for it, paste it in the end of *onCreate()*:

```
Button signOutButton = findViewById(R.id.logout_button);
signOutButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        // Here we sign out the user
        FirebaseAuth.getInstance().signOut();
        // Start log in activity
        startActivity(new Intent(MainPageActivity.this, MainActivity.class));
        // We finish current activity, since we don't want user to return here
        finish();
    }
});
```

Now let's implement our logic to check if the user is authenticated. This is done to make sure that unauthenticated user won't be able to access certain activities, for instance if the user is not authenticated, they shouldn't be able to access profile activity. Make sure that in each activity you have a class variable *FirebaseAuth mAuth.* Decalre it in *onCreate* method by calling *FirebaseAuth.getInstance():*

```
private FirebaseAuth mAuth;
```

```
mAuth = FirebaseAuth.getInstance();
```

After that, we should make a check for user instance in *onStart* method, so here is how it might look:

```
@Override
public void onStart() {
    super.onStart();
    // Check if user is signed in (non-null) and update UI accordingly.
    FirebaseUser currentUser = mAuth.getCurrentUser();
    if (currentUser == null) {
        startActivity(new Intent(getApplicationContext(), MainActivity.class));
    }
}
```

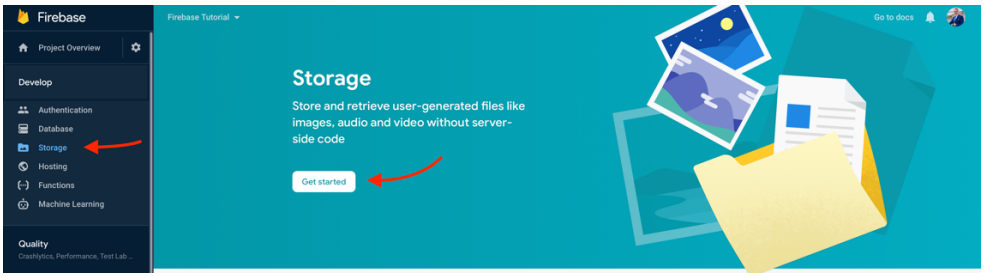
Replace the *startActivity* call for the following in Main and Registration activities:

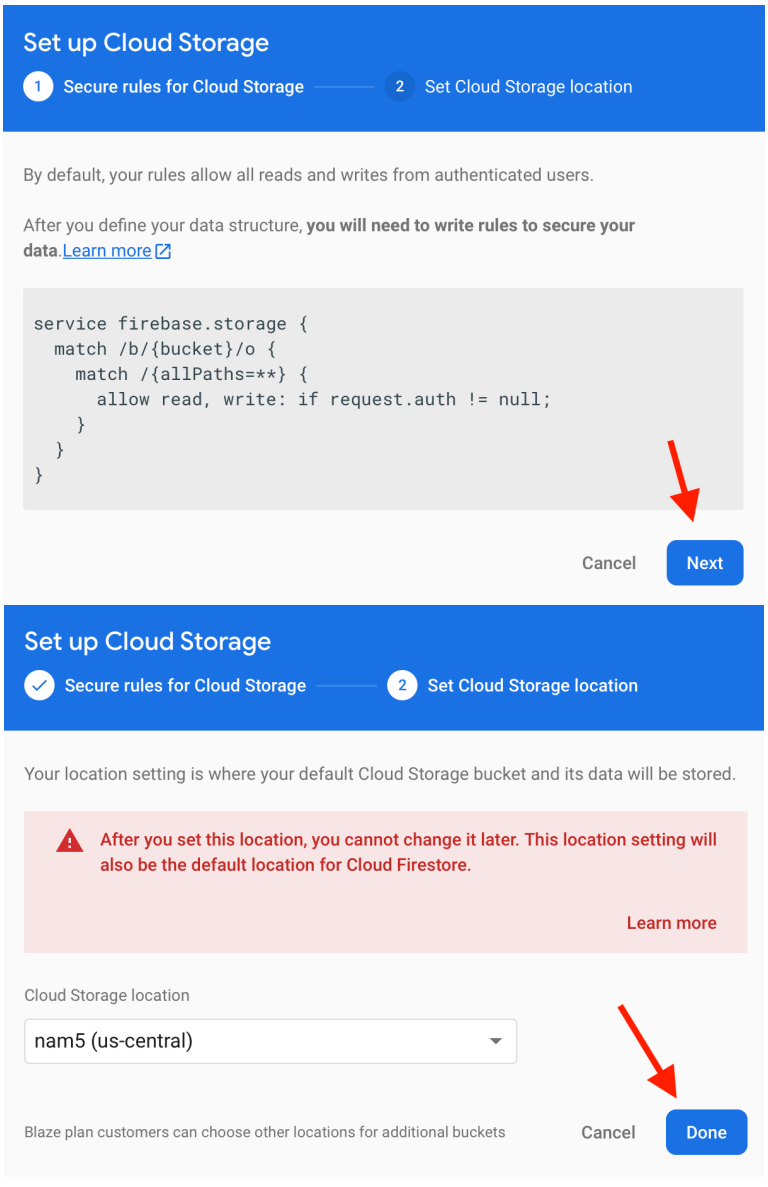
```
if (currentUser != null) {
    startActivity(new Intent(getApplicationContext(), MainPageActivity.class));
}
```

6. Working with Cloud Storage and Uploading Images

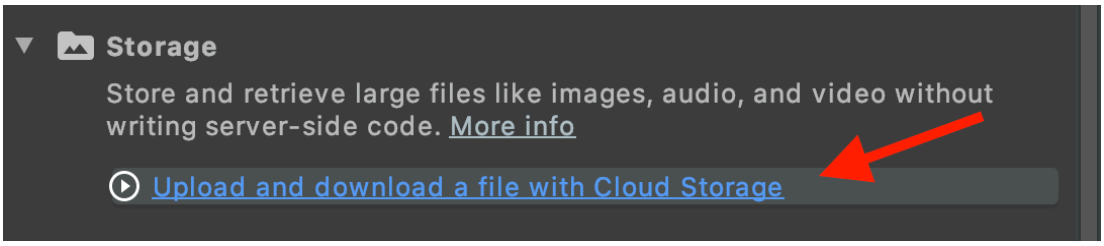
Now what is left to do is to let user update their profile image. For this purpose, we are going to use *Firebase Cloud Storage,* and as always, we need to do some extra set up. Follow visual instructions in Firebase Console and Android Studio:

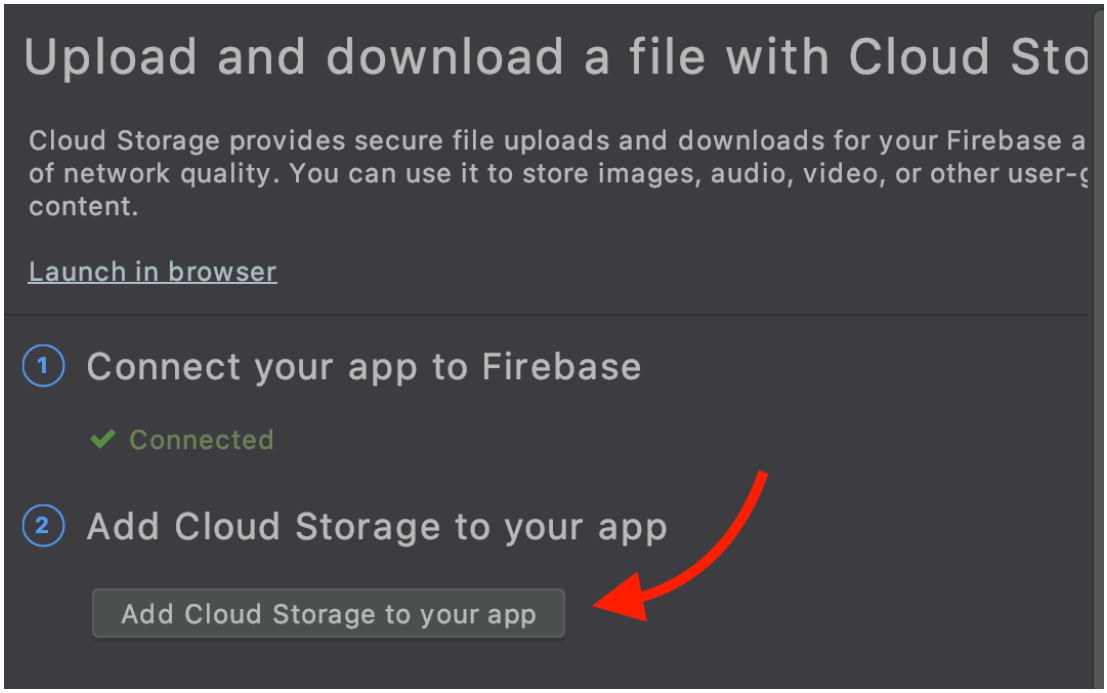
In Firebase Console:

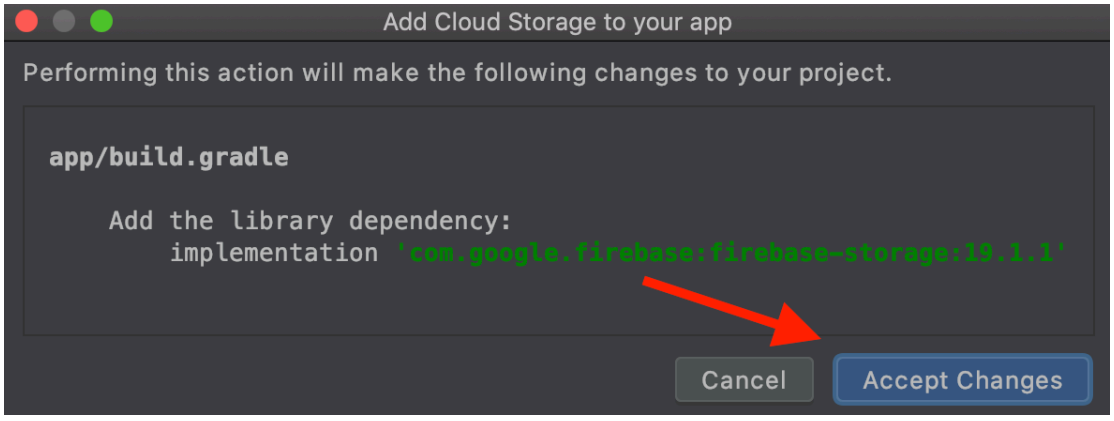




In Android Studio:

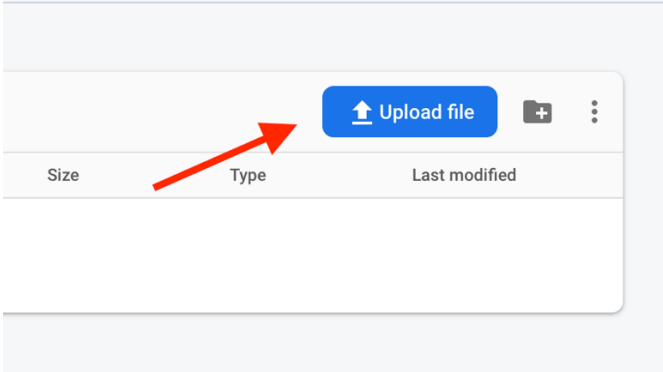




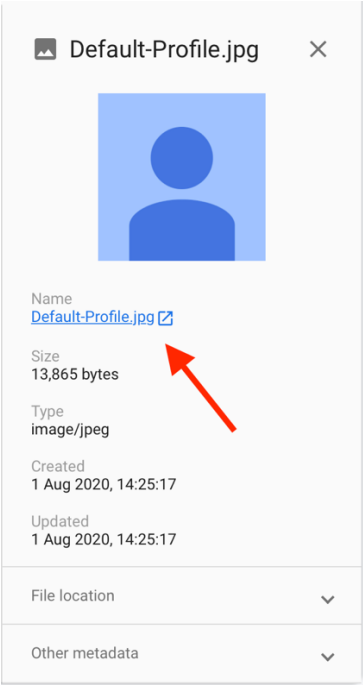


We are now done with set up. Cloud Storage allow us to upload and download file to it, so there is nothing complex about it. Let's get to some code related to it.

When the new account is created, we want to set default profile image to the user. I will use the following image as a default. I'm going to upload it to Storage in Firebase Console:



Now after uploading open the image and copy its link from URL bar:



When registering a new user instead of assigning '*default*' text for *profileImage* field let's set the link to this image. Create a constant value as class variable with the link that you've just copied:

```
// Link to default profile image
private static final String defaultProfileImage = /* LINK TO THE IMAGE IN "" */
```

In *addUserToDB* method replace line appropriate line with:

```
User newUser = new User(id, email, email, "default status", defaultProfileImage);
```

Now, in *ProfileActivity.java* retrieve the link to profile image after retrieving username and status:

```
String profileImage = (String) snapshotData.child("profileImage").getValue();
```

And paste the following code to place our image into appropriate ImageView:

```
// Set profile image
Glide.with(getApplicationContext())
        .load(profileImage)
        .into(profileImageView);
```

Earlier it was a little bit of pain to set the image using a link, now Firebase suggests using Glide when setting images from Cloud Storage. But we have not imported the library yet, so let's do it.
Place the following line into your import at the top of the file:

```
import com.bumptech.glide.Glide;
```

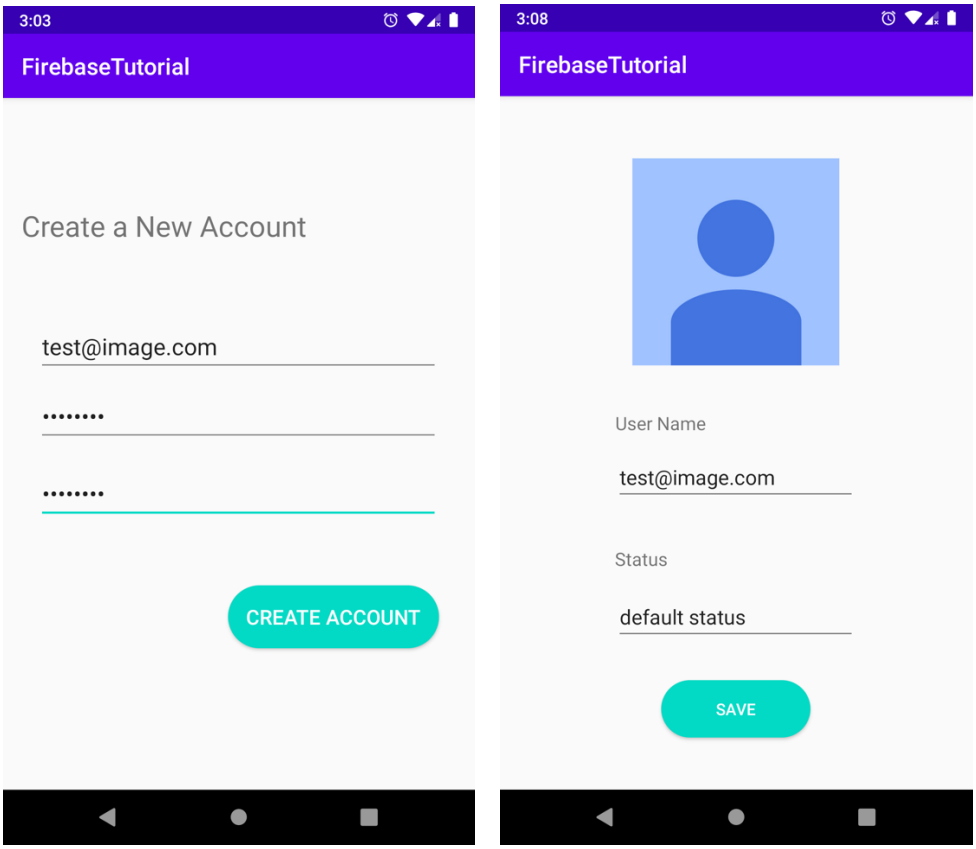
Paste this line your *build.gradle :app* file in dependencies and click '*Sync now*':

```
implementation 'com.firebaseui:firebase-ui-storage:6.2.0'
```

So it should like that now:

```
dependencies {
    implementation fileTree(dir: "libs", include: ["*.jar"])
    implementation 'androidx.appcompat:appcompat:1.1.0'
    implementation 'androidx.constraintlayout:constraintlayout:1.1.3'
    implementation 'com.google.firebase:firebase-auth:19.3.2'
    implementation 'com.google.firebase:firebase-database:19.3.1'
    implementation 'com.google.firebase:firebase-storage:19.1.1'
    testImplementation 'junit:junit:4.12'
    androidTestImplementation 'androidx.test.ext:junit:1.1.1'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.2.0'

//    Add this
    implementation 'com.firebaseui:firebase-ui-storage:6.2.0'
}

```

After that, you should be able to run the app, and here what it can do now:



But this is just the one capability of Cloud Storage. We also want to upload an image. To upload an image, the user should have an ability to select one. So, let's code it!
Add those two lines to the manifest file after Internet permission:

```
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

Then we have to set up an onClickListener for ImageView. Here it is:

```
profileImageView.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {

    }
});
```

In this activity we are going to describe how we are going to select an image.

To make an image selection you can start some systems activities like default gallery or file system and choose a desired image there. Though you can implement some fancy libraries that will allow cropping and other modifications, like Picasa. We are going to do it the first way and here is the code for it:

```
// Create intent to choose from file system
Intent getIntent = new Intent(Intent.ACTION_GET_CONTENT);
getIntent.setType("image/*");
// Create intent to choose from gallery
Intent pickIntent = new Intent(Intent.ACTION_PICK);
pickIntent.setDataAndType(MediaStore.Images.Media.EXTERNAL_CONTENT_URI, "image/*");
// Create chooser intent to select between two options
Intent chooserIntent = Intent.createChooser(getIntent, "Select Image");
chooserIntent.putExtra(Intent.EXTRA_INITIAL_INTENTS, new Intent[]{pickIntent});
// Start activity chooser to get result from it
startActivityForResult(chooserIntent, PICK_IMAGE);
```

The requirement is to implement *onActivityResult* method, since it will return us an image the user has selected. So, as soon as we have an image we want to upload to the storage and update the link for it in the DB.
By the way, you can see that *PICK_IMAGE* is red right now, because we haven't created it yet. Place this line in the beginning of current activity:

```
// Code for picking image
public static final int PICK_IMAGE = 252;
```

In our *onActivityResult* we get the data from the activity, create a reference in the storage, put it there, and then update the reference to the image in the DB. During the process you can see multiple event listeners there, they are mainly to notify the user about what is going on, and only in *onCompleteListener* we check if the task is successful and only then put the reference for the image into DB.

Also create this member variable for storage reference and declare it *onCreate():*

```
private StorageReference mStorageReference;
```

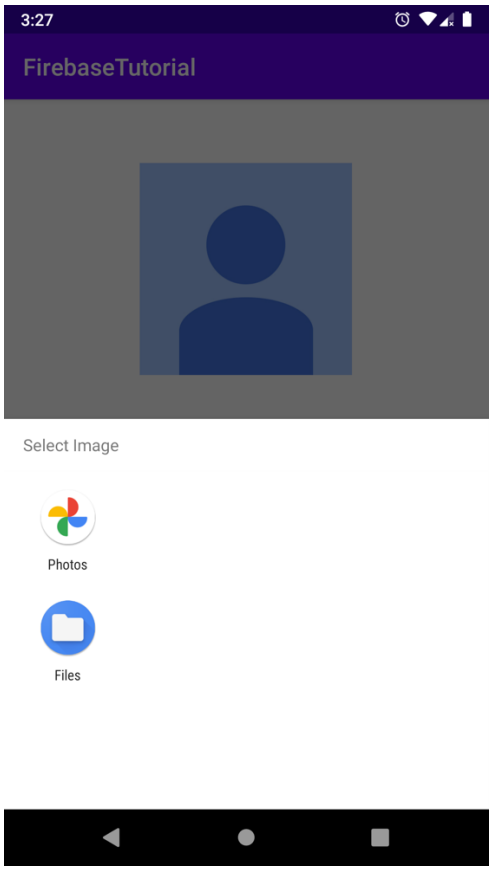
```
mStorageReference = FirebaseStorage.getInstance().getReference("profile_images");
```

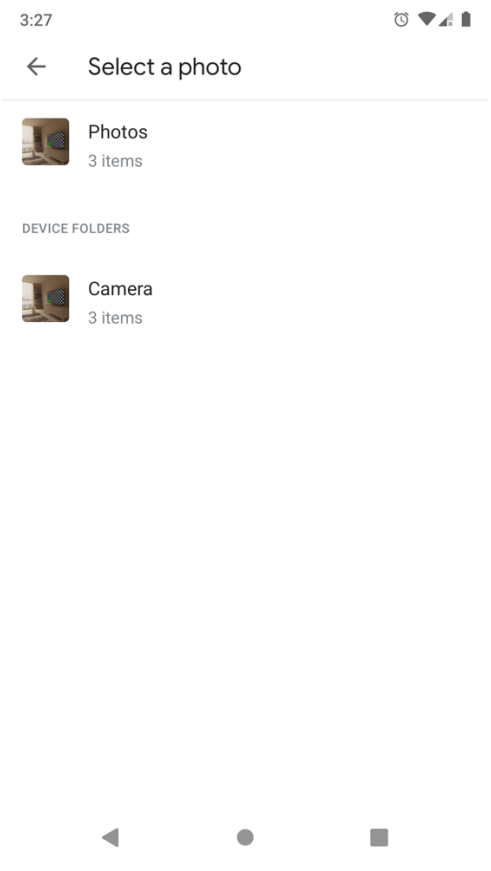
Here is the code for *onActivityResult* method:

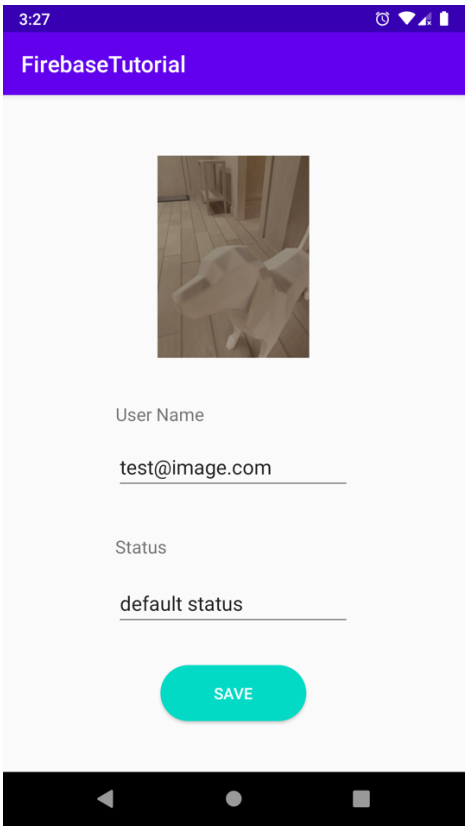
--see next page--

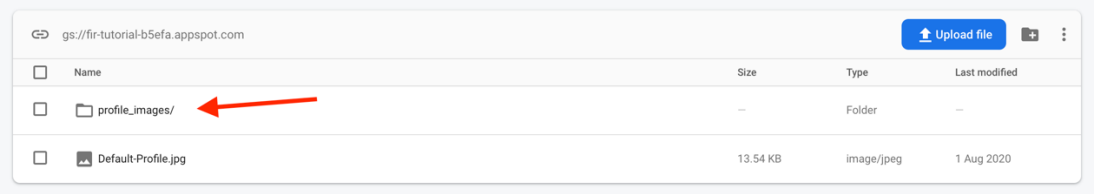
```java
@Override
public void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    // If the activity that brings the result is the one we've requested
    if (requestCode == PICK_IMAGE && resultCode == Activity.RESULT_OK) {
        if (data != null) {
            // Upload image to Firebase Storage
            // Extract Image resource from what
            // the user has selected
            Uri imageUri = data.getData();
            // Set image to the ImageView
            profileImageView.setImageURI(imageUri);
            // Create a file name consisting of user ID + file extension
            String fileName = user.getUid() + ".jpg";
            // Create s reference (spot) for selected image
            final StorageReference fileReference = mStorageReference.child(fileName);
            // Put the file to the reference
            fileReference
                    .putFile(imageUri)
                    .addOnSuccessListener(new
OnSuccessListener<UploadTask.TaskSnapshot>() {
                        @Override
                        public void onSuccess(UploadTask.TaskSnapshot taskSnapshot) {
                            Toast.makeText(ProfileActivity.this,
                                    "Image Updated Successfully!",
                                    Toast.LENGTH_LONG).show();
                        }
                    })
                    .addOnCompleteListener(new
OnCompleteListener<UploadTask.TaskSnapshot>() {
                        @Override
                        public void onComplete(@NonNull Task<UploadTask.TaskSnapshot>
task) {
                            if (task.isSuccessful()) {
fileReference.getDownloadUrl().addOnSuccessListener(new OnSuccessListener<Uri>() {
                                    @Override
                                    public void onSuccess(Uri uri) {
                                        // Download uri for an image
                                        String downloadUri = uri.toString();
                                        // Set value in DB
                                        mDatabaseReference.child("users")
.child(user.getUid()).child("profileImage").setValue(downloadUri);
                                    }
                                });
                            }
                        }
                    })
                    .addOnFailureListener(new OnFailureListener() {
                        @Override
                        public void onFailure(@NonNull Exception e) {
                            Toast.makeText(ProfileActivity.this, e.getMessage(),
Toast.LENGTH_LONG).show();
                        }
                    });
        } else {
            Toast.makeText(this, "Could Not Select the Image",
Toast.LENGTH_LONG).show();
        }
    } else {
        Toast.makeText(this, "You Have Not Selected an Image",
Toast.LENGTH_LONG).show();
    }
}
```

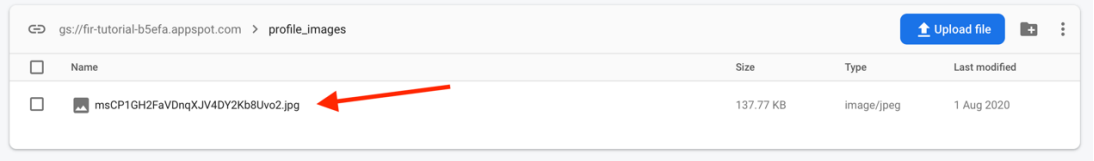
Now we can run our app and select a new image:

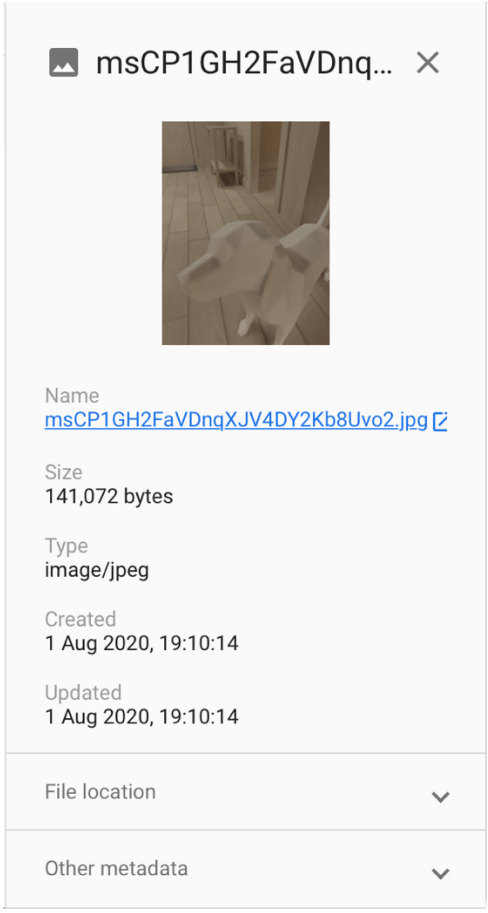












We have successfully updated profile image!

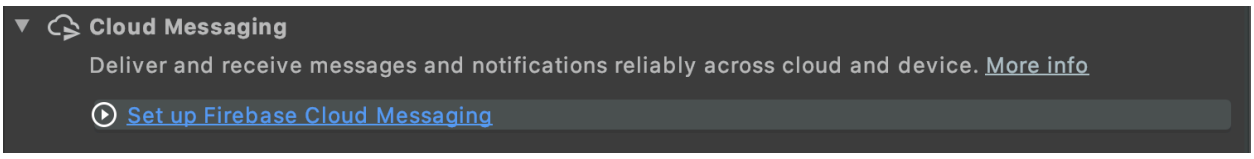
7. A little about Cloud Firestore

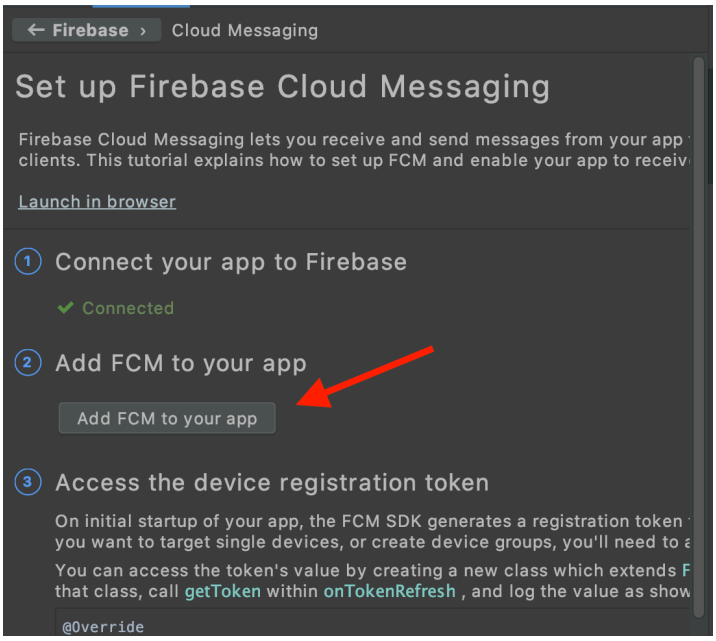
Firebase Cloud Firestore is another NoSQL database offered by Firebase. Using it is probably a more modern approach of using Firebase databases, though it really depends on the project and its scale. As per Firebase guide, it is recommended to perform a detailed analysis to determine which database is a better fit for someone's project. Here is a tool by Google to determine which option is better. In general, Realtime Database is good for most kind of the projects, but Cloud Firestore is much better for massive projects where large amount of storage is needed, frequent database interactions take place, and high uptime (availability) is required. You can familiarize yourself here with how does Cloud Firestore work and how to implement it in your application. Also, don't forget to check sample code in Firebase Assistant in Android Studio.

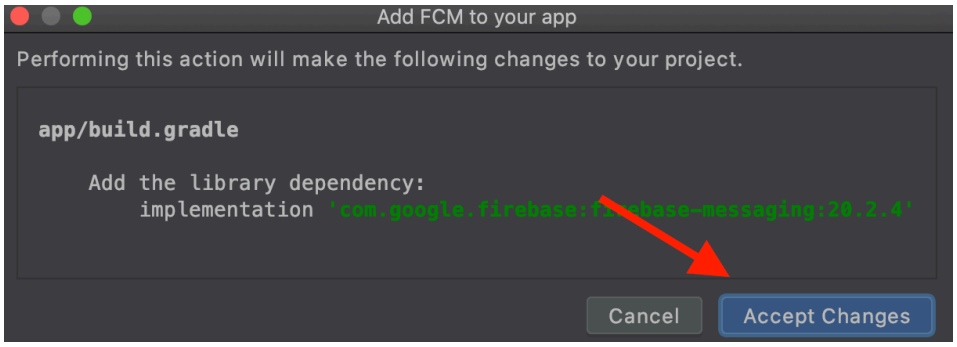
8. About Cloud Messaging

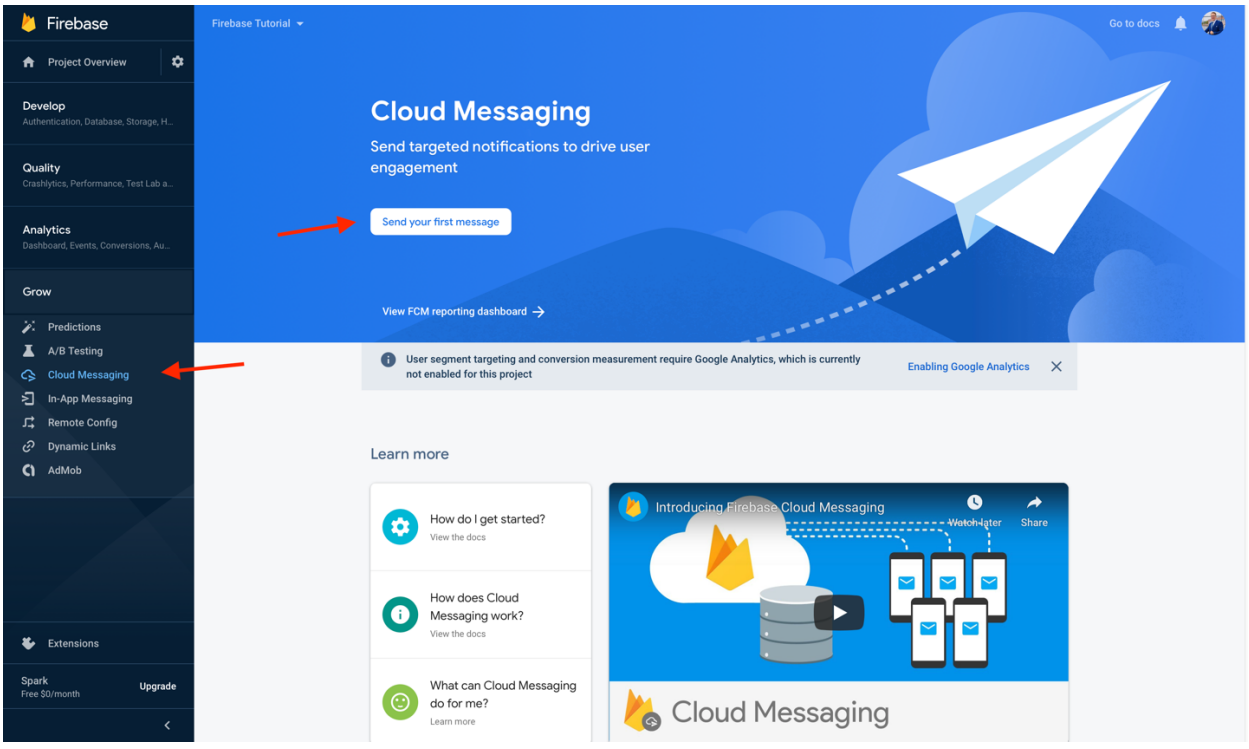
Cloud Messaging is a powerful tool to send notifications to users. The technology subdivides users into three categories: unique users, user groups, and users subscribed on topics. It requires you, as a developer, to implement the logic on your server to "explain" Firebase which users to send messages and when. Its bandwidth is very high; thus, the tool is very powerful and efficient. It is also a good practice for developers to process those messages or notifications on their own using Android SDK, therefore Cloud Messaging is just a very efficient service of carrying any notifications you like. Although we are not going to cover this topic in deep because it is far out of scope of this tutorial, we still can try to send some simple notification to our app from Firebase Console. Feel encouraged to familiarize yourself with the concept in this guide by Google.

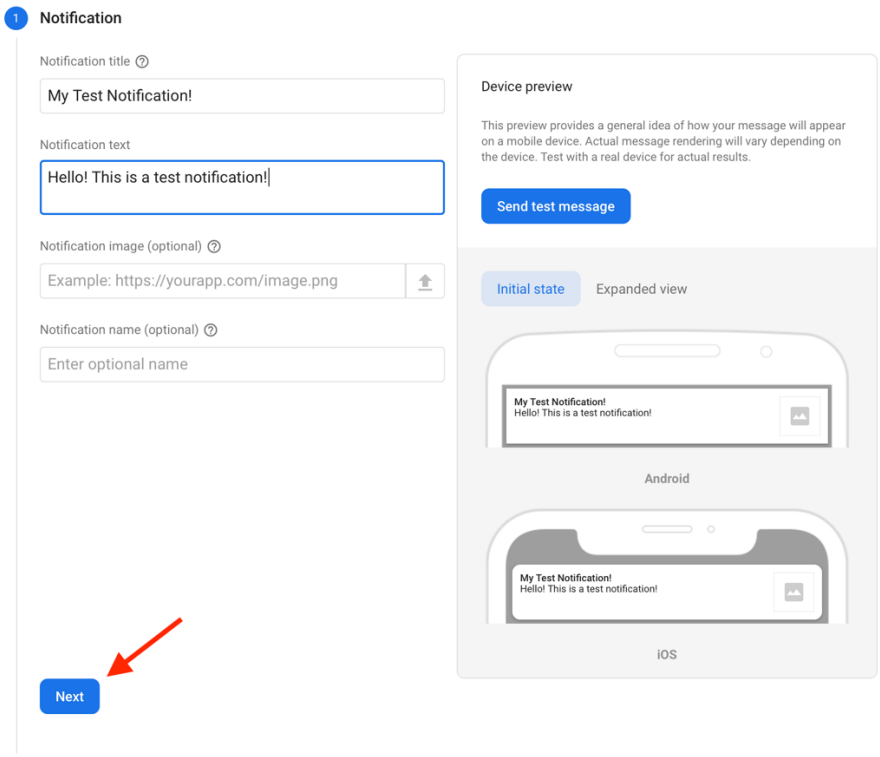
Here is how we can do it. Make sure you run your device first but not the app itself before sending notification. Follow this visual instruction in Firebase Console and Android Studio:

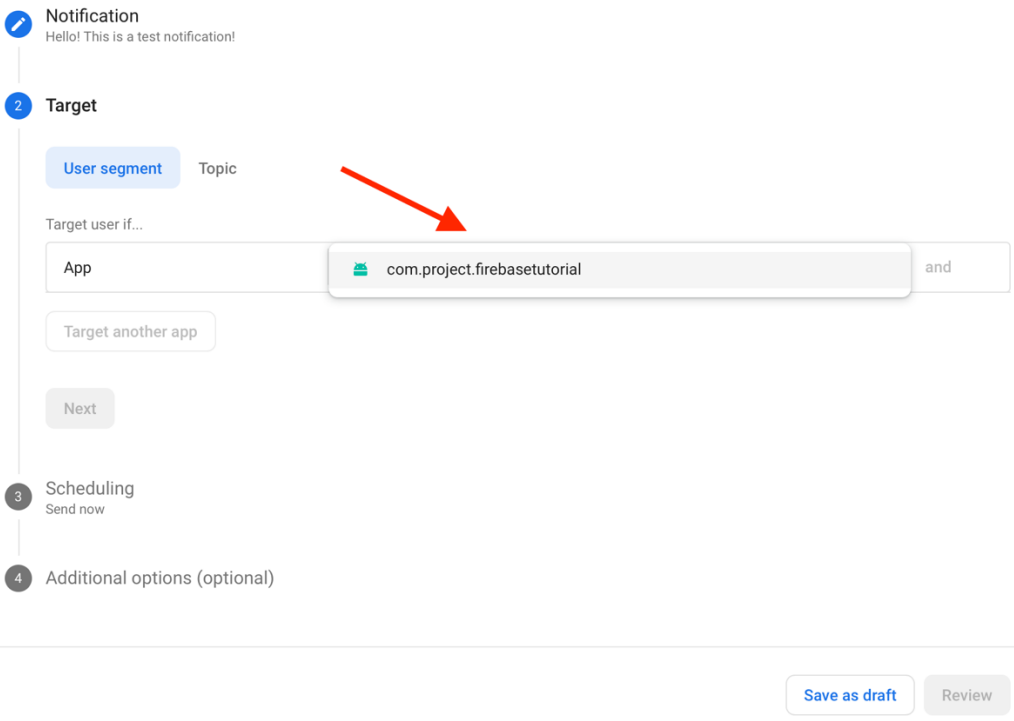


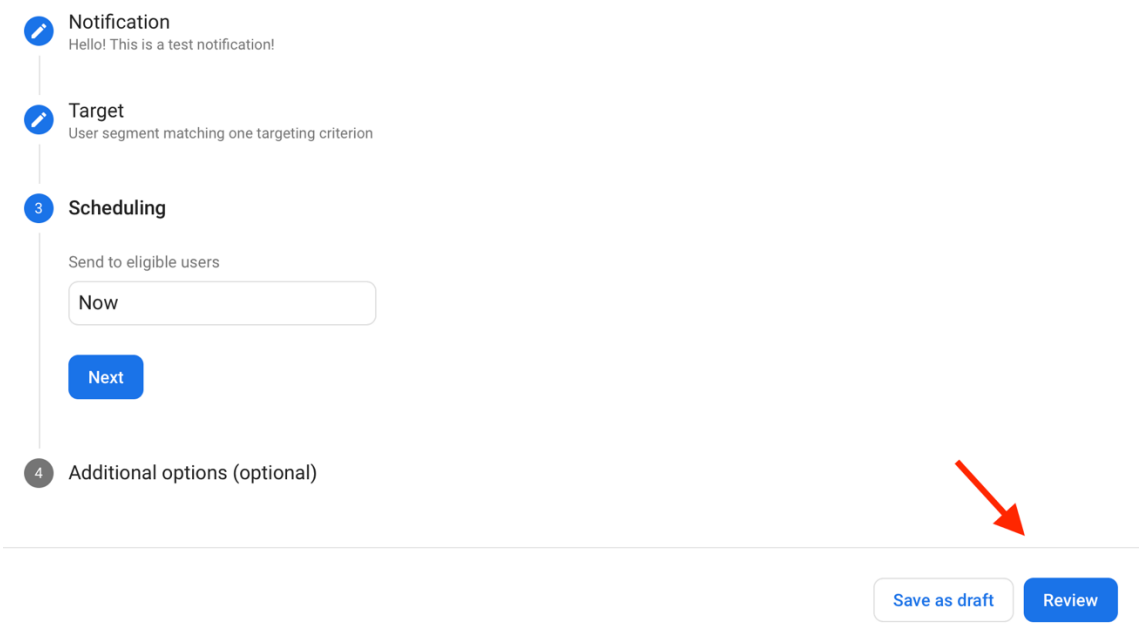

Firebase › Cloud Messaging
Set up Firebase Cloud Messaging
Firebase Cloud Messaging lets you receive and send messages from your app
clients. This tutorial explains how to set up FCM and enable your app to receiv
Launch in browser
1 Connect your app to Firebase
Connected
2 Add FCM to your app
Add FCM to your app
3 Access the device registration token
On initial startup of your app, the FCM SDK generates a registration token
you want to target single devices, or create device groups, you'll need to a
You can access the token's value by creating a new class which extends F
that class, call getToken within onTokenRefresh , and log the value as show
@Override

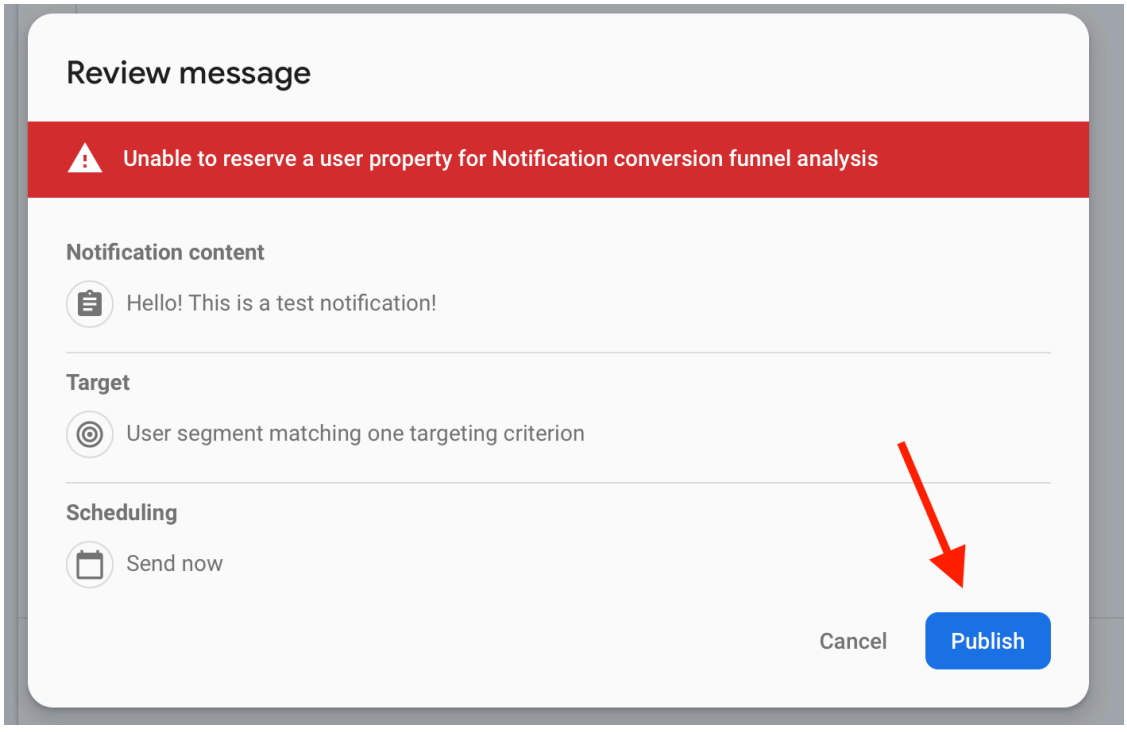

Add FCM to your app
Performing this action will make the following changes to your project.
app/build.gradle
Add the library dependency:
implementation 'com.google.firebase:f
base-messaging:20.2.4'
Cancel
Accept Changes


Firebase
Project Overview
Develop
Authentication, Database, Storage, H...
Quality
Crashlytics, Performance, Test Lab a...
Analytics
Dashboard, Events, Conversions, Au...
Grow
Predictions
A/B Testing
Cloud Messaging
In-App Messaging
Remote Config
Dynamic Links
AdMob
Extensions
Spark
Free $0/month
Upgrade
Firebase Tutorial
Go to docs
Cloud Messaging
Send targeted notifications to drive user engagement
Send your first message
View FCM reporting dashboard
User segment targeting and conversion measurement require Google Analytics, which is currently not enabled for this project
Enabling Google Analytics
Learn more
How do I get started?
View the docs
How does Cloud Messaging work?
View the docs
What can Cloud Messaging do for me?
Learn more
Introducing Firebase Cloud Messaging
Watch later
Share
Cloud Messaging

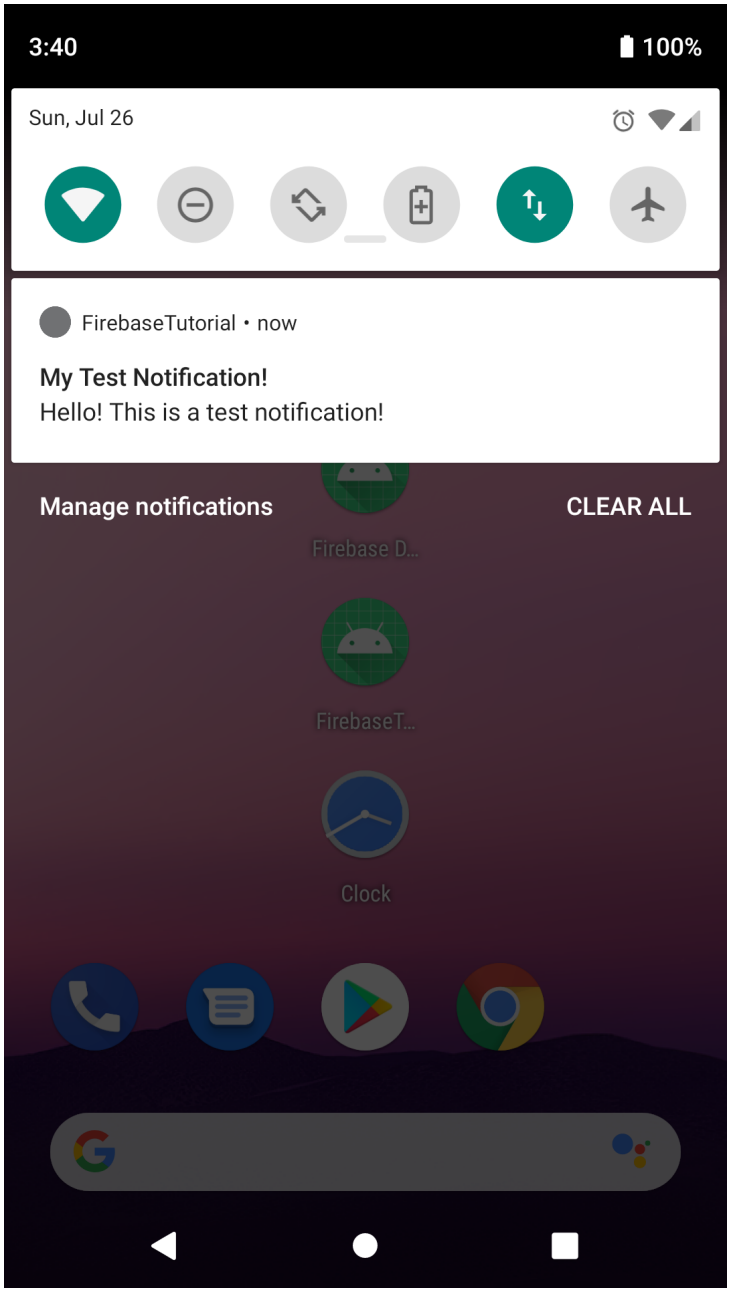

1
Notification
Notification title
My Test Notification!
Notification text
Hello! This is a test notification!
Notification image (optional)
Example: https://yourapp.com/image.png
Notification name (optional)
Enter optional name
Next
Device preview
This preview provides a general idea of how your message will appear on a mobile device. Actual message rendering will vary depending on the device. Test with a real device for actual results.
Send test message
Initial state
Expanded view
My Test Notification!
Hello! This is a test notification!
Android
My Test Notification!
Hello! This is a test notification!
iOS


Notification
Hello! This is a test notification!
2
Target
User segment
Topic
Target user if...
App
com.project.firebasetutorial
and
Target another app
Next
3
Scheduling
Send now
4
Additional options (optional)
Save as draft
Review





And here is the result:



The notification is received, and if the user clicks it, the application will open up.

9. Conclusion

Here is the end of this tutorial. Firebase is a very powerful platform, offering various tools for various needs. I highly recommend advance your knowledge in regard to this topic, since cloud infrastructures are everywhere nowadays.
You can also try to create some new projects and try to implement more complex features, or try the ones covered here in a more advanced and applicative manner (i.e. some chatting application implemented with database, or some online service that requires file exchange and interaction with file storage).

I wish you success in your learning and never give up!