HandsOn Cloud Firestore

Jigisha Patel

Agenda

- Setting up development environment
- CRUD using Cloud Firestore

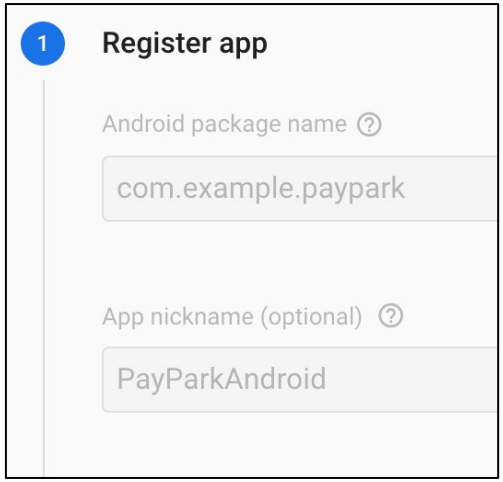
Integrating Firebase in Android project

1. Create Firebase Project
2. Register your app with Firebase
3. Add Firebase Configuration File
4. Add Firebase SDKs to your app
5. Initialize Firebase in your app

- Find the detailed instruction here.

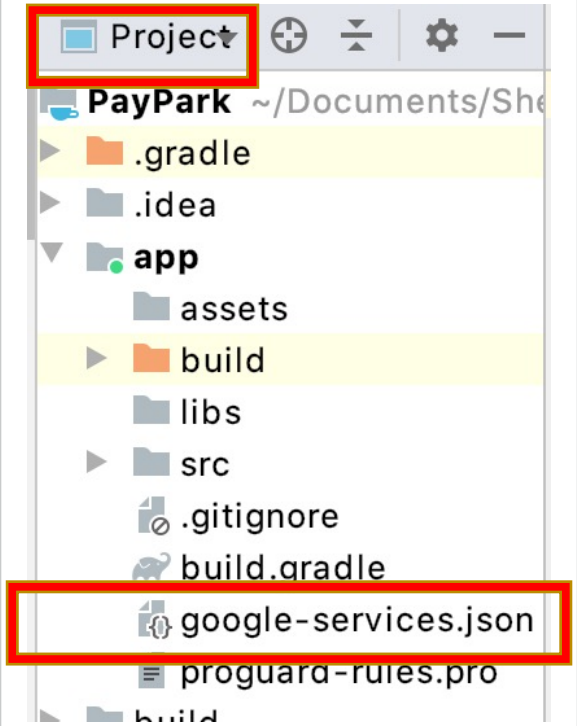
Step 1 : Create Firebase Project

1. In the Firebase console, click **Add project**
2. Enter a **Project name**
3. *(Optional)* If you are creating a new project, you can edit the **Project ID**.
4. Click **Continue**.
5. *(Optional)* Set up Google Analytics for your project
6. Click **Create project**



Step 2 : Register your app with Firebase

1. In the Firebase console, click on the project name you just created to visit the **Project Overview** page.
2. On the project overview page, click the **`Android`** icon to launch the setup workflow.
3. Enter your app's package name.
4. Click **Register App**.



Step 3 : Add Firebase Configuration File

1. Download the **`google-services.json`** file
2. In Android Studio project explorer, change the display type from Android to Project.
3. Copy the downloaded file and paste it under the app package.

Step 4 : Add Firebase dependency to your app

- In your **root-level (project-level) Gradle file**, add rules to include the Google Services Gradle plugin.
- Check that you have Google's Maven repository, as well.

```
buildscript {

    repositories {
        // Check that you have the following line (if not, add it):
        google()  // Google's Maven repository
    }

    dependencies {
        // ...

        // Add the following line:
        // Google Services plugin
        classpath 'com.google.gms:google-services:4.3.4'
    }
}

allprojects {
    // ...

    repositories {
        // Check that you have the following line (if not, add it):
        google()  // Google's Maven repository
        // ...
    }
}
```

- In your module (app-level) Gradle file, apply the Google Services Gradle plugin.

```
apply plugin: 'com.google.gms.google-services'
```

Step 5 : Add Firebase SDKs to your app

- In your module (app-level) Gradle file, using the Firebase Android BoM, declare the dependencies for the Firebase products that you want to use in your app.
- Sync your app to ensure that all dependencies have the necessary versions.

```
dependencies {
    // Import the BoM for the Firebase platform
    implementation platform('com.google.firebase:firebase-bom:26.4.0')

    // Declare the dependency for the Cloud Firestore library
    // When using the BoM, you don't specify versions in Firebase library dependencies
    implementation 'com.google.firebase:firebase-firestore'
}
```

Integrating Cloud Firestore in Android project

1. In the Firebase console, from the Project Overview page, select **Cloud Firestore**
2. Select **Create database**
3. Select a starting mode for your Cloud Firestore Security Rules:
   - Product mode – private data
   - **Test mode** – open data **[Select this option]**

- Find the detailed instruction here.

CRUD using Cloud Firestore

Initialize Cloud Firestore from app

- Before you begin add, retrieve, update or delete operations to/from database, you will need to initialize the database instance from your app.

```
// Access a Cloud Firestore instance from your Activity
FirebaseFirestore db = FirebaseFirestore.getInstance();
```

Add data to Cloud Firestore

- There are several ways to write data to Cloud Firestore:
  - Set the data of a document within a collection, explicitly specifying a document identifier.
  - Add a new document to a collection. In this case, Cloud Firestore automatically generates the document identifier.
  - Create an empty document with an automatically generated identifier, and assign data to it later

Custom Objects

- Cloud Firestore supports document creation from the custom classes.
- The custom class must contain data types which are compatible with Cloud Firestore data types.
- Cloud Firestore converts the objects to supported data types.

Example: Custom Class

```
public class City {


    private String name;
    private String state;
    private String country;
    private boolean capital;
    private long population;
    private List<String> regions;

    public City() {}

    public City(String name, String state, String country, boolean capital, long
            population, List<String> regions) {
        // ...
    }

    //getters and setters
```

Set a document

- You can use `set()` method to create a new document in the Cloud Firestore collection by specifying the id for the new document.
- If the specified collection doesn't exist on the Firestore, it will be created.

Example

```
Map<String, Object> city = new HashMap<>();
city.put("name", "Los Angeles");
city.put("state", "CA");
city.put("country", "USA");

db.collection("cities").document("LA")
    .set(city)
    .addOnSuccessListener(new OnSuccessListener<Void>() {
        @Override
        public void onSuccess(Void aVoid) {
            Log.d(TAG, "DocumentSnapshot successfully written!");
        }
    })
    .addOnFailureListener(new OnFailureListener() {
        @Override
        public void onFailure(@NonNull Exception e) {
            Log.w(TAG, "Error writing document", e);
        }
    });
```

Add a document

- While adding a data through set method, you have to specify the id of the document to be added.

- When you don’t have any id to be specified or you want Firestore to generate the id for you, you can use the add() method to add the document to Firestore.

Example

```
// Create a new user with a first and last name
Map<String, Object> user = new HashMap<>();
user.put("first", "Ada");
user.put("last", "Lovelace");
user.put("born", 1815);

// Add a new document with a generated ID
db.collection("users")
    .add(user)
    .addOnSuccessListener(new OnSuccessListener<DocumentReference>() {
        @Override
        public void onSuccess(DocumentReference documentReference) {
            Log.d(TAG, "DocumentSnapshot added with ID: " + documentReference.getId());
        }
    })
    .addOnFailureListener(new OnFailureListener() {
        @Override
        public void onFailure(@NonNull Exception e) {
            Log.w(TAG, "Error adding document", e);
        }
    });
```

Retrieving document

- You can use **SnapshotListener** to get realtime updates from Cloud Firestore.
- An initial call using the callback you provide creates a document snapshot immediately with the current contents of the single document.
- Then, each time the contents change, another call updates the document snapshot.
- The retrieved document change will provide a document which can be converted into custom object to fetch individual field values.

Example: Snapshot Listener

```
final DocumentReference docRef = db.collection("cities").document("SF");
docRef.addSnapshotListener(new EventListener<DocumentSnapshot>() {
    @Override
    public void onEvent(@Nullable DocumentSnapshot snapshot,
                        @Nullable FirebaseFirestoreException e) {
        if (e != null) {
            Log.w(TAG, "Listen failed.", e);
            return;
        }

        if (snapshot != null && snapshot.exists()) {
            Log.d(TAG, "Current data: " + snapshot.getData());
        } else {
            Log.d(TAG, "Current data: null");
        }
    }
});
```

Update a document

- You can update some fields of a document without overwriting the entire document by using **update()** method.

Example: Update a document

```
DocumentReference washingtonRef = db.collection("cities").document("DC");

// Set the "isCapital" field of the city 'DC'
washingtonRef
    .update("capital", true)
    .addOnSuccessListener(new OnSuccessListener<Void>() {
        @Override
        public void onSuccess(Void aVoid) {
            Log.d(TAG, "DocumentSnapshot successfully updated!");
        }
    })
    .addOnFailureListener(new OnFailureListener() {
        @Override
        public void onFailure(@NonNull Exception e) {
            Log.w(TAG, "Error updating document", e);
        }
    });
```

Delete a document

- You can use **delete()** method to delete the document once you have identified it.

- The delete() method **does not** delete any subcollections of a document.

Example: Delete a document

```
db.collection("cities").document("DC")
    .delete()
    .addOnSuccessListener(new OnSuccessListener<Void>() {
        @Override
        public void onSuccess(Void aVoid) {
            Log.d(TAG, "DocumentSnapshot successfully deleted!");
        }
    })
    .addOnFailureListener(new OnFailureListener() {
        @Override
        public void onFailure(@NonNull Exception e) {
            Log.w(TAG, "Error deleting document", e);
        }
    });
```

References

- https://firebase.google.com/docs/android/setup
- https://firebase.google.com/docs/firestore/quickstart
- https://firebase.google.com/docs/firestore
- https://firebase.google.com/docs/firestore/data-model
- https://firebase.google.com/docs/firestore/manage-data/add-data
- https://firebase.google.com/docs/firestore/manage-data/delete-data
- https://firebase.google.com/docs/firestore/query-data/listen