LECTURE 9: INTEGRATING FIREBASE INTO A FLUTTER APP (FIREBASE PART 1)

WHAT YOU WILL LEARN IN THIS CHAPTER

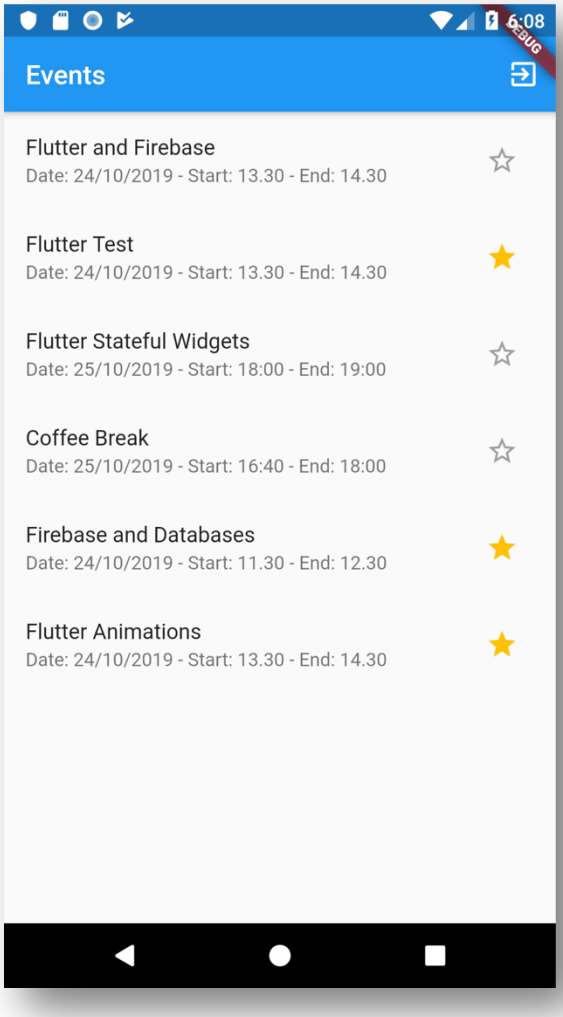
- Creating a Firebase project
- Adding Firebase and Firestore to your app
- Creating the data model

INTRODUCING FIREBASE

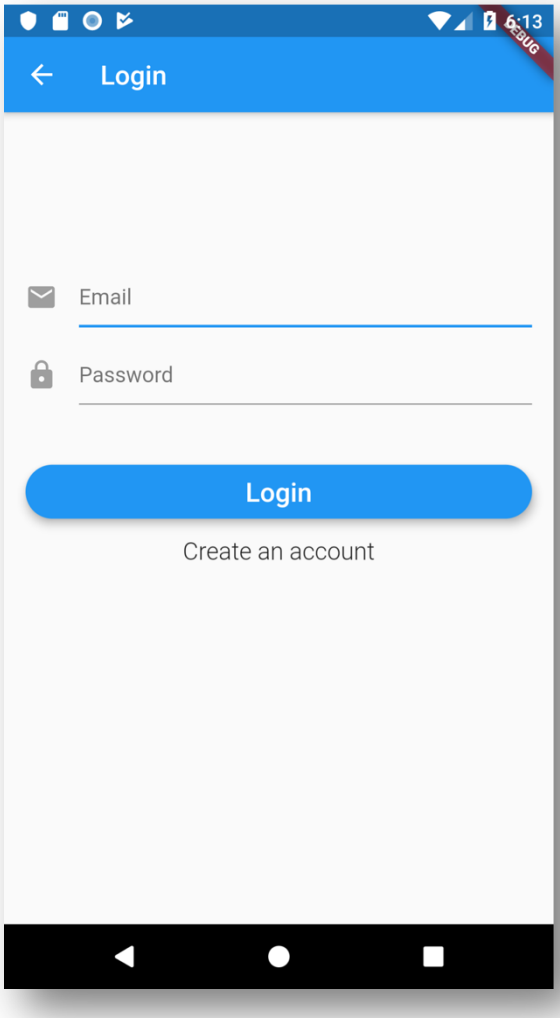
- Firebase is a set of tools with which to build scalable applications in the cloud. Among those tools, you'll find **authentication, storage, databases, notifications,** and **hosting**.
- You can actually choose between two databases: the **Realtime Database** and **Firestore Database**. In this chapter, we'll be using the **Firestore Database**, which is a NoSQL document database that simplifies storing, querying, and updating data in the cloud.
- You can use Firebase as the backend of your iOS and Android apps, with no need to write the code for a web service and, in many cases, without writing any code at all for your server-side service.
- In a NoSQL database, **all documents are JSON documents**, and, theoretically, each document could have different fields and values (or key-value pairs). For instance, in a users collection, the first user might have a **user_id**, **name**, and **password**, but another user could also contain a **user_role** field or a **user_age** field. Both documents would still be valid.
    - This is **not possible** in a **relational database**, where all records must follow a fixed schema – all records must have the same fields.

PROJECT OVERVIEW

- We'll build an event app where the user will be able to see the program of an event, with the details of the schedule. This project will take **four lectures to complete** and today we will complete only the first part of the app.
- All data will be hosted remotely, in a Firebase project. The events will be stored in a **Firestore Database**.
- **Once authenticated**, users will be able to choose their favorite parts of the event by pressing a star icon. In this way, the "favorites" will be also saved remotely.
- The following screenshot shows the app's main screen when completed:

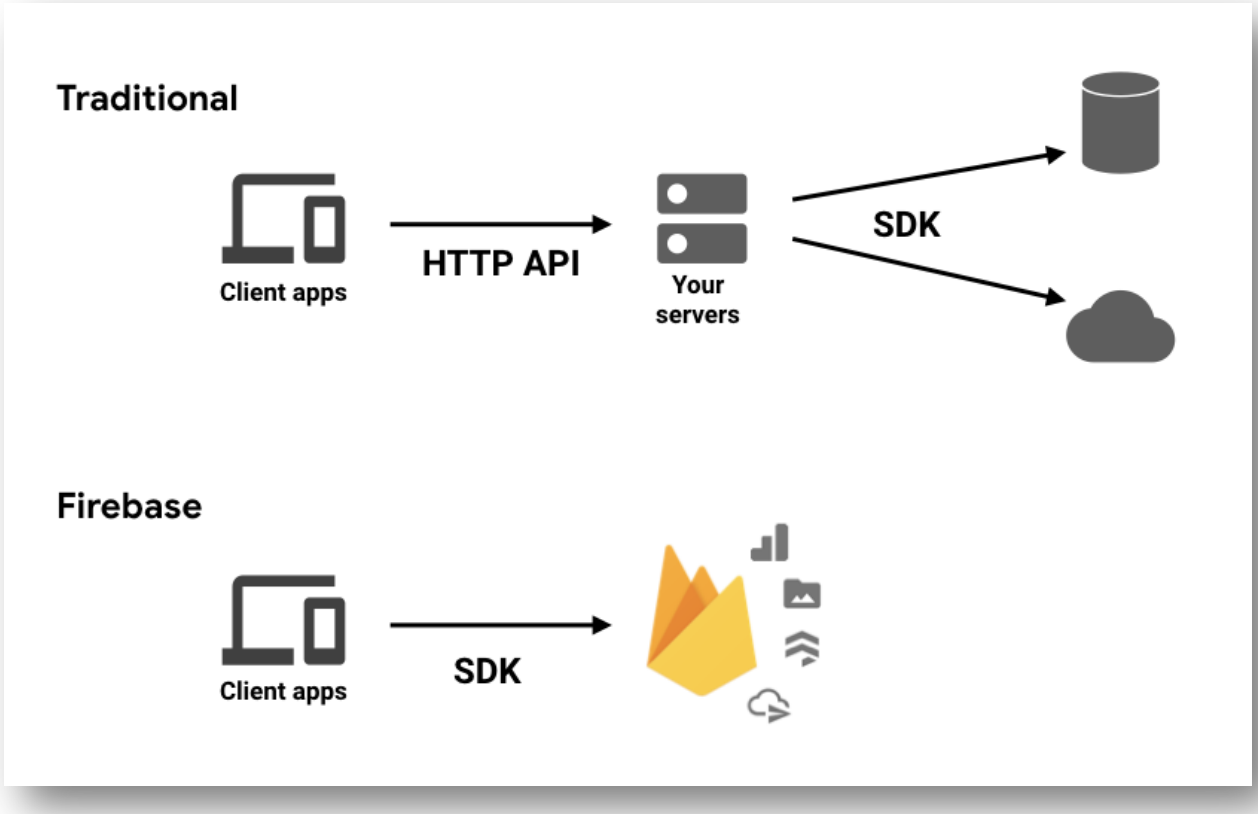


- Another interesting aspect of the app is dealing with **authentication**. This is generally a cumbersome process, but the good news is that dealing with authentication with Firebase and Flutter is rather straightforward. We will complete the **authentication part in a later lecture**. Here is the app's authentication screen in the following screenshot:



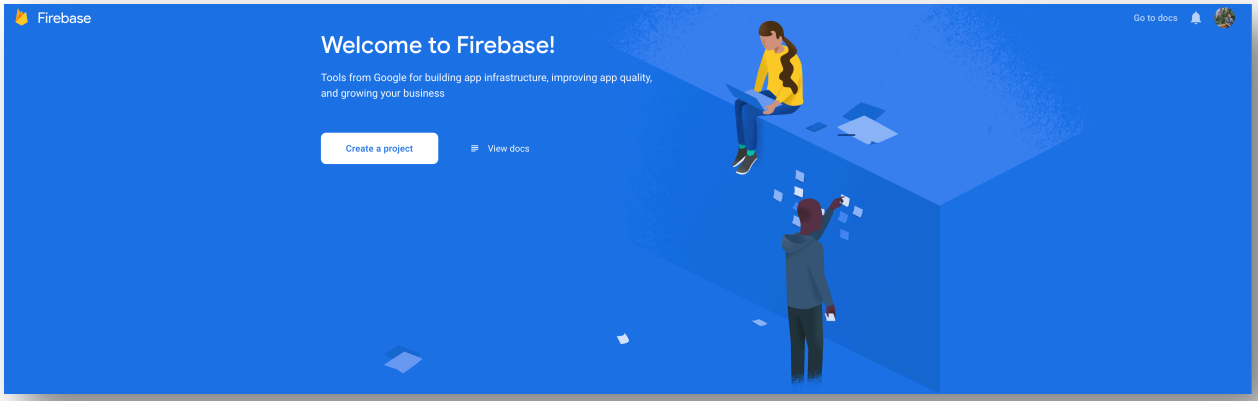
ADDING FIREBASE TO YOUR FLUTTER PROJECT

- What are the advantages of using Firebase, instead of following the traditional approach of writing a client-side app, and a server-side (or backend) service?
- In traditional app development, you need to write **both client and server software yourself** when developing an app that requires a backend service.
- However, with Firebase, the **main advantage** is that you won't need to write, install, or maintain a web service using PHP, Java or C#. You'll deal with Firebase directly from your Flutter app.

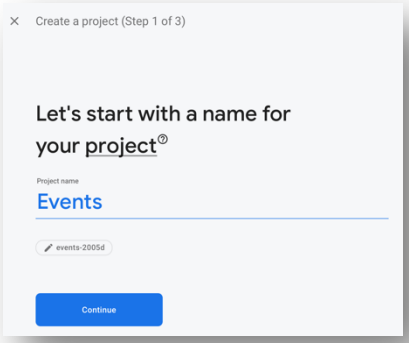


- Since Firebase is a **cloud service**, you won't need to install any software on your device. Firebase is operated by Google, so you will need a **Google account** to create your first project.
- The tools that are available within Firebase cover most of the services that you would typically have to build yourself, including **authentication**, **databases**, and **file storage**, just to name a few.

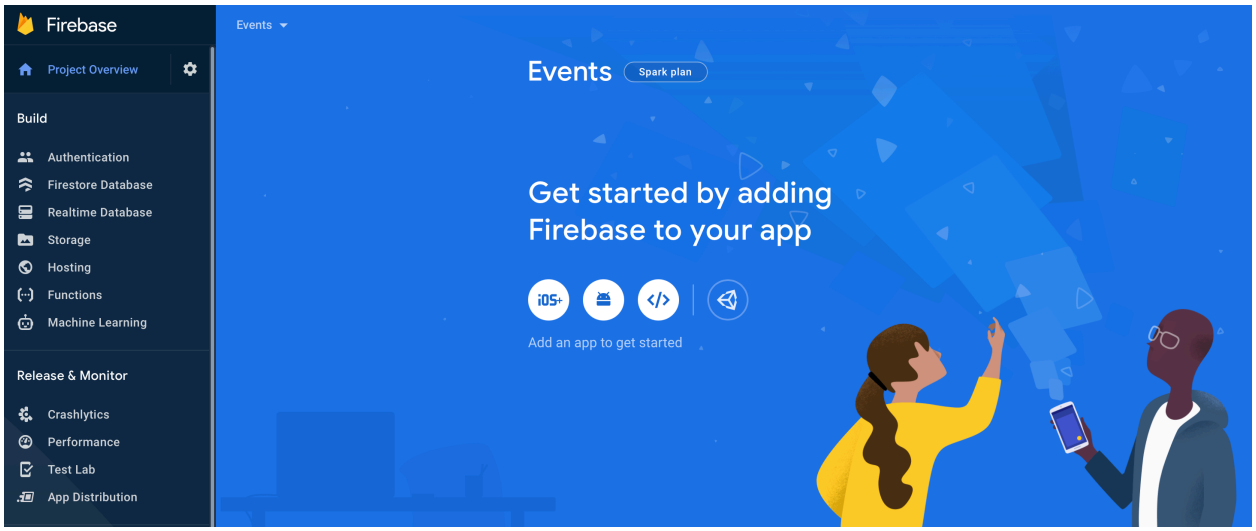
- The client that connects to Firebase—in our case, **a Flutter app**—interacts with **these backend services directly**, without any middleware server-side service. This means that, when you use the Firestore database, you'll write queries directly in your Flutter app!
- Every project that involves the use of Firebase begins with the Firebase console. You can reach it at the following address: **https://console.firebase.google.com/**.



- You will be asked to **authenticate yourself** before accessing the console. If you don’t have any Google accounts, you can create one for free from the authentication page. So, let's begin, as follows:



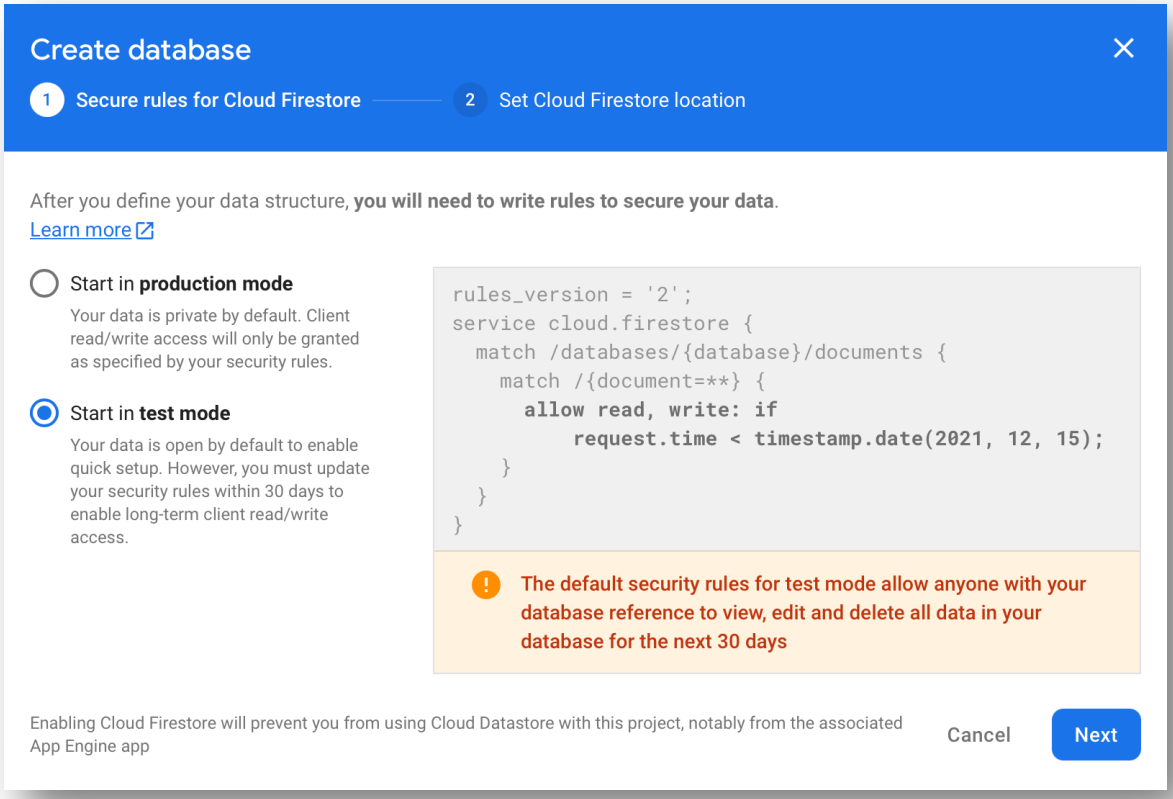
1. The container of all services in Firebase is a project. So, we'll begin building our app by **creating a new Firebase project.**
2. Once you click on the “**Create a project”** button, you'll need to choose a project name. Let's call it **Events**.
3. Next, just **press “Continue”** to keep configuring the new project and accept the Terms and Conditions.
4. In the next screen, you'll be asked to set up Google Analytics for your Firebase project. In the context of this chapter, **this is not necessary as we won't use it**, but it's generally recommended or real-world projects.
5. **Click “Continue”** again, and your project will be created soon. We now have the Firebase project that we'll use in our app. At the end of the process, you should see the following screenshot:



- This is the **Firebase Project Overview** page, which contains the name of the project (**Events**, in this case) and the billing plan (Spark plan means that it's a free plan); on the left side of the page, you have the main tools you can add to your project.
- Now that our Firebase project has been built, let's **create a Firestore database** and add some data that we'll read in our app.

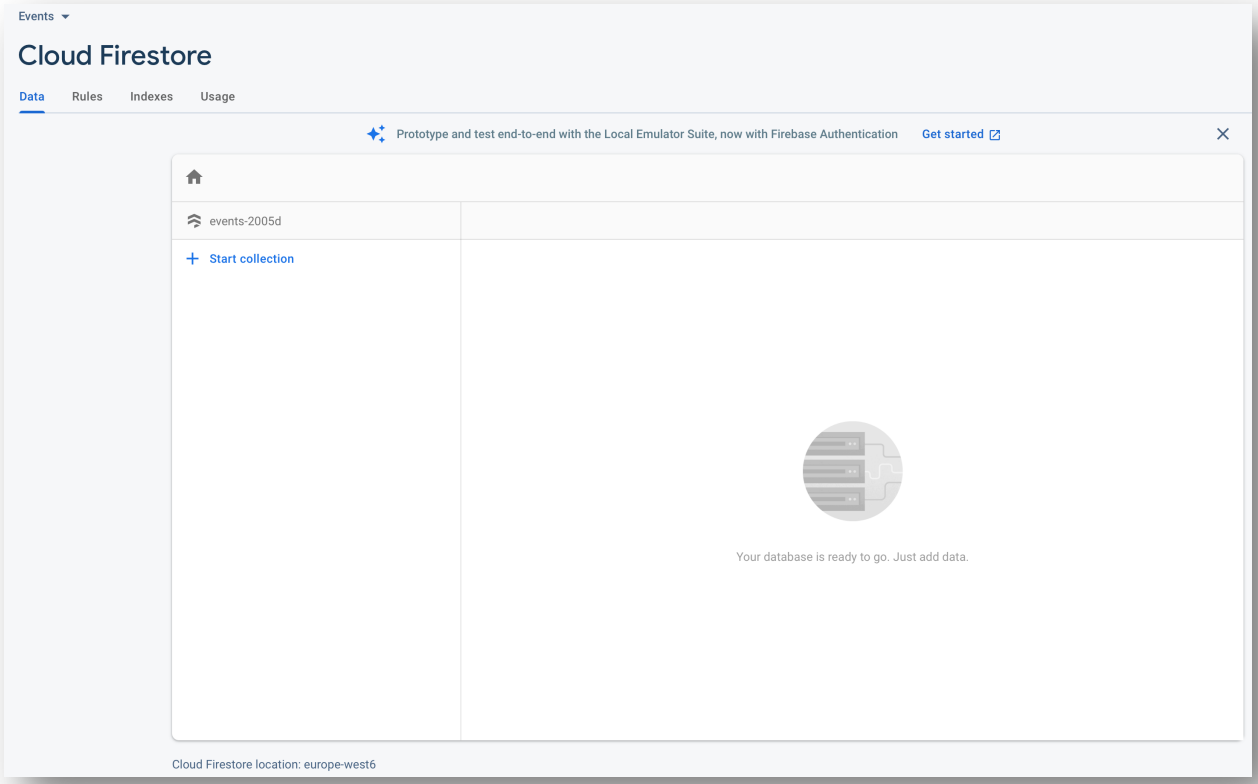
CREATING A FIRESTORE DATABASE

- In Firebase, you have two different database tools: the **Firestore Database** and the **Realtime Database**. Both are NoSQL databases, but their architecture is rather different. The **Firestore Database** is the most recent, and it's the recommended choice for most new projects as it features a more intuitive data model, with faster queries and enhanced scaling options.
- To create a **Firestore Database**, perform the following steps:
  1. On the Firebase Project Overview page, click on the **Firestore Database**.
  2. From there, click on the **Create database** button.
  3. Next, choose **Start in test mode**, as this is the option that allows access to data without authentication (we'll add authentication later on):



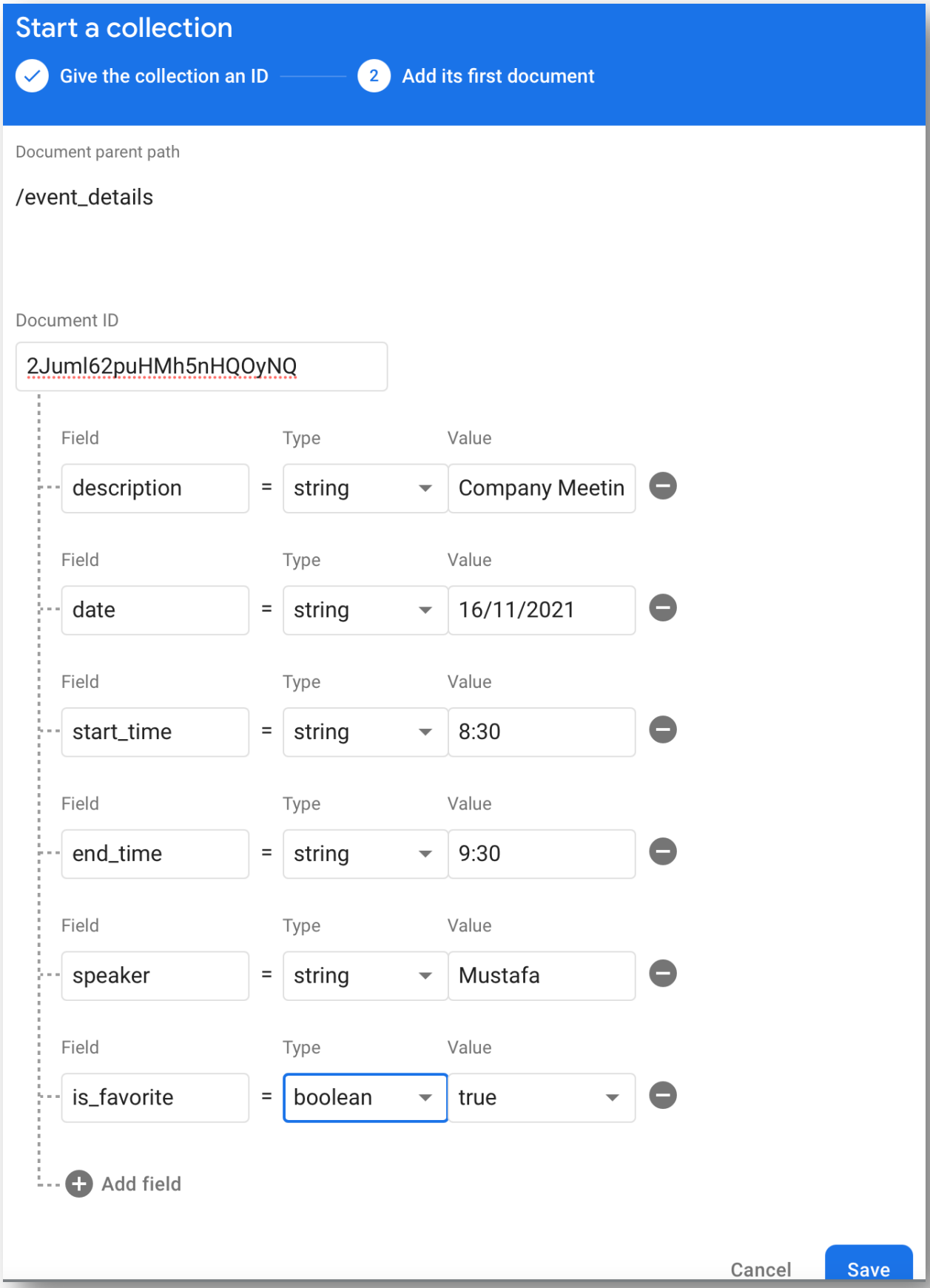
  4. Click Next. You'll be asked to choose among the **locations of the Cloud Firestore**. Choose one that is close to where you and your users will access data. I choose europe-west6 (Zurich) as it is the closest to where I live.
  5. Finally, click Done.

- You have now **created a Cloud Firestore database**, and should see a page like the one shown in the following screenshot:

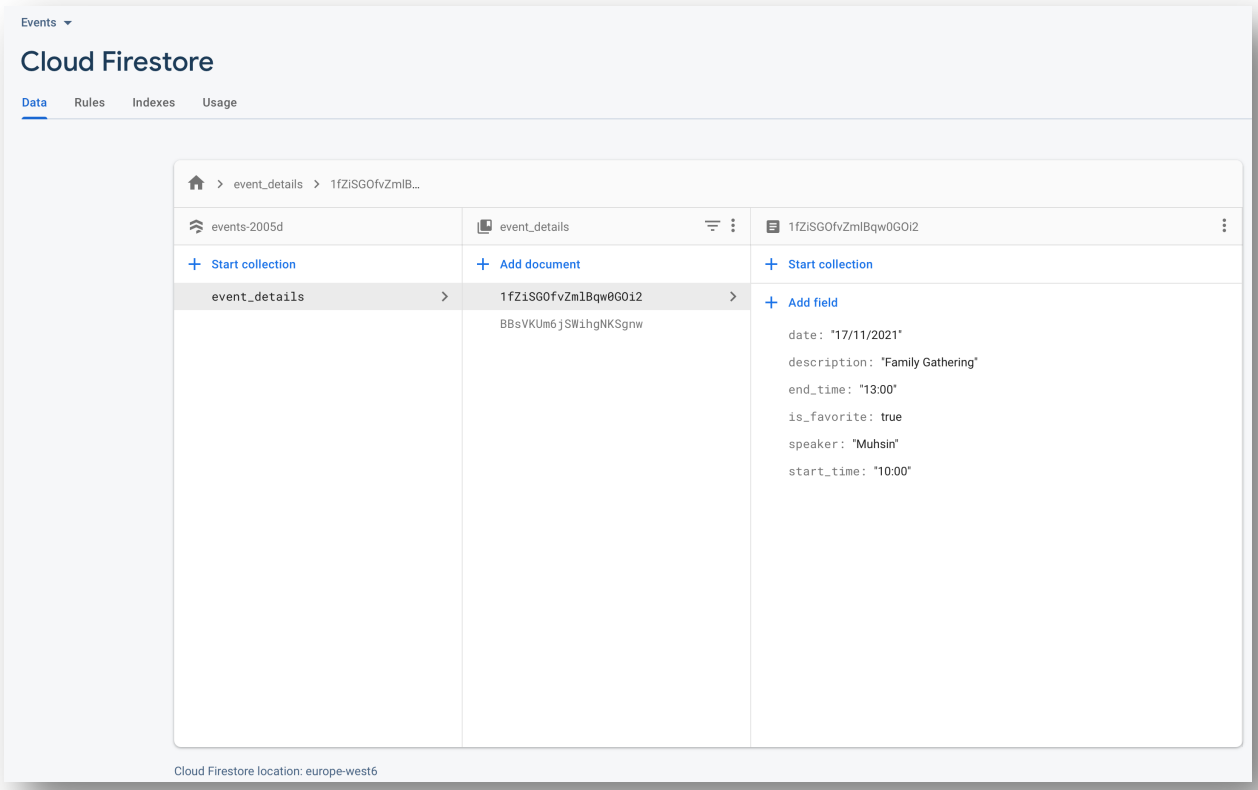


- We'll now insert some data, as follows:

1. Click on **Start collection**. A Collection is a container for a set of documents. Call this container **event_details**, and click **Next**. For example, collection "users" would contain a unique document for each user
2. From there, in the **Document ID** option, click on **Auto-ID**, then add a few fields and values, as shown in the following screenshot, then click **Save**:


Start a collection
Give the collection an ID
2
Add its first document
Document parent path
/event_details
Document ID
2Juml62puHMh5nHQOyNQ
Field
Type
Value
description
=
string
Company Meetin
Field
Type
Value
date
=
string
16/11/2021
Field
Type
Value
start_time
=
string
8:30
Field
Type
Value
end_time
=
string
9:30
Field
Type
Value
speaker
=
string
Mustafa
Field
Type
Value
is_favorite
=
boolean
true
Add field
Cancel
Save

3. **Within the same event_details collection, repeat the process to create another couple of documents**, using the same fields and changing the values in line with your preferences.



- There are a few rules when dealing with collections and documents in the Cloud Firestore database, as follows:
  - **Collections can only contain documents**, not other collections, strings, or blobs.
  - **A document can contain a sub-collection**, which can then contain other documents.
- Now that we have created our Firebase project, a Cloud Firestore database, and inserted some data, it's time to **create our Flutter app and integrate Firebase into it**.

INTEGRATING FIREBASE INTO A FLUTTER APP

- There are a few steps required to integrate a Cloud Firestore database into a Flutter app:
  1. **Create a Firebase project**. You can do this in the Firebase console. You'll need to log in to your Google account to do that.
  2. **Create a Firestore database instance**, then insert collections and documents as required.
  3. **Register your Android and/or iOS app in your project**, and download the configuration file that will be created in the process. If you are planning to release your app on both platforms, you'll need to repeat the process for both Android and iOS.
  4. **Create your Flutter project** and add the downloaded configuration files.
  5. **Add Google services to your projects** (platform-specific).
  6. **Add dependencies to the pubspec.yaml file**.
- We have already performed **Steps 1 and 2** in the task list outlined at the end of the previous section, so let's now see how to deal with the remaining steps.

CREATING A NEW FLUTTER APP

- Let's **create a new Flutter project** named **events** with main.dart as follows:

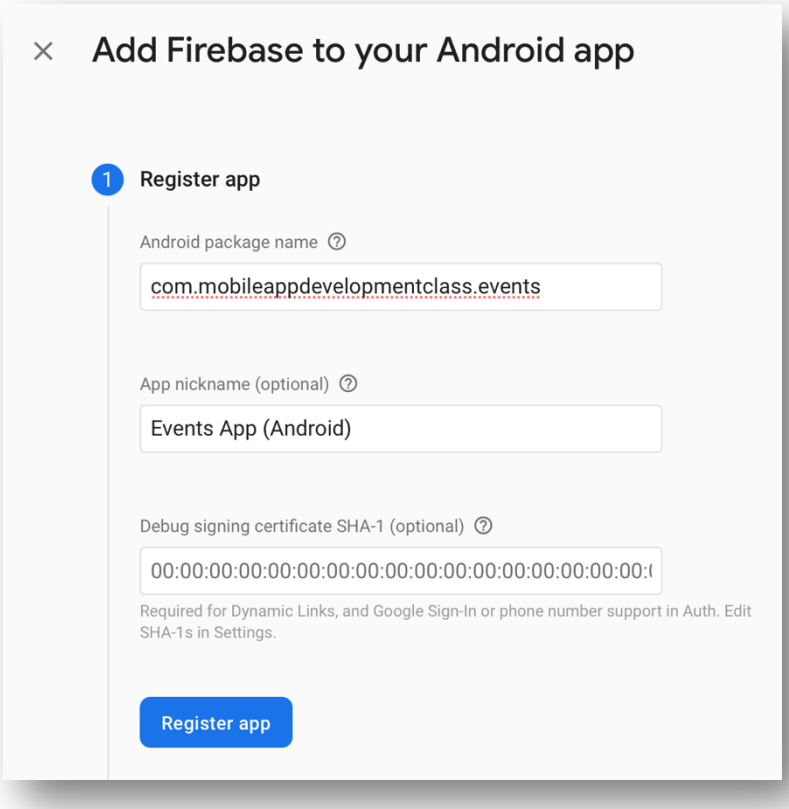
```dart
import 'package:flutter/material.dart';

void main() async {
  runApp(const MyApp());
}
class MyApp extends StatelessWidget {
  const MyApp({ Key? key }) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Events',
      theme: ThemeData(primarySwatch: Colors.orange),
      home: Scaffold(),
    );
  }
}
```

CONFIGURING YOUR ANDROID APP

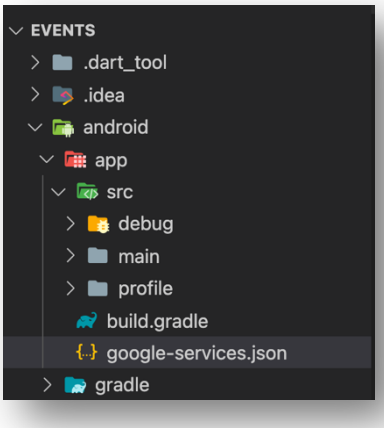
- When targeting Android devices, we'll need to **register our app as an Android app in the Firebase console**. This is needed to actually register the app with Firebase and to eventually publish the app into the Google Play Store if you wish to do so later on.
- First, open the following file: **<project-name>/android/app/build.gradle**
- In the **defaultConfig** node, you should find an **applicationId key**, with a com.example.events as the initial value. This is a unique identifier for your Android app, and you should change it to a name that uniquely identifies you. Keep in mind that Firebase does not allow bundle IDs with hyphens (for Android apps) and underscores (for iOS apps). For example, in my case, I'll change it to the following:

```
defaultConfig {
        // Modify the line below as follows
        applicationId "com.mobileappdevelopmentclass.events"
       …
    }
```

- Next, let's get to the **Project Overview page in Firebase**. Click on the **Android icon** to add an Android app.
- As the **Android package name,** enter the applicationId we have just set in the previous step.
- Optionally, you may enter a nickname for your app, which is not visible to users, but it's used throughout the Firebase console to represent your app.



- **Click on the Register app button**, and **download the google-services.json file**, then **put the file** into the **android/app folder** of your project.



- Next, **follow the steps** on the website to finish the configuration for Android.
- For our app, we'll need the following dependencies:
  - **firebase_core** for managing Firebase instances and credentials
  - **firebase_auth** for authentication
  - **cloud_firestore** for storing data
- Check the latest versions of these dependencies at https://pub.dev/.
- Next, **let's open the pubspec.yaml file** and **add the required dependencies**.

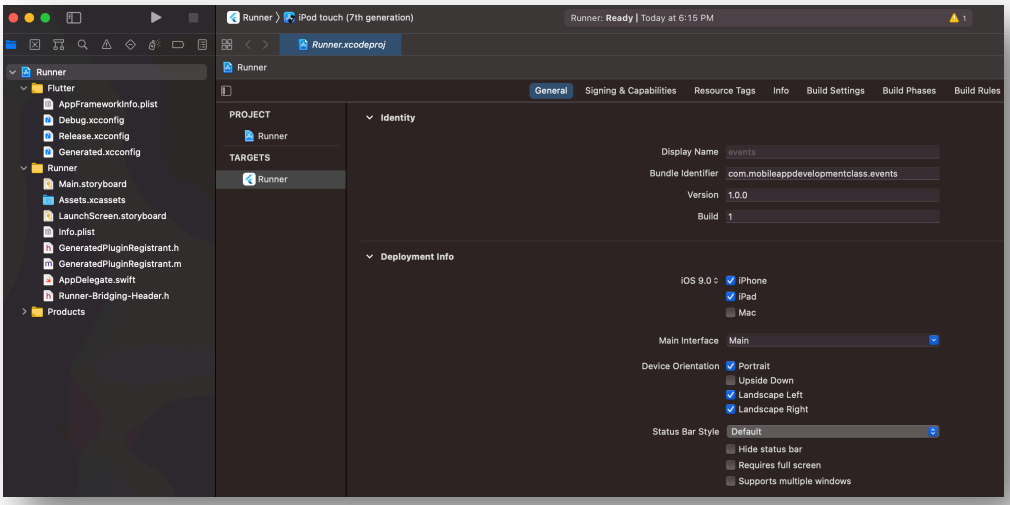
```
Firebase Dependencies
firebase_core: ^1.10.0
firebase_auth: ^3.2.0
cloud_firestore: ^3.1.0
```

- That's it. You're now ready to use Firebase for your Android app! Let's now see how to configure your app for iOS.

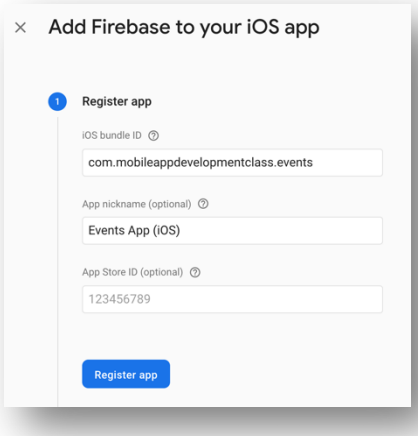
CONFIGURING YOUR IOS APP

- Let’s do the same configuration for the iOS app this time.
- We'll need to change the bundle ID of our Flutter project. This is the value that identifies your iOS app. Let's see how to do this here:

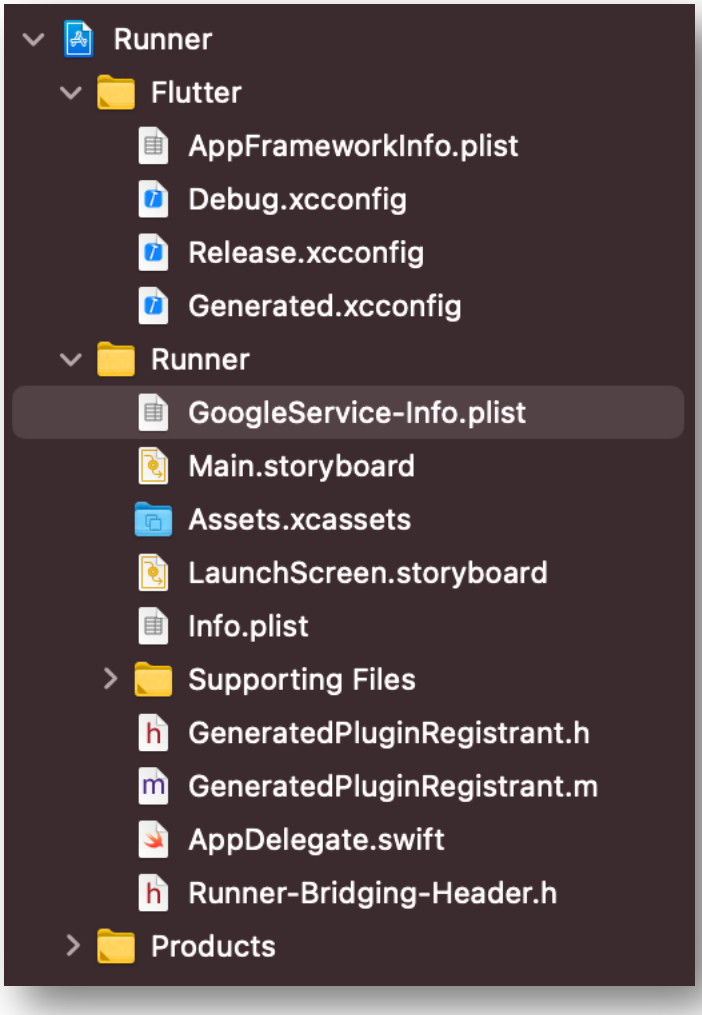
1. In VS Code, right-click on the ios folder and **open the app in Xcode**.
2. In Xcode, open the **project navigator** tab and click on the Runner project on the left.
3. Select the **General tab** to the right and change the **Bundle Identifier** value to the **same bundle ID** from the previous “Configuring Your Android App” section. This is needed to register the app with Firebase and to eventually publish the app to the App Store if you wish to do so later on.



4. Save your project and **get back to the Firebase console**.
5. Next, on the **Project Overview page**, click the **iOS icon** this time.
6. You'll be asked to insert the **iOS bundle ID**. Insert it, as shown in the following screenshot, then click on the **Register app** button.
7. **Download the GoogleService-Info.plist** to get the Firebase iOS configuration file.



8. Next, from Xcode, **move the downloaded file** into the **Runner directory** of your Flutter app, as shown in the following screenshot:



- Back in the Firebase console, **click Next**. Follow the steps to finalize the configuration for the iOS app.

TESTING FIREBASE INTEGRATION WITH YOUR APP

- Now that we have completed the app configuration to use Firebase, we need to test whether we can connect to the Firestore database. Let's begin, as follows:

1. In the main.dart file, **add the following import** to use the Cloud Firestore package:

```
import 'package:cloud_firestore/cloud_firestore.dart';
```

2. Then, before the build method in MyApp, **create an asynchronous method**, called **testData()**, that will attempt to connect to the Cloud Firestore database and print some data in the Debug console, as follows:

```
Future testData() async {}
```

3. Inside the testData method, we'll **create an instance of a Firestore database** and call it **db**, as illustrated in the following code snippet:

```
FirebaseFirestore db = FirebaseFirestore.instance;
```

4. Then, we'll call the **get ()** asynchronous method on the collection **event_details**, which will retrieve all the available data from the specified collection. We'll **put the results into a data variable** called **data**.

```
var data = await db.collection('event_details').get();
```

5. If data is not null, we'll get the documents, and place them into a new list named **details**.

```
var details = data.docs.toList();
```

6. **For each item in the details list**, we'll print the **id of the item**, which is the **unique identifier** inside a Firestore collection.

```
details.forEach((item) {
      print(item.id);
    });
```

7. **Add** the call to the testData() method in the build() method of MyApp:

```
testData();
```

8. In main.dart, **import firebase_core**.

```
import 'package:firebase_core/firebase_core.dart';
```

9. Add **async** to the main function and modify it with **the following lines**:

```
void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp();
  runApp(const MyApp());
}
```

10. Try the app. If everything worked as expected, in the Debug console you should see a line (or lines) like the following:

```
flutter: JFstv8NjRBCC4iUhyAbe
flutter: qy44otWUzLCEFUEE5QBL
```

- This means that we can **retrieve data** from the **Cloud Firestore**, and import it into our app!
- **In Android,** you might receive **a couple of errors**. If you receive an error;
    - relating to the **minSDKVersion**, check the console window first for the minSdkVersion and update it in the android/app/build.gradle file.
    - relating to the **number of references** in your app, add the following code in your app.gradle file, at the end of the defaultConfig node:

```
defaultConfig {
    // ...
    multiDexEnabled true
}
```

- Congratulations! We've now successfully connected our app to Firebase.
- Let's design the user interface (UI) so that the user will be able to see a list of event details on the screen.

REFERENCES

- Flutter Projects by Simone Alessandria, 2020, Packt Publishing