

# FLUTTER FIREBASE

## Flutter Course 2022

Revised Flutter 3.0

**Caspian** Consultancy Services

# Modules

## Firestore 2021

- Introduction
- Setup
- Create
- Read
- Update
- Delete
- Create a small Firestore project
- Create A Real Firestore project

# Introduction

Firestore 2021

CRUD operations are the main functions you want to know as a developer in every language or framework. You will create basic individual CRUD functions. This will help you create complex Firestore apps later.

# Firestore Setup

# Firestore Setup

Firestore 2021

## Prerequisites:

1. Firestore Account, Console (Link with Google Account).
2. A Basic Flutter Project, A Simple, **hellofirebase**.
3. iOS or Android App setup for the Flutter project.

# Firestore Setup

Firestore 2021

## Step 1:

1. You will need to register a Google Firestore Account. If you have a google account, just register Firestore account.
2. Goto <https://firebase.google.com>

# Firebase Setup

## Firebase 2021

### Create a demo project:

In the Firebase console click add project and give your project name. Example “**productapp**”

After that you can register your Android App or iOS App.

### Add a project

Project name  
productapp

Project ID  
productapp-1af84

Analytics location  
United States

☒ Use the default settings for sharing Google Analytics for Firebase data

- ✓ Share your Analytics data with all Firebase features
- ✓ Share your Analytics data with Google to improve Google Products and Services
- ✓ Share your Analytics data with Google to enable technical support
- ✓ Share your Analytics data with Google to enable Benchmarking
- ✓ Share your Analytics data with Google Account Specialists

☒ I accept the [controller-controller terms](#). This is required when sharing Analytics data to improve Google Products and Services. [Learn more](#)

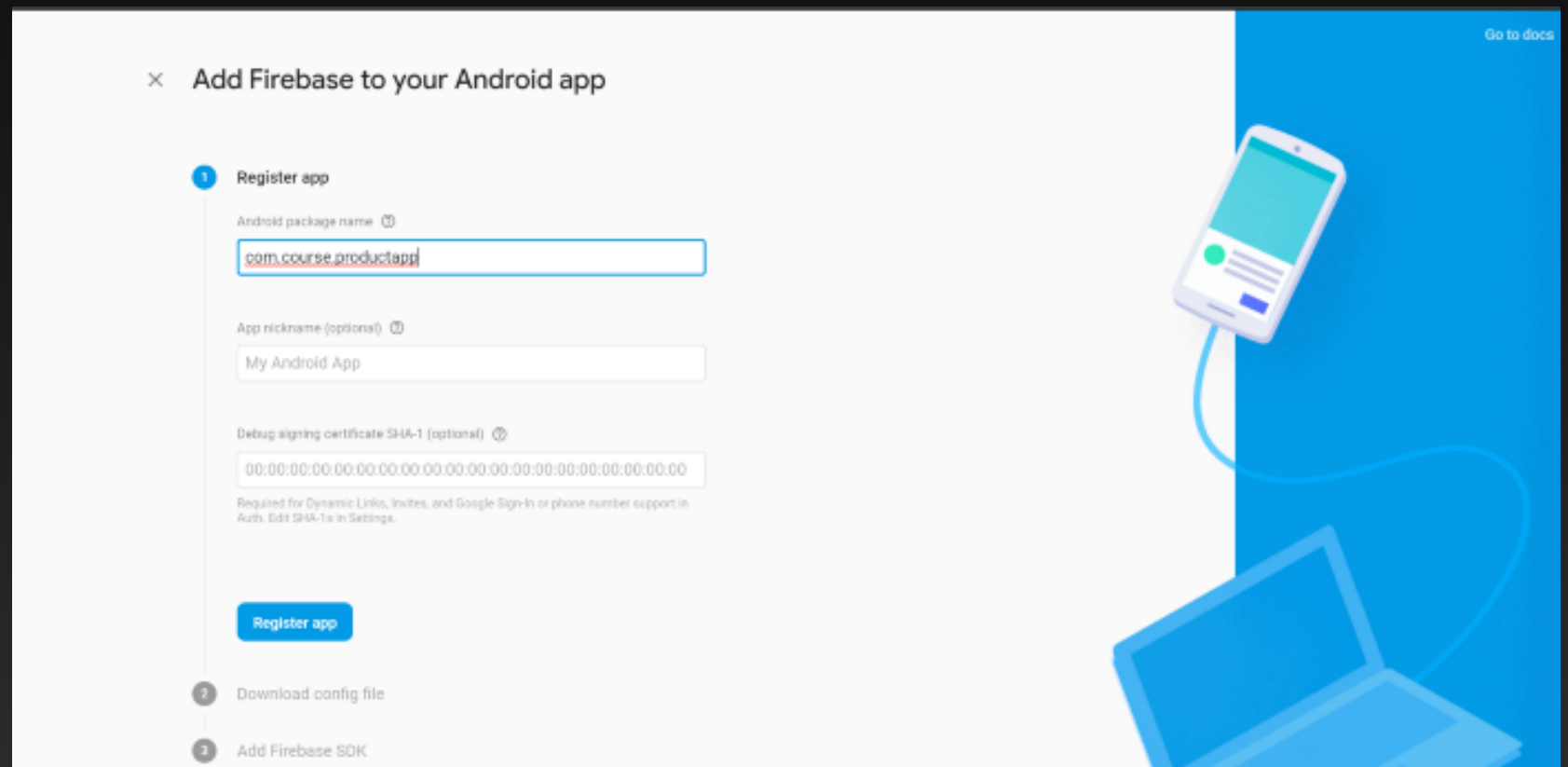
Cancel Create project

# Firebase Setup

## Firebase 2021

### Android setup:

Once the project is created, go to the Firebase Console and register your app by clicking Add Firebase to your app Android. Enter your AppID.



The screenshot shows the 'Add Firebase to your Android app' wizard. It has a title bar with a close button and the text 'Add Firebase to your Android app'. On the right side, there is a blue vertical bar with a 'Go to docs' link and an illustration of a smartphone connected to a laptop. The main content area has a progress indicator on the left with three steps: 1. Register app (active), 2. Download config file, and 3. Add Firebase SDK. The 'Register app' step contains three input fields: 'Android package name' with the value 'com.course.productapp', 'App nickname (optional)' with the value 'My Android App', and 'Debug signing certificate SHA-1 (optional)' with a placeholder value '00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00'. Below these fields is a blue 'Register app' button. A small note at the bottom of the third field states: 'Required for Dynamic Links, invites, and Google Sign-In or phone number support in Auth. Edit SHA-1s in Settings.'

× Add Firebase to your Android app

1 Register app

Android package name ⓘ

com.course.productapp

App nickname (optional) ⓘ

My Android App

Debug signing certificate SHA-1 (optional) ⓘ

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

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

Register app

2 Download config file

3 Add Firebase SDK

Go to docs

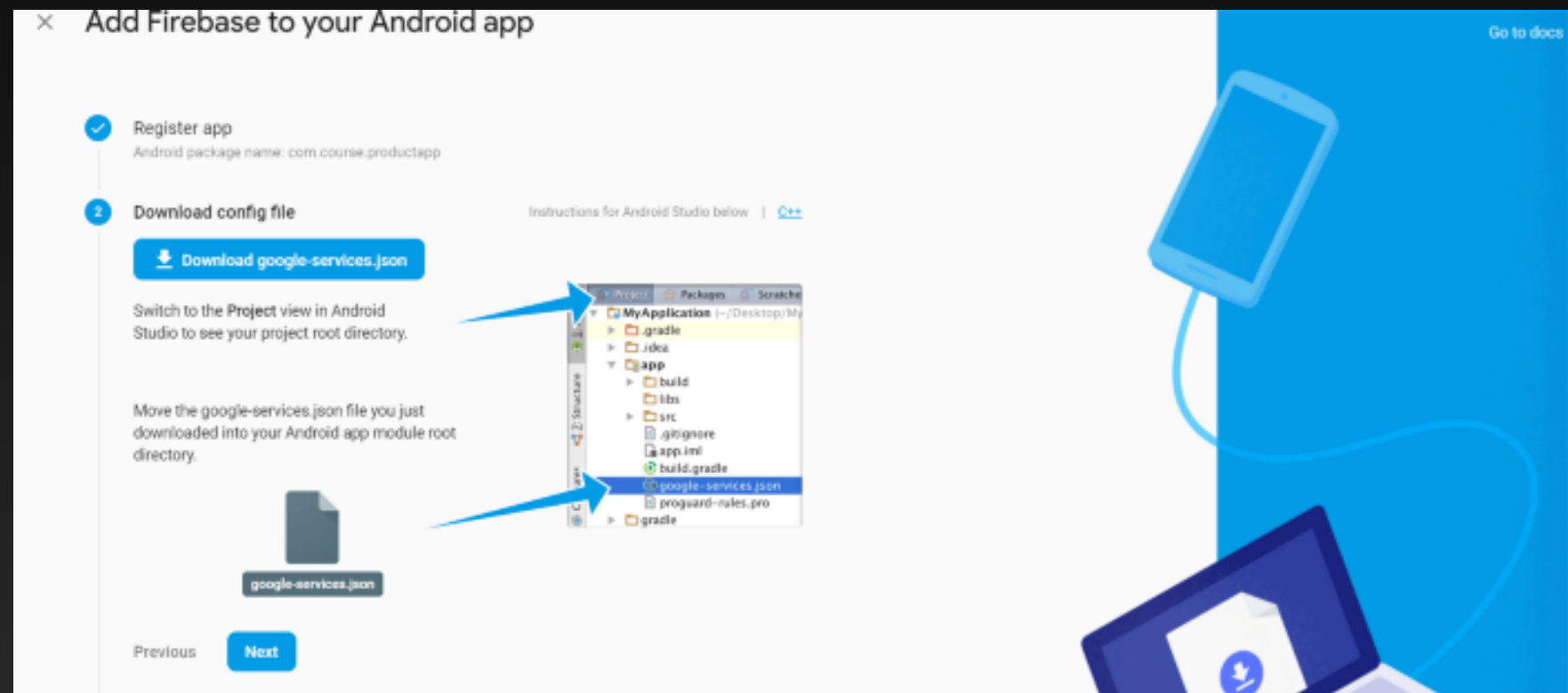


# Firebase Setup

## Firebase 2021

### Android setup:

Download the *google-services.json* file to the **android/app** directory



# Firebase Setup

## Firebase 2021

### Android setup:

Now we need to register our Google services in the Gradle build files under android/build.gradle

#### Android project/build.gradle

```
buildscript {  
    dependencies {  
        // ...  
        classpath 'com.google.gms:google-services:4.2.0' // <--  
    }  
}
```

#### App/build.gradle

```
defaultConfig {  
    applicationId "com.course.product" // <-- update this line  
    minSdkVersion 21 // <-- you might also need to change this  
    to 21  
    // ...  
}  
  
// ... bottom of file  
apply plugin: 'com.google.gms.google-services' // <-- add
```

# Firestore Setup

Firestore 2021

## iOS setup:

You need a MacBook or iMac. If you don't, just skip this.

The iOS setup is easier and can be completed in one step.

In your firestore project, Click add your iOS app then download the **GoogleService-Info.plist** file into the **iOS/Runner/** from Xcode (**Make sure to use Xcode**, Xcode will generate a runner-bridging-header).

# Flutter Setup

## Firebase 2021

### Plugins:

#### 1. **firebase\_core**

Enabling basic functions and connecting to Firebase

#### 2. **cloud\_firestore**

Using Firebase/fireStore API

#### 3. **firebase\_auth**

For Authentication purposes

#### 4. **google\_signin**

For Additional Google Signing Authentications

#### 5. **provider**

Flutter State management, If we need to pass down values to other inherited widgets

# Firebase Setup

## Firebase 2021

### In your Flutter Project:

#### 1. Add Dependencies

```
dependencies:  
  flutter:  
    sdk: flutter  
  
  cloud_firestore:  
  firebase_core :
```

#### 2. Import & Initialise your firebase in main( )

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

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

# Firebase Setup

## Firebase 2021

### If Error in Android compilation, Project>App Gradle

1. Add minSdk 19 (or Whatever Google requirement)
2. Add multiDex support

```
android {  
    defaultConfig {  
        ...  
        minSdk = 15  
        targetSdk = 28  
        multiDexEnabled = true  
    }  
    ...  
}  
  
dependencies {  
    implementation("androidx.multidex:multidex:2.0.1")  
}
```

Note: If your minSdkVersion is set to 21 or higher, multidex is enabled by default and you do not need the multidex library.

However, if your minSdkVersion is set to 20 or lower, then you must use the multidex library and make the following modifications to your app project:

# Create Data

## Firebase 2021

### Use:

```
void _create() async {  
  try {  
    await firestore.collection('users').doc('testUser').set({  
      'firstName': 'John',  
      'lastName': 'Peter',  
    });  
  } catch (e) {  
    print(e);  
  }  
}
```

# Update Data

## Firestore 2021

### Use:

```
void _update() async {  
  try {  
    firestore.collection('users').doc('testUser').update({  
      'firstName': 'Alan',  
    });  
  } catch (e) {  
    print(e);  
  }  
}
```



# Read Data

## Firebase 2021

### Use:

```
void _read() async {  
  DocumentSnapshot documentSnapshot;  
  try {  
    documentSnapshot = await  
    firestore.collection('users').doc('testUser').get();  
    print(documentSnapshot.data());  
  } catch (e) {  
    print(e);  
  }  
}
```

# Delete Data

## Firebase 2021

### Use:

```
void _delete() async {  
  try {  
    firestore.collection('users').doc('testUser').delete();  
  } catch (e) {  
    print(e);  
  }  
}
```

# A Simple Firebase App

## Firebase 2021

### Step 1: Initialize Firebase

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

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

class MyApp extends StatelessWidget {
  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: MyHomePage(),
    );
  }
}
```

# A Simple Firebase App

## Firebase 2021

### Step 2: Create a Firebase Instance & Associated Functions (Create, Read, Update, Delete)

```
class MyHomePage extends StatefulWidget {  
  @override  
  _MyHomePageState createState() => _MyHomePageState();  
}  
  
class _MyHomePageState extends State<MyHomePage> {  
  final FirebaseFirestore firestore = FirebaseFirestore.instance;  
  
  void _create() async {  
    try {  
      await firestore.collection('users').doc('testUser').set({  
        'firstName': 'John',  
        'lastName': 'Peter',  
      });  
    } catch (e) {  
      print(e);  
    }  
  }  
}
```

# A Simple Firebase App

## Firebase 2021

### Step 2: Create a Firebase Instance & Associated Functions (Create, Read, Update, Delete)

```
void _read() async {
  DocumentSnapshot documentSnapshot;
  try {
    documentSnapshot = await
firestore.collection('users').doc('testUser').get();
    print(documentSnapshot.data);
  } catch (e) {
    print(e);
  }
}

void _update() async {
  try {
    firestore.collection('users').doc('testUser').update({
      'firstName': 'Alan',
    });
  } catch (e) {
    print(e);
  }
}
```

# A Simple Firebase App

## Firebase 2021

### Step 2: Create a Firebase Instance & Associated Functions (Create, Read, Update, Delete)

```
void _delete() async {  
  try {  
    firestore.collection('users').doc('testUser').delete();  
  } catch (e) {  
    print(e);  
  }  
}
```

# A Simple Firebase App

## Firebase 2021

**Step 4: Create  
Buttons to run  
those Functions  
(Create, Read,  
Update, Delete)**

```
@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: Text("Flutter CRUD with Firebase"),
    ),
    body: Center(
      child: Column(mainAxisAlignment: MainAxisAlignment.center,
children: <Widget>[
  RaisedButton(
    child: Text("Create"),
    onPressed: _create,
  ),
  RaisedButton(
    child: Text("Read"),
    onPressed: _read,
  ),
  RaisedButton(
    child: Text("Update"),
    onPressed: _update,
  ),
  RaisedButton(
    child: Text("Delete"),
    onPressed: _delete,
  ),
],
      ),
    ),
  );
}
```

**Let's Do a Real Project**

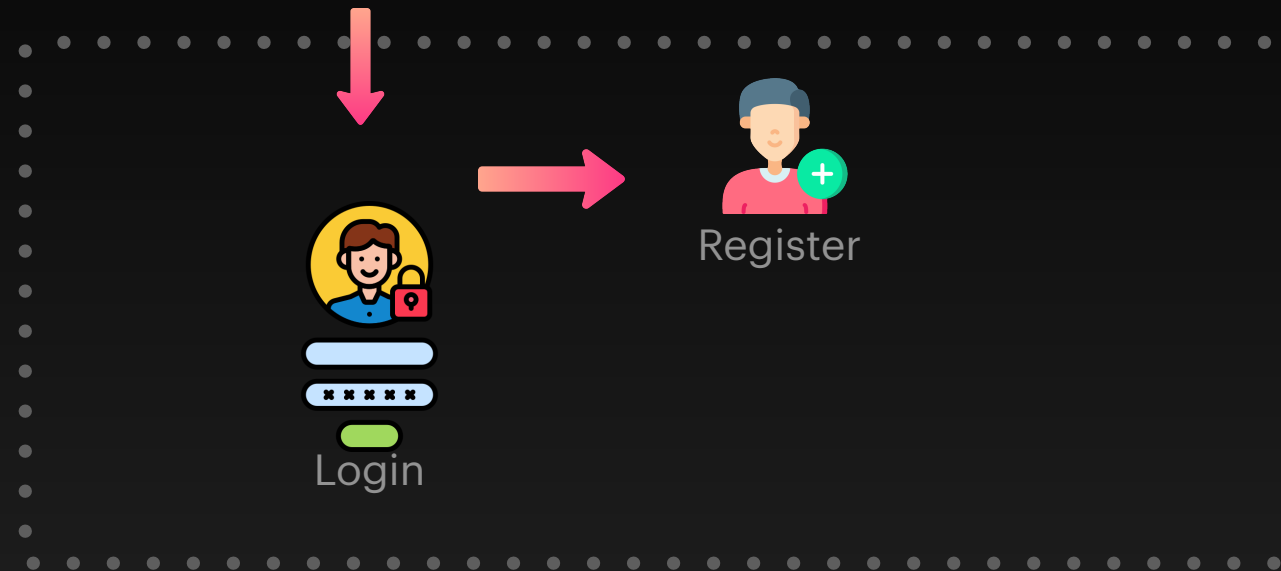




myapp

Check using Authenticate

We will learn &  
use Firebase  
Auth



Home



User Create  
Products



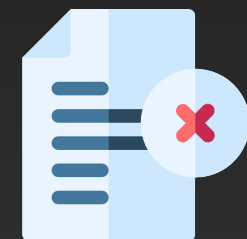
Show Detail



Create Query  
User products List



Edit product



Delete product



Show Detail

Refer to Project in my Github:  
<https://github.com/casfian/hellofirebase2>

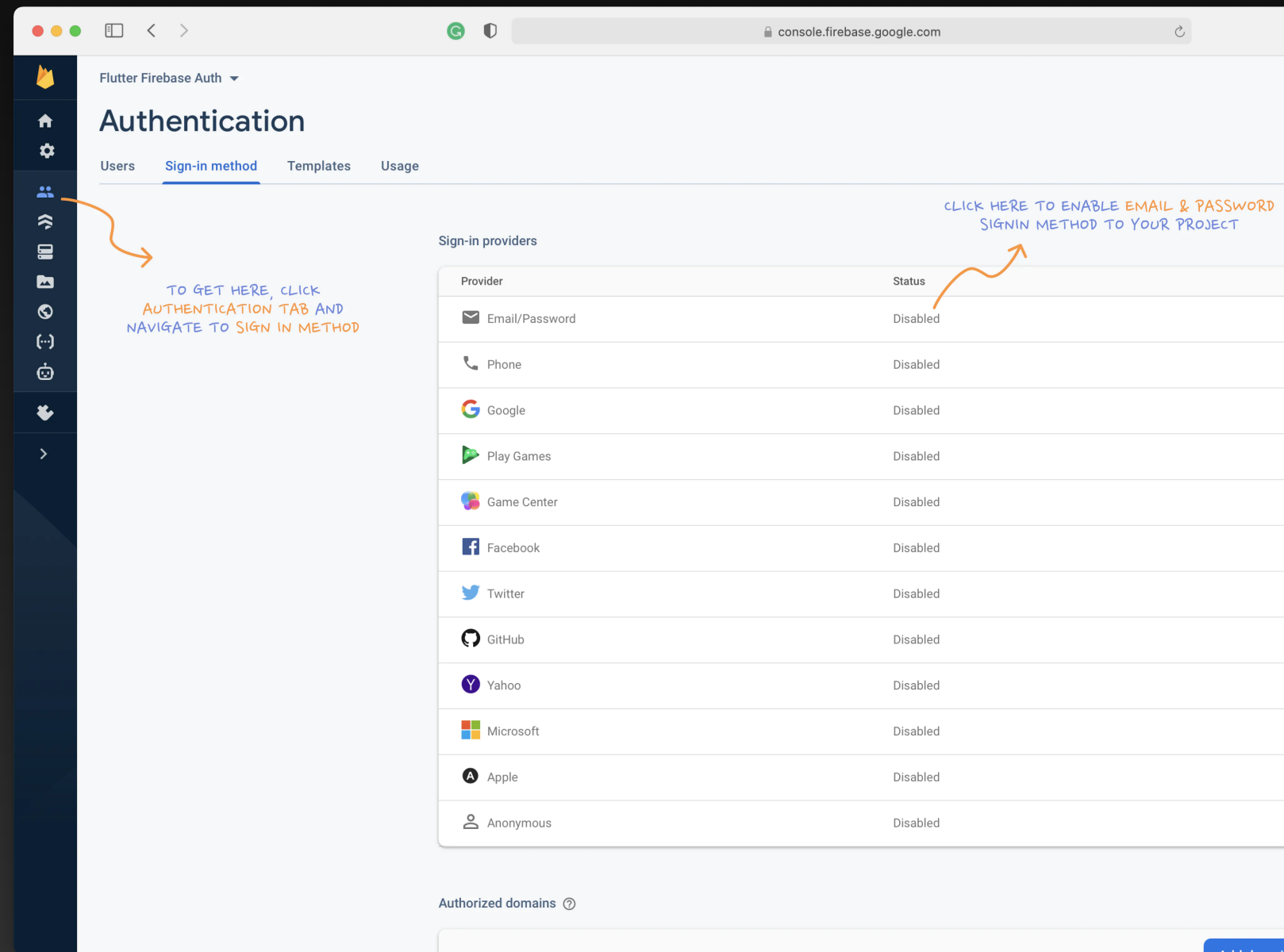
# Firestore Auth

# Firebase Auth

## Firebase 2021

### Step 1:

Open the Authentication tab from project home and navigate to the sign-in method. You can find many sign-in options, but we are only using email and password in our app.



# Firebase Auth

## Creating Authentication Functions

**Step 1:** Create an authentication.dart file (Service file) in lib folder.

Why create a service file? Easier to call functions if you doing a big project and it's reusable in other projects.

This Authentication Service is a class that has different methods that handle Firebase Authentication.

Create **AuthenticationService** class first, then add these functions

- 1. \_firebaseAuth** is a variable of type FirebaseAuth
- 2. The streamProvider** returns a User class if the user is signed in or null if they are not.
- 3. signUp()** accepts email and password for firebase sign up and returns a Future String
- 4. signIn()** accepts email and password for firebase sign in and returns a Future String
- 5. signOut()** is a simple method. On calling, it sign out the user and returns a Future String.
- 6. getUser()** returns the current logged-in user details.

In all the methods, if an exception occurs is handled under FirebaseAuthException.

# Firebase Auth

## Creating StreamProvider Getter

Create a StreamProvider Getter so that we can use in our App. This is useful For Provider so that we can pass the Getter data in our widgets.

Create Auth variable Instance and constructor

```
final FirebaseAuth firebaseAuth;  
//FirebaseAuth instance  
AuthenticationProvider(this.firebaseAuth);  
//Constuctor to inititalize the FirebaseAuth instance
```

# Firestore Auth

## Creating GetUser Function

Create getter where it returns the current user when there's a change in the Auth IDToken.  
Using Stream because the data should always be flowing in or always in the state of change

```
//Using Stream to listen to Authentication State  
Stream<User?> get authState => firebaseAuth.idTokenChanges();
```

Note: refer Stream as a stream of events flowing.

# Firebase Auth

## Creating SignUp Function

Create an email based **SignUp** function. Function return a String but you can also use a Boolean

```
//SIGN UP METHOD
Future<String?> signUp({required String email, required String password}) async {
  try {
    await firebaseAuth.createUserWithEmailAndPassword(
      email: email, password: password);
    return "Signed up!";
  } on FirebaseAuthException catch (e) {
    return e.message;
  }
}
```

# Firebase Auth

## Creating SignIn Function

Create an email based **SignIn** function. Function return a String but you can also use a Boolean.

```
//SIGN IN METHOD
Future<String?> signIn({required String email, required String password}) async {
  try {
    await firebaseAuth.signInWithEmailAndPassword(
      email: email, password: password);
    return "Signed in!";
  } on FirebaseAuthException catch (e) {
    return e.message;
  }
}
```



# Firebase Auth

## Creating SignOut Function

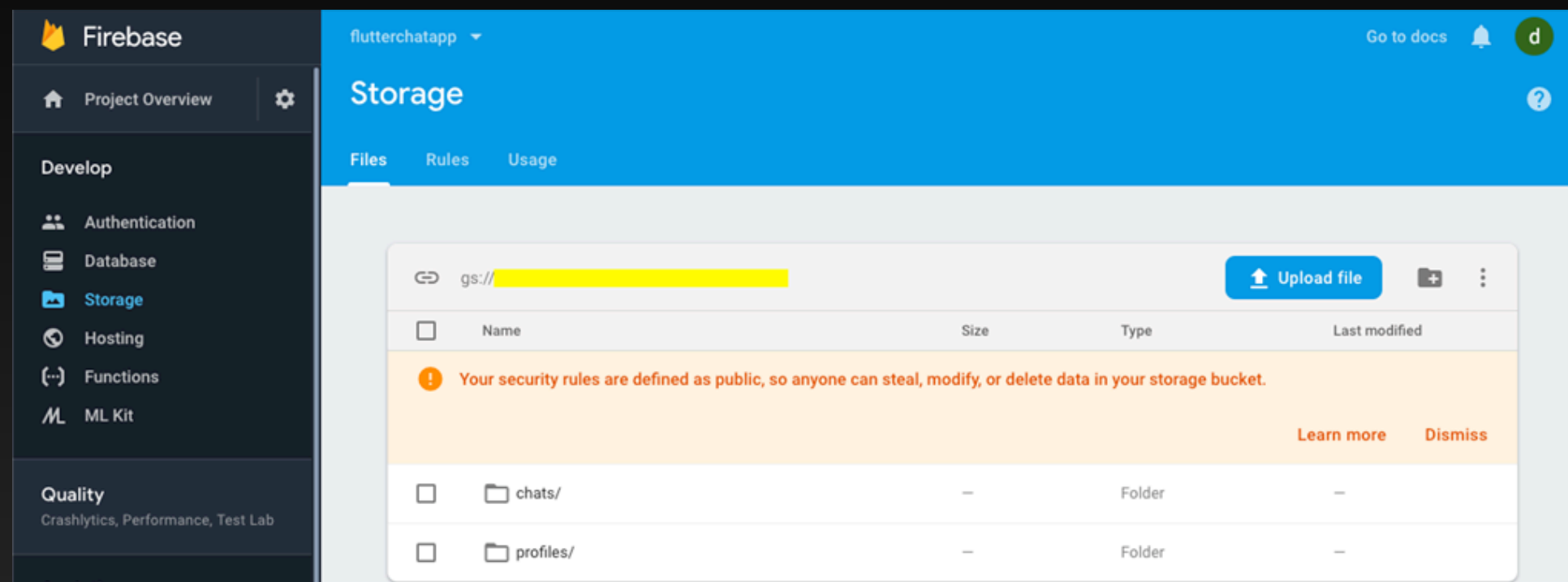
Create a SignOut function.

```
//SIGN OUT METHOD
Future<void> signOut() async {
  await firebaseAuth.signOut();
}
```

```
}
```

# Firestore Storage

# Firebase Storage Setup



You'll need **firebase\_storage** package dependency. Goto pub.dev and install it in your Flutter Project.

Make sure security rule to allow read and write access.

More info: <https://firebase.flutter.dev/docs/storage/overview/> and <https://firebase.flutter.dev/docs/storage/usage>

# Firestore Storage

## Storage Rule

You can change the rule later with access authorization



Guard your data with rules that define who has access to it and how it is structured

[View the docs](#)

```
1 rules_version = '2';
2 service firebase.storage {
3   match /b/{bucket}/o {
4     match /{allPaths=**} {
5       allow read, write;
6     }
7   }
8 }
```

# Firebase Storage

## Upload Files in Firebase

### File uploads

To upload a file, you must first create an absolute path to its on-device location. For example, if a file exists within the application's documents directory, we can use the official `path_provider` package to generate a file path:

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

Future<void> uploadExample() async {
  Directory appDocDir = await getApplicationDocumentsDirectory();
  String filePath = '${appDocDir.absolute}/file-to-upload.png';
  // ...
  // e.g. await uploadFile(filePath);
}
```

Once your absolute path has been created, it can be passed as a `File` instance to the `putFile` method:

```
Future<void> uploadFile(String filePath) async {
  File file = File(filePath);

  try {
    await firebase_storage.FirebaseStorage.instance
      .ref('uploads/file-to-upload.png')
      .putFile(file);
  } on firebase_core.FirebaseException catch (e) {
    // e.g, e.code == 'canceled'
  }
}
```

# Firestore Storage

## Download Files in Firestore

### Downloading Files

To download a file to the local device, you can call the `writeToFile` method on any storage bucket reference. The location of where the file will be downloaded to is determined by the absolute path of the `File` instance provided, for example:

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

Future<void> downloadFileExample() async {
  Directory appDocDir = await getApplicationDocumentsDirectory();
  File downloadToFile = File('${appDocDir.path}/download-logo.png');

  try {
    await firestore_storage.FirebaseStorage.instance
      .ref('uploads/logo.png')
      .writeToFile(downloadToFile);
  } on firestore_core.FirebaseException catch (e) {
    // e.g, e.code == 'canceled'
  }
}
```

If a file already exists at the provided path, it will be overwritten.

# Firestore Storage

## Listing Files in Firestore

### Listing files & directories

Firestore provides the ability to list the files and directories within a directory. There are two methods available which provide this ability; `list` & `listAll`. Both methods return a `ListResult` which contains any files, directories and pagination tokens from the request.

For example, to view all files and directories within the root of the default storage bucket:

```
Future<void> listExample() async {
  firestore_storage.ListResult result =
    await firestore_storage.FirebaseStorage.instance.ref().listAll();

  result.items.forEach((firestore_storage.Reference ref) {
    print('Found file: $ref');
  });

  result.prefixes.forEach((firestore_storage.Reference ref) {
    print('Found directory: $ref');
  });
}
```

The `items` property represents files within the bucket, and the `prefixes` property represents nested directories.

In cases where you have a large volume of files and directories, calling `listAll` may take a long time to return all results. In this case, calling `list` and limiting the results may result in a better user experience:

```
Future<void> listExample() async {
  firestore_storage.ListResult result = await firestore_storage
    .FirebaseStorage.instance
    .ref()
    .list(firestore_storage.ListOptions(maxResults: 10));
  // ...
}
```