UNIVERSITÉ DES MASCAREIGNES

SAVOIR, C'EST POUVOIR

Université de Limoges

LAB SESSION 4

CRUD OPERATIONS WITH SQLite using flutter

From:Kritika Bissessur
To:Mr Shiam Beeharry

UNIVERSITÉ DES MASCAREIGNES
SAVOIR, C'EST POUVOIR

Université de Limoges

Contents

1 Task 1

Task-1

Implement CRUD operations based on the base code and tutorial given
Using the tutorial link#1 implement the CRUD operations on a table of your choice in SQLite.
https://docs.flutter.dev/cookbook/persistence/sqlite
Provide a documentation with a short description and screenshots for all the steps of the lab and the Dart files.

1.1 Brief overview of task 1:

In summary, I implemented CRUD operations for a SQLite database in a Flutter application, guided by the tutorial link#1. The tutorial outlined the process of integrating SQLite for data persistence in Flutter, leading to the creation of a DatabaseHelper class. This class, extending ChangeNotifier, facilitates real-time UI updates upon changes in the database. The SQLite database, initialized in the class, was utilized to manage a table storing task data, represented by the Todo class. The implemented CRUD operations (Create, Read, Update, Delete) allowed seamless manipulation of task records within the application. Overall, the tutorial provided a structured and efficient approach to incorporating SQLite databases in Flutter, enhancing data management capabilities.

1.2 Steps

Import the dependencies.

Importing the sqflite and path packages is necessary in order to work with SQLite databases.

For interacting with a SQLite database, the sqflite package offers classes and functions.

The functions in the path package let you specify where the database will be stored on disk.

```
dependencies:
  flutter:
    sdk: flutter


  # The following adds the Cupertino Icons font to your application.
  # Use with the CupertinoIcons class for iOS style icons.
  cupertino_icons: ^1.0.2
  sqflite:
  path:
```

Additionally, we must import the packages into the file we'll be working in. As I use the main.Dart I'll be importing the packages into the main.dart.

```
import 'dart:async';

import 'package:flutter/widgets.dart';
import 'package:path/path.dart';
import 'package:sqflite/sqflite.dart';
```

We need to define the Dog data model.

We must first specify the data that must be stored before we can create the table to contain information about dogs. For the purposes of this illustration, we will define a Dog class that has three fields: a distinctive id, the name, and the age of each dog.

```
class Dog {
  final int id;
  final String name;
  final int age;

  const Dog({
    required this.id,
    required this.name,
    required this.age,
  });

  // Convert a Dog into a Map. The keys must correspond to the names of the
  // columns in the database.
  Map<String, dynamic> toMap() {
    return {
      'id': id,
      'name': name,
      'age': age,
    };
  }

  // Implement toString to make it easier to see information about
  // each dog when using the print statement.
  @override
  String toString() {
    return 'Dog{id: $id, name: $name, age: $age}';
  }
}
```

Open the database.

We must first establish a connection to the database before reading and writing data to it.

This involves two steps:

- Define the path to the database file using getDatabasesPath() from the sqflite package, combined with the join function from the path package.
- Open the database with the openDatabase() function from sqflite

```
WidgetsFlutterBinding.ensureInitialized();
// Open the database and store the reference.
final database = openDatabase(
  // Set the path to the database. Note: Using the `join` function from the
  // `path` package is best practice to ensure the path is correctly
  // constructed for each platform.
  join(await getDatabasesPath(), 'doggie_database.db'),
  // When the database is first created, create a table to store dogs.
  onCreate: (db, version) {
    // Run the CREATE TABLE statement on the database.
    return db.execute(
      'CREATE TABLE dogs(id INTEGER PRIMARY KEY, name TEXT, age INTEGER)',
    );
  },
  // Set the version. This executes the onCreate function and provides a
  // path to perform database upgrades and downgrades.
  version: 1,
);
```

We need to create the dogs table.

In the next steps , we must create a table to contain information about numerous Dogs. For this example, we will construct a table called dogs that outlines the data that can be saved. Each Dog has an identification number, a name, and an age.

As a result, the dogs table has three columns for these. The id is a Dart integer that is saved as an INTEGER SQLite Datatype.

The name is a Dart String and is stored as a TEXT SQLite Datatype. The age is also a Dart int, and is stored as an INTEGER Datatype

```
WidgetsFlutterBinding.ensureInitialized();
// Open the database and store the reference.
final database = openDatabase(
  // Set the path to the database. Note: Using the `join` function from the
  // `path` package is best practice to ensure the path is correctly
  // constructed for each platform.
  join(await getDatabasesPath(), 'doggie_database.db'),
  // When the database is first created, create a table to store dogs.
  onCreate: (db, version) {
    // Run the CREATE TABLE statement on the database.
    return db.execute(
      'CREATE TABLE dogs(id INTEGER PRIMARY KEY, name TEXT, age INTEGER)',
    );
  },
  // Set the version. This executes the onCreate function and provides a
  // path to perform database upgrades and downgrades.
  version: 1,
);
```

Insert a Dog into the database.

We need to read and write data now that we have a database with a table suited for storing information about numerous pets.

To begin, add a Dog to the dogs table.

There are two steps to this:

- Convert the Dog into a Map
- Insert the Map into the dogs table using the insert() method

```
class Dog {
  final int id;
  final String name;
  final int age;

  const Dog({
    required this.id,
    required this.name,
    required this.age,
  });

  // Convert a Dog into a Map. The keys must correspond to the names of the
  // columns in the database.
  Map<String, dynamic> toMap() {
    return {
      'id': id,
      'name': name,
      'age': age,
    };
  }

  // Implement toString to make it easier to see information about
  // each dog when using the print statement.
  @override
  String toString() {
    return 'Dog{id: $id, name: $name, age: $age}';
  }
}
```

```
// Define a function that inserts dogs into the database
Future<void> insertDog(Dog dog) async {
  // Get a reference to the database.
  final db = await database;

  // Insert the Dog into the correct table. You might also specify the
  // `conflictAlgorithm` to use in case the same dog is inserted twice.
  //
  // In this case, replace any previous data.
  await db.insert(
    'dogs',
    dog.toMap(),
    conflictAlgorithm: ConflictAlgorithm.replace,
  );
}
```

```
// Create a Dog and add it to the dogs table
var fido = const Dog(
  id: 0,
  name: 'Fido',
  age: 35,
);
```

Retrieve the list of dogs.

Now that a Dog has been saved in the database, we must query it for a single dog or a list of all dogs.

There are two steps to this:

- Execute a query on the dogs table. This produces a ListMap>.
- Change the ListMap> to a ListDog>.

```
// A method that retrieves all the dogs from the dogs table.
Future<List<Dog>> dogs() async {
  // Get a reference to the database.
  final db = await database;

  // Query the table for all The Dogs.
  final List<Map<String, dynamic>> maps = await db.query('dogs');

  // Convert the List<Map<String, dynamic> into a List<Dog>.
  return List.generate(maps.length, (i) {
    return Dog(
      id: maps[i]['id'],
      name: maps[i]['name'],
      age: maps[i]['age'],
    );  // Dog
  });  // List.generate
}
```

Update a Dog in the database.

There are two steps to this:

- Turn the Dog into a Map.
- Use a where clause to update the proper Dog.

```
Future<void> updateDog(Dog dog) async {
  // Get a reference to the database.
  final db = await database;

  // Update the given Dog.
  await db.update(
    'dogs',
    dog.toMap(),
    // Ensure that the Dog has a matching id.
    where: 'id = ?',
    // Pass the Dog's id as a whereArg to prevent SQL injection.
    whereArgs: [dog.id],
  );
}
```

8. Delete a Dog from the database

To delete data, we must utilize the sqflite library's delete() method.

In this section, we will write a function that accepts an id and deletes the dog with the same id from the database.

To make this work, we need to include a where clause to limit the records that are destroyed.

```
Future<void> deleteDog(int id) async {
  // Get a reference to the database.
  final db = await database;

  // Remove the Dog from the database.
  await db.delete(
    'dogs',
    // Use a `where` clause to delete a specific dog.
    where: 'id = ?',
    // Pass the Dog's id as a whereArg to prevent SQL injection.
    whereArgs: [id],
  );
}
```

1.3 Final Output: On Terminal

```
I/ViewRootImpl@6895bb9[MainActivity](19347): [DP] cancelDraw  isViewVisible: false
I/flutter (19347): [Dog{id: 0, name: Fido, age: 35}]
I/flutter (19347): [Dog{id: 0, name: Fido, age: 42}]
I/flutter (19347): []
```

1.4 Explanation of the code:

This Dart code is a simple demonstration of using the sqflite package to perform CRUD (Create, Read, Update, Delete) operations on a SQLite database in a Flutter application. Let's break down the code step by step:

1. Initialization:

- The WidgetsFlutterBinding.ensureInitialized() is called to ensure that Flutter widgets are initialized before any other operation.
- The openDatabase function is used to open or create a SQLite database. The database is named 'doggie_database.db' and its version is set to 1.

2. Database Table Definition:

- The onCreate callback is provided to create the 'dogs' table if it doesn't exist. This table has columns for 'id' (auto-incrementing primary key), 'name', and 'age'.

3. Dog Class:

- The Dog class represents a model for the data to be stored in the 'dogs' table.
- The class includes a toMap method to convert a Dog object into a Map, making it compatible with SQLite database operations.

4. Database Operations:

- insertDog: Inserts a new Dog record into the 'dogs' table.
- dogs: Retrieves all Dog records from the 'dogs' table.
- updateDog: Updates an existing Dog record in the 'dogs' table.
- deleteDog: Deletes a Dog record from the 'dogs' table based on the provided ID.

5. Example Usage:

- An instance of the Dog class named 'fido' is created and inserted into the 'dogs' table.
- The list of dogs is printed, displaying the inserted dog ('Fido').
- 'Fido's age is updated and the updated list of dogs is printed.
- 'Fido' is deleted from the database, and the final list of dogs (empty) is printed.

This code demonstrates a basic yet complete SQLite database interaction in Flutter, providing a foundation for managing data persistence within a Flutter application. The Dog class encapsulates the data model, and the DatabaseHelper functions facilitate the manipulation of the SQLite database. The example usage at the end showcases how to perform these CRUD operations in a practical scenario.

2 Task 2

Task-2

Firebase for Flutter
https://firebase.google.com/codelabs/firebase-auth-in-flutter-apps#0
Provide a documentation with a short description and screenshots for all the steps of the lab and the Dart files.

2.1 Brief overview of task 2

In this codelab, I delved into the integration of Firebase Authentication into Flutter applications by leveraging the FlutterFire UI package. This comprehensive guide not only covers the incorporation of email/password authentication but also extends to the integration of Google Sign-In authentication seamlessly within a Flutter app. The codelab walks through the essential steps of setting up a Firebase project, ensuring a robust foundation for app development. Additionally, iexplored the FlutterFire CLI, a powerful tool that simplifies the process of initializing Firebase within Flutter applications

2.2 Steps

Create and set up a Firebase project

In order to kickstart the integration of Firebase Authentication into my Flutter app using the FlutterFire UI package, the initial step is to create a dedicated Firebase project through the Firebase web console. I began by signing in to my Firebase account and navigating to the Firebase console. There, I initiated the project creation process by clicking on "Add Project" or "Create a project." I then named it "FlutterFire-UI-Codelab." This project will serve as the centralized hub for managing authentication functionalities within your Flutter app.

Get the starter code

To set up Firebase authentication and FlutterFire in your Flutter app, I started by cloning the provided sample code from the flutter-codelabs directory. Within the flutter-codelabs/firebase-auth-flutterfire-ui directory im given 2 choices,I selected the “start”one

Install Firebase CLI

After installing the CLI, authenticate with Firebase by running firebase login. This step connects your machine to Firebase and grants access to your projects. Confirm the CLI's installation and access by listing your Firebase projects with firebase projects:list. Ensure that the displayed projects match those in the Firebase console

```
C:\Users\kriti>firebase login
i  Firebase optionally collects CLI and Emulator Suite usage and error reporting information to help improve our product
s. Data is collected in accordance with Google's privacy policy (https://policies.google.com/privacy) and is not used to
 identify you.

? Allow Firebase to collect CLI and Emulator Suite usage and error reporting information? Yes
i  To change your data collection preference at any time, run `firebase logout` and log in again.

Visit this URL on this device to log in:
https://accounts.google.com/o/oauth2/auth?client_id=563584335869-fgrhgmd47bqnekij5i8b5pr03ho849e6.apps.googleusercontent
.com&scope=email%20openid%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fcloudplatformprojects.readonly%20https%3A%2F%2Fwww
.googleapis.com%2Fauth%2Ffirebase%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fcloud-platform&response_type=code&state=34
5157848&redirect_uri=http%3A%2F%2Flocalhost%3A9005

Waiting for authentication...

+  Success! Logged in as kritikabissessur3@gmail.com
```

Install the FlutterFire CLI

Next, install the FlutterFire CLI using dart pub global activate flutterfire_cli

```
C:\Users\kriti>dart pub global activate flutterfire_cli
Package flutterfire_cli is currently active at version 0.2.7.
The package flutterfire_cli is already activated at newest available version.
To recompile executables, first run `dart pub global deactivate flutterfire_cli`.
Installed executable flutterfire.
Activated flutterfire_cli 0.2.7.
```

Add your Firebase project to your Flutter app

To seamlessly integrate Firebase into your Flutter app, use FlutterFire to generate the necessary Dart code. Run flutterfire configure and follow the prompts to select your Firebase project and set up platforms. This process streamlines the setup of Firebase authentication and FlutterFire in your Flutter app, providing a foundation for implementing Firebase features across different platforms.

```
PS C:\Users\kriti\flutter-codelabs\firebase-auth-flutterfire-ui\start> flutterfire configure --project=complete-9a266
i Found 1 Firebase projects. Selecting project complete-9a266.
? Which platforms should your configuration support (use arrow keys & space to select)? ›
√ android
√ ios
√ macos
√ web
```

Configure FlutterFireAfter setup, check your Flutter app. The FlutterFire CLI creates firebase_options.dart with Firebase configuration. If you chose all platforms in flutterfire configure, find static values (web, android, ios, macos) in this file. They hold platform-specific Firebase settings for easy integration.

```dart
firebase_options.dart
lib > firebase_options.dart > ...
  }

  static const FirebaseOptions web = FirebaseOptions(
    apiKey: 'AIzaSyBK8h2vJ95khSxx0xlSMy5iF_x2zhtVV2Y',
    appId: '1:893054424251:web:4e16dfae6fedf6f58a859e',
    messagingSenderId: '893054424251',
    projectId: 'complete-453ec',
    authDomain: 'complete-453ec.firebaseapp.com',
    storageBucket: 'complete-453ec.appspot.com',
  );

  static const FirebaseOptions android = FirebaseOptions(
    apiKey: 'AIzaSyC4yrmAKRAhvfpPYIRwdy2BrJV53YdgqX4',
    appId: '1:893054424251:android:442ff3bb8979afe18a859e',
    messagingSenderId: '893054424251',
    projectId: 'complete-453ec',
    storageBucket: 'complete-453ec.appspot.com',
  );

  static const FirebaseOptions ios = FirebaseOptions(
    apiKey: 'AIzaSyACVOlwxmY5_XSH10w20nr8JQLvN-_AnrQ',
    appId: '1:893054424251:ios:c1be6f94a609f4c08a859e',
    messagingSenderId: '893054424251',
    projectId: 'complete-453ec',
    storageBucket: 'complete-453ec.appspot.com',
    iosBundleId: 'com.example.complete',
  );

  static const FirebaseOptions macos = FirebaseOptions(
    apiKey: 'AIzaSyACVOlwxmY5_XSH10w20nr8JQLvN-_AnrQ',
    appId: '1:893054424251:ios:db564d40e5ea35098a859e',
    messagingSenderId: '893054424251',
    projectId: 'complete-453ec',
    storageBucket: 'complete-453ec.appspot.com',
    iosBundleId: 'com.example.complete.RunnerTests',
  );
}
```

```dart
///   options: DefaultFirebaseOptions.currentPlatform,
/// );
/// ```
class DefaultFirebaseOptions {
  static FirebaseOptions get currentPlatform {
    if (kIsWeb) {
      return web;
    }
    switch (defaultTargetPlatform) {
      case TargetPlatform.android:
        return android;
      case TargetPlatform.iOS:
        return ios;
      case TargetPlatform.macOS:
        return macos;
      case TargetPlatform.windows:
        throw UnsupportedError(
          'DefaultFirebaseOptions have not been configured for windows - '
          'you can reconfigure this by running the FlutterFire CLI again.',
        );
      case TargetPlatform.linux:
        throw UnsupportedError(
          'DefaultFirebaseOptions have not been configured for linux - '
          'you can reconfigure this by running the FlutterFire CLI again.',
        );
      default:
        throw UnsupportedError(
          'DefaultFirebaseOptions are not supported for this platform.',
        );
    }
  }
```

For the last setup step, add essential Firebase packages to your Flutter project. The firebase_options.dart file might have errors initially, as it depends on packages not yet included. In the terminal, navigate to the project root at flutter-codelabs/firebase-emulator-suite/start, and execute three commands to resolve dependencies and integrate Firebase packages seamlessly.

```
Learn more about using this file and next steps from the documentation:
 > https://firebase.google.com/docs/flutter/setup
PS C:\Users\kriti\flutter-codelabs\firebase-auth-flutterfire-ui\start> cd..
PS C:\Users\kriti\flutter-codelabs\firebase-auth-flutterfire-ui> cd..
PS C:\Users\kriti\flutter-codelabs> cd firebase-emulator-suite/start
PS C:\Users\kriti\flutter-codelabs\firebase-emulator-suite\start>
```

Add Firebase packages to Flutter app

```
PS C:\Users\kriti\flutter-codelabs\firebase-emulator-suite\start> flutter pub add firebase_core
Resolving dependencies...
+ async 2.11.0
+ boolean_selector 2.1.1
+ characters 1.3.0
+ clock 1.1.1
+ collection 1.18.0
+ cupertino_icons 1.0.6
+ fake_async 1.3.1
+ firebase_core 2.24.2
+ firebase_core_platform_interface 5.0.0
+ firebase_core_web 2.10.0
+ flutter 0.0.0 from sdk flutter
+ flutter_lints 3.0.1
+ flutter_test 0.0.0 from sdk flutter
+ flutter_web_plugins 0.0.0 from sdk flutter
+ go_router 12.1.3
+ js 0.6.7
+ lints 3.0.0
+ logging 1.2.0
+ matcher 0.12.16
+ material_color_utilities 0.5.0 (0.8.0 available)
+ meta 1.10.0 (1.11.0 available)
+ path 1.8.3 (1.9.0 available)
+ plugin_platform_interface 2.1.7
+ sky_engine 0.0.99 from sdk flutter
+ source_span 1.10.0
+ stack_trace 1.11.1
+ stream_channel 2.1.2
+ string_scanner 1.2.0
+ term_glyph 1.2.1
```

```
Try `flutter pub outdated` for more information.
PS C:\Users\kriti\flutter-codelabs\firebase-emulator-suite\start> flutter pub add firebase_auth
Resolving dependencies...
+ _flutterfire_internals 1.3.16
+ firebase_auth 4.15.2
+ firebase_auth_platform_interface 7.0.8
+ firebase_auth_web 5.8.11
+ http_parser 4.0.2
  material_color_utilities 0.5.0 (0.8.0 available)
  meta 1.10.0 (1.11.0 available)
  path 1.8.3 (1.9.0 available)
+ typed_data 1.3.2
  web 0.3.0 (0.4.0 available)
Changed 6 dependencies!
4 packages have newer versions incompatible with dependency constraints.
Try `flutter pub outdated` for more information.
```

```
PS C:\Users\kriti\flutter-codelabs\firebase-emulator-suite\start> flutter pub add firebase_ui_auth
Resolving dependencies...
+ args 2.4.2
+ crypto 3.0.3
+ desktop_webview_auth 0.0.14
+ email_validator 2.1.17
+ firebase_dynamic_links 5.4.8
+ firebase_dynamic_links_platform_interface 0.2.6+16
+ firebase_ui_auth 1.11.0
+ firebase_ui_localizations 1.9.0
+ firebase_ui_oauth 1.4.15
+ firebase_ui_shared 1.4.1
+ flutter_localizations 0.0.0 from sdk flutter
+ flutter_svg 2.0.9
+ http 1.1.2
+ intl 0.18.1 (0.19.0 available)
  material_color_utilities 0.5.0 (0.8.0 available)
  meta 1.10.0 (1.11.0 available)
  path 1.8.3 (1.9.0 available)
+ path_parsing 1.0.1
+ petitparser 6.0.2
+ vector_graphics 1.1.9+1
+ vector_graphics_codec 1.1.9+1
+ vector_graphics_compiler 1.1.9+1
  web 0.3.0 (0.4.0 available)
+ xml 6.5.0
Changed 20 dependencies!
5 packages have newer versions incompatible with dependency constraints.
```

Initialize Firebase

In order to use the packages added, and the DefaultFirebaseOptions.currentPlatform, update the code in the main function in the main.dart file.

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

import 'app.dart';
import 'firebase_options.dart';

Run | Debug | Profile
void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp(
    options: DefaultFirebaseOptions.currentPlatform,
  );


  runApp(const MyApp());
}
```

This code snippet serves two main purposes:

WidgetsFlutterBinding.ensureInitialized() instructs Flutter to delay the execution of the application widget code until the Flutter framework is fully booted. This is crucial because Firebase utilizes native platform channels, and these require the Flutter framework to be up and running.

Firebase.initializeApp establishes a connection between your Flutter app and your Firebase project. The static value DefaultFirebaseOptions.currentPlatform is imported from the generated firebase_options.dart file. This value dynamically detects the platform on which the app is running and supplies the corresponding Firebase keys, ensuring seamless integration with Firebase services.

Add initial Firebase UI Auth page

Firebase UI for Auth offers widgets representing entire screens for various authentication flows like Sign In, Registration, Forgot Password, and User Profile. Start by adding an authentication guard landing page to your app. Ensure your app is wrapped in MaterialApp or CupertinoApp, with UI reflecting Material or Cupertino widgets automatically based on your choice. In this codelab, MaterialApp is used, already included in app.dart.

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

import 'auth_gate.dart';

class MyApp extends StatelessWidget {
  const MyApp({super.key});
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      theme: ThemeData(
        colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),
        useMaterial3: true,
      ), // ThemeData
      home: const AuthGate(),
    ); // MaterialApp
  }
}
```

Check authentication state

- Utilizing StreamBuilder.stream with FirebaseAuth.instance.authStateChanged to capture changes in user authentication status.
- The stream returns a Firebase User object upon authentication, otherwise null.
- Verification of authentication status using snapshot.hasData.
- If no User object is present, the code displays a SignInScreen widget.
- If authentication is successful, the code renders a HomeScreen accessible only to authenticated users.

```dart
auth_gate.dart
lib > auth_gate.dart > AuthGate
import 'package:firebase_auth/firebase_auth.dart' hide EmailAuthProvider;
import 'package:firebase_ui_auth/firebase_ui_auth.dart';
import 'package:flutter/material.dart';

import 'home.dart';

class AuthGate extends StatelessWidget {
  const AuthGate({super.key});

  @override
  Widget build(BuildContext context) {
    return StreamBuilder<User?>(
      stream: FirebaseAuth.instance.authStateChanges(),
      builder: (context, snapshot) {
        if (!snapshot.hasData) {
          return SignInScreen(
            providers: [],
          ); // SignInScreen
        }

        return const HomeScreen();
      },
    ); // StreamBuilder
  }
}
```

Sign-In screen

The SignInScreen widget, provided by FlutterFire UI, adds the following functionality:

Allows users to sign in

If users forgot their password, they can tap "Forgot password?" and be taken to a form to reset their password

If a user isn't yet registered, they can tap "Register", and be taken to another form that allows them to sign up.

```dart
auth_gate.dart
lib > auth_gate.dart > AuthGate
import 'package:firebase_auth/firebase_auth.dart' hide EmailAuthProvider;
import 'package:firebase_ui_auth/firebase_ui_auth.dart';
import 'package:flutter/material.dart';

import 'home.dart';

class AuthGate extends StatelessWidget {
  const AuthGate({super.key});

  @override
  Widget build(BuildContext context) {
    return StreamBuilder<User?>(
      stream: FirebaseAuth.instance.authStateChanges(),
      builder: (context, snapshot) {
        if (!snapshot.hasData) {
          return SignInScreen(
            providers: [
              EmailAuthProvider(), // new
            ],
          ); // SignInScreen
        }

        return const HomeScreen();
      },
    ); // StreamBuilder
  }
}
```

The SignInScreen widget, and its providers argument, is the only code required to get all the aforementioned functionality.

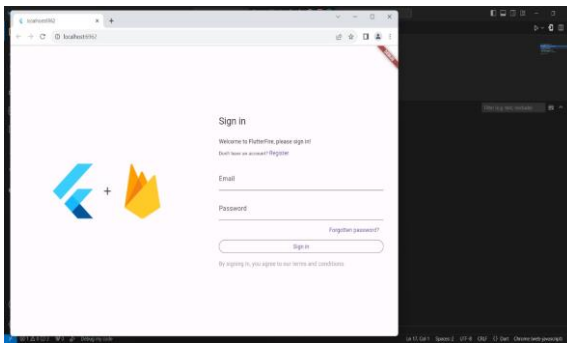
While functional, it lacks styling. The widget exposes parameters to customize the sign-in screen's look.

Customize the sign-in Screen

The headerBuilder argument requires a function of the type HeaderBuilder, which is defined in the FlutterFire UI package.

```
auth_gate.dart
lib > auth_gate.dart > AuthGate
import 'package:firebase_auth/firebase_auth.dart' hide EmailAuthProvider;
import 'package:firebase_ui_auth/firebase_ui_auth.dart';
import 'package:flutter/material.dart';

import 'home.dart';

class AuthGate extends StatelessWidget {
 const AuthGate({super.key});

 @override
 Widget build(BuildContext context) {
   return StreamBuilder<User?>(
     stream: FirebaseAuth.instance.authStateChanges(),
     builder: (context, snapshot) {
       if (!snapshot.hasData) {
         return SignInScreen(
           providers: [
             EmailAuthProvider(),
           ],
           headerBuilder: (context, constraints, shrinkOffset) {
             return Padding(
               padding: const EdgeInsets.all(20),
               child: AspectRatio(
                 aspectRatio: 1,
                 child: Image.asset('assets/flutterfire_300x.png'),
               ), // AspectRatio
             ); // Padding
           },
         ); // SignInScreen
       }

       return const HomeScreen();
     },
   ); // StreamBuilder
 }
}
```

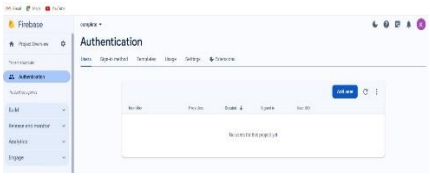
Now,it looks like this



Create a user

To create a user for sign-in, i manually created one via the Firebase console:

Access the Firebase console and navigate to the "Users" table.

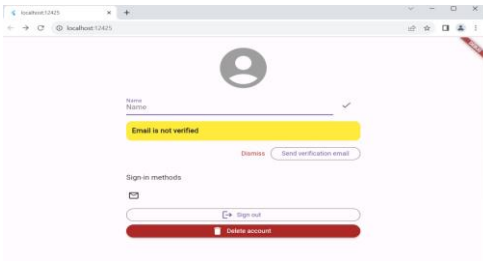


In the "Users" table, proceed with creating a new user through the provided options,here I have added my email and password to access the sign-in.

At the end when I have finished with the processes above,I should sign in again with the email and password I entered while adding user,Here is what my screen looks like now:

Then I updated the code in the home.dart file.

In the updated code, attention is drawn to the callback used with IconButton.isPressed. Upon pressing this particular button, the application generates a new anonymous route, navigating to it. This route is responsible for displaying the ProfileScreen widget, returned from the MaterialPageRoute.builder callback.



Note:The sign out button doesn't doesn't navigate me back to the AuthGate widget.In order to implement this,I have used Profile Screens Parameter

Profile Screen

When creating a ProfileScreen instance, include a list of actions in the ProfileScreen.actions argument. These actions, of type FlutterFireUiAction, guide your app's response to authentication state changes. For instance, the SignedOutAction triggers a callback, like Navigator.of(context).pop(), when the Firebase auth state indicates a null currentUser.

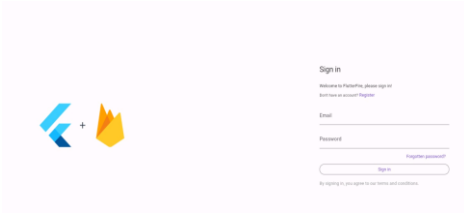
Note:Adding this callback ensures that when a user signs out, the app navigates to the previous page. In this app, with only one permanent route displaying either the sign-in or home page based on user status, signing out directs the app to the sign-in page.

In the ProfileScreen widget, the optional children argument takes a list of widgets. These widgets are vertically arranged within a Column widget used internally to construct the ProfileScreen. The Column widget, present in the ProfileScreen build method, positions the provided children above the "Sign out" button.

I need to modify the code in home.dart to display the company logo in a manner similar to the sign-in screen.

Upon reloading my app, i observed the following on the screen:

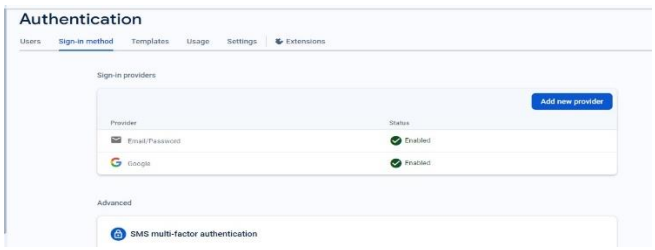
And the sign out button works perfectly



FlutterFire UI supports 3rd party provider authentication like Google, Twitter, Facebook, Apple, and Github. For Google authentication integration, install the firebase_ui_oauth_google plugin and its dependencies by running the provided command in my Flutter project's root directory.

```
PS C:\Users\kriti\flutter-codelabs\firebase-auth-flutterfire-ui\start> flutter pub add google_sign_in
"google_sign_in" is already in "dependencies". Will try to update the constraint.
Resolving dependencies...
  _fe_analyzer_shared 61.0.0 (65.0.0 available)
  analyzer 5.13.0 (6.3.0 available)
  intl 0.18.1 (0.19.0 available)
  material_color_utilities 0.5.0 (0.8.0 available)
  meta 1.10.0 (1.11.0 available)
  path 1.8.3 (1.9.0 available)
  pigeon 11.0.1 (14.0.1 available)
  web 0.3.0 (0.4.0 available)
Got dependencies!
```

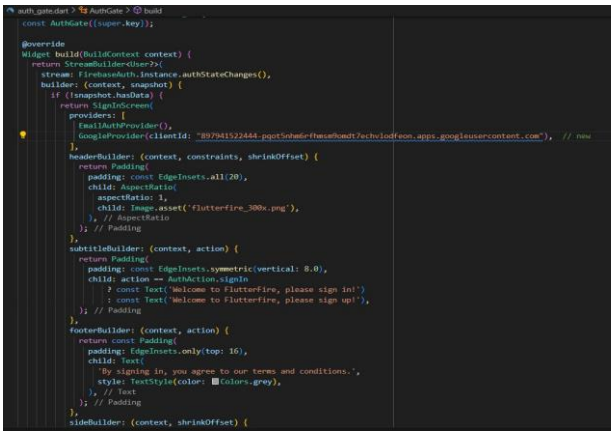
```
PS C:\Users\kriti\flutter-codelabs\firebase-auth-flutterfire-ui\start> flutter pub add firebase_ui_oauth_google
"firebase_ui_oauth_google" is already in "dependencies". Will try to update the constraint.
Resolving dependencies...
  _fe_analyzer_shared 61.0.0 (65.0.0 available)
  analyzer 5.13.0 (6.3.0 available)
  intl 0.18.1 (0.19.0 available)
  material_color_utilities 0.5.0 (0.8.0 available)
  meta 1.10.0 (1.11.0 available)
  path 1.8.3 (1.9.0 available)
  pigeon 11.0.1 (14.0.1 available)
  web 0.3.0 (0.4.0 available)
Got dependencies!
8 packages have newer versions incompatible with dependency constraints.
Try `flutter pub outdated` for more information.
PS C:\Users\kriti\flutter-codelabs\firebase-auth-flutterfire-ui\start>
```



To enable Google provider in Firebase Console:

1. Go to the Authentication sign-in providers screen.
2. Click "Add new provider" and select Google.

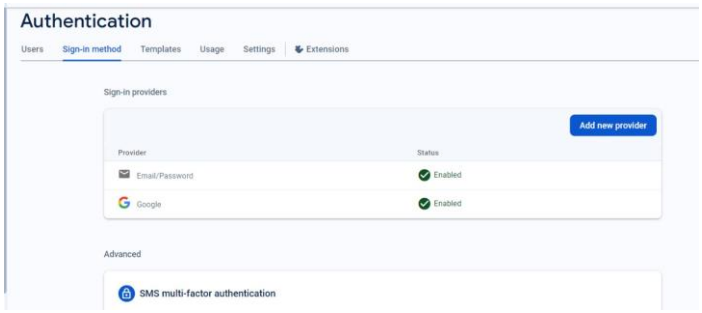
With Google sign-in enabled, add the widget needed to display a stylized Google sign-in button to the sign in page. Navigate to auth_gate.dart file and update the code to the following:



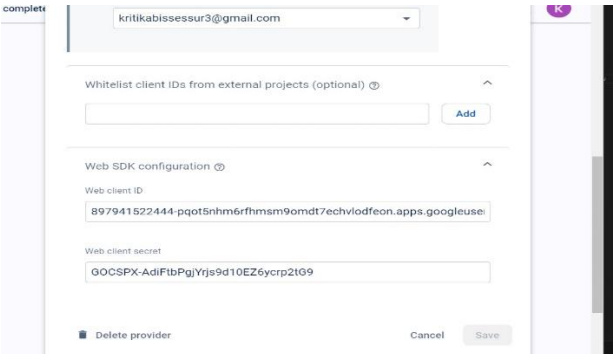
The button doesn't work without additional configuration. If you're developing with Flutter Web, this is the only step you have to add for this to work. Other platforms require additional steps, which are discussed in a bit.

Navigate to the Authentication providers page in the Firebase Console.

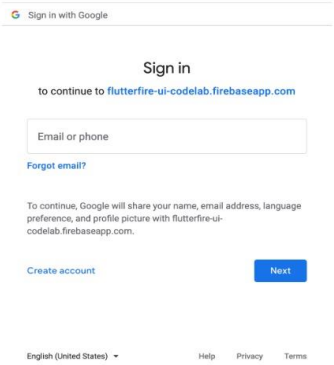
Click on Google provider.



I copied the value from ‘Web client ID"



Once the web client ID is entered, reload your app. When I press the "Sign in with Google" button, a new window will appear (if you're using web) that walks you through the Google sign in flow. Initially, it looks like this



3 Task 3

Task-3

Get to know Firebase for Flutter
https://firebase.google.com/codelabs/firebase-get-to-know-flutter#0
Provide a documentation with a short description and screenshots for all the steps of the lab and the Dart files.

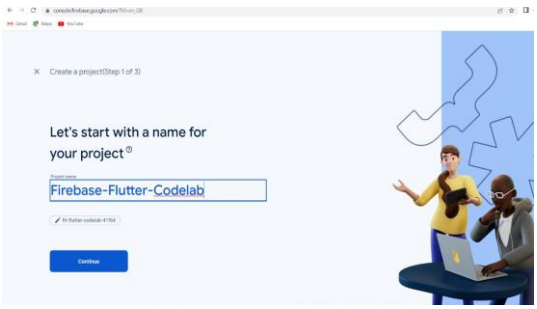
3.1 Brief overview

In Task 3, I learned how to construct a versatile event RSVP and guestbook chat app using Flutter for Android, iOS, the Web, and macOS. The tutorial guided me through the process of implementing user authentication with Firebase Authentication and efficiently synchronizing data with Firestore. This hands-on experience provided insights into building a cross-platform application that seamlessly integrates authentication mechanisms and real-time data synchronization

3.2 Steps

After cloning the codelabs,I should navigate to 'get to know' then step 2

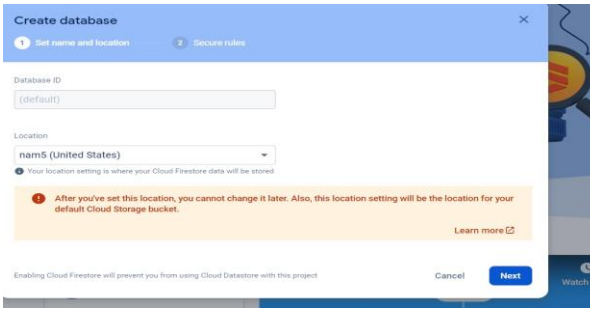
Next,I create a project in my firebase console and name it **Firebase-Flutter-Codelab**



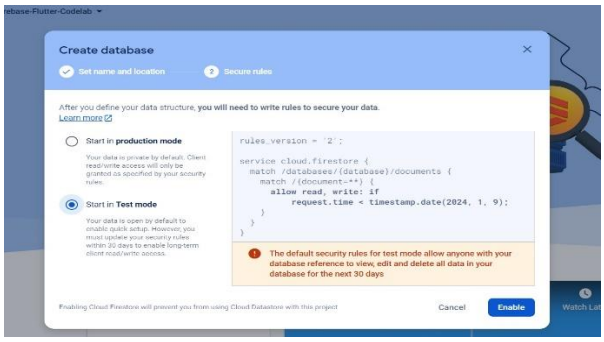
The web app uses Firestore to save chat messages and receive new chat messages.

Enable Firestore:

In the **Build** menu, click **Firestore Database > Create database**.Then I should Select **Start in test mode** and then read the disclaimer about the security rules. Test mode ensures that you can freely write to the database during development.



Click **Next** and then select the location for my database.



I need to add the FlutterFire libraries for the two Firebase products that I use in this app: Authentication and Firestore.

The **firebase_core package** is the common code required for all Firebase Flutter plugins.

The **firebase_auth package** enables integration with Authentication.

```
PS C:\Users\kriti\flutter-codelabs\firebase-get-to-know-flutter\step_02> flutter pub add firebase_auth
Resolving dependencies...
+ _flutterfire_internals 1.3.16
+ firebase_auth 4.15.2
+ firebase_auth_platform_interface 7.0.8
+ firebase_auth_web 5.8.11
  material_color_utilities 0.5.0 (0.8.0 available)
  meta 1.10.0 (1.11.0 available)
  path 1.8.3 (1.9.0 available)
  web 0.3.0 (0.4.0 available)
Changed 4 dependencies!
4 packages have newer versions incompatible with dependency constraints.
Try `flutter pub outdated` for more information.
PS C:\Users\kriti\flutter-codelabs\firebase-get-to-know-flutter\step_02>
```

The cloud_firestore package enables access to Firestore data storage.

```
PS C:\Users\kriti\flutter-codelabs\firebase-get-to-know-flutter\step_02> flutter pub add cloud_firestore
Resolving dependencies...
+ cloud_firestore 4.13.5
+ cloud_firestore_platform_interface 6.0.9
+ cloud_firestore_web 3.8.9
  material_color_utilities 0.5.0 (0.8.0 available)
  meta 1.10.0 (1.11.0 available)
  path 1.8.3 (1.9.0 available)
  web 0.3.0 (0.4.0 available)
Changed 3 dependencies!
4 packages have newer versions incompatible with dependency constraints.
Try `flutter pub outdated` for more information.
PS C:\Users\kriti\flutter-codelabs\firebase-get-to-know-flutter\step_02>
```

The firebase_ui_auth package provides a set of widgets and utilities to increase developer velocity with authentication flows.

```
PS C:\Users\kriti\flutter-codelabs\firebase-get-to-know-flutter\step_02> flutter pub add firebase_ui_auth
Resolving dependencies...
+ args 2.4.2
+ desktop_webview_auth 0.0.14
+ email_validator 2.1.17
+ firebase_dynamic_links 5.4.8
+ firebase_dynamic_links_platform_interface 0.2.6+16
+ firebase_ui_auth 1.11.0
+ firebase_ui_localizations 1.9.0
+ firebase_ui_oauth 1.6.15
+ firebase_ui_shared 1.4.1
+ flutter_localizations 0.0.0 from sdk flutter
+ flutter_svg 2.0.9
+ intl 0.18.1 (0.19.0 available)
  material_color_utilities 0.5.0 (0.8.0 available)
  meta 1.10.0 (1.11.0 available)
  path 1.8.3 (1.9.0 available)
+ path_parsing 1.0.1
+ petitparser 5.0.2
+ vector_graphics 1.1.9+1
+ vector_graphics_codec 1.1.9+1
+ vector_graphics_compiler 1.1.9+1
  web 0.3.0 (0.4.0 available)
+ xml 6.5.0
Changed 18 dependencies!
5 packages have newer versions incompatible with dependency constraints.
Try `flutter pub outdated` for more information.
PS C:\Users\kriti\flutter-codelabs\firebase-get-to-know-flutter\step_02>
```

The FlutterFire CLI depends on the underlying Firebase CLI.

```
PS C:\Users\kriti\flutter-codelabs\firebase-get-to-know-flutter\step_02> dart pub global activate flutterfire_cli
Package flutterfire_cli is currently active at version 0.2.7.
The package flutterfire_cli is already activated at newest available version.
To recompile executables, first run `dart pub global deactivate flutterfire_cli`.
Installed executable flutterfire.
Activated flutterfire_cli 0.2.7.
```

The CLI extracts information from my Firebase project and selected project apps to generate all the configuration for a specific platform.

In the root of my app, run the **configure** command:

```
PS C:\Users\kriti\flutter-codelabs\firebase-get-to-know-flutter\step_02> flutterfire configure --project=fir-flutter-codelab-ef3aa
i Found 3 Firebase projects. Selecting project fir-flutter-codelab-ef3aa.
? Which platforms should your configuration support (use arrow keys & space to select)? ›
√ android
√ ios
√ macos
√ web
```

Create a new file named `app_state.dart` with the following content:

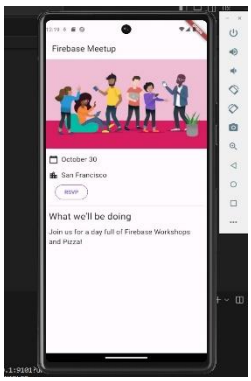
I then modify the imports at the top of the `lib/main.dart` file:

```
import 'package:firebase_ui_auth/firebase_ui_auth.dart'; // new
import 'package:go_router/go_router.dart';                // new
import 'package:provider/provider.dart';                  // new
import 'app_state.dart';                                  // new
```

Then I need to connect the app state with the app initialization and then add the authentication flow to `HomePage`:

Update your app to handle navigation to different screens that FirebaseUI provides for you, by creating a `GoRouter` configuration:

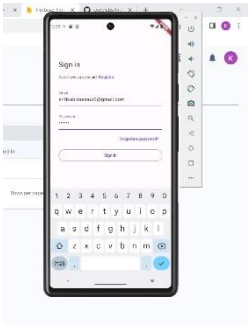
In the `HomePage` class's build method, integrate the app state with the `AuthFunc` widget:



Now I should test the authentication flow:

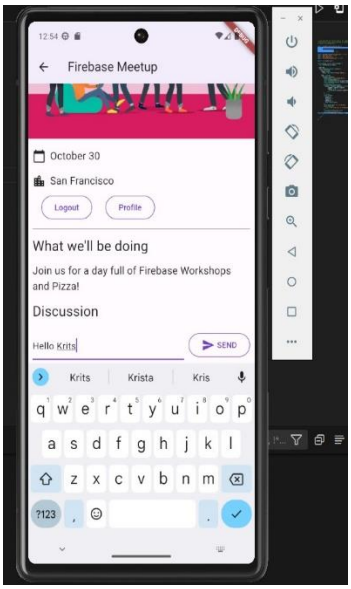
```
body: ListView(
  children: <Widget>[
    Image.asset('assets/codelab.png'),
    const SizedBox(height: 8),
    const IconAndDetail(Icons.calendar_today, 'October 30'),
    const IconAndDetail(Icons.location_city, 'San Francisco'),
    // Add from here
    Consumer<ApplicationState>(
      builder: (context, appState, _) => AuthFunc(
          loggedIn: appState.loggedIn,
          signOut: () {
            FirebaseAuth.instance.signOut();
          }), // AuthFunc
    ), // Consumer
    // to here
    const Divider(
      height: 8,
      thickness: 1,
      indent: 8,
      endindent: 8,
      color: Colors.grey,
    ), // Divider
    const Header("What we'll be doing"),
    const Paragraph(
      'Join us for a day full of Firebase Workshops and Pizza!',
    ), // Paragraph
  1. // <Widget>[]
```

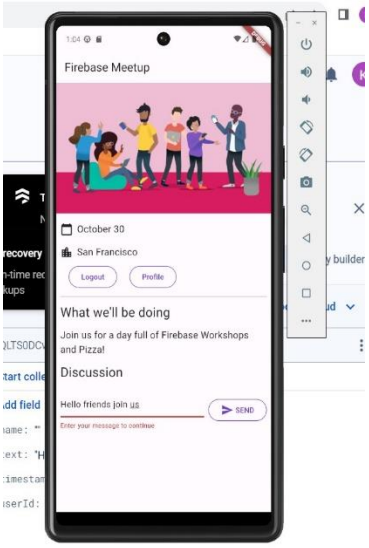
In the app, tap the **RSVP** button to initiate the SignInScreen.



Create a new file named guest_book.dart, add a GuestBook stateful widget to construct the UI elements of a message field and a send button:

When a user clicks **SEND**, it triggers the following code snippet. It adds the contents of the message input field to the guestbook collection of the database. Specifically, the addMessageToGuestBook method adds the message content to a new document with an automatically generated ID in the guestbook collection.





In the lib/app_state.dart file, I add the addMessageToGuestBook method. I connect this capability with the user interface in the next step.

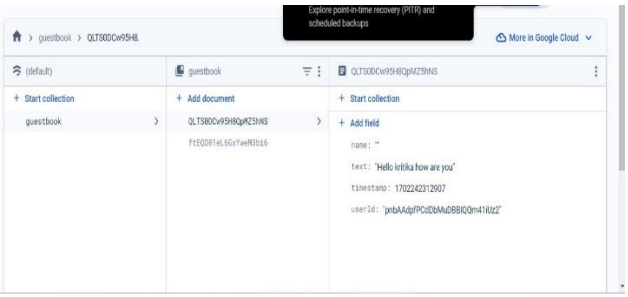
```
Future<DocumentReference> addMessageToGuestBook(String message) {
  if (!_loggedIn) {
    throw Exception('Must be logged in');
  }

  return FirebaseFirestore.instance
      .collection('guestbook')
      .add(<String, dynamic>{
    'text': message,
    'timestamp': DateTime.now().millisecondsSinceEpoch,
    'name': FirebaseAuth.instance.currentUser!.displayName,
    'userId': FirebaseAuth.instance.currentUser!.uid,
  });
}
// ...to here.
}
```

In the lib/home_page.dart file, make the following change to the HomePage widget:

```
import 'package:firebase_auth/firebase_auth.dart'
    hide EmailAuthProvider, PhoneAuthProvider;
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';

import 'app_state.dart';
import 'guest_book.dart';
import 'src/authentication.dart';
import 'src/widgets.dart';
```

```
  ), // Paragraph
  Consumer<ApplicationState>(
    builder: (context, appState, _) => Column(
      crossAxisAlignment: CrossAxisAlignment.start,
      children: [
        if (appState.loggedIn) ...[
          const Header('Discussion'),
          GuestBook(
            addMessage: (message) =>
                appState.addMessageToGuestBook(message),
          ), // GuestBook
        ],
      ],
    ), // Column
  ), // Consumer
], // <Widget>[]
 // ListView
/ Scaffold
```

Here I can see my added message in the guestbook collection

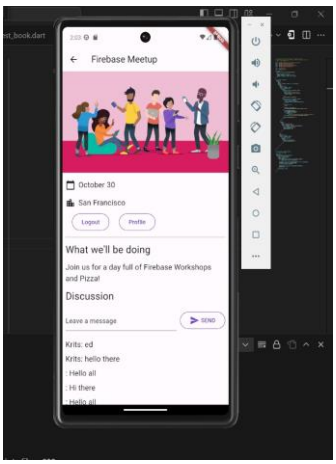


Create a new file guest_book_message.dart, add the following class to expose a structured view of the data that you store in Firestore.

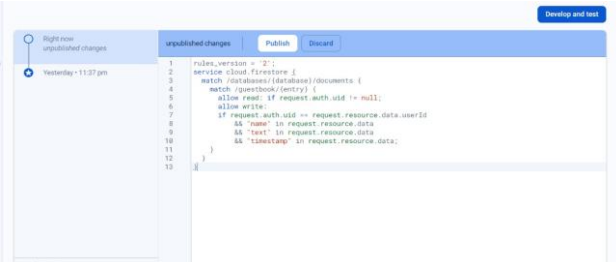
In the lib/app_state.dart file, add the following imports:

I update the body of HomePage to correctly construct GuestBook with the new messages parameter:

App Preview:



In match /databases/{database}/documents, I identify the collection that I want to secure:



```
        options: DefaultFirebaseOptions.currentPlatform);

    FirebaseUIAuth.configureProviders([
      EmailAuthProvider(),
    ]);
    // Add from here...
    FirebaseFirestore.instance
        .collection('attendees')
        .where('attending', isEqualTo: true)
        .snapshots()
        .listen((snapshot) {
      _attendees = snapshot.docs.length;
      notifyListeners();
    });
    // ...to here.

    FirebaseAuth.instance.userChanges().listen((user) {
```

Create a new file yes_no_selection.dart, define a new widget that acts like radio buttons:

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

import 'app_state.dart';
import 'src/widgets.dart';

class YesNoSelection extends StatelessWidget {
  const YesNoSelection({
    super.key,
    required this.state,
    required this.onSelection,
  });

  final Attending state;
  final void Function(Attending selection) onSelection;

  @override
  Widget build(BuildContext context) {
    switch (state) {
      case Attending.yes:
        return Padding(
          padding: const EdgeInsets.all(8.0),
          child: Row(
            children: [
              FilledButton(
                onPressed: () => onSelection(Attending.yes),
                child: const Text('YES'),
              ), // FilledButton
```

```
            children: [

// Add from here...
      switch (appState.attendees) {
        1 => const Paragraph('1 person going'),
        >= 2 => Paragraph('${appState.attendees} people going'),
        _ => const Paragraph('No one going'),
      },
      // ...to here.
```

```
          if (appState.loggedIn) ...[
            // Add from here...
      YesNoSelection(
        state: appState.attending,
        onSelection: (attending) => appState.attending = attending,
      ),
      // ...to here.
            const Header('Discussion'),
            GuestBook(
              addMessage: (message) =>
                  appState.addMessageToGuestBook(message),
              messages: appState.guestBookMessages,
            ), // GuestBook
          ],
        ],
```

I already set up some rules, so the data that I add with the buttons will be rejected. I need to update the rules to allow additions to the attendees collection.

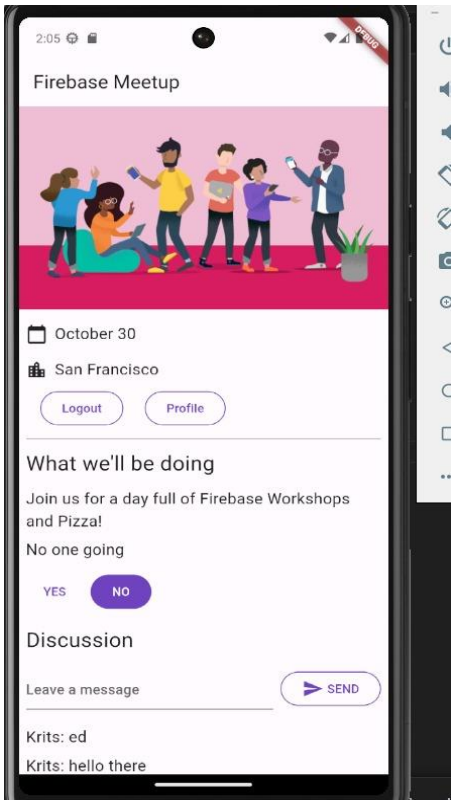
In the attendees collection, grab the Authentication UID that I used as the document name and verify that the submitter's uid is the same as the document they're writing:

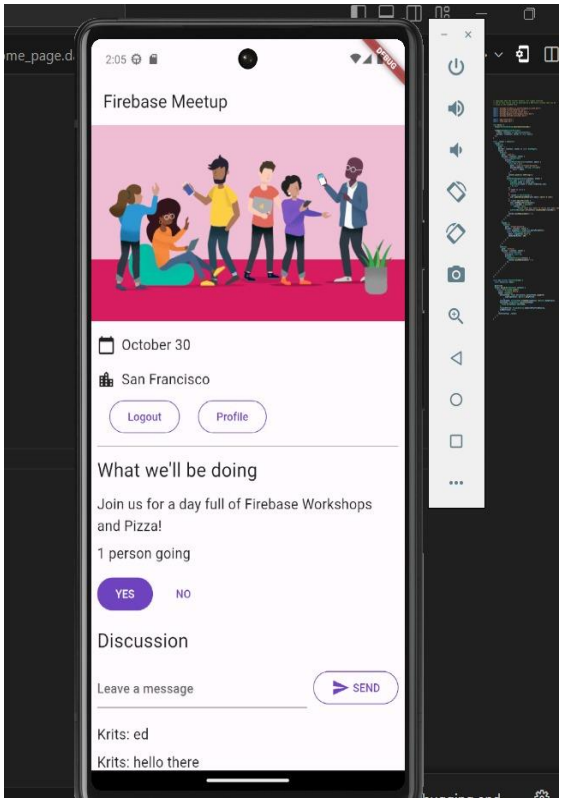
Add data validation to ensure that all the expected fields are present in the document:

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {
    // ... //
    match /attendees/{userId} {
      allow read: if true;
      allow write: if request.auth.uid == userId
          && "attending" in request.resource.data;

    }
  }
}
```

Note: This lets everyone read the attendees list because there's no private data there, but only the creator can update it.

3.3 Final Output:





4 Task 4

Task-4

Local development for your Flutter apps using the Firebase Emulator Suite
https://firebase.google.com/codelabs/get-started-firebase-emulators-and-flutter#0
Provide a documentation with a short description and screenshots for all the steps of the lab and the Dart files.

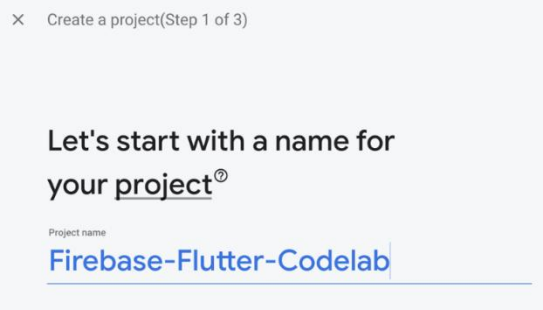
4.1 Brief Overview

In this codelab, I learned how to use the Firebase Emulator Suite with Flutter during local development. I learned how to use email-password authentication via the Emulator Suite and how to read and write data to the Firestore emulator. Finally, I worked with importing and exporting data from the emulators to work with the same faked data each time I returned to development.

4.2 Steps

Create and set up a Firebase project

The first was creating a Firebase project in Firebase's web console. A vast majority of this codelab will focus on the Emulator Suite, which uses a locally running UI



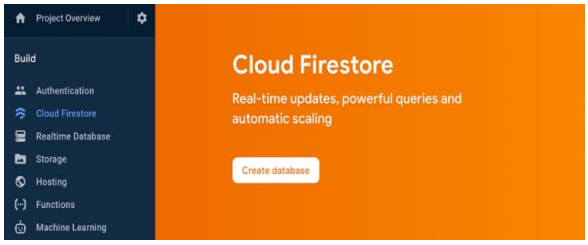
Enable Cloud Firestore

The Flutter app uses Cloud Firestore to save journal entries.

Enable Cloud Firestore:

In the Firebase console's **Build** section, click **Cloud Firestore**.

Click **Create database**.



Set up the Flutter app

I downloaded the starter code, and logged in the firebase account

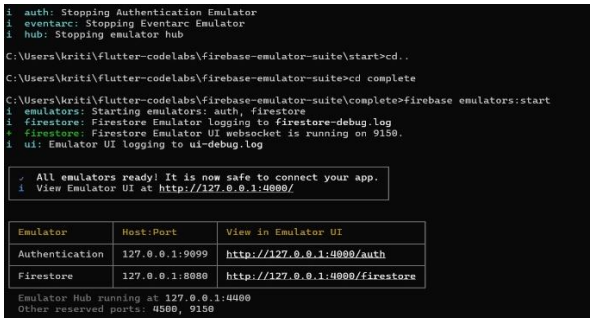
Use Firebase CLI and FlutterFire CLI to add your Firebase project to your Flutter app

With the two CLIs installed, you can set up individual Firebase products (like Firestore), download the emulators, and add Firebase to your Flutter app with just a couple of terminal commands.

First, finish Firebase set up by running the following:

firebase init

When prompted to select features, select "Firestore" and "Emulators". (There is no Authentication option, as it doesn't use configuration that's modifiable from your Flutter project files.)



Next, select "Use an existing project", when prompted.

```
C:\Users\kriti\flutter-codelabs\firebase-emulator-suite\start

? Are you ready to proceed? Yes
? Which Firebase features do you want to set up for this directory? Press Space to select features, then Enter to confirm your choices. Firestore: Configure
 security rules and indexes files for Firestore, Emulators: Set up local emulators for Firebase products

=== Project Setup

First, let's associate this project directory with a Firebase project.
You can create multiple project aliases by running firebase use --add,
but for now we'll just set up a default project.

? Please select an option: (Use arrow keys)
> Use an existing project
  Create a new project
  Add Firebase to an existing Google Cloud Platform project
  Don't set up a default project
```

Now, select the project you created in a previous step: flutter-firebase-codelab.

```
=== Project Setup

First, let's associate this project directory with a Firebase project.
You can create multiple project aliases by running firebase use --add,
but for now we'll just set up a default project.

? Please select an option: Use an existing project
? Select a default Firebase project for this directory:
  complete-9a266 (complete)
> fir-flutter-codelab-41f64 (Firebase-Flutter-Codelab)
  flutterfire-ui-codelab-97045 (FlutterFire-UI-Codelab)
```

Next, I am asked a series of questions about naming files that will be generated. I press "enter" for each question to select the default.

```
? What file should be used for Firestore Rules? firestore.rules

Firestore indexes allow you to perform complex queries while
maintaining performance that scales with the size of the result
set. You can keep index definitions in your project directory
and publish them with firebase deploy.

? What file should be used for Firestore indexes? firestore.indexes.json
```

Finally, I need to configure the emulators. I select Firestore and Authentication from the list, and then press "Enter" to each question about the specific ports to use for each emulator. I select the default, Yes, when asked if you I to use the Emulator UI.

```
? What file should be used for Firestore Indexes? firestore.indexes.json

=== Emulators Setup
? Which Firebase emulators do you want to set up? Press Space to select emulators, then Enter to confirm your choices. Authentication Emulator, Firestore
Emulator
? Which port do you want to use for the auth emulator? 9099
? Which port do you want to use for the firestore emulator? 8080
? Would you like to enable the Emulator UI? Yes
? Which port do you want to use for the Emulator UI (leave empty to use any available port)?
? Would you like to download the emulators now? Yes
i  firestore: downloading cloud-firestore-emulator-v1.18.2.jar...
Progress: ===================>------------------------------------------------------------------------------------------------------------ (18% of 64MB)
```

Configure FlutterFire

Next, I use FlutterFire to generate the needed Dart code to use Firebase in my Flutter app.

```
complete  C:\Users\kriti\flutter-codelabs\firebase-auth-flutterfire-ui
PROBLEMS 4   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

PS C:\Users\kriti\flutter-codelabs\firebase-emulator-suite> cd start
PS C:\Users\kriti\flutter-codelabs\firebase-emulator-suite\start> flutterfire configure --project=fir-flutter-codelab-41f64
i Found 3 Firebase projects. Selecting project fir-flutter-codelab-41f64.
✓ Which platforms should your configuration support (use arrow keys & space to select)? · android, ios, macos, web
⠋ Fetching registered android Firebase apps for project fir-flutter-codelab-41f64
```

When this command is run, I am prompted to select which Firebase project I want to use, and which platforms I want to set up.

```
i Found 3 Firebase projects. Selecting project fir-flutter-codelab-41f64.
✓ Which platforms should your configuration support (use arrow keys & space to select)? · android, ios, macos, web
i Firebase android app com.example.complete is not registered on Firebase project fir-flutter-codelab-41f64.
i Registered a new Firebase android app on Firebase project fir-flutter-codelab-41f64.
i Firebase ios app com.example.complete is not registered on Firebase project fir-flutter-codelab-41f64.
i Registered a new Firebase ios app on Firebase project fir-flutter-codelab-41f64.
i Firebase macos app com.example.complete.RunnerTests is not registered on Firebase project fir-flutter-codelab-41f64.
i Registered a new Firebase macos app on Firebase project fir-flutter-codelab-41f64.
i Firebase web app complete (web) is not registered on Firebase project fir-flutter-codelab-41f64.
i Registered a new Firebase web app on Firebase project fir-flutter-codelab-41f64.

Firebase configuration file lib\firebase_options.dart generated successfully with the following Firebase apps:

Platform  Firebase App Id
web       1:286459103273:web:72cc3b5e9a2bc5409bf438
android   1:286459103273:android:7609695ec091c2079bf438
ios       1:286459103273:ios:df08939d53cd346a9bf438
macos     1:286459103273:ios:0cfa1706fd6d73879bf438

Learn more about using this file and next steps from the documentation:
 > https://firebase.google.com/docs/flutter/setup
PS C:\Users\kriti\flutter-codelabs\firebase-emulator-suite\start>
```

Add Firebase packages to Flutter app

The final setup step is to add the relevant Firebase packages to my Flutter project. In the terminal, make sure I am in the root of the Flutter project at `flutter-codelabs/firebase-emulator-suite/start`. Then, run the three following commands:

```
PS C:\Users\kriti\flutter-codelabs\firebase-emulator-suite\start> flutter pub add cloud_firestore
Resolving dependencies...
+ cloud_firestore 4.13.5
+ cloud_firestore_platform_interface 6.0.9
+ cloud_firestore_web 3.8.9
  intl 0.18.1 (0.19.0 available)
  material_color_utilities 0.5.0 (0.8.0 available)
  meta 1.10.0 (1.11.0 available)
  path 1.8.3 (1.9.0 available)
  web 0.3.0 (0.4.0 available)
Changed 3 dependencies!
5 packages have newer versions incompatible with dependency constraints.
Try `flutter pub outdated` for more information.
```

```
PS C:\Users\kriti\flutter-codelabs\firebase-emulator-suite\start> flutter pub add firebase_core
"firebase_core" is already in "dependencies". Will try to update the constraint.
Resolving dependencies...
  intl 0.18.1 (0.19.0 available)
  material_color_utilities 0.5.0 (0.8.0 available)
  meta 1.10.0 (1.11.0 available)
  path 1.8.3 (1.9.0 available)
  web 0.3.0 (0.4.0 available)
Got dependencies!
5 packages have newer versions incompatible with dependency constraints.
Try `flutter pub outdated` for more information.
```

Enabling Firebase emulators

First, add the Firebase initialization code and emulator setup code to the main function in main.dart.

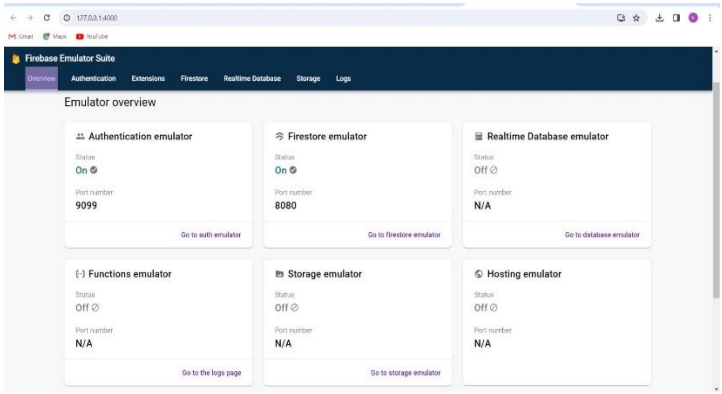
```dart
import 'package:firebase_core/firebase_core.dart';
import 'package:flutter/foundation.dart';
import 'package:flutter/gestures.dart';
import 'package:flutter/material.dart';
import 'package:go_router/go_router.dart';

import 'app_state.dart';
import 'firebase_options.dart';
import 'logged_in_view.dart';
import 'logged_out_view.dart';


Run | Debug | Profile
void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp(
    options: DefaultFirebaseOptions.currentPlatform,
  );

  if (kDebugMode) {
    try {
      FirebaseFirestore.instance.useFirestoreEmulator('localhost', 8080);
      await FirebaseAuth.instance.useAuthEmulator('localhost', 9099);
    } catch (e) {
      // ignore: avoid_print
      print(e);
    }
  }

  runApp(MyApp());
}
```

Start up the emulators

- I then start the emulators before you start the Flutter app. First, I start up the emulators by running this in the terminal

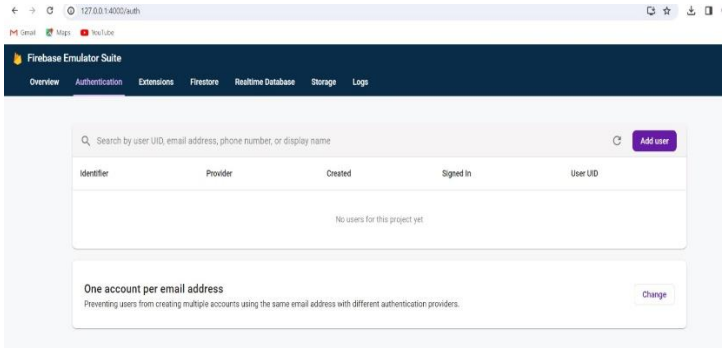
```
i  auth: Stopping Authentication Emulator
i  eventarc: Stopping Eventarc Emulator
i  hub: Stopping emulator hub

C:\Users\kriti\flutter-codelabs\firebase-emulator-suite\start>cd..

C:\Users\kriti\flutter-codelabs\firebase-emulator-suite>cd complete

C:\Users\kriti\flutter-codelabs\firebase-emulator-suite\complete>firebase emulators:start
i  emulators: Starting emulators: auth, firestore
i  firestore: Firestore Emulator logging to firestore-debug.log
+  firestore: Firestore Emulator UI websocket is running on 9150.
i  ui: Emulator UI logging to ui-debug.log

  ✓  All emulators ready! It is now safe to connect your app.
  i  View Emulator UI at http://127.0.0.1:4000/

| Emulator       | Host:Port      | View in Emulator UI |
|---|---|---|
| Authentication | 127.0.0.1:9099 | http://127.0.0.1:4000/auth |
| Firestore      | 127.0.0.1:8080 | http://127.0.0.1:4000/firestore |

  Emulator Hub running at 127.0.0.1:4400
  Other reserved ports: 4500, 9150
```

This output tells you which emulators are running, and where you can go to see the emulators. First, check out the emulator UI at localhost:4000.



The Firebase Auth emulator

The first emulator I'll use is the Authentication emulator. Start with the Auth emulator by clicking "Go to emulator" on the Authentication card in the UI, and I see a page that looks like this:



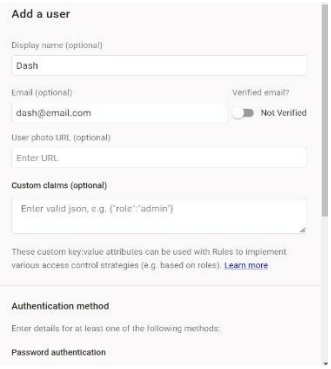
Add a user

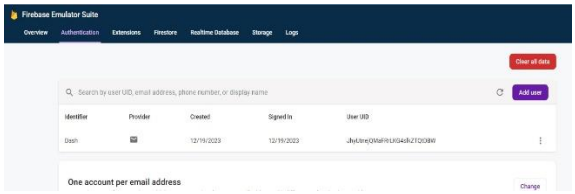
Click the "Add user" button, and fill out the form with this information:

Display name: Dash

Email: dash@email.com

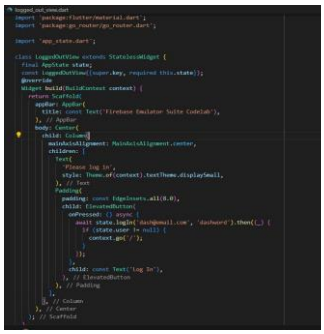
Password: dashword





[logged_out_view.dart]

The only code in the LoggedOutView widget that has to be updated is in the callback that's triggered when a user presses the login button. Update the code to look like this:



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

import 'app_state.dart';

class LoggedOutView extends StatelessWidget {
  final AppState state;
  const LoggedOutView({super.key, required this.state});
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Firebase Emulator Suite CodeLab'),
      ), // AppBar
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            Text(
              'Please log in',
              style: Theme.of(context).textTheme.displaySmall,
            ), // Text
            Padding(
              padding: const EdgeInsets.all(8.0),
              child: ElevatedButton(
                onPressed: () async {
                  await state.login('dash@email.com', 'dashword').then((_) {
                    if (state.user != null) {
                      context.go('/');
                    }
                  });
                },
                child: const Text('Log In'),
              ), // ElevatedButton
            ), // Padding
          ],
        ), // Column
      ), // Center
    ); // Scaffold
```

app_state.dart

Two portions of the code in AppState need to be updated. First, give the class member AppState.user the type User from the firebase_auth package, rather than the type Object.

Second, fill in the AppState.login method as shown below:

```
app_state.dart > AppState
import 'dart:async';

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

import 'entry.dart';

class AppState {
  AppState() {
    _entriesStreamController = StreamController.broadcast(onListen: () {
      _entriesStreamController.add([
        Entry(
          date: '10/09/2022',
          text: lorem,
          title: '[Example] My Journal Entry',
        ) // Entry
      ]);
    }); // StreamController.broadcast
  }

  User? user;
  Stream<List<Entry>> get entries => _entriesStreamController.stream;
  late final StreamController<List<Entry>> _entriesStreamController;

  Future<void> logIn(String email, String password) async {
    final credential = await FirebaseAuth.instance
        .signInWithEmailAndPassword(email: email, password: password);
    if (credential.user != null) {
      user = credential.user!;
      _listenForEntries();
    } else {
      print('No user!');
    }
  }
```

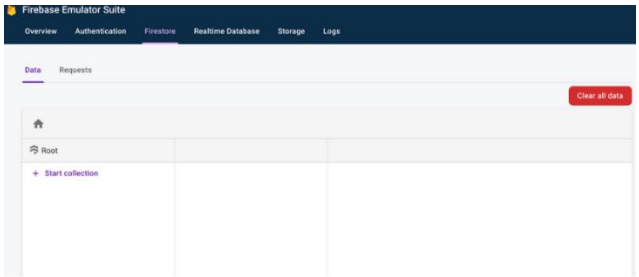
logged_in_view.dart

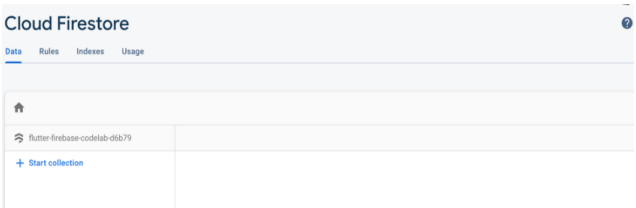
Change the first line in the LoggedInView.build method:

```
class LoggedInView extends StatelessWidget {
  final AppState state;
  LoggedInView({super.key, required this.state});

  final PageController _controller = PageController(initialPage: 1);

  @overview
  Widget build(BuildContext context) {
    final name = state.user!.displayName ?? 'No Name';

    return Scaffold(
      body: Column(
        children: [
          Center(
```

Read and Write data to Firestore emulator

First,I check out the Firestore emulator. On the Emulator UI homepage (`localhost:4000`), click "Go to emulator" on the Firestore card. It should look like this:

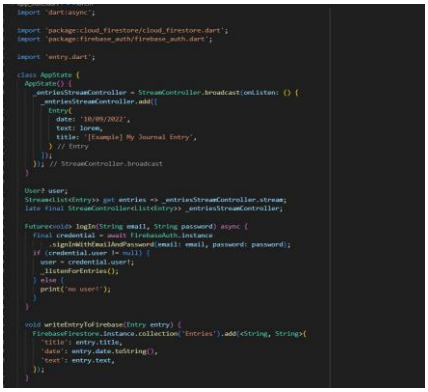




Write to Firestore

Before discussing the 'Requests' tab in the emulator, first make a request. This requires code updates. Start by wiring up the form in the app to write a new journal `Entry` to Firestore.

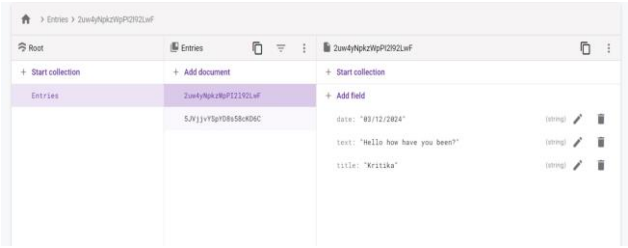
The following change should me made to `AppState.writeEntryToFirebase`.



```
import 'dart:async';

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

import 'entry.dart';

class AppState {
  AppState() {
    _entriesStreamController = StreamController.broadcast(onListen: () {
      _entriesStreamController.add([
        Entry(
          date: '10/09/2022',
          text: lorem,
          title: '[Example] My Journal Entry',
        ) // Entry
      ]);
    }); // StreamController.broadcast
  }

  User? user;
  Stream<List<Entry>> get entries => _entriesStreamController.stream;
  late final StreamController<List<Entry>> _entriesStreamController;

  Future<void> logIn(String email, String password) async {
    final credential = await FirebaseAuth.instance
        .signInWithEmailAndPassword(email: email, password: password);
    if (credential.user != null) {
      user = credential.user;
      _listenForEntries();
    } else {
      print('no user');
    }
  }

  void writeEntryToFirebase(Entry entry) {
    FirebaseFirestore.instance.collection('Entries').add(<String, String>{
      'title': entry.title,
      'date': entry.date.toString(),
      'text': entry.text,
      });
  }
```

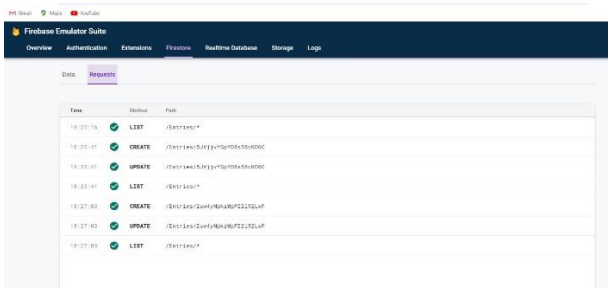
The request tab in the firebase emulator

In the UI, navigate to the Firestore emulator, and look at the "Data" tab. I can see that there's now a Collection at the root of my database called "Entries". That should have a document which contains the same information I entered into the form.

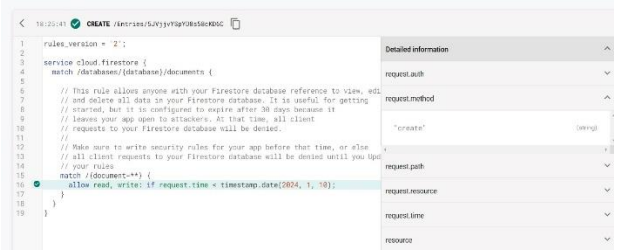


Firebase emulator requests

This is what I get as output



I click on the CREATE list item that corresponds to my request to create a new journal entry.



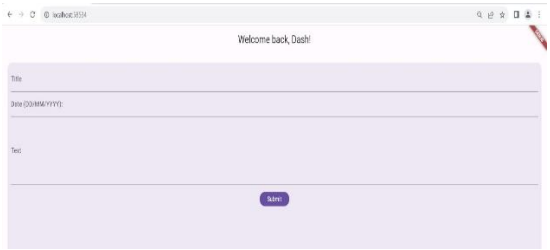
This view shows exactly what line in my security rules this request passed (or failed, if that was the case). In a more robust app, Security Rules can grow and have multiple authorization checks.

Reading from Firestore

Update **the _listenForEntries** method to match the code below:

```
void _listenForEntries() {
  FirebaseFirestore.instance
      .collection('Entries')
      .snapshots()
      .listen((event) {
    final entries = event.docs.map((doc) {
      final data = doc.data();
      return Entry(
        date: data['date'] as String,
        text: data['text'] as String,
        title: data['title'] as String,
      );
    }).toList();

    _entriesStreamController.add(entries);
  });
}

void dispose() {
  _entriesStreamController.close();
}
}
```

Now the app should look like this



Export and import data into emulator

First, I export the emulator data I already have. While the emulators are still running, I open a new terminal window, and enter the following command:

```
C:\Users\kriti>cd flutter-codelabs\firebase-emulator-suite\start

C:\Users\kriti\flutter-codelabs\firebase-emulator-suite\start>firebase emulators:export ./emulators_data
i  Found running emulator hub for project fir-flutter-codelab-41f64 at http://127.0.0.1:4400
i  Creating export directory C:\Users\kriti\flutter-codelabs\firebase-emulator-suite\start\emulators_data
i  Exporting data to: C:\Users\kriti\flutter-codelabs\firebase-emulator-suite\start\emulators_data
+  Export complete
```

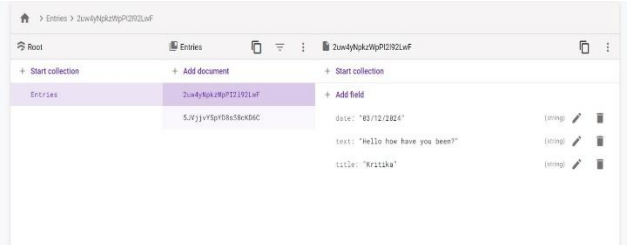
And if I switch to the terminal window where the emulators are running, I see this:

```
i  emulators: Received export request. Exporting data to C:\Users\kriti\flutter-codelabs\firebase-emulator-suite\start\e
mulators_data.
+  emulators: Export complete.
```

My data will now be saved and reloaded each time I work with the emulators for this project.

4.3 Final Output

Now run the app. It should look like this.The data is still present!



5 Task 5

5.1 Brief Overview

I am looking to implement CRUD operations using Flutter and Firebase Firestore for efficient data management in my mobile app. By integrating Flutter with Firestore, I aim to create, read, update, and delete data seamlessly, providing users with a smooth and dynamic experience.

Task-5

Implement the CRUD operations using Flutter and Firebase-FireStore cloud environment
Use the base code given during lecture
Provide a documentation with a short description and screenshots for all the steps of the lab and the Dart files.

Adding dependencies

I first start by adding some dependencies to my pubsec.yaml:

```
# The following adds the Cupertino Icons font to your application.
# Use with the CupertinoIcons class for iOS style icons.
cupertino_icons: ^1.0.2
cloud_firestore: ^4.13.5 # Cloud Firestore Plugin Link https://pub.dev/packages/cloud_firestore
firebase_core: ^2.24.2 # Firebase Core for Flutter Link https://pub.dev/packages/firebase_core
sqflite:^2.2.4+1

# After adding your reference dependencies run the following command:
```

Adding code in main.dart

I start by importing the following

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

Run | Debug | Profile
```

Initialising firebase

```
Run | Debug | Profile
void main() async {
  await configure(); // Initialize Firebase
  runApp(const MyApp());
}
```

This function initializes Firebase using Firebase.initializeApp. It ensures that Flutter is initialized and catches any initialization errors.

HomePageState

The HomePage widget is a StatefulWidget with an associated mutable state class (_HomePageState). The state class contains controllers for managing text input and a reference to the Firestore collection where notes are stored. This structure allows the HomePage widget to maintain dynamic state and update its UI accordingly.

```
class HomePage extends StatefulWidget {
  const HomePage({Key? key}) : super(key: key);

  @override
  State<HomePage> createState() => _HomePageState();
}

class _HomePageState extends State<HomePage> {
  // text fields' controllers
  final TextEditingController _titleController = TextEditingController();
  final TextEditingController _descriptionController = TextEditingController();

  final CollectionReference _notes =
      FirebaseFirestore.instance.collection('notes');
```

Logic in updating note creation

This function efficiently handles the logic to determine whether the user is creating a new note or updating an existing one. It sets the appropriate action ('create' or 'update') and pre-fills the text fields with existing data when updating. The pre-filling step ensures a seamless user experience when editing notes.

```
// If documentSnapshot != null then update an existing note
Future<void> _createOrUpdate([DocumentSnapshot? documentSnapshot]) async {
  // Determine whether it's a create or update action
  String action = 'create';
  if (documentSnapshot != null) {
    action = 'update';
    // Pre-fill text fields with existing data for update
    _titleController.text = documentSnapshot['title'];
    _descriptionController.text = documentSnapshot['description'].toString();
  }
```

Text Field Widgets

In summary, these TextField widgets are part of a form where users can input the title and description of a note. They are styled with specific decorations, including placeholder text, labels, and border properties. The SizedBox is used for spacing between the two text fields

```
  // Text fields for title and description
  TextField(
    controller: _titleController,
    decoration: const InputDecoration(
      hintText: "Enter the title",
      labelText: 'Title',
      border: OutlineInputBorder(
          borderSide: BorderSide(
            color: Colors.white,
            width: 0.75,
          ), // BorderSide
          borderRadius: BorderRadius.all(
            Radius.circular(10.0),
          )), // BorderRadius.all // OutlineInputBorder
    ), // InputDecoration
  ), // TextField
  const SizedBox(
    height: 20,
  ), // SizedBox
  TextField(
    controller: _descriptionController,
    decoration: const InputDecoration(
      hintText: "Enter the description",
      labelText: 'Description',
      border: OutlineInputBorder(
          borderSide: BorderSide(
            color: Colors.white,
            width: 0.75,
          ), // BorderSide
          borderRadius: BorderRadius.all(
            Radius.circular(10.0),
          )), // BorderRadius.all // OutlineInputBorder
    ), // InputDecoration
  ), // TextField
  const SizedBox(
    height: 20,
```

Delete_note

The _deletenote function in the Flutter code displays a confirmation dialog when a user attempts to delete a note. The dialog includes a title, a confirmation message, and two buttons ('Yes' and 'No'). The 'Yes' button, styled in red, triggers the deletion of the note from Firestore. Upon successful deletion, a snackbar appears, notifying the user of the action. The 'No' button dismisses the dialog. This function enhances user experience by ensuring users confirm their intention before deleting a note.

```
// Function to delete a note
Future<void> _deletenote(String noteId) async {
  showDialog(
    context: context,
    builder: (context) {
      return AlertDialog(
        title: const Row(children: [Text('Delete permanently!')]),
        content: const SingleChildScrollView(
          child: ListBody(
            children: <Widget>[
              Text('Are you sure, you want to delete this note?'),
            ], // <Widget>[]
          ), // ListBody
        ), // SingleChildScrollView
        actions: [
          ElevatedButton(
            style: ButtonStyle(
              backgroundColor: MaterialStateProperty.all(Colors.red)), // ButtonStyle
            onPressed: () async {
              await _notes.doc(noteId).delete();
              // Show a snackbar
              ScaffoldMessenger.of(context).showSnackBar(const SnackBar(
                content: Text('You have successfully deleted a note.')), // SnackBar
              Navigator.of(context).pop();
            },
            child: const Text('Yes'),
          ), // ElevatedButton
          ElevatedButton(
            onPressed: () => Navigator.pop(context),
            child: const Text('No'),
          ), // ElevatedButton
        ],
      ); // AlertDialog
    });
}
```

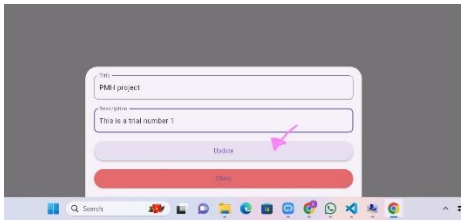
StreamBuilder

In summary, this Flutter code creates a real-time user interface for managing notes. It dynamically updates the UI as changes occur in the Firestore collection. The UI includes a scrollable list of notes with options to edit, delete, and add new notes. If there is no data, a loading indicator is shown.

5.2 Final Output

Let’s say I wish to create a note

For example, I wish to create a upload the previous note ,I click on create without forgetting to insert the title and description



Now it says successfully updated a note

The deleted function also works perfectly. As you can see the small bin, this is the enabled delete option

Now when I click on the bin, here is the message I receive