

**MAD PWA Lab**  
**EXPERIMENT NO.6**

**Name : Pranathi Narsupalli**  
**Div : D15B                      Roll no. : 45**

**AIM : To connect Flutter app UI with Firebase database**

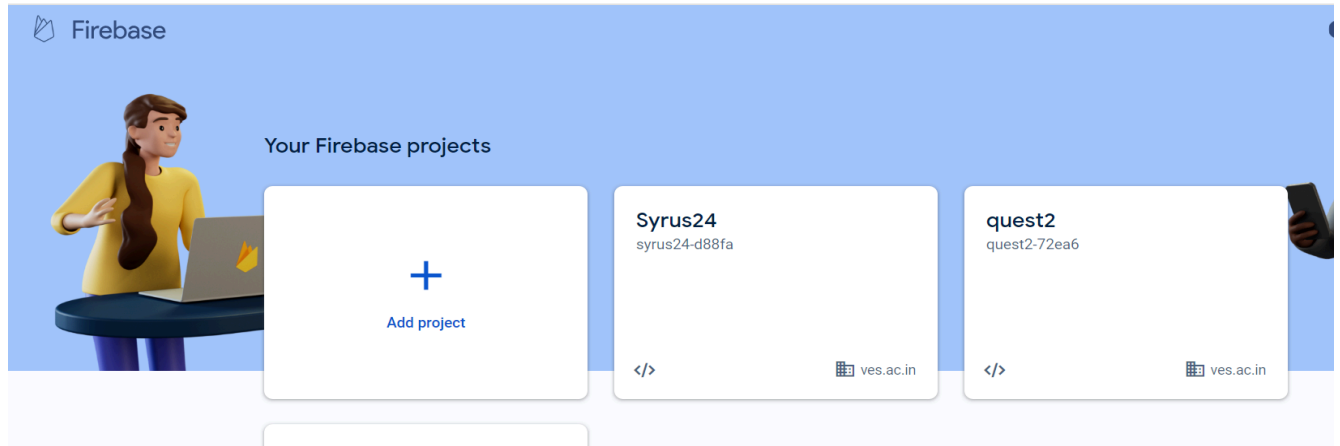
**THEORY :**

To integrate Flutter with Firebase, begin by setting up a Firebase project, linking your Flutter app, and configuring dependencies in the `pubspec.yaml` file. Initialize Firebase in your app's entry point and consider adding the optional `firebase\_auth` package for authentication. For database operations, utilize the `cloud\_firestore` package to interact with Firestore, performing CRUD operations and enabling real-time data updates. If your app involves file storage, integrate Firebase Storage using the `firebase\_storage` package. Implement robust error-handling mechanisms for asynchronous Firebase operations. Thoroughly test and debug your app, ensuring seamless functionality on both iOS and Android platforms. Once satisfied, deploy your Flutter app, now connected to Firebase, to make the most of the platform's features for authentication, real-time databases, and storage.

Connecting a Flutter app UI with a Firebase database involves several steps. Firebase is a mobile and web application development platform that provides various services, including a real-time NoSQL database.

**1. Create a Firebase Project:**

- Go to the [Firebase Console](<https://console.firebase.google.com/>).
- Click on "Add Project" and follow the setup instructions.



✕ Create a project (Step 1 of 3)

---

## Let's start with a name for your project<sup>?</sup>

Project name

**read-note**


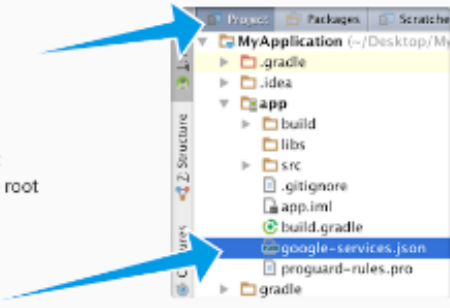
---

## 2. Add a Flutter App to Firebase Project:

- In the Firebase Console, select your project.
- Click on "Add app" and choose the Flutter icon.
- Follow the setup instructions to register your app.



## × Add Firebase to your Android app

- ✓ Register app  
Android package name: io.paulhalliday.myapplication, App nickname: Android App
- 2 Download config file  
Instructions for Android Studio below | [C++](#)  
[Download google-services.json](#)  
Switch to the **Project** view in Android Studio to see your project root directory.  
Move the google-services.json file you just downloaded into your Android app module root directory.  


### 3. Configure Firebase in Flutter:

## × Add Firebase to your Android app

- ✓ Register app  
Android package name: io.paulhalliday.myapplication, App nickname: Android App
- ✓ Download config file
- ✓ Add Firebase SDK
- 4 Run your app to verify installation

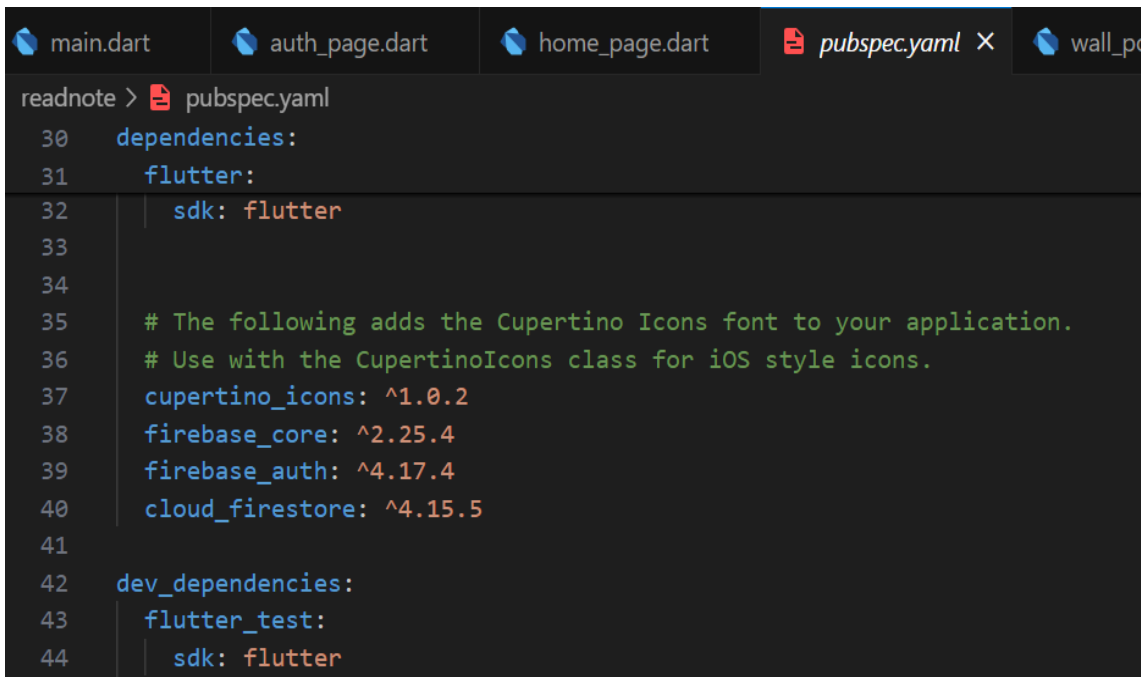
✓ Congratulations, you've successfully added Firebase to your app!

Previous [Continue to console](#)

- Add the necessary dependencies to your pubspec.yaml file:

```
yaml
dependencies:
  flutter:
    sdk: flutter
  firebase_core: ^latest_version
  firebase_database: ^latest_version
```

- Run `flutter pub get` to install the dependencies.

A screenshot of an IDE window showing the pubspec.yaml file. The window has tabs for main.dart, auth\_page.dart, home\_page.dart, pubspec.yaml (active), and wall\_po. The code in pubspec.yaml is as follows:

```
readnote > pubspec.yaml
30 dependencies:
31   flutter:
32     sdk: flutter
33
34
35   # The following adds the Cupertino Icons font to your application.
36   # Use with the CupertinoIcons class for iOS style icons.
37   cupertino_icons: ^1.0.2
38   firebase_core: ^2.25.4
39   firebase_auth: ^4.17.4
40   cloud_firestore: ^4.15.5
41
42 dev_dependencies:
43   flutter_test:
44     sdk: flutter
```

#### 4. Initialize Firebase in Flutter:

- Import the Firebase packages in your `main.dart` file:

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

- Initialize Firebase in the `main` function:

```
dart
void main() async {
  WidgetsFlutterBinding.ensureInitialized();
```

```
    await Firebase.initializeApp();  
    runApp(MyApp());  
  }
```

## 5. Set Up Firebase Realtime Database:

- In the Firebase Console, go to "Database" and click on "Create Database."
- Choose "Start in test mode" and click "Next."
- Set up your database rules.

## 6. Create Firebase Database Reference:

- Import the necessary package in your Dart file:

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

- Create a reference to your Firebase database:

```
dart  
final databaseReference = FirebaseDatabase.instance.reference();
```

## 7. Read Data from Firebase:

- To read data from Firebase, you can use methods like `once()` or `onValue()` :

```
Dart  
DatabaseReference reference = FirebaseDatabase.instance.reference();  
  
// Read data once  
DataSnapshot snapshot = await reference.once();  
  
// Access the data  
print('Data: ${snapshot.value}');
```

## 8. Write Data to Firebase:

- To write data to Firebase, you can use methods like `set()` or `push()` :

```
Dart  
DatabaseReference reference = FirebaseDatabase.instance.reference();
```

```
// Set data
```

```
reference.child('users').set({'username': 'JohnDoe', 'email': 'john@example.com'});
```

## **9. Update or Delete Data:**

- Use the `update()` or `remove()` methods to update or delete data:

**Dart**

```
DatabaseReference reference = FirebaseDatabase.instance.reference();
```

```
// Update data
```

```
reference.child('users').child('userId').update({'username': 'NewUsername'});
```

```
// Delete data
```

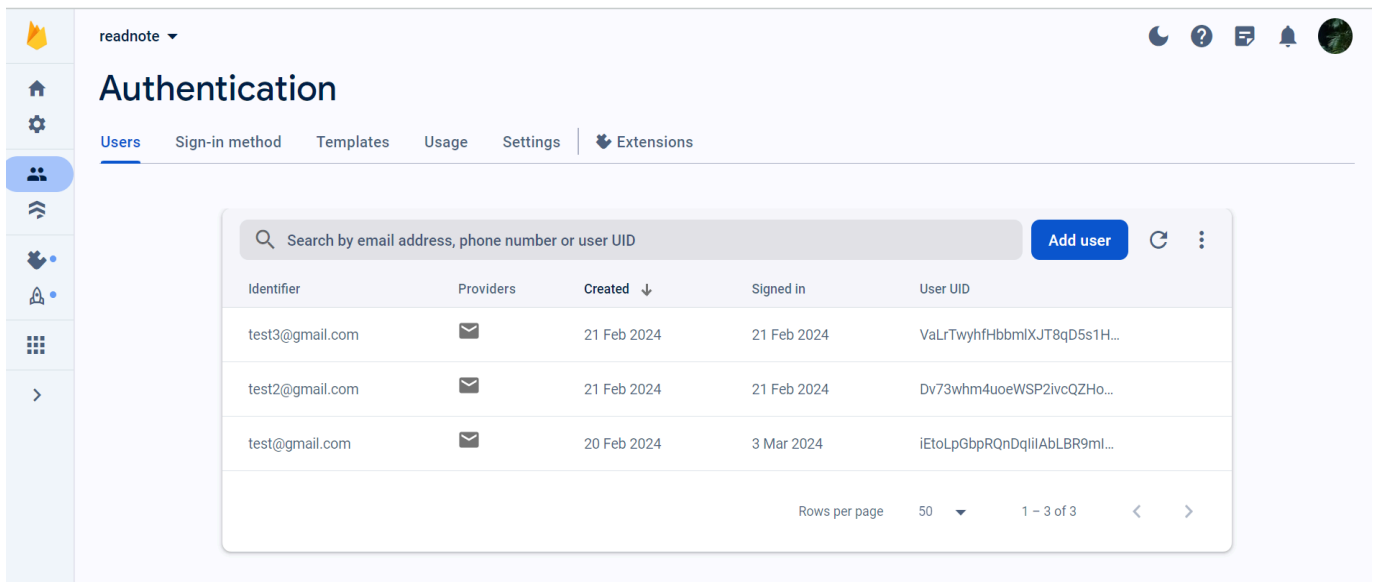
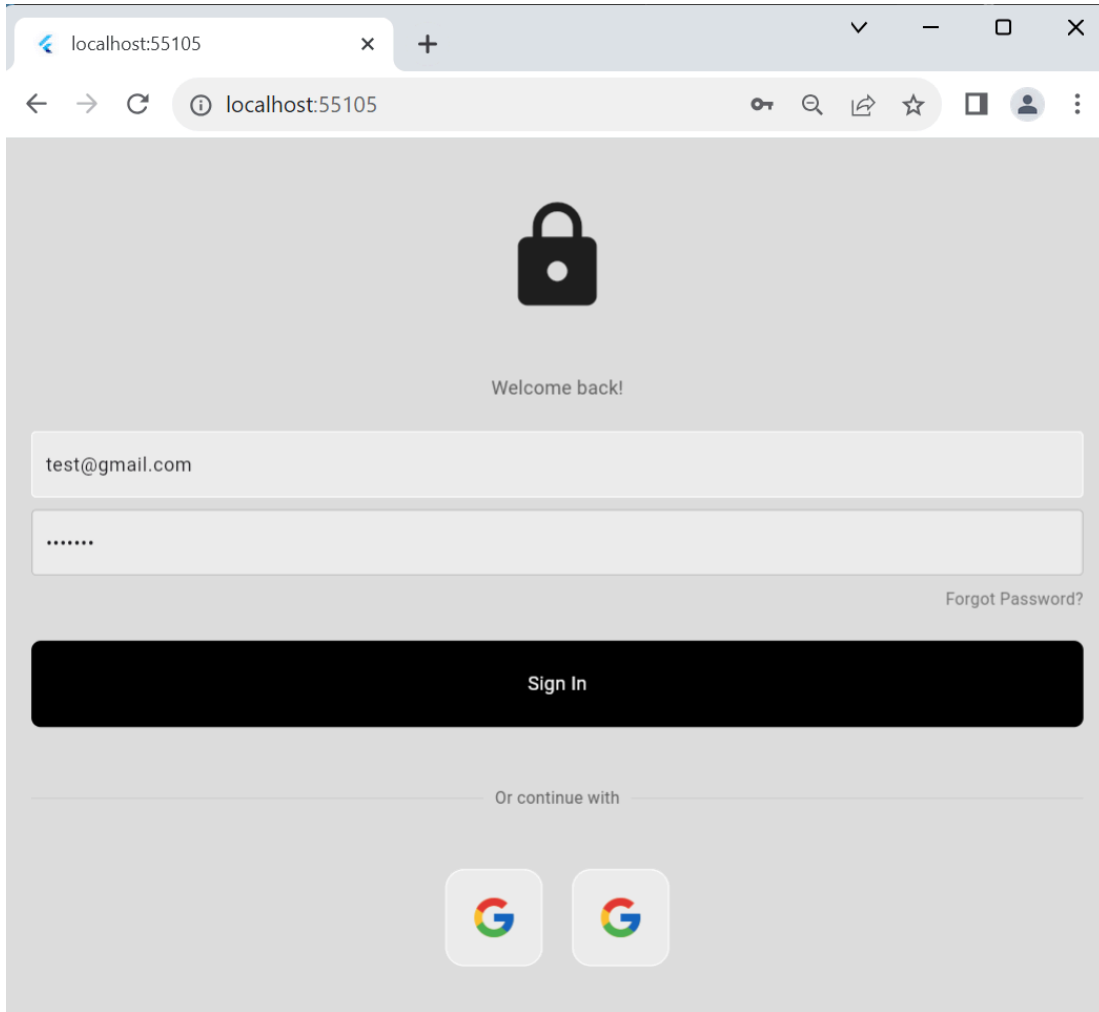
```
reference.child('users').child('userId').remove();
```

## **10. Handle Asynchronous Operations:**

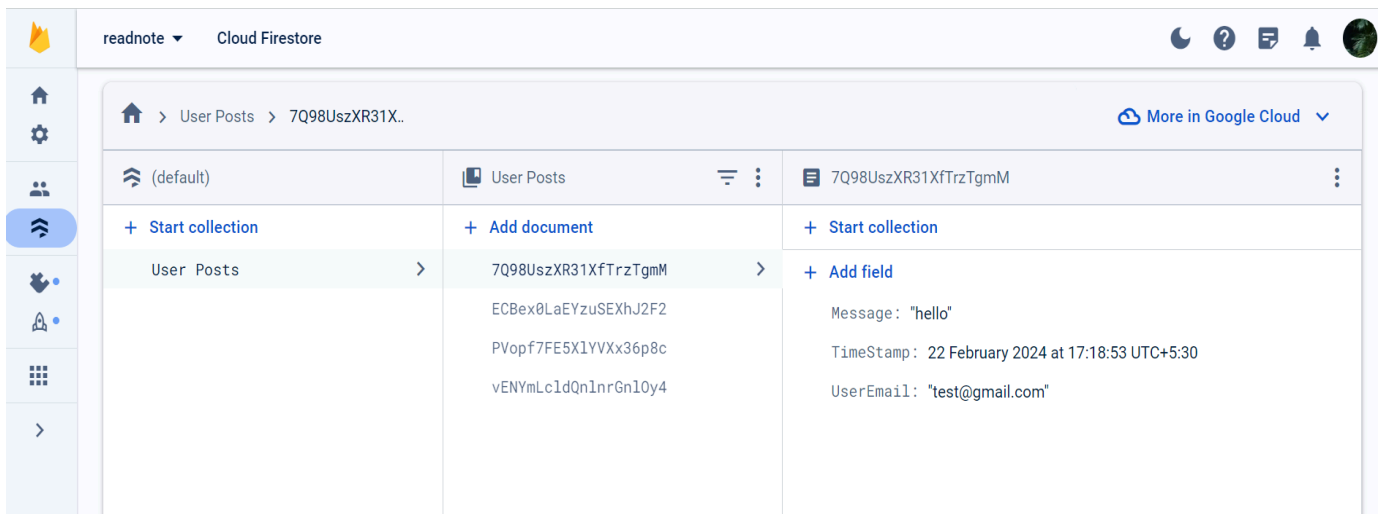
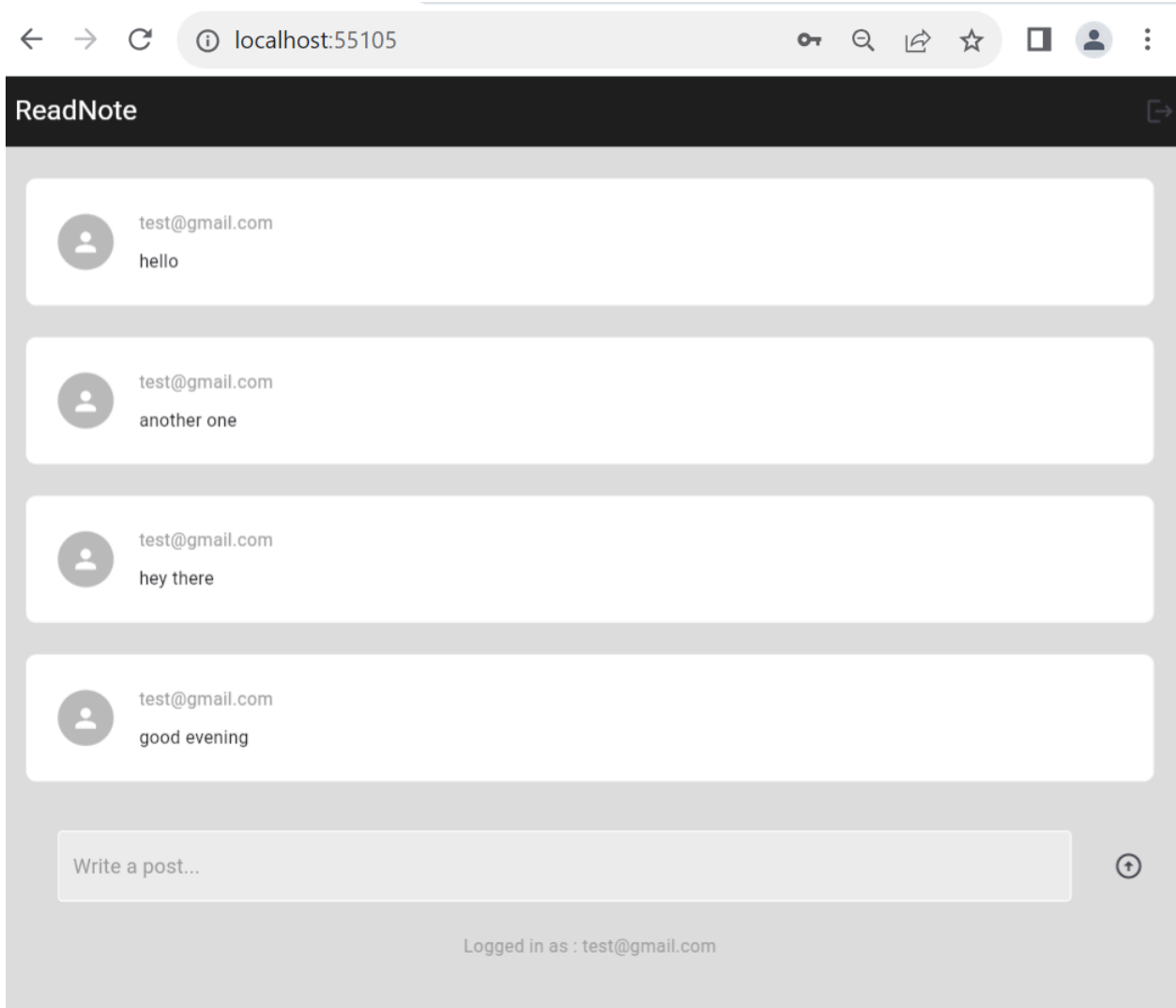
- Since Firebase operations are asynchronous, make sure to handle them using `async/await` or `.then()`.

## **11. Display Data in Flutter UI:**

- Use Flutter widgets to display the data retrieved from Firebase in your app's UI.







**CONCLUSION :** Hence, we have integrated Flutter with Firebase , which offers a robust solution for building powerful cross-platform applications. By combining Flutter's UI capabilities with Firebase's authentication, real-time database, and storage features, developers can create scalable, feature-rich apps. This streamlined process involves project setup, dependency configuration, initialization, and thorough testing, ultimately culminating in a seamless deployment across iOS and Android platforms.