Flutter Firebase Integration: Auth & Firestore Basics

Subtitle: Learn how to connect Flutter apps to Firebase for authentication and cloud data storage.

Website Name: haas.dev
Website Link: https://dev-roast-app.vercel.app

Introduction

Modern apps require **backend services** for authentication, database, and real-time updates. Firebase provides these services with minimal setup. This guide covers **Firebase Authentication and Firestore** for beginner Flutter apps.

Step 1: Setup Firebase Project

1. Go to Firebase Console
2. Create a new project
3. Add Android/iOS app and follow instructions
4. Download `google-services.json` (Android) or `GoogleService-Info.plist` (iOS)
5. Add Firebase SDK dependencies in `pubspec.yaml`:

```
dependencies:
  firebase_core: ^2.10.0
  firebase_auth: ^4.5.0
  cloud_firestore: ^5.6.0
```

Step 2: Initialize Firebase

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

Step 3: Firebase Authentication

Step 3: Firebase Authentication

Sign Up with Email & Password

```
final FirebaseAuth auth = FirebaseAuth.instance;

Future<void> signUp(String email, String password) async {
  try {
    UserCredential user = await auth.createUserWithEmailAndPassword(
      email: email,
      password: password,
    );
    print("User signed up: ${user.user?.uid}");
  } catch (e) {
    print("Error: $e");
  }
}
```

Sign In

```
Future<void> signIn(String email, String password) async {
  try {
    UserCredential user = await auth.signInWithEmailAndPassword(
      email: email,
      password: password,
    );
    print("User signed in: ${user.user?.uid}");
  } catch (e) {
    print("Error: $e");
  }
}
```

Exercise: Create a login screen with email & password fields.

Step 4: Firestore Setup

1. Go to Firebase Console → Firestore Database → Create database
2. Add rules (start in test mode)
3. Use the `cloud_firestore` package to store and retrieve data

Step 5: Adding Data to Firestore

```
FirebaseFirestore firestore = FirebaseFirestore.instance;
```

```
Future<void> addUser(String uid, String name, int age) async {
  await firestore.collection('users').doc(uid).set({
    'name': name,
    'age': age,
  });
}
```

Exercise: Save user info on signup.

Step 6: Reading Data from Firestore

```
StreamBuilder(
  stream: firestore.collection('users').snapshots(),
  builder: (context, snapshot) {
    if (!snapshot.hasData) return CircularProgressIndicator();
    final users = snapshot.data!.docs;
    return ListView.builder(
      itemCount: users.length,
      itemBuilder: (context, index) {
        return ListTile(
          title: Text(users[index]['name']),
          subtitle: Text("Age: ${users[index]['age']}"),
        );
      },
    );
  },
)
```

Exercise: Display all users in a list using StreamBuilder.

Step 7: Mini Project

Build a **Flutter Firebase Auth & Firestore App**:

- Sign Up / Sign In with Email & Password
- Save user info in Firestore
- Display users in a ListView
- Use real-time updates

Key Takeaways

- Firebase provides **auth and database** with minimal setup
- Firestore allows **real-time cloud storage**
- Combine Firebase Auth + Firestore for fully functional apps
- Async and streams are essential for handling live data
- Master this to build **real-world Flutter apps with backend support**

Skipping Firebase integration means your apps **cannot handle real users or persistent cloud data**, which is essential for production apps.

Visit **haas.dev** for more Flutter guides, tutorials, and complete project resources.

Website Name: haas.dev
Website Link: https://dev-roast-app.vercel.app
