Note-Taking App with Firebase

Project Overview

The Note-Taking App is a simple application that allows users to create, edit, delete, and view notes. It utilizes Firebase Authentication for user login and Firestore to store notes. The app is designed to provide a seamless experience for users to manage their notes securely.

Objective

To build a note-taking application that enables users to manage their notes effectively while ensuring secure access through Firebase Authentication.

Features

1. Firebase Authentication

- User Registration: Users can sign up using email and password.
- User Login: Users can log in to access their saved notes.
- User Session Management: Users can view their saved notes after logging in.

2. Note Management

- Create Notes: Users can add new notes.
- Edit Notes: Users can update existing notes.
- Delete Notes: Users can remove notes they no longer need.
- Data Storage: Notes are stored in Firebase Firestore.

3. UI Components

- Text: For displaying note titles, content, and labels.
- TextField: For creating and editing notes.
- Container & ElevatedButton: For action buttons (Add, Edit, Delete).
- Icon: For actions like saving or deleting notes.
- ListView: To display the list of saved notes.
- CircleAvatar (Optional): For user profile representation.

4. Navigation

- Login Screen: For user authentication (login/signup).
- Home Screen: Displays a list of the user's notes. Clicking on a note opens a screen to view and edit the note.
- Add Note Screen: A dedicated screen for users to add new notes.

5. State Management

- Provider: Used for managing user authentication state and note list state (adding, deleting, editing).

6. Firestore Integration

- Data Structure: Notes are stored in a Firestore collection named `notes`.
- Note Attributes:
    - `noteId`: Unique identifier for each note.
    - `title`: String representing the note title.
    - `content`: String representing the note content.
    - `createdAt`: Timestamp indicating when the note was created.
- Data Fetching: Notes are fetched from Firestore upon user login.

7. MVVM Folder Structure

- Model: Contains the Note model (Note class).
- View: UI screens (login screen, home screen, add/edit note screen).
- ViewModel: Logic for handling Firebase operations (authentication, fetching, adding, editing, deleting notes).

Code Snippets

1. Model: Note Class

The `Note` class represents the structure of a note in the application.

dart

Verify

Run

Copy code

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

class Note {
  final String noteId;
  final String title;
  final String content;
  final DateTime createdAt;

  Note({
    required this.noteId,
    required this.title,
    required this.content,
    required this.createdAt,
  });

  // Convert Firestore data to a Note object
  factory Note.fromMap(Map<String, dynamic> data) {
    return Note(
      noteId: data['noteId'] ?? '',
      title: data['title'] ?? '',
      content: data['content'] ?? '',
      createdAt: (data['createdAt'] as Timestamp).toDate(),
    );
  }

  // Convert a Note object to a map for Firestore
  Map<String, dynamic> toMap() {
    return {
      'noteId': noteId,
      'title': title,
      'content': content,
      'createdAt': createdAt,
    };
  }

}
```

2. View: Login Screen

The `LoginScreen` allows users to log in using their email and password.

dart

Verify

Run

Copy code

```dart
class LoginScreen extends StatefulWidget {
  @override
  _LoginScreenState createState() => _LoginScreenState();
}

class _LoginScreenState extends State<LoginScreen> {
  final TextEditingController emailController = TextEditingController();
  final TextEditingController passwordController = TextEditingController();
  String errorMessage = '';

   Future<void> _login() async {
     try {
       await FirebaseAuth.instance.signInWithEmailAndPassword(
         email: emailController.text.trim(),
         password: passwordController.text.trim(),
       );
       Navigator.pushReplacement(
         context,
         MaterialPageRoute(builder: (context) => NotesScreen()),
       );
     } catch (e) {
       setState(() {
         errorMessage = "Login Error: ${e.toString()}";
       });
     }
   }

   @override
   Widget build(BuildContext context) {
     return Scaffold(
       body: Center(
         child: Padding(
           padding: const EdgeInsets.all(20.0),
           child: Column(
             mainAxisAlignment: MainAxisAlignment.center,
             children: [
               TextField(controller: emailController, decoration: InputDecoration(labelText: "Email")),
               TextField(controller: passwordController, obscureText: true, decoration: InputDecoration(labelText: "Password")),
               SizedBox(height: 20),
               ElevatedButton(onPressed: _login, child: Text("Login")),
               SizedBox(height: 10),
               errorMessage.isNotEmpty ? Text(errorMessage, style: TextStyle(color: Colors.red)) : Container(),
             ],
           ),
         ),
       ),
     );
   }
```

```
}
```

3. ViewModel: NoteViewModel

The `NoteViewModel` handles the logic for fetching, adding, updating, and deleting notes.

dart

Verify

Run

Copy code

```dart
class NoteViewModel {
  final FirebaseFirestore _firestore = FirebaseFirestore.instance;
  final FirebaseAuth _auth = FirebaseAuth.instance;

  // Fetch Notes
  Stream<List<Note>> fetchNotes() {
    User? user = _auth.currentUser ;
    if (user == null) return Stream.value([]);

    return _firestore
        .collection('users')
        .doc(user.uid)
        .collection('notes')
        .orderBy('createdAt', descending: true)
        .snapshots()
        .map((snapshot) => snapshot.docs.map((doc) => Note.fromMap(doc.data())).toList());
  }

  // Add Note
  Future<void> addNote(String title, String content) async {
    User? user = _auth.currentUser ;
    if (user == null) return;

    DocumentReference docRef = _firestore.collection('users').doc(user.uid).collection('notes').doc();
    Note newNote = Note(
      noteId: docRef.id,
      title: title,
      content: content,
      createdAt: DateTime.now(),
    );

    await docRef.set(newNote.toMap());
  }

  // Update Note
  Future<void> updateNote(Note note) async {
    User? user = _auth.currentUser ;
    if (user == null) return;

    await _firestore
        .collection('users')
        .doc(user.uid)
        .collection('notes')
        .doc(note.noteId)
        .update(note.toMap());
  }

  // Delete Note
  Future<void> deleteNote(String noteId) async {
    User? user = _auth.currentUser ;
    if (user == null) return;

    await _firestore
        .collection('users')
        .doc(user.uid)
        .collection('notes')
        .doc(noteId)
        .delete();
  }

}
```

Steps to Complete

1. Setup Firebase

- Create a Firebase project.
- Enable Firebase Authentication and Firestore.
- Add Firebase to the Flutter project.

2. Create Authentication Screens

- Login Screen: Implement email/password login functionality.
- Signup Screen: Implement user registration functionality.
- Validation: Ensure proper validation for email and password fields.

3. Create Home Screen

- Fetch and display the list of notes from Firestore.
- Use ListView to display note titles.
- Provide options to edit or delete each note.

4. Create Add/Edit Note Screen

- Implement a form for adding or editing notes.
- Use TextField for title and content input.

5. Firestore Operations

- Implement CRUD functionality for notes:
    - Create: Add new notes to Firestore.
    - Read: Fetch notes from Firestore.
    - Update: Modify existing notes in Firestore.
    - Delete: Remove notes from Firestore.
- Store and retrieve notes under the authenticated user's UID.

6. State Management with Provider

- Manage user authentication state using Provider.
- Manage the state of the note list (add, delete, edit) using Provider.

7. UI Design

- Use Container and ElevatedButton for action buttons (Add/Edit/Delete).
- Use Icon for edit and delete actions.
- Use ListView to display notes in a scrollable format.
- Use TextField for adding/editing note content.

8. Test the App

- Test the app functionality by adding, editing, and deleting notes.
- Verify that only authenticated users can access their notes.

Conclusion

The Note-Taking App will provide users with a simple and effective way to manage their notes while ensuring secure access through Firebase Authentication. By following the outlined steps and utilizing the specified technologies, the app will be built to meet the project objectives successfully. # Note-Taking App with Firebase