CS310 – Mobile Application Development

Project Step 3: Firebase Backend & State Management

You already have a nice-looking UI from Step 2, now we're going to make it come alive by integrating it to Firebase so users can sign up and log in, and all the data (tasks, notes, chats, whatever your app does) gets saved and synced in real time.
On top of that, we'll add proper state management, so when something changes (like a new item getting added or updated), the whole app instantly reflects it without you having to refresh or write a ton of boilerplate code.
Basically, we're turning your beautiful prototype into a fully functional, data-driven app that feels fast and professional. Excited? Let's do this!

Tasks to Complete

- Implement user authentication using Firebase Authentication.
- Store and manage project-specific data using Cloud Firestore.
- Apply structured state management using (Provider, BLoC or Riverpod).
- Ensure the UI updates reactively based on authentication and Firestore data.
- Use SharedPreferences to save and restore at least one user preference (e.g., theme, username, last selected tab, etc.).

KEY REQUIREMENTS

1. Firebase Authentication
You must implement:

- **User Sign-Up** (email & password)
- **User Login**
- **User Logout**
- **Authentication-based access control**:
  - Logged-out users → Login/Register screen
  - Logged-in users → Main app screens
- **Error handling** with user-friendly messages (e.g., wrong password, network error).

2. Cloud Firestore Database
Your app must store its **main dynamic data in Firestore**.
Each Firestore document must include:

- A unique **id**
- App-specific fields (e.g., title, description, status, price, mood, etc.)
- A **createdBy** field (linked to user ID)
- A **createdAt** timestamp

You must implement **full CRUD operations**:

- **Create**: Add new data
- **Read**: Display Firestore data in the app
- **Update**: Modify existing data

- **Delete**: Remove data

Real-time updates are mandatory (using Firestore streams).
Security rules must restrict access to each user's own data or clearly defined public data.

3. State Management (Provider -preferably)
You must use **Provider with ChangeNotifier** to manage:
- **Authentication state** (logged in/out, current user)
- **Core app data state** (lists of items, loading states, errors)

Your app must:
- React automatically when:
  - A user logs in/out
  - Firestore data changes
- Avoid passing shared state deeply through constructors.

4. Local Persistence
Persist at least **one simple preference** using SharedPreferences:
- For example, Theme mode, last selected tab, onboarding status, etc.

DELIVERABLES

1. Source Code (GitHub)
Your GitHub repository must include:
- Authentication screens (login, signup, logout)
- Firestore integration with real-time updates
- Provider-based state management
- CRUD operations

If your project does not run, Step 3 grade may be 0.

2. Step 3 Demo Video (Max 5 Minutes)
Your demo must show:
- How Step 3 improves Step 2
- Authentication (signup, login, logout)
- Create / update / delete data
- Real-time UI update

Deadlines
- **Due: December 21, 2025 – 23:59**
- **Late: December 22, 2025 – 23:59** (**10% penalty**)
- **One submission per group** via SUCourse.

#	Rubrics	Max Points
1	Firebase is correctly set up and the app connects without errors	2
2	Users can sign up, log in, and log out properly (Firebase Authentication works)	3

3	Firestore collections are well organized and match the app's needs (good data model)	2
4	Model classes (Dart classes) correctly represent Firestore documents	2
5	All Firestore read/write operations go through a service/repository layer (no direct Firestore calls in UI/widgets)	2
6	Provider is set up correctly with MultiProvider and ChangeNotifier classes	2
7	Auth state (logged in / logged out) is managed with a provider and affects the whole app correctly	2
8	Data from Firestore is loaded with StreamBuilder/FutureBuilder, showing loading → success → error states properly	2
9	When data changes in Firestore, the UI updates automatically in real time	2
10	Navigation works correctly: logged-out users see login/signup, logged-in users see main screens and are protected	2
11	At least one user preference (theme, username, etc.) is saved and restored using SharedPreferences	2
12	Basic Firestore Security Rules are written and deployed (prevent unauthenticated reads/writes)	2