



# Flutter CHAT APP Project

**Prepared for :**

Eng. Sara Eldesouky

**Prepared by :**

Mohamed Osama	2206163
Mohamed Ayman	2206142
Philobatier Ayman	2206161
Ammar Essam	2206211
Omar Elsherbiny	2206206

# Project Overview

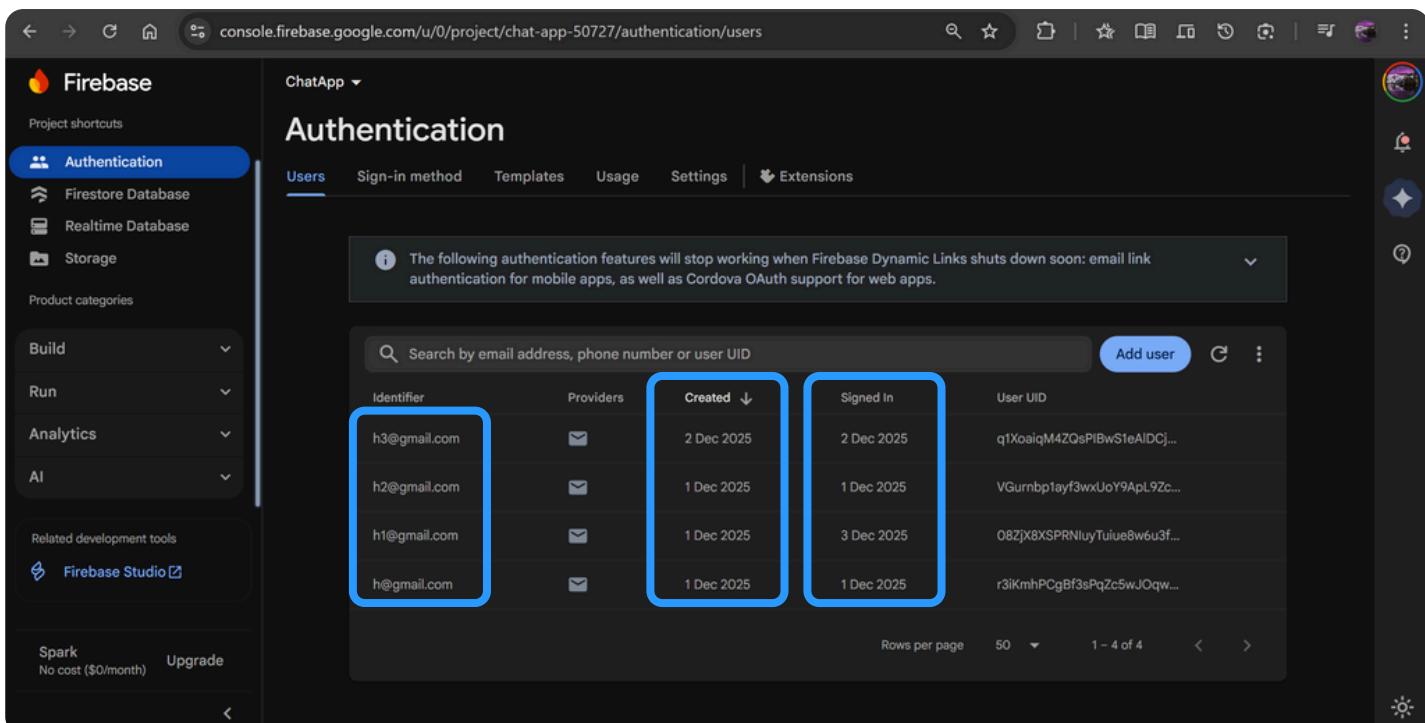
## Introduction :

- For our final project, we developed a fully functional real-time chat application using the Flutter framework. The objective was to create a mobile communication tool that mirrors modern messaging platforms like WhatsApp or Telegram. The application solves the problem of seamless, low-latency communication by leveraging the Firebase ecosystem.
- Key features include secure user authentication, persistent chat history, real-time presence detection (online/offline status), and typing indicators. The app features a modern AMOLED dark interface optimized for mobile devices.

## Implementation Details :

### • Firebase Authentication

- We implemented Firebase Authentication to manage user identity securely.
- Integration: We utilized the `firebase_auth` package to handle email and password registration and login.
- Flow: When a user signs up, a unique User ID (UID) is generated. This UID serves as the primary key across our database systems, ensuring data consistency. The app listens to the `authStateChanges()` stream to automatically persist login sessions (keeping the user logged in even after restarting the app) or redirect unauthenticated users to the Login screen.



The screenshot shows the Firebase Authentication section of the Firebase console. The left sidebar has 'Authentication' selected under 'Project shortcuts'. The main area is titled 'Authentication' with a sub-section 'Users'. A message at the top states: 'The following authentication features will stop working when Firebase Dynamic Links shuts down soon: email link authentication for mobile apps, as well as Cordova OAuth support for web apps.' Below this, there's a table with columns: Identifier, Providers, Created, Signed In, and User UID. Four rows of data are shown, each with an email address in the Identifier column and a timestamp in the Created and Signed In columns. The User UID column shows a long string of characters. The bottom of the table includes pagination controls for 'Rows per page' (set to 50), '1 – 4 of 4', and navigation arrows.

Identifier	Providers	Created	Signed In	User UID
h3@gmail.com	✉	2 Dec 2025	2 Dec 2025	q1XoaiqM4ZQsPBwS1eAlDCj...
h2@gmail.com	✉	1 Dec 2025	1 Dec 2025	VGurnbp1ayf3wxUoY9ApL9zc...
h1@gmail.com	✉	1 Dec 2025	3 Dec 2025	08ZjX8XSPRNluyTuiue8w6u3f...
h@gmail.com	✉	1 Dec 2025	1 Dec 2025	r3iKmhPCgBf3sPqZc5wJOqw...

- **Firebase Realtime Database**

- The Realtime Database (RTDB) was specifically employed to handle ephemeral (temporary) data that requires extremely low latency.
- Usage - Presence System: We used RTDB to track user status. When the app opens, a connection is established, setting a flag `status/{uid}/isOnline` to true. We utilized the `.info/connected` feature and `onDisconnect()` handler to automatically set this flag to false (and update the `lastSeen` timestamp) if the user loses internet or closes the app.
- Usage - Typing Indicators: Typing status is stored at `typing/{chatRoomId}/{uid}`. This data is frequent and temporary; it does not need to be permanently stored, making RTDB the ideal choice.

The screenshot shows the Firebase Realtime Database console for a project named "ChatApp". The left sidebar includes links for Project shortcuts, Authentication, Firestore Database, Realtime Database (which is selected), and Storage. The main area displays the Realtime Database interface with tabs for Data, Rules, Backups, Usage, and Extensions. A banner at the top right says "Need help with Realtime Database? Ask Gemini". Below the tabs, there's a warning about protecting resources from abuse and a "Configure App Check" button. The main view shows a hierarchical database structure under the URL `https://chat-app-50727-default-rtdb.europe-west1.firebaseio.database.app`. The structure includes a "presence" node with a "status" node containing three child nodes with random IDs. Below that is a "typing" node with three child nodes with random IDs. The entire "status" and "typing" node are highlighted with a blue box. At the bottom of the screen, it says "Database location: Belgium (europe-west1)".

- **Firebase Firestore Database**

- Cloud Firestore was implemented as the primary persistent storage solution.
- Usage - User Profiles: User details (Name, Email, UID) are stored in a users collection.
- Usage - Chat History: Messages are stored in a hierarchical structure: `chats/{chatRoomId}/messages/{messageId}`. Each message document contains the text, sender ID, receiver ID, timestamp, and read status.
- Integration: We used StreamBuilder in Flutter to listen to the Firestore collection. This allows the UI to update instantly whenever a new message is added to the database, without requiring a manual refresh.

The screenshot shows the Firebase Firestore Database interface. On the left, there's a sidebar with options like Project shortcuts, Authentication, Firestore Database (selected), Realtime Database, Storage, Product categories, Build, Run, Analytics, AI, Related development tools, and Spark (No cost (\$0/month)). The main area is titled 'Database' and shows a 'chats' collection. Inside 'chats', there are three subcollections: 'chats' (highlighted with a blue box), 'messages' (also highlighted with a blue box), and 'users'. Under 'chats', there are several documents, one of which is expanded to show fields: 'lastMessage' (value: ":"), 'lastMessageTime' (value: '1 December 2025 at 22:24:13 UTC+2'), and 'lastSenderId' (value: 'TbWhFU7QATd3rpUF3quxFDxCst62'). A 'participants' field is also visible. At the top, there's a banner about protecting resources from abuse and a 'Configure App Check' button.

## • Chat Application (Integration)

- The chat interface brings all components together:
- Auth: Determines who is sending the message (currentUser).
- Firestore: Loads and saves the message history in real-time.
- Realtime DB: Displays "Online" or "Typing..." under the user's name in the AppBar.
- Logic: When a message is opened, the app updates the read field in Firestore, which triggers a UI update to show blue ticks (Read Receipts).

home\_screen.dart (Offline & Online)

```

1 // Managing User Online Status
2 Future<void> _setUserStatus(bool isOnline) async {
3     // Using Realtime Database for low-latency updates
4     final DatabaseReference _database = FirebaseDatabase.instance.ref();
5
6     // Set status to online/offline with a timestamp
7     await _database.child('status/${currentUser.uid}').set({
8         'isOnline': isOnline,
9         'lastSeen': ServerValue.timestamp,
10    });
11
12     // Automatically handle disconnection (e.g., app crash or network loss)
13     if (isOnline) {
14         _database.child('status/${currentUser.uid}').onDisconnect().update({
15             'isOnline': false,
16             'lastSeen': ServerValue.timestamp,
17         });
18     }
19 }
```



## chat\_screen.dart (Sending &amp; Retrieving a message)

```
1 // Sending a Message to Firestore
2 Future<void> _sendMessage() async {
3   await FirebaseFirestore.instance
4     .collection('chats')
5     .doc(chatRoomId)
6     .collection('messages')
7     .add({
8       'text': _messageController.text.trim(),
9       'senderId': currentUser.uid,
10      'receiverId': widget.receiverId,
11      'timestamp': FieldValue.serverTimestamp(),
12      'read': false, // Initial read status
13    });
14 }
15
16 // Listening for New Messages in Real-time
17 StreamBuilder<QuerySnapshot>(
18   stream: FirebaseFirestore.instance
19     .collection('chats')
20     .doc(chatRoomId)
21     .collection('messages')
22     .orderBy('timestamp', descending: false) // Ordered chronologically
23     .snapshots(),
24   builder: (context, snapshot) {
25     if (!snapshot.hasData) return CircularProgressIndicator();
26
27     // UI rendering logic here...
28   },
29 )
```



```
1 Future<void> _submitForm() async {
2   if (_formKey.currentState!.validate()) return;
3
4   try {
5     UserCredential userCredential;
6     if (_isLogin) {
7       // Login existing user
8       userCredential = await FirebaseAuth.instance.signInWithEmailAndPassword(
9         email: _emailController.text.trim(),
10        password: _passwordController.text,
11      );
12    } else {
13      // Register new user
14      userCredential = await FirebaseAuth.instance.createUserWithEmailAndPassword(
15        email: _emailController.text.trim(),
16        password: _passwordController.text,
17      );
18
19      // Create user profile in Firestore
20      await
21        FirebaseFirestore.instance.collection('users').doc(userCredential.user!.uid).set({
22          'name': _nameController.text.trim(),
23          'email': _emailController.text.trim(),
24          'createdAt': FieldValue.serverTimestamp(),
25        });
26    }
27  } on FirebaseAuthException catch (e) {
28    // Handle errors (e.g., weak password, email already in use)
29    ScaffoldMessenger.of(context).showSnackBar(SnackBar(content: Text(e.message!)));
30  }
31 }
```

