ISSN: 2456-8503

International Journal for Development of Science and Technology (IJDST)

Real time chat messenger with state persistence

RESEARCH

Mayura[1], Rithika[1], Shanmathi[1], Senthil Prakash[1]*

Abstract

This work aims to design and implement a real-time messaging application that facilitates immediate interaction among users. The system uses modern web technologies to ensure messages are delivered without delay. It also includes state persistence, which allows chat history and user sessions to be saved even after refreshing or reconnecting. The application provides a user-friendly interface, efficient data handling, and reliable performance, making it suitable for modern communication needs.

Keywords: *Human-centered AI, artificial intelligence, explainable AI, automation.*

1. Introduction

In the present digital landscape, communication has become an essential component of everyday life, with rapid advancements improving both speed and accessibility. Traditional messaging approaches mainly rely on request-response mechanisms, which often lead to delays and inefficiencies in continuous communication. These limitations create a demand for more dynamic and responsive systems capable of supporting uninterrupted interaction. With the evolution of modern technologies, real-time communication has gained significant importance in web and mobile applications. Unlike conventional systems, real-time platforms enable instantaneous data exchange, allowing users to send and receive messages without noticeable latency. Technologies such as Web Socket facilitate persistent and bidirectional communication, improving overall system performance and user experience. Furthermore, scalability and data consistency remain critical challenges in existing systems, especially when handling a large number of concurrent users. To address these issues, advanced solutions integrating efficient synchronization and state management techniques are required. Therefore, this work focuses on developing a robust real-time messaging application that ensures reliable communication, improved responsiveness, and seamless user interaction [1].

2. Background and related work

2.1. Real-Time Communication

Real-time communication is a key feature in modern web applications that allows instant data exchange between users. Traditional systems were based on HTTP request-response models, which caused delays and required frequent page refreshes. With the introduction of Web Socket technology, continuous and bidirectional communication has become possible, improving speed and efficiency in messaging systems.

[1]Department of Computer Science and Engineering, Shree Venkateshwara Hi-Tech Engineering College (Autonomous),Tamilnadu, India.

[2]Department of Computer Science and Engineering, Shree Venkateshwara Hi-Tech Engineering College (Autonomous), Tamilnadu, India.

[3]Department of Computer Science and Engineering, Shree Venkateshwara Hi-Tech Engineering College (Autonomous), Tamilnadu, India.

[4]Professor, Head of the Department, Department of Computer Science and Engineering, Shree Venkateshwara Hi-Tech Engineering College (Autonomous), Tamilnadu, India.

** Corresponding Author:jtyesp14@gmail.com*

2.2. Existing Messaging Platform

Popular messaging applications such as WhatsApp and Telegram demonstrate the effectiveness of real-time communication. These platforms use scalable architectures and cloud-based storage to support instant messaging and large user bases while maintaining performance and reliability. In addition, these applications support functionalities such as media file exchange, alert mechanisms, and secure end-to-end data protection.

Figure1: Limitations of current messaging systems.

The (Figure 1) highlights key issues in traditional communication platforms, such as latency in message delivery, insufficient real-time synchronization, and challenges in scaling for multiple users. In addition, these systems often lack efficient state persistence mechanisms, resulting in the loss of chat history and user session data when the application is closed or refreshed. Most traditional systems are not designed to handle real-time synchronization, which causes inconsistency in message delivery across different devices Furthermore scalability is a major limitation, as many existing platforms struggle to support a large number of concurrent users efficiently [2].

2.3. Technical Approaches in Previous Work

Various technologies have been employed in the development of messaging applications to support efficient real-time communication. This (Figure 2) shows the Flutter, technologies such as many existing systems utilize protocols and frameworks that enable continuous data transmission between users without significant delays. Technologies such as Socket. IO and Web Socket are commonly adopted to establish persistent connections, allowing seamless bidirectional interaction. In terms of user interface development, frontend technologies including HTML, CSS, and JavaScript are widely used to design responsive and interactive applications. Additionally, modern frameworks and libraries enhance the overall user experience by improving performance and usability.

Figure 2: Technologies used in real-time applications

Firebase and Node.js used to build real-time chat applications. On the backend, cloud-based services like Firebase are frequently implemented to handle real-time databases, user authentication, and data storage. These platforms provide built-in features for synchronization and scalability, making them suitable for applications with a large number of users.

3. Proposed system architecture

The developed system represents a real-time messaging solution with built-in state persistence, utilizing Flutter for frontend development and Firebase for backend functionality. It enables rapid message exchange along with secure data storage and consistent synchronization between devices. The proposed architecture adopts a client–cloud structure, where the Flutter-based application connects with Firebase services such as Authentication, Cloud Fire store, and Firebase Cloud

Messaging (FCM) to facilitate live data exchange. By combining real-time synchronization with state persistence, the system ensures stable and consistent communication despite connectivity issues or device transitions.

Table 1: Key components of the proposed architecture

Component	Function	Technology used	Benefit
Flutter Client App	Provides chat UI and user interaction	Flutter(Dart)	Cross-platform support
Authentication module	Handles user login and signup	Firebase Authentication	Secure and easy login
Real-time database	Stores and syncs message s	Cloud fire store	Instant updates
State Management	Maintain s UI and chat state	Provider/ River pod/ set State	Smooth performance
Notification service	Sends message alerts	Firebase cloud Messaging	Real-time notification
Cloud storage	Stores media files	Firebase storage	Supports images/files

3.1. The User Interface and Frontend Layer Flutter Client

The frontend layer acts as the primary interface through which users interact with the application. Utilizing Flutter (Dart), the system delivers efficient performance using a unified codebase that supports both IOS and Android platforms. This component manages the display of chat interfaces, contact information, and multimedia content previews.

Key Technical Role: State Management (Core Controller) Frameworks such as Provider or River pod are utilized to manage application logic efficiently. This component monitors data updates from the backend and selectively triggers UI updates only when necessary. By avoiding full-screen refreshes for minor changes, it ensures a stable and seamless user experience with consistent performance.

3.2. The Backend and Data Management Layer Firebase

Rather than building a custom server from scratch, this architecture leverages Firebase to handle the heavy technical lifting. This is often referred to as Backend-as-a-Service (BaaS). Identity & Security (Authentication) this is the security gatekeeper. It handles the complexities of hashing passwords, verifying emails, and integrating third-party logins (like Google or Apple). It ensures that User A can never read the private messages of User B. The Live Database (Cloud Fire store) this is a NoSQL database that specializes in real-time synchronization [3]. Unlike a traditional website where you have to refresh to see new data, Firestore uses a listener system. When a message is written to the database, it is automatically pushed to all connected devices in milliseconds. Asset Management (Cloud Storage) While databases are well-suited for storing textual information, they are not ideal for handling large files. Therefore, cloud storage services are utilized to store bulky data such as high-resolution images, videos, and audio recordings. The application uploads these files to cloud storage and retains only the corresponding reference link (URL) within the database to ensure efficient data handling (Table 1).

3.3. The Communication and Engagement Layer FCM

A chat app is only useful if users know they have a message. Since mobile operating systems (IOS and Android) often kill apps running in the background to save battery, the Firebase Cloud Messaging (FCM) service is essential.

Key Technical Role: Push Notifications: When the backend detects a new message offline user, it sends a high-priority signal through Google's notification servers. This wakes up the device and displays an alert, ensuring the conversation remains continuous even when the app is closed.

3.4. Overall Architectural Benefits

- *Reduced Development Time:* By using pre-built modules for login and databases, developers can focus 100% on the user experience rather than server maintenance.
- *Auto-Scalability:* Whether you have 10 users or 10,000, the Firebase infrastructure automatically expands to handle the traffic without manual intervention.
- *Cross-Platform Consistency:* Because the logic (Dart) and the backend (Firebase) are unified, a user switching from an iPhone to an Android tablet will see the exact same messages and profile data instantly.

Table 2: Comparison with conventional systems

Feature	Proposed system (flutter and firebase)	Traditional systems
Message delivery	Real-time	Polling-based
Data storage	Persistent (cloud fire store)	Temporary
Synchronization	Instant	Delayed
Scalability	High	Limited
Offline Support	Available	Minimal
Notifications	Push based	Basic

Real-Time Connectivity vs. Polling: In traditional systems, the app has to ask the server every few seconds if there are new messages (Polling). This wastes battery and creates a lag [4]. The Proposed System uses Web Sockets and persistent connections. When a message is sent, the server pushes it to the recipient immediately. This results in Real-time delivery that feels like a natural conversation.

Cloud Persistence vs. Temporary Storage: Traditional chat systems (like older IRC or basic web chats) often relied on temporary memory or local sessions that could be lost if the server rebooted or the user cleared their cache. The Proposed System uses Cloud Firestore, which is a redundant, distributed database. Every message is saved permanently in the cloud the moment it is sent, ensuring that a user can log in from a new device and see their entire chat history perfectly preserved.

Instant Synchronization: Because the system is built on a Reactive framework, data stays in sync across multiple devices. If you read a message on your tablet, the notification on your phone disappears instantly. Conventional systems commonly experience synchronization delays, resulting in inconsistencies such as varying unread message counts and incomplete message updates across different devices.

High Scalability vs. Limited Growth: Traditional systems often require you to manually add more servers (Vertical Scaling) as your user base grows, which is expensive and prone to downtime. The Proposed System is Server less. Google's infrastructure handles the scaling automatically. Whether you have ten users or ten million, the architecture expands its resources behind the scenes without you needing to change a single line of code.

Robust Offline Support: One of the standout features of the Flutter/Firebase combo is its ability to handle spotty internet.

- *Traditional Systems:* Usually show an error or a Loading spinner if the connection drops.
- *Proposed System:* Uses local caching. Users can still type messages and browse old chats while in a tunnel or on a plane. Once the device reconnects to the internet, the app automatically syncs the pending message to the cloud.

Push-Based Notifications: Conventional applications typically depend on in-app notification mechanisms that function only when the application is active. In contrast, the proposed system employs Firebase Cloud Messaging (FCM) to deliver high-priority push notifications directly to the device's notification panel ensuring users receive alerts even when the application is not running (Table 2).

Table 3: Performance advantages

Parameter	Improvement Achieved
Latency	Very low (real-time updates)
Reliability	High message delivery
Scalability	Supports large user base
User experience	Smooth and responsive
Data Persistence	Messages stored permanently

Latency Near-Instant Communication: In older systems, a delay of even 2-3 seconds could make a conversation feel disjointed.

- The Achievement: By using Real-time Listeners in Firestore, the data is pushed to the device the moment it's written to the database.
- These results in Very Low latency, where messages appear on the recipient's screen almost simultaneously with the sender hitting Send, creating a true real-time conversational flow.

Reliability Guaranteed Delivery: Reliability in a chat app means a message should never simply disappear into the void.

The Achievement: The system utilizes ACID-compliant database transactions and retry logic. If a user's internet cuts out mid-send, the system tracks that pending state.

- *Result:* High Message Delivery ensures that once a connection is re-established, message is automatically retried and delivered, preventing data loss.

Scalability Built for Growth: A common failure for new apps is crashing when they suddenly get popular.

- *The Achievement:* Because the architecture is hosted on Google's global cloud infrastructure, it is not dependent on a single physical server for its operation.
- *Result:* The system can scale from 10 users to 100,000+ users without the development team needing to upgrade hardware or change the codebase, ensuring the app stays up during viral growth.

User Experience Fluidity and Speed: User experience (UX) isn't just about how the app looks, but how it feels.

- *The Achievement:* Flutter's rendering engine compiles to native machine code, allowing for smooth animations at 60 (or even 120) frames per second.
- *Result:* When combined with smart state management (like Riverpod), the UI remains Smooth and Responsive. There is no jank or freezing when scrolling through thousands of messages or switching between chat rooms.

Data Persistence: A Permanent History: Users expect to see their messages from three years ago just as easily as messages from three minutes ago.

The Achievement: Unlike session-based systems that might clear data to save space, this system uses Cloud Firestore for permanent storage.

- *Result:* Message is securely maintained across distributed data centers in different locations. As a result, users can quickly recover their complete chat history by signing into their account from another device, even in situations where their original device is lost (Table 3).

Table 4: System parameters

Parameter	Description
User ID	Unique user identifier
Message ID	Unique message identifier
Timestamp	Maintains message order
Chat Room ID	Identifies chat session

User ID (The Digital Identity):

- *Role:* This is a unique alphanumeric string (UUID) generated by Firebase Authentication.
- *Function:* It acts as the Primary Key for a user. Instead of identifying a person by their name (which can change or be duplicated), the system uses the User ID to link profile photos, and sent messages to one specific account.

Message ID (The Fingerprint):

- *Role:* Assigns a distinct identifier to each text or media message.
- *Function:* This mechanism prevents duplication or misidentification of messages. When a user chooses record in the database without impacting other messages.

Timestamp (The Chronology):

- *Role:* A high-precision record of exactly when a message was received by the server.
- *Function:* This is critical for Message Ordering. Because internet speeds vary, arrive at the server first. The Timestamp allows the Flutter app to sort the chat history correctly so the conversation makes logical sense.

Chat Room ID (The Container):

- *Role:* A unique ID that groups two or more User IDs together.
- *Function:* Think of this as the folder where messages live. Whether it is a private 1-on-1 DM or a large group chat, the Chat Room ID ensures that messages sent in Room A never accidentally appear in Room B.

Message Status (The Feedback Loop):

- *Role:* A dynamic field that updates as a message moves through the system (Sent ➔Delivered ➔ Seen).
- *Function:* This provides the Double Tick or Read Receipt functionality. When the recipient's app the database, which then triggers a real-time UI update on the sender's screen. Messages sent in Room A never accidentally appear in Room B.

Online Status (The Presence Engine):

- *Role:* A Boolean (True/False) or Last seen timestamp.
- *Function:* This indicates if a user is currently active in the app. In Firebase, this is often handled by a presence system that detects when a user's socket (Table 4).

4. Implementation methodology

The development of the proposed real-time messaging application is implemented using Flutter for the user interface and Firebase for backend functionalities. The system aims to deliver a fast and responsive communication experience by incorporating real-time data updates along with persistent storage mechanisms. Initially, the process begins with user authentication, where individuals can securely sign up and log in through Firebase Authentication. Each user is assigned a distinct identifier, which facilitates efficient management of user profiles and conversations within the application [5].

After successful authentication, users are directed to the chat interface designed using Flutter, which offers an intuitive and interactive platform for exchanging messages. When a message is sent, it is instantly recorded in Cloud Firestore, functioning as a real-time database. Each message entry includes important attributes such as sender identification, receiver identification, timestamp, and message content. The inclusion of timestamps ensures proper sequencing of messages, thereby maintaining a coherent flow of communication [6]. One of the primary features of the system is real-time data synchronization, enabled through the capabilities of Firestore. Whenever new data is inserted into the database, it is automatically updated across all connected devices without requiring user intervention. This approach allows users to receive messages instantly, ensuring a continuous and uninterrupted communication experience. Additionally, the system eliminates the need for traditional polling techniques, which helps in minimizing network overhead and enhancing overall efficiency. To enhance user engagement, the system integrates Firebase Cloud Messaging for push notifications. When a user receives a message while offline or when the application is running in the background, a notification is triggered to alert the user. The overall workflow of the system involves user authentication, message transmission, UI updates, and notification delivery. These processes work together to create a reliable and efficient chat application. By combining Flutter's frontend capabilities with Firebase's backend services, the system achieves high performance, scalability, and ease of maintenance [7].

5. Results and discussion

The proposed real-time chat messenger with state persistence was successfully developed and tested using Flutter and Firebase. The system was evaluated based on key factors such as real-time message delivery, data consistency, user experience, and overall performance. The outcomes indicate that the application supports rapid communication between users with very low latency, delivering a consistent and dependable messaging experience [8]. During testing, messages sent by users were delivered almost instantly to the receiver without requiring manual refresh. This confirms the effectiveness of Cloud Fire store's real-time synchronization capability. The system preserves the correct sequence of messages through the use of timestamps, allowing conversations to remain well-organized and easy to understand. Even when multiple users were active simultaneously, the application handled message delivery efficiently without noticeable lag [9]. The system's performance was further assessed across varying network conditions. It was observed that the application continued to function effectively even with unstable internet connectivity. In synchronized automatically once the connection was restored. This behavior confirms the reliability of the state persistence feature, which ensures that no data is lost during network interruptions. User experience was another important factor considered during evaluation. The Flutter-based interface provided smooth navigation and quick response to user interactions. Messages appeared instantly on the screen, and the application maintained consistent performance even when switching between different chat screens. Efficient state management methods enable the user interface to update in real time while maintaining the overall responsiveness of the application [10].

6. Challenges and future scope

The development of the real-time chat messenger with state persistence involved several challenges related to real-time synchronization, data consistency, and network dependency. Ensuring instant message delivery while maintaining correct message order required efficient handling of timestamps and database updates. Another challenge was managing application performance during unstable internet connectivity. Although Firebase provides offline caching, handling synchronization conflicts and ensuring data consistency across multiple devices required careful implementation. Even

with these limitations, the system establishes a solid base for future improvements. The application can be further developed by adding features such as group communication, media sharing, and support for voice and video interactions. Future scope also includes adding user presence indicators, typing status, and message read receipts to enhance interaction. Performance optimization techniques can be improve responsiveness. Integration with cloud storage can allow sharing of images, documents, and other media files. With these improvements, the system can evolve into a fully functional and scalable messaging platform suitable for real-world deployment.

7. Conclusion

The real-time chat messenger with state persistence was successfully developed using Flutter and Firebase. The application provides a reliable and efficient platform for instant communication between users. Firebase Authentication ensures secure user login and management, while Cloud Firestore enables real-time synchronization of messages. This allows users to send and receive messages instantly without delays. The use of timestamps helps maintain proper message ordering, ensuring clear and organized conversations. Firebase Cloud Messaging improves the system by enabling push notifications that alert users to new messages, even when the application is running in the background. The system incorporates state persistence to ensure that chat data is preserved during network interruptions or application restarts. Fire store's offline caching mechanism temporarily stores messages locally and synchronizes them automatically when connectivity is restored. This improves reliability and prevents data loss. The Flutter-based user interface provides smooth navigation and responsive performance, enhancing overall user experience. Additionally, the cloud-based architecture supports scalability, allowing multiple users to interact simultaneously without affecting performance.

Conflict of interest statement: The author declares that there is no conflict of interest regarding the publication of this research paper.

Funding information: This project did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors. The work was completed as part of academic coursework using freely available tools such as Flutter and Firebase.

Data availability statement: The data used in this project were obtained from publicly available documentation, online resources, and Firebase testing environments. No confidential or private datasets were used, and no sensitive personal data were collected during development.

Ethical approval statement: This study is based on a systematic review of published literature and does not involve human participants, animals, or sensitive personal data. Therefore, ethical approval was not required.

Acknowledgement: The authors express their sincere gratitude to the department faculty and institution for providing guidance, support, and the necessary resources to complete this project successfully. Their encouragement and technical assistance played a significant role in the completion of this work.

References

1. Napolitano A, Beginning flutter: A hands-on guide to app development, 1st Ed. Indianapolis (IN): Wrox (John Wiley and Sons); 2019, 432p. ISBN: 978-1119550822
2. Sadalage PJ, Fowler M, NoSQL Distilled: A brief guide to the emerging world of polyglot persistence, 1st Ed. Boston (MA): Addison-Wesley professional. 2012, 192p. doi.10.555 5/2381014

3. Newman S, Building micro services: Designing fine-grained systems. 1st Ed. Sebastopol (CA): O' Reilly Media; 2015, 280p. ISBN: 978-1491950357.
4. Banks A, Porcello E, Learning React: Modern patterns for developing react apps, 2nd Ed. Sebastopol (CA): O' Reilly Media. 2020, 310p. ISBN:978-1492044277
5. Windmill E. Flutter in Action, 1st Ed. Shelter Island (NY): Manning Publication, 2020, 350p. ISBN:978-1617296147
6. Payne R. Beginning App Development with Flutter: Create cross-platform mobile apps, 1st Ed. New York (NY): Apress; 2019, 251p. doi:10.1007/978-1-4842-5181-2
7. Heller K, Jellema L, Digital Transformation with Google cloud Platform. 1st Ed. Birmingham (UK): Packt Publishing; 2018, 458p. ISBN: 978-17882956 28
8. Kleppmann M, Designing Data-intensive applications: The big ideas behind reliable, scalable, and maintainable systems, 1st Ed. Sebastopol (CA): O' Reilly Media; 2017, 616p. ISBN: 978-1449373320
9. Kayfitz J, Boyd S, Flutter cookbook: Over 100 proven techniques and solutions for app development, 1st Ed. Birmingham (UK): Packt Publishing; 2020, 552p. ISBN: 978-1838823115
10. Rose D, Hands-On Server less Computing with Google Cloud, 1st Ed. Birmingham (UK): Packt Publishing; 2018, 334p. ISBN: 978-1788836586