Developing an efficient real time chat application on Android: Challenges and Solutions

Dr. Rashmi Shekhar, Assistant Director & Associate Professor
Abhijeet Kumar Mishra, MCA, A453145023016
Amity Institute Of Information Technology, Amity University Patna

Abstract

The demand for real-time communication capabilities within mobile applications continues to surge, reflecting the increasing need for instant connectivity in both personal and professional spheres. The Android platform, with its vast user base and diverse ecosystem of devices, presents a significant opportunity for developers looking to create engaging and interactive chat applications. However, building an efficient real- time chat application on Android is a complex undertaking that involves navigating a multitude of technical challenges while adhering to platform-specific constraints and best practices. This report delves into the fundamental challenges inherent in this development process and explores effective solutions and strategies for creating high-performing and reliable real-time chat experiences on the Android platform.

Keywords: *Real-time communication, Android, User engagement, High performance, Reliability*

Introduction

In the contemporary digital landscape, the ability to connect and communicate instantly has become a fundamental expectation, shaping how we interact both personally and professionally. From coordinating with friends and family to collaborating with colleagues across distances, the demand for seamless, real-time communication capabilities within digital platforms continues its upward trajectory. This surge is particularly evident within the mobile application domain, where users rely on their devices for immediate access to information and interaction. The Android platform, with its unparalleled global reach, diverse ecosystem of devices, and vast user base, stands as a critical battleground for developers aiming to meet this demand by creating engaging and interactive real-time experiences, most notably in the form of chat applications.

Developing a real-time chat application, however, is far from a trivial task. While the concept of sending and receiving messages instantly seems straightforward on the surface, implementing this functionality efficiently and reliably on a mobile operating system like Android introduces a complex array of technical challenges.

Developers must contend with issues such as maintaining persistent network connections without draining device battery, ensuring message delivery guarantee and order, handling offline scenarios and synchronization, managing background processes, optimizing data usage, and adapting to the inherent variability in network conditions and device specifications that characterize the Android ecosystem.

Successfully navigating these complexities while adhering to Android's specific architecture, lifecycle management, and performance best practices is paramount to delivering a high-performing, responsive, and stable chat experience that users expect. This report delves into the intricate world of building real-time chat applications on the Android platform. It aims to provide a comprehensive exploration of the fundamental challenges that developers encounter throughout the development lifecycle, from initial architectural design to deployment and maintenance. By identifying and analysing these critical hurdles, this research seeks to lay

the groundwork for the report will explore and evaluate effective solutions, design patterns, and strategic approaches that can be employed to overcome these obstacles, thereby enabling developers to create robust, efficient, and reliable real-time chat experiences that meet the growing demands of users on the Android
platform.

Literature Review

The evolution of real-time communication on mobile platforms, especially Android, has been a dynamic journey, marked by continuous innovation in protocols, architectural patterns, and platform-specific features. Early mobile chat applications often struggled with reliability and battery efficiency due to the limitations of mobile networks and operating system capabilities. Initially, rudimentary approaches like short polling were employed, where the Android client would repeatedly send HTTP requests to the server to check for new messages. This was highly inefficient, leading to significant battery drain and network overhead. The
introduction of long polling offered a partial improvement, where the server held the connection open until new data was available or a timeout occurred. While reducing the frequency of requests, it still involved
repeated connection setups. A pivotal shift occurred with the broader adoption of Web Sockets as part of HTML5. Web Sockets provide a full-duplex, persistent communication channel over a single TCP connection, significantly reducing latency and overhead compared to traditional HTTP-based polling methods. Libraries like OkHttp in Android provide robust WebSocket client implementations, enabling efficient bi-directional
communication. For server-side scalability with Web Sockets, Node.js with Socket.IO or Java with Spring WebSocket are common choices, designed to handle numerous concurrent connections. For ensuring reliable message delivery and notifications, Firebase Cloud Messaging (FCM), formerly Google Cloud Messaging (GCM), has become the industry standard for Android applications. FCM enables servers to send messages to Android devices without requiring the app to be constantly running in the foreground or maintaining an open WebSocket connection. This is crucial for battery optimization and ensuring messages are delivered even when the app is not active. Research often highlights FCM's role in wake-up calls for background sync and triggering real-time updates. Regarding architectural patterns, the move towards event-driven architectures
and message queues is evident. Server-side message brokers like RabbitMQ, Apache Kafka, or Redis Pub/Sub are frequently used to decouple the real-time communication service from other backend services, ensuring
scalability and reliability. On the Android client, this translates to subscribing to specific topics or channels and processing incoming events. Local data persistence on Android for chat history and offline access is
another area of significant research. Solutions like SQLite databases (often abstracted by libraries like Room Persistence Library) and Realm DB provide efficient ways to store and query structured data on the device (Android Developers, Room, n.d.; Realm, n.d.). The challenge lies in synchronizing this local data with the server's state, especially when the user is offline and then reconnects. The literature also extensively covers Android's background execution limits and power management features, such as Doze mode and App
Standby, introduced from Android 6.0 (Marshmallow) onwards. These features aim to conserve battery but impose strict limitations on what applications can do in the background. Developers must use APIs like Work Manager or Job Scheduler for deferrable background tasks (e.g., syncing chat history, uploading media) and design real-time components that are lifecycle-aware to minimize resource consumption . Security is a paramount concern. End-to-end encryption (E2EE) for message content is a recurring theme, with frameworks like Signal Protocol being a reference. On

the Android side, secure data storage using Encrypted Shared Preferences and Android KeyStore, along with ensuring secure network communication (TLS/SSL),

are well-documented best practices. In essence, the literature demonstrates a clear trajectory in Android real- time chat development: from basic polling to efficient WebSocket communication, complemented by robust push notification services like FCM, resilient backend architectures, and careful consideration of Android's
specific resource management and security features.

Methodology

This study follows a **qualitative research approach**, focusing on the analysis of official Android developer documentation to identify challenges and solutions related to real-time chat application development.

1. Data Collection: The primary source of data comprises official documentation, API references, and best practices provided by Android developers. • Additional insights were gathered from developer forums, technical blogs, to supplement the findings.
2. Data Analysis: A thematic analysis was conducted to categorize common challenges in real-time communication, such as latency management, concurrency handling, and network optimization. Best practices, libraries, and solutions recommended by Android experts were systematically reviewed to form a comparative study of viable approaches.
3. Limitations: The study relies on publicly available documentation, which may not cover proprietary frameworks or unpublished solutions. The effectiveness of solutions may vary based on specific use cases, network conditions, and device capabilities.

Findings

1. Challenges

Developing a real-time chat application on Android presents several key challenges that developers must address to ensure a seamless and efficient user experience. These challenges span various aspects of
application development, from network communication and concurrency management to user interface design and battery optimization.

1.1 Achieving Low-Latency and Reliable Real-Time Communication

One of the primary hurdles in building a real-time chat application is achieving low latency and reliable message delivery. Mobile networks, whether cellular or Wi-Fi, inherently introduce latency that can impact the responsiveness of real-time interactions.[1] Users expect messages to be delivered almost instantaneously, mirroring the immediacy of face-to-face conversations.[3] Any noticeable delay can lead to user frustration and a perception of poor application performance.

Furthermore, maintaining persistent connections, crucial for real-time data exchange, can be challenging in mobile environments characterized by frequent network changes, signal fluctuations, and intermittent
connectivity.[5] The application must be designed to handle these conditions gracefully, ensuring that messages are delivered reliably even when network conditions are less than ideal.

1.2 Managing Concurrency and Ensuring Scalability for a Large User Base

As chat applications gain popularity, they often need to support a large number of concurrent users and handle a high volume of messages in real time.[7] Managing this level of concurrency without compromising performance or stability is a significant challenge. The application's backend infrastructure must be capable of handling numerous active connections and efficiently routing messages to the intended recipients.[9] Moreover, the system needs to be scalable to accommodate future growth in the user base and message volume.[11] This requires careful architectural design, including the selection of appropriate backend technologies and the implementation of strategies for load balancing and efficient resource management.

1.3 Optimizing Network Usage for Efficient Data Transfer

Efficient data transfer is paramount for a positive user experience in mobile chat applications. Users are often conscious of their data usage, especially when on metered cellular networks.[14] Therefore, minimizing the amount of data transferred for each message and reducing overall network traffic is crucial. This necessitates the use of efficient data serialization techniques to reduce message sizes and minimize transmission overhead.[15] Additionally, employing strategies such as bundling multiple messages into a single request and prefetching relevant data can help reduce the number of network connections and improve efficiency.[14] By optimizing network usage, developers can ensure that the chat application is responsive and does not consume excessive data, leading to better user satisfaction.

1.4 Minimizing Battery Consumption in Persistent Chat Applications

Chat applications, by their nature, often run in the background to provide real-time notifications of new messages. This persistence can lead to significant battery drain if not managed carefully.[14] Frequent network activity, even when the application is not actively in use, can quickly deplete the device's battery, leading to a negative user experience.[14] Developers must employ strategies to minimize battery consumption, such as utilizing Android's power-saving features like Doze and App Standby effectively.[20] Optimizing background processes and network connections is essential to ensure that the chat application can run persistently without unduly impacting the device's battery life.

1.5 Designing an Intuitive and Responsive User Interface for Real-Time Interaction

The user interface (UI) of a real-time chat application plays a critical role in its success. It must be intuitive and responsive, facilitating seamless and instant communication between users.[21] Handling real-time updates and displaying new messages without causing UI lag or disruption is a key challenge.[24] The interface should be designed to provide clear visual cues for message status, such as sending, sent, delivered, and read indicators.[3] Features like typing indicators and presence status can further enhance the real-time feel of the application.[3] Developers must also address usability issues specific to mobile chat applications, ensuring that touch targets are appropriately sized, navigation is intuitive, and information is presented clearly without overwhelming the user.[26]

1.6 Effective Data Management: Storage, Synchronization, and Persistence

Managing data effectively is crucial for the functionality and performance of a real-time chat application. Developers need to choose appropriate data storage solutions, whether local databases like Room or cloud-based databases like Firebase, based on the specific requirements of the application.[24] Implementing efficient data synchronization mechanisms between the client and the server is essential to ensure that all users have the latest messages and information.[29] Furthermore, the application must ensure message persistence, allowing users to access their message history even when offline, and reliably deliver messages once a connection is re-established.[3]

1.7 Implementing Robust Security and Privacy Measures

Security and privacy are paramount concerns for users of chat applications. Developers must implement robust security measures to protect user data and chat messages from unauthorized access.[33] This includes employing encryption techniques, such as end-to-end encryption, to ensure that only the intended recipients can read the messages.[33] Secure user authentication and authorization mechanisms are also essential to verify user identities and control access to the application's features.[35] Additionally, developers must be vigilant in protecting against common security threats like data leaks and unauthorized access, adhering to security best practices throughout the development process.[38]

1.8 Handling Unreliable Network Conditions and Ensuring Message Delivery

Mobile networks are often unreliable, with intermittent connectivity being a common occurrence.[6] A robust real-time chat application must be able to handle these unreliable conditions and ensure that messages are eventually delivered to the recipients. This requires implementing mechanisms for message delivery confirmation, allowing the sender to know if a message has been successfully received.[42] Retry mechanisms with exponential backoff can be used for failed message sends, ensuring that messages are eventually delivered once the network connection is restored.[42] The application should also gracefully handle disconnections and reconnections without losing data, providing a consistent user experience even in challenging network environments.[25] Persisting messages locally until they are successfully sent can further enhance reliability in offline scenarios.[29]

1.9 Addressing Android-Specific Background Task Limitations

The Android operating system imposes limitations on background tasks to conserve device resources and improve battery life.[46] Developers of real-time chat applications must be aware of these limitations and choose the appropriate Android APIs for performing background operations efficiently. Work Manager is often the preferred solution for deferrable and guaranteed background tasks, such as sending undelivered messages or periodic data synchronization.[46] Foreground services can be used judiciously for essential real- time operations that require user awareness, such as maintaining an active voice or video call or a persistent connection when the app is in the background, always accompanied by a user-visible notification.[46]

Understanding the specific restrictions and best practices for using background services in different Android versions is crucial for developing an efficient and well-behaved chat application.[48]

2. Solutions

To overcome the challenges outlined above and build an efficient real-time chat application on Android, developers can leverage a variety of technologies and adhere to established best practices. These solutions encompass the selection of appropriate communication technologies, optimization of network and data usage, careful architectural design, and a focus on delivering a high-quality user experience.

2.1 Selecting the Optimal Real-Time Communication Technology for Android

Choosing the right technology for real-time communication is a foundational step in developing an efficient chat application. Several options are available for Android developers, each with its own strengths and trade- offs:

- **Firebase Realtime Database:** This backend-as-a-service from Google provides real-time data synchronization across connected clients.[29] Its ease of integration with Android makes it a popular choice for developers. Firebase simplifies backend development by handling the complexities of data storage and synchronization, allowing developers to focus on the application's frontend logic.

- **Web-Sockets:** This protocol enables persistent, full-duplex communication channels over a single TCP connection, offering low-latency data exchange between the client and server.[25] For Android development, libraries like OkHttp provide robust implementations of the WebSocket protocol, allowing developers to establish and manage WebSocket connections efficiently.[34] Web-Sockets are particularly well-suited for applications requiring very low latency and bidirectional communication, such as real- time chat.

- **MQTT (Message Queuing Telemetry Transport):** This lightweight publish/subscribe messaging protocol is designed for efficiency and reliability, especially in resource-constrained environments like IoT devices.[55] While often used in IoT, MQTT can also be beneficial for chat applications that need to be highly efficient in terms of bandwidth and power consumption, particularly when supporting a large number of concurrent users or devices.

- **Firebase Cloud Messaging (FCM):** This cross-platform messaging solution is primarily used for efficiently delivering notifications and data messages to Android devices.[20] FCM is particularly valuable for handling background updates and waking up the application to notify users of new messages, minimizing battery drain compared to maintaining persistent connections.

The following table provides a comparison of these technologies:

Feature	**Firebase Realtime Database**	**Web-Sockets**	**MQTT**	**FCM**
Latency	Low	Very Low	Low	High (for notifications)
Reliability	High	Medium (requires handling)	High	High (for notifications)

Scalability	High	High	Very High	Very High
Battery Efficiency	Medium	Medium (requires careful use)	High	High
Implementation	Easy	Medium	Medium	Easy
Use Case	Real-time data, chat	Real-time chat, gaming	IoT, efficient messaging	Notifications, background updates

Choosing the most appropriate technology depends on the specific requirements of the chat application, including the desired level of latency, the expected scale, and the importance of battery efficiency.

2.2 Implementing Efficient Data Serialization and Transfer Strategies

Efficiently handling data is crucial for optimizing network usage and application performance. Developers can employ several strategies to achieve this:

- **Using efficient serialization formats:** Formats like JSON are widely used and relatively lightweight, making them suitable for serializing chat messages.[15] Protocol Buffers offer even higher efficiency in terms of speed and size but may have a steeper learning curve.

- **Compressing data:** Compressing messages before transmission can significantly reduce bandwidth usage, especially when sending large amounts of text or metadata.[14] Libraries are available on Android to easily compress and decompress data.

- **Optimizing data structures:** Designing data structures to include only the necessary information in each message can minimize the amount of data being transferred.[29] Avoiding redundant data and using efficient data types can lead to smaller message sizes.

2.3 Network Optimization Techniques for Real-Time Data

To further optimize network usage in real-time chat applications, developers can implement the following techniques:

- **Bundling data transfers:** Grouping multiple messages or updates into a single network request can reduce the overhead associated with establishing and closing connections for each individual message.[14]

- **Prefetching data:** Anticipating the user's needs and pre-fetching relevant data, such as older messages in a conversation, can improve perceived performance and reduce latency when the user requests that data.[14]

- **Checking for connectivity:** Before initiating a network request, the application should check if a network connection is available to avoid unnecessary attempts and potential battery drain.[14]

- **Pooling connections:** Reusing existing network connections instead of establishing new ones for each request can be more efficient and reduce latency.[14]

2.4 Battery Optimization Strategies for Network Communication and Background Services

Minimizing battery consumption is critical for the long-term usability of a chat application. Developers can employ several strategies to achieve this:

- **Using FCM for push notifications:** Relying on FCM to notify the application of new messages instead of constantly polling the server is a key strategy for battery conservation.[20]
- **Scheduling background tasks with Work Manager:** For tasks that do not require immediate execution, such as sending undelivered messages or periodic synchronization, Work Manager allows developers to schedule these tasks with constraints like requiring Wi-Fi connectivity or the device being in a charging state.[46]
- **Judicious use of foreground services:** Foreground services should be reserved for essential real- time operations that require the application to run persistently in the background, such as active calls. These services should always be accompanied by a user-visible notification to inform the user of their activity.[46]
- **Respecting Doze and App Standby:** The application should be designed to function efficiently within Android's power-saving modes. Using high-priority FCM messages for urgent notifications can ensure timely delivery even when the device is in Doze mode.[20]

2.5 Architectural Patterns for Scalable and Concurrent Chat Applications

A well-defined architecture is essential for building a chat application that can scale to support a large number of concurrent users while maintaining performance and reliability:

- **Client-server architecture:** This is a common pattern where the Android application acts as the client, communicating with a backend server that handles message routing, data storage, and other core functionalities.[60]
- **Microservices architecture:** For larger and more complex chat applications, adopting a microservices architecture can offer benefits like increased flexibility and scalability. This involves breaking down the application into smaller, independent services that can be developed, deployed, and scaled individually.[62]
- **Message queues:** Using message queue technologies like RabbitMQ or Apache Kafka can help handle asynchronous message processing, ensuring reliable delivery and decoupling different components of the application.[64]
- **Load balancing:** Distributing incoming traffic across multiple servers is crucial for handling high concurrency and ensuring that no single server becomes a boflleneck.[11]

2.6 Ensuring Message Delivery Reliability and Handling Disconnections

Reliable message delivery is a fundamental requirement for a chat application. Developers can employ several techniques to achieve this:

- **Message acknowledgment:** Implementing mechanisms for the sender to receive confirmation that their message has been successfully delivered to the recipient can improve user trust.[42]
- **Retry mechanisms:** For messages that fail to send initially due to network issues, implementing automatic retry

mechanisms with strategies like exponential backoff can ensure that messages are eventually delivered once the connection is restored.[42]

- **Connection keep-alive:** Implementing strategies to maintain a persistent connection with the server, such as sending periodic heartbeat messages, can help prevent disconnections due to inactivity.[25]
- **Automatic reconnection:** The application should be designed to automatically attempt to reconnect to the server if the connection is lost, ensuring that users can resume their conversations quickly.[25]
- **Local message persistence:** Storing messages locally on the device until they are successfully sent can prevent data loss in case of temporary network outages.[29]

2.7 Leveraging Android Background Task APIs Effectively (Work Manager, Foreground Services)

Android provides specific APIs for managing background work, and using them correctly is essential for building an efficient chat application:

- **Work Manager:** This API is recommended for most background processing tasks that need to be reliable and can be deferred. For a chat application, Work Manager can be used to handle tasks like sending messages that failed due to network issues, syncing message history periodically, or uploading logs.[46] Work Manager allows developers to define constraints for when these tasks should run, such as only when the device is charging or connected to Wi-Fi, helping to optimize battery usage.
- **Foreground Services:** These services should be used for tasks that are critical to the user and need to continue running even when the app is in the background, such as maintaining an active voice or video call. When using a foreground service, the application must display a notification to the user, making them aware that the service is running and consuming resources.[46] For a chat application, a foreground service might be used to maintain a low-latency connection for receiving messages when the app is minimized. Developers need to be mindful of the restrictions on starting foreground services from the background in newer Android versions and use them judiciously.

2.8 Designing for Optimal User Experience in Real-Time Chat Interfaces

The user interface of a chat application should be designed to facilitate efficient and enjoyable real- time communication:

- **Clear message status:** Providing visual indicators for the status of each message, such as sending, sent, delivered, and read receipts, helps users understand the communication flow and reduces uncertainty.[3]
- **Typing indicators and presence status:** Displaying an indicator when another user is typing a message and showing their online status can enhance the sense of real-time interaction.[3]
- **Optimized message display:** Using Recycler View with appropriate optimizations for displaying chat messages can ensure smooth scrolling and efficient rendering, even in long conversations.[24]
- **Rich media support:** Allowing users to send and receive images, videos, and files seamlessly within the chat interface enhances the communication experience.[3] Efficient handling of these media types, including compression and caching, is important for performance.
- **Advanced chat features:** Implementing features like message threading to maintain context in group chats, message reactions for quick responses, and the ability to edit or delete sent messages can significantly improve the

user experience.[3]

- **Usability considerations:** Paying close attention to usability principles, such as ensuring adequate touch target sizes, intuitive navigation, and clear information hierarchy, is crucial for creating a user-friendly chat application.[26]

2.9 Implementing End-to-End Encryption and Secure Authentication on Android

Security and privacy are paramount for chat applications. Developers should implement robust measures to protect user data and communications:

- **End-to-end encryption:** Using established encryption protocols like AES to encrypt messages before they leave the sender's device and decrypting them only on the recipient's device ensures that the message content remains private, even from the server.[34] Libraries like the Android Keystore can be used to securely store the encryption keys. Considering the Signal Protocol, which is known for its strong end-to-end encryption, is also advisable.[70]

- **Secure authentication:** Implementing secure user authentication mechanisms, such as OAuth 2.0 or Firebase Authentication, is essential to verify user identities and protect against unauthorized access.[35] These methods often involve using tokens to maintain session security and avoid storing sensitive credentials directly on the device.

- **Secure network communication:** Ensuring that all communication between the Android application and the backend server occurs over a secure channel using HTTPS and TLS/SSL protocols is crucial to protect data in transit from eavesdropping and tampering.[34]

Case Studies

Examining popular real-time chat applications on Android can provide valuable insights into how these challenges are addressed in practice. Applications like WhatsApp, Telegram, and Signal have successfully implemented efficient real-time communication while managing scalability and battery consumption. These applications demonstrate various approaches to tackling the challenges of real- time chat development on Android. WhatsApp, known for its massive user base, leverages a robust backend in Erlang and the battle-tested Signal Protocol for encryption. Telegram emphasizes speed and a wide range of features, while Signal prioritizes security and transparency with its open-source Signal Protocol. All three applications utilize FCM for efficient push notifications, a cornerstone of battery-friendly real-time messaging on Android.

Application	Backend Technology	Real-Time Protocol	Encryption Method	Battery Optimization Techniques
WhatsApp	Erlang	Proprietary	End-to-End (Signal Protocol)	FCM for notifications, efficient network usage, aggressive connection pooling [63]

Telegram	Proprietary	MTProto	End-to-End (Secret Chats)	FCM for notifications, optimized data transfer, non-default end-to-end encryption [71]
Signal	Proprietary	Signal Protocol	End-to-End (Signal Protocol)	FCM for notifications, open-source encryption protocol, focus on privacy and security [40]

Conclusion

Developing an efficient real-time chat application on Android is a multifaceted endeavor that demands careful consideration of various technical challenges. Achieving low latency and reliable communication, managing concurrency at scale, optimizing network and battery usage, designing an intuitive user interface, ensuring effective data management, implementing robust security measures, handling unreliable network conditions, and adhering to Android's background task limitations are all critical aspects that developers must address. By selecting the optimal real-time communication technology based on the application's specific needs, implementing efficient data serialization and transfer strategies, employing network and battery optimization techniques, adopting scalable architectural patterns, ensuring message delivery reliability, leveraging Android background task APIs effectively, designing for optimal user experience, and prioritizing security and privacy, developers can build high-performance real-time chat applications that meet the demands of today's users. Examining successful applications in the market provides valuable insights into effective strategies and technology choices. Ultimately, a holistic approach that balances efficiency, real-time capabilities, scalability, security, and user experience is essential for creating a successful real-time chat application on the Android platform.

References:

1. People and conversations | Views - Android Developers, accessed on May 5, 2025, https://developer.android.com/develop/ui/views/notifications/conversations/
2. About notifications and conversations | Android social, accessed on May 5, 2025, https://developer.android.com/social-and-messaging/guides/communication/notifications- conversations /
3. Take your messaging to the next level — basic, better, and best | Android social, accessed on May 5, 2025, https://developer.android.com/social-and-messaging/guides/communication/basic-better-best
4. Run a sync adapter | Connectivity - Android Developers, accessed on May 5, 2025, https://developer.android.com/training/sync-adapters/running-sync-adapter

5. Android App Persistent Connection : r/homeassistant - Reddit, accessed on May 5, 2025, https://www.reddit.com/r/homeassistant/comments/1i0hgtl/android_app_persiste_nt_connection/
6. Best practice for persistent mobile connections on Android? - Stack Overflow, accessed on May 5, 2025, https://stackoverflow.com/questions/11142443/best-practice-for-persistent-mobi le-connections-on-android/
7. What it takes to build a real-time chat or messaging app, accessed on May 5, 2025, https://ably.com/blog/what-it-takes-to-build-a-realtime-chat-or-messaging-app
8. Scalable Messaging App Strategy for 2024 - Protocloud Technologies Pvt. Ltd., accessed on May 5, 2025, https://www.protocloudtechnologies.com/scalable-messaging-app-strategy-for- 2024-2/
9. high concurrency chat application - node.js - Stack Overflow, accessed on May 5, 2025, https://stackoverflow.com/questions/8447747/high-concurrency-chat-application
10. Build a realtime chat app which stores messages in a MySQL database, accessed on May 5, 2025, https://softwareengineering.stackexchange.com/questions/363928/build-a-realti me-chat-app-which-stores-messages-in-a-mysql-database/
11. Best Practices for Building Scalable Realtime Apps - PubNub, accessed on May 5, 2025, https://www.pubnub.com/blog/best-practices-building-scalable-realtime-apps/
12. The ultimate guide to chat app architecture: how to build a scalable and secure messaging platform | RST Software, accessed on May 5, 2025, https://www.rst.software/blog/chat-app- architecture
13. Mastering Scalable Android Apps: Proven Strategies for Success - Netguru, accessed on May 5, 2025, https://www.netguru.com/blog/scalable-android-apps
14. Optimize network access | Connectivity | Android Developers, accessed on May 5, 2025, https://developer.android.com/develop/connectivity/network-ops/network-acces_s-optimization/
15. Buffing Up Your Messages: Making Use Of 3rd-Party Data Serialization Systems - Solace, accessed on May 5, 2025, https://solace.com/blog/buffing-messages-3rd-party-data-serialization/
16. Android app and serialization/deserialization efficiency? - Stack Overflow, accessed on May 5, 2025, https://stackoverflow.com/questions/34519651/android-app-and-serialization-deserialization- efficiency
17. Data Handling with Kotlin Serialization | AppMaster, accessed on May 5, 2025, https://appmaster.io/blog/data-handling-with-kotlin-serialization
18. Analyze network traffic data with the Network Traffic tool | App quality ..., accessed on May 5, 2025, https://developer.android.com/topic/performance/power/network/analyze-data
19. Excessive battery usage | App quality - Android Developers, accessed on May 5, 2025, https://developer.android.com/topic/performance/vitals/excessive-battery-usage
20. Optimize for Doze and App Standby | App quality - Android Developers, accessed on May 5, 2025, https://developer.android.com/training/monitoring-device-state/doze-standby
21. (PDF) DESIGN AND IMPLEMENTATION OF A REAL TIME CHAT APPLICATION SYSTEM, accessed on May 5, 2025, https://www.researchgate.net/publication/390161526_DESIGN_AND_IMPLEMENTATION_OF_A_REAL_TIME_CHAT_APPLICATION_SYSTEM/
22. How to Build a Android Chat Application 2025 - CONTUS Tech, accessed on May 5, 2025, https://www.contus.com/blog/build-a-android-chat-app/
23. Text in social and messaging apps - Android Developers, accessed on May 5, 2025, https://developer.android.com/social-and-messaging/guides/communication/text
24. Chatting Application in Android with Kotlin | GeeksforGeeks, accessed on May 5, 2025, https://www.geeksforgeeks.org/chatting-application-in-android-with-kotlin/
25. Real-time chat application in JavaScript | GeeksforGeeks, accessed on May 5, 2025, https://www.geeksforgeeks.org/real-time-chat-application-in-javascript/
26. Identifying Usability Issues in Instant Messaging Apps on iOS and Android Platforms, accessed on May 5, 2025, https://www.researchgate.net/publication/328222223_Identifying_Usability_Issues_in_Instant_Messaging_Apps_on_iOS_and_Android_Platforms_/
27. Messaging Application Usability Test Cycle | Case Study - TesterWork, accessed on May 5, 2025, https://testerwork.com/messaging-application-usability-test-cycle-case-study/
28. Mobile App Usability Testing: Common Issues and Testing Methods, accessed on May 5, 2025, https://lollypop.design/blog/2025/february/mobile-app-usability-testing/
29. Firebase Realtime Database - Google, accessed on May 5, 2025, https://firebase.google.com/docs/database
30. Firebase Realtime Database Documentation | PDF | Software Engineering - Scribd, accessed on May 5, 2025, https://www.scribd.com/document/603223318/Firebase-Realtime-Database-Docmentation/
31. Realtime Database - React Native Firebase, accessed on May 5, 2025, https://rnfirebase.io/database/usage

32. Sync data | Wear OS - Android Developers, accessed on May 5, 2025, https://developer.android.com/training/wearables/data/sync/
33. https://developer.android.com/training/wearables/data/sync 33. Chat Application Using Homomorphic Encryption - ITM Web of Conferences, accessed on May 5, 2025, https://www.itm-conferences.org/articles/itmconf/pdf/2022/10/itmconf_icaect202_2_01011.pdf/
34. I am developing an app, how do I provide secure encryption of user data and chat messages? Or at least where do I start learning this? - Quora, accessed on May 5, 2025, https://www.quora.com/I- am-developing-an-app-how-do-I-provide-secure-encr yption-of-user-data-and-chat-messages-Or-at- least-where-do-I-start-learning -this
35. Firebase Android Codelab - Build Friendly Chat, accessed on May 5, 2025, https://firebase.google.com/codelabs/firebase-android
36. Firebase Authentication, accessed on May 5, 2025, https://firebase.google.com/docs/auth
37. Tokens & Authentication - Android Chat Messaging Docs - GetStream.io, accessed on May 5, 2025, https://getstream.io/chat/docs/android/tokens_and_authentication/
38. New Android Serialization Vulnerability Gives Underprivileged Apps Super Status - IBM's Security Intelligence, accessed on May 5, 2025, https://securityintelligence.com/one-class-to-rule- them-all-new-android-serializa tion-vulnerability-gives-underprivileged-apps-super-status/
39. How to Implement Data-at-Rest Encryption in Android Apps Using AI - Appdome, accessed on May 5, 2025, https://www.appdome.com/how-to/mobile-app-security/mobile-data-encryption /mobile-app-data-encryption-explained/
40. Most secure messaging apps - Kaspersky, accessed on May 5, 2025, https://usa.kaspersky.com/resource-center/preemptive-safety/messaging-app-security/
41. Minimize the effect of regular updates | Connectivity - Android Developers, accessed on May 5, 2025, https://developer.android.com/develop/connectivity/minimize-effect-regular-updates
42. The Best Private Messaging Apps for 2025 - PCMag, accessed on May 5, 2025, https://www.pcmag.com/picks/best-secure-messaging-apps
43. Any texting apps that give you a delivery report? - Android Central Forums, accessed on May 5, 2025, https://forums.androidcentral.com/threads/any-texting-apps-that-give-you-a-del ivery- report.960761/
44. Google Messages reliability has taken a dive off a cliff in the last week - Reddit, accessed on May 5, 2025, https://www.reddit.com/r/GooglePixel/comments/10m5s3e/google_messages_reli ability_has_taken_a_dive_off/
45. Online Chat Application Project in Software Development | GeeksforGeeks, accessed on May 5, 2025, https://www.geeksforgeeks.org/online-chat-application-project-in-software-dev elopment/
46. Background tasks overview | Background work | Android Developers, accessed on May 5, 2025, https://developer.android.com/develop/background-work/background-tasks
47. System restrictions on background work - Android Developers, accessed on May 5, 2025, https://developer.android.com/develop/background-work/background-tasks/bg- work-restrictions/
48. Background Execution Limits - Android Developers, accessed on May 5, 2025, https://developer.android.com/about/versions/oreo/background
49. Foreground service types | Background work - Android Developers, accessed on May 5, 2025, https://developer.android.com/develop/background-work/services/fgs/service-types/
50. Foreground service types are required - Android Developers, accessed on May 5, 2025, https://developer.android.com/about/versions/14/changes/fgs-types-required
51. Read and Write Data on the Web | Firebase Realtime Database - Google, accessed on May 5, 2025, https://firebase.google.com/docs/database/web/read-and-write/
52. Receive messages reliably | Android social - Android Developers, accessed on May 5, 2025, https://developer.android.com/social-and-messaging/guides/communication/rece iving-messages/
53. Media projection - Android Developers, accessed on May 5, 2025, https://developer.android.com/media/grow/media-projection/
54. [Tutorial] Learn to use WebSockets on Android with OkHttp | XDA Forums, accessed on May 5, 2025, https://xdaforums.com/t/tutorial-learn-to-use-websockets-on-android-with-okht tp.3558194/

55. MQTT - Home Assistant, accessed on May 5, 2025, https://www.home- assistant.io/integrations/mqtt/
56. MQTT man page - Eclipse Mosquitto, accessed on May 5, 2025, https://mosquitto.org/man/mqtt- 7.html/
57. MQTT Essentials - All Core Concepts Explained - HiveMQ, accessed on May 5, 2025, https://www.hivemq.com/mqtt/
58. Firebase Cloud Messaging, accessed on May 5, 2025, https://firebase.google.com/docs/cloud- messaging/
59. Development and Integration of AI-Powered Real-Time Chat Application Using OpenAI and Chat Engine APIs - ResearchGate, accessed on May 5, 2025, https://www.researchgate.net/publication/381105843_Development_and_Integrati on_of_AI- Powered_Real-Time_Chat_Application_Using_OpenAI_and_Chat_Engine_APIs/
60. Architecture for a mobile (Android) chat application - Software Engineering Stack Exchange, accessed on May 5, 2025, https://softwareengineering.stackexchange.com/questions/262497/architecture-f or-a-mobile-android-chat-application/
61. Understanding the Architecture & System Design of a Chat Application - CometChat, accessed on May 5, 2025, https://www.cometchat.com/blog/chat-application-architecture-and-system-design/
62. Strategies for Real-Time Communication Apps on Android - MoldStud, accessed on May 5, 2025, https://moldstud.com/articles/p-strategies-for-creating-apps-with-real-time-co mmunication- features-for-android-users/
63. Build a Real time Chat App For Android & iOS [2025 Guide] - MirrorFly, accessed on May 5, 2025, https://www.mirrorfly.com/blog/how-to-build-a-real-time-chat-app/
64. Design A Chat System - ByteByteGo | Technical Interview Prep, accessed on May 5, 2025, https://bytebytego.com/courses/system-design-interview/design-a-chat-system/
65. Message Queues – System Design | GeeksforGeeks, accessed on May 5, 2025, https://www.geeksforgeeks.org/message-queues-system-design/
66. Choosing The Right Message Queue Technology For Your Notification System, accessed on May 5, 2025, https://www.suprsend.com/post/choosing-the-right-message-queue-technology -for- your-notification-system/
67. Optimize RecycleView for Chat Android - Stack Overflow, accessed on May 5, 2025, https://stackoverflow.com/questions/54761297/optimize-recycleview-for-chat-an droid/
69. Encrypting data for Android mobile app - Information Security Stack Exchange, accessed on May 5, 2025, https://security.stackexchange.com/questions/25104/encrypting-data-for-androi d-mobile- app/
70. Key management for End-to-end encryption for Chat application, accessed on May 5, 2025, https://security.stackexchange.com/questions/257828/key-management-for-end -to-end-encryption- for-chat-application/
71. Best encrypted messaging app for Android of 2025 - TechRadar, accessed on May 5, 2025, https://www.techradar.com/best/best-encrypted-messaging-app-android/
72. Comparing Chat Applications Encryption, Privacy and Safe Usage - TeleMessage, accessed on May 5, 2025, https://www.telemessage.com/comparing-chat-applications-encryption-privacy- and- safe-usage/
73. Authentication using existing websites in Android chat apps - Stack Overflow, accessed on May 5, 2025, https://stackoverflow.com/questions/50909676/authentication-using-existing-we bsites-in- android-chat-apps/
74. Connect to the network - Android Developers, accessed on May 5, 2025, https://developer.android.com/develop/connectivity/network-ops/connecting/
75. Improve your app's security - Android Developers, accessed on May 5, 2025, https://developer.android.com/privacy-and-security/security-best-practices/
76. Case Study: How WhatsApp Became the Best Messaging App - Protocloud Technologies, accessed on May 5, 2025, https://www.protocloudtechnologies.com/whatsapp-case-study/
77. Android Developers. (n.d.). *Background execution limits*. Retrieved from https://developer.android.com/guide/components/background-execution-limits
78. Android Developers. (n.d.). *Room Persistence Library*. Retrieved from https://developer.android.com/jetpack/androidx/releases/room
79. Android Developers. (n.d.). *Security best practices*. Retrieved from https://developer.android.com/training/articles/security-tips
80. Android Developers. (n.d.). *Work Manager*. Retrieved from https://developer.android.com/topic/libraries/architecture/workmanager
81. Google Firebase. (n.d.). *Firebase Cloud Messaging*. Retrieved from https://firebase.google.com/docs/cloud-messaging
82. Realm. (n.d.). *Realm Database*. Retrieved from https://realm.io/
83. Square. (n.d.). *OkHttp*. Retrieved from https://square.github.io/okhttp/