End-To-End Development of a Robust Live Chat System for Real-Time Communication

G. Gangadevi[1], ChiragAgrawal[2], JigneshChouhan[3], Gautam Verma[4]
{ganga2007mtech@gmail.com[1], as9922@srmist.edu.in[2], jb1275@srmist.edu.in[3], gr6848@srmist.edu.in[4]}

Department of Computer Science and Engineering, SRM Institute of Science and Technology, Ramapuram, Chennai, Tamil Nadu, India[1, 2, 3, 4]

Abstract. Real-time interaction in social networks, collaborative work environments, and customer service operations is critical, but most systems have unsolved problems related to scalability, security, and interoperability. This work introduces the design and implementation of a chat application that allows real-time interaction, leveraging the capabilities of Python, Django, and WebSockets for real-time updates. The application promotes interactivity and responsiveness through one-to-one chats, group conversations, and broadcast capabilities. Scalability is achieved through optimization methods like load balancing, caching, and session-based encryption for enhanced security. Added user usability includes rich media sharing, multilingual capability, system-managed controls, and administrative dashboards with analytics. The solution aims to provide users with a reliable, versatile, and dynamically responsive high-performance system that chat platform users can depend on as their needs increase. The challenge is maintaining security while increasing interactivity.

Keywords: Real-time communication, scalability, security, interactivity, usability.

1 Introduction

The ever-increasing demand for social networking, online team collaboration, and customer support functions has real time communication as one of the most important chat interactions. In the context of traditional messaging, platforms oftentimes face issues regarding scalability, security, engagement, and a myriad of other problems which could certainly benefit from more advanced solutions [1]. Currently, digital text communication has outgrown and evolved into a widely sought means of interaction. Other functionalities like sending and accepting friend requests, managing profiles, and secure messaging significantly improves user engagement and experience. In addition, the capability to perform real-time sentiment assessment of messages adds to the novelty of the application where background of the chat interface can be altered based on the detected tone of messages, thus making conversations more interactive. The frontend is implemented on React, HTML, and CSS for an appealing interactive and user-friendly graphical interface, diego along with the baton rest framework provides responsive backend processing alongside scalable system performance. Furthermore, advice provided support for push notifications, file sharing, and multi-language settings, hostile within reach of a broad and diverse audience. To increase its reach further, the chat system can be integrated with external third-party APIs, greatly enhancing its adaptability. With rapidly shifting, this chat application stands as a versatile and scalable solution capable of handling increasing user demands while maintaining security and real-time efficiency.

ICITSM-Part I 2025, April 28-29, Tiruchengode, India
Copyright © 2025 EAI
DOI 10.4108/eai.28-4-2025.2357824

2 Existing Systems

Traditional chat systems were limited in functionality, focusing mainly on one-to-one messaging and using inefficient polling mechanisms for retrieving data, which caused unnecessary delays and resource wastage [1]. Over time, more advanced solutions were explored, such as WebSocket adoption, which revolutionized the landscape of real-time communication by providing bi-directional, low-latency connections [2].

However, older studies noted that communication and display of real-time data still relied on resource-heavy polling, which reduced efficiency [3]. Developers worked to enhance performance and responsiveness in real-time web applications by leveraging WebSockets, highlighting measurable improvements in throughput and latency [4]. Alongside performance, security researchers analyzed the attack surface of WebSocket, emphasizing the importance of robust encryption and secure user authentication [6].

The application of HTML5 WebSocket protocols in distributed computing further demonstrated the technology's scalability and adaptability across platforms [4]. Popular frameworks such as Socket.IO provided simplified integration and fallbacks, making it easier to deploy WebSocket-based systems widely [7]. Similarly, push technologies expanded the possibilities of real-time communication, though their scalability remained limited when compared to WebSockets [8].

To address scalability and backend management, Django Channels was introduced, allowing developers to integrate WebSockets directly into Django applications [9]. Tutorials and practical case studies showcased how Django Channels could be used to build chat applications with real-time messaging [10]. Blogs and technical evaluations also compared the advantages and drawbacks of Django Channels, helping developers weigh performance versus ease of implementation [11].

Research on real-time chat applications published in peer-reviewed journals confirmed the growing adoption of WebSockets in academic and practical implementations [2]. Additionally, surveys and documentation have highlighted WebSocket technology itself as a key enabler of reliable real-time communication across industries [12]. Comparative studies have further revealed insights into performance trade-offs of WebSockets in various environments [13].

Recent innovations include secure, real-time messaging systems for social and academic platforms, which integrate authentication and multimedia features [11]. Together, these studies highlight the steady evolution of chat systems from basic polling-based designs to modern solutions that emphasize scalability, security, and interactivity through WebSockets [14].

3 Proposed Work

The work in question is focused on the development of a real-time chat application that enables users to communicate via private or group chats in a secure and seamless manner. The user interface of the system will be intuitive and incorporate sophisticated elements like sentiment analysis, real-time message delivery, and the ability to add functionality through third-party APIs integrated into the system [5]. The system's backend will be data secure, user authenticated, encrypted, and unwaveringly reliable, all while being capable of handling vast amounts of data. This multi-pronged approach means that system can effortlessly scale while maintaining a positive and engaging experience across varying environments. System conducts reliability and scalability tests in correspondence to growing user numbers. Steps of primary importance are:

3.1 Collecting Datasets and Pre-Processing

Fulfilling system objectives begins with acquiring relevant data, encompassing user profiles, chat history, and multimedia files. This data undergoes cleaning and organizing to ensure order and security. Data needs tailored approaches to facilitate easier access to system requirements at different stages of system development. Strong foundations are set during initial phases such as basic user interaction with structured chat interfaces. Users expect reliable navigations through various system components without security and smooth chat as primary concerns.

3.2 Backend Architecture and Real-time Messaging

The application’s backend will be powered by Django alongside the Django REST framework as it offers a robust and reliable framework for managing high volumes of requests [7]. WebSockets will be added for real time messaging which will allow messages to be sent and received by the users instantaneously without any waiting time [6]. Whether a user is participating in a one-on-one conversation or in a group, real-time messaging will make sure communication feels alive without ringing pauses [8]. This is the core foundation of the system which provides real time acceleration and efficiency in the operations of the application.

3.3 Real-time Sentiment Analysis

The ability to assess the sentiment of some messages as they are being typed is one of the most fascinating functions that comes with this chat system. The mood or tone of conversations will be detected using real time sentiment analysis and the chat interface will be modified accordingly [9]. For example, background colors can change depending on whether the conversation being analyzed is jovial, neutral, or tense. Beyond just personalizing chats, this advanced emotional intelligence will help users engage with the conversation in a novel manner enabling them to effortlessly sync with the mood of the discussion.

3.4 Security and User Authentication

Securing the system will be of utmost importance. Security will be placed at the highest priority. Strong authentication methods will guarantee that only valid users will access the system. In addition to this, securing the user's system, while protecting the private information of the users during transmission through the system with session-based encryption will defend the data from unauthorized usage [10]. Users will have the confidence that the platform remains trustworthy and secure for everyone while ensuring their private conversations and the information is protected.

3.5 Performance Optimization and Testing

After incorporating the primary building features, the system will undergo extensive checking to validate that it is optimally functional in case of high load usage [11]. Testing the back end for heavy data loads, in addition to checking the number of concurrent users the platform can serve without delays are all part of the thorough checkup. Additionally, the claiming provision of the system having real-time sentiment analysis ability is put to check, whether it is indeed fast and accurate without disturbing the user's seamless experience. All these performed helps enhance the reliability of the system irrespective of the number of users on the system.

3.6 Integration with Third-Party APIs

In order to give the chat system more features, we will use third-party APIs for push notifications, file share and multilingual support [12]. These integrations will provide enhanced functionalities such as sending real-time alerts to users, sharing documents or images seamlessly and even support for communication in multiple languages. This will in turn increase the versatility of the app and make it more attractive to a wider, global audience as it will cater not only to what one group of users may need but rather meet various needs for other groups around the world.

3.7 Scalability and deployment

The system will be designed with scalability in mind. As more people use the chat platform, the system needs to handle increased traffic without slowing down. Using cloud technology and load balancing, we’ll ensure that the platform can grow and manage more users over time [13]. We will also make sure that the app performs optimally on a range of devices and platforms, delivering a seamless experience whether it's accessed on phones, tablets, or computers [14]. The system will be deployed, with strong checks to ensure that everything is going smoothly once it's open to the public.

4 Architecture Diagram

The architecture diagram above illustrates the flow of a real-time chat system, informing how data moves between the user, client, server, and database. The fig 1 shows architecture diagram. Below is a more detailed and informed breakdown of the system components:

4.1 Left Side (User Engagement and Client)

4.1.1 User

The journey begins with the user. This is anyone on the platform who intends to either send a message, receive a message, interact in a group chat, or generally chat with other users on the platform. The user initiates every action to be performed in the system.

4.1.2 Polling Request (Client)

As the user executes any of the system's provided functionalities like message sending or checking for fresh notifications, a "Polling request" is created on the client side. The user's device (Client/Workstation) issues a request which, in essence, constitutes a Polling request. The user request consists of sending/ receiving messages, emitting/ fetching new notifications, and so forth. The client plays the role of an interface for the user.

4.1.3 Data Display to User

After the server processes the user requests and sends back the relevant data, the Client displays this data to the user ensuring that all information presented computes the required structured information. The Client shows users new messages, chat activity, and notifications in a user-friendly format designed to enhance understandability and promote interaction through intuitive interfaces.

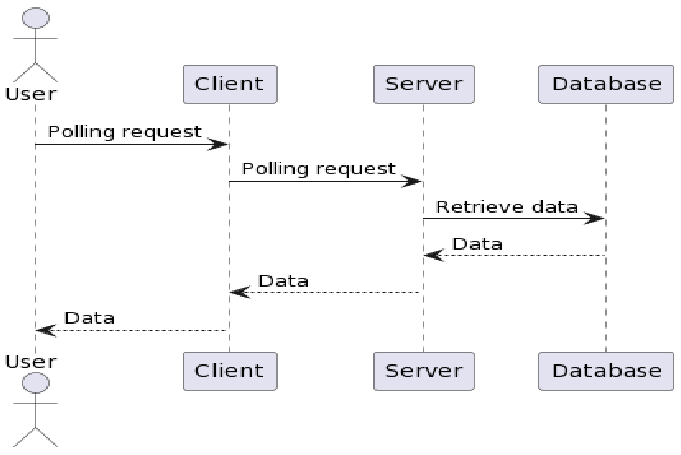


Fig. 1. Architecture diagram.

4.2 Right Side (System Workflow: Server & Database)

4.2.1 Database

Its central repository stores all information that is essential for functioning of the chat system, and this includes user profiles, messages, group chats, media files, and other information. The database can be seen as the heart of the system which stores all the crucial information as user profiles for messages, group chats, and other data are needed for the system to be functional.

4.2.2 Polling Request (Server)

The Polling request sent by the Client now reaches the Server. The Server serves as the heart or brain of the System, receiving each request and appropriately processing it. The Server manages operations that ensure adequate data flow from Client to the Database and vice versa. With the aid of the Database, the Server works out what information to gather or update and process requests in the system and issue the required data.

4.2.3 Data Retrieval

The data is requested from the Database Server. This may include fetching a user chat, retrieving a new message or even setting user properties. The Server governs the request and ensures requesting the correct data as needed be a personal message or a group chat update for the user.

4.2.4 Data Response

After the requested data has been retrieved from the database, it is captured by the Server. This could include a collection of messages, user information, or any other pertinent information. It is the responsibility of the Server to make sure this data is properly formatted before sending it off.

4.2.5 Data Forwarding

The Server dispatches the data after processing to the Client. In this stage, the server provides the requested information, such as displaying new messages, updating group chats, or user interfaces. The information is displayed with the write expectation such that The Client can update and reflect the underlying data on screen instantaneously.

4.2.6 Result

At the final step, the Client processes the data and presents it to the user in a structured manner such that it is comprehensible. This may be to display new messages in a chat block, to refresh the chatbox with the latest data, or to give pop-up notifications of an event. The information provided is now actionable for the user and continues the cycle of communication between users within the system with minimal response interaction delays.

5 Flow Diagram

The flow diagram for the chat application illustrates the user journey and how they interact with the core features of the app. The fig 2 shows Flow Diagram.

Start: The users are welcomed by the chat application with two options, either to log into their accounts or create new ones, guiding them towards streamlined navigation funnels. The user's journey starts when they open the app. Clearly confirming for each user, the step, they wish to take – whether accessing a personal account or setting up a new one.

Login / Signup: Upon launching the application, users are met with two pathways, Login for existing users and Signup for new users. For existing users, selecting Login allows them to enter the credentials, username/email and password, associated with their personal dashboard. New users are catered for by the Signup option which allows them to enter their name alongside a valid email address and password of choice. The Dashboard is the default destination following the successful completion of these steps.

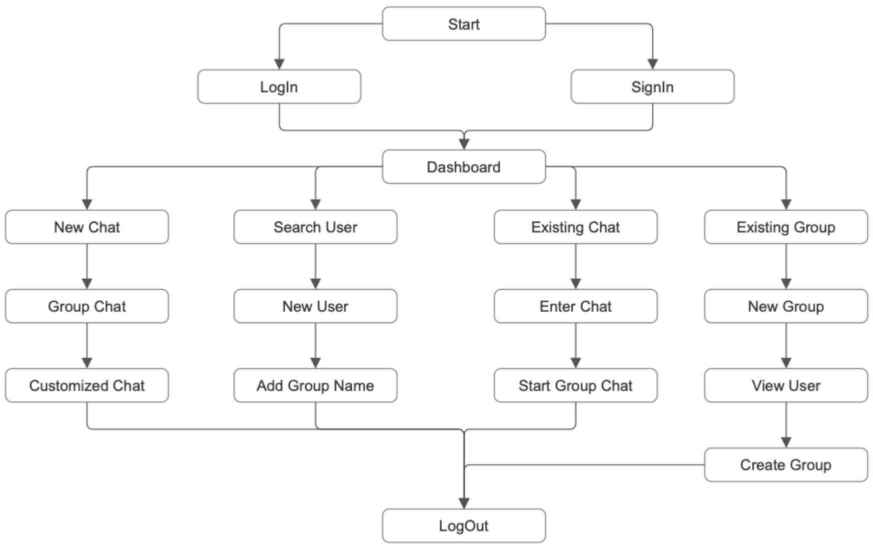


Fig. 2. Flow Diagram.

Dashboard: As part of the application, the Dashboard serves as the user's main control center. It is the main interface following a successful login or signup. It contains all essential tools you will need to interact with the features of the app in a simple and intuitive manner. From the dashboard, the user can start New Chats, search for other Users, open existing Conversations, and manage Group Chats. As a central hub, the dashboard is your primary navigational point serving all other functionalities. other tools the smooth and uncomplicated user experience results from the placement of all essential tools in a single location. The dashboard is clean in its simplicity, giving users a simple interface with clear pathways to explore the myriad features on offer. The dashboard works best when uncluttered.

New Chat: A new chat can be opened from the dashboard. Users can select a user and start a one- on-one conversation, create a group chat, and customize settings for the chat, including themes and notifications.

Search User: Users are of course able to find other people to connect within the platform. To do so, they will use the Search User function. New users can be invited and added to groups as well which can also be done through this page using name or username search.

Existing Chat: The Existing Chat option grants access and enables full revisited to the past conversations therefore making it easier review Important exchanges. When the user opts for Enter Chat, he/she is taken back to a particular chat thread and continue the chat as before. For more synergetic communications Users have the option of choosing Start Group Chat which may let them convert individual chat into group conversations. They can shift from the individual centered approaches to multi-person-centered approaches or vice versa depending on their needs. This part aids the users on how they can manage their still active conversations.

Existing Group: The Existing Group part deals with influencing and managing pre-group created group chats. If users oversee different groups, they can reach them from this section. If users are managing multiple groups, they can opt to New Group to add more users to formed groups or change settings of New Group Managed group. Through View User, users can profile check group members enabling deeper interaction within the group. If required users can also Create Group from the users, they view. This feature aids in forming new chats from users who already exist as contacts.

Log Out: The Log Out option allows users to securely exit the app, returning them to the login/signup screen. It ensures privacy and keeps their account secure.

6 Design and Implementation

6.1 Programming Languages

The backend of the conversation room system is built in Python and the frontend in JavaScript. Python was selected because of its strong web development capabilities (Django), real-time communication (WebSockets), and AI/ML model integration. The frontend interface is developed in JavaScript (React.js) to enhance responsiveness and interactivity.

6.2 Libraries

The additional libraries included in this system helps improve the system's efficiency and responsiveness. The backend is based on a Django 3 installation that integrates easily with MongoDB, providing scalable architecture for data storage and retrieval. It supports authentication, session management, and messaging services. Using React.js for frontend allows for implementation of a responsive UI that dynamically displays messages encouraging the users to customize the chat background. The users' data including chat histories, configuration of rooms, and user preferences are stored in unstructured databases such as MongoDB due to a lack of fixed schemas for message logs and mood data. WebSockets enables message delivery while chat participants are connected, Open client/server connections ensure real time message delivery by keeping the client and server connection open. Other analyzed sentiment TextBlob, NLTK, or spaCy are used to analyze user inputs whether text or voice to detect emotional tones and adjust the interface dynamically, providing a more personalized experience.

6.3 Frameworks

The application utilizes Django 3 for backend management and React.js for the frontend. Django is particularly well-suited for managing user authentication, session handling, and real-time interactions, which are the primary focus of this framework. It maintains the user database, allows group chats, and ensures effective communication between the client and server. React.js implements a real-responsive graphical user interface that changes during interactions and updates in real-time, which is responsive and adjusts seamlessly to the user's prevailing mood via mood-detection technology.

6.4 Data Management

The system handles different profiles of data such as system usernames, emails, session-related information, chat data comprising group chat messages alongside unique identifiers timestamps and text content, user preferences for customizable background images, graphical interface colors along with changes according to mood, and message history comprising logs of past conversations retrieval for reference purposes. All this information is managed in MongoDB which ensures scalability and high availability. In addition, WebSockets updates on dashboard Objects and real-time data to refresh Widgets. Real-time data from WebSockets is temporarily stored in in-memory caches like Redis, allowing for smooth performance and quick updates for active users.

6.5 Emotion detection framework interface

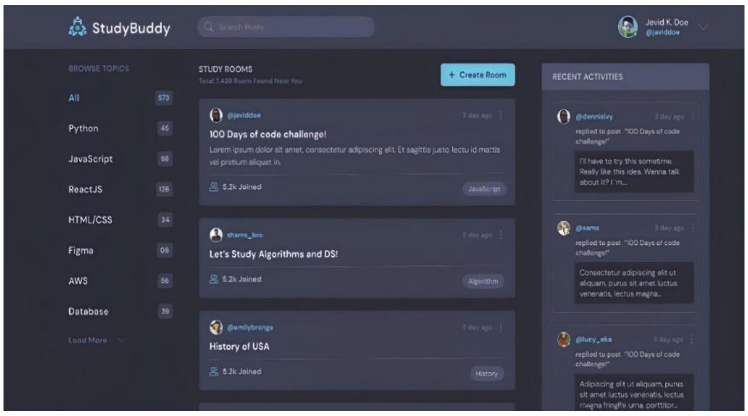
The system includes AI mood detection features that modify the user interface according to emotions derived from user input, be it by text or voice. With tools such as TextBlob or NLTK, the system analyzes messages or voice prompts to determine the user’s emotional state, like happy, sad, or angry. The user interface is further enhanced to reflect the commands issued based on the mood detected. For instance, the user’s emotional state can be determined from voice command and the background color, theme and other components altered to suit the mood. Colors display vibrant and cheerful background flora when the user responds positively, whereas in sad situations, the background transforms into a soothing picture. Further, depending on the user’s mood, the system can also change the tone of the automated responses where more soothing content is offered when frustration or sadness is detected.

6.6 Session Management

The system guarantees safe user management through an appropriately designed signup and login protocol. During account creation, users are required to provide an email and a password, which are authenticated through controlled processes using hashed passwords stored in MongoDB. Session management track active sessions through a log- in feature with time restrictions for manual or automated logouts after a specified period which balances security and privacy concerning personal user data.

6.7 Real time Messaging

The system offers a smooth real-time messaging experience through WebSockets, which makes sure that users can send an immersive chat experience, enabling users to chat freely while the interface responds to their emotional state. The group chat feature promotes social interaction, and the mood-based transitions improve user experience by making the interface more personal and empathetic. The integration of secure authentication, real-time messaging, and AI-driven features guarantees that users enjoy a seamless, delightful, and customized experience. The fig 3 shows Output.



@javiddoe 3 day ago
100 Days of code challenge!
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Et sagittis justo, lectu id mattis vel pretium aliquet in.
5.2k Joined JavaScript

shams_bro 3 day ago
Let's Study Algorithms and DS!
5.2k Joined Algorithm

@emilybronge 3 day ago
History of USA
5.2k Joined History

@dennisivy 3 day ago
replied to post "100 Days of code challenge!"
I'll have to try this sometime. Really like this idea. Wanna talk about it? I 'm...

@sams 3 day ago
replied to post "100 Days of code challenge!"
Consectetur adipiscing elit ut aliquam, purus sit amet luctus venenatis, lectus magna...

@lucy_aka 3 day ago
replied to post "100 Days of code challenge!"
Adipiscing elit ut aliquam, purus sit amet luctus venenatis, lectus magna fringilla urna, porttitor...

Fig. 3. Output.

7 Formula's

To accurately measure and analyze system performance, numerous various metrics to be measured are identified through mathematical equations. These equations will calculate message latency, system throughput, sentiment analysis, scalability, and load balancing performance. Through the utilization of these computations, system administrators and developers will be able to ensure they improve on performance, detect bottlenecks, and are able to sustain traffic. Below, are the primary formulas utilized to calculate these critical performance metrics.

7.1 Message Latency Calculation

Message latency is the delay between sending and receiving a message, which reflects the system's responsiveness. It is calculated as

$$L = T_r - T_s \tag{1}$$

Where:

- **T_r** = Timestamp when the message is received
- **T_s** = Timestamp when the message is sent. Latency means a faster system, while higher latency could signal network or processing delays.

7.2 System Throughput (MPS)

Throughput measures how many messages the system processes per second, indicating its capacity to handle traffic. The formula is:

$$Throughout = \frac{N}{T} \tag{2}$$

Where:

- N = Total messages processed
- T = Time in seconds

Higher throughput means the system is efficient and can manage a high volume of messages.

7.3 Sentiment Score Calculation

Sentiment score assesses the overall tone of a text. It’s calculated by summing the sentiment of individual words:

$$S = \sum_{i=1}^{n} w_i \times s_i \tag{3}$$

Where:

- w_i = Weight of word
- s_i = Sentiment polarity of word

A positive score means a positive tone, while a negative score indicates negativity. It’s often used in analyzing social media or customer feedback.

7.4 Scalability Performance (RPS)

Scalability performance is measured by Requests per Second (RPS), showing how many requests, a system can handle. The formula is:

$$RPS = \frac{Total\ Requests\ Processed}{Total\ Time} \tag{4}$$

A higher RPS means the system can scale efficiently to handle more requests.

7.5 Server Load Balancing Efficiency

This measure how well the system distributes workloads across servers. It is calculated as the average server utilization:

$$E = \frac{\sum_{i=1}^{n} U_i}{n} \tag{5}$$

Where:

- U_i = Utilization of each server
- n = Number of servers

A higher efficiency indicates an even load distribution, ensuring no single server is overwhelmed.

8 Table’s & Graph’s

Study both visual data analysis alongside structured data analysis to learn how a system performs. Users can observe trends over time which aids in understanding and predicting message latency, throughput, sentiment distribution, scalability, and load balancing. Graphs help visualize trends over time while tables provide a straightforward comparison of system indicators. Using both forms of data analysis assists in making informed decisions, improving system performance, and optimizing experiences for users.

8.1 Message Latency Graph

The graph measures the delay time in the system for sending a message and receiving a message back. This helps to assess responsiveness of the application. An ideal scenario depicts a smooth curve close to zero which indicates the system can process and route messages swiftly and the application is performing efficiently and out of the water with little delay. The Fig 4 shows Message Latency Graph. If there are fluctuations or spikes, this means there is network congestion or a delay at the server or routing and these should be addressed to ensure uninterrupted user experience.

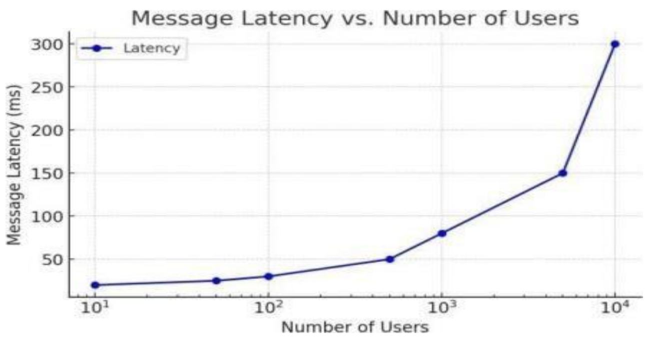


Fig. 4. Message Latency Graph.

8.2 System Throughput Chart

A system throughput chart captures system messages processed within a time frame. Throughput is an important indicator of performance. Higher throughput value improves system performance as messages are processed quickly. Stable, high throughput value indicates the best performance while low or declining throughput value suggests that the server is likely overloaded and suffers from resource concerns (low resources) or an inability to process messages quickly, bottleneck. The Fig 5 shows System Throughput Chart. Attention to the system will be needed to restore optimal performance.

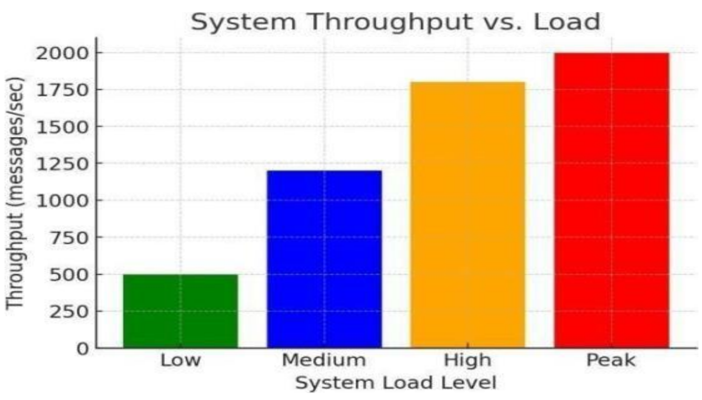


Fig. 5. System Throughput Chart.

8.3 Sentiment Score Distribution Graph

The graph depicts the computed sentiment of messages which were categorized based on range from negative (-1) to positive (1). It depicts how the average emotional sentiment of users is in comparison to other users on the platform. A balanced sentiment score distribution across the scale shows a wide range of varying opinions relative to user's interactions indicating unsaturated or neutral engagement. A heavily skewed result may illustrate overall trend sentiment. The fig 6 shows Sentiment Score Distribution Graph. This proved beneficial in gauging user satisfaction, identifying product issues, or detecting emerging user trends.

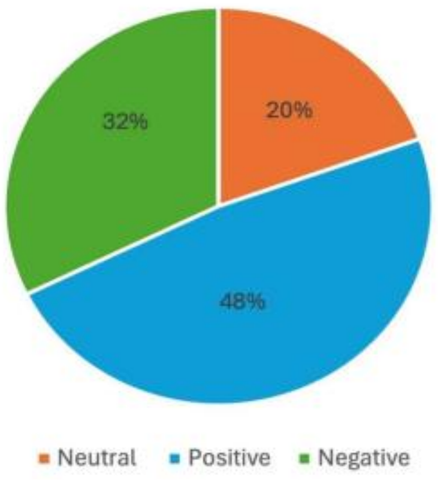


Fig. 6. Sentiment Score Distribution Graph.

8.4 Scalability Performance Graph

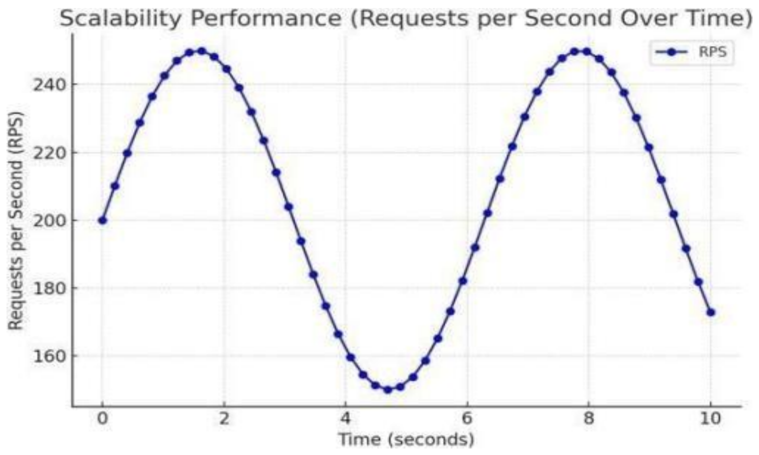


Fig. 7. Scalability Performance Graph.

This shows how well the system scales with an increase in demand capturing requests per second (RPS). If the graph is flat or has a positive slope, that indicates the system scaled well and continues to maintain performance while servicing additional requests. If the graph shows drop and fluctuations, the system might not scale properly, which uncovers some bottlenecks, limits on the server, or other tuned infrastructure components that are preventing optimal performance during high demand periods. The Fig 7 shows Scalability Performance Graph.

8.5 Server Load Balancing Efficiency Graph

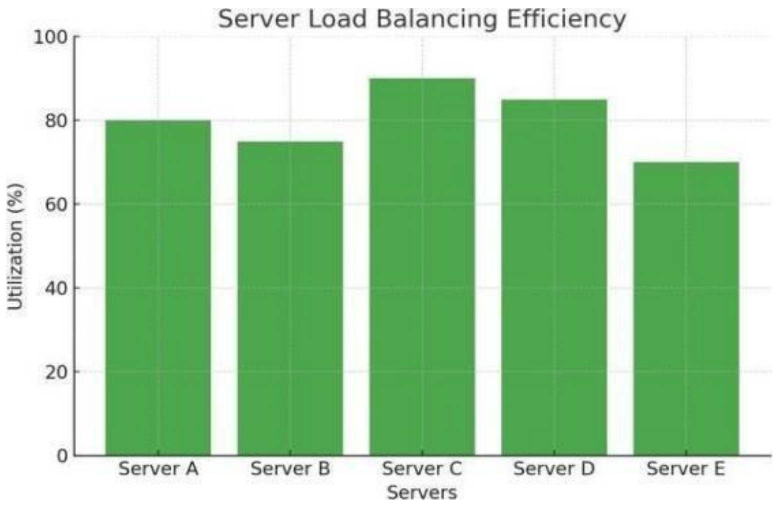


Fig. 8. Server Load Balancing Efficiency Graph.

This graph shows the system's workload efficiency relating to how workloads are allocated to its servers in a dynamic way. Even distribution of load helps all the servers to operate at their full capacity which improves the overall system performance and decreases the possibility to overburden an individual server. If the load is illustrated with large discrepancies in the graph, some servers are overloaded, and some are quiet that means there is load balancing inefficiency. This kind of imbalance can create slower system response times and increased downtime, indicating that an optimization strategy for resource distribution would provide significant improvements to overall efficiency. The fig 8 Server Load Balancing Efficiency Graph.

8.6 Comparison of Existing vs. Proposed System

This table describes a comparison of the conventional chat systems, and their evolution based on recommendations. In traditional systems, message delivery is done through polling; the suggested system uses WebSockets for real-time communication. Security changes from simple authentication via messaging to secret secured methods. In the new system, sentiment analysis is included for real-time mood detection which is absent in conventional systems. Scalability includes improved load balancing, support for multimedia files, images and voice, as well as multi-linguistic third-party API integration for push notifications add enhanced scalability. The table 1 shows Comparison of Existing vs. Proposed System Table.

Table 1. Comparison of Existing vs. Proposed System Table.

Feature	Traditional Chat System	Proposed System
Message Delivery Method	Polling	WebSockets (Real-time)
Security Mechanism	Basic Authentication	Encrypted & Secure Authentication
Sentiment Analysis	No	Yes (Real-time Mood Detection)
Scalability	Low	High (Load Balancing)
Multimedia Support	Limited	Extensive (Files, Images, Voice)

8.7 Performance Benchmarking

This table compares the performance and showcases great optimizations. Message latency dropped significantly from 200-500 ms to a much more refined 20-100 ms which translates to faster communication. throughput raised from 1,000 messages to 5,000 messages per second which demonstrates increased ability to handle load consistently. Sentiment analysis contradicts traditional systems by showing accuracy levels of 85-95% predicting sentiments which were previously absent. Server response times also improved considerably, dropping from 150-300 ms to a more responsive 50-120. The table 2 shows Performance Benchmarking Table.

Table 2. Performance Benchmarking Table.

Metric	Value Traditional	Value Proposed
Message Latency (ms)	200-500	20-100
Throughput (Messages/sec)	1000	5000
Sentiment Analysis Accuracy	N/A	85-95%
Server Response Time (ms)	150-300	50-120

9 Future Work

Future improvements include the addition of the AI chatbots which assist us in answering queries and supports questions which will lighten the work of human agents as well workers. There is an aim to include the feature of translating languages in real time which means the system will be able to provide all language translations and connect to an audience all over the world simultaneously. The chat will also be able to be embedded in websites and mobile apps and will go as far as integrating with well-known messaging applications which include WhatsApp, Facebook Messenger, and Slack. Meaning further customization options will be able to be provided such with customizable chat widgets, user profiles, and rich media including images, videos, and file attachments. They will also include sentiment analysis which means they will be able to evaluate the user's emotions in real time when messages are being received and even change the background of the chat room to the user's mood automatically. There are also plans of developing an administrator dashboard that will allow monitoring of the chats with the user satisfaction level and agent productivity which will allow insightful decision-making in real time.

10 Conclusion

The analysis results indicate that the chat system could be improved considerably with the application of more sophisticated AI and real-time technologies. With the implementation of AI chatbots, sentiment analysis, language translation, and other features that support rich media, the system is expected to provide an immersive, customized, and optimal communication experience. The capacity to modulate in real time the emotive tone of the user's messages adds an empathetic dimension to the exchanges, whereas real-time control of data dashboards by an administrator adds value to performance and satisfaction. The planned integration with the most preferred messaging systems, together with the ease of embedding the system in web pages and mobile applications, widens the system's flexibility, reach, and applicability, making it adaptable for international interactions. Enhancing these features will not only streamline interaction for users but will also further the multi-layered objective of improving customer care and user interactions as the system develops. Perpetual growth in sentiment analysis technology, and cross-platform integration will enable businesses to easily forge powerful connections with users across the globe through the chat system.

References

[1] Sathya, S., & Swetha, S. (2025). Real-time chat application with Web Socket for network efficiency. *International Journal of Scientific Research in Science and Technology, 12*(4), 830–835. https://doi.org/10.32628/IJSRST251362

[2] Jadon, S., Singh, P., & Singh, R. (2025). Real-time chat application development using MERN stack. International Journal for Research in Applied Science & Engineering Technology (IJRASET), 13(4), 365–370. https://doi.org/10.22214/ijraset.2025.68294

[3] Liu, Q., & Sun, X. (2012). Research of Web real-time communication based on WebSocket. International Journal of Communications, Network and System Sciences, 5(12), 797–801. https://doi.org/10.4236/ijcns.2012.512083

[4] Muller, G. L. (2014). HTML5 WebSocket protocol and its application to distributed computing. arXiv preprint arXiv:1409.3367. https://arxiv.org/abs/1409.3367

[5] Behnke, I., & Austad, H. (2023). Real-time performance of industrial IoT communication technologies: A review. arXiv preprint arXiv:2311.08852. https://arxiv.org/abs/2311.08852

[6] Ghasemshirazi, S., & Heydarabadi, P. (2021). Exploring the attack surface of WebSocket. arXiv preprint arXiv:2104.05324. https://arxiv.org/abs/2104.05324

[7] Dubey, A. (2023, August). Enhancing real time communication and efficiency with Websocket. *International Research Journal of Engineering and Technology (IRJET), 10*(8). https://www.irjet.net

[8] Dhanesh, G. L., & Praveen, K. S. (2024, July 23). *Real-time chat app*. *International Research Journal of Modernization in Engineering Technology and Science*. https://www.irjmets.com

[9] Sagar, A. (2024, April 16). *Real time chat application using React and Firebase*. *International Journal for Research in Applied Science and Engineering Technology (IJRASET)*. https://doi.org/10.22214/ijraset.2024.60475

[10] Ogundeyi, K. E., & Yinka-Banjo, C. O. (2019). WebSocket in real-time application. Nigerian Journal of Technology, 38(4), 1010–1020. https://doi.org/10.4314/njt.v38i4.26

[11] Tadvi, B., Parmar, B., & Gupta, S. (2025). A literature review: Next-gen React chat applications: Enhancing real-time communication. International Journal of Innovative Science & Research Technology, 10(3), 2056–2060. https://doi.org/10.38124/ijisrt/25mar1476

[12] Murley, P. (2021). WebSocket adoption and the landscape of the real-time web. In Proceedings of the Web Conference 2021 (WWW '21) (pp. 1234–1245). Association for Computing Machinery. https://faculty.cc.gatech.edu/~mbailey/publications/www21_websocket.pdf

[13] Mao, S., Xu, C., Wang, L., Xu, J., & Zhang, Y. (2019). Reinforcement learning for bandwidth estimation and congestion control in real-time communications. arXiv preprint arXiv:1912.02222. https://arxiv.org/abs/1912.02222

[14] Zhang, J., Wang, X., Wang, Y., Lin, C., & Hanzo, L. (2017). A survey on low latency towards 5G: RAN, core network and caching solutions. arXiv preprint arXiv:1708.02562. https://arxiv.org/abs/1708.02562