ISSN 2277-2685
IJESR/July-September. 2026/ Vol-16/Issue-3/170-178
Y Bharathi *et. al.*, / International Journal of Engineering & Science Research
Full Length Article

Real Time Multi-User Chat System Using TCP

S Manjula[1], Y Bharathi[2], P Manusha[3], A Nihitha[4]
[1]Associate Professor: Department Of Electronics And Communication Engineering, Bhoj Reddy Engineering College For Women Hyderabad India.
[2,3,4]B. Tech Students; Department Of Electronics And Communication Engineering, Bhoj Reddy Engineering College For Women Hyderabad India.
Mail Id's; ybharathireddy14@gmail.com[2], manushapaka01@gmail.com[3], anikeshnihitha8727@gmail.com[4]
Accepted 27-06-2026

Abstract
The Real-Time Multi-User Chat System is a web-based communication application developed using Python, Flask, and Socket.IO to provide seamless and instant messaging among multiple users. The system supports both public and private messaging, allowing users to communicate with all participants or selected individuals in a secure and efficient environment. It incorporates user authentication features, including registration and login, with password hashing to ensure secure access and protect user credentials. By utilizing WebSocket-based communication through Socket.IO, the application enables real-time message delivery with minimal latency, overcoming the limitations of traditional request-response communication models. In addition to text messaging, the system supports file sharing, allowing users to exchange documents and other files during conversations. Chat-related information, including messages, timestamps, user details, and file references, is stored in a SQLite database to ensure data persistence and efficient retrieval. The user interface is designed with a modern chat layout, incorporating features such as message timestamps, date-wise conversation grouping, and an intuitive messaging interface to enhance the user experience. This project demonstrates the practical implementation of real-time communication, event-driven programming, WebSocket technology, database management, and secure web application development, making it suitable for applications such as team collaboration platforms, instant messaging services, educational communication systems, and enterprise communication solutions.

Keywords- *Real-Time Chat System, Python, Flask, Socket.IO, WebSocket, Multi-User Communication, Public and Private Messaging, User Authentication, Password Hashing, SQLite Database, File Sharing, Real-Time Messaging, Event-Driven Programming, Web Application Development, TCP-Based Communication.*

Introduction

The rapid advancement of information and communication technologies has transformed the way individuals and organizations exchange information. With the widespread availability of high-speed internet and smart devices, users increasingly expect communication platforms that provide instant, reliable, and uninterrupted interaction. Real-time messaging applications have therefore become indispensable in various domains, including business collaboration, online education, customer support, healthcare, and social networking. These applications facilitate immediate information exchange, improve collaboration, and enhance user engagement by eliminating

Security represents a fundamental requirement in any communication platform. Unauthorized access to messaging services may compromise user privacy and system integrity. To address this challenge, the proposed system incorporates a secure authentication mechanism that requires users to register before accessing the platform. User credentials are protected using password hashing techniques, ensuring that sensitive information is securely stored within the database. After successful authentication, users can participate in public discussions, initiate private conversations, and securely exchange information with other registered users.

The application supports concurrent communication among multiple users through an event-driven architecture that efficiently manages user activities and message transmission. Socket.IO enables asynchronous event handling, allowing the server to process multiple client requests simultaneously without degrading system performance. This architecture enhances scalability and ensures smooth operation even under increased user activity, making the platform suitable for collaborative environments.

Beyond conventional text messaging, the proposed system includes integrated file-sharing functionality that enables users to exchange documents and multimedia files during conversations. Uploaded files are securely stored on the server, while corresponding references are maintained within the database for efficient retrieval. This capability significantly expands the practical applicability of the system by supporting collaborative document

sharing and information exchange in academic, professional, and organizational environments.

To ensure persistent data storage, SQLite has been adopted as the database management system. The database maintains user profiles, authentication details, chat messages, timestamps, and file references, allowing previous conversations to be retrieved whenever users reconnect to the application. Persistent message storage improves usability by preserving communication history and enabling continuous interaction across multiple sessions.

Considerable attention has also been given to user interface design. The chat interface follows the design principles adopted by modern messaging platforms, incorporating message alignment, chronological ordering, timestamps, date-based conversation grouping, and responsive layouts. These features simplify navigation, improve readability, and provide users with an intuitive communication experience across different devices and display resolutions.

Literature Survey

The growing demand for real-time digital communication has motivated extensive research on web-based messaging technologies, communication protocols, and distributed application architectures. Numerous studies have focused on improving communication efficiency, minimizing transmission latency, enhancing scalability, and strengthening security within modern messaging systems. This section reviews significant contributions related to communication protocols, real-time web technologies, event-driven architectures, and database management that collectively provide the foundation for the proposed Real-Time Multi-User Chat System.

The conceptual basis of network communication was established through the Open Systems Interconnection (OSI) reference model introduced by Zimmermann, which standardized communication across heterogeneous computer networks. The layered architecture proposed in the OSI model significantly influenced the design of modern networking protocols by defining independent communication functions at each layer. Although the OSI model primarily addressed reliable data transmission and interoperability, it did not specifically address the low-latency requirements of contemporary real-time communication applications.

The introduction of the WebSocket protocol by Ian Hickson represented a major advancement in web communication technology. Unlike conventional HTTP communication, which requires repeated request-response exchanges, WebSocket establishes a persistent bidirectional communication channel between the client and the server. This persistent connection enables efficient real-time data exchange with minimal overhead, making WebSocket an ideal technology for applications such as online messaging, multiplayer gaming, collaborative editing, and financial trading systems. The proposed chat application adopts this communication model to achieve instantaneous message delivery.

Research conducted by Miguel Grinberg demonstrated the effectiveness of integrating Flask with Socket.IO for developing lightweight real-time web applications. His implementation illustrated how Python-based frameworks could support event-driven communication while maintaining simplicity and development flexibility. Although the presented implementation successfully enabled basic messaging functionality, additional features such as secure authentication, private conversations, persistent message storage, and file sharing remained outside its scope. The proposed system extends this framework by incorporating these essential capabilities within a unified communication platform.

Network programming principles described by Fall and Stevens emphasized the importance of socket-based communication for reliable client-server interaction. Their work provided comprehensive insights into connection management, concurrency, and efficient handling of multiple client requests within distributed systems. Rather than implementing low-level socket programming directly, the proposed system leverages the high-level abstractions provided by Flask-SocketIO to simplify development while preserving communication efficiency and reliability.

The proposed Real-Time Multi-User Chat System addresses these challenges by integrating Flask, Flask-SocketIO, WebSocket communication, and SQLite within a lightweight yet comprehensive architecture. The application supports secure user authentication, simultaneous multi-user communication, public and private messaging, persistent message storage, and file sharing through a responsive web interface. Its modular design simplifies deployment while maintaining efficient real-time performance, making it well suited for academic research, collaborative learning environments, institutional communication, and small-to-medium-scale organizational applications.

System Design of Real Time Multi-User Chat System Using TCP

Existing System

Existing communication platforms, including electronic mail, Short Message Service (SMS), and conventional web-based chat applications, have provided effective mechanisms for exchanging information across computer networks. Although these technologies have contributed significantly to digital communication, they are not fully capable of satisfying the requirements of modern real-time interactive applications. Traditional messaging

systems primarily rely on the HTTP request-response model or store-and-forward communication techniques, which introduce transmission delays and reduce the immediacy of user interaction. Consequently, these approaches are less suitable for applications where instantaneous message delivery and continuous connectivity are essential.

Many existing chat applications also provide limited support for simultaneous multi-user communication. Several systems are designed primarily for one-to-one messaging and offer only basic group communication capabilities, restricting their applicability in collaborative environments such as virtual classrooms, project teams, and enterprise communication platforms. Furthermore, private messaging functionality is often implemented with limited flexibility, making selective communication among users less efficient.

Another drawback observed in conventional communication systems is the inadequate integration of file-sharing capabilities. While advanced commercial platforms support multimedia exchange, many lightweight chat applications either lack this functionality or depend on external storage services and additional software components. Such limitations reduce the effectiveness of the communication platform, particularly in scenarios requiring the exchange of documents, images, or other digital resources during conversations.

Proposed System

To address the limitations identified in existing communication platforms, a Real-Time Multi-User Chat System Using TCP is proposed. The application has been developed using the Python programming language, Flask web framework, Flask-SocketIO library, and SQLite database to provide an integrated environment for secure and instantaneous communication. The system utilizes WebSocket technology to establish persistent bidirectional communication channels between clients and the server, thereby eliminating the latency associated with repeated HTTP requests and enabling real-time message exchange.

The proposed platform supports simultaneous communication among multiple users through both public and private messaging channels. User authentication mechanisms, including secure registration, login verification, password hashing, and session management, are incorporated to ensure authorized system access and protect sensitive user information. The application further enhances communication by providing integrated file-sharing functionality, allowing users to exchange documents and multimedia resources directly within the chat environment.

SQLite is employed as the backend database to maintain user profiles, authentication records, conversation history, timestamps, and file references. Persistent data storage enables users to retrieve previous conversations after reconnecting to the system, thereby improving continuity and usability. The overall architecture is lightweight, scalable, and easy to deploy, making it suitable for educational institutions, research laboratories, collaborative teams, and small-to-medium-scale organizations requiring efficient real-time communication services.

Block Diagram

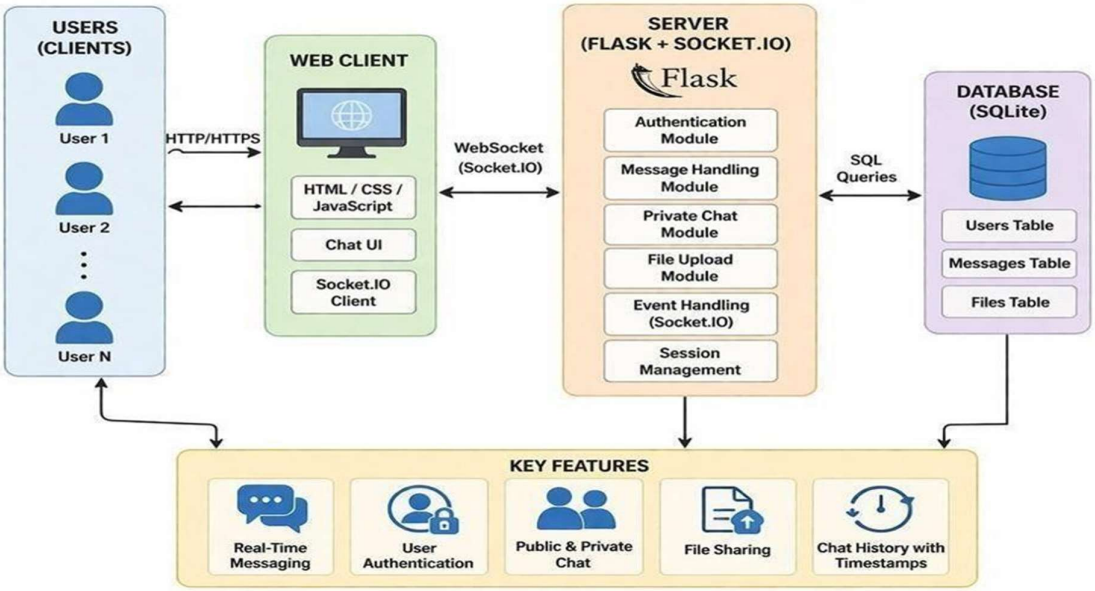

Figure 1: Block Diagram of Real Time Multi-User Chat System Using TCP

The architectural design of the proposed Real-Time Multi-User Chat System follows a layered client-server model that integrates frontend technologies, backend processing, real-time communication services, and persistent data storage. The operational workflow begins when users access the application through a standard web browser. New users complete the registration process, while existing users authenticate themselves using secure login credentials. Initial interactions, including authentication, page requests, and session establishment, are handled through standard HTTP/HTTPS communication between the client and the Flask server.

Following successful authentication, the system establishes a persistent WebSocket connection using the Socket.IO framework. Unlike conventional HTTP communication, this persistent connection enables continuous bidirectional data transfer between the client and the server, allowing messages to be delivered instantly without repeated connection establishment. The Flask server functions as the central processing unit of the application by managing client requests, maintaining user sessions, coordinating message routing, and controlling application logic.

The authentication module validates user credentials using password hashing techniques and ensures secure access throughout each communication session. After authentication, the message processing module manages the transmission, formatting, timestamp generation, and delivery of chat messages. Public messages are broadcast to all connected users, whereas private messages are transmitted only to authorized recipients through dedicated communication rooms created dynamically by Socket.IO.

The file-sharing module enables users to upload documents, images, and other digital resources. Uploaded files are securely stored on the server, while corresponding file references are maintained within the database for efficient retrieval during future communication sessions. SQLite serves as the persistent storage component by maintaining user accounts, chat history, timestamps, and uploaded file metadata. Whenever users reconnect to the application, historical conversations are retrieved from the database and displayed chronologically with appropriate timestamps and date-based grouping. This integrated workflow ensures reliable communication, efficient resource management, secure information exchange, and a responsive user experience.

Methodology

The development methodology of the proposed Real-Time Multi-User Chat System follows a structured client-server architecture designed to support efficient, secure, and low-latency communication over TCP-based networks. The implementation integrates Python, Flask, Flask-SocketIO, HTML, CSS, JavaScript, and SQLite into a unified software framework capable of supporting simultaneous communication among multiple users.

The operational process begins with user registration and authentication. During registration, user credentials are securely processed using password hashing algorithms before being stored in the SQLite database. Registered users subsequently authenticate themselves through the login module, where credential verification and session management ensure secure system access. Upon successful authentication, the server establishes a persistent WebSocket connection with each client using Socket.IO, enabling uninterrupted bidirectional communication throughout the user session.

Communication within the application follows an event-driven programming model. User activities such as message transmission, message reception, file uploads, user connections, and user disconnections are treated as independent events processed asynchronously by the Flask server. This architecture minimizes communication latency, supports concurrent client connections efficiently, and improves overall system scalability.

The messaging subsystem supports both public and private communication. Public messages are immediately broadcast to all connected clients, while private messages are routed through dedicated communication rooms to ensure confidentiality. Every transmitted message is timestamped and permanently stored within the SQLite database together with relevant user information. Similarly, uploaded files are securely stored on the server, and corresponding metadata are recorded within the database to facilitate subsequent retrieval.

Software Requirements

The development of the proposed Real-Time Multi-User Chat System Using TCP requires a software environment that supports efficient coding, debugging, testing, and deployment of web applications. To achieve these objectives, Visual Studio Code (VS Code) was selected as the primary integrated development environment (IDE) because of its lightweight architecture, extensive extension support, and compatibility with modern web development technologies. The application was implemented using the Python programming language, while Flask served as the backend web framework. Real-time communication functionality was enabled through the Flask-SocketIO library, with HTML, CSS, and JavaScript employed for designing the client-side interface. SQLite was utilized as the backend database to maintain user credentials, chat history, timestamps, and file metadata. Together, these software components provide a stable, flexible, and efficient development environment capable of supporting the

implementation of secure real-time communication services.

Visual Studio Code offers a comprehensive set of development features that significantly improve programming productivity and software quality. Syntax highlighting and intelligent code completion facilitate rapid software development by assisting developers with language constructs, library functions, and module imports. Integrated debugging capabilities enable efficient identification and correction of programming errors through breakpoint management, variable inspection, and stepwise execution. Furthermore, the built-in terminal allows developers to execute Python scripts, install software dependencies using the Python package manager (pip), and manage virtual environments directly within the development workspace. These capabilities streamline the overall development process by eliminating the need for external command-line interfaces.

The extensible architecture of VS Code further enhances its suitability for web application development. The Python and Pylance extensions provide advanced language support, including static code analysis, intelligent suggestions, and improved debugging performance. Additional extensions such as Flask Snippets, HTML CSS Support, and JavaScript language tools simplify frontend and backend development by offering syntax assistance, code templates, and automatic formatting. Integrated Git support enables efficient version control, allowing developers to monitor source code modifications, maintain project history, and collaborate effectively during software development. Despite providing extensive functionality, VS Code remains lightweight and consumes comparatively fewer system resources than many traditional integrated development environments, making it suitable for execution on a wide range of hardware platforms.

The development workflow begins by creating a dedicated project workspace within VS Code, where application files are organized into structured directories. The project follows a modular organization consisting of backend source files, frontend templates, static resources, and configuration components. The Explorer panel facilitates efficient navigation among project files, while the global search functionality enables developers to locate source code elements quickly. Such organization improves software maintainability and simplifies future enhancements to the application.

Development and testing activities are performed through the integrated terminal provided by VS Code. Required software libraries, including Flask, Flask-SocketIO, and Flask-Login, are installed using the Python package manager before application execution. A dedicated Python virtual environment is created and activated to isolate project dependencies and maintain consistency across different development systems. Once the required packages are installed, the application is launched by executing the main Python program, which initializes the Flask server, establishes Socket.IO communication services, and prepares the web application for client connections. This integrated workflow significantly reduces development complexity while ensuring efficient implementation, testing, and deployment of the proposed real-time multi-user chat system.

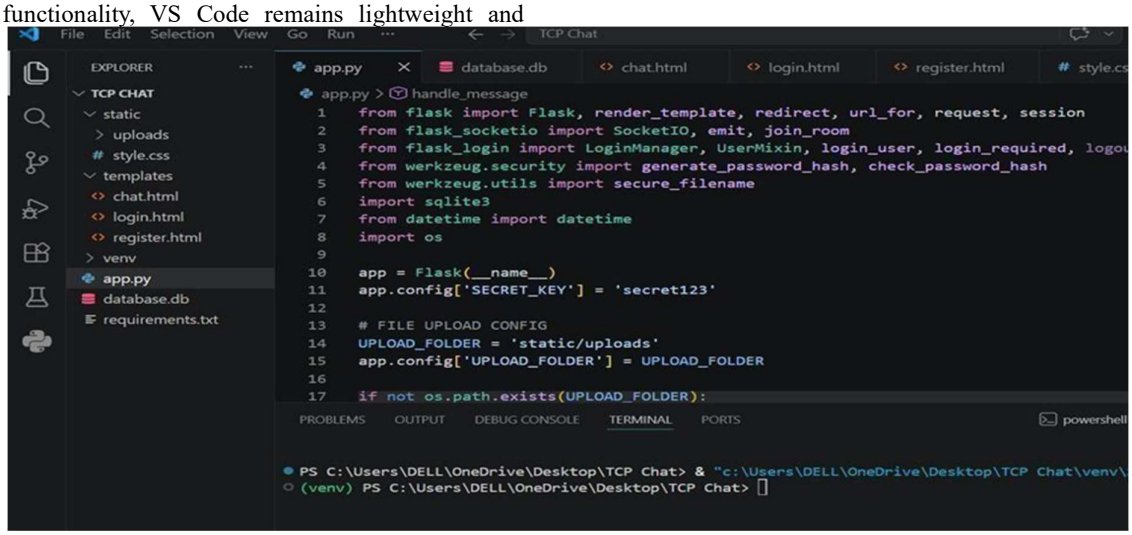

Figure 2 : VS Code Virtual Environment Window

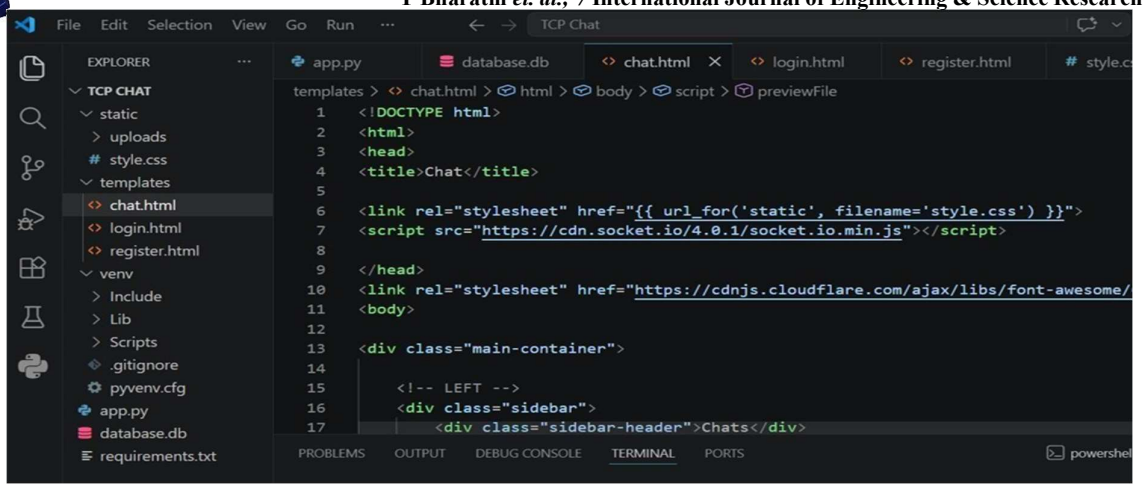

Figure 3 : VS Code Folder Directories

Figure 2 illustrates the Visual Studio Code development environment with the configured Python virtual environment used during implementation, while **Figure 3** presents the project directory structure, showing the organization of application modules, templates, static resources, and configuration files. These development tools collectively provide a reliable software platform for implementing, debugging, and maintaining the proposed Real-Time Multi-User Chat System.

Results

The performance of the proposed Real-Time Multi-User Chat System Using TCP was evaluated through a series of functional and performance experiments to assess its effectiveness in supporting real-time communication. The application was implemented using the Flask framework, Flask-SocketIO, and SQLite, and executed on a workstation equipped with an Intel Core i5 processor, 8 GB RAM, and the Windows 10 operating system. The evaluation focused on several key performance indicators, including message transmission latency, system responsiveness, concurrent user handling, file-sharing capability, database performance, and overall usability.

Experimental observations demonstrated that the proposed system successfully achieved real-time communication through the integration of WebSocket technology. By maintaining persistent bidirectional connections between the client and server, messages were transmitted with negligible latency, resulting in nearly instantaneous communication among connected users. The event-driven communication mechanism provided by Socket.IO eliminated the delays associated with conventional HTTP request-response communication, thereby ensuring continuous and efficient information exchange throughout the communication session. The multi-user communication capability of the application was evaluated by allowing several users to access the platform simultaneously. During concurrent operation, the server efficiently managed multiple client connections without experiencing noticeable performance degradation, message duplication, or transmission failures. Public communication was implemented through message broadcasting, enabling a single message to be delivered simultaneously to every active participant connected to the system. In contrast, private communication was achieved through dedicated communication rooms, ensuring that confidential messages were accessible only to their intended recipients. The successful execution of both communication modes confirms the reliability of the implemented message-routing mechanism.

The integrated file-sharing module was also evaluated during experimental testing. Users successfully uploaded and exchanged various file types, including documents and images, without interrupting ongoing communication sessions. Uploaded files were securely stored on the server, while corresponding metadata and references were maintained within the SQLite database. The files were correctly displayed within the chat interface, demonstrating seamless integration between the communication module and the storage subsystem. This functionality extends the practical applicability of the proposed system beyond conventional text-based messaging by supporting collaborative document exchange. Database performance was assessed through continuous storage and retrieval of user information, chat messages, timestamps, and file references. SQLite efficiently maintained communication records throughout the testing process, allowing historical conversations to be retrieved accurately whenever users reconnected to the application. Chronological organization of messages using timestamps and date-based grouping significantly improved conversation tracking and overall usability while demonstrating the reliability of the database management component.

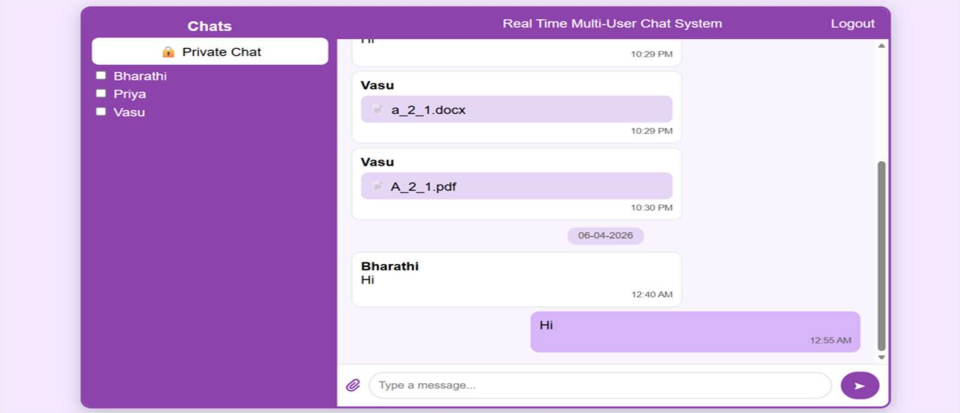

Figure 4 : Public Chat – Message Sent by User to All Users

illustrates the implementation of the public messaging mechanism. In this scenario, a user transmits a message without selecting any specific recipient, causing the application to automatically broadcast the message to all active users connected to the chat server. The interface displays the transmitted message together with its corresponding timestamp, confirming successful real-time message dissemination and validating the effectiveness of the broadcast communication mechanism.

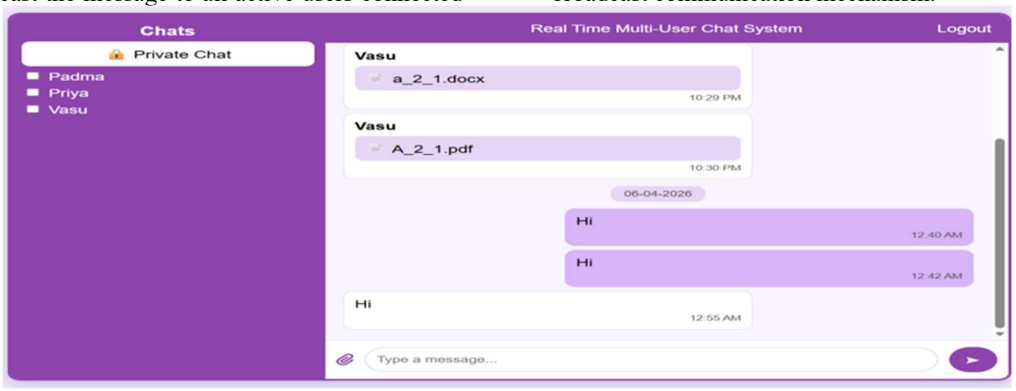

Figure 5 : Public Chat – Message Received by All Users

presents the reception of the broadcast message by multiple connected users. The identical message appears simultaneously within each user's chat interface, accompanied by sender identification and accurate timestamps. The consistent presentation of messages across multiple client sessions demonstrates reliable synchronization between the server and connected users while confirming the capability of the system to support efficient multi-user communication.

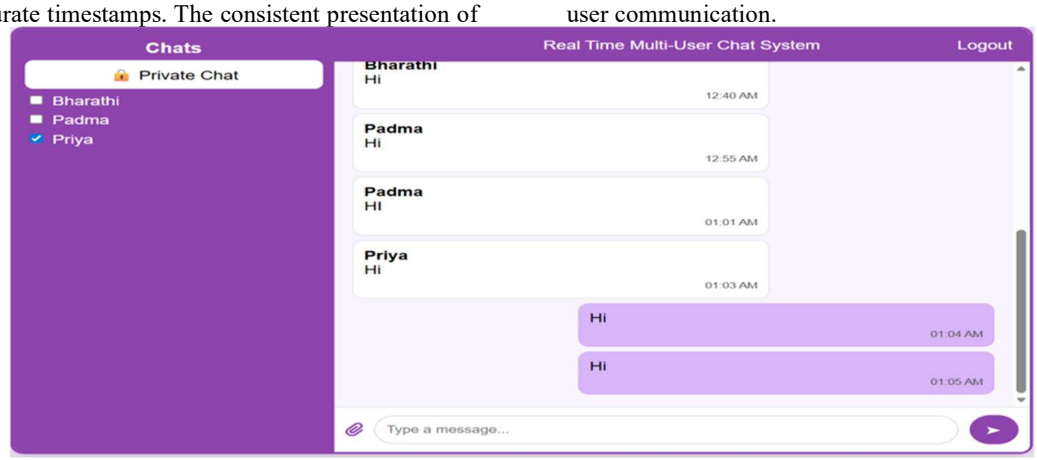

Figure 6 : Private Chat – Message Sent to Selected User

demonstrates the private messaging functionality implemented within the proposed application. In this experiment, the sender selects a specific recipient before transmitting the message. The server routes the communication through a dedicated Socket.IO room, ensuring that the message is delivered exclusively to the intended recipient. Other connected users remain unable to access the

transmitted content, thereby preserving communication privacy and validating the correctness of the selective message delivery mechanism.

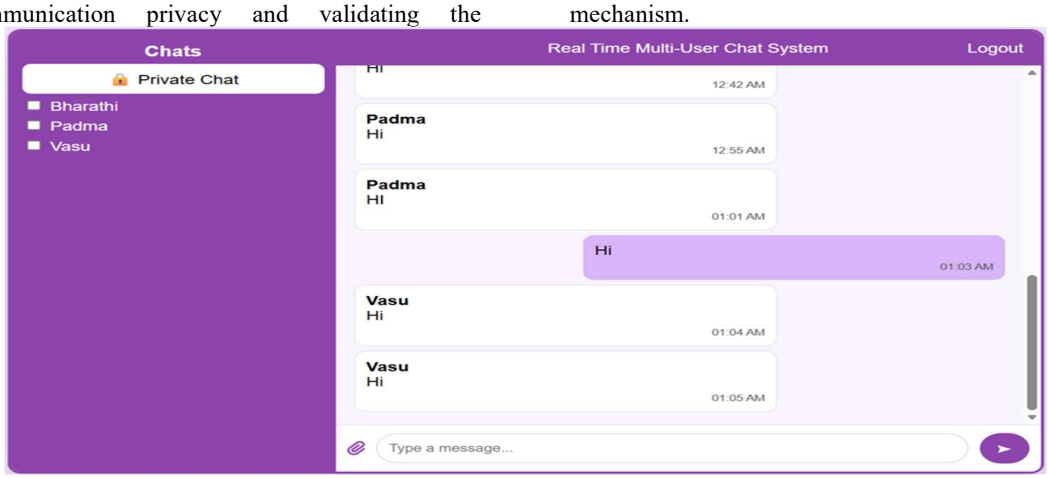

Figure 7 : Private Chat – Message Sent to Selected user

further validates the operation of the private communication module by illustrating continuous message exchange between selected users. The communication remains restricted to authorized participants while maintaining real-time responsiveness, accurate timestamp generation, and consistent message formatting. These observations confirm the effectiveness of the room-based communication architecture implemented within the application.

Conclusion

This research presented the design and implementation of a Real-Time Multi-User Chat System Using TCP, developed to provide secure, efficient, and low-latency communication through modern web technologies. The proposed application successfully integrates the Flask web framework, Flask-SocketIO, WebSocket communication, and SQLite database management to establish a reliable platform capable of supporting simultaneous interactions among multiple users. The adoption of persistent WebSocket connections enables instantaneous bidirectional communication, significantly reducing the latency commonly associated with conventional HTTP-based messaging systems and ensuring smooth real-time information exchange.

The implemented system effectively supports both public and private communication modes, allowing users to participate in group discussions or engage in confidential one-to-one conversations according to their communication requirements. Secure user authentication, password hashing, and session management mechanisms contribute to protecting user credentials and maintaining the integrity of the communication environment. Furthermore, the integration of file-sharing functionality enables users to exchange documents and multimedia resources directly within the application, thereby extending its practical utility beyond conventional text-based messaging.

The SQLite database provides reliable storage of user information, chat history, timestamps, and shared file references, ensuring data persistence and enabling users to retrieve previous conversations whenever required. Experimental evaluation demonstrated that the proposed application maintains high responsiveness while supporting multiple concurrent users without noticeable performance degradation or message loss. Features such as chronological message organization, timestamps, date-wise grouping, and an intuitive graphical interface contribute to an improved user experience and facilitate efficient communication.

Future Scope

Although the proposed system provides a comprehensive platform for real-time communication, several enhancements can be incorporated to further improve its functionality, security, scalability, and user experience. One important direction for future development is the integration of end-to-end encryption, which would ensure that communication remains confidential throughout message transmission and significantly strengthen data privacy. Additional authentication mechanisms, including multi-factor authentication and role-based access control, could further improve system security and protect against unauthorized access. The communication capabilities of the application can be expanded by incorporating voice calling, video conferencing, and screen-sharing services, thereby transforming the platform into a complete unified communication solution suitable for both personal and professional collaboration. User interaction can be enhanced through the implementation of presence management features, including online and offline status indicators, typing notifications, message delivery acknowledgments, and read receipts. These capabilities would provide

users with a richer and more interactive communication experience comparable to modern commercial messaging platforms. Future versions of the system may also support advanced message management functionalities such as message editing, deletion, reactions, forwarding, threaded conversations, and multimedia preview. The integration of cloud storage services would facilitate efficient management of large multimedia files while reducing storage limitations associated with local server deployment. Furthermore, migrating the backend database from SQLite to enterprise-level database management systems such as MySQL, PostgreSQL, or MongoDB would improve scalability, transaction management, and concurrent access capabilities required for large-scale deployments. Scalability can be further enhanced through cloud-based deployment using containerization technologies and orchestration platforms, enabling automatic resource allocation, load balancing, and fault tolerance. Deploying the application on public cloud infrastructures would provide global accessibility while supporting a substantially larger number of simultaneous users. In addition, the development of dedicated Android and iOS mobile applications would increase accessibility and improve user convenience by providing seamless communication across multiple devices.

References

[1] S. A. Triantafyllou, "A chat client-server application for E-learning," in *Proceedings of the 31st National Conference with International Participation (TELECOM)*, Sofia, Bulgaria, 2023, pp. 1–4.

[2] V. Rawat, P. Jain, A. Tomar, P. Guzar, V. Negi, and V. K. Gupta, "A web-based real-time chat application for user-friendly interaction," in *Proceedings of the 1st International Conference on Advanced Computing and Emerging Technologies (ACET)*, Ghaziabad, India, 2024, pp. 1–6.

[3] P. Saraswathi, S. Matheswaran, T. Nagulesh Waran, R. Palanivelrajan, G. A. Kabilaish, and G. Nithish Kumar, "Web-based real-time chat application with event-driven architecture," in *Proceedings of the 5th International Conference on Advancement in Electronics & Communication Engineering (AECE)*, Ghaziabad, India, 2025, pp. 1060–1063.

[4] S. Singh, S. Singh, and A. Sharma, "Real-time secure web-based chat application using Django," in *Proceedings of the 5th International Conference on Advances in Computing, Communication Control and Networking (ICAC3N)*, Greater Noida, India, 2023, pp. 1560–1565.

[5] V. S. R, S. Charan Krishnasagarapu, A. Kathula, B. K. Singh Bondili, and S. Reddy Baddam, "SignalR-based real-time chat application with ReactJS and .NET," in *Proceedings of the 8th International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud)*, Kirtipur, Nepal, 2024, pp. 957–965.