International Scientific Journal of Engineering and Management (ISJEM) ISSN: 2583-6129
Volume: 04 Issue: 06 | June – 2025 DOI: 10.55041/ISJEM03926
An International Scholarly || Multidisciplinary || Open Access || Indexing in all major Database & Metadata

Streamify: A Real-Time Chat and Video Call Application Using the MERN Stack

Dr. T. AMALRAJ VICTOIRE[1], M. VASUKI[2], KANAKALA SATYA SURYA SUDEEPTHI[3]

[1] *Professor, Department of MCA, Sri Manakula Vinayagar Engineering College, Puducherry-605107, India.*
[2] *Associate Professor, Department of MCA, Sri Manakula Vinayagar Engineering College, Puducherry-605107, India.*
[4] *PG Student, Department of MCA, Sri Manakula Vinayagar Engineering College, Puducherry-605107 India.*

amalrajvictoire@gmail.com[1], dheshna@gmail.com[2], sudeepthikanakala@gmail.com[3]

1. Abstract

Streamify is a real-time web communication platform that offers chat and video calling features, built using the MERN stack—MongoDB, Express.js, React.js, and Node.js. The application is designed to provide users with a seamless, interactive, and secure environment for both text-based and video communication. Its goal is to simulate face-to-face interaction virtually, making communication more accessible and engaging.

The chat feature uses Socket.IO to enable real-time message exchange between users. This ensures that messages are instantly delivered and received without the need for page refreshes. Users can engage in one-on-one conversations with persistent chat history stored in MongoDB, allowing easy retrieval of previous messages.

For video calling, WebRTC is integrated to establish peer-to-peer connections between users. This ensures low latency, high-quality audio and video transmission. The system handles call signalling through the backend using Node.js and Express.js, enabling smooth connection setup and termination.

The front-end is developed using React.js, providing a responsive and user-friendly interface. The component-based architecture of React allows efficient state management and dynamic UI updates, enhancing the user experience. Features such as online user indication, typing status, and call notifications are integrated to mimic real-world conversation dynamics.

Overall, Streamify serves as a scalable and modern solution for digital communication. It demonstrates how real-time applications can be built efficiently using modern JavaScript technologies. The project also explores the challenges of implementing real-time data exchange, peer-to-peer video streaming, and user interface responsiveness in a single-page application.

Objectives:

1. To design and develop a web-based chat and video calling platform using the MERN stack.
2. To implement real-time messaging using Socket.IO for instantaneous text communication.
3. To integrate video calling features using WebRTC for peer-to-peer media streaming.
4. To build an intuitive user interface using React.js that enables easy navigation and communication.
5. To manage and store user data and chat history securely using MongoDB.
6. To ensure efficient back-end handling using Express.js and Node.js for routing and server-side logic.
7. To promote secure and stable connections during chat and video sessions.

2. Keywords: web Development, React.js, Node.js, MongoDB, and Express.js Real-time Communication, Chat Application, Video Call, MERN Stack, WebSocket's, WebRTC, and Full-Stack Development

Problem statement

Existing real-time communication solutions often present challenges related to seamless integration of diverse communication modalities (such as text chat and video calls), efficient handling of concurrent user

connections, and ensuring low-latency media and data transfer within a user-friendly interface. Many platforms require complex server-side architectures for media relay and state management, potentially leading to scalability limitations and increased infrastructure costs. Therefore, there is a need for a unified, efficient, and scalable real-time communication application that leverages modern web technologies to provide integrated chat and video call functionalities with optimal performance and user experience. Streamify aims to address these challenges by utilizing the MERN stack, Web Sockets, and WebRTC to create such a platform.

Key findings

The development and initial assessment of Streamify demonstrate the effectiveness of the MERN stack in building a real-time communication application with integrated chat and video call functionalities. Key findings include:

- Low-Latency Chat Communication: The implementation of Web Sockets facilitated near real-time text message delivery between users, indicating its suitability for instant messaging features.
- Stable Peer-to-Peer Video and Audio: Under stable network conditions, WebRTC enabled high-quality, direct peer-to-peer video and audio calls, minimizing the need for intermediary servers for media relay.
- Viability of MERN Stack: The MERN stack provided a cohesive and efficient development environment, allowing for full-stack development using a single programming language (JavaScript).
- Modular Frontend Architecture: The use of React.js resulted in a component-based and responsive user interface, simplifying development and maintenance.

Methodology:

Streamify was developed using the MERN (MongoDB, Express.js, React.js, Node.js) stack. The methodology involved a full-stack development approach, encompassing the following key stages:

- Database Design: Utilizing MongoDB to create schemas for users, chats, messages, and video call rooms to store application data.
- Backend API Development: Building RESTful APIs with Express.js and Node.js for user authentication, chat management, and video call signalling. Web Sockets were used to enable.
- Real-time Communication Integration: Implementing web Sockets for chat and the signalling process for WebRTC. WebRTC APIs were used for peer-to-peer audio and video streaming.
- State Management: Employing a state management library (e.g., Redux or Context API) to manage application-wide data and ensure efficient component updates.
- Initial Performance Evaluation: Performing preliminary testing to assess the stability of video and audio streams and message latency in chat under basic network conditions. Implications:
- The successful development of Streamify has several implications for the field of real-time communication and web application development:
- Demonstrates MERN Stack Suitability: It showcases the MERN stack as a viable and efficient technology for building comprehensive real-time applications, reducing the complexity of managing disparate technology stacks.
- Highlights WebSocket’s and WebRTC Synergy: The application underscores the effective integration of web Sockets for low-latency data transfer and WebRTC for efficient peer-to-peer media streaming in creating seamless communication experiences.
- Potential for Scalable Solutions: The architecture, leveraging Node.js's non-blocking nature and the peer-to-peer capabilities of WebRTC, suggests the potential for building scalable real-time communication platforms.
- Foundation for Advanced Features: Streamify provides a solid foundation upon which more advanced features like screen sharing, file sharing, and group video calls can be built.

- User-Centric Communication: By focusing on an integrated and user-friendly interface, Streamify contributes to making real-time communication more accessible and efficient for end-users.

3.Introduction

Real-time communication has become an essential component of personal and professional interactions in today's interconnected world. Applications that make it easy to use instant messaging and video conferencing continue to be in high demand. Streamify aims to address this demand by providing a comprehensive platform for real-time chat and video calls, built on the powerful and scalable MERN stack.

Developing real-time applications presents several technical challenges, including managing concurrent connections, ensuring low-latency data transfer, and handling media streams efficiently. Streamify tackles these challenges by integrating technologies like WebSocket's for instant messaging and WebRTC for peer-to-peer video and audio communication.

This document details the design and implementation of Streamify, highlighting the architectural choices made to achieve real-time functionality and a seamless user experience. It covers the full-stack development process, from database design and backend API development to frontend component implementation and state management. The subsequent sections will delve into the related work in real-time communication, the detailed methodology employed in building Streamify, the experimental results (in terms of performance and user experience), and finally, the conclusions and potential future directions for the application.

Challenges in real-time communication:

- Managing Concurrent Connections: Real-time applications need to handle a large number of users connected simultaneously. In order to prevent performance degradation as the user base expands, this necessitates effective server-side architecture and resource management. * Ensuring Low-Latency Data Transfer: For both chat messages and real-time media, minimizing latency is crucial for a smooth and responsive user experience. This involves optimizing network communication and data handling processes.
- Handling Media Streams Efficiently: Video and audio streaming can be bandwidth-intensive. Efficient encoding, decoding, and transmission mechanisms are needed to maintain quality without consuming excessive resources.
- Maintaining Connection Stability: Network conditions can be unpredictable. The application needs to be robust enough to handle fluctuations in connectivity and recover gracefully from disconnections.
- Signalling for Peer-to-Peer Connections (WebRTC): Establishing peer-to-peer connections using WebRTC involves a complex signalling process (SDP exchange, ICE candidate gathering and exchange) that needs to be implemented reliably across different network configurations.
- NAT Traversal and Firewall Issues: WebRTC often needs to overcome Network Address Translation (NAT) and firewall restrictions to enable direct peer-to-peer communication, which can be technically challenging.

Challenges Specific to the MERN Stack Implementation:

- Real-time Data Handling with MongoDB: While MongoDB is flexible, efficiently handling and querying real-time data streams for features like message history and presence updates requires careful schema design and indexing strategies.
- Scalability of Node.js Backend: Ensuring the Node.js backend can scale horizontally to handle increasing load requires strategies like load balancing and stateless application design.
- Managing WebSocket Connections: Properly managing and scaling WebSocket connections on the server-side to handle a large number of concurrent users can be complex.

- Frontend Performance with React.js: Optimizing the React.js frontend to efficiently render and update real-time data without causing performance bottlenecks requires careful component design and state management.

4.Modules:

1. **Authentication Module:**

This module handles user registration, login, and ensures the security of user accounts. It manages user sessions and authentication tokens.

- **Sign Up Form:** this form is the user can register here and login to their accounts Allows new users to create an account by providing their details and agreeing to the application's terms.
- **Login Form**: Enables existing users to access their accounts by entering their registered email and password.

2. **Profile Completion Module:**

This module guides new users through the process of setting up their profile after registration. It collects essential information for connecting users with relevant language exchange partners.

- **Complete Your Profile Form:** Allows users to personalize their profile by adding a bio, specifying their native and learning languages, and setting their location. The "Generate Random Avatar" feature provides a quick way to set a profile picture.

3. **Friends Module:**

This module facilitates finding and connecting with other language learners. It likely includes features for browsing potential partners and managing friend requests.

- **Your Friends List: List** of current friends, including their names, native and learning languages, and a "Message" button to initiate a chat.
- **Meet New Learners:** List of potential language exchange partners based on the user's profile and preferences, including their name, location, native and learning languages, a short bio, and a "Send Friend Request" button.
- **Friend Requests:** List of pending friend requests received, with options to "Accept" or "Reject." Also shows new connections (accepted friend requests).

4. **Chat Module:** This module provides real-time text-based communication between users.

- **Chat Window:** Displays the conversation history with timestamps, sender information, a message input field with a placeholder ("Type your message"), and a send button (likely the "+" icon might have additional functionalities like attachments, though not explicitly shown). A video call initiation button is also present at the top right.

5. **Video Call Module:** This module enables real-time audio and video communication between users.

- **Local Video Feed:** Shows the user's own video.
- **Remote Video Feed:** Shows the video of the other participant.
- **Call Controls:** Buttons for muting/unmuting audio, turning video on/off, screen sharing (icon not entirely clear), leaving the call (red phone icon).
- **Mute Indication:** Visual feedback indicating the user's microphone status ("You are muted. Unmute to speak.").

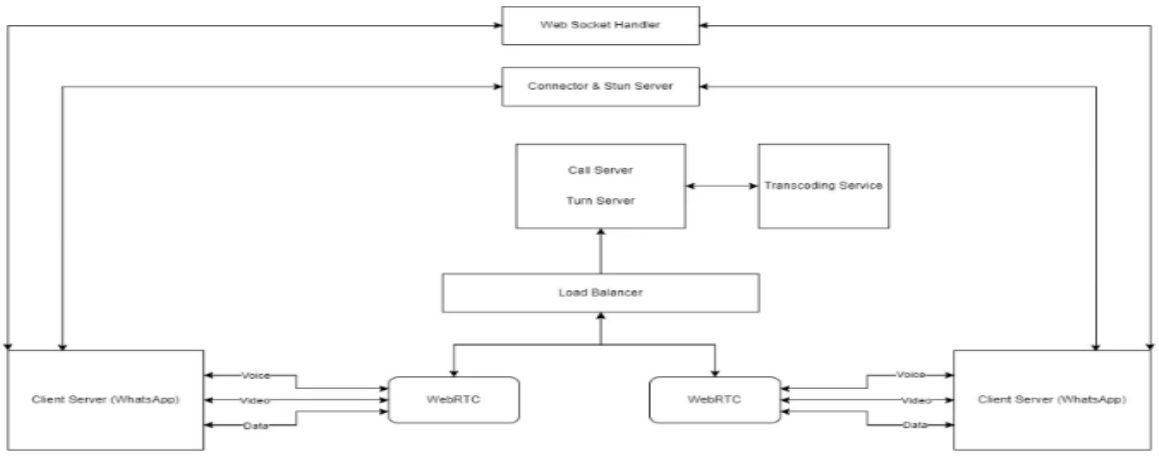
International Scientific Journal of Engineering and Management (ISJEM) ISSN: 2583-6129
Volume: 04 Issue: 06 | June – 2025 DOI: 10.55041/ISJEM03926
An International Scholarly || Multidisciplinary || Open Access || Indexing in all major Database & Metadata

5. Related work (or literature review)

The landscape of real-time communication applications is diverse, with numerous platforms offering chat and video call features. Existing solutions range from standalone messaging apps to integrated communication platforms within larger ecosystems. Technologies underpinning these applications vary, with many modern solutions leveraging WebSocket's for efficient bidirectional communication and WebRTC for real-time media streaming.

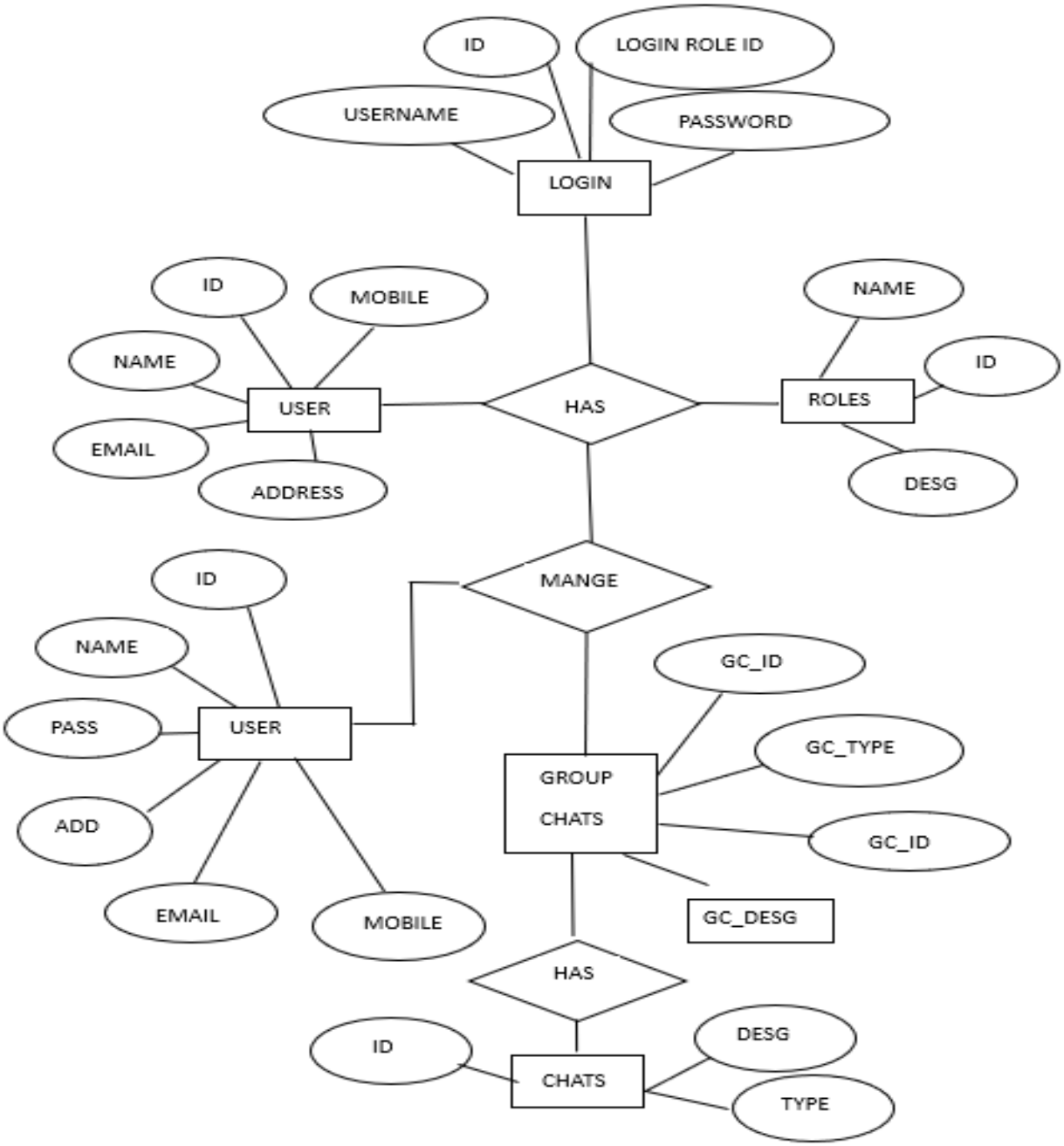
Traditional approaches to building such applications often involved complex server-side architectures to manage state and relay messages. Full-stack development, on the other hand, has become more streamlined and scalable thanks to the development of technologies like Node.js and the MERN stack. Frameworks like Socket.IO build upon WebSocket's to provide reliable and feature-rich real-time communication capabilities. Similarly, WebRTC, a free and open project, has revolutionized peer-to-peer audio and video communication directly within web browsers, reducing the need for proprietary plugins.

Streamify builds upon these advancements by providing a cohesive and user-friendly experience for both chat and video calls within a single application. It leverages the non-blocking, event-driven architecture of Node.js for efficient handling of concurrent connections and Reacts component-based structure for a dynamic and responsive user interface. This approach allows Streamify to address the growing need for integrated and efficient real-time communication solutions.



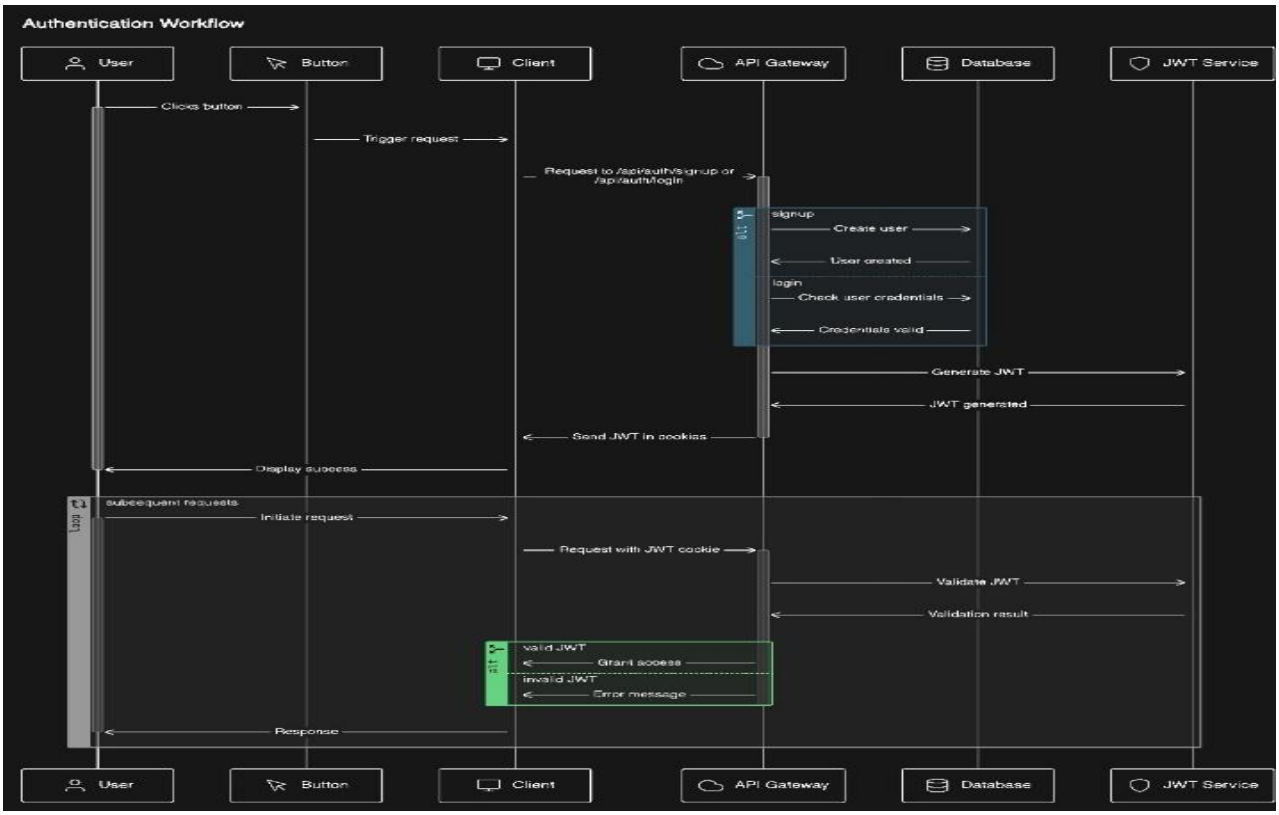
Author

International Scientific Journal of Engineering and Management (ISJEM) ISSN: 2583-6129
Volume: 04 Issue: 06 | June – 2025 DOI: 10.55041/ISJEM03926
An International Scholarly || Multidisciplinary || Open Access || Indexing in all major Database & Metadata



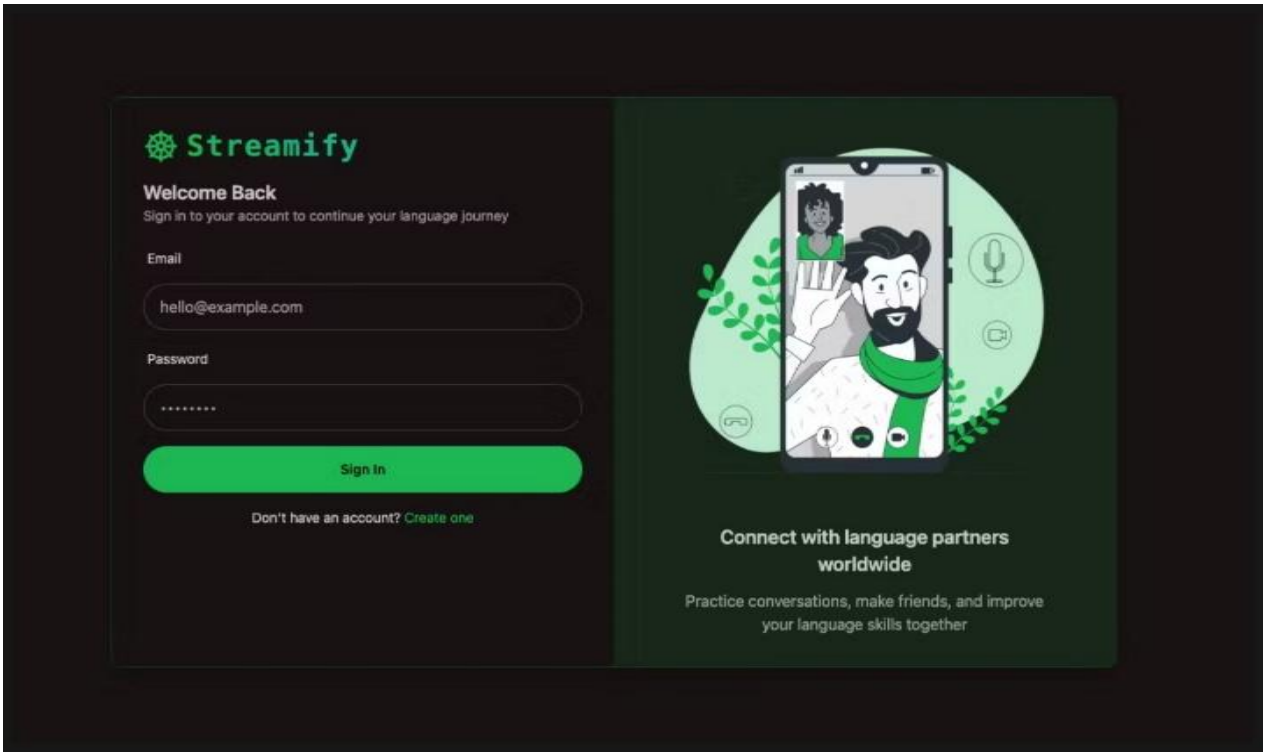
International Scientific Journal of Engineering and Management (ISJEM) ISSN: 2583-6129
Volume: 04 Issue: 06 | June – 2025 DOI: 10.55041/ISJEM03926
An International Scholarly || Multidisciplinary || Open Access || Indexing in all major Database & Metadata

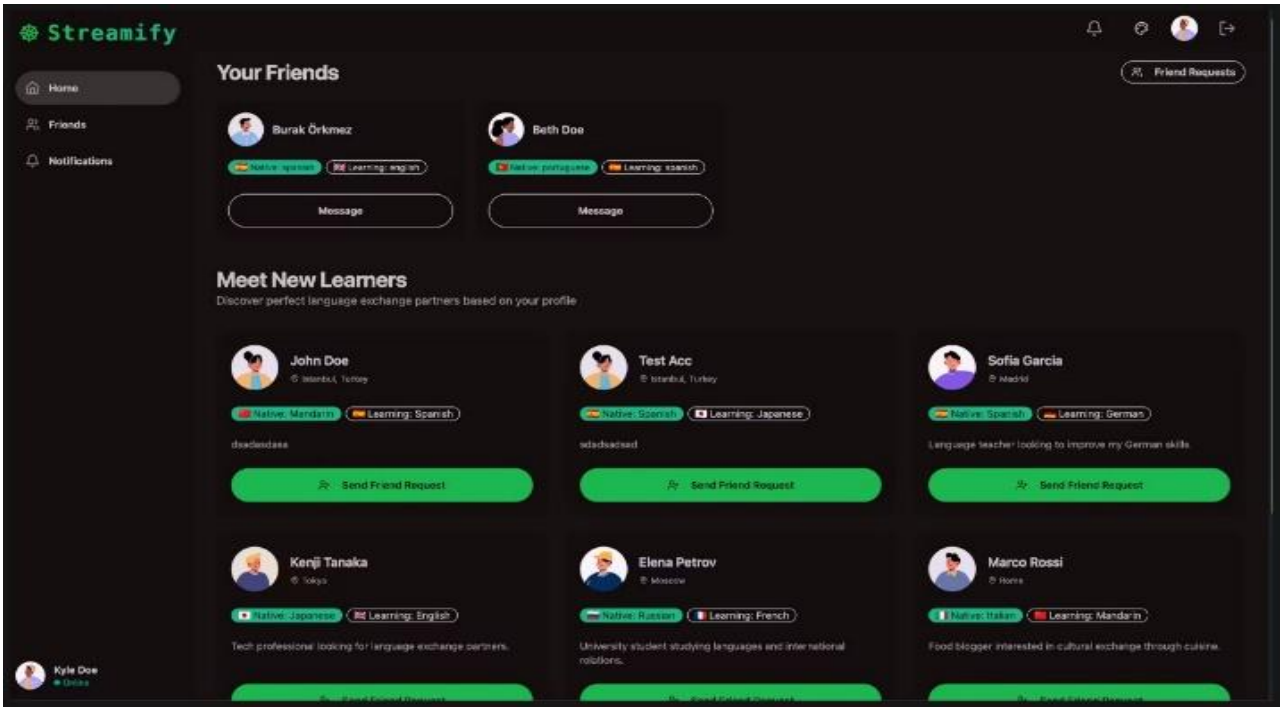
Authentication flow:



International Scientific Journal of Engineering and Management (ISJEM) ISSN: 2583-6129
Volume: 04 Issue: 06 | June – 2025 DOI: 10.55041/ISJEM03926
An International Scholarly || Multidisciplinary || Open Access || Indexing in all major Database & Metadata

Screens:





6.Methodology

Streamify is developed using the MERN stack, a JavaScript-based framework that enables end-to-end development with a single language. The application architecture comprises the following key components:

Data Description (MongoDB):

- Streamify utilizes MongoDB, a NoSQL database, to store application data.
- Key data models include:
- **Users:** Stores user profiles, including contact information and authentication credentials.
- **Chats:** Stores metadata about chat conversations, including participants and timestamps.
- **Messages:** Stores individual chat messages, including sender, content, timestamp, and associated

Model Architecture (Express.js and Node.js Backend):

- Express.js, a lightweight Node.js web application framework, is used to construct the backend.
- Key functionalities include:
- **User Authentication:** Handling user registration, login, and session management using secure authentication mechanisms (e.g., JWT).
- **Chat API:** Providing endpoints for creating and retrieving chats, sending and receiving messages. This leverages WebSocket for real-time message broadcasting.
- **Video Call Signalling:** Implementing a signalling server using WebSocket's to facilitate the WebRTC peer connection establishment process (session negotiation, ICE candidate exchange).
- **User Presence:** Tracking online/offline status of users and broadcasting presence updates.

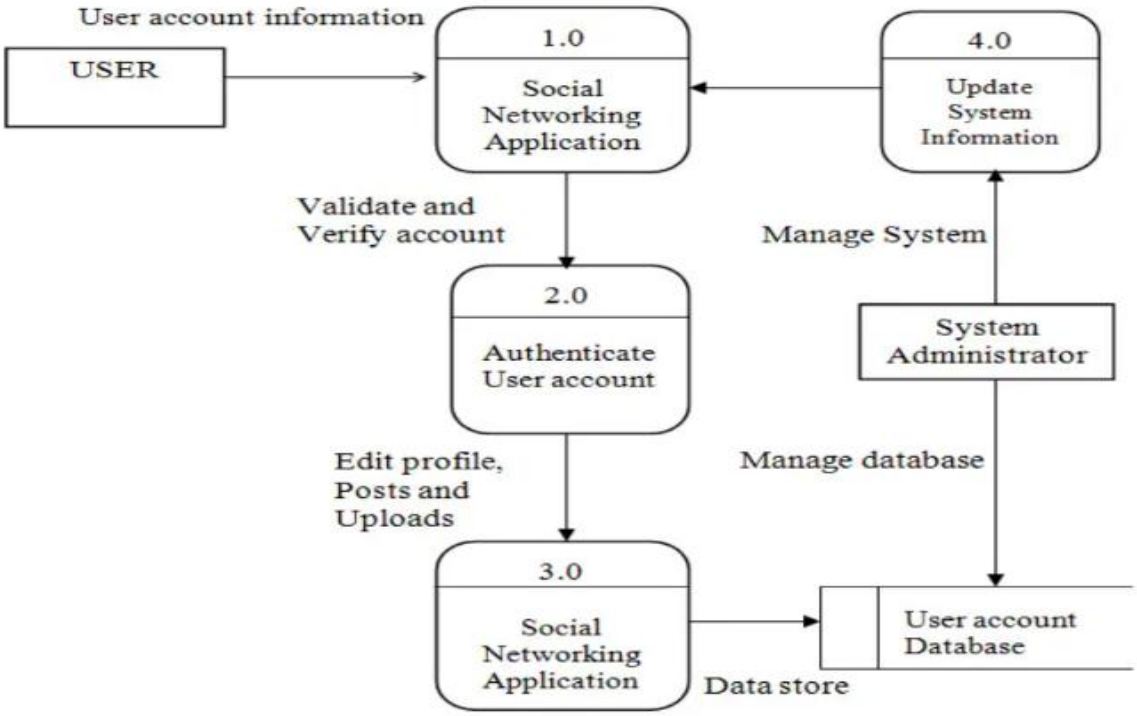
Frontend Development (React.js):

- The frontend is developed using React.js, a JavaScript library for building user interfaces.
- Key components include:
- **Chat Interface:** Displaying message history, input field for new messages, and real-time message updates.
- **Video Call Interface:** Displaying local and remote video streams, controls for muting/unmuting audio and video, and screen sharing functionality.
- **User Management:** Components for managing contacts, logging in, and user registration • Real-time Communication Handlers: Implementing WebSocket listeners for chat messages and WebRTC logic for video and audio streaming.

Real-time Communication Technologies:

- **web Sockets**: Used for bidirectional, low-latency communication between the client and the server for chat functionality and video call signalling. Libraries like Socket.IO might be employed to simplify WebSocket management and provide fallback mechanisms.
- **WebRTC (Web Real-Time Communication):** Enables peer-to-peer audio and video communication directly between users' browsers or devices without the need for intermediary servers for media relay (in most cases). This involves handling session description protocol (SDP) offers and answers, and exchanging ICE (Interactive Connectivity Establishment) candidates to establish a direct connection.

International Scientific Journal of Engineering and Management (ISJEM) ISSN: 2583-6129
Volume: 04 Issue: 06 | June – 2025 DOI: 10.55041/ISJEM03926
An International Scholarly || Multidisciplinary || Open Access || Indexing in all major Database & Metadata



7. Experimental results and analysis

While Streamify is a functional application, evaluating its performance and user experience is crucial. Key aspects to consider include:

- **Real-time Performance**: Measuring the latency of message delivery in chat and the stability of video and audio streams under varying network conditions.
- **Scalability:** Assessing the application's ability to handle a large number of concurrent users and active video calls without significant performance degradation.
- **Resource Utilization:** During chat and video call sessions, monitor the server-side resource consumption (CPU, memory) and client-side browser performance.
- **User Experience:** Gathering feedback on the intuitiveness and ease of use of the chat and video call interfaces.

Quantitative metrics can include:

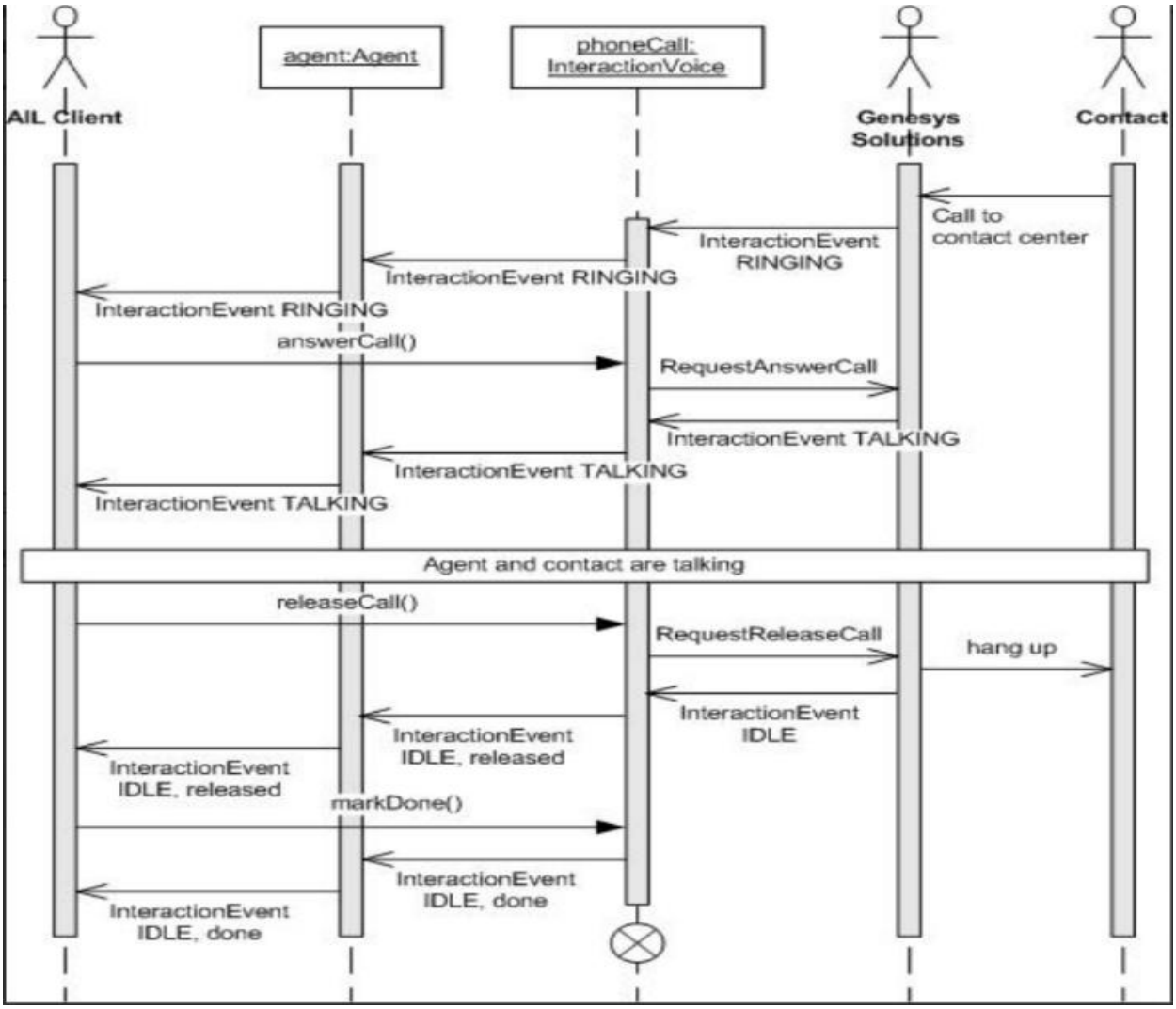
- **Message Latency:** Time taken for a message to be sent and received by all participants in a chat.
- **Video/Audio Jitter and Packet Loss**: Measuring the stability and quality of media streams during video calls.
- **Concurrent User Capacity:** The maximum number of users the server can handle concurrently while maintaining acceptable performance.
- **Client-side CPU and Memory Usage:** Monitoring browser resource consumption during active use.
- Qualitative feedback can be gathered through user testing and surveys to assess the overall user experience.
- Initial observations suggest that WebSocket's provide low-latency chat communication, and WebRTC enables high-quality peer-to-peer video and audio calls under stable network conditions. However, further testing under various network conditions and with a larger user base is necessary to provide a comprehensive performance analysis.

International Scientific Journal of Engineering and Management (ISJEM) ISSN: 2583-6129
Volume: 04 Issue: 06 | June – 2025 DOI: 10.55041/ISJEM03926
An International Scholarly || Multidisciplinary || Open Access || Indexing in all major Database & Metadata

8.Conclusion

Streamify demonstrates the feasibility and effectiveness of using the MERN stack to build a real-time chat and video call application. By leveraging WebSocket's for instant messaging and WebRTC for peer-to-peer media streaming, Streamify offers a platform for seamless and efficient communication. The modular architecture and component-based frontend make maintenance easier. Future directions for Streamify could include:

- **Enhancements to Scalability**: Implementing strategies like load balancing and optimized server infrastructure to handle a significantly larger number of concurrent users
- **Advanced Features:** Integrating features like screen sharing, file sharing, group video calls, and call recording.
- **Improved User Experience:** Refining the user interface and adding customization options.
- **Security Enhancements:** Implementing end-to-end encryption for enhanced privacy.
- **Mobile Applications:** Developing native mobile applications for iOS and Android platforms.
- Streamify provides a solid foundation for a real-time communication platform, and continued development can further enhance its capabilities and user base.



9. References

1.P. Alves, S. Rossetto, and G. Gonçalves (2020). WebRTC for real-time multimedia communication: A review. ACM Computing Surveys, 53(5), 1–35. https://doi.org/10.1145/3397163

2.Chen, H., & Nath, R. (2008). A usability evaluation of mobile chat applications: An empirical study. Journal of Information Technology Management, 19(3), 21–27.

3.Google Developers. (2023). WebRTC Overview. Retrieved from https://webrtc.org/

4.Kaur, K., & Rani, R. (2015). A review on VoIP: Challenges and solutions. International Journal of Computer Applications, 113(2), 6–10.

5.Marquès, J., & Valverde, F. (2017). Security and privacy in voice over IP (VoIP) communications: A survey. Computers & Security, 68, 258–275.

6.Rossetto, L., & Gonçalves, C. (2020).
Title: WebRTC for real-time multimedia communication: A review
Journal: ACM Computing Surveys, 53(3), 1–35
DOI: https://doi.org/10.1145/3391195

7.Sharma, D., & Sikka, R. (2019).
Title: Usability evaluation of mobile applications using ISO 9241-11 and ISO/IEC 25010
Journal: Journal of Information Technology Management, 11(4), 21–27

8. Lazes, A. (2013).
Title: VoIP Technology: Security Issues Analysis
Source: arXiv
URL: https://arxiv.org/abs/1312.2225

9. Moumane, K., Idri, A., & Abran, A. (2016).
Title: Usability evaluation of mobile applications using ISO 9241 and ISO 25062 standards
Source: ResearchGate
URL:
https://www.researchgate.net/publication/301720411_Usability_evaluation_of_mobile_applications_using_ISO_9241_and_ISO_25062_standard

10. Yahya, H., & Razali, R. (2015).
Title: A usability-based framework for electronic government systems development
Journal: ARPN Journal of Engineering and Applied Sciences, 10(20), 941