Secure Web Chat Application

Yasuri Amaratunga – IT21800764
SSS- IE3042
Sri Lanka Institute of Information Technology, Sri Lanka
it21800764@my.sliit.lk
Github Link: https://github.com/it21800764/IT21800764-SSS

Abstract-The design, development, and implementation of a secure web chat application, Connectify using modern technologies like React, Node.js, Socket.io, and WebRTC are presented in this article. In today's digital world, the application responds to the growing demand for secure communication platforms. To guarantee a strong and secure user experience, key aspects including End-to-End Encryption, Authentication and Authorization, Real-time Communication, User Privacy, and Scalability and Reliability are highlighted.

Completely All forms of communication, including text messages, voice conversations, and video calls, are guaranteed to stay private and unreadable by unauthorized parties thanks to encryption. To improve security, authentication and authorization systems make sure that only authorized users with the right permissions can access the functions and data of the program [1].

Technologies such as Socket.io and WebRTC offer real-time communication features that facilitate smooth and low-latency communication, hence promoting effective user cooperation. Strict privacy regulations are implemented to protect user privacy and prevent unwanted access or monitoring of user data [1].

The application's scalability and reliability are crucial components that guarantee it can handle an expanding user base while preserving high availability and performance. Adherence to industry-standard security norms and regulations is ensured through the application of contemporary technologies and best practices.Overall, in an increasingly connected digital world, the secure web chat application that is being offered provides a safe, easy-to-use, and scalable platform for people and organizations to communicate with confidence while protecting their privacy and data integrity.

I. Introduction

The Development of safe communication channels has become essential in a time of swift technological development and growing worries about digital security and privacy. This article provides an in-depth analysis of a secure web chat service that was developed using cutting-edge technologies including React, Node.js, Socket.io and WebRTC, This technical paper intends to answer the growing need for secure messaging solutions in today's linked world by improving user privacy, data integrity, and real-time communication capabilities.

The way people communicate and work within organizations has been completely transformed by the rise of internet communication tools. But this ease also carries some risk because traditional messaging apps frequently have poor security measures, which exposes users to dangers including data leaks, eavesdropping, and illegal access [2]. Understanding these difficulties, the goal of developing a secure web chat Application was to give consumers a dependable, private communication tool that successfully reduces these risks.

The primary objective of this is to create and deploy a safe online chat application that puts the needs of users' privacy, data security, and real-time communication first. Our goal is to create a user experience that is both smooth and simple while maintaining strong security measures. To achieve this, we are utilizing the power of contemporary web technologies, such as WebRTC for video calls, Socket.io for real-time communication, and React for the

front-end interface and backend infrastructure.

II. Literature Review

In the realm of secure communication, users have a range of options depending on their priorities. Depending on their preferences, consumers have a variety of possibilities in the world of secure communication. Wickr Me and Signal are the best options for people looking for the most privacy [3]. Both use strong, by default, open-source end-to-end encryption (E2EE) to scramble messages and calls, and even file transfers. As a donation-based organization, Signal reduces the financial incentives for gathering user data. Users who are concerned about security often choose it even if user base numbers aren't made public. With its emphasis on anonymity, Wickr Me enters the battle. No phone number or email address is needed to sign up, and messages can be set to disappear forever. Like Signal, the number of users isn't publicly disclosed, but Wickr Me serves users who want complete control over their online presence.

Wire is a strong option if you believe that security and features should be balanced. With its main office in Switzerland, a nation with strict privacy regulations, Wire provides E2EE and services to enterprises that require features like safe guest rooms [4]. Although Wire offers price levels, the extent of its user base is not publicly disclosed. Known for its enormous group chat capacity—which can support up to 20,000 users—Telegram adopts a different strategy [5]. Although it uses cloud-based storage for easy access across devices, E2EE encryption is not the default setting; rather, it is an optional feature. Since messages wouldn't be safeguarded if servers were compromised, this raises problems. Despite this, Telegram is a well-liked option for people who value extensive group communication, with over 500 million active users each month [5].

Applications such as WhatsApp and Messenger are very popular when it comes to convenience, but user privacy is not their first priority. WhatsApp, owned by Meta (formerly Facebook), provides default E2EE; nonetheless, user information is still gathered. With more than 2 billion active users each month, WhatsApp may still raise privacy concerns for people who value their privacy [6]. Messenger is owned by Meta as well, and although it provides an option for encrypted chats known as "Secret Conversations," user data harvesting is still a problem [6]. Because Messenger does not have separate user counts, it is challenging to compare standalone usage to Facebook. In the end, the safest chat app is determined by your unique requirements. Wickr Me or Signal are your best options if absolute privacy is your top concern. Wire is a good option if you want features and security in equal measure. Telegram is an alternative if your main requirement is for large group chats but be aware of the server-side storage trade-off. Recall that even the safest programs have their limits. Maintaining the security of your conversations also heavily depends on device security.

Many protocols are used in the world of encrypted chat applications, each with advantages and disadvantages. WebSocket create a persistent, two-way channel of communication between your device and the chat server. Think of it as an open phone line; great for communication in real time, but not very private. It's here that encryption becomes useful. WebRTC is a suite of protocols that allows web browsers to perform functions like video calls, Most importantly, though, WebRTC can encrypt data using Secure Real-time Transport Protocol (SRTP). Consider SRTP as a way to delay the discussion on an open line so that only the person who is meant to hear it can understand it.

Session Initiation Protocol (SIP) for voice-centric communication arises. With more than 3 billion VoIP (Voice over Internet Protocol) users worldwide, SIP is a voice call setup and management powerhouse [7]. But encryption is not handled by SIP itself. XMPP, or Extended Messaging and Presence Protocol, comes into play here. Applications such as Signal employ the flexible open-source messaging protocol provided by XMPP, which is estimated to have tens of millions of users. It supports multiple forms of

communication, such as instant messaging, and integrates with encryption layers for more protection. Thus, WebRTC and SRTP encrypt your discussions, WebSockets maintain open lines, SIP handles voice calls, and XMPP provides a secure message basis - all of which combine to create the core of secure chat applications.

Chat apps are undergoing a revolution thanks to machine learning (ML); each month, 3.5 billion users interact with the smart reply to capabilities [8]. Envision a chatbot that anticipates your responses, protects you from cyberattacks where 789 billion spam emails were identified in 2022, and provides round-the-clock assistance through chatbots [9]. By filtering spam, flagging offensive content, and suggesting replies based on data analysis, machine learning (ML) may eliminate the need for 4 million human moderators [10]. This transforms how we connect by making chat apps smarter, safer, and more effective.

III. System Overview

Secure chat programs put the privacy of their users first by encrypting calls and messages so that only the intended receivers can decipher them. These applications serve people and companies looking for a secure environment for private communication. With more than 3.5 billion users utilizing messaging apps' intelligent reply functions each month, there is an increasing need for secure chat substitutes [11].

Key components and their interactions:

Client Application: This is the user interface that users of tablets, PCs, and phones interact with. Users can create messages, make phone calls, and maintain contact lists with it. Prior to being sent to the server, messages are encrypted by the client application, which also decrypts incoming messages so you can view them.

Communication Protocol: The guidelines for data exchange between the client application and the server are specified in this protocol. Applications for secure chat frequently use a mix of protocols. While WebRTC allows capabilities like video calls and uses Secure Real-time Transport Protocol (SRTP) to encrypt the content, websocket provide a persistent connection

Server: This serves as the primary hub for user-to-user communication, relaying messages. The server may also handle functions like file transfers and group conversations in addition to storing user data and maintaining contact lists. In secure chat applications, message content is never decrypted by the server, guaranteeing confidentiality even in the event of a breach.

Encryption Keys: These distinct digital keys are utilized for communication encryption and decryption. There are primarily two kinds: widely dispersed public keys and user-kept private keys. The recipient's public key is used to encrypt messages, which can only be unlocked with their private key. This guarantees that the message can only read by the intended recipient.

The secret to the enchantment of secure chat apps is their capacity to jumble messages even before they leave your smartphone. This is how it operates: the client app acts as a digital lock, encrypting your message with the recipient's public key as soon as you push send. The communication protocol safely transports this encrypted message to the server, which simply serves as a postman by sending it to the intended recipient. When the message is delivered, the recipient's device unlocks and decrypts it so they may read it using their private key, which is the only key that matches the public key. In addition to machine learning capabilities like spam filtering, this secure communication flow protects your privacy and creates a secure environment for online interactions.

IV. Technical stack

Frontend

React: One of the best JavaScript libraries for creating user interfaces is this one. Because of its component-based architecture, reusable components such as video call interfaces, chat windows, and contact lists may be created.

This facilitates maintainability, streamlines development, and encourages modularity. For real-time chat applications, React's virtual DOM guarantees effective UI updates without needless browser re-renders, improving efficiency.

CSS (Styled Components): Cascading Style Sheets (CSS) specify how the programme is presented visually. With the help of the CSS-in-JS package Styled Components, developers can write CSS right inside of your React components. This encourages more organised code, minimises the need for distinct CSS files, and makes decorating intricate components easier.

Backend

Node.js: Node.js is an open-source JavaScript runtime environment with an event-driven, non-blocking I/O style that makes it ideal for the backend. Because of this, it can manage a large number of concurrent connections with ease, which is crucial for real-time chat systems that have a large user base. With Node.js, you can handle incoming and outgoing messages effectively without having to wait for one request to complete before handling another. Furthermore, reducing context switching for developers and simplifying development are two further benefits of using JavaScript on both the frontend and backend. Last but not least, Node.js has a huge ecosystem of third-party modules for features like user authentication, database interface, and encryption, which enables developers to employ pre-built solutions and concentrate on the essential application functionality.

Real time communication

Socket.io: A real-time communication package called socket.io makes it possible for clients and servers to communicate in both directions. It uses WebSockets, a persistent connection protocol, to create a permanent connection between clients and the server, enabling real-time communication. As a result, messages can be delivered right away without requiring frequent client polling. Furthermore, Socket.io provides versatility by supporting multiple transport techniques, maintaining compatibility with outdated browsers that might not support WebSockets, and scales well for large-scale chat applications.

Video calls: WebRTC: A web technology framework called WebRTC makes it possible to communicate in real time via audio and video from within web browsers. Users' video call experience is made simpler by not having to install any plugins. WebRTC places emphasis on establishing direct peer-to-peer (P2P) connections whenever feasible. This approach minimizes network delay, which can potentially improve video call quality and reduce server burden. Through its access to the Media Stream API, developers can record and share audio and video in real time during video chats using devices owned by users. Additionally, WebRTC can be integrated with encryption protocols such as SRTP to protect video call content and guarantee privacy while speaking.

API Testing: Postman: A program called Postman is used to test and develop APIs. Developers can use it to send HTTP requests, examine the answers, and troubleshoot API features. The secure chat application's backend APIs are tested with Postman, which is essential for making that messages are transmitted and received successfully, user authentication functions as intended, and other backend processes run smoothly.

These technologies are combined to create Connectify, a secure chat application with a user-friendly interface, real-time communication features, secure video calls, and a reliable backend for managing and storing data.

V. Software Architecture

Real-time communication is essential in most chat applications, and one popular way to accomplish this is by polling, in which the client periodically asks the server for updates. The procedure starts when an end-user called

the Sender interacts with the React-built client application to request fresh messages or updates. The Node.js-developed backend server receives this polling request from the client application. The server retrieves any fresh messages or pertinent user data by querying the database upon receiving the request. The most recent information is retrieved and prepared for relaying back to the client thanks to this database interaction.

The server replies to the client application with the most recent changes or messages after retrieving the required data. After processing the server's answer, the client modifies the user interface to reflect the changes, notifying the user of new activity or showing fresh messages in the chat window, for example. Lastly, the updated data is visible to the Receiver, another end-user, through the chat interface, guaranteeing that they are informed of the most recent exchanges and messages in real time. This constant communication keeps the chat programme current and responsive, giving users a flawless real-time communication experience.

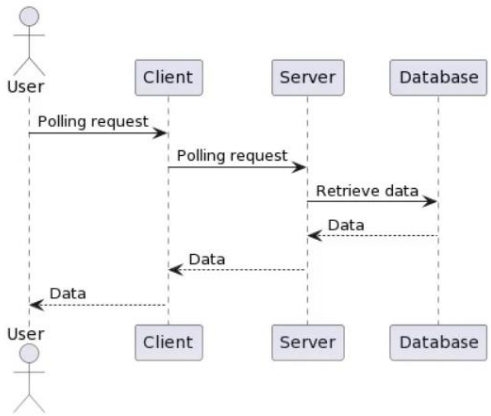

Fig 1. Real-time chat application architecture and workflow[12].

VI. Implementation Details

Connectify chat application is based on a strong development environment. For a more efficient development process, we were able to use JavaScript on both the frontend and backend by utilising Node.js and npm (Node Package Manager) for the backend server. We used React, a well-liked JavaScript toolkit renowned for its component-based architecture and effective UI rendering, for the front end. Code maintainability is enhanced by this modular approach, which makes it easier to create reusable components like contact lists and chat windows. To control the application's visual appearance, we also included CSS preprocessors or frameworks like Styled Components.

The focal point of communication is the Node.js-built backend server. It uses a database like MongoDB to manage user authentication, message routing, and data storage. To create a safe user base, features like login, and user registration are integrated. Incoming messages are likewise processed by the backend, and then delivers them to their designated recipients. React components render the user experience on the frontend, enabling users to browse messages, start video calls, and engage with chat features. Through the establishment of permanent connections between clients and the server, socket.io integration makes real-time communication easier. This makes it possible to send messages instantly and have a smooth chat experience.

Socket.io is implemented to allow real-time communication. This ensures that messages are sent instantly by doing away with the requirement for continuous polling. Moreover, WebRTC, a web technology foundation, was incorporated to enhance the application's video call features. Peer-to-peer connections are given priority over server load via WebRTC, which also has the potential to improve video call quality by lowering network latency. Additionally, it gives us access to the MediaStream API, which enables us to record audio and video from user devices and share it instantly with other users during video conferences. The privacy and security of video call content was guaranteed by the integration of encryption techniques with WebRTC.

VII. Security Measures

By putting strong security mechanisms in place at different levels, the Connectify chat Application prioritises user privacy.

Uncrackable Passwords: It is critical to safeguard user credentials. Strong password hashing methods like bcrypt is used to accomplish this. Passwords are rendered unreadable by these one-way processes, rendering them unreadable even in the event of a data breach. Keeping passwords in plain text would expose user accounts, which is a sharp contrast to this.

Securing Communication Channels: Connectify secure chat application uses HTTPS (Secure Hypertext Transfer Protocol) on all communication routes between clients and the server to ensure message confidentiality during transmission. can validate this security measure by using an API testing tool called Postman, which simulates requests and analyses responses.

Input Validation: strict input validation was put in place to stop malicious code injection attacks. This entails cleaning up all user input prior to the application processing it.

Secure Coding Practices: complied with secure coding guidelines to reduce the possibility of vulnerabilities that attackers could exploit. This entails utilizing secure and well-maintained libraries and frameworks, avoiding common coding errors, and adhering to recommended practices for memory management. From the beginning, secure coding standards are prioritized in an effort to create a solid and dependable application foundation.

VIII. Testing and Validation

Throughout development, the secure web chat application Connectify is put through a thorough testing process to guarantee a faultless user experience and strong security. Below is a summary of the main testing techniques used.

Testing Strategies:

Unit Testing: Unit testing is the basis of testing methodology. To ensure they function as intended, individual components—both frontend (React components) and backend (Node.js modules)—are tested separately. By doing this, bugs are found early in the development cycle and are kept from growing into more serious problems.

Integration Testing: Integration testing goes beyond evaluating individual components and instead examines how the application's many components function together. Testing communication between various backend services (like user authentication with the database) and interactions between the frontend and backend (like sending messages) are included in this. A smooth user experience is ensured by guaranteeing flawless data flow and operation throughout the whole program.

End-to-End Testing: It is essential to simulate actual user processes in order to find any usability problems. E2E testing includes simulating user activities such as messaging, video calls, registration, and application navigation. This all-encompassing strategy aids in the identification and resolution of any possible obstacles that can impair a user's experience.

Security Testing:

Static Code Analysis: It is critical to take proactive security measures. To examine the application code and find possible vulnerabilities like SQL injection or XSS attacks, we make use of static code analysis tools. We can stop these vulnerabilities from being used in the real world by fixing them as soon as possible.

Penetration Testing: The practice of ethical hacking replicates actual attacks by knowledgeable attackers. By attempting to take advantage of application vulnerabilities, penetration testers assist us in locating and fixing possible security flaws before attackers may take use of them. Connectify's security posture is strengthened by this thorough testing, which also protects user privacy and data.

Performance Testing:

Load Testing: Any online application must be scalable. Load testing evaluates Connectify's capacity to withstand peak loads without experiencing reduced performance by simulating many concurrent users. This guarantees that the application's responsiveness won't be compromised while serving an increasing user base.

IX. Future Work

The journey of Connectify is still not over. developments to investigate the possibilities of machine learning (ML) and improve user experience even further are to be put into practice as below:

Advanced Features: Expanding functionalities like file sharing beyond basic types (images, documents) and incorporating group chat capabilities.

Enhanced Search Functionality: Integrating natural language processing (NLP) techniques could enable users to search chat history more effectively using natural language queries.

Chatbots and Virtual Assistants: Machine learning could power chatbots or virtual assistants within Connectify, offering automated customer support, answering frequently asked questions, or performing basic tasks

X. Challenged and Limitations

There was a steep learning curve involved in Developing secure web chat product, Connectify. It used an entire new technology stack to complete the project: MongoDB, Socket.io, Node.js, and React. It took a lot of research and learning to understand the features of each technology and how they interact to produce a real-time chat application. This process of going from knowing nothing to creating a working application gave me a thorough understanding of the selected stack and its possibilities.

Setting up a reliable connection between Server and MongoDB was a big first step. For smooth communication, it was essential to comprehend database schemas, authentication procedures, and appropriate data modelling techniques. Investigating the MongoDB documentation, testing with connection setups, and identifying typical problems were all necessary to overcome this obstacle. Even though these first obstacles were big, the journey to overcome them turned out to be a priceless teaching moment.

Despite its strength, Connectify has limitations, such as the lack of functions that could increase its use, like group chat and advanced file sharing.

X. Conclusion

To sum up, Connectify offers a strong response to the growing need for safe, rapid communication systems. Using cutting-edge web technologies like WebRTC, React, Node.js, and Socket.io, Connectify provides a user-friendly interface together with advanced security features . Together, these components guarantee that all communication—including video calls, and text messages—remains confidential and safe from unwanted access. By emphasising the value of preserving user privacy, data integrity, and dependable performance, the application's architecture and execution make Connectify a safe and scalable communication tool for both individuals and businesses.

The project was successful in developing a secure and practical web chat application, even with the early difficulties arising from learning a new technology stack and setting up secure server connections. Connectify's capabilities are expected to be further enhanced in the future with the addition of complex features like group chat and sophisticated file sharing, as well as the application of machine learning for a better user experience. Connectify is proof that cutting-edge technology and strict security procedures can be used to provide a stable platform for safe online interactions as digital communication continues to develop.

XI. Acknowledgement

I would like to express my sincere appreciation to Ms. Cethana Liyanapathirana, our lecturer in charge, for her guidance during the development of this application. My heartfelt thanks also go to the lab assistants for their assistance throughout the implementation process. I am deeply grateful to my friends for their encouragement and invaluable insights. Additionally, I extend my gratitude to the multitude of online resources that served as valuable sources of information and assistance, enabling me to navigate challenges and make significant strides in the development of this application.

XII. Reference

[1] "Pubnub," [Online]. Available: https://www.pubnub.com/blog/building-secure-web-chat-apps/.

[2] "stack hawk," [Online]. Available: https://www.stackhawk.com/blog/10-web-application-security-threats-and-how-to-mitigate-them/.

[3] "avast," [Online]. Available: https://www.avast.com/c-most-secure-messaging-apps#:~:text=And%20Signal%20is%20funded%20by,like%20geotags%20and%20message%20times..

[4] "Wire," [Online]. Available: https://wire.com/en.

[5] "telegram.org," [Online]. Available: https://telegram.org/faq.

[6] "whatsapp," [Online]. Available: https://faq.whatsapp.com/820124435853543.

[7] "Reve Systems," [Online]. Available: https://www.revesoft.com/blog/telecom/sip-vs-voip/.

[8] "Medium," [Online]. Available: https://medium.com/@StartXLabs/the-ai-revolution-in-mobile-apps-a-deep-dive-into-proactive-intelligence-d1de14fa31b2.

[9] "bender," [Online]. Available: https://mybendersolutions.com/cyber-bytes-safeguard-yourself-from-ai-chatbot-scams/.

[10] "The impact of algorithms," 2020.

[11] "sendbird," [Online]. Available: https://sendbird.com/blog/best-chat-apps.

[12] k. bak, "Real-time chat application design: architecture and workflow," 2023.