VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"Jnana Sangama", Belagavi, Karnataka-590018

A Mini Project Report

On

" REAL TIME CHAT APPLICATION "

Submitted by

METRI KEERTI SOMASHEKHAR	**4GK21CS025**
NAVEEN J R	**4GK21CS028**
SHREEDHAR CHANNAPPA TIKOTI	**4GK21CS045**
VAISHNAVI HANAMANT BHAJANTRI	**4GK21CS052**

Under the Guidance of

Dr. SHIVASHANKARA S
Assistant professor
Department of Computer Science and Engineering

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

K R PETE KRISHNA GOVERNMENT ENGINEERING COLLEGE

K.R PET-571426

2023-24

K R PET KRISHNA GOVERNMENT ENGINEERING COLLEGE

K. R PETE -571426

(Affiliated to Visveswaraya Technological University, Belagavi, Approved by AICTE, New Delhi)

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

CERTIFICATE

This is to certify that Mini Project report entitled " **REAL TIME CHAT APPLICATION**"has been carried out by **METRI KEERTI SOMASHEKHAR [4GK21CS025], NAVEEN J R [4GK21CS028],SHREEDHAR CHANNAPPA TIKOTI [4GK21CS045],VAISHNAVI HANAMANT BHAJANTRI [4GK21CS052]** ,for the partial fulfilment of Bachelor of Engineering in Computer Science and Engineering of Visveswaraya Technological University, Belagavi during the year 2023-24. It is certified that all corrections/suggestions indicated for Internal Assessment have been incorporated in the report. The Mini Project work report has been approved as it satisfies the academic requirements in report of miniProject prescribed for the semester.

Signature of the Guide	**Signature of the HOD**
Dr. Shivashankara S	**Dr. Hareesh K**
Assistant professor	Associate Professor & HOD
Dept. of CSE	Dept. of CSE

ACKNOWLEDGEMENT

We feel great pleasure in submitting this report **“REAL TIME CHAT APPLICATION ” .**The successful completion of any task would be incomplete without the mention of people who made it possible and whose support had been a constant of encouragement which crowned our efforts with success.

We express my sincere gratitude to **Dr. K R Dinesh**, the principal of K R pete Krishna Government Engineering College, K R Pet for providing healthy environment in the college, which helped in concentrating on the task.

We thank to entire faculty of the Computer Science and Engineering Department, especially **Dr. Hareesh K,** Head of the Department CSE, who has given me confidence to believe in ourselves and complete the project.

We give a great sense of satisfaction in acknowledging the encouragement & immense support of our Guide, **Dr. Shivashankara S** Assistant Professor of Computer Science and Engineering Department, for his valuable guidance, suggestion and consistent encouragement during the course of my mini project

We are thankful to all the staff members of the Department of Computer Science and Engineering for their support and encouragement.

Finally, we thank each and everybody who have directly or indirectly contributed to this work. We always welcome suggestions regarding the presented report.

METRI KEERTI SOMASHEKHAR	**4GK21CS025**
NAVEEN J R	**4GK21CS028**
SHREEDHAR CHANNAPPA TIKOTI	**4GK21CS045**
VAISHNAVI HANAMANT BHAJANTRI	**4GK21CS052**

K R PET KRISHNA GOVERNMENT ENGINEERING COLLEGE

K. R PETE -571426

(Affiliated to Visveswaraya Technological University, Belagavi, Approved by AICTE, New Delhi)

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

DECLARATION

We, **METRI KEERTI SOMASHEKHAR [4GK21CS025], NAVEEN J R [4GK21CS028], SHREEDHAR CHANNAPPA TIKOTI[4GK21CS045], VAISHANVI HANAMANT BHAJANTRI [4GK21CS052]**,are respectively studying in the 6th semester of Bachelor of Engineering in Computer Science at K R Pete Krishna Government Engineering College K R Pet, Mandya. Hereby declare that the mini project entitled "**REAL TIME CHAT APPLICATION**" has been carried out under the guidance of Dr. Shivashankara S Assistant Professor Department of CSE for the partiall fulfilment of Bachelor of Engineering in Computer Science and Engineering of Visveswaraya Technological University, Belagavi during the year 2023-24.

Place : Krishnarajpet

Date:

METRI KEERTI SOMASHEKHAR [4GK21CS025]

NAVEEN J R [4GK21CS028]

SHREEDHAR CHANNAPPA TIKOTI[4GK21CS045]

VAISHANVI HANAMANT BHAJANTRI [4GK21CS052]

ABSTRACT

In recent years, the demand for efficient and instantaneous communication tools has significantly increased, spurred by the widespread adoption of digital platforms. Real-time chat applications have become integral to personal, professional, and social interactions, providing users with the ability to communicate instantly regardless of their geographic location. This paper presents the design and implementation of a real-time chat application, emphasizing its architecture, key features, and technological underpinnings.The application utilizes a client-server model, with Socket.IO as the primary framework for facilitating real-time, bidirectional communication between users. Key features of the application include user authentication, message broadcasting, private messaging, presence indicators, and a user-friendly interface. The backend is built using Node.js for its non-blocking, event-driven architecture, ensuring scalability and performance efficiency. MongoDB is employed as the database solution to manage user data and message histories.

Security measures, such as end-to-end encryption and secure user authentication protocols, are integrated to protect user data and maintain privacy. Additionally, the application is designed with a responsive user interface, ensuring accessibility across various devices, including desktops, tablets, and smartphones. This paper discusses the challenges encountered during the development process, including handling concurrency, ensuring low latency, and maintaining data integrity. It also explores future enhancements, such as the incorporation of multimedia messaging, advanced user analytics, and integration with other social platforms.

In conclusion, the developed real-time chat application demonstrates the potential for creating robust, secure, and scalable communication tools that meet the dynamic needs of modern users, paving the way for future innovations in digital communication.

CONTENTS

LIST OF FIGURES **Page No**

CHAPTER 1

INTRODUCTION

The popularity of the MERN stack is ever increasing in the field of web development for frontend and backend applications. The use of React stems from the fact that it was built on Javascript and JS has been the most popular programming language amongst developers for over 25 years. React was built on top of JS by developers from Facebook and made making the UI so much simpler than the normal HTML. It made rendering the components so much easier and faster. For instance, if we had to render a component time and again, we had to hard code it as many times as we needed it. However, in React, we only have to make a component just once and then import it in whichever component we need it.

MERN stack uses the Express JS and Node JS for developing the backend. Earlier, the entire backend client and server part used to be handled by solely Node. However, sending requests and getting back responses from the server was so much more difficult, tedious, painstaking and more complicated. Then, Express was introduced as the de facto application server for building backend API. The server and client applications were made so much easier as there was no need to build the server but it was introduced by Express itself. Using Express has become as easy as importing it in a backend file and creating its instance using the const app = require('express') and then listening it on a given port number just like the backend. The syntax becomes much more simpler with Express than using simple Node. Also, the react-router-dom provides us the Router functionality that helps us define routes in the frontend application without creating a backend directory for the routing part. The router consists of the routing endpoint and the handler function for the request handling portion.

The fourth pillar of the MERN stack is the Node JS framework that has been discussed before with the Express framework. Node JS revolutionized the running of Javascript outside of a web browser. Earlier, it could only be run on the web browser and not in the local run-time environment. Now, it has become easier to run Javascript in the local terminal. Since, it is not a multi-threaded language, Javascript has the concept of promises and callbacks to run code asynchronously and Node Js allows us to do that. The async await syntax allows us to run code asynchronously and has been an improvement over the normal promises and callbacks. Also, async await code makes the code look as though it was synchronous.

This is because it leads to a very common problem in Javascript called the callback hell. This problem makes the asynchronous code look really complicated and for someone who has been using a synchronous multi-threaded language, it can get really difficult to decode and debug it in case of errors. Hence, we use the async-await functional syntax for fetching data from the API and handling it in React templates. React hooks and custom hooks have been used to provide code reusability in the application over the normal usual class programming. Functional components have been built and are preferred to using class components in this application. Class components are still used but the easy syntax of functional components in React make them easier to be understood and easily applicable in the React apps.

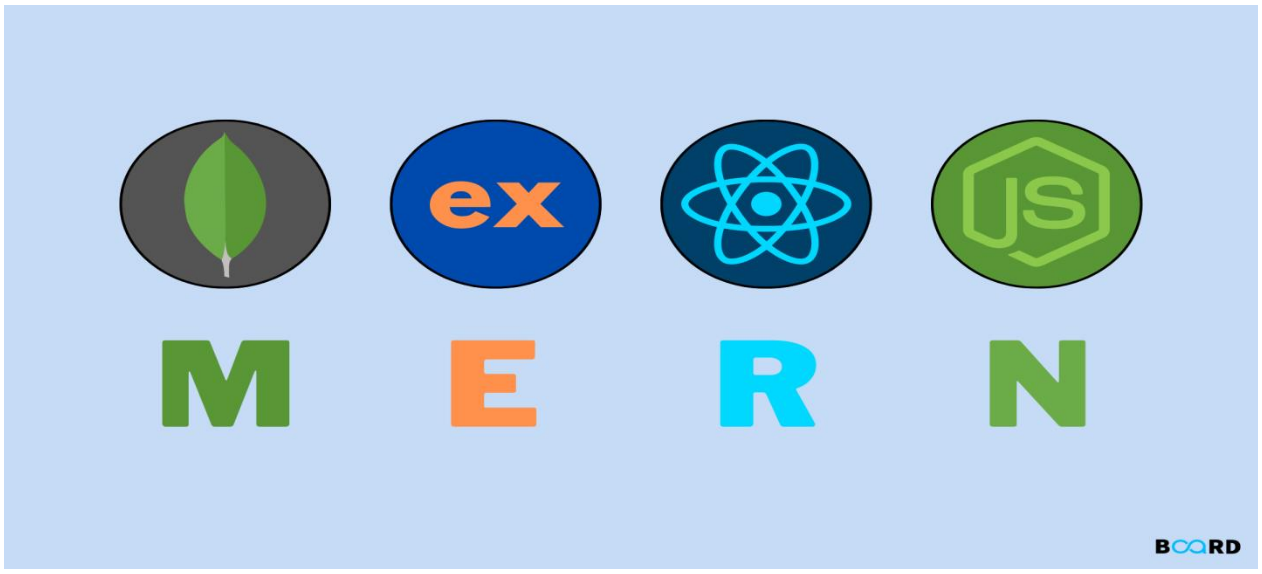


Figure 1.1: MERN Stack

1.1 Problem Statement

- The internship project focuses on building a chatting application in MERN stack using the frontend Javascript technology called React and Express and Node JS for the backend for building of client and server part. Google Firebase is a free backend service that stores the login credentials of the user and the chatroom information.
- This app also gives primary focus to the admin and security permissions to the users. Only, the admin of the chatroom has the permission to edit the name of the chatroom and the chatroom description.
- The project aims to be more than just a chatting application as it allows us to send more than just chats in the server,We can upload upto 5 files in one go. The Message File has the attachment icon to attach files and the send button to finally send the files over on the server.

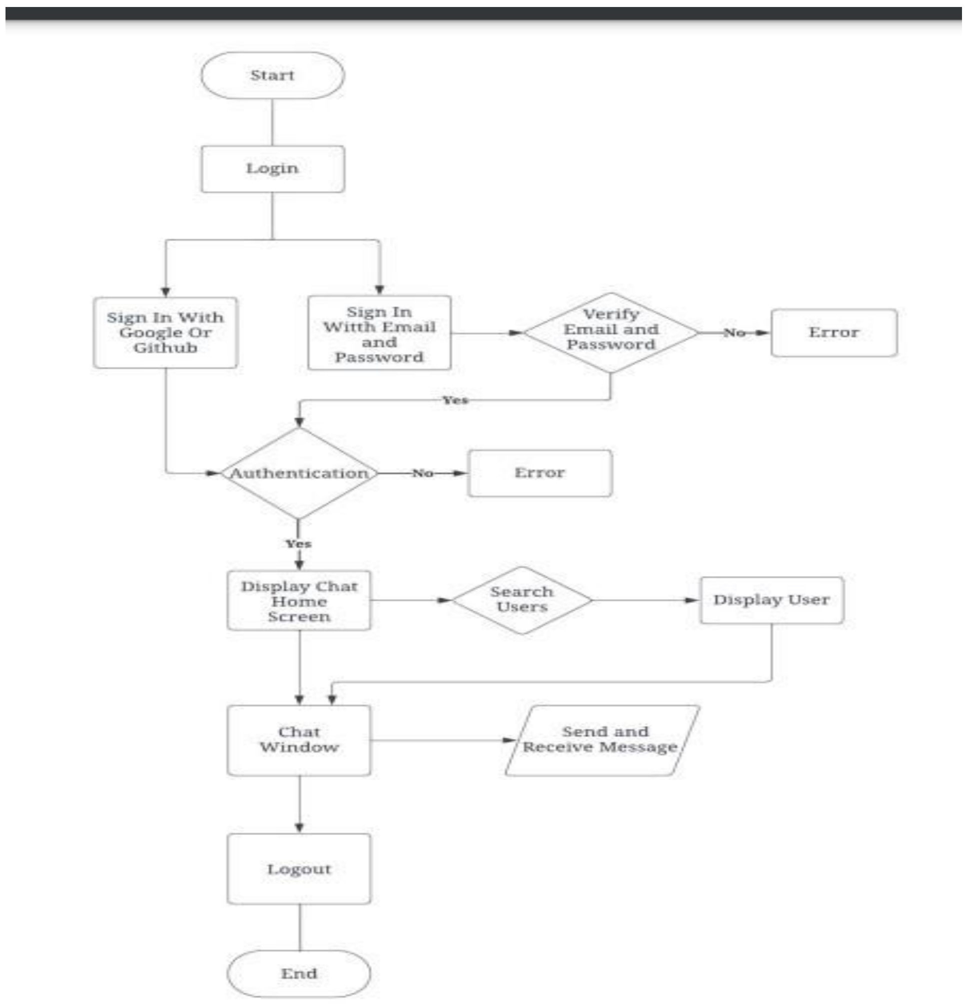
- The user can login the app using the authenticator provided by the Google Firebase. Also, when the user signs out from the app, the unsubscribe() function allows him to be unsubscribed from the database, The timestamp is also added which keeps a record of when the user logged in to the app.
- The user can like certain chats in the chat window and can also remove those likes. A heart emoticon will show up when the user likes the message. This is made possible through the concept of post transactions in React.
- The chats in the application are then organized by date and time when they were sent on the server. After the chats have occupied a certain space on the window screen, the **Load More** button will load the existing chats on the page that were hidden below the most recent chats in the browser.

1.2 Objectives

- The most primary objective of this project is to understand the concept of building applications with the MERN stack (MongoDB, Express, React and Node). In this particularproject, the focus has been built on a chatting application.
- The internship project focuses on developing a talking application in the MERN stack usingthe frontend Javascript technology known as React and Express, as well as Node JS for the backend. Google Firebase is a free backend service that keeps the user's login credentials aswell as chatroom data.
- The admin and security permissions for users are also prioritized in this app. Only the chatroom's administrator has the ability to change the name and description of the chatroom.
- The project aspires to be more than just a talking app by allowing us to submit more than just chats to the server; we can upload up to 5 files at once. In the chat box, the user can choose to like or unlike certain chats. When the user likes the message, a heart emoticon appears. This is feasible because to React's concept of post transactions.
- The user can log in to the app using the Google Firebase authenticator. The unsubscribe() function also allows the user to be unsubscribed from the database when he logs out of the app. A timestamp is also provided to keep track of when the user registered in to the app.
- The attachment icon is used to attach files to the message file, and the send button is usedto transfer the files to the server.

1.3 Methodology

- First and foremost, we must set up our react javascript project. Then, using some specified processes, connect our project to Google Firebase. On Firebase, enable Phone Number Authentication and Real-Time Firestore. Connect to socketio and start the server. Also, connect to the Google and Facebook servers. Create an express server and an API for real- time data transfer among users. Allow geolocation and integrate react native maps to see where your users are.
- Initialise your project in the root directory with the following command after installing Node, Express, NPM (Node Package Manager):
- For styling and conditional rendering, we have used an improvement over the normal css. The technology used for the css is Sass or commonly known as the SCSS. Also, the rsuite library has been used for the rendering of built-in components in React. The React-icons serve as the built-in library for the rendering of icons in the app, Also, styled components have been used in the app almost everywhere. The react-router-dom from React has the special Switch and Route feature to render the different routes in the app. The following can be installed by the following commands using either npm or yarn package managers.



1.4 Scope of future application

- Group Massaging anyone can enter into a Group chatting.
- Needs name for enter into a chat block.
- User can chat easily in their common language.
- Voice recognition makes chatting simpler and easier.
- Includes a pdf reader for reading and showing all types of pdfs shared.

1.5 Scope of the project

Creating a real-time chat application is a significant undertaking that involves several key components and considerations. Here's an overview of the scope for such a project:

1. Requirements Gathering

- **User Requirements:** Define the target audience, user personas, and their needs.
- **Functional Requirements:** List of features such as user authentication, message sending/receiving, real-time updates, notifications, file sharing, etc.
- **Non-Functional Requirements:** Performance, scalability, security, and reliability.

2. Design and Architecture

- **UI/UX Design:** Wireframes, mockups, and user experience flow.
- **System Architecture:** Client-server model, microservices architecture, and database design.
- **Tech Stack Selection:** Frontend (React, Angular, Vue.js), Backend (Node.js, Django, Ruby on Rails), Database (SQL, NoSQL), Real-time Technologies (WebSockets, Firebase, SignalR), Cloud Services (AWS, Azure, GCP).

3. Development

- **Frontend Development:**
    - User interface design and development.
    - Integration with backend APIs.
    - Handling real-time updates using WebSockets or similar technologies.
- **Backend Development:**
    - User authentication and authorization.
    - Message handling (send, receive, store).
    - Implementing real-time communication protocols.
- **Database Management:**
    - Schema design for users, messages, and chat rooms.
    - Optimizing for performance and scalability.
- **Real-time Features:**
    - Implementing WebSocket connections for real-time messaging.
    - Handling concurrent connections and message broadcasting.
- **File Sharing:**
    - Implementing file upload/download features.
    - Storage solutions (cloud storage, local storage).

4. Testing

- **Unit Testing:** Testing individual components and functions.
- **Integration Testing:** Ensuring that different parts of the application work together.
- **End-to-End Testing:** Simulating user scenarios to test the entire application flow.
- **Load Testing:** Ensuring the application can handle a high number of concurrent users.

5. Deployment

- **CI/CD Pipelines:** Setting up continuous integration and continuous deployment pipelines.
- **Server Setup:** Configuring servers and databases for production.
- **Monitoring and Logging:** Setting up monitoring tools to track application performance and errors.

6. Maintenance and Updates

- **Bug Fixes:** Addressing any issues that arise post-deployment.
- **Feature Enhancements:** Adding new features based on user feedback.
- **Performance Optimization:** Continuously improving the application's performance and scalability.

7. Security Considerations

- **Data Encryption:** Ensuring data is encrypted in transit and at rest.
- **Authentication:** Implementing secure user authentication mechanisms (OAuth, JWT).
- **Authorization:** Ensuring proper access controls and permissions.
- **Regular Security Audits:** Conducting regular security assessments and vulnerability scans.

8. Documentation

- **Technical Documentation:** Detailed documentation of the system architecture, APIs, and database schema.

- **User Documentation:** Guides and manuals for end-users on how to use the chat application.

9. Project Management

- **Agile Methodology:** Implementing agile practices such as sprint planning, daily stand-ups, and retrospectives.
- **Task Management:** Using tools like JIRA, Trello, or Asana to manage tasks and track progress.

Timeline and Milestones

- **Initial Planning and Design:** 2-4 weeks
- **Frontend and Backend Development:** 8-12 weeks
- **Testing Phase:** 2-4 weeks
- **Deployment and Monitoring:** 1-2 weeks
- **Maintenance and Updates:** Ongoing

Team Roles

- **Project Manager:** Oversees the project, manages timelines and deliverables.
- **Frontend Developer(s):** Responsible for the user interface and user experience.
- **Backend Developer(s):** Manages server-side logic, database interactions, and real-time communication.
- **QA Engineer(s):** Ensures the application is bug-free and meets quality standards.
- **DevOps Engineer:** Manages deployment, CI/CD pipelines, and infrastructure.

CHAPTER 2

LITERATURE SURVEY

Real-time chat applications have evolved significantly over the years, from basic text messaging systems to complex platforms supporting multimedia communication, including text, voice, and video. These applications have become integral to personal communication, business collaboration, customer support, and social networking. This literature survey explores the various technologies, protocols, architectural designs, scalability considerations, security measures, user experience enhancements, and future trends in real-time chat applications.

2.1 Key Technologies and Protocols

The foundation of real-time chat applications lies in the communication protocols and technologies they utilize. Web Sockets have emerged as a prominent technology, providing full-duplex communication channels over a single TCP connection. This enables real-time data transfer between clients and servers, facilitating instant messaging. WebSockets are widely supported across modern web browsers and are often used in conjunction with HTTP for initial connection setup.

Another essential protocol is XMPP (Extensible Messaging and Presence Protocol), an open-standard communication protocol for message-oriented middleware. XMPP is highly extensible and supports various messaging functions, including presence information, group chats, and file transfers. It has been adopted by several major chat applications due to its robustness and flexibility.

HTTP/2 and HTTP/3 have also contributed to the efficiency of real-time communication. These protocols introduce features such as multiplexing, header compression, and improved flow control, which reduce latency and improve the performance of chat applications. Additionally, RTC (Real-Time Communication) APIs, such as WebRTC, have revolutionized browser-based real-time communication by enabling peer-to-peer data sharing and streaming without the need for intermediary servers.

Backend technologies play a crucial role in handling real-time data. Node.js, with its event-driven architecture, is popular for building scalable chat applications. Its non-blocking I/O model allows for handling numerous simultaneous connections efficiently. Other frameworks like Django and Ruby on Rails are also used, offering different trade-offs in terms of scalability and ease of development.

2.2 Architecture and Design Patterns

The architecture of real-time chat applications typically follows either a client-server model or a peer-to-peer (P2P) approach. In the client-server model, clients communicate via a central server, which manages message delivery, user authentication, and presence information. This model is straightforward to implement and manage, but it can become a bottleneck as the number of users increases.In contrast, peer-to-peer architectures enable direct communication between clients, reducing latency and server load. However, P2P systems can be more complex to implement and may face challenges in ensuring message delivery and maintaining user presence information.

Microservices architecture is gaining popularity for building and scaling chat applications. This approach involves breaking down the application into smaller, independent services that can be developed, deployed, and scaled individually. Each microservice handles a specific aspect of the chat application, such as user authentication, message routing, or file storage. This modularity improves scalability and fault tolerance.Event-driven architecture is another design pattern commonly used in real-time chat applications. It relies on events to trigger updates and notifications, enabling the application to react to changes in real-time. Technologies like Apache Kafka and RabbitMQ are often used to implement event-driven systems, providing reliable message delivery and processing.

2.3 Scalability and Performance

Scalability is a critical consideration for real-time chat applications, as they must handle increasing numbers of users and messages without compromising performance. Load balancing techniques are essential for distributing the load across multiple servers, ensuring that no single server becomes a bottleneck. Various load balancing strategies, such as round-robin, least connections, and IP hash, can be employed depending on the specific requirements.Database optimization is also crucial for handling large volumes of real-time data. NoSQL databases like MongoDB and Cassandra are preferred due to their ability to scale horizontally and handle high-throughput read and write operations. In-memory data stores like Redis are used for caching frequently accessed data, reducing database load and improving response times.

2.4 Security and Privacy

Security is paramount in real-time chat applications, given the sensitive nature of the information exchanged. End-to-end encryption protocols, such as the Signal Protocol, ensure that messages remain confidential and tamper-proof during transmission. This approach encrypts messages on the sender’s device and decrypts them only on the recipient’s device, preventing intermediaries from accessing the content.

Authentication and authorization mechanisms are critical for controlling access to the chat application. OAuth and JWT (JSON Web Tokens) are commonly used for secure user authentication and session management. These mechanisms ensure that only authorized users can access the application and perform actions based on their roles and permissions.

Data privacy regulations, such as the General Data Protection Regulation (GDPR) and the California Consumer Privacy Act (CCPA), impose stringent requirements on how user data is handled. Compliance with these regulations necessitates robust data protection measures, including user consent management, data anonymization, and secure data storage.

2.5 User Experience (UX) and Interface Design

The success of a real-time chat application heavily depends on its user experience (UX) and interface design. Responsive design is crucial to ensure the application functions seamlessly across various devices and screen sizes. A well-designed user interface (UI) enhances user engagement by providing intuitive navigation, clear visual cues, and interactive elements.

Accessibility is another critical aspect, as chat applications should be usable by individuals with disabilities. This involves implementing features like screen reader compatibility, keyboard navigation, and adjustable text sizes to accommodate different user needs.

2.6 Challenges and Solutions

Real-time chat applications face several challenges, including latency, concurrency, and fault tolerance. Reducing latency is essential for providing a seamless user experience. Techniques such as content delivery networks (CDNs), edge computing, and optimizing network protocols can help minimize delays.

In contrast, peer-to-peer architectures enable direct communication between clients, reducing latency and server load. However, P2P systems can be more complex to implement and may face challenges in ensuring message delivery and maintaining user presence information.

Fault tolerance ensures that the application remains functional despite failures. This involves implementing redundancy, failover mechanisms, and automated recovery processes to handle unexpected issues. Monitoring and logging tools are also essential for detecting and addressing problems promptly.

2.7 Case Studies

Several real-time chat applications serve as benchmarks for best practices and innovations in the field. WhatsApp, for example, employs a sophisticated architecture that includes WebSockets, Erlang-based backend systems, and end-to-end encryption to ensure reliable and secure communication. Slack integrates with numerous third-party services and APIs, demonstrating the importance of extensibility and collaboration features in modern chat applications. Microsoft Teams showcases the integration of enterprise-level features, such as advanced security controls, compliance tools, and extensive administrative options.

2.8 Future Trends

The future of real-time chat applications is shaped by advancements in artificial intelligence (AI), augmented reality (AR), virtual reality (VR), and quantum computing. AI-powered chatbots are becoming increasingly prevalent, offering automated responses, customer support, and personalized interactions. AR and VR integration promises immersive communication experiences, enabling users to interact in virtual environments. Quantum computing, while still in its early stages, has the potential to revolutionize encryption and real-time data processing, offering unprecedented levels of security and performance.

CHAPTER 3

TECHNOLOGIES USED

3.1 Mongo DB

Mongo DB is a leading NoSQL database known for its document-oriented architecture, storing data in flexible JSON-like documents called BSON (Binary JSON). Its schema-less design allows for dynamic and evolving data structures within collections, eliminating the need for predefined schemas and enabling faster iteration in development. MongoDB excels in scalability and performance through features like indexing, sharding (horizontal scaling), and replication (high availability).

The database is widely adopted across industries for its ability to handle large volumes of data and high throughput applications. It supports a rich query language and aggregation framework, allowing for complex queries and computations directly within the database. MongoDB's JSON/BSON format facilitates seamless integration with modern application stacks and frameworks.

Figure 3.1: Mongo DB

3.2 Express.js

Express.js is a minimal and flexible web application framework for Node.js, designed to simplify the creation of web servers and APIs. It provides a robust set of features, including routing, middleware support, and easy integration with various templating engines for dynamic content generation.At its core, Express.js facilitates the handling of HTTP requests and responses, allowing developers to define routes based on HTTP methods and URL patterns. Middleware functions in Express enable tasks such as parsing incoming request data, authentication, logging, and error handling, enhancing code modularity and reusability.Express does not impose strict conventions or a rigid structure, which gives developers the freedom to organize their application logic as they see fit. This flexibility, combined with its extensive ecosystem of plugins and middleware, makes Express suitable for building both small-scale applications and large, complex systems.

Figure 3.2: Express.js

3.3 React.js

React.js is a JavaScript library primarily used for building user interfaces (UIs) in web applications. It enables developers to create UI components that are reusable, declarative, and efficient. React uses a virtual DOM to optimize updates to the actual DOM, resulting in faster rendering and improved performance.

Its component-based architecture promotes code modularity, making applications easier to maintain and scale. React employs JSX, a syntax extension that allows embedding HTML-like code within JavaScript, facilitating a seamless integration of UI components and logic. It follows a unidirectional data flow, where data updates propagate from parent to child components, ensuring predictable and manageable state management. React's popularity is driven by its simplicity, flexibility, and strong ecosystem, which includes libraries, tools, and community support. It is widely adopted by developers for building interactive and responsive web applications, leveraging its efficient rendering, component reusability, and robust developer tools to enhance productivity and user experience.

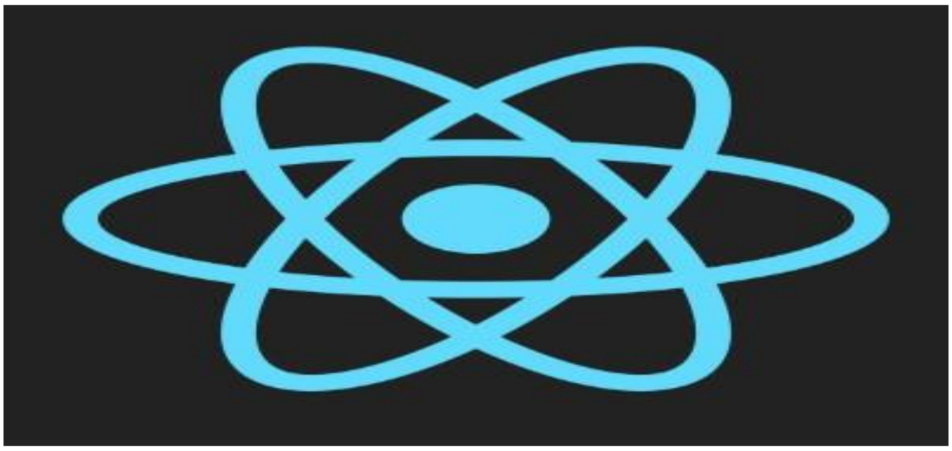


Figure 3.3: React.js

3.4 Node.js

Node.js is a runtime environment built on Chrome's V8 JavaScript engine that allows developers to execute JavaScript code outside of a web browser. It is designed to build scalable network applications, primarily focusing on server-side scripting to produce dynamic web page content before it is sent to the user's web browser.Node.js utilizes an event-driven, non-blocking I/O model, which makes it lightweight and efficient. This is particularly beneficial for data-intensive real-time applications that run across distributed devices. The non-blocking I/O model means that Node.js can handle multiple requests simultaneously, without waiting for any single operation to complete.

This is achieved through callbacks, promises, and async/await patterns, which help manage asynchronous operations.Additionally, Node.js has a vast ecosystem of libraries and modules available through npm (Node Package Manager), allowing developers to easily extend the functionality of their applications. It supports a wide range of use cases, including web servers, APIs, microservices, and real-time applications like chat apps and online gaming.

Figure 3.4: Node.js

CHAPTER 4

REQUIREMENETS SPECIFICATION

A real-time chat application encompasses several critical theoretical components to ensure effective and efficient communication. The requirements specification can be divided into key areas: user management, real-time communication, data handling, security, scalability, and cross-platform compatibility.

User Management:

Authentication and Authorization: Users must be able to securely register and log in. This is typically achieved using authentication protocols like OAuth 2.0 or JWT (JSON Web Tokens) for session management. Proper authorization mechanisms ensure that users have access only to their data and functionalities.

Real-Time Communication:

WebSockets: For real-time message transmission, WebSockets provide a persistent connection between the client and server, allowing for bidirectional data flow. This ensures messages are delivered instantly without the need for repeated HTTP requests.

Message Handling: The system must efficiently manage various message types (text, images, files) and ensure reliable delivery, often using acknowledgments and retries.

Data Handling:

Real-Time Database: Databases such as Firebase or MongoDB with real-time capabilities are used to store messages and user data, ensuring immediate synchronization across clients.

Data Consistency and Integrity: Mechanisms must be in place to maintain data consistency and integrity, especially during high load conditions or concurrent access.

Security:

End-to-End Encryption: Ensures that messages are encrypted on the sender’s device and decrypted only on the recipient’s device, safeguarding privacy.

Secure Data Storage: Encryption of data both at rest and in transit is essential. Security measures must protect against vulnerabilities such as XSS, CSRF, and SQL injection.

Scalability:

Load Balancing and Horizontal Scaling: To handle numerous concurrent users, the system should distribute the load efficiently across servers. Horizontal scaling allows adding more servers to handle increased traffic.

Microservices Architecture: This approach divides the application into smaller, independent services that can be scaled individually based on demand.

Cross-Platform Compatibility:

Consistent User Experience: The application should provide a seamless experience across web and mobile platforms. This involves responsive design and possibly using frameworks like React Native for mobile development.

Push Notifications: Effective implementation of push notifications ensures users are alerted to new messages or updates in real-time.

Logging and Monitoring:

System Performance Monitoring: Real-time monitoring tools track system performance and user activity to detect and resolve issues promptly.

Error Logging: Comprehensive logging mechanisms record errors and critical events, aiding in maintenance and troubleshooting.

Backup and Recovery:

Regular Backups: Periodic backups of the database prevent data loss and ensure recovery in case of failures.

Disaster Recovery Plan: A well-defined plan for quick recovery from system failures or data corruption ensures minimal downtime and data integrity.

4.1 Functional Requirements:

Functional requirements for a real-time chat application define the specific behaviours and functions the system must perform:

- **User Authentication**: Register, log in, and log out securely.
- **Real-Time Messaging**: Send and receive messages instantly using WebSockets.
- **User Profiles**: Create and update profiles with personal information and status.
- **Group Chats**: Create, manage, and participate in group conversations.
- **Media Sharing**: Upload and share images, videos, and files.
- **Notifications**: Real-time alerts for new messages and online status changes.
- **Message Storage**: Persist message history for retrieval.
- **Security**: Encrypt data in transit and at rest, manage sessions with JWT tokens.

4.2 Non Functional Requirements

Non-functional requirements define the system's operational attributes and quality characteristics:

1. **Performance**:
    - Handle at least 10,000 concurrent users with minimal latency.
    - Ensure message delivery within 2 seconds.
2. **Scalability**:
    - System must support scaling to accommodate increasing user load without performance degradation.
3. **Reliability**:
    - Achieve 99.9% uptime with failover mechanisms.
    - Implement robust error handling and recovery procedures.
4. **Security**:
    - Use SSL/TLS for secure data transmission.
    - Implement data encryption for messages at rest.
    - Ensure secure user authentication and authorization with JWT tokens.
5. **Usability**:
    - Intuitive user interface with easy navigation.
    - Accessible on various devices (desktop, mobile, tablets).
6. **Maintainability**:
    - Modular architecture for easy updates and bug fixes.
    - Comprehensive documentation for developers.
7. **Compliance**:
    - Adhere to GDPR and other relevant data protection regulations.

8. **Availability**:

    - Ensure system is accessible 24/7 with minimal downtime for maintenance.

9. **Compatibility**:

    - Support for major web browsers and mobile operating systems (iOS, Android).

4.3 Hardware Requirements

Hardware is a term which refers to all the physical parts that make up a computer. Various devices which are essential to form a hardware are called components.

Following are the hardware specifications which were required to develop this project: Components include:

- Computer, mouse, 2 gm of RAM for smooth functioning of application.
- Pendrive of 100 GB or more.
- Internet connection and server connectivity.

4.4 Software Requirements

Software can be termed as the group of instruction or command used by the computer to accomplish the given task. It can be said as a set of instructions or programs instructing a computer to do specific task. Software, in general term is used to describe the computer programs.

Following are the software specifications that is required to develop this project is as follows:

- Operating System: Microsoft Windows 10 or above versions.
- Language Used (Front End): Visual Studio Code [Index.html] (Version: 1.57.1 (user setup) Commit: 507ce72a4466fbb27b715c3722558bb15afa9f48
- Electron: 12.0.7
- Chrome: 89.0.4389.128
- Node.js: 14.16.0 V8: 8.9.255.25-electron.0
- OS: Windows_NT x64 10.0.17134).

CHAPTER 5

SYSTEM ANALYSIS

5.1 Feasibility Study:

Feasibility study is the preliminary study undertaken before the real work of the project starts to ascertain the like hood of the project success. It analyses the possible solutions to a problem and a recommendation on the best solutions to use. It involves the evaluation that how the solution will fit into the corporation. A Feasibility study is defined as a evolution or analysis of the potential impacts of a proposed project or system. A feasibility study is conducted to assist decision makers in determining whether or not to implement a particular project or system. On the basis of result of the initial study, feasibility study takes place. The feasibility study is basically the proposed system in the lights of its workability, meeting user requirements, and effective use of resources and of course, cost effectiveness. The main goal of feasibility study is not to solve the problem but to achieve this scope. In the process of feasibility study, the cost and benefits are estimated with the greater accuracy. It evaluates the benefits of the new system. The feasibility study will contain the extensive data related to financial and operational impact and will include advantage and disadvantages of both current situation and plan. The aim of feasibility study is to see whether it is possible to develop a reasonable cost. At the end of feasibility study a decision is taken whether or proceed or not. Feasibility study is to determine various solution of the problem and then picking up one of the best solutions. It is the measure of how beneficial the development of 13 information system will be to an organization. The study also shows the sensitivity of business to change in the basic assumption.

5.2 Economic Feasibility

For any system if the expected benefits equal or exceed the expected costs, the system can be judged to be economically feasible. In economic feasibility, cost benefit analysis is done in which expected costs and benefits are evaluated. Economic analysis is used for evaluating the effectiveness of the proposed system. In this type of feasibility study, the most important is cost and benefit analysis. As the name suggests, it is as analysis of the costs to be incurred in the system and benefits derivable out of the system.

5.3 Technical Feasibility

- In technical feasibility the following issues are taken into consideration.

- Whether the required technology is available or not.
- Whether the required resources are available like manpower, programmers, testers and debuggers, software and hardware.

5.4 Social feasibility

The affect that a proposed system may have on the social system in the project environment is addressed in the social feasibility. It may happen that particular category of employees may be short or not available as a result of ambient structure. The influence on the social status of the participants by the project should be evaluated on order to guarantee compatibility. It must be identified that the employees in the particular industries may have specific status symbols within the society.

5.5 Behavioral feasibility

It includes how strong the reaction of staff will be towards the development of new system that involves computer's use in their daily work. So resistant to change is identified. It considers human issue. All system development projects introduce change, and people generally resist change. Over resistance from employees may take the form of subrogating the new system (e.g., entering data incorrectly) or 14 deriding the new system to anyone who will listen. Convert resistance typically occurs when employees simply do their jobs using their old methods. Behavioral feasibility is concerned with assessing the skills and the training needed to use the new is. In some organizations, a proposed system may require mathematical or linguistic skills beyond what the workforce currently processes. In other words, a workforce may simply need to improve their skills. Behavioral feasibility is as much about "can they use it" as it is about "will they use it".

CHAPTER 6

SYSTEM DESIGN

Designing a real-time chat application involves several key components. The front end typically uses web technologies like HTML, CSS, and JavaScript, often with frameworks like React or Vue.js for responsiveness. The back end might employ Node.js with Express for handling HTTP requests, and Web Socket for real-time communication. A NoSQL database like MongoDB can be used for storing messages and user data. User authentication can be managed with JWT tokens. The system should also include a message broker like Redis or RabbitMQ for handling message queues, ensuring smooth delivery. Security measures, such as HTTPS and data encryption, are essential to protect user privacy.

6.1 Functional Design Concept

A functional design concept for a real-time chat application encompasses user interactions, system operations, and data flow. Users start by authenticating through a secure login or registration process, utilizing email, passwords, or social media accounts. Once authenticated, users access the chat interface displaying real-time messages and user statuses.The application utilizes WebSocket connections to enable instant message delivery and receipt acknowledgments. Messages are stored in a NoSQL database, ensuring efficient retrieval and scalability. The user interface features individual and group chat options, allowing for seamless conversation management.

Group chat functionality includes creating and managing groups, adding or removing members, and assigning roles. Notifications for new messages, mentions, and online status changes enhance user engagement and awareness.The application supports media sharing, allowing users to upload and share images, videos, and documents. These files are stored securely in cloud storage solutions, ensuring accessibility and reliability.Security measures are integral, with data encryption during transmission via SSL/TLS and JWT tokens for session management. Rate limiting is implemented to prevent spam and ensure system integrity.Scalability is achieved through load balancing and database sharding, allowing the application to handle increasing user loads and maintain performance. This comprehensive functional design ensures a robust, user-friendly real-time chat experience.

6.2 Input and Output Design

6.2.1 Input Design:

1. **User Authentication Inputs**:

   - **Login/Sign-Up Forms**: Fields for email, password, and optional social media login buttons.
   - **Validation**: Immediate feedback on input errors (e.g., invalid email format, weak password).

2. **Chat Interface Inputs**:

   - **Message Input Box**: Text box for typing messages with a send button or Enter key submission.
   - **Attachment Options**: Buttons for uploading files, images, and videos.
   - **Emoji/Sticker Selection**: Dropdown or popup for selecting emojis or stickers.

3. **User Profile Inputs**:

   - **Profile Update Forms**: Fields for updating username, profile picture, status, and other personal details.
   - **Settings**: Toggles or dropdowns for notification preferences and privacy settings.

4. **Group Management Inputs**:

   - **Group Creation Forms**: Fields for group name, description, and member selection.
   - **Member Management**: Options for adding/removing members and assigning roles.

6.2.2 Output Design:

1. **Authentication Outputs**:

   - **Success/Failure Messages**: Feedback on login or registration status.
   - **Token Generation**: JWT tokens for session management.

Chat Outputs:

- **Real-Time Messages**: Display of sent and received messages with timestamps.
- **Read Receipts**: Indicators showing message delivery and read status.
- **Notifications**: Real-time alerts for new messages, mentions, and online status changes.

2. **User Profile Outputs**:

- **Profile Display**: Updated user information including profile picture and status.
- **Settings Confirmation**: Feedback on saved preferences.

3. **Group Chat Outputs**:

- **Group Messages**: Real-time display of group conversations.
- **Member Activity**: Notifications and indicators of member actions (joining, leaving, etc.).

6.3 SYSTEM DEVELOPMENT METHOLOGY

System development methodology refers to the structured approach used to plan, develop, and deploy software systems. Here are some commonly used methodologies:

6.3.1 Waterfall Model

Description:

- A linear and sequential approach where each phase must be completed before the next begins.
- Phases include Requirements, Design, Implementation, Testing, Deployment, and Maintenance.

Advantages:

- Clear milestones and deliverables.
- Easy to manage due to rigidity.

Disadvantages:

- Inflexible to changes once a phase is completed.
- Late testing phase may result in finding errors late in the process.

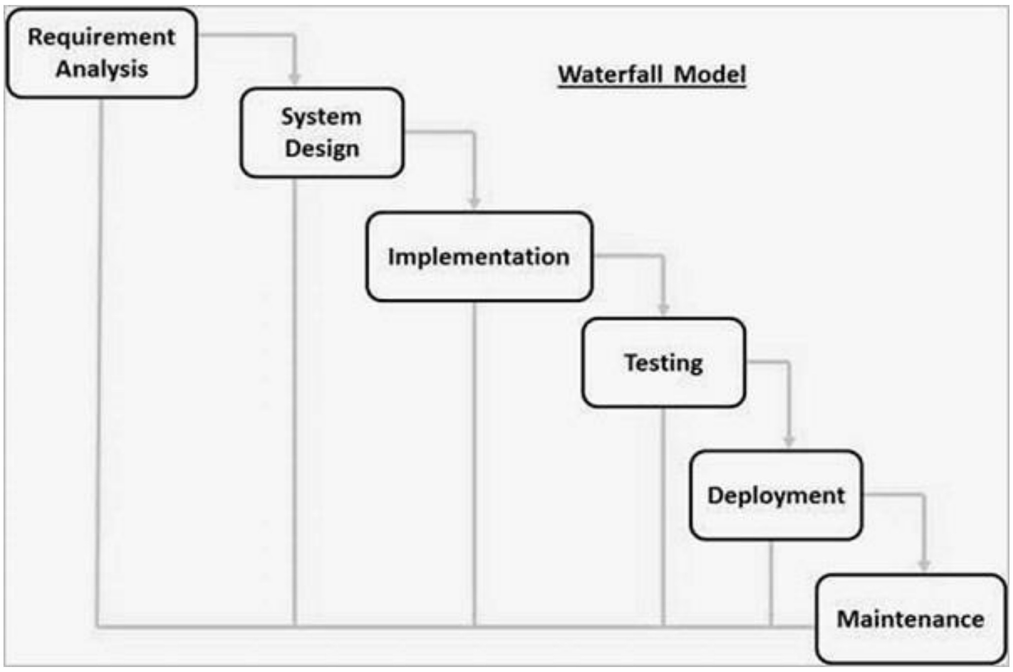


Figure 6.1:waterfall Model

6.3.2 Agile Methodology

Description:

- An iterative approach that focuses on collaboration, customer feedback, and small, rapid releases.
- Popular frameworks include Scrum and Kanban.

Advantages:

- Flexibility to changes and continuous improvement.
- Increased customer satisfaction through regular updates and feedback.

Disadvantages:

- Requires constant communication and collaboration.
- Can be challenging to predict end outcomes early on.

CHAPTER 7

IMPLEMENTATION

Implementing a real-time chat application involves several steps, including setting up the front-end, back-end, real-time communication, database management, and ensuring security and scalability. Here's a detailed outline:

7.1 Front-End Development:

- Use frameworks like React or Vue.js for building an interactive user interface.
- Develop components for chat windows, user lists, and input fields, ensuring responsiveness across devices.

7.2 Back-End Development:

- Employ Node.js with Express for managing HTTP requests and WebSocket connections.
- Create RESTful APIs for user authentication, profile management, and message retrieval.

7.3 Database Management:

- Choose a NoSQL database like MongoDB for scalability and performance.
- Design schemas for user data, messages, and groups, implementing CRUD operations.

7.4 User Authentication:

- Utilize JWT tokens for secure authentication and session management.
- Optionally integrate OAuth for social logins.

7.5 Real-Time Communication:

- Use Web Socket protocol to facilitate instant messaging and real-time updates.
- Implement logic for message broadcasting and user presence notifications.

7.6 Media Sharing:

- Enable file uploads and sharing of images, videos, and documents.
- Store media files securely using cloud storage solutions like AWS S3.

7.7 **Security:**

- Ensure data encryption in transit (SSL/TLS) and at rest.
- Implement input validation and sanitization to prevent security vulnerabilities.

7.8 Scalability:

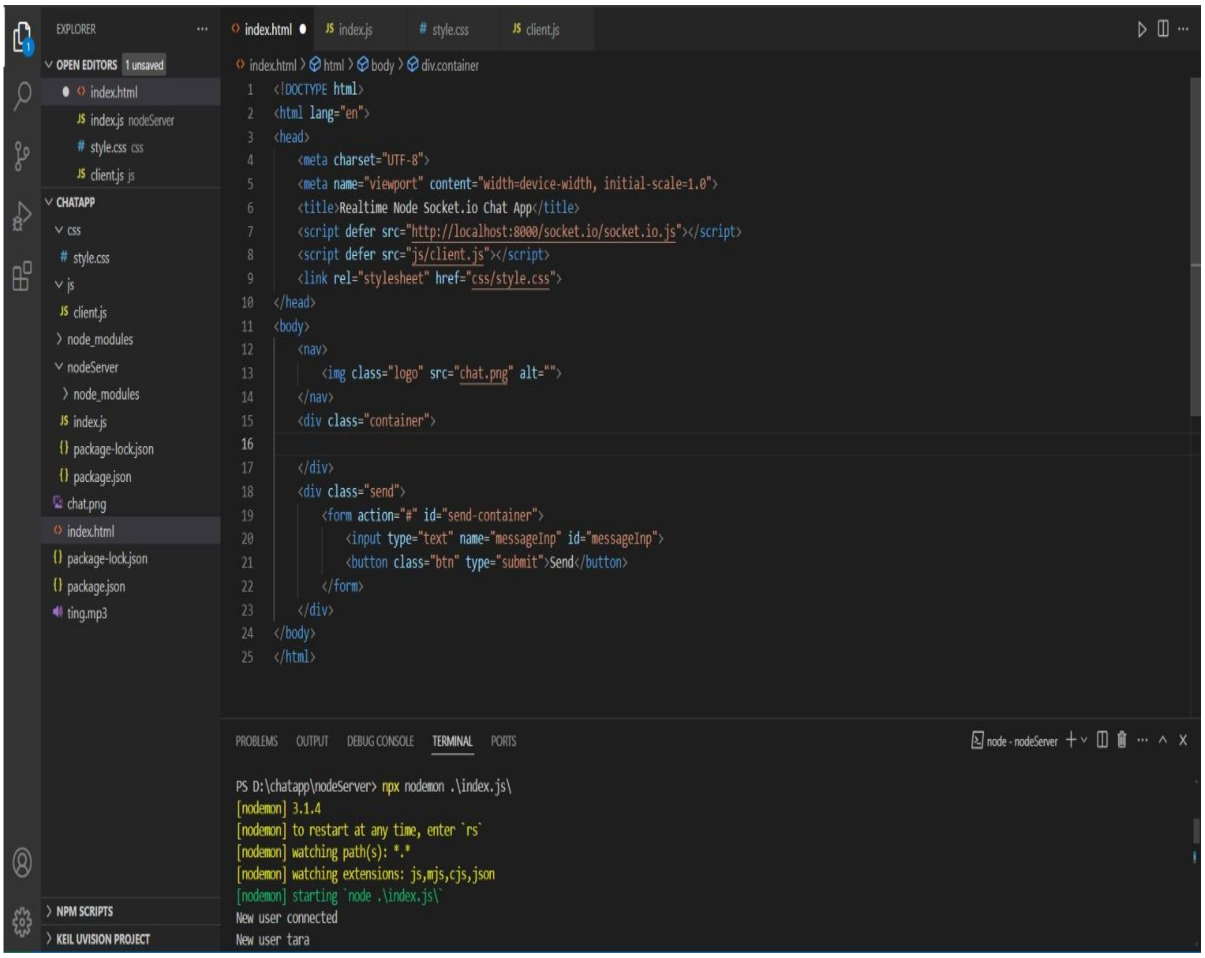
- Use load balancing and database sharding to handle increased traffic and improve performance.
- Implement caching strategies to enhance response times.

SOURCE CODE:

```
<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Realtime Node Socket.io Chat App</title>

    <script defer src="http://localhost:8000/socket.io/socket.io.js"></script>

    <script defer src="js/client.js"></script>

    <link rel="stylesheet" href="css/style.css">

</head>

<body>

    <nav>

 <img class="logo" src="chat.png" alt="">
```

```
    </nav>
    <div class="container">
         </div>
    <div class="send">
        <form action="#" id="send-container">
            <input type="text" name="messageInp" id="messageInp">
            <button class="btn" type="submit">Send</button>
        </form>
    </div>
</body>
</html>
[21:16, 7/26/2024] Vaish,CS: index.js
[21:16, 7/26/2024] Vaish,CS: .logo{
    display:block;
    margin: auto;
    width: 50px;
    height: 50px;
}
.container{
    max-width: 955px;
    border: 2px solid black;
    margin:auto;
    height: 60vh;
    padding:33px;
    overflow-y:scroll;
    margin-bottom: 23px;
}
.message{
    background-color: grey;
    width: 24%;
    padding:10px;
    margin:17px 3px;
    border:2px solid black;
    border-radius: 10px;
}
.right{
    float: right;
    clear: both;
}
.left{
    float: left;
    clear: both;
}
#send-container{
    display: block;
    margin: auto;
    text-align: center;
    max-width: 985px;
    width: 100%;
}
```

```
#messageInp{
    width: 91%;
    border: 2px solid black;
    border-radius: 6px;
    height: 34px;
}
.btn{
    cursor: pointer;
    border:2px solid black;
    border-radius: 6px;
    height: 34px;
}
body{
    height: 100vh;
    background-image: linear-gradient(rgb(255,255,255),purple);
}
[21:16, 7/26/2024] Vaish,CS: style.css
[21:16, 7/26/2024] Vaish,CS: const socket = io('http://localhost:8000');
const form = document.getElementById('send-container');
const messageInput = document.getElementById('messageInp');
const messageContainer = document.querySelector('.container');
var audio = new Audio('D:\chatapp\ting.mp3');
const appendMessage = (message, position) => {
    const messageElement = document.createElement('div');
    messageElement.innerText = message;
    messageElement.classList.add('message');
    messageElement.classList.add(position);
    messageContainer.append(messageElement);
    if(position == 'left'){
        audio.play();
    }
};
const name = prompt('Enter your name to join');
socket.emit('new-user-joined', name);
socket.on('user-joined', name => {
    appendMessage(${name} joined the chat, 'right');
});
socket.on('receive', data => {
    appendMessage(${data.name}: ${data.message}, 'left');
});
socket.on('left', name => {
    appendMessage(${name} left the chat, 'left');
});
form.addEventListener('submit', (e) => {
    e.preventDefault();
    const message = messageInput.value;
    appendMessage(You: ${message}, 'right');
    socket.emit('send', message);
    messageInput.value = '';
});
```



```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Realtime Node Socket.io Chat App</title>
    <script defer src="http://localhost:8000/socket.io/socket.io.js"></script>
    <script defer src="js/client.js"></script>
    <link rel="stylesheet" href="css/style.css">
</head>
<body>
    <nav>
        <img class="logo" src="chat.png" alt="">
    </nav>
    <div class="container">

    </div>
    <div class="send">
        <form action="#" id="send-container">
            <input type="text" name="messageInp" id="messageInp">
            <button class="btn" type="submit">Send</button>
        </form>
    </div>
</body>
</html>
```

```
PS D:\chatapp\nodeServer> npx nodemon .\index.js\
[nodemon] 3.1.4
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node .\index.js\`
New user connected
New user tara
```

Figure 5.1:Index.html

CHAPTER 8

TESTING

Testing a real-time chat application involves several strategies to ensure functionality, performance, and security. Here’s a structured approach:

8.1 Unit Testing

- **Purpose**: Validate individual components and functions in isolation.
- **Tools**: Use testing frameworks like Jest or Mocha for JavaScript.
- **Focus Areas**: Test Web Socket connection handling, message formatting, and API endpoints.

8.2 Integration Testing

- **Purpose**: Verify that different modules work together as expected.
- **Tools**: Employ tools like Supertest for API testing and Web Socket testing libraries.
- **Focus Areas**: Ensure the interaction between front-end components, back-end services, and the database is correct. Test the complete flow of sending and receiving messages, user authentication, and media uploads.

8.3 Functional Testing

- **Purpose**: Validate that the application performs its intended functions.
- **Tools**: Use end-to-end testing tools like Selenium or Cypress.
- **Focus Areas**: Test user scenarios such as logging in, sending and receiving messages in real-time, group chat functionalities, and media sharing. Ensure all features work as intended from the user's perspective.

8.4 Performance Testing

- **Purpose**: Assess the application’s performance under various conditions.
- **Tools**: Use tools like JMeter or LoadRunner.

- **Focus Areas**: Test the system’s ability to handle a high number of concurrent users, message throughput, and latency. Assess server load and response times during peak usage.

8.5 Security Testing

- **Purpose**: Identify and fix security vulnerabilities.
- **Tools**: Employ security testing tools like OWASP ZAP or Burp Suite.
- **Focus Areas**: Test for vulnerabilities like SQL injection, XSS, and CSRF. Ensure data encryption, secure authentication, and authorization mechanisms are robust.

8.6 Usability Testing

- **Purpose**: Ensure the application is user-friendly.
- **Tools**: Conduct manual testing or use tools like UserTesting.
- **Focus Areas**: Evaluate the intuitiveness of the interface, ease of navigation, and overall user experience.

8.7 Regression Testing

- **Purpose**: Ensure new changes do not adversely affect existing functionality.
- **Tools**: Automate tests using frameworks like Selenium or Cypress.
- **Focus Areas**: Re-run previously passed test cases to confirm that existing features are still functioning after updates or bug fixes.

8.8 Stress Testing

- **Purpose**: Evaluate the application’s behavior under extreme conditions.
- **Tools**: Use tools like Apache JMeter or custom scripts.
- **Focus Areas**: Test the system’s stability and error handling when subjected to maximum load or unexpected scenarios.

CHAPTER 9

EXPERIMENTAL RESULTS

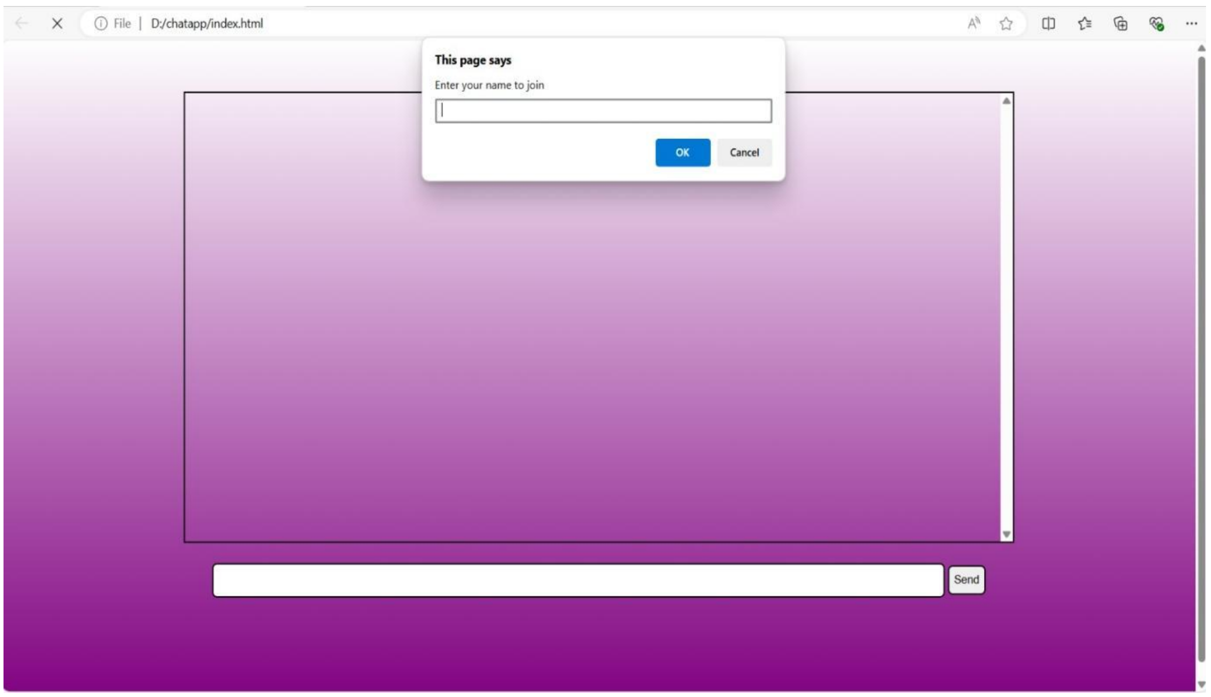


Figure 9.1:Output Result

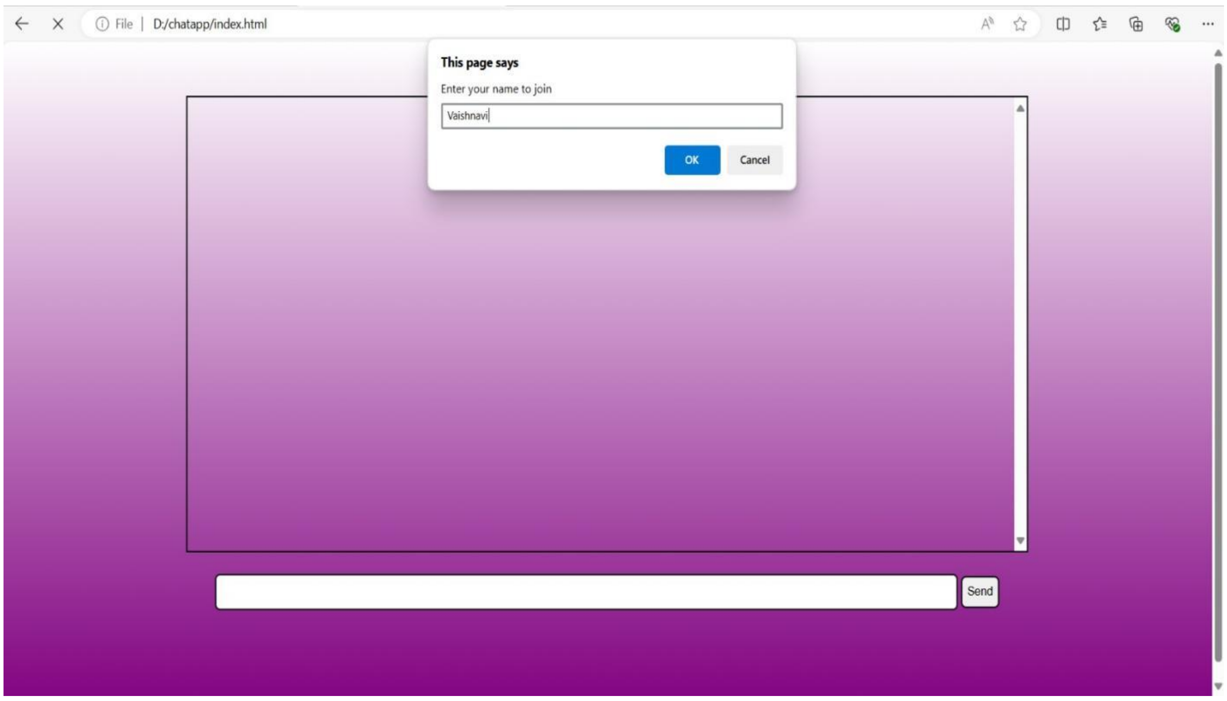
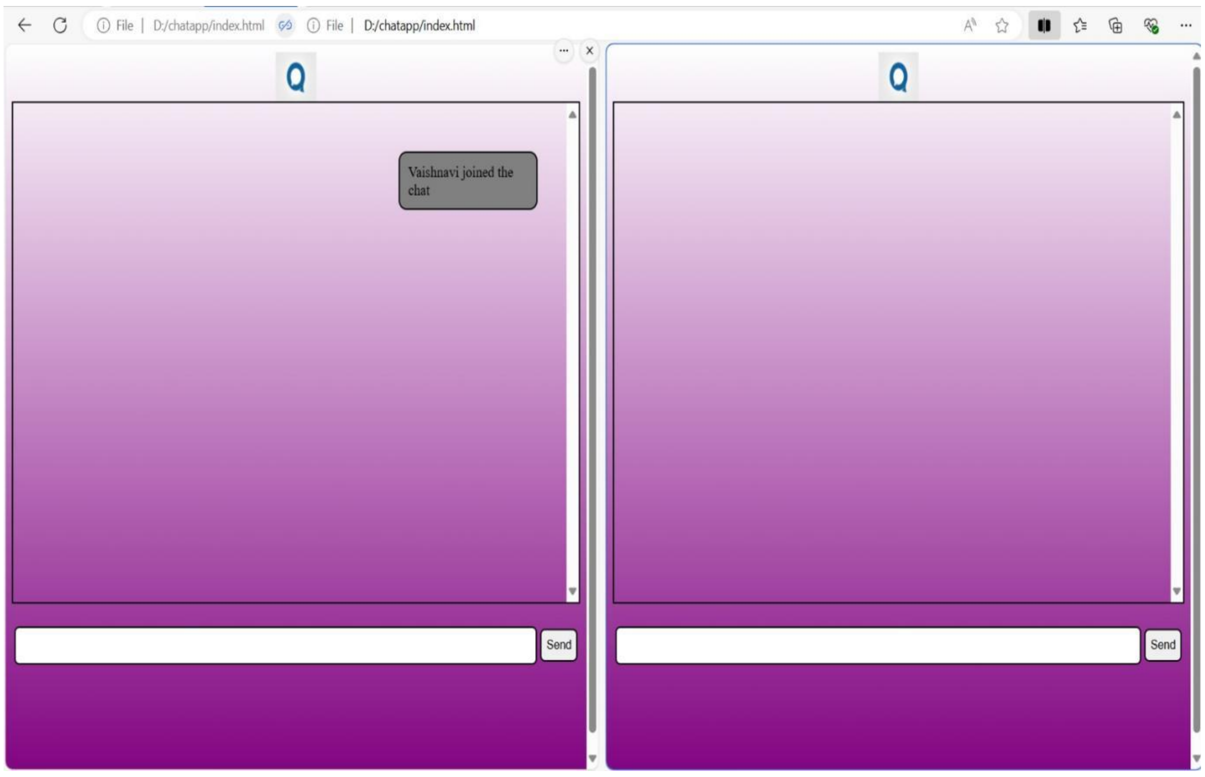
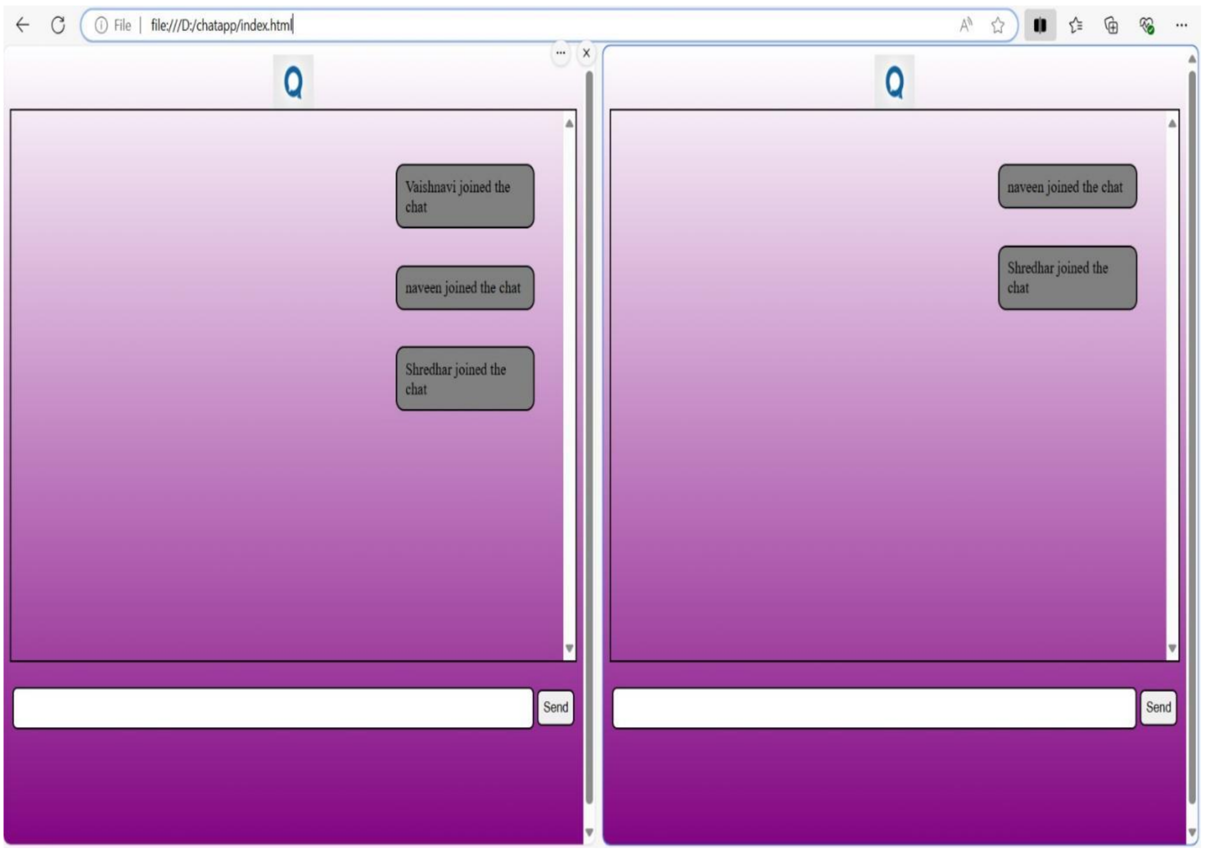


Figure 9.2: Joining the chat application



Vaishnavi joined the chat

Figure 9.3:Joined the chat



Vaishnavi joined the chat

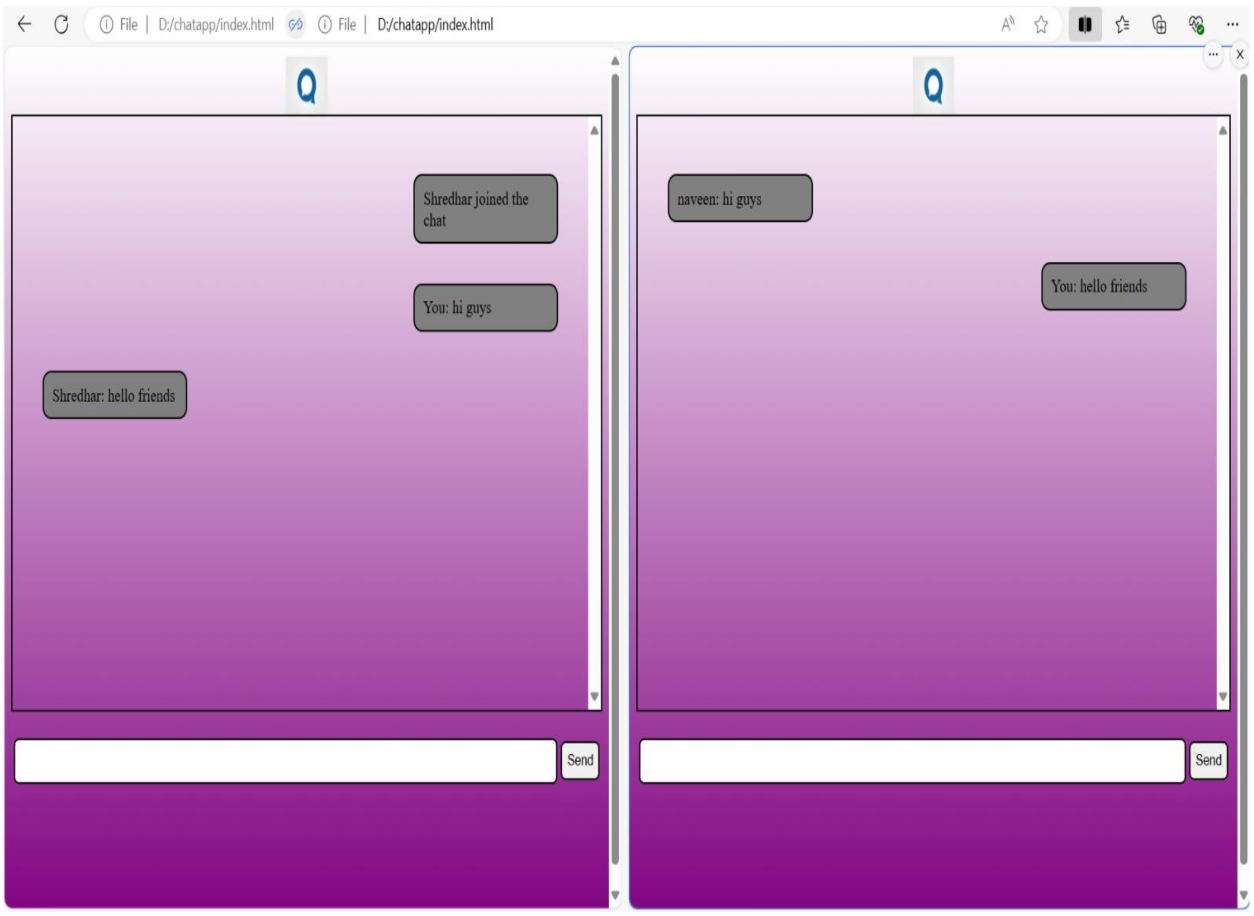
naveen joined the chat

Shredhar joined the chat

naveen joined the chat

Shredhar joined the chat

Figure 9.4:More user joined the chat



Shredhar joined the chat

You: hi guys

Shredhar: hello friends

naveen: hi guys

You: hello friends

Figure 9.5:Users texting

CHAPTER 10

CONCLUSIONS AND FUTURE SCOPE

The sole aim of the project is to understand the concept and basics of MERN stack web development application by building an advanced chatting application. The way it is different from the other applications is in the following ways:

- The internship project focuses on building a chatting application in MERN stack using the frontend Javascript technology called React and Express and Node JS for the backend for building of client and server part. Google Firebase is a free backend service that stores the login credentials of the user and the chatroom information.
- This app also gives primary focus to the admin and security permissions to the users. Only, the admin of the chatroom has the permission to edit the name of the chatroom and the chatroom description.
- The project aims to be more than just a chatting application as it allows us to send more than just chats in the server. We can upload upto 5 files in one go. The Message File has the attachment icon to attach files and the send button to finally send the files over on the server.
- The user can login the app using the authenticator provided by the Google Firebase. Also, when the user signs out from the app, the unsubscribe() function allows him to be unsubscribed from the database, The timestamp is also added which keeps a record of when the user logged in to the app.
- The user can like certain chats in the chat window and can also remove those likes. A heart emoticon will show up when the user likes the message. This is made possible through the concept of post transactions in React. There is still a lot of backend that needs to be incorporated into making the app faster and more reliable than other chat applications in the market. For example, the video calling feature needs to be incorporated for multiple people to video chat at the same time. Also, chat reactions, dark mode for the entire app are some of the features that need to be coded in the app.

Future Scope

- There is still a lot of backend work to be done in order to make the app faster and more dependable than other chat apps available. For example, if numerous individuals want to video chat at the same time, the video calling feature must be included.
- Chat reactions, as well as a dark mode for the full app, are some of the features that must be coded and incorporated.
- Animation Libraries such as Framer Motion have to be used to provide UI strength and more unique design to the app,
- Alongwith Firebase database, the app needs to be tested on other NOSQL platforms such as MongoDB Atlas and NOSQL Booster to see if is compatible with other platforms,
- The app needs to load faster. Hence, useMemo and other hooks have to be used to give the app speed and reduce the DOM complexity while loading components on the screen and putting them together.

REFERENCES

1. Buddhini Gayathri Jayatilleke, Gaya R. Ranawaka and Chamali Wijesekera ,Malinda C.B. Kumarasinha, “Development of mobile application through design-based research”, Asian Association of Open Universities Journal Vol. 13 No. 2, 2018

2. H. K. Flora, X. Wang, and S. V.Chande, “An Investigation into Mobile Application Development Processes: Challenges and Best Practices”, Int. J. Mod. Educ. Comput. Sci., vol. 6, no. 6, pp. 1–9, 2014

3. R. Colomo-Palacios, J. Calvo-Manzano, A. De Amescua, and T. San Feliu, “Agile Estimation Techniques and Innovative Approaches to SW Process Improvement”. 2014.

4. H. K. Flora, X. Wang, and S. V.Chande, “An Investigation into Mobile Application Development Processes: Challenges and Best Practices,” Int. J. Mod. Educ. Comput. Sci., vol. 6, no. 6, pp. 1–9, 2014

5. R. Alkadhi, T. Lata, E. Guzman, and B. Bruegge. Rationale in Development Chat Messages: An Exploratory Study. In Proc. of the 14th Work. Conf. on Min. Softw. Repos., MSR'17, pages 436-446, 2017.