VSampark: A Real-Time Chat Application Using MERN Stack and Socket Programming

Project Report for the course BCSE308L - Computer Networks

by

Sakshi Biyani (22BLC1385)

Vidushi Agnihotri (22BLC1387)

Submitted to

Dr. Hemanth C. SENSE

SCHOOL OF ELECTRONICS ENGINEERING

VELLORE INSTITUTE OF TECHNOLOGY

CHENNAI - 600127

November 2024

Certificate

This is to certify that the Project work titled “VSampark: A real-time chat application using MERN stack and socket programming” is being submitted by Sakshi Biyani (22BLC1385) and Vidushi Agnihotri (22BLC1387) for the course Computer Networks, is a record of bonafide work done under my guidance. The contents of this project work, in full or in parts, have neither been taken from any other source nor have been submitted to any other Institute or University.

ABSTRACT

VSampark is a feature-rich, real-time chat application built using the MERN stack (MongoDB, Express.js, React.js, and Node.js). It is designed to provide users with a seamless communication experience, including functionalities like private messaging, group chats, and secure user authentication. The application utilizes **Socket.IO** for real-time messaging, ensuring that users receive instant updates and message notifications without noticeable delay.

The scalable architecture of VSampark allows it to cater to a wide range of users, from casual individuals to professional teams. It features a responsive user interface built with **Material UI**, which adapts well to various devices, ensuring a smooth experience across desktops and mobiles. On the backend, **Node.js** and **Express.js** handle data processing and communication, while **MongoDB** manages user and message data.

This report provides a comprehensive analysis of the **design**, **implementation**, **testing**, and **future enhancements** of the VSampark application. It covers the project's goals, architectural decisions, and key technologies used, along with a discussion on the **technical feasibility**, **economic viability**, and the overall **impact** of the application. By showcasing the development and real-world applications of VSampark, this project highlights how the MERN stack can be used to build modern, scalable, and responsive web applications that cater to the evolving demands of communication in today's digital world.

Table of Contents

CHAPTER 1
Introduction

In recent years, instant messaging applications have become an integral part of both personal and professional communication. With the increasing reliance on digital communication, real-time messaging platforms are critical for enhancing productivity and connectivity. However, many existing chat applications struggle with issues like performance bottlenecks, latency, and limited functionality for group collaboration. VSampark aims to address these challenges by providing a reliable, scalable, and user-friendly chat application using modern web technologies.

VSampark is built on the **MERN stack** (MongoDB, Express.js, React.js, and Node.js), a powerful set of technologies known for their efficiency and scalability. The application leverages **Socket.IO** for real-time communication, ensuring that messages are delivered almost instantly and without delay. The interface is designed using **Material UI**, providing a modern and responsive layout that works seamlessly across various devices, from desktops to smartphones. The backend, powered by **Node.js** and **Express.js**, ensures smooth data handling and communication, while **MongoDB** serves as the NoSQL database for storing user data, messages, and group chat information.

VSampark aims to be an all-in-one chat solution, integrating essential features like user authentication, message notifications, and private/group messaging into one cohesive platform. Whether it's for casual users, business teams, or educational purposes, VSampark offers a robust communication platform that is both secure and scalable.VSampark integrates core features such as **user authentication**, **message notifications**, and **private/group messaging**, offering a complete communication solution. Whether for casual users, business teams, or educational purposes, VSampark provides a secure, scalable platform designed to meet diverse communication needs. By combining real-time messaging with a responsive interface and secure user management, VSampark aims to provide a reliable and flexible chat experience for users across different environments.

1.1 Purpose:

The following are the objectives of this project:

- **Real-time Communication:**Provide an instant messaging solution where messages are delivered in real-time with no noticeable delay, enabling dynamic interactions.
- **Secure User Authentication:**Ensure the security and privacy of user data by implementing a secure authentication system with token-based session management, preventing unauthorized access to user accounts.
- **User-Friendly Interface:**Build an intuitive and responsive user interface using React and Material UI, ensuring that the application is easy to use and accessible across different devices.
- **Scalability:**Develop an application that can easily scale to accommodate a growing number of users and messages, ensuring long-term reliability and performance.
- **Group Collaboration:**Enable users to collaborate and communicate effectively in group settings, making it suitable for professional teams, educational groups, and communities.
- **Future-Proofing:**Ensure that VSampark is built with flexibility in mind, so that it can be extended with additional features such as video calls, media sharing, and enhanced security features in future versions.

1.2 Scope:

The scope of the VSampark project is to develop a real-time, scalable chat application that provides a range of core functionalities. It begins with **User Authentication and Authorization**, where users can securely register, log in, and authenticate using their credentials. The application uses **JWT (JSON Web Tokens)** to ensure secure authentication and efficient session management, allowing only authorized users to access their accounts. A key feature of VSampark is **Real-time Messaging**, which leverages **Socket.IO** to enable users to send and receive messages instantly, facilitating seamless communication both for individual users and in group chats. For **Private Messaging**, users can engage in one-on-one conversations that are private and secure, ensuring that sensitive information remains protected. Additionally, the platform supports **Group Chat**, allowing users to create or join groups, making it ideal for team collaboration and group discussions. **Profile Management** gives users the flexibility to create and update their profiles, including adding a display name, avatar, and personal status, providing a more personalized experience. The application also includes **Instant Notifications**, alerting users to new messages or updates, keeping them informed in real-time. Designed with **Scalability** in mind, VSampark is built to handle an increasing number of users and messages as the application grows, ensuring it remains efficient and reliable. Lastly, the platform features a **Responsive Design** powered by **Material UI**, ensuring a visually appealing and functional interface that works smoothly across a variety of devices, including desktops and mobiles.

CHAPTER 2

Design

The **Design Approach** of VSampark focuses on creating a robust, scalable, and user-friendly communication platform. By leveraging modern web technologies and a structured architecture, the design ensures that the application is not only efficient in handling real-time messaging but also easy to maintain and scale as user demand increases. The goal of the design approach is to provide an intuitive, secure, and seamless chat experience for users while ensuring optimal performance and flexibility in the system's architecture.

2.1 Architectural Design

The architecture of VSampark follows a **three-tier model** to ensure clear separation of concerns, making the system easier to maintain, scale, and update. Each layer has a distinct responsibility, which contributes to a modular design and optimizes the system's performance.

- **Presentation Layer (Frontend)**: This layer is built using **React.js**, a powerful JavaScript library for building dynamic and interactive user interfaces. React.js allows the application to efficiently update and render components as the state changes, ensuring a responsive and smooth user experience. This layer handles all user interactions, such as sending and receiving messages, user authentication, and displaying data fetched from the backend, such as user profiles and chat histories.
- **Business Logic Layer (Backend)**: The backend is powered by **Node.js** and **Express.js**. Node.js is a server-side JavaScript runtime that allows for high-performance handling of multiple requests simultaneously. Express.js, a web application framework for Node.js, simplifies the handling of HTTP requests, middleware, and routing. This layer is responsible for processing user requests, managing authentication through **JWT tokens**, handling the routing of messages, and managing the overall business logic, including validating data and ensuring that users receive their requested information or perform actions like sending

messages.

- **Data Layer (Database)**: The database used for VSampark is **MongoDB**, a NoSQL database that stores data in a flexible, JSON-like format, making it easier to scale horizontally. MongoDB is well-suited for storing large volumes of dynamic data, such as user information, chat histories, and group data, which may change frequently. This layer is responsible for storing, retrieving, and managing the application's data. It also ensures that the system can easily accommodate growing data needs as the user base expands.

This three-tier architecture is designed to ensure scalability, performance, and ease of maintenance. The separation of concerns means that changes or updates to one layer (e.g., the frontend or the backend) can be made without affecting the other layers, making it easier to manage the system as it grows.

2.2 User Interface Design

The **user interface (UI)** of VSampark is designed with a focus on **usability**, **responsiveness**, and **modern aesthetics**. To achieve this, the interface is built using **Material UI**, a popular React UI framework that implements Google's Material Design principles. Material UI provides a set of pre-built components, such as buttons, text fields, and cards, which help to ensure a consistent and visually appealing user experience across devices.

The UI includes the following key components:

- **Login Page**: The login page allows users to authenticate securely. It includes input fields for the username/email and password, along with error handling to display messages if the login fails.
- **Dashboard**: After logging in, users are presented with a **dashboard** that shows their contacts, recent chat messages, and group chats. The dashboard provides an overview of the user's activity and quick access to ongoing conversations.
- **Chat Window**: The chat window is the main feature of VSampark, where users can engage in real-time conversations. It displays the message history and includes a text input box for typing new messages. The chat window is designed for both individual (private) and group

conversations.

- **Profile Page**: The profile page allows users to view and edit their personal information, such as their display name, avatar, and status. Users can update their details here, offering a personalized experience within the app.

2.3 Data Flow Diagram

A **Data Flow Diagram (DFD)** is used to represent the flow of data within the VSampark application. It illustrates the system's high-level processes and how data moves between components.

- **Level 0 (Context Diagram)**: The highest level of the DFD, which shows the interaction between the user and the system. The user can perform actions such as registering, logging in, sending messages, and managing their profile.
- **Level 1**: In this level, we detail the flow of data when a user sends a message. The user inputs a message, which is sent to the **Socket.IO server** for processing. The message is then stored in **MongoDB** along with the sender's and recipient's details. The recipient's client receives the message in real-time, ensuring immediate communication.

2.4 Use Case Diagram

A **Use Case Diagram** visually represents the interactions between the user and the system, outlining the key functionalities of the VSampark application. The main use cases are:

2 **Register/Login**: Users can create a new account or log in to an existing one, allowing them to authenticate securely using their credentials.

3 **Send Message**: Users can send messages to individuals or groups. This involves inputting the text in the chat window, which is then transmitted to the backend and stored in the database.

4 **Receive Message**: Users are notified in real-time when they receive new messages. Notifications are pushed to the frontend via **Socket.IO**, ensuring immediate delivery.

5 **Update Profile**: Users can edit their profile information, including updating their avatar, name, or status.

CHAPTER 3
Our Approach

3.1 Implementation

3.1.1 Frontend (React.js)

The frontend of VSampark is built using **React.js**, which allows for the creation of dynamic and interactive user interfaces. React uses **React Hooks** such as `useState` and `useEffect` to manage component states and lifecycle events efficiently. **React Router** is used for navigating between different pages (such as the login page, dashboard, and profile page), while **Material UI** ensures a consistent, responsive, and modern design. Additionally, **Socket.IO client** is integrated into the frontend to enable real-time communication, ensuring that users can send and receive messages instantly.

3.1.2 Backend (Node.js and Express.js)

The backend is developed using **Node.js** with the **Express.js** framework. **Express middleware** handles API requests and manages error handling. For secure user authentication, **JWT (JSON Web Tokens)** are used to ensure that only authorized users can access the application. **Socket.IO** is also integrated on the server side to manage real-time communication, enabling message delivery without delays.

3.1.3 Database (MongoDB)

The database for VSampark is built with **MongoDB**, which uses a **NoSQL** format to store data. The database includes collections for:

- **Users**: Stores user information such as their username, email, password hash, and profile data.
- **Messages**: Stores individual chat messages, including the sender, recipient, message content, and timestamp.
- **Groups**: Stores information about group chats, including group names, member details, and group chat history.

3.2 Feasibility Analysis

3.2.1 Economic Feasibility

The VSampark project leverages **open-source technologies** (such as Node.js, React.js, MongoDB, and Socket.IO), which significantly reduces development and operational costs. Hosting the application on cloud platforms like **Heroku** or **Vercel** provides access to free-tier options, making deployment affordable. The scalability of the application ensures that it can handle growing user numbers without substantial additional investment in infrastructure.

3.2.2 Operational Feasibility

VSampark's **intuitive user interface** and **real-time messaging capabilities** ensure that it is easy to use and meets the needs of fast-paced communication environments. The responsive design, built with **Material UI**, ensures that the platform provides a smooth experience on desktops, tablets, and smartphones. The system is designed to be highly functional for casual and professional users alike.

3.2.3 Technical Feasibility

The **MERN stack** offers excellent compatibility for the development of scalable, high-performance applications. Node.js's event-driven architecture and **Socket.IO**'s real-time messaging capabilities allow VSampark to handle large volumes of messages with low latency. The system can efficiently handle multiple users and concurrent requests, ensuring reliable performance under heavy traffic.

3.3 Software Requirements for VSampark

The **software requirements** for VSampark outline the necessary technologies, tools, and configurations needed to develop, deploy, and run the application. These requirements ensure that the system operates smoothly, efficiently, and securely. Below are the primary software requirements for both the development and deployment of VSampark:

3.3.1. Operating System

VSampark is developed and tested on **Windows** operating systems. We used **Windows 11** to run and develop the application.These operating systems are fully compatible with the technologies used in VSampark, such as **Node.js**, **React.js**, **MongoDB**, and **Socket.IO**.

3.3.2. Development Tools

For development, the following tools are essential:

- **Code Editor/IDE**: A code editor is required to write and edit the source code. Popular choices include:
  - **Visual Studio Code**: A lightweight, extensible code editor with support for JavaScript and React.js.
  - **WebStorm**: A full-featured IDE designed for JavaScript and web development.
- **Version Control**: Git is used for source code versioning, and platforms like **GitHub** or **GitLab** are used to host repositories, enabling collaboration and tracking changes.
  - **Git**: A distributed version control system used to manage source code history and enable collaboration.
  - **GitHub/GitLab**: Online platforms for hosting Git repositories.

3.3.3. Frontend Requirements

The frontend is built using **React.js**, and the following tools are required for development:

- **React.js**: A JavaScript library for building dynamic user interfaces.
  - **React Router**: A library for handling routing and navigation between pages within the React

app.

  - **Material UI**: A component library that provides pre-built, customizable UI components for React, ensuring a consistent and responsive design.
  - **Axios or Fetch API**: Libraries for making HTTP requests to the backend API.

Additionally, developers must install and configure **Node.js** for running JavaScript in the development environment.

3.3.4. Backend Requirements

The backend is built using **Node.js** and **Express.js**, and the following software requirements are necessary:

- **Node.js**: A JavaScript runtime built on Chrome's V8 engine. Node.js allows for the development of scalable backend services using JavaScript.
  - **Express.js**: A lightweight framework for building RESTful APIs with Node.js. It simplifies the creation of routes, handling HTTP requests, and managing middleware.
- **Socket.IO**: A library that facilitates real-time, bi-directional communication between the frontend and backend. This is essential for enabling real-time messaging features in VSampark.
- **JWT (JSON Web Tokens)**: A compact, URL-safe means of representing claims to be transferred between two parties, ensuring secure user authentication and session management.

3.3.5. Database Requirements

VSampark uses **MongoDB** as its database to store user data, chat history, and group information:

- **MongoDB**: A NoSQL database that stores data in flexible, JSON-like format, making it highly suitable for the dynamic nature of chat applications.
  - **MongoDB Atlas** (optional): A cloud database service for MongoDB that provides managed instances with automated scaling and backups.
- **Mongoose**: An ODM (Object Data Modeling) library for MongoDB, used to define models for the database and interact with it in a more structured way.

3.3.6. Real-Time Communication

To ensure real-time communication, VSampark uses **Socket.IO** for establishing WebSocket connections between the client and server. The necessary libraries include:

- **Socket.IO server**: A library for enabling real-time, event-driven communication between the server and client.
- **Socket.IO client**: The frontend counterpart to the server library that listens for and emits events to enable real-time interactions like instant messaging.

3.3.7. Authentication and Security

For secure user authentication, VSampark integrates the following technologies:

- **JWT (JSON Web Tokens)**: Used for secure, token-based authentication, allowing users to log in and maintain their session without the need for repeated logins.
- **bcrypt.js**: A library for hashing and salting passwords before storing them in the database to ensure security.

3.3.8. Deployment and Hosting Requirements

For deployment, VSampark uses **Render**, a cloud platform for deploying web applications. The following software requirements are necessary for deployment:

- **Render**: A cloud platform that provides simple deployment options for web applications. Render supports both frontend and backend services, allowing VSampark to be deployed with ease. Render also provides auto-scaling and automated deployments from GitHub or GitLab repositories.

Additionally, **MongoDB Atlas** (optional) can be used to host the database in the cloud, ensuring that the application is fully managed and scalable.

3.3.9. Testing and Debugging Tools

Testing tools are important to ensure the application is bug-free and functions as expected:

- **Postman**: A popular tool for testing and debugging API endpoints. Developers can use Postman to send requests to the backend, check the server's responses, and ensure that API routes are functioning as expected.

3.3.10. Additional Libraries and Tools

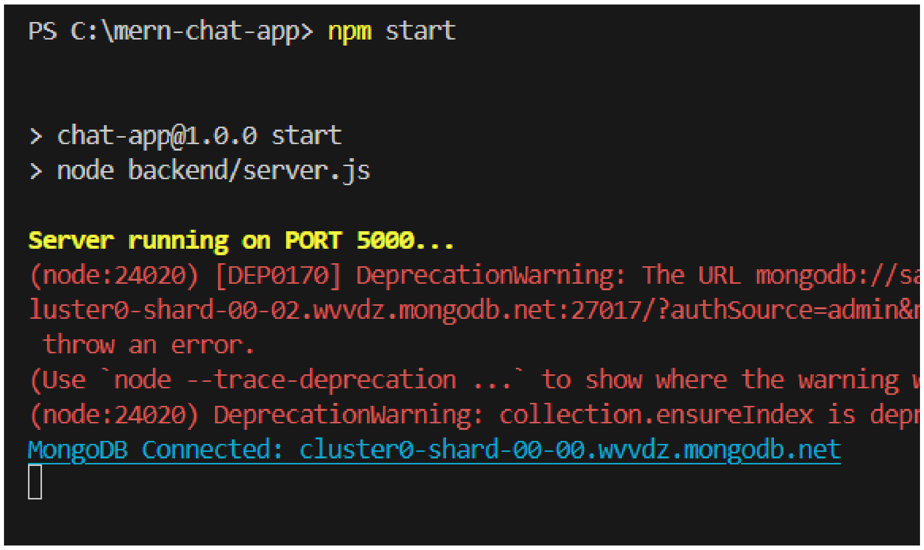
- **ESLint/Prettier**: For maintaining consistent code quality and formatting across the codebase.

CHAPTER 4
Result Analysis / Testing / Summary

4.1 Execution

4.1.1 Backend (Node.js + Express):

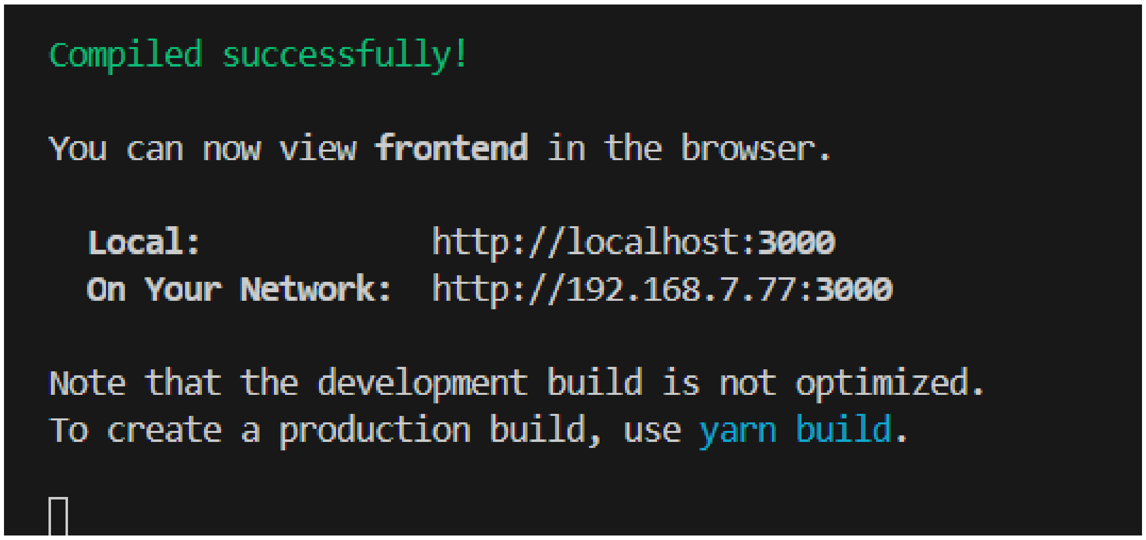
- The backend server is running on **port 5000** using **Node.js** and **Express**.
- MongoDB (Database) is connected successfully, enabling data storage for users, chats, and messages.
- API routes are set up for user authentication, chat messages, and real-time interactions using **REST API** and **Socket.IO**.
- Backend logs show successful server startup and MongoDB connection status.



```
PS C:\mern-chat-app> npm start


> chat-app@1.0.0 start
> node backend/server.js

Server running on PORT 5000...
(node:24020) [DEP0170] DeprecationWarning: The URL mongodb://sa
luster0-shard-00-02.wvvdz.mongodb.net:27017/?authSource=admin&r
 throw an error.
(Use `node --trace-deprecation ...` to show where the warning w
(node:24020) DeprecationWarning: collection.ensureIndex is depr
MongoDB Connected: cluster0-shard-00-00.wvvdz.mongodb.net
```

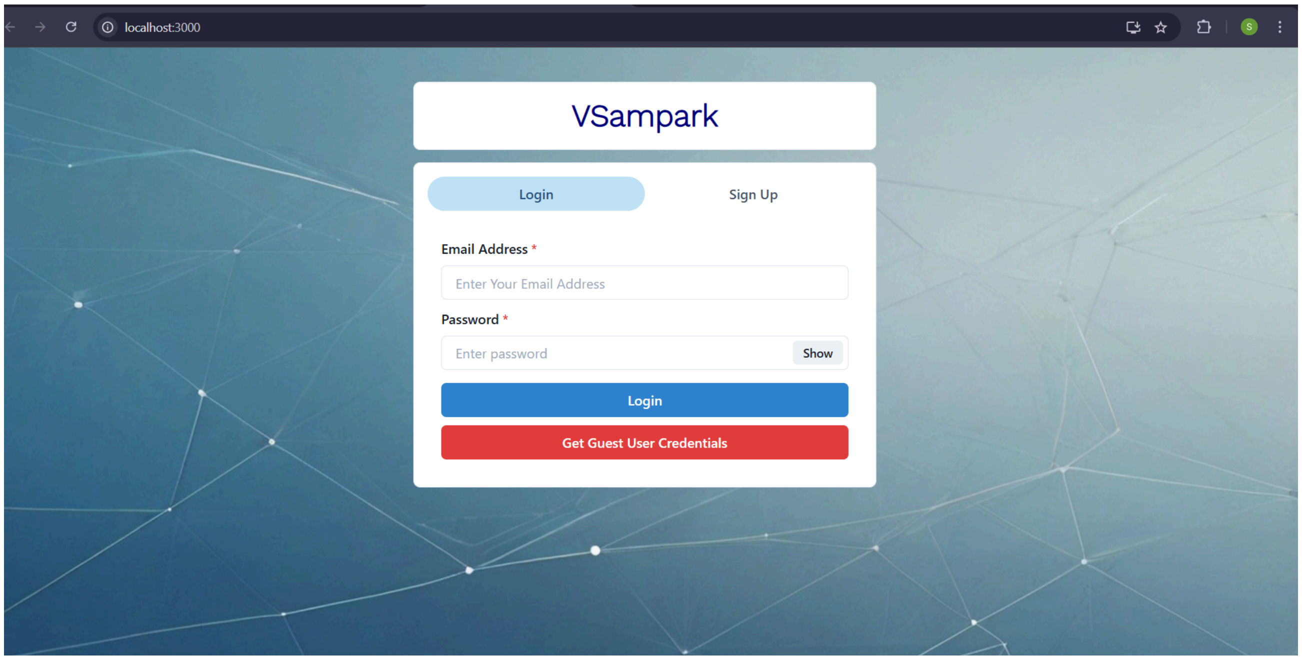
4.1.2 Frontend (React):

- The frontend application is developed using **React** and runs on **port 3000**.
- The React app is compiled successfully with no errors, ready for client-side interaction.
- It connects seamlessly with the backend via **Axios** or **Fetch API** for data fetching.
- The chat interface is live and can be accessed at **http://localhost:3000**.


```
Compiled successfully!

You can now view frontend in the browser.

  Local:            http://localhost:3000
  On Your Network:  http://192.168.7.77:3000

Note that the development build is not optimized.
To create a production build, use yarn build.
```

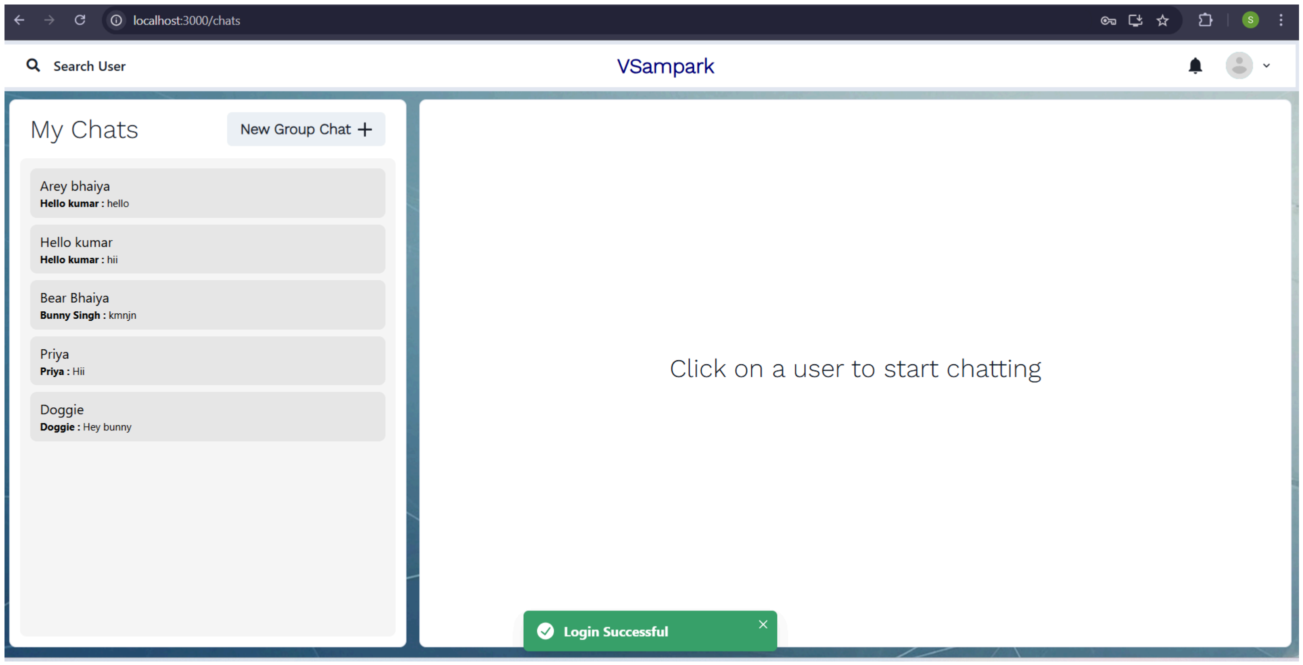
4.1.3 Home Page (Login & Sign-Up):

- The home page serves as the **login and registration interface** of the chat application.
- It features a user-friendly design with a clean background and responsive layout.
- Users can log in using their email and password or opt for guest user credentials to quickly access the chat.
- The page includes:
  - **Login Form:** Input fields for email and password with "Show" toggle for password visibility.
  - **Sign-Up Option:** Allows new users to create an account if they are not registered.
  - **Guest Login Button:** Provides a quick access option for users without an account.



4.1.4 Chat Page:

- The chat page is the main interface after successful login, where users can view their chats and interact in real-time.
- Key sections include:
  - **Chat List:** Displays all active chats on the left, with user names and recent messages.
  - **Search Bar:** Users can search for contacts or existing chats.
  - **Chat Window:** The main area displays chat conversations when a user is selected.
  - **New Group Chat Button:** Allows users to create group chats and add multiple participants.
  - **Notifications and User Profile:** Icons for notifications and user settings are accessible in the top-right corner.



Arey bhaiya
Hello kumar : hello

Hello kumar
Hello kumar : hii

Bear Bhaiya
Bunny Singh : kmnjn

Priya
Priya : Hii

Doggie
Doggie : Hey bunny

Click on a user to start chatting

Login Successful

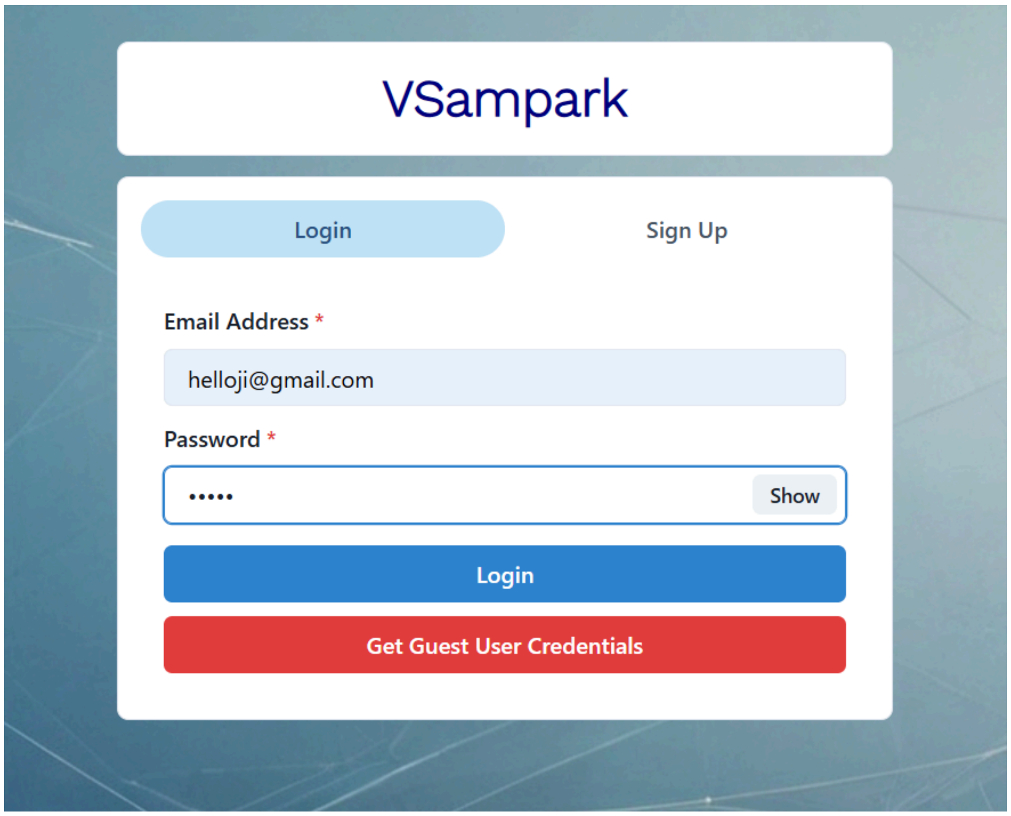
This setup provides an interactive and seamless experience, leveraging real-time updates through **Socket.IO** for message exchanges and MongoDB for persistent chat storage.

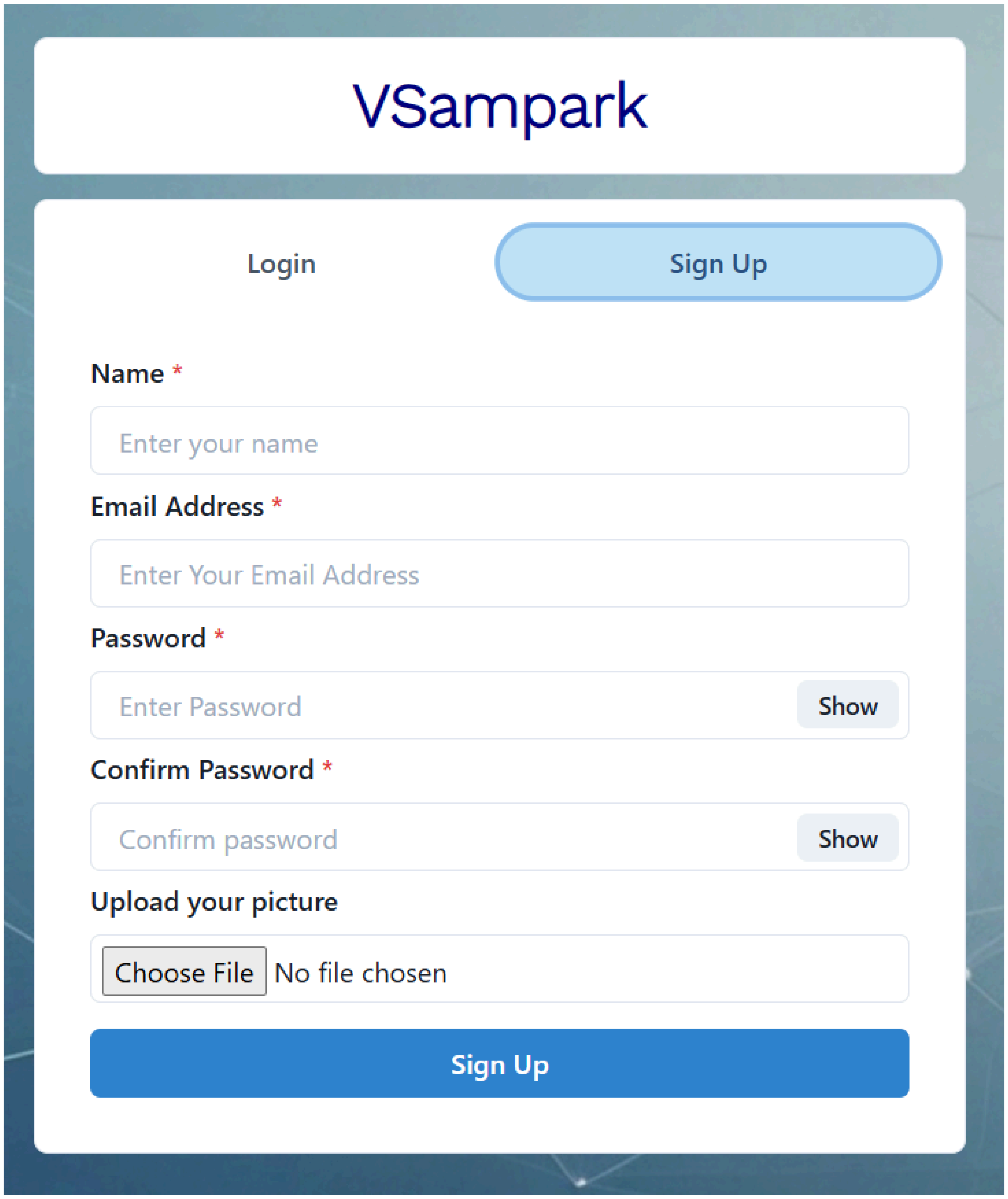
4.2 Features

The chat application is designed to offer a comprehensive, real-time communication experience. It includes robust user authentication, seamless one-on-one and group chats, and convenient features like user search and profile viewing. Additionally, real-time updates, notifications and typing indicators ensure an interactive and engaging experience for all users. Below is a brief description of each feature:

4.2.1 Authentication

- **Secure Login & Sign-Up:** Users can register and log in using email and password.
- **JWT (JSON Web Token) Integration:** Ensures secure access with token-based authentication.
- **Guest Login Option:** Allows users to quickly access the app without creating an account.





VSampark

Name *

Enter your name

Email Address *

Enter Your Email Address

Password *

Enter Password

Confirm Password *

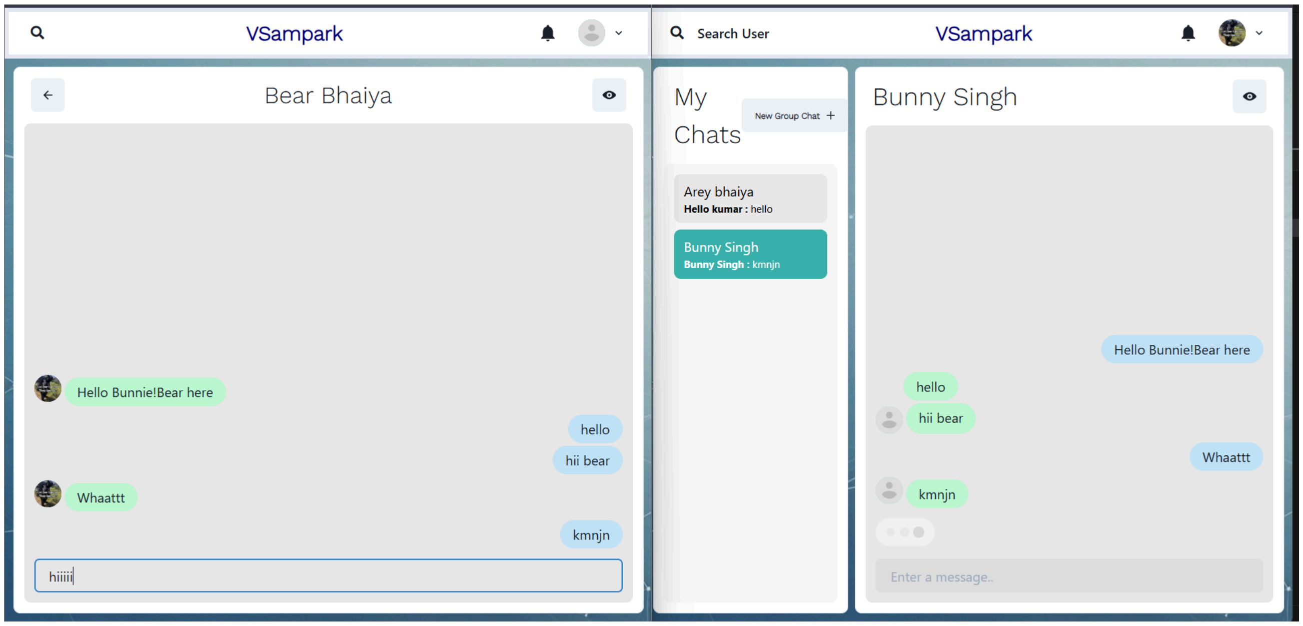
Confirm password

Upload your picture

No file chosen

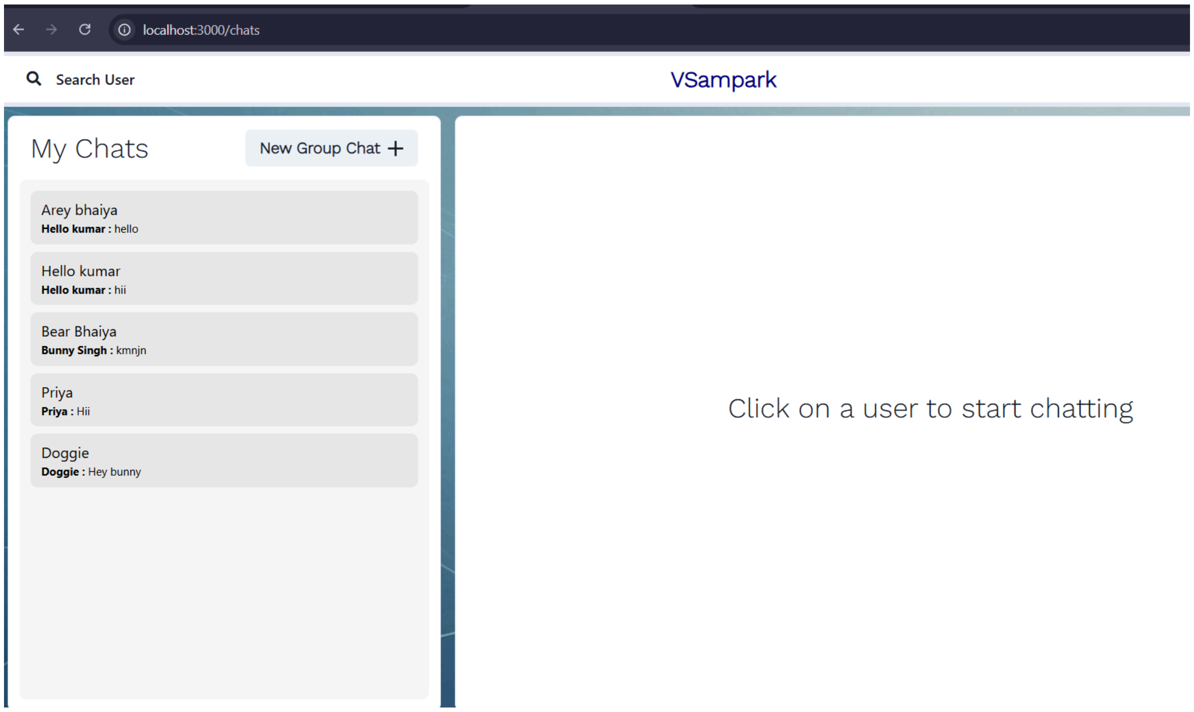
4.2.2 Real Time Chatting with Typing indicators

- **Instant Messaging:** Messages are sent and received in real-time using **Socket.IO**.
- **Typing Indicator:** Displays when the other user is typing, enhancing the chat experience.



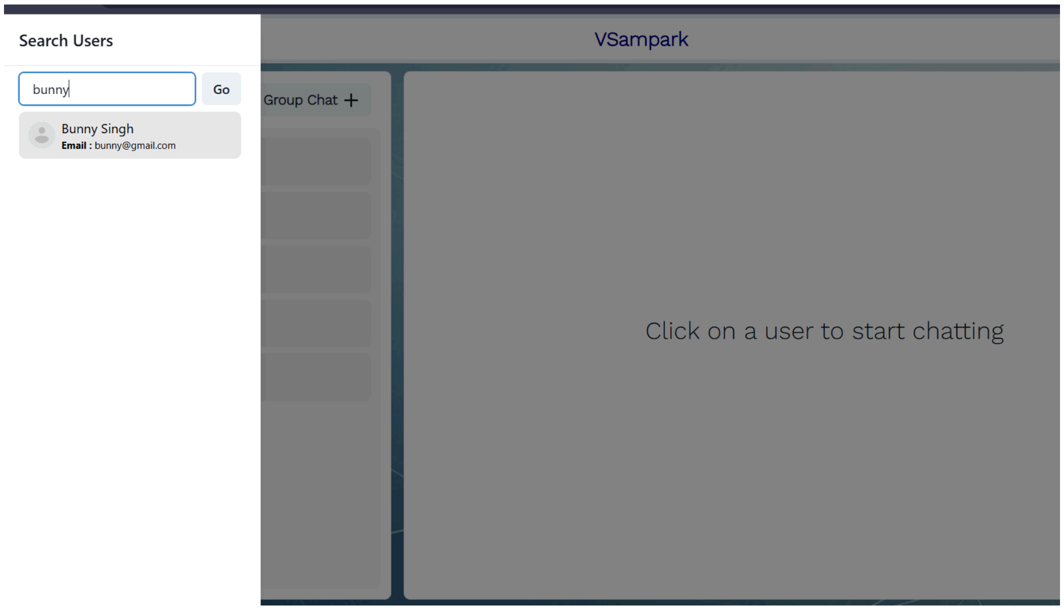
4.2.3 One to One chat

- **Direct Messaging:** Users can chat privately with individual contacts.
- **Message History:** Conversations are saved in MongoDB, allowing users to view previous messages.



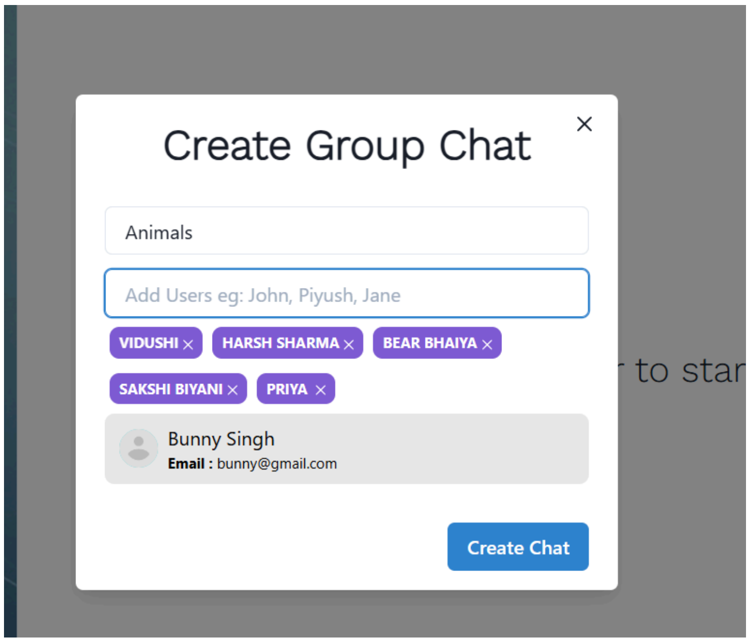
4.2.4 Search Users

- **User Search Functionality:** Users can search for other registered users by name or email.
- **Autocomplete Suggestions:** Provides dynamic search suggestions as users type



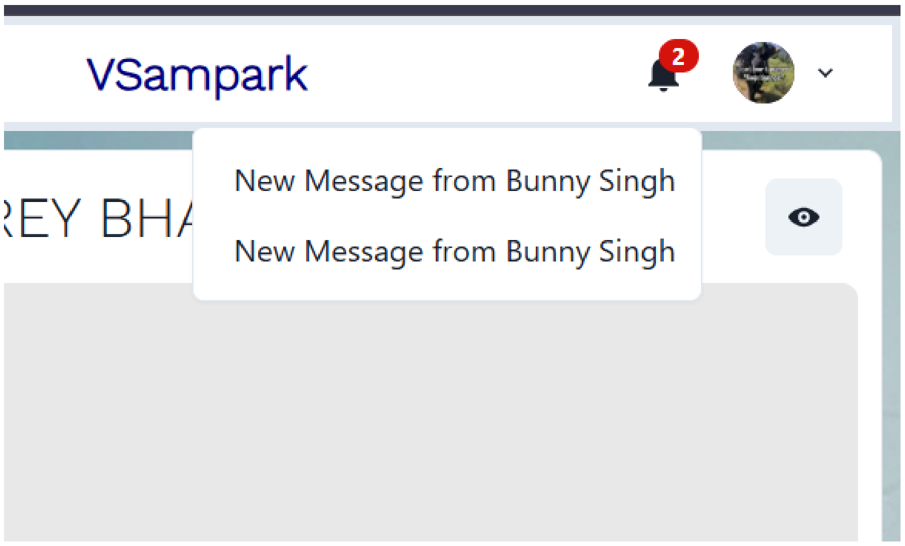
4.2.5 Create Group Chats

- **Group Chat Creation:** Users can create new group chats and invite multiple participants.
- **Group Management:** Customizable group names and icons for easy identification.



4.2.6 Notifications

- **Message Notifications:** Users are alerted with real-time notifications for new messages.
- **Visual Indicators:** Unread messages and new chat alerts are highlighted for user attention

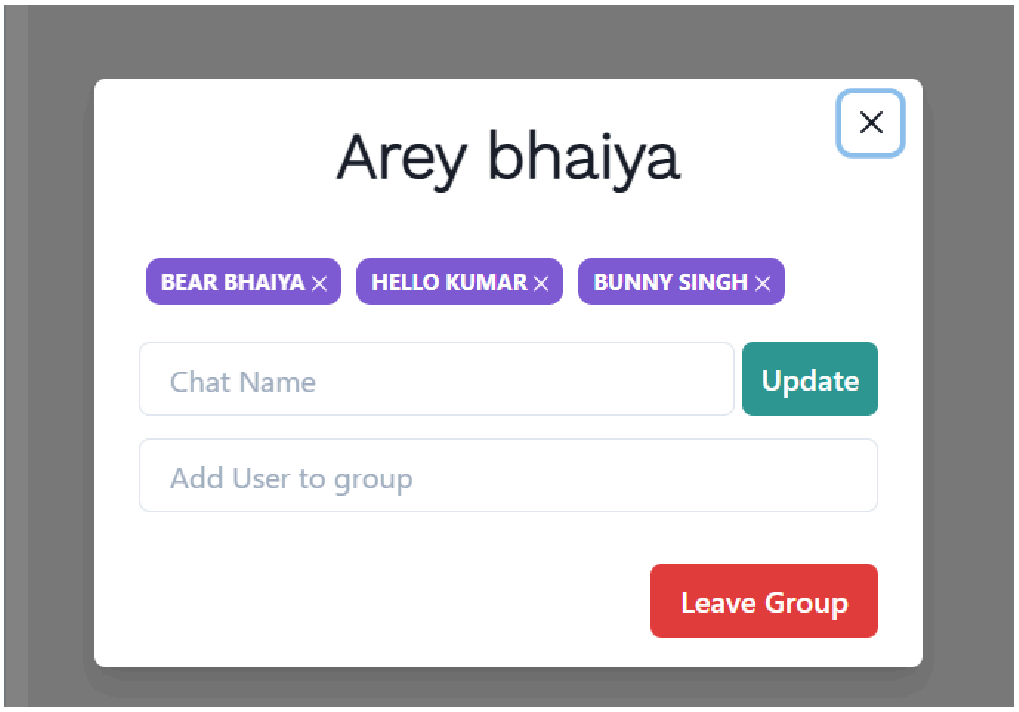


New Message from Bunny Singh

New Message from Bunny Singh

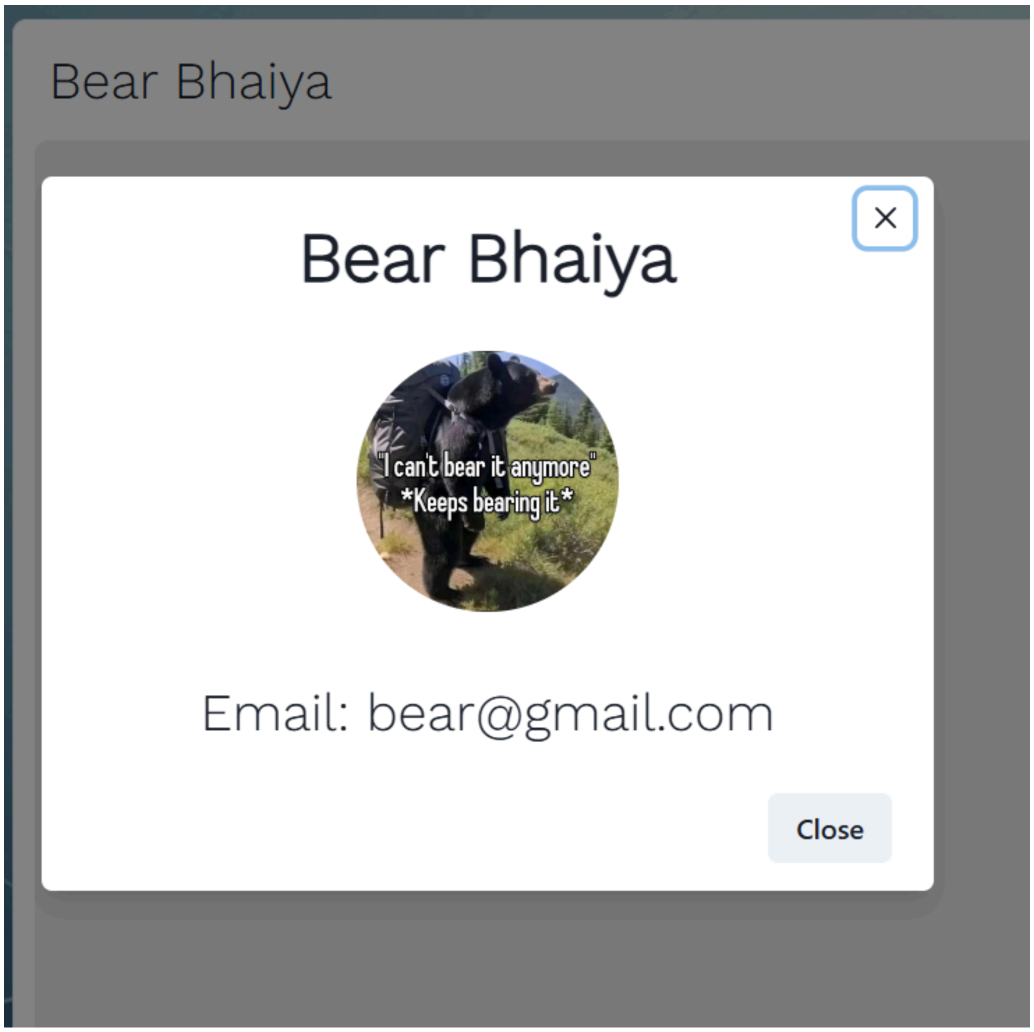
4.2.7 Add or Remove users from group

- **Manage Group Participants:** Admins can add new users to the group or remove existing members.
- **Permission Control:** Only group admins have the authority to modify the participant list.



4.2.8 View Other user Profile

- **Profile Viewing:** Users can click on a contact to view their profile information.
- **User Details:** Displays user name, email, and status for better context during chats.


Bear Bhaiya

Bear Bhaiya

Email: bear@gmail.com

4.3 Testing

Testing and validation are essential steps in ensuring that VSampark functions as expected, providing a reliable and bug-free user experience. The following types of testing are employed:

4.3.1. Unit Testing

Unit testing involves testing individual components or functions of the application to ensure they work as expected. This includes:

- **Frontend Components**: React components are tested to ensure that UI elements such as buttons, input fields, and notifications work correctly.
- **Backend Functions**: Functions like message sending, user authentication, and database interactions are tested for correct functionality.

Testing frameworks like **Jest** are used to run automated tests for individual components.

4.3.2. Integration Testing

Integration testing ensures that different parts of the system work together as expected. This includes:

- **Frontend-Backend Interaction**: Verifying that the frontend can successfully send and receive data from the backend, especially in real-time messaging scenarios.
- **Database Interaction**: Ensuring that the backend interacts with MongoDB correctly, storing and retrieving data like messages, user profiles, and group information.

4.3.3. User Acceptance Testing (UAT)

User Acceptance Testing involves gathering feedback from real users to evaluate the application’s usability and functionality. During this phase:

- **User Feedback**: Users test the application to identify any issues in the interface, performance, or functionality. Feedback is gathered on features such as messaging, notifications, and user authentication.

- **Usability**: Users provide insights on the ease of use, layout, and accessibility of the application. Based on this feedback, adjustments are made to improve the user experience.

4.4 Summary

VSampark follows a **three-tier architecture** consisting of the **presentation layer (frontend)** built with **React.js**, the **business logic layer (backend)** powered by **Node.js** and **Express.js**, and the **data layer** managed by **MongoDB**. This architecture ensures a clear separation of concerns, improving maintainability and scalability. The user interface is designed using **Material UI** to ensure responsiveness across various devices. Key features like real-time messaging, user authentication, and profile management are seamlessly integrated into the system. Additionally, data flows between components are efficiently managed through technologies like **Socket.IO** for real-time communication and **JWT** for secure authentication. This design approach ensures that VSampark is a reliable and scalable chat application, ready to meet the demands of both individual and team communication

CHAPTER 5
FUTURE ENHANCEMENT AND CONCLUSION

5.1 Future Enhancement

As VSampark continues to grow and adapt to the changing landscape of digital communication, there are numerous opportunities for expanding its features and improving the overall user experience. These future enhancements aim to address emerging user needs, take advantage of new technologies, and ensure that VSampark remains at the forefront of chat application development. Some of the most promising improvements include:

5.1.1. Voice and Video Calls

Voice and video calls are rapidly becoming essential features in modern communication platforms. By adding **voice and video calling functionality**, VSampark can transform from a text-based chat application to a more comprehensive communication tool. This enhancement would provide users with the ability to:

- **Engage in real-time voice communication**: Users could place direct voice calls to one another, creating a richer, more personal way of interacting than text messaging alone.
- **Enable video conferencing**: Groups of users could participate in live video calls, which would be especially useful for remote teams, educational purposes, or virtual social interactions.
- **Cross-platform support**: The ability to conduct calls between users on various platforms (i.e., desktop, mobile) would ensure accessibility across different devices.

This feature would be implemented using **WebRTC (Web Real-Time Communication)** or similar technologies. WebRTC is an open-source project that allows audio, video, and data sharing in real-time without needing to install plugins. It can handle peer-to-peer communication, making it an ideal solution for building high-quality voice and video calling features in a web application.

5.1.2. End-to-End Encryption

In today’s digital landscape, **data privacy and security** are of paramount importance. To enhance user trust and provide the highest level of security, VSampark could implement **end-to-end encryption (E2EE)**. This encryption ensures that:

- **Only the sender and receiver can read the messages**: With E2EE, messages are encrypted before they leave the sender's device and only decrypted once they reach the recipient's device. Even if the messages are intercepted during transmission, they will remain unreadable without the appropriate decryption keys.
- **Higher privacy and protection**: End-to-end encryption would protect sensitive communications, making it impossible for unauthorized parties, including service providers, to access or eavesdrop on private conversations.
- **Building user trust**: By implementing E2EE, VSampark can assure its users that their private data and communications are fully protected from external threats or government surveillance.

This feature would require changes to the backend infrastructure to ensure that messages are encrypted at the point of sending and only decrypted at the point of delivery. It would likely involve adopting strong encryption standards, such as **AES (Advanced Encryption Standard)**, to secure the data and keys.

5.1.3. Mobile Application Support

While VSampark is currently a **web-based application**, expanding its capabilities to support **native mobile applications** for **iOS** and **Android** would significantly enhance the accessibility and versatility of the platform. A mobile application would provide several advantages, including:

- **On-the-go access**: Users would be able to access VSampark anytime and anywhere, even when they are away from their computers. This flexibility would make the platform more convenient for users who need to stay connected during commutes or while traveling.
- **Push notifications**: Mobile applications allow for instant push notifications, which can notify users of new messages, calls, or other important events even when the app is not open.
- **Optimized for mobile devices**: A native mobile app can be designed to take full advantage of

the mobile device's capabilities, such as camera, microphone, and GPS features, offering a smoother, more tailored experience than the web version.

Developing mobile applications would involve using **React Native**, which allows for building cross-platform apps using the same core codebase, or alternatively, creating separate native applications using **Swift** for iOS and **Kotlin** for Android. This would involve additional considerations for app store deployment and maintaining versions across platforms.

5.1.4. File and Media Sharing

In many modern communication platforms, the ability to share **files, images, videos**, and other types of media is an essential feature. VSampark could greatly benefit from the addition of **file and media sharing capabilities**, including:

- **Support for various file types**: Users could upload and share files like PDFs, Word documents, images, audio recordings, and even video files within their chats. This would enable richer communication and collaboration between users.
- **Group collaboration**: In group chats, users would be able to share important resources such as meeting notes, presentations, or design files, enhancing the collaborative experience.
- **Cloud storage integration**: To support large file uploads, VSampark could integrate with cloud storage services such as **AWS S3**, **Google Drive**, or **Dropbox** to store files, ensuring that users can share and access media without worrying about file size limits.

5.1.5 Additional Improvements

In addition to the core improvements mentioned above, VSampark has the potential to implement a range of other exciting features that could further enhance its utility and user appeal:

- **Multilingual Support**: Offering support for multiple languages could attract a wider audience and improve accessibility for users from different linguistic backgrounds.
- **Chatbot Integration**: Implementing intelligent **chatbots** could help users with common queries, automate certain tasks, or provide personalized recommendations within the chat interface.
- **Advanced Search Functionality**: Enabling advanced search capabilities within

conversations could allow users to quickly find important messages, files, or group chats based on keywords, timestamps, or other criteria.

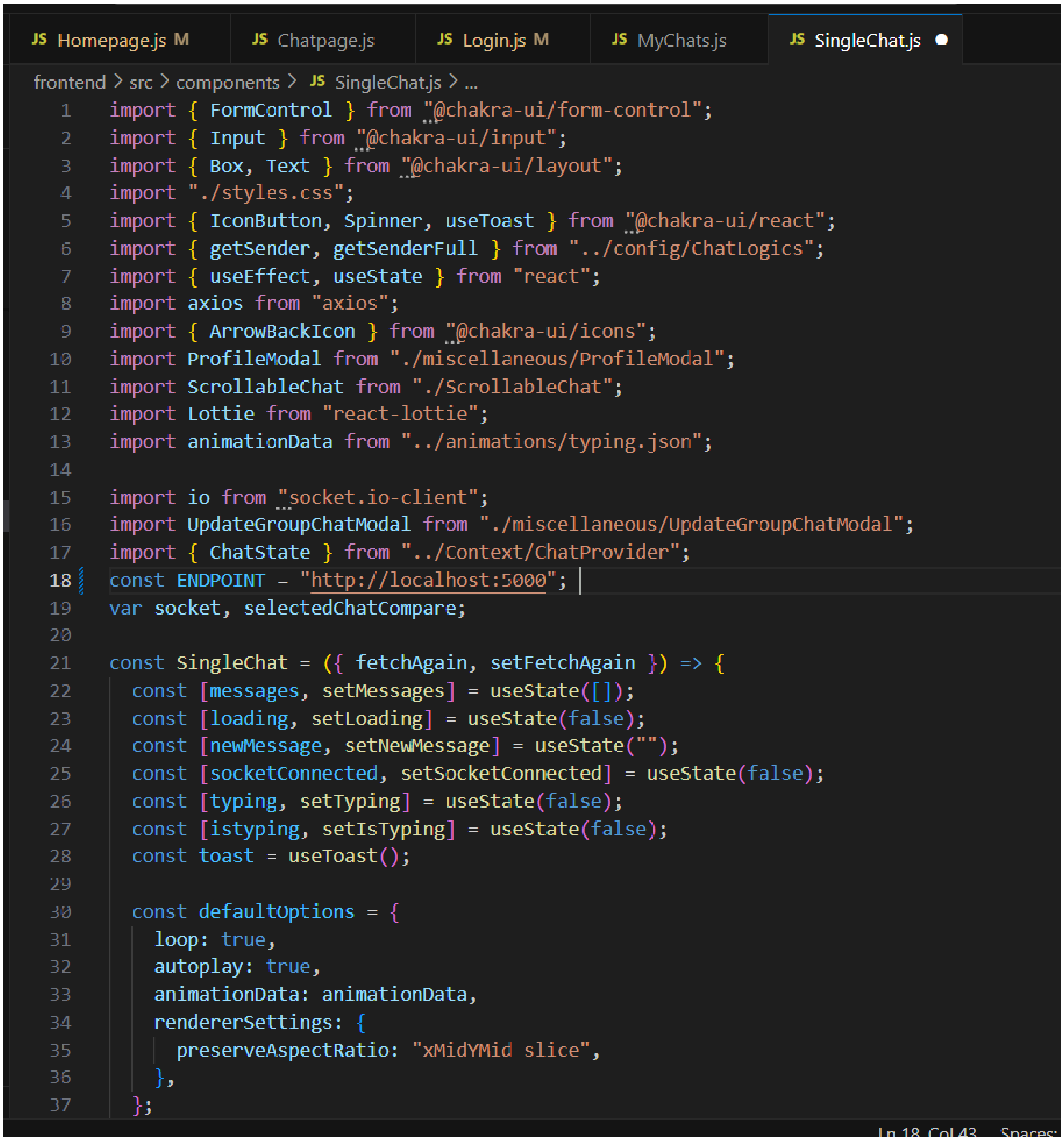
- **Themes and Customization**: Giving users the ability to customize their app experience with different themes, color schemes, or custom avatars would improve user satisfaction and engagement.

By integrating these features, VSampark could expand its reach and functionality, making it a truly comprehensive platform that meets the evolving needs of its users.

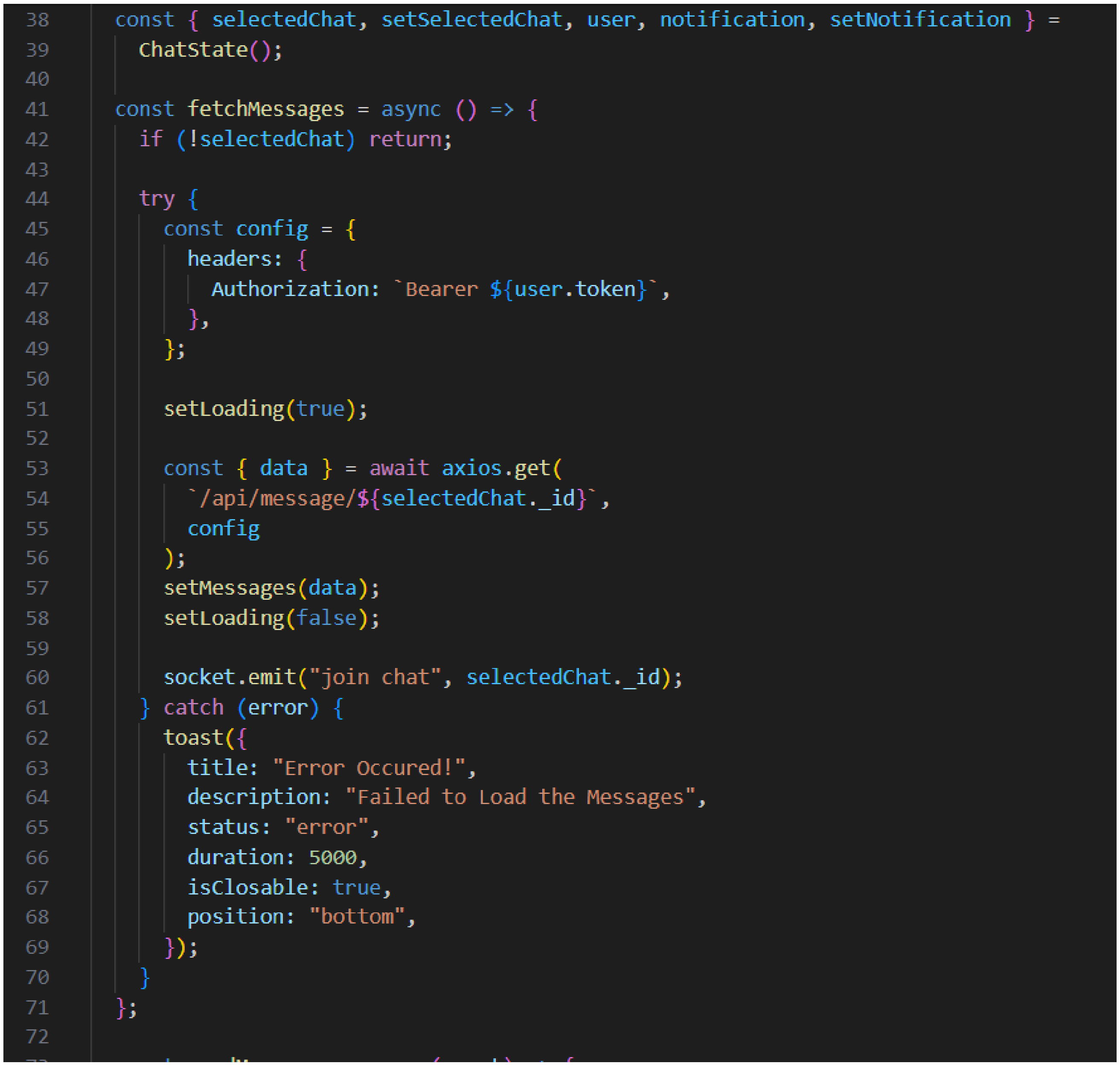
5.2 Conclusion

The future of VSampark looks promising with numerous opportunities for growth and enhancement. With the addition of **voice and video calls**, **end-to-end encryption**, **mobile application support**, and **file/media sharing**, VSampark can offer a more holistic communication experience that meets the needs of both individual users and professional teams. Furthermore, the potential for additional features, such as multilingual support and chatbots, demonstrates the platform’s flexibility and capacity to adapt to emerging user demands. As VSampark continues to evolve, it will not only enhance its functionality but also ensure a secure, seamless, and scalable environment for users across the globe.

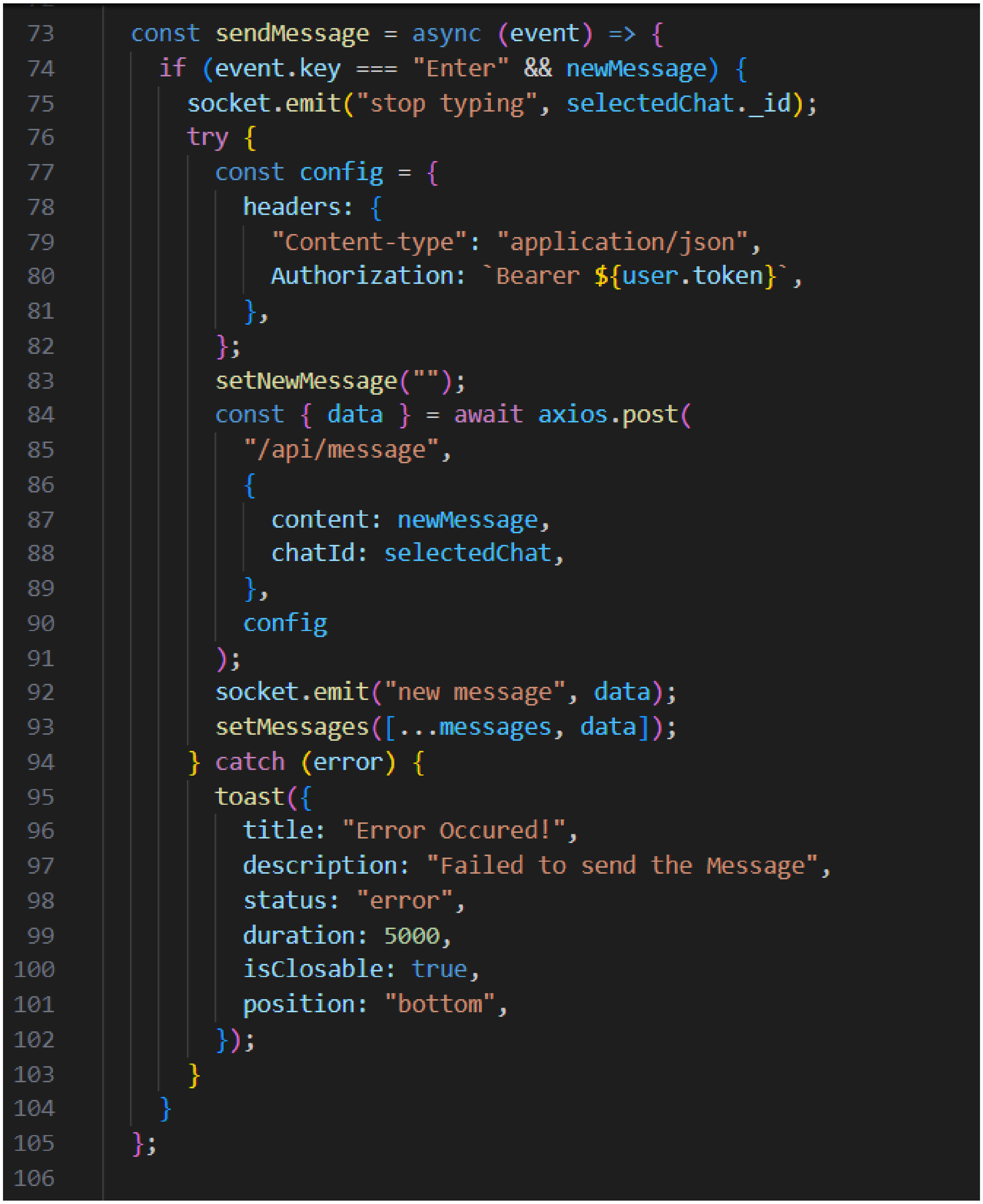
CHAPTER 6
APPENDIX



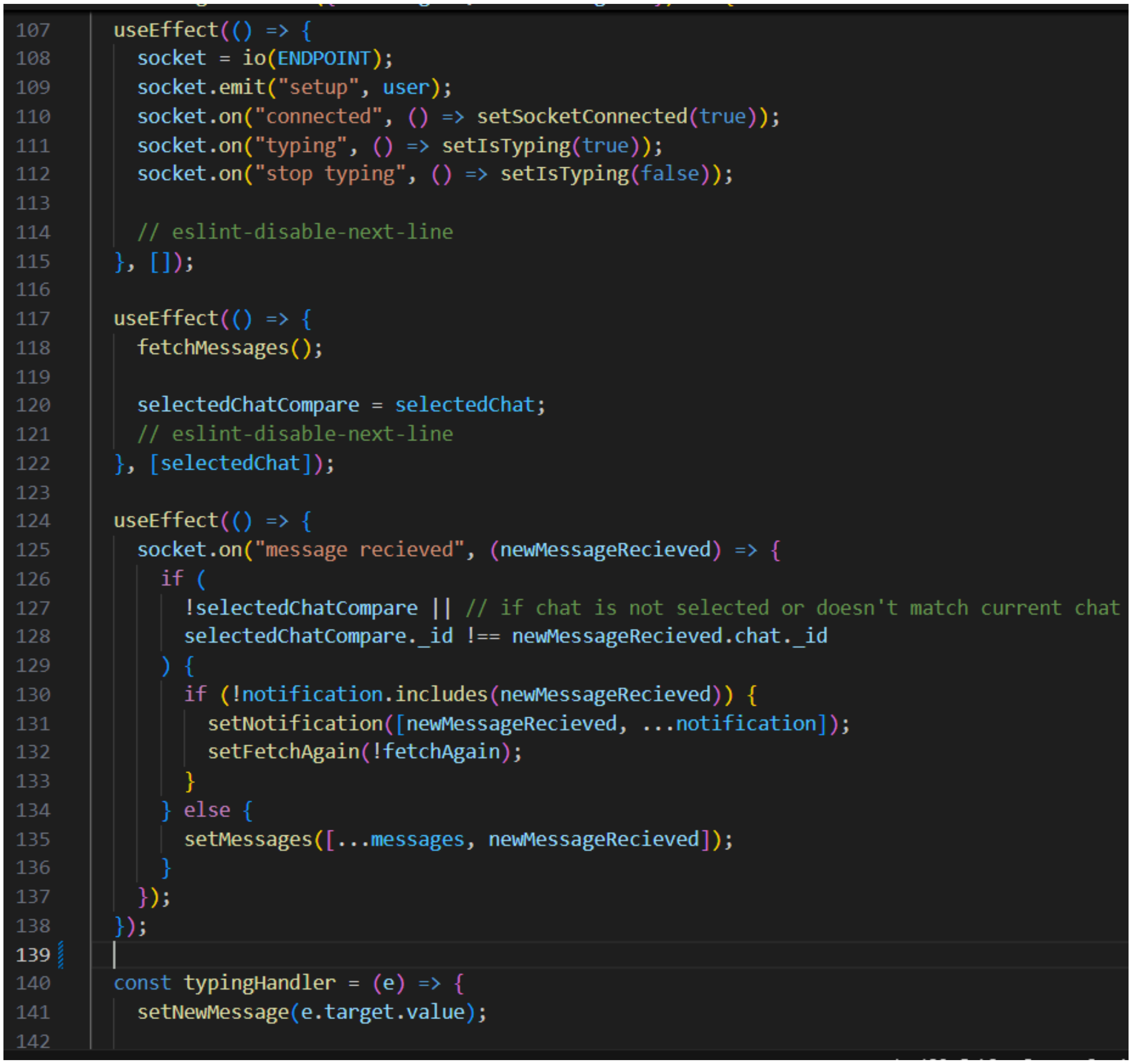
```
import { FormControl } from "@chakra-ui/form-control";
import { Input } from "@chakra-ui/input";
import { Box, Text } from "@chakra-ui/layout";
import "./styles.css";
import { IconButton, Spinner, useToast } from "@chakra-ui/react";
import { getSender, getSenderFull } from "../config/ChatLogics";
import { useEffect, useState } from "react";
import axios from "axios";
import { ArrowBackIcon } from "@chakra-ui/icons";
import ProfileModal from "./miscellaneous/ProfileModal";
import ScrollableChat from "./ScrollableChat";
import Lottie from "react-lottie";
import animationData from "../animations/typing.json";

import io from "socket.io-client";
import UpdateGroupChatModal from "./miscellaneous/UpdateGroupChatModal";
import { ChatState } from "../Context/ChatProvider";
const ENDPOINT = "http://localhost:5000";
var socket, selectedChatCompare;

const SingleChat = ({ fetchAgain, setFetchAgain }) => {
  const [messages, setMessages] = useState([]);
  const [loading, setLoading] = useState(false);
  const [newMessage, setNewMessage] = useState("");
  const [socketConnected, setSocketConnected] = useState(false);
  const [typing, setTyping] = useState(false);
  const [istyping, setIsTyping] = useState(false);
  const toast = useToast();

  const defaultOptions = {
    loop: true,
    autoplay: true,
    animationData: animationData,
    rendererSettings: {
      preserveAspectRatio: "xMidYMid slice",
    },
  };
```



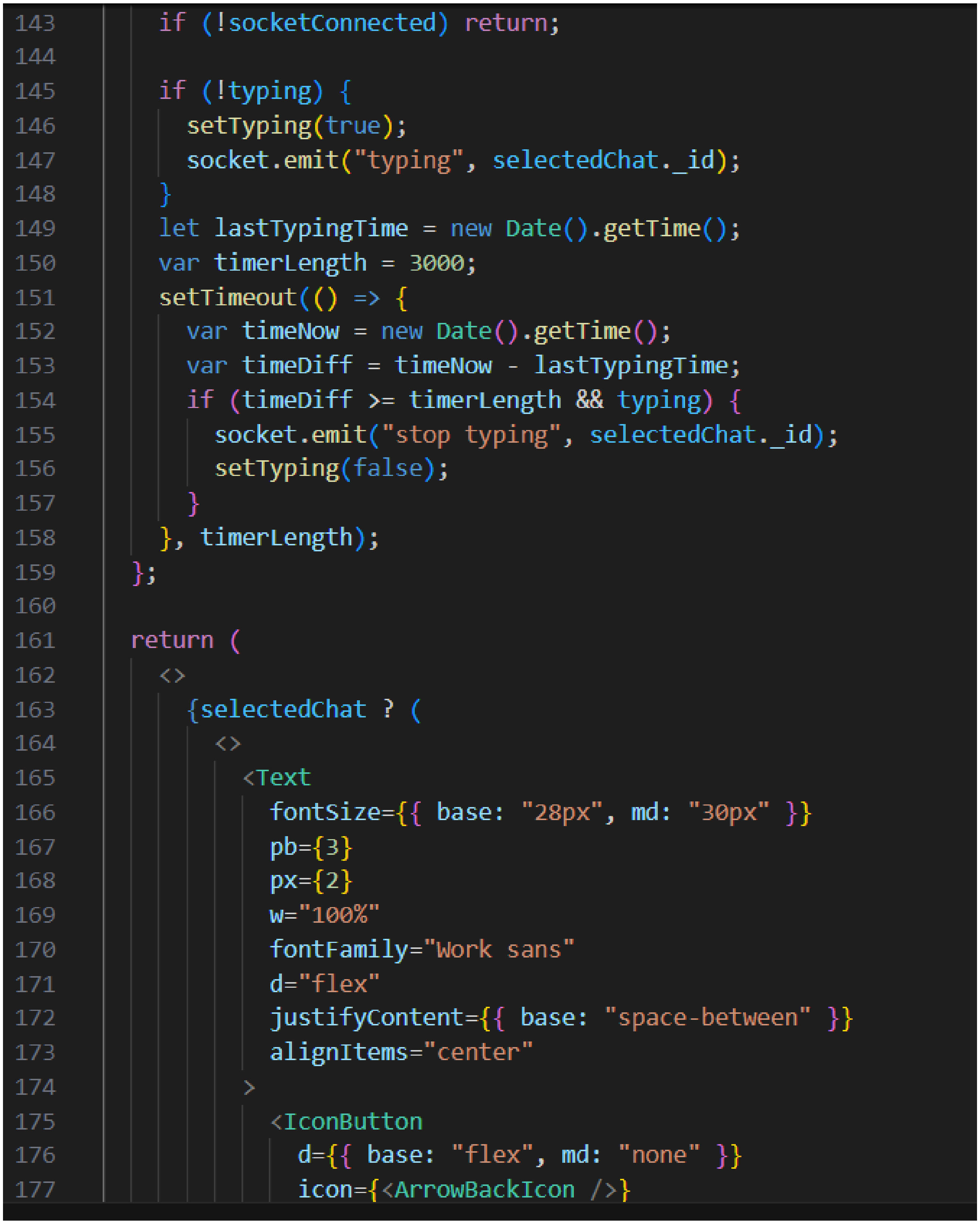
```
const { selectedChat, setSelectedChat, user, notification, setNotification } =
  ChatState();

const fetchMessages = async () => {
  if (!selectedChat) return;

  try {
    const config = {
      headers: {
        Authorization: `Bearer ${user.token}`,
      },
    };

    setLoading(true);

    const { data } = await axios.get(
      `/api/message/${selectedChat._id}`,
      config
    );
    setMessages(data);
    setLoading(false);

    socket.emit("join chat", selectedChat._id);
  } catch (error) {
    toast({
      title: "Error Occured!",
      description: "Failed to Load the Messages",
      status: "error",
      duration: 5000,
      isClosable: true,
      position: "bottom",
    });
  }
};
```



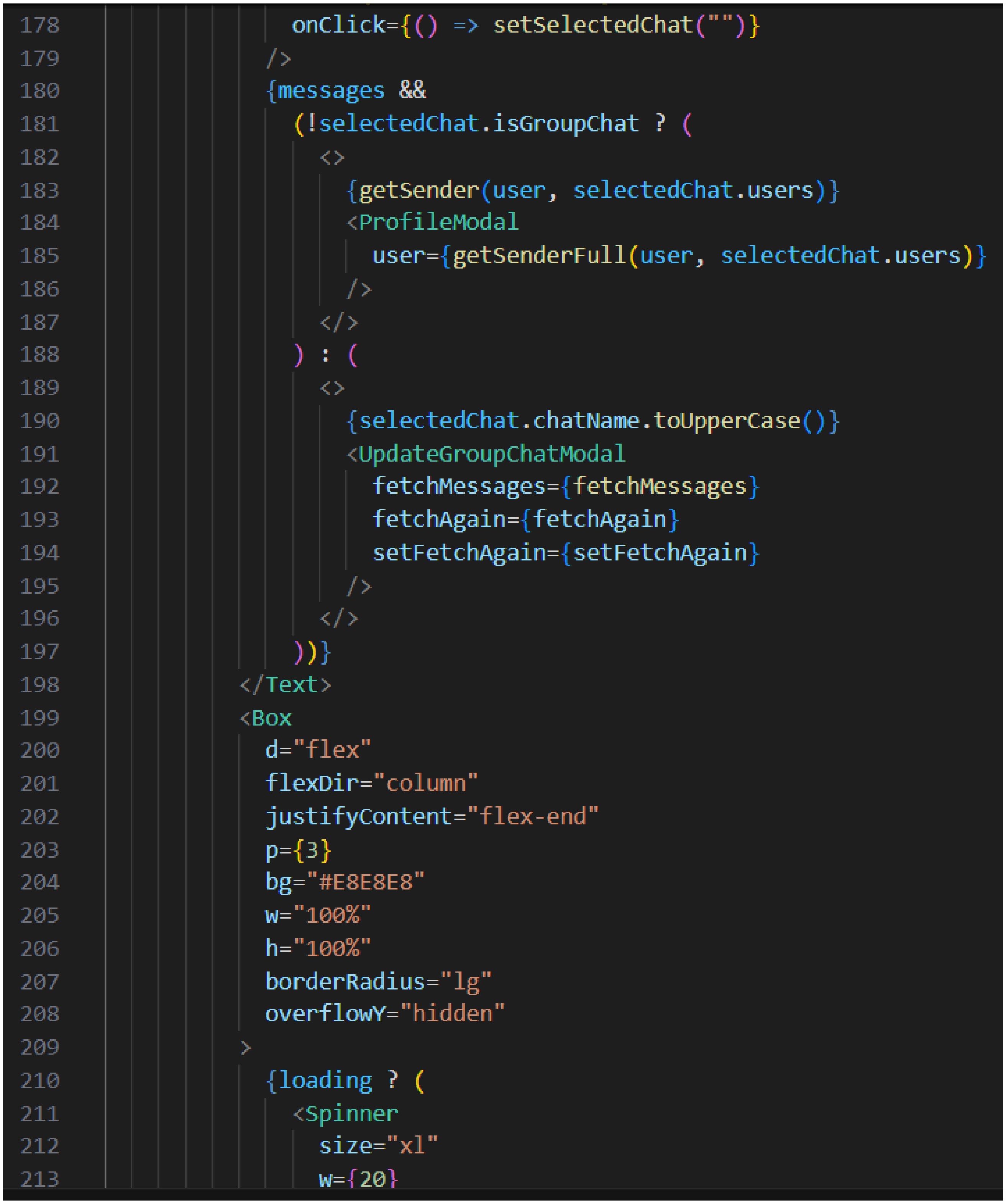
```
const sendMessage = async (event) => {
  if (event.key === "Enter" && newMessage) {
    socket.emit("stop typing", selectedChat._id);
    try {
      const config = {
        headers: {
          "Content-type": "application/json",
          Authorization: `Bearer ${user.token}`,
        },
      };
      setNewMessage("");
      const { data } = await axios.post(
        "/api/message",
        {
          content: newMessage,
          chatId: selectedChat,
        },
        config
      );
      socket.emit("new message", data);
      setMessages([...messages, data]);
    } catch (error) {
      toast({
        title: "Error Occured!",
        description: "Failed to send the Message",
        status: "error",
        duration: 5000,
        isClosable: true,
        position: "bottom",
      });
    }
  }
};
```



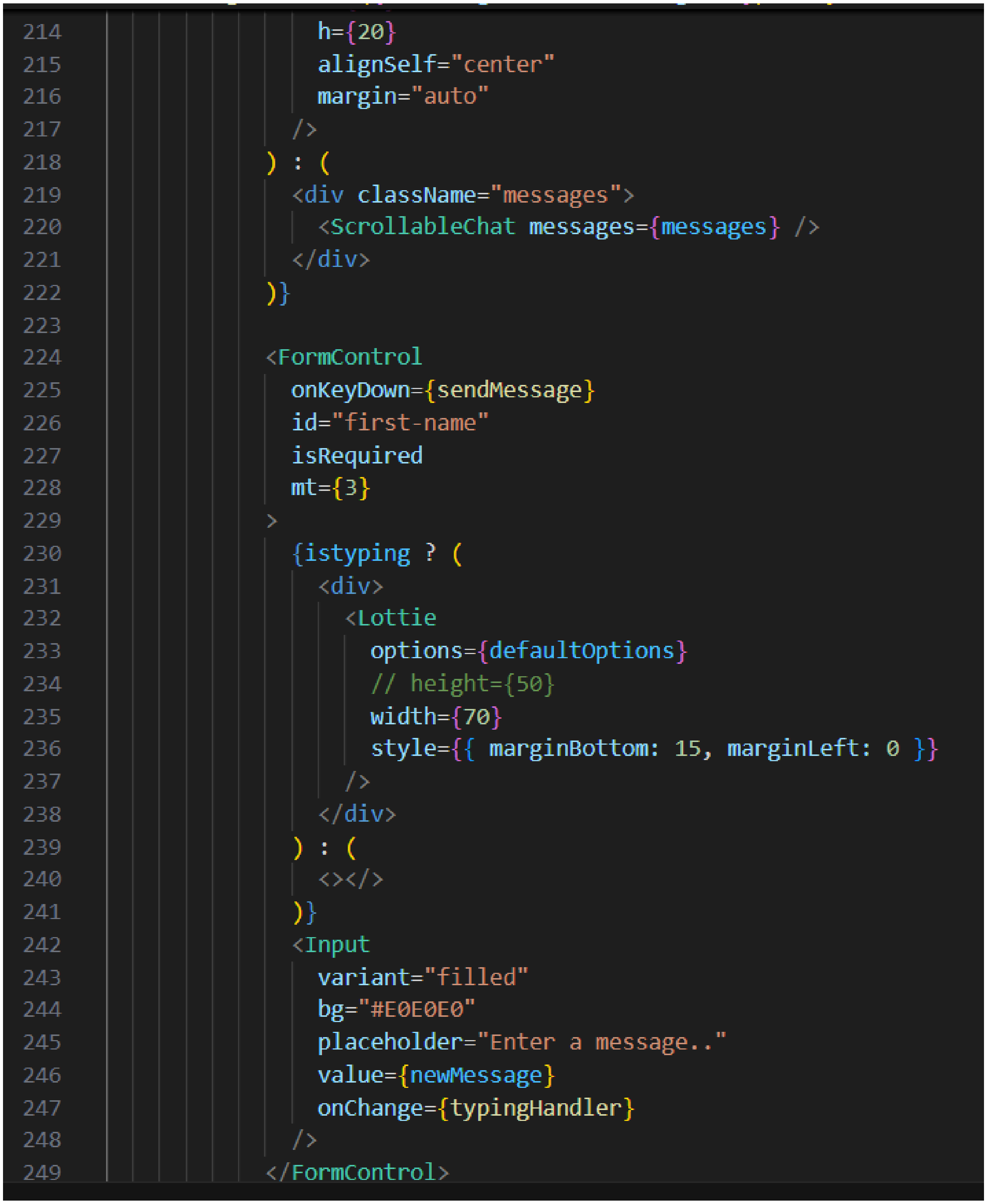
```
useEffect(() => {
  socket = io(ENDPOINT);
  socket.emit("setup", user);
  socket.on("connected", () => setSocketConnected(true));
  socket.on("typing", () => setIsTyping(true));
  socket.on("stop typing", () => setIsTyping(false));

  // eslint-disable-next-line
}, []);

useEffect(() => {
  fetchMessages();

  selectedChatCompare = selectedChat;
  // eslint-disable-next-line
}, [selectedChat]);

useEffect(() => {
  socket.on("message recieved", (newMessageRecieved) => {
    if (
      !selectedChatCompare || // if chat is not selected or doesn't match current chat
      selectedChatCompare._id !== newMessageRecieved.chat._id
    ) {
      if (!notification.includes(newMessageRecieved)) {
        setNotification([newMessageRecieved, ...notification]);
        setFetchAgain(!fetchAgain);
      }
    } else {
      setMessages([...messages, newMessageRecieved]);
    }
  });
});

const typingHandler = (e) => {
  setNewMessage(e.target.value);

```



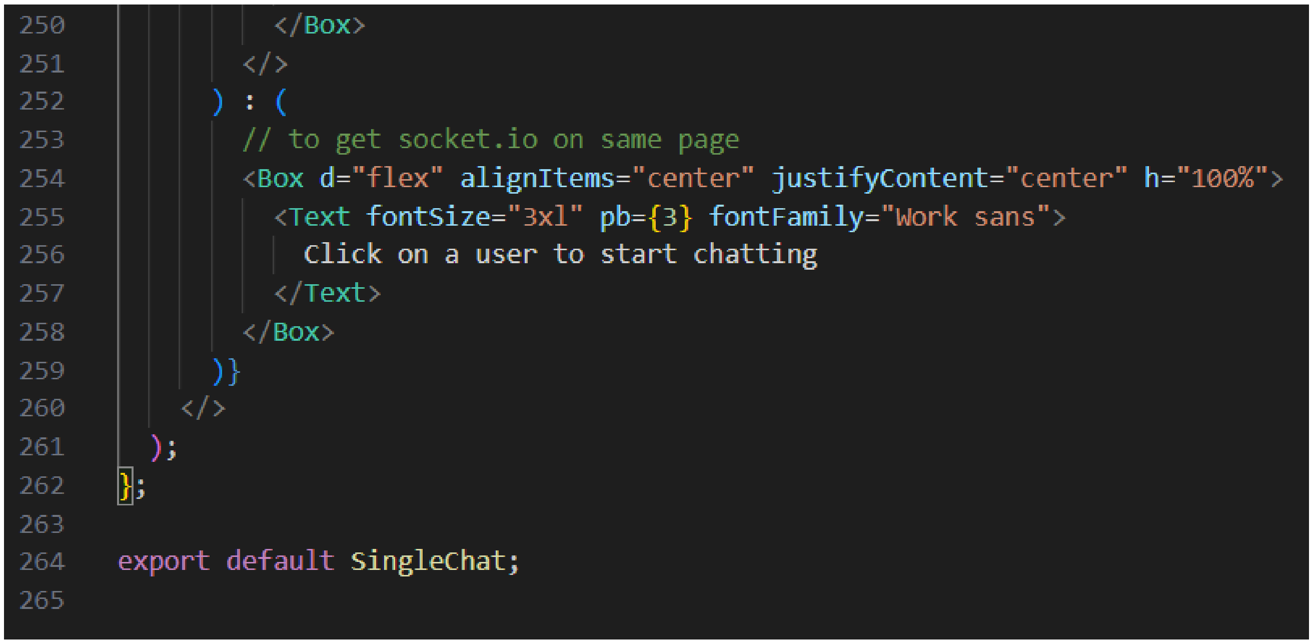
```
    if (!socketConnected) return;

    if (!typing) {
      setTyping(true);
      socket.emit("typing", selectedChat._id);
    }
    let lastTypingTime = new Date().getTime();
    var timerLength = 3000;
    setTimeout(() => {
      var timeNow = new Date().getTime();
      var timeDiff = timeNow - lastTypingTime;
      if (timeDiff >= timerLength && typing) {
        socket.emit("stop typing", selectedChat._id);
        setTyping(false);
      }
    }, timerLength);
  };

  return (
    <>
      {selectedChat ? (
        <>
          <Text
            fontSize={{ base: "28px", md: "30px" }}
            pb={3}
            px={2}
            w="100%"
            fontFamily="Work sans"
            d="flex"
            justifyContent={{ base: "space-between" }}
            alignItems="center"
          >
            <IconButton
              d={{ base: "flex", md: "none" }}
              icon={<ArrowBackIcon />}
```



```
                onClick={() => setSelectedChat("")}
              />
              {messages &&
                (!selectedChat.isGroupChat ? (
                  <>
                    {getSender(user, selectedChat.users)}
                    <ProfileModal
                      user={getSenderFull(user, selectedChat.users)}
                    />
                  </>
                ) : (
                  <>
                    {selectedChat.chatName.toUpperCase()}
                    <UpdateGroupChatModal
                      fetchMessages={fetchMessages}
                      fetchAgain={fetchAgain}
                      setFetchAgain={setFetchAgain}
                    />
                  </>
                ))}
            </Text>
            <Box
              d="flex"
              flexDir="column"
              justifyContent="flex-end"
              p={3}
              bg="#E8E8E8"
              w="100%"
              h="100%"
              borderRadius="lg"
              overflowY="hidden"
            >
              {loading ? (
                <Spinner
                  size="xl"
                  w={20}
```



```
                  h={20}
                  alignSelf="center"
                  margin="auto"
                />
              ) : (
                <div className="messages">
                  <ScrollableChat messages={messages} />
                </div>
              )}

              <FormControl
                onKeyDown={sendMessage}
                id="first-name"
                isRequired
                mt={3}
              >
                {istyping ? (
                  <div>
                    <Lottie
                      options={defaultOptions}
                      // height={50}
                      width={70}
                      style={{ marginBottom: 15, marginLeft: 0 }}
                    />
                  </div>
                ) : (
                  <></>
                )}
                <Input
                  variant="filled"
                  bg="#E0E0E0"
                  placeholder="Enter a message.."
                  value={newMessage}
                  onChange={typingHandler}
                />
              </FormControl>
```



```
            </Box>
          </>
        ) : (
          // to get socket.io on same page
          <Box d="flex" alignItems="center" justifyContent="center" h="100%">
            <Text fontSize="3xl" pb={3} fontFamily="Work sans">
              Click on a user to start chatting
            </Text>
          </Box>
        )}
      </>
    );
};

export default SingleChat;
```

CHAPTER 7
REFERENCES

[1]Real-Time Chat Application April 2023 by International Journal of Scientific Research in Science Engineering and Technology

https://www.researchgate.net/publication/370516068_Real-Time_Chat_Application

[2]Development of Chat Application by Dr. Abhay Kasetwar, Ritik Gajbhiye, Gopal Papewar, Rohan Nikhare, Priya Warade

https://www.ijraset.com/research-paper/development-of-chat-application

[3]Chat Application

Authors: Manish Kolambe, Saurabh Sable, Venkatesh Kashivale, Prajkta Khaire

https://www.ijraset.com/research-paper/chat-application

[4]Real-Time Chat Application by International Journal of Scientific Research in Science, Engineering and Technology

https://www.academia.edu/39345355/Chat_Application

[5]Using JWT (JSON Web Tokens) to authorize users and protect API routes

https://medium.com/@maison.moa/using-jwt-json-web-tokens-to-authorize-users-and-protect-api-routes-3e04a1453c3e

CHAPTER 8
BIODATA

Name	:	Sakshi Biyani
Mobile Number	:	9571484735
E-mail	:	sakshi.biyani2022@vitstudent.ac.in
Permanent Address	:	Behind Fort, Churu, Rajasthan

Name	:	Vidushi Agnihotri
Mobile Number	:	8779775094
E-mail	:	vidushi.agnihotri2022@vitstudent.ac.in
Permanent Address	:	JVLR,Andheri East,Mumbai