Front End Engineering-II /Artificial Intelligence and Machine Learning

Project Report

Semester-VI (Batch-2022)

Real Time Chat App

Supervised By:
Mr. Rahul

Submitted By:
Saikat Hazra (2210992216)
Sachidanand Gupta (2210992202)
Sachin Kumar (2210992204)
Sarbraj Singh (2210992258)

Department of Computer Science and Engineering
Chitkara University Institute of Engineering & Technology,
Chitkara University, Punjab

Abstract

This project presents a Real-Time Chat Application developed using the MERN (MongoDB, Express.js, React.js, Node.js) stack, enabling seamless communication between users. The application integrates WebSockets (Socket.io) to provide instant messaging capabilities with features like one-on-one chats, group messaging, real-time notifications, and typing indicators. Authentication is implemented using JWT (JSON Web Token), ensuring secure user access. The backend leverages Node.js and Express.js for API development, while MongoDB stores user data and chat history efficiently. The React.js frontend delivers a dynamic and responsive user interface. This project aims to provide a scalable, secure, and efficient real-time communication platform suitable for modern web applications.

Table of Contents

1. Introduction

1.1 Background

With the increasing demand for **real-time communication**, chat applications have become essential for personal, professional, and business interactions. Traditional messaging platforms rely on periodic polling, which leads to delays and inefficient use of resources. The **MERN stack** (MongoDB, Express.js, React.js, and Node.js) combined with **WebSockets (Socket.io)** provides an efficient solution for instant communication by enabling **bi-directional data transfer** between clients and servers.

1.2 Objectives

The primary objectives of the **Real-Time Chat Application using MERN Stack** are:

1. **Enable Instant Messaging:** Implement real-time one-on-one and group messaging using WebSockets (Socket.io) for seamless communication.
2. **Develop a Scalable Backend:** Use Node.js and Express.js to create a robust and efficient server capable of handling multiple concurrent users.
3. **Secure User Authentication:** Implement JWT (JSON Web Token) for secure login and authentication, along with bcrypt-based password hashing for data protection.
4. **Responsive and User-Friendly UI:** Build an interactive React.js frontend with a clean and intuitive user experience for easy navigation.
5. **Efficient Data Storage:** Utilize MongoDB to store user information, messages, and chat history efficiently, ensuring fast retrieval and scalability.
6. **Implement Real-Time Features:** Include functionalities like typing indicators**,** online/offline status, message notifications, and read receipts to enhance user engagement.
7. **Ensure Data Privacy and Security:** Use end-to-end encryption techniques and secure API routes to protect user conversations and prevent unauthorized access.
8. **Optimize Performance:** Minimize latency by optimizing API calls, database queries**,** and WebSocket events for a smooth user experience.
9. **Support Scalability and Future Enhancements:** Design the architecture to allow easy integration of additional features like voice/video calls, media sharing, and AI-powered chatbots.

10. **Cross-Platform Accessibility:** Ensure compatibility across different devices and screen sizes for a seamless experience on desktops, tablets, and mobile phones.

1.3 Significance

The **Real-Time Chat Application using MERN Stack** holds significant value in various domains due to the growing reliance on instant communication. Its key significance includes:

1. **Enhancing Communication Efficiency:** The application enables real-time, seamless**,** and instant messaging, making communication more efficient for users across personal, professional, and business environments.
2. **Scalability for Large-Scale Applications:** Built using the MERN stack, the app can scale to accommodate a large number of users, making it suitable for both small communities and large enterprises.
3. **Security and Privacy:** With features like JWT authentication, bcrypt password hashing, and secure API endpoints, the app ensures safe and private communication for users.
4. **User Engagement and Experience:** Features like typing indicators, online status**,** real-time notifications, and message read receipts enhance user interaction and engagement.
5. **Real-World Application in Various Sectors:** The chat app can be utilized in multiple domains such as customer support, business collaboration, education platforms, gaming communities, and social networking.
6. **Cost-Effective and Open-Source:** By leveraging open-source technologies like MongoDB, Express.js, React.js, and Node.js, the project offers a cost-effective solution for building customized communication platforms.

2. Problem Definition and Requirements

2.1 Problem Statement

In today's digital era, **real-time communication** has become essential for personal, professional, and business interactions. However, many existing chat applications suffer from issues such as **high latency, lack of scalability, security vulnerabilities, and poor user experience**. Traditional messaging systems rely on periodic polling, which leads to delays and inefficient resource utilization.

Furthermore, many real-time chat solutions are either **costly, lack customization, or do not provide end-to-end security** for user conversations. Businesses and developers require a **scalable, secure, and efficient** chat application that can be easily integrated into various platforms while providing a smooth user experience.

2.2 Software Requirements

To develop the **Real-Time Chat Application using MERN Stack**, the following software tools and technologies are required:

1. Frontend (Client-Side)

- **React.js** – For building the user interface.
- **Redux (Optional)** – For state management (if required).
- **React Router** – For navigation between different pages.
- **Axios or Fetch API** – For handling API requests.
- **Tailwind CSS / Material-UI / Bootstrap** – For responsive UI design.

2. Backend (Server-Side)

- **Node.js** – For backend development and running the server.
- **Express.js** – For handling API routes and middleware.
- **Socket.io** – For real-time communication using WebSockets.
- **Bcrypt.js** – For password hashing and security.
- **JSON Web Token (JWT)** – For user authentication and authorization.
- **Mongoose** – For database modeling and interaction with MongoDB.
- **Dotenv** – For managing environment variables.

3. Database

- **MongoDB** – For storing user data, chat messages, and related information.
- **MongoDB Atlas (Optional)** – For cloud-based database hosting.

2.3 Hardware Requirements

To develop and deploy the **Real-Time Chat Application using MERN Stack**, the following hardware resources are required:

1. Development Machine (Local System)

- **Processor:** Intel Core i5 (8th Gen or higher) / AMD Ryzen 5 or equivalent
- **RAM:** Minimum 8GB (Recommended 16GB for smooth multitasking)
- **Storage:** Minimum 256GB SSD (Recommended 512GB SSD or higher for faster performance)
- **Graphics Card:** Integrated GPU (Dedicated GPU optional for better UI rendering)
- **Operating System:** Windows 10/11, macOS, or Linux (Ubuntu preferred for backend development)
- **Internet Connection:** Stable broadband connection for package installations, API calls, and real-time data transfer

2. Server Requirements (For Hosting Backend & Database)

- **Processor:** Quad-core processor (Intel Xeon, AMD EPYC, or equivalent)
- **RAM:** Minimum 4GB (Recommended 8GB or more for handling multiple concurrent users)
- **Storage:** Minimum 50GB SSD (Recommended 100GB+ SSD for scalability)
- **Network Bandwidth:** Minimum 10Mbps (Higher bandwidth required for large-scale deployment)
- **Hosting Provider:** AWS, DigitalOcean, Heroku, Render, Railway, or any VPS with Node.js & MongoDB support

2.4 Data Sets

Since this is a **real-time chat application**, the dataset primarily consists of **user-generated data** rather than predefined datasets. The key datasets required for the application are:

1. User Data

- User ID (Unique Identifier)
- Username

- Email
- Password (Hashed using bcrypt)
- Profile Picture (URL or Base64)
- Online/Offline Status
- Last Seen Timestamp
- Created At / Updated At

2. Chat Messages Data

- Message ID (Unique Identifier)
- Sender ID (User who sent the message)
- Receiver ID(s) (For individual or group chat)
- Message Content (Text, Media, or Attachments)
- Timestamp (Sent and Delivered time)
- Message Status (Sent, Delivered, Read, Deleted)
- Attachments (Image, Video, File URL, etc.)

3. Group Chats Data (If Group Chat is Implemented)

- Group ID (Unique Identifier)
- Group Name
- Group Admin(s)
- List of Members (User IDs)
- Messages (Linked to Message Dataset)
- Created At / Updated At

4. Media and File Data

- File ID (Unique Identifier)
- User ID (Uploader)
- File Type (Image, Video, Document, etc.)
- File URL (Cloud Storage Link or Base64 Encoding)
- Size (In MB/KB)
- Timestamp

3. Proposed Design / Methodology

3.1 System Architecture

The system is divided into the following components:

1. **Frontend (Client) –** React.js
2. **Backend (Server) –** Node.js & Express.js
3. **Database –** MongoDB
4. **Real-Time Communication –** Socket.io (WebSockets)
5. **Authentication –** JWT (JSON Web Token)
6. **Cloud Storage (Optional) –** AWS S3 / Firebase Storage
7. **Hosting & Deployment –** Vercel (Frontend), Render/Heroku (Backend), MongoDB Atlas (Database)

3.2 Core Features and Functionalities

1. User Authentication & Security

i.)**Sign Up & Login** – Users can register and log in securely using **JWT authentication**.

ii.)**Password Encryption** – Uses **bcrypt.js** to hash passwords before storing them in MongoDB.

iii.)**Token-Based Authentication** – Users receive a **JWT token** after login for secure API access.

iv.)**Forgot Password & Reset** – Allows users to reset passwords using an email verification system.

2. Real-Time Chat Functionalities

i.) **One-on-One Chat** – Users can send direct messages in real time.

ii.)**Group Chat** – Supports group conversations with multiple participants.

iii.)**Instant Messaging (WebSockets)** – Uses **Socket.io** for **real-time, bi-directional** communication.

iv.)**Typing Indicator** – Displays a "User is typing..." status to improve engagement.

v.)**Online/Offline Status** – Shows when users are online, offline, or last seen.

vi.)**Message Read Receipts** – Indicates if a message has been sent, delivered, or read.

vii.)**Message Timestamps** – Displays the time when a message was sent.

3. Message Management

i.) **Text Messaging** – Supports sending and receiving text-based messages.

ii.) **Media & File Sharing** – Allows users to send **images, videos, PDFs, and other files** (stored in AWS S3/Firebase).

iii.) **Message Deletion & Editing** – Users can delete or edit sent messages within a limited time.

iv.) **Pinned Messages** – Users can pin important messages in a chat.

v.) **Message Search & Filters** – Enables searching chat history by keywords, date, or sender.

4. Group Chat Features

i.) **Create & Manage Groups** – Users can create groups, add/remove members, and assign admins.

ii.) **Admin Controls** – Group admins can **mute, remove members, or delete messages**.

iii.) **Group Notifications** – Users receive updates on **new messages, added members, or admin changes**.

3.3 Technology Stack

1. Frontend (Client-Side)

- React.js – UI development
- Redux / Context API – State management
- Tailwind CSS / Material-UI / Bootstrap – Styling & UI components
- Axios / Fetch API – API communication
- Socket.io Client – Real-time messaging

2. Backend (Server-Side)

- Node.js – JavaScript runtime environment
- Express.js – Backend framework for handling API requests
- Socket.io – WebSockets for real-time communication
- JWT (JSON Web Token) – Authentication & security
- bcrypt.js – Password encryption

3. Database

- MongoDB (NoSQL Database) – Stores user, message, and chat data
- Mongoose – ODM for MongoDB

3.4 Development Methodology

Phase 1: Planning & Requirement Analysis

- Define project scope, objectives, and user requirements.
- Identify core features and functionalities.
- Select technology stack (MERN, Socket.io, JWT, etc.).
- Create wireframes & UI/UX design.

Phase 2: System Design & Architecture

- Design the frontend UI structure (React.js + Redux).
- Define the backend API routes & database schema (Node.js + MongoDB).
- Plan WebSockets architecture for real-time messaging.

Phase 3: Development (Sprint-Based)

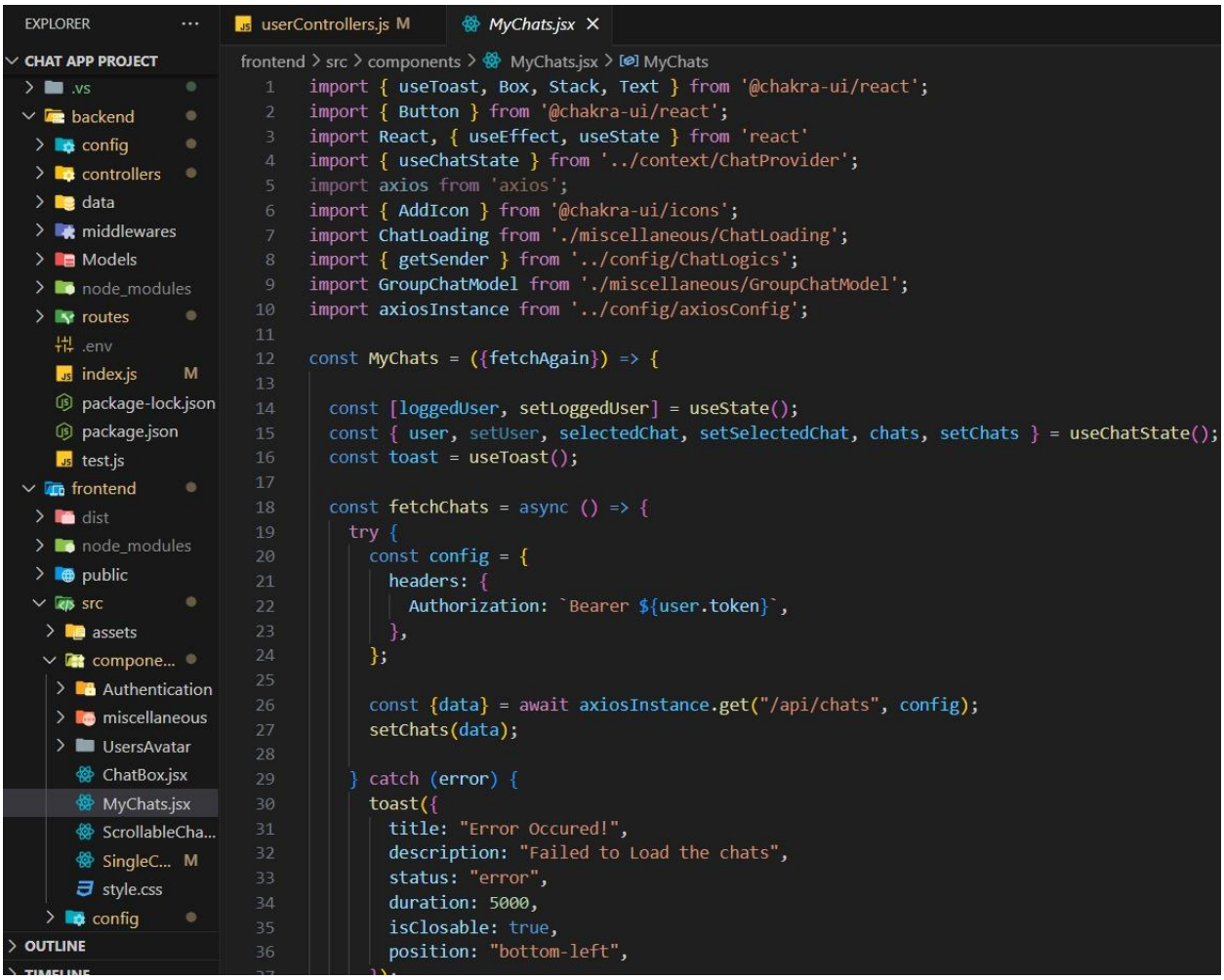
- **Sprint 1:** User authentication (JWT, bcrypt, MongoDB setup).
- **Sprint 2:** Real-time messaging (Socket.io integration, chat UI).
- **Sprint 3:** Group chat, message status, typing indicators.
- **Sprint 4:** File sharing, notifications, and search.
- **Sprint 5:** Security enhancements, performance optimization.

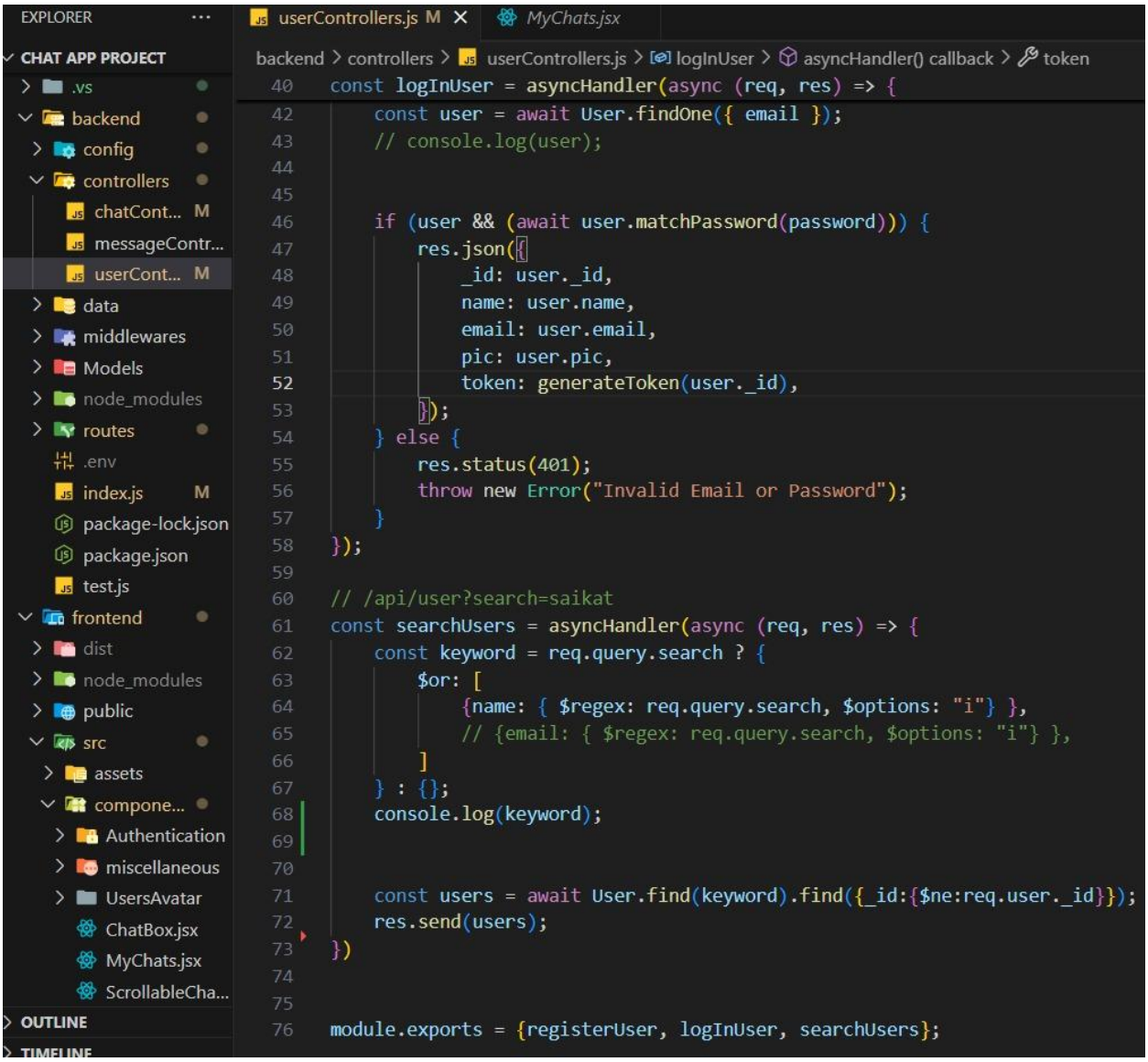
3.5 Security Measures

i.) JWT (JSON Web Token) Authentication – Secure user sessions.
ii.) Bcrypt.js for Password Hashing – Encrypt passwords before storing.
iii.) Role-Based Access Control (RBAC) – Define user roles (admin, user, guest).
iv.) Session Expiration & Auto Logout – Invalidate expired sessions.

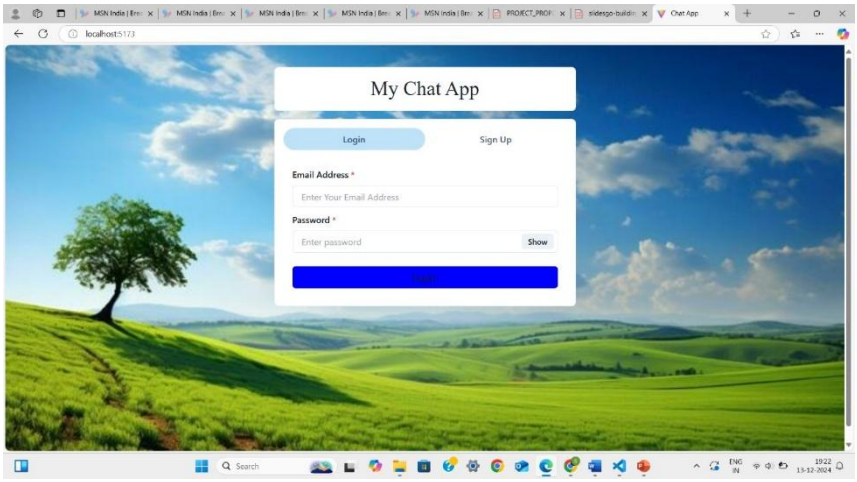
4. Result

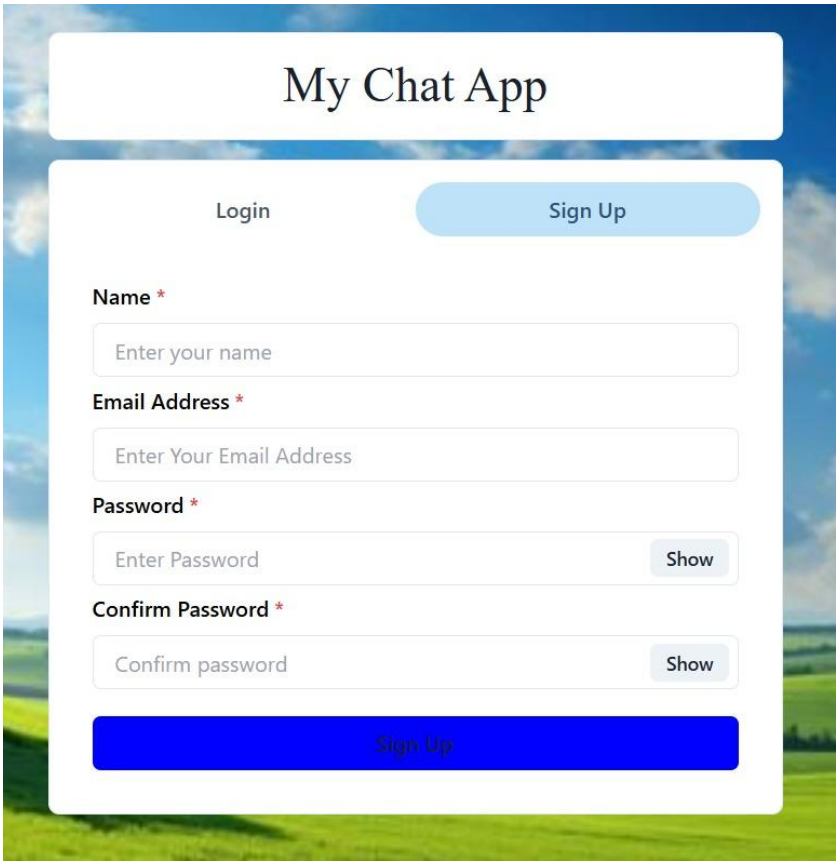
4.1 Code snippets:

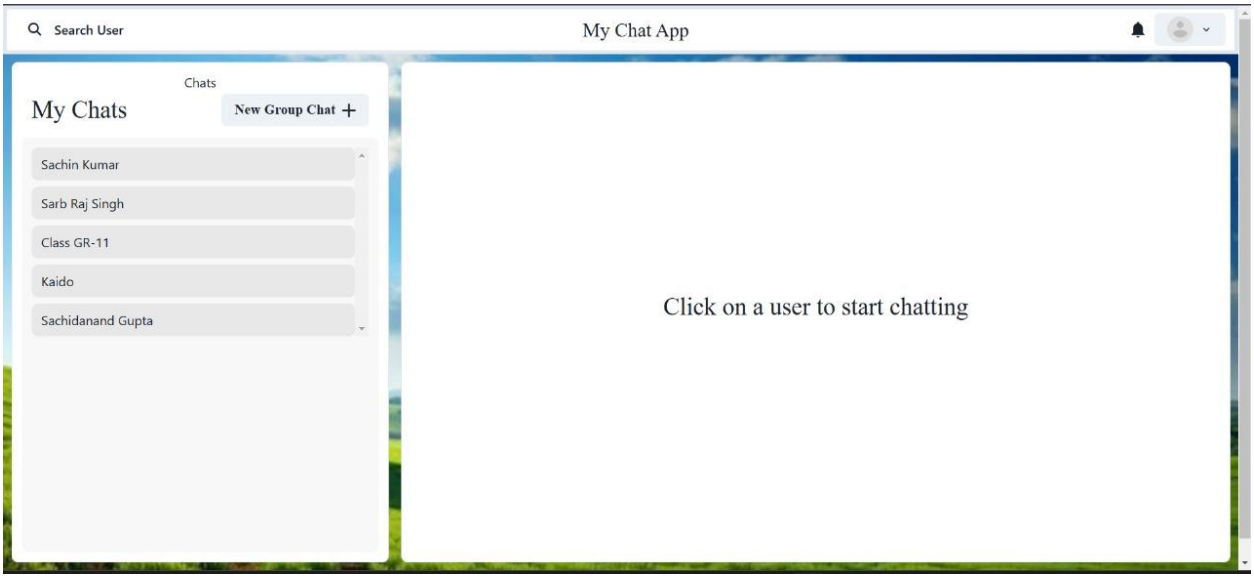

EXPLORER
userControllers.js M
MyChats.jsx
CHAT APP PROJECT
frontend > src > components > MyChats.jsx > MyChats
.vs
backend
config
controllers
data
middlewares
Models
node_modules
routes
.env
index.js M
package-lock.json
package.json
test.js
frontend
dist
node_modules
public
src
assets
compone...
Authentication
miscellaneous
UsersAvatar
ChatBox.jsx
MyChats.jsx
ScrollableCha...
SingleC... M
style.css
config
OUTLINE
1 import { useToast, Box, Stack, Text } from '@chakra-ui/react';
2 import { Button } from '@chakra-ui/react';
3 import React, { useEffect, useState } from 'react'
4 import { useChatState } from '../context/ChatProvider';
5 import axios from 'axios';
6 import { AddIcon } from '@chakra-ui/icons';
7 import ChatLoading from './miscellaneous/ChatLoading';
8 import { getSender } from '../config/ChatLogics';
9 import GroupChatModel from './miscellaneous/GroupChatModel';
10 import axiosInstance from '../config/axiosConfig';
11
12 const MyChats = ({fetchAgain}) => {
13
14 const [loggedUser, setLoggedUser] = useState();
15 const { user, setUser, selectedChat, setSelectedChat, chats, setChats } = useChatState();
16 const toast = useToast();
17
18 const fetchChats = async () => {
19 try {
20 const config = {
21 headers: {
22 Authorization: `Bearer ${user.token}`,
23 },
24 };
25
26 const {data} = await axiosInstance.get("/api/chats", config);
27 setChats(data);
28
29 } catch (error) {
30 toast({
31 title: "Error Occured!",
32 description: "Failed to Load the chats",
33 status: "error",
34 duration: 5000,
35 isClosable: true,
36 position: "bottom-left",

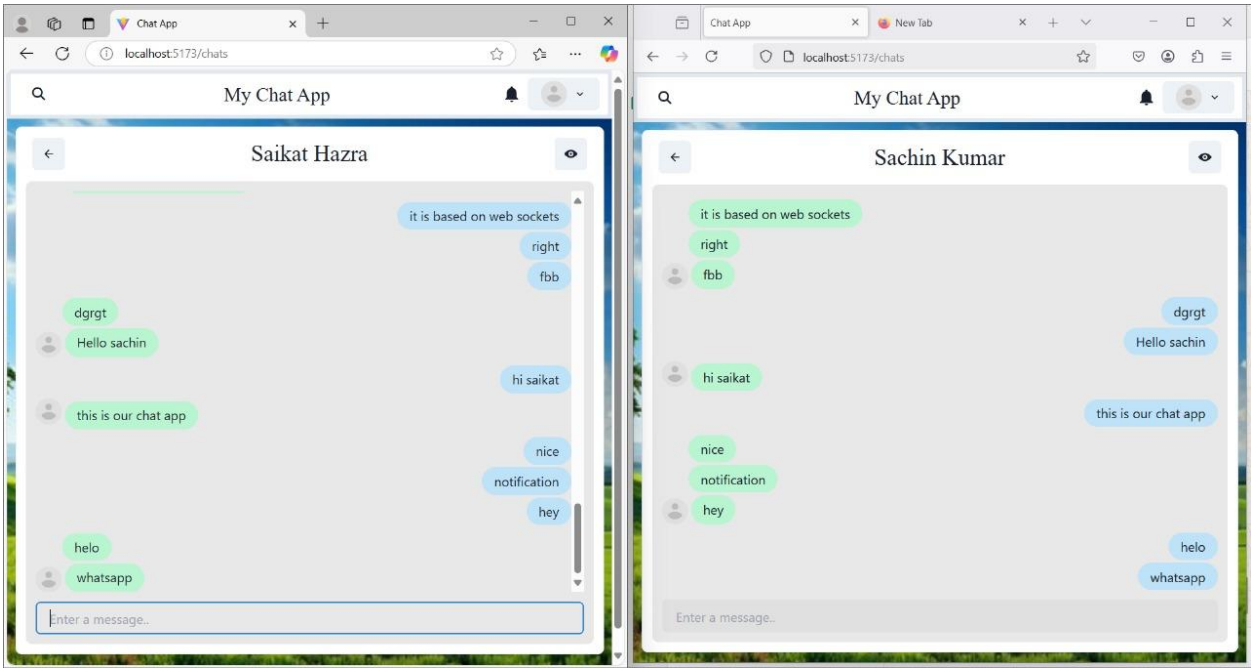

EXPLORER
userControllers.js M
MyChats.jsx
CHAT APP PROJECT
backend > controllers > userControllers.js > logInUser > asyncHandler() callback > token
.vs
backend
config
controllers
chatCont... M
messageContr...
userCont... M
data
middlewares
Models
node_modules
routes
.env
index.js M
package-lock.json
package.json
test.js
frontend
dist
node_modules
public
src
assets
compone...
Authentication
miscellaneous
UsersAvatar
ChatBox.jsx
MyChats.jsx
ScrollableCha...
OUTLINE
TIMELINE
40 const logInUser = asyncHandler(async (req, res) => {
42 const user = await User.findOne({ email });
43 // console.log(user);
44
45
46 if (user && (await user.matchPassword(password))) {
47 res.json({
48 _id: user._id,
49 name: user.name,
50 email: user.email,
51 pic: user.pic,
52 token: generateToken(user._id),
53 });
54 } else {
55 res.status(401);
56 throw new Error("Invalid Email or Password");
57 }
58 });
59
60 // /api/user?search=saikat
61 const searchUsers = asyncHandler(async (req, res) => {
62 const keyword = req.query.search ? {
63 $or: [
64 {name: { $regex: req.query.search, $options: "i"} },
65 // {email: { $regex: req.query.search, $options: "i"} },
66]
67 } : {};
68 console.log(keyword);
69
70
71 const users = await User.find(keyword).find({_id:{$ne:req.user._id}});
72 res.send(users);
73 })
74
75
76 module.exports = {registerUser, logInUser, searchUsers};

4.2 Project Snippets:






Search User
My Chat App
Chats
My Chats
New Group Chat +
Sachin Kumar
Sarb Raj Singh
Class GR-11
Kaido
Sachidanand Gupta
Click on a user to start chatting


Chat App
localhost:5173/chats
My Chat App
Saikat Hazra
it is based on web sockets
right
fbb
dgrgt
Hello sachin
hi saikat
this is our chat app
nice
notification
hey
helo
whatsapp
Enter a message..
Chat App
New Tab
localhost:5173/chats
My Chat App
Sachin Kumar
it is based on web sockets
right
fbb
dgrgt
Hello sachin
hi saikat
this is our chat app
nice
notification
hey
helo
whatsapp
Enter a message..

5. References:

- “Run JavaScript everywhere,” Nodejs.org. [Online]. Available: https://nodejs.org/en. [Accessed: 18-Sep-2024].
- “React,” React.dev. [Online]. Available: https://react.dev/. [Accessed: 18-Sep- 2024].
- “Express - Node.js web application framework,” Expressjs.com. [Online]. Available: https://expressjs.com/. [Accessed: 18-Sep-2024].
- “Documentation - Tailwind CSS,” Tailwindcss.com. [Online]. Available: https://v2.tailwindcss.com/docs. [Accessed: 18-Sep-2024].
- “Github Docs,” github.com Available: https://docs.github.com/en