RAMAIAH
College of Arts, Science & Commerce - Autonomous

A Project Report on

# REALTIME CHAT WEB APPLICATION

Submitted in partial fulfilment of the requirements for the award of

**BACHELOR OF COMPUTER APPLICATIONS**

**FROM**

**BENGALURU CITY UNIVERSITY**

**By**

| | | |
|---|---|---|
| Suman Maharana | Akash Kumar Jaiswal | Abhimanyu |
| U18EV22S0199 | U18EV22S0237 | U18EV22S0237 |

**Under the Esteemed Guidance of**

BASAVARAJ C M
HEAD OF DEPARTMENT

RAMAIAH
College of Arts, Science
& Commerce - Autonomous

# DECLARATION

The project titled **REALTIME CHAT WEB APPLICATION** is developed by us in the partial fulfilment of **Bengaluru City University**.

It is a systematic work carried out by us under the guidance of **BASAVARAJ C M, HEAD OF DEPARTMENT** in Computer Science Department**, MS RAMAIAH COLLEGE OF ARTS, SCIENCE AND COMMERCE - Autonomous**, Bengaluru-54

**We declare that this project has not been submitted to any Degree or Diploma to Bengaluru City University or any other University.**

**Date:** / / 2025
**Place :** Bengaluru

Submitted By

| | | |
|---|---|---|
| **Signature** | **Signature** | **Signature** |
| Suman Maharana | Akash Kumar Jaiswal | Abhimanyu |
| U18EV22S0199 | U18EV22S0237 | U18EV22S0237 |

RAMAIAH
College of Arts, Science
& Commerce - Autonomous

# CERTIFICATE

This is to certify that **SUMAN MAHARANA**, bearing USN **U18EV22S0199** and **AKASH KUMAR JAISWAL**, bearing USN **U18EV22S0237** and **ABHIMANYU**, bearing USN **U18EV22S0237** have successfully completed the project titled **REALTIME CHAT WEB APPLICATION** for the 6th Semester, in partial fulfilment of the requirements for the award of the degree in **Bachelor of Computer Applications**, awarded by **Bengaluru City University** for the Academic Year 2024-25

| **GUIDE** | **HOD** |
|---|---|
| Basavaraj C M | Prof. Basavaraj C M |

**External Examiners**

1.

2.

**Examination Centre Name** :

**Date** :

RAMAIAH
College of Arts, Science
& Commerce - Autonomous

# ACKNOWLEDGEMENT

The Department of Computer Science is often known for its complexity and the time commitment it demands. The successful development of our project titled **REALTIME CHAT WEB APPLICATION** would not have been possible without the support, guidance, and encouragement of several kind and helpful individuals. We take this opportunity to express our heartfelt gratitude to all of them.

We are deeply thankful to our management and trustees for providing us with the opportunity and resources to undertake and complete this project successfully.

We extend our sincere thanks to our respected Principal for her constant support and encouragement throughout the project.

We would also like to express our gratitude to our Head of the Department, **Prof. Basavaraj C M**, Department of Computer Science, for his valuable guidance, motivation, and continuous support. Without his encouragement and mentorship, this project would not have been possible.

RAMAIAH
College of Arts, Science
& Commerce - Autonomous

# TABLE OF CONTENTS

# INTRODUCTION

## 1.1 Abstract

Communication has become the cornerstone of our digital society. With the exponential growth in internet users, the need for fast, reliable, and real-time communication systems has become paramount. This project focuses on building a real-time web chat application using the MERN stack (MongoDB, Express.js, React.js, Node.js) integrated with Socket.IO for real-time data exchange and Cloudinary for media uploads.

The system offers users the ability to **register/login securely**, **send and receive messages in real-time**, and **upload media files** through a user-friendly web interface. Authentication is managed using **Bcrypt for password hashing**, ensuring security. The application is designed to mimic modern chat systems and provides a strong foundation for future expansion, including group chat, voice/video calls, and more.

This report outlines the development process, challenges faced, design methodology, business logic, and future scope of this project, offering a comprehensive guide to full-stack real-time application development.

## 1.2 Project Overview

The goal of this project is to develop a robust, real-time, browser-based chat application that supports text communication and media sharing. The key features include:

- **User Authentication:** Secure login and registration using JWT and Bcrypt.
- **User Interface:** Clean and intuitive frontend built with React.
- **Chat System:** Real-time messaging using WebSockets via Socket.IO.
- **Media Uploads:** Integration with Cloudinary to handle image uploads.
- **Settings and Profile Management:** Users can manage their profile and application settings.
- **Contact Page:** Enables users to submit feedback or support queries.

### 1.2.1 Purpose of the Chat App

The application aims to solve the need for real-time, secure, and user-friendly messaging systems in modern communication. It is particularly helpful for:

- Students collaborating on projects
- Employees communicating remotely
- Internal office communication

### 1.2.2 Importance of Real-Time Communication

In today's digital-first era, users expect immediate feedback and interaction. Real-time communication brings advantages such as:

- **Instant Message Delivery:** Powered by Socket.IO, messages are delivered instantly without reloading the page.
- **Engagement:** Real-time apps foster user engagement and keep users connected.
- **Efficiency:** Reduces time delays and improves communication in collaborative environments.

### 1.2.3 Brief Feature Set

| Features | Description |
|---|---|
| User Auth | Register, login, and logout using JWT and Bcrypt. |
| User Profile | View and update profile details, including image uploads via Cloudinary. |
| Real-Time Chat | One-to-one messaging using Socket.IO for real-time data transfer. |
| Chat UI | Interactive and intuitive React-based user interface. |
| Media Upload | Upload images securely to Cloudinary and share within chats. |
| Settings Page | Basic application information. |

## 1.3 Problem Statement

Despite the widespread use of messaging platforms, several issues persist in current systems, particularly in open-source and lightweight implementations:

- **Outdated User Experience (UX) and Interface Design:** Many open-source messaging apps suffer from clunky, non-intuitive interfaces that fail to meet modern user expectations for smooth, engaging, and aesthetically pleasing experiences.
- **Fragmented Tech Stacks:** There's a lack of cohesive, centralized codebases that integrate modern technologies such as **React**, **Socket.IO**, and **MongoDB** in a lightweight, scalable full-stack architecture. This makes it difficult for developers to build, extend, or contribute effectively.
- **Lack of Entertainment Integration:** Users often switch between multiple apps for communication and entertainment (e.g., messaging and memes/news). Existing messaging platforms rarely offer built-in features for light content consumption, such as scrolling through memes or reading trending news, all within the same environment.

- **Poor Real-Time Performance in Lightweight Apps:** Many open-source solutions either overcomplicate real-time communication with bloated dependencies or lack robust WebSocket support altogether, leading to performance issues and poor scalability.

## 1.4 Scope of the Project

In Scope (Current Version):

- One-to-one chat system with real-time functionality.
- Secure user registration and login
- Display of chat history and user lists
- Sending and receiving messages instantly
- Image uploads using Cloudinary
- Responsive frontend and user-centric UI
- Full-stack integration using modern technologies

Out of Scope (Future Enhancements):

- Group messaging
- Voice and video calling
- Reactions, and chat statuses
- Mobile application version

## 1.5 Significance

There is an increasing reliance on **remote communication tools**. Whether it is for business communication, customer support, or student collaborations, a real-time chat application offers significant value. This project is important because:

- **Educational Value:** Teaches the full development lifecycle using the MERN stack and real-time systems.
- **Security First:** Integrates Bcrypt for password hashing and JWTs for secure authentication.
- **Real-Time Foundation:** Demonstrates practical WebSocket implementation for messaging.
- **Practical Use Cases:** Can be used directly by small teams, startups, or freelancers.
- **Customizability:** Unlike commercial software, the code is adaptable and extendable.

# STEPS TO DEVELOP THE PROJECT

## 2.1 Project Planning & Research Steps

Before initiating the development of the Web Chat App,it was essential to conduct comprehensive planning and research. This phase focused on identifying the problem, understanding the requirements, selecting the right technologies, and defining goals.

### 2.1.1 Requirement Gathering

- Defined the purpose of the chat app: secure, real-time communication over the web.
- Listed key features: user authentication, one-on-one messaging, media sharing.

### 2.1.2 Technology Research

- **Frontend:** React.js was selected for building a fast, responsive interface with component-based architecture.
- **Backend:** Node.js with Express.js was chosen for API development and handling real-time sockets.
- **Database:** MongoDB was selected for its flexibility and schema-less structure.
- **WebSocket:** Socket.IO was integrated for bi-directional real-time communication.
- **File Upload:** Cloudinary was used for scalable, secure image storage.
- **Security:** Bcrypt for hashing passwords and JWT for user session handling.

### 2.1.3 Design Research

- **Wireframes** and concept designs were created to map out core components: login, chat, profile, and settings and fi

### 2.1.4 Risk Analysis

- Real-time messaging implementation using sockets posed synchronization challenges.
- Cloudinary integration required safe token management for uploads.

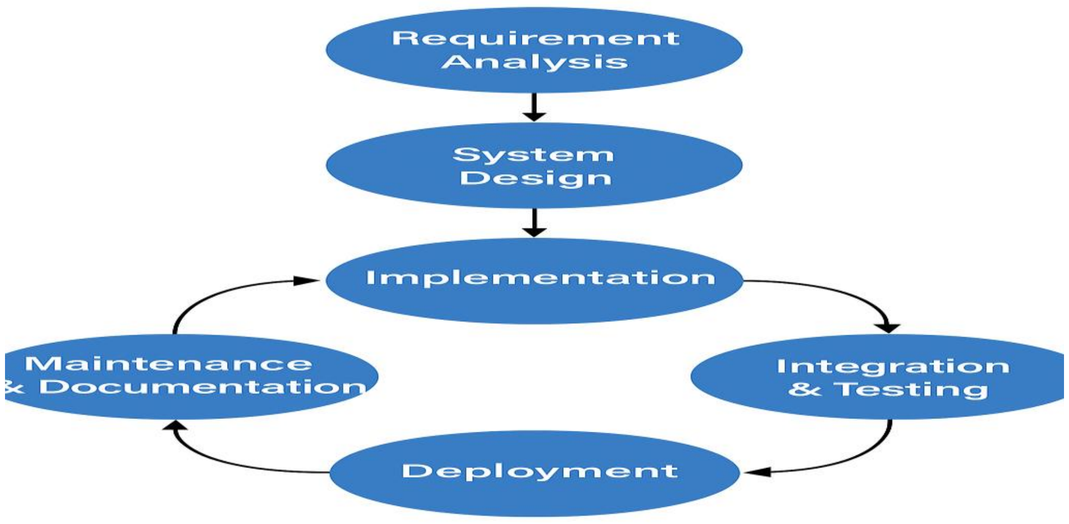
- Authentication mechanisms needed secure handling to avoid data leaks.

## 2.2 Milestone Chart

A timeline-based plan was created to guide the development phases.

| Week | Task | Status |
|---|---|---|
| Week 1 | Project planning, requirement gathering | COMPLETED |
| Week 2 | UI/UX wireframes, tech stack finalization | COMPLETED |
| Week 3 | UI design showcase, Set up backend with Node, Express, MongoDB | COMPLETED |
| Week 3 | UX testing and Integrate JWT auth, Bcrypt hashing | COMPLETED |
| Week 5 | Login, Register, Profile UI Design finalization and implementation Frontend: Login, Register, Profile UI | COMPLETED |
| Week 6 | Setup Socket.IO for real-time messaging | COMPLETED |
| Week 6 | Message persistence in database | COMPLETED |
| Week 6 | Media upload via Cloudinary | COMPLETED |
| Week 7 | Add Home & Message Pages | COMPLETED |
| Week 8 | Testing, optimization, deployment | COMPLETED |
| Week 9 | Documentation & Report Writing | COMPLETED |

## 2.3 Development Cycle Phases



## 2.4 Literature Review & Related Work

### Existing Chat Systems

Popular chat platforms like WhatsApp Web, Slack, and Discord provide robust real-time messaging features:

- **WhatsApp Web** syncs messages with the mobile device using WebSocket, supporting instant messaging and media sharing.
- **Slack** targets professional teams with channels, bots, and integrations, using a mix of WebSockets and HTTP polling.
- **Discord** offers voice, video, and text chat in persistent rooms, optimized for low-latency real-time communication using WebSockets.

### How This Project Differs

This Web Chat App combines core features of these platforms with custom implementation using the MERN stack (MongoDB, Express, React, Node.js) and Socket.IO for real-time interaction. Unlike WhatsApp Web, it is independent of mobile apps, and unlike Slack/Discord, it is lightweight and focuses only on essential chat functionalities like user authentication, media upload, and clean UI.

### MERN vs. LAMP for Chat Apps

While LAMP (Linux, Apache, MySQL, PHP) is suitable for traditional websites, **MERN** is better for real-time applications. MERN uses a single language (JavaScript) across the stack, making it faster for real-time operations. MongoDB (NoSQL) handles unstructured chat data more flexibly than MySQL.

### Importance of Socket.IO Over Polling

Polling sends repeated HTTP requests, increasing server load and latency. **Socket.IO**, built on WebSockets, creates a persistent connection, allowing low-latency, bi-directional communication. It supports real-time message delivery, typing indicators, and live updates—making it ideal for chat applications.

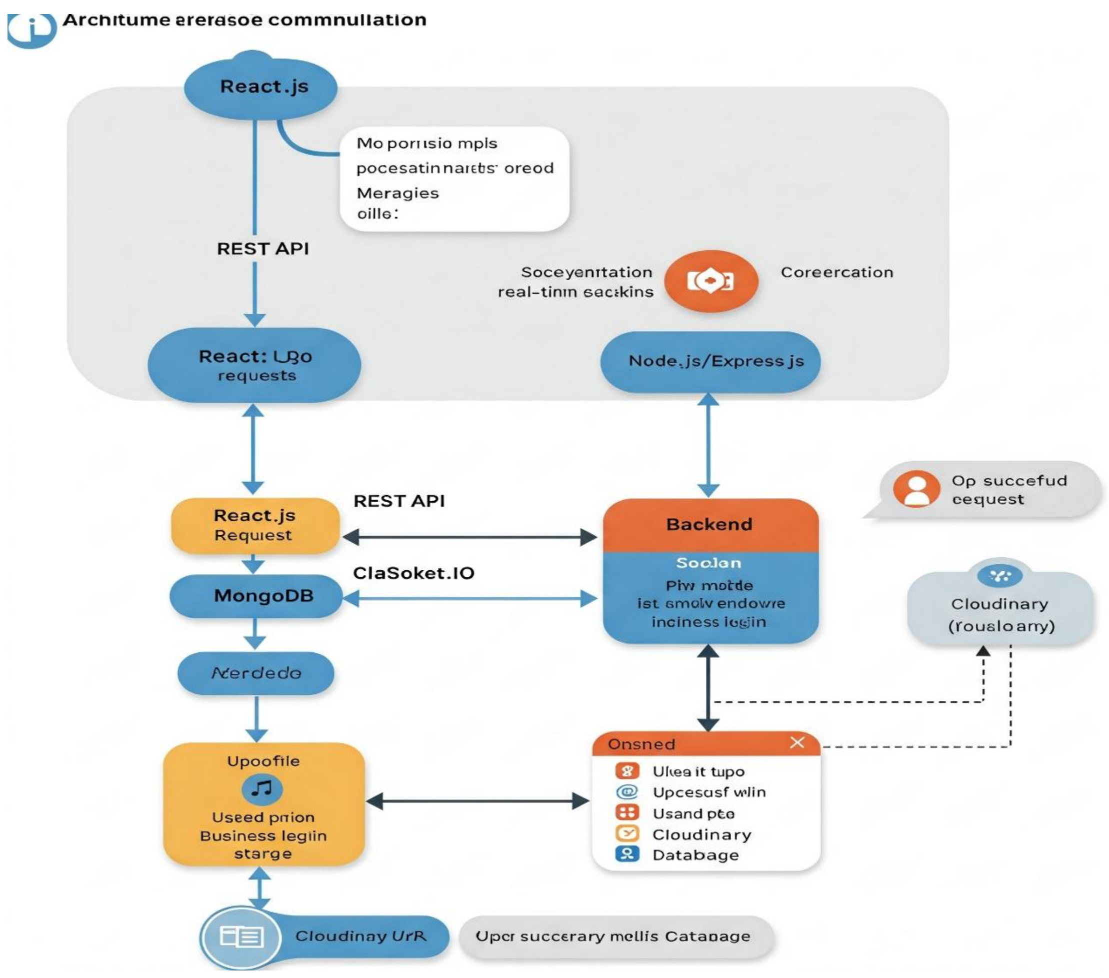
# METHODOLOGY

## 3.1 Technology Stack

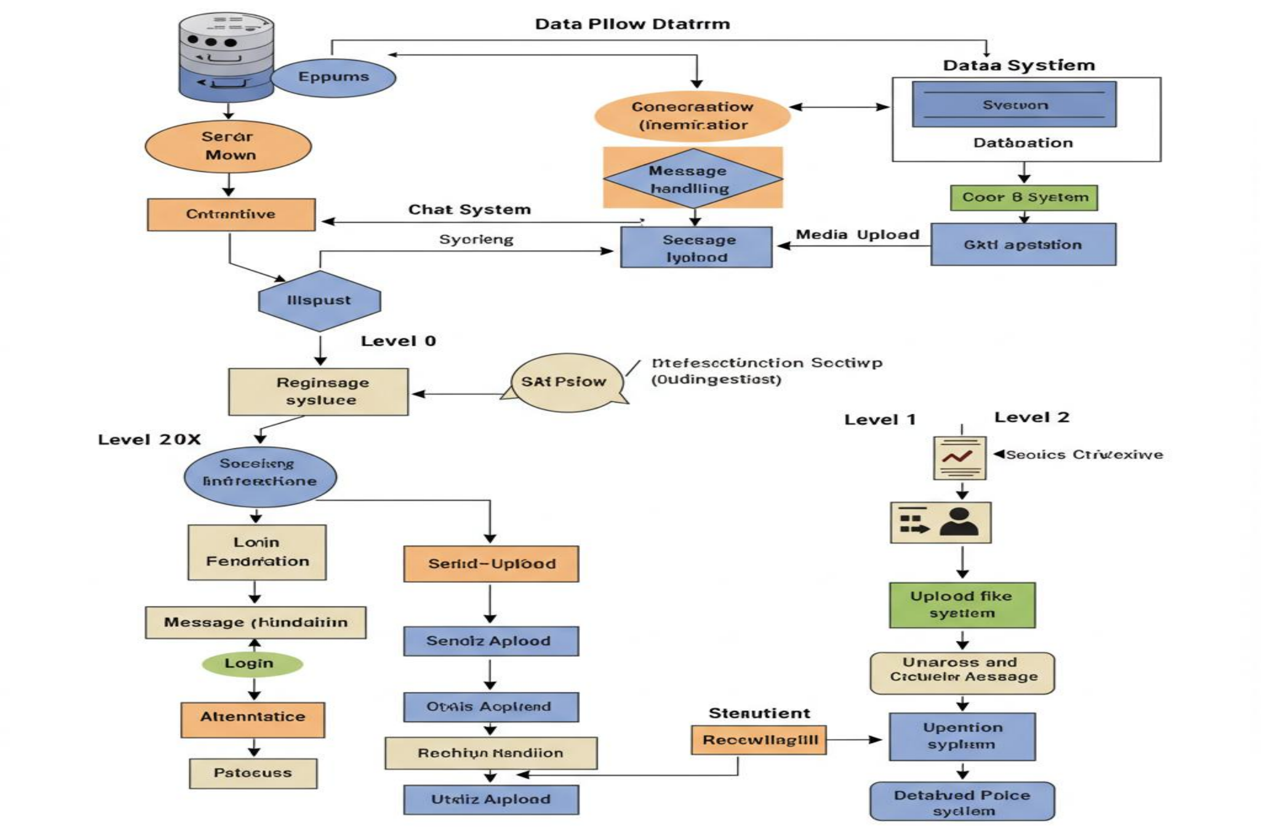
Our Web Chat App is built using the **MERN stack**, integrated with **Socket.IO** and **Cloudinary** for media management. Here's a brief overview:

- **MongoDB**: NoSQL database used to store user profiles, messages, and chat sessions.
- **Express.js**: Web framework for building RESTful APIs.
- **React.js**: Frontend JavaScript library for building interactive UI.
- **Node.js**: Backend runtime environment that handles server logic.
- **Socket.IO**: Enables real-time, bi-directional communication between client and server.
- **Cloudinary**: Manages image and video uploads, transformations, and delivery.

## 3.2 Architecture Diagram



## 3.3 Data Flow Diagram (DFD)



## Level 0

- Shows interaction between users, chat system, and database.

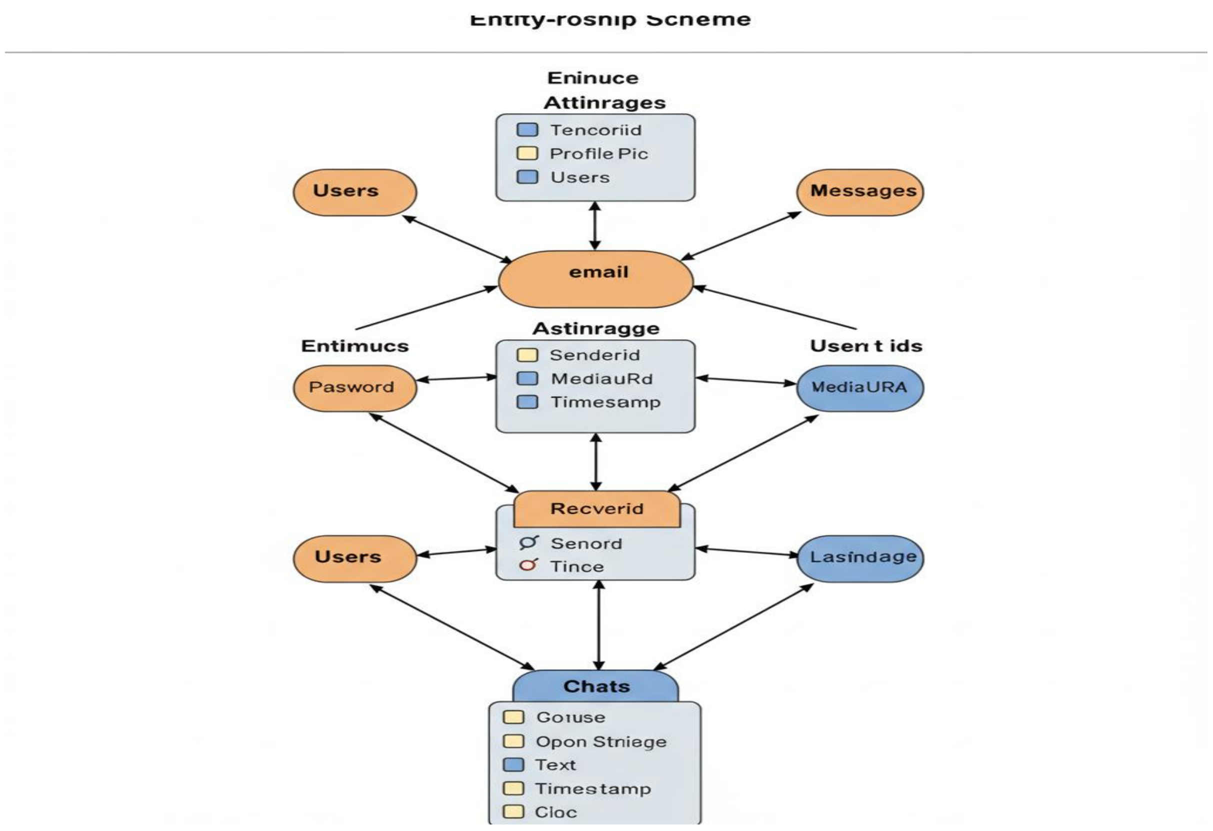
## Level 1

- Breaks down functions like authentication, message handling, and media upload.

## Level 2

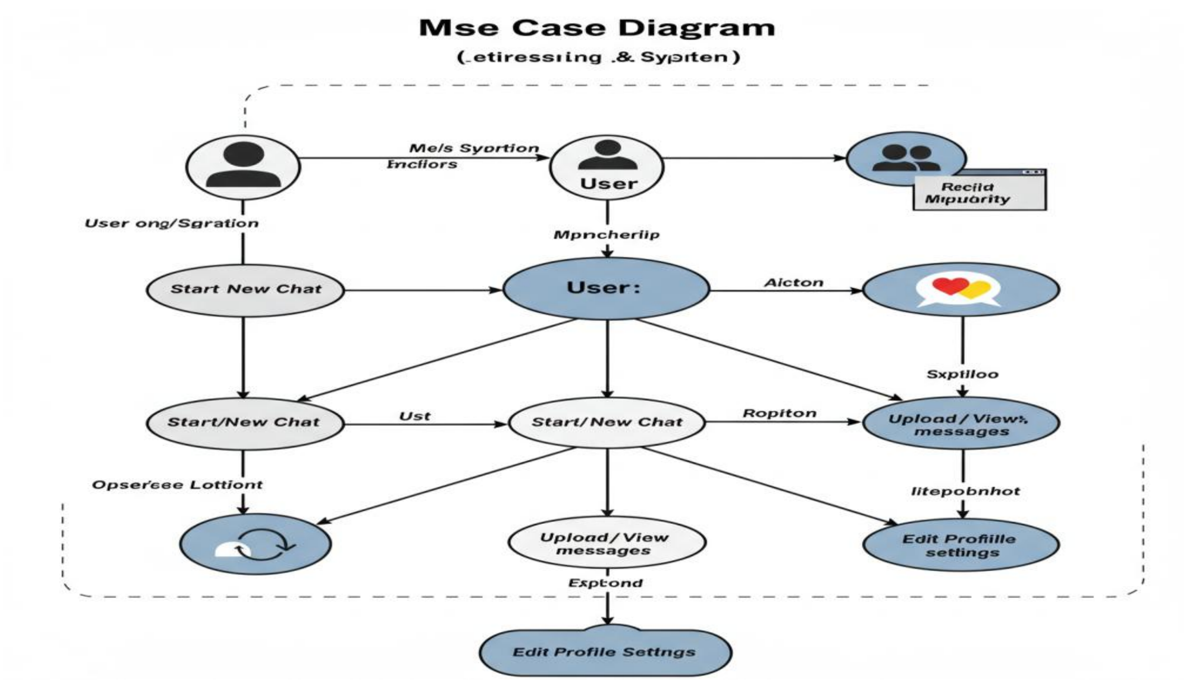
- Details interactions like:
    - Login/Register
    - Sending & receiving messages
    - Uploading media files

## 3.4 ER Diagram



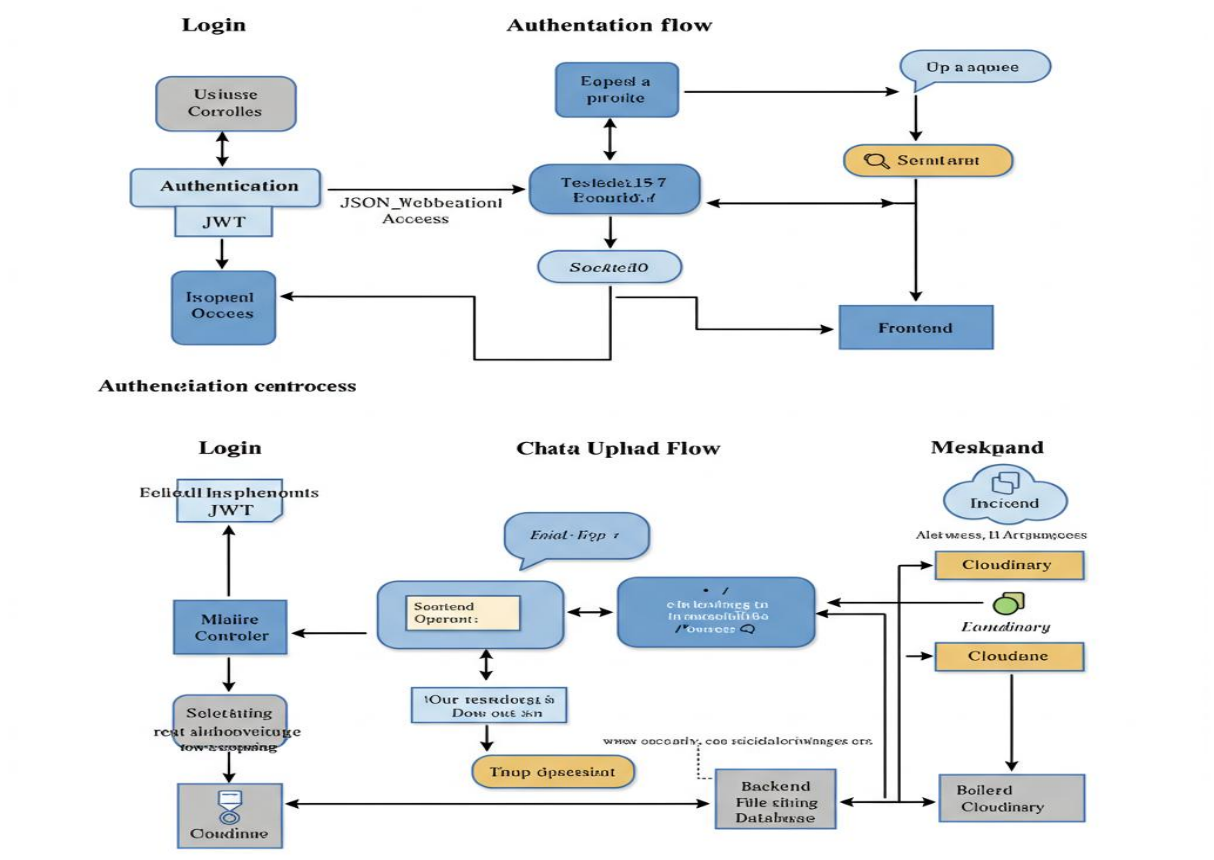
- U Users: (id, name, email, password, profilePic)
- M Messages: (id, senderId, receiverId, text, mediaURL, timestamp)
- C Chats: (id, userIds[], lastMessageId)

## 3.5 Use case Diagram



- U User Login/Registration
- S Start New Chat
- S Send/Receive Messages
- U Upload/View Media Files

## 3.6 Flowcharts



## Login Flow:

- User inputs -> Auth controller -> JWT token -> Response

## Chat Flow:

- User selects chat -> Frontend connects via Socket.IO -> Messages exchanged in real time

## Media Upload Flow:

- User selects file -> File sent to backend -> Cloudinary stores & returns URL -> URL stored in DB

## 3.6 <u>Cloudinary Media Handling Flow</u>

- User selects image/video file.
- Frontend sends file via API to Express server.
- Server uploads to Cloudinary using API key & secret.
- Cloudinary returns secure URL.
- URL stored in MongoDB with corresponding message.
- URL retrieved and rendered in chat interface.

# SYSTEM SETUP

## 4.1 Software & Hardware Requirements

Hardware Requirements**:**

- Minimum 4 GB RAM (8 GB recommended)
- 1GB free disk space
- Intel i3 processor or higher

Hardware Requirements:

- Operating System: Windows 10 / Linux / macOS
- Node.js (v18.x or higher)
- MongoDB (Atlas or local)
- Code Editor: Visual Studio Code
- Postman (for API testing)
- Git (for version control)
- Browser: Chrome/Firefox (for testing frontend)

## 4.2 Environment Setup

1. Install Node.js & npm
   - Download from https://nodejs.org
2. MongoDB Atlas Setup
   - Create a cluster on https://www.mongodb.com/cloud/atlas
   - Create a database and user
   - Get connection URI to use in backend (.env)
3. Create Project Structure
   - /client – React frontend
   - /server – Node.js backend

## 4.3 MongoDB Setup

- Create chat-app-db with collections: users, chats, messages

- Use MongoDB Compass or Mongoose to manage schemas
- Mongoose models:
  - User.js
  - Chat.js
  - Message.js

## 4.4 Node.js + Express Setup

- Create index.js and setup server
- Define routes: /auth, /chat, /message
- Connect to MongoDB using mongoose

```
mkdir server
cd server
npm init -y
npm install express mongoose dotenv bcryptjs cors socket.io
```

## 4.5 React.js Setup

- Structure pages: Login, Register, Chat, Profile, Settings.
- Use Context API for user and chat state.
- Integrate real-time updates with Socket.IO

```
npx create-react-app client
cd client
npm install axios socket.io-client react-router-dom
```

## 4.6 Socket.IO Setup

Backend:

```
const io = new Server(server, {
  cors: { origin: "http://localhost:3000" }
});
```

Frontend:

```
const socket = io("http://localhost:5000");
```

## 4.7 Cloudinary Integration

- Register at https://cloudinary.com
- Install SDK:

```
npm install cloudinary
```

- Configure in backend:

```
cloudinary.config({
  cloud_name: process.env.CLOUD_NAME,
  api_key: process.env.API_KEY,
  api_secret: process.env.API_SECRET,
});
```

- Upload media using cloudinary.uploader.upload

# IMPLEMENTATION

## 5.1 <u>Login & Register Module</u>

### 5.1.1 <u>Purpose:</u>

- Allows users to create an account or sign in securely using their credentials.

### 5.1.2 <u>Register Functionality:</u>

- Users input name, email, password, and optional profile image.
- Password is hashed using bcrypt before storing.
- Form validation is handled on both client and server.

### 5.1.3 <u>Login Functionality:</u>

- Email and password fields validated.
- If matched with DB records, JWT token is issued.
- Token is stored in localStorage for session persistence.

### 5.1.4 <u>Login Functionality:</u>

- Users input name, email, password, and optional profile image.
- Password is hashed using bcrypt before storing.
- Form validation is handled on both client and server.

### 5.1.5 Backend Apis:

- Users input name, email, password, and optional profile image.
- Password is hashed using bcrypt before storing.
- Form validation is handled on both client and server.

### 5.1.6 Backend Apis:

- Bcryptjs, JWT, MongoDB, React forms

## 5.2 <u>Profile Page</u>

**Purpose:**
Displays and allows users to edit their profile information.

**Features:**

- View profile picture, name, email
- Edit name or profile picture
- Update button triggers PUT request to backend

**Backend API:**

- PUT /api/users/:id

**Media Handling:**
Profile picture uploads use Cloudinary, and the secure URL is saved in MongoDB.

## 5.3 <u>Home / Chat UI</u>

Purpose:

Displays the main chat dashboard with contacts and active conversations.

Components:

- Sidebar: List of all recent chats or contacts
- Chat Box: Displays selected conversation
- Message Input Bar: Textbox with media upload and send button

Features:

- Real-time updates using Socket.IO
- Scroll to bottom on new messages
- Chat preview with last message & timestamp

UI Libraries:

- Tailwind CSS for layout and responsiveness

## 5.4 <u>Real-time Chat Logic</u>

**Purpose:**

Implements the WebSocket-based real-time chat system.

**How It Works:**

- Socket.IO client connects to server on component load.
- When a user sends a message:
    1. Socket emits sendMessage
    2. Backend broadcasts receiveMessage to target user
- Messages are stored in MongoDB with timestamps

**Socket Events:**

```
socket.emit("sendMessage", {
  senderId,
  receiverId,
  text,
});
```

```
socket.on("receiveMessage", (data) => {
  // update message list in UI
});
```

**Key Advantage:**

No need to refresh or poll the server – messages appear instantly.

## 5.5 <u>Media Upload</u>

**Media Upload**

**Purpose:**

Allows users to send images/videos along with text messages.

**Flow:**

1. User selects a file from device.
2. File is sent via API to Express backend.
3. Backend uploads to Cloudinary.
4. Cloudinary returns a secure URL.
5. URL is stored in message document and rendered in chat.

**Supported Formats:**

- Images: JPG, PNG
- Videos: MP4 (under size limit)

**UI Preview:**

Uploaded media appears inline in the chat box.

# CODE

## 1.FRONTEND SECTION

### App.jsx

```
import React, { useEffect } from "react";

import "./App.css";

import { Routes, Route, Navigate, useLocation } from "react-router-
dom";

import Navbar from "./components/Navbar.jsx";

import HomePage from "./pages/HomePage.jsx";

import ProfilePage from "./pages/ProfilePage.jsx";

import SettingsPage from "./pages/SettingsPage.jsx";

import LoginPage from "./pages/LoginPage.jsx";

import SignupPage from "./pages/SignupPage.jsx";

import { useAuthStore } from "./store/useAuthStore.js";

import { Loader } from "lucide-react";

import { Toaster } from "react-hot-toast";

import ChatPanel from "./components/ChatPanel.jsx";

const App = () => {

  const { authUser, checkAuth, isCheckingAuth, onlineUsers } =
useAuthStore();

  const location = useLocation();

  console.log("users", onlineUsers);

console.log({ onlineUsers });

  useEffect(() => {

    checkAuth();

  }, [checkAuth]);

  console.log({ authUser });

  if (isCheckingAuth && !authUser) {

 return ( <div className="flex justify-center items-center h-
screen">

<Loader className="size-10 animate-spin" /> </div> ); }const
hideNavbar =

    location.pathname === "/login" || location.pathname ===
"/signup";
```

```
 return (
    <div className="flex bg-gray-300 items-center relative overflow-
hidden rounded-xl">
      <div className="absolute top-0 right-0 w-[300px] h-[300px] bg-
[rgb(97,143,250)] blur-2xl pointer-events-none" />
      <div className="absolute bottom-0 left-0 w-[300px] h-[300px]
bg-[rgba(105,236,241,0.92)] blur-2xl  pointer-events-none" />
      {!hideNavbar && <Navbar />}
      <Routes><Route path="/"
  element={authUser ? <HomePage /> : <Navigate to="/login" />/>
        <Route path="/signup"
  element={!authUser ? <SignupPage /> : <Navigate to="/" />}/>
        <Route path="/login"
  element={!authUser ? <LoginPage /> : <Navigate to="/" />} />
        <Route path="/profile"
  element={authUser ? <ProfilePage /> : <Navigate to="/login" />}/>
        <Route path="/message"
  element={authUser ? <ChatPanel /> : <Navigate to="/login"/>}/>
      </Routes> <Toaster /> </div>);};
export default App;
```

## PAGES

### 1.LoginPage.jsx

```
import React, { useState } from "react";
import { Eye, EyeOff, Loader2 } from "lucide-react";
import { useNavigate } from "react-router-dom";
import { Typewriter } from "react-simple-typewriter";
import { useAuthStore } from "../store/useAuthStore.js";
const SignupPage = ({ onToggle }) => {
const [showPassword, setShowPassword] = useState(false);
const [formData, setFormData] = useState({
email: "",
password: "",});
const { login, isLoggingIn } = useAuthStore();
const handleSubmit = async (e) => {

e.preventDefault();

await login(formData);};

const navigate = useNavigate();

return (

<div className="min-h-[100dhv] w-full relative bg-white overflow-
hidden">

<div

className="absolute top-0 left-[-80px] w-[850px] h-[500px] rounded-
full mix-blend-multiply filter blur-2xl opacity-80 -translate-y-2/4"

style={{ backgroundColor: "rgba(170, 252, 255, 1)" }}></div>

<div

className="absolute top-0 right-[-80px] w-[900px] h-[500px] rounded-
full mix-blend-multiply filter blur-2xl opacity-80 -translate-y-1/4"

style={{ backgroundColor: "rgba(159, 188, 255, 1)" }}></div>

<div className="container mx-auto min-h-screen flex items-center
justify-center px-4 py-0">

<div className="w-full max-w-6xl grid grid-cols-1 md:grid-cols-2
gap-16 items-center">

<div className="flex flex-col md:flex-row items-center justify-
center text-center space-y-4 md:space-y-0 md:space-x-6 w-full max-w-
[40rem] mx-auto z-10 md:h-screen">

<img src="./chatIcon.png" className="w-[80px] sm:w-[100px] md:w-
[100px] max-w-full m-3 md:m-1"/>

<h1 className="text-black text-2xl sm:text-4xl md:text-4xl font-bold
leading-tight">

<span className="inline-block w-[300px] text-center md:text-left
lg:text-left ">

<Typewriter

words={["Unlock Possibilities,", "One Click at a Time!"]}

loop={0} cursor cursorStyle="_" typeSpeed={70}

deleteSpeed={30}

delaySpeed={1000/></span></h1></div>

<div className="bg-white p-8 rounded-2xl shadow-xl backdrop-blur-sm
bg-opacity-80 overflow-y-auto max-h-[90vh]">

<form onSubmit={handleSubmit} className="space-y-6">

<h2 className="text-3xl font-bold text-gray-900">Login</h2>

<div className="space-y-4"><div>
```

```
<label

htmlFor="email"

className="block text-sm font-medium text-gray-700">

Email</label>

<input type="email" id="email"

className="mt-1 block w-full text-black bg-white px-3 py-2 border
border-gray-300 rounded-md shadow-sm focus:outline-none focus:ring-
blue-500 focus:border-blue-500"

placeholder="you@example.com"

value={formData.email}

onChange={(e) =>

setFormData({ ...formData, email: e.target.value })}/></div><div
className="relative">

<label

htmlFor="password"

className="block text-sm font-medium text-gray-700">

Password</label>

<input type={showPassword ? "text" : "password"} id="password"

className="mt-1 block text-black bg-white w-full px-3 py-2 border
border-gray-300 rounded-md shadow-sm focus:outline-none focus:ring-
blue-500 focus:border-blue-500"

placeholder="••••••••"

value={formData.password}

onChange={(e) =>

setFormData({ ...formData, password: e.target.value }}

<button

type="button"

onClick={() => setShowPassword(!showPassword)}

className="absolute right-3 top-9 text-gray-500">

{showPassword ? <EyeOff size={20} /> : <Eye size={20}
/>}</button></div></div>

<button type="submit"

className="w-full py-2 px-4 flex items-center justify-center border
border-transparent rounded-md shadow-sm text-white font-medium

bg-gradient-to-r from-blue-500 to-blue-300 transition-all duration-
300

hover:bg-blue-500 hover:from-blue-500 hover:to-blue-500"
disabled={isLoggingIn}>
{isLoggingIn ? (<>
<Loader2 className="size-5 animate-spin" /></) : (
"Sign In")}</button>
<div className="text-center">
<p className="text-sm text-gray-600">
Dont have an account?{" "}
<button
type="button"
onClick={() => navigate("/signup")}
className="text-blue-600 hover:text-blue-500 font-medium">Sign up
</button></p></div></form></div></div></div></div>);};
export default SignupPage;
```

**2.SignupPage.jsx**

```
import React, { useState } from "react";
import { Eye, EyeOff, Loader2 } from "lucide-react";
import TermsModal from "../components/TermsModal.jsx";
import { useNavigate } from "react-router-dom";
import { Typewriter } from "react-simple-typewriter";
import { useAuthStore } from "../store/useAuthStore.js";
import toast from "react-hot-toast";
const SignupPage = ({ onToggle }) => {
const [showPassword, setShowPassword] = useState(false);
const [formData, setFormData] = useState({
fullName: "",email: "",password: "",});
const { signup, isSigningUp } = useAuthStore();
const validateForm = () => {
if (!formData.fullName.trim()) return toast.error("Full name is
required");
if (!formData.email.trim()) return toast.error("Email is required");
if (!/\S+@\S+\.\S+/.test(formData.email))
return toast.error("Invalid email format");
if (!formData.password) return toast.error("Password is required");
if (formData.password.length < 6)

return toast.error("Password must be at least 6 characters");return
true;};

const handleSubmit = async (e) => {

e.preventDefault();

const success = validateForm();

if(success === true) await signup(formData);};

const [showTerms, setShowTerms] = useState(false);

const navigate = useNavigate();return (

<div className="min-h-screen w-full relative bg-white overflow-
hidden"><div

className="absolute top-0 left-[-80px] w-[850px] h-[500px] rounded-
full mix-blend-multiply filter blur-2xl opacity-80 -translate-y-2/4"

style={{ backgroundColor: "rgba(170, 252, 255, 1)" }}></div>

<div

className="absolute top-0 right-[-80px] w-[900px] h-[500px] rounded-
full mix-blend-multiply filter blur-2xl opacity-80 -translate-y-1/4"

style={{ backgroundColor: "rgba(159, 188, 255, 1)" }}></div>

<div className="container mx-auto min-h-screen flex items-center
justify-center px-4">

<div className="w-full max-w-6xl grid grid-cols-1 md:grid-cols-2
gap-8 items-center">

<div className="flex flex-col md:flex-row items-center justify-
center text-center space-y-4 md:space-y-0 md:space-x-6 w-full max-w-
[40rem] mx-auto z-10 md:h-screen">

<img src="./chatIcon.png"

className="w-[80px] sm:w-[100px] md:w-[100px] max-w-full m-3
md:m1"/>

<h1 className="text-black text-3xl sm:text-4xl md:text-5xl font-bold
leading-tight">

<span className="inline-block w-[300px] text-center lg:text-left">

<Typewriter words={["Fast,", "Reliable,", "Responsive!"]} loop={0}

Cursor cursorStyle="_" typeSpeed={80} deleteSpeed={30}
delaySpeed={1000}/></span></h1></div>

<div className="bg-white p-8 rounded-2xl shadow-xl backdrop-blur-sm
bg-opacity-80">

<form onSubmit={handleSubmit} className="space-y-6">

<h2 className="text-3xl font-bold text-gray-900">Sign Up</h2>
```

```
<div className="space-y-4"><div><label

htmlFor="name"

className="block text-sm font-medium text-gray-700">Name</label>

<input type="text" id="name"

className="mt-1 block w-full px-3 py-2 border text-black bg-white
border-gray-300 rounded-md shadow-sm focus:outline-none focus:ring-
blue-500 focus:border-blue-500"

placeholder="name"

value={formData.fullName}

onChange={(e) =>

setFormData({ ...formData, fullName: e.target.value
})}/></div><div><label htmlFor="email"

className="block text-sm font-medium text-gray-700>Email</label>

<input type="email" id="email"

className="mt-1 block w-full text-black bg-white px-3 py-2 border
border-gray-300 rounded-md shadow-sm focus:outline-none focus:ring-
blue-500 focus:border-blue-500"

placeholder="you@example.com"

value={formData.email} onChange={(e) =>

setFormData({ ...formData, email: e.target.value })}/></div

<div className="relative"><label htmlFor="password"

className="block text-sm font-medium text-gray-700">
Password</label>

<input type={showPassword ? "text" : "password"} id="password"

className="mt-1 block text-black bg-white w-full px-3 py-2 border
border-gray-300 rounded-md shadow-sm focus:outline-none focus:ring-
blue-500 focus:border-blue-500" placeholder="••••••••"

value={formData.password} onChange={(e) =>

setFormData({ ...formData, password: e.target.value })}/>

<button type="button"

onClick={() => setShowPassword(!showPassword)}

className="absolute right-3 top-9 text-gray-500">

{showPassword ? <EyeOff size={20} /> : <Eye size={20}
/>}</button></div>

<div className="flex items-center"><input type="checkbox" id="terms"

className="h-4 w-4 bg-white appearance-none border border-gray-300
checked:bg-purple-600 checked:border-transparent text-blue-600
focus:ring-blue-500 rounded" required/><label
```

```
htmlFor="terms"
className="ml-2 block text-sm text-gray-900">
I accept the{" "}<button
onClick={() => setShowTerms(true)}
className="text-blue-600 hover:text-blue-500">
Terms</button></label></div></div>
<button type="submit"
className="w-full py-2 px-4 flex items-center justify-center border
border-transparent rounded-md shadow-sm text-white font-medium
bg-gradient-to-r from-blue-500 to-blue-300 transition-all duration-
300
hover:bg-blue-500 hover:from-blue-500 hover:to-blue-
500"disabled={isSigningUp}>
{isSigningUp ? (<><Loader2 className="size-5 animate-spin" /></>) :
("Sign Up")}</button>
<div className="text-center">
<p className="text-sm text-gray-600">
Already have an account?{" "}<button onClick={() =>
navigate("/login")}
className="text-blue-600 hover:text-blue-500 font-medium">Sign in
</button></p></div></form></div></div></div>
{showTerms && <TermsModal onClose={() => setShowTerms(false)}
/>}</div>);};
export default SignupPage;
```

## 3.ProfilePage.jsx

```
import React from "react";
import "@flaticon/flaticon-uicons/css/all/all.css";
import { Camera } from "lucide-react";
import { useAuthStore } from "../store/useAuthStore";
const ProfilePage = () => {
const { authUser, isUpdatingProfile, updateProfile,
isUpdatingProfileInfo, updateProfileInfo, isEditingProfileBio,
editingProfileBio } = useAuthStore();
const [selectedImage, setSelectedImage] = React.useState(null);
const [isEditingInfo, setIsEditingInfo] = React.useState(false);
const [isEditingBio, setIsEditingBio] = React.useState(false);
const [fullName, setFullName] = React.useState(authUser?.fullName ||
"");
const [email, setEmail] = React.useState(authUser?.email || "");
const [bio, setBio] = React.useState(authUser?.bio || "");
const isSaving = isUpdatingProfileInfo || isEditingProfileBio;
const handleImageUpload = async (e) => {
const file = e.target.files[0];  if (!file) return;
const reader = new FileReader();
reader.readAsDataURL(file);
reader.onload = async () => {
const base64Image = reader.result;
setSelectedImage(base64Image);
await updateProfile({ profilePic: base64Image }); }; };
  const handleSave = async () => {
try {
if (fullName !== authUser.fullName || email !== authUser.email) {
await updateProfileInfo({ fullName, email }); }
if (bio !== authUser.bio) {
await editingProfileBio({ bio }); }
  setIsEditingInfo(false);
setIsEditingBio(false);} catch (err) {
console.error("Error updating profile info or bio:", err);}};
return (
<div className="flex flex-col bg-white items-center justify-between
text-black h-[95vh] mr-3 my-4 w-screen rounded-[14px] border border-
gray-300 shadow-md p-10 gap-8 z-30">
<div id="heading" className="h-[13%] w-[88vw] flex flex-col items-
center">
<h1 className="text-3xl">Edit Profile</h1>
<p className="tracking-wider">Your profile information</p></div>
<div id="profile" className="h-[20%] w-[88vw] flex items-centergap
8"><div className="relative">
<img
src={selectedImage || authUser.profilePic || "/avatar.png"}
```

```
alt="Profile"

className="size-32 rounded-full object-cover border-4"/>

<label

htmlFor="avatar-upload"

className={`absolute bottom-0 right-0 bg-base-content hover:scale-
105 p-2 rounded-full cursor-pointer transition-all duration-200 ${

isUpdatingProfile ? "animate-pulse pointer-events-none" :  }` >

<Camera className="w-5 h-5 text-base-200" />

<input type="file" id="avatar-upload" className="hidden"
accept="image/*" onChange={handleImageUpload}

disabled={isUpdatingProfile}/</label></div></div>

<div id="personal-info"

className="rounded-[14px] border border-gray-300 h-[30%] w-[88vw]
flex flex-col py-2 px-6">

<div className="flex items-center justify-between mb-1">

<h3 className="text-xl">Personal Info</h3>

<div onClick={() => setIsEditingInfo(!isEditingInfo)}

className="rounded-[6px] border border-gray-300 py-1 px-5 cursor-
pointer" >

<i className="fi fi-rr-edit text-sm"></i>{" "}

{isEditingInfo ? "Cancel" : "Edit"}</div></div

<div id="label" className="flex justify-between items-center">
<label htmlFor="name"

className="flex items-center w-[40%] text-sm gap-2">

<i className="fi fi-rr-user"></i>Full Name</label>

<label htmlFor="email" className="flex items-center w-[40%] text-sm
gap-2"><i className="fi fi-rr-envelope"></i>Email</label></div><div
id="fields" className="flex justify-between h-[73%] items-center">

<div className="h-11 w-[40%] rounded-[8px] p-2 border border-gray-
300 bg-gray-100">

{isEditingInfo ? ( <input type="text" value={fullName}

onChange={(e) => setFullName(e.target.value)}

className="w-full h-full bg-transparent outline-none"/>) :
(<span>{authUser?.fullName}</span>)}</div>

<div className="h-11 w-[40%] rounded-[8px] p-2 border border-gray-
300 bg-gray-100">

{isEditingInfo ? (<input type="email" value={email}
onChange={(e) => setEmail(e.target.value)}

className="w-full h-full bg-transparent outline-none"/>) :
(<span>{authUser?.email}</span>)}</div></div></div>

<div id="bio"

className="rounded-[14px] border border-gray-300 h-[20%] w-[88vw]
flex flex-col py-2 px-6 gap-2">

<div className="flex items-center justify-between">

<h3 className="text-xl">Bio</h3>

<div onClick={() => setIsEditingBio(!isEditingBio)}

className="rounded-[6px] border border-gray-300 py-1 px-5 cursor-
pointer flex items-center justify-center gap-1">

<i className="fi fi-rr-edit text-sm"></i>{" "}

{isEditingBio ? "Cancel" : "Edit"}</div>

<div className="text-lg font-thin"> {isEditingBio ? (

<textarea

className="w-full h-full bg-transparent outline-none border border-
gray-300 rounded-lg p-2" rows={1} value={bio}

onChange={(e) => setBio(e.target.value)}/>) : (

<span>{authUser?.bio || "No bio yet"}</span>)}</div></div>

<div id="save" className="h-[15%] w-[88vw] py-2 px-6 flex justify-
between items-center">

<div id="status" className="text-blue-500">

<label htmlFor="status" className="text-black">

Member since: {authUser.createdAt?.split("T")[0]} /</label>{" "}

Active</div>

<button onClick={handleSave} disabled={isSaving}

style={{ backgroundColor: "rgba(201, 255, 255, 1)" }}

className={`rounded-[6px] border border-gray-300 py-1 px-5 h-8 ${

isSaving ? "opacity-50 cursor-not-allowed" : "cursor-pointer"}`}>

<i className="fi fi-rr-disk text-sm"></i>
Save</button></div></div>);};

export default ProfilePage;
```

## COMPONENTS

### 1.ChatContainer.jsx

```
import React, { useEffect, useState, useRef } from "react";
import { useChatStore } from "../store/useChatStore";
import MessageInput from "./MessageInput.jsx";
import ChatHeader from "./ChatHeader.jsx";
import MsgSkeleton from "./skeletons/MsgSkeleton.jsx";
import { useAuthStore } from "../store/useAuthStore.js";
import { formatMessageTime } from "../lib/utils.js";
import { HiOutlineDotsVertical } from "react-icons/hi";
const ChatContainer = () => {
const messagesEndRef = useRef(null);
const {
messages,
getMessages,
isMessagesLoading,
selectedUser,
deleteMessage,
subscribeToNewMessage,
unsubscribeFromNewMessage,
} = useChatStore();
const { authUser } = useAuthStore();
const [openDropdown, setOpenDropdown] = useState(null);
useEffect(() => {
if (selectedUser?._id) {
getMessages(selectedUser._id);
subscribeToNewMessage();  }return () => {
unsubscribeFromNewMessage();};}, [
selectedUser?._id,
getMessages,
subscribeToNewMessage,
unsubscribeFromNewMessage,]);
```

```
const handleDelete = async (messageId) => {
await deleteMessage(messageId, authUser._id);
getMessages(selectedUser._id);
setOpenDropdown(null);};
useEffect(() => {
messagesEndRef.current?.scrollIntoView({ behavior: "smooth" });
}, [messages]);
if (isMessagesLoading) {
return (
<div className="flex-1 flex flex-col overflow-auto"><ChatHeader
/><MsgSkeleton /><MessageInput /> </div>);}
return (
<div className="flex flex-col flex-1 h-full relative">
<ChatHeader /><div
className="flex-1 overflow-auto p-4 space-y-4 pb-28 bg-center"
style={{ backgroundImage: "url('/chatTheme.png')" }}>
{messages.map((message) => {
const isSender = message.senderId === authUser._id;
const sentTime = new Date(message.createdAt);
const now = new Date();
const timeDiff = (now - sentTime) / 1000; // seconds
const canDelete = isSender && timeDiff < 60 && !message.isDeleted;
  return (<div key={message._id}
className={`chat ${
isSender ? "chat-end" : "chat-start"
} relative`}>
<div className="chat-image avatar">
<div className="size-10 rounded-full border">
<img src={ isSender
? authUser.profilePic || "/avatar.png": selectedUser.profilePic ||
"/avatar.png"}alt="profile pic"/></div></div>
<div className="chat-header mb-1 flex items-center gap-2">
<time className="text-xs text-black opacity-50 ml-1">
{formatMessageTime(message.createdAt)}</time>
{isSender && (

<div className="relative group">

<button className="text-gray-500 hover:text-black"

onClick={() => setOpenDropdown((prev) =>

prev === message._id ? null : message._id)}><HiOutlineDotsVertical
size={14} /></button>{openDropdown === message._id && (

<div className="absolute z-10 right-0 top-5 bg-white shadow-md
border rounded px-2 py-1 w-36">

{canDelete ? (

<button onClick={() => handleDelete(message._id)}

className="h-8 w-full flex items-center justify-center text-sm text-
white bg-red-500 hover:bg-red-600 focus:ring-4 focus:ring-red-300
rounded-md transition duration-200 ease-in-out transform
hover:scale-105">Delete</button>) : (

<div className="text-xs text-gray-400 text-center p-1">

Deletion disabled after 1 min</div>)}</div>)}</div>)}</div>

<div

className="chat-bubble text-black flex flex-col"

style={{ background: isSender

? "linear-gradient(270deg, #FFFFFF 1.92%, #D2FFFF 30.29%, #ADFFFF
57.21%, #69FFFF 92.79%)"

: "white",}}>{message.isDeleted ? (

<p className="italic text-gray-500">message deleted</p>) : (<>

{message.text && <p>{message.text}</p>} {message.image && (

<img src={message.image} alt="attachment" className="sm:max-w-
[200px] rounded-md mt-2"/>)}</>)}</div></div>);})}

<div ref={messagesEndRef} /></div><div className="absolute bottom-0
left-0 right-0 bg-white">

<MessageInput /></div></div>);};

export default ChatContainer;
```

## 2.MessageInput.jsx

```
import React, { useRef, useState } from "react";

import { useChatStore } from "../store/useChatStore";

import { Image, Send, X, Smile } from "lucide-react";

import EmojiPicker from "emoji-picker-react";
import toast from "react-hot-toast";
const MessageInput = () => {
const [text, setText] = useState("");
const [imagePreview, setImagePreview] = useState(null);
const [showEmojiPicker, setShowEmojiPicker] = useState(false);
const fileInputRef = useRef(null);
const { sendMessage } = useChatStore();
const handleImageChange = (e) => {
const file = e.target.files[0];
if (!file?.type?.startsWith("image/")) {
toast.error("Please select an image file");return;}
  const reader = new FileReader(); reader.onloadend = () => {
setImagePreview(reader.result);};
reader.readAsDataURL(file);}
const removeImage = () => {
setImagePreview(null);
if (fileInputRef.current) fileInputRef.current.value = ""; };
  const handleSendMessage = async (e) => {
e.preventDefault();
if (!text.trim() && !imagePreview) return;
try { await sendMessage({
text: text.trim(),
image: imagePreview,});
setText("");
setImagePreview(null);
if (fileInputRef.current) fileInputRef.current.value = "";
} catch (error) {
console.error("Failed to send message:", error);} };
  const handleEmojiClick = (emojiData) => {
setText((prevText) => prevText + emojiData.emoji); };
  return (<div className="p-2 w-full relative">
{imagePreview && (<div className="mb-3 flex items-center gap-2">
<div className="relative"> <img
src={imagePreview} alt="Preview"
className="w-20 h-20 object-cover rounded-lg border border-zinc-7/>

<button onClick={removeImage}

className="absolute -top-1.5 -right-1.5 w-5 h-5 rounded-full bg-
base-300 flex items-center justify-center" type="button>

<X className="size-3" /></button></div></div>)}

  {showEmojiPicker && (<div className="absolute bottom-[60px] left-2
  z-50"><EmojiPicker onEmojiClick={handleEmojiClick} /> </div> )}

  <form onSubmit={handleSendMessage} className="flex items-center
  gap-2">

<div className="flex-1 flex items-center gap-2 relative">

<button type="button" onClick={() =>
setShowEmojiPicker(!showEmojiPicker)}

className="p-2 rounded-full hover:bg-gray-200 transition">

<Smile className="size-50" stroke="#71717A" fill="none" /></button>

<input type="text" placeholder="Type a message..."

value={text} onChange={(e) => setText(e.target.value)}

className="input w-full rounded-[22px] text-black px-4 py-6
sm:input-md input-sm focus:outline-none focus:ring-2 focus:ring-
cyan-400"

style={{ backgroundColor: "rgba(0, 0, 0, 0.05)"}/>

<input

type="file"

accept="image/*"

ref={fileInputRef}

onChange={handleImageChange}

className="hidden" />

<button

type="button"

onClick={() => fileInputRef.current?.click()}

className={`hidden sm:flex items-center justify-center rounded-full
p-2 border border-gray-300 bg-transparent hover:bg-gray-100
transition ${

imagePreview ? "text-emerald-500" : "text-zinc-400"}`}>

<Image

className="size-50"

stroke={imagePreview ? "#10B981" : "#71717A"}

fill="none"/></button></div>
```

```
<button type="submit"

className="flex items-center justify-center rounded-full p-2 bg-
transparent hover:bg-gray-100 transition"style={{

background:

"linear-gradient(123.69deg, #CAFBFF 15.61%, #94F8FF 39.09%, #ACF9FF
48.14%, #8EF7FF 58.2%, #9AF8FF 69.27%, #E2FDFF 85.37%)",}}

disabled={!text.trim() && !imagePreview}>

<Send className="size-50"

stroke={text || imagePreview ? "#10B981" : "#71717A"}

fill="none" </button></form></div> );};

export default MessageInput;
```

## 3.<u>Navbar.jsx.</u>

```
import React, { useState } from "react";

import { useAuthStore } from "../store/useAuthStore";

import { Link, useLocation } from "react-router-dom";

import SettingsPage from "../pages/SettingsPage";

const Navbar = () => {

const { authUser, logout } = useAuthStore();

const [showSettings, setShowSettings] = useState(false);

const location = useLocation();

const toggleSettings = () => {

setShowSettings(prev => !prev);

};return (

<div

className="flex flex-col items-center justify-between text-black h-
[95vh] mx-3 my-4 w-14 rounded-full border border-gray-300 shadow-md
z-50" style={{

background:

"linear-gradient(to bottom, rgba(255, 255, 255, 1) 0%, rgba(154,
243, 249, 1) 25%, rgba(120, 225, 235, 1) 50%, rgba(154, 243, 249, 1)
75%, rgba(230, 253, 255, 1) 100%)",}}>

<div className="cursor-pointer mt-[6px]"><Link to="/profile">

<img src={authUser?.profilePic || "profile.png"} alt="logo"

className="w-10 h-10 rounded-full object-cover"/></Link></div>

<div className="flex cursor-pointer flex-col py-12 gap-8 w-7">{[
{ path: "/", icon: "home.png", alt: "home" },
{ path: "/message", icon: "message.png", alt: "msg" },
{ path: "/status", icon: "status.png", alt: "status" },
{ path: "/add-contact", icon: "addContact.png", alt: "add" },
].map((item) => (
<Link key={item.path} to={item.path} className="relative flex items-
center">
{location.pathname === item.path && (
<div
style={{ backgroundColor: "#08C5F9" }}
className="absolute -right-3 w-1 h-6  rounded-full"
></div>)}
<img src={item.icon} alt={item.alt} /></Link>))}</div><div
className="flex cursor-pointer flex-col pb-3 gap-5 w-7 items-
center">
<img
src="settings.png"
onClick={toggleSettings}
alt="setting"
className="w-6 h-6"/>
{showSettings && (
<SettingsPage onClose={() => setShowSettings(false)} />)}
<div onClick={logout}
className="flex justify-center items-center w-10 h-10 rounded-full
border border-white/30 backdrop-blur-lg bg-black/10">
<img className="w-8 h-8 mt-1.5 ml-0.5" src="logout.png" alt="log"
/></div></div></div>);};
export default Navbar;
```

**4.<u>ChatPanel.jsx.</u>**

```
import React from "react";
import { useLocation } from "react-router-dom";
import { useChatStore } from "../store/useChatStore";
import Sidebar from "./Sidebar";
import NoChatSelected from "./NoChatSelected";
import ChatContainer from "./ChatContainer";
const ChatPanel = () => {
  const location = useLocation();
  const isChatOpen = location.pathname === "/message";
  const { selectedUser } = useChatStore();
  const showSidebarOnMobile = !selectedUser; // show only if no chat
selected
  return (
    <div className="flex h-[95vh] my-4 w-full"><div
        className={` h-full
          ${showSidebarOnMobile ? "flex" : "hidden"}
          md:flex transition-all
          duration-300 ease-in-out
        `}style={{width: "270px",minWidth: "270px", }} >
        <Sidebar isChatOpen={isChatOpen} /></div>
      <div
        className={`flex-1  h-full
        ${selectedUser ? "flex" : "hidden"}
        md:flex  ml-3 mr-3items-center justify-center`}
        style={{ backdropFilter: "blur(5px)">
        <div className="bg-white rounded-lg shadow-cl w-full max-w-
6xl h-[calc(100vh-2rem)]">
          <div className="flex h-full rounded-lg overflow-hidden">
            {!selectedUser ? <NoChatSelected /> : <ChatContainer />}
          </div> </div</div </div>};
export default ChatPanel;
```

## axios.js

```
import axios from 'axios';
export const axiosInstance = axios.create({
    baseURL: import.meta.env.MODE === "development" ?
'http://localhost:5001/api' : "/api",
    withCredentials: true,});
```

## STORES (STATE MANAGEMENT)

### 1.useAuthStore.js

```
import { create } from 'zustand';

import { axiosInstance } from '../lib/axios.js';

import toast from 'react-hot-toast';

import { io } from 'socket.io-client';

const BASE_URL = import.meta.env.MODE === "development" ?
'http://localhost:5001' : "/";

export const useAuthStore = create((set, get) => ({

  authUser: null,

  isSigningUp: false,

  isLoggingIn: false,

  isUpdatingProfile: false,

  isCheckingAuth: true,

  isUpdatingProfileInfo: false,

  isEditingProfileBio: false,

  onlineUsers: [],

  socket: null,

  checkAuth: async () => {

    try {

      const res = await axiosInstance.get('/auth/check');

      set({ authUser: res.data })

      get().connectSocket();

    } catch (e) {

      console.log(e, 'error checking auth')

      set({ authUser: null })

    } finally {

      set({ isCheckingAuth: false })}},

  signup: async (data) => {set({ isSigningUp: true })

    try {

      const res = await axiosInstance.post('/auth/signup', data);

      set({ authUser: res.data })

      toast.success("Account created successfully");
      get().connectSocket();
    } catch (error) {
      toast.error(error.response.data.message)
    } finally {
      set({ isSigningUp: false }); }},
  login: async (data) => {set({ isLoggingIn: true })
    try {
      const res = await axiosInstance.post('/auth/login', data);
      set({ authUser: res.data })
      toast.success("Logged in successfully");
      get().connectSocket();
    } catch (error) {
      toast.error(error.response.data.message)
    } finally { set({ isLoggingIn: false });  }},
  logout: async () => {
    try {
      await axiosInstance.post("/auth/logout");
      set({ authUser: null });
      toast.success("Logged out successfully")
      get().disconnectSocket();
    } catch (error) {
      toast.error(error.response.data.message); } },
  updateProfile: async (data) => {
    set({ isUpdatingProfile: true });
    try {
   const res = await axiosInstance.put("/auth/update-profile",data);
      set({ authUser: res.data });
      toast.success("Profile updated successfully")
    } catch (error) {
      console.log('error updating profile', error)
      toast.error(error.response.data.message);
    } finally {set({ isUpdatingProfile: false });} },
  updateProfileInfo: async (data) => {
    set({ isUpdatingProfileInfo: true });
    try {
      const res = await axiosInstance.put("/auth/update-profile-
info", data);
      set({ authUser: res.data });
      toast.success("Profile info updated successfully")
    } catch (error) {
      console.log('error updating profile info', error)
      toast.error(error.response.data.message);
    } finally {
      set({ isUpdatingProfileInfo: false });} },
  editingProfileBio: async (data) => {
    set({ isEditingProfileBio: true });
    try {
      const res = await axiosInstance.put("/auth/update-bio", data);
      set({ authUser: res.data });
      toast.success("Bio updated successfully")
    } catch (error) {
      console.log('error updating bio', error)
      toast.error(error.response.data.message);
    } finally {
      set({ isEditingProfileBio: false });}},
  connectSocket: () => {
    const { authUser } = get();
    if (!authUser || get().socket?.connected) return;
    const socket = io(BASE_URL,{
      query: {userId: authUser._id,}, });
    socket.connect()
    set({ socket: socket });
    socket.on('userConnected', (userIds) => {
  set({ onlineUsers: userIds });
      console.log('user connected', userIds)});},
  disconnectSocket: () => {
    if (get().socket?.disconnect())get().socket.disconnect();
    set({ socket: null });},}));
```

## 2.useChatStore.js

```
import { create } from 'zustand';
import toast from 'react-hot-toast';
import { axiosInstance } from '../lib/axios.js';
import { useAuthStore } from './useAuthStore.js';
export const useChatStore = create((set, get) => ({
messages: [],
users: [],
selectedUser: null,
isUsersLoading: false,
isMessagesLoading: false,
getUsers: async () => {
set({ isUsersLoading: true });
try {
const res = await axiosInstance.get("/messages/users");
console.log("fetched", res.data);
set({ users: res.data });
} catch (error) {
toast.error(error.response.data.message);
} finally {
set({ isUsersLoading: false });}},
getMessages: async (userId) => {
set({ isMessagesLoading: true });try {
const res = await axiosInstance.get(`/messages/${userId}`);
set({ messages: res.data });
} catch (error) {
toast.error(error?.response?.data?.message || 'Failed to fetch
messages');} finally {set({ isMessagesLoading: false });}},
sendMessage: async (messageData) => {
const { selectedUser, messages } = get();
try {const res = await
axiosInstance.post(`/messages/send/${selectedUser._id}`,
messageData);
set({ messages: [...messages, res.data] });
} catch (error) {
```

```
toast.error(error?.response?.data?.message || 'Failed to send
message');console.log(error);}},

deleteMessage: async (messageId) => {

const { messages } = get();

console.log("Deleting message with ID:", messageId);

if (!messageId) {

toast.error("Message ID is required to delete.");return;}

const existingMessage = messages.find((msg) => msg._id ===
messageId);

if (!existingMessage) {toast.error("Message not found.");return;}

try {

const response = await
axiosInstance.delete(`/messages/${messageId}`);

if (response.status === 200) {

const updatedMessages = messages.map((msg) =>msg._id === messageId

? { ...msg, text: null, image: null, isDeleted: true }: msg);

set({ messages: updatedMessages });

toast.success("Message deleted successfully.");}} catch (error) {

toast.error(error?.response?.data || "Failed to delete message");

console.error(error);}},

subscribeToNewMessage: () => {

const { selectedUser } = get();

if (!selectedUser) return;

const socket = useAuthStore.getState().socket;

socket.on("newMessage", (newMessage) => {

const isMessageSentFromSelectedUser = newMessage.senderId ===
selectedUser._id;

if (!isMessageSentFromSelectedUser) return;

set({ messages: [...get().messages, newMessage], })});},

unsubscribeFromNewMessage: () => {

const socket = useAuthStore.getState().socket;

socket.off("newMessage");},

setSelectedUser: (selectedUser) => set({ selectedUser }),

}));
```

## 2.BACKEND SECTION

### index.js

```
import express from 'express';
import dotenv from 'dotenv';
import authRoutes from './routes/auth.route.js';
import messageRoutes from './routes/message.route.js';
import { connectDB } from './lib/db.js';
import cookieParser from 'cookie-parser';
import cors from 'cors';
import path from 'path';
import { app,server } from './lib/socket.js';
dotenv.config();
app.use(express.json({ limit: '5mb' }));
app.use(express.urlencoded({ extended: true, limit: '5mb' }));
app.use(cookieParser());
app.use(cors({
origin: "http://localhost:5173",
credentials: true,}));
app.use('/api/auth', authRoutes);
app.use('/api/messages', messageRoutes)
const PORT = process.env.PORT || 5000;
const __dirname = path.resolve();
if(process.env.NODE_ENV==="production"){
app.use(express.static(path.join(__dirname, '../frontend/dist')));
app.get('*', (req, res) => {
res.sendFile(path.join(__dirname, "../frontend", "dist",
"index.html"));});
}
server.listen(PORT, () => {
console.log(`Server is running on URL : http://localhost:${PORT}`);
connectDB();
});
```

## ROUTES

### 1.auth.route.js

```
import express from 'express';

import { checkAuth, login, logout, signup, updateProfile,
updateProfileInfo, editingProfileBio } from
'../controllers/auth.controller.js';

import { protectRoute } from '../middleware/auth.middleware.js';

const router = express.Router();

router.post('/signup', signup);

router.post('/login', login);

router.post('/logout', logout);

router.put('/update-profile', protectRoute, updateProfile);

router.put('/update-profile-info', protectRoute, updateProfileInfo);

router.put('/update-bio', protectRoute, editingProfileBio);

router.get('/check', protectRoute, checkAuth);

export default router;
```

### 2.message.route.js

```
import express from 'express';

import { getMessages, getUsersForSidebar, sendMessage, deleteMessage
} from '../controllers/message.controller.js';

import { protectRoute } from '../middleware/auth.middleware.js';

const router = express.Router();

router.get("/users", protectRoute, getUsersForSidebar);

router.get("/:id", protectRoute, getMessages);

router.post("/send/:id", protectRoute, sendMessage);

router.delete("/:messageId", protectRoute, deleteMessage);

export default router;
```

## CONTROLLERS

### 1.auth.controller.js

```
import User from '../models/user.model.js';

import bcrypt from 'bcrypt';

import { generateToken } from '../lib/utils.js';
import cloudinary from "../lib/cloudinary.js";
export const signup = async (req, res) => {
const { fullName, email, password } = req.body;
try {
if (!fullName || !email || !password) {
return res.status(400).json({ message: "All fields are required" });
if (password.length < 6) {
return res.status(400).json({ message: "Password must be at least 6
characters long" });}
const user = await User.findOne({ email });
if (user) {
return res.status(400).json({ message: "User already exists with
this email" });}
const salt = await bcrypt.genSalt(10);
const hashedPassword = await bcrypt.hash(password, salt);
const newUser = new User({
email,
fullName,
password: hashedPassword});
if (newUser) {
generateToken(newUser._id, res);
await newUser.save();
res.status(201).json({ _id: newUser._id, email: newUser.email,
fullName: newUser.fullName, profilePic: newUser.profilePic });
} else {
return res.status(400).json({ message: "Invalid user data" });}
} catch (error) {
console.log("Error in signup controller", error.message);
res.status(500).json({ message: "Internal Server error" });}}
export const login = async (req, res) => {
const { email, password } = req.body;
try {
if (!email || !password) {return res.status(400).json({ message:
"All fields are required" });}
if (password.length < 6) {
```

```
return res.status(400).json({ message: "Password must be at least 6
characters long" });}
const user = await User.findOne({ email })if (!user) {
return res.status(400).json({ message: "Invalid credentials" });}
const isPasswordCorrect = await bcrypt.compare(password,
user.password);
if (!isPasswordCorrect) {
return res.status(400).json({ message: "Invalid credentials" });}
generateToken(user._id, res);
res.status(200).json({ _id: user._id, email: user.email, fullName:
user.fullName, profilePic: user.profilePic, });
} catch (error) {
console.log("Error in login controller", error.message);
res.status(500).json({ message: "Internal Server error" });}};
export const logout = (req, res) => {try {
res.cookie("jwt","",{maxAge:0});
res.status(200).json({message: "Logged out successfully"});
} catch (error) {
console.log("Error in logout controller", error.message);
res.status(500).json({ message: "Internal Server error" });}}
export const updateProfile = async (req, res) => {try{
const {profilePic} = req.body;
const userId = req.user._id
if(!profilePic) {
return res.status(400).json({ message: "Profile picture is
required"});}
const uploadResponse = await cloudinary.uploader.upload(profilePic);
const updatedUser = await User.findByIdAndUpdate(userId, {
profilePic: uploadResponse.secure_url
}, { new: true });
res.status(200).json(updatedUser)} catch (error) {
console.log("Error in updateProfile controller", error.message);
res.status(500).json({ message: "Internal Server error" });}};
export const updateProfileInfo = async (req, res) => {
const { fullName, email } = req.body;
```

```
try {
const userId = req.user._id;
const updatedUser = await User.findByIdAndUpdate(userId, {
fullName, email}, { new: true });
res.status(200).json(updatedUser);
} catch (error) {
console.log("Error in updateProfileInfo controller", error.message);
res.status(500).json({ message: "Internal Server error" });}};
export const editingProfileBio = async (req, res) => {
const { bio } = req.body;
try {
const userId = req.user._id;
const updatedUser = await User.findByIdAndUpdate(userId, {
bio}, { new: true });
res.status(200).json(updatedUser);} catch (error) {
console.log("Error in editingProfileBio controller", error.message);
res.status(500).json({ message: "Internal Server error" });}};
export const checkAuth = (req, res) => {
try {
res.status(200).json(req.user);
} catch (error) {
console.log("Error in checkAuth controller", error.message);
res.status(500).json({ message: "Internal Server error" });
}}
```

## 2.auth.controller.js

```
import Message from '../models/message.model.js';
import User from '../models/user.model.js';
import cloudinary from '../lib/cloudinary.js';
import { io, getReceiverSocketId } from '../lib/socket.js';
export const getUsersForSidebar = async (req, res) => {
try {
const loggedInUserId = req.user._id;
const filteredUsers = await User.find({ _id: { $ne: loggedInUserId }
}).select("-password");
res.status(200).json(filteredUsers);
} catch (err) {
console.log("Error in getUsserForSidebar: ", err.message);
res.status(500).json({ message: "Internal server error" })}}
export const getMessages = async (req, res) => {
try {
const { id: userToChatId } = req.params;
const myId = req.user._id;
const messages = await Message.find({
$or: [
{ senderId: myId, receiverId: userToChatId },
{ senderId: userToChatId, receiverId: myId }]})
res.status(200).json(messages);
} catch (err) {
console.log("Error in getMessages controller: ", err.message);
res.status(500).json({ message: "Internal server error" })}}
export const sendMessage = async (req, res) => {
try {
const { text, image } = req.body;
const { id: receiverId } = req.params;
const senderId = req.user._id;
let imageUrl;
if(image) {
const uploadResponse = await cloudinary.uploader.upload(image);
imageUrl = uploadResponse.secure_url;}
const newMessage = new Message({
senderId,
receiverId,text,
image: imageUrl});
await newMessage.save();
const receiverSocketId = getReceiverSocketId(receiverId);
if(receiverSocketId) {
```

```
io.to(receiverSocketId).emit("newMessage", newMessage);}
res.status(201).json( newMessage );
} catch (err) {
console.log("Error in sendMessage controller: ", err.message);
res.status(500).json({ message: "Internal server error" })}}
export const deleteMessage = async (req, res) => {
try {
const message = await Message.findById(req.params.messageId);
if (!message) return res.status(404).send("Message not found");
const isSender = message.senderId.toString() === req.user.id;
const timeDiff = (Date.now() - new
Date(message.createdAt).getTime()) / 1000;
if (!isSender || timeDiff > 60) {
return res.status(403).send("Can't delete after 1 minute");}
const updateResult = await Message.updateMany({
_id: req.params.messageId,
$or: [{ senderId: req.user.id }, { receiverId: req.user.id }],},
{ $set: { text: null, image: null, isDeleted: true } });
if (updateResult.nModified === 0) {
return res.status(404).send("Message not found for the user");}
res.status(200).send("Message marked as deleted for both users");
} catch (error) {
console.error(error);
res.status(500).send("Server error");}};
```

## cloudinary.js

```
import { v2 as cloudinary } from "cloudinary";
import { config } from "dotenv";
config();
cloudinary.config({
cloud_name: process.env.CLOUDINARY_CLOUD_NAME,
api_key: process.env.CLOUDINARY_API_KEY,
api_secret: process.env.CLOUDINARY_API_SECRET});
export default cloudinary;
```

## db.js

```
import mongoose from "mongoose";
export const connectDB = async () => {
try{
const conn = await mongoose.connect(process.env.MONGODB_URI);
console.log(`MongoDB Connected: ${conn.connection.host}`);
} catch (error) {
console.log("MongoDB connection error:",error);
}};
```

## socket.js

```
import {Server} from 'socket.io';
import http from 'http';
import express from 'express';
const app = express();
const server = http.createServer(app);
const io = new Server(server, {
cors: {
origin: ['http://localhost:5173'],},
export function getReceiverSocketId(userId) {
return userSocketMap[userId];}
const userSocketMap = {};
io.on('connection', (socket) => {
console.log('New client connected', socket.id);
const userId = socket.handshake.query.userId;
if(userId) userSocketMap[userId] = socket.id;
io.emit('userConnected', Object.keys(userSocketMap));
socket.on('disconnect', () => {
console.log('Client disconnected', socket.id);
delete userSocketMap[userId];
io.emit('userDisconnected', Object.keys(userSocketMap));});});
export {io, app, server};
```

## utils.js

```
import jwt from 'jsonwebtoken';
export const generateToken = (userId, res) => {
const token = jwt.sign({ userId }, process.env.JWt_SECRET, {
        expiresIn: '7d'});
res.cookie('jwt', token, {
httpOnly: true, maxAge: 7 * 24 * 60 * 60 * 1000,
sameSite: "strict",
secure: process.env.NODE_ENV !== 'development'});return token;}
```

## MIDDLEWARES

### auth.middleware.js

```
import jwt from 'jsonwebtoken';
import User from '../models/user.model.js';
export const protectRoute = async (req, res, next) => {
try{
const token = req.cookies.jwt;
if(!token) {
return res.status(401).json({ message: "Unauthorized - No Token
Provided" });}
const decoded = jwt.verify(token, process.env.JWt_SECRET);
if(!decoded) {
return res.status(401).json({ message: "Unauthorized - Invalid
Token" });}
const user = await User.findById(decoded.userId).select("-
password");if(!user) {
return res.status(404).json({ message: "User not found" });}
req.user = user;
next();
} catch(error) {
console.log("Error in protectRoute middleware", error.message);
res.status(500).json({ message: "Internal Server error"});
}
}
```

## 3.DATABASE SECTION

### message.model.js

```
import mongoose from "mongoose";

const messageSchema = new mongoose.Schema({

senderId: {type: mongoose.Schema.Types.ObjectId,ref:
"User",required: true,},

receiverId: {type: mongoose.Schema.Types.ObjectId,ref:
"User",required: true,},

text: {type: String,},

isDeleted: {type: Boolean,default: false,},

image: {type: String,},}, {timestamps: true});

const Message = mongoose.model("Message", messageSchema);

export default Message;
```

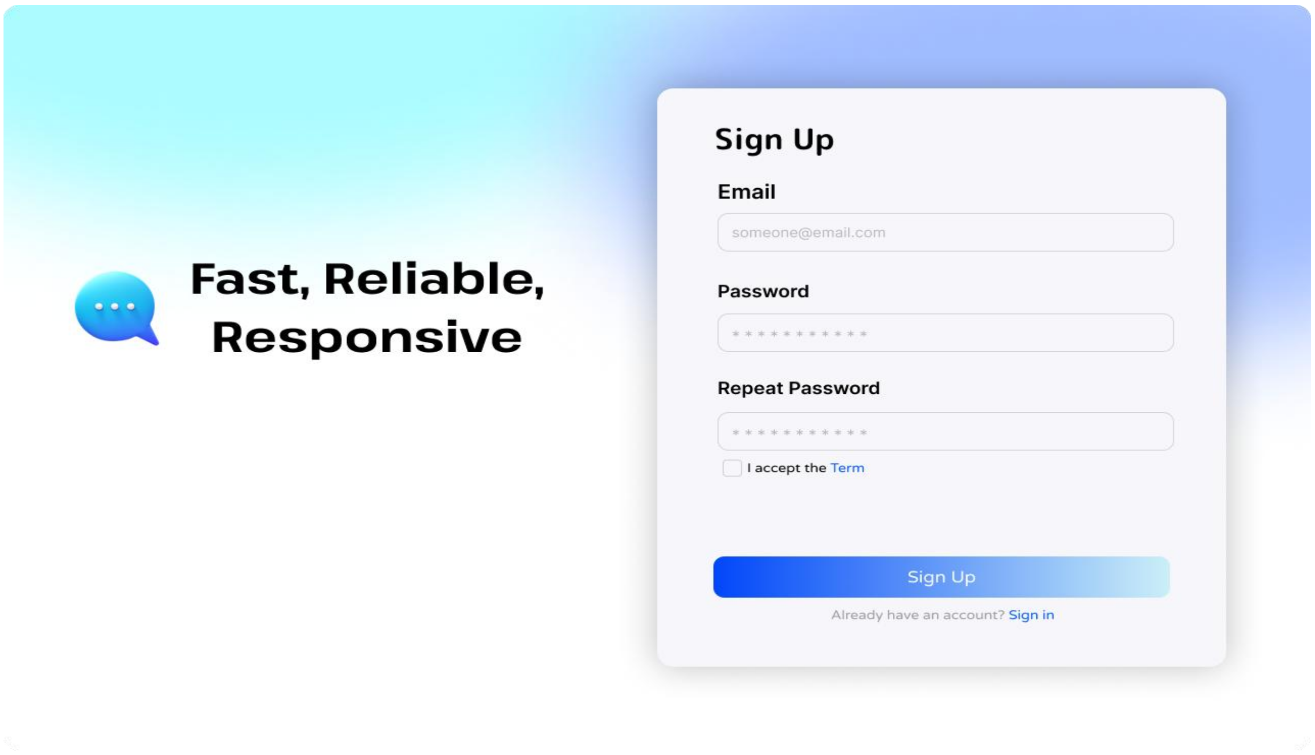
### user.model.js

```
import mongoose from "mongoose";

const userSchema = new mongoose.Schema({

email: {type: String,required: true,unique: true},

fullName: {type: String,required: true},

bio: {type: String,default: ""},

password: {type: String,required: true,minlength: 6,},

profilePic: {type: String,default: ""}}, {timestamps: true});

const User = mongoose.model("User", userSchema);

export default User;
```

# PROTOTYPE AND OUTPUT

## 8.1 OUTPUT

➢ **Sign up page:**



Fast, Reliable, Responsive

Sign Up

Email

someone@email.com

Password

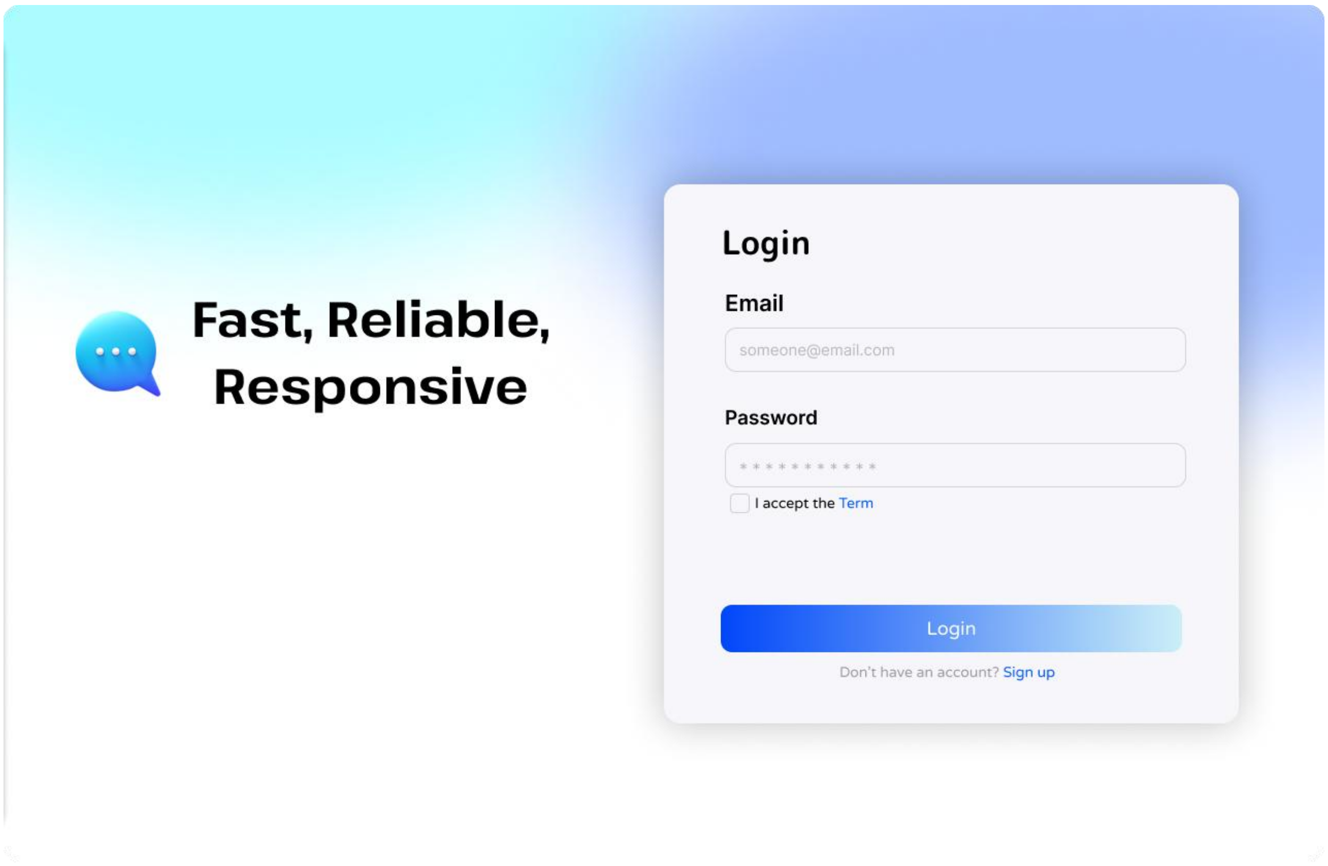
Repeat Password

I accept the Term

Sign Up

Already have an account? Sign in

➢ **Login page:**



Fast, Reliable, Responsive

Login

Email

someone@email.com

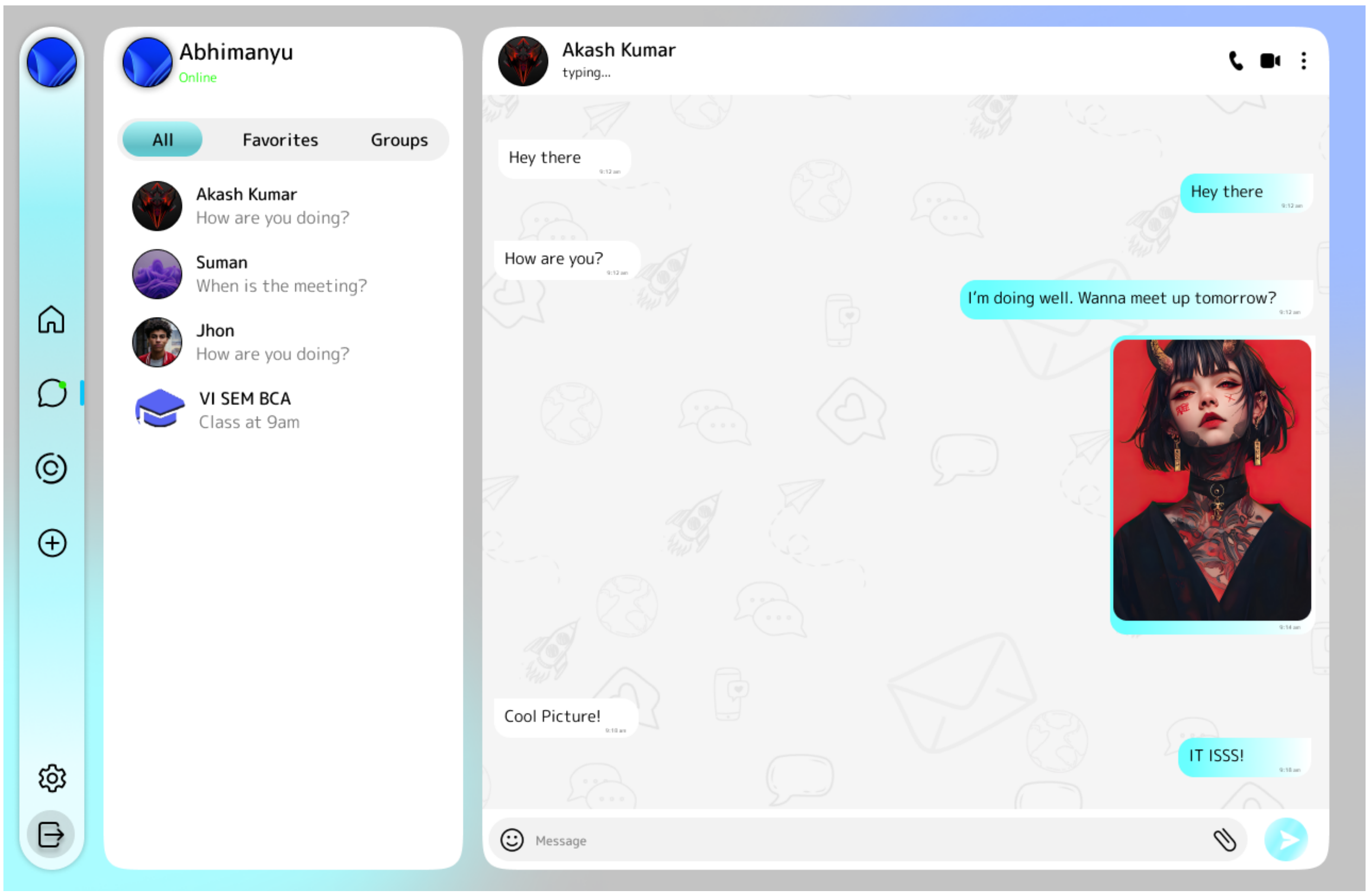
Password

I accept the Term

Login

Don't have an account? Sign up

## ➢ Chat page:



Hey there

Hey there

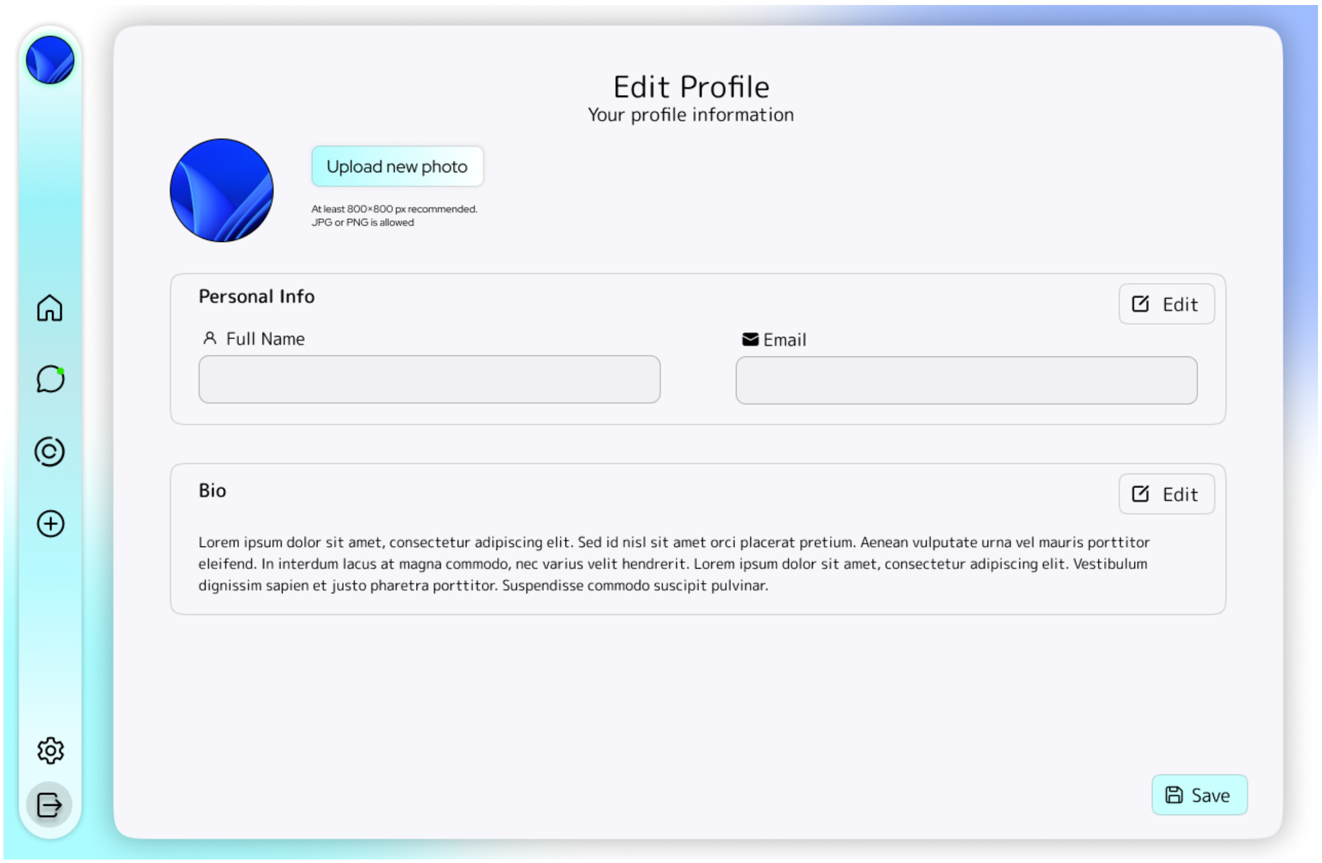
How are you?

I'm doing well. Wanna meet up tomorrow?

Cool Picture!

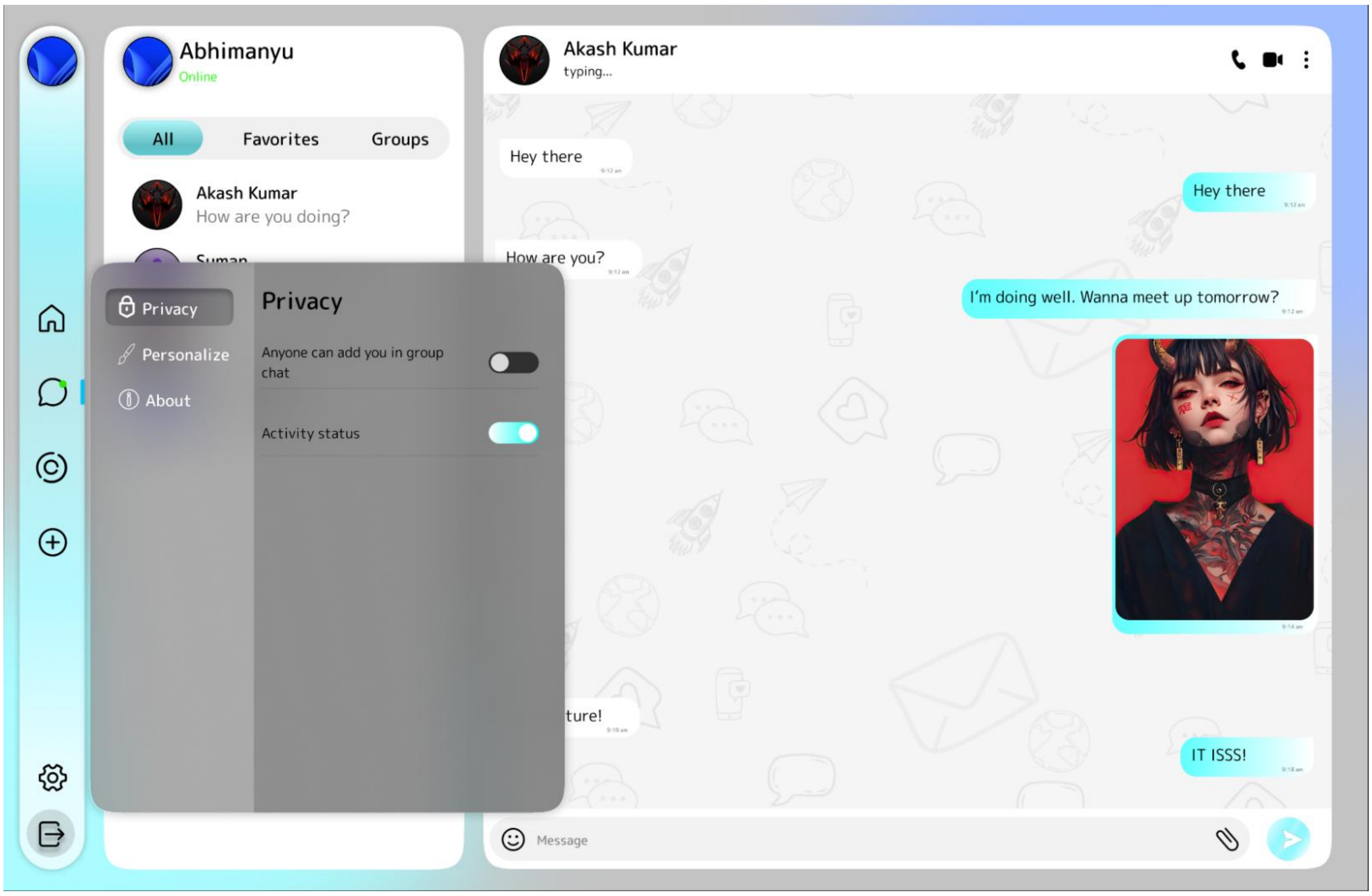
IT ISSS!

## ➢ Profile Page



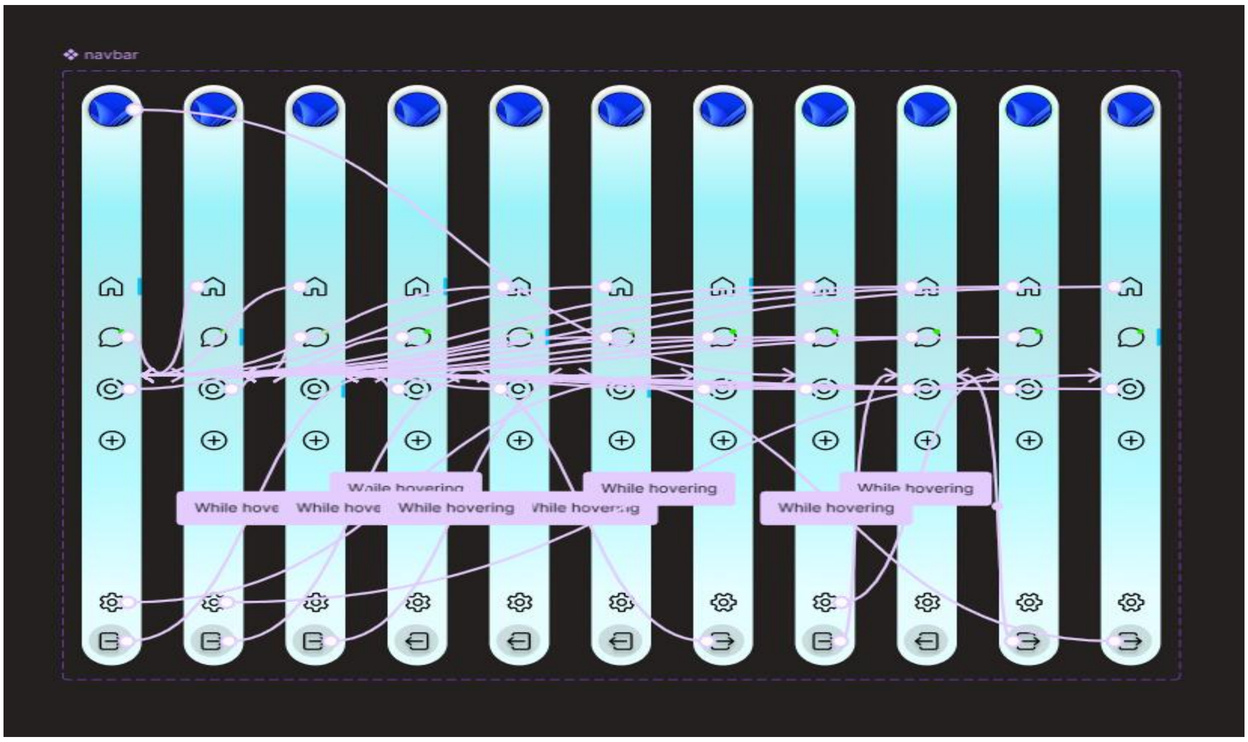
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed id nisl sit amet orci placerat pretium. Aenean vulputate urna vel mauris porttitor eleifend. In interdum lacus at magna commodo, nec varius velit hendrerit. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vestibulum dignissim sapien et justo pharetra porttitor. Suspendisse commodo suscipit pulvinar.

## ➢ Settings page:

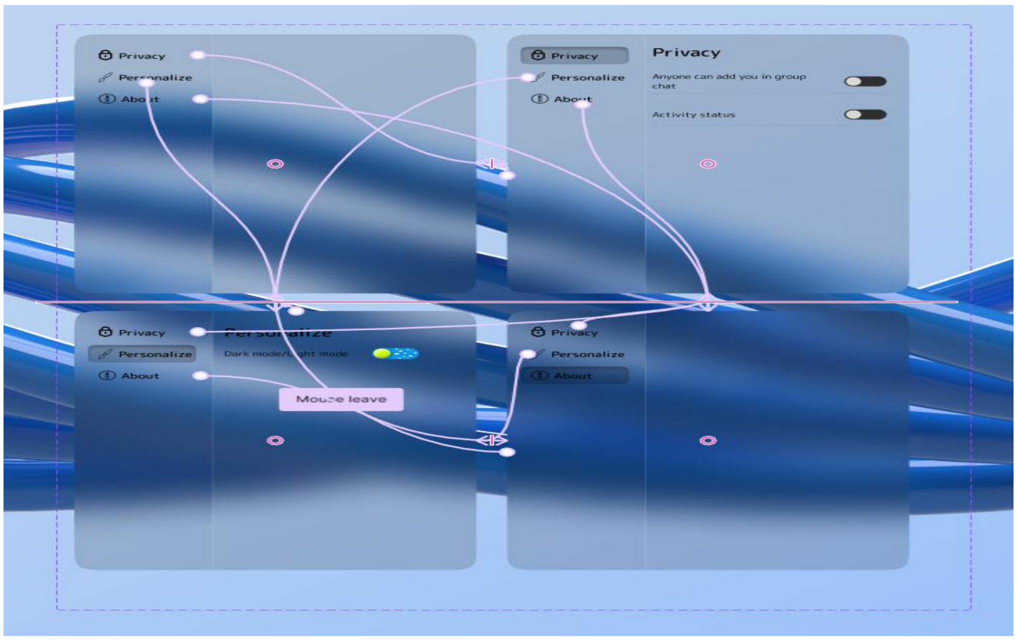

Abhimanyu
Online
All Favorites Groups
Akash Kumar
How are you doing?
Suman
Privacy
Personalize
About
Privacy
Anyone can add you in group chat
Activity status
Akash Kumar
typing...
Hey there
Hey there
How are you?
I'm doing well. Wanna meet up tomorrow?
ture!
IT ISSS!
Message

## 8.2 PROTOTYPE

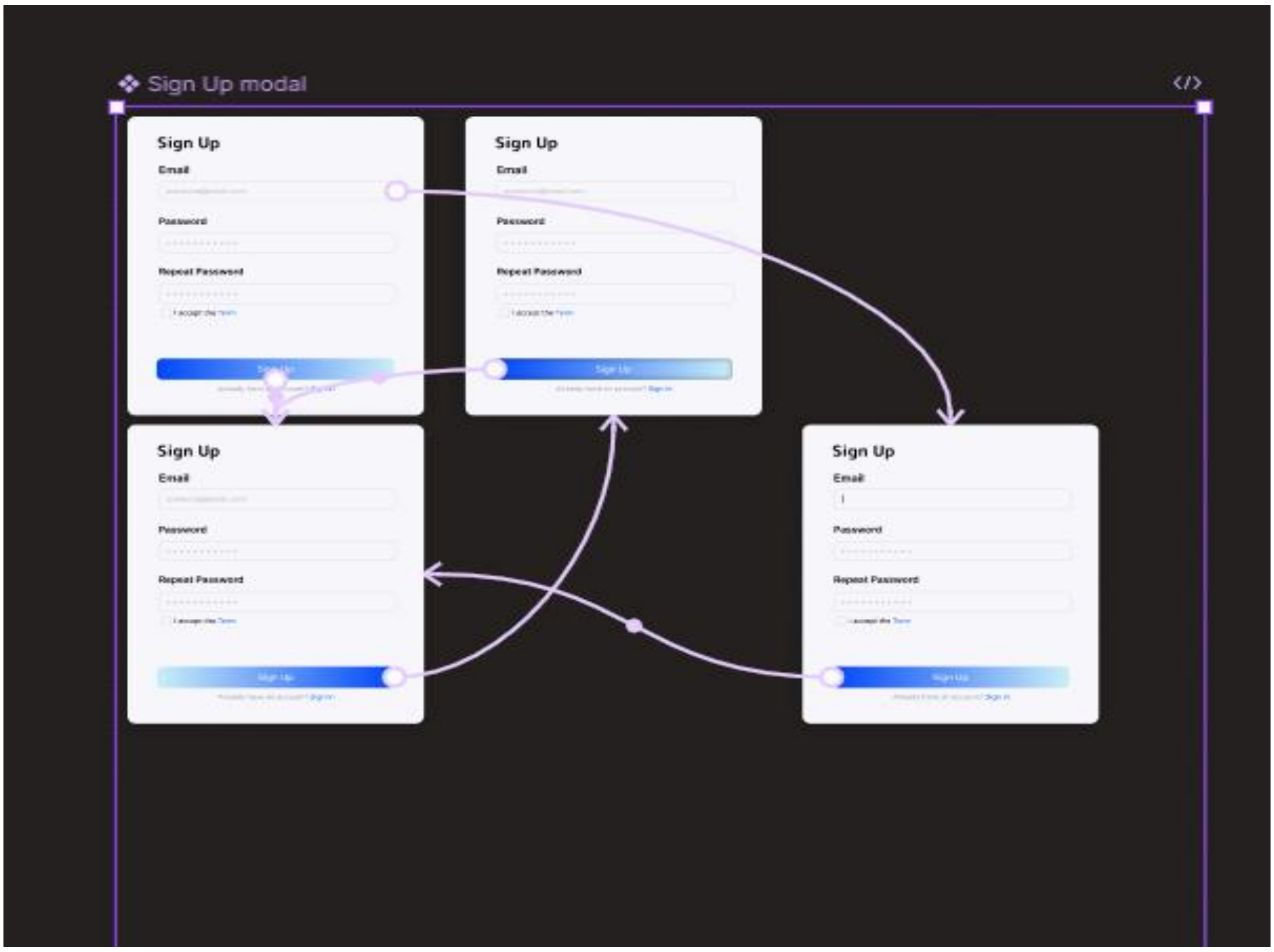
## ➢ Sidebar Prototype:



## ➢ Settings popup Prototype:



## ➢ Sign up modal Prototype:



# RESULT AND ANALYSIS

The Realtime Chat Web Application was tested across various modules to verify functionality, performance, and user experience. Below are the observations and findings:

## 1. Functional Testing:

- User Registration and Login: Successfully authenticated users using JWT and Bcrypt.
- Real-time Messaging: Socket.IO handled message delivery instantly without page refresh.
- Media Upload: Images and videos were uploaded and shared via Cloudinary without delay.
- Profile Management: Users were able to update their names, emails, and profile pictures.

## 2. Performance Analysis:

- Socket.IO provided a latency of less than 100ms under normal load conditions.
- Backend APIs using Node.js and Express responded under 200ms in 95% of test cases.
- MongoDB's NoSQL structure efficiently handled unstructured chat and media data.

## 3. Usability Evaluation:

- The user interface built with React and styled using Tailwind CSS offered a clean, responsive design.
- Navigation between login, chat, and profile settings was smooth and intuitive.
- Real-time notifications and image previews enhanced interactivity and engagement.

## 4. Limitations Identified:

- Group messaging and voice/video calls are not implemented in the current version.
- Offline support and push notifications are out of scope but are planned for future development.

In conclusion, the application met its objectives in delivering a functional, responsive, and real-time communication platform. It forms a solid basis for future iterations and real-world deployment.

# CHALLENGES

During the development of the live web chat application, several challenges were encountered:

## <u>Real-Time Communication:</u>

Implementing real-time chat using WebSockets required proper synchronization between client and server. Issues such as message duplication and delayed delivery initially arose during testing.

## <u>User Authentication:</u>

Ensuring secure authentication and storing hashed passwords involved learning and correctly implementing libraries such as bcrypt. Proper session handling and token management were also tricky initially.

## <u>File Uploads and Storage:</u>

Allowing users to upload images introduced challenges related to file size limits, type validation, and storage location. Ensuring uploads didn't pose security risks required extra precautions.

## <u>Online/Offline Status Tracking:</u>

Maintaining accurate online status in real time required frequent state updates and careful connection tracking to avoid showing incorrect user statuses.

## <u>UI Consistency Across Devices:</u>

Making the UI responsive and consistent across mobile, tablet, and desktop browsers took time. Layout issues and misalignments were found and resolved through extensive CSS testing.

# CONCLUSION

The Realtime Chat Web Application successfully demonstrates the potential of modern web technologies to create scalable, responsive, and secure communication platforms. By leveraging the MERN stack in combination with Socket.IO, the application delivers seamless real-time interactions between users. Additionally, the integration of Cloudinary enhances the user experience by enabling media sharing, a crucial feature in contemporary messaging systems.

Throughout the project, the team navigated various challenges such as socket synchronization, secure authentication using JWT and Bcrypt, and dynamic UI rendering using React. These challenges were addressed using best practices in software engineering and full-stack development.

This project not only serves as a robust prototype for real-world applications but also offers a strong foundation for future enhancements such as group chats, voice/video calling, and analytics. The experience gained through this development process has equipped the team with critical skills in full-stack development, DevOps, and secure application architecture, making it a significant academic and practical achievement.

# BIBLIOGRAPHY

1. MongoDB Documentation. (n.d.). Retrieved from: https://www.mongodb.com/docs/
2. Express.js Guide. (n.d.). Retrieved from: https://expressjs.com/
3. React.js Documentation. (n.d.). Retrieved from: https://reactjs.org/docs/getting-started.html
4. Node.js Documentation. (n.d.). Retrieved from: https://nodejs.org/en/docs
5. Socket.IO Documentation. (n.d.). Retrieved from: https://socket.io/docs/
6. Cloudinary Developer Documentation. (n.d.). Retrieved from: https://cloudinary.com/documentation
7. JWT.io – JSON Web Tokens. (n.d.). Retrieved from: https://jwt.io/introduction
8. Bcrypt GitHub Repository. (n.d.). Retrieved from: https://github.com/kelektiv/node.bcrypt.js
9. Tailwind CSS Documentation. (n.d.). Retrieved from: https://tailwindcss.com/docs

# APPENDIX

## Appendix A: Architecture Diagram

A layered architecture showcasing client-server interaction using React, Node, and MongoDB.
Real-time updates via Socket.IO.

## Appendix B: Data Flow Diagrams (DFD)

Level 0: User ↔ Chat System ↔ Database
Level 1: Auth, Message Handling, Media Upload
Level 2: Token Authentication, Cloudinary API Flow, Socket Events

## Appendix C: ER Diagram

Users: id, name, email, password, profilePic
Messages: id, senderId, receiverId, text, mediaURL, timestamp
Chats: id, userIds[], lastMessageId

## Appendix D: Tools & Technologies

Frontend: React.js, Tailwind CSS
Backend: Node.js, Express.js
Database: MongoDB (Atlas)
Real-time: Socket.IO
Media Storage: Cloudinary
Version Control: GitHub
Dev Tools: VSCode, Postman

Design Tools: Figma, Adobe XD

## Appendix E: Screenshots

Login Page
Signup Page
Chat Interface
Profile Editing
Media Sharing UI