Chat Messenger App – "Chatty"

By

Aradhya Kumar Mishra (23BIT0261)

Harsh Singh (23BIT242)

Aditya Subrahmanya Sri Lanka (23BIT0024)

Under the supervision of

Sh. (Dr.) Balasubramani M

School of Computer Science Engineering

and

Information Systems - (SCORE)

Course Project Report

Document

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

<u>**DECLARATION**</u>

I, *Aradhya Kumar Mishra (23BIT0261)* hereby declare that the BITE304L - entitled *"Chatty Messenger App* " submitted by me, for the award of the degree of Bachelor of Information Technology in Department of Information Technology, School of Computer Science Engineering and Information Systems to VIT is a record of Bonafide work carried out by me under the supervision of *Sh. (Dr.) Balasubramani S.*

I further declare that the work reported in this project has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Place: Vellore, India

Date: 4th April 2025

Aradhya

Signature of the Candidate

<u>CERTIFICATE</u>

This is to certify that the BITE304L- Course Project entitled "*Chatty Messenger App*" submitted by Student Name & Reg. No, SCORE, VIT, for the award of the degree of Bachelor of Information Technology in the Department of Information Technology provision during the period, 13. 12. 2024 to 17.04.2025, as per the VIT code of academic and research ethics.

The contents of this report have not been submitted and will not be submitted either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university. The Capstone project fulfills the requirements and regulations of the University and in my opinion meets the necessary standards for submission.

Place: Vellore

Date: 4th April 2025

Signature of the VIT-SCORE - Guide

Outline of the report

1. **Project Title**
    - "Chatty" - A Real-Time Chat Messenger Application
    - Built using MERN Stack (MongoDB, Express.js, React.js, Node.js) and Socket.io
2. **Abstract**
    - Overview of "Chatty" and its purpose.
    - Importance of real-time communication in modern applications.
    - Technologies used and key features.
    - Expected impact and significance.
3. **Problem Statement**
    - Issues with traditional messaging systems.
    - Challenges of outdated polling mechanisms and security concerns.
    - How "Chatty" addresses these problems.
4. **Methodology**
    - Requirement Analysis
    - System Design
    - Technology Selection
    - Implementation
    - Testing and Deployement
5. **Architecture**
    - **Frontend:** React.js for UI, Redux for state management.
    - **Backend:** Node.js and Express.js for API handling.
    - **Database:** MongoDB and Mongoose ORM.
    - **WebSockets:** Real-time messaging with Socket.io.
    - **Authentication & Security:** JWT authentication and bcrypt encryption.
    - **Scalability & Performance Optimization.**

Abstract

"Chatty" is a real-time chat messenger application designed to provide seamless and efficient communication between users. Built using the MERN (MongoDB, Express.js, React.js, Node.js) stack, the application integrates modern web technologies to deliver an intuitive and scalable messaging experience. A key feature of "Chatty" is its real-time, bidirectional communication, achieved through the integration of Socket.io. This ensures instant message delivery while minimizing server load, making the application responsive and efficient.

The application supports both private and group chats, allowing users to engage in one-on-one conversations or participate in discussions with multiple users. To ensure security and protect user data, "Chatty" employs JWT (JSON Web Token)-based authentication. This authentication method helps prevent unauthorized access and secures user credentials, ensuring a safe messaging environment. Additionally, the system is designed to provide an intuitive user interface (UI) that enhances the overall experience by offering a clean, user-friendly layout with easy navigation and responsive design.

At the core of "Chatty" is WebSocket technology, which enables persistent connections between the client and the server, ensuring real-time communication without the need for continuous polling. This approach reduces server load and enhances performance, making the system more scalable and efficient. Unlike traditional HTTP requests that require frequent communication with the server, WebSockets provide a more optimized solution for instant messaging applications.

One of the main objectives of "Chatty" is to create a robust and scalable platform that meets the needs of modern communication. Whether for personal conversations or professional discussions, the application offers a smooth and engaging chat experience. Scalability is a key design principle, allowing the platform to accommodate future enhancements without significant structural changes.

In the future, "Chatty" aims to introduce several advanced features to improve its functionality. One such enhancement is multimedia sharing, which will allow users to exchange images, videos, documents, and other media files directly within the chat interface. Another planned feature is the integration of AI-driven chatbots that can assist

users with automated responses, customer support, and personalized recommendations. Additionally, to further enhance security and privacy, "Chatty" intends to implement end-to-end encryption, ensuring that messages remain confidential and can only be accessed by the intended recipients.

By leveraging the power of modern web technologies, "Chatty" ensures a fast, secure, and dynamic messaging experience. The application is built to handle real-time interactions efficiently, making it suitable for diverse use cases, from casual conversations to professional collaborations. The emphasis on performance, security, and scalability positions "Chatty" as a reliable communication platform capable of evolving with technological advancements and user needs.

In conclusion, "Chatty" represents a next-generation messaging solution that combines real-time capabilities with a user-friendly design. Through its robust architecture, security measures, and future scalability plans, the application is well-equipped to meet the communication demands of the modern digital era. Whether for individual users or teams, "Chatty" is designed to facilitate seamless interactions, making digital communication more efficient and enjoyable.

Problem Statement

Traditional messaging platforms face several challenges that hinder their effectiveness in providing a seamless communication experience. Issues such as latency, security vulnerabilities, and inefficient data handling significantly impact user satisfaction and system performance. Many existing chat applications rely on outdated polling mechanisms, which require frequent client-server interactions to check for new messages. This approach results in delayed message delivery, increased server load, and poor scalability.

Latency remains one of the most significant issues in traditional messaging systems. Many applications still use polling or long polling techniques, which introduce unnecessary delays in message transmission. As a result, users experience lags in real-time conversations, making interactions less engaging and responsive. This inefficiency is particularly problematic for applications that require instant communication, such as professional collaborations, customer support, and live events.

Security vulnerabilities are another major concern for messaging applications. Unauthorized access, data breaches, and weak authentication mechanisms expose users to privacy risks. Many traditional chat platforms lack robust security measures, making them susceptible to cyber-attacks and data theft. Without proper encryption and authentication protocols, user data can be intercepted, altered, or accessed by malicious actors. This compromises trust in the platform and discourages users from engaging in meaningful conversations.

To combat these challenges, "Chatty" incorporates WebSockets for real-time, low-latency communication. Unlike traditional polling methods, WebSockets establish a persistent connection between the client and the server, allowing instant, bidirectional data exchange. This drastically reduces latency and ensures seamless message delivery without overloading the server. The result is a highly responsive and efficient messaging system that enhances user experience.

Security is a top priority for "Chatty." To protect user data and prevent unauthorized access, the application implements JWT (JSON Web Token)-based authentication. JWT ensures secure user authentication by verifying and authorizing each request, minimizing the risk of account takeovers. Additionally, "Chatty" employs bcrypt password hashing, which enhances security by encrypting stored user credentials, making them resistant to brute-force attacks and data leaks.

Scalability is another crucial challenge in chat applications. As user numbers grow, many traditional platforms struggle with performance degradation, slow message delivery, and system crashes. "Chatty" addresses this problem by utilizing an optimized backend architecture and efficient database management with MongoDB. MongoDB's flexible schema and indexing capabilities allow seamless scaling, ensuring that the system remains stable and efficient even as the user base expands.

By leveraging the MERN stack and integrating Socket.io, "Chatty" overcomes the limitations of traditional messaging platforms. It provides a fast, secure, and scalable alternative that ensures real-time interactions without compromising performance. Future enhancements, such as multimedia sharing, AI-powered chatbots, and end-to-end encryption, will further strengthen the platform's capabilities, making it an ideal solution for modern communication needs.

In conclusion, "Chatty" is designed to address the inefficiencies of existing messaging applications by offering a real-time, secure, and scalable solution. By tackling key concerns such as latency, security, and scalability, "Chatty" emerges as a robust and reliable chat platform suitable for both personal and professional use.

Methodology

The development of "Chatty" follows a structured and iterative methodology that ensures a robust, scalable, and user-friendly chat application. The methodology consists of multiple phases, including requirement analysis, system design, technology selection, implementation, testing, and deployment. By following a systematic approach, "Chatty" achieves high performance, security, and real-time communication efficiency.

1. Requirement Analysis: This phase involves identifying the key features and functionalities required for "Chatty." Research is conducted on existing chat applications to understand their limitations and identify areas for improvement. User expectations, security requirements, and scalability factors are documented to create a well-defined development roadmap.

2. System Design: The architecture of "Chatty" is planned using a microservices-based approach, ensuring modularity and scalability. The application is divided into frontend, backend, database, and real-time communication layers. The WebSocket protocol is integrated to enable real-time, bidirectional messaging, while JWT-based authentication and bcrypt password hashing are incorporated to enhance security.

3. Technology Selection: "Chatty" is developed using the MERN stack, leveraging MongoDB for database management, Express.js and Node.js for backend logic, and React.js for an interactive user interface. Socket.io is used for real-time communication, ensuring instant message delivery. Additional libraries and frameworks are chosen to optimize performance and enhance the user experience.

4. Implementation: The development follows an agile methodology, with continuous iteration and feedback loops. Features such as user authentication, private and group chats, and real-time notifications are implemented in phases, ensuring smooth integration and functionality validation.

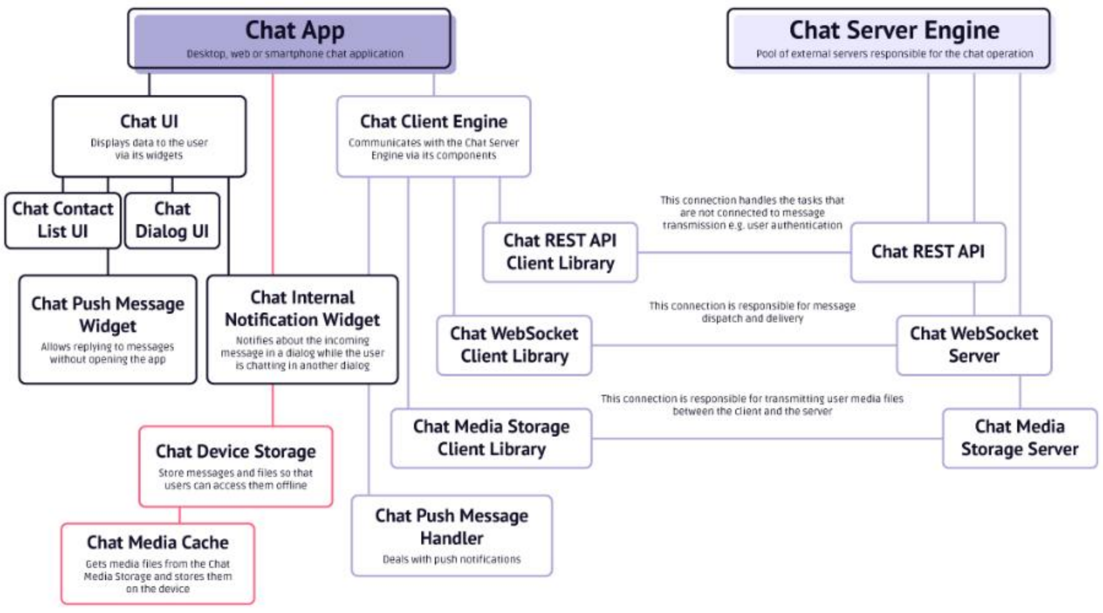
5. Testing and Deployment: Unit testing, integration testing, and security audits are performed to ensure reliability. The application is deployed on cloud servers, allowing scalability and high availability.

This structured methodology ensures that "Chatty" is a secure, scalable, and real-time messaging solution.

<u>Architecture</u>

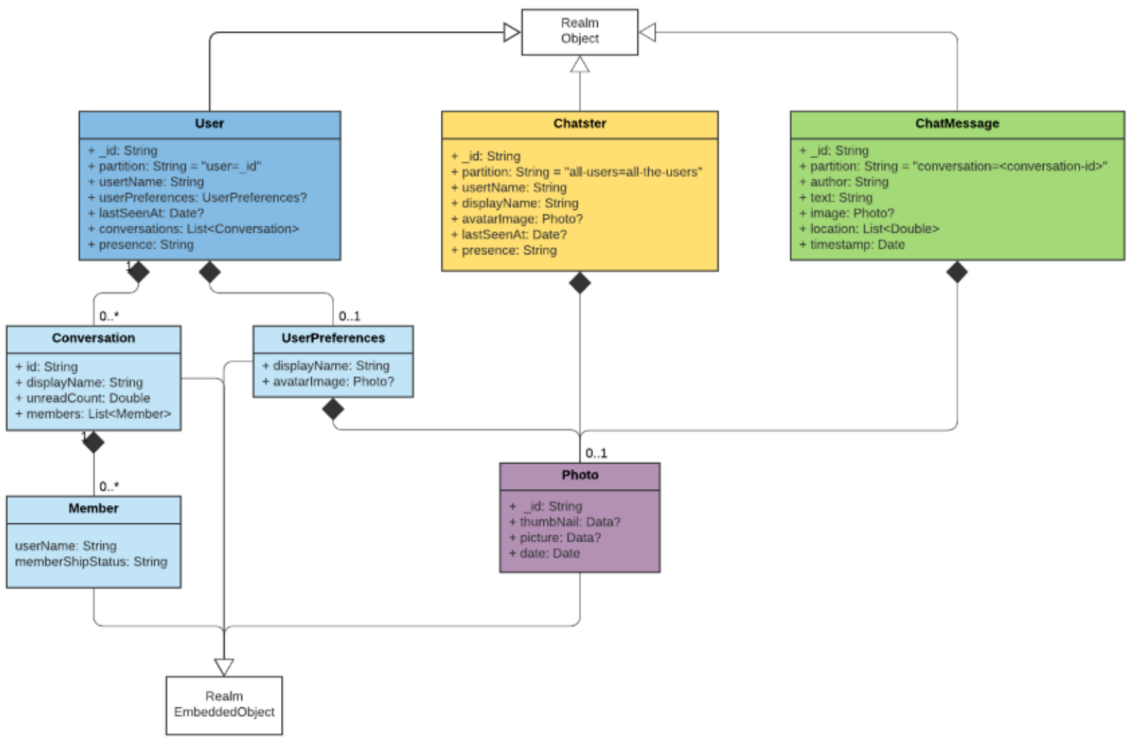
"Chatty" follows a modular and scalable architecture:

- **Frontend:** Developed with React.js and styled with CSS for an intuitive user experience.
- **Backend:** Powered by Node.js and Express.js, managing API requests, authentication, and database interactions.
- **Database:** Uses MongoDB for storing user data and chat history, with Mongoose ORM for efficient querying.
- **WebSockets:** Implements Socket.io for real-time, event-driven communication between users.
- **Authentication & Security:** Secure login via JWT tokens, and role-based access control.
- **Scalability:** Designed for performance with load balancing, optimized queries, and efficient event handling, making it adaptable for future enhancements.



Chat App
Desktop, web or smartphone chat application
Chat Server Engine
Pool of external servers responsible for the chat operation
Chat UI
Displays data to the user via its widgets
Chat Client Engine
Communicates with the Chat Server Engine via its components
Chat Contact List UI
Chat Dialog UI
Chat REST API Client Library
This connection handles the tasks that are not connected to message transmission e.g. user authentication
Chat REST API
Chat Push Message Widget
Allows replying to messages without opening the app
Chat Internal Notification Widget
Notifies about the incoming message in a dialog while the user is chatting in another dialog
Chat WebSocket Client Library
This connection is responsible for message dispatch and delivery
Chat WebSocket Server
This connection is responsible for transmitting user media files between the client and the server
Chat Media Storage Client Library
Chat Media Storage Server
Chat Device Storage
Store messages and files so that users can access them offline
Chat Media Cache
Gets media files from the Chat Media Storage and stores them on the device
Chat Push Message Handler
Deals with push notifications

^ ***above diagram represents the components architecture for our app.***



<u>^ class diagram for our chat app</u>

<u>Installation Procedure and Deployment on GitHub</u>

- **Cloning the Repository:** git clone <repo url>
- **Installing Dependencies:** Navigate to the backend and frontend directories and run npm install.
- **Environment Setup:** Create a .env file and define variables like database URI, JWT secret, and API keys.
- **Running the Application:**
- **Installing Necessary libraries:**

  Mongo DB: npm install mongodb

  Mongoose: npm install mongoose

  React: npm/x create-react-app frontend

  Json-Web Token: npm jsonwebtoken

  Socket.io : npm install socket.io

  - Start the backend: npm run dev
  - Start the frontend: npm start
- **Deployment (only Fronend):**
  - Frontend: Deployed on github.
- **Pushing to GitHub:**
  - Stage changes: git add .
  - Commit changes: git commit -m "Initial commit"
  - Push to GitHub: git push origin main

Conclusion

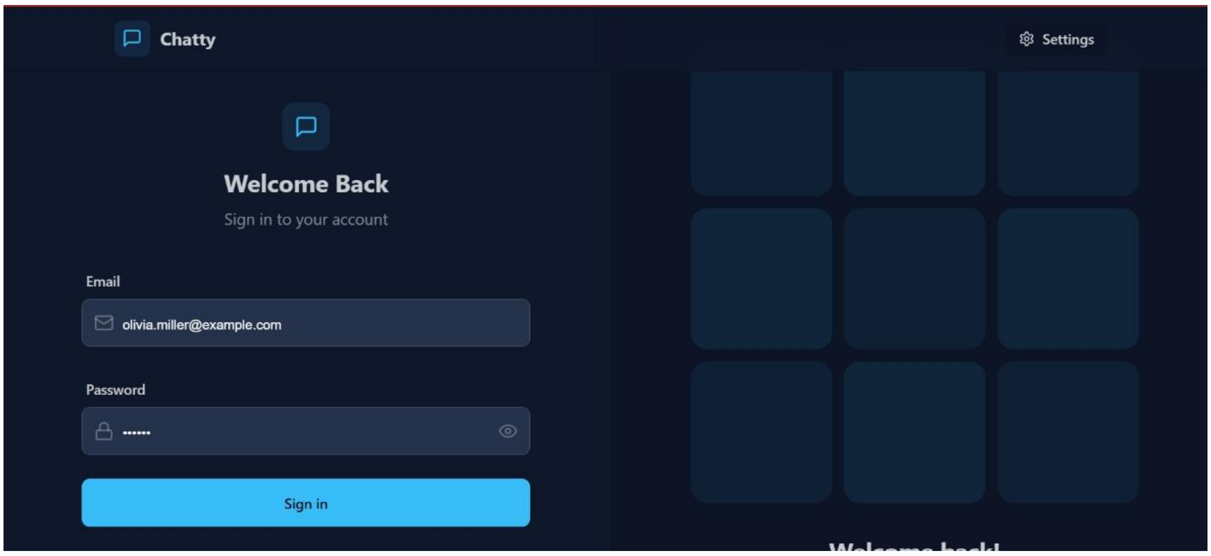
"Chatty" successfully demonstrates the power of the MERN stack combined with WebSockets for real-time messaging. By incorporating a well-structured architecture, robust security mechanisms, and an intuitive user experience, the application provides seamless and efficient communication.

The use of WebSockets ensures instant message delivery, while MongoDB's scalability allows the platform to handle a growing user base. The application is designed with future enhancements in mind, such as voice and video chat, AI-powered assistance, and additional encryption techniques.

As an open-source project, "Chatty" can be extended and improved by the developer community, making it a valuable contribution to the field of real-time web applications.

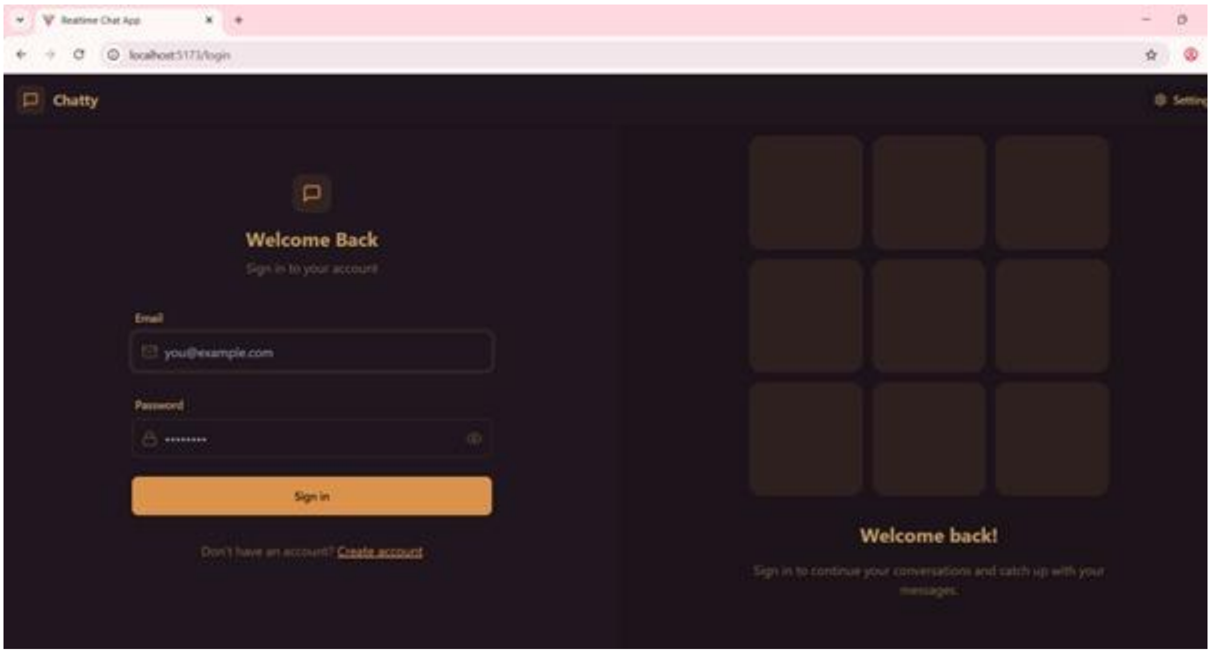
This project highlights the importance of scalable, real-time communication systems and serves as a strong foundation for future developments in the field of messaging applications.

<u>Output of the Project:</u>



Email: olivia.miller@example.com

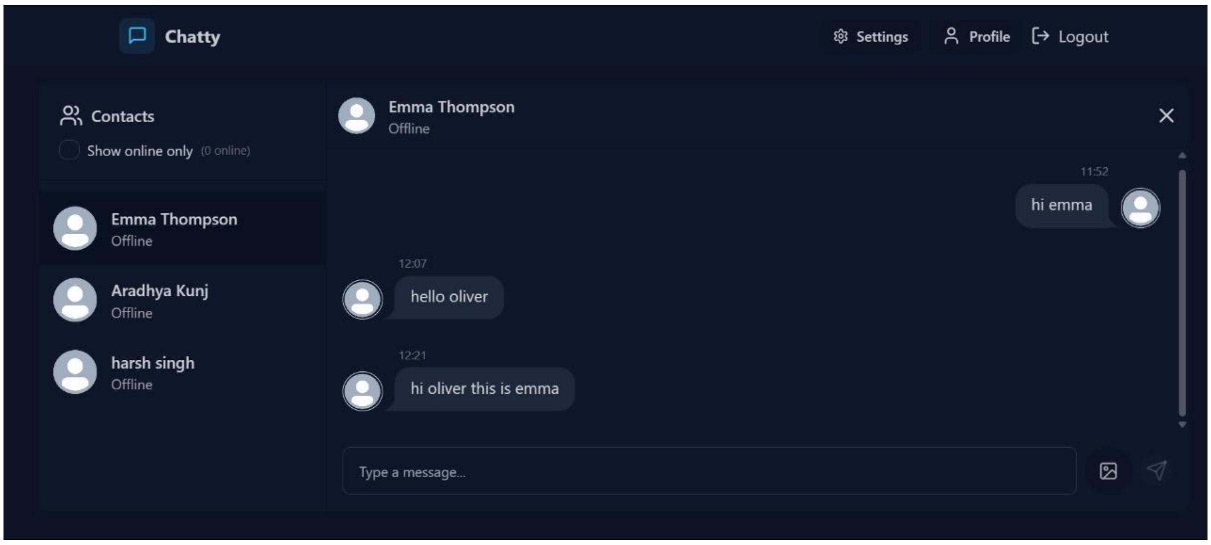
Password: ••••••



Email: you@example.com

Password: •••••••••

Sign in to continue your conversations and catch up with your messages.



Emma Thompson
Offline

Aradhya Kunj
Offline

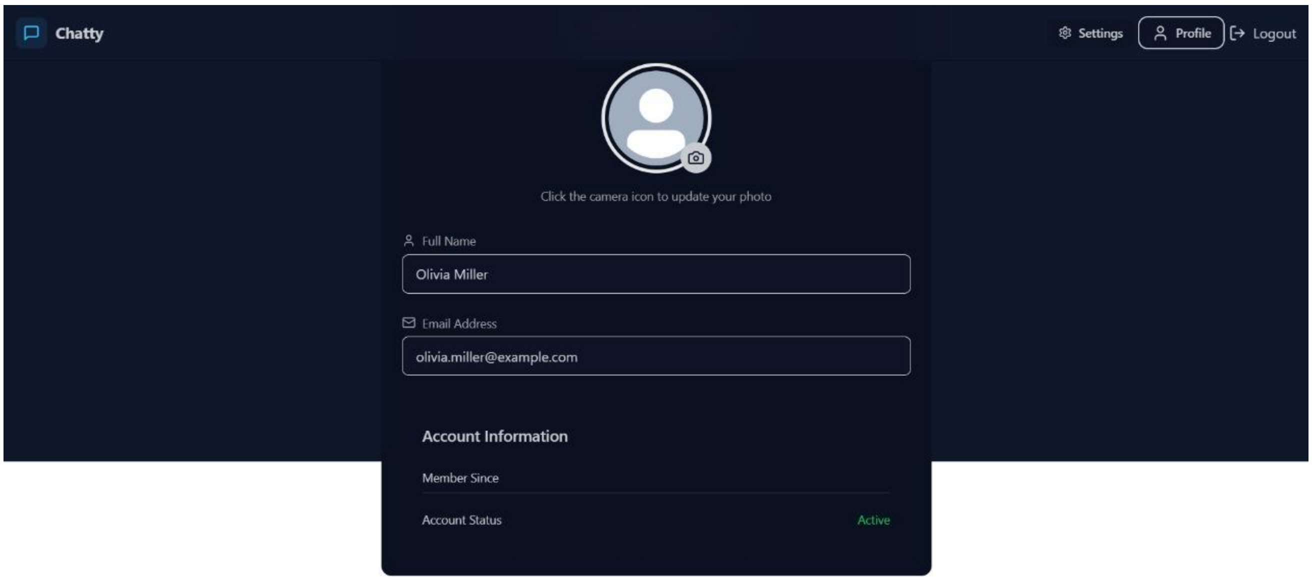
harsh singh
Offline

Emma Thompson
Offline

11:52 hi emma

12:07 hello oliver

12:21 hi oliver this is emma



Full Name

Olivia Miller

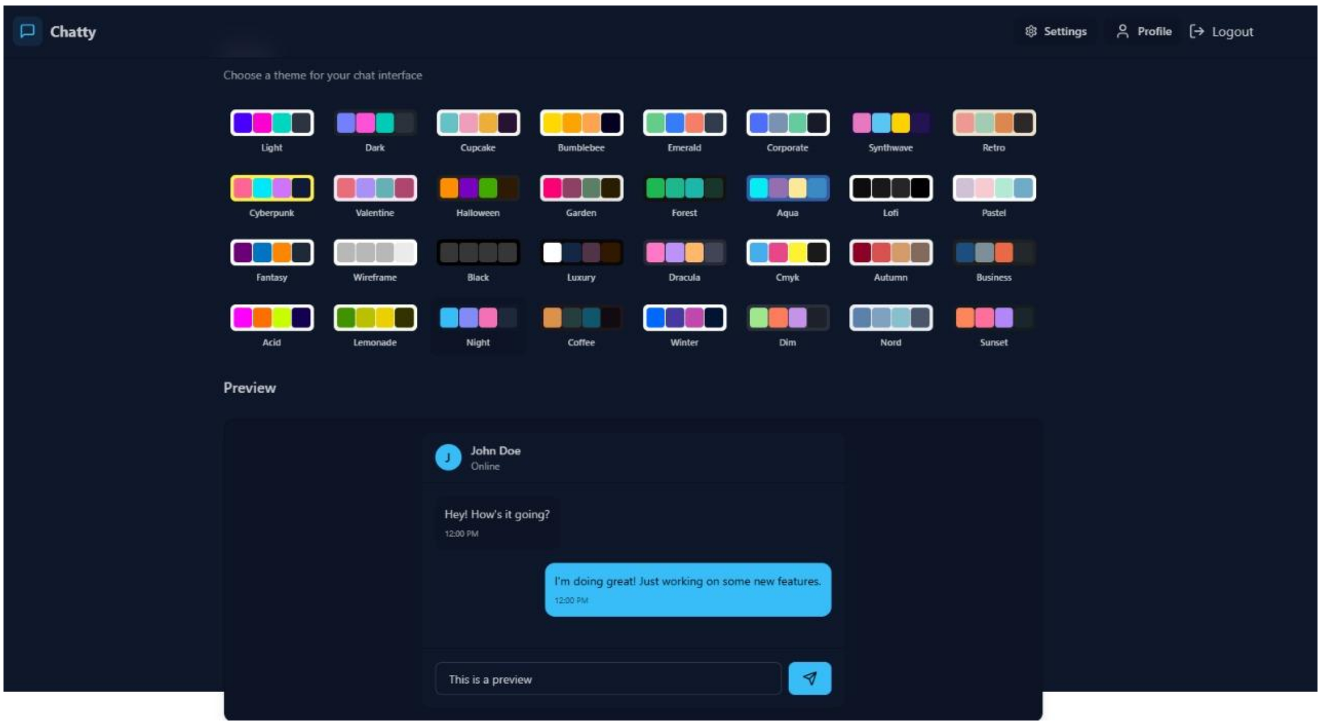
Email Address

olivia.miller@example.com

Account Information

Member Since

Account Status Active

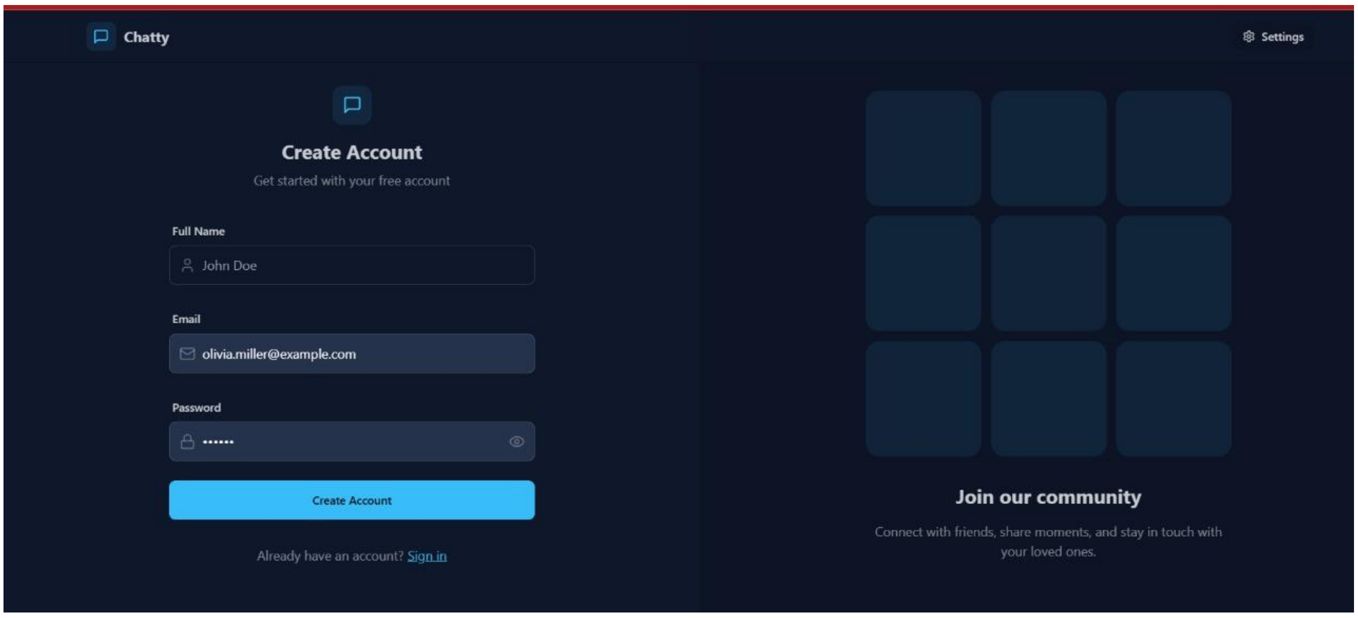


John Doe
Online

Hey! How's it going?
12:00 PM

I'm doing great! Just working on some new features.
12:00 PM

This is a preview



Create Account

Get started with your free account

Full Name: John Doe

Email: olivia.miller@example.com

Password: ••••••

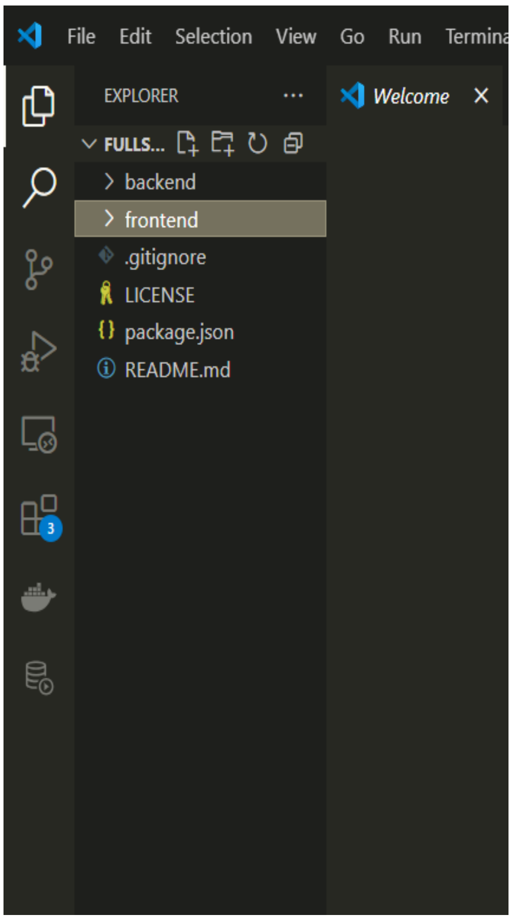
Already have an account? Sign in

Join our community

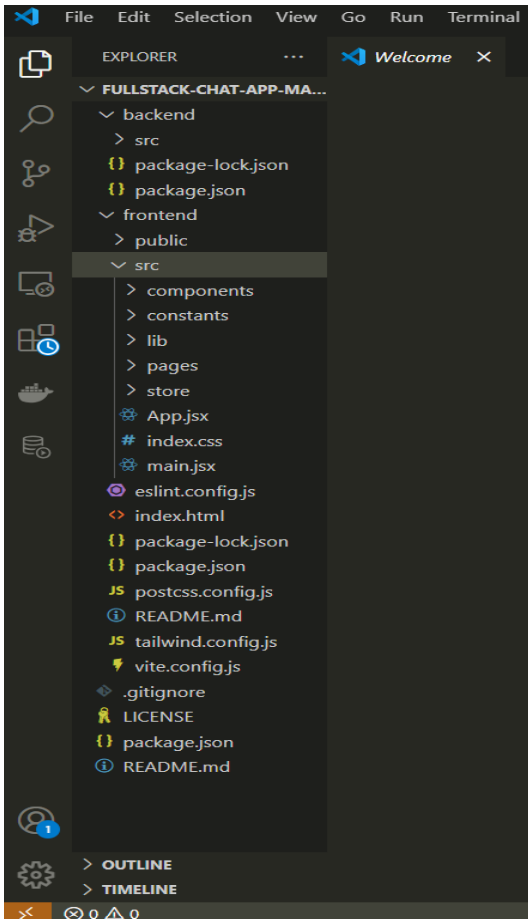
Connect with friends, share moments, and stay in touch with your loved ones.

Screenshots of Source Code for the project is attached herewith:

Directory Structure:



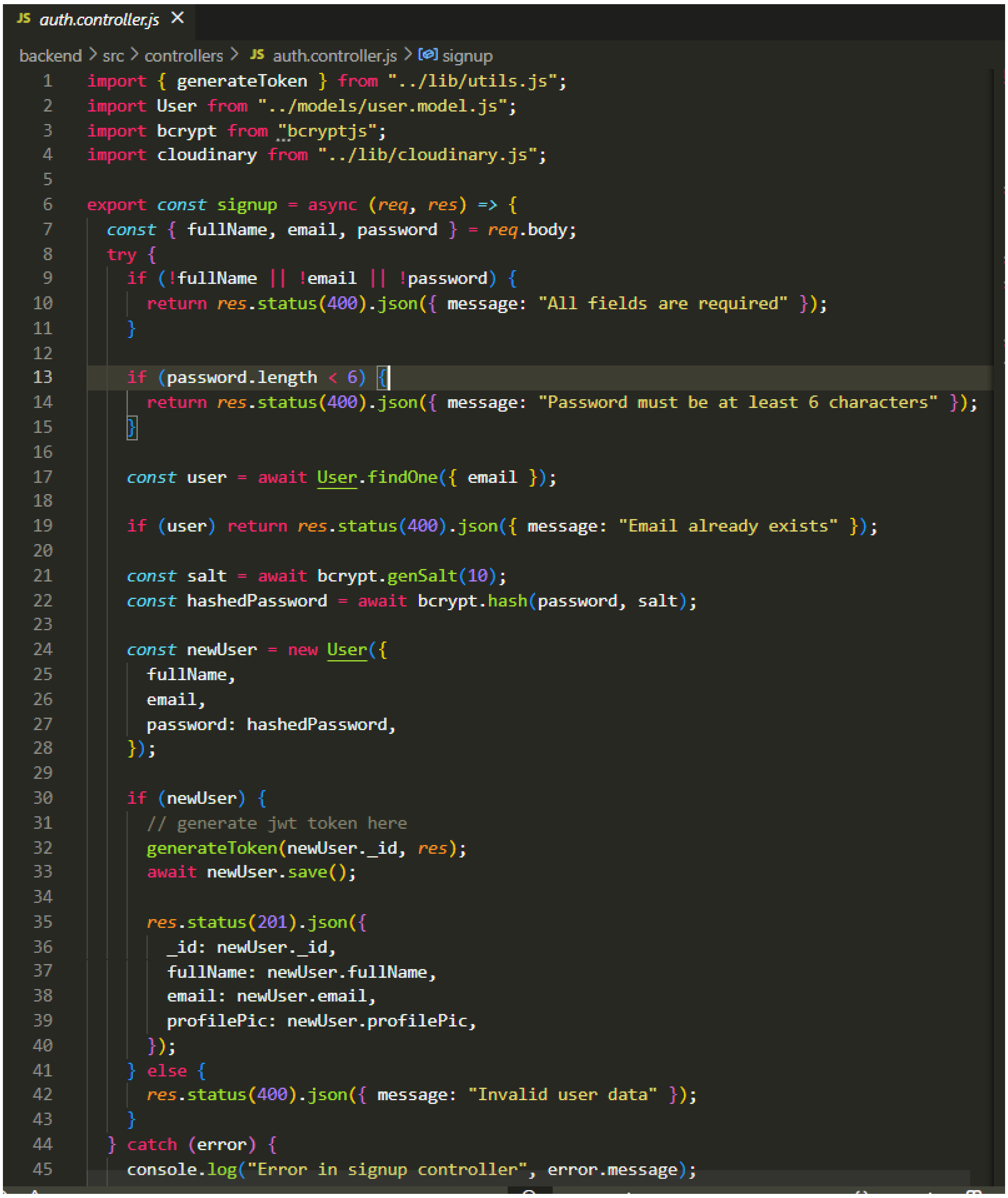
```
FULLS...
> backend
> frontend
  .gitignore
  LICENSE
  package.json
  README.md
```



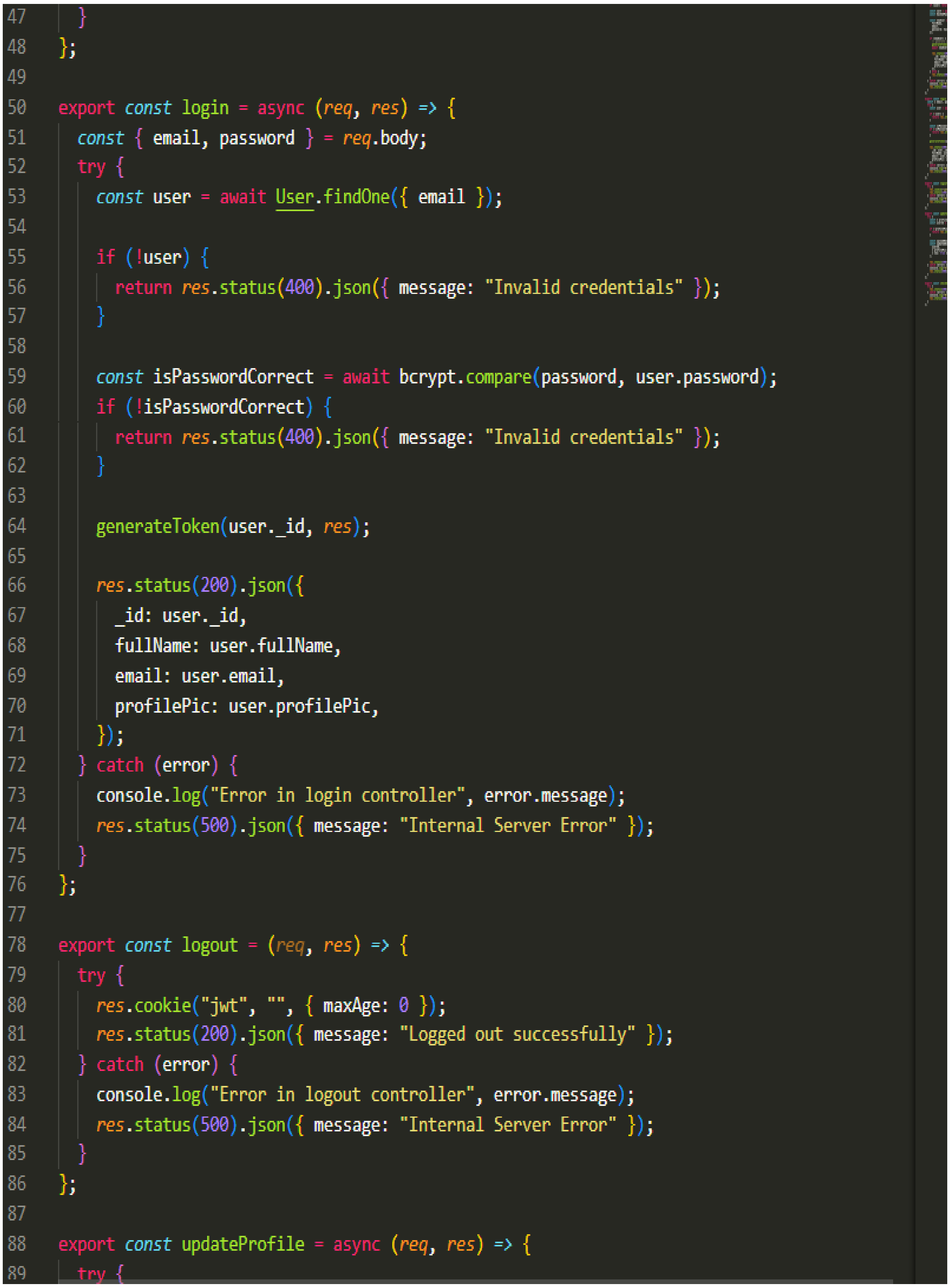
```
FULLSTACK-CHAT-APP-MA...
  backend
    > src
    package-lock.json
    package.json
  frontend
    > public
    src
      > components
      > constants
      > lib
      > pages
      > store
      App.jsx
      index.css
      main.jsx
    eslint.config.js
    index.html
    package-lock.json
    package.json
    postcss.config.js
    README.md
    tailwind.config.js
    vite.config.js
  .gitignore
  LICENSE
  package.json
  README.md
```

Backend

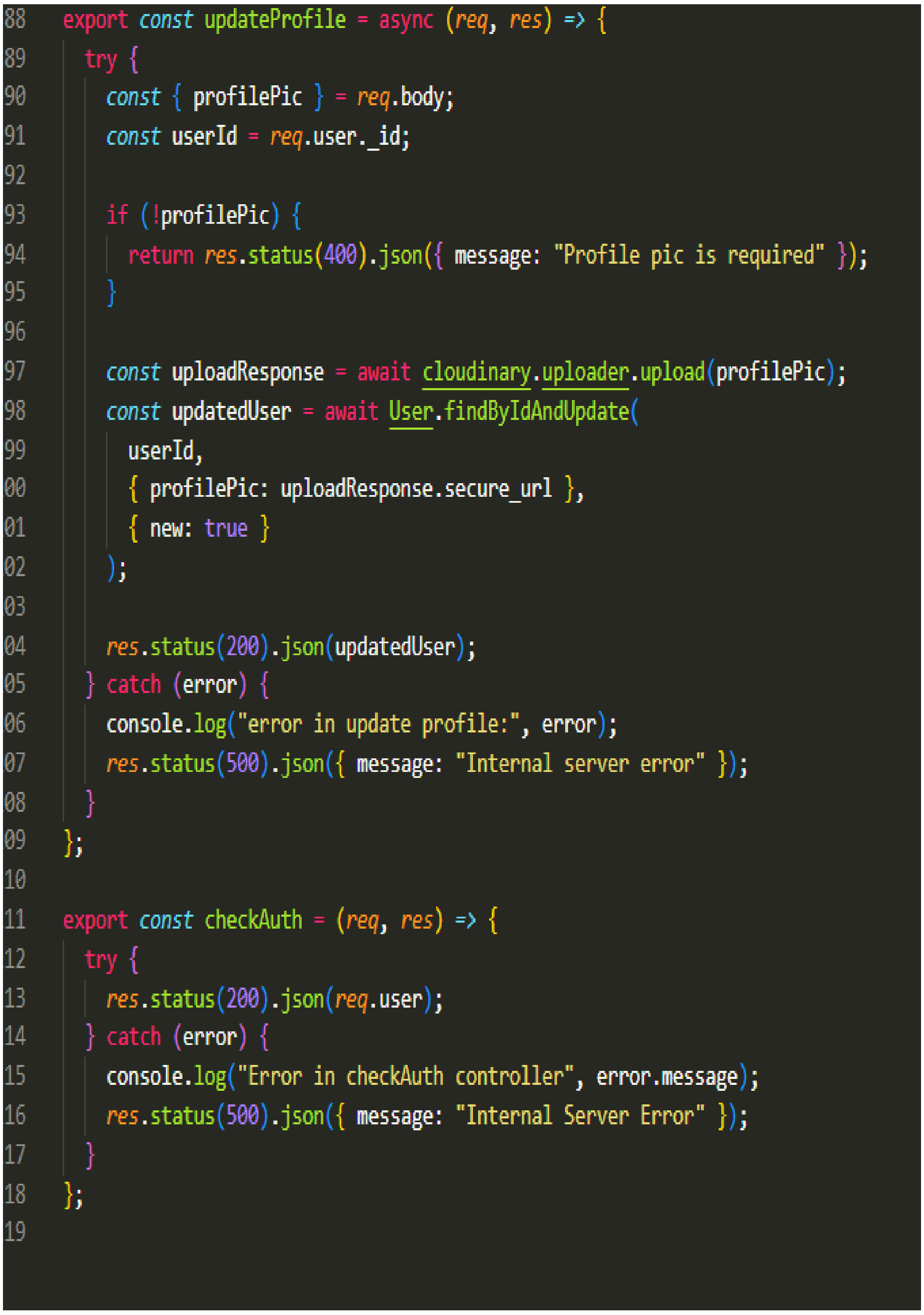
Src/controllers/auth.controllers.js



```javascript
import { generateToken } from "../lib/utils.js";
import User from "../models/user.model.js";
import bcrypt from "bcryptjs";
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" });
    }

    const user = await User.findOne({ email });

    if (user) return res.status(400).json({ message: "Email already exists" });

    const salt = await bcrypt.genSalt(10);
    const hashedPassword = await bcrypt.hash(password, salt);

    const newUser = new User({
      fullName,
      email,
      password: hashedPassword,
    });

    if (newUser) {
      // generate jwt token here
      generateToken(newUser._id, res);
      await newUser.save();

      res.status(201).json({
        _id: newUser._id,
        fullName: newUser.fullName,
        email: newUser.email,
        profilePic: newUser.profilePic,
      });
    } else {
      res.status(400).json({ message: "Invalid user data" });
    }
  } catch (error) {
    console.log("Error in signup controller", error.message);
```

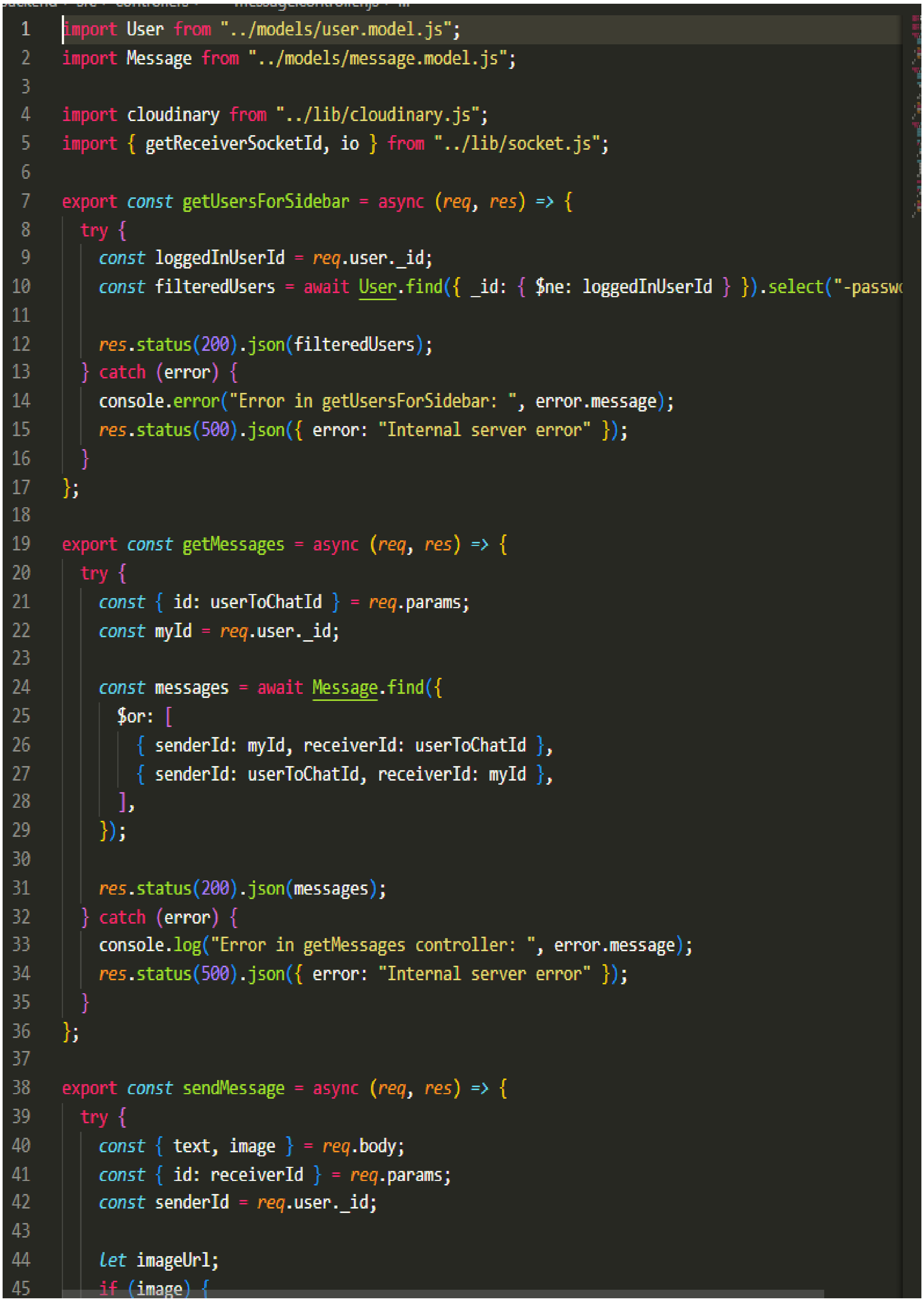


```
  }
};

export const login = async (req, res) => {
  const { email, password } = req.body;
  try {
    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,
      fullName: user.fullName,
      email: user.email,
      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 {
```

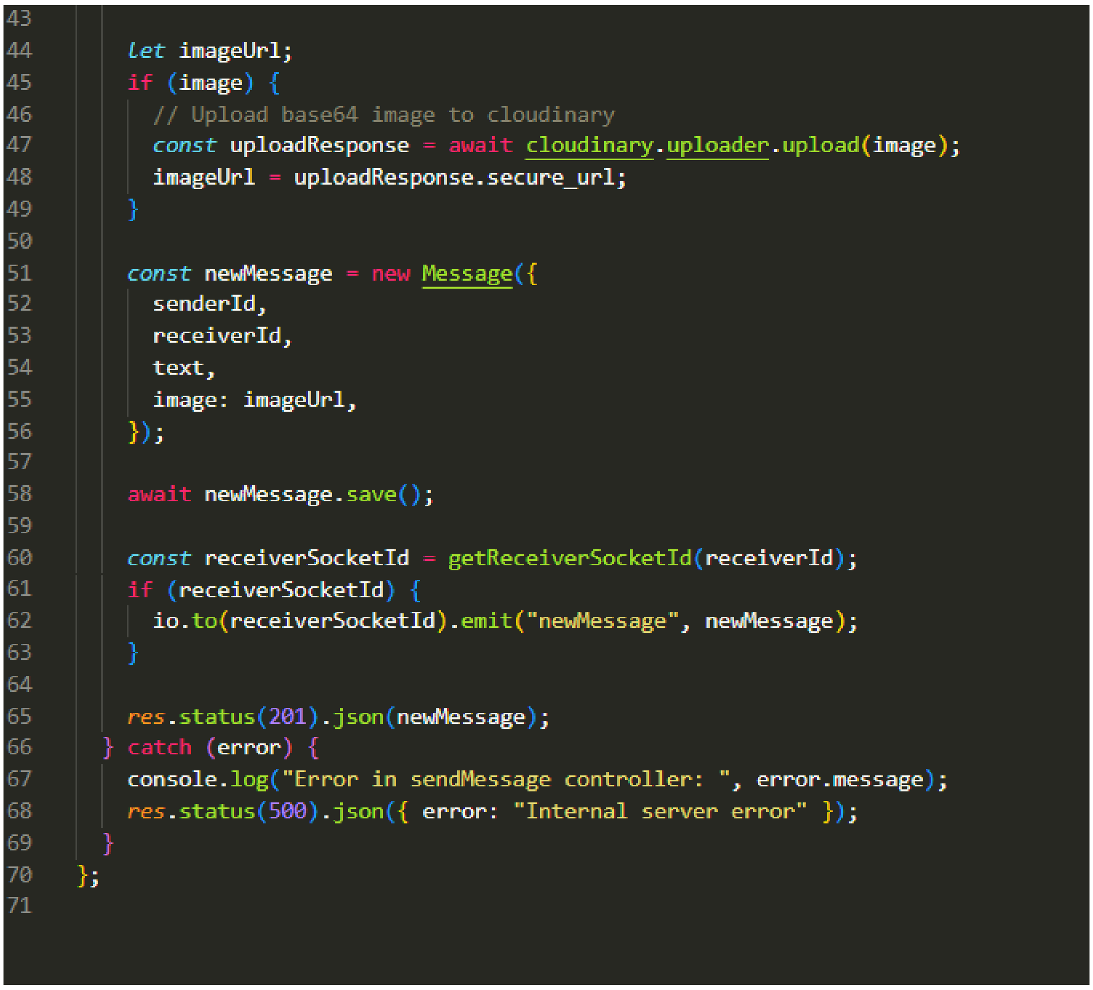


```
export const updateProfile = async (req, res) => {
  try {
    const { profilePic } = req.body;
    const userId = req.user._id;

    if (!profilePic) {
      return res.status(400).json({ message: "Profile pic 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 update profile:", error);
    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" });
  }
};
```

Src/controllers/msg.controller.js

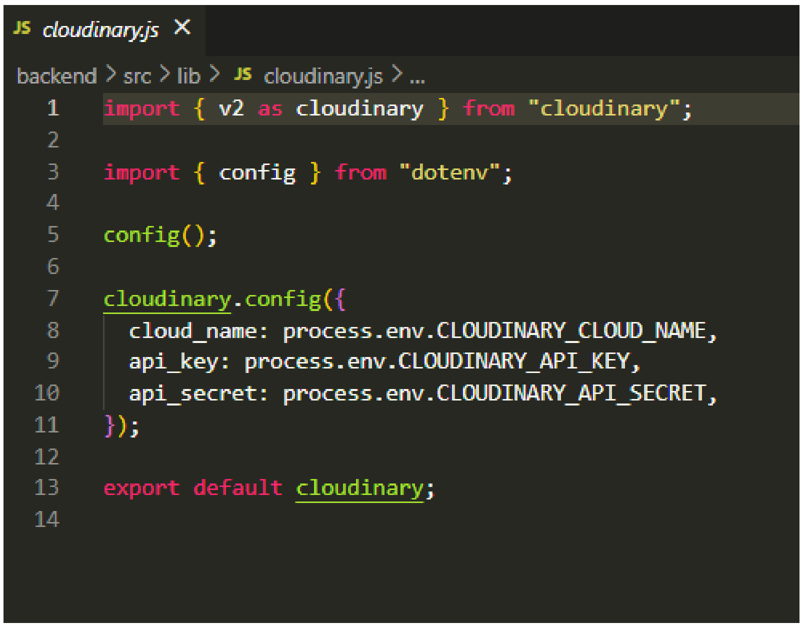


```
import User from "../models/user.model.js";
import Message from "../models/message.model.js";

import cloudinary from "../lib/cloudinary.js";
import { getReceiverSocketId, io } 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("-passw

    res.status(200).json(filteredUsers);
  } catch (error) {
    console.error("Error in getUsersForSidebar: ", error.message);
    res.status(500).json({ error: "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 (error) {
    console.log("Error in getMessages controller: ", error.message);
    res.status(500).json({ error: "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) {
```



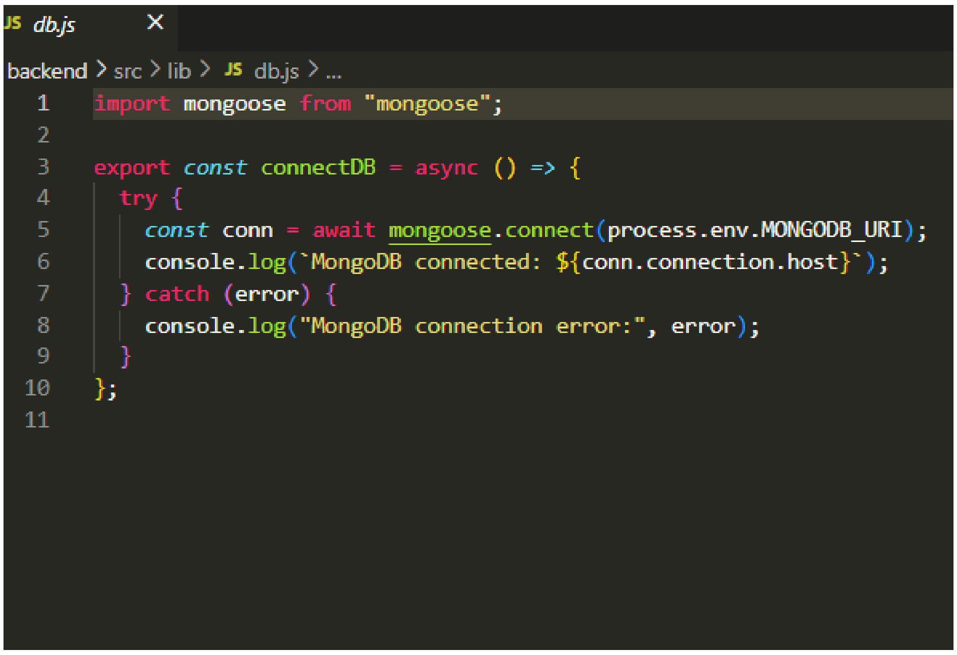
```
      let imageUrl;
      if (image) {
        // Upload base64 image to cloudinary
        const uploadResponse = await cloudinary.uploader.upload(image);
        imageUrl = uploadResponse.secure_url;
      }

      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 (error) {
      console.log("Error in sendMessage controller: ", error.message);
      res.status(500).json({ error: "Internal server error" });
    }
  };

```

Src/lib/cloudinary.js



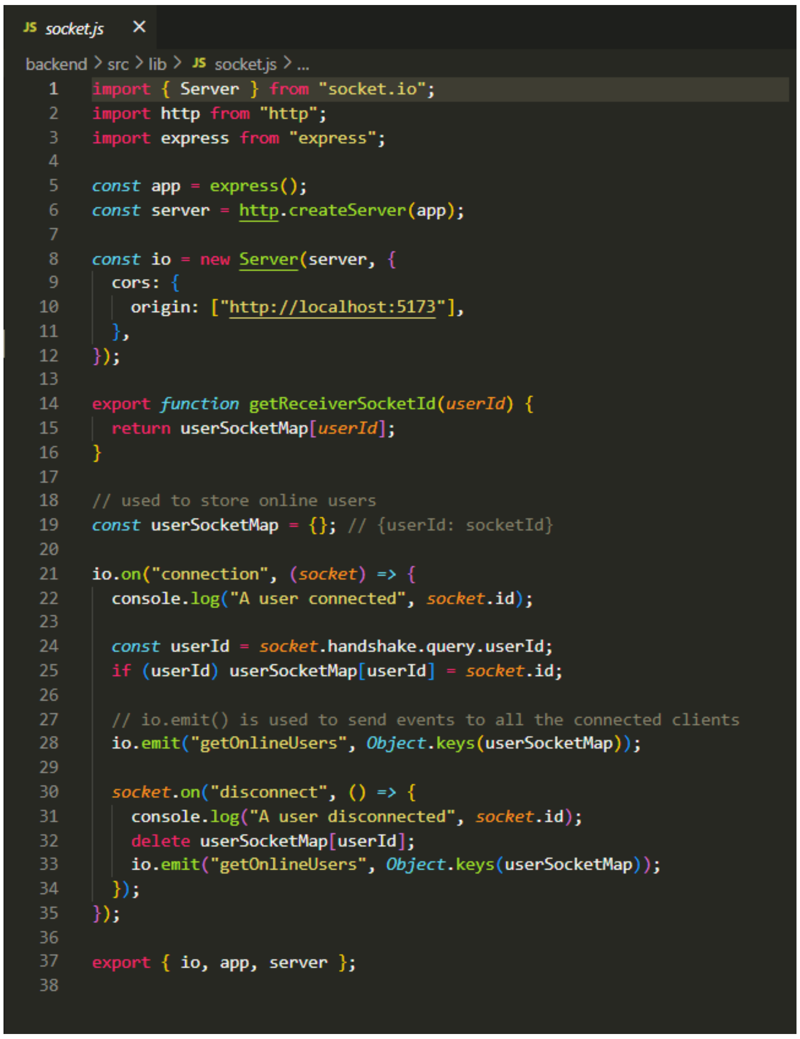
```
import { v2 as cloudinary } from "cloudinary";

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;

```

Src/lib/db.js



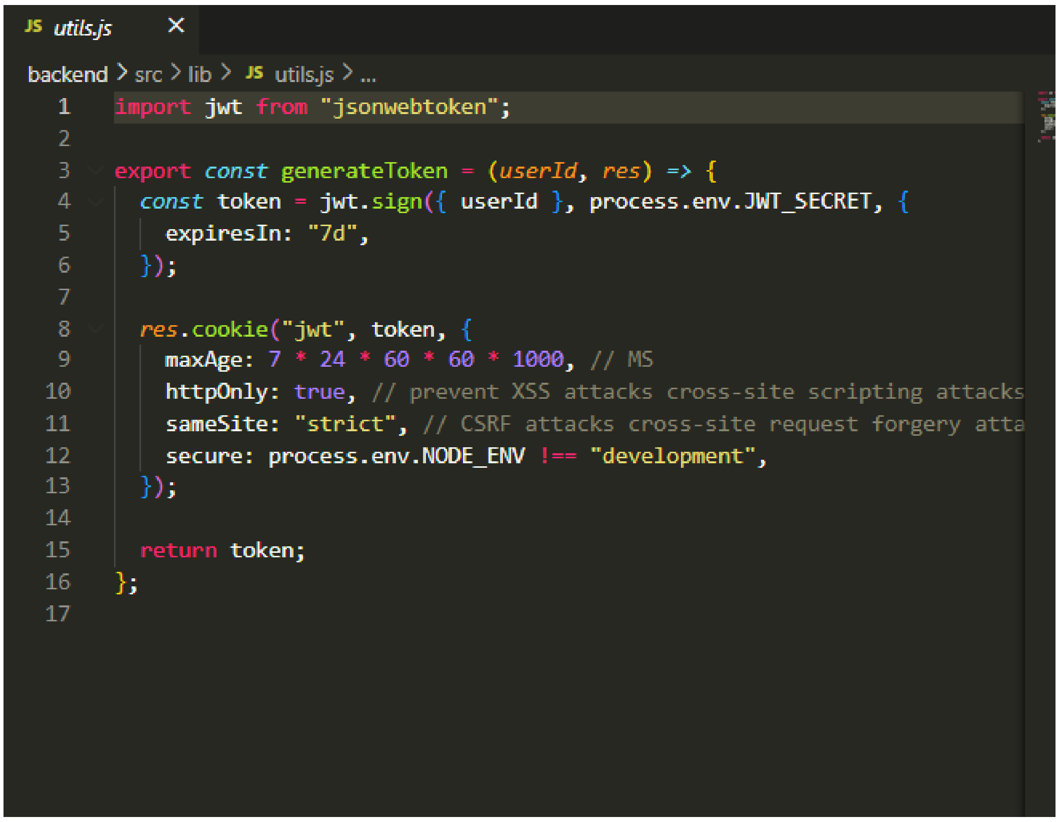
```javascript
import mongoose from "mongoose";

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);
  }
};
```

Src/lib/socket.js



```javascript
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];
}

// used to store online users
const userSocketMap = {}; // {userId: socketId}

io.on("connection", (socket) => {
  console.log("A user connected", socket.id);

  const userId = socket.handshake.query.userId;
  if (userId) userSocketMap[userId] = socket.id;

  // io.emit() is used to send events to all the connected clients
  io.emit("getOnlineUsers", Object.keys(userSocketMap));

  socket.on("disconnect", () => {
    console.log("A user disconnected", socket.id);
    delete userSocketMap[userId];
    io.emit("getOnlineUsers", Object.keys(userSocketMap));
  });
});

export { io, app, server };
```

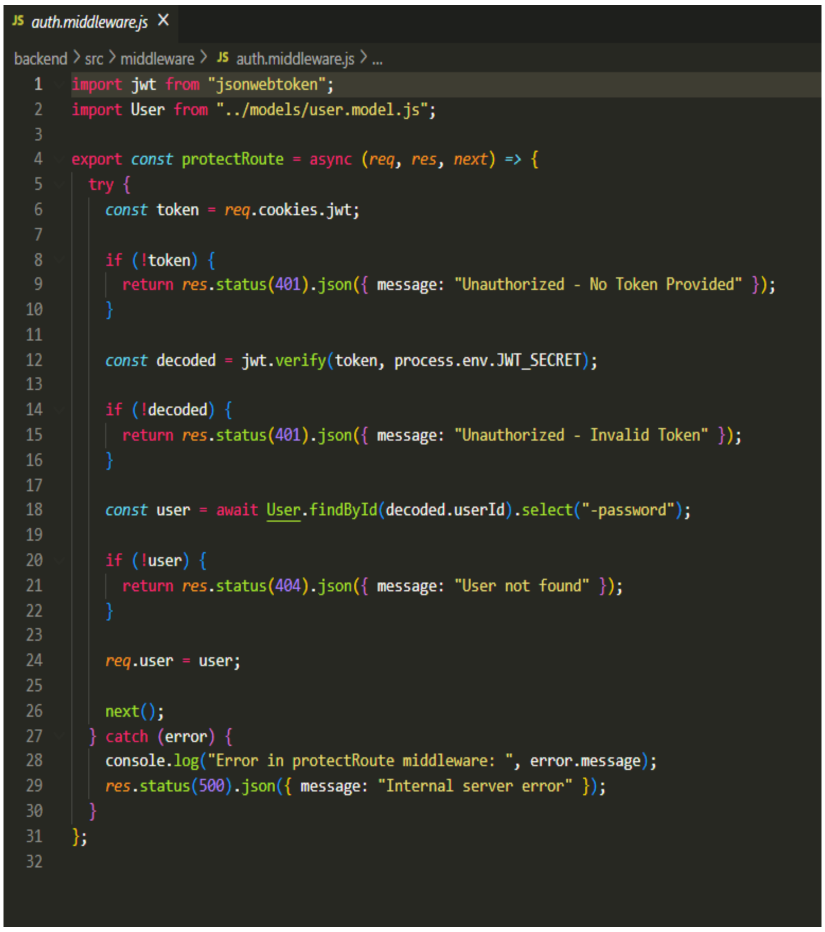
Src/lib/util.js


utils.js

backend > src > lib > utils.js > ...

```javascript
import jwt from "jsonwebtoken";

export const generateToken = (userId, res) => {
  const token = jwt.sign({ userId }, process.env.JWT_SECRET, {
    expiresIn: "7d",
  });

  res.cookie("jwt", token, {
    maxAge: 7 * 24 * 60 * 60 * 1000, // MS
    httpOnly: true, // prevent XSS attacks cross-site scripting attacks
    sameSite: "strict", // CSRF attacks cross-site request forgery atta
    secure: process.env.NODE_ENV !== "development",
  });

  return token;
};
```

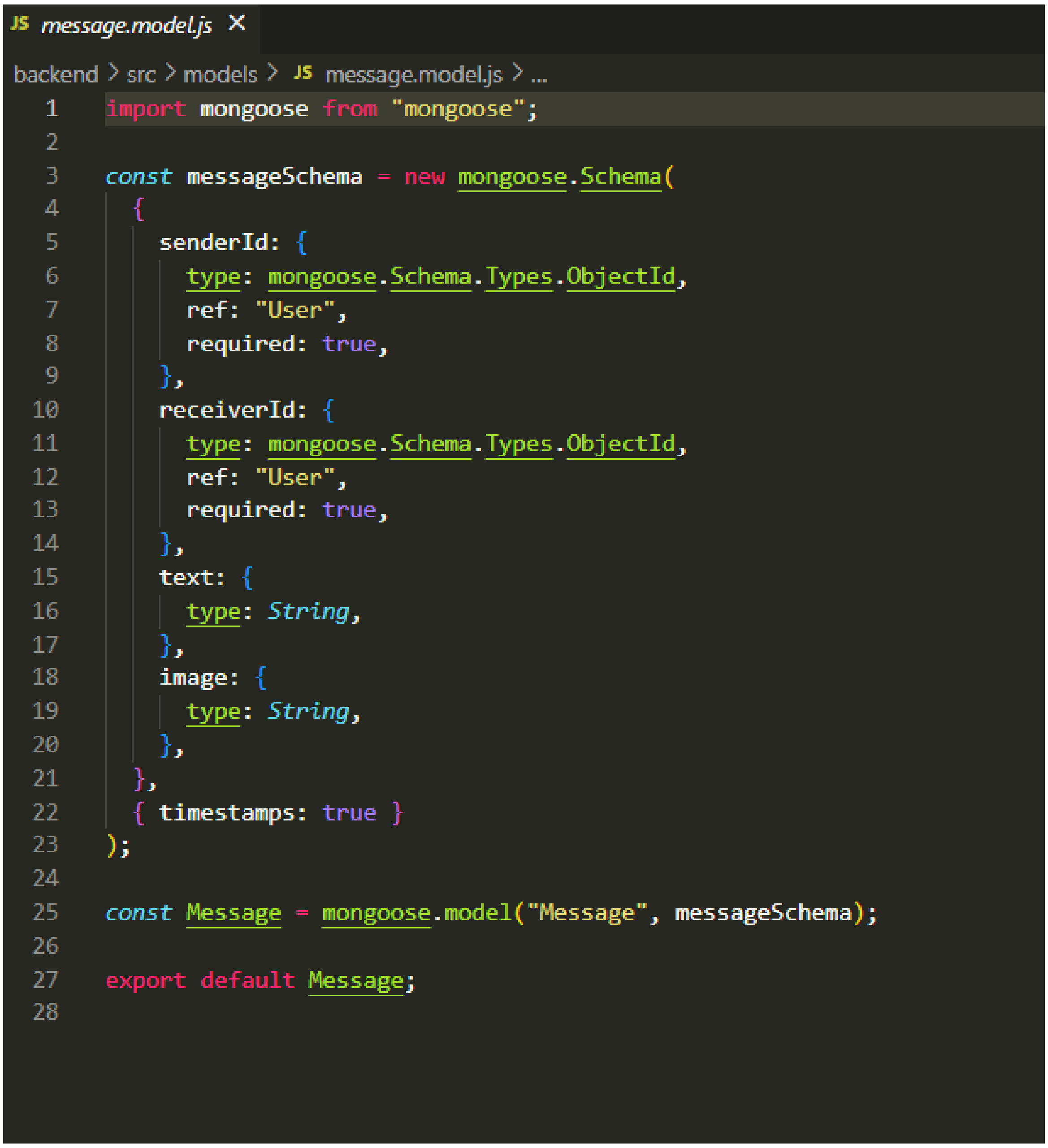
Src/Middleware.js


auth.middleware.js

backend > src > middleware > auth.middleware.js > ...

```javascript
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" });
  }
};
```

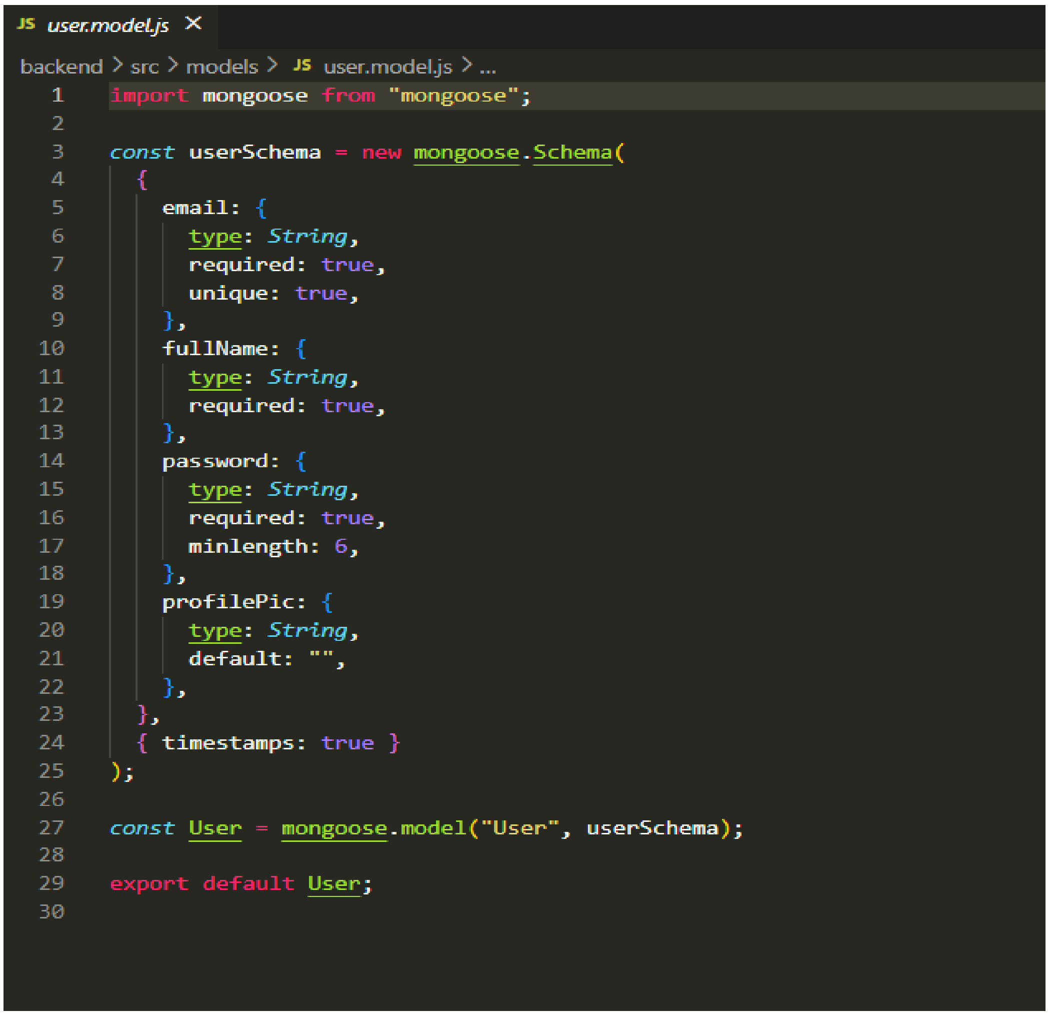
Src/Model/Msg.model.js



```
JS message.model.js ×
backend > src > models > JS message.model.js > ...
```

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

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

export default Message;
```

Src/Model/User.model.js

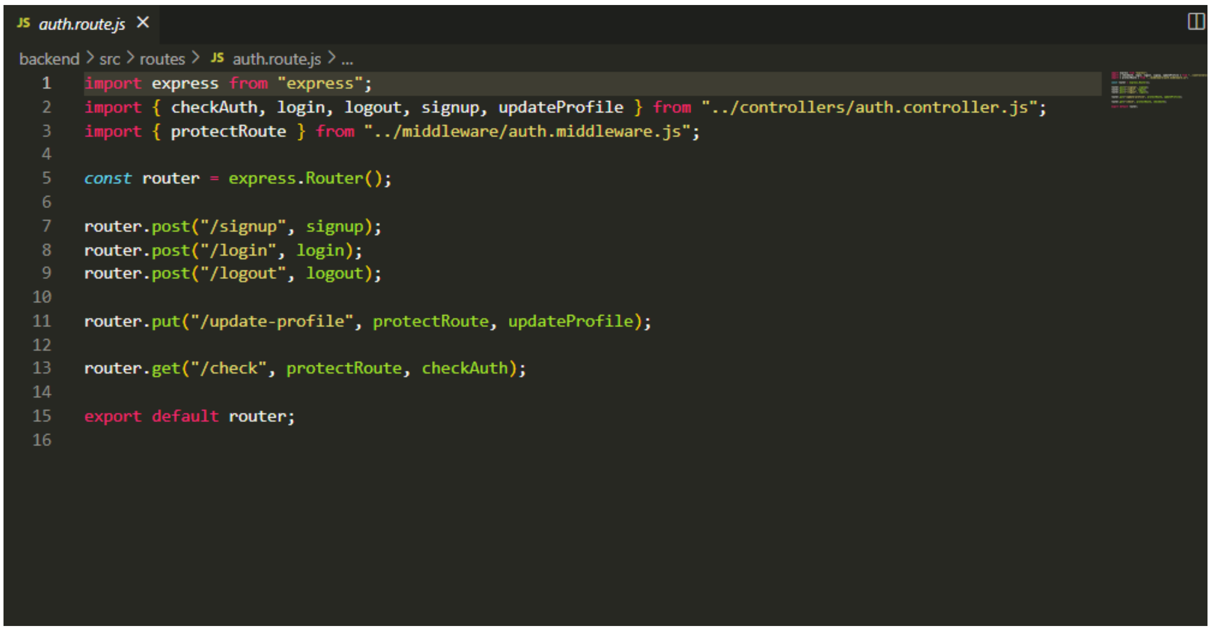


user.model.js

backend > src > models > user.model.js > ...

```javascript
import mongoose from "mongoose";

const userSchema = new mongoose.Schema(
  {
    email: {
      type: String,
      required: true,
      unique: true,
    },
    fullName: {
      type: String,
      required: true,
    },
    password: {
      type: String,
      required: true,
      minlength: 6,
    },
    profilePic: {
      type: String,
      default: "",
    },
  },
  { timestamps: true }
);

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

export default User;

```

Src/Routes/Auth.route.js

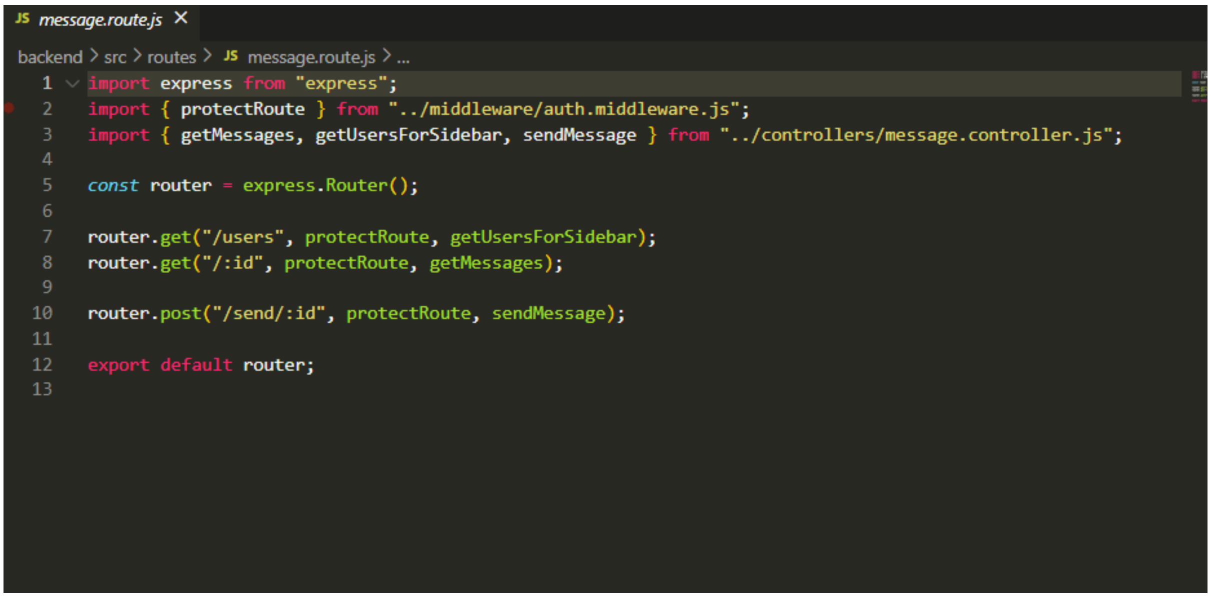


auth.route.js

backend > src > routes > auth.route.js > ...

```javascript
import express from "express";
import { checkAuth, login, logout, signup, updateProfile } 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.get("/check", protectRoute, checkAuth);

export default router;

```

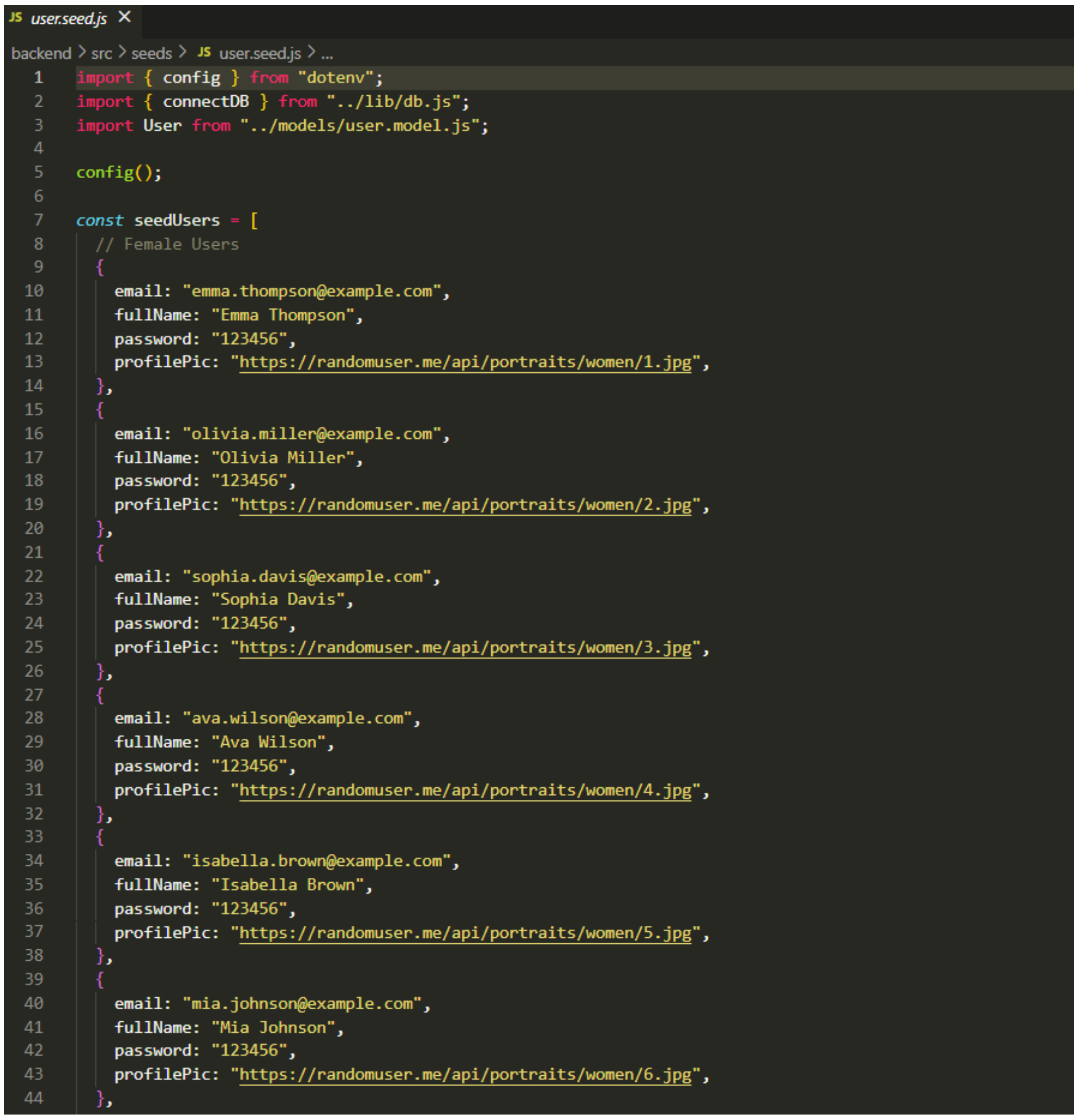
Src/Routes/Msg.route.js


JS message.route.js ×

backend > src > routes > JS message.route.js > ...

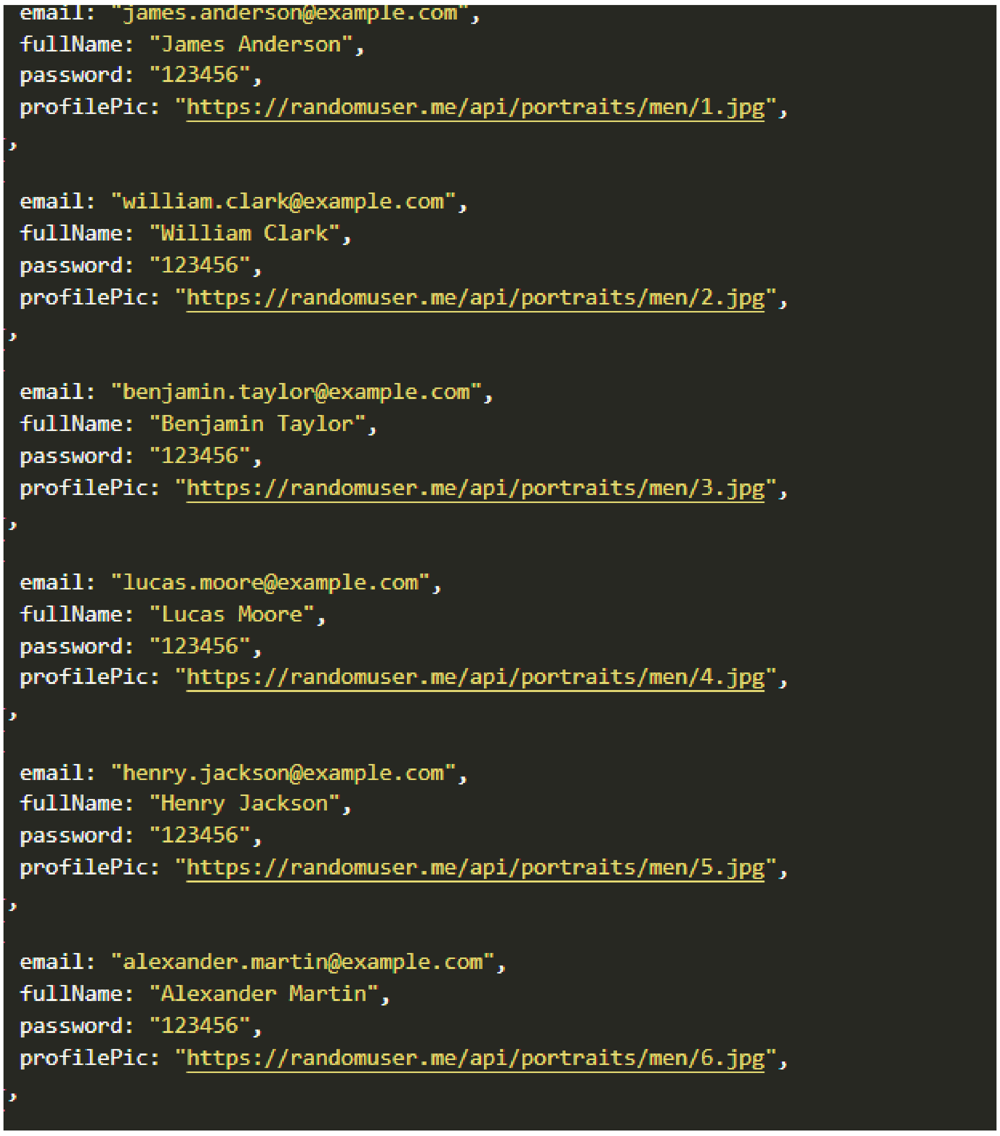
```js
import express from "express";
import { protectRoute } from "../middleware/auth.middleware.js";
import { getMessages, getUsersForSidebar, sendMessage } from "../controllers/message.controller.js";

const router = express.Router();

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

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

export default router;

```

Src/Seeds/user.seeds.js


JS user.seed.js ×

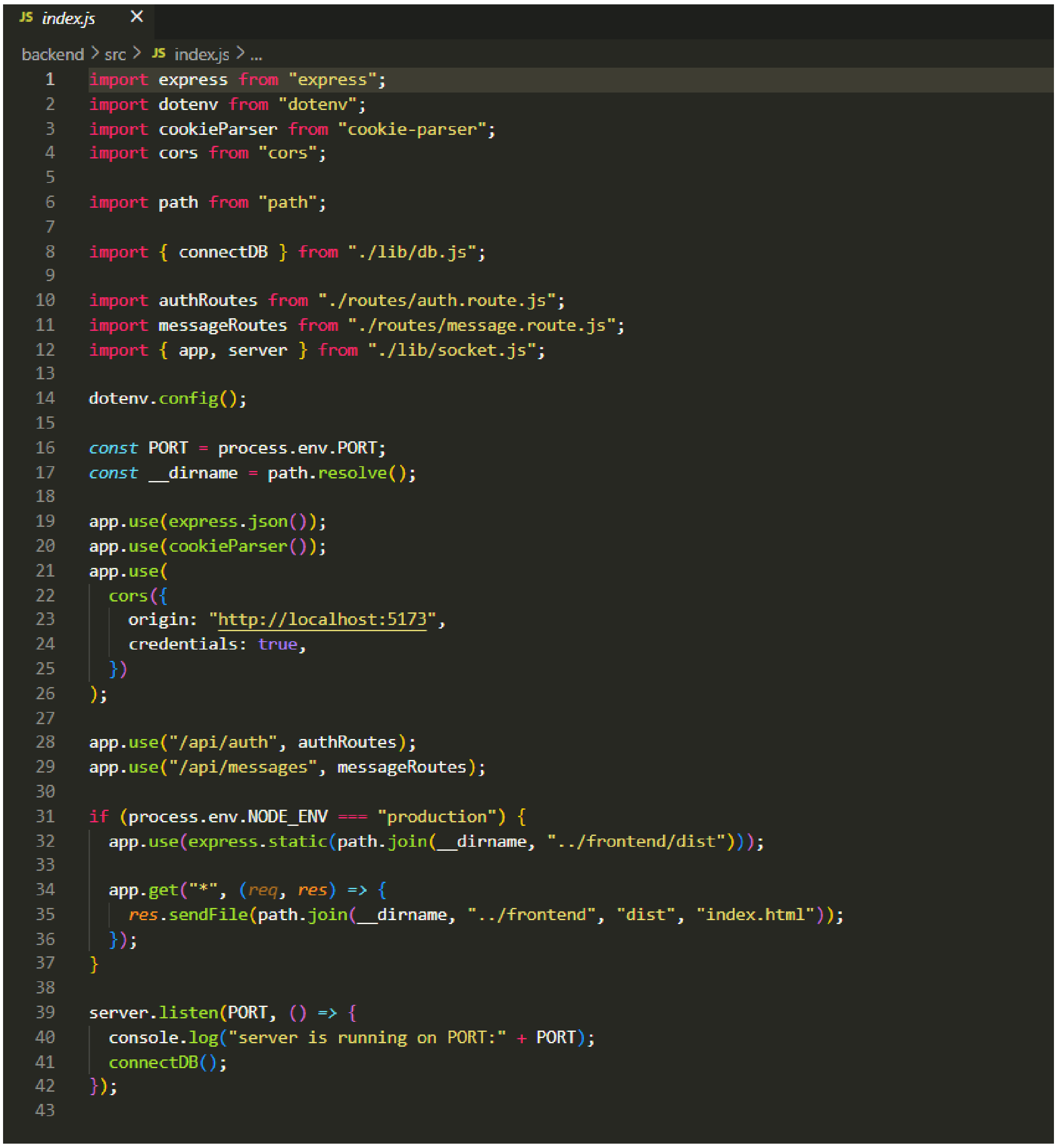
backend > src > seeds > JS user.seed.js > ...

```js
import { config } from "dotenv";
import { connectDB } from "../lib/db.js";
import User from "../models/user.model.js";

config();

const seedUsers = [
  // Female Users
  {
    email: "emma.thompson@example.com",
    fullName: "Emma Thompson",
    password: "123456",
    profilePic: "https://randomuser.me/api/portraits/women/1.jpg",
  },
  {
    email: "olivia.miller@example.com",
    fullName: "Olivia Miller",
    password: "123456",
    profilePic: "https://randomuser.me/api/portraits/women/2.jpg",
  },
  {
    email: "sophia.davis@example.com",
    fullName: "Sophia Davis",
    password: "123456",
    profilePic: "https://randomuser.me/api/portraits/women/3.jpg",
  },
  {
    email: "ava.wilson@example.com",
    fullName: "Ava Wilson",
    password: "123456",
    profilePic: "https://randomuser.me/api/portraits/women/4.jpg",
  },
  {
    email: "isabella.brown@example.com",
    fullName: "Isabella Brown",
    password: "123456",
    profilePic: "https://randomuser.me/api/portraits/women/5.jpg",
  },
  {
    email: "mia.johnson@example.com",
    fullName: "Mia Johnson",
    password: "123456",
    profilePic: "https://randomuser.me/api/portraits/women/6.jpg",
  },
```



```
 email: "james.anderson@example.com",
 fullName: "James Anderson",
 password: "123456",
 profilePic: "https://randomuser.me/api/portraits/men/1.jpg",
,

 email: "william.clark@example.com",
 fullName: "William Clark",
 password: "123456",
 profilePic: "https://randomuser.me/api/portraits/men/2.jpg",
,

 email: "benjamin.taylor@example.com",
 fullName: "Benjamin Taylor",
 password: "123456",
 profilePic: "https://randomuser.me/api/portraits/men/3.jpg",
,

 email: "lucas.moore@example.com",
 fullName: "Lucas Moore",
 password: "123456",
 profilePic: "https://randomuser.me/api/portraits/men/4.jpg",
,

 email: "henry.jackson@example.com",
 fullName: "Henry Jackson",
 password: "123456",
 profilePic: "https://randomuser.me/api/portraits/men/5.jpg",
,

 email: "alexander.martin@example.com",
 fullName: "Alexander Martin",
 password: "123456",
 profilePic: "https://randomuser.me/api/portraits/men/6.jpg",
,
```

Index.js

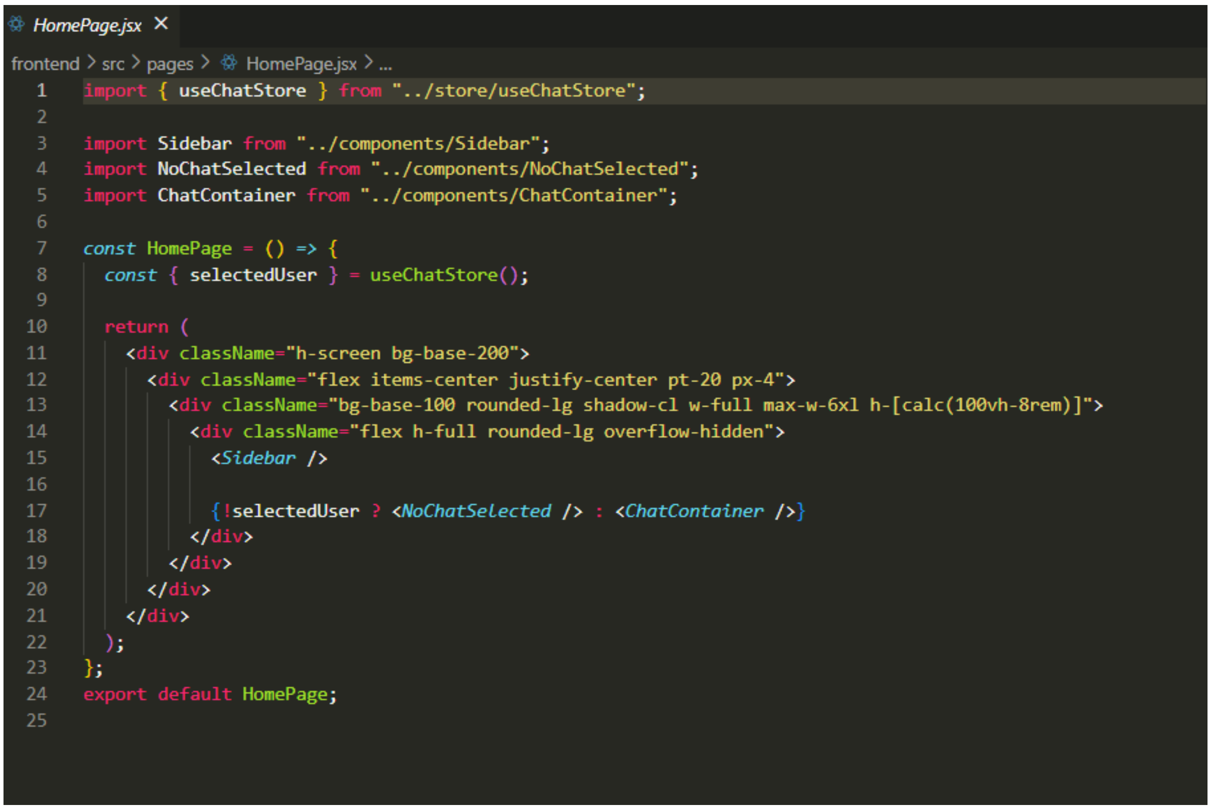


```
import express from "express";
import dotenv from "dotenv";
import cookieParser from "cookie-parser";
import cors from "cors";

import path from "path";

import { connectDB } from "./lib/db.js";

import authRoutes from "./routes/auth.route.js";
import messageRoutes from "./routes/message.route.js";
import { app, server } from "./lib/socket.js";

dotenv.config();

const PORT = process.env.PORT;
const __dirname = path.resolve();

app.use(express.json());
app.use(cookieParser());
app.use(
  cors({
    origin: "http://localhost:5173",
    credentials: true,
  })
);

app.use("/api/auth", authRoutes);
app.use("/api/messages", messageRoutes);

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 PORT:" + PORT);
  connectDB();
});
```

Frontend

Pages

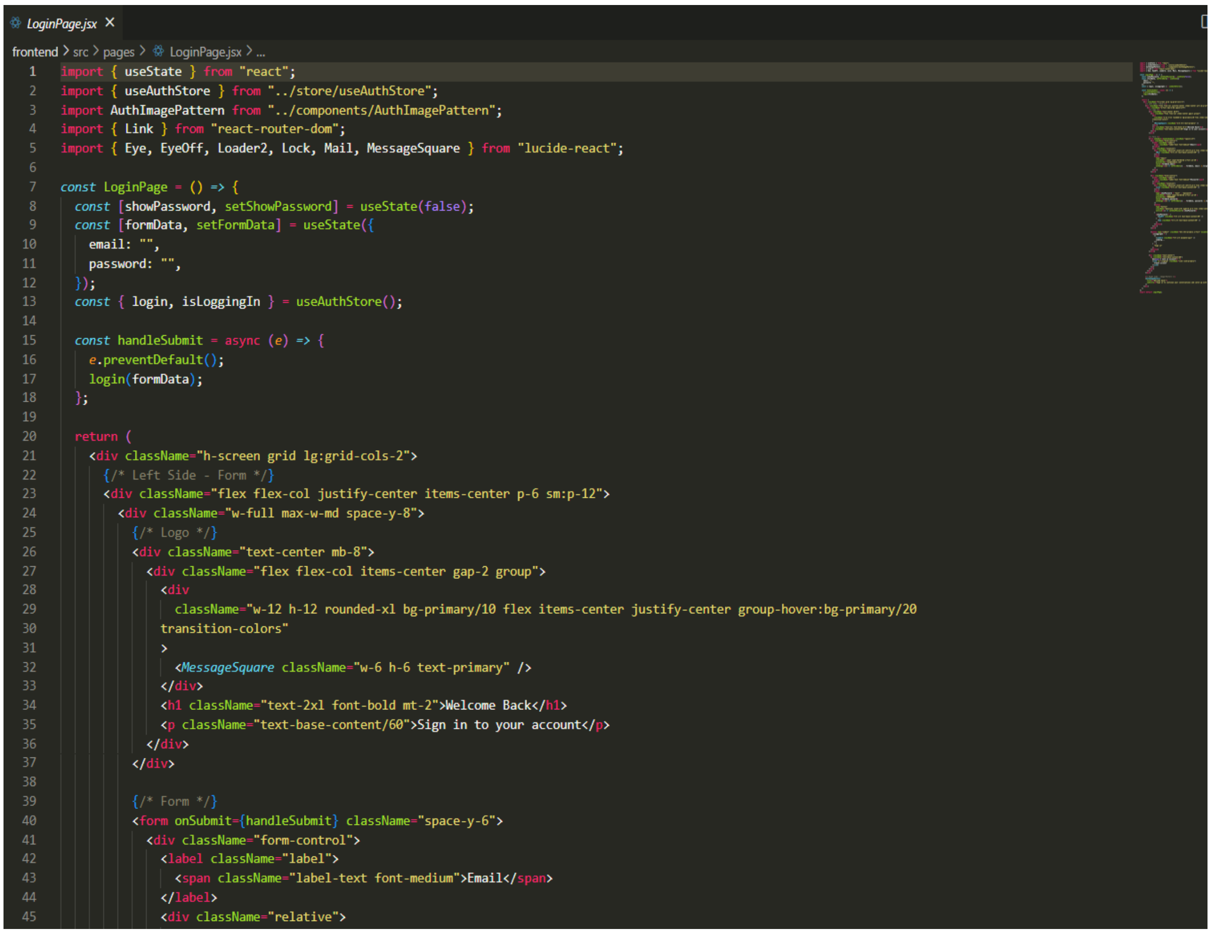
Homepage


HomePage.jsx

frontend > src > pages > HomePage.jsx > ...

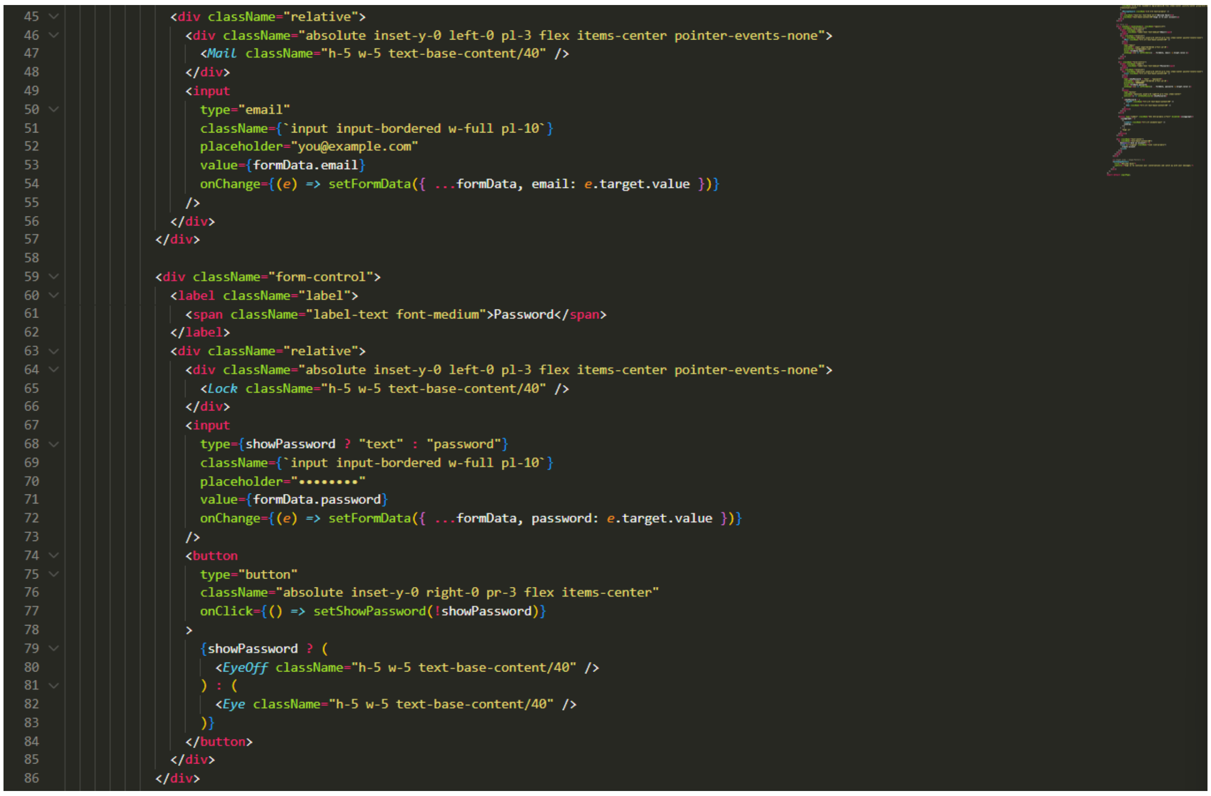
```jsx
import { useChatStore } from "../store/useChatStore";

import Sidebar from "../components/Sidebar";
import NoChatSelected from "../components/NoChatSelected";
import ChatContainer from "../components/ChatContainer";

const HomePage = () => {
  const { selectedUser } = useChatStore();

  return (
    <div className="h-screen bg-base-200">
      <div className="flex items-center justify-center pt-20 px-4">
        <div className="bg-base-100 rounded-lg shadow-cl w-full max-w-6xl h-[calc(100vh-8rem)]">
          <div className="flex h-full rounded-lg overflow-hidden">
            <Sidebar />

            {!selectedUser ? <NoChatSelected /> : <ChatContainer />}
          </div>
        </div>
      </div>
    </div>
  );
};
export default HomePage;

```

Loginpage

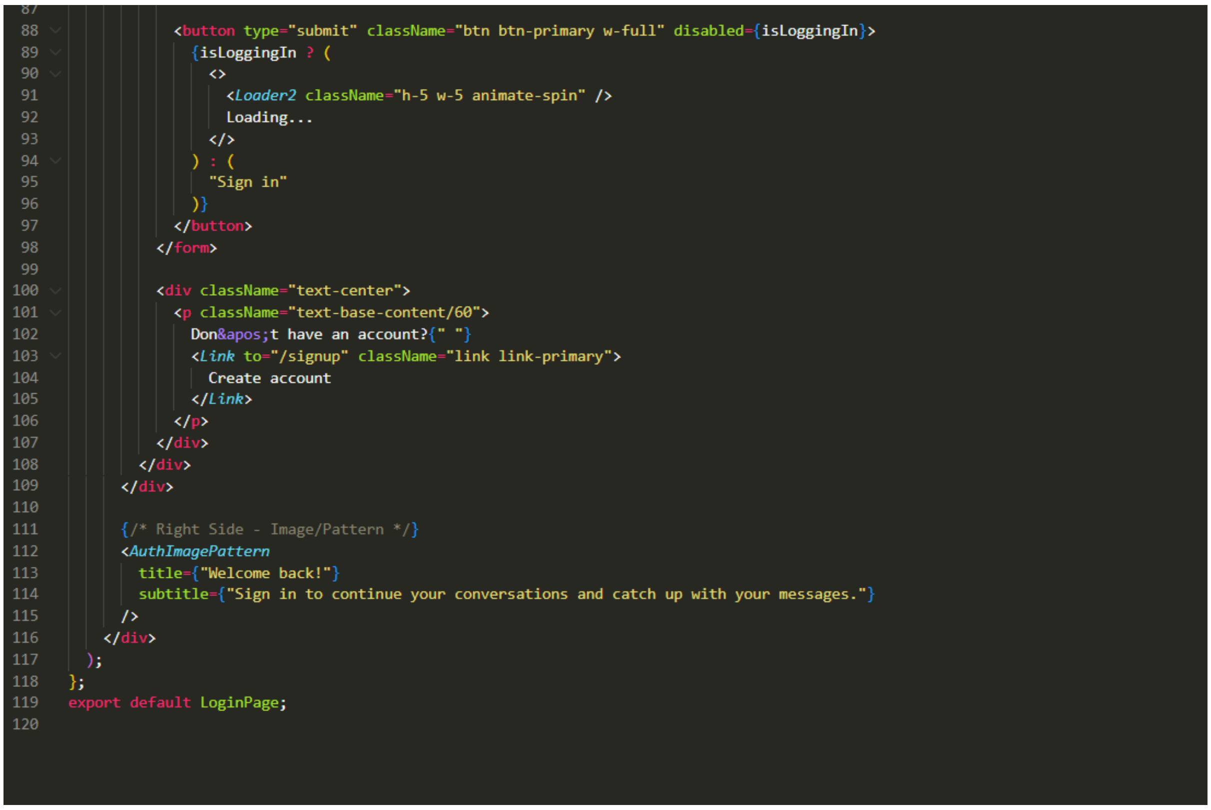

LoginPage.jsx

frontend > src > pages > LoginPage.jsx > ...

```jsx
import { useState } from "react";
import { useAuthStore } from "../store/useAuthStore";
import AuthImagePattern from "../components/AuthImagePattern";
import { Link } from "react-router-dom";
import { Eye, EyeOff, Loader2, Lock, Mail, MessageSquare } from "lucide-react";

const LoginPage = () => {
  const [showPassword, setShowPassword] = useState(false);
  const [formData, setFormData] = useState({
    email: "",
    password: "",
  });
  const { login, isLoggingIn } = useAuthStore();

  const handleSubmit = async (e) => {
    e.preventDefault();
    login(formData);
  };

  return (
    <div className="h-screen grid lg:grid-cols-2">
      {/* Left Side - Form */}
      <div className="flex flex-col justify-center items-center p-6 sm:p-12">
        <div className="w-full max-w-md space-y-8">
          {/* Logo */}
          <div className="text-center mb-8">
            <div className="flex flex-col items-center gap-2 group">
              <div
                className="w-12 h-12 rounded-xl bg-primary/10 flex items-center justify-center group-hover:bg-primary/20
              transition-colors"
              >
                <MessageSquare className="w-6 h-6 text-primary" />
              </div>
              <h1 className="text-2xl font-bold mt-2">Welcome Back</h1>
              <p className="text-base-content/60">Sign in to your account</p>
            </div>
          </div>

          {/* Form */}
          <form onSubmit={handleSubmit} className="space-y-6">
            <div className="form-control">
              <label className="label">
                <span className="label-text font-medium">Email</span>
              </label>
              <div className="relative">
```

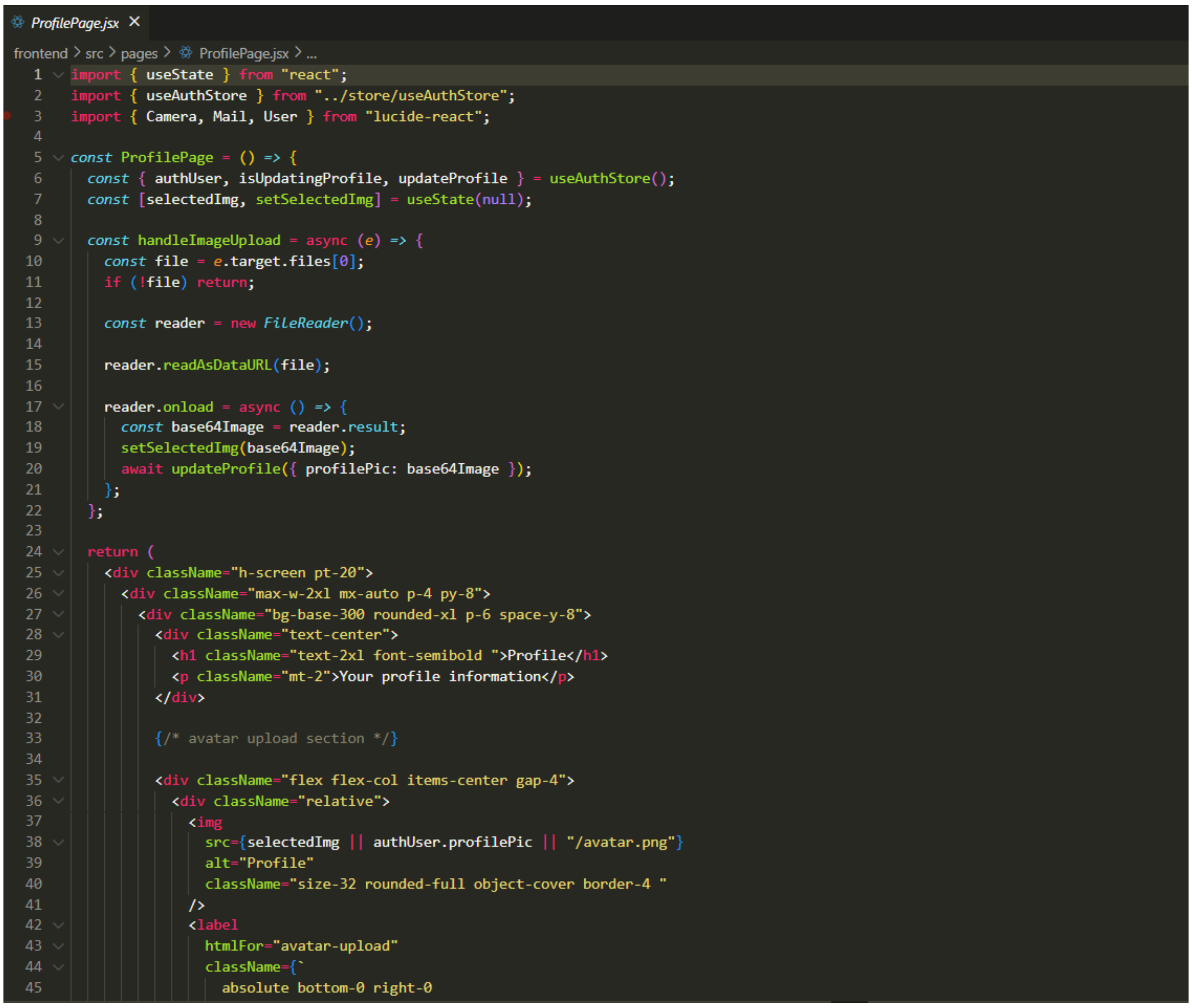


```jsx
              <div className="relative">
                <div className="absolute inset-y-0 left-0 pl-3 flex items-center pointer-events-none">
                  <Mail className="h-5 w-5 text-base-content/40" />
                </div>
                <input
                  type="email"
                  className={`input input-bordered w-full pl-10`}
                  placeholder="you@example.com"
                  value={formData.email}
                  onChange={(e) => setFormData({ ...formData, email: e.target.value })}
                />
              </div>
            </div>

            <div className="form-control">
              <label className="label">
                <span className="label-text font-medium">Password</span>
              </label>
              <div className="relative">
                <div className="absolute inset-y-0 left-0 pl-3 flex items-center pointer-events-none">
                  <Lock className="h-5 w-5 text-base-content/40" />
                </div>
                <input
                  type={showPassword ? "text" : "password"}
                  className={`input input-bordered w-full pl-10`}
                  placeholder="••••••••"
                  value={formData.password}
                  onChange={(e) => setFormData({ ...formData, password: e.target.value })}
                />
                <button
                  type="button"
                  className="absolute inset-y-0 right-0 pr-3 flex items-center"
                  onClick={() => setShowPassword(!showPassword)}
                >
                  {showPassword ? (
                    <EyeOff className="h-5 w-5 text-base-content/40" />
                  ) : (
                    <Eye className="h-5 w-5 text-base-content/40" />
                  )}
                </button>
              </div>
            </div>
```

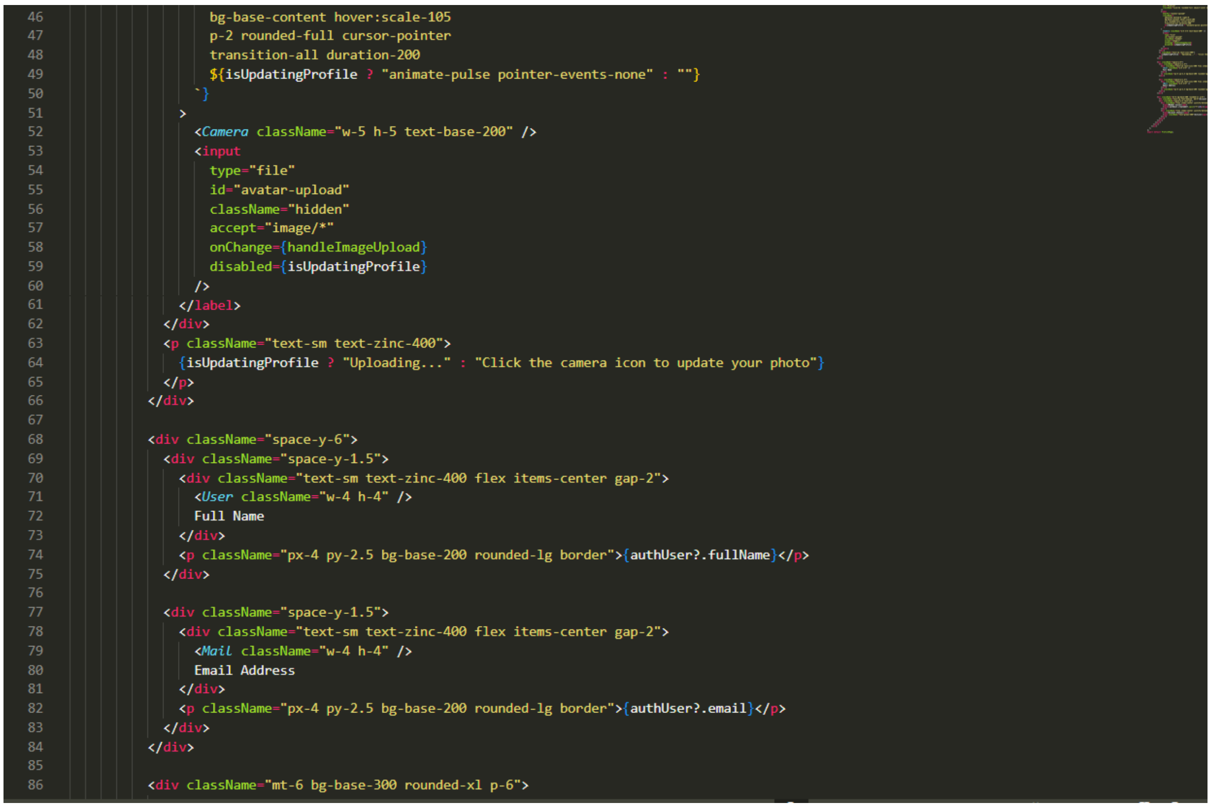


```jsx
            <button type="submit" className="btn btn-primary w-full" disabled={isLoggingIn}>
              {isLoggingIn ? (
                <>
                  <Loader2 className="h-5 w-5 animate-spin" />
                  Loading...
                </>
              ) : (
                "Sign in"
              )}
            </button>
          </form>

          <div className="text-center">
            <p className="text-base-content/60">
              Don&apos;t have an account?{" "}
              <Link to="/signup" className="link link-primary">
                Create account
              </Link>
            </p>
          </div>
        </div>
      </div>

      {/* Right Side - Image/Pattern */}
      <AuthImagePattern
        title={"Welcome back!"}
        subtitle={"Sign in to continue your conversations and catch up with your messages."}
      />
    </div>
  );
};
export default LoginPage;
```

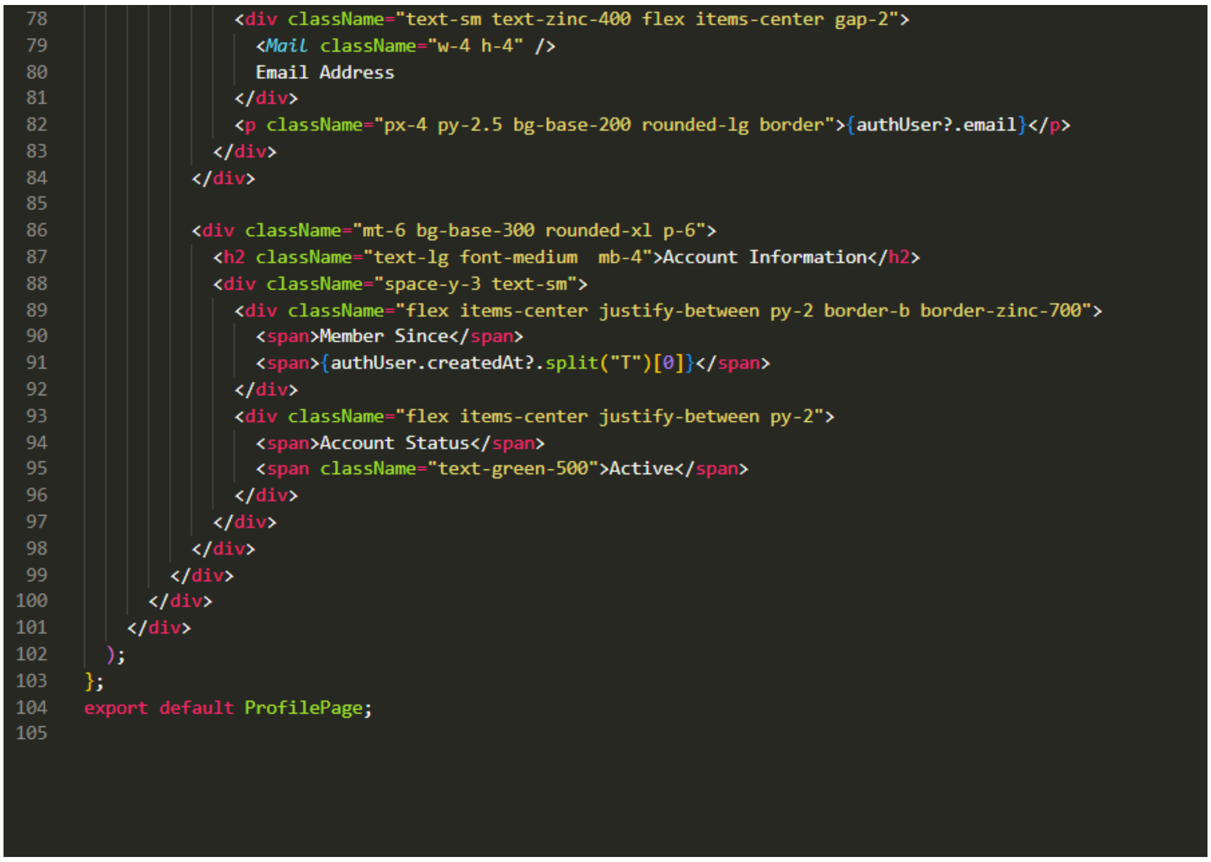
Profilepage



```jsx
import { useState } from "react";
import { useAuthStore } from "../store/useAuthStore";
import { Camera, Mail, User } from "lucide-react";

const ProfilePage = () => {
  const { authUser, isUpdatingProfile, updateProfile } = useAuthStore();
  const [selectedImg, setSelectedImg] = useState(null);

  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;
      setSelectedImg(base64Image);
      await updateProfile({ profilePic: base64Image });
    };
  };

  return (
    <div className="h-screen pt-20">
      <div className="max-w-2xl mx-auto p-4 py-8">
        <div className="bg-base-300 rounded-xl p-6 space-y-8">
          <div className="text-center">
            <h1 className="text-2xl font-semibold ">Profile</h1>
            <p className="mt-2">Your profile information</p>
          </div>

          {/* avatar upload section */}

          <div className="flex flex-col items-center gap-4">
            <div className="relative">
              <img
                src={selectedImg || 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
```

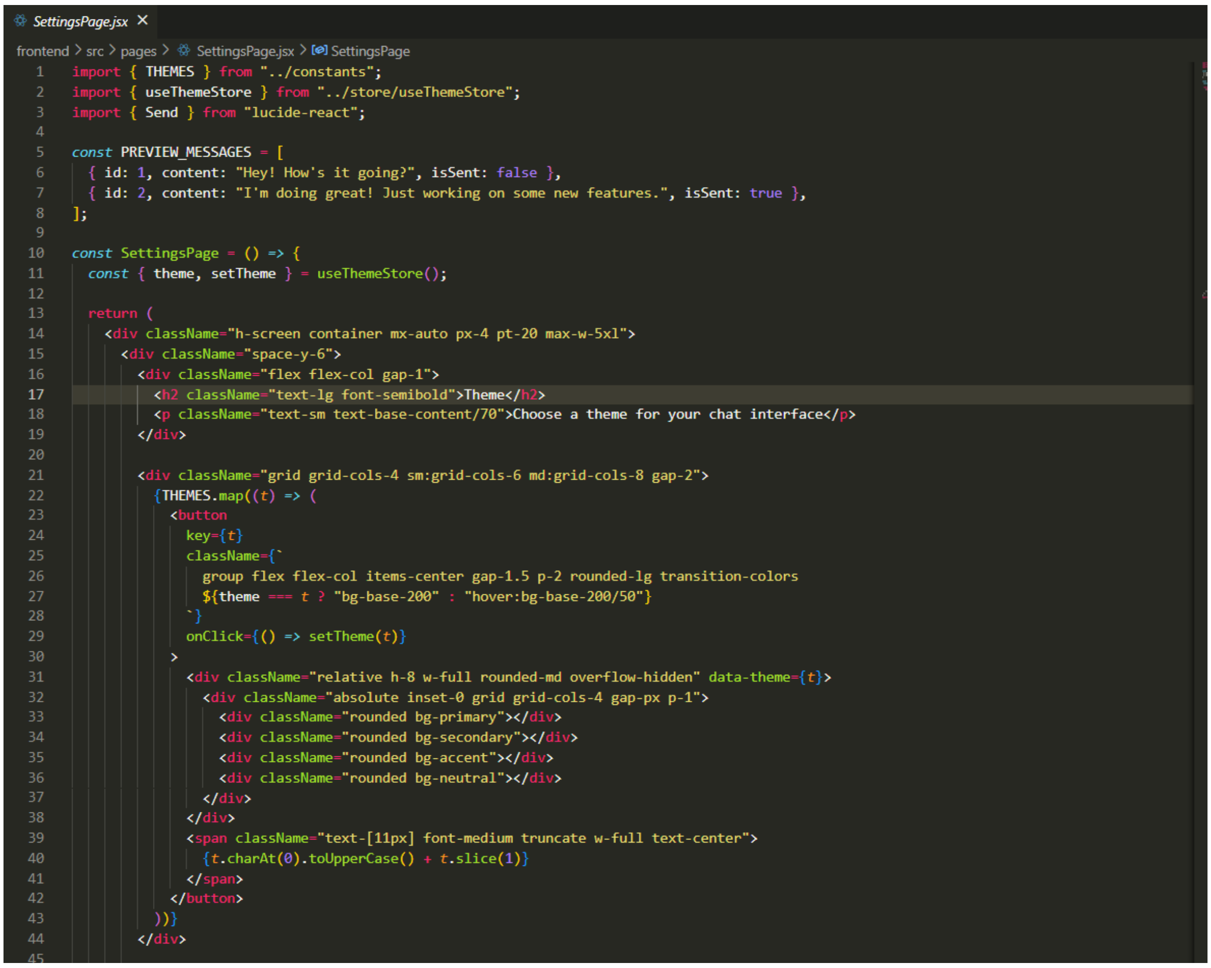


```jsx
                  bg-base-content hover:scale-105
                  p-2 rounded-full cursor-pointer
                  transition-all duration-200
                  ${isUpdatingProfile ? "animate-pulse pointer-events-none" : ""}
                `}
              >
                <Camera className="w-5 h-5 text-base-200" />
                <input
                  type="file"
                  id="avatar-upload"
                  className="hidden"
                  accept="image/*"
                  onChange={handleImageUpload}
                  disabled={isUpdatingProfile}
                />
              </label>
            </div>
            <p className="text-sm text-zinc-400">
              {isUpdatingProfile ? "Uploading..." : "Click the camera icon to update your photo"}
            </p>
          </div>

          <div className="space-y-6">
            <div className="space-y-1.5">
              <div className="text-sm text-zinc-400 flex items-center gap-2">
                <User className="w-4 h-4" />
                Full Name
              </div>
              <p className="px-4 py-2.5 bg-base-200 rounded-lg border">{authUser?.fullName}</p>
            </div>

            <div className="space-y-1.5">
              <div className="text-sm text-zinc-400 flex items-center gap-2">
                <Mail className="w-4 h-4" />
                Email Address
              </div>
              <p className="px-4 py-2.5 bg-base-200 rounded-lg border">{authUser?.email}</p>
            </div>
          </div>

          <div className="mt-6 bg-base-300 rounded-xl p-6">
```

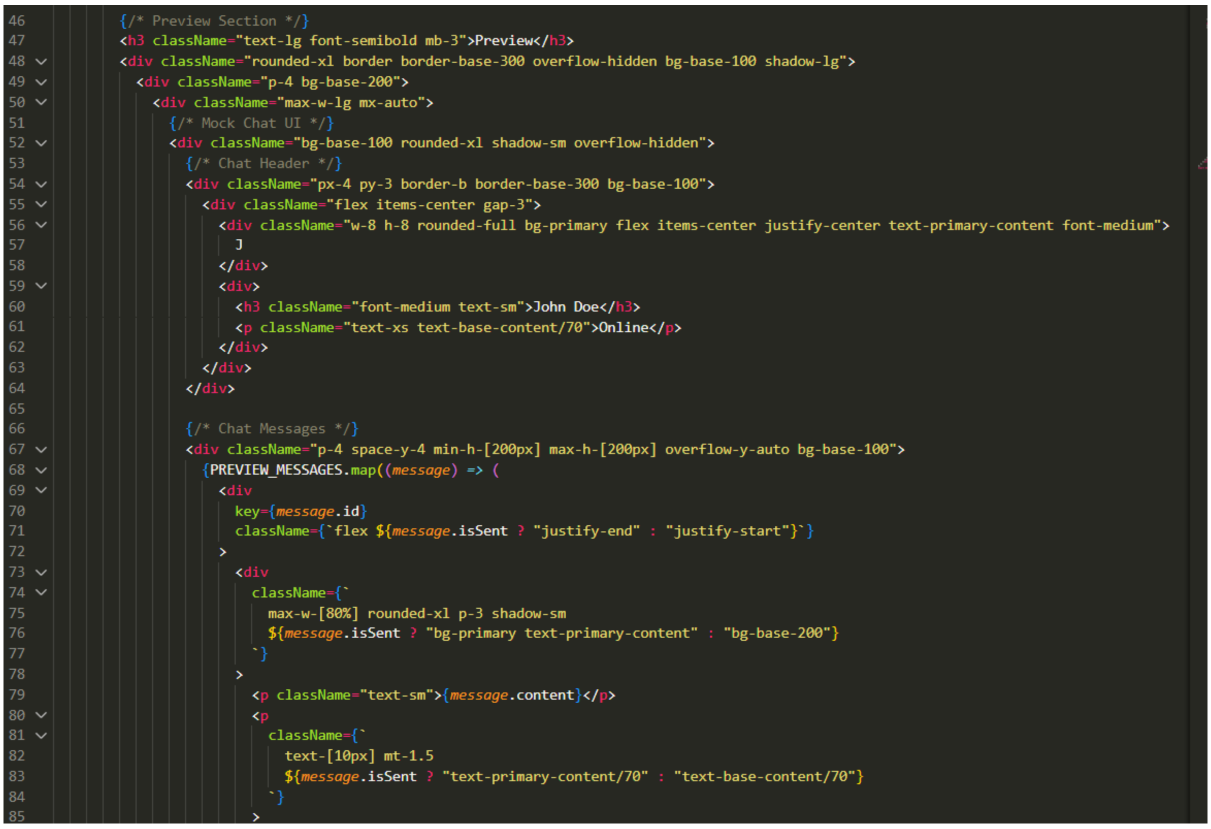


```
              <div className="text-sm text-zinc-400 flex items-center gap-2">
                <Mail className="w-4 h-4" />
                Email Address
              </div>
              <p className="px-4 py-2.5 bg-base-200 rounded-lg border">{authUser?.email}</p>
            </div>
          </div>

          <div className="mt-6 bg-base-300 rounded-xl p-6">
            <h2 className="text-lg font-medium  mb-4">Account Information</h2>
            <div className="space-y-3 text-sm">
              <div className="flex items-center justify-between py-2 border-b border-zinc-700">
                <span>Member Since</span>
                <span>{authUser.createdAt?.split("T")[0]}</span>
              </div>
              <div className="flex items-center justify-between py-2">
                <span>Account Status</span>
                <span className="text-green-500">Active</span>
              </div>
            </div>
          </div>
        </div>
      </div>
    </div>
  );
};
export default ProfilePage;
```

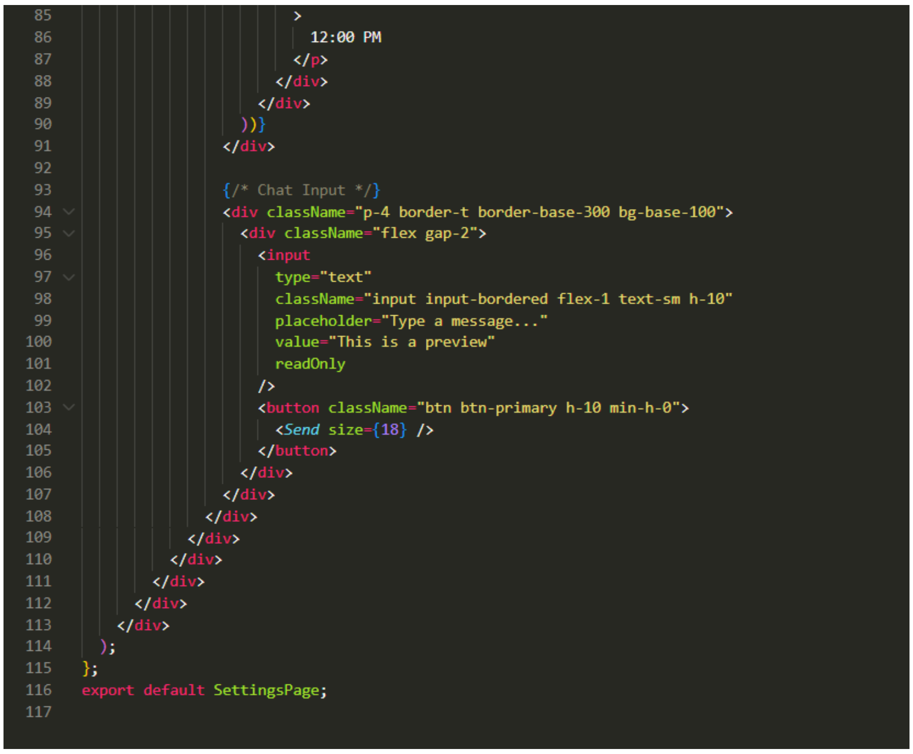
Settingspage



```jsx
import { THEMES } from "../constants";
import { useThemeStore } from "../store/useThemeStore";
import { Send } from "lucide-react";

const PREVIEW_MESSAGES = [
  { id: 1, content: "Hey! How's it going?", isSent: false },
  { id: 2, content: "I'm doing great! Just working on some new features.", isSent: true },
];

const SettingsPage = () => {
  const { theme, setTheme } = useThemeStore();

  return (
    <div className="h-screen container mx-auto px-4 pt-20 max-w-5xl">
      <div className="space-y-6">
        <div className="flex flex-col gap-1">
          <h2 className="text-lg font-semibold">Theme</h2>
          <p className="text-sm text-base-content/70">Choose a theme for your chat interface</p>
        </div>

        <div className="grid grid-cols-4 sm:grid-cols-6 md:grid-cols-8 gap-2">
          {THEMES.map((t) => (
            <button
              key={t}
              className={`
                group flex flex-col items-center gap-1.5 p-2 rounded-lg transition-colors
                ${theme === t ? "bg-base-200" : "hover:bg-base-200/50"}
              `}
              onClick={() => setTheme(t)}
            >
              <div className="relative h-8 w-full rounded-md overflow-hidden" data-theme={t}>
                <div className="absolute inset-0 grid grid-cols-4 gap-px p-1">
                  <div className="rounded bg-primary"></div>
                  <div className="rounded bg-secondary"></div>
                  <div className="rounded bg-accent"></div>
                  <div className="rounded bg-neutral"></div>
                </div>
              </div>
              <span className="text-[11px] font-medium truncate w-full text-center">
                {t.charAt(0).toUpperCase() + t.slice(1)}
              </span>
            </button>
          ))}
        </div>

```

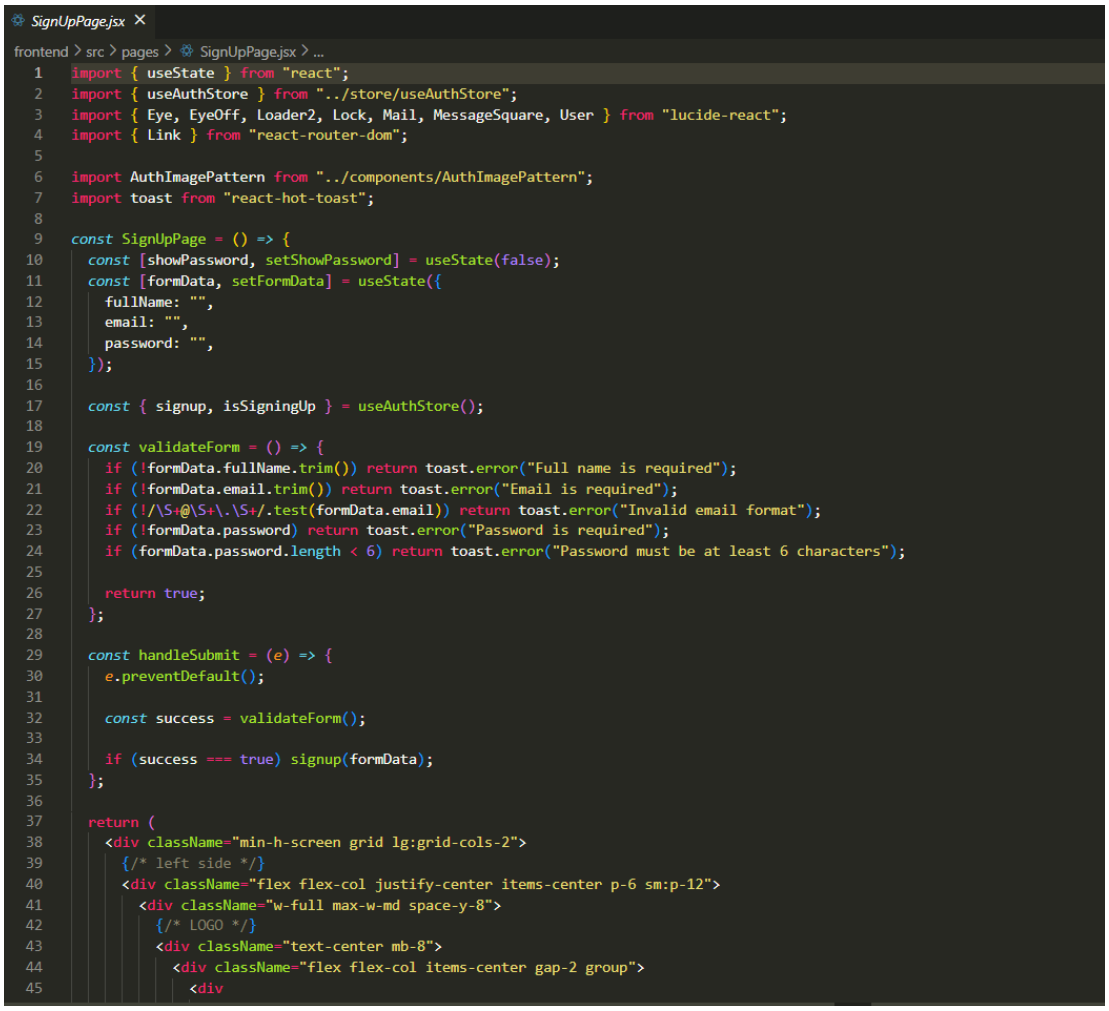


```jsx
        {/* Preview Section */}
        <h3 className="text-lg font-semibold mb-3">Preview</h3>
        <div className="rounded-xl border border-base-300 overflow-hidden bg-base-100 shadow-lg">
          <div className="p-4 bg-base-200">
            <div className="max-w-lg mx-auto">
              {/* Mock Chat UI */}
              <div className="bg-base-100 rounded-xl shadow-sm overflow-hidden">
                {/* Chat Header */}
                <div className="px-4 py-3 border-b border-base-300 bg-base-100">
                  <div className="flex items-center gap-3">
                    <div className="w-8 h-8 rounded-full bg-primary flex items-center justify-center text-primary-content font-medium">
                      J
                    </div>
                    <div>
                      <h3 className="font-medium text-sm">John Doe</h3>
                      <p className="text-xs text-base-content/70">Online</p>
                    </div>
                  </div>
                </div>

                {/* Chat Messages */}
                <div className="p-4 space-y-4 min-h-[200px] max-h-[200px] overflow-y-auto bg-base-100">
                  {PREVIEW_MESSAGES.map((message) => (
                    <div
                      key={message.id}
                      className={`flex ${message.isSent ? "justify-end" : "justify-start"}`}
                    >
                      <div
                        className={`
                          max-w-[80%] rounded-xl p-3 shadow-sm
                          ${message.isSent ? "bg-primary text-primary-content" : "bg-base-200"}
                        `}
                      >
                        <p className="text-sm">{message.content}</p>
                        <p
                          className={`
                            text-[10px] mt-1.5
                            ${message.isSent ? "text-primary-content/70" : "text-base-content/70"}
                          `}
                        >
```

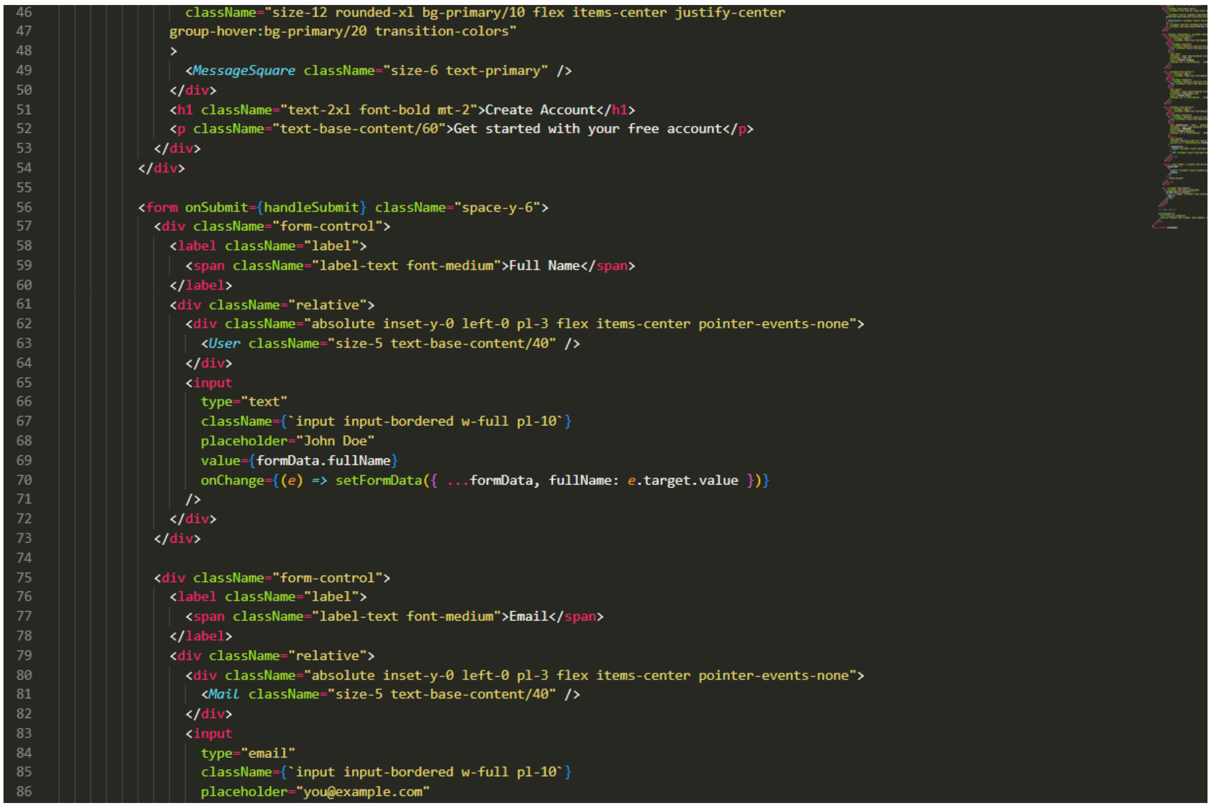


```
                        >
                          12:00 PM
                        </p>
                      </div>
                    </div>
                  ))}
                </div>

                {/* Chat Input */}
                <div className="p-4 border-t border-base-300 bg-base-100">
                  <div className="flex gap-2">
                    <input
                      type="text"
                      className="input input-bordered flex-1 text-sm h-10"
                      placeholder="Type a message..."
                      value="This is a preview"
                      readOnly
                    />
                    <button className="btn btn-primary h-10 min-h-0">
                      <Send size={18} />
                    </button>
                  </div>
                </div>
              </div>
            </div>
          </div>
        </div>
      </div>
    </div>
  );
};
export default SettingsPage;
```

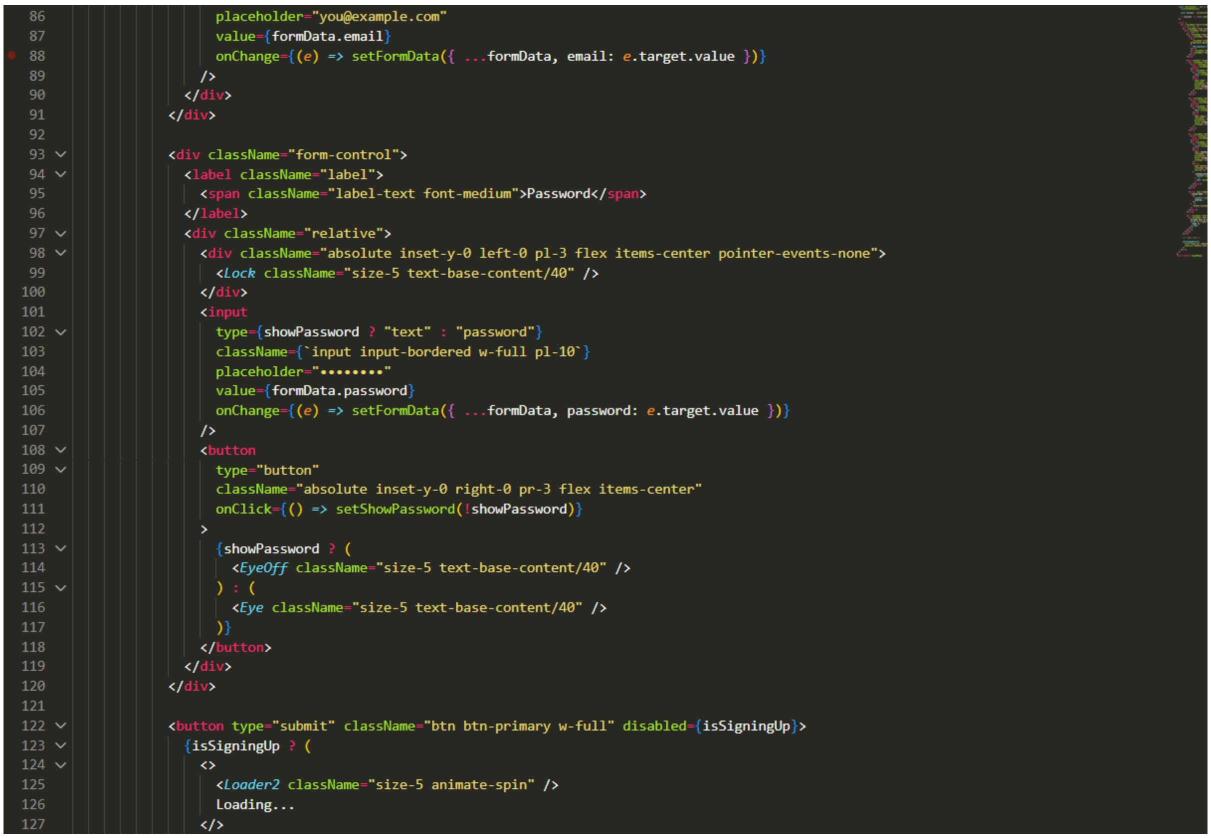
Signup page



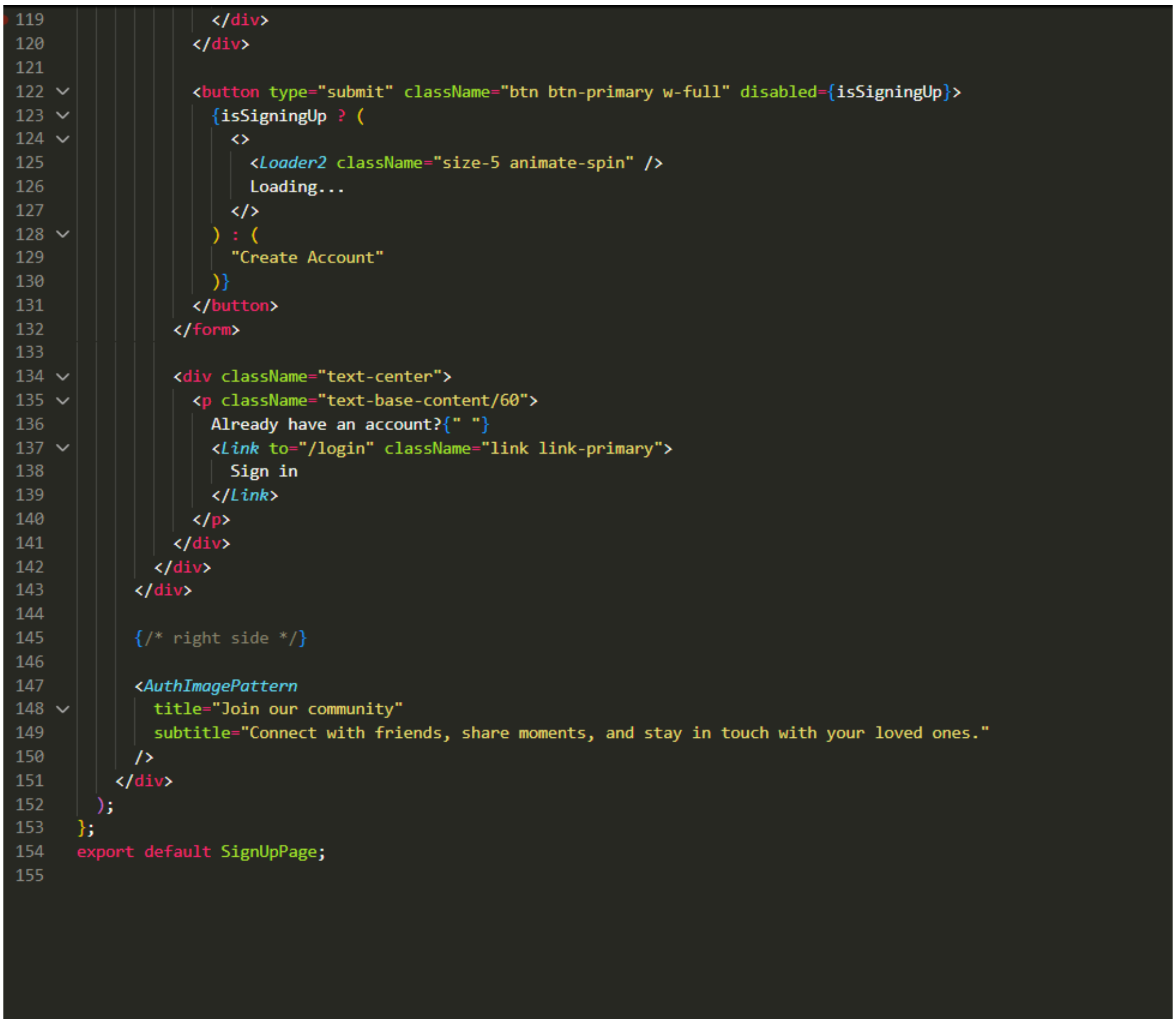
```
import { useState } from "react";
import { useAuthStore } from "../store/useAuthStore";
import { Eye, EyeOff, Loader2, Lock, Mail, MessageSquare, User } from "lucide-react";
import { Link } from "react-router-dom";

import AuthImagePattern from "../components/AuthImagePattern";
import toast from "react-hot-toast";

const SignUpPage = () => {
  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 = (e) => {
    e.preventDefault();

    const success = validateForm();

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

  return (
    <div className="min-h-screen grid lg:grid-cols-2">
      {/* left side */}
      <div className="flex flex-col justify-center items-center p-6 sm:p-12">
        <div className="w-full max-w-md space-y-8">
          {/* LOGO */}
          <div className="text-center mb-8">
            <div className="flex flex-col items-center gap-2 group">
              <div
```



```jsx
      className="size-12 rounded-xl bg-primary/10 flex items-center justify-center
    group-hover:bg-primary/20 transition-colors"
    >
      <MessageSquare className="size-6 text-primary" />
    </div>
    <h1 className="text-2xl font-bold mt-2">Create Account</h1>
    <p className="text-base-content/60">Get started with your free account</p>
  </div>
</div>

<form onSubmit={handleSubmit} className="space-y-6">
  <div className="form-control">
    <label className="label">
      <span className="label-text font-medium">Full Name</span>
    </label>
    <div className="relative">
      <div className="absolute inset-y-0 left-0 pl-3 flex items-center pointer-events-none">
        <User className="size-5 text-base-content/40" />
      </div>
      <input
        type="text"
        className={`input input-bordered w-full pl-10`}
        placeholder="John Doe"
        value={formData.fullName}
        onChange={(e) => setFormData({ ...formData, fullName: e.target.value })}
      />
    </div>
  </div>

  <div className="form-control">
    <label className="label">
      <span className="label-text font-medium">Email</span>
    </label>
    <div className="relative">
      <div className="absolute inset-y-0 left-0 pl-3 flex items-center pointer-events-none">
        <Mail className="size-5 text-base-content/40" />
      </div>
      <input
        type="email"
        className={`input input-bordered w-full pl-10`}
        placeholder="you@example.com"
```



```jsx
        placeholder="you@example.com"
        value={formData.email}
        onChange={(e) => setFormData({ ...formData, email: e.target.value })}
      />
    </div>
  </div>

  <div className="form-control">
    <label className="label">
      <span className="label-text font-medium">Password</span>
    </label>
    <div className="relative">
      <div className="absolute inset-y-0 left-0 pl-3 flex items-center pointer-events-none">
        <Lock className="size-5 text-base-content/40" />
      </div>
      <input
        type={showPassword ? "text" : "password"}
        className={`input input-bordered w-full pl-10`}
        placeholder="••••••••"
        value={formData.password}
        onChange={(e) => setFormData({ ...formData, password: e.target.value })}
      />
      <button
        type="button"
        className="absolute inset-y-0 right-0 pr-3 flex items-center"
        onClick={() => setShowPassword(!showPassword)}
      >
        {showPassword ? (
          <EyeOff className="size-5 text-base-content/40" />
        ) : (
          <Eye className="size-5 text-base-content/40" />
        )}
      </button>
    </div>
  </div>

  <button type="submit" className="btn btn-primary w-full" disabled={isSigningUp}>
    {isSigningUp ? (
      <>
        <Loader2 className="size-5 animate-spin" />
        Loading...
      </>
```


```
                </div>
              </div>

              <button type="submit" className="btn btn-primary w-full" disabled={isSigningUp}>
                {isSigningUp ? (
                  <>
                    <Loader2 className="size-5 animate-spin" />
                    Loading...
                  </>
                ) : (
                  "Create Account"
                )}
              </button>
            </form>

            <div className="text-center">
              <p className="text-base-content/60">
                Already have an account?{" "}
                <Link to="/login" className="link link-primary">
                  Sign in
                </Link>
              </p>
            </div>
          </div>
        </div>

        {/* right side */}

        <AuthImagePattern
          title="Join our community"
          subtitle="Connect with friends, share moments, and stay in touch with your loved ones."
        />
      </div>
    );
};
export default SignUpPage;
```