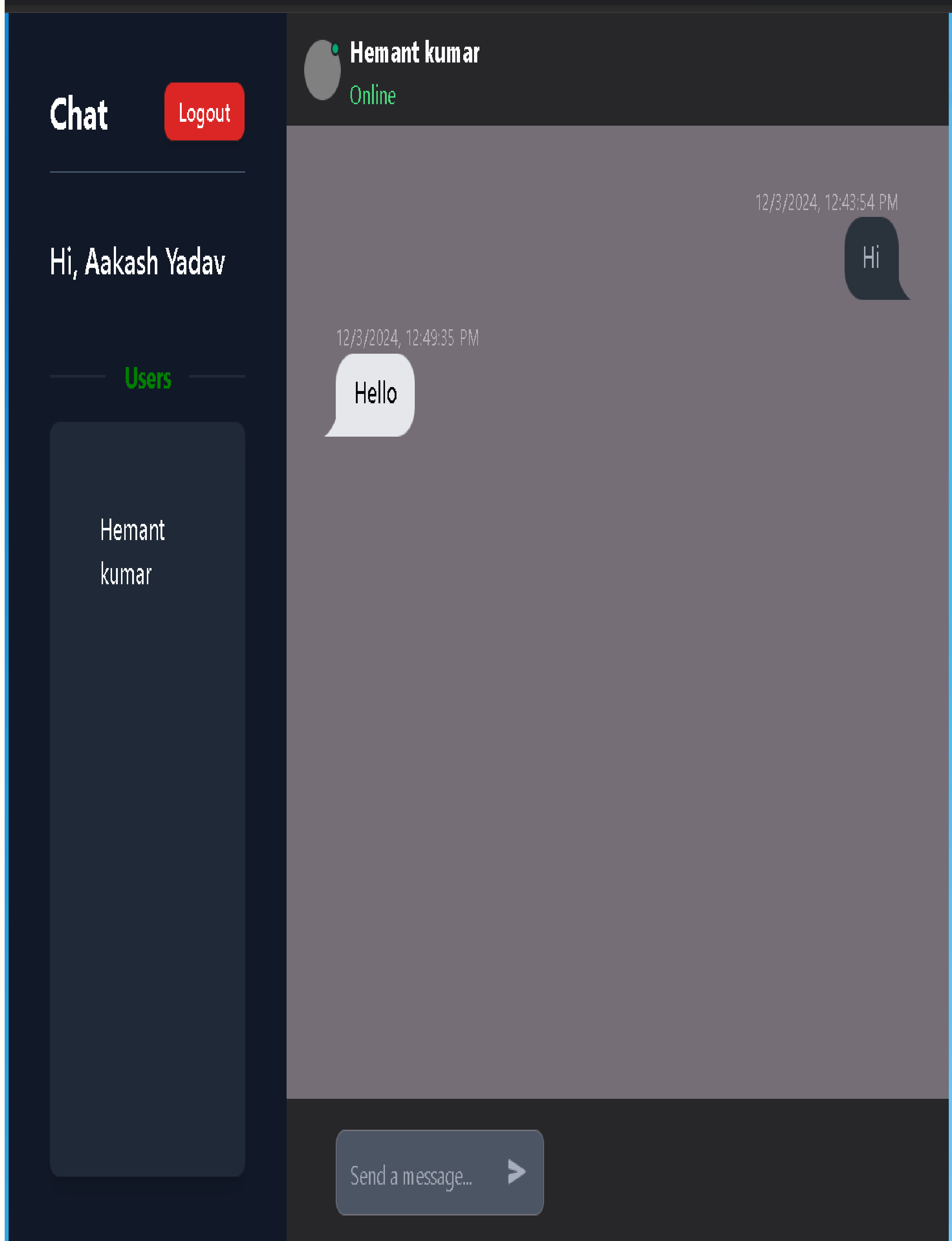
MERN Chat App: Building a Real-time Communication Platform

Welcome to this presentation exploring the development of a real-time chat application built with the MERN stack and Socket.IO. This project demonstrates the power of these technologies for building interactive and engaging communication platforms. We'll delve into the key features, technology stack, and development process, showcasing a seamless and dynamic chat experience.

HK **by Hemant Kumar**


Chat
Logout
Hi, Aakash Yadav
Users
Hemant kumar
Hemant kumar
Online
12/3/2024, 12:43:54 PM
Hi
12/3/2024, 12:49:35 PM
Hello
Send a message...

Introduction to the Project

The MERN Chat App is a full-stack project designed to provide users with a secure and user-friendly platform for real-time messaging. Built using the popular MERN stack (MongoDB, Express.js, React.js, Node.js) and enhanced with Socket.IO for real-time communication, the app enables private messaging and file sharing.

This project aims to showcase the capabilities of these technologies in creating a dynamic and interactive chat experience. We'll explore the core features, technical architecture, and the development process, emphasizing the benefits of using MERN and Socket.IO for real-time communication applications.

Application Features

User Authentication and Authorization

Secure user accounts with login/signup functionalities. Role-based access control for private messaging and user profiles.

Real-time Messaging

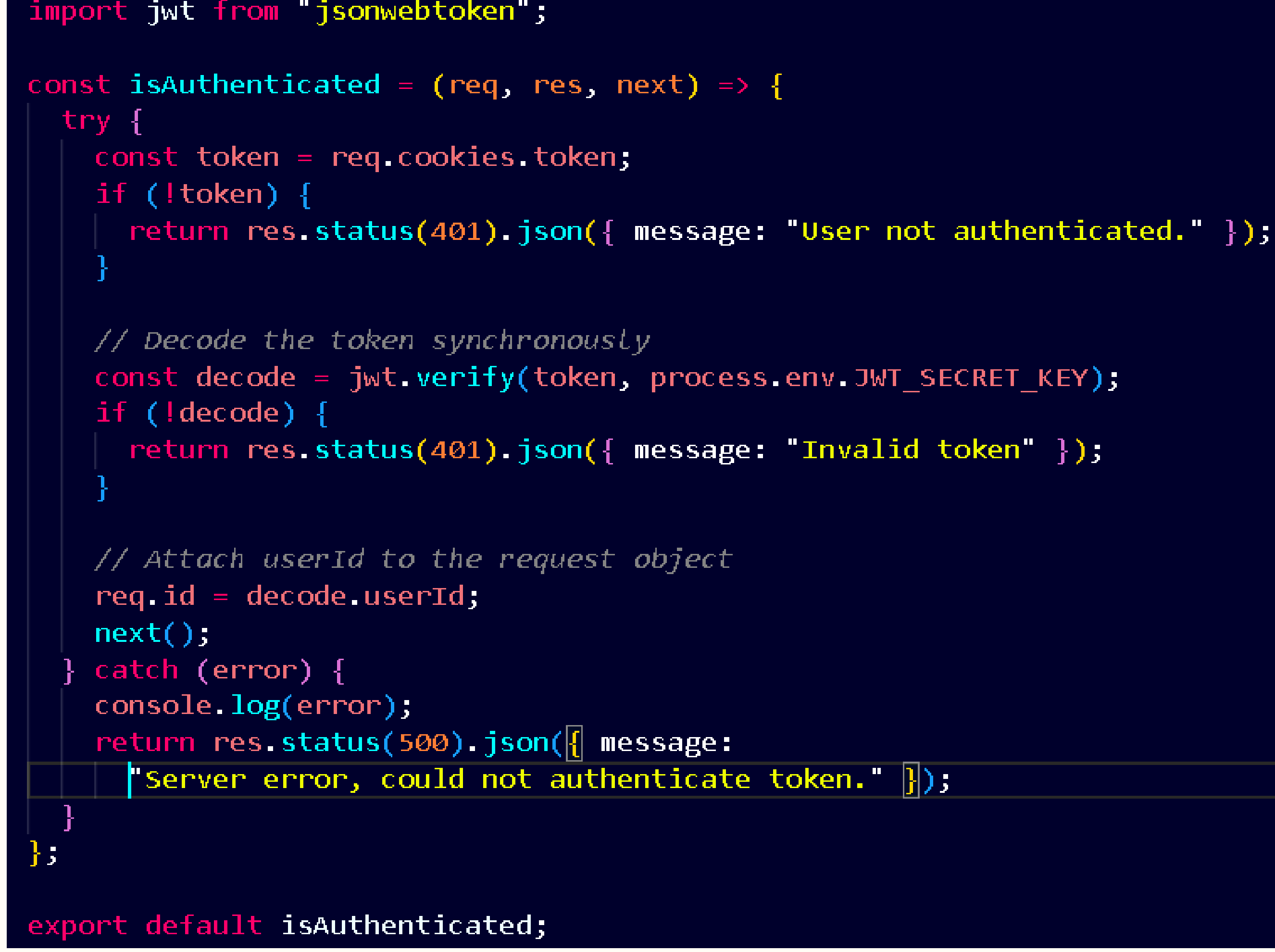
Instant message delivery and updates using Socket.IO. Notification system for new messages and online status.

Private Messaging

Direct communication between users. Sending and receiving private message.

Database Integration

Secure storage of user data, messages, and file uploads using MongoDB. Efficient database design and management.



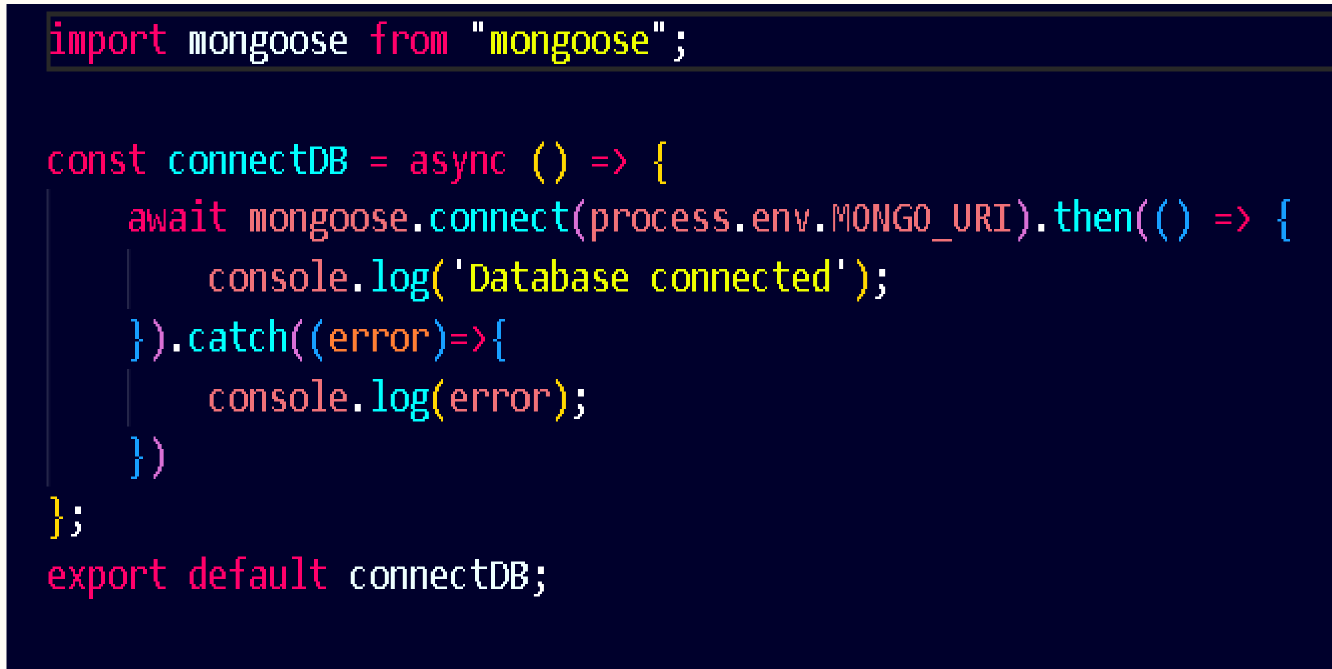
```
import jwt from "jsonwebtoken";

const isAuthenticated = (req, res, next) => {
  try {
    const token = req.cookies.token;
    if (!token) {
      return res.status(401).json({ message: "User not authenticated." });
    }

    // Decode the token synchronously
    const decode = jwt.verify(token, process.env.JWT_SECRET_KEY);
    if (!decode) {
      return res.status(401).json({ message: "Invalid token" });
    }

    // Attach userId to the request object
    req.id = decode.userId;
    next();
  } catch (error) {
    console.log(error);
    return res.status(500).json({ message:
      "Server error, could not authenticate token." });
  }
};

export default isAuthenticated;
```



```
import mongoose from "mongoose";

const connectDB = async () => {
    await mongoose.connect(process.env.MONGO_URI).then(() => {
        console.log('Database connected');
    }).catch((error)=>{
        console.log(error);
    })
};
export default connectDB;
```

Technology Stack

MongoDB

NoSQL database for storing user data, messages, and file uploads.

Express.js

Node.js web application framework for API endpoints and backend logic.

React.js

JavaScript library for building the user interface and handling dynamic content.

Node.js

JavaScript runtime environment for server-side development and backend logic.

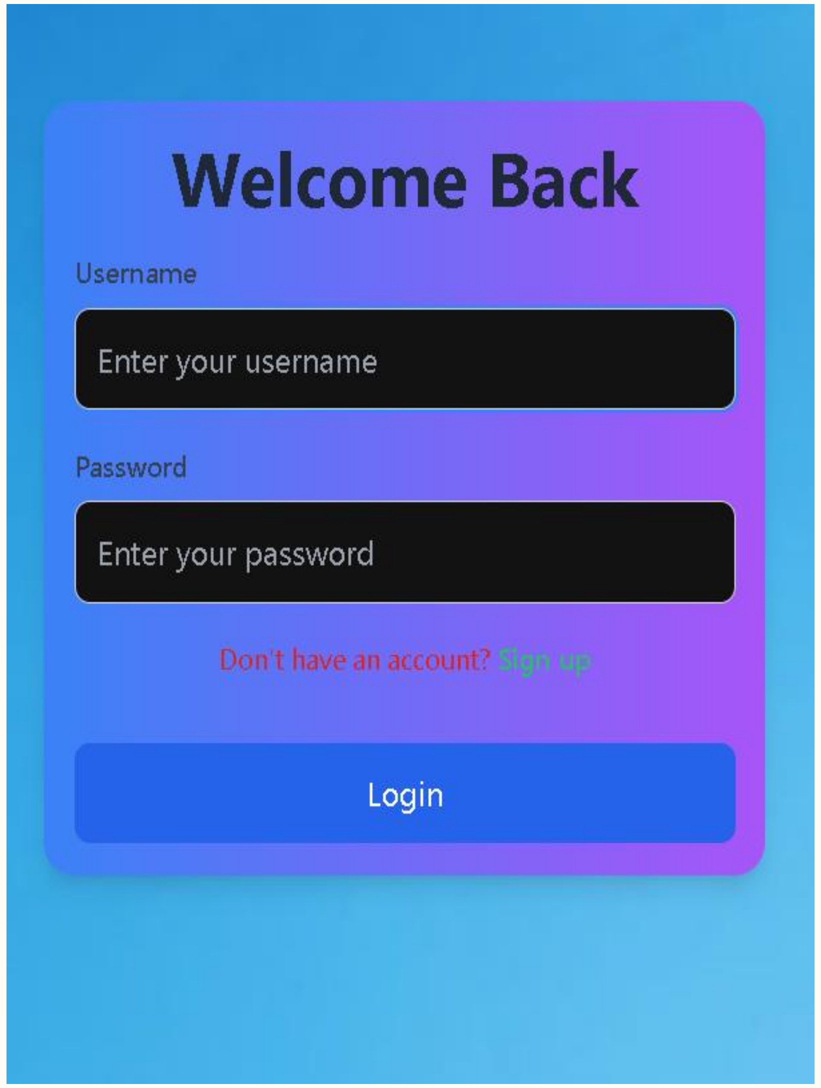
Socket.IO

Real-time communication library for instant message delivery and updates.

User Authentication and Authorization

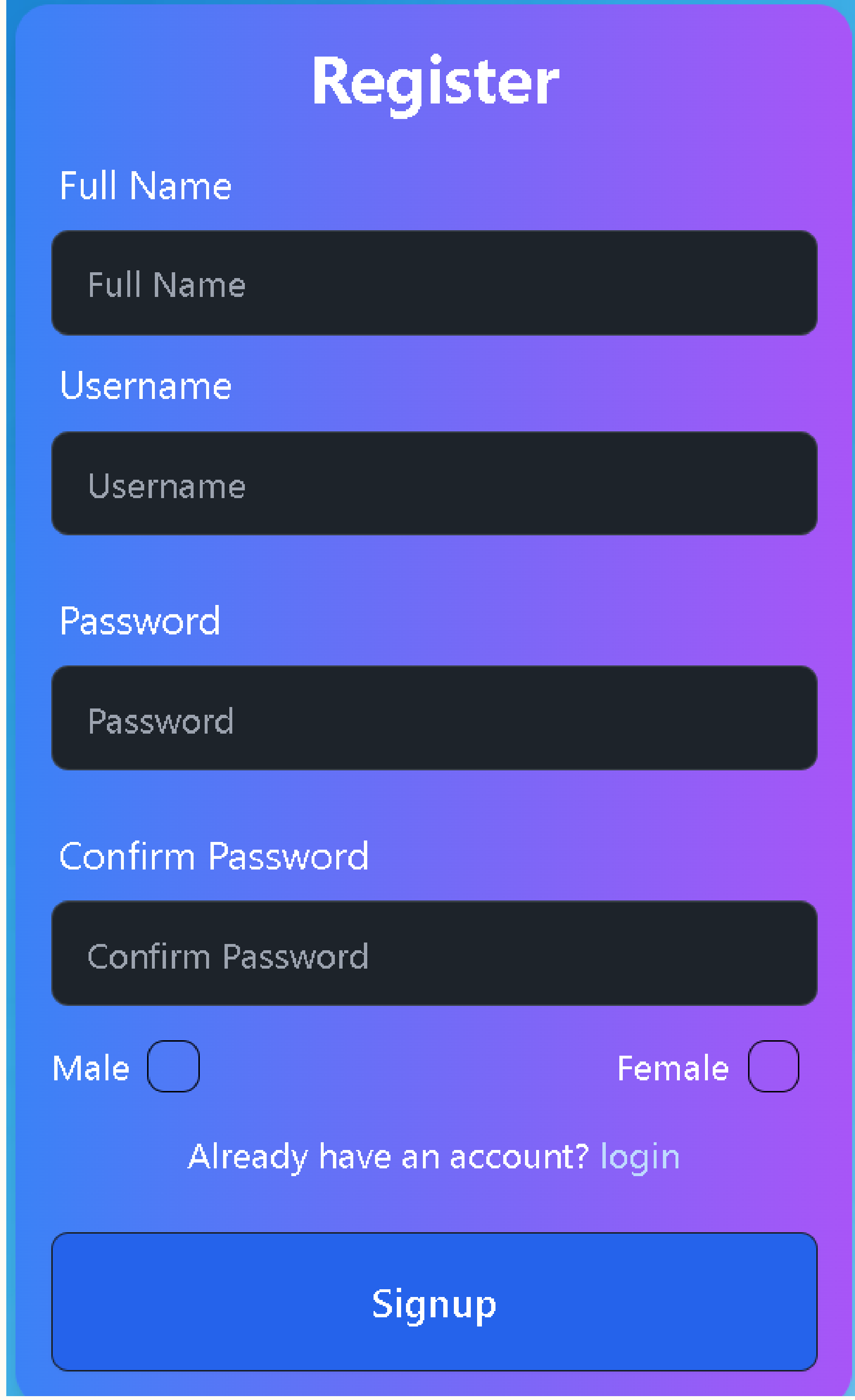
User Registration

A secure registration form allows users to create accounts with unique usernames and passwords. Email verification and password hashing ensure account security.



Login and Logout

Users can log in with their credentials, granting access to private messaging features. Secure logout functionality ensures user data privacy.



Authorization

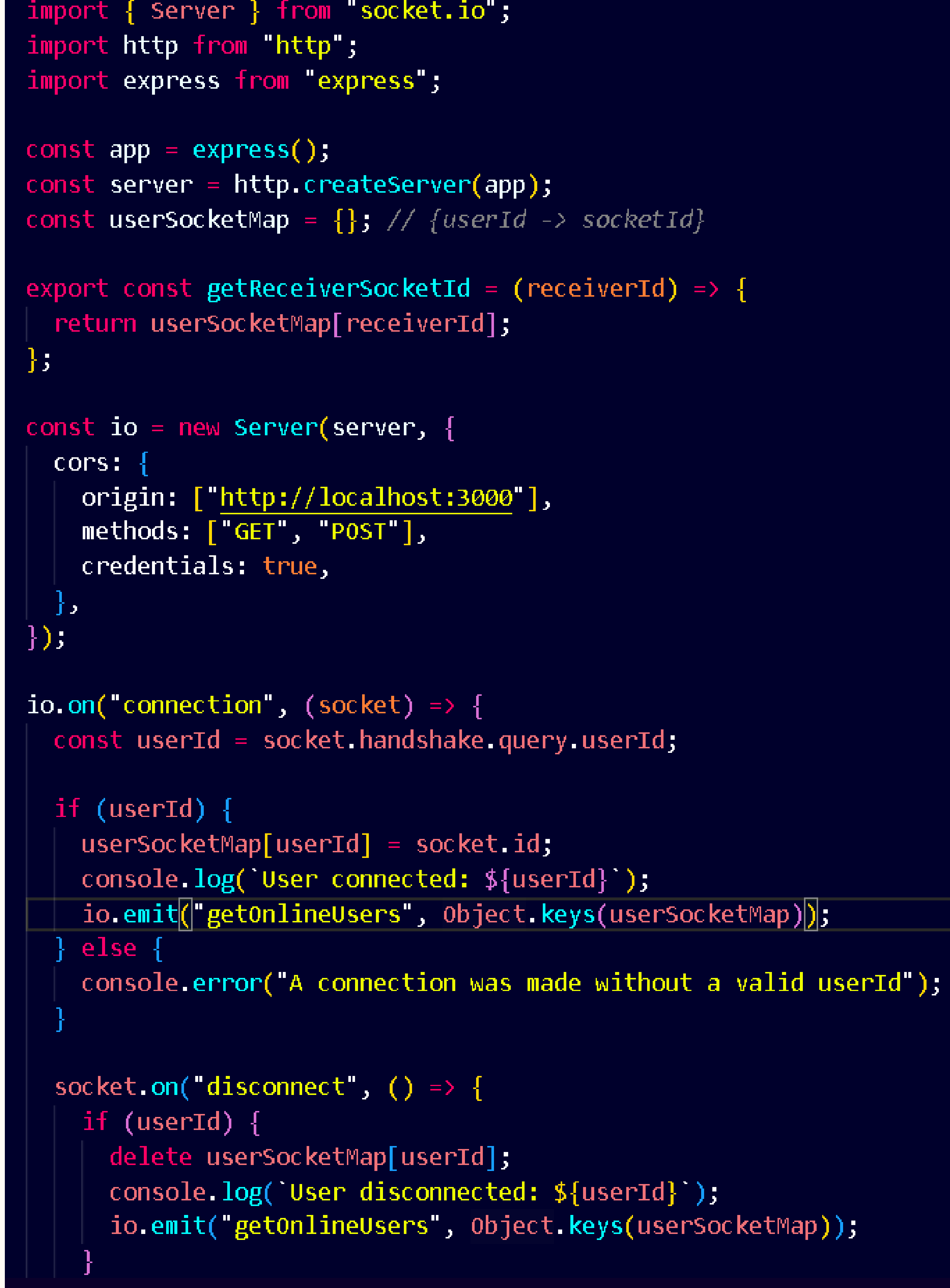
Role-based access control restricts unauthorized access to sensitive data, ensuring privacy and security.

Real-time Messaging with Socket.IO

1 Socket.IO establishes a persistent connection between the client (user's browser) and the server (Node.js application).

2 When a user sends a message, Socket.IO emits an event to the server. This event carries the message content and the recipient's information.

3 The server broadcasts the message to the intended recipient using Socket.IO's event-driven architecture. The recipient's browser receives the message in real-time, updating the chat interface.



```
import { Server } from "socket.io";
import http from "http";
import express from "express";

const app = express();
const server = http.createServer(app);
const userSocketMap = {}; // {userId -> socketId}

export const getReceiverSocketId = (receiverId) => {
  return userSocketMap[receiverId];
};

const io = new Server(server, {
  cors: {
    origin: ["http://localhost:3000"],
    methods: ["GET", "POST"],
    credentials: true,
  },
});

io.on("connection", (socket) => {
  const userId = socket.handshake.query.userId;

  if (userId) {
    userSocketMap[userId] = socket.id;
    console.log(`User connected: ${userId}`);
    io.emit("getOnlineUsers", Object.keys(userSocketMap));
  } else {
    console.error("A connection was made without a valid userId");
  }

  socket.on("disconnect", () => {
    if (userId) {
      delete userSocketMap[userId];
      console.log(`User disconnected: ${userId}`);
      io.emit("getOnlineUsers", Object.keys(userSocketMap));
    }
```

Thank You