VIT®

Vellore Institute of Technology

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

School of Electronics Engineering (SENSE)

PROJECT BASED LEARNING (CAMP) - REPORT

COURSE CODE / NAME	BCSE308L – Computer Networks	
PROGRAM / YEAR	B.Tech (Electronics and Computer Engineering)	
LAST DATE FOR REPORT SUBMISSION	20-11-2024	
DATE OF SUBMISSION	19-11-2024	
TEAM MEMBERS DETAILS	**REGISTER NO.**	**NAME**
	22BLC1290	Meghank kishan
	22BLC1146	Soham Rajendra
	22BLC1171	Krish Sahu
	22BLC1214	Arnav Vyas
PROJECT TITLE	**Build & Deploy a Real-Time Chat Messaging App**	

COURSE HANDLER'S NAME	**Dr. T. Jayavignesh**	**REMARKS**	
COURSE HANDLER'S SIGN			

Table of content:

ABSTRACT:

The emergence of new technologies has brought about significant changes in the way people communicate with each other. One of the most popular ways of communication in today's digital age is through messaging applications. To facilitate this need, several chat applications have been developed. In this thesis, we introduce a chat application built using the MERN stack, which is a popular technology stack used for building web applications. This application provides users with the ability to create accounts, join chat rooms, and send messages to other users in real-time. With the use of web sockets and Angular's two-way binding, the application allows users to see messages as soon as they are sent. Moreover, the application also includes features such as user authentication and authorization to ensure secure access to the chat rooms. Through this project, we aim to demonstrate the feasibility and effectiveness of building real-time chat applications using the Mern stack.

INTRODUCTION:

Chat applications have become an integral part of our day-to-day life and have had a significant impact on how we communicate with each other. With numerous chat applications available in the market, each offering unique features and capabilities, users are spoilt for choice. Companies that develop these applications compete with each other to add new features and improve the user experience with each release. This competition has led to the development of some of the world's top companies, generating high revenue and employing a large number of people.

Our team recognized the need for a reliable and user- friendly chat app that could be used across different platforms and devices, and we decided to build it using the Mern stack. This allowed us to take advantage of the strengths of each technology and create a seamless user experience between the server and the client, allowing for real time updates and notifications.

User authentication was also an important feature of the chat application, and this was implemented using web socket server. WSS is a secure and easy-to use authentication mechanism that allows users to securely transmit information between parties.

ALGORITHM:

Technical explanation of how the code works, flow charts schematics

1. User Authentication
Protocols: HTTPS, OAuth2, OpenID Connect

HTTPS (Hypertext Transfer Protocol Secure):

Purpose: Ensures secure communication between the client (React frontend) and Firebase services during authentication.
How It Works: HTTPS encrypts data sent over the internet using TLS (Transport Layer Security). When a user logs in or signs up, sensitive data like passwords or tokens are encrypted, preventing unauthorized access during transmission.
OAuth3:

Purpose: OAuth2 is a token-based protocol used to grant limited access to a user's resources (e.g., their Firebase profile).

How It Works: When a user logs in with a third-party provider (e.g., Google), Firebase handles OAuth2 behind the scenes, issuing access tokens. These tokens represent the user's session, allowing access to restricted parts of the app without exposing passwords.
OpenID Connect:

Purpose: Extends OAuth2 to add authentication capabilities. OpenID Connect manages user identity by verifying the token’s authenticity.
How It Works: Firebase Authentication uses OpenID Connect to handle user identity verification. When the user logs in, they receive an ID token from the provider, which Firebase uses to confirm the user’s identity.

2. Real-Time Chat Functionality
Protocols: WebSocket Protocol, HTTPS

WebSocket Protocol:

Purpose: WebSocket enables persistent, two-way communication between the client and Firebase (or another server handling chat functionality).
How It Works: WebSocket establishes a single connection that remains open, allowing data to flow both ways in real-time. When a user sends a message, it’s transmitted instantly to the recipient through this continuous connection, which minimizes latency and avoids repeated HTTP requests.
HTTPS:

Purpose: Used to initially load resources and establish a connection to Firebase services.
How It Works: HTTPS is also employed when WebSocket connections are not supported (e.g., on some older networks), where it falls back to long-polling methods. Firebase manages these requests to keep the chat app in sync with real-time updates.

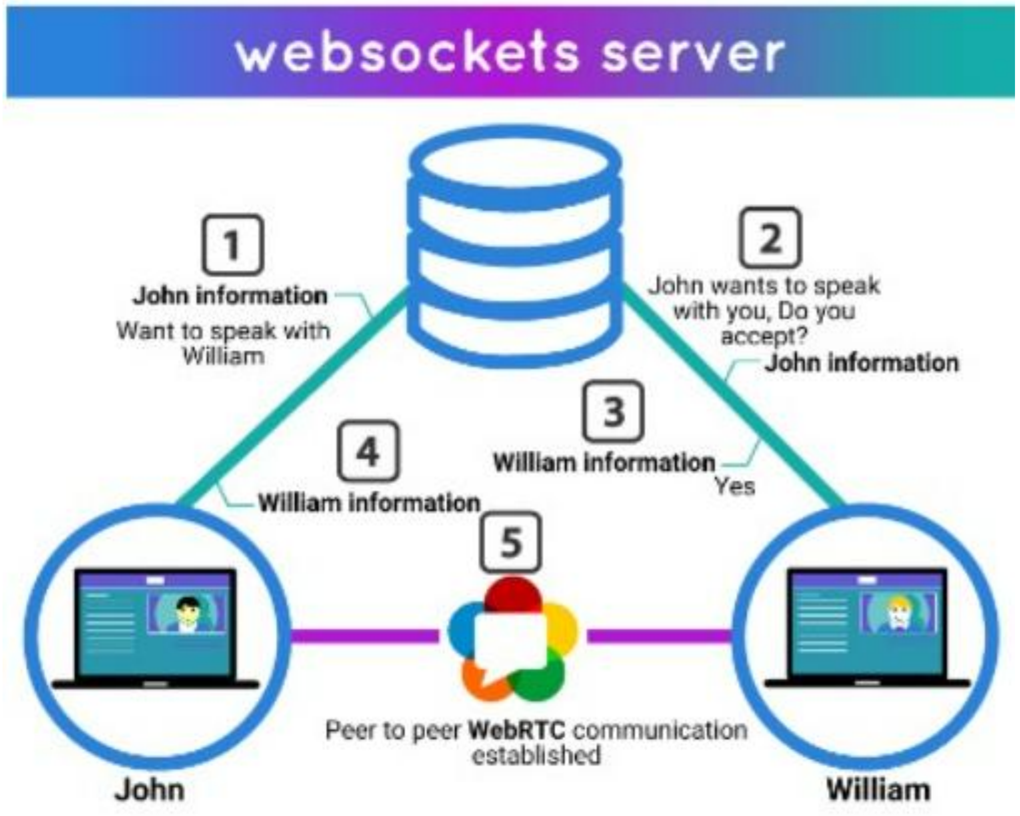
3. Image Sharing
Protocols: HTTPS

HTTPS:
Purpose: Secures the transfer of images between the client and Firebase Storage.
How It Works: When a user uploads an image, the image data is encrypted and sent to Firebase Storage over HTTPS, protecting it from interception. Similarly, when another user accesses the image, HTTPS ensures secure download, preserving user privacy and data integrity.

4. Data Synchronization (Backend Database and Firebase Firestore)
Protocols: WebSocket Protocol, HTTPS

WebSocket Protocol:

Purpose: Keeps chat messages and other data synchronized across devices in real-time.
How It Works: Firebase Firestore or Firebase Realtime Database typically uses WebSocket connections to sync data changes (like new messages or profile updates) instantly. When a user sends a message, it's stored in the database and simultaneously pushed to other connected users via the WebSocket connection.


websockets server
1
John information
Want to speak with William
2
John wants to speak with you, Do you accept?
John information
3
William information
Yes
4
William information
5
Peer to peer WebRTC communication established
John
William

HTTPS:

Purpose: Initial data synchronization and offline caching.
How It Works: When users first log in, they fetch existing chat data over HTTPS. Firebase also uses HTTPS to send data when WebSockets are unavailable, falling back to polling techniques or other methods to sync data.

5. Forgot Password Feature
Protocols: HTTPS, Email Protocols (SMTP, IMAP/POP3)

HTTPS:

Purpose: Provides a secure channel for requesting a password reset and confirming the reset action.
How It Works: When a user initiates a password reset, Firebase Authentication sends a secure link to their registered email. The reset request, link generation, and

confirmation of the reset are all done over HTTPS to ensure data privacy and prevent unauthorized access to the reset link.
Email Protocols (SMTP, IMAP, POP3):

Purpose: Enables Firebase to deliver the password reset email to the user's inbox.
How It Works: Firebase Authentication uses Simple Mail Transfer Protocol (SMTP) to send the password reset email, containing a secure link for resetting the password. When the user clicks the link, they are redirected to a secure Firebase-hosted page where they can enter a new password. IMAP or POP3 protocols might be used by the user's email client to retrieve this reset email.
How the Forgot Password Feature Works with Firebase
User Request: When a user clicks "Forgot Password" in your app, Firebase's sendPasswordResetEmail method is triggered, sending a password reset request to Firebase Authentication.

Reset Link Generation: Firebase generates a unique, time-limited reset link, which is only usable once. This link is encrypted and can only be accessed by the user who received the email.

Secure Delivery: Firebase sends the reset link to the user's registered email address using SMTP. This email protocol ensures that the reset message reaches the correct inbox securely.

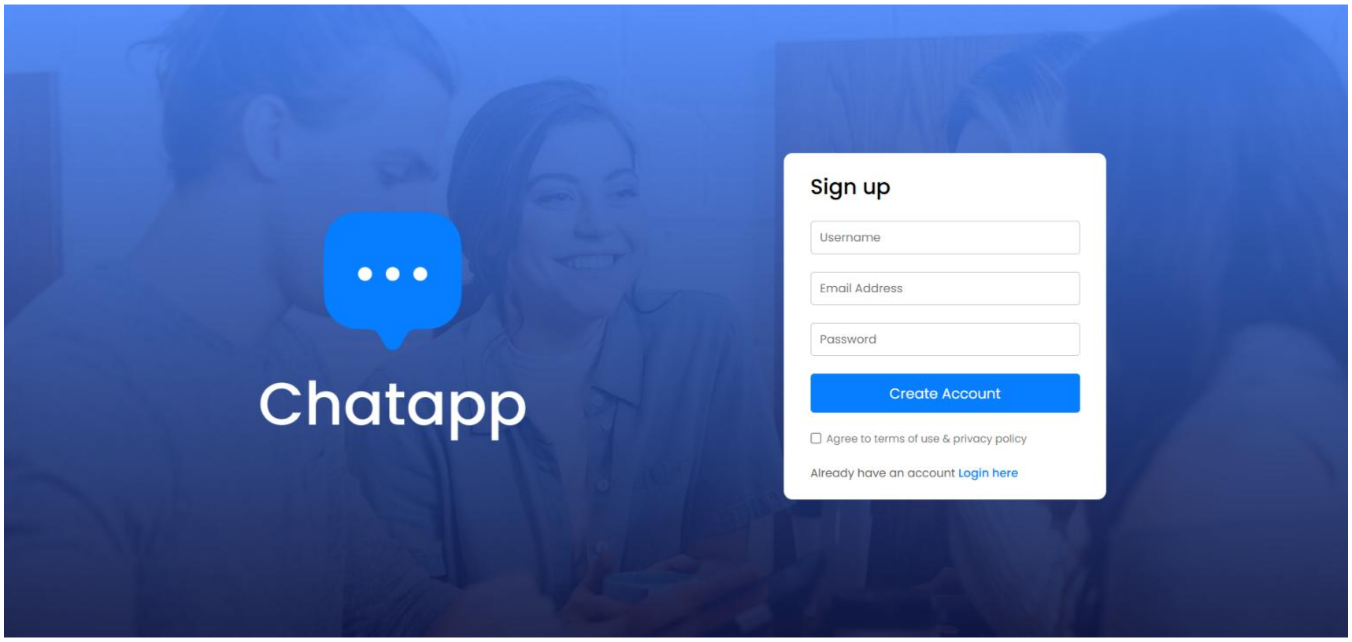
Password Reset Confirmation: The user clicks the reset link, which redirects them to a secure HTTPS page hosted by Firebase. Here, they can enter a new password, which Firebase updates in the user's authentication profile.

Security Measures for Forgot Password
One-Time Link: Firebase ensures that the reset link is single-use and time-bound, which minimizes the risk of unauthorized access if the link is intercepted.
TLS Encryption: Both the link and the data entered on the reset page are encrypted via HTTPS, ensuring that the reset password and other sensitive information cannot be intercepted.

IMPLEMENTATION (REAL TIME):

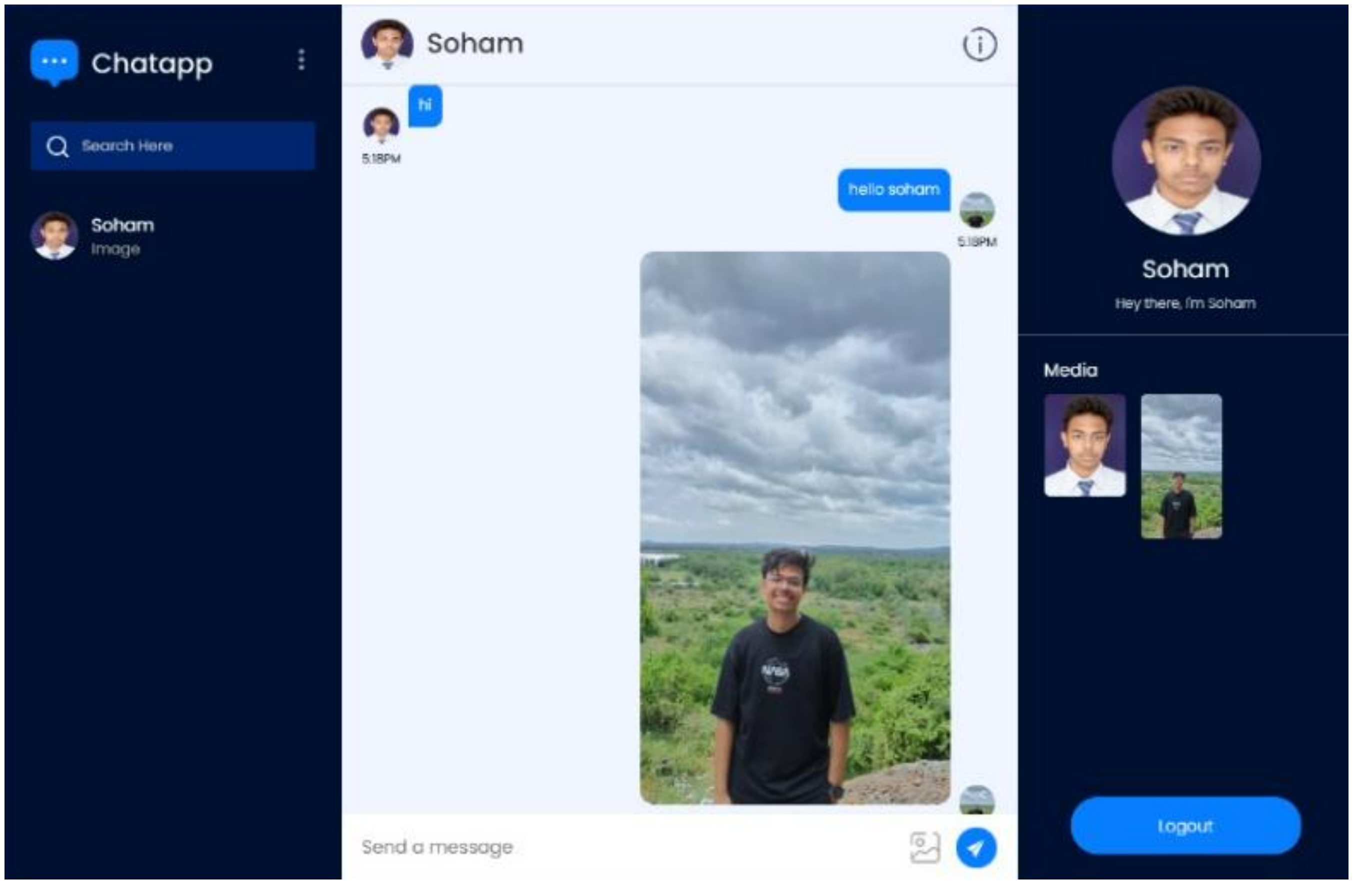
1.Signup

The user must get a unique ID for the account in order to indicate his/her independent existence in accordance with the registration and login features. The user might need to keep track of his/her activity, including his/her usage history, likes, favourite content, etc. Additionally, the user must connect with other users in order to get their concrete facts or detailed information as necessary. Figure 4 represents the view of the registration or sign up page. It asks user to create an account by entering Username, Email address and password followed by a sign up button. It also says if the user already have an account , he/she can click on the login option. The login option will redirect the user to a new page called login page as shown in figure 6. This page allows user to login with username and password. The login page also has an register here option which will redirect the user to registration page.


Chatapp
Sign up
Username
Email Address
Password
Create Account
Agree to terms of use & privacy policy
Already have an account Login here

2. Dashboard

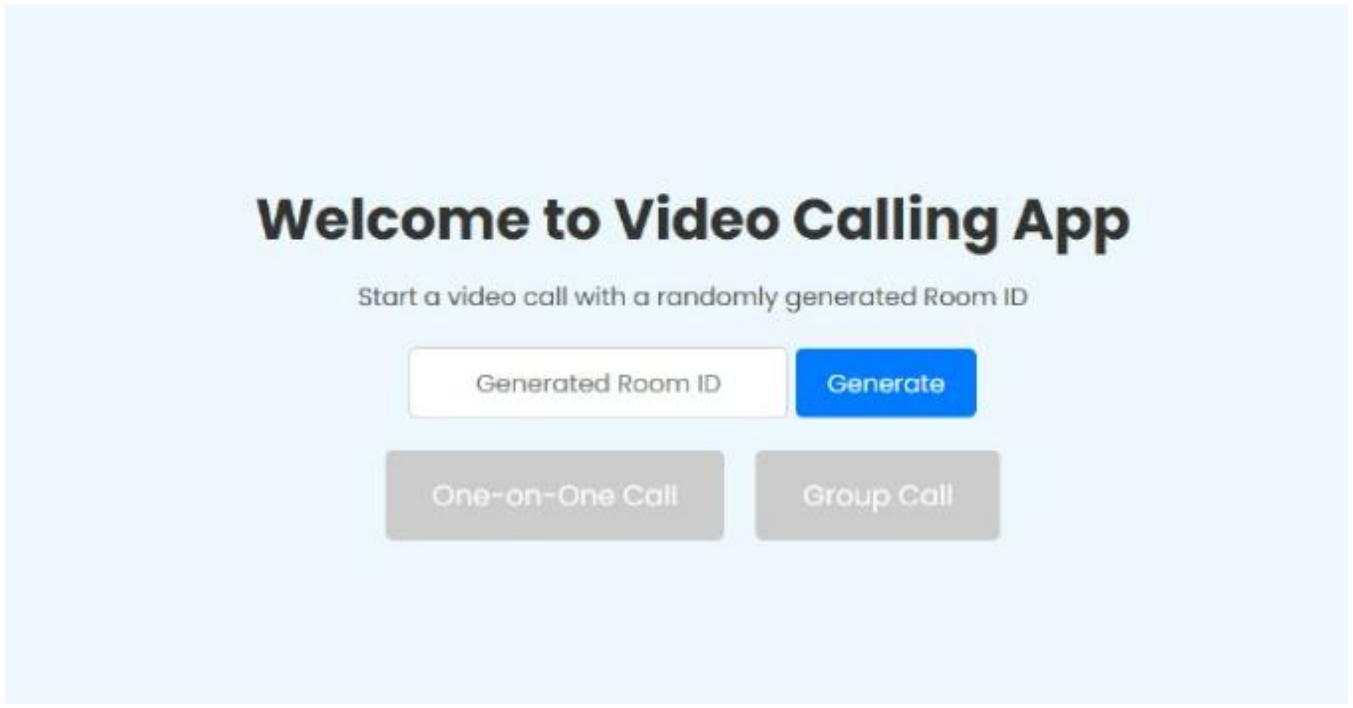
After logging in, the user will be directed to a dashboard . The dashboard has various tabs to indicate various features of the application. A heading "To start chatting- select a friend for conversation" is seen at the center of the display.To select a friend for chatting, there is a list of added friends on the left side of the display.It also shows a green dot to indicate that the friend is online. There is "Add friend" button on the top left corner of the display which allows the user to invite a friend to use this web application. There is a list of private messages and all invitations sent followed by the list of friends. There is a three line menu button at the top right corner of the display which indicates the username of the account and also has an option to logout.



3. Conversation

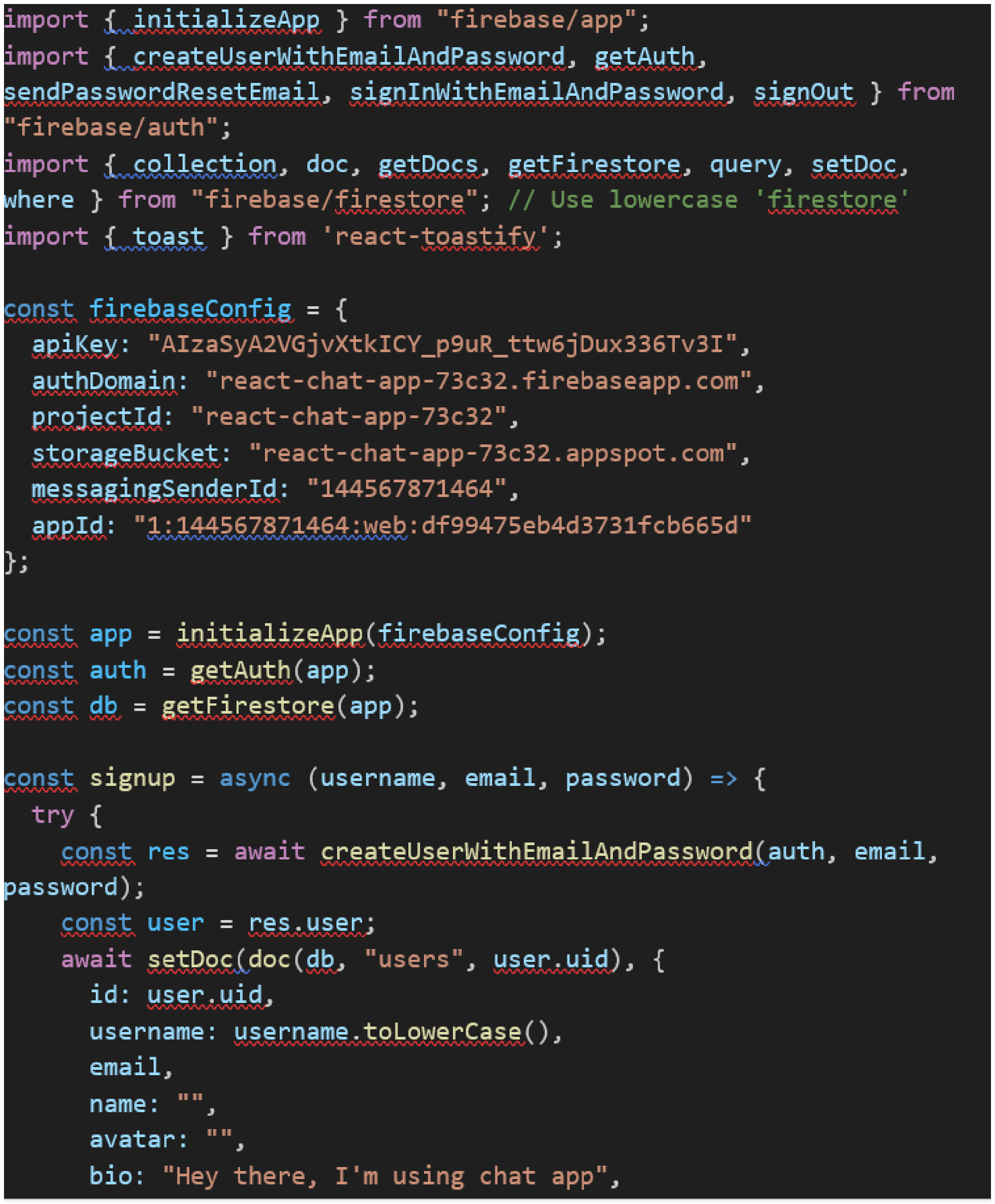
The web application allows user to chat with each other in a private chat . The chat box also notifies when the other user is typing. The chat box separates messages according to date and time. The messages sent are in blue color box theme and the messages received are in white color box theme. Also, the messages also indicate time of receiving and sending.Just like a normal messaging application the user can open a chat and chat further with the person he is friend with. For having the conversation the person must be the friend else he wont be able to send any messages to the person. As shown above the invitation must be sent via email. Thus this way a secure system allows you to chat with people. Futhermore, more features like stickers or gifs could be added to chatterbox for making it similar to application.

4. Audio call, video call :
Our web application has a new feature of screen sharing as shown in figure 10.Sharing screen's contents with another device is referred to as screen sharing or desktop sharing. This gives user total control over how much of their desktop is shown while still ensuring user's privacy. It can encompass just a window or all the objects that are on a screen. User may display friends, employees, or clients any media that is on the device to share screen without ever sending any files; this includes presentations, documents, photographs, and even films. Additionally, the receiver may watch while the user interacts with the shared device in real-time, navigating the interface and making changes, while simultaneously viewing the content on it. This web application also allow user to make audio call as well as video call.

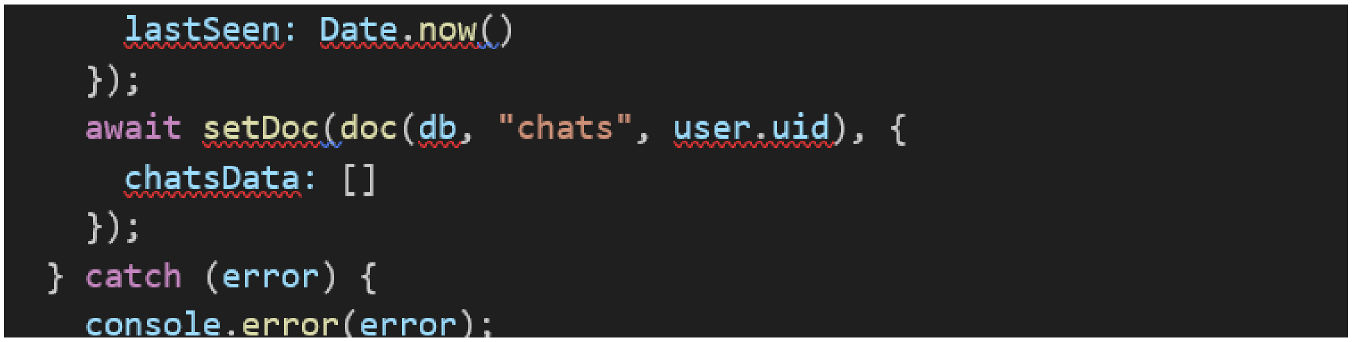

Welcome to Video Calling App
Start a video call with a randomly generated Room ID
Generated Room ID
Generate
One-on-One Call
Group Call

CODING:

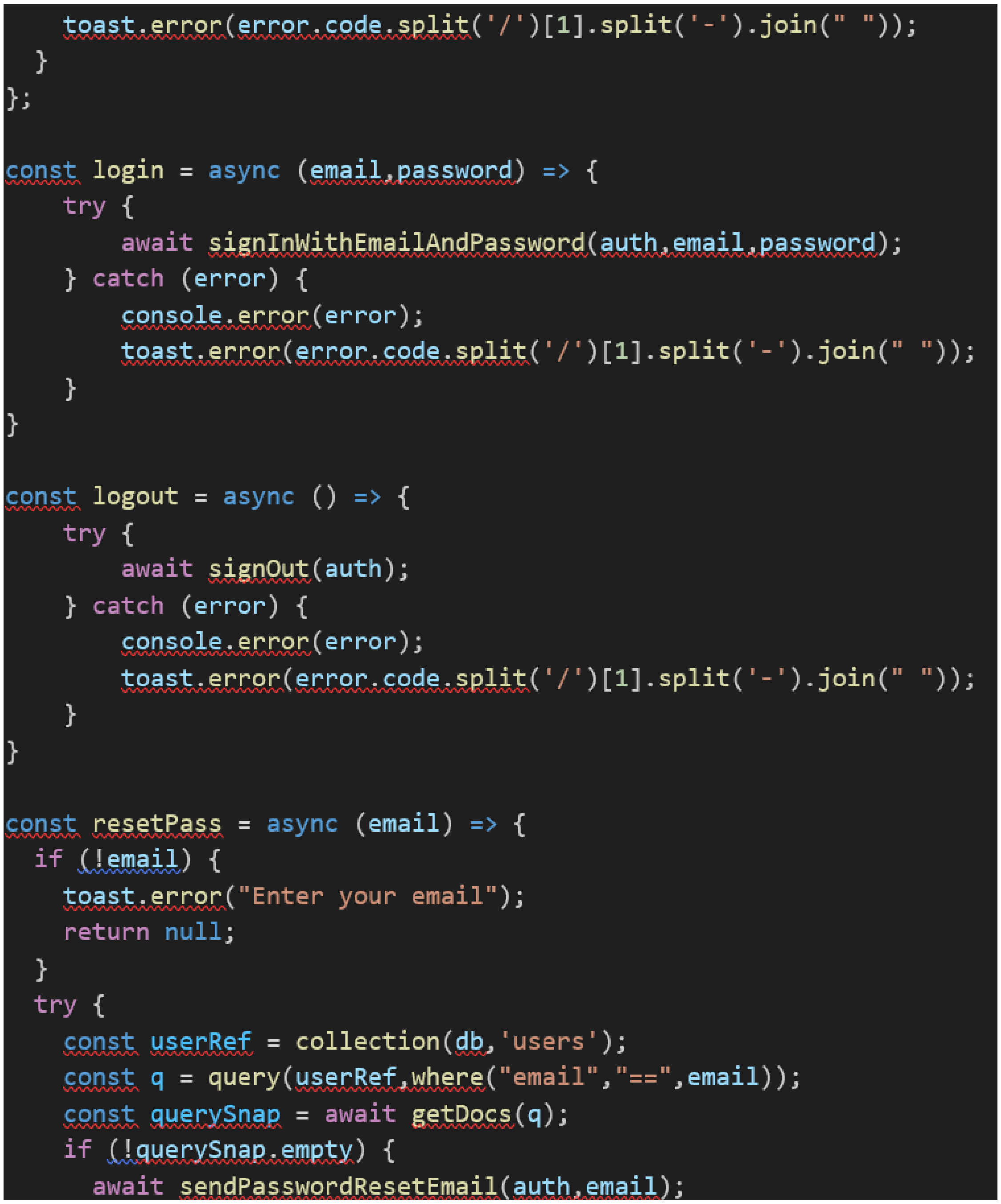
1.Firebase configuaration(user authentication):



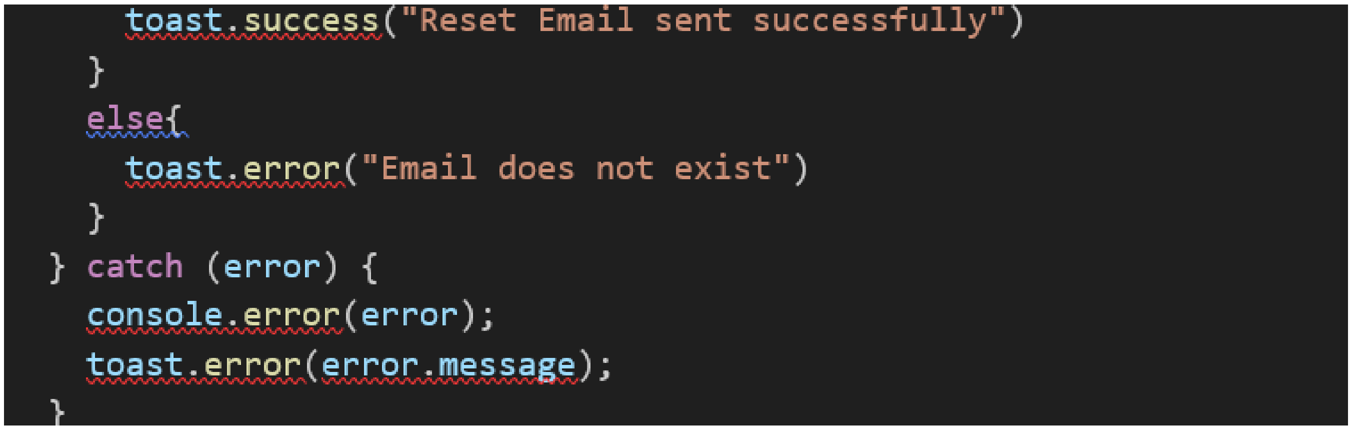
```
import { initializeApp } from "firebase/app";
import { createUserWithEmailAndPassword, getAuth,
sendPasswordResetEmail, signInWithEmailAndPassword, signOut } from
"firebase/auth";
import { collection, doc, getDocs, getFirestore, query, setDoc,
where } from "firebase/firestore"; // Use lowercase 'firestore'
import { toast } from 'react-toastify';

const firebaseConfig = {
  apiKey: "AIzaSyA2VGjvXtkICY_p9uR_ttw6jDux336Tv3I",
  authDomain: "react-chat-app-73c32.firebaseapp.com",
  projectId: "react-chat-app-73c32",
  storageBucket: "react-chat-app-73c32.appspot.com",
  messagingSenderId: "144567871464",
  appId: "1:144567871464:web:df99475eb4d3731fcb665d"
};

const app = initializeApp(firebaseConfig);
const auth = getAuth(app);
const db = getFirestore(app);

const signup = async (username, email, password) => {
  try {
    const res = await createUserWithEmailAndPassword(auth, email,
password);
    const user = res.user;
    await setDoc(doc(db, "users", user.uid), {
      id: user.uid,
      username: username.toLowerCase(),
      email,
      name: "",
      avatar: "",
      bio: "Hey there, I'm using chat app",
```



```
      lastSeen: Date.now()
    });
    await setDoc(doc(db, "chats", user.uid), {
      chatsData: []
    });
  } catch (error) {
    console.error(error);
```

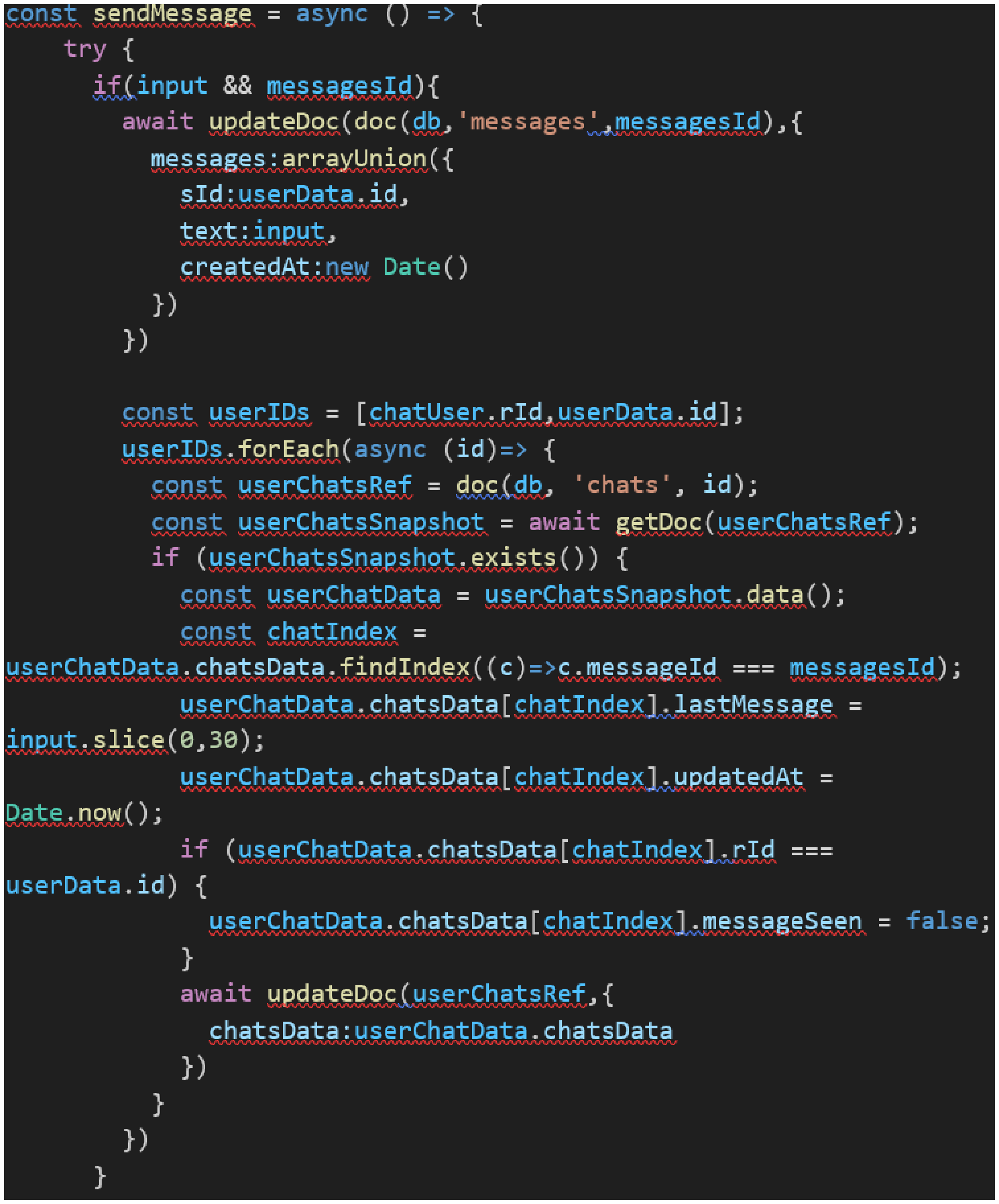


```
    toast.error(error.code.split('/')[1].split('-').join(" "));
  }
};

const login = async (email,password) => {
    try {
        await signInWithEmailAndPassword(auth,email,password);
    } catch (error) {
        console.error(error);
        toast.error(error.code.split('/')[1].split('-').join(" "));
    }
}

const logout = async () => {
    try {
        await signOut(auth);
    } catch (error) {
        console.error(error);
        toast.error(error.code.split('/')[1].split('-').join(" "));
    }
}

const resetPass = async (email) => {
  if (!email) {
    toast.error("Enter your email");
    return null;
  }
  try {
    const userRef = collection(db,'users');
    const q = query(userRef,where("email","==",email));
    const querySnap = await getDocs(q);
    if (!querySnap.empty) {
      await sendPasswordResetEmail(auth,email);
```

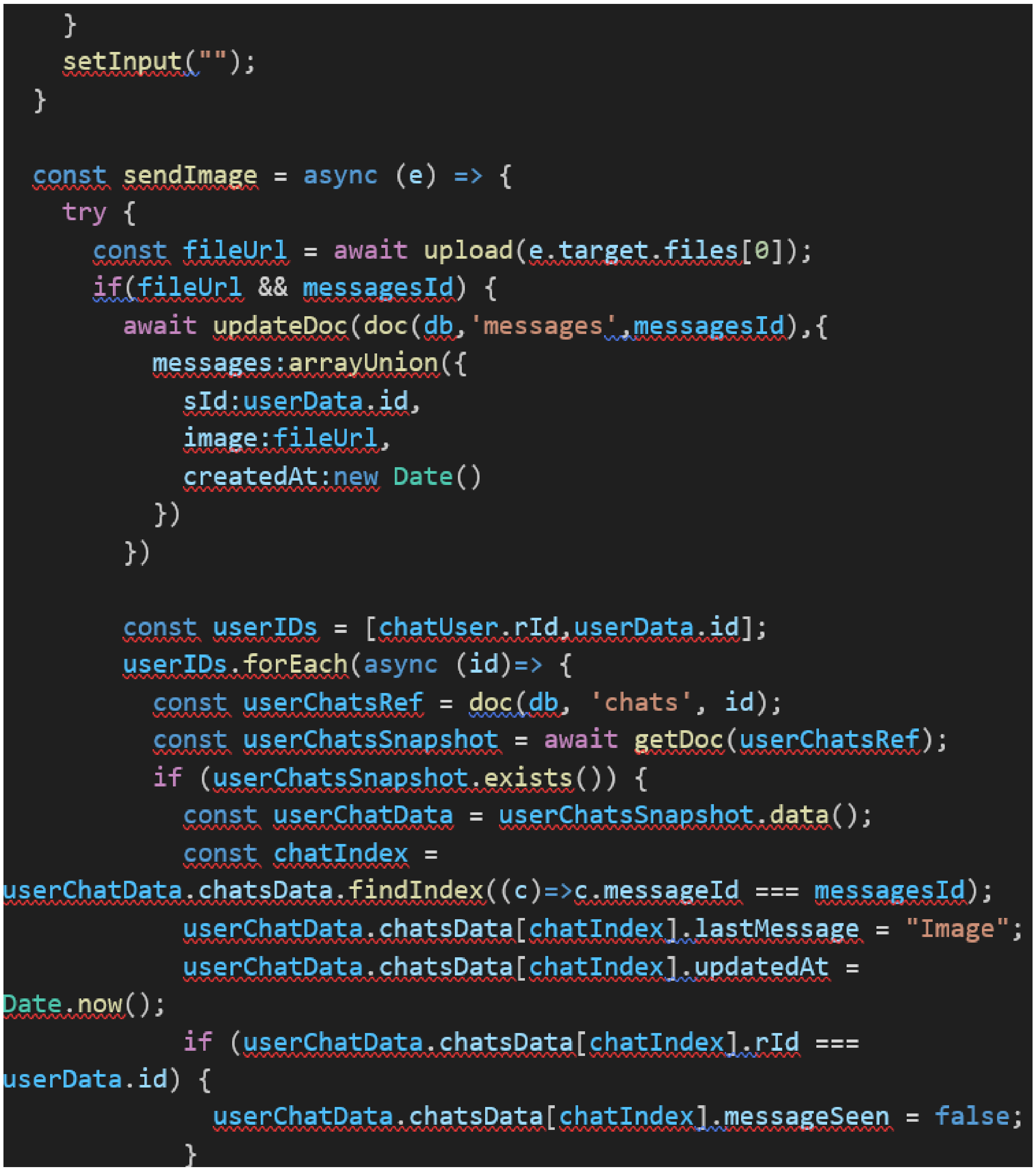


```
      toast.success("Reset Email sent successfully")
    }
    else{
      toast.error("Email does not exist")
    }
  } catch (error) {
    console.error(error);
    toast.error(error.message);
  }
```

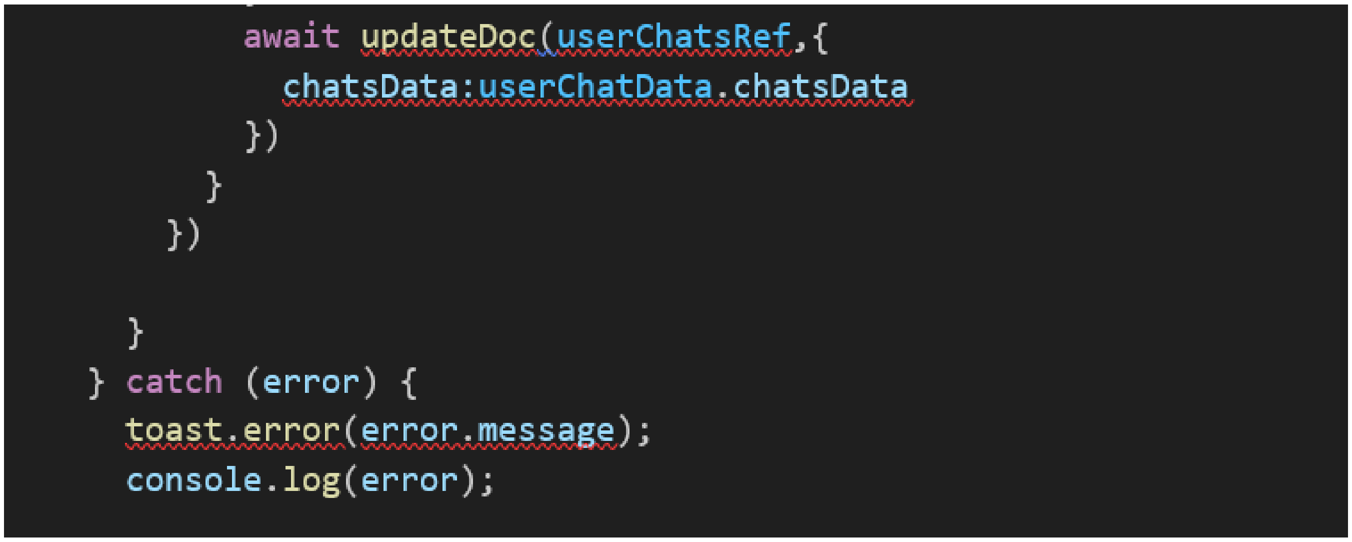
2.Chat and image sharing function:



```
const sendMessage = async () => {
    try {
      if(input && messagesId){
        await updateDoc(doc(db,'messages',messagesId),{
          messages:arrayUnion({
            sId:userData.id,
            text:input,
            createdAt:new Date()
          })
        })

        const userIDs = [chatUser.rId,userData.id];
        userIDs.forEach(async (id)=> {
          const userChatsRef = doc(db, 'chats', id);
          const userChatsSnapshot = await getDoc(userChatsRef);
          if (userChatsSnapshot.exists()) {
            const userChatData = userChatsSnapshot.data();
            const chatIndex =
userChatData.chatsData.findIndex((c)=>c.messageId === messagesId);
            userChatData.chatsData[chatIndex].lastMessage =
input.slice(0,30);
            userChatData.chatsData[chatIndex].updatedAt =
Date.now();
            if (userChatData.chatsData[chatIndex].rId ===
userData.id) {
              userChatData.chatsData[chatIndex].messageSeen = false;
            }
            await updateDoc(userChatsRef,{
              chatsData:userChatData.chatsData
            })
          }
        })
      }
```

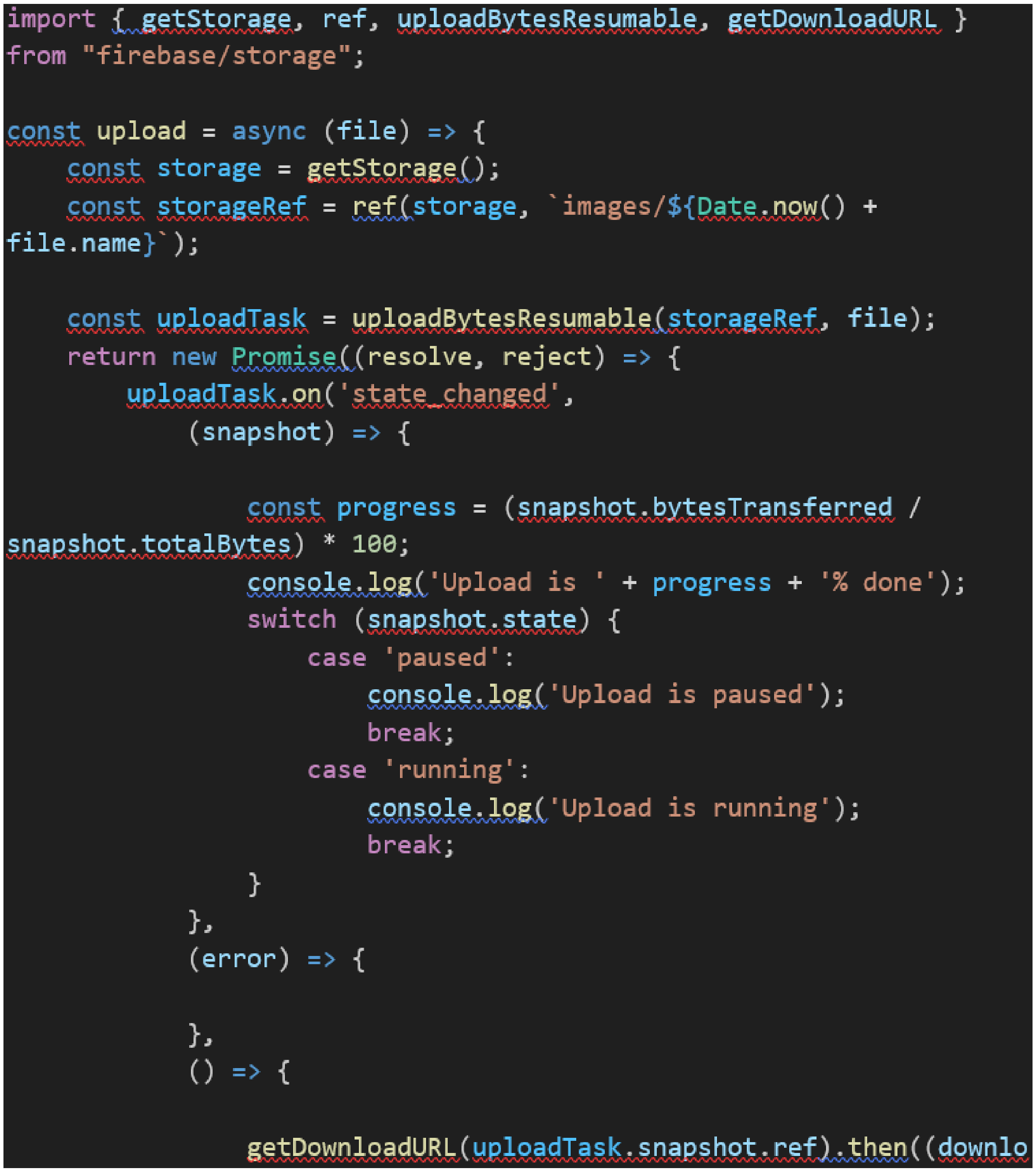


```
    }
    setInput("");
  }

  const sendImage = async (e) => {
    try {
      const fileUrl = await upload(e.target.files[0]);
      if(fileUrl && messagesId) {
        await updateDoc(doc(db,'messages',messagesId),{
          messages:arrayUnion({
            sId:userData.id,
            image:fileUrl,
            createdAt:new Date()
          })
        })

        const userIDs = [chatUser.rId,userData.id];
        userIDs.forEach(async (id)=> {
          const userChatsRef = doc(db, 'chats', id);
          const userChatsSnapshot = await getDoc(userChatsRef);
          if (userChatsSnapshot.exists()) {
            const userChatData = userChatsSnapshot.data();
            const chatIndex =
userChatData.chatsData.findIndex((c)=>c.messageId === messagesId);
            userChatData.chatsData[chatIndex].lastMessage = "Image";
            userChatData.chatsData[chatIndex].updatedAt =
Date.now();
            if (userChatData.chatsData[chatIndex].rId ===
userData.id) {
              userChatData.chatsData[chatIndex].messageSeen = false;
            }
```

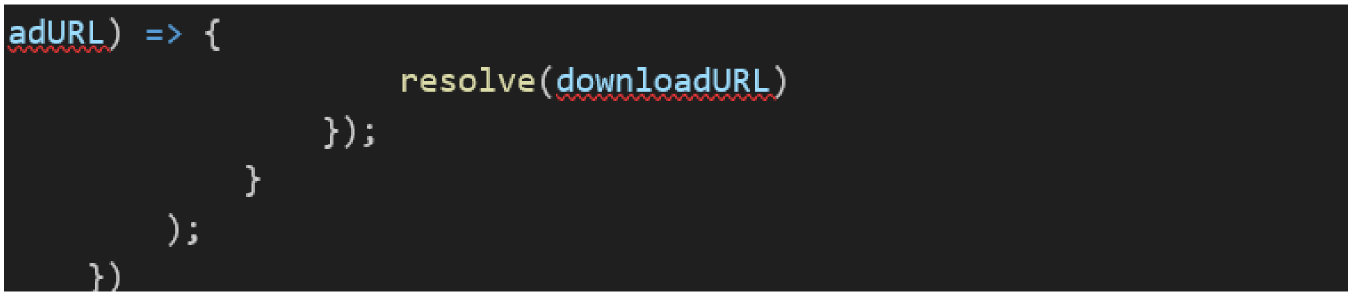


```
            await updateDoc(userChatsRef,{
              chatsData:userChatData.chatsData
            })
          }
        })

      }
    } catch (error) {
      toast.error(error.message);
      console.log(error);
```

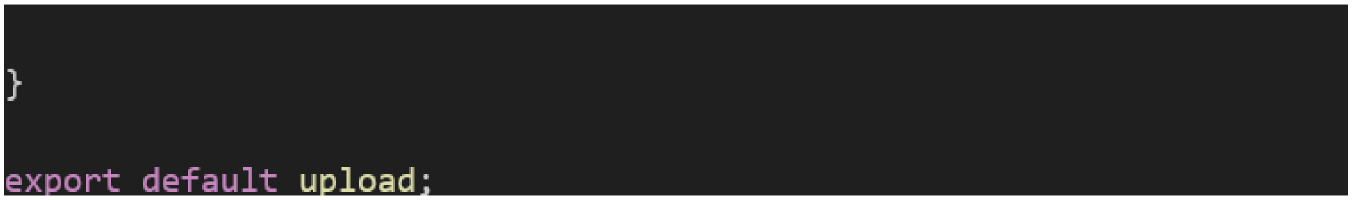
3. Photo upload



```
import { getStorage, ref, uploadBytesResumable, getDownloadURL }
from "firebase/storage";

const upload = async (file) => {
    const storage = getStorage();
    const storageRef = ref(storage, `images/${Date.now() +
file.name}`);

    const uploadTask = uploadBytesResumable(storageRef, file);
    return new Promise((resolve, reject) => {
        uploadTask.on('state_changed',
            (snapshot) => {

                const progress = (snapshot.bytesTransferred /
snapshot.totalBytes) * 100;
                console.log('Upload is ' + progress + '% done');
                switch (snapshot.state) {
                    case 'paused':
                        console.log('Upload is paused');
                        break;
                    case 'running':
                        console.log('Upload is running');
                        break;
                }
            },
            (error) => {

            },
            () => {

                getDownloadURL(uploadTask.snapshot.ref).then((downlo
```

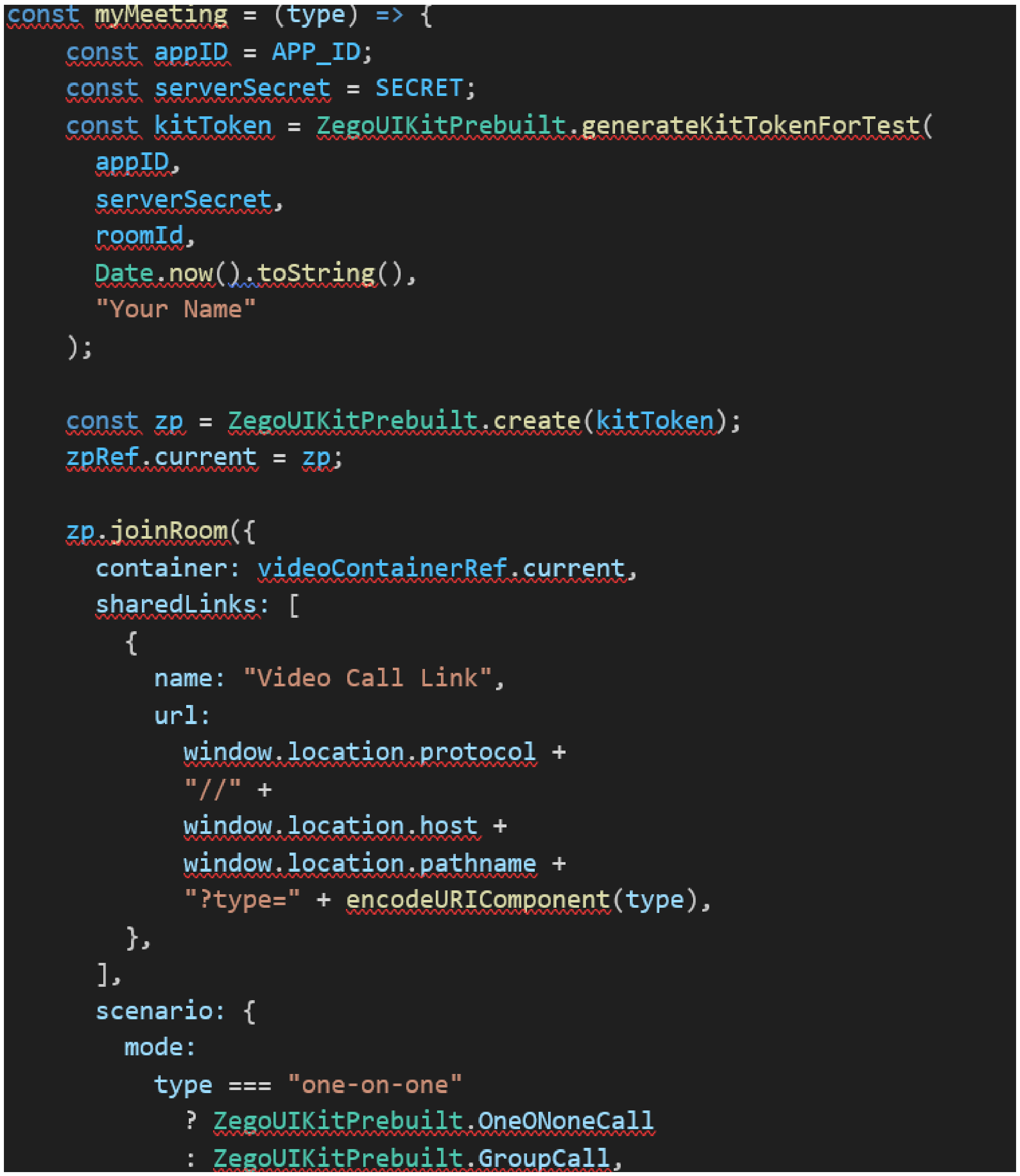


```
adURL) => {
                    resolve(downloadURL)
                });
            }
        );
    })
```

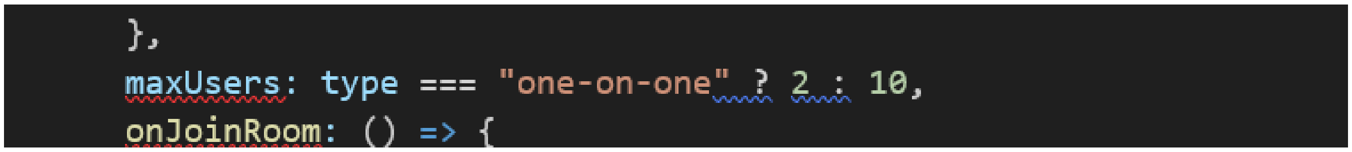


```

}

export default upload;
```

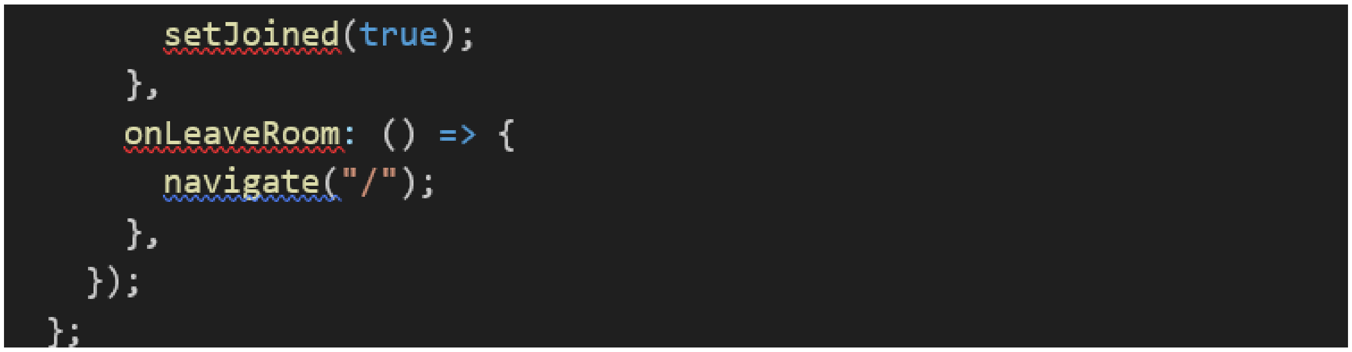
4.Cloudvideocall config functions:



```
const myMeeting = (type) => {
    const appID = APP_ID;
    const serverSecret = SECRET;
    const kitToken = ZegoUIKitPrebuilt.generateKitTokenForTest(
      appID,
      serverSecret,
      roomId,
      Date.now().toString(),
      "Your Name"
    );

    const zp = ZegoUIKitPrebuilt.create(kitToken);
    zpRef.current = zp;

    zp.joinRoom({
      container: videoContainerRef.current,
      sharedLinks: [
        {
          name: "Video Call Link",
          url:
            window.location.protocol +
            "//" +
            window.location.host +
            window.location.pathname +
            "?type=" + encodeURIComponent(type),
        },
      ],
      scenario: {
        mode:
          type === "one-on-one"
            ? ZegoUIKitPrebuilt.OneONoneCall
            : ZegoUIKitPrebuilt.GroupCall,
```



```
    },
    maxUsers: type === "one-on-one" ? 2 : 10,
    onJoinRoom: () => {
```

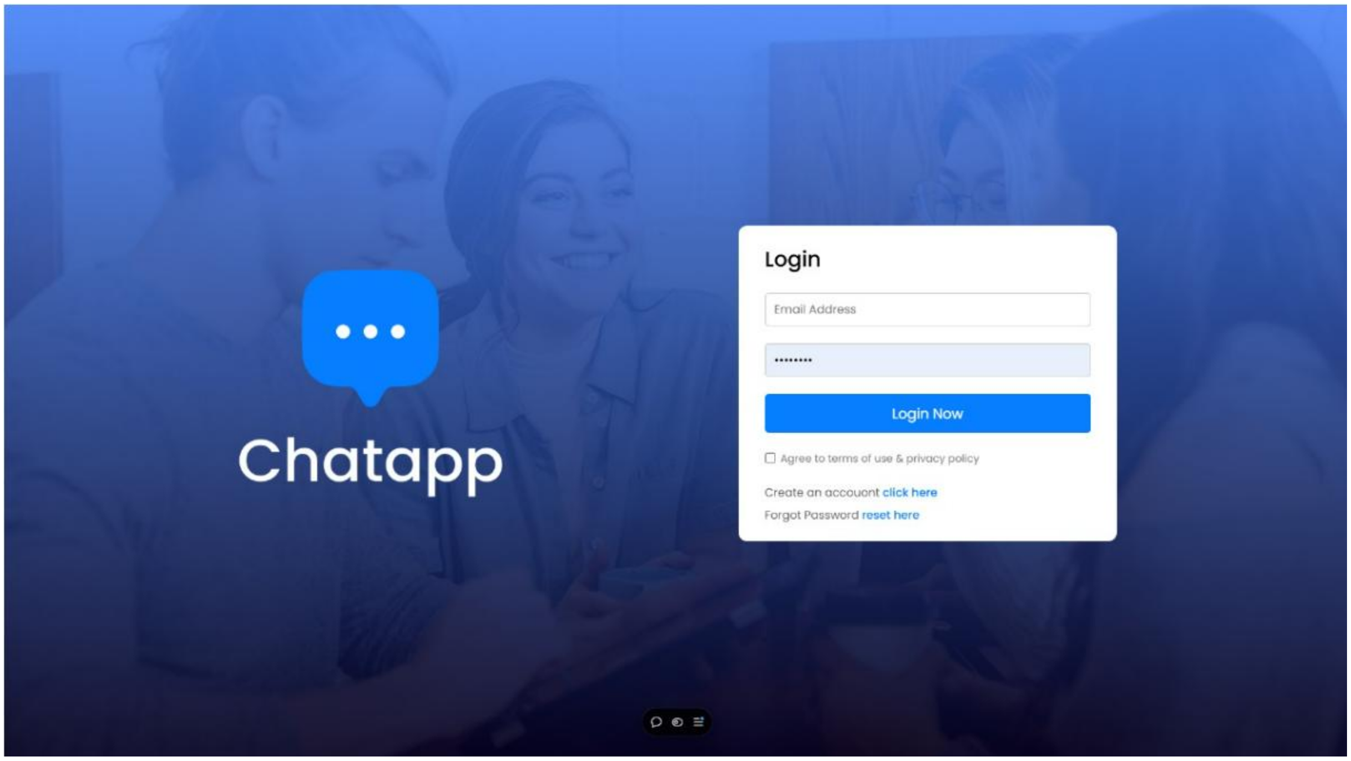


```
      setJoined(true);
    },
    onLeaveRoom: () => {
      navigate("/");
    },
  });
};
```

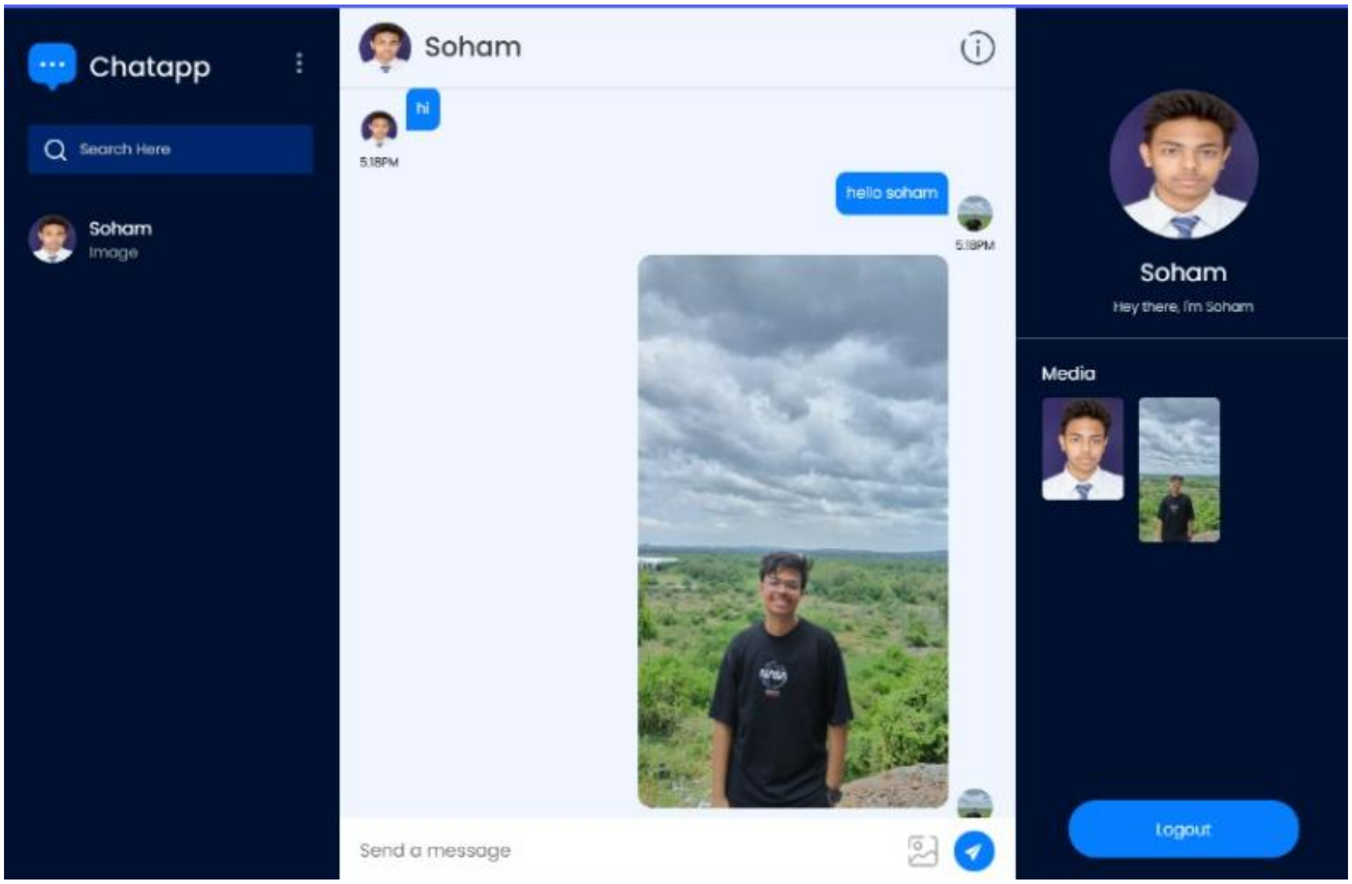
RESULTS & INFERENCES:

Following are some of the results from our application:

 Sign-up Page This is our signup page where users have to enter their details like name ,email address, password and also can upload your pictures to signup our applications.


Chatapp
Login
Email Address
Login Now
Agree to terms of use & privacy policy
Create an accouont click here
Forgot Password reset here

This is the main interface of our application where different options are available like user can create groups , can message any person in personal , can see how many users are online ,etc.



The application interface allows users to communicate with any registered user directly, enabling seamless communication and fostering community engagement.

APPLICATION ORIENTED LEARNING:

Here are the some important fields in which our chat application can be used

For Customer Support Systems :

- Live chat support for e-commerce websites
- Ticket management systems for IT help desks
- Customer service portals for banks and financial institutions
- Integration with CRM systems

For Educational Platforms:

- Virtual classrooms with real-time interaction
- Student-teacher consultation systems
- Group study rooms
- Educational resource sharing platforms

For Healthcare Communications:

- Doctor-patient consultation platforms
- Hospital internal communication systems
- Emergency response coordination
- Medical team collaboration tools

For Business Applications:

- Internal team communication tools
- Project management platforms
- Remote team collaboration systems
- Client communication portals

Here are the some Core Computer Communication Concepts Learned during the building of our chat application

1. Network Protocols
    a. WebSocket Protocol
    b. HTTP/HTTPS
    c. TCP/IP fundamentals
    d. Real-time data transmission
2. Data Management
    a. Database CRUD operations
    b. Data persistence
    c. Caching mechanisms
    d. Data synchronization
3. Security Concepts
    a. Authentication systems
    b. Authorization protocols
    c. Data encryption

    d. Session management
4. Distributed Systems
    a. Event-driven architecture
    b. Message queuing
    c. Load balancing
    d. Scalability patterns
5. Real-Time Processing
    a. Event handling
    b. Broadcasting
    c. Pub/Sub patterns
    d. Connection management

CONCLUSION:

To conclude, the real time chat application chatterbox has been developed with much recent technologies like E-Express.js, R-React.js, N-Node.js making it more extensible. The applications has features like audio call, video call, screen sharing, private messages and some secure invitation method to add friends in the application. It also comes with an option of screen share while video call making it more handy for a user. Thus, the application is a complete package for user real time chatting with some major features to use and if the developer resources increase the application can be extended by adding group chatting models and hidden chatting features to make it more useful for a user.

REFERENCES:

1.https://chat-app-smoky-mu.vercel.app/

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

3.http://www.ir.juit.ac.in:8080/jspui/bitstream/123456789/11513/1/Real-Time%20Chat%20Application.pdf

4. https://developers.google.com/workspace/chat/quickstart/gcf-app

5.https://www.researchgate.net/publication/360483603_Research_paper_on_Group_chatting_Application