Chat application using WebRTC, NodeJS, Socket.io

A COURSE PROJECT REPORT

By

AKASH KOOTTUNGAL (RA1911026010010)
HARIKRISHNAA S (RA1911026010012)
THEJASWIN S(RA1911026010029)

Under the guidance of

Dr. P SARAVANAN

In partial fulfillment for the Course

of

18CSC302J - COMPUTER NETWORKS

in

Computer Science-AIML

FACULTY OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

Kattankulathur, Chenpalpattu District

NOVEMBER 2021

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

(Under Section 3 of UGC Act, 1956)

BONAFIDE CERTIFICATE

Certified that this project report "**Chat application using WebRTC, Node.js and Socket.io**" is the bonafide work of **Akash Koottungal (RA1911026010010), Harikrishnaa S (RA1911026010012), Thejaswin S (RA1911026010029)** who carried out the project work under my supervision.

SIGNATURE

Dr. P Saravanan
Course Faculty
Associate Professor
SRM Institute of Science and Technology
Potheri, SRM Nagar, Kattankulathur,
Tamil Nadu 603203

SIGNATURE

Dr.E. Sasikala,
Course Coordinator
Associate Professor
Data Science and Business Systems
SRM Institute of Science and Technology
Potheri, SRM Nagar, Kattankulathur,
Tamil Nadu 603203

ABSTRACT

One of the last major challenges for the web is to enable human communication via voice and video without using special plugins and without having to pay for these services. WebRTC (Web Real-Time Communication) is a new web standard currently supported by Google, Mozilla, and Opera. It allows peer-to-peer communication between browsers. Its mission is to enable rich, high-quality RTC applications for the browser, mobile platforms, and the Web of Things (WoT), and allow them to communicate via a common set of protocols. WebRTC defines open standards for real-time, plugin-free video, audio, and data communication. Currently, many web services already use RTC but require downloads, native apps or plugins. These include Skype, Facebook (which uses Skype), and Google Hangouts (which use the Google Talk plugin). Downloading, installing, and updating plugins can be complex, error-prone, and annoying and it's often difficult to convince people to install plugins in the first place. This report discusses the implementation of WebRTC to build a web application for real-time communication including video/audio communication along with chat and many more features.

ACKNOWLEDGEMENT

We express our heartfelt thanks to our honorable **Vice Chancellor Dr. C. MUTHAMIZHCHELVAN**, for being the beacon in all our endeavors.

We would like to express my warmth of gratitude to our **Registrar Dr. S. Ponnusamy,** for his encouragement

We express our profound gratitude to our **Dean (College of Engineering and Technology) Dr. T. V.Gopal,** for bringing out novelty in all executions.

We would like to express my heartfelt thanks to Chairperson, School of Computing **Dr. Revathi Venkataraman,** for imparting confidence to complete my course project

We wish to express my sincere thanks to **Course Audit Professor Dr.M.LAKSHMI, Professor and Head, Data Science and Business Systems** and **Course Coordinator Dr.E. Sasikala, Associate Professor, Data Science and Business Systems** for their constant encouragement and support.

We are highly thankful to our course project Internal guide **Dr. P SARAVANAN , Assistant Professor , Department of Data Science and Business Systems,** for his assistance, timely suggestion and guidance throughout the duration of this course project.

We extend my gratitude to **Student HOD** and my Departmental colleagues for their Support.

Finally, we thank our parents and friends near and dear ones who directly and indirectly contributed to the successful completion of our project. Above all, I thank the almighty for showering his blessings on me to complete my Course project

TABLE OF CONTENTS

1. ABSTRACT

One of the last major challenges for the web is to enable human communication via voice and video without using special plugins and without having to pay for these services. WebRTC (Web Real-Time Communication) is a new web standard currently supported by Google, Mozilla, and Opera. It allows peer-to-peer communication between browsers. Its mission is to enable rich, high-quality RTC applications for the browser, mobile platforms, and the Web of Things (WoT), and allow them to communicate via a common set of protocols. WebRTC defines open standards for real-time, plugin-free video, audio, and data communication. Currently, many web services already use RTC but require downloads, native apps or plugins. These include Skype, Facebook (which uses Skype), and Google Hangouts (which use the Google Talk plugin). Downloading, installing, and updating plugins can be complex, error-prone, and annoying and it's often difficult to convince people to install plugins in the first place. This report discusses the implementation of WebRTC to build a web application for real-time communication including video/audio communication along with chat and many more features.

2. INTRODUCTION

WebRTC (Web Real-Time Communication) is an open-source technology that enables Web applications and sites to capture and optionally stream audio and/or video media, as well as to exchange arbitrary data between browsers without requiring an intermediary. The set of standards that comprise WebRTC makes it possible to share data and perform teleconferencing peer-to-peer, without requiring that the user install plug-ins or any other third-party software.

Some of the main use cases of this technology include the following:

- Real-time audio and/or video communication
- Web conferencing
- Direct data transfers

Unlike most real-time systems (e.g. SIP), WebRTC communications are directly controlled by some Web server, via a JavaScript API. WebRTC consists of several interrelated APIs and protocols which work together to achieve this.

Some of the several **JavaScript APIs** are:

- getUserMedia() : capture audio and video
- MediaRecorder : record audio and video
- RTCPeerConnection : stream audio and video between users
- RTCDataChannel : stream data between users

WebRTC serves multiple purposes; together with the Media Capture and Streams API, they provide powerful multimedia capabilities to the Web, including support for audio and video conferencing, file exchange, screen sharing, identity management, and interfacing with legacy telephone systems including support for sending DTMF (touch-tone dialing) signals. Connections between peers can be made without requiring any special drivers or plug-ins, and can often be made without any intermediary servers.

Connections between two peers are represented by the RTCPeerConnection interface. Once a connection has been established and opened using RTCPeerConnection, media streams (MediaStreams) and/or data channels (RTCDataChannels) can be added to the connection.

Media streams can consist of any number of tracks of media information; tracks, which are represented by objects based on the MediaStreamTrack interface, may contain one of a number of types of media data, including audio, video, and text (such as subtitles or even chapter names). Most streams consist of at least one audio track and likely also a video track, and can be used to send and receive both live media or stored media information (such as a streamed movie).

Features

- Multi-participants
- Toggling of the video stream
- Toggling of the audio stream (mute & unmute)
- Screen sharing
- Text chat
- Mute individual participant
- Expand participants' stream
- Screen Recording
- Video Recording

Connection setup and management

There are interfaces, dictionaries, and types used to set up, open, and manage WebRTC connections. Interfaces representing peer media connections, data channels, and interfaces are used when exchanging information on the capabilities of each peer in order to select the best possible configuration for a two-way media connection.

3. REQUIREMENT ANALYSIS

1. Hardware Requirements

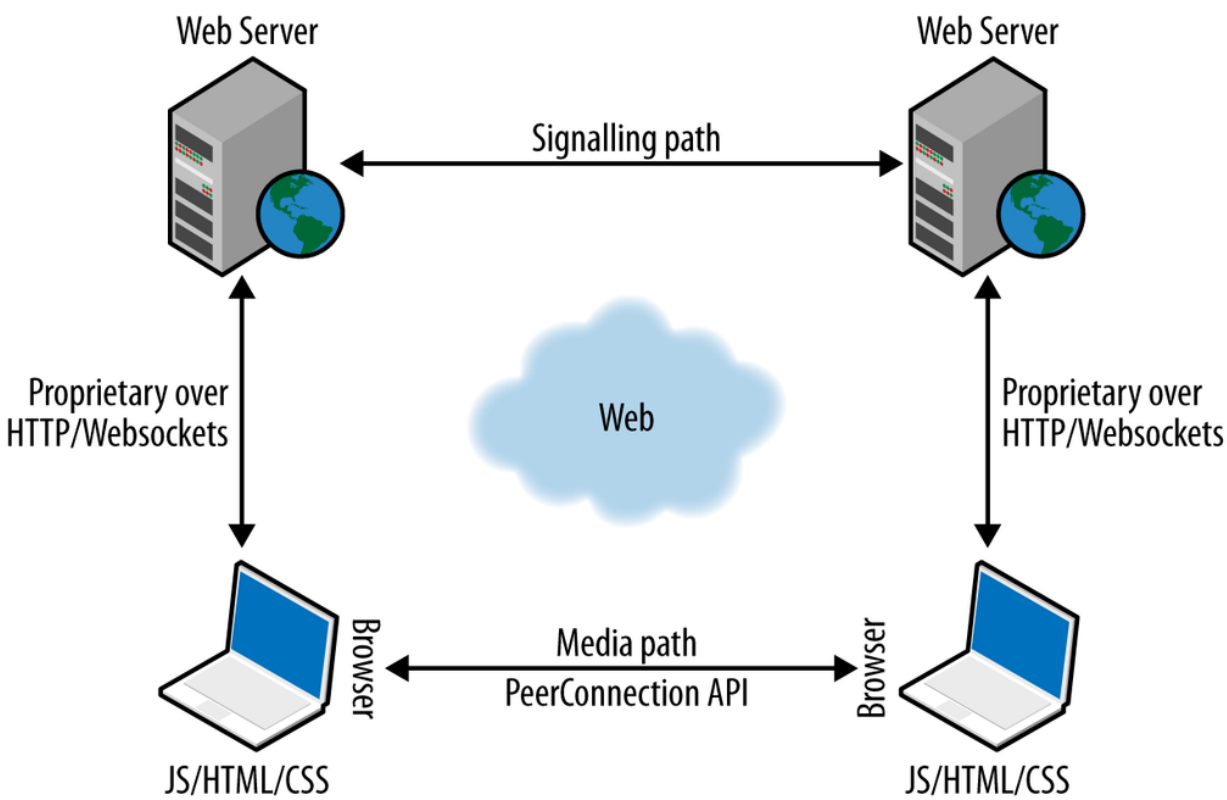
- Processor : 2.4 Ghz
- RAM : 3GB
- Storage space : 500 MB (Minimum free space)

2. Software Requirements

- Operating System: Windows 10
- Tools: VScode, Node.js, GitHub
- Frontend: HTML,CSS
- Backend: Javascript
- Dependencies:
  - express: 4.17.1
  - serve-favicon: 2.5.0
  - socket.io: 2.4.0
- DevDependencies: nodemon: 2.0.6

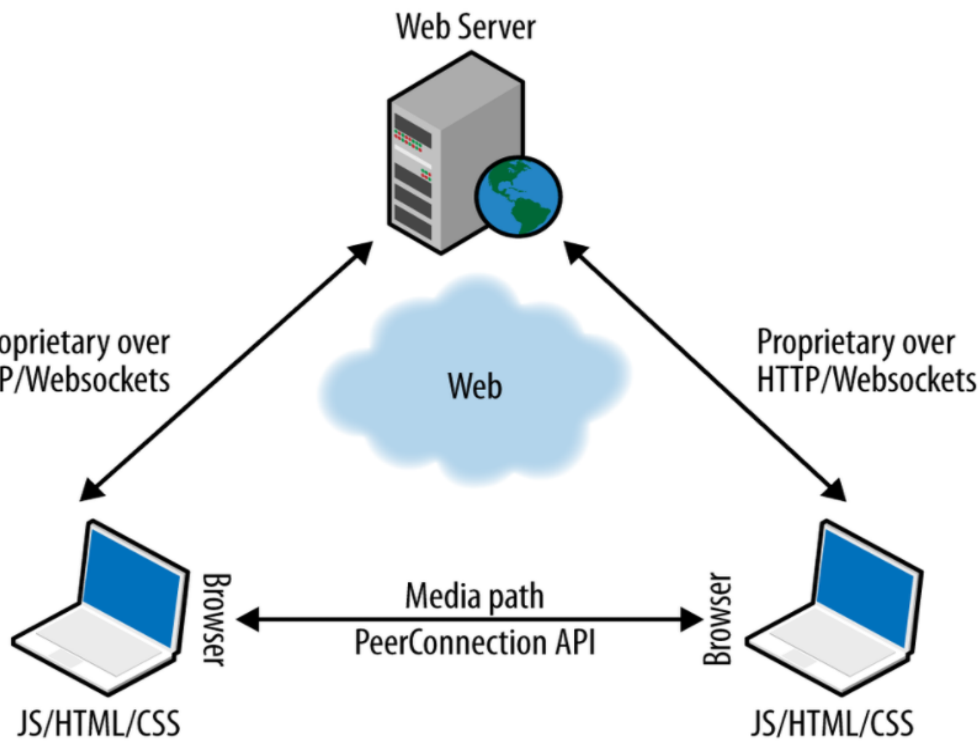
4. ARCHITECTURE & DESIGN

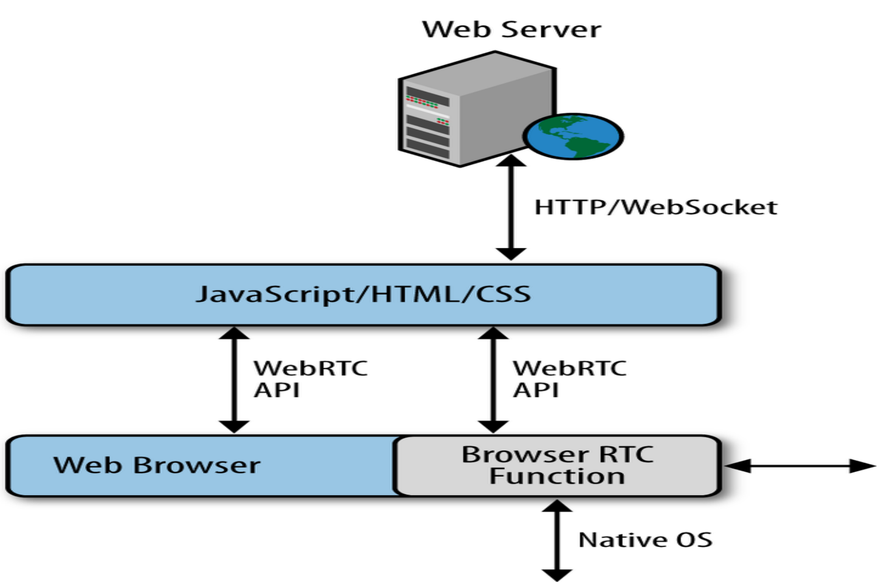
WebRTC extends the client-server semantics by introducing a peer-to-peer communication paradigm between browsers. The most general WebRTC architectural model (see Figure 1-1) draws its inspiration from the so-called SIP (Session Initiation Protocol) Trapezoid (RFC3261).



In the WebRTC Trapezoid model, both browsers are running a web application, which is downloaded from a different web server. Signaling messages are used to set up and terminate communications. They are transported by the HTTP or WebSocket protocol via web servers that can modify, translate, or manage them as needed. It is worth noting that the signaling between browser and server is not standardized in WebRTC, as it is considered to be part of the application (see Signaling). As to the data path, a PeerConnection allows media to flow directly between browsers without any intervening servers. The two web servers can communicate using a standard signaling protocol such as SIP or Jingle (XEP-0166). Otherwise, they can use a proprietary signaling protocol.

The most common WebRTC scenario is likely to be the one where both browsers are running the same web application, downloaded from the same web page. In this case, the Trapezoid becomes a Triangle.





Real-time flow of data is streamed across the network in order to allow direct communication between two browsers, with no further intermediaries along the path. This clearly represents a revolutionary approach to web-based communication.

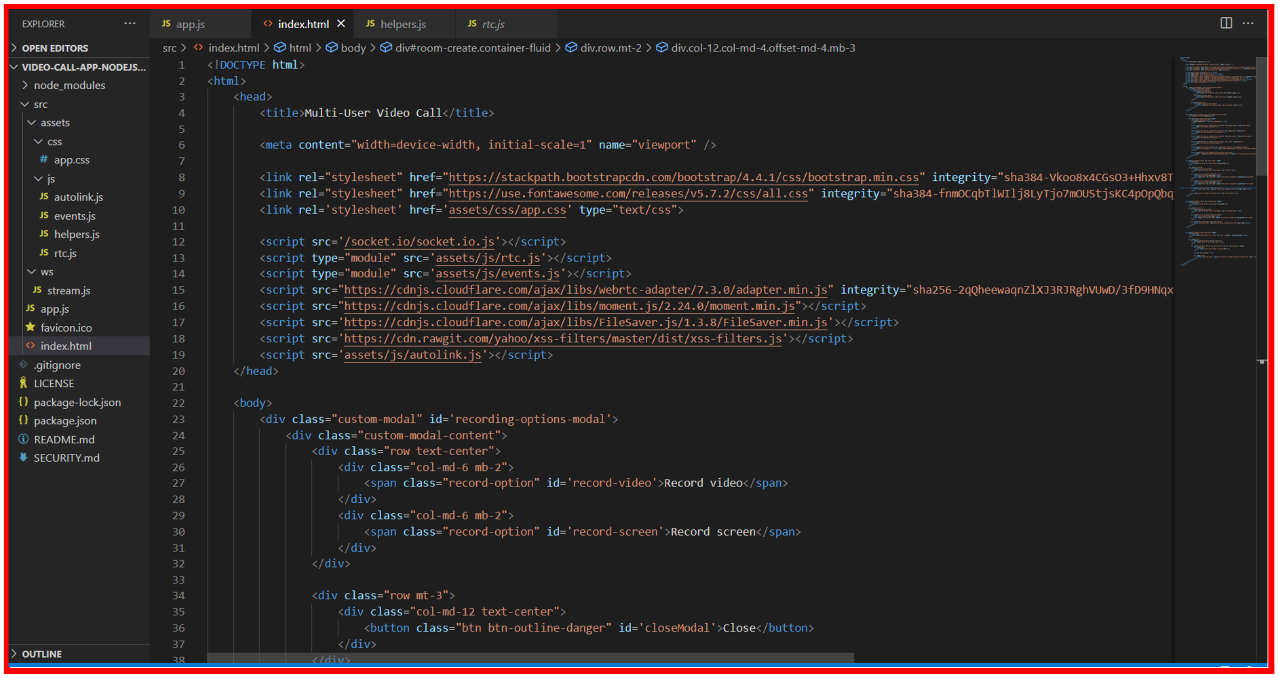
Communication involves direct media streams between the two browsers, with the media path negotiated and instantiated through a complex sequence of interactions involving the following entities:

- The caller browser and the caller JavaScript application (e.g., through the mentioned JavaScript API)
- The caller JavaScript application and the application provider (typically, a web server)
- The application provider and the callee JavaScript application
- The callee JavaScript application and the callee browser (again through the application-browser JavaScript API)

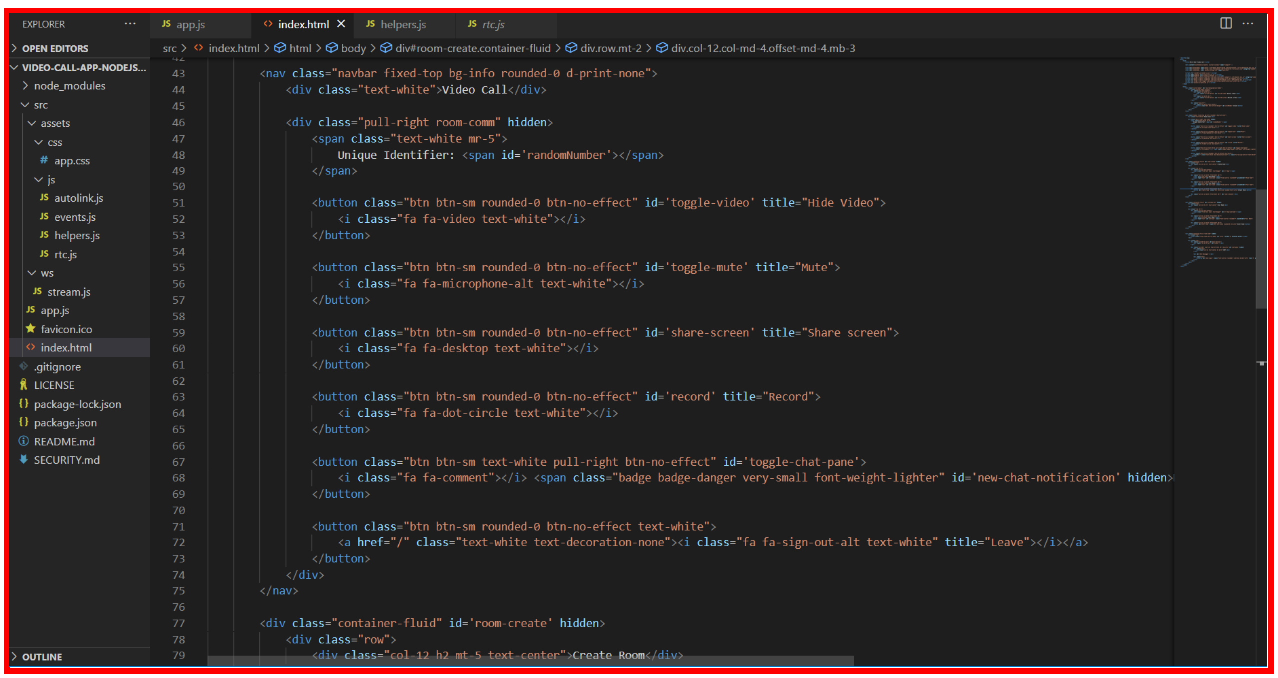
5. IMPLEMENTATION

CODE:

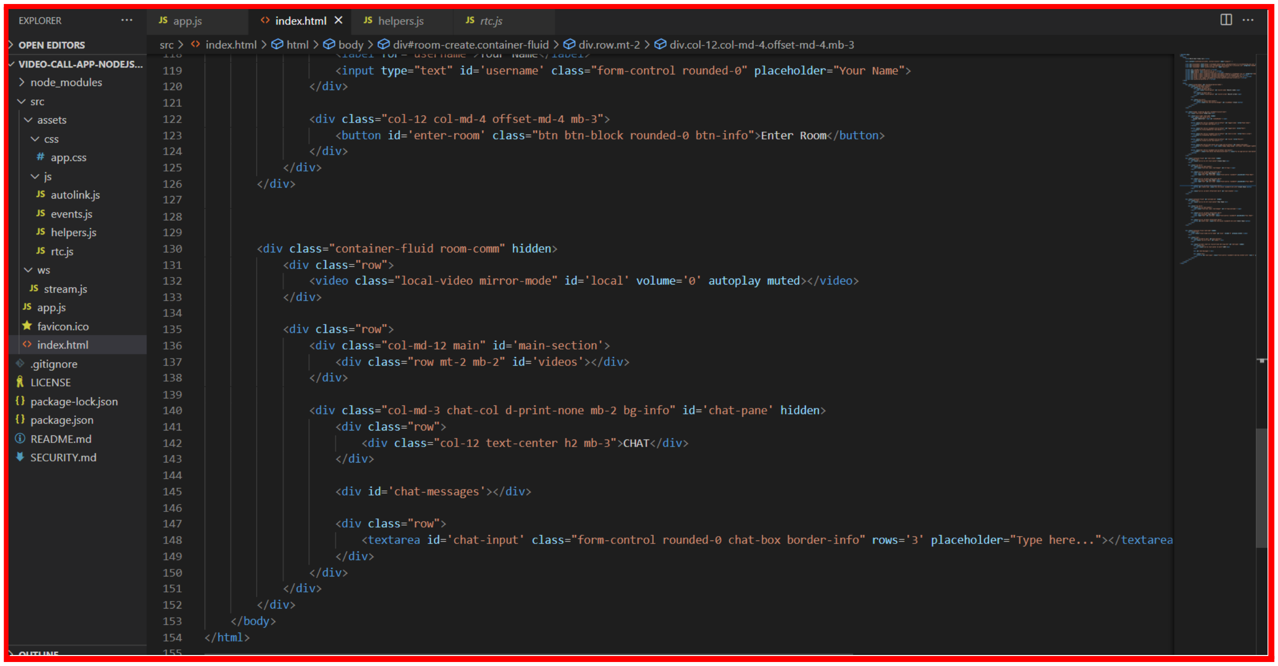
5.1. FRONTEND HTML



```html
<!DOCTYPE html>
<html>
    <head>
        <title>Multi-User Video Call</title>

        <meta content="width=device-width, initial-scale=1" name="viewport" />

        <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.min.css" integrity="sha384-Vkoo8x4CGsO3+Hhxv8T
        <link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.7.2/css/all.css" integrity="sha384-fnmOCqbTlWIlj8LyTjo7mOUStjsKC4pOpQbq
        <link rel='stylesheet' href='assets/css/app.css' type="text/css">

        <script src='/socket.io/socket.io.js'></script>
        <script type="module" src='assets/js/rtc.js'></script>
        <script type="module" src='assets/js/events.js'></script>
        <script src="https://cdnjs.cloudflare.com/ajax/libs/webrtc-adapter/7.3.0/adapter.min.js" integrity="sha256-2qQheewaqnZlXJ3RJRghVUwD/3fD9HNqx
        <script src="https://cdnjs.cloudflare.com/ajax/libs/moment.js/2.24.0/moment.min.js"></script>
        <script src='https://cdnjs.cloudflare.com/ajax/libs/FileSaver.js/1.3.8/FileSaver.min.js'></script>
        <script src='https://cdn.rawgit.com/yahoo/xss-filters/master/dist/xss-filters.js'></script>
        <script src='assets/js/autolink.js'></script>
    </head>

    <body>
        <div class="custom-modal" id='recording-options-modal'>
            <div class="custom-modal-content">
                <div class="row text-center">
                    <div class="col-md-6 mb-2">
                        <span class="record-option" id='record-video'>Record video</span>
                    </div>
                    <div class="col-md-6 mb-2">
                        <span class="record-option" id='record-screen'>Record screen</span>
                    </div>
                </div>

                <div class="row mt-3">
                    <div class="col-md-12 text-center">
                        <button class="btn btn-outline-danger" id='closeModal'>Close</button>
                    </div>
                </div>
```

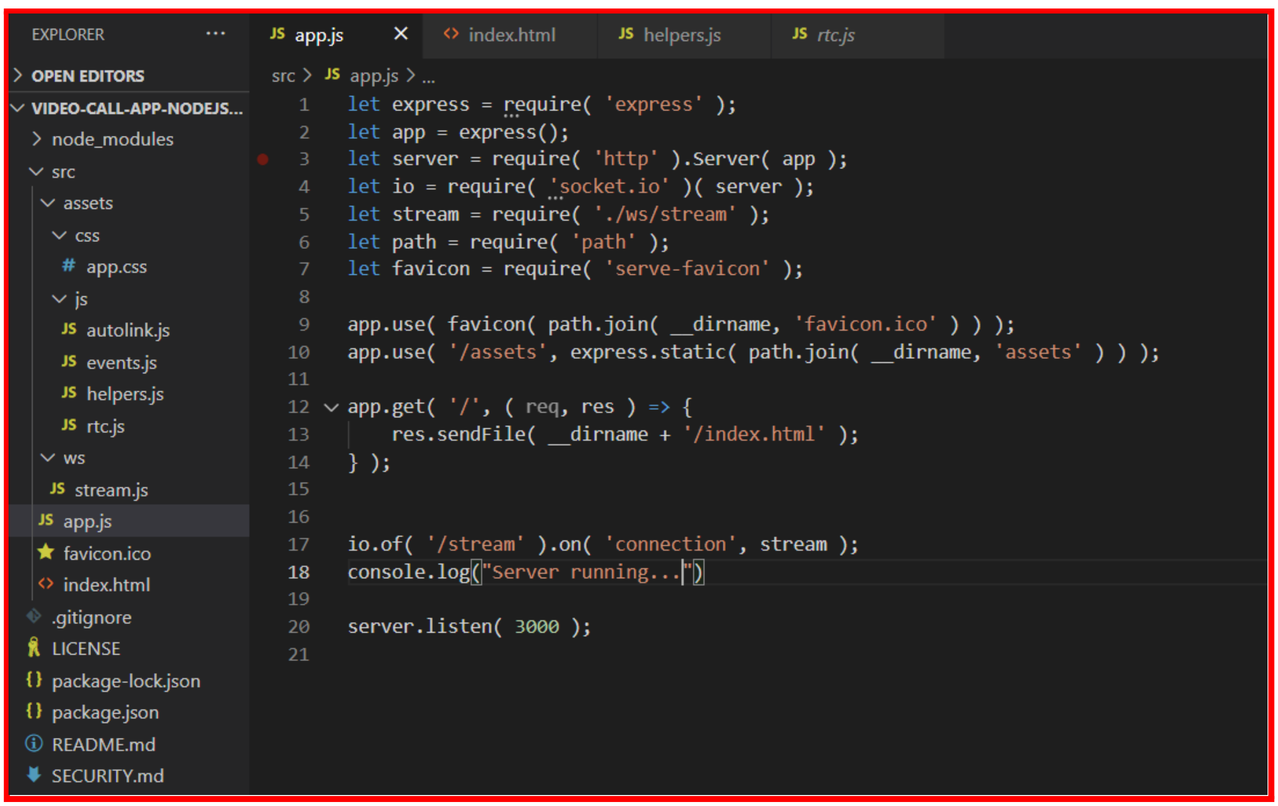


```html
        <nav class="navbar fixed-top bg-info rounded-0 d-print-none">
            <div class="text-white">Video Call</div>

            <div class="pull-right room-comm" hidden>
                <span class="text-white mr-5">
                    Unique Identifier: <span id='randomNumber'></span>
                </span>

                <button class="btn btn-sm rounded-0 btn-no-effect" id='toggle-video' title="Hide Video">
                    <i class="fa fa-video text-white"></i>
                </button>

                <button class="btn btn-sm rounded-0 btn-no-effect" id='toggle-mute' title="Mute">
                    <i class="fa fa-microphone-alt text-white"></i>
                </button>

                <button class="btn btn-sm rounded-0 btn-no-effect" id='share-screen' title="Share screen">
                    <i class="fa fa-desktop text-white"></i>
                </button>

                <button class="btn btn-sm rounded-0 btn-no-effect" id='record' title="Record">
                    <i class="fa fa-dot-circle text-white"></i>
                </button>

                <button class="btn btn-sm text-white pull-right btn-no-effect" id='toggle-chat-pane'>
                    <i class="fa fa-comment"></i> <span class="badge badge-danger very-small font-weight-lighter" id='new-chat-notification' hidden>
                </button>

                <button class="btn btn-sm rounded-0 btn-no-effect text-white">
                    <a href="/" class="text-white text-decoration-none"><i class="fa fa-sign-out-alt text-white" title="Leave"></i></a>
                </button>
            </div>
        </nav>

        <div class="container-fluid" id='room-create' hidden>
            <div class="row">
                <div class="col-12 h2 mt-5 text-center">Create Room</div>
```

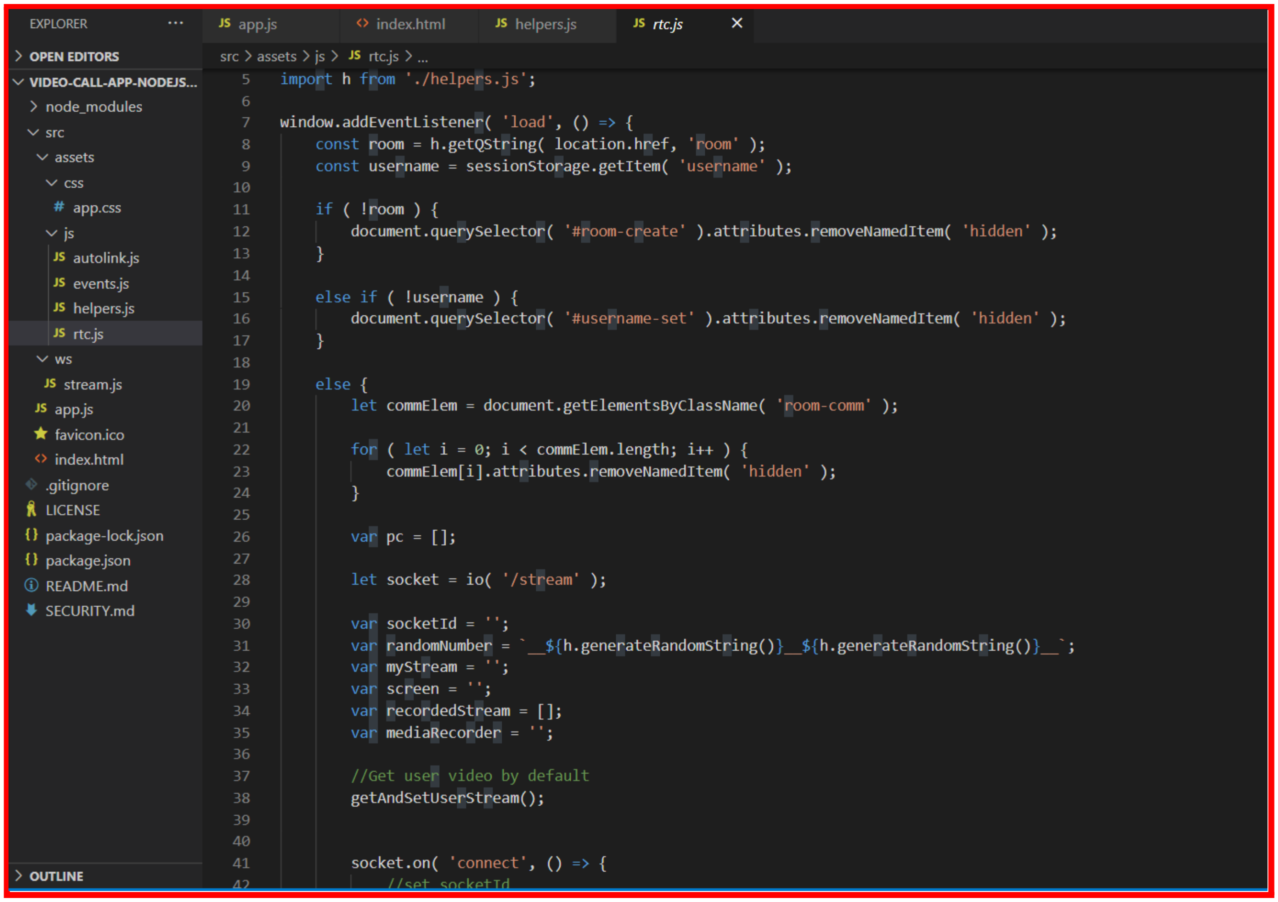


5.2 BACKEND MODULES

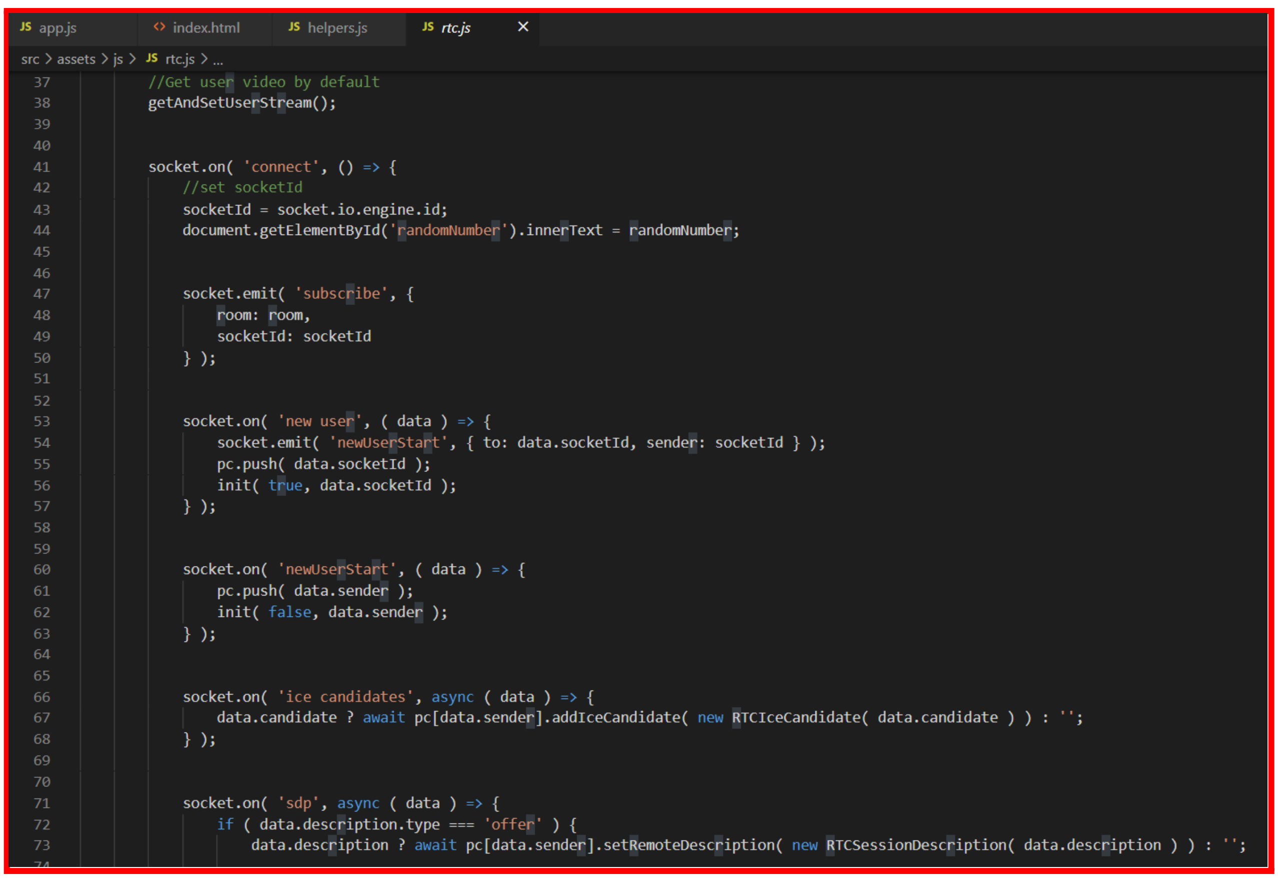
5.2.1 app.js



5.2.2 rtc.js




```
import h from './helpers.js';

window.addEventListener( 'load', () => {
    const room = h.getQString( location.href, 'room' );
    const username = sessionStorage.getItem( 'username' );

    if ( !room ) {
        document.querySelector( '#room-create' ).attributes.removeNamedItem( 'hidden' );
    }

    else if ( !username ) {
        document.querySelector( '#username-set' ).attributes.removeNamedItem( 'hidden' );
    }

    else {
        let commElem = document.getElementsByClassName( 'room-comm' );

        for ( let i = 0; i < commElem.length; i++ ) {
            commElem[i].attributes.removeNamedItem( 'hidden' );
        }

        var pc = [];

        let socket = io( '/stream' );

        var socketId = '';
        var randomNumber = `__${h.generateRandomString()}__${h.generateRandomString()}__`;
        var myStream = '';
        var screen = '';
        var recordedStream = [];
        var mediaRecorder = '';

        //Get user video by default
        getAndSetUserStream();


        socket.on( 'connect', () => {
            //set socketId
```




```
        //Get user video by default
        getAndSetUserStream();


        socket.on( 'connect', () => {
            //set socketId
            socketId = socket.io.engine.id;
            document.getElementById('randomNumber').innerText = randomNumber;


            socket.emit( 'subscribe', {
                room: room,
                socketId: socketId
            } );


            socket.on( 'new user', ( data ) => {
                socket.emit( 'newUserStart', { to: data.socketId, sender: socketId } );
                pc.push( data.socketId );
                init( true, data.socketId );
            } );


            socket.on( 'newUserStart', ( data ) => {
                pc.push( data.sender );
                init( false, data.sender );
            } );


            socket.on( 'ice candidates', async ( data ) => {
                data.candidate ? await pc[data.sender].addIceCandidate( new RTCIceCandidate( data.candidate ) ) : '';
            } );


            socket.on( 'sdp', async ( data ) => {
                if ( data.description.type === 'offer' ) {
                    data.description ? await pc[data.sender].setRemoteDescription( new RTCSessionDescription( data.description ) ) : '';
```

src > assets > js > JS rtc.js > ...

```
        function shareScreen() {
            h.shareScreen().then( ( stream ) => {
                h.toggleShareIcons( true );

                //disable the video toggle btns while sharing screen. This is to ensure clicking on the btn does not interfere with the screen shari
                //It will be enabled was user stopped sharing screen
                h.toggleVideoBtnDisabled( true );

                //save my screen stream
                screen = stream;

                //share the new stream with all partners
                broadcastNewTracks( stream, 'video', false );

                //When the stop sharing button shown by the browser is clicked
                screen.getVideoTracks()[0].addEventListener( 'ended', () => {
                    stopSharingScreen();
                } );
            } ).catch( ( e ) => {
                console.error( e );
            } );
        }



        function stopSharingScreen() {
            //enable video toggle btn
            h.toggleVideoBtnDisabled( false );

            return new Promise( ( res, rej ) => {
                screen.getTracks().length ? screen.getTracks().forEach( track => track.stop() ) : '';

                res();
            } ).then( () => {
                h.toggleShareIcons( false );
                broadcastNewTracks( myStream, 'video' );
            } ).catch( ( e ) => {
                console.error( e );
            } );
        }

```

Ln 2, Col 11 (1 selected) Spaces: 4 UTF-8 LF {} Java

src > assets > js > JS rtc.js > ...

```

        function startRecording( stream ) {
            mediaRecorder = new MediaRecorder( stream, {
                mimeType: 'video/webm;codecs=vp9'
            } );

            mediaRecorder.start( 1000 );
            toggleRecordingIcons( true );

            mediaRecorder.ondataavailable = function ( e ) {
                recordedStream.push( e.data );
            };

            mediaRecorder.onstop = function () {
                toggleRecordingIcons( false );

                h.saveRecordedStream( recordedStream, username );

                setTimeout( () => {
                    recordedStream = [];
                }, 3000 );
            };

            mediaRecorder.onerror = function ( e ) {
                console.error( e );
            };
        }


        //Chat textarea
        document.getElementById( 'chat-input' ).addEventListener( 'keypress', ( e ) => {
            if ( e.which === 13 && ( e.target.value.trim() ) ) {
                e.preventDefault();

                sendMsg( e.target.value );

                setTimeout( () => {
                    e.target.value = '';
                }, 50 );
            }
        } );
```

src > assets > js > JS rtc.js > ...

```
        //When record button is clicked
        document.getElementById( 'record' ).addEventListener( 'click', ( e ) => {
            /**
             * Ask user what they want to record.
             * Get the stream based on selection and start recording
             */
            if ( !mediaRecorder || mediaRecorder.state == 'inactive' ) {
                h.toggleModal( 'recording-options-modal', true );
            }

            else if ( mediaRecorder.state == 'paused' ) {
                mediaRecorder.resume();
            }

            else if ( mediaRecorder.state == 'recording' ) {
                mediaRecorder.stop();
            }
        } );


        //When user choose to record screen
        document.getElementById( 'record-screen' ).addEventListener( 'click', () => {
            h.toggleModal( 'recording-options-modal', false );

            if ( screen && screen.getVideoTracks().length ) {
                startRecording( screen );
            }

            else {
                h.shareScreen().then( ( screenStream ) => {
                    startRecording( screenStream );
                } ).catch( () => { } );
            }
        } );


        //When user choose to record own video
```

src > assets > js > rtc.js > window.addEventListener('load') callback > addEventListener('click') callback

```
        //When the mute icon is clicked
        document.getElementById( 'toggle-mute' ).addEventListener( 'click', ( e ) => {
            e.preventDefault();

            let elem = document.getElementById( 'toggle-mute' );

            if ( myStream.getAudioTracks()[0].enabled ) {
                e.target.classList.remove( 'fa-microphone-alt' );
                e.target.classList.add( 'fa-microphone-alt-slash' );
                elem.setAttribute( 'title', 'Unmute' );

                myStream.getAudioTracks()[0].enabled = false;
            }

            else {
                e.target.classList.remove( 'fa-microphone-alt-slash' );
                e.target.classList.add( 'fa-microphone-alt' );
                elem.setAttribute( 'title', 'Mute' );

                myStream.getAudioTracks()[0].enabled = true;
            }

            broadcastNewTracks( myStream, 'audio' );
        } );


        //When user clicks the 'Share screen' button
        document.getElementById( 'share-screen' ).addEventListener( 'click', ( e ) => {
            e.preventDefault();

            if ( screen && screen.getVideoTracks().length && screen.getVideoTracks()[0].readyState != 'ended' ) {
                stopSharingScreen();
            }

            else {
                shareScreen();
            }
        } );
```

5.2.3 helper.js

src > assets > js > helpers.js > default > closeVideo

```javascript
export default {
    generateRandomString() {
        const crypto = window.crypto || window.msCrypto;
        let array = new Uint32Array(1);

        return crypto.getRandomValues(array);
    },


    closeVideo( elemId ) {
        if ( document.getElementById( elemId ) ) {
            document.getElementById( elemId ).remove();
            this.adjustVideoElemSize();
        }
    },


    pageHasFocus() {
        return !( document.hidden || document.onfocusout || window.onpagehide || window.onblur );
    },


    getQString( url = '', keyToReturn = '' ) {
        url = url ? url : location.href;
        let queryStrings = decodeURIComponent( url ).split( '#', 2 )[0].split( '?', 2 )[1];

        if ( queryStrings ) {
            let splittedQStrings = queryStrings.split( '&' );

            if ( splittedQStrings.length ) {
                let queryStringObj = {};

                splittedQStrings.forEach( function ( keyValuePair ) {
                    let keyValue = keyValuePair.split( '=', 2 );

                    if ( keyValue.length ) {
                        queryStringObj[keyValue[0]] = keyValue[1];
                    }
```

src > assets > js > helpers.js > default > closeVideo

```javascript

    userMediaAvailable() {
        return !!( navigator.getUserMedia || navigator.webkitGetUserMedia || navigator.mozGetUserMedia || navigator.msGetUserMedia );
    },


    getUserFullMedia() {
        if ( this.userMediaAvailable() ) {
            return navigator.mediaDevices.getUserMedia( {
                video: true,
                audio: {
                    echoCancellation: true,
                    noiseSuppression: true
                }
            } );
        }

        else {
            throw new Error( 'User media not available' );
        }
    },


    getUserAudio() {
        if ( this.userMediaAvailable() ) {
            return navigator.mediaDevices.getUserMedia( {
                audio: {
                    echoCancellation: true,
                    noiseSuppression: true
                }
            } );
        }

        else {
            throw new Error( 'User media not available' );
        }
    },
```

src > assets > js > helpers.js > default > closeVideo

```javascript

    addChat( data, senderType ) {
        let chatMsgDiv = document.querySelector( '#chat-messages' );
        let contentAlign = 'justify-content-end';
        let senderName = 'You';
        let msgBg = 'bg-white';

        if ( senderType === 'remote' ) {
            contentAlign = 'justify-content-start';
            senderName = data.sender;
            msgBg = '';

            this.toggleChatNotificationBadge();
        }

        let infoDiv = document.createElement( 'div' );
        infoDiv.className = 'sender-info';
        infoDiv.innerText = `${ senderName } - ${ moment().format( 'Do MMMM, YYYY h:mm a' ) }`;

        let colDiv = document.createElement( 'div' );
        colDiv.className = `col-10 card chat-card msg ${ msgBg }`;
        colDiv.innerHTML = xssFilters.inHTMLData( data.msg ).autoLink( { target: "_blank", rel: "nofollow"});

        let rowDiv = document.createElement( 'div' );
        rowDiv.className = `row ${ contentAlign } mb-2`;


        colDiv.appendChild( infoDiv );
        rowDiv.appendChild( colDiv );

        chatMsgDiv.appendChild( rowDiv );

```

src > assets > js > helpers.js > default > closeVideo

```
    shareScreen() {
        if ( this.userMediaAvailable() ) {
            return navigator.mediaDevices.getDisplayMedia( {
                video: {
                    cursor: "always"
                },
                audio: {
                    echoCancellation: true,
                    noiseSuppression: true,
                    sampleRate: 44100
                }
            } );
        }

        else {
            throw new Error( 'User media not available' );
        }
    },


    getIceServer() {
        return {
            iceServers: [
                {
                     urls: [ "stun:bn-turn1.xirsys.com" ]
                },
                {   username: "BjoSbQASFNChhXJsrYBS7CACxeBiiCRWn-krWzxQZ0oRY5bPUnqxHBueXab9sfi0AAAAAGGA1R5ha2FzaDcxNw==",
                    credential: "db2a0ca4-3ba2-11ec-a0e7-0242ac140004",
                    urls: [ "turn:bn-turn1.xirsys.com:80?transport=udp",
                            "turn:bn-turn1.xirsys.com:3478?transport=udp",
                            "turn:bn-turn1.xirsys.com:80?transport=tcp",
                            "turn:bn-turn1.xirsys.com:3478?transport=tcp",
                            "turns:bn-turn1.xirsys.com:443?transport=tcp",
                            "turns:bn-turn1.xirsys.com:5349?transport=tcp"    ]
                }
            ]
        };
```

5.2.4 events.js

src > assets > js > events.js > window.addEventListener('load') callback

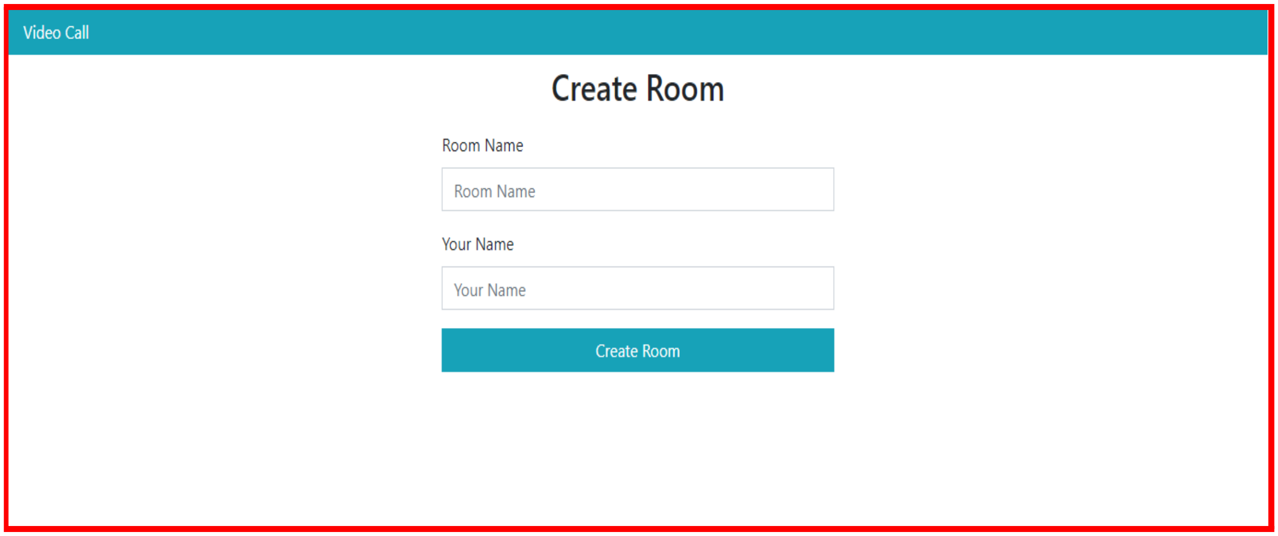
```
import helpers from './helpers.js';

window.addEventListener( 'load', () => {
    //When the chat icon is clicked
    document.querySelector( '#toggle-chat-pane' ).addEventListener( 'click', ( e ) => {
        let chatElem = document.querySelector( '#chat-pane' );
        let mainSecElem = document.querySelector( '#main-section' );

        if ( chatElem.classList.contains( 'chat-opened' ) ) {
            chatElem.setAttribute( 'hidden', true );
            mainSecElem.classList.remove( 'col-md-9' );
            mainSecElem.classList.add( 'col-md-12' );
            chatElem.classList.remove( 'chat-opened' );
        }

        else {
            chatElem.attributes.removeNamedItem( 'hidden' );
            mainSecElem.classList.remove( 'col-md-12' );
            mainSecElem.classList.add( 'col-md-9' );
            chatElem.classList.add( 'chat-opened' );
        }

        //remove the 'New' badge on chat icon (if any) once chat is opened.
        setTimeout( () => {
            if ( document.querySelector( '#chat-pane' ).classList.contains( 'chat-opened' ) ) {
                helpers.toggleChatNotificationBadge();
            }
        }, 300 );
    } );


    //When the video frame is clicked. This will enable picture-in-picture
    document.getElementById( 'local' ).addEventListener( 'click', () => {
        if ( !document.pictureInPictureElement ) {
            document.getElementById( 'local' ).requestPictureInPicture()
                .catch( error => {
                    // Video failed to enter Picture-in-Picture mode.
                    console.error( error );
```

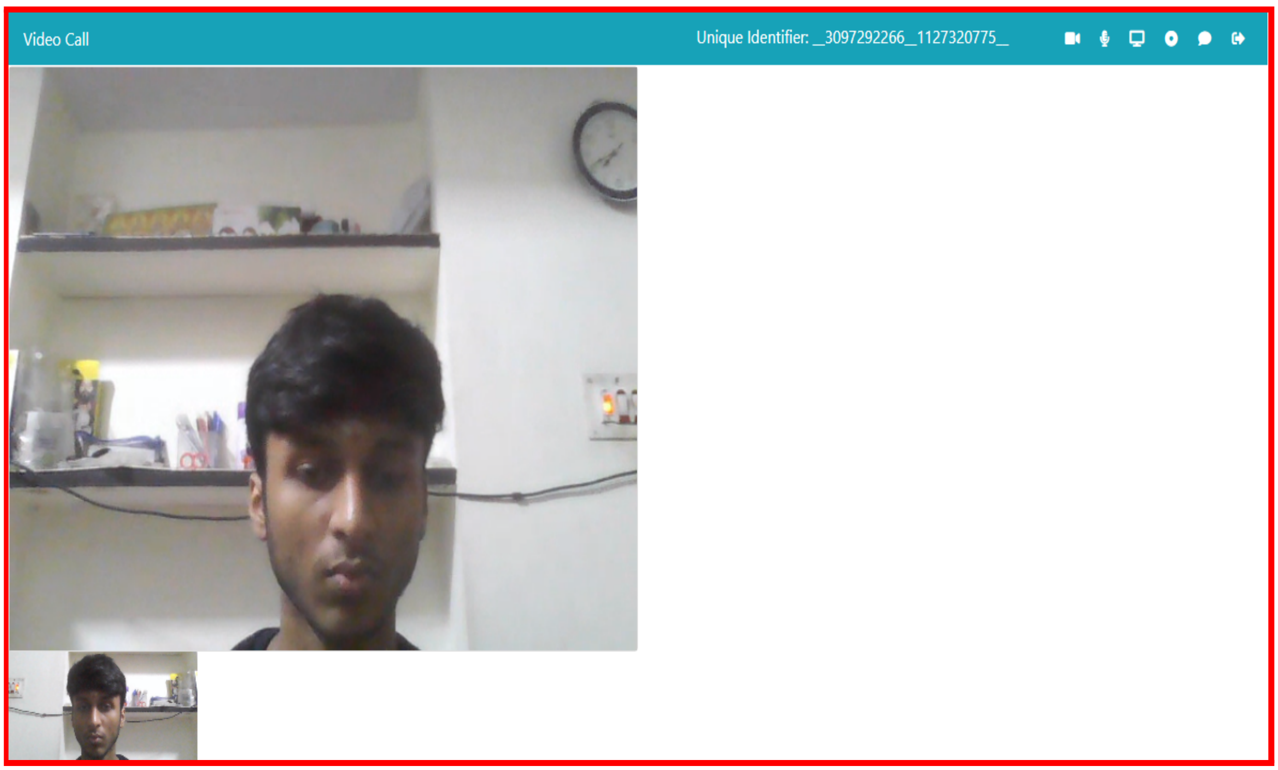
src > assets > js > events.js > window.addEventListener('load') callback

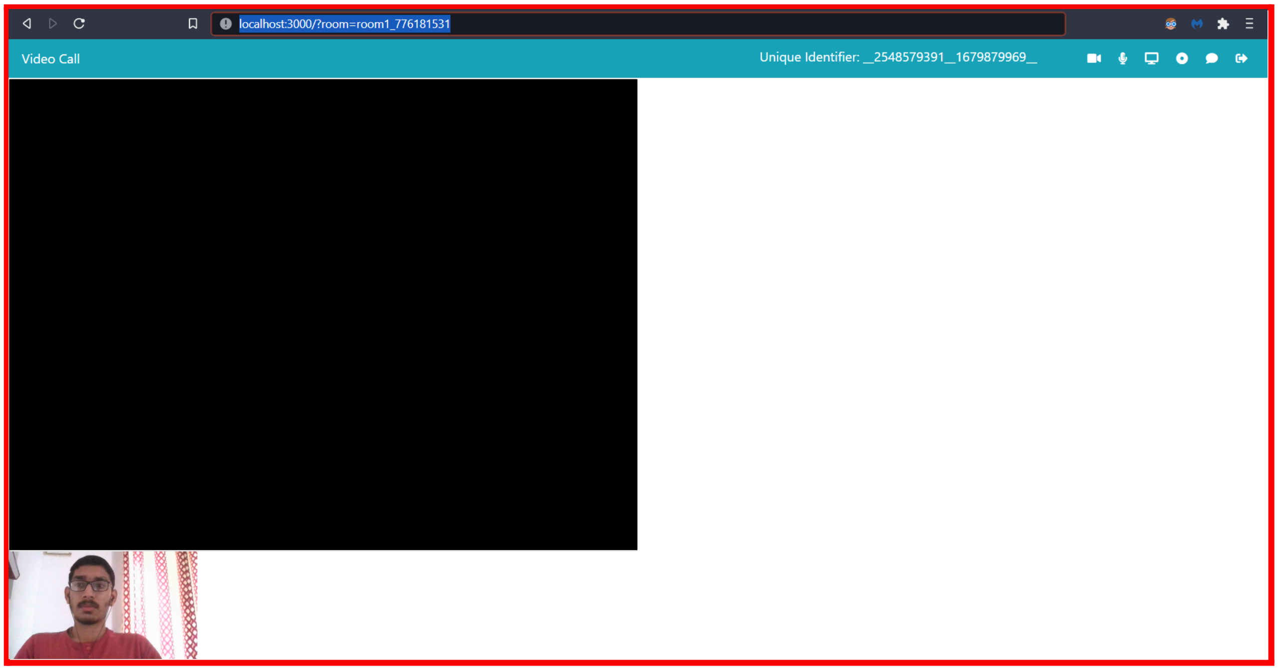
```
    //When the video frame is clicked. This will enable picture-in-picture
    document.getElementById( 'local' ).addEventListener( 'click', () => {
        if ( !document.pictureInPictureElement ) {
            document.getElementById( 'local' ).requestPictureInPicture()
                .catch( error => {
                    // Video failed to enter Picture-in-Picture mode.
                    console.error( error );
                } );
        }

        else {
            document.exitPictureInPicture()
                .catch( error => {
                    // Video failed to leave Picture-in-Picture mode.
                    console.error( error );
                } );
        }
    } );


    //When the 'Create room" is button is clicked
    document.getElementById( 'create-room' ).addEventListener( 'click', ( e ) => {
        e.preventDefault();

        let roomName = document.querySelector( '#room-name' ).value;
        let yourName = document.querySelector( '#your-name' ).value;

        if ( roomName && yourName ) {
            //remove error message, if any
            document.querySelector('#err-msg').innerText = "";

            //save the user's name in sessionStorage
            sessionStorage.setItem( 'username', yourName );

            //create room link
            let roomLink = `${ location.origin }?room=${ roomName.trim().replace( ' ', '_' ) }_${ helpers.generateRandomString() }`;
```

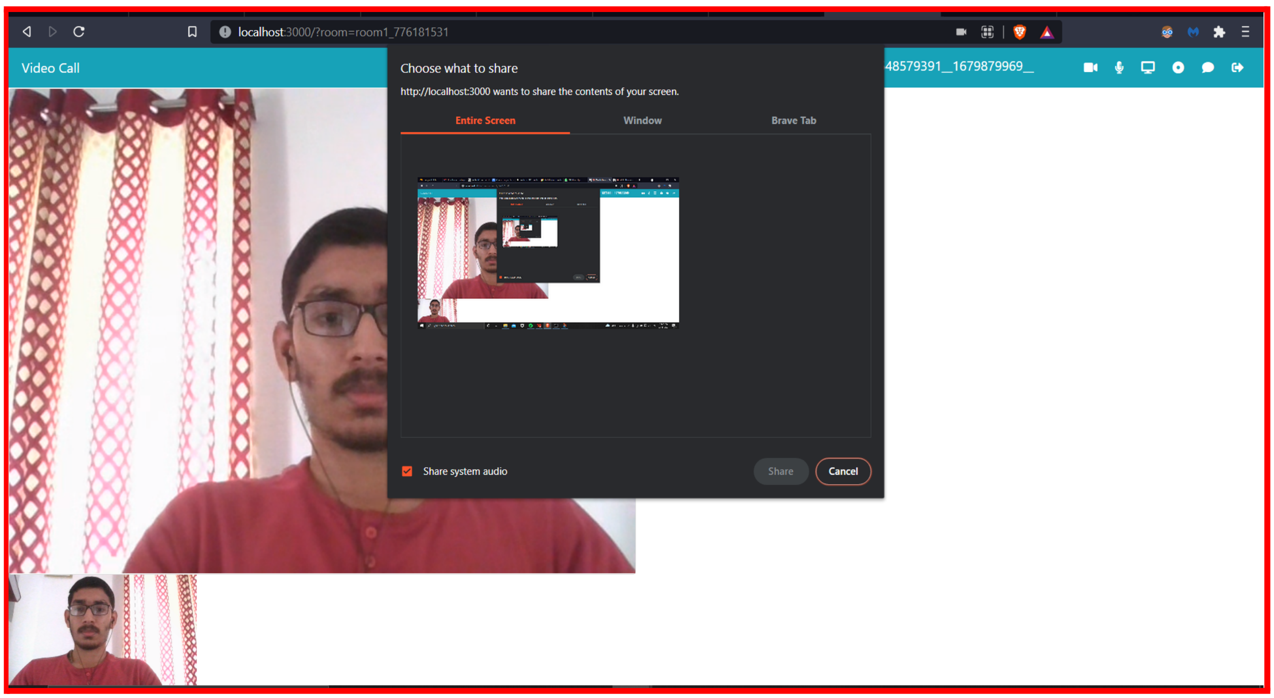
6. EXPERIMENT RESULTS & ANALYSIS

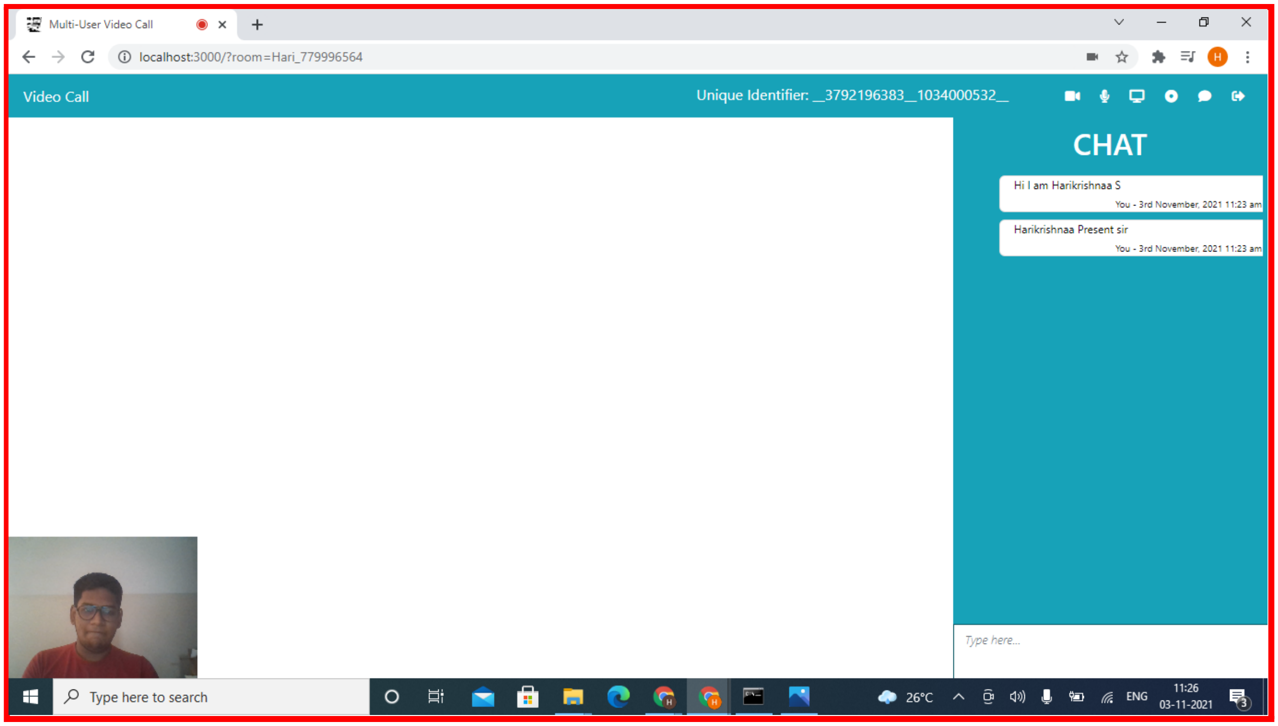
6.1 RESULTS

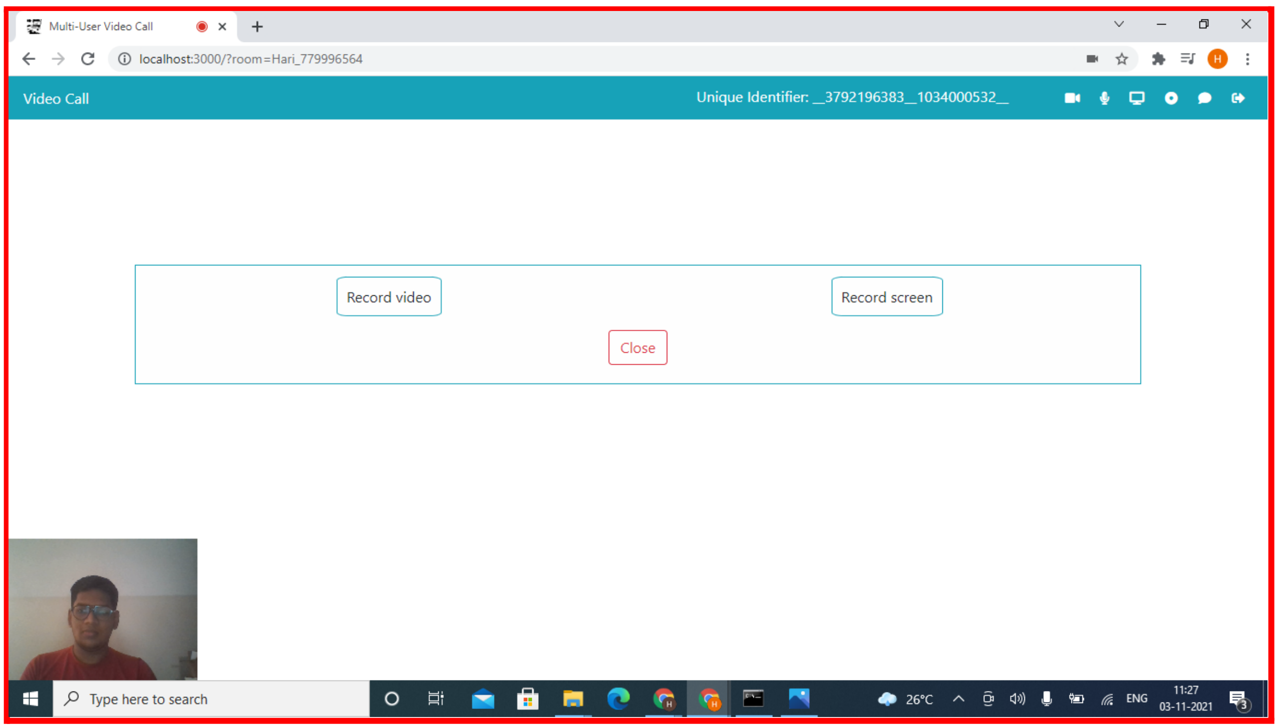





localhost:3000/?room=room1_776181531
Video Call
Unique Identifier: __2548579391__1679879969__


localhost:3000/?room=room1_776181531
Video Call
Choose what to share
http://localhost:3000 wants to share the contents of your screen.
Entire Screen
Window
Brave Tab
Share system audio
Share
Cancel
48579391__1679879969__


Multi-User Video Call
localhost:3000/?room=Hari_779996564
Video Call
Unique Identifier: __3792196383__1034000532__
CHAT
Hi I am Harikrishnaa S
You - 3rd November, 2021 11:23 am
Harikrishnaa Present sir
You - 3rd November, 2021 11:23 am
Type here...
Type here to search
26°C
ENG
11:26
03-11-2021


Multi-User Video Call
localhost:3000/?room=Hari_779996564
Video Call
Unique Identifier: __3792196383__1034000532__
Record video
Record screen
Close
Type here to search
26°C
ENG
11:27
03-11-2021

6.2 RESULT ANALYSIS

A simple WebRTC chat application has been created in Javascript using the NodeJS framework and other JavaScript APIs. The web application has features similar to Google Meet. It provides features such as real-time communication with audio or/and video along with options for disabling and enabling video as well as audio, chat, screen sharing, and meet recording.

The application is run on the localhost and it can be deployed on a server and people from across distant regions can communicate with each other.

6.3 CONCLUSION & FUTURE WORK

Thus a WebRTC web application for real-time communication can be created using NodeJS and socket.io and Javascript APIs.

This project can be further expanded by improving the UI and implementing other features such as Visual effects for the background as in recent WebRTC applications such as Google Meet, Microsoft Teams, etc…

The web application can be deployed on a server and hosted for people across different regions to communicate with each other.

7. REFERENCES

1. https://developer.mozilla.org/en-US/docs/Web/API/WebRTC_API

2. https://webrtc.github.io/webrtc-org/architecture/

3. https://www.oreilly.com/library/view/real-time-communication-with/9781449371869/ch01.html

4. https://web-engineering.info/node/57

5. https://www.tutorialspoint.com/webrtc/webrtc_architecture.htm

6. https://github.com/amirsanni/Video-Call-App-NodeJS