Network Architecture 1

Spring 2020 Semester

Final-Project (Report)

Submitted By

VINEETH MOHAN EDUKULLA (16307571)

HARITHA ANDEM (16315094)

SAI KRISHNA (16315949)

ABINASH BANDA (16298668)

Department of Computer Science & Electrical Engineering

University of Missouri Kansas City

Objective:

Developing a video and screen share application between two clients. Displaying the video and screen of one client on the other client's screen.

The application provides real-time peer-To-Peer streaming between web pages. your browser will directly stream audio & video to other participants' browsers without any server in-between.

Design Flow:

- Client-server tries to connect to STUNT server to get the unique IP address for each client and send IP addresses to fire store database. This address or unique identification token is taken by another client-server to join the call and communicate with each other.
- Client gives access to microphone and screen and sends video and audio data to another client over the network using another receiver client's unique IP address without the help of the server.
- Server can handle two clients at a time on a first come first serve basis and connects them to share screen, video, and audio.
- Server handles continuous streaming of video, audio, and screen sharing data.

Implemented Features:

➔ Implemented User Interface Home page to provide options to share his screen or video.
➔ Implemented code to access microphones and Cameras and screen display data.
➔ Implemented a web page to show his video on the web page.
➔ Implemented code to access desktop screen.
➔ Implemented code to show his screen on the web page.
➔ Implemented code to connect remote users using Web-Rtc library.
➔ Implemented code to retrieve another client's video or desktop screen data and pipelined over the network and showed it on the host client.
➔ Implemented options to create a call or hang up the call at any time between the ongoing call.
➔ Implemented the code to connect to stun server, fire store database when required as per the design flow.

Technologies Used:

Note: In this project, we used VanillaJS, JavaScript, and VanillaJS running on Vite Server using Visual Studio code IDE version 2021.

In this project, the client-server nodes are my local system with IPv6.

Mid Project progress outputs:

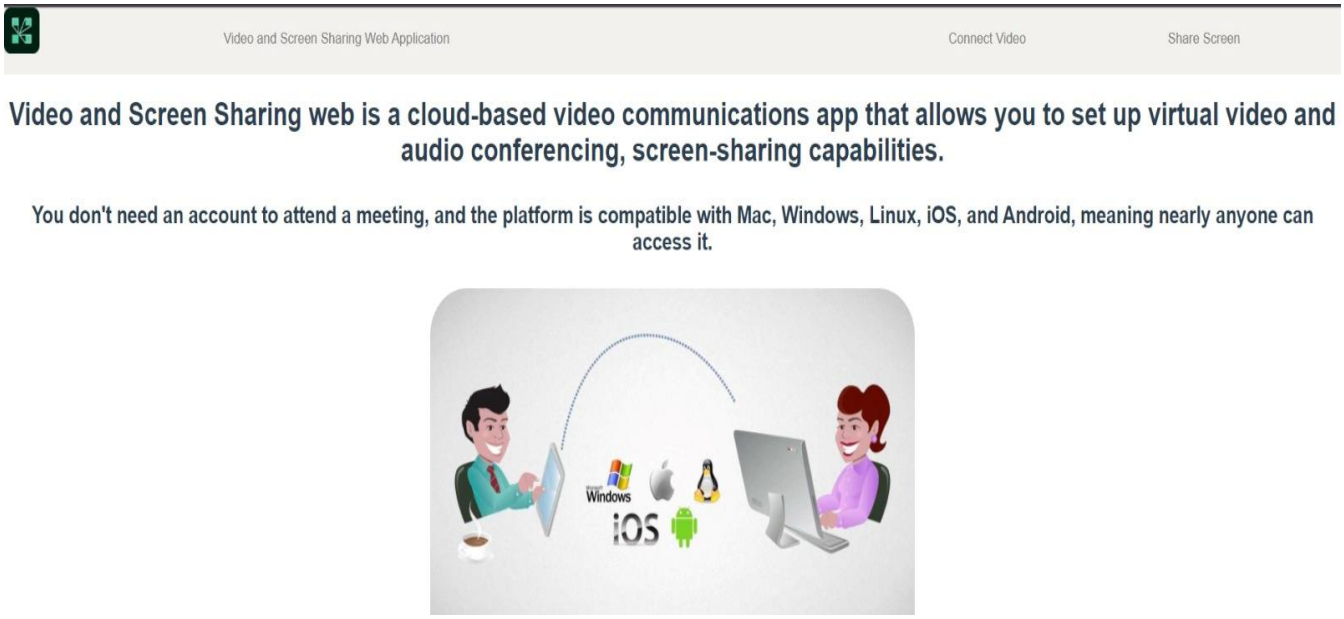
1) **Vite Server log: On starting the client side Vite server, below are the logs of the server.**

```
C:\Users\chakr\Music\webrtc-firebase-demo-main-youtube\webrtc-firebase-demo-main>npm run dev

> webrtc-demo@0.0.0 dev
> vite

Port 3000 is in use, trying another one...

  vite v2.0.5 dev server running at:

  > Network:  http://192.168.86.71:3001/
  > Local:    http://localhost:3001/
```

2) **Client-Side Home Page User Interface:**

Once the server is up, the user will access the home page and he has options to choose to share his video or his screen.

Below is the screenshot of the home page implementation

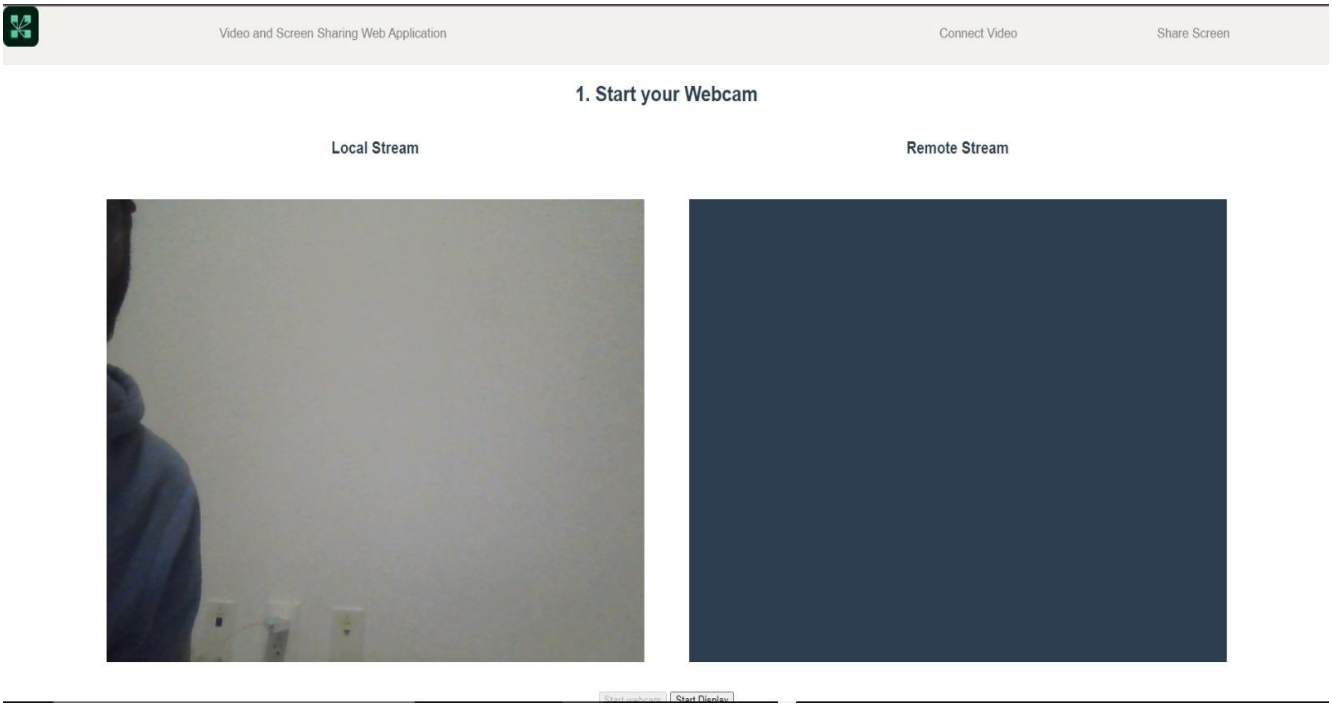


3) Video Sharing Webpage:

If the user selects to share his video button, he will be directed to the webpage as shown in the screenshot.

Once he gives access to the camera and mic, his video and audio are displayed on the webpage as seen in the below screenshot.

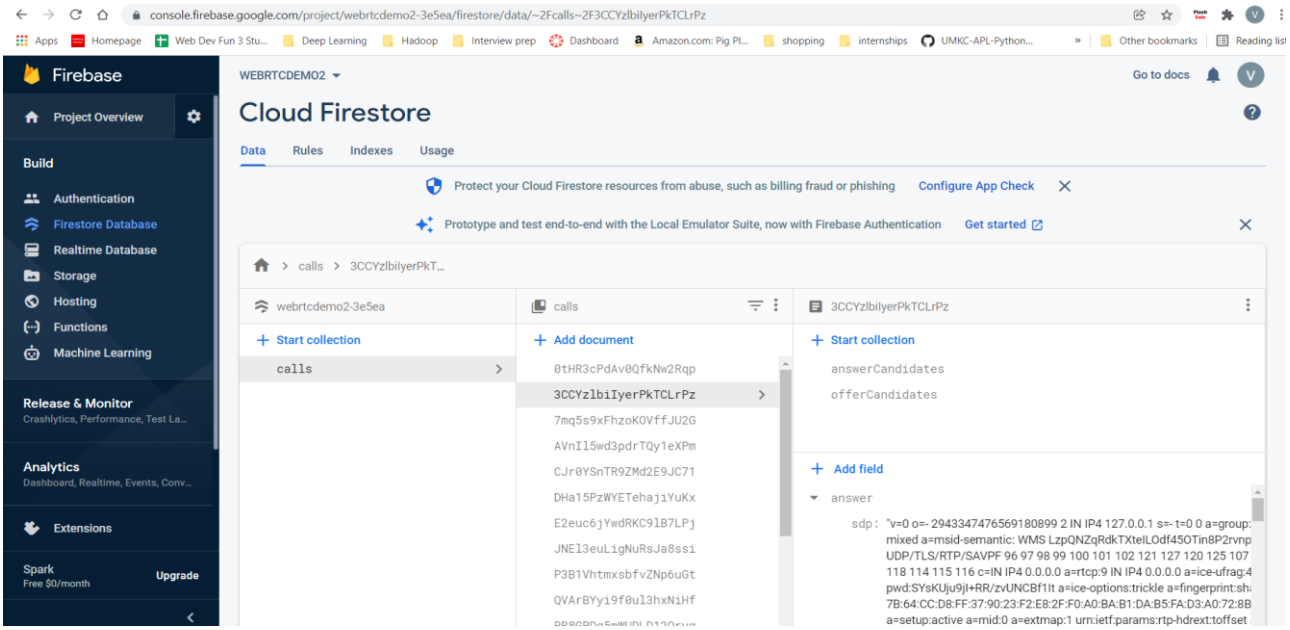
we have developed the JavaScript code to access the camera and showed the continuous data stream using HTML and CSS on the webpage.



Create offer:

Once we create a call or initiate a call, we will connect to the stun server, and that data is pushed into the fire store database.

Below is the screenshot of the unique identification of the client.

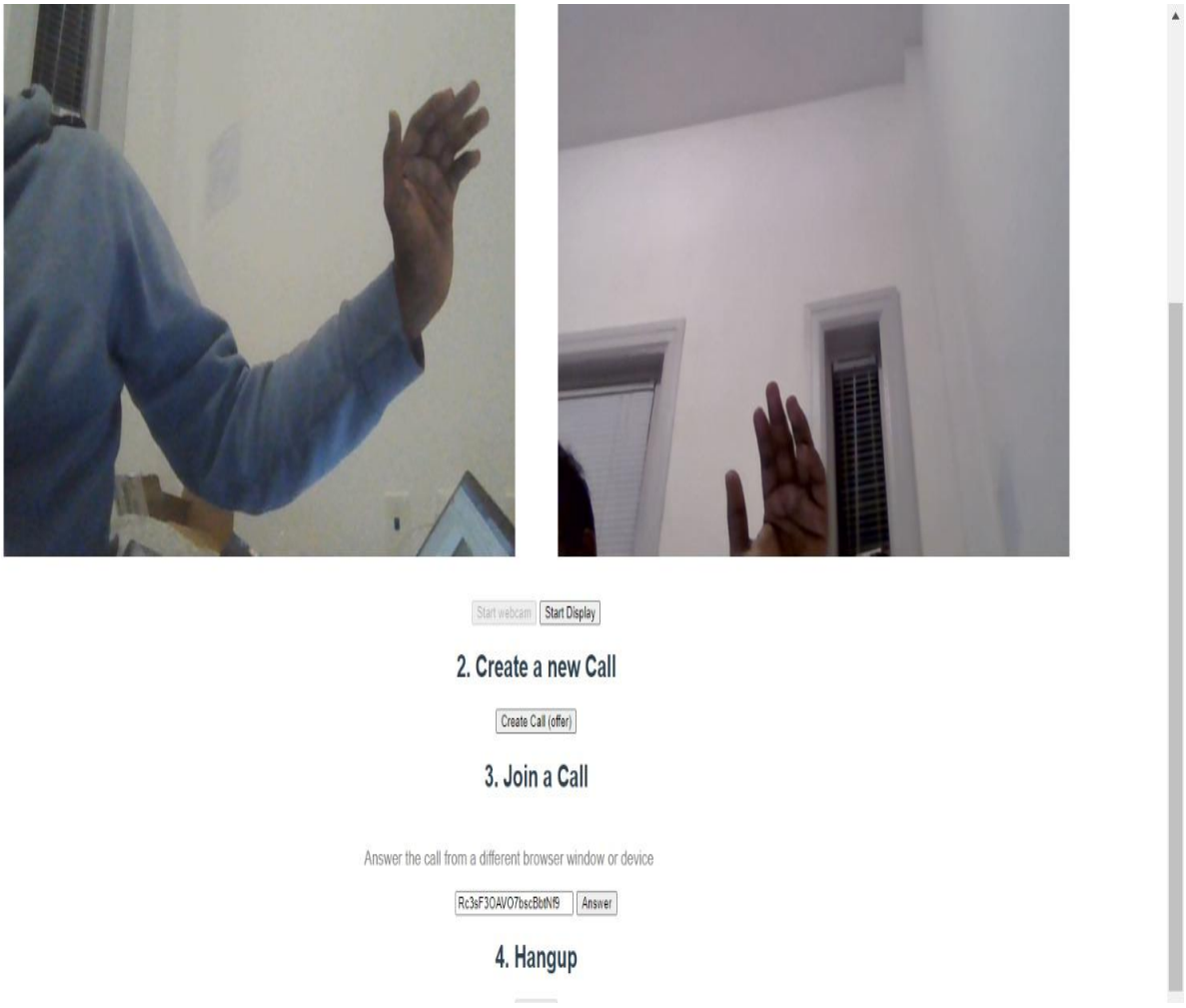


Answer call

This unique identification is used to answer the call.

4) Both clients connected and Sharing Video Webpage:

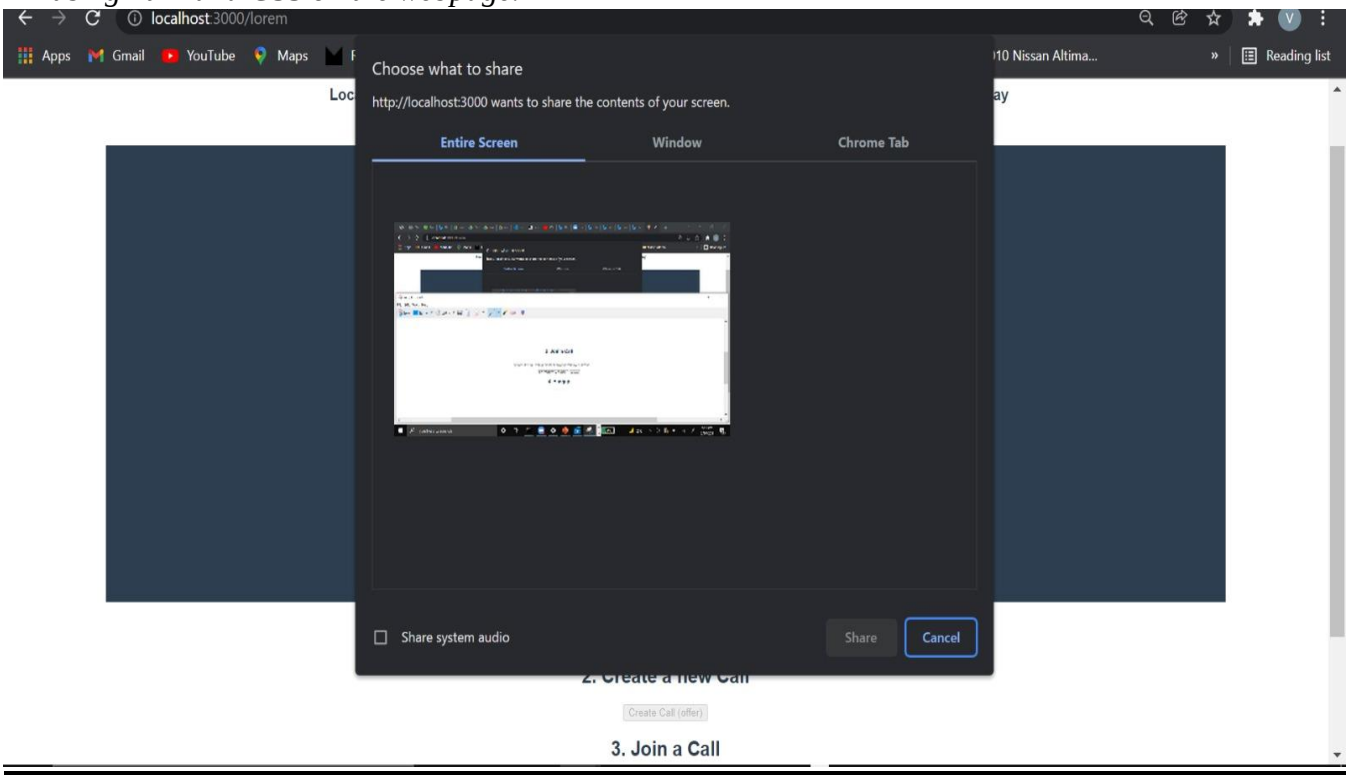
Below is the screenshot which shows two people have connected and started sharing their video. We have provided options to answer the call or initiate the call or hang up the call anytime.



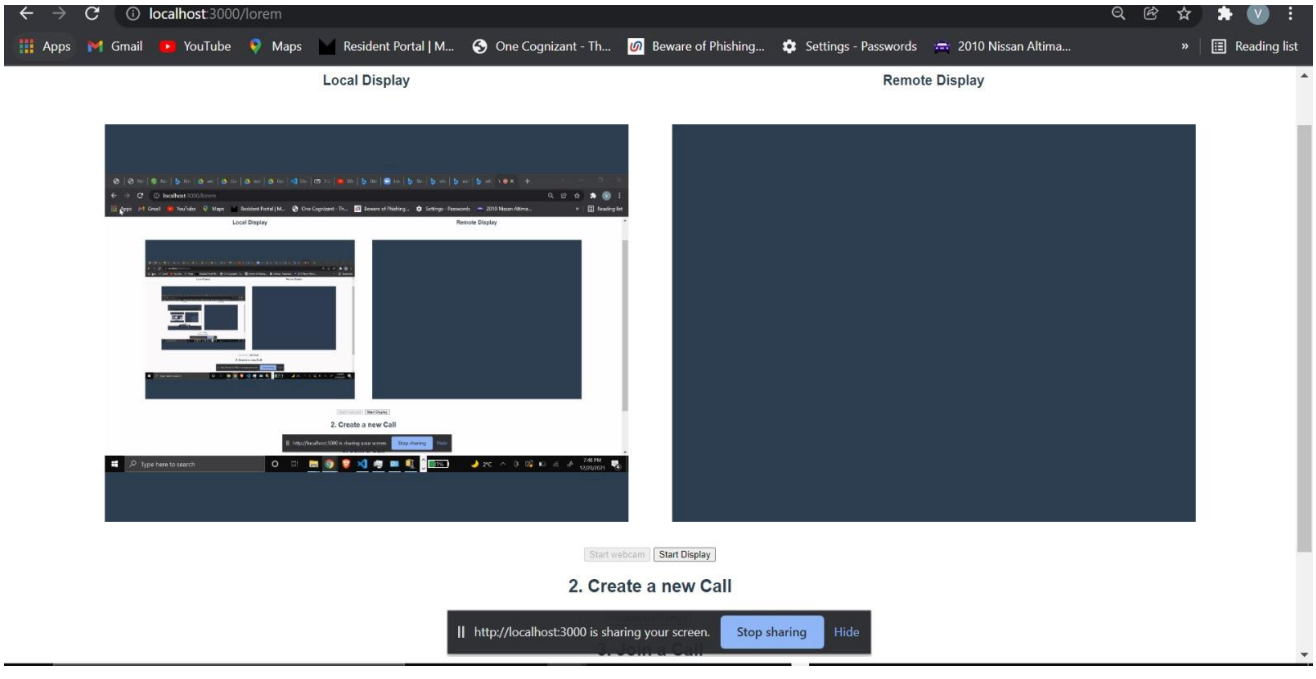
5) Client-side Screen Sharing Webpage:

If the user selects to share his screen button, he will be directed to the webpage as shown in the screenshot. Once he gives access to the screen and particular window, his screen will be displayed on the webpage as seen in the below screenshot.

We have developed the JavaScript code to access the screen and showed the continuous data stream using html and CSS on the webpage.

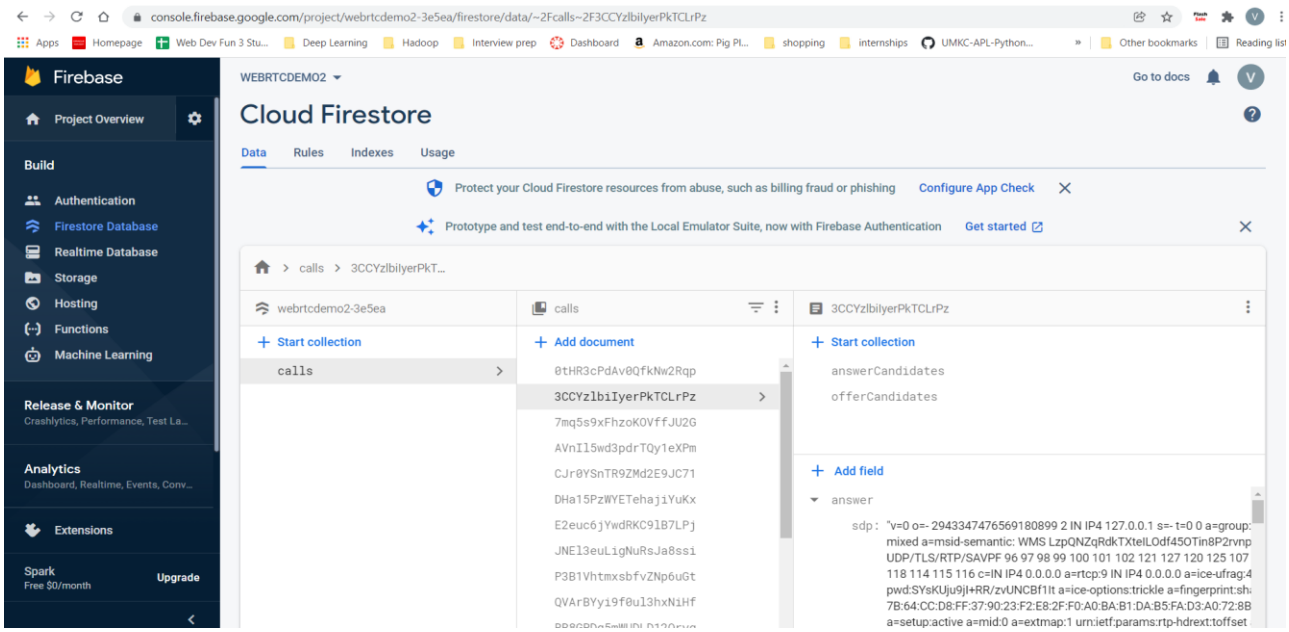


His own screen share is seen on the webpage.



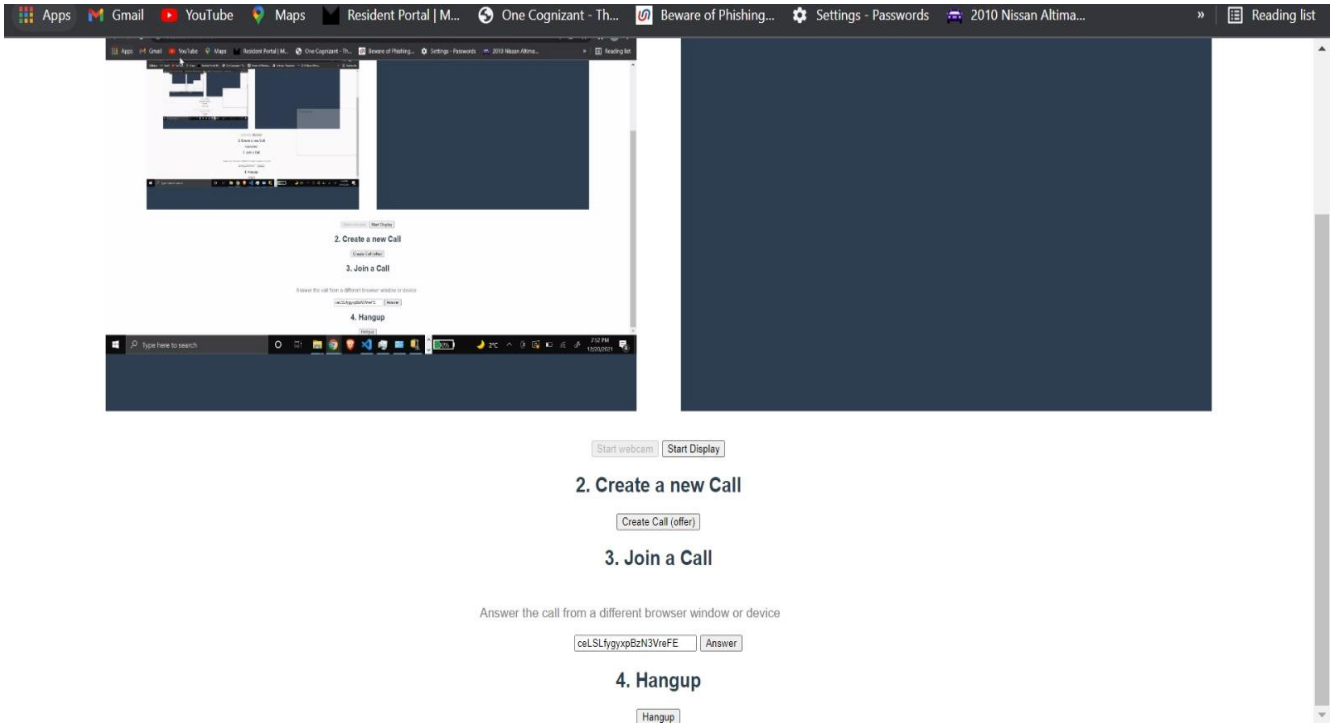
Create Offer:

Once the user selects create call button, the client-server connects to the stun server and gets the unique identification per session, and saves it in the fire store database. The entry is shown in the below screenshot.



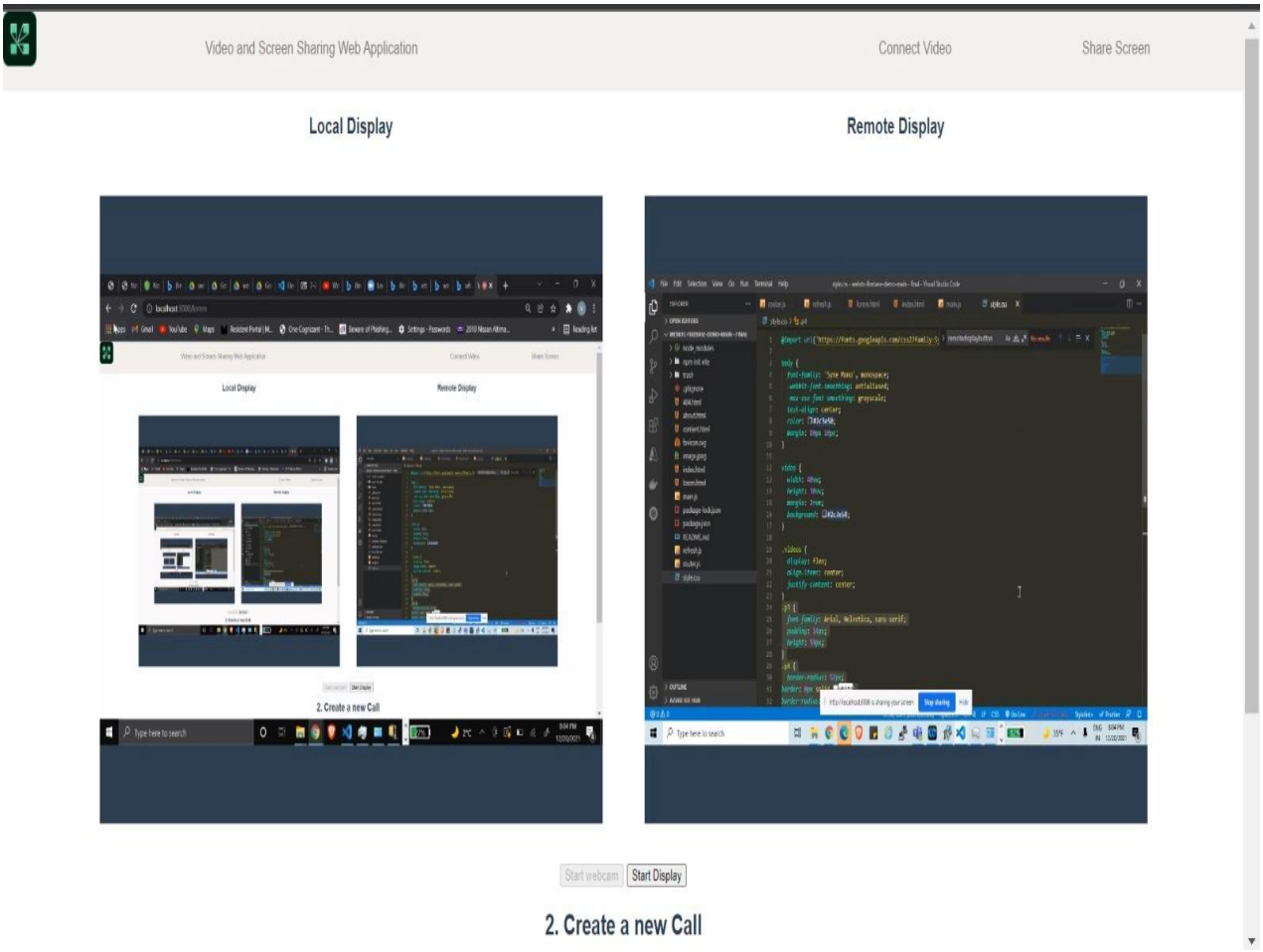
Answer Call:

The unique identification code is used to answer and connect on to the screen sharing call.



6) Both clients connected and Sharing Desktop Screen:

In the below screenshot, both clients got connected and the screen sharing is going on between the clients.



CLIENT-SIDE CODE FOR HOME PAGE:

Below is the code for styling the webpage and providing options to select video share and screen share.

HTML and CSS code:

index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <link rel="icon" type="image/svg+xml" href="favicon.svg" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Video and Screen Sharing Web Application</title>
  </head>
  <body>
    <style>
      .container {
          display: flex;
          background-color: #f3f1ee;
          padding: 0px;
          margin: 0px;
          height: 73px;
      }

      #logo {
          flex : 5;
          align-items: flex-start;
          display: block;
          margin-left: 20px;
          margin-right: 20px;
      }

      p {
          flex: 1;
          align-items: center;
          margin-top: 0px;
          margin-left: -82px;
          padding-left: -26px;
          padding-top: 24px;
          text-decoration: none;
          text-decoration-line: none;
          text-decoration-thickness: initial;
          text-decoration-style: initial;
          text-decoration-color: initial;
```

```
<div class="container">
  <img href="/" onclick="route(); refresh()" style="float: left; width: auto; height: 50px"
       src="https://upload.wikimedia.org/wikipedia/commons/thumb/d/d0/Adobe_Connect_icon_%282020%29.svg/1200px-Adobe_Connect_icon_%282020
  <p></p>
  <a href="/" onclick="route(); refresh()">Video and Screen Sharing Web Application</a>
  <p></p>
  <p></p>
  <a href="/about" onclick="route(); refresh()">Connect Video</a>
  <a href="/lorem" onclick="route(); refresh()">Share Screen</a>
<!--    <p>cards</p>-->
<!--    <p> Insurance</p>-->
<!--    <p>  Online features</p>-->
</div>
<div class="hrcontainer">
  <hr>
  <hr>
  <hr>
  <hr>
  <hr>
  <hr>
  <hr id="borrow">
  <hr id="invest">
  <hr id="cards">
  <hr id="insurance">
  <hr id="online">
</div>
    <div id="main-page"></div>
```

JavaScript code to route to video share and screen share webpages:

router.js

```
const route = (event) => {
    event = event || window.event;
    event.preventDefault();
    window.history.pushState({}, "", event.target.href);
    handleLocation();
};

const routes = {
    404: "404.html",
    "/": "content.html",
    "/about": "about.html",
    "/lorem": "lorem.html",
};

const handleLocation = async () => {
    const path = window.location.pathname;
    const route = routes[path] || routes[404];
    const html = await fetch(route).then((data) => data.text());
    document.getElementById("main-page").innerHTML = html;
    console.log();
    console.log(html);
};

window.onpopstate = handleLocation;
window.route = route;

handleLocation();
```

CLIENT-SIDE USER INTERFACE FOR VIDEO-SHARING WEBPAGE:

Below is the HTML and CSS code to script the web page for video sharing.

about.html

```
<h2>1. Start your Webcam</h2>
    <div class="videos">
      <span>
        <h3>Local Stream</h3>
        <video id="webcamVideo" autoplay playsinline></video>
      </span>
      <span>
        <h3>Remote Stream</h3>
        <video id="remoteVideo" autoplay playsinline></video>
      </span>
      <!-- <span>
        <h3>Local Display</h3>
        <video id="webcamdisplay" autoplay playsinline></video>
      </span>
      <span>
        <h3>Remote Display</h3>
        <video id="remotedisplay" autoplay playsinline></video>
      </span> -->

    </div>
    <button id="webcamButton">Start webcam</button>
    <button id="remotedisplaybutton">Start Display</button>
    <h2>2. Create a new Call</h2>
    <button id="callButton" disabled>Create Call (offer)</button>

    <h2>3. Join a Call</h2>
    <p>Answer the call from a different browser window or device</p>

    <input id="callInput"/>
    <button id="answerButton" disabled>Answer</button>

    <h2>4. Hangup</h2>

    <button id="hangupButton" disabled onclick = "refresh()">Hangup</button>
```

Below is the JavaScript code to connect to Stun Server and store unique identification in the fire store database.

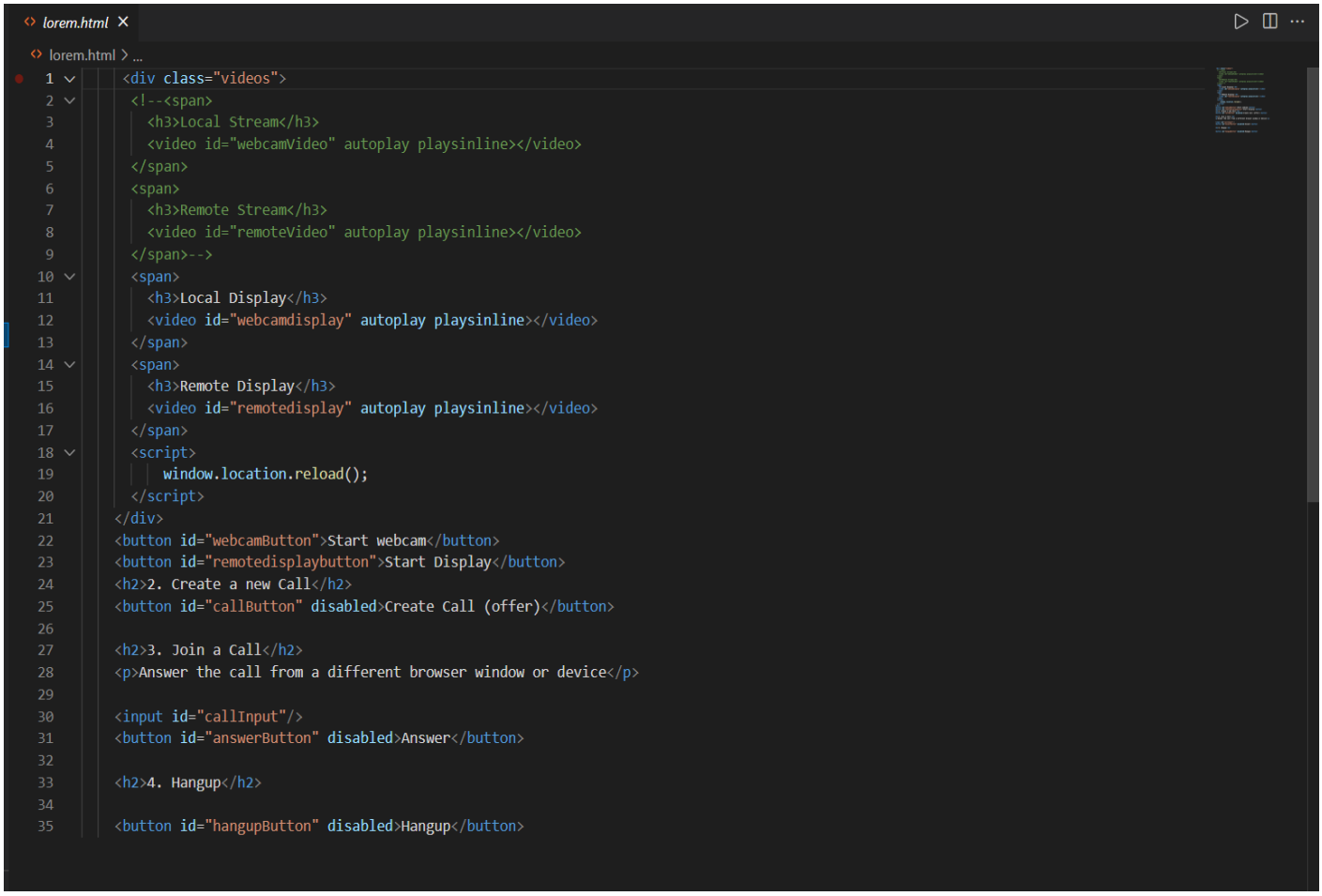
main.js

```
import './style.css';

import firebase from 'firebase/app';
import 'firebase/firestore';

const firebaseConfig = {
  apiKey: "[redacted]zaSyAKmcGocXn6hUyYtcE2w6YB5eH_jeVeov0",
  authDomain: "webrtcdemo2-3e5ea.firebaseapp.com",
  projectId: "webrtcdemo2-3e5ea",
  storageBucket: "webrtcdemo2-3e5ea.appspot.com",
  messagingSenderId: "878339553629",
  appId: "1:878339553629:web:5107524dbd792800e450b4",
  measurementId: "G-FQ5R8SYZ3C"
};

if (!firebase.apps.length) {
  firebase.initializeApp(firebaseConfig);
}
const firestore = firebase.firestore();

const servers = {
  iceServers: [
    {
      urls: ['stun:stun1.l.google.com:19302', 'stun:stun2.l.google.com:19302'],
    },
  ],
  iceCandidatePoolSize: 10,
};

// Global State
const pc = new RTCPeerConnection(servers);
let localStream = null;
let remoteStream = null;

// HTML elements
const webcamButton = document.getElementById('webcamButton');
const webcamVideo = document.getElementById('webcamVideo');
const callButton = document.getElementById('callButton');
```

```
// 1. Setup media sources

webcamButton.onclick = async () => {
  localStream = await navigator.mediaDevices.getUserMedia({ video: true, audio: true });
  remoteStream = new MediaStream();

  // Push tracks from local stream to peer connection
  localStream.getTracks().forEach((track) => {
    pc.addTrack(track, localStream);
  });

  // Pull tracks from remote stream, add to video stream
  pc.ontrack = (event) => {
    event.streams[0].getTracks().forEach((track) => {
      console.log("hello");
      console.log(track);
      remoteStream.addTrack(track);
    });
  };
  console.log("vineeth"+remoteStream);
  webcamVideo.srcObject = localStream;
  remoteVideo.srcObject = remoteStream;

  callButton.disabled = false;
  answerButton.disabled = false;
  webcamButton.disabled = true;
};

// vineeth - setup media resources for display sharing
remotedisplaybutton.onclick = async () => {
  localStream = await navigator.mediaDevices.getDisplayMedia({ video: true, audio: true });
  remoteStream = new MediaStream();
```

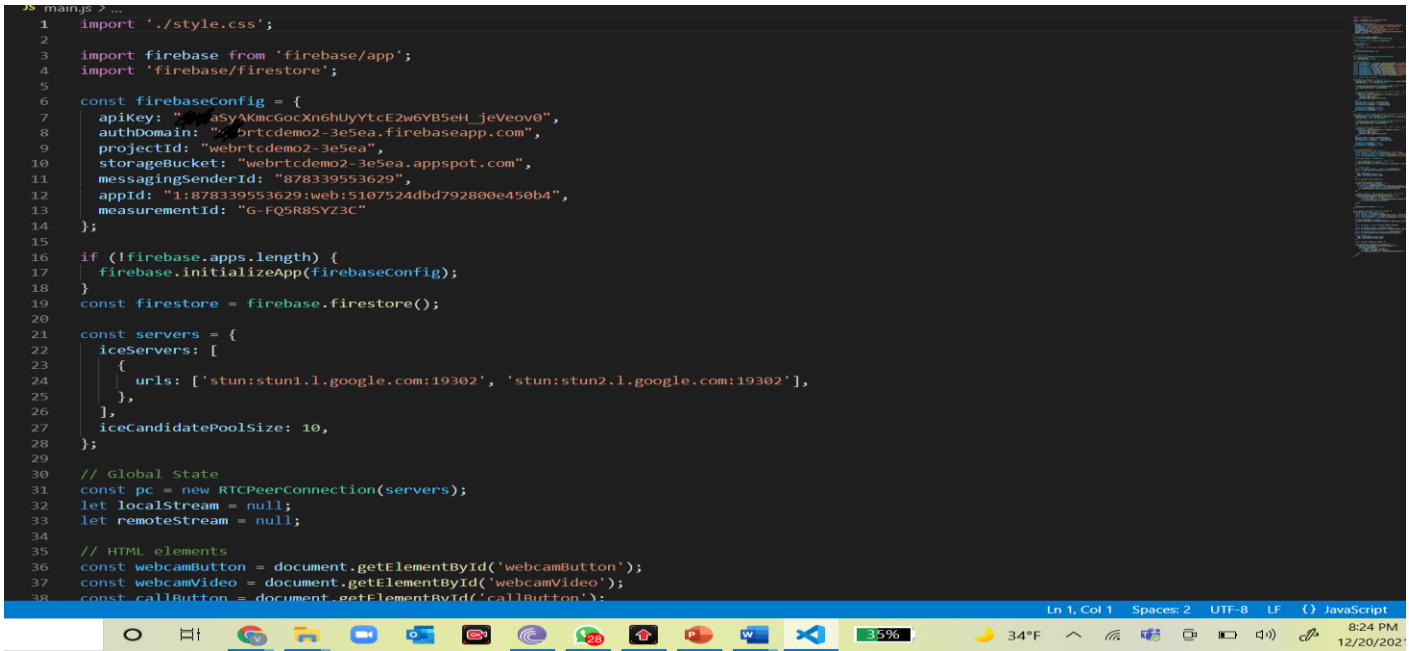
CLIENT-SIDE USER INTERFACE FOR SCREEN-SHARING WEBPAGE:

HTML and CSS code to display the screen sharing on the webpage:



```
<div class="videos">
  <!--<span>
    <h3>Local Stream</h3>
    <video id="webcamVideo" autoplay playsinline></video>
  </span>
  <span>
    <h3>Remote Stream</h3>
    <video id="remoteVideo" autoplay playsinline></video>
  </span>-->
  <span>
    <h3>Local Display</h3>
    <video id="webcamdisplay" autoplay playsinline></video>
  </span>
  <span>
    <h3>Remote Display</h3>
    <video id="remotedisplay" autoplay playsinline></video>
  </span>
  <script>
    window.location.reload();
  </script>
</div>
<button id="webcamButton">Start webcam</button>
<button id="remotedisplaybutton">Start Display</button>
<h2>2. Create a new Call</h2>
<button id="callButton" disabled>Create Call (offer)</button>

<h2>3. Join a Call</h2>
<p>Answer the call from a different browser window or device</p>

<input id="callInput"/>
<button id="answerButton" disabled>Answer</button>

<h2>4. Hangup</h2>

<button id="hangupButton" disabled>Hangup</button>
```

Java script code to connect to stun server and store the unique identification in the fire store database.



```
import './style.css';

import firebase from 'firebase/app';
import 'firebase/firestore';

const firebaseConfig = {
  apiKey: "[redacted]",
  authDomain: "[redacted]rtcdemo2-3e5ea.firebaseapp.com",
  projectId: "webrtcdemo2-3e5ea",
  storageBucket: "webrtcdemo2-3e5ea.appspot.com",
  messagingSenderId: "878339553629",
  appId: "1:878339553629:web:5107524dbd792800e450b4",
  measurementId: "G-FQ5R8SYZ3C"
};

if (!firebase.apps.length) {
  firebase.initializeApp(firebaseConfig);
}
const firestore = firebase.firestore();

const servers = {
  iceServers: [
    {
      urls: ['stun:stun1.l.google.com:19302', 'stun:stun2.l.google.com:19302'],
    },
  ],
  iceCandidatePoolSize: 10,
};

// Global State
const pc = new RTCPeerConnection(servers);
let localStream = null;
let remoteStream = null;

// HTML elements
const webcamButton = document.getElementById('webcamButton');
const webcamVideo = document.getElementById('webcamVideo');
const callButton = document.getElementById('callButton');
```

```
// vineeth - setup media resources for display sharing
remotedisplaybutton.onclick = async () => {
  localStream = await navigator.mediaDevices.getDisplayMedia({ video: true, audio: true });
  remoteStream = new MediaStream();

  // Push tracks from local stream to peer connection
  localStream.getTracks().forEach((track) => {
    pc.addTrack(track, localStream);
  });

  // Pull tracks from remote stream, add to video stream
  pc.ontrack = (event) => {
    event.streams[0].getTracks().forEach((track) => {
      console.log("hello");
      console.log(track);
      remoteStream.addTrack(track);
    });
  };
  console.log("vineeth"+remoteStream);
  webcamdisplay.srcObject = localStream;
  remotedisplay.srcObject = remoteStream;

  callButton.disabled = false;
  answerButton.disabled = false;
  webcamButton.disabled = true;
};
```

Java script code to create and answer the call

```
// 2. Create an offer
callButton.onclick = async () => {
  // Reference Firestore collections for signaling
  const callDoc = firestore.collection('calls').doc();
  const offerCandidates = callDoc.collection('offerCandidates');
  const answerCandidates = callDoc.collection('answerCandidates');

  callInput.value = callDoc.id;

  // Get candidates for caller, save to db
  pc.onicecandidate = (event) => {
    event.candidate && offerCandidates.add(event.candidate.toJSON());
  };

  // Create offer
  const offerDescription = await pc.createOffer();
  await pc.setLocalDescription(offerDescription);

  const offer = {
    sdp: offerDescription.sdp,
    type: offerDescription.type,
  };

  await callDoc.set({ offer });

  // Listen for remote answer
  callDoc.onSnapshot((snapshot) => {
    const data = snapshot.data();
    if (!pc.currentRemoteDescription && data?.answer) {
      const answerDescription = new RTCSessionDescription(data.answer);
      pc.setRemoteDescription(answerDescription);
    }
  });
```

```
        const candidate = new RTCIceCandidate(change.doc.data());
        pc.addIceCandidate(candidate);
      }
    });
  });

  hangupButton.disabled = false;
};

// 3. Answer the call with the unique ID
answerButton.onclick = async () => {
  const callId = callInput.value;
  const callDoc = firestore.collection('calls').doc(callId);
  const answerCandidates = callDoc.collection('answerCandidates');
  const offerCandidates = callDoc.collection('offerCandidates');

  pc.onicecandidate = (event) => {
    event.candidate && answerCandidates.add(event.candidate.toJSON());
  };

  const callData = (await callDoc.get()).data();

  const offerDescription = callData.offer;
```

References:

https://webrtc.org/

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

https://vitejs.dev/guide/

https://firebase.google.com/products/firestore?gclid=Cj0KCQiA8ICOBhDmARIsAEGI6o0e32HazMClJn_giENN2VEqCfJTXisvnHPHI2vImqndQZIfZnrNqgcaAnHoEALw_wcB&gclsrc=aw.ds

https://medium.com/@otacorporation0520/set-up-stun-turn-server-for-the-video-chat-app-ee8dc6fb9b15

https://medium.com/firebase-developers/cloud-firestore-basics-in-android-98ccabbc949b

https://www.wowza.com/blog/webrtc-encryption-and-security