International Journal of
Computational Learning & Intelligence
Vol. 03, Issue 01 (January) 2024

RESEARCH ARTICLE OPEN ACCESS

Enhancing Real-Time Communication: A WebRTC-Based Video Chat Application

Soumya Dhanappa . Sahnaz Parveen . Sanya Yadav . Naveen Chandra Gowda

School of Computer Science and Engineering
REVA University, Bengaluru, India

DOI: **10.5281/zenodo.10634598**
Received: 21 October 2023 / Revised: 22 November 2023 / Accepted: 29 December 2023
©Milestone Research Publications, Part of CLOCKSS archiving

Abstract – Video chat applications are widely used for various purposes, such as social, educational, and professional communication. However, developing a video chat application can be challenging, as it requires handling complex issues such as network latency, bandwidth, security, and compatibility. Web Real-Time Communication (WebRTC) is a cutting-edge technology that enables peer-to-peer real-time communication without the need for additional software or plugins. WebRTC is a set of APIs and protocols that allow browsers and mobile applications to exchange audio, video, and data streams directly, without relying on intermediate servers or platforms. In this paper, we propose a video chat application that uses WebRTC and Node.js, a JavaScript runtime environment that allows for fast and scalable network applications3. We describe the design and implementation of our application, which consists of a web-based user interface, a signaling server, and a peer connection manager. We also evaluate the performance and usability of our application and compare it with other existing video chat applications. We demonstrate that our application can provide high-quality, secure, and user-friendly video chat services using WebRTC and Node.js

Index Terms – Web Real-Time Communication, Teleconferencing, Peer-To-Peer, Media Stream, RTCPeerConnection.

I. INTRODUCTION

The use of video chat applications has become increasingly popular in recent years, with millions of users worldwide relying on these platforms for communication and collaboration. However, the development of such applications can be challenging, particularly when it comes to ensuring their reliability and performance. The importance of this topic lies in its potential to address the limitations of existing video chat applications, particularly in terms of their reliability and performance. By exploring the use of WebRTC (Web Real Time Communication), Firebase platform, and Vite tool in the development of video chat applications, this research aims to contribute to the advancement of this field and provide valuable insights for beginners, developers, and users alike. We will explore the process of deploying a Vite application using Firebase hosting. Vite, a fast and modern build tool for JavaScript applications, combined with Firebase's powerful hosting platform, provides beginners with an effortless way to deploy their applications.

Fig. 1: Integrate Video and chat functionality with WebRTC, Socket.IO & Node.js[12]

II. LITERATURE SURVEY

Previous research on video chat applications has primarily focused on their design and user experience, with less attention given to their development and implementation. However, there is a growing interest in the use of WebRTC and Firebase platform in the development of real-time applications, including video chat applications. Despite the growing interest in the use of WebRTC and Firebase in the development of video chat applications, there is a lack of research on the specific challenges and opportunities associated with this approach. There is a need for further investigation into the technical and practical aspects of implementing these technologies in the development of reliable and performant video chat applications. The development of reliable and performant video chat applications is of critical importance, particularly in today's increasingly connected world. By exploring the use of WebRTC using Firebase platform in the development of such applications, this research aims to address the limitations of existing approaches and provide valuable insights for beginners, developers, and users alike. The aim of this research is to investigate the potential of WebRTC, Firebase, and Vite tools in the development of reliable and performant video chat applications, and to identify the challenges and opportunities associated with this approach. Based on previous research and practical experience, it is hypothesized that the use of WebRTC and Firebase platform in the development of video chat applications using peer to peer service can improve their reliability and performance, and that this approach can address some of the limitations of existing video chat applications.

It address the signaling issue or the use of a third-party service (Licode-Erizo) to run the application [12]. Moreover, and point out that using MCU is very expensive and consumes a lot of bandwidth. Some video conferencing codecs may support a limited number of multipoint connections (e.g. up to 4 users), but this may not be enough for larger groups. Therefore, finding a scalable and cost-effective solution for WebRTC signaling and multi-party communication remains an open challenge [8]. Based on the related work reviewed above, it can be concluded that signaling between browser-to-browser and server is not standardized in WebRTC.

III. PROPOSED WORK

It works like this the first peer will create an offer asking for another peer to connect to them this will result in an STP object or session description protocol which contains information describing the peer-to-peer connection like the video codes timing and so on. The data then will be saved in a server where it can be read by another peer to answer the call which is achieved by creating an STP answer and writing that to the server this process is known as SIGNALING and it is handled by their party server. The signaling server allows the two parties to securely exchange the connection data but never touches the media that's actually transmitted between the peers .But here's when the things get tricky most devices in the real world sit behind firewalls and Ip addresses constantly change , thanks to network address translation this makes peer to peer connection complicated from a networking standpoint but luckily there's a standard called INTERACTIVE CONNECTIVITY ESTABLISHMENT or ICE which helps the clients coordinate the discovery of their public facing Ip addresses . Now both peers will generate a list of ICE candidates which contain Ip address and port which peer one can use to connect to peer two. In the background WebRTC will do this by making a series of requests to a STUN SERVER. A stun server is not something you need to set up on your own because there are many options out there from reliable sources like google. Each peer will save their ICE candidate in the database where they can be read by the other peer. The algorithm will then automatically determine which candidate is best and at which point real-time media can begin flowing between the two peers.

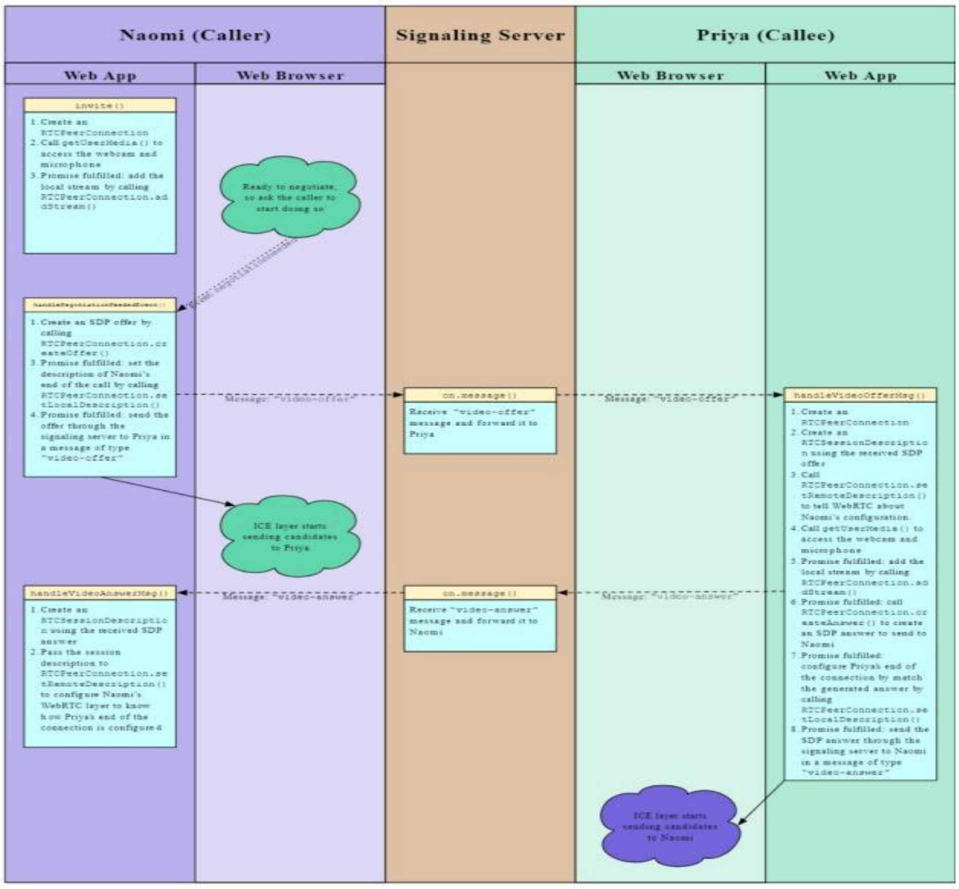
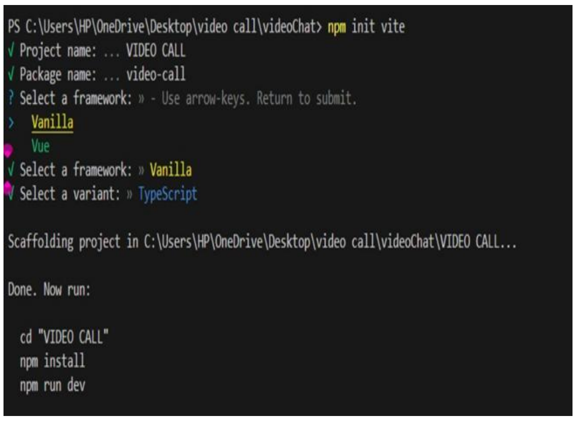

Naomi (Caller)
Signaling Server
Priya (Callee)
Web App
Web Browser
Web Browser
Web App
invite()
1. Create an RTCPeerConnection
2. Call getUserMedia() to access the webcam and microphone
3. Promise fulfilled: add the local stream by calling RTCPeerConnection.addStream()
Ready to negotiate, so ask the caller to start doing so
handleNegotiationNeededEvent()
1. Create an SDP offer by calling RTCPeerConnection.createOffer()
3. Promise fulfilled: set the description of Naomi's end of the call by calling RTCPeerConnection.setLocalDescription()
4. Promise fulfilled: send the offer through the signaling server to Priya in a message of type "video-offer"
Message: "video-offer"
on.message()
Receive "video-offer" message and forward it to Priya
Message: "video-offer"
handleVideoOfferMsg()
1. Create an RTCPeerConnection
2. Create an RTCSessionDescription using the received SDP offer
3. Call RTCPeerConnection.setRemoteDescription() to tell WebRTC about Naomi's configuration.
4. Call getUserMedia() to access the webcam and microphone
5. Promise fulfilled: add the local stream by calling RTCPeerConnection.addStream()
6. Promise fulfilled: call RTCPeerConnection.createAnswer() to create an SDP answer to send to Naomi
7. Promise fulfilled: configure Priya's end of the connection by match the generated answer by calling RTCPeerConnection.setLocalDescription()
8. Promise fulfilled: send the SDP answer through the signaling server to Naomi in a message of type "video-answer"
ICE layer starts sending candidates to Priya
handleVideoAnswerMsg()
1. Create an RTCSessionDescription using the received SDP answer
2. Pass the session description to RTCPeerConnection.setRemoteDescription() to configure Naomi's WebRTC layer to know how Priya's end of the connection is configured
Message: "video-answer"
on.message()
Receive "video-answer" message and forward it to Naomi
Message: "video-answer"
ICE layer starts sending candidates to Naomi

Fig. 2: Breakdown of WebRTC signaling process

IV. IMPLEMENTATION AND RESULTS

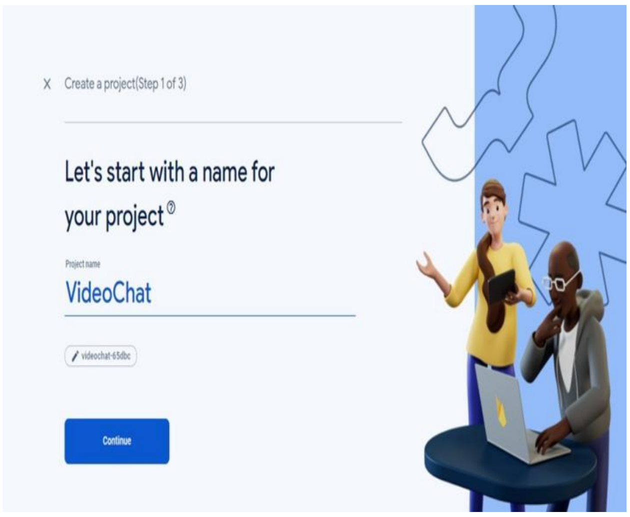
For the implementation of this project, we need to install vite tool in our folder. So, we need to type the Command: “npm init vite” as shown in terminal (Fig.3). Give project name and package name as shown in below figure (3). After naming the project, in the explorer the folder gets created with the required node modules. Then after giving project name, we can specify the specific framework required and as well as the variant like TypeScript, JavaScript etc. Then change the folder and move to VIDEO CALL. Now run command npm install: This command installs a package and any packages that it depends on. If the package has a package-lock, or an shrink wrap file, or a yarn lock file, the installation of dependencies will be driven by that, respecting the following order of precedence: npm-shrinkwrap.json.



```
PS C:\Users\HP\OneDrive\Desktop\video call\videoChat> npm init vite
√ Project name: ... VIDEO CALL
√ Package name: ... video-call
? Select a framework: » - Use arrow-keys. Return to submit.
>   Vanilla
    Vue
√ Select a framework: » Vanilla
√ Select a variant: » TypeScript

Scaffolding project in C:\Users\HP\OneDrive\Desktop\video call\videoChat\VIDEO CALL...

Done. Now run:

  cd "VIDEO CALL"
  npm install
  npm run dev
```

Fig. 3: Terminals

Next step before running the file we need to set up the firebase platform as it helps to make calls and provides an interface and configuration. So, in the browser type www.firebase.com .Now in Firebase create a project.



X Create a project(Step 1 of 3)

Let's start with a name for your project

Project name

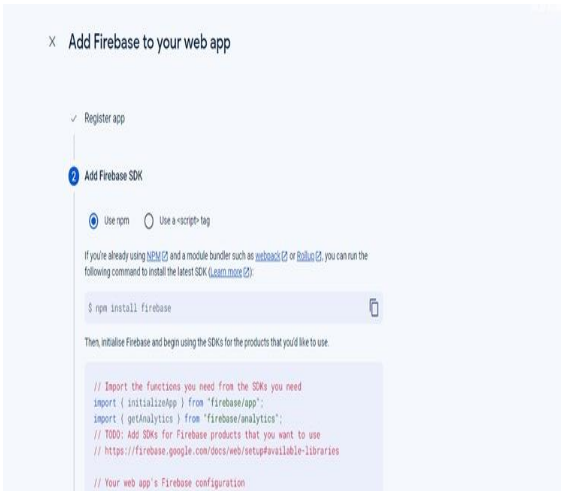
VideoChat

videochat-65dbc

Continue

Fig. 4: Fire Base

Next step after creating a project we need to connect firebase to your web app. Follow the given steps as given in fig.5 and fig.6.



Add Firebase to your web app

✓ Register app

2 Add Firebase SDK

[✓ Use npm] Use a <script> tag

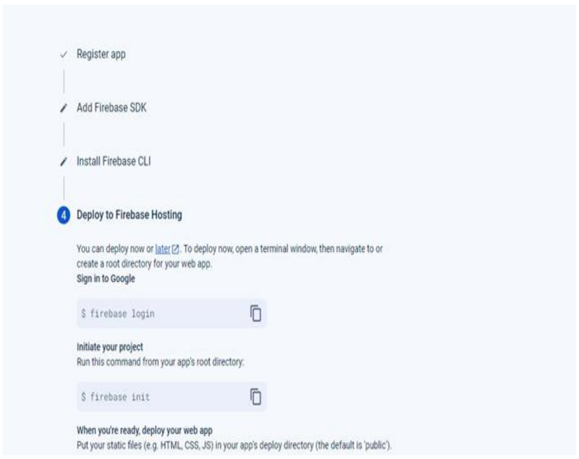
If you're already using NPM and a module bundler such as webpack or Rollup, you can run the following command to install the latest SDK (Learn more):

```
$ npm install firebase
```

Then, initialise Firebase and begin using the SDKs for the products that you'd like to use.

```
// Import the functions you need from the SDKs you need
import { initializeApp } from "firebase/app";
import { getAnalytics } from "firebase/analytics";
// TODO: Add SDKs for Firebase products that you want to use
// https://firebase.google.com/docs/web/setup#available-libraries

// Your web app's Firebase configuration
```

Fig.5. Fire Base



✓ Register app

Add Firebase SDK

Install Firebase CLI

4 Deploy to Firebase Hosting

You can deploy now or later. To deploy now, open a terminal window, then navigate to or create a root directory for your web app.

Sign in to Google

```
$ firebase login
```

Initiate your project

Run this command from your app's root directory:

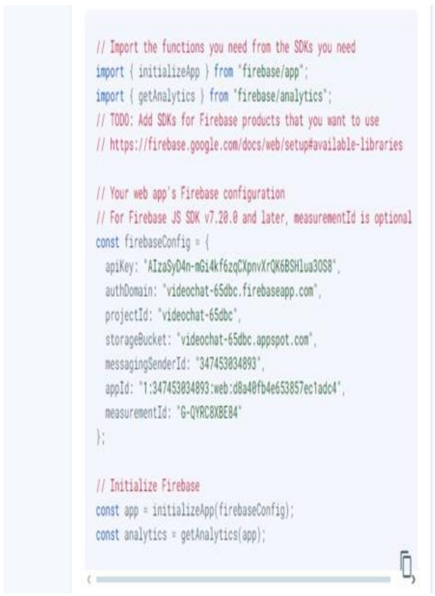
```
$ firebase init
```

When you're ready, deploy your web app

Put your static files (e.g. HTML, CSS, JS) in your app's deploy directory (the default is 'public').

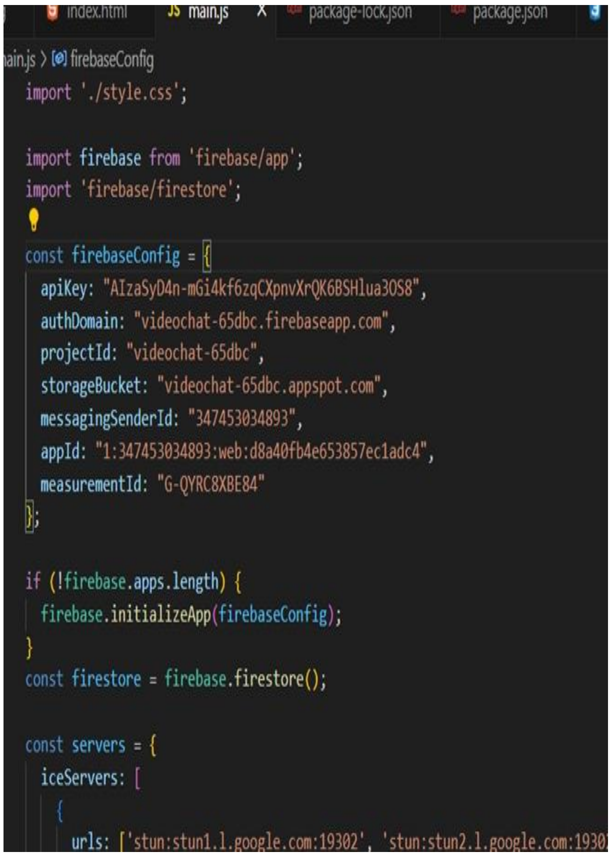
Fig.6. FireBase

Now you need to copy the code from firebaseconfig (Fig.7) and paste it into the main.js file as shown below Fig.8:



```
// Import the functions you need from the SDKs you need
import { initializeApp } from "firebase/app";
import { getAnalytics } from "firebase/analytics";
// TODO: Add SDKs for Firebase products that you want to use
// https://firebase.google.com/docs/web/setup#available-libraries

// Your web app's Firebase configuration
// For Firebase JS SDK v7.20.0 and later, measurementId is optional
const firebaseConfig = {
  apiKey: "AIzaSyD4n-mGi4kf6zqCXpnvXrQK6BSHlua3OS8",
  authDomain: "videochat-65dbc.firebaseapp.com",
  projectId: "videochat-65dbc",
  storageBucket: "videochat-65dbc.appspot.com",
  messagingSenderId: "347453034893",
  appId: "1:347453034893:web:d8a40fb4e653857ec1adc4",
  measurementId: "G-QYRC8XBE84"
};

// Initialize Firebase
const app = initializeApp(firebaseConfig);
const analytics = getAnalytics(app);
```

Fig.7. FireBase



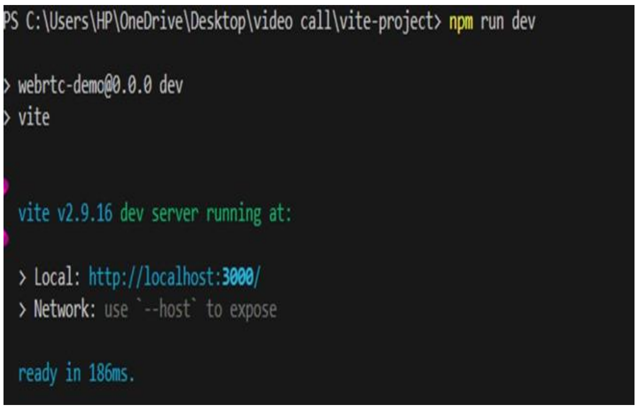
index.html | main.js | package-lock.json | package.json

ain.js > firebaseConfig

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

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

const firebaseConfig = {
  apiKey: "AIzaSyD4n-mGi4kf6zqCXpnvXrQK6BSHlua30S8",
  authDomain: "videochat-65dbc.firebaseapp.com",
  projectId: "videochat-65dbc",
  storageBucket: "videochat-65dbc.appspot.com",
  messagingSenderId: "347453034893",
  appId: "1:347453034893:web:d8a40fb4e653857ec1adc4",
  measurementId: "G-QYRC8XBE84"
};

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:1930
```

Fig.8. Terminals

Now in terminal run the command: npm run dev.



```
PS C:\Users\HP\OneDrive\Desktop\video call\vite-project> npm run dev

> webrtc-demo@0.0.0 dev
> vite


  vite v2.9.16 dev server running at:

  > Local: http://localhost:3000/
  > Network: use `--host` to expose

  ready in 186ms.
```

Fig.9. Terminals

Next click on the http link shown in the above Fig.9:

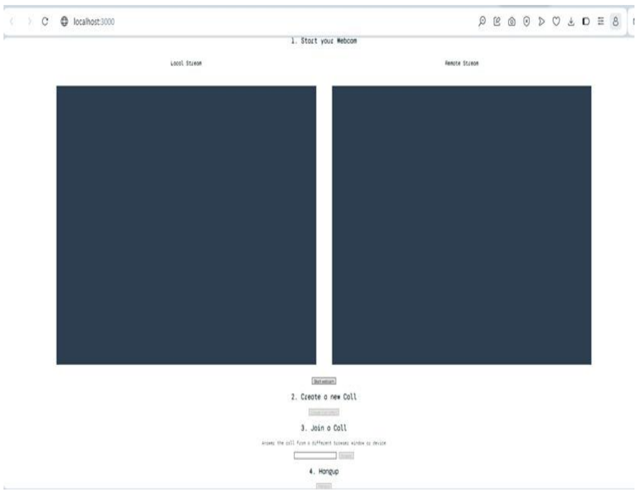
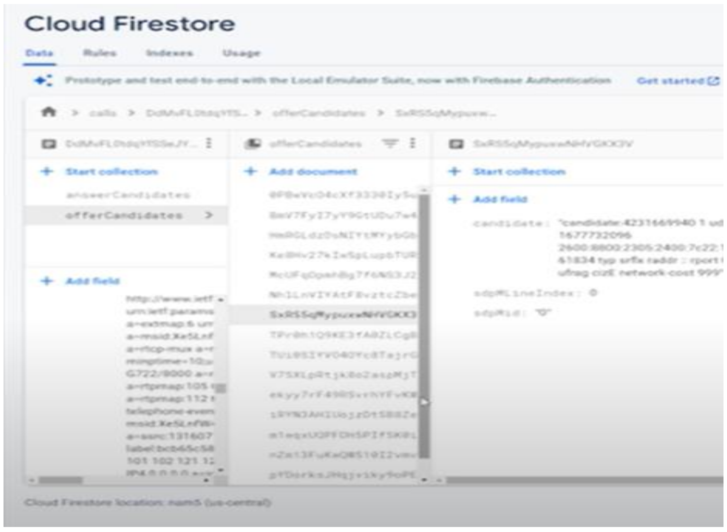


Fig.10. Web Browser

We can see there are two cameras local stream and remote stream and an option to create new call and to join a call. We can join a call using another http page or device. Before we need to create a firestore database to make the local stream and remote stream work. Create a firestore database as shown in the below Fig.11.



candidate: "candidate:4231669940 1 udp 1677732096 2600:8800:2305:2400:7c22:1 61834 typ srflx raddr :: rport ufrag cizE network-cost 999"
sdpMLineIndex: 0
sdpMid: "0"

Cloud Firestore location: nam5 (us-central)

Fig.11. FireStore Database in Firebase

Add two collections called answer Candidate for local stream user and after Candidate for remote stream user (fig.11). Now from another browser the remote stream answers calls using the id created and joins the call.

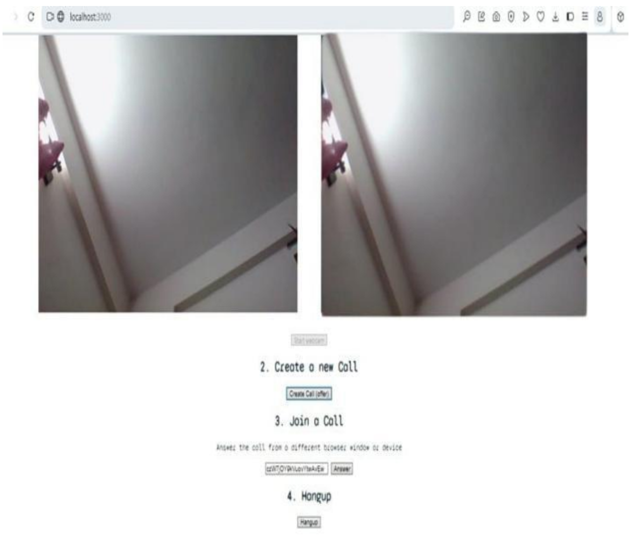


Fig.12. Web Browser

V. CONCLUSION

In conclusion, the project successfully leverages WebRTC to enable real-time audio and video potential to build applications similar to Zoom or Skype. The implementation focuses on a 1-to-1 video chat scenario, where peers communicate directly with each other, eliminating the need for an intermediary server to handle video content. One of the key features of the project is the utilization of a third-party server for signaling purposes. This server stores shared data for stream negotiation, facilitating the exchange of information between peers. Firebase is specifically chosen for this role due to its seamless integration with WebRTC. The decision to use Firebase is justified by its ability to handle real-time updates to the database, ensuring efficient communication between peers.

By adopting WebRTC and integrating it with Firebase for signaling, the project not only demonstrates the technical capabilities of enabling peer-to-peer communication but also highlights the importance of a reliable and responsive signaling mechanism. This approach enhances the overall user experience by providing a smooth and uninterrupted real-time communication channel. In summary, the project effectively combines WebRTC for direct peer-to-peer communication and Firebase for efficient signaling, showcasing a practical and scalable solution for building real-time audio/video communication applications on the web.

REFERENCES

1. Damayanti, F. U. (2018). Research of web real-time communication-The unified communication platform using node. js signaling server. *Journal of Applied Information, Communication and Technology*, *5*(2), 53-61.
2. De Groef, W., Subramanian, D., Johns, M., Piessens, F., & Desmet, L. (2016, April). Ensuring endpoint authenticity in webrtc peer-to-peer communication. In *Proceedings of the 31st Annual ACM Symposium on Applied Computing* (pp. 2103-2110).
3. Ouya, S., Seyed, C., Mbacke, A. B., Mendy, G., & Niang, I. (2015, December). WebRTC platform proposition as a support to the educational system of universities in a limited Internet connection context. In *2015 5th World Congress on Information and Communication Technologies (WICT)* (pp. 47-52). IEEE.

4. Sredojev, B., Samardzija, D., & Posarac, D. (2015, May). WebRTC technology overview and signaling solution design and implementation. In *2015 38th international convention on information and communication technology, electronics and microelectronics (MIPRO)* (pp. 1006-1009). IEEE.
5. Jang-Jaccard, J., Nepal, S., Celler, B., & Yan, B. (2016). WebRTC-based video conferencing service for telehealth. *Computing*, *98*(1-2), 169-193.
6. Ahmed, S. T., Sreedhar Kumar, S., Anusha, B., Bhumika, P., Gunashree, M., & Ishwarya, B. (2020). A generalized study on data mining and clustering algorithms. *New Trends in Computational Vision and Bio-inspired Computing: Selected works presented at the ICCVBIC 2018, Coimbatore, India*, 1121-1129.
7. Carullo, G., Tambasco, M., Di Mauro, M., & Longo, M. (2016, January). A performance evaluation of WebRTC over LTE. In *2016 12th Annual Conference on Wireless On-demand Network Systems and Services (WONS)* (pp. 1-6). IEEE.
8. Liao, Y., Wang, Z., & Luo, Y. (2016, October). The design and implementation of a WebRTC based online video teaching system. In *2016 IEEE Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC)* (pp. 137-140). IEEE.
9. Emmanuel, E. A., & Dirting, B. D. (2017). A peer-to-peer architecture for real-time communication using Webrtc. *J Multidiscipl Eng Sci Stud (JMESS)*, *3*(4), 1671-1683.
10. Chiang, C. Y., Chen, Y. L., Tsai, P. S., & Yuan, S. M. (2014, June). A video conferencing system based on WebRTC for seniors. In *2014 International Conference on Trustworthy Systems and their Applications* (pp. 51-56). IEEE.
11. Ahmed, S. T., & Basha, S. M. (2022). *Information and Communication Theory-Source Coding Techniques-Part II*. MileStone Research Publications.
12. Edan, N. M., Al-Sherbaz, A., & Turner, S. (2017, October). Design and evaluation of browser-to-browser video conferencing in WebRTC. In *2017 Global Information Infrastructure and Networking Symposium (GIIS)* (pp. 75-78). IEEE.
13. Suciu, G., Stefanescu, S., Beceanu, C., & Ceaparu, M. (2020, June). WebRTC role in real-time communication and video conferencing. In *2020 Global Internet of Things Summit (GIoTS)* (pp. 1-6). IEEE.
14. Al-Shammari, N. K., Alshammari, A. S., Albadarn, S. M., Ahmed, S. T., Basha, S. M., Alzamil, A. A., & Gabr, A. M. (2021). Development of soft actuators for stroke rehabilitation using deep learning. *International Journal of Advanced and Applied Sciences*, *8*(11), 22-29.
15. Abidi, S., Suleman, S., Dhanush, V., & Harshith, J. M. WEBRTC BASED VIDEO CONFERENCE APP.