Phuc Truong

Video Conference Room Implementation with WebRTC and React

Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology

Bachelor's Thesis

6 May 2021

Abstract

Author:	Phuc Truong
Title:	Video Conference Room Implementation with WebRTC and React
Number of Pages:	36 pages
Date:	6 May 2021
Degree:	Bachelor of Engineering
Degree Programme:	Information Technology
Professional Major:	Software Engineering
Instructors:	Janne Salonen, Head of School

The target of the project was to implement a video conference room application solely on a web platform. Taking advantage of the lightness of the WebRTC resource, the application can be embedded as an extension for more sizable web services.

The project uses JavaScript as the exclusive programming language and applies the WebRTC protocol for serving media communications on the web platform. WebRTC provides a standard peer-to-peer connection directly in browsers, without the installations of plug-ins or third-party extensions. The front-end of the application was developed with React components, and it consists of the fundamental features of a media communication controlling service, with the additional support of sending text messages.

The application manages to provide a user-friendly and significant real-time media communication service between a mobile and a personal computer, and to deploy to a cloud hosting server for the presentation. Being strongly attached to browsers, WebRTC and the whole application are guaranteed to be frequently updated.

Keywords: React, WebRTC, TCP, UDP, communication, application, media, call, video, audio, messaging

Contents

List of Abbreviations

WebRTC:	Web Real-Time Communication
TCP:	Transmission Control Protocol
UDP:	User Datagram Protocol
JS:	JavaScript
API:	Application Programming Interface
FS:	Financial Firms
CEO:	Chief Executive Officer
CFO:	Chief Financial Officer
SaaP:	Service as a Product
UI:	User Interface
JSX:	JavaScript XML
XML:	Extensible Markup Language
HTML:	Hypertext Markup Language
DOM:	Document Object Model
OSI:	Open Systems Interconnection
UUID:	Universally Unique Identifier

1 Introduction

1.1 Case Study: Remote Work

The COVID-19 pandemic has made an enormous impact on the world business processes, as well as the working culture. In the course of multiple immunization programmes around the globe, the pursuit of the “New Normal” habituation has become gradually intense. Many companies and organizations had endeavoured to break through the technological and geographical barriers even prior the Covid-19 pandemic. According to a PwC Middle East report, Saudi Arabia has spent around $15 billion on ICT infrastructure since the establishment of the National Digital Transformation Unit in 2017. This is “a critical first step towards realising the Digital Saudi 2030 vision of an innovative, digitally savvy national workforce”, and it results in more than 93% of Saudi Arabian citizens between the ages of 10 and 73 being internet users. [1.]

In general, remote work demands have tremendously increased since the outbreak of the disease. In fact, 29% of employers had at least 60% of their employees working from home at least once a week pre-pandemic, as reported by PwC on United States FS firms with a survey conducted from 1st to 12th June 2020. Additionally, the study shows that eventually 69% out of the firms expect partially three-fifth of their crew to telework on a weekly basis. Additionally, 61% of FS CFOs claim their plan is to make remote work permanent, as long as the position requirements are met. [2.]

1.2 Case Study: Video Conferencing Services

Telecommunication in business contains risks of security breaches and quality limitations. According to Reuters, the daily users of the video communication application Zoom, from a total of 10 million in December 2019, inflated to more than 200 million in March 2020. With more than 90,000 schools across 20 countries using Zoom services to conduct classes remotely, Zoom’s CEO – Eric

Yuan – had to officially apologize, and to acknowledge that the company failed to meet the expectations of customers and companies' privacy and security specifications, especially after multiple reports about "Zoombombing" incidents. [3.]

1.3 Project Overview

From the point of view of the author, it is necessary for companies to reduce their reliance on Service as a Product (SaaP) from external parties. Focusing on implementing features locally helps them become self-determining in controlling the workflow, maintaining the stability and the privacy of projects, as well as handling error situations. The objective of the final year project was to implement an independent video conferencing application, with fundamental features that help users organize group video calls and messages, for up to four people.

The application employs WebRTC, which is an open source framework for video conferencing, allowing direct integration into web browsers as its core library resource. It is hosted on the cloud deployment platform Heroku for demonstration purposes. [4.] The project was to provide a separate video conferencing application with the capability of self-integration into more comprehensive projects, dismissing the dependence upon third-party services, which assists enterprises to moderate the cost and the security threats when organizing a meeting.

2 Theoretical Background

As introduced in the previous section, the application was built with React for the web platform, and WebRTC was used as the core library resource. Before going to the implementation section, it is necessary to have a prior understanding about the technologies intended to be applied in the thesis project.

2.1 React

Developed by Jordan Walke and officially deployed on Facebook in 2011, React JS (or React.js or simply React) is an open-source library that is written in JavaScript, with the intention to build user interfaces exclusively for single-page applications. [5.] According to the annual developer survey conducted by StackOverflow, in 2020 React was the second most popular library for web development, with 35.9% respondents, just behind jQuery (43.3%). [6.] Despite the fact of not being the most used option in web development, React literally had a little shrinkage in difference in comparision to jQuery, when comparing to the survey of the previous year 2019: 31.3% respondents used React and 48.7% jQuery. [7.]

There are many features that makes React the most favourite web library among the companies' interests. One of the fundamental features of React is the ability to create reuseable UI *components*. The Components are considered to be the basic unit of the React application structure: "a message popup, a button, an icon, a passage of text, or a container to hold all of those inside" are all *components*. [8.] Basically they are functions with a set of application states and properties as the input, and an UI description as the output. When implementing the functions, developers need to provide the information on how all the components should look depending on the specific parameters states or properties. React then renders and makes appropriate changes to the view according to the state changes. The functions can be reused and put together to form larger units. [9.]

The programming method leads us another potential of React: declarative programming. As mentioned above, when working with React, developers need to demonstrate the abstract of the user interface, not to describe the methods and instructions to build the appearance. For instance, Listing 1 shows imperative code which manipulates the DOM by adding a book's name to an h1 tag, which is wrapped in a div element and append that to the body element:

```
function addBookNameToBody() {
    const bodyTag = document.querySelector("body");
    const divTag = document.createElement("div");
    let h1Tag = document.createElement("h1");
    h1Tag.innerText = "Life of Pi";
    divTag.append(h1Tag);
    bodyTag.append(divTag);
}
```

Listing 1. Example of an imperative implementation to add a book's name to an HTML tag.

This is extremely imperative since it first has to thoroughly find the appropriate tags and append the appropriate elements inside out, in step by step instructions. Meanwhile, with React to achieve the same goal developers only have to implement as Listing 2.

```
class Book extends Component {
    render() {
        return(
            <div>
                <h1>{this.props.name}</h1>
            </div>
        )
    }
}
```

Listing 2. Example of a React implementation to add a book's name to an HTML tag.

This examples of comparison show how React focuses exclusively on the result and clearly describes the user interface in the implementation snippets, instead of giving the instructions as the traditional method. React will process the translation of the declarative descriptions to the UIs in the browser. Additionally,

React shares this declarative ability with HTML UIs that represent dynamic data, not just static data as in the imperative example. [10.]

Along with the imlementation convenience, React also resolves another challenge that this declarative approach causes in return: performance. By introducing *virtual DOM* to keep a representation of the actual DOM elements in local browsers, and by applying them for the render process, React can compare the difference between the contents already displayed on the page and the changes, which have been made in the code, and then execute the necessary commands to visualize the updates, as shown in Figure 1. For that reason, React guarantees that the render process will be “done in a performant way”, but the declarative code will still be preserved. [11.]

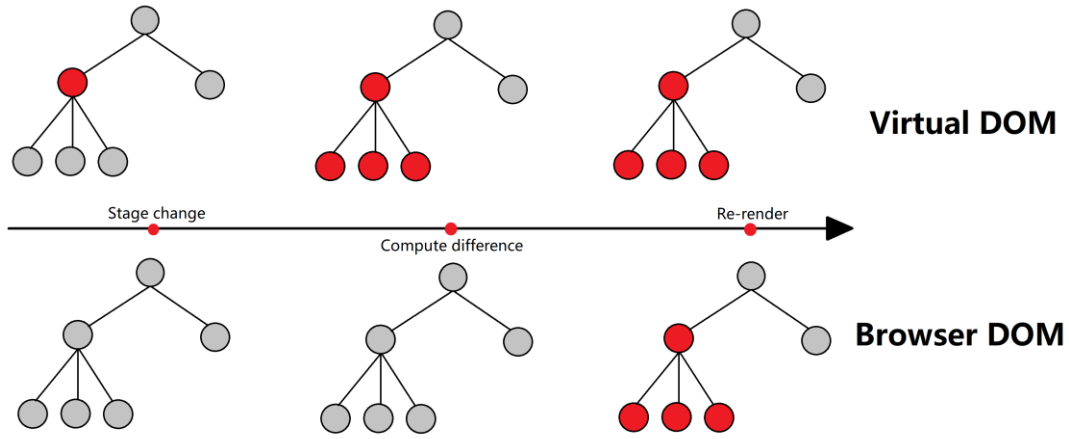

Figure 1. Performing calculations in the Virtual DOM limits rerendering in the Browser DOM. Adapted from Eisenman (2015). [12.]

React utilizes the performance of implementation process by making difference comparision and data updates in web applications without refreshing. Besides, it provides developers a simple and concrete syntax system, with JSX is the fundamental UI structure language. With JSX, programmers are able to describe the UI components as HTML markup and mixed with other custom tags for React components. [11.] This declarative JSX syntax approach tremendously reduces the time and effort needed to update components.

Furthermore, it eases the future observation of the projects, making it more understandable and easier to collaborate.

React has the potential to become even more popular in web service development and among the corporations that are limited in financing and human resources. On the other hand, with the current open state for contribution, this library empowers developers to expand and spread out their creativity in front-end implementation.

2.2 The Transport Layer

Computer networking became very popular in the 1970s, and the OSI conceptual model was introduced in 1984 to give a description about telecommunication system functionalities. Consisting of seven layers, the model gives a straightfoward portrait about how those systems work, and has been familiar to IT professionals until the present day. [13.] Network managers and developers can follow the OSI model concept to troubleshoot and navigate the problem sources whenever computer networking systems are down. On the other hand, tech merchants can use the model to explain and advertise the device’s features to customers.

In the OSI model, the transport layer is the middle layer, lying at the boundary between the physical hosts and the internetwork of routers, communication links that delivers information between hosts. [14.] This layer can be seen in Figure 2 below.

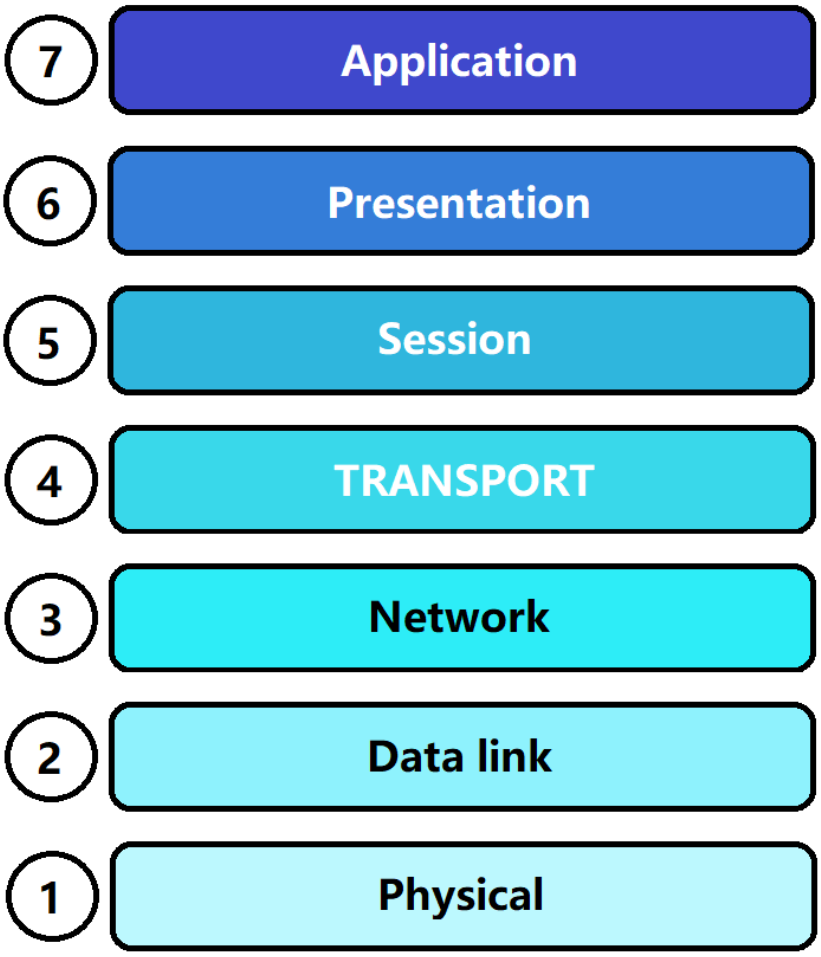

Figure 2. The Open Systems Interconnection model. Adapted from Day (1984). [15.]

The transport layer handles the data transfer process between networking systems and end hosts. According to Suherman, Suwendri and Marwan, there are two types of transport layer protocol: reliable and unreliable protocols. The protocol is considered to be reliable when it can guarantee the packet transmission to arrive at the receiver by the attempt of “using connection establishment, connection termination, acknowledgement, retransmission and flow control schemas”. [16.] In contrast, when the transmission arrival is not guaranteed, the protocol is claimed to be unreliable.

2.2.1 TCP

According to CISCO – a multinational network and cyber-security organization, TCP is a “connection-oriented transport protocol”. Before sending and receiving data between devices, it is mandatory to initially establish and preserve a connection to provide the data flow. TCP guarantees that the connection will stay being maintained until the message exchanging process between devices is completed. [17.]

To fulfil the duty, the data transmitted between locations are sliced into a series of packets and delivered individually. Each packet transmission is assigned with a sequence number for packets ordering, as an attempt to handle the cases of interrupted connection, or unwanted receiving of packets (for example, missing, out of order, or duplicate). Then the sender sends the packet with the synchronization flag (SYN) to the destination. At the receiving station, these packets are reassembled, following the appropriate order, to recreate the disseminated data stream. [18.] Each time receiving a packet, the receiving transmits the SYN signal, along with the acknowledgement (ACK) signal to the dispatching end, in order to request to receive the following packet in the sliced set. As soon as the established connection has succeeded, the sending returns the ACK signal to fulfil the request. This whole signalling process is well known as “Three-way Handshake”, which is visually described as in Figure 3.

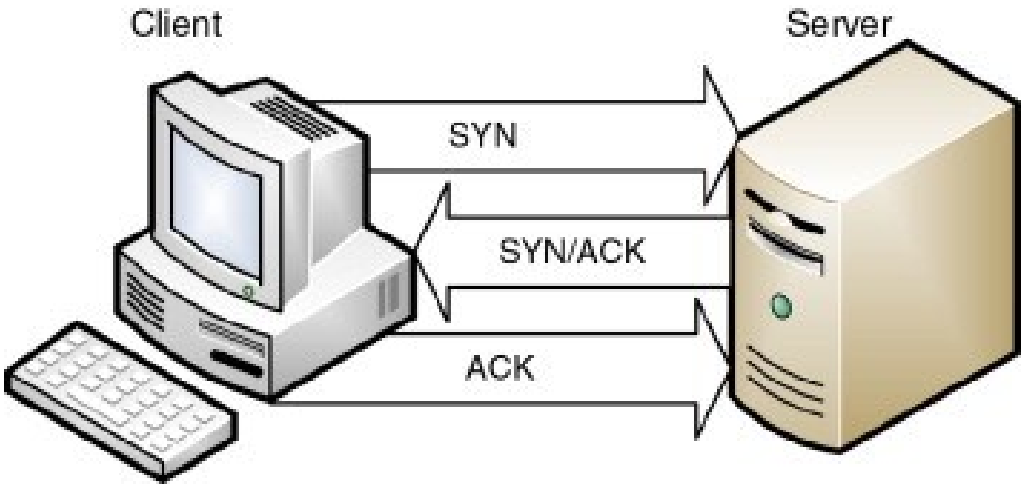

Figure 3. TCP Three-way Handshake Transmission Process. Reprinted from Conrad, Misenar and Feldman (2012). [19.]

TCP guarantees the data integrity when transferring through networks, no matter how much is the amount of the transferring data. This protocol is capable of organizing data in such a way that the secure transmission between the server and client is preserved. Consequently, TCP is used to convey data from upper protocols that require delivered data to arrive intact. [20.] The representative protocols which are suitable for TCP are listed as follows:

- Hypertext Transfer Protocol (HTTP)
- Secure Shell (SSH)
- File Transfer Protocol (FTP)
- Simple Mail Transfer Protocol (SMTP)
- Post Office Protocol (POP)
- Internet Message Access Protocol (IMAP)

2.2.2 UDP

In contrast to TCP, where the communication is required to maintain a stable connection during the process, UDP is acknowledged to be a “connectionless protocol”, which does not require providing a connection as a prerequisite to

deliver the packets. [21.] While TCP is targeted for stability, UDP is used for agility purposes. In order to accelerate the transmission, UDP initiates the process before the receiving devices send the agreement signals, and packets take multiple ways to respond between communicating devices, as shown in Figure 4. Consequently, there are possibilities that packets get lost during the delivery, or become out of order when arriving at the other end.

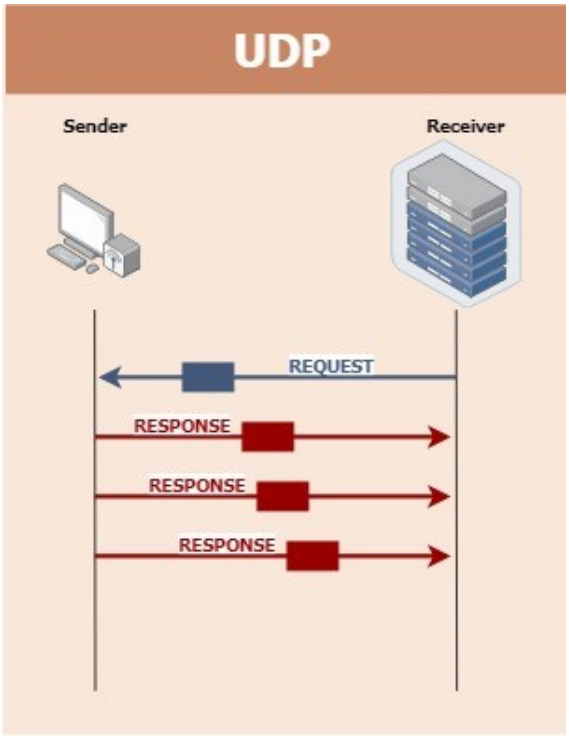

Figure 4. UDP transmission process. Reprinted from Purnendu (2021). [22.]

UDP is an unreliable protocol since it does not maintain the integrity of data transmission. Because of that, it is generally applied for low-latency and low loss-tolerant connections. The connections particularly support real-time applications that might allow some out-of-sequence or lost packets. [23.] Some of the mentioned communications, for instances, are:

- Transaction based protocols: Domain name system (DNS), Network time protocol (NTP)
- Voice over Internet Protocol (VoIP)

- Media playback/streaming

In conclusion, TCP is more secured and endorsed in data transfering, which causes a longer withholding time to process the transportation. For that reason, the protocol is more suitable in the applications that need to ensure the data successfully and thoroughly reach the destination, like email or web browsers. On the other hand, when it comes to responsiveness, there are more demands for receiving the data real-time; for example, receivers need to have the data as soon as possible. This is when UDP becomes more beneficial and potential to fulfil the duty. The applications that rely more on this protocol that can be mentioned here are primarily media streaming players or contacting applications, which is the core objective of the implementation in this thesis project.

The subsequent section provides further analysis and explanation of the advantages and the potential of WebRTC – the open-source real-time communication library, which is the core resource of the thesis project application.

2.3 WebRTC

2.3.1 Introduction

The Web Real-Time Communication protocol (in short WebRTC) is an open source project that aims to empower people to process real-time media communication. The project was released its original stable version to the audience in 2018, on the behalf of World Wide Web Consortium, by Justin Uberti and Peter Thatcher. The project provides the support in the web and native apps platform, in web browsers like Firefox or Chrome, which are popular nowadays, and in mobile native applications (iOS and Android) as well. [24.]

WebRTC provides a standard peer-to-peer connection between two browsers. This is the first protocol that literally initiates a peer-to-peer connection directly inside a browser, without any supports from extension plug-ins or third-party integrations. This advantage leads to the capability of extending the actions that are involved with peer connections to WebRTC. Nowadays there are multiple applications including peer-to-peer capabilities in their features: text messaging, file sharing and transfering, multiplayer gaming, and many more.

The question that needs to be resolved in this situation is, what makes those applications consider WebRTC suitable and trustworthy for their demands? What is the common capability that the open-source repository is able to provide concerning all the mentioned services? While other communications in web browsers, in particular, are processed over TCP, WebRTC operates data transportation over UDP. As being demonstrated in section 2.2.2, UDP provides its most potential in a low-latency and high-speed connection between both parties. By taking the advantages of low-level protocols to deliver high performance in data transmission, WebRTC is capable of stimulating data flow between networks, which enables large data amounts to be transported in a short period of time. [25.]

2.3.2 Architecture

The WebRTC API particularly supports transport modules to handle session and signalling management independently, which assists developers in setting up conference calls. The architecture also includes particular media engine frameworks: VoiceEngine and VideoEngine. Figure 4 visually demonstrates the internal WebRTC audio and video codecs, as segments of the internal WebRTC architecture.

As Figure 5 illustrates, the VoiceEngine framework delivers audio contents into the network from the sound card. The wideband codec iSAC and the narrowband codec iLBC control the audio streams. They were both old products that belonged to the Global IP Solutions, before becoming a part of WebRTC in 2011. VoiceEngine is capable of retaining the voice quality from microphones at a high level but the voice latency and bandwidth at a low level. In more detail, the module includes a “dynamic jitter buffer” and an “error concealment algorithm”, which camouflage the network’s negative effects, as well as packet loss situations. Additionally, VoiceEngine obscures the negative effects which are caused by data compression and encryption, voice activity detection, noise reduction or echo cancellation. [26.]

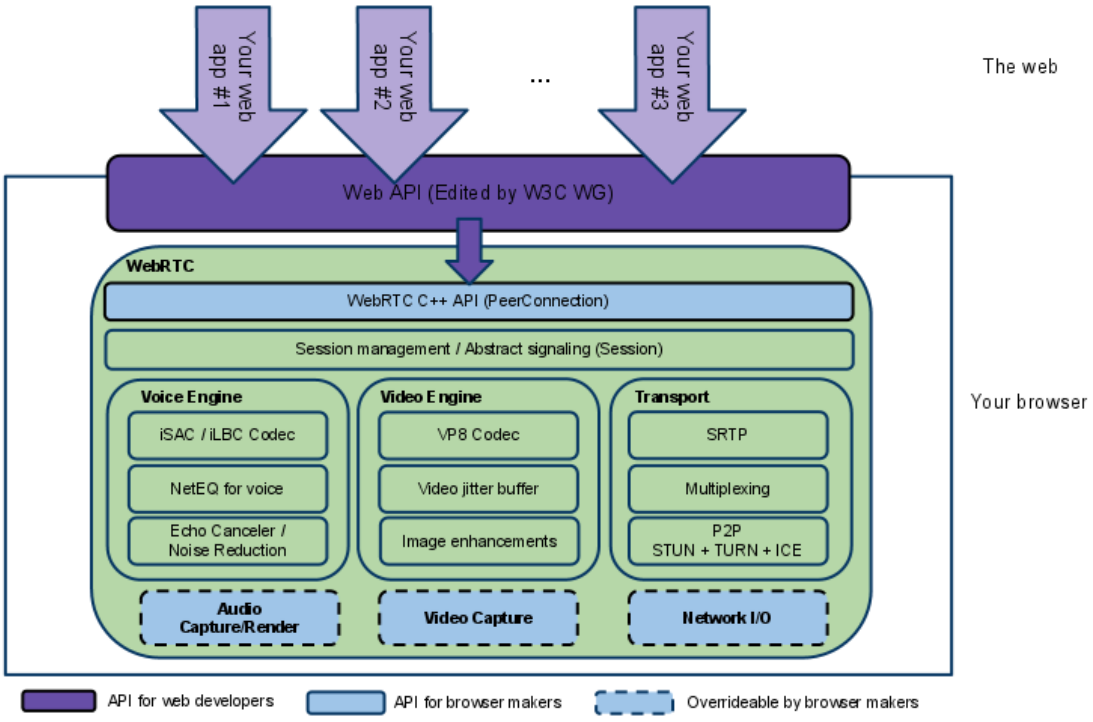

Figure 5. WebRTC architecture. Reprinted from Sequeira (2017). [27.]

On the other hand, the VideoEngine framework, with the inclusion of VP8 and H.264 codecs, manages moving direction pairs of video contents from recording devices to the network, and from the network to the user's device screen. The framework consists of features for image capturing, video processing, as well as video image quality improvement from the camera. Together with the mentioned capability, it also provides the "Dynamic Jitter Buffer" in order to enhance video quality and dissemble any packet loss, de-jitter, as well as bandwidth management. [28.]

2.3.3 Why WebRTC?

According to the architecture analysis in the previous section, it is understandable that although WebRTC provides peer-to-peer communication, it still needs servers to maintain the data exchanging for communication integration by the signalling process, as well as for firewalls and network address translators (NATs) management.

Before the creation of WebRTC, RTMP was a beneficial protocol, being based on TCP. It supported high-performance video, audio and data transportation between dedicated streaming servers throughout the networks. This protocol was created by Macromedia before becoming a property of Adobe. Still, despite the benefits it has been providing for such a long period, RTMP struggles in fulfilling the necessary duties. The demands in streaming responsiveness and agility are becoming higher over the years. Furthermore, with the retirement of Flash Player at the beginning of 2021, a stronger data transmission protocol is needed to resolve the challenges.

Among the most considerable preferences of WebRTC is the converting ability of millions of browsers into streaming terminals, without any external plug-ins required. Additionally, WebRTC reduces the delay struggles by providing sub-second latency. Besides, by using a unique bitrate technology, the protocol can manage to systematically modify video quality, which helps avoid any interruptions during the streaming process. [29.]

Meanwhile on the security perspective, in order to enhance the privacy level between communicating devices, WebRTC facilitates a secure connection between them. Transmitting traffic over a peer connection takes a direct route to the other destination besides being encrypted, which means packets sent between different connections on the cyberspace are able to take different routes. This feature enables users to be anonymous when using WebRTC applications, which is not guaranteed when networking with an application server.

With the depreciation of Flash media streaming applications, HTML5 has turned out to be an ideal replacement with multiple beneficial options, such as HTTP Live Streaming (HLS), and Dynamic Adaptive Streaming over HTTP (DASH), an adaptive streaming protocol that maintains video presentation during streaming, by enabling switching the video stream between bit rates on the basis of network performance. [30.] Being an HTML5-based solution, WebRTC has been an exclusive option for ultra-low latency media transmission. First of all, the

resource library is lightweight and does not require external plug-ins for media presentation. On the other hand, WebRTC is capable of taking advantage of mapping procedures to deal with data transfers between sessions. Furthermore, the open-source library provides the most efficient functions for live media transmission, as it is developed on the Web groundwork, specifically by JavaScript. [31.] For these reasons, WebRTC was identified as the core factor in the Video Conference Room application project.

3 Development Process

3.1 Prerequisites

3.1.1 PeerJS

There are multiple JavaScript library resources that support the WebRTC integration. For the thesis project, the author chose **PeerJS** as the most ideal resource option for the implementation. PeerJS is a light-weighted library that simplifies the peer-to-peer networking applying the WebRTC APIs. The library contains the client side script that connects to other WebRTC clients, and a server component implemented in Node.js to maintain the connection flows between devices.

PeerJS is able to wrap a friendly-to-use, delicate, yet still consistent API around the WebRTC implementation already existing in web browsers. Furthermore, PeerJS handles those complicated procedures of WebRTC API, such as handshaking or temporary binary string encoding. From that point onwards, WebRTC deals with other complicated processes of the current transmission, such as NAT traversal or the original established peer-to-peer connections.

Despite the peer-to-peer communication insurance from WebRTC itself, another separate server is still needed however, to perform as a handle signalling, and as a broker of the connections. [32.] PeerJS provides the PeerJS Server - an open-source implementation in Node.js of the mentioned connection broker

server. This is beneficial for the users who do not want to implement the server on their owned cloud hosting, due to limitations in financing, or other private matters.

3.1.2 UUID

Along with the integration of PeerJS, the project used **UUID** as the dependency to randomly generate the ID for the meeting room identifying, so that other users might take the ID information and mutually participate in the conversation.

3.1.3 Express

For the web application project setup, it is necessary to have a resource that utilizes the application architect sketching process in order to take care of multiple HTTP requests at a particular URL, which is the time for **Express** to come in. Express is a tractable essential framework for web applications, built with Node.js for the purpose of simplifying the websites implementation, web applications, and APIs. Express provides functions to identify the appropriate requests, followed by HTTP verbs such as `GET`, `POST`, or `SET`, as well as the URL pattern "Route". Additionally, the framework supports functions to define the view template engine being used, the template file locations, and the appropriate template for the response rendering. [33.]

3.1.4 Socket.IO

For the connection maintenance, the project applied Socket.IO in order to take advantage of those open connections to establish real-time and event-based transmission for the application. Socket.IO allows bilateral communication between the client and the server. In contemplation of organizing the connection and data transferring between end devices, Socket.IO uses Engine.IO, which is an undisclosed lower-level repository operated for the server implementation. At this stage of the project, Engine.IO-client was used for the client. In fact, the

Socket.IO project is divided into multiple repositories, relationships of which are visually described in Figure 6.

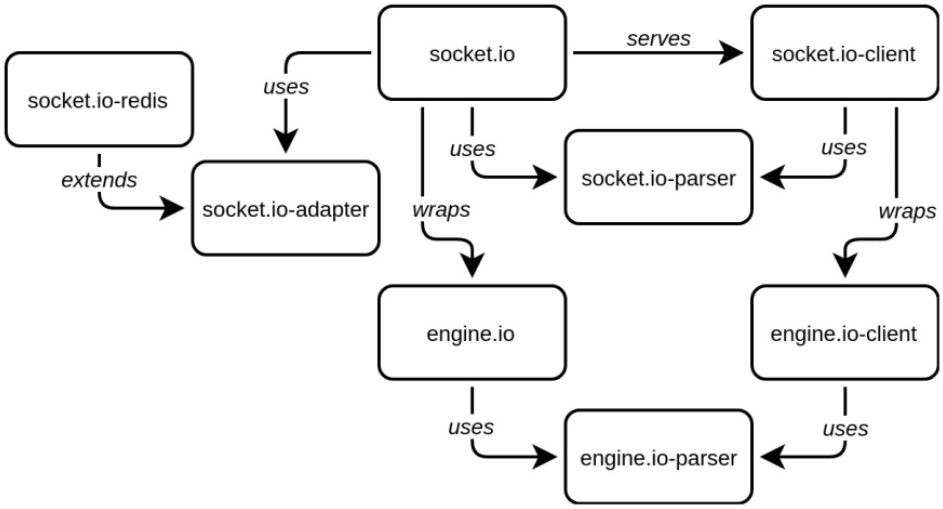

Figure 6. Relationship of Socket.IO dependencies. Reprinted from Socket.IO documentation page (2021). [31.]

Eventually, Listing 3 below illustrates what the dependency snippet looks like, with the list of those necessary library resources in the package.json file that are beneficial for the project:

```
"dependencies": {
    "peer": "^0.6.1",
    "peerjs": "^1.3.1",
    "express": "^4.17.1",
    "socket.io": "^3.0.3",
    "uuid": "^8.3.1"
  }
```

Listing 3. Project dependencies list.

3.1.5 Nodemon

To simplify the development of the project, the author also integrates Nodemon as the dependency specialized for local implementation and testing. Nodemon is a library resource that supports Node.js application development by automatically recompiling the app as soon as any updates in the code base are recognized.

```
"devDependencies": {
    "nodemon": "^2.0.7"
  }
```

Listing 4. Testing dependencies list.

One of the most potential features of Nodemon is the simplicity in executing the commands, without making any additional changes to the source code. Acting as a replacement wrapper for the word “node”, “nodemon” can replace “node” in the command prompt when processing the script execution.

3.2 Server-side Setup

After installing the necessary dependencies and generating the project folder from scratch, it is time to integrate those resources into the project. Technically what needs to be done beforehand for the application is to initialize a server module which listens to events on a particular port number. This depends on the selection of developers. For this project, the author chooses 3030 as the port number to set up for the video conference services. Whenever a request is made through the browser on port 3030, the server application shall handle the appropriate responses, which will be discussed in the subsequent section.

Listing 5 below shows what the file `server.js`, which is placed in the main repository, looks like when defining the compulsory elements for the application initialization.

```
const express = require("express");
const app = express();
const server = require("http").Server(app);
const io = require("socket.io")(server);
const { ExpressPeerServer } = require("peer");
const peerServer = ExpressPeerServer(server, {
  debug: true
});
```

Listing 5. Definition of compulsory elements in `server.js`.

It should be noted here that the unique ID of a conference room will be generated randomly. As introduced in the previous section, the project uses UUIDs for this process. The UUID is 128-bit in length data, representing 32 hexadecimal digits, as the following listing shows.

```
import { uuid } from 'uuidv4';
console.log(uuid());
// => ' cd16058a-f379-4f0d-9917-106840b1eadd'
```

Listing 6. An example of UUID.

Technically there are different types of UUIDs, each type has its own properties and is suitable for particular purposes. The first version is one of the most common UUIDs to be observed in some cases when working with computers. A Version 1 UUID is generated from the consolidation of the hosting computers' MAC address, and the date and time at the moment of the creation process, plus another random segment to preserve the uniqueness of the identifier.

By using Version 1, identifier numbers are guaranteed to get a unique ID; however, it still is possible to detect whether those numbers are generated from the same device or not and even whether this happens at the precisely same time. In spite of that fact, the collision possibilities are extremely small, nearly impossible, thanks to the randomness of the bits.

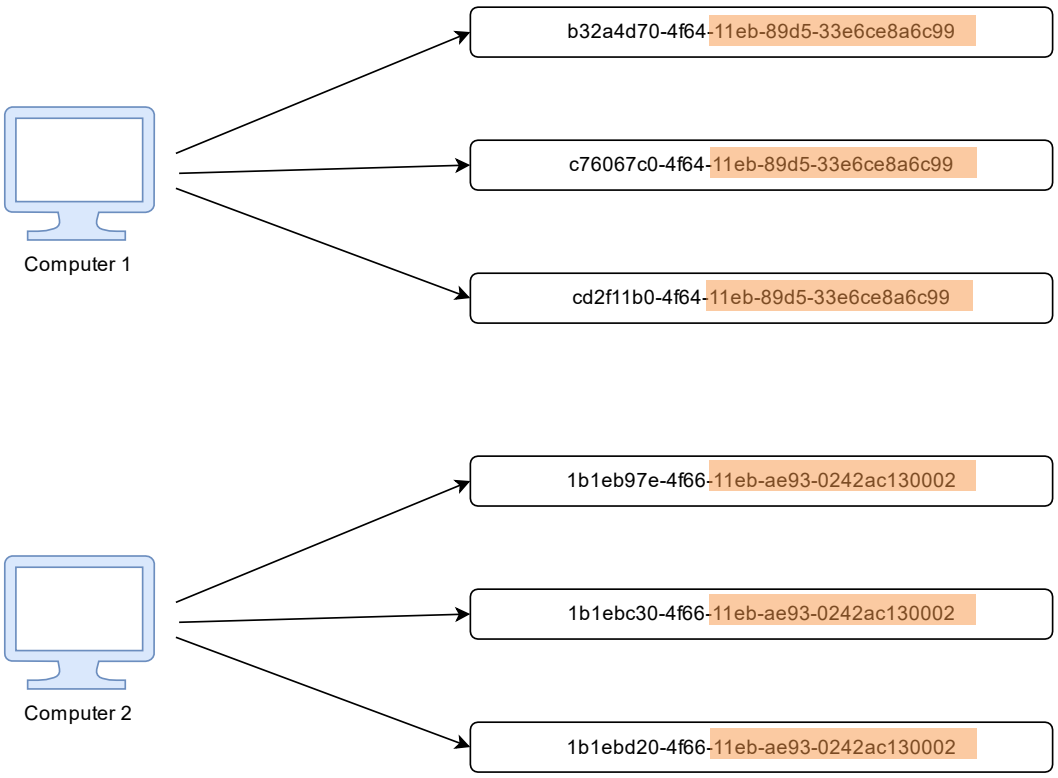

Figure 7. Creation examples of Version 1 UUID, the highlighted segments representing the device identifiers.

Based on what is illustrated in Figure 6, it is clear that the highlighted portions of the digits remain constant if they are created from the same computer, while the other part keeps increasing persistently throughout time, so the creation order could even be detected. This comes as one of the disadvantages of Version 1 UUID, when it risks the anonymity of users. Since it applies the MAC address of the host computers and the time into production, this will expose information to outsiders and it is possible for them to identify the time and the location just by examining the generated UUIDs. This is a problem that Version 4 UUID can resolve.

The bits that make up a Version 4 UUID are generated randomly, without any characteristic logic attached. Thanks to this randomness, it is pratically impossible to let the data information be compromised by examining the UUIDs such as Version 1. For that reason, the project chose Version 4 as the ideal type

for the conference room identifiers, since it is necessary to secure information and keep it exclusively for the participants.

```
const { v4: uuidV4 } = require("uuid");
```

Listing 7. The definition of the UUID variable.

The file `server.js` contained the implementation to set up the server, where is supposed to host the conversations. Within this file, an instance of ExpressPeerServer was created for the application, and was served at "`/peerjs`".

```
app.use("/peerjs", peerServer);
```

Listing 8. The PeerJS server calling.

We need to configure our application to use EJS, as well as arrange our routes for those pages we need for the application. For this project, there is particularly one page to be represented: the conference calling page, which is also the index page. This page is the conference "room" that is targeted to serve users. The page contains a bottom view with all the action buttons to handle the communication media settings, and a side view where users can use their keyboards to deliver the conversation. This route will use the generated UUID and render the official address of the conference room, as well as the whole page. When users enter the main URL in the browser, a new identifier number will be created and it will be automatically attached to the address, and a new conference room will be opened. Other users intending to join the conversation will use the created UUID to participate, or they just simply paste the whole address given by the conversation host.

```
app.set("view engine", "ejs");
app.use(express.static("public"));

app.get("/", (req, res) => {
  res.redirect(`/${uuidV4()}`);
});

app.get("/:room", (req, res) => {
  res.render("room", { roomId: req.params.room });
});
```

Listing 9. Application routes arrangement.

After initializing a new server with Express, a connection between the client and the server has to be organized as well, which is handled with the assist from Socket.IO. With Socket.IO, it is more advantaged to set up the connection to the server and retrieve the responses in the console. Each `socket.on()` line is an event which triggers as soon as a connection in the browsers, which is attached to the parameter `socket`, is fulfilled. A callback function is attached to the event, handling appropriate actions, based on what developers might give. For instance, in the scripts shown below, there are three events to handle:

- "`join-room`": when a new user participates in the current conversation.
- "`message`": when a new message is completely composed and sent to the texting conversation view of the same conversation.
- "`disconnect`": when a user departs from the current conversation.

Taking the "`message`" event for consideration, when a participant sends a message, Socket.IO will transmits the signal to deliver the message to the others in the current conference room. As shown also in the following listing, this action is handled by the `io.to(roomId).emit()` command, with roomId is the UUID of the room that the event is triggered, and "`createMessage`" is the label of this transmission.

```
io.on("connection", socket => {
  socket.on("join-room", (roomId, userId) => {
    socket.join(roomId);
    socket.to(roomId).broadcast.emit("user-
connected", userId);
    socket.on("message", message => {
      io.to(roomId).emit("createMessage", message);
    });

    socket.on("disconnect", () => {
      socket.to(roomId).broadcast.emit("user-
disconnected", userId);
    });
  });
});
```

Listing 10. Socket.IO handling connection event.

Eventually, the `server.listen` command arranges a port for listening to those requests. As mentioned above, 3030 is the port for all the actions that take place within the scope of this project.

```
server.listen(3030);
```

Listing 11. Server's listening command.

3.3 Media Retriving and Playback Functions

After sucessfully setting up the PeerJS Server and getting it in operation, it is time for the client side implementation, which is performed in the `PeerConnection` class. This file can be simply placed in the same directory with the `server.js` file created earlier, but it is recommended to put them in a separate folder for better organizing.

Taking advantages of the unique IDs created by UUID dependency, which was demonstrated in the previous section, PeerJS is able to identify and distinguish between the available peers.

```
const peer = new Peer([id], [options]);
```

Listing 12. The creation of a new `Peer` object.

As shown in the snippet above, when generating a new `Peer` object, there are two arguments that should be taken into consideration:

- `[id]`: This argument is the provided “address” that other peers use to connect to the host. In case the ID is null, the brokering server will automatically generates one, which is obliged to issue and conclude with an alphanumeric character, while symbols such as underscores (_) or dashes (-) are allowed in the middle of the series. Nonetheless, according to the official PeerJS documentation, setting the metadata option for identifying is recommended instead of using `[id]`, unless this is a production from brokering connections [33]. This argument is presented as `String` type. Nonetheless, this application has `[id]` already generated automatically when the peer server is initialized, so it can be skipped by leaving `[id]` as `undefined`.
- `[options]`: This argument contains a set of attributes containing the information regarding the `PeerServer` object, including the path, the host and the port number that the server is in operation (at this stage the port number is `3030`, as mentioned in the previous section).

Eventually, this is how the `Peer` object is provided in our `PeerConnection`:

```
const myPeer = new Peer(undefined, {
  path: "/peerjs",
  host: "/",
  port: "3030"
});
```

Listing 13. The `Peer` object’s creation for the project.

Besides setting up the servers and peers for the communication, it is compulsory to acknowledge the permission to access cameras and microphones on computers or smartphones. Thanks to the support of the WebRTC specification,

these devices are able to be called out and assigned as Media Devices, which can be easily accessed with through the navigator.mediaDevices object. This object is technically an implementation of the MediaDevices interface, from which the program can listen to device changes, as soon as a new peer participates (or departs).

```
myPeer.on("open", id => {
  socket.emit("join-room", ROOM_ID, id);
});
```

Listing 13. The `Peer` object handling the `"open"` signal.

The server, at this moment, should emit the “open” signal to the peer, so that the web socket shall transmit the room identifier for the participant to join the Media Stream, which is retrieved as shown in the following scripts.

```
let myVideoStream;
const myVideo = document.createElement("video");

navigator.mediaDevices
  .getUserMedia({
    video: true,
    audio: true
  })
  .then(stream => {

    //handle connecting events here

  });
```

Listing 14. The navigation of media devices.

With this implementation, the media devices (camera or microphone) are navigated by the `getUserMedia()` function, which is the most popular approach to retrieve the media data. This function takes those requirements regarding audio and video as the `MediaStreamConstraints` parameter object. In return, `getUserMedia()` returns a promise, which is going to be applied for handling connecting events, also using the `Promise` function.

Within the scope of the `Promise` function, it is imperative to assign the `MediaStream` object, which is taken as the variable `stream`, into `myVideoStream` (this variable is created as shown at the beginning of Listing 14). This `MediaStream` object is resolved for the matching devices. Then the `Video` element shall be appened to the stream as well through the `addVideoStream()` function. Meanwhile, the peer is set to automatically answer the upcoming call, as soon as the peer detects the `"call"` signal emitted from the other peer. Then the peer executes the `connectToNewUser()` function to connect the new participant to the current connection when he/she is granted the permission to take part in the conversation.

```
myVideoStream = stream;
addVideoStream(myVideo, stream);

socket.on("user-connected", userId => {
    connectToNewUser(userId, stream);
});

myPeer.on("call", call => {
    call.answer(stream);
});
```

Listing 15. Assigning `MediaStream` object and handling connection events.

Regarding the video stream appending function, after assigning the current stream to the source object of the video element, the video shall be injected to the video grid layout in the main page (the design of which will be discussed in the subsequent section) and be turned on. On mobile devices, another implementation is needed to handle the permission granting for the application, in order to access the camera and the microphone. On web application platforms, however, this method becomes unnecessary. For first time running, the browsers will trigger the option dialog for users to grant the permission for the access.

```
function addVideoStream(video, stream) {
  video.srcObject = stream;
  video.addEventListener("loadedmetadata", () => {
    video.play();
  });
  videoGrid.append(video);
}
```

Listing 16. The video stream adding function.

In order to connect the new user to the calling stream, the peer client needs to emits the calling signal, including the user ID and attach the video element to the room on the event `"stream"`. This event also executes the `addVideoStream()` function to inject the guest peer to the page, where the host video is currently presented.

```
function connectToNewUser(userId, stream) {
  const call = myPeer.call(userId, stream);
  const video = document.createElement("video");
  call.on("stream", userVideoStream => {
    addVideoStream(video, userVideoStream);
  });
  call.on("close", () => {
    video.remove();
  });
}
```

Listing 17. The new user connecting function.

Alongside the `"user-connected"` handling, when a user leaves the conversation, the server shall emit the `"user-disconnected"` event to the hosting peer to close the connection and eject the departing peer from the participants array.

```
socket.on("user-disconnected", userId => {
  if (peers[userId])
    peers[userId].close();
});
```

Listing 18. Socket.IO handling the `"user-disconnected"` event.

Now that the peer connection and the video call presentation are set up successfully, more features need to be included in the application to make it more reactive. There will be a button to trigger the `muteUnmute()` function, which will toggle the device microphone muted or unmuted each time the user taps it. This can simply be approached by retrieving the first track in the `AudioTrackList` (this track is referred to the device microphone's) of the video stream, and by setting the boolean `enabled` property to `true` to unmute the track. Otherwise it is set to `false`.

```
const muteUnmute = () => {
  const enabled = myVideoStream
                    .getAudioTracks()[0].enabled;
  if (enabled) {
    myVideoStream.getAudioTracks()[0].enabled = false;
    setUnmuteButton();
  } else {
    myVideoStream.getAudioTracks()[0].enabled = true;
    setMuteButton();
  }
};
```

Listing 19. The device's microphone toggle function.

Similarly, with the `playStop()` function to toggle the device camera on/off, the attempt applies to the video track. It is important to notice that each time after changing the video/audio track properties, the function also changes the layout of the action buttons as well.

```
const playStop = () => {
  let enabled = myVideoStream.getVideoTracks()[0].enabled;
  if (enabled) {
    myVideoStream.getVideoTracks()[0].enabled = false;
    setPlayVideo();
  } else {
    myVideoStream.getVideoTracks()[0].enabled = true;
    setStopVideo();
  }
};
```

Listing 20. The device's camera toggle function.

3.4 Layout Implementation

Now that the action scripts are fundamentally ready, it is time for the layout design. For the very first stage of the project, as set in the App routes, there is exclusively one main page responsible for presenting the video call, and the address will automatically get a unique room ID attached in the end. Figure 7 demonstrates the main components of the application.

The app consists of three main view components:

a. *Video Presentation view*: This expands for the largest area of the main screen, containing the video footages of the participants in a flex displaying grid of rounded rectangular shapes. The objective is to hold four people at maximum in the same conference room. Whether this limit on the number of people can be increased in the future depends on the latency maintenance of the connection. When a user turns off his/her camera, the rectangle that holds his/her video stays being active. However, instead of showing the camera image, the shape only shows an alternative potrait icon with the user name, or his/her profile picture with his/her name labeled below.
b. *Chat view*: This component has the second largest area of the page and is placed permanently on the right hand side. The view contains the text conversation between the participants within the conference room. Users may input the message in the dialog box at the bottom of the view and press the Enter key to send it. Once the message is sent, it cannot be revoked or unsent. For narrow views such as on mobile devices, this view can be hidden and toggled through the action button in component (c).
c. *Options view*: The view is located at the very bottom of the page and expands the smallest area. Consisting of the fundamental action buttons to control the media during the communication, the component includes the End button to close the call and leave the meeting room, the Microphone Mute/Unmute and the Camera toggle buttons. In addition, there is a Chat button to toggle the chat component (b).

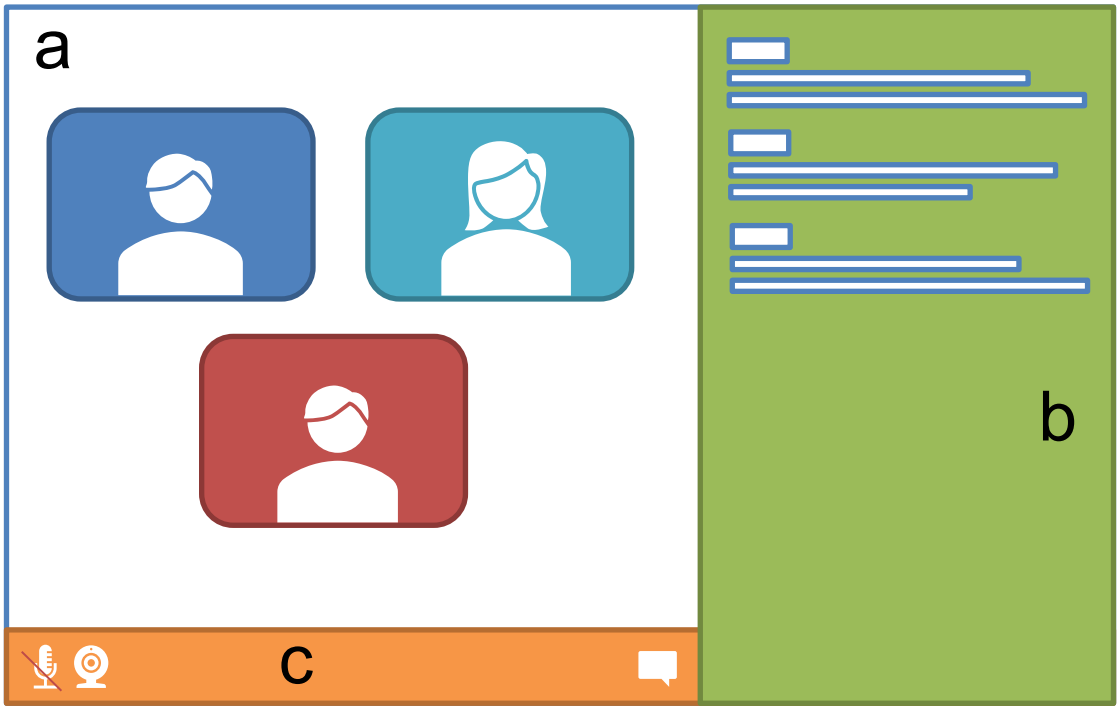

Figure 8. Application Layout Sketch.

An important factor is that since the application is using PeerJS for maintaining the video calling, a script tag of the PeerJS library needs to be injected into the App HTML document as well:

```
<script defer src="https://unpkg.com/peerjs@1.3.1/dist/peerjs.min.js" />
```

Listing 21. The injection of PeerJS library in the application's HTML document.

Regarding the toggle action buttons, by default the HTML document contains the active stylesheets and when the onclick property is triggered, the stylesheets will be toggled too.

```
<div class="main__controls__block">

    <div onclick="muteUnmute()"
        class="main__controls__button main__mute_button">
        <i class="fas fa-microphone"></i>
          <span>Mute</span>
    </div>

     <div onclick="playStop()"
        class="main__controls__button main__video_button" >
        <i class="fas fa-video"></i>
          <span>Stop Video</span>
    </div>

</div>
```

Listing 22. The implementation of toogle action buttons in the application's HTML document.

Now it is time to compile the implementation. In the package.json file, it is necessary to include the following command definition in the object of the script:

```
"scripts": {
    "devStart": "nodemon server.js"
}
```

Listing 23. The implementation of `nodemon` command.

With this definition, from this moment on, `nodemon` will be called when the project compilation is executed, instead of the `node` command. By using nodemon command, the project keeps refreshing each time detecting an update in the repository.

After executing this command in the Command Prompt, from the project directory, the server shall be initialized and put into operation.

```
yarn devStart
```

Listing 24. The implementation of `nodemon` command.

The following listing shows what the Command Prompt looks like when compiling the project:

```
$ nodemon server.js
[nodemon] 2.0.7
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node server.js`
```

Listing 25. The `nodemon` execution process display in Command Prompt.

Navigating to the address `localhost:3030` in the web browser, a new conference room with a random unique identifier shall be produced and the host camera may appear on screen, obviously after users have granted the permission to access the camera and the microphone on the browser.

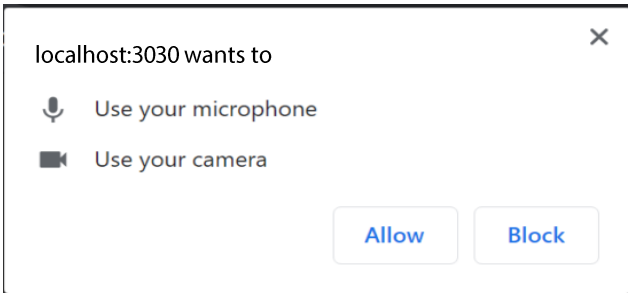

Figure 9. Google Chrome's permission dialog

Now the host can take the newly generated room ID and send it to other participants to join the conference. As soon as a new person successfully connected to the media stream, their video footage will appear immediately on all the participants' devices, as shown in Figure 10, with the unique ID is generated automatically when the `localhost:3030` address is entered.

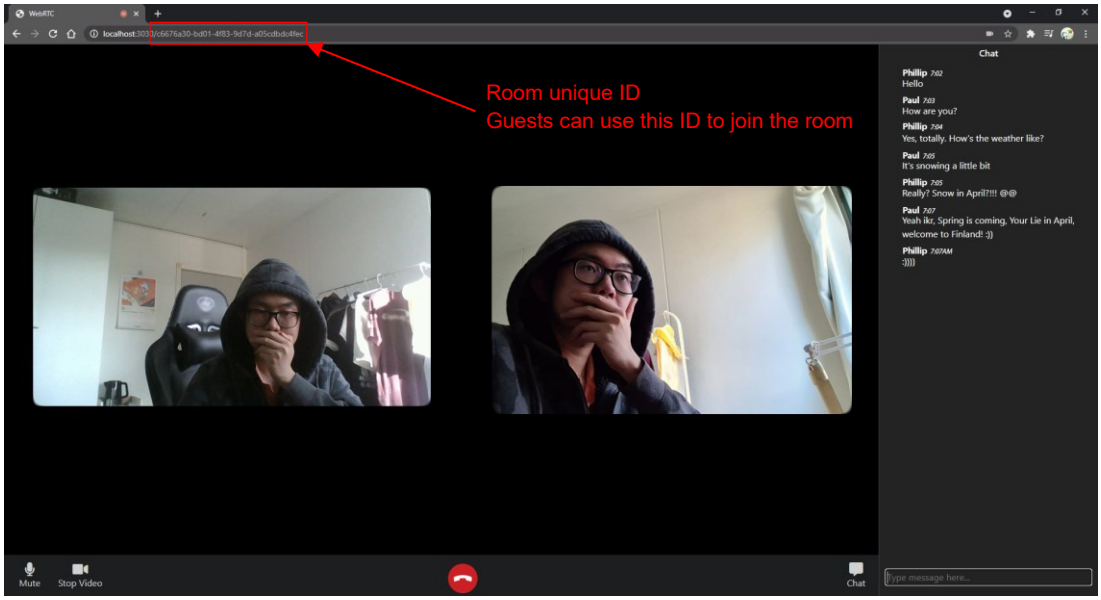

Figure 10. Application main page layout.

3.5 Evaluation

For the testing purposes, the author has deployed the repository to Heroku - a cloud platform that is popular for distributing, and supervising scalable apps. On the local side, he has attempted to handle the observation on Google Chrome 90.0.4430.93 (64-bit), with the built-in camera and microphone of his personal computer. The other partakers of this experiment included his father and one of his colleagues in Vietnam, and one of his partners currently working outside of the Information Technology field in Helsinki, Finland. His partner's testing browser was unclassified, while his father's browser was Microsoft Edge. His colleague in Vietnam used a Vietnamese browser, which was developed from a Chromnium version.

In order to assess of the outcome of the project, the author examined the following factors:

- *Latency:* The evaluation showed that it frequently took less than 0.5 seconds to arrange the connection when a person in the same country

as the host attempted to network with the media stream. However, if the geographical distance between the host and the tester was long, it took nearly double the delay time, for around one second to successfully connect all the participants to the same conference room.

- *Media quality:* Once more the geographical distance became an obstacle that made the quality of even the host's camera footage diminish. Only about 60-70% of the video sharpness remained. In addition, interruptions happened frequently. Additionally, the internet bandwidth is also another factor that affected this. Without taking these issues too seriously, the application has achieved maintaining and delivering communication at a tolerable level.
- *User-friendliness:* The application has accomplished to perform without giving too many instructions to the testers. Even the author's father had no difficulties in accessing and controling the conference room, thanks to him being already familiar with other conference organizing tools.

Despite having some positive overview about the features and the potential, the application contains numerous worrying aspects. The security of the web app is among the perspectives that the author still needs to investigate. Despite the simplicity and the lightness in the development process, it is possible that the application becomes more vulnerable for the outsiders to squeeze in and interrupt the stream, if more people are using the service. The stream can be compromised more easily and the private metadata could be exposed to the public.

Another possible issue is problems with the PeerJS server, which the author experienced during the implementation period. There was a moment when the PeerJS server became unavailable, which delayed the evaluation procedure and cost the author a significant amount of time. Considering future development, it is expected that the application will become more stable and that up to ten people at maximum at the same time will be able to join a conference session. In addition, the encryption of the private information of the users participating in the conference room is expected to improve.

4 Conclusion

The WebRTC protocol is a web API that was developed to provide a standard peer-to-peer connection between browsers. This open source framework succeeds in initiating the connection directly inside the browser, without the installation of any third-party plug-ins. In addition, WebRTC operates data transportation over UDP, which provides efficiently a low-latency and high-speed connection between connecting devices. WebRTC can also stimulate data flow between networks, and assist in the transportation of large data very fast. Because of the mentioned features, WebRTC is suitable and trustworthy for real-time media communication.

The thesis project was inspired by the large amount of demands for remote work during the COVID-19 pandemic situation. Companies and organizations have had to adapt to a new working culture for the last fourteen months. This has caused vulnerabilities to their business processes. To help companies cope in this situation, the goal of the final year project was to provide an active, user-friendly and significant application. The application takes advantage of WebRTC - a video conferencing library resource - and is directly integrated into web browsers. The application also contains a self-integration capability into larger and more expanding projects, instead of requiring the complicated methods and memory-consuming external plug-ins, which can leverage the business running, while the financial and time cost can be reduced as well.

One of the strengths of WebRTC is that it can put all the text and media conversations together, directly on a web browser. Instead of having a large number of communication software and applications that apply different connection protocols, now people can connect with each other on the same platform, using the same application. WebRTC is a significant solution as it dismisses dependence on external installations and enables effective real-time

media conversation. Now the web browser is the only prerequisite for this purpose.

For future developments, the author has the ambition to enhance the encryption level of the application, avoiding exposed information and interruption threats from the outside. Since being strongly attached to browsers, which are updated periodically, WebRTC can be guaranteed to stay up to date. The reliance of WebRTC on browsers makes it more trustworthy in terms of security. Based on personal experience, the author has the belief in the potential that WebRTC can offer in the field of real-time communication technology in the future.

References

1 Komati, Fadi; Saifan, Derar; Shahid Ali, Talha. Lessons from Lockdown: New Ways of Remote Working in Saudi Arabia. PwC Middle East; 2020. Available from: https://www.pwc.com/m1/en/publications/lessons-from-lockdown-new-ways-remote-working-in-sa.html [Accessed 24th March 2021]

2 Caglar, Deniz; Faccio, Ed; Jain, Ashish; Sethi, Bhushan; Mueller, Tim; Lamm, Julia. Financial Services Firms Look to a Future That Balances Remote and in-Office Work. PwC United States; 2020. Available from: https://www.pwc.com/us/en/industries/financial-services/library/balancing-remote-and-in-office-work.html [Accessed 24th March 2021]

3 Patnaik, Subrat. Zoom Pulls in More Than 200 Million Daily Video Users During Worldwide Lockdowns. Reuters; 2nd April 2020. Available from: https://www.reuters.com/article/us-health-coronavirus-zoom-idUSKBN21K1C7 [Accessed 24th March 2021]

4 Liu, Wei-Li; Zhang, Kai; Locatis, Craig; Ackerman, Michael. Cloud and Traditional Videoconferencing Technology for Telemedicine and Distance Learning. Telemed J E Health; 1st May 2015. Available from: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4432776/ [Accessed 24th March 2021]

5 Pandit, Nitin. What and Why React.js. C# Corner; 10th February 2021. Available from: https://www.c-sharpcorner.com/article/what-and-why-reactjs/ [Accessed 27th March 2021]

6 StackOverflow Developer Survey; 2020. Available from: https://insights.stackoverflow.com/survey/2020#most-popular-technologies [Accessed 27th March 2021]

7 StackOverflow Developer Survey; 2019. Available from: https://insights.stackoverflow.com/survey/2019/#most-popular-technologies [Accessed 27th March 2021]

8 Kozłowski, Maciek. What Is React? Is It a Framework, and Why Should You Care? Startup House Development; 28th February 2020. Available at: https://start-up.house/en/blog/articles/what-is-react [Accessed 29th March 2021]

9 Buna, Samer. All the Fundamental react.JS Concepts, Jammed into This Single Medium Article. Medium; 18th August 2017. Available at: https://medium.com/edge-coders/all-the-fundamental-react-js-concepts-jammed-into-this-single-medium-article-c83f9b53eac2 [Accessed 29th March 2021]

10 Kim, Myung. Declarative vs Imperative - Transition into React from JavaScript. Medium; 2nd April 2019. Available from: https://medium.com/@myung.kim287/declarative-vs-imperative-251ce99c6c44 [Accessed 29th March 2021]

11 Derks, Roy; Boduch, Adam. React and React Native - Third Edition. Packt Publishing; 2020

12 Eisenman, Bonnie. Learning React Native. O'Reilly Media, Inc.; 2015

13 Shaw, Keith. The Osi Model Explained and How to Easily Remember Its 7 Layers. Network World; 14th October 2020. Available from: https://www.networkworld.com/article/3239677/the-osi-model-explained-and-how-to-easily-remember-its-7-layers.html [Accessed 3rd April 2021]

14 Iren, Sami; Conrad T., Phillip. The Transport Layer: Tutorial and Survey. ACM Computing Surveys; April 2001. Available from: https://www.researchgate.net/publication/2401663_The_Transport_Layer_Tutorial_and_Survey [Accessed 2nd April 2021]

15 Day, John. The OSI Reference Model. IEEE Xplore; January 1984. Available from: https://www.researchgate.net/publication/2996873_The_OSI_reference_model [Accessed 25th May 2021]

16 Suherman, Suwendri, Marwan A. A Review on Transport Layer Protocol Performance for Delivering Video on an Adhoc Network. IOP Publishing Ltd; 2017. Available from: https://iopscience.iop.org/article/10.1088/1757-899X/237/1/012018 [Accessed 3rd April 2021]

17 TCP/IP Overview. CISCO; 10th August 2005. Available from: https://www.cisco.com/c/en/us/support/docs/ip/routing-information-protocol-rip/13769-5.html#tcp [Accessed 6th April 2021]

18 Thomas, George. Introduction to the Transmission Control Protocol. Automation.com; 3rd February 2003. Availble from: https://www.automation.com/en-us/articles/2003-1/introduction-to-the-transmission-control-protocol [Accessed 8th April 2021]

19 Conrad, Eric; Misenar, Seth; Feldman, Joshua. CISSP Study Guide. Syngress; 2012.

20 Lutkevich, Ben. Definition: TCP (Transmission Control Protocol). TechTarget [Last updated May 2020]. Available from: https://searchnetworking.techtarget.com/definition/TCP [Accessed 8th April 2021]

21 Rosencrance, Linda. Definition: UDP (User Datagram Protocol). TechTarget [Last updated April 2020]. Available from: https://searchnetworking.techtarget.com/definition/UDP-User-Datagram-Protocol [Accessed 8th April 2021]

22 Purnendu, Kar. Why Did Google & Facebook Stop Using The Traditional Network Connection? Medium; 3rd May 2021. Available from: https://purnendukar.medium.com/why-did-google-facebook-stop-using-the-traditional-network-connection-ae936926f1ee [Accessed 25th May 2021]

23 Garcia, Nuno; Gil, Fábio; Matos, Bárbara; Yahaya, Coulibaly; Pombo, Nuno; Goleva, Rossitza. Keyed User Datagram Protocol: Concepts and Operation of an Almost Reliable Connectionless Transport Protocol. IEEE Access; 7th February 2019. Available from: https://www.researchgate.net/publication/330946435_Keyed_User_Datagram_Protocol_Concepts_and_Operation_of_an_Almost_Reliable_Connectionless_Transport_Protocol [Accessed 8th April 2021]

24 Real Time Communication With WebRTC. Google [Last updated 18th February 2021]. Available from: https://codelabs.developers.google.com/codelabs/webrtc-web [Accessed 14th April 2021]

25 Ristic, Dan. Learning WebRTC. Packt Publishing; 2015

26 Azom, Edim; Dunka, Bakwa. A Peer-To-Peer Architecture For Real-Time Communication Using Webrtc. Journal of Multidisciplinary Engineering Science Studies (JMESS) Vol. 3 Issue 4; April 2017. Available from: https://www.researchgate.net/publication/332415582_A_Peer-To-Peer_Architecture_For_Real-Time_Communication_Using_Webrtc [Accessed 15th April 2021]

27 Sequeira, Princiya. WebRTC – Architecture & Protocols. WordPress; 19th August 2017. Available from: https://princiya777.wordpress.com/2017/08/19/webrtc-architecture-protocols/ [Accessed 15th April 2021]

28 DASH Adaptive Streaming for HTML 5 Video. MDN Web Docs. Available from: https://developer.mozilla.org/en-US/docs/Web/Media/DASH_Adaptive_Streaming_for_HTML_5_Video [Accessed 16th April 2021]

29 Govett, Devon. Peerjs: A Peer To Peer Networking Library In Javascript Using Webrtc. Internet; 14th February 2013. Available from: https://bada**js.com/post/43090030238/peerjs-a-peer-to-peer-networking-library-in [Accessed 22nd April 2021]

30 Express/Node Introduction. MDN Web Docs. Last updated 17th February 2021. Available from: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction [Accessed 25th April 2021]

31 Socket.IO Internals Overview. Available from: https://socket.io/docs/v2/internals/index.html [Accessed 25th April 2021]

32 Kamani, Soham. Which UUID Version Should You Use? UUID v1, v4 and v5 Explained (With Examples). Last updated 5th January 2021. Available from: https://www.sohamkamani.com/uuid-versions-explained/ [Accessed 27th April 2021]

33 PeerJS Documentation. Available from: https://peerjs.com/docs.html [Accessed 28th April 2021]