

**Vietnam General Confederation of Labor
TON DUC THANG UNIVERSITY
FACULTY OF INFORMATION TECHNOLOGY**



MIDTERM REPORT WEB APPLICATION DEVELOPMENT USING NODEJS

Web RTC

Instructor: **Mr. MAI VAN MANH**

Student: **Do Minh Quan – 521H0290**

Ho Huu An – 521H0489

Class : **21H50301**

Year : **25**

HO CHI MINH CITY, 2023

Vietnam General Confederation of Labor
TON DUC THANG UNIVERSITY
FACULTY OF INFORMATION TECHNOLOGY



MIDTERM REPORT

WEB APPLICATION DEVELOPMENT

USING NODEJS

Web RTC

Instructor: **Mr. MAI VAN MANH**

Student: **Do Minh Quan – 521H0290**

Ho Huu An – 521H0489

Class : **21H50301**

Year : **25**

HO CHI MINH CITY, 2023

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to Mr. Mai Van Manh from Ton Duc Thang University for his invaluable support and guidance throughout the development of this web application. His expertise in the field of WebRTC has been instrumental in shaping the direction of this project.

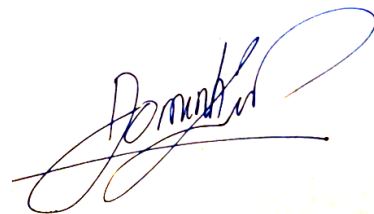
Mr. Mai Van Manh's commitment to excellence and his willingness to share his knowledge have significantly contributed to the success of this endeavor. His mentorship has not only enhanced my understanding of Node.js and web application development but has also inspired me to explore new horizons in the realm of real-time communication over the web.

I extend my heartfelt thanks to Mr. Mai Van Manh for his continuous encouragement and support, which have played a pivotal role in the completion of this project. His dedication to fostering a learning environment has been a source of inspiration, and I am grateful for the opportunity to work under his guidance.

Ho Chi Minh city, 8th December, 2023

Author

(Sign and write full name)



Do Minh Quan



Ho Huu An

THIS PROJECT WAS COMPLETED AT TON DUC THANG UNIVERSITY

I fully declare that this is my own project and is guided by Mr. Mai Van Manh; The research contents and results in this topic are honest and have not been published in any form before. The data in the tables for analysis, comments and evaluation are collected by the author himself from different sources, clearly stated in the reference section.

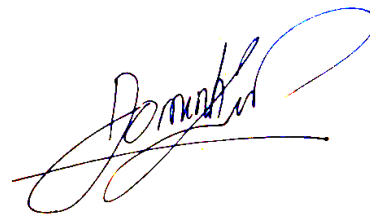
Besides that, the project also uses a number of comments, assessments as well as data from other authors, other agencies and organizations, with citations and source annotations.

Should any frauds were found, I will take full responsibility for the content of my report. Ton Duc Thang University is not related to copyright and copyright violations caused by me during the implementation process (if any).

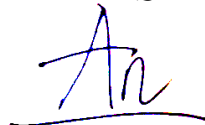
Ho Chi Minh city, 8th December, 2023

Author

(Sign and write full name)



Do Minh Quan



Ho Huu An

CONFIRMATION AND ASSESSMENT SECTION

Instructor confirmation section

Ho Chi Minh December, 2023
(Sign and write full name)

Evaluation section for grading instructor

Ho Chi Minh December, 2023
(Sign and write full name)

SUMMARY

Under the heading "Web RTC" - Analyzing Web RTC technology and creating a website that permits online video calls, the group has deliberated, investigated, and evaluated the prerequisites for creating an application that facilitates online chat and video conversations. This sample application needs to be simple to use, have a user-friendly design, and have chat and real-time video capabilities in order to enhance ease of usage. The team has provided a detailed explanation of the application's core functions, along with a list of key constraints, using use case diagrams, use case specifications, class diagrams, sequence diagrams, activity diagrams, Entity-Relationship Diagrams (ERD), and relational data models. This makes Web RTC technology's functionality more clear.

With this interesting and useful topic, the team hopes to put theoretical understanding to use in conjunction with research, system analysis, outside information, and firsthand experiences to create an application that maximizes efficiency by supporting people in their work and learning.

It is vital to keep in mind that while translating, specific terminology and context must be taken into account. Therefore, changes may be necessary depending on the exact technical words and standards used in your project or report.

INDEX

ACKNOWLEDGEMENT	i
CONFIRMATION AND ASSESSMENT SECTION	iii
SUMMARY	iv
LIST OF ABBREVIATIONS	3
LIST OF TABLES, DRAWINGS, GRAPHICS.....	4
CHAPTER 1 – INTRODUCTION TOPIC WEBRTC	6
1.1 Introduction.....	6
1.2 Significance and Real-World Applications of WebRTC.....	6
CHAPTER 2 – THE THEORETICAL BASIC OF THE STUDY	7
2.1 Web RTC	7
2.1.1 Core components of WebRTC and practical applications.....	7
2.1.2 How it works.....	8
2.1.3 Practical application.....	10
2.1.4 Advantages and disadvantages of WebRTC.....	10
2.2 NodeJS	12
2.3 Bootstrap	13
2.4 ExpressJS	14
2.5 RTCPeerConnection	14
2.6 SocketIO.....	17
2.6.1 Definition and features	17
2.6.2 How it works	19
2.7 STUN and TURN server.....	20
2.7.1 STUN server.....	22
2.7.1.1 Introduction	22
2.7.1.2 How it works	22
2.7.2 TURN server.....	23

2.7.2.1	Introduction	23
2.7.2.2	Internal Working of a TURN Server	25
2.8	Comparing RTCPeerConnection with other WebRTC libraries	28
CHAPTER 3 - IMPLEMENT WEBRTC		30
3.1	Request analysis	30
3.1.1	Require function	30
3.1.2	Non-functional requirements	30
3.1.2.1	Performance requirements	30
3.1.2.2	Software quality attributes	30
3.2	UML Diagrams	31
3.3	Result of application after deploying	33
CHAPTER 4 – SUMMARY		37
4.1.	User interface	37
4.2.	Limitations	41
4.3.	Future Directions	42
TASK ASSIGNMENT AND EVALUATION		44
Task Assignment		44
Member evaluation		45

LIST OF ABBREVIATIONS

P2P : Peer-to-Peer

STUN : Session Traversal Utilities for NAT

NAT : Network Address Translation

WebRTC : Web Real-Time Communications

AV1 : AOMedia Video 1

LIST OF TABLES, DRAWINGS, GRAPHICS

List of Figures

Figure 1: Logo of WebRTC	8
Figure 2: How WebRTC works	9
Figure 3: Logo of NodeJS	13
Figure 4: Logo of Bootstrap	14
Figure 5: Logo of ExpressJS	14
Figure 6: Get Static TURN Credentials	16
Figure 7: Logo of SocketIO	17
Figure 8: How socketIO works	19
Figure 9: WebRTC communications in real-world connectivity	21
Figure 10: STUN message exchange process	23
Figure 11: Internal Working of a TURN Server	26
Figure 12: One client with a single allocation and one peer	27
Figure 13: Two clients with respective allocations	28
Figure 14: General Use Case.....	31
Figure 15: Example repository of github	34
Figure 16: Create a New Service on Render.....	35
Figure 17: Connect GitHub Repository	35
Figure 18: Configure Build Settings	36
Figure 19: Room creation interface.....	37
Figure 21: Interface video call with chat after deploy	38
Figure 22: Interface share screen	39
Figure 23: Interface record function	40

List of Tables

Table 1: Comparing RTCPeerConnection with others	29
Table 2: Use Case Descriptions	33
Table 3: Task Assignment.....	44
Table 4: Member evaluation	45

CHAPTER 1 – INTRODUCTION TOPIC WEBRTC

1.1 Introduction

WebRTC (Web Real-Time Communication) is a free and open-source project that provides web browsers and mobile applications with real-time communication (RTC) via application programming interfaces (APIs). It allows audio and video communication to work inside web pages by allowing direct peer-to-peer communication, eliminating the need to install plugins or download native apps. WebRTC is a powerful tool that can be used to create a wide variety of real-time communication applications. It is easy to use and supported by all major browsers and mobile platforms.

1.2 Significance and Real-World Applications of WebRTC

- **Seamless Communication:** Enhances user experience through direct integration of real-time audio and video communication in web browsers.
- **Virtual Meetings:** Facilitates online meetings and conferences without the need for additional software installations, promoting convenience.
- **Interactive Websites:** Powers live interactions on websites, enabling features like live customer support and direct buyer-seller engagement.
- **Online Education:** Transforms online education by facilitating real-time interaction between teachers and students through video.
- **Gaming Experiences:** Enhances online gaming with interactive features for players, supporting both competitive and cooperative gameplay.
- **Live Streaming:** Enables direct live streaming on the web for events, performances, and online broadcasts, eliminating intermediaries.

CHAPTER 2 – THE THEORETICAL BASIC OF THE STUDY

2.1 Web RTC

2.1.1 Core components of WebRTC and practical applications

WebRTC (Web Real-Time Communication) is a web API developed by the W3C, enabling browsers to communicate with each other through features like video calls, voice calls, or peer-to-peer (P2P) connections without the need for additional plugins or external software. Its main components include:

1. `getUserMedia`:

- **Media Capture:** `getUserMedia` allows web applications to access and capture media streams from the user's device, including video from the camera and audio from the microphone.

Permissions: It operates within the user's browser security context, requiring user permission before accessing camera and microphone resources.

2. `RTCPeerConnection`:

- **Peer-to-Peer Communication:** `RTCPeerConnection` facilitates real-time communication between browsers. It establishes a direct connection between peers for efficient data exchange without the need for intermediary servers.
- **Signaling:** Before establishing a connection, a signaling mechanism is needed to exchange information about network addresses, session control messages, and media metadata. While WebRTC doesn't define a specific signaling protocol, it provides the necessary hooks for developers to implement their own.

ICE (Interactive Connectivity Establishment): WebRTC uses ICE to deal with various network conditions, such as NAT traversal, ensuring that peers can establish a direct connection even when both are behind firewalls or routers.

3. `RTCDataChannel`:

- P2P Data Transfer: RTCDataChannel allows browsers to establish a bidirectional communication channel for the peer-to-peer exchange of arbitrary data. This can be useful for applications that require real-time data sharing beyond audio and video.
- Use Cases: Examples of applications using RTCDataChannel include online gaming, file sharing, collaborative document editing, and any scenario where low-latency, real-time data transfer is crucial.

Common Use Cases:

- Video and Voice Calls: WebRTC is widely used for implementing video and voice calling features directly within web browsers, eliminating the need for users to install additional plugins or software.
- Live Streaming: It's employed for real-time streaming scenarios, enabling the broadcasting of live video and audio content over the web.
- Collaborative Tools: WebRTC powers collaborative tools that involve real-time interactions, such as virtual meetings, webinars, and online classrooms.
- Remote Assistance: Applications providing remote assistance or support often utilize WebRTC for live video communication.



Figure 1: Logo of WebRTC

2.1.2 How it works

JavaScript, APIs, and Hypertext Markup Language are used by WebRTC to integrate communication technologies into web browsers. Its goal is to simplify and ease

of use browser-to-browser data, video, and audio communication. The majority of popular web browsers support WebRTC.

WebRTC APIs carry out a number of essential tasks, such as enabling bidirectional data transmission over different data channels, initiating, monitoring, and terminating P2P connections between devices via browsers, and accessing and recording text, audio, and video-based data from devices.

Most of the time, P2P communications are used by WebRTC to establish connections between devices by sending data, audio, and video in real time. Session Traversal Utilities for NAT (STUN) servers and WebRTC can be utilized when users are on separate IP networks and there are firewalls blocking Network Address Translation (NAT) that impede RTC. In order to create peer connections, this allows a specified IP address to be transformed into a public internet address.

However, some networks are so restricted that IP address translation is not even possible with a STUN server. In these situations, a Traversal Using Relays over NAT (TURN) server is used in conjunction with WebRTC to facilitate user connection by relaying communication between users. To determine the optimal connection, using the Interactive Connectivity Establishment protocol.

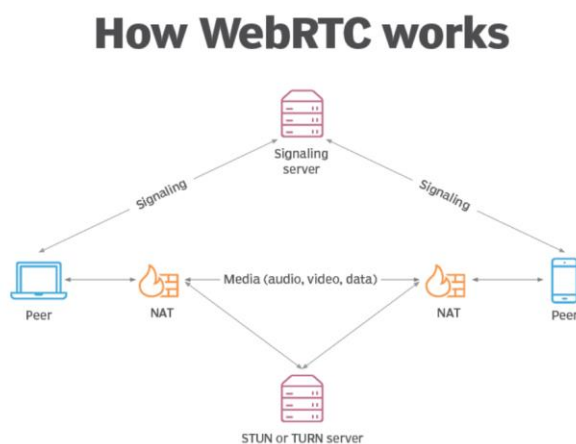


Figure 2: How WebRTC works

2.1.3 Practical application

Enabling real-time P2P conversations via the internet is the aim of WebRTC. WebRTC has a number of use cases, such as the following:

- Zoom, Microsoft Teams, Slack, Google Meet, and other video calling applications employ WebRTC for video chats and meetings.
- WebRTC is used by a number of industries, including internet of things, surveillance and monitoring, and healthcare. For instance, the usage of WebRTC in telehealth allows physicians to see patients virtually using a web browser.
- WebRTC serves as a bridge between browsers and security cameras in the realm of home and business security and surveillance.
- Real-time media significantly relies on WebRTC.
- The fundamental link for online learning between teachers and students is provided by WebRTC.

2.1.4 Advantages and disadvantages of WebRTC

WebRTC presents both opportunities and challenges for organizations in the realm of real-time communication. On the positive side, WebRTC offers several advantages:

- **Streamlined Integration:** WebRTC eliminates the need for extensive manual integration work by IT departments. It provides a standardized framework for real-time communication, reducing development effort and time.
- **Adaptive Communication:** WebRTC allows for dynamic adjustment of communication quality, bandwidth allocation, and traffic flow based on changing network conditions. This adaptability ensures optimal performance and user experience.
- **Broad Browser Support:** Most major web browsers, including Google Chrome, Mozilla Firefox, and Safari, support WebRTC. This wide browser compatibility enables broader adoption and accessibility for users across different platforms.

- Cross-Platform Compatibility: WebRTC works on any operating system that supports compatible browsers, providing flexibility and interoperability.
- Plugin-Free Experience: WebRTC does not require third-party components or plugins, simplifying the user experience and reducing dependency on external software.
- Open Source and Free: WebRTC is an open-source technology, meaning it is freely available and customizable, allowing organizations to leverage its features without incurring licensing costs.

However, there are also some challenges associated with WebRTC adoption:

- Bandwidth Considerations: Each user establishing a peer-to-peer browser connection in WebRTC can consume significant bandwidth, which may pose challenges in scenarios with limited network resources or large user populations.
- Server Requirements: WebRTC may require powerful servers to handle the processing and routing of media streams, which can lead to increased maintenance costs for organizations.
- Security and Privacy: While WebRTC provides mechanisms for secure communication, the implementation of security and privacy standards is the responsibility of organizations. Clear guidelines and standards are still evolving, requiring IT departments to ensure compliance with corporate security and privacy policies.
- Quality of Service: WebRTC lacks definitive quality of service (QoS) standards, which can result in inconsistent video or audio quality over the internet.

Organizations may need to implement their own mechanisms to monitor and manage QoS.

2.2 NodeJS

Node.js is an open-source, server-side JavaScript runtime environment that allows developers to run JavaScript code outside of a web browser. It is built on the V8 JavaScript runtime engine, which is the same engine that powers the Google Chrome browser. Key features and characteristics of Node.js include:

- Event-driven and asynchronous: Node.js is intended to be an asynchronous, non-blocking programming language. To effectively manage several connections at once, it employs a single-threaded, event-driven approach. It is hence ideally suited for creating high-performance, scalable network applications.
- JavaScript on the Server Side: Developers may build server-side JavaScript code by utilizing Node.js. Because of this, developers may use JavaScript, the same language for both client-side and server-side programming, to unify the development language.
- NPM (Node Package Manager): One of the biggest ecosystems of open-source libraries is included with Node.js and includes a package manager named npm. With npm, code developers can install, manage, and distribute code packages with ease.
- Cross-Platform: Node.js is a versatile framework for creating applications that function in a variety of environments because it is cross-platform and compatible with Windows, macOS, and Linux.
- Scalability: Node.js is renowned for its adept handling of numerous concurrent connections. Because of its event-driven architecture and asynchronous nature, it can be used to create scalable network applications, including real-time web applications.
- Community and Ecosystem: There is a thriving and active development community for Node.js. Rich libraries and frameworks are available for a variety of uses, such as web development, APIs, and other uses, within the Node.js ecosystem.



Figure 3: Logo of NodeJS

2.3 Bootstrap

Bootstrap is an open-source front-end framework that simplifies web development. It includes a responsive grid system, pre-designed CSS components, and JavaScript plugins. Developers can use Bootstrap to create mobile-friendly and visually appealing websites by integrating its files into their projects and utilizing its ready-made components. It's widely used for its ease of use, consistency, and time-saving features. Bootstrap is known for its:

- Grid System: A flexible 12-column grid system that allows developers to create responsive layouts easily.
- CSS Components: Pre-styled components for typography, forms, buttons, navigation bars, and more, providing a consistent and visually pleasing design.
- JavaScript Components: Ready-to-use JavaScript plugins like carousels, modals, tooltips, and popovers that enhance the functionality of web pages.
- Responsive Design: Ensures that websites look and function well on various devices, adapting to different screen sizes.
- Theming: Allows customization through a theming system, enabling developers to create unique designs by adjusting variables and styles.



Figure 4: Logo of Bootstrap

2.4 ExpressJS

Express.js is a popular web application framework for Node.js. It is a minimal and flexible framework that provides a set of robust features and utilities for building web applications and APIs.

Express.js simplifies the process of handling HTTP requests, routing, middleware management, and rendering dynamic content. It follows the middleware pattern, allowing developers to add modular functions (middleware) to process requests and responses. This makes it easier to implement various functionalities like authentication, error handling, and logging.

With Express.js, you can create server-side applications, build RESTful APIs, handle routing, serve static files, and more. It provides a straightforward and intuitive API for handling HTTP methods (GET, POST, PUT, DELETE, etc.), URL routing, request parameters, and response handling.



Figure 5: Logo of ExpressJS

2.5 RTCPeerConnection

RTCPeerConnection is a JavaScript interface in the Web Real-Time Communication (WebRTC) API that enables the establishment of a peer-to-peer

communication link between browsers. It is a crucial component for setting up real-time audio, video, and data channels between users without the need for a central server. WebRTC is commonly used for applications like video calls, voice calls, and other forms of real-time communication.

Peer-to-Peer Communication: `RTCPeerConnection` allows browsers to establish a direct communication channel between two peers. This is important for scenarios where real-time data needs to be exchanged directly between users, such as in video calls.

Signaling: Before `RTCPeerConnection` can be established, a signaling process is required. Signaling is the exchange of information between peers to coordinate the connection. This includes details like network addresses, session control messages, and media metadata. However, WebRTC does not define a specific signaling protocol, leaving it to developers to implement their preferred method for signaling.

ICE (Interactive Connectivity Establishment): ICE is a framework used by `RTCPeerConnection` to handle network address translation (NAT) traversal and establish a connection between peers. It helps overcome network challenges such as firewalls and routers, ensuring that peers can communicate even if they are behind different NAT devices.

SDP (Session Description Protocol): `RTCPeerConnection` uses SDP to describe the session's media details, such as the types of media (audio, video), codecs, and other connection-related parameters. SDP is exchanged between peers during the signaling process.

Media Streams: `RTCPeerConnection` allows the addition of media streams, such as video or audio, to be transmitted between peers. This involves capturing media from devices like cameras and microphones and sending it to the remote peer.

Example of setting up an `RTCPeerConnection` in JavaScript:

```

getIceServer = () => {
  return {
    iceServers: [{ urls: ["stun:hk-turn1.xirsys.com"] }],
    {
      username:
"reo1tPI46xcHp9TJ6x0rDL3GvhGfKEwlye3M_WvdnLAhfGyUpp0wf1AdQ8WqfYJ1AAAAAGVorJBEB
01pbmhrdWfu",
      credential: "92bfb816-8f96-11ee-9cd5-0242ac120004",
      urls: [
        "turn:hk-turn1.xirsys.com:80?transport=udp",
        "turn:hk-turn1.xirsys.com:3478?transport=udp",
      ]
    }
  ]
};
}
const peerConnection = new RTCPeerConnection(getIceServer());

```

This version retains the functionality of obtaining ICE servers from xirsys.com with Static TURN Credentials.

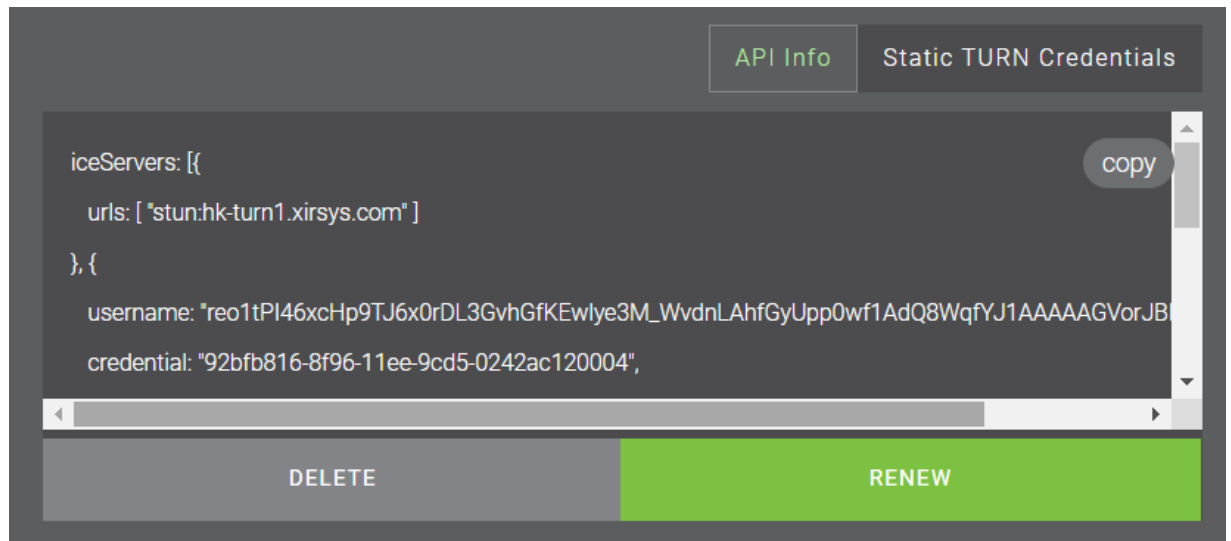


Figure 6: Get Static TURN Credentials

2.6 SocketIO

2.6.1 *Definition and features*

A JavaScript library called socket.IO allows web clients and servers to communicate in real time and both ways. Building real-time applications, such chat programs, online games, collaborative editing, and live updates, is a typical usage for it. The following are some important Socket.IO points:



Figure 7: Logo of SocketIO

- Event-driven, real-time communication is made possible with Socket.IO. It creates a steady connection that enables data transmission and reception at all times between the client and the server.
- WebSocket Protocol: The main means of real-time data transfer for Socket.IO is the WebSocket protocol. With WebSocket, a single, persistent connection may support a full-duplex communication channel.
- Fallback Mechanisms: WebSocket connections might not be supported in some contexts, although Socket.IO is made to function in such cases. It has backup features like extended polling, which enables communication in constrained or restricted contexts.

- **Event-Driven Model:** Socket.IO relies on events for communication. Events may be sent by the client and server, and they can also both listen for events coming from the other end. With this event-driven paradigm, managing various activities and real-time changes is made easier.
- **Support for Multiple Platforms:** Socket.IO is adaptable for developing real-time applications in a variety of settings since it includes client libraries for a number of platforms, including web browsers, Node.js, and other programming languages.

Example usage :

Server-side(nodeJS)

```
const io = require('socket.io')(httpServer);

io.on('connection', (socket) => {
  console.log('A user connected');

  // Listen for a custom event named 'chat message'
  socket.on('chat message', (msg) => {
    console.log('Message:', msg);

    // Broadcast the message to all connected clients
    io.emit('chat message', msg);
  });

  // Handle disconnection
  socket.on('disconnect', () => {
    console.log('User disconnected');
  });
});
```

Client-side (JavaScript in a web browser):

```
<script
src="https://cdnjs.cloudflare.com/ajax/libs/socket.io/4.0.1/socket.io.js"></sc
ript>
<script>
  const socket = io();
```



```
// Emit a custom event named 'chat message' with a message
socket.emit('chat message', 'Hello, Socket.IO!');

// Listen for the 'chat message' event
socket.on('chat message', (msg) => {
  console.log('Received message:', msg);
});
</script>
```

2.6.2 How it works

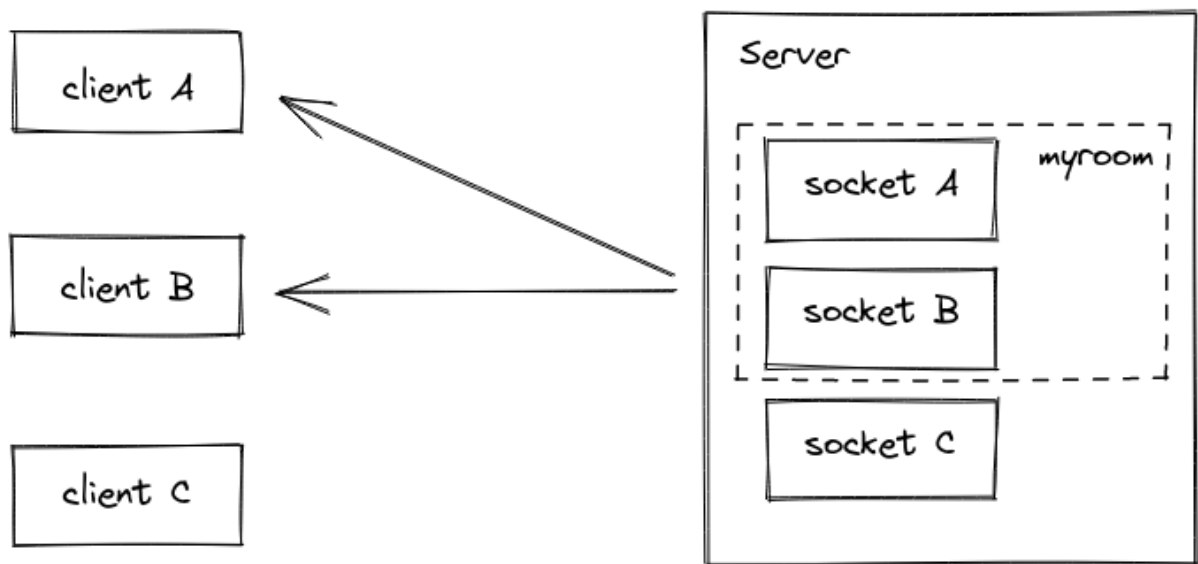


Figure 8: How socketIO works

Connection Establishment: A WebSocket handshake is started by the client submitting a unique HTTP request to the server in order to establish a real-time connection.

WebSocket Handshake: In response, the server upgrades the connection to a WebSocket connection after identifying the WebSocket handshake request. Through an HTTP upgrade response, this is accomplished.

Bi-Directional Communication: Without requiring further handshakes, messages can be sent at any moment between the client and the server after a WebSocket connection has been made.

Event-Driven Communication: Event-driven communication is used in socket.IO. Events with corresponding data can be emitted by the server and the client. Events with payloads of data, such "chat message" or "user connected," can be customized.

Room and Namespace Support: The idea of rooms and namespaces is supported by Socket.IO, enabling clients to join particular groups (rooms) and to communicate specifically with particular client subsets.

Fallback Mechanisms: Fallback techniques are built into Socket.IO to guarantee communication in situations when WebSocket connections may be prohibited or unsupported. Long polling is one of these ways; the server keeps requests open until fresh data is available..

Disconnect Handling: Client disconnect handling is supported natively by Socket.IO. When a client disconnects, the server and clients can both listen for "disconnect" events and take appropriate action, such as cleaning up.

2.7 STUN and TURN server

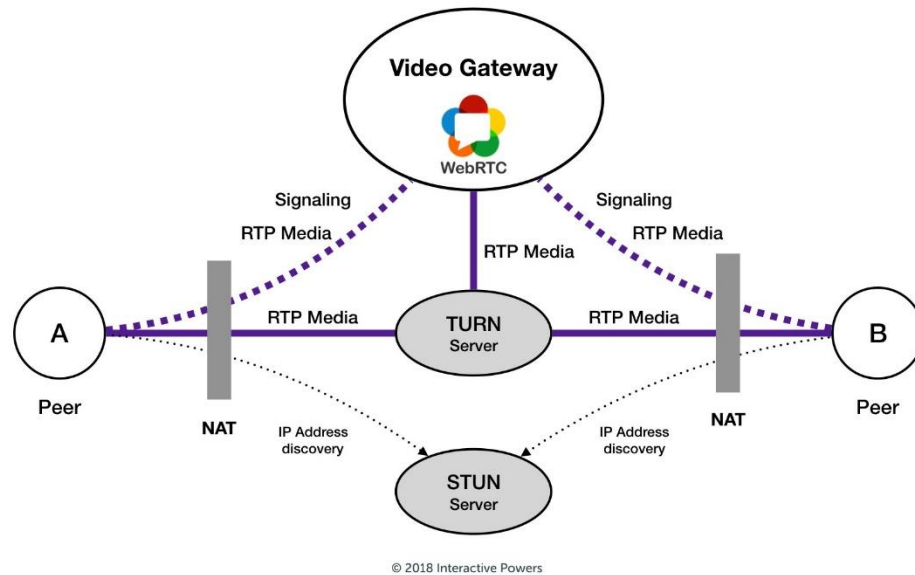


Figure 9: WebRTC communications in real-world connectivity

All clients have access to a complicated real-time video communications solution through our Video Gateway (WebRTC) platform, which transmits all voice, video, and data streams. A video gateway often has to be set up over the public Internet, requiring any user to connect and transfer media fragments via RTP (Real-time Transport Protocol) ports without experiencing any particular network problems.

However, sometimes this is not enough. Some users attempt to connect over IP networks that may have firewalls and network address translators (NATs) with special policies that forbid RTC interactions of any type. The optimal connection solution is identified using the ICE (Interactive Connectivity Establishment) protocol. It outlines a methodical approach to determining potential channels of communication between a peer and the WebRTC Video Gateway.

2.7.1 *STUN server*

2.7.1.1 Introduction

STUN (Session Traversal Utilities for NAT) is a protocol used in WebRTC to assist with establishing communication between peers located behind Network Address Translation (NAT) devices. NAT devices are commonly used in network setups to conserve IP addresses and provide a level of security.

When two devices want to establish a peer-to-peer connection for media transmission in WebRTC, they need to know their public IP addresses and the type of NAT they are behind. This is where STUN comes into play. A STUN server acts as an intermediary between the peers and helps discover their public IP addresses. The process typically works:

The WebRTC client sends a STUN request to a STUN server.

- The STUN server responds with the client's public IP address and port information.
- The client uses this information to determine its network configuration, including the type of NAT it is behind (e.g., full cone NAT, symmetric NAT, etc.).
- The client then shares its public IP address and port information with the peer(s) it wants to establish a connection with.
- The peers can use this information to establish a direct media connection, bypassing the NAT devices if possible.

2.7.1.2 How it works

A STUN server may identify a NAT device and find the IP address and port number that the NAT device has assigned to the STUN client by exchanging messages with a STUN client. STUN clients can communicate with one another after a data channel has been established. The two stages of the STUN message exchange procedure are hole punching and NAT detection. The procedure is shown in detail in the accompanying figure.

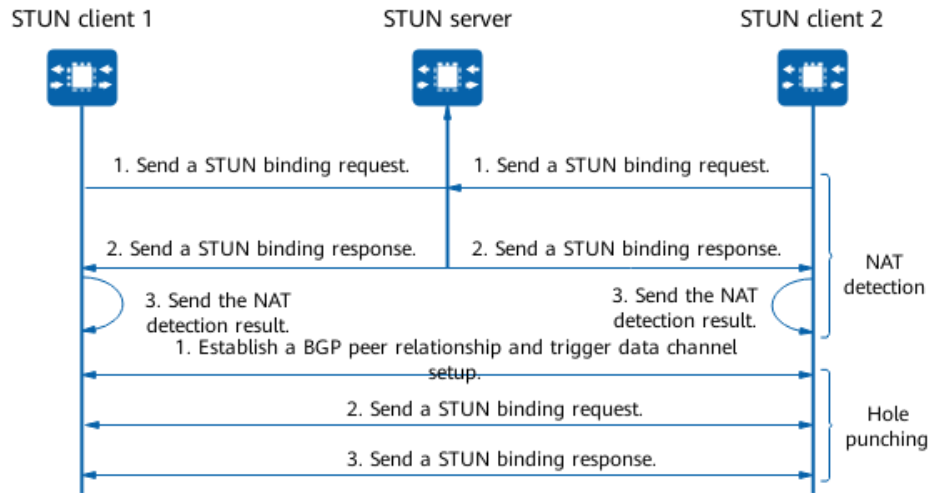


Figure 10: STUN message exchange process

NAT Detection

- A binding request is sent to the STUN server by every STUN client. The STUN server receives binding requests, gets the source IP addresses and port numbers, and then replies with a binding answer to every STUN client. The properties MAPPED-ADDRESS, XOR-MAPPED-ADDRESS, and RESPONSE-ORIGIN are carried by the binding response message.
- The binding response's MAPPED-ADDRESS or XOR-MAPPED-ADDRESS property provides the STUN client with an IP address and port number. It then compares those values to the source IP address and port number sent in the binding request. A NAT device is placed in front of the STUN client if they vary. Each STUN client tells the service module of the outcome after verifying the NAT detection result.

2.7.2 TURN server

2.7.2.1 Introduction

- WebRTC is the standard protocol that all devices employ for real-time media streaming over the Internet, utilizing shared network connections. Despite its widespread use, security measures such as firewalls or private IP addresses can pose challenges, preventing devices from forming the necessary connections to communicate effectively. In such scenarios, a TURN (Traversal Using Relays around NAT) server plays a vital role. Specifically designed for applications utilizing the WebRTC protocol, a TURN server serves as a solution for devices encountering obstacles like firewalls, private IP addresses, or issues related to Network Address Translation (NAT). Its primary function is to facilitate data transmission over the Internet, acting as a relay that allows devices to establish connections when direct connections are hindered
- A TURN server serves as a cloud-based intermediary between devices, facilitating communication when direct peer-to-peer connections are obstructed. It acts as a relay, relaying media traffic between devices that cannot establish a direct connection.
- This enables devices to engage in standard web conferencing, live streaming, or any other WebRTC-based communication.
- When devices attempt to establish a WebRTC connection, they typically start by using a STUN (Session Traversal Utilities for NAT) server to gather network information such as IP addresses and NAT type. However, if direct connectivity is not feasible due to restrictive firewalls or symmetric NATs, the devices will employ a TURN server instead.
- The term "TURN" stands for "Traversal Using Relays around NAT." The TURN server allows devices to transmit data through a relay mechanism, bypassing the connectivity obstacles posed by firewalls or NAT devices. By relaying media traffic through the server, devices can establish a secure connection and exchange real-time data using the WebRTC protocol.

- It's important to note that using a TURN server introduces additional latency and bandwidth usage compared to direct peer-to-peer connections. Therefore, TURN servers are typically used as a fallback option when direct connectivity is not possible.

Hole Punching

A method for creating a direct connection between P2P communication partners hidden behind NAT devices is called "hole punching." In particular, it uses NAT session entries on NAT devices to establish a data connection between P2P STUN clients over intermediary devices (such as RRs). The following is how STUN hole punching is done:

1. Through BGP, STUN clients are able to get TNP information from an RR.
2. STUN client 2 is informed about hole punching by STUN client 1.
3. STUN clients trade requests for hole punching in binding.
4. When binding requests are successful, NAT devices provide session entries.
5. Binding requests are still coming in from STUN clients until both NAT devices have created session entries.
6. A data channel that permits packet traversal across NAT devices is formed between STUN clients following the exchange of binding requests.

2.7.2.2 Internal Working of a TURN Server

Two individuals are connected to a TURN session:

- A host connected to the TURN server for communication is called a client (TURN client). It communicates with the server by sending TURN messages in accordance with the standard.
- Peers: The hosts with whom the client wants to speak. The traffic between the client and peers is redirected by the TURN server. All peers communicate with the server in the same way that they typically do with one another.

The Transport Addresses connected to Peers and TURN clients:

- Host Transport Address: A client's or peer's real transport address. It is made up of the local network's private IP address.
- Server-Reflexive Transport Address: A peer's or client's transport address that is accessible from outside of its network address translator (NAT). Outside of the local network, it is made up of the public IP address. The NAT assigns this address to match a certain host transport address.

The Transport Addresses associated with a TURN server:

- Server Transport Address: The TURN server's transport address, which is utilized to transmit TURN messages to the server. In order to interact with the server, the client utilizes this transport address.
- Relayed Transport Address: The TURN server's transport address used to relay packets between a peer and a client. The packet is sent to the client by the TURN server once a peer delivers data to this address.

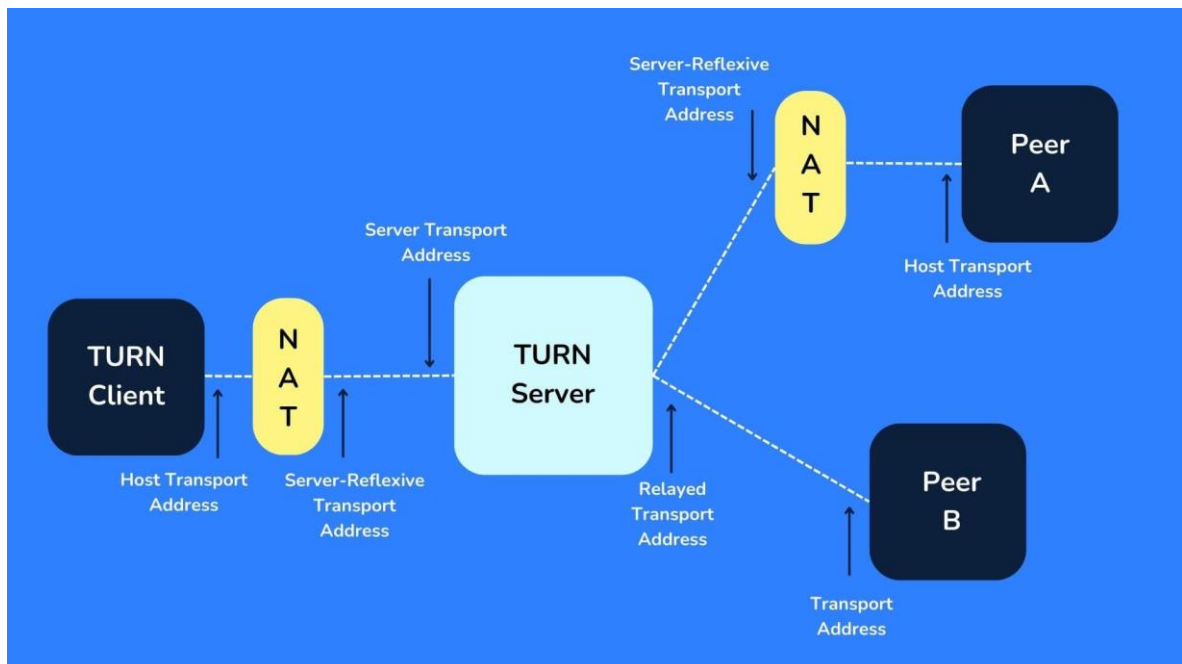


Figure 11: Internal Working of a TURN Server

2.1.1.3 TURN Usage

There are mainly 2 ways a TURN server can be used to achieve connectivity. We will look into them by considering one-on-one connections as an example.

- One client with a single Allocation and one peer.
- Two clients with respective Allocations.

One client with a single allocation and one peer

There is just one peer and one TURN client in this instance. This is often the suggested approach when there are problems with the direct connection solely for one side.

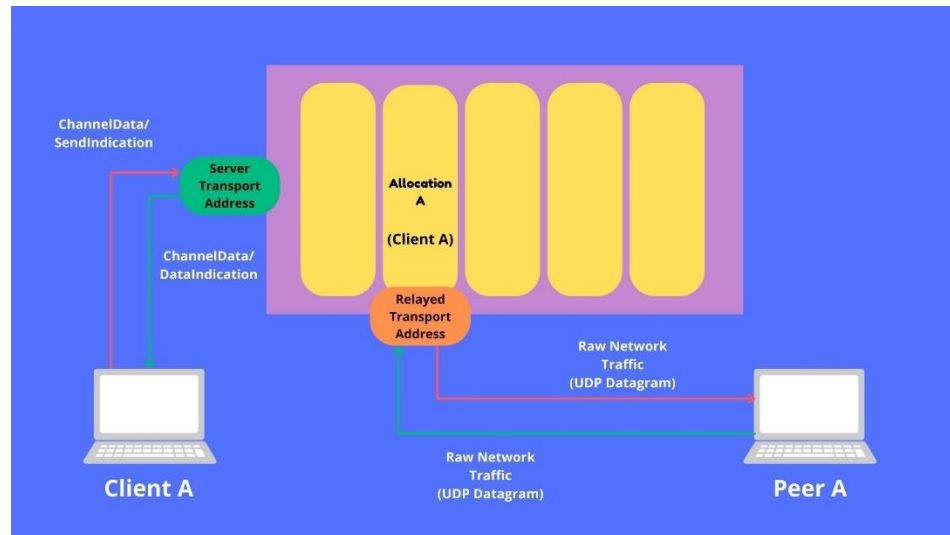


Figure 12: One client with a single allocation and one peer

Two clients with respective allocations

In this instance, the two people corresponding are TURN clients. When neither party is in favor of a direct connection, this is the way to go, and it also blocks UDP traffic.

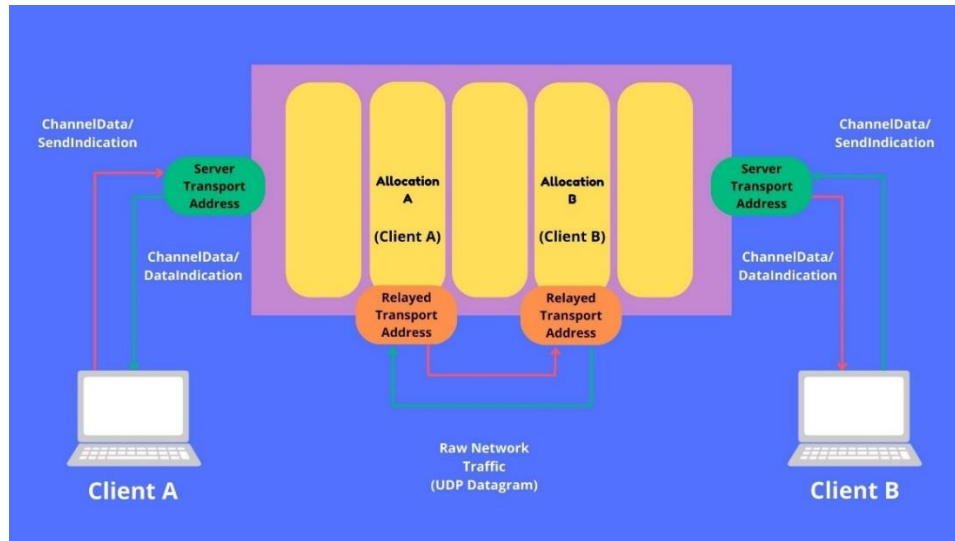


Figure 13: Two clients with respective allocations

Let's attempt to configure and test our own TURN server now that we've seen much too much about it conceptually. CoTURN, a very well-liked open-source TURN server solution, will be what we use.

2.8 Comparing RTCPeerConnection with other WebRTC libraries

RTCPeerConnection is a fundamental building block of WebRTC, providing the core functionality for establishing and managing peer-to-peer connections. However, it's not a complete solution for building real-time communication applications. Here's a comparison with other popular WebRTC libraries:

Criteria	RTCPeerConnection	Pion	PeerJS	EasyRTC	Socket.io with WebRTC adapter
Functionality	Core WebRTC API	High-performance framework	Simple and lightweight library	Feature-rich platform	Combines Socket.io with WebRTC

Ease of use	Moderate	Easy	Easy	Easy	Moderate
Features	Basic media streaming	Extensive features (media streaming, recording, data channels)	Basic media streaming	Extensive features (media streaming, recording, data channels, chat)	Basic media streaming
Performance	Good	Excellent	Good	Moderate	Moderate
Scalability	Good	Excellent	Moderate	Good	Moderate
Documentation	Good	Excellent	Good	Good	Moderate
Cost	Free	Free	Free	Freemium	Free
Community	Large	Growing	Active	Large	Moderate
STUN/TURN Server Integration	No	Yes	No	Yes	Yes
ICE (Interactive Connectivity Establishment)	Yes	Yes	Yes	Yes	Yes
Mobile Support	Yes	Yes	Yes	Yes	Yes

Table 1: Comparing RTCPeerConnection with others

CHAPTER 3 - IMPLEMENT WEBRTC

3.1 Request analysis

3.1.1 Require function

Individuals have the option to engage in both text-based chatting and video calls via a website, accessible from either a computer or a mobile device. The meeting spaces provided are separate from one another, ensuring privacy and independence. Upon establishing a room, participants have the ability to invite others by sharing a unique link. Additionally, individuals can join a meeting by entering a specific room code provided by the host. Users can manage various functionalities such as sound control, screen sharing, video options, chatting, and exiting the room. These features offer a comprehensive control over the meeting environment, allowing for a seamless and efficient collaborative experience.

3.1.2 Non-functional requirements

3.1.2.1 Performance requirements

Data is updated consistently and in real time; Information is efficiently and rapidly absorbed; a user-friendly interface with quick reaction times; ability to sustain consistent and high access times.

3.1.2.2 Software quality attributes

Create and utilize websites with Web RTC technology from anywhere at any time; After deployment, the program is simple to build, maintain, and update. robustness, dependability, portability, and interoperability; total security, safety, and control over data.

3.2 UML Diagrams

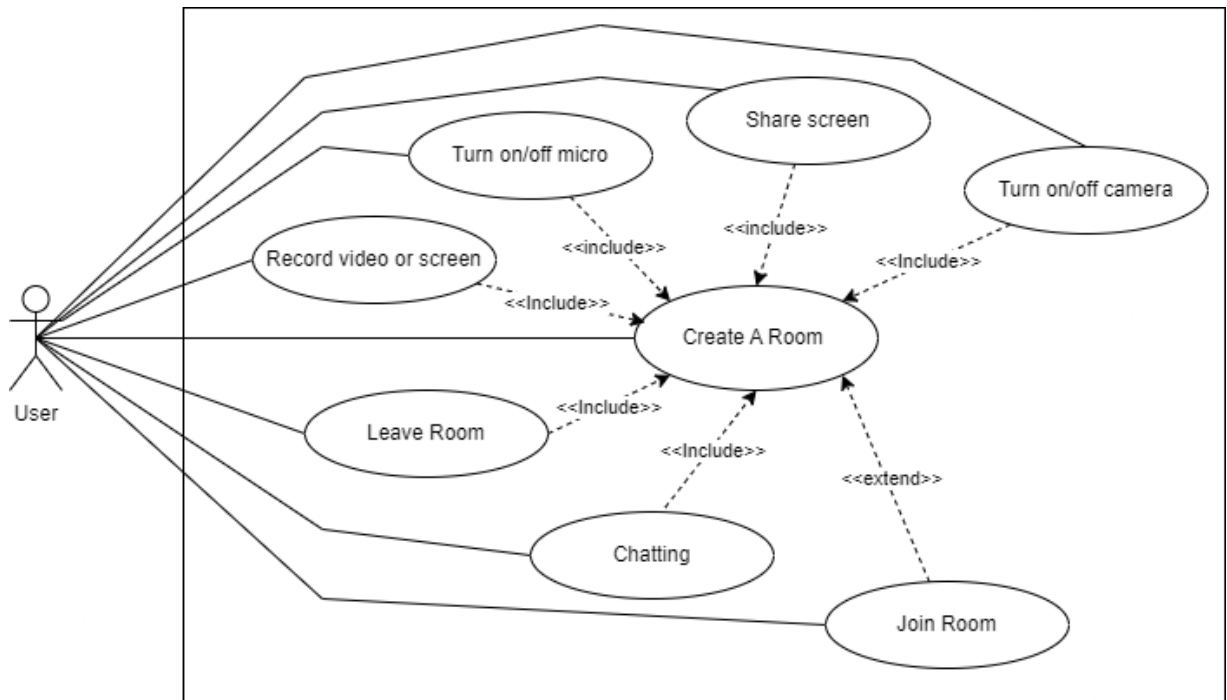


Figure 14: General Use Case

The use case diagram of a video recording system shows the different ways that users can interact with the system to record and share videos. The primary actor in the diagram is the User, who can perform the following use cases:

- Create a room: The user creates a room where they can record a video or screen share with other users.
- Join a room: The user joins a room that has been created by another user.
- Record video or screen: The user starts recording their video or screen.
- Share screen: The user shares their screen with other users in the room.
- Turn on/off camera: The user turns their camera on or off.
- Turn on/off micro: The user turns their microphone on or off.
- Leave room: The user leaves the room.

The diagram also shows a number of include relationships, which indicate that the use cases they are associated with are included in the other use case. For example, the Record video or screen use case includes the Turn on/off camera and Turn on/off micro use cases. This means that the user must first turn on their camera and microphone before they can start recording.

Use Case	Description	Actor	Precondition	Postcondition
Create A Room	The user creates a new room.	User	None	The room is created and the user is added to the room.
Join Room	The user joins an existing room.	User	The room exists.	The user is added to the room.
Leave Room	The user leaves a room.	User	The user is in the room.	The user is removed from the room.
Turn on/off micro	The user turns their microphone on or off.	User	The user is in a room.	The user's microphone is turned on or off.
Record video or screen	The user starts or stops recording the video or screen of the room.	User	The user is in a room.	The video or screen recording is started or stopped.

Share screen	The user starts or stops sharing their screen with the other users in the room.	User	The user is in a room.	The user's screen is shared with the other users in the room.
Chatting	The user sends or receives chat messages with the other users in the room.	User	The user is in a room.	The user sends or receives chat messages with the other users in the room.

Table 2: Use Case Descriptions

3.3 Result of application after deploying

Render.com is a cloud platform that provides services related to application and website deployment.

Here's an introduction to deploy your website on render.com with the supports of github

Create a GitHub Repository: If you haven't already, create a GitHub repository for your project. Push your code to this repository.

webrtc_midtermnodejs Public

main 1 branch 0 tags

Go to file Add file <> Code

dominhquan2003 first commit 9beb64c 32 minutes ago 1 commit

File	Commit	Time
src	first commit	32 minutes ago
.gitignore	first commit	32 minutes ago
README.md	first commit	32 minutes ago
package-lock.json	first commit	32 minutes ago
package.json	first commit	32 minutes ago

README.md

Web RTC video call

A conference call implementation using WebRTC, Socket.io and Node.js.

Getting Started

clone source https://github.com/dominhquan2003/webrtc_midtermnodejs.git

- Run `npm install`
- `cd src`
- `node app.js` or `npm start`

NOTE

About
No description, website, or topics provided.

Readme
Activity
0 stars
1 watching
0 forks

Releases
No releases published
[Create a new release](#)

Packages
No packages published
[Publish your first package](#)

Languages

Language	Percentage
JavaScript	64.5%
HTML	24.7%
CSS	10.8%

Suggested workflows
Based on your tech stack

Figure 15: Example repository of github

Sign Up on Render.com: Sign up for an account on Render.com.

Create a New Service on Render: After signing in, click on the "+" button to create a new service. Choose the type of service you want to deploy (e.g., Web Service for a web application).

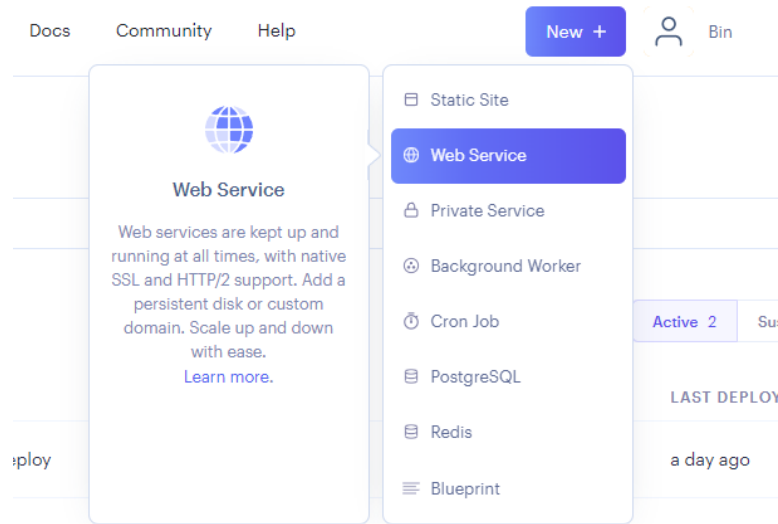


Figure 16: Create a New Service on Render

Connect GitHub Repository: Under the "Repository" section, choose the repository where your code is hosted on GitHub.

Connect a repository

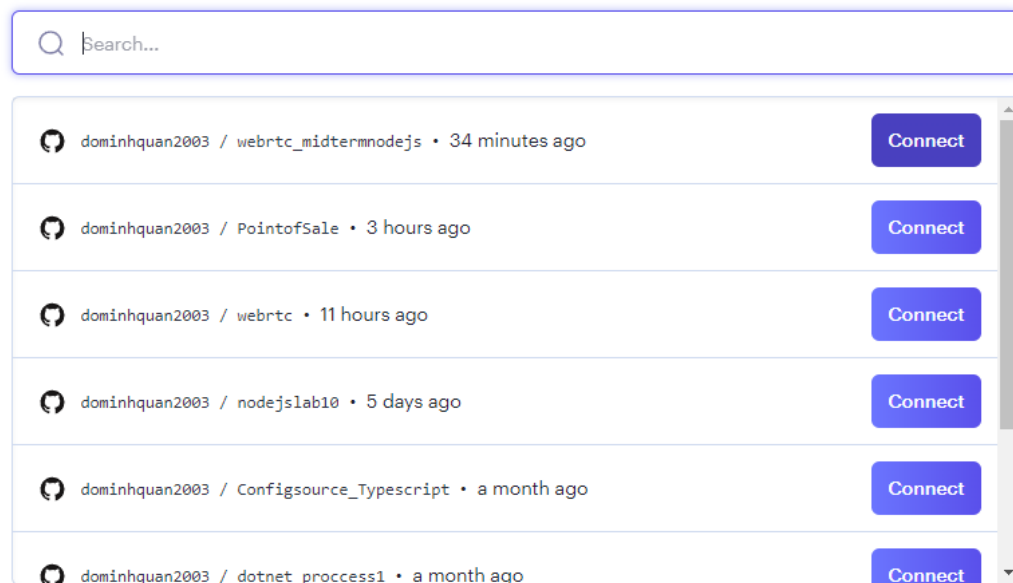


Figure 17: Connect GitHub Repository

Configure Build Settings: Configure build settings if needed. Render.com can automatically detect many types of applications, but you might need to specify the build command, environment variables, or other settings.

You are deploying a web service for [dominhquan2003/webrtc_midtermnodejs](#).

You seem to be using Node, so we've autofilled some fields accordingly. Make sure the values look right to you!

<p>Name A unique name for your web service.</p>	<input type="text" value="midterm-nodejsOfficial"/>
<p>Region The region where your web service runs. Services must be in the same region to communicate privately and you currently have services running in Singapore.</p>	<input type="text" value="Singapore (Southeast Asia)"/>
<p>Branch The repository branch used for your web service.</p>	<input type="text" value="main"/>
<p>Root Directory Optional Defaults to repository root. When you specify a root directory that is different from your repository root, Render runs all your commands in the specified directory and ignores changes outside the directory.</p>	<input type="text" value="E.g., src"/>
<p>Runtime The runtime for your web service.</p>	<input type="text" value="Node"/>
<p>Build Command This command runs in the root directory of your repository when a new version of your code is pushed, or when you deploy manually. It is typically a script that installs libraries, runs migrations, or compiles resources needed by your app.</p>	<input type="text" value="\$ yarn"/>

Figure 18: Configure Build Settings

Deploy: Click the "Create Web Service" or equivalent button to deploy your application.

CHAPTER 4 – SUMMARY

4.1. User interface

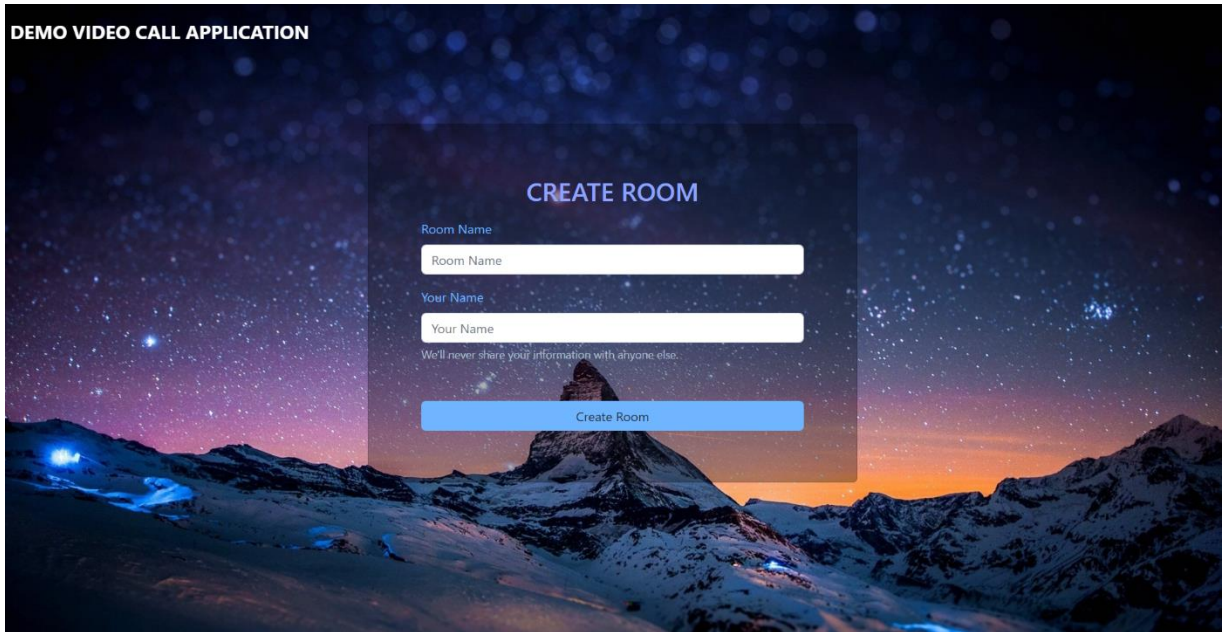


Figure 19: Room creation interface

To create a room and invite others to join, users typically enter their name and the desired name of the room. This process simplifies the setup and allows users to quickly establish a collaborative space. Here's an expanded explanation of the process:

- Entering User Name
- Naming the Room
- Creating the Room
- Sharing the Invitation
- Joining the Room

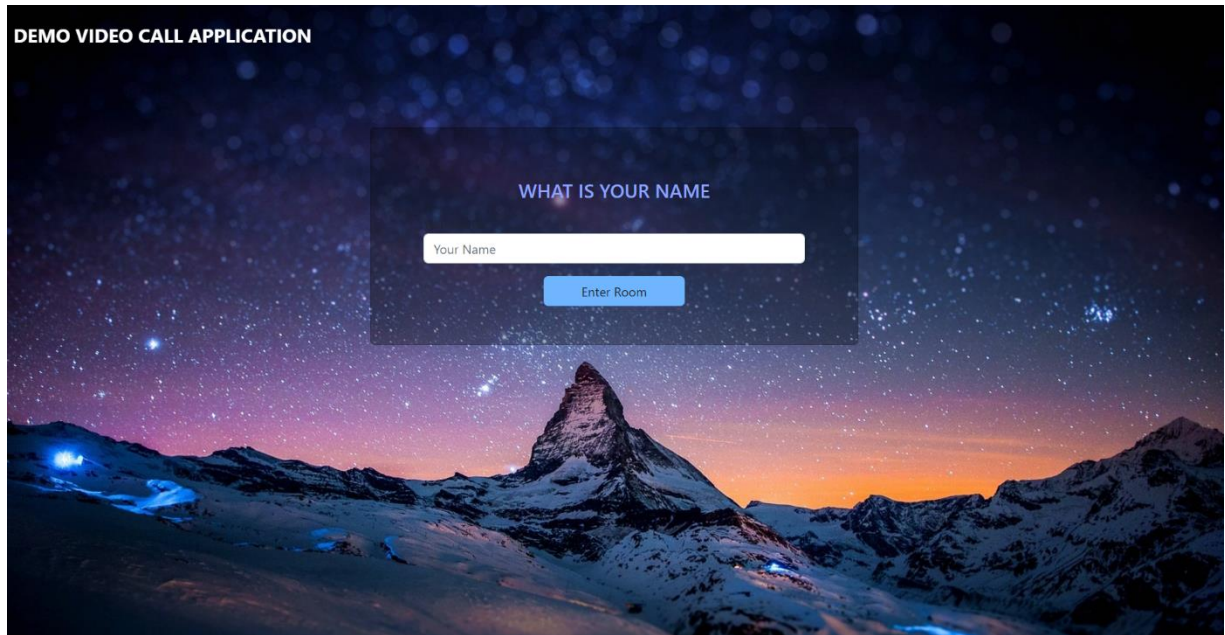


Figure 20: Interface for room entry

In room entry, user just type name and click “enter room” button in order to join room with other users

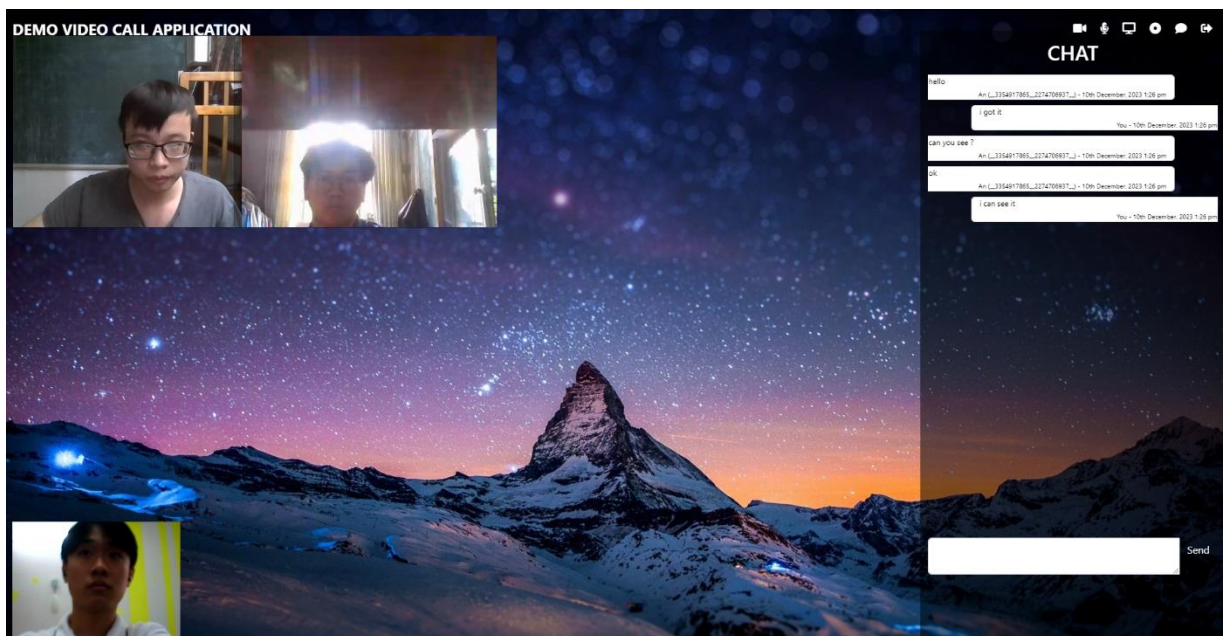


Figure 20: Interface video call with chat after deploy

In this interface, demonstration for meeting with many user

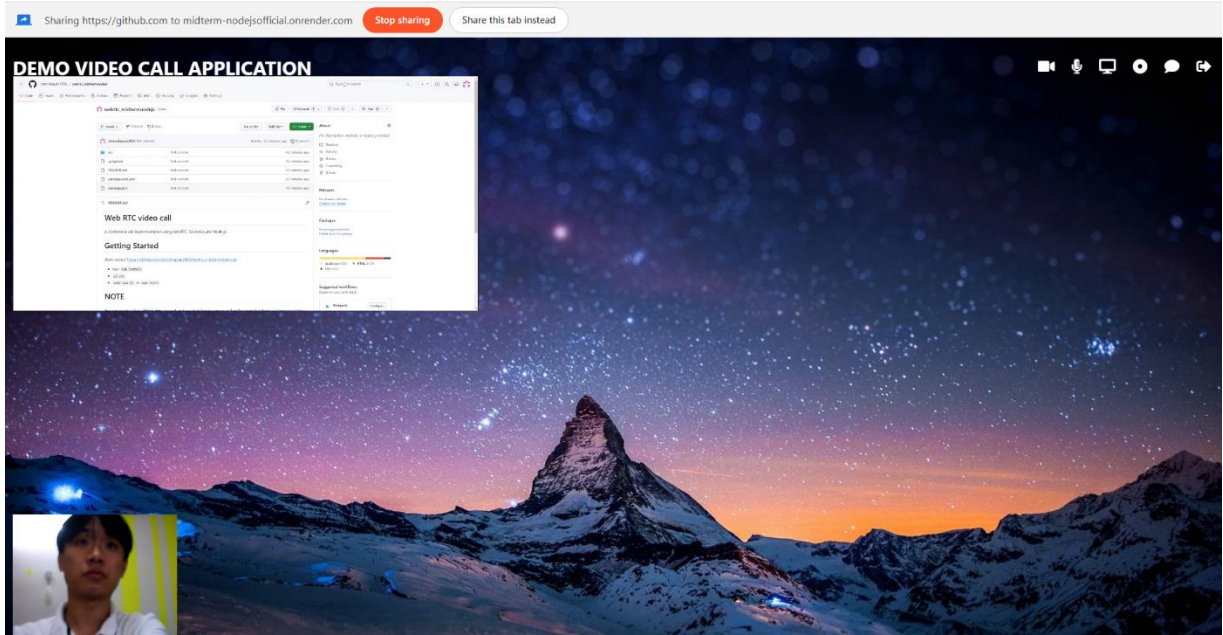


Figure 21: Interface share screen

Once the user wants to view the screen shared by other users, they can initiate the process by clicking on a button with an icon resembling a screen. This button is typically located within the application's user interface, specifically designed to facilitate screen sharing and collaboration.

Upon clicking the screen-sharing button, a dialog box or menu will appear, presenting the user with various options for sharing screens. This dialog box serves as a convenient and user-friendly interface, providing the user with flexibility and control over the screen-sharing process..

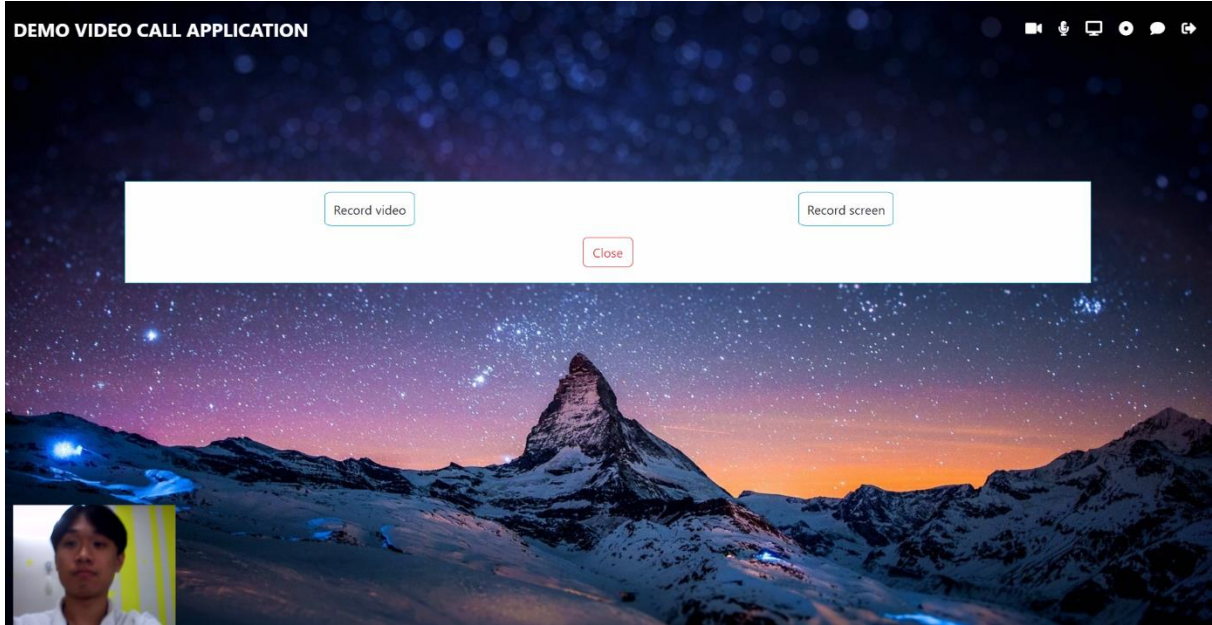


Figure 22: Interface record function

After clicking “record” button, we can see a dialog popped up for user choose. Now, user have two selections the first is “Record Video” button which is used for recording video of users during meeting and the second is “Record Screen” button which is used for recording all of screens of web during meeting.

After the recording process is complete, it is common for applications to provide users with the option to name the file and choose a specific path to save the recorded video on their computer. This functionality allows users to organize and save the recorded video in a location of their choice, making it easily accessible for further tasks or sharing.

4.2. Limitations

NAT Traversal: WebRTC uses STUN (Session Traversal Utilities for NAT) servers to discover public IP addresses and traverse NAT (Network Address Translation) devices. However, certain network configurations, such as symmetric NATs or firewalls, can still pose challenges for establishing direct peer-to-peer connections. In such cases, TURN (Traversal Using Relays around NAT) servers act as relay points to facilitate communication, but they introduce additional latency.

Firewall Restrictions: Some firewalls or network configurations may block WebRTC traffic, especially if it is not using standard ports (80, 443). This can prevent successful connection establishment or cause connectivity issues.

Network Bandwidth and Quality: WebRTC applications heavily rely on network bandwidth and quality to deliver real-time audio and video streams. Insufficient bandwidth or high network latency can lead to degraded audio/video quality or dropped connections.

TURN Server Costs: While using a TURN server can improve connectivity in challenging network scenarios, it introduces additional costs. TURN servers consume bandwidth and may require significant server resources, especially for high-traffic applications. Consider the cost implications and scalability requirements when deploying a TURN server.

Server-Side Implementation: WebRTC applications typically require server-side implementation to handle signaling, user authentication, and managing peer connections. This server-side code needs to be carefully designed and secured to prevent potential security vulnerabilities.

Deployment and Scaling: Deploying WebRTC applications can be more complex than traditional web applications due to the real-time nature and peer-to-peer connections. Scaling can be challenging, especially for large-scale deployments with high concurrency. Proper load balancing, clustering, and infrastructure planning are essential to ensure smooth operation and scalability.

Browser Compatibility: While WebRTC is supported by most modern browsers, there might still be slight differences in implementation, leading to compatibility issues across different browser versions. It's important to test and ensure compatibility across target browsers.

Security Considerations: WebRTC applications involve transmitting audio and video data between peers, so security measures like encryption and authentication are crucial to protect user privacy and prevent unauthorized access to media streams.

4.3. Future Directions

Improved Interoperability: Enhancing interoperability across browsers and platforms is a crucial area for WebRTC development. Efforts are underway to ensure consistent implementation of WebRTC APIs across different browsers, reducing inconsistencies and improving cross-browser compatibility.

Scalability and Performance: WebRTC applications often require scalability and efficient handling of large numbers of concurrent connections. Future developments may focus on improving the scalability and performance of WebRTC implementations, enabling them to handle higher traffic loads and increasing the number of simultaneous connections.

Advanced Codec Support: WebRTC currently supports codecs like VP8 and Opus, which provide good quality and compression. However, there is ongoing research and development in video and audio codecs, such as AV1 and Opus enhancements, which offer better compression efficiency and improved quality. Future WebRTC updates may integrate support for these advanced codecs.

Augmented Reality (AR) and Virtual Reality (VR): Integrating WebRTC with AR and VR technologies holds great potential for immersive real-time communication experiences. Future developments may focus on enabling WebRTC to facilitate real-time sharing of AR/VR content, enabling collaborative experiences and remote interactions.

Simulcast and Scalable Video Coding (SVC): Simulcast allows transmitting multiple video streams with different resolutions and quality levels, which can be beneficial for adapting to diverse network conditions and device capabilities. Similarly, SVC enables encoding video streams in a way that allows decoding at different quality levels. Future advancements in WebRTC could incorporate native support for simulcast and SVC to optimize video transmission and playback.

Advanced Network Congestion Control: WebRTC relies on congestion control algorithms to adapt to changing network conditions. Ongoing research aims to improve these algorithms, allowing WebRTC applications to better handle congestion scenarios, reduce latency, and optimize bandwidth utilization.

TASK ASSIGNMENT AND EVALUATION

Task Assignment

Full name	Time	Content of work assignment
Do Minh Quan	01/10/2023 - 20/10/2023	Discuss as a team, align on ideas and the technology to be used, and delegate tasks. Research on the technology for creating a chat and video call app using WebRTC and Socket.IO. Researching stun and turn server and Presenting Chapter 1
	5/11/2023 - 2/12/2023	Perform video call function and turn on/off the microphone
	4/12/2023 - 13/12/2023	Push code to github and research how to deploy web on render.
Ho Huu An	01/10/2023 - 16/10/2023	Discuss as a team, align on ideas and the technology to be used, and delegate tasks. Research on the technology for creating a chat and video call app using WebRTC Making plan to design UI and researching all webrtc famous before like zoom, discord, google meet
	20/10/2023- 27/11/2023	Implement room creation, chat functions, record function log out function and share screen
	29/11/2023 - 14/12/2023	Implement content of chapter 2, chapter 3 and chapter 4

Table 3: Task Assignment

Member evaluation

Full name	Assigned tasks (100%)	Completion level
Do Minh Quan	50%	On schedule as assigned
Ho Huu An	50%	On schedule as assigned

Table 4: Member evaluation