

SECURE MULTIMEDIA CHAT APPLICATION

1. CHAPTER ONE

1- Two METHODS OF COMMUNICATION

In this project, both **peer-to-peer (P2P) communication** and **optional end-to-end encryption (E2EE)** are utilized, providing users with flexibility in their communication choices. P2P communication offers distinct privacy advantages, ensuring direct and secure exchanges between users. However, it requires both parties to be online simultaneously for effective communication. On the other hand, E2EE offers the advantage of offline communication capabilities, allowing users to exchange encrypted messages even when not connected to the network. By offering both modes of communication, users have the freedom to select the approach that best suits their specific needs and circumstances. Whether prioritizing enhanced privacy through P2P communication or the convenience of offline communication with E2EE, users can opt for the mode that aligns with their preferences.

This project aims to empower users by providing them with a range of choices in terms of communication modes, enabling them to tailor their experience according to their desired level of privacy, availability, and convenience. By incorporating both P2P communication and E2EE, the project seeks to deliver a versatile and secure communication solution that caters to the diverse needs of its users.

1.1 Peer to Peer Communication

1.1.1 This Technology used is Web Rtc:

Stream Control Transmission Protocol (SCTP)

The Stream Control Transmission Protocol (SCTP), a transport layer protocol designed to replace TCP or UDP, is what WebRTC utilizes. We utilize WebRTC as an application layer protocol for our DTLS connection while using it for WebRTC.



Figure 1 - webRtcImage

This project leverages the power of peer-to-peer communication within web browsers using the built-in webRTC technology, which seamlessly integrates with any server. The project implements a signaling process to facilitate the initiation of connections between users. By employing webRTC, the application enables direct communication between users without relying on centralized servers, enhancing security and privacy. The signaling process serves as a crucial mechanism for users to establish and coordinate their connections, ensuring a smooth and secure communication experience. With the utilization of webRTC and the signaling process, this project aims to provide a robust and user-friendly platform for secure peer-to-peer communication in web browsers, allowing users to exchange information in a private and protected manner.

1.1.2 Web RTC Working

WebRTC is a technology that allows for real-time communication within web browsers without the use of extra plugins or software, including audio and video chat. WebRTC working:

Establishing a Connection

The WebRTC process begins with two parties, typically referred to as peers, who want to establish a direct connection. Each peer creates a session description, which includes information about their media capabilities, such as supported audio and video codecs. Peer A generates an offer and Peer B generates an answer. These offers and answers contain the session descriptions and are exchanged between the peers.

Signaling

The offers and answers are not sent directly between the peers. Instead, they are exchanged through a process known as signaling. Signaling is the mechanism used to exchange network information between the peers, allowing them to coordinate and establish a connection. The signaling process can use different methods, such as WebSocket, HTTP, or a signaling server, to send the session descriptions between the peers.

Media Stream Setup

Once the peers have established a direct connection using ICE, they can begin setting up their media streams. Each peer can access the user's media devices, such as a camera or microphone, using the browser's APIs. The media streams are captured from the devices and encoded into a suitable format for transmission. The Real-Time Transport Protocol (RTP) and Secure Real-Time Transport Protocol (SRTP) are used to encrypt the media before it is transferred across the established peer-to-peer connection.

Media Processing and Rendering

On the receiving end, the media streams are received, decoded, and processed. The audio and video data are rendered using the browser's APIs, allowing the users to see and hear each other in real-time. The rendered media is displayed on the respective user interfaces, enabling the real-time communication experience.

Closing the Connection

When the communication session is complete, either peer can initiate the closing of the connection. The closing signal is sent to the other peer, and both peers perform the necessary cleanup tasks to release the resources and close the connection. WebRTC is a complex technology that involves several protocols and mechanisms working together to enable real-time communication. This step-by-step explanation provides a simplified overview of the process, highlighting the key stages involved in establishing and maintaining a WebRTC connection.

1.1.3 Peer to peer Online testing

Successfully hosted and tested at the given domain



Figure 2 - peer to peer online testing

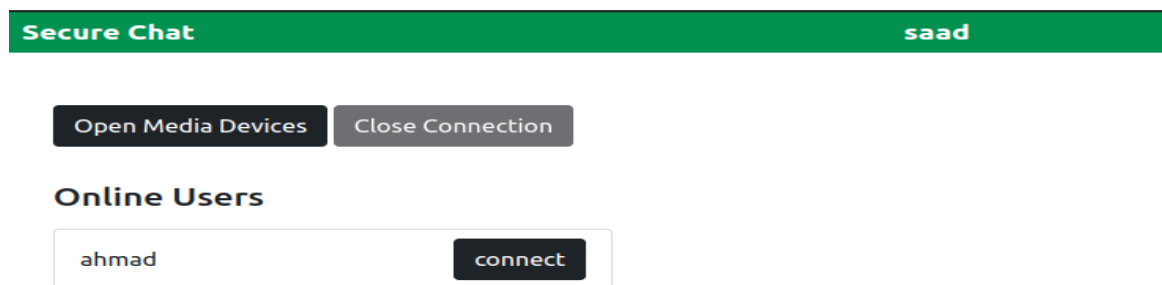


Figure 3 - Online peer to peer demo

1.2 End to End Encryption

This project employs end-to-end encryption, utilizing the secure and robust RSA algorithm in asymmetric encryption, in a server mode communication setting. Unlike peer-to-peer mode, this approach eliminates the need for both users to remain online for communication to occur. Through the implementation of end-to-end encryption, the project ensures that data remains confidential and protected throughout its transmission, with only the intended recipients possessing the means to decrypt and access the information. By leveraging the RSA algorithm, known for its strength and reliability, the project establishes a highly secure communication framework. The server mode communication not only guarantees enhanced security but also provides flexibility for users, enabling asynchronous communication without the need for both parties to be online simultaneously. This project aims to deliver a secure and efficient communication solution, maintaining the utmost privacy for users' information while accommodating their varied availability and ensuring seamless communication experiences.

1.3 RSA

In the project, asymmetric encryption is implemented using the RSA algorithm for secure communication. Using a key pair, a public key and a private key is essential for asymmetric encryption. Its power and security in asymmetric encryption have earned the RSA algorithm much acclaim.

In this project, data is encrypted and decrypted during transmission using the RSA technique. Every user has a special key pair, consisting of a public key that is accessible to anybody wishing to send them an encrypted message and a private key that is kept private and only known to the intended recipient.

A user generates their key pair and stores the private key safely on their device before starting a secure conversation. To encrypt communications meant for the recipient, the public key is sent to other users. The message can only be decoded using the recipient's private key after being encrypted using the recipient's public key, guaranteeing message secrecy.

By taking use of the computational difficulty of prime factorization, the RSA algorithm offers a strong and trustworthy mechanism for secure communication. The project's implementation of it assures that the transmitted data is kept private and secure against unwanted access.

By utilizing RSA for asymmetric encryption, this project establishes a strong foundation for secure communication, ensuring that sensitive information remains secure and accessible only to authorized recipients.

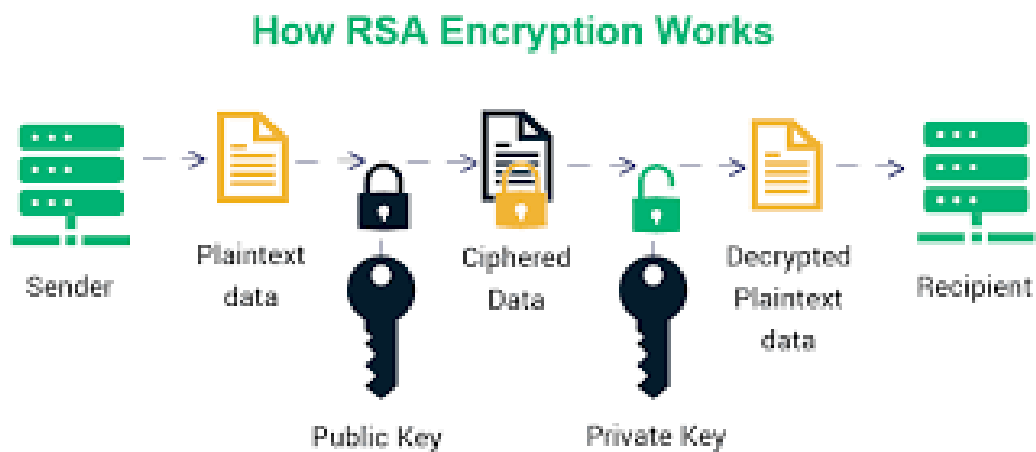


Figure 4 - RSA Algorithm

1.4 RSA Algorithm:

A popular cryptographic technique for safe key exchange and communication is the RSA algorithm. For encryption and decryption, public and private keys are utilized.

How the RSA Works:

Generation of Key:

Pick p and q , two separate prime numbers.

$n = p * q$, which is the product of the two. This modulus is what the algorithm uses.

Calculate the totient function of Euler: $\phi(n) = (p - 1) * (q - 1)$. The number of positive integers smaller than n that are relatively prime to n is determined by this function.

Select an integer e such that $\gcd(e, n) = 1$ and e is coprime with n . In other words, $1 < e < n$. The public exponent will be this number, e .

Determine e modulo (n) 's modular multiplicative inverse. The private exponent, which is commonly represented by the letter d , will be this inverse.

Encryption

A message M can be encrypted by representing it as a number in the $[0, n-1]$ range.

Calculate the ciphertext C by using the recipient's public key (e, n) : $C = M^e \pmod{n}$.

The encrypted message is transferred securely as ciphertext C .

Decryption

Make use of the recipient's private key (d, n) to decrypt the ciphertext C .

Use the following formula to calculate the plaintext message M : $C^d \pmod{n} = M$.

The original message that was encrypted is contained in the resultant M .

That was a general description of the RSA algorithm. It is essential to remember that RSA's security depends on how hard it is to factor huge integers.

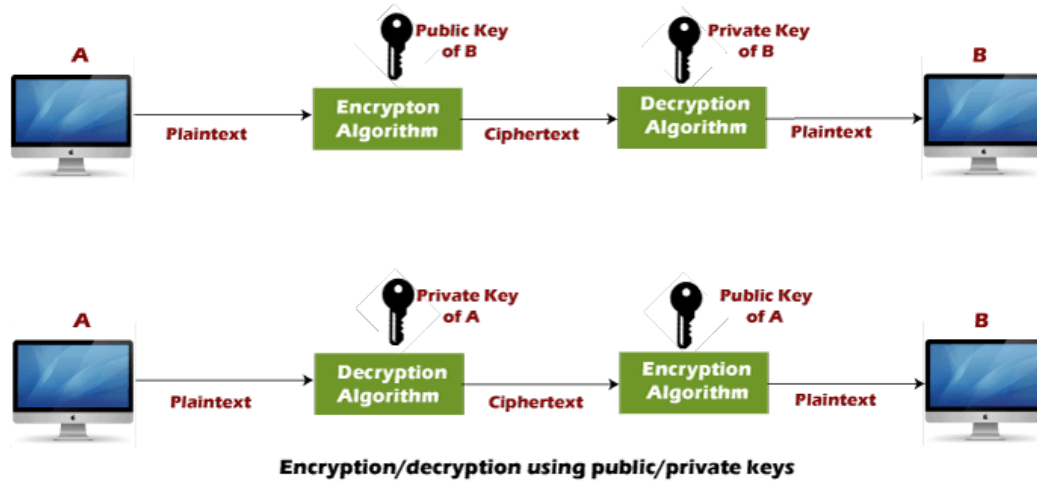


Figure 5 - RSA Algorithm

1.5 RSA Algorithm Code:

```
function generateKeys(keysize) { ...
}

function encrypt(encodedMsg, n, e) {
  return bigInt(encodedMsg).modPow(e, n);
}

function decrypt(encryptedMsg, d, n) {
  return bigInt(encryptedMsg).modPow(d, n);
}

function encode(str) {
  const codes = str
    .split('')
    .map(i => i.charCodeAt())
    .join('');
  return bigInt(codes);
}
```

Figure 6 - Encryption and Decryption in RSA


```

function decode(code) {
  const stringified = code.toString();
  let string = '';

  for (let i = 0; i < stringified.length; i += 2) {
    let num = Number(stringified.substr(i, 2));

    if (num <= 30) {
      string += String.fromCharCode(Number(stringified.substr(i, 3)));
      i++;
    } else {
      string += String.fromCharCode(num);
    }
  }

  return string;
}

```

Figure 7 - Decoding after decryption

```

const message = 'random messages';

const keys = generateKeys(500);
const encoded_message = encode(message);
const encrypted_message = encrypt(encoded_message, keys.n, keys.e);
const decrypted_message = decrypt(encrypted_message, keys.d, keys.n);
const decoded_message = decode(decrypted_message);
console.log('Correct?', message === decoded_message);

if(message === decoded_message){

  let publicKey = [keys.n, keys.e];
  let privateKey = [keys.d, keys.n];

  // store private key in local stroage

  localStorage.setItem(username, JSON.stringify(privateKey));

  const data = {
    username,
    password,
    publicKey
  };
}

```

Figure 8 - function calling and storing data in the given format

1.6 Technology Used:

- **JavaScript**

JavaScript is a flexible programming language used for client-side web development that allows for dynamic components and interactivity on webpages.

- **Socket.io**

Real-time, two-way communication between clients and servers is made possible via a JavaScript library, which also makes event-based messaging and reconnection management easier.

- **Node.js**

Node.js is commonly used for web servers and APIs, the server-side JavaScript runtime environment enables scalable and high-performance network applications.

- **MongoDB**

Unstructured data may be stored using a NoSQL database system that offers scalability, high availability, and distributed data support. frequently applied in big data analytics and online applications.

2. CHAPTER TWO

2 CHAT APPLICATION DETAILED OVERVIEW

2.1 Application Main Page

Users first visit the home page of our web-based secure multimedia chat application by clicking on a link that is supplied. Once there, users are given the choice of signing up or logging in. Users can carry out these actions anyway they choose by clicking the respective buttons.

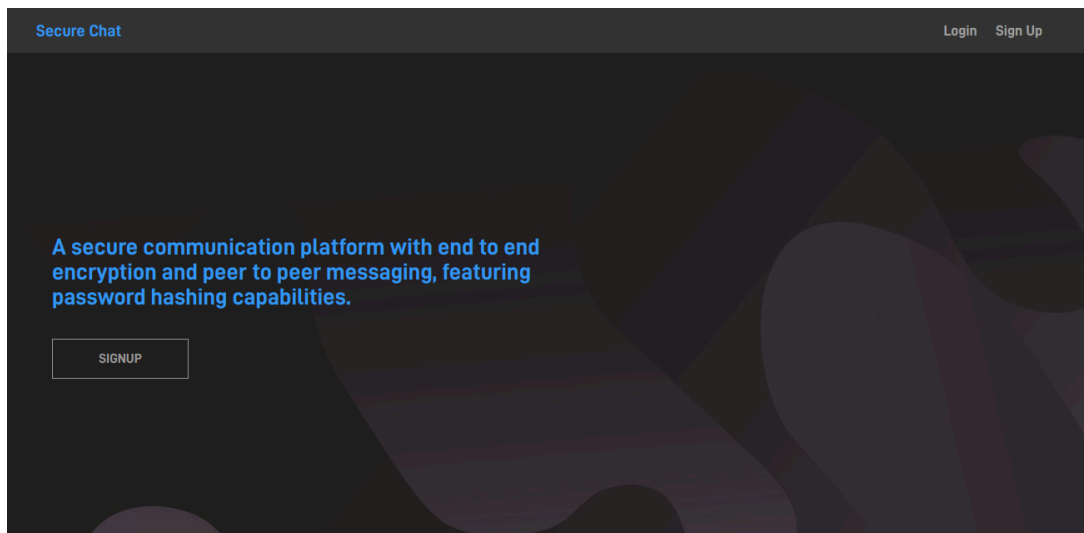


Figure 9 - Application Main Page

2.2 Application Login Page

The login page is the page of our program that comes after the main page. Users simply need to provide a username to register here. We don't require your first name, last name, or date of birth, unlike some other programs, in order to log in. A user must first register and establish an account if they don't already have one. Once created, users may log in to the program using that account.

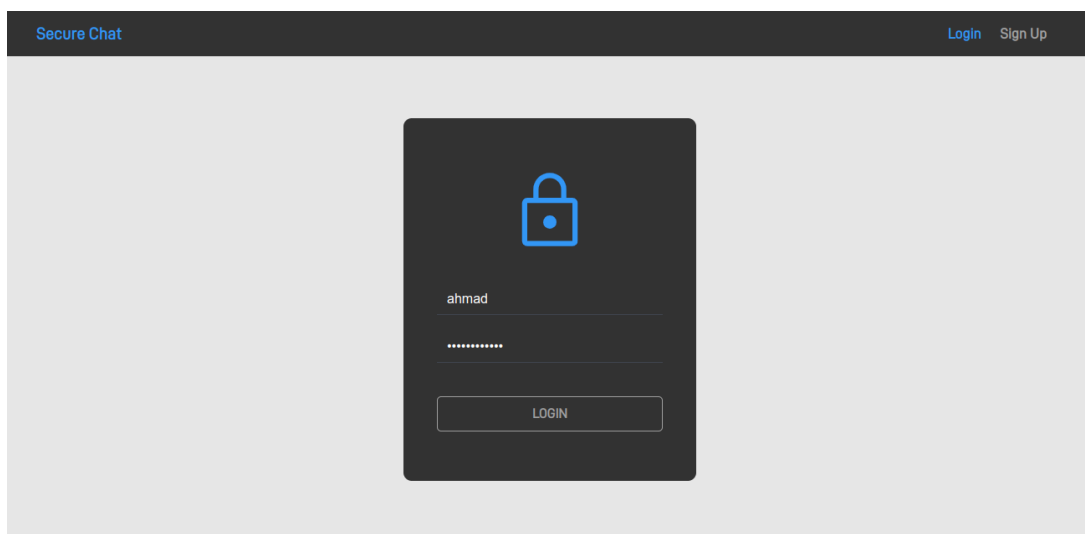


Figure 10 - Application Login Page

2.3 Application Chat Page

The user will be sent to our application's chat page after successfully logging in. A list of users who are online is displayed here, each with a green dot denoting their level of availability. Users can send and receive messages using the messaging window on the chat page.

There are also two additional choices available. The first choice is to switch to peer-to-peer mode, which allows for direct user contact. Calling someone is the second choice. Users can complete these tasks by simply clicking on the corresponding buttons that are available.

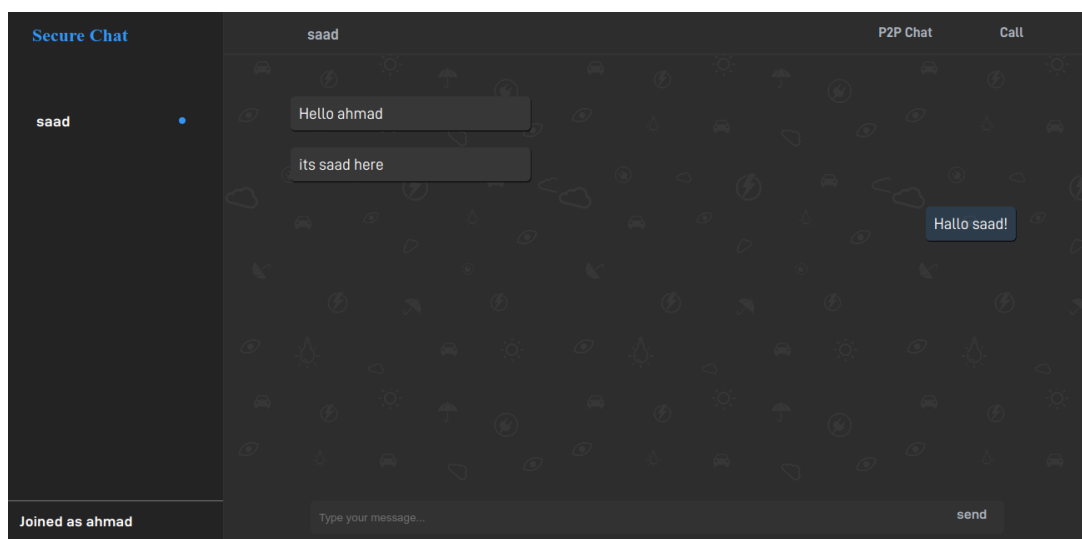


Figure 11 - Application Chat Page

2.4 Encrypted messages stored in database

This shows how users encrypted message store in the database and database owner have not access to the users' original messages.



Figure 12 - Database Encryption message stored

This is a database record of how user profile data store in the database it also represents the structure of the database for users' collection

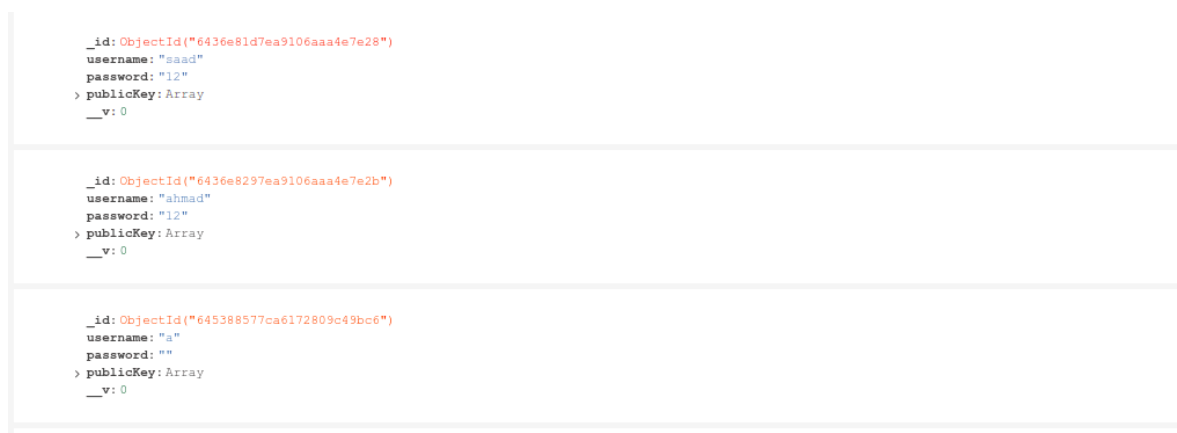


Figure 13 - Storing Decrypted Messages

2.5 Requesting for Peer-to-Peer Chat

In our application we are providing a mode of switching mode from the normal mode and this action will be done by pressing the button p2p chat which is available on the user screen. The pop-up chat button is clicked by any user and a requesting pop-up will be generated on the user's screen for waiting to switch to peer-to-peer chat mode.

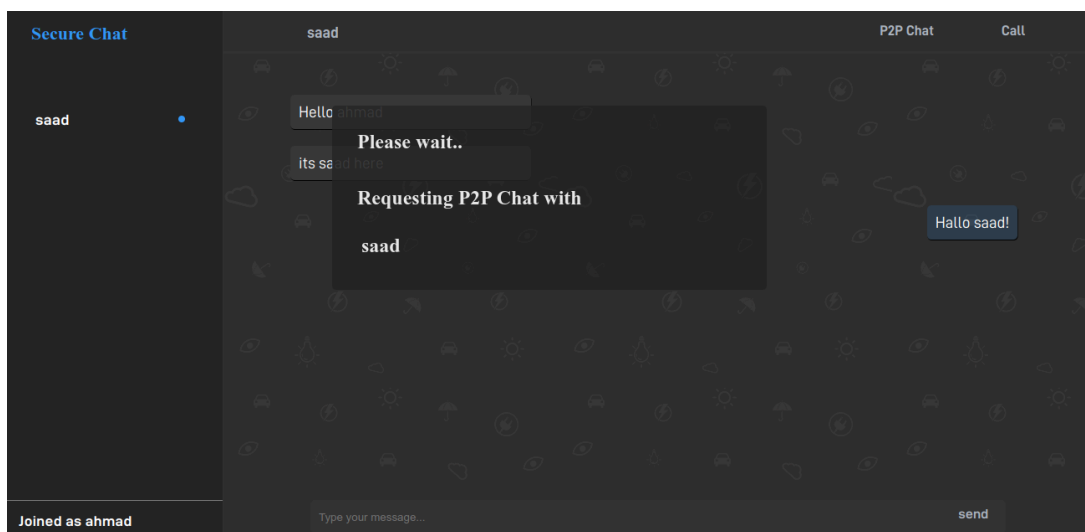


Figure 14 - peer to peer Request

2.6 Request Acceptance and Rejection

After clicking the button of peer to peer chat another popup will show on the second user's screen and on that pop up two options will available first one is to except the request and other one is to reject the request think that request both users will shift to peer-to-peer communication chat mode.

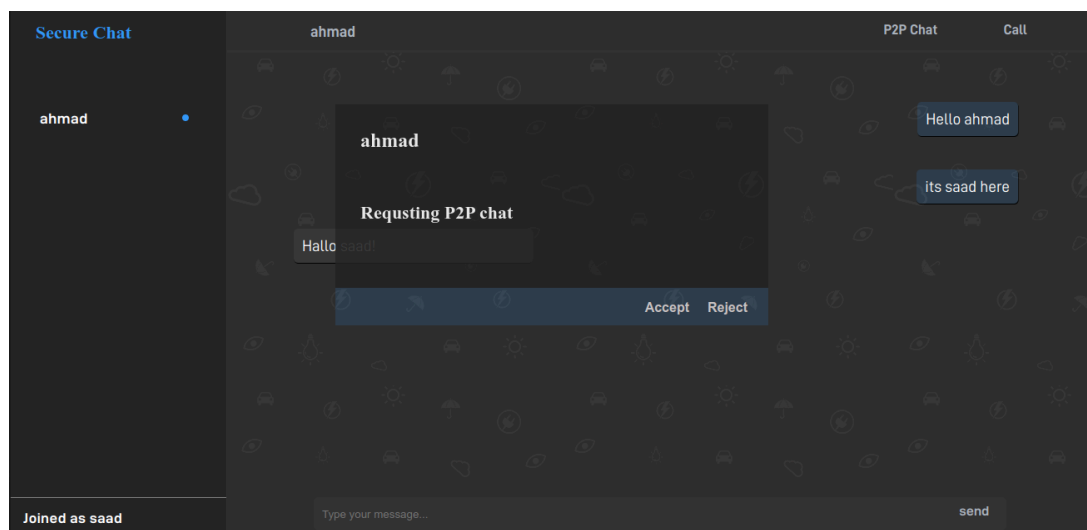


Figure 15 - Request Acceptance and Rejection

2.7 Peer To Peer Chat Mode

This is peer to peer communication chat mode. In this mode the users can connect and speak with one another directly without the usage of a centralized server while using peer-to-peer communication chat mode. Since the data doesn't have to pass via a third-party server, it enables quicker communication and better privacy. Direct connections between users' devices, however, can necessitate further configuration.

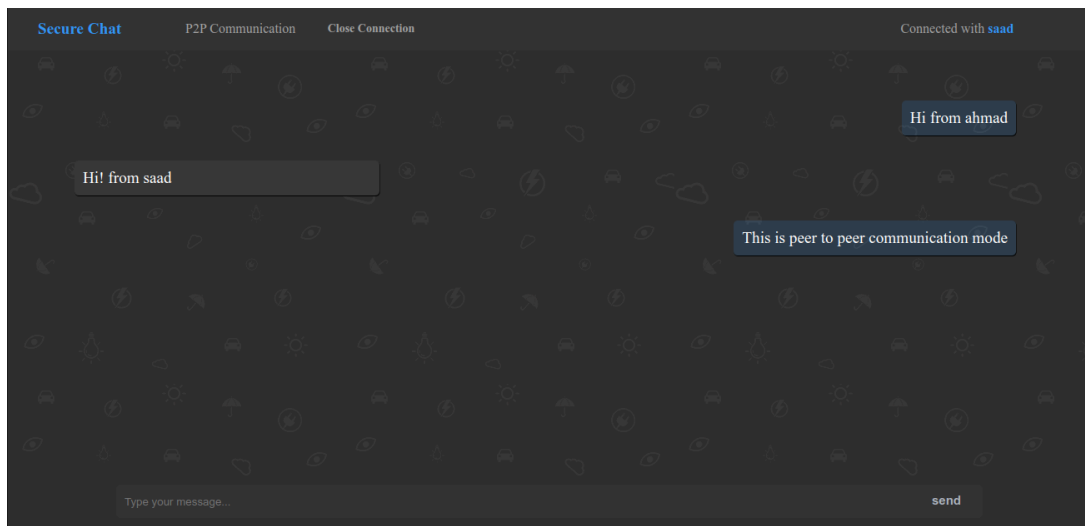


Figure 16 - Peer to Peer Chat Mode

2.8 Peer To Peer Calling

In our application we are providing an open of calling at peer-to-peer mode the video calling is made possible in peer-to-peer mode by creating a direct link between the participants' devices. Each user's device records video and audio streams, which are then encoded and transferred straight to the other user's device across the established link. Low latency and effective communication are the results of this direct connection, which enables real-time video and audio transmission without the need for the data to transit via a centralized server.

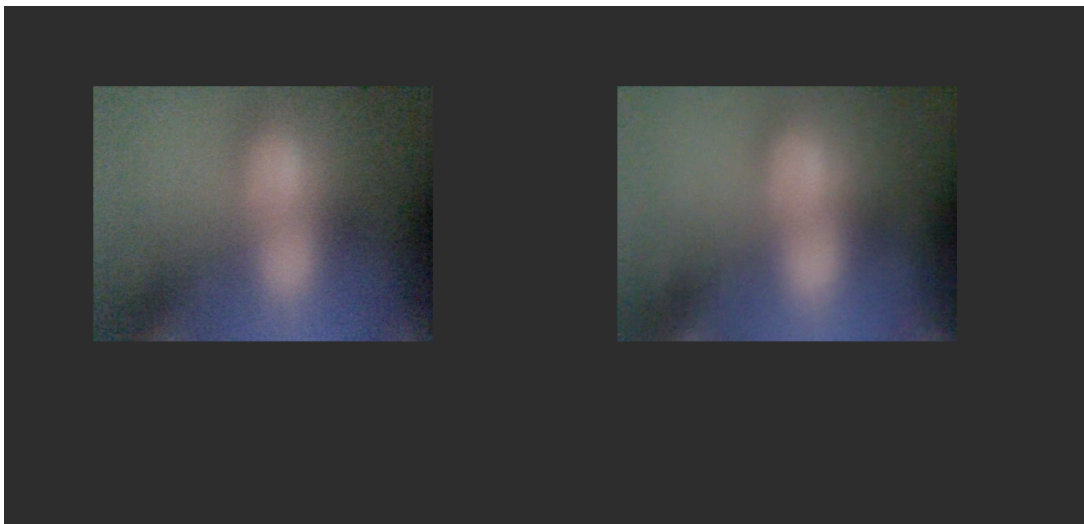
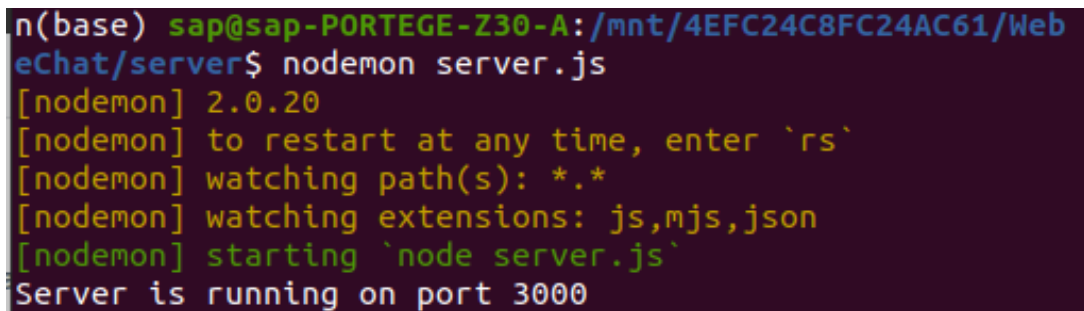


Figure 17 - Peer to Peer Calling

2.9 Server Connective for peer-to-peer communication

In our application we use server for peer-to-peer communication as server facilitates information transmission and creates a direct peer-to-peer link. Direct peer-to-peer communication might be hampered by Network Address Translation. The user experience can be improved by a server's ability to retain and forward messages when a user is not connected, ensuring that messages are delivered when they reconnect.

A terminal window with a dark background and light-colored text. The prompt is 'n(base) sap@sap-PORTEGE-Z30-A: /mnt/4EFC24C8FC24AC61/Web eChat/server\$'. The command 'nodemon server.js' has been executed. The output shows the version '2.0.20', instructions to restart with 'rs', the paths and extensions being watched, and the message 'Server is running on port 3000'.

```
n(base) sap@sap-PORTEGE-Z30-A: /mnt/4EFC24C8FC24AC61/Web eChat/server$ nodemon server.js
[nodemon] 2.0.20
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node server.js`
Server is running on port 3000
```

Figure 18 - Server