Table of Contents

[1. Introduction 4](#_Toc184570581)

[**1.1.** **Brief Overview:** 4](#_Toc184570582)

[**1.2.** **Objectives:** 4](#_Toc184570583)

[**1.3.** **Setup:** 4](#_Toc184570584)

[2. Features or Highlights 4](#_Toc184570585)

[**2.1.** **Text Messaging:** 4](#_Toc184570586)

[2.1.1. End-to-end encryption 4](#_Toc184570587)

[2.1.2. Message storage 4](#_Toc184570588)

[**2.2.** **File Sharing:** 4](#_Toc184570589)

[**2.3.** **Voice Messaging** 4](#_Toc184570590)

[**2.4.** **Real-time Connectivity** 4](#_Toc184570591)

[**2.5.** **Emoji Picker** 4](#_Toc184570592)

[**2.6.** **Active User Management** 5](#_Toc184570593)

[**2.7.** **Encryption** 5](#_Toc184570594)

[3. Workflow/Architecture 5](#_Toc184570595)

[**3.1.** **Step-by-Step Process:** 5](#_Toc184570596)

[3.1.1. Signaling 5](#_Toc184570597)

[**3.1.2.** **Peer Connections** 5](#_Toc184570598)

[**3.1.3.** **Data Transmission** 5](#_Toc184570599)

[4. Screenshots or Outputs 6](#_Toc184570600)

[**4.1.** **Visual Representation of the Project:** 6](#_Toc184570601)

[**4.1.1.** **Chat Interface:** 6](#_Toc184570602)

[**4.1.2.** **File Sharing:** 6](#_Toc184570603)

[**4.1.3.** **Voice Messaging:** 6](#_Toc184570604)

[**4.1.4.** **Active Users:** 6](#_Toc184570605)

[**4.1.5.** **Encryption:** 6](#_Toc184570606)

[5. Challenges and Solutions 11](#_Toc184570607)

[**5.1.** **Briefly Describe Hurdles:** 11](#_Toc184570608)

[5.1.1. Connectivity Issues 11](#_Toc184570609)

[5.1.2. Encryption and Security: 11](#_Toc184570610)

[5.1.3. User Interface Design: 11](#_Toc184570611)

[5.1.4. Error Handling 11](#_Toc184570612)

[**5.2.** **Solutions:** 11](#_Toc184570613)

[5.2.1. Connectivity 11](#_Toc184570614)

[5.2.2. Encryption 11](#_Toc184570615)

[5.2.3. UI Design 11](#_Toc184570616)

[5.2.4. Error Handling 11](#_Toc184570617)

[6. Conclusion and Future Work 11](#_Toc184570618)

[**6.1.** **Summary of Results:** 11](#_Toc184570619)

[6.1.1. Successes 11](#_Toc184570620)

[6.1.2. Lessons Learned 11](#_Toc184570621)

[**6.2.** **Potential Improvements:** 11](#_Toc184570622)

[6.2.1. Scalability 11](#_Toc184570623)

[6.2.2. Advanced Features 11](#_Toc184570624)

[6.2.3. User Feedback 11](#_Toc184570625)

[6.2.4. Security 11](#_Toc184570626)

[**6.3.** **Next Steps:** 12](#_Toc184570627)

[6.3.1. Develop video calling functionality. 12](#_Toc184570628)

[6.3.2. Implement better user management. 12](#_Toc184570629)

[6.3.3. Enhance error handling and recovery mechanisms. 12](#_Toc184570630)

[6.3.4. Conduct user testing: 12](#_Toc184570631)

# Introduction

## **Brief Overview:**

* + What: Describe the core idea behind your project. A P2P messenger application enables direct communication between users without a centralized server.
  + Why: Discuss the importance of building a P2P messenger. What challenges does it address compared to traditional client-server architectures?

## **Objectives:**

* + - To build a secure and efficient P2P messenger using WebRTC for real-time communication.
    - To implement encryption for secure message transmission.
    - To support features like text messaging, file sharing, and voice messaging.

## **Setup:**

To run this P2P messenger, users must have Node.js installed on their system. The application can be started by navigating to the project directory and executing the command: npm start

# Features or Highlights

## **Text Messaging:**

* + 1. End-to-end encryption: Messages are encrypted before transmission and decrypted on the receiver’s end to ensure privacy.
    2. Message storage: Stored temporarily in local storage and not on a server, minimizing privacy risks.

## **File Sharing:**

Supports sending and receiving various file types (documents, images, audio, video).

Utilizes asynchronous transfer protocols to handle large files efficiently.

* 1. **Voice Messaging:**

Enables users to send voice messages via WebRTC.

Messages are recorded, encoded, and transmitted as audio files.

* 1. **Real-time Connectivity:**

Establishes peer-to-peer connections via WebRTC, allowing low-latency communication.

Utilizes STUN and TURN servers to handle NAT traversal and network reliability.

* 1. **Emoji Picker:**

Allows users to insert emojis into their messages, enhancing interaction.

* 1. **Active User Management:**

Displays the list of active users, their statuses, and real-time connection status.

Updates when users join or leave the chat.

* 1. **Encryption:**

Uses advanced encryption algorithms (e.g., RSA, AES) to secure communications.

Public-key cryptography is implemented to exchange keys securely.

# Workflow/Architecture

## **Step-by-Step Process:**

* + 1. Signaling: The process of exchanging signaling messages to establish a connection between peers.
    - Offer-Answer model: One peer creates an offer which is sent to the other peer. The receiving peer responds with an answer, and the connection is established.
    - Ice Candidates: Peer-to-peer connection relies on ICE (Interactive Connectivity Establishment) framework, which handles NAT traversal.
    1. **Peer Connections:**
    - WebRTC: A peer-to-peer connection is established using WebRTC, which enables low-latency, high-quality data streaming.
    - DataChannel: Used for real-time data exchange (messages, files, voice).
    - Connection Status: Visual indicators show the connection status of peers **(connected or disconnected).**
    1. **Data Transmission:**

Encrypted Messages: All messages are encrypted using the public-key cryptography method.

File Sharing: Files are encoded into JSON objects and transmitted over the data channel.

Voice Messaging: Voice messages are recorded as audio files, converted to base64 for transmission, and decoded on the receiving end.

* 1. **User Interface:**
     1. Chat Interface: Simple, user-friendly design to facilitate text and multimedia communication.
     2. Active Users Display: A sidebar displays a list of active users along with their statuses.
  2. **Error Handling:**
     1. Implemented mechanisms for dealing with connectivity issues, signal loss, and unexpected disconnections.

# Screenshots or Outputs

## **Visual Representation of the Project:**

### **Chat Interface:**

Screenshots showing how the chat interface looks and functions. Include messages, emojis, and file attachments.

Demonstrate the message input, send button state, and encryption indicators.

### **File Sharing:**

Screenshots showing file transfer in action, including uploading, downloading, and viewing received files.

Display file sizes and types for easy identification.

### **Voice Messaging:**

Demonstrate voice message recording and playback. Show audio controls and how users can send voice messages.

### **Active Users:**

Screenshot of the active users list, highlighting user statuses and connection indicators.

### **Encryption:**

Visuals or confirmation messages indicating that messages or files are being encrypted.

Example of how decrypted messages are displayed to users.

# Challenges and Solutions

## **Briefly Describe Hurdles:**

* + 1. Connectivity Issues: Handling NAT traversal and ensuring stable peer-to-peer connections.
    2. Encryption and Security: Ensuring secure transmission of data and handling encryption keys properly.
    3. User Interface Design: Designing an intuitive interface that supports text, files, and voice messages.
    4. Error Handling: Dealing with unexpected issues like dropped connections, data corruption, and user feedback.

## **Solutions:**

* + 1. Connectivity: Implementing STUN and TURN servers to facilitate NAT traversal.
    2. Encryption: Using strong encryption standards and handling key exchanges through WebRTC DataChannels.
    3. UI Design: User testing and iterative development to refine the chat interface and usability.
    4. Error Handling: Implemented retry mechanisms, status indicators, and alerts for users to notify them of connection issues.

# Conclusion and Future Work

## **Summary of Results:**

* + 1. Successes: Achieved key project objectives such as establishing secure connections, implementing end-to-end encryption, and creating a responsive UI.
    2. Lessons Learned: Gained experience in handling WebRTC, encryption, error handling, and user interface design.

## **Potential Improvements:**

* + 1. Scalability: Improve the performance and scalability of the application to handle more users and larger file sizes.
    2. Advanced Features: Introduce additional features such as video conferencing, enhanced multimedia sharing, and integration with third-party services.
    3. User Feedback: Collect user feedback and make UI/UX adjustments based on user experience.
    4. Security: Enhance encryption protocols and implement additional security measures to protect user data.

## **Next Steps:**

### Develop video calling functionality.

### Implement better user management.

### Enhance error handling and recovery mechanisms.

### Conduct user testing:

to get real-world feedback and improve the application