EXPERIMENT NO

Aim- Enable real time communication via Websockets

Theory-

1. What is a WebSocket?

- **Definition**: A WebSocket is a protocol (built on top of TCP) that provides full-duplex communication channels over a single, long-lived connection between a client (e.g., browser) and a server. Unlike HTTP, which is request-response based, WebSockets allow both parties to send data independently at any time.
- **Protocol**: Initiated with an HTTP handshake (using the Upgrade header to switch from HTTP to WebSocket), it uses the ws:// (or wss:// for secure connections) scheme.
- **Relevance to Your Project**: Your chat application uses ws://localhost:8081 to enable real-time message exchange between multiple clients via a Node.js server.

2. Client-Server Architecture

- **Components**:
  - **Client**: The index.html file, loaded in the browser, contains HTML for the UI and JavaScript to establish a WebSocket connection, send messages, and handle incoming data.
  - **Server**: The server.js file, running with Node.js and the ws library, listens for connections, processes messages, and broadcasts them to all connected clients.
- **Interaction**:
  - The client initiates a WebSocket connection to ws://localhost:8081.
  - When you click "Send," the client sends a JSON payload (e.g., { type: 'message', username: 'User795', message: 'hello' }).
  - The server receives this, parses it, and broadcasts it to all clients, including a confirmation flag for the sender.
- **Issue Context**: The "Cannot send: Check connection or input" error suggests a disconnect or state mismatch between client and server, possibly due to the WebSocket closing unexpectedly or a send failure.

3. Real-Time Communication

- **Mechanism**: WebSockets eliminate the need for polling (repeated HTTP requests) by maintaining an open connection. This allows instant message delivery, ideal for chat applications.
- **Events**:
  - onopen: Triggered when the connection is established (e.g., "Connected as User795").
  - onmessage: Handles incoming messages (e.g., broadcasts or typing indicators).

    - onclose: Fired when the connection closes (e.g., "Disconnected from server").
    - onerror: Catches errors (e.g., network issues).
- **Your Implementation**: The chat uses onmessage to display messages and join notifications, with sendMessage triggering the send event. The confirmation ("Hello Khushi which is send successfully") relies on the server echoing back the message with a confirmation flag.

4. How Messages Are Processed

- **Client-Side**:
    - The sendMessage function checks ws.readyState === WebSocket.OPEN (state 1) before sending.
    - It stringifies a JSON object and sends it via ws.send.
    - Immediate feedback is added locally, with server confirmation triggering the custom message.
- **Server-Side**:
    - The wss.on('connection') handler listens for new clients.
    - ws.on('message') parses the incoming JSON and broadcasts it using wss.clients.forEach.
    - The confirmation flag is set only for the sender, enabling the client to display the success message.
- **Potential Failure Point**: If the server doesn't receive the message (e.g., due to port issues or parsing errors), the client's send might fail silently, leading to the "Cannot send" notification.

5. Debugging Theory

- **State Management**: The WebSocket readyState (0: connecting, 1: open, 2: closing, 3: closed) is critical. A state change to 3 after opening could explain the issue.
- **Event Handling**: The Send button's onclick and addEventListener must fire correctly. DOM readiness (window.onload) ensures this.
- **Logging**: Console logs (e.g., "Send button clicked", "Received raw message") help trace the flow. Absence of server logs suggests a network or port problem.
- **Error Handling**: try/catch blocks catch JSON parsing or send errors, but network issues (e.g., firewall) might not be caught this way.

6. Why the Send Button Might Not Work

- **Connection Drop**: The WebSocket might close after onopen due to network instability or server restart.
- **Port Conflict**: Port 8081 might be blocked or used by another process (e.g., your earlier java.exe on 8080 could indicate port management issues).
- **Event Listener Failure**: If the button's event isn't attaching, the sendMessage function won't trigger.
- **Server Parsing Error**: Malformed JSON or a server crash could prevent broadcasting.

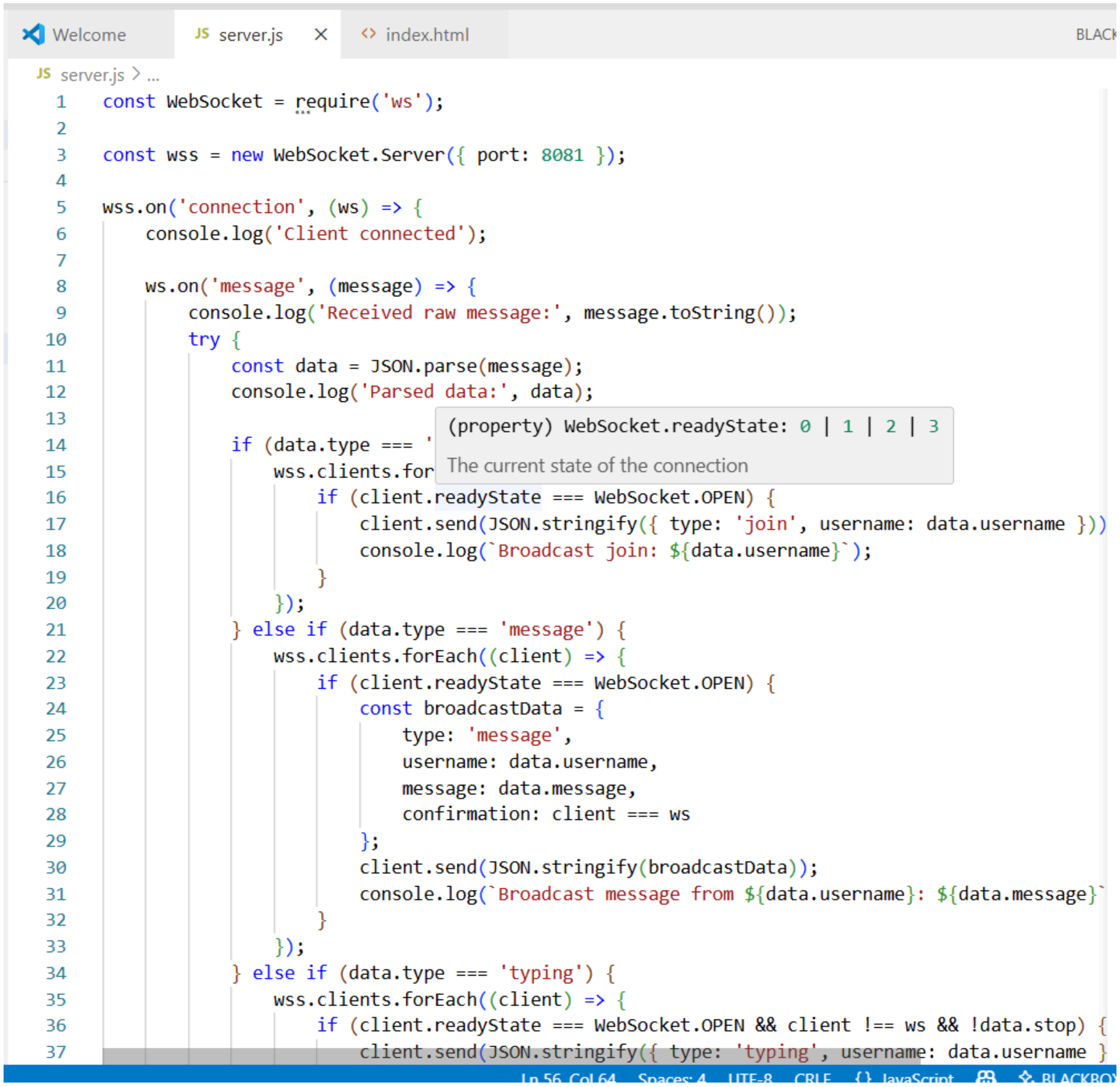
7. Confirmation Mechanism Theory

- **Design**: The server includes a confirmation: true flag in the broadcast for the sender, which the client checks to display "Hello Khushi which is send successfully".
- **Purpose**: Provides user feedback, enhancing the professional feel of the chat.
- **Challenge**: Requires a successful round-trip, so if the send fails, no confirmation appears—aligning with your current "Cannot send" issue.

8. Theoretical Solution Approach

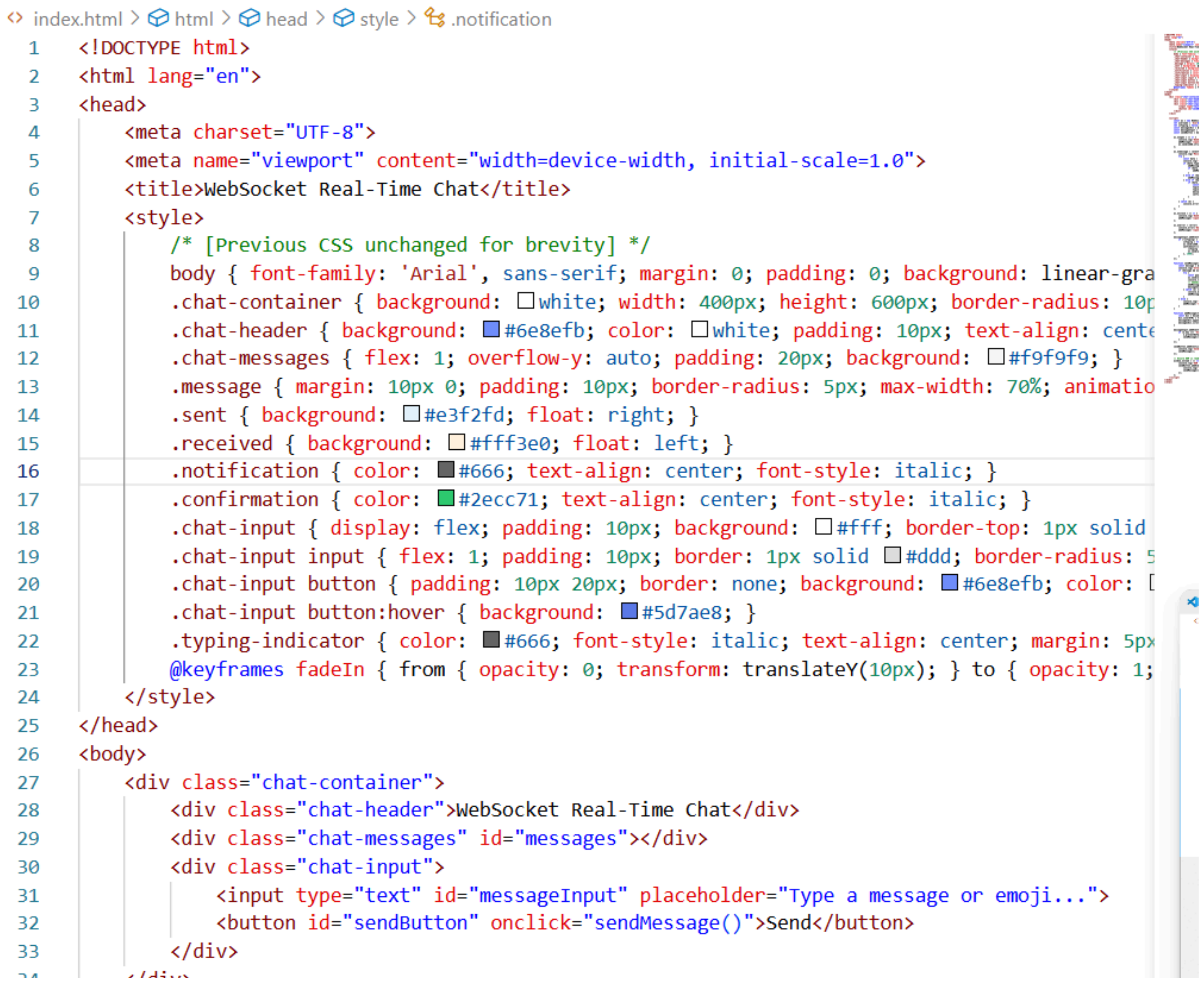
- **Verify Connection**: Ensure ws.readyState remains 1 throughout the session.
- **Check Port**: Confirm 8081 is free (netstat -aon | findstr :8081) and accessible.
- **Test Basic Send**: Use plain text to bypass JSON parsing and confirm connectivity.
- **Enhance Feedback**: The immediate local message plus server confirmation balances real-time response with reliability.

Source code-



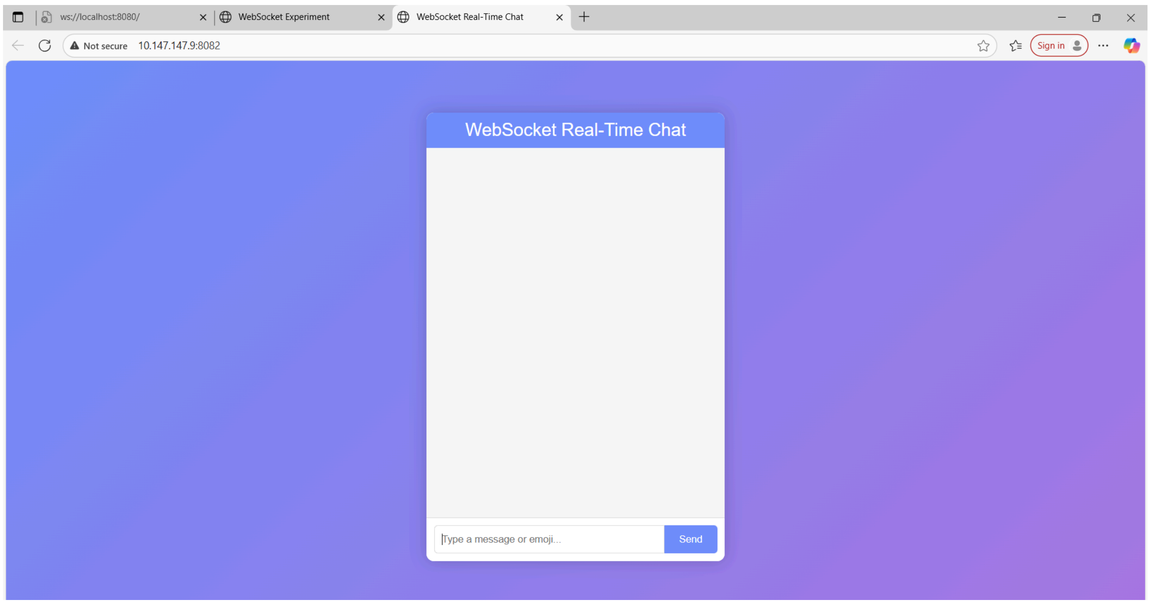
```javascript
const WebSocket = require('ws');

const wss = new WebSocket.Server({ port: 8081 });

wss.on('connection', (ws) => {
    console.log('Client connected');

    ws.on('message', (message) => {
        console.log('Received raw message:', message.toString());
        try {
            const data = JSON.parse(message);
            console.log('Parsed data:', data);

            if (data.type === '
                wss.clients.for
                    if (client.readyState === WebSocket.OPEN) {
                        client.send(JSON.stringify({ type: 'join', username: data.username }))
                        console.log(`Broadcast join: ${data.username}`);
                    }
                });
            } else if (data.type === 'message') {
                wss.clients.forEach((client) => {
                    if (client.readyState === WebSocket.OPEN) {
                        const broadcastData = {
                            type: 'message',
                            username: data.username,
                            message: data.message,
                            confirmation: client === ws
                        };
                        client.send(JSON.stringify(broadcastData));
                        console.log(`Broadcast message from ${data.username}: ${data.message}`
                    }
                });
            } else if (data.type === 'typing') {
                wss.clients.forEach((client) => {
                    if (client.readyState === WebSocket.OPEN && client !== ws && !data.stop) {
                        client.send(JSON.stringify({ type: 'typing', username: data.username }
```

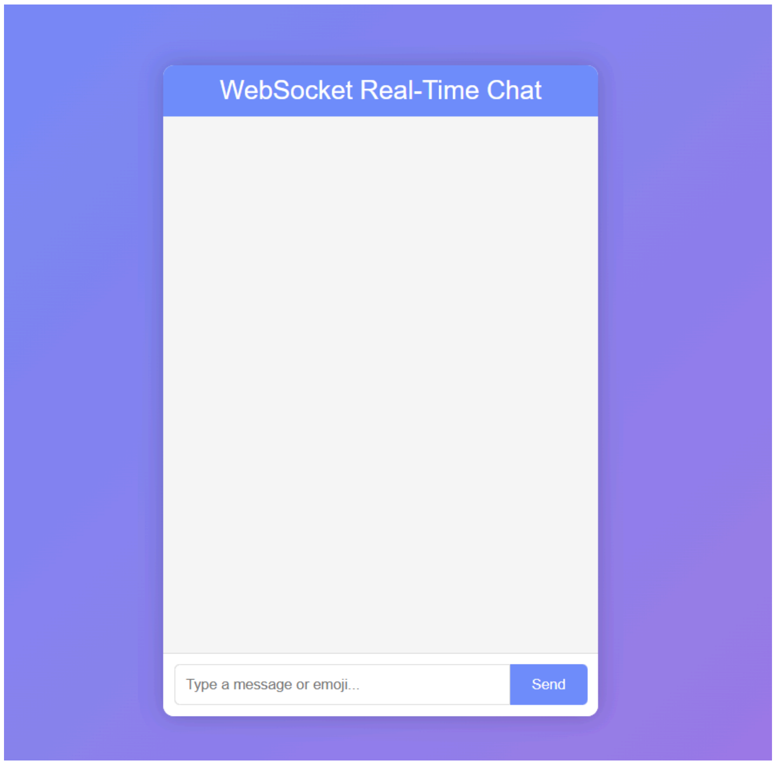
```
(property) WebSocket.readyState: 0 | 1 | 2 | 3
The current state of the connection
```

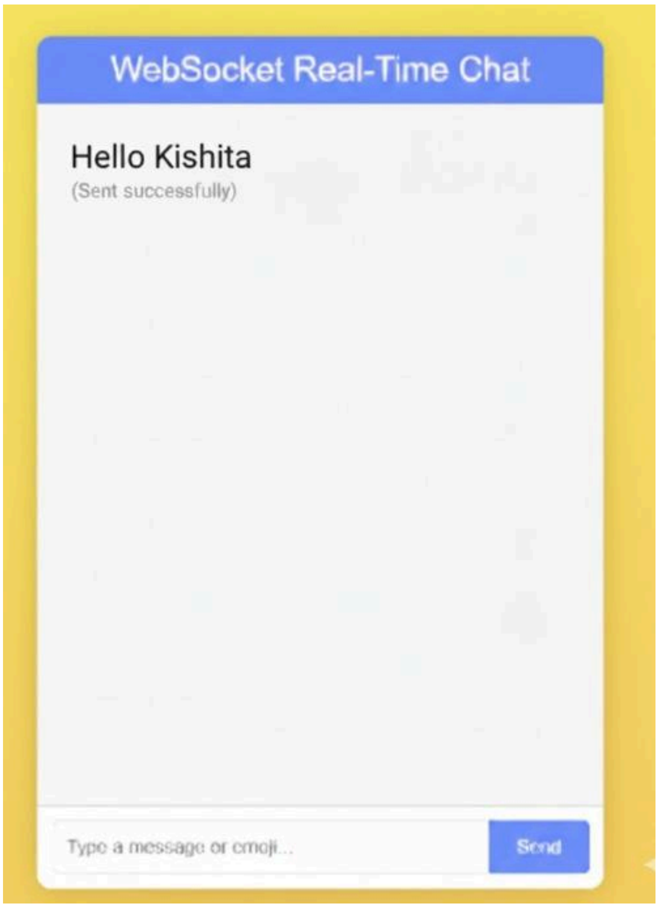


index.html > html > head > style > .notification

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>WebSocket Real-Time Chat</title>
    <style>
        /* [Previous CSS unchanged for brevity] */
        body { font-family: 'Arial', sans-serif; margin: 0; padding: 0; background: linear-gra
        .chat-container { background: white; width: 400px; height: 600px; border-radius: 10p
        .chat-header { background: #6e8efb; color: white; padding: 10px; text-align: cente
        .chat-messages { flex: 1; overflow-y: auto; padding: 20px; background: #f9f9f9; }
        .message { margin: 10px 0; padding: 10px; border-radius: 5px; max-width: 70%; animatio
        .sent { background: #e3f2fd; float: right; }
        .received { background: #fff3e0; float: left; }
        .notification { color: #666; text-align: center; font-style: italic; }
        .confirmation { color: #2ecc71; text-align: center; font-style: italic; }
        .chat-input { display: flex; padding: 10px; background: #fff; border-top: 1px solid
        .chat-input input { flex: 1; padding: 10px; border: 1px solid #ddd; border-radius: 5
        .chat-input button { padding: 10px 20px; border: none; background: #6e8efb; color:
        .chat-input button:hover { background: #5d7ae8; }
        .typing-indicator { color: #666; font-style: italic; text-align: center; margin: 5px
        @keyframes fadeIn { from { opacity: 0; transform: translateY(10px); } to { opacity: 1;
    </style>
</head>
<body>
    <div class="chat-container">
        <div class="chat-header">WebSocket Real-Time Chat</div>
        <div class="chat-messages" id="messages"></div>
        <div class="chat-input">
            <input type="text" id="messageInput" placeholder="Type a message or emoji...">
            <button id="sendButton" onclick="sendMessage()">Send</button>
        </div>
```







Hello Kishita
(Sent successfully)

Conclusion

The theory underscores that your WebSocket chat relies on a persistent, bidirectional connection, with the Send button acting as the user interface to initiate this process. The confirmation feature adds value but depends on successful communication. The current issue likely stems from a post-connection failure (e.g., state change or port issue), which logs will clarify. Please test with the updated code and provide the console outputs, and we'll apply this theory to fix it—rest if needed, as it's late (12:43 AM IST)!