|  |  |
| --- | --- |
| NAME | Aman kumar |
| UID | 23BCS11038 |

* Full Stack PRACTICE 7.3
* CODE

server.js :::

// realtime-chat-backend/server.js

const express = require('express');

const http = require('http');

const { Server } = require('socket.io');

const cors = require('cors');

const app = express();

// Enable CORS for Express (if needed for REST routes, but mainly for Socket.io)

app.use(cors({

    origin: "http://localhost:3000", // Allow connection from React app

    methods: ["GET", "POST"]

}));

// 1. Create an HTTP server instance using the Express app

const server = http.createServer(app);

// 2. Initialize Socket.io and attach it to the HTTP server

const io = new Server(server, {

    cors: {

        origin: "http://localhost:3000",

        methods: ["GET", "POST"]

    }

});

const PORT = 5000;

// Store connections for logging (optional)

let userCount = 0;

// 3. Socket.io connection handler

io.on('connection', (socket) => {

    userCount++;

    console.log(`User connected: ${socket.id}. Total users: ${userCount}`);

    // --- Message Broadcasting Event ---

    // Listen for 'sendMessage' event from any client

    socket.on('sendMessage', (messageData) => {

        // Log the message on the server

        console.log(`New message from ${messageData.user}: ${messageData.text}`);

        // Broadcast the message to ALL connected clients (including the sender)

        io.emit('receiveMessage', messageData);

    });

    // --- Disconnect Event ---

    socket.on('disconnect', () => {

        userCount--;

        console.log(`User disconnected: ${socket.id}. Total users: ${userCount}`);

    });

});

// Start the server

server.listen(PORT, () => {

    console.log(`Socket.io server listening on http://localhost:${PORT}`);

});

App.js ::::

// realtime-chat-frontend/src/App.js

import React, { useState, useEffect, useRef } from 'react';

import io from 'socket.io-client';

import './App.css'; // For basic styling

// 1. Initialize the socket connection outside the component

// so it doesn't reconnect on every render (or use useMemo/useRef inside)

const SOCKET\_SERVER\_URL = 'http://localhost:5000';

const socket = io(SOCKET\_SERVER\_URL);

// Helper to format time

const getTime = () => {

    const now = new Date();

    return `${now.getHours().toString().padStart(2, '0')}:${now.getMinutes().toString().padStart(2, '0')}:${now.getSeconds().toString().padStart(2, '0')}`;

};

function App() {

    const [username, setUsername] = useState('');

    const [message, setMessage] = useState('');

    const [chatLog, setChatLog] = useState([]);

    // Ref for auto-scrolling the chat window

    const messagesEndRef = useRef(null);

    // Effect for connecting and listening to socket events

    useEffect(() => {

        // Auto-scroll whenever chatLog updates

        messagesEndRef.current?.scrollIntoView({ behavior: "smooth" });

    }, [chatLog]);

    useEffect(() => {

        console.log('Attempting to connect to socket...');

        // 2. Listen for 'receiveMessage' event from the server

        socket.on('receiveMessage', (data) => {

            // Add the new message to the chat log

            setChatLog((prevLog) => [...prevLog, data]);

        });

        // Clean up on component unmount

        return () => {

            socket.off('receiveMessage');

        };

    }, []); // Run only once on mount

    const sendMessage = (e) => {

        e.preventDefault();

        if (!username || !message) return;

        const messageData = {

            user: username,

            text: message,

            time: getTime(),

        };

        // 3. Emit the 'sendMessage' event to the server

        socket.emit('sendMessage', messageData);

        // Clear the input field

        setMessage('');

    };

    return (

        <div className="chat-container">

            <div className="chat-box">

                <h1 className="title">Real-Time Chat</h1>

                {/\* Username Input \*/}

                <input

                    type="text"

                    placeholder="Enter your name (e.g., Alice)"

                    value={username}

                    onChange={(e) => setUsername(e.target.value)}

                    className="username-input"

                />

                {/\* Chat Log Display \*/}

                <div className="message-list">

                    {chatLog.map((msg, index) => (

                        <div key={index} className="message-item">

                            <span className="user-time">

                                <strong>{msg.user} [{msg.time}]:</strong>

                            </span>

                            <span className="message-text"> {msg.text}</span>

                        </div>

                    ))}

                    <div ref={messagesEndRef} /> {/\* For auto-scrolling \*/}

                </div>

                {/\* Message Input and Send Button \*/}

                <form onSubmit={sendMessage} className="input-area">

                    <input

                        type="text"

                        placeholder="Type your message..."

                        value={message}

                        onChange={(e) => setMessage(e.target.value)}

                        disabled={!username}

                    />

                    <button type="submit" disabled={!username || !message}>

                        Send

                    </button>

                </form>

            </div>

        </div>

    );

}

export default App;

App.css :::

/\* realtime-chat-frontend/src/App.css \*/

body {

    background-color: #f0f0f0;

    font-family: Arial, sans-serif;

    display: flex;

    justify-content: center;

    align-items: center;

    min-height: 100vh;

    margin: 0;

}

.chat-container {

    padding: 20px;

}

.chat-box {

    width: 400px;

    height: 550px;

    background-color: #fff;

    border: 1px solid #ccc;

    border-radius: 8px;

    box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);

    display: flex;

    flex-direction: column;

    padding: 15px;

}

.title {

    text-align: center;

    margin-bottom: 10px;

    font-size: 1.5em;

}

.username-input {

    padding: 8px;

    margin-bottom: 10px;

    border: 1px solid #ccc;

    border-radius: 4px;

    width: 100%;

    box-sizing: border-box;

}

.message-list {

    flex-grow: 1;

    border: 1px solid #ddd;

    padding: 10px;

    overflow-y: auto;

    margin-bottom: 10px;

    background-color: #f9f9f9;

}

.message-item {

    margin-bottom: 5px;

    line-height: 1.4;

    word-wrap: break-word;

}

.user-time strong {

    font-weight: bold;

}

.input-area {

    display: flex;

}

.input-area input[type="text"] {

    flex-grow: 1;

    padding: 10px;

    border: 1px solid #ccc;

    border-radius: 4px 0 0 4px;

    box-sizing: border-box;

}

.input-area button {

    padding: 10px 15px;

    background-color: #007bff;

    color: white;

    border: none;

    border-radius: 0 4px 4px 0;

    cursor: pointer;

    transition: background-color 0.2s;

}

.input-area button:hover:not(:disabled) {

    background-color: #0056b3;

}

.input-area button:disabled {

    background-color: #ccc;

    cursor: not-allowed;

}

Screens screenshot of a chat

AI-generated content may be incorrect.OUTPUT: