Practical No 8

Aim : Enable **real-time communication** via WebSockets

Code :

Socket.io initialization :

```
// --- Server Setup ---
const app = express();
const server = http.createServer(app);

// Initialize Socket.IO and allow all origins for easy local testing
const io = new Server(server, {
  cors: {
    origin: "*",
    methods: ["GET", "POST"]
  }
});

const PORT = 3000;
let userCount = 0; // Simple counter to assign a generic ID to clients
```

Socket logic

```
// --- Socket.IO Real-Time Logic ---
io.on('connection', (socket) => {
  userCount++;
  const userId = `User-${userCount}`;

  console.log(`[CONNECTED] ${userId} (${socket.id})`);

  // 1. Notify the connecting client of their ID and current user count
  socket.emit('status update', {
    status: 'Connected',
    userId: userId,
    userCount: io.engine.clientsCount
  });

  // 2. Broadcast connection event to all others
  socket.broadcast.emit('chat message', {
    id: 'System',
    message: `${userId} has joined the chat.`,
    timestamp: new Date().toLocaleTimeString()
  });

  // 3. Handle incoming chat messages
  socket.on('chat message', (msg) => {
    const messageData = {
      id: msg.id || userId,
      message: msg.text,
      timestamp: new Date().toLocaleTimeString()
    };

    // Broadcast the message to ALL connected clients, including the sender
    io.emit('chat message', messageData);
    console.log(`[MESSAGE] ${messageData.id}: ${messageData.message}`);
  });
```

Handle Client Disconnection

```
// 4. Handle client disconnection
socket.on('disconnect', () => {
  userCount--; // This simple counter might undercount slightly on rapid reconnects, but works for demonstration
  console.log(`[DISCONNECTED] ${userId} (${socket.id})`);

  // Broadcast disconnection event to all others
  io.emit('chat message', {
    id: 'System',
    message: `${userId} has left the chat.`,
    timestamp: new Date().toLocaleTimeString()
  });

  // Send updated user count to everyone
  io.emit('status update', {
      status: 'Updated',
      userCount: io.engine.clientsCount
  });
 });
});
```

Form Submission Handler :

```
// 1. Handle form submission (sending a message)
form.addEventListener('submit', function(e) {
    e.preventDefault();
    const text = input.value.trim();
    if (text) {
        // Emit the 'chat message' event to the server
        socket.emit('chat message', { id: myUserId, text: text });
        input.value = ''; // Clear the input field
        input.focus();
    }
});
```

Chat Message Listener :

```
// 2. Listen for 'chat message' event from the server
socket.on('chat message', function(msg) {
    // Render the message received from the server (either broadcast or self-echo)
    renderMessage(msg.id, msg.message, msg.timestamp);
});

// 3. Listen for connection status updates
socket.on('status update', function(data) {
    if (data.status === 'Connected' || data.status === 'Updated') {
        statusDot.classList.remove('bg-yellow-400', 'bg-red-500');
        statusDot.classList.add('bg-green-500');
        statusText.textContent = 'Connected';
        sendButton.disabled = false;

        if (data.userId) {
            myUserId = data.userId;
            userInfo.textContent = `Your ID: ${myUserId}`;
        }
        if ( const userCountElement: HTMLElement | null
            userCountElement.textContent = `Active Users: ${data.userCount}`;
        }
    }
});
```

Connection and Disconnection Logic :

```
// 4. Handle initial connection
socket.on('connect', function() {
    // Note: Status update will come from the server, but this sets the initial state
    statusDot.classList.remove('bg-red-500');
    statusDot.classList.add('bg-yellow-400');
    statusText.textContent = 'Connecting...';
    sendButton.disabled = true;
});

// 5. Handle disconnection
socket.on('disconnect', function() {
    statusDot.classList.remove('bg-green-500', 'bg-yellow-400');
    statusDot.classList.add('bg-red-500');
    statusText.textContent = 'Disconnected';
    sendButton.disabled = true;
    renderMessage('System', 'Lost connection to the server. Attempting to reconnect...', new Date().toLocale
    userCountElement.textContent = 'Active Users: 0';
});
```

Output :

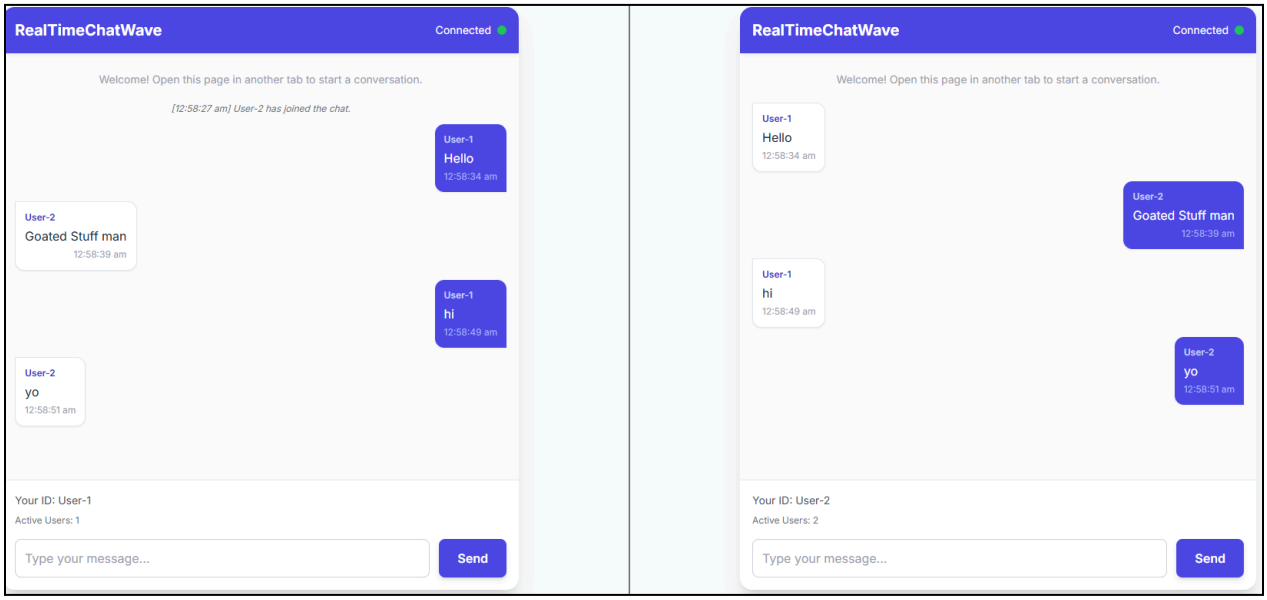

Figure : 8.1 - User1 and User2 Chatting

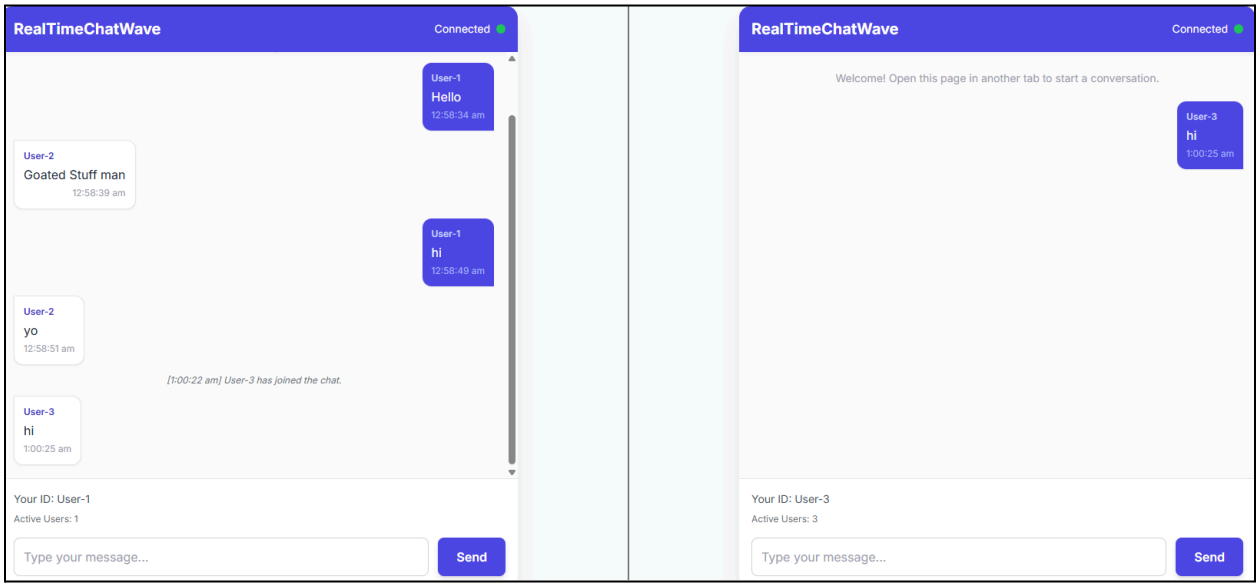

Figure 8.2 - User3 Joins

Figure 8.3 - Backend Console Output

Conclusion : Thus we have used Socket.io to create usable websocket connections and added live chatting functionality.