

**Digital Empowerment Pakistan**

**TASK#04**

**Submitted by:**

**MUBASHRA MOBEEN MALIK**

**WEB DEVELPOMENT INTERNSHIP**

**TASK:**

Implement real-time features like chat or notifications using

WebSockets or a similar technology in a web application.

**CODE:**

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>WebSocket Chat</title>

    <style>

        body {

            font-family: Arial, sans-serif;

            display: flex;

            justify-content: center;

            align-items: center;

            height: 100vh;

            background-color: #b52121;

        }

        #chat-container {

            width: 400px;

            border: 1px solid #160b0b;

            border-radius: 5px;

            background-color: #fff;

            padding: 20px;

            box-shadow: 0 0 10px rgba(0,0,0,0.1);

        }

        #messages {

            height: 200px;

            overflow-y: scroll;

            border: 1px solid #600808;

            padding: 10px;

            margin-bottom: 10px;

            border-radius: 5px;

        }

        #messages div {

            margin-bottom: 5px;

        }

        #input-container {

            display: flex;

        }

        #message {

            flex: 1;

            padding: 10px;

            border: 1px solid #ccc;

            border-radius: 5px;

            margin-right: 10px;

        }

        #send {

            padding: 10px 20px;

            background-color: #007bff;

            color: white;

            border: none;

            border-radius: 5px;

            cursor: pointer;

        }

        #send:hover {

            background-color: #0056b3;

        }

    </style>

</head>

<body>

<div id="chat-container">

    <div id="messages"></div>

    <div id="input-container">

        <input type="text" id="message" placeholder="Type a message..." />

        <button id="send">Send</button>

    </div>

</div>

<script>

    const ws = new WebSocket('ws://localhost:8080');

    const messages = document.getElementById('messages');

    const messageInput = document.getElementById('message');

    const sendButton = document.getElementById('send');

    ws.onopen = () => {

        console.log('Connected to the server');

    };

    ws.onmessage = (event) => {

        const message = document.createElement('div');

        message.textContent = event.data;

        messages.appendChild(message);

        messages.scrollTop = messages.scrollHeight;

    };

    sendButton.addEventListener('click', () => {

        const message = messageInput.value;

        if (message) {

            ws.send(message);

            messageInput.value = '';

        }

    });

    messageInput.addEventListener('keypress', (event) => {

        if (event.key === 'Enter') {

            sendButton.click();

        }

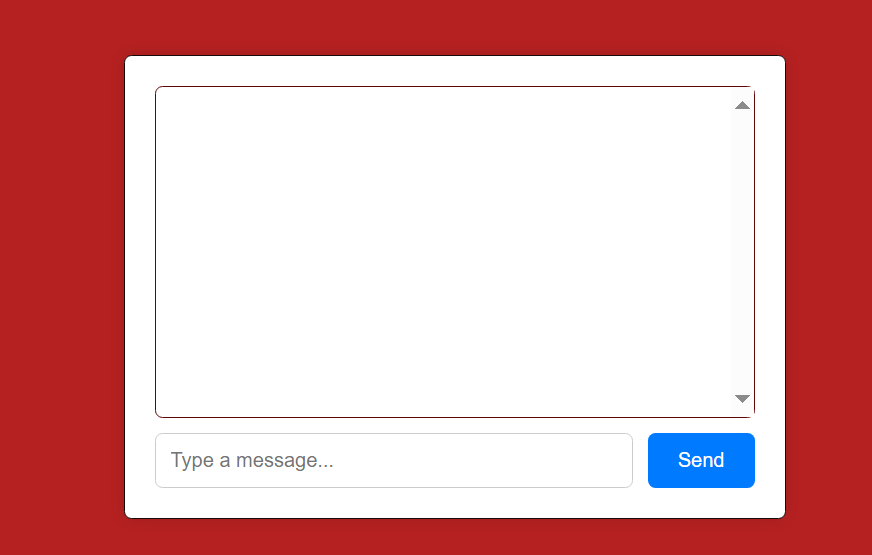
    });

</script>

</body>

</html>

**Output:**

****

**Explanation:**

This code demonstrates a simple real-time chat application using WebSockets, HTML, internal CSS, and JavaScript. The HTML structure includes a chat container with an input field and a send button, designed to create a user-friendly chat interface. Internal CSS styles the chat elements, ensuring a visually appealing layout with centered alignment, a fixed-width chat box, and aesthetic borders and paddings. The JavaScript initializes a WebSocket connection to the server at ws://localhost:8080 and includes event handlers to manage WebSocket communication. When a connection is established, a message is logged to the console. Incoming messages are dynamically added to the messages container, and the container scrolls to show the latest messages. The send button and input field are configured to send messages to the server and clear the input field upon sending. Additionally, pressing the Enter key in the input field triggers the send action, enhancing user experience. This setup provides a straightforward yet functional chat application, illustrating the core principles of real-time communication with WebSockets.

**Conclusion:**

This example demonstrates a simple real-time chat application using WebSockets. The HTML provides the structure, internal CSS styles the components, and JavaScript manages the WebSocket connection and user interactions. The application allows users to send and receive messages in real time, with a clean and responsive interface. This setup is suitable for basic chat functionality and can be extended with additional features like user authentication, message history, and more advanced styling.