Lab: Build a Chat App with Node.js and Socket.io

Objectives

By the end of this lab, you will be able to:

1. Create a Node.js project.
2. Install and use **npm** packages (**express, socket.io**)
3. Create a simple HTTP server using Express.
4. Implement real-time messaging using Socket.io.
5. Store chat messages in a local JSON file.
6. Serve an HTML page that displays and sends chat messages in real-time.

Prerequisites

- Node.js and npm installed.
- Basic knowledge of JavaScript and HTML.
- Terminal/Command Prompt access.

What the app does

You'll build a live chat app where multiple people can send and receive messages instantly with no page refresh. To do that, you'll need a way for the browser and server to keep a live connection open using sockets. Normally, a web page just sends a request and waits for a response Sockets are more like a phone call where both sides stay connected and can talk at any time.
Socket.io makes it easy to build that phone call between your server and all the connected browsers.

Using your JavaScript app, the **server** listens for new messages and broadcasts them to everyone. You'll use a **file** to store messages so they don't disappear when you restart the server.

Component parts of your app

Create the socket server

```
const app = express();                    // this is your website
const server = http.createServer(app);    // server is the actual HTTP server
const io = new Server(server);            // io is where all the socket magic happens!
```

When someone connects (runs every time a new browser opens the chat page**)**

```
io.on('connection', (socket) => {
  console.log('User connected:', socket.id);
});
```

Each user gets their own socket object, like their own phone line.

Sending and receiving messages

```
socket.on('chat message', (data) => {
  io.emit('chat message', data);
});
```

When someone sends a message, the server gets it (**on**), and then broadcasts it to everyone (**emit**). Think of the server as a radio station, it receives messages and transmits them to all listeners.

Step 1: Create a project folder

mkdir chatApp
cd chatApp

Step 2: Initialise Node.js project

npm init -y

- This creates a package.json file with default settings (**-y** so *yes* is replied for each question)

Step 3: Install required modules

npm install express socket.io

- express → HTTP server
- socket.io → Real-time WebSocket communication

Step 4: Create project structure

Inside chatApp:

chatApp/
├ **server.js**
├ **messages.json** (will be created automatically)
└ **public/**
  └ **index.html**

- Create a public folder for static HTML files.

Step 5: Create server.js

Paste the following code:

```
import express from 'express';
import http from 'http';
import { Server } from 'socket.io';
import fs from 'fs';
import { fileURLToPath } from 'url';
import { dirname, join } from 'path';

const __filename = fileURLToPath(import.meta.url);
const __dirname = dirname(__filename);

const app = express();
const server = http.createServer(app);
const io = new Server(server);

const PORT = 3000;
const MESSAGES_FILE = join(__dirname, 'messages.json');

// Load messages from file or empty array
let messages = [];
if (fs.existsSync(MESSAGES_FILE)) {
```

```
    messages = JSON.parse(fs.readFileSync(MESSAGES_FILE));
}

app.use(express.static(join(__dirname, 'public')));

io.on('connection', (socket) => {
    console.log('User connected:', socket.id);
    socket.emit('load messages', messages);

    socket.on('chat message', (data) => {
        const msg = { username: data.username || 'Anonymous', message: data.message,
                      timestamp: new Date().toISOString() };
        messages.push(msg);
        fs.writeFileSync(MESSAGES_FILE, JSON.stringify(messages, null, 2));
        io.emit('chat message', msg);
    });

    socket.on('disconnect', () => console.log('User disconnected'));
});

server.listen(PORT, () => console.log(`Server running on http://localhost:${PORT}`));
```

Step 6: Create public/index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>File-based Chat App</title>
    <style>
        body {
            font-family: Arial, sans-serif;
        }

        #messages {
            list-style: none;
            padding: 0;
            max-height: 300px;
            overflow-y: auto;
            border: 1px solid #ccc;
            margin-bottom: 10px;
        }

        li {
            padding: 5px 10px;
        }

        input {
            margin: 5px;
        }
    </style>
</head>
<body>
    <h2>Chat</h2>
    <ul id="messages"></ul>
    <input id="username" placeholder="Your name" />
    <input id="message" placeholder="Type a message" autocomplete="off" />
    <button id="send">Send</button>
```

```
    <script src="/socket.io/socket.io.js"></script>
    <script>
        const socket = io();

        const messagesList = document.getElementById('messages');
        const usernameInput = document.getElementById('username');
        const messageInput = document.getElementById('message');
        const sendBtn = document.getElementById('send');

        socket.on('load messages', (msgs) => { messagesList.innerHTML = '';
                                               msgs.forEach(addMessage); });
        socket.on('chat message', addMessage);

        sendBtn.addEventListener('click', () => {
            const msg = { username: usernameInput.value || 'Anonymous', message:
                                  messageInput.value };
            if (msg.message.trim() !== '') { socket.emit('chat message', msg);
                                               messageInput.value = ''; }
        });

        function addMessage(msg) {
            const li = document.createElement('li');
            const time = new Date(msg.timestamp).toLocaleTimeString();
            li.textContent = `[${time}] ${msg.username}: ${msg.message}`;
            messagesList.appendChild(li);
            messagesList.scrollTop = messagesList.scrollHeight;
        }
    </script>
</body>
</html>
```

Step 7: Run the server

node server.js

- Open browser: **http://localhost:3000/**
- Enter a name and message → click **Send**
- Messages are stored in **messages.json** and update **in real-time** for all clients

Step 8: Go Global (optional!)

- Save you app on GitHub
- Sign in or make a free account on https://onrender.com
- Create a **Static website** type of project
- Grant onrender.com access to you public GitHub sites
- Select you chat app GitHub URI from the list
- Take a note of your site's address on onrender.com
- That's it!