**Real-time Chat app**

**Project setup**

* Use npm init in root directory to create package.json file
* Install following dependencies using npm install
  + Express (web framework)
  + Socket.io (framework for dealing with web sockets)
  + Moment (moment.js used to format dates/times)
  + Use npm install -D nodemon to install nodemon as a dev dependency.
* Under scripts edit the start portion to say “ node server.js”

"scripts": {

    "start": "node server.js",

    "dev": "nodemon server.js"

  },

**Creating server with Express**

* Start with server.js file:

const express = require('express');

const app = express();

const PORT = 3000 || process.env.PORT;

server.listen(PORT, () => console.log(`Server running on port ${PORT}`));

* Can now use command “npm run dev” to start the server.

**Set static folder for localhost:3000**

* By setting a static folder, when you navigate to localhost:3000 in your browser, it will open the index.html file in that folder.
* <https://nodejs.dev/learn/the-nodejs-path-module>
* looks for public folder to open index.html from

const path = require('path');

*// Set static folder*

app.use(express.static(path.join(\_\_dirname, 'public')));

**Setting up socket.io**

* <https://nodejs.dev/learn/build-an-http-server>
* With express there is a method called createServer but we need to access it directly to use socket.io
* The http.createServer() method turns your computer into an HTTP server. <https://www.w3schools.com/nodejs/met_http_createserver.asp>

const http = require('http');

const server = http.createServer(app);

*// Can now change app.listen to server.listen*

server.listen(PORT, () => console.log(`Server running on port ${PORT}`));

* In order to run socket.io when a client connects:

const socketio = require('socket.io')

const io = socketio(server);

*// Run socket.io when client connects*

*// .on will listen to some kind of event*

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

    console.log('New web socket connection', socket);

})

* <https://socket.io/docs/v3/client-socket-instance/>
* Back in chat.html

<script type="text/javascript" src="/socket.io/socket.io.js"></script>

    <script src="js/main.js"></script>

* This is done so that the two scripts can interact with each other on the chat.html page

**Setting up client-side**

* In main.js

*// We have access to io() here from the server.js file because they're both in chat.html*

const socket = io();

* Back in server.js

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

    console.log('New web socket connection');

    socket.emit('message', 'Welcome to the chatroom');

})

* Socket.emit will send this message ‘Welcome…’ so that the client-side can grab it.
  + message’ here can be called whatever you want
* Back in main.js

socket.on('message', message => {

    console.log(message)

})

* When the ‘message’ event is received by the socket, we have access to the payload, i.e. ‘Welcome to the chatroom’ on the client-side.
* As a result the message “welcome …” will be logged to the browser console now.

Overview of events so far

1. Refresh page in browser
2. This fires off in server.js

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

    console.log('New web socket connection');

    socket.emit('message', 'Welcome to the chatroom');

})

1. Gets sent to main.js and is sent to browser console.

**Handling events & messages**

* In server.js

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

*// Welcome current user*

    socket.emit('message', 'Welcome to the chatroom');

*// Broadcast when a user connects*

*// socket.broadcast.emit will notify everyone except user connecting*

    socket.broadcast.emit('message', 'A user has joined the chatroom');

*// Runs when client disconnects*

    socket.on('disconnect', () => {

        io.emit('message', 'A user has left the chatroom');

    })

})

* In main.js we need to handle the message form submission

chatForm.addEventListener('submit', (event) => {

    event.preventDefault();

*// gets value of*

    const msg = event.target.elements.msg.value

})

* This value comes from the id of ‘msg’ from the chat.html form

   <form id="chat-form">

                <input id="msg" type="text" placeholder="Enter Message" required autocomplete="off" />

                <button class="btn"><i class="fas fa-paper-plane"></i> Send</button>

            </form>

* Once this has been done and the msg is grabbed:

*// Emit message to server here*

    socket.emit('chatMessage', msg);

* Now back in server.js we need to grab the chatMessage that was emitted

*// Listen for chatMessage from form submission on client-side*

    socket.on('chatMessage', (msg) => {

        console.log(msg)

    })

* Now when a message is sent on client-side it will log to the console on the server.\
* Next we have to make sure that once the server grabs the message it will be emitted to everyone else in the chat

*// Listen for chatMessage from form submission on client-side*

    socket.on('chatMessage', (msg) => {

*// using io.emit will send to everyone*

        io.emit('message', msg)

    })

* Now that the message has been created and sent to the server to be shown to everyone, we have to add it to the DOM
* Back in main.js

*// Output message to DOM*

function outputMessage(message) {

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

*// each div has class of message*

    div.classList.add('message');

    div.innerHTML = `<p class="meta">Brad <span>9:12pm</span></p>

    <p class="text">

    ${message}

    </p>`;

    document.querySelector('.chat-messages').appendChild(div);

}

* Now the message that is submitted will show on the page
* Scroll to message

*// Every time a message is received DOM will scroll to it*

    chatMessages.scrollTop = chatMessages.scrollHeight;

* Then to clear inputs and focus

*// Clear inputs*

    event.target.elements.msg.value = '';

    event.target.elements.msg.focus();

**Adding users ID’s and timestamps**

* Create new folder in main directory called utils for utilities
* Create messages.js to add username and time. Uses moment.js library for time format

const moment = require('moment');

function formatMessage(username, text) {

    return {

        username,

        text,

        time: moment().format('h:mm a')

    }

}

module.exports = formatMessage;

* Now we can import this function into server.js to use to format messages
* Back in server.js, create a bot variable

const botName = 'Admin Bot';

* Now in the io.on function, we can emit the message with more dynamic formatting

*// Welcome current user*

socket.emit('message', formatMessage(botName, 'Welcome to the chatroom!'));

socket.broadcast.emit('message', formatMessage(botName, 'A user has joined the chatroom'));

*// Runs when client disconnects*

    socket.on('disconnect', () => {

        io.emit('message', formatMessage(botName, 'A user has left the chatroom'));

    })

* Now that formatMessage has been used on the message that was submitted via the form, the value being returned is no longer a string, it is an object.
* Back in main.js we adjust the following to utilize the object that is returned

*// Output message to DOM*

function outputMessage(message) {

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

*// each div has class of message*

    div.classList.add('message');

    div.innerHTML = `<p class="meta">${message.username} <span>${message.time}</span></p>

    <p class="text">

    ${message.text}

    </p>`;

    document.querySelector('.chat-messages').appendChild(div);

}

**Creating usernames and rooms**

* Add minified version of Qs CDN to chat.html file
  + Now have access to query string library
  + <https://github.com/ljharb/qs?utm_source=cdnjs&utm_medium=cdnjs_link&utm_campaign=cdnjs_library>
* Need to first grab the username and room ID from the URL in main.js

*// Get username and room from URL*

const { username, room } = Qs.parse(location.search, {

*// this will ignore prefix characters in URL*

    ignoreQueryPrefix: true

});

* Then we will send this object back to the server

*// Join chatroom*

socket.emit('joinRoom', { username, room })

* Back in server.js once the joinRoom event is emitted, inside the connection function the user will be welcomed and the broadcast is sent to the other users

 socket.on('joinRoom', ({ username, room }) => {

*// Welcome current user*

        socket.emit('message', formatMessage(botName, 'Welcome to the chatroom!'));

*// Broadcast when a user connects*

*// socket.broadcast.emit will notify everyone except user connecting*

        socket.broadcast.emit('message', formatMessage(botName, 'A user has joined the chatroom'));

    });

* Now create users.js in utils directory.
* This will allow us to create a user

const users = [];

*// Join user to chat*

function userJoin(id, username, room) {

    const user = { id, username, room };

    users.push(user);

    return user;

}

*// Get the current user*

function getCurrentUser(id) {

    return users.find(user => user.id === id);

}

module.exports = {

    userJoin,

    getCurrentUser

}

* Back in server.js require the file to use these functions

const { userJoin, getCurrentUser } = require('./utils/users.js');

* Now inside of the connection function a user variable can be made

socket.on('joinRoom', ({ username, room }) => {

*// socket.id comes from socket in main func*

        const user = userJoin(socket.id, username, room);

        socket.join(user.room)

*// Welcome current user*

        socket.emit('message', formatMessage(botName, 'Welcome to the chatroom!'));

*// Broadcast when a user connects*

*// socket.broadcast.emit will notify everyone excpet user connecting*

        socket.broadcast.emit('message', formatMessage(botName, 'A user has joined the chatroom'));

    });

* Now we have to change the broadcast so that it’s only sent out to the room that the user just joined, not everyone

*// Broadcast when a user connects*

*// socket.broadcast.emit will notify everyone except user connecting*

        socket.broadcast.to(user.room).emit('message', formatMessage(botName, 'A user has joined the chatroom'));

* Now we can check in the browser and verify that when a user joins a specific room, only the users in that room can communicate with each other
* In server.js we can now obtain the user’s ID using the ID from the socket and the getCurrentUser function

*// Listen for chatMessage from form submission on client-side*

    socket.on('chatMessage', (msg) => {

        const user = getCurrentUser(socket.id)

*// using io.emit will send to everyone*

        io.to(user.room).emit('message', formatMessage(user.username, msg))

        console.log(msg)

    })

* Now we have to deal with when a user leaves the chatroom in users.js

*// User leaves chatroom*

function userLeave(id) {

*// check users array for matching ID and return index*

    const index = users.findIndex(user => user.id === id);

*// If the users id is there, return the array without the user so they're removed*

    if (!index !== -1) {

*// need to include [0] to return user, not entire array*

        return users.splice(index, 1)[0];

    }

}

*// Get room users*

function getRoomUsers(room) {

*// Gets only the users with the matchind room*

    return users.filter(user => user.room === room)

}

* Export these functions also to use in server.js Now we can send a message that the user has left the chat room and then the user can be removed from the users array in users.js

*// Runs when client disconnects*

    socket.on('disconnect', () => {

        const user = userLeave(socket.id);

        if (user) {

            io.to(user.room).emit('message', formatMessage(botName, `${user.username} has left the chatroom`));

        }

    })

**Displaying usernames and rooms when joined**

* We need to be able to display all of the current users within a given chatroom
* Start by sending users and room info using the getRoomUsers function.

*// Send users and room info*

        io.to(user.room).emit('roomUsers', {

            room: user.room,

            users: getRoomUsers(user.room)

        })

* Next we need to take in this data in main.js and do something with it (mutating the DOM)
* In main.js

*// Get room and users*

socket.on('roomUsers', ({ room, users }) => {

    outputRoomName(room);

    outPutUsers(users);

});

* Grab the elements from the chat.html DOM to append the room and user to

const roomName = document.querySelector('#room-name');

const userList = document.querySelector('#users');

adsv