Specifications

Using Node.js, create an IRC-like chatroom using WebSockets. WebSockets are a great fit for building a real-time chat application because they allow for persistent, bidirectional communication between the server and connected clients.

You only need to create one chatroom that is automatically joined, but you must implement the following:

- The user must set a nickname before joining the chatroom.
- When the user sends a message, broadcast it to all connected users.
- When a user joins the chatroom or leaves it, broadcast a message to all connected users.
- Implement the following commands:
  - `/nick <name>` – Change the user's nickname
  - `/list` – Prints all connected users
  - `/me <action>` – Lets the user perform an action in the third person. For example, `/me laughs` would display as "CoolUser laughs" in the chat.
  - `/help` – Displays a list of available commands.

Project Creation Instructions

Use the following instructions to get your project created.

1. Create an empty directory in the cloned repository.
   - The name should be all lowercase.
2. Install packages by running the following commands:
   - `npm install express ws concurrently`
   - `npm install --save-dev nodemon parcel`
3. Create directories "server" and "public".
4. In the new "server" directory, create a file called server.mjs
5. In the new "public" directory, create a file called index.html.
6. Change the "scripts" field in the "package.json" file to the following:
   - ```
     "scripts": {
         "server": "nodemon server/server.mjs",
         "client": "parcel serve public/*.html",
         "start": "concurrently \"npm run server\" \"npm run client\""
       }
     ```
7. Create an npm runtime configuration that runs the "start" script.
     ```

Constraints

- Do not use any other npm packages besides the ones listed in the “Project Creation Instructions” section.
  - As an exception, if you’d like, you can use the Bootstrap npm package.
- Do not use any “polling” techniques. Use only web socket connections for communication.
- Every user must have a username. Usernames do not need to be unique.

Extra Credit Opportunities

- (5 pts) Implement a direct message command.
  - Example: `/msg <nickname> <message>` | `/msg CoolUser Hey friend.`
- (8 pts) Host your application publicly using a home server, Azure, AWS, or any other service.
  - You cannot use a paid service. If you use a 3rd party service, you must use a free plan.

Submission

You will submit the commit ID for the commit you want graded. Submit the commit ID on the Canvas assignment.

Tips

- See the example code on the last page for a small example of how to implement a chatroom on the server side.
- You cannot have more than one WebSocket instance per server.

Academic Honesty Policy

All code submissions must be original and authored by the student. Any code sourced from another student, falsely presented as one's own, or derived from third-party websites or generated by AI, will constitute a breach of the school's academic honesty policy.

Daily commits and pushes to your assignment GitHub repository are <u>mandatory</u> while working on your project. Failure to comply will result in deductions from your final grade at minimum, and could lead to academic honesty violations.

Example Code

```
const wss = new WebSocketServer({ server });

const clients = new Map();

wss.on('connection', (ws) =>
{
    const id = // Some unique identifier
    clients.set(id, ws);
    ws.send('Welcome to the chat! Use /nick <name> to set a nickname.');

    ws.on('message', (message) =>
    {
        // Command handling (e.g., /nick)
        if (message.startsWith('/nick '))
        {
            const nickname = message.split(' ')[1];
            clients.set(id, { ws, nickname });
            // Send a message back to the user
            ws.send(`Nickname set to ${nickname}`);
        }
        else
        {
            // Broadcast message to all clients
            const nickname = clients.get(id).nickname;
            const fullMessage = `${nickname}: ${message}`;
            clients.forEach((client) =>
            {
                if (client.ws !== ws)
                {
                    client.ws.send(fullMessage);
                }
            });
        }
    });

    ws.on('close', () =>
    {
        clients.delete(id);
    });
});
```