이 코드는 **Node.js**와 **Socket.IO**를 사용하여 간단한 실시간 채팅 애플리케이션을 만드는 예시입니다. 이 애플리케이션은 **Express**를 사용해 서버를 설정하고, **Socket.IO**를 이용해 클라이언트 간의 실시간 소켓 통신을 처리합니다.

---

1. 필요한 모듈 가져오기

```
const express = require('express');
const http = require('http');
const socketIo = require('socket.io');
const path = require('path');
```

- **express**: 웹 서버를 쉽게 만들 수 있게 해주는 라이브러리입니다.
- **http**: Node.js의 기본 HTTP 서버 모듈로, Express 앱을 HTTP 서버로 만드는데 필요합니다.
- **socket.io**: 실시간 양방향 통신을 제공하는 라이브러리입니다. 채팅 등 실시간 애플리케이션에 적합합니다.
- **path**: 경로를 다룰 때 사용하는 Node.js의 기본 모듈입니다.

---

2. 서버와 Socket.IO 설정

```
const app = express();
const server = http.createServer(app);
const io = socketIo(server);
```

- `app` : Express 애플리케이션을 생성합니다.
- `server` : Express 애플리케이션을 HTTP 서버로 변환합니다.
- `io` : `socket.io` 객체로, 이 객체를 통해 클라이언트와 실시간 통신을 설정하고 관리합니다.

---

3. 정적 파일 제공

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

- `app.use(express.static(...))` : `public` 폴더를 정적 파일 경로로 설정합니다. `index.html` 과 같은 HTML, CSS, JS 파일을 `public` 폴더에 넣으면 클라이언트에서 쉽게 접근할 수 있습니다.

4. 라우트 설정

```
app.get('/', (req, res) => {
  res.sendFile('index.html');
});

app.get('/temp', (req, res) => {
  res.sendFile('temp.html', { root: path.join(__dirname, 'public') });
});
```

- `app.get('/', ...)` : 루트 경로 `/` 에 접속하면 `index.html` 파일을 전송합니다.
- `app.get('/temp', ...)` : `/temp` 경로에 접속하면 `public` 폴더에 있는 `temp.html` 파일을 전송합니다.

---

5. 사용자와 채팅방 목록

```
let users = {}; // 사용자 목록
let rooms = {}; // 채팅방 목록
```

- `users` : 소켓 ID와 사용자 이름을 연결하는 객체입니다. 각 연결된 사용자마다 고유의 ID가 할당됩니다.
- `rooms` : 채팅방 목록을 저장하는 객체입니다.

---

6. Socket.IO 이벤트 설정

6.1 클라이언트 연결 처리

```
io.on('connection', (socket) => {
  console.log(`user connected => ${JSON.stringify(users)} :
${JSON.stringify(rooms)}`);
```

- `io.on('connection', ...)` : 클라이언트가 서버에 연결되면 실행되는 기본 이벤트입니다. 여기서 `socket` 객체는 특정 클라이언트와의 연결을 나타냅니다.

6.2 사용자 로그인

```
socket.on('login', (username) => {
  users[socket.id] = username;
  console.log(`login => ${username}`);
  socket.emit('login:success', { username, rooms: Object.keys(rooms) });
  io.emit('update:users', Object.values(users));
});
```

- `socket.on('login', ...)` : 클라이언트가 `login` 이벤트를 통해 사용자 이름을 전송하면 실행됩니다.
  - `users[socket.id] = username;` : `users` 객체에 새로운 사용자 정보를 저장합니다.
  - `socket.emit('login:success', ...)` : 클라이언트에게 로그인 성공을 알리고, 현재 방 목록을 전송합니다.
  - `io.emit('update:users', ...)` : 모든 클라이언트에게 업데이트된 사용자 목록을 전송합니다.

6.3 채팅방 생성

```
socket.on('create:room', (room) => {
  if (!rooms[room]) {
    rooms[room] = [];
    socket.join(room);
    socket.emit('room:created', room);
    io.emit('update:rooms', Object.keys(rooms));
  } else {
    socket.emit('room:exists', room);
  }
});
```

- `socket.on('create:room', ...)` : 클라이언트가 방을 생성하려고 할 때 실행됩니다.
  - `rooms[room] = [];` : 새로운 방을 `rooms` 객체에 추가합니다.
  - `socket.join(room);` : 사용자를 새 방에 참여시킵니다.
  - `socket.emit('room:created', room);` : 클라이언트에게 방이 생성되었음을 알립니다.
  - `io.emit('update:rooms', ...)` : 모든 클라이언트에게 방 목록을 전송합니다.

6.4 채팅방 참가

```
socket.on('join:room', (room) => {
  if (rooms[room]) {
    socket.join(room);
    socket.emit('room:joined', room);
```

```
      socket.to(room).emit('user:joined', users[socket.id]);
    } else {
      socket.emit('room:notfound', room);
    }
  });
```

- `socket.on('join:room', ...)` : 클라이언트가 특정 방에 참여할 때 실행됩니다.
  - `socket.join(room);` : 해당 방에 사용자를 추가합니다.
  - `socket.emit('room:joined', room);` : 방에 성공적으로 참여했음을 클라이언트에게 알립니다.
  - `socket.to(room).emit('user:joined', ...)` : 방에 있는 다른 사용자들에게 새로운 사용자가 참여했음을 알립니다.

6.5 메시지 전송

```
  socket.on('chat:message', ({ room, message }) => {
    io.to(room).emit('chat:message', { user: users[socket.id], message });
  });
```

- `socket.on('chat:message', ...)` : 클라이언트가 메시지를 전송하면 실행됩니다.
  - `io.to(room).emit('chat:message', ...)` : 해당 방에 있는 모든 사용자들에게 메시지를 전송합니다.

6.6 사용자 연결 해제

```
  socket.on('disconnect', () => {
    if (users[socket.id]) {
      io.emit('user:left', users[socket.id]);
      delete users[socket.id];
      io.emit('update:users', Object.values(users));
    }
  });
});
```

- `socket.on('disconnect', ...)` : 사용자가 서버와의 연결을 끊을 때 실행됩니다.
  - `io.emit('user:left', users[socket.id]);` : 모든 클라이언트에게 해당 사용자가 떠났음을 알립니다.
  - `delete users[socket.id];` : `users` 객체에서 해당 사용자를 제거합니다.
  - `io.emit('update:users', ...)` : 모든 클라이언트에게 업데이트된 사용자 목록을 전송합니다.

7. 서버 시작

```
const PORT = process.env.PORT || 3000;
server.listen(PORT, () => {
  console.log(`Server is running on port ${PORT}`);
});
```

- `PORT` : 서버가 실행될 포트를 지정합니다.
- `server.listen(...)` : 지정된 포트에서 서버를 실행합니다.

이 코드는 HTML, CSS, 그리고 JavaScript를 사용하여 **Socket.IO** 기반의 실시간 채팅 웹 애플리케이션을 만듭니다. 여기서 `index.html` 은 클라이언트 측 코드로, 서버와의 통신을 통해 채팅, 로그인, 방 생성 등의 기능을 구현합니다.

---

1. HTML 헤더와 스타일링

```
<!DOCTYPE html>
<html>
<head>
  <title>Socket.IO Chat</title>
  <!-- Minified version -->
  <link rel="stylesheet" href="https://cdn.simplecss.org/simple.min.css">
</head>
<body>
```

- `<!DOCTYPE html>` : HTML 문서가 HTML5임을 선언합니다.
- `<title>Socket.IO Chat</title>` : 브라우저 탭에 표시될 페이지 제목을 설정합니다.
- `<link rel="stylesheet" href="https://cdn.simplecss.org/simple.min.css">` : 외부 CSS 라이브러리 **Simple.css**를 가져옵니다. 이 라이브러리를 통해 기본적인 스타일을 설정할 수 있습니다.

---

2. HTML 구조 (로그인, 방 생성, 채팅창)

```
<section>
  <div class="container py-5">
    <!-- 로그인 섹션 -->
    <div class="col-md-8 col-lg-6 col-xl-4">
      <h2>Login</h2>
      <input id="username" placeholder="Enter username"
```

```
autocomplete="off"/>
      <button id="login" class="btn btn-danger">Login</button>
    </div>
```

- `username`과 `login` 버튼을 통해 사용자가 로그인할 수 있도록 설정했습니다.
- `username` 입력 필드는 사용자가 채팅에서 사용할 닉네임을 입력받는 필드입니다.

2.1 채팅 컨테이너 (방 생성, 방 목록, 사용자 목록, 채팅창)

```
    <!-- 채팅 컨테이너 -->
    <div class="card" id="chat-container" style="display:none; border-
radius: 15px;">
      <div class="card-body">
        <h2>Create Room</h2>
        <input id="room" name="room" placeholder="Enter room name"
autocomplete="off"/>
        <button id="create-room" class="btn btn-primary">Create
Room</button>
      </div>
```

- `chat-container`는 로그인 이후 나타나는 메인 채팅 영역입니다. 초기에는 `display:none`으로 설정되어 숨겨져 있다가, 로그인 후에 표시됩니다.
- `room` 입력 필드는 새로운 채팅방 이름을 입력하는 필드입니다.
- `create-room` 버튼을 클릭하면 방이 생성됩니다.

방 목록, 사용자 목록, 채팅창

```
      <!-- 룸 목록 -->
      <div class="card-body">
        <h2>Rooms</h2>
        <ul id="rooms"></ul>
      </div>
      <!-- 사용자 목록 -->
      <div class="card-body">
        <h2>Users</h2>
        <ul id="users"></ul>
      </div>
      <!-- 채팅창 -->
      <div class="card-body">
        <h2>Chat</h2>
        <ul id="chat"></ul>
        <form id="form" action="">
          <input id="input" autocomplete="off" placeholder="Type your
message here"/>
```

```
            <button class="btn btn-info">Send</button>
          </form>
        </div>
      </div>
    </div>
  </section>
```

- **Rooms**: `rooms` 리스트에 생성된 방 목록이 표시됩니다.
- **Users**: `users` 리스트에 현재 접속한 사용자 목록이 표시됩니다.
- **Chat**: 채팅창은 `chat` 리스트와 메시지 입력 `input`으로 구성되어 있습니다.

---

3. JavaScript와 Socket.IO 설정

```
<script src="/socket.io/socket.io.js"></script>
<script>
  var socket = io();
  var currentRoom = '';
```

- `/socket.io/socket.io.js`: Socket.IO의 클라이언트 라이브러리를 로드합니다.
- `socket = io()`: Socket.IO 클라이언트를 생성하여 서버와 실시간 연결을 만듭니다.
- `currentRoom`: 사용자가 현재 참여한 방을 저장합니다.

---

4. HTML 요소 참조

```
  var loginButton = document.getElementById('login');
  var usernameInput = document.getElementById('username');
  var chatContainer = document.getElementById('chat-container');
  var form = document.getElementById('form');
  var input = document.getElementById('input');
  var chatList = document.getElementById('chat');
  var roomInput = document.getElementById('room');
  var createRoomButton = document.getElementById('create-room');
  var roomsList = document.getElementById('rooms');
  var usersList = document.getElementById('users');
```

- HTML 요소들을 JavaScript에서 쉽게 접근하고 조작할 수 있도록 변수에 저장합니다.

---

5. 이벤트 리스너 설정

로그인

```
loginButton.addEventListener('click', function() {
  var username = usernameInput.value;
  if (username) {
    console.log(`login : ${username}`)
    socket.emit('login', username); // 사용자 로그인 요청
  }
});
```

- `loginButton`: 사용자가 닉네임을 입력하고 로그인 버튼을 클릭하면, `login` 이벤트를 서버로 전송하여 로그인 요청을 보냅니다.

방 생성

```
createRoomButton.addEventListener('click', function() {
  var room = roomInput.value;
  if (room) {
    socket.emit('create:room', room); // 방 생성 요청
  }
});
```

- `createRoomButton`: 방 이름을 입력하고 버튼을 클릭하면, `create:room` 이벤트를 서버로 전송하여 새로운 방을 생성합니다.

메시지 전송

```
form.addEventListener('submit', function(e) {
  e.preventDefault();
  if (input.value) {
    var room = currentRoom;
    if(room){
      console.log(`submit => ${room} ${input.value}`);
      socket.emit('chat:message', { room, message: input.value });
    }
    input.value = '';
  }
});
```

- `form`: 메시지를 입력하고 엔터를 누르거나 전송 버튼을 클릭하면, `chat:message` 이벤트를 서버로 전송하여 현재 방에 메시지를 보냅니다.

---

6. Socket.IO 이벤트 리스너 (서버에서 오는 이벤트 처리)

로그인 성공

```
socket.on('login:success', function(data) {
  chatContainer.style.display = 'block';
  usernameInput.disabled = true;
  loginButton.disabled = true;
  updateRooms(data.rooms);  // 룸 정보 업데이트
});
```

- `login:success` : 서버에서 `login:success` 이벤트가 오면, 로그인 UI를 숨기고 채팅 UI를 보여줍니다. 또한 방 목록을 업데이트합니다.

방 생성

```
socket.on('room:created', function(room) {
  var item = document.createElement('li');
  item.textContent = room;
  roomsList.appendChild(item);
  item.addEventListener('click', function() {
    currentRoom = room;
    item.style = 'background-color:orange;'
    socket.emit('join:room', room);
  });
});
```

- `room:created` : 서버가 새 방을 생성했다고 알리면, 방 이름을 목록에 추가하고 클릭 시 해당 방에 참여합니다.

방 참가, 방 존재 확인

```
socket.on('room:exists', function(room) {
  alert('Room ' + room + ' already exists.');
});
socket.on('room:joined', function(room) {
  alert('Joined room: ' + room);
});
```

- `room:exists` : 방이 이미 존재한다는 메시지를 표시합니다.
- `room:joined` : 방에 성공적으로 참여했다는 메시지를 표시합니다.

사용자 및 방 목록 업데이트

```
socket.on('update:rooms', function(rooms) {
  updateRooms(rooms);
});
socket.on('update:users', function(users) {
  updateUsers(users);
});
```

- `update:rooms`: 서버가 방 목록을 갱신하면, `updateRooms` 함수가 실행됩니다.
- `update:users`: 서버가 사용자 목록을 갱신하면, `updateUsers` 함수가 실행됩니다.

채팅 메시지 수신

```
socket.on('chat:message', function(data) {
  var item = document.createElement('li');
  item.textContent = data.user + ': ' + data.message;
  chatList.appendChild(item);
  window.scrollTo(0, document.body.scrollHeight);
});
```

- `chat:message`: 서버로부터 메시지를 수신하면, 이를 채팅 목록에 추가합니다.

사용자 참여 및 나감 알림

```
socket.on('user:joined', function(user) {
  var item = document.createElement('li');
  item.textContent = user + ' joined the room';
  chatList.appendChild(item);
  window.scrollTo(0, document.body.scrollHeight);
});

socket.on('user:left', function(user) {
  var item = document.createElement('li');
  item.textContent = user + ' left the chat';
  chatList.appendChild(item

);
  window.scrollTo(0, document.body.scrollHeight);
});
```

- `user:joined`, `user:left`: 사용자가 채팅방에 들어오거나 나갈 때 알림을 표시합니다.

---

7. 방 및 사용자 업데이트 함수

```
function updateRooms(rooms) {
  roomsList.innerHTML = '';
  rooms.forEach(function(room) {
    var item = document.createElement('li');
    item.textContent = room;
    item.addEventListener('click', function() {
      currentRoom = room;
      item.style = 'background-color:orange;'
      socket.emit('join:room', room);
    });
    roomsList.appendChild(item);
  });
}

function updateUsers(users) {
  usersList.innerHTML = '';
  users.forEach(function(user) {
    var item = document.createElement('li');
    item.textContent = user;
    usersList.appendChild(item);
  });
}
```

- `updateRooms` : 방 목록을 갱신하여 최신화된 방 목록을 표시합니다.
- `updateUsers` : 사용자 목록을 갱신하여 현재 채팅에 참여 중인 사용자를 표시합니다.