
웹 채팅 프로그램

2022.02.26



이름	최승원
학부	경영학부
학번	2018126103

목차

1. 서론

1.1 웹 채팅 프로그램 구현 목적 및 기능 개요

1.2 프로그램 구현 방식

1.2.1 socket.io 모듈

2. 본론

2.1 기능 및 구현 방식 소개

2.1.1 웹 소켓 통신

2.2.2 실시간 채팅

2.2.3 방 기능

2.2.4 참여자 정보

2.2 웹 채팅 프로그램 UI

2.2.1 사용자 정보 입력

2.2.2 방 입장 및 채팅 전송

2.2.3 응답 메시지

3. 결론

3.1 개선 방향

3.1.1 참여자 목록

3.2 느낀점

1.1 웹 채팅 프로그램 기능 개요

서버와 클라이언트 사이의 통신 방식을 이해하기 위해 웹 node.js 를 사용하여 웹 채팅 프로그램을 구현하였다. 웹 채팅 프로그램의 주요 기능은 크게 세 가지로 구성되어 있다.

첫 번째 기능은 실시간 채팅 기능이다. 실시간 채팅 기능은 클라이언트가 전송한 메시지를 서버에서 실시간으로 같은 서버에 접속한 다른 클라이언트들에게 전송할 수 있도록 구현하였다. 따라서 페이지를 새로고침할 필요 없어 지연시간의 감소로 실시간 채팅이 가능해진다.

두 번째는 채팅방 별 실시간 채팅이다. 클라이언트가 서버에 접속할 때 어떤 방으로 접속할건지 입력하게 된다. 이때 입력된 방 번호를 통해 입장하게 되는 방이 나누어 지며, 자신이 속해있는 채팅방 내의 참여자가 보낸 채팅만 받아볼 수 있다.

세 번째는 참여자 정보 알림 기능이다. 클라이언트가 서버에 접속할 때 사용자의 이름을 입력하여 서버에 전달하게 되어 각 사용자가 전송하는 메시지에 위에는 사용자의 이름 정보가 함께 나타난다.

1.2 웹 채팅 프로그램 구현 방식

1.2.1 Socket.io

실시간 웹 채팅 프로그램 구현을 위해 socket.io 모듈을 사용했다. HTTP 를 사용하면 클라이언트가 원하는 정보를 서버에 요청하고, 서버는 요청을 받아 요청받은 것에 대해 응답하는 방식으로 클라이언트와 서버 사이의 통신이 이루어졌다. 하지만 HTTP 를 이용하면 클라이언트가 서버로 요청을 하는 경우에만 서버에서 응답으로 보내준 새로운 데이터를 받아 페이지 새로고침을 통해 웹브라우저로 보여줄 수 있었다. 하지만 클라이언트가 요청을 보내고 데이터를 응답 받을 때마다 페이지를 전환해야하고, 요청을 보내야만 데이터를 전달받을 수 있다는 단점이 있다. 이를 보완하고자 자바스크립트로 돔을 조작해 페이지 전체를 전환하지 않고도 서버로부터 전달받은 데이터를 업데이트 할 수 있는 Ajax 방식을 사용할 수 있지만, 여전히 클라이언트가 버튼을 누르는 등의 요청을 보낼 때만 서버로부터 응답받은 새로 업데이트된 데이터를 보여줄 수 있다.

이러한 데이터 교환 방식을 보완하기 위해 웹 소켓이라는 데이터 통신 방식이 등장하였고, 해당 방식을 사용하면 클라이언트의 요청이나 클라이언트 측에서의 페이지 전환 없어도 새로운 정보가 업데이트 되었을 때 서버에서 연결되어있는 클라이언트로 데이터를 전송할 수 있는 양방향 통신이 가능하게 되었다.

실시간 웹 채팅 기능은 자신이 보낸 메시지가 아닌 상대가 보낸 메시지에 대해서도 데이터를 받기 위한 특별한 요청 없이도 상대가 보낸 메시지 데이터가 업데이트 되어야한다. 따라서 웹

소켓 방식을 이용하여 실시간 웹 채팅 기능을 구현한다. 이때 웹 소켓을 사용할 수 있는 라이브러리로 socket.io 모듈을 사용하였다.

2.1 기능 및 구현 방식 소개

실시간 채팅을 구현하기 위해서는 클라이언트와 서버 사이의 실시간 양방향 통신이 가능해야 한다. 이를 구현하기 위해 socket.io 라이브러리를 사용하며 node.js 로 서버 코드를 구현하고, html 로 클라이언트 페이지를 구현하여 서버와 클라이언트 간의 통신이 이루어진다.

2.1.1 웹 소켓 통신 방식

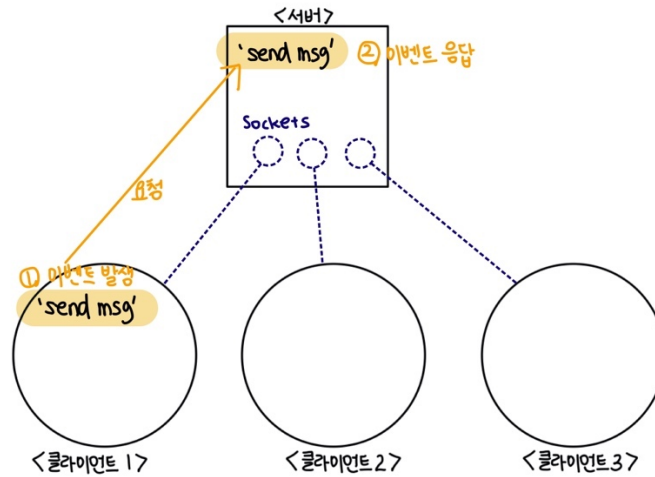
웹 서버는 클라이언트가 요청한 것에 대해 응답하는 것을 가능하게 하는 기능이다. 서버를 구현하기 위해서는 http 모듈에 있는 Server 클래스의 인스턴스를 만들어 사용할 수 있다. 따라서 서버와 클라이언트의 통신을 가능하게 하기 위해서는 서버가 존재해야 하고, 웹 소켓에 이 웹 서버를 넘겨줌으로써 웹 소켓이 서버를 통해 클라이언트와 데이터를 교환할 수 있도록 할 수 있다.

이때 http 내장 모듈을 이용하여 서버를 실행하거나 클라이언트의 요청과 이에 대한 응답을 처리하게 된다면 복잡한 과정을 반복하여 거쳐야하기 때문에 express API 를 사용하여 서버를 생성하고, 이 서버에 웹 소켓 기능을 부착해 웹 소켓 기능을 갖는 서버를 생성한다.

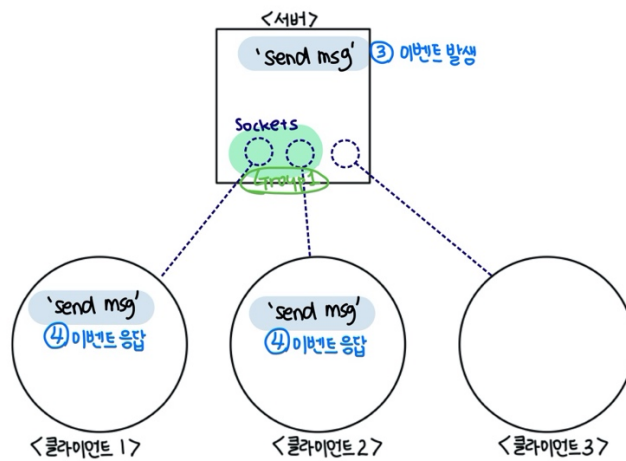
즉, 웹 소켓이 http 서버를 가지고 이것을 통해 클라이언트와 통신이 가능하게 되며, 웹 소켓 서버가 웹 소켓 클라이언트와 연결될 때 소켓에 각 클라이언트가 할당되어 그 연결상태를 지속할 수 있으며 필요시 연결되어 있는 소켓으로 이벤트를 전송할 수 있게 된다.

이와 같은 방법으로 실시간 채팅 프로그램을 구현하였고, 통신 방식을 간단히 그림으로 나타내면 <이미지 1>과 <이미지 2>와 같다. 웹 소켓 서버에는 접속한 클라이언트의 정보가 저장되어 있으며 일정 시간 동안 연결되어 있고, 소켓이 지속적으로 연결되어 있기 때문에 클라이언트에서 요청을 보내지 않아도 클라이언트로 이벤트를 전달할 수 있다.

<이미지 1>은 클라이언트가 메시지를 전송하면 서버에서 해당 이벤트를 전달받아 이벤트에 대한 처리를 하는 과정이고, <이미지 2>는 클라이언트가 메시지를 전송하여 이벤트를 처리한 이후, 서버에서 클라이언트로 이벤트를 발생시키는 과정이다.



<이미지 1>



<이미지 2>

2.1.2 실시간 채팅

< 코드 1-1 >은 index.html 에 작성된 form 태그 코드, <코드 1-2>는 form 내부의 submit 버튼을 클릭하는 이벤트 발생시 실행하게 될 코드이다.

```
<main>
  <ul id="messages"></ul>
  <form action="" id="form">
    <input type="text" id="input">
    <input type="submit" id="submit">
  </form>
</main>
```

< 코드 1-1 >

```
const form = document.getElementById('form');
const submit = document.getElementById('submit');
const ul = document.getElementById('messages');
const clientsList = document.getElementById('clients-list');
const header = document.querySelector('header > h1');

form.addEventListener('submit', (e) => {
  e.preventDefault();

  var msg = document.getElementById('input');
  if(msg.value) {
    socket.emit('send msg',{
      message : msg.value
    });
    msg.value = '';
  }
})
```

<코드 1-2>

클라이언트가 메시지를 입력하여 전송 버튼을 누르면 form 태그의 submit 이벤트에 대한 이벤트 리스너가 실행되고, 입력한 메시지가 존재한다면 웹 소켓 서버로 이벤트를 전달할 수 있다. 이때 전달하는 이벤트의 이름은 'send msg'이고, 따라서 웹 소켓 서버는 'send msg'의 이름으로 이벤트를 받아 이벤트에 대한 처리를 할 수 있게 된다.

<코드 1-3>은 웹 소켓 클라이언트에게 이벤트 'send msg'를 전달받았을 때 실행하게 될 코드이다.

```
socket.on('send msg', (msg) => {
  io.sockets.in(socket.room).emit('send msg', {
    message : msg.message,
    name : socket.name,
    id : socket.id,
  })
})
```

<코드 1-3>

웹 소켓 클라이언트에서 웹 소켓 서버로 이벤트를 발생시킬 수 있고, 마찬가지로 웹 소켓 서버에서도 웹 소켓 클라이언트로 이벤트를 발생시킬 수 있다.

첫 번째 줄에 있는 socket.on ('send msg', () => { 익명 함수 })는 웹 소켓 클라이언트가 요청한 이벤트에 대해 처리를 위한 코드이다. 즉, 해당 코드에서는 'send msg' 이벤트를 요청받았을 때 io.sockets.in().emit() 코드를 수행하게 된다.

하지만 두 번째 줄의 io.sockets.in().emit()은 이벤트를 발생시키는 코드이다. 웹 소켓을 이용하면 서버와 클라이언트의 양방향 통신이 가능하게 되었고 여기에서 양방향 통신이 발생하는 것이다. 즉, 웹 서버에서 'send msg'라는 이름의 이벤트를 웹 소켓 클라이언트로 전달하게 된다.

여기서 첫 번째 줄의 'send msg'와 두 번째 줄의 'send msg'는 다른 이벤트이며 첫 번째 이벤트는 클라이언트가 발생시키는 이벤트, 두 번째 이벤트는 서버가 발생시키는 이벤트이다.

따라서 클라이언트에서 'send msg' 이벤트를 발생시키면 해당 메시지 내용을 서버에 전달하고, 내용을 전달받은 서버는 내용을 보낸 소켓의 정보와 메시지를 함께 서버에 연결된 클라이언트 측에 다시 보내는 'send msg'라는 이벤트를 발생시키게 된다. 이때 client1 이 보낸 메시지에 대한 정보를 client2, client3 도 받아볼 수 있게 된다.

2.1.3 방 기능

Socket.io 가 제공하는 socket.join() 내장함수를 이용하면 웹 소켓 서버와 연결되어 있는 클라이언트들을 특정 이름을 갖는 그룹으로 묶을 수 있다.

```
const socket = io();
var conn = 0; // 접속 횟수

if(conn == 0) {
  console.log(conn);
  var room = prompt('입장할 방 번호를 입력하세요. ');
  var name = prompt('이름을 입력하세요. ');
  conn++;
  socket.emit('makeRoom', {
    roomNumber : room,
    name : name
  });
}
```

<코드 2-1>

<코드 2-1>은 클라이언트가 서버에 접속했을 때 클라이언트 측에서 실행되는 코드이다. 먼저 prompt()를 통해 방 번호와 사용자 이름을 입력받고 'makeRoom'이라는 이벤트 이름으로 사용자가 입력한 방 번호와 사용자 이름을 담은 객체를 웹 소켓 서버에 전달한다.

```
socket.on('makeRoom', (socketInfo) => {
  socket.join(socketInfo.roomNumber);
  // 이름, 방번호 저장
  socket.name = socketInfo.name;
  socket.room = socketInfo.roomNumber;

  // 접속자 현황
  io.sockets.in(socketInfo.roomNumber).emit('connectToRoom', {
    message : socket.name + ' now in room ' + socketInfo.roomNumber,
    id : socket.id
  });
})
```

<코드 2-2>

<코드 2-2>는 'makeRoom' 이벤트를 전달받았을 때 실행되는 코드이다.

Socket.join('그룹 이름')을 한다면 이벤트를 발생한 socket 을 '그룹 이름'에 해당하는 그룹으로 묶어주고, 사용자 방 번호와 사용자 이름이 담긴 객체를 socketInfo 에 전달받아 소켓의 속성에 추가해준다.

이후 `io.sockets.in('그룹 이름').emit()`을 통해 해당 방에 속한 클라이언트에게만 `'connectToRoom'` 이벤트를 발생시킨다.

여기서 `join` 을 할 때는 `socket.join()`을, 클라이언트에게 이벤트를 전송할 때는 `io.sockets.in()`을 사용하는 이유는 전자의 경우에는 이벤트를 발생한 `socket` 이 연결될 때 객체로 전달되어 전달받은 객체를 그룹에 추가시켜야하기 때문이고, 후자의 경우에는 현재 웹 소켓 서버에 연결된 소켓들에게 이벤트를 발생시켜야 하기 때문이다.

따라서 클라이언트가 서버에 접속했을 때 입력받은 방 번호와 이름 정보를 `'makeRoom'` 이벤트를 발생시키면서 전달하게 되며, 이 정보들을 소켓 객체 속성에 저장하고 같은 방 번호를 가진 소켓과 묶여 그룹을 형성하게 된다. 이후 같은 그룹에 속한 클라이언트에게만 `'connectToRoom'` 이벤트를 발생시키며 해당 그룹에 속한 소켓에게는 `'(name) now in room (roomNumber)'` 메시지를 전송하여 클라이언트는 서버에 특별한 요청을 하지 않아도 해당 메시지를 받아볼 수 있게 된다.

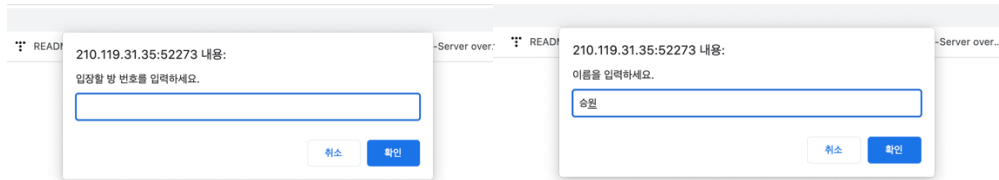
2.1.4 참여자 정보

<코드 2-1>에서 입력받은 참여자 정보는 처음 서버에 접속했을 때 서버에 `'makeRoom'`이벤트와 함께 전달하고, 따라서 서버에서는 <코드 2-2>에서 `'makeRoom'` 이벤트를 전달받을 때 참여자 정보를 전달받아서 소켓의 속성에 추가할 수 있다. 따라서 각 소켓 객체에는 `name` 과 `roomNumber` 속성이 추가되어 필요시에 접속한 소켓의 이름을 활용할 수 있다.

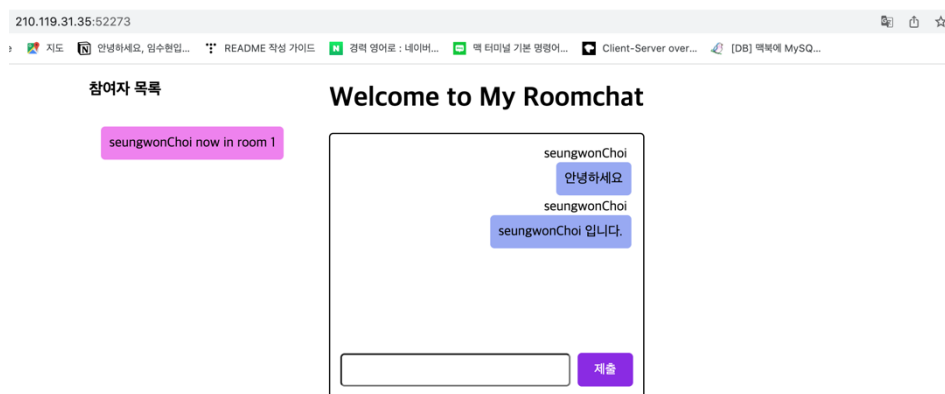
또한 참여자가 채팅방을 나가게 되는 경우에는, 즉 서버에서 `socket.on('disconnect', {})` 이벤트를 전달받았을 때 서버에서 `'close'` 이벤트를 발생시켜 `'disconnect'`이벤트를 발생시킨 소켓의 아이디를 `'close'`이벤트와 함께 클라이언트로 전달한다. 클라이언트는 해당 `id` 를 `id` 속성으로 가진 돔을 찾고 삭제하여 참여자 목록에서 삭제한다.

2.2 웹 채팅 프로그램 UI

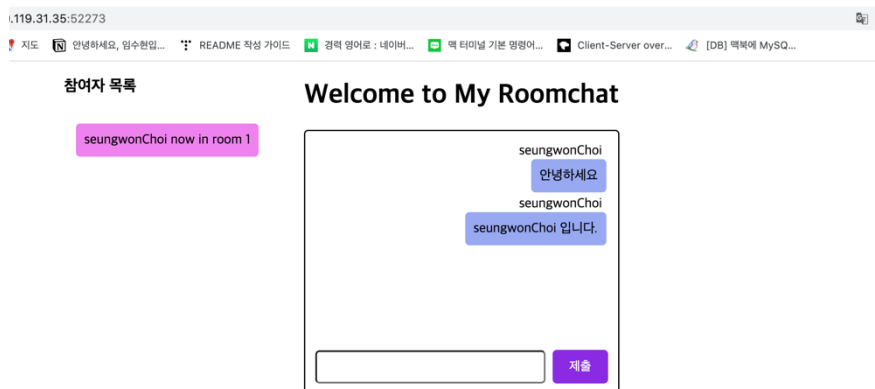
2.2.1 사용자 정보 입력



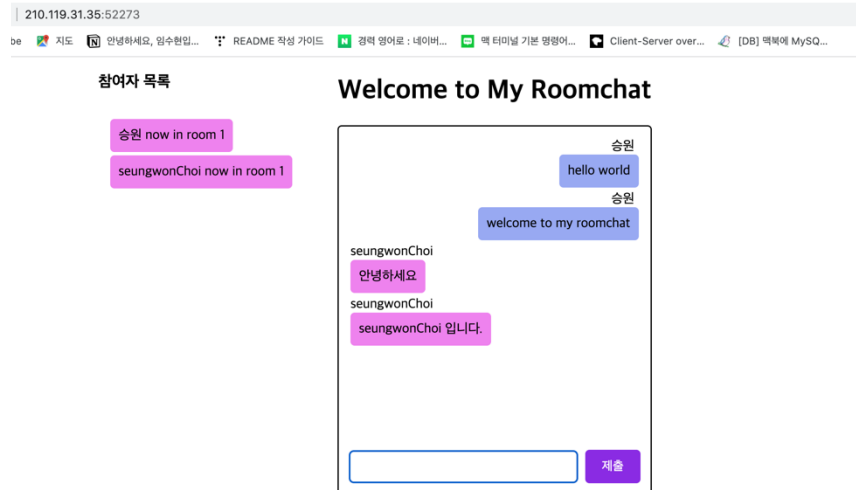
2.2.2 방 입장 및 채팅 전송



2.2.3 응답 메시지



< 'seungwonChoi'가 접속한 채팅방 >



< '승원'이 접속한 채팅방 >

3.1 개선 방향

3.1.1 참여자 목록

현재 참여자 목록은 클라이언트 A 가 채팅방에 접속한 시점 이전에 채팅방에 접속했던 클라이언트들의 정보는 없고 그 이후 접속한 클라이언트의 정보만 담고 있다. 예컨대 카카오톡의 'OO 님이 들어왔습니다.'와 같은 알림기능과 비슷하다.

이유는 참여자가 서버에 접속했을 때 웹 소켓 서버에서 발생시키는 'connectToRoom'이벤트에서 새로 접속한 클라이언트가 속한 그룹에 새로 접속한 클라이언트 정보만을 전달하기 때문이다.

따라서 '클라이언트가 접속했습니다' 알림의 기능과 비슷하면서도 특정 클라이언트가 접속을 끊으면 참여자 목록에서 지워지는 부분에서는 알림의 기능보다도 참여자 목록의 기능에 가깝기 때문에 수정이 필요하다.

현재 채팅방의 참여자 목록을 모두 표시하기 위해서는 서버에서 소켓별 그룹을 join 할 때 그룹별 배열을 생성하여 참여자 목록을 저장하거나, 반복문을 통해 socket.group 속성이 찾는 그룹과 같은 경우에 특정 이벤트를 발생시킬 수 있도록 수정이 필요하다.

3.2 느낀점

실시간 웹 채팅 프로그램을 구현하기 위해 클라이언트와 서버가 통신하는 방식을 공부하면서 데이터를 주고받는 과정을 배우게 되었다.

클라이언트와 서버의 개념부터 생소한 개념이었기 때문에 처음 공부를 시작했을 때는 쉽게 이해되지 않았지만 이해가 되지 않는 부분이 있다면 최대한 이해해보려고 노력한 뒤 뒷 내용으로 먼저 넘어가 전체적인 흐름을 확인하고 다시 돌아가 이해해보려고 하였다. 여러번 봐도 와닿지 않는 개념이 있었는데, 직접 구현하는 과정에서 이해되는 부분도 있었고 레포트를 작성하면서 정리되는 개념도 있었다. 특히 웹 소켓과 관련된 내용은 레포트를 정리하면서 납득하고 이해할 수 있게 되었다.

생소한 용어가 이해되지 않는다고 해서 하나의 단어가 이해될 때까지 잡고있는 것이 아니라, 전체적인 흐름을 이해하고 직접 구현해보고 질문하고 스스로도 직접 찾아보면서 시간과 노력을 투자하는 과정이 처음 접하는 개념을 나의 것으로 체화되는 데 있어 필수적이라는 것을 깨달았다.