Seoul Software ACademy

웹 개발자 부트캠프 과정

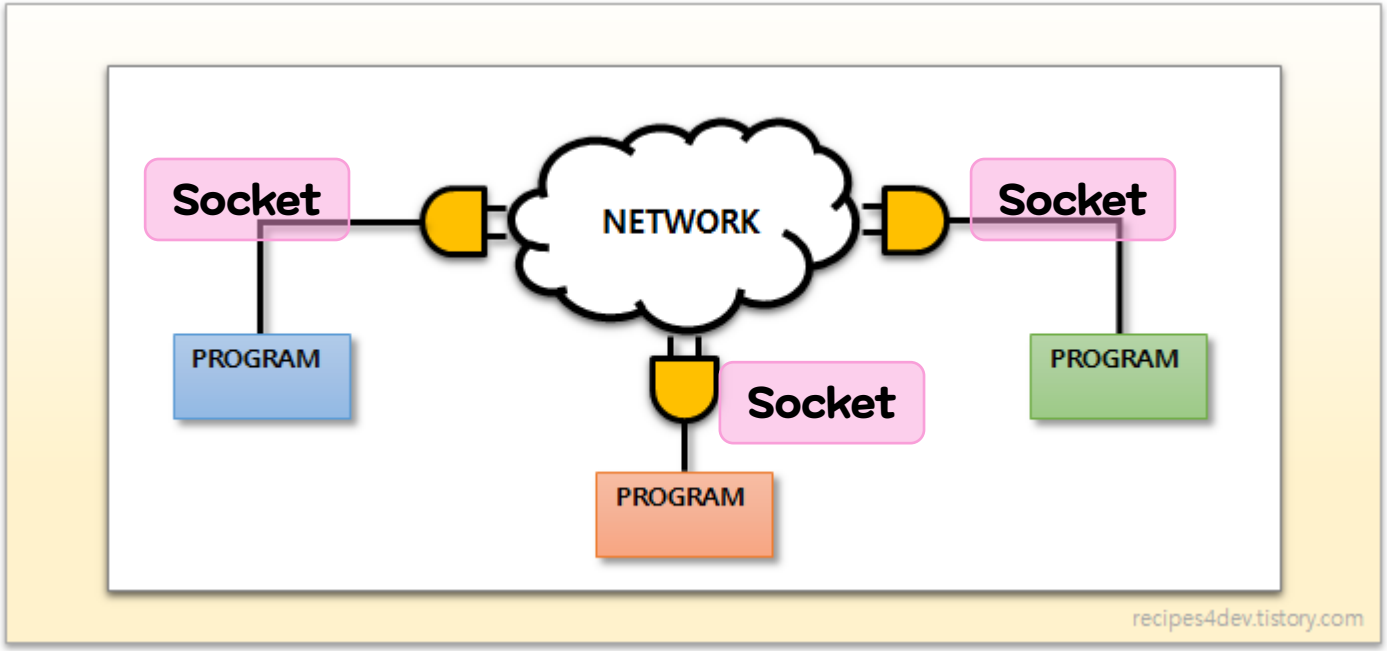
SeSAC x CODINGOn

With. 팀 뤼쳐드

Socket

소켓(Socket)

- 프로세스가 네트워크로 **데이터를 내보내거나 데이터를 받기 위한** 실제적인 창구역할을 하는 것



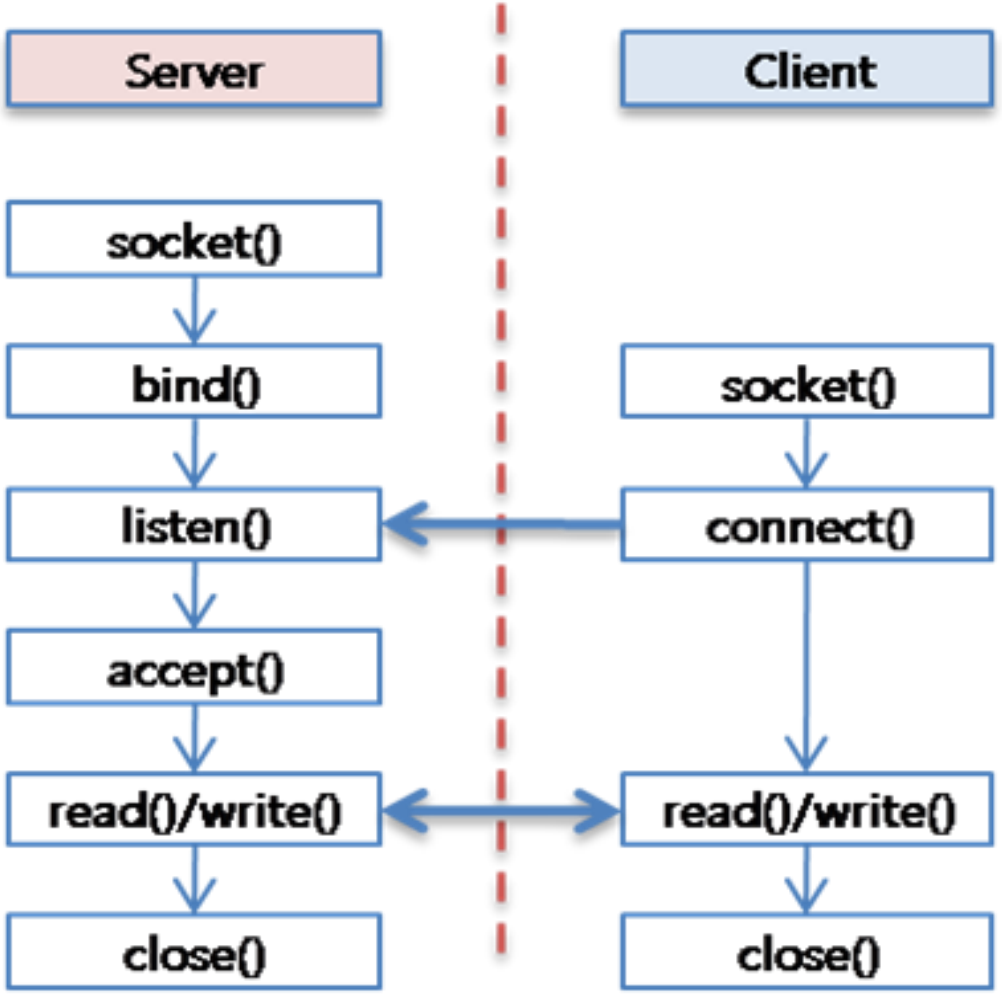
소켓(Socket)

- 서버와 클라이언트를 연결해주는 도구로써 인터페이스 역할을 하는 것
  - 서버 : 클라이언트 소켓의 연결 요청을 대기하고, 연결 요청이 오면 클라이언트 소켓을 생성해 통신을 가능하게 하는 것
  - 클라이언트 : 실제로 데이터 송수신이 일어나는 곳
- 소켓은 프로토콜, IP 주소, 포트 넘버로 정의된다.
- TCP와 UDP 프로토콜을 사용하여 데이터를 전송

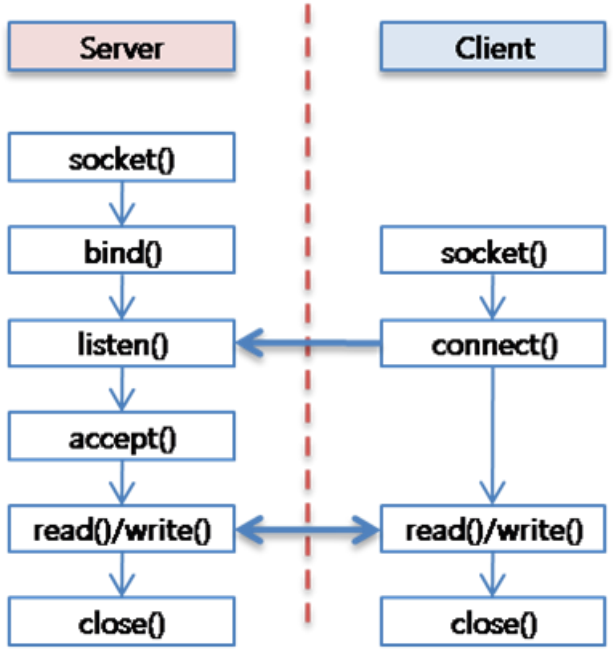
소켓 프로그래밍

- 서버 소켓
  - 서버 소켓은 연결 요청을 받아들이는 역할.
  - 클라이언트의 요청을 받아들여 실제 통신을 위한 소켓을 생성.
- 클라이언트 소켓
  - 클라이언트 소켓은 서버에 연결을 요청하고, 연결이 수락되면 서버와 데이터를 주고받을 수 있는 소켓.
- 포트
  - 컴퓨터 내에서 소프트웨어 간에 통신을 할 때 사용되는 식별자.
  - 포트를 이용하여 특정 소켓을 찾고 연결.

소켓 프로그래밍 흐름

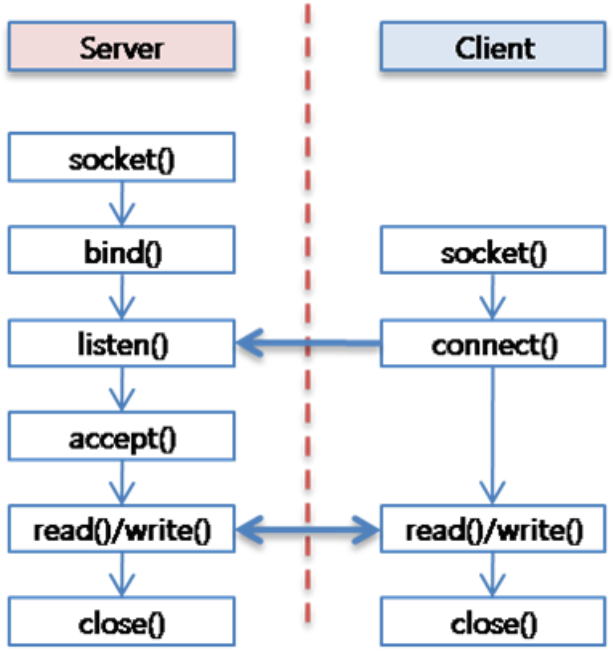

Server
Client
socket()
bind()
listen()
accept()
read()/write()
close()
socket()
connect()
read()/write()
close()

소켓 프로그래밍 흐름



- 서버(Server)
  - **socket()** : Socket 생성 함수
  - **bind()** : ip와 port 번호 설정 함수
  - **listen()** : 클라이언트의 요청에 수신 대기열을 만드는 함수
  - **accept()** : 클라이언트와의 연결을 기다리는 함수

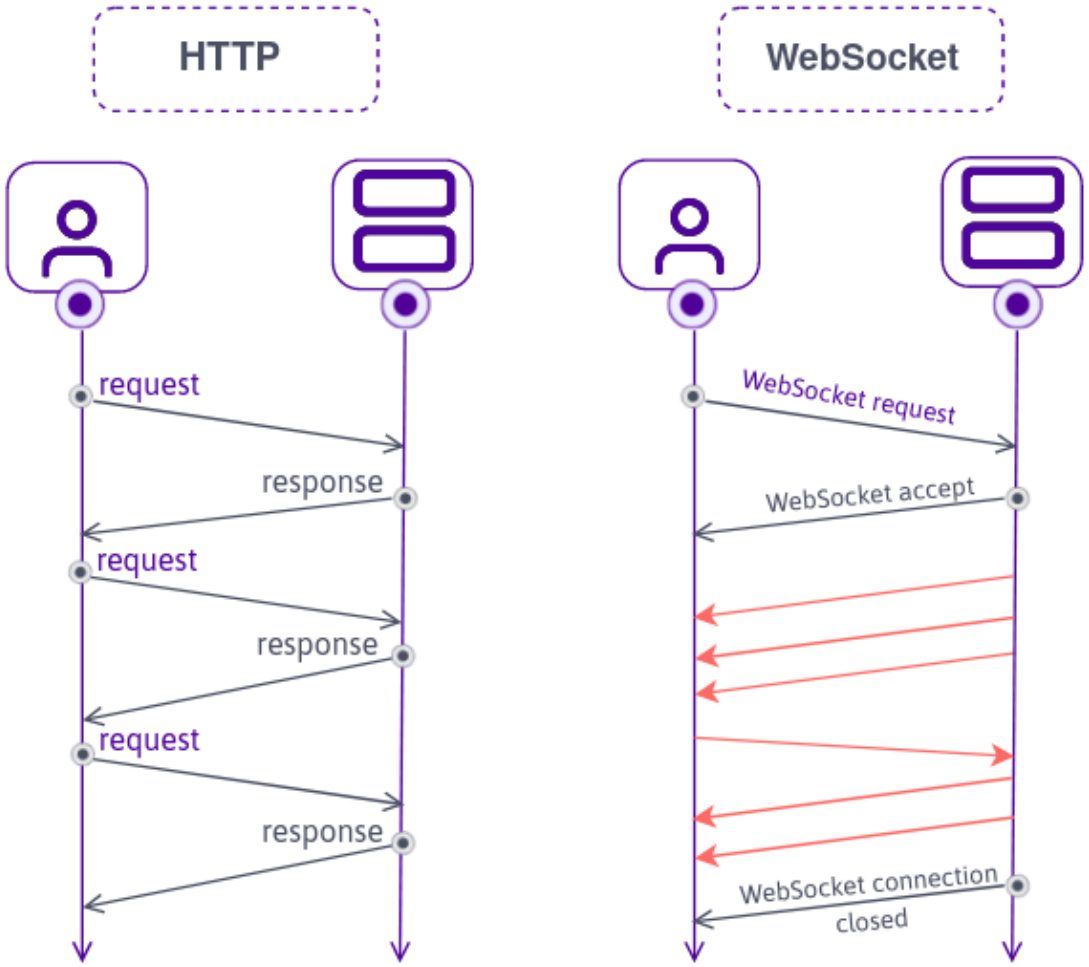
소켓 프로그래밍 흐름



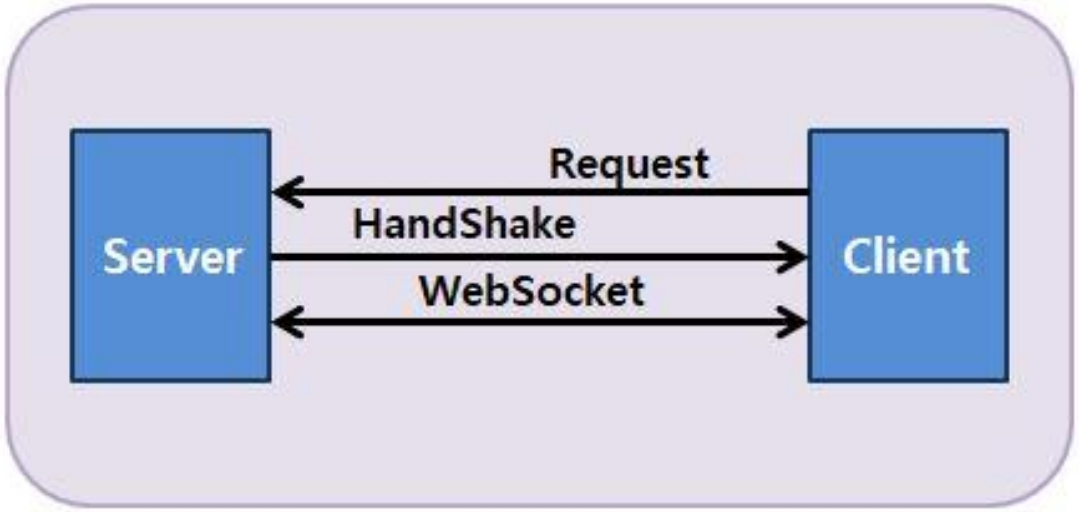
- 클라이언트(client)
  - **socket()** : 소켓을 여는 함수
  - **connect()** : 통신할 서버의 설정된 ip와 port 번호에 통신을 시도하는 함수
  - 통신 시도 시, 서버가 accept()함수를 이용해 클라이언트의 socket descriptor를 반환
  - 이를 통해 클라이언트와 서버가 서로 read() write()를 반복하며 통신

WebSocket

HTTP vs WebSocket



WebSocket이란?



양방향 소통을 위한 프로토콜(약속)

- HTML5 웹 표준 기술
- 빠르게 작동하며 통신할 때
  아주 적은 데이터를 이용
- 이벤트를 단순히 듣고, 보내는 것만 가능
- Handshake(핸드셰이크): 클라이언트가 서버로 웹소켓 연결을 요청할 때, 서버와 클라이언트 간에 초기 핸드셰이크가 이루어며 이 핸드셰이크 과정을 통해 웹소켓 연결
- 클라이언트 측에서는 브라우저의 WebSocket 객체를 사용하여 웹소켓 연결을 생성하고 관리

WebSocket 이벤트

- open: 웹소켓 연결이 성공적으로 열렸을 때 발생
- message: 웹소켓을 통해 데이터를 주고받을 때 발생
- error: 웹소켓 연결 중 오류가 발생했을 때 발생. 연결 실패, 통신 오류 등이 해당
- close: 웹소켓 연결이 종료되었을 때 발생

클라이언트

```
<script>
    const socket = new WebSocket("ws://localhost:8000");

    socket.addEventListener("open", (event) => {
        console.log("서버에 연결되었습니다.");
    });

    socket.addEventListener("message", (event) => {
        console.log(`서버로부터 받은 메시지: ${event.data}`);
    });

    socket.addEventListener("error", (event) => {
        console.error("오류가 발생했습니다:", event.error);
    });

    socket.addEventListener("close", (event) => {
        console.log("서버와 연결이 종료되었습니다.");
    });
</script>
```

백엔드

브라우저 환경에서는 WebSocket API를 사용하여 웹소켓 클라이언트를 만들 수 있지만, 서버를 만들려면 별도의 라이브러리나 모듈이 필요

```
npm install ws
```

https://www.npmjs.com/package/ws

ws 모듈 이벤트

- connection: 클라이언트가 웹소켓 서버에 연결되었을 때 발생. 이 이벤트의 콜백 함수는 새로운 클라이언트 연결마다 실행
- message: 클라이언트로부터 메시지를 받았을 때 발생
- error: 웹소켓 연결 중 오류가 발생했을 때 발생
- close: 클라이언트와의 연결이 종료되었을 때 발생

백엔드

```
const ws = require("ws");
const PORT = 8080;

const wss = new ws.Server({ port: PORT });

wss.on("connection", (socket) => {
    console.log("클라이언트가 연결되었습니다.");

    socket.on("message", (message) => {
        console.log(`클라이언트로부터 받은 메시지: ${message}`);
        // 클라이언트로 응답 메시지 전송
        socket.send(`서버에서 받은 메시지: ${message}`);
    });
    socket.on("error", (error) => {
        console.error("오류가 발생했습니다:", error);
    });
    socket.on("close", () => {
        console.log("클라이언트와 연결이 종료되었습니다.");
    });
});
```

Socket.io

Socket.io

- 클라이언트와 서버 간의 짧은 지연 시간, 양방향 및 이벤트 기반 통신을 실시간으로 가능하게 하는 라이브러리
- WebSocket 프로토콜 위에서 구축되었으며 통신 과정을 단순화하고 개선하기 위한 추가 기능을 제공
- 특징
  - 1) 이벤트 기반
  - 2) 자동 재연결

socket.io

https://socket.io/

기본 이벤트

- connection
  - 클라이언트가 서버에 연결되었을 때 발생.
  - 클라이언트와의 상호작용을 초기화하거나 초기 데이터를 전달할 수 있음
- disconnect: 클라이언트가 연결을 해제했을 때 발생
- disconnecting: 클라이언트가 연결을 해제하려는 경우에 발생
- error: 연결 중에 오류가 발생했을 때 발생

사용자 지정 이벤트

socket.emit(eventName[, ...args][, ack])

```
socket.emit("hello", "world", (response) => {
  console.log(response); // "got it"
});
```

이벤트를 보내는 쪽

```
socket.on("hello", (arg, callback) => {
  console.log(arg); // "world"
  callback("got it");
});
```

이벤트를 받는 쪽

룸 관련 함수

```
//클라이언트를 특정 방으로 추가
socket.join("roomA");
//클라이언트를 특정 방에서 제거
socket.leave("roomA");
//특정 방에 속한 모든 클라이언트에게 특정 이벤트와 데이터를 발송
io.to("roomA").emit("message", "Hello, Room A!");
//특정 방에 속한 클라이언트들에게만 특정 이벤트와 데이터를 발송.
//이때 이벤트는 송신자 클라이언트를 제외한 다른 클라이언트들에게 전달
socket.to("roomA").emit("message", "Hello, Room A!");
socket.broadcast.to("roomA").emit("message", "A new user has joined the room.");
//송신자를 제외한 서버에 연결된 모든 클라이언트에게 이벤트와 데이터를 발송
socket.broadcast.emit("message", "A new user has joined the room.");
//송신자를 포함하여 모든 클라이언트에게 이벤트와 데이터를 발송
io.socket.emit("message", "Hello, Room A!");
//특정 방에 속한 클라이언트들의 정보를 확인
const roomInfo = io.sockets.adapter.rooms.get(room);
console.log(roomInfo); // 방에 속한 클라이언트 id정보가 출력됩니다.
```

Socket.io 실습

Socket.io 불러오기 (server)

```
npm install socket.io
```

```
const http = require("http");
const express = require("express");
const SocketIO = require("socket.io");
const app = express();
const PORT = 8000;
const server = http.createServer(app);
const io = SocketIO(server);

//...

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

Socket.io 불러오기 (client)

localhost:8000/socket.io/socket.io.js

서버가 연결되면 브라우저를 통해 소켓io 파일에 접근이 가능

```
<script src="/socket.io/socket.io.js"></script>
```

```
const socket = io();
```

io() 함수는 클라이언트 소켓을 생성하고, 서버와의 연결을 설정

실습1 Socket 연습

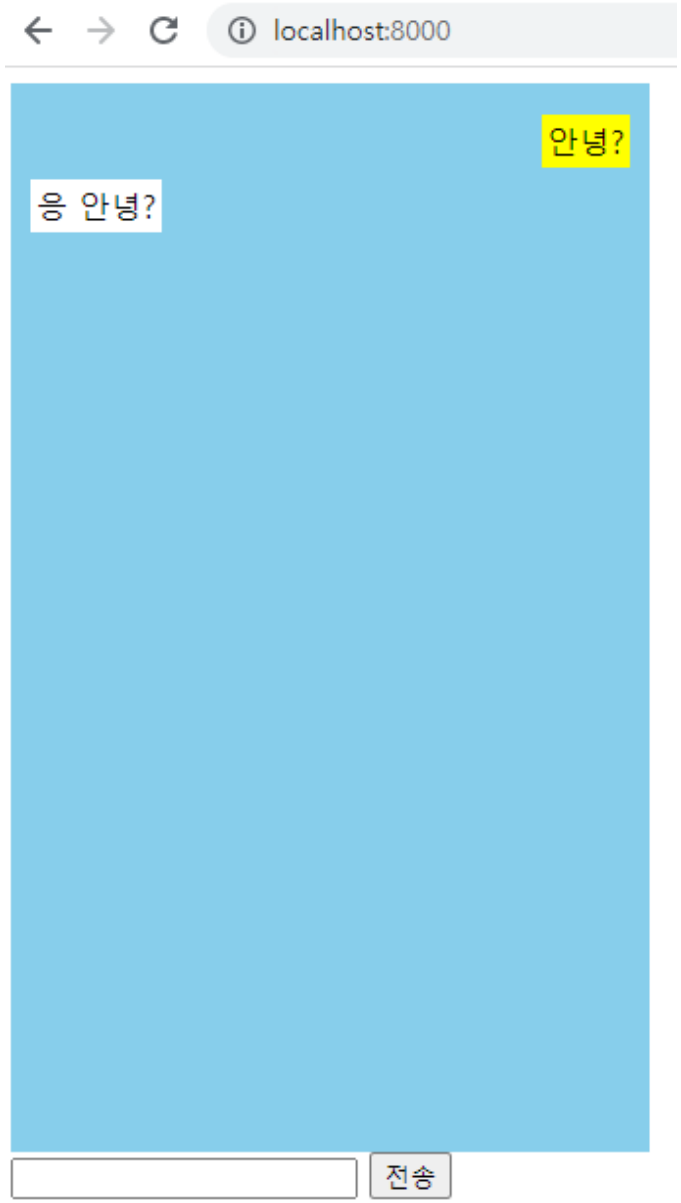
- 각 버튼을 누를 때마다 서버로 메시지 전송
- 서버
  - **클라이언트로부터 받은 메시지**를 console 에 찍고, 그에 대한 응답을 보내기
- 클라이언트
  - 버튼을 누를 때 서버로 메시지 보내기
  - **서버로부터 받은 응답 메시지**를 화면에 보여주기


```
// 이벤트 명을 지정하고 메세지를 보냄
socket.emit('전송할 이벤트 명', msg);

// 해당 이벤트를 받고 콜백함수를 실행
socket.on('받을 이벤트 명', (msg) => { ... });
```

실습2 채팅창 UI 만들기

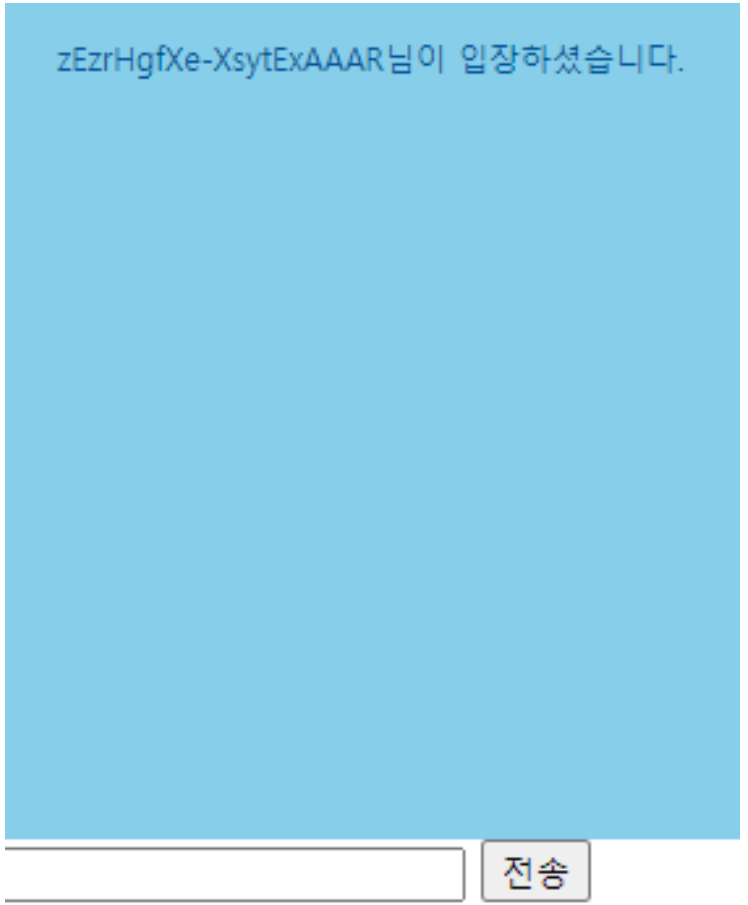
- 내가 보낸 채팅은 오른쪽에
- 다른 사람이 보낸 채팅은 왼쪽에 표시될 수 있도록



실습3 채팅창 입장 안내 문구

- Socket에 담겨 있는 id 정보를 이용하여 OOO 님이 입장했습니다. 라는 메시지 보여주기

```
io.on( "connection", function ( socket ){
    // 최초 입장했을 때
    console.log( "Server Socket Connected", socket.id );
    io.emit( "notice", `${socket.id}님이 입장하셨습니다.`);
});
```



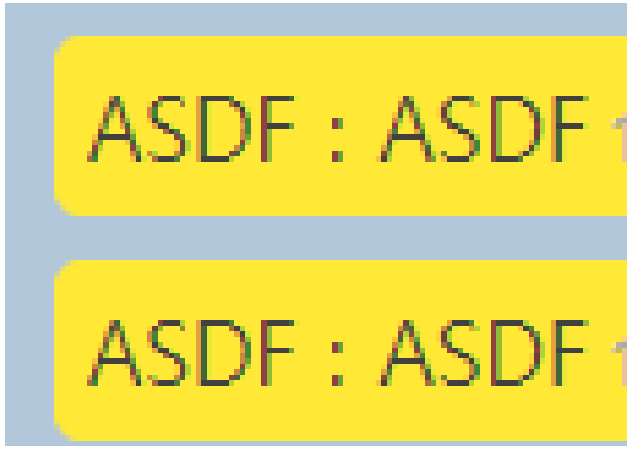
실습3-1. 닉네임 입력 받고 입장 시키기

실습3-2. 닉네임 중복 방지

실습3-3. 퇴장 시키기

실습4 채팅창 메세지 전송

- 메시지에 누가 작성한 메시지인지 작성자 이름 받고 이름 보여주기
- 아래와 같은 형식으로 보여줘도 OK
  - 작성자 : MESSAGE



실습5 DM기능 추가하기

- 특정 사람에게만 메시지를 보낼 수 있도록 DM 기능 추가해보기
- DM 메시지는 일반 메시지와 다르게 ui 상으로 변화 주기