NETWORKING

자바스크립트(JAVASCRIPT) SOCKET.IO를 활용한 채팅 프로그램 만들기

socket.io

- Socket.IO는 실시간 웹 애플리케이션을위한 JavaScript 라이브러리
- 웹 클라이언트와 서버간에 실시간 양방향 통신이 가능
- 브라우저에서 실행되는 클라이언트 측 라이브러리와 Node.js 용 서버 측 라이브러리의 두 부분으로 구성
- Javascript에서 WebSocket을 보다 편하게 사용할 수 있게 개발된 라이브러리

Express

- Node.js로 http 모듈을 사용해 웹 서버에 필요한 기능들을 하나하나 다 구현하면 복잡하고 손이 많이 가는 단점 발생
- 간단하게 처리해주는 것이 Express!
- Express.js는 HTTP 요청에 대해 라우팅과 미들웨어 기능을 제공하는 웹 프레임 워크

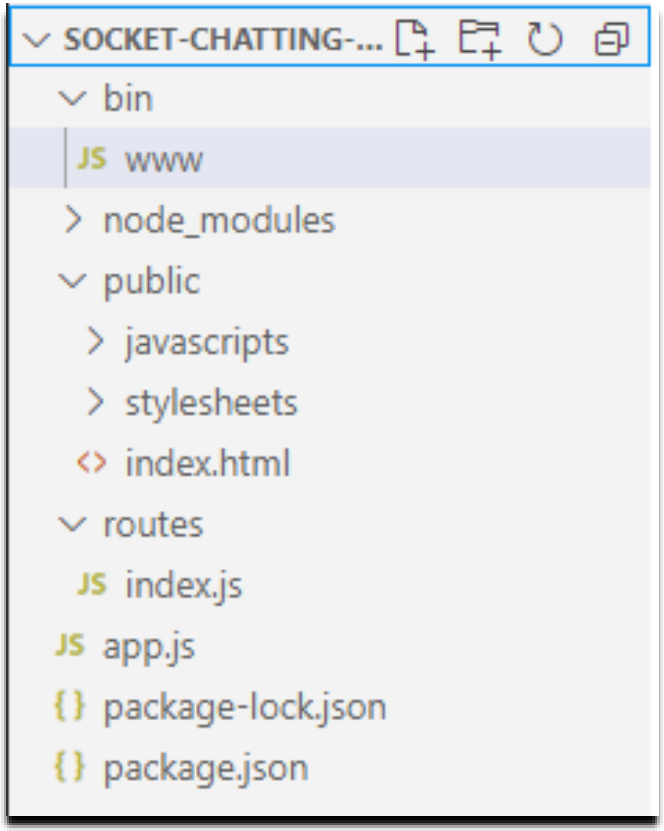
Express-generator

- Express framework는 익스프레스 외에도 여러가지 패키지를 사용해서 구조를 잡는 도중에는 필요한 패키지를 하나하나 찾아서 설치해야하는 번거로움이 존재

- Express-generator라는 패키지를 통해서 프레임워크에 필요한 pakage.json과 기본 구조까지 잡을 수 있음

Express-generator

Express 구조

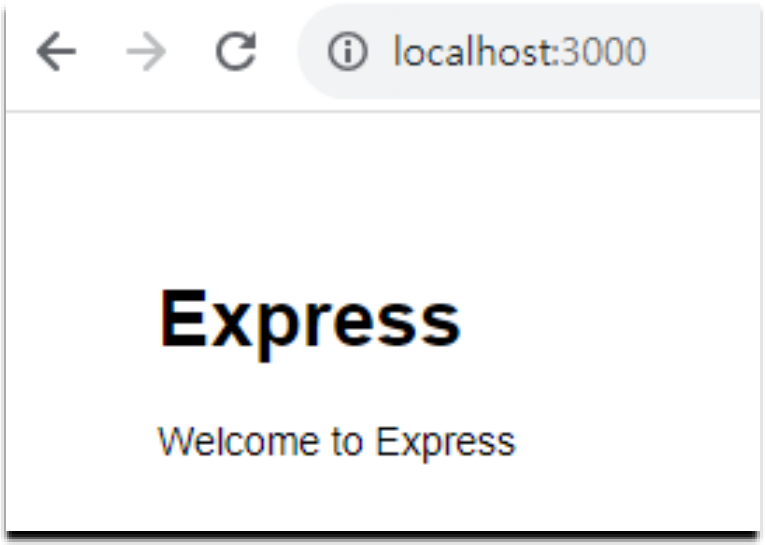


Express-generator

- public
  - 각종 리소스들을 모아놓은 폴더로 외부(브라우저 등의 클라이언트)에서 접근 가능한 파일들을 모아 둔 디렉토리이다. images, javascripts, stylesheets 파일들이 존재
- routes
  - 라우터들을 관리하는 곳으로 서버의 로직은 모두 routes 폴더 안의 파일에 작성
  - Index.js를 기반으로 라우팅 관리
  - routes 디렉토리 안에 또 폴더를 만들어 관리하던지 파일을 만들어 관리. 다만 index.js가 루트가 되게!
- views
  - view 파일들을 관리하는 곳으로 웹서버로 사용 시 이 디렉토리에 잇는 파일들을 사용해서 렌더링 시킴
- app.js
  - 핵심적인 서버의 역할을 하며 미들웨어 관리를 하는 곳
- pakage.json
  - 프로젝트의 이름, 버전, 의존 패키지 리스트 등의 정보를 담고 있는 파일

Express-generator

- C:₩Users₩dw> npm install
- C:₩Users₩dw> npm start



- C:₩Users₩dw> npm install --save socket.io

HTML 파일 작성

```
//public/index.html

<html>
    <head>
    <title>DW-Academy</title>
    <link rel="stylesheet" href="/stylesheets/style.css">
    </head>

    <body>
    <ul id="messages"></ul>
        <form action="" id="chat-form">
        <input id="m" autocomplete="off" /><button>Send</button>
        </form>
    </body>
</html>
```

CSS 파일 작성

```
//public/stylesheets/style.css

* { margin: 0; padding: 0; box-sizing: border-box; }
body {
  font: 13px Helvetica, Arial;
  background-color: #CEECF5;
}
form {
  background: #eee;
  padding: 3px;
  position:
  fixed;
  bottom: 0;
  width: 100%;
}
form input {
  border: 0;
  border-radius: 5px;
  padding: 10px;
  width: 90%;
  margin-right:
  0.5%;
}
form button {
  width: 9%;
  border-radius: 5px;
  background: #ffd600;
  border: none;
  padding: 10px;
}
#messages {
  list-style-type: none;
  margin: 0;
  padding: 0;
}
```

Server Code 작성

```
//bin/www
const app = require('../app');
const debug = require('debug')('socket-example:server');
const http = require('http');

const port = normalizePort(process.env.PORT || '4991');
app.set('port', port);

const server = http.createServer(app);
const io = require('socket.io')(server);

io.on('connection', (socket) => {
  socket.on('chat message', (message) => {
    socket.broadcast.emit('chat message', message);
  });
});

server.listen(port);
server.on('error', onError);
server.on('listening', onListening);

...
```

Client Code 작성

- public/javascripts 폴더에 socket.js라는 파일을 추가
- public/index.html을 수정

```
<html>

<head>
  <title>DW-Academy</title>
  <link rel="stylesheet" href="/stylesheets/style.css">
  <script defer src="/socket.io/socket.io.js"></script>
  <script defer src="javascripts/socket.js"></script>
</head>

<body>
  <div id="messages"></div>
    <form action="" id="chat-form">
      <input id="m" autocomplete="off" /><button>Send</button>
    </form>
</body>

</html>
```

```
//public/javascripts/socket.js

const socket = io();
```

Client Code 작성

- <script defer src="/socket.io/socket.io.js"></script>
- socket.io-client을 불러오는 코드
- io를 global하게 사용할 수 있게 함
- socket.js에서 io()를 통해 socket을 초기화 할 수 있음

- 작성 후 서버를 다시 실행해보면 터미널에
- a user connected가 뜨는 것을 확인할 수 있음
  - 클라이언트와 서버간 소켓이 연결 성공

```
C:\Users\dw>npm start

> dw@0.0.0 start
> node ./bin/www

a user connected
```

채팅 기능 구현

- 서버쪽과 클라이언트 쪽에서 필요한 이벤트를 emit/on 함
- emit은 이벤트를 발생시키는 것이고, on은 발생한 이벤트를 받는 부분

```
//bin/www

const app = require('../app');
const debug = require('debug')('socket-example:server');
const http = require('http');

const port = normalizePort(process.env.PORT || '3000');
app.set('port', port);

const server = http.createServer(app);
const io = require('socket.io')(server);

io.on('connection', (socket) => {
  socket.on('chat message', (message) => {
    io.emit('chat message', message);
  });
});

server.listen(port);
server.on('error', onError);
server.on('listening', onListening);

...
```

서버쪽에서는 chat message라는 이벤트를 클라이언트에게 받고 이를 다시 클라이언트에게 알려주는 코드가 추가

채팅 기능 구현

```
//public/javascripts/socket.js

const socket = io();

const chatForm = document.getElementById('chat-form');
const chatBox = document.getElementById('messages');

chatForm.addEventListener('submit', (e) => {
    e.preventDefault();
    const message = e.target.m.value
    socket.emit('chat message', message);
    e.target.m.value = '';
})

socket.on('chat message', (message) => {
    chatBox.appendChild(makeMessage(message));
})

const makeMessage = (message) => {
    const msgBox = document.createElement('div');
    msgBox.className = "message-wrapper";
    msgBox.innerText = message;
    return msgBox;
}
```

채팅 기능 구현

- 클라이언트쪽에서는 필요한 DOM에 접근하여 이벤트를 할당
- chatForm에 이벤트 리스너를 등록
- submit이 발생할 때 마다 socket.emit('chat meesage', meessage)를 통해 서버로 이벤트 발생 전달

- 또한 socket.on('chat message')를 통해 서버가 해당 이벤트를 emit했을 경우 설정해놓은 callback 함수를 동작

- 여기서는 chatBox에 받은 message를 추가하는 함수가 실행

Broadcast 기능 및 CSS 적용

- 대부분 대화방 또는 채팅을 치면 본인과 상대방이 대화 내용이 다르게 보이는 경우가 대부분 (예를 들면 카카오톡)
- broadcast를 활용하여 이를 구현
- 현재까지의 로직은 서버에서 연결된 모든 socket에 이벤트를 알리는 방식
- broadcast를 사용하면 자신을 제외한 나머지 클라이언트에게 이벤트를 전달
- 자신의 메세지는 클라이언트에서 바로 처리하게 하고
- 다른 사람의 메세지는 서버에서 받은 후 처리하는 방식

Broadcast 기능 및 CSS 적용

```
//public/javascripts/socket.js

const socket = io();

const chatForm = document.getElementById('chat-form');
const chatBox = document.getElementById('messages');

chatForm.addEventListener('submit', (e) => {
    e.preventDefault();
    const message = e.target.m.value
    socket.emit('chat message', message);
    e.target.m.value = '';
    // chatBox.appendChild(makeMessage(message, false));
})

socket.on('chat message', (message) => {
    chatBox.appendChild(makeMessage(message, true));
})

const makeMessage = (message, isOthers) => {
    const msgBox = document.createElement('div');
    const classname = isOthers ? "others-message-wrapper" : "my-message-wrapper";
    msgBox.className = classname;
    msgBox.innerText = message;
    return msgBox;
}
```

Broadcast 기능 및 CSS 적용

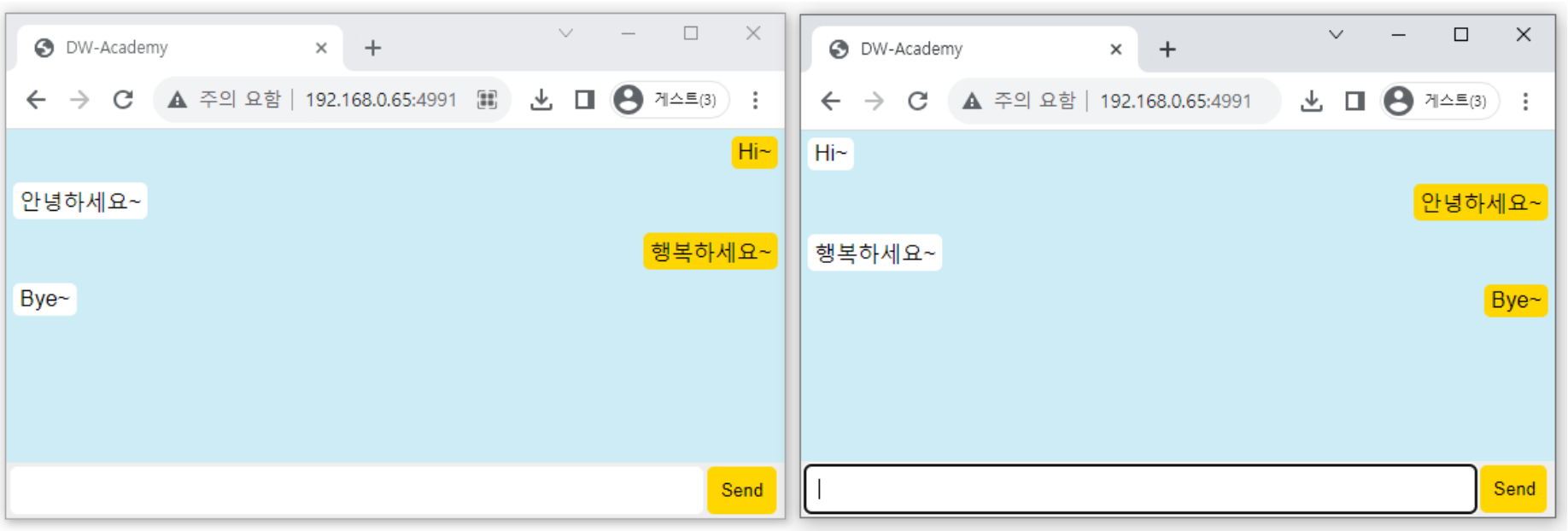
```
//bin/www

const app = require('../app');
const debug = require('debug')('socket-example:server');
const http = require('http');

const port = normalizePort(process.env.PORT || '4991');
app.set('port', port);

const server = http.createServer(app);
const io = require('socket.io')(server);

io.on('connection', (socket) => {
  socket.on('chat message', (message) => {
    socket.broadcast.emit('chat message', message);
  });
});

...
```

Broadcast 기능 및 CSS 적용

```
//public/stylesheets/style.css

...

#messages {
  display: flex;
  flex-direction: column;
  list-style-type: none;
  margin: 0;
  padding: 0;
}
```

```
.others-message-wrapper {
  max-width: 70%;
  align-items: flex-start;
  background-color: white;
  border-radius: 5px;
  padding: 3px 5px;
  margin: 5px auto 5px 5px;
}

.my-message-wrapper {
  max-width: 70%;
  align-items: flex-end;
  background-color: #ffd600;
  border-radius: 5px;
  padding: 3px 5px;
  margin: 5px 5px 5px auto;
}
```

Broadcast 기능 및 CSS 적용



질의 응답

- Q & A ???