

# 프로젝트 명세서

웹소켓(WebSocket) 통신 실습

## 목차

1. 과제 개요 .....	3
2. 참고 자료 .....	5
3. 기능 명세 .....	6
4. 산출물 제출 .....	13

## 1. 과제 개요

---

기본적으로 HTTP 통신은 클라이언트가 요청(Request)을 보내면 서버가 응답(Response)하는 구조입니다. 이로 인해 서버는 능동적으로 클라이언트에 먼저 데이터를 전송할 수 없습니다. 클라이언트는 서버의 데이터를 얻기 위해 항상 요청을 먼저 해야 하고, 서버는 이에 수동적으로 응답해주는 구조로 구성되어 있습니다.

이에 반해 웹소켓(WebSocket) 통신은 클라이언트와 서버를 연결하고 실시간으로 통신이 가능하도록 하는 기술입니다. 앞서 설명한 HTTP와는 다르게 별도의 요청을 보내지 않고도 데이터를 수신할 수 있습니다. 요청을 보낼 필요없이 바로 메시지를 수신할 수 있습니다. 지속적으로 업데이트되는 정보를 수신해야 하는 채팅이나 주식 보고서에서 웹 소켓 통신을 사용하는 이유입니다. 이 프로토콜은 정보를 동시에 송수신할 수 있으므로 정보 교환이 더 빨라집니다.

웹소켓에서 클라이언트와 서버 간의 연결은 둘 중 하나에 의해 종료되거나 시간 초과에 의해 닫힐 때까지 열린 상태로 유지됩니다. 그리고 연결이 종료될 때까지 동일한 채널을 사용하여 통신이 수행됩니다. 메시지는 양방향으로 교환되며, 데이터를 암호화할 수 있습니다.

웹소켓 통신은 실시간 데이터 업데이트와 클라이언트 메시지를 보내는 기능이 필요로 할 때 주로 사용됩니다. 다음은 웹소켓 통신 사용 사례입니다.

- 교환 플랫폼
- 게임 애플리케이션
- 챗봇
- 푸시 알림
- 소셜 네트워크
- 채팅 애플리케이션
- IoT 애플리케이션

이처럼 웹소켓 통신은 다양한 곳에서 사용되고 있습니다. 본 과제는 이러한 웹소켓 통신을 이해하고, 간단히 실습해보는 과제입니다.

## 2. 참고 자료

본 과제를 진행하기 위해서는 먼저 웹소켓 통신을 주고 받을 서버와 클라이언트가 있어야 합니다. 이를 위해 간단하고 빠르게 서버를 구현할 수 있는 Node.js 의 Express.js 프레임워크를 사용하도록 하겠습니다.

Node.js 는 자바스크립트(Javascript) 언어로 프론트엔드(Frontend)뿐만 아니라 백엔드(Backend) 개발 환경을 구성할 수 있기 때문에 생산성이 높고 러닝 커브를 줄일 수 있습니다. 빠르게 개발 환경을 구성하여 개발을 해야 하는 경우 매우 유용합니다. 이러한 이유로 Node.js 를 이용하여 쉽게 서버를 구성할 수 있는 Express.js 프레임워크를 이용하여 실습하고자 합니다.

### 과제를 위한 참고 자료

구분	제목	링크
이해	Node.js	<a href="https://nodejs.org">https://nodejs.org</a>
이해	Express.js	<a href="https://expressjs.com">https://expressjs.com</a>
이해	ws (WebSocket Library)	<a href="https://www.npmjs.com/package/ws">https://www.npmjs.com/package/ws</a>
이해	HTTP 에서부터 WebSocket 까지	<a href="https://url.kr/a3coyv">https://url.kr/a3coyv</a>

### 3. 기능 명세

#### 1. 기능/과제 목록

Req.	Category
1	웹소켓 개발 환경 구축 (Node.js 와 Express.js 프레임워크를 이용한 서버 구축)
2	웹소켓 통신을 이용한 단방향 데이터 전송 (클라이언트에서 서버로)
3	웹소켓 통신을 이용한 양방향 데이터 전송

#### 2. 기능/과제 상세

##### Req. 1. 웹소켓 개발 환경 구축 (Node.js 와 Express.js 프레임워크를 이용하여 서버 구축)

Req. 1-1	Node.js 및 Express.js 프레임워크 설치
기능 상세	<p>본 과제에서는 앞서 말씀 드린대로 Node.js 와 Express.js 프레임워크를 이용하여 서버를 구축할 예정입니다.</p> <p>1. Node.js 설치</p> <p>아래 사이트에서 운영체제에 맞는 설치파일을 다운 받아 설치합니다. (명세서에서는 18.16.0 LTS 버전을 사용하도록 하겠습니다.)</p> <ul style="list-style-type: none"> <li>- <a href="https://nodejs.org/en/download">https://nodejs.org/en/download</a></li> </ul> <p>설치 완료 후 사용하는 터미널 프로그램에서 'node -v'를 입력하여 다운받아 설치한 node.js 가 정상적으로 설치되었는지 확인합니다.</p> <pre>&gt; node -v v18.16.0</pre> <p>2. Express.js 설치</p> <p>Express.js 는 npm 을 이용해서 설치합니다. npm 은 Node.js 패키지 매니저로 Node.js 로 만들어진 모듈을 웹에서 받아 설치하고 관리해주는 프로그램입니다.</p> <p>먼저 과제를 진행할 폴더를 생성해주시고, 터미널 프로그램에서 해당 폴더로 이동 후 'npm init' 명령을 실행합니다. 해당 명령어를 통해 앞으로 설치될 패키지를 관리할 package.json 파일을 생성합니다.</p>

	<pre> &gt; npm init This utility will walk you through creating a package.json file. It only covers the most common items, and tries to guess sensible defaults.  See 'npm help init' for definitive documentation on these fields and exactly what they do.  Use 'npm install &lt;pkg&gt;' afterwards to install a package and save it as a dependency in the package.json file.  Press ^C at any time to quit. package name: (test_websocket) version: (1.0.0) description: entry point: (index.js) test command: git repository: keywords: author: license: (ISC) About to write to D:\03_Personal\test_websocket\package.json:  {   "name": "test_websocket",   "version": "1.0.0",   "description": "",   "main": "index.js",   "scripts": {     "test": "echo \"Error: no test specified\" &amp;&amp; exit 1"   },   "author": "",   "license": "ISC" }  Is this OK? (yes) </pre> <p>이후 'npm install express' 명령을 터미널에 입력하면 Express.js 가 설치됩니다.</p> <pre> &gt; npm install express  added 58 packages, and audited 59 packages in 927ms  8 packages are looking for funding   run `npm fund` for details  found 0 vulnerabilities </pre> <p>정상적으로 설치되었다면, package.json 파일에 다음과 같이 설치된 express.js 버전이 기록됩니다.</p> <pre> "dependencies": {   "express": "^4.18.2" } </pre>
Req. 1-2	Node.js 의 Express.js 프레임워크를 이용한 서버 구동
기능 상세	<p>간단한 클라이언트와 서버 코드를 작성하여 웹서버 구동 및 웹소켓 통신 구현을 위한 환경을 구축해보겠습니다.</p> <p>먼저 클라이언트 코드를 작성해보겠습니다. 과제 루트 디렉토리에서 'front' 라는 폴더를 생성하고 해당 폴더 내에 'index.html' 파일을 작성합니다. 아래 코드는 간단한 예제이며, 본인이 원하는 대로 코드를 수정하여 진행합니다.</p>

```
<!DOCTYPE html>
<html lang="ko">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-
scale=1.0" />
    <title>Websocket 실습</title>
  </head>
  <body>
    <h1>WebSocket 실습 과제</h1>
  </body>
</html>
```

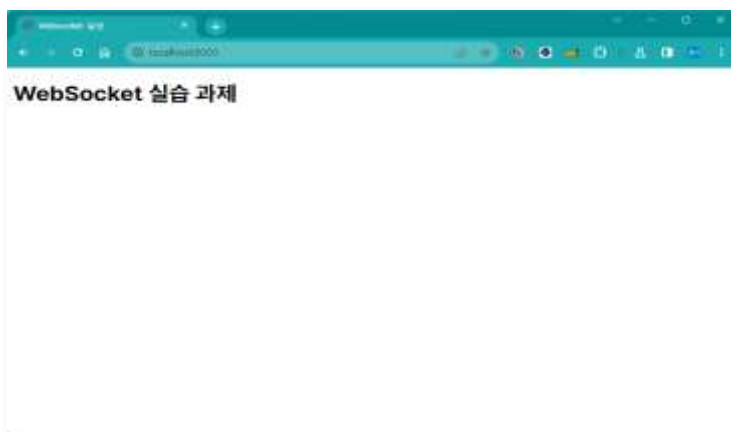
다음으로 Node.js 서버를 생성해보겠습니다. 과제 루트 디렉토리에 'index.js' 파일을 생성합니다. 일단 앞서 작성한 html 파일을 서빙해 주도록 하겠습니다.

```
const express = require("express")
const app = express()

app.use(express.static("front"))

app.listen(8000, () => {
  console.log(`Server listening on port 8000`)
})
```

작성 완료 후 터미널에서 과제 루트 디렉토리로 이동 후 'node index.js'를 입력하여 실행하면 서버가 구동됩니다. 그리고 크롬 브라우저에서 'localhost:8000'으로 접속하면 다음과 같이 앞서 작성했던 화면을 확인할 수 있습니다.





	이로써 간단하게 Node.js 와 Express.js 프레임워크를 이용하여 웹소켓 통신 개발을 위한 서버 환경을 구축해 보았습니다.
--	---

## Req. 2. 웹소켓을 이용한 단방향 데이터 전송 (클라이언트에서 서버로)

Req. 2-1	웹소켓 통신을 위한 ws 라이브러리 설치
기능 상세	<p>Express.js 프레임워크 설치 방법과 동일하게 npm 을 이용하여 ws 웹소켓 라이브러리를 설치하겠습니다. 터미널에서 과제 루트 디렉토리로 이동후 'npm install ws' 명령어를 입력하면 ws 라이브러리가 설치됩니다.</p> <pre>&gt; npm install ws added 1 package, and audited 60 packages in 540ms 8 packages are looking for funding   run `npm fund` for details found 0 vulnerabilities</pre> <p>역시나 package.json 파일을 확인해보면 다음과 같이 ws 라이브러리가 추가되어 있는 것을 확인하실 수 있습니다.</p> <pre>"dependencies": {   "express": "^4.18.2",   "ws": "^8.13.0" }</pre>
Req. 2-2	ws 라이브러리를 이용하여 클라이언트에서 서버로 메시지 전송
기능 상세	<p>앞서 작성한 index.js 파일에 ws 모듈을 불러온 후 포트 8001 번을 통해서 접속할 수 있는 웹소켓 서버를 열어주도록 코드를 추가합니다. 그리고 클라이언트로부터 전달받은 메시지를 콘솔 로그로 출력하여 정상적으로 수신하였는지 확인 할 수 있도록 합니다.</p> <pre>const express = require("express") const { WebSocketServer } = require("ws") const app = express()  app.use(express.static("front"))  app.listen(8000, () =&gt; {   console.log(`Server listening on port 8000`)</pre>

```

})

const wss = new WebSocketServer({ port: 8001 })

wss.on("connection", ws => {
  ws.on("message", data => {
    console.log(`Received from client: ${data}`)
  })
})

```

front 폴더내 index.html 파일을 수정하여 클라이언트에서 버튼을 만들고 만든 버튼을 클릭했을 때 'Hello'라는 메시지를 WebSocket 통신을 통해 서버로 전송하도록 코드를 추가해 줍니다.

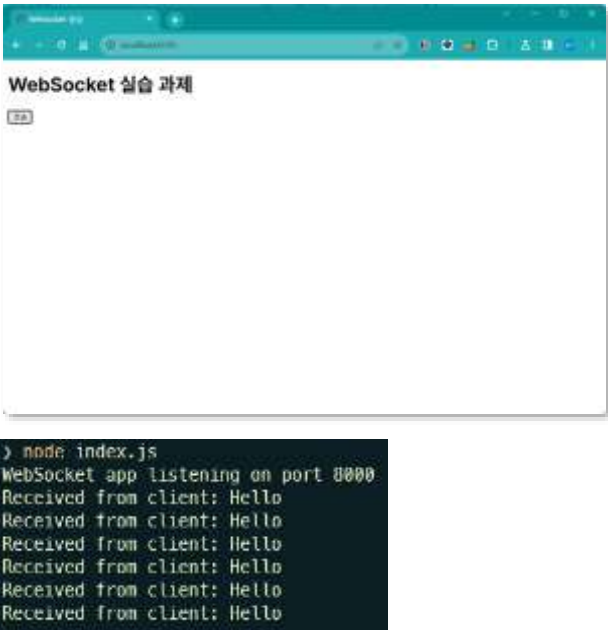
```

<!DOCTYPE html>
<html lang="ko">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Websocket 실습</title>
    <script>
      const ws = new WebSocket("ws://localhost:8001")

      function sendMsg() {
        ws.send("Hello")
      }
    </script>
  </head>
  <body>
    <h1>WebSocket 실습 과제</h1>
    <button onclick="sendMsg()">전송</button>
  </body>
</html>

```

Node.js 서버를 재구동한 후 'localhost:8000'로 접속하면 '전송'이라는 버튼을 확인 할 수 있습니다. 해당 버튼을 클릭하면 'Hello'라는 메시지가 WebSocket 통신을 통해 서버에 전달됩니다. 터미널에서 서버 로그를 확인하면 그림처럼 클라이언트로부터 'Hello'라는 메시지를 받았는 로그를 확인할 수 있습니다.

	
--	--

### Req. 3. 웹소켓 통신을 이용한 양방향 데이터 전송

Req. 3-1	웹소켓 통신을 이용한 간단한 채팅 기능 구현
기능 상세	<p>간단한 채팅 기능 구현을 위해서는 먼저 서버에서 접속된 모든 클라이언트들에게 메시지를 전송할 수 있는 기능이 필요합니다. 일반적으로 이를 브로드캐스트(Broadcast)라고 하는데, 이 브로드캐스트를 구현하는게 핵심이라고 할 수 있습니다. 이를 이용하여 다음과 같이 서버 접속 및 연결 해제 정보와 접속한 사용자 이름과 메시지를 모든 클라이언트에 전송하는 간단한 채팅 기능을 구현해보도록 합니다.</p> <ol style="list-style-type: none"> <li>1. 사용자 접속 정보 메시지 <ul style="list-style-type: none"> <li>- 새로운 유저 접속시 : "새로운 유저 접속 [현재: X 명]"</li> <li>- 기존 유저 연결 해제시: "유저 연결 해제 [현재: X 명]"</li> </ul> </li> <li>2. 사용자 채팅 메시지 <ul style="list-style-type: none"> <li>- 사용자 이름과 메시지를 입력받아 전송 버튼을 클릭 메시지 전송</li> <li>- 접속되어 있는 모든 클라이언트에 사용자 이름과 메시지 표시</li> </ul> </li> </ol> <p>다음은 예시 화면입니다. 예시 화면을 참고하여 해당 기능을 구현합니다.</p>

## 1. 사용자 접속 정보 및 채팅 메시지 입력시

WebSocket 실습 과제

새로운 유저 접속 [현재: 1명]  
 새로운 유저 접속 [현재: 2명]  
 AAA: 환영합니다.  
 BBB: 반갑습니다.

AAA

## 2. 사용자 연결 해제시

WebSocket 실습 과제

새로운 유저 접속 [현재: 1명]  
 새로운 유저 접속 [현재: 2명]  
 AAA: 환영합니다.  
 BBB: 반갑습니다.  
 유저 연결 해제 [현재: 1명]

AAA

## 4. 산출물 제출

---

[https://lab.ssafy.com/s12-study/seasonal\\_fesw](https://lab.ssafy.com/s12-study/seasonal_fesw) 의 “산출물 제출 가이드” 참조