[스파르타코딩클럽] Node.js 심화반 - 4주차

매 주차 강의자료 시작에 PDF파일을 올려두었어요!

▼ PDF 파일

[수업 목표]

1. 소켓의 기본 동작 원리를 알게 된다.
2. 웹소켓 라이브러리인 socket.io를 이용해 실시간 통신을 할 줄 알게 된다.
3. 소켓을 이용한 기능을 쇼핑몰에 구현한다.

[목차]

모든 토글을 열고 닫는 단축키
Windows : `Ctrl` + `alt` + `t`
Mac : `⌘` + `⌥` + `t`

---

00. 선행지식

이 지식들은 기초 수업 또는 앞선 수업에서 배운 내용이기 때문에 본 수업에서는 자세한 설명을 생략해요!
그러나 기억이 나지 않을 여러분을 위해 아주 간단한 내용이라도 정리해두었어요 😇

▼ TCP란?

- 서버와 클라이언트간 신뢰성 있는 데이터 송수신을 위해 만들어진 프로토콜입니다.
- 연결 지향성 프로토콜이라고도 부릅니다.
- 데이터를 나눠서 보낼수 있으며, 데이터를 받는쪽에서 나눠 받은 데이터를 재조립합니다.
  만약 누락된 데이터가 존재하면 다시 요청해서 받아와 완전한 데이터를 만듭니다.
- TCP로 서버/클라이언트간 연결이 된 경우 데이터를 양방향으로 주고 받을수 있습니다.

- 데이터의 순서가 뒤바뀌는 일 없이 안정적이라 신뢰가 가능합니다.
- UDP에 비해 데이터 송수신 비용(부담)이 크다는 특성을 가졌습니다.
- UDP보다 전송 속도가 느립니다.

▼ UDP란?

- TCP와 다르게 비연결성 프로토콜입니다.
- 데이터를 보내고 제대로 받았는지 확인하지 않아, 데이터가 제대로 도착했는지 보장하지 않는 신뢰도가 비교적 낮습니다.
- 데이터를 순차적으로 보내도 받는 쪽에서는 다른 순서로 전달받을 수 있습니다.
- 데이터를 보내기만 하고 별 다른 처리를 하지 않기 때문에 TCP에 비해 비용(부담)이 적다는 특성을 가졌습니다.
- TCP보다 전송 속도가 빠릅니다.

01. 4주차 오늘 배울 것

▼ 1) 소켓에 대해 학습 및 실습

풍문으로만 들어오던 소켓! 이번 주차 수업에서 이것이 무엇인지 학습하게 됩니다 ☺
그리고 node.js에서 쉽게 소켓을 사용 가능할 수 있도록 구현된 socket.io라는 라이브러리를 사용해봅니다!!

▼ 2) 쇼핑몰에 실시간 구매 알림 기능 구현

그 동안처럼 프론트엔드는 저희가 이미 구현해두었으니 걱정마세요!

02. 소켓이란 무엇인가?

▼ 1) 소켓에 대한 간단한 설명

▼ 소켓의 역할?

현실로 비유하자면 마치 벽에 있는 콘센트 구멍과 비슷합니다!
우리가 전기를 사용하기 위해 반드시 거쳐야 하는 연결부에 해당하죠!

그럼 네트워크에서의 소켓은?
우리가 네트워크에서 데이터를 송수신하기 위해 반드시 거쳐야 하는 연결부에 해당합니다!

▼ 소켓의 종류?

소켓의 역할은 언제나 같지만 종류는 여러가지가 있습니다!
대표적으로 TCP, UDP 프로토콜을 사용하는 2가지의 소켓이 있는데요,

아주 일반적으로는 안정적인 데이터 송수신을 위해 TCP 소켓을 사용하는 경우가 대부분이지만, 일부 패킷이 손실되어도 괜찮거나 빠른 전송 속도가 필요한 경우 UDP 소켓을 사용하기도 합니다.

▼ 패킷이란?

네트워크 소켓이 현실의 콘센트와 비슷하다면, 패킷은 쉽게 말해 콘센트 배선에 흐르는 전기와 비슷합니다.
소켓을 통해 송수신하는 데이터 덩어리 하나가 한개의 패킷이라고 표현합니다.

▼ 2) 웹소켓은 무엇인가?

실시간 웹 서비스를 제공하기 위해 만들어진 Socket이라고 생각하면 됩니다.

최근 Google Docs 등 여러 협업툴이 실시간 공동 편집 기능, 웹 메신저 등에서 많이 사용되는 기술로 최근 점점 많이 사용하는 기술이지만 일부 브라우저들이 웹소켓을 지원하지 않기 때문에 모든 브라우저에서의 동작을 보장하지는 못합니다.

▼ 3) socket.io?

자바스크립트를 사용해 웹소켓을 사용하길 원한다면 가장 많이 사용되는 라이브러리입니다.
그러나 이 라이브러리는 순수한 웹소켓 기술만 이용한 라이브러리가 아닙니다.

위에서 말했듯 웹소켓 기술은 아직 모든 브라우저에서 동작하지는 못하기 때문에, 모든 사용자를 고려해야 하는 경우 실시간성 기능 구현에 어려움이 생기게 됩니다.

이 어려움을 해결하기 위해 socket.io는 웹소켓을 사용할 수 없는 브라우저인 경우 서버에서 데이터를 일정 간격마다 받아오는 polling 기능으로 실시간 기능 구현을 가능케 해줍니다.

▼ 4) socket.io는 웹소켓과 다르다?

그렇습니다! 엄밀히 따지면 socket.io는 웹소켓을 포함하여, 웹소켓을 사용하지 못하는 환경에서도 웹소켓과 비슷하게 사용이 가능하도록 구현해놓은 라이브러리입니다.

그렇기 때문에 socket.io는 웹소켓과 완전히 동일하다고 오해하지 않으시길 바랍니다!

추가 참고 내용: https://d2.naver.com/helloworld/1336

03. socket.io 사용해보기

▼ 1) socket.io 테스트 할 새 프로젝트 세팅

▼ 폴더 생성

이제 socket.io를 테스트해보기 위해 아무것도 없는 새 폴더를 생성해줍니다.

▼ 모듈 설치

```
npm i socket.io -S
```

▼ index.html 파일 추가

생성한 폴더 안에 `index.html` 파일을 생성하고, 아래의 코드스니펫을 붙여넣습니다.

▼ **[코드스니펫] index.html 생성**

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <script src="https://cdn.socket.io/socket.io-3.0.1.min.js"></script>
    <title>Hello Socket.io!</title>
  </head>
  <body>
    <script>
      const socket = io("ws://localhost:3000");
      socket.on("connect", () => {
        socket.send("Hello!");
      });

      socket.on("message", (data) => {
        console.log(data);
      });
    </script>
  </body>
</html>
```

이 코드스니펫은 프론트엔드에서 socket.io를 사용할 수 있도록 script 태그로 socket.io 파일을 불러와 서버에 연결 요청하는 코드가 포함되어 있습니다.

▼ app.js 파일 생성

생성한 폴더 안에 `app.js` 파일을 새로 생성합니다.

▼ 2) socket.io 공식 문서를 따라서 서버에서 연결 준비하기

언제나처럼 처음 사용하는 도구는 공식 문서에 있는 내용을 따라서 시도합니다!

▼ 서버 연결 준비 코드

```
const io = require("socket.io")(3000, {
  cors: {
    origin: "*",
    methods: ["GET", "POST"],
  },
});

io.on("connection", (socket) => {
  console.log("새로운 소켓이 연결됐어요!");

  socket.on("message", (data) => {
    console.log(data);
  });
});
```

▼ 3) socket 연결 여부 확인하기

서버에서 socket 연결 준비를 했으니 프론트엔드와 연결이 잘 되는지 확인해야겠죠?

▼ 백엔드에서 연결 확인하기

1. 서버를 켭니다.
2. 크롬으로 index.html 파일을 엽니다.
3. 서버를 실행한 터미널에 `새로운 소켓이 연결됐어요!` 라는 메세지가 출력되면 성공입니다! 🎉

   프론트엔드에서 새로고침 할때마다 소켓이 다시 연결되는것이 정상이에요!

▼ 프론트엔드에서 연결 확인하기

1. 서버를 켭니다.
2. 크롬으로 index.html 파일을 엽니다.
3. 크롬의 개발자 도구(F12)를 엽니다.
4. Network 탭을 누릅니다.
5. 페이지를 새로고침 합니다.
6. 이 때, 아래 사진과 같은 항목의 "Name"이 빨간색이 없으면 됩니다.

▼ 4) 서버에서 데이터 보내는 코드 작성

1. 데이터를 보내는 방법은 매우 간단합니다!

   `socket.send("메세지값")` 처럼 작성하면 끝!

   ▼ 예시

```
io.on("connection", (socket) => {
  socket.send("Hello!");

  socket.on("message", (data) => {
    console.log(data);
  });
});
```

2. 서버를 켜봅니다!

   이렇게 하면 socket.io가 3000 포트로 서버를 열어줍니다!

```
node app.js
```

3. html 파일을 크롬으로 열어봅니다.

   html 파일을 마우스로 잡아서 크롬 탭에 놓으면 열 수 있습니다!

▼ 5) index.html 파일에 커스텀 이벤트 핸들링 코드 추가하기

아래 코드를 index.html 파일 안의 script 태그 맨 아래에 넣어주세요!

▼ **[코드스니펫] 프론트엔드 이벤트 핸들링 코드**

```
socket.on("customEventName", (data) => {
  console.log(data);
});
```

▼ 6) 서버에 커스텀 이벤트로 데이터 발행하기

`socket.emit("이벤트이름", "데이터")` 이렇게 함수를 호출하면 특정 이벤트에 특정 데이터를 보낼 수 있게 됩니다.

▼ 예시

```
socket.emit("customEventName", "this is custom event data");
```

서버를 껐다가 켠 뒤 html 파일을 열어 확인해보세요!

▼ 7) express와 같이 사용해보기

이 방법도 공식 문서에서 잘 알려주고 있으니 참고해서 해보겠습니다!

1. express 모듈 설치

```
npm i express -S
```

2. express 앱 생성, http 서버 생성

   공식 문서에 나온것처럼 express 앱을 생성하고 http 모듈로 서버를 생성합니다!

   ▼ 예시

```
const express = require("express");
const { createServer } = require("http");

const app = express();
const http = createServer(app);
```

3. socket.io 모듈 붙이기

위에서 생성한 `http` 서버 객체를 원래 있던 코드에서 "3000"을 지우고 대신 넣어주면 express와 socket.io를 동시에 사용할 수 있습니다!

▼ 예시

```
const express = require("express");
const { createServer } = require("http");

const app = express();
const http = createServer(app);
const io = require('socket.io')(http, {
  cors: {
    origin: "*",
    methods: ["GET", "POST"],
  },
});
```

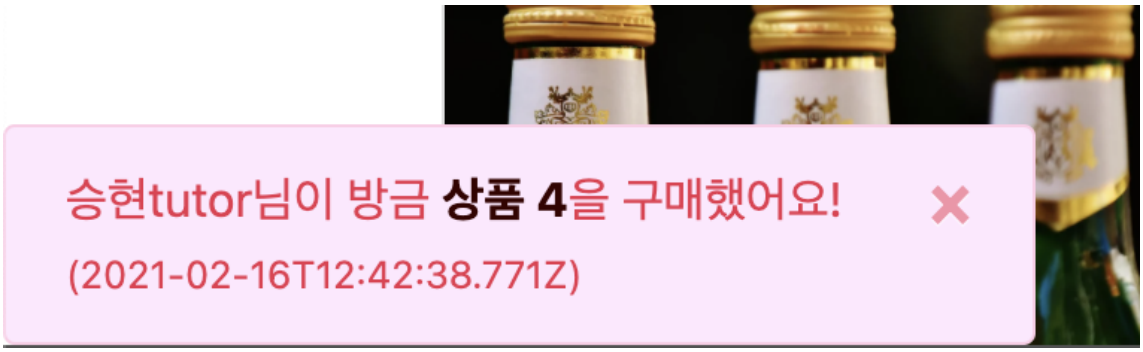
`app` 객체는 express로 기존처럼 API을 개발하거나 프론트엔드 파일을 서빙하는 용도로 사용할 수 있고, `io` 객체도 기존처럼 클라이언트와 데이터를 주고 받는 용도로 사용이 가능합니다!

04. 쇼핑몰 실시간 구매 알림 기능 구현 준비

지금까지 해보면서 Socket.io의 기능을 간단하게 배워봤습니다! 신기하고 재밌지 않나요? 😆
이제 다시 쇼핑몰 프로젝트에 기능을 추가해보겠습니다!

▼ 1) 이미 구현된 쇼핑몰 구경하기

누군가 쇼핑몰에서 구매를 하면 모든 사용자들에게 아래 사진처럼 좌측 하단에 알림이 뜨게 구현되어 있어요!



http://3.36.86.60:8888/ 에서 직접 확인할 수 있습니다 😉
페이지를 여러개 띄워서 확인할 수 있어요!

자, 그럼 이제 이것을 어떻게 구현할 수 있는지 알아보겠습니다!

▼ 2) 구현에 무엇이 필요한지 고민해보기

- socket.io로 클라이언트와 서버가 소켓 연결을 해야합니다.
- 클라이언트에서 사용자가 구매 버튼을 누른 경우 서버로 "구매했어요"와 같은 데이터를 보내줘야만 접속중인 사용자들에게 알릴 수 있습니다!
- 다른 사용자가 구매를 한 경우 서버에서 모든 클라이언트에게 "누군가 구매를 했어요!"와 같은 데이터를 보내줘야 실시간 구매 알림을 띄울수 있습니다!

▼ 3) 프론트엔드 파일 다운로드

무엇이 필요한지 정했으니 이제 본격적으로 시작에 앞서 프론트엔드 파일을 다운받아 적용해볼게요!

▼ [코드스니펫] 스파르타 쇼핑몰 프론트엔트 파일

```
https://s3.ap-northeast-2.amazonaws.com/materials.spartacodingclub.kr/nodejs_advanced/week04/week4-shopping.zip
```

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/28c3e3a9-cefc-45b9-95e0-c22821b6abad/week4-shopping.zip

위 파일을 다운받아 2주차에 했던것처럼 압축을 풀어서 나온 파일들을 static 폴더에 덮어씌워줍니다!

▼ 4) 프론트엔드 구현 설명

기능이 동작하기 위해 필요한 내용들은 프론트엔드에 이미 구현해두었습니다!
다만, 제가 미리 구현해둔 코드이기 때문에 여러분이 제가 짜둔 데이터 형식을 맞춰서 주고 받도록 해야하므로 어떻게 구현했는지 설명하고 어떻게 데이터를 보내면 되는지 알려드릴게요!

1. 로그인을 하면 서버에 소켓 연결을 시도하도록 구현해두었습니다.
   연결을 시도하는 서버 주소는 위 프론트엔드 파일을 서빙하는 서버 주소로 지정되게 해놓았으니 이제부터는 html 파일을 직접 열지 않고 **반드시 서버가 제공하는 주소(예: http://localhost:8080)로 접속해야 정상적으로 동작합니다!**
2. "BUY_GOODS" 이벤트를 서버에게 수신받으면 실시간 구매 알림이 뜨도록 구현해두었습니다.
   아래의 데이터 형식으로 받는 경우에만 정상적으로 동작합니다.

```
{
  nickname: '서버가 보내준 구매자 닉네임',
  goodsId: 10, // 서버가 보내준 상품 데이터 고유 ID
  goodsName: '서버가 보내준 구매자가 구매한 상품 이름',
  date: '서버가 보내준 구매 일시'
}
```

3. 상품을 구매하면 "BUY" 이벤트를 아래 데이터 형식과 함께 서버로 보냅니다.

```
{
  nickname: '로그인한 사용자 닉네임',
  goodsId: 10, // 로그인한 사용자가 구매한 상품 고유 ID
  goodsName: '로그인한 사용자가 구매한 상품 이름',
}
```

▼ 5) 모듈 설치

프론트엔드에서 데이터를 어떻게 주고 받는지 알았으니, 이제 본격적으로 코드를 작성해보겠습니다!

우선 터미널을 열어 쇼핑몰 프로젝트 경로에서 아래 명령어를 입력하여 모듈을 설치해주세요!

```
npm i socket.io -S
```

05. 쇼핑몰 실시간 구매 알림 구현(1)

▼ 1) 소켓 연결을 위한 서버 코드 작성

▼ 어떻게 해야 할까요?

02. socket.io 사용해보기 에서 했던 내용과 완전히 동일해요!

express 앱을 http 서버로 한번 감싼 뒤, socket.io 모듈에 http 서버 객체를 넘겨주면 socket.io 연결 준비 완료!

▼ 예시

```
const express = require("express");
const { Server } = require("http"); // 1. 모듈 불러오기
const socketIo = require("socket.io"); // 1. 모듈 불러오기

... // 생략

const app = express();
const http = Server(app); // 2. express app을 http 서버로 감싸기
const io = socketIo(http); // 3. http 객체를 Socket.io 모듈에 넘겨서 소켓 핸들러 생성

// 4. 소켓 연결 이벤트 핸들링
io.on("connection", (sock) => {
  console.log("새로운 소켓이 연결됐어요!");

  sock.on("disconnect", () => {
    console.log(sock.id, "연결이 끊어졌어요!");
  });
});

... // 생략

// 5. app 대신 http 객체로 서버 열기
http.listen(8080, () => {
  console.log("서버가 요청을 받을 준비가 됐어요");
});
```

구현을 했다면 서버를 켜서 프론트엔드와 제대로 소켓 연결이 되는지 확인해보세요!
프론트엔드엔 이미 소켓 연결 코드가 작성되어 있습니다!

▼ 2) 프론트엔드가 "BUY_GOODS" 이벤트를 제대로 받아 동작하는지 확인해보기

▼ 어떻게 해야 할까요?

프론트엔드가 제대로 동작하는지 확인하려면 서버에서 "BUY_GOODS" 이벤트로 데이터를 보내보면 되겠죠?
일단 테스트니까 소켓이 연결될 때마다 연결된 소켓에 데이터를 보내는걸로 해보세요!

우선 저는 위에서 말했던 데이터 형식 그대로 복사해서 코드를 작성해보겠습니다 😎

▼ **[코드스니펫] 가짜 구매 알림 데이터 복사**

```
{
  nickname: '서버가 보내준 구매자 닉네임',
  goodsId: 10, // 서버가 보내준 상품 데이터 고유 ID
  goodsName: '서버가 보내준 구매자가 구매한 상품 이름',
  date: '서버가 보내준 구매 일시'
}
```

▼ 예시

```
... // 생략

io.on("connection", (sock) => {
  console.log("새로운 소켓이 연결됐어요!");

  sock.emit("BUY_GOODS", {
    nickname: "서버가 보내준 구매자 닉네임",
    goodsId: 10, // 서버가 보내준 상품 데이터 고유 ID
    goodsName: "서버가 보내준 구매자가 구매한 상품 이름",
    date: "서버가 보내준 구매 일시",
  });

  sock.on("disconnect", () => {
    console.log(sock.id, "연결이 끊어졌어요!");
  });
});

... // 생략
```

여러분이 잘 따라오셨다면 서버를 껐다 켜고 크롬에서 새로고침을 하자마자 아래처럼 좌측 하단에 뜰거예요!

> 서버가 보내준 구매자 닉네임님이 방금 **서버가 보내준 구매자가 구매한 상품 이름**을 구매했어요!
> (서버가 보내준 구매 일시)

프론트엔드가 제대로 동작하는것은 확인했으니 다음 시간에 서버가 진짜 데이터를 보낼수 있도록 구현해볼 예정이예요!

▼ 3) 프론트엔드에서 구매를 하는 경우 "BUY" 이벤트를 제대로 보내는지 확인해보기

▼ 어떻게 해야 할까요?

이번에도 간단합니다! "BUY" 이벤트로 어떤 데이터를 보내는지 `console.log` 함수로 확인할 수 있도록 간단한 코드를 작성해보세요!

▼ 예시

```
io.on("connection", (sock) => {
  console.log("새로운 소켓이 연결됐어요!");

... // 생략

  sock.on("BUY", (data) => {
    console.log(data);
  });

... // 생략
});
```

이번에도 여러분이 잘 따라주셨다면, 쇼핑몰에서 구매 버튼을 눌렀을 때 터미널에 아래처럼 데이터가 뜰거예요!
위에서 제가 말씀드린 데이터 형식으로 전달이 잘 됐는지 확인해주세요!

이제 여러분은 이 데이터를 받아서 소켓이 연결된 모든 클라이언트에게 "BUY_GOODS" 이벤트로 데이터를 보내면 실시간 구매 알림 기능 구현이 끝납니다!

다음 시간에 실제로 동작하는 코드를 작성해볼게요! 😄

06. 쇼핑몰 실시간 구매 알림 구현(2)

저번 시간에서 클라이언트와 소켓 연결도 하고, 소켓으로 데이터를 간단하게 주고 받는 방법에 대해서 알아봤다면, 이번에는 정말로 누군가 구매를 하면 실시간으로 알림이 뜨도록 구현을 해볼게요!

저번 시간에 배운것을 잘 이해했다면 아주 간단하니 걱정하지 마세요!

▼ 1) 저번 시간에 테스트했던 코드 지우기

저번 시간에 아래와 같은 코드를 작성했던 적이 있었죠? 이것은 서버에서 보낸 데이터를 클라이언트가 잘 받아서 알림을 띄워주는지 테스트하기 위함이었으니 이제 지워볼게요!

아래처럼 생긴 코드를 지워주세요! 저와 똑같이 했다면 `app.js` 파일에 있을거예요 😉

```
sock.emit("BUY_GOODS", {
  nickname: "서버가 보내준 구매자 닉네임",
  goodsId: 10, // 서버가 보내준 상품 데이터 고유 ID
  goodsName: "서버가 보내준 구매자가 구매한 상품 이름",
  date: "서버가 보내준 구매 일시",
});
```

▼ 2) 서버에서 "BUY" 이벤트를 받아 데이터를 원하는 형태로 가공합니다!

여기에서도 저번에 작성한 코드를 기반으로 할건데요, 우리는 아래와 같은 코드를 작성한적 있었죠?

```
sock.on("BUY", (data) => {
  console.log(data);
});
```

클라이언트가 "BUY" 이벤트로 보내주는 데이터를 가공해주세요!

▼ 어떻게 해야 할까요?

"BUY_GOODS"로 보낼 데이터의 형태로 바꿔주세요!
데이터 형태가 기억이 안날것같아 아래에 다시 적어두었어요! 😁

```
{
  nickname: '서버가 보내준 구매자 닉네임',
  goodsId: 10, // 서버가 보내준 상품 데이터 고유 ID
  goodsName: '서버가 보내준 구매자가 구매한 상품 이름',
  date: '서버가 보내준 구매 일시'
}
```

구현을 하다보면 "서버가 보내준 구매 일시" 데이터는 클라이언트가 "BUY" 데이터에 포함하지 않고 있는걸 볼수 있을텐데요,
이 시간 정보는 서버에서 만들어서 보내주도록 합시다!

▼ 서버에서 시간 정보 만드는 방법

```
console.log(new Date().toISOString());
```

이렇게 하면 간단히 시간 정보를 보여줄 수 있어요!
(단, 여러분에게 익숙한 양식은 아닐수 있지만 간단히 구현해보기 위해 이렇게 작성하고 넘어갈게요!)

▼ 예시

```
  sock.on("BUY", (data) => {
    const emitData = {
      ...data,
      date: new Date().toISOString(),
    };
  });
```

▼ 3) 가공한 데이터를 클라이언트에게 "BUY_GOODS" 이벤트로 보내줍니다!

위에서 가공한 데이터를 모든 클라이언트에게 보내면 됩니다!

▼ 서버에서 모든 클라이언트로 데이터를 보내는 방법

그동안 우리는 `io.on("connection", (sock) => { ... });` 이런 코드 안에서 `sock` 객체를 사용하여 소켓에 데이터를 보내고 있었을텐데요,
이렇게 하면 서버 입장에서는 연결된 소켓 하나 하나에만 각자 접근하기 때문에 서버와 클라이언트가 1:1로 개인 채팅방을 갖는 형태가 됩니다.

단체 채팅방처럼 서버가 연결된 모든 클라이언트에 데이터를 한번에 보내기 위해서는 아래처럼 하면 됩니다!

```
io.emit("BUY_GOODS", emitData);
```

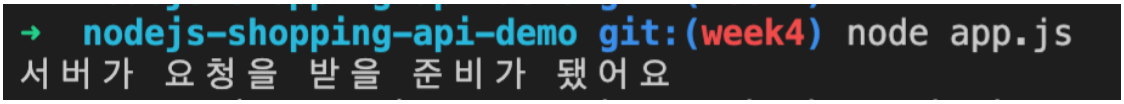
▼ 예시

```
sock.on("BUY", (data) => {
  const emitData = {
    ...data,
    date: new Date().toISOString(),
  };

  io.emit("BUY_GOODS", emitData);
});
```

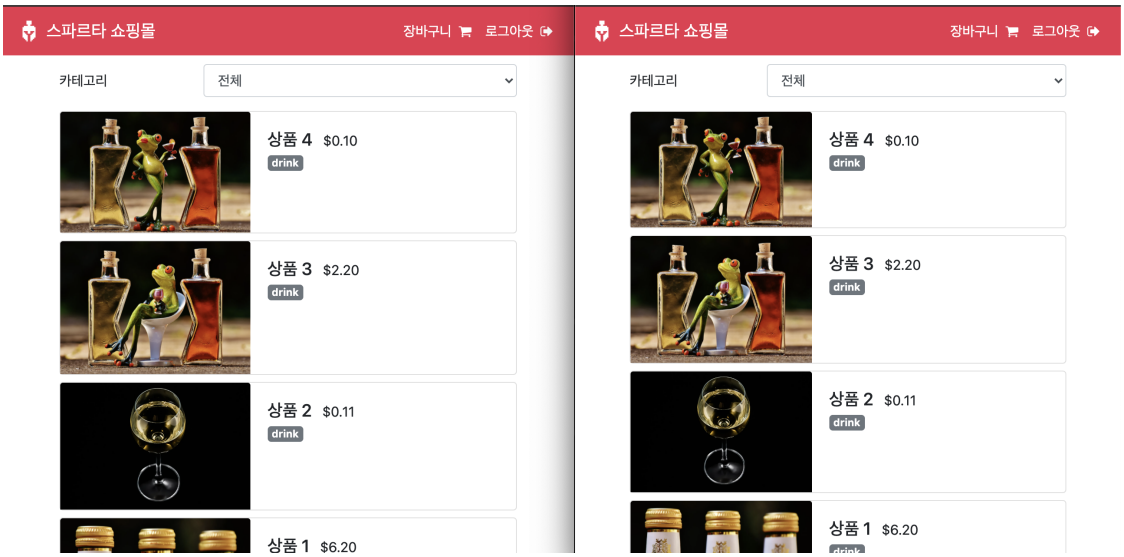
▼ 4) 잘 동작하는지 확인해보기

1. 우선 서버를 실행해주세요!

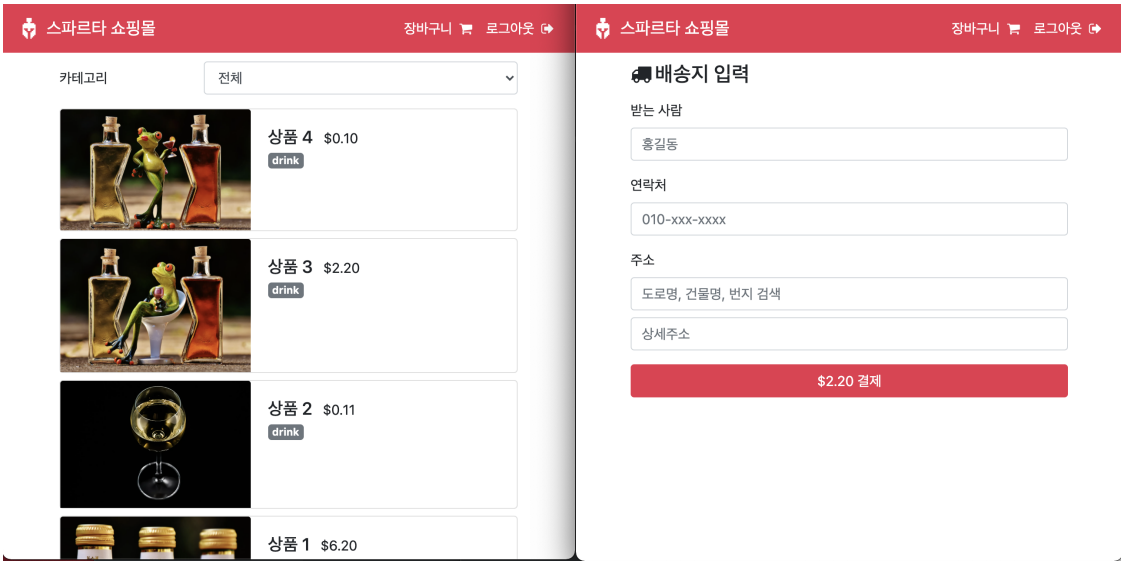


2. http://127.0.0.1:8080 으로 들어가서 로그인한 상태의 창 두개를 띄우기

   하나는 다른 브라우저 또는, 시크릿 탭으로 열어주세요!

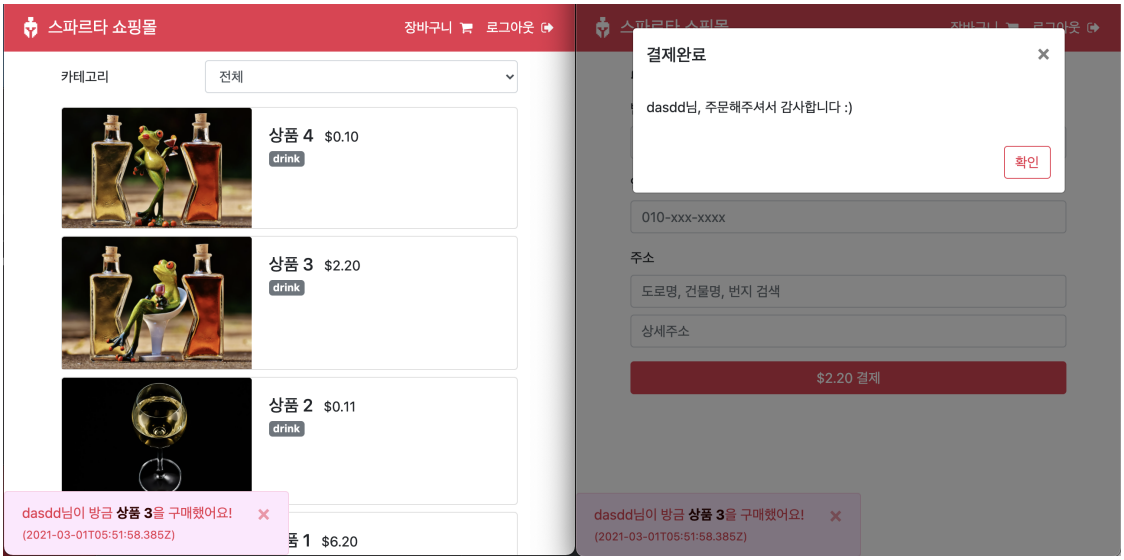


3. 한쪽 탭에서 상품 아무거나 구매해보기!



구매 기능은 구현한게 없어서 바로 결제 버튼을 눌러도 구매했다고 인식해요!

4. 구매 알림이 잘 뜨는지 확인



위처럼 잘 뜨면 기능 구현이 끝났습니다!

아주 간단하지 않나요? 😄

07. 4주차 끝 & 숙제 설명

▼ 실시간 카운터 기능 구현

이 기능도 프론트엔드에는 이미 구현되어 있습니다!
아래 내용을 참고해서 백엔드 코드를 작성해보세요!

▼ 로그인 상태에서 상세 페이지로 이동하면 "CHANGED_PAGE" 이벤트를 서버로 전송합니다.

클라이언트가 보내는 데이터는 현재 접속중인 경로를 문자열 그대로 보냅니다!

ex) http://localhost:8080/detail.html?goodsId=4 → `/detail.html?goodsId=4`

▼ 프론트엔드로 "SAME_PAGE_VIEWER_COUNT" 이벤트에 2 이상인 값을 보내면 상세 페이지에서 "총 n명이 이 상품을 구경하고 있습니다!" 라는 문구가 표현됩니다.

한번 직접 위 이벤트를 프론트엔드로 보내서 확인해보세요!