[스파르타코딩클럽] Node.js 심화반 - 1주차

매 주차 강의자료 시작에 PDF파일을 올려두었어요!

▼ PDF 파일

[수업 목표]

1. 기초반에서 배웠던 내용을 토대로 간단한 할 일 메모 기능이 있는 사이트를 만들어본다.
2. Validation이 무엇인지 알게 된다.
3. REST API란 무엇인지 알게 된다.

[목차]

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

---

01. 지식 수준 맞추기

주로 앞에서 배운 내용들이니 너무 걱정하지 마세요!

▼ 1) 이것만은 오해 없이!

안그래도 머리가 복잡하다면 그냥 넘어가도 괜찮습니다!
나중에 마음의 여유를 가지고 확인해보세요 😉

▼ Q. **Node.js는 프로그래밍 언어이다!?**

A. 💁‍♂️! Node.js는 JavaScript를 브라우저 없이 단독으로 실행할 수 있는 하나의 플랫폼입니다. (위키피디아 참고)

▼ Q. **npm은 Node.js와 같은 프로그램이다!?**

A. 💁‍♂️! npm은 여러분이 Node.js에서 오픈소스 라이브러리를 쉽게 사용하기 위해 개발된 "패키지 관리자"라는 형식의 프로그램입니다.

▼ Q. **express는 서버라고 불러도 된다?!**

A. 💁‍♂️! `express`라는 라이브러리는 여러분이 쉽게 서버 프로그램을 구성할 수 있게 만들어진 "오픈소스 라이브러리"입니다.

▼ 2) 우리가 평소에 이용하는 웹 사이트나 웹 브라우저에서 사용되는 기술들에 대한 이해

1. 🌐 **HTTP란?** 우리가 일반적으로 데이터를 주고 받을 때 사용되는 통신 규약! (모두의 약속 🤙)
(참고)
2. 🕸 **웹 브라우저는?** HTML로 이루어진 데이터를 읽어서 화면에 그려주는 역할!
   - 단순히 웹 문서를 가져와 보여주는것 뿐만 아니라, 여러가지 프로토콜(http, ftp, file 등)을 지원하며 다른 웹 서버에 데이터를 보낼수 있기도 합니다.
   - 웹 개발자들에게는 그 무엇보다 제일 중요한 도구중 하나인것이죠!
   - 추가 정보는 위키피디아 참고
3. 🍪 **쿠키란?** 웹 브라우저에 구현된 기술 중 하나. 보통 상태를 저장하기 위해서 사용합니다. (참고)

   ▼ T.M.I.

   서버에서 쿠키를 노릇노릇 구워서 Response에 담아 보내면 웹 브라우저는 받은 데이터를 그대로 저장합니다.
   브라우저는 가지고 있는 쿠키가 있다면 서버에 Request를 할 때 항상 가지고 있는 쿠키 데이터를 포함해서 보냅니다.
   단, 쿠키는 별도의 암호화 없이 데이터를 그대로 주고받기 때문에 클라이언트에서 마음대로 조작하기 쉬워 보안에 취약합니다.
4. 🔐 **세션이란?** 웹 브라우저에 구현된 기술중 하나. 그러나 세션은 쿠키의 특성을 이용한 기술입니다.

   ▼ T.M.I.

   세션 데이터는 서버에 저장되고 데이터마다 고유한 세션 ID가 만들어집니다.
   이 ID를 쿠키를 이용해 주고 받기 때문에 세션 데이터에 접근이 가능한것은 오직 서버뿐이기 때문에 쿠키가 가지고 있던 보안 취약점을 해결합니다.
5. 🤵 **서버(Server) 프로그램이란?** 일반적으로 클라이언트에게 요청을 받아 응답을 주는 프로그램의 유형입니다.
6. 🤵 **서버(Server) 컴퓨터란?** 위에서 설명된 "서버 프로그램"을 실행하고 있는 컴퓨터입니다.

▼ 3) 라이브러리 `express`에 대한 이해

1. `express`를 이용하여 서버를 만들 수 있다.
2. 미들웨어의 개념을 이해할 수 있다.
3. 내가 만든 서버로 "정적 파일(Static file)"을 제공할 수 있다.

   > 💡 **정적 파일?** 말 그대로 "변하지 않는 파일"이라고 생각하면 좋습니다. ("정적"의 사전적 의미)
   > 서버에서 파일 내용을 변형하여 사용하지 않고, 클라이언트(요청자)에게 그대로 전달하기 위한 목적의 파일입니다.

   우리가 사용하는 `express` 라는 라이브러리는 정적 파일을 쉽게 제공할 수 있게 해주는 미들웨어가 존재합니다.
4. 템플릿 엔진이 무엇인지 이해할 수 있다.

5. 라우터를 이용해 원하는 Method와 경로로 HTTP 요청을 받아 처리하는 방법을 안다.

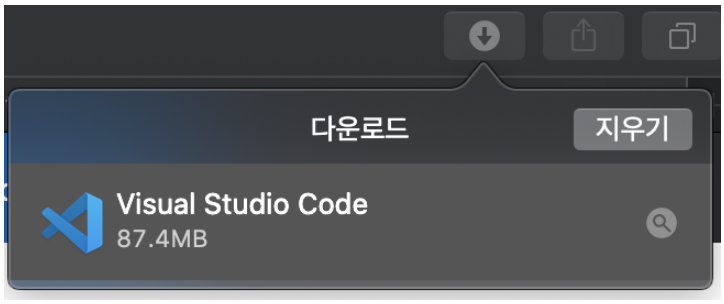
02. 필수 프로그램 설치

▼ 4) Visual Studio Code (VS Code) **(다운로드 링크)**

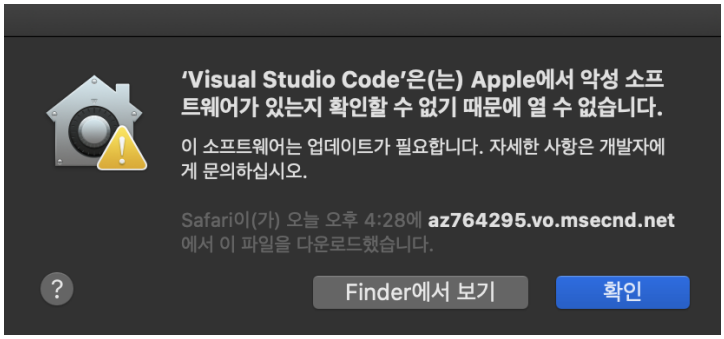
- 윈도우는 바로 설치가 됩니다.

▼ (←눌러보기) 맥북 설치방법

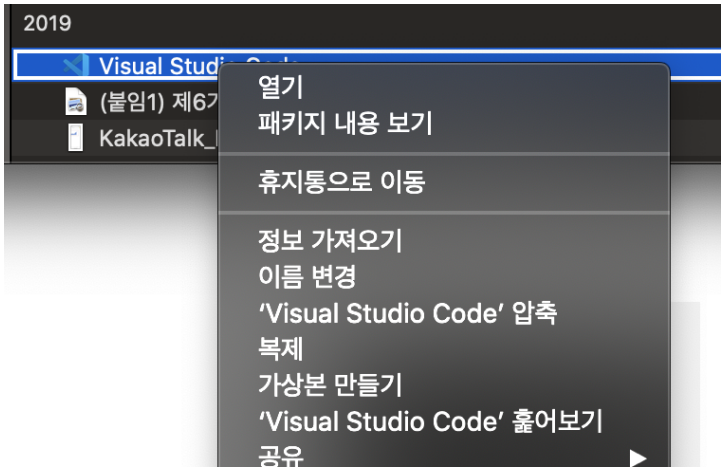
1. 위 링크에서 Visual Studio Code를 다운로드 받습니다.
2. Visual Studio Code파일을 더블클릭하여 실행합니다.



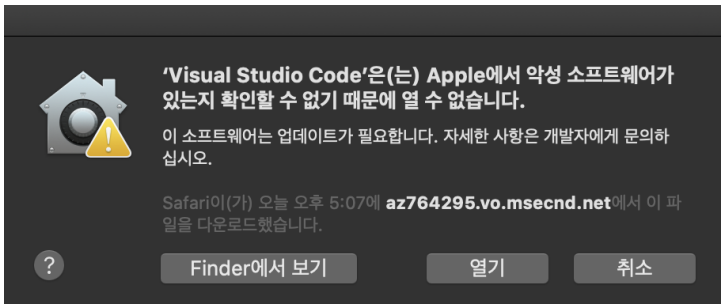
3. 아래와 같은 메시지가 뜹니다. **[Finder에서 보기]**를 클릭합니다.



4. Visual Studio Code를 우클릭하고, **[열기]** 를 누릅니다.



5. **[열기]**를 누르시면 설치가 끝납니다! 이제부턴 그냥 여실 수 있습니다! :-)



03. 앞으로 우리가 배울 것

▼ 5) 1~5주차 배울 순서

- 1주차: 간단한 할 일 메모 사이트를 만들어보면서 Node.js 기초 지식 복습하기

  → 입문 수업에서 배운 내용 중 핵심 개념인 restAPI와 validation에 대해서 구현해봅니다.

- 2주차: 스파르타 쇼핑몰에 로그인/회원가입 기능 붙이기

  → JWT의 사용법을 익히고, 로그인 기능을 직접 구현해봅니다.

- 3주차: 데이터베이스 유형에 구애받지 않기

  → MySQL 서버를 키고, sequelize 라이브러리를 사용해 MySQL 데이터를 읽고 써봅니다.

- 4주차: 스파르타 쇼핑몰에 실시간 통신 구현하기

  → 웹소켓 라이브러리인 socket.io를 활용하여 실시간 통신 기능을 구현해봅니다.

- 5주차: 실무 꿀팁 - 테스트 코드, 코드 정리법

  → 테스트 코드를 작성하는 방법과 코드를 나눠서 정리하는 법 그리고 코드 포맷을 일관적으로 정리하는 법에 대해 배워봅니다.

▼ 6) 만들 서비스 살펴보기

▼ [코드스니펫] 할 일 메모 사이트

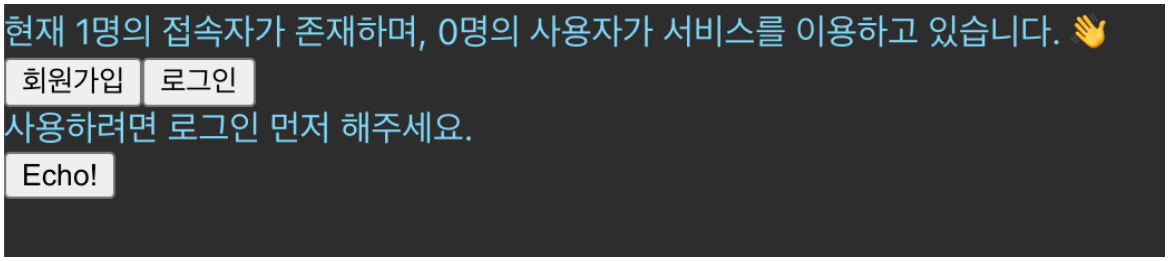
```
http://3.36.86.60:3000/
```

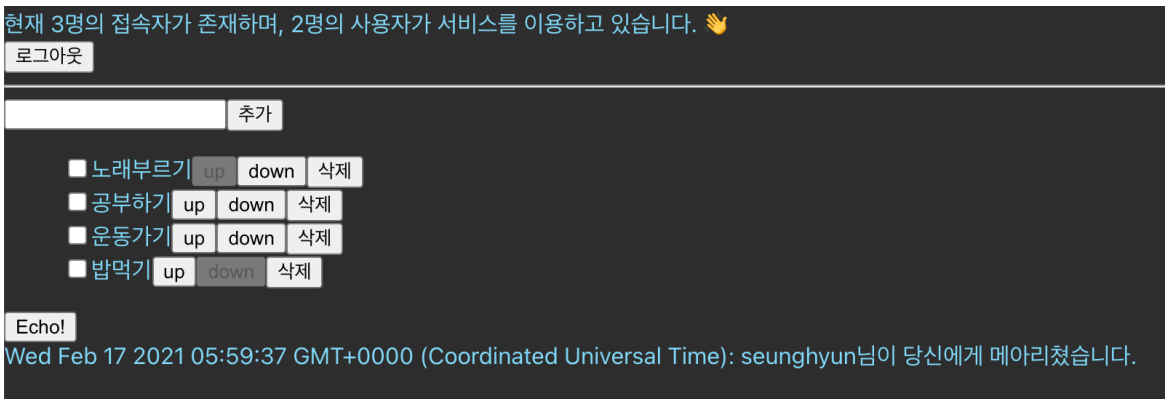
▼ [코드스니펫] 스파르타 쇼핑몰

```
http://3.36.86.60:8888/
```

04. [할 일 메모 사이트] - 준비 단계

▼ 7) 이번주에 만들 것! - 할 일 메모 사이트 (링크)





5주차가 끝났을 때 여러분은 위 사이트에 있는 기능을 전부 직접 구현할 수 있게 될 거예요!

- 로그인
- 회원가입
- 할 일 추가
- 할 일 수정
- 할 일 삭제
- 실시간 접속자 카운터
- 실시간 메아리 기능

▼ 8) 어떤 기능이 필요한지 고민해보기

1. **할 일 추가하기**
2. **할 일 목록 보기**
3. **할 일 순서 변경하기**
4. **할 일 완료하기**
5. **할 일 완료 해제하기**
6. 로그인
7. 회원가입

우선 1~3번 기능을 먼저 구현해볼까요?

▼ 9) 위 기능을 구현하려면 무엇이 필요한지 고민해보기

1. 일단 할 일을 기록할 수 있게 해주는 페이지가 있어야겠죠?
   걱정 마세요! 여러분이 Node.js 학습에 집중할 수 있도록 제가 준비해두었습니다 😉
   아래에서 파일을 드릴 거예요!
2. API 구현을 위해 express를 이용한 서버 코드를 짜야합니다!
3. 여기까지 하면 기능을 구현하기 위한 준비가 끝났어요!

05. [할 일 메모 사이트] - API 서버 준비하기(1)

▼ 10) 간단한 HTTP 요청을 받을 수 있도록 express 라이브러리를 이용하여 서버를 만들어봅니다!

우리는 Node.js 서버 하나로 API와 프론트엔드 파일을 모두 제공할건데, 혹시라도 프론트엔드로 제공하는 경로와 API의 경로가 겹치면 곤란하겠죠?
그렇기 때문에 폴더 하나를 만들고, `app.js` 라는 파일을 만들고 API는 항상 `api` 라는 주소를 맨 앞에 붙도록 아래처럼 구성했어요 😆

▼ **[코드스니펫] 할 일 메모 사이트 - app.js**

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

const app = express();
const router = express.Router();

router.get("/", (req, res) => {
  res.send("Hi!");
});
app.use("/api", express.json(), router);

app.listen(8080, () => {
  console.log("서버가 켜졌어요!");
});
```

▼ ✅ `app.use("/api", express.json(), router);` 코드 추가 설명

`app.use` 는 미들웨어를 사용하게 해주는 코드예요! (기초 수업에서 배웠었죠?)
그리고 맨 처음 인자 값에 들어간 `/api` 에 의해서 "http://127.0.0.1:8080/api" 경로로 접근하는 경우에만 json 미들웨어를 거친 뒤, router로 연결되도록 하는것이죠!

▼ json 미들웨어?

> 💡 json 미들웨어는 HTTP 요청을 받을때 body에 있는 데이터를 정상적으로 사용할 수 있게 분석해주는 역할을 해요!
> 우리는 지금 JSON 형태의 body를 입력받을 수 있게 된것이죠 😆

▼ ✅ `express.Router()` 코드 추가 설명

이번 수업에서는 지금 선언한 `router`에 우리가 개발한 기능을 연결해서 API를 제공할 예정이에요 😉
더 자세한 내용은 공식 사이트 > express.Router

이제 `node app.js` 라는 명령어로 API 서버를 켠 뒤, `http://127.0.0.1:8080/api` 주소로 들어가 "Hi" 라는 문구가 잘 보이는지 확인해보세요!

▼ 11) 할 일 메모 사이트 프론트엔드 파일을 서버에 추가합니다!

▼ **[코드스니펫] 할 일 메모 사이트 - 프론트엔드**

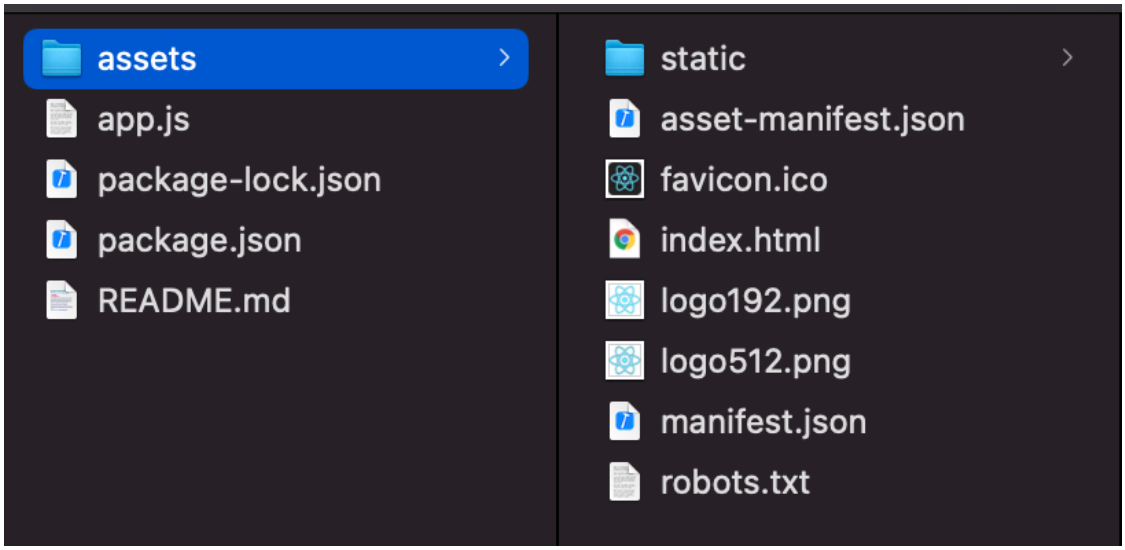
```
https://s3.ap-northeast-2.amazonaws.com/materials.spartacodingclub.kr/nodejs_advanced/week01/todo-web-week1.zip
```

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/f27e195c-1b60-4d06-aeec-3f18f28809d3/todo-web-week1.zip

위 파일을 다운로드해서 아래 모양처럼 `assets` 이라는 폴더를 만들고 `todo-web-week1.zip` 압축을 해제해서 나온 파일을 모두 넣어줬어요!

```
내 프로젝트 폴더 이름
├── app.js
├── assets
│   ├── asset-manifest.json
│   ├── favicon.ico
│   ├── index.html
│   ├── logo192.png
│   ├── logo512.png
│   ├── manifest.json
│   ├── robots.txt
│   └── static
│       ├── css
│       │   ├── main.6842b365.chunk.css
│       │   └── main.6842b365.chunk.css.map
│       └── js
│           ├── 2.96129310.chunk.js
│           ├── 2.96129310.chunk.js.LICENSE.txt
│           ├── 2.96129310.chunk.js.map
│           ├── 3.0962e34f.chunk.js
│           ├── 3.0962e34f.chunk.js.map
│           ├── main.c5ceafa0.chunk.js
│           ├── main.c5ceafa0.chunk.js.map
│           ├── runtime-main.2ec3457b.js
│           └── runtime-main.2ec3457b.js.map
├── package-lock.json
└── package.json
```

▼ 사진으로 보고싶다면?



▼ 12) assets 파일을 서빙할 수 있도록 static 미들웨어를 추가해줄게요.

클라이언트가 body에 실어 보낸 데이터를 사용하기 위해 bodyParser 미들웨어도 같이 넣어줍니다.

▼ **[코드스니펫] 할 일 메모 사이트 - static 미들웨어**

```
const express = require("express");
const bodyParser = require("body-parser");

const app = express();
const router = express.Router();

router.get("/", (req, res) => {
  res.send("Hi!");
});
```

```
app.use("/api", bodyParser.json(), router);
app.use(express.static("./assets"));

app.listen(8080, () => {
  console.log("서버가 켜졌어요!");
});
```

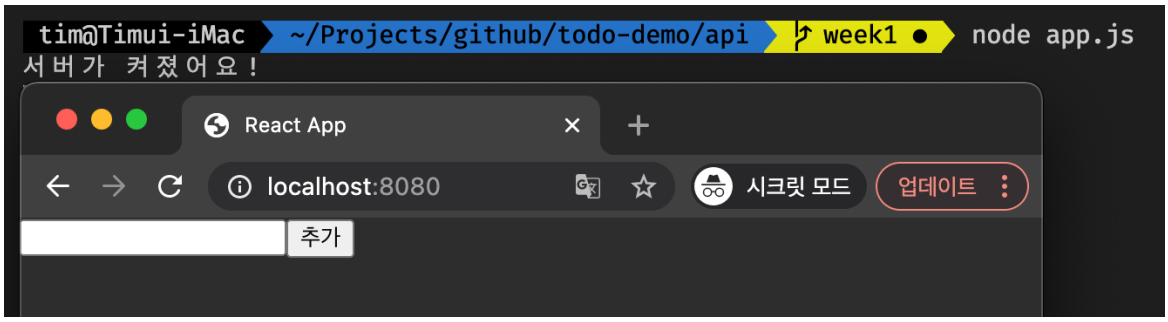
▼ ✅ `app.use(express.static("./assets"));` 코드 추가 설명

express.static 함수는 `app.js` 파일 기준으로, 입력 값(지금은 "./assets") 경로에 있는 파일을 아무런 가공 없이 그대로 전달해주는 미들웨어에요!

이 코드 덕분에 저희가 미리 준비해놓은 프론트엔드 파일을 아래처럼 서빙할 수 있게 되는것이죠!

더 자세한 내용은 공식 사이트 > Serving static files 참고

실행 중인 서버를 껐다가 다시 켜서 브라우저에서 새로고침 해보세요! 아래처럼 보이면 성공!



이제 우리는 `/api` 경로에서 API를 만들어서 제공할거고, `/` 경로에서는 저희가 미리 준비해둔 웹 서비스를 제공하기 시작했어요!

▼ ✅ 어떻게 갑자기 input과 button이 보이나요?

여러분이 Node.js 학습에 집중할 수 있게 하기 위해, 제가 정해둔 경로에 정해둔 데이터를 요청하는 코드를 짜놓았습니다 😉

이 코드는 아까 다운받아 static 미들웨어로 연결해서 서빙하는 index.html 파일과 같이 있는 static 폴더 안에 있는 js 파일들이 여러분의 서버를 통해 제공되고 있기 때문에 가능한 일이죠!

06. [할 일 메모 사이트] - API 서버 준비하기(2)

▼ 13) MongoDB 연결 준비

우린 mongoose를 사용하여 MongoDB에 데이터를 저장해볼건데요, 이를 위해 공식 문서를 따라 app.js 파일에 아래처럼 준비합니다!

데이터베이스의 이름은 `todo-demo` 으로 정해두고 진행할게요!
아, 그리고 MongoDB는 기초 과정에서 이미 배운 내용이기 때문에 자세한 설명은 생략하도록 하겠습니다 😉

▼ **[코드스니펫] mongoose 연결**

```
const mongoose = require("mongoose");

mongoose.connect("mongodb://localhost/todo-demo", {
  useNewUrlParser: true,
  useUnifiedTopology: true,
});
const db = mongoose.connection;
db.on("error", console.error.bind(console, "connection error:"));
```

▼ 14) API를 구현하기 전에 짚고 넘어갈것!

▼ REST API란 무엇인가?

- REST 아키텍쳐를 따라 구현된 API를 REST API라고 부릅니다.
- 간단히 말하면 원래 있던 방법보다 더 쉽고 사람이 읽기 편한 방식으로 원칙을 세워놨고, 개발자들의 생산성과 상호작용을 증진시키는것에 목적이 있습니다.
  (다른 방식에 비해 조금 더 아름답다고 이해해도 됩니다 😋)
- 참고할 글
  - https://greatkim91.tistory.com/13
  - http://haah.kr/2017/06/28/rest-the-coclusion/

▼ Validation이란 무엇인가?

- 특별한게 아니지만 개발을 하면서 가장 중요한것중 하나입니다.
  Validation은 말 그대로 어떤것을 검증한다고 보면 됩니다.
- 코드로 보는게 가장 쉽겠죠?

```
function is1(value) {
  return value === 1;
}
```

- 위 코드는 단순히 값이 1인지 아닌지 판단해서 Boolean 타입의 값을 반환하는 함수입니다.
  이렇게 단순한 함수조차 Validation. 즉, 검증을 위한 코드가 됩니다.
- 여러분은 날이 갈수록 당연히 더 복잡하고 어려운 검증 로직을 짜게 될텐데요, 우리는 심화 과정에서 이런 로직을 더 쉽고 간결하게 작성하도록 도와주는 라이브러리를 사용해볼 예정입니다 😎

07. [할 일 메모 사이트] - API 구현하기(1)

▼ 15) mongoose - Todo 모델 작성하기

MongoDB에 연결 할 준비가 됐으니 이제 `Todo` 모델을 작성하고 기능을 구현해봅니다.

▼ 예시

```
const mongoose = require("mongoose");

const TodoSchema = new mongoose.Schema({
  value: String,
  doneAt: Date,
  order: Number
});
TodoSchema.virtual("todoId").get(function () {
  return this._id.toHexString();
});
TodoSchema.set("toJSON", {
  virtuals: true,
});
module.exports = mongoose.model("Todo", TodoSchema);
```

참고 - 몽구스 > 가이드 > Virtual

mongoose에 대한 설명은 기초 수업에서 많이 다뤘으니 자세한 설명은 생략합니다!
기억이 잘 안나신다면 공식 가이드 문서를 참고하시면 좋아요 😉

▼ 16) 할 일 추가 API 만들기

👉 모델을 작성했으니 이제 할 일 추가 API를 짜볼게요!
간단히 입력받은 값을 저장하고 저장한 값을 응답값으로 주도록 하겠습니다 🙂

단, order라는 숫자 값을 같이 저장할 건데요!
이 값은 추가한 Todo 데이터들의 순서를 가지는 값이 될 예정입니다.
order가 높을 수록 해당 Todo 데이터가 상단에 위치하도록 하는 것이죠!

추가된 항목은 항상 맨 위에 추가될 수 있게 이미 저장된 order 값보다 1이 더해진 값으로 저장하도록 구현해보세요!

▼ 할 일 추가 API

```
const Todo = require("./models/todo");

router.post("/todos", async (req, res) => {
  const { value } = req.body;
  const maxOrderByUserId = await Todo.findOne().sort("-order").exec();

  const order = maxOrderByUserId ? maxOrderByUserId.order + 1 : 1;
  const todo = new Todo({ value, order });
  await todo.save();
  res.send({ todo });
});
```

08. [할 일 메모 사이트] - API 구현하기(2)

▼ 17) 할 일 목록 가져오는 API 만들기

👉 이번엔 할 일 목록을 가져오는 코드를 짜볼 건데요, 이번엔 아주 쉽습니다.
order 라는 값을 기준으로 내림차순해서 Todo 데이터를 가져와보세요!

▼ 예시

```
router.get("/todos", async (req, res) => {
  const todos = await Todo.find().sort("-order").exec();

  res.send({ todos });
});
```

▼ 18) 할 일 순서 변경하는 API 만들기

👉 자, 우선 여기서는 순서를 변경하는 기능만 구현해볼게요!
어떤 것부터 해야할까요?

2개의 Todo 항목의 order 값을 서로 바꿔서 저장하면 어떻게 될까요?
맞습니다! order 값을 바꿨으니까 위치가 바뀌는거죠!

그러면 url에서 params로 입력받은 todoId를 ID로 가진 항목과,
내가 바꾸려는 order 값을 가진 항목의 order 값을 바꿔서 저장해주면 되겠죠!

그럼 이제 코드를 작성해볼까요? 😎

▼ 예시

```
router.patch("/todos/:todoId", async (req, res) => {
  const { todoId } = req.params;
  const { order } = req.body;

  const currentTodo = await Todo.findById(todoId);
  if (!currentTodo) {
    throw new Error("존재하지 않는 todo 데이터입니다.");
  }

  if (order) {
    const targetTodo = await Todo.findOne({ order }).exec();
    if (targetTodo) {
      targetTodo.order = currentTodo.order;
      await targetTodo.save();
    }
    currentTodo.order = order;
  }

  await currentTodo.save();

  res.send({});
});
```

09. 1주차 끝 & 숙제 설명

- 할 일 메모 사이트에 기능 추가하기

  ▼ 할 일 삭제 API 개발

  DELETE 메서드와 "/todos/:todoId" 경로로 구성된 API를 구현해보세요!

  ▼ 할 일 내용/체크박스 수정 API 개발

  PATCH 메서드와 "/todos/:todoId" 경로로 구성한 API를 수정해서 아래의 기능도 같이 구현해보세요!
  프론트엔드로부터 전송되는 데이터는 아래에 적어두었습니다.

  - 내용 데이터 수정

    ```
    { value: '변경될 데이터' }
    ```

  - 할일 체크 여부 수정

    ```
    { done: true }
    { done: false }
    ```

- [2주차 예습] HTTP 인증에 대해서 공부하고, 어떤 인증 유형이 있는지 정리해보기(유형별로 동작 원리까지 정리하면 완전 👍)

10. 1주차 숙제 답안 코드

▼ **[코드스니펫] - 1주차 숙제 답안 코드**

전체 코드

▼ app.js

```
const express = require("express");
const bodyParser = require("body-parser");
const mongoose = require("mongoose");
const Todo = require("./models/todo");
```

```
mongoose.connect("mongodb://localhost/todo-demo", {
  useNewUrlParser: true,
  useUnifiedTopology: true,
});
const db = mongoose.connection;
db.on("error", console.error.bind(console, "connection error:"));

const app = express();
const router = express.Router();

router.get("/", (req, res) => {
  res.send("Hi!");
});

router.get("/todos", async (req, res) => {
  const todos = await Todo.find().sort("-order").exec();

  res.send({ todos });
});

router.post("/todos", async (req, res) => {
  const { value } = req.body;
  const maxOrderTodo = await Todo.findOne().sort("-order").exec();
  let order = 1;

  if (maxOrderTodo) {
    order = maxOrderTodo.order + 1;
  }

  const todo = new Todo({ value, order });
  await todo.save();

  res.send({ todo });
});

router.patch("/todos/:todoId", async (req, res) => {
  const { todoId } = req.params;
  const { order, value, done } = req.body;

  const todo = await Todo.findById(todoId).exec();

  if (order) {
    const targetTodo = await Todo.findOne({ order }).exec();
    if (targetTodo) {
      targetTodo.order = todo.order;
      await targetTodo.save();
    }
    todo.order = order;
  } else if (value) {
    todo.value = value;
  } else if (done !== undefined) {
    todo.doneAt = done ? new Date() : null;
  }

  await todo.save();

  res.send({});
});

router.delete("/todos/:todoId", async (req, res) => {
  const { todoId } = req.params;

  const todo = await Todo.findById(todoId).exec();
  await todo.delete();

  res.send({});
});

app.use("/api", bodyParser.json(), router);
app.use(express.static("./assets"));

app.listen(8080, () => {
  console.log("서버가 켜졌어요!");
});
```