[스파르타코딩클럽] Node.js 심화반 - 3주차

매 주차 강의자료 시작에 PDF파일을 올려두었어요!

▼ PDF 파일

[수업 목표]

1. 데이터베이스 유형에 구애받지 않고 원하는 기능을 구현할 수 있다는것을 체험한다.
2. Docker를 이용해 로컬 환경에서 MySQL 서버를 켤 수 있게 된다.

[목차]

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

---

00. 선행 지식

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 😇

▼ 1) 관계형 데이터베이스(SQL)와 비관계형 데이터베이스(NoSQL)의 개념

그 동안 우리가 사용했던 MongoDB는 비관계형 데이터베이스에 해당합니다.
그리고 이번 수업에서 사용해 볼 MySQL라는게 바로 관계형 데이터베이스예요!

데이터 형식이 자유로웠던 MongoDB와 달리 관계형 데이터베이스는 **"스키마"** 라는 개념이 존재하는데요,
MySQL에서는 "스키마"가 MySQL 서버의 가상 개념인 "데이터베이스"와 동일합니다.

그리고 제일 언급이 많이 되는 **"테이블"**이라는 개념은 쉽게 말해서 스키마 하위의 개념입니다.

표 1

userId	email	nickname	password
1	test@email.com	테스트닉네임	1234

MySQL은 테이블이라는 개념으로, 사진처럼 엑셀과 유사한 형태로 데이터를 관리하며 이 테이블에 우리가 원하는 데이터를 차곡차곡 쌓을거예요.

더 궁금하다면 여러 글들을 찾아보세요! (참고1) (참고2)

▼ 2) mongoose라는 라이브러리의 개념

mongoose는 자바스크립트로 MongoDB에 데이터를 읽고 쓰기 쉽게 해주는 라이브러리였어요!

이것을 ODM(Object Document Mapper)이라고도 부르는데, 자바스크립트의 Object와 MongoDB의 Document를 서로 Mapping해주는 도구라고 볼 수 있죠.
mongoose가 ODM의 기능을 충실하게 잘 해주고 있기 때문에 우리는 MongoDB에 쉽게 데이터를 넣고 쓰고, 관리할 수 있게 되는 것이예요 ☺

01. 3주차에서 배울 것

▼ 1) Docker를 이용해 MySQL 서버 켜는 법

> 👉 Docker는 다른 사람들이 이미 구성해놓은 이미지를 "컨테이너"라는 개념으로서 구동할 수 있게 해주는 프로그램입니다!
> 예를 들어 MongoDB 개발자가 `mongo` 라는 이름으로 이미지를 만들어서 Docker 서버에 올려두었을때, 우리는 컴퓨터에 직접 MongoDB를 설치하지 않아도 `mongo` 라는 이미지를 다운받아 손쉽게 MongoDB 서버가 실행되는 "컨테이너"를 띄울 수 있는것입니다.
>
> 조금 이해하기 어렵다면, VirtualBox 또는 VMware 보다 부담이 적은 운영체제 가상머신이라고 생각하셔도 됩니다. 😉

▼ 2) sequelize라는 라이브러리를 사용해 MySQL에 데이터 읽고 쓰기

비관계형 데이터베이스인 MongoDB를 이용할때는 ODM 도구중 하나인 mongoose라는 라이브러리를 사용했었죠?
관계형 데이터베이스인 MySQL를 이용할때는 ODM 대신 ORM(Object Relational Mapper)를 사용할 수 있습니다!

그리고 우리는 ORM중 가장 유명한 Sequlize라는 라이브러리를 사용해볼 예정입니다.

02. MySQL 서버를 켜고, 접속해보기

> 🔥 일단 라이브러리를 사용해보려면 MySQL 서버가 있어야겠죠?
> 그러기 위해 MySQL 서버를 간단한 방법으로 켜볼건데요, 저번 주 숙제였던 Docker Desktop은 모두 설치하셨다고 믿고 진행해볼게요!

▼ 1) 터미널에 MySQL 실행 명령어 입력

아래 명령어는 Docker를 이용하여 MySQL 서버를 띄울수 있게 해주는 명령어입니다!

▼ **[코드스니펫] MySQL 실행 명령어**

```
docker run --rm -p 3306:3306 --name test-db -e MYSQL_ROOT_PASSWORD=1234 mysql:5.7 mysqld --character-set-server=utf8mb4 --col
```

- `host`: 127.0.0.1
- `user`: root
- `password`: 1234

위 정보를 가지고 서버로 접속할 수 있어요!

> 👉 Docker에 대해 더 자세히 알려드리고 싶지만 컨테이너 기술에 대해서는 많은 설명이 필요하므로 Node.js 수업의 목적을 벗어나게 돼요!
> 나중에 기회가 된다면 별도의 수업으로 다뤄보겠습니다!

📌 **주의!!** 이 명령어를 실행한 터미널을 닫으면 MySQL 서버가 꺼지게 되므로 수업을 진행하는 동안에는 항상 이 명령어로 MySQL 서버를 켜두어주세요!
이것은 여러분이 따로 MySQL 서버를 설치, 설정하지 않고 테스트를 할 수 있게 하기 위한 임시방편이니 참고해주세요 😇

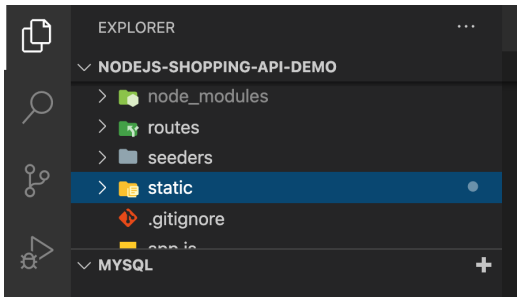
▼ 2) VS Code(Visual Studio Code)에서 MySQL 확장 플러그인 설치

- https://marketplace.visualstudio.com/items?itemName=formulahendry.vscode-mysql

위 확장 플러그인을 VS Code에 설치해주세요!

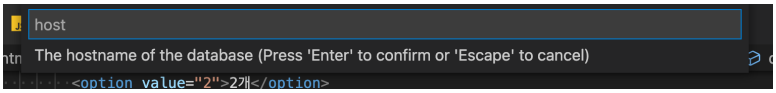
▼ 3) DB 연결 정보 등록

1. VS Code 좌측 사이드바 → 탐색기 → MYSQL 탭 우측 → + 버튼 클릭

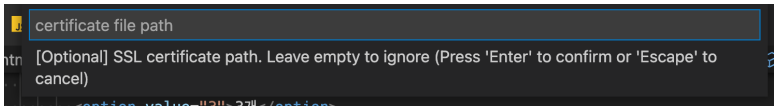


2. MySQL 서버 접속 정보 입력

   + 버튼을 누르면 아래처럼 입력 상자가 뜨는데, 위에서 말했던 접속 정보 그대로 입력합니다!

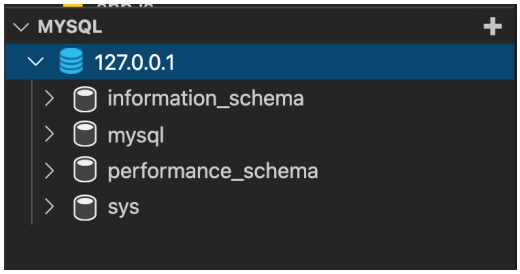


   1. host: 127.0.0.1
   2. user: root
   3. passowrd: 1234
   4. port: 3306
   5. certificate file path: 그냥 엔터



   위같은 창이 뜨는 경우 아무것도 입력하지 않고 엔터를 눌러주시면 됩니다.

3. 서버 등록된것 확인하기



   파란 아이콘의 좌측 `>` 아이콘을 누르면 위 사진처럼 펼쳐지며 보입니다!

   결국 위처럼 `127.0.0.1` 서버에 존재하는 데이터베이스 목록이 보여져야 합니다.

03. Sequelize 설정

▼ 1) 모듈 설치

터미널을 열어 프로젝트가 존재하는 위치로 이동하고 아래처럼 모듈을 설치해주세요!

```
npm i sequelize mysql2 -S
npm i sequelize-cli -D
```

- 첫번째 줄은 Sequelize를 설치해서 우리가 Node.js 코드에서 sequelize를 사용할 수 있게 해요!
- 두번째 줄은 우리가 Sequelize를 조금 더 쉽게 사용하도록 도와주는 도구를 설치하는 명령어예요!

▼ 2) Sequelize 사용 준비 (with sequelize-cli)

공식 문서를 참고해서 터미널에서 아래와 같은 명령어를 입력해줍니다!

```
npx sequelize init
```

▼ 3) 어떤것이 바뀌었는지 확인

2번이 문제 없이 잘 되었나요? 아래의 항목이 모두 똑같이 되어있다면 성공이예요!

1. models 폴더 안에 index.js가 생성됨

   👉 이 파일은 우리가 구현할 sequelize 모델을 편리하게 사용할 수 있게 해주는 파일이니 절대 지우면 안돼요!

2. config 폴더 안에 config.json 파일이 생성됨
   이 파일을 열어보면 데이터베이스에 연결하기 위한 설정 데이터가 JSON 형식으로 들어가있어요.
3. migrations 폴더가 생성됨
4. seeder 폴더가 생성됨

3~4번에 해당하는 폴더는 모두 빈 폴더로 생성되어야 해요!

📌 **주의!!** 여기서 생성된 폴더나 파일들은 절대로 임의로 옮기지 마세요!
sequelize-cli는 정해진 경로에 있는 파일을 사용하고 저장하기 때문에 임의로 옮기면 오동작 할 가능성이 높아요!

▼ 4) config/config.json 파일 수정하기

이 파일은 데이터베이스에 연결하기 위한 설정이 들어있는 파일이라고 위에서 설명했었죠?
이 파일을 열어서 우리 컴퓨터에 띄워둔 데이터베이스 정보를 입력해봅시다!

development, test, production 3개의 키가 존재할텐데 우린 지금은 development 안에 있는 내용만 수정해볼게요!

▼ **[코드스니펫] config/config.json 수정**

```
{
  "development": {
    "username": "root",
    "password": "1234",
    "database": "database_development",
    "host": "127.0.0.1",
    "dialect": "mysql"
  },
  "test": {
    "username": "root",
    "password": null,
    "database": "database_test",
    "host": "127.0.0.1",
    "dialect": "mysql"
  },
  "production": {
    "username": "root",
    "password": null,
    "database": "database_production",
    "host": "127.0.0.1",
    "dialect": "mysql"
```

```
  }
}
```

어떤게 바뀌었는지 눈치 챘나요?
password가 null 이던것을 "1234"로 바꿔줬어요! 이렇게 해야 우리가 컴퓨터에 띄워둔 데이터베이스 서버에 sequelize를 이용해 접속할 수 있습니다 😉

▼ 5) 데이터베이스 생성하기

아래 명령어를 실행하면 설정파일에 있는 데이터베이스 서버에 데이터베이스가 자동으로 생성될거예요!

```
npx sequelize db:create
```

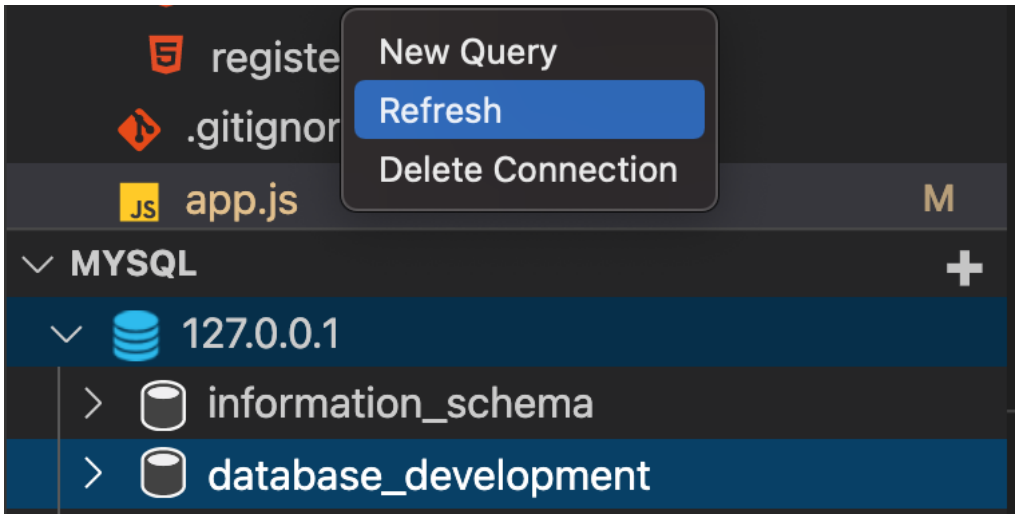
▼ ✅ 데이터베이스? 이걸 왜 생성해야 하나요?

> 👉 지금부터 언급하는 데이터베이스 는 일반적으로 "DB 서버" 또는 "데이터베이스"라고 부르는 MySQL 서버와는 다른 MySQL이 정의해둔 가상의 개념입니다.
> 하나의 서버에 여러개의 가상의 데이터베이스를, 데이터베이스 안에 여러개의 테이블을 생성해서 관리하는 방식이죠 😉
>
> 정리하자면 1개의 MySQL 서버 안에 n개의 데이터베이스가 존재할 수 있고,
> 1개의 데이터베이스 안에 n개의 테이블이 존재할 수 있습니다.

▼ 잘 생성됐는지 확인하기!

아까 VS Code에 설치한 확장 플러그인을 사용해볼게요!

탐색기 → MYSQL 탭에서 파란 아이콘이 있는 "127.0.0.1" 위에서 오른쪽 마우스를 누르면 아래처럼 "database_development"라는 까만 원통형 아이콘이 생겨납니다.



이 까만 원통 하나하나가 여러분 컴퓨터에 있는 데이터베이스예요!

04. 로그인/회원가입 기능을 Sequelize로 구현하기(1)

> 👉 Node.js 기초반에서는 mongoose를 이용해서 로그인 및 회원가입 기능을 구현한 적이 있어요!
> 그러나 우리가 구현하는 기능들은 데이터베이스의 유형에 구애받지 않는다는것을 직접 체험해보기 위해 사용자 모델을 sequelize로 다시 구현하고 기능이 잘 동작하는지 확인해볼거예요 😎

▼ 1) models/user.js 파일 제거하기

지금 존재하는 models 폴더 안의 user.js 파일은 mongoose로 구현한 모델이예요.
이 파일을 지우고 sequelize로 다시 만들 준비를 할게요!

▼ 2) 사용자 모델 생성하기

▼ **[코드스니펫] User 모델 생성 명령어**

```
npx sequelize model:generate --name User --attributes email:string,nickname:string,password:string
```

터미널에 위 명령어를 그대로 입력해보세요!
models 폴더에 user.js가 생기고, migrations 폴더에 `숫자-create-user.js` 같은 이름으로 파일이 하나 생겼을거예요. (숫자는 생성한 시간마다 달라요!)

- models/user.js 의 역할:
  - 데이터베이스에 읽고 쓰고, 변경하는 데이터를 모델이라는 형태로 구현해서 정해둔 형식대로 다루게 해주는 역할을 합니다.
  - 이 파일을 직접 만들어보고 싶다면 공식 문서의 모델 페이지를 참고해보세요!
- migrations 안에 있는 파일의 역할:
  - 데이터베이스에 테이블을 생성하고 제거할 때 여기에 있는 파일을 사용합니다.

▼ 3) 모델과 마이그레이션 파일 수정하기

우리는 명령어 한줄로 간단하게 사용자 모델을 만들었지만, 두 파일 모두 한가지씩 수정할거예요!
Sequelize는 데이터의 고유 ID를 기본적으로 "id" 로 저장하도록 하는데, 저희가 준비한 프론트엔드 코드는 이 고유 ID의 이름이 "id"가 아닌 "userId"여야 제대로 동작하도록 짜여 있답니다!

그렇기 때문에 id 대신 userId라는 이름으로 ID를 다루도록 수정해볼게요!

▼ **[코드스니펫] models/user.js 파일 수정**

```
'use strict';
const {
  Model
} = require('sequelize');
module.exports = (sequelize, DataTypes) => {
  class User extends Model {
    /**
     * Helper method for defining associations.
     * This method is not a part of Sequelize lifecycle.
     * The `models/index` file will call this method automatically.
     */
    static associate(models) {
      // define association here
    }
  };
  User.init({
    userId: {
      primaryKey: true,
      type: DataTypes.INTEGER,
    },
    email: DataTypes.STRING,
    nickname: DataTypes.STRING,
    password: DataTypes.STRING
  }, {
    sequelize,
    modelName: 'User',
  });
  return User;
};
```

▼ **[코드스니펫] migrations/숫자-create-user.js 파일 수정**

```
'use strict';
module.exports = {
  up: async (queryInterface, Sequelize) => {
    await queryInterface.createTable('Users', {
      userId: {
        allowNull: false,
        autoIncrement: true,
        primaryKey: true,
        type: Sequelize.INTEGER
```

```
      },
      email: {
        type: Sequelize.STRING
      },
      nickname: {
        type: Sequelize.STRING
      },
      password: {
        type: Sequelize.STRING
      },
      createdAt: {
        allowNull: false,
        type: Sequelize.DATE
      },
      updatedAt: {
        allowNull: false,
        type: Sequelize.DATE
      }
    });
  },
  down: async (queryInterface, Sequelize) => {
    await queryInterface.dropTable('Users');
  }
};
```

📌 모델에서는 id가 없었는데 migration 파일에는 존재했죠?
이 차이는 Sequelize가 "id"라는 키를 모델 데이터의 ID로 사용하도록 기본 설정 해두었기 때문에 모델에 따로 표현하지 않아도 잘 동작했던것입니다!

여기서 포인트는 ID는 있어도 되고 없어도 되는것이 아니라, 모델에 id가 없다면 Sequelize가 기본적으로 "id"가 테이블에 존재한다고 가정하고 id가 있는것처럼 동작합니다.
그렇기 때문에 테이블에 id가 존재하지 않으면 안된다는 뜻과도 같고, 테이블을 생성하는 코드인 이 파일에서 id가 존재했던것입니다 😳

▼ 4) 테이블 생성하기

```
npx sequelize db:migrate
```

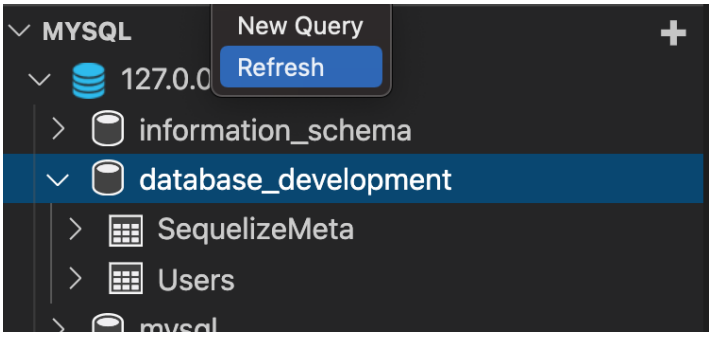
위 명령어를 터미널에 입력하고 나서 이후 아래와 같은 메세지가 떠야 합니다!

```
Sequelize CLI [Node: 15.5.1, CLI: 6.2.0, ORM: 6.4.0]

Loaded configuration file "config/config.json".
Using environment "development".
== 20210214073234-create-user: migrating =======
== 20210214073234-create-user: migrated (0.031s)
```

▼ 5) 테이블이 잘 생성됐는지 확인하기

탐색기 → MYSQL 탭에서 "database_development" 테이블 위에서 오른쪽 마우스를 누르면 아래처럼 "Refresh" 메뉴가 뜨고, 이것을 누르면 아래 사진처럼 SequelizeMeta, Users 라는 테이블이 보여야 합니다!



05. 로그인/회원가입 기능을 Sequelize로 구현하기(2)

▼ 1) 사용자 모델을 이용해 회원가입 기능 수정하기

▼ 구현하려면 어떻게 해야 할까요?

이번에 구현한 사용자 모델을 이용해서 MongoDB가 아닌 MySQL에 사용자 데이터를 저장하도록 바꿔야겠죠?
더불어, 이메일과 닉네임 중복 확인하는 코드도 제대로 동작하는지 확인해보세요!

사용자 모델을 불러오는것은 아래 사용자 모델 불러오기를 참고해주세요 😉

▼ 사용자 모델 불러오기 예시

```
const { User } = require("./models");
```

`models/index.js` 파일이 데이터베이스 연결 설정과 함께 Sequelize 모델 파일을 불러와서 다시 내보내주기 때문에 위의 코드처럼 사용자 모델을 사용해야 합니다.

▼ 회원가입 API 수정 예시

아래 코드에 있는 `...` 는 이미 이전에 구현했던 코드를 생략했다고 표현하기 위함이니 참고하세요!

```
...
const { Op } = require("sequelize");
const { User } = require("./models");

...

router.post("/users", async (req, res) => {
  const { email, nickname, password, confirmPassword } = req.body;

  if (password !== confirmPassword) {
    res.status(400).send({
      errorMessage: "패스워드가 패스워드 확인란과 다릅니다.",
    });
    return;
  }

  // email or nickname이 동일한게 이미 있는지 확인하기 위해 가져온다.
  const existsUsers = await User.findAll({
    where: {
      [Op.or]: [{ email }, { nickname }],
    },
  });
  if (existsUsers.length) {
    res.status(400).send({
      errorMessage: "이메일 또는 닉네임이 이미 사용중입니다.",
    });
    return;
  }

  await User.create({ email, nickname, password });
  res.status(201).send({});
});
```

▼ 2) 사용자 모델을 이용해 로그인 기능 수정하기

▼ 구현하려면 어떻게 해야 할까요?

여기서는 사용자 정보를 데이터베이스에서 가져오는 부분만 수정하면 될거예요!

나머지 코드는 데이터베이스에 의존적인 코드가 아니기 때문이죠 🙂

▼ 로그인 API 수정 예시

```
router.post("/auth", async (req, res) => {
  const { email, password } = req.body;

  const user = await User.findOne({
    where: {
      email,
    },
  });

  // NOTE: 인증 메세지는 자세히 설명하지 않는것을 원칙으로 한다: https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Shee
  if (!user || password !== user.password) {
    res.status(400).send({
      errorMessage: "이메일 또는 패스워드가 틀렸습니다.",
```

```
    });
    return;
  }

  res.send({
    token: jwt.sign({ userId: user.userId }, "customized-secret-key"),
  });
});
```

▼ 3) 사용자 모델을 이용해 로그인 확인 미들웨어 수정하기

▼ 구현하려면 어떻게 해야 할까요?

userId를 가지고 데이터베이스에서 사용자 정보를 불러오도록 해야겠죠?

공식 문서를 참고해보니 findByPk를 사용하면 적당해보이네요!

▼ 로그인 확인 미들웨어 수정 예시

```
const jwt = require("jsonwebtoken");
const { User } = require("../models");

module.exports = (req, res, next) => {
  const { authorization } = req.headers;
  const [authType, authToken] = (authorization || "").split(" ");

  if (!authToken || authType !== "Bearer") {
    res.status(401).send({
      errorMessage: "로그인 후 이용 가능한 기능입니다.",
    });
    return;
  }

  try {
    const { userId } = jwt.verify(authToken, "customized-secret-key");
    User.findByPk(userId).then((user) => {
      res.locals.user = user;
      next();
    });
  } catch (err) {
    res.status(401).send({
      errorMessage: "로그인 후 이용 가능한 기능입니다.",
    });
  }
};
```

▼ 4) 정리하자면...

어떤가요 여러분? 데이터베이스가 바뀌었다고 무언가 크게 달라지는게 있었나요?
우리는 3주차 수업동안 쇼핑몰이 MongoDB 대신 MySQL라는 데이터베이스를 이용하도록 바꾸는 큰 변화가 있었지만 겉보기엔 큰 변화가 없었을거예요!
그리고 코드도 조금만 바꾸면 완전히 기존과 동일하게 동작하도록 구현할 수 있었죠?

이러한 경험을 비추어볼때, 아직은 여러분에게 "도구"의 결정이 생각보다 중요하지 않은 영역일 수 있습니다.
그렇게 때문에 여러분은 소프트웨어를 만들때 "어떤 데이터베이스를 이용할까?" 에 대한 고민처럼 **"도구"**에 얽매이기보다 여러분이 지금 쇼핑몰 기능을 구현하기 위해 작성했던 **"비즈니스 로직"**에 더 집중해보는것을 추천드립니다.
도구에 대한 고민은 시간이 지나 경험이 쌓이면서 자연스럽게 결정하는 능력이 생길거예요 :)

개인적으로 저는 개발자가 어떤 도구를 사용하든 잘 동작하는 소프트웨어를 만드는 개발자야 말로 여러분이 제일 처음 목표로 해야 할 수준이라고 생각하고 있어요.
이 수준에 도달하게 되면 그제서야 어떤 데이터베이스 유형이 자신의 소프트웨어와 어울리는지, 어떤 디자인 패턴으로 구현해야 하는지 신경쓸 수 있는 개발자로 진화할 수 있다고 생각합니다. 😉
그 전까지 여러분은 제대로 동작하는 비즈니스 로직을 작성하는것에 집중하시길 바랍니다.

앞으로 남은 수업 화이팅 하시길 바라면서 명언 하나 보고 가세요! ☺

> "God is in the details. (신은 디테일에 있다.)"
> — Ludwig Mies Van Der Rohe

06. 3주차 끝 & 숙제 설명

- 남아있는 API 모두 MongoDB 대신 MySQL에 데이터를 저장하고 읽어오도록 바꿔두기
  - 다 바꾸고 나면 mongoose는 필요가 없겠죠? 내 코드에서 MongoDB로 연결하는 mongoose 코드도 전부 제거해주세요!
- TCP와 UDP 차이점 조사하기
- Socket이라는 개념에 대해 조사하기
- Web socket에 대해 조사하기